Automated Stateful Protocol Verification

Andreas V. Hess
Sebastian Mödersheim
Achim D. Brucker
Anders Schlichtkrull

October 27, 2022

*DTU Compute, Technical University of Denmark, Lyngby, Denmark
{avhe, samo}@dtu.dk

† Department of Computer Science, University of Exeter, Exeter, UK
a.brucker@exeter.ac.uk

‡ Department of Computer Science, Aalborg University, Copenhagen, Denmark
andsch@cs.aau.dk
Abstract

In protocol verification we observe a wide spectrum from fully automated methods to interactive theorem proving with proof assistants like Isabelle/HOL. In this AFP entry, we present a fully-automated approach for verifying stateful security protocols, i.e., protocols with mutable state that may span several sessions. The approach supports reachability goals like secrecy and authentication. We also include a simple user-friendly transaction-based protocol specification language that is embedded into Isabelle.

**Keywords:** Fully automated verification, stateful security protocols
Contents

1 Introduction 7

2 The PSPSP Manual 9
   2.1 Introduction 9
   2.2 Installation 9
   2.3 A Brief Overview of Isabelle/PSPSP 10
   2.4 Common Pitfalls 15
   2.5 Reference Manual 17

3 Stateful Protocol Verification 21
   3.1 Protocol Transactions 21
   3.2 Term Abstraction 34
   3.3 Stateful Protocol Model 38
   3.4 Term Variants 130
   3.5 Term Implication 137
   3.6 Stateful Protocol Verification 180

4 Trac Support and Automation 295
   4.1 Useful Eisbach Methods for Automating Protocol Verification 295
   4.2 ML Yacc Library 296
   4.3 Abstract Syntax for Trac Terms 296
   4.4 Parser for Trac FP definitions 330
   4.5 Parser for the Trac Format 331
   4.6 Support for the Trac Format 332

5 Examples 375
   5.1 The Keyserver Protocol 375
   5.2 A Variant of the Keyserver Protocol 376
   5.3 The Composition of the Two Keyserver Protocols 378
   5.4 The PKCS Model, Scenario 3 380
   5.5 The PKCS Protocol, Scenario 7 382
   5.6 The PKCS Protocol, Scenario 9 385
1 Introduction

In protocol verification we observe a wide spectrum from fully automated methods to interactive theorem proving with proof assistants like Isabelle/HOL. The latter provide overwhelmingly high assurance of the correctness, which automated methods often cannot: due to their complexity, bugs in such automated verification tools are likely and thus the risk of erroneously verifying a flawed protocol is non-negligible. There are a few works that try to combine advantages from both ends of the spectrum: a high degree of automation and assurance.

Inspired by [1], we present here a first step towards achieving this for a more challenging class of protocols, namely those that work with a mutable long-term state. To our knowledge this is the first approach that achieves fully automated verification of stateful protocols in an LCF-style theorem prover. The approach also includes a simple user-friendly transaction-based protocol specification language embedded into Isabelle, and can also leverage a number of existing results such as soundness of a typed model (see, e.g., [3, 4, 6]) and compositionality (see, e.g., [3, 7]). The Isabelle formalization extends the AFP entry on stateful protocol composition and typing [8].

The rest of this document is automatically generated from the formalization in Isabelle/HOL, i.e., all content is checked by Isabelle. [chapter 2] provides a manual of our automated protocol verification tool, called PSPSP, that is provided as part of this AFP entry. Thereafter, the structure of this document follows the theory dependencies (see Figure 1.1): After introducing the formal framework for verifying stateful security protocols [chapter 3], we continue with the setup for supporting the high-level protocol specifications language for security protocols (the Trac format) and the implementation of the fully automated proof tactics [chapter 4]. Finally, we present examples [chapter 5].

Acknowledgments This work was supported by the Sapere-Aude project “Composec: Secure Composition of Distributed Systems”, grant 4184-00334B of the Danish Council for Independent Research, by the EU H2020 project no. 700321 “LIGHTest: Lightweight Infrastructure for Global Heterogeneous Trust management in support of an open Ecosystem of Trust schemes” (lightest.eu) and by the “CyberSec4Europe” European Union’s Horizon 2020 research and innovation programme under grant agreement No 830929.
Figure 1.1: The Dependency Graph of the Isabelle Theories.
2 The PSPSP Manual

2.1 Introduction

In this section, we describe the installation and use of Isabelle/PSPSP, the system implementing the approach described in our CSF submission.

Isabelle/PSPSP is built on top of the latest version of Isabelle/HOL [9]. While Isabelle is widely perceived as an interactive theorem prover for HOL (Higher-order Logic), we would mention that Isabelle can be understood as a framework that provides various extension points. In our work, we make use of this fact by extending Isabelle/HOL with:

- a formalization of the protocol-independent aspects of our approach that is based on a large formalization (the session is called \texttt{Automated\_Stateful\_Protocol\_Verification}) of security protocols in Isabelle/HOL that, among others, includes proofs for typing results and protocol compositionality. The main entry for the security analysis of concrete protocols using Isabelle/PSPSP is the theory \texttt{Automated\_Stateful\_Protocol\_Verification.PSPSP}.

- an encoder (datatype package) that translates a high-level protocol specification (called “trac”) into HOL. This datatype package provides the high-level command \texttt{trac}.

- a command (called \texttt{compute\_fixpoint}) that computes an over-approximation of all messages that a security protocol can generate.

- a command that, for a specific class of protocols, can fully-automatically prove their security (\texttt{proto-colsecurity\_proof}).

- a command that generates a list of proof obligations (sub-goals) for proving the security of the specified protocol interactively (\texttt{manual\_protocol\_security\_proof}).

- several proof methods that either can be used interactively or that are used internally by the fully automated proof setup (\texttt{protocol\_security\_proof}).

2.2 Installation

Isabelle/PSPSP extends Isabelle/HOL. Thus, the first step is to install Isabelle. Moreover, we make use of the Archive of Formal Proofs (AFP), which needs to be installed in a second step. Finally, we need to register the new Isabelle components and compile the session heaps for faster start up.

2.2.1 Installing Isabelle

Isabelle can be downloaded from the Isabelle website [http://isabelle.in.tum.de/]. Detailed installation instructions for all supported operating systems are available at [https://isabelle.in.tum.de/installation.html](https://isabelle.in.tum.de/installation.html).

2.2.2 Installing the Archive of Formal Proofs

After installing Isabelle, we now need to install the AFP (Archive of Formal Proofs). The AFP [https://www.isa-afp.org](https://www.isa-afp.org) is a large library of Isabelle formalizations. Please install the latest version, following the instructions from [https://www.isa-afp.org/using.html](https://www.isa-afp.org/using.html).
2.2.3 Compiling Session Heaps and Final Setup

We recommend to “compile” Isabelle/PSPSP (in Isabelle lingo: building the session heaps) on the command line. This can be done by executing (please take care of the full qualified path of the isabelle binary for your operating system):

```
achim@logicalhacking:$ isabelle build -b Automated_Stateful_Protocol_Verification
Building Pure ...
Finished Pure (0:00:50 elapsed time, 0:00:50 cpu time, factor 1.00)
Building HOL ...
Finished HOL (0:09:50 elapsed time, 0:31:02 cpu time, factor 3.16)
Building HOL-Library ...
Finished HOL-Library (0:04:49 elapsed time, 0:24:43 cpu time, factor 5.13)
Building Abstract-Rewriting ...
Finished Abstract-Rewriting (0:01:28 elapsed time, 0:04:00 cpu time, factor 2.71)
Building First_Order_Terms ...
Finished First_Order_Terms (0:00:47 elapsed time, 0:01:54 cpu time, factor 2.39)
Building Stateful_Protocol_Composition_and_Typing ...
Finished Stateful_Protocol_Composition_and_Typing (0:08:18 elapsed time, 0:36:38 cpu time, \factor 4.41)
Building Automated_Stateful_Protocol_Verification ...
Finished Automated_Stateful_Protocol_Verification (0:15:11 elapsed time, 0:50:57 cpu time, \factor 3.36)
0:41:46 elapsed time, 2:30:06 cpu time, factor 3.59
achim@logicalhacking:$
```

Isabelle will build all sessions that are required. Note that you might have already some of the heaps available and, hence, only a subset of the list shown above might be build on your system.

Finally, please start the (graphical) Isabelle application by clicking on the Isabelle icon (macOS) or by starting Isabelle2021-1 (this example is for Isabelle version 2021-1) on the command line (Linux and macOS):

```
achim@logicalhacking:$ ./Isabelle2021-1/Isabelle2021-1
```

and select the session Automated_Stateful_Protocol_Verification. For doing so, you need to select the “Theories”-pane on the right hand side and select the session from drop-down menu (see Figure 2.1). To persist this configuration, you need to restart Isabelle, i.e., please close Isabelle/jEdit now. On the next start, Automated_Stateful_Protocol_Verification will be the default session.

2.3 A Brief Overview of Isabelle/PSPSP

In this section, we briefly explain how to use Isabelle/PSPSP for proving the security of protocols. As Isabelle/PSPSP is build on top of Isabelle/HOL, the overall user interface and the high-level language (called Isar) are inherited from Isabelle. We refer the reader to [9] and the system manuals that are part of the Isabelle distribution. The latter are accessible within Isabelle/jEdit in the documentation pane on the left-hand side of the main window.

In the following, we will illustrate the use of our system by analysing a simple keyserver protocol (this theory is stored in the file PSPSP-Manual/KeyserverEx.thy. When loadign this theory in Isabelle/jEdit, please ensure that the session Automated_Stateful_Protocol_Verification is active (this session provides Isabelle/PSPSP).

When done, please move the text cursor to the section “Proof of Security”. There are some orange question marks at the side of some lines. These are the comments from Isabelle that indicate the timing results we ask for: when moving the cursor to the corresponding line, and selecting the Output-Tab on the bottom of the Isabelle window (ensure that there is a tick-mark on “Auto update”), you see the timing information provided by Isabelle for each step. Your Isabelle should look similar to Figure 2.2.

---

1The sessions should also be build automatically on the start of Isabelle’s graphical user interface Isabelle/jEdit. For this, it is important that you select the session Automated_Stateful_Protocol_Verification as described in the following paragraph and restart Isabelle. For us, building on the command line has easier to reproduce on different machines.
2.3 A Brief Overview of Isabelle/PSPSP

Figure 2.1: Isabelle/jEdit on its first startup. Please click on the “Theories” tab on the right hand side and select the session “Automated_Stateful_Protocol_Verification.”

Figure 2.2: Opening KeyserverEx.thy in Isabelle/jEdit.
The Isabelle IDE (called Isabelle/jEdit) is a front-end for Isabelle that supports most features known from
IDEs for programming languages. The input area (in the middle of the upper part of the window) supports, e.g.,
auto completion, syntax highlighting, and automated proof generation as well as interactive proof development.
The lower part shows the current output (response) with respect to the cursor position.

We will now briefly explain this example in more detail. First, we start with the theory header: As in
Isabelle/HOL, formalization happens within theories. A theory is a unit with a name that can import other
theories. Consider the following theory header:

```
theory KeyserverEx
imports Automated_Stateful_Protocol_Verification.PSPSP
begin
```
which opens a new theory `KeyserverEx` that is based on the top-level theory of Isabelle/PSPSP, called
`Automated_Stateful_Protocol_Verification.PSPSP`. Within this theory, we can use all definitions and tools pro-
vided by Isabelle/PSPSP. For example, Isabelle/PSPSP provides a mechanism for measuring the run-time of
certain commands. This mechanism can be turned on as follows:

```
declare [[pssp_timing]]
```

### 2.3.1 Protocol Specification

The protocol is specified using a domain-specific language that, e.g., could also be used by a security protocol
model checker. We call this language “trac” and provide a dedicated environment (command) `trac` for it:

```
trac<
Protocol: Keyserver

Enumerations:
  honest = {a,b,c}
  dishonest = {i}
  agent = honest ++ dishonest

Sets:
  ring/1 valid/1 revoked/1 deleted/1

Functions:
  Public sign/2 crypt/2 pair/2
  Private inv/1

Analysis:
  sign(X,Y) -> Y
  crypt(X,Y) ? inv(X) -> Y
  pair(X,Y) -> X,Y

Transactions:
# Out-of-band registration
outOfBand(A:honest)
  new PK
  insert PK ring(A)
  insert PK valid(A)
  send PK.

# Out-of-band registration (for dishonest users; they reveal their private keys to the intruder)
outOfBandD(A:dishonest)
  new PK
  insert PK valid(A)
  send PK
  send inv(PK).

# User update key
keyUpdateUser(A:honest,PK:value)
```
2.3 A Brief Overview of Isabelle/PSPSP

PK in ring(A)
new NPK
delete PK ring(A)
insert PK deleted(A)
insert NPK ring(A)
send sign(inv(PK),pair(A,NPK)).

# Server update key
keyUpdateServer(A:agent,PK:value,NPK:value)
receive sign(inv(PK),pair(A,NPK))
PK in valid(A)
NPKnotin valid(_)
NPKnotin revoked(_)
delete PK valid(A)
insert PK revoked(A)
insert NPK valid(A)
send inv(PK).

# Attack definition
attackDef(A:honest,PK:value)
receive inv(PK)
PK in valid(A)
attack.

The command trac automatically translates this specification into a family of formal HOL definitions. Moreover, basic properties of these definitions are also already proven automatically (i.e., without any user interaction): for this simple example, already over 350 definitions and theorems are automatically generated, respectively, formally proven. For example, the following induction rule is derived:

\[
\begin{align*}
& Keyserver_Ana_dom ?a.0; Keyserver_Ana_dom sign = ?P sign; \\
& Keyserver_Ana_dom crypt = ?P crypt; Keyserver_Ana_dom pair = ?P pair; \\
& Keyserver_Ana_dom Keyserver_fun.inv = ?P Keyserver_fun.inv; \\
& Keyserver_Ana_dom PrivFunSec = ?P PrivFunSec; \\
& \bigwedge uu_. Keyserver_Ana_dom (enum uu_) = ?P (enum uu_)] \\
& \Rightarrow ?P ?a.0.0
\end{align*}
\]

2.3.2 Protocol Model Setup

Next, we show that the defined protocol satisfies the requirement of our protocol model (technically, this is done by instantiating several Isabelle locales, resulting in over 1750 theorems “for free.”). The underlying instantiation proofs are fully automated by our tool:

protocol_model_setup spm: Keyserver

2.3.3 Fixpoint Computation

Now we compute the fixed-point:

compute_fixpoint Keyserver_protocol Keyserver_fixpoint

We can inspect the fixed-point with the following command:

thm Keyserver_fixpoint_def

Moreover, we can use Isabelle’s value-command to compute its size:

value "let (FP,_,TI) = Keyserver_fixpoint in (size FP, size TI)"

2.3.4 Proof of Security

After these steps, all definitions and auxiliary lemmas for the security proof are available. Note that the security proof will fail, if any of the previous commands did fail. A failing command is sometimes hard to spot for non Isabelle experts: the status bar next to the scroll bar on the right-hand side of the window should not have any “dark red” markers.
We can do a fully automated security proof using a new command `protocol_security_proof`:

```
protocol_security_proof ssp: Keyserver
```

This command proves the security protocol only using Isabelle's simplifier (and, hence, everything is checked by Isabelle's LCF-style kernel).

Moreover, we provide two alternative configuration, one using an approach called “normalization by evaluation” (nbe) and one using Isabelle's code generator for direct code evaluation (eval). Please see section 2.5 and Isabelle’s code generator manual \[\textbf{2}\] for details.

```
protocol_security_proof [nbe] ssp: Keyserver
```

While the stack of code that needs to be trusted for the normalization by evaluation is much smaller than for the direct code evaluation, direct code evaluation is usually much faster:

```
protocol_security_proof [unsafe] ssp: Keyserver
```

Moreover, there is the option to only generate the proof obligations (as sub-goals) for an interactive security proof:

```
manual_protocol_security_proof ssp: Keyserver
   for Keyserver_protocol Keyserver_fixpoint
   apply check_protocol_intro
   subgoal by (timeit code_simp)
   subgoal by (timeit eval)
   subgoal by (timeit code_simp)
   subgoal by (timeit normalization)
   subgoal by (timeit code_simp)
   done
```

Such an interactive proof allows us to interactively inspect intermediate proof states or to use protocol-specific proof strategies (e.g., only partially unfolding the fixed-point).

### 2.3.5 Inspecting the Generated Theorems and Definitions

We can inspect the generated proofs using the `thm`:

```
thm ssp.protocol_secure
thm ssm.constraint_model_def
thm ssm.reachable_constraints.simps

thm Keyserver_enum_consts.nchotomy
thm Keyserver_sets.nchotomy
thm Keyserver_atom.nchotomy
thm Keyserver_arity.simps
thm Keyserver_sets_arity.simps
thm Keyserver_public.simps
thm Keyserver_. Gamma.simps
thm Keyserver_ana.simps

thm Keyserver_protocol_def
thm Keyserver_transaction_intruderValueGen_def
thm Keyserver_transaction_outOfBand_def
thm Keyserver_transaction_outOfBand_def
thm Keyserver_transaction_keyUpdateUser_def
thm Keyserver_transaction_keyUpdateServer_def
thm Keyserver_transaction_attackDef_def

thm Keyserver_fixpoint_def
```

Finally, the theory needs to be closed:

```
end
```
2.4 Common Pitfalls

This section explains some common pitfalls, along with solutions, that one may encounter when writing trac specifications.

2.4.1 Not Including an Initial Value-Producing Transaction

Trac specifications that contain value-typed variables should also declare a transaction that produces fresh values. Take, for instance, a trac specification that contains only one transaction:

\[
\begin{align*}
\text{Transactions:} \\
\text{attackDef(PK:value)} \\
\text{receive PK} \\
\text{attack.}
\end{align*}
\]

This protocol is technically secure because no values are ever produced. Similarly, if we just look at the protocol with the following transaction then we find that it is also secure:

\[
\begin{align*}
\text{Transactions:} \\
\text{attackDef(PK:value)} \\
\text{attack.}
\end{align*}
\]

The reason it is secure is because of the occurs-message transformation that is being applied to each transaction \( T \) of the protocol for technical reasons: A \text{receive occurs}(PK) action is added to \( T \) for each value-typed variable PK declared in \( T \), and a \text{send occurs}(PK) is added to \( T \) for each \text{new PK} action occurring in \( T \). Since no values are actually produced in any protocol run, then no occurs-message is produced, and so the attackDef transaction cannot ever be applied. One would, however, naturally expect that such a protocol is not secure. For this reason we require that each trac specification includes a value-producing transaction if there are any value-typed variables occurring in the trac specification at all. For instance, when including such a transaction to our example we get a valid trac transaction specification:

\[
\begin{align*}
\text{Transactions:} \\
\text{valueProducer()} \\
\text{new PK} \\
\text{send PK.} \\
\text{attackDef1(PK:value)} \\
\text{attack.}
\end{align*}
\]

Another example is the following which is also a valid trac transaction specification because it does not declare any value-typed variables:

\[
\begin{align*}
\text{Transactions:} \\
\text{attackDef2()} \\
\text{attack.}
\end{align*}
\]

Both protocols have attacks, as expected. Examining the generated Isabelle definitions reveals that the \text{valueProducer} transaction produces an occurs message while the \text{attackDef1} transaction expects to receive an occurs message:

```
trac<
Protocol: ex1

Enumerations:
dummy_type = {dummy_constant}

Sets:
dummy_set/0
```
Transactions:
valueProducer()
    new PK
    send PK.

attackDef1(PK:value)
    attack.

thm  ex1_transaction_valueProducer_def
thm  ex1_transaction_attackDef1_def

2.4.2 Using Value-Typed Database-Parameters in Database-Expressions

Due to the nature of the abstraction that is at the core of our verification approach it is simply not possible to use value-typed variables in parameters to databases. Hence, a trac specification with the following transaction would be rejected:

```
f(PK:value,A:value)
    PK in db(A).
```

As an alternative one could declare A with a type—say, `agent`—that is itself declared in the `Enumerations` section of the trac specification:

```
Enumerations:
    agent = {a,b,c}

Transactions:
    f(PK:value,A:agent)
        PK in db(A).
```

2.4.3 Not Ordering the Action Sequences in Transactions Correctly

The actions of a transaction should occur in the correct order; first receive actions, then database checks, then new actions and database updates, and finally send actions.

Hence, the following is an invalid transaction:

```
invalid(PK:value)
    send f(PK)
    receive g(PK).
```

whereas the following is valid:

```
valid(PK:value)
    receive f(PK)
    send g(PK).
```

2.4.4 Declaring Ill-Formed Analysis Rules

Each analysis rule must either be of the form

```
Ana(f(X₁,...,Xₙ)) ? t'₁,...,t'ₖ -> t₁,...,tₘ
```

or of the form
Ana(f(X_1,...,X_n)) \rightarrow t_1,...,t_m

where \( f \) is a function symbol of arity \( n \), the variables \( X_i \) are all distinct, and the variables occurring in the \( t_i \) and \( t'_i \) terms are among the \( X_i \) variables.

### 2.4.5 Declaring Public Constants of Type Value

It is not possible to directly refer to constants of type value. A possible workaround is to instead add a transaction that generates fresh values and releases them to the intruder (thereby making them “public”):

```isabelle
freshPublicValues():
  new K
  send K.
```

It is usually beneficial to ensure that all fresh values are inserted into a database before being transmitted over the network. In this example one could use a database that is not used anywhere else:

```isabelle
freshPublicValues():
  new K
  insert K publicvalues
  send K.
```

Under the set-based abstraction this prevents accidentally identifying values produced from this transaction with values produced elsewhere in the protocol, since they are now identified with their own unique abstract value \{publicvalues\} instead of the more common “empty” abstract value \{\}.

### 2.4.6 Forgetting to Terminate Transactions With a period

Transactions must end with a period. Forgetting this period may result in a confusing error message from the parser. For instance, suppose that we have the following `Transaction` section where we forgot to terminate the `valueProducer` transaction:

```isabelle
valueProducer()
  new PK
  send PK
  attackDef(PK:value)
  attack.
```

This could result in an error message like the following:

```
Error, line .... 14.13, syntax error: deleting COLON LOWER_STRING_LITERAL
```

### 2.5 Reference Manual

In this section, we briefly introduce the syntax of the most important commands and methods of Isabelle/PSPSP. We follow, in our presentation, the style of the Isabelle/Isar manual \([10]\). For details about the standard Isabelle commands and methods, we refer to the reader to this manual \([10]\).

#### 2.5.1 Top-Level Isabelle Commands

trac
This command takes a protocol in the trac language as argument. The command translates this high-level protocol specification into a family of HOL definitions and also proves already a number of properties basic properties over these definitions. The generated definitions are all prefixed with the name of the protocol, as given as part of the trac specification.

**protocol\_model\_setup**

This command takes one argument, the name of the protocol (as given in the trac specification). In general, this command proves a large number of properties over the protocol specification that are later used by our security proof. In particular, the command does internally instantiation proofs showing, e.g., that the protocol specifications satisfies the requirements of the typing results of \[5\].

**compute\_fixpoint**

This command computes the fixed-point of the protocol. It takes two arguments, first the protocol name (as given in the trac specification) and, second, the name that should be used for constant to which the generated fixed point is bound. The algorithm for computing the fixed-point has been specified in HOL. Internally, Isabelle’s code generator is used for deriving an SML implementation that is actually used. Note that our approach does not rely on the correctness of this algorithm neither on the correctness of the code generator.

**compute\_SMP**

This command computes the SMP set of the protocol. It takes two arguments, first the protocol name (as given in the trac specification) and, second, the name that should be used for constant to which the generated SMP set is bound.

**protocol\_security\_proof**

This command executes the formal security proof for the given security protocol. Its internal behavior can be configured using one of the following three options:

- **[safe]** (default): use Isabelle’s simplifier to prove the goal by symbolic evaluation. In this mode, all proof steps are checked by Isabelle’s LCF-style kernel.
- **[nbe]**: use normalization by evaluation, a partial symbolic evaluation which permits also normalization of functions and uninterpreted symbols. This setup uses the well-tested default configuration of Isabelle’s code generator for HOL. While the stack of code to be trusted is considerable, we consider this still a highly trustworthy setup, as it cannot be influenced by end-user configurations of the code generator.
• [unsafe]: use Isabelle’s code-generator for evaluating the proof goal on the SML-level. While this is, by far, the fastest setup, it depends on the full-blown code-generator setup. As we do not modify the code-generator setup in our formalisation, we consider the setup to be nearly as trustworthy as the normalization by evaluation setup. Still, end-user configurations of the code generator could, inadvertently, introduce inconsistencies.

For a detailed discussion of these three modes and the different software stacks that need to be trusted, we refer the reader to the tutorial describing the code generator [2] Section 5.1.

```
manual_protocol_security_proof

(manual_protocol_security_proof) (ssp: protocol-name)
```

This command allows to interactively prove the security of a protocol. As the fully automated version, it takes the protocol name as argument but it does not execute a proof. Instead, it generates a proof state with the necessary proof obligations. It is the responsibility of the user to discharge these proof obligations. Application of this command results in a regular Isabelle proof state and, hence, all proof methods of Isabelle can be used.

### 2.5.2 Proof Methods

In addition to the Isar commands discussed in the previous section, Isabelle/PSPSP also provides a number of proof methods such as `check_protocol_intro` or `coverage_check_unfold`. These domain specific proof methods are used internally by, e.g., the command `manual_protocol_security_proof` and can also be used in interactive mode.
3 Stateful Protocol Verification

3.1 Protocol Transactions

theory Transactions
imports
  Stateful_Protocol_Composition_and_Typing.Typed_Model
  Stateful_Protocol_Composition_and_Typing.Labeled_Stateful_Strands
begin

3.1.1Definitions

datatype 'b prot_atom =
  is_Atom: Atom 'b
  | Value
  | SetType
  | AttackType
  | Bottom
  | OccursSecType
  | AbsValue

datatype ('a,'b,'c,'d) prot_fun =
  Fu (the_Fu: 'a)
  | Set (the_Set: 'c)
  | Val (the_Val: "nat")
  | Abs (the_Abs: "'c set")
  | Attack (the_Attack_label: "'d strand_label")
  | Pair
  | PubConst (the_PubConst_type: "'b prot_atom") nat
  | OccursFact
  | OccursSec

definition "is_Fun_Set t ≡ is_Fun t ∧ args t = [] ∧ is_Set (the_Fun t)"
definition "is_Fun_Attack t ≡ is_Fun t ∧ args t = [] ∧ is_Attack (the_Fun t)"
definition "is_PubConstValue f ≡ is_PubConst f ∧ the_PubConst_type f = Value"

abbreviation occurs where
  "occurs t ≡ Fun OccursFact [Fun OccursSec [], t]"

type_synonym ('a,'b,'c,'d) prot_term_type = "((a,'b,'c,'d) prot_fun,'b prot_atom) term_type"
type_synonym ('a,'b,'c,'d) prot_var = "('a,'b,'c,'d) prot_term_type × nat"
type_synonym ('a,'b,'c,'d) prot_term = "((a,'b,'c,'d) prot_fun,(a,'b,'c,'d) prot_var) term"
type_synonym ('a,'b,'c,'d) prot_terms = "((a,'b,'c,'d) prot_term set"
type_synonym ('a,'b,'c,'d) prot_subst = "((a,'b,'c,'d) prot_fun, (a,'b,'c,'d) prot_var) subst"
type_synonym ('a,'b,'c,'d) prot_strand_step = "((a,'b,'c,'d) prot_fun, (a,'b,'c,'d) prot_var, 'd) labeled_stateful_strand_step"
type_synonym ('a,'b,'c,'d) prot_strand = "((a,'b,'c,'d) prot_strand_step list"
type_synonym ('a,'b,'c,'d) prot_constr = "((a,'b,'c,'d) prot_strand_step list"

datatype ('a,'b,'c,'d) prot_transaction =
Transaction
(transaction_decl: "unit ⇒ (('a,'b,'c,'d) prot_var × 'a set) list")
(transaction_fresh: "('a,'b,'c,'d) prot_var list")
(transaction_receive: "('a,'b,'c,'d) prot_strand")
(transaction_checks: "('a,'b,'c,'d) prot_strand")
(transaction_updates: "('a,'b,'c,'d) prot_strand")
(transaction_send: "('a,'b,'c,'d) prot_strand")

definition transaction_strand where
"transaction_strand T ≡
  transaction_receive T@transaction_checks T@
  transaction_updates T@transaction_send T"

fun transaction_proj where
"transaction_proj l (Transaction A B C D E F) = (let f = proj l
  in Transaction A B (f C) (f D) (f E) (f F))"

fun transaction_star_proj where
"transaction_star_proj (Transaction A B C D E F) = (let f = filter has_LabelS
  in Transaction A B (f C) (f D) (f E) (f F))"

abbreviation fv_transaction where
"fv_transaction T ≡ fv lsst (transaction_strand T)"

abbreviation bvars_transaction where
"bvars_transaction T ≡ bvar lsst (transaction_strand T)"

abbreviation vars_transaction where
"vars_transaction T ≡ vars lsst (transaction_strand T)"

abbreviation trms_transaction where
"trms_transaction T ≡ trms lsst (transaction_strand T)"

abbreviation setops_transaction where
"setops_transaction T ≡ setops lsst (unlabel (transaction_strand T))"

definition wellformed_transaction where
"wellformed_transaction T ≡
  list_all is_Receive (unlabel (transaction_receive T)) ∧
  list_all is_Check_or_Assignment (unlabel (transaction_checks T)) ∧
  list_all is_Update (unlabel (transaction_updates T)) ∧
  list_all is_Send (unlabel (transaction_send T)) ∧
  distinct (map fst (transaction_decl T ())) ∧
  distinct (transaction_fresh T) ∧
  set (transaction_fresh T) ∩ fst · set (transaction_decl T ()) = {} ∧
  set (transaction_fresh T) ∩ fv lsst (transaction_receive T) = {} ∧
  set (transaction_fresh T) ∩ fv lsst (transaction_checks T) = {} ∧
  set (transaction_fresh T) ∩ bvars_transaction T = {} ∧
  fv_transaction T ∩ bvars_transaction T = {} ∧
  wf lsst (fst · set (transaction_decl T ())) ∪ set (transaction_fresh T)
  (unlabel (dual lsst (transaction_strand T))))"

type_synonym ('a,'b,'c,'d) prot = "('a,'b,'c,'d) prot_transaction list"

abbreviation Var_Value_term ("⟨_: value⟩_v") where
"⟨n: value⟩_v ≡ Var (Var Value, n)::('a,'b,'c,'d) prot_term"

abbreviation Var_SetType_term ("⟨_: SetType⟩_v") where
"⟨n: SetType⟩_v ≡ Var (Var SetType, n)::('a,'b,'c,'d) prot_term"

abbreviation Var_AttackType_term ("⟨_: AttackType⟩_v") where...
3.1 Protocol Transactions

\(\langle n: \text{AttackType}\rangle v \equiv \text{Var} (\text{Var AttackType}, n)::('a,'b,'c,'d) \text{ prot_term}''\)

abbreviation \(\text{Var}_{\text{Atom}}\text{term} (\langle \_\_\rangle v)\) where
\(\langle a\rangle v \equiv \text{Var} (\text{Var} (\text{Atom} a), n)::('a,'b,'c,'d) \text{ prot_term_type}''\)

abbreviation \(\text{Var}_{\text{CompFu}}\text{term} (\langle \_\_\rangle v)\) where
\(\langle f T\rangle v \equiv \text{Fun} (\text{Fu} f) T::('a,'b,'c,'d) \text{ prot_term}''\)

abbreviation \(\text{TAtom}_{\text{Atom}}\text{term} (\langle \_\rangle v)\) where
\(\langle a\rangle v \equiv \text{Var} (\text{Atom} a)::('a,'b,'c,'d) \text{ prot_term}''\)

abbreviation \(\text{TComp}_{\text{Fu}}\text{term} (\langle \_\_\_\rangle v)\) where
\(\langle f T\rangle v \equiv \text{Fun} (\text{Fu} f) T::('a,'b,'c,'d) \text{ prot_term}''\)

abbreviation \(\text{TAtom}_{\text{Atom}}\text{term} (\langle \_\rangle v)\) where
\(\langle a\rangle v \equiv \text{Var} (\text{Atom} a)::('a,'b,'c,'d) \text{ prot_term_type}''\)

abbreviation \(\text{TComp}_{\text{Fu}}\text{term} (\langle \_\_\_\rangle v)\) where
\(\langle f T\rangle v \equiv \text{Fun} (\text{Fu} f) T::('a,'b,'c,'d) \text{ prot_term_type}''\)

abbreviation \(\text{Fun}_{\text{Abs}}\text{const}\text{term} (\langle \_\_\rangle v)\) where
\(\langle a\rangle v \equiv \text{Fun} (\text{Abs} a) []::('a,'b,'c,'d) \text{ prot_term}''\)

abbreviation \(\text{Fun}_{\text{Attack}}\text{const}\text{term} (\langle \_\rangle v)\) where
\(\langle a\rangle v \equiv \text{Fun} (\text{Attack} a) []::('a,'b,'c,'d) \text{ prot_term}''\)

3.1.2 Lemmata

lemma \(\text{prot}_{\text{atom}}\text{UNIV}:\)
\(\langle \text{UNIV}::'b \text{ prot}_{\text{atom}} \text{ set} \rangle = \text{range} \text{Atom} \cup \{\text{Value}, \text{SetType}, \text{AttackType}, \text{Bottom}, \text{OccursSecType}, \text{AbsValue}\}''\)

proof -
  have \(\langle a\rangle \in \text{range} \text{Atom} \lor a = \text{Value} \lor a = \text{SetType} \lor a = \text{AttackType} \lor a = \text{Bottom} \lor a = \text{OccursSecType} \lor a = \text{AbsValue}''\)
  for \(a::'b \text{ prot}_{\text{atom}}''\)
  by (cases \(a\)) auto
  thus \(?\text{thesis}\) by auto
qed

instance \(\text{prot}_{\text{atom}}::\text{(finite)} \text{ finite}\)
by intro_classes (simp add: \(\text{prot}_{\text{atom}}\text{UNIV}''\)

instantiation \(\text{prot}_{\text{atom}}::\text{(enum)} \text{ enum}\)
begin
  definition \(\text{enum}_{\text{prot}_{\text{atom}}} = \text{map} \text{Atom} \text{enum}_{\text{class}}\text{.enum}[@]\langle \text{Value}, \text{SetType}, \text{AttackType}, \text{Bottom}, \text{OccursSecType}, \text{AbsValue}''\)
  definition \(\text{enum}_{\text{all}}\text{prot}_{\text{atom}} P = \text{list}_\text{all} P \text{ (map} \text{Atom} \text{enum}_{\text{class}}\text{.enum}[@]\langle \text{Value}, \text{SetType}, \text{AttackType}, \text{Bottom}, \text{OccursSecType}, \text{AbsValue}''\)
  definition \(\text{enum}_{\text{ex}}\text{prot}_{\text{atom}} P = \text{list}_\text{ex} P \text{ (map} \text{Atom} \text{enum}_{\text{class}}\text{.enum}[@]\langle \text{Value}, \text{SetType}, \text{AttackType}, \text{Bottom}, \text{OccursSecType}, \text{AbsValue}''\)

23
3 Stateful Protocol Verification

instance

proof (intro_classes)

have *: "set (map Atom (enum_class.enum::'a list)) = range Atom"
  using UNIV_enum enum_distinct by auto

show "(UNIV::'a prot_atom set) = set enum_class.enum"
  using *(1) by (simp add: prot_atom_UNIV enum_prot_atom_def)

have "set (map Atom enum_class.enum) ∩ set [Value, SetType, AttackType, Bottom, OccursSecType, AbsValue] = {}" by auto

moreover have "inj_on Atom (set (enum_class.enum::'a list))" unfolding inj_on_def by auto

hence "distinct (map Atom (enum_class.enum::'a list))" by (metis *(2) distinct_map)

ultimately show "distinct (enum_class.enum::'a prot_atom list)" by (simp add: enum_prot_atom_def)

qed

end

lemma wellformed_transaction_cases:
  assumes "wellformed_transaction T"
  shows "?A = ⇒ ?A'"
    (is "$A \Longrightarrow \text{?A}'$"
    "\(1, x\) ∈ set (transaction_receive T) \Longrightarrow \exists t. x = receive(t)" (is "$?A \Longrightarrow ?A'"))
  "(1, x) ∈ set (transaction_checks T) \Longrightarrow \\(\exists ac t s. x = (ac: t = s)\) \vee \\(\exists ac t s. x = (ac: t \in s)\) \vee \\(\exists F G. x = \forall X(\neg:\ F \vee\ : G)\)"
    (is "$?B \Longrightarrow ?B'"))
  "(1, x) ∈ set (transaction_updates T) \Longrightarrow \\(\exists t s. x = insert(t, s)\) \vee \\(\exists t s. x = delete(t, s)\)" (is "$?C \Longrightarrow ?C'"))
  "(1, x) ∈ set (transaction_send T) \Longrightarrow \exists t. x = send(t)" (is "$?D \Longrightarrow ?D'"))

proof -

have a: "list_all is_Receive (unlabel (transaction_receive T))"
    "list_all is_Check_or_Assignment (unlabel (transaction_checks T))"
    "list_all is_Update (unlabel (transaction_updates T))"
    "list_all is_Send (unlabel (transaction_send T))"

using assms unfolding wellformed_transaction_def by metis+

note b = Ball_set unlabel_in
note c = stateful_strand_step.collapse

show "$?A \Longrightarrow ?A'" by (metis (mono_tags, lifting) a(1) b c(2))
show "$?B \Longrightarrow ?B'" by (metis (no_types, lifting) a(2) b c(3, 6, 7))
show "$?C \Longrightarrow ?C'" by (metis (mono_tags, lifting) a(3) b c(4, 5))
show "$?D \Longrightarrow ?D'" by (metis (mono_tags, lifting) a(4) b c(1))

qed
3.1 Protocol Transactions

lemma wellformed_transaction_unlabel_cases: assumes "wellformed_transaction T" shows
  "x ∈ set (unlabel (transaction_receive T)) ⇒ ∃ t. x = receive(t)" (is "'?A ⇒ '?A'")
  "x ∈ set (unlabel (transaction_updates T)) ⇒ ∃ s. x = updates(s)" (is "'?B ⇒ '?B'")
  "x ∈ set (unlabel (transaction_checks T)) ⇒ ∃ v. x = checks(v)" (is "'?C ⇒ '?C'")
  "x ∈ set (unlabel (transaction_send T)) ⇒ ∃ t. x = send(t)" (is "'?D ⇒ '?D'")
proof -
  have a:
    "list_all is_Receive (unlabel (transaction_receive T))"
    "list_all is_Update (unlabel (transaction_updates T))"
    "list_all is_Check_or_Assignment (unlabel (transaction_checks T))"
    "list_all is_Receive (unlabel (transaction_send T))"
    unfolding wellformed_transaction_def by metis+
  note b = Ball_set
  note c = stateful_strand_step-collapse
  show "'?A ⇒ '?A'" by (metis (mono_tags, lifting) a(1) b c(2))
  show "'?B ⇒ '?B'" by (metis (no_types, lifting) a(2) b c(3,6,7))
  show "'?C ⇒ '?C'" by (metis (mono_tags, lifting) a(3) b c(4,5))
  show "'?D ⇒ '?D'" by (metis (mono_tags, lifting) a(4) b c(1))
qed

lemma transaction_strand_subsets[simp]:
  "set (transaction_receive T) ⊆ set (transaction_strand T)"
  "set (transaction_checks T) ⊆ set (transaction_strand T)"
  "set (transaction_updates T) ⊆ set (transaction_strand T)"
  "set (transaction_send T) ⊆ set (transaction_strand T)"
  "set (unlabel (transaction_receive T)) ⊆ set (unlabel (transaction_strand T))"
  "set (unlabel (transaction_updates T)) ⊆ set (unlabel (transaction_strand T))"
  "set (unlabel (transaction_checks T)) ⊆ set (unlabel (transaction_strand T))"
  "set (unlabel (transaction_send T)) ⊆ set (unlabel (transaction_strand T))"
unfolding transaction_strand_def unlabel_def by force+

lemma transaction_strand_subst_subsets[simp]:
  "set (transaction_receive T v) ⊆ set (transaction_strand T v)"
  "set (transaction_checks T v) ⊆ set (transaction_strand T v)"
  "set (transaction_updates T v) ⊆ set (transaction_strand T v)"
  "set (transaction_send T v) ⊆ set (transaction_strand T v)"
  "set (unlabel (transaction_receive T v)) ⊆ set (unlabel (transaction_strand T v))"
  "set (unlabel (transaction_updates T v)) ⊆ set (unlabel (transaction_strand T v))"
  "set (unlabel (transaction_checks T v)) ⊆ set (unlabel (transaction_strand T v))"
  "set (unlabel (transaction_send T v)) ⊆ set (unlabel (transaction_strand T v))"
unfolding transaction_strand_def unlabel_def subst_apply_labeled_stateful_strand_def by force+

lemma transaction_dual_subst_unfold:
  "dual_v ((transaction_strand T v) =
  dual_v (transaction_receive T v)∪
  dual_v (transaction_checks T v)∪
  dual_v (transaction_updates T v)∪
  dual_v (transaction_send T v)"
by (simp add: transaction_strand_def dual_v_append subst_is_append)

lemma transaction_dual_unlabel_unfold:
  "unlabel (dual_v (transaction_strand T v) =
  unlabel (dual_v (transaction_receive T v))∪
  unlabel (dual_v (transaction_checks T v))∪
  unlabel (dual_v (transaction_updates T v))∪
  unlabel (dual_v (transaction_send T v))"
unlabel (duallsst (transaction_send T · lsst ϑ))"
by (simp add: transaction_dual_subset_unfold unlabel_append)

lemma trms_transaction_unfold:
"trms_transaction T =
trms_{lsst} (transaction_receive T) ∪ trms_{lsst} (transaction_checks T) ∪
trms_{lsst} (transaction_updates T) ∪ trms_{lsst} (transaction_send T)"
by (metis trms_{lsst}_append unlabel_append append_assoc transaction_strand_def)

lemma trms_transaction_subst_unfold:
"trms_{lsst} (transaction_strand T · lsst ϑ) =
trms_{lsst} (transaction_receive T · lsst ϑ) ∪ trms_{lsst} (transaction_checks T · lsst ϑ) ∪
trms_{lsst} (transaction_updates T · lsst ϑ) ∪ trms_{lsst} (transaction_send T · lsst ϑ)"
by (metis trms_{lsst}_append unlabel_append append_assoc transaction_strand_def subst_lsst_append)

lemma vars_transaction_unfold:
"vars_transaction T =
vars_{lsst} (transaction_receive T) ∪ vars_{lsst} (transaction_checks T) ∪
vars_{lsst} (transaction_updates T) ∪ vars_{lsst} (transaction_send T)"
by (metis vars_{lsst}_append unlabel_append append_assoc transaction_strand_def)

lemma vars_transaction_subst_unfold:
"vars_{lsst} (transaction_strand T · lsst ϑ) =
vars_{lsst} (transaction_receive T · lsst ϑ) ∪ vars_{lsst} (transaction_checks T · lsst ϑ) ∪
vars_{lsst} (transaction_updates T · lsst ϑ) ∪ vars_{lsst} (transaction_send T · lsst ϑ)"
by (metis vars_{lsst}_append unlabel_append append_assoc transaction_strand_def subst_lsst_append)

lemma fv_transaction_unfold:
"fv_transaction T =
fv_{lsst} (transaction_receive T) ∪ fv_{lsst} (transaction_checks T) ∪
fv_{lsst} (transaction_updates T) ∪ fv_{lsst} (transaction_send T)"
by (metis fv_{lsst}_append unlabel_append append_assoc transaction_strand_def)

lemma fv_transaction_subst_unfold:
"fv_{lsst} (transaction_strand T · lsst ϑ) =
fv_{lsst} (transaction_receive T · lsst ϑ) ∪ fv_{lsst} (transaction_checks T · lsst ϑ) ∪
fv_{lsst} (transaction_updates T · lsst ϑ) ∪ fv_{lsst} (transaction_send T · lsst ϑ)"
by (metis fv_{lsst}_append unlabel_append append_assoc transaction_strand_def subst_lsst_append)

lemma bvars_transaction_unfold:
"bvars_transaction T =
bvars_{lsst} (transaction_receive T) ∪ bvars_{lsst} (transaction_checks T) ∪
bvars_{lsst} (transaction_updates T) ∪ bvars_{lsst} (transaction_send T)"
by (metis bvars_{lsst}_append unlabel_append append_assoc transaction_strand_def)

lemma bvars_transaction_subst_unfold:
"bvars_{lsst} (transaction_strand T · lsst ϑ) =
bvars_{lsst} (transaction_receive T · lsst ϑ) ∪ bvars_{lsst} (transaction_checks T · lsst ϑ) ∪
bvars_{lsst} (transaction_updates T · lsst ϑ) ∪ bvars_{lsst} (transaction_send T · lsst ϑ)"
by (metis bvars_{lsst}_append unlabel_append append_assoc transaction_strand_def subst_lsst_append)

lemma bvars_wellformed_transaction_unfold:
assumes "wellformed_transaction T"
shows "bvars_transaction T = bvars_{lsst} (transaction_checks T)" (is ?A)
and "bvars_{lsst} (transaction_receive T) = {}" (is ?B)
and "bvars_{lsst} (transaction_updates T) = {}" (is ?C)
and "bvars_{lsst} (transaction_send T) = {}" (is ?D)
proof -
have 0: "list_all is_Receive (unlabel (transaction_receive T))"
"list_all is_Update (unlabel (transaction_updates T))"
"list_all is_Send (unlabel (transaction_send T))"
using assms unfolding wellformed_transaction_def by metis+
have "filter is_NegChecks (unlabel (transaction_receive T)) = []" 
"filter is_NegChecks (unlabel (transaction_updates T)) = []" 
"filter is_NegChecks (unlabel (transaction_send T)) = []"

using list_all_filter_nil[OF 0(1), of is_NegChecks] 
list_all_filter_nil[OF 0(2), of is_NegChecks] 
list_all_filter_nil[OF 0(3), of is_NegChecks] 
stateful_strand_step.distinct_disc(11,21,29,35,39,41)

by blast


using bvars_transaction_unfold[of T] 
bvars transaction_unfold[of "unlabel (transaction_receive T)"] 
bvars transaction_unfold[of "unlabel (transaction_updates T)"] 
bvars transaction_unfold[of "unlabel (transaction_send T)"]

by (metis bvars_stateful_strand_def UnionE emptyE list.set(1) list.simps(8) subsetI subset_Un_eq sup_commute)+

qed

lemma transaction_strand_memberD[dest]:
assumes "x ∈ set (transaction_strand T)"
shows "x ∈ set (transaction_receive T) ∨ x ∈ set (transaction_checks T) ∨ x ∈ set (transaction_updates T) ∨ x ∈ set (transaction_send T)"
using assms by (simp add: transaction_strand_def)

lemma transaction_strand_unlabel_memberD[dest]:
assumes "x ∈ set (unlabel (transaction_strand T))"
shows "x ∈ set (unlabel (transaction_receive T)) ∨ x ∈ set (unlabel (transaction_checks T)) ∨ x ∈ set (unlabel (transaction_updates T)) ∨ x ∈ set (unlabel (transaction_send T))"
using assms by (simp add: unlabel_def transaction_strand_def)

lemma wellformed_transaction_strand_memberD[dest]:
assumes "wellformed_transaction T" and "(1,x) ∈ set (transaction_strand T)"
shows "(1,x) ∈ set (transaction_receive T) ∨ (1,x) ∈ set (transaction_checks T) ∨ (1,x) ∈ set (transaction_updates T) ∨ (1,x) ∈ set (transaction_send T)"
using assms(2) by auto

thus !?A !?A" !?B !?B' !?C !?C' !?D !?D' "?E !?E' !?F !?F' !?G !?G' !?H !?H'

using wellformed_transaction_cases[OF assms(1)] by fast+

qed

lemma wellformed_transaction_strand_unlabel_memberD[dest]:
assumes "wellformed_transaction T" and "x ∈ set (unlabel (transaction_strand T))"
shows "x ∈ set (unlabel (transaction_receive T))" (is "!?A !?A'")
"x ∈ set (unlabel (transaction_checks T))" (is "!?B !?B'")
"x ∈ set (unlabel (transaction_updates T))" (is "!?C !?C'" 
"!?D !?D'"
"!?E !?E'" 
"!?F !?F'" 
"!?G !?G'" 
"!?H !?H'"

using wellformed_transaction_cases[OF assms(1)] by fast+

proof

have "(?A) ∈ set (unlabel (transaction_receive T)) ∨ (?A) ∈ set (unlabel (transaction_checks T)) ∨ (?A) ∈ set (unlabel (transaction_updates T)) ∨ (?A) ∈ set (unlabel (transaction_send T))"
using assms(2) by auto

thus !?A !?A" !?B !?B' !?C !?C' !?D !?D' "?E !?E' !?F !?F' !?G !?G' !?H !?H'

using wellformed_transaction_cases[OF assms(1)] by fast+

qed
3 Stateful Protocol Verification

thus "A" "B" "C" "D"
using wellformed_transaction_unlabel_cases[of assms(1)] by fast+ qed

lemma wellformed_transaction_send_receive_trm_cases:
  assumes T: "wellformed_transaction T"
  shows "t ∈ trms_sst (transaction_send T) ⟺ ∃ ts. t ∈ set ts ∧ send(ts) ∈ set (unlabel (transaction_send T))"
  and "t ∈ trms_sst (transaction_send T) ⟺ ∃ ts. t ∈ set ts ∧ receive(ts) ∈ set (unlabel (transaction_send T))"
  using wellformed_transaction_unlabel_cases(1,4)[OF T]
by fastforce

lemma wellformed_transaction_send_receive_subset_trm_cases:
  assumes T: "wellformed_transaction T"
  shows "t ∈ trms_sst (transaction_send T) ⟺ ∃ ts. t ∈ set ts ∧ receive(ts) ∈ set (unlabel (transaction_send T))"
  and "t ∈ trms_sst (transaction_send T) ⟺ ∃ ts. t ∈ set ts ∧ send(ts) ∈ set (unlabel (transaction_send T))"
  using wellformed_transaction_unlabel_cases(1,4)[OF T]
proof -
  assume "t ∈ trms_sst (transaction_send T)" 
  then obtain s where s: "s ∈ trms_sst (transaction_send T)" "t = s · _" by blast
  hence "∃ ts. s ∈ set ts ∧ receive(ts) ∈ set (unlabel (transaction_send T))"
    using wellformed_transaction_send_receive_trm_cases[of _ _ T] by simp
  thus "∃ ts. t ∈ set ts ∧ receive(ts) ∈ set (unlabel (transaction_send T) · t)"
    using s(2) unlabel_subst[of _ _ ] stateful_strand_step_subset_inI(2)
    by (smt image_eqI list.set_map)

next
  assume "t ∈ trms_sst (transaction_send T)" 
  then obtain s where s: "s ∈ trms_sst (transaction_send T)" "t = s · _" by blast
  hence "∃ ts. s ∈ set ts ∧ send(ts) ∈ set (unlabel (transaction_send T))"
    using wellformed_transaction_send_receive_trm_cases[of _ _ T] by simp
  thus "∃ ts. t ∈ set ts ∧ send(ts) ∈ set (unlabel (transaction_send T) · t)"
    using s(2) unlabel_subst[of _ _ ] stateful_strand_step_subset_inI(1)
    by (smt image_eqI list.set_map)
qed

lemma wellformed_transaction_send_receive_fn_subset:
  assumes T: "wellformed_transaction T"
  shows "t ∈ trms_sst (transaction_send T) ⟺ fv t ⊆ fv_transaction T" (is "?A ⟺ ?A")
  and "t ∈ trms_sst (transaction_send T) ⟺ fv t ⊆ fv_transaction T" (is "?B ⟺ ?B")
  using wellformed_transaction_send_receive_trm_cases[of _ T, of _]
proof -
  let ?P = "∃ ts. t ∈ set ts ∧ receive(ts) ∈ set (unlabel (transaction_strand T))"
  let ?Q = "∃ ts. t ∈ set ts ∧ send(ts) ∈ set (unlabel (transaction_strand T))"
  have "*: t ∈ trms_sst (transaction_send T) ⟺ ?P" "t ∈ trms_sst (transaction_send T) ⟺ ?Q"
    using wellformed_transaction_send_receive_trm_cases[of _ _ T] by simp
  unfolding transaction_strand_def by force+
  show "?A ⟺ ?A" using *(1) by (induct "transaction_strand T") (simp, force)
  show "?B ⟺ ?B" using *(2) by (induct "transaction_strand T") (simp, force)
qed

lemma dual_wellformed_transaction_ident_cases: 
  "list_all is_Assignment (unlabel S) ⟺ dual_sst S = S"
  "list_all is_Check (unlabel S) ⟺ dual_sst S = S"
  "list_all is_Update (unlabel S) ⟺ dual_sst S = S"
proof (induction S)
3.1 Protocol Transactions

case (Cons s S)
obtain l x where "s = (l,x)" by moura
{ case 1 thus ?case using Cons s unfolding unlabel_def dual lsst_def by (cases x) auto }
{ case 2 thus ?case using Cons s unfolding unlabel_def dual lsst_def by (cases x) auto }
{ case 3 thus ?case using Cons s unfolding dual lsst_def by (cases x) auto }
qed simp_all

lemma wellformed_transaction_wf_sst:
  fixes T :: "('a, 'b, 'c, 'd) prot_transaction"
  assumes T: "wellformed_transaction T"
  shows "wf' sst (fst ` set (transaction_decl T ()) ∪ set (transaction_fresh T))
    (unlabel (dual lsst (transaction_strand T)))"
  and "fv_transaction T ∩ bvars_transaction T = {}"
  using T unfolding wellformed_transaction_def by simp_all

lemma dual_wellformed_transaction_ident_cases' [dest]:
  assumes "wellformed_transaction T"
  shows "dual lsst (transaction_checks T) = transaction_checks T" (is ?A)
    "dual lsst (transaction_updates T) = transaction_updates T" (is ?B)
  proof -
    have "list_all is_Check_or_Assignment (unlabel (transaction_checks T))"
      "list_all is_Update (unlabel (transaction_updates T))"
    using assms is_Check_or_Assignment_iff unfolding wellformed_transaction_def by auto
    thus ?A ?B
      using dual lsst _list_all_same(9)[of "transaction_checks T"]
      dual lsst _list_all_same(8)[of "transaction_updates T"]
      by (blast, blast)
  qed

lemma dual_transaction_strand:
  assumes "wellformed_transaction T"
  shows "dual lsst (transaction_strand T) =
    dual lsst (transaction_receive T) @
    transaction_checks T @
    transaction_updates T @
    dual lsst (transaction_send T)"
  using dual_wellformed_transaction_ident_cases'[OF assms] dual lsst _append unfolding transaction_strand_def by metis

lemma dual_unlabel_transaction_strand:
  assumes "wellformed_transaction T"
  shows "unlabel (dual lsst (transaction_strand T)) =
    (unlabel (dual lsst (transaction_receive T))) @
    (unlabel (transaction_checks T)) @
    (unlabel (transaction_updates T)) @
    (unlabel (dual lsst (transaction_send T)))"
  using dual_transaction_strand[OF assms] by (simp add: unlabel_def)

lemma dual_transaction_strand_subst:
  assumes "wellformed_transaction T"
  shows "dual lsst (transaction_strand T · t · δ) =
    (dual lsst (transaction_receive T) @
    transaction_checks T @
    transaction_updates T @
    dual lsst (transaction_send T)) · t · δ"
  proof -
    have "dual lsst (transaction_strand T · t · δ) = dual lsst (transaction_strand T) · t · δ"
    using dual lsst _subst by metis
    thus ?thesis using dual_transaction_strand[OF assms] by argo
  qed

lemma dual_transaction_ik_is_transaction_send:
  assumes "wellformed_transaction T"
  shows "ik_sst (unlabel (dual lsst (transaction_strand T))) = trms_sst (unlabel (transaction_send T))"
  (is "?A = ?B")
  proof -
    { fix t assume "t ∈ ?A"
      then obtain ts where ts:
        "t ∈ set ts" "receive(ts) ∈ set (unlabel (dual lsst (transaction_strand T)))"
      qed
by (auto simp add: ik_sst_def)
hence ": "send(ts) ∈ set (unlabel (transaction_strand T))" using dual_sst_unlabel_steps_iff(1) by metis
have "t ∈ ?B"
  using ts(1) wellformed_transaction_strand_unlabel_memberD[OF assms *, of ts] by force
 } moreover {
  fix t assume "t ∈ ?B"
  then obtain ts where ts: "t ∈ set ts" "send⟨ts⟩ ∈ set (unlabel (transaction_send T))"
  using wellformed_transaction_unlabel_cases[OF assms] by fastforce
  hence "receive⟨ts⟩ ∈ set (unlabel (dual lsst (transaction_send T)))"
  using dual_unlabel_transaction_send[OF assms] by simp
  hence "receive⟨ts⟩ ∈ set (unlabel (dual lsst (transaction_strand T)))" by auto
  hence "t ∈ ?A" using ts(1) by (auto simp add: ik_sst_def)
} ultimately show "?A = ?B" by auto

qed

lemma dual_transaction_ik_is_transaction_send':
fixes δ ::"('a,'b,'c,'d) prot_subst"
assumes "wellformed_transaction T"
shows "ik_sst(unlabel (dual lsst (transaction_strand T · lsst δ))) = trms_sst(unlabel (transaction_send T)) · set δ "
proof -
  show "?thesis" by (auto simp add: ik_sst subst dual_lsst subst)

lemma db_sst_transaction_prefix_eq:
assumes T: "wellformed_transaction T" and S: "prefix S (transaction_receive T@transaction_checks T)"
shows "db_lsst A = db_lsst (A@dual lsst (S · lsst δ))"
proof -
  let ?T1 = "transaction_receive T"
  let ?T2 = "transaction_checks T"
  have "prefix (unlabel S) (unlabel (?T1@?T2))" using S prefix_unlabel by blast
  have "list_all is_Receive (unlabel (?) S) (unlabel (?) (???))" using S prefix_unlabel by blast
  have "list_all is_Check_or_Assignment (unlabel ?T2)"
    using T by (simp_all add: wellformed_transaction_def)
  hence "∀b ∈ set (unlabel (?T1)). ¬is_Insert b ∧ ¬is_Delete b"
  have "∀b ∈ set (unlabel (?T2)). ¬is_Insert b ∧ ¬is_Delete b"
    by (metis mono_tags, lifting) Ball_set stateful_strand_step.distinct_disc(16,18),
    metis (mono_tags, lifting) Ball_set stateful_strand_step.distinct_disc(24,26,33,35,37,39)
  hence "∀b ∈ set (unlabel (?T1)?T2)). ¬is_Insert b ∧ ¬is_Delete b"
    by (auto simp add: unlabel_def)
  hence "∀b ∈ set (unlabel (?T2)). ¬is_Insert b ∧ ¬is_Delete b"
    using * unfolding prefix_def by fastforce
  hence "∀b ∈ set (unlabel (dual_sst S) · sst δ). ¬is_Insert b ∧ ¬is_Delete b"
    by (induction S)
    case (Cons a S)
    then obtain l b where "a = (l,b)" by (metis surj_pair)
    thus ?case
      using Cons unfolding dual_sst_def unlabel_def subst_apply_stateful_strand_def
      by (cases b) auto
  qed simp
  hence "∀b ∈ set (unlabel (dual_sst (S · sst δ))). ¬is_Insert b ∧ ¬is_Delete b"
    by (metis dual_sst_subst_unlabel)
  show "?thesis" using db_sst_no_upd_append[OF **] unlabel_append
unfolding _db_{s,t}_def by metis

qed

lemma _db_{s,t}_dual_{s,t}_set_ex:
assumes "d ∈ set (db_{lsst} (dual_{lsst} A (lsst ϑ) I D))"
"∀ t u. insert⟨t,u⟩ ∈ set (unlabel A) −→ (∃ s. u = Fun (Set s) [])"
"∀ t u. delete⟨t,u⟩ ∈ set (unlabel A) −→ (∃ s. u = Fun (Set s) [])"
shows "∃ s. snd d = Fun (Set s) []"
using assms proof (induction A arbitrary: D)
case (Cons a A)
obtain l b where a: "a = (l,b)"
by (metis surj_pair)
have 1: "unlabel (dual_{lsst} (a#A) (lsst ϑ)) = receive⟨ts·listϑ⟩ #unlabel (dual_{lsst} A (lsst ϑ))"
when "b = send⟨ts⟩" for ts
by (simp add: a that subst_lsst_unlabel_cons)
have 2: "unlabel (dual_{lsst} (a#A) (lsst ϑ)) = send⟨ts·listϑ⟩ #unlabel (dual_{lsst} A (lsst ϑ))"
when "b = receive⟨ts⟩" for ts
by (simp add: a that subst_lsst_unlabel_cons)
have 3: "unlabel (dual_{lsst} (a#A) (lsst ϑ)) = (b·sstpϑ) #unlabel (dual_{lsst} A (lsst ϑ))"
when "¬ ∃ ts. b = send⟨ts⟩ ∨ b = receive⟨ts⟩" using a that dual_{lsst}_Cons subst_lsst_unlabel_cons[of l b]
by (cases b) auto
show ?case using 1 2 3 a Cons
by (cases b) fastforce+
qed simp

lemma is_Fun_SetE[elim]:
assumes t: "is_Fun_Set t"
obtains s where "t = Fun (Set s) []"
proof (cases t)
case (Fun f T)
then obtain s where "f = Set s" using t
unfolding is_Fun_Set_def
by (cases f) moura+
moreover have "T = []" using Fun t
unfolding is_Fun_Set_def
by (cases T) auto
ultimately show ?thesis using Fun that
by fast
qed (use t is_Fun_Set_def in fast)

lemma Fun_Set_InSet_iff:
"(u = ⟨a: Var x ∈ Fun (Set s) []⟩) ←→
(is_InSet u ∧ is_Var (the_elem_term u) ∧ is_Fun_Set (the_set_term u) ∧
the_Set (the_Fun (the_set_term u)) = s ∧ the_Var (the_elem_term u) = x ∧ the_check u = a)"
(is "?A ←→ ?B")
proof
show "?A" unfolding is_Fun_Set_def by auto
assume B: ?B
thus ?A
proof (cases u)
case (InSet b t t')
hence "b = a" "t = Var x" "t' = Fun (Set s) []"
using B by (simp, fastforce, fastforce)
thus ?thesis using InSet by fast
qed auto
qed

lemma Fun_Set_NotInSet_iff:
"(u = ⟨Var x not in Fun (Set s) []⟩) ←→
(is_NegChecks u ∧ bvars_{ss} u = [] ∧ the_eqs u = [] ∧ length (the_ins u) = 1 ∧
is_Var (fst (hd (the_ins u))) ∧ is_Fun_Set (snd (hd (the_ins u)))) ∧
the_Set (the_Fun (snd (hd (the_ins u)))) = s ∧ the_Var (fst (hd (the_ins u))) = x"
3 Stateful Protocol Verification

(is "?A ↔ ?B")

proof
  show "?A ⇒ ?B" unfolding is_Fun_Set_def by auto

assume B: ?B
show ?A
proof (cases u)
  case (NegChecks X F F')
  hence "X = []" "F = []"
  using B by auto
  moreover have "fst (hd (the_ins u)) = Var x" "snd (hd (the_ins u)) = Fun (Set s) []"
    using B is_Fun_SetE[of "snd (hd (the_ins u))"]
    by (force, fastforce)
  hence "F' = [(Var x, Fun (Set s) [])]"
  using NegChecks B by (cases "the_ins u") auto
  ultimately show ?thesis
  using NegChecks by fast
qed (use B in auto)

qed

lemma is_Fun_Set_exi: "is_Fun_Set x ↔ (∃ s. x = Fun (Set s) [])"
by (metis prot_fun.collapse(2) term.collapse(2) prot_fun.disc(11) term.disc(2)
term.sel(2,4) is_Fun_Set_def un_Fun1_def)

lemma is_Fun_Set_subst: assumes "is_Fun_Set S'" shows "is_Fun_Set (S' · σ)"
using assms by (fastforce simp add: is_Fun_Set_def)

lemma is_Update_in_transaction_updates: assumes tu: "is_Update t" assumes t: "t ∈ set (unlabel (transaction_strand TT))" assumes vt: "wellformed_transaction TT" shows "t ∈ set (unlabel (transaction_updates TT))"
using t tu vt unfolding transaction_strand_def wellformed_transaction_def list_all_iff by (auto simp add: unlabel_append)

lemma transaction_proj_member: assumes "T ∈ set P" shows "transaction_proj n T ∈ set (map (transaction_proj n) P)"
using assms by simp

lemma transaction_strand_proj: "transaction_strand (transaction_proj n T) = proj n (transaction_strand T)"
proof -
  obtain A B C D E F where "T = Transaction A B C D E F" by (cases T) simp thus ?thesis
  unfolding transaction_strand_proj.simps[of n A B C D E F] by auto
qed

lemma transaction_proj_decl_eq: "transaction_decl (transaction_proj n T) = transaction_decl T"
proof -
  obtain A B C D E F where "T = Transaction A B C D E F" by (cases T) simp thus ?thesis
  unfolding proj_def Let_def by auto
qed

lemma transaction_proj_fresh_eq: "transaction_fresh (transaction_proj n T) = transaction_fresh T"
proof -
  obtain A B C D E F where "T = Transaction A B C D E F" by (cases T) simp
thus \( \text{thesis} \)
using \( \text{transaction_proj.simps[of} \ n \ A \ B \ C \ D \ E \ F] \)
unfolding \( \text{proj_def Let_def} \) by \( \text{auto} \)
\( \text{qed} \)

\text{lemma transaction_proj_trms_subset:}
"trms_transaction (transaction_proj n T) \subseteq \text{trms_transaction} T"
\text{proof -}
obtain \( A \ B \ C \ D \ E \ F \) where "\( T = \text{Transaction} \ A \ B \ C \ D \ E \ F \)" by \( \text{(cases T)} \) simp
thus \( \text{thesis} \)
using \( \text{transaction_proj.simps[of} \ n \ A \ B \ C \ D \ E \ F] \) \( \text{trms._proj_subset(1)[of n]} \)
unfolding \( \text{transaction_fresh_def Let_def transaction_strand_def} \) by \( \text{auto} \)
\( \text{qed} \)

\text{lemma transaction_proj_vars_subset:}
"\text{vars_transaction} (transaction_proj n T) \subseteq \\text{vars_transaction} T"
\text{proof -}
obtain \( A \ B \ C \ D \ E \ F \) where "\( T = \text{Transaction} \ A \ B \ C \ D \ E \ F \)" by \( \text{(cases T)} \) simp
thus \( \text{thesis} \)
using \( \text{transaction_proj.simps[of} \ n \ A \ B \ C \ D \ E \ F] \)
\( \text{sst-vars_proj_subset(3)[of n "transaction_strand T"]} \)
unfolding \( \text{transaction_fresh_def Let_def transaction_strand_def} \) by \( \text{simp} \)
\( \text{qed} \)

\text{lemma transaction_proj_labels:}
fixes \( T::(\text{\texttt{\texttt{\texttt{}\texttt{\texttt{}\texttt{('a,'b,'c,'d) prot_transaction}}}}} \)
shows "\( \text{list_all (λa. has_LabelN l a \lor has_LabelS a)} (\text{transaction_strand} (\text{transaction_proj l} T))\)"
\text{proof -}
define \( h \) where "\( h \equiv \lambda a::(\text{\texttt{\texttt{\texttt{\texttt{\texttt{\texttt{}\texttt{('a,'b,'c,'d) prot_strand_step}}}}} \text{\texttt{\texttt{}has_LabelN l a \lor has_LabelS a}}}) \)
let \( ?f = \"filter h\"
let \( ?g = \"list_all h\"
obtain \( T1 \ T2 \ T3 \ T4 \ T5 \ T6 \) where "\( T = \text{Transaction} \ T1 \ T2 \ T3 \ T4 \ T5 \ T6 \)" by \( \text{(cases T)} \) simp
note \( 0 = \text{transaction_proj.simps[unfolded Let_def, of l T1 T2 T3 T4 T5 T6]} \)
show \( \text{thesis using T 0 unfolding list_all_iff proj_def} \) by \( \text{auto} \)
\( \text{qed} \)

\text{lemma transaction_proj_ident_iff:}
fixes \( T::(\text{\texttt{\texttt{}\texttt{\texttt{}\texttt{\texttt{}\texttt{\texttt{('a,'b,'c,'d) prot_transaction}}}}}}} \)
shows "\( \text{list_all (λa. has_LabelN l a \lor has_LabelS a)} (\text{transaction_strand} T) \leftarrow\rightarrow \text{transaction_proj l T} = T\)"
(is "\( \text{?A \leftarrow\rightarrow \ ?B}\)"
\text{proof}
obtain \( T1 \ T2 \ T3 \ T4 \ T5 \ T6 \) where "\( T = \text{Transaction} \ T1 \ T2 \ T3 \ T4 \ T5 \ T6 \)" by \( \text{(cases T)} \) simp
hence "\( \text{transaction_strand} T = T30T40T50T6\)" unfolding \( \text{transaction_strand_def} \) by \( \text{simp} \)
thus "\( \text{?A \leftarrow\rightarrow \ ?B}\)"
using \( T \text{ transaction_proj.simps[unfolded Let_def, of l T1 T2 T3 T4 T5 T6]} \)
unfolding \( \text{list_all_iff proj_def} \) by \( \text{auto} \)
show "\( \text{?B \leftarrow\rightarrow \ ?A}\)" using \( \text{transaction_proj_labels[of l T]} \) by \( \text{presburger} \)
\( \text{qed} \)

\text{lemma transaction_proj_idem:}
fixes \( T::(\text{\texttt{\texttt{}\texttt{\texttt{}\texttt{\texttt{}\texttt{\texttt{('a,'b,'c,'d) prot_transaction}}}}}}} \)
shows "\( \text{transaction_proj l (transaction_proj l T)} = \text{transaction_proj l T}\)"
by \( \text{(meson transaction_proj_ident_iff transaction_proj_labels)} \)

\text{lemma transaction_proj_ball_subst:}
assumes "\( \text{set Ps = (λn. map (transaction_proj n) P) \set L} \)
"\( \forall p \in \text{set Ps}. \ Q \ p\)"
shows "∀ l ∈ set L. Q (map (transaction_proj l) P)"
using assms by auto

lemma transaction_star_proj_has_star_labels:
  "list_all has_LabelS (transaction_strand (transaction_star_proj T))"
proof -
  let ?f = "filter has_LabelS"
  obtain T1 T2 T3 T4 T5 T6 where T: "T = Transaction T1 T2 T3 T4 T5 T6" by (cases T) simp
  hence T': "transaction_strand (transaction_star_proj T) = ?f T3@?f T4@?f T5@?f T6"
  unfolding transaction_star_proj.simps[unfolded Let_def, of T1 T2 T3 T4 T5 T6]
  using transaction_star_proj.simps by auto
  unfolding transaction_strand_def by auto
  show ?thesis
  using Ball_set unfolding T'
  by fastforce
qed

lemma transaction_star_proj_ident_iff:
  "list_all has_LabelS (transaction_strand T) ←→ transaction_star_proj T = T" (is "?A ←→ ?B")
proof
  obtain T1 T2 T3 T4 T5 T6 where T: "T = Transaction T1 T2 T3 T4 T5 T6" by (cases T) simp
  hence T': "transaction_strand T = T3@T4@T5@T6" unfolding transaction_strand_def by simp
  show "?A =⇒ ?B" using T T' unfolding list_all_iff by auto
  show "?B =⇒ ?A" using transaction_star_proj_has_star_labels[of T] by auto
qed

lemma transaction_star_proj_negates_transaction_proj:
  "transaction_star_proj (transaction_proj l T) = transaction_star_proj T" (is "?A l T")
  "k ≠ l =⇒ transaction_proj k (transaction_proj l T) = transaction_star_proj T" (is "?B =⇒ ?B'")
proof -
  show "?A l T" for l T
  proof -
    obtain T1 T2 T3 T4 T5 T6 where T: "T = Transaction T1 T2 T3 T4 T5 T6" by (cases T) simp
    thus ?thesis
    by (metis dbproj_def transaction_proj.simps transaction_star_proj.simps proj_dbproj(2))
  qed
  thus "?B =⇒ ?B'"
    by (metis (no_types) has_LabelS_proj_iff_not_has_LabelN proj_elims_label
        transaction_star_proj_ident_iff transaction_strand_proj)
qed

end

3.2 Term Abstraction

theory Term_Abstraction
imports Transactions
begin

3.2.1 Definitions

fun to_abs ("α0") where
  "α0 [] = {}"
| "α0 (Fun (Val n) D)#D) n =
  (if m = n then insert s (α0 D n) else α0 D n)"
| "α0 (_#D) n = α0 D n"

fun abs_apply_term (infixl "·") where
  "Var x · α = Var x"
| "Fun (Val n) T · α = Fun (Abs (α n)) (map (λt. t · α) T)"
| "Fun f T · α = Fun f (map (λt. t · α) T)"

34
3.2 Term Abstraction

definition abs_apply_list (infixl "· αlist" 67) where
"M · αlist α ≡ map (λt. t · α α) M"

definition abs_apply_terms (infixl "· αset" 67) where
"M · αset α ≡ (λ(s,t). (s · α α, t · α α)) M"

definition abs_apply_pairs (infixl "· αpairs" 67) where
"F · αpairs α ≡ map (λ(t,s). (s · α α, t · α α)) F"

definition abs_apply_strand_step (infixl "· αstp" 67) where
"s · αstp α ≡ (case s of
  (l, send ⟨ts⟩) ⇒ (l, send ⟨ts · αlist α⟩)
  | (l, receive ⟨ts⟩) ⇒ (l, receive ⟨ts · αlist α⟩)
  | (l, ⟨ac: t = t'⟩) ⇒ (l, ⟨ac: (t · α α) = (t' · α α)⟩)
  | (l, insert ⟨t, t'⟩) ⇒ (l, insert ⟨t · α α, t' · α α⟩)
  | (l, delete ⟨t, t'⟩) ⇒ (l, delete ⟨t · α α, t' · α α⟩)
  | (l, get ⟨ac: t ∈ t'⟩) ⇒ (l, get ⟨ac: (t · α α) ∈ (t' · α α)⟩)
  | (l, ∀X (∨̸= F ∨ ∈ F')) ⇒ (l, ∀X (∨̸= (F · αpairs α) ∨ ∈ (F' · αpairs α))))"

definition abs_apply_strand (infixl "· αst" 67) where
"S · αst α ≡ map (λx. x · αstp α) S"

3.2.2 Lemmata

lemma to_abs_alt_def:
"α₀ D n = {s. ∃S. (Fun (Val n) []) · Fun (Set s) S) ∈ set D}"
by (induct D n rule: to_abs.induct) auto

lemma abs_term_apply_const[simp]:
"is_Val f =⇒ Fun f [] · α a = Fun (Abs (a (the_Val f))) []"
"¬is_Val f =⇒ Fun f [] · α a = Fun f []"
by (cases f; auto)+

lemma abs_eq_if_no_Val:
assumes "∀f ∈ funs_term t. ¬is_Val f"
shows "t · α α = t · α α"
proof (induction t)
  case (Fun f T) thus ?case by (cases f) simp_all
qed simp

lemma abs_list_set_is_set_abs_set: "set (M · αlist α) = (set M) · αset α"
unfolding abs_apply_list_def abs_apply_terms_def
by simp

lemma abs_set_empty[simp]: "{} · αset α = {}"
unfolding abs_apply_terms_def

lemma abs_in:
assumes "t ∈ M"
shows "t · α α ∈ M · αset α"
using assms unfolding abs_apply_terms_def
by (induct t α rule: abs_apply_term.induct) blast

lemma abs_set_union: "(A ∪ B) · αset a = (A · αset a) ∪ (B · αset a)"
unfolding abs_apply_terms_def

lemma abs_subterms: "subterms (t · α α) = subterms t · αset α"
proof (induction t)
  case (Fun f T) thus ?case by (cases f) (auto simp add: abs_apply_terms_def)

35
3 Stateful Protocol Verification

qed (simp add: abs_apply_terms_def)

lemma abs_subterms_in: "s ∈ subterms t ⟷ s · α a ∈ subterms (t · α a)"
proof (induction t)
case (Fun f T)
  thus ?case by (cases f) auto
qed simp

lemma abs_ik_append: "((ksst (AΩB) · set I) · αset a = (ksst A · set I) · αset a ∪ (ksst B · set I) · αset a)"
unfolding abs_apply_terms_def ik_sst_def by fastforce

lemma to_abs_in:
assumes "(Fun (Val n) [], Fun (Set s) []) ∈ set D"
shows "s ∈ α0 D n"
using assms by (induct rule: to_abs.induct) auto

lemma to_abs_empty_iff_notin_db:
"Fun (Val n) [] · α α0 D = Fun (Abs {}) [] ←→ (∄ S. (Fun (Val n) [], Fun (Set s) S) ∈ set D)"
by (simp add: to_abs_alt_def)

lemma to_abs_list_insert:
assumes "Fun (Val n) [] ≠ t"
shows "α0 D n = α0 (List.insert (t,s) D) n"
using assms to_abs_alt_def[of D n] to_abs_alt_def[of "List.insert (t,s) D" n] by auto

lemma to_abs_list_insert':
"α0 D n - {s} = α0 (filter (λd. ∄ S. d = (Fun (Val n) [], Fun (Set s) S)) D) n"
using to_abs_alt_def[of D n] to_abs_alt_def[of "filter (λd. ∄ S. d = (Fun (Val n) [], Fun (Set s) S)) D" n] by auto

lemma to_abs_db_sst_appen:
assumes "∀ u s. insert(u, s) ∈ set B ⟷ Fun (Val n) [] ≠ u · I"
  and "∀ u s. delete(u, s) ∈ set B ⟷ Fun (Val n) [] ≠ u · I"
shows "α0 (db' sst A I D) n = α0 (db' sst (AΩB) I D) n"
using assms proof (induction B rule: List.rev_induct)
case (snoc b B)
hence IH: "α0 (db' sst A I D) n = α0 (db' sst (AΩB) I D) n" by auto
have #: "∀ u s. b = insert(u, s) ⟶ Fun (Val n) [] ≠ u · I"
  and "∀ u s. b = delete(u, s) ⟷ Fun (Val n) [] ≠ u · I"
    using snoc.prems by simp_all
  show ?case
  proof (cases b)
case (Insert u s)
hence **: "db' sst (AΩB[b]) I D = List.insert (u · I, s · I) (db' sst (AΩB) I D)"
      using db_sst_appen[of "AΩB" "[b]"] by simp
  have "Fun (Val n) [] ≠ u · I" using **(1) Insert by auto
  thus ?thesis using IH ** to_abs_list_insert by metis
next
case (Delete u s)
hence **: "db',sst (A@B) I D = List.removeAll (u · I, s · I) (db',sst (A@B) I D)"
using db',sst_append[of "A@B" "[b]""] by simp
have "Fun (Val n) [] ≠ u · I" using */(2) Delete by auto
thus ?thesis using IH ** to_abs_list_remove_all by metis
qed simp_all add: db',sst_no_upd_append[of "[b]" "A@B"] IH
qed simp

lemma to_abs_neq_imp_db_update:
assumes "α0 (db',sst A I) n ≠ α0 (db',sst (A@B) I) n"
shows "∃ u s. u · I = Fun (Val n) [] ∧ (insert⟨u,s⟩ ∈ set B ∨ delete⟨u,s⟩ ∈ set B)"
proof -
  { fix D have ?thesis when "α0 D n ≠ α0 (db',sst B I D) n" using that
    proof (induction B I D rule: db',sst_induct)
    case 2 thus ?case
      by (metis db',sst.simps(2) list.set_intros(1,2) subst_apply_pair_pair to_abs_list_insert)
    next
case 3 thus ?case
      by (metis db',sst.simps(3) list.set_intros(1,2) subst_apply_pair_pair to_abs_list_remove_all)
    qed simp_all
  } thus ?thesis using assms by (metis db',sst_append db',sst_def)
qed

lemma abs_term_subst_eq:
fixes δ ϑ ::"(('a,'b,'c,'d) prot_fun, ('e,'f prot_atom) term × nat) subst"
assumes "∀ x ∈ fv t. δ x · α a = ϑ x · α b"
and "∀ n T. Fun (Val n) T ∈ subterms t"
shows "t · δ · α a = t · ϑ · α b"
using assms
proof (induction t)
  case (Fun f T) thus ?case
    proof (cases f)
      case (Val n)
      hence False using Fun.prems(2) by blast
      thus ?thesis by metis
    qed auto
  qed simp

lemma abs_term_subst_eq':
fixes δ ϑ ::"(('a,'b,'c,'d) prot_fun, ('e,'f prot_atom) term × nat) subst"
assumes "∀ x ∈ fv t. δ x · α a = ϑ x"
and "∀ n T. Fun (Val n) T ∈ subterms t"
shows "t · δ · α a = t · ϑ"
using assms
proof (induction t)
  case (Fun f T) thus ?case
    proof (cases f)
      case (Val n)
      hence False using Fun.prems(2) by blast
      thus ?thesis by metis
    qed auto
  qed simp

lemma abs_val_in_funs_term:
assumes "f ∈ funs_term t" "is_Val f"
shows "Abs (α (the_Val f)) ∈ funs_term (t · α)"
using assms by (induct t α rule: abs_apply_term.induct) auto
end
theory Stateful_Protocol_Model
  imports Stateful_Protocol_Composition_and_Typing.Stateful_Compositionality Transactions Term_Abstraction
begin

3.3 Stateful Protocol Model

locale stateful_protocol_model = fixes arity::"'fun ⇒ nat" and arity::"'sets ⇒ nat" and public::"'fun ⇒ bool" and Ana::"'fun ⇒ (((('fun,'atom::finite,'sets,'lbl) prot_fun, nat) term list × nat list)" and Γ::"'fun ⇒ 'atom option" and label_witness1::"'lbl" and label_witness2::"'lbl"
assumes Ana_f_assm1: "∀ f. let (K, M) = Ana f f in (∀ k ∈ subterms set (set K). is_Fun k −→ (is_Fu (the_Fun k)) ∧ length (args k) = arity f f (the_Fun k))" and Ana_f_assm2: "∀ f. let (K, M) = Ana f f in (∀ i ∈ fv set (set K) ∪ set M. i < arity f f)" and public_assm: "∀ f. arity f f > (0::nat) → public f f" and Γ_f_assm: "∀ f. arity f f = (0::nat) → Γ f f ≠ None" and label_witness_assm: "label_witness1 ≠ label_witness2"

begin

lemma Ana_f_assm1_alt:
  assumes "Ana f f = (K,M)" "k ∈ subterms set (set K)"
  shows "(∃ x. k = Var x) ∨ (∃ h T. k = Fun (Fu h) T ∧ length T = arity h)"
proof (cases k)
  case (Fun g T)
  let ?P = "λ k. is_Fun k −→ (is_Fu (the_Fun k)) ∧ length (args k) = arity f f (the_Fun k))" 
  let ?Q = "λ K M. (∀ k ∈ subterms set (set K). ?P k)" 
  have "?Q (fst (Ana f f)) (snd (Ana f f))" using Ana_f_assm1 split_beta[of "Ana f f"] by meson 
  hence "?Q K M" using assms(1) by simp 
  hence "?P k" using assms(2) by blast 
  thus ?thesis using Fun by (cases g) auto 
qed simp

lemma Ana_f_assm2_alt:
  assumes "Ana f f = (K,M)" "i ∈ fv set (set K) ∪ set M"
  shows "i < arity f f"
using Ana_f_assm2 assms by fastforce

3.3.2 Definitions

fun arity where
  "arity (Fu f) = arity f f"
| "arity (Set s) = arity s s"
| "arity (Val _) = 0"
| "arity (Abs _) = 0"
| "arity Pair = 2"
| "arity (Attack _) = 0"
| "arity OccursFact = 2"
| "arity OccursSec = 0"
| "arity (PubConst _ _) = 0"

fun public where
  "public (Fu f) = public f f"
| "public (Set s) = (arity s s > 0)"
| "public (Val n) = False"
| "public (Abs _) = False"
| "public Pair = True"
fun Ana where
"Ana (Fun (Fu f) T) = (if arity f f = length T ∧ arity f f > 0 then let (K,M) = Ana f f in (K ,map (!) T) else ([], []))"

fun Γ where
"Γ (Var v) = Γ v v" 
| "Γ (Fun f T) = (if arity f f = 0 then case f of (Fu g) ⇒ TAtom Value |
| (Val _) ⇒ TAtom AbsValue |
| (Set _) ⇒ TAtom SetType |
| (Attack _) ⇒ TAtom AttackType |
| OccursSec ⇒ TAtom OccursSecType |
| _ ⇒ TAtom Bottom) else TComp f (map Γ T))"

lemma Γconsts_simp[simp]:
"arity f f = 0 ⇒ Γ (Fun (Fun (Fu g) g of Some a ⇒ Atom a | None ⇒ Bottom)
| (Val _) ⇒ TAtom Value |
| (Abs _) ⇒ TAtom AbsValue |
| (Set _) ⇒ TAtom SetType |
| (Attack _) ⇒ TAtom AttackType |
| _ ⇒ TAtom Bottom)" 
| Γ (Fun (Val n) g of Some a ⇒ Atom a | None ⇒ Bottom)
| Γ (Fun (Abs b) g of Some a ⇒ Atom a | None ⇒ Bottom)
| Γ (Fun (Set s) g of Some a ⇒ Atom a | None ⇒ Bottom)
| Γ (Fun (Attack x) g of Some a ⇒ Atom a | None ⇒ Bottom)
| Γ (Fun (OccursSec a) g of Some a ⇒ Atom a | None ⇒ Bottom)
| Γ (Fun (PubConst a t) g of Some a ⇒ Atom a | None ⇒ Bottom)

by simp

lemma ΓFu_simp[simp]:
"arity f f = 0 ⇒ Γ (Fun (Fun (Fu f) T) = TComp (Fun f) (map Γ T))" (is "?A1 ⇒ ?A2")
| "arity f f = 0 ⇒ Γ (Fun (Fun (Fun (Fu f) T) = TAtom (Atom a))" (is "?B1 ⇒ ?B2 ⇒ ?B3")
| "arity f f = 0 ⇒ Γ (Fun (Fun (Fun (Fun (Fu f) T) = TAtom Bottom))" (is "?C1 ⇒ ?C2 ⇒ ?C3")
| Γ (Fun (Fu f) T) ≠ TAtom Value" (is ?D)
| Γ (Fun (Fu f) T) ≠ TAtom AttackType" (is ?E)
| Γ (Fun (Fun (Fu f) T) ≠ TAtom OccursSecType" (is ?F)

proof -
| show "?A1 ⇒ ?A2" by simp
| show "?B1 ⇒ ?B2 ⇒ ?B3" by simp
| show "?C1 ⇒ ?C2 ⇒ ?C3" by simp
| show ?D by (cases "Γ (Fu f)") simp_all
| show ?E by (cases "Γ (Fu f)") simp_all
| show ?F by (cases "Γ (Fu f)") simp_all

qed

lemma ΓSet_simp[simp]:
"arity, s ≠ 0 ⇒ Γ (Fun (Set s) T) = TComp (Set s) (map Γ T)"
3 Stateful Protocol Verification

Γ (Fun (Set s) T) = TAtom SetType ∨ Γ (Fun (Set s) T) = TComp (Set s) (map Γ T)
Γ (Fun (Set s) T) ≠ TAtom Value
Γ (Fun (Set s) T) ≠ TAtom (Atom a)
Γ (Fun (Set s) T) ≠ TAtom AttackType
Γ (Fun (Set s) T) ≠ TAtom OccursSecType
Γ (Fun (Set s) T) ≠ TAtom Bottom

by auto

3.3.3 Locale Interpretations

lemma Ana_Fu_cases:
  assumes "Ana (Fun f T) = (K,M)"
  and "f = Fu g"
  and "Ana f g = (K',M')"
  shows "(K,M) = (if arity f g = length T ∧ arity f g > 0
  then (K' ‚ i list (!) T, map ((!) T) M')
  else ([],[]))" (is ?A)
  and "(K,M) = (K' ‚ i list (!) T, map ((!) T) M') ∨ (K,M) = ([],[])" (is ?B)
  proof -
  show ?A using assms by (cases "arity f g = length T ∧ arity f g > 0") auto
  thus ?B by metis
  qed

lemma Ana_Fu_intro:
  assumes "arity f f = length T" "arity f f > 0"
  and "Ana f f = (K',M')"
  shows "Ana (Fun (Fu f) T) = (K' ‚ i list (!) T, map ((!) T) M')"
  using assms by simp

lemma Ana_Fu_elim:
  assumes "Ana (Fun f T) = (K,M)"
  and "f = Fu g"
  and "Ana f g = (K',M')"
  and "(K,M) ≠ ([],[])"
  shows "arity f g = length T" (is ?A)
  and "(K,M) = (K' ‚ i list (!) T, map ((!) T) M')" (is ?B)
  proof -
  show ?A using assms by force
  moreover have "arity f g > 0" using assms by force
  ultimately show ?B using assms by auto
  qed

lemma Ana_nonempty_inv:
  assumes "Ana t ≠ ([],[])"
  shows "∃ f T. t = Fun (Fu f) T ∧ arity f f = length T ∨ arity f f > 0 ∧
  (∃ K M. Ana f f = (K, M) ∧ Ana t = (K' ‚ i list (!) T, map ((!) T) M'))"
  using assms
  proof (induction t rule: term.induct)
    case (1 f T)
    hence *: "arity f f = length T" "0 < arity f f"
    "Ana (Fun (Fu f) T) = (case Ana f f of (K, M) ⇒ (K' ‚ i list (!) T, map ((!) T) M'))"
    using Ana.simps(1)[of f T] unfolding Let_def by metis*
  obtain K M where **: "Ana f f = (K, M)" by (metis surj_pair)
  hence "Ana (Fun (Fu f) T) = (K' ‚ i list (!) T, map ((!) T) M')" using *(3) by simp
  thus ?case using ** *(1,2) by blast
  qed simp_all

lemma assm1:
  assumes "Ana t = (K,M)"
  shows "fv set (set K) ⊆ fv t"
  using assms
  proof (induction t rule: term.induct)
3.3 Stateful Protocol Model

case (Fun f T)
have aux: "fv_set (set K) (set (!) T) ⊆ fv_set (set T)"
when K: "∀ i ∈ fv_set (set K). i < length T"
for K::"((fun, 'atom, 'sets, 'lbl) prot_fun, nat) term list"
proof
  fix x assume "x ∈ fv_set (set K) (set (!) T)"
then obtain k where k: "k ∈ set K" "x ∈ fv (k · (!) T)"
by moura
  have "∀ i ∈ fv k. i < length T" using K k(1) by simp
  thus "x ∈ fv_set (set T)"
  by (metis (no_types, lifting) k(2) contra_subsetD fv_set_mono image_subsetI nth_mem subst_apply_fv_unfold)
qed

lemma assm2:
  assumes "Ana t = (K, M)"
  and "∀ S'. Fun g S' ⊑ t =⇒ length S' = arity g"
  and "k ∈ set K"
  and "Fun f T' ⊑ k"
  shows "length T' = arity f"
using assms
proof
  (induction t rule: term.induct)
  case (Fun g T)
  obtain h where 2: "g = Fu h"
  using Fun.prems(1,3)
  by (cases g) auto
  obtain K' M' where 1: "Ana f h = (K', M')" by moura
  have "(K, M) ≠ ([], [])" using K by simp
  hence "(K, M) = (K' · list (!) T, map ((!) T) M')" "arity f = length T"
  using Ana_Fu_cases(1)[OF Fun.prems f *]
  by presburger+
  hence ?case using aux[of K'] Ana_f_assm2_alt[of *] by auto
  } thus ?case using Fun by (cases f) fastforce+
qed simp

lemma assm4:
  assumes "Ana (Fun f T) = (K, M)"
  shows "set M ⊆ set T"
using assms
proof (cases f)
  case (Fu g)
obtain $K', M'$ where $\ast$: "Ana$_f g = (K',M')" by moura

have "$M = [] \lor (\text{arity}_f g = \text{length} T \land M = \text{map} (((!) T) M'))"
  using Ana Fu_cases(1)[OF assms Fu *]
  by (meson prod.inject)
thus $?thesis$ using Ana$_f$ assm2_alt[OF *] by auto
qed auto

lemma assm5: "Ana t = (K,M) = \Rightarrow K \not= [] \lor M \not= [] = \Rightarrow Ana (t \cdot \delta) = (K \cdot \text{list} \delta, M \cdot \text{list} \delta)"
proof (induction t rule: term.induct)
case [Fun f T]
thus $?case$
proof (cases f)
case [Fu g]
obtain $K' M'$ where $\ast$: "Ana$_f g = (K',M')" by moura

have $**$: "K = K' \cdot \text{list} (\lambda t. t \cdot \delta) T" "M = \text{map} ((\lambda t. t \cdot \delta) T) M'"
  using subst_idx_map[OF **(2), of \delta]
  subst_idx_map'[OF **(1), of \delta]
  by fast+
have $***$: "\forall i \in \text{fv set } (\text{set } K'). i < \text{length } T" "\forall i \in \text{set } M'. i < \text{length } T"
  using **(3,4)
  by auto

have "$K \cdot \text{list} \delta = K' \cdot \text{list} (\lambda t. t \cdot \delta) (\text{map} (((!) T) \delta)) T""$ $M \cdot \text{list} \delta = \text{map} ((\lambda t. t \cdot \delta) T) M'"
  using subst_idx_map[OF ***(2), of \delta]
  subst_idx_map'[OF ***(1), of \delta]
  by fast+
thus $?thesis$ using Fu * **(3,5) by auto
qed auto

 qed

sublocale intruder_model arity public Ana
apply unfold_locales
by (metis assm1, metis assm2, rule Ana.simps, metis assm4, metis assm5)

adhoc_overloading INTRUDER_SYNTH intruder_synth
adhoc_overloading INTRUDER_DEDUCT intruder_deduct

lemma assm6: "\text{arity } c = 0 = \Rightarrow \exists a. \forall X. \Gamma (\text{Fun } c X) = TAtom a" by (cases c) auto

lemma assm7: "0 < \text{arity } f = \Rightarrow \Gamma (\text{Fun } f T) = TComp f (\text{map } \Gamma T)" by auto

lemma assm8: "\text{infinite } \{c. \Gamma (\text{Fun } c []) :: (\text{fun},\text{atom},\text{sets},\text{lbl}) \text{ prot_term}) = TAtom a \land \text{public } c\}"
(is "$?P a")
proof -
let $?T = "\lambda f. (\text{range } f) :: (\text{fun},\text{atom},\text{sets},\text{lbl}) \text{ prot.fun set}"
let $?A = "\lambda f. \forall x::\text{nat} \in \text{UNIV}. \forall y::\text{nat} \in \text{UNIV}. (f x = f y) = (x = y)"
let $?B = "\lambda f. \forall x::\text{nat} \in \text{UNIV}. f x \in {?T f}"
let $?C = "\lambda f. \forall y::(\text{fun},\text{atom},\text{sets},\text{lbl}) \text{ prot.fun} \in {?T f}. \exists x \in \text{UNIV}. y = f x"
let $?D = "\lambda f b. {?T f} \subseteq \{c. \Gamma (\text{Fun } c []) :: (\text{fun},\text{atom},\text{sets},\text{lbl}) \text{ prot_term}) = TAtom b \land \text{public } c\}"

have sub_lmm: "$?P b" when "$?A f" "$?C f" "$?C f" "$?D f b" for b f
proof -
  have "$?C g::\text{nat} \Rightarrow (\text{fun},\text{atom},\text{sets},\text{lbl}) \text{ prot.fun}. \text{bij_betw } g \text{ UNIV } (?T f)"
    using bij_betwI'[of UNIV f "?T f" that(1,2,3)] by blast
  hence "$?D f b" by (metis nat_not_finite bij_betw_finite)
  thus $?thesis$ using infinite_super[OF that(4)] by blast
qed

show $?thesis
proof (cases a)
next
case [Value]
thus $?thesis$ using sub_lmm[of "PubConst Value" a] by force

42
3.3 Stateful Protocol Model

next
  case SetType thus ?thesis using sub_lmm[of "PubConst SetType" a] by fastforce
next
  case AttackType thus ?thesis using sub_lmm[of "PubConst AttackType" a] by fastforce
next
  case Bottom thus ?thesis using sub_lmm[of "PubConst Bottom" a] by fastforce
next
  case OccursSecType thus ?thesis using sub_lmm[of "PubConst OccursSecType" a] by fastforce
next
  case AbsValue thus ?thesis using sub_lmm[of "PubConst AbsValue" a] by force
qed

lemma assm9: "TComp f T ⊑ Γ t =⇒ arity f > 0"
proof (induction t rule: term.induct)
  case (Var x)
  hence "Γ (Var x) ≠ TAtom Bottom" by force
  hence "∀ t ∈ subterms (fst x). case t of TComp f T ⇒ arity f > 0 ∧ arity f = length T |
  _ => True"
  using Var Γ.v_def simp add: Γ.v_def by meson
  thus ?case using Var by (fastforce simp add: Γ.v_def)
next
  case (Fun g S)
  have "arity g ≠ 0" using Fun.prems Var_subtermeq assm6 by force
  thus ?case using Fun by (cases "TComp f T = TComp g (map Γ S)") auto
qed

lemma assm10: "wf_trm (Γ (Var x))"
proof (auto simp add: Γ.v_def)
lemma assm11: "arity f > 0 =⇒ public f" using public_f_assm by (cases f) auto
lemma assm12: "Γ (Var (τ, n)) = Γ (Var (τ, m))" by (simp add: Γ.v_def)
lemma assm13: "arity c = 0 =⇒ Ana (Fun c T) = ([],[])" by (cases c) simp_all
lemma assm14:
  assumes "Ana (Fun f T) = (K,M)"
  shows "Ana (Fun f T · δ) = (K · list δ, M · list δ)"
proof
  show ?thesis
  proof
    { fix g assume f: "f = Fu g"
      obtain K' M' where "Ana_f g = (K',M')" by moura
      hence ?thesis using assms f True by auto
    } thus ?thesis using True assms by (cases f) auto
next
  case False
  then obtain g where ***: "f = Fu g" using assms by (cases f) auto
  obtain K' M' where ***: "Ana_f g = (K',M')" by moura
  have ***: "K = K' · list (!) T" "M = map (!) T" "arity_f g = length T" "∀ i ∈ fvset (set K') ∪ set M'. i < arity_f g"
  using Ana_Fu_cases(1)[OF assms ***] False Ana_f_assm2_alt[OF ***] by (meson prod.inject)+
  have ***: "∀ i ∈ fvset (set K'). i < length T" "∀ i ∈ set M'. i < length T" using ***(3,4) by auto
  have "K' · list δ = K' · list (!) (map (λ t. t · δ) T)" "M · list δ = map (!) (map (λ t. t · δ) T) M" using subst_idx_map[OF ***(2), of δ]
  by auto
  qed
thus ?thesis using assms * ** ***(3) by auto
qed
qed

sublocale labeled_stateful_typing' arity public Ana Γ Pair label_witness1 label_witness2
apply unfold_locales
subgoal by (metis assms6)
subgoal by (metis assms7)
subgoal by (metis assms9)
subgoal by (rule assms10)
subgoal by (metis assms12)
subgoal by (metis assms13)
subgoal by (metis assms14)
subgoal by (rule label_witness_assm)
subgoal by (rule arity.simps(5))
subgoal by (metis assms14)
subgoal by (metis assms8)
subgoal by (metis assms11)
done

3.3.4 The Protocol Transition System, Defined in Terms of the Reachable Constraints

definition transaction_decl_subst where
"transaction_decl_subst (ξ::('fun,'atom,'sets,'lbl) prot_subst) T ≡
subt_domain ξ = fst ` set (transaction_decl T ()) ∧
(∀(x,cs) ∈ set (transaction_decl T ())). ∃c ∈ cs.
x = Fun (Fu c) [] ∧
arity (Fu c::('fun,'atom,'sets,'lbl) prot_fun) = 0) ∧
wt subst ξ"

definition transaction_fresh_subst where
"transaction_fresh_subst σ T A ≡
subt_domain σ = set (transaction_fresh T) ∧
(∀t ∈ subt_range σ. ∃c. t = Fun c [] ∧ ¬public c ∧ arity c = 0) ∧
(∀t ∈ subt_range σ. t ∉ subterms (trms_transaction T)) ∧
wt subst σ ∧ inj_on σ (subt_domain σ)"

definition transaction_renaming_subst where
"transaction_renaming_subst α P A ≡
∃n ≥ max_var_set (∪(vars_transaction ` set P) ∪ vars lsst A). α = var_rename n"

definition (in intruder_model) constraint_model where
"constraint_model I A ≡
constr_sem_stateful I (unlabel A) ∧
interpretation subst I ∧
wf trms (subst_range I)"

definition (in typed_model) welltyped_constraint_model where
"welltyped_constraint_model I A ≡
w subst I ∧ constraint_model I A"

The set of symbolic constraints reachable in any symbolic run of the protocol P.
ξ instantiates the “declared variables” of transaction T with ground terms. σ instantiates the fresh variables of transaction T with fresh terms. α is a variable-renaming whose range consists of fresh variables.

inductive_set reachable_constraints::
"(('fun,'atom,'sets,'lbl) prot ⇒ ('fun,'atom,'sets,'lbl) prot_constr set" for P:
"('fun,'atom,'sets,'lbl) prot"
where
init[simp]:
"[] ∈ reachable_constraints P"
| step: 
"[A ∈ reachable_constraints P;

44
3.3 Stateful Protocol Model

\[ T \in \text{set } P; \]
\[ \text{transaction\_decl\_subst } \xi; \]
\[ \text{transaction\_fresh\_subst } \sigma; \]
\[ \text{transaction\_renaming\_subst } \alpha; \]
\[ \implies \text{A\_\text{dual\_lssl}} (\text{transaction\_strand } T \cdot \text{lsst } \xi \circ \sigma \circ \alpha) \in \text{reachable\_constraints } P^* \]

3.3.5 Minor Lemmata

**Lemma** \( \Gamma_v \_TAtom [\text{simp}]: \) \( \Gamma_v (TAtom a, n) = TAtom a \)

unfolding \( \Gamma_v \_def \) by simp

**Lemma** \( \Gamma_v \_TAtom': \)

assumes \( "a \neq \text{Bottom}" \)

shows \( \Gamma_v (\tau, n) = TAtom a \iff \tau = TAtom a \)

proof

assume \( \Gamma_v (\tau, n) = TAtom a \)

thus \( \tau = TAtom a \) by (metis (no_types, lifting) assms \( \Gamma_v \_def \) \( \text{fst\_conv} \) \( \text{term\_inject} (1) \))

qed simp

**Lemma** \( \Gamma_v \_TAtom\_inv: \)

\( \Gamma_v x = TAtom (Atom a) \iff \exists m. x = (TAtom (Atom a), m) \)

\( \Gamma_v x = TAtom Value \iff \exists m. x = (TAtom Value, m) \)

\( \Gamma_v x = TAtom SetType \iff \exists m. x = (TAtom SetType, m) \)

\( \Gamma_v x = TAtom AttackType \iff \exists m. x = (TAtom AttackType, m) \)

\( \Gamma_v x = TAtom OccursSecType \iff \exists m. x = (TAtom OccursSecType, m) \)

by (metis \( \Gamma_v \_TAtom\_prim\_surj\_pair \) \( \text{prot\_atom\_distinct} (7) \),

metis \( \Gamma_v \_TAtom\_prim\_surj\_pair \) \( \text{prot\_atom\_distinct} (18) \),

metis \( \Gamma_v \_TAtom\_prim\_surj\_pair \) \( \text{prot\_atom\_distinct} (26) \),

metis \( \Gamma_v \_TAtom\_prim\_surj\_pair \) \( \text{prot\_atom\_distinct} (32) \),

metis \( \Gamma_v \_TAtom\_prim\_surj\_pair \) \( \text{prot\_atom\_distinct} (38) \))

**Lemma** \( \Gamma_v \_TAtom\_''': \)

\( (\text{fst } x = TAtom (Atom a)) = (\Gamma_v x = TAtom (Atom a)) \) (is "\?A = \?A'")

\( (\text{fst } x = TAtom Value) = (\Gamma_v x = TAtom Value) \) (is "\?B = \?B'")

\( (\text{fst } x = TAtom SetType) = (\Gamma_v x = TAtom SetType) \) (is "\?C = \?C'")

\( (\text{fst } x = TAtom AttackType) = (\Gamma_v x = TAtom AttackType) \) (is "\?D = \?D'")

\( (\text{fst } x = TAtom OccursSecType) = (\Gamma_v x = TAtom OccursSecType) \) (is "\?E = \?E'")

proof

have 1: "\?A \implies \?A'" "\?B \implies \?B'" "\?C' \implies \?C" "\?D \implies \?D'" "\?E \implies \?E'"

by (metis \( \Gamma_v \_TAtom\_prod\_collapse\) +

have 2: "\?A' \implies \?A" "\?B' \implies \?B" "\?C' \implies \?C" "\?D' \implies \?D" "\?E' \implies \?E"

using \( \Gamma_v \_TAtom\_prim\_inv (1) \) apply fastforce

using \( \Gamma_v \_TAtom\_prim\_inv (2) \) apply fastforce

using \( \Gamma_v \_TAtom\_prim\_inv (3) \) apply fastforce

using \( \Gamma_v \_TAtom\_prim\_inv (4) \) apply fastforce

using \( \Gamma_v \_TAtom\_prim\_inv (5) \) by fastforce

show "\?A = \?A'" "\?B = \?B'" "\?C = \?C'" "\?D = \?D'" "\?E = \?E'"

using 1 2 by metis+

qed

**Lemma** \( \Gamma_v \_Var\_image: \)

\( \Gamma_v \_Var \_X = \Gamma \_Var \_\_X \)

by \text{force}

**Lemma** \( \Gamma \_Fu\_const: \)

assumes "\text{arity } f = 0"

shows "\exists a. \Gamma (\text{Fun} (\text{Fu } g) T) = TAtom (\text{Atom } a)"

proof

have "\text{\Gamma \_Fun \_const\_thesis}\) using \( \text{assms} \ \Gamma \_assm \) by blast

thus \?thesis using \( \text{assms} \) by \text{force}

qed
lemma Fun_Value_type_inv:
  fixes T::"('fun,'atom,'sets,'lbl) prot_term list"
  assumes "T (Fun f T) = TAtom Value"
  shows "(∃ n. f = Val n) ∨ (∃ bs. f = Abs bs) ∨ (∃ n. f = PubConst Value n)"
proof -
  have *: "arity f = 0" by (metis const_type_inv assms)
  show ?thesis using assms proof (cases f)
    case (Fu g)
    hence "arity f = 0" using * by simp
    hence False using Fu Γ _Fu_const[of g T] assms by auto
    thus ?thesis by metis
  next
    case (Set s)
    hence "arity s = 0" using * by simp
    hence False using Set assms by auto
    thus ?thesis by metis
  qed simp_all
qed

lemma Ana_f_keys_not_val_terms:
  assumes "Ana f = (K, T)"
  and "k ∈ set K"
  and "g ∈ funs_term k"
  shows "¬ is_Val g"
  and "¬ is_PubConstValue g"
  and "¬ is_Abs g"
proof -
  { assume "is_Val g"
    then obtain S where *: "Fun (Val n) S ∈ subterms set (set K)"
      using assms(2) funs_term_Fun_subterm[OF assms(3)]
      by (cases g) auto
    hence False using Ana_f_assm1_alt[OF assms(1) *] by simp
  }
  moreover {
    assume "is_PubConstValue g"
    then obtain S where *: "Fun (PubConst Value n) S ∈ subterms set (set K)"
      using assms(2) funs_term_Fun_subterm[OF assms(3)]
      unfolding is_PubConstValue_def by (cases g) auto
    hence False using Ana_f_assm1_alt[OF assms(1) *] by simp
  }
  moreover {
    assume "is_Abs g"
    then obtain S where *: "Fun (Abs a) S ∈ subterms set (set K)"
      using assms(2) funs_term_Fun_subterm[OF assms(3)]
      by (cases g) auto
    hence False using Ana_f_assm1_alt[OF assms(1) *] by simp
  }
  ultimately show "¬ is_Val g" "¬ is_PubConstValue g" "¬ is_Abs g" by metis
qed

lemma Ana_f_keys_not_pairs:
  assumes "Ana f = (K, T)"
  and "k ∈ set K"
  and "g ∈ funs_term k"
  shows "g ≠ Pair"
proof
  assume "g = Pair"
  then obtain S where *: "Fun Pair S ∈ subterms set (set K)"
    using assms(2) funs_term_Fun_subterm[OF assms(3)]
    by (cases g) auto
  show False using Ana_f_assm1_alt[OF assms(1) *] by simp
qed

lemma Ana_Fu_keys_funs_term_subset:
fixes $K::('fun,'atom,'sets,'lbl) prot_term list$
assumes "Ana (Fun (Fu f) S) = (K, T)"
and "Ana_f f = (K', T')"
shows $\bigcup (funs_term ` set K) \subseteq \bigcup (funs_term ` set K') \cup funs_term (Fun (Fu f) S)$

proof -
\begin{itemize}
\item \text{fix $k$ assume $k \in set K$}
\item then obtain $k'$ where $k'$:
\begin{itemize}
\item $k' \in set K''$ "arity $f = length S"'
\item "subterms $k' \subseteq subterms_{set} (set K')"
\end{itemize}
using \text{assms Ana_Fu_elim[OF assms(1) _ assms(2)] by fastforce}
\item have 1: "funs_term $k' \subseteq \bigcup (funs_term ` set K')" using $k'(1)$ by auto
\item have "i < length S" when "i \in fv k'" for i using that Ana_f assm2_alt[OF assms(2), of i] $k'(1,3)$ by auto
\item hence 2: "funs_term (S ! i) \subseteq funs_term (Fun (Fu f) S)" when "i \in fv k'" for i using that by force
\item have "funs_term $k \subseteq \bigcup (funs_term ` set K') \cup funs_term (Fun (Fu f) S)"
using funs_term_subst[of $k' "(!) S"'] $k'(2) 1 2$ by fast
\end{itemize}
thus \text{?thesis by blast}
\end{proof}

lemma \text{Ana_Fu_keys_not_pubval_terms}:
fixes $k::('fun,'atom,'sets,'lbl) prot_term$
assumes "Ana (Fun (Fu f) S) = (K, T)"
and "Ana_f f = (K', T')"
and "$k \in set K"
and "$g \in funs_term (Fun (Fu f) S). \neg is_PubConstValue g"
shows "$g \in funs_term k. \neg is_PubConstValue g"
using assms(3,4) Ana_f_keys_not_val_terms(1,2)[OF assms(2)]
Ana_Fu_keys_funs_term_subset[OF assms(1,2)]
by blast

lemma \text{Ana_Fu_keys_not_abs_terms}:
fixes $k::('fun,'atom,'sets,'lbl) prot_term$
assumes "Ana (Fun (Fu f) S) = (K, T)"
and "Ana_f f = (K', T')"
and "$k \in set K"
and "$g \in funs_term (Fun (Fu f) S). \neg is_Abs g"
shows "$g \in funs_term k. \neg is_Abs g"
using assms(3,4) Ana_f_keys_not_val_terms(3)[OF assms(2)]
Ana_Fu_keys_funs_term_subset[OF assms(1,2)]
by blast

lemma \text{Ana_Fu_keys_not_pairs}:
fixes $k::('fun,'atom,'sets,'lbl) prot_term$
assumes "Ana (Fun (Fu f) S) = (K, T)"
and "Ana_f f = (K', T')"
and "$k \in set K"
and "$g \in funs_term (Fun (Fu f) S). g \neq Pair"
shows "$g \in funs_term k. g \neq Pair"
using assms(3,4) Ana_f_keys_not_pairs[OF assms(2)]
Ana_Fu_keys_funs_term_subset[OF assms(1,2)]
by blast

lemma \text{Ana_Fu_keys_length_eq}:
assumes "length T = length S"
shows "length (fst (Ana (Fun (Fu f) T))) = length (fst (Ana (Fun (Fu f) S)))"
proof (cases "arity f = length T \land arity f > 0")
case True thus ?thesis using assms by (cases "Ana_f f")
next
case False thus ?thesis using assms by force

qed

lemma deduct_occurs_in_ik:
  fixes t::"('fun,'atom,'sets,'lbl) prot_term"
  assumes t: "M ⊢ occurs t"
  and M: "∀ s ∈ subterms, M. OccursFact ∉ \bigcup (funs_term ` set (snd (Ana s)))"
  "∀ s ∈ subterms, M. OccursSec ∉ \bigcup (funs_term ` set (snd (Ana s)))"
  "Fun OccursSec [] ∉ M"
  shows "occurs t ∈ M"
using private_fun_deduct_in_ik'[of M OccursFact "[Fun OccursSec [], t]" OccursSec] t M
by fastforce

lemma constraint_model_prefix:
  assumes "constraint_model I (A@B)"
  shows "constraint_model I A"
by (metis assms strand_sem_append_stateful unlabel_append constraint_model_def)

lemma welltyped_constraint_model_prefix:
  assumes "welltyped_constraint_model I (A@B)"
  shows "welltyped_constraint_model I A"
by (metis assms constraint_model_prefix welltyped_constraint_model_def)

lemma welltyped_constraint_model_deduct_append:
  assumes "welltyped_constraint_model I A"
  and "ik lsvt A ∙ set I ⊢ s ∙ I"
  shows "welltyped_constraint_model I (A@[l,send⟨s⟩])"
using assms strand_sem_append_stateful[of "{}" "{}" "unlabel A" _ I]
unfolding welltyped_constraint_model_def constraint_model_def
by simp

lemma welltyped_constraint_model_deduct_split:
  assumes "welltyped_constraint_model I (A@[l,send⟨s⟩])"
  shows "welltyped_constraint_model I A ∧ ik lsvt A ∙ set I ⊢ s ∙ I"
using assms strand_sem_append_stateful[of "{}" "{}" "unlabel A" _ I]
unfolding welltyped_constraint_model_def constraint_model_def by simp_all

lemma welltyped_constraint_model_deduct_iff:
  "welltyped_constraint_model I (A@[l,send⟨s⟩]) \iff welltyped_constraint_model I A ∧ ik lsvt A ∙ set I ⊢ s ∙ I"
by (metis welltyped_constraint_model_deduct_append welltyped_constraint_model_deduct_split)

lemma constraint_model_Val_is_Value_term:
  assumes "welltyped_constraint_model I A"
  and "t · I = Fun (Val n) []"
  shows "t = Fun (Val n) [] ∨ ∃ m. t = Var (TAtom Value, m)"
proof -
  have "wt subst I" using assms(1) unfolding welltyped_constraint_model_def by simp
  moreover have "∀ Γ (Fun (Val n) []) = TAtom Value" by auto
  ultimately have "Γ t = TAtom Value" by (metis (no_types) assms(2) wt_subset_trm')
  show ?thesis
  proof (cases t)
    case (Var x)
    obtain τ m where x: "x = (τ, m)" by (metis surj_pair)
    have "Γ x = TAtom Value" using * Var by auto
    hence "τ = TAtom Value" using x Γ TAtom[of Value τ m] by simp
    thus ?thesis using x Var by metis
  next
    case (Fun f T)
    thus ?thesis using assms(2) by auto
  qed
  qed
lemma wellformed_transaction_sem_receives:
fixes T::"('fun,'atom,'sets,'lbl) prot_transaction"
assumes T_valid: "wellformed_transaction T" and I: "strand_sem_stateful IK DB (unlabel (dual lset (transaction_strand T lset φ))) I" and s: "receive ts ∈ set (unlabel (transaction_receive T lset φ))" shows "∀ t ∈ set ts. IK ⊢ t · I" proof -
let ?R = "unlabel (dual lset (transaction_receive T lset φ))" let ?S = "λA. unlabel (dual lset (A lset φ))" let ?R' = "?S (transaction_receive T)"

obtain l B s where B:
"(l,send (ts)) = dual lset ((l,s) lset φ)"
"prefix ((l,s) lset φ)@(l,s) lset φ) (transaction_receive T lset φ)"
using s dual lset_unlabel_steps_iff[of ts "transaction_receive T lset φ"]
dual lset_in_set_prefix_obtain_subst[of "send ts" "transaction_receive T lset φ"]
by blast

have 1: "unlabel (dual lset ((l,s) lset φ))@[(l,s) lset φ]) = unlabel (dual lset (l lset φ))@[send ts]"
using B(1) unlabel_append dual lset_subst dual lset_subst singleton_list_last_proj(4)
dual lset_subst_enoc subset_last_append subset_lst_singleton
by (metis (no_types, lifting) subst_apply_labeled_stateful_strand_step.simps)

have "strand_sem_stateful IK DB ?S' I" using I strand_sem_append_stateful[of IK DB _ _ I] transaction_dual_subst_unfold[of T φ]
by fastforce

hence "strand_sem_stateful IK DB (unlabel (dual lset (B lset φ))@[send ts]) I" using B(0) unfolding prefix_def unlabel_def
by (metis dual lset_def map_append strand_sem_append_stateful)

hence t_deduct: "∀ t ∈ set ts. IK ∪ (ik lset (dual lset (B lset φ)) lset φ) ⊢ t · I" using strand_sem_append_stateful[of IK DB "unlabel (dual lset (B lset φ))" "(send ts)" I]
by simp

have "∀ s ∈ set (unlabel (transaction_receive T))). ∃ t. s = receive t) using T_valid wellformed_transaction_unlabel_cases(1)[OF T_valid] by auto
moreover { fix A::="(l lset) prot_strand" and φ assume "∀ s ∈ set (unlabel A). ∃ t. s = receive t)"
hence "∀ s ∈ set (unlabel (A lset φ)). ∀ t. s = receive t)"
by (simp add: list.pred_set is_Receive_def)

proof (induction A)
case (Cons a A) thus ?case using subst_lst_cons[of a A φ] by (cases a) auto
qed simp

hence "∀ s ∈ set (unlabel (A lset φ)). ∀ t. s = receive t)"
by (simp add: list.pred_set is_Receive_def)

hence "∀ s ∈ set (dual lset (A lset φ))). ∀ t. s = send t)"
by (metis dual lset_memberD dual lset_inv(2) unlabel_in unlabel_mem_has_label)

ultimately have "∀ s ∈ set ?R. ∃ t. s = send (ts)" by simp
hence "ik lset ?R = {}" unfolding unlabel_def ik lset_def by fast
hence "ik lset (dual lset (B lset φ)) = {}"
using B(2) 1 ik lset_append dual lset_append
by (metis (no_types, lifting) Un_empty map_append prefix_def unlabel_def)
thus ?thesis using t_deduct by simp
qed

lemma wellformed_transaction_sem_pos_checks:
assumes T_valid: "wellformed_transaction T" and I: "strand_sem_stateful IK DB (unlabel (dual lset (transaction_strand T lset φ))) I" and "(ac: t ∈ u) ∈ set (unlabel (transaction_checks T lset φ))" shows "(t · I, u · I) ∈ DB"
proof -
let ?S = "(ac: t ∈ u)" let ?R = "transaction_receive T#transaction_checks T" let ?R' = "unlabel (dual lset (?R lset φ))"

3.3 Stateful Protocol Model
3 Stateful Protocol Verification

let ?S = "\AA. unlabel (\cdot (\cdot t)\)"

let ?S' = "?S (transaction_receive T)\)

let ?P = "\\lam a. is_Receive a \lor is_Check_or_Assignment a"

let ?Q = "\lam a. is_Send a \lor is_Check_or_Assignment a"

have s: "?s \in set (unlabel (?R (\cdot t)))"
  using assms(3) subset_last_append[of "transaction_receive T"]
  unlabel_append[of "transaction_receive T"]
  by auto

obtain I B s where B:
  "(l,?s) = dual_lstp ((l,?) t)"
  "prefix ((B (\cdot t))@[((l,?) t)]) (?R (\cdot t))"
  using B(1) unlabel_append dual_lstp_subst dual_lstp_subst singleton_lst_proj(4)
  dual_lstp_subst_enoc subset_last_append subset_lstst_singleton
  by (metis (no_types, lifting) subst_apply_labeled_stateful_strand_step.simps )

have 1: "unlabel (dual_lstp ((B (\cdot t))@[((l,?) t)]) ) = unlabel (dual_lstp (B (\cdot t)))@[?s]"
  using B(1) unlabel_append dual_lstp_subst dual_lstp_subst singleton_lst_proj(4)
  dual_lstp_subst_enoc subset_last_append subset_lstst_singleton
  by (metis (no_types) map_append)

have "\forall a \in set (unlabel (dual_lstp (B (\cdot t)))). ?Q a" proof
  fix a assume a: "a \in set (unlabel (dual_lstp (B (\cdot t))))"
  have "\forall a " when a: "a \in set (unlabel ?R)" for a
    using a wellformed_transaction_unlabel_cases(1,2)[OF T_valid]
    unfolding unlabel_def by fastforce
  hence "\forall a " when a: "a \in set (unlabel (?R (\cdot t)))" for a
    using a stateful_strand_step_cases_subst(2,11)[of \_ \_] subset_lstst_unlabel[of ?R \_]
    unfolding subst_apply_stateful_strand_def by auto
  hence B_P: "\forall a \in set (unlabel (B (\cdot t))). ?P a"
    using unlabel_mono[OF set_mono_prefix[OF append_prefixD[OF B(2)]]]
    by blast

obtain 1 where "(1,a) \in set (dual_lstp (B (\cdot t)))"
  using a by (meson unlabel_mem_has_label)
then obtain b where b: "(1,b) \in set (B (\cdot t))" dual_lstp_subst(1,b) = (1,a)
  by blast

hence "\forall b " using B_P unfolding unlabel_def by fastforce
thus "\forall a " using dual_lstp_inv[OF B(2)] by (cases b) auto
qed

hence "\forall a \in set (unlabel (dual_lstp (B (\cdot t)))). \\neg is_Insert a \land \neg is_Delete a" by fastforce
thus thesis using dpudp_lst_no_upd[of "unlabel (dual_lstp (B (\cdot t)))"] I DB in_db by simp

lemma wellformed_transaction_sem_negChecks:
  assumes T_valid: "wellformed_transaction T"
  and I: "\lam s. strand_sem_stateful IK DB (unlabel (dual_lstp (transaction_strand T (\cdot t)))) I"
  and NegChecks X \ I "s (t,u) \in set (unlabel (transaction_checks T (\cdot t)))"
  shows "\lam \delta. subst_domain \delta = set X \land (subst_range \delta) \rightarrow (t \cdot \delta \cdot I, u \cdot \delta \cdot I) \notin DB" (is ?A)
and "X = [] \implies (t \cdot I, u \cdot I) \notin DB" (is "\neg B \implies \neg B")

proof -

let ?s = "NegChecks X [] [(t,u)]"
let ?R = "transaction_receive T\&\&transaction_checks T"
let ?R' = "unlabel (dual_{set} (?R \cdot \text{last} \cdot \emptyset))"
let ?S = "A. unlabel (dual_{set} (\text{A} \cdot \text{last} \cdot \emptyset))"
let ?S' = "?S (transaction_receive T)\&\&\&?S (transaction_checks T)"
let ?P = "\forall a. is_Receive a \lor is_Check_or_Assignment a"
let ?Q = "\lambda a. is_Send a \lor is_Check_or_Assignment a"
let ?U = "\lambda \delta. \text{sub-domain} \delta = \text{set} X \land \text{ground} (\text{sub-domain-range})"

have "s: \"?s \in \text{set} (\text{unlabel} (?R \cdot \text{last} \cdot \emptyset))"
  using assms(3) sub_last_append[of "transaction_receive T"]
unlabel_append[of "transaction_receive T"]
by auto

obtain I B s where B:
  "(1,?s) = dual_{laststep} ((1,s) \cdot \text{laststep} \cdot \emptyset)"
  "prefix ((B \cdot \text{last} \cdot \emptyset)@[[(1,s) \cdot \text{laststep} \cdot \emptyset]]) (?R \cdot \text{last} \cdot \emptyset)"
using s dual_{set}_unlabel_steps_iff(7)[of X "[]" "[(t,u)]"]
dual_{set}_in_set_prefix Obtain subst[of ?s ?R ?]\by blast

have 1: "unlabel (dual_{set} ((B \cdot \text{last} \cdot \emptyset)@[[(1,s) \cdot \text{laststep} \cdot \emptyset]]) = unlabel (dual_{set} (B \cdot \text{last} \cdot \emptyset))@[?s]"
  unfolding subst_lsst_unlabel[of ?R]
by (metis (no_types) map_append)

have "negchecks_model I (dbdup_{set} (unlabel (dual_{set} (B \cdot \text{last} \cdot \emptyset))) I DB) X [] [(t,u)]"
  using strand_sem_append_stateful[of IK DB _ _ I] transaction_dual_subst_unfold[of T \emptyset]
by fastforce

have "negchecks_model I (dbdup_{set} (unlabel (dual_{set} (B \cdot \text{last} \cdot \emptyset))) I DB) X [] [(t,u)]"
  using strand_sem_append_stateful[of IK DB _ _ I] transaction_dual_subst_unfold[of T \emptyset]
by fastforce

have "\forall a \in \text{set} (\text{unlabel} (dual_{set} (B \cdot \text{last} \cdot \emptyset))). ?Q a"
  proof
    fix a assume a: "a \in \text{set} (\text{unlabel} (dual_{set} (B \cdot \text{last} \cdot \emptyset)))"
    have "\forall ?P a" when a: "a \in \text{set} (\text{unlabel} (?R))" for a
      using a wellformed_transaction_unlabel_cases(1,2,3)[OF \_valid]
      unfolding unlabel_def by fastforce
    hence "\forall ?P a" when a: "a \in \text{set} (\text{unlabel} (?R \cdot \text{last} \cdot \emptyset))" for a
      using a stateful_strand_step_cases_subst(2,11)[of _] subst_lsst_unlabel[of ?R \emptyset]
      unfolding subst_apply_stateful_strand_def by auto
  hence B_P: "\forall a \in \text{set} (\text{unlabel} (B \cdot \text{last} \cdot \emptyset)). ?P a"
    using unlabel_mono[OF set_mono_prefix[OF append_prefixD[OF \_ valid(2)]]]
  by blast

  obtain 1 where "(1,a) \in \text{set} (dual_{set} (B \cdot \text{last} \cdot \emptyset))"
    using a by (meson unlabel_mem_has_label)
  then obtain b where b: "(1,b) \in \text{set} (B \cdot \text{last} \cdot \emptyset)" "dual_{setstep} (1,b) = (1,a)"
    using dual_{set}_memberD by blast
  hence "\forall ?P b" using B_P unfolding unlabel_def by fastforce
  thus "\forall ?Q a" using dual_{setstep}_inv[OF \_ valid(2)] by (cases b) auto
qed
lemma dual_transaction_ik_is_transaction_send':
  fixes T' ::"('fun, 'atom, 'sets, 'lbl) prot_subst"
  assumes "wellformed_transaction T"
  shows "((ik clam (transaction_strand T) "iq set δ) "iq set T) "iq set T' a =
         (trms clam (transaction_strand send T) "iq set δ) "iq set T' a" (is "?A = ?B")
using dual_transaction_ik_is_transaction_send[OF assms]
proof -
  define f where "f ≡ λM'. M ⊃ UNION (subterms M') UNION ((set o fst o Ana) M')"
  define S where "S ≡ (a · δ) a. a ∈ N ∧ wtsubst δ ∧ wftrms (subst_range δ)"
  note 0 = Value_vars_only
have "t ∈ S" when "t ∈ SMP M" for t
using that
proof (induction t rule: SMP.induct)
  case (MP t)
  hence "t ∈ N" "wtsubst Var" "wftrms (subst_range Var)" using N_supset by auto
  hence "t · Var ∈ S" unfolding S_def by blast
  thus ?case by simp
next
case (Subterm t t')
  then obtain δ a where: "a · δ = t" "a ∈ N" "wtsubst δ" "wftrms (subst_range δ)"
  by (auto simp add: S_def)
  hence "∀x ∈ fv a. ∃τ. Γ (Var x) = TAtom τ" using 0 by auto
  hence #: "∀x ∈ fv a. (∃f. δ x = Fun f []) ∨ (∃y. δ x = Var y)"
              using a(3) TAtom_term_cases[OF wt_trm_subst_rangeD[OF a(4)]]
    by (metis wtsubst_def)
  obtain b where: "b · δ = t'' "b ∈ subterms a"
    using subterms_subst_subterm[OF #, of t'] Subterm.hyps(2) a(1)
    by fast
  hence "b ∈ N" using N_supset a(2) by blast
  thus ?case using a b(1) unfolding S_def by blast
next
case (Substitution t δ)
  then obtain δ a where: "a · δ = t" "a ∈ N" "wtsubst δ" "wftrms (subst_range δ)"
  by (auto simp add: S_def)
  have "wtsubst (δ o a)" "wftrms (subst_range (δ o a))"
    by (fact wt_subst_compose[OF a(3) Substitution.hyps(2)],
        fact wt_trms_subst_compose[OF a(4) Substitution.hyps(3)])
3.3 Stateful Protocol Model

moreover have \( t \cdot \vartheta = a \cdot \delta \circ_s \vartheta \) using a(1) subst_subst_compose[of a \( \delta \vartheta \)] by simp

ultimately show ?case using a(2) unfolding S_def by blast

next

case (Ana t K T k)

then obtain \( \delta a \) where \( a : "a \cdot \vartheta = t" \) "a \( \in \mathbb{N} " \)

by (auto simp add: S_def)

obtain Ka Ta where a': "Ana a = (Ka,Ta)" by moura

have #: "K = Ka \\

proof (cases a)

case (Var x)

then obtain g U where gU: "t = Fun g U"

using a(1) Ana.hyps(2,3) Ana_var

by (cases t) simp_all

hence "T (Var x) = TAtom Value" using Var a(2) 0 by auto

using a(1,3) Var gU wt_subst_trm''[OF a(3), of a]

by argo

thus ?thesis using gU Fun_Value_type_inv Ana.hyps(2,3) by fastforce

next

case (Fun g U)

thus ?thesis using a(1) a' Ana.hyps(2) Ana_subst'[of g U] by simp

qed

then obtain ka where ka: "k = ka \\

hence "ka \( \in \mathbb{N} " Using ka(2) a' by simp

hence "ka \( \in \mathbb{N} " Using ka(2) N_supset by auto

thus ?case using ka a(3,4) unfolding S_def by blast

qed

thus ?thesis unfolding S_def by blast

qed

3.3.6 Admissible Transactions

definition admissible_transaction_checks where

"admissible_transaction_checks T \( \equiv \)

\( \forall x \in \text{set} (\text{unlabel} (\text{transaction_checks} T)). \)

\( \text{is_InSet } x \longrightarrow \)

is_Var (the_elem_term x) \wedge is_Fun_Set (the_set_term x) \wedge

\( \text{is_Var } (\text{the_elem_term } x) = TAtom Value \wedge \)

\( \text{is_NegChecks } x \longrightarrow \)

bvars_sstp x = \[] \\

((\text{the_eqs } x = \[] \wedge \text{length } (\text{the_ins } x) = 1) \lor \\

(\text{the_eqs } x = \[] \wedge \text{length } (\text{the_eqs } x) = 1)) \\

\( \text{is_NegChecks } x \wedge \text{the_eqs } x = \[] \longrightarrow (\text{let } h = \text{hd } (\text{the_ins } x) \text{ in} \\

is_Var (\text{fst } h) \wedge is_Fun_Set (\text{snd } h) \wedge \\

\text{fst } (\text{the_Var } (\text{fst } h)) = TAtom Value))";

definition admissible_transaction_updates where

"admissible_transaction_updates T \( \equiv \)

\( \forall x \in \text{set} (\text{unlabel} (\text{transaction_updates} T)). \)

\( \text{is_Update } x \wedge \text{is_Var } (\text{the_elem_term } x) \wedge \\

\text{is_Fun_Set } (\text{the_set_term } x) \wedge \\

\text{is_Var } (\text{the_elem_term } x) = TAtom Value";

definition admissible_transaction_terms where

"admissible_transaction_terms T \( \equiv \)

\( \text{wf}_{trms} \cdot \text{arity } (\text{trms_transaction } T)) \wedge \\

(\forall f \in \text{fun_term } \cdot \text{trms_transaction } T). \\

\neg is_Val f \wedge \neg is_Abs f \wedge \neg is_PubConst f \wedge f \neq \text{Pair} \wedge \\

(\forall r \in \text{set} (\text{unlabel} (\text{transaction_strand } T)). \\

(\exists f \in \text{fun_term } \cdot (\text{trms} r)). \text{is_Attack } f \longrightarrow \\

\text{is_Send } r \wedge \text{length } (\text{the_msgs } r) = 1 \wedge \text{is_Fun_Attack } (\text{hd } (\text{the_msgs } r)))";

definition admissible_transaction_occurs_checks where

"admissible_transaction_occurs_checks T \( \equiv \) ( \\

let occ_in = \lambda x S. occurs (Var x) \in \text{set} (\text{the_msgs } (\text{hd } (\text{unlabel } S))));
\[ \text{rcvs} = \text{transaction\_receive} \ T; \]
\[ \text{snds} = \text{transaction\_send} \ T; \]
\[ \text{frsh} = \text{transaction\_fresh} \ T; \]
\[ \text{fvs} = \text{fv\_transaction} \ T \]
\[ \text{in} \ (\exists x \in \text{fvs} - \text{set\_frsh. fst} \ x = \text{TAtom Value}) \rightarrow (\text{rcvs} \neq [] \land \text{is\_Receive} (\text{hd} (\text{unlabel} \ \text{rcvs})) \land (\forall x \in \text{fvs} - \text{set\_frsh. fst} \ x = \text{TAtom Value} \rightarrow \text{occ\_in x rcvs})) \land (\text{frsh} \neq [] \land \text{is\_Send} (\text{hd} (\text{unlabel} \ \text{snds})) \land (\forall x \in \text{set\_frsh. occ\_in x snds})) \land (\forall t \in \text{trms\_lsst\_snds. OccursFact} \in \text{funs\_term} \ t \lor \text{OccursSec} \in \text{funs\_term} \ t) \rightarrow (\exists x \in \text{set (transaction\_fresh T)}. t = \text{occurs (Var x)}) \]

\[ \text{definition admissible\_transaction where} \]
\[ \text{admissible\_transaction} \ T \equiv (\text{wellformed\_transaction} \ T \land \text{transaction\_decl} \ () = [] \land \text{list\_all} (\lambda x. \text{fst} \ x = \text{TAtom Value}) \land (\forall x \in \text{vars\_transaction} \ T. \text{is\_Var} (\text{fst} \ x) \land (\text{the\_Var (fst x)} = \text{Value})) \land \text{bvars\_lsst} (\text{transaction\_strand} \ T) = {} \land \text{set (transaction\_fresh T)} \subseteq \text{fv\_lsst (filter (is\_Insert o snd) (transaction\_updates T))} \cup \text{fv\_lsst (transaction\_send T)} \land (\forall x \in \text{fv\_transaction} \ T - \text{set (transaction\_fresh T)}. \forall y \in \text{fv\_transaction} \ T - \text{set (transaction\_fresh T)}. x \neq y \rightarrow (\text{Var x} \neq \text{Var y}) \in \text{set (unlabel (transaction\_checks T))} \lor (\text{Var y} \neq \text{Var x}) \in \text{set (unlabel (transaction\_checks T))}) \land \text{fv\_lsst (transaction\_updates T)} \cup \text{fv\_lsst (transaction\_send T) - set (transaction\_fresh T)} \subseteq \text{fv\_lsst (transaction\_receive T)} \cup \text{fv\_lsst (transaction\_checks T)} \land \text{admissible\_transaction\_terms} \ T \land \text{admissible\_transaction\_occurs\_checks} \ T \]

\[ \text{lemma admissible\_transactionE:} \]
\[ \text{assumes T: "admissible\_transaction} \ T \text{"} \]
\[ \text{shows "transaction\_decl} \ () = [] \" (is \ ?A) \]
\[ \text{and "\forall x \in \text{set (transaction\_fresh T). } \Gamma_v \ x = \text{TAtom Value}" (is \ ?B) \]
\[ \text{and "\forall x \in \text{vars\_lsst} (transaction\_strand T). } \Gamma_v \ x = \text{TAtom Value}" (is \ ?C) \]
\[ \text{and "\text{bvars\_lsst} (transaction\_strand T) = {}" (is \ ?D1) \]
\[ \text{and "\text{fv\_lsst (transaction\_updates T)} \subseteq \text{fv\_lsst (transaction\_receive T)} \lor \text{fv\_lsst (transaction\_checks T)} \land \text{admissible\_transaction\_terms} \ T \land \text{admissible\_transaction\_occurs\_checks} \ T \"
\]

\[ \text{is } ? \text{E} \]
\[ \text{and "\text{set (transaction\_fresh T) } \subseteq \text{fv\_lsst (transaction\_updates T)} \lor \text{fv\_lsst (transaction\_send T)}" (is } ? \text{F) \]
\[ \text{and "\forall x \in \text{fv\_transaction} \ T - \text{set (transaction\_fresh T)}. } \forall y \in \text{fv\_transaction} \ T - \text{set (transaction\_fresh T)}. x \neq y \rightarrow (\text{Var x} \neq \text{Var y}) \in \text{set (unlabel (transaction\_checks T))} \lor (\text{Var y} \neq \text{Var x}) \in \text{set (unlabel (transaction\_checks T))}" (is } ? \text{G) \]
\[ \text{and "\forall x \in \text{fv\_lsst (transaction\_checks T)}. } x \in \text{fv\_lsst (transaction\_receive T)} \lor (\exists t. s. \text{select}(t, s) \in \text{set (unlabel (transaction\_checks T))} \land x \in \text{fv t} \lor \text{fv s})" (is } ? \text{H) \]
\[ \text{and "fv\_lsst (transaction\_updates T) } \cup \text{fv\_lsst (transaction\_send T) - set (transaction\_fresh T) } \subseteq \]
3.3 Stateful Protocol Model

\[ \text{fv}_{\text{sst}} (\text{transaction\_receive } T) \cup \text{fv}_{\text{sst}} (\text{transaction\_checks } T) \]
(is \(\text{?I}\))

and "\(x \in \text{set } (\text{unlabel } (\text{transaction\_checks } T))\).
\[\text{is\_Equality } x \rightarrow \text{fv } (\text{the\_rhs } x) \subseteq \text{fv}_{\text{sst}} (\text{transaction\_receive } T)\]
(is \(\text{?J}\))

and "set (\text{transaction\_fresh } T) \cap \text{fv}_{\text{sst}} (\text{transaction\_receive } T) = \{\}\) (is \(\text{?K1}\))

and "set (\text{transaction\_fresh } T) \cap \text{fv}_{\text{sst}} (\text{transaction\_checks } T) = \{\}\) (is \(\text{?K2}\))

and "\(\lambda x. \text{fst } x = \text{Var } \text{Value}\) (\text{transaction\_fresh } T)" (is \(\text{?K3}\))

and "\(\forall x \in \text{vars\_transaction } T. \neg \text{TAtom\_AttackType } \subseteq \Gamma_v x\)" (is \(\text{?K4}\))

proof -
using \(T\) unfolding \text{admissible\_transaction\_def}
by (blast, blast, blast, blast, blast, blast, blast)

have "\(\lambda x. \text{fst } x = \text{Var } \text{Value}\) (\text{transaction\_fresh } T)"
\(\forall x \in \text{vars\_transaction } T. \text{is\_Var } (\text{fst } x) \land \text{the\_Var } (\text{fst } x) = \text{Value}\)
using \(T\) unfolding \text{admissible\_transaction\_def} by (blast, blast)

thus \(\text{?E}\) using \(\Gamma_v \neg \text{TAtom')'(2}\) unfolding list_all_iff by (blast, force, force)

show \(\text{?B}\) using \(T\) unfolding \text{admissible\_transaction\_def} by argo
thus \(\text{?F}\) unfolding \text{unlabel\_def} by auto

let \(?\text{selects} = \text{"filter } (\lambda s. \text{is\_InSet } (\text{snd } s) \land \text{the\_check } (\text{snd } s) = \text{Assign}) (\text{transaction\_checks } T)"

show \(\text{?H}\)
proof
fix \(x\) assume "\(x \in \text{fv}_{\text{sst}} (\text{transaction\_checks } T)\)"

hence "\(x \in \text{fv}_{\text{sst}} (\text{transaction\_receive } T) \lor x \in \text{fv}_{\text{sst}} \text{?selects}\)"
using \(T\) unfolding \text{admissible\_transaction\_def} by blast

thus "\(x \in \text{fv}_{\text{sst}} \text{?selects}\)"
\(\exists t s. \text{select}(t,s) \in \text{set } (\text{unlabel } (\text{transaction\_checks } T)) \land x \in \text{fv } t \cup \text{fv } s)"

proof
assume "\(x \in \text{fv}_{\text{sst}} \text{?selects}\)"
then obtain \(r\) where \(\"x \in \text{fv}_{\text{sst}} r\)" "\(r \in \text{set } (\text{unlabel } (\text{transaction\_checks } T))\)"
"\text{is\_InSet } r = \text{Assign}\"

unfolding \text{unlabel\_def} by force
thus \(\text{?thesis}\) by (cases \(r\)) auto
qed simp
qed

lemma \text{admissible\_transaction\_is\_wellformed\_transaction:}
assumes "\text{admissible\_transaction } T"
shows "\text{wellformed\_transaction } T"
and "\text{admissible\_transaction\_checks } T"
and "\text{admissible\_transaction\_updates } T"
and "\text{admissible\_transaction\_terms } T"
and "\text{admissible\_transaction\_occurs\_checks } T"
using \(\text{assms}\) unfolding \text{admissible\_transaction\_def} by blast+

lemma \text{admissible\_transaction\_fresh\_vars\_notin:}
assumes \(T\): "\text{admissible\_transaction } T"
and \(x\): "\(x \in \text{set } (\text{transaction\_fresh } T)\)"
shows "\(x \notin \text{fv}_{\text{sst}} (\text{transaction\_receive } T)\)" (is \(\text{?A}\))
and "\(x \notin \text{fv}_{\text{sst}} (\text{transaction\_checks } T)\)" (is \(\text{?B}\))
and "\(x \notin \text{vars}_{\text{sst}} (\text{transaction\_receive } T)\)" (is \(\text{?C}\))
and "\(x \notin \text{vars}_{\text{sst}} (\text{transaction\_checks } T)\)" (is \(\text{?D}\))
and "\(x \notin \text{bvars}_{\text{sst}} (\text{transaction\_receive } T)\)" (is \(\text{?E}\))
and "\(x \notin \text{bvars}_{\text{sst}} (\text{transaction\_checks } T)\)" (is \(\text{?F}\))

proof -
3 Stateful Protocol Verification

have 0:
"set (transaction_fresh T) ⊆ fv_{t,s} (transaction_updates T) ∪ fv_{t,s} (transaction_send T)"
"set (transaction_fresh T) ∩ fv_{t,s} (transaction_receive T) = {}"
"set (transaction_fresh T) ∩ fv_{t,s} (transaction_checks T) = {}"
"fv_{transaction T} ∩ bvars_{transaction T} = {}"
using admissible_transactionE[OF T] by argo+

have 1: "set (transaction_fresh T) ∩ bvars_{transaction T} (transaction_checks T) = {}"
using 0(1,4) fv_{transaction unfold}[of T] bvars_{transaction unfold}[of T] by blast

have 2:
"vars_{transaction T} = fv_{transaction T}
"bvars_{transaction T} = {}"
using bvars_wellformed_transaction_unfold[OF T_wf] vars_{transaction T} is fv_{transaction T} bvars_{transaction T}[of "unlabel (transaction_receive T)"]
by blast+


show ?D using 0(3) 1 x vars_{transaction T} is fv_{transaction T} bvars_{transaction T}[of "unlabel (transaction_checks T)"
by fast

lemma admissible_transaction_fv_in_receives_or_selects:
assumes T: "admissible_transaction T"
and x: "x ∈ fv_{transaction T} "x ∉ set (transaction_fresh T)"
shows "x ∈ fv_{t,s} (transaction_receive T) ∨ (x ∈ fv_{t,s} (transaction_checks T) ∨
(∃ t s. select⟨t,s⟩ ∈ set (unlabel (transaction_checks T)) ∧ x ∈ fv t ∪ fv s))"
proof -
have "x ∈ fv_{t,s} (transaction_receive T) ∪ fv_{t,s} (transaction_checks T) ∪
fv_{t,s} (transaction_updates T) ∪ fv_{t,s} (transaction_send T)"
using x(1) fv_{append} unravel_append
by (metis transaction_strand_def append_assoc)
thus ?thesis using x(2) admissible_transactionE(9,10)[OF T] by blast

lemma admissible_transaction_decl_subst_empty':
assumes T: "admissible_transaction T"
and ξ: "transaction_decl_subst ξ T"
shows "ξ = Var"
proof -
have "subst_domain ξ = {}"
using ξ T unfolding transaction_decl_subst_def by auto
thus ?thesis by auto

lemma admissible_transaction_decl_subst_empty:
assumes T: "admissible_transaction T"
and ξ: "transaction_decl_subst ξ T"
shows "ξ = Var"
by (rule admissible_transaction_decl_subst_empty'[OF admissible_transactionE(1)[OF T] ξ))

lemma admissible_transaction_no_bvars:
assumes "admissible_transaction T"
shows "fv_{transaction T} = vars_{transaction T}"
and "bvars_{transaction T} = {}"
using admissible_transactionE(4)[OF assms] bvars_wellformed_transaction_unfold vars_{transaction T} bvars_{transaction T}
by (fast, fast)

lemma admissible_transactions_fv_bvars_disj:
assumes "∀T ∈ set P. admissible_transaction T"
shows "(∪T ∈ set P. fv_transaction T) ∩ (∪T ∈ set P. bvars_transaction T) = {}"
using assms admissible_transaction_no_bvars(2) by fast

lemma admissible_transaction_occurs_fv_types:
assumes "admissible_transaction T"
and "x ∈ vars_transaction T"
shows "x ∈ fv_transaction T"
using assms admissible_transaction_def by blast

proof

"moreover have "is_Var (fst x)" "the_Var (fst x) = Value"
using assms unfolding admissible_transaction_def by blast+
thus ?thesis using Γ∧TAtom'"(2)[of x] by force
qed

lemma admissible_transaction_Value_vars_are_fv:
assumes "admissible_transaction T" and "Γ x = TAtom Value"
shows "x ∈ fv_transaction T"
using assms unfolding admissible_transaction_def by blast+

proof

define θ where θ ≡ "ξ oₐ σ oₐ α"

have t': "send(ts) ∈ set (unlabel (dual:<ts> (transaction_receive T :ts ∩;;;;;)))" by simp
then obtain T1 T2 where T: "unlabel (dual:<ts> (transaction_receive T :ts ∩;;;;;)) = T1@send(ts)#T2" using t' by (meson split_list)

have "constr_sem_stateful (unlabel A@unlabel (dual:<ts> (transaction_strand T :ts ∩;;;;;)))" using I (unlabel_append[of A]) unfolding constraint_model_def θ_def by simp
hence "constr_sem_stateful I (unlabel A@T1@[send(ts)])" using strand_sem_append_stateful[of "{ts}" "{ts}" "unlabel A@T1@[send(ts)]"] Γ I transaction_dual_subst_unlabel_unfold[of T θ] T by (metis append_assoc append_cons append_nil)

hence "∀t ∈ set ts. ikₐA@T1@T ⊢ t . I" using strand_sem_append_stateful[of "{ts}" "{ts}" "unlabel A@T1" "[send(ts)]"] I T by force

moreover have "¬is_Receive x"
when x: "x ∈ set (unlabel (dual:<ts> (transaction_receive T :ts ∩;;;;;)))" for x

proof -

have x: "is_Receive a" when "a ∈ set (unlabel (transaction_receive T))" for a
using T_wf Ball_set[of "unlabel (transaction_receive T)" is_Receive] unfolding wellformed_transaction_def by blast

obtain 1 where 1: "(1,x) ∈ set (dual:<ts> (transaction_receive T :ts ∩;;;;;))" using x unfolding unlabel_def by fastforce

then obtain ly where ly: "ly ∈ set (transaction_receive T :ts ∩;;;;;)" "(1,x) = dual:<step> ly" unfolding dual:<step>_def by auto

obtain j y where j: "ly = (j,y)" by (metis surj_pair)

hence "j = 1" using ly(2) by (cases y) auto

57
3 Stateful Protocol Verification

hence y: "(l,y) ∈ set (transaction_receive T ·ιsst δ)" "(l,x) = dualιsstp (l,y)"
by (metis j ly(1), metis j ly(2))

obtain z where z:
"z ∈ set (unlabel (transaction_receive T))" 
"(l,z) ∈ set (transaction_receive T)"
"(l,y) = (l,z) ·ιsst ϑ"
using y(1) unfolding subst_apply_labeled_stateful_strand_def unlabel_def by force

have "is_Receive y" using *[OF z(1)] z(3) by (cases z) auto
thus "¬is_Receive x" using l y by (cases y) auto
qed 
hence "¬is_Receive x" when "x ∈ set T1" for x using T that by simp

have "iksst T1 = {}" unfolding iksst_def is_Receive_def by fast

ultimately show ?thesis by (simp add: ϑ_def)
qed

lemma transaction_checks_db:
assumes T: "admissible_transaction T"
and I : "constraint_model I (A@dual ·ιsst (transaction_strand T ·ιsst ξ ◦ισ ◦ισ ◦ια))"
and ξ: "transaction_decl_subst ξ T"
and σ: "transaction_fresh_subst σ T A"
and α: "transaction_renaming_subst α P A"
shows "⟨Var (TAtom Value, n) in Fun (Set s) []⟩ ∈ set (unlabel (transaction_checks T))" 
⇒ ⟨α (TAtom Value, n) · I, Fun (Set s) []⟩ ∈ set (db ·ιsst A I)"
(is "?A ⇒ ?B")
and "⟨Var (TAtom Value, n) not in Fun (Set s) []⟩ ∈ set (unlabel (transaction_checks T))" 
⇒ ⟨α (TAtom Value, n) · I, Fun (Set s) []⟩ /∈ set (db ·ιsst A I)"
(is "?C ⇒ ?D")
proof -
let ?x = "λn. (TAtom Value, n)"
let ?s = "Fun (Set s) []"
let ?T = "transaction_receive T@transaction_checks T"
let ?T' = "λS. ?T S S ·ιsst ξ ◦ισ ◦ισ ◦ια"
let ?S = "λS. transaction_receive T@S"
let ?S' = "λS. ?S S ·ιsst ξ ◦ισ ◦ισ ◦ια"

note ξ_empty = admissible_transaction_decl_subst_empty[OF T ξ]

note T_wf = admissible_transaction_is_wellformed_transaction(1)[OF T]

have "constr_sem_stateful I (unlabel (A@dualιsst (transaction_strand T ·ιsst ξ ◦ισ ◦ισ ◦ια))))" 
using I unfolding constraint_model_def by simp
moreover have 
"dualιsst (transaction_strand T ·ιsst δ) = dualιsst (?S (T1@[(l,c)])) ·ιsst δ)@[ιsst δ]" 
dualιsst (T@transaction_updates T@transaction_send T ·ιsst δ)
when "transaction_checks T = T1@T2" for T1 T2 c δ
using that dualιsst_append subst_lsst_append
unfolding transaction_strand_def
by (metis append.assoc append.Cons append.Nil)

ultimately have T'_model: "constr_sem_stateful I (unlabel (A@dualιsst (?S' (T1@[(l,c)])))))" 
when "transaction_checks T = T1@T2" for T1 T2 l c
using strand_sem_append_stateful[of _ _ _ _ T] 
by (simp add: that transaction_strand_def)

show "?A ⇒ ?B"
proof -
assume a: ?A
hence "⟨Var (?x n) in ?s⟩ ∈ set (unlabel ?T)"
unfolding transaction_strand_def unlabel_def by simp
then obtain 1 T1 T2 where T1: "transaction_checks T = T1@l,(Var (?x n) in ?s))#T2"
proof
  have "?x n ∈ fv_{s\alpha} (transaction_checks T)"
  using by force
  hence "?x n ∉ set (transaction_fresh T)"
  using a admisible_transaction_fresh_vars_notin[OF T] by fast
  hence "unlabel (Aドル_{s\alpha} (\{?x n\} in ?s))) = unlabel (Aドル_{s\alpha} (?s' T1)))@\{(\alpha (?x n) in ?s)\)"
  using T T1 dual_singleton_append subst_last_append unlabel_append λ_empty
  by (fastforce simp add: transaction_fresh_subst_def unlabel_def dual_{s\alpha} subst_apply_labeled_stateful_strand_def)
  moreover have "db_{\alpha} (\text{unlabel} A) = db_{\alpha} (\text{unlabel} (Aドル_{s\alpha} (?s' T1)))"
  by (simp add: T1 db_{\alpha} transaction_prefix_eq[OF T_wf del: unlable_append])
  ultimately have "I. strand_sem_stateful N (set (db_{\alpha} (\text{unlabel} A) I)) \{(\alpha (?x n) in ?s)\} I" using T'_model[OF T1] db_{\alpha} set_is_dbu_{\alpha}(of _ _ I) strand_sem_append_stateful[of _ _ _ I]
  by (simp add: db_{\alpha} subst_def del: unlabel_append)
  qed

show "?C \implies ?D"
proof -
  assume a: ?C
  hence *: "\{(\text{Var} (?x n) not in ?s) \in set (unlabel \{T\})\}"
  unfolding transaction_strand_def unlabel_def by simp
  then obtain T T1 T2 where T1: "transaction_checks T = T1@l, Fun (Set s) []" by (metis a split_list unlable_mem_has_label)
  have "?x n ∈ vars_{s\alpha} (\text{Var} (?x n) not in ?s)"
  using vars_{s\alpha} cases[of ""] [where ?x n = ?x n] by auto
  hence "?x n ∈ vars_{s\alpha} (transaction_checks T)"
  using a unfolding vars_{s\alpha}_def by force
  hence "?x n ∉ set (transaction_fresh T)"
  using a admisible_transaction_fresh_vars_notin[OF T] by fast
  hence "unlabel (Aドル_{s\alpha} (\{\alpha (?x n) in ?s)\}))) = unlabel (Aドル_{s\alpha} (?s' T1)))@\{(\alpha (?x n) not in ?s)\}"
  using T T1 dual_singleton_append subst_last_append unlabel_append λ_empty
  by (fastforce simp add: transaction_fresh_subst_def unlabel_def dual_{s\alpha} subst_apply_labeled_stateful_strand_def)
  moreover have "db_{\alpha} (\text{unlabel} A) = db_{\alpha} (\text{unlabel} (Aドル_{s\alpha} (?s' T1)))"
  by (simp add: T1 db_{\alpha} transaction_prefix_eq[OF T_wf del: unlable_append])
  ultimately have "I. strand_sem_stateful N (set (db_{\alpha} (\text{unlabel} A) I)) \{(\alpha (?x n) not in ?s)\} I" using T'_model[OF T1] db_{\alpha} set_is_dbu_{\alpha}(of _ _ _ I) strand_sem_append_stateful[of _ _ _ _ _ _]
  by (simp add: db_{\alpha} subst_def del: unlabel_append)
  thus ?D using stateful_strand_sem_NegChecks_no_bvars[of _ _ _ ?s I] by simp
qed

lemma transaction_selects_db:
  assumes T: "admisible_transaction T"
  and I: "constraint_model I (Aドル_{s\alpha} (transaction_strand T \weis\xi\alpha\sigma\alpha))"
  and λ: "transaction_decl_subst \weis\xi\alpha\sigma\alpha" 
  and σ: "transaction_fresh_subst \weis\xi\alpha\sigma\alpha"
  and α: "transaction_removing_subst \weis\xi\alpha\sigma\alpha P A"
  shows "\{select(Var (TAtom Value, n), Fun (Set s) []) \in set (unlabel (transaction_checks T)) \implies (\alpha (TAtom Value, n) \cdot I, Fun (Set s) []) \in set (db_{\alpha} A I)\}" (is \"?A \implies ?B\"")
proof -
  let ?x = "\lambda n. (TAtom Value, n)"
  let ?s = "Fun (Set s) []"
  let ?T = "transaction_receive T@\text{transaction_checks} T"
  let ?T' = ?T \weis\xi\alpha\sigma\alpha
  let ?S = "\lambda s. \text{transaction_receive} T@S"
  let ?S' = "\lambda s. ?S \weis\xi\alpha\sigma\alpha"

  have "?x n ∈ fv_{s\alpha} (\text{transaction} \weis\xi\alpha\sigma\alpha)"
  using by force
  hence "?x n ∉ set (transaction_fresh T)"
  using a admisible_transaction_fresh_vars_notin[OF T] by fast
3 Stateful Protocol Verification

note \( \xi_{\text{empty}} = \text{admissible}\_\text{transaction}\_\text{decl} \_\text{subst}_{\text{empty}}[\text{OF} \ T \ \xi] \)

note \( T_{\text{wf}} = \text{admissible}\_\text{transaction}\_\text{is}\_\text{wellformed}\_\text{transaction}(1)[\text{OF} \ T] \)

have "\text{constr}\_\text{sem} \_\text{stateful} \ I \ (\text{unlabel} \ (A@ dual_{\text{lsst}} \ (\text{transaction}\_\text{strand} \ T \ _{\text{stateful}} \ \xi_{\text{stateful}} \ o_{\text{stateful}} \ o_{\text{stateful}})))"
using \( I \) unfolding constraint_model_def by simp
moreover have "dual_{\text{lsst}} \ (\text{transaction}\_\text{strand} \ T \ _{\text{stateful}} \ \delta) = dual_{\text{lsst}} \ (\text{transaction}\_\text{updates} \ T \ _{\text{stateful}} \ \delta) @ dual_{\text{lsst}} \ (\text{transaction}\_\text{send} \ T \ _{\text{stateful}} \ \delta)"
when "\text{transaction}\_\text{checks} \ T = T1@c#T2" for \( T1 \ T2 \ c \)
using strand_sem_append_stateful[of _ _ _ _]
by (simp add: that transaction_strand_def)
ultimately have "T'_{\text{model}}: \text{constr}\_\text{sem} \_\text{stateful} \ I \ (\text{unlabel} \ (A@ dual_{\text{lsst}} \ (?S' \ (T1@[(l,c)]))))"
when "\text{transaction}\_\text{checks} \ T = T1@(l,c)#T2" for \( T1 \ T2 \ l \ c \)
by (simp add: that transaction_strand_def)

show "?A \implies \ ?B"
proof -
assume a: ?A
hence *: "\text{select} \langle \text{Var} \ (?x n), ?s \rangle \in \text{set} \ (\text{unlabel} \ ?T)"
unfolding transaction_strand_def unlabel_def by simp
then obtain \( l \ T1 \ T2 \)
where T1: "\text{transaction}\_\text{checks} \ T = T1@(l,\text{select} \langle \text{Var} \ (?x n), ?s \rangle)\#T2"
by (metis a split_list unlabel_mem_has_label)
hence "\( ?x n \in \text{fv} \ \text{lsst} \ (\text{transaction}\_\text{checks} \ T) \)"
using a by force
hence "\( ?x n \notin \text{set} \ (\text{transaction}\_\text{fresh} \ T) \)"
using a admissible_transaction_fresh_vars_notin[of T] by fast
hence "\text{unlabel} \ (A@dual_{\text{lsst}} \ (?S' \ (T1@[(l,c)]))) = unlabel \ (A@dual_{\text{lsst}} \ (?S' \ T1))@[\text{select}(\alpha \ (?x n), ?s)]"
using T a \( \sigma \) dual_{\text{lsst}} append subst_last_append unlabel_append \( \xi_{\text{empty}} \)
by (fastforce simp add: transaction_fresh_subst_def unlabel_def dual_{\text{lsst}}_def subst_apply_labeled_stateful_strand_def)
moreover have "db_{\text{sst}} \ (\text{unlabel} \ A) = db_{\text{sst}} \ (\text{unlabel} \ (A@dual_{\text{lsst}} \ (?S' \ T1)))"
by (simp add: T1 db_{\text{sst}}_transaction_prefix_eq[of T wf] del: unlabel_append)
ultimately have "\exists M. strand_{\text{sem}} \_\text{stateful} \ M \ (\text{set} \ (db_{\text{sst}} \ (\text{unlabel} \ A) \ I)) ![\alpha \ (?x n) \in \ ?s] \ I"
using T'_{\text{model}}[OF T1] db_{\text{sst}}_set_is_dbupd_{\text{sst}}[of _ _ _ _] strand_sem_append_stateful[of _ _ _ _]
by (simp add: db_{\text{sst}}_def del: unlabel_append)
thus \( ?B \) by simp
qed

lemma admissible_transaction_terms_no_Value_consts:
assumes "\text{admissible}\_\text{transaction}\_\text{terms} \ T"
and "\( t \in \text{subsetterms}_{\text{sst}} \ (\text{trms}_{\text{lsst}} \ (\text{transaction}\_\text{strand} \ T)) \)"
shows "\( \neg \exists a \ T. t = \text{Fun} \ (\text{Val} \ a) \ T \) (is \( ?A \))"
and "\( \neg \exists a \ T. t = \text{Fun} \ (\text{Abs} \ a) \ T \) (is \( ?B \))"
and "\( \neg \exists a \ T. t = \text{Fun} \ (\text{PubConst Value} \ a) \ T \) (is \( ?C \))"
proof -
have "\( \neg \text{is}\_\text{Val} \ f \) "\( \neg \text{is}\_\text{Abs} \ f \) "\( \neg \text{is}\_\text{PubConstValue} \ f \)"
when "\( f \in \bigcup \ (\text{funs}\_\text{term} \ (\text{trms}\_\text{transaction} \ T)) \)" for \( f \)
using that assms(1)[unfolded admissible_transaction_terms_def]
unfolding is_PubConstValue_def by (blast,blast,blast)
moreover have "\( \neg \exists a \ T. t = \text{Fun} \ (\text{id}_{\text{term}} \ (\text{trms}\_\text{transaction} \ T)) \)"
when "\( f \in \text{funs}\_\text{term} \ t \)" for \( f \)
using that assms(2)[funs_term_subterms_eq(2)[of "\text{trms}\_\text{transaction} \ T"] by blast+
ultimately have *: "\( \neg \text{is}\_\text{Val} \ f \) "\( \neg \text{is}\_\text{Abs} \ f \) "\( \neg \text{is}\_\text{PubConstValue} \ f \)"
when "\( f \in \text{funs}\_\text{term} \ t \)" for \( f \)
using that by presburger+
show \(?A\) using \(*\)(1) by force
show \(?B\) using \(*\)(2) by force
show \(?C\) using \(*\)(3) unfolding is_PubConstValue_def by force
qed

lemma admissible_transactions_no_Value_consts:
  assumes "admissible_transaction T"
  and "t ∈ subterms set (trms lsst (transaction_strand T))"
  shows "\(∃ a. T. t = Fun (Val a) T\)" (is \(?A\))
  and "\(∃ a. T. t = Fun (Abs a) T\)" (is \(?B\))
  and "\(∃ a. T. t = Fun (PubConst Value a) T\)" (is \(?C\))
using admissible_transactions_terms_no_Value_consts[OF assms(1) assms(2)]
by auto

lemma admissible_transactions_no_Value_consts':
  assumes "admissible_transaction T"
  and "t ∈ trms lsst (transaction_strand T)"
  shows "\(\exists n. Fun (Val n) T \notin subterms t\)"
  and "\(\exists n. Fun (Abs n) T \notin subterms t\)"
using assms
unfolding admissible_transaction_def admissible_transaction_terms_def
by (meson prot_fun.discI(6), meson prot_fun.discI(4))

lemma admissible_transactions_no_Value_consts'':
  assumes "admissible_transaction T"
  shows "\(∀ n. PubConst Value n \notin ⋃ (funs_term ` trms_transaction T)\)"
  and "\(∀ n. Abs n \notin ⋃ (funs_term ` trms_transaction T)\)"
using assms
unfolding admissible_transaction_def admissible_transaction_terms_def
by (meson prot_fun.discI(6), meson prot_fun.discI(4))

lemma admissible_transactions_no_PubConsts:
  assumes "admissible_transaction T"
  and "t ∈ subterms set (trms lsst (transaction_strand T))"
  shows "\(\nexists a n T. t = Fun (PubConst a n) T\)"
proof -
  have "¬ is_PubConst f"
    when "f ∈ ⋃ (funs_term ` (trms_transaction T))" for f
    using that conjunct1[OF conjunct2[OF admissible_transaction_is_wellformed_transaction(4) OF assms(1), unfolded admissible_transaction_terms_def]]
    by blast
  moreover have "f ∈ ⋃ (funs_term ` (trms_transaction T))"
    when "f ∈ funs_term t" for f
    using that assms(2) funs_term_subterms_eq(2)[of "trms_transaction T"] by blast+
  ultimately have *: "¬is_PubConst f"
    when "f ∈ funs_term t" for f
    using that by presburger+
  show \(?thesis\) using * by force
qed

lemma admissible_transactions_no_PubConsts':
  assumes "admissible_transaction T"
  and "t ∈ trms lsst (transaction_strand T)"
  shows "\(∃ a n T. TAtom Value n ∈ subterms T\)"
using admissible_transactions_no_PubConsts[OF assms(1)] assms(2)
by fast+

lemma transaction_inserts_are_Value_vars:
  assumes T_valid: "wellformed_transaction T"
  and "admissible_transaction_updates T"
  and "insert(t,s) ∈ set (unlabel (transaction_strand T))"
  shows "\(\exists n. t = Var (TAtom Value, n)\)"
  and "\(\exists u. s = Fun (Set u) []\)"
proof -

61
let \(?x = \text{"insert}(t,s)"

have \(?x \in \text{set } \text{(unlabel (transaction_updates } T))\)
  using assms(3) wellformed_transaction_unlabel_cases[OF T_valid, of ?x]
  by (auto simp add: transaction_strand_def unlabel_def)

hence \(*\): \("\text{is_Var } (\text{the}\_\text{elem}\_\text{term } ?x)" \ "\text{fst } (\text{the}\_\text{Var } (\text{the}\_\text{elem}\_\text{term } ?x)) = T\text{Atom Value}\"
  "\text{is_Fun } (\text{the}\_\text{set}\_\text{term } ?x)" \ "\text{args } (\text{the}\_\text{set}\_\text{term } ?x) = []"
  "\text{is_Set } (\text{the}\_\text{Fun } (\text{the}\_\text{set}\_\text{term } ?x))"
  using assms(2) unfolding admissible_transaction_updates_def is_Fun_Set_def by fastforce+

show \("\exists n. t = \text{Var } (T\text{Atom Value}, n)"\) using *(1,2) by (cases t) auto
show \("\exists u. s = \text{Fun } (\text{Set } u) []"\) using *(3,4,5) unfolding is_Set_def by (cases s) auto
qed

lemma transaction_deletes_are_Value_vars:
  assumes T_valid: \("\text{wellformed}\_\text{transaction } T"\)
  and \("\text{admissible}\_\text{transaction}\_\text{updates } T"\)
  and \("\text{delete } \langle t,s \rangle \in \text{set } \text{(unlabel } (\text{transaction}\_\text{strand } T))"\)
  shows \("\exists n. t = \text{Var } (T\text{Atom Value}, n)"\) \\
   and \("\exists u. s = \text{Fun } (\text{Set } u) []"\)
proof -
  let ?x = \("\text{delete } \langle t,s \rangle"\)

  have \("\?x \in \text{set } \text{(unlabel } (\text{transaction}\_\text{updates } T))"\)
    using assms(3) wellformed_transaction_unlabel_cases[OF T_valid, of ?x]
    by (auto simp add: transaction_strand_def unlabel_def)

  hence \(*\): \("\text{is_Var } (\text{the}\_\text{elem}\_\text{term } ?x)" \ "\text{fst } (\text{the}\_\text{Var } (\text{the}\_\text{elem}\_\text{term } ?x)) = T\text{Atom Value}\"
    "\text{is_Fun } (\text{the}\_\text{set}\_\text{term } ?x)" \ "\text{args } (\text{the}\_\text{set}\_\text{term } ?x) = []"
    "\text{is_Set } (\text{the}\_\text{Fun } (\text{the}\_\text{set}\_\text{term } ?x))"
    using assms(2) unfolding admissible_transaction_updates_def is_Fun_Set_def by fastforce+

  show \("\exists n. t = \text{Var } (T\text{Atom Value}, n)"\) using *(1,2) by (cases t) auto
  show \("\exists u. s = \text{Fun } (\text{Set } u) []"\) using *(3,4,5) unfolding is_Set_def by (cases s) auto
qed

lemma transaction_selects_are_Value_vars:
  assumes T_valid: \("\text{wellformed}\_\text{transaction } T"\)
  and \("\text{admissible}\_\text{transaction}\_\text{checks } T"\)
  and \("\text{select } \langle t,s \rangle \in \text{set } \text{(unlabel } (\text{transaction}\_\text{strand } T))"\)
  shows \("\exists n. t = \text{Var } (T\text{Atom Value}, n) \wedge (T\text{Atom Value}, n) \notin \text{set } (\text{transaction}\_\text{fresh } T)" \ (is \(?A\))\)
   and \("\exists u. s = \text{Fun } (\text{Set } u) []" \ (is \(?B\))\)
proof -
  let ?x = \("\text{select } \langle t,s \rangle"\)

  have \("\?x \in \text{set } \text{(unlabel } (\text{transaction}\_\text{checks } T))"\)
    using assms(3) wellformed_transaction_unlabel_cases[OF T_valid, of ?x]
    by (auto simp add: transaction_strand_def unlabel_def)

  have \(*\): \("\text{is_Var } (\text{the}\_\text{elem}\_\text{term } ?x)" \ "\text{fst } (\text{the}\_\text{Var } (\text{the}\_\text{elem}\_\text{term } ?x)) = T\text{Atom Value}\"
    "\text{is_Fun } (\text{the}\_\text{set}\_\text{term } ?x)" \ "\text{args } (\text{the}\_\text{set}\_\text{term } ?x) = []"
    "\text{is_Set } (\text{the}\_\text{Fun } (\text{the}\_\text{set}\_\text{term } ?x))"
    using assms(2) unfolding admissible_transaction_checks_def is_Fun_Set_def by fastforce+

  have \("\text{fv}_{\text{sstp}} ?x \subseteq \text{fv}_{\text{lsst}} (\text{transaction}\_\text{checks } T)"\)
    using \(*\) by force

  hence \(*\*\): \("\text{fv}_{\text{sstp}} ?x \cap \text{set } (\text{transaction}\_\text{fresh } T) = {}"\)
    using T_valid unfolding wellformed_transaction_def by fast

  show \(?A\) using \(*\)(1,2) \(*\*) by (cases t) auto
  show \(?B\) using \(*\)(3,4,5) unfolding is_Set_def by (cases s) auto
qed

lemma transaction_inset_checks_are_Value_vars:
assumes $T_{\text{valid}}$: "admissible\_transaction $T$"
and $t$: "$(t \in s) \in \text{set (unlabel (transaction\_strand $T$))}" shows "$\exists n. t = \text{Var} (\text{TAtom Value}, n) \land (\text{TAtom Value}, n) \notin \text{set (transaction\_fresh $T$)}$" (Is ?A)
and "$\exists u. s = \text{Fun} (\text{Set} u) [\]" (Is ?B)

proof -
let $?x = "(t \in s)"

note $T_{\text{wf}} = \text{admissible\_transaction_is\_wellformed\_transaction(1)[OF } T_{\text{valid}}$
note $T_{\text{adm\_checks}} = \text{admissible\_transaction_is\_wellformed\_transaction(2)[OF } T_{\text{valid}}$

have $*: "?x \in \text{set (unlabel (transaction\_checks $T$))}"$
using $t \text{ wellformed\_transaction\_unlabel\_cases[OF } T_{\text{wf}}, \text{of } ?x]$
unfolding transaction\_strand\_def unlabel\_def by fastforce

have $**: "\text{is\_Var (the\_elem\_term ?x)}" \ "\text{fst (the\_Var (the\_elem\_term ?x))} = \text{TAtom Value}\"$
"\text{is\_Fun (the\_set\_term ?x)}" \ "\text{args (the\_set\_term ?x)} = [\]" \ "\text{is\_Set (the\_Fun (the\_set\_term ?x))}\"
using $* \ T_{\text{adm\_checks}}$ unfolding admissible\_transaction\_checks\_def is\_Fun\_Set\_def by fastforce+

have "$\text{fv\_sstp } ?x \subseteq \text{fv\_lstate (transaction\_checks $T$)}$" using $* \ by \ force$
hence $***: "\text{fv\_sstp } ?x \cap \text{set (transaction\_fresh $T$)} = \{\}"
using $T_{\text{wf}}$ unfolding wellformed\_transaction\_def by fast

show ?A using $**(1,2)$ *** by (cases $t$) auto
show ?B using $**(3,4,5)$ unfolding is\_Set\_def by (cases $s$) auto

qed

lemma transaction\_notinset\_checks\_are\_Value\_vars:
assumes $T_{\text{adm}}$: "admissible\_transaction $T$"
and $FG$: "$\forall X \langle \vee\neg \neg F \vee\neg \neg G \rangle \in \text{set (unlabel (transaction\_strand $T$))}"$
and $t$: "$(t,s) \in \text{set } G$"
shows "$\exists n. t = \text{Var} (\text{TAtom Value}, n) \land (\text{TAtom Value}, n) \notin \text{set (transaction\_fresh $T$)}" (Is ?A)
and "$\exists u. s = \text{Fun} (\text{Set} u) [\]" (Is ?B)

proof -
let $?x = "$\forall X \langle \vee\neg \neg F \vee\neg \neg G \rangle$"

note $T_{\text{wf}} = \text{admissible\_transaction_is\_wellformed\_transaction(1)[OF } T_{\text{adm}}$
note $T_{\text{adm\_checks}} = \text{admissible\_transaction_is\_wellformed\_transaction(2)[OF } T_{\text{adm}}$

have $0: "?x \in \text{set (unlabel (transaction\_checks $T$))}"$
using $FG$ wellformed\_transaction\_unlabel\_cases[ OF $T_{\text{wf}}, \text{of } ?x]$ by (auto simp add: transaction\_strand\_def unlabel\_def)

hence $1: "F = [\] \land \text{length } G = 1"$
using $T_{\text{adm\_checks}}$ $t$ unfolding admissible\_transaction\_checks\_def by fastforce

hence $"\text{hd } G = (t,s)"$ using $t$ by (cases "\text{the\_ins } ?x") auto

hence $**: "\text{is\_Var } t" \ "\text{fst (the\_Var } t) = \text{TAtom Value}\" \ "\text{is\_Fun } s" \ "\text{args } s = [\]" \ "\text{is\_Set (the\_Fun } s)"
using $1$ Set.bspec[OF $T_{\text{adm\_checks[unfolded admmissible\_transaction\_checks\_def]} 0}$]

unfolding is\_Fun\_Set\_def by auto

have "$\text{fv\_sstp } ?x \subseteq \text{fv\_lstate (transaction\_checks $T$)}$" using $0$ by force+
moreover have "$\text{set (transaction\_fresh $T$) \cap \text{fv\_lstate (transaction\_receive $T$)} = \{\}}$" 
"$\text{set (transaction\_fresh $T$) \cap \text{fv\_lstate (transaction\_checks $T$)} = \{\}}$"
using $T_{\text{wf}}$ unfolding wellformed\_transaction\_def by fast*

ultimately have "$\text{fv\_sstp } ?x \cap \text{set (transaction\_fresh $T$)} = \{\}$" 
"$\text{set (bvars\_sstp } ?x) \cap \text{set (transaction\_fresh $T$)} = \{\}$"
using admissible\_transaction\_E(7)[OF $T_{\text{adm}}$
wellformed\_transaction\_wf(2)[OF $T_{\text{wf}}$
fv\_transaction\_unfold[of $T$] bvars\_transaction\_unfold[of $T$]

63
by (blast, blast)
hence **: "fv t \cap set (transaction_fresh T) = {}"
using t by auto

show ?A using **(1,2) *** by (cases t) auto
show ?B using **(3,4,5) unfolding is_Set_def by (cases s) auto
qed

lemma admissible_transaction_strand_step_cases:
assumes T_adm: "admissible_transaction T"
shows "r \in set (unlabel (transaction_receive T)) = \exists t. r = receive(t)"
(is "?A = \Rightarrow ?A'")
and "r \in set (unlabel (transaction_checks T)) =
(\exists s t. r = (\langle s \Rightarrow t \rangle \lor r = (s := t) \lor r = (s \not\Rightarrow t))"
(is "?B = \Rightarrow ?B'")
and "r \in set (unlabel (transaction_updates T)) =
(\exists a s t. r = (\langle s\not\Rightarrow t \rangle \lor r = (s = t) \lor r = (s \Rightarrow t))"
(is "?C = \Rightarrow ?C'")
and "r \in set (unlabel (transaction_send T)) = \exists t. r = send(t)"
(is "?D = \Rightarrow ?D'")
proof -
  note T_wf = admissible_transaction_is_wellformed_transaction(1)[OF T_adm]

  show "?A = \Rightarrow ?A'"
  using T_wf Ball_set[of "unlabel (transaction_receive T)" is_Receive]
  unfolding wellformed_transaction_def is_Receive_def
  by blast

  show "?D = \Rightarrow ?D'"
  using T_wf Ball_set[of "unlabel (transaction_send T)" is_Send]
  unfolding wellformed_transaction_def is_Send_def
  by blast

  show "?B = \Rightarrow ?B'"
  proof -
    assume r: ?B
    note adm_checks = admissible_transaction_is_wellformed_transaction(1,2)[OF T_adm]

    have fvr1: "fv sstp r \subseteq fv_transaction T"
    using r fv_transaction_unfold[of T] by auto

    have fvr2: "fv sstp r \cap set (transaction_fresh T) = {}"
    using r T_wf unfolding wellformed_transaction_def by fastforce

    have "list_all is_Check_or_Assignment (unlabel (transaction_checks T))"
    using adm_checks(1) unfolding wellformed_transaction_def by blast
    hence "is_InSet r \lor is_Equality r \lor is_NegChecks r"
    using r unfolding list_all_iff by blast
    thus ?B' proof (elim disjE conjE)
      assume *: "is_InSet r"
      hence **: "is_Var (the_elem_term r)" "is_Fun (the_set_term r)"
      "is_Set (the_Fun (the_set_term r))" "args (the_set_term r) = []"
      "fst (the_Var (the_elem_term r)) = TAtom Value"
      using r adm_checks unfolding admissible_transaction_checks_def is_Fun_Set_def
      by fast+

      obtain ac rt rs where r': "r = (\langle ac: rt \in rs \rangle)" using * by (cases r)
      obtain x where x: "rt = Var x" "fst x = TAtom Value" using **(1,5) r' by auto

  64
 obtains \( f S \) where \( fS : r s = \text{Fun} f S \) using **(2)** \( r' \) by auto
 obtains \( s \) where \( s : f = \text{Set} s \) using **(3)** \( fS r' \) by (cases \( f \)) auto
 hence \( S : S = [\] \) using **(4)** \( fS r' \) by auto

show \(?B'\) using \( r' x fS S f v_r1 f v_r2 \) by (cases \( ac \)) simp_all

next
 assume \(*: \text{is-NegChecks} r\)
 hence **: \("bvars\), r = [\]
  \("the_eqs r = [\] \land \text{length} (\text{the_ins} r) = 1\) \lor
  \("the_ins r = [\] \land \text{length} (\text{the_eqs} r) = 1\)"

 using \( r \text{adm_checks unfolding admissible_transaction_checks_def by fast+} \)
 show \(?B'\) using **(2)**

 proof (elim disjE conjE)
 assume \(**: \"the_eqs r = [\]" \"length (\text{the_ins} r) = 1\"
 then obtain \( t s \) where \( \"the_ins r = [(t,s)]\" \) by (cases \("\text{the_ins} r\") auto
 hence \( \"hd (\text{the_ins} r) = (t,s)\" \) by simp
 hence \(**: \"\text{is-Var} \ (\text{fst} (t,s))" \ "\text{is-Fun} \ (\text{snd} (t,s))""
  \("\text{is-Set} (\text{the Fun} (\text{snd} (t,s)))" \ "\text{args} (\text{snd} (t,s)) = [\]"
  \("\text{fst} (\text{the Var} (\text{fst} (t,s))) = \text{TAtom Value}\"

 using \(* *\*(1) \text{Set.bspec}[OF \text{adm_checks}(2)[\text{unfolded admissible_transaction_checks_def} \ r]\)
 unfolding \( \text{is-Fun-Set_def by simp_all} \)
 obtain \( x \) where \( x: \"t = \text{Var} x" \ "\text{fst} x = \text{TAtom Value}\" \) using \(* *\*(5)\) by (cases \( t \)) simp_all
 obtain \( f S \) where \( fS : \"s = \text{Fun} f S\" \) using \(* *\*(2)\) by (cases \( s \)) simp_all
 obtain \( ss \) where \( \"f = \text{Set} ss\" \) using \( fS \) \(* *\*(3)\) by (cases \( f \)) simp_all

 have \( S : S = [\] \) using \(* *\*(4)\) simp_all

 show \(?B'\) using \( ts x fS ss S \)

 next
 assume \(**: \"\text{the_eqs r} = [\]" \"length (\text{the_ins} r) = 1\"
 then obtain \( t s \) where \( \"\text{the_ins r} = [(t,s)]\" \) by (cases \("\text{the_ins} r\") auto
 thus \(?B'\) using ** **(1) \* by (cases \( r \)) auto

 qed

 qed (auto simp add: \text{is-Equality_def the_check_def intro: poscheckvariant.exhaust})

 qed

 show \(?C \implies \ ?C'\"
 proof -
 assume \( r : ?C \)
 note \( \text{adm_upds = admissible_transaction_is_wellformed_transaction}(3)[OF \ T_{adm}] \)

 have \(*: \text{is-Update} r\ "\text{is-Var} (\text{the elem term} r)" \ "\text{is-Fun} (\text{the set term} r)"
  \("\text{is-Set} (\text{the Fun} (\text{the set term} r))" \ "\text{args} (\text{the set term} r) = [\]"
  \("\text{fst} (\text{the Var} (\text{the elem term} r)) = \text{TAtom Value}\"

 using \( r \text{adm_upds unfolding admissible_transaction_updates_def is-Fun-Set_def by fast+} \)
 obtain \( t s \) where \( \"r = \text{insert}(t,s) \lor r = \text{delete}(t,s)\" \) using \*(1) by (cases \( r \)) auto
 obtain \( x \) where \( x: \"t = \text{Var} x" \ "\text{fst} x = \text{TAtom Value}\" \) using \* \( (2,6) \) by (cases \( t \)) auto
 obtain \( f T \) where \( fT : \"s = \text{Fun} f T\" \) using \* \( (3) \) by (cases \( s \)) auto
 obtain \( ss \) where \( \"f = \text{Set ss}\" \) using \( fT \) \* \( (4) \) by (cases \( f \)) fastforce+

 have \( T : T = [\] \) using \( fT \) \* \( (5) \) by (cases \( T \)) auto

 show \(?C'\"
 using \( ts x fT ss T \) by blast

 qed

 lemma \text{protocol_transaction_vars_TAtom_typed:}
 assumes \( T_{adm}: \text{admissible_transaction T} \)
 shows \("\forall x \in \text{vars_transaction} T. \ \Gamma_v x = \text{TAtom Value} \lor (\exists a. \ \Gamma_v x = \text{TAtom (Atom a)})"
 and \("\forall x \in \text{fv_transaction} T. \ \Gamma_v x = \text{TAtom Value} \lor (\exists a. \ \Gamma_v x = \text{TAtom (Atom a)})"
 and \("\forall x \in \text{set (transaction_fresh T)}. \ \Gamma_v x = \text{TAtom Value}\"

 proof -
 note \( T_{wf} = \text{admissible_transaction_is_wellformed_transaction}(1)[OF \ T_{adm}] \)
show \( \forall x \in \text{vars\_transaction } T. \Gamma_v x = \text{TAtom Value} \lor (\exists a. \Gamma_v x = \text{TAtom (Atom a)}) \) using admisible\_transactionE(3)[OF T\_adm] by fast
thus \( \forall x \in \text{fv\_transaction } T. \Gamma_v x = \text{TAtom Value} \lor (\exists a. \Gamma_v x = \text{TAtom (Atom a)}) \) using vars\_st\_is\_fv\_st\_bvars\_st by fast

show \( \forall x \in \text{set (transaction\_fresh } T). \Gamma_v x = \text{TAtom Value} \) using admisible\_transactionE(2)[OF T\_adm] by argo
qed

lemma protocol\_transactions\_no\_pubconsts:
assumes "admissible\_transaction T"
shows "Fun (Val n) S / \in \text{subterms set (trms\_transaction } T)"
and "Fun (PubConst Value n) S / \in \text{subterms set (trms\_transaction } T)"
using assms admisible\_transactions\_no\_Value\_consts(1,3) by (blast, blast)

lemma protocol\_transactions\_no\_abss:
assumes "admissible\_transaction T"
shows "Fun (Abs n) S / \in \text{subterms set (trms\_transaction } T)"
using assms admisible\_transactions\_no\_Value\_consts(2) by fast

lemma admissible\_transaction\_strand\_sem\_fv\_ineq:
assumes T\_adm: "admissible\_transaction T"
and I: "strand\_sem\_stateful IK DB (unlabel (\text{dual queryString } T \cdot \text{lsst } \vartheta)) I"
and x: "x \in \text{fv\_transaction } T \cdot \text{set (transaction\_fresh } T)"
and y: "y \in \text{fv\_transaction } T \cdot \text{set (transaction\_fresh } T)"
and x\_not\_y: "x \neq y"
shows "\( \vartheta x \cdot I \neq \vartheta y \cdot I \)"
proof -
have "\( \langle \text{Var } x \neq \text{Var } y \rangle \in \text{set (unlabel (transaction\_checks } T)) \lor \langle \text{Var } y \neq \text{Var } x \rangle \in \text{set (unlabel (transaction\_checks } T)) \)"
  using x y x\_not\_y admisible\_transactionE(8)[OF T\_adm] by auto
hence "\( \langle \text{Var } x \neq \text{Var } y \rangle \in \text{set (unlabel (transaction\_strand } T) \cdot \text{lsst } \vartheta) \lor \langle \text{Var } y \neq \text{Var } x \rangle \in \text{set (unlabel (transaction\_strand } T) \cdot \text{lsst } \vartheta) \)"
  unfolding transaction\_strand\_def unlabel\_def by auto
hence "\( \langle \vartheta x \neq \vartheta y \rangle \in \text{set (unlabel (\text{dual queryString } T \cdot \text{lsst } \vartheta))} \lor \langle \vartheta y \neq \vartheta x \rangle \in \text{set (unlabel (\text{dual queryString } T \cdot \text{lsst } \vartheta))} \)"
  using stateful\_strand\_step\_subst\_inI(8)[of _ _ "unlabel (transaction\_strand } T)" \vartheta]
  dual\_lst\_unlabel\_steps\_iff(7)[of "[]" _ "[]"]
  by force
then obtain B where B: "prefix (B @ \langle \vartheta x \neq \vartheta y \rangle) (\text{unlabel (\text{dual queryString } T \cdot \text{lsst } \vartheta)}) \lor prefix (B @ \langle \vartheta y \neq \vartheta x \rangle) (\text{unlabel (\text{dual queryString } T \cdot \text{lsst } \vartheta)))"
  unfolding prefix\_def
  by (metis (no_types, opaque_lifting) append.assoc append.Cons append.Nil split_list)
thus \?thesis
using I strand\_sem\_append\_stateful[of IK DB _ _ I]
  stateful\_strand\_sem\_NegChecks\_no\_bvars(2)
  unfolding prefix\_def
  by metis
qed

lemma admissible\_transaction\_terms\_wf\_trms:\nassumes "admissible\_transaction\_terms T"
shows "wf\_trms (trms\_transaction T)"
by (rule conjunct1[OF assms[unfolded admissible\_transaction\_terms\_def wf\_trms\_code[symmetric]]])

lemma admissible\_transactions\_wf\_trms:\nassumes "admissible\_transaction T"
shows "wf\_trms (trms\_transaction T)"
proof -
have "admissible_transaction_terms T" using assms[unfolded admissible_transaction_def] by fast 
thus ?thesis by (metis admissible_transaction_terms_wf trms)
qed

lemma admissible_transaction_no_Ana_Attack:
  assumes "admissible_transaction_terms T"
  and "t ∈ substterms set (trms_transaction T)"
  shows "attack(n) ∉ set (snd (Ana t))"
proof -
  obtain r where r: "r ∈ set (unlabel (transaction_strand T))" "t ∈ substterms set (trms_sstp r)"
  using assms(2) by force
  obtain K M where t: "Ana t = (K, M)"
  by (metis surj_pair)
  show ?thesis proof
    assume n: "attack⟨n⟩ ∈ set (snd (Ana t))"
    hence "attack⟨n⟩ ∈ set M" using t by simp
    hence n': "attack⟨n⟩ ∈ substterms set (trms_sstp r)" unfolding is_Attack_def by fast
    hence "∃ f ∈ ∪ (funs_term ` trms_sstp r). is_Attack f" using funs_term_Fun_subterm' unfolding is_Attack_def by fast
    hence "is_Send r" "length (the_msgs r) = 1" "is_Fun (hd (the_msgs r))" "args (hd (the_msgs r)) = []" using assms(1) r(1) unfolding admissible_transaction_terms_def is_Fun_Attack_def by meson+
    hence "t = attack⟨n⟩" using n' r(2) unfolding is_Send_def is_Attack_def by (cases "the_msgs r") auto
    thus False using n by fastforce
  qed
qed

lemma admissible_transaction_Value_vars:
  assumes T: "admissible_transaction T" and x: "x ∈ fv_transaction T"
  shows "Γᵥ x = TAtom Value"
proof -
  have "x ∈ vars_transaction T" using x vars_sst is_fv by blast
  thus "Γᵥ x = TAtom Value" using admissible_transactionE(3)[OF T] by simp
qed

lemma admissible_transaction_occurs_checksE1:
  assumes T: "admissible_transaction_occurs_checks T" and x: "x ∈ fv_transaction T - set (transaction_fresh T)" "Γᵥ x = TAtom Value"
  obtains l ts S where "transaction_receive T = (l, receive⟨ts⟩)#S" "occurs (Var x) ∈ set ts"
proof -
  let ?rcvs = "transaction_receive T" let ?frsh = "transaction_fresh T"
  let ?fvs = "fv_transaction T"
  have "∀ x ∈ fvs. is_Fun (hd (?rcvs))" using ?rcvs ≠ [] unfolding is_Fun_def by blast
  "∀ x ∈ fvs. is_Fun (hd (?frsh))" using ?frsh ≠ [] unfolding is_Fun_def by blast
  "∀ x ∈ fvs. occurs (Var x) ∈ set (the_msgs (hd (unlabel (?rcvs))))" using ?rcvs ≠ [] unfolding admissible_transaction_occurs_checks_def Γᵥ=TAtom'"(2) by meson+
  obtain r S where S: "(?rcvs = r)#S" using *(1) by (cases ?rcvs) auto
  obtain l ts where r: "r = (l, receive(ts))"
by (metis *(1,2) S list.map_sel(1) list.sel(1) prod.collapse is_Receive_def unlabel_def)

have 0: "occurs (Var x) ∈ set ts" using *(3) x S r by fastforce

show ?thesis using that[unfolded S r, of l ts S] 0 by blast
qed

lemma admissible_transaction_occurs_checksE2:
  assumes T: "admissible_transaction_occurs_checks T"
  and x: "x ∈ set (transaction_fresh T)"
  obtains l ts S where
    "transaction_send T = (l,send⟨ts⟩)#S" "occurs (Var x) ∈ set ts"
proof -
  let ?snds = "transaction_send T"
  let ?frsh = "transaction_fresh T"
  let ?fvs = "fv_transaction T"
  define ts where "ts ≡ the_msgs (hd (unlabel ?snds))"
  let ?P = "∀ x ∈ set ?frsh. occurs (Var x) ∈ set ts"
  have *: "?_sndts = r#S" "is_Send (hd (unlabel ?snds))" ?P
    using T x unfolding admissible_transaction_occurs_checks_def ts_def
    by meson+
  obtain r S where S: "?snds = r#S"
    using *(1) by (cases ?snds) auto
  obtain l where r: "r = (l,send⟨ts⟩)"
    by (metis *(1,2) S list.map_sel(1) list.sel(1) prod.collapse unlabel_def ts_def
       stateful_strand_step.collapse(1))
  have ts: "occurs (Var x) ∈ set ts"
    using x *(3) unfolding S by auto
  show ?thesis using that[unfolded S r, of l ts S] ts
    by blast
qed

lemma admissible_transaction_occurs_checksE3:
  assumes T: "admissible_transaction_occurs_checks T"
  and t: "OccursFact ∈ funs_term t ∨ OccursSec ∈ funs_term t" "t ∈ set ts"
  and ts: "send⟨ts⟩ ∈ set (unlabel (transaction_send T))"
  obtains x where "t = occurs (Var x)" "x ∈ set (transaction_fresh T)"
proof -
  let ?P = "∀ t. ∃ x ∈ set (transaction_fresh T). t = occurs (Var x)"
  have "?P t" when "t ∈ trms_sst (transaction_send T)" "OccursFact ∈ funs_term t ∨ OccursSec ∈ funs_term t"
    for t
    using assms that unfolding admissible_transaction_occurs_checks_def by metis
  moreover have "t ∈ trms_sst (transaction_send T)"
    using t(2) ts unfolding trms_sst_def by fastforce
  ultimately have "?P t" using t(1) by blast
  thus thesis using that by blast
qed

lemma admissible_transaction_occurs_checksE4:
  assumes T: "admissible_transaction_occurs_checks T"
  and ts: "send⟨ts⟩ ∈ set (unlabel (transaction_send T))"
  and t: "occurs t ∈ set ts"
  obtains x where "t = Var x" "x ∈ set (transaction_fresh T)"
  using admissible_transaction_occurs_checksE3[OF T _ t ts] by auto

lemma admissible_transaction_occurs_checksE5:
assumes $T$: "admissible_transaction_occurs_checks $T"
shows "Fun OccursSec $[\_]$ $\notin$ trms$_{\mathfrak{s,t}}$ (transaction_send $T$)"
proof -
  have "$\exists x \in \text{set (transaction_fresh} T). t = \text{occurs} (\text{Var} x)"
  where "t $\in$ trms$_{\mathfrak{s,t}}$ (transaction_send $T$)" "\text{OccursFact} \in \text{funs_term} t \lor \text{OccursSec} \in \text{funs_term} t"
  for t
  using assms that unfolding admissible_transaction_occurs_checks_def by metis
thus \text{thesis} by fastforce
qed

lemma admissible_transaction_occurs_checksE6:
  assumes $T$: "admissible_transaction_occurs_checks $T"
  and t: "t $\subseteq$ \text{set (transaction_send} T)"
  shows "Fun OccursSec $[\_]$ $\notin$ \text{set (snd (Ana} t))" (is $?A$)  
  and "\text{occurs} k $\notin$ \text{set (snd (Ana} t))" (is $?B$)
proof -
  obtain t' where t': "t' $\in$ trms$_{\mathfrak{s,t}}$ (transaction_send $T$)" "t $\subseteq$ t'" using t by blast
  have "$?A \land ?B$"
  proof (rule ccontr)
    assume *: "$\neg (\neg A \land \neg B)$"
    hence "\text{OccursSec} \in \text{funs_term} t' \lor \text{OccursFact} \in \text{funs_term} t'"
      by (meson t'(2) Ana_subterm Fun_subterm term.order.trans)
    then obtain x where x: "x $\in$ \text{set (transaction_fresh} T)" "t' = \text{occurs} (\text{Var} x)"
      using t'(1) T unfolding admissible_transaction_occurs_checks_def by metis
    have "t = \text{occurs} (\text{Var} x) \lor t = \text{Var} x \lor t = \text{Fun OccursSec} [\_]" using x(2) t'(2) by auto
    thus False using * by fastforce
  qed
  thus $?A$ $?B$ by simp_all
qed

3.3.7 Lemmata: Renaming, Declaration, and Fresh Substitutions

lemma transaction_decl_subst_empty_inv:
  assumes "transaction_decl_subst Var $T$"
  shows "transaction_decl $T$ () = [\]"
  using assms unfolding transaction_decl_subst_def subst_domain_Var by blast

lemma transaction_decl_subst_domain:
  fixes $\xi$ :: "('fun,'atom,'sets,'lbl) prot_subst"
  assumes "transaction_decl_subst $\xi$ $T$"
  shows "\text{subst_domain} $\xi$ = \text{fst ` set (transaction_decl} T ()\)"
  using assms unfolding transaction_decl_subst_def by argo

lemma transaction_decl_subst_grounds_domain:
  fixes $\xi$ :: "('fun,'atom,'sets,'lbl) prot_subst"
  assumes "transaction_decl_subst $\xi$ $T$"
  and "x $\in$ \text{fst ` set (transaction_decl} T ())"
  shows "\text{fv} (\xi \ x) = [\]"
  proof -
    obtain c where "\xi \ x = \text{Fun} \ c [\]"
      using assms unfolding transaction_decl_subst_def by moura
    thus \text{thesis} by simp
  qed

lemma transaction_decl_subst_range_vars_empty:
  fixes $\xi$ :: "('fun,'atom,'sets,'lbl) prot_subst"
  assumes "transaction_decl_subst $\xi$ $T$"
  shows "\text{range_vars} $\xi$ = [\]"
  using assms unfolding transaction_decl_subst_def range_vars_def by auto

lemma transaction_decl_subst_wr:
  fixes $\xi$ :: "('fun,'atom,'sets,'lbl) prot_subst"
  assumes "transaction_decl_subst $\xi$ $T$"
shows "\( w_{\text{subst}} \xi \)"
using assms unfolding transaction_decl_subst_def by blast

lemma transaction_decl_subst_is_wf_trm:
fixes \( \xi \) :: "('fun,'atom,'sets,'lbl) prot_subst"
assumes "transaction_decl_subst \( \xi \) P"
shows "\( \text{wf}(x) \)"
proof (cases "\( v \in \text{subst\_domain} \xi \)"
  True thus ?thesis using assms unfolding transaction_decl_subst_def by fastforce
qed auto

lemma transaction_decl_subst_range_wf_trms:
fixes \( \xi \) :: "('fun,'atom,'sets,'lbl) prot_subst"
assumes "transaction_decl_subst \( \xi \) P"
shows "\( \text{wf}(\text{subst\_range} \xi) \)"
by (metis transaction_decl_subst_is_wf_trm[OF assms] wf_trm_subst_range_iff)

lemma transaction_renaming_subst_is_renaming:
fixes \( \alpha \) :: "('fun,'atom,'sets,'lbl) prot_subst"
assumes \( \alpha \) : "transaction_renaming_subst \( \alpha \) P A"
shows "\( \exists m. \forall \tau n. \alpha (\tau,n) = \text{Var} (\tau,n+Suc m)\)" (is ?A)
and "\( \exists y. \alpha x = \text{Var} y\)" (is ?B)
and "\( \alpha x \neq \text{Var} x\)" (is ?C)
and "\( \text{subst\_domain} \alpha = \text{UNIV}\)" (is ?D)
and "\( \text{subst\_range} \alpha \subseteq \text{range Var}\)" (is ?E)
and "\( \text{fv} (t \cdot \alpha) \subseteq \text{range_vars} \alpha\)" (is ?F)
proof
- show 0: ?A using \( \alpha \) unfolding transaction_renaming_subst_def var_rename_def by force
- show 1: ?D using 0 by (cases x) auto
- show 1: ?E using 0 by fastforce
- show 1: ?F by (induct t) (auto simp add: 1 subst_dom_vars_in_subst subst_fv_imgI)
qed

lemma transaction_renaming_subst_vars_disj:
fixes \( \alpha \) :: "('fun,'atom,'sets,'lbl) prot_subst"
assumes "transaction_renaming_subst \( \alpha \) P A"
shows "\( \text{fv}(\text{set} (\alpha ` (\bigcup (\text{vars\_transaction} ` \text{set} P)))) \cap (\bigcup (\text{vars\_transaction} ` \text{set} P)) = \{\}\)" (is ?A)
and "\( \text{fv}(\alpha ` \text{vars\_set} A) \cap \text{vars\_set} A = \{\}\)" (is ?B)
and "\( \exists t \in \text{set} P \Rightarrow \text{vars\_transaction} T \cap \text{range_vars} \alpha = \{\}\)" (is ?C1)
and "\( \exists t \in \text{set} P \Rightarrow \text{bvars\_transaction} T \cap \text{range_vars} \alpha = \{\}\)" (is ?C2)
and "\( \exists t \in \text{set} P \Rightarrow \text{fv\_transaction} T \cap \text{range_vars} \alpha = \{\}\)" (is ?C3)
and "\( \text{vars\_set} A \cap \text{range_vars} \alpha = \{\}\)" (is ?D1)
and "\( \text{bvars\_set} A \cap \text{range_vars} \alpha = \{\}\)" (is ?D2)
and "\( \text{fv\_set} A \cap \text{range_vars} \alpha = \{\}\)" (is ?D3)

70
3.3 Stateful Protocol Model

proof
define X where "X ≡ \bigcup (vars_transaction \setminus P) \cup vars_{stt} A"

have 1: "finite X" by (simp add: X_def)

obtain n where n: "n ≥ max_var_set X" "α = var_rename n"
  using assms unfolding transaction_renaming_subst_def X_def by moura

hence 2: "∀ x ∈ X. snd x < Suc n" using less_Suc_max_var_set[OF _ 1]
  unfolding var_rename_def by fastforce

have 3: "x /∈ fv(set(α`X))" "fv(α x) ∩ X = {}" "x /∈ range_vars α" when x: "x ∈ X" for x
  using 2 n unfolding var_rename_def by force+

show ?A ?B using 3(1,2) unfolding X_def by auto

show ?C using T: "T ∈ set P" unfolding X_def by blast

thus ?C1 ?C2 ?C3 by (simp_all add: disjoint_iff_not_equal vars_{stt}_is_fv vars_{stt}_bvars)

show ?D using T: "T ∈ set P" unfolding X_def by blast

thus ?D1 ?D2 ?D3 by (simp_all add: disjoint_iff_not_equal vars_{stt}_is_fv vars_{stt}_bvars)

qed

lemma transaction_renaming_subst_wt:
  fixes α ::"('fun,'atom,'sets,'lbl) prot_subst"
  assumes "transaction_renaming_subst α P A"
  shows "wt subst α"
proof -
  { fix x::"('fun,'atom,'sets,'lbl) prot_var"
    obtain τ n where x: "x = (τ,n)" by moura
    then obtain m where m: "α x = Var (τ,n+Suc m)" using assms transaction_renaming_subst_is_renaming(1)
    hence "Γ(α x) = Γ_v x" using x by (simp add: Γ_v_def)
  } thus ?thesis by (simp add: wt subst_def)

qed

lemma transaction_renaming_subst_is_wf_trm:
  fixes α ::"('fun,'atom,'sets,'lbl) prot_subst"
  assumes "transaction_renaming_subst α P A"
  shows "wf_trm (α v)"
proof -
  obtain τ n where "v = (τ,n)" by moura
  then obtain m where "α v = Var (τ,n+Suc m)" using transaction_renaming_subst_is_renaming(1)
  hence "Γ(α v) = Γ_v x" using x by (simp add: Γ_v_def)
  thus ?thesis by (metis wf_trm_Var)

qed

lemma transaction_renaming_subst_range_wf_trms:
  fixes α ::"('fun,'atom,'sets,'lbl) prot_subst"
  assumes "transaction_renaming_subst α P A"
  shows "wf_trms (subst_range α)"
by (metis transaction_renaming_subst_is_wf_trm[OF assms] wf_trm_subst_range_iff)

lemma transaction_renaming_subst_range_notin_vars:
  fixes α ::"('fun,'atom,'sets,'lbl) prot_subst"
  assumes "transaction_renaming_subst α P A"
  shows "∀ y. α x = Var y ∧ y \in \bigcup (vars_transaction \setminus P) \cup vars_{stt} A"
proof -
  obtain τ n where x: "x = (τ,n)" by (metis surj_pair)
  define y where "y ≡ λm. (τ,n+Suc m)"


have \( \exists m \geq \text{max\_var\_set} (\bigcup (\text{vars\_transaction} \setminus \text{set} \, P) \cup \text{vars\_sst} \, A) \). \( \alpha \, x = \text{Var} \, (y \, m) \)"

using `assms x` by (auto simp add: `y_def transaction\_renaming\_subst\_def var\_rename\_def`)

moreover have "finite (\( \bigcup (\text{vars\_transaction} \setminus \text{set} \, P) \cup \text{vars\_sst} \, A \))"

by auto

ultimately show ?thesis using `x` unfolding `y_def` by force

qed

lemma `transaction\_renaming\_subst\_var\_obtain`:

fixes \( \alpha :: \text{('fun,'atom,'sets,'lbl) prot\_subst} \)

assumes \( \alpha : \text{transaction\_renaming\_subst} \, \alpha \, P \, A \)

shows "x \in \text{fv} \, \text{sst} \, \text{S} \cdot \text{sst} \, \alpha = \Rightarrow \exists \, y. \alpha \, y = \text{Var} \, x" (is "?A1 = \Rightarrow ?B1")

and "x \in \text{fv} \, (t \cdot \alpha) = \Rightarrow \exists \, y \in \text{fv} \, t. \alpha \, y = \text{Var} \, x" (is "?A2 = \Rightarrow ?B2")

proof

- assume \( x : ?A1 \)

  obtain \( y \) where \( y \in \text{fv} \, \text{sst} \, \text{S} \) "x \in \text{fv} \, (\alpha \, y)"

  using `fv\_sst\_subst\_obtain\_var[of x]` by moura

  thus \( ?B1 \) using `transaction\_renaming\_subst\_is\_renaming(2)[OF \, \alpha, \, of \, y]` by fastforce

- assume \( x : ?A2 \)

  obtain \( y \) where \( y \in \text{fv} \, t \) "x \in \text{fv} \, (\alpha \, y)"

  using `fv\_subst\_obtain\_var[of x]` by moura

  thus \( ?B2 \) using `transaction\_renaming\_subst\_is\_renaming(2)[OF \, \alpha, \, of \, y]` by fastforce

qed

lemma `transaction\_renaming\_subst\_set\_eq`:

assumes "set \, P1 = set \, P2"

shows "\text{transaction\_renaming\_subst} \, \alpha \, P1 \, A = \text{transaction\_renaming\_subst} \, \alpha \, P2 \, A" (is "?A = ?B")

using `assms` unfolding `transaction\_renaming\_subst\_def` by presburger

lemma `transaction\_fresh\_subst\_is\_wf\_trm`:

fixes \( \sigma :: \text{('fun,'atom,'sets,'lbl) prot\_subst} \)

assumes "\text{transaction\_fresh\_subst} \, \sigma \, T \, A"

shows "\text{wf\_trm} \, \sigma \, v"

proof (cases "v \in \text{subst\_domain} \, \sigma")

  case True

  then obtain \( c \) where \( \sigma \, v = \text{Fun} \, c \) "arity \, c = 0"

  using `assms` unfolding `transaction\_fresh\_subst\_def` by moura

  thus ?thesis by auto

qed auto

lemma `transaction\_fresh\_subst\_wt`:

fixes \( \sigma :: \text{('fun,'atom,'sets,'lbl) prot\_subst} \)

assumes "\text{transaction\_fresh\_subst} \, \sigma \, T \, A"

shows "\text{wt\_subst} \, \sigma"

using `assms` unfolding `transaction\_fresh\_subst\_def` by blast

lemma `transaction\_fresh\_subst\_domain`:

fixes \( \sigma :: \text{('fun,'atom,'sets,'lbl) prot\_subst} \)

assumes "\text{transaction\_fresh\_subst} \, \sigma \, T \, A"

shows "\text{subst\_domain} \, \sigma = \, \text{set} (\text{transaction\_fresh} \, T)"

using `assms` unfolding `transaction\_fresh\_subst\_def` by fast

lemma `transaction\_fresh\_subst\_range\_wf\_trms`:

fixes \( \sigma :: \text{('fun,'atom,'sets,'lbl) prot\_subst} \)

assumes "\text{transaction\_fresh\_subst} \, \sigma \, T \, A"

shows "\text{wf\_trms} \, (\text{subst\_range} \, \sigma)"

by (metis `transaction\_fresh\_subst\_is\_wf\_trm[of \, assms] \, \text{wf\_trm\_subst\_range\_iff}`)

lemma `transaction\_fresh\_subst\_range\_fresh`:

fixes \( \sigma :: \text{('fun,'atom,'sets,'lbl) prot\_subst} \)

assumes "\text{transaction\_fresh\_subst} \, \sigma \, T \, A"

shows "\forall t \in \text{subst\_range} \, \sigma. \, t \notin \text{subterms\_sst} \, \text{sst} \, (\text{trms\_sst} \, \text{A})"

and "\forall t \in \text{subst\_range} \, \sigma. \, t \notin \text{subterms\_sst} \, \text{sst} \, (\text{transaction\_strand} \, T)"

using `assms` unfolding `transaction\_fresh\_subst\_def` by meson+
3.3 Stateful Protocol Model

lemma transaction_fresh_subst_sends_to_val:
fixes $\sigma :: (\text{fun}, \text{atom}, \text{sets}, \text{lbl}) \text{ protsubst}$
assumes $\sigma :: \text{transaction_fresh_subst } \sigma T A$
and $y :: y \in \text{set (transaction_fresh T)}$ "\Gamma, y = TAtom Value"
obtains $n$ where "$\sigma y = \text{Fun (Val n)} []" "\text{Fun (Val n)} [] \in \text{subst_range } \sigma$
proof -
have "$\sigma y \in \text{subst_range } \sigma$" using assms unfolding transaction_fresh_subst_def by simp
obtain $c$ where:
$c :: \sigma y = \text{Fun c} []$ "\text{Fun c} [] \in \text{subst_range } \sigma$
using transaction_fresh_subst_sends_to_val[OF assms] by moura
thus $?thesis$ using $c$ that unfolding transaction_fresh_subst_def by fastforce
qed

lemma transaction_fresh_subst_sends_to_val':
fixes $\sigma \alpha :: (\text{fun}, \text{atom}, \text{sets}, \text{lbl}) \text{ protsubst}$
assumes $\text{transaction_fresh_subst } \sigma T A$
and $y :: y \in \text{set (transaction_fresh T)}$ "\Gamma v y = TAtom Value"
obtains $n$ where "$\sigma \circ s \alpha y \cdot I = \text{Fun (Val n)} []" "\text{Fun (Val n)} [] \in \text{subst_range } \sigma$
proof -
obtain $n$ where "$\sigma y = \text{Fun (Val n)} []" "\text{Fun (Val n)} [] \in \text{subst_range } \sigma$
using transaction_fresh_subst_sends_to_val[OF assms] by moura
thus $?thesis$ using that by (fastforce simp add: subst_compose_def)
qed

lemma transaction_fresh_subst_grounds_domain:
fixes $\sigma :: (\text{fun}, \text{atom}, \text{sets}, \text{lbl}) \text{ protsubst}$
assumes $\text{transaction_fresh_subst } \sigma T A$
and $y :: y \in \text{set (transaction_fresh T)}$
shows "$\text{fv (} \sigma y \text{)} = \{\}$"
proof -
obtain $c$ where "$\sigma y = \text{Fun c} []$
using assms unfolding transaction_fresh_subst_def by moura
thus $?thesis$ by simp
qed

lemma transaction_fresh_subst_range_vars_empty:
fixes $\sigma :: (\text{fun}, \text{atom}, \text{sets}, \text{lbl}) \text{ protsubst}$
assumes $\text{transaction_fresh_subst } \sigma T A$
shows "$\text{range_vars } \sigma = \{\}$"
proof -
have "$\text{fv t = \{\}}$" when "$t \in \text{subst_range } \sigma$" for $t$
using assms that unfolding transaction_fresh_subst_def by blast
thus $?thesis$ unfolding range_vars_def by fastforce
qed

lemma transaction_decl_fresh_renaming_substs_range:
fixes $\xi \sigma :: (\text{fun}, \text{atom}, \text{sets}, \text{lbl}) \text{ protsubst}$
assumes $\xi :: \text{transaction_decl_subst } \xi T$
and $\sigma :: \text{transaction_fresh_subst } \sigma T A$
and $\alpha :: \text{transaction_renaming_subst } \alpha P A$
shows "$x \in \text{fst } \text{set (transaction_decl T ()) } \implies$
$\exists c. (\xi \circ \sigma \circ \alpha) x = \text{Fun c} [] \land \text{arity c} = 0$
and "$x \notin \text{fst } \text{set (transaction_decl T ()) } \implies$
$\exists c. (\xi \circ \sigma \circ \alpha) x = \text{Fun c} [] \land \neg \text{public c} \land \text{arity c} = 0$"
and "x \notin \text{fst ` set (transaction_decl T ()))} \implies 
x \in \text{set (transaction_fresh T)} \implies 
fst x = TAtom Value \implies 
\exists n. (\xi_{\sigma_{o_3}} \circ_{o_3} \alpha) x = \text{Fun (Val n) []}"

and "x \notin \text{fst ` set (transaction_decl T ()))} \implies 
x \notin \text{set (transaction_fresh T)} \implies 
\exists y. (\xi_{\sigma_{o_3}} \circ_{o_3} \alpha) x = \text{Var } y"

proof -

assume "x \notin \text{fst ` set (transaction_decl T ()))}"
then obtain c where c: "\xi x = \text{Fun c []}" "\text{arity c = 0}"
using \xi unfolding transaction_decl_subst_def by fastforce
thus "\exists c. (\xi_{\sigma_{o_3}} \circ_{o_3} \alpha) x = \text{Fun c [] \land \text{arity c = 0}}"
using subst_compose[of "\xi_{\sigma_{o_3}} \circ_{o_3} \alpha x"] subst_compose[of \xi by simp]

next
assume x: "x \notin \text{fst ` set (transaction_decl T ()))}"
"x \notin \text{set (transaction_fresh T)}"

have *: "(\xi_{\sigma_{o_3}}) x = \sigma x"
using x(1) \xi unfolding transaction_decl_subst_def
by (metis (no_types, opaque_lifting) subst_comp_notin_dom)
then obtain c where c: "(\xi_{\sigma_{o_3}}) x = \text{Fun c []}" "\text{arity c = 0}"
using \sigma x(2) unfolding transaction_fresh_subst_def by fastforce
thus "\exists c. (\xi_{\sigma_{o_3}} \circ_{o_3} \alpha) x = \text{Fun c [] \land \text{arity c = 0}}"
using subst_compose[of "\xi_{\sigma_{o_3}} \circ_{o_3} \alpha x"] subst_compose[of \xi by simp]

assume "fst x = TAtom Value"

hence "\xi (\xi_{\sigma_{o_3}} x) = TAtom Value"

using * \sigma T''(2)[of x] wt_subst_trm''[of \sigma "\text{Var x}"]
unfolding transaction_fresh_subst_def by simp

then obtain n where "c = \text{Val n}"
using c by (cases c) (auto split: option.splits)

thus "\exists n. (\xi_{\sigma_{o_3}} \circ_{o_3} \alpha) x = \text{Fun (Val n) []}"
using c subst_compose[of "\xi_{\sigma_{o_3}} \circ_{o_3} \alpha x"] subst_compose[of \xi by simp]

next
assume x: "x \notin \text{fst ` set (transaction_decl T ()))}"
"x \notin \text{set (transaction_fresh T)}"

hence "(\xi_{\sigma_{o_3}}) x = \text{Var } x"

using \xi unfolding transaction_decl_subst_def transaction_fresh_subst_def
by (metis (no_types, opaque_lifting) subst_comp_notin_dom_eq subst_dom)

thus "\exists y. (\xi_{\sigma_{o_3}} \circ_{o_3} \alpha) x = \text{Var } y"
using transaction_renaming_subst_is_renaming(1)[OF \alpha]

substitute[of "\xi_{\sigma_{o_3}} \circ_{o_3} \alpha x"] subst_compose[of \xi by simp]

by (cases x) force

qed

lemma transaction_decl_fresh_renaming_substs_range':
fixes \sigma :: "('fun,'atom,'sets,'lbl) prot_subst"
assumes \xi: "transaction_decl_subst \xi T"
and \sigma: "transaction_fresh_subst \sigma T A"
and \alpha: "transaction_renaming_subst \alpha P A"
and t: "t \in \text{substs_range} (\xi_{\sigma_{o_3}} \circ_{o_3} \alpha)"

shows "(\exists c. t = \text{Fun c [] \land \text{arity c = 0}}) \lor (\exists x. t = \text{Var x})"
and "\xi = \text{Var } \implies (\exists c. t = \text{Fun c [] \land \text{arity c = 0}}) \lor (\exists x. t = \text{Var x})"
and "\xi = \text{Var } \implies \forall x \in \text{set (transaction_fresh T)}. \Gamma_{\sigma_{o_3}} x = TAtom Value \implies 
(\exists n. t = \text{Fun (Val n) []}) \lor (\exists x. t = \text{Var x})"
and "\xi = \text{Var } \implies \text{is_Fun } t \implies t \in \text{substs_range } \sigma"

proof -

obtain x where x: "x \in \text{substs_domain} (\xi_{\sigma_{o_3}} \circ_{o_3} \alpha) " "(\xi_{\sigma_{o_3}} \circ_{o_3} \alpha) x = t"
using t by auto

note 0 = x transaction_decl_fresh_renaming_substs_range[OF \xi \sigma, of x]
3.3 Stateful Protocol Model

show "(\exists c. t = Fun c []) \land \text{arity} c = 0 \lor (\exists x. t = \text{Var} x)"
using 0 unfolding \( \Gamma_v.\text{TAtom}'' \) by auto

assume 1: "\xi = \text{Var}"

note 2 = transaction_decl_subst_empty_inv[OF \xi[unfolded 1]]

show "(\exists n. t = \text{Fun} (\text{Val} n) []) \lor (\exists x. t = \text{Var} x)"
when "\forall x \in \text{set} (\text{transaction_fresh} T). \Gamma_v x = \text{TAtom Value}"
using 0 2 unfolding \( \Gamma_v.\text{TAtom}'' \) by auto

show "t \in \text{subst_range} \sigma" when t': "is_Fun t"

proof -
obtain x where x: "(\sigma \circ s \alpha) x = t" using t 1 by auto

show ?thesis
proof (cases "x \in \text{subst_domain} \sigma")
case True thus ?thesis
  by (metis subst_dom_vars_in_subst subst_ground_ident_compose(1) subst_imgI x
transaction_fresh_subst_grounds_domain[OF \sigma]
transaction_fresh_subst_domain[OF \sigma])
next
case False thus ?thesis
  by (metis (no_types, lifting) subst_compose_def subst.domI term.disc(1) that
transaction_renaming_subst_is_renaming(5)[OF \alpha] var_renaming_is_Fun_iff x)
qed
qed

lemma transaction_decl_fresh_renaming_substs_range'':
fixes \xi \sigma \alpha::"('fun,'atom,'sets,'lbl) prot_subst"
assumes \xi: "transaction_decl_subst \xi T"
and \sigma: "transaction_fresh_subst \sigma T A"
and \alpha: "transaction_renaming_subst \alpha P A"
and y: "y \in \text{fv} ((\xi \circ \sigma \circ \sigma) x)"
shows "\xi x = \text{Var} x"
  and "\sigma x = \text{Var} x"
  and "\alpha x = \text{Var} y"
  and "((\xi \circ \sigma \circ \sigma) x) = \text{Var} y"
proof -
  have "\exists z. z \in \text{fv} (\xi x)" by (metis y subst_compose_fv')
  hence "x \notin \text{subst_domain} \xi"
    using y transaction_decl_subst_domain[OF \xi]
    transaction_decl_subst_grounds_domain[OF \xi, of x]
    by blast
  hence 0: "\xi x = \text{Var} x" by blast
  hence "y \in \text{fv} ((\sigma \circ \alpha) x)" using y by (simp add: subst_compose)
  hence "\exists z. z \in \text{fv} (\sigma x)" by (metis subst_compose_fv')
  hence "x \notin \text{subst_domain} \sigma"
    using y transaction_fresh_subst_domain[OF \sigma]
    transaction_fresh_subst_grounds_domain[OF \sigma, of x]
    by blast
  hence 1: "\sigma x = \text{Var} x" by blast

show "\alpha x = \text{Var} y" "((\xi \circ \sigma \circ \sigma) x) = \text{Var} y"
  using 0 1 y transaction_renaming_subst_is_renaming(2)[OF \alpha, of x]
  unfolding subst_compose_def by (fastforce,fastforce)
qed

lemma transaction_decl_fresh_renaming_substs_vars_subset:
3 Stateful Protocol Verification

fixes \( \xi \), \( \sigma \)::"('fun', 'atom', 'sets', 'lbl') prot_subst"
assumes \( \xi \): "transaction_decl_subst \( \xi \) \( T \)"
and \( \sigma \): "transaction_fresh_subst \( \sigma \) \( T \) \( A \)"
and \( \alpha \): "transaction renaming subst \( \alpha \) \( P \) \( A \)"

shows \( \{ fvars \subseteq \text{subdomain} (\xi o_s \sigma o_s A) \} \) (is \( \text{?A} \))
and \( \{ fvars A \subseteq \text{subdomain} (\xi o_s \sigma o_s A) \} \) (is \( \text{?B} \))
and \( \{ T' \in \text{set P} \implies fvars T' \subseteq \text{subdomain} (\xi o_s \sigma o_s A) \} \) (is \( \text{?C1} \))
and \( \{ T' \in \text{set P} \implies fvars T' \subseteq \text{subdomain} (\xi o_s \sigma o_s A) \} \) (is \( \text{?C2} \))
and \( \{ \text{set P} \implies fvars T' \subseteq \text{range_vars} (\xi o_s \sigma o_s A) \} \) (is \( \text{?C3} \))
and \( \{ \text{set P} \implies \text{range_vars} (\xi o_s \sigma o_s A) = {} \} \) (is \( \text{?D1} \))
and \( \{ \text{set P} \implies \text{range_vars} (\xi o_s \sigma o_s A) = {} \} \) (is \( \text{?D2} \))

qed

lemma \( \text{transaction decl fresh renaming subs var disj} \):

fixes \( \xi \), \( \sigma \)::"('fun', 'atom', 'sets', 'lbl') prot_subst"
assumes \( \xi \): "transaction_decl_subst \( \xi \) \( T \)"
and \( \sigma \): "transaction_fresh_subst \( \sigma \) \( T \) \( A \)"
and \( \alpha \): "transaction renaming subst \( \alpha \) \( P \) \( A \)"

shows \( \{ \text{transaction_strand} T' \lnot \subseteq \text{unlabel} \{ \text{transaction strand} T' \} \} \) \( \subseteq \text{range_vars} (\xi o_s \sigma o_s A) \) (is \( \text{?B} \))
and \( \{ \text{transaction_strand} T' \lnot \subseteq \text{range_vars} (\xi o_s \sigma o_s A) \} \) (is \( \text{?C} \))
and \( \{ \text{transaction_strand} T' \lnot \subseteq \text{range_vars} (\xi o_s \sigma o_s A) \} \) (is \( \text{?D} \))

proof -
  have \( \text{?thesis} \) for \( \text{cases} \) "\( x \in \text{subdomain} \( \xi \) ""
  case True thus \( \text{?thesis} \)
  using \( \text{transaction decl subst domain}[OF \xi \] \( \text{transaction decl subst grounds domain}[OF \xi \] by (simp add: \( \text{subst domI subst dom vars in subst subst ground ident compose}(1) \))

next
case False
  hence \( \text{?thesis} \) unfolding \( \text{subst compose def} \) by \( \text{fastforce} \)

qed

show \( ?A \)?B using * by blast+

show \( ?C \) when \( \text{T} \): "\( T' \in \text{set P} \)" using \( \text{T} \) * by blast

thus \( \text{?D} \) when \( \text{T} \): "\( T' \in \text{set P} \)" by (metis \( \text{T unlabel subst} \))

qed


and \( f_{\text{vset}} \ A \cap \text{range_vars} \ (\xi \circ_o \sigma \circ_o \alpha) = {} \) (is \( ?D3 \))

**proof**

\[
\begin{align*}
\text{note } 0 & = \text{transaction_renaming_subst_vars_disj[OF } \alpha) \\
\text{show } ?A \\
\text{proof } (\text{cases } "f_{\text{vset}} ((\xi \circ_o \sigma \circ_o \alpha) \cup (\text{vars_transaction} \cup \text{set } P)) = {}") \\
\text{case False} \\
\text{hence } "x \in (\cup (\text{vars_transaction} \cup \text{set } P)). (\xi \circ_o \sigma \circ_o \alpha) x = \alpha x \lor f_{\text{v}} ((\xi \circ_o \sigma \circ_o \alpha) x) = {}" \\
\text{using transaction_decl_fresh_renaming_subsets_range' [OF } \xi \circ_o \sigma \circ_o \alpha) \text{ by auto} \\
\text{thus } ?\text{thesis using } 0(1) \text{ by force} \\
\text{qed blast} \\
\text{thus } "?B' = ?B" \text{ by auto} \\
\text{have } "\text{range_vars } \xi = {}" "\text{range_vars } \sigma = {}" \\
\text{using transaction_fresh_subst_grounds_domain[OF } \sigma \text{] transaction_decl_subst_grounds_domain[OF } \xi \text{] unfolding transaction_fresh_subst_domain[OF } \xi \text{, symmetric} \text{ transaction_decl_subst_domain[OF } \xi \text{, symmetric} \text{ by (fastforce, fastforce)} \\
\text{hence } 1: "\text{range_vars } (\xi \circ_o \sigma \circ_o \alpha) \subseteq \text{range_vars } \alpha" \\
\text{using range_vars_subst_compose_subset[of } \xi \circ_o \sigma \circ_o \alpha) \text{ by blast} \\
\text{show } ?C1 ?C2 ?C3 \text{ when } T: "T' \in \text{set } P" \text{ using } T 1 0(3,4,5)[of } T' \text{ by blast+} \\
\text{show } ?D1 ?D2 ?D3 \text{ using } 1 0(6,7,8) \text{ by blast+} \\
\text{qed}
\end{align*}
\]

**lemma** transaction_decl_fresh_renaming_substs_trms:

\[
\begin{align*}
\text{fixes } \xi \circ_o \sigma \circ_o \alpha & :="('fun,'atom,'sets,'lbl) prot_subst" \\
\text{assumes } \xi & :="\text{transaction_decl_subst } \xi T" \\
\text{and } \sigma & :="\text{transaction_fresh_subst } \sigma T A" \\
\text{and } \alpha & :="\text{transaction_renaming_subst } \alpha P A" \\
\text{and } \text{bvars}_{\text{lsst}} S \cap \text{subst_domain } \xi & = {} \\
\text{and } \text{bvars}_{\text{lsst}} S \cap \text{subst_domain } \sigma & = {} \\
\text{and } \text{bvars}_{\text{lsst}} S \cap \text{subst_domain } \alpha & = {} \\
\text{shows } "\text{subterms}_{\text{set}} (\text{trms}_{\text{lsst}} (S \cdot \text{lsst} ((\xi \circ_o \sigma \circ_o \alpha))) = \text{subterms}_{\text{set}} (\text{trms}_{\text{lsst}} S \cdot \text{lsst} ((\xi \circ_o \sigma \circ_o \alpha)))" \\
\text{proof } - \\
\text{have 1: } "x \in f_{\text{vset}} (\text{trms}_{\text{lsst}} S). (\exists f. (\xi \circ_o \sigma \circ_o \alpha) x = \text{Fun f (})) \lor (\exists y. (\xi \circ_o \sigma \circ_o \alpha) x = \text{Var y})" \\
\text{using transaction_decl_fresh_renaming_subsets_range'[OF } \xi \circ_o \sigma \circ_o \alpha) \text{ by blast} \\
\text{have 2: } "\text{bvars}_{\text{lsst}} S \cap \text{subst_domain } (\xi \circ_o \sigma \circ_o \alpha) = {}" \\
\text{using assms(4-6) subst_domain_compose[of } \xi \circ_o \sigma \circ_o \alpha) \text{ subst_domain_compose[of } \xi \circ_o \sigma \circ_o \alpha) \text{ by blast} \\
\text{show } ?\text{thesis using } \text{subterms}_{\text{subst}}_{\text{lsst}}[OF 1 2] \text{ by simp} \\
\text{qed}
\end{align*}
\]

**lemma** transaction_decl_fresh_renaming_substs_wt:

\[
\begin{align*}
\text{fixes } \xi \circ_o \sigma \circ_o \alpha & :="('fun,'atom,'sets,'lbl) prot_subst" \\
\text{assumes } \xi & :="\text{transaction_decl_subst } \xi T" "\text{transaction_fresh_subst } \sigma T A" "\text{transaction_renaming_subst } \alpha P A" \\
\text{shows } "\text{wt}_{\text{subst}} ((\xi \circ_o \sigma \circ_o \alpha))" \\
\text{using transaction_renaming_subst_wt[OF assms(3)] transaction_fresh_subst_wt[OF assms(2)] transaction_decl_subst_wt[OF assms(1)] by (metis wt_subst_compose) \\
\text{by (metis wt_subst_compose) \\
\text{lemma} transaction_decl_fresh_renaming_substs_range_wf_trms:
\text{fixes } \xi \circ_o \sigma \circ_o \alpha & :="('fun,'atom,'sets,'lbl) prot_subst" \\
\text{assumes } \xi & :="\text{transaction_decl_subst } \xi T" "\text{transaction_fresh_subst } \sigma T A" "\text{transaction_renaming_subst } \alpha P A" \\
\text{shows } "\text{wf}_{\text{trms}} \ (\text{subst_range } (\xi \circ_o \sigma \circ_o \alpha)))" \\
\text{proof } - \\
\text{have 1: } "x \in f_{\text{vset}} (\text{trms}_{\text{lsst}} S). (\exists f. (\xi \circ_o \sigma \circ_o \alpha) x = \text{Fun f (})) \lor (\exists y. (\xi \circ_o \sigma \circ_o \alpha) x = \text{Var y})" \\
\text{using transaction_decl_fresh_renaming_subsets_range'[OF } \xi \circ_o \sigma \circ_o \alpha) \text{ by blast} \\
\text{have 2: } "\text{bvars}_{\text{lsst}} S \cap \text{subst_domain } (\xi \circ_o \sigma \circ_o \alpha) = {}" \\
\text{using assms(4-6) subst_domain_compose[of } \xi \circ_o \sigma \circ_o \alpha) \text{ subst_domain_compose[of } \xi \circ_o \sigma \circ_o \alpha) \text{ by blast} \\
\text{show } ?\text{thesis using } \text{subterms}_{\text{subst}}_{\text{lsst}}[OF 1 2] \text{ by simp} \\
\text{qed}
\end{align*}
\]
3 Stateful Protocol Verification

using transaction_renaming_subst_range_wf_trms[OF assms(3)]
transaction_fresh_subst_range_wf_trms[OF assms(2)]
transaction_decl_subst_range_wf_trms[OF assms(1)]
wf_trms_subst_compose[of \(\xi\) \(\sigma\)]
wf_trms_subst_compose[of \(\xi \circ \sigma\) \(\alpha\)]
by metis

lemma transaction_decl_fresh_renaming_substs_fv:
fixes \(\sigma\) \(\alpha\) ::\("\fun,\'atom,\'sets,\'lbl\) prot_subst"
assumes \(\xi\): "transaction_decl_subst \(\xi\) \(T\)"
and \(\sigma\): "transaction_fresh_subst \(\sigma\) \(T\) \(A\)"
and \(\alpha\): "transaction_renaming_subst \(\alpha\) \(P\) \(A\)"
and \(x\): "\(x\) \(\in\) \(fv\) (\(\text{transaction_strand} \(T\) \cdot \(\text{lsst} \(\xi\) \circ \(s\) \(\sigma\) \circ \(s\) \(\alpha\)\))"
shows "\(\exists y \(\in\) \(fv\) \(\text{transaction}\) \(T\) - \(\text{set}\) \(\text{transaction_fresh}\) \(T\). \(\xi\) \(\circ\) \(s\) \(\sigma\) \(\circ\) \(s\) \(\alpha\) \(y\) = \(\text{Var}\) \(x\)"
proof
- have "\(x\) \(\in\) \(fv\) (\(\text{transaction_strand} \(T\) \cdot \(\text{lsst} \(\xi\) \circ \(s\) \(\sigma\) \circ \(s\) \(\alpha\)\))"
using \(x\) by auto
then obtain \(y\) where "\(y\) \(\in\) \(fv\) \(\text{transaction}\) \(T\)" "\(x\) \(\in\) \(fv\) \((\(\xi\) \(\circ\) \(s\) \(\sigma\) \(\circ\) \(s\) \(\alpha\)\) \(y\))"
by (metis fv_stmt_obtain_var)
thus \(?\)thesis
using transaction_decl_fresh_renaming_substs_range[OF \(\xi\) \(\sigma\) \(\alpha\), \(\text{of}\) \(y\)]
by (cases "\(y\) \(\in\) \(\text{set}\) \(\text{transaction_fresh}\) \(T\)") force+
qed

definition transaction_decl_fresh_renaming_substs_range_no_attack_const:
fixes \(\xi\) \(\sigma\) \(\alpha\) ::\("\fun,\'atom,\'sets,\'lbl\) prot_subst"
assumes \(\xi\): "transaction_decl_subst \(\xi\) \(T\)"
and \(\sigma\): "transaction_fresh_subst \(\sigma\) \(T\) \(A\)"
and \(\alpha\): "transaction_renaming_subst \(\alpha\) \(P\) \(A\)"
and \(T\): "\(\forall\) \(x\) \(\in\) \(\text{set}\) \(\text{transaction_fresh}\) \(T\). \(\Gamma\) \(\nu\) \(x\) = \(T\text{Atom Value}\) \(\lor\) \((\exists\ a. \(\Gamma\) \(\nu\) \(x\) = \(T\text{Atom (Atom} a)\))"
and \(t\): "\(t\) \(\in\) \(\text{subst_range}\) \((\(\xi\) \(\circ\) \(s\) \(\sigma\) \(\circ\) \(s\) \(\alpha\))\)"
shows "\(\nexists\ n. \(t\) = \text{attack} \(\langle\ n\ \rangle\)"
proof
- note \(\xi\sigma\alpha\)_wt = transaction_decl_fresh_renaming_substs_wt[OF \(\xi\) \(\sigma\) \(\alpha\)]
obtain \(x\) where "\(\xi\) \(\circ\) \(s\) \(\sigma\) \(\circ\) \(s\) \(\alpha\) \(x\) = \(t\)"
using \(t\) by auto
have \(\text{x_type}\): "\(\Gamma\ \text{Var} \(x\)\) = \(\Gamma\ \text{Var} \(x\) \cdot \(\xi\)\)" "\(\Gamma\ \text{Var} \(x\)\) = \(\Gamma\ \text{Var} \(x\) \cdot \(\xi\) \(\circ\) \(s\) \(\sigma\) \(\circ\) \(s\) \(\alpha\)\)"
using \(\xi\) wt_subst_trm''[of \(\text{Var} \(x\)\)] wt_subst_trm''[OF \(\xi\sigma\alpha\) _wt, of \(\text{Var} \(x\)\)]
unfolding transaction_decl_subst_def by (blast, blast)
show \(?\)thesis
proof (cases \(t\))
case (Fun \(f\) \(S\))
hence "\(x\) \(\in\) \(\text{set}\) \(\text{transaction_fresh}\) \(T\) \(\lor\) \(x\) \(\in\) \(\text{fst}\) \(\cdot\) \(\text{set}\) \(\text{transaction_decl}\) \(T\) ()\)"
using transaction_decl_fresh_renaming_substs_range[OF \(\xi\) \(\sigma\) \(\alpha\), \(\text{of}\) \(x\)] \(x\) by force
thus \(?\)thesis
proof
assume "\(x\) \(\in\) \(\text{set}\) \(\text{transaction_fresh}\) \(T\)"
hence "\(\Gamma\ t = \text{TAAtom Value} \(\lor\) \((\exists a. \(\Gamma\ t = \text{TAAtom (Atom} a)\))\)"
using \(T\) x_type(2) \(x\) by (metis \(\Gamma\) _consts_simps(1) subst_apply_term.simps(1))
thus \(?\)thesis by auto
next
assume "\(x\) \(\in\) \(\text{fst}\) \(\cdot\) \(\text{set}\) \(\text{transaction_decl}\) \(T\) ()\)"
then obtain \(c\) where \(\text{c where c: "\(\text{Var} \(x\) = Fun \(\text{Fun} \(c\)\) ()" \(\text{arity} \(f\) c = 0\)"
using \(\xi\) unfolding transaction_decl_subst_def by auto
have "\(\Gamma\ t = \text{TAAtom Bottom} \(\lor\) \((\exists a. \(\Gamma\ t = \text{TAAtom (Atom} a)\))\)"
using \(c\)1 \(\Gamma\) _consts_simps(1) \(\text{of}\) \(x\) \(\text{of}\) \(\text{Var} \(x\)\) subst_apply_term.simps(1) \(\text{of}\) \(x\) \(\text{of}\) \(\xi\) \(\sigma\) \(\alpha\)"
by (cases "\(\Gamma\) \(f\) \(c\)") simp_all
### 3.3 Stateful Protocol Model

Thus ?thesis by auto

qed

qed simp

qed

**Lemma** transaction_decl_fresh_renaming_substs_occurs_fact_send_receive:

fixes \( t :: ('fun, 'atom, 'sets, 'lbl) prot_term \)

assumes \( \xi : \text{transaction_decl_subst } T \)

and \( \sigma : \text{transaction_fresh_subst } \sigma T \)

and \( \alpha : \text{transaction_renaming_subst } \alpha P \)

and \( T : \text{admissible_transaction } T \)

and \( t : \text{occurs } t \in \text{set } ts \)

shows "send(\langle ts \rangle ) \in \text{set (unlabel (transaction_strand } T) ) \wedge \text{occurs } t = s \cdot \xi s \sigma s \alpha"

(is "?A \implies ?A'")

and "receive(\langle ts \rangle ) \in \text{set (unlabel (transaction_strand } T) ) \wedge \text{occurs } s \in \text{set ts}' \wedge t = s \cdot \xi s \sigma s \alpha"

(is "?B \implies ?B'")

proof -

assume ?A

then obtain \( s ts' \) where

\( s \in \text{set ts}' \) "send(\langle ts' \rangle ) \in \text{set (unlabel (transaction_strand } T) ) \wedge \text{occurs } t = s \cdot \xi s \sigma s \alpha"

using t stateful_strand_step_mem_substD(1)[of ts "unlabel (transaction_strand T)" "\xi s \sigma s \alpha"]

by auto

note \( \xi _\text{empty} = \text{admissible_transaction_decl_subst_empty}[OF T \xi] \)

have T_decl_notin: "x \notin \text{fst } \text{set (transaction_decl } T())" for x

using transaction_decl_subst_empty_inv[OF \( \xi _\text{empty}\) [unfolded \( \xi _\text{empty}\)]) by simp

note T_wf = admissible_transaction_is_wellformed_transaction(1)[OF T]

note T_fresh = admissible_transactionE(14)[OF T]

have "\exists u. s = \text{occurs } u"

proof (cases s)

case (Var x)

hence "\((\exists c. s \cdot \xi s o s a \cdot c = \text{Fun } c []) \lor (\exists y. s \cdot \xi s o s a \cdot y = \text{Var } y)\)"

using 0(2-5)[of x] \( \xi _\text{empty} \) by (auto simp del: subst_subst_compose)

thus ?thesis

using 0(1) by simp

next

case (Fun f T)

hence 1: "f = \text{OccursFact} "length T = 2" "T ! 0 \cdot \xi o s a = \text{Fun } \text{OccursSec } []"

"T ! 1 \cdot \xi o s a = t"

using 0(1) by auto

have "T ! 0 = \text{Fun } \text{OccursSec } []"

proof (cases "T ! 0")

case (Var x) thus ?thesis

using 0(2-5)[of x] 1(3) T_fresh T_decl_notin

unfolding list_all_iff by (auto simp del: subst_subst_compose)

qed (use 1(3) in simp)

thus ?thesis using Fun I 0(1) by (auto simp del: subst_subst_compose)

qed

then obtain u where u: "s = \text{occurs } u" by moura

hence "s = u \cdot \xi o s a" using s(3) by fastforce

thus ?A' using a u wellformed_transaction_strand_unlabel_memberD(8)[OF T_wf] by metis

next
assume \(?B\)
then obtain \(s\ ts'\) where \(s\):
  \("s \in \text{set ts}'\) "receive(ts') \in \text{set (unlabel (transaction_strand T))}"
"occurs t = s \cdot \xi o_s o_s o^\alpha"
using \(t\) \text{stateful_strand_step_mem_substD}(2)[
  of ts "unlabel (transaction_strand T)"
"\(\xi o_s o_s o^\alpha\)"
unlabel_subst[of "transaction_strand T" "\(\xi o_s o_s o^\alpha\)"
by auto

note \(\xi_{\text{empty}} = \text{admissible_transaction_decl_subst_empty[OF } T \xi)\]
have \(T\_\text{decl_notin}: "x \notin \text{fst } \text{set (transaction_decl T ())}" for \(x\)
using transaction_decl_subst_empty_inv[OF \(\xi\)\[unfolded \(\xi_{\text{empty}}\)]
by simp

note \(T\_\text{wf} = \text{admissible_transaction_is_wellformed_transaction}(1)[OF } T\]
note \(T\_\text{fresh} = \text{admissible_transactionE}(14)[OF } T\]

have \("\exists u. s = \text{occurs u}""
proof (cases \(s\))
  case (Var \(x\))
  hence \("(\exists c. s \cdot \xi o_s o_s o^\alpha = \text{Fun c } []) \lor (\exists y. s \cdot \xi o_s o_s o^\alpha = \text{Var y})""
using \(0(2-5)[of x]\) \(\xi_{\text{empty}}\) by (auto simp del: subst_subst_compose)
  thus \(?thesis\)
  using \(0(1)\) by simp
next
  case (Fun \(f T\))
  hence \("T \cdot 1 \cdot \xi o_s o_s o^\alpha = \text{Fun OccursSec } []""
using \(0(1)\) by auto
  have \("T \cdot 0 = \text{Fun OccursSec } []""
proof (cases "T \cdot 0")
    case (Var \(x\))
    thus \(?thesis\)
    using \(0(2-5)[of x]\) \(1(3)\) \(T\_\text{fresh} T\_\text{decl_notin}\)
    unfolding list_all_iff by (auto simp del: subst_subst_compose)
  qed (use \(1(3)\) in simp)
  thus \(?thesis\) using \(\text{Fun 1 0}(1)\) by (auto simp del: subst_subst_compose)
  qed
then obtain \(u\) where \(u: "s = \text{occurs u}"\) by moura
hence \("t = u \cdot \xi o_s o_s o^\alpha"\) using \(s(3)\) by fastforce
  thus \(?B'\) using \(s\ u\) wellformed_transaction_strand_unlabel_memberD(1)[OF } T\_\text{wf}\) by metis
  qed

lemma transaction_decl_subst_proj:
  assumes "transaction_decl_subst \(\xi\ T)"
  shows "transaction_decl_subst \(\xi\ (\text{transaction_proj n T})"
using assms transaction_proj_decl_eq[of \(n\) \(T\)]
unfolding transaction_decl_subst_def by presburger

lemma transaction_fresh_subst_proj:
  assumes "transaction_fresh_subst \(\sigma\ A)"
  shows "transaction_fresh_subst \(\sigma\ (\text{transaction_proj n T}) (\text{proj n A})"
using assms transaction_proj_fresh_eq[of \(n\) \(T\)]
contra_subsetD[OF subterms_set_mono[OF transaction_proj_trms_subset[of \(n\) \(T\)]]
contra_subsetD[OF subterms_set_mono[OF trms_set_proj_subset(1)[of \(n\) \(A\)]]
unfolding transaction_fresh_subst_def by metis

lemma transaction_renaming_subst_proj:
  assumes "transaction_renaming_subst \(\alpha\ P A)"
  shows "transaction_renaming_subst \(\alpha\ (\text{map (transaction_proj n) P}) (\text{proj n A})"
proof -
  let \(?X = "\lambda P. A. \bigcup (\text{vars_transaction } \set P) \cup \text{vars_set} A"
  define \(Y\) where \("Y \equiv ?X (\text{map (transaction_proj n) P}) (\text{proj n A})"

  ...
define Z where \( Z \equiv \exists Y \subseteq Z \)

using \texttt{sst\_vars\_proj\_subset}(3)[of n A] \texttt{transaction\_proj\_vars\_subset}[of n]

unfolding \texttt{Y\_def \_Z\_def} by \texttt{fastforce}

hence \( \exists n \geq \max\_\text{var\_set} Y. \alpha = \text{var\_rename} n \)

using \texttt{assms\_unfolding \_Y\_def \_Z\_def} by \texttt{blast}

thus \texttt{?thesis\_unfolding \_transaction\_renaming\_subst\_def \_Y\_def} by \texttt{blast}

qed
## 3.3.8 Lemmata: Reachable Constraints

**Lemma reachable_constraints_as_transaction_lists:**

**Fixes** \( f \)

**Defines** 

\[
\begin{align*}
\text{fixes} & \quad f & \equiv & \lambda (T, \xi, \sigma, \alpha). \ \text{dual}_{\text{lst}} (\text{transactionstrand} T \cdot \text{lst} \xi \circ_{\text{o}} \sigma \circ_{\text{o}} \alpha) \\
\text{and} & \quad g & \equiv & \text{concat} \circ \text{map} f
\end{align*}
\]

**Assumes** 

\( A \in \text{reachable_constraints}_P \)

**Obtains** \( Ts \) where 

\[
\begin{align*}
\text{\( A = g \ Ts \) and} & \quad \forall B. \ \text{prefix} \ B \ Ts \rightarrow g B \in \text{reachable_constraints}_P \\
\text{and} & \quad \forall B T \xi \sigma \alpha. \ \text{prefix}\ (B\emptyset(T, \xi, \sigma, \alpha)) \ Ts \rightarrow T \in \text{set}_P \land \text{transaction_decl_subst} \xi T \land \\
\text{transaction_fresh_subst} \sigma \ T \land \text{transaction_renaming_subst} \alpha P (g B)
\end{align*}
\]

**Proof**

- **let** \( ?P1 = \lambda A Ts. A = g \ Ts \)
  - **let** \( ?P2 = \lambda Ts. \forall B. \ \text{prefix} \ B \ Ts \rightarrow g B \in \text{reachable_constraints}_P \)
  - **let** \( ?P3 = \lambda A Ts. \forall B T \xi \sigma \alpha. \ \text{prefix}\ (B\emptyset(T, \xi, \sigma, \alpha)) \ Ts \rightarrow T \in \text{set}_P \land \text{transaction_decl_subst} \xi T \land \\
  \text{transaction_fresh_subst} \sigma \ T \land \text{transaction_renaming_subst} \alpha P (g B) \)

**Have** 

\[
\exists \ Ts. \ ?P1 A Ts \land \ ?P2 Ts \land \ ?P3 A Ts
\]

**Using** \( A \) **Proof (induction A rule: reachable_constraints.induct)**

**Case** init

- **have** 
  
  \[
  \begin{align*}
  \text{\( ?P1 \)} \ [] \ [] \ "\text{\( ?P2 \)} \ [] \ []\ " \text{unfolding} \ g \text{\_def} \ f \text{\_def} \ \text{by simp\_all} \\
  \text{thus} \ ?\text{case by blast}
  \end{align*}
  \]

**Next**

- **let** \( ?A' = \lambda A T \xi \sigma \alpha. \ \text{dual}_{\text{lst}} (\text{transactionstrand} T \cdot \text{lst} \xi \circ_{\text{o}} \sigma \circ_{\text{o}} \alpha) \)
  - **obtain** \( Ts \) where 
    \[
    \begin{align*}
    \text{\( A = g \ Ts \) and} & \quad \forall B T \xi \sigma \alpha. \ \text{prefix}\ (B\emptyset(T, \xi, \sigma, \alpha)) \ Ts \rightarrow T \in \text{set}_P \land \text{transaction_decl_subst} \xi T \land \\
    \text{transaction_fresh_subst} \sigma \ T \land \text{transaction_renaming_subst} \alpha P (g B) \\
    \end{align*}
    \]

**Have** 

\[
\begin{align*}
\text{\( ?P1 \ A' \)} \ (Ts\emptyset(T, \xi, \sigma, \alpha))
\end{align*}
\]

**Using** \( Ts\emptyset(1) \) **unfolding** \( g \text{\_def} \ f \text{\_def} \ \text{by simp} \)

**Have 2:** 

\[
\begin{align*}
\text{\( ?P2 \ (Ts\emptyset(T, \xi, \sigma, \alpha)) \)}
\end{align*}
\]

**Proof (intro allI impl)**

- **fix** \( B \) **assume** 
  
  \[
  \begin{align*}
  \text{\( \text{prefix} \ B \ (Ts\emptyset(T, \xi, \sigma, \alpha)) \)}
  \end{align*}
  \]

**Hence** 

\[
\begin{align*}
\text{\( \text{prefix} \ B \ Ts \lor B = Ts\emptyset(T, \xi, \sigma, \alpha) \)} \ \text{by fastforce}
\end{align*}
\]

**Thus** 

\[
\begin{align*}
\text{\( g B \in \text{reachable_constraints}_P \) and} & \quad \forall B T \xi \sigma \alpha. \ \text{prefix}\ (B\emptyset(T, \xi, \sigma, \alpha)) \ Ts \rightarrow T \in \text{set}_P \land \text{transaction_decl_subst} \xi T \land \\
\text{transaction_fresh_subst} \sigma \ T \land \text{transaction_renaming_subst} \alpha P (g B) \\
\end{align*}
\]

**Unfolding** \( g \text{\_def} \ f \text{\_def} \ \text{by auto} \)

**QED**

**Have 3:** 

\[
\begin{align*}
\text{\( ?P3 \ A' \)} \ (Ts\emptyset(T, \xi, \sigma, \alpha))
\end{align*}
\]

**Using** \( Ts\emptyset(1,3) \) **step.hyps(2-5) unfolding** \( g \text{\_def} \ f \text{\_def} \ \text{by auto} \)

**Show** ?case using 1 2 3 by blast

**QED**

**Lemma reachable_constraints_transaction_action_obtain:**

**Assumes** 

\( A \in \text{reachable_constraints}_P \) and 

\( a \in \text{set}_A \)

**Obtains** \( T b B \alpha \sigma \xi \)

where 

\[
\begin{align*}
\text{\( \text{prefix}\ (B\emptyset\text{dual}_{\text{lst}} (\text{transactionstrand} T \cdot \text{lst} \xi \circ_{\text{o}} \sigma \circ_{\text{o}} \alpha)) A' \) and} & \quad T \in \text{set}_P \land \text{transaction_decl_subst} \xi T \land \\
\text{transaction_fresh_subst} \sigma \ T \land \text{transaction_renaming_subst} \alpha P B' \\
\text{and} & \quad B \in \text{set}_P (\text{transactionstrand} T) \ "a = \text{dual}_{\text{lst}} b \cdot \text{lst} \xi \circ_{\text{o}} \sigma \circ_{\text{o}} \alpha" \ "\text{fst} \ a = \text{fst} \ b" \\
\end{align*}
\]

**Proof**

- **define** \( f \) where 
  
  \[
  \begin{align*}
  f & \equiv \lambda(T, \xi, \sigma, \alpha). (\text{fun, atom, sets, lbl}) \text{prot_subst}, \alpha) \\
  \text{dual}_{\text{lst}} (\text{transactionstrand} T \cdot \text{lst} \xi \circ_{\text{o}} \sigma \circ_{\text{o}} \alpha)
  \end{align*}
  \]

**Define** \( g \) where 

\[
\begin{align*}
\text{\( g \equiv \text{concat} \circ \text{map} f"}
\end{align*}
\]

**Obtain** \( Ts \) where 

\[
\begin{align*}
\text{Ts}
\end{align*}
\]
"A = g Ts" "∀ B. prefix B Ts → g B ∈ reachable_constraints P"
"∀ B T (T,ξ,σ,α) prefix (B@[T,ξ,σ,α]) Ts →
T ∈ set P ∧ transaction_decl_subst ξ T ∧
transaction_fresh_subst σ T (g B) ∧ transaction_renaming_subst α P (g B)"
using reachable_constraints_as_transaction_lists[OF A] unfolding g_def f_def by blast

obtain T α ξ σ where T: "(T,ξ,σ,α) ∈ set Ts" "a ∈ set (f (T,ξ,σ,α))"
using Ts(1) unfolding g_def by auto

obtain B where B: "prefix (B@[T,ξ,σ,α]) Ts" using T(1) by (meson prefix_triv_in_iff)

obtain b where b:
  "b ∈ set (transaction_strand T)" "a = dual_step b · (T,ξ,σ,α)" "fst a = fst b"
using T(2) dual_step_subst[of "transaction_strand T" "(ξ,σ,α)"]
  unfolding f_def by simp

have 0: "prefix (g B@[T,ξ,σ,α]) A" using concat_map_mono_prefix[OF B, of f] unfolding g_def Ts(1) by simp

have 1: "T ∈ set P" "transaction_decl_subst ξ T (g B)" "transaction_renaming_subst α P (g B)"
using B Ts(3) by (blast,blast,blast,blast)

show thesis using 0[unfolded f_def] that[OF _ 1 b] by fast

lemma reachable_constraints_unlabel_eq:
defines "transaction_unlabel_eq ≡ λ T1 T2.
  transaction_decl T1 = transaction_decl T2 ∧
  transaction_fresh T1 = transaction_fresh T2 ∧
  unlabel (transaction_receive T1) = unlabel (transaction_receive T2) ∧
  unlabel (transaction_checks T1) = unlabel (transaction_checks T2) ∧
  unlabel (transaction_updates T1) = unlabel (transaction_updates T2) ∧
  unlabel (transaction_send T1) = unlabel (transaction_send T2)"
assumes Peq: "list_all2 transaction_unlabel_eq P1 P2"
shows "unlabel ` reachable_constraints P1 = unlabel ` reachable_constraints P2" (is "?A = ?B")
proof (intro antisym subsetI)
  have "transaction_unlabel_eq T2 T1 = transaction_unlabel_eq T1 T2" for T1 T2
    unfolding transaction_unlabel_eq_def by argo
  hence Peq': "list_all2 transaction_unlabel_eq P1 P2" using Peq list_all2_sym by metis

have 0: "unlabel (transaction_strand T1) = unlabel (transaction_strand T2)"
  when "transaction_unlabel_eq T1 T2" for T1 T2
  using that unfolding transaction_unlabel_eq_def transaction_strand_def by force

have "vars_transaction T1 = vars_transaction T2" when "transaction_unlabel_eq T1 T2" for T1 T2
  using 0[OF that] by simp
  hence "vars_transaction (P1 # i) = vars_transaction (P2 # i)" when "i < length P1" for i
    using that Peq list_all2_conv_all_nth by blast
moreover have "length P1 = length P2" using Peq unfolding list_all2_iff by argo
ultimately have 1: "∪ (vars_transaction ` set P1) = ∪ (vars_transaction ` set P2)"

have 2:
  "transaction_decl_subst ξ T1 = transaction_decl_subst ξ T2" (is "?A1 = ?A2")
  "transaction_fresh_subst σ T1 A = transaction_fresh_subst σ T2 B" (is "?B1 = ?B2")
  "transaction_renaming_subst α P1 A = transaction_renaming_subst α P2 B" (is "?C1 = ?C2")
  "transaction_renaming_subst α P2 A = transaction_renaming_subst α P1 B" (is "?D1 = ?D2")
when "transaction_unlabel_eq T1 T2" "unlabel A = unlabel B"
for T1 T2::("fun,atom,sets,'lbl) prot_transaction"
and \( A \) \( B \) \( C \) 
\[
\begin{align*}
& \text{fun, atom, sets, lbl} \propto \text{prot_strand} \\
& \text{fun, atom, sets, lbl} \propto \text{prot_subst}
\end{align*}
\]

proof -

have \(*\): 
\[
\begin{align*}
& \text{transaction_decl \ T1} = \text{transaction_decl \ T2} \\
& \text{transaction_fresh \ T1} = \text{transaction_fresh \ T2} \\
& \text{trms_transaction \ T1} = \text{trms_transaction \ T2}
\end{align*}
\]

using that unfolding transaction_unlabel_eq_def transaction_strand_def by force+

show 
\[
\begin{align*}
& ?A1 \longrightarrow \ ?A2 \text{ using } \(*\) \text{ unfolding transaction_decl_subst_def by argo} \\
& ?B1 \longrightarrow \ ?B2 \text{ using that } \text{ unfolding transaction_fresh_subst_def by force} \\
& ?C1 \longrightarrow \ ?C2 \text{ using that } \text{ unfolding transaction_renaming_subst_def by metis} \\
& ?D1 \longrightarrow \ ?D2 \text{ using that } \text{ unfolding transaction_renaming_subst_def by metis}
\end{align*}
\]

qed

have 3: 
\[
\begin{align*}
& \text{unlabel} \left( \text{dual} \text{lsst} \left( \text{transaction_strand} \ T1 \cdot \text{lsst} \ \vartheta \right) \right) = \text{unlabel} \left( \text{dual} \text{lsst} \left( \text{transaction_strand} \ T2 \cdot \text{lsst} \ \vartheta \right) \right) \\
& \text{when } \text{transaction_unlabel_eq} \ T1 \ T2 \text{ for } \ T1 \ T2 \ \vartheta \\
\end{align*}
\]

using 0[of that] unlabel_subst[of _ \ \vartheta] dual \text{lsst}_\text{unlabel_cong} by metis

have \( \exists B \in \text{reachable_constraints} \ P2. \text{unlabel} \ A = \text{unlabel} \ B \)

when \( A \in \text{reachable_constraints} \ P2 \)

proof (induction \( A \) rule: reachable_constraints.induct)

obtain \( B \) where IH: 
\[
\begin{align*}
& B \in \text{reachable_constraints} \ P2 \\
& \text{unlabel} \ A = \text{unlabel} \ B
\end{align*}
\]

by (meson step.IH)

obtain \( T' \) where \( T' \in \text{set} \ P2 \) 
\[
\begin{align*}
& \text{transaction_unlabel_eq} \ T T' \\
& \text{using list_all2_in_set_ex[OF Peq step.hyps(2)] by auto}
\end{align*}
\]

show \( ?\text{case} \)

using 3[OF \( T' \), of \( \xi \circ \sigma \circ \sigma \circ \alpha \)] IH(2) reachable_constraints.step[OF IH(1) T'(1)]
\[
\begin{align*}
& \text{step.hyps}(3-5) \\
& \text{by (metis unlabel_append[of \( A \)] unlabel_append[of \( B \)]})
\end{align*}
\]

qed (simp add: unlabel_def)

thus \( A \in \ ?A \Longrightarrow A \in \ ?B \) for \( A \) by fast

have \( \exists B \in \text{reachable_constraints} \ P1. \text{unlabel} \ A = \text{unlabel} \ B \)

when \( A \in \text{reachable_constraints} \ P2 \)

proof (induction \( A \) rule: reachable_constraints.induct)

obtain \( B \) where IH: 
\[
\begin{align*}
& B \in \text{reachable_constraints} \ P1 \\
& \text{unlabel} \ A = \text{unlabel} \ B
\end{align*}
\]

by (meson step.IH)

obtain \( T' \) where \( T' \in \text{set} \ P1 \) 
\[
\begin{align*}
& \text{transaction_unlabel_eq} \ T T' \\
& \text{using list_all2_in_set_ex[OF Peq step.hyps(2)] by auto}
\end{align*}
\]

show \( ?\text{case} \)

using 3[OF \( T' \), of \( \xi \circ \sigma \circ \sigma \circ \alpha \)] IH(2) reachable_constraints.step[OF IH(1) T'(1)]
\[
\begin{align*}
& \text{step.hyps}(3-5) \\
& \text{by (metis unlabel_append[of \( A \)] unlabel_append[of \( B \)]})
\end{align*}
\]

qed (simp add: unlabel_def)

thus \( A \in \ ?B \Longrightarrow A \in \ ?A \) for \( A \) by fast

d as

lemma reachable_constraints_set_eq:

assumes \( \text{set} \ P1 = \text{set} \ P2 \)

shows \( \text{reachable_constraints} \ P1 = \text{reachable_constraints} \ P2 \) (is \( \isasymforall A \in \ ?A \ |
\isasymforall B \in \ ?B \))

proof (intro antisym subsetI)

note 0 = assms transaction_renaming_subst_set_eq[OF assms]

note 1 = reachable_constraints.intros

show \( \isasymforall A \in \ ?A \Longrightarrow A \in \ ?B \) for \( A \)

by (induct \( A \) rule: reachable_constraints.induct) (auto simp add: 0 intro: 1)
show "A ∈ ?B ⟷ A ∈ ?A" for A
by (induct A rule: reachable_constraints.induct) (auto simp add: 0 intro: 1)
qed

lemma reachable_constraints_set_subst:
  assumes "set P1 = set P2"
  and "Q (reachable_constraints P1)"
  shows "Q (reachable_constraints P2)"
by (rule subst[of _ _ Q, OF reachable_constraints_set_eq[OF assms(1)] assms(2)])

lemma reachable_constraints_wftrs:
  assumes "∀T ∈ set P. wftrs (trms_transaction T)"
  and "A ∈ reachable_constraints P"
  shows "wftrs (trmŝ_set A)"
using assms(2)
proof (induction A rule: reachable_constraints.induct)
  case (case T ξ σ α)
  have "wftrs (trms_transaction T)"
    using assms(1) step.hyps(2) by blast
  hence "wftrs (trms_transaction T' _set ξ σ o σ α)"
    using transaction_decl_fresh_renaming_subst_range_wf_trms[OF step.hyps(3-5)]
    by (metis wftrs_trms subst)
  hence "wftrs (trmŝ_set (transaction_strand T' _set ξ o σ o o α))"
    using wftrs_trms_set_subst[of "transaction_strand T' _set ξ o σ o α" by metis]
    by (metis trms_set_unlabel_dual setDate_eq by blast)
  thus ?case using step.IH unlabel_append[of A] trmŝ_set_append[of "unlabel A"] by auto
qed simp

lemma reachable_constraints_var_types_in_transactions:
  fixes A::"('fun,'atom,'sets,'lbl) prot_constr"
  assumes A: "A ∈ reachable_constraints P"
  and T: "∀T ∈ set P. ∀x ∈ set (transaction_fresh T).
    Γv x = TAtom Value ∨ (∃a. Γv x = TAtom (Atom a))"
  shows "Tv · fv̂_set A ⊆ (∪T ∈ set P. Γv · fv_transaction T)" (is "?A A")
  and "Tv · bvarŝ_set A ⊆ (∪T ∈ set P. Γv · bvars_transaction T)" (is "?B A")
  and "Tv · varŝ_set A ⊆ (∪T ∈ set P. Γv · vars_transaction T)" (is "?C A")
using A
proof (induction A rule: reachable_constraints.induct)
  case (case T ξ σ α)
  define T' where "T' ≡ dual̂_set (transaction_strand T _set ξ o σ o σ α)"

  note 2 = transaction_decl_fresh_renaming_subst_range'wftrms[OF step.hyps(3-5)]

  have 3: "∀t ∈ sub_range (ξ o σ o σ α). fv t = {x} ∨ (∃x. t = Var x)"
    using transaction_decl_fresh_renaming_subst_range'(1)[OF step.hyps(3-5)]
    by fastforce

  have "fv̂_set T' = fv̂_set (transaction_strand T _set ξ o σ o σ α)"
    "bvarŝ_set T' = bvarŝ_set (transaction_strand T _set ξ o σ o σ α)"
    "varŝ_set T' = varŝ_set (transaction_strand T _set ξ o σ o σ α)"
    unfolding T'_def
    by (metis fv̂_set_unlabel_dual setDate_eq,
      metis bvarŝ_set_unlabel_dual setDate_eq,
      metis varŝ_set_unlabel_dual setDate_eq)
  hence "Γv · Var · fv̂_set T' ⊆ Γv · Var · fv_transaction T"
    "Γv · bvarŝ_set T' = Γv · bvars_transaction T"
    "Γv · varŝ_set T' ⊆ Γv · vars_transaction T"
    using wt_subst_list_vars_type_subset[OF 2 3, of "transaction_strand T"]
    by argo*
  hence "Γv · fv̂_set T' ⊆ Γv · fv_transaction T"
    "Γv · bvarŝ_set T' = Γv · bvars_transaction T"
"Γ_v ` vars_{ts} T' ⊆ Γ_v ` vars_{transaction T}"
by (metis Γ_v_Var_image)+

hence 4: "Γ_v ` fv_{ts} T' ⊆ (∪ T ∈ set P. Γ_v ` fv_{ts} T')"
"Γ_v ` bvars_{ts} T' ⊆ (∪ T ∈ set P. Γ_v ` bvars_{ts} T')"
"Γ_v ` vars_{ts} T' ⊆ (∪ T ∈ set P. Γ_v ` vars_{ts} T)"

using step.hyps(2) by fast+

have 5: "Γ_v ` fv_{ts} (A @ T') = (Γ_v ` fv_{ts} A) ∪ (Γ_v ` fv_{ts} T')"
"Γ_v ` bvars_{ts} (A @ T') = (Γ_v ` bvars_{ts} A) ∪ (Γ_v ` bvars_{ts} T')"
"Γ_v ` vars_{ts} (A @ T') = (Γ_v ` vars_{ts} A) ∪ (Γ_v ` vars_{ts} T')"

using unlabel_append[of A T']
fv_{ts}_append[of "unlabel A" "unlabel T"]
bvars_{ts}_append[of "unlabel A" "unlabel T"]
vars_{ts}_append[of "unlabel A" "unlabel T"]

by auto

{ case 1 thus ?case
  using step.IH(1) 4(1) 5(1)
  unfolding T'_def by (simp del: subst_subst_compose fv_{ts} def)
}

{ case 2 thus ?case
  using step.IH(2) 4(2) 5(2)
  unfolding T'_def by (simp del: subst_subst_compose bvars_{ts} def)
}

{ case 3 thus ?case
  using step.IH(3) 4(3) 5(3)
  unfolding T'_def by (simp del: subst_subst_compose)
}

qed simp_all

lemma reachable_constraints_no_bvars:
  assumes A: "A ∈ reachable_constraints P"
  and P: "∀ S ∈ set P. admissible_transaction S"
  shows "bvars_{ts} A = {}"

proof (induction)
  case init
  then show ?case
  unfolding unlabel_def by auto
next
  case (step A T ξ σ α)
  then have "bvars_{ts} A = {}"
    by metis
  moreover
  have "bvars_{ts} (dual_{ts} (transaction_strand T_\xi_\sigma_\alpha)) = {}"
    using step by (metis bvars_{ts}_subst bvars_{ts}_unlabel_dual_{ts}_eq)
  ultimately
  show ?case
  using bvars_{ts}_append unlabel_append by (metis sup_bot.left_neutral)

qed

lemma reachable_constraints_fv_bvars_disj:
  fixes A::"('fun,'atom,'sets,'lbl) prot_constr"
  assumes A:="A ∈ reachable_constraints P"
  and P: "∀ S ∈ set P. admissible_transaction S"
  shows "fv_{ts} A ∩ bvars_{ts} A = {}"

proof -
  let ?X = "∪ T ∈ set P. bvars_{transaction T}"
  note 0 = admissible_transactions_fv_bvars_disj[of P]
  have 1: "bvars_{ts} A ⊆ ?X" using A_reach
lemma reachable_constraints_vars_TAtom_typed:
fixes A::"('fun,'atom,'sets,'lbl) prot_constr"
assumes A_reach: "A ∈ reachable_constraints P"
and P: "∀T ∈ set P. admissible_transaction T"
and x: "x ∈ vars_T" A"
shows "Γv x = TAtom Value ∨ (∃a. Γv x = TAtom (Atom a))"
proof -
have A_wftrs: "wftrs (trms|set A)"
  by (metis reachable_constraints_wftrs A_reach)
have T_adm: "admissible_transaction T" when "T ∈ set P" for T
  by (meson that Ball_set P)
have "∀T ∈ set P. ∀x ∈ set (transaction_fresh T). Γv x = TAtom Value"
  using protocol_transaction-vars_TAtom_typed(3) P by blast
hence *: "Γv ` vars_TAtom A ⊆ (∪ T ∈ set P. Γv ` vars_transaction T)"
"
using reachable_constraints_var_types_in_transactions[of A P, OF A_reach] by auto

have "\(\Gamma \cdot \vars_{lsst} A \subseteq T\text{Atom} \cdot \text{insert Value (range Atom)}\)"
  proof -
  have "\(\Gamma \cdot x = T\text{Atom Value} \lor (\exists a. \Gamma \cdot x = T\text{Atom (Atom a)})\)"
    when "\(T \in \text{set P}\)" "\(x \in \text{vars\_transaction T}\)" for \(T\ x\)
    using that protocol_transaction_vars_TAtom_typed(1)[of T] P
    admissible_transactionE(5)
    by blast
  hence "\(\bigcup T \in \text{set P}. \Gamma \cdot \vars_{\text{transaction T}} \subseteq T\text{Atom} \cdot \text{insert Value (range Atom)}\)"
    using P by blast
  thus "\(\Gamma \cdot \vars_{\text{lsst}} A \subseteq T\text{Atom} \cdot \text{insert Value (range Atom)}\)"
    using * by auto
  qed

thus \(?\thesis\) using \(x\) by auto
  qed

lemma reachable_constraints_vars_not_attack_typed:
  fixes \(A::('fun,'atom,'sets,'lbl)\ \text{prot\_constr}\)
  assumes \(A\_\text{reach}: \(A \in \text{reachable\_constraints P}\)\)
  and \(P:\ \forall T \in \text{set P.}\ \forall x \in \text{vars\_transaction T}.\ 
  \Gamma \cdot x = T\text{Atom Value} \lor (\exists a. \Gamma \cdot x = T\text{Atom (Atom a)})\)"
  and \(x: x \in \text{vars\_lsst A}\) shows "\(\neg\ T\text{Atom AttackType} \subseteq \Gamma \cdot x\)"
  using reachable_constraints_var_types_in_transactions(3)[OF \(A\_\text{reach}\) P(1)] P(2) x
  by fastforce

lemma reachable_constraints_Value_vars_are_fv:
  assumes \(A\_\text{reach}: \(A \in \text{reachable\_constraints P}\)\)
  and \(P:\ \forall T \in \text{set P.} \ \text{admissible\_transaction T}\)" and \(x: x \in \text{vars\_lsst A}\) and "\(\Gamma \cdot x = T\text{Atom Value}\)"
  shows "\(x \in \text{fv\_lsst A}\)"
  proof -
  have "\(\forall T \in \text{set P.} \text{bvars\_transaction T} = {}\)"
    using \(P\) admissible_transactionE(4) by metis
  hence \(A\_\text{no\_bvars}: \ (\text{bvars}\_lsst A) = {}\)"
  using reachable_constraints_no_bvars[OF \(A\_\text{reach}\)] by metis
  thus \(?\thesis\) using \(x\) vars\_lsst\_is\_fv\_lsst\_bvars\_set[of "unlabel A"] by blast
  qed

lemma reachable_constraints_subterms_subst:
  assumes \(A\_\text{reach}: \(A \in \text{reachable\_constraints P}\)\)
  and \(I::\text{welltyped\_constraint\_model I A}\)" and \(P:\ \forall T \in \text{set P.} \ \text{admissible\_transaction T}\)" shows "\(\text{subterms\_set (trms\_lsst (A \cdot \text{lsst } I))} = (\text{subterms\_set (trms\_lsst A)}) \cdot \text{set } I\)"
  proof -
  have \(A\_\text{wf\_trms}: \ (\text{wf\_trms (trms\_lsst A)})\)"
    by (metis reachable_constraints_wf_trms admissible_transactions_wf_trms \(P\) \(A\_\text{reach}\))
  from \(I\) have \(I':\ \text{welltyped\_constraint\_model I A}\)" using welltyped_constraint_model_prefix by auto
  have 1: "\(\forall x \in \text{fv\_set (trms\_lsst A)}. \ (\exists f. I \ x = \text{Fun } f \ []) \lor (\exists y. I \ x = \text{Var } y)\)"
    proof
      fix \(x\)
      assume xa: "\(x \in \text{fv\_set (trms\_lsst A)}\)"
      have "\(\exists f. I \ x = \text{Fun } f \ T\)"
        using I interpretation_grounds[of I "Var x"] unfolding welltyped_constraint_model_def constraint_model_def
        by (cases "I \ x") automated
      then obtain \(f \ T\) where \(fT\_p: \ (I \ x = \text{Fun } f \ T)\)
        by auto
      qed

88
hence \( \psi_{\text{trm}} \ (\text{Fun } f \ T) \)

using \( \bar{I} \)

unfolding welltyped_constraint_model_def constraint_model_def

using \( \psi_{\text{trm}} \) subst_rangeD

by metis

moreover

have \( \exists a. \Gamma, x = \text{TAtom } a \)

using \( \text{reachable_constraints_vars_TAtom_typed} \) \( A \) reach \( P \)

by blast

hence \( \exists a. \Gamma (\text{Var } x) = \text{TAtom } a \)

by simp

hence \( \exists a. \Gamma (\text{Fun } f \ T) = \text{TAtom } a \)

by (metis (no_types, opaque_lifting) I welltyped_constraint_model_def fT_p wt subst_def)

ultimately show \( (\exists f. \bar{I} x = \text{Fun } f) \lor (\exists y. \bar{I} x = \text{Var } y) \)

using TAtom_term_cases fT_p

by metis

qed

have \( \forall T \in \text{set } P. \text{bvars_transaction } T = \{\} \)

using \( \text{assms admissible_transactionE}(4) \) by metis

then have \( \text{bvars}_{\text{lsst}} A = \{\} \)

by auto

show ?thesis

using \( \text{subterms_subst last}[\text{OF_2}] \) \( I \)

by simp

qed

lemma reachable_constraints_val_funs_private':

fixes \( A::\tuple{\text{fun},\text{atom},\text{sets},\text{lbl}} \) prot_constr

assumes \( A \) reach: \( \text{"A } \in \text{reachable_constraints } P" \)

and \( P: \ "\forall T \in \text{set } P. \text{admissible_transaction_terms } T" \)

and \( f: \ f \in \bigcup \ (\text{funs_term } \text{ ` } \text{trms}_{\text{sst}} A) \)\)

shows \( \neg \text{is_PubConstValue } f\)

and \( \neg \text{is_Abs } f\)

proof -

have \( \neg \text{is_PubConstValue } f \land \neg \text{is_Abs } f\) using \( A \) reach \( f \)

proof (induction \( A \) rule: \( \text{reachable_constraints.induct} \))

case (step \( A \) \( T \) \( \xi \) \( \sigma \) \( \alpha \))

let \( \xi' = \text{"unlabel } (\text{transaction_strand } T) \cdot \text{sst } \xi \circ \sigma \circ \alpha \)\)

let \( \xi'' = \text{"transaction_strand } T \cdot \text{sst } \xi \circ \sigma \circ \alpha \)\)

note \( \xi\text{ empty } = \)

admissible_transaction_decl_subst_empty'[OF bspec[OF \( P(2) \) step.hyps(2)] step.hyps(3)]

have \( T: \ "\text{admissible_transaction_terms } T" \)

using \( P(1) \) step.hyps(2) by metis

have \( T\text{ fresh}: \ "\forall x \in \text{set } (\text{transaction_fresh } T). \text{fst } x = \text{TAtom Value}" \) when \( "T \in \text{set } P" \) for \( T \)

using \( P \) that \( \text{admissible_transactionE}(14) \) unfolding list_all_iff \( \text{Gamma}_{\text{TAtom'}} \) by fast

show ?thesis using step

proof (cases \( f \in \bigcup \ (\text{funs_term } \text{ ` } \text{trms}_{\text{sst}} A) \))

case False

then obtain \( t \) where \( t: \ "t \in \text{trms}_{\text{sst}} \ ?T'' \" \) \( f \in \text{funs_term } t\)

using step.prems \( \text{trms}_{\text{sst}} \cdot \text{unlabel_dual}_{\text{sst}} \cdot \text{eq}[of } ?T''\)

\( \text{trms}_{\text{sst}} \cdot \text{append}[of } \text{"unlabel } A \) \text{ "unlabel } (\text{dual}_{\text{sst}} \ ?T'')\)\]

\( \text{unlabel_append}[of } A \) \text{ "dual}_{\text{sst}} \ ?T'')\] \text{unlabel_subst}[of "\text{transaction_strand } T"]
by fastforce
show ?thesis using trms, funs_term_cases[OF t]
proof
  assume "∃ u ∈ trms_transaction T. f ∈ funs_term u"
  thus ?thesis
    using conjunct1[OF conjunct2[OF T[unfolded admissible_transaction_terms_def]]]
next
  assume "∃ x ∈ fv_transaction T. f ∈ funs_term ((ξ ◦ s ◦ s ◦ α) x)"
  then obtain x where "x ∈ fv_transaction T" "f ∈ funs_term ((ξ ◦ s ◦ s ◦ α) x)
    unfolding is_PubConstValue_def by blast
qed
lemmas reachable_constraints_val_funs_private = reachable_constraints_val_funs_private'
lemmas reachable_constraints_occurs_fact_ik_case = reachable_constraints_occurs_fact_ik_case'
proof (induction A rule: reachable_constraints.induct)
  case (step A T ξ σ α)
  define ϑ where "ϑ ≡ ξ ◦ s ◦ s ◦ α"
  have T_adm: "admissible_transaction T" using P step.hyps(2) by blast
  hence T: "wellformed_transaction T" "admissible_transaction_occurs_checks T"
    using admissible_transaction_is_wellformed_transaction(1,5) by (blast,fast)
  have T_fresh: "∀ x ∈ set (transaction_fresh T). fst x = TAtom Value" using admissible_transactionE(14)[OF T_adm] unfolding list_all_iff by blast
  note ξ_empty = admissible_transaction_decl1_subst_empty[OF T_adm step.hyps(3)]
  have ξ_dom_empty: "z \notin fst ` set (transaction_decl T (\{\})" for z
    using transaction_decl1_subst_empty_inv[OF step.hyps(3)[unfolded ξ_empty]] by simp
  show ?case
  proof (cases "occurs t ∈ ik(lsst A)"
    case False
      hence "occurs t ∈ ik(lsst (transaction_strand T (\{\}))" using step.prems unfolding ϑ_def by simp
hence \[\exists t \in \text{set ts} \land \text{receive}(ts) \subseteq \text{set (unlabel (\text{transaction_strand} T \cdot \text{lsst} \theta)))}\]

unfolding \text{lsst_def} by force

hence \[\exists t \in \text{set ts} \land \text{send}(ts) \subseteq \text{set (unlabel (\text{transaction_strand} T \cdot \text{lsst} \theta)))}\]

using \text{dual_unlabel_steps_iff(1)} by blast

then obtain ts s where s: \[s \in \text{set ts} \land \text{send}(ts) \subseteq \text{set (unlabel (\text{transaction_strand} T \cdot \text{lsst} \theta)))}\]

by force

note 0 = \text{transaction_decl_fresh_renaming_substs_range[OF step.hyps(3-5)]}

have 1: \[\text{send}(ts) \subseteq \text{set (unlabel (\text{transaction_send} T)))}\]

using s(2) \text{wellformed_transaction_strand_unlabel_memberD(8)[OF T(1)]} by blast

have 2: \[\text{is_Send (send(ts))}\]

unfolding \text{is_Send_def} by simp

have 3: \[\exists u. s = \text{occurs u}\]

proof -

{ fix z

have \[\exists n. \theta z = \text{Fun (Val n)} \lor \exists y. \theta z = \text{Var y}\]

using 0(3,4) \text{T_fresh} unfolding \text{val_def} by blast

hence \[\exists u. \theta z = \text{occurs u}\]

\text{\theta z \neq \text{Fun OccursSec} []}\]

by auto

} note * = this

obtain u u' where T: \[s = \text{Fun OccursFact [u,u']}\]

using *(1) s(3) by (cases s) auto

thus ?thesis using *(2) s(3) by (cases u) auto

qed

obtain x where x: \[x \in \text{set (transaction_fresh T)}\]

using 3 s(1) \text{admissible_transaction_occurs_checksE4[OF T(2) 1]} by metis

have \[t = \theta x\]

using s(3) x(2) by auto

thus ?thesis using 0(3)[OF \text{T_fresh} unfolding \text{val_def} by fast

qed (simp add: step.IH)

qed simp

lemma reachable_constraints_occurs_fact_send_ex:

fixes A::"('fun,'atom,'sets,'lbl) prot_constr"

assumes A: \[A \in \text{reachable_constraints P}\]

and P: \[\forall T \in \text{set P}. \text{admissible_transaction T}\]

and x: \[\exists u. x = \text{TAtom Value}\]

shows \[\exists t \in \text{set ts} \land \text{send}(ts) \subseteq \text{set (unlabel A)}\]

using A_reach x(2)

proof (induction A rule: reachable_constraints.induct)

case (step A T l x a)

note \_empty = \text{admissible_transaction_decl_subst_empty[OF bspec[OF P step.hyps(2)] step.hyps(3)]}

have T: \[\text{admissible_transaction_occurs_checks T}\]

using P step.hyps(2) \text{admissible_transaction_is_wellformed_transaction(5)} by blast

show ?case

proof (cases "x \in \text{fv}_{lsst} A")

case True

show ?thesis

using step.IH[OF True] \text{unlabel_append[of A]} by auto
3 Stateful Protocol Verification

next
  case False
  then obtain y where y:
    "y ∈ fv_transaction T - set (transaction_fresh T)" "(ξ o₅ σ o₅ α) y = Var x"
  using transaction_decl_fresh_renaming_substs_fv[of step.hyps(3-5), of x]
  step.prems(1) fvs_{type, append}(of "unlabel A") unlabeled_append[of A]
  by auto

have "σ y = Var y" using y(1) step.hyps(4) unfolding transaction_fresh_subst_def by auto
hence "ο y = Var x" using (2) unfolding subst_context_def _empty by simp
hence y.val: "fst y = TAtom Value" "Γₜ y = TAtom Value"
  using x(1) Γₜ "unlabel"[of x] Γₜ "unlabel"[of y]
  wt_subst_trm'[OF transaction_renaming_subst_wt[OF step.hyps(5)], of "Var y"]
  by force+

obtain ts where ts:
  "occurs (Var y) ∈ set ts" "receive(ts) ∈ set (unlabel (transaction_receive T))"
  using admissible_transaction_occurs_checksEl[OF T y(1) y.val(2)]
  by (metis list.set_intros(1) unlabeled.Cons(1))

hence "receive(ts) ∈ set (unlabel (transaction_strand T))"
  using transaction_strand_subsets(5) by blast

hence #: "receive(ts \is_{\is\is} ξ o₅ σ o₅ α) ∈ set (unlabel (transaction_strand T \is_{\is\is} ξ o₅ σ o₅ α))"
  using unlabeled_subst[of "transaction_strand T" "ξ o₅ σ o₅ α"
  stateful_strand_step_subst_inI(2)[of _ _ "ξ o₅ σ o₅ α"]
  by force

have "occurs (Var y) · ξ o₅ σ o₅ α = occurs (Var x)"
  using y(2) by (auto simp del: subst_subst_compose)

hence #: "occurs (Var y) ∈ set ts · set ts subst-compose" "occurs (Var y)
  using ts(1) by force

have "send(ts \is_{\is\is} ξ o₅ σ o₅ α) ∈ set (unlabel (dual\is_{\is\is} (transaction_strand T \is_{\is\is} ξ o₅ σ o₅ α)))"
  using * dual\is_{\is\is} unlabeled_steps_iff(2) by blast
thus ?thesis using #: unlabeled_append[of A] by force
qed simp

lemma reachable_constraints_db_set_args_empty:
  assumes A: "A ∈ reachable_constraints P"
  and PP: "list_all wellformed_transaction P"
  and f: "let f = (λT. ∀x ∈ set (unlabel (transaction_updates T)).
  is_Update x ∧ is_VAR (the_elem_term x) ∧ is_FUN_Set (the_set_term x) ∧
  fst (the_VAR (the_elem_term x)) = TAtom Value)
  in list_all f P"
  and d: "(t, s) ∈ set (db_set (A I))"
  shows "∃ss. ss = Fun (Set ss) []"
  using A d
proof (induction)
  case (step A TT ξ σ α)
  let ?TT = "transaction_strand TT \is_{\is\is} ξ o₅ σ o₅ α"
  let ?TTu = "unlabel ?TT"
  let ?TTd = "dual\is_{\is\is} ?TT"
  let ?TTdu = "unlabel ?TTd"

  from step(6) have "(t, s) ∈ set (db_set ?TTd I (db_set (unlabel A) I []))"
    by (metis db_set_append db_set_def step.prems unlabeled_append)
  hence "(t, s) ∈ set (db_set (unlabel A) I []) ∨
  (∃t' s'. insert(t',s') ∈ set ?TTd ∧ t = t' · I ∧ s = s' · I)"
    using db_set_in_cases[of t "s" ?TTd I] by metis
  thus ?case
proof
  assume "∃t' s'. insert(t',s') ∈ set ?TTd ∧ t = t' · I ∧ s = s' · I"
  then obtain t' s' where t's': "insert(t',s') ∈ set ?TTd" "t = t' · I" "s = s' · I" by metis

  next
  case False
  then obtain y where y:
    "y ∈ fv_transaction T - set (transaction_fresh T)" "(ξ o₅ σ o₅ α) y = Var x"
  using transaction_decl_fresh_renaming_substs_fv[of step.hyps(3-5), of x]
  step.prems(1) fvs_{type, append}(of "unlabel A") unlabeled_append[of A]
  by auto

  have "σ y = Var y" using y(1) step.hyps(4) unfolding transaction_fresh_subst_def by auto
  hence "ο y = Var x" using (2) unfolding subst_context_def _empty by simp
  hence y.val: "fst y = TAtom Value" "Γₜ y = TAtom Value"
    using x(1) Γₜ "unlabel"[of x] Γₜ "unlabel"[of y]
    wt_subst_trm'[OF transaction_renaming_subst_wt[OF step.hyps(5)], of "Var y"]
    by force+

  obtain ts where ts:
    "occurs (Var y) ∈ set ts" "receive(ts) ∈ set (unlabel (transaction_receive T))"
    using admissible_transaction_occurs_checksEl[OF T y(1) y.val(2)]
    by (metis list.set_intros(1) unlabeled.Cons(1))

  hence "receive(ts) ∈ set (unlabel (transaction_strand T))"
    using transaction_strand_subsets(5) by blast

  hence #: "receive(ts \is_{\is\is} ξ o₅ σ o₅ α) ∈ set (unlabel (transaction_strand T \is_{\is\is} ξ o₅ σ o₅ α))"
    using unlabeled_subst[of "transaction_strand T" "ξ o₅ σ o₅ α"
    stateful_strand_step_subst_inI(2)[of _ _ "ξ o₅ σ o₅ α"]
    by force

  have "occurs (Var y) · ξ o₅ σ o₅ α = occurs (Var x)"
    using y(2) by (auto simp del: subst_subst_compose)

  hence #: "occurs (Var y) ∈ set ts · set ts subst-compose" "occurs (Var y)
    using ts(1) by force

  have "send(ts \is_{\is\is} ξ o₅ σ o₅ α) ∈ set (unlabel (dual\is_{\is\is} (transaction_strand T \is_{\is\is} ξ o₅ σ o₅ α)))"
    using * dual\is_{\is\is} unlabeled_steps_iff(2) by blast
  thus ?thesis using #: unlabeled_append[of A] by force
qed simp


then obtain $l$ where \((l, \text{insert}(t',s'))\) $\in$ set $\sigma_T$ by (meson unlabel_mem_has_label)

hence \("(l, \text{insert}(t',s'))\) $\in$ set (transaction_strand $T'$ $\times_{\sigma_T}$ $\xi$ $\alpha$ $\sigma$ $\alpha$)

using dual$_{\sigma_T}$steps_iff(4) by blast

hence \("\text{insert}(t',s')\) $\in$ set $\sigma_T$ by (meson unlabel_in)

hence \("\text{insert}(t',s')\) $\in$ set ((unlabel (transaction_strand $T'$)) $\times_{\sigma_T}$ $\xi$ $\alpha$ $\sigma$ $\alpha$)

by (simp add: subset_list_unlabel)

hence \("\text{insert}(t',s')\) $\in$ (\(\lambda x. x \times_{\sigma_T} \xi\) $\sigma$ $\alpha$ $\sigma$ $\alpha$) $\cdot$ set (unlabel (transaction_strand $T'$))" unfolding subapply_stateful_strand_def by auto

then obtain $u$ where

\(u \in\) set (unlabel (transaction_strand $T'$)) $\wedge\ u \times_{\sigma_T} \xi\) $\sigma$ $\alpha$ $\sigma$ $\alpha$ $=$ insert\(\text{insert}(t',s')\)

by auto

hence \("\exists t'' s'. \text{insert}(t'',s')\) $\in$ set (unlabel (transaction_strand $T'$)) $\wedge$

\(t' = t'' \cdot \xi\) $\sigma$ $\alpha$ $\wedge\ s' = s'' \cdot \xi\) $\sigma$ $\alpha$ $\alpha$

by auto

hence \("\text{insert}(t'',s')\) $\in$ set (unlabel (transaction_updates $T'$))" using is_Update_in_transaction_updates[of "\text{insert}(t'',s')" $T'$]

using PP step(2) unfolding list_all_iff by auto

moreover have \("\forall x\in\text{set}\ (\text{unlabel (transaction_updates $T'$)})\). \text{is_Fun_Set (the_set_term $x$)}\)" using step(2) admissible_transaction_updates unfolding is_Fun_Set_def list_all_iff by auto

ultimately have \("\text{is_Fun_Set (the_set_term (\text{insert}(t'',s')))}\)" by auto

moreover have \("s' = s'' \cdot \xi\) $\sigma$ $\alpha$ $\sigma$ $\alpha$ using \(t''\) s' p by blast

ultimately have \("\text{is_Fun_Set (\text{the_set_term (\text{insert}(t',s'))})}\)" by (auto simp add: is_Fun_Set_subst)

hence \("\text{is_Fun_Set (\text{set (unlabel (transaction_updates $T'$)))}}\)" by simp add: t's_p(3) is_Fun_Set_subst

thus ?case using is_Fun_Set_exi by auto

qed (auto simp add: step db$_{\sigma_T}$def)

qed auto

lemma reachable_constraints_occurs_fact_ik_ground:

fixes $A\::\:\text{('fun,,'atom,'sets,'lbl) prot_constr}$

assumes $A\text{-reach}: A \in \text{reachable_constraints } P$

and $P:\ \"\forall T \in \text{set } P. \text{admissible_transaction } T"$

and $t:\ \"\text{occurs } t \in \text{ik}_{\sigma\alpha} A\"$

shows \("\forall v\ (\text{occurs } t) = \{\}\)"

proof -

have $0: \"\text{admissible_transaction } T"$

when $T \in \text{set } P$ for $T$

using $P$ that unfolding list_all_iff by simp

note $l = \text{admissible_transaction_is_wellformed_transaction(1,5)[OF 0]}

have $2: \"\text{ik}_{\sigma\alpha} (A_{\text{dual}_{\sigma\alpha}} (\text{transaction_strand } T \times_{\sigma_T} \emptyset)) = (\text{ik}_{\sigma\alpha} A) \cup (\text{trms}_{\sigma\alpha} (\text{transaction_send } T \times_{\sigma_T} \emptyset))\"$

when $T \in \text{set } P$ for $T$ $\emptyset$ and $A:\:"\text{('fun,,'atom,'sets,'lbl) prot_constr}\"$

using dual_transaction_ik_is_transaction_send'[OF 1(1)[OF that]] by fastforce

show ?thesis using A_reach t

proof (induction $A$ rule: reachable_constraints.induct)

case (step $A\ T\ \xi\ \sigma\ \alpha$

note $\xi\text{-empty} = \text{admissible_transaction_dec1_subst_empty}[OF 0][OF step.hyps(2)] step.hyps(3])

from step show ?case

proof (cases \("\text{occurs } t \in \text{ik}_{\sigma\alpha} A'\"$

case False

hence \("\text{occurs } t \in \text{trms}_{\sigma\alpha} (\text{transaction_send } T \times_{\sigma_T} \xi\) \alpha\) \alpha\) $\alpha$ $\alpha$ $\alpha$ $\alpha$

using 2[OF step.hyps(2)] step.prems $\xi\text{-empty}$ by blast

then obtain $ts$ where $ts$:

\("\text{occurs } t \in \text{set } ts\" \text{send}(ts) \in \text{set (unlabel (transaction_send } T \times_{\sigma_T} \xi\) \alpha\) \alpha\) $\alpha$ $\alpha$ $\alpha$

using wellformed_transaction_send_receive_subst_trm_cases(2)[OF 1(1)[OF step.hyps(2)]] by blast

93
then obtain $ts'$ where $s$:

"occurs $s \in set ts'$"  "send($ts'$) $\in set (unlabel (transaction_send T))"

using $\text{transaction_decl_fresh_renaming_substs_occurs_fact_send_receive}(1)[$

$\text{OF step.hyps(3-5) OF step.hyps(2) ts(1)}$

$\text{transaction_strand_subst_subsets(8) [of T] "(ξ o σ o α "}$]

by blast

obtain $x$ where $x$:

"x $\in set (transaction_fresh T)"  "s = Var x"

using $\text{admissible_transaction_occurs_checksE}[$

$\text{OF step.hyps(2) s(2,1)}$

$\text{by metis}$

have "$fv t = {}$"

using $\text{transaction_decl_fresh_renaming_substs_range(2) [of T] "(ξ o σ o α "}$]

by (auto simp del: subst_subst_compose)

thus $\text{thesis}$ by simp

qed simp

qed

lemma reachable_constraints_occurs_fact_ik_funs_terms:

fixes $A$::"('fun,'atom,'sets,'lbl) prot_constr"

assumes $A$ _reach: "$A$ $\in$ reachable_constraints $P$

and $I$ : "welltyped_constraint_model I $A$"

and $P$ : "$\forall T \in set P. admissible_transaction T"

shows "$s \in subterms set (ik lsst A · set I). OccursFact $\notin\bigcup (funs_term · set (snd (Ana s)))" (is "$?A A")

and "$s \in subterms set (ik lsst A · set I). OccursSec $\notin\bigcup (funs_term · set (snd (Ana s)))" (is "$?B A")

and "Fun OccursSec [] $\notin ik lsst A · set I" (is "$?C A")

and "$x \in vars lsst A. I x $\neq$ Fun OccursSec []" (is "$?D A")

proof -

have $T_{adm}$: "admissible_transaction $T$" when "$T \in set P" for $T$

using $P$ that unfolding list_all_iff by simp

note $T_{wf}$ = $\text{admissible_transaction_is_wellformed_transaction}(1)$[$OF $T_{adm}$]

note $T_{occ}$ = $\text{admissible_transaction_is_wellformed_transaction}(5)$[$OF $T_{adm}$]

note $ξ _empty$ = $\text{admissible_transaction_decl_subst_empty}$[$OF $T_{adm}$]

have $I_{wt}$: "w subst I" by (metis $I$ welltyped_constraint_model_def)

have $I_{trms}$: "$w_{trms}$ (subst_range $I$)"

by (metis $I$ welltyped_constraint_model_def subst_range_def)

have $I_{grounds}$: "$fv (I x) = {}" "$f T. I x = Fun f T" for $x$

using $I$ interpretation_grounds[$of I, of "Var x"] empty_fv_exists_fun[$of "I x"]

unfolding welltyped_constraint_model_def subst_range_def by auto

have $00$: "$fv set (trms lsst (transaction_send T)) \subseteq vars_fv transaction T$"

"fv set (subterms set (trms lsst (transaction_send T))) = fv set (trms lsst (transaction_send T))"

for $T$: "("fun,atom,'sets,,'lbl) prot_transaction"

using $fv_{trms lsst}$[subst(1)][of "unlabel (transaction_send T)"] vars_transaction_unfold

$fv_{subterms set}$[of "trms lsst (transaction_send T)"]

by blast+

have $0$: "$\forall x \in fv set (trms lsst (transaction_send T)). \exists a. Γ (Var x) = TAtom a$"

"$\forall x \in fv set (trms lsst (transaction_send T)). Γ (Var x) ≠ TAtom OccursSecType"*

"$\forall x \in fv set (subterms set (trms lsst (transaction_send T))). \exists a. Γ (Var x) = TAtom a$"

"$\forall x \in fv set (subterms set (trms lsst (transaction_send T))). Γ (Var x) ≠ TAtom OccursSecType"*

"$\forall x \in vars_transaction T. \exists a. Γ (Var x) = TAtom a$"

"$\forall x \in vars_transaction T. Γ (Var x) ≠ TAtom OccursSecType"*

when "$T \in set P$" for $T$

using $\text{admissible_transaction_occurs_fv_types}$[$OF T_{adm} [OF that]] $00$

by blast+
note \(T_{\text{fresh\_type}} = \text{admissible\_transactionE}(2)[\text{OF } T_{\text{adm}}]\)

have 1: "ilk_{\text{aiset}} (A&\text{dual}_{\text{aiset}} (\text{transaction\_strand } T \ '\text{aiset } \emptyset)) \ \text{set } I = \ (ilk_{\text{aiset}} A \ '\text{aiset } I) \cup (\text{trms}_{\text{aiset}} (\text{transaction\_send } T \ '\text{aiset } \emptyset \ '\text{aiset } I))"
when \(T \in \text{set } P\) for \(T \emptyset\) and \(A::("\text{fun},\text{atom},\text{sets},'\text{lb1}) \ \text{prot\_constr}\"
using dual\_transaction_ik_is_transaction_send'[OF T_wf[OF that]]
by fastforce

have 2: "subterms\_set (trms_{\text{aiset}} (\text{transaction\_send } T) \ '\text{aiset } \emptyset '\text{aiset } I) = \ subterms\_set (\text{trms}_{\text{aiset}} (\text{transaction\_send } T) \ '\text{aiset } \emptyset '\text{aiset } I)"
when \(T \in \text{set } P\) and \(\emptyset::"\text{wt\_subst }\emptyset" \ "\text{wf\_trms\_subst} (\text{subst\_range } \emptyset)"\) for \(T \emptyset\)
using wt\_subst\_TAtom\_subterms\_set\_subst[OF wt\_subst\_compose[OF \(\emptyset\)_\text{wt}] 0(1)[OF that(1)]]
fw\_trms\_subst\_rangeD[OF fw\_trms\_subst\_compose[OF \(\emptyset\)_\text{wf}\_trms]]
by auto

have 3: "\text{wt\_subst} (\xi \circ s \circ \sigma \circ \alpha)" "\text{wf\_trms\_subst\_range } (\xi \circ s \circ \sigma \circ \alpha)"
when \(T \in \text{set } P\) "\text{transaction\_decl\_subst } \xi T" 
"\text{transaction\_fresh\_subst } \sigma T A" "\text{transaction\_renaming\_subst } \alpha P A"
for \(\xi \sigma \alpha\) and \(T::"(\text{fun},\text{atom},\text{sets},'\text{lb1}) \ \text{prot\_transaction}\"
and \(A::("\text{fun},\text{atom},\text{sets},'\text{lb1}) \ \text{prot\_constr}\"
using protocol\_transaction\_vars\_TAtom\_typed(3)[of T] P that(1)
transaction\_decl\_fresh\_renaming\_subsists\_wt[OF that(2-4)]
transaction\_decl\_fresh\_renaming\_subsists\_range\_wf\_trms[OF that(2-4)]
fw\_trms\_subst\_compose
by simp_all

have 4: "\forall s \in \text{subterms\_set} (\text{trms}_{\text{aiset}} (\text{transaction\_send } T)).
\text{OccursFact} \notin \bigcup (\text{funs\_term } \set (\text{snd } (\text{Ana } s))) \land
\text{OccursSec} \notin \bigcup (\text{funs\_term } \set (\text{snd } (\text{Ana } s)))"
when \(T \in \text{set } P\) for \(T\)
proof
fix \(t\) assume \(t::"t \in \text{subterms\_set} (\text{trms}_{\text{aiset}} (\text{transaction\_send } T))\"
then obtain \(ts s\) where \(s::"\text{send} \langle ts \rangle \in \text{set } (\text{unlabel } (\text{transaction\_send } T))" \ "s \in \text{set } ts" \ "t \in \text{subterms } s"
using wellformed\_transaction\_unlabel\_cases(4)[OF T_wf[OF T]]
by fastforce

have \(s_{\text{occ}}::"\exists x. s = \text{occurs } (\text{Var } x)"\) when "\text{OccursFact} \in \text{funs\_term } t \lor \text{OccursSec} \in \text{funs\_term } t"
using that(1) subterms\_eq\_fun\_term\_set\_subst[OF s(3)]
admissible\_transaction\_occurs\_checksE3[OF T\_occ[OF T] _ s(2)]
by blast

obtain \(K \ T'\) where \(K::"\text{Ana } t = (K,T')"\) by moura

show "\text{OccursFact} \notin \bigcup (\text{funs\_term } \set (\text{snd } (\text{Ana } t))) \land
\text{OccursSec} \notin \bigcup (\text{funs\_term } \set (\text{snd } (\text{Ana } t)))"
proof (rule ccontr)
assume "\neg (\text{OccursFact} \notin \bigcup (\text{funs\_term } \set (\text{snd } (\text{Ana } t))) \land
\text{OccursSec} \notin \bigcup (\text{funs\_term } \set (\text{snd } (\text{Ana } t))))"

hence a: "\text{OccursFact} \in \bigcup (\text{funs\_term } \set (\text{snd } (\text{Ana } t))) \lor
\text{OccursSec} \in \bigcup (\text{funs\_term } \set (\text{snd } (\text{Ana } t)))"
by simp
hence "\text{OccursFact} \in \bigcup (\text{funs\_term } \set (\text{snd } t') \lor \text{OccursSec} \in \bigcup (\text{funs\_term } \set (\text{snd } t'))"
using K by simp

hence "\text{OccursFact} \in \text{funs\_term } t \lor \text{OccursSec} \in \text{funs\_term } t"
using Ana\_subterm[OF K] funs\_term\_subterms\_eq(1)[OF t] by blast
then obtain \(x\) where \(x::"t \in \text{subterms } (\text{occurs } (\text{Var } x))"
using s(3) s\_occ by blast
thus False using a by fastforce
qed
have 5: "OccursSec \notin \bigcup (\text{funs_term ` subst_range} (\xi, \sigma, \alpha))" \
"\text{OccursFact} \notin \bigcup (\text{funs_term ` subst_range} (\xi, \sigma, \alpha))" 
when \( \sigma: \text{transaction_decl_subst} \xi T \text{ "transaction_fresh_subst} \sigma T A" \
\text{transaction_renaming_subst} \alpha P A" 
and \( T: "T \in \text{set} P" 
for \( \xi, \sigma, \alpha \) and \( T:="(\text{fun, atom, sets, lbl}) \text{ prot_transaction}" 
and \( A:="(\text{fun, atom, sets, lbl}) \text{ prot_constr}" 

proof - 

have "\text{OccursFact} \notin \bigcup (\text{funs_term ` subst_range} (\xi, \sigma, \alpha))" "\text{OccursSec} \notin \bigcup (\text{funs_term ` subst_range} (\xi, \sigma, \alpha))" 
when \( t \in \text{subst_range} (\xi, \sigma, \alpha)" \) for \( t \) 
using \( \text{transaction_decl_fresh_renaming_substs_range} (3)[ \
\text{OF} \xi \sigma \text{ that } \xi \emptyset [\text{OF} T \xi \alpha (1)] T\_\text{fresh_type} [\text{OF} T] ] \) 
by auto 
thus "\text{OccursFact} \notin \bigcup (\text{funs_term ` subst_range} (\xi, \sigma, \alpha))" "\text{OccursSec} \notin \bigcup (\text{funs_term ` subst_range} (\xi, \sigma, \alpha))" 
by blast+ 

qed 

have 6: "I x \neq \text{Fun OccursSec} [] "\( \exists t. I x = \text{occurs} t " \exists a. \Gamma (I x) = T\text{Atom} a \land a \neq \text{OccursSecType}" 
when \( T: "T \in \text{set} P" 
and \( \sigma: \text{transaction_decl_subst} \xi T \text{ "transaction_fresh_subst} \sigma T A" \
\text{transaction_renaming_subst} \alpha P A" 
and \( x: "\text{Var} x \in \text{trms_set} (\text{transaction_send} T) \cdot \text{set} \xi \sigma \alpha" \) 
for \( x \xi, \sigma, \alpha \) and \( T:="(\text{fun, atom, sets, lbl}) \text{ prot_transaction}" 
and \( A:="(\text{fun, atom, sets, lbl}) \text{ prot_constr}" 

proof - 

obtain \( t \) where \( t: "t \in \text{trms_set} (\text{transaction_send} T)" "t \cdot (\xi, \sigma, \alpha) = \text{Var} x" 
using \( x \) by moura 
then obtain \( y \) where \( y: "t = \text{Var} y" \) by (cases \( t \)) auto 

have "\exists a. \Gamma t = T\text{Atom} a \land a \neq \text{OccursSecType}" 
using \( \text{O(1,2)} [\text{OF} T] \) \( t (1) y \) 
by force 
thus "\exists a. \Gamma (I x) = T\text{Atom} a \land a \neq \text{OccursSecType}" 
using \( \text{wt_subst_trm''}[\text{OF} 3(1)[\text{OF} T \xi \alpha (1)] \text{wt_subst_trm''}[\text{OF} I, wt] t (2) \) 
by (metis \( \text{subst_apply_term} \simps (1) \) ) 
thus "I x \neq \text{Fun OccursSec} [] "\( \exists t. I x = \text{occurs} t " \)
by auto 

qed 

have 7: "I x \neq \text{Fun OccursSec} [] "\( \exists t. I x = \text{occurs} t " \exists a. \Gamma (I x) = T\text{Atom} a \land a \neq \text{OccursSecType}" 
when \( T: "T \in \text{set} P" 
and \( \sigma: \text{transaction_decl_subst} \xi T \text{ "transaction_fresh_subst} \sigma T A" \
\text{transaction_renaming_subst} \alpha P A" 
and \( x: "x \in \text{fv_set} (((\xi, \sigma, \alpha) \cdot \text{vars_transaction} T)" 
for \( x \xi, \sigma, \alpha \) and \( T:="(\text{fun, atom, sets, lbl}) \text{ prot_transaction}" 
and \( A:="(\text{fun, atom, sets, lbl}) \text{ prot_constr}" 

proof - 

obtain \( y \) where \( y: "y \in \text{vars_transaction} T" "x \in \text{fv} (((\xi, \sigma, \alpha) y)" 
using \( x \) by auto 
hence \( y: "((\xi, \sigma, \alpha) y) \neq \text{Var} x" 
using \( \text{transaction_decl_fresh_renaming_substs_range} (3)[ \
\text{OF} \xi \alpha \cdot \text{\_empty}[\text{OF} T \xi \alpha (1)] T\_\text{fresh_type} [\text{OF} T] ] \) 
by (cases "((\xi, \sigma, \alpha) y \in \text{subst_range} (((\xi, \sigma, \alpha) y))") force+ 

have "\exists a. \Gamma (\text{Var} y) = T\text{Atom} a \land a \neq \text{OccursSecType}" 
using \( \text{O(5,6)} [\text{OF} T] y \) 
by force 
thus "\exists a. \Gamma (I x) = T\text{Atom} a \land a \neq \text{OccursSecType}" 
using \( \text{wt_subst_trm''}[\text{OF} 3(1)[\text{OF} T \xi \alpha (1)] \text{wt_subst_trm''}[\text{OF} I, wt] y \) 
by (metis \( \text{subst_apply_term} \simps (1) \) ) 
thus "I x \neq \text{Fun OccursSec} [] "\( \exists t. I x = \text{occurs} t " \)
by auto
3.3 Stateful Protocol Model

qed

have S: "\( \exists t. \ I = \text{occurs } t \) "\( \exists a. \ \Gamma(I) = \text{TAtom } a \land a \neq \text{OccursType} \)"
when T: "\( T \in \text{set } P \)"
and \( \xi; \sigma; \alpha: \text{transaction_decl_subst } \xi; \sigma; \alpha \) T A"
and \( x: \text{Var } x \in \text{subterms_set } (\text{trms}_{\text{set }}(\text{transaction_send } T)) \) set \( \xi; \sigma; \alpha \) for \( x \in \xi; \sigma; \alpha \) and T: "\( ('fun', 'atom', 'sets', 'lbl') \text{ prot_transaction} \)"
and \( A: ('fun', 'atom', 'sets', 'lbl') \text{ prot_constr} \) proof -
  obtain t where t: "\( t \in \text{subterms_set } (\text{trms}_{\text{set }}(\text{transaction_send } T)) \) "\( t \cdot (\xi; \sigma; \alpha) = \text{Var } x \)"
  using x by moura
  then obtain y where y: "t = \text{Var } y" by (cases t) auto

have \( \exists a. \ \Gamma(t = \text{TAtom } a \land a \neq \text{OccursType}) \)"
using \( 0(3,4)[\text{OF } T] t(1) \) y
by force
thus \( \exists a. \ \Gamma(I) = \text{TAtom } a \land a \neq \text{OccursType} \)"
using \( \text{wt_subst_trm}''[\text{OF } 3(1)[\text{OF } T \xi; \sigma; \alpha]] \text{ wt_subst_trm}''[\text{OF } I, \text{wt}] t(2) \)
by (metis \text{subst_apply_term.simps}(1))
thus "\( I \neq \text{Fun } \text{OccursSec } I \) "\( \exists t. \ I = \text{occurs } t \)"
by auto

qed

have s_fv: "\( \text{fv } s \subseteq \text{fv_set } ((\xi; \sigma; \alpha) \setminus \text{vars_transaction } T) \)"
when s: "\( s \in \text{subterms_set } (\text{trms}_{\text{set }}(\text{transaction_send } T)) \) set \( \xi; \sigma; \alpha \)"
and T: "\( T \in \text{set } P \)"
for s and \( \xi; \sigma; \alpha: ('fun', 'atom', 'sets', 'lbl') \text{ prot_subst} \)"
and T: "('fun', 'atom', 'sets', 'lbl') \text{ prot_transaction}"
proof -
  obtain t where t: "\( t \in \text{trms}_{\text{set }}(\text{transaction_send } T) \) "\( x \in \text{fv } (t \cdot (\xi; \sigma; \alpha)) \)"
  using * by fastforce
  hence "\( \text{fv } t \subseteq \text{vars_set } (\text{transaction_send } T) \)"
  using \( \text{fv_subterms_set_subst} \) by fast
  have "\( x \in \text{fv_set } ((\xi; \sigma; \alpha) \setminus \text{vars_set } (\text{transaction_send } T)) \)"
  proof -
    obtain t where t: "\( t \in \text{trms}_{\text{set }}(\text{transaction_send } T) \) "\( x \in \text{fv } \)"\( t \cdot (\xi; \sigma; \alpha) \)"
    using * by fastforce
    hence "\( \text{fv } t \subseteq \text{vars_set } (\text{transaction_send } T) \)"
    using \( \text{fv_trms_set_subset}(1)[\text{of } (\text{unlabel } (\text{transaction_send } T))] \)
    by auto
    thus "\( x \in \text{fv_set } ((\xi; \sigma; \alpha) \setminus \text{vars_transaction } T) \)"
    using \( \text{vars_transaction_unfold}[\text{of } T] \) by fastforce
  qed

show "?A A" using _reach proof (induction A rule: reachable_constraints.induct)
case (step A T \xi; \sigma; \alpha)
  have *: "\( \forall s \in \text{subterms_set } (\text{trms}_{\text{set }}(\text{transaction_send } T)) \)"
    OccursFact \( \notin \bigcup(\text{funs_term } \setminus \text{set } (\text{and } (\text{Ana } a))) \)"
  using \( 4(\text{OF } \text{step.hyps}(2)) \) by blast

have "\( \forall s \in \text{subterms_set } (\text{trms}_{\text{set }}(\text{transaction_send } T)) \) set \( \xi; \sigma; \alpha \) set I."
  OccursFact \( \notin \bigcup(\text{funs_term } \setminus \text{set } (\text{and } (\text{Ana } a))) \)"
proof
  fix t assume t: "\( t \in \text{subterms_set } (\text{trms}_{\text{set }}(\text{transaction_send } T)) \) set \( \xi; \sigma; \alpha \) set I"
  then obtain s u where s: "\( s \in \text{subterms_set } (\text{trms}_{\text{set }}(\text{transaction_send } T)) \) set \( \xi; \sigma; \alpha \) "s \cdot I = t"
3 Stateful Protocol Verification

"u ∈ subterms_set (trms _ t set (transaction_send T))" "u · ξ o s σ o s α = s"
by force

obtain Ku Tu where KT u: "Ana u = (Ku, Tu)" by moura

have "①: "OccursFact \notin (funs_term set Tu)"
  "OccursFact \notin \bigcup(funs_term \· subst_range (ξ o s σ o s α))"
  "OccursFact \notin \bigcup(funs_term \· \bigcup((set o snd o Ana) \· subst_range (ξ o s σ o s α)))"
using transaction_decl_fresh_renaming_substs_range(3)
  OF step.hyps(3-5) _ empty[OF step.hyps(2,3)] T_fresh_type[OF step.hyps(2)]
4[OF step.hyps(2)] su(3) KT u
by (fastforce, fastforce, fastforce)

have "OccursFact \notin \bigcup(funs_term set (Tu t list ξ o s σ o s α))"
proof -
  { fix f assume f: "f ∈ \bigcup(funs_term set (Tu t list ξ o s σ o s α))"
    then obtain tf where tf: "f ∈ set Tu" "f ∈ funs_term tf (tf · ξ o s σ o s α)" by moura
    hence "f ∈ funs_term tf \∨ f ∈ \bigcup(funs_term \· subst_range (ξ o s σ o s α))"
      using funs_term subst[of tf "ξ o s σ o s α"] by force
    hence "f \neqOccursFact" using *(1,2) tf(1) by blast
  } thus ?thesis by blast
qed

hence "OccursFact \notin \bigcup(funs_term set (snd (Ana s)))"
proof (cases u)
  case (Var xu)
  hence "s = (ξ o s σ o s α) xu" using su(4) by (metis subst_apply_term.simps(1))
  thus ?thesis using *(3) by fastforce
qed (use su(4) Ku Tu Ana_subst'[of _ _ Ku Tu "ξ o s σ o s α" in simp]

show "OccursFact \notin \bigcup(funs_term set (snd (Ana t)))"
proof (cases s)
  case (Var sx)
  then obtain a where a: "Γ (I sx) = Var a"
    using su(1) B(3)[OF step.hyps(2-5), of sx] by fast
  hence "Ana (I sx) = (([],[])" by (metis I_grounds(2) const_type_inv[THEN Ana_const])
  thus ?thesis using Var su(2) by simp
next
  case (Fun f S)
  hence snd Ana t: "snd (Ana t) = snd (Ana s) · t list I"
    using su(2) Ana_subst'[of f S _ "snd (Ana s)" I] by (cases "Ana s") simp_all

  { fix g assume "g ∈ \bigcup(funs_term set (snd (Ana t)))"
    hence "g ∈ \bigcup(funs_term set (snd (Ana s))) \∨
      \exists x ∈ fv_set (set (snd (Ana s))). g ∈ funs_term (I x)"
      using snd Ana_t funs_term subst[of _ I] by auto
    hence "g \neqOccursFact"
    proof
      assume "∃ x ∈ fv_set (set (snd (Ana s))). g ∈ funs_term (I x)"
      then obtain x where x: "x ∈ fv_set (set (snd (Ana s)))" "g ∈ funs_term (I x)" by moura
      hence "x ∈ fv s" using x(1) Ana_vars(2)[of s] by (cases "Ana s") auto
      hence "x ∈ fv_set ((ξ o s σ o s α) · var_transaction T)"
        using s_fv[OF su(1) step.hyps(2)] by blast
      then obtain a h U where h:
        "I x = Fun h U" "I (I x) = Var a" "a \neqOccursType" "arity h = 0"
        using I_grounds(2) 7(3)[OF step.hyps(2-5)] const_type_inv
      by metis
      hence "h \neqOccursFact" by auto
      moreover have "U = []" using h(1,2,4) const_type_inv uf[of h U a] I uf_trms by fastforce
      ultimately show ?thesis using h(1) x(2) by auto
    qed (case ** in blast)
  } thus ?thesis by blast
qed

qed
thus \(\text{?case}\)

\[
\text{using step.IH step.prems 1[OF step.hyps(2), of A "\(\xi o_s \sigma o_s \alpha\)"]}
\]

\[
2[OF step.hyps(2) 3[OF step.hyps(2-5)]]
\]

\[
\text{by auto}
\]

\text{qed simp}

show "\(\text{?B A} \) using \(A\)_reach

\text{proof (induction \(A\) rule: reachable_constraints.induct)

case (step \(A\) \(T\) \(\xi \sigma \alpha\))

have "\(\forall s \in \text{subterms}_{\text{set}} (\text{trms}_{\text{set}} (\text{transaction_send } T)) \cdot \text{set } \xi o_s \sigma o_s \alpha \cdot \text{set } I.

\text{OccursSec} \notin \bigcup (\text{funs_term } \cdot \text{set } (\text{snd } (\text{Ana } s)))"

\text{proof}

\text{fix } t \text{ assume } t: "t \in \text{subterms}_{\text{set}} (\text{trms}_{\text{set}} (\text{transaction_send } T)) \cdot \text{set } \xi o_s \sigma o_s \alpha \cdot \text{set } I"

\text{then obtain } s \text{ u where } s:

"s \in \text{subterms}_{\text{set}} (\text{trms}_{\text{set}} (\text{transaction_send } T)) \cdot \text{set } \xi o_s \sigma o_s \alpha" "s \cdot I = t"

\text{by force}

\text{obtain Ku Tu where } KTu: "Ana u = (Ku,Tu)" \text{ by moura}

have \(\emptyset\): "\text{OccursSec} \notin \bigcup (\text{funs_term } \cdot \text{set } (\text{snd } (\text{Ana } s)))"

\text{using transaction_decl_fresh_renaming_substs_range'(3) [OF step.hyps(3-5) _ \text{empty}[OF step.hyps(2,3)] \text{I_fresh_type}[OF step.hyps(2)]]

4[OF step.hyps(2) \text{su}(3) \text{KTu}]

\text{by (fastforce,fastforce,fastforce)}

have "\text{OccursSec} \notin \bigcup (\text{funs_term } \cdot \text{set } (\text{Tu } '\text{list } \xi o_s \sigma o_s \alpha'))"

\text{proof -}

\text{fix } f \text{ assume } f: "f \in \bigcup (\text{funs_term } \cdot \text{set } (\text{Tu } '\text{list } \xi o_s \sigma o_s \alpha'))"

\text{then obtain } tf \text{ where } tf: "tf \in \text{set } Tu" "f \in \text{funs_term } (tf \cdot \xi o_s \sigma o_s \alpha)" \text{ by moura}

\text{hence } "\text{f} \in \text{funs_term } (tf \lor f) \in \bigcup (\text{funs_term } \cdot \text{subst_range } (\xi o_s \sigma o_s \alpha'))"

\text{using funs_term_subst[of tf "\text{Ana } s"] by force}

\text{hence } "f \neq \text{OccursSec}" \text{ using } *(1,2) tf(1) \text{ by blast}

\text{thus } \text{thesis by metis}

\text{qed}

\text{hence } **: "\text{OccursSec} \notin \bigcup (\text{funs_term } \cdot \text{set } (\text{snd } (\text{Ana } s)))"

\text{proof (cases u)}

\text{case (Var xu)}

\text{hence } "s = (\xi o_s \sigma o_s \alpha) xu" \text{ using } \text{su}(4) \text{ by (metis subst_apply_term.simps(1))}

\text{thus } \text{thesis using } *(3) \text{ by fastforce}

\text{qed (use } \text{su}(4) \text{ KTu Ana_subst'[of } _ _ \text{ Ku Tu } "\xi o_s \sigma o_s \alpha") \text{ in simp)}

\text{show } "\text{OccursSec} \notin \bigcup (\text{funs_term } \cdot \text{set } (\text{snd } (\text{Ana } t)))"

\text{proof (cases s) }

\text{case (Var sx)}

\text{then obtain } a \text{ where } a: "\text{I } (\text{I } sx) = \text{Var } a"

\text{using } \text{su}(1) 8(3)[OF step.hyps(2-5), of sx] \text{ by fast}

\text{hence } "\text{Ana } (\text{I } sx) = ([],[],[])" \text{ by (metis I.grounds(2) const_type_inv[THEN Ana.const])}

\text{thus } \text{thesis using Var } \text{su}(2) \text{ by simp}

next

\text{case } (\text{Fun } f S)

\text{hence snd_Ana_t: } "\text{snd } (\text{Ana } t) = \text{snd } (\text{Ana } s) \cdot '\text{list } I"

\text{using } \text{su}(2) \text{ Ana_subst'[of } _ _ \text{ "snd } (\text{Ana } s)" I] \text{ by (cases } \text{Ana } s \text{) simp_all}

\text{next}

\text{case } (\text{Fun } f S)

\text{hence snd_Ana_t: } "\text{snd } (\text{Ana } t) = \text{snd } (\text{Ana } s) \cdot '\text{list } I"

\text{using } \text{su}(2) \text{ Ana_subst'[of } _ _ \text{ "snd } (\text{Ana } s)" I] \text{ by (cases } \text{Ana } s \text{) simp_all}

\{ \text{fix } g \text{ assume } "g \in \bigcup (\text{funs_term } \cdot \text{set } (\text{snd } (\text{Ana } t)))"

\text{hence } "g \in \bigcup (\text{funs_term } \cdot \text{set } (\text{snd } (\text{Ana } s))) \lor (\exists x \in f_{v,\text{set}} (\text{set } (\text{snd } (\text{Ana } s))). g \in \text{funs_term } (I x))"

\text{using } \text{snd_Ana_t funs_term_subst[of } _ _ \text{ by auto}

\text{hence } "g \neq \text{OccursSec}"

\text{proof}

\text{assume } "\exists x \in f_{v,\text{set}} (\text{set } (\text{snd } (\text{Ana } s))). g \in \text{funs_term } (I x)"
then obtain x where x: "x ∈ fv_set (set (snd (Ana s)))" "g ∈ funs_term (I x)" by moura
have "x ∈ fv_set" using x(1) Ana_vars(2)[of s] by (cases "Ana s") auto
hence "x ∈ fv_set ((ξ o_σ o_σ o_σ) ~ vars_transaction T)"
using s_fv[OF su(1) step.hyps(2)] by blast
then obtain a U where h:
  "I x = Fun h U" "I (I x) = Var a" "a ≠ OccursSecType" "arity h = 0"
using I_grounds(2) T(3)[OF step.hyps(2-5)] const_type_inv
by metis
hence "h ≠ OccursSec" by auto
moreover have "U = []" using h(1,2,4) const_type_inv uf[of h a] I_uf vars by fastforce
ultimately have "Fun OccursSec []" using h(1) x(2) by auto
qed (use ** in blast)
} thus ?thesis by blast

qed simp

show "?C A" using A_reach
proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α)
  have *: "Fun OccursSec [] \in trms_set (transaction_send T)"
    using admissible Transaction_occurs_checkSE5[OF T_occ[OF step.hyps(2)]] by blast
  have **: "Fun OccursSec [] \in subst_range (ξ o_σ o_σ o_σ)"
    using transaction_decl_fresh_renaming_substs_range'(3)[OF step.hyps(3-5) _ξ_empty[OF step.hyps(2,3)] T_fresh_type[OF step.hyps(2)]]
    by auto
  have "Fun OccursSec [] \in trms_set (transaction_send T) \set ξ o_σ o_σ o_σ \set I"
    proof
    assume "Fun OccursSec [] \in trms_set (transaction_send T) \set ξ o_σ o_σ o_σ \set I"
    then obtain s where "s = trms_set (transaction_send T) \set ξ o_σ o_σ o_σ \set I = Fun OccursSec []"
      by moura
    moreover have "Fun OccursSec [] \in trms_set (transaction_send T) \set ξ o_σ o_σ o_σ" by moura
    proof
      assume "Fun OccursSec [] \in trms_set (transaction_send T) \set ξ o_σ o_σ o_σ"
      then obtain u where "u ∈ trms_set (transaction_send T)\set ξ o_σ o_σ o_σ = Fun OccursSec []" by moura
      thus False using * ** by (cases u) (force simp del: subst_subst_compose)+
    qed
    ultimately show False using 6[OF step.hyps(2-5)] by (cases s) auto
    qed
  thus ?case using step.IH step.prems 1[OF step.hyps(2), of A "ξ o_σ o_σ o_σ"] by fast
  qed simp

show "?D A" using A_reach
proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α)
  { fix x assume x: "x ∈ vars_set (dual_set (transaction_send T) \set ξ o_σ o_σ o_σ)"
    hence x’: "x ∈ vars_set (unlabel (transaction_send T) \set ξ o_σ o_σ o_σ)"
      by (metis vars_set_unlabel_dual_set_eq unlabel_subst)
    hence "x ∈ vars_transaction T ∨ x ∈ fv_set ((ξ o_σ o_σ o_σ) ~ vars_transaction T)"
      using vars_set_subst_cases[OF x’] by metis
  moreover have "I x ≠ Fun OccursSec []" when "x ∈ vars_transaction T" using That 0(5,6)[OF step.hyps(2)] wt_subst_trm'[OF I_wt, of "Var x"] by fastforce
  ultimately have "I x ≠ Fun OccursSec []" using 7(1)[OF step.hyps(2-5), of x]
3.3 Stateful Protocol Model

lemma reachable_constraints_occurs_fact_ik_subst_aux:
  assumes A_reach: "A ∈ reachable_constraints P"
  and I: "welltyped_constraint_model I A"
  and P: "∀T ∈ set P. admissible_transaction T"
  and t: "t ∈ ik lsst A" "t · I = occurs s"
  shows "∃u. t = occurs u"
proof -
  have "wt subst I"
  using unfolding welltyped_constraint_model_def constraint_model_def by metis
  hence 0: "Γ t = Γ (occurs s)"
  using wt_subst_trm'' by metis
  have 1: "Γ v ` fv lsst A ⊆ (⋃T ∈ set P. Γ v ` fv_transaction T)"
  "∀T ∈ set P. ∀x ∈ fv_transaction T. Γ v x = TAtom Value ∨ (∃a. Γ v x = TAtom (Atom a))"
  using reachable_constraints_var_types_in_transactions(1)[OF A_reach] protocol_transaction_vars_TAtom_typed(2,3) P
  by fast+
  show ?thesis
  proof (cases t)
    case (Var x)
    thus ?thesis
    using 0 1 t(1) var_subterm_ik_sst_is_fv_sst[of x "unlabel A"] by fastforce
  next
    case (Fun f T)
    hence 2: "f = OccursFact" "length T = Suc (Suc 0)" "T ! 0 · I = Fun OccursSec []"
    using t(2) by auto
    have "T ! 0 = Fun OccursSec []"
    proof (cases "T ! 0")
      case (Var y)
      hence "y = Fun OccursSec []" using Fun 2(3) by simp
      moreover have "Var y ∈ set T" using Var 2(2) length_Suc_conv[of T 1] by auto
      hence "y ∈ fv_transaction (ik lsst A)" using Fun t(1) by force
      hence "y ∈ vars lsst A"
      using fv_ik_subset_fv_sst_prime[of "unlabel A"] vars_sst_is_fv_sst_bvars[of "unlabel A"]
      by blast
      ultimately have False
      using reachable_constraints_occurs_fact_ik_funs_terms(4)[OF A_reach I P]
      by blast
      thus ?thesis by simp
    qed (use 2(3) in simp)
  qed
  moreover have "∃u u'. T = [u,u']"
  using iffD1[OF length_Suc_conv 2(2)] iffD1[OF length_Suc_conv[of _ 0]] length_0_conv by fast
  ultimately show ?thesis using Fun 2(1,2) by force
  qed

lemma reachable_constraints_occurs_fact_ik_subst:
  assumes A_reach: "A ∈ reachable_constraints P"
  and I: "welltyped_constraint_model I A"
  and P: "∀T ∈ set P. admissible_transaction T"
  and t: "occurs t ∈ ik lsst A · set I"
  shows "occurs t ∈ ik lsst A"
proof -
  have "wt subst I"
  using unfolding welltyped_constraint_model_def constraint_model_def by metis
obtain \( s \) where \( s: "s \in ik_{lsst} A" \) "\( s \cdot I = \text{occurs } t\)"
using \( t \) by auto
hence \( u: "\exists u. s = \text{occurs } u"\)
using \( \_ \_ \cdot \text{occurs_fact_ik_subst} \cdot \text{OF } A_{\text{reach}} I P\) by blast
hence "\( fv s = \{}\)"
using \( \text{occurs_fact_ik_ground} \cdot \text{OF } A_{\text{reach}} P \cdot s\)
by fast
thus \( ?\text{thesis} \)
using \( s u \cdot \text{subst_ground_ident} \cdot \text{OF } s I\)
by argo
qed

lemma reachable_constraints_occurs_fact_send_in_ik:
assumes \( A_{\text{reach}}: "A \in \text{reachable_constraints } P"\)
and \( I: "\text{welltyped_constraint_model } I \ A"\)
and \( P: "\forall T \in \text{set } P. \text{admissible_transaction } T"\)
and \( x: "\text{occurs (Var } x) \in \text{set } ts" \) "\( \text{send } \langle ts \rangle \in \text{set } (\text{unlabel } A)\)"
shows "\( \text{occurs } (I x) \in ik_{lsst} A"\)
using \( A_{\text{reach}} I x\)
proof (induction \( A \) rule: reachable_constraints.induct)
case \( \text{(step } A T \xi \sigma \alpha) \)
define \( \vartheta \) where "\( \vartheta \equiv \xi \circ s \sigma \circ s \alpha \)"
define \( T' \) where "\( T' \equiv \text{dual}_{lsst} (\text{transaction_strand } T \cdot \text{lsst } \vartheta)"\)

have \( T_{\text{adm}}: "\text{admissible_transaction } T"\)
using \( P \cdot \text{step.hyps(2)} \cdot \text{unfolding } \text{list_all_iff} \) by blast

note \( T_{\text{wf}} = \text{admissible_transaction_is_wellformed_transaction}(1) \cdot \text{OF } T_{\text{adm}}\)
note \( T_{\text{adm_occ}} = \text{admissible_transaction_is_wellformed_transaction}(5) \cdot \text{OF } T_{\text{adm}}\)

have \( I_{\text{is } T_{\text{model}}}: "\text{strand_sem_stateful } (ik_{lsst} A \cdot set I) \cdot (\text{set } (\text{db}_{lsst} A I)) \cdot \text{(unlabel } T') \cdot I"\)
using \( \text{step.prems } \text{unlabel_append}[\text{OF } A T'] \cdot \text{db}_{lsst} \cdot \text{is_dbupd}_{lsst}[\text{OF } (\text{unlabel } A \cdot I) \cdot []]\)
\( \cdot \text{strand_sem_append_stateful}[\text{OF } \{} \cdot \{} \cdot \text{"unlabel } A \cdot \text{"unlabel } T' \cdot I]\)
by (simp add: \( T'_{\text{def}} \cdot \vartheta_{\text{def}} \cdot \text{welltyped_constraint_model_def } \cdot \text{constraint_model_def } \cdot \text{db}_{lsst\_def}\))

show \( ?\text{case} \)

proof (cases "\( \text{send}(ts) \in \text{set } (\text{unlabel } A)\)" )
case False
using \( \text{step.prems(3)} \cdot \text{unfolding } T_{\text{def}} \cdot \vartheta_{\text{def}} \) by simp

hence "\( \text{receive}(ts) \in \text{set } (\text{unlabel } T')\)"
using \( \text{step.prems(3)} \cdot \text{unfolding } T_{\text{def}} \cdot \vartheta_{\text{def}} \) by blast
then obtain \( y \) \( ts' \) where \( y: "\text{receive}(ts') \in \text{set } (\text{unlabel } (\text{transaction_receive } T))"\)
\( \vartheta y = \text{Var } x " \text{occurs } (\text{Var } y) \in \text{set } ts'"\)
using \( \text{transaction_decl_fresh_renaming_substs_occurs_fact_send_receive}(2)[\cdot \text{OF } \text{step.prems(3-5)} \cdot \text{T_{adm}}\]
\( \cdot \text{subst_to_var_is_var}[\text{OF } \cdot \vartheta x] \cdot \text{step.prems(2)}\)
unfolding \( \vartheta_{\text{def}} \) by (force simp del: \( \cdot \text{subst_subst_compose} \))

hence "\( \text{occurs } (\text{Var } y) \cdot \vartheta \in \text{set } ts' \cdot \text{lsst } \vartheta"\)
\( \cdot \text{receive}(ts' \cdot \text{lsst } \vartheta) \in \text{set } (\text{unlabel } (\text{transaction_receive } T \cdot \text{lsst } \vartheta))"\)
using \( \text{subst_last_unlabel_member}[\text{OF } "\text{receive}(ts')" \cdot \text{"transaction_receive } T \cdot \vartheta]\)
by fastforce

hence "\( ik_{lsst} A \cdot set I = \text{occurs } (\text{Var } y) \cdot \vartheta \cdot I"\)
using \( \text{wellformed_transaction_sem_receives}[\cdot \text{OF } T_{\text{wf}}, \text{OF } "ik_{lsst} A \cdot set I" \cdot \text{"set } (\text{db}_{lsst} A I)" \cdot \vartheta I \cdot ts' \cdot \text{lsst } \vartheta]\)
\( I_{\text{is } T_{\text{model}}} \)
unfolding \( T_{\text{def}} \cdot \text{list_all_iff} \) by fastforce
hence \(?: "ik_{lsst} A \cdot set I = \text{occurs } (\vartheta y \cdot I)"\)
by auto
3.3 Stateful Protocol Model

have "occurs (ϑ y · I) ∈ ik lsst A"
using deduct_occurs_in_ik[OF *]
reachable_constraints_occurs_fact_ik_subst[
reachable_constraints_occurs_fact_ik_funs_terms[
reachable_constraints_occurs_fact_ik_funs_terms(1-3)[OF A_reach I P]
by blast
thus ?thesis using y(2) by simp
qed (simp add: step.IH[OF welltyped_constraint_model_prefix[OF step.prems(1)]] step.prems(2))

lemma reachable_constraints_occurs_fact_deduct_in_ik:
assumes A_reach: "A ∈ reachable_constraints P"
and I: "welltyped_constraint_model I A"
and P: "∀ T ∈ set P. admissible_transaction T"
and k: "ik lsst A · set I ⊢ occurs k"
shows "occurs k ∈ ik lsst A · set I"
and "occurs k ∈ ik lsst A"
using reachable_constraints_occurs_fact_ik_funs_terms[OF A_reach I P]
reachable_constraints_occurs_fact_ik_subst[OF A_reach I P]
deduct_occurs_in_ik[OF k]
by (presburger, presburger)

lemma reachable_constraints_fv_bvars_subset:
assumes A: "A ∈ reachable_constraints P"
shows "bvars lsst A ⊆ (∪ T ∈ set P. bvars_transaction T)"
using assms
proof (induction A rule: reachable_constraints.induct)
  case (step A T ξ σ α)
  let ?T' = "transaction_strand T · lsst ξ ◦ s σ ◦ s α"
  show ?case
    using step.IH step.hyps(2)
bvars lsst_unlabel_dual lsst_eq[of ?T']
bvars lsst_unlabel[of "transaction_strand T" "ξ ◦ s σ ◦ s α"]
bvars lsst_append[of "unlabel A" "unlabel (dual lsst ?T')"]
unlabel_append[of A "dual lsst ?T'"]
by (metis (no_types, lifting) SUP_upper Un_subset_iff)
qed simp

lemma reachable_constraints_fv_disj:
  fixes A::"('fun,'atom,'sets,'lbl) prot_constr"
  assumes A: "A ∈ reachable_constraints P"
  shows "fv lsst A ∩ (∪ T ∈ set P. bvars_transaction T) = {}"
  using A
proof (induction A rule: reachable_constraints.induct)
  case (step A T ξ σ α)
  define T' where "T' ≡ transaction_strand T · lsst ξ ◦ s σ ◦ s α"
  define X where "X ≡ (∪ T ∈ set P. bvars_transaction T)"
  have "fv lsst T' ∩ X = {}"
    using transaction_decl_fresh_renaming_substs_vars_disj(4)[OF step.hyps(3-5)]
    transaction_decl_fresh_renaming_substs_vars_subset(4)[OF step.hyps(3-5, 2)]
    unfolding T'_def X_def by blast
  hence "fv lsst (A @ dual lsst T') ∩ X = {}"
    using step.IH[unfolded X_def[symmetric]] fv lsst_unlabel_dual lsst_eq[of T'] by auto
  thus ?case unfolding T'_def X_def by blast
qed simp

lemma reachable_constraints_fv_bvars_disj:
  fixes A::"('fun,'atom,'sets,'lbl) prot_constr"
  assumes P: "∀ T ∈ set P. wellformed_transaction T"
  and A: "A ∈ reachable_constraints P"
  shows "fv lsst A ∩ bvars lsst A = {}"
using A
proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α)
define T' where "T' ≡ dual lsst T · lsst ξ ◦ s σ ◦ s α"

note 0 = transaction_decl_fresh_renaming_substs_vars_disj[OF step.hyps(3-5)]
note 1 = transaction_decl_fresh_renaming_substs_vars_subset[OF step.hyps(3-5)]

have 2: "bvars lsst A ∩ fv lsst T' = {}" using 0(7) 1(4)[OF step.hyps(2)]

have 3: "fv lsst A ∩ bvars lsst T' = {}" by blast

have "fv lsst (A @ T') ∩ bvars lsst (A @ T') = {}" using 2 3 4 step.IH unfolding unlabel_append[of "unlabel A" "unlabel T"]

lemma reachable_constraints_wf:
assumes P: "∀ T ∈ set P. wellformed_transaction T"
and A: "A ∈ reachable_constraints P"
shows "wf sst (unlabel A)" and "wf trms (trms lsst A)"
proof -
let ?X = "λ T. fst ` set (transaction_decl T ()) ∪ set (transaction_fresh T)"

have "wellformed_transaction T" when "T ∈ set P" for T using P(1) that by fast+
hence 0: "fv lsst (?X T) (unlabel (dual lsst (transaction_strand T)))" "fv lsst (dual lsst (transaction_strand T)) ∩ bvars lsst (dual lsst (transaction_strand T)) = {}" "wf trms (trms lsst A)" when T: "T ∈ set P" for T unfolding admissible_transaction_terms_def by (metis T wellformed_transaction_wf_sst(1), T wellformed_transaction_wf_sst(2) fv lsst unlabel dual lsst eq bvars lsst unlabel dual lsst eq, T \{unlabel A\} )

from A have "wf sst (unlabel A) ∧ wf trms (trms lsst A)"

proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α)

have IH: "wf sst {} (unlabel A)" "fv lsst A ∩ bvars lsst A = {}" "wf trms (trms lsst A)" using step.IH by metis
3.3 Stateful Protocol Model

have 1: "wf \_set \{ (unlabel (A @?T'))\}"
using transaction_decl_fresh_renaming_substs_wf_sst[\OF 0(1)[\OF step.hyps(2)] step.hyps(3-5)]
wf\_set_vars_mono[of "{}"] wf\_set_append[of IH(1)]
by simp

have 2: "fv\_set (A @?T') \cap bvvars\_set (A @?T') = {}"
using reachable_constraints_fv_bvars_disj[of P(1)]
reachable_constraints.step[of \OF step.hyps]
by blast

have "wf\_trms (trms\_set (A @?T'))"
using trms\_set_unlabel_dual\_set_eq unlabel_subst
wf\_trms_subst[
    OF transaction_decl_fresh_renaming_substs_range_wf\_trms[of \OF step.hyps(3-5)],
    THEN wf\_trms_trms\_set_subst,
    OF 0(3)[\OF step.hyps(2)]]
by metis

hence 3: "wf\_trms (trms\_set (A @?T'))"
using IH(3)
by auto

show ?case using 1 2 3 by force
qed simp
thus "wf\_set (unlabel A)" "wf\_trms (trms\_set A)" by metis+
qed

lemma reachable_constraints_no_Ana_attack:
  assumes A: "A \in \text{reachable\_constraints} P"
  and P: "\forall T \in \text{set} P. \text{wellformed\_transaction} T"
  "\forall T \in \text{set} P. \text{admissible\_transaction\_terms} T"
  "\forall T \in \text{set} P. \forall x \in \text{set} (\text{transaction\_fresh} T). \Gamma_v x = \text{TAtom Value} \lor (\exists a. \Gamma_v x = \text{TAtom (Atom a)})"
  and t: "t \in \text{subterms\_set} (ik\_set A)"
shows "attack(n) \notin \text{set} (\text{snd (Ana t)})"
proof -
  have T_adm_term: "\text{admissible\_transaction\_terms} T" when "T \in \text{set} P" for T
  using P that
  by blast
  have T_wf: "\text{wellformed\_transaction} T" when "T \in \text{set} P" for T
  using P that
  by blast
  have T_fresh: "\forall x \in \text{set} (\text{transaction\_fresh} T). \Gamma_v x = \text{TAtom Value} \lor (\exists a. \Gamma_v x = \text{TAtom (Atom a)})" when "T \in \text{set} P" for T
  using P(3) that
  by fast
  show ?thesis using A t
  proof (induction A rule: reachable_constraints.induct)
    case (step A T \xi \sigma \alpha) thus ?case
    proof (cases "t \in \text{subterms\_set} (ik\_set A)"
      case False
      hence "t \in \text{subterms\_set} (ik\_set (\text{dual\_set} (\text{transaction\_strand} T \cdot \text{set} A) \cdot \text{set} (\text{transaction\_send} T) \cdot \text{set} \xi \cdot \sigma \cdot \alpha)))"
      using step.prems by simp
      hence "t \in \text{subterms\_set} (\text{trms\_set} (\text{transaction\_send} T) \cdot \text{set} \xi \cdot \sigma \cdot \alpha)"
      using dual_transaction_ik_is_transaction_send'[\OF T_wf[of \OF step.hyps(2)]]
      by metis
      hence "t \in \text{subterms\_set} (\text{trms\_set} (\text{transaction\_send} T)) \cdot \text{set} \xi \cdot \sigma \cdot \alpha"
      using transaction_decl_fresh_renaming_substs_trms[
          OF step.hyps(3-5), of "\text{transaction\_send} T"
        ]
      wellformed_transaction_unlabel_cases[of P(4)[\OF T_wf[\OF step.hyps(2)]]]
      by fastforce
    then obtain s where s: "s \in \text{subterms\_set} (\text{trms\_set} (\text{transaction\_send} T))" "t = s \cdot \xi \cdot \sigma \cdot \alpha"
      by moura
hence $s'$: "attack($n$) $\notin$ set (snd (Ana $s$))"

using admissible_transaction_no_Ana_Attack[of T_adm_term[of step.hyps(2)]

trms_transaction_unfold[of T]

by blast

note * = transaction_decl_fresh_renaming_substs_range'(1-3)[OF step.hyps(3-5)]

transaction_decl_fresh_renaming_subsets_range_no_attack_const[

OF step.hyps(3-5) T_fresh[of step.hyps(2)]

show ?thesis

proof

assume n: "attack($n$) $\in$ set (snd (Ana $t$))"

thus False

proof (cases $s$)

case (Var $x$)

hence "($\exists c. t = \text{Fun} f []) \lor (\exists y. t = \text{Var} y)"

using *(1)[of $t$] n s(2) by (force simp del: subst_subst_compose)

thus ?thesis using n Ana_subterm' by fastforce

next

case (Fun $f$ $S$)

hence "attack($n$) $\in$ set (snd (Ana $s$)) $\cdot$ set $\xi$ $\circ$ $\circ$ $\alpha$"

using Ana_subst'[of $f$ $S$ _ "snd (Ana $s$)" $\cdot$ set $\xi$ $\circ$ $\circ$ $\alpha$] s(2) $n$ $s'$

by (cases "Ana $s$") auto

hence "attack($n$) $\in$ set (snd (Ana $s$)) $\lor$ attack($n$) $\in$ subst_range ($\xi$ $\circ$ $\circ$ $\alpha$)"

using const_mem_subst_cases' by fast

thus ?thesis using *(4) $s'$ by fast

qed

qed simp

qed simp

qed

lemma reachable_constraints_receive_attack_if_attack:

assumes $A$: "$A$ $\in$ reachable_constraints $P$"

and P: "$\forall T \in$ set $P$. wellformed_transaction $T$"

"$\forall T \in$ set $P$. admissible_transaction_terms $T$"

"$\forall T \in$ set $P$. $\forall x \in$ set (transaction_fresh $T$).

$\Gamma_v x = TAtom \text{ Value} \lor (\exists a. \Gamma_v x = TAtom (\text{Atom} a))"

"$\forall T \in$ set $P$. $\forall x \in$ set $\text{trms}$ $T$.

$\not= -\text{Atom} \text{ AttackType} \sqsubseteq \Gamma_v x"

and $I$: "welltyped_constraint_model $I$ $A$"

and 1: "$\text{ik}_{\text{set}} A \cdot set I \vdash \text{attack}(1)"

shows "receive([\text{attack}(1)]) $\in$ set (unlabel $A$)"

and "$\forall T \in$ set $P$. $\forall x \in$ set (transaction_strand $T$).

is_Send (snd $s$) $\land$ length (the_msgs (snd $s$)) = 1 $\land$

is_Fun_Attack (hd (the_msgs (snd $s$))))

$\rightarrow$ the_Attack_label (the_Fun (hd (the_msgs (snd $s$))))) = fas $s$

$\Rightarrow$ (1, receive([\text{attack}(1)])) $\in$ set $A$" (is "$?Q \Rightarrow (1, receive([\text{attack}(1)]))$ $\in$ set $A$)

proof -

have $I'$: "constr_sn_terminate $I$ (unlabel $A$)" "interpretation_subst $I$"

"$\text{wf}_{\text{trms}}$ (subst_range $I$)" "$\text{wt}_{\text{subst}} I$"

using $I$ unfolding welltyped_constraint_model_def constraint_model_def by metis+

have 0: "$\text{wf}_{\text{trms}}$ (ik\text{set} $A$ $\cdot$ set $I$)"

when $A$: "$A$ $\in$ reachable_constraints $P$" for $A$

using reachable_constraints_wftrms[of _ $A$] admissible_transaction_terms_wftrms, P(2)

ik\text{set}_\text{trms}_{\text{set}}$ subset[of "unlabel $A$"] $\text{wf}_{\text{trms}}$ $\text{subst}$[of $\text{OF} I', (3)]$

by fast

have 1: "$\forall x \in$ fv\text{set} (ik\text{set} $A$). $\not= -\text{Atom} \text{ AttackType} \sqsubseteq \Gamma_v x"

when $A$: "$A$ $\in$ reachable_constraints $P$" for $A$

using reachable_constraints_vars_not_attack_typed[of $A$ P(3,4)]

fv\text{ik} \text{set}_\text{vars}_{\text{set}}$[of "unlabel $A$"]
by fast

have 2: "attack(l) \notin \text{set (snd (Ana t))} \cdot \text{set I}" when t: "t \in \text{subterms}_{set} (ik_{last} A)" for t
proof
  assume "attack(l) \in \text{set (snd (Ana t))} \cdot \text{set I}"
  then obtain s where s: "s \in \text{set (snd (Ana t))}" "s \cdot I = attack(l)" by moura

obtain x where x: "s = \text{Var x}"
  by (cases s) (use s reachable_constraints_no_Ana_attack[OF A P(1-3) t] in auto)

have "x \infv t" using x Ana_subterm[OF s(1)] vars_iff_subtermeq by force
hence "x \infv_{set} (ik_{last} A)" using t fv_subterms by fastforce
hence "\Gamma_{x} \neq TAtom \text{AttackType}" using 1[OF A] by fastforce
thus False using s(2) x wt_subst_trm''[OF I'(4), of "Var x"] by fastforce
qed

have 3: "attack(l) \notin \text{set (snd (Ana t))} \cdot \text{set I}"
when t: "t \in \text{subterms}_{set} (ik_{last} A \cdot \text{set I})" for t
proof
  assume "attack(l) \in \text{set (snd (Ana t))}"
  then obtain s where s: "s \in \text{subterms}_{set} (ik_{last} A)" "attack(l) \in \text{set (snd (Ana s))}
  using Ana_subst_subterms_cases[OF t] by fast
  then obtain x where x: "x \infv_{set} (ik_{last} A)" "s \subseteq \text{I} x" by moura
  hence "\text{I} x \in \text{subterms}_{set} (ik_{last} A \cdot \text{set I})"
  using var_is_subterm[of x] subterms_subst_subset'[of \text{I} "ik_{last} A"]
  by force
  hence *: "\text{wf trm} (\text{I} x)" "\text{wf trm} s"
  using wf_trms_subterms[OF 0[OF A]] wf_trm_subtermeq[OF _ x(2)]
  by auto
  show False
    using term.order_trans[OF substrange_imp_subtermtypeeq[OF s(2) Ana_subterm'[OF s(2)]]
    subtermtypeeq_imp_subtermrange[OF s(1) x(2)]
    1[OF A] x(1) wt_subst_trm''[OF I'(4), of "Var x"]
    by auto
qed

have 4: "t = attack(n)"
when t: "t \cdot \xi \cdot \sigma \cdot \alpha = attack(n)"
  and hyps: "\text{transaction_decl} \cdot \xi \cdot T"
  "\text{transaction_fresh} \cdot \sigma \cdot T \cdot A"
  "\text{transaction_renaming} \cdot \alpha \cdot P \cdot A"
  and T: "\forall x \in \text{set (transaction_fresh T)}. \Gamma_{x} = TAtom \text{Value} \lor (\exists a. \Gamma_{x} = TAtom (\text{Atom} a))"
for n
  and T: "\text{fun}, \text{atom}, \text{sets}, \text{lbl} \cdot \text{prot_transaction}"
  \xi \cdot \sigma \cdot \alpha::"\text{fun}, \text{atom}, \text{sets}, \text{lbl} \cdot \text{prot_subst}"
  \text{and A::"fun}, \text{atom}, \text{sets}, \text{lbl} \cdot \text{prot_strand}"
proof (cases t)
  case (Var x)
  hence "attack(n) \in \text{subterm_range} (\xi \cdot \sigma \cdot \alpha)"
    by (metis (no_types, lifting) t subst_apply_term.simps(1) subst_imgI term.distinct(1))
  thus thesis
    using transaction_decl_fresh_renaming_substs_range_no_attack_const[OF hyps T]
    by blast
qed (use t in simp)

have 5: "\exists ts'. ts = ts' \cdot \text{last} \cdot (l, \text{send}(ts')) \in \text{set (transaction_strand T)}"
when ts: "(l, \text{receive}(ts')) \in \text{set \text{dual}_{last} (transaction_strand T \cdot \text{last} 0))"
for l ts \in and T: "\text{fun}, \text{atom}, \text{sets}, \text{lbl} \cdot \text{prot_transaction}"
  using subst_last_memD'[OF ts[unfolded dual_last_steps_iff(1)[symmetric]]]
  by auto
have 6: "If = 1" when "(l', receive([attack(1)]) ∈ set A" and Q: "?Q" for l'
using A that(1)
proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α) show ?case
proof (cases "(l', receive([attack(1)]) ∈ set A")
case False
hence #: "(l', receive([attack(1)]) ∈ set (transacti on_strand T ·x†, ξ, σ, σ, α))"
using step.prems by simp
have "(l', send([attack(1)]) ∈ set (transaction_strand T)"
using (OF _ step.hyps(3-5)) P(3) step.hyps(2) 5[OF *] by force
thus ?thesis using Q step.hyps(2) unfolding isFunAttack_def by fastforce
qed (simp)

have simp
have 7: "∃ t. ts = [t] ∧ t = attack(l)"
when ts: "receive(ts) ∈ set (unlabel A)" "attack(l) ∈ set ts ·I for ts
using A ts(l)
proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α)
obtain t where t: "t ∈ set ts" "attack(l) = t · I" using ts(2) by blast
hence t_in_ik: "t ∈ ik set (transacti on_strand T ·x†, ξ, σ, σ, α)"
using step.prems(1) in_ik_iff[of t] by blast

have t_attack_eq: "t = attack(l)"
proof (cases t)
case (Var x)
hence "TAtom AttackType ⊈ subterms (Γ t)"
using t_in_ik I[OF reachable_constraints.step[OF step.hyps]] by fastforce
thus ?thesis using t(2) wt_subst_term'[OF I'(4), of t] by force
qed (use t(2) in simp)

show ?case
proof (cases "receive(ts) ∈ set (unlabel A)"
case False
then obtain l' where l':
"(l', receive(ts)) ∈ set (transacti on_strand T ·x†, ξ, σ, σ, α)"
using step.prems(1) unfolding unlabel_def by force
then obtain ts' where ts':
"ts = ts' ·I, ξ, σ, σ, α" "(l', send(ts')) ∈ set (transaction_strand T)"
using 5 by meson
then obtain t' where t': "t' ∈ set ts" "t' · ξ, σ, σ, α = attack(l)"
using t(1) t_attack_eq by force

note * = t'(1) 4[OF t'(2) step.hyps(3-5)]

have "send(ts') ∈ set (transacti on_strand T)"
using ts'(2) step.hyps(2) P(2) unfolding unlabel_def by force
hence "length ts' = 1"
using step.hyps(2) P(2,3) * unfolding admissible_transaction_terms_def by fastforce
hence "ts' = [attack(l)]" using * P(3) step.hyps(2) by (cases ts') auto
thus ?thesis by (simp add: ts'(1))
qed (use step.IH in simp)

qed simp

show "attack(l) ∈ ik set A ·I, I"
using private_const_deduct[OF _ 1] 3 by simp
then obtain ts where ts: "receive(ts) ∈ set (unlabel A)" "attack(l) ∈ set ts ·I, I"
using in_ik_iff[of _ A] unfolding unlabel_def by force
then obtain t where "ts = [t]" "t = attack(l)"
using 7 by blast
thus "receive([attack(1)]) ∈ set (unlabel A)"
using ts(1) by blast
hence "∃ t. (l', receive([attack(1)]) ∈ set A"
unlabel_def by fastforce
thus "(l, receive[⟨attack(l)⟩]) ∈ set A" when ?Q
using that 6 by fast
qed

lemma reachable_constraints_receive_attack_if_attack':
  assumes A: "A ∈ reachable_constraints P"
  and P: "∀ T ∈ set P. admissible_transaction T"
  and I: "welltyped_constraint_model I A"
  and n: "iklsst A · set I ⊢ attack⟨n⟩"
  shows "attack⟨n⟩ ∈ iklsst A · set I"
  and "receive⟨[attack⟨n⟩]⟩ ∈ set (unlabel A)"
proof -

  have P': "∀ T ∈ set P. wellformed_transaction T"
  "∀ T ∈ set P. admissible_transaction_terms T"
  "∀ T ∈ set P. ∀ x ∈ vars_transaction T. ¬TAtom AttackType ⊑ Γ_v x"
  using admissible_transaction_is_wellformed_transaction(1,4)[OF bspec[OF P]]
  admissible_transactionE(2,15)[OF bspec[OF P]]
  by (blast, blast, blast, blast)

  show "attack⟨n⟩ ∈ iklsst A · set I" "receive⟨[attack⟨n⟩]⟩ ∈ set (unlabel A)"
  using reachable_constraints_receive_attack_if_attack(1,2)[OF A P'(1,2) _ P'(4) I n] P'(3)
  by (metis, metis)
qed

lemma constraint_model_Value_term_is_Val:
  assumes A_reach: "A ∈ reachable_constraints P"
  and I: "welltyped_constraint_model I A"
  and P: "∀ T ∈ set P. admissible_transaction T"
  and x: "Γ_v x = TAtom Value" "x ∈ fv llsst A"
  shows "∃ n. I x = Fun (Val n) []" 
  using reachable_constraints_occurs_fact_send_ex[OF A_reach P x]
  reachable_constraints_occurs_fact_send_in_ik[OF A_reach I P]
  reachable_constraints_occurs_fact_ik_case[OF A_reach P]
  by fast

lemma constraint_model_Value_term_is_Val':
  assumes A_reach: "A ∈ reachable_constraints P"
  and I: "welltyped_constraint_model I A"
  and P: "∀ T ∈ set P. admissible_transaction T"
  and x: "(TAtom Value, m) ∈ fv llsst A"
  shows "∃ n. I (TAtom Value, m) = Fun (Val n) []" 
  using constraint_model_Value_term_is_Val[OF A_reach I P _ x] by simp

lemma constraint_model_Value_var_in_constr_prefix:
  assumes A_reach: "A ∈ reachable_constraints P"
  and I: "welltyped_constraint_model I A"
  and P: "∀ T ∈ set P. admissible_transaction T"
  shows "∀ x ∈ fv llsst A. Γ_v x = TAtom Value → (∃ B. prefix B A ∧ x /∈ fv llsst B ∧ I x ⊑ set trms llsst B)"
  (is "∀ x ∈ ?X A. ?R x → ?Q x A")
  using A_reach I
  proof (induction A rule: reachable_constraints.induct)
    case (step A T ξ σ α)
    let ?P = "∀ x. ?R x → ?Q x A"
    define T' where "T' ≡ dual llsst ⟨transaction_strand T · llsst ξ o_a σ o_a α⟩"
    have IH: "?P A" using step welltyped_constraint_model_prefix by fast
    note t_empty = admissible_transaction_decl_subst_empty[OF bspec[OF P step.hyps(2)] step.hyps(3)]
have $T_{adm}$: "admissible_transaction $T$" by (metis $P$ step.hyps(2))

note $T_{wf} = \text{admissible_transaction_is_wellformed_transaction}(T_{adm})$

have $I_{is_T_model}$: "strand sem stateful $(ik_{isst} A \cdot set I) \text{ (unlabel } T') I$" using step.prems unlabel_append[of $A T']$ db_ioset_is_dbupd[of "$\text{unlabel } A' I' [I']$"

and $I_{is_T_model}$: "strandsem_append_stateful(of $T'$) (of "$\text{unlabel } A' I' [I']$"

\text{by (simp add: $T'_\text{def}$ welltyped_constraint_model_def constraint_model_def db_ioset_def)}

have $I_{interp}$: "interpretation subst $I$"

and $I_{vt}$: "vt subst $I$"

and $I_{vt}$: "vt subst (sub range $I$)"

by (metis $I$ welltyped_constraint_model_def constraint_model_def, $I$ welltyped_constraint_model_def, metis $I$ welltyped_constraint_model_def constraint_model_def constraint_model_def db_ioset_def)

have 1: "?Q x $A$ when x: "x $\in fv_{st}$ $T''" $\Gamma_v x = T\text{Atom Value}" for x

\text{proof -}

obtain n where n: "$I x = Fun n []" "is Val n" "public n"

using constraint_model_Value_term_is_Val[

OF reachable_constraints.step[OF step.hyps] step.prems P x(2)]

x(1) $fv_{st}$ append[of "$\text{unlabel } A" "unlabel } T'"

unlabel_append[of "$\text{unlabel } A T'" $T'_\text{def}$]

by moura

have "$x \in fv_{st}$ (transaction strand $T'$) $\xi$ $\alpha$

using x(1) $fv_{st}$ unlabel dualisst_eq unfolding $T'_\text{def}$ by fastforce

then obtain y where y: "$y \in fv_{st}$ (transactionstrand $T''$) "$x \infv (\xi \alpha) y"

using $fv_{st}$ subst obtain var[of x "unlabel (transaction strand $T'$) $\xi \alpha"]

unlabel subst[of "transaction strand $T'$ $\xi \alpha"]

by auto

have $y_x: "(\xi \alpha) y = \text{Var } x"$ and $y_not_fresh: "y \notin set (transaction fresh $T')"

using y(2) transaction_decl_fresh_renaming_substs_range[OF step.hyps(3-5), of y]

by (force, fastforce)

have "$I ((\xi \alpha) y) = T\text{Atom Value}" using x(2) $y_x$ by simp

moreover have "$vt subst (\xi \alpha) y"

by (rule transaction_decl_fresh_renaming_substs wt[OF step.hyps(3-5)])

ultimately have $y_val$: "$\Gamma_v y = T\text{Atom Value}" by (metis $vt subst def I.$simp(1))

have "$\text{Fun } n [] = (?\xi \alpha) y . I" using n $y_x$ by simp

hence $y_n$: "$\text{Fun } n [] = (?\xi \alpha) y . I" $y$ by (metis subst subst compose[of "$\text{Var } y" (?\xi \alpha) I set apply_term.$simp(1))

have $A_{ik I}$ vals: "$\exists f. I x = \text{Fun } f []"

\text{proof -}

have "$\exists f. I (x) = \text{Var } a" when x $\in fv_{st}$ $A$ for x

using that reachable constraints vars $T\text{Atom typed}$[OF step.hyps(1) P, of x] vars_sst is $fv_{st}$ vars_sst[of "$\text{unlabel } A"] wt subst trm'[OF $I_wt$, of "$\text{Var } x"]

by force

hence "$f. I x = \text{Fun } f []" when x $\in fv_{st}$ $A$ for x

using wt trm subst[OF $I_wt$ wt forms, of "$\text{Var } x"] wt trm Var[of x] const type inv wt empty fv exists fun[OF interpretation grounds[OF $I interp$, of "$\text{Var } x"]]

by (metis subst apply term.$simp(1)[of x I])

thus $\text{thesis}$

using $fv_{ik subset}$ $fv_{st}$[of "$\text{unlabel } A"] vars_sst is $fv_{st}$ bvars_sst[of "$\text{unlabel } A"]

by blast

qed

hence $A_{subterms subst cong}$: "subterms $A (ik_{isst} A \cdot set I) = subterms $A (ik_{isst} A) \cdot set I$

by (metis $ik subset$ of "$\text{unlabel } A I" unlabel subst[of $A I$] subterms subst last $ik$[of $A I$])
have \( x \notin FV_{\text{set}}(A) \)

proof -

have "\( x \in FV_{\text{set}}(\text{transaction_strand} T \cdot \text{ts} \xi \sigma \alpha \)" using \( x(1) FV_{\text{set}}(\text{unlabel} \cdot \text{transaction_strand} T \cdot \text{ts} \xi \sigma \alpha) \) unfolding \( T_{\text{def}} \) by fast

hence "\( x \in FV_{\text{set}}((\text{unlabel} \cdot \text{transaction_strand} T) \cdot \text{ts} \xi \sigma \alpha) \)"

using transaction_fresh_subst_grounds_domain[OF step.hyps(4)] step.hyps(4)

labeled_stateful_strand_subst_comp[of \( \sigma \) "\( \text{transaction_strand} T \sigma \)"
    unlabeled_subst[of "\( \text{transaction_strand} T \sigma \)"
    unlabeled_subst[of "\( \text{transaction_strand} T \sigma \)"

by (simp add: \( \xi \cdot \text{empty transaction_fresh_subst_def range_vars_alt_def} \)

then obtain \( y \) where "\( \alpha \cdot y = \text{Var } x \)"

using transaction_renaming_subst_var_obtain(1)[OF step.hyps(5)] by blast

thus ?thesis

using transaction_renaming_subst_range_notin_vars[OF step.hyps(5), of \( y \)]

vars_is_fv set_bvars(set[of "\( \text{unlabel} A \)"

by auto

qed

from admissible_transaction_fv_in_receives_or_selects[OF \( T_{\text{adm}}(y(1) y_{\text{not fresh}}) \)

have n_cases: "Fun n \( \{} \subseteq_{\text{set}} \text{trms}_{\text{set}} A \lor (\exists z \in FV_{\text{set}} A. \Gamma_v z = TAtom Value \land I z = \text{Fun n } \{\}) \)"

proof

assume y_in: "\( y \in FV_{\text{set}}(\text{transaction_receive} T) \)"

then obtain ts where ts: "\( \text{receive}(ts) \in \text{set} \cdot (\text{unlabel} \cdot \text{transaction_receive} T) \)"

"\( y \in FV_{\text{set}}(\text{set} ts) \)"

using admissible_transaction_strand_step_cases(1)[OF \( T_{\text{adm}} \)]

by force

hence ts_deduct: "\( \text{list_all} \cdot (\lambda t. ik_{\text{set}} A \cdot \text{ts} t \cdot \xi \sigma \alpha \cdot I) \) ts"

using wellformed_transaction_sems_receives[

OF \( T_{\text{wf}}, \) of "\( ik_{\text{set}} A \cdot \text{ts} t \cdot \xi \sigma \alpha \cdot I) \) ts 
    set (db_{\text{set}} I) I \cdot \text{ts} t \cdot \xi \sigma \alpha \cdot I) \)

\( I_{\text{is}_T_{\text{model}}}

\( \text{subst}_{\text{set}} \cdot \text{unlabel} \cdot \text{member} \cdot (\text{of} \cdot \text{receive}(ts) \cdot \text{transaction_receive} T \cdot \xi \sigma \alpha) \)

unfolding \( T_{\text{def}} \) list_all_iff by force

obtain ty where ty: "\( ty \in \text{set} ts \)" 

"\( y \in FV ty \)" using ts(2) by fastforce

have "Fun n \( \{} \subseteq_{\text{set}} ik_{\text{set}} A \lor (\exists z \in FV_{\text{set}} (ik_{\text{set}} A). \Gamma_v z = TAtom Value \land I z = \text{Fun n } \{\}) \)"

proof -

have "Fun n \( \{} \subseteq_{\text{ty} \cdot \xi \sigma \alpha \cdot I} \)"

using imageI[of "\( \text{Var } y \)" "\( \text{subterms } ty \)" "\( \langle t, ik_{\text{set}} A \cdot \xi \sigma \alpha \cdot I \rangle \)"

var_is_subterm[OF ty(2)] subterms_subst_subset[of "\( \xi \sigma \alpha \cdot I \) ty"

subst_subst_compose[of ty "\( \xi \sigma \alpha \cdot I \) y n"

by (auto simp del: subst_subst_compose)

hence "Fun n \( \{} \subseteq_{\text{ty} \cdot \xi \sigma \alpha \cdot I} \)"

using ty(1) private_fun_deduct_in_ik[of _ _ "\{\}"] n(2,3) ts_deduct

unfolding is_Val_def is_Abs_def list_all_iff by fast

hence "Fun n \( \{} \subseteq_{\text{ty} \cdot \xi \sigma \alpha \cdot I} \) 
    "(\exists z \in FV_{\text{set}} (ik_{\text{set}} A). I z = \text{Fun n } \{\})"

using const_subterm_subst_cases[of n _ I] A_ik_I_vals by fastforce

thus ?thesis

using I_is_T_wf n(2) unfolding wt_subst_def is_Val_def is_Abs_def by force

qed

thus ?thesis

using fv_ik_subset_fv_sat' ik_ik_set_\text{trms}_{\text{set}}_{\text{set}}[of "\( \text{unlabel} A \)"

A_subterms_subst_cong by fast

next

assume y_in: "\( y \in FV_{\text{set}}(\text{transaction_checks} T) \land \)

(\exists t s. select(t, s) \in \text{set} \cdot (\text{unlabel} \cdot \text{transaction_checks} T) \land y \in FV t \cup FV s)"

then obtain s where s: "\( \text{select}(\text{Var } y, \text{Fun } (\text{Set } s) \{\}) \in \text{set} \cdot (\text{unlabel} \cdot \text{transaction_checks} T) \)"

using admissible_transaction_strand_step_cases(2)[OF \( T_{\text{adm}} \)] by force

hence "\( \text{select}(\xi \sigma \alpha, y, \text{Fun } (\text{Set } s) \{\}) \in \text{set} \cdot (\text{unlabel} \cdot \text{transaction_checks} T \cdot \text{ts} \xi \sigma \alpha) \)"

using subst_last_unlabel_member by fastforce

hence n_in_db: "\( \text{Fun } n \{\}, \text{Fun } (\text{Set } s) \{\} \in \text{set} \cdot (\text{db}_{\text{set}} \cdot (\text{unlabel} A) I \{\})\)"
using wellformed_transaction_sem_pos_checks
OF T wf, of "ik lsst A I" "set (db lsst A I)" "\xi o_\sigma o_\alpha I"
assign "(\xi o_\sigma o_\alpha y) "Fun (Set s) []"

_I is_T_model n y x
unfolding T'_def db sts_def by fastforce
obtain tn sn where tsn: "insert(tn,sn) \in set (unlabel A)" "Fun n [] = tn \cdot I"
using db sts_in_cases[OF n_in_db] by force
have "Fun n [] = tn \lor (\exists z. \Gamma v z = TAtom Value \land tn = Var z)"
using n wt tsn(2) n(2) unfolding wt subst_def is_Val_def is_Abs_def by (cases tn) auto
moreover have "tn \in subterms set (trms lsst A)" "fv tn \subseteq fv lsst A"
using tsn(1) in_subterms_Union by force+
ultimately show ?thesis using tsn(2) by auto
qed

from n_cases show ?thesis
proof assume "\exists x \in fv lsst (A @ T'). \Gamma v x = TAtom Value" show "?Q x (A @ T')"
proof (cases "x \in fv lsst A")
case False hence "x \in fv lsst T'" using x(1) unlabel_append[of A] fv lsst_append[of "unlabel A"] by simp
then obtain B where: "prefix B A" "Fun n [] \in subterms set (trms lsst B)"
using x(2) 1 by moura
thus ?thesis using prefix_prefix by fast
qed (use x(2) IH prefix_prefix in fast)
qed thus ?case unfolding T'_def by blast
qed simp

lemma constraint_model_Val_const_in_constr_prefix:
assumes A_reach: "A \in reachable_constraints P"
and I: "welltyped_constraint_model I A"
and P: "\forall T \in set P. wellformed_transaction T"
"\forall T \in set P. admissible_transaction_terms T"
and n: "Fun (Val n) [] \subseteq ik lsst A \cdot set I"
shows "Fun (Val n) [] \subseteq set trms lsst A"
proof
have "wf lsst (unlabel A)"
"constr_sem_stateful I (unlabel A)"
"interpretation_subst I"
"wf trms (subst_range I)"
"wf subst I"
using reachable_constraints_wf[of P(1) _ A_reach]
admissible_transaction_terms_wf[of trms, I P(2) n]
unfolding welltyped_constraint_model_def constraint_model_def wf trms code by fast+

show ?thesis
using constraint_model_priv_const_in_constr_prefix[of * _ _ n]
by simp
3.3 Stateful Protocol Model

lemma constraint_model_Val_const_in_constr_prefix':
assumes A_reach: "A ∈ reachable_constraints P"
and I: "welltyped_constraint_model I A"
and P: "∀ T ∈ set P. admissible_transaction T" 
and n: "Fun (Val n) [] ⊑ set ik lst A · set I"
shows "Fun (Val n) [] ⊑ set trms lst A"
using constraint_model_Val_const_in_constr_prefix[OF A_reach I _ _ n] 
proof
by fast

lemma constraint_model_Value_in_constr_prefix_fresh_action':
fixes P::"('fun, 'atom, 'sets, 'lbl) prot_transaction list"
assumes A: "A ∈ reachable_constraints P"
and P: "∀ T ∈ set P. admissible_transaction_terms T" 
and ∀ T ∈ set P. transaction_decl T () = []" 
and ∀ T ∈ set P. bvars_transaction T = {}" 
and n: "Fun (Val n) [] ⊑ set trms lst A"
obtains B T ξ σ α where
"prefix (B@dual lst (transaction_strand T · ξ o σ o α )) A" 
and "T ∈ set P" "transaction_decl_subst ξ T B" "transaction_fresh_subst σ T B" 
and "Fun (Val n) [] ∈ subst_range σ"
proof -

obtain T whereTs:A = g Ts "∀ B. prefix B Ts → g B ∈ reachable_constraints P" 
"∀ B T ξ σ α. prefix (B@[T, ξ, σ, α])) Ts → T ∈ set P ∧ transaction_decl_subst ξ T ∧ 
transaction_fresh_subst σ T (g B) ∧ transaction_renaming_subst α P (g B)"
using reachable_constraints_as_transaction_lists[OF A] unfolding g_def f_def by blast

obtain T ξ σ α where T:"(T, ξ, σ, α) ∈ set Ts" "Fun (Val n) [] ⊑ set trms lst (transaction_strand T · ξ o σ o α)"
using n trms lst_unlabel_dual lst_eq unfolding Ts(1) g_def f_def unlabel_def trms lst_def 
proof -

obtain B where B: "prefix (B@[T, ξ, σ, α])) Ts" "g B ∈ reachable_constraints P" "T ∈ set P" 
"transaction_decl_subst ξ T" "transaction_fresh_subst σ T (g B)" 
"transaction_renaming_subst α P (g B)"
proof -

obtain B where B:"∃ C. B@[T, ξ, σ, α]#C = Ts" by (metis T(1) split_list)
thus ?thesis using Ts(2-) that[of B] by auto
qed

note T_adm_terms = bspec[OF P(1) B(3)]
note T_decl_empty = bspec[OF P(2) B(3)]
note T_no_bvars = bspec[OF P(3) B(3)]
note ξ_empty = admissible_transaction_decl_subst_empty'[OF T_decl_empty B(4)]

have "trms lst (transaction_strand T · ξ o σ o α) = trms_transaction T · ξ o σ o α" 
using trms lst_subst[of _ "ξ o σ o α"] T_no_bvars by blast
hence "Fun (Val n) [] ⊑ set trms_transaction T · ξ o σ o α"
by (metis T(2) unlabeled_subst)

hence "Fun (Val n) [] ⊑ subst_range (ξ ◦ s σ ◦ s α)"

using admissible_transaction_terms_no_Value_consts(1) [OF T_adm_terms]

const_subterms_subst_cases' [of "Fun (Val n) [] ⊑ subst_range (ξ ◦ s σ ◦ s α)"

by blast

then obtain tn where

then obtain tn where "tn ∈ subst_range (ξ ◦ s σ ◦ s α)"

"Fun (Val n) [] ⊑ tn" "is_Fun tn"

by fastforce

have "Fun (Val n) [] ∈ subst_range σ" using tn(1) transaction_decl_fresh_renaming_substs_range'(2,4) [OF B(4-6) tn(1) ξ_empty]

by fastforce

moreover have "prefix (g B@dual lsst (transaction_strand T · lsst ξ ◦ s σ ◦ s α)) A"

using Ts(1) B(1) unfolding g_def f_def prefix_def by fastforce

ultimately show thesis using that B(2-) by blast

defined ϑ

have T_adm: "admissible_transaction T" using P B(3)

by blast

hence T_wf: "wellformed_transaction T" "admissible_transaction_occurs_checks T"

using admissible_transaction_is_wellformed_transaction(1,5) by (blast, blast)

obtain x where "x ∈ set (transaction_fresh T)"

"ϑ x = Fun (Val n) []"

using transaction_fresh_subst_domain [OF B(5)] B(7)

admissible_transaction_decl_subst_empty [OF T_adm B(4)]
by (force simp add: subst_compose ϑ_def)

obtain ts where ts: "send(ts) ∈ set (unlabel (transaction_send T))" "occurs (Var x) ∈ set ts"
using admissible_transaction_occurs_checksE2[OF T_wf(2) x(1)]
by (metis (mono_tags, lifting) list.set_intros(1) unlabel_Cons(1))

have "occurs (Var x) ∈ trms lsst (transaction_send T)":
using dual_transaction_ik_is_transaction_send'[OF T_wf(1), of ϑ]
by fast
hence "occurs (Fun (Val n) []) ∈ ik lsst (dual lsst (transaction_strand T · lsst ϑ))"
using x(2) by simp
thus ?thesis
using B(1)[unfolded ϑ_def[symmetric]
unlabel_append[of B "dual lsst (transaction_strand T · lsst ϑ)"
iksst _append[of "unlabel B" "unlabel (dual lsst (transaction_strand T · lsst ϑ))"]
unfolding prefix_def by force
qed

lemma admissible_transaction_occurs_checks_prop:
assumes A: A_reach: "A ∈ reachable_constraints P"
and I: I_welltyped: "welltyped_constraint_model I A"
and P: P: ∀ T ∈ set P. admissible_transaction T
and f: f ∈ Ω: "f ∈ ∪ (funs_term ` (I ` fv lsst A))"
shows "¬ is_PubConstValue f" and "¬ is_Abs f"
proof (cases "Γ v x = TAtom AbsValue"
  cases "Γ v x = TAtom Value"
l by simp)
have 1: "I (Var x) = Γ (I x)" using wt_subst_trm'[OF I_wt, of "Var x"] by simp
hence "Γ v x = Var a" using 0 by force
hence "f. I x = Fun f []" using x(1) wt_trm_subst[of I_wt, of "Var x"]
empty_fv_exists_fun[of "interpretation_grounds[OF I_inter, of "Var x"]]
by (metis subst_apply_term.simps(1)[of x I])
hence 2: "I x = Fun f []" using x(2) by force
hence 3: "Γ v x ≠ TAtom AbsValue" using 0 by force
hence "¬ is_PubConstValue f ∧ ¬ is_Abs f" by force
using reachable_constraints_val_funs_private[OF A_reach P]
by blast

next
case False thus ?thesis using x 1 2 3 unfolding is_PubConstValue_def by (cases f) auto
qed
thus "¬is_PubConstValue f" "¬is_Abs f" by metis+
qed

lemma admissible_transaction_occurs_checks_prop':
assumes A_reach: "A ∈ reachable_constraints P"
and I: "welltyped_constraint_model I A"
and P: ∀T ∈ set P. admissible_transaction T
and f: "f ∈ ∪ (funs_term ` (I`fv lsst A))"
shows "¬∃n. f = PubConstValue n"
and "¬∃n. f = Abs n"
using admissible_transaction_occurs_checks_prop[OF A_reach I P f]
unfolding is_PubConstValue_def by auto

lemma transaction_var_becomes_Val:
assumes A_reach: "A @dual lsst (transaction_strand T ` lsst ξ ◦ s σ ◦ s α) ∈ reachable_constraints P"
and I: "welltyped_constraint_model I (A@dual lsst (transaction_strand T ` lsst ξ ◦ s σ ◦ s α))"
and ξ: "transaction_decl_subst ξ T"
and σ: "transaction_fresh_subst σ T A"
and α: "transaction_renaming_subst α P A"
and P: ∀T ∈ set P. admissible_transaction T
and T: "T ∈ set P"
and x: "x ∈ fv_transaction T" "fst x = TAtom Value"
shows "∃n. Fun (Val n) [] = (ξ ◦ s σ ◦ s α) x · I"
proof -
obtain m where m: "x = (TAtom Value, m)" by (metis x(2) eq_fst_iff)

note ξ_empty = admissible_transaction_decl_subst_empty[OF bspec[OF P T] ξ]

have x_not_bvar: "x ∉ bvars_transaction T" "fv ((ξ ◦ s σ ◦ s α) x) ∩ bvars_transaction T = {}"
using x(1) admissible_transactions_fv_bvars_disj[OF P]
transaction_decl_fresh_renaming_substs_vars_disj(2)[OF ξ σ α, of x]
vars lsst_is_fv lsst_bvars lsst[where "unlabel (transaction_strand T)"
by (blast, blast)

have σx_type: "Γ (σ x) = TAtom Value"
using σ x Γ v TAtom''(2)[of x] wt_subst_trm''[of σ "Var x"]
unfolding transaction_fresh_subst_def by simp

show ?thesis
proof (cases "x ∈ subst_domain σ")
case True
then obtain c where c: "σ x = Fun c []" "¬public c" "arity c = 0"
using σ unfolding transaction_fresh_subst_def by fastforce
then obtain n where n: "c = Val n" using σx_type by (cases c) (auto split: option.splits)
show ?thesis using c n subst_compose[of σ α x] ξ_empty by simp
next
case False
hence "σ x = Var x" by auto
then obtain n where n: "(σ ◦ α x) x = Var (TAtom Value, n)"
using m transaction_renaming_subst_is_renaming(1)[OF α] subst_compose[of σ α x] ξ_empty by simp


lemmas reachable_constraints_SMP_subset:
assumes A: "A ∈ reachable_constraints P"
shows "SMP (trms_Sset A) ⊆ SMP (⋃ T ∈ P. trms_transaction T)" (is "'?A A"")
and "SMP (pair ` setops_Sset (unlabel A)) ⊆ SMP (⋃ T∈P. pair ` setops_transaction T)" (is "'?B A")
proof -
have ?A A ∧ ?B A using A
proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α)
define T' where T' ≡ transaction_strand T · Sset ξ ⊡ σ ⊡ α
define M where M ≡ ⋃ T ∈ P. trms_transaction T
define N where N ≡ ⋃ T ∈ P. pair ` setops_transaction T
let ?P = "λ t. ∃ s δ. s ∈ M ∧ wt subst δ ∧ wf trms (subst_range δ) ∧ t = s · δ"
let ?Q = "λ t. ∃ s δ. s ∈ N ∧ wt subst δ ∧ wf trms (subst_range δ) ∧ t = s · δ"

have IH: "SMP (trms_Sset A) ⊆ SMP M" "SMP (pair ` setops_Sset (unlabel A)) ⊆ SMP N"
using step.IH by (metis M_def, metis N_def)

note ξσα_wt = transaction_decl_fresh_renaming_substs_wt[OF step.hyps(3-5)]
note ξσα_wf = transaction_decl_fresh_renaming_substs_range_wf_trms[OF step.hyps(3-5)]

have 0: "SMP (trms_Sset (A@dual_Sset T')) = SMP (trms_Sset A) ∪ SMP (trms_Sset T')"
"SMP (pair ` setops_Sset (unlabel (A@dual_Sset T'))) = SMP (pair ` setops_Sset (unlabel A)) ∪ SMP (pair ` setops_Sset (unlabel T'))"
using trms_Sset_unlabel_dual_Sset_eq[of T']
setops_Sset_unlabel_dual_Sset_eq[of T']
trms_Sset.Append[of "unlabel A" "unlabel (dual_Sset T')"]
setops_Sset.Append[of "unlabel A" "unlabel (dual_Sset T')"]
unlabel_append[of A "dual_Sset T'"]
image_Un[of pair "setops_Sset (unlabel A)" "setops_Sset (unlabel T')"]
SMP_union[of "trms_Sset A" "trms_Sset T'"]
SMP_union[of "pair ` setops_Sset (unlabel A)" "pair ` setops_Sset (unlabel T')"]
by argo+

have 1: "SMP (trms_Sset (T')) ⊆ SMP M"
proof (intro SMP_subset_I ballI)
fix t show "t ∈ trms_Sset T' =⇒ ?P t"
using trms_Sset_wt_subst_ex[OF ξσα_wt ξσα_wf, of t "unlabel (transaction_strand T)"]
unlabel_subst[of "transaction_strand T" "ξ o σ o α"] step.hyps(2)
unfolding T'_def M_def by auto
qed

have 2: "SMP (pair ` setops_Sset (unlabel T')) ⊆ SMP N"
proof (intro SMP_subset_I ballI)
fix t show "t ∈ pair ` setops_Sset (unlabel T') =⇒ ?Q t"
using setops_Sset_wt_subst_ex[OF ξσα_wt ξσα_wf, of t "unlabel (transaction_strand T)"]
unlabel_subst[of "transaction_strand T" "ξ o σ o α"] step.hyps(2)
unfolding T'_def N_def by auto
qed
lemma reachable_constraints_no_Pair_fun':
amssumes A: "A ∈ reachable_constraints P"
and P: "∀ T ∈ set P. ∀ x ∈ set (transaction_fresh T). Γ_v x = TAtom Value"
"∀ T ∈ set P. transaction_decl T () = []"
"∀ T ∈ set P. admissible_transaction_terms T"
"∀ T ∈ set P. ∀ x ∈ vars_transaction T. Γ_v x = TAtom Value ∨ (∃ a. Γ_v x = ⟨a⟩)
shows "Pair / ∈ \{ \bigcup (funs_term ` SMP (trms_sst A))\}"
using A
proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α)
define T' where "T' ≡ dual lsst (transaction_strand T · lsst ξ ◦ s σ ◦ s α)"
note T_fresh_type = bspec[OF P(1) step.hyps(2)]
note ξ_empty = admissible_transaction_decl_subst_empty'[OF bspec[OF P(2) step.hyps(2)] step.hyps(3)]
note T_adm_terms = bspec[OF P(3) step.hyps(2)]
note ξσα_wt = transaction_decl_fresh_renaming_substs_wt[OF step.hyps(3-5)]
note ξσα_wf = transaction_decl_fresh_renaming_substs_range_wf_trms[OF step.hyps(3-5)]
have 0: "SMP (trms_sst (A@T')) = SMP (trms_sst A) ∪ SMP (trms_sst T')"
using SMP_union[OF "trms_sst A" "trms_sst T'"
unlabel_append[of A T'] trms_sst_append[of "unlabel A" "unlabel T'"
by simp
have 1: "wf_trms (trms_sst T')"
using reachable_constraints_wf_trms[OF _ reachable_constraints.step[OF step.hyps]]
admissible_transaction_terms_wf_trms, P(3)
trms_sst_append[of "unlabel A"] unlabel_append[of A]
unfolding T'_def by force
have 2: "Pair / ∈ \{ \bigcup (funs_term ` (trms_transaction T))\}"
using T_adm_terms unfolding admissible_transaction_terms_def by blast
hence "Pair / ∈ funs_term t"
when t: "t ∈ trms_sst (unlabel (transaction_strand T · ξ ◦ s σ ◦ s α))" for t
using 2 trms_sst_funs_term_cases[OF t]
by force
hence 3: "Pair / ∈ funs_term t"
when t: "t ∈ trms_sst T'" for t
using t unlabel_sub[of "transaction_strand T · ξ ◦ s σ ◦ s α"
trms_sst_unlabel_dual_sst_eq[of "transaction_strand T · ξ ◦ s σ ◦ s α"]
unfolding T'_def by metis
have "∃ a. Γ_v x = TAtom a" when "x ∈ vars_transaction T" for x
using that protocol_transaction_vars_TAtom_typed(1) bspec[OF P(4) step.hyps(2)]
by fast
hence "∃ a. Γ_v x = TAtom a" when "x ∈ vars_sst (unlabel (transaction_strand T · ξ ◦ s σ ◦ s α))" for x
using wt_subst_fv_termtype_subterm[OF _ ξσα_wt ξσα_wf, of x "vars_transaction T"]
vars_sst_subst_cases[OF that]
by fastforce
hence "∃ a. Γ_v x = TAtom a" when "x ∈ vars_sst T'" for x
using that unlabel_sub[of "transaction_strand T · ξ ◦ s σ ◦ s α"]
vars_sst_unlabel_dual_sst_eq[of "transaction_strand T · ξ ◦ s σ ◦ s α"]
unfolding T'_def by simp
hence "\exists a. \Gamma_v x = TAtom a" when "x \in f_{vset} (trms_{lsst} T')" for x using that f_{vset\_subset}(1) by fast

hence "Pair \notin \text{funs\_term} (\Gamma (\text{Var} x))" when "x \in f_{vset} (trms_{lsst} T')" for x using that by fastforce

moreover have "Pair \notin \text{funs\_term} s"
when s: "Ana s = (K, M)" for s::"('fun,'atom,'sets,'lbl) prot_term"
proof (cases s)
  case (Fun f S) thus ?thesis
    using s Ana_Fu_keys_not_pairs[of _ S K M] by (cases f) force+
qed simp

ultimately have "Pair \notin \text{funs\_term} t"
when t: "t \in \text{SMP (trms_{lsst} A)}" for t
using t 3 SMP_funs_term[of Pair] by fastforce

thus ?case using 0 step.IH(1) unfolding T'_def by blast
qed simp

lemma reachable_constraints_no_Pair_fun:
  assumes A: "A \in \text{reachable\_constraints P}"
  and P: "\forall T \in \text{set P. admissible\_transaction T}"
  shows "Pair \notin \bigcup (\text{funs\_term \_ SMP (trms_{lsst} A)})"
using reachable_constraints_no_Pair_fun'[OF A]
  P admissible_transactionE(1,2,3)
  admissible_transaction_is_wellformed_transaction(4)
by blast

lemma reachable_constraints_setops_form:
  assumes A: "A \in \text{reachable\_constraints P}"
  and P: "\forall T \in \text{set P. admissible\_transaction T}"
  and t: "t \in \text{pair \_ setops_{sst} (unlabel A)}"
  shows "\exists c s. t = pair (c, \text{Fun (Set s)}) \land \Gamma c = TAtom Value"
using A t
proof (induction A rule: reachable_constraints.induct)
  case (step A T \xi \sigma \alpha)
  have T_adm: "admissible\_transaction T" when "T \in \text{set P}" for T
    using P that unfolding list_all_iff by simp
  note T_adm' = admissible_transaction_is_wellformed_transaction(2,3)[OF T_adm]
  note T_wf = admissible_transaction_is_wellformed_transaction(1)[OF T_adm]
  note \xi\sigma\alpha_wt = transaction_decl_fresh_renaming_substs_wt[of \xi\sigma\alpha\wt]
  note \xi\sigma\alpha_wf = transaction_decl_fresh_renaming_substs_range_wf_trms[of \xi\sigma\alpha\wf]
  show ?case using step.IH
    proof (cases "t \in \text{pair \_ setops_{sst} (unlabel A)}")
      case False
      hence "t \in \text{pair \_ setops_{sst} (unlabel (transaction\_strand T) \_sst \xi o_s o_s \alpha)}"
        using step.prems setops_{sst}_append unlabel_append
        setops_{sst} unlabel_dual_{sst} eq[of "transaction\_strand T \_sst \xi o_s o_s \alpha"
        unlabel_subst[of "transaction\_strand T" \_sst \xi o_s o_s \alpha"]
        by fastforce
      then obtain t' \delta where t':
        "t' \in \text{pair \_ setops_{sst} (unlabel (transaction\_strand T))}"
        "\text{\_sst \_sst _sst \_sst _sst\_sst eq [OF \xi o_s o_s \alpha\wt \xi o_s o_s \alpha\wf]}" by blast
      then obtain s s' where s: "t' = pair (s, s')"
        using setops_{sst} are_pairs by fastforce
      moreover have "InSet ac s s' = InSet Assign s s' \lor InSet ac s s' = InSet Check s s'" for ac
        by (cases ac) simp_all
      ultimately have "\exists n. s = \text{Var (Var Value, n)}" "\exists u. s' = \text{Fun (Set u) []}"
        using t'(1) setops_{sst} member_iff[of s s' "unlabel (transaction\_strand T)"]
        pair_in_pair_image_iff[of s s']
      transaction_inserts_are_Value_vars[OF T_wf[of step.hyps(2)] T_adm'[OF step.hyps(2)], of s s']
transaction_deletes_are_Value vars[ 
  OF T_wf[OF step.hyps(2)] T_adm'(2)[OF step.hyps(2)], of s s']
transaction_selects_are_Value vars[ 
  OF T_wf[OF step.hyps(2)] T_adm'(1)[OF step.hyps(2)], of s s']
transaction_inset_checks_are_Value vars[ 
  OF T_adm[OF step.hyps(2)], of s s']
transaction_notinset_checks_are_Value vars[ 
  OF T_adm[OF step.hyps(2)], of _ _ _ s s']

by sets
then obtain ss n where ss: "t = pair (δ (Var Value, n), Fun (Set ss) [])" 
using t'(4) s unfolding pair_def by force
have "Γ (δ (Var Value, n)) = TAtom Value" "wf trm (δ (Var Value, n))" 
using reachable_constraints_setops_form[OF t'(2) wt_subst_trm''[OF t'(2), of "Var (Var Value, n)"]] 
apply simp using t'(3) by (cases "(Var Value, n) ∈ subst_domain δ") auto 
thus ?thesis using ss unfolding pair_def by fastforce
qed simp
qed (simp add: setops_sst_def)

lemma reachable_constraints_setops_type:
  fixes t::"('fun,'atom,'sets,'lbl) prot_term"
  assumes A: "A ∈ reachable_constraints P" 
  and P: "∀ T ∈ set P. admissible_transaction T" 
  and t: "t ∈ pair ` setops_sst (unlabel A)" 
  shows "Γ t = TComp Pair [TAtom Value, TAtom SetType]"
proof -
  obtain s c where s: "t = pair (c, Fun (Set s) [])" "Γ c = TAtom Value" 
  using reachable_constraints_setops_form[OF A P t] by moura
  hence "Fun (Set s) [] ::('fun,'atom,'sets,'lbl) prot_term ∈ trms lsst A" 
  using t setops_sst_member_iff[of c "Fun (Set s) []" "unlabel A"] by force
  hence "wf trm (Fun (Set s) [])" 
  using reachable_constraints_wf(2) P A admissible_transaction_is_wellformed_transaction(1,4) 
  unfolding admissible_transaction_terms_def by blast
  hence "arity (Set s) = 0" 
  unfolding wf_trm_def by simp
  thus ?thesis using s unfolding pair_def by fastforce
qed

lemma reachable_constraints_setops_same_type_if_unifiable:
  assumes A: "A ∈ reachable_constraints P" 
  and P: "∀ T ∈ set P. admissible_transaction T" 
  shows "∀ s ∈ pair ` setops_sst (unlabel A). ∀ t ∈ pair ` setops_sst (unlabel A). 
  (∃ δ. Unifier δ s t) −→ Γ s = Γ t" 
(is "?P A")
using reachable_constraints_setops_type[OF A P] by simp

lemma reachable_constraints_setops_unifiable_if_wt_instance_unifiable:
  assumes A: "A ∈ reachable_constraints P" 
  and P: "∀ T ∈ set P. admissible_transaction T" 
  shows "∀ s ∈ pair ` setops_sst (unlabel A). ∀ t ∈ pair ` setops_sst (unlabel A). 
  (∃ σ δ. Unifier δ s t) −→ (∃ σ. Unifier δ s t)" 
(proof (intro ballI impI) 
fix s t assume st: "s ∈ pair ` setops_sst (unlabel A)" "t ∈ pair ` setops_sst (unlabel A)" and 
  "∃ σ δ. Unifier δ s t" 
then obtain σ δ where σ:
  "Unifier δ s t" 
by moura
obtain fs ft cs ct where c:
3.3 Stateful Protocol Model

\[
s = \text{pair } (cs, \text{Fun } (\text{Set } fs) [\]) \quad t = \text{pair } (ct, \text{Fun } (\text{Set } ft) [\])
\]

Γ \text{cs} = T\text{Atom Value} \quad Γ \text{ct} = T\text{Atom Value}

using reachable_constraints_setops_form[OF A P st(1)]
reachable_constraints_setops_form[OF A P st(2)]
by moura

have \( cs \in \text{subterms}_{\text{set}} (\text{trms}_{\text{lsst}} A) \) \( ct \in \text{subterms}_{\text{set}} (\text{trms}_{\text{lsst}} A) \)
using c(1,2) setops_subterm_trms[OF st(1), of cs] setops_subterm_trms[OF st(2), of ct]
Fun_param_is_subterm[of cs "args s"] Fun_param_is_subterm[of ct "args t"]
unfolding pair_def by simp_all
moreover have
\[
\forall T \in \text{set } P. \text{wellformed_transaction } T
\quad \forall T \in \text{set } P. \text{wf_trms arity (trms_transaction } T)\]
using P admissible_transaction_is_wellformed_transaction(1,4)
unfolding admissible_transaction_terms_def by fast+
ultimately have *: \( \text{wf_trm cs} \) \( \text{wf_trm ct} \)
using reachable_constraints_wf(2)[OF _ _ A] wf_trms_subterms by blast+

have \( \exists x. cs = \text{Var } x \) \( \exists x. ct = \text{Var } x \)
using const_type_inv_wf c(3) *(1) by (cases cs) auto
moreover have \( \exists x. ct = \text{Var } x \) \( \exists x. ct = \text{Var } x \)
using const_type_inv_wf c(4) *(2) by (cases ct) auto
ultimately show \( \exists \delta. \text{Unifier } \delta s t \)
using reachable_constraints_setops_form[OF A P] reachable_constraints_setops_type[OF A P] st σ c
unfolding pair_def by auto

qed

lemma reachable_constraints_tfr:
assumes M: \( M \equiv \bigcup T \in \text{set } P. \text{trms_transaction } T \)
\( \text{has_all_wt_instances_of } \Gamma M N \)
\( \text{finite } N \)
\( \text{tfr_set } N \)
\( \text{wf_trms } N \)
and P: \( \forall T \in \text{set } P. \text{admissible_transaction } T \)
\( \forall T \in \text{set } P. \text{list_all tfr_setp (unlabel (transaction_strand } T))\)
and A: \( A \in \text{reachable_constraints } P \)
shows \( \text{tfr_set (unlabel } A) \)
using A
proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α)
define T' where \( T' \equiv \text{dual}_{\text{lsst}} (\text{transaction_strand } T \cdot \text{lsst } ξ \circ σ \circ s α) \)
have AT'_reach: \( A@T' \in \text{reachable_constraints } P \)
using reachable_constraints.step[OF step.hyps] unfolding T'_def by metis

have T_adm: \( \text{admissible_transaction } T \)
using step.hyps(2) P(1) by blast
note ξ_empty = admissible_transaction_decl_subst_empty[OF T_adm step.hyps(3)]

note ξσoα_wt = transaction_decl_fresh_renaming_substs_wt[OF step.hyps(3-5)]

note ξσoα_wf = transaction_decl_fresh_renaming_substs_range_wf_trms[OF step.hyps(3-5)]

have ξσoα_bvars_disj: \( \text{bvars}_{\text{lsst}} (\text{transaction_strand } T) \cap \text{range_vars } (ξ o_σ o_s α) = {} \)
using transaction_decl_fresh_renaming_substs_vars_disj(4)[OF step.hyps(3,4,5,2)]
ξ_empty
by simp

have wf_trms_M: \( \text{wf_trms } M \)
using admissible_transactions_wf_trms P(1)
unfolding M(1) by blast
have "tfr_set (trms_set (A@T'))"
using reachable_constraints_SMP_subset[of AT'_reach]
tfr_subset[of M(4), of "trms (A@T')"
SMP_SMP_subset[of M N] SMP_I[of wf_trms_M M(5,2)]
unfolding M(1) by blast
moreover have "∀ p. Ana (pair p) = ([],[])" unfolding pair_def by auto
ultimately have 1: "tfr_set (trms lsst (A@T') ∪ pair ` setops_sst (unlabel (A@T')))"
using tfr_setops_if_tfr_trms[of "unlabel (A@T')"]
reachable_constraints_no_Pair_fun[of AT'_reach P(1)]
reachable_constraints_setops_same_type_if_unifiable[of AT'_reach P(1)]
reachable_constraints_setops_unifiable_if_wt_instance_unifiable[of AT'_reach P(1)]
by blast
have "list_all tfr_sstp (unlabel (transaction_strand T))"
using step.hyps(2) P(2) tfr_sstp_is_comp_tfr_sstp
unfolding comp_tfr_sstp_def tfr_sstp_def by fastforce
hence "list_all tfr_sstp (unlabel T')"
using step.IH unlabel_append
unfolding tfr_sstp_def by auto
have "tfr_sst (unlabel (A@T'))" using 1 2 by (metis tfr_sst_def)
thus ?case by (metis T'_def)
qed simp

lemma reachable_constraints_tfr':
assumes M:
"M ≡ ⋃ T ∈ set P. trms_transaction T ∪ pair ` setops_transaction T"
"has_all_wt_instances_of Γ M N"
"finite N"
"tfrset N"
"wf_trms M"
and P:
"∀ T ∈ set P. wf_trms' arity (trms_transaction T)"
"∀ T ∈ set P. list_all tfr_sstp (unlabel (transaction_strand T))"
and A: "A ∈ reachable_constraints P"
shows "tfr_set (unlabel A)"
using A
proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α)
define T' where "T' ≡ dual_sst (transaction_strand T · lsst ξ ◦ s σ ◦ s α)"
have AT'_reach: "A@T' ∈ reachable_constraints P"
using reachable_constraints.step[of step.hyps] unfolding T'_def by metis
note ξσα_wt = transaction_decl_fresh_renaming_substs_wt[of step.hyps(3-5)]
note ξσα_wf = transaction_decl_fresh_renaming_substs_range_wf_trms[of step.hyps(3-5)]
have ξσα_bvars_disj: "bvars_sst (transaction_strand T) ∩ range_vars (ξ ◦ s σ ◦ s α) = {}"
by (rule transaction_decl_fresh_renaming_substs_vars_disj[of step.hyps(3,4,5,2)])
have wf_trms_M: "wf_trms M"
using P(1) setops_sst_wf_trms(2) unfolding M(1) pair_code wf_trms_code[symmetric] by fast
have "SMP (trms_set (A@T')) ⊆ SMP M" "SMP (pair ` setops_set (unlabel (A@T'))) ⊆ SMP M"
using reachable_constraints_SMP_subset[of AT'_reach]
SMP_mono[of "⋃ T ∈ set P. trms_transaction T" M]
SMP_mono[of "⋃ T ∈ set P. pair ` setops_transaction T" M]
unfolding M(1) pair_code[symmetric] by blast+
hence 1: "tfr_set (trms_sst (A\$T')) \cup \text{pair} \ ` \text{setops_sst} (\text{unlabel} (A\$T'))" 
using tfr_subset(3)[OF M(4), of "trms_sst (A\$T') \cup \text{pair} \ ` \text{setops_sst} (\text{unlabel} (A\$T'))"]
SHP_union[of "trms_sst (A\$T')" "pair \ ` \text{setops_sst} (\text{unlabel} (A\$T'))"]
SHP_SMP_subset[of "M N"] SMP_I'[OF wf_trms_M M(5,2)]
by blast

have "list_all tfr_sstp (\text{unlabel} (\text{transaction_strand} T'))" 
using step_hyps(2) P(2) tfr_sstp_is_comp_tfr_sstp
unfolding comp_tfr_sst_def tfr_sst_def by fastforce

hence "list_all tfr_sstp (unlabel T')"
using tfr_sstp_all_wt_subst_apply[OF _ T \in \text{wt_subst} \sigma \sigma T \in \text{transaction_strand} T \in \text{set} \text{P} \in \text{wellformed_transaction} T"
unlabel_subset[of "\text{transaction_strand} T" "\text{set} \text{P}" "\text{wellformed_transaction} T" "\text{set} \text{P}""]
unfolding T'_def by argo

hence 2: "list_all tfr_sstp (\text{unlabel} (A\$T'))" 
using step.1H unlabel_append
unfolding tfr_sst_def by auto

have "tfr_sst (\text{unlabel} (A\$T'))" using 1 2 by (metis tfr_sst_def)
thus \texttt{case} by (metis T'_def)
qed simp

lemma reachable_constraints_typing_ctxt:
assumes M:
"M \equiv \bigcup T \in \text{set} P. \text{trms_transaction} T \cup \text{pair} \ ` \text{setops_transaction} T"
"\text{has_all_wt_instances_of} \Gamma M N"
"finite N"
"tfr_sst N"
"wf_trms N"
and P:
"\forall T \in \text{set} P. \text{wellformed_transaction} T"
"\forall T \in \text{set} P. \text{arity (trms_transaction T)}"
"\forall T \in \text{set} P. \text{list_all tfr_sstp (unlabel (transaction_strand T))}"
and A: "A \in \text{reachable_constraints} P"
shows "typing_ctxt (unlabel A)"
using reachable_constraints_wf[OF P(1,2) \text{A}] reachable_constraints_tfr'[OF M P(2,3) \text{A}]
unfolding typing_ctxt_def by blast

context
begin

private lemma reachable_constraints_typing_result_aux:
assumes 0: "wf_sst (\text{unlabel} \text{A})" "tfr_sst (\text{unlabel} \text{A})" "wf_trms (trms_sst \text{A})"
shows "wf_sst (unlabel (A\$T(1,send(\text{attack}(n))))))" "tfr_sst (unlabel (A\$T(1,send(\text{attack}(n))))))")" "wf_trms (trms_sst (A\$T(1,send(\text{attack}(n))))))")"

proof -
let ?n = "[(1,send(\text{attack}(n)))]"
let ?A = "A\$?n"

show "wf_sst (\text{unlabel} \text{A})"
using 0(1) wf_sst_append_suffix'[of "{}" "\text{unlabel} \text{A}" "\text{unlabel} \text{?n}\] unlabel_append[of \text{A} \text{?n}]
by simp

show "wf_trms (trms_sst ?A)"
using 0(3) trms_sst_append[of "\text{unlabel} \text{A}" "\text{unlabel} \text{?n}\] unlabel_append[of \text{A} \text{?n}]
by fastforce

have "\forall t \in trms_sst ?n \cup \text{pair} \ ` \text{setops_sst} (\text{unlabel} \text{?n}). \exists c. t = Fun c []" 
"\forall t \in trms_sst ?n \cup \text{pair} \ ` \text{setops_sst} (\text{unlabel} \text{?n}). \text{Ana} t = ([],[])"
by (simp_all add: setops_sst_def)

hence "tfr_sst (trms_sst \text{A} \cup \text{pair} \ ` \text{setops_sst} (\text{unlabel} \text{A}) \cup 
(trms_sst ?n \cup \text{pair} \ ` \text{setops_sst} (\text{unlabel} ?n)))"
using 0(2) tfr_consts_mono unfolding tfr_sst_def by blast
hence "tfr_sst (trms_sst (A\$?n) \cup \text{pair} \ ` \text{setops_sst} (\text{unlabel} (A\$?n)))"
using unlabel_append[of A ?n] trms_set_append[of "unlabel A" "unlabel ?n"]
setops_set_append[of "unlabel A" "unlabel ?n"]
by (simp add: setops_set_def)
thus "tfr_set (unlabel A)"
using 0(2) unlabel_append[of ?A ?n]
unfolding tfr_set_def by auto
qed

lemma reachable_constraints_typing_result:
  fixes P
  assumes M:  
    "has_all_wt_instances_of Γ (⋃T ∈ set P. trms_transaction T) N"
    "finite N"
    "tfr_set N"
    "wftrms N"
  and P:  
    "∀ T ∈ set P. wellformed_transaction T"
    "∀ T ∈ set P. list_all tfrstp (unlabel (transaction_strand T))"
  and A:  
    "A ∈ reachable_constraints P"
  and I: "constraint_model I (A@[l, send⟨[attack n]⟩])"
  shows "∃I. welltyped_constraint_model I (A@[(l, send⟨[attack n]⟩)])"
proof -
  have I:  
    "interpretation subst I "wftrms (subst_range I)"
    "constr_sem_stateful I (unlabel (A@[l, send⟨[attack n]⟩]))"
  unfolding constraint_model_def by metis+
  have "∀ T ∈ set P. wellformed_transaction T"
    "∀ T ∈ set P. admissible_transaction_terms T"
  using P(1,2) admissible_transaction_is_wellformed_transaction(1,4) by blast+
  moreover have "∀ T ∈ set P. wftrms' arity (trms_transaction T)"
    "∀ T ∈ set P. list_all tfrstp (unlabel (transaction_strand T))"
  using P(1,2) admissible_transaction_is_wellformed_transaction(4)
  unfolding admissible_transaction_terms_def by blast
  ultimately have 0: "wf_set (unlabel A)" "tfr_set (unlabel A)" "wftrms (trms_set A)"
  using reachable_constraints_tfr[OF _ M P A] reachable_constraints_wf[OF _ _ A]
  by metis+
  show ?thesis
  using stateful_typing_result[OF reachable_constraints_typing_result_aux[OF 0] I(1,3)]
  by (metis welltyped_constraint_model_def constraint_model_def)
qed

lemma reachable_constraints_typing_result':
  fixes P
  assumes M:  
    "M ≡ ⋃T ∈ set P. trms_transaction T ∪ pair' Pair ` setops_transaction T"
    "has_all_wt_instances_of Γ M N"
    "finite N"
    "tfr_set N"
    "wftrms N"
  and P:  
    "∀ T ∈ set P. wellformed_transaction T"
    "∀ T ∈ set P. wftrms' arity (trms_transaction T)"
    "∀ T ∈ set P. list_all tfrstp (unlabel (transaction_strand T))"
  and A:  
    "A ∈ reachable_constraints P"
  and I: "constraint_model I (A@[l, send⟨[attack n]⟩])"
  shows "∃I. welltyped_constraint_model I (A@[(l, send⟨[attack n]⟩)])"
proof -
  have I:  
    "interpretation subst I "wftrms (subst_range I)"
    "constr_sem_stateful I (unlabel (A@[l, send⟨[attack n]⟩]))"
  unfolding constraint_model_def by metis+
  have 0: "wf_set (unlabel A)" "tfr_set (unlabel A)" "wftrms (trms_set A)"
  using stateful_typing_result[OF reachable_constraints_typing_result_aux[OF 0] I(1,3)]
  by (metis welltyped_constraint_model_def constraint_model_def)
qed
3.3 Stateful Protocol Model

using reachable_constraints_tfr[OF P (2-3) A]
reachable_constraints_wf[OF P (1,2) A]
by metis+

show ?thesis
using stateful_typing_result[OF reachable_constraints_typing_result_aux[OF 0] I(1,3)]
by (metis welltyped_constraint_model_def constraint_model_def)
qed

end

lemma reachable_constraints_transaction_proj:
assumes "A ∈ reachable_constraints P"
shows "proj n A ∈ reachable_constraints (map (transaction_proj n) P)"
using assms
proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α)
show ?case
using step.hyps(2)
by (simp add: proj_dual lsst proj_subst transaction_strand_proj)
qed (simp add: reachable_constraints.init)

context
begin
private lemma reachable_constraints_par_comp
fixes P
defines "Ts ≡ concat (map transaction_strand P)"
assumes A: "A ∈ reachable_constraints P"
shows "∀ b ∈ set (dual lsst A). ∃ a ∈ set Ts. ∃ δ. b = a · lsstp δ ∧ wt subst δ ∧ wf trms (subst_range δ) ∧ (∀ t ∈ subst_range δ. (∃ x. t = Var x) ∨ (∃ c. t = Fun c []))"
(is "∀ b ∈ set (dual lsst A). ∃ a ∈ set Ts. ?P b a")
using A
proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α)
define Q where "Q ≡ ?P"
define ϑ where "ϑ ≡ ξ o s σ o s α"
let ?R = "λ A Ts. ∀ b ∈ set A. ∃ a ∈ set Ts. Q b a"
have "wt subst ϑ" "wf trms (subst_range ϑ)"
"∀ t ∈ subst_range ϑ. (∃ x. t = Var x) ∨ (∃ c. t = Fun c [])"
using transaction_decl_fresh_renaming_substs_wt[OF step.hyps(3-5)]
transaction_decl_fresh_renaming_substs_range_wf_trms[OF step.hyps(3-5)]
transaction_decl_fresh_renaming_substs_range'[1][OF step.hyps(3-5)]
unfolding ϑ_def by (metis,metis,fastforce)
hence "?R (dual lsst (dual lsst (transaction_strand T) · ϑ) · ϑ) (transaction_strand T)"
using dual lsst_self_inverse[of "transaction_strand T"]
by (auto simp add: Q_def subst_apply_labeled_stateful_strand_def)
hence "?R (dual lsst (transaction_strand T · ϑ)) (transaction_strand T)"
by (metis dual lsst lsst_def)
hence "?R (dual lsst (dual lsst (transaction_strand T · ϑ)) · ϑ) Ts" using step.hyps(2) unfolding Ts_def dual lsst_def by fastforce
thus ?case using step.1H unfolding Q_def ϑ_def by auto
qed simp

lemma reachable_constraints_par_comp lsst:
fixes P
defines "f ≡ λM. {t · δ | t δ. t ∈ M ∧ wt subst δ ∧ wf trms (subst_range δ) ∧ fv (t · δ) = {}}"
and "Ts ≡ concat (map transaction_strand P)"
assumes P_pc: "comp_par comp lsst public arity Ana Γ Pair Ts M S"
and A: "A ∈ reachable_constraints P"
shows "par_comp_sst A ((f S) - {m. intruder_synh \{ m \}})"
using par_comp_sst_if_comp_par_compsst[OF P Pc, of "dualsst A", THEN par_comp_sst_dualsst]
reachable_constraints_par_comp_sst_aux[OF A, unfolded Ts_def[symmetric]]
unfolding f_def dualsst_self_inverse by fast
end

lemma reachable_constraints_par_comp_constr:
fixes P i S
defines "f ≡ λ M. \{ f · t · t' · \tau M. \Delta ∈ M \land wftrms \Delta (subj_range \delta) \land fv (t · \delta) = \{ \} \}"
and "Ts ≡ \bigcup i∈ S. trms_transaction i (unlabel (transaction_strand i))"
and "Sec ≡ f S - \{ m. intruder_synh \{ m \} \}"
and "\M ≡ \bigcup T ∈ set P. trms_transaction T \cup pair' Pair \ setops_transaction T"
assumes M:
"has_all_wt_instances_of \Gamma M N"
"finite N"
"trfset N"
"wftrms N"
and P:
"\forall T ∈ set P. wellformed_transaction T"
"\forall T ∈ set P. uftrms' arity (trms_transaction T)"
"\forall T ∈ set P. list_all trfsetp (unlabel (transaction_strand T))"
"comp_par_compsst public arity Ana \Gamma Pair Ts M fun S"
and A: "A ∈ reachable_constraints P"
and I: "constraint_model I A"
s shows "∃ I. welltyped_constraint_model I, A ∧
(∃ n. welltyped_constraint_model I, (proj n A)) ∨
(∃ A' t. prefix A' A ∧ suffix [(1, receive(t))] A' ∧ strand_leakssst A' Sec I,))"
proof -
have I: "constraint_sem_stateful I (unlabel A)" "interpretation_subst I"
using I unfolding constraint_model_def by blast

show ?thesis
using reachable_constraints_par_compsst[OF P(4)[unfolded Ts_def] A]
reachable_constraints_typing_cond_sst[OF M_def M P(1-3) A]
par_comp_constr_stateful[OF _ I', of Sec]
unfolding I_def Sec_def welltyped_constraint_model_def constraint_model_def by blast

qed

lemma reachable_constraints_component_leaks_if_composed_leaks:
fixes Sec Q
defines 'leaks' ≡ λ A. ∃ I. A'.
Q. I. ∧ prefix A' A ∧ (∀ t. suffix [(1, receive(t))] A') ∧ strand_leakssst A' Sec I,"
assumes "∀ s ∈ Sec. \{ s \} ⊢ s" "ground Sec"
and composed_leaks: "∃ A ∈ reachable_constraints Ps. leaks A"
s shows "∃ I. A ∈ reachable_constraints (map (transaction_proj 1) Ps). leaks A"
proof -
from composed_leaks obtain A I, A' s n where
A: "A ∈ reachable_constraints Ps" and
A': "prefix A' A" "constraint_sem_stateful I, (proj_unl n A'#[send([s])])" and
I: "Q I, ∧ s ∈ Sec - declassifiedsst A' I,"
unfolding leaks_def strand_leakssst_def by fast
have "\{ s \} ⊢ s" "s · I, = s" using s Sec by auto
then obtain B k' u where
"constraint_sem_stateful I, (proj_unl n B#[send([s])])"
"prefix (proj n B) (proj n A)" "suffix [(k', receive(u))] (proj n B)"
"s ∈ Sec - declassifiedsst (proj n B) I,"
using constr_sem_stateful_proj_priv_term_prefix_obtain[OF A' s]
unfolding welltyped_constraint_model_def constraint_model_def by metis
thus ?thesis
using I, reachable_constraints_transaction_proj[OF A, of n] proj_idem[of n B]
unfolding leaks_def strand_leakssst_def
3.3 Stateful Protocol Model

by metis

lemma reachable_constraints_preserves_labels:
assumes A: "A ∈ reachable_constraints P"
shows "∀ a ∈ set A. ∃ T ∈ set P. ∃ b ∈ set (transaction_strand T). fst b = fst a"
(is "∀ a ∈ set A. ∃ T ∈ set P. ?P T a")
using A
proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α)
  have "∀ a ∈ set (dual lsst (transaction_strand T · lsst ξ ◦ s σ ◦ s α)). ?P T a"
    unfolding dual_lsst_def by auto
  then obtain c where c: "c ∈ set (transaction_strand T)" "b = c · lsstp ξ ◦ s σ ◦ s α"
    unfolding subst_apply_labeled_stateful_strand_def by auto
  have "?P T c" using c(1) by blast
  hence "?P T b" using c(2) by (simp add: subst_lsstp_fst_eq)
  thus "?P T a" using b(2) dual_lsstp_fst_eq[of b] by presburger
qed simp

lemma reachable_constraints_preserves_labels':
assumes P: "∀ T ∈ set P. ∀ a ∈ set (transaction_strand T). has_LabelN l a ∨ has_LabelS a" and A: "A ∈ reachable_constraints P" shows "∀ a ∈ set A. has_LabelN l a ∨ has_LabelS a"
using reachable_constraints_preserves_labels[OF A] P by fastforce

lemma reachable_constraints_transaction_proj_proj_eq:
assumes A: "A ∈ reachable_constraints (map (transaction_proj l) P)"
and k_neq_l: "k ≠ l"
shows "proj k A ∈ reachable_constraints (map transaction_star_proj P)"
using A proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α)
  let ?T = "dual lsst (transaction_strand T · lsst ξ ◦ s σ ◦ s α)"
  note A = reachable_constraints.step[OF step.hyps]
  have P: "∀ T ∈ set P. ∀ a ∈ set (transaction_strand T). has_LabelN l a ∨ has_LabelS a"
    unfolding list_all_iff by auto
  note * = reachable_constraints_preserves_labels'[OF P A]
  have **: "∀ a ∈ set A. has_LabelN l a ∨ has_LabelS a" when "∀ a ∈ set B. has_LabelN l a ∨ has_LabelS a" "prefix A' B" for B using that assms unfolding prefix_def by auto
  note *** = proj_ident[unfolded list_all_iff]
  { case 1 thus ?case using *[THEN ***] by blast }
  { case 2 thus ?case using *[THEN **, THEN ***] by blast }
qed (simp_all add: reachable_constraints.init)
case (step A T ξ σ α)
have "map (transaction_proj k) (map (transaction_proj 1) P) = map transaction_star_proj P"
  using transaction_star_proj_negates_transaction_proj(2)[OF k_neq_l]
  by fastforce
thus ?case
  using reachable_constraints_transaction_proj[OF reachable_constraints.step[OF step.hyps], of k]
  by argo
qed (simp add: reachable_constraints.init)

lemma reachable_constraints_aligned_prefix_ex:
  fixes P
defines "f ≡ λT. list_all is_Receive (unlabel (transaction_receive T)) ∧ list_all is_Check_or_Assignment (unlabel (transaction_checks T)) ∧ list_all is_Update (unlabel (transaction_updates T)) ∧ list_all is_Send (unlabel (transaction_send T))"
assumes P: "list_all f P" "list_all ((list_all (Not ◦ has_LabelS)) ◦ tl ◦ transaction_send) P" and s: "¬{} ⊢ c s" "fv s = {}" and A: "A ∈ reachable_constraints P" "prefix B A" and B: "∃ l ts. suffix [(l, receive ⟨ts⟩)] B" "constr_sem_stateful I (unlabel B@[send ⟨[s]⟩])"
shows "∃ C ∈ reachable_constraints P. prefix C A ∧ (∃ l ts. suffix [(l, receive ⟨ts⟩)] C) ∧ declassified lsst B I = declassified lsst C I ∧ constr_sem_stateful I (unlabel C@[send ⟨[s]⟩])"

proof (induction A rule: reachable_constraints.induct)
case (step A T ξ σ α)
define ϑ where "ϑ ≡ ξ ◦ s ◦ σ ◦ s ◦ α"
let ?T = "dual lsst (transaction_strand T · lsst ϑ)"

note AT_reach = reachable_constraints.step[OF step.hyps]

obtain lb tsb B'
  where B': "B = B'@[lb, receive ⟨tsb⟩]" using B(1) unfolding suffix_def by blast

define decl_ik where "decl_ik ≡ λS::('fun,'atom,'sets,'lbl) prot_strand. ⋃{set ts | ts. ⟨⋆, receive ⟨ts⟩⟩ ∈ set S} · set I"

have decl_ik_append: "decl_ik (M@N) = decl_ik M ∪ decl_ik N" for M N unfolding decl_ik_def by fastforce

have decl_ik_star: "decl_ik M = decl_ik (M@N)" when "⋆ /∈ fst ` set N" for M N using that unfolding decl_ik_def

have decl_decl_ik: "declassified lsst S I = {t. decl_ik S ⊢ t}" for S unfolding declassified_list_all_alt_def decl_ik_def by blast

have "f T" using P(1) step.hyps(2) by (simp add: list_all_iff)

have "list_all is_Send (unlabel (dual lsst (transaction_receive T · ϑ)))"
  "list_all is_Check_or_Assignment (unlabel (dual lsst (transaction_checks T · ϑ)))"
  "list_all is_Update (unlabel (dual lsst (transaction_updates T · ϑ)))"
  "list_all is.Receive (unlabel (dual lsst (transaction_send T · ϑ))))"
  using subst_sst_list_all(2)[of "unlabel (transaction_receive T)" ϑ]
  subst_sst_list_all(11)[of "unlabel (transaction_checks T)" ϑ]
  subst_sst_list_all(10)[of "unlabel (transaction_updates T)" ϑ]
  subst_sst_list_all(1)[of "unlabel (transaction_send T)" ϑ]
  dual lsst_list_all(1)[of "transaction_receive T · ϑ"]
  dual lsst_list_all(11)[of "transaction_checks T · ϑ"]
  dual lsst_list_all(10)[of "transaction_updates T · ϑ"]
  dual lsst_list_all(2)[of "transaction_send T · ϑ"]
  unfolding t_def by (metis unlabel_subst[of _ ϑ])

128
3.3 Stateful Protocol Model

hence "$ \neg \text{list} \text{ex is} \text{Receive} (\text{unlabel (dual}_{\text{lsst}} \text{(transaction} \text{receive} T \cdot \text{lsst} \text{)}))$" "$ \neg \text{list} \text{ex is} \text{Receive} (\text{unlabel (dual}_{\text{lsst}} \text{(transaction} \text{checks} T \cdot \text{lsst} \text{)}))$" "$ \neg \text{list} \text{ex is} \text{Receive} (\text{unlabel (dual}_{\text{lsst}} \text{(transaction} \text{updates} T \cdot \text{lsst} \text{)}))$" "$ \text{list} \text{all is} \text{Receive} (\text{unlabel (dual}_{\text{lsst}} \text{(transaction} \text{send} T \cdot \text{lsst} \text{)}))$"

unfolding list_ex_iff list_all_iff by blast*

then obtain TA TB where T:
"?T = TA@TB" "$ \neg \text{list} \text{ex is} \text{Receive} (\text{unlabel} TA)$" "$ \text{list} \text{all is} \text{Receive} (\text{unlabel} TB)$"

using transaction_dual_subst_unfold[of T ϑ] unfolding ϑ_def by fastforce

have 0: "$ \neg \text{prefix} A (A@TA@TB)$" using step.prems B' T by argo

have 1: "$ \neg \text{prefix} A A$" when "$ \neg \text{prefix} A (A@TA)$"

using that T(2) B' prefix_prefix_inv unfolding list_ex_iff unlabel_def by fastforce

have 2: "$ \neg \text{prefix} B A$" when "$ \neg \text{prefix} B (A@TA)$"

using that subst_lsst_map_fst_eq[of "tl (transaction_send T)" ϑ]

unfolding T(4) dual_lsst_tl subst_lsst_tl by simp

moreover have "set TB2 ⊆ set (tl TB)"

ultimately show ?thesis by auto

qed

have 3: "$ \text{declassified}_{\text{lsst}} \text{TB} I = \text{declassified}_{\text{lsst}} (TB1@[l,x]) I" when "$ \text{TB} = TB1@[l,x]#TB2$" for TB1 1 x TB2

using decl_ik_star[OF 2[OF that], of "TB1@[l,x]"

unfolding decl_decl_ik by simp

show ?case

proof (cases "prefix B A")

case False

have 5: "$ \exists l ts. \text{suffix} [(l, receive\langle ts \rangle)] (A@TA)$"

proof -

have "[(l, receive\langle ts \rangle)] \in set TB"

using 0 4 prefix_prefix_inv[of _ suffixI[of B'], of "A@TA" TB] by (metis append_assoc)

hence "receive\langle ts \rangle \in set (unlabel TB)"

unfolding unlabel_def by force

hence "$ \exists ts. \text{suffix} [receive\langle ts \rangle] (unlabel TB)$"

using T(3) unfolding list_all_iff is_Receive_def suffix_def

by (metis in_set_conv_decomp list.distinct(1) list.set_cases rev_exhaust)

then obtain TB' ts where "unlabel TB = TB'@[receive\langle ts \rangle]" unfolding suffix_def by blast

then obtain TB' x where "$ \text{TB} = TB'@[x]" "snd x = receive\langle ts \rangle"

by (smt unlabel_def Nil_is_map_conv map_eq_consD map_eq_append_conv)

then obtain l where "$ \text{suffix} [(l, receive\langle ts \rangle)] TB$" by (metis surj_pair prod.sel(2) suffix_def)

thus ?thesis

using T(4) transaction_dual_subst_unfold[of T ϑ]

suffix_append[of "[(l, receive\langle ts \rangle)]"]

unfolding ϑ_def by metis

qed

obtain TB1 where TB:
"B = A@TA@TB1@[l, receive\langle ts \rangle]" "prefix (TB1@[l, receive\langle ts \rangle]) TB"

using 0 4 B' prefix_snoc_obtain[of B' "(l, receive\langle ts \rangle)" "A@TA" TB thesis]"
by (metis append_assoc)
then obtain TB2 where TB2: "TB = TB1@([lb, receive⟨tsb⟩])#TB2"
unfolding prefix_def by fastforce
hence TB2': "list_all is_Receive (unlabel TB2)"
using T(3) unfolding list_all_iff is_Receive_def proj_def unlabel_def by auto
have 6: "constr_sem_stateful I (unlabel B)" "ik lsst B · set I ⊢ s · I"
using B(2) strand_sem_append_stateful[of "{}" "{}" "unlabel B" "[send⟨[s]⟩]" I]
by auto
have "constr_sem_stateful I (unlabel (A@TA@TB1@[lb, receive⟨tsb⟩]))"
using 6(1) TB(1) by blast
hence "constr_sem_stateful I (unlabel (A@TA))"
using T(1) TB2 strand_sem_receive_prepend_stateful[of "{}" "{}" "unlabel (A@TA@TB1@[lb, receive⟨tsb⟩])" I, OF _ TB2']
by auto
moreover have "set (unlabel B) ⊆ set (unlabel (A@TA))"
using step.prems(1) unfolding prefix_def by force
hence "ik lsst (A@TA) · set I ⊢ s · I"
using ideduct_mono[OF 6(2)] subst_all_mono[of _ _ I] ik lsst_set_subset[of "unlabel B" "unlabel (A@TA)"]
by meson
ultimately have 7: "constr_sem_stateful I (unlabel (A@TA@TB1@[lb, receive⟨tsb⟩]))"
using strand_sem_append_stateful[of "{}" "{}" "unlabel (A@TA@TB1@[lb, receive⟨tsb⟩])" I]
by auto
have "declassified lsst B I = declassified lsst (A@TA) I"
proof -
have 3[of _ lb "receive⟨tsb⟩"] TB(2) unfolding prefix_def by auto
hence "(decl_ik TB ⊢ t) ←→ decl_ik (TB1@[lb, receive⟨tsb⟩]) ⊢ t" for t
unfolding TB(1) T(1) decl_decl_ik by blast
hence "(decl_ik (A@TA@TB) ⊢ t) ←→ decl_ik (A@TA@TB1@[lb, receive⟨tsb⟩]) ⊢ t" for t
using ideduct_mono_eq[of "decl_ik TB" "decl_ik (TB1@[lb, receive⟨tsb⟩])" "decl_ik (A@TA)"]
by (metis decl_ik_append[of "A@TA"] Un_commute[of _ "decl_ik (A@TA)" append_assoc])
thus ?thesis unfolding TB(1) T(1) decl_decl_ik by blast
qed
thus ?thesis using step.prems AT_reach B(1) 5 7 by force
qed (use step.IH prefix_append in blast)
qed (use B(1) suffix_def in simp)
end

3.4 Term Variants

theory Term_Variants
imports Stateful_Protocol_Composition_and_Typeing.Intruder_Deduction
begin

fun term_variants where
| "term_variants P (Var x) = [Var x]"
| "term_variants P (Fun f T) = (let S = product_lists (map (term_variants P) T) in map (Fun f) S@concat (map (λg. map (Fun g) S) (P f)))"

inductive term_variants_pred for P where
| "term_variants_pred P (Var x) (Var x)"

end
3.4 Term Variants

"[length T = length S; \( i. \ i < length T \Rightarrow \text{term_variants_pred} P (T ! i) (S ! i); \ g \in \text{set} (P f)]\)

\[ \Rightarrow \text{term_variants_pred} P (\text{Fun f T}) (\text{Fun g S}) \]"

/ term_variants_Fun:

"[length T = length S; \( i. \ i < length T \Rightarrow \text{term_variants_pred} P (T ! i) (S ! i)]\)

\[ \Rightarrow \text{term_variants_pred} P (\text{Fun f T}) (\text{Fun f S}) \]"

lemma term_variants_pred_inv:

assumes "term_variants_pred P (Fun f T) (Fun h S)"

shows "length T = length S"

and "\( i. \ i < length T \Rightarrow \text{term_variants_pred} P (T ! i) (S ! i)\)"

and "f \neq h \Rightarrow h \in \text{set} (P f)"

using assms by (auto elim: term_variants_pred.cases)

lemma term_variants_pred_inv':

assumes "term_variants_pred P (Fun f T) t"

shows "is_Fun t"

and "length T = length (args t)"

and "\( i. \ i < length T \Rightarrow \text{term_variants_pred} P (\text{args t }+ i) (\text{args t }+ i)\)"

and "f \neq \text{the_Fun t} \Rightarrow \text{the_Fun t} \in \text{set} (P (\text{the_Fun t}))"

and "P \equiv (\lambda _\cdot [\cdot])(g := [h]) \Rightarrow f \neq \text{the_Fun t} \Rightarrow f = g \land \text{the_Fun t} = h"

using assms by (auto elim: term_variants_pred.cases)

lemma term_variants_pred_inv'':

assumes "term_variants_pred P t (Fun f T)"

shows "is_Fun t"

and "length T = length (args t)"

and "\( i. \ i < length T \Rightarrow \text{term_variants_pred} P (\text{args t }+ i) (\text{args t }+ i)\)"

and "f \neq \text{the_Fun t} \Rightarrow \text{the_Fun t} \in \text{set} (P (\text{the_Fun t}))"

and "P \equiv (\lambda _\cdot [\cdot])(g := [h]) \Rightarrow f \neq \text{the_Fun t} \Rightarrow f = g \land \text{the_Fun t} = h"

using assms by (auto elim: term_variants_pred.cases)

lemma term_variants_pred_refl: "term_variants_pred P t t"

by (induct t) (auto intro: term_variants_pred.intros)

lemma term_variants_pred_refl_inv:

assumes st: "term_variants_pred P s t"

and P: "\( \forall f. \ \forall g \in \text{set} (P f). \ f = g \)"

shows "s = t"

using st P

proof (induction s t rule: term_variants_pred.induct)

case (term_variants_Var x) thus ?case by blast

next
case (term_variants_P T S g f)

hence "T ! i = S ! i" when i: "\( i < length T \)" for i using i by blast

hence "T = S" using term_variants_P.hyps(1) by (simp add: nth_equalityI)

thus ?case using term_variants_P.prems term_variants_P.hyps(3) by fast

next
case (term_variants_Fun T S f)

hence "T ! i = S ! i" when i: "\( i < length T \)" for i using i by blast

hence "T = S" using term_variants_Fun.hyps(1) by (simp add: nth_equalityI)

thus ?case by fast

qed

lemma term_variants_pred_const:
assumes "b ∈ set (P a)"
shows "term_variants_pred P (Fun a []) (Fun b [])"

using term_variants_P[of "[]" "[]"] assms by simp

lemma term_variants_pred_const_cases:
  assumes "P a ≠ []" "P a ∈ set (P a)"
  shows "term_variants_pred P (Fun a []) t ≡ t = Fun a []"
  using term_variants_P[of "[]" "[]"] assms by simp

lemma term_variants_pred_param:
  assumes "P a = []" "P a ∈ set (P a)"
  shows "term_variants_pred P (Fun a []) t ≡ t = Fun a []"
3.4 Term Variants

using IH[OF i hyps(4,5,6)] unfolding F_def by presburger
then obtain U where U:
"length T = length U" "\i. i < length T \implies term_variants_pred (F P gs) (T ! i) (U ! i)"
"length U = length S" "\i. i < length U \implies term_variants_pred (F Q fs) (U ! i) (S ! i)"
by moura

show ?case
  using term_variants_pred.term_variants_P[OF U(1,2), of g h]
  term_variants_pred.term_variants_P[OF U(3,4), of h' g]
hyps(3)[unfolded hyps(6)] hyps(4,5)
unfolding F_def by force
next

  case (term_variants_Fun T S h' g gs)
  note hyps = term_variants_Fun.hyps(1,2,4,5,6)

  note IH = term_variants_Fun.hyps(3)

  have "\s. term_variants_pred (F P gs) (T ! i) s \land term_variants_pred (F Q fs) s (S ! i)"
  when i: "i < length T"
  using IH[OF i hyps(3,4,5)] unfolding F_def by presburger

thus ?case
  using term_variants_pred.term_variants_Fun[OF U(1,2)]
  term_variants_pred.term_variants_Fun[OF U(3,4)]
unfolding F_def by meson

qed

lemma term_variants_pred_dense':
  assumes ut: "term_variants_pred ((λ_. [])(a := [b])) u t"
  shows "\s. term_variants_pred ((λ_. [])(a := [c])) u s \land term_variants_pred ((λ_. [])(c := [b])) s t"
  using ut term_variants_pred_dense[of "{a}" "[b]" u t "{c}" "[c]"
unfolding fun_upd_def by simp

lemma term_variants_pred_eq_case:
  fixes t s::"('a,'b) term"
  assumes "term_variants_pred P t s" "∀ f ∈ funs_term t. P f = []"
  shows "t = s"
  using assms
  proof (induction t s rule: term_variants_pred.induct)
    case (term_variants_Fun T S f) thus ?case
    using subtermeq_imp_funs_term_subset[OF Fun_param_in_subterms[OF nth_mem], of _ T f]
    nth_equalityI[of T S]
    by blast
  qed (simp_all add: term_variants_pred_refl)

lemma term_variants_pred_subst:
  assumes "term_variants_pred P t s" "f ∈ funs_term t. P f = []"
  shows "term_variants_pred P (t · δ) (s · δ)"
  using assms
  proof (induction t s rule: term_variants_pred.induct)
    case (term_variants_P T S f g)
    have l: "length (map (λt. t · δ)) T) = length (map (λt. t · δ) S)"
      using term_variants_P.hyps
    by simp
have 2: "\text{term\_variants\_pred } P ((\text{map } (\lambda t \cdot \delta) T) ! i) ((\text{map } (\lambda t \cdot \delta) S) ! i)"
when "i < length (\text{map } (\lambda t \cdot \delta) T)" for i
using term\_variants\_P that
by fastforce

show ?case
using term\_variants\_pred.term\_variants\_P[OF 1 2 term\_variants\_P.hyps(3)]
by fastforce

next

  case (term\_variants\_Fun T S f)
  have 1: "length (\text{map } (\lambda t \cdot \delta) T) = length (\text{map } (\lambda t \cdot \delta) S)"
using term\_variants\_Fun.hyps
by simp

  have 2: "\text{term\_variants\_pred } P ((\text{map } (\lambda t \cdot \delta) T) ! i) ((\text{map } (\lambda t \cdot \delta) S) ! i)"
when "i < length (\text{map } (\lambda t \cdot \delta) T)" for i
using term\_variants\_Fun that
by fastforce

show ?case
using term\_variants\_pred.term\_variants\_Fun[OF 1 2]
by fastforce

qed (simp add: term\_variants\_pred\_refl)

lemma term\_variants\_pred\_subst':
fixes t::"('a,'b) term" and \delta::"('a,'b) subst"
assumes "\text{term\_variants\_pred } P (t \cdot \delta) s"
and ",\forall x \in \text{fv } t \cup \text{fv } s. (\exists y. \delta x = \text{Var } y) \lor (\exists f. \delta x = \text{Fun } f []) \land P f = []"
shows "\exists \nu. \text{term\_variants\_pred } P t \nu \land s = \nu \cdot \delta"
using assms
proof (induction "t \cdot \delta" s arbitrary: t rule: term\_variants\_pred.induct)
  case (term\_variants\_Var x g)
  thus ?thesis
using term\_variants\_P.hyps(4,5) term\_variants\_P.prems
by fastforce

  next
  case (term\_variants\_P T S g f)
  show ?case
  proof (cases t)
    case (Var x)
    thus ?thesis
using term\_variants\_P.hyps(4) term\_variants\_P.prems
by fastforce

    next
  case (Fun h U)
  hence 1: "h = f" "T = map (\lambda s \cdot \delta) U" "length U = length T"
using term\_variants\_P.hyps(5) by simp_all
  hence 2: "T ! i = U ! i \cdot \delta" when "i < length U" for i
using that by simp

  have "\forall x \in \text{fv } (U ! i) \cup \text{fv } (S ! i). (\exists y. \delta x = \text{Var } y) \lor (\exists f. \delta x = \text{Fun } f []) \land P f = []"
when "i < length U" for i
using that Fun term\_variants\_P.prems term\_variants\_P.hyps(1) 1(3)
by force
  hence IH: "\forall i < length U. \exists \nu. \text{term\_variants\_pred } P (U ! i) \nu \land S ! i = \nu \cdot \delta"
by (metis 1(3) term\_variants\_P.hyps(3)[OF _ 2])

  have "\exists \nu. \text{length } U = \text{length } V \land \text{length } S = \text{map } (\lambda v \cdot \delta) V \land
  (\forall i < length U. \text{term\_variants\_pred } P (U ! i) (V ! i))"
using term\_variants\_P.hyps(1) 1(3) subst\_term\_list\_obtain[OF IH] by metis
  then obtain \nu where \nu: "\text{length } U = \text{length } V \land \text{length } S = \text{map } (\lambda v \cdot \delta) V"
  by (metis \nu(1) term\_variants\_P.hyps(4))

  have "term\_variants\_pred P (Fun f U) (Fun g V)"
  by (metis term\_variants\_pred.term\_variants\_P[OF V(1,3) term\_variants\_P.hyps(4)])
moreover have "Fun g S = Fun g V \cdot \delta" using V(2) by simp

have "term\_variants\_pred P (Fun f U) (Fun g V)"
by (metis term\_variants\_pred.term\_variants\_P[OF V(1,3) term\_variants\_P.hyps(4)])
ultimately show ?thesis using term_variants_P.hyps(1,4) Fun 1 by blast
next
case (term_variants_Fun T S f t) show ?case
  proof (cases t)
    case (Var x)
    hence "T = []" "P f = []" using term_variants_Fun.hyps(4) term_variants_Fun.prems by fastforce+
    thus ?thesis using term_variants_pred_refl Var term_variants_Fun.hyps(1,4) by fastforce
  next
    case (Fun h U)
    hence 1: "h = f" "T = map (λs. s · δ) U" "length U = length T"
      using term_variants_Fun.hyps(4)
    hence 2: "T ! i = U ! i · δ" when "i < length T" for i
      using that by simp
    have "∀x ∈ fv (U ! i) ∪ fv (S ! i). (∃y. δ x = Var y) ∨ (∃f. δ x = Fun f [] ∧ P f = []))"
      when "i < length U" for i
      using Fun term_variants_Fun.prems term_variants_Fun.hyps(1,2)
      by force
    hence IH: "∀i < length U. ∃u. term_variants_pred P (U ! i) u ∧ S ! i = u · δ"
      by (metis 1(3) term_variants_Fun.hyps(3)[OF _ 2])
    have "∃V. length U = length V ∧ S = map (λv. v · δ) V ∧
      (∀i < length U. term_variants_pred P (U ! i) (V ! i))" by moura
    moreover have "Fun f S = Fun f (λv. v · δ)" using V(2)
    ultimately show ?thesis using term_variants_Fun.hyps(1) Fun 1 by blast
  qed

lemma term_variants_pred_subst'':
  assumes "∀x ∈ fv t. term_variants_pred P (δ x) (ϑ x)"
  shows "term_variants_pred P (t · δ) (t · ϑ)"
  using assms
  proof (induction t)
    case (Fun f ts)
    thus ?case
      using term_variants_Fun[of "map (λt. t · δ) ts" "map (λt. t · ϑ) ts" P f] by force
  qed simp

lemma term_variants_pred_iff_in_term_variants:
  fixes t::"('a,'b) term"
  shows "term_variants_pred P t s ←→ s ∈ set (term_variants P t)"
  (is "?A t s ←→ ?B t s")
  proof
define U where "U ≡ λP (T::('a,'b) term list). product_lists (map (term_variants P) T)"
  have a:
    "g ∈ set (P f) ⇒ set (map (Fun g) (U P T)) ⊆ set (term_variants P (Fun f T))"
    "set (map (Fun f) (U P T)) ⊆ set (term_variants P (Fun f T))"
  for f P g and T::"('a,'b) term list"
  using term_variants.simps(2)[of P f T]
  unfolding U_def Let_def by auto
  have b: "∃S ∈ set (U P T). s = Fun f S ∨ (∃g ∈ set (P f). s = Fun g S)"
    when "s ∈ set (term_variants P (Fun f T))" for P T f s
    using that by (cases "P f") (auto simp add: U_def Let_def)
have \( c: \text{"length }T = \text{length }S\) when \( S \in \text{set (}U \text{ } P \text{ } T)\) for \( S \text{ } P \text{ } T\)
using that unfolding \( U\text{\_def} \)
by (simp add: in_set_product_lists_length)

show \( ?A \text{ } t \text{ } s \implies ?B \text{ } t \text{ } s\)
proof (induction \( t \text{ } s \) rule: term_variants_pred.induct)
  case (term_variants_P \( T \text{ } S \text{ } g \text{ } f)\)
  note hyps = term_variants_P.hyps
  note IH = term_variants_P.IH

  have \( \text{"S } \in \text{set (}U \text{ } P \text{ } T)\)"
  using IH hyps(1) product_lists_in_set_nth[of _ S]
  unfolding U_def by simp
  thus ?case using a(1)[of _ P, OF hyps(3)] by auto
next
  case (term_variants_Fun \( T \text{ } S \text{ } f)\)
  note hyps = term_variants_Fun.hyps
  note IH = term_variants_Fun.IH

  have \( \text{"S } \in \text{set (}U \text{ } P \text{ } T)\)"
  using IH hyps(1) product_lists_in_set_nth[of _ S]
  unfolding U_def by simp
  thus ?case using a(2)[of f P T] by (cases "P \text{ } f") auto
qed (simp add: term_variants_Var)

show \( ?\neg \text{ } t \text{ } s \implies ?\neg \text{ } t \text{ } s\)
proof (induction \( P \text{ } t \) arbitrary: \( s \) rule: term_variants.induct)
  case (2 \( P \text{ } f \text{ } T)\)
  obtain \( S \) where
  \( \text{\"s = Fun } f \text{ } S \lor (\exists g \text{ } \in \text{set (}P \text{ } f). \text{ } s = Fun } g \text{ } S\)"
  \( \text{\"S } \in \text{set (}U \text{ } P \text{ } T)\)" \( \text{\"length }T = \text{length }S\)"
  using c b[OF "2.prems"] by moura

  have \("\forall i < \text{length }T. \text{term_variants_pred } P \text{ } (T ! i) \text{ } (S ! i)\)"
  using "2.IH" S product_lists_in_set_nth by (fastforce simp add: U_def)
  thus ?case using S by (auto intro: term_variants_pred.intros)
qed (simp add: term_variants_Var)

lemma term_variants_pred_finite:
  "finite \{s. \text{term_variants_pred } P \text{ } t \text{ } s\}"
using term_variants_pred_iff_in_term_variants[of \( P \text{ } t \)]
by simp

lemma term_variants_pred_fv_eq:
  assumes \"\text{term_variants_pred } P \text{ } s \text{ } t\"
  shows \"fv } s = \text{fv } t\"
using assms
by (induct rule: term_variants_pred.induct)
  (metis, metis fv_eq_FunI, metis fv_eq_FunI)

lemma (in intruder_model) term_variants_pred_wf_trms:
  assumes \"\text{term_variants_pred } P \text{ } s \text{ } t\"
and \"\forall g. \ g \in \text{set (}P \text{ } f) \implies \text{arity } f = \text{arity } g\"
and \"\text{wf_trm } s\"
  shows \"\text{wf_trm } t\"
using assms
apply (induction rule: term_variants_pred.induct, simp)
by (metis (no_types) wf_trmI uf_trm_arity in_set_conv_nth wf_trm_param_idx)+

lemma term_variants_pred_funs_term:
  assumes \"\text{term_variants_pred } P \text{ } s \text{ } t\"
and \"f \in \text{funs_term } t\"
3.5 Term Implication

shows "f ∈ funs_term s ∨ (∃ g ∈ funs_term s. f ∈ set (P g))"
using assms

proof (induction rule: term_variants_pred.induct)
case (term_variants_P T S g h) thus ?case
proof (cases "f = g")
  case False
  then obtain s where "s ∈ set S" "f ∈ funs_term s"
  using funs_term_subterms_eq(1)[of "Fun g S"] term_variants_P.prems by auto
  thus ?thesis
  using term_variants_P.IH term_variants_P.hyps(1) in_set_conv_nth[of s S] by force
qed simp

next
  case (term_variants_Fun T S h)
  thus ?case
  proof (cases "f = h")
    case False
    then obtain s where "s ∈ set S" "f ∈ funs_term s"
    using funs_term_subterms_eq(1)[of "Fun h S"] term_variants_Fun.prems
    by auto
    thus ?thesis
    using term_variants_Fun.IH term_variants_Fun.hyps(1) in_set_conv_nth[of s S] by force
  qed simp
qed fast

end

3.5 Term Implication

theory Term_Implication
imports Stateful_Protocol_Model Term_Variants
begin

3.5.1 Single Term Implications

definition timpl_apply_term ("⟨ a --» b ⟩ ⟨ t ⟩") where
"⟨ a --» b ⟩ ⟨ t ⟩ ≡ term_variants ((λ_. t) (Abs a := Abs b)) t"

definition timpl_apply_terms ("⟨ a --» b ⟩ ⟨ M ⟩") where
"⟨ a --» b ⟩ ⟨ M ⟩ ≡ ⋃ ((set o timpl_apply_term a b) ` M)"

lemma timpl_apply_Fun:
assumes "(∀ i. i < length T =⇒ S ! i ∈ set (a --» b)(T ! i))"
and "length T = length S"
shows "Fun f S ∈ set (a --» b)(Fun (Abs a) T)"
using assms(1) term_variants_Fun term_variants_pred_iff_in_term_variants
by (metis timpl_apply_term_def)

lemma timpl_apply_Abs:
assumes "(∀ i. i < length T =⇒ S ! i ∈ set (a --» b)(T ! i))"
and "length T = length S"
shows "Fun (Abs b) S ∈ set (a --» b)(Fun (Abs a) T)"
using assms(1) term_variants_Abs funs_term_pred_iff_in_term_variants
by (metis timpl_apply_term_def)

lemma timpl_apply_refl: "t ∈ set (a --» b)(t)"
unfolding timpl_apply_term_def
by (metis terms_term_variants_pred_refl term_variants_pred iff in term variants)

lemma timpl_apply_const: "Fun (Abs b) [] ∈ set (a --» b)(Fun (Abs a) [])"
using timpl_apply_term_def by auto

lemma timpl_apply_const':

"c = a \Rightarrow \text{set} \ (a \to b)(\text{Fun} \ (\text{Abs} \ c) \ []) = \{\text{Fun} \ (\text{Abs} \ b) \ [], \ \text{Fun} \ (\text{Abs} \ c) \ []\}"
"c \neq a \Rightarrow \text{set} \ (a \to b)(\text{Fun} \ (\text{Abs} \ c) \ []) = \{\text{Fun} \ (\text{Abs} \ c) \ []\}"

using \text{term}_\text{variants}_\text{pred}_\text{const}_\text{cases}[\text{of} \ "(\lambda_\_ \ [])(\text{Abs} \ a := [\text{Abs} \ b])" \ "\text{Abs} \ c"]

\text{term}_\text{variants}_\text{pred}_\text{iff}_\text{in}_\text{term}_\text{variants}[\text{of} \ "(\lambda_\_ \ [])(\text{Abs} \ a := [\text{Abs} \ b])"]

unfolding \text{timpl}_\text{apply}_\text{term}_\text{def} \ by \ auto

lemma \text{timpl}_\text{apply}_\text{term}_\text{subst}:
\ "s \in \text{set} \ (a \to b)(\text{Fun} \ f \ T) = \Rightarrow s \cdot \delta \in \text{set} \ (a \to b)(T \cdot \delta)"

by (\text{metis} \ \text{term}_\text{variants}_\text{pred}_\text{iff}_\text{in}_\text{term}_\text{variants} \ \text{term}_\text{variants}_\text{pred}_\text{subst} \ \text{timpl}_\text{apply}_\text{term}_\text{def})

lemma \text{timpl}_\text{apply}_\text{inv}:
\ assumes "\text{Fun} \ h \ S \in \text{set} \ (a \to b)(\text{Fun} \ f \ T)"
\ shows "\text{length} T = \text{length} S"
\ and "\forall i. \ i < \text{length} T = \Rightarrow S ! i \in \text{set} \ (a \to b)(T ! i)"
\ and "f \neq h = \Rightarrow f = \text{Abs} \ a \land h = \text{Abs} \ b"

using \text{assms} \ \text{term}_\text{variants}_\text{pred}_\text{iff}_\text{in}_\text{term}_\text{variants}[\text{of} \ "(\lambda_\_ \ [])(\text{Abs} \ a := [\text{Abs} \ b])"]

unfolding \text{timpl}_\text{apply}_\text{term}_\text{def} \ by \ (\text{metis} \ \text{full}_\text{types} \ \text{term}_\text{variants}_\text{pred}_\text{inv}(1), \ \text{metis} \ \text{full}_\text{types} \ \text{term}_\text{variants}_\text{pred}_\text{inv}(2), \ \text{fastforce} \ \text{dest}: \ \text{term}_\text{variants}_\text{pred}_\text{inv}(3))

lemma \text{timpl}_\text{apply}_\text{inv}':
\ assumes "s \in \text{set} \ (a \to b)(\text{Fun} \ f \ T)"
\ shows "\exists g S. s = \text{Fun} \ g \ S"

proof -
\ have \ "\text{term}_\text{variants}_\text{pred} \ ((\lambda_\_ \ [])(\text{Abs} \ a := [\text{Abs} \ b])) \ (\text{Fun} \ f \ T) \ s"
\ using \text{assms} \ \text{term}_\text{variants}_\text{pred}_\text{iff}_\text{in}_\text{term}_\text{variants}[\text{of} \ "(\lambda_\_ \ [])(\text{Abs} \ a := [\text{Abs} \ b])"]

unfolding \text{timpl}_\text{apply}_\text{term}_\text{def} \ by \ \text{force}

show \ "?thesis" \ using \ \text{term}_\text{variants}_\text{pred}_\text{cases}[\text{OF} \ *, \ \text{of} \ ?thesis] \ by \ \text{fastforce}

qed

lemma \text{timpl}_\text{apply}_\text{Term}_\text{Var}_\text{iff}:
\ "\text{Var} \ x \in \text{set} \ (a \to b)(t) = \leftrightarrow t = \text{Var} \ x"

using \text{term}_\text{variants}_\text{pred}_\text{iff}_\text{Term}_\text{Var}_\text{iff} \ \text{term}_\text{variants}_\text{pred}_\text{iff}_\text{in}_\text{term}_\text{variants}

unfolding \text{timpl}_\text{apply}_\text{term}_\text{def} \ by \ \text{metis}

3.5.2 Term Implication Closure

inductive_set \text{timpl}_\text{closure} \ for \ t \ TI \ where
\ FP: \ "t \in \text{timpl}_\text{closure} \ t \ TI"
\ TI: \ "\{u \in \text{timpl}_\text{closure} \ t \ TI; (a,b) \in TI; \ \text{term}_\text{variants}_\text{pred} \ ((\lambda_\_ \ [])(\text{Abs} \ a := [\text{Abs} \ b])) \ u \ s\}
\ \Rightarrow s \in \text{timpl}_\text{closure} \ t \ TI"

definition "\text{timpl}_\text{closure}_\text{set} M TI \equiv (\bigcup t \in M. \text{timpl}_\text{closure} \ t \ TI)"

inductive_set \text{timpl}_\text{closure}'_\text{step} \ for \ TI \ where
\ "\{ (a,b) \in TI; \ \text{term}_\text{variants}_\text{pred} \ ((\lambda_\_ \ [])(\text{Abs} \ a := [\text{Abs} \ b])) \ t \ s\}
\ \Rightarrow (t,s) \in \text{timpl}_\text{closure}'_\text{step} \ TI"

definition "\text{timpl}_\text{closure}' \ TI \equiv (\text{timpl}_\text{closure}'_\text{step} \ TI)""

definition \text{comp}_\text{timpl}_\text{closure} \ where
\ "\text{comp}_\text{timpl}_\text{closure} \ FP \ TI \equiv
\ \lambda f = \lambda x. \ FP \cup \ (\bigcup x \in X. \bigcup (a,b) \in TI. \ \text{set} \ (a \to b)(x))
\ \text{in while} \ (\lambda x. \ f \ X \neq X) \ f \ {}"

definition \text{comp}_\text{timpl}_\text{closure}_\text{list} \ where
\ "\text{comp}_\text{timpl}_\text{closure}_\text{list} \ FP \ TI \equiv
\ \lambda f = \lambda x. \ \text{remdups} \ (\text{concat} \ (\text{map} \ (\lambda x. \ \text{concat} \ (\text{map} \ (\lambda(a,b). \ (a \to b)(x)) \ TI) \ X) @ X))
\ \text{in while} \ (\lambda x. \ \text{set} \ (f \ X) \neq \text{set} \ X) \ f \ FP"

lemma \text{timpl}_\text{closure}_\text{set}_\text{I}:
\ "t \in M \Longrightarrow t \in \text{timpl}_\text{closure}_\text{set} M TI"
3.5 Term Implication

unfolding timpl_closure_set_def by (auto intro: timpl_closure.FP)

lemma timpl_closure_set_empty_timpls:
  "timpl_closure t {} = {t}" (is "?A = ?B")
proof (intro subset_antisym subsetI)
  fix s show "s ∈ ?A ⇒ s ∈ ?B"
    by (induct s rule: timpl_closure.induct) auto
qed (simp add: timpl_closure.FP)

lemmas timpl_closure_set_is_timpl_closure_union = meta_eq_to_obj_eq[OF timpl_closure_set_def]

lemma term_variants_pred_eq_case_Abs:
  fixes a b
  defines "P ≡ (λ _. [])(Abs a := [Abs b])"
  assumes "term_variants_pred P t s" "∀ f ∈ funs_term s. ¬ is_Abs f"
  shows "t = s"
using assms(2,3)
proof (induction t s rule: term_variants_pred.induct)
  case (term_variants_Fun T S f)
  have "¬ is_Abs h" when i: "i < length S" and h: "h ∈ funs_term (S ! i)" for i h
    using i h term_variants_Fun.prems by auto
  hence "T ! i = S ! i" when i: "i < length T" for i
    using i term_variants_Fun.hyps(1) term_variants_Fun.IH by auto
  hence "T = S" using term_variants_Fun.hyps(1) nth_equalityI[of T S] by fast
  thus ?case using term_variants_Fun.hyps(1) by blast
qed (simp_all add: term_variants_pred_refl P_def)

lemma timpl_closure'_step_inv:
  assumes "(t,s) ∈ timpl_closure'_stepTI"
  obtains a b where "(a,b) ∈ TI" "term_variants_pred ((λ _. [])(Abs a := [Abs b])) t s"
using assms by (auto elim: timpl_closure'_step.cases)

lemma timpl_closure_mono:
  assumes "TI ⊆ TI'"
  shows "timpl_closure t TI ⊆ timpl_closure t TI'"
proof
  fix s show "s ∈ timpl_closure t TI ⇒ s ∈ timpl_closure t TI'"
    apply (induct rule: timpl_closure.induct)
    using assms by (auto intro: timpl_closure.intros)
qed

lemma timpl_closure_set_mono:
  assumes "M ⊆ M'" "TI ⊆ TI'"
  shows "timpl_closure_set M TI ⊆ timpl_closure_set M' TI'"
using assms(1) timpl_closure_mono[OF assms(2)] unfolding timpl_closure_set_def by fast

lemma timpl_closure_idem:
  "timpl_closure_set (timpl_closure t TI) TI = timpl_closure t TI" (is "?A = ?B")
proof
  have "s ∈ timpl_closure t TI"
    when "s ∈ timpl_closure u TI" "u ∈ timpl_closure t TI" for s u
    using that
    by (induction rule: timpl_closure.induct)
    (auto intro: timpl_closure.intros)
  thus "?A ⊆ ?B" unfolding timpl_closure_set_def by blast
show "?B ⊆ ?A"
  unfolding timpl_closure_set_def
  by (blast intro: timpl_closure.FP)
qed

lemma timpl_closure_set_idem:
"\text{timpl\_closure\_set (timpl\_closure\_set M TI) TI = timpl\_closure\_set M TI}"
using timpl\_closure\_idem[of _ TI]unfolding timpl\_closure\_set\_def by auto

lemma timpl\_closure\_set\_mono_timpl\_closure\_set:
  assumes N: "N \subseteq timpl\_closure\_set M TI"
  shows "timpl\_closure\_set N TI \subseteq timpl\_closure\_set M TI"
using timpl\_closure\_set\_mono[OF N, of TI TI] timpl\_closure\_set\_idem[of M TI] by simp

lemma timpl\_closure\_is_timpl\_closure':
  "s \in timpl\_closure t TI \iff (t,s) \in timpl\_closure' TI"
proof
  show "s \in timpl\_closure t TI \implies (t,s) \in timpl\_closure' TI"
  unfolding timpl\_closure\'_def
  by (induct rule: timpl\_closure\_induct)
    (auto intro: rtrancl\_into\_rtrancl timpl\_closure\'_step\_intros)

  show "(t,s) \in timpl\_closure' TI \implies s \in timpl\_closure t TI"
  unfolding timpl\_closure\'_def
  by (induct rule: rtrancl\_induct)
    (auto dest: timpl\_closure\'_step\_inv intro: timpl\_closure\_FP timpl\_closure\_TI)
qed

lemma timpl\_closure\'_mono:
  assumes "TI \subseteq TI'"
  shows "timpl\_closure' TI \subseteq timpl\_closure' TI'"
using timpl\_closure\_mono[OF assms] timpl\_closure\_is_timpl\_closure'[of _ _ TI] timpl\_closure\_is_timpl\_closure'[of _ _ TI'] by fast

lemma timpl\_closure\_ton\_is_timpl\_closure:
  "timpl\_closure\_set \{t\} TI = timpl\_closure t TI"
by (simp add: timpl\_closure\_set\_is_timpl\_closure\_union)

lemma timpl\_closure\_'\_timpl\_s\_trancl\_subset:
  "timpl\_closure' (c +) \subseteq timpl\_closure' c"
unfolding timpl\_closure\'_def
proof
  fix s t :: "(('a,'b,'c,'d) prot\_fun,'e) term"
  show "(s,t) \in (timpl\_closure\_step (c +)) \ast \implies (s,t) \in (timpl\_closure\_step c) \ast"
  proof (induction rule: rtrancl\_induct)
    case (step u t)
    obtain a b where ab:
      "(a,b) \in c +" "term\_variants\_pred ((\_ _. []) \{Abs a := [Abs b]\}) u t"
    using step.hyps(2) timpl\_closure\_'\_step\_inv by blast
    hence "(u,t) \in (timpl\_closure\_step c)\"
    proof (induction arbitrary: t rule: trancl\_induct)
      case (step d e)
      obtain s where s:
        "term\_variants\_pred ((\_ _. []) \{Abs a := [Abs d]\}) s t"
        "term\_variants\_pred ((\_ _. []) \{Abs d := [Abs e]\}) s t" 
        using term\_variants\_pred\_dense\'_OF step\_prems, of "Abs d"] by blast
      have "(u,s) \in (timpl\_closure\_step c)\" 
      "(s,t) \in timpl\_closure\_step c"
      using step.hyps(2) s(2) step\_IH[OF s(1)]
      by (auto intro: timpl\_closure\_'\_step\_intros)
      thus ?case by simp
    qed (auto intro: timpl\_closure\_'\_step\_intros)
    thus ?case using step\_IH by simp
  qed simp

140
3.5 Term Implication

lemma timpl_closure'_timpls_trancl_subset':
    "timpl_closure' \{(a,b) ∈ c⁺. a ≠ b\} ⊆ timpl_closure' c"
using timpl_closure'_timpls_trancl_subset
    timpl_closure'_mono[of "\{(a,b) ∈ c⁺. a ≠ b\}" c] by fast

lemma timpl_closure_set_timpls_trancl_subset:
    "timpl_closure_set M (c⁺) ⊆ timpl_closure_set M c"
using timpl_closure'_timpls_trancl_subset[of c]
    timpl_closure_is_timpl_closure'[of _ _ c]
    timpl_closure_is_timpl_closure'[of _ "c⁺"]
    timpl_closure_set_is_timpl_closure_union[of M c]
    timpl_closure_set_is_timpl_closure_union[of M "c⁺"] by fastforce

lemma timpl_closure_set_timpls_trancl_subset':
    "timpl_closure_set M \{(a,b) ∈ c⁺. a ≠ b\} ⊆ timpl_closure_set M c"
using timpl_closure'_timpls_trancl_subset'[of c]
    timpl_closure_is_timpl_closure'[of _ _ "c⁺"]
    timpl_closure_set_is_timpl_closure_union[of M c]
    timpl_closure_set_is_timpl_closure_union[of M "\{(a,b) ∈ c⁺. a ≠ b\}"]
by fastforce

lemma timpl_closure'_timpls_trancl_supset':
    "timpl_closure' c ⊆ timpl_closure' \{(a,b) ∈ c⁺. a ≠ b\}" unfolding timpl_closure'_def
proof
    let ?cl = "\{(a,b) ∈ c⁺. a ≠ b\}"
    fix s t::"(('a,'b,'c,'d) prot_fun,'e) term"
    show "(s,t) ∈ (timpl_closure'_step c)∗ ⇒ (s,t) ∈ (timpl_closure'_step ?cl)∗"
    proof (induction rule: rtrancl_induct)
        case (step u t)
        obtain a b where ab:
            "(a,b) ∈ c" "term_variants_pred ((λ_. [])\{Abs a := [Abs b]\}) u t"
        using step.hyps(2) timpl_closure'_step_inv by blast
        hence "(a,b) ∈ c⁺" by simp
        hence "(u,t) ∈ (timpl_closure'_step ?cl)∗" using ab(2)
        proof (induction arbitrary: t rule: trancl_induct)
            case (base d)
            show ?case
            proof (cases "a = d")
                case True thus ?thesis
                using base term_variants_pred_refl_inv[of _ u t] by force
            next
                case False thus ?thesis
                using base timpl_closure'_step.intros[of a d ?cl]
                by fast
            qed
        next
        case (step d e)
        obtain s where s:
            "term_variants_pred ((λ_. [])\{Abs a := [Abs d]\}) u s"
            "term_variants_pred ((λ_. [])\{Abs d := [Abs e]\}) s t"
        using term_variants_pred_dense'[OF step.prems, of "Abs d"] by blast
        show ?case
        proof (cases "d = e")
            case True thus ?thesis
        qed
    qed

qed
using step.prems step.IH[of t]
by blast
next
case False
hence "((u,s) ∈ (timpl_closure'_step ?cl)"
  
  "(s,t) ∈ (timpl_closure'_step ?cl)"
using step.hyps(2) s(2) step.IH[OF s(1)]
by (auto intro: timpl_closure'_step.intros)
thus ?thesis by simp
qed

due ?case using step.IH by simp
qed simp

lemma timpl_closure'_timpls_trancl_subset:
  "timpl_closure' c ⊆ timpl_closure' (c+)"
using timpl_closure'_timpls_trancl_subset[of c]
timpl_closure'_mono[of "{(a,b) ∈ c+. a ≠ b}" "c+"]
by fast

lemma timpl_closure'_timpls_trancl_eq:
  "timpl_closure' (c+) = timpl_closure' c"
using timpl_closure'_timpls_trancl_subset timpl_closure'_timpls_trancl_supset
by blast

lemma timpl_closure'_timpls_trancl_eq':
  "timpl_closure' {(a,b) ∈ c+. a ≠ b} = timpl_closure' c"
using timpl_closure'_timpls_trancl_subset' timpl_closure'_timpls_trancl_supset'
by blast

lemma timpl_closure'_timpls_rtrancl_subset:
  "timpl_closure' (c∗) ⊆ timpl_closure' c"
unfolding timpl_closure'_def
proof
  fix s t :: 
  
  "((a,b,c,d) prot Fun,'e) term"
  show "((s,t) ∈ (timpl_closure'_step (c∗))) =⇒ (s,t) ∈ (timpl_closure'_step c)"
  proof (induction rule: rtrancl_induct)
  case (step u t)
  obtain a b where ab:
    "(a,b) ∈ c" "term variants pred ((λ_. []). (Abs a := [Abs b])) u t"
  using step.hyps(2) timpl_closure'_step_inv by blast
  hence "(u,t) ∈ (timpl_closure'_step c)"
  proof (induction arbitrary: t rule: rtrancl_induct)
  case base
  hence "u = t" using term variants pred refl_inv by fastforce
  thus ?case by simp
  next
case (step d e)
  obtain s where s:
    "term variants pred ((λ_. []). (Abs a := [Abs d])) u s"
    "term variants pred ((λ_. []). (Abs d := [Abs e])) s t"
  using term variants pred dense' [OF step.prems, of "Abs d"] by blast
  have "((u,s) ∈ (timpl_closure'_step c)"
    "(s,t) ∈ (timpl_closure'_step c)"
  using step.hyps(2) s(2) step.IH[OF s(1)]
  by (auto intro: timpl_closure'_step.intros)
  thus ?case by simp
  qed
  thus ?case using step.IH by simp
  qed simp
  qed

142
3.5 Term Implication

lemma timpl_closure'_timpls_rtrancl_subset:
"timpl_closure' c ⊆ timpl_closure' (c')"

unfolding timpl_closure'_def

proof
  fix s t::"(('a,'b,'c,'d) prot_fun,'e) term"
  show "(s,t) ∈ (timpl_closure'_step c)∗ ≡ (s,t) ∈ (timpl_closure'_step (c'))"*

proof (induction rule: rtrancl_induct)
  case (step u t)
  obtain a b where "(a,b) ∈ c" "term_variants_pred ((λ_. [])(Abs a := [Abs b])) u t" using step.hyps(2) timpl_closure'_step_inv by blast
  hence "(a,b) ∈ c" by simp
  hence "(u,t) ∈ (timpl_closure'_step (c'))" using ab(2)

proof (induction arbitrary: t rule: rtrancl_induct)
  case (base t) thus ?case using term_variants_pred_refl_inv[of _ u t] by fastforce
  next
  case (step d e)
  obtain s where "term_variants_pred ((λ_. [])(Abs a := [Abs d])) u s" "term_variants_pred ((λ_. [])(Abs d := [Abs e])) s t" using term_variants_pred_dense'[OF step.prems, of "Abs d"] by blast
  show ?case
  proof (cases "d = e")
    case True thus ?thesis using step.prems step.IH[of t] by blast
    next
    case False hence "(u,s) ∈ (timpl_closure'_step (c'))" "(s,t) ∈ (timpl_closure'_step (c'))" using step.hyps(2) s(2) step.IH[OF s(1)] by (auto intro: timpl_closure'_step.intros)
    thus ?thesis by simp
  qed
  qed simp
  qed

lemma timpl_closure'_timpls_rtrancl_eq:
"timpl_closure' (c') = timpl_closure' c"

using timpl_closure'_timpls_rtrancl_subset timpl_closure'_timpls_rtrancl_subset by blast

lemma timpl_closure'_timpls_trancl_eq:
"timpl_closure t (c') = timpl_closure t c"

using timpl_closure'_timpls_trancl_eq[of c]
  timpl_closure_is_timpl_closure[of _ _ c]
  timpl_closure_is_timpl_closure[of _ _ "c'"] by fastforce

lemma timpl_closure_set_timpls_trancl_eq:
"timpl_closure_set M (c') = timpl_closure_set M c"

using timpl_closure_set_timpls_trancl_eq
  timpl_closure_set_is_timpl_closure_union[of M c]
  timpl_closure_set_is_timpl_closure_union[of M "c'"] by fastforce

lemma timpl_closure_set_timpls_trancl_eq'::
"timpl_closure_set M {(a,b) ∈ c'. a ≠ b} = timpl_closure_set M c"
using `timpl_closure'_timpls_trancl_eq[of c]
  timpl_closure_is_timpl_closure[of _ _ c]
  timpl_closure_is_timpl_closure[of _ _ "{(a,b) ∈ c⁺. a ≠ b}"
  timpl_closure_set_is_timpl_closure_union[of M c]
  timpl_closure_set_is_timpl_closure_union[of M "{(a,b) ∈ c⁺. a ≠ b}"
by fastforce

lemma timpl_closure_Var_in_iff:
  "Var x ∈ timpl_closure t TI ←→ t = Var x" (is "?A ←→ ?B")
proof
  have "s ∈ timpl_closure t TI ⇒ s = Var x ⇒ s = t" for s
    apply (induction rule: timpl_closure.induct)
    by (simp, metis term_variants_pred_inv_Var(2))
  thus "?A ⇒ ?B" by blast
qed (blast intro: timpl_closure.FP)

lemma timpl_closure_set_Var_in_iff:
  "Var x ∈ timpl_closure_set M TI ←→ Var x ∈ M"
unfolding timpl_closure_set_def by (simp add: timpl_closure_Var_in_iff[of x _ TI])

lemma timpl_closure_Var_inv:
  assumes "t ∈ timpl_closure (Var x) TI"
  shows "t = Var x"
using assms proof
  (induction rule: timpl_closure.induct)
  case (TI u a b s)
  thus ?case using term_variants_pred_inv.Var by fast
qed simp

lemma timpls_Un_mono: "mono (λX. FP ∪ (⋃ x ∈ X. ⋃ (a,b) ∈ TI. set (a → b)(x)))"
by (auto intro!: monoI)

lemma timpl_closure_set_lfp:
  fixes M TI
  defines "f ≡ (λX. M ∪ (⋃ x ∈ X. ⋃ (a,b) ∈ TI. set (a → b)(x)))"
  shows "lfp f = timpl_closure_set M TI"
proof
  note 0 = timpls_Un_mono[of M TI, unfolded f_def[symmetric]]
  let ?N = "timpl_closure_set M TI"
  show "lfp f ⊆ ?N" (is "?B")
    proof (induction rule: lfp_induct)
      case 2 thus ?case proof
        fix t assume "t ∈ f (lfp f ∩ ?N)"
        hence "t ∈ M ∨ t ∈ (⋃ x ∈ ?N. ⋃ (a,b) ∈ TI. set (a → b)(x))" (is "?A ∨ ?B")
          unfolding f_def by blast
        thus "t ∈ ?N" proof
          assume ?B
          then obtain s a b where s: "s ∈ ?N" "(a,b) ∈ TI" "t ∈ set (a → b)(s)" by moura
          thus ?thesis using term_variants_pred_iff_in_term_variants[of "(λ_. [])(Abs a := [Abs b])" s]
            unfolding timpl_closure_set_def timpl_apply_term_def
            by (auto intro: timpl_closure.intros)
        qed (auto simp add: timpl_closure_set_def intro: timpl_closure.intros)
    qed (rule 0)
  qed

have "t ∈ lfp f" when t: "t ∈ timpl_closure s TI" and s: "s ∈ M" for t s
  using t
proof (induction t rule: timpl_closure.induct)
  case (TI u a b v) thus ?case

144
3.5 Term Implication

using term_variants_pred_iff_in_term_variants[of "\(\lambda_\cdot ~[]\)(Abs a := [Abs b])"]

\textit{lfp\_fixpoint}[\text{OF 0}]

unfolding \textit{timpl\_apply\_term\_def} f_def by fastforce

\textbf{lemmas} \textit{timpl\_closure\_set\_supset:}
\begin{itemize}
  \item \textit{assumes} "\(\forall t \in FP. \, t \in \text{closure}\)"
  \item \textit{and} "\(\forall t \in \text{closure}. \, \forall (a,b) \in \text{TI}. \, \forall s \in \text{set} \langle a \rightarrow b \rangle(t). \, s \in \text{closure}\)"
  \item \textit{shows} "\(\text{timpl\_closure\_set FP TI} \subseteq \text{F}\)"
\end{itemize}

\textbf{proof -}
\begin{itemize}
  \item \textit{have} "\(t \in \text{closure}\) when \(t: t \in \text{timpl\_closure s TI}\) and \(s \in \text{FP}\) for \(t \, s\) using \(t\) proof (induction rule: timpl\_closure\_induct)
  \item case \((\text{TI} \cup \text{a \ b\ s'})\) thus ?case using \(\text{assms}(1)\) unfolding \textit{timpl\_apply\_term\_def} by fastforce
\end{itemize}

\textbf{thus} ?thesis unfolding \textit{timpl\_closure\_set\_def} by blast

\textbf{qed}

\textbf{lemmas} \textit{timpl\_closure\_set\_supset':}
\begin{itemize}
  \item \textit{assumes} "\(\forall t \in \text{FP}. \, \forall (a,b) \in \text{TI}. \, \forall s \in \text{set} \langle a \rightarrow b \rangle(t). \, s \in \text{FP}\)"
  \item \textit{shows} "\(\text{timpl\_closure\_set FP TI} \subseteq \text{FP}\)"
\end{itemize}

\textbf{using} \textit{timpl\_closure\_set\_supset}[\text{OF _ assms}] by blast

\textbf{lemmas} \textit{timpl\_closure\_'param:}
\begin{itemize}
  \item \textit{assumes} "\((t,s) \in \text{timpl\_closure\'} c\)" and \(f g: \text{if} = g \lor (\exists a b. \, (a,b) \in c \land f = \text{Abs a} \land g = \text{Abs b})\)"
  \item \textit{shows} "\((\text{Fun f (S@s#T)}, \text{Fun g (S@s#T)}) \in \text{timpl\_closure\'} c\)"
\end{itemize}

\textbf{using} \textit{assms}(1) unfolding \textit{timpl\_closure\_'def}

\textbf{proof} (induction rule: \textit{rtrancl\_induct})
\begin{itemize}
  \item \textit{case base thus ?case using \textit{assms}\_intros}\_\textit{by moura}
  \item \textit{case (\text{TI} \cup \text{a \ b\ s'}) thus ?case using \textit{assms}(2) unfolding \textit{timpl\_closure\_'def} by fastforce
\end{itemize}

\textbf{thus} ?thesis unfolding \textit{timpl\_closure\_set\_def} by blast

\textbf{qed}

\textbf{lemmas} \textit{timpl\_closure\_'param':}
\begin{itemize}
  \item \textit{assumes} "\((t,s) \in \text{timpl\_closure\'} c\)"
  \item \textit{shows} "\((\text{Fun f (S@s#T)}, \text{Fun f (S@s#T)}) \in \text{timpl\_closure\'} c\)"
\end{itemize}

\textbf{using} \textit{timpl\_closure\_'param}[\text{OF _ assms}] by simp

\textbf{lemmas} \textit{timpl\_closure\_FunI:}
\begin{itemize}
  \item \textit{assumes} \textit{IH}: "\(\forall i. \, i < \text{length} \, T \implies (T ! i, S ! i) \in \text{timpl\_closure\'} c\)"
  \item \textit{and len: "\text{length} \, T = \text{length} \, S\)"
\end{itemize}
and \(fg: f = g \lor (\exists a\ b. (a, b) \in c^+ \land f = \text{Abs} a \land g = \text{Abs} b)\)
shows "(Fun f T, Fun g S) \in \text{timpl\_closure'} c"
proof 
have aux: "(Fun f T, Fun g (take n S@drop n T)) \in \text{timpl\_closure'} c"
when "n \leq \text{length} T" for n
using that
proof (induction n)
case 0
have "(T ! n, T ! n) \in \text{timpl\_closure'} c" when "n < \text{length} T" for n
using ?thesis
proof 
let ?thesis = "(Fun f T, Fun g (take n S@drop n T)) \in \text{timpl\_closure'} c"
next 
case (Suc n)
hence "(Fun f T, Fun g (take n S@drop n T)) \in \text{timpl\_closure'} c"
and "n < \text{length} T" by simp
ultimately show ?thesis using \(timpl\_closure'_\text{param} \) IH' (1)
by \(\text{metis} \ \text{timpl\_closure'} - \text{rtrancl}\text\_trans\)
qed
show ?thesis using aux[of "\text{length} T"] len by simp
qed

lemma \(timpl\_closure\_FunI'\):
assumes IH: "\(\forall i. i < \text{length} T \implies (T ! i, S ! i) \in \text{timpl\_closure'} c\)"
and len: "\(\text{length} T = \text{length} S\)"
shows "(Fun f T, Fun f S) \in \text{timpl\_closure'} c"
using \(\text{timpl\_closure\_FunI}[OF IH len]\) by simp

lemma \(timpl\_closure\_FunI2\):
fixes f g: "\("a, \ b, \ c, \ d\) \text{ prot\_fun}\"
assumes IH: "\(\forall i. i < \text{length} T \implies \exists u. (T ! i, u) \in \text{timpl\_closure'} c \land (S ! i, u) \in \text{timpl\_closure'} c\)"
and len: "\(\text{length} T = \text{length} S\)"
and fg: "f = g \lor (\exists a\ b d. (a, d) \in c^+ \land (b, d) \in c^+ \land f = \text{Abs} a \land g = \text{Abs} b)"
shows "\(\exists h. U. (Fun f T, Fun h U) \in \text{timpl\_closure'} c \land (Fun g S, Fun h U) \in \text{timpl\_closure'} c\)"
proof 
let \(\lambda U. (Fun f T, Fun h U) \in \text{timpl\_closure'} c \land (Fun g S, Fun h U) \in \text{timpl\_closure'} c\)
define $U$ where 

\[ U \equiv \text{map (}\lambda i. \text{SOME } u. ?P i u) [0..<\text{length } T] \]

have U1: "\text{length } U = \text{length } T"
  unfolding U_def by auto

have U2: "\text{(}T ! i, U ! i) \in \text{timpl\_closure'} C \land \text{(}S ! i, U ! i) \in \text{timpl\_closure'} C"
  when i: "i < \text{length } U"
  using i someI_ex[of "?P i"] IH[of i] U1 len
  unfolding U_def by simp

show \(?\text{thesis}\)
proof (cases "f = g"
  case False
  then obtain a b d
    where \(a, d) \in C + (b, d) \in C + \text{Abs } a = f \land \text{Abs } b = g\)
  using fg
  by (metis wt_subst_trm''[OF \(\text{_}\) \(\text{_}\)] wt_subst_const_fv_type_eq[OF _ \(\text{_}\) \(\text{_}\)] \\ _ \(\text{_}\) \(\text{_}\) \\ \(\text{_}\) \(\text{_}\) \(\text{_}\) \(\text{_}\) \(\text{_}\)  \\ wt_subst_const_fv_type_eq[OF _ \(\text{_}\) \(\text{_}\) \(\text{_}\) \(\text{_}\) \(\text{_}\) \(\text{_}\)] by blast+

thus \(?\text{thesis}\) by (metis timpl_closure_FunI len U1 U2)
qed (metis timpl_closure_FunI' len U1 U2)

lemma timpl_closure_FunI3:
fixes f g::"('a, 'b, 'c, 'd) prot_fun"
assumes IH: "\(\forall i. i < \text{length } T \Rightarrow \exists u. (T!i, u) \in \text{timpl\_closure'} C \land (S!i, u) \in \text{timpl\_closure'} C\)"
and len: "\text{length } T = \text{length } S"
and fg: "f = g \lor (\\exists a b d. (a, d) \in C \land (b, d) \in C \land f = \text{Abs } a \land g = \text{Abs } b)"
shows "\exists h U. (\text{Fun } f T, \text{Fun } h U) \in \text{timpl\_closure'} C \land (\text{Fun } g S, \text{Fun } h U) \in \text{timpl\_closure'} C"
proof (metis timpl_closure_FunI2[OF IH len] fg unfolding timpl_closure'_timpls_trancl_eq by blast)

lemma timpl_closure_fv_eq:
assumes "s \in \text{timpl\_closure } t T"
shows "fv s = fv t"
using assms
by (induct rule: timpl_closure.induct)
  (metis, metis term_variants_pred_fv_eq)

lemma (in stateful_protocol_model) timpl_closure_subst:
assumes t: "\text{wf } trm } t" "\forall x \in \text{fv } t. \exists a. \Gamma_v x = TAtom (\text{Atom } a)"
and \(\delta\): "\text{wt } subst } \delta" "\text{wt } subst\_range } \delta"
shows "\text{timpl\_closure } (t \cdot \delta) T = \text{timpl\_closure } t \cdot set } \delta"
proof
have "s \in \text{timpl\_closure } t T \cdot set } \delta"
  when "s \in \text{timpl\_closure } (t \cdot \delta) T" for s
  using that
proof (induction s rule: timpl_closure.induct)
  case FP thus \(?\text{case}\) using timpl_closure.FP[of t T] by simp
next
  case (TI u a b s)
  then obtain u' where u': "u' \in \text{timpl\_closure } t T" "u = u' \cdot \delta" by moura
  have u'_fv: "\forall x \in \text{fv } u'. \exists a. \Gamma_v x = TAtom (\text{Atom } a)"
    unfolding timpl_closure_fv_eq[OF u'(1)] t(2) by simp
  hence u_fv: "\forall x \in \text{fv } u. \exists a. \Gamma_v x = TAtom (\text{Atom } a)"
    unfolding timpl_closure_fv_eq[OF _ (1)] t(1) (1,2, of u')
    using _ by fastforce
  have "\forall x \in \text{fv } u' \cup \text{fv } s. (\exists y. \delta x = \text{Var } y) \lor (\exists f. \delta x = \text{Fun } f [] \land \text{Abs } a \neq f)"
    proof (intro ballI)
fix x assume: "x ∈ fv u' ∪ fv s"
then obtain c where c: "Γ v x = TAtom (Atom c)"
using u'_fv u_fv term_variants_pred_fv_eq[OF TI.hyps(3)]
by blast

show "(∃y. δ x = Var y) ∨ (∃f. δ x = Fun f [] ∧ Abs a ≠ f)"
proof (cases "δ x")
case (Fun f T)
hence **: "Γ (Fun f T) = TAtom (Atom c)" and "wf_trm (Fun f T)"
using c wt_subst_trm''[OF δ(1), of "Var x"] δ(2)
by fastforce+
hence "δ x = Fun f []" using Fun const_type_inv_wf by metis
thus ?thesis using ** by force
qed

hence *: "∀ x ∈ fv u' ∪ fv s. (∃y. δ x = Var y) ∨ (∃f. δ x = Fun f [] ∧ Abs a ≠ f)"
by fastforce

obtain s' where s': "term_variants_pred ((λ_. []))(Abs a := [Abs b])) u' s'" "s = s' · δ"
using term_variants_pred_subst'[OF _ *] u'(2) TI.hyps(3)
by blast

show ?case using timpl_closure.TI[OF u'(1) TI.hyps(2) s'(1)] s'(2) by blast
qed

thus "timpl_closure (t · δ) T ⊆ timpl_closure t T · set δ" by fast

have "s ∈ timpl_closure (t · δ) T"
when s: "s ∈ timpl_closure t T · set δ" for s
proof -
obtain s' where s': "s' ∈ timpl_closure t T" "s = s' · δ" using s by moura
have "s' · δ ∈ timpl_closure (t · δ) T" using s'(1)
proof (induction s' rule: timpl_closure.induct)
case FP thus ?case using timpl_closure.FP[of "t · δ"] T by simp
next
case (TI u' a b s') show ?case
using timpl_closure.TI[OF TI.INH TI.hyps(2)]
  term_variants_pred_subst[OF TI.hyps(3)]
by blast
qed
thus ?thesis using s'(2) by metis
qed

thus "timpl_closure t T · set δ ⊆ timpl_closure (t · δ) T" by fast
qed

lemma (in stateful_protocol_model) timpl_closure_subst_subset:
assumes t: "t ∈ M" and M: "wf_trms M" "∀ x ∈ fv_M. ∃a. Γ v x = TAtom (Atom a)"
and δ: "wt_subst δ" "wf_trms (subst_range δ)" "ground (subst_range δ)" "subst_domain δ ⊆ fv_M" and M_supset: "timpl_closure t T ⊆ M"
shows "timpl_closure (t · δ) T ⊆ M · set δ"
proof -
have t': "wf_trms t" "∀ x ∈ fv t. ∃a. Γ v x = TAtom (Atom a)" using t M by auto
show ?thesis using timpl_closure_subst[OF t' δ(1,2), of T] M_supset by blast
qed

lemma (in stateful_protocol_model) timpl_closure_set_subst_subset:
assumes M: "wf_trms M" "∀ x ∈ fv_M. ∃a. Γ v x = TAtom (Atom a)"
and δ: "wt_subst δ" "wf_trms (subst_range δ)" "ground (subst_range δ)" "subst_domain δ ⊆ fv_M" and M_supset: "timpl_closure_set M T ⊆ M" shows "timpl_closure_set (M · set δ) T ⊆ M · set δ"
using timpl_closure_subst_subset[OF _ M δ, of _ T] M_supset timpl_closure_set_is_timpl_closure_union[of "M · set δ"]
3.5 Term Implication

timpl_closure_set_is_timpl_closure_union[of M T]
by auto

lemma timpl_closure_set_Union:
  "timpl_closure_set (\bigcup M \in Ms. timpl_closure_set M T)"
using timpl_closure_set_is_timpl_closure_union[of "\bigcup M \in Ms. T"
  timpl_closure_set_is_timpl_closure_union[of _ T]
by force

lemma timpl_closure_set_Union_subst_set:
  assumes "s \in timpl_closure_set (\bigcup \{M \cdot set \delta \mid \delta \cdot P \delta\}) T"
shows "\exists \delta. P \delta \land s \in timpl_closure_set (M \cdot set \delta) T"
using assms timpl_closure_set_is_timpl_closure_union[of "\bigcup \{M \cdot set \delta \mid \delta \cdot P \delta\}" T]
timpl_closure_set_is_timpl_closure_union[of _ T]
by blast

lemma timpl_closure'_inv:
  assumes "(s, t) \in timpl_closure' TI"
shows "\exists x. s = Var x \land t = Var x \lor \exists f g S T. s = Fun f S \land t = Fun g T \land length S = length T"
using assms unfolding timpl_closure'_def
proof (induction rule: rtrancl_induct)
  case base thus ?case by (cases s) auto
next
  case (step t u)
  obtain a b where ab: "(a, b) \in TI" "term_variants_pred ((\lambda _. [])(Abs a := [Abs b])) t u"
  using timpl_closure'_step_inv[OF step.hyps(2)] by blast
  show ?case using step.IH proof
    assume "\exists x. s = Var x \land t = Var x"
    thus ?case using step.hyps(2) term_variants_pred_inv_Var ab by fastforce
  next
    assume "\exists f g S T. s = Fun f S \land t = Fun g T \land length S = length T"
    then obtain f g S T where st: "s = Fun f S" "t = Fun g T" "length S = length T" by moura
    thus ?case
    using ab step.hyps(2) term_variants_pred_inv[of "(\lambda _. [])(Abs a := [Abs b])" g T u]
    by auto
  qed
qed

lemma timpl_closure'_inv':
  assumes "(s, t) \in timpl_closure' TI"
shows "\exists x. s = Var x \land t = Var x \lor \exists f g S T. s = Fun f S \land t = Fun g T \land length S = length T \land
  (\forall i < length T. (S ! i, T ! i) \in timpl_closure' TI) \land
  (f \neq g \longrightarrow is_Abs f \land is_Abs g \land (the_Abs f, the_Abs g) \in TI^+))"
(is "?A s t \lor ?B s t (timpl_closure' TI)")
using assms unfolding timpl_closure'_def
proof (induction rule: rtrancl_induct)
  case base thus ?case by (cases s) auto
next
  case (step t u)
  obtain a b where ab: "(a, b) \in TI" "term_variants_pred ((\lambda _. [])(Abs a := [Abs b])) t u"
  using timpl_closure'_step_inv[OF step.hyps(2)] by blast
  show ?case using step.IH proof
    assume "?A s t"
thus \( ?\text{case using step.hyps(2) term_variants_pred_inv_Var ab by fastforce} \)

next

assume "\( \exists \text{s t } ((\text{timpl_closure\'_step TI} ))\)"

then obtain \( f \ g \ S \ T \) where \( st \):

\[
\begin{align*}
&\langle i. i < \text{length T} \implies (S ! i, T ! i) \in (\text{timpl_closure\'_step TI}) \rangle \\
&\langle f \neq g \implies \text{is_Abs f } \wedge \text{is_Abs g } \wedge (\text{the_Abs f }, \text{the_Abs g }) \in \text{TI}^+ \rangle \\
&\langle g \neq h \implies \text{is_Abs g } \wedge \text{is_Abs h } \wedge (\text{the_Abs g }, \text{the_Abs h }) \in \text{TI}^+ \rangle
\end{align*}
\]

by moura

obtain \( u \) where

\[
\begin{align*}
&\langle u = \text{Fun h U } \rangle \\
&\langle i. i < \text{length U} \implies (S ! i, U ! i) \in (\text{timpl_closure\'_step TI}) \rangle \\
&\langle f \neq h \implies \text{is_Abs f } \wedge \text{is_Abs h } \wedge (\text{the_Abs f }, \text{the_Abs h }) \in \text{TI}^+ \rangle
\end{align*}
\]

unfolding \text{is_Abs_def the_Abs_def} by force

have "\( (S ! i, U ! i) \in (\text{timpl_closure\'_step TI}) \)" when \( i: \langle i < \text{length U} \rangle \)

using \( u(2) \ rtrancl.rtrancl_into_rtrancl[of \langle i \rangle \text{timpl_closure\'_step.intros[of \langle i \rangle \langle i \rangle \langle i \rangle \langle i \rangle \langle i \rangle \langle i \rangle \langle i \rangle \langle i \rangle \rangle }] \)

by argo

moreover have "\( \text{length S } = \text{length U}\)" using \( st \ u \) by argo

moreover have "\( \text{is_Abs f } \wedge \text{is_Abs h } \wedge (\text{the_Abs f }, \text{the_Abs h }) \in \text{TI}^+ \)" when \( fh: \langle f \neq h \rangle \)

using \( fh \ st \ u \) by fastforce

ultimately show \( ?\text{case using st(1) u(1) by blast} \)

qed

lemma \( \text{timpl_closure\'_inv''} \):

assumes "\( \langle \text{Fun f S, Fun g T } \rangle \in \text{timpl_closure\'} \text{ TI} \)"

shows "\( \text{length S } = \text{length T}\)"

and "\( \langle i. i < \text{length T } \implies (S ! i, T ! i) \in \text{timpl_closure\'} \text{ TI} \rangle \\
\langle f \neq g \implies \text{is_Abs f } \wedge \text{is_Abs g } \wedge (\text{the_Abs f }, \text{the_Abs g }) \in \text{TI}^+ \rangle \\
\langle g \neq h \implies \text{is_Abs g } \wedge \text{is_Abs h } \wedge (\text{the_Abs g }, \text{the_Abs h }) \in \text{TI}^+ \rangle"

using \( \text{assms timpl_closure\'_inv''} \) by auto

lemma \( \text{timpl_closure\'_Fun_inv} \):

assumes "\( s \in \text{timpl_closure } \text{Fun f T } \text{ TI} \)"

shows "\( \exists g S. s = \text{Fun g S}\)"

using \( \text{assms timpl_closure\'_is_timpl_closure\'_inv \text{ timpl_closure\'_inv} } \) by fastforce

lemma \( \text{timpl_closure\'_Fun_inv'} \):

assumes "\( \text{Fun g S } \in \text{timpl_closure } \text{Fun f T } \text{ TI} \)"

shows "\( \text{length S } = \text{length T}\)"

and "\( \langle i. i < \text{length S } \implies S ! i \in \text{timpl_closure } \text{ (T \ i) TI} \)" and "\( f \neq g \implies \text{is_Abs f } \wedge \text{is_Abs g } \wedge (\text{the_Abs f }, \text{the_Abs g }) \in \text{TI}^+ \)"

using \( \text{assms timpl_closure\'_is_timpl_closure\'} \) by (metis \( \text{timpl_closure\'_inv''(1), metis timpl_closure\'_inv''(2), metis timpl_closure\'_inv''(3)} \))

lemma \( \text{timpl_closure\'_Fun_not_Var[simp]} \):

"\( \text{Fun f T } \notin \text{timpl_closure } \text{Var x } \text{ TI} \)"

using \( \text{timpl_closure\'_Var_inv} \) by fast

lemma \( \text{timpl_closure\'_Var_not-Fun[simp]} \):

"\( \text{Var x } \notin \text{timpl_closure } \text{Fun f T } \text{ TI} \)"

using \( \text{timpl_closure\'_Fun_inv} \) by fast

lemma (in stateful_protocol_model) \( \text{timpl_closure_wf_trms} \):

assumes \( m: \langle \text{w_term } m \rangle \)

shows "\( \text{w_term } (\text{timpl_closure } m \text{ TI}) \)"

proof

fix t assume "\( t \in \text{timpl_closure } m \text{ TI} \)"

thus "\( \text{w_term } t \)"
3.5 Term Implication

proof (induction t rule: timpl_closure.induct)
  case TI thus ?case using term_variants_pred_wf_trms by force
qed (rule m)

qed

lemma (in stateful_protocol_model) timpl_closure_set_wf_trms:
  assumes M: "wf trms M"
  shows "wf trms (timpl_closure_set M TI)"
proof
  fix t assume "t ∈ timpl_closure_set M TI"
  then obtain m where "t ∈ timpl_closure m TI" "m ∈ M" "wf trm m"
    using M timpl_closure_set_is_timpl_closure_union by blast
  thus "wf trm t" using timpl_closure_wf_trms by blast
qed

lemma timpl_closure_Fu_inv:
  assumes "t ∈ timpl_closure (Fun (Fu f) T) TI"
  shows "∃ S. length S = length T ∧ t = Fun (Fu f) S"
proof (induction t rule: timpl_closure.induct)
  case (TI u a b)
  obtain g U where U: "u = Fun g U" "length U = length T" "Fu f = Abs a = g ∧ Fu f = Abs b"
    using term_variants_pred_inv''[OF TI.hyps(4)] by fastforce
  have g: "g = Fu f" using U(3) by blast
  show ?case using TI.hyps(2)[OF U(1)[unfolded g]] U(2) by auto
qed simp

lemma timpl_closure_no_Abs_eq:
  assumes "t ∈ timpl_closure s TI"
  and "∀ f ∈ funs_term t. ¬ is_Abs f"
  shows "t = s"
proof (induction t rule: timpl_closure.induct)
  case (TI t a b s) thus ?case
    using term_variants_pred_eq_case_Abs[of a b t s]
    unfolding timpl_apply_term_def term_variants_pred_iff_in_term_variants[symmetric]
    by metis
qed simp

lemma timpl_closure_set_no_Abs_in_set:
  assumes "t ∈ timpl_closure_set FP TI"
  and "∀ f ∈ funs_term t. ¬ is_Abs f"
  shows "t ∈ FP"
using assms timpl_closure_no_Abs_eq unfolding timpl Closure_set_def by blast

lemma timpl_closure_funs_term_subset:
  "\(\bigcup (funs_term \ \cdot (timpl_closure t TI)) \subseteq \bigcup (funs_term t \cup Abs \ \cdot \ and \ \cdot TI)\) (is \(\forall A \subseteq ?B \cup ?C\))
proof
  fix f assume "f \in ?A"
  then obtain s where "s \in timpl_closure t TI" "f \in funs_term s" by moura
  thus "f \in ?B \cup ?C"
proof (induction s rule: timpl Closure.induct)
  case (TI u a b s)
  have "Abs b \in Abs \cdot snd \cdot TI" using TI.hyps(2) by force
  thus \(\text{?case}\) using term_variants_pred_funs_term[OF TI.hyps(3) TI.prems] TI.IH by force
  qed blast
qed

lemma timpl_closure_set_funs_term_subset:
  "\(\bigcup (funs_term \ \cdot (timpl_closure_set FP TI)) \subseteq \bigcup (funs_term \ FP) \cup Abs \ \cdot \ snd \ \cdot TI)\"
using timpl Closure_funs_term_subset[of _ TI] timpl Closure_set_is_timpl Closure_union[of FP TI]
by auto

lemma funs_term_OCC_TI_subset:
defines "absc ≡ λa. Fun (Abs a) []"
assumes OCC1: "\(\forall t \in FP. \forall f \in funs_term t. is_Abs f \rightarrow f \in Abs \cdot OCC\)"
and OCC2: "snd \cdot TI \subseteq OCC"
shows "\(\forall t \in timpl_closure_set FP TI. \forall f \in funs_term t. is_Abs f \rightarrow f \in Abs \cdot OCC\) (is \(\forall A\))
and "\(\forall t \in absc \cdot OCC. \forall (a, b) \in TI. \forall s \in set (a --> b)(t). s \in absc \cdot OCC\) (is \(\forall B\))"
proof
  let \(\?F = \bigcup (funs_term \ \cdot FP)\)
  let \(\?G = Abs \cdot snd \cdot TI\)
  show \(?A\)
proof (intro ballI impI)
  fix t f assume t: "t \in timpl_closure_set FP TI" and f: "f \in funs_term t" "is_Abs f"
  hence "f \in \?F \vee f \in \?G" using timpl Closure_set_funs_term_subset[of FP TI] by auto
  thus "f \in Abs \cdot OCC"
  proof
    assume "f \in \?F" thus \(\text{?thesis}\) using OCC1 f(2) by fast
  next
    assume "f \in \?G" thus \(\text{?thesis}\) using OCC2 by auto
  qed
qed

\{
  fix s t a b
  assume t: "t \in absc \cdot OCC"
  and ab: "(a, b) \in TI"
  and s: "s \in set (a --> b)(t)"
  obtain c where c: "t = absc c" "c \in OCC" using t by moura
  hence "s \in absc b \vee s = absc c"
  using ab s timpl Closure_apply_const'[of c a b] unfolding absc_def by auto
  moreover have "b \in OCC" using ab OCC2 by auto
  ultimately have "s \in absc \cdot OCC" using c(2) by blast
\}
thus \(?B\) by blast
qed

lemma \(\text{(in stateful protocol model)}\) intruder_synth_timpl Closure_set:
fixes M::\(\text{('fun, 'atom, 'sets, 'lbl) prot terms}\) and t::\(\text{('fun, 'atom, 'sets, 'lbl) prot term}\)
assumes "M \vdash c t"
and "s \in timpl Closure t TI"
shows "timpl Closure_set M TI \vdash s"
using assms
proof (induction t arbitrary: s rule: intruder_synth_induct)
3.5 Term Implication

case (AxiomC t)

hence "s ∈ timpl_closure_set M TI"
  using timpl_closure_set_is_timpl_closure_union[of M TI]
  by blast

thus ?case by simp

next

case (ComposeC T f)

obtain g S where s: "s = Fun g S"
  using timpl_closure_Fun_inv[of T f TI] ComposeC.prems ComposeC.hyps(2)

hence s': "f = g" "length S = length T"
  "∀i. i < length S ⇒ S ! i ∈ timpl_closure (T ! i) TI"

have "timpl_closure_set M TI ⊢ u" when u: "u ∈ set S" for u
  using ComposeC.IH u s'(2,3) in_set_conv_nth[of _ T] in_set_conv_nth[of u S]
  by auto

thus ?case
  using s s'(1,2) ComposeC.hyps(1,2) intruder_synth.ComposeC[of S g "timpl_closure_set M TI"]
  by argo

qed

lemma (in stateful_protocol_model) intruder_synth_timpl_closure':
  fixes M::"('fun,'atom,'sets,'lbl) prot_terms" and t::"('fun,'atom,'sets,'lbl) prot_term"
  assumes "timpl_closure_set M TI ⊢ c t"
  and "s ∈ timpl_closure t TI"
  shows "timpl_closure_set M TI ⊢ c s"
  by (metis intruder_synth_timpl_closure_set[of assms] timpl_closure_set_idem)

lemma timpl_closure_set_absc_subset_in:
  defines "absc ≡ λ a. Fun (Abs a) []"
  assumes A: "timpl_closure_set (absc ` A) TI ⊆ absc ` A"
  and a: "a ∈ A" "(a,b) ∈ TI"+
  shows "b ∈ A"
proof -

  have "timpl_closure (absc a) (TI+) ⊆ absc ` A"
    using a(1) A timpl_closure_timpls_trancl_eq
    unfolding timpl_closure_set_def by fast

  thus ?thesis
    using timpl_closure.TI[of A "absc a" a(2), of "absc b"]
    unfolding absc_def by auto

qed

lemma timpl_closure_Abs_ex:
  assumes t: "s ∈ timpl_closure t TI"
  and a: "Abs a ∈ funs_term t"
  shows "∃t bs. (a,b) ∈ TI ∧ Fun (Abs b) ts ⊑ s"

proof (induction rule: timpl_closure.induct)
  case (TI u b c s)

  obtain d ts where d: "(a,d) ∈ TI"" "Fun (Abs d) ts ⊑ u" using TI.IH by blast
  note 0 = TI.hyps(2) d(1) term_variants_pred_inv'(5)[OF term_variants_pred_const]

  show ?case using TI.hyps(3) d(2)
    using d(2)
proof (induction rule: term_variants_pred.induct)
  case (term_variants_P T S g f)

  note hypeq = term_variants_P.hyps
  note prems = term_variants_P.prems
  note IH = term_variants_P.IH

  show ?case
    proof (cases "Fun (Abs d) ts = Fun f T")
      case False
      hence "∃t ∈ set T. Fun (Abs d) ts ⊑ t" using prems(1) by force

153
then obtain \( i \) where \( i: \text{"}\ i < \text{length } T \text{"} \) \( \text{Fun (Abs } d \text{) ts} \sqsubseteq T ! i \) by (metis \text{in_set_conv_nth})

qed (metis \text{hyps(3) 0 prot_fun.sel(4) r_into_rtrancl rtrancl_trans term.eq_refl term.sel(2)})

next

\begin{itemize}
\item \text{case } (\text{term_variants_Fun } T \ S \ f)
\item \text{note } hyps = \text{term_variants_Fun.hyps}
\item \text{note } prems = \text{term_variants_Fun.prems}
\item \text{note } IH = \text{term_variants_Fun.IH}
\end{itemize}

show \(?\text{case}\)

proof

(\text{cases } \text{"Fun (Abs } d \text{) ts} = \text{Fun } f \ T\"")

\begin{itemize}
\item \text{case } \text{False}
\end{itemize}

hence \("\exists t \in \text{set } T. \text{Fun (Abs } d \text{) ts} \sqsubseteq t\"")\ using \text{prems(1)} by force

then obtain \( i \) where \( i: \text{"}\ i < \text{length } T \text{"} \) \( \text{Fun (Abs } d \text{) ts} \sqsubseteq T ! i \) by (metis \text{in_set_conv_nth})

show \(?\text{thesis}\)

by (metis \text{IH[OF } i\text{] } i(1) \text{ hyps(1) nth_mem subtermeqI'' term.order.trans})

qed (metis \text{hyps(3) 0 prot_fun.sel(4) r_into_rtrancl rtrancl_trans term.eq_refl term.sel(2)})

qed simp

qed (meson \text{a funs_term_Fun_subterm rtrancl_eq_or_trancl})
3.5 Term Implication

using \( A[\text{OF } fg] \) s t t' \( (2,3) \) Ana_nonempty_inv[of s] Ana_nonempty_inv[of t]

unfolding fg s' t'(1) by fastforce

show \(?thesis\)

proof (intro allI impI)

fix i assume i: "i < length Ks"

have Ks: "Ks = Kh \cdot ! i ss" and Kt: "Kt = Kh \cdot ! i ts"

using h(6,7) s t by auto

have 0: "Kh ! i \in set Kh" using Ks i by simp

have 1: "\( \forall x \in \text{fv } (Kh ! i). x < length ss \)"

using 0 Ana_f_assm2_alt[OF h(5)]

unfolding h(1-3) by fastforce

have "term_variants_pred P (Kh ! i \cdot (!) ss) (Kh ! i \cdot (!) ts)"

using term_variants_pred_Ana_keys[OF t'(2) i t'(3)]

unfolding P_def by fast

thus "term_variants_pred P (Ks ! i) (Kt ! i)"

using i unfolding Ks Kt by simp

qed

ultimately show \(?A \?B\) by fast+

qed

ultimately show \(?A \?B\) by (cases s; simp_all)+

qed

lemma (in stateful_protocol_model) timpl_closure_Ana_keys:

fixes s t::"('fun,'atom,'sets,'lbl) prot_term"

assumes "t \in timpl_closure s TI"

and "Ana s = (Ks, Rs)"

and "Ana t = (Kt, Rt)"

shows "length Kt = length Ks" (is \(?A\))

and "\( \forall i < length Ks. Kt ! i \in timpl_closure (Ks ! i) TI \)" (is \(?B\))

using assms

proof (induction arbitrary: Ks Rs Kt Rt rule: timpl_closure.induct)

next

obtain Ku Ru where u: "Ana u = (Ku, Ru)" by (metis surj_pair)

have 0 = term_variants_pred_Ana_keys[OF TI.hyps(3) u]

{ case 1 thus case using 0(1) TI.IH(1) u by fastforce }

{ case 2 thus case using (metis 0(2)[OF 2(2)] TI.IH[OF 2(1) u] timpl_closure.TI[OF _ TI.hyps(2)]) } 

qed

lemma (in stateful_protocol_model) timpl_closure_Ana_keys_length_eq:

fixes s t::"('fun,'atom,'sets,'lbl) prot_term"

assumes "t \in timpl_closure s TI"

and "Ana s = (Ks, Rs)"

and "Ana t = (Kt, Rt)"

shows "length Kt = length Ks"

by (rule timpl_closure_Ana_keys(1,2)[OF assms])

lemma (in stateful_protocol_model) timpl_closure_Ana_keys_subset:

fixes s t::"('fun,'atom,'sets,'lbl) prot_term"

assumes "t \in timpl_closure s TI"

and "Ana s = (Ks, Rs)"

and "Ana t = (Kt, Rt)"

shows "set Kt \subseteq timpl_closure_set (set Ks) TI"

proof -

have "\( \forall i < length Ks. Kt ! i \in timpl_closure_set (set Ks) TI \)"


using in_set_conv_nth 0(2) unfolding timpl_closure_set_def by auto
thus "set Kt ⊆ timpl_closure_set (set Ks) TI"
using 0(1) by (metis subsetI in_set_conv_nth)
qed

3.5.3 Composition-only Intruder Deduction Modulo Term Implication Closure of the Intruder Knowledge

context stateful_protocol_model
begin

fun in_trancl where
"in_trancl TI a b = (if (a,b) ∈ set TI then True
else list_ex (λ(c,d). c = a ∧ in_trancl (removeAll (c,d) TI) d b) TI)"
definition in_rtrancl where
"in_rtrancl TI a b ≡ a = b ∨ in_trancl TI a b"
declare in_trancl.simps[simp del]

fun timpls_transformable_to where
"timpls_transformable_to TI (Var x) (Var y) = (x = y)"
| "timpls_transformable_to TI (Fun f T) (Fun g S) = (f = g ∨ (is_Abs f ∧ is_Abs g ∧ (the_Abs f, the_Abs g) ∈ set TI)) ∧ list_all2 (timpls_transformable_to TI) T S)"
| "timpls_transformable_to \_ \_ \_ = False"

fun timpls_transformable_to' where
"timpls_transformable_to' TI (Var x) (Var y) = (x = y)"
| "timpls_transformable_to' TI (Fun f T) (Fun g S) = (f = g ∨ (is_Abs f ∧ is_Abs g ∧ in_trancl TI (the_Abs f) (the_Abs g))) ∧ list_all2 (timpls_transformable_to' TI) T S)"
| "timpls_transformable_to' \_ \_ \_ = False"

fun equal_mod_timpls where
"equal_mod_timpls TI (Var x) (Var y) = (x = y)"
| "equal_mod_timpls TI (Fun f T) (Fun g S) = (f = g ∨ (is_Abs f ∧ is_Abs g ∧ (the_Abs f, the_Abs g) ∈ set TI) ∨ (the_Abs g, the_Abs f) ∈ set TI ∨ (∃ti ∈ set TI. (the_Abs f, snd ti) ∈ set TI ∧ (the_Abs g, snd ti) ∈ set TI))) ∧ list_all2 (equal_mod_timpls TI) T S)"
| "equal_mod_timpls \_ \_ \_ = False"

fun intruder_synth_mod_timpls where
"intruder_synth_mod_timpls M TI (Var x) = List.member M (Var x)"
| "intruder_synth_mod_timpls M TI (Fun f T) = (list_ex (λt. timpls_transformable_to TI t (Fun f T)) M) ∨ (public f ∧ length T = arity f ∧ list_all (intruder_synth_mod_timpls M TI) T)"

fun intruder_synth_mod_timpls' where
"intruder_synth_mod_timpls' M TI (Var x) = List.member M (Var x)"
| "intruder_synth_mod_timpls' M TI (Fun f T) = (list_ex (λt. timpls_transformable_to' TI t (Fun f T)) M) ∨ (public f ∧ length T = arity f ∧ list_all (intruder_synth_mod_timpls' M TI) T)"

fun intruder_synth_mod_eq_timpls where
"intruder_synth_mod_eq_timpls M TI (Var x) = (Var x ∈ M)"
| "intruder_synth_mod_eq_timpls M TI (Fun f T) = (∃t ∈ M. equal_mod_timpls TI t (Fun f T)) ∨ (public f ∧ length T = arity f ∧ list_all (intruder_synth_mod_eq_timpls M TI) T)"

156
3.5 Term Implication

definition \text{analyzed} \_ \text{closed} \_ \text{mod} \_ \text{timpls} \text{ where}
\text{"analyzed} \_ \text{closed} \_ \text{mod} \_ \text{timpls} M TI \equiv \text{let} ti = \text{intruder} \_ \text{synth} \_ \text{mod} \_ \text{timpls} M TI; \text{ cl } = \lambda ts. \text{ comp} \_ \text{timpl} \_ \text{closure} ts \text{ (set} TI); f = \text{ list} \_ \text{all} ti; g = \lambda t. \text{ if} f \text{ (fst} (\text{Ana} t)) \text{ then} f \text{ (snd} (\text{Ana} t)) \text{ else list} \_ \text{all} (\text{ At. } \forall f \in \text{ funs} \_ \text{term} t. \text{ ~is} \_ \text{Abs} f) \text{ (fst} (\text{Ana} t)) \text{ then True else } \forall s \in \text{ cl} \text{ (set} (\text{fst} (\text{Ana} t))) \cdot ~ti s \text{ then True else } \forall s \in \text{ comp} \_ \text{timpl} \_ \text{closure} \{t\} \text{ (set} TI) \cdot \text{ case} \text{ Ana} s \text{ of} \langle K,R \rangle \Rightarrow f K \rightarrow f R \text{ in list} \_ \text{all} g M"

definition \text{analyzed} \_ \text{closed}\_ \text{mod} \_ \text{timpls'} \text{ where}
\text{"analyzed} \_ \text{closed} \_ \text{mod} \_ \text{timpls'} M TI \equiv \text{let} f = \text{ list} \_ \text{all} (\text{intruder} \_ \text{synth} \_ \text{mod} \_ \text{timpls'} M TI); g = \lambda t. \text{ if} f \text{ (fst} (\text{Ana} t)) \text{ then} f \text{ (snd} (\text{Ana} t)) \text{ else list} \_ \text{all} (\lambda t. \forall f \in \text{ funs} \_ \text{term} t. \text{ ~is} \_ \text{Abs} f) \text{ (fst} (\text{Ana} t)) \text{ then True else } \forall s \in \text{ comp} \_ \text{timpl} \_ \text{closure} \{t\} \text{ (set} TI) \cdot \text{ case} \text{ Ana} s \text{ of} \langle K,R \rangle \Rightarrow f K \rightarrow f R \text{ in list} \_ \text{all} g M"

lemma term_variants\_pred\_Abs\_Ana\_keys:
fixes a b
defines \text{"P } \equiv \text{ (}(\lambda_. [])\text{(Abs} a := [\text{Abs} b]))"
assumes st: \text{"term} \_ \text{variants} \_ \text{pred} P s t"
shows \text{"length} \text{ (fst} (\text{Ana} s)) = \text{length} \text{ (fst} (\text{Ana} t))" \text{ (is } \text{"?P s t")}
and \text{"}\forall i < \text{length} \text{ (fst} (\text{Ana} s)). \text{ term} \_ \text{variants} \_ \text{pred} P \text{ (fst} (\text{Ana} s) ! i) \text{ (fst} (\text{Ana} t) ! i)" \text{ (is } \text{"?Q s t")}
proof -
show \"?P s t" using st
proof (induction s t rule: term_variants\_pred.induct)
case (term_variants_Fun T S f) show \case
proof (cases f)
case (Fu g) thus \?thesis using term_variants_Fun Ana_Fu\_keys\_length\_eq by blast
qed simp_all
qed (simp_all add: P_def)

show \"?Q s t" using st
proof (induction s t rule: term_variants\_pred.induct)
case (term_variants_Fun T S f)
note hyps = term_variants_Fun.hyps
let \?K = \"\lambda U. \text{ fst} (\text{Ana} (\text{Fun} f U))\"

show \?case
proof (cases f)
case (Fu g) show \?thesis
proof (cases \"arity_f, g = length T \land arity_f, g > 0\")
case True
hence \*: \"?K T = \text{ fst} (\text{Ana}_f g) \cdot \text{list (!) T}" 
\"?K S = \text{ fst} (\text{Ana}_f g) \cdot \text{list (!) S}" using Fu Ana_Fu\_intro fst\_conv prod\_collapse
by (metis (mono_tags, lifting), metis (mono_tags, lifting) hyps(1))

have K: \"j < \text{length} T\" when j: \"j \in \text{fv}_{\text{set}} \text{ (set} \text{ (fst} (\text{Ana}_f g)))\" for j
using True Ana_f\_assm2\_alt[of g \"\text{fst} (\text{Ana}_f g)" _ j ]
by (metis UnI1 prod\_collapse that)

show \?thesis
proof (intro allI impI)
fix i assume i: \"i < \text{length} \text{ (?K T)}\"
let \?u = \"\text{fst} (\text{Ana}_f g) ! i\"

have **: \"?K T ! i = \?u \cdot (!) T" 
\"?K S ! i = \?u \cdot (!) S"
using * i by simp_all

157
have ***: "x < length T" when "x ∈ fv (fst (Ana_f g) ! i)" for x
using that K Ana_f_assm2_alt[of g "fst (Ana_f g)" _ x] i hyps(1)
unfolding * by force

show "term_variants_pred P (?K T ! i) (?K S ! i)"
using i hyps K *** term_variants_pred_subst"[of ?u P "(!) T" "(!) S"]
unfolding * by auto
qed

lemma term_variants_pred_Abs_eq_case:
assumes t: "term_variants_pred (λ_. [])(Abs a := [Abs b]) s t" (is "?R s t")
and s: "∀f ∈ funs_term s. ¬is_Abs f" (is "?P s")
shows "s = t"
using s term_variants_pred_eq_case[OF t]
by fastforce

lemma term_variants_Ana_keys_no_Abs_eq_case:
fixes s t::"(('fun,'atom,'sets,'lbl) prot_fun,'v) term"
assumes t: "term_variants_pred (λ_. [])(Abs a := [Abs b]) s t" (is "?R s t")
and s: "∀t ∈ set (fst (Ana s)). ∀f ∈ funs_term t. ¬is_Abs f" (is "?P s")
shows "fst (Ana t) = fst (Ana s)" (is "?Q t s")
using s term_variants_pred_Abs_Ana_keys[OF t] term_variants_pred_Abs_eq_case[of a b]
by (metis nth_equalityI nth_mem)

lemma timpl_closure_Ana_keys_no_Abs_eq_case:
assumes t: "t ∈ timpl_closure s TI"
and s: "∀t ∈ set (fst (Ana s)). ∀f ∈ funs_term t. ¬is_Abs f" (is "?P s")
shows "fst (Ana t) = fst (Ana s)"
using t
proof (induction t rule: timpl_closure.induct)
case (TI u a b t)
thus ?case
using s term_variants_Ana_keys_no_Abs_eq_case by fastforce
qed simp

lemma in_trancl_closure_iff_in_trancl_fun:
"(a,b) ∈ (set TI)^+ ←→ in_trancl TI a b" (is "?A TI a b ←→ ?B TI a b")
proof
show "?A TI a b ⇒ ?B TI a b"
proof (induction rule: trancl_induct)
case (step c d)
show ?case using step.IH step.hyps(2)
proof (induction TI a c rule: in_trancl.induct)
case (1 TI a b) thus ?case using in_trancl.simps
by (smt Bex_set case_prodE case_prodI member_remove prod.sel(2) remove_code(1))
qed

show "?B TI a b ⇒ ?A TI a b"
proof (induction TI a b rule: in_trancl.induct)
case (1 TI a b)
let ?P = "∀TI a b c d. in_trancl (?set TI) (c,d) TI d b"
have *: "∃(c,d) ∈ set TI. c = a ∧ ?P TI a b c d" when "(a,b) ∉ set TI"
using that "1.prems" list_ex_iff[of _ TI] in_trancl.simps[of TI a b]
by auto
show ?case
proof (cases "(a,b) ∈ set TI")
case False
hence "∃(c,d) ∈ set TI. c = a ∧ ?P TI a b c d" using * by blast
then obtain d where d: "(a,d) ∈ set TI" "?P TI a b a d" by blast
have "(d,b) ∈ (set (removeAll (a,d) TI))^+" using "1.IH"[OF False d(1)] d(2) by blast
moreover have \(\text{"set (removeAll (a,d) TI) \(\subseteq\) set TI" by simp}\)
ultimately have \(\text{"(d,b) \(\in\) (set TI)" using trancl_mono by blast}\)
thus \(?\)thesis using \(d(1)\) by fastforce
qed simp
qed

lemma in_rtrancl_closure_iff_in_rtrancl_fun:
\("(a,b) \in (set TI)' \iff in_rtrancl TI a b"
by (metis rtrancl_eq_or_trancl in_trancl_closure_iff_in_trancl_fun in_rtrancl_def)

lemma in_trancl_mono:
assumes \("set TI \(\subseteq\) set TI'\)
and \("in_trancl TI a b\"
shows \("in_trancl TI' a b\"
by (metis assms in_trancl_closure_iff_in_trancl_fun trancl_mono)

lemma equal_mod_timpls_refl:
\("equal_mod_timpls TI t t"\)
proof (induction t)
  case (Fun f T)
  thus \(?\)case
    using list_all2_conv_all_nth[of \"equal_mod_timpls TI" T T]
    by force
qed simp

lemma equal_mod_timpls_inv_V:
\("equal_mod_timpls TI (Var x) t \Longrightarrow t = Var x" (is \"?A \Longrightarrow ?C\")
"equal_mod_timpls TI t (Var x) \Longrightarrow t = Var x" (is \"?B \Longrightarrow ?C\")
proof -
  show \"?A \Longrightarrow ?C" by (cases t) auto
  show \"?B \Longrightarrow ?C" by (cases t) auto
qed

lemma equal_mod_timpls_inv:
assumes \("equal_mod_timpls TI (Fun f T) (Fun g S)\"
shows \"length T = length S"
  and \("\forall i. i < length T \Longrightarrow equal_mod_timpls TI (T ! i) (S ! i)"\)
  and \("f \neq g \Longrightarrow (is_Abs f \land is_Abs g \land (\begin{array}{l}
      (the_Abs f, the_Abs g) \in set TI \lor (the_Abs g, the_Abs f) \in set TI \lor
      (\exists ti \in set TI. (the_Abs f, snd ti) \in set TI \land
      (the_Abs g, snd ti) \in set TI)))\end{array})"\"
using assms list_all2_conv_all_nth[of \"equal_mod_timpls TI" T S]
by (auto elim: equal_mod_timpls.cases)

lemma equal_mod_timpls_inv':
assumes \("equal_mod_timpls TI (Fun f T) t"\)
shows \"is_Fun t"
and \"length T = length (args t)"
  and \("\forall i. i < length T \Longrightarrow equal_mod_timpls TI (T ! i) (args t ! i)"\)
  and \("f \neq \text{the}_\text{Fun} t \Longrightarrow (is_Abs f \land is_Abs (\text{the}_\text{Fun} t) \land (\begin{array}{l}
      (the_Abs f, the_Abs (\text{the}_\text{Fun} t)) \in set TI \lor
      (the_Abs (\text{the}_\text{Fun} t), the_Abs f) \in set TI \lor
      (\exists ti \in set TI. (the_Abs f, snd ti) \in set TI \land
      (the_Abs (\text{the}_\text{Fun} t), snd ti) \in set TI)))\end{array})"\"
and \("\neg is_Abs f \Longrightarrow f = \text{the}_\text{Fun} t"\)
using assms list_all2_conv_all_nth[of \"equal_mod_timpls TI" T]
by (cases t; auto)+

lemma equal_mod_timpls_if_term_variants:
fixes s t::\("('a, 'b, 'c, 'd) prot_fun, 'e) term" and a b::\"'c set"
defines \("P \equiv (\lambda_. [])\)(\text{Abs} a := [\text{Abs} b])\"
assumes st: \"term_variants_pred P s t" and ab: \"(a,b) \in set TI" and c: \"c set"}
shows \"equal_mod_timpls TI s t"
using st P_def
proof (induction rule: term_variants_pred.induct)
case (term_variants_P T S f) thus ?case
  using ab list_all2_conv_all_nth[of "equal_mod_timpls TI" T S]
  in_trancl_closure_iff_in_trancl_fun[of _ _ TI]
  by auto
next
case (term_variants_Fun T S f) thus ?case
  using ab list_all2_conv_all_nth[of "equal_mod_timpls TI" T S]
  in_trancl_closure_iff_in_trancl_fun[of _ _ TI]
  by auto
qed simp

lemma equal_mod_timpls_mono:
  assumes "set TI ⊆ set TI'"
  and "equal_mod_timpls TI s t"
  shows "equal_mod_timpls TI' s t"
  using assms
  proof (induction TI s t rule: equal_mod_timpls.induct)
    case (2 TI f T g S)
    have *: "f = g ∨ (is_Abs f ∧ is_Abs g ∧ ((the_Abs f, the_Abs g) ∈ set TI ∨
      (the_Abs g, the_Abs f) ∈ set TI ∨
      (∃ti ∈ set TI. (the_Abs f, snd ti) ∈ set TI ∧
      (the_Abs g, snd ti) ∈ set TI)))"
      "list_all2 (equal_mod_timpls TI) T S"
      using "2.prems" by simp_all
    show ?case
      using "2.IH" "2.prems"(1) list.rel_mono_strong[OF *(2)] *(1) in_trancl_mono[of TI TI']
      by (metis (no_types, lifting) equal_mod_timpls.simps(2) set_rev_mp)
  qed auto

lemma equal_mod_timpls_refl_minus_eq:
  "equal_mod_timpls TI s t ↔ equal_mod_timpls (filter (λ(a,b). a ≠ b) TI) s t"
  (is "?A ↔ ?B")
  proof
    show ?A when ?B using that equal_mod_timpls_mono[of "filter (λ(a,b). a ≠ b) TI" TI] by auto
    show ?B when ?A using that
    proof (induction TI s t rule: equal_mod_timpls.induct)
      case (2 TI f T g S)
      define TI' where "TI' ≡ filter (λ(a,b). a ≠ b) TI"
      let ?P = "λX Y. f = g ∨ (is_Abs f ∧ is_Abs g ∧ ((the_Abs f, the_Abs g) ∈ set X ∨
        (the_Abs g, the_Abs f) ∈ set X ∨
        (∃ti ∈ set Y. (the_Abs f, snd ti) ∈ set X ∧
        (the_Abs g, snd ti) ∈ set X)))"
      have *: "?P TI TI' "list_all2 (equal_mod_timpls TI) T S"
        using "2.prems" by simp_all
      have "?P TI' TI" using *(1) unfolding TI'_def is_Abs_def by auto
      hence "?P TI' TI'" by (metis (no_types, lifting) snd_conv)
      moreover have "list_all2 (equal_mod_timpls TI') T S"
        using *(2) "2.IH" list.rel_mono_strong unfolding TI'_def by blast
      ultimately show ?case unfolding TI'_def by force
    qed auto
  qed

lemma timpls_transformable_to_refl:
  "timpls_transformable_to TI t t" (is ?A)
  "timpls_transformable_to' TI t t" (is ?B)
by (induct t) (auto simp add: list_all2_conv_all_nth)

lemma timpls_transformable_to_inv_Var:
  "timpls_transformable_to TI (Var x) t ⟹ t = Var x" (is "?A ☑ ?C")
  "timpls_transformable_to TI t (Var x) ⟹ t = Var x" (is "?B ☑ ?C")
  "timpls_transformable_to' TI (Var x) t ⟹ t = Var x" (is "?A' ☑ ?C")
  "timpls_transformable_to' TI t (Var x) ⟹ t = Var x" (is "?B' ☑ ?C")
by (cases t; auto)+

lemma timpls_transformable_to_inv:
  assumes "timpls_transformable_to TI (Fun f T) (Fun g S)"
  shows "length T = length S"
    "⋀ i. i < length T ⟹ timpls_transformable_to TI (T ! i) (S ! i)"
  and "f ≠ g ⟹ (is_Abs f ∧ is_Abs g ∧ (the_Abs f, the_Abs g) ∈ set TI)"
using assms list_all2_conv_all_nth[of "timpls_transformable_to TI" T S] by auto

lemma timpls_transformable_to'_inv:
  assumes "timpls_transformable_to' TI (Fun f T) (Fun g S)"
  shows "length T = length S"
    "⋀ i. i < length T ⟹ timpls_transformable_to' TI (T ! i) (S ! i)"
  and "f ≠ g ⟹ (is_Abs f ∧ is_Abs g ∧ in_trancl TI (the_Abs f) (the_Abs g))"
using assms list_all2_conv_all_nth[of "timpls_transformable_to' TI" T S] by auto

lemma timpls_transformable_to_inv':
  assumes "timpls_transformable_to TI (Fun f T) t"
  shows "is_Fun t"
    "length T = length (args t)"
    "⋀ i. i < length T ⟹ timpls_transformable_to TI (T ! i) (args t ! i)"
  and "f ≠ the_Fun t ⟹ (is_Abs f ∧ (the_Abs f, the_Abs (the_Fun t)) ∈ set TI)"
  and "¬ is_Abs f ⟹ f = the_Fun t"
using assms list_all2_conv_all_nth[of "timpls_transformable_to TI" T] by (cases t; auto)+

lemma timpls_transformable_to'_inv':
  assumes "timpls_transformable_to' TI (Fun f T) t"
  shows "is_Fun t"
    "length T = length (args t)"
    "⋀ i. i < length T ⟹ timpls_transformable_to' TI (T ! i) (args t ! i)"
  and "f ≠ the_Fun t ⟹ (is_Abs f ∧ in_trancl TI (the_Abs f) (the_Abs (the_Fun t)))"
  and "¬ is_Abs f ⟹ f = the_Fun t"
using assms list_all2_conv_all_nth[of "timpls_transformable_to' TI" T] by (cases t; auto)+

lemma timpls_transformable_to_size_eq:
  shows "size_list size T = size_list size S" (is "size_list size T ☑ size_list size S")
  shows "timpls_transformable_to TI s t ⟷ size s = size t" (is "size_list size T ☑ size_list size S")
proof -
  have "size_list size T = size_list size S" (is "size_list size T ☑ size_list size S")
    "size_list size T = size_list size S" (is "size_list size T ☑ size_list size S")
    "length T = length S" (is "length T ☑ length S")
    "⋀ i. i < length T ⟹ size (T ! i) = size (S ! i)"
    "⋀ i. i < length T ⟹ size (T ! i) = size (S ! i)"
  using that
proof (induction T arbitrary: S)
  case (Cons x T)
  then obtain y S' where "S = y#S'" by (cases S) auto
  hence "size_list size T = size_list size S'" (is "size_list size T ☑ size_list size S'")
  using Cons.prems Cons.IH[of S'] by force+
  thus ?case using y by simp
qed simp

show ?C when ?A using that
proof (induction rule: timpls_transformable_to.induct)
  case (2 TI f T g S)
  hence "length T = length S" "\forall i. i < length T \implies size (T ! i) = size (S ! i)"
  using timpls_transformable_to_inv(1,2)[of TI f T g S] by auto
  thus ?case using *[of S T] by simp
qed simp_all

show ?C when ?B using that
proof (induction rule: timpls_transformable_to.induct)
  case (2 TI f T g S)
  hence "length T = length S" "\forall i. i < length T \implies size (T ! i) = size (S ! i)"
  using timpls_transformable_to'_inv(1,2)[of TI f T g S] by auto
  thus ?case using *[of S T] by simp
qed simp_all

lemma timpls_transformable_to_if_term_variants:
  fixes s t::"(('a, 'b, 'c, 'd) prot_fun, 'e) term" and a b::"'c set"
  defines P_eq \equiv \((\lambda _. []). (Abs a := [Abs b])\"
  assumes st: "term_variants_pred P s t"
  and ab: "(a,b) \in set TI"
  shows "timpls_transformable_to TI s t"
  using st P_def
  proof (induction rule: term_variants_pred.induct)
    case (term_variants_P T S f)
    thus ?case
      using ab list_all2_conv_all_nth[of "timpls_transformable_to TI" T S]
      by auto
    next
    case (term_variants_Fun T S f)
    thus ?case
      using ab list_all2_conv_all_nth[of "timpls_transformable_to TI" T S]
      by auto
  qed

lemma timpls_transformable_to'_if_term_variants:
  fixes s t::"(('a, 'b, 'c, 'd) prot_fun, 'e) term" and a b::"'c set"
  defines P_eq \equiv \((\lambda _. []). (Abs a := [Abs b])\"
  assumes st: "term_variants_pred P s t"
  and ab: "(a,b) \in (set TI)\"
  shows "timpls_transformable_to' TI s t"
  using st P_def
  proof (induction rule: term_variants_pred.induct)
    case (term_variants_P T S f)
    thus ?case
      using ab list_all2_conv_all_nth[of "timpls_transformable_to' TI" T S]
      in_trancl_closure_iff_in_trancl_fun[of _ _ TI]
      by auto
    next
    case (term_variants_Fun T S f)
    thus ?case
      using ab list_all2_conv_all_nth[of "timpls_transformable_to' TI" T S]
      in_trancl_closure_iff_in_trancl_fun[of _ _ TI]
      by auto
  qed simp

lemma timpls_transformable_to_trans:
  assumes TI_trancl: "\forall (a,b) \in (set TI). a \neq b \longrightarrow (a,b) \in set TI"
  and st: "timpls_transformable_to TI s t"
  and tu: "timpls_transformable_to TI t u"
  shows "timpls_transformable_to TI s u"
  using st tu
  proof (induction s arbitrary: t u)
    case (Var x)
    thus ?case using tu timpls_transformable_to_inv_Var(1) by fast
  next
    case (Fun f T)
    obtain g S where t:
3.5 Term Implication

"t = Fun g S" "length T = length S"
"\[A_i. \ i < \text{length } T \implies \text{timpls_transformable_to } TI (T ! i) (S ! i)\]"
"f \neq g \implies is_Abs f \land is_Abs g \land (\text{the_Abs } f, \text{the_Abs } g) \in \text{set } TI"
using timpls_transformable_to_inv[OF Fun.prems(1)] TI_trancl by moura

obtain h U where u: "u = Fun h U" "length S = length U"
"\[A_i. \ i < \text{length } S \implies \text{timpls_transformable_to } TI (S ! i) (U ! i)\]"
"g \neq h \implies is_Abs g \land \text{the_Abs } g \land (\text{the_Abs } h) \in \text{set } TI"
using timpls_transformable_to_inv'[OF Fun.prems(2)[unfolded t(1)]] TI_trancl by moura

have "list_all2 (timpls_transformable_to TI) T U"
using t(1,2,3) u(1,2,3) Fun.IH
list_all2_conv_all_nth[of "timpls_transformable_to TI" T S]
list_all2_conv_all_nth[of "timpls_transformable_to TI" S U]
list_all2_conv_all_nth[of "timpls_transformable_to TI" T U]
by force

moreover have "(\text{the_Abs } f, \text{the_Abs } h) \in \text{set } TI"
when "(\text{the_Abs } f, \text{the_Abs } g) \in \text{set } TI" "(\text{the_Abs } g, \text{the_Abs } h) \in \text{set } TI"
using that(3,4,5) TI_trancl trancl_into_trancl[OF r_into_trancl[OF that(1)] that(2)]
unfolding is_Abs_def the_Abs_def by force

hence "is_Abs f \land is_Abs h \land (\text{the_Abs } f, \text{the_Abs } h) \in \text{set } TI"
when "f \neq h"
using that TI_trancl t(4) u(4) by fast

ultimately show ?case using t(1) u(1) by force

qed

lemma timpls_transformable_to'_trans:
assumes st: "timpls_transformable_to' TI s t"
and tu: "timpls_transformable_to' TI t u"
shows "timpls_transformable_to' TI s u"
using st tu
proof (induction s arbitrary: t u)
  case (Var x) thus ?case using tu timpls_transformable_to_inv_Var(3) by fast
next
case (Fun f T)
  note 0 = in_trancl_closure_iff_in_trancl_fun[of _ _ TI]

obtain g S where t:
  "t = Fun g S" "length T = length S"
"\[A_i. \ i < \text{length } T \implies \text{timpls_transformable_to' } TI (T ! i) (S ! i)\]"
"f \neq g \implies is_Abs f \land is_Abs g \land (\text{the_Abs } f, \text{the_Abs } g) \in (\text{set } TI)^+""
using timpls_transformable_to'_inv'[OF Fun.prems(1)] 0 by moura

obtain h U where u: "u = Fun h U" "length S = length U"
"\[A_i. \ i < \text{length } S \implies \text{timpls_transformable_to' } TI (S ! i) (U ! i)\]"
"g \neq h \implies is_Abs g \land \text{the_Abs } g \land (\text{the_Abs } h) \in (\text{set } TI)^+""
using timpls_transformable_to'_inv'[OF Fun.prems(2)[unfolded t(1)]] 0 by moura

have "list_all2 (timpls_transformable_to' TI) T U"
using t(1,2,3) u(1,2,3) Fun.IH
list_all2_conv_all_nth[of "timpls_transformable_to' TI" T S]
list_all2_conv_all_nth[of "timpls_transformable_to' TI" S U]
list_all2_conv_all_nth[of "timpls_transformable_to' TI" T U]
by force

moreover have "(\text{the_Abs } f, \text{the_Abs } h) \in (\text{set } TI)^+"
when "(\text{the_Abs } f, \text{the_Abs } g) \in (\text{set } TI)^+" "(\text{the_Abs } g, \text{the_Abs } h) \in (\text{set } TI)^+"
using that by simp

hence "is_Abs f \land is_Abs h \land (\text{the_Abs } f, \text{the_Abs } h) \in (\text{set } TI)^+"
when "f \neq h"
by (metis that t(4) u(4))
ultimately show ?case using t(1) u(1) 0 by force
qed

lemma timpls_transformable_to_mono:
  assumes "set TI ⊆ set TI'"
  and "timpls_transformable_to TI s t"
  shows "timpls_transformable_to TI' s t"
using assms
proof (induction TI s t rule: timpls_transformable_to.induct)
case (2 TI f T g S)
  have *: "f = g ∨ (is_Abs f ∧ is_Abs g ∧ (the_Abs f, the_Abs g) ∈ set TI)"
  "list_all2 (timpls_transformable_to TI) T S"
  using "2.prems" by simp_all
  show ?case
  using "2.IH" "2.prems"(1) list.rel_mono_strong[OF *(2)] *(1) in_trancl_mono[of TI TI']
  by (metis (no_types, lifting) timpls_transformable_to.simps(2) set_rev_mp)
qed auto

lemma timpls_transformable_to'_mono:
  assumes "set TI ⊆ set TI'"
  and "timpls_transformable_to' TI s t"
  shows "timpls_transformable_to' TI' s t"
using assms
proof (induction TI s t rule: timpls_transformable_to'.induct)
case (2 TI f T g S)
  have *: "f = g ∨ (is_Abs f ∧ is_Abs g ∧ in_trancl TI (the_Abs f) (the_Abs g))"
  "list_all2 (timpls_transformable_to' TI) T S"
  using "2.prems" by simp_all
  show ?case
  using "2.IH" "2.prems"(1) list.rel_mono_strong[OF *(2)] *(1) in_trancl_mono[of TI TI']
  by (metis (no_types, lifting) timpls_transformable_to'.simps(2))
qed auto

lemma timpls_transformable_to_refl_minus_eq:
  "timpls_transformable_to TI s t ←→ timpls_transformable_to (filter (λ (a,b). a ≠ b) TI) s t"
  (is "?A s t ←→ ?B")
proof
  let ?TI' = "λTI. filter (λ (a,b). a ≠ b) TI"
show ?A when ?B using that timpls_transformable_toMono[of "?TI'" TI] by auto
show ?B when ?A using that
proof (induction TI s t rule: timpls_transformable_to.induct)
case (2 TI f T g S)
  have *: "f = g ∨ (is_Abs f ∧ is_Abs g ∧ (the_Abs f, the_Abs g) ∈ set TI)"
  "list_all2 (timpls_transformable_to TI) T S"
  using "2.prems" by simp_all
  have *: "f = g ∨ (is_Abs f ∧ is_Abs g ∧ (the_Abs f, the_Abs g) ∈ set (?TI' TI))"
  using *(1) unfolding is_Abs_def by auto
  moreover have "list_all2 (timpls_transformable_to (?TI' TI)) T S"
  using *(2) "2.IH" list.rel_mono_strong by blast
  ultimately show ?case by force
qed auto

lemma timpls_transformable_to_iff_in_timpl_closure:
  assumes "set TI' = {(a,b) ∈ (set TI)++. a ≠ b)"
  shows "timpls_transformable_to TI' s t ←→ t ∈ timpls_closure s (set TI)" (is "?A s t ←→ ?B s t")
proof
show  "?A s t =⇒ ?B s t"  using  assms
 proof (induction s t rule: timpls_transformable_to.induct)
  case (2 TI f T g S)
  note prems = "2.prems"
  note IH = "2.IH"
  have 1: "length T = length S"  "∀i<length T. timpls_transformable_to TI' (T ! i) (S ! i)"
    using  prems(1)  list_all2_conv_all_nth[of "timpls_transformable_to TI'" T S]  by  simp_all
  note 2 = timpl_closure_is_timpl_closure'
  note 3 = in_set_conv_nth[of _ T]  in_set_conv_nth[of _ S]
  have 4: "timpl_closure' (set TI') = timpl_closure' (set TI)"
    using  timpl_closure'_timpls_trancl_eq'[of "set TI"]  prems(2)  by  simp
  have IH': "((T ! i, S ! i) ∈ timpl_closure' (set TI'))"  when  i: "i < length S"  for  i
    proof
      have  "timpls_transformable_to TI' (T ! i) (S ! i)"
        using  i 1  by  presburger
      hence  "S ! i ∈ timpl_closure (T ! i) (set TI)"
        using  IH[of "T ! i" "S ! i"]  i 1(1)  prems(2)  by  force
      thus  ?thesis  using  2[of "S ! i" "T ! i" "set TI"]  4  by  blast
    qed
  have 5: "f = g ∨ (∃a b. (a, b) ∈ (set TI')⁺ ∧ f = Abs a ∧ g = Abs b)"
    using  prems(1)  the_Abs_def[of f]  the_Abs_def[of g]  is_Abs_def[of f]  is_Abs_def[of g]
    by  fastforce
  show  ?case  using  2 4  timpl_closure_FunI[OF IH' 1(1) 5]  1(1)  by  auto
 qed (simp_all add: timpl_closure.FP)
 show  "?B s t =⇒ ?A s t"
 proof (induction t rule: timpl_closure.induct)
  case (TI u a b v)
  show  ?case
    proof (cases "a = b")
      case True
      thus  ?thesis  using  TI.hyps(3)  TI.IH  term_variants_pred_refl_inv
        by  fastforce
    next
      case False
      hence  1: "timpls_transformable_to TI' u v"  by  blast
      using  TI.hyps(2)  assms  timpls_transformable_to_if_term_variants[OF TI.hyps(3), of TI']
        by  blast
      have 2: "((c,d) ∈ set TI'"  when  cd: "(c,d) ∈ (set TI')⁺"  "c ≠ d"  for  c d
        proof
          let ?cla = "¬(∀X. (a,b) ∈ X⁺. a ≠ b)"
          have  "(?cla (set TI') = ?cla (?cla (set TI)))"  using  assms  by  presburger
          hence  "set TI' = ?cla (set TI')"  using  assms  trancl_minus_refl_idem[of "set TI"]  by  argo
          thus  ?thesis  using  cd  by  blast
        qed
      show  ?thesis  using  timpls_transformable_to_trans[OF _ TI.IH 1]  2  by  blast
      qed
      qed
      (use  timpls_transformable_to_refl in  fast)
      qed

lemma  timpls_transformable_to'_iff_in_timpl_closure:
  "timpls_transformable_to' TI s t =⇒ t ∈ timpl_closure s (set TI)"  (is  "=?A s t =⇒ ?B s t")
 proof
  show  "?A s t =⇒ ?B s t"
 proof (induction s t rule: timpls_transformable_to'.induct)
   case (2 TI f T g S)
   note prems = "2.prems"
   note IH = "2.IH"
   have 1: "length T = length S"  "∀i<length T. timpls_transformable_to TI' (T ! i) (S ! i)"
     using  prems  list_all2_conv_all_nth[of "timpls_transformable_to TI'" T S]  by  simp_all
  qed
3 Stateful Protocol Verification

note 2 = timpl_closure_is_timpl_closure'

have IH': "(T ! i, S ! i) ∈ timpl_closure' (set TI)" when i: "i < length S" for i
proof -
  have "timpls_transformable_to' TI (T ! i) (S ! i)" using i 1 by presburger
  hence "S ! i ∈ timpl Closure (T ! i) (set TI)" using IH[of "T ! i" "S ! i"] i 1(1) by force
  thus ?thesis using 2[of "S ! i" "T ! i" "set TI"] by blast
qed

have 4: "f = g ∨ (∃ a b. (a, b) ∈ (set TI)+ ∧ f = Abs a ∧ g = Abs b)"
  using prems the_Abs_def[of f] the_Abs_def[of g] is_Abs_def[of f] is_Abs_def[of g]
  in trancl_closure_iff_in_trancl_fun[of _ TI]
  by auto

show ?case using 2 timpl_closure_FunI[OF IH' 1(1) 4] 1(1) by auto
qed (simp_all add: timpl_closure.FP)

show "?B s t =⇒ ?A s t" proof (induction t rule: timpl_closure.induct)
  case (TI u a b v)
  thus ?case using timpls_transformable_to'_trans
    timpls_transformable_to'_if_term_variants
    by blast
qed (use timpls_transformable_to_refl(2) in fast)

lemma equal_mod_timpls_iff_ex_in_timpl_closure:
  assumes "set TI' = {(a,b) ∈ TI+ | a ≠ b}"
  shows "equal_mod_timpls TI' s t ←→ (∃ u. u ∈ timpl_closure s TI ∧ u ∈ timpl_closure t TI)"
(proof)
  using prems list_all2_conv_all_nth[of equal_mod_timpls TI' T S]
  by simp_all

have 1: "length T = length S" "∀ i<length T. equal_mod_timpls (TI') (T ! i) (S ! i)"
  using prems list_all2_conv_all_nth[of equal_mod_timpls TI' T S] by simp_all

note 2 = timpl_closure_is_timpl_closure'

have IH: "∃ u. (T ! i, u) ∈ timpl_closure' TI ∧ (S ! i, u) ∈ timpl_closure' TI" when i: "i < length S" for i
proof -
  have "equal_mod_timpls TI' (T ! i) (S ! i)" using i 1 by presburger
  hence "∃ u. u ∈ timpl Closure (T ! i) TI ∧ u ∈ timpl Closure (S ! i) TI" using IH[of "T ! i" "S ! i"] i 1(1) prems by force
  thus ?thesis using 4 unfolding 2 by blast
qed

let ?P = "λG. f = g ∨ (∃ a b. (a, b) ∈ G ∧ f = Abs a ∧ g = Abs b) ∨
             (∃ a b. (a, b) ∈ G ∧ f = Abs b ∧ g = Abs a) ∨
             (∃ a b c. (a, c) ∈ G ∧ (b, c) ∈ G ∧ f = Abs a ∧ g = Abs b)"

have "?P (set TI')"
using prems the_Abs_def[of f] the_Abs_def[of g] is_Abs_def[of f] is_Abs_def[of g]
by fastforce
hence "?P (TI')" unfolding prems by blast
hence "?P (rtrancl TI)" by (metis (no_types, lifting) trancl_into_rtrancl)
hence 5: "f = g ∨ (∃a b c. (a, c) ∈ TI' ∧ (b, c) ∈ TI' ∧ f = Abs a ∧ g = Abs b)" by blast

show ?case
  using timpl_closure_FunI3[OF _ 1(1) 5] IH 1(1)
  unfolding timpl_closure'_timpls_rtrancl_eq 2
  by auto
qed (use timpl_closure.FP in auto)

show "?A s t" when B: "?B s t"
proof -
  obtain u where "u ∈ timpl_closure s TI" "u ∈ timpl_closure t TI"
    using B by moura
  thus ?thesis using assms
  proof (induction u arbitrary: s t rule: term.induct)
    case (Var x s t)
    thus ?case
      using timpl_closure_Var_in_iff[of x s TI]
      by blast
    next
    case (Fun f U s t)
    obtain g S where "s = Fun g S" "length U = length S"
      "∀i. i < length U ⇒ U ! i ∈ timpl_closure (S ! i) TI"
      "g ≠ f ⇒ is_Abs g ∧ is_Abs f ∧ (the_Abs g, the_Abs f) ∈ TI'"
    using Fun.prems(1) timpl_closure_Fun_inv'[of f U _ _ TI]
    by (cases s) auto

    obtain h T where "t = Fun h T" "length U = length T"
      "∀i. i < length U ⇒ U ! i ∈ timpl_closure (T ! i) TI"
      "h ≠ f ⇒ is_Abs h ∧ is_Abs f ∧ (the_Abs h, the_Abs f) ∈ TI'"
    using Fun.prems(2) timpl_closure_Fun_inv'[of f U _ _ TI]
    by (cases t) auto

    have g: "(the_Abs g, the_Abs f) ∈ set TI'" "is_Abs f" "is_Abs g" when neq_f: "g ≠ f"
      proof -
        obtain ga fa where "g = Abs ga" "f = Abs fa"
          using s(4)[OF neq_f] unfolding is_Abs_def by presburger
        hence "the_Abs g ≠ the_Abs f" using neq_f by simp
        thus "(the_Abs g, the_Abs f) ∈ set TI'" "is_Abs f" "is_Abs g"
          using s(4)[OF neq_f] Fun.prems by blast+
      qed

    have h: "(the_Abs h, the_Abs f) ∈ set TI'" "is_Abs f" "is_Abs h" when neq_f: "h ≠ f"
      proof -
        obtain ha fa where "h = Abs ha" "f = Abs fa"
          using t(4)[OF neq_f] unfolding is_Abs_def by presburger
        hence "the_Abs h ≠ the_Abs f" using neq_f by simp
        thus "(the_Abs h, the_Abs f) ∈ set TI'" "is_Abs f" "is_Abs h"
          using t(4)[OF neq_f] Fun.prems by blast+
      qed

    have "equal_mod_timpls TI' (S ! i) (T ! i)"
      when i: "i < length U" for i
      using i Fun.IH s(1,2,3) t(1,2,3) nth_mem[OF i] Fun.prems by meson
    hence "list_all2 (equal_mod_timpls TI') S T"
      using list_all2_conv_all_nth[of "equal_mod_timpls TI'" S T] s(2) t(2) by presburger
    thus ?case using s(1) t(1) g h by fastforce

167
3 Stateful Protocol Verification

qed
qed
qed

context
begin

private inductive timpls_transformable_to_pred where
Var: "timpls_transformable_to_pred A (Var x) (Var x)"
| Fun: "¬is_Abs f; length T = length S; \(\forall i. i < length T \implies timpls_transformable_to_pred A (T ! i) (S ! i)\)" \(\implies timpls_transformable_to_pred A (Fun f T) (Fun f S)\)"
| Abs: "b \in A \implies timpls_transformable_to_pred A (Fun (Abs a) []) (Fun (Abs b) [])"

private lemma timpls_transformable_to_pred_inv_Var: assumes "timpls_transformable_to_pred A (Var x) t" shows "t = Var x" using assms by (auto elim: timpls_transformable_to_pred.cases)

private lemma timpls_transformable_to_pred_inv: assumes "timpls_transformable_to_pred A (Fun f T) t" shows "is_Fun t" and "length T = length (args t)" and "\(\forall i. i < length T \implies timpls_transformable_to_pred A (T ! i) (args t ! i)\)" and "¬is_Abs f \implies f = the_Fun t" and "is_Abs f \implies (is_Abs (the_Fun t) \land the_Abs (the_Fun t) \in A)"
using assms by (auto elim!: timpls_transformable_to_pred.cases[of A])

private lemma timpls_transformable_to_pred_finite_aux1: assumes f: "¬is_Abs f" shows "\{s. timpls_transformable_to_pred A (Fun f T) s\} \subseteq (\lambda S. Fun f S) ` \{S. length T = length S \land (\forall s \in set S. \exists t \in set T. timpls_transformable_to_pred A t s)\}" (is "?B \subseteq ?C")
proof
fix s assume s: "s \in ?B"
hence ": "timpls_transformable_to_pred A (Fun f T) s" by blast
obtain S where S: "s = Fun f S" "length T = length S" "\(\forall i. i < length T \implies timpls_transformable_to_pred A (T ! i) (S ! i)\)"
using f timpls_transformable_to_pred_inv[OF *] unfolding the_Abs_def is_Abs_def by auto
have "\(\forall s \in set S. \exists t \in set T. timpls_transformable_to_pred A t s\)" using S(2,3) in_set_conv_nth by sets
thus "s \in ?C" using S(1,2) by blast
qed

private lemma timpls_transformable_to_pred_finite_aux2: 
"\{s. timpls_transformable_to_pred A (Fun (Abs a) []) s\} \subseteq (\lambda b. Fun (Abs b) []) \ ` A" (is "?B \subseteq ?C")
proof
fix s assume s: "s \in ?B"
hence ": "timpls_transformable_to_pred A (Fun (Abs a) []) s" by blast
obtain b where b: "s = Fun (Abs b) []" "b \in A"
using timpls_transformable_to_pred_inv[OF *] unfolding the_Abs_def is_Abs_def by auto
thus "s \in ?C" by blast
qed

private lemma timpls_transformable_to_pred_finite: fixes t::"(('fun,'atom,'sets,'lbl) prot_fun, 'a) term" assumes A: "finite A"

168
3.5 Term Implication

and t: "wf_trm t"
shows "finite {s. timpls_transformable_to_pred A t s}"
using t

proof (induction t)
case (Var x)
have "{(s::(('fun,'atom,'sets,'lbl) prot_fun,'a) term. timpls_transformable_to_pred A (Var x) s) = (Var x)}"
by (auto intro: timpls_transformable_to_pred.Var elim: timpls_transformable_to_pred_inv_Var)
thus ?case by simp

next
case (Fun f T)
have IH: "finite {s. timpls_transformable_to_pred A t s}" when t: "t ∈ set T" for t
using Fun.IH[OF t] wf_trm_param[OF Fun.prems t]
by blast

show ?case
proof (cases "is_Abs f")
  case True
  then obtain a where "f = Abs a" unfolding is_Abs_def by presburger
  hence "T = []" using wf_trm_arity[OF Fun.prems] by simp_all
  hence "[a. timpls_transformable_to_pred A (Fun f T) a] ⊆ (λb. Fun (Abs b) []) " A"
  using timpls_transformable_to_pred_finite_aux2[of A a] a by auto
  thus ?thesis using A finite_subset by fast

next
case False
  thus ?thesis using IH finite_lists_length_eq' timpls_transformable_to_pred_finite_aux1[of f A T] finite_subset by blast
qed

private lemma timpls_transformable_to_pred_if_timpls_transformable_to:
assumes s: "timpls_transformable_to TI t s"
and t: "wf_trm t" "∀ f ∈ funs_term t. is_Abs f −→ the_Abs f ∈ A"
shows "timpls_transformable_to_pred (A ∪ fst ` (set TI)∪ snd ` (set TI)++) t s"
using s t
proof (induction rule: timpls_transformable_to.induct)
case (2 TI f T g S)
  let ?A = "A ∪ fst ` (set TI)∪ snd ` (set TI)++"

  note prems = "2.prems"
  note IH = "2.IH"

  note 0 = timpls_transformable_to_inv[OF prems(1)]

  have 1: "T = []" "S = []" when f: "f = Abs a" for a
  using f wf_trm_arity[OF prems(2)] 0(1) by simp_all

  have "∀ f ∈ funs_term t. is_Abs f −→ the_Abs f ∈ A" when t: "t ∈ set T" for t
  using t prems(3) funs_term_subterms_eq(1)[of "Fun f T"] by blast

  hence 2: "timpls_transformable_to_pred ?A (T ! i) (S ! i)"
  when i: "i < length T" for i
  using i IH 0(1,2) wf_trm_param[OF prems(2)]
  by (metis (no_types) in_set_conv_nth)

  have 3: "the_Abs f ∈ ?A" when f: "is_Abs f" using prems(3) f by force

  show ?case
  proof (cases "f = g")
    case True
    note fg = True
    show ?thesis
    proof (cases "is_Abs f")
      case True
      show ?thesis
    qed
  qed

qed
then obtain a where: "f = Abs a" unfolding is_Abs_def by moura
qed (use fg timpls_transformable_to_pred.Fun[of _ 0(1) 2, of f] in blast)

next
case False
then obtain a b where: "f = Abs a" "g = Abs b" "(a, b) ∈ (set TI) +" unfolding is_Abs_def the_Abs_def by fastforce
hence "a ∈ ?A" "b ∈ ?A" by force+
thus ?thesis using timpls_transformable_to_pred.Abs ab(1,2) 1[of ab(1)] by metis
qed (simp_all add: timpls_transformable_to_pred.Var)

private lemma timpls_transformable_to_pred_if_timpls_transformable_to':
assumes s: "timpls_transformable_to' TI t s"
and t: "wf_trm t" "∀ f ∈ funs_term t. is_Abs f −→ the_Abs f ∈ A"
shows "timpls_transformable_to_pred (A ∪ fst ` (set TI) + ∪ snd ` (set TI) +) t s"
proof (induction rule: timpls_transformable_to.induct)
case (2 TI f T g S)

have 1: "T = []" "S = []" when f: "f = Abs a" for a
using f wf_trm_arity[of prems(2)] 0(1) by simp_all

have "∀ f ∈ funs_term t. is_Abs f −→ the_Abs f ∈ A" when t: "t ∈ set T" for t
using t prems(3) funs_term_subterms_eq(1)[of "Fun f T"] by blast

have 2: "timpls_transformable_to_pred ?A (T ! i) (S ! i)"
when i: "i < length T" for i
using i IH 0(1,2) wf_trm_param[of prems(2)]
by (metis (no_types) in_set_conv_nth)

have 3: "the_Abs f ∈ ?A" when f: "is_Abs f" using prems(3) f by force

show ?case
proof (cases "f = g")
case True
note fg = True

show ?thesis
proof (cases "is_Abs f")
case True
then obtain a where: "f = Abs a" unfolding is_Abs_def by moura
qed (use fg timpls_transformable_to_pred.Fun[of _ 0(1) 2, of f] in blast)

next
case False
then obtain a b where: "f = Abs a" "g = Abs b" "(a, b) ∈ (set TI) +" unfolding is_Abs_def the_Abs_def by fastforce
hence "a ∈ ?A" "b ∈ ?A" by force+
thus ?thesis using timpls_transformable_to_pred.Abs ab(1,2) 1[of ab(1)] by metis
qed (simp_all add: timpls_transformable_to_pred.Var)

private lemma timpls_transformable_to_pred_if_equal_mod_timpls:
assumes s: "equal_mod_timpls TI t s"
and t: "wf_trm t" "∀ f ∈ funs_term t. is_Abs f −→ the_Abs f ∈ A"
shows "timpls_transformable_to_pred (A ∪ fst ` (set TI) + ∪ snd ` (set TI) +) t s"
using s t

proof (induction rule: equal_mod_timpls.induct)
  case (2 TI f T g S)
  let ?A = "A ∪ fst ` (set TI)⁺ ∪ snd ` (set TI)⁺"

  note prems = "2.prems"
  note IH = "2.IH"

  note 0 = equal_mod_timpls_inv[OF prems(1)]
  have 1: "T = []" "S = []" when f: "f = Abs a" for a
    using f wf_trm_arity[OF prems(2)] 0(1) by simp

  have "∀ f ∈ funs_term t. is_Abs f → the_Abs f ∈ A" when t: "t ∈ set T" for t
    using t prems(3) funs_term_subterms_eq(1)[of "Fun f T"] by blast
  hence 2: "timpls_transformable_to_pred ?A (T ! i) (S ! i)"
    when i: "i < length T" for i
    using IH 0(1,2) wf_trm_param[OF prems(2)]
    by (metis (no_types) in_set_conv_nth)

  have 3: "the_Abs f ∈ ?A" when f: "is_Abs f"
    using prems(3) f by force

  show ?case
    proof (cases "f = g")
      case True
      note fg = True
      show ?thesis
        proof (cases "is_Abs f")
          case True
          then obtain a where "f = Abs a" unfolding is_Abs_def by moura
            by simp
        qed
      next
      case False
      then obtain a b where "f = Abs a" "g = Abs b"
        "(a, b) ∈ (set TI)⁺ " "(b, a) ∈ (set TI)⁺"
        "(a, ti) ∈ set TI. (a, snd ti) ∈ (set TI)⁺ "
        "(b, ti) ∈ set TI. (b, snd ti) ∈ (set TI)⁺"
        unfolding in_trancl_closure_iff_in_trancl_fun[of _ _ TI]
        using 0(3) in_trancl_closure_iff_in_trancl_fun[of _ _ TI]
        by fastforce
      hence "a ∈ ?A" "b ∈ ?A" by force
      thus ?thesis using timpls_transformable_to_pred.Abs ab(1,2) 1[OF ab(1)] by metis
    qed
  qed (simp_all add: timpls_transformable_to_pred.Var)

lemma timpls_transformable_to_finite:
  assumes t: "wf_trm t"
  shows "finite {s. timpls_transformable_to TI t s}" (is ?P)
  and "finite {s. timpls_transformable_to' TI t s}" (is ?Q)
  proof -
    let ?A = "the_Abs`{f ∈ funs_term t. is_Abs f} ∪ fst` (set TI)⁺ ∪ snd` (set TI)⁺"
    have 0: "finite ?A" by auto

    have 1: "{s. timpls_transformable_to TI t s} ⊆ {s. timpls_transformable_to_pred ?A t s}" using timpls_transformable_to_pred_finite[OF 0 t] finite_subset[OF 1] by blast
    have 2: "{s. timpls_transformable_to' TI t s} ⊆ {s. timpls_transformable_to_pred ?A t s}" using timpls_transformable_to_pred_finite[OF 0 t] finite_subset[OF 2] by blast
    qed
lemma equal_mod_timpls_finite:
  assumes t: "wf_trm t"
  shows "finite {s. equal_mod_timpls TI t s}"
proof -
  let ?A = "the_Abs ` {f ∈ funs_term t. is_Abs f} ∪ fst ` (set TI)⁺ ∪ snd ` (set TI)⁺"
  have 0: "finite ?A" by auto
  have 1: "{s. equal_mod_timpls TI t s} ⊆ {s. timpls_transformable_to_pred ?A t s}"
        using timpls_transformable_to_pred_if_equal_mod_timpls[OF _ t] by auto
  show ?thesis using timpls_transformable_to_pred_finite[OF 0 t] finite_subset[OF 1] by blast
qed

lemma intruder_synth_mod_timpls_is_synth_timpl_closure_set:
  fixes t::"(('fun,'atom,'sets,'lbl) prot_fun, 'a) term" and TI TI'
  assumes "set TI' = {(a,b) ∈ (set TI)⁺. a ≠ b}"
  shows "intruder_synth_mod_timpls M TI' t ←→ timpl_closure_set (set M) (set TI) ⊢ c t"
        (is "?C t ←→ ?D t")
proof -
  have *: "(∃ m ∈ M. timpls_transformable_to TI' m t) ←→ t ∈ timpl_closure_set M (set TI)"
        when "set TI' = {(a,b) ∈ (set TI)⁺. a ≠ b}"
  proof (induction t arbitrary: M TI TI' rule: intruder_synth_mod_timpls.induct)
    case (1 M TI' x)
    hence "Var x ∈ timpl_closure_set (set M) (set TI)"
    unfolding timpl_closure_set_def by force
    thus ?case by simp
  next
    case (2 M TI f T)
    show ?case
      proof
        case (inr f T)
        hence "¬ (list_ex (λ t. timpls_transformable_to TI' t (Fun f T)))" by blast
        thus ?thesis
          using intruder_synthesis.AxiomC[of "Fun f T" timpl_closure_set (set M) (set TI)"
          unfolding list_ex_iff by blast
  next
    case False
    hence "¬ (list_ex (∀ t. timpls_transformable_to TI' t (Fun f T)))"
    unfolding list_ex_iff by blast
    hence "¬ ((Fun f T) " length T = arity f") "list_all (intruder_synth_mod_timpls M TI') T"
    using "2.prems"(1) by force+
    thus ?thesis using "2.IH"[OF _ "2.prems"(2)] unfolding list_all_iff by force
  qed
qed

show "?C t ←→ ?D t"
proof (induction t rule: intruder_synth_induct)
  case (AxiomC t)
  thus ?case
    using timpl_closure_set_Var_in_iff[of _ "set M" "set TI"
    by (cases t rule: term.exhaust) (force simp add: member_def list_ex_iff)+
next
3.5 Term Implication

Theorem

intruder_synth_mod_timpls'_is_synth_timpl_closure_set:

fixes t::"('fun,'atom,'sets,'lbl) prot_fun, 'a) term" and TI

shows "intruder_synth_mod_timpls' M TI t ←→ timpl_closure_set (set M) (set TI)

(is "?A t ←→ ?B t")

proof -

have #: "("fun,'atom,'sets,'lbl) prot_fun, 'a) term" and TI

shows "intruder_synth_mod_timpls' M TI t ←→ timpl_closure_set (set M) (set TI)"

for M TI and t::"('fun,'atom,'sets,'lbl) prot_fun, 'a) term"

using timpls_transformable_to'_iff_in_timpl_closure[of TI _ t]

and timpl_closure_set_is_timpl_closure_union[of M "set TI"]

by blast+

thus ?case by simp

show "?A t ←→ ?B t"

proof

show "?A t =⇒ ?B t"

proof (induction t arbitrary: M TI rule: intruder_synth_mod_timpls'.induct)

  case (1 M TI x)

  hence "Var x ∈ timpl_closure_set (set M) (set TI)"

  using timpl_closure.FP List.member_def[of M]

  unfolding timpl_closure_set_def

  by auto

  thus ?case by simp

next

  case (2 M TI f T)

  show ?case

  proof

    case (True)

    hence "public f" "length T = arity f" "list_all (intruder_synth_mod_timpls' M TI) T"

    using "2.prems" list_ex_iff[of _ M]

    by force+

    thus ?thesis

    using "2.IH"[of _ M TI] list_all_iff[of "intruder_synth_mod_timpls' M TI" T]

    by force

  qed

next

  case (False)

  hence "public f" "length T = arity f" "list_all (intruder_synth_mod_timpls' M TI) T"

  using "2.prems" list_ex_iff[of _ M]

  by force+

  thus ?thesis

  using "2.IH"[of _ M TI] list_all_iff[of "intruder_synth_mod_timpls' M TI" T]

  by force

qed

show "?B t =⇒ ?A t"

proof (induction t rule: intruder_synth_induct)

  case (AxiomC t)

  hence "("fun,'atom,'sets,'lbl) prot_fun, 'a) term" and TI

  shows "intruder_synth_mod_timpls' M TI t ←→ timpl_closure_set (set M) (set TI)"

  using AxiomC timpl_closure_set_Var_in_iff[of _ "set M" "set TI"]

  by (cases t rule: term.exhaust) force+

next

  case (ComposeC T f)

  hence "("fun,'atom,'sets,'lbl) prot_fun, 'a) term" and TI

  shows "intruder_synth_mod_timpls' M TI t ←→ timpl_closure_set (set M) (set TI)"

  using list_all_iff[of "intruder_synth_mod_timpls' M TI" T]

  by blast

qed

lemma intruder_synth_mod_eq_timpls_is_synth_timpl_closure_set:

fixes t::"('fun,'atom,'sets,'lbl) prot_fun, 'a) term" and TI

defines "c1 ≡ λTI. (a,b) ∈ TI". (a,b) ∈ TI". a ≠ b)"
shows \[ \text{"set } TI' = \{(a,b) \in (\text{set } TI) \mid a \neq b\} \Rightarrow \]
\[ \text{intruder_synth_mod_eq_timpls } M \text{ } TI' \text{ } \mapsto \]
\[ (\exists s \in \text{timpl_closure } t \text{ (set } TI). \text{ timpl_closure_set } M \text{ (set } TI) \vdash c \text{ } s)\]
(is "?Q TI TI' \implies ?C t \iff ?D t")

proof -

have **: \[ (\exists m \in M. \text{equal_mod_timpls } TI' \text{ } m \text{ } t) \iff \]
\[ (\exists s \in \text{timpl_closure } t \text{ (set } TI). s \in \text{timpl_closure_set } M \text{ (set } TI))\]
when \(Q: \text{"?Q TI TI'"}\)

for \(M \text{ } TI'\) and \(t:\text{"('fun,'atom,'sets,'lbl) prot_fun, 'a) term"}\)
using equal_mod_timpls_iff_ex_in_timpl_closure[OF Q]
timpl_closure_set_is_timpl_closure_union[of M "set TI"]
timpl_closure_set_timpls_trancl_eq'[of M "set TI"]
by fastforce

show "?C t \iff ?D t" when \(Q: \text{"?Q TI TI'"}\)

proof

show "?C t \implies ?D t" using Q

proof (induction \(t\) arbitrary: \(M \text{ } TI\) rule: intruder_synth_mod_eq_timpls.induct)
case (1 \(M \text{ } TI' \text{ } x \text{ } M \text{ } TI\))
hence "Var x \in \text{timpl_closure_set } M \text{ (set } TI)" "Var x \in \text{timpl_closure } (Var x) \text{ (set } TI)"
using timpl_closure.FP unfolding timpl_closure_set_def by auto
thus ?case by force

next
case (2 \(M \text{ } TI' \text{ } f \text{ } T \text{ } M \text{ } TI\))
show ?case

proof (cases "\(\exists m \in M. \text{equal_mod_timpls } TI' \text{ } m \text{ } (\text{Fun } f \text{ } T)\")
case True thus ?thesis
using **[OF "2.prems"(2), of M "Fun f T"]
intruder_synth.AxiomC[of _ "timpl_closure_set M (set TI)"]
by blast

next
case False
hence f: "public f" "length T = arity f" "\list_all (intruder_synth_mod_eq_timpls M TI') T" using "2.prems" by force+

let \(?sy = \text{"intruder_synth } (\text{timpl_closure_set } M \text{ (set } TI))"\)

have IH: "\(\exists u \in \text{timpl_closure } (T ! i) \text{ (set } TI). ?sy u)\"
when i: "\(i < length T\)" for i
using "2.IH[of _ M TI] f(3) nth_mem[OF i] "2.prems"(2)
unfolding list_all_iff by blast

define \(S\) where "\(S \equiv \text{map } (\lambda u. \text{SOME } v. v \in \text{timpl_closure } u \text{ (set } TI) \land ?sy v) T\)"

have S1: "length T = length S"
unfolding S_def by simp

have S2: "\(S ! i \in \text{timpl_closure } (T ! i) \text{ (set } TI)\)"
\[ \text{timpl_closure_set } M \text{ (set } TI) \vdash c \text{ } S ! i\]
when i: "\(i < length S\)" for i
using i IH somel_ex[of "\(\forall v \in \text{timpl_closure } (T ! i) \text{ (set } TI) \land ?sy v\)"
unfolding S_def by auto

have "\(\textFun f S \in \text{timpl_closure } (\textFun f T) \text{ (set } TI)\)"
using timpl_closure_Fun[of T S "set TI" f f] S1 S2(1)
unfolding timpl_closure_is_timpl_closure' by prebnger
thus ?thesis
by (metis intruder_synth.ComposeC[of S f] f(1,2) S1 S2(2) in_set_conv_nth[of _ S])
qed
3.5 Term Implication

show "?C t" when D: "?D t"
proof -
  obtain s where "timpl_closure_set M (set TI) \vdash s" "s \in timpl_closure t (set TI)"
    using D by moura
thus ?thesis
proof (induction s arbitrary: t rule: intruder_synth_induct)
case (AxiomC s t)
  note 1 = timpl_closure_set_Var_in_iff[of _ M "set TI"] timpl_closure_Var_inv[of s _ "set TI"]
  note 2 = **[OF Q, of M]
  show ?case
  proof
    (cases t)
    case Var
    thus ?thesis
      using 1 AxiomC by auto
  next
    case Fun
    thus ?thesis
      using 2 AxiomC by auto
  qed
next
case (ComposeC T f t)
  obtain g S where gS:
    "t = Fun g S" "length S = length T"
    "\vdash i < length T. T ! i \in timpl_closure (S ! i) (set TI)"
    "g \neq f \Longrightarrow is_Abs g \land is_Abs f \land (the_Abs g, the_Abs f) \in (set TI) +"
    using ComposeC.prems(1) timpl_closure'_inv'[of t "Fun f T" "set TI"]
    timpl_closure_is_timpl_closure'[of _ _ "set TI"]
    by fastforce
  have IH: "intruder_synch_mod_eq_timpls M TI' u" when u: "u \in set S" for u
    by (metis u gS(2,3) ComposeC.IH in_set_conv_nth)
  note 0 = list_all_iff[of "intruder_synch_mod_eq_timpls M TI'" S]
    intruder_synch_mod_eq_timpls.simps(2)[of M TI' g S]
  have "f = g" using ComposeC.hyps gS(4) unfolding is_Abs_def by fastforce
  thus ?case
    by (metis ComposeC.hyps(1,2) gS(1,2) IH 0)
qed
qed

qed

lemma timpl_closure_finite:
  assumes t: "$wfrm t"
  shows "finite (timpl_closure t (set TI))"
using timpls_transformable_to'_iff_in_timpl_closure[of t t]
  timpls_transformable_to_finite[of _ _]
by auto

lemma timpl_closure_set_finite:
  fixes TI::"('sets set \times 'sets set) list"
  assumes M_finite: "finite M" and M_wf: "$wfrm, M" showing "finite (timpl_closure_set M (set TI))"
using timpl_closure_set_is_timpl_closure_union[of M "set TI"]
  timpl_closure_finite[of _ TI] M_finite M_wf finite
by auto

lemma comp_timpl_closure_is_timpl_closure_set:
  fixes M and TI::"('sets set \times 'sets set) list"
  assumes M_finite: "finite M" and M_wf: "$wfrm, M" showing "comp_timpl_closure M (set TI) = timpl_closure_set M (set TI)"
using lfp_while'[OF timpls_Un_mono[of M "set TI"]]
  timpl_closure_set_finite[OF M_finite M_wf]

private lemma analyzed_closed_mod_timpls_is_analyzed_closed_timpl_closure_set_aux1:
fixes M::"('fun,'atom,'sets,'lbl) prot_terms"
assumes f: "arity f = length T" "arity f > 0" "Ana f = (K, R)"
and i: "i < length R"
and M: "timpl_closure_set M TI \models T ! (R ! i)"
and m: "Fun (Fu f) T \in M"
and t: "Fun (Fu f) S \in timpl_closure (Fun (Fu f) T) TI"
shows "timpl_closure_set M TI \models S ! (R ! i)"
proof -
  have "R ! i < length T" using i Ana_f[of f(3)] f(1) by simp
  thus ?thesis using timpl_closure_Fun_inv'(1,2)[OF t] intruder_synth_timpl_closure'[OF M]
  by presburger
qed

private lemma analyzed_closed_mod_timpls_is_analyzed_closed_timpl_closure_set_aux2:
fixes M::"('fun,'atom,'sets,'lbl) prot_terms"
assumes M: "\forall s \in set (snd (Ana m)). timpl_closure_set M TI \models s"
and m: "m \in M"
and t: "t \in timpl_closure m TI"
and s: "s \in set (snd (Ana t))"
shows "timpl_closure_set M TI \models s"
proof -
  obtain f S K N where fS: "t = Fun (Fu f) S" "arity f = length S" "0 < arity f" 
    and Ana_f: "Ana f = (K, N)"
  using Ana_nonempty_inv[of t] s by fastforce
  then obtain T where T: "m = Fun (Fu f) T" "length T = length S"
    using timpl_closure_Fu_inv'[of f S m TI]
    by moura
  hence Ana_m: "Ana m = (K \cdot list (!) T, map (!) T) N"
    using fS(2,3) Ana_f by auto
obtain i where i: "i < length N" "s = S ! (N ! i)"
  using s[unfolded fS(1)] Ana_t[unfolded fS(1)] T(2) 
    in_set_conv_nth[of s "map (\lambda i. S ! i) N"]
  by auto
  hence "timpl_closure_set M TI \models T ! (N ! i)"
  using M[unfolded T(1)] Ana_m[unfolded T(1)] T(2)
  by simp
  thus ?thesis using analyzed_closed_mod_timpls_is_analyzed_closed_timpl_closure_set_aux1[ 
    OF fS(2)[unfolded T(2)[symmetric]] fS(3) Ana_f 
    i(1) \_ m[unfolded T(1)] t[unfolded fS(1) T(1)]]
  i(2)
  by argo
qed

lemma analyzed_closed_mod_timpls_is_analyzed_timpl_closure_set:
fixes M::"('fun,'atom,'sets,'lbl) prot_term list"
assumes TI': "set TI' = {(a,b) \in (set TI)\+. a \neq b}"
and M_wf: "wterm (set M)"
shows "analyzed_closed_mod_timpls M TI' \iff analyzed (timpl_closure_set (set M) (set TI))"
(is "?A \iff ?B")
proof
  let ?C = "\forall t \in timpl Closure (set M) (set TI). 
    analyzed_in t (timpl_closure_set (set M) (set TI))"

176
let $\pi_P = \lambda T. \forall t \in \text{set } T. \text{timpl\_closure\_set (set } M) (\text{set } TI) \vdash c_t$
let $\pi_Q = \lambda t. \forall s \in \text{comp\_timpl\_closure } \{t\} (\text{set } TI'). \text{case Ana } s \text{ of } (K, R) \Rightarrow \pi_P K \rightarrowtail \pi_P R$
let $\pi_W = \lambda t. \forall t \in \text{set (fst } (\text{Ana } t)). \forall f \in \text{funs\_term } t. \neg \text{is\_Abs } f$
let $\pi_V = \lambda t. \forall s \in \text{comp\_timpl\_closure } (\text{set (fst } (\text{Ana } t))) (\text{set } TI'). \neg \text{timpl\_closure\_set (set } M) (\text{set } TI) \vdash c_s$

note defs = analyzed\_closed\_mod\_timpls\_def analyzed\_in\_code
note $0 = \text{intruder\_synth\_mod\_timpls\_is\_synth\_timpl\_closure\_set[of } TI', \text{ of } M$
note $1 = \text{timpl\_closure\_set\_is\_timpl\_closure\_union[of } _{\text{set } TI'})$

have $2: \text{comp\_timpl\_closure } N (\text{set } TI') = \text{timpl\_closure\_set } N (\text{set } TI)$
  when $\text{wf } trms N$ $\text{finite } N$
  for $N::(\text{fun},\text{atom},\text{sets},\text{lbl}) \text{ prot\_terms}$
  using $\text{timpl\_closure\_set\_timpls\_trancl\_eq'[of } N \text{ set } TI']$
  $\text{comp\_timpl\_closure\_is\_timpl\_closure\_set[of } N \text{ TI']}$
  unfolding $\text{TI'}[\text{symmetric}]$ by presburger

hence $3: \text{comp\_timpl\_closure } \{t\} (\text{set } TI') \subseteq \text{timpl\_closure\_set } (\text{set } M) (\text{set } TI)$
  when $t: t \in \text{set } M$ $\text{wf } \text{trm } t$
  for $t$
  using $\text{timpl\_closure\_set\_mono[of } \{t\}\text{ set } M]$ by simp

have $?A$ when $C: ?C$
  unfolding analyzed\_closed\_mod\_timpls\_def
  intruder\_synth\_mod\_timpls\_is\_synth\_timpl\_closure\_set[of } TI'
  list\_all\_iff Let\_def
  proof (intro ballI)
    fix $t$ assume $t: t \in \text{set } M$
    show "if $\pi_P (\text{fst } (\text{Ana } t))$ then $\pi_P (\text{snd } (\text{Ana } t))$
    else if $\pi_W t$ then True
    else if $\pi_V t$ then True
    else $\pi_Q t" (is $?R$)
    proof (cases "$\pi_P (\text{fst } (\text{Ana } t))")$
      case True
      hence "$\pi_P (\text{snd } (\text{Ana } t))$" using $C$ $t$
      unfolding analyzed\_in\_def
      list\_all\_iff Let\_def
      proof (intro ballI allI impI; elim conjE)
        fix $t$ $K$ $T$ $s$
        assume $t: t \in \text{timpl\_closure\_set } (\text{set } M) (\text{set } TI)$
        and $s: s \in \text{set } T$
        and $\text{Ana}_t: \"\text{Ana } t = (K, T)\"
        and $K: \"\forall k \in \text{set } K. \text{timpl\_closure\_set } (\text{set } M) (\text{set } TI) \vdash c_k\"

        obtain $m$ where $m: m \in \text{set } M" t \in \text{timpl\_closure } m (\text{set } TI)"
          using $\text{timpl\_closure\_set\_is\_timpl\_closure\_union } t$ by moura

        show "$\text{timpl\_closure\_set } (\text{set } M) (\text{set } TI) \vdash c_s$"
          proof (cases "$\forall k \in \text{set } (\text{fst } (\text{Ana } m)). \text{timpl\_closure\_set } (\text{set } M) (\text{set } TI) \vdash c_k")$
            case True
            hence $*: \"\forall r \in \text{set } (\text{snd } (\text{Ana } m)). \text{timpl\_closure\_set } (\text{set } M) (\text{set } TI) \vdash c_r"$
              using $m(1)$ $K$
              unfolding analyzed\_closed\_mod\_timpls\_def
              intruder\_synth\_mod\_timpls\_is\_synth\_timpl\_closure\_set[of } TI'
              list\_all\_iff Let\_def

have $?C$ when $A: ?A$
  unfolding analyzed\_in\_def Let\_def
  proof (intro ballI allI impI; elim conjE)
    fix $t$ $K$ $T$ $s$
    assume $t: t \in \text{timpl\_closure\_set } (\text{set } M) (\text{set } TI)$
    and $s: s \in \text{set } T$
    and $\text{Ana}_t: \"\text{Ana } t = (K, T)\"
    and $K: \"\forall k \in \text{set } K. \text{timpl\_closure\_set } (\text{set } M) (\text{set } TI) \vdash c_k\"

    obtain $m$ where $m: m \in \text{set } M" t \in \text{timpl\_closure } m (\text{set } TI)"
      using $\text{timpl\_closure\_set\_is\_timpl\_closure\_union } t$ by moura

    show "$\text{timpl\_closure\_set } (\text{set } M) (\text{set } TI) \vdash c_s$"
      proof (cases "$\forall k \in \text{set } (\text{fst } m). \text{timpl\_closure\_set } (\text{set } M) (\text{set } TI) \vdash c_k")$
        case True
        hence $*: \"\forall r \in \text{set } (\text{snd } m). \text{timpl\_closure\_set } (\text{set } M) (\text{set } TI) \vdash c_r"$
          using $m(1)$ $K$
          unfolding analyzed\_closed\_mod\_timpls\_def
          intruder\_synth\_mod\_timpls\_is\_synth\_timpl\_closure\_set[of } TI'
          list\_all\_iff Let\_def

    qed
3 Stateful Protocol Verification

by simp

show ?thesis
  using K s Ana_t A
  analyzed_closed_mod_timpls_is_analyzed_closed_timpl_closure_set_aux2[OF * m]
  by simp
next
case False
  note F = this

have *: "comp_timpl_closure {m} (set TI') = timpl_closure m (set TI)"
  using 2[of "{m}"] timpl_closureton_is_timpl_closure M_wf m(1)
  by blast

have "wftrms (set (fst (Ana m)))"
  using Ana_keys_wf'[of m "fst (Ana m)"] M_wf m(1) surj_pair[of "Ana m"] by fastforce

hence **: "comp_timpl_closure (set (fst (Ana m))) (set TI') =
  timpl_closure_set (set (fst (Ana m))) (set TI)"
  using 2[of "set (fst (Ana m))"] by blast

have ***: "set K ⊆ timpl_closure_set (set (fst (Ana m))) (set TI)"
  "length K = length (fst (Ana m))"
  using timpl_closure_Ana_keys_subset[OF m(2) _ Ana_t]
  timpl_closure_Ana_keys_length_eq[OF m(2) _ Ana_t]
  surj_pair[of "Ana m"]
  by fastforce+

show ?thesis
proof (cases "?W m")
  case True
  hence "fst (Ana t) = fst (Ana m)" using m timpl_closure_Ana_keys_no_Abs_eq_case by fast
  thus ?thesis using F K Ana_t by simp
next
case False
  note F' = this

show ?thesis
proof (cases "?V m")
  case True
  hence "∀ k ∈ set K. ¬ timpl_closure_set (set M) (set TI) ⊢ c k"
    using F K Ana_t m *** unfolding ** by blast
  thus ?thesis using K F' *** by simp
next
case False
  hence "?Q m"
    using m(1) A F F'
    unfolding analyzed_closed_mod_timpls_def
    intruder_synth_mod_timpls_is_synth_timpl_closure_set[OF TI']
    list_all_iff Let_def
    by auto
  thus ?thesis
    using * m(2) K s Ana_t
    unfolding Let_def by auto
qed

qed

thus ?B when A: ?A using A analyzed_is_all_analyzed_in by metis
qed

lemma analyzed_closed_mod_timpls'_is_analyzed_timpl_closure_set:
  fixes M::"('(fun,'atom,'sets,'lbl) prot_term list"
  assumes M_wf: "wftrms (set M)"
shows "analyzed_closed_mod_timpls' M TI ←→ analyzed (timpl_closure_set (set M) (set TI))"
(is "?A ←→ ?B")
proof
let ?C = "∀ t ∈ timpl_closure_set (set M) (set TI). analyzed_in t (timpl_closure_set (set M) (set TI))"

let ?P = "∀ t ∈ set T. timpl_closure_set (set M) (set TI) ⊢ t"
let ?Q = "∀ t s ∈ comp_timpl_closure {t} (set TI). case Ana t of (K, R) ⇒ ?P K → ?P R"
let ?W = "∀ t ∈ set (fst (Ana t)). ∀ f ∈ funs_term t. ¬ is_Abs f"

note defs = analyzed_closed_mod_timpls'_def analyzed_in_code
note 0 = intruder_synth_mod_timpls'_is_synth_timpl_closure_set[of M TI]
note 1 = timpl_closure_set_is_timpl_closure_union[of _ "set TI"]

have 2: "comp_timpl_closure {t} (set TI) = timpl_closure_set {t} (set TI)"
  when t: "t ∈ set M" "wf trm t" for t
  using t timpl_closure_set_timpls_trancl_eq[of "{t}" "set TI"]
  by blast
hence 3: "comp_timpl_closure {t} (set TI) ⊆ timpl_closure_set (set M) (set TI)"
  when t: "t ∈ set M" "wf trm t" for t
  using t timpl_closure_set_mono[of "{t}" "set M"]
  by fast

have ?A when C: ?C
  unfolding analyzed_closed_mod_timpls'_def
  intruder_synth_mod_timpls'_is_synth_timpl_closure_set
  list_all_iff Let_def
  proof (intro ballI allI impI; elim conjE)
    fix t assume t: "t ∈ set M"
    show "if ?P (fst (Ana t)) then ?P (snd (Ana t)) else if ?W t then True else ?Q t" (is ?R)
      proof (cases "?P (fst (Ana t))")
        case True
        hence "?P (snd (Ana t))"
          using C timpl_closure_setI[of t, of "set TI"] prod.exhaust_sel
          unfolding analyzed_in_def by blast
        thus ?thesis using True by simp
      next
      case False
      have "?Q t" using 3[of t] C M wf t unfolding analyzed_in_def by auto
      thus ?thesis using False by argo
    qed
  qed
thus ?A when B: ?B using B analyzed_is_all_analyzed_in by metis

have ?C when A: ?A unfolding analyzed_in_def Let_def
proof (intro ballI allI impI; elim conjE)
  fix t K T s
  assume t: "t ∈ timpl_closure_set (set M) (set TI)"
  and s: "s ∈ set T"
  and Ana_t: "Ana t = (K, T)"
  and K: "∀ k ∈ set K. timpl_closure_set (set M) (set TI) ⊢ c k"

  obtain m where m: "m ∈ set M" "t ∈ timpl_closure m (set TI)"
    using timpl_closure_set_is_timpl_closure_union t by moura

  show "timpl_closure_set (set M) (set TI) ⊢ c s"
  proof (cases "∀ k ∈ set (fst (Ana m)). timpl_closure_set (set M) (set TI) ⊢ c k")
    case True
    hence "∀ r ∈ set (snd (Ana m)). timpl_closure_set (set M) (set TI) ⊢ c r"
      using m(1) A
    unfolding analyzed_closed_mod_timpls'_def
    intruder_synth_mod_timpls'_is_synth_timpl_closure_set
3 Stateful Protocol Verification

list_all_iff
by simp

show ?thesis
using K s Ana_t A
analyzed_closed_mod_timpls_is_analyzed_closed_timpl_closure_set_aux2[OF * m]
by simp
next
case False
note F = this

have *: "comp_timpl_closure {m} (set TI) = timpl_closure m (set TI)"
using 2[OF m(1)] timpl_closureton_is_timpl_closure M_wf m(1)
by blast

show ?thesis
proof (cases "?W m")
case True
hence "fst (Ana t) = fst (Ana m)" using m timpl_closure_Ana_keys_no_Abs_eq_case by fast
thus ?thesis using F K Ana_t by simp
next
case False
hence "?Q m"
using m(1) A F
unfolding intruder_synth_mod_timpls'_is_synth_timpl_closure_set
list_all_iff Let_def
by auto
thus ?thesis
using * m(2) K s Ana_t
unfolding Let_def by auto
qed
qed

thus ?B when A: ?A using A analyzed_is_all_analyzed_in by metis
qed

end

3.6 Stateful Protocol Verification

theory Stateful_Protocol_Verification
imports Stateful_Protocol_Model Term_Implication
begin

3.6.1 Fixed-Point Intruder Deduction Lemma

context stateful_protocol_model
begin

abbreviation pubval_terms::"('fun,'atom,'sets,'lbl) prot_terms" where
"pubval_terms ≡ {t. ∃ f ∈ funs_term t. is_PubConstValue f}"

abbreviation abs_terms::"('fun,'atom,'sets,'lbl) prot_terms" where
"abs_terms ≡ {t. ∃ f ∈ funs_term t. is_Abs f}"

definition intruder_deduct_GSMP::
"[('fun,'atom,'sets,'lbl) prot_terms,
3.6 Stateful Protocol Verification

(‘fun,’atom,’sets,’lbl) prot_terms,
(‘fun,’atom,’sets,’lbl) prot_term
⇒ bool" ("(_,__" _50)

where
"⟨⟨M; T⟩ ⊢ GSM P t = intruder_deduct_restricted M (λt. t ∈ GSMP T - (pubval_terms ∪ abs_terms)) t"

lemma intruder_deduct_GSMP_induct[consumes 1, case_names AxiomH ComposeH DecomposeH]:
  assumes "⟨⟨M; T⟩ ⊢ GSM P t" "⋀t. t ∈ M =⇒ P M t"
  shows "P M t"
proof -
  let ?Q = "λt. t ∈ GSMP T - (pubval_terms ∪ abs_terms)"
  show ?thesis
    using intruder_deduct_restricted_induct[of M ?Q t "λM Q t. P M t"
                            assms]
    unfolding intruder_deduct_GSMP_def
    by blast
  qed

lemma pubval_terms_subst:
  assumes "t · ϑ ∈ pubval_terms" "ϑ ` fv t ∩ pubval_terms = {}"
  shows "t ∈ pubval_terms"
using assms(1,2)
proof (induction t)
  case (Fun f T)
  let ?P = "λf. is_PubConstValue f"
  from Fun show ?case
  proof (cases "?P f")
    case False
    then obtain t where "t ∈ set T" "t · ϑ ∈ pubval_terms"
      using Fun.prems by auto
    hence "ϑ ` fv t ∩ pubval_terms = {}" using Fun.prems(2) by auto
    thus ?thesis using Fun.IH[OF t] t(1) by auto
  qed force
  qed simp

lemma abs_terms_subst:
  assumes "t · ϑ ∈ abs_terms" "ϑ ` fv t ∩ abs_terms = {}"
  shows "t ∈ abs_terms"
using assms(1,2)
proof (induction t)
  case (Fun f T)
  let ?P = "λf. is_Abs f"
  from Fun show ?case
  proof (cases "?P f")
    case False
    then obtain t where "t ∈ set T" "t · ϑ ∈ abs_terms"
      using Fun.prems by auto
    hence "ϑ ` fv t ∩ abs_terms = {}" using Fun.prems(2) by auto
    thus ?thesis using Fun.IH[OF t] t(1) by auto
  qed force
  qed simp

lemma pubval_terms_subst':
  assumes "t · ϑ ∈ pubval_terms" "∀n. PubConst Value n /∈ \bigcup (funs_term ` (ϑ ` fv t))"
  shows "t ∈ pubval_terms"
proof -
  have False
when $fs$: "$f \in funs_term s" "s \in subterms_{set} (\emptyset \setminus fv t)" "is_PubConstValue f" for $f$ $s$
proof -
obtain $T$ where $T$: "Fun $f$ $T \in subterms s" using funs_term_Fun_subterm[of $fs(1)$] by moura
hence "Fun $f$ $T \in subterms_{set} (\emptyset \setminus fv t)" using $fs(2)$ in_subterms_subset_Union by blast
thus $\text{thesis}$
  unfolding is_PubConstValue_def
by (cases $f$) force+
qed

thus $\text{thesis}$ using pubval_terms_subst[of $assms(1)$] by auto
qed

lemma abs_terms_subst':
assumes "$t \cdot \emptyset \in abs_terms" 
\( \forall n. Abs n \notin \bigcup (funs_term ` (\emptyset \setminus fv t))" 
shows "$t \in abs_terms"
proof -
have "\neg is_Abs $f" when $fs$: "$f \in funs_term s" "s \in subterms_{set} (\emptyset \setminus fv t)" for $f$ $s$
proof -
obtain $T$ where $T$: "Fun $f$ $T \in subterms s" using funs_term_Fun_subterm[of $fs(1)$] by moura
hence "Fun $f$ $T \in subterms_{set} (\emptyset \setminus fv t)" using $fs(2)$ in_subterms_subset_Union by blast
thus $\text{thesis}$
  unfolding is_PubConstValue_def
by (cases $f$) auto
qed

thus $\text{thesis}$ using abs_terms_subst[of $assms(1)$] by force
qed

lemma pubval_terms_subst_range_disj:
"subst_range $\emptyset \cap pubval_terms = {} = \emptyset \setminus fv t \cap pubval_terms = {}"
proof (induction $t$
  case (Var $x$) thus $\text{case}$ by (cases "$x \in subst_domain \emptyset") auto
qed auto

lemma abs_terms_subst_range_disj:
"subst_range $\emptyset \cap abs_terms = {} = \emptyset \setminus fv t \cap abs_terms = {}"
proof (induction $t$
  case (Var $x$) thus $\text{case}$ by (cases "$x \in subst_domain \emptyset") auto
qed auto

lemma pubval_terms_subst_range_comp:
assumes "subst_range $\emptyset \cap pubval_terms = {}" 
shows "subst_range $\emptyset \cap pubval_terms = {}"
proof -
\{ fix $t$ $f$ assume $t$:
  "$t \in subst_range (\emptyset \cdot o_\delta)" "f \in funs_term t" "is_PubConstValue f"
  then obtain $x$ where $x$: "(\emptyset \cdot o_\delta) x = t" by auto
  have "\emptyset x \notin pubval_terms" using $assms(1)$ by (cases "$x \in subst_range \emptyset") force+
  hence "(\emptyset \cdot o_\delta) x \notin pubval_terms"
  unfolding pubval_terms_subst[of "\emptyset x" \delta] pubval_terms_subst_range_disj
  by (metis (mono_tags, lifting) subst_compose_def)
  hence False using $t(2,3)$ $x$ by blast
\} thus $\text{thesis}$ by fast
qed

lemma pubval_terms_subst_range_comp':
assumes "((\emptyset \cdot X) \cap pubval_terms = {}" 
shows "((\emptyset \cdot fv_{set} (\emptyset \cdot X)) \cap pubval_terms = {}"
proof -
\{ fix $t$ $f$ assume $t$:
  "$t \in (\emptyset \cdot o_\delta) \cdot X" "f \in funs_term t" "is_PubConstValue f"
  then obtain $x$ where $x$: "(\emptyset \cdot o_\delta) x = t" "x \in X" by auto
  have "\emptyset x \notin pubval_terms" using $assms(1)$ $x(2)$ by force
  moreover have "\emptyset \cdot fv (\emptyset x) \subseteq fv_{set} (\emptyset \cdot X)" using $x(2)$ by (auto simp add: fv_subset)
  hence "\emptyset \cdot fv (\emptyset x) \cap pubval_terms = {}" using $assms(2)$ by auto
\}
ultimately have "((a, a) x \notin \text{pubval_terms}"
using \text{pubval_terms_subst[of "(a) x" a]}
by (metis (mono_tags, lifting) subst-compose_def)
hence False using t(2,3) x by blast
} thus \?thesis by fast

lemma abs_terms_subst_range_comp:
assumes "\text{subst_range} \delta \cap \text{abs_terms} = \{\}" "\text{subst_range} \delta \cap \text{abs_terms} = \{\}"
shows "\text{subst_range} (\delta o \delta) \cap \text{abs_terms} = \{\}"
proof -
{ fix t f assume t: "t \in \text{subst_range} (\delta o \delta)" "f \in \text{funs_term} t" "\text{is_Abs} f"
then obtain x where x: "(\delta o \delta) x = t" by auto
have "\delta x \notin \text{abs_terms}" using assms(1) by (cases "\delta x \in \text{subst_range} \delta") force+
hence "(\delta o \delta) x \notin \text{abs_terms}"
using assms(2) abs_terms_subst[of "\delta x\delta"] abs_terms_subst_range_disj
by (metis (mono_tags, lifting) subst-compose_def)
hence False using t(2,3) x by blast
} thus \?thesis by fast

lemma abs_terms_subst_range_comp':
assumes "((\delta \ X) \cap \text{abs_terms} = \{\})" "((\delta \ X) \cap \text{abs_terms} = \{\})"
shows "(((\delta o \delta) \ X) \cap \text{abs_terms} = \{\})"
proof -
{ fix t f assume t: "t \in ((\delta o \delta) \ X)" "f \in \text{funs_term} t" "\text{is_Abs} f"
then obtain x where x: "(\delta o \delta) x = t" by auto
have "\delta x \notin \text{abs_terms}" using assms(1) x(2) by force
moreover have "fv (\delta x) \subseteq \text{fv_set} (\delta \ X)^*" using x(2) by (auto simp add: fv_subset)
hence "\delta \ X \notin \text{abs_terms}" using assms(2) by auto
ultimately have "((\delta o \delta) \ X) \notin \text{abs_terms}"
using abs_terms_subst[of "\delta x\delta"]
by (metis (mono_tags, lifting) subst-compose_def)
hence False using t(2,3) x by blast
} thus \?thesis by fast

context
begin
private lemma Ana_abs_aux1:
fixes \delta::"(('fun,'atom,'sets,'lbl) prot_fun, nat) term list"
and \alpha::"nat \Rightarrow \text{sets set}"
assumes "\text{Ana} f f = (K,T)"
shows "((K \cdot \text{list} \delta) \cdot \alpha\text{list} \alpha = K \cdot \text{list} (\lambda n. \delta n \cdot \alpha))"
proof -
{ fix k assume k: "k \in \text{set} K"
  hence "k \in \text{subterms_set} (\text{set} K)" by force
  hence "k \cdot \delta \cdot \alpha = k \cdot (\lambda n. \delta n \cdot \alpha)"
  proof (induction k)
  case (Fun g S)
  have "\forall s. s \in \text{set} S \Rightarrow s \cdot \delta \cdot \alpha = s \cdot (\lambda n. \delta n \cdot \alpha)"
  using Fun.IH in_subterms_subset_Union[OF Fun.prems] Fun_params_in_subterms[of _ S g]
  by (meson contra_subsetD)
  thus \?case using Ana_assml_alt[OF assms Fun.prems] by (cases g) auto
  qed simp
} thus \?thesis unfolding abs_apply_list_def by force

private lemma Ana_abs_aux2:
fixes \alpha::"nat \Rightarrow \text{sets set}"
and K::"(('fun,'atom,'sets,'lbl) prot_fun, nat) term list"
and M::"nat list"

3.6 Stateful Protocol Verification
3 Stateful Protocol Verification

and T::"('fun,'atom,'sets,'lbl) prot_term list"
assumes "∀i ∈ fvset (set K) ∪ set M. i < length T"
and "(K \{set (!) T \} \ α set α = K \{set (!) (λn. T ! n \ α)\})"
shows "(K \{set (!) T \} \ α set α = K \{set (!) (map (λs. s \ α) T)\})" (is "?A1 = ?A2")
and "(map (!) T) M \ α list α = map (!) (map (λs. s \ α) T) M" (is "?B1 = ?B2")
proof -
  have "T \ i \ α α = (map (λs. s \ α) T) \ i" when "i ∈ fvset (set K)" for i
using assms(1) by auto
hence "k \ (λi. T \ i \ α α) = k \ (λi. (map (λs. s \ α) T) \ i)" when "k ∈ set k" for k
using that term_subst_eq_conv[of k "λi. T \ i \ α α" "λi. (map (λs. s \ α) T) \ i"]
by auto
thus "?A1 = ?A2" using assms(2) by (force simp add: abs_apply_terms_def)

have "T \ i \ α α = map (λs. s \ α) T \ i" when "i ∈ set M" for i
using assms(1) by auto
thus "?B1 = ?B2" by (force simp add: abs_apply_list_def)
qed

private lemma Ana_abs_aux1_set:
  fixes δ::"('fun,'atom,'sets,'lbl) prot_fun, nat, ('fun,'atom,'sets,'lbl) prot_var\ gsubst"
  and α::"nat \ 'sets set"
assumes "Ana \ t \ = (K, T)"
shows "(set K \ set δ \ α set α = set K \ set (λn. δ \ n \ α))"
proof -
  { fix k assume "k ∈ set k"
    hence "k ∈ subterms_set (set K)" by force
    hence "k \ δ \ α α = k \ (λn. δ \ n \ α)"
    proof (induction k)
      case (Fun g S)
      have "∀s. s ∈ set S \→ s \ δ \ α α = s \ (λn. δ \ n \ α)"
        using Fun.IH in_subterms_subset_Union[of Fun.prems] Fun_param_in_subterms[of _ S g]
        by (meson contra_subsetD)
      thus ?case using Ana_abs_asm1_alt[of assms Fun.prems] by (cases g) auto
    qed
  } thus \thesis unfolding abs_apply_terms_def by force
qed

private lemma Ana_abs_aux2_set:
  fixes α::"nat \ 'sets set"
  and K::"('fun,'atom,'sets,'lbl) prot_fun, nat\ terms"
  and M::"nat set"
assumes "∀i ∈ fvset K \ M. i < length T"
  and "(K \{set (!) T \} \ α set α = K \{set (!) (map (λs. s \ α) T)\})" (is "?A1 = ?A2")
  and "(!) T \ M \ α list α = (!) (map (λs. s \ α) T) \ M" (is "?B1 = ?B2")
proof -
  have "T \ i \ α α = (map (λs. s \ α) T) \ i" when "i ∈ fvset K" for i
using assms(1) by auto
hence "k \ (λi. T \ i \ α α) = k \ (λi. (map (λs. s \ α) T) \ i)" when "k ∈ K" for k
using that term_subst_eq_conv[of k "λi. T \ i \ α α" "λi. (map (λs. s \ α) T) \ i"]
by auto
thus "?A1 = ?A2" using assms(2) by (force simp add: abs_apply_terms_def)

have "T \ i \ α α = map (λs. s \ α) T \ i" when "i ∈ M" for i
using assms(1) by auto
thus "?B1 = ?B2" by (force simp add: abs_apply_terms_def)
qed

lemma Ana_abs:
  fixes t::"('fun,'atom,'sets,'lbl) prot_term"
assumes "Ana t = (K, T)"
shows "Ana (t \ α) = (K \ α list α, T \ α list α)"

184
3.6 Stateful Protocol Verification

using assms
proof (induction t rule: Ana.induct)
case (1 f S)
  obtain K' T' where \*: "Ana f (K', T')" by moura
  show ?case using 1
proof (cases "arity f = length S ∧ arity f > 0")
  case True
  hence "K = K' · list (!) S" "T = map (!) (map (λs. s ∘ α) S)" "arity f > 0"
    using 1 by auto
  hence "K · αlist α = K' · list (map (λs. s · α) S)" "T · αlist α = map (!) (map (λs. s · α) S)"
    using Ana_assm2_alt[OF \*] Ana_abs_aux2[OF _ Ana_abs_aux1[OF \*], of T' S α]
    unfolding abs_apply_list_def by auto
moreover have "Fun (Fu f) S · α = Fun (Fu f) (map (λs. s · α) S)"
  by simp
ultimately show ?thesis using Ana_Fu_intro[OF \*] by metis
qed (auto simp add: abs_apply_list_def)
qed (simp_all add: abs_apply_list_def)
end

lemma deduct_FP_if_deduct:
  fixes M IK FP::"('fun,'atom,'sets,'lbl) prot_terms"
  assumes IK: "IK ⊆ GSMP M - (pubval_terms ∪ abs_terms)" 
             "∀ t ∈ IK. αset α. FP ⊢ c t"
  shows "FP ⊢ t · α"
proof -
  let ?P = "λf. ¬is_PubConstValue f"
  let ?GSMP = "GSMP M - (pubval_terms ∪ abs_terms)"
  have 1: "∀ m ∈ IK. m ∈ ?GSMP" using IK(1) by blast
  have 2: "∀ t t'. t ∈ ?GSMP −→ t' ⊑ t −→ t' ∈ ?GSMP"
    proof (intro allI impI)
      fix t t'
      assume t: "t ∈ ?GSMP" "t' ⊑ t"
      hence "t' ∈ GSMP M" using ground_subterm unfolding GSMP_def by auto
      moreover have "¬is_PubConstValue f" when "f ∈ funs_term t" using t(1) by auto
      with t(2) have "¬is_PubConstValue f" when "f ∈ funs_term t'" by auto
      moreover have "¬is_Abs f" when "f ∈ funs_term t" using t(1) by auto
      with t(2) have "¬is_Abs f" when "f ∈ funs_term t'" by auto
      ultimately show "t' ∈ ?GSMP" by simp
    qed
  have 3: "∀ t K T k. t ∈ ?GSMP −→ Ana t = (K, T) −→ k ∈ set K −→ k ∈ ?GSMP"
  proof (intro allI impI)
    fix t K T k assume t: "t ∈ ?GSMP" "Ana t = (K, T)" "k ∈ set K"
    hence "k ∈ GSMP M" using GSMP_Anakey by blast
    moreover have "∀ f ∈ funs_term t. ¬is_Abs f" using t(1) by auto
    with t(2,3) have "∀ f ∈ funs_term k. ¬is_Abs f" proof (induction t arbitrary: k rule: Ana.induct)
      case 1 thus ?case by (metis Ana_Fu_keys_not_pubval_terms surj_pair)
    qed auto
    moreover have "∀ f ∈ funs_term t. is_PubConstValue f" using t(1) by auto
    with t(2,3) have "∀ f ∈ funs_term k. is_PubConstValue f" proof (induction t arbitrary: k rule: Ana.induct)
      case 1 thus ?case by (metis Ana_Fu_keys_not_abs_terms surj_pair)
    qed auto
ultimately show "k ∈ ?GSMP" by simp
qed

have "(IK; M) ⊢_{GSMP} t"
  unfolding intruder_deduct_GSMP_def
  by (rule restricted_deduct_if_deduct'[OF 1 2 3 t])
thus ?thesis
proof (induction t rule: intruder_deduct_GSMP_induct)
  case (AxiomH t)
  show ?case using IK(2) abs_in[OF AxiomH.hyps] by force
next
  case (ComposeH T f)
  have *: "Fun f T · α = Fun f (map (λt. t · α) T)"
    using ComposeH.hyps(2,4)
    by (cases f) auto
  have **: "length (map (λt. t · α) T) = arity f"
    using ComposeH.hyps(1)
    by auto
  show ?case
    using intruder_deduct.Compose[OF ** ComposeH.hyps(2)] ComposeH.IH(1) *
    by auto
next
  case (DecomposeH t K T' t)
  have *: "Ana (t · α) = (K · αlist α, T' · αlist α)"
    using Ana_abs[OF DecomposeH.hyps(2)]
    by metis
  have **: "t · α ∈ set (T' · αlist α)"
    using DecomposeH.hyps(4) abs_in abs_list_set_is_set_abs_set[of T]
    by auto
  have ***: "FP ⊢ k"
    when k: "k ∈ set (K · αlist α)" for k
    proof -
      obtain k' where "k' ∈ set K" "k = k' · α"
        by (metis (no_types) k abs_apply_terms_def imageE abs_list_is_set_abs_set)
    show "FP ⊢ k"
      using DecomposeH.IH k' by blast
    qed
  qed
qed

end

3.6.2 Computing and Checking Term Implications and Messages

context stateful_protocol_model
begin
abbreviation (input) "absc s ≡ (Fun (Abs s) [])::('fun,'atom,'sets,'lbl) prot_term"
fun absdbupd where
  "absdbupd [] _ a = a"
| "absdbupd (insert(Var y, Fun (Set s) T)#D) x a = (if x = y then absdbupd D x (insert s a) else absdbupd D x a)"

186
3.6 Stateful Protocol Verification

lemma absdbupd_cons_cases:

```
"absdbupd (insert⟨Var x, Fun (Set s) T⟩#D) x d = absdbupd D x (insert s d)"
"absdbupd (delete⟨Var x, Fun (Set s) T⟩#D) x d = absdbupd D x (d - {s})"
"t ≠ Var x ∨ (∀s T. u = Fun (Set s) T) --- absdbupd (insert⟨t,u⟩#D) x d = absdbupd D x d"
"t ≠ Var x ∨ (∀s T. u = Fun (Set s) T) --- absdbupd (delete⟨t,u⟩#D) x d = absdbupd D x d"
```

proof -

assume (*): "t ≠ Var x ∨ (∀s T. u = Fun (Set s) T)"

let ?P = "absdbupd (insert⟨t,u⟩#D) x d = absdbupd D x d"

let ?Q = "absdbupd (delete⟨t,u⟩#D) x d = absdbupd D x d"

{ fix y f T assume "t = Fun f T ∨ u = Var y" hence ?P ?Q by auto }

moreover {

fix y f T assume "t = Var y" "u = Fun f T" hence ?P using * by (cases f) auto
}

moreover {

fix y f T assume "t = Var y" "u = Fun f T" hence ?Q using * by (cases f) auto
}

ultimately show ?P ?Q by (metis term.exhaust)+

qed simp_all

lemma absdbupd_filter: "absdbupd S x d = absdbupd (filter is_Update S) x d"

by (induction S x d rule: absdbupd.induct) simp_all

lemma absdbupd_append:

"absdbupd (A@B) x d = absdbupd B x (absdbupd A x d)"

proof (induction A arbitrary: d)

case (Cons a A) thus ?case

proof (cases a)

case (Insert t u)

then obtain s T where "t = Var x" "u = Fun (Set s) T" by moura

thus ?thesis by (simp add: Insert Cons.IH absdbupd_cons_cases(1))

qed (simp_all add: Cons.IH absdbupd_cons_cases(3))

next

case (Delete t u)

then obtain s T where "t = Var x" "u = Fun (Set s) T" by moura

thus ?thesis by (simp add: Delete Cons.IH absdbupd_cons_cases(2))

qed (simp_all add: Cons.IH absdbupd_cons_cases(4))

qed simp_all

qed simp

lemma absdbupd_wellformed_transaction:

assumes T: "wellformed_transaction T"

shows "absdbupd (unlabel (transaction_strand T)) = absdbupd (unlabel (transaction_updates T))"

proof -

define S0 where "S0 ≡ unlabel (transaction_strand T)"

define S1 where "S1 ≡ unlabel (transaction_receive T)"

define S2 where "S2 ≡ unlabel (transaction_checks T)"

define S3 where "S3 ≡ unlabel (transaction_updates T)"

define S4 where "S4 ≡ unlabel (transaction_send T)"

note S_defs = S0_def S1_def S2_def S3_def S4_def

have 0: "list_all is_Receive S1"

"list_all is_Check_or_Assignment S2"

"list_all is_Update S3"

"list_all is_Send S4"

using T unfolding wellformed_transaction_def S_defs by metis+
have "filter is_Update S1 = []" 
  "filter is_Update S2 = []" 
  "filter is_Update S3 = S3" 
  "filter is_Update S4 = []"
using list_all_filter_nil[of 0(1), of is_Update]
  list_all_filter_nil[of 0(2), of is_Update]
  list_all_filter_eq[of 0(3)]
  list_all_filter_nil[of 0(4), of is_Update]
by blast+
moreover have "S0 = S1@S2@S3@S4"
unfolding S_defs transaction_strand_def unlabel_def by auto
ultimately have "filter is_Update S0 = S3"
using filter_append[of is_Update] list_all_append[of is_Update]
by simp
thus ?thesis
using absdbupd_filter[of S0]
unfolding S_defs by presburger

definition fun abs_substs_set::
  "[('fun,'atom,'sets,'lbl) prot_var list,
    'sets set list,
    ('fun,'atom,'sets,'lbl) prot_var ⇒ 'sets set,
    ('fun,'atom,'sets,'lbl) prot_var ⇒ 'sets set,
    ('fun,'atom,'sets,'lbl) prot_var ⇒ 'sets set ⇒ bool]
⇒ ((('fun,'atom,'sets,'lbl) prot_var × 'sets set) list) list"
where
  "abs_substs_set [] _ _ _ _ = [[]]"
  "abs_substs_set (x#xs) as posconstrs negconstrs msgconstrs = 
    let bs = filter (λa. posconstrs x ⊆ a ∧ a ∩ negconstrs x = {}) as;
    ∆ = abs_substs_set xs as posconstrs negconstrs msgconstrs
    in concat (map (λb. map (λδ. (x, b)#δ) ∆) bs)"
definition fun abs_substs_fun::
  "[('fun,'atom,'sets,'lbl) prot_var × 'sets set) list,
      ('fun,'atom,'sets,'lbl) prot_var]
⇒ 'sets set"
where
  "abs_substs_fun δ x = (case find (λb. fst b = x) δ of Some (_,a) ⇒ a | None ⇒ {})"

lemmas abs_substs_set_induct = abs_substs_set.induct[case_names Nil Cons]

fun transaction_poschecks_comp::
  "[('fun,'atom,'sets,'lbl) prot_fun, ('fun,'atom,'sets,'lbl) prot_var) stateful_strand
  ⇒ ((('fun,'atom,'sets,'lbl) prot_var × 'sets set) list) list"
where
  "transaction_poschecks_comp [] = (λ_. {})"
  "transaction_poschecks_comp (⟨_: Var x ∈ Fun (Set s) []#T⟩#T) = ( 
      let f = transaction_poschecks_comp T in f(x := insert s (f x)))" 
  "transaction_poschecks_comp (_#T) = transaction_poschecks_comp T"

fun transaction_negchecks_comp::
  "[('fun,'atom,'sets,'lbl) prot_fun, ('fun,'atom,'sets,'lbl) prot_var) stateful_strand
  ⇒ ((('fun,'atom,'sets,'lbl) prot_var × 'sets set) list) list"
where
  "transaction_negchecks_comp [] = (λ_. {})"
  "transaction_negchecks_comp (⟨Var x not in Fun (Set s) []#T⟩#T) = ( 
      let f = transaction_negchecks_comp T in f(x := insert s (f x)))" 
  "transaction_negchecks_comp (_#T) = transaction_negchecks_comp T"
definition fun transaction_check_pre where
  "transaction_check_pre FPT T δ = " 
  let (FP, _, TI) = FPT;
  (FP, _, TI) = FPT;
C = set (unlabel (transaction_checks T));
xs = fv_list_set lookup (unlabel (transaction_strand T));
\emptyset = absenv x. if fat x = TAtom Value then (absc o \emptyset) x else Var x

\forall x \in set (transaction_fresh T). \emptyset x = \{\}\) \land
(\forall t \in transactions_send T). intruder_synch_mod_timpls FP TI (t \cdot \emptyset) \land
(\forall u \in C.
  (is_InSet u \implies (let x = the_elem_term u; s = the_set_term u
   in (is_Var x x \land is_Fun_Set s) \implies the_Set (the_Fun s) \in \delta (the_Var x))) \land
  ((is_NegChecks u \land bvars,sp, u = [] \land length (the_ins u) = 1) \implies (let x = first (hd (the_ins u)); s = snd (hd (the_ins u))
   in (is_Var x x \land is_Fun_Set s) \implies the_set (the_Fun s) \notin \delta (the_Var x))))

\text{definition transaction_check_post where}
\text{"transaction_check_post FPT T \delta \equiv}
\text{let (FP, \_, TI) = FPT;}
x = fv_list_set lookup (transaction_strand T);
\emptyset = absenv x. if fat x = TAtom Value then (absc o \emptyset) x else Var x;
u = absenv x. absupd (unlabel (transaction_updates T)) x (\emptyset x)
in (\forall x \in set xs - set (transaction_fresh T). \emptyset x \neq u \delta x \implies List.member TI (\delta x, u \delta x)) \land
(\forall t \in transactions_send T). intruder_synch_mod_timpls FP TI (t \cdot \emptyset (u \delta))

\text{definition fun_point_inter where "fun_point_inter f g \equiv \lambda x. f x \cap g x"}
\text{definition fun_point_union where "fun_point_union f g \equiv \lambda x. f x \cup g x"}
\text{definition fun_point_Inter where "fun_point_Inter fs \equiv \lambda x. \bigcap f \in fs. f x"}
\text{definition fun_point_Union where "fun_point_Union fs \equiv \lambda x. \bigcup f \in fs. f x"}
\text{definition fun_point_Inter_list where "fun_point_Inter_list fs \equiv \lambda x. \bigcap (set (map (\lambda f. f x) fs))"}
\text{definition fun_point_Union_list where "fun_point_Union_list fs \equiv \lambda x. \bigcup (set (map (\lambda f. f x) fs))"}
\text{definition ticl_abs where "ticl_abs TI a \equiv set (\#map \lambda p. \text{fat p = a} TI)"}
\text{definition ticl_abss where "ticl_abss TI as \equiv \bigcup a \in as. ticl_abs TI a"}
\text{lemma fun_point_Inter_set_eq:}
\text{"fun_point_Inter (set fs) = fun_point_Inter_list fs"}
\text{unfolding fun_point_inter_def fun_point_Inter_list_def by simp}
\text{lemma fun_point_Union_set_eq:}
\text{"fun_point_Union (set fs) = fun_point_Union_list fs"}
\text{unfolding fun_point_Union_def fun_point_Union_list_def by simp}
\text{lemma ticl_abs_refl_in: "x \in ticl_abs TI x"}
\text{unfolding ticl_abs_def by simp}
\text{lemma ticl_abs_iff:}
\text{assumes TI: "set TI = \{(a,b) \in (set TI)^+. a \neq b\}"
  shows "ticl_abs TI a = \{b. (a,b) \in (set TI)^+\}"
proof (intro order_antisym subsetI)
  fix x assume: "x \in \{b. (a,b) \in (set TI)^+\}"
  hence "{x = a \\lor (x \neq a \land (a,x) \in (set TI)^+)}" by (metis mem_Collect_eq rtranclD)
  moreover have "ticl_abs TI a = \{a\} \cup \{b. (a,b) \in set TI\}" unfolding ticl_abs_def by force
  ultimately show "x \in ticl_abs TI a" using TI by blast
qed
\text{lemma ticl_abs_Inters:}
\text{assumes xs: "\bigcap (\{ticl_abs TI \times xs\} \neq \{\})
and TI: "set TI = \{(a,b) \in (set TI)^+. a \neq b\}"
shows "\bigcap (\{ticl_abs TI \times \bigcap (\{ticl_abs TI \times xs\})\} \subseteq \bigcap (\{ticl_abs TI \times xs\})"
proof
  fix x assume: "x \in \bigcap (\{ticl_abs TI \times \bigcap (\{ticl_abs TI \times xs\})\})"
  have: "\bigcap (\{ticl_abs TI \times xs\}) = \{b. \forall a \in xs. (a,b) \in (set TI)^+\}"
  unfolding ticl_abs_iff[OF TI] by blast
  have "\{(b,x) \in (set TI)^+\}" when b: "\forall a \in xs. (a,b) \in (set TI)^+" for b
using x b unfolding ticl_abs_iff[OF TI] by blast

189
3 Stateful Protocol Verification

hence \((a,x) \in (\text{set TI})\) when \(a \in xs\) for a

using that \(xs\) \text{rtrancl.rtrancl_into_rtrancl[of a _ (\text{set TI})]} \text{by blast}

thus \(x \in \bigcap (\text{ticl_abs TI } \cdot \text{xs})\) unfolding \(*\) by blast

qed

function \((\text{sequential})\) \text{match_abss'} ::\("(\text{('a,'b,'c,'d) prot_fun, 'e) term} \Rightarrow
\((\text{('a,'b,'c,'d) prot_fun, 'e) term} \Rightarrow
\((\text{'e} \Rightarrow \text{'c set set}) \text{option}\)

where

\text{match_abss'} \text{Var x} \text{Fun (Abs a _) = Some ((λ_. \{\})(x := \{a\}))}

| match_abss' \text{Fun f ts} \text{Fun g ss} = ( \text{if f = g ∧ length ts = length ss}
then \text{map_option fun_point_Union_list (those (map2 match_abss' ts ss))}
else None)

| match_abss' _ _ = None

by \text{pat_completeness auto}

termination

proof -

let \?m = "measures [size o fst]"

have 0: \"\text{wf ?m}\" by \text{simp}

show ?thesis

apply (standard, use 0 in fast)

by \text{metis (no_types) comp_def fst_conv measures_less Fun_zip_size_lt(1)}

qed

definition \text{match_abss where}

\text{match_abss OCC TI t s} ≡ ( \text{let xs = fv t;}
\text{OCC' = set OCC;}
\text{f = λδ x. if x ∈ xs then δ x else OCC';}
\text{g = λδ x. \bigcap (\text{ticl_abs TI } \cdot \text{δ x})}
\text{in case match_abss' t s of}
\text{Some δ ⇒}
\text{let δ' = g δ}
\text{in if \(∀ x \in xs. \delta' x \neq \{\} \text{then Some (f δ') else None}}

| \text{None ⇒ None)"

lemma \text{match_abss'_Var_inv:}

assumes \(\delta\): \text{match_abss' \text{(Var x) Fun (Abs a _) = Some ((λ_. \{\})(x := \{a\}))}}\)

shows \(∃ a ts. t = \text{Fun (Abs a) ts ∧ \delta = (λ_. \{\})(x := \{a\})}\)

proof -

obtain \(f ts\) where \(t = \text{Fun f ts}\) using \(\delta\) by (cases t) auto
then obtain \(a\) where \(f = \text{Abs a}\) using \(\delta\) by (cases f) auto

show ?thesis using \(\delta\) unfolding \(t\ a\) by \text{simp}

qed

lemma \text{match_abss'_Fun_inv:}

assumes \(\text{match_abss' (Fun f ts) Fun g ss = Some \delta}\)

shows \(f = g\) (is \(?A\))

and \(\text{length ts = length ss}\) (is \(?B\))

and \(∃ \upsilon. \text{Some \upsilon = those (map2 match_abss' ts ss) ∧ \delta = fun_point_Union_list \upsilon}\) (is \(?C\))

and \(∀ (t,s) ∈ \text{set (zip ts ss)}. \text{∃} σ. \text{match_abss' t s} = \text{Some} σ\) (is \(?D\))

proof -

note 0 = \text{assms match_abss'.simp(2)[of f ts g ss] option.distinct(1)}

show \(?A\) by \text{metis 0}

show \(?B\) by \text{metis 0}

show \(?C\) by \text{metis (no_types, opaque_lifting) 0 map_option_eq_Some}

thus \(?D\) using \text{map2_those_Some_case[of match_abss' ts ss]} by \text{fastforce}

qed
3.6 Stateful Protocol Verification

lemma match_abss'_FunI:
assumes \( \Delta : \forall i. i < \text{length } T \implies \text{match_abss'} (U ! i) (T ! i) = \text{Some } (\Delta i) \)
and \( T : \text{length } T = \text{length } U \)
shows \( \text{match_abss'} (\text{Fun } f U) (\text{Fun } f T) = \text{Some } (\text{fun_point}_\bigcup (\text{map } \Delta [0..<\text{length } T])) \)
proof -
have \( \text{match_abss'} (\text{Fun } f U) (\text{Fun } f T) = \text{map_option } \text{fun_point}_\bigcup (\text{map2 } \text{match_abss'} U T) \)
using \( T \) match_abss'.simps(2)[of \( f U f T \)] by presburger
moreover have \( \text{those } (\text{map2 } \text{match_abss'} U T) = \text{Some } (\text{map } \Delta [0..<\text{length } T]) \)
using \( \Delta \) \( T \) those_map2_SomeI by metis
ultimately show ?thesis by simp
qed

lemma match_abss'_Fun_param_subset:
assumes \( \text{match_abss'} (\text{Fun } f ts) (\text{Fun } g ss) = \text{Some } \delta \)
and \( (t,s) \in \text{set } (\text{zip } ts ss) \)
and \( \text{match_abss'} t s = \text{Some } \sigma \)
shows \( \sigma x \subseteq \delta x \)
proof -
obtain \( \vartheta \) where \( \vartheta \):
\( \text{those } (\text{map2 } \text{match_abss'} ts ss) = \text{Some } \vartheta \)
\( \delta = \text{fun_point}_\bigcup \vartheta \)
using \( \text{match_abss'}_\text{Fun_inv}[OF assms(1)] \) by metis
have \( \sigma \in \text{set } \vartheta \) using \( \vartheta(1) \) assms(2-) those_Some_iff[of "map2 match_abss' ts ss" \( \vartheta \)] by force
thus ?thesis using \( \vartheta(2) \) unfolding fun_point_Union_list_def by auto
qed

lemma match_abss'_fv_is_nonempty:
assumes \( \text{match_abss'} t s = \text{Some } \delta \)
and \( x \in \text{fv } t \)
shows \( \delta x \neq {} \) (is \( \"?P \delta \)"
using assms
proof (induction \( t s \) arbitrary: \( \delta \) rule: match_abss'.induct)
case \( (2 f ts g ss) \)
note prems = \( 2.\text{prems} \)
note IH = \( 2.\text{IH} \)
have \( 0: \forall (t,s) \in \text{set } (\text{zip } ts ss). \exists \sigma. \text{match_abss'} t s = \text{Some } \sigma " f = g " \text{length ts = length ss} " \)
using \( \text{match_abss'}_\text{Fun_inv}[OF prems(1)] \) by simp_all
obtain \( t \) where \( t : \text{t \in set ts} " x \in \text{fv t} \) using \( \text{prems(2)} \) by auto
then obtain \( s \) where \( s : \text{s \in set ss} " (t,s) \in \text{set } (\text{zip } ts ss) " \)
by (meson 0(3) in_set_impl_in_set_zip1 in_set_zipE)
then obtain \( \sigma \) where \( \sigma : \text{match_abss'} t s = \text{Some } \sigma " \text{using } 0(1) \) by fast
show \?case
using IH[OF conjI[OF 0(2,3) s(2) _ \( \sigma \) t(2) match_abss'_Fun_param_subset[OF prems(1) s(2) \( \sigma \)]] by auto
qed auto

lemma match_abss'_nonempty_is_fv:
fixes \( s t :: \(((\text{a, b, c, d}) \text{ prot_fun, } \text{v}) \text{ term} \)
assumes \( \text{match_abss'} s t = \text{Some } \delta \)
and \( \delta x \neq {} \)
shows \( x \in \text{fv } s \)
using assms
proof (induction \( s t \) arbitrary: \( \delta \) rule: match_abss'.induct)
case \( (2 f ts g ss) \)
note prems = \( 2.\text{prems} \)
note IH = \( 2.\text{IH} \)
3 Stateful Protocol Verification

obtain \( \vartheta \) where \( \vartheta \): “Some \( \vartheta = \text{those (map2 match\_abss' ts ss)} \)” “\( \delta = \text{fun\_point\_Union\_list} \vartheta \)” and \( fg \): “f = g” “length ts = length ss” using match\_abss’\_Fun_inv[OF prems(1)] by fast

have “\( \exists \sigma \in \text{set} \vartheta . \sigma \cdot x \neq {} \)” using \( fg(2) \) prems \( \vartheta \) unfolding fun\_point\_Union\_list_def by auto

then obtain t’ s’ \( \sigma \) where t’s’: “(t’,s’) \in \text{set (zip ts ss)}” “\text{match\_abss'} t' s' = Some \sigma” “\sigma \cdot x \neq {}” using those_map2\_Somedef[OF \( \vartheta \) (1)[symmetric]] by blast

show ?case using ts'(3) IH[OF conjI[OF fg] ts'(1) _ ts'(2)] set\_zip\_leftD[OF ts'(1)] by force
qed auto

lemma match\_abss’\_Abs\_in\_funs\_term:
fixes s t::“(('a,'b,'c,'d) prot_func, 'v) term”
assumes “match\_abss' s t = Some \delta” and “a \in \delta \cdot x”
shows “Abs a \in \text{funs\_term} t” using assms proof (induction s t arbitrary: a \( \delta \) rule: match\_abss’.induct)

next

have \( \vartheta \) where \( \vartheta \): “Some \( \vartheta = \text{those (map2 match\_abss' ts ss)} \)” “\( \delta = \text{fun\_point\_Union\_list} \vartheta \)” and \( fg \): “f = g” “length ts = length ss” using match\_abss’\_Fun_inv[OF prems(1)] by fast

have 0: “\( \exists \sigma \in \text{set} \vartheta . \sigma \cdot x \neq {} \)” using \( \vartheta \)(1) map2_map_subst unfolding fg(3) by metis

have 1: “\( \forall t \in \text{set ts} . \exists \sigma . \text{match\_abss'} t \cdot t = \text{Some} \sigma” using ts zip\_map\_subst[of ts \( \delta \)] unfolding fg(3) by simp
3.6 Stateful Protocol Verification

have 2: "σ' ∈ set ϑ"
  when t: "t ∈ set ts" "match_abss' t (t · δ) = Some σ'" for t σ'
  using t 0 those_Some_iff[of "map (λt. match_abss' t (t · δ)) ts" ϑ] by force

have 3: "?P t σ'" "σ' x ≠ {}"
  when t: "t ∈ set ts" "x ∈ fv t" "match_abss' t (t · δ) = Some σ'" for t σ' x
  using t hyps(1)[OF conjI[OF fg(1,2)], of "(t, t · δ)" t σ'] zip_map_subst[of ts δ]
  unfolding fg(3) by auto

have 4: "σ' x = {}"
  when t: "x /∈ fv t" "match_abss' t (t · δ) = Some σ'" for t σ' x
  by (meson t match_abss'_nonempty_is_fv)

show ?case
proof
  fix x assume "x ∈ fv (Fun f ts)"
  then obtain t σ' where t: "t ∈ set ts" "x ∈ fv t" and σ': "match_abss' t (t · δ) = Some σ'"
    using 1 by auto
  then obtain a tsa where a: "δ x = Fun (Abs a) tsa"
    using 3[OF t σ'] by fast
  have "σ'' x = {a} ∨ σ'' x = {}"
    when "σ'' ∈ set ϑ" for σ''
    using that a 0 3[of _ x] 4[of x]
    unfolding those_Some_iff by fastforce
  thus "∃ a ts. δ x = Fun (Abs a) ts ∧ σ x = {a}" using a 2[OF t(1) σ'] 3[OF t σ']
    unfolding ϑ(2) fun_point_Union_list_def by auto
qed

qed auto

lemma match_abss'_subst_disj_nonempty:
  assumes TI: "set TI = {(a,b) ∈ (set TI). a ≠ b}"
  and "match_abss' s (s · δ) = Some σ"
  and "x ∈ fv s"
  shows "∀ (t,s) ∈ set (zip ts ss). ∃ σ. match_abss' t s = Some σ" (is "?P σ")
  using match_abss'_Fun_inv[OF assms(1)[unfolded hyps(2)[symmetric]]] hyps(2)
  unfolding tiicl_abs_def by force

next
  case (2 f ts g ss)
  note prem = "2.prems"
  note hyps = "2.hyps"

obtain ϑ where ϑ: "Some ϑ = those (map2 match_abss' ts ss)" "σ = fun_point_Union_list ϑ"
  and fg: "f = g" "length ts = length ss" "ss = ts · list δ"
  and ts: "∀ t,s ∈ set (zip ts ss). ∃ σ. match_abss' t s = Some σ"
  using match_abss'_Fun_inv[OF prem(1)[unfolded hyps(2)[symmetric]]] hyps(2)
  unfolding tiicl_abs_def by force

define ts' where "ts' ≡ filter (λt. x ∈ fv t) ts"
define ϑ' where "ϑ' ≡ map (λt. the (match_abss' t (t · δ))) ts"
define ϑ'' where "ϑ'' ≡ map (λt. the (match_abss' t (t · δ))) ts'"

have 0: "those (map (λt. match_abss' t (t · δ)) ts) = Some ϑ''"
  using ϑ(1) map2_map_subst unfolding fg(3) by metis

have 1: "ϑ t ∈ set ts. ∃ σ. match_abss' t (t · δ) = Some σ" using ts zip_map_subst[of ts δ]
  unfolding fg(3) by simp

have ts_not_nil: "ts ≠ []" using prem(2) by fastforce
hence "∃ t ∈ set ts. x ∈ fv t" using prem(2) by simp
then obtain a tsa where a: "δ x = Fun (Abs a) tsa"
using 1 match_abss'_subst_fv_ex_abs[OF _ TI, of _ δ]
bymetis
hence a': "σ' x = {a}"
when "t ∈ set ts" "x ∈ fv t" "match_abss' t (t · δ) = Some σ''" for t σ'
using that match_abss'_subst_fv_ex_abs[OF _ TI, of _ δ]
by fastforce
have "ts' ≠ []" using prems(2) unfolding ts'_def by (simp add: filter_empty_conv)
hence ϑ''_not_nil: "ϑ'' ≠ []" unfolding ϑ''_def by simp
have 2: "σ' ∈ set ϑ" when t: "t ∈ set ts" "match_abss' t (t · δ) = Some σ''" for t σ'
using 0 those_Some_iff[of "map (λt. match_abss' t (t · δ)) ts" ϑ] by force
have 3: "P σ'' "σ' x ≠ {}"
when t: "t ∈ set ts" "match_abss' t (t · δ) = Some σ''" for t σ'
using t hyps(1)[OF conjI[OF fg(1,2)], of "(t, t · δ) t σ'] zip_map_subst[of ts δ]
match_abss' fv_is_nonempty[of t t · δ σ' x]
unfolding fg(3) ts'_def by (force, force)
have 4: "σ' x = {}" when t: "x ∉ fv t" "match_abss' t (t · δ) = Some σ''" for t σ'
by (meson t match_abss'_nonempty_is_fv)
have 5: "ϑ = map and ϑ''" using 0 1 unfolding ϑ''_def by (induct ts arbitrary: ϑ) auto
have "fun_point_Union_list (map and ϑ'') x =
  fun_point_Union_list (map and (filter (λ(_,). x ∈ fv t) ϑ'')) x" using 1 4 unfolding ϑ''_def fun_point_Union_list_def by fastforce
hence 6: "fun_point_Union_list ϑ'' x = fun_point_Union_list ϑ'' x" using 0 1 4 unfolding 5 ϑ''_def ϑ''_def fun_point_Union_list_def ts'_def by auto
have 7: "P ϑ'' "σ' x ≠ {}"
when σ': "σ' ∈ set ϑ''" for σ'
using that 1 3 unfolding ϑ''_def ts'_def by auto
have "σ' x = {a}"
when σ': "σ' ∈ set ϑ''" for σ'
using σ' a' 1 unfolding ϑ''_def ts'_def by fastforce
hence "fun_point_Union_list ϑ'' x = {b | b σ'. σ' ∈ set ϑ'' ∧ b ∈ {a}}"
using ϑ''_not_nil unfolding fun_point_Union_list_def by auto
hence 8: "fun_point_Union_list ϑ'' x = {a}"
using ϑ''_not_nil by auto
show ?case
using 8 a
unfolding ϑ(2) 6 ticl_abs_iff[OF TI] by auto
qed simp_all

lemma match_abssD:
fixes OCC TI σ
defines "f ≡ (λδ x. if x ∈ fv s then δ x else set OCC)"
and "g ≡ (λδ x. set OCC − δ x)"
assumes δ: "match_abss OCC TI a t = Some δ" shows "∃δ'. match_abss s t = Some δ ∧ δ' = f (g δ) ∧ (∀x ∈ fv s. δ x ≠ {}) ∧ f (g δ) x ≠ {})) ∧ (set OCC ≠ {}) → (∀x. f (g δ) x ≠ {}))"
proof -
obtain δ where δ: "match_abss' s t = Some δ"
using δ' unfolding match_abss_def by force
hence "Some δ' = (if (∀x ∈ fv s. g δ x ≠ {}) then Some (f (g δ)) else None)"
using δ' unfolding match_abss_def f_def g_def Let_def by simp
3.6 Stateful Protocol Verification

hence "δ' = f (g δ)" "∀x ∈ fv s. δ x ≠ {} ∧ f (g δ) x ≠ {}"
by (metis (no_types, lifting) option.inject option.distinct(1),
    metis (no_types, lifting) f_def option.distinct(1) match_abss'_fv_is_nonempty[OF δ])
thus ?thesis using δ unfolding f_def by force
qed

lemma match_abss_ticl_abs_Inter_subset:
  assumes TI: "set TI = {(a,b). (a,b) ∈ (set TI) + ∧ a ≠ b}"
  and δ: "match_abss OCC TI s t = Some δ'
  and x: "x ∈ fv s"
  shows "⋂ (ticl_abs TI ` δ x) ⊆ δ x"
proof -
  let ?h1 = "λδ x. if x ∈ fv s then δ x else set OCC"
  let ?h2 = "λδ x. ⋂ (ticl_abs TI ` δ x)"
  obtain δ' where δ': "match_abss' s t = Some δ'" "∀x ∈ fv s. δ' x ≠ {} ∧ δ x ≠ {}"
    using match_abssD[OF δ'] unfolding f_def g_def
  thus ?thesis
  using ticl_abs_Inter TI by simp
qed

lemma match_abss_fv_has_abs:
  assumes "match_abss OCC TI s t = Some δ"
  and "x ∈ fv s"
  shows "δ x ≠ {}"
using assms match_abssD by fast

lemma match_abss_OCC_if_not_fv:
  fixes s t::"(('a,'b,'c,'d) prot_fun, 'v) term"
  assumes δ': "match_abss OCC TI s t = Some δ'
  and x: "δ' x ≠ {}" "x /∈ fv s"
  shows "δ' x = set OCC"
proof -
  define f where "f ≡ λs::(('a,'b,'c,'d) prot_fun, 'v) term. λδ x. if x ∈ fv s then δ x else set OCC"
  define g where "g ≡ λδ. λx::'v. ⋂ (ticl_abs TI ` δ x)"
  obtain δ where "match_abss' s t = Some δ" "δ' = f s (g δ)"
    using match_abssD[OF δ'] unfolding f_def g_def by blast
  thus ?thesis
  using x match_abss'_nonempty_is_fv unfolding f_def by presburger
qed

inductive synth_abs_substs_constrs_rel for FP OCC TI where
  SolveNil:
  "synth_abs_substs_constrs_rel FP OCC TI [] (λ_. set OCC)"
| SolveCons:
  "ts ≠ [] ⇒ ∀t ∈ set ts. synth_abs_substs_constrs_rel FP OCC TI [t] (λ t)
  ⇒ synth_abs_substs_constrs_rel FP OCC TI ts (fun_point_Inter (λ ` set ts))"
| SolvePubConst:
  "arity c = 0 ⇒ public c
  ⇒ synth_abs_substs_constrs_rel FP OCC TI [Fun c []] (λ_. set OCC)"
| SolvePrivConstIn:
  "arity c = 0 ⇒ ¬public c ⇒ Fun c [] ∈ set FP
  ⇒ synth_abs_substs_constrs_rel FP OCC TI [Fun c []] (λ_. set OCC)"
| SolvePrivConstNotIn:
  "arity c = 0 ⇒ ¬public c ⇒ Fun c [] /∈ set FP
  ⇒ synth_abs_substs_constrs_rel FP OCC TI [Fun c []] (λ_. { })"
3 Stateful Protocol Verification

| SolveValueVar: |
| "ϑ = ((λ_. set OCC)(x := ticl_abs abs TI (a ∈ set OCC. (a)abs ∈ set FP)) (a ∈ set OCC)) | ⇒ synth_abs_substs_constrs_rel FP OCC TI [Var x] ϑ"

| SolveComposed: |
| "arity f > 0 ⇒ length ts = arity f |
| ⇒ ∀δ. δ ∈ Δ →→ (δs ∈ set FP. match_abs OCC TI (Fun f ts) s = Some δ) |
| ⇒ Θ = (λδ x. if δ x ≠ {} then δ x else set OCC) |
| ⇒ Θ1 = fun_point_Union (Θ ` Δ) |
| ⇒ synth_abs_substs_constrs_rel FP OCC TI ts ϑ2 |
| ⇒ synth_abs_substs_constrs_rel FP OCC TI [Fun f ts] (fun_point_union Θ1 Θ2)"

fun synth_abs_substs_constrs_aux where |
| "synth_abs_substs_constrs_aux FP OCC TI (Var x) = (λ_. set OCC)(x := ticl_abs abs TI (set (filter (λa. ⟨a⟩ absorbs ∈ set FP) OCC))))" |
| "synth_abs_substs_constrs_aux FP OCC TI (Fun f ts) = ( |
| if ts = [] |
| then if ¬public f ∧ Fun f ts ∉ set FP then (λ_. {}) else (λ_. set OCC) |
| else let Δ = map the (filter (λδ. δ ≠ None) (map (match_abs OCC TI (Fun f ts)) FP)); |
| Θ = (λδ x. let as = δ x in if as ≠ {} then as else set OCC; |
| Θ1 = fun_point_Union_list (map Θ Δ); |
| Θ2 = fun_point_Inter_list (map (synth_abs_substs_constrs_aux FP OCC TI) ts) |
| in fun_point_union Θ1 Θ2)"

definition synth_abs_substs_constrs where |
| "synth_abs_substs_constrs FPT T ≡ |
| let (FP, OCC, TI) = FPT; |
| ts = trms_listsst (unlabel (transaction_receive T)); |
| f = fun_point_Inter_list ◦ map (synth_abs_substs_constrs_aux FP OCC TI) |
| in if ts = [] then (λ_. set OCC) else f ts"

definition transaction_check_comp:: |
| "[((fun,'atom,'sets,'lbl) prot_var ⇒ 'sets set ⇒ bool, |
| ('fun,'atom,'sets,'lbl) prot_term list × |
| 'sets set list × ('sets set × 'sets set) list, |
| ('fun,'atom,'sets,'lbl) prot_transaction] |
| ⇒ (((fun,'atom,'sets,'lbl) prot_var × 'sets set list) list) list"

where |
| "transaction_check_comp msgcs FPT T ≡ |
| let (_, OCC, _) = FPT; |
| S = unlabel (transaction_strand T); |
| C = unlabel (transaction_checks T); |
| xs = filter (λx. x ∈ set (transaction_fresh T) ∧ fst x = TAtom Value) (fv_listsst S); |
| posconstrs = transaction_poschecks_comp C; |
| negconstrs = transaction_negchecks_comp C; |
| pre_check = transaction_check_pre FPT T; |
| Δ = abs_substs_set xs OCC posconstrs negconstrs msgcs |
| in filter (λδ. pre_check (abs_substs_fun δ)) Δ" |

definition transaction_check:: |
| "[((fun,'atom,'sets,'lbl) prot_var ⇒ 'sets set ⇒ bool, |
| ('fun,'atom,'sets,'lbl) prot_term list × |
| 'sets set list × ('sets set × 'sets set) list, |
| ('fun,'atom,'sets,'lbl) prot_transaction] |
| ⇒ bool"

where |
| "transaction_check' msgcs FPT T ≡ |
| list_all (λδ. transaction_check_post FPT T (abs_substs_fun δ)) |
| (transaction_check_comp msgcs FPT T)*"
3.6 Stateful Protocol Verification

**definition** transaction_check::

\[ \text{[('fun,'atom,'sets,'lbl) prot_term list } \times \]
\[ \text{ 'sets set list } \times \]
\[ \text{ ( 'sets set } \times \text{ 'sets set list,} \]
\[ \text{ ( 'fun,'atom,'sets,'lbl) prot_transaction]} \]
\[ \Rightarrow \text{ bool} \]

where

\[ \text{transaction_check } \equiv \text{ transaction_check' } (\lambda_ _. \text{ True}) \]

**definition** transaction_check_alt1::

\[ \text{[('fun,'atom,'sets,'lbl) prot_term list } \times \]
\[ \text{ 'sets set list } \times \]
\[ \text{ ( 'sets set } \times \text{ 'sets set list,} \]
\[ \text{ ( 'fun,'atom,'sets,'lbl) prot_transaction]} \]
\[ \Rightarrow \text{ bool} \]

where

\[ \text{transaction_check_alt1 } FPT \ T \equiv \]
\[ \text{let } \text{msgcs } = \text{ synth_abs_substs_constrs } FPT \ T \]
\[ \text{in } \text{transaction_check'} \ (\lambda \ x \ a. \ a \in \text{msgcs } x) \ FPT \ T \]

**lemma** abs_subst_fun_cons:

\[ \text{"abs_substs_fun } ((x,b)#\delta) = (\text{abs_substs_fun } \delta)(x := b)" \]

**unfolding** abs_substs_fun_def by fastforce

**lemma** abs_substs_cons:

assumes \[ \text{"}\delta \in \text{set } (\text{abs_substs_set } xs \text{ as poss negs msgcs)}" \]
\[ \text{"b } \in \text{ set as}" \]
\[ \text{"poss } x \subseteq b" \]
\[ \text{"b } \cap \text{ negs } x = \{\}" \]
\[ \text{"msgcs } x \ b" \]

shows \[ \text{"}(x,b)#\delta \in \text{set } (\text{abs_substs_set } (x#xs) \text{ as poss negs msgcs)}" \]

using assms by auto

**lemma** abs_substs_cons':

assumes \[ \text{\"}\delta \in \text{abs_substs_fun ' set } (\text{abs_substs_set } xs \text{ as poss negs msgcs)}\"
\[ \text{\"and b: "} \]
\[ \text{\"b } \in \text{ set as}" \]
\[ \text{\"poss } x \subseteq b" \]
\[ \text{\"b } \cap \text{ negs } x = \{\}" \]
\[ \text{\"msgcs } x \ b" \]

shows \[ \text{\"}(x,b)#\delta \in \text{set } (\text{abs_substs_set } (x#xs) \text{ as poss negs msgcs)}\"

proof -

obtain \[ \text{\"}\vartheta \text{ where } \delta = \text{abs_substs_fun } \vartheta\"
\[ \text{\"\vartheta } \in \text{ set } (\text{abs_substs_set } xs \text{ as poss negs msgcs)}\"

using \[ \text{\"}\delta\text{ by moura}\]

have \[ \text{\"abs_substs_fun } ((x,b)#\vartheta) \in \text{abs_substs_fun ' set } (\text{abs_substs_set } (x#xs) \text{ as poss negs msgcs)}\"

using abs_substs_cons[OF \[ \text{\"}\vartheta\text{ (2) b}\]

by blast

thus \[ \text{\"?thesis}\]

using \[ \text{\"}\vartheta(1) \text{ abs_substs_fun_cons[of x b } \vartheta\text{] by argo}\]

qed

**lemma** abs_substs_has_abs:

assumes \[ \text{\"}\forall x. \ x \in \text{set } xs \rightarrow \delta x \in \text{ set as}" \]
\[ \text{\"and } \]
\[ \text{\"}\forall x. \ x \in \text{set } xs \rightarrow \text{poss } x \subseteq \delta x" \]
\[ \text{\"and } \]
\[ \text{\"}\forall x. \ x \in \text{set } xs \rightarrow \text{msgcs } x \ (\delta x)" \]
\[ \text{\"and } \]
\[ \text{\"}\forall x. \ x /\in \text{ set } xs \rightarrow \delta x = \{\}" \]

shows \[ \text{\"}\delta \in \text{abs_substs_fun ' set } (\text{abs_substs_set } xs \text{ as poss negs msgcs)}\"

using assms

proof (induction xs arbitrary: \[ \text{\"}\delta\])

case (Cons x xs)

define \[ \text{\"}\vartheta \text{ where } \delta \equiv \lambda y. \text{ if } y \in \text{ set } xs \text{ then } \delta y \text{ else } \{\}\"

have \[ \text{\"}\vartheta \in \text{abs_substs_fun ' set } (\text{abs_substs_set } xs \text{ as poss negs msgcs)}\"

using Cons.prems Cons.IH by (simp add: \[ \text{\"}\vartheta_def\]

moreover have \[ \text{\"}\delta x \in \text{ set as}" \]
\[ \text{\"\delta x } \subseteq \text{ \delta x" \"\delta x } \cap \text{ negs } x = \{\}" \]
\[ \text{\"msgcs } x \ (\delta x)" \]

by (simp_all add: Cons.prems(1,2,3,4))

ultimately have \[ \text{\"}\vartheta(x := \delta x) \in \text{abs_substs_fun ' set } (\text{abs_substs_set } (x#xs) \text{ as poss negs msgcs)}\"

by (metis abs_substs_cons)

have \[ \text{\"}\delta = \vartheta(x := \delta x)" \]

qed
proof
fix y show "\( \delta \ y = (\delta (x := \delta \ x)) \ y \)"
proof (cases "y \in\ set (x#xs)"
  case False thus \(?thesis\) using Cons.prems(5) by (fastforce simp add: \( \delta \_\ def \) )
  qed
thus \(?case\) by (metis 0)
  qed (auto simp add: abs_substs_fun_def)
lemma abs_substs_abss_bounded:
  assumes "\( \delta \in\ abs_substs_fun \ \cdot\ \set (abs_substs_set \ xs \ as \ poss \ negs \ msgcs) \)"
  and "\( x \in\ set \ xs \)"
  shows "\( \delta \ \set (abs_substs_set \ xs \ as \ negs) \)"
    and "\( x \ \set (\delta \ x) \)"
    and "\( x \ \set negs \ x = \{\} \)"
    and "\( x \ \set msgcs \ x \ (\delta \ x) \)"
  using assms
proof (induct \( \delta \) rule: abs_substs_set_induct)
  case (Cons y xs as poss negs msgcs)
  { case 1
    thus \(?case\) using Cons.hyps(1)
    unfolding abs_substs_fun_def
    by fastforce
  }{ case 2
    thus \(?case\) using Cons.hyps(2)
    proof (cases "x = y"
      case False
      then obtain \( \delta' \) where
        "\( \delta' \in\ abs_substs_fun \ \cdot\ \set (abs_substs_set \ xs \ as \ poss \ negs \ msgcs) \)"
        "\( \delta' \ x = \delta x \)"
      using 2 unfolding abs_substs_fun_def by force
      moreover have "\( x \in\ set \ xs \)" using 2(2) False by simp
      moreover have "\( \exists b. b \in\ set \ as \ \set poss \ y \subseteq\ b \land b \ \set negs \ y = \{\} \)"
      using 2 False by auto
      ultimately show \(?thesis\) using Cons.hyps(3) by fastforce
    qed (auto simp add: abs_substs_fun_def)
  }{ case 3
    thus \(?case\) using Cons.hyps(3)
    proof (cases "x = y"
      case False
      then obtain \( \delta' \) where
        "\( \delta' \in\ abs_substs_fun \ \cdot\ \set (abs_substs_set \ xs \ as \ poss \ negs \ msgcs) \)"
        "\( \delta' \ x = \delta \)"
      using 3 unfolding abs_substs_fun_def by force
      moreover have "\( x \in\ set \ xs \)" using 3(2) False by simp
      moreover have "\( \exists b. b \in\ set \ as \ \set poss \ y \subseteq\ b \land b \ \set negs \ y = \{\} \)"
      using 3 False by auto
      ultimately show \(?thesis\) using Cons.hyps(4) by fastforce
    qed (auto simp add: abs_substs_fun_def)
  }{ case 4
    thus \(?case\) using Cons.hyps(4)
    proof (cases "x = y"
      case False
      then obtain \( \delta' \) where
        "\( \delta' \in\ abs_substs_fun \ \cdot\ \set (abs_substs_set \ xs \ as \ poss \ negs \ msgcs) \)"
        "\( \delta' \ x = \delta \)"
      using 4 unfolding abs_substs_fun_def by force
      moreover have "\( x \in\ set \ xs \)" using 4(2) False by simp
      moreover have "\( \exists b. b \in\ set \ as \ \set poss \ y \subseteq\ b \land b \ \set negs \ y = \{\} \)"
      using 4 False by auto
      ultimately show \(?thesis\) using Cons.hyps(5) by fastforce
    qed (auto simp add: abs_substs_fun_def)
  }
  qed (simp_all add: abs_substs_fun_def)
lemma abs_substs_abss_bounded':
  assumes "\( \delta \in\ abs_substs_fun \ \cdot\ \set (abs_substs_set \ xs \ as \ poss \ negs \ msgcs) \)"
3.6 Stateful Protocol Verification

and "x \notin set xs"
shows "\delta x = {}"
using assms unfolding abs_substs_fun_def
by (induct xs as poss negs msgcs arbitrary: \delta rule: abs_substs_set_induct) (force, fastforce)

lemma transaction_poschecks_comp_unfold:
"transaction_poschecks_comp C x = \{s. \exists a. \langle a: Var x \in Fun (Set s) [] \rangle \in set C\}"
proof (induction C)
case (Cons c C) thus ?case
proof (cases "\exists a y s. c = \langle a: Var y \in Fun (Set s) [] \rangle")
case True
then obtain a y s where: "c = \langle a: Var y \in Fun (Set s) [] \rangle" by moura

define f where "f \equiv transaction_poschecks_comp C"

have "transaction_poschecks_comp (c#C) = f(y := insert s (f y))"
using c by (simp add: f_def Let_def)
moreover have "f x = \{s. \exists a. \langle a: Var x \in Fun (Set s) [] \rangle \in set C\}"
using Cons.IH unfolding f_def by blast
ultimately show ?thesis using c by auto
next
case False
hence "transaction_poschecks_comp (c#C) = transaction_poschecks_comp C" (is ?P)
using transaction_poschecks_comp.cases[of "c#C" ?P] by force
thus ?thesis using False Cons.IH by auto
qed simp

lemma transaction_poschecks_comp_notin_fv_empty:
assumes "x \notin fv sst C"
shows "transaction_poschecks_comp C x = {}"
using assms transaction_poschecks_comp_unfold[of C x] by fastforce

lemma transaction_negchecks_comp_unfold:
"transaction_negchecks_comp C x = \{s. \langle Var x not in Fun (Set s) [] \rangle \in set C\}"
proof (induction C)
case (Cons c C) thus ?case
proof (cases "\exists y s. c = \langle Var y not in Fun (Set s) [] \rangle")
case True
then obtain y s where: "c = \langle Var y not in Fun (Set s) [] \rangle" by moura

define f where "f \equiv transaction_negchecks_comp C"

have "transaction_negchecks_comp (c#C) = f(y := insert s (f y))"
using c by (simp add: f_def Let_def)
moreover have "f x = \{s. \langle Var x not in Fun (Set s) [] \rangle \in set C\}"
using Cons.IH unfolding f_def by blast
ultimately show ?thesis using c by auto
next
case False
hence "transaction_negchecks_comp (c#C) = transaction_negchecks_comp C" (is ?P)
using transaction_negchecks_comp.cases[of "c#C" ?P] by force
thus ?thesis using False Cons.IH by fastforce
qed simp

lemma transaction_negchecks_comp_notin_fv_empty:
assumes "x \notin fv sst C"
shows "transaction_negchecks_comp C x = {}"
using assms transaction_negchecks_comp_unfold[of C x] by fastforce

lemma transaction_check_preI[intro]:
3 Stateful Protocol Verification

fixes T
defines "ϑ ≡ λδ x. if fst x = TAtom Value then (absc ◦ δ) x else Var x"
and "C ≡ set (unlabel (transaction_checks T))"

assumes a0: "∀x ∈ set (transaction_fresh T). δ x = {}"
and a1: "∀x ∈ fv_transaction T - set (transaction_fresh T). fst x = TAtom Value −→ δ x ∈ set OCC"
and a2: "∀t ∈ trms lsst (transaction_receive T). intruder_synth_mod_timpls FP TI (t · ϑ δ)"
and a3: "∀a x s. ⟨ a: Var x ∈ Fun (Set s) ⟩ ∈ C −→ s ∈ δ x"
and a4: "∀x s. ⟨ Var x not in Fun (Set s) ⟩ ∈ C −→ s /∈ δ x"

shows "transaction_check_pre (FP, OCC, TI) T δ"

proof -
let ?P = "λu. is_InSet u −→ (let x = the_elem_term u; s = the_set_term u
in (is_Var x ∧ is_Fun_Set s) −→ the_Set (the_Fun s) ∈ δ (the_Var x))"

let ?Q = "λu. (is_NegChecks u ∧ bvars sstp u = [] ∧ the_eqs u = [] ∧ length (the_ins u) = 1) −→ (let x = fst (hd (the_ins u)); s = snd (hd (the_ins u))
in (is_Var x ∧ is_Fun_Set s) −→ the_Set (the_Fun s) /∈ δ (the_Var x))"

have 1: "?P u" when u: "u ∈ C" for u
apply (unfold Let_def, intro impI, elim conjE)
using u a3 Fun_Set_InSet_iff[of u]
by metis

have 2: "?Q u" when u: "u ∈ C" for u
apply (unfold Let_def, intro impI, elim conjE)
using u a4 Fun_Set_NotInSet_iff[of u]
by metis

show ?thesis
using a0 a1 a2 1 2 fv_list sst lsst_is_fv[sst][of "unlabel (transaction_strand T)"]
unfolding transaction_check_pre_def ϑ_def C_def Let_def
by blast

qed

lemma transaction_check_pre_InSetE:
assumes T: "transaction_check_pre FPT T δ"
and u: "u = ⟨ a: Var x ∈ Fun (Set s) ⟩"
shows "s ∈ δ x"

proof -
have "is_InSet u −→ is_Var (the_elem_term u) ∧ is_Fun_Set (the_set_term u) −→
the_Set (the_Fun (the_set_term u)) ∈ δ (the_Var (the_elem_term u))"
using T u unfolding transaction_check_pre_def Let_def by blast
thus ?thesis using Fun_Set_InSet_iff[of u a x s] u by argo

qed

lemma transaction_check_pre_NotInSetE:
assumes T: "transaction_check_pre FPT T δ"
and u: "u = ⟨ Var x not in Fun (Set s) ⟩"
shows "s /∈ δ x"

proof -
have "is_NegChecks u ∧ bvars sstp u = [] ∧ the_eqs u = [] ∧ length (the_ins u) = 1 −→
is_Var (fst (hd (the_ins u))) ∧ is_Fun_Set (snd (hd (the_ins u))) −→
the_Set (the_Fun (snd (hd (the_ins u)))) /∈ δ (the_Var (fst (hd (the_ins u))))"
using T u unfolding transaction_check_pre_def Let_def by blast
thus ?thesis using Fun_Set_NotInSet_iff[of u x s] u by argo

qed

lemma transaction_check_pre_ReceiveE:
defines "ϑ ≡ λδ x. if fst x = TAtom Value then (absc ◦ δ) x else Var x"
assumes T: "transaction_check_pre (FP, OCC, TI) T δ"
and t: "t ∈ trms lsst (transaction_receive T)"
shows "intruder_synth_mod_timpls FP TI (t · ϑ δ)"

200
using $T \vdash$ unfolding \texttt{transaction}\_check\_pre\_def Let\_def $\emptyset$\_def by blast

\textbf{lemma} \texttt{transaction}\_check\_compI[\textit{intro}]:
\begin{itemize}
  \item \texttt{assumes} $T$: "\texttt{transaction}\_check\_pre (FP, OCC, TI) $T$ $\delta$"
  \item \texttt{and} $T\_adm$: "admissible\_transaction $T$"
  \item \texttt{x1}: "$\forall x. (x \in \text{fv\_transaction} T - \text{set} (\text{transaction}\_fresh T) \land \text{fst} x = T\text{Atom Value}) \rightarrow \delta x \in \text{set OCC} \land \text{msgcs} x (\delta x)"
  \item \texttt{x2}: "$\forall x. (x \notin \text{fv\_transaction} T - \text{set} (\text{transaction}\_fresh T) \lor \text{fst} x \neq T\text{Atom Value}) \rightarrow \delta x = \{\}"
\end{itemize}
shows "$\delta \in \text{abs\_subs}\_fun \cdot \text{set} (\text{transaction}\_check\_comp \text{msgcs} (FP, OCC, TI) T)"

\textbf{proof}
-\begin{itemize}
  \item define $S$ where "$S \equiv \text{unlabel} (\text{transaction}\_strand T)"
  \item define $C$ where "$C \equiv \text{unlabel} (\text{transaction}\_checks T)"
  \item let $?xs = \text{"fv\_list}_{\text{sst}} S"
  \item define $\text{poss}$ where "$\text{poss} \equiv \text{transaction}\_poschecks\_comp C"
  \item define $\text{negr}$ where "$\text{negr} \equiv \text{transaction}\_negchecks\_comp C"
  \item define $ys$ where "$ys \equiv \text{filter} (\lambda x. x \notin \text{set} (\text{transaction}\_fresh T) \land \text{fst} x = T\text{Atom Value}) ?xs"
  \item have $\{x \in \text{fv\_transaction} T - \text{set} (\text{transaction}\_fresh T). \text{fst} x = T\text{Atom Value}\} = \text{set} ys$
  \item using $\text{fv\_list}_{\text{sst}}\_\text{is\_fv\_sst}$ of $S$
  \item unfolding $ys\_\text{def} S\_\text{def}$ by force
  \item have "$\delta x \in \text{set OCC} \land \text{msgcs} x (\delta x)"
    \item when $x$: "$x \in \text{set} ys$" for $x$
    \item using $x1$ $x$ $ys$ by (blast, blast)
  \item moreover have "$\delta x = \{\}"
    \item when $x$: "$x \notin \text{set} ys$" for $x$
    \item using $x2$ $x$ $ys$ by blast
  \item moreover have "$\text{poss} x \subseteq \delta x$" when $x$: "$x \in \text{set} ys$" for $x$
  \item proof
    \begin{itemize}
      \item have "$s \in \delta x$" when $u$: "$u = (a: \text{Var} x \in \text{Fun} (\text{Set} s) [])" "u \in \text{set} C" for $u$ $a$ $s$
        \item using $T$ $u$ $\text{transaction}\_check\_pre\_InSetE$ of \"(FP, OCC, TI) $T$ $\delta$\"
        \item unfolding $C\_\text{def}$ by blast
        \item thus $?\text{thesis}$
    \end{itemize}
    \item using $\text{transaction}\_poschecks\_comp\_unfold$ of $C$ $x$
    \item unfolding $\text{poss}\_\text{def}$ by blast
    \item qed
  \item moreover have "$\delta x \cap \text{negr} x = \{\}" when $x$: "$x \in \text{set} ys$" for $x$
  \item proof (cases "$x \in \text{fv}_{\text{sst}} C")$
    \begin{itemize}
      \item case True
        \item hence "$s \notin \delta x$" when $u$: "$u = (\text{Var} x \text{ not in} \text{Fun} (\text{Set} s) [])" "u \in \text{set} C" for $u$ $s$
          \item using $T$ $u$ $\text{transaction}\_check\_pre\_NotInSetE$ of \"(FP, OCC, TI) $T$ $\delta$\"
          \item unfolding $C\_\text{def}$ by blast
          \item thus $?\text{thesis}$
    \end{itemize}
    \item using $\text{transaction}\_negchecks\_comp\_unfold$ of $C$ $x$
    \item unfolding $\text{negr}\_\text{def}$ by blast
    \item qed
  \item next
    \begin{itemize}
      \item case False
        \item hence "$\text{negr} x = \{\}$"
          \item using $x$ $\text{transaction}\_negchecks\_comp\_notin\_fv\_empty$
            \item unfolding $\text{negr}\_\text{def}$ by blast
            \item thus $?\text{thesis}$ by blast
    \end{itemize}
    \item qed
  \item ultimately have "$\delta \in \text{abs\_subs}\_fun \cdot \text{set} (\text{abs\_subs}\_\text{set} ys \text{ OCC} \text{poss} \text{negr} \text{msgcs})$"
    \item using $\text{abs\_subs}\_\text{has\_abs}$ of $ys$ $\delta$ $\text{OCC} \text{poss} \text{negr} \text{msgcs}$
    \item by fast
    \item thus $?\text{thesis}$
    \item using $?$
      \item unfolding $\text{transaction}\_check\_comp\_def Let\_def S\_\text{def} C\_\text{def} ys\_\text{def} poss\_\text{def} nega\_\text{def}$
        \item by fastforce
    \item qed
\end{itemize}
3 Stateful Protocol Verification

context

begin

private lemma transaction_check_comp_in_aux:
  fixes T
  defines "C ≡ set (unlabel (transaction_checks T))"
  assumes T_adm: "admissible_transaction T"
  and a1: "∀ x ∈ fv_transaction T - set (transaction_fresh T). fst x = TAtom Value −→ (∀ s. select[Var x, Fun (Set s) |] ∈ C −→ s ∈ α x)"
  and a2: "∀ x ∈ fv_transaction T - set (transaction_fresh T). fst x = TAtom Value −→ (∀ s. ⟨Var x in Fun (Set s) |⟩ ∈ C −→ s ∈ α x)"
  and a3: "∀ x ∈ fv_transaction T - set (transaction_fresh T). fst x = TAtom Value −→ (∀ s. ⟨Var x not in Fun (Set s) |⟩ ∈ C −→ s /∈ α x)"
  shows "∀ a x s. ⟨a: Var x ∈ Fun (Set s) |⟩ ∈ C −→ s ∈ α x" (is ?A)
  and "∀ x s. ⟨Var x not in Fun (Set s) |⟩ ∈ C −→ s /∈ α x" (is ?B)

proof -
  note * = admissible_transaction_strand_step_cases(2,3)[OF T_adm]
  have 1: "fst x = TAtom Value" "x ∈ fv_transaction T - set (transaction_fresh T)"
    when x: "⟨a: Var x ∈ Fun (Set s) |⟩ ∈ C" for a x s
    using * x unfolding C_def by fast+
  have 2: "fst x = TAtom Value" "x ∈ fv_transaction T - set (transaction_fresh T)"
    when x: "⟨Var x not in Fun (Set s) |⟩ ∈ C" for x s
    using * x unfolding C_def by fast+
  show ?A
    proof (intro allI impI)
      fix a x s assume u: "⟨a: Var x ∈ Fun (Set s) |⟩ ∈ C"
      thus "s ∈ α x" using 1 a1 a2 (cases a) metis+
    qed
  show ?B
    proof (intro allI impI)
      fix x s assume u: "⟨Var x not in Fun (Set s) |⟩ ∈ C"
      thus "s /∈ α x" using 2 a3 by meson
    qed

lemma transaction_check_comp_in:
  fixes T
  defines "ϑ ≡ λδ x. if fst x = TAtom Value then (absc ◦ δ) x else Var x"
  and "C ≡ set (unlabel (transaction_checks T))"
  assumes T_adm: "admissible_transaction T"
  and a1: "∀ x ∈ set (transaction_fresh T). α x = {}"
  and a2: "∀ t ∈ trms lsst (transaction_receive T). intruder_synth_mod_timpls FP TI (t · ϑ α)
  and a3: "∀ x ∈ fv_transaction T - set (transaction_fresh T). ∀ s. ⟨Var x, Fun (Set s) |⟩ ∈ C −→ s ∈ α x"
  and a4: "∀ x ∈ fv_transaction T - set (transaction_fresh T). ∀ s. ⟨Var x in Fun (Set s) |⟩ ∈ C −→ s ∈ α x"
  and a5: "∀ x ∈ fv_transaction T - set (transaction_fresh T). ∀ s. ⟨Var x not in Fun (Set s) |⟩ ∈ C −→ s /∈ α x"
  and a6: "∀ x ∈ fv_transaction T - set (transaction_fresh T). fst x = TAtom Value −→ α x ∈ set OCC"
  and a7: "∀ x ∈ fv_transaction T - set (transaction_fresh T). fst x = TAtom Value −→ msgcs x (α x)"
  shows "∃ δ ∈ abs_substs_fun · set (transaction_check_comp msgcs (FP, OCC, TI) T). ∀ x ∈ fv_transaction T. fst x = TAtom Value −→ α x = δ x"

proof -
  let ?xs = "fv_list as (unlabel (transaction_strand T))"
  let ?ys = "filter (λx. x /∈ set (transaction_fresh T)) ?xs"
  define α' where "α' ≡ λx."
if $x \in \text{fv}_{\text{transaction}} T - \text{set}(\text{transaction}_{\text{fresh}} T) \land \text{fst} x = \text{TAtom Value}$
then $\alpha x$
else {}"}

note $T_{\text{wf}} = \text{admissible}_{\text{transaction}} \text{is}_{\text{wellformed}} \text{transaction}(1)[\text{OF} T_{\text{adm}}]$

have $\vartheta_3 \text{Fun}: " \text{is}_{\text{Fun}} (t \cdot \vartheta x) \longleftrightarrow \text{is}_{\text{Fun}} (t \cdot \vartheta x')"$ for $t$
unfolding $\alpha'_\text{def} \vartheta_\text{def}$
by (induct $t$) auto

have ""\(t \in \text{trms}_{\text{sst}} (\text{transaction}_{\text{receive}} T). \text{intruder}_{\text{synth}}_{\text{mod}}_{\text{timpls}} \text{FP} \text{ TI} (t \cdot \vartheta x')""
proof (intro ballI impI)
fix $t$
assume $t: "t \in \text{trms}_{\text{sst}} (\text{transaction}_{\text{receive}} T)"

have 1: "intruder_{\text{synth}}_{\text{mod}}_{\text{timpls}} \text{FP} \text{ TI} (t \cdot \vartheta x')"
using $t$ a2
by auto

obtain $r$ where $r: "r \in \text{set}(\text{unlabel}(\text{transaction}_{\text{receive}} T))"$
"t \in \text{trms}_{\text{sst}} r"
using $t$
by (simp)

hence "\(\exists t s. r = \text{receive}(t s) \land t \in \text{set} t s""
using wellformed_transaction_unlabel_cases(1)[OF $T_{\text{wf}}$]
by (fastforce)

hence 2: "fv t \subseteq \text{fv}_{\text{transaction}} T" using $r$
by blast

ultimately have 3: "\(\vartheta_3 x = \vartheta x' x" when \"x \in \text{fv} t\" for $x$
using $t$ a2
by simp

have ""\(\forall x \in \text{fv}_{\text{transaction}} T. \text{fst} x = \text{TAtom Value} \longrightarrow (\forall s. \text{select}(\text{Var} x, \text{Fun} (\text{Set} s) [/]) \in C \longrightarrow s \in \alpha' x)""

"\(\forall x \in \text{fv}_{\text{transaction}} T - \text{set}(\text{transaction}_{\text{fresh}} T). \text{fst} x = \text{TAtom Value} \longrightarrow (\forall s. (\text{Var} x \in \text{Fun} (\text{Set} s) [/]) \in C \longrightarrow s \in \alpha' x)""

"\(\forall x \in \text{fv}_{\text{transaction}} T - \text{set}(\text{transaction}_{\text{fresh}} T). \text{fst} x = \text{TAtom Value} \longrightarrow (\forall s. (\text{Var} x \not\in \text{Fun} (\text{Set} s) [/]) \in C \longrightarrow s \not\in \alpha' x)""

using a3 a4 a5
unfolding $\alpha'_\text{def} \vartheta_\text{def} C_\text{def}$
by (meson+)

hence "\(\forall x s. (a: \text{Var} x \in \text{Fun} (\text{Set} s) [/]) \in C \longrightarrow s \in \alpha' x)"$
"\(\forall x s. (\text{Var} x \not\in \text{Fun} (\text{Set} s) [/]) \in C \longrightarrow s \not\in \alpha' x)"
using transaction_check_comp_in_aux[OF $T_{\text{adm}}$, of $\alpha'$]
unfolding $C_\text{def}$
by (fast, fast)

ultimately have 4: "\(\text{transaction}_{\text{check}}_{\text{pre}} (\text{FP}, \text{OCC}, \text{TI}) T \alpha'""
using a6 transaction_check_pre[of $T \alpha' \text{ OCC} \text{ FP} \text{ TI}]
unfolding $\alpha'_\text{def} \vartheta_\text{def} C_\text{def}$
by simp

have 5: "\(\forall x \in \text{fv}_{\text{transaction}} T. \text{fst} x = \text{TAtom Value} \longrightarrow \alpha x = \alpha' x)"
using a1 by (auto simp add: $\alpha'_\text{def}$)
3 Stateful Protocol Verification

have 6: \( \alpha' \in \text{abs_substs_fun} \cdot \text{set} (\text{transaction_check_comp} \ \text{msgcs} \ (\text{FP}, \ \text{OCC}, \ \text{TI}) \ T) \)
using transaction_check_compI[OF 4 T_adm, of msgcs] a6 a7
unfolding \( \alpha' \_\_\text{def} \)
by auto

show ?thesis using 5 6 by blast
qed
end

lemma transaction_check_trivial_case:
assumes "\( \text{transaction_updates} \ T = \[] \)"
and "\( \text{transaction_send} \ T = \[] \)"
shows "\( \text{transaction_check} \ FPT \ T \)"
using assms
by (simp add: list_all_iff transaction_check_def transaction_check'_def transaction_check_post_def)
end

3.6.3 Automatically Checking Protocol Security in a Typed Model

context stateful_protocol_model

begin

definition abs_intruder_knowledge ("\( \alpha_{ik} \)") where
"\( \alpha_{ik} \ S \ I \equiv (\text{ik}_{\text{sst}} \ S \cdot \text{set} \ I) \cdot \alpha_{\text{set}} \ (\text{db}_{\text{sst}} \ S \ I) \)"

definition abs_value_constants ("\( \alpha_{va} \)") where
"\( \alpha_{va} \ S \ I \equiv \{t \in \text{subterms}_{\text{set}} (\text{trms}_{\text{sst}} \ S) \cdot \text{set} \ I. \ \exists n. t = \text{Fun} (\text{Val} n) \} \cdot \alpha_{\text{set}} \ (\text{db}_{\text{sst}} \ S \ I) \)"

definition abs_term_implications ("\( \alpha_{ti} \)") where
"\( \alpha_{ti} \ A \ T \ \varphi \ I \equiv \{ (s,t) | s \neq t \land x \in \text{fv}_{\text{transaction}} T \land x \notin \text{set} (\text{transaction_fresh} T) \land \ \text{Fun} (\text{Abs} s) \cdot I \cdot \alpha_{\text{set}} \ (\text{db}_{\text{sst}} \ A \ I) \land \ \text{Fun} (\text{Abs} t) \cdot I \cdot \alpha_{\text{set}} \ (\text{db}_{\text{sst}} (\text{A@dual}_{\text{sst}} \ (\text{transaction_strand} T \cdot \text{sst} \ \varphi)) \ I) \} \)"

lemma abs_intruder_knowledge_append:
"\( \alpha_{ik} \ (A@B) \ I = (\text{ik}_{\text{sst}} \ A \cdot \text{set} \ I) \cdot \alpha_{\text{set}} \ (\text{db}_{\text{sst}} \ (A@B) \ I) \cup (\text{ik}_{\text{sst}} \ B \cdot \text{set} \ I) \cdot \alpha_{\text{set}} \ (\text{db}_{\text{sst}} \ (A@B) \ I) \)"
by (metis unlabel_append abs_set_union image_Un ik_sst_append abs_intruder_knowledge_def)

lemma abs_value_constants_append:
fixes A B::"('a,'b,'c,'d) prot_strand"
shows "\( \alpha_{va} \ (A@B) \ I = \{ t \in \text{subterms}_{\text{set}} (\text{trms}_{\text{sst}} \ A) \cdot \text{set} \ I. \ \exists n. t = \text{Fun} (\text{Val} n) \} \cdot \alpha_{\text{set}} \ (\text{db}_{\text{sst}} \ (A@B) \ I) \cup (\{ t \in \text{subterms}_{\text{set}} (\text{trms}_{\text{sst}} \ B) \cdot \text{set} \ I. \ \exists n. t = \text{Fun} (\text{Val} n) \} \cdot \alpha_{\text{set}} \ (\text{db}_{\text{sst}} \ (A@B) \ I) \)"
proof -
define a0 where "\( a0 \equiv \alpha_{0} \ (\text{db}_{\text{sst}} (A@B) \ I) \)"
define M where "\( M \equiv \lambda a::('a,'b,'c,'d) prot_strand. \{ t \in \text{subterms}_{\text{set}} (\text{trms}_{\text{sst}} a) \cdot \text{set} \ I. \ \exists n. t = \text{Fun} (\text{Val} n) \} \)"

have "\( M (A@B) = M A \cup M B \)"
using unlabel_append[of A B] trms_sst_append[of "unlabel A" "unlabel B" ]
image_Un[of "\( \lambda x. \cdot I \)" "\{ t \in subterms_{set} (trms_{sst} A) | subst_{set} (trms_{sst} A) \} \cup subst_{set} (trms_{sst} B)" ]
unfolding N_def by force
hence "\( M (A@B) \cdot \alpha_{set} \ a0 = (M A \cdot \alpha_{set} a0) \cup (M B \cdot \alpha_{set} a0) \)" by (simp add: abs_set_union)
thus ?thesis unfolding abs_value_constants_def a0_def M_def by force
qed

lemma transaction_renaming_subst_has_no_pubconsts_abss:
fixes a::"('fun,'atom,'sets,'lbl) prot_subst"
assumes "\( \text{transaction_renaming_subst} \ a P A \)"
shows "\(\text{subst\_range } \alpha \cap \text{pubval\_terms} = \{\}\)" (is \(?A\))
and "\(\text{subst\_range } \alpha \cap \text{abs\_terms} = \{\}\)" (is \(?B\))
proof - 
{ fix \(t\) assume "\(t \in \text{subst\_range } \alpha\)"
then obtain \(x\) where "\(t = \text{Var } x\)"
using transaction_renaming_subst_is_renaming[OF assms]
by force
hence "\(t \notin \text{pubval\_terms}\) "\(t \notin \text{abs\_terms}\)" by simp_all
} thus \(?A \notin \text{?B}\) by auto
qed

lemma transaction_fresh_subst_has_no_pubconsts_abss:
fixes \(\sigma\) :: "('fun,'atom,'sets,'lbl) prot_subst"
assumes "transaction_fresh_subst \(\sigma\) \(T\) \(A\)" "\(x \in \text{set (transaction_fresh } T\). \Gamma\)\(, x = 7\text{Atom Value}\)"
shows "\(\text{subst\_range } \alpha \cap \text{pubval\_terms} = \{\}\)" (is \(?A\))
and "\(\text{subst\_range } \alpha \cap \text{abs\_terms} = \{\}\)" (is \(?B\))
proof - 
{ fix \(t\) assume "\(t \in \text{subst\_range } \sigma\)"
then obtain \(x\) where "\(x \in \text{set (transaction_fresh } T)\)" "\(\sigma x = t\)"
using assms(1) unfolding transaction_fresh_subst_def
by auto
then obtain \(n\) where "\(t = \text{Fun (Val } n) [\]\)"
using transaction_fresh_subst_sends_to_val[OF assms(1)] assms(2)
by meson
hence "\(t \notin \text{pubval\_terms}\) "\(t \notin \text{abs\_terms}\)" unfolding is_PubConstValue_def by simp_all
} thus \(?A \notin \text{?B}\) by auto
qed

lemma reachable_constraints_GSMP_no_pubvals_abss:
assumes "\(A\) \(\in\) reachable\_constraints \(P\)"
and \(P\): "\(\forall T \in \text{set } P. \text{admissible\_transaction } T\)"
and \(I\) : "\(\text{interpretation\_subst } I\)" "\(\text{wt\_subst } I\)" "\(\text{wt\_terms } (\text{subst\_range } I)\)"
"\(\forall n. \text{PubConst Value } n \notin \bigcup \{\text{funs\_term } \sigma \circ I\} v_{lss}, A\)"
"\(\forall n. \text{Abs } n \notin \bigcup \{\text{funs\_term } \sigma \circ I\} v_{lss}, A\)"
shows "\(\text{trms}_{\text{sub} } A \circ I \subseteq \text{GSMP } (\bigcup T \in \text{set } P. \text{trms\_transaction } T) - \{\text{pubval\_terms } \cup \text{abs\_terms}\}\)"
(is \("\text{GSMP}\)\(\subseteq \text{?B}\)"
using assms(1) I(4,5)
proof (induction \(\lambda A \rightarrow \text{reachable\_constraints}\_\text{induct})
\(\text{case } (\text{step } A \text{ } T \text{ } \xi \text{ } \sigma \text{ } \alpha)\)
define trms\(\_P\) where "\(\text{trms}\_P \equiv (\bigcup T \in \text{set } P. \text{trms\_transaction } T)\)"
define \(T'\) where "\(\text{\(T'\) } \equiv\) transaction\_strand \(T \text{ } t_{lss} \xi \text{ } \sigma \text{ } o_{\nu} \text{ } \alpha\)"

have \(_\text{elim}_1\): "\(\xi \circ o_{\nu} \text{ } \sigma \text{ } o_{\nu} \text{ } \alpha \equiv \sigma \circ o_{\nu} \alpha\)"
using admissible\_transaction\_decl\_subst\_empty[OF bespec[OF P step.hyps(2)] step.hyps(3)]
by simp

note \(\text{T\_fresh} = \text{admissible\_transaction}_\text{E}(2)\) [OF bespec[OF P step.hyps(2)]]

note \(\text{T\_no\_bvars} = \text{admissible\_transaction}_\text{E}(4)\) [OF bespec[OF P step.hyps(2)]]

have \(\text{T\_no\_PubVal}\): "\(\forall T \in \text{set } P. \forall n. \text{PubConst Value } n \notin \bigcup \{\text{funs\_term } \sigma \circ I\} v_{lss} , A\)"
and \(\text{T\_no\_Abs}\): "\(\forall T \in \text{set } P. \forall n. \text{Abs } n \notin \bigcup \{\text{funs\_term } \sigma \circ I\} v_{lss} , A\)"
using admissible\_transactions\_no\_Value\_consts'[OF bespec[OF P]] by metis+

have \(J\) : "\(\forall n. \text{PubConst Value } n \notin \bigcup \{\text{funs\_term } \sigma \circ I\} v_{lss} , A\)"
"\(\forall n. \text{Abs } n \notin \bigcup \{\text{funs\_term } \sigma \circ I\} v_{lss} , A\)"
using step.prems fv_{lss} . append[of "unlabel \(A\)"] unlabel_append[of \(A\)]
by auto

have "\(\text{wt}_{\text{subst}} (\text{rm\_vars } (\text{set } X) \ (\xi \circ o_{\nu} \text{ } \sigma \text{ } o_{\nu} \text{ } \alpha))\)" for \(X\)
using wt_subst_rm_vars[of "\(\xi \circ o_{\nu} \text{ } \sigma \text{ } o_{\nu} \text{ } \alpha\)" "set \(X\)"
transaction\_decl\_fresh\_renaming\_subs\_wt[OF step.hyps(3-5)]
by metis

hence \(\text{wt} : "\(\text{wt}_{\text{subst}} (\text{rm\_vars } (\text{set } X) \ (\xi \circ o_{\nu} \text{ } \sigma \text{ } o_{\nu} \text{ } \alpha)) \circ I\)" for \(X\)
using I(2) wt_subst\_compose by fast
have \( \text{wf_trms: } \text{wf}\text{trms} ((\text{rm_vars } (\text{set } X) (\xi \circ \sigma \circ \alpha) \circ_o \mathcal{I}))^{\circ} \) for \( X \) using \text{wf_trms_substCompose}(\text{OF } \text{wf_trms_subst_rm_vars'} \mathcal{I}(3))

\text{transaction_decl_fresh_renaming_substs_range_wf_trms}[\text{OF } \text{step.hyps}(3-5)]

by fast

have \( \text{trms}\equiv\text{trms}_{\text{set\( \cdot \text{dual}_{\text{set}} \text{T'} \)}} \cdot \mathcal{I} \subseteq \mathcal{B} \)

proof

fix \( t \) assume \( t \in \text{trms}_{\text{set\( \cdot \text{dual}_{\text{set}} \text{T'} \)}} \cdot \mathcal{I} \)

then obtain \( s X \) where \( s: \)

\[
\begin{align*}
& t \in \text{trms}_{\text{set\( \cdot \text{dual}_{\text{set}} \text{T'} \)}} \cdot \mathcal{I} \\
& t = s \cdot \text{rm_vars } (\text{set } X) (\xi \circ \sigma \circ \alpha) \circ_o \mathcal{I} \\
& \text{set } X \subseteq \text{bvars_transaction } T
\end{align*}
\]

using \text{trms_sst_unlabel_dual}_{\text{lsst}}\equiv\text{lsst}\text{_eq}

by blast

define \( \vartheta \) where \( \vartheta: \)

\[
\begin{align*}
& \vartheta \equiv \text{rm_vars } (\text{set } X) (\xi \circ \sigma \circ \alpha)
\end{align*}
\]

have 1: \( s \in \text{trms}_P \)

using \text{step.hyps}(2) \( s(1) \)

unfolding \text{trms}_P\text{_def}

by (fastforce simp add: \text{is_PubConstValue_def} \text{is_Abs_def} \text{is_PubConst_def})

have \( s_{\text{nin}}: \) \( s \notin \text{pubval_terms} \) \( s \notin \text{abs_terms} \)

using \( T_{\text{no_PubVal}} T_{\text{no_Abs}} \) \( \text{funs_term_Fun_subterm} \)

unfolding \text{trms}_P\text{_def}

by (fastforce simp add: \text{is_PubConstValue_def} \text{is_Abs_def} \text{is_PubConst_def} \text{ξ_elim})

have 2: \( (\mathcal{I} \cdot \text{fv}_{\text{lsst}} (A@\text{dual}_{\text{set}} \text{T'})) \cap \text{pubval_terms} = \{\} \)

\( (\mathcal{I} \cdot \text{fv}_{\text{lsst}} (A@\text{dual}_{\text{set}} \text{T'})) \cap \text{abs_terms} = \{\} \)

\( \text{subst_range } (\xi \circ \sigma \circ \alpha) \cap \text{pubval_terms} = \{\} \)

\( \text{subst_range } (\xi \circ \sigma \circ \alpha) \cap \text{abs_terms} = \{\} \)

\( (\vartheta \cdot \text{fv } s) \cap \text{pubval_terms} = \{\} \)

\( (\vartheta \cdot \text{fv } s) \cap \text{abs_terms} = \{\} \)

unfolding \text{T'_def} \text{ϑ_def}

using \text{step.prems} \text{funs_term_Fun_subterm}

apply (fastforce simp add: \text{is_PubConstValue_def} \text{is_Abs_def} \text{is_PubConst_def})

by (fastforce simp add: \text{is_PubConstValue_def} \text{is_Abs_def} \text{ξ_elim})

have \( (\mathcal{I} \cdot \text{fv } (s \cdot \vartheta)) \cap \text{pubval_terms} = \{\} \)

\( (\mathcal{I} \cdot \text{fv } (s \cdot \vartheta)) \cap \text{abs_terms} = \{\} \)

proof

have \( \vartheta = \xi \circ \sigma \circ \alpha \) \( "\text{bvars_transaction } T = \{\} \) \( "\text{vars}_{\text{set}} T' = \text{fv}_{\text{lsst}} T' \)

using \( s(3) \) \( T_{\text{no_bvars}} \) \( \text{step.hyps}(2) \) \( \text{rm_vars_empty} \)

\( \text{vars}_{\text{set \_is_fv}} \text{_sst}_{\text{is_fv}} \text{_bvars}_{\text{set\_sst}} [\text{of } \text{unlabel } T'] \)

\( \text{bvars}_{\text{set\_sst}} [\text{of } \text{unlabel } (\text{transaction_strand } T) \equiv \xi \circ \sigma \circ \alpha] \)

\( \text{unlabel}_{\text{sst}} [\text{of } \text{transaction_strand } T \equiv \xi \circ \sigma \circ \alpha] \)

unfolding \( \vartheta_{\text{def}} T'_{\text{def}} \) by simp_all

hence \( \text{fv } (s \cdot \vartheta) \subseteq \text{fv}_{\text{sst}} T' \)

using \( \text{trms}_{\text{sst}} \cdot \text{fv}_{\text{sst\_subsets}} [\text{OF } s(1), \text{of } \vartheta] \) \( \text{unlabel}_{\text{sst}} [\text{of } \text{transaction_strand } T \equiv \vartheta] \)

unfolding \( T'_{\text{def}} \) by auto

moreover have \( \text{fv}_{\text{sst}} T' \subseteq \text{fv}_{\text{sst}} (A@\text{dual}_{\text{set}} T') \)

using \( \text{fv}_{\text{sst\_append}} [\text{of } \text{unlabel } A \equiv \text{unlabel } (\text{dual}_{\text{set}} T')] \)

\( \text{unlabel}_{\text{sst}} [\text{of } \text{A } \equiv \text{dual}_{\text{set}} T'] \)

by simp_all
hence "I ` fv_{last} T` ∩ pubval_terms = {}" "I ` fv_{last} T` ∩ abs_terms = {}"
using 2(1,2) by blast+
ultimately show "((I ` fv (s · ø)) ∩ pubval_terms = {}" "((I ` fv (s · ø)) ∩ abs_terms = {}"
by blast+

qed

hence σI_disj: "((ø o, I) ` fv s) ∩ pubval_terms = {}"
"((ø o, I) ` fv s) ∩ abs_terms = {}"

using pubval_terms_subst_range_comp[of s "fv s" I]
abs_terms_subst_range_comp[of s "fv s" I]
2(7,8)
by (simp_all add: subst_apply fv_unfold)

have 3: "t ∉ pubval_terms" "t ∉ abs_terms"
using s(2) s_min σI_disj
pubval_terms_subst[of s "rm_vars (set X) (ξ o, σ o, α) o, I"]
abs_terms_subst[of s "rm_vars (set X) (ξ o, σ o, α) o, I"]

unfolding θ_def
by blast+

have "t ∈ SMP trms_P" "fv t = {}"
by (metis s(2) SMP.Substitution[OF SMP.MP[OF 1] wt wftrms, of X],
metis s(2) subterms_subst_range_comp[of s "rm_vars (set X) (ξ o, σ o, α) o, I"]
interpretation_grounds[OF I(1), of "s · rm_vars (set X) (ξ o, σ o, α)"
]

hence 4: "t ∈ GSMP trms_P" unfolding GSMP_def by simp

show "t ∈ ?B" using 3 4 by (auto simp add: trms_P_def)

qed

thus ?case
using step.IH[OF I'] trms_{last} append[of "unlabel A"] unlabel_append[of A]
image_Un[of "λx. x · I" "trms_{last} A"]
by (simp add: T_def)

qed simp

lemma α₁₁.covers_αₐux:
assumes A_reach: "A ∈ reachable_constraints P"
and T: "T ∈ set P"

and I: "welltyped_constraint_model I (AÅ dual_{last} (transaction_strand T · I_{last} ξ o, σ o, α))"
and ξ: "transaction_decl_subst ξ T" 
and σ: "transaction_fresh_subst σ T A"
and α: "transaction_renaming_subst α P A"
and P: "∀T ∈ set P. admissible_transaction T"

and t: "t ∈ subterms_{set} (trms_{last} A)"
   "t = Fun (Val n) [] ∨ t = Var x"

and neq:
   "t · I ` α o o (db_{last} A I) ≠ t · I ` α o o (db_{last} (AÅ dual_{last} (transaction_strand T · I_{last} ξ o, σ o, α)) I)"
shows "y ∈ fv_transaction T - set (transaction_fresh T). 
   t · I = (ξ o, σ o, α) y · I ∧ I_o y = TAtom Value"

proof -
let ?A' = "AÅ dual_{last} (transaction_strand T · I_{last} ξ o, σ o, α)"
let ?B = "unlabel (dual_{last} (transaction_strand T))"
let ?B' = "?B · ξ o, σ o, α"
let ?B'' = "unlabel (dual_{last} (transaction_strand T · I_{last} ξ o, σ o, α))"

have I_inter: "interpretation_subst I"
   and I_wt: "wt_{subst} I"
   and I_vf: "vftrms (subterm_range I)"
by (metis I welltyped_constraint_model_def constraint_model_def,
   metis I welltyped_constraint_model_def,
   metis I welltyped_constraint_model_def constraint_model_def)
have \(T_{\text{adm}}: \text{"admissible\_transaction \(T\)"}
using \(T \ P(1)\) by blast

note \(T_{\text{wf}} = \text{admissible\_transaction\_is\_wellformed\_transaction}(1)[\text{OF } T_{\text{adm}}]\)
note \(T_{\text{adm\_upds}} = \text{admissible\_transaction\_is\_wellformed\_transaction}(3)[\text{OF } T_{\text{adm}}]\)

have \(T_{\text{fresh\_vars\_value\_typed}}: \text{"\(\forall x \in \text{set (transaction\_fresh \(T\)).} \Gamma_v \ x = \text{TAtom Value}\"}
using \(T \ P(1)\) protocol\_transaction\_vars\_TAtom\_typed(3)[\text{of } T] \ P(1)\) by simp

note \(\xi_{\text{empty}} = \text{admissible\_transaction\_decl\_subst\_empty}[\text{OF } T_{\text{adm}} \xi]\)
note \(\xi_{\sigma\_wt} = \text{transaction\_decl\_fresh\_renaming\_subs\_wt}[\text{OF } \xi \sigma \alpha]\)

have \(A_{\text{no\_val\_bvars}}: \text{"\(\neg \text{TAtom Value} \subseteq \Gamma_v \ x\"}
when \(\text{"}x \in \text{bvars}\_\text{lsst} \ A\text{"}\) for \(x\)
using \(P(1)\) reachable\_constraints\_no\_bvars \(A\_\text{reach}\)
vars\_\text{lsst}\_\text{is\_fv\_sst} \vars\_\text{sst} \[\text{of } \text{unlabel } A\] that
admissible\_transaction\_E(4)
by fast

have \(x': \text{"}x \in \text{var}\_\text{lsst} \ A\text{"}\) when \(\text{"}t = \text{Var } x\text{"}\)
using \(t\) by (simp add: var\_subterm\_trms\_sst\_is\_vars\_sst\_bvars\_sst\[of \text{unlabel } A\])

hence \(x_{\text{val}}: \text{"}\Gamma_v x = \text{TAtom Value}\text{"}\) when \(\text{"}t = \text{Var } x\text{"}\)
using \(t\) by force

hence \(x_{\text{fv}}: \text{"}x \in \text{fv}\_\text{lsst} \ A\text{"}\) when \(\text{"}t = \text{Var } x\text{"}\)
using \(t\) by force

then obtain \(m\) where \(\text{"}t \cdot I = \text{Fun } \text{(Val } m)\text{"}\)
using constraint\_model\_Value\_term\_is\_Val[
\(\text{OF } A\_\text{reach welltyped\_constraint\_model\_prefix}[\text{OF } I] P, \text{of } x\]
t(2) \(x_{\text{val}}\)
by force

hence \(0: \text{"}\alpha_0 (db\_\text{lsst} A I) m \neq \alpha_0 (db\_\text{sst} (\text{unlabel } A@?B'')) I m\text{"}\)
using neq by (simp add: unlabel\_def)

have \(t_{\text{val}}: \text{"}I \ t = \text{TAtom Value}\text{"}\) using \(x_{\text{val}} t\) by force

obtain \(u \ s\) where \(s: \text{"}t \cdot I = u \cdot I\text{"}\) \("insert\(u, s\) \in \text{set } ?B' \lor \text{delete}(u, s) \in \text{set } ?B'\"
using to\_abs\_eq\_if\_no\_Val\_neg[\text{OF } O] m\)
by (metis (no_types, lifting) dual\_lsst\_subst subst\_lsst\_unlabel)

then obtain \(u' \ s'\) where \(s'\):
\("u = u' \cdot \xi o_\sigma o_\alpha\text{"}\)
\("s = s' \cdot \xi o_\sigma o_\alpha\text{"}\)
"insert\(u', s') \in \text{set } ?B \lor \text{delete}(u', s') \in \text{set } ?B\"
using stateful\_strand\_step\_mem\_subst\_D(4,5)
by blast

hence \(s'': \text{"}insert\(u', s'\) \in \text{set } (\text{unlabel } (\text{transaction\_strand } T)) \lor \text{delete}(u', s') \in \text{set } (\text{unlabel } (\text{transaction\_strand } T))\"
using dual\_lsst\_unlabel\_steps\_iff(4,5)[\text{of } u' s' "transaction\_strand \(T\)"

208
3.6 Stateful Protocol Verification

by simp_all

then obtain y where y: "y ∈ fv_transaction T" "u' = Var y"
using transaction_inserts_are_Value_vars[OF T_wf T_adm_upds, of u' s']
transaction_deletes_are_Value_vars[OF T_wf T_adm_upds, of u' s']
stateful_strand_step_fv_subset_cases(4,5)[of u' s' "unlabel (transaction_strand T)"]

by auto

hence 1: "t · I = (ξ ∘ σ ∘ o) y · I" using y s(1) s'(1) by (metis subst_apply_term.simps(1))

have 2: "y ∉ set (transaction_fresh T)" when "(ξ ∘ σ ∘ o) y · I = y" using transaction_fresh_subst_grounds_domain[OF σ, of y] subst_compose[of σ o y] that ξ_empty by (auto simp add: subst_ground_ident)

have 3: "y ∉ set (transaction_fresh T)" when "(ξ ∘ σ ∘ o) y · I ∈ subtermsset (trmsiset A)"
using 2 that σ unfolding transaction_fresh_subst_def by fastforce

have 4: "y x ∈ fviset A. Γ y x = TAtom Value → (∃ B. prefix B A ∧ x ∉ fviset B ∧ I x ∈ subtermsset (trmsiset B))" using (metis welltyped_constraint_model_prefix[OF I]
constraint_model_Value_var_in_constr_prefix[OF A_reach P])

have 5: "Γ y = TAtom Value"
using t t_val
wt_subst_trm'[OF ξo wt, of "Var y"]
wt_subst_trm'[OF I wt, of t]
wt_subst_trm'[OF I wt, of "(ξ ∘ σ ∘ o) y"]
by (auto simp del: subst_compose)

have "y ∉ set (transaction_fresh T)"
proof (cases "t = Var x")
  case True
  hence #: "I x = Fun (Val m) (I)" "x ∈ fviset A" "I x = (ξ ∘ σ ∘ o) y · I"
  using m t(1) 1 x_fv x' by (force, blast, force)
  obtain B where B: "prefix B A" "I x ∈ subtermsset (trmsiset B)"
  using #(2) 2 x_val[OF True] by fastforce
  hence "t ∈ subst_range σ. t ∉ substset (trmsiset B)"
using transaction_fresh_subst_range_fresh[OF σ] trmsiset_unlabel_prefix_subset(1)[of B]
unfolding prefix_def by fast
  thus ?thesis using *(1,3) B(2) 2 by (metis subst_imgI term.distinct(1))
next
  case False
  hence "y ∈ set (transaction_fresh T)" using t by simp
  thus ?thesis using 1 3 by argo
qed

thus ?thesis using 1 5 y(1) by fast

qed

lemma αιι_covers_αιι_Var:
assumes A_reach: "A ∈ reachable_constraints P"
and T: "T ∈ set P"
and I: "welltyped_constraint_model I (A@dualiset (transaction_strand T @iset ξ ∘ σ ∘ o))"
and ξ: "transaction_decl_subst ξ T"
and σ: "transaction_fresh_subst σ T A"
and α: "transaction_renaming_subst α P A"
and P: "∀ T ∈ set P. admissible_transaction T"
and x: "x ∈ fviset A"
shows "I x ∘ α (dbiset (A@dualiset (transaction_strand T @iset ξ ∘ σ ∘ o)) I) ∈ timp closure_set (I x ∘ α (dbiset A I)) (αiι, A T (ξ ∘ σ ∘ o) I)"

proof -
define a0 where "a0 ≡ α (dbiset A I)"
define a0' where "a0' ≡ α (dbiset (A@dualiset (transaction_strand T @iset ξ ∘ σ ∘ o)) I)"
define a3 where "a3 ≡ αiι, A T (ξ ∘ σ ∘ o) I"

209
have \( A_{\text{wf_terms}} : \text{"wf_terms (trms_{set} A)"} \)
by (metis reachable_constraints_wf_terms admissible_transactions_wf_terms P(1) \( A_{\text{reach}} \))

have \( T_{\text{adm}} : \text{"admissible_transaction T"} \)
by (metis P(1) \( T \))

note \( \xi_{\text{empty}} = \text{admissible_transaction_decl_subst_empty[OF T_{adm} \( \xi \)} \)

note \( \xi_{\sigma_{\text{wt}}} = \text{transaction_decl_fresh_renaming_substs_wt[OF \( \xi_{\sigma_{\text{wt}}} \)} \)

have \( \tilde{I}_{\text{interp}} : \text{"interpretation_{subst I}"} \)
and \( \tilde{I}_{\text{wt}} : \text{"wt_{subst I}"} \)
by (metis \( I \) welltyped_constraint_model_def constraint_model_def, metis \( I \) welltyped_constraint_model_def, metis \( I \) welltyped_constraint_model_def constraint_model_def)

have \( \Gamma v x = \text{Var Value} \lor (\exists a. \Gamma v x = \text{Var (prot_atom.Atom a)}) \)
using reachable_constraints_vars_TAtom_typed[OF \( A_{\text{reach}} P \)} of \( x \]
by auto

disjunctive conjunctive conjunction implication disjunction existential universal quantifier
3.6 Stateful Protocol Verification

by auto
qed (auto intro: timpl_closure_setI)

next
assume \(\exists a. \Gamma_v x = T\text{Atom (Atom }a)\)
then obtain a where x_atom: \(\Gamma_v x = T\text{Atom (Atom }a)\)
by (cases \(\Gamma_v x\)) auto

have fT: \(\Gamma (Fun f T) = T\text{Atom (Atom }a)\)
using interpretation_grounds[OF I_interp, of "Var x"]
by (cases \(\Gamma x\)) auto

have fT_atom: \(\Gamma (Fun f T) = T\text{Atom (Atom }a)\)
using wt_subst_trm'[OF I_wt, of "Var x"] x_atom fT
by simp

have T: "T = []"
using fT wf_trm_subst[OF I_wf trms, of "Var x"]
by fastforce

have f: "¬ is_Val f" using fT_atom unfolding is_Val_def
by (auto)

have I T x α a0': "I x · α a0' ∈ timpl_closure_set \{I x · α a0\} a3"
by (auto intro: timpl_closure_setI)

thus \(?thesis\) by (metis a0_def a0'_def a3_def)
qed

lemma αti_covers_α0_val:
assumes A_reach: "A ∈ reachable_constraints P"
and T: "T ∈ set P"
and I: "welltyped_constraint_model I (A@dual lsst (transaction_strand T · I))" 
and σ: "transaction_fresh_subst σ T A" 
and α: "transaction_renaming_subst α P A" 
and P: "∀ T ∈ set P. admissible_transaction T"
and n: "Fun (Val n) [] ∈ subterms set (trms lsst A)"
shows "Fun (Val n) [] · α (db lsst (A@T)) I ∈ timpl_closure_set {Fun (Val n) [] · α (db lsst A I)} (ατ, A T (ξ o, σ o, α) I)"
proof -

define T' where "T' ≡ dual lsst (transaction_strand T · I)"

define a0 where "a0 ≡ α0 (db lsst A I)"

define a0' where "a0' ≡ α0 (db lsst (A@T')) I"

define a3 where "a3 ≡ ατ, A T (ξ o, σ o, α) I"

have A_wf_trms: "wf_trms (trms lsst A)"
by (metis reachable_constraints_wf trms admissible_transactions_wf_trms P(1) A_reach)

have T_adm: "admissible_transaction T" by (metis P(1) T)

have "Fun (Abs (a0' n)) [] ∈ timpl_closure_set \{Fun (Abs (a0 n)) []\} a3"

proof (cases "a0 n = a0' n")
case False
then obtain x where:
  "x ∈ fv_transaction T - set (transaction_fresh T)" "Fun (Val n) [] = (ξ o, σ o, α) x · I"
using αti_covers_α0_aux[OF A_reach I I x A P n]
by (fastforce simp add: a0_def a0'_def T_def)
hence "absb (a0 n) = (ξ o, σ o, α) x · I · a0" "absb (a0' n) = (ξ o, σ o, α) x · I · a0'" by simp

hence I: "(a0 n, a0' n) ∈ a3"
using False x(1)
unsfolding a0_def a0'_def a3_def abs_term_implications_def T'_def
by blast
show ?thesis
using timpl_apply_Abs[of "[]" "[]" "a0 n" "a0' n"]
  timpl_closure.TI[OF timpl_closure.FP[of "Fun (Abs (a0 n)) [] a3] 1]
  term_variants_pred_iff_in_term_variants[of 
    "λ_. []" "λ_. [Abs (a0 n) := [Abs (a0' n)]]"
] unfolding timpl_closure_set_def timpl_apply_term_def
by force
qed (auto intro: timpl_closure_set)
thus ?thesis by (simp add: a0_def a0'_def a3_def T'_def)
qed

lemma αι,ι covers αι,ι Ik:
  assumes A_reach: "A ∈ reachable_constraints P"
  and T: "T ∈ set P"
  and I: "welltyped_constraint_model I (A#dualι#set (transaction_strand T #just (t#set I)), α#set (db A #set P))"
  and ξ: "transaction_decl_subset ξ T"
  and σ: "transaction_fresh_subset σ T A"
  and α: "transaction_renaming_subset α P A"
  and P: "∀T ∈ set P. admissible_transaction T"
  and t: "t ∈ ik#set A"
  shows "t · I · α (A#dualι#set (transaction_strand T #just (t#set I)), α#set (db A #set P)) I) ∈ 
    timpl_closure_set {t · I · α (A#dualι#set I))} (α, A T (ξ # set P,"ι,ι) I)"
proof -
define a0 where "a0 ≡ α0 (db#set A #set P)"
define a0' where "a0' ≡ a0 (db#set (A#dualι#set (transaction_strand T #just (t#set I)), α#set P))) I)"
define a3 where "a3 ≡ αι#set (A T (ξ # set P,"ι,ι) I)"
let ?U = "λT a. map (λs · I · a) T"
have A wf trms: "wf trms (trms#set A)"
  by (metis reachable_constraints wf trms, admissible_transactions wf trms P (1) A_reach)
have t adm: "admissible_transaction T" by (metis P (1) T)

have "t ∈ subst_terms set (ik#set A) "wf trms t" using A wf trms t ik#set trms#set subst by force
hence "∀t0 ∈ subst_terms t. t0 · I · α0 ∈ timpl_closure_set {t0 · I · α0} a3"

proof (induction t)
case (Var x) thus ?case
  using αι,ι covers αι,ι Var[of A reach T I (σ # α P) of x]
  ik#set var_is_fv[of "unlabel A"] vars#set_is_fv#set bv#vars #set[of "unlabel A"]
  by (simp add: a0_def a0'_def a3_def)
next
case (Fun f S)
have IH: "∀t0 ∈ subst_terms t. t0 · I · α0 ∈ timpl_closure_set {t0 · I · α0} a3"
  "when t ∈ subst_set S" for t
  using Fun.prems(1) wf_trm_param[of Fun.prems(2)] Fun.IH
  by (meson in_subterms_subset_Union params_subterms subsetCE)
hence "t · α0 ∈ timpl_closure_set {t · α0} a3"
  "when t ∈ subst (map (λs · I · S))" for t
  using that by auto
hence "t · α0 ∈ timpl_closure (t · α0) a3"
  "when t ∈ subst (map (λs · I · S))" for t
  using that timpl_closure_on_is_timpl_closure by auto
hence "(t · α0, t · α0') ∈ timpl_closure' a3"
  "when t ∈ subst (map (λs · I · S))" for t
  using that timpl_closure_is_timpl_closure' by auto
hence IH': "((?U S a0)!i, (?U S a0')!i) ∈ timpl_closure' a3"
  when "i < length (map (λs · I · α0) S)" for i
  using that by auto

show ?case
proof (cases "∃n. f = Val n")
case True

212
then obtain n where "Fun f S = Fun (Val n) []"
  using Fun.prems(2) unfolding wterm_def by force
moreover have "Fun f S ∈ subterms_set (trms lsst A)"
  using ikrst.trmsrst_subset Fun.prems(1) by blast
ultimately show ?thesis
  using αi.covers_α0.Val[OF A_reach T I ξ σ π] by (simp add: a0_def a0'_def a3_def)
next 
case False 
hence "Fun f S · I · a = Fun f (map (λ t. t · I · a) S)" for a by (cases f) simp_all
hence "(Fun f S · I · a0, Fun f S · I · a0') ∈ timpl_closure' a3"
  using timpl_closure_FunI[OF IH'] by simp
hence "Fun f S · I · a0' ∈ timpl_closure_set {Fun f S · I · a0} a3"
  using timpl_closureton_is_timpl_closure timpl_closure_is_timpl_closure'
  by metis
thus ?thesis using IH by (simp add: a0_def a0'_def a3_def)
qed
proof -
let ?upd = "λx. absdbupd (unlabel (transaction_updates T)) x (δ x)"
have 0: "fv_transaction T = set (fv_list sst (unlabel (transaction_strand T)))"
  by (metis fv_list sst_is_fv sst[OF "unlabel (transaction_strand T)"])
have 1: "transaction_check_post (FP, OCC, TI) T δ"
  using assms(1,5) unfolding transaction_check_def transaction_check'_def list_all_iff
  by blast
have "(δ x, ?upd x) ∈ set TI ↔ (δ x, ?upd x) ∈ (set TI) + "
  using TI using assms(4)
  by blast
thus ?thesis
  using assms(2,3,4) 0 1 in_trancl_closure_iff_in_trancl_fun[of _ _ TI]
  unfolding transaction_check_post_def List.member_def Let_def
  by blast
qed
lemma transaction_prop1:
  assumes δ: "δ ∈ abs_substs_fun \ set (transaction_check_comp msgcs (FP, OCC, TI) T)"
  and "x ∈ fv_transaction T"
  and "x ∉ set (transaction_fresh T)"
  and "δ x ≠ absdbupd (unlabel (transaction_updates T)) x (δ x)"
  and "transaction_check' msgcs (FP, OCC, TI) T"
  and TI: "set TI = {(a,b) ∈ (set TI) + . a ≠ b}"
  shows "(δ x, absdbupd (unlabel (transaction_updates T)) x (δ x)) ∈ (set TI) + "
proof -
  let ?upd = "λx. absdbupd (unlabel (transaction_updates T)) x (δ x)"
  have 0: "fv_transaction T = set (fv_list sst (unlabel (transaction_strand T)))"
    by (metis fv_list sst_is_fv sst[OF "unlabel (transaction_strand T)"])
  have 1: "transaction_check_post (FP, OCC, TI) T δ"
    using assms(1,5) unfolding transaction_check_def transaction_check'_def list_all_iff
    by blast
  have "(δ x, ?upd x) ∈ set TI ↔ (δ x, ?upd x) ∈ (set TI) + "
    using TI using assms(4)
    by blast
  thus ?thesis
    using assms(2,3,4) 0 1 in_trancl_closure_iff_in_trancl_fun[of _ _ TI]
    unfolding transaction_check_post_def List.member_def Let_def
    by blast
qed
lemma transaction_prop2:
  assumes δ: "δ ∈ abs_substs_fun \ set (transaction_check_comp msgcs (FP, OCC, TI) T)"
  and "x ∈ fv_transaction T"  "fst x = TAtom Value"
  and T_check: "transaction_check' msgcs (FP, OCC, TI) T" 
  and T_adm: "admissible_transaction T"
  and FP:
    "analyzed (timpl_closure_set (set FP) (set TI))"
    "wterm (set FP)"
  and OCC:
    "∀ t ∈ timpl_closure_set (set FP) (set TI). ∀ f ∈ funs_term t. is_Abs f → f ∈ Abs ` set OCC"
    "timpl_closure_set (absb ` set OCC) (set TI) ⊆ absc ` set OCC"
  and TI:
    "set TI = {(a,b) ∈ (set TI) + . a ≠ b}"
  shows "x ∉ set (transaction_fresh T) → δ x ∈ set OCC" (is "?A' → ?A")
and "absdbupd (unlabel (transaction_updates T)) x (δ x) ∈ set OCC" (is ?B)

proof -
let ?xs = "fv_list_sst (unlabel (transaction_strand T))"
let ?ys = "filter (λx. x ∉ set (transaction_fresh T) ∧ fst x = TAtom Value) ?xs"
let ?C = "unlabel (transaction_checks T)"
let ?poss = "transaction_poschecks_comp ?C"
let ?negs = "transaction_negchecks_comp ?C"
let ?δupd = "λy. absdbupd (unlabel (transaction_updates T)) y (δ y)"

note T wf = admissible_transaction_is_wellformed_transaction(1)[OF T adm]
note T occ = admissible_transaction_is_wellformed_transaction(5)[OF T adm]

have 0: "{(x ∈ fv_transaction T - set (transaction_fresh T). fst x = TAtom Value) = set ?ys"
using fv_list_sst_is_fv[of "unlabel (transaction_strand T)"
by force

have 1: "transaction_check_pre (FP, OCC, TI) T δ" using δ unfolding transaction_check_comp_def Let_def by fastforce

have 2: "transaction_check_post (FP, OCC, TI) T δ" using δ unfolding transaction_check_post_def transaction_check'_def list_all_iff by auto

have 3: "δ ∈ abs_substs_fun \ set (abs_substs_set ?ys OCC ?poss ?negs msgcs)"
using δ unfolding transaction_check_comp_def Let_def by force

show A: ?A when ?A' using that 0 3 x abs_substs_abss_bounded by blast

have 4: "x ∈ fv_transaction T ∪ fv_transaction (transaction_send T)"
when x': "x ∈ set (transaction_fresh T)"
using x' unfolding suffix_def[λx. x ∉ set (transaction_fresh T) ∧ fst x = TAtom Value] by blast

have "intruder_synth_mod_timpls FP TI (occurs (absc (?δupd x)))"
when x': "x ∈ set (transaction_fresh T)"

proof -
obtain l ts S where ts:
"transaction_send T = (l, send ⟨ts⟩) # S" "occurs (Var x) ∈ set ts"
using admissible_transaction_occurs_checksE2[OF T occ x'] by blast
hence "occurs (Var x) ∈ set ts" "send⟨ts⟩ ∈ set (unlabel (transaction_send T))"
using x' unfolding suffix_def by (fastforce, fastforce)
thus ?thesis using 2 x unfolding transaction_check_post_def by fastforce
qed

hence "timpl_closure_set (set FP) (set TI) ⊢ occurs (absc (?δupd x))"
when x': "x ∈ set (transaction_fresh T)"
using x' intruder_synth_mod_timpls_is_synth_timpl_closure_set[OF TI, of FP "occurs (absc (?δupd x))"]
by argo

hence "Abs (?δupd x) ∈ \{funs_term \ timpl_closure_set (set FP) (set TI)\}"
when x': "x ∈ set (transaction_fresh T)"
using x' ideduct_synth_priv_fun_in_ik[λx. x ∉ set (transaction_fresh T)"
by simp

hence "t ∈ timpl_closure_set (set FP) (set TI). Abs (?δupd x) ∈ funs_term t"
when x': "x ∈ set (transaction_fresh T)"
using x' by force

hence 5: "?δupd x ∈ set OCC" when x': "x ∈ set (transaction_fresh T)"
using x' OCC by fastforce

have 6: "?δupd x ∈ set OCC" when x': "x ∉ set (transaction_fresh T)"
proof (cases "δ x = ?δupd x")
case False
hence "(δ x, ?δupd x) ∈ (set TI)\{x ∈ set OCC"
using A 2 x' x TI
unfolding transaction_check_post_def fv_list_sst_is_fv_sst Let_def
lemma transaction_prop3:
assumes A_reach: "A ∈ reachable_constraints P"
and T: "T ∈ set P"
and I: "welltyped_constraint_model I (A◊dualiset (transaction_strand T ∘trans∈ set (transaction_fresh T)))"
and ξ: "transaction_decl_subst ξ T" 
and σ: "transaction_fresh_subst σ T A" 
and α: "transaction_renaming_subst α P A"
and FP:
"analyzed (timpl_closure_set (set FP) (set TI))"
"wftrans (set FP)"
"∀ t ∈ α. A I. timpl_closure_set (set FP) (set TI) ⊢ c t"
and OCC:
"∀ t ∈ timpl_closure_set (set FP) (set TI). ∀ f ∈ funs_term τ. is.Abs f → f ∈ Abs ` set OCC"
"timpl_closure_set (absc ` set OCC) (set TI) ⊆ absc ` set OCC"
"αvals. A I ⊆ absc ` set OCC"
and TI:
"set TI = {(a, b) ∈ (set TI) +. a ≠ b}"
and P:
"∀ T ∈ set P. admissible_transaction T"
shows "∀ x ∈ set (transaction_fresh T). (ξ σ σ α) x · I · α0 (dbiset A I) = absc {x}" (is ?A) 
and "∀ t ∈ trms_set (transaction_receive T).
  intruder_synch_mod_timpls FP TI (t · ξ σ σ α · I · α0 (dbiset A I))" (is ?B) 
and "∀ x ∈ fv_transaction T - set (transaction_fresh T).
  ∀ s. select (Var x,Fun (Set s) []) ∈ set (unlabel (transaction_checks T))
    (iff ss. (ξ σ σ α x · I · α0 (dbiset A I)) = absc ss ∧ s ∈ ss)" (is ?C) 
and "∀ x ∈ fv_transaction T - set (transaction_fresh T).
  ∀ s. (Var x in Fun (Set s) []) ∈ set (unlabel (transaction_checks T))
    (iff ss. (ξ σ σ α x · I · α0 (dbiset A I)) = absc ss ∧ s ∈ ss)" (is ?D) 
and "∀ x ∈ fv_transaction T - set (transaction_fresh T).
  ∀ s. (Var x not in Fun (Set s) []) ∈ set (unlabel (transaction_checks T))
    (iff ss. (ξ σ σ α x · I · α0 (dbiset A I)) = absc ss ∧ s ∉ ss)" (is ?E) 
and "∀ x ∈ fv_transaction T - set (transaction_fresh T). (ξ σ σ α) x · I · α0 (dbiset A I) ∈ absc ` set OCC" (is ?F) 
proof -
let ?I = "dualiset (transaction_strand T ∘trans∈ set (transaction_fresh T))"
let θ = "ξ σ σ α"

define a0 where "a0 ≡ α0 (dbiset A I)"
define a0' where "a0' ≡ α0 (dbiset (A◊?T') I)"
define fv_AT' where "fv_AT' ≡ fviset (A◊?T')"

have T_wf: "admissible_transaction T"
  using T(P) by blast

note T_wf = admissible_transaction_is_wellformed_transaction(1)[OF T_adm]
note T_adm' = admissible_transaction_is_wellformed_transaction(2-4)[OF T_adm]

note ξ_empty = admissible_transaction_decl_subst_empty[OF T_adm ξ]
hence ξ_elim: "?θ = σ σ α" by simp

have I': "interpretation_subst I" "wfsubst I" "wftrans (sub_set_range I)"
  "∀ n. PubConst Value n ∉ \{ (funs_term · (I · fviset A)) | n \}"
  "∀ n. Abs n ∉ \{ (funs_term · (I · fviset A)) | n \}"
  "∀ n. PubConst Value n ∉ \{ (funs_term · (I · fv_AT')) | n \}"

215
3 Stateful Protocol Verification

"∀ n. Abs n ∉ \{\text{fv}_\mathbf{AT}'\}

using \(I\) admissible_transaction_occurs_checks_prop'[\n
OF A_reach welltyped_constraint_model_prefix[OF \(I\)] P]\n
admissible_transaction_occurs_checks_prop'[\n
OF reachable_constraints.step[OF A_reach Τ ξ σ \(I\)] P]\n
unfolding welltyped_constraint_model_def constraint_model_def is_Val_def is_Abs_def fv_AT'_def\n
by (meson, meson, meson,metis,metis,metis,metis)

have \(T\) no_pubconsts: "∀ n. PubConst Value n ∉ \{\text{fv}_\mathbf{term} \cdot \text{trs}_\mathbf{transaction} T\}"

and \(T\) no_abs: "∀ n. Abs n ∉ \{\text{fv}_\mathbf{term} \cdot \text{trs}_\mathbf{transaction} T\}"

and \(T\) fresh vars value typed: "∀ x ∈ set \{\text{transaction_fresh} T\}. \Gamma_v x = \text{TAtom Value}"

and \(T\) fv const typed: "∀ x ∈ fv_transaction \(\cdot\) T. \Gamma_v x = \text{TAtom Value} ∨ (∃ a. \Gamma_v x = \text{TAtom (Atom a)}"

using protocol_transaction_vars\_{TAtom} typed

protocol_transactions_no_pubconsts

protocol_transactions_no_abs

\text{uns} term Fun \text{subst} P T

by (fast, fast, fast, fast)

have \(\text{wt}_\text{σ} \cdot ω I\): "\text{wt}_\text{subst} (σ ◦ ω ◦ ω \(I\))"

using \(I\)'(2) \text{wt}_\text{subst_compose} transaction_fresh_subst_unf[\text{OF} \(σ\) \text{transaction renaming subst un}\text{f}[\text{OF} \(σ\)]

by blast

have 1: "∀ y \cdot I = σ y" when "y ∈ set \{\text{transaction_fresh} T\}" for y

using transaction_fresh_subst_grounds_domain[\text{OF} \(σ\) that] subst_compose[of \(σ\) y]

unfolding \(ξ\) elim by (simp add: subst_ground_ident)

have 2: "∀ y \cdot I ≠ \text{subterms}_{\text{set}} \{\text{trs}_{\text{set}} \cdot \text{A}\}" when "y ∈ set \{\text{transaction_fresh} T\}" for y

using 1[\text{OF} that] then \(σ\) unfolding transaction_fresh_subst_unf by auto

have 3: "∀ x ∈ fv_transaction \(\cdot\) A. \(\Gamma_v x = \text{TAtom Value} → (\exists B. \text{prefix} B A ∧ x ∉ \text{fv}_{\text{set}} B ∧ I \cdot x ∈ \text{subterms}_{\text{set}} \{\text{trs}_{\text{set}} B\})"

by (metis welltyped_constraint_model_prefix[\text{OF} \(I\)]

constraint_model_value_var_in_constr_prefix[\text{OF} A_reach _P])

have 4: "∃ n. \(\exists y \cdot I = \text{Fun} (\text{Val} n) []\)

when "y ∈ fv_transaction \(\cdot\) T" "\(Γ_v y = \text{TAtom Value}\)" for y

using transaction_var_becomes_val[\text{OF} reachable_constraints.step[\text{OF} A_reach Τ ξ σ \(I\)] Τ ξ σ P T]

that Tfv const typed \(\text{Γ}_v \cdot \text{TAtom}'\)[of y]

by metis

have \(I\) is T model: "\text{strand}_{\text{sem}} \text{stateful} (\text{ik}_{\text{set}} A \cdot \text{set} I) \{\text{set} \text{db}_{\text{set}} A I\}) \{\text{unlabel} \ ?T\'} I"\n
using I unlavel append[of A \(\equiv\) I] \text{db}_{\text{set}} set is dbupd set[of "unlabel A" I "[]"]

\text{strand}_{\text{sem}} append stateful[of "\{\}" "\{\}" "unlabel A" "unlabel \ ?T'" I]

by (simp add: welltyped_constraint_model_def constraint_model_def db_{set def})

have \(T\) rsv no val bvars: "\text{bvars}_{\text{set}} \{\text{transaction receive} T\} ∩ \text{subst domain} \(\equiv \) []"\n
using admissible_transaction_no bvars[\text{OF} \(T\) \text{adm}] \text{bvars transaction unfold}[of T] by blast

show \(?A\)

proof

fix y assume y: "y ∈ set \{\text{transaction fresh } T\}"

then obtain \(y n\) where \(y n\): "(ξ σ o σ o σ) y · I = \text{Fun} (\text{Val} y n) []\) "\text{Fun} (\text{Val} y n) []\) ∈ subst_range σ" by (metis \(ξ\) elim Tfv fresh vars value typed transaction fresh subst sends to \text{val}'[\text{OF} \(σ\)]

{ since y is fresh (ξ σ o σ o σ) y · I cannot be part of the database state of \(I\) A

fix t' s assume t': "\text{insert}(t', s) ∈ set (unlabel A)" "t' · I = \text{Fun} (\text{Val} y n) []\) then obtain z where t'_z: "t' = \text{Var} z" using 2[\text{OF} y] \(y n\)(1) by (cases t') auto

hence z: "z ∈ fv_{\text{set}} A" "z = (ξ σ o σ o σ) y · I" using t' \(y n\)(1) by force

hence z': "\(Γ_v z = \text{TAtom Value}\)" by (metis \(Γ\).simp(1) Γ_consta.simps(2) t' \(2\) t'_z wt subst trm' \(I\)'(2))
obtain $B$ where $B$: "prefix $B$ $A" \rightarrow \exists z \in \text{subterms}_{\text{set}} (\text{trms}_{\text{set}} B)"$ using $z'$ 3 by fastforce

hence "\( t \in \text{subterms}_{\text{set}} (\text{trms}_{\text{set}} B) \)"

using $\text{transaction}_{\text{fresh}}_{\text{subst}}_{\text{range}}_{\text{fresh}}(1)[\text{OF } \sigma] \text{trms}_{\text{set}}_{\text{unlabel}}_{\text{prefix}}_{\text{subset}}(1)[\text{of } B]$

unfolding $\text{prefix}_{\text{def}}$ by fastforce

hence False using $B(2) \ 1[\text{OF } y \ z \ \text{yn}(1)]$ by (metis $\text{subst}_{\text{img}} 1 \ \text{term}_{\text{distinct}}(1))

} hence \text{"} \exists s. (\exists y \cdot \overline{I}, s) \in \text{set} (\text{db}_{\text{set}} A \overline{I})^* \}\text{"}

using $\text{db}_{\text{set}}_{\text{in}}_{\text{cases}}[\text{of } \exists y \cdot \overline{I}^* _\text{unlabel} A \overline{I} \"[\]" \ \text{yn}(1)]$

by (force simp add: $\text{db}_{\text{set}}_{\text{def}}$)

thus \text{"} \exists y \cdot \overline{I} \alpha \alpha_0 (\text{db}_{\text{set}} A \overline{I}) = \text{abs} \{}

using $\text{to}_{\text{abs}}_{\text{empty}}_{\text{iff}}_{\text{notin}}_{\text{db}}[\text{of } y \ \text{"db'}_{\text{set}} A \overline{I} \"[\]" \ \text{yn}(1)]$

by (simp add: $\text{db}_{\text{set}}_{\text{def}}$)

qed

show receives_covered: \text{?B}

proof

fix $t$ assume $t$: "$t \in \text{trms}_{\text{set}} (\text{transaction}_{\text{receive}} T)"

hence $t \in \text{trms}_{\text{set}} (\text{transaction}_{\text{receive}} T)"

using $\text{trms}_{\text{set}}_{\text{unlabel}}_{\text{prefix}}_{\text{subset}}(1)[\text{of } \text{transaction}_{\text{receive}} T]$

unfolding $\text{transaction}_{\text{strand}}_{\text{def}}$ by fastforce

obtain $ts$ where $ts$: "$t \in \text{set} \text{ts}" \ \text{receive}(\text{ts}) \in \text{set} (\text{unlabel} \ (\text{transaction}_{\text{receive}} T))"

using $t \ \text{wellformed}_{\text{transaction}}_{\text{send}_{\text{receive}}_{\text{trm}}}_{\text{cases}}(1)[\text{OF } T \ \text{wf}]$ by blast

have $t_{\text{rcv}}$: "$\text{receive}(\text{ts} \cdot \text{t}_{\text{rcv}} \ ?\theta) \in \text{set} (\text{unlabel} \ (\text{transaction}_{\text{receive}} T \cdot \text{t}_{\text{rcv}} \ ?\theta))"$

using $\text{subt}_{\text{set}}_{\text{unlabel}}_{\text{member}}[\text{of } \text{receive}(\text{ts}) "\text{transaction}_{\text{receive}} T \ ?\theta]$

$\text{trms}_{\text{set}}_{\text{unlabel}}_{\text{prefix}}_{\text{subset}}(1)[\text{of } T \ \text{ts}]$

by fastforce

have \text{"} \ \text{list}_{\text{all}} \ (\forall \alpha \ \text{ik}_{\alpha} A \ ' \cdot \ set \ I \ | \ t \ - \ ?\theta \cdot \overline{I}) \ \text{ts}"$

using $\text{wellformed}_{\text{transaction}}_{\text{sem}}_{\text{receives}}[\text{OF } T \ \text{wf} \ \overline{I} \ \text{is}_{\text{T}}_{\text{model}} \ t_{\text{rcv}}]$

unfolding $\text{list}_{\text{all}}_{\text{iff}}$ by fastforce

hence \text{"} \ \text{ik}_{\alpha} A \ ' \cdot \ set \ I \ | \ t \ - \ ?\theta \cdot \overline{I}^* \ \text{using} \ \text{ts}(1) \ \text{unfolding} \ \text{list}_{\text{all}}_{\text{iff}} \ \text{by} \ \text{fastforce}"

have \text{"} \ \text{fv}_{\text{def}}: \ "\text{fv} (t \cdot ?\theta) \subseteq \text{fv}_{\overline{A}T}^*"$

using $\text{fv}_{\alpha} _{\text{set}} _{\text{append}}[\text{of } \\text{unlabel} A] \ \text{unlabel}_{\text{append}}[\text{of } A]$

$\text{fv}_{\alpha} _{\text{set}} _{\text{unlabel}} _{\text{dual}} _{\text{set}} _{\text{eq}}[\text{of } \text{transaction}_{\text{strand}} T \cdot \text{t}_{\text{set}} \ ?\theta]$

$\text{ts}(1) \ t_{\text{rcv}} \ \text{fv}_{\text{transaction}}_{\text{subterm}}_{\text{unfold}}[\text{of } T \ ?\theta]$

unfolding $\text{fv}_{\overline{A}T}^*_{\text{def}}$ by force

have \text{"} \ \text{fv}_{\text{def}}: \ "\forall \alpha \in (\text{ik}_{\alpha} A \ ' \cdot \ set) \ \text{a0}. \ \text{timpl}_{\text{closure}}_{\text{set}} \ (\text{set} \ \text{FP}) \ (\text{set} \ \text{TI}) \ | \ t^*"

using $\text{FP}(3)$ by (auto simp add: a0_def abs_intruder_knowledge_def)

note \text{rms1} = $\ \text{pubval}_{\text{terms}}_{\text{subst}}[\text{OF } \ \text{pubval}_{\text{terms}}_{\text{subst}}_{\text{range}}_{\text{disj}}[

\text{OF } \ \text{transaction}_{\text{fresh}}_{\text{subst}}_{\text{has}_{\text{no}}_{\text{pubconsts}}}_{\text{abs}}(1)[\text{OF } \sigma], \ \text{of } t]$

$\ \text{pubval}_{\text{terms}}_{\text{subst}}[\text{OF } \ \text{pubval}_{\text{terms}}_{\text{subst}}_{\text{range}}_{\text{disj}}[

\text{OF } \ \text{transaction}_{\text{renaming}}_{\text{subst}}_{\text{has}_{\text{no}}_{\text{pubconsts}}}_{\text{abs}}(1)[\text{OF } \alpha], \ \text{of } t \cdot \sigma^*]]$

note \text{rms2} = $\ \text{abs}_{\text{terms}}_{\text{subst}}[\text{OF } \ \text{abs}_{\text{terms}}_{\text{subst}}_{\text{range}}_{\text{disj}}[

\text{OF } \ \text{transaction}_{\text{fresh}}_{\text{subst}}_{\text{has}_{\text{no}}_{\text{pubconsts}}}_{\text{abs}}(2)[\text{OF } \sigma], \ \text{of } t]$

$\ \text{abs}_{\text{terms}}_{\text{subst}}[\text{OF } \ \text{abs}_{\text{terms}}_{\text{subst}}_{\text{range}}_{\text{disj}}[

\text{OF } \ \text{transaction}_{\text{renaming}}_{\text{subst}}_{\text{has}_{\text{no}}_{\text{pubconsts}}}_{\text{abs}}(2)[\text{OF } \alpha], \ \text{of } t \cdot \sigma^*]]$

have "$t \in (\bigcup T \in \text{set} P. \ \text{trms}_{\text{transaction}} T) " \ "\text{fv} (t \cdot \sigma \cdot \alpha \cdot \overline{I}) = \{\}"

using $t_{\in T} \ T \ \text{interpretation}_{\text{grounds}}[\text{OF } \overline{I}(1)]$ by fastforce

moreover have "$\text{wf}_{\text{trm}}_{\text{subst}}_{\text{range}}[\text{of } \sigma, \ \text{OF } \ \text{transaction}_{\text{fresh}}_{\text{subst}}_{\text{is}} _{\text{wf}_{\text{trm}}}[\text{OF } \sigma]$

\text{wf}_{\text{trm}}_{\text{subst}}_{\text{range}}[\text{of } \alpha, \ \text{OF } \ \text{transaction}_{\text{renaming}}_{\text{subst}}_{\text{is}} _{\text{wf}_{\text{trm}}}[\text{OF } \alpha]]$

\text{wf}_{\text{trm}}_{\text{subst}}_{\text{compose}}[\text{of } \sigma \cdot \alpha, \ \text{THEN} \ \text{wf}_{\text{trm}}_{\text{subst}}_{\text{compose}}[\text{OF } \overline{I}(3)]]$

by blast

moreover

have "$t \notin \text{pubval}_{\text{terms}}$"

using $t_{\in T} \ T \ \text{no}_{\text{pubconsts}}_\text{fun}_{\text{set}} \ \text{Fun}_{\text{set}} \ \text{Fun}_{\text{term}} \ \text{Fun}_{\text{subterm}}$

unfolding is_PubConstValue_def is_PubConst_def by fastforce
hence "t · ?θ ∉ pubval_terms"
  using lms1 T_fresh_vars_value_typed
unfolding ξ_elim by auto
hence "t · ?θ · I ∉ pubval_terms"
  using I' \( t \_fv \) pubval_terms_subst[of "t · ?θ" I]
by auto
moreover have "t ∉ abs_terms"
  using t_in_T T_no_abs funs_term_Fun_subterm
unfolding is_Abs_def by force
hence "t · ?θ · I ∉ abs_terms"
  using lms2 T_fresh_vars_value_typed
unfolding ξ_elim by auto
ultimately have 
  "t · ξ \circ s · σ \circ s · α · I ∈ GSMP (\( \bigcup T \in \text{set P. trms_transaction T} \) - (pubval_terms ∪ abs_terms))"
using SMP.Substitution[OF SMP.MP[of t "(\( \bigcup T \in \text{set P. trms_transaction T} \) - (pubval_terms ∪ abs_terms))"]]
subst_subst_compose[of t ?ϑ I] wt_σα
unfolding GSMP_def ξ_elim by fastforce

have 
  "iiklsst A · set I ⊆ GSMP (\( \bigcup T \in \text{set P. trms_transaction T} \) - (pubval_terms ∪ abs_terms))"
using reachable_constraints_GSMP_no_pubvals_abss[OF A_reach P I'(1-5)]
iiklsst_trms iiklsst_subset[of "unlabel A"]
by blast

show "intruder_synth_mod_timpls FP TI (t · ?θ · I · α · 0 (db lsst A I))"
using deduct_FP_if_deduct[OF *** ** * ***] deducts_eq_if_analyzed[OF FP(1)]
intruder_synth_mod_timpls_is_synth_timpl_closure_set[OF TI, of FP]
unfolding a0_def by force
qed

show ?C
proof (intro ballI allI impI)
  fix y s
  assume y: "y ∈ fv_transaction T - set (transaction_fresh T)"
  and s: "select(Var y, Fun (Set s) []) ∈ set (unlabel (transaction_checks T))"
hence "select(Var y, Fun (Set s) []) ∈ set (unlabel (transaction_strand T))"
unfolding transaction_strand_def unlabel_def by auto
hence y_val: "Γ v y = TAtom Value"
  using transaction_selects_are_Value_vars[OF T_wf T_adm'(1)]
by fastforce
have "select(?θ y, Fun (Set s) []) ∈ set (unlabel (transaction_checks T · iiklsst ?θ))"
  using subst_lsst_unlabel_member[OF T_wf S]
by fastforce
hence "(" ξ \circ s · α · 0 · y · I, Fun (Set s) []) ∈ set (db lsst A I)"
using wellformed_transaction_sem_pos_checks[of T_wf I_iis_T_model, of Assign "(" ξ \circ s · α · 0 · y" Fun (Set s) [])"]
by simp
thus "∃ ss. (ξ \circ s · α · 0 · y · I · α · 00 · (db lsst A I) = absc ss ∧ s ∈ ss)"
  using to_abs_alt_def[of "db lsst A I"] 4[of y] y y_val by auto
qed

show ?D
proof (intro ballI allI impI)
  fix y s
  assume y: "y ∈ fv_transaction T - set (transaction_fresh T)"
  and s: "(Var y in Fun (Set s) []) ∈ set (unlabel (transaction_checks T))"
hence "(Var y in Fun (Set s) []) ∈ set (unlabel (transaction_strand T))"
unfolding transaction_strand_def unlabel_def by auto
3.6 Stateful Protocol Verification

hence \(\text{\(y\_val\): } \Gamma v = TAtom Value\)

using transaction_inset_checks_are_Value_vars[OF T_adm]

by fastforce

have "(?\(\theta\) \(y\) in Fun (Set s) []) \(\in\) set (unlabel (transaction_checks T \(\cdot\)_is_T_model ?\(\theta\)))"

using subst_last_unlabel_member[OF s]

by fastforce

hence "(?\(\theta\) \(y\) \(\cdot\) I, Fun (Set s) []) \(\notin\) set (\(\text{db}_{\text{Is}}\), \(A\) I)"

using wellformed_transaction_sem_pos_checks

OF T wf I_is_T_model,

of Check "(?\(\theta\) \(y\) "Fun (Set s) []""

by simp

thus "\exists ss. ?\(\theta\) \(y\) \(\cdot\) I \(\cdot\) \(\alpha\) \(\alpha\) \(0\) \(\text{db}_{\text{Is}}\), \(A\) I) = \text{absc}\ ss \land s \in ss"

using to_abs_alt_def[of "\(\text{db}_{\text{Is}}\), \(A\) I\"

by auto

qed

show \(\exists F\)

proof (intro ballI allI impI)

fix \(y\)

assume \(y\): "\(y\) \(\in\) \text{fv} \text{transaction} \(T\) - set (\(\text{transaction_fresh}\) \(T\))"

and \(s\): "(\(\text{Var} \(y\) \(\not\in\) \text{Fun} \text{Set} s\) []) \(\in\) set (\(\text{unlabel}\) \(\text{transaction_checks}\) \(T\))"

hence "(\(\text{Var} \(y\) \(\not\in\) \text{Fun} \text{Set} s\) []) \(\notin\) set (\(\text{unlabel}\) \(\text{transaction_strand}\) \(T\))"

unfolding transaction_strand_def unlabel_def by auto

hence \(y\_val\): "\(\Gamma v\) \(y\) = TAtom Value"

using transaction_notinset_checks_are_Value_vars[OF T_adm]

by fastforce

have "(?\(\theta\) \(y\) \(\cdot\) I, Fun (Set s) []) \(\notin\) set (\(\text{db}_{\text{Is}}\), \(A\) I)"

using subst_lsst_unlabel_member[OF s]

by fastforce

hence "(?\(\theta\) \(y\) \(\cdot\) I, Fun (Set s) []) \(\in\) set (\(\text{unlabel}\) \(\text{transaction_strand}\) \(T\))"

using wellformed_transaction_sem_neg_checks(2)[OF T wf I_is_T_model, of ""] "(?\(\theta\) \(y\) "Fun (Set s) []"

by simp

moreover have "\text{list_all admissible_transaction_updates} \(P\)"

using Ball_set[of P "\text{admissible_transaction}\"] P(1)

Ball_set[of P \text{admissible_transaction_updates}]

\text{admissible_transaction_is_wellformed_transaction}(3)

by fast

moreover have "\text{list_all wellformed_transaction} \(P\)"

using P(1) Ball_set[of P "\text{admissible_transaction}\"] Ball_set[of P \text{wellformed_transaction}]

\text{admissible_transaction_is_wellformed_transaction}(1)

by blast

ultimately have "(\(\xi\) \(\circ\) \(s\) \(\circ\) \(\alpha\) \(\cdot\) \(I\) \(\cdot\) \(\sigma\) \(\circ\) \(s\) \(\alpha\) \(0\) \(\text{db}_{\text{Is}}\), \(A\) I) = \text{absc}\ ss \land s \notin ss"

using reachable_constraints_db_is_set_args_empty[OF A_reach]

unfolding \text{admissible_transaction_updates_def} by auto

thus "\exists ss. (\(\xi\) \(\circ\) \(s\) \(\circ\) \(\alpha\) \(\cdot\) \(I\) \(\cdot\) \(\sigma\) \(\circ\) \(s\) \(\alpha\) \(0\) \(\text{db}_{\text{Is}}\), \(A\) I) = \text{absc}\ ss \land s \notin ss"

using to_abs_alt_def[of "\(\text{db}_{\text{Is}}\), \(A\) I\]

by auto

qed

show \(\exists E\)

proof (intro ballI allI impI)

fix \(y\)

assume \(y\): "\(y\) \(\in\) \text{fv} \text{transaction} \(T\) - set (\(\text{transaction_fresh}\) \(T\))"

then obtain \(yn\) where \(yn\): "(?\(\theta\) \(y\) \(=\) \(\text{Fun} \text{Val} \(yn\)) \(\cdot\) I)"

using \(\text{abs}\) by moura

hence \(y\_abs\): "(?\(\theta\) \(y\) \(\cdot\) I \(\cdot\) \(\alpha\) \(\alpha\) \(0\) \(\text{db}_{\text{Is}}\), \(A\) I) = \text{abs}\ \(\text{Abs}\ \(\alpha\) \(0\) \(\text{db}_{\text{Is}}\), \(A\) I) \(\cdot\) I)"

by simp

have "\(y\) \(\in\) \text{fv} \text{Is} \text{receive} \(T\) \(\lor\) (\(y\) \(\in\) \text{fv} \text{Is} \text{select} (\(t\), \(s\)) \(\in\) \text{set} (\(\text{unlabel}\) \(\text{transaction_checks}\) \(T\)) \(\land\) \(y\) \(\in\) \text{fv} \(\text{t} \(\cup\) \(\text{fv}\) \(s\)))"

using \text{admissible_transaction_fv_in_receives_or_selects[OF T_adm]} \(y\) by blast

thus "(?\(\theta\) \(y\) \(\cdot\) I \(\cdot\) \(\sigma\) \(\circ\) \(s\) \(\alpha\) \(0\) \(\text{db}_{\text{Is}}\), \(A\) I) \(\in\) \text{abs}\ \(\text{set}\ \text{OCC}\)"

proof
assume "y ∈ fv_{set} (transaction_receive T)"
then obtain ts where ts: "receive(ts) ∈ set (unlabel (transaction_receive T))" "y ∈ fv_{set} (set ts)"
using wellformed_transaction_unlabel_cases[OF T_wf]
by (force simp add: unlabel_def)

have "∃? y . I ∈ ts: "I ∈ set (unlabel (transaction_receive T))" "? y ∈ set ts" "? y ∈ set (unlabel (transaction_receive T))"
using ts
by blast

have "∀ y . I ∈ ts: "I ∈ set (unlabel (transaction_receive T))" "? y ∈ set ts" "? y ∈ set (unlabel (transaction_receive T))"
using t
by blast

next
assume "y ∈ fv_{set} (transaction_checks T) ∧ (∃t s. select(t,s) ∈ set (unlabel (transaction_checks T)) ∧ y ∈ fv t ∪ fv s)"
then obtain t' where "select(t,t') ∈ set (unlabel (transaction_checks T))" "y ∈ fv t ∪ fv t'"
by blast

then obtain l s where "(1,select(Var y, Fun (Set s) []) ) ∈ set (transaction_checks T)"
using admissible_transaction_strand_step_cases[OF T_adm]

unfolding unlabel_def by fastforce
then obtain U where "prefix U[1,select(Var y, Fun (Set s) []) ] (transaction_checks T)"
using in_set_conv_decomp[OF "(1, select(Var y, Fun (Set s) []) ) ∈ set (transaction_checks T)"
by (auto simp add: unlabel_def)

hence "select(Var y, Fun (Set s) []) ∈ set (unlabel (transaction_checks T))"
by (force simp add: prefix_def unlabel_def)

hence "select(? y, Fun (Set s) []) ∈ set (unlabel (transaction_checks T))"
by (force simp add: unlabel_def)

hence "select(? y, Fun (Set s) []) ∈ set (unlabel (transaction_checks T))"
by (force simp add: prefix_def unlabel_def)

by blast

by blast

hence "Fun (Val yn) [] ∈ set (db_{set} A I)"
using yn wellformed_transaction_sem_pos_checks[OF T_wf I_is_T_model, of Assign "? y" "Fun (Set s) []"]
by blast

hence "Fun (Val yn) [] ∈ set (db_{set} A I)"
using db_{set}_in_cases[OF "Fun (Val yn) []"]
by (force simp add: db_{set}_def)

thus ?thesis
using OCC(3) yn abs_in[OF "Fun (Val yn) []" _ "α0 (db_{set} A I)"]
unfolding abs_value_constants_def
by (metis (mono_tags, lifting) mem_Collect_eq subsetCE)

qed

lemma transaction_prop4:
assumes A: reachable_constraints P
and T: "T ∈ set P"
and I: "welltyped_constraint_model I (A@dual_{set} (transaction_strand T :_{set} ω_σ _σ _σ))"
3.6 Stateful Protocol Verification

and $\xi$: "transaction_decl_subst $\xi$ $T"
and $\sigma$: "transaction_fresh_subst $\sigma$ $T$ $A"
and $\alpha$: "transaction_renaming_subst $\alpha$ $P$ $A"
and $P$: "\(\forall T \in \text{set } P. \text{admissible_transaction } T\)"
and $x$: "\(x \in \text{set (transaction_fresh } T)\)"
and $y$: "\(y \in \text{fv_transaction } T \setminus \text{set (transaction_fresh } T)\)"

shows "\(\langle \xi_0, \sigma_0, \alpha_0 \rangle x \cdot I \notin \text{subterms_set (trms_{\text{lst}} (A \cdot \text{lst_I}))} \)" (is $\exists A$)
and "\(\langle \xi_0, \sigma_0, \alpha_0 \rangle y \cdot I \in \text{subterms_set (trms_{\text{lst}} (A \cdot \text{lst_I}))} \)" (is $\exists B$)

proof -
let $?T' = \text{dual_{\text{lst}} (transaction_strand } T \cdot \text{lst}_{\xi} \circ s \circ \sigma \circ s \alpha)"

from $I$ have $I'$: "welltyped_constraint_model $I$ $A"
using welltyped_constraint_model_prefix by auto

have $T_P\_\text{adm}$: "admissible_transaction $T'" when $T'$: "\(T' \in \text{set } P \)" for $T'$
by (meson $T' \in P$)

have $T\_\text{adm}$: "admissible_transaction $T"
by (metis (full_types) $P$ $T$)

note $T\_\text{wf}$ = admissible_transaction_is_wellformed_transaction(1)[OF $T\_\text{adm}$]

have be: "\(\text{bvars}_{\text{lst}} A = \{\}\)"
using $T_P\_\text{adm} \_\text{A}_\text{reach} \ \text{reachable_constraints_no_bvars} \ \text{admissible_transaction_no_bvars}(2)"
by blast

have $T\_\text{no_bvars}$: "\(\text{fv_transaction } T = \text{vars_transaction } T\)"
using admissible_transaction_no_bvars[OF $T\_\text{adm}$]
by blast

note $\xi\_\text{empty}$ = admissible_transaction_decl_subst_empty[OF $T\_\text{adm} \xi$]

have $I\_\text{wt}$: "\(\text{wt} \text{subst } I\)"
by (metis $I$ welltyped_constraint_model_def)

have $x\_\text{val}$: "\(\text{fst x = TAtom Value}\)"
using $x$ admissible_transactionE(14)[OF $T\_\text{adm}$] unfolding list_all_iff by blast

obtain $xn$ where $xn$: "\(\sigma x = \text{Fun (Val } xn) \)"
using $x\_\text{val} \ \text{transaction_fresh_subst_sends_to_val}[OF $\sigma$ $x$ $I\_\text{val} \_\text{TAtom'}(2)[of } x\) by meson

hence $x\_\text{nxn}$: "\(\langle \xi_0, \sigma_0, \alpha_0 \rangle x = \text{Fun (Val } xn) \)"

unfolding subst_compose_def $\xi\_\text{empty}$ by auto

from $xn$ $x\_\text{nxn}$ have $a0$: "\(\langle \xi_0, \sigma_0, \alpha_0 \rangle x \cdot I = \text{Fun (Val } xn) \)"
by auto

have $b0$: "\(T_\nu x = T\text{Atom Value}\)"
using $P \times T$ $\text{protocol_transaction_vars_TAtom_typed}(3)$
by metis

note $0 = a0 \ b0$

have $\sigma\_x\_\text{nin}_A$: "\(\sigma x \notin \text{subterms_set (trms}_{\text{lst}} A)\)"

proof -
have "\(\sigma x \in \text{subterms}\_\text{range } \sigma\)"
by (metis $\sigma\_x\_\text{nin}_A$)

moreover have "\(\forall t \in \text{subterms}\_\text{range } \sigma. t \notin \text{subterms_set (trms}_{\text{lst}} A)\)"
using $\sigma \text{transaction_fresh_subst_def}[of } \sigma T A$ by auto

ultimately show "$?thesis"
by auto

qed

have $\#: "y \notin \text{set (transaction_fresh } T)"
using assms by auto

have **: "y ∈ fv x s (transactionReceive T) ∨ (y ∈ fv x s (transactionChecks T) ∧ 
(∃ t s. select(t,s) ∈ set (unlabel (transactionChecks T))) ∧ y ∈ fv x t s)"
  using * y admissibleTransaction_fv_in_receives_or_selects[OF T_adm]
  by blast

have y_fv: "y ∈ fv x S" using y_fv_transaction unfolding by blast

have y_val: "fst y = TAtom Value" using y(2) Γ x TAtom''(2) by blast

have σ x I fsubterms x (trms x (A x I I))
  proof (rule ccontr)
    assume "¬σ x I fsubterms x (trms x (A x I I))"
    then have a: "σ x I x subterms x (trms x (A x I I))"
      by auto
    then have σ x I in A: "σ x I ∈ subterms x (trms x A) x set I"
      using reachableConstraints_subterms_subst[OF A_reach I P] by blast
    have "∃ u. u = fv x (A x I I) u = σ x"
      proof -
        from σ x I in A have "∃ tu. tu ∈ (subterms x (trms x A)) ∧ tu I = σ x I"
          by force
        then obtain tu where tu: "tu ∈ (subterms x (trms x A)) ∧ tu I = σ x I"
          by auto
        then have "tu ≠ σ x"
          using σ x nin_A by auto
        moreover
        have "∃ u. tu = Var u"
          unfolding xn by (cases tu) auto
        then obtain u where "tu = Var u"
          by auto
        have "u ∈ fv x (A x I I) u = σ x"
          proof -
            have "u ∈ vars x (A x I I)"
              using ⟨tu = Var u⟩ tu varSubterm_trms_set_is_vars_set by fastforce
            then have "u ∈ fv x (A x I I)"
              using be vars_set_is fv set bvset[of "unlabel A"] by blast
            moreover
            have "I u = σ x"
              using ⟨tu = Var u⟩ ⟨tu I = σ x⟩ by auto
            ultimately
            show ?thesis
              by auto
          qed
        then show "∃ u. u ∈ fv x (A x I I) u = σ x"
          by metis
      qed
    then obtain u where u: "u ∈ fv x (A x I I) "I u = σ x"
      by auto
    then have u_TA: "Γ x u = TAtom Value"
      using P(1) T x_val Γ x TAtom''(2)[of x]
      wtSubterm_trm''[OF I_wt, of "Var u"] wtSubterm_trm''[of σ "Var x"]
      transactionFreshSubterm_trms[OF σ] protocolTransaction_vars_TAAtom_TYPED3
      by force
    have "∃ B. prefix B A ∧ u ∉ fv x (B x I I) u ∈ subterms x (trms x (B x I I))"
      using u u_TA
      by (metis welltypedConstraint_model_prefix[OF I])
then obtain $B$ where "prefix $B \ A \ \land \ u \ \notin \ \text{fv}_{\text{set}} \ B \ \land \ \tau \ u \ \in \ \text{subterms}_{\text{set}} \ (\text{trms}_{\text{set}} \ B)"

by blast

moreover have "$\bigcup (\text{subterms} \ \text{`} \ \text{trms}_{\text{set}} \ \text{xs}) \ \subseteq \ \bigcup (\text{subterms} \ \text{`} \ \text{trms}_{\text{set}} \ \text{ys})"

when "prefix $xs \ ys" for $xs$:$ys$::"('fun,'atom,'sets,'lbl) prot_strand"

using that $\text{subterms}_{\text{set}} \ \text{mono} \ \text{trms}_{\text{set}} \ \text{mono} \ \text{unlabel}_{\text{mono}} \ \text{set}_{\text{mono}} \ \text{prefix} \ \text{by} \ \text{metis}

ultimately have "$\exists u \ \in \ \text{subterms}_{\text{set}} \ (\text{trms}_{\text{set}} A)"

by blast

then have "$\sigma \ x \ \in \ \text{subterms}_{\text{set}} \ (\text{trms}_{\text{set}} A)"

using $u$ by auto

then show "False"

using $\sigma \ x \ \notin \ A$ by auto

qed

then show $?A$

using subst_apply_term.simps(1)[of $x$ $\sigma$]

unfolding subst_compose_def xn $\xi \ _\emptyset$ by auto

from ** show $?B$

proof

define $T'$ where "$T' \ \equiv \ \text{transaction}_{\text{receive}} \ T"

define $\vartheta$ where "$\vartheta \ \equiv \ \xi \ _\circ \ \sigma \ _\circ \ \alpha"

assume $y$: "$y \ \in \ \text{fv}_{\text{set}} \ (\text{transaction}_{\text{receive}} \ T)"

hence "$\text{Var} \ y \ \in \ \text{subterms}_{\text{set}} \ (\text{trms}_{\text{set}} \ T')" by (metis $T'_{\text{def}}$ $\text{fv}_{\text{set}} \ _\text{is}_{\text{subterm}_{\text{trms}_{\text{set}}}}$)

then obtain $z$ where $z$: "$z \ \in \ \text{set} \ (\text{unlabel} \ T')" "$\text{Var} \ y \ \in \ \text{subterms}_{\text{set}} \ (\text{trms}_{\text{set}} \ y)"

by (induct $T'$) auto

have "$\text{is}_{\text{Receive}} \ z"

using $\text{Ball}_{\text{set}}$[of "$\text{unlabel} \ T'" $\text{is}_{\text{Receive}}$] $z$(1)

admissible_transaction_is_wellformed_transaction(1)[OF $T_{\text{adm}}$]

unfolding wellformed_transaction_def $T'_{\text{def}}$

by blast

then obtain tys where "$z = \text{receive}(\text{tys})" by (cases $z$) auto

hence tys: "receive($\text{tys} \ \_\text{set} \ \vartheta" C $\in \ \text{set} \ (\text{unlabel} \ (T' \ _\text{set} \ \vartheta))" "$\vartheta \ y \ \in \ \text{subterms}_{\text{set}} \ (\text{set} \ \text{tys} \ _\text{set} \ \vartheta)"

using $z$ subterm_mono unfolding subst_apply_labeled_stateful_strand_def unlabel_def by force+

hence $y_{\text{deduct}}$: "list_all ($\lambda \ t. \ \text{ik}_{\text{set}} \ A \ _\text{set} \ I \ \vdash \ t \ \cdot \ \vartheta \ \cdot \ I" \ \text{tys}"

using transaction_receive_deduct[OF $T_{\text{wf}}$ $\xi \ _\sigma \ _\alpha$] $I$

unfolding $T'_{\text{def}}$ $\vartheta_{\text{def}}$ welltyped_constraint_model_def list_all_iff by auto

obtain ty where ty: "$ty \ \in \ \text{set} \ \text{tys}" "$\vartheta \ y \ \subseteq \ ty \ \cdot \ \vartheta" "$\text{ik}_{\text{set}} \ A \ _\text{set} \ I \ \vdash \ ty \ \cdot \ \vartheta \ \cdot \ I" \ \text{tys}"

using tys $y_{\text{deduct}}$ unfolding list_all_iff by blast

obtain zn where zn: "($\xi \ _\circ \ \sigma \ _\circ \ \alpha$) $y \ \cdot \ I = \text{Fun} \ (\text{Val} \ zn) []"

using transaction_var_becomes_Val[

OF reachable_constraints.step[OF $A_{\text{reach}} \ T \ _\xi \ _\sigma \ _\alpha$] $I \ \xi \ _\sigma \ _P \ T \ y_{\text{fv}} \ y_{\text{val}}$]

by metis

have "($\xi \ _\circ \ \sigma \ _\circ \ \alpha$) $y \ \cdot \ I \ \in \ \text{subterms}_{\text{set}} \ (\text{ik}_{\text{set}} \ A \ _\text{set} \ I)"

using $\text{ty}$ tys(2) $y_{\text{deduct}}$ private_fun_deduct_in_ik[of _ _ "\text{Val} \ zn"]

by (metis $\vartheta_{\text{def}}$ zn subst_mono public.simps(3))

thus $?B$

using $\text{ik}_{\text{set}}$ subst[of "$\text{unlabel} \ A" $I$] unlabel subst[of $A$ $I$]

subterms_{set}_{mono}[OF $\text{ik}_{\text{set}}$ _trms_{set}_{set}_{subset}[of "$\text{unlabel} \ (A \ _\text{set} \ I)"]]

by fastforce

next

assume $y'$: "$y \ \in \ \text{fv}_{\text{set}} \ (\text{transaction}_{\text{checks}} \ T) \ \land \ (\exists \ t. \ \text{select} \ (t, s) \ \in \ \text{set} \ (\text{unlabel} \ (\text{transaction}_{\text{checks}} \ T)) \ \land \ y \ \in \ \text{fv} \ t \ \cup \ \text{fv} \ s)"

then obtain $s$ where $s$: "$\text{select} \ (\text{Var} \ y, s) \ \in \ \text{set} \ (\text{unlabel} \ (\text{transaction}_{\text{checks}} \ T))"

"$\text{fst} \ y = \text{TAtom} \ \text{Value}"

using admissible_transaction_strand_step_cases(1,2)[OF $T_{\text{adm}}$] by fastforce

obtain $z$ zn where zn: "($\xi \ _\circ \ \sigma \ _\circ \ \alpha$) $y = \text{Var} \ z" "$I \ z = \text{Fun} \ (\text{Val} \ zn) []"

\"
using transaction_var_becomes_Val[OF reachable_constraints.step[OF A_reach T ξ σ α I ξ σ α P T y_fv s(2)]]
transaction_decl_fresh_renaming_substs_range[OF A_reach welltyped_constraint_model_def]
transaction_decl_subst_empty_inv[OF [unfolded ξ_empty]]
by auto

have transaction_selects_db_here:
  "∀ n s. select(Var (TAtom Value, n), Fun (Set s) []) ∈ set (unlabel (transaction_checks T))
  ⟷ ((α (TAtom Value, n) · I, Fun (Set s) []) ∈ set (dbsst A I))" unfolding transaction_selects_db[OF T_adm _]
by blast
moreover
have "∃ ss. s = Fun (Set ss) []" using is_Fun_Set_exi by auto
ultimately obtain n ss where nss: "y = (TAtom Value, n)" "s = Fun (Set ss) []" by auto
then have "select(⟨Var (TAtom Value, n), Fun (Set ss) []⟩) ∈ set (unlabel (transaction_checks T))"
using s by auto
then have "(I z, s) ∈ set (dbsst A I)" proof -
  have "((α y · I, s) ∈ set (dbsst A I))" using in_db nss by auto
moreover
have "α y = Var z" using zn _empty * σ by (metis (no_types, opaque_lifting) subst_compose_def subst_imgI subst_to_var_is_var_term.distinct(1) transaction_fresh_subst_def var_comp(2))
then have "α y · I = I z" by auto
ultimately show "(I z, s) ∈ set (dbsst A I)" by auto
qed
then have "∃ t' s'. insert(t',s') ∈ set (unlabel A) ∧ I z = t' · I ∧ s = s' · I" using dbsst_in_cases[of "I z" s "unlabel A" I ""] unfolding dbsst_def by auto
then obtain t' s' where t's': "insert(t',s') ∈ set (unlabel A) ∧ I z = t' · I ∧ s = s' · I" by auto
then have "t' ∈ subterms_set (trmssst A)" by force
then have "t' · I ∈ (subterms_set (trmssst A)) ·set I" by auto
then have "I z ∈ (subterms_set (trmssst A)) ·set I" using t's' by auto
then have "I z ∈ subterms_set (trmssst (A ·sst I))" using reachable_constraints_subterms_subst[OF A_reach welltyped_constraint_model_prefix[OF I P]] by auto
then show ?B using zn(1) by simp
lemma transaction_prop5:
fixes T ξ σ α A I T' a0 a0' ϑ
defines "T' ≡ dual lsst (transaction_strand T · lsst ξ ◦ s σ ◦ s α)"
and "a0 ≡ α0 (db lsst A I)"
and "a0' ≡ α0 (db lsst (A@T') I)"
and "ϑ ≡ λδ x. if fst x = TAtom Value then (absc δ) x else Var x"
assumes A_reach: "A ∈ reachable_constraints P"
and T: "T ∈ set P"
and I: "welltyped_constraint_model I (A@T')"
and ξ: "transaction_decl_subst ξ T"
and σ: "transaction_fresh_subst σ T A"
and α: "transaction_renaming_subst α P A"
and FP:
"analyzed (timpl_closure_set (set FP) (set TI))"
"wf trms (set FP)"
"∀ t ∈ α ik A I. timpl_closure_set (set FP) (set TI) ⊢ c t"
and OCC:
"∀ t ∈ timpl_closure_set (set FP) (set TI). ∀ f ∈ funs_term t. is_Abs f → f ∈ Abs ` set OCC"
"timpl_closure_set (absc ` set OCC) (set TI) ⊆ absc ` set OCC"
and TI:
"set TI = {(a, b) ∈ (set TI) +. a ≠ b}"
and P:
"∀ T ∈ set P. admissible_transaction T"
and step: "list_all (transaction_check (FP, OCC, TI)) P"
support.: "∃ δ ∈ abs_substs_fun ` set (transaction_check_comp (λ _ _. True) (FP, OCC, TI) T).
∀ x ∈ fv_transaction T. Γ v x = TAtom Value →
(ξ ◦ s σ ◦ s α) x · I · α a0 = absc (δ x) ∧
(ξ ◦ s σ ◦ s α) x · I · α a0' = absc (absdupd (unlabel (transaction_updates T)) x (δ x))"
proof -
define comp0 where
"comp0 ≡ abs_substs_fun ` set (transaction_check_comp (λ _ _. True) (FP, OCC, TI) T)"
define check0 where "check0 ≡ transaction_check (FP, OCC, TI) T"
define upd where "upd ≡ transaction_update_check (FP, OCC, TI) T"
define b0 where "b0 ≡ λ x. THE b. absc b = (ξ ◦ s σ ◦ s α) x · I · α a0"

note all_defs = comp0_def check0_def a0_def a0'_def upd_def b0_def ϑ_def T'_def

have A_wftrms: "wftrms (trms set A)"
by (metis reachable_constraints_wftrms, admissible_transactions_wftrms P(1) A_reach)

have I_interp: "interpretation subst I"
and I_wt: "wt subst I"
and I_wftrms: "wftrms (subest_range I)"
by (metis I welltyped_constraint_model_def constraint_model_def, 
metis I welltyped_constraint_model_def, 
metis I welltyped_constraint_model_def constraint_model_def)

have I_is_T_model: "strand_sem_stateful (ik lsst A · set I) (set (db lsst A I)) (unlabel T') I"
using I unlabel_append[of A T'] db lsst_set_is_dbbusset[of "unlabel A" I "[]"]
strand_sem_append_stateful[of "[]" "[]" "unlabel A" "unlabel T" I]
by (simp add: welltyped_constraint_model_def constraint_model_def db lsst_def)

have T_adm: "admissible_transaction T"
using T P(1) Ball_set[of P "admissible_transaction"]
by blast

note T_wf = admissible_transaction_is_wellformed_transaction(1)[OF T_adm]

have T_no_bvars: "fv_transaction T = vars_transaction T" "bvars_transaction T = {}"
using admissible_transaction_no_bvars[OF T_adm] by simp_all

have T_vars_const_typed: "∀x ∈ fv_transaction T. Γv x = TAtom Value ∨ (∃a. Γv x = TAtom (Atom a))" and T_fresh_vars_value_typed: "∀x ∈ set (transaction_fresh T). Γv x = TAtom Value"
using T P protocol_transaction_vars_TAtom_typed(2,3)[of T] by simp_all

note T_vars_absc = admissible_transaction_decl_subst_empty[OF T_adm T]

have wt_σα: "wt_subσa (ξ oₜ_oₜ oₜ α, I)" and wt_σα: "wt_subσa (ξ oₜ_oₜ oₜ α)"
using I wt wt_subσa composition transaction_decl_fresh_renaming_substs_wt[OF ξ σ α]
(by (blast, blast)

have T_vars_vals: "∀x ∈ fv_transaction T. ∃n. (ξ oₜ_oₜ oₜ α) x · I = Fun (Val n) []"
proof
fix x assume x: "x ∈ fv_transaction T"

have "∃n. (σ oₜ oₜ α) x · I = Fun (Val n) []"
proof (cases "x ∈ subst_domain σ")
case True
then obtain n where "σ x = Fun (Val n) []"
using transaction_fresh_subst_sends_to_val[OF σ]
transaction_fresh_subst_domain[OF σ]
by (metis subst_domain)
thus ?thesis by (simp add: subst_compose_def)
next
case False
hence *: "(σ oₜ oₜ α) x = α x" by (auto simp add: subst_compose_def)

obtain y where y: "Γv x = Γv y" "α x = Var y"
using transaction_renaming_subst_wt[OF α]
transaction_renaming_subst_is_renaming(1)[OF α]
by (metis Γ.simps(1) prod.exhaust wt_subσa_def)

hence "y ∈ fvl₁ₙ₁ (transaction_strand T ⊕ₙ₁ σ oₜ α)"
using x * T_no_bvars(2) unlabel_subσa[of "transaction_strand T" "σ oₜ α"]

fvl₁ₙ₁_subst_fv_subσa[of x "unlabel (transaction_strand T)" "σ oₜ α"]
by (auto)

hence "y ∈ fvl₁ₙ₁ (A↑dual)₁ₙ₁ (transaction_strand T ⊕ₙ₁ σ oₜ α)"
using fv₁ₙ₁_unlabel_dual₁ₙ₁_eq[of "transaction_strand T ⊕ₙ₁ σ oₜ α"]

fvl₁ₙ₁_append[of "unlabel A"] unlabel_append[of A]
by (auto)

thus ?thesis

using y ∈ fvl₁ₙ₁ (A↑dual)₁ₙ₁ (transaction_strand T ⊕ₙ₁ σ oₜ α)

fvl₁ₙ₁_append[of "unlabel A"] unlabel_append[of A]

constraint_model_Value_term_is_Val[ OF reachable_constraints.step[OF A_reach T σ α] I[unfolded T'_def] P(1), of y]

admissible_transaction_Value_vars[of T] T_vars_absc
by simp

thus "∀n. (ξ oₜ_oₜ oₜ α) x · I = Fun (Val n) []" using T_vars_absc by simp

qed

have T_vars_absc: "∀x ∈ fv_transaction T. ∃n. (ξ oₜ_oₜ oₜ α) x · I · α.oₜ a₀ = absc n"

using T_vars_vals by fastforce

hence "(absb o b₀) x = (ξ oₜ_oₜ oₜ α) x · I · α.oₜ a₀" when "x ∈ fv_transaction T" for x
using that unfolding absb_def by fastforce

hence T_vars_absc': "t · (absb o b₀) = t · (ξ oₜ_oₜ oₜ α) · I · α.oₜ a₀"

when "fv t ⊆ fv_transaction T" "∃n T. Fun (Val n) T ∈ substterms t" for t
using that(1) abs_term_subσ_eq'[OF that(2), of "ξ oₜ_oₜ oₜ α, I" a₀ "absb o b₀"]

subσa_compose[of "ξ oₜ_oₜ oₜ α, I"] substσa_compose[of t "ξ oₜ_oₜ oₜ α, I"]

by fastforce

have "∃δ ∈ comp₀. ∀x ∈ fv_transaction T. fst x = TAtom Value → b₀ x = δ x"
proof -
  let ?C = "set (unlabel (transaction_checks T))"
let ?xs = "fv_transaction T = set (transaction_fresh T)"

note * = transaction_prop3[OF A_reach T I[unfolded T'_def] ⋆ σ α FP OCC TI P(1)]

have **: 
  "\(\forall x \in set (transaction_fresh T). b0 x = \{\}\)"
  "\(\forall t \in trms_{set} (transaction_receive T). intruder synth_mod_timpls FP TI (t \cdot \emptyset b0)\)"
(is ?B)
proof - 
  show ?B
  proof (intro ballI impI)
    fix t assume t: "t \in trms_{set} (transaction_receive T)"
    hence t': "fv t \subseteq fv_transaction T" "\(\exists n. T. Fun (Val n) T \subseteq\) subterms t"
    using trms_transaction_unfold[of T] vars_transaction_unfold[of T]
    trms_{set} fv_vars_{set} subset[of t "unlabel (transaction_strand T)"
    admissible_transactions_no_Value_consts'[OF T_adm]
    wellformed_transaction_send_receive_fv_subset(1)[OF T_wf t(1)]
    by blast

    have "intruder synth_mod_timpls FP TI (t \cdot (absc \circ b0))"
    using t(1) t' *(2) T_vars_absc'
    by (metis a0_def)
    moreover have "(absc \circ b0) x = (\emptyset b0) x" when "x \in fv t" for x
    using that T P admissible_transaction_Value_vars[of T]
    <fv t \subseteq fv_transaction T> \(\Gamma _n \cdot \text{TAtom'}(2)[of x] \)
    unfolding ϑ_def by fastforce
    hence "t \cdot (absc \circ b0) = t \cdot \emptyset b0"
    using term_subst_eq[of t "absc \circ b0" "\emptyset b0"] by argo
    ultimately show "intruder synth_mod_timpls FP TI (t \cdot \emptyset b0)"
    using intruder synth.simps[of "set FP"] by (cases "t \cdot \emptyset b0")
  qed
  qed (simp add: *(1) a0_def b0_def)

have ***: 
  "\(\forall x \in ?xs. \forall s. \text{select}(\text{Var } x, \text{Fun}(\text{Set} s) []) \in ?C \rightarrow s \in b0 x)\)"
  "\(\forall x \in ?xs. \forall s. (\text{Var } x \in \text{Fun}(\text{Set} s) []) \in ?C \rightarrow s \in b0 x)\)"
  "\(\forall x \in ?xs. \forall s. (\text{Var } x \notin \text{Fun}(\text{Set} s) []) \in ?C \rightarrow s \notin b0 x)\)"
  "\(\forall x \in ?xs. \text{fst } x = \text{TAtom Value} \rightarrow b0 x \in \text{set OCC})\)

unfolding a0_def b0_def
using *(3,4) apply (force, force)
using *(5) apply force
using *(6) admissible_transaction_Value_vars[OF bspec[OF P T]] by force

show ?thesis
using transaction_check_comp_in[OF T_adm ***[unfolded ϑ_def] ***]
unfolding comp0_def
by metis

qed

hence 1: "\(\exists \emptyset \in \text{comp0}. \forall x \in \text{fv_transaction T}. \)
  \(\text{fst } x = \text{TAtom Value} \rightarrow (\xi o_0 o_\alpha \alpha) x : I \cdot_\alpha a0 = \text{absc}(\emptyset x)\)"

using T_vars_absc unfolding b0_def a0_def by fastforce

obtain δ where δ:
  "\(\delta \in \text{comp0}" \ "\forall x \in \text{fv_transaction T}. \text{fst } x = \text{TAtom Value} \rightarrow (\xi o_0 o_\alpha \alpha) x : I \cdot_\alpha a0 = \text{absc}(\delta x)\)"

using 1 by moura

have 2: "\(\emptyset x : I \cdot_\alpha a00 (\text{db}_{\text{isst}} (\text{dual}_{\text{isst}} (A \cdot_\alpha x) \emptyset I D)) \text{ D} = \text{absc}(\text{absdbupd}(\text{unlabel} A) x d)\)"
  when "\(\emptyset x : I \cdot_\alpha a00 \text{ D} = \text{absc} d\)"
and "\(\forall t. \text{insert}(t, u) \in \text{set}(\text{unlabel} A) \rightarrow (\exists y. s. t = \text{Var } y \land u = \text{Fun}(\text{Set} s) [])\)"
and "\(\forall t. \text{delete}(t, u) \in \text{set}(\text{unlabel} A) \rightarrow (\exists y. s. t = \text{Var } y \land u = \text{Fun}(\text{Set} s) [])\)"
and "\(\forall y \in \text{fv}_{\text{isst}} A. \emptyset x : I = \emptyset y : I \rightarrow x = y\)"
and "\(\forall y \in \text{fv}_{\text{isst}} A. \exists n. \emptyset y : I = \text{Fun}(\text{Val } n) []\)"
and x: "\(\emptyset x : I = \text{Fun}(\text{Val } n) []\)"
3 Stateful Protocol Verification

and D: "\(d \in D \exists s. e = \text{Fun } (\{s\}) []\)"
for A: "\("\text{'fun}, \text{'atom}, \text{'sets}, \text{'lbl}\) \text{prot_strand}\) and \(x \notin D \land d \in D\) using
that (2,3,4,5)

proof (induction A rule: List.rev_induct)
case (snoc a A)
then obtain l b where \(a = (l,b)\) by (metis surj_pair)

have IH: "\(\alpha _0 (\text{db}' _{\text{last}} (\text{dual}_\text{last} (A \cdot \text{last} \emptyset)) I D) n = \text{abedbupd (unlabel A)} x d\)"
using snoc unlabeled_append[of A "[a]" ] a x
by (simp del: unlabeled_append)

have b_prem: "\(\forall y \in fv_{\text{step}} b. \emptyset x \cdot I = \emptyset y \cdot I \rightarrow x = y\)"
using db_filter[of "unlabel (dual A)""]

have \(*\): "\(\text{filter is_Update (unlabel (dual}_\text{last} (A \cdot \text{last} \emptyset))) = \text{filter is_Update (unlabel (dual}_\text{last} (A @ [a])))}\"
when "\(-\text{is_Update b}\)" using

by (cases b, simp_all add: dual_last_def unlabeled_def subst_apply_labeled_stateful_strand_def)+

note ** = * abedbupd_filter[of "unlabel (A@[a])"]

abedbupd_filter[of "unlabel A"]
dual_last_filter[of "unlabel (dual_last (A@[a] @ [a]))"]
dual_last_filter[of "unlabel (dual_last (A @ [a]))"]

note **** = **(2,3) dual_last_subst_snoc[of A a \emptyset] unlabeled_append[of "unlabel (dual_last (A _last \emptyset))" dual_last_app[of "unlabel (dual_last A _last \emptyset)" "unlabel (dual_last A _last \emptyset)"]]

have "\(\alpha _0 (\text{db}' _{\text{last}} (\text{dual}_\text{last} (A@[a] @ [a])) I D) n = \text{abedbupd (unlabel (A@[a]))} x d\)"
using ** ****

proof (cases b)
case (Insert t t')
then obtain y s m where \(y = \text{Var y} \quad t = \text{Fun } (\{s\}) []\) \("y \cdot I = \text{Fun } (\text{Val m}) []\)"
using snoc_prem(1) b_prem(2) a by (fastforce simp add: unlabeled_def)
hence a': "\(\text{db}' _{\text{last}} (\text{dual}_\text{last} (A@[a] @ [a])) I D = \text{List.insert } ((\text{Fun } (\text{Val m}) [], \text{Fun } (\text{Set s}) []) (\text{insert}(\emptyset y, \text{Fun } (\text{Set s}) []) I D)\)"
"\(\text{unlabel (dual}_\text{last} a _\text{last} \emptyset) = \text{insert}(\emptyset y, \text{Fun } (\text{Set s}) [])\)"
"\(\text{unlabel [a] = insert (Var y, Fun (Set s [])])}\)
using **** Insert by simp_all

show \(?thesis\)

proof (cases "x = y")
case True
hence "\(\emptyset x \cdot I = \emptyset y \cdot I\)" by simp
hence "\(\alpha _0 (\text{db}' _{\text{last}} (\text{dual}_\text{last} (A@[a] @ [a])) I D) n = \text{insert s } (\alpha _0 (\text{db}' _{\text{last}} (\text{dual}_\text{last} (A @ [a])) I D) n)\)"
by (metis (no_types, lifting) y(3) a'(1) x dual_last_subst_to_abs_list_insert')
thus \(?thesis\) using True IH a'(3) abedbupd_append[of "unlabel A"] by (simp add: unlabeled_def)
next
case False
hence "\(\emptyset x \cdot I \neq \emptyset y \cdot I\)" using b_prem(1) y insert by simp
hence "\(\alpha _0 (\text{db}' _{\text{last}} (\text{dual}_\text{last} (A@[a] @ [a])) I D) n = \alpha _0 (\text{db}' _{\text{last}} (\text{dual}_\text{last} (A @ [a])) I D) n\)"
by (metis (no_types, lifting) y(3) a'(1) x dual_last_subst_to_abs_list_insert)
thus \(?thesis\) using False IH a'(3) abedbupd_append[of "unlabel A"] by (simp add: unlabeled_def)
qed

next
case (Delete t t')
then obtain y s m where \(y = \text{Var y} \quad t' = \text{Fun } (\{s\}) []\) \("y \cdot I = \text{Fun } (\text{Val m}) []\)"
using snoc_prem(2) b_prem(2) a by (fastforce simp add: unlabeled_def)
3.6 Stateful Protocol Verification

hence a': "db'_isx (db'_isx (A@[a] 'isx @)) I D = 
List.removeAll ((Fun (Val m) [] , Fun (Set s) [])) (db'_isx (db'_isx A 'isx @) I D)"
"unlabel [db'_isx a 'isx @] = [delete(@ y, Fun (Set s) [])]"
"unlabel [a] = [delete(Var y, Fun (Set s) [])]"
using **** Delete by simp_all

have "\exists S. and d = Fun (Set s) []" when "d ∈ set (db'_isx (db'_isx A 'isx @) I D)" for d
using snoc.prems(1,2) db'_isx db'_isx_set_ex[OF that(1)_] by (simp add: unlabel_def)
moreover {
  fix t::"('fun,'atom,'sets,'lbl) prot_term"
  and D:="(('fun,'atom,'sets,'lbl) prot_term × ('fun,'atom,'sets,'lbl) prot_term) list"
  assume "d ∈ set D. \exists s. d = Fun (Set s) []"
  hence "removeAll (t, Fun (Set s) []) D = filter (λd. \exists S. d = (t, Fun (Set s) S)) D" by (induct D) auto
} ultimately have a'':
"List.removeAll ((Fun (Val m) [], Fun (Set s) []) (db'_isx (db'_isx A 'isx @) I D) = 
filter (λd. \exists S. d = (Fun (Val m) [], Fun (Set s) )) (db'_isx (db'_isx A 'isx @) I D)"
by simp

show ?thesis

proof (cases "x = y")
  case True
  hence "\emptyset x : I = \emptyset y : I" by simp
  hence "\emptyset 0 (db'_isx (db'_isx (A@[a] 'isx @))) I D) n = 
\emptyset 0 (db'_isx (db'_isx (A 'isx @))) I D) n - \{a\}"
  using y(3) a' a'(1) x by ( simp add: db'_isx_subst_to_abs_list_remove_all')
  thus ?thesis using True IH a'(3) absdbupd_append[of "unlabel A'"] by (simp add: unlabel_def)
next
  case False
  hence "\emptyset x : I \neq \emptyset y : I" using b_prems(1) y Delete by simp
  hence "\emptyset 0 (db'_isx (db'_isx (A@[a] 'isx @))) I D) n = \emptyset 0 (db'_isx (db'_isx (A 'isx @))) I D) n"
  by (metis (no_types, lifting) y(3) a'(1) x dual_isx_subst_to_abs_list_remove_all)
  thus ?thesis using False IH a'(3) absdbupd_append[of "unlabel A'"] by (simp add: unlabel_def)
qed

qed simp_all

thus ?case by (simp add: x:)

qed (simp add: that(1))

have 3: "x = y"
  when xy: "(ξ o, σ o, α) x : I = (ξ o, σ o, α) y : I" "x ∈ fv_transaction T" "y ∈ fv_transaction T"
  for x y
proof -
  have "x \notin set (transaction_fresh T) \implies y \notin set (transaction_fresh T) \implies ?thesis"
  using xy adm iss transaction_strand_sem fv_ineq[OF T_adm T_is_T_model[unfolded T_def]]
  by fast
moreover {
  assume *: "x ∈ set (transaction_fresh T)" "y ∈ set (transaction_fresh T)"
  hence "I, x = TAtom Value" "I, y = TAtom Value"
  using T_fresh_vars_valueTyped by (blast, blast)
  then obtain xn yn where "σ x = Fun (Val xn) []" "σ y = Fun (Val yn) []"
  using * transaction_fresh_subst_sends_to_val[OF σ] by meson
  hence "σ x = σ y" using that(1) _empty by (simp add: subst_compose)
  moreover have "inj_on σ (subst_domain σ)" "x ∈ subst_domain σ" "y ∈ subst_domain σ"
  using * σ unfolding transaction_fresh_subst_def by auto
  ultimately have ?thesis unfolding inj_on_def by blast
} moreover have False when "x ∈ set (transaction_fresh T)" "y \notin set (transaction_fresh T)"
  using that(2) xy N_no_bvars adm iss Value_vars[OF bspec[OF P T], of y]
  transaction_prop4[OF A_reach T I[unfolded T_def] σ α P that(1), of y]
  by auto
moreover have False when "x \notin set (transaction_fresh T)" "y ∈ set (transaction_fresh T)"
  using that(1) xy N_no_bvars adm iss Value_vars[OF bspec[OF P T], of x]
  transaction_prop4[OF A_reach T I[unfolded T_def] σ α P that(2), of x]

229
by fastforce
ultimately show ?thesis by metis

have 4: "∃ y s. t = Var y ∧ u = Fun (Set s) []" when "insert(t, u) ∈ set (unlabel (transaction_strand T))" for t u using that admissible_transaction_strand_step_cases(3)[OF T_adm] T_wf by blast

have 5: "∃ y s. t = Var y ∧ u = Fun (Set s) []" when "delete(t, u) ∈ set (unlabel (transaction_strand T))" for t u using that admissible_transaction_strand_step_cases(3)[OF T_adm] T_wf by blast

have 6: "∃ n. (ξ ◦ s σ ◦ s α) y · I = Fun (Val n) []" when "y ∈ fv_transaction T" for y using that by (simp add: T_vars_vals)

have 7: "∃ s. snd d = Fun (Set s) []" when "d ∈ set (db lsst A I)" for d using that reachable_constraints_db lsst_set_args_empty[OF A_reach]

unfolding admissible_transaction_updates_def by (cases d) simp

have "(ξ ◦ s σ ◦ s α) x · I · α a0' = absc (upd δ x)" when "x ∈ fv_transaction T" "fst x = TAtom Value" for x proof -

have "(ξ ◦ s σ ◦ s α) x · I · α a0 (db lsst (transaction_strand T) I) (db lsst A I) = absc (absdupd (unlabel (transaction_strand T)) x (δ x))" using 2[of "ξ ◦ s σ ◦ s α" x "db lsst A I" "delta x" "transaction_strand T"] 3[of _ x(1)] 4 5 6[of that(1)] 6 7 x δ(2)

unfolding admissible_transaction_updates_def by blast

thus ?thesis using δ Γ I_v TAtom''(2) unfolding all_defs by blast

qed

thus ?thesis using δ Γ L TAtom'' unfolding all_defs db lsst_def by force

qed

lemma transaction_prop6:
fixes T ξ σ α A I T' a0 a0'
defines "T' ≡ dual lsst (transaction_strand T :lsst (ξ ◦ s σ ◦ s α))"
and "a0 ≡ α0 (db lsst A I)"
and "a0' ≡ α0 (db lsst (A @ T') I)"

assumes A_reach: "A ∈ reachable_constraints P"
and T: "T ∈ set P"
and I: "welltyped_constraint_model I (A @ T')"
and ξ: "transaction_decl_subst ξ T A" and σ: "transaction_fresh_subst σ T A" and α: "transaction_renaming_subst α P A" and FP:
"analyzed (timpl_closure_set (set FP) (set TI))"
"wftrms (set FP)"
"∀ t ∈ α0s A I. timpl_closure_set (set FP) (set TI) ⊢ c t"

and OCC:
"∀ t ∈ timpl_closure_set (set FP) (set TI). ∀ f ∈ funs_term t. is_Abs f → f ∈ Abs ` set OCC"
"timpl_closure_set (absc ` set OCC) (set TI) ⊆ absc ` set OCC"
"oc vars A I ⊆ absc ` set OCC"

and TI:
"set TI = {(a,b) ∈ (set TI)+. a ≠ b}"
and $P$:

> "∀T ∈ set P. admissible_transaction T"

and step: "list_all (transaction_check (FP, OCC, TI)) P"

shows "∀t ∈ timpl_closure_set ($α_1$, $A$ $I$) ($α_1$, $A$ $T$ ($ξ_0$, $σ_0$, $α$) $I$).

proof -

- define comp0 where

> "comp0 ≡ abs_substs_fun ∘ set (transaction_check_comp ($λ_-$ _. True) (FP, OCC, TI) T)"

- define check0 where "check0 ≡ transaction_check (FP, OCC, TI) T"

- define upd where "upd ≡ absdbupd (unlabel (transaction_updates T)) x (δ x)"

- define $θ$ where "$θ ≡ λδ x. if fst x = TAtom Value then (abs ∘ δ) x else Var x"

have $T$_adm: "admissible_transaction T" using $T$ P(1) by metis

note $T$ _wf = "admissible_transaction_is_wellformed_transaction(1)[OF $T$ _adm]

have $θ$ _prop: "$θ σ x = abs (∑σ x)" when "$T$ _v x = TAtom Value" for $x σ x$

using that $Γ_v$ $θ$ _prop(2)[of $x$] unfolding $θ$ _prop by simp

have 0: "∃δ ∈ comp0. ∀x ∈ fv_transaction T. Γv x = TAtom Value →

> (ξ $o_0$, $σ_0$, $α_0$) x · $I$ · $α$ = abs (∑δ x) ∧

> (ξ $o_0$, $σ_0$, $α_0$) x · $I$ · $α′$ = abs (∑upd δ x)"

using transaction_prop5[OF $A$ _reach T $I$[unfolded $T$ _def] $ξ$ $σ$ $α$ FP OCC TI $P$ step]

unfolding a0_def a0′_def T'_def upd_def comp0_def by blast

have 1: "((δ x, upd δ x) ∈ set (transaction_fresh T)) + x ∈ fv_transaction T" "x ∈ set (transaction_fresh T)"

for $x δ$

using $T$ that step Ball_set[of $P$ "transaction_check (FP, OCC, TI)"

transaction_prop1[of δ "$λ_-$ _. True" FP OCC TI $P$ $x$ $T$]

unfolding upd_def comp0_def transaction_check_def by blast

have 2: "upd δ x ∈ set OCC"

when "δ ∈ comp0" "x ∈ fv_transaction T" "fst x = TAtom Value" for $x δ$

using $T$ that step Ball_set[of $P$ "transaction_check (FP, OCC, TI)"

$T$ _adm $FP$ OCC TI transaction_prop2[of δ "$λ_-$ _. True" FP OCC TI $T$ $x$]

unfolding upd_def comp0_def transaction_check_def by blast

obtain $δ$ where $δ$:

> "δ ∈ comp0"

> "∀x ∈ fv_transaction T. Γv x = TAtom Value →

> (ξ $o_0$, $σ_0$, $α_0$) x · $I$ · $α$ = abs (∑δ x) ∧

> (ξ $o_0$, $σ_0$, $α_0$) x · $I$ · $α′$ = abs (∑upd δ x)"

using $θ$ by moura

have "∃x. ab = (∑δ x, upd δ x) ∧ x ∈ fv_transaction T - set (transaction_fresh T) ∧ δ x ≠ upd δ x" when ab: "ab ∈ α_1 ∧ $A$ $T$ ($ξ_0$, $σ_0$, $α$) $I$" for ab

proof -

obtain a b where ab': "ab = (a,b)" by (metis surj_pair)

then obtain $x$ where $x$:

> "a ≠ b" "x ∈ fv_transaction T" "x ∉ set (transaction_fresh T)"

231
lemma reachable_constraints_covered_step:
  fixes A::"('fun,'atom,'sets,'lbl) prot_constr"
  assumes A_reach: "A ∈ reachable_constraints P" and T: "T ∈ set P"
  and I: "welltyped_constraint_model I (A@dual, transf (transaction_strand T @transf \xi \sigma \alpha))" and x: "transaction_decl_subst \xi T" and σ: "transaction_fresh_subst σ T A" and α: "transaction_renaming_subst α P A"
  shows "∀ a. a ∈ fv T \implies (∃ t. subterms t) (∃ T ∈ trms_isat (transaction_send T) for t using that admissible_transactions_no_Value_consts'[OF T_adm] trms_transaction_unfold[of T]) by blast
  show ?D using 2[OF δ(1)] δ(2) I̸_TAtom''(2) unfolding a0'_def T'_def by blast
  show ?C using 3 abs_term_subst_eq'[OF 4 5] by simp
  qed
and FP:
"analyzed (timpl_closure_set (set FP) (set TI))"
"w\textsubscript{terms} (set FP)"
"\(\forall t \in \alpha_{\text{abs}} \cdot A \cdot I.\) timpl_closure_set (set FP) (set TI) \vdash_c t"
"ground (set FP)"
and OCC:
"\(\forall t \in \text{timpl_closure_set} (\text{set FP}) (\text{set TI}).\) \(\forall f \in \text{fun}_\text{term} t.\) is_Abs f \(\rightarrow f \in \text{Abs} \cdot \text{set OCC}\)
"\(\text{timpl_closure_set} (\text{set Abs} \cdot \text{set OCC}) (\text{set} \text{TI}) \subseteq \text{abs} \cdot \text{set OCC}\)
"\(\forall \alpha, \alpha_{\text{abs}} \cdot A \cdot I \subseteq \text{abs} \cdot \text{set OCC}\)
and TI:
"\(\text{set} \text{TI} = \{(a, b) \in (\text{set} \text{TI})^+ . \ a \neq b\}\)
and P:
"\(\forall A \cdot I \in \text{set} \text{P} .\) admissible_transaction T"
and transactions_covered: "list_all (transaction_check (FP, OCC, TI)) P"
shows "\(\forall t \in \alpha_{\text{abs}} (\text{A\textunderscore d\textunderscore dual\textunderscore i\textunderscore set} (\text{transaction\textunderscore strand} T \cdot \text{is\textunderscore set} \xi \cdot o \cdot \sigma \cdot o \cdot \alpha)) \cdot I)\). timpl_closure_set (set FP) (set TI) \vdash_c t" (im ?A)
and "\(\forall \alpha, \alpha_{\text{abs}} (\text{A\textunderscore d\textunderscore dual\textunderscore i\textunderscore set} (\text{transaction\textunderscore strand} T \cdot \text{is\textunderscore set} \xi \cdot o \cdot o \cdot \sigma \cdot o \cdot \alpha)) \cdot I \subseteq \text{abs} \cdot \text{set OCC}\) (im ?B)
proof -

note step_props = transaction_prop6[OF A\textunderscore reach T I \xi \cdot \sigma \cdot \alpha FP(1,2,3) OCC TI P transactions\_covered]

define T' where "T' \equiv \text{dual\textunderscore i\textunderscore set} (\text{transaction\textunderscore strand} T \cdot \text{is\textunderscore set} \xi \cdot o \cdot \sigma \cdot o \cdot \alpha)"
define a0 where "a0 \equiv \alpha (\text{db\textunderscore i\textunderscore set} A) I)"
define a' where "a' \equiv \alpha (\text{db\textunderscore i\textunderscore set} (\text{A\textunderscore d\textunderscore T'}) I)"

define vals where "vals \equiv \lambda S::\{\text{fun},\text{atom},\text{sets},\text{lbl}\} . \text{prot\textunderscore constr}.\{t \in \text{subterms\_set} (\text{trms\_set} S) \cdot \text{is\textunderscore set} I.\} \exists n. t = \text{Fun} (\text{Val} n) []"

define vals\_sym where "vals\_sym \equiv \lambda S::\{\text{fun},\text{atom},\text{sets},\text{lbl}\} . \text{prot\textunderscore constr}.\{t \in \text{subterms\_set} (\text{trms\_set} S) . \} (\exists n. t = \text{Var} (\text{TAtom Value}, m))\}"

have I\_wt: "\(\text{wt\_\textunderscore\_\_set} I\) by (metis I welltyped\_constraint\_model\_def)

have I\_grounds: "\(\text{fv} (t \cdot I) = \{\}\) for t using I interpretation\_grounds[of I]

unfolding welltyped\_constraint\_model\_def constraint\_model\_def by auto

have wt\_\sigma\alpha I: "\(\text{wt\_\_\_set} (\xi o o o o o o I)\) and wt\_\sigma\alpha: "\(\text{wt\_\_\_set} (\xi o o o o o o I)\)"

using I\_wt wt\_\_\_set compose transaction\_decl\_fresh\_renaming\_subsets\_wt[of \(\xi o o\)]

by (blast, blast)

have "\(\forall t \in \text{set} \text{P}.\) bvars\_transaction T = \{\}"

using P admissible\_transaction\_E(4) by metis

hence A\_no\_bvars: "\(\text{bvars\_\_set} A = \{\}\)"

using reachable\_constraints\_no\_bvars[of A\_reach] by metis

have I\_vals: "\(\exists n. \ I \cdot (\text{TAtom Value}, m) = \text{Fun} (\text{Val} n) []\)"

when "(\text{TAtom Value}, m) \in \text{fv\_\_set} A" for m

using constraint\_model\_Value\_term\_is\_Val'[OF A\_reach welltyped\_constraint\_model\_def constraint\_model\_prefix[of I] P(1)]

A\_no\_bvars vars\_\_set is\_fv\_\_set bvars\_\_set[of "unlabel A"] that by blast

have vals\_sym\_vals: "t \cdot I \in \text{vals} A" when t: "t \in \text{vals\_sym} A" for t

proof (cases t)
  case (Var x)

  then obtain n where "x = (\text{TAtom Value}, m)" using t unfolding vals\_sym\_def by blast

  moreover have "t \in \text{subterms\_set} (\text{trms\_set} A)" using t unfolding vals\_sym\_def by blast

  hence "t \cdot I \in \text{subterms\_set} (\text{trms\_set} A) \cdot set I = \exists n. I \cdot (\text{Var Value}, m) = \text{Fun} (\text{Val} n) []"

  using Var * I\_vals[of m] var\_\_subterm\_trms\_\_set is\_vars\_\_set[of x "unlabel A"]

  using I\_\text{TAtom} of Value m reachable\_constraints\_Value\_vars\_\_set\_are\_fv[of A\_reach P(1), of x]

  by blast

  ultimately show ?thesis using Var unfolding vals\_def by auto

next
case (Fun f T)
then obtain a where "t = Val n" "T = []" using t unfolding vals_sym_def by blast
moreover have "t ∈ subterms_set (trms_set A)" using t unfolding vals_sym_def by blast
hence "t · I ∈ subterms_set (trms_set A) · set I" using Fun by blast
ultimately show thesis using Fun unfolding vals_def by auto
qed

have vals vals_sym: "$3. s ∈ vals A ∧ t = s · I" where "t ∈ vals A" for t
using that constraint_model_Val_is_Value_term[OF I]
unfolding vals_def vals_sym_def by fast

have T_adm: "admissible_transaction T" by (metis P(1) T)
note T_wf = admissible_transaction_is_wellformed_transaction(1)[OF T_adm]

have 0:
"\alpha_{ik} (A0') I = (\alpha_{k=x} A · set I) · \alpha_{set a0'} ∪ (\alpha_{k=x} T' · set I) · \alpha_{set a0''}"
"\alpha_{vals} (A0') I = vals A · \alpha_{set a0'} ∪ vals T' · \alpha_{set a0''}"
by (metis abs_value_constants_append[of A T' I] a0'_def_def, metis abs_value_constants_append[of A T' I] a0'_def vals_def)

have 1: "\alpha_{ik(x)} T' · set I) · \alpha_{set a0'} =
(trms_set (transaction_send T) · set (\xi · \sigma · \alpha) · set I) · \alpha_{set a0''}"
by (metis T'_def dual_transaction_ik_is_transaction_send''[OF T_wf])

have 2: "bvars_set (transaction_strand T) ∩ subst_domain \xi = {}"
bvars_set (transaction_strand T) ∩ subst_domain \sigma = {}
bvars_set (transaction_strand T) ∩ subst_domain \alpha = {}
using admissible_transactionE(4)[OF T_adm] by blast+

have "vals T' ⊆ (\xi · \sigma · \alpha) · fv_transaction T · set I"
proof
  fix t assume "t ∈ vals T'"
  then obtain a where s: "s ∈ subterms_set (trms_set T')" "t = s · I" "t = Fun (Val n) []"
  unfolding vals_def by fast

  then obtain u where u: "u ∈ subterms_set (trms_set (transaction_strand T))"
  "s = u · (\xi · \sigma · \alpha)"
  using transaction_decl_fresh_renaming_substs_trms[OF \xi \sigma 2]
  trms_set_unlabel_dual_set_eq[of "transaction_strand T · set I · \alpha"]
  unfolding T'_def by blast

  have 2*: "t = u · (\xi · \sigma · \alpha) · fv_transaction T · set I" by (metis subst_subst_compose s(2) u(2))
  then obtain x where x: "u = Var x" using s(3) admissible_transactions_no_Value_consts(1)[OF T_adm u(1)] by (cases u) force+
  hence 2**: "x ∈ vars_transaction T" by (metis u(1) var_subterm_trms_is-vars set)

  have 1': "T, x = TAtom Value"
  using * x s(3) wt_subst_trm''[OF wt_oaI, of u]
  by simp
  thus "t ∈ (\xi · \sigma · \alpha) · fv_transaction T · set I" using admissible_transaction_Value_vars_are_fv[OF T_adm 2*] x *
  by (metis subst_comp_set_image rev_image_eqI subst_apply_term.simps(1))
qed

hence 3: "vals T' · \alpha_{set a0'} ⊆ (\xi · \sigma · \alpha) · fv_transaction T · set I) · \alpha_{set a0''}"
by (simp add: abs_apply_terms_def image_mono)

have "t · I · \alpha_{set a0'} ∈ timpl_closure_set (\alpha_{ik} A I) (\alpha_{t,i} A T (\xi · \sigma · \alpha · I))" when "t ∈ ik_set A" for t
using that abs_in[OF imageI[OF that]]
\alpha_{ik} covers_\alpha ik[OF A_reach T I \xi \sigma \alpha P(1)]
timpl_closure_set_mono[of "t · I · \alpha_{set a0'}" "\alpha_{ik} A I" "\alpha_{t,i} A T (\xi · \sigma · \alpha · I)"
"t · I · \alpha_{set a0''}"
"\alpha_{ik} A I" "\alpha_{t,i} A T (\xi · \sigma · \alpha · I)"
234
unfolding a0_def a0'_def T'_def abs_intruder_knowledge_def by fast

hence A: "A@[th] I ⊆
timpl_close_set (α_i [A I]) (α_i [A T (ξ o s σ o s α) I])
∪
(t s transaction_send T) (\set (ξ o s σ o s α)) T \set I α set a0'"
using 0(1) 1 by (auto simp add: abs_apply_terms_def)

have "t ∩ A α a0' ∈ timpl_close_set (t ∩ A α a0) (α_i [A T (ξ o s σ o s α) I])"
when t: "t ∈ vals_sym A" for t

proof -
  have "(\exists n. t = Fun (Val n)) ∨
      (∃ n. t = Var (TAtom Value, n) ∧ (TAtom Value, n) ∈ fv set A)"
(is "?P ∨ ?Q")
using t var_subterms_trms is_vars_set[of _ "unlabel A"]
Γν TAtom[Value] reachable_constraints_Value_vars are fv[OF A reach P(1)]
unfolding vals_sym_def by fast

next
assume ?Q
thus ?thesis
using α_i_covers_con_val[of _ "unlabel A"]
unfolding a0_def a0'_def T'_def by fastforce

qed

moreover have "t ∩ A α a0' ∈ α vals_sym A I"
when t: "t ∈ vals_sym A" for t
using that abs in vals_sym vals

unfolding a0_def abs_value_constants_def vals_sym_def vals_def
by (metis mono_tags, lifting)

ultimately have "t ∩ A α a0' ∈ timpl_close_set (α vals A I) (α_i [A T (ξ o s σ o s α) I])"
when t: "t ∈ vals_sym A" for t
using t timpl_close_set mono[of "t ∩ A α a0" "α vals A I" "α_i [A T (ξ o s σ o s α) I]"
"α_i [A T (ξ o s σ o s α) I]"
by blast

hence "t ∩ A α a0' ∈ timpl_close_set (α vals A I) (α_i [A T (ξ o s σ o s α) I])"
when t: "t ∈ vals A" for t
using vals_vals_sym[OF t] by blast

hence B: "α vals (A@[th]) I ⊆
timpl_close_set (α vals A I) (α_i [A T (ξ o s σ o s α) I])
∪
(\xi o s σ o s α) \set t s transaction T \set I α set a0'"
using 0(2) 3
by (simp add: abs_apply_terms_def image_subset_iff)

have 4: "fv (t ∩ A α a0' ∩ a) = \{}" for t a
using _grounds[of "t ∩ A α a0' ∩ a" abs_fv[of "t ∩ A α a0' ∩ a"]]
by argo

have "is Fun (t ∩ A α a0' ∩ a)" for t
using 4[of t a0'] by force

thus ?A
using A step_props(1,3)
unfolding T'_def a0_def a0'_def abs_apply_terms_def
by blast

show ?B
using B step_props(2,4) admissible_transaction_Value_vars[OF bspec[OF P T]]
by (auto simp add: T'_def a0_def a0'_def abs_apply_terms_def)
lemma reachable_constraints_covered:
  assumes A_reach: "A ∈ reachable_constraints P"
  and I: "welltyped_constraint_model I A"
  and FP:
  "analyzed (timpl_closure_set (set FP) (set TI))"
  "wf_term (set FP)"
  "ground (set FP)"
  and OCC:
  "∀ t ∈ timpl_closure_set (set FP) (set TI). ∀ f ∈ funs_term t. is_Abs f → f ∈ Abs ` set OCC"
  and TI:
  "set TI = {(a,b) ∈ (set TI) | a ≠ b}"
  and P:
  "∀ T ∈ set P. admissible_transaction T"
  and transactions_covered: "list_all (transaction_check (FP, OCC, TI)) P" 
  shows "∀ t ∈ α ik A I. timpl_closure_set (set FP) (set TI) ⊢ c t" 
  and "α vals A I ⊆ absc ` set OCC"
using A_reach I
proof (induction rule: reachable_constraints.induct)
case init
  { case 1 show ?case by (simp add: abs_intruder_knowledge_def) }
  { case 2 show ?case by (simp add: abs_value_constants_def) }
next
  case (step A T ξ σ α)
  hence "welltyped_constraint_model I A" 
    by (metis welltyped_constraint_model_prefix)
  hence IH: "∀ t ∈ α ik A I. timpl_closure_set (set FP) (set TI) ⊢ c t" 
    "α vals A I ⊆ absc ` set OCC"
  using step.IH by metis
  show ?case
    using reachable_constraints_covered_step[
    OF step.hyps(1,2) "1.prems" step.hyps(3-5) FP(1,2) IH(1)
    FP(3) OCC IH(2) TI P transactions_covered]
    by metis
  }
  { case 2
    hence "welltyped_constraint_model I A" 
      by (metis welltyped_constraint_model_prefix)
    hence IH: "∀ t ∈ α ik A I. timpl_closure_set (set FP) (set TI) ⊢ c t" 
      "α vals A I ⊆ absc ` set OCC"
    using step.IH by metis
    show ?case
      using reachable_constraints_covered_step[
      OF step.hyps(1,2) "2.prems" step.hyps(3-5) FP(1,2) IH(1)
      FP(3) OCC IH(2) TI P transactions_covered]
      by metis
  }
qed

lemma attack_in_fixpoint_if_attack_in_ik:
  fixes FP::"('fun,'atom,'sets,'lbl) prot_terms"
  assumes "∀ t ∈ IK · a ∈ α set a. FP ⊢ c t"
  and "attack(n) ∈ IK"
  shows "attack(n) ∈ FP"
proof -
  have "attack(n) · a ∈ IK · a ∈ α set a" by (rule abs_in[OF assms(2)])
  hence "FP ⊢ c attack(n) · a" using assms(1) by blast
  moreover have "attack(n) · a = attack(n)" by simp
  ultimately have "FP ⊢ c attack(n)" by metis
  thus ?thesis using ideduct_synch_priv_const_in_ik[of FP "Attack n"] by simp
3.6 Stateful Protocol Verification

lemma attack_in_fixpoint_if_attack_in_timpl_closure_set:
  fixes \( FP::('fun,'atom,'sets,'lbl) prot_terms \)
  assumes \( "attack(n) \in \text{timpl\_closure\_set}\ \text{FP} \\text{TI}" \)
  shows \( "attack(n) \in \text{FP}" \)
  proof
    have \( \"\forall f \in \text{funs\_term} \ (attack(n)). \neg\ \text{is\_Abs} f" \) by auto
    thus \( \text{thesis} \) using timpl_closure_set_no_Abs_in_set[OF assms] by blast
  qed

theorem prot_secure_if_fixpoint_covered:
  fixes \( P \)
  assumes \( \text{FP}: \"\text{analyzed} \ (\text{timpl\_closure\_set} \ (\text{set} \ \text{FP}) \ (\text{set} \ \text{TI}))" \)
  and \( \text{OCC}: \"\forall t \in \text{timpl\_closure\_set} \ (\text{set} \ \text{FP}) \ (\text{set} \ \text{TI}). \forall f \in \text{funs\_term} \ t. \ \text{is\_Abs} f \longrightarrow f \in \text{Abs} \ \text{set} \ \text{OCC}" \)
  and \( \text{TI}: \"\text{set} \ \text{TI} = \{(a,b) \in (\text{set} \ \text{TI})^+. a \neq b\}" \)
  and \( \text{P}: \"\forall T \in \text{set} \ \text{P}. \ \text{admissible\_transaction} \ T" \)
  and \( \text{transactions\_covered}: \"\text{list\_all} (\text{transaction\_check} \ (\text{FP}, \ \text{OCC}, \ \text{TI})) \ \text{P}\" \)
  shows \( \"\exists I. \ \text{welltyped\_constraint\_model} \ I \ (A@[1, \ \text{send}(\{\text{attack(n)}\})]) \) (is \( \"\exists I. \ ?P I\")
  proof
    assume \( \exists I. \ ?P I" \)
    then obtain \( I \) where \( I:\ \text{welltyped\_constraint\_model} \ I \ (A@[1, \ \text{send}(\{\text{attack(n)}\})])" \) by moura
    have \( 0: \"\text{attack(n)} \notin \text{ik}_\text{lst} A \ \text{set} I" \)
      using welltyped_constraint_model_prefix[OF I]
      reachable_constraints_covered(1)[OF A FP OCC TI P transactions_covered]
      attack_in_fixpoint_if_attack_in_ik[of "\text{ik}_\text{lst} A \ \text{set} I" "A@[1, \ \text{send}(\{\text{attack(n)}\})]" n]
      attack_in_timpl_closure_set
      attack_notin_FP
      unfolding abs_intruder_knowledge_def by blast
    have \( 1: \"\text{ik}_\text{lst} A \ \text{set} I \vdash \text{attack(n)}" \)
      using I strand_sem_append_stateful[of \\{} \\{} \\"unlabel\ A" _ I]
      unfolding welltyped_constraint_model_def constraint_model_def by force
    show False
      using 0 private_const_deduct[OF _ I]
      reachable_constraints_receive_attack_if_attack'(1)[OF A P welltyped_constraint_model_prefix[OF I] 1]
      by simp
  qed

end

3.6.4 Theorem: A Protocol is Secure if it is Covered by a Fixed-Point

context stateful_protocol_model
begin

theorem prot_secure_if_fixpoint_covered:
  fixes \( P \)
  assumes \( \text{FP}: \"\text{analyzed} \ (\text{timpl\_closure\_set} \ (\text{set} \ \text{FP}) \ (\text{set} \ \text{TI}))" \)

and OCC:

\[
\forall t \in \text{timpl\_closure\_set (set FP) (set TI)}. \forall f \in \text{funs\_term t. is\_Abs f} \rightarrow f \in \text{Abs \ set OCC}
\]

and TI:

\[
\text{set TI} = \{(a,b) \in (\text{set TI})^+. a \neq b\}
\]

and M:

\[
\text{has\_all\_wt\_instances\_of } \Gamma (\bigcup T \in \text{set P. trms\_transaction T}) N
\]

\[
\text{finite N}
\]

\[
\text{tfr}\_\text{set N}
\]

\[
\text{wf trms N}
\]

and P:

\[
\forall T \in \text{set P. admissible\_transaction T}
\]

\[
\forall T \in \text{set P. list\_all\_tfr\_stt (unlabel (transaction\_strand T))}
\]

and transactions_covered:

\[
\text{list\_all (transaction\_check (FP, OCC, TI)) P}
\]

and attack_notin_FP:

\[
\text{attack}\langle n \rangle/ \notin \text{set FP}
\]

and A:

\[
A \in \text{reachable\_constraints P}
\]

shows

\[
\exists I. \text{constraint\_model } I (A@[\{1, send([attack(n)])\}])
\]

(is "\exists I. \text{constraint\_model } I \ A")

proof

assume "\exists I. \text{constraint\_model } I \ A" then obtain I where "constraint\_model I \ A" by moura then obtain I, where I: "\text{well\_typed\_constraint\_model } I, \ A" using reachable\_constraints\_typing\_result[OF M P A] by blast

show False using prot\_secure\_if\_fixpoint\_covered\_typed[OF A I] by force

qed

end

3.6.5 Alternative Protocol-Coverage Check

context stateful\_protocol\_model begin

context begin

private lemma transaction\_check\_variant\_soundness\_aux0:

assumes S: "S \equiv \text{unlabel (transaction\_strand T)}"

and xs: "xs \equiv \text{filter } (\lambda x. x \notin \text{set (transaction\_fresh T)} \land \text{fst } x = \text{TAtom Value}) (\text{fv\_list\_sst } S)"

and x: "\text{fst } x = \text{Var Value} " \text{x }\notin \text{set (transaction\_fresh T)}"

shows "x \in \text{set xs}"

using unfold \text{fv\_list\_sst\_is\_fv\_sst [of "unlabel (transaction\_strand T)"]}

unfolding xs S by auto

private lemma transaction\_check\_variant\_soundness\_aux1:

fixes T FP S C xs OCC negs poss as

assumes S: "S \equiv \text{unlabel (transaction\_checks T)}"

and xs: "xs \equiv \text{filter } (\lambda x. x \notin \text{set (transaction\_fresh T)} \land \text{fst } x = \text{TAtom Value}) (\text{fv\_list\_sst } S)"

and poss: "\text{poss } \equiv \text{transaction\_poschecks\_comp C}"

and negs: "\text{necs } \equiv \text{transaction\_negchecks\_comp C}"

and as: "as \equiv \text{map } (\lambda x. (x, \text{set (filter } (\lambda \text{ab. poss } x \subseteq \text{ab }\land \text{necs } x \cap \text{ab } = \{\}) \text{OCC})) x) xs"

and f: "f \equiv \lambda \text{x. case List\_find } (\lambda p. \text{fst } p = x) \text{ as of Some } p \Rightarrow \text{snd } p | \text{None } \Rightarrow \{\}"

and x: "x \in \text{set xs}"

shows "f x = \text{set (filter } (\lambda \text{ab. poss } x \subseteq \text{ab }\land \text{necs } x \cap \text{ab } = \{\}) \text{OCC})"

proof
3.6 Stateful Protocol Verification

define \( g \) where \( g \equiv \lambda x. \text{set} (\text{filter} (\lambda ab. \text{poss} x \subseteq ab \land \text{negs} x \cap ab = \{\}) \text{OCC}) \)

define \( gs \) where \( gs \equiv \text{map} (\lambda x. (x, g x)) \text{xs} \)

have 1: \( (x, g x) \in \text{set} gs \) using \( x \) unfolding \( gs \_ \text{def} \) by simp

have 2: "distinct xs" unfolding \( xs \_ \text{fv_list} \) _def by auto

have "\exists i < length xs. xs ! i = x \land (\forall j < i. xs ! j \neq x)"

proof (rule ex1E[OF distinct_Ex1[OF 2 x]])

fix i
assume i: "i < length xs \land xs ! i = x"
and "\( \forall j. j < length xs \land xs ! j = x \rightarrow j = i \)" by blast

hence "\( \forall j < i. xs ! j \neq x \)" using i by auto

hence "\( \exists i < length gs. gs ! i = (x, g x) \land (\forall j < i. gs ! j \neq (x, g x)) \)"

using 1 unfolding \( gs \_ \text{def} \) by fastforce

qed

private lemma transaction_check_variant_soundness_aux2:
fixes T FP S C xs OCC negs poss as
assumes C: "C \equiv \text{unlabel} \ (\text{transaction_checks} T)"
and S: "S \equiv \text{unlabel} \ (\text{transaction_strand} T)"
and xs: "xs \equiv \text{filter} (\lambda x. x \notin \text{set} (\text{transaction_fresh} T) \land \text{fst} x = T\text{Atom Value}) \ (\text{fv_list} sst S)"
and poss: "poss \equiv \text{transaction_poschecks_comp} C"
and negs: "negs \equiv \text{transaction_negchecks_comp} C"
and as: "as \equiv \text{map} (\lambda x. (x, g x)) \xs"
and f: "f \equiv \lambda x. \text{List.find} (\lambda p. \text{fst} p = x) as \Rightarrow \text{snd} p | \text{None}"
and x: "x \notin \text{set} xs"
shows "f x = \{}"

proof -

define \( g \) where \( g \equiv \lambda x. \text{set} (\text{filter} (\lambda ab. \text{poss} x \subseteq ab \land \text{negs} x \cap ab = \{\}) \text{OCC}) \)

define \( gs \) where \( gs \equiv \text{map} (\lambda x. (x, g x)) \text{xs} \)

have "\( x, y \notin \text{set} gs \)" for y using \( x \) unfolding \( gs \_ \text{def} \) by force

thus \( \text{thesis} \)

using find_None_iff[of "\( \lambda p. \text{fst} p = x \) gs" (x, g x)] by blast

unfolding \( f \) as \( gs \_ \text{def} \) \( g \_ \text{def} \) by fastforce

qed

private lemma synth_abs_substs_constrs_rel_if_synth_abs_substs_constrs:
fixes T OCC negs poss as
defines \( \vartheta \equiv \lambda \delta x. \text{if} \ \text{fst} x = T\text{Atom Value} \ \text{then} \ (\text{absc} \circ \delta) x \ \text{else} \ \text{Var} x \)
and \( ts \equiv \text{trms_list} sst (\text{unlabel} \ (\text{transaction_receive} T)) \)
assumes \( \exists t \_ \text{wf} \): "\( \forall t \in \text{set ts} \. \text{wf} t \)"
and FP_ground: "\( \text{ground} \ (\text{set} FP) \)"
and \( \exists \! f \_ \text{wf} \): "\( \text{wf} \_ \text{trms} (\text{set} FP) \)"
shows "\( \text{synth_abs_substs_constrs_rel} \ FP \ OCC \ TI \ ts \ (\text{synth_abs_substs_constrs} \ (FP, OCC, TI)) \)"

proof -

let \( ?R = \"\text{synth_abs_substs_constrs_rel} \ FP \ OCC \ TI \)"

let \( ?D = \"\text{synth_abs_substs_constrs_aux} \ FP \ OCC \ TI \)"

have *: "\( \?R \ [t] \ (\?D \ t) \)" when \( \text{wf} \_ \text{trms} \ t \)" using t that

proof (induction t)

case (Var x)

thus \( \text{case} \)

using synth_abs_substs_constrs_rel.SolveValueVar[of "\( \?D \ (\text{Var} x) \) OCC x TI FP"

239
by fastforce
next
case (Fun f ss)
let ?xs = "fv (Fun f ss)"
let ?lst = "map (match_abss OCC TI (Fun f ss)) FP"
define flt where
"flt = (\delta::(('fun,'atom,'sets,'lbl) prot_var \to 'sets set set) option. \delta \neq None)"
define \( \Theta \) where "\( \Theta = (\lambda::('fun,'atom,'sets,'lbl) prot_var \to 'sets set set. \lambda x. if \delta x \neq \emptyset then \delta x else set OCC)""
define \( \forall \) where "\( \forall 1 = fun_point_Union (\Theta \cdot set \Delta) \)"
define \( \forall \) where "\( \forall 2 = fun_point_Inter (?D \cdot set ss) \)"

have f: "arity f = length ss" using \( \text{wf_trm_arity}[\text{OF Fun.prems}] \) by simp

have IH: "\( ?R [s] (?D s) \)" when s: "s \in set ss" for s
using Fun.IH[\( \text{OF s \text{ wf_trm_subterm}[\text{OF Fun.prems Fun_param_is_subterm}[\text{OF s}]] } \) s]
by force

have \( \Delta 3: \forall \delta. \delta \in set \Delta \longleftrightarrow (\exists s \in set FP. match_abss OCC TI (Fun f ss) s = Some \delta)" \)
(is "\forall \delta. \delta \in set \Delta \longleftrightarrow ?P \delta")
proof (intro allI iffI)
fix \delta assume "\( \delta \in set \Delta \)"
then obtain u where "u \in set FP" "match_abss OCC TI (Fun f ss) u = Some \delta"
unfolding \( \Delta \_def flt\_def \) by fastforce
thus "?P \delta" by blast
next
fix \delta assume "?P \delta"
then obtain u where u: "u \in set FP" "match_abss OCC TI (Fun f ss) u = Some \delta" by blast

have "Some \delta \in set ?lst" using u unfolding flt_def by force
hence "Some \delta \in set (filter flt ?lst)" unfolding flt_def by force
moreover have "\( \exists \delta. d = Some \delta \)" when d: "d \in set (filter flt ?lst)" for d
using d unfolding flt_def by simp
ultimately have "\delta \in set (map the (filter flt ?lst))" by force
thus "\delta \in set \Delta" unfolding \( \Delta \_def \) by blast
qed

show ?case
proof (cases "ss = []")
case True
note ss = this
show ?thesis
proof (cases "\neg public f \land Fun f ss \notin set FP")
case True thus ?thesis
using synth_abss_substs_constrs_rel.SolvePrivConstNotin[of f FP OCC TI]
unfolding ss by force
next
case False thus ?thesis
using f synth_abss_substs_constrs_rel.SolvePubConst[of f FP OCC TI]
synth_abss_constrs_rel.SolvePrivConstIn[of f FP OCC TI]
unfolding ss by auto
qed
next
case False
note ss = this
hence f': "arity f > 0" using f by auto
have IH': "\( ?R ss (fun_point_Inter (?D \cdot set ss)) \)"
using IH synth_abss_substs_constrs_rel.SolveCons[of ss, of FP OCC TI ?D] by blast

have "?D (Fun f ss) = (fun_point_union (fun_point_Union_list (map \( \Theta \Delta \) \cdot fun_point_Inter_list (map ?D ss)))"

240
using ss synth_abs_substs_constrs_aux.simps(2)[of FP OCC TI f ss]
unfolding Let_def Δ_def ϑ_def Θ_def by argo
hence "?D (Fun f ss) = fun_point_union ϑ1 ϑ2"
using fun_point_Inter_set_eq[of "map ϑ ss"] fun_point_Union_set_eq[of "map Θ Δ"]
unfolding ϑ1_def ϑ2_def by simp
thus ?thesis
using synth_abs_substs_constrs_rel.SolveComposed[of f' f'[symmetric] Δ3 Θ_def ϑ1_def IH']
unfolding ϑ2_def by argo
qed

private function (sequential) match_abss'_timpls_transform::"(('c set × 'c set) list ⇒ ('a,'b,'c,'d) prot_subst ⇒ ('a,'b,'c,'d) prot_term ⇒ ('a,'b,'c,'d) prot_var ⇒ ('c set set) option"
where
"match_abss'_timpls_transform TI δ (Var x) (Fun (Abs a) _) = (if ∃b ts. δ x = Fun (Abs b) ts ∧ (a = b ∨ (a,b) ∈ set TI) then Some (((λ_. {}).{a}))(x := {a})) else None)"
| "match_abss'_timpls_transform TI δ (Fun f ts) (Fun g ss) = (if f = g ∧ length ts = length ss then map_option fun_point_Union_list (those (map2 (match_abss'_timpls_transform TI δ) ts ss)) else None)"
| "match_abss'_timpls_transform _ _ _ _ = None"
by pat_completeness auto
termination
proof -
let ?m = "measures [size o fst o snd o snd]"

have 0: "wf ?m" by simp

show ?thesis
apply (standard, use 0 in fast)
by (metis (no_types) comp_def fst_conv snd_conv measures_less Fun_zip_size_lt(1))
qued

private lemma match_abss'_timpls_transform_Var_inv:
assumes "match_abss'_timpls_transform TI δ (Var x) (Fun (Abs a) _ ) = Some σ"
shows "∃b ts. δ x = Fun (Abs b) ts ∧ (a = b ∨ (a,b) ∈ set TI)"
and "σ = (((λ_. {}).{a}))(x := {a}))"
using assms match_abss'_timpls_transform.simps(1)[of TI δ x a ts]
by (metis option.option.inject)

private lemma match_abss'_timpls_transform_Fun_inv:
assumes "match_abss'_timpls_transform TI δ (Fun f ts) (Fun g ss) = Some σ"

qed
shows "f = g" (is ?A)
and "length ts = length ss" (is ?B)
and "∃v. Some v = those (map2 (match_abss’timpls_transform TI δ) ts ss) ∧ σ = fun_point_Union_list v" (is ?C)
and "(t,a) ∈ set (zip ts ss). ∃σ’. match_abss’timpls_transform TI δ t a = Some σ’" (is ?D)

proof -
  note 0 = assms match_abss’timpls_transform.simps(2)[of TI δ f ts g ss] option.distinct(1)
  show ?A by (metis 0)
  show ?B by (metis 0)
  show ?C by (metis (no_types, opaque_lifting) 0 map_option_eq_Some)
  thus ?D using map2_those_Some_case[of "match_abss’timpls_transform TI δ" ts ss] by fastforce
qed

private lemma match_abss’timpls_transform_nonempty_is_fv:
  assumes "match_abss’timpls_transform TI δ s t = Some σ"
  and "σ x ≠ {}"
  shows "x ∈ fv s"
using assms
proof (induction s t arbitrary: TI δ σ rule: match_abss’timpls_transform.induct)
  case (1 TI δ y a ts)
  show ?case using match_abss’timpls_transform_Var_inv[OF "1.prems"(1)] "1.prems"(2) by fastforce
  next
  case (2 TI δ f ts g ss)
  note prems = "2.prems"
  note IH = "2.IH"
  obtain ϑ where ϑ: "Some ϑ = those (map2 (match_abss’timpls_transform TI δ) ts ss)"
  and fg: "f = g" "length ts = length ss"
  using match_abss’timpls_transform_Fun_inv[OF prems(1)] by fast
  have "∃σ ∈ set ϑ. σ x ≠ {}"
  using fg(2) prems ϑ unfolding fun_point_Union_list_def by auto
  then obtain t’ s’ σ where ts': "(t’,s’) ∈ set (zip ts ss)" "match_abss’timpls_transform TI δ t’ s’ = Some σ" "σ x ≠ {}"
  using those_map2_SomeD[OF ϑ(1)[symmetric]] by blast
  show ?case using ts'(3) IH[OF conjI[OF fg] ts'(1) _ ts'(2)] set_zip_leftD[OF ts'(1)] by force
qed auto

private lemma match_abss’timpls_transformI:
  fixes s t::"('a,'b,'c,'d) prot_term"
  and δ::"('a,'b,'c,'d) prot_subst"
  and σ::"('a,'b,'c,'d) prot_var ⇒ 'c set set"
  assumes TI: "set TI = {(a,b) ∈ (set TI)+. a ≠ b}" and δ: "timpls_transformable_to TI t (s · δ)" and σ: "match_abss’ s t = Some σ"
  and t: "fv t = {}" and s: "∀f ∈ funs_term s. ¬is_Abs f"
  "∀x ∈ fv s. ∃a. δ x = (a)abs"
  shows "match_abss’timpls_transform TI δ s t = Some σ"
using δ σ t s
proof (induction t arbitrary: s δ σ)
  case (Fun f ts)
  note prems = Fun.prems
  note IH = Fun.IH
  show ?case
  proof (cases s)
    case (Var x)
    obtain a b where a: "f = Abs a" "σ = (λ_. {})⟨x := {a}⟩" and b: "δ x = ⟨b⟩abs"
using match_abss'_Var_inv[OF prems(2)[unfolded Var]] prems(5)[unfolded Var]
by auto
thus \?thesis
using prems(1) timpls_transformable_to_inv[OF TI f ts "Abs b" "[]"]
unfolding Var by auto
next
case (Fun g ss)
note 0 = timpls_transformable_to_inv[OF prems(1)[unfolded Fun subst_apply_term.simps(2)]]
note 1 = match_abss'_Fun_inv[OF prems(2)[unfolded Fun]]

obtain \varphi where "\varphi = \{\text{some } \sigma \mid \text{fun_point_Union_list } \varphi = \text{map2 match_abss' ss ts} \}"
using 1(3) by force
have "timpls_transformable_to_TI t' (s' \cdot \delta)" "\exists \sigma. \text{match_abss' s' t'} = Some \sigma'"
when "(t',s') \in \{(\text{set } s)\} fn s" by (metis 0(2) nth_map[of _ ss] zip_arg_index[OF that],
use that I(4) in_set_zip_swap[of t' s' ts ss] in fast)

hence IH': "match_abss'_timpls_transform TI \delta s' t' = Some \sigma'" when "(t',s') \in \{(\text{set } s)\} fn s" by (metis (no_types, lifting) set_zip_leftD set_zip_rightD subsetI subset_empty
term.set_intros(2) term.set_intros(4))

have "\text{map2 } (\text{match_abss' timpls_transform } \text{TI } \delta) \text{ ss ts} = \text{some } \sigma"
using IH' \varphi (1) 1(4) in_set_zip_swap[of _ _ ss ts]
those_Some_iff[of \varphi "\text{map2 } (\text{match_abss' timpls_transform } \text{TI } \delta) \text{ ss ts}"

by auto
thus \?thesis using \varphi (2) 1(1,2) Fun by simp
qed simp

lemma timpls_transformable_to_match_abss'_nonempty_disj':
fixes s t::"('a,'b,'c,'d) prot_term"
and \delta::"('a,'b,'c,'d) prot_subst"
and \sigma::"('a,'b,'c,'d) prot_var \Rightarrow 'c set set"
assumes TI: "\text{set } \text{TI} = \{(a,b) \in \text{set } \text{TI} \cdot a \neq b}"
and \delta: "timpls_transformable_to_TI t (s \cdot \delta)"
and \sigma: "match_abss' s t = Some \sigma"

shows "\forall b \in \sigma x. (b,a) \in \text{set } \text{TI}" (is "?P \sigma x")
proof -

note 0 = match_abss'_subst_disj_nonempty[OF TI]

have 1: "s \cdot \delta \in \text{timpl Closure t (set } \text{TI})"
using timpls_transformable_to_iff_in_timpl_closure[OF TI \delta] by blast

have 2: "match_abss'_timpls_transform TI \delta s t = Some \sigma"
using match_abss'_timpls_transformI[OF TI \delta s t] by simp

show "?P \sigma x" using 2 TI x t s a

proof (induction TI \delta s t arbitrary: \sigma rule: match_abss'_timpls_transform.induct)
case (1 TI \delta y c ts) thus \?case

using match_abss'_timpls_transform_Var_inv[OF "1.prems"(1)] by auto
next
case (2 TI \delta f ts g ss)

obtain \varphi where \varphi: "\forall f g. \text{length ts = length ss}"
and \varphi: "Some \varphi = \{\text{some } \sigma \mid \text{match_abss' timpls_transform } \text{TI } \delta) \text{ ts ss}"
by auto

"\sigma = \text{fun_point_Union_list } \varphi"
∀(t, s)∈set (zip ts ss). ∃σ'. match_abss'_timpls_transform TI δ t s = Some σ'' using match_abss'_timpls_transform_Fun_inv[OF "2.prems"(1)] by blast

have "(b,a)∈(set TI)" when δ': "δ'∈set "b ∈ δ' x" for δ' b
proof -
obtain t' s' where t': "(t',s')∈set (zip ts ss)" "match_abss'_timpls_transform TI δ t' s' = Some δ''" using those_map2_SomeD[OF "2.prems"(1)[symmetric] δ'(1)] by blast

have *: "f x ∈ {}" "f∈funs_term t'. ¬is_Abs f" "f x ∈ fv t'. ∀a. δ x = ⟨a⟩abs" using "2.prems"(4-6) set_zip_leftD[OF t'(1)] set_zip_rightD[OF t'(1)] by (fastforce, fastforce, fastforce)

have **: "x ∈ fv t'" using δ'(2) match_abss'_timpls_transform_nonempty_is_fv[OF t'(2)] by blast

show ?thesis using δ'(2) fun_point_Union_list_def by simp

qed

lemma timpls_transformable_to_match_abss'_nonempty_disj:
fixes s t::"('a,'b,'c,'d) prot_term" and δ σ::"('a,'b,'c,'d) prot_subst" and σ::"('a,'b,'c,'d) prot_var ⇒ 'c set set"
assumes TI: "set TI = {(a,b)∈(set TI). a ≠ b}" and δ: "timpls_transformable_to TI t (s · δ)" and σ: "match_abss' s t = Some σ" and x: "x ∈ fv s" and t: "fv t = {}" and s: "∀f∈funs_term s. ¬is_Abs f" "∀x∈fv s. ∃a. δ x = ⟨a⟩abs"
shows "⋂(ticl_abs TI ` σ x) ≠ {}"
proof -
have 0: "(a,b)∈(set TI)" when y: "y∈fv s" "a∈σ y" "δ y = ⟨b⟩abs," for a b y using timpls_transformable_to_match_abss'_nonempty_disj[OF TI δ y(1) t s y(3)] y(2) by blast

obtain b where b: "δ x = ⟨b⟩abs," using x s(2) by blast

have "b∈ticl_abs TI a" when a: "a∈σ x" for a using 0[OF x a b] unfolding ticl_abs_iff[OF TI] by blast
thus ?thesis by blast

qed

lemma timpls_transformable_to_subst_subterm:
fixes s t::"(('a,'b,'c,'d) prot_term, 'v) term" and δ σ::"(('a,'b,'c,'d) prot_term, 'v) subst" and σ::"('a,'b,'c,'d) prot_term, 'v' subst" and s::"('a,'b,'c,'d) prot_term, 'v' subst"
assumes "timpls_transformable_to TI t (s · δ)" and σ: "match_abss' s t = Some σ"
and s: "fv t = {}" and "∀f∈funs_term s. ¬is_Abs f" "∀x∈fv s. ∃a. δ x = ⟨a⟩abs"
shows "⋂(ticl_abs TI ` σ x) ≠ {}"
using assms
proof (induction "t · δ" "t · σ" arbitrary: t rule: timpls_transformable_to.induct)
case (1 TI x y) thus ?case using (cases t)
next
case (2 TI f T g S)

note prems = "2.prems"

note hyps = "2.hyps"(2-)

note IH = "2.hyps"(1)

show ?case
proof (cases "s = t")

244
case False
then obtain \( h \ U u \) where \( t = \text{Fun } h \ U \) \( u \in \text{set } U \) \( s \subseteq u \)
using prems(2) by (cases \( t \)) auto
then obtain \( i \) where \( i < \text{length } U \) \( U ! i = u \)
by (metis in_set_conv_nth)

have \( \text{"timpls_transformable_to_TI } (u \cdot \delta) (u \cdot \sigma)" \)
using \( t \ i \ \text{prems(1)} \ \text{timpls_transformable_to_inv(2)} \) [of TI \( h \) \( U \cdot list \delta \) \( h \) \( U \cdot list \sigma \) \( i \)] by simp
thus \( \text{?thesis} \) using IH hyps \( t \)
by auto
qed (use prems in auto)
qed simp_all

lemma timpls_transformable_to_subst_match_case:
assumes \( \text{"timpls_transformable_to_TI } s (t \cdot \vartheta)" \)
and \( \text{fv } s = \{ \} \)
and \( \forall \ f \in \text{funs_term } t. \ \neg \text{is_Abs } f \)
and \( \text{distinct } (\text{fv_list } t) \)
and \( \forall \ x \in \text{fv } t. \ \exists \ a. \vartheta \ x = \langle a \rangle_{\text{abs}} \)
shows \( \exists \ \delta. \ s = t \cdot \delta \)
using assms
proof (induction \( s = t \cdot \vartheta \) arbitrary: \( t \) rule: timpls_transformable_to.induct)
case (2 \( TI f T g S \))
note prems = "2.prems"
note hyps = "2.hyps"(2-)
note IH = "2.hyps"(1)
show \( \text{?case} \)
proof (cases \( t \))
  case (Var \( x \))
  then obtain \( a \) where \( t \cdot \vartheta = \langle a \rangle_{\text{abs}} \)
  using prems(5)
  by fastforce
  show \( \text{?thesis} \)
  using hyps timpls_transformable_to_inv'[OF prems(1)[unfolded \( a \)]]
  unfolding Var by force
  next
  case (Fun \( h \ U \))
  have \( g = h \) and \( S = U \cdot list \vartheta \)
  using hyps unfolding Fun by simp_all
  note 0 = distinct_fv_list_Fun_param[OF prems(4)[unfolded Fun]]
  have 1: \( \forall \ f \in \text{funs_term } u. \ \neg \text{is_Abs } f \) when \( u \in \text{set } U \)
  using prems(3) \( u \) unfolding Fun by fastforce
  have 2: \( \text{fv } t' = \{ \} \) when \( t' \in \text{set } T \) for \( t' \)
  using \( t' \) prems(2) by simp
  have 3: \( \forall \ x \in \text{fv } u. \ \exists \ a. \vartheta \ x = \langle a \rangle_{\text{abs}} \) when \( u \in \text{set } U \)
  using \( u \) prems(5) unfolding Fun by simp
  have \( \neg \text{is_Abs } f \)
  using prems(3) timpls_transformable_to_inv'[OF prems(1)[unfolded hyps[symmetric] S g]]
  unfolding Fun by auto
  hence \( f = h \) and \( T. \ \text{length } T = \text{length } U \)
  using prems(1) timpls_transformable_to_inv(1,3)[of TI \( f T \) \( h \) \( U \cdot list \vartheta \) \( i \)]
  unfolding Fun by fastforce

  define \( \Delta \) where \( \Delta \equiv \lambda i. \text{if } i < \text{length } T \text{ then SOME } \delta. \ T ! i = U ! i \cdot \delta \text{ else undefined} \)
  have \( \text{"timpls_transformable_to_TI } (T ! i) (U ! i \cdot \vartheta)" \) when \( i \in \text{set } T \) for \( i \)
  using prems(1)[unfolded Fun] \( T \ i \ \text{timpls_transformable_to_inv(2)} \) [of TI \( f T \) \( h \) \( U \cdot list \vartheta \) \( i \)]
  by auto
  hence \( \exists \ \delta. \ T ! i = U ! i \cdot \delta \) when \( i \in \text{set } T \) for \( i \)
  using \( i \) IH[of _ _ _ 2 1 0 3, of "T ! i" "U ! i"]

qed
3 Stateful Protocol Verification

unfolding Fun g S by simp
hence $\Delta$: $T \cdot i = U \cdot i \cdot \Delta$ when $i$: "$i < \text{length T}$" for $i$
using $i$ someI2[of "$\lambda \delta \cdot T \cdot i = U \cdot i \cdot \delta" \_ $\lambda \delta \cdot T \cdot i = U \cdot i \cdot \delta"$]
unfolding $\Delta_{\text{def}}$ by fastforce

define $\delta$ where "$\delta \equiv \lambda x. \text{if } \exists i < \text{length T}. x \in \text{fv (U \cdot i)}$ then $\Delta \cdot (\text{SOME } i. i < \text{length T} \land x \in \text{fv (U \cdot i)}) \cdot x$
else undefined"

have "$T \cdot i = U \cdot i \cdot \delta$" when $i$: "$i < \text{length T}$" for $i$
proof -
  have "$j = i$"
    when $x$: "$x \in \text{fv (U \cdot i)}$" and $j$: "$j < \text{length T} \land x \in \text{fv (U \cdot i)}$" for $j$
    using $x$ i $j$ $T$ distinct_fv_list_idx_fv_disjoint[OF prems(4)[unfolded Fun], of $h$ $U$]
    by (metis (no_types, lifting) disjoint_iff_not_equal neqE term.dual_order.refl)
  hence "$\delta \cdot x = \Delta \cdot i \cdot x$" when $x$: "$x \in \text{fv (U \cdot i)}$" for $x$
  unfolding $\delta_{\text{def}}$ by (metis (no_types, lifting))
thus $?\text{thesis}$ by (metis $\Delta \cdot i \cdot \text{term_subst_eq}$)
qed

hence "$T = U \cdot \text{list } \delta$" by (metis (no_types, lifting) $T$ length_map nth_equalityI nth_map)
hence "$\text{Fun } f \cdot T = \text{Fun } f \cdot U \cdot \delta"$ by simp
thus $?\text{thesis}$ using $\text{Fun } f$ by fast
qed simp_all

lemma timpls_transformable_to_match_abs's_case:
assumes "$\text{timpls_transformable_to } TI \cdot s \cdot (t \cdot \vartheta)" and "$\text{fv } s = \{\}$" and "$\forall f \in \text{funs_term } u. \neg \text{is_Abs } f$" and "$\forall x \in \text{fv } t. \exists a. \vartheta \cdot x = \langle a \rangle_{\text{abs}}$"
shows "$\exists \delta. \text{match_abs'} } t \cdot s = \text{Some } \delta$"
using assms
proof (induction $s$: "t \cdot \vartheta" arbitrary: $t$ rule: timpls_transformable_to.induct)
case (2 TI $f \cdot T$ $g$ $S$)
note prems = "2.prems"
note hyps = "$2.\text{hyps}"(2-)
note IH = "$2.\text{hyps}"(1)

show $?\text{case}$
proof (cases $t$)
  case (Var $x$)
  then obtain $a$ where: "$t \cdot \vartheta = \langle a \rangle_{\text{abs}}$" using prems(4) by fastforce
  thus $?\text{thesis}$ using timpls_transformable_to_inv'(4)[OF prems(1)[unfolded hyps[ symmetric] $S$ $g$]] unfolding Fun

next
case (Fun $h$ $U$)

have $1$: "$\forall f \in \text{funs_term } u. \neg \text{is_Abs } f$" when $u$: "$u \in \text{set } U$" for $u$
using prems(3) $u$ unfolding Fun by fastforce

have $2$: "$\text{fv } t' = \{\}$" when $t'$: "$t' \in \text{set } T$" for $t'$
using $t'$ prems(2) by simp

have $3$: "$\forall x \in \text{fv } u. \exists a. \vartheta \cdot x = \langle a \rangle_{\text{abs}}$" when $u$: "$u \in \text{set } U$" for $u$
using $u$ prems(4) unfolding Fun by simp

have "$\neg \text{is_Abs } f$"
using prems(3) timpls_transformable_to_inv(3)[OF prems(1)[unfolded hyps[ symmetric] $S$ $g$]] unfolding Fun by auto

246
hence $f = h$ and $T$: "length $T = length U"
using prems(1) timpls_transformable_to_inv(1,3)[of TI T h "U $\cdot_{t acting \emptyset}^\emptyset" ]
unfolding Fun by fastforce+

define $\Delta$ where "$\Delta \equiv \lambda x_i.
if i < length T
then SOME $\delta$. match_abss' (U ! i) (T ! i) = Some $\delta$
else undefined"

have "timpls_transformable_to TI (T ! i) (U ! i) (T ! i) = Some $\delta" when i: "i < length T" for i
using prems(1)[unfolded Fun] T i timpls_transformable_to_inv(2)[of TI T h "U $\cdot_{t acting \emptyset}^\emptyset" i] by auto

hence "$\exists i. match_abss' (U ! i) (T ! i) = Some (\Delta i)" when i: "i < length T" for i
using i T IH[OF _ _ _ 2 1 3, of "T ! i" "U ! i"]
unfolding Fun g S by simp

hence "match abss' (U ! i) (T ! i) = Some (\Delta i)" when i: "i < length T" for i
using i someI2[of "$\lambda i. match_abss' (U ! i) (T ! i) = Some \delta\_"
            "$\lambda i. match_abss' (U ! i) (T ! i) = Some \delta\_"
unfolding $\Delta$ def by fastforce

thus $\text{thesis}$
using match_abss'_FunI[OF _ T] unfolding Fun $f$ by auto
qed
qed simp_all

lemma timpls_transformable_to_match_abss_case:
assumes TI: "set TI = {(a,b) \in (set TI)^+. a \neq b}"
and "timpls_transformable_to TI s (t \cdot \emptyset)"
and "fv s = {}"
and "\forall f \in funs_term. \neg is_Abs f"
and "\forall x \in fv t. \exists a. \emptyset x = (a)_{abs}"

shows "\exists \delta. match_abss OCC TI t s = Some \delta"

proof -
obtain $\delta$ where $\delta$: "match_abss' t s = Some $\delta"
using timpls_transformable_to_match_abss_case[OF assms(2-)] by blast

show $\text{thesis}$
using $\delta$ timpls_transformable_to_match_abss'_nonempty_disj[OF assms(1,2) $\delta$ _ assms(3-5)]

unfolding match_abss_def by simp

qed

private lemma transaction_check_variant_soundness_aux3:
fixes T FP S C xs OCC negs poss as

defines "$\equiv \lambda x. if \text{fst } x = \text{TAtom Value then (absc } \circ \delta) \times \text{else Var } x"
and "$C \equiv \text{unlabel (transaction_checks } T)"
and "$S \equiv \text{unlabel (transaction_strand } T)"
and "$ts \equiv \text{trms_list_sst (unlabel (transaction_receive } T))"
and "$xs \equiv \text{filter (\lambda x. x \notin set (transaction_fresh } T) \wedge \text{fst x} = \text{TAtom Value}) (fv_list_sst S)"

assumes TI0: "\forall (a,b) \in \text{set TI}. \forall (c,d) \in \text{set TI}. b = c \wedge a \neq d \rightarrow (a,d) \in \text{set TI}"

\"\forall (a,b) \in \text{set TI}. a \neq b"

and OCC: "\forall t \in \text{set FP}. \forall a. Abs a \in \text{funs_term } t \rightarrow a \in \text{set OCC}"

and FP: "\text{ground (set FP)}"

and x: "x \in \text{set xs}"

and xs: "\forall x. x \in \text{set xs} \rightarrow \delta x \in \text{set OCC}"
"\forall x. x \in \text{set xs} \rightarrow \text{poss } x \subseteq \delta x"
"\forall x. x \in \text{set xs} \rightarrow \delta x \cap \text{negs } x = {}"
"\forall x. x \notin \text{set xs} \rightarrow \delta x = {}"

and ts: "\forall t \in \text{trms_sst } (\text{transaction_receive } T). \text{intruder_synch_mod_timpls FP } TI (t \cdot \emptyset \delta)"

\"\forall t \in \text{trms_sst } (\text{transaction_receive } T). \forall f \in \text{funs_term. } \neg \text{is_Abs } f""

\"\forall x \in \text{fv_sst } (\text{trms_sst } (\text{transaction_receive } T)). \text{fst x = TAtom Value}"

and C: "\forall x s. \langle a: \text{Var } x \in \text{Fun (Set s) } \emptyset \rangle \in \text{set C} \rightarrow s \in \delta x"

\"\forall x s. \langle \text{Var } x \text{ not in Fun (Set s) } \emptyset \rangle \in \text{set C} \rightarrow s \notin \delta x"

and \sigma: "\text{synth_abss_substa_constr_rel FP OCC TI ts } \sigma"

shows "$\delta x \in \sigma x"
3 Stateful Protocol Verification

proof -
  note defs = assms(1-5)

  note TI = trancl_eqI'[OF TI0]

  have δx0: "δ x ∈ set OCC" "poss x ⊆ δ x" "δ x ∩ negs x = {}" using x xs by (blast,blast,blast)

  have ts0: "∀ t ∈ set ts. intruder_synth_mod_timpls FP TI (t · ϑ δ)
    using ts(1) trms_list_is_trms_unfold unfolding ts_def by blast

  have ts1: "¬ Fun (Abs n) S ⊑ set set ts" for n S
    using ts(2) funs_term_Fun_subterm' unfolding ts_def trms_transaction_unfold trms_list_sst_is_trms[symmetric] is_Abs_def by blast

  have ts2: "∀ x ∈ fv set(set ts). fst x = TAtom Value"
    using ts(3) unfolding ts_def trms_transaction_unfold trms_list_sst_is_trms[symmetric] by blast

  show ?thesis using σ ts0 ts1 ts2
    proof (induction rule: synth_abs_substs_constrs_rel.induct)
      case (SolvePrivConstNotin c)
      hence "intruder_synth_mod_timpls FP TI (Fun c [])" by force
      hence "list_ex (λ.t. timpls_transformable_to TI t (Fun c [])) FP"
        using SolvePrivConstNotin.hyps(1,2)
        unfolding list_ex_iff by blast
      then obtain t where t: "t ∈ set FP" "timpls_transformable_to TI t (Fun c [])"
        unfolding list_ex_iff by blast
      have "¬ is_Abs c"
        using SolvePrivConstNotin.prems(2)[of _ "[]"]
        by (metis in_subterms_Union is_Abs_def list.set_intros(1))
      hence "t = Fun c []"
        using t(2) timpls_transformable_to_inv[OF TI] by (cases t) auto
      thus ?case using t(1) SolvePrivConstNotin.hyps(3) by fast
    next
      case (SolveComposed g us Δ Θ ϑ1 ϑ2)
      show ?case
        proof (cases "∀ t ∈ set us. intruder_synth_mod_timpls FP TI (t · ϑ δ)"
          have "a ∈ set OCC"
            using t(1)[unfolded a(1)] OCC by auto
          thus ?case
            unfolding SolveValueVar.hyps(1) ticl_abss_def ticl_abs_def
            by force
        next
          case (SolveComposed g us Δ Θ ϑ1 ϑ2) show ?case
          proof (cases "∀ t ∈ set us. intruder_synth_mod_timpls FP TI (t · ϑ δ)"
            qed
3.6 Stateful Protocol Verification

Case True

Hence \( \delta \in \varnothing \)

Using \( \text{SolveComposed.IH SolveComposed.prems(2,3)} \)

Distinct fv_list_Fun_param[of g us]

By auto

Thus \(?thesis unfolding fun_point_union_def by simp \)

Next

Case False

Hence \( \text{"list_ex (\lambda t. timpls_transformable_to TI t (Fun g us \cdot \varnothing \delta)) FP"} \)

Using \( \text{SolveComposed.prems(1) intruder_synth_mod_timpls.simps(2)[of FP TI g "us \cdot \varnothing \delta"]} \)

Unfolding list_all_iff by auto

Then obtain \( t \) where \( t \in \text{set FP} \)

\( \text{"timpls_transformable_to TI t (Fun g us \cdot \varnothing \delta)\"} \)

Unfolding list_ex_iff by blast

Have \( t \_\text{ground} \): \( "fv t = \{\}\"

Using \( t(1) \) FP\_ground by simp

Have \( g\_\text{no_abs}: \"\text{\~{\textut{is}} abs f\"} \) when \( f: \"f \in funs_term (Fun g us)\" \) for \( f \)

Proof -

Obtain \( fts \) where \( \"\text{Fun f fts \sqsubseteq Fun g us}\" \) using \( \text{funs_term_Fun_subterm[OF f]} \) by blast

Thus \(?thesis using \text{SolveComposed.prems(2)[of _ fts]} \) by (cases f) auto

Qed

Have \( g\_\varnothing\_\text{abs}: \"\exists a. \\varnothing \text{ \delta y = \langle a\rangle_{abs} Document size: 595.3x841.9 Document dimension: 394x807.09984 \} \)

Using \( y \text{ SolveComposed.prems(3)} \)

Unfolding \( \text{\varnothing \_def} \) by fastforce

Obtain \( \delta' \) where \( \delta' : \"\text{match_abs OCC TI (Fun g us) t = Some \delta\''}\"

Using \( g\_\text{no_abs g\_\varnothing\_abs timpls_transformable_to_match_abs_case[OF TI t(2) t\_ground ]} \)

By blast

Let \( ?h1 = \"\lambda \delta x. if x \in fv (Fun g us) then \delta x else set OCC\"

Let \( ?h2 = \"\lambda \delta x. \cap (ticl\_abs TI \cdot \delta x)\"

Obtain \( \delta'' \) where \( \delta'' : \"\text{match_abs OCC TI (Fun g us) t = Some \delta\''}\"

When \( a: \"a \in \delta'' x\" \) and \( x\_\text{in}_g: \"x \in fv (Fun g us)\" \) for \( a \)

Proof -

Have \( \"\text{fst x = TAtom Value}\" \) using \( x\_\text{in}_g \text{ SolveComposed.prems(3)} \) by auto

Hence \( \"\varnothing \delta x = \langle \delta x\rangle_{abs}\" \) Unfolding \( \varnothing \_def by simp \)

Hence \( \"(a, \delta x) \in (set TI)\"\"

Using \( \text{timpls_transformable_to_match_abs\'_nonempty_disj'[OF TI t(2) \delta''(1) x\_in}_g \_t\_ground]} \)

By fastforce

Thus \( \"\delta x \in ticl\_abs TI a\" unfolding ticl\_abs iff[OF TI] by force \)

Qed

Hence \( \"\delta x \in \delta' x\" when x\_in_g: \"x \in fv (Fun g us)\" \) for \( \delta' \)

Using \( \delta''(2,3) x\_in_g\) unfolding \( \delta'''(1) by simp \)

Hence \( \"\delta x \in \delta' x\" when \( \delta' x \neq \{\}\" using \text{match_abs OCC if not_fv[OF \delta' that]} \)

By blast

Hence \( \"\delta x \in \varnothing1 x\"

Using \( \delta\_\Delta \_\Delta x0(1)\) Unfolding \( \text{SolveComposed.hyps(4,5) fun_point_Union_def by auto \}

Thus \(?thesis unfolding fun_point_union_def by simp \)

Qed

(\text{auto simp add: \delta0 fun_point_Inter_def} \)

Qed
proof -
let \( ?P = "\text{FP}, \text{OCC}, \text{TI}" \)
let \( ?P = \lambda s u. \text{let } \delta = \text{mgu } s u \text{ in } \delta \neq \text{None }\rightarrow (\forall x. x \in f v s. \text{is_Fun } (\text{the } \delta x) \rightarrow \text{is_Abs } (\text{the_Fun } (\text{the } \delta x)))" \)
define \( \varnothing 0 \) where \( \varnothing 0 \equiv \lambda x. \text{if } \text{fst } x = \text{TAtom Value } \text{then } (\text{absc } \delta x) \text{ else } \text{Var } x \)\)
define \( g \) where \( g \equiv \lambda x. \text{set } (\text{filter } (\lambda ab. \text{poss } x \subseteq ab \land \text{negs } x \cap ab = \{\}) \text{ OCC})" \)
define \( gs \) where \( gs \equiv \text{map } (\lambda x. (x, g x)) x s" \)

note \( \text{def} = \text{assms}(3-10) \varnothing 0\text{ def} \)

note \( \text{assm} = \text{assms}(11-)[\text{unfolded def}] \)

have \( \text{ts0: } "\forall t \in \text{ts set } ts \cdot wf \cdot\text{ms t }"\)
using \( \text{admissible_transaction_is_wellformed_transaction(4)[OF T_adm]} \)
unfolding \( \text{admissible_transaction_terms_def} [\text{symmetric}] \)
\( \text{ts_def trms_list,sst, is_trms,sst, [symmetric]} \)
\( \text{trms_transaction_unfold} \)
by \( \text{fast} \)

have \( \text{ts1: } "\forall t \in \text{ts set } ts. \forall f \in \text{fun s term t } \rightarrow \text{is_Abs f}"\)
using \( \text{protocol_transactions_no_abss}[\text{OF T_adm}] \text{ funs_term Fun subterm} \)
\( \text{trms_set,unlabel_prefix_subset(1)} \)
unfolding \( \text{ts_def trms_list,sst, is_trms,sst, [symmetric]} \text{ is_Abs_def transaction_strand_def} \)
by \( \text{metis (no_types, opaque_lifting) in_subtermsUnion in_subterms_subset_Union_subset_iff} \)

have \( \text{ts2: } "\forall x \in \text{fv set } (\text{set } ts). \text{fst } x = \text{TAtom Value }"\)
using \( \text{admissible_transaction_Value_vars}[\text{OF T_adm}] \)
\( \text{wellformed_transaction_send_receive_fv_subset(1)[OF admissible_transaction_is_wellformed_transaction(1)[OF T_adm]]} \)
unfolding \( \text{ts_def trms_transaction_unfold trms_list,sst, is_trms,sst, [symmetric]} \text{ sset TAtom''(2)} \)
by \( \text{fastforce} \)

have \( "f x = \varnothing 0 x" \) for \( x \)
proof (cases "fst x = Var Value ∧ x ∈ fv_transaction T ∧ x /∈ set (transaction_fresh T)"
  case True
  hence "∅ x = {ab ∈ set OCC. poss x ⊆ ab ∧ negs x ∩ ab = {}}" unfolding ∅_def by argo
  thus ?thesis
    using True transaction_check_variant_soundness_aux0[OF S_def xs_def, of x]
      transaction_check_variant_soundness_aux1[OF C_def S_def xs_def pos_def negs_def as_def f_def, of x]
      by simp
next
  case False
  hence 0: "∅ x = {}" unfolding ∅_def by argo
  have "x /∈ set xs"
    using False fv_list_list_is_fv_sst[of "unlabel (transaction_strand T)"
    unfolding xs_def S_def by fastforce
  hence "List.find (λp. fst p = x) gs = None"
    using find_None_iff[of "λp. fst p = x" gs]
      unfolding gs_def by simp
  hence "f x = {}"
    unfolding f_def as_def gs_def g_def
    by force
  thus ?thesis using 0 by simp
qed

private lemma transaction_check_variant_soundness_aux5:
  fixes FP OCC TI T S C
  defines "msgcs ≡ λx a. a ∈ synth_abs_substs_constrs (FP,OCC,TI) T x"
  and "S ≡ unlabel (transaction_strand T)"
  and "C ≡ unlabel (transaction_checks T)"
  and "xs ≡ filter (λx. x /∈ set (transaction_fresh T) ∧ fst x = TAtom Value) (fv_list_sst S)"
  and "poss ≡ transaction_poschecks_comp C"
  and "negs ≡ transaction_negchecks_comp C"
  assumes T_adm: "admissible_transaction T"
  and TI: "∀(a,b) ∈ set TI. ∀(c,d) ∈ set TI. b = c ∧ a ≠ d −→ (a,d) ∈ set TI"
  and OCC: "∀t ∈ set FP. ∀a. Abs a ∈ funs_term t −→ a ∈ set OCC"
  and FP: "ground (set FP)"
  and δ: "δ ∈ abs_substs_fun ` set (abs_substs_set xs OCC poss negs (λ_ _. True))"
  "transaction_check_pre (FP,OCC,TI) T δ"
  shows "δ ∈ abs_substs_fun ` set (abs_substs_set xs OCC poss negs msgcs)"
proof -
  have 0: "δ x ∈ set OCC" "poss x ⊆ δ x" "δ x ∩ negs x = {}" when x: "x ∈ set xs" for x
    using abs_substs_abss_bounded[OF δ(1) x] by simp_all
  have 1: "δ x = {}" when x: "x /∈ set xs" for x
    by (rule abs_substs_abss_bounded[OF δ(1) x])
  have 2: "msgcs x (δ x)" when x: "x ∈ set xs" for x
    using 0 1 x transaction_check_variant_soundness_aux4[OF T_adm TI OCC FP, of x δ]
      transaction_check_pre_ReceiveE[OF δ(2)]
      transaction_check_pre_InSetE[OF δ(2)]
      transaction_check_pre_NotInSetE[OF δ(2)]
    unfolding msgcs_def xs_def C_def S_def negs_def pos_def by fast
show ?thesis
  using abs_substs_has_abs[of xs δ OCC poss negs msgcs] 0 1 2 by blast
qed

lemma transaction_check_variant_soundness:
assumes P_adm: "∀ T ∈ set P. admissible_transaction T"
and TI: "∀ (a,b) ∈ set TI. ∀ (c,d) ∈ set TI. b = c ∧ a ≠ d --> (a,d) ∈ set TI"
and OCC: "∀ t ∈ set FP. ∀ a. Abs a ∈ funs_term t --> a ∈ set OCC"
and FP: "ground (set FP)"
and T_in: "T ∈ set P"
and T_check: "transaction_check_alt1 (FP,OCC,TI) T"
shows "transaction_check (FP,OCC,TI) T"
proof -
  have 0: "admissible_transaction T"
  using P_adm T_in by blast
  show ?thesis using T_check transaction_check_variant_soundness_aux5[OF 0 TI OCC FP]
  unfolding transaction_check_def transaction_check'_def transaction_check_alt1_def
  transaction_check_comp_def list_all_iff Let_def
  by force
qed
end

3.6.6 Automatic Fixed-Point Computation

context stateful_protocol_model
begin

definition compute_fixpoint_fun' where
  "compute_fixpoint_fun' P (n::nat option) enable_traces Δ S0 ≡
  let sy = intruder_synth_mod_timpls;
  FP' = λS. fst (fst S);
  TI' = λS. snd (fst S);
  OCC' = λS. remdups (map (λt. the_Abs (the_Fun (args t ! 1)))
  (filter (λt. is_Fun t ∧ the_Fun t = OccursFact) (FP' S)))@
  (map snd (TI' S));
  equal_states = λS S'. set (FP' S) = set (FP' S') ∧ set (TI' S) = set (TI' S');
  trace' = λS. snd S;
  close = λM f. let g = remdups o f in while (∀A. set (g A) ≠ set A) g M;
  close' = λM f. let g = remdups o f in while (∀A. set (g A) ≠ set A) g M;
  trancl_minus_refl = λTI.
  let aux = λts p. map (λq. (fst p,snd q)) (filter ((=) (snd p) o fst) ts)
in filter (∀. fst p ≠ snd p) (close' TI (λts. concat (map (aux ts) ts)))@ts);
  snd_Ana = λN M TI. let N' = filter (λt. ¬ sy M TI t) N in
  filter (λt. ¬ sy M TI t) (concat (map (λt. filter (λs. s ∈ set (snd (Ana t))) (args t))) N');
  Ana_cl = λFP TI.
  close FP (λM. (M#snd_Ana M M TI));
  TI_cl = λFP TI.
  close FP (λM. (M#filter (λt. ¬ sy M TI t) (concat (map (λm. concat (map (λ(a,b). (a --> b)|m) TI)) M)))))
  Ana_cl' = λFP TI.

252
let K = λt. set (fst (Ana t));

flt = λM t. (∃k ∈ K t. ¬sy M TI k) ∧ (∃k ∈ K t. ∃f ∈ funs_term k. is_Abs f);

N = λM. comp_timpl_closure_list (filter (flt M) M) TI
in close FP (λM. M@snd_Ana (N M) M TI);

Δ' = λS. (FP' S, OCC' S, TI' S);

result = λS T δ.
  let not_fresh = λx. x /∈ set (transaction_fresh T);
  xs = filter not_fresh (fv_list sst (unlabel (transaction_strand T)));
  u = λδ x. absdbupd (unlabel (transaction_strand T)) x (δ x)
  in (remdups (filter (λt. ¬sy (FP' S) (TI' S) t)
  (concat (map (λts. the_msgs ts · list (absc ◦ u δ))
  (filter is_Send (unlabel (transaction_send T)))))),
  remdups (filter (λx. fst x ≠ snd x) (concat (map snd U)@TI' S)),
  trace);

result_tuple = λS T δ.
  let results = map (λT. map (result_tuple S T) (abs_substs_fun δ)) P;
  newtrace_flt = (λn. let x = map fst (results ! n); y = map snd x
  in set (concat y) ≠ set (FP' S) \ newtrace_flt 
  (λn. let x = map fst (results ! n); y = map snd x
  in set (concat y) ≠ set (TI' S));
  trace = (Ana_cl (TI_cl (FP' V) (TI' V)) (TI' V),
  trancl_minus_refl (TI' V)),
  trace' V)
  in if ¬equal_states W S then W
  else let results = remdups (filter (λx. ¬equal_states S (h S) x
  (Some m ⇒ h ^^ m) update_state S0)
  S = (λh. case n of None ⇒ while (¬equal_states S (h S)) h | Some m ⇒ h ^^ m)
  in ((FP' S, OCC' S, TI' S), trace' S)

definition compute_fixpoint_fun
where
"compute_fixpoint_fun P ≡
  let P' = remdups (filter (λT. transaction_updates T ≠ [] ∨ transaction_send T ≠ [])) P;
  f = transaction_check_comp (λ_. True)
  in fst (compute_fixpoint_fun' P None False f (([],[]),[],[]))"

definition compute_fixpoint_with_trace
where
"compute_fixpoint_with_trace P ≡
  compute_fixpoint_fun' P None True (transaction_check_comp (λ_. True)) (([],[]),[],[])"

definition compute_fixpoint_from_trace
where
"compute_fixpoint_from_trace P ≡
  let Δ = λFPT T.
    let pre_check = transaction_check_pre FPT T;
    δs = map snd (filter (λ(i,as). P ! i = T) (concat trace))
    in filter (λδ. pre_check (abs_substs_fun δ) δs)
    f = compute_fixpoint_fun' ◦ map (nth P);
    g = (λL FPT. fst ((f L (Some 1) False Δ ((fst FPT,snd (snd FPT)),[],[],[])))
    in fold g (map (map fst) trace) ([],[],[],[]))"

definition compute_reduced_attack_trace
where
"compute_reduced_attack_trace P ≡
  let attack_in_fixpoint = list_ex (λt. ∃f ∈ funs_term t. is_Attack f) o fst;
  is_attack_trace = attack_in_fixpoint o compute_fixpoint_from_trace P;"
3 Stateful Protocol Verification

trace' =
let is_attack_transaction =
  list_ex is_Fun_Attack ◦ concat ◦ map the_msgs ◦
  filter is_Send ◦ unlabel ◦ transaction_send;
trace' =
  if trace = [] then []
  else butlast trace@[filter (is_attack_transaction ◦ nth P ◦ fst) (last trace)]
in trace';

iter = λtrace_prev trace_rest elem (prev,rest).
let next =
  if is_attack_trace (trace_prev@(prev@rest)#trace_rest)
  then prev
  else prev@[elem]
in (next, tl rest);
iter' = λtrace_part (trace_prev,trace_rest).
let updated = foldr (iter trace_prev (tl trace_rest)) trace_part ([],tl (rev trace_part))
in (trace_prev@[rev (fst updated)], tl trace_rest);
reduced_trace = fst (fold iter' trace' ([],trace'))
in concat reduced_trace"

end

3.6.7 Locales for Protocols Proven Secure through Fixed-Point Coverage

type_synonym ('f,'a,'s,'l) fixpoint_triple =
"('f,'a,'s,'l) prot_term list × 's set list × ('s set × 's set) list"

context stateful_protocol_model
begin

definition "attack_notin_fixpoint (FPT::('fun,'atom,'sets,'lbl) fixpoint_triple) ≡
  list_all (λt. ∀f ∈ funs_term t. ¬is_Attack f) (fst FPT)"

definition "protocol_covered_by_fixpoint (FPT::('fun,'atom,'sets,'lbl) fixpoint_triple) P ≡
  list_all (transaction_check FPT)
  (filter (λT. transaction_updates T \neq [] \lor transaction_send T \neq [])) P"

definition "protocol_covered_by_fixpoint_alt1 (FPT::('fun,'atom,'sets,'lbl) fixpoint_triple) P ≡
  list_all (transaction_check_alt1 FPT)
  (filter (λT. transaction_updates T \neq [] \lor transaction_send T \neq [])) P"

definition "analyzed_fixpoint (FPT::('fun,'atom,'sets,'lbl) fixpoint_triple) ≡
  let (FP, _, TI) = FPT
  in analyzed_closed_mod_timpls FP TI"

definition "wellformed_protocol_SMP_set (P::('fun,'atom,'sets,'lbl) prot) N ≡
  has_all_wt_instances_of Γ (∪T ∈ set P. trms_transaction T) (set N) \land
  comp_tfr, set arity Ana Γ (set N) \land
  list_all (ΛT. list_all (comp_tfr, sstp Γ Pair) (unlabel (transaction_strand T))) P"

definition "wellformed_protocol' (P::('fun,'atom,'sets,'lbl) prot) N ≡
  list_all admissible_transaction P \land
  wellformed_protocol_SMP_set P N"

definition "wellformed_protocol (P::('fun,'atom,'sets,'lbl) prot) ≡
  let f = λM. remdups (concat (map subterms_list M@map (fst ◦ Ana) M));
  NO = remdups (concat (map (trms_list sst ◦ unlabel ◦ transaction_strand) P));
  N = while (λA. set (f A) \neq set A) f NO
  in wellformed_protocol' P N"

definition "wellformed_fixpoint' (FPT::('fun,'atom,'sets,'lbl) fixpoint_triple) ≡
3.6 Stateful Protocol Verification

let (FP, OCC, TI) = FPT; OCC' = set OCC
in list_all (\t. \. \( \lambda \. a \in OCC\). (map snd TI) \∧
list_all (\t. \( \lambda \. f \in funs_term t. is_Abs f \→ the_Abs f \in OCC\)) FP) \∧
list_all (\a. a \∈ OCC') (map snd TI) \∧
list_all (\t. \( \lambda \. \\forall f \in funs_term t. is_Abs f \→ the_Abs f \in OCC\)) FP

definition "wellformed_term_implication_graph (FPT::('fun,'atom,'sets,'lbl) fixpoint_triple) \equiv
let (_, _, TI) = FPT
in list_all (\(a,b). list_all (\(c,d). b = c \∧ a \neq d \→ List.member TI (a,d)) TI) TI \∧
list_all (\p. fst p \neq snd p) TI"
definition "wellformed_fixpoint (FPT::('fun,'atom,'sets,'lbl) fixpoint_triple) \equiv
wellformed_fixpoint' FPT \∧
wellformed_term_implication_graph FPT"
definition "wellformed_protocol_SMP_set_mono:
assumes "wellformed_protocol_SMP_set P S"
and "\set P' \𝑆⊆ \set P"
shows "wellformed_protocol_SMP_set P' S"
using assms
unfolding wellformed_protocol_SMP_set_def comp_tfr, set_def has_all_wt_instances_of_def
wf trms'_def list_all_iff
by fast
lemma wellformed_protocol'_mono:
assumes "wellformed_protocol' P S"
and "\set P' \𝑆⊆ \set P"
shows "wellformed_protocol' P' S"
using assms wellformed_protocol_SMP_set_mono[of P S P']
unfolding wellformed_protocol'_def list_all_iff by blast
lemma protocol_covered_by_fixpoint_if_protocol_covered_by_fixpoint_alt1:
assumes P: "wellformed_protocol' P P_SMP"
and FPT: "wellformed_fixpoint FPT"
and covered: "protocol_covered_by_fixpoint_alt1 FPT P"
shows "protocol_covered_by_fixpoint FPT P"
proof -
obtain FP OCC TI where FPT': "FPT = (FP, OCC, TI)" by (metis surj_pair)

have P_adm: "\∀ T \∈ \set P. admissible_transaction T"
and TI: "\∀ (a,b) \∈ \set TI. \∀ (c,d) \∈ \set TI. b = c \∧ a \neq d \→ (a,d) \∈ \set TI"

have OCC: "\∀ t \∈ \set FP. \∀ a. Abs a \in funs_term t \→ a \in \set OCC"
and FP: "\ground (set FP)"
"\wf trms (set FP)"

using P FPT unfolding defs by (blast, simp, simp, fastforce, simp, simp)

show ?thesis
using covered transaction_check_variant_soundness[OF P_adm TI OCC FP]
unfolding protocol_covered_by_fixpoint_def protocol_covered_by_fixpoint_alt1_def
FPT' list_all_iff
by fastforce
qed

lemma protocol_covered_by_fixpoint_if_protocol_covered_by_fixpoint_alt1':
assumes P: "wellformed_protocol' P P_SMP"
and P': "\set P' \Ｓ⊆ \set P"
and FPT: "wellformed_fixpoint FPT"
and covered: "protocol_covered_by_fixpoint_alt1 FPT P"
shows "protocol_covered_by_fixpoint FPT P"
using protocol_covered_by_fixpoint_if_protocol_covered_by_fixpoint_alt1[OF _ FPT covered]  
wellformed_protocol'_mono[OF P P']

by simp

lemma protocol_covered_by_fixpoint_trivial_case:  
  assumes "list_all (λ T. transaction_updates T = [] ∧ transaction_send T = []) P"  
  shows "protocol_covered_by_fixpoint FPT P"
using assms
by (simp add: list_all_iff transaction_check_trivial_case protocol_covered_by_fixpoint_def)

lemma protocol_covered_by_fixpoint_empty[simp]:  
  "protocol_covered_by_fixpoint FPT []"
by (simp add: protocol_covered_by_fixpoint_def)

lemma protocol_covered_by_fixpoint_Cons[simp]:  
  "protocol_covered_by_fixpoint FPT (T#P) ←→  
    transaction_check FPT T ∧ protocol_covered_by_fixpoint FPT P"
using transaction_check_trivial_case  
unfolding protocol_covered_by_fixpoint_def case_prod_unfold
by simp

lemma protocol_covered_by_fixpoint_append[simp]:  
  "protocol_covered_by_fixpoint FPT (P1@P2) ←→  
    protocol_covered_by_fixpoint FPT P1 ∧ protocol_covered_by_fixpoint FPT P2"
by (simp add: protocol_covered_by_fixpoint_def case_prod_unfold)

lemma protocol_covered_by_fixpoint_I1[intro]:  
  assumes "list_all (protocol_covered_by_fixpoint FPT) P"  
  shows "protocol_covered_by_fixpoint FPT (concat P)"
using assms
by (auto simp add: protocol_covered_by_fixpoint_def list_all_iff)

lemma protocol_covered_by_fixpoint_I2[intro]:  
  assumes "protocol_covered_by_fixpoint FPT P1"  
  and "protocol_covered_by_fixpoint FPT P2"  
  shows "protocol_covered_by_fixpoint FPT (P1@P2)"
using assms
by (auto simp add: protocol_covered_by_fixpoint_def)

lemma protocol_covered_by_fixpoint_I3:  
  assumes "∀ T ∈ set P.  ∀ δ ::('fun,'atom,'sets,'lbl) prot_var ⇒ 'sets set.  
    transaction_check_pre FPT T δ −→ transaction_check_post FPT T δ"  
  shows "protocol_covered_by_fixpoint FPT P"
using assms
unfolding protocol_covered_by_fixpoint_def transaction_check_def transaction_check'_def 
transaction_check_comp_def list_all_iff Let_def case_prod_unfold  
Product_Type.fst_conv Product_Type.snd_conv
by fastforce

lemmas protocol_covered_by_fixpoint_intros =  
protocol_covered_by_fixpoint_I1  
protocol_covered_by_fixpoint_I2  
protocol_covered_by_fixpoint_I3

lemma prot_secure_if_prot_checks:  
  fixes P::"('fun,'atom,'sets,'lbl) prot_transaction list"  
  and FP_OCC_TI:: "('fun,'atom,'sets,'lbl) fixpoint triple"  
  assumes attack_notin_fixpoint: "attack_notin_fixpoint FP_OCC_TI"  
  and transactions_covered: "protocol_covered_by_fixpoint FP_OCC_TI P"  
  and analyzed_fixpoint: "analyzed_fixpoint FP_OCC_TI"  
  and wellformed_protocol: "wellformed_protocol' P N"  
  and wellformed_fixpoint: "wellformed_fixpoint FP_OCC_TI"  
  shows "∀ A ∈ reachable_constraints P.  ∀ I.  
    constraint_model I (A@[(l, send ⟨[attack⟨n⟩]⟩)])"
proof -  
define FP where "FP ≡ let (FP,_,_) = FP_OCC_TI in FP"  
define OCC where "OCC ≡ let (_,OCC,_) = FP_OCC_TI in OCC"
3.6 Stateful Protocol Verification

define TI where "TI \equiv \text{let } (\_, \_, TI) = \text{FP\_OCC\_TI in } TI"

have attack_notin_FP: "\text{attack}(n) \notin \text{set } FP"
  using attack_notin_fixpoint
  unfolding list_all_iff FP_def by force

have 1: "\forall (a,b) \in \text{set } TI. \forall (c,d) \in \text{set } TI. b = c \land a \neq d \rightarrow (a,d) \in \text{set } TI"
  using wellformed_fixpoint
  unfolding wellformed_fixpoint_def wf\_trms\_code[symmetric] Let_def TI_def
  list_all_iff member_def case_prod_unfold
  wellformed_term_implication_graph_def
  by auto

have 0: "\text{wf\_trms\_code (set } FP)"
and 2: "\forall (a,b) \in \text{set } TI. a \neq b"
and 3: "\text{snd } \text{set } TI \subseteq \text{set OCC}"
and 4: "\forall t \in \text{set } FP. \forall f \in \text{funs\_term } t. \text{is\_Abs } f \rightarrow f \in \text{Abs } \text{set OCC}"
and 5: "\text{ground (set } FP)"
  using wellformed_fixpoint
  unfolding wellformed_fixpoint_def wf\_trms\_code[symmetric] is\_Abs_def the\_Abs_def
  list_all_iff Let_def case_prod_unfold set_map FP_def OCC_def TI_def
  wellformed_fixpoint'_def wellformed_term_implication_graph_def
  by (fast, fast, blast, fastforce, simp)

have 8: "\text{finite (set } N)"
and 9: "\text{has\_all\_wt\_instances\_of } \Gamma (\bigcup T \in \text{set } P. \text{trms\_transaction } T) (\text{set } N)"
and 10: "\text{tfr\_set (set } N)"
and 11: "\forall T \in \text{set } P. \text{list\_all } \text{tfr\_stp} (\text{unlabel (transaction\_strand } T))"
and 12: "\forall T \in \text{set } P. \text{admissible\_transaction } T"
  using wellformed_protocol
  unfolding wellformed_protocol'_def wellformed_protocol_SMP_set_def
  \text{tfr\_set\_if\_comp\_tfr\_set\_def (of } \text{set } N)"
  unfolding Let_def list_all_iff wf\_trms\_code[symmetric] \text{tfr\_stp\_is\_comp\_tfr\_stp\_symmetric} by fast+

have 13: "\text{wf\_trms\_code (set } N)"
  using wellformed_protocol
  unfolding wellformed_protocol'_def wellformed_protocol_SMP_set_def
  \text{finite\_SMP\_representationD} by fast+

note TIO = trancl_eqI'[OF 1 2]

have "\text{analyzed (timpl\_closure\_set (set } FP) (\text{set } TI))"
  using analyzed_fixpoint
  unfolding analyzed_closed_mod_timpls_is_analyzed_timpl_closure_set[OF TIO 0]
  unfolding \text{FP\_def } TI_def
  by force

note FP0 = this 0 5

note OCC0 = funs_term_OCC_TI_subset(1)[OF 4 3]
  timpl\_closure\_set\_supset'[OF funs_term_OCC_TI_subset(2)][OF 4 3]]

note M0 = 9 8 10 13

have "\text{list\_all (transaction\_check (FP, OCC, TI)) } P"
  using transactions_covered
  unfolding \text{transaction\_check\_trivial\_case[of } _ FP\_OCC\_TI]
  transaction_check_trivial_case
  \text{FP\_def } OCC\_def TI\_def list\_all\_iff Let\_def case\_prod\_unfold
  by auto

note P0 = 12 11 this attack_notin_FP

show ?thesis by (metis prot\_secure\_if\_fixpoint\_covered[OF FP0 OCC0 TIO M0 P0])

qed
lemma prot_secure_if_prot_checks_alt1:
  fixes P::"('fun,'atom,'sets,'lbl) prot_transaction list"
  and FP_OCC_TI:: "('fun,'atom,'sets,'lbl) fixpoint_triple"
  assumes attack_notin_fixpoint: "attack_notin_fixpoint FP_OCC_TI"
  and transactions_covered: "protocol_covered_by_fixpoint_alt1 FP_OCC_TI P"
  and analyzed_fixpoint: "analyzed_fixpoint FP_OCC_TI"
  and wellformed_protocol: "wellformed_protocol' P N"
  and wellformed_fixpoint: "wellformed_fixpoint FP_OCC_TI"
  shows "\forall A \in pm.reachable_constraints P. \exists I. pm.constraint_model I (A@[1, send([attack(n)])])"
using prot_secure_if_prot_checks[of attack_notin_fixpoint transactions_covered analyzed_fixpoint wellformed_protocol wellformed_fixpoint]
protocol_covered_by_fixpoint_if_protocol_covered_by_fixpoint_alt1[of wellformed_protocol wellformed_fixpoint transactions_covered]
by blast
end
locale secure_stateful_protocol =
  pm: stateful_protocol_model arity f arity s public f Ana f Γ f label_witness1 label_witness2
  for arity::"'fun \Rightarrow nat"
  and arity::"'sets \Rightarrow nat"
  and public::"'fun \Rightarrow bool"
  and Ana::"'fun \Rightarrow ((('fun,'atom::finite,'sets,'lbl) prot_fun, nat) term list \times nat list)"
  and Γ::"'fun \Rightarrow 'atom option"
  and label_witness1::"'lbl"
  and label_witness2::"'lbl"
+ fixes P::"('fun,'atom,'sets,'lbl) prot_transaction list"
  and FP_OCC_TI:: "('fun,'atom,'sets,'lbl) fixpoint_triple"
  and P_SMP::"('fun,'atom,'sets,'lbl) prot_term list"
  assumes attack_notin_fixpoint: "pm.attack_notin_fixpoint FP_OCC_TI"
  and transactions_covered: "pm.protocol_covered_by_fixpoint FP_OCC_TI P"
  and analyzed_fixpoint: "pm.analyzed_fixpoint FP_OCC_TI"
  and wellformed_protocol: "pm.wellformed_protocol' P P_SMP"
  and wellformed_fixpoint: "pm.wellformed_fixpoint FP_OCC_TI"
begin
  theorem protocol_secure:
  "\forall A \in pm.reachable_constraints P. \exists I. pm.constraint_model I (A@[1, send([attack(n)])])"
by (rule pm.prot_secure_if_prot_checks[of attack_notin_fixpoint transactions_covered analyzed_fixpoint wellformed_protocol wellformed_fixpoint])
corollary protocol_welltyped_secure:
  "\forall A \in pm.reachable_constraints P. \exists I. pm.welltyped_constraint_model I (A@[1, send([attack(n)])])"
using protocol_secure unfolding pm.welltyped_constraint_model_def by fast
end
locale secure_stateful_protocol' =
  pm: stateful_protocol_model arity f arity s public f Ana f Γ f label_witness1 label_witness2
  for arity::"'fun \Rightarrow nat"
  and arity::"'sets \Rightarrow nat"
  and public::"'fun \Rightarrow bool"
  and Ana::"'fun \Rightarrow ((('fun,'atom::finite,'sets,'lbl) prot_fun, nat) term list \times nat list)"
  and Γ::"'fun \Rightarrow 'atom option"
  and label_witness1::"'lbl"
  and label_witness2::"'lbl"
+ fixes P::"('fun,'atom,'sets,'lbl) prot_transaction list"
  and FP_OCC_TI:: "('fun,'atom,'sets,'lbl) fixpoint_triple"
assumes attack_notin_fixpoint': "pm.attack_notin_fixpointFP_OCC_TI"
and transactions_covered': "pm.protocol_covered_by_fixpointFP_OCC_TIP"
and analyzed_fixpoint': "pm.analyzed_fixpointFP_OCC_TI"
and wellformed_protocol': "pm.wellformed_protocolP"
and wellformed_fixpoint': "pm.wellformed_fixpointFP_OCC_TI"

begin

sublocale secure_stateful_protocol
arity, arity, public, Ana, Γ, label_witness1 label_witness2 P
FP_OCC_TI
"let f = λM. remdups (concat (map subterms_list M@map (fst ◦ pm.Ana) M));
N0 = remdups (concat (map (trms_list ◦ unlabel ◦ transaction_strand) P))
in while (λA. set (f A) ≠ set A) f N0"
apply unfold_locales
using attack_notin_fixpoint' transactions_covered' analyzed_fixpoint'
wellformed_protocol'['unfolded pm.wellformed_protocol_def Let_def]
wellformed_fixpoint'
unfolding Let_def by blast+
end

locale secure_stateful_protocol'' =
pm: stateful_protocol_model arity, arity, public, Ana, Γ, label_witness1 label_witness2
for arity:="fun ⇒ nat"
and arity,::="sets ⇒ nat"
and public::="fun ⇒ bool"
and Ana::="fun ⇒ ((('fun,atom::finite,'sets,'lbl) prot_fun, nat) term list × nat list)"
and Γ::="fun ⇒ 'atom option"
and label_witness1::="lbl"
and label_witness2::="lbl"
+ fixes P::"('fun,atom,'sets,'lbl) prot_transaction list"
assumes checks: "let FPT = pm.compute_fixpoint_fun P
in pm.attack_notin_fixpointFPT ∧ pm.protocol_covered_by_fixpointFPTP ∧
pm.analyzed_fixpointFPT ∧ pm.wellformed_protocolP ∧ pm.wellformed_fixpointFPT"

begin
sublocale secure_stateful_protocol'
arity, arity, public, Ana, Γ, label_witness1 label_witness2 P "pm.compute_fixpoint_fun P"
using checks['unfolded Let_def case_prod_unfold] by unfold_locales meson+
end

locale secure_stateful_protocol''' =
pm: stateful_protocol_model arity, arity, public, Ana, Γ, label_witness1 label_witness2
for arity:="fun ⇒ nat"
and arity,::="sets ⇒ nat"
and public::="fun ⇒ bool"
and Ana::="fun ⇒ ((('fun,atom::finite,'sets,'lbl) prot_fun, nat) term list × nat list)"
and Γ::="fun ⇒ 'atom option"
and label_witness1::="lbl"
and label_witness2::="lbl"
+ fixes P::"('fun,atom,'sets,'lbl) prot_transaction list"
and FP_OCC_TI:: "('fun,atom,'sets,'lbl) fixpoint_triple"
and P_SMP::"('fun,atom,'sets,'lbl) prot_term list"
assumes checks': "let P' = P; FPT = FP_OCC_TIP; P'_SMP = P_SMP
in pm.attack_notin_fixpointFPT ∧
pm.protocol_covered_by_fixpointFPTP' ∧
pm.analyzed_fixpointFPT ∧
pm.wellformed_protocolP' P'_SMPP ∧
pm.wellformed_fixpointFPT"
locale secure_stateful_protocol'''' =
  pm: stateful_protocol_model arity, arity, public, Ana, Γ, label_witness1 label_witness2
for arity:="fun ⇒ nat"
  and arity:="sets ⇒ nat"
  and public:="fun ⇒ bool"
  and Ana:="fun ⇒ ((('fun,'atom::finite,'sets,'lbl) prot_fun, nat) term list × nat list)"
  and Γ:="fun ⇒ 'atom option"
  and label_witness1:="lbl"
  and label_witness2:="lbl"
+ fixes P:="('fun,'atom,'sets,'lbl) prot_transaction list"
  and FP_OCC_TI:="('fun,'atom,'sets,'lbl) fixpoint_triple"
assumes checks'': "let P' = P; FPT = FP_OCC_TI
in pm.attack_notin_fixpoint FPT ∧
  pm.protocol_covered_by_fixpoint FPT P' ∧
  pm.analyzed_fixpoint FPT ∧
  pm.wellformed_protocol P' ∧
  pm.wellformed_fixpoint FPT"
begin
sublocale secure_stateful_protocol
  arity, arity, public, Ana, Γ, label_witness1 label_witness2 P FP_OCC_TI P_SMP
using checks''[unfolded Let_def case_prod_unfold] by unfold_locales meson+
end
locale secure_stateful_protocol_alt1 =
  pm: stateful_protocol_model arity, arity, public, Ana, Γ, label_witness1 label_witness2
for arity:="fun ⇒ nat"
  and arity:="sets ⇒ nat"
  and public:="fun ⇒ bool"
  and Ana:="fun ⇒ ((('fun,'atom::finite,'sets,'lbl) prot_fun, nat) term list × nat list)"
  and Γ:="fun ⇒ 'atom option"
  and label_witness1:="lbl"
  and label_witness2:="lbl"
+ fixes P:="('fun,'atom,'sets,'lbl) prot_transaction list"
  and FP_OCC_TI:="('fun,'atom,'sets,'lbl) fixpoint_triple"
assumes attack_notin_fixpoint_alt1: "pm.attack_notin_fixpoint FP_OCC_TI"
  and transactions_covered_alt1: "pm.protocol_covered_by_fixpoint_alt1 FP_OCC_TI P"
  and analyzed_fixpoint_alt1: "pm.analyzed_fixpoint FP_OCC_TI"
  and wellformed_protocol_alt1: "pm.wellformed_protocol' P P_SMP"
  and wellformed_fixpoint_alt1: "pm.wellformed_fixpoint FP_OCC_TI"
begin
sublocale secure_stateful_protocol
  arity, arity, public, Ana, Γ, label_witness1 label_witness2 P FP_OCC_TI P_SMP
using pm.protocol_covered_by_fixpoint_if_protocol_covered_by_fixpoint_alt1[OF wellformed_protocol_alt1 wellformed_fixpoint_alt1 transactions_covered_alt1]
  attack_notin_fixpoint_alt1 analyzed_fixpoint_alt1 wellformed_protocol_alt1 wellformed_fixpoint_alt1
by unfold_locales meson+
end
3.6.8 Automatic Protocol Composition

definition welltyped_leakage_free_protocol where "welltyped_leakage_free_protocol S P ≡ let f = λ M. let f = λ M. {t · δ | t · δ. t ∈ M ∧ wt subst δ ∧ wf trms (subst_range δ) ∧ fv (t · δ) = {}}; Sec = (f (set S)) - {m. intruder synth {} m}; in ∀ A ∈ reachable_constraints P. ∀ I τ s. (∃ l ts. suffix [(l, receive ⟨ts⟩)]) A ∧ s ∈ Sec - declassified lsst A I τ ∧ welltyped constraint model I τ (A# ((x, send ⟨[s]⟩)))"

definition wellformed_composable_protocols where "wellformed_composable_protocols (P::('fun,'atom,'sets,'lbl) prot list) N ≡ let T s = concat P; steps = remdups (concat (map transaction strand Ts)); MPO = \bigcup T \in set Ts. trms transaction T \cup Pair` setops transaction T in list_all (wf trm arity) N ∧ has_all wt instances of Γ MPO (set N) ∧ comp tfr arity Ana Γ (set N) ∧ list_all (comp tfr sstp, Γ Pair o snd) steps ∧ list_all wellformed transaction Ts ∧ list_all admissible transaction terms Ts ∧ list_all (list_all (λ x. Γ v x = TAtom Value ∨ (is Var (Γ v x) ∧ is Atom (the Var (Γ v x)))) o transaction_fresh) Ts ∧ list_all (list_all (λ p. let (x,cs) = p in is_Var (Γ v x) ∧ is Atom (the Var (Γ v x)) ∧ (∀ c ∈ cs. Γ v x = Γ (Fun (Fu c) [])::('fun,'atom,'sets,'lbl) prot_term)) o transaction_decl T ())) Ts ∧ list_all (λ T. ∀ x ∈ vars transaction T. ¬ TAtom AttackType ⊑ Γ v x) Ts ∧ list_all (λ T. ∀ f ∈ funs term (Γ v x). f ≠ Pair ∧ f ≠ OccursFact) Ts ∧ list_all (list_all (λ s. is Send (snd s) ∧ length (the msgs (snd s)) = 1 ∧ is Fun Attack (hd (the msgs (snd s))) → the Attack label (the Fun (hd (the msgs (snd s)))) = fst s) o transaction_strand) Ts ∧ list_all (λ r. (∃ f ∈ (funss term `(trms sstp, and r))). f = OccursFact ∨ f = OccursSec → (is Receive (snd r) ∨ is Send (snd r)) ∧ fst r = * ∧ (∀ t ∈ set (the msgs (snd r)). (OccursFact ∈ funs term t ∨ OccursSec ∈ funs term t) → is Fun t ∧ length (args t) = 2 ∧ t = occurs (args t ! 1) ∧ is Var (args t ! 1) ∧ (Γ (args t ! 1) = TAtom Value))) steps”

definition composable_protocols where "composable_protocols (P::('fun,'atom,'sets,'lbl) prot list) Ms S ≡ let steps = concat (map transaction strand (concat P)); M fun = (λ l. case find ((=) l o fst) Ms of Some M ⇒ set (snd M) | None ⇒ {} in comp par comp Ms sstp public arity Ana Γ Pair steps M fun (set S))"

lemma composable_protocols_par_comp_constr: fixes S f defines "f ≡ λ M. {t · δ | t · δ. t ∈ M ∧ wt subst δ ∧ wf trms (subst_range δ) ∧ fv (t · δ) = {}}" and "Sec ≡ (f (set S)) - {m. intruder synth {} m}"
assumes Ps_pc: "wellformed_composable_protocols Ps N" "composable_protocols Ps Ms S"
3 Stateful Protocol Verification

shows "∀A ∈ reachable_constraints (concat Ps). ∀I. constraint_model I A →
(∃I₁. welltyped_constraint_model I₁ A ∧
(∀n. welltyped_constraint_model I₁ (proj n A)) ∧
(∃A' l t. prefix A' A ∧ suffix [(1, receive⟨t⟩)] A' ∧
strand_leaks lsst A Sec I₁)))"
(is "∀A ∈ __. ∀I. constraint_model I A → ?Q A I")

proof (intro allI ballI impI)
  fix A I
  assume A: "A ∈ reachable_constraints (concat Ps)" and I: "constraint_model I A"

  let ?Ts = "concat Ps"
  let ?steps = "concat (map transaction_strand ?Ts)"
  let ?M0 = "∪T ∈ set ?Ts. trms_transaction T ∪ pair' Pair ` setops_transaction T"
  let ?M_fun = "λl. case find ((=) l ◦ fst) Ms of Some M ⇒ set (snd M) | None ⇒ {}"

  have M:
    "has_all_wt_instances_of Γ ?M0 (set N)"
    "finite (set N)" "wf_trms (set N)"
  using Ps_pc tfr set_if_comp_tfr[of "set N"]
  unfolding composable_protocols_def wellformed_composable_protocols_def
    Let_def list_all_iff wf_trms_code[ symmetric ]
  by (fast+)

  have P:
    "∀T ∈ set ?Ts. wellformed_transaction T"
    "∀T ∈ set ?Ts. wf_trms' arity (trms_transaction T)"
    "∀T ∈ set ?Ts. list_all ws_trms (unlabel (transaction_strand T))"
    "comp_par_comp lsst public arity Ana Γ Pair ?steps ?M_fun (set S)"
  using Ps_pc tfr_solve_leq_comp_trms[p_of "set S"]
  unfolding wellformed_composable_protocols_def composable_protocols_def
    Let_def list_all_iff unlabel_def wf_trms_code[ symmetric ]
    admissible_transaction_terms_def
  by (meson, meson, fastforce, blast)

  show "?Q A I"
  using reachable_constraints_par_comp_constr[of M P A I]
  unfolding Sec_def f_def
  by fast

  qed

context
begin

private lemma reachable_constraints_no_leakage_alt_aux:
  fixes P lbls L
  defines "lbls ≡ λT. map (the_labelN o fst) (filter (Not o has_LabelS) (transaction_strand T))"
  and "L ≡ set (remdups (concat (map lbls P)))"
  assumes l: "l /∈ L"
  shows "map (transaction_proj l) P = map transaction_star_proj P"

proof -
  have 0: "¬list_ex (has_LabelN l) (transaction_strand T)" when "T ∈ set P" for T
    using that unfolding L_def lbls_def list_ex_iff by force

  have 1: "¬list_ex (has_LabelN l) (transaction_strand T)"
    when T: "T ∈ set (map (transaction_proj l) P)" for T
    proof -
      obtain T' where T': "T' ∈ set P" "T = transaction_proj l T'" using T by auto
      show ?thesis
        using T'(2) 0[of T'(1)] proj_set_subset[of l "transaction_strand T'"]
        transaction_strand_proj[of l T']
      unfolding list_ex_iff by fastforce
    qed

  have "list_all has_LabelS (transaction_strand T)"
    when "T ∈ set (map (transaction_proj l) P)" for T
proof
  (intro allI ballI)
  has_LabelS_proj_iff_not_has_LabelN[of 1 "transaction_proj 1 T"]
  by (metis (no_types) ex_map_conv)
thus thesis
  using transaction_star_proj_ident_iff transaction_proj_member
  transaction_star_proj_negates_transaction_proj(1)
by (metis (mono_tags, lifting) map_eq_conv)
qed

private lemma reachable_constraints_star_no_leakage:
  fixes Sec P lbls k
  defines "no_leakage ≡ λA. \∀I. A' s.
  prefix A' ⋁ A ∧ (∃1 ts. suffix [(l, receive(ts))] A') ∧ s ∈ Sec - declassifiedₚₚ A' I, r
  welltyped_constraint_model I, (A'Φ[(k,send([s]))])"
  assumes Sec: "∀s ∈ Sec. ¬(t ⊢ t) "ground Sec"
  shows "∀A ∈ reachable_constraints (map transaction_star_proj P). no_leakage A"
proof
  fix A assume A: "A ∈ reachable_constraints (map transaction_star_proj P)"
  have A': "∀(l.a) ∈ set A. l = *"
  using reachable_constraints_preserves_labels[of A] transaction_star_proj_has_star_labels
  unfolding list_all_iff by fastforce
  show "no_leakage A" using constr_sem_stateful_star_proj_no_leakage[of Sec(2) A']
  unfolding unlabel_append[of A] singleton_lst_proj(4)[of k]
  unfolding no_leakage_def welltyped_constraint_model_def constraint_model_def by fastforce
qed

private lemma reachable_constraints_no_leakage_alt:
  fixes Sec P lbls k
  defines "no_leakage ≡ λA. \∀I. A' s.
  prefix A' ⋁ A ∧ (∃1 ts. suffix [(l, receive(ts))] A') ∧ s ∈ Sec - declassifiedₚₚ A' I, r
  welltyped_constraint_model I, (A'Φ[(k,send([s]))])"
  and " lbls ≡ \∀I. map (the_LabelN o fst) (filter (Not o has_LabelS) (transaction_strand T))"
  and "L ≡ set (remdup (concat (map lbls P)))"
  assumes Sec: "∀s ∈ Sec. ¬(t ⊢ t) "ground Sec"
  and lbl: "∀l ∈ L. ∀A ∈ reachable_constraints (map (transaction_proj l) P). no_leakage A"
  shows "∀l. ∀A ∈ reachable_constraints (map (transaction_proj l) P). \∀I. A'.
  interpretationₚₚₚₚ I, r ∧ wt_sub Iₚₚₚₚ I, r ∧ w_terms (sub_range I, r) ∧
  prefix A' ⋁ A ∧ (∃1 ts. suffix [(l', receive(ts))] A') ∧ strand_leaksₚₚₚₚ A' Sec I, r"
proof (intro allI ballI)
  fix l A
  assume A: "A ∈ reachable_constraints (map (transaction_proj l) P)"
  let ?Q = "∀I, r. A'.
  interpretationₚₚₚₚ I, r ∧ wt_sub Iₚₚₚₚ I, r ∧ w_terms (sub_range I, r) ∧
  prefix A' ⋁ A ∧ (∃1 ts. suffix [(l', receive(ts))] A') ∧ strand_leaksₚₚₚₚ A' Sec I, r"
  show "∀I. A'. ?Q I, A'" proof
    assume "∃I, r. A'. ?Q I, A'" proof
    then obtain I, r, l' ts' where
      Iₚₚₚₚ: "interpretationₚₚₚₚ I, r" "prefix A' ⋁ A" "suffix [(l', receive(ts))] A'" and
      t: "t ∈ Sec - declassifiedₚₚₚₚ A' I, r" and
      n: "prefix A' ⋁ A" "suffix [(l', receive(ts))] A'" and
      l: "∀l ∈ L.∀A ∈ reachable_constraints (map (transaction_proj l) P). no_leakage A"
    unfolding strand_leaksₚₚₚₚ_def by blast
    hence 0: "welltyped_constraint_model I, r (proj n A'Φ[(m,send([t]))])" for m
    unfolding welltyped_constraint_model_def constraint_model_def by fastforce
    have t_Sec: "¬(t ⊢ t) "t · I, r = t"
3 Stateful Protocol Verification

using \( t \) \( \text{Sec} \) \( \text{substate} \) \( \text{ground \ ident} \) \( [\text{of} \ t \ I,] \) by \( \text{auto} \)

obtain \( B \) \( k' \) \( s \) where \( B: \)

\[
\text{"constr}\_\text{sem}\_\text{stateful} \ I, \ (\text{proj}\_\text{unl} \ n \ B@\{\text{send} (t)\})"
\]

\[
\text{"prefix} \ (\text{proj} \ n \ B) \ (\text{proj} \ n \ A)" \text{"suffix} \ [\text{(} k', \ \text{receive}(s)\text{)}] \ (\text{proj} \ n \ B)"
\]

\[
\text{"} t \in \text{Sec} \ \text{declassified}_s \ (\text{proj} \ n \ B) \ I,\text{"}
\]

using constr_sem_stateful \( \text{proj} \_\text{priv} \_\text{term} \_\text{prefix} \_\text{obtain}[OF \ A'(1) \ n \ t \_\text{Sec}] \) by \( \text{metis} \)

hence \( 1: \) \text{"welltyped}\_\text{constraint}\_\text{model} \ I, \ (\text{proj} \ n \ B@\{\text{send} (t)\})" for \( m \)

using 0 unfolding welltyped_constraint_model_def constraint_model_def by \( \text{fastforce} \)

note 2 = reachable_constr\_\text{transaction} \_\text{proj} \_\text{proj}\_\text{eq}

note 3 = reachable_constr\_\text{transaction} \_\text{proj} \_\text{star} \_\text{proj}

note 4 = reachable_constr\_\text{no}\_\text{leakage} \_\text{alt} \_\text{aux}

note star_case = 0 \( t \) \( \text{Sec}(1) \) reachable_constr\_\text{star} \_\text{no}\_\text{leakage}[OF \text{Sec}]

\[
\text{A'}(2) \ 3[\text{OF} \ A] \ \text{prefix} \_\text{proj}(1)[\text{OF} \ A'(1)]
\]

\[
\text{suffix} \_\text{proj}(1)[\text{OF} \ A'(2)] \ \text{declassified}_s \ \text{proj}\_\text{eq}
\]

note lbl_case = 0 \( t(1) \) \( A \ A' \) \( \text{lbl} 2(2) [\text{OF} \ A A'(1)] \)

show \( \text{False} \)

proof (cases \( "1 = n" \))

case True thus \( ?\text{thesis} \)

proof (cases \( "1 \in L'" \))

case False

hence \( \text{"map} \ (\text{transaction} \_\text{proj} \ l) \ P = \text{map} \ \text{transaction} \_\text{star} \_\text{proj} \ P" \)

using 4 unfolding \( L\_\text{def} \) \( \text{lbis} \_\text{def} \) by \( \text{fast} \)

thus \( ?\text{thesis} \)

using \( \text{lbl\_case}(1-4,7) \) \( \text{star\_case}(4,5) \) \( \text{True} \) by \( \text{metis} \)

qed (metis \( \text{lbis} \_\text{def} \) \( \text{star\_case}(4,5) \) \( \text{True} \) \( \text{by} \) \( \text{metis} \))

next

case False

hence \( \text{"no}\_\text{leakage} \ (\text{proj} \ n \ A)" \) using \( \text{star\_case}(4,6) \) unfolding \( \text{no}\_\text{leakage}\_\text{def} \) by \( \text{fast} \)

thus \( ?\text{thesis} \)

by (metis \( \text{lbis} \_\text{def} \) \( \text{no}\_\text{leakage}\_\text{def} \) \( \text{by} \) \( \text{metis} \))

qed

private lemma reachable_constr\_\text{no}\_\text{leakage} \_\text{alt} \_\text{aux1}:

fixes \( P::\text{"('a,'b,'c,'d) prot\_\text{transaction} list"} \)

defines \( f \equiv \text{list}\_\text{all} \ (\text{list}\_\text{all} (\text{Not} \circ \text{has\_LabelS}) \circ \text{tl} \circ \text{transaction} \_\text{send})" \)

assumes \( P: \) \( "f\ P" \)

shows \( "f \ (\text{map} \ (\text{transaction} \_\text{proj} \ l) \ P)" \)

and \( "f \ (\text{map} \ \text{transaction} \_\text{star} \_\text{proj} \ P)" \)

proof -

let \( ?g = \lambda T. \ \text{tl} \ (\text{transaction} \_\text{send} \ T)" \)

have \( \text{"set} \ (\text{\#g} \ (\text{transaction} \_\text{proj} \ l \ T)) \subseteq \text{set} \ (\text{\#g} \ T)" \) \( \text{(is \ "?A \subseteq \ ?C")}\)

and \( \text{"set} \ (\text{\#g} \ (\text{transaction} \_\text{star} \_\text{proj} \ T)) \subseteq \text{set} \ (\text{\#g} \ T)" \) \( \text{(is \ "?B \subseteq \ ?C")}\)

for \( T::\text{"('a,'b,'c,'d) prot\_\text{transaction}"} \)

proof -

obtain \( T1 \ T2 \ T3 \ T4 \ T5 \ T6 \) where \( T::\text{"T = transaction} \ T1 \ T2 \ T3 \ T4 \ T5 \ T6" \) by (cases \( T \) simp)

have \( \text{"\text{transaction} \_\text{send} \ (\text{transaction} \_\text{proj} \ l \ T) = \text{proj} \ l \ (\text{transaction} \_\text{send} \ T)"} \)

using \( \text{transaction} \_\text{proj}\_\text{simp}(\text{of} \ l \ T1 \ T2 \ T3 \ T4 \ T5 \ T6) \)

\( \text{transaction} \_\text{star} \_\text{proj}\_\text{simp}(\text{of} \ l \ T1 \ T2 \ T3 \ T4 \ T5 \ T6) \)

unfolding \( T \) \( \text{proj}\_\text{def} \ Let\_\text{def} \) by \( \text{auto} \)

hence \( \text{"set} \ (\text{\#g} \ (\text{transaction} \_\text{proj} \ l \ T)) \subseteq \text{set} \ (\text{\#g} \ l \ T)" \)

using \( \text{proj}\_\text{def} \) \( \text{by} \) \( \text{metis} \) \( \text{(no}\_\text{types}, \ \text{lifting}) \) \( \text{filter}\_\text{simp}(2) \) \( \text{list}\_\text{collapse} \) \( \text{list}\_\text{sel}(2,3) \)

thus \( "?A \subseteq \ ?C" \) \( "?B \subseteq \ ?C" \) using \( T \) \( \text{unfolding} \) \( \text{proj}\_\text{def} \) by \( \text{auto} \)
3.6 Stateful Protocol Verification

thus "f (map (transaction_proj l) P)" "f (map transaction_star_proj P)"
using P unfolding f_def list_all_iff by fastforce+

qed

private lemma reachable_constraints_no_leakage_alt'_aux2:
  fixes P
  defines "f ≡ λT.
  list_all is_Receive (unlabel (transaction_receive T)) ∧
  list_all is_Check_or_Assignment (unlabel (transaction_checks T)) ∧
  list_all is_Update (unlabel (transaction_updates T)) ∧
  list_all is_Send (unlabel (transaction_send T))"
assumes P: "list_all f P"
shows "list_all f (map (transaction_proj l) P)" (is ?A)
and "list_all f (map transaction_star_proj P)" (is ?B)
proof -
  have "f (transaction_proj l T)" (is ?A')
and "f (transaction_star_proj T)" (is ?B')
when T_in: "T ∈ set P" for T
proof -
  obtain T1 T2 T3 T4 T5 T6
  where T: "T = Transaction T1 T2 T3 T4 T5 T6"
  by (cases T)
  have "f T" using P T_in unfolding list_all_iff by simp
  thus ?A' ?B'
  unfolding f_def T unlabel_def proj_def Let_def list_all_iff
  transaction_proj.simps[of l T1 T2 T3 T4 T5 T6]
  transaction_star_proj.simps[of T1 T2 T3 T4 T5 T6]
  by auto
  qed
  thus ?A ?B unfolding list_all_iff by auto
qed

private lemma reachable_constraints_no_leakage_alt':
  fixes Sec P lbls k
  defines "no_leakage' ≡ λA. ∀I τ s. prefix A' A ∧ (∃l ts. suffix [(l, receive ⟨ts⟩)] A' s) ∧ s ∈ Sec - declassified lsst A I τ ∧ welltyped_constraint_model I τ (A'@[k, send ⟨[s]⟩])"
and "no_leakage' ≡ λA. ∀I τ s. (∃l ts. suffix [(l, receive ⟨ts⟩)] A) ∧ s ∈ Sec - declassified lsst A I τ ∧ welltyped_constraint_model I τ (A@[k, send ⟨[s]⟩])"
assumes P: "list_all wellformed_transaction P"
"list_all ((list_all (Not ◦ has_LabelS)) ◦ tl ◦ transaction_send) P"
and Sec: "∀s ∈ Sec. ¬{s} ⊆ Sec" and lbl: "∀l ∈ L. ∀A ∈ reachable_constraints (map (transaction_proj l) P). no_leakage' A"
says "∀l ∈ L. ∀A ∈ reachable_constraints (map (transaction_proj l) P). no_leakage A" (is ?A)
and "∀A ∈ reachable_constraints (map transaction_star_proj P). no_leakage A" (is ?B)
proof -
  define f where "f ≡ λT::('fun, atom, 'sets, 'lbl) prot_transaction.
  list_all is_Receive (unlabel (transaction_receive T)) ∧
  list_all is_Check_or_Assignment (unlabel (transaction_checks T)) ∧
  list_all is_Update (unlabel (transaction_updates T)) ∧
  list_all is_Send (unlabel (transaction_send T))"

  define g where "g::('fun, atom, 'sets, 'lbl) prot_transaction ⇒ bool ≡ list_all (Not ◦ has_LabelS) ◦ tl ◦ transaction_send"

  have P': "list_all f P"
  using P(1) unfolding wellformed_transaction_def f_def list_all_iff by fastforce

  note 0 = reachable_constraints_no_leakage_alt'_aux1[OF P(2), unfolded g_def[symmetric]]

  note 1 = reachable_constraints_no_leakage_alt'_aux2
  OF P'[unfolded f_def], unfolded f_def[symmetric]]
lemma composable_protocols_par_comp_prot_alt:

defines "f ≡ λA. {t ∈ M | prefix A (prefix [1, receive[ts]]) A'}" and "Sec ≡ (f (set S)) - {m. {} ⊢ m}" and "lbs ≡ λA. map (the_LabelN o fst) (filter (Not o has_LabelS) (transaction_strand T))" and "I ≡ set (remdupas (concat (map lbs (concat Ps))))" and "no_leakage ≡ λA. A ∈ reachable_constraints (map transaction_star_proj P) ∧ no_leakage' A" using reachable_constraints_par_no_leakage[OF Sec unfolding no_leakage'_def by blast

show ?A
proof (intro ballI)
  fix l A assume 1: "l ∈ L" and A: "A ∈ reachable_constraints (map (transaction_proj l) P)" show "no_leakage A" proof (rule ccontr)
  assume "¬ no_leakage A"
  then obtain I r A' s where A': "prefix A' A" "∃ l ts. suffix [l, receive[ts]] A'" "s ∈ Sec - declassified(A' I r)" "welltyped_constraint_model I r (A'@[(k, send[s])])"
  unfolding no_leakage_def by blast
  have s: "¬(¬l ⊢ s)" "fv s = {}" using A'(3) Sec unfolding welltyped_constraint_model_def by auto
  have I_r: "constr_sem_stateful I r (unlabel A'@[send[s]])" "wt[s] I r" "interpretation I r" "wt[s] (subst_range I r)" using A'(4) unfolding welltyped_constraint_model_def by auto
  show False using 2[OF 1(1) 0(1) s A A'(1,2) I_r(1)] 1 lb1 A'(3) I_r(2,3,4) singleton_list_proj(4)[of k "send[s]"]' unlabel_append[of _ "[(k, send[s])]"] unfolding no_leakage_def welltyped_constraint_model_def by metis
qed

show ?B
proof (intro ballI)
  fix A assume A: "A ∈ reachable_constraints (map transaction_star_proj P)" show "no_leakage A" proof (rule ccontr)
  assume "¬ no_leakage A"
  then obtain I r A' s where A': "prefix A' A" "∃ l ts. suffix [l, receive[ts]] A'" "s ∈ Sec - declassified(A' I r)" "welltyped_constraint_model I r (A'@[(k, send[s])])"
  unfolding no_leakage_def by blast
  have s: "¬(¬l ⊢ s)" "fv s = {}" using A'(3) Sec unfolding welltyped_constraint_model_def by auto
  have I_r: "constr_sem_stateful I r (unlabel A'@[send[s]])" "wt[s] I r" "interpretation I r" "wt[s] (subst_range I r)" using A'(4) unfolding welltyped_constraint_model_def by auto
  show False using 2[OF 1(2) 0(2) s A A'(1,2) I_r(1)] 3 A'(3) I_r(2,3,4) singleton_list_proj(4)[of k "send[s]"]' unlabel_append[of _ "[(k, send[s])]"] unfolding no_leakage_def welltyped_constraint_model_def by metis
qed

lemma composable_protocols_par_comp_prot_alt:

defines "f ≡ λA. {t ∈ M | prefix A (prefix [1, receive[ts]]) A'}" and "Sec ≡ (f (set S)) - {m. {} ⊢ m}" and "lbs ≡ λA. map (the_LabelN o fst) (filter (Not o has_LabelS) (transaction_strand T))" and "I ≡ set (remdupas (concat (map lbs (concat Ps))))" and "no_leakage ≡ λA. A ∈ reachable_constraints (map transaction_star_proj P) ∧ no_leakage' A" using reachable_constraints_par_no_leakage[unfolded f_def[symmetric] g_def[symmetric]] unfolding no_leakage_def by blast

show ?A
proof (intro ballI)
  fix l A assume 1: "l ∈ L" and A: "A ∈ reachable_constraints (map (transaction_proj l) P)" show "no_leakage A" proof (rule ccontr)
  assume "¬ no_leakage A"
  then obtain I_r A' s where A': "prefix A' A" "∃ l ts. suffix [l, receive[ts]] A'" "s ∈ Sec - declassified(A' I_r)" "welltyped_constraint_model I r (A'@[(k, send[s])])"
  unfolding no_leakage_def by blast
  have s: "¬(¬l ⊢ s)" "fv s = {}" using A'(3) Sec unfolding welltyped_constraint_model_def by auto
  have I_r: "constr_sem_stateful I r (unlabel A'@[send[s]])" "wt[s] I r" "interpretation I r" "wt[s] (subst_range I r)" using A'(4) unfolding welltyped_constraint_model_def by auto
  show False using 2[OF 1(1) 0(1) s A A'(1,2) I_r(1)] 1 lb1 A'(3) I_r(2,3,4) singleton_list_proj(4)[of k "send[s]"]' unlabel_append[of _ "[(k, send[s])]"] unfolding no_leakage_def welltyped_constraint_model_def by metis
qed

show ?B
proof (intro ballI)
  fix A assume A: "A ∈ reachable_constraints (map transaction_star_proj P)" show "no_leakage A" proof (rule ccontr)
  assume "¬ no_leakage A"
  then obtain I_r A' s where A': "prefix A' A" "∃ l ts. suffix [l, receive[ts]] A'" "s ∈ Sec - declassified(A' I_r)" "welltyped_constraint_model I r (A'@[(k, send[s])])"
  unfolding no_leakage_def by blast
  have s: "¬(¬l ⊢ s)" "fv s = {}" using A'(3) Sec unfolding welltyped_constraint_model_def by auto
  have I_r: "constr_sem_stateful I r (unlabel A'@[send[s]])" "wt[s] I r" "interpretation I r" "wt[s] (subst_range I r)" using A'(4) unfolding welltyped_constraint_model_def by auto
  show False using 2[OF 1(2) 0(2) s A A'(1,2) I_r(1)] 3 A'(3) I_r(2,3,4) singleton_list_proj(4)[of k "send[s]"]' unlabel_append[of _ "[(k, send[s])]"] unfolding no_leakage_def welltyped_constraint_model_def by metis
qed
assumes \( P_{pc} \): "wellformed_composable_protocols Ps N" "composable_protocols Ps Ms S" and component_secure:
\[ \forall A \in \text{reachable_constraints} (\text{map} (\text{transaction_proj} 1) (\text{concat} Ps)) \text{.} \exists I. \text{welltyped_constraint_model} I (A \emptyset [\{1, \text{send}([\text{attack} \{1, 1\}])\})]\nand no_leakage:
\[ \forall l \in L. \forall A \in \text{reachable_constraints} (\text{map} (\text{transaction_proj} 1) (\text{concat} Ps)) \text{.} \text{no_leakage} A \text{.} \]
shows "\( \forall A \in \text{reachable_constraints} (\text{concat} Ps) \text{.} \exists I. \text{constraint_model} I (A \emptyset [\{1, \text{send}([\text{attack} \{1, 1\}])\})]\n
proof

fix \( A \)
assume \( A : \ "A \in \text{reachable_constraints} (\text{concat} Ps)" \)
let \( ?\text{att} = [(1, \text{send}([\text{attack} \{1, 1\}])]) \)

define \( Q \) where "\( Q \equiv \lambda I_r. \text{interpretation}_{\text{sub}} I_r \land \text{wt}_{\text{sub}} I_r \land \text{wt}_{\text{rms}} (\text{sub}_{\text{range}} I_r) \)"

define \( R \) where "\( R \equiv \lambda A I_r. \exists A' 1 t. \text{prefix} A' A \land \text{suffix} [(1, \text{receive}(t))] A' \land \text{strand}_\text{leaks}_{\text{sub}} A' \text{ Sec} I_r \)"

define \( M \) where "\( M \equiv \bigcup T \in \text{set} (\text{concat} Ps) \text{.} \text{trms}_\text{transaction} T \cup \text{pair' Pair} \setminus \text{setops}_\text{transaction} T \)"
have Sec: "\( \forall s \in \text{Sec}. \neg(\cdot \vdash_c s" "\text{ground} \text{Sec} " \) unfolding Sec_def f_def by auto

have par_comp':
\[ \forall A \in \text{reachable_constraints} (\text{concat} Ps). \forall I_r. \text{constraint_model} I A \rightarrow (\exists I_r. \text{welltyped_constraint_model} I_r A \land 
(\forall n. \text{welltyped_constraint_model} I_r (\text{proj} n A) \lor R A I_r)) \]
using \( A \) composable_protocols_par_comp_constr[OF \( P_{pc} \)] unfolding Sec_def f_def by fast

have "\( \forall I_r. Q I_r \land \forall R A I_r \)"
using reachable_constraints_no_leakage_alt[OF Sec no_leakage[unfolded no_leakage_def L_def lbls_def]]

unfolding Q_def R_def by blast

hence no_leakage':
\[ \forall A \in \text{reachable_constraints} (\text{concat} Ps). \exists I_r. Q I_r \land \forall R A I_r \]
using reachable_constraints_component_leaks_if_composed_leaks[OF Sec, of "concat Ps"
\( \lambda I_r. \text{interpretation}_{\text{sub}} I_r \land \text{wt}_{\text{sub}} I_r \land \text{wt}_{\text{rms}} (\text{sub}_{\text{range}} I_r) \)]

unfolding Q_def R_def by blast

have M: "\( \text{has_all_wt_instances_of} \ \Gamma M (\text{set} N)" "\text{finite} (\text{set} N)" "\text{tfr}_{\text{set}} (\text{set} N)" "\text{trms}_{\text{set}} (\text{set} N)" and P: "\( \forall T \in \text{set} (\text{concat} Ps) . \text{wellformed_transaction} T \)"
\[ \forall T \in \text{set} (\text{concat} Ps) . \text{admissible_transaction_terms} T \]" 
\[ \forall T \in \text{set} (\text{concat} Ps) . \forall x \in \text{vars}_\text{transaction} T . \neg \text{Atom} \text{AttackType} \subseteq \Gamma_v x \]"" 
\[ \forall T \in \text{set} (\text{concat} Ps) . \forall s \in \text{set} (\text{transaction_strand} T) . 
\text{is}_\text{Send} (s) \land \text{length} (\text{the} \text{msgs} (s)) = 1 \land 
\text{is}_\text{Fun} \text{Attack} (hd (\text{the} \text{msgs} (s))) 
\rightarrow 
\text{the} \text{Attack_label} (hd (\text{the} \text{msgs} (s))) = \text{fst} s \]"" 
\[ \forall T \in \text{set} (\text{concat} Ps) . \text{list}_\text{all} \text{tfr}_{\text{step}} (\text{unlabel} (\text{transaction_strand} T)) \]"using Ps_pc(1) tfr_set_if_comp_tfr_set tfr_setp_is_comp_tfr_setp

unfolding wellformed_composable_protocols_def list_all_iff Let_def M_def
\( \text{trms}_{
\text{set}} \) _def wfrms_code unlabel_def \( \Gamma_v \text{Atom}' '(1, 2) \) by (force, force, fast, fast, fast, fast, fast, simp, simp)

have P_fresh: "\( \forall T \in \text{set} (\text{concat} Ps) . \forall x \in \text{set} (\text{transaction_fresh} T) . 
\Gamma_v x = \text{Atom}_v \text{Value} \lor (\exists a. \Gamma_v x = \text{Atom}_v (\text{Atom} a)) \)" (is "\( \forall T \in \text{set} (\text{concat} Ps) . \forall x \in \text{set} (\text{transaction_fresh} T) . 
(\exists a. \Gamma_v x = \text{Atom}_v (\text{Atom} a)) \)"

proof

(intro ballI) 
fix \( T \) assume \( T : \ "T \in \text{set} (\text{concat} Ps)" \)

hence "\( \Gamma_v x = \text{Atom}_v \text{Value} \lor (\exists a. \Gamma_v x = \text{Atom}_v (\text{Atom} a)) \)" using Ps_pc(1) unfolding wellformed_composable_protocols_def list_all_iff Let_def by fastforce

thus "\( \forall Q x \) by (metis prot_atom.is_Atom_def term.collapse(1))

qed

have P': "\( \forall T \in \text{set} (\text{concat} Ps) . \text{trms}_{\text{transaction} T} \)"
using P(2) admissible_transaction_terms_def by fast

have "¬welltyped_constraint_model I (A?att)" for I
  proof
    assume "welltyped_constraint_model I (A?att)"
    hence I: "welltyped_constraint_model I A" "ik∣A ∣set I ⊢ attack(ln 1)"
      using strand_sem_append_stateful[of "\{\}" "\{\}" "unlabel A" "unlabel ?att"]
    unfolding welltyped_constraint_model_def constraint_model_def by auto

obtain I', where I': "welltyped_constraint_model I' A"
  unfolding Q_def welltyped_constraint_model_def constraint_model_def by metis

have "\{1, receive\{attack\{ln 1\}\}\} ∈ set A"
  unfolding reachable_constraints_receive_attack_if_attack[of M P(1-2) P_fresh P(3) I P(4)]
  by auto

hence "\{ik\}_set (proj 1 A) ∣set I' ⊢ attack(ln 1)"
  using in_proj_set[of 1 "receive\{attack\{ln 1\}\}\ A"] in_ik_iff[of "attack\{ln 1\}" proj 1 A]
  intruder_deduct.Axiom[of "attack\{ln 1\}" "ik\set (proj 1 A) ∣set I'"]
  by fastforce

hence "welltyped_constraint_model I' (proj 1 A?att)"
  using I', strand_sem_append_stateful[of "\{\}" "\{\}" "unlabel (proj 1 A)" "unlabel ?att" I']
  unfolding welltyped_constraint_model_def constraint_model_def by auto

thus False
  using component_secure reachable_constraints_transaction.proj[of M, of I] by simp

qed

lemma composable_protocols_par_comp_prot:
  fixes S I Sec lbsa Ps
  defines "f ≡ λδ. (t · δ) ∈ t · δ. t ∈ M ∧ wt subst δ ∧ wt tran (subst_range δ) ∧ fv (t · δ) = {}"
  and "Sec ≡ ((f (set S)) - {n. n |∈ |m})" and "lbsa ≡ λA. map (the_LabelN o fst) (filter (Not o has_LabelS) (transaction_strand T))"
  and "L ≡ (set (redups (concat (map lbsa (concat Ps)))))"
  and "no_leakage ≡ λA. ∀I. ∀s. (\{1 ts. suffix\{\{1, receive\{ts\}\}\}\} ∧ s ∈ Sec - declassified; A I ∨ welltyped_constraint_model I. (AΘ(\{, send\{s\}\}))\)
  assumes Ps_pc: "wellformed_composable_protocols Ps N" "composable_protocols Ps Ms S"
  "list_all ((\{list_all (Not o has_LabelS)\} o tl o transaction_send) (concat Ps))"
  and component_secure:
    "∀A ∈ reachable_constraints (map (transaction_proj 1) (concat Ps)). \exists I. welltyped_constraint_model I (AΘ[\{1, send\{\{\}\}\}])"
  and no_leakage:
    "∀I ∈ L. ∀A ∈ reachable_constraints (map (transaction_proj 1) (concat Ps)). no_leakage A"
  shows "∀A ∈ reachable_constraints (concat Ps). \exists I. constraint_model I (AΘ[\{1, send\{\{\}\}\}])"
  proof
  have P': "\exists list_all wellformed_transaction (concat Ps)"
    using Ps_pc(l) unfolding wellformed_composable_protocols_def by meson

have Sec: "∀s ∈ Sec. ¬\{\} ⊢ s" "ground Sec" unfolding Sec_def f_def by auto

note 0 = composable_protocols_par_comp_prot_alt[OF Ps_pc(l,2) component_secure, unfolded lbsa_def[symmetric] L_def[symmetric]]

note 1 = reachable_constraints_no_leakage_alt'[OF P' Ps_pc(3) Sec no_leakage[unfolded no_leakage_def]]
show ?thesis using 0 1 unfolding t_def Sec_def by argo
qed

lemma composable_protocols_par_comp_prot':
  assumes
  "Pc = concat Ps"
  "set Ps_with_stars = (λn. map (transaction_proj n) Pc) ` set (remdups (concat (map (λT. map (the_LabelN ◦ fst) (filter (Not ◦ has_LabelS) (transaction_strand T))) Pc)))"
  and
  "list_all (list_all (Not ◦ has_LabelS) ◦ tl ◦ transaction_send) Pc"
  "wellformed_composable_protocols Ps N"
  "composable_protocols Ps Ms S"
  and
  "list_all (welltyped_leakage_free_protocol S) Ps_with_stars"
  and
  "P = map (transaction_proj n) Pc"
  and
  "∀A ∈ reachable_constraints P. ∄I. welltyped_constraint_model I (A@[⟨n, send ⟨attc ⟨ln n⟩⟩⟩])"
shows
"∀A ∈ reachable_constraints Pc. ∄I. welltyped_constraint_model I (A@[⟨n, send ⟨attc ⟨ln n⟩⟩⟩])"
by (rule composable_protocols_par_comp_prot[OF Ps_wellformed(2,3,1)[unfolded P_defs(1)]
P_wt_secure[unfolded P_def[unfolded P_defs(1)]
transaction_proj_ball_subst[OF P_defs(2)[unfolded P_defs(1)]
Ps_no_leakage(1)[unfolded list_all_iff welltyped_leakage_free_protocol_def Let_def]],
unfolded P_defs(1)[symmetric])
end

context
begin

lemma welltyped_constraint_model_leakage_model_swap:
  fixes
  I α δ ::"('fun,'atom,'sets,'lbl) prot_subst" and
  s
  assumes
  A: "welltyped_constraint_model I (A@[⟨⋆, send ⟨s · δ⟩⟩])"
  and
  α: "transaction_renaming_subst α A"
  and
  δ: "wt subst δ" "wf trms (subst_range δ)" "subst_domain δ = fv s" "ground (subst_range δ)"
obtains J
where
  "welltyped_constraint_model J (A@[⟨⋆, send ⟨s · δ⟩⟩])"
  and
  "ik lsst A · set J ⊢ s · α · J"
proof
  note defs = welltyped_constraint_model_def construct_model_def
  note δ_s = subst_fv_dom_ground_if_ground_img[OF equalityD2[OF δ(3)] δ(4)]
  note α' = transaction_renaming_subst_is_renaming(2)[OF α]
inj_on_subset[OF transaction_renaming_subst_is_injective[OF α]
  subset_UNIV[OF "fv s"]]
  transaction_renaming_subst_var_obtain(2)[OF α, of _ s]
  transaction_renaming_subst_is_renaming(6)[OF α, of s]
  transaction_renaming_subst_vars_disj(8)[OF α]
  transaction_renaming_subst_ut[OF α]
define αinv where "αinv ≡ subst_var_inv α (fv s)"
define δ' where "δ' ≡ rm_vars (UNIV - fv (s · α)) (αinv o α)"
define J where "J ≡ λx. if x ∈ fv (s · α) then δ' x else I x"

have α_invertible: "s = s · α o αinv"
  using α'(1) inj_var_ran_subst_is_invertible'[of α s] inj_on_subset[OF α'(2)]
unfolding ainv_def by blast

have δ'_domain: "subst_domain δ' = fv (s · α)"
  proof -
    have "x ∈ subst_domain (ainv oα δ)" when x: "x ∈ fv (s · α)" for x
      proof -
        obtain y where y: "y ∈ fv s" "α y = Var x"
          using α'(3)[OF x] by blast

        have "y ∈ subst_domain δ" using y(1) δ(3) by blast
        moreover have "(α(α δ) x = δ y"
          using α'(3)[OF x] α_invachable
        vars_term_subset_subst_eq[of "Var y" s "α oα αinv Var"
        unfolding δ'_def ainv_def
        by (metis (no_types, lifting) fv_subst_subset subst_apply_term.simps(1)
         subst_apply_term_empty subst_compose)

      ultimatey show δ(4) by fastforce
    qed

  thus ?thesis using rm_var_dom[of "UNIV - fv (s · α)" "α oα αinv"] unfolding δ'_def by blast

have δ'_range: "fv t = {}" when t: "t ∈ (subst_range δ')" for t
  proof -
    obtain x where "x ∈ fv (s · α)" "δ' x = t" using t δ'_domain by auto
    thus ?thesis
      by (metis (no_types, lifting) δ'_def subst_compose_def δ(3,4) α_invachable fv_subst_subset
         subst_fv_dom_ground_if_ground_img subst_compose Diff_iff)

have J0: "x ∈ fv (s · α) ==> J x = δ' x"
  "x ∉ fv (s · α) ==> J x = I x" for x
  unfolding J_def by (cases "x ∈ fv (s · α)") (simp_all add: subst_compose)

have J1: "subst_range J ⊆ subst_range δ' ∪ subst_range I"
  proof
    fix t assume "t ∈ subst_range J"
    then obtain x where x: "x ∈ subst_domain J" "J x = t" by auto
    hence "t = δ' x ==> x ∈ subst_domain δ'" "t = I x ==> x ∈ subst_domain I"
    by (metis subst_domI subst_dom_vars_in_subst)
    thus "t ∈ subst_range δ' ∪ subst_range I" using x(2) J0[of x] by auto

have "x ∉ fv (s · α)" when x: "x ∈ fvistr (AΘ[(s, send([s · δ]))])" for x
  using x δ s α(4) α'(5) by auto
  hence "I x = J x" when x: "x ∈ fvistr (AΘ[(s, send([s · δ]))])" for x
    using x unfolding J_def δ'_def by auto
  hence "constr_sem_stateful J (unlabel (AΘ[(s, send([s · δ]))]))"
    using A strand_sem_model_swap[of "unlabel (AΘ[(s, send([s · δ]))])"] I J "{t}" "{}"
  unfolding defs by blast

  moreover have "wt subst J"
    using A subst_var_inv_wt[OF α'(6), of "fv s"]
    wt_subst_trm[of δ(1)] subst_compose[of "subst_var_inv α (fv s)" δ]
  unfolding defs J_def δ'_def ainv_def wt subst_def by presburger

  moreover have "interpretation subst J"
    proof -
      have "fv t = {}" when t: "t ∈ (subst_range J)" for t
        using t A J1 δ'_range unfolding defs by auto
      moreover have "x ∈ subst_domain J" for x
        proof (cases "x ∈ fv (s · α)")
          case True thus ?thesis using J0(1)[of x] δ'_domain unfolding subst_domain_def by auto
          next
          case False
          have "subst_domain I = UNIV" using A unfolding defs by fast
thus \( \text{?thesis using } \text{J0(2)[OF False]} \) unfolding subst_domain_def by auto

ultimately show \( \text{?thesis by auto} \)

qed

moreover have \( "wfrms, (subst_range \delta')" \)

using wfrms_subst-compose[OF subst_var_inv_wf_trms[of \( \alpha "fv s" \) \( \delta(2) \)]

unfolding \( \delta'_\text{def} \) \( \alpha_{\text{inv_def by force}} \)

hence \( "wfrms, (subst_range J)" \) using A J1 unfolding defs by fast

ultimately show \( "\text{welltyped_constraint_model J} \ (A\&[(s, send([s \cdot \delta]))])" \)

unfolding defs by blast

hence \( "ik_{sct} A \cdot s_{ct} J \vdash s \cdot \delta" \)

using \( \delta_s \) strand_sem_append_stateful[of \( "() " "()" "unlabel A " [send([s \cdot \delta])] \) J]

unfolding defs by (simp add: subst_ground_ident)

moreover have \( "s \cdot \alpha \cdot J = s \cdot \delta" \)

proof -

have \( "J x = \delta' \ x" \) when \( x: "x \in fv (s \cdot \alpha)" \) for \( x \) using \( x \) unfolding \( J \) def by argo

hence \( "s \cdot \alpha \cdot J = s \cdot \alpha \cdot \delta'" \) using subst_agreement[of \( "s \cdot \alpha" \) \( J \) \( \delta' \)] by force

thus \( \text{?thesis} \)

using \( \alpha_{\text{invertible unfolding \( \delta'_\text{def} \)}} \)

by (metis rm_vars_subst_eq' subst_subst_compose)

hence \( "s \cdot \alpha \cdot J = s \cdot \delta" \) by auto

ultimately show \( "ik_{sct} A \cdot s_{ct} J \vdash s \cdot \alpha \cdot J" \) by argo

qed

lemma welltyped_leakage_free_protocol_pointwise:

"welltyped_leakage_free_protocol S P \iff list_all (\( \lambda s. \) welltyped_leakage_free_protocol \( s \) P) S"

unfolding welltyped_leakage_free_protocol_def list_all_iff Let_def by fastforce

lemma welltyped_leakage_free_no_deduct_constI:

fixes \( c \)

defines \( s \equiv \text{Fun c } []::('\text{fun},'atom,'sets,'lbl) \text{prot_term} \)

assumes \( s: "\forall A \in \text{reachable_constraints P}. \forall L. \text{welltyped_constraint_model I, } (A\&[(s, send([s]))])" \)

shows \( \text{welltyped_leakage_free_constraint \( s \) P} \)

using s unfolding welltyped_leakage_free_protocol_def s_def by auto

lemma welltyped_leakage_free_pub_termI:

assumes \( \{s\} \vdash s \) P

shows \( \text{welltyped_leakage_free_protocol \( s \) P} \)

proof -

define \( f \) where \( "\forall t \cdot t \in M \land wt_{\text{subt}} \delta \land wfrms \ (subst_range \delta) \land fv (t \cdot \delta) = \{\}" \)

define \( \text{Sec} \) where \( \"\text{Sec} \equiv f\text{ (set} [s]) \) - \( \{s. \{\} \vdash c\} \)"

have 0: \( "fv s = \{\}" \) using s pgwt_ground pgwt_is_empty_synth by blast

have 1: \( "s \cdot \delta = s" \) for \( \delta \) by (rule subst_ground_ident[OF 0])

have 2: \( "wtrms, (subst_range Var)" \)

using wt_subt_Var wfrms_subst_range_Var by (blast,blast)

have \( "f \text{ (set} [s]) = \{s\}" \)

proof

show \( "f \text{ (set} [s]) \subseteq \{s\}" \) using 0 1 unfolding f_def Q_def by auto

have \( "Q \ {s} \ Var" \) using 0 2 unfolding Q_def by auto

thus \( "\{s\} \subseteq f \text{ (set} [s])" \) using 1[of Var] unfolding f_def by force

hence \( "\text{Sec} \ = \{\}" \) using s unfolding Sec_def by simp

thus \( \text{?thesis unfolding welltyped_leakage_free_protocol_def \( \text{Let_def Sec_def f_def Q_def by blast} \) \) \)

qed

lemma welltyped_leakage_free_pub_constI:

assumes \( c: \"\text{"public}, c \"\text{"arity}, c = 0" \)

shows \( \text{welltyped_leakage_free_protocol} \ (\{c\}) P" \)

using c welltyped_leakage_free_pub_term[OF intruder_synch.ComposeC[of \( "[]" "\text{Fu c}" (\{\})\) by simp

271
proof (rule ccontr)

lemma welltyped_leakage_free_long_term_secretI:
fixes n
defines "Tatt ≡ λs'. Transaction (λ(). [ ] [] [(n, receive([s']))] [] [(n, send([attack(in n)])])]
assumes P_wt_secure:
"∀ A ∈ reachable_constraints P. A I
welltyped_constraint_model I (A@[(n, send([attack(in n)])])]
and s_long_term_secret:
"∃ δ, wt_subst δ' ∧ inj_on δ (fv s) ∧ δ′ : fv s ⊆ range Var ∧ Tatt (s · δ) ∈ set P"
shows "welltyped_leakage_free_protocol [s] P"
proof (rule ccontr)

obtain δ where "Q ≡ λM t δ. t ∈ M ∧ wt_subst δ ∧ wt_trms (sub_range δ) ∧ fv (t · δ) = {}"

define f where "f ≡ λM. t · δ | t δ. Q t δ"

define Sec where "Sec ≡ f (set [s]) - {m. {a} ⊆ m}"

note def = Sec_def f_def Q_def
note def' = welltyped_constraint_model_def constraint_model_def

assume "¬ welltyped_leakage_free_protocol [s] P"
then obtain A I s' where A:
"A ∈ reachable_constraints P" "s' ∈ Sec - declassified at A I"
"welltyped_constraint_model I (A@[s'])"

unfolding welltyped_leakage_free_protocol_def defs by fastforce

obtain ϑ where "(wt_subst δ) ϑ ∧ (fv s) ϑ ∧ inj_on ϑ (fv s)" "Tatt (s · ϑ) ∈ set P"
using s_long_term_secret by blast

obtain δ where δ:
"wt_subst δ" "wt_trms (sub_range δ)" "sub_domain δ = fv (s · ϑ)" "ground (sub_range δ)"
"s' = s · ϑ · δ'

proof -
obtain δ where "*: wt_subst δ" "wt_trms (sub_range δ)" "fv s' = {}" "s' = s · δ"
using A(2) unfolding defs by auto

define σ where "σ ≡ subst_var_inv ϑ (fv s) o ϑ δ""σ'
define δ' where "δ' ≡ rm_vars (UNIV - fv (s · ϑ)) σ"

have **: "s' = s · ϑ · σ"
using *(4) inj_var_range_subst_is_invertible[OF ϑ(3,2)]
unfolding σ_def by simp

have "s' = s · ϑ · δ'"
using ** rm_vars_subst_eq'[of s · ϑ σ]
unfolding δ'_def by simp
moreover have "wt_subst σ"
using ϑ(1) *(1) subst_var_inv wt wt_subst_compose
unfolding σ_def by premsugar
hence "wt_subst δ'" using wt_subst_rmvars unfolding δ'_def by blast
moreover have "wt_trms (sub_range σ)"
using wt_trms_subst_compose[OF subst_var_inv_wf_trms *(2)] unfolding σ_def by blast
hence "wt_trms (sub_range δ')" using wt_trms_subst_rmvars'[of σ] unfolding δ'_def by blast
moreover have "fv (s · ϑ) ⊆ sub_domain σ"
using *(3) ** ground_term_subst_domain fv_subst unfolding σ_def by blast
hence "sub_domain δ' = fv (s · ϑ)"
using rm_vars_dom[of "UNIV - fv (s · ϑ)" σ] unfolding δ'_def by blast
moreover have "ground (sub_range δ')"
proof -
{ fix t assume "t ∈ sub_range δ'"
then obtain x where "x ∈ fv (s · ϑ)" "δ' x = t"
using <sub_domain δ' = fv (s · ϑ)> by auto
hence "t ⊆ s · ϑ · δ'" by (meson subst_mono_fv)
hence "fv t = {}" using <s' = s · ϑ · δ'> *(3) ground_subterm by blast

272
3.6 Stateful Protocol Verification

thus thesis by force

ultimately show thesis using that[of δ'] by fast

have ξ: "transaction_decl_subst Var (Tatt t)"
and σ: "transaction_fresh_subst Var (Tatt t) A"
for t
unfolding transaction_decl_subst_def transaction_fresh_subst_def Tatt_def
by simp_all

obtain α::"(‘fun,’atom,’sets,’lbl) prot_subst"
where α: "transaction_renaming_subst α P A"
unfolding transaction_renaming_subst_def
by blast

obtain J where J: "welltyped_constraint_model J (A@[⟨⋆, send[⟨s · ϑ · δ⟩⟩⟩])"
using welltyped_constraint_model_leakage_model_swap[OF A(3)[unfolded δ(5)] α δ](1-4)
by blast

define T where T = dual lsst (transaction_strand (Tatt (s · ϑ)) · lsst α)

define B where B ≡ A@T

have "transaction_receive (Tatt t) = [⟨n, receive[⟨t⟩]⟩]"
"transaction_checks (Tatt t) = []"
"transaction_updates (Tatt t) = []"
"transaction_send (Tatt t) = [⟨n, send[⟨attack⟨ln n⟩⟩]⟩]"
for t
unfolding Tatt_def
by simp_all

hence T_def': "T = [⟨n, send[⟨s · ϑ · α⟩]⟩, ⟨n, receive[⟨attack⟨ln n⟩⟩]⟩]"
using subst_lsst_append[of "transaction_receive (Tatt (s · ϑ))" _ α]
subst_lsst_singleton[of "ln n" "receive[⟨s · ϑ⟩]" α]
subst_lsst_singleton[of "ln n" "send[⟨attack⟨ln n⟩⟩]" α]
unfolding transaction_strand_def T_def by fastforce

have B0: "ik lsst B · set J ⊢ attack⟨ln n⟩"
using in_ik_sst_iff[of "attack⟨ln n⟩" "unlabel T"]
unfolding B_def T_def' by (force intro!: intruder_deduct.Axiom)

have B1: "B ∈ reachable_constraints P"
using reachable_constraints.step[OF A(1) θ(4) ξ σ]
unfolding B_def T_def by simp

have "welltyped_constraint_model J B"
using J strand_sem_append_stateful[of "{}" "{}" "unlabel A" _ J]
unfolding defs' B_def T_def' by fastforce

hence B2: "welltyped_constraint_model J (B@[⟨n, send[⟨attack⟨ln n⟩⟩]⟩])"
using B0 strand_sem_append_stateful[of "{}" "{}" "unlabel B" "[send[⟨attack⟨ln n⟩⟩]]" J]
unfolding defs' B_def by auto

show False using P_wt_secure B1 B2 by blast

qed

lemma welltyped_leakage_free_value_constI:
assumes P:
"∀ T ∈ set P. wellformed_transaction T"
"∀ T ∈ set P. admissible_transaction_terms T"
"∀ T ∈ set P. transaction_decl T () = []"
"∀ T ∈ set P. bvars_transaction T = {}"
and P_fresh_declass:
"∀ T ∈ set P. transaction_fresh T ≠ [] → 
(transaction_send T ≠ [] ∧ (let (l,a) = hd (transaction_send T)
in l = ⋆ ∧ is_Send a ∧ Var ` set (transaction_fresh T) ⊆ set (the_msgs a)))"
shows "welltyped_leakage_free_protocol [⟨m: value⟩] P"
proof (rule ccontr)
define Q where "Q ≡ λM t δ. t ∈ M ∧ wtsubst δ ∧ wterms (subst_range δ) ∧ fv (t · δ) = {}"
define f where "f ≡ λM. {t · δ | t ∈ M. Q M t}" 
define Sec where "Sec ≡ f (set ([m: value]v)) - {m. {} ⊢ c m}"

note defs = Sec_def f_def Q_def
note defs' = welltyped_constraint_model_def constraint_model_def

assume "¬ welltyped_leakage_free_protocol ([m: value]v) P"
then obtain A I s where A:
  "A ∈ reachable_constraints P" "s ∈ Sec - declassified lsst A I" 
  "welltyped_constraint_model I (A@[⟨⋆, send⟩])" 
  unfolding welltyped_leakage_free_protocol_def defs by fastforce
have "ik lsst (A · lsst I) ⊑ set (trms lsst A)" 
  using constraint_model_Val_const_in_constr_prefix[of A(1) I0 P(1,2)]
  unfolding sn by presburger

obtain B T ξ σ α where B:
  "prefix (B@dual lsst (transaction_strand T · lsst ξ o σ o α)) A" 
  "B ∈ reachable_constraints P" "T ∈ set P" "transaction_fresh_subst ξ T" 
  "transaction_renaming_subst α P B" 
  "s ∈ subst_range σ" 
  using constraint_model_Value_in_constr_prefix_fresh_action[of A(1) P(2-) s4[unfolded sn]] sn
  by blast

obtain Tts Tsnds sx where T:
  "transaction_send T = ⟨⋆, send(Tts)⟩#Tsnds" "Var ` set (transaction_fresh T) ⊆ set Tts" 
  and sx: "Var sx ∈ set Tts" "σ sx = s" 
  using P_fresh_declass B(3,5,7)
  unfolding transaction_fresh_subst_def is_Send_def 
  by (cases "transaction_send T") (fastforce,fastforce)

have ξ_elim: "ξ o σ o α = σ o α" 
  using admissible_transaction_decl_subst_empty[of bspec[OF P(3) B(3)] B(4)]
  by simp
lemma welltyped_leakage_free_priv_constI:

fixes c

defines "s ≡ Fun c [] :: ('fun, 'atom, 'sets, 'lbl) prot_term"

assumes c: "¬ public c" "arity c = 0" "Γ s = TAtom ca" "ca ≠ Value"

and P: "∀ T ∈ set P. ∀ x ∈ vars_transaction T. is_Var (Γ_v x)"
    "∀ T ∈ set P. ∀ t ∈ vars_transaction T. is_Var (Γ_v t)"
    "∀ T ∈ set P. ∀ t ∈ trms transaction_send T. s /∈ set (snd (Ana t))"
    "∀ T ∈ set P. s /∈ trms transaction_send T"
    "∀ T ∈ set P. ∀ t ∈ set (transaction_fresh T). Γ_v x = Γ_v t" "∀ T ∈ set P. wellformed_transaction T"

shows "∀ A ∈ reachable_constraints P. ∃ I. welltyped_constraint_model I A (A@[(*, receive(ts)])"
    (is "∀ A ∈ ?R. ?P A")
    and "welltyped_leakage_free_protocol [s] P"

proof -
show "¬P A" proof

let ?P_s_cases = "∀ T ∈ set P. s /∈ (M ∪ (∃ m ∈ subterms transaction_send T. s /∈ set (snd (Ana m))))"

let ?P_s_cases' = "∀ T ∈ set P. s /∈ (M ∪ (∃ m ∈ subterms transaction_send T. s /∈ set (snd (Ana m))))"

note defs = Sec_def f_def Q_def note defs' = Sec_def f_def Q_def constraint_model_def

have s5: "s ∈ set (Tts 1_set ξ o_ o_ α _ o_ I)"
using sx unfolding ξ_elim sn by force

have s6: "(*, receive(Tts 1_set ξ o_ o_ α _ o_ I)) ∈ set (A 1_set I)"
proof -
  have "(*, send(Tts)) ∈ set (transaction_send T)"
  using T(1) by simp
  hence "(*, send(Tts 1_set δ)) ∈ set (transaction_send T 1_set δ)" for δ
  unfolding subst_apply_labeled_stateful_strand_def by force
  hence "(*, receive(Tts 1_set δ)) ∈ set (transaction_strand T 1_set δ)" for δ
  using transaction_strand_subset_subsets(4)[of T δ] by fast
  hence "(*, receive(Tts 1_set δ)) ∈ set (dual 1_set (transaction_strand T 1_set δ))" for δ
  using dual1_set_steps_iff(1)[of "Tts 1_set δ"] by blast

have "(*, receive(Tts 1_set ξ o_ o_ α _ o_ α)) ∈ set A"
using B(1) *[of "ξ o_ o_ α"] unfolding prefix_def by force
thus ?thesis
qed

unfolding subst_apply_labeled_stateful_strand_def by force

qed

lemma welltyped_leakage_free_priv_constI:

proof

show False
using s6 f(4) ideduct_mono[OF Axiom[OF s5], of "⋃{set ts|ts. (*, receive(ts)) ∈ set (A 1_set I)}"]
unfolding declassified1_set_def by blast

qed

3.6 Stateful Protocol Verification
276

3 Stateful Protocol Verification

"A = g'  Ts"  "\forall B.  prefix B Ts \rightarrow g'  B \in reachable_constraints P"

"\forall B T \xi \sigma A.  prefix (BG[(T,\xi,\sigma,\alpha)]) Ts \rightarrow
T \in set P \land transaction_decl_subst \xi T \land
transaction_fresh_subst \sigma T (g'  B) \land transaction_renaming_subst \alpha P (g'  B)"
using reachable_constraints_as_transaction_lists[OF A(1)] unfolding g'_def f'_def by blast

have "\text{\textit{ik}}_{\text{set}} A \cdot \text{set} I \vdash s \cdot I" and I s: "s \cdot I = s"
using welltyped_constraint_model_deduct_split[OF I]
unfolding s_def by simp_all

hence s_deduct: "\text{\textit{ik}}_{\text{set}} (A \cdot \text{set} I) \vdash s" "\text{\textit{ik}}_{\text{set}} A \cdot \text{set} I \vdash s"
by (metis \text{\textit{ik}}_{\text{set}}_subst unlabel_subst, metis)

have I wt: "\text{\textit{wt}}_{\text{subst}} I"
and I wf: "\text{\textit{wf}}_{\text{terms}} (\text{\textit{subst_range}} I)"
and I grounds: "\text{ground (\text{\textit{subst_range}} I)}"
and I interp: "\text{interpretation}_{\text{subst}} I"
using I unfolding defeq by (blast,blast,blast,blast)

have Sec unfold: "Sec = \{a\}"
proof
have "\neg \{a\} \vdash c s" using ideeduct_synth_prv_const_in_ik[OF _ c(1)] unfolding s_def by blast
thus "\{s\} \subseteq Sec" unfolding defeq by fastforce
qed (auto simp add: defeq s_def)

have s2: "\text{\textit{wf}}_{\text{terms}} s"
using c(1,2)
unfolding s_def by fastforce

have A ik fv: "\exists a. \Gamma_v x = TAtom a \land a \neq ca" when x: "x \in fv_set (ik_{\text{set}} A)"
for x
proof -
  obtain T where T: "T \in set P"  "\Gamma_v x \in \Gamma_v \setminus \text{\textit{fv}}_{\text{transaction}} T"
using fv ik set is fv set [OF x] reachable_constraints_var_types_in_transactions(1)[OF A P(S)]
by fast
  then obtain y where y: "y \in \text{\textit{vars}}_{\text{transaction}} T"  "\Gamma_v y = \Gamma_v x"
using vars set is fv set bvars set [OF subst_range T] by fastforce
  then obtain a where a: "\Gamma_v y = TAtom a" using P(1) T(1) by blast
  hence \\begin{itemize}
  \item \(\Gamma_v x = TAtom a\)
  \item \(\Gamma s \neq \Gamma_v x\)
  \item \(\Gamma s = TAtom ca\)
\end{itemize}
using y P(2) T(1) c(3) by auto
thus \text{\textit{thesis by force}}
qed

have I s x: "\neg s \subseteq I x" when x: "x \in fv_set (ik_{\text{set}} A)"
for x
proof -
  obtain a where a: "T v x = TAtom a" "a \neq ca" using A ik fv[OF x]
by blast
  hence a': "\Gamma (I x) = TAtom a" using wt subst_trm'[OF I wt, of "Var x"]
by simp
obtain f ts where f: "I x = Fun f ts"
by (meson empty fv exists fun interpretation grounds all[OF I interp])
  hence ts: "ts = []"
using I wf const type inv wf[OF a'[unfolded f]] by fastforce

have "c \neq f" using f[unfolded ts] a a' c(3)[unfolded s_def] by force
thus \text{\textit{thesis using f ts unfolding s_def by simp}}
qed

have A ik I const: "\exists f. \text{\textit{arity}} f = 0 \land I x = \text{Fun} f []" when x: "x \in fv_set (ik_{\text{set}} A)"
for x
using x A ik fv I wt empty fv exists fun[INTERpretation grounds all[OF I interp, of x]]
wf trm subst range D[IF I wf, of x] const type inv const type inv wf
by (metis (no types, lifting) \Gamma.simps(1) wt subst def)

hence A ik subst: "\text{\textit{subterms}}_{\text{set}} (ik_{\text{set}} A \cdot \text{set} I) = \text{\textit{subterms}}_{\text{set}} (ik_{\text{set}} A) \cdot \text{set} I"
using subterms subst'[OF "ik_{\text{set}} A"] by blast

have substlmm1: "s \in set (\Gamma \text{\textit{and}} (Ana m))"
when m: "m \subseteq set N" "s \in set (\Gamma \text{\textit{and}} (Ana m \cdot \delta))"
and N: \begin{itemize}
\item \(\forall y. y \in fv_set N \implies \neg s \subseteq \delta y\)
\end{itemize}


3.6 Stateful Protocol Verification

for \( m \) \( M \) \( \delta \)

proof -

have \( \text{m_fun}: \text{is_Fun} \ m \)
using \( m \) \( M \) Ana_subterm' vars_iff_subtermq_set

unfolding s_def is_Var_def by (metis subst_apply_term.simps(1))

obtain \( f K R ts i \) where:
\( m = \langle f \ ts \rangle \) \( \text{arity} \ f = \text{length} ts \) \( \text{arity} \ f > 0 \) \( \text{Ana} \ f = (K, R) \)
and \( i : \text{i < length} R \) \( s = ts ! (R ! i) \) \( \delta \)

and \( R_i : \forall i < \text{length} R. \text{map} (! ts) R ! i = ts ! (R ! i) \)

proof -

obtain \( f ts K R \) where:
\( m = \langle f \ ts \rangle \) \( \text{arity} \ f = \text{length} ts \) \( \text{arity} \ f > 0 \) \( \text{Ana} \ f = (K, R) \) \( \text{Ana} (m \cdot \delta) = (K \cdot \text{list} (! ts), \text{map} (! ts) R) \)

using \( m(2) \) Ana_nonempty_inv[of \( m \cdot \delta \)] by force

obtain \( ts' \) where:
\( m' = \langle f \ ts' \rangle \) \( ts = ts' \cdot \text{list} \delta \)

using \( f(1) \) m_fun by auto

have \( R_i : \text{arity} \ f = \text{length} ts' \) \( \text{length} ts = \text{length} ts' \)

when \( i : \text{i < length} R \) for \( i \)

thus \( ?\text{thesis} \) using \( i M \) \( \text{Ana} \_\text{asm} \_\text{alt} \) \( f(1) \) \( f(2) \) \( \text{simp} \_\text{all} \)

obtain \( i \) where:
\( s = ts ! (R ! i) \) \( \text{i < length} R \)

by (metis (no_types, lifting) \( m(2) \) \( f(5) \) in_set_conv_nth length_map snd_conv)

then obtain \( i \) where:
\( s = ts' ! (R ! i) \)

by (metis (no_types, lifting) \( m(2) \) \( f(5) \) in_set_conv_nth length_map snd_conv)

have \( s' : \text{arity} \ f = \text{length} ts' \) \( \text{length} ts = \text{length} ts' \)

using \( i(1) \) Ana_Fu_intro[of \( f(2) \) \( f(2) \) \( f(4) \)] \( i(1) \) \( \text{unfolding} \ ts'(2) \) \( m'(2) \)

by simp

show \( \text{thesis} \) using \( ?\text{thesis} \) \( f \) \( m' \) \( R_i \) \( ts' \) \( s' \) \( i \) by auto

qed

have \( s = ts ! (R ! i) \)

proof (cases \( ts ! (R ! i) \))

case (Var \( x \))

hence \( \text{Var} \ x \in \text{set} ts' \) using \( R_i \) \( i \) nth_mem by fastforce

hence \( \text{x} \in \text{fv_set} M' \) using \( f(1) \) \( f(1) \) \( \text{fv_subterms_set} \) by fastforce

thus \( ?\text{thesis} \) using \( i M \) \( \text{Var} \) \( \text{by} \) \( \text{fastforce} \)

qed (use \( i \) \( s \_\text{def} \_\text{in} \text{fastforce} \))

thus \( ?s \in \text{set} (\text{snd} (\text{Ana} \ (m \cdot \delta))) \) using \( f(1) \) Ana_Fu_intro[of \( f(2) \) \( f(4) \)] \( i(1) \) \( \text{by} \) \( \text{simp} \)

have \( "s \subseteq \delta \ y" \)

when \( m : \text{m} \subseteq \text{set} \text{trms}_{\text{sst}} (\text{transaction_send} T)" \)
\( s \in \text{set} (\text{snd} (\text{Ana} (\text{m} \cdot \delta)))" \)

and \( T : \text{T} \in \text{set} P" \) and \( \delta wt : \text{\"wt_subst_trm"} \)

and \( \delta \_\text{ran} : \forall t. t \in \text{subst_range} \delta \implies (\exists c. t = \text{Fun} c [] \land \text{arity} c = 0) \lor (\exists x. t = \text{Var} x)" \)

and \( y : \text{y} \in \text{fv_set} (\text{trms}_{\text{sst}} (\text{transaction_send} T))" \)

for \( m T \) \( \delta y \)

proof -

assume \( "s \subseteq \delta \ y" \)

hence \( "T, y = \Gamma \ s" \) using \( \text{wt_subst_trm}'[\text{of} \ \delta wt, \ \text{of} \ \text{\"Var} y\"] \ \text{\delta ran}[\text{of} \ \text{\"\delta y\"}] \) by fastforce

moreover have \( "y \in \text{vars_transaction} T" \)

using \( y \) \( \text{trms}_{\text{sst}, \text{fv vars set}} \) subset unfolding \( \text{vars_transaction_unfold}[\text{of} \ T] \) by fastforce

ultimately show False using \( P(2) \) \( T \) by force

qed

hence \( \text{sublmm2:} \ "s \in \text{set} (\text{snd} (\text{Ana} \ m))" \)

when \( m : \text{m} \subseteq \text{set} \text{trms}_{\text{sst}} (\text{transaction_send} T)" \)
\( s \in \text{set} (\text{snd} (\text{Ana} (\text{m} \cdot \delta)))" \)

and \( T : \text{T} \in \text{set} P" \) and \( \delta wt : \text{\"wt_subst_trm"} \)

and \( \delta \_\text{ran} : \forall t. t \in \text{subst_range} \delta \implies (\exists c. t = \text{Fun} c [] \land \text{arity} c = 0) \lor (\exists x. t = \text{Var} x)" \)

for \( m T \) \( \delta \)

using \( \text{sublmm1}[\text{of} \ m] \) \( m T \) \( \delta wt \) \( \delta ran \) by blast

have \( "s \in \text{ik}_{\text{sst}} A \lor (\exists m \in \text{subterms set} (\text{ik}_{\text{sst}} A) \cdot \text{set} I. s \in \text{set} (\text{snd} (\text{Ana} \ m)))" \)

277
using private_const_deduct[OF c(1) s_deduct(2)[unfolded s_def]]
  I_s_x const_mem_subst_cases[of c] AIk_subst
unfolding s_def by blast

hence "?P s_cases (ikist A)" using sublmm1[of _ "ikist A"] I_s_x by blast
then obtain T ξ σ α where T: "(T, ξ, σ, α) ∈ set Ts" "?P s_cases (ikist (f' (T, ξ, σ, α)))"
  using ikist_concat[of "map f' Ta"] Ts(1)[unfolded g'_def] by fastforce

obtain B where "prefix (B@[T(1), ξ, σ, α]) Ts" by (metis T(1) prefix_snoc_in_iff)
  hence T_in_P: "T ∈ set P"
  and T_wf: "wellformed_transaction T"
  and ξσα: "transaction_decl_subst ξ σ α T (concat (map f' B))"
  and α: "transaction_renaming_subst α P (concat (map f' B))"
  using P(6) Ts(3)[unfolded g'_def] unfolding comp_def by (metis,metis,metis,metis,metis)

note σωt = transaction_decl_fresh_renaming_subs_wt[OF ξ σ α]
note σωran = transaction_decl_fresh_renaming_subs_range'(1)[OF ξ σ α]

have "subterms (M _set ξ σ σ σ α) = subterms (M _set ξ σ σ σ α)" for M
  using σωran subterms_subterm[of _ "ξ σ σ σ α"] by (meson subst_imgI)

hence s_cases: "?P s_cases' (trms ist (transaction_send T)) (ξ, σ, σ, α)"
  using T(2) dual_transaction_ik_is_transaction_send'[OF T_wf, of "ξ σ σ σ α"]
  unfolding f'_def by auto

from s_cases show False
proof
  assume "s ∈ trms ist (transaction_send T) _set ξ σ σ σ α"
  then obtain t where t: "t ∈ trms ist (transaction_send T)" "s = t · ξ σ σ σ α" by moura
  have "s ≠ t" using P(4) T_in_P t(1) by blast
  then obtain x where x: "x = TAtom Value" using t(2) unfolding s_def by (cases t) auto
  have "Γ_v x = Γ s" using x t(2) wt_subst_trm'[OF ξσωt, of "Var x"] by simp
  moreover have "x ∈ vars_transaction T" using t(1) trms,_,fv,vars,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.,.
3.6 Stateful Protocol Verification

lemma welltyped_leakage_free_occurs_factI:
  shows \("\forall A \in \text{reachable_constraints } P. \, \exists I_r. \, \text{welltyped_constraint_model } I_r. \, (A@[\lambda, \text{send}\{\{c\}\}]))\)  
and \("\text{welltyped_leakage_free_protocol } [(c),] P\) \nproof -
  have "c'': "(c), \notin \text{subterms } t"  
  when T: "T \in \text{set } P\) and t: "t \in \text{subterms}_{set} \, (\text{transaction_send } T)\) for T t  
  using t \text{bspec}\{\text{of } P(2) \, T\} \text{subtermeq_imp_fun_subterm_subset}\{\text{of } t\}  
  \text{funs_term_Fun_subterm'}\{\text{of } \text{"Set } c\} (\text{"fun}, \text{atom}, \text{"sets}, \text{"lbl}) \text{prot_term_list}\}  
  by fastforce  
  have "P'': 
  "\forall T \in \text{set } P. \, \forall t \in \text{subterms}_{set} \, (\text{transaction_send } T). \, (c), \notin \text{set } (\text{snd } (\text{Ana } t))\)"  
  "\forall T \in \text{set } P. \, (c), \notin \text{subterms}_{set} \, (\text{transaction_send } T)\)  
  subgoal using Ana_subterm' c'' by fast  
  done  
  have "P''":  
  "\forall T \in \text{set } P. \, \forall x \in \text{vars_transaction } T \cup \text{set } (\text{transaction_fresh } T). \, \Gamma (c), \notin \Gamma_v x"  
  using P(3) \text{\_const_simps}\{\text{of } P(c)\} by fastforce  
  show \("\forall A \in \text{reachable_constraints } P. \, \exists I_r. \, \text{welltyped_constraint_model } I_r. \, (A@[\lambda, \text{send}\{\{c\}\}])\)  
  "\text{welltyped_leakage_free_protocol } [(c),] P\) \n  using c \text{welltyped_leakage_free_priv_constI}\{\text{of } P(4)\} P' P(5,1), \text{of } \text{SetType}\}  
  by (force, force)  
  qed

lemma welltyped_leakage_free_occurssec_constI:
  defines "s \equiv \text{Fun OccursSec } []\)  
  assumes P:  
  "\forall T \in \text{set } P. \, \text{wellformed_transaction } T\)"  
  "\forall T \in \text{set } P. \, \forall x \in \text{vars_transaction } T \cup \text{set } (\text{transaction_fresh } T). \, \Gamma (c), \notin \Gamma_v x"  
  using P(2) by auto  
  show \("\forall A \in \text{reachable_constraints } P. \, \exists I_r. \, \text{welltyped_constraint_model } I_r. \, (A@[\lambda, \text{send}\{\{}\})\)  
  "\text{welltyped_leakage_free_protocol } [s] P\) \n  using welltyped_leakage_free_priv_constI[of _ _ _ P(5) P' P(3,4,6,1), \text{of } \text{SetType}\}  
  unfolding s_def by auto  
  qed

lemma welltyped_leakage_free_occurs_factI:
  assumes P: "\forall T \in \text{set } P. \, \text{admissible_transaction } T\)  
  and P_odd_star:  
  "\forall T \in \text{set } P. \, \forall x \in \text{set } (\text{transaction_send } T). \, \text{OccursFact} \in \{\text{funs_term } (\text{subterms}_{set} (\text{snd } r))\} \rightarrow \text{fst } r = \lambda"  
  shows \("\text{welltyped_leakage_free_protocol } [\text{occurs } x] P\) \n  proof -
  define Q where "Q \equiv \lambda M \, t \, \delta. \, t \in M \land \text{wt}_{\text{subt}_M} \, \text{delta} \land \text{wt}_{\text{trms}_{M}} \, (\text{subt_range } \delta) \land \text{fv } (t \cdot \delta) = \{\}\)  
  define f where "f \equiv \lambda M. \, (t \cdot \delta) \, \{m \, \{t \cdot \delta\} = \{\}\)  
  define Sec where "Sec \equiv f (\text{set } \{\text{occurs } x\}) - \{m. \, \{t \cdot c\} = \{\}\)"  

279
define $f'$ where 
\[
  f' \equiv \lambda (T, \xi, \sigma :: ('fun,'atom,'sets,'lbl) \text{ prot_subst}, \alpha). \\
  \text{dual} \_\text{lsst} (\text{transaction_strand } T \cdot \xi, \sigma \circ_0 \alpha)
\]

define $g'$ where 
\[
  g' \equiv \text{concat} \circ \text{map } f'
\]

note defeq = Sec_def f_def Q_def 

note defeqs = welltyped_constraint_model_def constraint_model_def 

show ?thesis 
proof (rule ccontr) 
  assume "¬ welltyped_leakage_free_protocol [occurs x] P" 
  then obtain $A$, $I$, $k$ where: 
  "$A \in \text{reachable_constraints } P""  "occurs k \in \text{Sec - declassified}_{\text{lsst}} A I"  "welltyped_constraint_model I (A \otimes ([*, \text{send}(\text{occurs k}])))"
  unfolding welltyped_leakage_free_protocol_def defeqs by fastforce 

note defeqs' = welltyped_constraint_model_prefix[OF A(3)] 

have occ_I: "occurs k \cdot I = occurs k" using A(2) unfolding defeqs by auto 

hence occ_in_ik: "occurs k \in \text{ik}_{\text{lsst}} A I" 
  using reachable_constraints_occurs_fact_deduct_in_ik[OF A(3) A' P, of k] by (argo, argo) 

then obtain $l$, $ts$ where: "$(1,\text{receive}(ts)) \in A" "occurs k \in \text{set } ts" 
  using in_ik_iff[of "occurs k" "unlabel A" ] unfolding unlabel_def by moura 

obtain $T$, $a$, $B$, $\alpha$, $\sigma$, $\xi$ 
  where $B$: "prefix ([B@f' (T, \xi, \sigma, \alpha)) A]" 
  and $T$: "$T \in \text{set } P"" "\text{transaction_decl_subst } \xi T B"  "\text{transaction_renaming_subst } \alpha P B" 
  and $a$: "a \in \text{set } (\text{transaction_strand } T)" "f_{\text{stu}} (1,\text{receive}(ts)) = f_{\text{stu}} a"  "(1,\text{receive}(ts)) = \text{dual}_{\text{lsstp}} a \cdot \text{lsstp} \xi \circ_0 \alpha" 
  using reachable_constraints_transaction_action_obtain[OF A(1) ts(1), of thesis] unfolding f'_def by simp 

obtain $ts'$ where $ts'$: "a = (l,\text{send}(ts'))" "ts = ts' \cdot \text{lsst} \sigma \circ_0 \alpha" 
  using surj_pair[of a] a(2,3) by (cases "snd a") force+ 

obtain $t$ where $t$: "t \in \text{set } ts'" "occurs k = t \cdot \xi \circ_0 \alpha" 
  using ts(2) unfolding ts'(2) by force 

have occ_t: "OccursFact \in \text{funs_term } t" 
  proof (cases t) 
    case (Var y) thus ?thesis 
    using t(2) subst_apply_term.simps(1)[of y "\xi \circ_0 \alpha"] 
    unfolding transaction_decl_fresh_renaming_substs_range'(1)[OF T(2-), of "occurs k"] by fastforce 
  qed (use t(2) in simp) 

have P_wf: "\forall T \in \text{set } P. \text{wellformed_transaction } T" 
  using P admissible_transaction_is_wellformed_transaction(1) by blast 

have l: "l = *" 
  using wellformed_transaction_strand_memberD[OF bspec[OF P_wf T(1)] a(1)[unfolded ts'(1)]] t(1) T(1) P_occ_star occ_t unfolding ts'(1) by fastforce 

have "occurs k \in \bigcup \{\text{set } ts \mid ts \cdot (*, \text{receive}(ts)) \in \text{set } (A_{\text{lsst}} I)\}" 
  using subst_last_memI[OF ts(1), of I] subst_set_map[OF ts(2), of I] unfolding occ_I l by auto 
  thus False using A(2) unfolding declassified_{lsst}_def by simp 
  qed
lemma welltyped_leakage_free_setup_pairI:
assumes P:
"∀ T ∈ set P. wellformed_transaction T"
"∀ T ∈ set P. ∃ x ∈ vars_transaction T. Γ_v x = TAtom Value ∨ (∃ a. Γ_v x = (a)_ref)"
"∀ T ∈ set P. ∀ f ∈ (fun_term · (trms|set (transaction_send T))). ¬is_Set f"
"∀ T ∈ set P. ∀ x ∈ set (transaction_fresh T). Γ_v x = TAtom Value"
"∀ T ∈ set P. transaction_decl T () = []"
"∀ T ∈ set P. admisible_transaction_terms T"
and c: "arity c = 0"
shows "welltyped_leakage_free_protocol [pair (x, (c)_s)] P"
proof -
define Q where "Q ≡ λ M t δ. t ∈ M ∧ wt (subset_range δ) ∧ f v (t · δ) = {}"
define f where "f ≡ λ M. {t · δ | t δ, Q M t δ}"
define Sec where "Sec ≡ f (set (pair (x, (c)_s))) - {m. {} ⊆ c m}"

define f' where "f' ≡ λ T,ξ.σ::('fun,'atom,'sets,1lbl) prot_subst,a).
       dual_set (transaction_strand T 'lassst ξ o σ o a)"
define g' where "g' ≡ concat o map f'"

note defs = Sec_def f_def Q_def
note defs' = welltyped_constraint_model_def constraint_model_def

have P':
"∀ T ∈ set P. ∀ x ∈ vars_transaction T ∪ set (transaction_fresh T). Γ_v x ≠ TAtom SetType"
"∀ T ∈ set P. ∀ x ∈ vars_transaction T. is_Var (Γ_v x)"
"∀ T ∈ set P. ∀ x ∈ set (transaction_fresh T). Γ_v x = TAtom Value ∨ (∃ a. Γ_v x = (a)_ref)"
subgoal using P(2,4) by fastforce
subgoal using P(2) by fastforce
subgoal using P(4) by fastforce

done

note 0 = welltyped_leakage_free_setup_constI[1]OF P(1,3) P' c

show ?thesis
proof (rule ccontr)
  assume A x k where A:
"A ∈ reachable_constraints P" "pair (k, (c)_s) ∈ Sec - declassified|set A I"
"welltyped_constraint_model I (A@[(*, send[pair (k, (c)_s)]]))"

  unfolding welltyped_leakage_free_protocol_def defs pair_def by fastforce

note A' = welltyped_constraint_model_prefix[OF A(3)]

have "pair (k, (c)_s) · I = pair (k, (c)_s)" using A(2) unfolding defs by auto
  hence "ik_{set} A ·set I ⊢ pair (k, (c)_s)"
    using welltyped_constraint_model_deduct_split[OF A(3)] by argo
  then obtain n where n: "intruder_deduct_num (ik_{set} A ·set I) n (pair (k, (c)_s))"
    using deduct_num_if_deduct by fast

  have "wt_{set} I" "wt_{set} (subset_range I)" "ik_{set} A ⊆ trms|set A"
    using A(3) ik_{set} trms|set subset unfolding defs' by simp_all
  hence "Aik_{set} A ·set I ⊆ SMP (trms|set A)" by blast
  hence "Pair ∉ (fun_term · (ik_{set} A ·set I))"
    using reachable_constraints_no_Pair_fun[OF A(1) P(4-6)] P by blast
  hence 1: "¬pair (k, (c)_s) ∈set ik_{set} A ·set I"
    using fun_term_FUN_subterm'[of Pair] unfolding_pair_def by auto

  have 2: "pair (k, (c)_s) ∉ set (snd (Ana m))" when m: "m ∈set ik_{set} A ·set I" for m
    using m i term dual_order tran Ana_subterm'[of "pair (k, (c)_s)" m] by auto
  have "¬ik_{set} A ·set I ⊢ (c)_s"
    using A(1) A' welltyped_constraint_model_deduct_if[of I A ·"(c)_s"] by force
    moreover have "ik_{set} A ·set I ⊢ (c)_s"

  note 0 = welltyped_leakage_free_setup_constI[1]OF P(1,3) P' c
lemmas [simp del] welltyped_leakage_free_protocol_def

proof (auto)

assumes "S: [x: value]" "P: [s]" "Q: [t]"

shows "welltyped_constraint_model (A' [s] I) (A' [t] Q)" "interpretation [s] A" "interpretation [t] A"

using (blast,blast,blast)

qed

end


documented in the appendix.

In addition, we can prove the following lemmas:

lemma welltyped_leakage_free.short_term_secretI:

proof -

end

lemma welltyped_leakage_free.short_term_secretI:

proof -

end

end
have k: "∀ k = TAtom Value" "fv k = {}" "wf_term k"
subgoal using δ(1) wt_subst_term' [OF δ(2), of "(x: value)"] by simp
subgoal using δ(1,4) by blast
subgoal using δ(1,3) by force
done
then obtain fk where fk: "k = Fun fk []"
  using const_type_inv_wf by (cases k) auto

have ""welltyped_constraint_model I (A@[x, send([(c), c])][])""
  using subterms_sec(1) A(1) by blast
hence ""ink I ⊑ (c)""
  using A' strand_see_append_stateful[of "{}" "{}" "unlabel A" "unlabel [(x, send([(c), c])][])" I]
unfolding def's by auto
hence ""(fk I) (c) ∈ ik I""
  using fk' deduct_inv'[OF fk'] by force
moreover have "k ⊑ (fk I) (c)" by simp
ultimately have k_in_ik: "k ⊑ ik I" by blast
hence "k ⊑ ik I" by blast
hence "k ⊑ ik I" by blast
moreover have "I ⊑ (c)" by blast
ultimately have k_in_ik: "k ⊑ ik I" using in_subterms_subset_Union by blast
hence "k ⊑ ik I" by blast

obtain α::"('fun, 'atom, 'sets, 'lbl) prot_subst" where α: "transaction_renaming_subst α P A"
unfolding transaction_renaming_subst_def by blast

obtain y' y'n' where y':
  "α (the_Var y') = Var y'" "y' ≠ y'n'" "Var y' = (y'n': value)"
using transaction_renaming_subst_is_renaming(1,2)[OF α] by force

define B where "B ≡ A @ dual I" (transaction_strand Tatt · I α)
def J where "J ≡ I(y' := k)"

have "y' ∈ range_vars α"
  using y'(1) transaction_renaming_subst_is_renaming(3)[OF α]
  by (metis (no_types, lifting) in_mono subst_fv_imgI term.set_intros(3))
hence y': "y' ≠ vars B α A"
  using transaction_renaming_subst_vars_disj(6)[OF α] by blast
have 0: "(k,⟨d⟩) ∉ set (db I A)"

283
3 Stateful Protocol Verification

proof
assume a: "(k, (d, s)) ∈ set (db, set (A I))"

obtain t t’ where t: "(l, insert(t, t’)) ∈ set A" "t ∙ I = k" "t’ ∙ I = (d, s)"
using db, set (cases A [OF A [unfolded db, set (def)] unfolding unlabeled_def by auto)

obtain T b B α σ ξ where T:
  "prefix (db, dual, set (transaction_strand T) = dual, set (ξ = o, σ = o, α)) A"
  "T ∈ set P" "transaction_decl_subst (ξ = T)"
  "transaction_fresh_subst σ τ T B" "transaction_renaming_subst_subst α P B"
  "b ∈ set (transaction_strand T)" "(l, insert(t, t’)) = dual, subst (t, t’), insert (l, insert (t, t’)) = T wf (l, insert (t, t’)) = T uf b"
using reachable_constraints_transaction_action_obtain [OF A (1)] (t (1)) by metis

define φ where "φ ∋ ξ = o, σ = o, α"

obtain b’ where "b = (l, b’)"
using T (8) by (cases b) simp
then obtain tb tb’
  where b’: "b = (l, insert(tb, tb’))"
  and tb: "t = t’ ∙ φ"
  and tb’: "t’ = t’ ∙ φ"
using T (7) unfolding φ_def by (cases b’) auto

note T_adm = b spec [OF P (1)] (T (2))

note T wf = admissible_transaction_is_wellformed_transaction (1, 3) [OF T adm]

have b: "b ∈ set (transaction_updates T)"
using transaction_strand_member [OF T (6) [unfolded b’]]
wellformed_transaction_cases [OF T uf (1)]

unfolding b’ unfolding by blast

have "∃ n. tb = (n, value, s)" using tb tb’ T (6) (t (3)) transaction_inserts_are_value_vars [OF T uf, of tb tb’]
unfolding b’ unfolding by (force, force)

have "is, Insert (and b)" "the_set_term (and b) = (d, s)" "the_set_term (and b) = tb"
unfolding b’ unfolding by simp_all

hence "transaction_send T ≠ [1]"
  "let (l, a) = hd (transaction_send T)"
  in l = * ∧ is, Send a ∧ [f [[(c), tb]]ₜ] ∈ set (themsgs a)"
using P d T (2) b by (fast, fast)

hence "∃ ts. (*, send (ts)) ∈ set (transaction_send T) ∧ (f [[(c), tb]]ₜ) ∈ set ts" unfolding is, Send def (cases "transaction_send T") auto
then obtain ts where ts: "(*, send (ts)) ∈ set (transaction_strand T)" "(f [[(c), tb]]ₜ) ∈ set ts"
unfolding transaction_strand_def by auto

have "(*, receive (ts ‘isat φ) ∈ set A)" "(f [[(c), t]]ₜ) ∈ set (ts ‘isat φ)"
using subst_last_mem [OF ts (1), of φ] subst_set_map [OF ts (2), of φ]
dual, subst_step_iff (1) [of * ts ‘isat φ] "transaction_strand T ‘isat φ"
set_mono_prefix [OF T (1) [unfolded φ_def [symmetric]]]
unfolding by auto

hence "(f [[(c), k]]ₜ) ∈ (set ts | ts. (*, receive (ts) ∈ set (A ‘isat I)))" using (2) subst_last_mem [of "(*, receive (ts ‘isat φ)”) A I] by force

thus False using (A (2)) unfolding declassified ‘isat_def by auto

qed

have "y’ ∉ fv_set (ik ‘isat A)"
using y’ fv ‘ik subset vars ‘sst’ [of “unlabel A"] by blast

hence I: "ik ‘isat A ‘isat I = ik ‘isat A ‘isat J" unfolding J def by (metis (no_types, lifting) fv ‘isat subset image ‘cong in_mono repl_invariance)

have "(k, (d, s)) ∉ db, set (unlabel A) I (1)"
hence "(\langle d, n \rangle \not\in \text{dupb}_{\text{sset}} (\text{unlabel } A) \ J \}"

using \(y'\) \text{vars}_{\text{sset}} \text{isfv}_{\text{sset}} \text{bvars}_{\text{sset}}[\text{of } \text{unlabel } A]
\text{db}_{\text{sset}} \text{subst}_{\text{swap}}[\text{of } \text{unlabel } A] \ J \{\}
\text{db}_{\text{sset}} \text{isdupb}_{\text{sset}}[\text{of } \text{unlabel } A] \ J \{\}

unfolding \text{db}_{\text{sset}} \text{def} \ J \text{def} by \text{metis} \ (\text{no_types, lifting}) \ \text{Un}_{\text{iff}} \ \text{empty}_{\text{set}} \ \text{fun}_{\text{upd_other}}

hence "((\langle d, n \rangle - J, \langle d, n \rangle - J) \not\in \text{dupb}_{\text{sset}} (\text{unlabel } A) \ J \{\})"

unfolding \text{J}_{\text{def}} \text{fk} by \text{simp}

hence "\text{strand}_{\text{sset}} \text{sem}_{\text{sset}} \{\} \{\} \ (\text{unlabel } \{A \not\in \text{vars}_{\text{sset}} \text{isfv}_{\text{sset}} \text{bvars}_{\text{sset}}[\text{of } \text{unlabel } A]\}) \ (\text{unlabel } \{A \not\in \text{vars}_{\text{sset}} \text{isfv}_{\text{sset}} \text{bvars}_{\text{sset}}[\text{of } \text{unlabel } A]\}) \ (\text{unlabel } \{A \not\in \text{vars}_{\text{sset}} \text{isfv}_{\text{sset}} \text{bvars}_{\text{sset}}[\text{of } \text{unlabel } A]\}) \ J"

have \(B \in \text{reachable}_{\text{constraints}} P\)
using \text{reachable}_{\text{constraints}}.\text{step}[\text{OF } A(1) P(1) \text{ Tatt}_{\text{def}} \ A', \alpha, \text{of } \text{Var } \text{Var}]

unfolding \text{B}_{\text{def}} \text{Tatt}_{\text{def}} \text{transaction}_{\text{decl}_{\text{sub}}}_{\text{def}} \text{transaction}_{\text{fresh}_{\text{sub}}}_{\text{def}} \text{by} \text{simp}

have \(Tatt\) ': "\text{dual}_{\text{sset}} \ (\text{transaction}_{\text{strand}} \text{Tatt}_{\text{def}} A_{\text{sset}} \alpha) =
\[\langle *, \text{send}\{\text{occurs } (\text{Var } y')\}\rangle,
\langle 1, \text{send}\{\text{if } \langle c, \text{send } y' \rangle\}\rangle,
\langle 1', \langle \text{Var } y' \not\in \text{in } d)\rangle\rangle,
\langle n, \text{receive}\{\text{attack}\{\text{\text{ln } n}\}\}\rangle\]\n
using \(y'\)

unfolding \text{Tatt}_{\text{def}} \text{transaction}_{\text{strand}_{\text{def}}} \text{dual}_{\text{sset}}_{\text{def}} \text{sub} \text{apply}_{\text{labeled}_{\text{stateful}} \text{strand}_{\text{def}}}

by \text{auto}

have \(J\) ': "\text{wt}_{\text{sset}} \ J'" \ "\text{interpretation}_{\text{sub} \text{set}} \ J''" \ "\text{wt}_{\text{frms}} (\text{sub} \text{set}_{\text{range}} J)"

unfolding \text{J}_{\text{def}} \text{fk}

subgoal using \text{wt}_{\text{sub} \text{set}} \text{sub} \text{upd}[\text{OF } I(1) K(1) y'(3) \text{by} \text{simp}]

subgoal by \text{metis} I(2) K(2) \text{fun}_{\text{upd}_{\text{app}} \text{interpretation}_{\text{grounds}} \text{all} \text{interpretation}_{\text{sub} \text{set}I})

subgoal using I(3) K(3) \text{by} \text{fastforce}
done

have \(3\) ': "\text{ik}_{\text{sset}} \text{A}_{\text{sset}} \ J \cup \{\langle c, \text{send } y' \rangle\} \cdot J"

using 1 \text{fk} \text{fk}' \text{unfolding} \text{J}_{\text{def}} \text{by} \text{auto}

have \(4\) ': "\text{ik}_{\text{sset}} \text{A}_{\text{sset}} \ J \cup \text{occurs } (\text{Var } y') \cdot J"

using \text{reachable}_{\text{constraints}}.\text{occurs}_{\text{fact}_{\text{ik}_{\text{case}'}}}[\text{OF } A(1) P(1) \text{constraint}_{\text{model}} \text{Val}_{\text{const}} \text{in} \text{const}_{\text{prefix}}'[\text{OF } A(1) A' P(1) K(1) \text{\text{ik}_{\text{in}}\text{\text{\text{ik}_{\text{unfolded}} \text{fk} \text{kn}}}}]

\text{intruder}_{\text{ deduct}}.\text{Axiom}[\text{of } \text{occurs } k' \ "\text{ik}_{\text{sset}} \text{A}_{\text{sset}} \ J'"\]

unfolding \text{J}_{\text{def}} \text{fk} \text{kn} \text{by} \text{fastforce}

have "\text{strand}_{\text{sset}} \text{sem}_{\text{sset}} \{\} \{\} \ (\text{unlabel } A) \ J"

using 2 \text{strand}_{\text{sset}} \text{append}_{\text{stateful}} \text{by} \text{force}

hence "\text{strand}_{\text{sset}} \text{sem}_{\text{sset}} \{\} \{\}
\langle \text{unlabel } \{A \not\in \{*, \text{send}\{\text{occurs } (\text{Var } y')\}\}\rangle,
\langle n, \text{send}\{\text{if } \langle c, \text{send } y' \rangle\}\rangle\rangle,
\langle n, \langle \text{Var } y' \not\in \text{in } d)\rangle\rangle\rangle\ J"

using 2 3 4 \text{strand}_{\text{sset}} \text{append}_{\text{stateful}}[\text{of } \{\} \{\} \ _ \ _ \ J]

unfolding \text{unlabel}_{\text{def}} \text{ik}_{\text{sset}}_{\text{def}} \text{by} \text{force}

hence "\text{strand}_{\text{sset}} \text{sem}_{\text{sset}} \{\} \{\} \ (\text{unlabel } \{B \not\in \{n, \\text{send}\{\text{attack}\{\text{\text{ln } n}\}\}\}\} \ J"

using \text{strand}_{\text{sset}} \text{receive}_{\text{send}} \text{append}[\text{of } \{\} \{\} \ _ \ _ \ \text{attack}\{\text{\text{ln } n}\}]

\text{strand}_{\text{sset}} \text{append}_{\text{stateful}}[\text{of } \{\} \{\} \ _ \ _ \ J]

unfolding \text{B}_{\text{def}} \text{Tatt'} \text{by} \text{simp}

hence "\text{well}_{\text{typed}_{\text{constraint}_{\text{model}}} \ J \{B \not\in \{n, \\text{send}\{\text{attack}\{\text{\text{ln } n}\}\}\}})"
lemma welltyped_leakage_free_short_term_secretI':

fixes c x y f n d l l' · T

defines "s = \{ \{c\}, \Var(T\Atom\tau,x)\}\"
and "Tatt = Transaction (\lambda(t). [] ) []
[(1', receive([\{c\}, \Var(T\Atom\tau,y)\}] )]
[(1', \Var(T\Atom\tau,y) \not\in \{d\}_s ) ]
[(n, \send([\attack(ln n)])] )"

assumes P:

"\forall T \in set P. wellformed_transaction T"

"\forall T \in set P. \forall x \in set (transaction_updates T) .
\is_Update x \longrightarrow \is_Fun (the_set_term x)"

and subterms_sec:

"\forall A \in reachable_constraints P. \#I . welltyped_constraint_model I, (A\emptyset([*, \send([\{c\},I])] ) )"

and P_sec:

"\forall A \in reachable_constraints P. \#I .
welltyped_constraint_model I, (A\emptyset([n , \send([\attack(ln n)])] ) )"

and P,Tatt: "Tatt \in set P"

and P_d:

"\forall T \in set P. \forall a \in set (transaction_updates T).
\is_Insert (snd a) \land the_set_term (snd a) = (d)_s \longrightarrow
\transaction_send T \not\in [] \land (let (1,b) = hd (transaction_send T)
in l = t \land \is_Send b \land [\{c\}, the_elem_term (snd a)]_l \in set (the_msgs b) )"

shows "welltyped_leakage_free_protocol [s] P"

proof -

define Q where "Q = \lambda M t . \delta . t \in M \land wt_subst \delta \land \wf_subterms (subst_range \delta) \land \fv (t \cdot \delta) = {}"

define Sec where "Sec \equiv \{ t \cdot \delta | t \cdot \delta . Q (set [s]) t \delta \} - \{. \} \vdash c . m\}"

define f' where "f' = \lambda (T,\xi,\alpha '::('fun,'atom,'sets,'lbl) prot_subst,\alpha) .
dual_i (transaction_strand T \cdot (:\xi)\circ\alpha\circ\alpha )"

define g' where "g' = \equiv \concat \circ \map f'"

note defs = Sec_def Q_def

note defs' = welltyped_constraint_model_def constraint_model_def

show \?thesis

proof (rule ccontr)
assume "¬ welltyped_leakage_free_protocol [s] P"

then obtain A I k where A:

"A \in reachable_constraints P" 
"\{ \{c\}, k\}_I \in Sec - \declassified_i : A I" 
"welltyped_constraint_model I (A\emptyset([*, \send([\{c\},I])] ) )"

unfolding welltyped_leakage_free_protocol_def defs s_def by fastforce

have I: "\wf_subst I" "interpretation_subscript I" "\wf_subterms (subst_range I)"

using A(3) unfolding defs' by (blast,blast,blast)

note A' = welltyped_constraint_model_prefix[OF A(3)]

have "\strand_sem_stateful \{\} \{\} (unlabel A) I"

using A' unfolding defs' by simp

hence A': "\strand_sem_stateful \{\} \{\} (unlabel A) (I(z := k) )"

when z: "z \notin fv_i : A" for z

using z strand_sem_model_swap[of "unlabel A" I "I(z := k)" "\{\} " "\{\}"] by auto

obtain \delta where \delta:

"\delta (T\Atom\tau,x) = k" "\wf_substit \delta" "\wf_subterms (subst_range \delta)" "\fv (\delta (T\Atom\tau,x)) = {}"

using A(2) unfolding defs a_def by auto
have k: "k = TAtom τ" "fv k = {}" "wf_term k"
subgoal using δ(1) wt_subst_term'[OF δ(2), of "Var (TAtom τ,x)"] by simp
subgoal using δ(1,4) by blast
subgoal using δ(1,3) by force
done
then obtain fk where fk: "k = Fun fk []"
using const_type_inv_wf by (cases k) auto

have fk': "lsst A · set I ⊢ ⟨c, k⟩ t"
using fk welltyped_constraint_model_deduct_split(2)[OF A(3)] by auto
obtain α::"('fun,'atom,'sets,'lbl) prot_subst" where α: "transaction_renaming_subst α P A"
unfolding transaction_renaming_subst_def by blast
obtain y' yn' where y': "α (TAtom τ,y) = Var y'" "y' ≠ yn'" "y' = (TAtom τ,yn')"
using transaction_renaming_subst_is_renaming(1,2)[OF α] by force

define B where "B ≡ A@dual lsst (transaction_strand Tatt · lsst α)"
define J where "J ≡ I(y' := k)"

have y' ∈ range_vars α
using y'(1) transaction_renaming_subst_is_renaming(3)[OF α] by (metis (no_types, lifting) in_mono subst_fv_imgI term.set_intros(3))
hence y'': "y' /∈ vars lsst A"
using transaction_renaming_subst_vars_disj(6)[OF α] by blast

have 0: "(k,(d,s)) /∈ set (db lsst A I)"
proof
assume a: "(k,(d,s)) ∈ set (db lsst A I)"
obtain l t t' where t: "l,insert(t,t') ∈ set A" "t · I = k" "t' · I = (d,s)"
using db_lsst_in_cases[unfolded db_lsst_def] by (auto)

obtain b B α σ ξ where T:
"prefix (B@dual lsst (transaction_strand T · lsst ξ ◦ s σ ◦ s α)) A"
"T ∈ set P" "transaction_decl_subst ξ T"
"transaction_fresh_subst α P B" "transaction_renaming_subst α P B"
"b ∈ set (transaction_strand T)" "(l, insert(t,t')) = dual lsstp b lsstp ξ ◦ s ◦ s α"
"fst (l, insert(t,t')) = fst b"
using reachable_constraints_transaction_action_obtain[OF A(1) t(1)] by metis

define ϑ where "ϑ ≡ ξ ◦ s ◦ s α"
obtain b' where "b = (l,b')"
using T(8) by (cases b) simp
then obtain tb tb'
where b': "b = (l,insert(tb,tb'))"
and tb: "t = tb · ϑ"
and tb': "t' = tb' · ϑ"
using T(7) unfolding ϑ_def by (cases b') auto

note T_wf = bspec[OF P(1) T(2)] bspec[OF P(2) T(2)]
hence [*]: "tb' = ⟨d⟩," using t(3) unfolding b' tb' by force

have "is_Insert (and b)" "the_set_term (and b) = ⟨d⟩," "the_elem_term (and b) = tb"
unfolding b' * by simp_all
hence "transaction_send T ≠ {}"

let (a, d) = hd (transaction_send T)
in 1 = * ∧ is_Send a ∧ [f ∥ (c) ∥, tb] ∈ set (the_mags a)
using P_d (2) b by (fast, fast)

hence "ts. ⟨*, send(ts)⟩ ∈ set (transaction_send T) ∧ ⟨f ∥ (c) ∥, tb⟩ ∈ set ts"
unfolding is_Send_def by (cases "transaction_send T") auto
then obtain ts where ts: "⟨*, send(ts)⟩ ∈ set (transaction_strand T)" "⟨f ∥ (c) ∥, tb⟩ ∈ set ts"
unfolding transaction_strand_def by auto

have "(⋆, receive(ts ∨ ist ∨)) ∈ set A" "f ∥ ⟨(c) ∥, t⟩ ∥ ∈ set (ts ∨ ist ∨)
using subst_last_mem[OF ts(1), of ∅] subst_set_map[OF ts(2), of ∅]
dual_step_steps_iff(1)[of "⋆ ts ∨ ist ∨" "transaction_strand T ∨ ist ∨"]
set_mono_prefix[OF T(1)[unfolded ∅ def[symmetric]]]
unfolding tb by auto

hence "f ∥ ⟨(c) ∥, k⟩ ∥ ∈ set ts ∨ ist ∨. ⟨*, receive(ts)⟩ ∈ set (A ∨ ist ∨)
using t(2) subst_last_mem[of "⋆ ts ∨ ist ∨"] A I] by force
thus False
using A(2) unfolding declassified_set_def by auto

qed

have "y' ∉ fv_set (ik ∪ J)" using 0 db_set_is_dbupd[unlabel A] I ()
unfolding db_set_def by force

hence "(Var y' ∙ J, ⟨d⟩ ∙ J) ∉ dbupd_set (unlabel A) J {}")
unfolding dbupd_def by simp

hence "ik ∪ J = ik ∪ J ∪ J"
unfolding subst_def by (metis (no_types, lifting) fv_subset_image_cong_in_mono repl_invariance)

have "(Var y' ∙ J, ⟨d⟩ ∙ J) ∉ dbupd_set (unlabel A) J {}")
unfolding dbupd_def by simp

hence "ik ∪ J = ik ∪ J ∪ J"
unfolding subst_def by simp

hence "strand_sem_stateful (ik ∪ J) (dbupd_set (unlabel A) J {}")
   (unlabel [(n, (Var y' ∉ in (d)) )]) J"
using stateful_strand_sem_NegChecks_no_bvars(1)
   of "ik ∪ J = dbupd_set (unlabel A) J " "Var y' ∉ (d), J " J"
by simp

hence "strand_sem_stateful {() ∉ unlabel A} (A@[n, (Var y' ∉ in (d))]) J"
using A' y' y' | vars_set_is fv_set_bvars[unlabel A]
strand_sem_append_stateful
   of "{()} " () " unlabel A " unlabel [(n, (Var y' ∉ in (d)))] J"
unfolding J_def by simp

have B: "B ∈ reachable_constraints P"
using reachable_constraints.step[OF A(1) P_Tatt _ α, of Var.Var]
unfolding B_def Tatt_def transaction_decl_subset_def transaction_fresh_subst_def by simp

have Tatt': "dual_set (transaction_strand Tatt ∪ α) =
   [(l, send([(c, Var y')])),
   (l', (Var y' ∉ in (d)))]
   n, receive[source](ln n)]")"
using y'
unfolding Tatt_def transaction_strand_def dual_set_def subst_apply_labeled_stateful_strand_def
by auto

288
have J: "\text{interpretation_subst } J" "\text{interpretation_terms } J" "\text{interpretation_subst_range } J"
  unfolding J_def
  subgoal using \text{interpretation_subst_upd}[OF I(1)] k(1) y'(3) by simp
  subgoal by (metis I(2) k(2) fun_upd_apply interpretation_grounds_all interpretation_subst)
  subgoal using I(3) k(3) by fastforce
done

have 3: "\text{ik}_\text{set} A \vdash \{ f \mid \text{Var } y' \} \cdot J"
  using i fk fk' unfolding J_def by auto

have "\text{strand_sem}_{\text{stateful}} \{ \} \{ \} (\text{unlabel } A) J"
  using 2 strand_sem_append_stateful by force

have "\text{strand_sem}_{\text{stateful}} \{ \} \{ \} (\text{unlabel } A) J"
  unfolding \text{unlabel_def} ik_set_def by force

have "\text{strand_sem}_{\text{stateful}} \{ \} \{ \} (\text{unlabel } B(1) P_{\text{sec}}) J"
  using B J unfolding declassified set by blast
thus False using B(1) P_{\text{sec}} by blast

qed

definition \text{welltyped_leakage_free_invkey_conditions}" where
"\text{welltyped_leakage_free_invkey_conditions}" invfun privfunsec declassified set S n P ≡
let f = \lambda s. is_Var s ∧ fast (the_Var s) = TAtom Value;
  g = \lambda s. is_Fun s ∧ args s = [] ∧ is_Set (the_Fun s) ∧
  arity (the_set_term (snd a)) = 0;
  h = \lambda s. is_Fun s ∧ args s = [] ∧ is_Fu (the_Fun s) ∧
  public f (the_Fun (the_Fun s)) ∧ arity f (the_Fun (the_Fun s)) = 0
in (∀s∈set S. f s ∨
  (is_Fun s ∧ the_Fun s = Pair ∧ length (args s) = 2 ∧ g (args s ! 1)) ∨
  g s ∨ h s ∨ s = privfunsec ∨ s = Fun OccursFact [] ∨
  (is_Fun s ∧ the_Fun s = OccursFact ∧ length (args s) = 2 ∧
    args s ! 0 = Fun OccursFact []) ∨
  (is_Fun s ∧ the_Fun s = Fun invfun ∧ length (args s) = 2 ∧
    args s ! 0 = privfunsec ∧ f (args s ! 1)) ∨
  (is_Fun s ∧ is_Fu (the_Fun s) ∧ fv s = {}) ∧
  Transaction (λ(). [] [] (\text{in}, receive{[s]})) [] [] (\text{in}, send{[attack(ln n)]})\in set P) ∧
  (¬public f privfunsec ∧ arity f privfunsec = 0 ∧ Γ_f privfunsec = None) ∧
(∀T∈set P. \text{transaction_fresh } T \neq [] ⟷
  transaction_send T \neq [] ∧
  (let (1, a) = hd (transaction_send T) ∧
    in 1 = * ∧ \text{is_Send } a ∧ \text{Var } \text{set } (\text{transaction_fresh } T) \subseteq \text{set } (\text{the_msgs } a)) ∧
  (∀T∈set P. ∀x∈vars transaction T. \text{is_Var } (Γ_v x) ) ∧
  (∀T∈set P. ∀x∈set (\text{transaction_fresh } T). Γ_v x = \text{Var Value} ∨ (∃a. Γ_v x = (a)_x)) ∧
  (∀T∈set P. ∀f∈∪(\text{funs_term } \setminus \text{trms}_{ss}_{\text{set}}) (\text{transaction_send } T)). \text{is_Set } f ) ∧
  (∀T∈set P. ∀r∈set (\text{transaction_send } T).
    OccursFact ∈ ∪(\text{funs_term } \setminus \text{trms}_{ss}_{\text{sp}} (\text{snd } r)) ⟷ \text{has_Label } r ) ∧
  (∀T∈set P. ∀t∈\text{subterms}_{ss}_{\text{set}} (\text{trms}_{ss}_{\text{set}}) (\text{transaction_send } T)).
    \text{privfunsec}_e ∉ set (\text{snd } (\text{Ana } t)) ∧
  (∀T∈set P. \text{privfunsec}_e ∉ \text{trms}_{ss}_{\text{set}} (\text{transaction_send } T)) ∧
  (∀T∈set P. ∀ac∈set (\text{transaction_updates } T).
    \text{is_Insert } (\text{snd } a) ∧ \text{the_set_term } (\text{snd } a) = (\text{declassified set})_s ⟷
    \text{transaction_send } T \neq [] ∧
    (let (1, b) = hd (transaction_send T) ∧
      in 1 = * ∧ \text{is_Send } b ∧
      (invfun [(\text{privfunsec})_e, the_elem_term (\text{snd } a)]_t ∈ set (\text{the_msgs } b)))"
3 Stateful Protocol Verification

definition welltyped_leakage_free_invkey_conditions where
"welltyped_leakage_free_invkey_conditions invfun privfunsec declassifiedset S n P ≡
let Tatts = λR. Transaction (λ(). []) []
(R#(n, receive(⟨⟨invfun (⟨prinvfunsec⟩), (0: value)⟩⟩)),)]])
[(x, ⟨(0: value)⟩, not in (declassifiedset)])]
[(n, send([attack ln n]))]
in welltyped_leakage_free_invkey_conditions' invfun privfunsec declassifiedset S n P ∧
(if Tatts [(*, receive([occurs (0: value)])]) ∈ set P
then ∀T∈set P. admissible_transaction T
else Tatts [] ∈ set P ∧
(∀T∈set P. wellformed_transaction T) ∧
(∀T∈set P. admissible_transaction_terms T) ∧
(∀T∈set P. bvars_transaction T = {} ) ∧
(∀T∈set P. transaction_decl T () = []) ∧
(∀T∈set P. ∀x ∈ set (transaction_fresh T). let τ = fst x
in τ = TAtom Value ∧ τ ≠ Γ (privfunsec) ) ∧
(∀T∈set P. ∀x ∈ set (transaction_send T). let τ = fst x
in is_Var τ ∧ (the_Var τ = Value ∨ is_Atom (the_Var τ)) ∧ τ ≠ Γ (privfunsec) ) ∧
(∀T∈set P. ∀x ∈ set (transaction_send T)).
Fun OccursSec [] ∉ set (snd (Ana t))) ∧
(∀T∈set P. Fun OccursSec [] ∉ trms s.et (transaction_send T)) ∧
(∀T∈set P. ∀x ∈ set (transaction_updates T)).
is_Update x → is_Fun (the_set_term x) ∧
(∀x ∈ set S. is_Fun s → the_Fun s ≠ OccursFact))"

lemma welltyped_leakage_free_invkeyI:
assumes P_wt_secure: "∀A ∈ reachable_constraints P. 
∃I. welltyped_constraint_model I (A#(n, send([attack ln n]))))"
and a: "welltyped_leakage_free_invkey_conditions invfun privsec declassifiedset S n P"
shows "welltyped_leakage_free_protocol S P"

proof -
let Tatt' = "λR C. Transaction (λ(). []) [] R C [] [(n, send([attack ln n]))]]
          ::(fun, atom, sets, lbl) prot_transaction"
let Tatt = "λR. Tatt' (R#(n, receive(⟨⟨invfun (⟨prinvfunsec⟩), (0: value)⟩⟩)),)]])
          [(x, ⟨(0: value)⟩, not in (declassifiedset)])]

define Tatt1 where "Tatt1 ≡ Tatt [(*, receive([occurs (0: value)])])"
define Tatt2 where "Tatt2 ≡ Tatt []"
define Tatt_lts where "Tatt_lts ≡ λs. ?Tatt' [(n, receive(s)]) [] []"

note defs = welltyped_leakage_free_invkey_conditions_def Let_def
note defs' = defs welltyped_leakage_free_invkey_conditions'_def
note Tatts = Tatt_def Tatt2_def Tatt_lts_def

obtain at where: “public, privsec "arity privsec = 0" Γ ′ privsec = Some at"
using a unfoldingdefs by fast

have *: "∀T∈set P. admissible_transaction T" when "Tatt1 ∈ set P"
using that unfolding defs Tatts by meson

have **: "Tatt1 ∈ set P ∨ Tatt2 ∈ set P" using a unfoldingdefs Tatts by argo

have ***: "∀T∈set P. ∀x ∈ set (transaction_fresh T). Γv x = TAtom Value ∧ Γv x ≠ Γ (privsec) "
"∀T∈set P. ∀x ∈ vars_transaction T. ∃a. Γv x = TAtom a ∧
(a = Value ∨ (∃b. a = Atom b)) ∧ TAtom a ≠ Γ (privsec) 
when "Tatt1 ∉ set P"
subgoal using a that Γv "TAtom'"(2) unfolding defs Tatts by meson

90

3.6 Stateful Protocol Verification

using a that \( \Gamma \vdash \text{TAtom}'(1,2) \)

unfolding \text{defs} \text{Tatts[symmetric]} \text{is_Atom_def} \text{is_Var_def} by fastforce

done

have ****: "s \notin occurs x"

when "Tatt1 \notin set P" "s \in set S" for a x

using a that ** unfolding \text{defs} \text{Tatts} the_Fun_def by fastforce

have 1:

\( \forall T \in \text{set P}. \text{transaction_fresh } T \not= [] \longrightarrow\)

\( \text{transaction_send } T \not= [] \land\)

(\( \text{let } (1, a) = \text{hd } (\text{transaction_send } T) \)

in \( 1 = * \land \text{is_Send } a \land \text{Var } T \subset (\text{transaction_fresh } T) \subset \text{set } (\text{the_msgs } a)\))

\( \forall T \in \text{set P}. \forall x \in \text{vars_transaction } T. \text{is_Var } (\Gamma_v x)\)

\( \forall T \in \text{set P}. \forall x \in \text{set } (\text{transaction_fresh } T). \Gamma_v x = \text{TAtom Value} \lor (\exists a. \Gamma_v x = (a)_{\text{ra}})\)

\( \forall T \in \text{set P}. \forall x \in \text{set } (\text{transaction_fresh } T). \Gamma_v x \not= \text{TAtom OccursType}\)

\( \forall T \in \text{set P}. \forall x \in \text{set } (\text{transaction_fresh } T) \cup \text{set } (\text{transaction_send } T) \). \text{Fun OccursSec } [] \not\in \text{set } (\text{snd } (\text{Ana } t))\)

\( \forall T \in \text{set P}. \forall x \in \text{subterms } (\text{trms}_\text{set } (\text{transaction_send } T)). \text{Fun OccursSec } [] \not\in \text{set } (\text{snd } (\text{Ana } t))\)

\( \forall T \in \text{set P}. \forall x \in \text{subterms } (\text{trms}_\text{set } (\text{transaction_send } T)) \). \text{Fun OccursSec } [] \not\in \text{set } (\text{snd } (\text{Ana } t))\)

\( \forall T \in \text{set P}. \text{bvars_transaction } T = []\)

\( \forall T \in \text{set P}. \forall x \in \text{set } (\text{unlabel } (\text{transaction_updates } T)). \text{is_Update } x \longrightarrow \text{is_Fun } (\text{the_set_term } x)\)

subgoal using a * unfolding \text{defs} by (metis admissible_transaction_is_wellformed_transaction(1))

subgoal

apply (cases "Tatt1 \notin set P")

subgoal using a * admissible_transactionE(2,3) \( \Gamma_{\text{Fu_simps}}(4) \) unfolding \text{defs} \text{Tatts} by force

subgoal using a \( \Gamma_{\text{Fu_simps}}(4) \) unfolding \text{defs} \text{Tatts} by fastforce

done

subgoal using a * admissible_transaction_is_wellformed_transaction(4) unfolding \text{defs} by metis

subgoal using a * admissible_transactionE(2) unfolding \text{defs} \text{Tatts[symmetric]} by fastforce

subgoal using a * admissible_transactionE(1) unfolding \text{defs} \text{Tatts[symmetric]} by metis

subgoal using * admissible_transactionE(3) by fast

subgoal using * admissible_transactionE(2,3) by (cases "Tatt1 \notin set P") (force, fastforce)

subgoal using * admissible_transactionE(2,3) by (cases "Tatt1 \notin set P") (force, fastforce)

subgoal

using a * admissible_transaction_is_wellformed_transaction(5)


admissible_transaction_occurs_checksE6

unfolding \text{defs} by metis

subgoal

using a * admissible_transaction_is_wellformed_transaction(5)

admissible_transaction_occurs_checksE5

unfolding \text{defs} by metis

subgoal

using a * admissible_transaction_no_bvars(2)
unfoldingdefsTatts[symmetric]byfastforce
subgoal
usinga*admissible_transaction_is_wellformed_transaction(3)
unfoldingdefsTatts[symmetric]admissible_transaction_updates_defbyfastforce
done

haveTatt_lts_case:
"∃ϑ. wt subst ϑ ∧ inj_on ϑ (fv s) ∧ ϑ ` fv s ⊆ range Var ∧
?Tatt' [(n, receive([s · ϑ]))] [] ∈ set P"
when s: "fv s = {}" "Tatt_lts s ∈ set P" for s
 proof -
 have "wt subst Var" "inj_on Var (fv s)" "Var ` fv s ⊆ range Var" "s · Var = s"
 using s(1) by simp_all
 thus ?thesis using s(2) unfolding Tatts by metis
 qed

haveTatt1_case:
"?Tatt' [(⋆, receive([occurs ⟨0: value⟩ v]))]
[(⋆, ⟨⟨0: value⟩, not in (declassset)⟩ s)] ∈ set P"
when "Tatt1 ∈ set P"
using that unfolding Tatts by auto

haveTatt2_case:
"?Tatt' [(n, receive([invfun ⟨⟨privsec⟩ c, ⟨0: value⟩ v⟩ t])]
[(⋆, ⟨⟨0: value⟩, not in (declassset)⟩ s)] ∈ set P"
when "Tatt2 ∈ set P"
using that unfolding Tatts by auto

note 3 = pair_def case_prod_conv
note 4 = welltyped_leakage_free_priv_const[I[OF 0(1-3) 2(1,2) 1(2,3,6,7)]
note 5 = welltyped_leakage_free_setop_pair[I[OF 2(1,6) 1(4) 2(4,5,3), unfolded 3]
welltyped_leakage_free_set_const[I(2)[OF 2(1) 1(4) 2(7) 1(2,3), unfolded 3]
welltyped_leakage_free_pub_const[I
4(2)
welltyped_leakage_free_occurssec_const[I[OF 2(1,8-10) 1(2,3)]
welltyped_leakage_free_value_const[I[OF 2(1,3,5,11) 1(1)]
welltyped_leakage_free_short_term_secret'I[
OF 2(1,12) 4(1) P wt_secure Tatt2_case 1(8)]
welltyped_leakage_free_long_term_secret[I[OF P wt_secure Tatt_lts_case]
welltyped_leakage_free_short_term_secret[I
OF * 1(3) 4(1) P wt_secure Tatt1_case 1(8)]
welltyped_leakage_free_occurs_fact[I[OF * 1(5)]

** ****

have 6: "is_Fun s ∧ length (args s) = 2 ↔ (∃ f t u. s = Fun f [t, u])"
for s::"('fun,'atom,'sets,'lbl) prot_term"
by auto

define pubconst_cond where
"pubconst_cond ≡ λs::('fun,'atom,'sets,'lbl) prot_term.
is_Fun s ∧ args s = [] ∧ is_Fu (the_Fun s) ∧
publicy (the_Fu (the_Fun s)) ∧ arity_f (the_Fu (the_Fun s)) = 0"

define valuevar_cond where
"valuevar_cond ≡ λs::('fun,'atom,'sets,'lbl) prot_term.
is_Var s ∧ fst (the_Var s) = TAtom Value"

define setconst_cond where
"setconst_cond ≡ λs::('fun,'atom,'sets,'lbl) prot_term.
is_Fun s ∧ args s = [] ∧ is_Set (the_Fun s) ∧ arity, (the_Set (the_Fun s)) = 0"
3.6 Stateful Protocol Verification

define setop_pair_cond where
\[ setop_pair_cond \equiv \lambda s. \text{is\_Fun } s \land \text{the\_Fun } s = \text{Pair} \land \text{length } (\text{args } s) = 2 \land \text{setconst\_cond } (\text{args } s ! 1) \]

define occursfact_cond where
\[ \text{occursfact\_cond } \equiv \lambda s::('\text{fun},'\text{atom},'\text{sets},'\text{lbl}) \text{ prot\_term}. \text{is\_Fun } s \land \text{the\_Fun } s = \text{OccursFact} \land \text{length } (\text{args } s) = 2 \land \text{args } s ! 0 = \text{Fun OccursSec } [] \]

define invkey_cond where
\[ \text{invkey\_cond } \equiv \lambda s. \text{is\_Fun } s \land \text{the\_Fun } s = \text{Fu invfun} \land \text{length } (\text{args } s) = 2 \land \text{args } s ! 0 = \langle \text{privsec} \rangle c \land \text{valuevar\_cond } (\text{args } s ! 1) \]

define ground_lts_cond where
\[ \text{ground\_lts\_cond } \equiv \lambda s. \text{is\_Fun } s \land \text{is\_Fu } (\text{the\_Fun } s) \land \text{fv } s = {} \land \text{Tatt\_lts } s \in \text{set } P \]

note cond_defs =
\[ \text{pubconst\_cond_def valuevar\_cond_def setconst\_cond_def setop\_pair\_cond_def occursfact\_cond_def invkey\_cond_def ground\_lts\_cond_def} \]

have "(\exists m. s = \langle m: \text{value} \rangle v) \leftrightarrow \text{valuevar\_cond } s"
\[ (\exists x c. \text{arity } s c = 0 \land s = \text{Fun Pair } [x, \langle c \rangle s]) \leftrightarrow \text{setop\_pair\_cond } s \]
\[ (\exists c. \text{public } c \land \text{arity } f c = 0 \land s = \langle c \rangle s) \leftrightarrow \text{pubconst\_cond } s \]
\[ (\exists x. s = \text{occurs } x) \leftrightarrow \text{occursfact\_cond } s \]
\[ (\exists x. s = (\text{invfun } [(\text{privsec}) c, (x: \text{value} )])v) \leftrightarrow \text{invkey\_cond } s \]
\[ (\exists f ts s = (f ts), \land \text{fv } s = {} \land \text{Tatt\_lts } s \in \text{set } P) \leftrightarrow \text{ground\_lts\_cond } s \]

for s::"('fun,'atom,'sets,'lbl) prot\_term"
unfolding is\_Set\_def the\_Set\_def is\_Fu\_def cond\_defs
by (fastforce,use 6[of s] in fastforce,fastforce,force,fastforce,fastforce,fastforce)

moreover have "(\forall s \in \text{set } S. \text{valuevar\_cond } s \lor \text{setop\_pair\_cond } s \lor \text{setconst\_cond } s \lor \text{pubconst\_cond } s \lor s = \langle \text{privsec} \rangle c \lor s = \text{Fun OccursSec } [] \lor \text{occursfact\_cond } s \lor \text{invkey\_cond } s \lor \text{ground\_lts\_cond } s) \land
(\neg \text{public } f \text{privsec} \land \text{arity } f \text{privsec} = 0 \land \text{\Gamma } f \text{privsec} \neq \text{None})"

using a unfolding defs' cond\_defs Tatts by meson

ultimately have 7:
"(\forall s \in \text{set } S. (\exists x c. \text{arity } c = 0 \land s = \text{Fun Pair } [x, \langle c \rangle s]) \lor
(\exists c. \text{arity } c = 0 \land s = \langle c \rangle s) \lor
(\exists c. \text{public } c \land \text{arity } f c = 0 \land s = \langle c \rangle s) \lor
s = (\text{privsec}) c \lor s = \text{Fun OccursSec } [] \lor
(\exists m. s = \langle m: \text{value} \rangle v) \lor
(\exists x. s = \text{occurs } x) \lor
(\exists x. s = (\text{invfun } [(\text{privsec}) c, (x: \text{value} )])v) \lor
(\exists f ts s = (f ts), \land \text{fv } s = {} \land \text{Tatt\_lts } s \in \text{set } P)"

unfolding Let\_def by fastforce

show ?thesis
by (rule iffD2[OF welltyped\_leakage\_free\_protocol\_pointwise]; unfold list\_all\_iff; intro ballI)
(use bspec[OF 7] 5 in blast)
 qed

end

locale composable\_stateful\_protocols =
\[ pm: \text{stateful\_protocol\_model } \text{arity}, \text{arity}, \text{public}, \text{Ana}, \text{\Gamma }, \text{label\_witness1}, \text{label\_witness2}\]
for \text{arity},":::'fun \Rightarrow \text{nat}"
and \text{arity},":::\text{sets } \Rightarrow \text{nat}"

293
and public_f::"'fun ⇒ bool"
and Ana_f::"'fun ⇒ (('fun,'atom::finite,'sets,nat) prot_fun, nat) term list × nat list)"
and Γ_f::"'fun ⇒ 'atom option"
and label_witness::"nat"
and label_witness2::"nat"
+
fixes Pc::"('fun,'atom,'sets,nat) prot_transaction list"
and Ps Ps_with_star_projs::"('fun,'atom,'sets,nat) prot_transaction list list"
and Ps_SMP Sec_symbolic::"('fun,'atom,'sets,nat) prot_term list"
and Ps_GSMPs::"(nat × ('fun,'atom,'sets,nat) prot_term list) list"
assumes Pc_def: "Pc = concat Ps"
and Ps_with_star_projs_def: "let Pc' = Pc; L = [0..<length Ps] in
Ps_with_star_projs = map (λn. (map (transaction_proj n) Pc')) L ∧
set L = set (remdups (concat (map (λT. map (the_LabelN ◦ fst)
(filter (Not ◦ has_LabelS) (transaction_strand T)))
Pc'))))"
and Pc_wellformed_composable:
"list_all (list_all (Not ◦ has_LabelS) ◦ tl ◦ transaction_send) Pc"
"pm.wellformed_composable_protocols Ps Pc_SMP"
"pm.composable_protocols Ps Ps_GSMPs Sec_symbolic"

begin

theorem composed_protocol_preserves_component_goals:
assumes components_leakage_free:
"list_all (pm.welltyped_leakage_free_protocol Sec_symbolic) Ps_with_star_projs"
and n_def: "n < length Ps_with_star_projs"
and P_def: "P = Ps_with_star_projs ! n"
and P_welltyped_secure:
"∀A ∈ pm.reachable_constraints P. 不存在 I.
pm.welltyped_constraint_model I (A0[[n, send[attack[ln n]]]])"
shows "∀A ∈ pm.reachable_constraints Pc. 不存在 I.
pm.constraint_model I (A0[[n, send[attack[ln n]]]])"
proof -

note 0 = Ps_with_star_projs_def[unfolded Let_def]

have 1:
"set Ps_with_star_projs =
(λn. map (transaction_proj n) Pc) ◦
set (remdups (concat (map (λT. map (the_LabelN ◦ fst)
(filter (Not ◦ has_LabelS) (transaction_strand T)))
Pc')))"
by (metis (mono_tags, lifting) 0 image_set)

have 2: "Ps_with_star_projs ! n = map (transaction_proj n) Pc"
using conjunct1[OF 0] n_def by fastforce

show ?thesis
by (rule pm.composable_protocols_par_comp Prot'
[ OF Pc_def 1 Pc_wellformed_composable
components_leakage_free 2 P_welltyped_secure[unfolded P_def]])

qed

end

end

294
4 Trac Support and Automation

4.1 Useful Eisbach Methods for Automating Protocol Verification

theory Eisbach_Protocol_Verification
  imports Stateful_Protocol_Composition_and_Typing.Stateful_Compositionality
      "HOL-Eisbach.Eisbach_Tools"
begin

named_theorems exhausts
named_theorems type_class_instance_lemmata
named_theorems protocol_checks
named_theorems protocol_checks'
named_theorems coverage_check_unfold_protocol_lemma
named_theorems coverage_check_unfold_transaction_lemma
named_theorems coverage_check_unfold_lemmata
named_theorems protocol_check_intro_lemmata
named_theorems transaction_coverage_lemmata

method UNIV_lemma =
  (rule UNIV_eq_I; (subst insert_iff)+; subst empty_iff; smt exhausts)+

method type_class_instance =
  (intro_classes; auto simp add: type_class_instance_lemmata)

method protocol_model_subgoal =
  (((rule allI, case_tac f); (erule forw_subst)+)?; simp_all)

method protocol_model_interpretation =
  (unfold_locales; protocol_model_subgoal+)

method composable_protocols_intro =
  (unfold protocol_checks' Let_def;
    intro comp_par_comp I';
    simp only: list.map(1,2) prod.sel(1)?;
    (intro list_set_ballI)?;
    simp only: if_P if_not_P)\)

method coverage_check_intro =
  (((unfold coverage_check_unfold_protocol_lemma);
    intro protocol_check_intro_lemmata;
    simp only: list_all_simps list_all_append list.map concat.simps map_append product_concat_map;
    intro conjI TrueI);
    clarsimp?;
    (intro conjI TrueI)?;
    (rule transaction_coverage_lemmata))\)

method coverage_check_unfold =
  (unfold coverage_check_unfold_lemmata
    Let_def case_prod_unfold Product_Type.fst_conv Product_Type.snd_conv;
    simp only: list_all_simps;
    intro conjI TrueI)

method coverage_check_intro' =
  (((unfold coverage_check_unfold_protocol_lemma coverage_check_unfold_transaction_lemma);
    intro protocol_check_intro_lemmata;
    simp only: list_all_simps list_all_append list.map concat.simps map_append product_concat_map;
4 Trac Support and Automation

4.2 ML Yacc Library

theory "ml_yacc_lib"
imports Main
begin
ML_file "ml-yacc-lib/base.sig"
ML_file "ml-yacc-lib/join.sml"
ML_file "ml-yacc-lib/lrtable.sml"
ML_file "ml-yacc-lib/stream.sml"
ML_file "ml-yacc-lib/parser2.sml"
end

4.3 Abstract Syntax for Trac Terms

theory trac_term
imports "First_Order_Terms.Term" "ml_yacc_lib"
begin
ML<
structure Trac_Utils =
struct
val valN = "val"
val timpliesN = "timplies"
val occursN = "occurs"
val enumN = "enum"
end
4.3 Abstract Syntax for Trac Terms

val enum_trac_typeN = "enum"
val value_trac_typeN = "value"
val priv_fun_secN = "PrivFunSec"
val secret_typeN = "SecretType"
val enum_typeN = "EnumType"
val other_pubconsts_typeN = "PubConstType"

val default_extra_types = [enum_typeN, secret_typeN]
val extended_extra_types = default_extra_types@other_pubconsts_typeN
val all_special_types = value_trac_typeN::enum_trac_typeN::extended_extra_types
val special_funs = ["occurs", "zero", valN, priv_fun_secN]

fun infenumN e = enumN"_"^e

fun list_find p ts = let
  fun aux _ [] = NONE
  | aux n (t::ts) = if p t then SOME (t,n) else aux (n+1) ts
in aux 0 ts end

fun map_prod f (a,b) = (f a, f b)

fun list_product [] = [[]]
| list_product (xs::xss) = List.concat (map (fn x => map (fn ys => x::ys) (list_product xss)) xs)

fun list_triangle_product _ [] = []
| list_triangle_product f (x::xs) = map (f x) xs@list_triangle_product f xs

fun list_subseqs [] = [[]]
| list_subseqs (x::xs) = let val xss = list_subseqs xs in map (cons x) xss@xss end

fun list_intersect xs ys = List.exists (fn x => member (op =) ys x) xs orelse List.exists (fn y => member (op =) xs y) ys

fun list_partitions xs constrs = let
  val peq = eq_set (op =)
  val pseq = eq_set peq
  val psseq = eq_set pseq
  fun illegal p q = let
    val pq = union (op =) p q
    fun f (a,b) = member (op =) pq a andalso member (op =) pq b
  in List.exists f constrs end
  fun merges _ [] = []
  | merges q (p::ps) = if illegal p q then map (cons p) (merges q ps) else (union (op =) p q::ps)::(map (cons p) (merges q ps))
  fun merges_all [] = []
  | merges_all (p::ps) = merges p ps@map (cons p) (merges_all ps)
fun step pss = fold (union pseq) (map merges_all pss) []

fun loop pss pssprev = 
  let val pss' = step pss 
  in if pseq (pss,pss') then pssprev else loop pss' (union pseq pss' pssprev) end

val init = [map single xs]
in loop init init end

fun list_rm_pair sel l x = filter (fn e => sel e <> x) l

fun list_minus list_rm l m = List.foldl (fn (a,b) => list_rm b a) l m

fun list_upto n = 
  let fun aux m = if m >= n then [] else m::aux (m+1) in 
  aux 0 
  end
end

ML

structure Trac_Term (* : TRAC_TERM *) =
struct
  open Trac_Utils
  exception TypeError

  type TypeDecl = string * string

datatype MsgType = TAtom of string
                  | TComp of string * MsgType list

datatype Msg = Var of string
              | Const of string
              | Fun of string * Msg list
              | Abbrev of string * Msg list
              | Attack

  (* TODO: maybe add a set-type *)
datatype cType = Enumeration of string
               | InfiniteEnumeration of string
               | EnumType
               | ValueType
               | PrivFunSecType
               | AtomicType of string
               | ComposedType of string * cType list
               | Untyped

datatype cMsg = cVar of string * cType
               | cConst of string
               | cFun of string * cMsg list
               | cAttack
               | cSet of string * cMsg list
               | cAbs of (string * string list) list
               | cOccursFact of cMsg
               | cPrivFunSec
               | cEnum of string

298
fun MsgType_str (TAtom a) = a 
| MsgType_str (TComp (f,ts)) = f ^ "(" ^ String.concatWith "," (map MsgType_str ts) ^ ")"

fun Msg_str (Var x) = x 
| Msg_str (Const x) = x 
| Msg_str (Fun (f,ps)) = 
  if ps = [] then f else f ^ "(" ^ String.concatWith "," (map Msg_str ps) ^ ")"
| Msg_str (Abbrev (f,ps)) = 
  if ps = [] then f else f ^ "![" ^ String.concatWith "," (map Msg_str ps) ^ "]"
| Msg_str Attack = "attack"

fun msg_vars t = 
  let fun f (Var x) = [x] 
    | f (Fun (_,ps)) = List.concat (map f ps) 
    | f (Abbrev (_,ps)) = List.concat (map f ps) 
    | f (Const _) = [] 
    | f Attack = [] 
  in distinct (op =) (f t) 
  end

fun cType_str (Enumeration e) = e 
| cType_str (InfiniteEnumeration e) = e 
| cType_str EnumType = enum_trac_typeN 
| cType_str ValueType = value_trac_typeN 
| cType_str PrivFunSecType = secret_typeN 
| cType_str AtomicType a = a 
| cType_str (ComposedType (f,ts)) = f ^ "(" ^ String.concatWith "," (map cType_str ts) ^ ")"
| cType_str Untyped = "untyped"

fun cMsg_str' notypes (cVar (x,tau)) = x ^ (if notypes then "" else ":" ^ cType_str tau) 
| cMsg_str' _ (cConst s) = s 
| cMsg_str' notypes (cFun (f,ts)) = 
  f ^ "(" ^ String.concatWith "," (map (cMsg_str' notypes) ts) ^ ")"
| cMsg_str' _ cAttack = "attack"
| cMsg_str' notypes (cSet (s,ts)) = 
  s ^ "(" ^ String.concatWith "," (map (cMsg_str' notypes) ts) ^ ")"
| cMsg_str' _ (cAbs bs) = valN ^ "(" ^ String.concatWith "," 
  (map (fn (c,cs) => c ^ "(" ^ String.concatWith "," cs ^ ")") bs) ^ ")"
| cMsg_str' notypes (cOccursFact t) = occursN ^ "(" ^ cMsg_str' notypes t ^ ")"
| cMsg_str' cPrivFunSec = priv_fun_secN 
| cMsg_str' _ (cEnum e) = e

val cMsg_str = cMsg_str' false

fun subst_apply_cMsg' (delta:(string * cType) -> cMsg) (t:cMsg) = 
  case t of 
    cVar x => delta x 
| cFun (f,ts) => cFun (f, map (subst_apply_cMsg' delta) ts) 
| cSet (s,ts) => cSet (s, map (subst_apply_cMsg' delta) ts) 
| cOccursFact bs => cOccursFact (subst_apply_cMsg' delta bs) 
| c => c

fun subst_apply_cMsg (delta:(string * cMsg) list) = 
  subst_apply_cMsg' (fn (n,tau) => 
    case List.find (fn x => fst x = n) delta of 
    SOME x => snd x 
  | NONE => cVar (n,tau)))

fun subst_apply_Msg d (Var x) = 
  case List.find (fn (y,_) => x = y) d of 
  SOME (_,t) => t
fun certifyMsgType' finite_enums infinite_enums (TAtom a) =
if a = enum_trac_typeN then EnumType
else if a = value_trac_typeN then ValueType
else if List.exists (fn b => a = b) finite_enums then Enumeration a
else if List.exists (fn b => a = b) infinite_enums then InfiniteEnumeration a
else AtomicType a

fun certifyMsgType' finite_enums infinite_enums (TComp (f,ts)) =
ComposedType (f,map (certifyMsgType' finite_enums infinite_enums) ts)

fun certifyMsgType ((finite_enums:string list),
(infinite_enums:string list),
(decls:(string * MsgType) list),
(fresh:(string * MsgType option) list)) n =
case List.find (fn (v,_) => v = n) decls of
SOME (_,tau) => certifyMsgType' finite_enums infinite_enums tau
NONE => (case List.find (fn (v,_) => v = n) fresh of
SOME (_,NONE) => ValueType
SOME (_,SOME tau) => certifyMsgType' finite_enums infinite_enums tau
NONE => error ("Error: Missing or invalid type annotation for variable " ^ n))

fun certifyMsg' notypes params (Var n) =
if notypes then cVar (n, Untyped) else cVar (n, certifyMsgType params n)
cConst c

fun certifyMsg' notypes params (Const c) = cConst c

fun certifyMsg' notypes params (Fun (f,ts)) =
cFun (f, map (certifyMsg' notypes params) ts)
cAttack

fun mk_Value_cVar x = cVar (x,ValueType)

val certifyMsg = certifyMsg' false
val certifyMsgUntyped = certifyMsg' true ([], [], [], [])

fun mk_Value_cVar x = cVar (x,ValueType)

val certifyMsg = certifyMsg' false
val certifyMsgUntyped = certifyMsg' true ([], [], [], [])

fun mk_Value_cVar x = cVar (x,ValueType)

val certifyMsg = certifyMsg' false
val certifyMsgUntyped = certifyMsg' true ([], [], [], [])

fun mk_Value_cVar x = cVar (x,ValueType)

val certifyMsg = certifyMsg' false
val certifyMsgUntyped = certifyMsg' true ([], [], [], [])

fun mk_Value_cVar x = cVar (x,ValueType)
ML

structure TracProtocol (∗ : TRAC_TERM ∗) =
  struct
  open Trac_Utils Trac_Term
  datatype enum_spec_elem =
    Consts of string list
  | Union of string list
  | InfiniteSet
  fun is_Consts t = case t of Consts _ => true | _ => false
  fun the_Consts t = case t of Consts cs => cs | _ => error "Consts"
  type type_spec = string list
  type enum_spec = (string * enum_spec_elem) list
  type set_spec_elem = (string * int * bool)
  type set_spec = set_spec_elem list
  fun extract_Consts (spec:enum_spec) =
    (List.concat o map the_Consts o filter is_Consts o map snd) spec
  type funT = (string * int * MsgType option)
  type fun_spec = {private: funT list, public: funT list}
  type ruleT = (string * string list) * Msg list * string list
  type anaT = ruleT list
  datatype prot_label = LabelN | LabelS
  type Bvars = (string * MsgType) list
  datatype Negcheck = INEQ of Msg * Msg
  | NOTIN of Msg * (string * Msg list)
  datatype action = RECEIVE of Msg list
  | SEND of Msg list
  | EQUATION of Msg * Msg
  | LETBINDING of Msg * Msg
  | IN of Msg * (string * Msg list)
  | NOTINANY of Msg * string
  | NEGCHECKS of Bvars * Negcheck list
  | INSERT of Msg * (string * Msg list)
  | DELETE of Msg * (string * Msg list)
  | NEW of (string * MsgType option) list
  | ATTACK
  datatype labeled_action =
    LABELED_ACTION of prot_label * action
  | ABBREVIATION of string * Msg list
  type transaction_name = string * (string * MsgType) list * (string * string) list
  type transaction={transaction:transaction_name,actions:labeled_action list}
  val action_str =
    let
      fun new_param_str (x,NONE) = x
      | new_param_str (x,SOME tau) = x ^ " : " ^ MsgType_str tau
      fun bvar_str (x,tau) = x ^ " : " ^ MsgType_str tau
      fun set_action_str (t,(s,ps)) pre mid =
pre ^ Ms_str t ^ mid ^ s ^ ( 
  if ps = [] then "" else "(" ^ String.concatWith "," (map Ms_str ps) ^ ")")
fun negcheck_str (INEQ (t,t')) = Ms_str t ^ " != " ^ Ms_str t'
  | negcheck_str (NOTIN p) = set_action_str p " notin "
fun to_str (SEND ts) = "send " ^ String.concatWith ", " (map Ms_str ts)
  | to_str (RECEIVE ts) = "receive " ^ String.concatWith ", " (map Ms_str ts)
  | to_str (LETBINDING (t,t')) = "let " ^ Ms_str t ^ " = " ^ Ms_str t'
  | to_str (EQUATION (t,t')) = Ms_str t ^ " == " ^ Ms_str t'
  | to_str (IN p) = set_action_str p " in "
  | to_str (NOTINANY (t,s)) = set_action_str (t,(s,[])) " notin " ^ "(_)
  | to_str (NEGCHECKS (bvars,ns)) = String.concatWith " or " (map negcheck_str ns) ^
    (if null bvars then "" else "forall ") ^ String.concatWith ", " (map bvar_str bvars)
  | to_str (INSERT p) = set_action_str p "insert " ^ ""
  | to_str (DELETE p) = set_action_str p "delete " ^ ""
  | to_str (NEW xs) = "new " ^ String.concatWith ", " (map new_param_str xs)
  | to_str ATTACK = "attack"
in
  to_str end

fun labeled_action_str (LABELED_ACTION (lbl,act)) =
  (case lbl of LabelN => " " | LabelS => "* ") ^ action_str act
  | labeled_action_str (ABBREVIATION (f,ts)) = f ^ "![^ String.concatWith "," (map Ms_str ts) ^ "]"
fun action_fvs (RECEIVE ts) = distinct (op =) (List.concat (map msg_vars ts))
  | action_fvs (LETBINDING (t,t')) = distinct (op =) (msg_vars t@msg_vars t')
  | action_fvs (EQUATION (t,t')) = distinct (op =) (msg_vars t@msg_vars t')
  | action_fvs (IN (t,(_,p))) = distinct (op =) (msg_vars t@List.concat (map msg_vars p))
  | action_fvs (NOTINANY (t,_,)) = msg_vars t
  | action_fvs (NEGCHECKS (bvars,ns)) =
    let
      fun f (INEQ (t,t')) = msg_vars t@msg_vars t'
        | f (NOTIN (t,(_,p))) = msg_vars t@List.concat (map msg_vars p)
      in
        filter_out (member (op =) (map fst bvars)) (distinct (op =) (List.concat (map f ns)))
    end
  | action_fvs (NEW xs) = distinct (op =) (map fst xs)
  | action_fvs (INSERT (t,(_,p))) = distinct (op =) (msg_vars t@List.concat (map msg_vars p))
  | action_fvs (DELETE (t,(_,p))) = distinct (op =) (msg_vars t@List.concat (map msg_vars p))
  | action_fvs (SEND ts) = distinct (op =) (List.concat (map msg_vars ts))
  | action_fvs ATTACK = []

fun mkTransaction transaction actions = {transaction=transaction,
  actions=actions}:transaction

fun is_RECEIVE a = case a of RECEIVE _ => true | _ => false
fun is_SEND a = case a of SEND _ => true | _ => false
fun is_LETBINDING a = case a of LETBINDING _ => true | _ => false
fun is_EQUATION a = case a of EQUATION _ => true | _ => false
fun is_IN a = case a of IN _ => true | _ => false
fun is_NEGCHECKS a = case a of NEGCHECKS _ => true | _ => false
fun is_NOTINANY a = case a of NOTINANY _ => true | _ => false
fun is_INSERT a = case a of INSERT _ => true | _ => false
fun is_DELETE a = case a of DELETE _ => true | _ => false
fun is_NEW a = case a of NEW _ => true | _ => false
fun is_ATTACK a = case a of ATTACK => true | _ => false

fun the_RECEIVE a = case a of RECEIVE t => t | _ => error "RECEIVE"
fun the_SEND a = case a of SEND t => t | _ => error "SEND"
fun the_LETBINDING a = case a of LETBINDING t => t | _ => error "LETBINDING"
fun the_EQUATION a = case a of EQUATION t => t | _ => error "EQUATION"
fun the_IN a = case a of IN t => t | _ => error "IN"
fun the_NEGCHECKS a = case a of NEGCHECKS t => t | _ => error "NEGCHECKS"
fun the_NOTINANY a = case a of NOTINANY t => t | _ => error "NOTINANY"
fun the_INSERT a = case a of INSERT t => t | _ => error "INSERT"
fun the_DELETE a = case a of DELETE t => t | _ => error "DELETE"
fun the_NEW a = case a of NEW t => t | _ => error "FRESH"

fun maybe_the_RECEIVE a = case a of RECEIVE t => SOME t | _ => NONE
fun maybe_the_SEND a = case a of SEND t => SOME t | _ => NONE
fun maybe_the_LETBINDING a = case a of LETBINDING t => SOME t | _ => NONE
fun maybe_the_EQUATION a = case a of EQUATION t => SOME t | _ => NONE
fun maybe_the_IN a = case a of IN t => SOME t | _ => NONE
fun maybe_the_NEGCHECKS a = case a of NEGCHECKS t => SOME t | _ => NONE
fun maybe_the_NOTINANY a = case a of NOTINANY t => SOME t | _ => NONE
fun maybe_the_INSERT a = case a of INSERT t => SOME t | _ => NONE
fun maybe_the_DELETE a = case a of DELETE t => SOME t | _ => NONE
fun maybe_the_NEW a = case a of NEW t => SOME t | _ => NONE

fun subst_apply_labeled_action d (LABELED_ACTION (lbl,a)) =
  let
    val ap = subst_apply_Msg d
    fun rm_vars_ap ys = subst_apply_Msg (filter (fn (x,_) => List.all (fn (y,_) => x <> y) ys) d)
    fun ap_negcheck xs (INEQ (t,t')) = INEQ (rm_vars_ap xs t, rm_vars_ap xs t')
    fun ap_negcheck xs (NOTIN (t,(f,ts))) = NOTIN (rm_vars_ap xs t,(f,map (rm_vars_ap xs) ts))
    fun aux (RECEIVE ts) = RECEIVE (map ap ts)
    fun aux (SEND ts) = SEND (map ap ts)
    fun aux (EQUATION (t,t')) = EQUATION (ap t, ap t')
    fun aux (LETBINDING (t,t')) = LETBINDING (ap t, ap t')
    fun aux (IN (t,(f,ts))) = IN (ap t,(f,map ap ts))
    fun aux (NOTINANY (t,f)) = NOTINANY (ap t, f)
    fun aux (NEGCHECKS (xs,ns)) = NEGCHECKS (xs,map (ap_negcheck xs) ns)
    fun aux (INSERT (t,(f,ts))) = INSERT (ap t,(f,map ap ts))
    fun aux (DELETE (t,(f,ts))) = DELETE (ap t,(f,map ap ts))
    fun aux (NEW p) = NEW p
    fun aux ATTACK = ATTACK
  in
    LABELED_ACTION (lbl,aux a)
  end

fun expand_term_abbreviations d (ABBREVIATION (f,ts')) = ABBREVIATION (f,map (expand_term_abbreviations d) ts')

fun expand_term_abbreviations Var x = Var x
fun expand_term_abbreviations Const c = Const c
fun expand_term_abbreviations (Fun (f,ts)) = Fun (f,map (expand_term_abbreviations) ts)
fun expand_term_abbreviations (Abbrev (f,ts)) = (case List.find (fn ((g,_),_) => f = g) abbrevs of
  SOME ((_,xs),t) =>
    if length xs <> length ts
    then error ("Error: The number of parameters given to the term abbreviation " ^
                Msg_str (Abbrev(f,ts)) ^ " does not match the number of parameters " ^
                "in its declaration")
    else
      let val delta = xs -- ts
      in expand_term_abbreviations (map (expand_term_abbreviations d) delta t)
      end
  | NONE => error ("Error: Cannot find term abbreviation " ^ f))
fun expand_term_abbreviations ATTACK = ATTACK

fun expand_term_abbreviations_in_action abbrevs ac =
let
val exp = expand_term_abbreviations abbrevs
fun exp_n (INEQ (t,t')) = INEQ (exp t, exp t')
| exp_n (NOTIN (t,(s,ts))) = NOTIN (exp t, (s, map exp ts))
in case ac of
  RECEIVE ts => RECEIVE (map exp ts)
| SEND ts => SEND (map exp ts)
| EQUATION (t,(s,t')) => EQUATION (exp t, exp t')
| IN (t,(s,ts)) => IN (exp t, (s, map exp ts))
| NOTINANY (t,s) => NOTINANY (exp t, s)
| NEGCHECKS (xs,ns) => NEGCHECKS (xs, map exp_n ns)
| INSERT (t,(s,ts)) => INSERT (exp t, (s, map exp ts))
| DELETE (t,(s,ts)) => DELETE (exp t, (s, map exp ts))
| NEW xs => NEW xs
| ATTACK => ATTACK
end

fun expand_action_abbreviations (abbrevs:(string * string list) * labeled_action list) list) =
let
  fun get abbr = case List.find (fn ((a,_),_) => abbr = a) abbrevs of
    SOME ((_,xs),acs) => (xs,acs)
  | NONE => error ("Error: Action sequence abbreviation " ^ abbr ^ " has not been declared")

  fun expand (abbr,ts) =
    let
      val (xs,acs) = get abbr
      val _ = if length xs <> length ts
        then error ("Error: Action sequence abbreviation " ^ abbr ^ " has been applied " ^
          "with the wrong number of parameters: Expected " ^
          Int.toString (length xs) ^ " parameters but got " ^
          Int.toString (length ts))
      else ()
      val delta = xs ~~ ts
      in expand_action_abbreviations abbrevs (map (subst_apply_labeled_action delta) acs) end
    in
    List.concat o
      map (fn a => case a of LABELED_ACTION p => [p]
    | ABBREVIATION p => expand p)
end

datatype abbreviation =
  TermAbbreviation of (string * string list) * Msg
| ActionsAbbreviation of (string * string list) * labeled_action list

type abbreviation_spec = abbreviation list

exception TypeError

304
4.3 Abstract Syntax for Trac Terms

val fun_empty = {
  public=[],
  private=[]
}:fun_spec

fun update_fun_public (fun_spec:fun_spec) public =
  (public = public
  ,private = #private fun_spec
  ):fun_spec

fun update_fun_private (fun_spec:fun_spec) private =
  (public = #public fun_spec
  ,private = private
  ):fun_spec

val empty ={
  name="",
  type_spec=[],
  enum_spec=[],
  set_spec=[],
  function_spec=NONE,
  analysis_spec=[],
  abbreviation_spec=[],
  transaction_spec=[],
  fixed_point=NONE
}:protocol

fun update_name (protocol_spec:protocol) name =
  (name = name
  ,type_spec = #type_spec protocol_spec
  ,enum_spec = #enum_spec protocol_spec
  ,set_spec = #set_spec protocol_spec
  ,function_spec = #function_spec protocol_spec
  ,analysis_spec = #analysis_spec protocol_spec
  ,abbreviation_spec = #abbreviation_spec protocol_spec
  ,transaction_spec = #transaction_spec protocol_spec
  ,fixed_point = #fixed_point protocol_spec
  ):protocol

fun update_sets (protocol_spec:protocol) set_spec =
  (name = #name protocol_spec
  ,type_spec = #type_spec protocol_spec
  ,enum_spec = #enum_spec protocol_spec
  ,set_spec = set_spec
  ,function_spec = #function_spec protocol_spec
  ,analysis_spec = #analysis_spec protocol_spec
  ,abbreviation_spec = #abbreviation_spec protocol_spec
  ,transaction_spec = #transaction_spec protocol_spec
  ,fixed_point = #fixed_point protocol_spec
  ):protocol

fun update_type_spec (protocol_spec:protocol) type_spec =
  (name = #name protocol_spec
  ,type_spec = type_spec
  ,enum_spec = #enum_spec protocol_spec
  ,set_spec = #set_spec protocol_spec
  ,function_spec = #function_spec protocol_spec
  ,analysis_spec = #analysis_spec protocol_spec
  ,abbreviation_spec = #abbreviation_spec protocol_spec
  ,transaction_spec = #transaction_spec protocol_spec
  ,fixed_point = #fixed_point protocol_spec
  ):protocol

fun update_enum_spec (protocol_spec:protocol) enum_spec =
{(name = #name protocol_spec
 ,type_spec = #type_spec protocol_spec
 ,enum_spec = enum_spec
 ,set_spec = #set_spec protocol_spec
 ,function_spec = #function_spec protocol_spec
 ,analysis_spec = #analysis_spec protocol_spec
 ,abbreviation_spec = #abbreviation_spec protocol_spec
 ,transaction_spec = #transaction_spec protocol_spec
 ,fixed_point = #fixed_point protocol_spec
 }):protocol

fun update_functions (protocol_spec:protocol) function_spec =
{(name = #name protocol_spec
 ,type_spec = #type_spec protocol_spec
 ,enum_spec = #enum_spec protocol_spec
 ,set_spec = #set_spec protocol_spec
 ,function_spec = SOME function_spec
 ,analysis_spec = #analysis_spec protocol_spec
 ,abbreviation_spec = #abbreviation_spec protocol_spec
 ,transaction_spec = #transaction_spec protocol_spec
 ,fixed_point = #fixed_point protocol_spec
 }):protocol

fun update_analysis (protocol_spec:protocol) analysis_spec =
{(name = #name protocol_spec
 ,type_spec = #type_spec protocol_spec
 ,enum_spec = #enum_spec protocol_spec
 ,set_spec = #set_spec protocol_spec
 ,function_spec = #function_spec protocol_spec
 ,analysis_spec = analysis规格
 ,abbreviation_spec = #abbreviation_spec protocol_spec
 ,transaction_spec = #transaction_spec protocol_spec
 ,fixed_point = #fixed_point protocol_spec
 }):protocol

fun update_abbreviations (protocol_spec:protocol) abbreviation_spec =
{(name = #name protocol_spec
 ,type_spec = #type_spec protocol_spec
 ,enum_spec = #enum_spec protocol_spec
 ,set_spec = #set_spec protocol_spec
 ,function_spec = #function_spec protocol_spec
 ,analysis_spec = #analysis_spec protocol_spec
 ,abbreviation_spec = abbreviation规格
 ,transaction_spec = #transaction_spec protocol_spec
 ,fixed_point = #fixed_point protocol_spec
 }):protocol

fun update_transactions (prot_name:string option) (protocol_spec:protocol) transaction_spec =
{(name = #name protocol_spec
 ,type_spec = #type_spec protocol_spec
 ,enum_spec = #enum_spec protocol_spec
 ,set_spec = #set_spec protocol_spec
 ,function_spec = #function_spec protocol_spec
 ,analysis_spec = #analysis_spec protocol_spec
 ,abbreviation_spec = #abbreviation_spec protocol_spec
 ,transaction_spec = (prot_name,transaction_spec)::(#transaction_spec protocol_spec)
 ,fixed_point = #fixed_point protocol_spec
 }):protocol

fun update_fixed_point (protocol_spec:protocol) fixed_point =
{(name = #name protocol_spec
 ,type_spec = #type_spec protocol_spec
 ,enum_spec = #enum_spec protocol_spec
 ,set_spec = #set_spec protocol_spec
 ,function_spec = #function_spec protocol_spec
 ,analysis_spec = #analysis_spec protocol_spec
 ,abbreviation_spec = #abbreviation_spec protocol_spec
 ,transaction_spec = #transaction_spec protocol_spec
 ,fixed_point = #fixed_point protocol_spec
 }):protocol
4.3 Abstract Syntax for Trac Terms

```ml
structure TracProtocolCert (* : TRAC_TERM *) =
  struct
    open Trac_Utils Trac_Term TracProtocol
    type cBvars = (string * cType) list
    datatype cNegCheckVariant = cInequality of cMsg * cMsg
                                | cNotInSet of cMsg * cMsg
    datatype cPosCheckVariant = cCheck
                                | cAssignment
    datatype cAction = cReceive of cMsg list
                       | cSend of cMsg list
                       | cEquality of cPosCheckVariant * (cMsg + cMsg)
                       | cInSet of cPosCheckVariant * (cMsg + cMsg)
                       | cNotInAny of cMsg * string
                       | cNegChecks of cBvars * cNegCheckVariant list
                       | cInsert of cMsg * cMsg
                       | cDelete of cMsg * cMsg
                       | cNew of (string * cType) list
                       | cAssertAttack
    type flat_enum_spec = (string * string list * string list) list
    type cFunT = (string * int)
    type cConstsT = (string * string option)
    type cFunSpec = {public_funs: cFunT list, public_consts: cConstsT list,
                     private_consts: cConstsT list}
    type cAnaRule = {head: (string * string list), keys: cMsg list,
                     results: string list, is_priv_fun: bool}
    type cAnaSpec = cAnaRule list
    type cTransaction_name = string * (string * cType) list * (string * string) list
    type cTransaction = {
      transaction:cTransaction_name
                         ,receive_actions:(prot_label * cAction) list
                         ,checkssingle_actions:(prot_label * cAction) list
                         ,checkall_actions:(prot_label * cAction) list
                         ,fresh_actions:(prot_label * cAction) list
                         ,update_actions:(prot_label * cAction) list
                         ,send_actions:(prot_label * cAction) list
                         ,attack_actions:(prot_label * cAction) list
    }
    type cProtocol = {
      name:string
                   ,type_spec:type_spec
                   ,enum_spec:flat_enum_spec
                   ,set_spec:set_spec
                   ,function_spec:cFunSpec option
                   ,analysis_spec:cAnaSpec
                   ,transaction_spec:(string option * cTransaction list) list
                   ,fixed_point: (cMsg list * (string * string list) list list *
                                  ((string * string list) list * (string * string list) list) list) option
    }
```

307
fun is_Receive a = case a of cReceive _ => true | _ => false
fun is_Send a = case a of cSend _ => true | _ => false
fun is_Equality a = case a of cEquality _ => true | _ => false
fun is_InSet a = case a of cInSet _ => true | _ => false
fun is_NegChecks a = case a of cNegChecks _ => true | _ => false
fun is_NotInAny a = case a of cNotInAny _ => true | _ => false
fun is_Insert a = case a of cInsert _ => true | _ => false
fun is_Delete a = case a of cDelete _ => true | _ => false
fun is_Fresh a = case a of cNew _ => true | _ => false
fun is_Attack a = case a of cAssertAttack => true | _ => false
fun is_Inequality a = case a of cInequality _ => true | _ => false
fun is_NotInSet a = case a of cNotInSet _ => true | _ => false

fun the_Receive a = case a of cReceive t => t | _ => error "Receive"
fun the_Send a = case a of cSend t => t | _ => error "Send"
fun the_Equality a = case a of cEquality t => t | _ => error "Equality"
fun the_InSet a = case a of cInSet t => t | _ => error "InSet"
fun the_NegChecks a = case a of cNegChecks t => t | _ => error "NegChecks"
fun the_NotInAny a = case a of cNotInAny t => t | _ => error "NotInAny"
fun the_Insert a = case a of cInsert t => t | _ => error "Insert"
fun the_Delete a = case a of cDelete t => t | _ => error "Delete"
fun the_Fresh a = case a of cNew ts => ts | _ => error "New"
fun the_Inequality a = case a of cInequality p => p | _ => error "Inequality"
fun the_NotInSet a = case a of cNotInSet p => p | _ => error "NotInSet"

fun maybe_the_Receive a = case a of cReceive t => SOME t | _ => NONE
fun maybe_the_Send a = case a of cSend t => SOME t | _ => NONE
fun maybe_the_Equality a = case a of c Equality (_,t) => SOME t | _ => NONE
fun maybe_the_InSet a = case a of cInSet (_,t) => SOME t | _ => NONE
fun maybe_the_NegChecks a = case a of cNegChecks t => SOME t | _ => NONE
fun maybe_the_NotInAny a = case a of cNotInAny t => SOME t | _ => NONE
fun maybe_the_Insert a = case a of cInsert t => SOME t | _ => NONE
fun maybe_the_Delete a = case a of cDelete t => SOME t | _ => NONE
fun maybe_the_Fresh a = case a of cNew ts => SOME ts | _ => NONE
fun maybe_the_Inequality a = case a of cInequality p => SOME p | _ => NONE
fun maybe_the_NotInSet a = case a of cNotInSet p => SOME p | _ => NONE

fun subst_apply_cAction (delta:(string * cMsg) list) (lbl:prot_label,a:cAction) =
  let
    val ap = subst_apply_cMsg
    val apply = ap delta
    fun rm_vars_apply ys = ap (filter (fn (x,_) => List.all (fn (y,_) => x <> y) ys) delta)
    fun rm_vars_apply_pair xs (t,t') = (rm_vars_apply xs t, rm_vars_apply xs t')
    fun apply_negcheck xs (cInequality p) = cInequality (rm_vars_apply_pair xs p)
    fun apply_negcheck xs (cNotInSet p) = cNotInSet (rm_vars_apply_pair xs p)
    in
      case a of
        cReceive ts => (lbl,cReceive (map apply ts))
      | cSend ts => (lbl,cSend (map apply ts))
      | cEquality (v,(t,t')) => (lbl,cEquality (v,(apply t, apply t'))) 
      | cInSet (v,(x,s)) => (lbl,cInSet (v,(apply x, apply s)))
      | cNotInAny (x,s) => (lbl,cNotInAny (apply x, s))
      | cNegChecks (bvars,ps) => (lbl,cNegChecks (bvars,map (apply_negcheck bvars) ps))
      | cInsert (x,s) => (lbl,cInsert (apply x, apply s))
      | cDelete (x,s) => (lbl,cDelete (apply x, apply s))
      | cNew xs => (lbl,cNew xs)
      | cAssertAttack => (lbl,cAssertAttack)
    end

fun subst_apply_cActions delta =
  map (subst_apply_cAction delta)

fun maybe_the_Receive a = case a of cReceive t => SOME t | _ => NONE
fun maybe_the_Send a = case a of cSend t => SOME t | _ => NONE
fun maybe_the_Equality a = case a of cEquality (_,t) => SOME t | _ => NONE
fun maybe_the_InSet a = case a of cInSet (_,t) => SOME t | _ => NONE
fun maybe_the_NegChecks a = case a of cNegChecks t => SOME t | _ => NONE
fun maybe_the_NotInAny a = case a of cNotInAny t => SOME t | _ => NONE
fun maybe_the_Insert a = case a of cInsert t => SOME t | _ => NONE
fun maybe_the_Delete a = case a of cDelete t => SOME t | _ => NONE
fun maybe_the_Fresh a = case a of cNew ts => SOME ts | _ => NONE
fun maybe_the_Inequality a = case a of cInequality p => SOME p | _ => NONE
fun maybe_the_NotInSet a = case a of cNotInSet p => SOME p | _ => NONE
val cAction_str = 
let
val cmsg_str = cMsg_str' true
fun var_str (x,tau) = x ^ " ^ t ^ " ^ cType_str tau
fun set_action_str (t,s) pre mid = pre ^ t ^ mid ^ cmsg_str s
fun negcheck_str (cInequality (t,t')) = t ^ " != " ^ t'
few fun to_str (cSend ts) = "send " ^ String.concatWith " , " (map cmsg_str ts)
bfun to_str (cReceive ts) = "receive " ^ String.concatWith " , " (map cmsg_str ts)
few fun to_str (cEquality (psv,(t,t'))) = 
  case psv of
    cCheck => t ^ " == " ^ t'
| cAssignment => "let " ^ t ^ " = " ^ t'
fun to_str (cInSet (psv,p)) = 
  case psv of
    cCheck => (t, s) ^ " notin " ^ s ^ "(_"
| cAssignment => (t, s) select " in"
fun to_str (cNotInAny (t,s)) = t ^ " notin " ^ s
fun to_str (cNegChecks (bvars,ns)) = String.concatWith " or " (map negcheck_str ns) ^ 
  if null bvars then " else " forall " ^ String.concatWith " , " (map var_str bvars)
few fun to_str (cInsert p) = (t, s) insert " "
| to_str (cDelete p) = set_action_str p "delete " "
| to_str (cNew xs) = "new " ^ String.concatWith " , " (map var_str xs)
| to_str cAssertAttack = "attack"
end
fun cTransaction_str (tr:cTransaction) = 
  let
    fun lbl_act_str (lbl,act) = (case lbl of LabelN => "" | LabelS => "") ^ cAction_str act

fun name_str (name, decls, ineqs) = 
  name ^ "(" ^ String.concatWith " , " (map (fn (x,t) => x ^ " ^ t ^ " ^ cType_str t) decls) ^ "")" ^ 
  (if null ineqs then " else " where ") ^ 
  String.concatWith " , " (map (fn (a,b) => a ^ " != " ^ b) ineqs)
in
  name_str (#transaction tr) ^ "\n" ^ 
  String.concatWith "\n" (map lbl_act_str
    (#receive_actions tr)
    @(#checksingle_actions tr)
    @(#checkall_actions tr)
    @(#fresh_actions tr)
    @(#update_actions tr)
    @(#send_actions tr)
    @(#attack_actions tr)) ^ 
    ","
end

fun is_priv_fun_trac (trac:TracProtocol.protocol) f = 
  let
    val funs = #private (Option.valOf (#function_spec trac))
in
    List.exists (fn (g,n,_) => f = g andalso n <> 0) funs
end

fun get_enum_consts_trac (trac:TracProtocol.protocol) = 
  distinct (op =) (TracProtocol.extract_Consts (#enum_spec trac))

fun flatten_enum_spec_trac (trac:TracProtocol.protocol) = 
  let
    open TracProtocol
    fun step taus (a,e) = 
      case e of
        Union es =>
let
  fun f e = case List.find (fn (a,_) => e = a) taus of
    SOME (_,Union es') =>
      let
        val _ = if List.exists (fn a => e = a) es'
          then error ("Error: There is a cyclic dependency for " ^
            "enumeration " ^ e)
          else ()
      in es' end
    | SOME _ => [e]
    | NONE => error ("Error: Enumeration " ^ e ^ " has not been declared")
    in
      (s,Union (distinct (op =) (List.concat (map f es))))
  end
in c => (a,c)
fun loop taus =
  let
    val taus' = map (step taus) taus
  in
    if taus = taus'
      then taus
      else loop taus'
  end
fun postproc _ (e,InfiniteSet) = (e,[],[e])
| postproc _ (e,Consts cs) = (e,distinct (op =) cs,[]) 
| postproc spec (e,Union es) =
    let
      fun get e' = case List.find (fn (x,_) => x = e') spec of
          SOME p => p
        | NONE => error ("Error: Enumeration " ^ e ^ " has not been declared")
      fun ins (_,Consts cs) (fes,ies) = (distinct (op =) (fes@cs),ies)
        | ins (e',InfiniteSet) (fes,ies) = (fes,distinct (op =) (ies@[e']))
        | ins _ _ = error "Error: Couldn't flatten the enumerations"
      val (fes,ies) = fold (ins o get) es ([],[])
      in (e,fes,ies) end
    in (e,fes,ies) end
  fun flat_enum_spec = loop (#enum_spec trac)
  in map (postproc flat_enum_spec) flat_enum_spec
  end
4.3 Abstract Syntax for Trac Terms

fun transform_cMsg trac =
let
  open Trac_Term

  fun conv_enum_consts trac (t:cMsg) =
    let
      val enums = get_enum_consts_trac trac
      fun aux (cFun (f,ts)) =
        if List.exists (fn x => x = f) enums
        then if null ts
        then cEnum f
        else error ("Error: Enumeration constant " ^ f ^ 
                      " should not have a parameter list")
        else
cFun (f,map aux ts)
in
      fun aux (cConst c) =
        if List.exists (fn x => x = c) enums
        then cEnum c
        else cConst c
      in
      aux t
    end

  fun val_to_abs (t:cMsg) =
    let
      fun aux t = case t of cEnum b => b | _ => error "Error: Invalid val parameter list"
in
    case t of
      cFun (f,ts) =>
        if f = valN
        then cAbs (val_to_abs_list ts)
        else cFun (f,map aux val_to_abs_list ts)
      cSet (s,ts) =>
        cSet (s,map aux val_to_abs_list ts)
      cOccursFact bs =>
        cOccursFact (val_to_abs bs)
      t => t
    end

  fun occurs_enc t =
    let
      fun aux [cVar x] = cVar x
      in
      case t of
        cFun (f,ts) =>
          if f = occursN
          then cAbs (val_to_abs_list ts)
          else cFun (f,map aux val_to_abs_list ts)
        cSet (s,ts) =>
          cSet (s,map val_to_abs_list ts)
        cOccursFact bs =>
          cOccursFact (val_to_abs bs)
        t => t
      end
    end

  fun aux [cVar x] = cVar x
  in
  aux [cAbs bs] = cAbs bs
  aux t = error ("Error: Invalid occurs parameter list: [" ^ 
                  String.concatWith ", " (map cMsg_str ts) ^ "]")
  fun enc (cFun (f,ts)) = ( 
    if f = occursN
    then cOccursFact (aux ts)
    else cFun (f,map aux_enc ts))
  end

Trac Support and Automation

enc (cSet (s,ts)) = cSet (s, map enc ts)

enc (cOccursFact bs) = cOccursFact (enc bs)

enc t = t

enc t

occurs_enc o val_to_abs o conv_enum_consts trac o priv_fun_enc trac

fun certify_fixpoint trac fp =
  let
    open Trac_Term

    fun mk_enum_substs (vars:(string * cType) list) =
      let
        val flat_enum_spec = flatten_finite_enum_spec_trac trac
        val deltas =
          let
            fun f (s,Enumeration tau) =
              case List.find (fn x => fst x = tau) flat_enum_spec of
                SOME x => map (fn c => (s,c)) (snd x)
              | NONE => error ("Error: Enumeration " ^ tau ^ " was not found in the finite enumeration specification")
            | f (s,_) = error ("Error: Variable " ^ s ^ " is not of finite enumeration type")
          in
            list_product (map f vars)
          end
      in
        map (fn d => map (fn (x,t) => (x,cEnum t)) d) deltas
      end

    fun ground_enum_variables (fp:cMsg list) =
      let
        fun do_grounding t = map (fn d => subst_apply_cMsg d t) (mk_enum_substs (fv_cMsg t))
      in
        List.concat (map do_grounding fp)
      end

    fun split_fp (fp:cMsg list) =
      let
        fun fa t = case t of cFun (s,_) => s <> timpliesN | _ => true
        fun fb (t,ts) = case t of cOccursFact (cAbs bs) => bs::ts | _ => ts
        fun fc (cFun (s, [cAbs bs, cAbs cs]),ts) =
          if s = timpliesN
            then (bs,cs)::ts
            else ts
        | fc (_,ts) = ts

        val eq = eq_set (op =)
        fun eq_pairs ((a,b),(c,d)) = eq (a,c) andalso eq (b,d)

        val timplies_trancl =
          let
            fun trans_step ts =
              let
                fun aux (s,t) = map (fn (_,u) => (s,u)) (filter (fn (v,_)) eq pairs (ts@List.concat (map aux ts)))
              in
                distinct eq_pairs (filter (not o eq) (ts@List.concat (map aux ts)))
              end
          in
            aux (s,t)
          end
      in
        list_product (map f vars)
      end

    fun mk_enum_substs (vars:(string * cType) list) =
      let
        val flat_enum_spec = flatten_finite_enum_spec_trac trac
        val deltas =
          let
            fun f (s,Enumeration tau) =
              case List.find (fn x => fst x = tau) flat_enum_spec of
                SOME x => map (fn c => (s,c)) (snd x)
              | NONE => error ("Error: Enumeration " ^ tau ^ " was not found in the finite enumeration specification")
            | f (s,_) = error ("Error: Variable " ^ s ^ " is not of finite enumeration type")
          in
            list_product (map f vars)
          end
      in
        map (fn d => map (fn (x,t) => (x,cEnum t)) d) deltas
      end

    fun ground_enum_variables (fp:cMsg list) =
      let
        fun do_grounding t = map (fn d => subst_apply_cMsg d t) (mk_enum_substs (fv_cMsg t))
      in
        List.concat (map do_grounding fp)
      end

    fun split_fp (fp:cMsg list) =
      let
        fun fa t = case t of cFun (s,_) => s <> timpliesN | _ => true
        fun fb (t,ts) = case t of cOccursFact (cAbs bs) => bs::ts | _ => ts
        fun fc (cFun (s, [cAbs bs, cAbs cs]),ts) =
          if s = timpliesN
            then (bs,cs)::ts
            else ts
        | fc (_,ts) = ts

        val eq = eq_set (op =)
        fun eq_pairs ((a,b),(c,d)) = eq (a,c) andalso eq (b,d)

        val timplies_trancl =
          let
            fun trans_step ts =
              let
                fun aux (s,t) = map (fn (_,u) => (s,u)) (filter (fn (v,_)) eq pairs (ts@List.concat (map aux ts)))
              in
                distinct eq_pairs (filter (not o eq) (ts@List.concat (map aux ts)))
              end
          in
            aux (s,t)
          end
      in
        list_product (map f vars)
      end

end
let
val ts' = trans_step ts
in
if eq_set eq_pairs (ts,ts')
then ts
else loop ts'
end
in
loop
end

val ti = List.foldl fc [] fp
in
(filter fa fp, distinct eq (List.foldl fb [] fp@map snd ti), timplies_trancl ti)
end

fun check_no_vars_and_consts (fp:cMsg list) =
let
fun aux (cVar _) = false
| aux (cConst _) = false
| aux (cFun (_,ts)) = List.all aux ts
| aux (cSet (_,ts)) = List.all aux ts
| aux (cOccursFact bs) = aux bs
| aux _ = true
in
if List.all aux fp
then fp
else error ("There shouldn't be any cVars and cConsts at this point in the " ^
"fixpoint translation")
end

in
fp |> map (fn (m,t) => certifyMsg (map snd t, [], map (fn (a,b) => (a,TAtom b)) t, [])) m
 |> ground_enum_variables
 |> map (transform_cMsg trac)
 |> check_no_vars_and_consts
 |> split_fp
end

fun certifyAction params (lbl,SEND ts) = (lbl,cSend (map (certifyMsg params) ts))
| certifyAction params (lbl,RECEIVE ts) = (lbl,cReceive (map (certifyMsg params) ts))
| certifyAction params (lbl,LETBINDING (t,t')) = (lbl,cEquality (cAssignment, (certifyMsg params t, certifyMsg params t')))
| certifyAction params (lbl,EQUATION (t,t')) = (lbl,cEquality (cCheck, (certifyMsg params t, certifyMsg params t')))
| certifyAction params (lbl,IN (x,(s,ps))) =
let
fun f (Enumeration _) = true
| f (InfiniteEnumeration _) = true
| f EnumType = true
| f ValueType = true
| f _ = false
val taus = distinct (op =) (map (certifyMsgType params) (action_fvs (IN (x,(s,ps))))))
val poscheckvariant = if List.all f taus then cCheck else cAssignment
in
(lbl,cInSet (poscheckvariant, (certifyMsg params x, cSet (s, map (certifyMsg params) ps))))
end
| certifyAction params (lbl,NOTINANY (x,s)) = (lbl,cNotInAny (certifyMsg params x, s))
| certifyAction params (lbl,NEGCHECKS (xs,ns)) = (lbl,cNegChecks (map (fn (x,tau) => (x,certifyMsgType' (#1 params) (#2 params) tau)) xs,
map (fn n => case n of
    INEQ (t,t') => cInequality (certifyMsg params t, certifyMsg params t')
    | NOTIN (t,(s,ps)) => cNotInSet (certifyMsg params t, cSet (s, map (certifyMsg params) ps)))
ns))

| certifyAction params (lbl,INSERT (x,(s,ps))) = (lbl,cInsert (certifyMsg params x, cSet (s, map (certifyMsg params) ps)))
| certifyAction params (lbl,DELETE (x,(s,ps))) = (lbl,cDelete (certifyMsg params x, cSet (s, map (certifyMsg params) ps)))
| certifyAction (fenums,ienums,...) (lbl,NEW xs) = (lbl,cNew (map (fn (x,tau) => case tau of
    NONE => (x,ValueType)
    | SOME (TAtom a) => if a = value_trac_typeN then (x,ValueType)
    else if List.exists (fn e => a = e) (fenums@ienums) then error "Error: The special enum type is not allowed in "new" actions"
    else (x,AtomicType a)
    | SOME (TComp _) => error "Error: Composed type annotations in "new" actions are not allowed"
    xs)
| certifyAction _ (lbl,ATTACK) = (lbl,cAssertAttack)

fun certifyTransactionName fenums ienums ((name, vars, ineqs):transaction_name) = (name, map (fn (x,_) => (x,certifyMsgType (fenums, ienums, vars, []) x)) vars, ineqs)

fun certifyTransaction finite_enumerations infinite_enumerations (tr:transaction) =
  let
    val tr_acs = map (fn a => case a of
      LABELED_ACTION p => p
      | ABBREVIATION p => error ("Error: Got an unexpected action sequence abbreviation " ^
        "(they should have all been expanded and removed at " ^
        "this point): " ^
        labeled_action_str (ABBREVIATION p)))
    (actions tr)
    val mk_cOccurs = cOccursFact
    fun mk_Value_cVar x = cVar (x,ValueType)
    fun mk_cInequality x y = cNegChecks ([],[cInequality (mk_Value_cVar x, mk_Value_cVar y)])
    val mk_cInequalities = list_triangle_product mk_cInequality
    val fresh_vars = List.concat (map_filter (maybe_the_NEW o snd) tr_acs)
    val fresh_vals = map_filter (fn (v,tau) =>
      if tau = NONE orelse tau = SOME (TAtom value_trac_typeN) then SOME v
      else NONE)
    fresh_vars
    val decl_vars = #2 (#transaction tr)
    val decl_vars' = map fst decl_vars
    val neq_constrs = #3 (#transaction tr)
    val bvars = List.concat (map fst (map_filter (maybe_the_NEGCHECKS o snd) tr_acs))
    val = if List.exists (fn x => List.exists (fn (y,_) => x = y) fresh_vars) decl_vars' then error "The fresh and the declared variables must not overlap" else ()
    val = case List.find (fn (x,y) => x = y) neq_constrs of
      SOME (x,y) => error ("Illegal inequality constraint: " ^
        "^ x ^" != "^ y)
4.3 Abstract Syntax for Trac Terms

| NONE => () |

val nonfresh_vals = map fst (filter (fn x => snd x = TAtom value_trac_typeN) (#2 (#transaction tr)))
val enum_vars = map_filter (fn (x,tau) => case tau of TAtom e => if List.exists (fn e' => e = e') (finite_enumerations@infinite_enumerations) then SOME (x,e) else NONE | TComp _ => NONE) (#2 (#transaction tr))
val nonenum_decl_vars = filter (fn (x,_) => not (List.exists (fn (y,_) => x = y) enum_vars)) decl_vars
fun lblS t = (LabelS,t)
val cactions = map (certifyAction (finite_enumerations, infinite_enumerations, decl_vars@bvars, fresh_vars)) tr_acs
val cname = certifyTransactionName finite_enumerations infinite_enumerations (#transaction tr)

fun mk_occurs_step f xs = if null xs then NONE else SOME ((lblS o f o map (mk_cOccurs o mk_Value_cVar)) xs)

fun is_poscheck1 (_,a) = is_Equality a orelse is_InSet a
fun is_check1 p = is_poscheck1 p orelse is_NegChecks (snd p)
val nonfresh_occurs = mk_occurs_step cReceive nonfresh_vals
val receives = filter (is_Receive o snd) cactions
val value_inequalities = map lblS (mk_cInequalities nonfresh_vals)
val checksingles = filter is_check1 cactions
val checkalls = filter (is_NotInAny o snd) cactions
val updates = filter (fn (_,a) => is_Insert a orelse is_Delete a) cactions
val sends = filter (is_Send o snd) cactions
val fresh_occurs = mk_occurs_step cSend fresh_vals
val sends' = filter (is_Send o snd) cactions
val fresh_occurs' = (labelsS o cSend fresh_vals)
val attack_signals = filter (is_Attack o snd) cactions

{|transaction = cname,
receive_actions = case nonfresh_occurs of NONE => receives |
SOME occs => occs::receives,
checksingle_actions = value_inequalities@checksingles,
checkall_actions = checkalls,
fresh_actions = fresh,
update_actions = updates,
send_actions = case fresh_occurs of NONE => sends |
SOME occs => case sends of |
((LabelS, cSend ts)::sends') => (labelsS o cSend) ((the_Send o snd) occs@ts)::sends' |
_ => occs::sends,
attack_actions = attack_signals):cTransaction
end

fun get_finite_enum_spec_trac (trac:protocol) = let
val spec = #enum_spec trac
val finite_enum_spec = let

fun is_finite e =  
  List.exists  
    (fn (s,t) => s = e andalso (case t of  
        TracProtocol.Consts _ => true  
      | TracProtocol.Union ts => List.all is_finite ts  
      | TracProtocol.InfiniteSet => false))  
  spec  
in  
  filter (is_finite o fst) spec  
end  

fun get_infinite_enum_spec_trac (trac:protocol) =  
  filter_out (member (op =)) (get_finite_enum_spec_trac trac)  
  (#enum_spec trac)

fun get_finite_enum_names_trac (trac:protocol) =  
  map fst (get_finite_enum_spec_trac trac)

fun get_infinite_enum_names_trac (trac:protocol) =  
  map fst (get_infinite_enum_spec_trac trac)

fun get_enum_names_trac (trac:protocol) =  
  map fst (get_finite_enum_spec trac)

fun get_funs_trac (trac:protocol) =  
  let  
    fun rm_special_funs sel l = list_minus (list_rm_pair sel) l special_funs  
    fun inc_ar (s,n,tau) = (s, 1+n, tau)  
    in  
    case (#function_spec trac) of  
      NONE => ([],[],[])  
    | SOME ((public=pub, private=priv)) =>  
      let  
        val pub_symbols = rm_special_funs #1 (pub@map inc_ar (filter_funs priv))  
        val pub_funs = filter_funs pub_symbols  
        val pub_consts = filter_consts pub_symbols  
        val priv_consts = append_sec (rm_special_funs #1 (filter_consts priv))  
        in  
        (pub_funs, pub_consts, priv_consts)  
      end  
    end  
end

fun get_term_abbreviations_trac (trac:protocol) =  
  map_filter (fn a => case a of TracProtocol.TermAbbreviation t => SOME t | _ => NONE)  
  (#abbreviation_spec trac)

fun get_action_abbreviations_trac (trac:protocol) =  
  map_filter (fn a => case a of TracProtocol.ActionsAbbreviation t => SOME t | _ => NONE)  
  (#abbreviation_spec trac)

fun check_for_invalid_trac_specification (trac:TracProtocol.protocol) = let  
  open Trac_Term TracProtocol  
  datatype action_status =  
    Passed | InvalidSetParam | WrongPosition | IllformedVars | InvalidAnnotationNewAction |  
    InvalidFunctionSymbols of (string * int) list |  
    InvalidSetSymbols of (string * int option) list
4.3 Abstract Syntax for Trac Terms

val has_dups = has_duplicates (op =)
val dups_str = String.concatWith "", " o duplicates (op =)

val expand_abbrevs =
  expand_action_abbreviations (get_action_abbreviations_trac trac)

val enumerations = get_enum_names_trac trac
val finite_enumerations = get_finite_enum_names_trac trac
val infinite_enumerations = get_infinite_enum_names_trac trac
val set_names = map #1 (#set_spec trac)
val set_spec = map (fn (s,n,_) => (s,n)) (#set_spec trac)
val enum_consts = get_enum_consts_trac trac
val fun_names = case #function_spec trac of
  SOME fs => map #1 ((#public fs)@(#private fs))
| NONE => []
val fun_spec = case #function_spec trac of
  SOME fs => map_filter
    (fn (s,n,tau) => if n > 0 andalso tau = NONE then SOME (s,n) else NONE)
    ((#public fs)@(#private fs))
| NONE => []

val ana_funs = map (#1 o #1) (#analysis_spec trac)
val ana_args = map (#2 o #1) (#analysis_spec trac)
val ana_has_illegal_var_in_body = not o
  (fn ((_,xs),ts,ys) => subset (op =) (ys@List.concat (map Trac_Term.fv_Msg ts), xs))
val abb_funs = map (fn a => case a of
  TermAbbreviation ((f,_,_)) => f
  | ActionsAbbreviation ((f,_,_),_) => f)
  (#abbreviation_spec trac)
val abb_args = map (fn a => case a of
  TermAbbreviation ((_,xs),_) => xs
  | ActionsAbbreviation ((_,xs),_,_) => xs)
  (#abbreviation_spec trac)
fun abb_has_illegal_var_in_body (TermAbbreviation ((_,xs),t)) =
  not (subset (op =) (Trac_Term.fv_Msg t, xs))
| abb_has_illegal_var_in_body (ActionsAbbreviation ((_,xs),acs)) =
  not (subset (op =) (List.concat (map (action_fvs o snd) (expand_abbrevs acs)), xs))

val trs = List.concat (map snd (#transaction_spec trac))
val tr_names = map (#1 o #transaction) trs
val tr_sec_names = map_filter #1 (#transaction_spec trac)
val tr_hds =
  map (fn tr => (#1 (#transaction tr), #2 (#transaction tr))) trs
val tr_acs =
  map (fn tr => (#1 (#transaction tr), #2 (#transaction tr),
    map snd (expand_abbrevs (#actions tr)))) trs
val tr_mem_acs_sets =
  let
    val tr_mem_acs = filter (fn a => is_IN a orelse is_NOTINANY a orelse is_NEGCHECKS a)
      (List.concat (map #3 tr_acs))
    fun f a =
      case a of
        IN (_, (s, _)) => [s]
        | NOTINANY (_, s) => [s]
        | NEGCHECKS (_, bs) =>
          map_filter (fn b => case b of NOTIN (_, (s, _)) => SOME s | _ => NONE) bs
        | _ => []
    val s = tr_mem_acs |> map f |> List.concat |> distinct (op =)
in s end

val illegal_atomic_types = extended_extra_types

317
val new_action_illegal_annotations = enumerations@enum_trac_typeN::illegal_atomic_types
val illegal_composed_type_subterms = enumerations@value_trac_typeN::illegal_atomic_types

val user_types_overlapping Enums =
  filter (member (op =) (#type_spec trac)) enumerations

fun value_free_type (TAtom e) = e <> value_trac_typeN
| value_free_type (TComp (_,ts)) = List.all value_free_type ts

fun var_decl_has_illegal_type (_,TAtom a) = List.exists (fn b => a = b) illegal_atomic_types
| var_decl_has_illegal_type (_,TComp ts) =
  let
    val funs =
      case (#function_spec trac) of
        NONE => []
      | SOME {private=privs, public=pubs} => pubs@privs
    fun illegal_symbol a = List.exists (fn b => a = b) illegal_composed_type_subterms
    fun wrong_arity a bs =
      null bs orelse List.exists (fn (f,n,_) => f = a andalso length bs <> n) funs
    fun check (TAtom a) = illegal_symbol a
    | check (TComp (s,ts)) =
      illegal_symbol s orelse wrong_arity s ts orelse List.exists check ts
  in
    check (TComp ts)
  end

fun no_value_vars_inDecl (tr:transaction) =
  List.all (value_free_type o snd) (#2 (#transaction tr))

fun no_value_vars_inDecl_andNew_Actions (tr:transaction) =
  no_value_vars_inDecl tr andalso
  List.all (List.all (fn (_,t) => case t of SOME tau => value_free_type tau | _ => false))
    (map_filter (maybe_the_NEW o snd) (expand_abbrevs (#actions tr)))

fun is_value_init_transaction (tr:transaction) =
  let
    val acs = map snd (expand_abbrevs (#actions tr))
    val priv_funs = case #function_spec trac of SOME fs => map #1 (#private fs) | NONE => []
    val decl = #2 (#transaction tr)
    fun is_not_value_var x = List.exists (fn (y,t) => x = y andalso value_free_type t) decl
    fun is_not_priv f = List.all (fn g => f <> g) priv_funs
    fun valid_msg (Var x) = is_not_value_var x
    | valid_msg (Const c) = is_not_priv c
    | valid_msg (Fun (f,ts)) = is_not_priv f andalso List.all valid_msg ts
    | valid_msg (Abbrev _) = false
    | valid_msg Attack = true
    fun NEW_action_with_value_annotations_only a =
      case a of
        NEW ts => List.all (fn (_,t) => t = NONE orelse t = SOME (TAtom value_trac_typeN)) ts
      | _ => false
    in
      no_value_vars_inDecl tr andalso
      List.exists NEW_action_with_value_annotations_only acs andalso
      List.all (List.all valid_msg) (map_filter maybe_the_RECEIVE acs) andalso
      List.all (fn a => is_NEW a orelse is_INSERT a orelse is_SEND a) acs andalso
      List.all (fn (_,(_,s,_)) => not (member (op =) tr_mem_acs_sets s))
        (map_filter maybe_the_INSERT acs)
      end
      in
    fun value_producing_transactions_requirement tr_secs =
      List.all (List.exists is_value_init_transaction o snd) tr_secs orelse

4.3 Abstract Syntax for Trac Terms

List.all (List.all no_value_vars_in_decl_and_new_acs o snd) tr_secs

fun set_action_enum_params decls ps =
  let
    fun is_enum_var x = List.exists
      (fn (y,t) => x = y andalso List.exists (fn e => t = TAtom e) finite_enumerations)
      decls
  in
    List.all (fn p => case p of
      Var x => is_enum_var x
    | Const c => List.exists (fn b => b = c) enum_consts
    | Fun (c,ps) => ps = [] andalso List.exists (fn b => b = c) enum_consts
    | _ => false) ps
  end

fun set_action_param_check f ds (INSERT (_,(_,ps))) = f ds ps
  | set_action_param_check f ds (DELETE (_,(_,ps))) = f ds ps
  | set_action_param_check f ds (IN (_,(_,ps))) = f ds ps
  | set_action_param_check f ds (NEGCHECKS (_,ns)) =
    List.all (fn n => case n of NOTIN (_,(_,ps)) => f ds ps | _ => true) ns
  | set_action_param_check _ _ _ = true

(* TODO: deprecate *)
fun action_vars_check decl _ (RECEIVE ts) = subset (op =) (action_fvs (RECEIVE ts), decl)
  | action_vars_check decl _ (LETBINDING (t,t')) = subset (op =) (msg_vars t, decl) andalso
    subset (op =) (msg_vars t', decl)
  | action_vars_check decl _ (SEND ts) = subset (op =) (action_fvs (SEND ts), decl)
  | action_vars_check decl _ (EQUATION p) = subset (op =) (action_fvs (EQUATION p), decl)
  | action_vars_check decl _ (NEGCHECKS p) = subset (op =) (action_fvs (NEGCHECKS p), decl)
  | action_vars_check decl prev_acs (NEW xs) =
    not (has_dups (map fst xs)) andalso
    List.all (fn (x,_) =>
      not (member (op =) decl x) andalso
      not (member (op =) (List.concat (map action_fvs prev_acs)) x))
    xs
  | action_vars_check decl prev_acs (INSERT p) =
    subset (op =) (map action_fvs prev_acs) decl
  | action_vars_check decl prev_acs (DELETE p) =
    subset (op =) (map action_fvs prev_acs) decl
  | action_vars_check decl prev_acs (SEND ts) =
    subset (op =) (map action_fvs prev_acs) decl
  | action_vars_check decl prev_acs _ _ ATTACK = true

(* TODO: test *)
fun wfst' xs [] = xs
  | wfst' xs (a::acs) = case a of
    (RECEIVE ts) => wfst' (distinct (op =) (xs@action_fvs (RECEIVE ts))) acs
  | (LETBINDING (t,)) => wfst' (distinct (op =) (xs@msg_vars t)) acs
  | (IN p) => wfst' (distinct (op =) (xs@action_fvs (IN p))) acs
  | (NEW ys) => wfst' (xs@map fst ys) acs
  | _ => wfst' xs acs

fun wfstp decl xs prev_acs a = case a of
  (RECEIVE ts) => subset (op =) (map action_fvs (RECEIVE ts), decl)
  | (SEND ts) => subset (op =) (map action_fvs (SEND ts), decl)
  | (EQUATION p) => subset (op =) (map action_fvs (EQUATION p), decl)
  | (LETBINDING (t,t')) =>
    subset (op =) (msg_vars t, decl) andalso
    subset (op =) (msg_vars t', decl)
  | (IN p) => subset (op =) (map action_fvs (IN p), decl)
  | (NEGCHECKS p) => subset (op =) (map action_fvs (NEGCHECKS p), decl)
  | (INSERT p) => subset (op =) (map action_fvs (INSERT p), decl)

319
(DELETE p) => subset (op =) (action_fvs (DELETE p), decl@xs)

(NEW ys) =>
    not (has_dups (map fst ys)) andalso
    List.all (fn (y,_) =>
        not (member (op =) decl y) andalso
        not (member (op =) (List.concat (map action_fvs prev_acs)) y))
    ys

ATTACK => true

fun wfst decl prev_acs a =
    let val f = map fst
        fun g (_,tau) = case tau of TAtom ta => member (op =) enumerations ta | _ => true
    in wfstp (f decl) (wfst' (f (filter g decl)) prev_acs) prev_acs a end

fun action_order_check _ (RECEIVE _) = true

| action_order_check next_acs (LETBINDING _) = List.all (not o is_RECEIVE) next_acs
| action_order_check next_acs (EQUATION _) = List.all (not o is_RECEIVE) next_acs
| action_order_check next_acs (NEGCHECKS _) = List.all (not o is_RECEIVE) next_acs
| action_order_check next_acs (NOTINANY _) = List.all (not o is_RECEIVE) next_acs
| action_order_check next_acs (IN _) = List.all (not o is_RECEIVE) next_acs
| action_order_check next_acs (NOTINANY _) = List.all (not o is_RECEIVE) next_acs
| action_order_check next_acs (NEW _) = List.all
    (fn a => is_NEW a orelse is_INSERT a orelse is_DELETE a orelse is_SEND a)
    next_acs
| action_order_check next_acs (INSERT _) = List.all
    (fn a => is_NEW a orelse is_INSERT a orelse is_DELETE a orelse is_SEND a)
    next_acs
| action_order_check next_acs (DELETE _) = List.all
    (fn a => is_NEW a orelse is_INSERT a orelse is_DELETE a orelse is_SEND a)
    next_acs
| action_order_check next_acs (SEND _) = List.all is_SEND next_acs
| action_order_check next_acs ATTACK = next_acs = []

fun new_action_legal_type_annotations a =
    let
        fun f (TAtom a) = List.all (fn b => a <> b) new_action_illegal_annotations
        | f (TComp _) = false
    in
        case a of
            (NEW ts) => List.all f (map_filter snd ts)
            | _ => true
    end

val invalid_funs_in_msg =
    let
        val dist = distinct (op =)
        val conc = dist o List.concat o (fn (f,ms) => map f ms)
        fun f (Var _) = []
        | f (Const _) = []
        | f (Fun (g,ms)) =
            let val n = (g,length ms)
                val ns = conc (f,ms)
            in
                if member (op =) fun_spec n then ns else dist (n::ns)
            end
        | f (Abbrev (x,ms)) = conc (f,ms)
        | f Attack = []
    in
        f
    end

val invalid_funs_in_action a =
    let
        val dist = distinct (op =)
    in
        f
    end
val conc = dist o List.concat o (fn (f,ms) => map f ms)
val f = invalid_funs_in_msg
fun fn fnc [] = []
    | fnc (INEQ (t,t')::ps) = t::t'::fnc ps
    | fnc (NOTIN (t,(_:ts))::ps) = t::ts@fnc ps
in
    case a of
    (RECEIVE ts) => conc (f,ts)
    | (SEND ts) => conc (f,ts)
    | (EQUATION (t,t')) => conc (f,[t,t'])
    | (LETBINDING (t,t')) => conc (f,[t,t'])
    | (IN (t,(_:ts))) => conc (f,t::ts)
    | (NOTINANY (t,(_:)) => f t
    | (NEGCHECKS (_,ps)) => conc (f,fnc ps)
    | (INSERT (t,(_:ts))) => conc (f,t::ts)
    | (DELETE (t,(_:ts))) => conc (f,t::ts)
    | (NEW _) => []
    | ATTACK => []
end

fun invalid_sets_in_action a =
let
    val dist = distinct (op =)
    val conc = dist o (fn (f,ns) => f ns)
    fun f [] = []
        | f ((s,SOME n)::ns) = if member (op =) set_spec (s,n) then f ns else (s,SOME n)::f ns
        | f ((s,NONE)::ns) = if member (op =) set_names s then f ns else (s,NONE)::f ns
    fun fn fnc [] = []
        | fnc (INEQ _::ps) = fnc ps
        | fnc (NOTIN (_,(_,ts))::ps) = (s,SOME (length ts))::fnc ps
    in
    case a of
    (RECEIVE _) => []
    | (SEND _) => []
    | (EQUATION _) => []
    | (LETBINDING _) => []
    | (IN (_,(_:ts))) => conc (f,[s,SOME (length ts)])
    | (NOTINANY (_,s)) => conc (f,[s,NONE])
    | (NEGCHECKS (_,ps)) => conc (f,fnc ps)
    | (INSERT (_,(_:ts))) => conc (f,[s,SOME (length ts)])
    | (DELETE (_,(_:ts))) => conc (f,[s,SOME (length ts)])
    | (NEW _) => []
    | ATTACK => []
end

val invalid_funs_in_abbrevs =
let
    val distconc = distinct (op =) o List.concat
    fun f (LABELED_ACTION (_,a)) = invalid_funs_in_action a
        | f (ABBREVIATION (_,ms)) = distconc (map invalid_funs_in_msg ms)
    fun g (TermAbbreviation ((s,_),m)) = (s,invalid_funs_in_msg m)
        | g (ActionsAbbreviation ((s,_),acs)) = (s,distconc (map f acs))
    in
    filter (fn (_,l) => l <> []) (map g (#abbreviation_spec trac))
end

val invalid_sets_in_abbrevs =
let
    val distconc = distinct (op =) o List.concat
    fun f (LABELED_ACTION (_,a)) = invalid_sets_in_action a
        | f (ABBREVIATION _) = []
    in

fun g (TermAbbreviation ((s,_),_)) = (s,[]) 
| g (ActionsAbbreviation ((s,_),acs)) = (s,distconc (map f acs)) 
in 
filter (fn (_,l) => l <> []) (map g (#abbreviation_spec trac)) end

fun check_actions (tr_name,decl,acs) = 
  let fun chk i = 
    let val a = nth acs i 
    fun result st = (st,tr_name,a) 
    val fs = invalid_funs_in_action a 
    val gs = invalid_sets_in_action a 
    in if fs <> [] 
      then result (InvalidFunctionSymbols fs) 
    else if gs <> [] 
      then result (InvalidSetSymbols gs) 
      else if not (set_action_param_check set_action_enum_params decl a) 
        then result InvalidSetParam 
      else if not (action_order_check (List.drop (acs,i+1)) a) 
        then result WrongPosition 
      else if not (action_vars_check (map fst decl) (List.take (acs,i)) a) orelse 
        not (wfst decl (List.take (acs,i)) a) 
        then result IllformedVars 
      else if not (new_action_legal_type_annotations a) 
        then result InvalidAnnotationNewAction 
      else result Passed 
      end 
  in map chk (0 upto (length acs - 1)) 
  end 
val checked_tr_acs = List.concat (map check_actions tr_acs) 

fun violating_action_exists' f = 
  List.exists (f o #1) checked_tr_acs 

fun violating_action_exists status = 
  violating_action_exists' (fn a => a = status) 

val violating_action_exists_unk_fun_sym = 
  violating_action_exists' (fn a => case a of InvalidFunctionSymbols _ => true | _ => false) 

val violating_action_exists_unk_set_sym = 
  violating_action_exists' (fn a => case a of InvalidSetSymbols _ => true | _ => false) 

fun violating_actions_str' f g = 
  String.concatWith "\n" ( 
    map (fn (st,n,a) => g (st,n,action_str a)) 
    (filter (f o #1) checked_tr_acs)) 

val violating_actions_str_unk_fun_sym = 
  let 
    fun f a = case a of InvalidFunctionSymbols fs => SOME fs | _ => NONE 
    in 
      violating_actions_str' (fn a => f a <> NONE) 
      (fn (st,n,_)) => "symbol(a) " ~ 
      String.concatWith "", " (map (fn (s,n) => s ^ "/" ^ Int.toString n) 
      (Option.getOpt (f st, []))) " 
      " in transaction \"" ^ n ^ "\n\n") 
  end 

val violating_actions_str_unk_set_sym = 
  let 
    fun f a = case a of InvalidSetSymbols fs => SOME fs | _ => NONE 
    in 
      violating_actions_str' (fn a => f a <> NONE) 
      (fn (st,n,_)) => "symbol(a) " ~ 
      String.concatWith "", " (map (fn (s,n) => s ^ "/" ^ Int.toString n) 
      (Option.getOpt (f st, []))) " 
      " in transaction \"" ^ n ^ "\n\n") 
  end 

4.3 Abstract Syntax for Trac Terms

fun violating_actions_str status = 
  violating_actions_str'
  (fn a => a = status)
  (fn (_,n,a) => "action " ^ a ^ " in transaction " ^ n ^ "")

in 

(* if not (value_producing_transactions_requirement (#transaction_spec trac)) then error ( 
  "Missing initial value-producing transaction.\n"
) 
else *)
if has_dups tr_sec_names
  then error ( 
    "Multiple Transactions sections declared with the same name: \n      " ^ (dups_str tr_sec_names) ^ "\n"
)
else if has_dups tr_names
  then error ( 
    "Duplicate transaction declarations: \n      " ^ (dups_str tr_names) ^ "\n"
)
else if has_dups enumerations
  then error ( 
    "Multiple declarations of the same enumeration: \n      " ^ (dups_str enumerations) ^ "\n"
)
else if List.exists (fn n => n = value_trac_typeN) enumerations
  then error ( 
    "The special type \" ^ value_trac_typeN ^ " should not be declared in the trac " ^ "\n"
)
else if List.exists (fn n => n = enum_trac_typeN) enumerations
  then error ( 
    "The special type \" ^ enum_trac_typeN ^ " should not be declared in the trac " ^ "\n"
)
else if has_dups set_names
  then error ( 
    "Multiple declarations of the same set families: \n      " ^ (dups_str set_names) ^ "\n"
)
else if has_dups (fun_names@enum_consts)
  then error ( 
    "Multiple declarations of the same constant or function symbols: \n"
)
4 Trac Support and Automation

dups_str (fun_names@enum_consts))
else if has_dups ana_funs
then error (
"Multiple analysis rules declared for the same function symbols:
" ^ dups_str ana_funs)
else if has_dups abb_funs
then error (
"Multiple abbreviations declared with the same name:
" ^ dups_str abb_funs)
else if List.exists has_dups ana_args
then error (
"The heads of the analysis rules must be linear terms, " ^
"i.e., of the form f(X1,...,Xn) for distinct X1,...,Xn."
"The analysis rules with the following heads violate this condition:
" ^ String.concatWith "\n"
(map (fn i => nth ana_funs i ^ "(" ^ String.concatWith "," (nth ana_args i) ^ ")")
(filter (has_dups o (nth ana_args)) (0 upto (length (#analysis_spec trac) - 1)))))
else if List.exists has_dups abb_args
then error (
"The heads of the abbreviation declarations must be linear terms, " ^
"i.e., of the form f[X1,...,Xn] for distinct X1,...,Xn."
"The abbreviation declaration with the following heads violate this condition:
" ^ String.concatWith "\n"
(map (fn i => nth abb_funs i ^ "![" ^ String.concatWith "," (nth abb_args i) ^ "]")
(filter (has_dups o (nth abb_args)) (0 upto (length (#abbreviation_spec trac) - 1)))))
else if not (null user_types_overlapping_enums)
then error (
"Types declared in the "Types" section cannot also be declared as enumerations in "^ "the "Enumerations" section.\nThe following types violate this condition:
" ^ String.concatWith ", " user_types_overlapping_enums)
else if List.exists (List.exists var_decl_has_illegal_type o snd) tr_hds
then error (
"Transactions must satisfy certain well-formedness requirements on the variables " ^
"declared in their heads:"
"The only special atomic types that may occur in the variable declarations are " ^
"value_trac_typeN ^ " and " ^ enum_trac_typeN ^ ". In particular, the " ^
"following special types are not allowed: " ^
String.concatWith ", " illegal_atomic_types ^ "\n"
"2. For variables declared with composed types no enumeration or special type besides " ^
"enum_trac_typeN ^ " may occur in their types. In particular, the following " ^
"cannot occur in composed types: "^ 
String.concatWith ", " illegal_composed_type_subterms ^ "\n"
"3. The number of parameters applied to a composed type must agree with the arity of " ^
"the function symbol associated with that type."
"The following variable declarations violate these requirements:
" ^ String.concatWith "\n"
(map_filter (fn (n,decls) =>
let val ds = filter var_decl_has_illegal_type decls

in if null ds then NONE
  else SOME (String.concatWith "\n" (map (fn (s,t) => s " : " ^ MsgType_str t) ds) in transaction ^ " n)
end)
tr_hds))
else if invalid_funs_in_abbrevs <> []
then error ("Function symbols occurring in abbreviations in the \"Abbreviations\" section must be " ^ "declared in the \"Functions\" section and must be applied with the correct number of " ^ "arguments.\nThe following function symbols violate this requirement:\n" String.concatWith "\n" (map (fn (ab,fs) => "symbol(s) " ^ String.concatWith ", " (map (fn (f,n) => f " / " ^ Int.toString n) fs) in abbreviation ^ " ab ^ "\n") invalid_funs_in_abbrevs))
else if invalid_sets_in_abbrevs <> []
then error ("Set symbols occurring in abbreviations in the \"Abbreviations\" section must be " ^ "declared in the \"Sets\" section and must be applied with the correct number of " ^ "arguments.\nThe following set symbols violate this requirement:\n" String.concatWith "\n" (map (fn (ab,fs) => "symbol(s) " ^ String.concatWith ", " (map (fn (f,n) => f " / " ^ Int.toString n | _ => "")) fs in abbreviation ^ " ab ^ "\n") invalid_sets_in_abbrevs))
else if violating_action_exists_unk_fun_sym
then error ("Function symbols occurring in transactions in the \"Transactions\" section must be " ^ "declared in the \"Functions\" section and must be applied with the correct number of " ^ "arguments.\nThe following function symbols violate this requirement:\n" violating_actions_str_unk_fun_sym)
else if violating_action_exists_unk_set_sym
then error ("Set symbols occurring in transactions in the \"Transactions\" section must be " ^ "declared in the \"Sets\" section and must be applied with the correct number of " ^ "arguments.\nThe following set symbols violate this requirement:\n" violating_actions_str_unk_set_sym)
else if violating_action_exists WrongPosition
then error ("The sequence of actions occurring in each transaction must either be of the form " ^ "(written here in standard regular expression syntax)\n" (receive t)* (x in s | x notin s | let t = t' | t == t' | t != t')* (send t)* (new x | insert x s | delete x s)* (send t)* or of the form\n" (receive t)* (x in s | x notin s | let t = t' | t == t' | t != t')* attack\n" The following actions lead to violations of these requirements:\n" violating_actions_str WrongPosition)
else if violating_action_exists IllformedVars
then error ("The following well-formedness requirement on the occurrences of variables in " ^ "transactions must be satisfied:\n" "1. Variables in \"send\", \"in\", \"notin\", \"let\", \"==\", and \"!=\" actions must be " ^ "declared in the head of the transaction where these actions occur, or, in the case " ^ "of negative checks, be bound by a \"forall\" quantifier.\n" "2. Variables in a \"new\" action must not occur previously in the same transaction, and they must furthermore be distinct.\n" "3. Variables in \"insert\", \"delete\", and \"send\" actions must occur previously in the same transaction.\n" 325
"The following actions lead to violations of these requirements:

Annotating variables in "new" actions with either enumerations, composed types, or "special types besides \"value_trac_typeN \" is not allowed.
In particular, the following enumerations and atomic types cannot be used in "new" actions:

The following actions violate this requirement:

The following actions violate these requirements:

The parameters to a set-expression must be finite enumerations declared in the "Enumerations" section of the trac specification, and must furthermore be "declared in the transaction where the set-expression occurs. In particular, they must not be variables of type "value_trac_typeN \".

The following actions violate these requirements:

Warning: The specification is not suitable for automated verification. To enable automation the following issues need to be resolved:

fun certifyProtocol (trac:protocol) =
  fun expand_abbreviations (trac:protocol) =
    fun transform_cAction (trac:protocol) =
      fun transform_cMsg trac =
        priv_fun_type_enc trac =
          cInequality (pfe t,pfe t')
4.3 Abstract Syntax for Trac Terms

```haskell
| pne (cNotInSet (t,t')) = cNotInSet (pfe t,pfe t')
| fun aux (cReceive ts) = cReceive (map pfe ts)
| aux (cSend ts) = cSend (map pfe ts)
| aux (cEquality (psv,(t,t'))) = cEquality (psv,(pfe t,pfe t'))
| aux (cInSet (psv,(t,t'))) = cInSet (psv,(pfe t,pfe t'))
| aux (cNotInAny (t,s)) = cNotInAny (pfe t,s)
| aux (cNegChecks (xs,ns)) = cNegChecks (map (fn (x,tau) => (x,pte tau)) xs, map pne ns)
| aux (cInsert (t,t')) = cInsert (pfe t,pfe t')
| aux (cDelete (t,t')) = cDelete (pfe t,pfe t')
| aux (cNew xs) = cNew (map (fn (x,tau) => (x,pte tau)) xs)
| aux cAssertAttack = cAssertAttack
in aux end

fun transform_cTransaction (trac:protocol) (tr:cTransaction) =
  let
    val pae = map (fn (lbl,ac) => (lbl,transform_cAction trac ac))
    val pte = priv_fun_type_enc trac
  in
    {transaction=(case (#transaction tr) of (a,b,c) => (a,map (fn (x,tau) => (x,pte tau)) b,c))
    ,receive_actions=pae (#receive_actions tr)
    ,checksingle_actions=pae (#checksingle_actions tr)
    ,checkall_actions=pae (#checkall_actions tr)
    ,fresh_actions=pae (#fresh_actions tr)
    ,send_actions=pae (#send_actions tr)
    ,attack_actions=pae (#attack_actions tr)}
  end

fun certify (trac:protocol) =
  let
    val certify_ana_msg = transform_cMsg trac o certifyMsgUntyped
    val certify_type = priv_fun_type_enc trac o certifyMsgType' (get_finite_enum_names_trac trac)
    val certify_transaction = transform_cTransaction trac o certifyTransaction (get_finite_enum_names_trac trac)
    val cert_fun_spec =
      let
        fun invalid (_,n,SOME (Trac_Term.TAtom _)) = n <> 0
        | invalid (_,_,SOME (Trac_Term.TComp _)) = true
        | invalid (_,_,NONE) = false
        val _ = case #function_spec trac of
          SOME {private=priv, public=pub} =>
            if List.exists invalid (priv@pub)
            then error ("Error: Invalid type annotation in function specification. " ^
              "Only constants may be annotated with types, and only with " ^
              "atomic types."
            )
            else ()
        | NONE => ()
        fun cert_const (a,b,c) =
          if b = 0
          then (a,Option.map (fn tau =>
                       (case certify_type tau of
                        AtomicType s => s
                        | _ => error ("Error: Invalid type annotation in function " ^
                          "specification: " ^ MsgType_str tau))) c)
          else error ("Error: Expected arity 0 for function symbol " ^ a ^
            " but got " ^ Int.toString b)
        fun cert_fun (a,b,c) =
          case c of
          | ONE => ()
```

327
NONE => (a,b)  
/SOME tau =>  
error ("Error: Expected no type annotation for function symbol " ^ a ^ 
" but got " ^ MsgType_str tau) 

in 
if #function_spec trac = NONE then NONE 
else case get_funs_trac trac of 
(SOME ((private_consts=map cert_const priv_consts, 
public_consts=map cert_const pub_consts, 
public_funs=map cert_fun pub_funs))

end

val cert_ana_spec = 
let 
val (pub_f, _, _) = get_funs_trac trac 
fun ana_arity (f,n) = (if is_priv_fun_trac trac f then n-1 else n) 
fun check_valid_arity ((f,ps),ks,rs) = 
case List.find (fn g => f = #1 g) pub_f of 
SOME (f',n,_) =>
if length ps <> ana_arity (f',n) 
thен error ("Error: Invalid number of parameters in the analysis rule for ",f
" (expected " ^ Int.toString (ana_arity (f',n)) ^ 
" but got " ^ Int.toString (length ps) ^ ")") 
else ((f,ps),ks,rs)

| NONE => error ("Error: " ^ f ^ 
" is not a declared function symbol of arity greater than zero") 
in 
map (fn (h,ks,rs) => 
head=h, keys=map certify_ana_msg ks, 
results=rs, is_priv_fun=isPrivFun_trac trac (fst h))
(map check_valid_arity (#analysis_spec trac))

end

val (cert_transaction_spec:(string option * cTransaction list) list) = 
map (fn (n,trs) => (n,map certify_transaction trs))
(#transaction_spec trac)

val cert_fp = 
Option.map (certify_fixpoint trac) (#fixed_point trac)
in 
({name = #name trac 
,type_spec = #type_spec trac 
,enum_spec = flatten_enum_spec_trac trac 
,set_spec = #set_spec trac 
,function_spec = cert_fun_spec 
,analysis_spec = cert_ana_spec 
,transaction_spec = cert_transaction_spec 
,fixed_point = cert_fp })
end

fun add_intruder_value_gen_transaction (trac:protocol) = 
let 
val spec_tr_names = 
List.concat (map (map (#1 o #transaction) o snd) (#transaction_spec trac))
val spec_set_names = map #1 (#set_spec trac)
val spec_protnames = 
let 
val optnames = map #1 (#transaction_spec trac)
val names = map_index (fn (n,opn) => Option.getOpt (opn,Int.toString n)) optnames
in names end
val spec_atmost1prot = case spec_protnames of (_::_::_) => false | _ => true
fun name_free ns n = List.all (fn s => s <> n) ns
fun gen_name prefix names n =  
  if name_free names prefix then prefix
  else if name_free names (prefix ^ Int.toString n) then prefix ^ Int.toString n
  else gen_name prefix names (n+1)

val set_def = (gen_name "intruderValues" spec_set_names 0,0,false):set_spec_elem

fun tr_name suffix =  
  let val s = if spec_atmost1prot then "" else "_" ^ suffix
  in (gen_name ("intruderValueGen" ^ s) spec_tr_names 0,[],[]):transaction_name end

fun valuegentr protnam = {  
  transaction=tr_name protnam,  
  actions=[  
    LABELED_ACTION (LabelS,NEW [("X",NONE)]),  
    LABELED_ACTION (LabelS,INSERT (Var "X",(#1 set_def,[]))),  
    LABELED_ACTION (LabelS,SEND [Var "X"])
  ]:transaction

val expand_abbrevs =  
  expand_action_abbreviations (get_action_abbreviations_trac trac trac)

val checks_and_deletes_sets =  
  let fun f a = case a of
    DELETE (_,(_,s,_)) => [s]
    | IN (_,(_,s,_)) => [s]
    | NOTINANY (_,s) => [s]
    | NEGCHECKS (_,bs) =>
      map_filter (fn b => case b of NOTIN (_,(_,s,_)) => SOME s | _ => NONE) bs
    | _ => []
  val acs = List.concat (map #actions (List.concat (map (#2) (#transaction_spec trac))))
  val sets = List.concat (map (f o #2) (expand_abbrevs acs))
  in sets end

fun has_valuegentr (trs:transaction list) =  
  let fun is_valuegentr_variant1 acs = case acs of
    [LABELED_ACTION (LabelS,NEW [(x,NONE)]),
     LABELED_ACTION (LabelS,SEND ts)] => member (op =) ts (Var x) andalso (lbl = LabelS orelse spec_atmost1prot)
    | _ => false
  fun is_valuegentr_variant2 acs = case acs of
    [LABELED_ACTION (LabelS,NEW [(x,NONE)]),
     LABELED_ACTION (LabelS,INSERT (y,(s,[]))),
     LABELED_ACTION (LabelS',SEND ta)]
    => member (op =) ts (Var x) andalso y = Var x andalso
      not (member (op =) checks_and_deletes_sets s) andalso
      ((lbl = LabelS andalso lbl' = LabelS) orelse spec_atmost1prot)
    | _ => false
  fun is_valuegentr {transaction=(_,args,ineqs),actions=acs} =  
    List.null args andalso ineqs andalso
    (is_valuegentr_variant1 acs orelse is_valuegentr_variant2 acs)
  in List.exists (is_valuegentr) trs end
4 Trac Support and Automation

,set_spec = set_def::(#set_spec trac)
,abbreviation_spec = #abbreviation_spec trac
,transaction_spec = #transaction_spec trac
,analysis_spec = #analysis_spec trac
,function_spec = #function_spec trac
,abbreviation_spec = #abbreviation_spec trac

(map_index (fn (i,(n,trs)) =>
  if has_valuegentr trs then (n,trs)
  else (n,valuegentr (nth spec_protnames i)::trs))
(#transaction_spec trac)
,fixed_point = #fixed_point trac
):protocol

end

( trac |> check_for_invalid_trac_specification
  |> add_intruder_value_gen_transaction
  |> expand_abbreviations
  |> certify
 ):cProtocol

end

end

4.4 Parser for Trac FP definitions

theory
  trac_fp_parser
imports
  "trac_term"
begin

ML_file "trac_parser/trac_fp.grm.sig"
ML_file "trac_parser/trac_fp.lex.sml"
ML_file "trac_parser/trac_fp.grm.sml"

ML:<
structure TracFpParser : sig
  val parse_file: string -> (Trac_Term.Msg * (string * string) list) list
  val parse_str: string -> (Trac_Term.Msg * (string * string) list) list
end =
struct

open Trac_Term

structure TracLrVals =
  TracLrValsFun(structure Token = LrParser.Token)
structure TracLex =
  TracLexFun(structure Tokens = TracLrVals.Tokens)
structure TracParser =
  Join(structure LrParser = LrParser
  structure ParserData = TracLrVals.ParserData
  structure Lex = TracLex)

fun invoke lexstream =
  let fun print_error (s,i:(int * int * int),_) =
    TextIO.output(TextIO.stdOut,
    print_error s
in
  TracParser.invoke lexstream
end

end
4.5 Parser for the Trac Format

theory
  trac_protocol_parser
imports
  "trac_term"
begin

ML_file "trac_parser/trac_protocol.grm.sig"
ML_file "trac_parser/trac_protocol.lex.sml"
ML_file "trac_parser/trac_protocol.grm.sml"

ML<
structure TracProtocolParser : sig
  val parse_file: string -> TracProtocol.protocol
  val parse_str: string -> TracProtocol.protocol
end =
struct

structure TracLrVals =
  TracTransactionLrValsFun(structure Token = LrParser.Token)

structure TracLex =
  TracTransactionLexFun(structure Tokens = TracLrVals.Tokens)
structure TracParser = 
  Join(structure LrParser = LrParser 
structure ParserData = TracLrVals.ParserData 
structure Lex = TracLex)

fun invoke lexstream = 
  let fun print_error (s,i,(int * int * int),_) = 
    error("Error, line .... " ^ (Int.toString (#1 i)) ^ "." ^ (Int.toString (#2 i ))^ ", " ^ s ^ "\n") 
  in TracParser.parse(0,lexstream,print_error,()) 
  end 

fun parse_fp lexer = let 
  val dummyEOF = TracLrVals.Tokens.EOF((0,0,0),(0,0,0)) 
  fun loop lexer = 
    let 
      val _ = (TracLex.UserDeclarations.pos := (0,0,0);()) 
      val (res,lexer) = invoke lexer 
      val (nextToken,lexer) = TracParser.Stream.get lexer 
      in if TracParser.sameToken(nextToken,dummyEOF) then ((),res) 
         else loop lexer 
      end 
    in (#2(loop lexer)) 
  end 

fun parse_file tracFile = let 
  val infile = TextIO.openIn tracFile 
  val lexer = TracParser.makeLexer (fn _ => case ((TextIO.inputLine) infile) of 
    SOME s => s 
    | NONE => "") 
  in parse_fp lexer 
  handle LrParser.ParseError => TracProtocol.empty 
  end 

fun parse_str str = let 
  val parsed = Unsynchronized.ref false 
  fun input_string _ = if !parsed then "" else (parsed := true ;str) 
  val lexer = TracParser.makeLexer input_string 
  in parse_fp lexer 
  handle LrParser.ParseError => TracProtocol.empty 
  end 

end 

4.6 Support for the Trac Format

theory 
  "trac" 
imports 
trac_fp_parser 
trac_protocol_parser 
keywords 
  "trac" :: thy_decl 
and "trac_import" :: thy_decl 
and "print_transaction_strand" :: thy_decl
4.6 Support for the Trac Format

and "print_transaction_strand_list" :: thy_decl
and "print_attack_trace" :: thy_decl
and "print_fixpoint" :: thy_decl
and "save_fixpoint" :: thy_decl
and "load_fixpoint" :: thy_decl
and "protocol_model_setup" :: thy_decl
and "protocol_security_proof" :: thy_decl
and "protocol_composition_proof" :: thy_decl
and "manual_protocol_model_setup" :: thy_decl
and "manual_protocol_security_proof" :: thy_decl
and "compute_fixpoint" :: thy_decl
and "compute_SMP" :: thy_decl
and "compute_shared_secrets" :: thy_decl
and "setup_protocol_checks" :: thy_decl

begin

ML<
val pspsp_timing = let
  val (pspsp_timing_config, pspsp_timing_setup) = 
    Attrib.config_bool (Binding.name "pspsp_timing") (K false)
  in
    Context.>>(Context.map_theory pspsp_timing_setup);
    pspsp_timing_config
  end

structure trac_time = struct

  fun ap_thy thy msg f x = if Config.get_global thy pspsp_timing
    then Timing.timeap_msg ("PSPSP Timing: ": msg) f x
    else f x

  fun ap_lthy lthy = ap_thy (Proof_Context.theory_of lthy)
end

ML <

(* Some of this is based on code from the following files distributed with Isabelle 2018: 
  * HOL/Tools/value_command.ML 
  * HOL/Code_Evaluation.thy 
  * Pure.thy *)

fun assert_nonempty_name n = 
  if n = "" then error "Error: No name given" else n

fun is_defined lthy name = 
  let
    val full_name = Local_Theory.full_name lthy (Binding.name name)
    val thy = Proof_Context.theory_of lthy
    in
      Sign.const_type thy full_name <> NONE
    end

fun protocol_model_interpretationDefs name = 
  let
    fun f s = 
      (Binding.empty_atts : Attrib.binding, ((Binding.name s, NoSyn), name ^ ".^ " ^ s))
    in
fun assert_defined lthy def = 
  if is_defined lthy def then () 
  else error ("Error: The constant " ^ def ^ " is not defined.")

fun assert_not_defined lthy def = 
  if not (is_defined lthy def) then () 
  else error ("Error: The constant " ^ def ^ " has already been defined.")

fun assert_all_defined lthy name defs = 
  let 
    fun errmsg s = 
      "Error: The following constants were expected to be defined, but are not:\n" ^ 
      String.concatWith "", " s " ^ 
      "\n\nProbable causes:\n\n1. The trac command failed to parse the protocol specification.\n2. The provided protocol specification name (" " ^ name ^ " ") " ^ 
      "does not match the name given in the trac specification.\n3. Manually provided parameters (e.g., " " ^ name ^ " _fixpoint, " ^ name ^ " _SMP) " ^ 
      "may have been misspelled.\n4. Any of the following commands were used before a call to the (manual_)" ^ 
      "protocol_model_setup command:\n" ^ 
      "compute_fixpoint, compute_SMP, protocol_security_proof, manual_protocol_security_proof" 
    val undefs = filter (not o is_defined lthy) defs 
    in 
      if null undefs then defs else error (errmsg undefs) 
    end

fun protocol_model_interpretation_params name lthy = 
  let 
    fun f s = name ^ " _" ^ s 
    val defs = 
      [f "arity", f "sets_arity", f "public", f "Ana", f "Γ"] 
    in 
      map SOME (defs@["0::nat", "1::nat"])
    end

fun declare_thm_attr attribute name print lthy = 
  let 
    val arg = [(Facts.named name, [[Token.make_string (attribute, Position.none)]])] 
    val (_) = Specification.theorems_cmd "" [(Binding.empty_atts, arg)] [] print lthy in 
      lthy' 
    end

fun declare_def_attr attribute name = declare_thm_attr attribute (name ^ "_.def")

val declare_code_eqn = declare_def_attr "code"
val declare_protocol_check = declare_def_attr "protocol_checks"

fun declare_protocol_checks print =
  declare_protocol_check "attack_notin_fixpoint" print #>
  declare_protocol_check "protocol_covered_by_fixpoint" print #>
  declare_protocol_check "protocol_covered_by_fixpoint_alt1" print #>
  declare_protocol_check "analyzed_fixpoint" print #>
  declare_protocol_check "wellformed_protocol" print #>
  declare_protocol_check "wellformed_fixpoint" print #>
  declare_protocol_check "compute_fixpoint_fun" print

fun eval_term lthy t =
  Code_Evaluation.dynamic_value_strict lthy t

fun eval_define (name, t) lthy =
  let
    val t' = eval_term lthy t
    val arg = ((Binding.name name, NoSyn), ((Binding.name (name ^ "_def"),[]), t'))
    val (_, lthy') = Local_Theory.define arg lthy
    in
      (t', lthy')
  end

fun eval_define_declare (name, t) print =
  eval_define (name, t) ##> declare_code_eqn name print

fun eval_define_nbe (name, t) lthy =
  let
    val t' = Nbe.dynamic_value lthy t
    val arg = ((Binding.name name, NoSyn), ((Binding.name (name ^ "_def"),[]), t'))
    val (_, lthy') = Local_Theory.define arg lthy
    in
      (t', lthy')
  end

fun eval_define_declare_nbe (name, t) print =
  eval_define_nbe (name, t) ##> declare_code_eqn name print

ML<
structure ml_isar_wrapper = struct
  fun define_constant_definition (constname, trm) lthy =
    let
      val arg = ((Binding.name constname, NoSyn), ((Binding.name (constname ^ "_def"),[]), trm))
      val (_, (_, thm), lthy') = Local_Theory.define arg lthy
      in
        (thm, lthy')
    end

  fun define_constant_definition' (constname, trm) print lthy =
    let
      val arg = ((Binding.name constname, NoSyn), ((Binding.name (constname ^ "_def"),[]), trm))
      val (_, (_, thm), lthy') = Local_Theory.define arg lthy
      val lthy'' = declare_code_eqn constname print lthy'
      in
        (thm, lthy'')
    end

  fun define_simple_abbrev (constname, trm) lthy =
    let
      val arg = ((Binding.name constname, NoSyn), trm)
      val lthy'' = declare_code_eqn constname print lthy'
      in
        (thm, lthy'')
    end

ML>
val ((_, _), lthy') = Local_Theory.abbrev Syntax.mode_default arg lthy
in
lthy'
end

fun define_simple_type_synonym (name, typedecl) lthy =
let
  val (_, lthy') = Typedecl.abbrev_global (Binding.name name, [], NoSyn) typedecl lthy
in
  lthy'
end

fun define_simple_datatype (dt_tyargs, dt_name) constructors =
let
  val options = Plugin_Name.default_filter
  fun lift_c (tyargs, name) = (((Binding.empty, Binding.name name), map (fn t => (Binding.empty, t))) tyargs, NoSyn)
  val c_spec = map lift_c constructors
  val datatyp = ((map (fn ty => (NONE, ty)) dt_tyargs, Binding.name dt_name), NoSyn)
  val dtspec = ((options, false),
    [((datatyp, c_spec), (Binding.empty, Binding.empty, Binding.empty)), ()])
  in
    BNF_FP_Def_Sugar.co_datatypes BNF_Util.Least_FP BNF_LFP.construct_lfp dtspec
  end

fun define_simple_primrec pname precs lthy =
let
  val rec_eqs = map (fn (lhs, rhs) => (((Binding.empty, []), HOLogic.mk_Trueprop (HOLogic.mk_eq (lhs, rhs)))), [], []) precs
  in
    snd (BNF_LFP_Rec_Sugar.primrec false [] [(Binding.name pname, NONE, NoSyn)] rec_eqs lthy)
  end

fun define_simple_fun pname precs lthy =
let
  val rec_eqs = map (fn (lhs, rhs) => (((Binding.empty, []), HOLogic.mk_Trueprop (HOLogic.mk_eq (lhs, rhs)))), [], []) precs
  in
    Function_Fun.add_fun [(Binding.name pname, NONE, NoSyn)] rec_eqs Function_Common.default_config lthy
  end

fun prove_simple name stmt tactic lthy =
let
  val thm = Goal.prove lthy [] [] stmt (fn {context, ...} => tactic context)
    |> Goal.norm_result lthy
    |> Goal.check_finished lthy
  in
    lthy |> snd o Local_Theory.note ((Binding.name name, []), [thm])
  end

fun prove_state_simple method proof_state =
  Seq.the_result "error in proof state" ( (Proof.refine method proof_state))
    |> Proof.global_done_proof
end

ML<

structure trac_definitorial_package = struct
4.6 Support for the Trac Format

type hide_tvar_tab = (TracProtocol.protocol) Symtab.table
fun trac_eq (a, a') = (#name a) = (#name a')
fun merge_trac_tab (tab, tab') = Symtab.merge trac_eq (tab, tab')

structure Data = Generic_Data
(
  type T = hide_tvar_tab
  val empty = Symtab.empty:hide_tvar_tab
  val extend = I
  fun merge(t1, t2) = merge_trac_tab (t1, t2)
);

fun update p thy = Context.theory_of
  ((Data.map (fn tab => Symtab.update (#name p, p) tab) (Context.Theory thy)))
fun lookup name thy = (Symtab.lookup ((Data.get o Context.Theory) thy) name, thy)
fun lookup_trac (pname:string) lthy =
  Option.valOf (fst (lookup pname (Proof_Context.theory_of lthy)))

(* constant names *)
open Trac_Utils
val enum_constsN="enum_consts"
val setsN="sets"
val funN="fun"
val atomN="atom"
val arityN="arity"
val set_arityN=setsN^"_"^arityN
val publicN = "public"
val gammaN = "Γ"
val anaN = "Ana"

fun mk_listT T = Type ("List.list", [T])
val mk_setT = HOLogic.mk_setT
val boolT = HOLogic.boolT
val natT = HOLogic.natT
val mk_tupleT = HOLogic.mk_tupleT
val mk_prodT = HOLogic.mk_prodT

val mk_set = HOLogic.mk_set
val mk_list = HOLogic.mk_list
val mk_nat = HOLogic.mk_nat
val mk_eq = HOLogic.mk_eq
val mk_Trueprop = HOLogic.mk_Trueprop
val mk_tuple = HOLogic.mk_tuple
val mk_prod = HOLogic.mk_prod

fun mkN (a,b) = a^"_"^b

val info = Output.information

fun full_name name lthy =
  Local_Theory.full_name lthy (Binding.name name)

fun full_name' n (trac:TracProtocolCert.cProtocol) lthy = full_name (mkN (#name trac, n)) lthy

fun mk_prot_type name targs (trac:TracProtocolCert.cProtocol) lthy =
  Term.Type (full_name' name trac lthy, targs)
val enum_constsT = mk_prot_type enum_constsN []

fun mk_enum_const a trac lthy =
  Term.Const (full_name' enum_constsN trac lthy ^ "." ^ a, enum_constsT trac lthy)
val setexprT = mk_prot_type setsN []

val funT = mk_prot_type funN []
val atomT = mk_prot_type atomN []

fun messageT (trac:TracProtocolCert.cProtocol) lthy =
  Term.Type ("Transactions.prot_term", [funT trac lthy, atomT trac lthy, setexprT trac lthy, natT])

fun message_funT (trac:TracProtocolCert.cProtocol) lthy =
  Term.Type ("Transactions.prot_fun", [funT trac lthy, atomT trac lthy, setexprT trac lthy, natT])

fun message_varT (trac:TracProtocolCert.cProtocol) lthy =
  Term.Type ("Transactions.prot_var", [funT trac lthy, atomT trac lthy, setexprT trac lthy, natT])

fun message_term_typeT (trac:TracProtocolCert.cProtocol) lthy =
  Term.Type ("Transactions.prot_term_type",
    [funT trac lthy, atomT trac lthy, setexprT trac lthy, natT])

fun message_term_type_listT (trac:TracProtocolCert.cProtocol) lthy =
  mk_listT (message_term_typeT trac lthy)

fun message_atomT (trac:TracProtocolCert.cProtocol) lthy =
  Term.Type ("Transactions.prot_atom", [atomT trac lthy])

fun message_varT' (trac:TracProtocolCert.cProtocol) lthy =
  Term.Type ("Stateful_Strands.stateful_strand_step",
    [funT trac lthy, message_varT trac lthy])

fun message_listT (trac:TracProtocolCert.cProtocol) lthy =
  mk_listT (messageT trac lthy)

fun message_listT' (trac:TracProtocolCert.cProtocol) lthy =
  mk_listT (messageT' trac lthy)

fun absT (trac:TracProtocolCert.cProtocol) lthy =
  mk_setT (setexprT trac lthy)

fun absT' (trac:TracProtocolCert.cProtocol) lthy =
  mk_setT (absT trac lthy)

val poscheckvariantT =
  Term.Type ("Strands_and_Constraints.poscheckvariant", [])

val strand_labelT =
  Term.Type ("Labeled_Strands.strand_label", [natT])

fun strand_stepT (trac:TracProtocolCert.cProtocol) lthy =
  Term.Type ("Stateful_Strands.stateful_strand_step",
    [funT trac lthy, message_varT trac lthy])

fun labeled_strand_stepT (trac:TracProtocolCert.cProtocol) lthy =
  mk_prodT (strand_labelT, strand_stepT trac lthy)

fun prot_strandT (trac:TracProtocolCert.cProtocol) lthy =
  mk_listT (labeled_strand_stepT trac lthy)

fun prot_transactionT (trac:TracProtocolCert.cProtocol) lthy =
  Term.Type ("Transactions.prot_transaction",
    [funT trac lthy, atomT trac lthy, setexprT trac lthy, natT])

val mk_star_label =
  Term.Const ("Labeled_Strands.strand_label.LabelS", strand_labelT)

fun mk_prot_label (lbl:int) =
4.6 Support for the Trac Format

```
mk_nat lbl

fun mk_labeled_step (label:term) (step:term) =
        mk_prod (label, step)

fun mk_Send_step (trac:TracProtocolCert.cProtocol) lthy (label:term) (msgs:term list) =
        mk_labeled_step label
(Term.Const ("Stateful_Strands.stateful_strand_step.Send",
mk_listT (messageT trac lthy) --> strand_stepT trac lthy) $}$
mk_list (messageT trac lthy) msgs)

fun mk_Receive_step (trac:TracProtocolCert.cProtocol) lthy (label:term) (msgs:term list) =
        mk_labeled_step label
(Term.Const ("Stateful_Strands.stateful_strand_step.Receive",
mk_listT (messageT trac lthy) --> strand_stepT trac lthy) $}
mk_list (messageT trac lthy) msgs)

        let
            val psT = [poscheckvariantT, messageT trac lthy, messageT trac lthy]
            val psvN =
                case psv of TracProtocolCert.cCheck => "Check" | TracProtocolCert.cAssignment => "Assign"
in
                mk_labeled_step label
(Term.Const ("Stateful_Strands.stateful_strand_step.InSet",
psT ---> strand_stepT trac lthy) $}
Term.Const ("Strands_and_Constraints.poscheckvariant." ^ psvN, poscheckvariantT) $}
elem $ set)
        end

fun mk_NegChecks_step (trac:TracProtocolCert.cProtocol) lthy (label:term)
        (bvars:term list) (ineqs:(term*term) list) (notins:(term*term) list) =
        let
            val msgT = messageT trac lthy
            val varT = message_varT trac lthy
            val trm_prodT = mk_prodT (messageT trac lthy, messageT trac lthy)
            val psT = [mk_listT varT, mk_listT trm_prodT, mk_listT trm_prodT]
in
                mk_labeled_step label
(Term.Const ("Stateful_Strands.stateful_strand_step.NegChecks",
psT ---> strand_stepT trac lthy) $}
(case bvars of
    [] => mk_list varT []
  | [x] => mk_list varT [Term.Const (@{const_name "the_Var"}, msgT --> varT) $ x]
  | xs => Term.Const (@{const_name "map"}, [msgT --> varT, mk_listT msgT] ---> mk_listT varT) $}
      Term.Const (@{const_name "the_Var"}, msgT --> varT) $}
      mk_list msgT xs) $}
mk_list trm_prodT (map mk_prod ineqs) $}
mk_list trm_prodT (map mk_prod notins))
        end

fun mk_NotInSet_step (trac:TracProtocolCert.cProtocol) lthy (label:term)
        (elem:term) (set:term) =
        mk_NegChecks_step trac lthy label [] [] [(elem, set)]

        let
            val psT = [poscheckvariantT, messageT trac lthy, messageT trac lthy]
            val psvN =
                case psv of TracProtocolCert.cCheck => "Check" | TracProtocolCert.cAssignment => "Assign"
in
                mk_labeled_step label
```
4 Trac Support and Automation

(Term.Const ("Stateful_Strands.stateful_strand_step.Equality", 
  psT ---> strand_stepT trac lthy) $ 
  Term.Const ("Strands_and_Constraints.poscheckvariant." ^ psvN, poscheckvariantT) $ t1 $ t2) 
end

mk_labeled_step label 
  (Term.Const ("Stateful_Strands.stateful_strand_step.Insert", 
      [messageT trac lthy, messageT trac lthy] ---> strand_stepT trac lthy) $ 
  elem $ set)

mk_labeled_step label 
  (Term.Const ("Stateful_Strands.stateful_strand_step.Delete", 
      [messageT trac lthy, messageT trac lthy] ---> strand_stepT trac lthy) $ 
  elem $ set)

fun mk_Transaction (trac:TracProtocolCert.cProtocol) lthy S0 S1 S2 S3 S4 S5 S6 = 
  let 
    val varT = message_varT trac lthy 
    val msgT = messageT trac lthy 
    val var_listT = mk_listT varT 
    val msg_listT = mk_listT msgT 
    val fun_setT = mk_setT (funT trac lthy) 
    val trT = prot_transactionT trac lthy 
    val declT = mk_prodT (varT, fun_setT) 
    val decl_listT = mk_listT declT 
    val decl_list_funT = HOLogic.unitT --> decl_listT 
    val stepT = labeled_strand_stepT trac lthy 
    val strandT = prot_strandT trac lthy 
    val strandsT = mk_listT strandT 
    in 
      Term.Const ("Transactions.prot_transaction.Transaction", paramsT ---> trT) $ 
      (Term.Const ("Product_Type.unit.case_unit", decl_listT --> decl_list_funT) $ 
        if null S4 then mk_list varT [] 
        else Term.Const (@{const_name "map"}, [msgT --> varT, msg_listT] ---> var_listT) $ 
          Term.Const (@{const_name "the_Var"}, msgT --> varT) $ 
          mk_list msgT S4) $ 
        mk_list stepT S1 $ 
        if null S3 then mk_list stepT S2 
        else Term.Const (@{const_name "append"}, [strandT,strandT] ---> strandT) $ 
          mk_list stepT S2 $ 
          (Term.Const (@{const_name "concat"}, strandsT --> strandT) $ mk_list strandT S3)) $ 
        mk_list stepT S5 $ 
        mk_list stepT S6 
    end

fun get_funs (trac:TracProtocolCert.cProtocol) = 
case #function_spec trac of 
  NONE => ([],[],[]) 
  | SOME ({public_funs=pub_funs, public_consts=pub_consts, private_consts=priv_consts}) => 
    (pub_funs, pub_consts, priv_consts)

(* TODO: consider differentiating between "/" sets and "/" sets *)
fun get_set_spec (trac:TracProtocolCert.cProtocol) = 
distinct (op =) (map (fn (s,n,) => (s,n)) (#set_spec trac))

fun get_general_set_family_set_spec (trac:TracProtocolCert.cProtocol) = 
distinct (op =) (map_filter (fn (s,n,b) => if b then SOME (s,n) else NONE) (#set_spec trac))

fun is_general_set_family (trac:TracProtocolCert.cProtocol) s =
4.6 Support for the Trac Format

```plaintext
List.exists (fn (s',_) => s = s') (get_general_set_family_set_spec trac)

fun set_arity (trac:TracProtocolCert.cProtocol) s =
  case List.find (fn x => fst x = s) (get_set_spec trac) of
    SOME (_,n) => SOME n
  | NONE => NONE

fun get_enum_consts (trac:TracProtocolCert.cProtocol) =
  distinct (op =) (List.concat (map #2 (#enum_spec trac)))

fun get_finite_enum_spec (trac:TracProtocolCert.cProtocol) =
  filter (null o #3) (#enum_spec trac)

fun get_infinite_enum_spec (trac:TracProtocolCert.cProtocol) =
  filter_out (null o #3) (#enum_spec trac)

fun get_nonunion_infinite_enum_spec (trac:TracProtocolCert.cProtocol) =
  filter (fn (e,cs,ies) => null cs andalso ies = [e])
  (get_infinite_enum_spec trac)

fun get_typed_constants_in_function_spec (trac:TracProtocolCert.cProtocol) =
  case #function_spec trac of
    SOME {private_consts=priv, public_consts=pub, ...} =>
      map_filter (fn (c,t) => Option.map (fn a => (c,a)) t) (priv@pub)
    | NONE => []

fun get_user_atom_spec_pre (trac:TracProtocolCert.cProtocol) =
  map (fn s => (s,([boolT,natT],s"constant"))) (#type_spec trac)

fun get_user_atom_spec (trac:TracProtocolCert.cProtocol) =
  map (fn (c,a) => (a,([],c))) (get_typed_constants_in_function_spec trac)@
  get_user_atom_spec_pre trac

fun is_attack_transaction (tr:TracProtocolCert.cTransaction) =
  not (null (#attack_actions tr))

fun get_transaction_name (tr:TracProtocolCert.cTransaction) =
  #1 (#transaction tr)

fun get_transaction_head_variables (tr:TracProtocolCert.cTransaction) =
  #2 (#transaction tr)

fun get_bound_variables (tr:TracProtocolCert.cTransaction) =
  let
    val a = map_filter (TracProtocolCert.maybe_the_NegChecks o snd) (#checksingle_actions tr)
  in
    distinct (op =) (List.concat (map fst a))
  end

fun get_fresh_variables (tr:TracProtocolCert.cTransaction) =
  List.concat (map_filter (TracProtocolCert.maybe_the_Fresh o snd) (#fresh_actions tr))

fun get_fresh_value_variables (tr:TracProtocolCert.cTransaction) =
  map_filter (fn (x,tau) => case tau of Trac_Term.ValueType => SOME x | _ => NONE)
  (get_fresh_variables tr)

fun get_nonfresh_value_variables (tr:TracProtocolCert.cTransaction) =
  map fst (filter (fn x => snd x = Trac_Term.ValueType) (get_transaction_head_variables tr))

fun get_value_variables (tr:TracProtocolCert.cTransaction) =
  get_nonfresh_value_variables tr@get_fresh_value_variables tr

fun get_finite_enum_variables (tr:TracProtocolCert.cTransaction) =
  []
```

341
4 Trac Support and Automation

distinct (op =) (filter (fn (_, tau) => case tau of
        Trac_Term.Enumeration _ => true
      | _ => false)
    (get_transaction_head_variables tr))

fun get_infinite_enum_variables (tr:TracProtocolCert.cTransaction) =
distinct (op =) (filter (fn (_, tau) => case tau of
        Trac_Term.InfiniteEnumeration _ => true
      | _ => false)
    (get_transaction_head_variables tr))

fun get_enumtype_variables (tr:TracProtocolCert.cTransaction) =
distinct (op =) (filter (fn (_, tau) => tau = Trac_Term.EnumType)
    (get_transaction_head_variables tr))

fun get_nonenum_variables (tr:TracProtocolCert.cTransaction) =
  map_filter (fn (x, tau) => case tau of
      Trac_Term.Enumeration _ => NONE
    | Trac_Term.InfiniteEnumeration _ => NONE
    | _ => SOME (x, tau))
    (get_transaction_head_variables tr @ get_fresh_variables tr)

fun get_variable_restrictions (tr:TracProtocolCert.cTransaction) =
  let
    val enum_vars = get_finite_enum_variables tr
    val value_vars = get_value_variables tr
    fun enum_member x = List.exists (fn y => x = fst y)
    fun value_member x = List.exists (fn y => x = y)
    fun aux [] = ([], [])
    | aux ((a, b)::rs) =
      if enum_member a enum_vars andalso enum_member b enum_vars
        then let val (es, vs) = aux rs in ((a, b)::es, vs) end
      else if value_member a value_vars andalso value_member b value_vars
        then let val (es, vs) = aux rs in (es, (a, b)::vs) end
      else error "Error: Ill-formed or ill-typed variable restriction: {} = " ^ a ^ " != " ^ b
    in aux (#3 (#transaction tr))
  end

fun setexpr_to_hol (db: (string * Trac_Term.cMsg list) list) (tr: TracProtocolCert.cProtocol) lthy =
  let
    open Trac_Term
    fun mkN' n = mkN (#name tr, n)
    val s_prefix = full_name (mkN' setsN) lthy ^ "."
    val e_prefix = full_name (mkN' enum_constsN) lthy ^ "."
    val (s, es) = db
    val tau = enum_constsT tr lthy
    val setexprT = setexprT tr lthy
    val a = Term.Const (s_prefix ^ s, map (fn _ => tau) es ---> setexprT)
    fun param_to_hol (cVar (x, Enumeration _)) = Term.Free (x, tau)
    | param_to_hol (cVar (x, EnumType)) = Term.Free (x, tau)
    | param_to_hol (cEnum e) = Term.Const (e_prefix ^ e, tau)
    | param_to_hol t = error "Error: Invalid set parameter: " ^ cMsg_str t
    in
      fold (fn e => fn b => b $ param_to_hol e) a es a
  end

fun abs_to_hol (bs: (string * string list) list) (tr: TracProtocolCert.cProtocol) lthy =
  mk_set (setexprT tr lthy)
    (map (fn (s, cs) => setexpr_to_hol (s, map Trac_Term.cEnum cs) tr lthy) bs)

fun cType_to_hol (t: Trac_Term.cType) tr lthy =
4.6 Support for the Trac Format

let

open Trac_Term
val atomT = atomT trac lthy
val prot_atomT = message_atomT trac lthy
val tT = message_term_typeT trac lthy
val fT = message_funT trac lthy
val tsT = message_term_type_listT trac lthy
val TAtomT = prot_atomT --> tT
val TCompT = [fT, tsT] ---> tT
val funT = funT trac lthy
val setexprT = setexprT trac lthy
val SetT = setexprT --> fT
val FuT = funT --> fT
val TAtomC = Term.Const (@{const_name "Var"}, TAtomT)
val TCompC = Term.Const (@{const_name "Fun"}, TCompT)
val AtomC = Term.Const (@{const_name "Transactions.prot_atom.Atom"}, atomT --> prot_atomT)
val full_name'' n = full_name' n trac lthy
val mk_prot_fun_trm f tau = Term.Const (@{const_name "Transactions.prot_fun." ^ f, tau})
val mk_Fu_trm f = mk_prot_fun_trm "Fu" FuT $ Term.Const (full_name'' funN ^ ".", ^ f, funT)
val mk_Set_trm (s,ts) = mk_prot_fun_trm "Set" SetT $ setexpr_to_hol (s,ts) trac lthy
val c_to_h s = cType_to_hol s trac lthy
val c_list_to_h ts = mk_list tT (map c_to_h ts)
val mk_atom_trm n = Term.Const (full_name'' atomN ^ ".", ^ n, atomT)
val EnumType_trm = TAtomC $ (AtomC $ mk_atom_trm enum_typeN)
val ValueType_trm = TAtomC $ Term.Const (@{const_name "Transactions.prot_atom.Value"}, prot_atomT)

in

case t of
  Enumeration _ => EnumType_trm
  InfiniteEnumeration _ => EnumType_trm
  EnumType => EnumType_trm
  ValueType => ValueType_trm
  PrivFunSecType => TAtomC $ (AtomC $ mk_atom_trm secret_typeN)
  AtomicType s => TAtomC $ (AtomC $ mk_atom_trm s)
  ComposedType (f,ts) => TCompC $ mk_Fu_trm f $ c_list_to_h ts
  Untyped => error "Error: Expected a type but got untyped"

end

fun cMsg_to_hol (t:Trac_Term.cMsg) lbl varT var_map free_enum_var free_message_var trac lthy = let

open Trac_Term
val tT = messageT' varT trac lthy
val fT = message_funT trac lthy
val enum_constsT = enum_constsT trac lthy
val tsT = message_listT' varT trac lthy
val VarT = varT --> tT
val FunT = [fT, tsT] ---> tT
val absT = absT trac lthy
val setexprT = setexprT trac lthy
val AbsT = absT --> fT
val funT = funT trac lthy
val FuT = funT --> fT
val SetT = setexprT --> fT
val enumT = enum_constsT --> funT
val VarC = Term.Const (@{const_name "Var"}, VarT)
val FunC = Term.Const (@{const_name "Fun"}, FunT)
val NilC = Term.Const (@{const_name "Nil"}, tsT)
val prot_label = mk_prot_label lbl
fun full_name'' n = full_name' n trac lthy
fun mk_enum_const' a = mk_enum_const a trac lthy
fun mk_prot_fun_trm f tau = Term.Const (@{const_name "Transactions.prot_fun." ^ f, tau})

in


fun mk_enum_trm etrm =  
  mk_prot_fun_trm "Fu" FuT $(Term.Const (full_name'' funN ^ "." ^ enumN, enumT)) $ etrm
fun mk_Fu_trm f =  
  mk_prot_fun_trm "Fu" FuT $(Term.Const (full_name'' funN ^ "." ^ f, funT))
fun c_to_h s = cMsg_to_hol s lbl varT var_map free_enum_var free_message_var trac lthy
fun c_list_to_h ts = mk_list tT (map c_to_h ts)
in
  case t of
    cVar x =>
      if free_enum_var x
      then FunC $ mk_enum_trm (Term.Free (fst x, enum_constsT)) $ NilC
      else if free_message_var x
      then (* Term.Free (fst x, tT) *) (* TODO: somehow Isabelle doesn't realize that tT is the same as messageT when varT is the right type - maybe it's the type synonyms in messageT which is to blame *)
        Term.Free (fst x, messageT trac lthy)
      else VarC $ var_map x
    | cConst f =>
      FunC $ mk_Fu_trm f $
      NilC
    | cFun (f,ts) =>
      FunC $ mk_Fu_trm f $
      c_list_to_h ts
    | cSet (s,ts) =>
      if is_general_set_family trac s
      then FunC $ (mk_prot_fun_trm "Set" SetT $ setexpr_to_hol (s,[]) trac lthy) $
                    mk_list tT (map c_to_h (cPrivFunSec::ts))
      else FunC $ (mk_prot_fun_trm "Set" SetT $ setexpr_to_hol (s,ts) trac lthy) $
                    NilC
    | cAttack =>
      FunC $ (mk_prot_fun_trm "Attack" (strand_labelT --> fT) $ prot_label) $
      NilC
    | cAbs bs =>
      FunC $ (mk_prot_fun_trm "Abs" AbsT $ abs_to_hol bs trac lthy) $
      NilC
    | cOccursFact bs =>
      FunC $ mk_prot_fun_trm "OccursFact" fT $
      mk_list tT [ 
                    FunC $ mk_prot_fun_trm "OccursSec" fT $ NilC,
                    c_to_h bs]
    | cPrivFunSec =>
      FunC $ mk_Fu_trm priv_fun_secN $
      NilC
    | cEnum a =>
      FunC $ mk_enum_trm (mk_enum_const' a) $
      NilC
  end

fun ground_cMsg_to_hol t lbl trac lthy =  
cMsg_to_hol t lbl (message_varT trac lthy) (fn _ => error "Error: Term not ground")
  (fn _ => false) (fn _ => false) trac lthy
fun ana_cMsg_to_hol inc_vars t (ana_var_map:string list) =
4.6 Support for the Trac Format

let

open Trac_Term

fun var_map (x, Untyped) = (  
  case list_find (fn y => x = y) ana_var_map of  
  SOME (_, n) => if inc_vars then mk_nat (1+n) else mk_nat n  
  | NONE => error ("Error: Analysis variable " ^ x ^ " not found")  
  | var_map _ = error "Error: Analysis variables must be untyped"

val lbl = 0 (* There's no constants in analysis messages requiring labels anyway *)

in

cMsg_to_hol t lbl natT var_map (fn _ => false) (fn _ => false)
end

fun transaction_cMsg_to_hol t lbl transaction_var_map free_vars trac lthy =  
let

open Trac_Term
val varT = message_varT trac lthy

fun var_map (x, tau) =  
  case list_find (fn y => (x, tau) = y) transaction_var_map of  
  SOME (_, n) => HOLogic.mk_prod (cType_to_hol tau trac lthy, mk_nat n)  
  | NONE => error ("Error: Transaction variable " ^ cMsg_str (cVar (x, tau)) ^ " not found")

fun free_enum_var (_, Enumeration _) = true
| free_enum_var _ = false

in

cMsg_to_hol t lbl varT var_map free_enum_var (fn _ => free_vars) trac lthy
end

fun fp_triple_to_hol (fp, occ, ti) trac lthy =  
let

val prot_label = 0
val tau_abs = absT trac lthy
val tau_fp_elem = messageT trac lthy
val tau_occ_elem = tau_abs
val tau_ti_elem = mk_prodT (tau_abs, tau_abs)

fun a_to_h bs = abs_to_hol bs trac lthy
fun c_to_h t = ground_cMsg_to_hol t prot_label trac lthy
val fp' = mk_list tau_fp_elem (map c_to_h fp)
val occ' = mk_list tau_occ_elem (map a_to_h occ)
val ti' = mk_list tau_ti_elem (map (mk_prod o map_prod a_to_h) ti)

in
mk_tuple [fp', occ', ti']
end

fun absfreeprod tau xs trm =  
let

val tau_out = Term.fastype_of trm
fun absfree' x = absfree (x, tau)

fun aux _ [] = trm
| aux _ [x] = absfree' x trm
| aux len (x::y::xs) =  
  Term.Const ("case_prod",  
  [[tau, mk_tupleT (replicate (len-1) tau)] -->> tau_out,  
    mk_tupleT (replicate len tau)] -->> tau_out)$  
  absfree' x (aux (len-1) (y::xs))

in
aux (length xs) xs
end

fun abstract_over_finite_enum_vars enum_vars enum_ineqs trm trac lthy =  
let

val enum_constsT = enum_constsT trac lthy
val absfreeprod' = absfreeprod enum_constsT

fun enumlistelemT n = mk_tupleT (replicate n enum_constsT)
fun enumlistT n = mk_listT (enumlistelemT n)
fun mk_enum_const' a = mk_enum_const a trac lthy

fun mk_enumlist ns = mk_list enum_constsT (map mk_enum_const' ns)

fun mk_enum_neq (a,b) = (HOLogic.mk_not o HOLogic.mk_eq)
  (Term.Free a, enum_constsT), Term.Free (b, enum_constsT))

fun mk_enum_neqs_list [] = Term.Const (@{const_name "True"}, HOLogic.boolT)
| mk_enum_neqs_list [x] = mk_enum_neq x
| mk_enum_neqs_list (x::y::xs) = HOLogic.mk_conj (mk_enum_neq x, mk_enum_neqs_list (y::xs))

val enum_types =
let
  open Trac_Term

  val flat_enum_spec = map (fn (a,b,_) => (a,b)) (get_finite_enum_spec trac)
  val err_pre = "Error: Expected a finite enumeration, but got "

  fun aux (Enumeration t) = (a, enum_constsT)
    case List.find (fn (s,_) => t = s) flat_enum_spec of
      SOME (_,cs) => (t,cs)
    | NONE => error (err_pre ^ "^ t ^ has not been declared as an enumeration")
    | aux Untyped = (enum_constsN, get_enum_consts trac)
    | aux (InfiniteEnumeration t) = error (err_pre ^ "an infinite enumeration: ^ t")
    | aux tau = error (err_pre ^ "type ^ cType_str tau")

  in
    map (aux o snd) enum_vars
  end

fun enumlist_product f nil_case =
let
  fun aux _ [] = nil_case ()
| aux _ [ns] = f ns
| aux len (ns::ms::elists) =
    Term.Const ("List.product", [enumlistT 1, enumlistT (len-1)] ---> enumlistT len) $ f ns $ aux (len-1) (ms::elists)

  in
    aux (length enum_types) enum_types
  end

val enable_let_bindings = false
val absfp = absfreeprod' (map fst enum_vars) trm
val eptrm = if length enum_vars > 1 andalso enable_let_bindings
then enumlist_product
  (fn (x,_) => Term.Free (x, mk_listT enum_constsT))
  (fn () => error "Error: Nil in enumlist_product")
else enumlist_product (mk_enumlist o snd) (fn () => mk_enumlist [])

val typo = Term.fastype_of
val evseT = enumlistelemT (length enum_vars)
val evslT = enumlistT (length enum_vars)
val eneqs = absfreeprod' (map fst enum_vars) (mk_enum_neqs_list enum_ineqs)
in
if null enum_vars
then mk_list (typo trm) [trm]
else let
    val a = Term.Const (@{const_name "map"},
      [typo absfp, typo eptrm] ---> mk_listT (typo trm)) $ absfp

    val b = if null enum_ineqs
      then eptrm
    else Term.Const (@{const_name "filter"},
      [evseT ---> HOLogic.boolT, evslT] ---> evslT) $
4.6 Support for the Trac Format

eneqs $ eptrm
val c = absfreeprod (mk_listT enum_constsT) (distinct (op =) (map fst enum_types)) (a$b)
val d = mk_tuple (map mk_enumlist (distinct (op =) (map snd enum_types)))
val e = Term.Const (@{const_name "Let"}, [typof d, typof c] ---› typof (c$d))d$c
in if length enum_vars > 1 andalso enable_let_bindings then e else a $ b end
end

fun mk_type_of_name lthy pname name ty_args
  = Type(Local_Theory.full_name lthy (Binding.name (mkN(pname, name))), ty_args)
fun mk_mt_list t = Term.Const (@{const_name "Nil"}, mk_listT t)

fun name_of_typ (Type (s, _)) = s
  | name_of_typ (TFree _) = error "name_of_type: unexpected TFree"
  | name_of_typ (TVar _ ) = error "name_of_type: unexpected TVAR"

fun prove_UNIV name typ elems thmsN lthy =
  let
    val rhs = mk_set typ elems
    val lhs = Const("Set.UNIV",mk_setT typ)
    val stmt = mk_Trueprop (mk_eq (lhs,rhs))
    val fq_tname = name_of_typ typ
    fun inst_and_prove_enum thy =
      let
        val _ = writeln("Inst enum: ")
        val lthy = Class.instantiation ([fq_tname], [], @{sort enum}) thy
        val enum_eq = Const("Pure.eq",mk_listT typ --> mk_listT typ --> propT)
        $Const(@{const_name "enum_class.enum"},mk_listT typ)$
        $(mk_list typ elems)
        val ((_, (_, enum_def')), lthy) = Specification.definition NONE [] []
        ((Binding.name ("enum_"^name),[]), enum_eq) lthy
        val enum_def = singleton (Proof_Context.export lthy ctxt_thy) enum_def'
        val enum_all_eq = Const("Pure.eq", boolT --> boolT --> propT)
        $Const(@{const_name "enum_class.enum_all"},(typ --> boolT) --> boolT)$
        $Free("P",typ --> boolT)$(mk_list typ elems)$
        val ((_, (_, enum_all_def')), lthy) = Specification.definition NONE [] []
        ((Binding.name ("enum_all_"^name),[]), enum_all_eq) lthy
        val enum_all_def = singleton (Proof_Context.export lthy ctxt_thy) enum_all_def'
        val enum_ex_eq = Const("Pure.eq", boolT --> boolT --> propT)
        $Const(@{const_name "enum_class.enum_ex"},(typ --> boolT) --> boolT)$
        $Free("P",typ --> boolT)$(mk_setT typ elems)$
        val ((_, (_, enum_ex_def')), lthy) = Specification.definition NONE [] []
        ((Binding.name ("enum_ex_"^name),[]), enum_ex_eq) lthy
        val enum_ex_def = singleton (Proof_Context.export lthy ctxt_thy) enum_ex_def'
      in
        Class.prove_instantiation_exit (fn ctxt =>
          (Class.intro_classes_tac ctxt []) THEN
          ALLGOALS (simp_tac (ctxt addsimps [Proof_Context.get_thm ctxt (name"_UNIV"),

in
fun inst_and_prove_finite thy = 
  let
    val lthy = Class.instantiation ([fq_tname], [], @{sort finite}) thy
    in
    Class.prove_instantiation_exit (fn ctxt =>
      (Class.intro_classes_tac ctxt []) THEN
      (simp_tac (ctxt addsimps [Proof_Context.get_thm ctxt (name^"_UNIV"))]) 1) lthy
    end
  in
  lthy
  end

fun def_enum_consts (trac:TracProtocolCert.cProtocol) lthy = 
  let
    val pname = #name trac
    val defname = mkN(pname, enum_constsN)
    val _ = info(" Defining "defname)
    val enames = get_enum_consts trac
    val econsts = map (fn x => ([],x)) enames
    in
    ([defname], ml_isar_wrapper.define_simple_datatype ([], defname) econsts lthy)
  end

fun def_sets (trac:TracProtocolCert.cProtocol) lthy = 
  let
    val pname = #name trac
    val defname = mkN(pname, setsN)
    val _ = info(" Defining "defname)
    val sspec = get_set_spec trac
    val gsspec = get_general_set_family_set_spec trac
    val tfqn = full_name' enum_constsN trac lthy
    val ttyp = Type(tfqn, [])
    val eqs = map (fn (x,n) => if member (op =) gsspec (x,n) then ([],x) else (replicate n ttyp,x)) sspec
    in
    lthy
    |> ml_isar_wrapper.define_simple_datatype ([], defname) eqs
  end

fun def_funs (trac:TracProtocolCert.cProtocol) lthy = 
  let
    val pname = #name trac
    val (pub_f, pub_c, priv_c) = get_funs trac
    val pub = (map (fn (f,n) => (f,n,NONE)) pub_f)@map (fn (f,a) => (f,0,a)) pub_c
    val priv = map (fn (f,a) => (f,0,a)) priv_c
    val declared_types = #type_spec trac
    fun def_atom lthy = 
      let
        val def_atomname = mkN(pname, atomN)
      end
  end
4.6 Support for the Trac Format

val extra_types =  
  if null pub_c  
  then default_extra_types  
  else extended_extra_types
val types = declared_types@extra_types
fun define_atom_dt lthy =  
  let  
    val _ = info(" Defining "^def_atomname)  
    in  
      lthy  
      |> ml_isar_wrapper.define_simple_datatype ([], def_atomname) (map (fn x => ([],x)) types)  
    end
fun prove_UNIV_atom lthy =  
  let  
    val _ = info (" Proving "^def_atomname^"_UNIV")  
    val thmsN = [def_atomname^".exhaust"]  
    val fqn = full_name (mkN(pname, atomN)) lthy  
    val typ = Type(fqn, [])  
    in  
      lthy  
      |> prove_UNIV (def_atomname) typ (map (fn c => Const(fqn^"."^c,typ)) types) thmsN  
    end
  in  
    lthy  
    |> define_atom_dt  
    |> prove_UNIV_atom  
  end

fun def_fun_dt lthy =  
  let  
    val def_funname = mkN(pname, funN)  
    val _ = info(" Defining "^def_funname)  
    val decl_funs = map (fn x => ([],x)) (map #1 (pub@priv))  
    val enum_fun = ([Type (full_name (mkN(pname, enum_constsN)) lthy, []),enumN])  
    val all_funs = decl_funs@enum_fun::map snd (get_user_atom_spec_pre trac)@  
      map (fn (e,_,_) => ([natT],infenumN e)) (get_nonunion_infinite_enum_spec trac)  
    in  
      ml_isar_wrapper.define_simple_datatype ([], def_funname) all_funs lthy  
    end
fun def_fun_arity lthy =  
  let  
    val fqn_name = full_name (mkN(pname, funN)) lthy  
    val ctyp = Type (fqn_name, [])  
    val ctyp' = Type (full_name (mkN(pname, enum_constsN)) lthy, [])  
    val name = mkN(pname, arityN)  
    fun mk_rec_eq typs (fname,arity,_) =  
      let  
        val a = Const(fqn_name^"."^fname, typs ---> ctyp)  
        val b = fold (fn t => fn p => p$(Term.dummy_pattern t)) a typs  
      in  
        (Free(name,ctyp --> natT)$b, mk_nat(arity))  
      end
    val _ = info(" Defining "^name)  
    in  
      ml_isar_wrapper.define_simple_fun name  
      (map (mk_rec_eq []) (pub@priv))@  
        mk_rec_eq [ctyp'] (enumN,0,NONE)::
fun def_set_arity lthy = 
  let
    val fqn_name = full_name' setsN trac lthy
    val ctyp = Type (fqn_name, [])
    val ctyp' = Type (full_name' enum_constsN trac lthy, [])
    val name = mkN(pname, set_arityN)
    val sspec = get_set_spec trac
    val gsspec = get_general_set_family_set_spec trac
    val sspec' = 
      map (fn (x,n) => if member (op =) gsspec (x,n)
             then (x,n+1,[]) 
             else (x,0,replicate n ctyp')) sspec
    val _ = info(" Defining "^name)
  in
    ml_isar_wrapper.define_simple_fun name
      (map mk_rec_eq sspec')
    lthy
  end

fun def_public lthy = 
  let
    val fqn_name = full_name (mkN(pname, funN)) lthy
    val ctyp = Type (fqn_name, [])
    val ctyp' = Type (full_name (mkN(pname, enum_constsN)) lthy, [])
    val name = mkN(pname, publicN)
    fun mk_rec_eq bool_trm types fname = 
      let
        val a = Const(fqn_name"."^fname, types ---> ctyp)
        val b = fold (fn t => fn p => p$(Term.dummy_pattern t)) types a
        in
          (Free(name,ctyp --> boolT)$b, bool_trm)
        end
    fun mk_rec_eq' fname = 
      let
        val a = Const(fqn_name"."^fname, [boolT,natT] ---> ctyp)
        val b = a$Term.Free ("b", boolT)$Term.dummy_pattern natT
        in
          (Free(name,ctyp --> boolT)$b, Term.Free ("b", boolT))
        end
    val _ = info(" Defining "^name)
  in
    ml_isar_wrapper.define_simple_fun name
  end
4.6 Support for the Trac Format

```ml
((map (mk_rec_eq (@{term "False"}) [c]) (map #1 priv))
@{(map (mk_rec_eq (@{term "True"}) [c]) (map #1 pub))
@mk_rec_eq (@{term "True"}) [ctyp'] enumN
::map (mk_rec_eq' o snd o snd) (get_user_atom_spec_pre trac)
@map (fn (e,_,_) => mk_rec_eq (@{term "True"}) [natT] (infenumN e))
(get_nonunion_infinite_enum_spec trac)
) lthy
end

fun def_gamma lthy =
let
fun optionT t = Type (@{type_name "option"}, [t])
fun mk_Some t = Const (@{const_name "Some"}, t --> optionT t)
fun mk_None t = Const (@{const_name "None"}, optionT t)

val fqn_name = full_name (mkN(pname, funN)) lthy
val ctyp = Type (fqn_name, [])
val atomFQN = full_name (mkN(pname, atomN)) lthy
val atomT = Type (atomFQN, [])
val ctyp' = Type (full_name (mkN(pname, enum_constsN)) lthy, [])
val name = mkN(pname, gammaN)

fun mk_atomT_trm tau = mk_Some atomT$Const(atomFQN^"."^tau, atomT)
fun mk_rec_eq' (typname,(typs,fname)) =
let
val typtrm = case typname of NONE => mk_None atomT | SOME tau => mk_atomT_trm tau
val a = Const(fqn_name^"."^fname, typs ---> ctyp)
val b = fold (fn t => fn p => p$(Term.dummy_pattern t)) typs a
in
(Free(name,ctyp --> optionT atomT)$b, typtrm)
end
fun mk_rec_eq typname fname = mk_rec_eq' (typname,([],fname))

val user_atom_spec = get_user_atom_spec trac
val priv_rest = filter_out (member (op =) (map (snd o snd) user_atom_spec) o #1) priv
val pub_c_rest = filter_out (member (op =) (map (snd o snd) user_atom_spec) o #1) pub_c
val _ = info(" Defining "^name)
in
ml_isar_wrapper.define_simple_fun name
(map (fn (s,p) => mk_rec_eq' (SOME s,p)) user_atom_spec
@map (mk_rec_eq (SOME secret_typeN) o #1) priv_rest
@map (mk_rec_eq (SOME other_pubconsts_typeN) o #1) pub_c_rest
@mk_rec_eq' (SOME enum_typeN,([ctyp'],enumN))
::map (mk_rec_eq NONE o #1) pub_f
@map (fn (e,_,_) => mk_rec_eq' (SOME enum_typeN,([natT],infenumN e)))
(get_nonunion_infinite_enum_spec trac)
) lthy
end

fun def_ana lthy = let

fun optionT t = Type (@{type_name "option"}, [t])
fun mk_Some t = Const (@{const_name "Some"}, t --> optionT t)
fun mk_None t = Const (@{const_name "None"}, optionT t)

val fqn_name = full_name (mkN(pname, funN)) lthy
val ctyp = Type (fqn_name, [])
val ctyp' = Type (full_name (mkN(pname, enum_constsN)) lthy, [])

fun mk_atomT_trm tau = mk_Some atomT$Const(atomFQN^"."^tau, atomT)
fun mk_rec_eq' (typname,(typs,fname)) =
let
val typtrm = case typname of NONE => mk_None atomT | SOME tau => mk_atomT_trm tau
val a = Const(fqn_name^"."^fname, typs ---> ctyp)
val b = fold (fn t => fn p => p$(Term.dummy_pattern t)) typs a
in
(Free(name,ctyp --> optionT atomT)$b, typtrm)
end
fun mk_rec_eq typname fname = mk_rec_eq' (typname,([],fname))

val user_atom_spec = get_user_atom_spec trac
val priv_rest = filter_out (member (op =) (map (snd o snd) user_atom_spec) o #1) priv
val pub_c_rest = filter_out (member (op =) (map (snd o snd) user_atom_spec) o #1) pub_c
val _ = info(" Defining "^name)
in
ml_isar_wrapper.define_simple_fun name
(map (fn (s,p) => mk_rec_eq' (SOME s,p)) user_atom_spec
@map (mk_rec_eq (SOME secret_typeN) o #1) priv_rest
@map (mk_rec_eq (SOME other_pubconsts_typeN) o #1) pub_c_rest
@mk_rec_eq' (SOME enum_typeN,([ctyp'],enumN))
::map (mk_rec_eq NONE o #1) pub_f
@map (fn (e,_,_) => mk_rec_eq' (SOME enum_typeN,([natT],infenumN e)))
(get_nonunion_infinite_enum_spec trac)
) lthy
end
```
val name = mkN(pname, anaN)

val ana_outputT = mk_prodT (mk_listT keyT, mk_listT natT)

val default_output = mk_prod (mk_list keyT [], mk_list natT [])

fun mk_ana_output ks rs = mk_prod (mk_list keyT ks, mk_list natT rs)

fun mk_rec_eq ana_output_trm typs fname = 
  let
    val a = Const(fqn_name^"."^fname, typs ---> ctyp)
    val b = fold (fn t => fn p => p$(Term.dummy_pattern t)) typs a
  in
    (Free(name,ctyp --> ana_outputT)$b, ana_output_trm)
  end

val _ = info(" Defining "^name)

val ana_spec = 
  let
    fun var_to_nat is_priv_fun f xs x = 
      let
        val n = snd (Option.valOf ((list_find (fn y => y = x) xs)))
      in
        if is_priv_fun then mk_nat (1+n) else mk_nat n
      end
    fun c_to_h is_priv_fun f xs t = ana_cMsg_to_hol is_priv_fun t xs trac lthy
    fun keys is_priv_fun f ps ks = map (c_to_h is_priv_fun f ps) ks
    fun results is_priv_fun f ps rs = map (var_to_nat is_priv_fun f ps) rs
    fun aux ({head=(f,ps), keys=ks, results=rs, is_priv_fun=b}:TracProtocolCert.cAnaRule) = 
      (f, mk_ana_output (keys b f ps ks) (results b f ps rs))
  in
    map aux (#analysis_spec trac)
  end

val other_funs = 
  filter (fn f => not (List.exists (fn g => f = g) (map fst ana_spec))) (map #1 (pub@priv))
in
ml_isar_wrapper.define_simple_fun name
  (map (fn (f,out) => mk_rec_eq out [] f) ana_spec
    @map (mk_rec_eq default_output []) other_funs
    @mk_rec_eq default_output [ctyp'] enumN
     ::=map (fn (_,(type,f)) => mk_rec_eq default_output output f) (get_user_atom_spec_pre trac)
    @map (fn (e,_) => mk_rec_eq default_output [natT] (infenumN e))
    (get_nonunion_infinite_enum_spec trac)
  )
lthy
end

in
lthy |> def_atom
  |> def_fun_dt
  |> def_fun arity
  |> def_set arity
  |> def_public
  |> def_gamma
  |> def_ana
end

fun define_term_model (trac:TracProtocolCert.cProtocol) lthy = 
  let
    val _ = info("Defining term model")
in
4.6 Support for the Trac Format

```plaintext
lthy |> snd o def_enum_consts trac
    |> def_sets trac
    |> def_funs trac
end

fun define_fixpoint fp_triple trac print lthy =
  let
    val fp_name = mkN (#name trac, "fixpoint")
    val _ = info("Defining fixpoint")
    val _ = info(" Defining "^fp_name)
    val fp_triple_trm = fp_triple_to_hol fp_triple trac lthy
  in
    (trac, #2 (ml_isar_wrapper.define_constant_definition' (fp_name, fp_triple_trm) print lthy))
  end

fun define_protocol print ((trac:TracProtocolCert.cProtocol), lthy) = let
  val _ =
    if length (#transaction_spec trac) > 1
    then info("Defining protocols")
    else info("Defining protocol")
  val pname = #name trac
  val mk_Send = mk_Send_step trac lthy
  val mk_Receive = mk_Receive_step trac lthy
  val mk_InSet = mk_InSet_step trac lthy
  val mk_NotInSet = mk_NotInSet_step trac lthy
  val mk_NegChecks = mk_NegChecks_step trac lthy
  val mk_Equality = mk_Equality_step trac lthy
  val mk_Insert = mk_Insert_step trac lthy
  val mk_Delete = mk_Delete_step trac lthy

  val star_label = mk_star_label
  val prot_label = mk_prot_label

  fun mk_tname i tr =
    let
      val x = #1 tr
      val y = case i of NONE => x | SOME n => mkN(n, x)
      val z = mkN("transaction", y)
    in mkN(pname, z)
    end

  fun def_transaction name_prefix prot_num (transaction:TracProtocolCert.cTransaction) lthy = let
    val defname = mk_tname name_prefix (#transaction transaction)
    val _ = info(" Defining "^defname)

    val receives = #receive_actions transaction
    val checkssingle = #checksingle_actions transaction
    val checksall = #checkall_actions transaction
    val updates = #update_actions transaction
    val sends = #send_actions transaction
    val fresh = get_fresh_variables transaction
    val attack_signals = #attack_actions transaction
    val fresh_vars = get_fresh_variables transaction
    val nonfresher_value_vars = get_nonfresh_value_variables transaction
    val finenum_vars = get_finite_enum_variables transaction
    val enumtype_vars = get_enumtype_variables transaction
    val nonenum_vars = get_nonenum_variables transaction
    val infenum_vars = get_infinite_enum_variables transaction
    val all_decl_vars = get_transaction_head_variables transaction
    val bvars = get_bound_variables transaction
  end
```
4 Trac Support and Automation
val nonfinenum_vars =
filter (member (op =) (nonenum_vars@infenum_vars)) (all_decl_vars@fresh_vars)
val infenum_enumtype_vars =
filter (member (op =) (enumtype_vars@infenum_vars)) (all_decl_vars@fresh_vars)
val (enum_ineqs, value_ineqs) = get_variable_restrictions transaction
val enable_let_bindings = true
fun c_to_h' b trm = transaction_cMsg_to_hol
trm prot_num
(nonfinenum_vars@bvars)
b trac lthy
val c_to_h = c_to_h' enable_let_bindings
val abstract_over_enum_vars = fn x => fn y => fn z =>
abstract_over_finite_enum_vars x y z trac lthy
fun mk_transaction_term (rcvs, chcksingle, chckall, upds, snds, frsh, atcks) =
let
open Trac_Term TracProtocolCert
fun action_filter f (lbl,a) = case f a of SOME x => SOME (lbl,x) | NONE => NONE
fun lbl_to_h LabelS = star_label
| lbl_to_h LabelN = prot_label prot_num
fun lbl_trms_to_h f (lbl,ts) = f (lbl_to_h lbl) (map c_to_h ts)
val S0 =
let
val msgT = messageT trac lthy
val varT = message_varT trac lthy
val funN = full_name' funN trac lthy
val funT = funT trac lthy
val enum_constsT = enum_constsT trac lthy
val infenumspec = get_infinite_enum_spec trac
val botinfenums = map #1 (get_nonunion_infinite_enum_spec trac)
val enum_constructor = Term.Const (funN ^ "." ^ enumN, enum_constsT --> funT)
fun mk_enum_const' a = mk_enum_const a trac lthy
fun mk_union typ [] = Term.Const ("Set.empty", mk_setT typ)
| mk_union typ (t::ts) =
fold (fn s => fn u =>
u $ s) ts t
val ran_trm_finenums =
Term.Const ("Set.range", (enum_constsT --> funT) --> mk_setT funT) $
enum_constructor
fun ran_trm_botinfenum e =
Term.Const ("Set.range", (natT --> funT) --> mk_setT funT) $
Term.Const (funN ^ "." ^ infenumN e, natT --> funT)
fun ran_trm_infenums e =
case List.find (fn (a,_,_) => a = e) infenumspec of
SOME (_,cs,es) => mk_union funT (map ran_trm_botinfenum es@
(if null cs then []
else [mk_set funT (map (fn c => enum_constructor $ mk_enum_const' c) cs)]))
| NONE => error ("Couldn't find enumeration " ^ e)
fun consts (_,EnumType) =
mk_union funT (ran_trm_finenums::map ran_trm_botinfenum botinfenums)
| consts (_,InfiniteEnumeration e) = ran_trm_infenums e
| consts x = error ("Error: Expected an enumeration variable or a variable of " ^
"type " ^ enum_typeN ^ ", but got " ^ cMsg_str (cVar x))

354


fun var_trm x = 
  Term.Const (@{const_name "the_Var"}, msgT --> varT) $ c_to_h (cVar x) 
in 
  map (fn x => mk_prod (var_trm x, consts x)) infenum_enumtype_vars 
end

val S1 = map (lbl_trms_to_h mk_Receive) 
  (map_filter (action_filter maybe_the_Receive) rcvs)

val S2 = 
let 
  fun aux (lbl,cEquality (pcv,(x,y))) = 
    SOME (mk_Equality pcv (lbl_to_h lbl) (c_to_h x) (c_to_h y)) 
  | aux (lbl,cInSet (pcv,(e,s))) = 
    SOME (mk_InSet pcv (lbl_to_h lbl) (c_to_h e) (c_to_h s)) 
  | aux (lbl,cNegChecks (xs,ns)) = 
    let 
      fun f (a,b) = (c_to_h a, c_to_h b) 
      val ineqs = map f (map_filter maybe_the_Inequality ns) 
      val notins = map f (map_filter maybe_the_NotInSet ns) 
      val bvars = map (c_to_h o cVar) xs 
      in 
        SOME (mk_NegChecks (lbl_to_h lbl) bvars ineqs notins) 
      end 
  | aux _ = NONE 
  in 
    map_filter aux chcksingle
  end

val S3 = 
let 
  fun arity s = case set_arity trac s of 
    SOME n => n 
  | NONE => error ("Error: Not a set family: " ^ s) 
  fun mk_evs s = 
    map (fn n => ("X" ^ Int.toString n, Untyped)) (0 upto ((arity s) -1)) 
  fun mk_trm (lbl,e,s) = 
    let 
      val ps = map (fn x => cVar (x,EnumType)) (map fst (mk_evs s)) 
      in 
        mk_NotInSet (lbl_to_h lbl) (c_to_h e) (c_to_h (cSet (s,ps))) 
      end 
  fun mk_trms (lbl,(e,s)) = 
    abstract_over_enum_vars (mk_evs s) [] (mk_trm (lbl,e,s)) 
  in 
    map mk_trms (map_filter (action_filter maybe_the_NotInAny) chckall)
  end

val S4 = map (c_to_h o cVar) frsh

val S5 = 
let 
  fun aux (lbl,cInsert (e,s)) = SOME (mk_Insert (lbl_to_h lbl) (c_to_h e) (c_to_h s)) 
  | aux (lbl,cDelete (e,s)) = SOME (mk_Delete (lbl_to_h lbl) (c_to_h e) (c_to_h s)) 
  | aux _ = NONE 
  in 
    map_filter aux upds
  end

val S6 =
let val snds' = map_filter (action_filter maybe_the_Send) snds
  in map (lbl_trms_to_h mk_Send) (snds' @ map (fn (lbl,_) => (lbl,[cAttack])) atcks) end

in mk_Transaction trac lthy S0 S1 S2 S3 S4 S5 S6

|> abstract_over_enum_vars finenum_vars enum_ineqs
|> (fn trm =>
  if not (null nonenum_vars) andalso enable_let_bindings
  then let
    val typof = Term.fastype_of
    val xs = nonfinenum_vars@bvars
    val a = absfreeprod (messageT trac lthy) (map fst xs) trm
    val b = mk_tuple (map (c_to_h' false o cVar) xs)
    val c = Term.Const (@{const_name "Let"}, [typof b, typof a] ---> typof (a$b))
  in c$b$a end
  else trm)
end

fun def_trm trm print lthy =
  #2 (ml_isar_wrapper.define_constant_definition' (defname, trm) print lthy)

val additional_value_ineqs =
  let
    open Trac_Term TracProtocolCert
    val poschecks = map_filter (maybe_the_InSet o snd) checkssingle
    val negchecks_single = List.concat (map (map_filter maybe_the_NotInSet o snd)
      (map_filter (maybe_the_NegChecks o snd) checkssingle))
    val negchecks_all = map_filter (maybe_the_NotInAny o snd) checksall
    fun aux' (cVar (x,ValueType),s) (cVar (y,ValueType),t) =
      if s = t then SOME (x,y) else NONE
    / aux' _ _ = NONE
    fun aux (x,cSet (s,ps)) = SOME (map_filter (aux' (x,cSet (s,ps))) negchecks_single@
      map_filter (aux' (x,s)) negchecks_all
    )
    / aux _ = NONE
  in
    List.concat (map_filter aux poschecks)
  end

val all_value_ineqs = distinct (op =) (value_ineqs@additional_value_ineqs)

val valvarsprod =
  filter (fn p => not (List.exists (fn q => p = q orelse swap p = q) all_value_ineqs))
  (list_triangle_product (fn x => fn y => (x,y)) nonfresh_value_vars)

val transaction_trm0 = mk_transaction_term
  (receives, checkssingle, checksall, updates, sends, fresh, attack_signals)
in
if null valvarsprod
  then def_trm transaction_trm0 print lthy
else let
  open Trac_Term TracProtocolCert
  val partitions = list_partitions nonfresh_value_vars all_value_ineqs
  val ps = filter (not o null) (map (filter (fn x => length x > 1)) partitions)

  fun mk_subst ps =
    let
      fun aux [] = NONE
      / aux (x::xs) = SOME (map (fn y => (y,cVar (x,ValueType))) xs)
    in
      mk_subst ps
    end
  in
    def_trm transaction_trm0 print lthy
  end


4.6 Support for the Trac Format

List.concat (map_filter aux ps)
end

fun apply d =
  let
    val ap = subst_apply cActions d
    val checksingle' =
      filter (fn ((_,a)) => case a of
            cNegChecks ([],[cInequality (x,y)]) => x <> y
            | _ => true)
      (ap checkssingle)
    in
      (ap receives, checksingle', ap checksall, ap updates, ap sends, fresh, attack_signals)
  end

val transaction_trms = transaction_trm0::map (mk_transaction_term o apply o mk_subst) ps
val transaction_typ = Term.fastype_of transaction_trm0

fun mk_concat_trm tau trms =
  Term.Const (@{const_name "concat"}, mk_listT tau --> tau) $ mk_list tau trms
in
  def_trm (mk_concat_trm transaction_typ transaction_trms) print lthy
end

val def_transactions =
  let
    val prots = map (fn (n,pr) => map (fn tr => (n,tr)) pr) (#transaction_spec trac)
    val lbls = list_upto (length prots)
    val lbl_prots = List.concat (map (fn i => map (fn tr => (i,tr)) (nth prots i)) lbls)
    val f = fold (fn (i,(n,tr)) => def_transaction n i tr)
  in
    f lbl_prots
  end

val def_protocols lthy = let
  fun mk_prot_def (name,trm) lthy =
    let val _ = info(" Defining ",name)
    in #2 (ml_isar_wrapper.define_constant_definition' (name,trm) print lthy)
    end
  val prots = #transaction_spec trac
  val num_prots = length prots
  val pdefname = mkN(pname, "protocol")
  fun mk_tnames i =
    let
      val trs = case nth prots i of (j,prot) => map (fn tr => (j,tr)) prot
      in map (fn (j,s) => full_name (mk_tname j (#transaction s)) lthy) trs
    end
  val tnames = List.concat (map mk_tnames (list_upto num_prots))
  val pnames =
    let
      val f = fn i => (Int.toString i,nth prots i)
      val g = fn (i,(n._)) => case n of NONE => i | SOME m => m
      val h = fn s => mkN (pdefname,s)
    in map (h o g o f) (list_upto num_prots)
    end
  val trtyp = prot_transactionT trac lthy
val trstyp = mk_listT trtyp

fun mk_prot_trm names =  
  Term.Const (@{const_name "concat"}, mk_listT trstyp --> trstyp) $  
  mk_list trstyp (map (fn x => Term.Const (x, trstyp)) names)

val lthy =  
  if num_prots > 1  
  then fold (fn (i,pname) => mk_prot_def (pname, mk_prot_trm (mk_tnames i)))  
    (map (fn i => (i, nth pnames i)) (list_upto num_prots)) lthy  
  else lthy

val pnames' = map (fn n => full_name n lthy) pnames

fun mk_prot_trm_with_star i =  
  fun f j =  
    if j = i  
    then Term.Const (nth pnames' j, trstyp)  
    else (Term.Const (@{const_name "map"}, [trtyp --> trtyp, trstyp] ---> trstyp) $  
          Term.Const (@{const_name "concat"}, mk_listT trstyp --> trstyp) $  
          Term.Const (nth pnames' j, trstyp))
  in  
    Term.Const (@{const_name "concat"}, mk_listT trstyp --> trstyp) $  
    mk_list trstyp (map f (list_upto num_prots))
  end

fun mk_star_prot_trm () =  
  fun f j =  
    (Term.Const (@{const_name "map"}, [trtyp --> trtyp, trstyp] ---> trstyp) $  
     Term.Const (@{const_name "concat"}, mk_listT trstyp --> trstyp) $  
     Term.Const (nth pnames' j, trstyp))
  in  
    Term.Const (@{const_name "concat"}, mk_listT trstyp --> trstyp) $  
    mk_list trstyp (map f (list_upto num_prots))
  end

val lthy =  
  if num_prots > 1  
  then fold (fn (i,pname) => mk_prot_def (pname, mk_prot_trm_with_star i))  
    (map (fn i => (i, nth pnames i ^ "_with_star_projs")) (list_upto num_prots)) lthy  
  else lthy

val lthy =  
  if num_prots > 1  
  then mk_prot_def (pdefname ^ "_star_projs", mk_star_prot_trm ()) lthy  
  else lthy
  in  
    mk_prot_def (pdefname, mk_prot_trm (if num_prots > 1 then pnames' else tnames)) lthy
  end
  in  
    mk_prot_def (pdefname, mk_prot_trm (if num_prots > 1 then pnames' else tnames)) lthy
  end
  in  
    (trac, lthy |> def_transactions |> def_protocols)
  end
end

structure trac = struct
  open Trac_Term
val info = Output.information

fun mk_abs_filename thy filename = 
  let
    val filename = Path.explode filename
    val master_dir = Resources.master_directory thy
  in
    Path.implode (if (Path.is_absolute filename)
      then filename
      else Path.append master_dir filename)
  end

fun def_fp print (trac:TracProtocolCert.cProtocol, lthy) = 
  case #fixed_point trac of
  SOME fp => trac_definitorial_package.define_fixpoint fp trac print lthy
  | NONE => (trac, lthy)
  (* let
    val fp = TracFpParser.parse_str fp_str 
    val (trac,lthy) = trac_definitorial_package.define_fixpoint fp trac print lthy
    val lthy = Local_Theory.raw_theory (update trac) lthy
  in
    (trac, lthy)
  end *)

fun def_trac_term_model trac lthy = 
  let
    val lthy:local_theory = trac_definitorial_package.define_term_model trac lthy
  in
    (trac, lthy)
  end

val def_trac_protocol = trac_definitorial_package.define_protocol

fun def_trac trac_str opt_fp_str print lthy = 
  let
    val trac = TracProtocolParser.parse_str trac_str
    val trac = case opt_fp_str of
      SOME fp_str =>
        TracProtocol.update_fixed_point trac (SOME (TracFpParser.parse_str fp_str))
      | NONE => trac
    val lthy = Local_Theory.raw_theory (trac_definitorial_package.update trac) lthy
    val ctrac = TracProtocolCert.certifyProtocol trac
  in
    (def_fp print o def_trac_protocol print o def_trac_term_model ctrac) lthy
  end

fun def_trac_file trac_filename opt_fp_filename print lthy = 
  let
    fun read_file filename = 
      File.read (Path.explode (mk_abs_filename (Proof_Context.theory_of lthy) filename))
    val trac_str = read_file trac_filename
    val opt_fp_str = Option.map (fn fp_filename => read_file fp_filename) opt_fp_filename
    val (trac,lthy) = def_trac trac_str opt_fp_str print lthy
  in
    (trac, lthy)
  end

ML

val fileNameP = Parse.name -- Parse.name
4 Trac Support and Automation

```ml
val _ = Outer_Syntax.local_theory' @{command_keyword "trac"}
"Define protocol and (optionally) fixpoint using trac format."
((Parse.cartouche -- Scan.optional Parse.cartouche "" >> (  
  fn (trac,fp) => fn print => fn lthy =>
  let
  val opt_fp = if fp = "" then NONE else SOME fp
  val trac = trac.def_trac trac opt_fp print #> snd
  in
  trac_time.ap_lthy lthy ("trac") trac lthy
end)))

val _ = Outer_Syntax.local_theory' @{command_keyword "trac_import"}
"Import protocol and (optionally) fixpoint from trac files."
((Parse.name -- Scan.optional Parse.name "" >> (  
  fn (trac_filename, fp_filename) => fn print => fn lthy =>
  let
  val opt_fp_filename = if fp_filename = "" then NONE else SOME fp_filename
  val trac = trac.def_trac_file trac_filename opt_fp_filename print #> snd
  in
  trac_time.ap_lthy lthy ("trac_import") trac lthy
end)))

val name_prefix_parser = Parse.!!! (Parse.name --| Parse.$$$ ":" -- Parse.name)

(* Original definition (opt_evaluator) copied from value_command.ml *)
val opt_proof_method_choice =  
  Scan.optional (keyword [ ]|-- Parse.name --| keyword [ ]|-- Parse.name) "safe";

(* Original definition (locale_expression) copied from parse_spec.ML *)
val security_proof_locale_opt_defs_list = Scan.optional  
  (keyword for|-- Scan.repeat1 Parse.name >> (fn xs => if length xs > 3 then error "Too many optional arguments" else xs))
  [];
val composed_protocol_locale_defs_list =  
  (keyword for|-- Parse.!!! (Parse.name -- (* The composed protocol *)
               Parse.name -- (* Its SMP set *)
               Parse.name)) -- (* The (symbolic) list of shared secrets *)
  (keyword<and|-- Scan.repeat1 Parse.name >> (fn xs => if length xs < 2 then error "Too few arguments" else xs)) --
  (keyword<and|-- Scan.repeat1 Parse.name >> (fn xs => if length xs < 2 then error "Too few arguments" else xs)) --
  (keyword<and|-- Scan.repeat1 Parse.name >> (fn xs => if length xs < 2 then error "Too few arguments" else xs)) --

val composed_protocol_locale_parser =  
  name_prefix_parser -- composed_protocol_locale_defs_list
val composed_protocol_locale_parser_with_method_choice =  
  opt_proof_method_choice -- name_prefix_parser -- composed_protocol_locale_defs_list

fun protocol_model_setup_proof_state name prefix lthy =  
  let
    fun f x y z = (((x,Position.none),((y,true),(Expression.Positional z,[]))),[])
  in
```

---
4.6 Support for the Trac Format

val _ = assert_nonempty_name name
val pexpr = f "stateful_protocol_model" name (protocol_model_interpretation_params prefix lthy)
val pdefs = protocol_model_interpretation_defs name
val proof_state = Interpretation.global_interpretation_cmd pexpr pdefs lthy
in
proof_state
end

fun protocol_security_proof_defs manual_proof name prefix opt_defs lthy =
  let
    fun f x y z = (((x,Position.none),((y,true),(Expression.Positional z,[[]]))),[])
    val _ = assert_nonempty_name name
    val num_defs = length opt_defs
    val pparams = protocol_model_interpretation_params prefix lthy
    val default_defs = [prefix ^ "_" ^ "protocol", prefix ^ "_" ^ "fixpoint"]
    fun g locale_name extra_params = f locale_name name (pparams@map SOME extra_params)
    val (prot_fp_smp_names, pexpr) = if manual_proof
      then (case num_defs of
        0 => (default_defs, g "secure_stateful_protocol''''" default_defs)
      | 1 => (opt_defs, g "secure_stateful_protocol''" opt_defs)
      | 2 => (opt_defs, g "secure_stateful_protocol''''" opt_defs)
      | _ => (opt_defs, g "secure_stateful_protocol" opt_defs))
    else (case num_defs of
      0 => (default_defs, g "secure_stateful_protocol''''" default_defs)
    | 1 => (opt_defs, g "secure_stateful_protocol''" opt_defs)
    | 2 => (opt_defs, g "secure_stateful_protocol''''" opt_defs)
    | _ => (opt_defs, g "secure_stateful_protocol" opt_defs))
    val _ = assert_all_defined lthy prefix prot_fp_smp_names
  in
    (prot_fp_smp_names, pexpr)
  end

fun protocol_security_proof_proof_state manual_proof name prefix opt_defs print lthy =
  let
    val (prot_fp_smp_names, pexpr) = protocol_security_proof_defs manual_proof name prefix opt_defs lthy
    val proof_state = lthy |> declare_protocol_checks print
      |> Interpretation.global_interpretation_cmd pexpr []
  in
    (prot_fp_smp_names, proof_state)
  end

fun protocol_composition_proof_defs name prefix remaining_params lthy =
  let
    fun f x y z = (((x,Position.none),((y,true),(Expression.Positional z,[[]]))),[])
    fun g xs = "[" ^ String.concatWith ", " xs ^ "]"
    fun h xs = g (map_index (fn (i,x) => "(" ^ Int.toString i ^ ", " ^ x ^ ")") xs)
    val _ = assert_nonempty_name name
    val (((((pc,smp),sec),ps),psstarprojs),gsmps) = remaining_params
    val _ = assert_all_defined lthy prefix (pc,smp,sec]@ps@psstarprojs@gsmps)
    val _ = if length ps = length psstarprojs andalso length ps = length gsmps then ()
      else error "Missing arguments"
    val pparams = protocol_model_interpretation_params prefix lthy
    val params = [pc, g ps, g psstarprojs, smp, sec, h gsmps]
    val pexpr = f "composable_stateful_protocols" name (pparams@map SOME params)
    in
      pexpr
    end

fun protocol_composition_proof_proof_state name prefix params print lthy =
  let
    val pexpr = protocol_composition_proof_defs name prefix params lthy
    val state = lthy |> (declare_protocol_check "wellformed_composable_protocols" print #>
val select_proof_method_error_prefix = "Error: Invalid option: 

fun select_proof_method _ "safe" = "check_protocol"
| select_proof_method _ "nbe" = "check_protocol_nbe"
| select_proof_method _ "unsafe" = "check_protocol_unsafe"
| select_proof_method msg opt_meth_level = error (select_proof_method_error_prefix ^ opt_meth_level ^ 

"Valid options:
1. safe: Instructs Isabelle to " ^ msg ^ " using \"code_simp\".
2. nbe: Instructs Isabelle to use \"normalization\" instead of \"code_simp\".
3. unsafe: Instructs Isabelle to use \"eval\" instead of \"code_simp\".")

val _ = Outer_Syntax.local_theory command_keyword<protocol_model_setup>
"prove interpretation of protocol model locale into global theory"
(name_prefix_parser >> (fn (name,prefix) => fn lthy =>
let fun protocol_model_setup ((name,prefix),lthy) =

let
val proof_state = protocol_model_setup_proof_state name prefix lthy
val meth =
let
val m = "protocol_model_interpretation"
val _ = Output.information ("Proving protocol model locale instance with proof method " ^ m)
in Method.Source (Token.make_src (m, Position.none) [])
end

in
ml_isar_wrapper.prove_state_simple meth proof_state
end

in
trac_time.ap_lthy lthy ("protocol_model_setup (" ^ name ^ ")) protocol_model_setup ((name,prefix),lthy)
end);

val _ = Outer_Syntax.local_theory_to_proof command_keyword<manual_protocol_model_setup>
"prove interpretation of protocol model locale into global theory"
(name_prefix_parser >> (fn (name,prefix) => fn lthy =>
let
val proof_state = protocol_model_setup_proof_state name prefix lthy
val subgoal_proof = " subgoal by protocol_model_subgoal
val _ = Output.information ("Example proof:
Active.sendback_markup_command (" apply unfold_locales
subgoal_proof
subgoal_proof
subgoal_proof
subgoal_proof
" done")
in proof_state
end));

val _ = Outer_Syntax.local_theory' command_keyword<protocol_security_proof>
"prove interpretation of secure protocol locale into global theory"
(security_proof_locale Parser_with_method_choice >>
(fn params => fn print => fn lthy =>
let
val ((_,(name,prefix)),opt_defs) = params
fun protocol_security_proof (params, print, lthy) =
let
val ((opt_meth_level,(name,prefix)),opt_defs) = params
val (defs, proof_state) =
protocol_security_proof_proof_state false name prefix opt_defs print lthy
val num_defs = length defs
val meth =
let
val m = select_proof_method "prove the protocol secure" opt_meth_level
val info = Output.information
val _ = info ("Proving security of protocol " ^ nth defs 0^
  " with proof method " ^ m)
val _ = if num_defs > 1 then info ("Using fixed point " ^ nth defs 1) else ()
val _ = if num_defs > 2 then info ("Using SMP set " ^ nth defs 2) else ()
in
Method.Source (Token.make_src (m, Position.none) [])
end
in
ml_isar_wrapper.prove_state_simple meth proof_state
end
fun protocol_security_proof_with_error_messages (params, print, lthy) =
protocol_security_proof (params, print, lthy)
handle
ERROR msg =>
if String.isPrefix "Duplicate fact declaration" msg
then error ("Failed to finalize proof because of duplicate fact declarations.\n" ^
  "This might happen if \"" ^ name ^ "\" was used previously.\n" ^
  \"\n\nOriginal error message:\n" ^ msg)
else if String.isPrefix select_proof_method_error_prefix msg
then error msg
else (* if String.isPrefix "Wellsortedness error" msg orelse
String.isPrefix "Failed to finish proof" msg orelse
String.isPrefix "error in proof state" msg
then *)
let
val (def_names,_) = protocol_security_proof_defs false name prefix opt_defs lthy
val (prot_name,fp_name,smp_name) = case length def_names of
  0 => (prefix"_protocol", prefix"_fixpoint", prefix"_SMP")
  1 => (nth def_names 0, prefix"_fixpoint", prefix"_SMP")
  2 => (nth def_names 0, nth def_names 1, prefix"_SMP")
  _ => (nth def_names 0, nth def_names 1, nth def_names 2)
in
error ("Failed to prove the protocol secure.\n" ^
"Click on the following to inspect which parts of the proof failed:\n" ^
Active.sendback_markup_command (if length def_names < 2
then "— First compute a fixed-point\n" ^
  "compute_fixpoint " ^ prot_name ^ "fp_name"\n" ^
else "")
  "— Is the fixed point free of attack signals?\n" ^
  "value \"attack_notin_fixpoint " ^ fp_name ^ "\n" ^
  "— Is the protocol covered by the fixed point?\n" ^
  "value \"protocol_covered_by_fixpoint " ^ fp_name ^ "\n" ^
  "— Is the fixed point analyzed?\n" ^
  "value \"analyzed_fixpoint " ^ fp_name ^ "\n" ^
  "— Is the protocol well-formed?\n" ^
(if length def_names < 3

then "value "wellformed_protocol \"prot_name\""\n\n" else "value "wellformed_protocol\ "prot_name\" "$smp_name\""\n\n"

"— Is the fixed point well-formed?\n
"value "wellformed_fixpoint \"fp_name\"\n
\nOriginal error message: \n
end

(* else error msg *)
in
trac_time.ap_lthy lthy (\"protocol_security_proof \"name\"\")
protocol_security_proof_with_error_messages (params, print, lthy)
end));

val _ = Outer_Syntax.local_theory_to_proof' command_keyword<manual_protocol_security_proof>
"prove interpretation of secure protocol locale into global theory"
(security_proof_locale_parser >> (fn params => fn print => fn lthy =>
let
val ((name,prefix),opt_defs) = params
val (defs, proof_state) =
protocol_security_proof_proof_state true name prefix opt_defs print lthy
val subgoal_proof =
let
val m = "code_simp" (* case opt_meth_level of
"safe" => "code_simp"
| "nbe" => "normalization"
| "unsafe" => "eval"
| _ => error (\"Invalid option: \"^ opt_meth_level \"\")
in
  subgoal by "^ m "\nend
val _ = Output.information (\"Example proof:
" ^
Active.sendback_markup_command (\" apply check_protocol_intro\n" ^
subgoal_proof
(if length defs = 1 then ""
else subgoal_proof
subgoal_proof
subgoal_proof
subgoal_proof)
" done\n")
in
proof_state
end
));

val _ = Outer_Syntax.local_theory' command_keyword<protocol_composition_proof>
"prove interpretation of composed protocol locale into global theory"
(composed_protocol_locale_parser_with_method_choice >> (fn params => fn print => fn lthy =>
let val ((_,(name,_)),_) = params
fun protocol_composition_proof (params,lthy) =
let
val ((opt_meth_level,(name,prefix)),remaining_params) = params
val proof_state =
protocol_composition_proof_proof_state name prefix remaining_params print lthy
val meth =
let
val m = select_proof_method "use" opt_meth_level
val _ = Output.information ("Proving composability of protocol \"^ name \" with proof method \"^ m")
in
Method.Source (Token.make_src (m, Position.none) [])
end
in

4.6 Support for the Trac Format

ml_isar_wrapper.prove_state_simple meth proof_state
end
in
trac_time.ap_lthy lthy
(\"protocol\composition\_proof (\"\"name\")\")
protocol\composition\_proof (params,lthy)
end));
val _ =
Outer_Syntax.local_theory_to_proof\' command_key\wedge\langle manual\_protocol\_composition\_proof \rangle
"prove interpretation of composed protocol locale into global theory"
(composed\_protocol\_locale\_parser >> (fn params => fn print => fn lthy =>
let
val ((name,prefix),remaining\_params) = params
val proof\_state =
protocol\_composition\_proof\_proof\_state name prefix remaining\_params print lthy
val subgoal\_proof = \" subgoal by code_simp\\n\"
val _ = Output.information (\"Example proof:\n\"
Active.sendback_markup_command (\" apply check\_protocol\_intro\\n"^subgoal\_proof^subgoal\_proof^subgoal\_proof^subgoal\_proof^subgoal\_proof^" done\\n\")
in
proof\_state
end )
);>

ML<
fun listterm\_to\_list lthy (Const (\"List.list.Nil",_)) = []
| listterm\_to\_list lthy ((Const (\"List.list.Cons",_) $ t1) $ t2) = t1::listterm\_to\_list lthy t2
| listterm\_to\_list lthy t =
error (\"Unexpected term (expected a list constructor): \" ^ Syntax.string\_of\_term lthy t)
fun pairterm\_to\_pair lthy t = (x,y)
| pairterm\_to\_pair lthy t = error (\"Error: Expected a pair term but got \" ^ Syntax.string\_of\_term lthy t)

fun constexpr\_to\_string lthy protocol t = let
val trac = trac_definitorial\_package.lookup\_trac protocol lthy
val trac\_name = #name trac
val sets\_type\_name = Local\_Theory.full\_name lthy (Binding.name (trac\_name ^ \"\_sets\")
val enum\_type\_name = Local\_Theory.full\_name lthy (Binding.name (trac\_name ^ \"\_enum\_consts\")
val fun\_type\_name = Local\_Theory.full\_name lthy (Binding.name (trac\_name ^ \"\_fun\")
val atom\_type\_name = Local\_Theory.full\_name lthy (Binding.name (trac\_name ^ \"\_atom\")
fun print\_constexpr x =
if String.isPrefix sets\_type\_name x
then String.extract (x,size sets\_type\_name+1,NONE)
else if String.isPrefix enum\_type\_name x
then String.extract (x,size enum\_type\_name+1,NONE)
else if String.isPrefix fun\_type\_name x
then String.extract (x,size fun\_type\_name+1,NONE)
else if String.isPrefix atom\_type\_name x
then String.extract (x,size atom\_type\_name+1,NONE)
else error (\"Unexpected constant expression: \" ^ x)
in print\_constexpr t end
fun setexpr\_to\_string lthy protocol t = let
fun err msg t = error (msg ^ " ^ Syntax.string_of_term lthy t)

fun print_set_expr' (Const (x, _)) = [constexpr_to_string lthy protocol x]
| print_set_expr' (t1 $ t2) = print_set_expr' t1 @ print_set_expr' t2
| print_set_expr' t = err "Unexpected set expression subterm" t

fun print_set_expr t = 
  case print_set_expr' t of 
    [x] => x 
  | x::xs => x ^ "(" ^ String.concatWith "," xs ^ ")" 
  | _ => err "Unexpected set expression" t
in print_set_expr t end

fun abstractionexpr_to_list lthy protocol t = let 
  fun print_abs (Const ("Orderings.bot_class.bot", _)) = [] 
    | print_abs (t $ Const ("Orderings.bot_class.bot", _)) = print_abs t 
    | print_abs (Const ("Set.insert", _) $ t) = [setexpr_to_string lthy protocol t] 
    | print_abs (t1 $ t2) = print_abs t1 @ print_abs t2 
    | print_abs t = error ("Unexpected abstract value expression: " ^ Syntax.string_of_term lthy t)
  in print_abs t end

fun protterm_to_string_no_eval var_printer protocol protterm lthy = let 
  fun print_raw (Const (x, _)) = "Const (" ^ x ^ ",_"") 
    | print_raw (t $ s) = "(" ^ print_raw t ^ " $ " ^ print_raw s ^ ")" 
    | print_raw _ = "_"
  fun err msg t = error (msg ^ " ^ Syntax.string_of_term lthy t ^ "\n" ^ print_raw t)
  val trac = trac_definitorial_package.lookup_trac protocol lthy 
  val trac_name = #name trac 
  val trac_fun_spec = Option.getOpt (#function_spec trac, {private = [], public = []}) 
  val is_priv_fun = member (fn (s, t) => s = #1 t) (#private trac_fun_spec) 
  val fun_type_name = Local_Theory.full_name lthy (Binding.name (trac_name ^ "_fun")) 
  val print_list = listterm_to_list lthy 
  val print_const_expr = constexpr_to_string lthy protocol 
  val print_set_expr = setexpr_to_string lthy protocol 
  fun print_abs t = 
    let val a = abstractionexpr_to_list lthy protocol t 
    in "val (" ^ (if a = [] then "0" else String.concatWith "," a ^ ")") end 
  fun print_trm (Const ("Term.term.Var", _) $ t) = (case t of 
    ((Const ("Product_Type.Pair", _) $ _) $ _) => var_printer (pairterm_to_pair lthy t) 
    | _ => print_trm t) 
    | print_trm ((Const ("Term.term.Fun", _) $ (Const ("Transactions.prot_fun.Abs", _) $ t)) $ 
      Const ("List.list.Nil", _)) = print_trm (print_trm (Const ("List.list.Cons", _)$ t) $ Const ("List.list.Nil", _)) 
    ) = print_abs t 
    | print_trm ( 
      (Const ("Term.term.Fun", _)$Const ("Transactions.prot_fun.OccursFact", _)) 
    $((Const ("List.list.Cons", _)$Const ("Transactions.prot_fun.OccursSec", _)) 
      $Const ("List.list.Nil", _))) 
    $((Const ("List.list.Cons", _)$ t) $ Const ("List.list.Nil", _))) 
    ) = "occurs(" ^ print_trm t ^ "")" 
    | print_trm ( 
      (Const ("Term.term.Fun", _) $ (Const ("Transactions.prot_fun.Attack", _) $ _)) $ 
      _ ) = "attack" 
    | print_trm ( 
      (Const ("Term.term.Fun", _) $ (Const ("Transactions.prot_fun.Set", _) $ t)) $ ts 
    ) = let val gs = map print_trm (print_list ts) 
      in (case gs of 

4.6 Support for the Trac Format

[] => print_set_expr f
| xs => print_trm f "("^ String.concatWith ","^ xs ^ ")"
end

| print_trm (Const ("Term.term.Fun",_) $ (Const ("Transactions.prot_fun.Fu",_) $ f)) $ ts
  ) = let val g = print_trm f; val gs = map print_trm (print_list ts)
in (case (if is_priv_fun g then (case gs of [] => [] | _::gs' => gs') else gs) of
  [] => g
| xs => g ^ "("^ String.concatWith ","^ xs ^ ")"
end

| print_trm (Const ("Transactions.prot_atom.Atom",_) $ Const (t,_)) = print_const_expr t
| print_trm ((Const ("Product_Type.Pair",_) $ t1) $ t2) = "("^ print_abs t1 ^ ","^ print_abs t2 ^ ")"
| print_trm (Const (x,ty) $ Const (y,ty)) =
if x = fun_type_name ^ ".enum"
then print_const_expr y
else err "Unexpected protocol/fixpoint term" (Const (x,ty) $ Const (y,ty))
| print_trm (Const (x,_) ) =
if String.isPrefix "Transactions.prot_atom" x
then String.extract (x,size "Transactions.prot_atom"+1,NONE)
else print_const_expr x
| print_trm t = err "Unexpected protocol/fixpoint term" t;
in
print_trm protterm end

fun prottermtype_to_string_no_eval var_printer protocol protterm lthy =
protterm_to_string_no_eval var_printer protocol protterm lthy

fun fixpoint_to_string protocol fixpoint lthy = let
fun err msg t = error (msg ^ ": "^ Syntax.string_of_term lthy t)
val fpterm = "let (FP,_,TI) = ("^ fixpoint ^ ") in (FP,TI)"

fun print_fp' s = protterm_to_string_no_eval
(fn _ => error "Unexpected term variable in fixpoint")
protocol s lthy

fun print_fp ((Const ("Product_Type.Pair",_) $ t1) $ t2) = String.concatWith "\n" (map print_fp' (listterm_to_list lthy t1)@map (fn x => "implies" ^ print_fp' x) (listterm_to_list lthy t2))
in
(print_fp o eval_term lthy o Syntax.read_term lthy) fpterm end

fun transaction_label_to_string t lthy = let
fun err msg t = error (msg ^ ": "^ Syntax.string_of_term lthy t)

fun print_label (Const ("Labeled_Strands.strand_label.LabelN",_) $ _) = " "
(* Syntax.string_of_term lthy t *)
| print_label (Const ("Labeled_Strands.strand_label.LabelS",_)) = "*"
| print_label t = err "Unexpected action label term" t
in
print_label t end

fun transaction_action_to_string var_printer protocol t lthy = let
fun err msg t = error (msg ^ ": "^ Syntax.string_of_term lthy t)

fun print_action (Const ("Stateful_Strands.stateful_strand_step.Send",_) $ ts)
  ) = "send "^ String.concatWith ","^ (map print_trm (listterm_to_list lthy ts))
in
(print_action o eval_term lthy o Syntax.read_term lthy) t
end
4  Trac Support and Automation

\[
\begin{align*}
&= \text{"receive "} ^\wedge \text{String.concatWith ", "} (\text{map print_trm (listterm_to_list lthy ts)})
\end{align*}
\]

\[
| \text{print_action (}(\text{Const ("Stateful_Strands.stateful_strand_step.Equality", _) $ _) $ t1) $ t2) \rangle \rightarrow \text{print_trm t1} \wedge \text{"== "} \text{print_trm t2}
\]

\[
| \text{print_action (}(\text{Const ("Stateful_Strands.stateful_strand_step.InSet", _) $ _) $ t1) $ t2) \rangle \rightarrow \text{"insert "} \text{print_trm t1} \wedge \text{"in "} \text{print_trm t2}
\]

\[
| \text{print_action (}(\text{Const ("Stateful_Strands.stateful_strand_step.Delete", _) $ _) $ t1) $ t2) \rangle \rightarrow \text{"delete "} \text{print_trm t1} \wedge \text{" delete "} \text{print_trm t2}
\]

\[
| \text{print_action (}(\text{Const ("Stateful_Strands.stateful_strand_step.NegChecks", _) $ xs) $ ts1) $ ts2) \rangle \rightarrow \text{let fun f (a,b) = print_trm a \wedge \text{"not in "} \text{print_trm b}
\]

\[
| \text{fun g (a,b) = print_trm a \wedge \text{print_trm b}}
\]

\[
| \text{val ys = map print_trm (listterm_to_list lthy xs)}
\]

\[
| \text{val ss1 = map (f o pairterm_to_pair lthy) (listterm_to_list lthy ts1)}
\]

\[
| \text{val ss2 = map (g o pairterm_to_pair lthy) (listterm_to_list lthy ts2)}
\]

\[
\text{in (if ys = [] then "\emptyset" else \text{forall "} \text{String.concatWith ", " ys \wedge "")} ^\wedge \text{String.concatWith ", or "} (\text{ss1@ss2})
\]

\[
| \text{end}
\]

\[
| \text{print_action t = err "Unexpected transaction action term" t}
\]

\[
\text{in print_action t end}
\]

\[
\text{fun transaction_labeled_action_to_string var_printer protocol t lthy = let}
\]

\[
\text{fun err msg t = error (msg \wedge \text{" "} Syntax.string_of_term lthy t)}
\]

\[
\text{fun print_laction (}(\text{Const ("Product_Type.Pair", _) $ t1) $ t2) = 
\text{\"transaction_label_to_string t1 lthy \wedge \" \text{transaction_action_to_string var_printer protocol t2 lthy}
\]

\[
| \text{in print_laction t = err "Unexpected labeled transaction action term" t}
\]

\[
\text{in print_laction t end}
\]

\[
\text{fun transaction_to_string var_printer protocol transactionterm lthy = let}
\]

\[
\text{val evaltrm = eval_term lthy \circ Syntax.read_term lthy}
\]

\[
| \text{val trfresh = \"Transactions.transaction_fresh (\" transactionterm \wedge \")\"}
\]

\[
| \text{val trreceive = \"Transactions.transaction_receive (\" transactionterm \wedge \")\"}
\]

\[
| \text{val trchecks = \"Transactions.transaction_checks (\" transactionterm \wedge \")\"}
\]

\[
| \text{val trupdates = \"Transactions.transaction_updates (\" transactionterm \wedge \")\"}
\]

\[
| \text{val trsend = \"Transactions.transaction_send (\" transactionterm \wedge \")\"}
\]

\[
\text{fun print_fresh xs =}
\]

\[
| \text{if xs = [] then []}
\]

\[
| \text{else [\" new " \text{String.concatWith ", " (map (var_printer o pairterm_to_pair lthy) xs)]}
\]

\[
\text{fun print_tr_s s = transaction_labeled_action_to_string var_printer protocol s lthy}
\]

\[
| \text{val print_tr =}
\]

\[
\text{String.concatWith \"\" (}
\]

\[
| \text{(map print_tr_s \circ listterm_to_list lthy \circ evaltrm) trreceive0}
\]

\[
| \text{(map print_tr_s \circ listterm_to_list lthy \circ evaltrm) trchecks0}
\]

\[
| \text{(print_fresh \circ listterm_to_list lthy \circ evaltrm) trfresh0}
\]

\[
| \text{(map print_tr_s \circ listterm_to_list lthy \circ evaltrm) trupdates0}
\]

\[
| \text{(map print_tr_s \circ listterm_to_list lthy \circ evaltrm) trsend0)
\]

\[
\text{in}
\]

\[
\text{print_tr}
\]

\[
\text{end}
\]

\[
\text{fun transaction_list_to_string var_printer protocol transactionlistterm lthy = let}
\]

\[
\text{fun err msg t = error (msg \wedge \text{" "} Syntax.string_of_term lthy t)}
\]

\[
| \text{val evaltrm = eval_term lthy \circ Syntax.read_term lthy}
\]

\[
\text{fun print_fresh i xs =}
\]

\[
| \text{if xs = [] then []}
\]

\[
| \text{else [\" new " \text{String.concatWith ", " (map (var_printer i o pairterm_to_pair lthy) xs)]}
\]

\[
368
\]
fun print_tr i s = 
transaction_labeled_action_to_string (var_printer i) protocol s lthy

fun print_tr i ((Const ("Product_Type.Pair",_)$ trfresh)$
(Const ("Product_Type.Pair",_)$ trreceive)$
(Const ("Product_Type.Pair",_)$ trchecks)$
(Const ("Product_Type.Pair",_)$ trupdates)$
(Const ("Product_Type.Pair",_)$ trsend))
= String.concatWith "\n" ( 
(map (print_tr' i) o listterm_to_list lthy) trreceive@
(map (print_tr' i) o listterm_to_list lthy) trchecks@
(print_fresh i o listterm_to_list lthy) trfresh@
(map (print_tr' i) o listterm_to_list lthy) trupdates@
(map (print_tr' i) o listterm_to_list lthy) trsend)

| print_tr _ t = err "Unexpected term" t

fun print_trs trs = String.concatWith "\n\n" (map_index (fn (i,x) => print_tr i x) trs)

fun trfresh s = "Transactions.transaction_fresh ("^ s ^")"
fun trreceive s = "Transactions.transaction_receive ("^ s ^")"
fun trchecks s = "Transactions.transaction_checks ("^ s ^")"
fun trupdates s = "Transactions.transaction_updates ("^ s ^")"
fun trsend s = "Transactions.transaction_send ("^ s ^")"

val f = "let f = \X. ("^ trfresh "X" ^", ".^ trreceive "X" ^", ", ^ trchecks "X" ^", ", ^ trupdates "X" ^", ", ^ trsend "X" ^") ^"
"in map f ("^ transactionlistterm ^")"

val transactionlistterm = listterm_to_list lthy (evaltrm f)
in
print_trs transactionlistterm
end

val _ = Outer_Syntax.local_theory' @{command_keyword "print_transaction_strand"}
"print protocol transaction as transaction strand"
(Parse.name -- Parse.name >>
(fn (protocol, transaction) => fn print => fn lthy =>
let fun print_tr ((protocol,transaction), _, lthy) =
let
val _ = assert_defined lthy transaction
fun f a = protterm_to_string_no_eval (fn _ => error "Unexpected variable")
protocol a lthy
fun g (a,b) =
if f a = "Value" orelse f a = "value"
then "X" ^ Syntax.string_of_term lthy b
else "Y[" ^ f a "^ , " ^ Syntax.string_of_term lthy b "^ ]"^ 
val _ = Output.information (transaction_to_string g protocol transaction lthy)
in
lthy
end
in
trac_time.ap_lthy lthy
("print_transaction_strand ("^protocol^")")
print_tr
((protocol,transaction), print, lthy)
end );

val _ = Outer_Syntax.local_theory' @{command_keyword "print_transaction_strand_list"}
"print protocol transaction list as transaction strand list"
(Parse.name -- Parse.name =>
(fn (protocol, transaction_list) => fn print => fn lthy =>
let fun print_tr ((protocol,transaction_list), _, lthy) =
let

val _ = assert_defined lthy transaction_list
fun f a = protterm_to_string_no_eval (fn n => error "Unexpected variable")
    protocol a lthy
fun g i (a,b) = 
  if f a = "Value" orelse f a = "value"
  then "X" ^ b ^ "_" ^ Int.toString i
  else "Y[" ^ f a ^ ", " ^ Syntax.string_of_term lthy b ^ "_" ^ Int.toString i ^ "]"
val _ = Output.information (transaction_list_to_string g protocol transaction_list lthy)
in
lthy
end
in
trac_time.ap_lthy lthy
("print_transaction_strand_list ("^protocol^")")
print_tr
((protocol,transaction_list), print, lthy)
end );
val _ = Outer_Syntax.local_theory' @{command_keyword "print_attack_trace"}
"print attack trace"
(Parse.name -- Parse.name -- Parse.name >>
(fn ((protocol, protocol_def), attack_trace) => fn print => fn lthy =>
let fun print_tr ((protocol,protocol_def,attack_trace), _, lthy) = 
let
val evaltrm = eval_term lthy o Syntax.read_term lthy
val _ = assert_defined lthy protocol_def
val _ = assert_defined lthy attack_trace
fun f i b = "X" ^ b ^ "_" ^ Int.toString i
fun g i (_,b) = f i (Syntax.string_of_term lthy b)
val t1 = "map (\(i,_.). " ^ protocol_def ^ " ! i) " ^ attack_trace
val t2 = "map (map (\((_,i),xs). (i,xs))) (map snd " ^ attack_trace ^ ")"
val s = t2 |> evaltrm
  |> listterm_to_list lthy
  |> map (listterm_to_list lthy)
  |> map (map (pairterm_to_pair lthy))
  |> map (map (fn (a,b) => (Syntax.string_of_term lthy a,
                               abstractionexpr_to_list protocol b))))
  |> map_index (fn (i,ts) =>
                map (fn (a,xs) => f i a ^ ": {" ^ String.concatWith ", " ^ xs ^ "}")) ts)
  |> List.concat
val u = transaction_list_to_string g protocol t1 lthy
val _ = Output.information ( 
  (if s <> []
      then "Abstractions:
             String.concatWith "\n           s ^ "\n         else "") ^
     ("Attack trace:
      String.concatWith "\n         u))
in
lthy
end
in
trac_time.ap_lthy lthy
("print_attack_trace ("^protocol^","^protocol_def^","^attack_trace^")")
print_tr
((protocol,protocol_def,attack_trace), print, lthy)
end );
val _ = Outer_Syntax.local_theory' @{command_keyword "print_fixpoint"}
"print protocol fixpoint"
(Parse.name -- Parse.name >>
(fn (protocol, fixpoint) => fn print => fn lthy =>
let fun print_fixpoint ((protocol,fixpoint), _, lthy) = 
let
val _ = assert_defined lthy fixpoint

370
4.6 Support for the Trac Format

```ml
val _ = Output.information (fixpoint_to_string protocol fixpoint lthy)
in
lthy
end
in
trac_time.ap_lthy lthy ("print_fixpoint ("\"protocol\")") print_fixpoint((protocol,fixpoint), print, lthy)
end ));

val _ = Outer_Syntax.local_theory' @{command_keyword "save_fixpoint"}
"Write fixpoint to file."
((Parse.name -- Parse.name -- Parse.name >> (fn ((protocol_name, fixpoint_filename), fixpoint_name) => fn _ => fn lthy =>
let
fun save_fixpoint ((protocol_name, fixpoint_name), fixpoint_filename, lthy) =
let
val _ = assert_defined lthy fixpoint_name
fun write f s =
if File.exists f
then error ("Error: Cannot write to file: File already exists")
else File.write f s
val filename =
Path.explode (trac.mk_abs_filename (Proof_Context.theory_of lthy) fixpoint_filename)
val _ = Output.information ("Evaluation fixed-point term " ^ fixpoint_name)
val fp_str = fixpoint_to_string protocol_name fixpoint_name lthy
val _ = Output.information ("Writing fixed point to file " ^ Path.print filename)
val _ = write filename fp_str
in
lthy
end
in
trac_time.ap_lthy lthy ("save_fixpoint") save_fixpoint ((protocol_name, fixpoint_name), fixpoint_filename, lthy)
end));

val _ = Outer_Syntax.local_theory' @{command_keyword "load_fixpoint"}
"Import fixpoint from file."
((Parse.name -- Parse.name -- Parse.name >> (fn ((protocol_name, fixpoint_filename), fixpoint_name) => fn print => fn lthy =>
let
fun load_fixpoint ((protocol_name, fixpoint_filename), fixpoint_name, lthy) =
let
val _ = assert_not_defined lthy fixpoint_name
val filename =
Path.explode (trac.mk_abs_filename (Proof_Context.theory_of lthy) fixpoint_filename)
val _ = Output.information ("Reading fixed point from file " ^ Path.print filename)
fun read f =
if File.exists f
then File.read f
else error ("Error: Cannot read file: File does not exist")
val fp_str = TracFpParser.parse_str (read filename)
val trac = trac_definitorial_package.lookup_trac protocol_name lthy
val cert_fp = TracProtocolCert.certify_fixpoint trac fp_str
val cert_trac = TracProtocolCert.certifyProtocol trac
val fp_trm = trac_definitorial_package.fp_triple_to_hol cert_fp cert_trac lthy
in
#2 (ml_isar_wrapper.define_constant_definition' (fixpoint_name, fp_trm) print lthy)
end
in
```
4 Trac Support and Automation

    trac_time.ap_lthy lthy ("load_fixpoint") load_fixpoint ((protocol_name, fixpoint_filename), fixpoint_name, lthy)
    end));

val _ = Outer_Syntax.local_theory' @{command_keyword "compute_fixpoint"}
  evaluate and define protocol fixpoint
  (Parse.name -- Parse.name -- Scan.option Parse.name >>
   (fn ((protocol, fixpoint), opt_trace) => fn print => fn lthy =>
    let fun compute_fixpoint (((protocol,fixpoint),opt_trace), print, lthy) =
    let
      val _ = assert_defined lthy protocol
      val _ = assert_not_defined lthy fixpoint
      val _ = Option.app (assert_not_defined lthy) opt_trace
      val _ = Output.information ("Computing a fixed point for protocol " ^ protocol)
      val fp = (fixpoint, Syntax.read_term lthy ("compute_fixpoint_fun " ^ protocol))
      val opt_tr = Option.map (* TODO: don't compute the fixpoint twice *)
        (fn trace =>
         (trace, Syntax.read_term lthy
          ("(compute_reduced_attack_trace " ^ protocol ^
            o snd o compute_fixpoint_with_trace) " ^ protocol)))
    in
      ((snd o eval_define_declare fp print) lthy |
       (fn lthy => case opt_tr of
        SOME tr => (snd o eval_define_declare tr print) lthy
                   | NONE => lthy))
    handle ERROR msg =>
      let
        val _ = warning ("Failed to compute the set with eval. Retrying with NBE.\n" ^
          "Original error message: \n" ^ msg)
      in
        ((snd o eval_define_declare_nbe fp print) lthy |
         (fn lthy => case opt_tr of
          SOME tr => (snd o eval_define_declare_nbe tr print) lthy
                     | NONE => lthy))
      end
    end
    in
    trac_time.ap_lthy lthy ("compute_fixpoint (" ^ protocol ^ ")") compute_fixpoint
    (((protocol,fixpoint),opt_trace), print, lthy)
    end });

val _ = Outer_Syntax.local_theory' @{command_keyword "compute_SMP"}
  evaluate and define a finite representation of the sub-message patterns of a protocol
  ((Scan.optional (keyword [ ! |-- Parse.name --| keyword [ ] |]) "no_optimizations") --
   Parse.name -- Parse.name >> (fn ((opt,proto), smp) => fn print => fn lthy =>
    let fun compute_smp (((opt, proto), smp), print, lthy) =
    let
      val prot' = prot
      val rmd = "List.remdups"
      val f = "Stateful_Strands.trms_list
      val g =
        "(\AT. " ^ f ^ " T@map (pair' prot_fun.Pair) (Stateful_Strands.setops_list
          AT T))" (" ^ rmd ^ ") (List.concat (List.map (" ^ trms ^
          o Labeled_Strands.unlabel o transaction_strand) " ^ prot' ^ ")))"
      val opt1 = "remove_superfluous_terms Γ" (" ^ generalize_terms Γ \is_Var"
      val opt2 = "generalize_terms Γ \is_Var"
      val gemp_opt =
        "generalize_terms Γ (\At. is_Var t ∧ t ≠ TAtom AttackType ∧ " ^
          "t ≠ TAtomsetType ∧ t ≠ TAtom OccursSecType ∧ ¬is_Atom (the_Var t))"
4.6 Support for the Trac Format

val smp_fun = "SMP0 Ana Γ"
fun smp_fun' opts =
  "(λT. let T' = (" ^ rmd ^ " o " ^ opts ^ " o " ^ smp_fun ^"
  "(Messages.fv.set (set (T@T'))))) T')"
val cmd =
  if opt = "no_optimizations" then smp_fun ^ " ^ s f
else if opt = "optimized"
then smp_fun' (opt1 ^ " o " ^ opt2) ^ " ^ s f
else if opt = "GSMP"
then smp_fun' (opt1 ^ " o " ^ gsmp_opt) ^ " ^ s g
else if opt = "composition"
then smp_fun' " ^ s g
else if opt = "composition_optimized"
then smp_fun' (opt1 ^ " o " ^ opt2) ^ " ^ s g
else error ("Error: Invalid option: " ^ opt ^ "\nValid options:\n1. no_optimizations: Computes the finite SMP representation set " ^
  "without any optimizations (this is the default setting).\n" ^
  "2. optimized: Applies optimizations to reduce the size of the computed " ^
  "set, but this might not be sound.\n" ^
  "3. GSMP: Computes a set suitable for use in checking GSMP disjointness.\n" ^
  "4. composition: Computes a set suitable for checking type-flaw resistance " ^
  "of composed protocols.\n" ^
  "5. composition_optimized: An optimized variant of the previous setting."
"
val _ = assert_defined lthy prot
val _ = assert_not_defined lthy smp
val _ = Output.information ("Computing a finite SMP representation set for protocol " ^
  prot)
in (snd o eval_define_declare (smp, Syntax.read_term lthy cmd) print) lthy
handle ERROR msg =>
  let val _ = warning ("Failed to compute the set with eval. Retrying with NBE.\n" ^
    "Original error message:\n" ^
    msg)
in (snd o eval_define_declare_nbe (smp, Syntax.read_term lthy cmd) print) lthy
end
in
trac_time.ap_lthy lthy ("compute_SMP (" ^ prot ^ "")") compute_smp (((opt, prot), smp), print,
  lthy)
end);
val _ = Outer_Syntax.local_theory' @{command_keyword "compute_shared_secrets"}
"evaluate and define a finite representation of shared secrets as the intersection of GSMP
sets"
(Scan.repeat1 Parse.name >> (fn params => fn print => fn lthy =>
  let fun compute_shared_secrets (params, print, lthy) =
    let
      val _ = if length params < 3 then error "Not enough arguments" else ()
    val (gsmps, sec) = split_last params
    val xs = "xs"
    val cmd =
      "let " ^ xs ^ " ^ = [" ^ String.concatWith ", " ^ gsmps ^ "] in " ^
      "(* remove_superfluous_terms Γ o generalize_terms Γ ((=) (TAtom SetType)) o " ^ *
      "remove_superfluous_terms Γ o " ^
      "(*
      "concat o map " ^
      "(λp. filter " ^
      "(λt. list_ex (λs. Γ t = Γ s ∧ mgu t s ≠ None) (" ^ xs ^ " ! snd p))" ^
      "(" ^ xs ^ " ! fst p))" ^
      ")
Trac Support and Automation

("filter (\p. fst p \neq snd p) ((\p. List.product p p) [0..<length "^xs"]))")

val _ = map (assert_defined lthy) gsmps
val _ = assert_not_defined lthy sec
val _ = Output.information ("Computing a finite representation of the shared secrets for the protocols with GSMP sets " ^ String.concatWith ", " gsmps)
in (snd o eval_define Declare (sec, Syntax.read_term lthy cmd) print) lthy
end in
trac_time.ap_lthy lthy
(\"compute_shared_secrets ([\"-String,concatWith ", " params\"]\")\")
compute_shared_secrets (params, print, lthy)
end);

val _ = Outer_Syntax_LOCAL_theory' @{command_keyword "setup_protocol_checks"}
"setup protocol checks"
(Parse.name -- Scan.repeat Parse.name >>
(fn params => fn print => fn lthy =>
let fun setup_protocol_checks ((protocol_model, protocol_names), print, lthy) =
let
  fun f s = protocol_model ^ ".n" ^ s
  val a1 = "protocol_check_intro_lemmata"
  val a2 = "coverage_check_unfold_lemmata"
  val a3 = "coverage_check_unfold_protocol_lemma"
  val a4 = "protocol_checks'"
in (declare_protocol_checks print #>
    declare_thm_attr a1 (f "protocol_covered_by_fixpoint_intros") print #>
    declare_def_attr a2 (f "protocol_covered_by_fixpoint") print #>
    fold (fn s => declare_def_attr a3 s print) protocol_names #>
    declare_def_attr a4 (f "wellformed_fixpoint") print #>
    declare_def_attr a4 (f "wellformed_protocol") print #>
    declare_def_attr a4 (f "wellformed_protocol'") print #>
    declare_def_attr a4 (f "composable_protocols") print) lthy
end in
trac_time.ap_lthy lthy
(\"setup_protocol_checks (([^fst params\"],[^"String,concatWith ", " (snd params)\"]\")\")\")
setup_protocol_checks (params, print, lthy)
end});
5 Examples

5.1 The Keyserver Protocol

theory Keyserver
  imports "../PSPSP"
begin

declare [[code_timing,pspsp_timing]]

trac<
Protocol: keyserver

Enumerations:
honest = \{a,b,c\}
sserver = \{s\}
agents = honest ++ server

Sets:
ring/1 valid/2 revoked/2

Functions:
Public sign/2 crypt/2 pair/2
Private inv/1

Analysis:
sign(X,Y) -> Y
crypt(X,Y) ? inv(X) -> Y
pair(X,Y) -> X,Y

Transactions:
# Out-of-band registration
outOfBand(A:honest,S:server)
  new NPK
  insert NPK ring(A)
  send NPK.

# User update key
keyUpdateUser(A:honest,PK:value)
  PK in ring(A)
  new NPK
  delete PK ring(A)
  insert NPK ring(A)
  send sign(inv(PK),pair(A,NPK)).

# Server update key
keyUpdateServer(A:honest,S:server,PK:value,NPK:value)
  receive sign(inv(PK),pair(A,NPK))
  PK in valid(A,S)
  NPK notin valid(_)
  NPK notin revoked(_)
  delete PK valid(A,S)
  insert PK revoked(A,S)
  insert NPK valid(A,S)
  send inv(PK).
5 Examples

# Attack definition
authAttack(A:honest, S:server, PK:value)
  receive inv(PK)
  PK in valid(A,S)
  attack.

val(intruderValues)
val(ring(A)) where A:honest
sign(inv(val(0)), pair(A, val(ring(A)))) where A:honest
inv(val(revoked(A,S))) where A:honest S:server
pair(A, val(ring(A))) where A:honest

occurs(val(intruderValues))
occurs(val(ring(A))) where A:honest

\text{timplies}(\text{val}(\text{ring}(A)), \text{val}(\text{ring}(A), \text{valid}(A,S))) \text{ where } A: \text{honest } S: \text{server}
\text{timplies}(\text{val}(\text{ring}(A)), \text{val}(0)) \text{ where } A: \text{honest}
\text{timplies}(\text{val}(\text{ring}(A), \text{valid}(A,S)), \text{val}(\text{valid}(A,S))) \text{ where } A: \text{honest } S: \text{server}
\text{timplies}(\text{val}(0), \text{val}(\text{valid}(A,S))) \text{ where } A: \text{honest } S: \text{server}
\text{timplies}(\text{val}(\text{valid}(A,S)), \text{val}(\text{revoked}(A,S))) \text{ where } A: \text{honest } S: \text{server}

5.1.1 Proof of security

protocol_model_setup spm: keyserver
compute_SMP [optimized] keyserver_protocol keyserver_SMP
manual_protocol_security_proof ssp: keyserver
  for keyserver_protocol keyserver_fixpoint keyserver_SMP
  apply check_protocol_intro
  subgoal by code_simp
  subgoal by code_simp
  subgoal by code_simp
  subgoal by code_simp
  done
end

5.2 A Variant of the Keyserver Protocol

theory Keyserver2
  imports "../PSPSP"
begin
declare [[code_timing]]

trac<
Protocol: keyserver2

Enumerations:
honest = \{a,b,c\}
dishonest = \{i\}
agent = honest ++ dishonest

Sets:
ring'/1 seen/1 pubkeys/0 valid/1

Functions:
Public h/1 sign/2 crypt/2 scrypt/2 pair/2 update/3
Private inv/1 pw/1
5.2 Analysis:
\begin{align*}
sign(X, Y) & \rightarrow Y \\
crypt(X, Y) \ ? \ inv(X) & \rightarrow Y \\
scrypt(X, Y) \ ? & X \rightarrow Y \\
pair(X, Y) & \rightarrow X, Y \\
update(X, Y, Z) & \rightarrow X, Y, Z
\end{align*}

Transactions:
passwordGenD(A: dishonest)
\begin{itemize}
\item send pw(A).
\end{itemize}

pubkeysGen()
\begin{itemize}
\item new PK
\item insert PK pubkeys
\item send PK.
\end{itemize}

updateKeyPu(A: honest, PK: value)
\begin{itemize}
\item PK in pubkeys
\item new NPK
\item insert NPK ring'(A)
\item send NPK
\item send crypt(PK, update(A, NPK, pw(A))).
\end{itemize}

updateKeyServerPw(A: agent, PK: value, NPK: value)
\begin{itemize}
\item receive crypt(PK, update(A, NPK, pw(A)))
\item PK in pubkeys
\item NPK notin pubkeys
\item NPK notin seen(_)
\item insert NPK valid(A)
\item insert NPK seen(A).
\end{itemize}

authAttack2(A: honest, PK: value)
\begin{itemize}
\item receive inv(PK)
\item PK in valid(A)
\item attack.
\end{itemize}

5.2.1 Proof of security

protocol_model_setup smp: keyserver2
compute_fixpoint keyserver2_protocol keyserver2_fixpoint
protocol_security_proof ssp: keyserver2

5.2.2 The generated theorems and definitions

thm ssp.protocol_secure

thm keyserver2_enum_consts.nchotomy
thm keyserver2_sets.nchotomy
thm keyserver2_fun.nchotomy
thm keyserver2_atom.nchotomy
thm keyserver2_arity.simps
thm keyserver2_public.simps
thm keyserver2_G.simps
thm keyserver2_Ana.simps
thm keyserver2_transaction_passwordGenD_def
thm keyserver2_transaction_pubkeysGen_def
thm keyserver2_transaction_updateKeyPu_def
thm keyserver2_transaction_updateKeyServerPw_def
thm keyserver2_transaction_authAttack2_def
thm keyserver2_protocol_def
5 Examples

thm keyserver2_fixpoint_def
end

5.3 The Composition of the Two Keyserver Protocols

theory Keyserver_Composition
  imports "../PSPSP"
begin

declare [[pspsp_timing]]

trac:
Protocol: kscomp

Enumerations:
honest = {a,b,c}
dishonest = {i}
agent = honest ++ dishonest

Sets:
ring/1 valid/1 revoked/1 deleted/1
ring'/1 seen/1 pubkeys/0

Functions:
Public h/1 sign/2 crypt/2 scrypt/2 pair/2 update/3
Private inv/1 pw/1

Analysis:
sign(X,Y) -> Y
crypt(X,Y) ? inv(X) -> Y
scrypt(X,Y) ? X -> Y
pair(X,Y) -> X,Y
update(X,Y,Z) -> X,Y,Z

### The signature-based keyserver protocol
Transactions of pi:
intruderGen()
  new PK
  * send PK, inv(PK).

outOfBand(A:honest)
  new PK
  insert PK ring(A)
  * insert PK valid(A)
  * send PK.

outOfBandD(A:dishonest)
  new PK
  * insert PK valid(A)
  * send PK, inv(PK).

updateKey(A:honest,PK:value)
  PK in ring(A)
  new NPK
  delete PK ring(A)
  insert PK deleted(A)
  insert NPK ring(A)
  send sign(inv(PK),pair(A,NPK)).

updateKeyServer(A:agent,PK:value,NPK:value)
5.3 The Composition of the Two Keyserver Protocols

receive sign(inv(PK),pair(A,NPK))
* PK in valid(A)
* NPK notin valid(_,)
  NPK notin revoked(_)
* delete PK valid(A)
  insert PK revoked(A)
* insert NPK valid(A)
* send inv(PK).

authAttack(A:honest,PK:value)
  receive inv(PK)
  * PK in valid(A)
  attack.

### The password-based keyserver protocol
Transactions of p2:
intruderGen'()
  new PK
  * send PK, inv(PK).

passwordGenD(A:dishonest)
  send pw(A).

pubkeysGen()
  new PK
  insert PK pubkeys
  * send PK.

updateKeyPw(A:honest,PK:value)
  PK in pubkeys
  new NPK
* NOTE: The ring' sets are not used elsewhere, but we have to avoid that the fresh keys generated
  by this rule are abstracted to the empty abstraction, and so we insert them into a ring'
  set. Otherwise the two protocols would have too many abstractions in common (in particular,
  the empty abstraction) which leads to false attacks in the composed protocol (probably
  because the term implication graphs of the two protocols then become 'linked' through the
  empty abstraction)
  insert NPK ring'(A)
  * send NPK
  send crypt(PK,update(A,NPK,pw(A))).

updateKeyServerPw(A:agent,PK:value,NPK:value)
  receive crypt(PK,update(A,NPK,pw(A)))
  PK in pubkeys
  NPK notin pubkeys
  NPK notin seen(_)
  * insert NPK valid(A)
  insert NPK seen(A).

authAttack2(A:honest,PK:value)
  receive inv(PK)
  * PK in valid(A)
  attack.

5.3.1 Proof: The composition of the two keyserver protocols is secure

protocol_model_setup spm: kscomp
setup_protocol_checks spm kscomp_protocol kscomp_protocol_p1 kscomp_protocol_p2
compute_fixpoint kscomp_protocol kscomp_fixpoint
manual_protocol_security_proof ssp: kscomp
  for kscomp_protocol kscomp_fixpoint
5.3.2 The generated theorems and definitions

thm ssp.protocol_secure

thm kscomp_enum_consts.nchotomy
thm kscomp_sets.nchotomy
thm kscomp_fun.nchotomy
thm kscomp_atom.nchotomy
thm kscomp_arity.simps
thm kscomp_public.simps
thm kscomp_Γ.simps
thm kscomp_Ana.simps

thm kscomp_transaction_p1_outOfBand_def
thm kscomp_transaction_p1_outOfBandD_def
thm kscomp_transaction_p1_updateKey_def
thm kscomp_transaction_p1_updateKeyServer_def
thm kscomp_transaction_p1_authAttack_def
thm kscomp_transaction_p2_passwordGenD_def
thm kscomp_transaction_p2_pubkeysGen_def
thm kscomp_transaction_p2_updateKeyPu_def
thm kscomp_transaction_p2_updateKeyServerPu_def
thm kscomp_transaction_p2_authAttack2_def
thm kscomp_protocol_def

thm kscomp_fixpoint_def

end

5.4 The PKCS Model, Scenario 3

theory PKCS_Model03
  imports "././PSPSP"
begin
declare [[code_timing]]

trac<
Protocol: ATTACK_UNSET

Enumerations:
token = {token1}

Sets:
extract/1 wrap/1 decrypt/1 sensitive/1

Functions:
Public senc/2 h/1
Private inv/1

Analysis:
senc(M,K2) ? K2 -> M #This analysis rule corresponds to the decrypt2 rule in the AIF-omega specification.

#M was type untyped

Transactions:
iik1()
  new K1
  insert K1 sensitive(token1)
  insert K1 extract(token1)
  send h(K1).

iik2()
  new K2
  insert K2 wrap(token1)
  send h(K2).

# ===============wrap===============
wrap(K1:value,K2:value)
  receive h(K1)
  receive h(K2)
  K1 in extract(token1)
  K2 in wrap(token1)
  send senc(K1,K2).

# ===============set wrap==============
setwrap(K2:value)
  receive h(K2)
  K2 notin decrypt(token1)
  insert K2 wrap(token1).

# ===============set decrypt==============
setdecrypt(K2:value)
  receive h(K2)
  K2 notin wrap(token1)
  insert K2 decrypt(token1).

# ===============decrypt==============
decrypt1(K2:value,M:value) #M was untyped in the AIF-omega specification.
  receive h(K2)
  receive senc(M,K2)
  K2 in decrypt(token1)
  send M.

# ===============attacks===============
attack1(K1:value)
  receive K1
  K1 in sensitive(token1)
5 Examples

attack.

5.4.1 Protocol model setup

protocol_model_setup apm: ATTACK_UNSET

5.4.2 Fixpoint computation

compute_fixpoint ATTACK_UNSET_protocol ATTACK_UNSET_fixpoint attack_trace

The fixpoint contains an attack signal

lemma "attack(ln 0) ∈ set (fst ATTACK_UNSET_fixpoint)"
by code_simp

The attack trace can be inspected as follows

print_attack_trace ATTACK_UNSET ATTACK_UNSET_protocol attack_trace

5.4.3 The generated theorems and definitions

thm ATTACK_UNSET_enum_consts.nchotomy
thm ATTACK_UNSET_sets.nchotomy
thm ATTACK_UNSET_fun.nchotomy
thm ATTACK_UNSET_atom.nchotomy
thm ATTACK_UNSET_arity.simps
thm ATTACK_UNSET_public.simps
thm ATTACK_UNSET_\Gamma .simp
thm ATTACK_UNSET_Ana.simps
thm ATTACK_UNSET_transaction_iik1_def
thm ATTACK_UNSET_transaction_iik2_def
thm ATTACK_UNSET_transaction_wrap_def
thm ATTACK_UNSET_transaction_setwrap_def
thm ATTACK_UNSET_transaction_decrypt1_def
thm ATTACK_UNSET_transaction_attack1_def
thm ATTACK_UNSET_protocol_def
thm ATTACK_UNSET_fixpoint_def

end

5.5 The PKCS Protocol, Scenario 7

theory PKCS_Model07
  imports "./../PSPSP"
begin

declare [[code_timing]]

trace
Protocol: RE_IMPORT_ATT

Enumerations:
token = {token1}

Sets:
extract/1 wrap/1 unwrap/1 decrypt/1 sensitive/1

Functions:
Public senc/2 h/2 bind/2
Private inv/1

Analysis:
senc(M1,K2) ? K2 -> M1  #This analysis rule corresponds to the decrypt2 rule in the AIF-omega specification.

#M1 was type untyped

Transactions:
iik1()
new K1
new N1
insert N1 sensitive(token1)
insert N1 extract(token1)
insert K1 sensitive(token1)
send h(N1,K1).

iik2()
new K2
new N2
insert N2 wrap(token1)
insert N2 extract(token1)
send h(N2,K2).

# =====set wrap=====
setwrap(N2:value,K2:value)
receive h(N2,K2)
N2 notin sensitive(token1)
N2 notin decrypt(token1)
insert N2 wrap(token1).

# =====set unwrap===
setunwrap(N2:value,K2:value)
receive h(N2,K2)
N2 notin sensitive(token1)
insert N2 unwrap(token1).

# =====unwrap, generate new handler======
#-----------the sensitive attr copy-------------
unwrapsensitive(M2:value, K2:value, N1:value, N2:value) #M2 was untyped in the AIF-omega specification.
receive senc(M2,K2)
receive bind(N1,M2)
receive h(N2,K2)
N1 in sensitive(token1)
N2 in unwrap(token1)
new Nnew
insert Nnew sensitive(token1)
send h(Nnew,M2).

#-----------the wrap attr copy-------------
wrapattr(M2:value, K2:value, N1:value, N2:value) #M2 was untyped in the AIF-omega specification.
receive senc(M2,K2)
receive bind(N1,M2)
receive h(N2,K2)
N1 in wrap(token1)
N2 in unwrap(token1)
new Nnew
insert Nnew wrap(token1)
send h(Nnew,M2).

#-----------the decrypt attr copy-------------
decryptattr(M2:value,K2:value,N1:value,N2:value) #M2 was untyped in the AIF-omega specification.
5 Examples

```plaintext
cipher(M2,K2)
cipher(bind(N1,M2)
cipher h(N2,K2)
N1 in decrypt(token1)
N2 in unwrap(token1)
new Nnew
insert Nnew decrypt(token1)
send h(Nnew,M2).
cipher(M2:value,K2:value,N1:value,N2:value) #M2 was untyped in the AIF-omega specification.
cipher(M2,K2)
cipher bind(N1,M2)
cipher h(N2,K2)
N1 notin sensitive(token1)
N1 notin wrap(token1)
N1 notin decrypt(token1)
N2 in unwrap(token1)
new Nnew
send h(Nnew,M2).

# ======================wrap================

# wrap(N1:value,K1:value,N2:value,K2:value)
cipher h(N1,K1)
cipher h(N2,K2)
N1 in extract(token1)
N2 in wrap(token1)
send senc(K1,K2)
send bind(N1,K1).

# ======set decrypt===

# setdecrypt(Nnew:value, K2:value)
cipher h(Nnew,K2)
Nnew notin wrap(token1)
insert Nnew decrypt(token1).

cipher(Nnew:value, K2:value,M1:value) #M1 was untyped in the AIF-omega specification.
cipher h(Nnew,K2)
cipher senc(M1,K2)
Nnew in decrypt(token1)
delete Nnew decrypt(token1)
send M1.

# ======================attacks================

cipher(K1:value)
cipher K1
K1 in sensitive(token1)
attack.
```
5.6 The PKCS Protocol, Scenario 9

theory PKCS_Model09
  imports "../../PSPSP"
begin

declare [[code_timing]]

trac:<
Protocol: LOSS_KEY_ATT

Enumerations:
  token = {token1}

Sets:
  extract/1 wrap/1 unwrap/1 decrypt/1 sensitive/1

Functions:
  Public senc/2 h/2 bind/3
  Private inv/1

Analysis:
  senc(M1,K2) ? K2 -> M1  #This analysis rule corresponds to the decrypt2 rule in the AIF-omega specification.
    #M1 was type untyped

Transactions:
  intruderValueGen()

  new K
5 Examples

send K.

iik1()
  new K1
  new N1
  insert N1 sensitive(token1)
  insert N1 extract(token1)
  insert K1 sensitive(token1)
  send h(N1,K1).

iik2()
  new K2
  new N2
  insert N2 wrap(token1)
  insert N2 extract(token1)
  send h(N2,K2).

iik3()
  new K3
  new N3
  insert N3 extract(token1)
  insert N3 decrypt(token1)
  insert K3 decrypt(token1)
  send h(N3,K3)
  send K3.

# =====set wrap=====
setwrap(N2:value,K2:value) where N2 != K2
  receive h(N2,K2)
  N2 notin sensitive(token1)
  N2 notin decrypt(token1)
  insert N2 wrap(token1).

# =====set unwrap===
setunwrap(N2:value,K2:value) where N2 != K2
  receive h(N2,K2)
  N2 notin sensitive(token1)
  insert N2 unwrap(token1).

# =====unwrap, generate new handler======
#-----------add the wrap attr copy-------------
unwrapWrap(M2:value,K2:value,N1:value,N2:value) where M2 != K2, M2 != N1, M2 != N2, K2 != N1, K2 != N2, N1 != N2 #M2 was untyped in the AIF-omega specification.
  receive senc(M2,K2)
  receive bind(N1,M2,K2)
  receive h(N2,K2)
  N1 in wrap(token1)
  N2 in unwrap(token1)
  new Nnew
  insert Nnew wrap(token1)
  send h(Nnew,M2).

#-----------add the sensitive attr copy-------------
unwrapSens(M2:value,K2:value,N1:value,N2:value) where M2 != K2, M2 != N1, M2 != N2, K2 != N1, K2 != N2, N1 != N2 #M2 was untyped in the AIF-omega specification.
  receive senc(M2,K2)
  receive bind(N1,M2,K2)
  receive h(N2,K2)
  N1 in sensitive(token1)
  N2 in unwrap(token1)
  new Nnew
  insert Nnew sensitive(token1)
  send h(Nnew,M2).
#----------add the decrypt attr copy----------
decrypt1Attr(M2:value, K2:value, N1:value, N2:value) where M2 != K2, M2 != N1, M2 != N2, K2 != N1, K2 != N2, N1 != N2 #M2 was untyped in the AIF-omega specification.
  receive senc(M2,K2)
  receive bind(N1,M2,K2)
  receive h(N2,K2)
  N1 in decrypt(token1)
  N2 in unwrap(token1)
  new Nnew
  insert Nnew decrypt(token1)
  send h(Nnew,M2).

decrypt2Attr(M2:value, K2:value, N1:value, N2:value) where M2 != K2, M2 != N1, M2 != N2, K2 != N1, K2 != N2, N1 != N2 #M2 was untyped in the AIF-omega specification.
  receive senc(M2,K2)
  receive bind(N1,M2,K2)
  receive h(N2,K2)
  N1 in decrypt(token1)
  N2 in unwrap(token1)
  new Nnew
  send h(Nnew,M2).

# ======================wrap================
wrap(N1:value, K1:value, N2:value, K2:value) where N1 != N2, N1 != K2, N1 != K1, N2 != K2, N2 != K1, K2 != K1
  receive h(N1,K1)
  receive h(N2,K2)
  N1 in extract(token1)
  N2 in wrap(token1)
  send senc(K1,K2)
  send bind(N1,K1,K2).

# ======================bind generation================
bind1(K3:value, N2:value, K2:value, K1:value) where K3 != N2, K3 != K2, K3 != K1, N2 != K2, N2 != K1, K2 != K1
  receive K3
  receive h(N2,K2)
  send bind(N2,K3,K3).

bind2(K3:value, N2:value, K2:value, K1:value) where K3 != N2, K3 != K2, K3 != K1, N2 != K2, N2 != K1, K2 != K1
  receive K3
  receive K1
  receive h(N2,K2)
  send bind(N2,K1,K3)
  send bind(N2,K3,K1).

# =====set decrypt===
setdecrypt(Nnew:value, K2:value) where Nnew != K2
  receive h(Nnew,K2)
  Nnew notin wrap(token1)
  insert Nnew decrypt(token1).

# ======================decrypt================
decrypt1(Nnew:value, K2:value, M1:value) where Nnew != K2, Nnew != M1, K2 != M1 #M1 was untyped in the AIF-omega specification.
  receive h(Nnew,K2)
  receive senc(M1,K2)
  Nnew in decrypt(token1)
  send M1.
5 Examples

# =============attacks============
attack1(K1:value)
  receive K1
  K1 in sensitive(token1)
  attack.
>
5.6.1 Protocol model setup
protocol_model_setup apm: LOSS_KEY_ATT

5.6.2 Fixpoint computation
compute_fixpoint LOSS_KEY_ATT_protocol LOSS_KEY_ATT_fixpoint attack_trace
  The fixpoint contains an attack signal

lemma "attack(ln 0) ∈ set (fst LOSS_KEY_ATT_fixpoint)"
by code simp

  The attack trace can be inspected as follows

print_attack_trace LOSS_KEY_ATT LOSS_KEY_ATT_protocol attack_trace

5.6.3 The generated theorems and definitions

thm LOSS_KEY_ATT_enum_consts.nchotomy
thm LOSS_KEY_ATT_sets.nchotomy
thm LOSS_KEY_ATT_fun.nchotomy
thm LOSS_KEY_ATT_atom.nchotomy
thm LOSS_KEY_ATT_arity.simps
thm LOSS_KEY_ATT_public.simps
thm LOSS_KEY_ATT_Γ '..simps
thm LOSS_KEY_ATT_Ana.simps

thm LOSS_KEY_ATT_transaction_iik1_def
thm LOSS_KEY_ATT_transaction_iik2_def
thm LOSS_KEY_ATT_transaction_iik3_def
thm LOSS_KEY_ATT_transaction_setwrap_def
thm LOSS_KEY_ATT_transaction_setunwrap_def
thm LOSS_KEY_ATT_transaction_unwrapWrap_def
thm LOSS_KEY_ATT_transaction_unwrapSens_def
thm LOSS_KEY_ATT_transaction_decrypt1Attr_def
thm LOSS_KEY_ATT_transaction_decrypt2Attr_def
thm LOSS_KEY_ATT_transaction_wrap_def
thm LOSS_KEY_ATT_transaction_bind1_def
thm LOSS_KEY_ATT_transaction_bind2_def
thm LOSS_KEY_ATT_transaction_setdecrypt_def
thm LOSS_KEY_ATT_transaction_decrypt1_def
thm LOSS_KEY_ATT_transaction_attack1_def

thm LOSS_KEY_ATT_protocol_def
thm LOSS_KEY_ATT_fixpoint_def

end
Bibliography


