

Stateful Protocol Composition and Typing

Andreas V. Hess* Sebastian Mödersheim* Achim D. Brucker†

February 23, 2021

*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`

Abstract

We provide in this AFP entry several relative soundness results for security protocols. In particular, we prove typing and compositionality results for stateful protocols (i.e., protocols with mutable state that may span several sessions), and that focuses on reachability properties. Such results are useful to simplify protocol verification by reducing it to a simpler problem: Typing results give conditions under which it is safe to verify a protocol in a typed model where only “well-typed” attacks can occur whereas compositionality results allow us to verify a composed protocol by only verifying the component protocols in isolation. The conditions on the protocols under which the results hold are furthermore syntactic in nature allowing for full automation. The foundation presented here is used in another entry to provide fully automated and formalized security proofs of stateful protocols.

Keywords: Security protocols, stateful protocols, relative soundness results, proof assistants, Isabelle/HOL, compositionality

Contents

1	Introduction	7
2	Preliminaries and Intruder Model	9
2.1	Miscellaneous Lemmata (Miscellaneous)	9
2.2	Protocol Messages as (First-Order) Terms (Messages)	12
2.3	Definitions and Properties Related to Substitutions and Unification (More_Unification)	17
2.4	Dolev-Yao Intruder Model (Intruder_Deduction)	39
3	The Typing Result for Non-Stateful Protocols	51
3.1	Strands and Symbolic Intruder Constraints (Strands_and_Constraints)	51
3.2	The Lazy Intruder (Lazy_Intruder)	69
3.3	The Typed Model (Typed_Model)	71
3.4	The Typing Result (Typing_Result)	85
4	The Typing Result for Stateful Protocols	97
4.1	Stateful Strands (Stateful_Strands)	97
4.2	Extending the Typing Result to Stateful Constraints (Stateful_Typing)	111
5	The Parallel Composition Result for Non-Stateful Protocols	119
5.1	Labeled Strands (Labeled_Strands)	119
5.2	Parallel Compositionality of Security Protocols (Parallel_Compositionality)	122
6	The Stateful Protocol Composition Result	131
6.1	Labeled Stateful Strands (Labeled_Stateful_Strands)	131
6.2	Stateful Protocol Compositionality (Stateful_Compositionality)	139
7	Examples	149
7.1	Proving Type-Flaw Resistance of the TLS Handshake Protocol (Example_TLS)	149
7.2	The Keyserver Example (Example_Keyserver)	153

1 Introduction

The rest of this document is automatically generated from the formalization in Isabelle/HOL, i.e., all content is checked by Isabelle. The formalization presented in this entry is described in more detail in several publications:

- The typing result (section 3 “Typing_Result”) for stateless protocols, the TLS formalization (section 7 “Example_TLS”), and the theories depending on those (see Figure 1.1) are described in [2] and [1, chapter 3].
- The typing result for stateful protocols (section 4 “Stateful_Typing”) and the keyserver example (section 7 “Example_Keyserver”) are described in [3] and [1, chapter 4].
- The results on parallel composition for stateless protocols (section 5 “Parallel_Compositionality”) and stateful protocols (?? “Stateful_Compositionality”) are described in [4] and [1, chapter 5].

Overall, the structure of this document follows the theory dependencies (see Figure 1.1): we start with introducing the technical preliminaries of our formalization (chapter 2). Next, we introduce the typing results in chapter 3 and chapter 4. We introduce our compositionality results in chapter 5 and chapter 6. Finally, we present two example protocols chapter 7.

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.

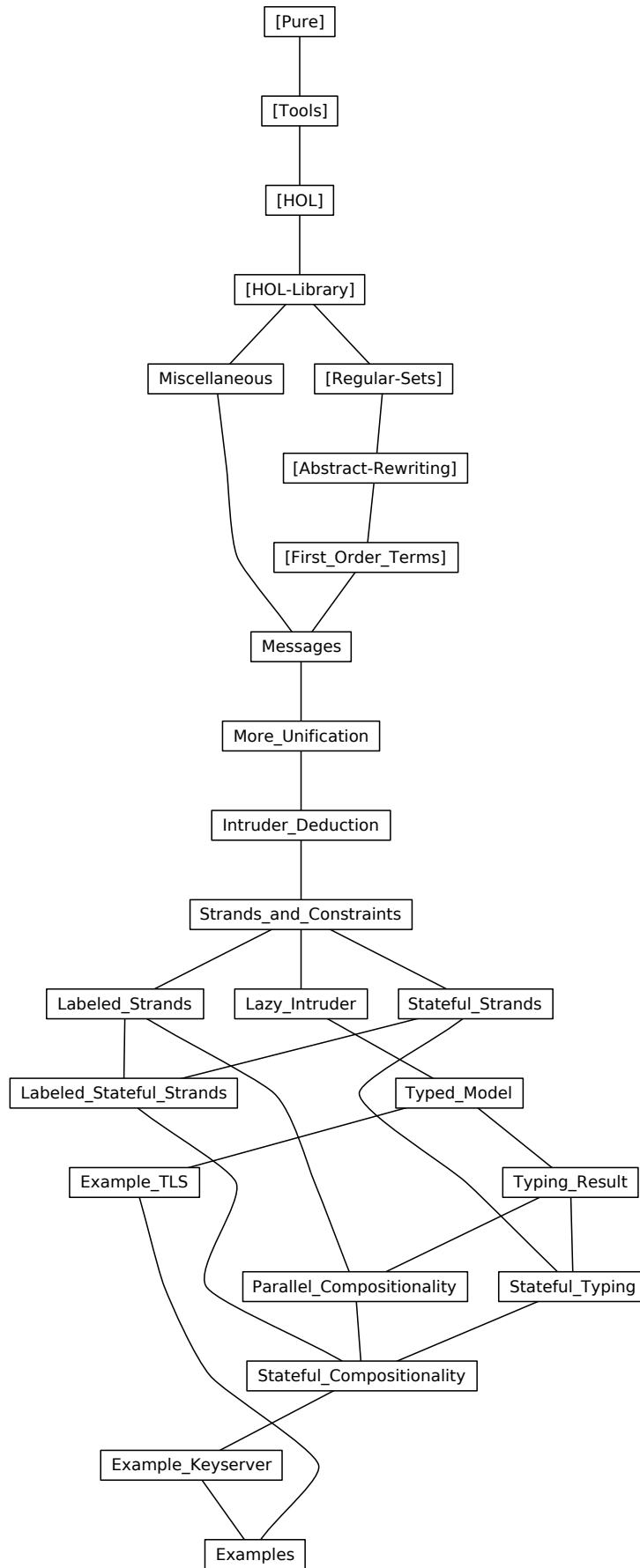


Figure 1.1: The Dependency Graph of the Isabelle Theories.

2 Preliminaries and Intruder Model

In this chapter, we introduce the formal preliminaries, including the intruder model and related lemmata.

2.1 Miscellaneous Lemmata (Miscellaneous)

```
theory Miscellaneous
imports Main "HOL-Library.Sublist" "HOL-Library.While_Combinator"
begin
```

2.1.1 List: zip, filter, map

```
lemma zip_arg_subterm_split:
  assumes "(x,y) ∈ set (zip xs ys)"
  obtains xs' xs'' ys' ys'' where "xs = xs'@x#xs''" "ys = ys'@y#ys''" "length xs' = length ys'"
⟨proof⟩
```

```
lemma zip_arg_index:
  assumes "(x,y) ∈ set (zip xs ys)"
  obtains i where "xs ! i = x" "ys ! i = y" "i < length xs" "i < length ys"
⟨proof⟩
```

```
lemma filter_nth: "i < length (filter P xs) ⟹ P (filter P xs ! i)"
⟨proof⟩
```

```
lemma list_all_filter_eq: "list_all P xs ⟹ filter P xs = xs"
⟨proof⟩
```

```
lemma list_all_filter_nil:
  assumes "list_all P xs"
  and "∧x. P x ⟹ ¬Q x"
  shows "filter Q xs = []"
⟨proof⟩
```

```
lemma list_all_concat: "list_all (list_all f) P ⟷ list_all f (concat P)"
⟨proof⟩
```

```
lemma map_upt_index_eq:
  assumes "j < length xs"
  shows "(map (λi. xs ! is i) [0..
```

```
lemma map_snd_list_insert_distrib:
  assumes "∀(i,p) ∈ insert x (set xs). ∀(i',p') ∈ insert x (set xs). p = p' ⟹ i = i'"
  shows "map snd (List.insert x xs) = List.insert (snd x) (map snd xs)"
⟨proof⟩
```

```
lemma map_append_inv: "map f xs = ys@zs ⟹ ∃vs ws. xs = vs@ws ∧ map f vs = ys ∧ map f ws = zs"
⟨proof⟩
```

2.1.2 List: subsequences

```
lemma subseqs_set_subset:
  assumes "ys ∈ set (subseqs xs)"
  shows "set ys ⊆ set xs"
⟨proof⟩
```

```
lemma subset_sublist_exists:
  "ys  $\subseteq$  set xs  $\implies$   $\exists$  zs. set zs = ys  $\wedge$  zs  $\in$  set (subseqs xs)"
<proof>
```

```
lemma map_subseqs: "map (map f) (subseqs xs) = subseqs (map f xs)"
<proof>
```

```
lemma subseqs_Cons:
  assumes "ys  $\in$  set (subseqs xs)"
  shows "ys  $\in$  set (subseqs (x#xs))"
<proof>
```

```
lemma subseqs_subset:
  assumes "ys  $\in$  set (subseqs xs)"
  shows "set ys  $\subseteq$  set xs"
<proof>
```

2.1.3 List: prefixes, suffixes

```
lemma suffix_Cons': "suffix [x] (y#ys)  $\implies$  suffix [x] ys  $\vee$  (y = x  $\wedge$  ys = [])"
<proof>
```

```
lemma prefix_Cons': "prefix (x#xs) (x#ys)  $\implies$  prefix xs ys"
<proof>
```

```
lemma prefix_map: "prefix xs (map f ys)  $\implies$   $\exists$  zs. prefix zs ys  $\wedge$  map f zs = xs"
<proof>
```

```
lemma length_prefix_ex:
  assumes "n  $\leq$  length xs"
  shows " $\exists$  ys zs. xs = ys@zs  $\wedge$  length ys = n"
<proof>
```

```
lemma length_prefix_ex':
  assumes "n < length xs"
  shows " $\exists$  ys zs. xs = ys@xs ! n#zs  $\wedge$  length ys = n"
<proof>
```

```
lemma length_prefix_ex2:
  assumes "i < length xs" "j < length xs" "i < j"
  shows " $\exists$  ys zs vs. xs = ys@xs ! i#zs@xs ! j#vs  $\wedge$  length ys = i  $\wedge$  length zs = j - i - 1"
<proof>
```

2.1.4 List: products

```
lemma product_lists_Cons:
  "x#xs  $\in$  set (product_lists (y#ys))  $\longleftrightarrow$  (xs  $\in$  set (product_lists ys)  $\wedge$  x  $\in$  set y)"
<proof>
```

```
lemma product_lists_in_set_nth:
  assumes "xs  $\in$  set (product_lists ys)"
  shows " $\forall$  i < length ys. xs ! i  $\in$  set (ys ! i)"
<proof>
```

```
lemma product_lists_in_set_nth':
  assumes " $\forall$  i < length xs. ys ! i  $\in$  set (xs ! i)"
  and "length xs = length ys"
  shows "ys  $\in$  set (product_lists xs)"
<proof>
```

2.1.5 Other Lemmata

```
lemma inv_set_fset: "finite M  $\implies$  set (inv set M) = M"
```

<proof>

```
lemma lfp_eqI':
  assumes "mono f"
    and "f C = C"
    and " $\forall X \in \text{Pow } C. f X = X \longrightarrow X = C$ "
  shows "lfp f = C"
<proof>
```

```
lemma lfp_while':
  fixes f::"'a set  $\Rightarrow$  'a set" and M::"'a set"
  defines "N  $\equiv$  while ( $\lambda A. f A \neq A$ ) f {}"
  assumes f_mono: "mono f"
    and N_finite: "finite N"
    and N_supset: "f N  $\subseteq$  N"
  shows "lfp f = N"
<proof>
```

```
lemma lfp_while'':
  fixes f::"'a set  $\Rightarrow$  'a set" and M::"'a set"
  defines "N  $\equiv$  while ( $\lambda A. f A \neq A$ ) f {}"
  assumes f_mono: "mono f"
    and lfp_finite: "finite (lfp f)"
  shows "lfp f = N"
<proof>
```

```
lemma preordered_finite_set_has_maxima:
  assumes "finite A" "A  $\neq$  {}"
  shows " $\exists a::'a::\{\text{preorder}\} \in A. \forall b \in A. \neg(a < b)$ "
<proof>
```

```
lemma partition_index_bij:
  fixes n::nat
  obtains I k where
    "bij_betw I {0.. $n$ } {0.. $n$ }" "k  $\leq$  n"
    " $\forall i. i < k \longrightarrow P (I i)$ "
    " $\forall i. k \leq i \wedge i < n \longrightarrow \neg(P (I i))$ "
<proof>
```

```
lemma finite_lists_length_eq':
  assumes " $\bigwedge x. x \in \text{set } xs \implies \text{finite } \{y. P x y\}$ "
  shows "finite {ys. length xs = length ys  $\wedge$  ( $\forall y \in \text{set } ys. \exists x \in \text{set } xs. P x y$ )}"
<proof>
```

```
lemma trancl_eqI:
  assumes " $\forall (a,b) \in A. \forall (c,d) \in A. b = c \longrightarrow (a,d) \in A$ "
  shows "A = A+"
<proof>
```

```
lemma trancl_eqI':
  assumes " $\forall (a,b) \in A. \forall (c,d) \in A. b = c \wedge a \neq d \longrightarrow (a,d) \in A$ "
    and " $\forall (a,b) \in A. a \neq b$ "
  shows "A = {(a,b)  $\in$  A+. a  $\neq$  b}"
<proof>
```

```
lemma distinct_concat_idx_disjoint:
  assumes xs: "distinct (concat xs)"
    and ij: "i < length xs" "j < length xs" "i < j"
  shows "set (xs ! i)  $\cap$  set (xs ! j) = {}"
<proof>
```

```
lemma remdups_ex2:
  "length (remdups xs) > 1  $\implies \exists a \in \text{set } xs. \exists b \in \text{set } xs. a \neq b$ "
```

<proof>

```
lemma trancl_minus_refl_idem:
  defines "cl  $\equiv$   $\lambda ts. \{(a,b) \in ts^+. a \neq b\}$ "
  shows "cl (cl ts) = cl ts"
<proof>
```

2.1.6 Infinite Paths in Relations as Mappings from Naturals to States

context
begin

```
private fun rel_chain_fun::"nat  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  ('a  $\times$  'a) set  $\Rightarrow$  'a" where
  "rel_chain_fun 0 x _ _ = x"
| "rel_chain_fun (Suc i) x y r = (if i = 0 then y else SOME z. (rel_chain_fun i x y r, z)  $\in$  r)"
```

```
lemma infinite_chain_intro:
  fixes r::"('a  $\times$  'a) set"
  assumes " $\forall (a,b) \in r. \exists c. (b,c) \in r$ " "r  $\neq$  {}"
  shows " $\exists f. \forall i::nat. (f i, f (Suc i)) \in r$ "
<proof>
```

end

```
lemma infinite_chain_intro':
  fixes r::"('a  $\times$  'a) set"
  assumes base: " $\exists b. (x,b) \in r$ " and step: " $\forall b. (x,b) \in r^+ \longrightarrow (\exists c. (b,c) \in r)$ "
  shows " $\exists f. \forall i::nat. (f i, f (Suc i)) \in r$ "
<proof>
```

```
lemma infinite_chain_mono:
  assumes "S  $\subseteq$  T" " $\exists f. \forall i::nat. (f i, f (Suc i)) \in S$ "
  shows " $\exists f. \forall i::nat. (f i, f (Suc i)) \in T$ "
<proof>
```

end

2.2 Protocol Messages as (First-Order) Terms (Messages)

```
theory Messages
  imports Miscellaneous "First_Order_Terms.Term"
begin
```

2.2.1 Term-related definitions: subterms and free variables

```
abbreviation "the_Fun  $\equiv$  un_Fun1"
lemmas the_Fun_def = un_Fun1_def
```

```
fun subterms::"('a, 'b) term  $\Rightarrow$  ('a, 'b) terms" where
  "subterms (Var x) = {Var x}"
| "subterms (Fun f T) = {Fun f T}  $\cup$  ( $\bigcup t \in \text{set } T. \text{subterms } t$ )"
```

```
abbreviation subtermeq (infix " $\sqsubseteq$ " 50) where "t'  $\sqsubseteq$  t  $\equiv$  (t'  $\in$  subterms t)"
abbreviation subterm (infix " $\sqsubset$ " 50) where "t'  $\sqsubset$  t  $\equiv$  (t'  $\sqsubseteq$  t  $\wedge$  t'  $\neq$  t)"
```

```
abbreviation "subtermsset M  $\equiv$   $\bigcup$  (subterms ` M)"
abbreviation subtermeqset (infix " $\sqsubseteq_{\text{set}}$ " 50) where "t  $\sqsubseteq_{\text{set}}$  M  $\equiv$  (t  $\in$  subtermsset M)"
```

```
abbreviation fv where "fv  $\equiv$  vars_term"
lemmas fv_simps = term.simps(17,18)
```

```
fun fvset where "fvset M =  $\bigcup$  (fv ` M)"
```

abbreviation fv_{pair} where $"fv_{pair} p \equiv \text{case } p \text{ of } (t, t') \Rightarrow fv\ t \cup fv\ t'"$

fun fv_{pairs} where $"fv_{pairs} F = \bigcup (fv_{pair} \ ` \ \text{set } F)"$

abbreviation $ground$ where $"ground\ M \equiv fv_{set}\ M = \{\}"$

2.2.2 Variants that return lists instead of sets

fun fv_list where

$"fv_list\ (Var\ x) = [x]"$
 $| "fv_list\ (Fun\ f\ T) = \text{concat}\ (\text{map}\ fv_list\ T)"$

definition fv_list_{pairs} where

$"fv_list_{pairs}\ F \equiv \text{concat}\ (\text{map}\ (\lambda(t, t').\ fv_list\ t @ fv_list\ t')\ F)"$

fun $subterms_list::('a, 'b)\ \text{term} \Rightarrow ('a, 'b)\ \text{term}\ \text{list}$ where

$"subterms_list\ (Var\ x) = [Var\ x]"$
 $| "subterms_list\ (Fun\ f\ T) = \text{remdups}\ (\text{Fun}\ f\ T \# \text{concat}\ (\text{map}\ subterms_list\ T))"$

lemma $fv_list_is_fv$: $"fv\ t = \text{set}\ (fv_list\ t)"$

$\langle \text{proof} \rangle$

lemma $fv_list_{pairs}_is_fv_{pairs}$: $"fv_{pairs}\ F = \text{set}\ (fv_list_{pairs}\ F)"$

$\langle \text{proof} \rangle$

lemma $subterms_list_is_subterms$: $"subterms\ t = \text{set}\ (subterms_list\ t)"$

$\langle \text{proof} \rangle$

2.2.3 The subterm relation defined as a function

fun $subterm_of$ where

$"subterm_of\ t\ (Var\ y) = (t = Var\ y)"$
 $| "subterm_of\ t\ (Fun\ f\ T) = (t = Fun\ f\ T \vee \text{list_ex}\ (subterm_of\ t)\ T)"$

lemma $subterm_of_iff_subtermeq$ $[code_unfold]$: $"t \sqsubseteq t' = subterm_of\ t\ t'"$

$\langle \text{proof} \rangle$

lemma $subterm_of_ex_set_iff_subtermeqset$ $[code_unfold]$: $"t \sqsubseteq_{set}\ M = (\exists t' \in M.\ subterm_of\ t\ t'"$

$\langle \text{proof} \rangle$

2.2.4 The subterm relation is a partial order on terms

interpretation $"term"$: order $"(\sqsubseteq)"\ "(\sqsubset)"$

$\langle \text{proof} \rangle$

2.2.5 Lemmata concerning subterms and free variables

lemma $fv_list_{pairs}_append$: $"fv_list_{pairs}\ (F@G) = fv_list_{pairs}\ F @ fv_list_{pairs}\ G"$

$\langle \text{proof} \rangle$

lemma $distinct_fv_list_idx_fv_disjoint$:

assumes t : $"distinct\ (fv_list\ t)"\ "Fun\ f\ T \sqsubseteq t"$
 and ij : $"i < \text{length}\ T"\ "j < \text{length}\ T"\ "i < j"$
 shows $"fv\ (T ! i) \cap fv\ (T ! j) = \{\}"$

$\langle \text{proof} \rangle$

lemmas $subtermeqI'$ $[intro]$ = $term.eq_refl$

lemma $subtermeqI''$ $[intro]$: $"t \in \text{set}\ T \Longrightarrow t \sqsubseteq Fun\ f\ T"$

$\langle \text{proof} \rangle$

lemma $finite_fv_set$ $[intro]$: $"finite\ M \Longrightarrow finite\ (fv_{set}\ M)"$

<proof>

lemma *finite_fun_symbols[simp]*: "finite (funs_term t)"

<proof>

lemma *fv_set_mono*: " $M \subseteq N \implies \text{fv}_{\text{set}} M \subseteq \text{fv}_{\text{set}} N$ "

<proof>

lemma *subterms_set_mono*: " $M \subseteq N \implies \text{subterms}_{\text{set}} M \subseteq \text{subterms}_{\text{set}} N$ "

<proof>

lemma *ground_empty[simp]*: "ground {}"

<proof>

lemma *ground_subset*: " $M \subseteq N \implies \text{ground } N \implies \text{ground } M$ "

<proof>

lemma *fv_map_fv_set*: " $\bigcup (\text{set } (\text{map } \text{fv } L)) = \text{fv}_{\text{set}} (\text{set } L)$ "

<proof>

lemma *fv_set_union*: " $\text{fv}_{\text{set}} (M \cup N) = \text{fv}_{\text{set}} M \cup \text{fv}_{\text{set}} N$ "

<proof>

lemma *finite_subset_Union*:

fixes $A::\text{'a set}$ and $f::\text{'a} \Rightarrow \text{'b set}$

assumes "finite $(\bigcup a \in A. f a)$ "

shows " $\exists B. \text{finite } B \wedge B \subseteq A \wedge (\bigcup b \in B. f b) = (\bigcup a \in A. f a)$ "

<proof>

lemma *inv_set_fv*: "finite $M \implies \bigcup (\text{set } (\text{map } \text{fv } (\text{inv set } M))) = \text{fv}_{\text{set}} M$ "

<proof>

lemma *ground_subterm*: " $\text{fv } t = \{\} \implies t' \sqsubseteq t \implies \text{fv } t' = \{\}$ " *<proof>*

lemma *empty_fv_not_var*: " $\text{fv } t = \{\} \implies t \neq \text{Var } x$ " *<proof>*

lemma *empty_fv_exists_fun*: " $\text{fv } t = \{\} \implies \exists f X. t = \text{Fun } f X$ " *<proof>*

lemma *vars_iff_subtermeq*: " $x \in \text{fv } t \iff \text{Var } x \sqsubseteq t$ " *<proof>*

lemma *vars_iff_subtermeq_set*: " $x \in \text{fv}_{\text{set}} M \iff \text{Var } x \in \text{subterms}_{\text{set}} M$ "

<proof>

lemma *vars_if_subtermeq_set*: " $\text{Var } x \in \text{subterms}_{\text{set}} M \implies x \in \text{fv}_{\text{set}} M$ "

<proof>

lemma *subtermeq_set_if_vars*: " $x \in \text{fv}_{\text{set}} M \implies \text{Var } x \in \text{subterms}_{\text{set}} M$ "

<proof>

lemma *vars_iff_subterm_or_eq*: " $x \in \text{fv } t \iff \text{Var } x \sqsubset t \vee \text{Var } x = t$ "

<proof>

lemma *var_is_subterm*: " $x \in \text{fv } t \implies \text{Var } x \in \text{subterms } t$ "

<proof>

lemma *subterm_is_var*: " $\text{Var } x \in \text{subterms } t \implies x \in \text{fv } t$ "

<proof>

lemma *no_var_subterm*: " $\neg t \sqsubset \text{Var } v$ " *<proof>*

lemma *fun_if_subterm*: " $t \sqsubset u \implies \exists f X. u = \text{Fun } f X$ " *<proof>*

lemma *subtermeq_vars_subset*: " $M \sqsubseteq N \implies \text{fv } M \subseteq \text{fv } N$ " *<proof>*

```

lemma fv_subterms[simp]: "fvset (subterms t) = fv t"
⟨proof⟩

lemma fv_subterms_set[simp]: "fvset (subtermsset M) = fvset M"
⟨proof⟩

lemma fv_subset: "t ∈ M ⇒ fv t ⊆ fvset M"
⟨proof⟩

lemma fv_subset_subterms: "t ∈ subtermsset M ⇒ fv t ⊆ fvset M"
⟨proof⟩

lemma subterms_finite[simp]: "finite (subterms t)" ⟨proof⟩

lemma subterms_union_finite: "finite M ⇒ finite (⋃ t ∈ M. subterms t)"
⟨proof⟩

lemma subterms_subset: "t' ⊆ t ⇒ subterms t' ⊆ subterms t"
⟨proof⟩

lemma subterms_subset_set: "M ⊆ subterms t ⇒ subtermsset M ⊆ subterms t"
⟨proof⟩

lemma subset_subterms_Union[simp]: "M ⊆ subtermsset M" ⟨proof⟩

lemma in_subterms_Union: "t ∈ M ⇒ t ∈ subtermsset M" ⟨proof⟩

lemma in_subterms_subset_Union: "t ∈ subtermsset M ⇒ subterms t ⊆ subtermsset M"
⟨proof⟩

lemma subterm_param_split:
  assumes "t ⊆ Fun f X"
  shows "∃ pre x suf. t ⊆ x ∧ X = pre@x#suf"
⟨proof⟩

lemma ground_iff_no_vars: "ground (M::('a,'b) terms) ↔ (∀ v. Var v ∉ (⋃ m ∈ M. subterms m))"
⟨proof⟩

lemma index_Fun_subterms_subset[simp]: "i < length T ⇒ subterms (T ! i) ⊆ subterms (Fun f T)"
⟨proof⟩

lemma index_Fun_fv_subset[simp]: "i < length T ⇒ fv (T ! i) ⊆ fv (Fun f T)"
⟨proof⟩

lemma subterms_union_ground:
  assumes "ground M"
  shows "ground (subtermsset M)"
⟨proof⟩

lemma Var_subtermeq: "t ⊆ Var v ⇒ t = Var v" ⟨proof⟩

lemma subtermeq_imp_funs_term_subset: "s ⊆ t ⇒ funs_term s ⊆ funs_term t"
⟨proof⟩

lemma subterms_const: "subterms (Fun f []) = {Fun f []}" ⟨proof⟩

lemma subterm_subtermeq_neq: "[t ⊆ u; u ⊆ v] ⇒ t ≠ v"
⟨proof⟩

lemma subtermeq_subterm_neq: "[t ⊆ u; u ⊆ v] ⇒ t ≠ v"
⟨proof⟩

```

lemma `subterm_size_lt`: " $x \sqsubset y \implies \text{size } x < \text{size } y$ "
 $\langle \text{proof} \rangle$

lemma `in_subterms_eq`: " $\llbracket x \in \text{subterms } y; y \in \text{subterms } x \rrbracket \implies \text{subterms } x = \text{subterms } y$ "
 $\langle \text{proof} \rangle$

lemma `Fun_gt_params`: " $\text{Fun } f \ X \notin \bigcup_{x \in \text{set } X. \text{subterms } x}$ "
 $\langle \text{proof} \rangle$

lemma `params_subterms[simp]`: " $\text{set } X \subseteq \text{subterms } (\text{Fun } f \ X)$ " $\langle \text{proof} \rangle$

lemma `params_subterms_Union[simp]`: " $\text{subterms}_{\text{set}} (\text{set } X) \subseteq \text{subterms } (\text{Fun } f \ X)$ " $\langle \text{proof} \rangle$

lemma `Fun_subterm_inside_params`: " $t \sqsubset \text{Fun } f \ X \longleftrightarrow t \in \bigcup_{x \in (\text{set } X). \text{subterms } x}$ "
 $\langle \text{proof} \rangle$

lemma `Fun_param_is_subterm`: " $x \in \text{set } X \implies x \sqsubset \text{Fun } f \ X$ "
 $\langle \text{proof} \rangle$

lemma `Fun_param_in_subterms`: " $x \in \text{set } X \implies x \in \text{subterms } (\text{Fun } f \ X)$ "
 $\langle \text{proof} \rangle$

lemma `Fun_not_in_param`: " $x \in \text{set } X \implies \neg \text{Fun } f \ X \sqsubset x$ "
 $\langle \text{proof} \rangle$

lemma `Fun_ex_if_subterm`: " $t \sqsubset s \implies \exists f \ T. \text{Fun } f \ T \sqsubseteq s \wedge t \in \text{set } T$ "
 $\langle \text{proof} \rangle$

lemma `const_subterm_obtain`:
 assumes " $\text{fv } t = \{\}$ "
 obtains c where " $\text{Fun } c \ [] \sqsubseteq t$ "
 $\langle \text{proof} \rangle$

lemma `const_subterm_obtain'`: " $\text{fv } t = \{\} \implies \exists c. \text{Fun } c \ [] \sqsubseteq t$ "
 $\langle \text{proof} \rangle$

lemma `subterms_singleton`:
 assumes " $(\exists v. t = \text{Var } v) \vee (\exists f. t = \text{Fun } f \ [])$ "
 shows " $\text{subterms } t = \{t\}$ "
 $\langle \text{proof} \rangle$

lemma `subtermeq_Var_const`:
 assumes " $s \sqsubseteq t$ "
 shows " $t = \text{Var } v \implies s = \text{Var } v$ " " $t = \text{Fun } f \ [] \implies s = \text{Fun } f \ []$ "
 $\langle \text{proof} \rangle$

lemma `subterms_singleton'`:
 assumes " $\text{subterms } t = \{t\}$ "
 shows " $(\exists v. t = \text{Var } v) \vee (\exists f. t = \text{Fun } f \ [])$ "
 $\langle \text{proof} \rangle$

lemma `funs_term_subterms_eq[simp]`:
 " $\bigcup_{s \in \text{subterms } t. \text{funs_term } s} = \text{funs_term } t$ "
 " $\bigcup_{s \in \text{subterms}_{\text{set}} M. \text{funs_term } s} = \bigcup (\text{funs_term } ` M)$ "
 $\langle \text{proof} \rangle$

lemmas `subtermI'[intro]` = `Fun_param_is_subterm`

lemma `funs_term_Fun_subterm`: " $f \in \text{funs_term } t \implies \exists T. \text{Fun } f \ T \in \text{subterms } t$ "
 $\langle \text{proof} \rangle$

lemma `funs_term_Fun_subterm'`: " $\text{Fun } f \ T \in \text{subterms } t \implies f \in \text{funs_term } t$ "
 $\langle \text{proof} \rangle$


```

lemma zip_arg_subterm:
  assumes "(s,t) ∈ set (zip X Y)"
  shows "s ⊆ Fun f X" "t ⊆ Fun g Y"
⟨proof⟩

lemma fv_disj_Fun_subterm_param_cases:
  assumes "fv t ∩ X = {}" "Fun f T ∈ subterms t"
  shows "T = [] ∨ (∃ s ∈ set T. s ∉ Var ` X)"
⟨proof⟩

lemma fv_eq_FunI:
  assumes "length T = length S" "∧i. i < length T ⇒ fv (T ! i) = fv (S ! i)"
  shows "fv (Fun f T) = fv (Fun g S)"
⟨proof⟩

lemma fv_eq_FunI':
  assumes "length T = length S" "∧i. i < length T ⇒ x ∈ fv (T ! i) ↔ x ∈ fv (S ! i)"
  shows "x ∈ fv (Fun f T) ↔ x ∈ fv (Fun g S)"
⟨proof⟩

lemma finite_fv_pairs[simp]: "finite (fv_pairs x)" ⟨proof⟩

lemma fv_pairs_Nil[simp]: "fv_pairs [] = {}" ⟨proof⟩

lemma fv_pairs_singleton[simp]: "fv_pairs [(t,s)] = fv t ∪ fv s" ⟨proof⟩

lemma fv_pairs_Cons: "fv_pairs ((s,t)#F) = fv s ∪ fv t ∪ fv_pairs F" ⟨proof⟩

lemma fv_pairs_append: "fv_pairs (F@G) = fv_pairs F ∪ fv_pairs G" ⟨proof⟩

lemma fv_pairs_mono: "set M ⊆ set N ⇒ fv_pairs M ⊆ fv_pairs N" ⟨proof⟩

lemma fv_pairs_inI[intro]:
  "f ∈ set F ⇒ x ∈ fv_pair f ⇒ x ∈ fv_pairs F"
  "f ∈ set F ⇒ x ∈ fv (fst f) ⇒ x ∈ fv_pairs F"
  "f ∈ set F ⇒ x ∈ fv (snd f) ⇒ x ∈ fv_pairs F"
  "(t,s) ∈ set F ⇒ x ∈ fv t ⇒ x ∈ fv_pairs F"
  "(t,s) ∈ set F ⇒ x ∈ fv s ⇒ x ∈ fv_pairs F"
⟨proof⟩

lemma fv_pairs_cons_subset: "fv_pairs F ⊆ fv_pairs (f#F)"
⟨proof⟩

```

2.2.6 Other lemmata

```

lemma nonvar_term_has_composed_shallow_term:
  fixes t:: "('f, 'v) term"
  assumes "¬(∃ x. t = Var x)"
  shows "∃ f T. Fun f T ⊆ t ∧ (∀ s ∈ set T. (∃ c. s = Fun c []) ∨ (∃ x. s = Var x))"
⟨proof⟩

end

```

2.3 Definitions and Properties Related to Substitutions and Unification (More_Unification)

```

theory More_Unification
  imports Messages "First_Order_Terms.Unification"
begin

```

2.3.1 Substitutions

abbreviation `subst_apply_list` (infix `"·list"` 51) where
`"T ·list $\vartheta \equiv \text{map } (\lambda t. t \cdot \vartheta) T$ "`

abbreviation `subst_apply_pair` (infixl `"·p"` 60) where
`"d ·p $\vartheta \equiv (\text{case } d \text{ of } (t, t') \Rightarrow (t \cdot \vartheta, t' \cdot \vartheta))$ "`

abbreviation `subst_apply_pair_set` (infixl `"·pset"` 60) where
`"M ·pset $\vartheta \equiv (\lambda d. d \cdot_p \vartheta) ` M$ "`

definition `subst_apply_pairs` (infix `"·pairs"` 51) where
`"F ·pairs $\vartheta \equiv \text{map } (\lambda f. f \cdot_p \vartheta) F$ "`

abbreviation `subst_more_general_than` (infixl `" \preceq_o "` 50) where
`" $\sigma \preceq_o \vartheta \equiv \exists \gamma. \vartheta = \sigma \circ_s \gamma$ "`

abbreviation `subst_support` (infix `"supports"` 50) where
`" $\vartheta \text{ supports } \delta \equiv (\forall x. \vartheta x \cdot \delta = \delta x)$ "`

abbreviation `rm_var` where
`"rm_var v s $\equiv s(v := \text{Var } v)$ "`

abbreviation `rm_vars` where
`"rm_vars vs $\sigma \equiv (\lambda v. \text{if } v \in \text{vs then Var } v \text{ else } \sigma v)$ "`

definition `subst_elim` where
`"subst_elim $\sigma v \equiv \forall t. v \notin \text{fv } t \Rightarrow (t \cdot \sigma)$ "`

definition `subst_idem` where
`"subst_idem s $\equiv s \circ_s s = s$ "`

lemma `subst_support_def`: " `$\vartheta \text{ supports } \tau \iff \tau = \vartheta \circ_s \tau$` "
`<proof>`

lemma `subst_supportD`: " `$\vartheta \text{ supports } \delta \implies \vartheta \preceq_o \delta$` "
`<proof>`

lemma `rm_vars_empty[simp]`: "`rm_vars {} s = s`" "`rm_vars (set []) s = s`"
`<proof>`

lemma `rm_vars_singleton`: "`rm_vars {v} s = rm_var v s`"
`<proof>`

lemma `subst_apply_terms_empty`: "`M ·set Var = M`"
`<proof>`

lemma `subst_agreement`: "`(t · r = t · s) $\iff (\forall v \in \text{fv } t. \text{Var } v \cdot r = \text{Var } v \cdot s)$` "
`<proof>`

lemma `repl_invariance[dest?]`: "`v $\notin \text{fv } t \implies t \cdot s(v := u) = t \cdot s$` "
`<proof>`

lemma `subst_idx_map`:
 assumes " `$\forall i \in \text{set } I. i < \text{length } T$` "
 shows "`(map (!! T) I) ·list $\delta = \text{map } (!! (\text{map } (\lambda t. t \cdot \delta) T)) I$` "
`<proof>`

lemma `subst_idx_map'`:
 assumes " `$\forall i \in \text{fv}_{\text{set}} (\text{set } K). i < \text{length } T$` "
 shows "`(K ·list (!) T) ·list $\delta = K \cdot_{\text{list}} (!! (\text{map } (\lambda t. t \cdot \delta) T))$` " (is "`?A = ?B`")
`<proof>`

lemma `subst_remove_var`: " $v \notin \text{fv } s \implies v \notin \text{fv } (t \cdot \text{Var}(v := s))$ "
 <proof>

lemma `subst_set_map`: " $x \in \text{set } X \implies x \cdot s \in \text{set } (\text{map } (\lambda x. x \cdot s) X)$ "
 <proof>

lemma `subst_set_idx_map`:
 assumes " $\forall i \in I. i < \text{length } T$ "
 shows " $(!) T \cdot I \cdot_{\text{set}} \delta = (!) (\text{map } (\lambda t. t \cdot \delta) T) \cdot I$ " (is "?A = ?B")
 <proof>

lemma `subst_set_idx_map'`:
 assumes " $\forall i \in \text{fv}_{\text{set}} K. i < \text{length } T$ "
 shows " $K \cdot_{\text{set}} (!) T \cdot_{\text{set}} \delta = K \cdot_{\text{set}} (!) (\text{map } (\lambda t. t \cdot \delta) T)$ " (is "?A = ?B")
 <proof>

lemma `subst_term_list_obtain`:
 assumes " $\forall i < \text{length } T. \exists s. P (T ! i) s \wedge S ! i = s \cdot \delta$ "
 and " $\text{length } T = \text{length } S$ "
 shows " $\exists U. \text{length } T = \text{length } U \wedge (\forall i < \text{length } T. P (T ! i) (U ! i)) \wedge S = \text{map } (\lambda u. u \cdot \delta) U$ "
 <proof>

lemma `subst_mono`: " $t \sqsubseteq u \implies t \cdot s \sqsubseteq u \cdot s$ "
 <proof>

lemma `subst_mono_fv`: " $x \in \text{fv } t \implies s x \sqsubseteq t \cdot s$ "
 <proof>

lemma `subst_mono_neq`:
 assumes " $t \sqsubset u$ "
 shows " $t \cdot s \sqsubset u \cdot s$ "
 <proof>

lemma `subst_no_occs[dest]`: " $\neg \text{Var } v \sqsubseteq t \implies t \cdot \text{Var}(v := s) = t$ "
 <proof>

lemma `var_comp[simp]`: " $\sigma \circ_s \text{Var} = \sigma$ " " $\text{Var} \circ_s \sigma = \sigma$ "
 <proof>

lemma `subst_comp_all`: " $M \cdot_{\text{set}} (\delta \circ_s \vartheta) = (M \cdot_{\text{set}} \delta) \cdot_{\text{set}} \vartheta$ "
 <proof>

lemma `subst_all_mono`: " $M \subseteq M' \implies M \cdot_{\text{set}} s \subseteq M' \cdot_{\text{set}} s$ "
 <proof>

lemma `subst_comp_set_image`: " $(\delta \circ_s \vartheta) \cdot X = \delta \cdot X \cdot_{\text{set}} \vartheta$ "
 <proof>

lemma `subst_ground_ident[dest?]`: " $\text{fv } t = \{\} \implies t \cdot s = t$ "
 <proof>

lemma `subst_ground_ident_compose`:
 " $\text{fv } (\sigma x) = \{\} \implies (\sigma \circ_s \vartheta) x = \sigma x$ "
 " $\text{fv } (t \cdot \sigma) = \{\} \implies t \cdot (\sigma \circ_s \vartheta) = t \cdot \sigma$ "
 <proof>

lemma `subst_all_ground_ident[dest?]`: " $\text{ground } M \implies M \cdot_{\text{set}} s = M$ "
 <proof>

lemma `subst_eqI[intro]`: " $(\bigwedge t. t \cdot \sigma = t \cdot \vartheta) \implies \sigma = \vartheta$ "
 <proof>

lemma `subst_cong`: " $\llbracket \sigma = \sigma'; \vartheta = \vartheta' \rrbracket \implies (\sigma \circ_s \vartheta) = (\sigma' \circ_s \vartheta')$ "

<proof>

lemma subst_mgt_bot[simp]: "Var \preceq_o ϑ "

<proof>

lemma subst_mgt_refl[simp]: " $\vartheta \preceq_o \vartheta$ "

<proof>

lemma subst_mgt_trans: " $[\vartheta \preceq_o \delta; \delta \preceq_o \sigma] \implies \vartheta \preceq_o \sigma$ "

<proof>

lemma subst_mgt_comp: " $\vartheta \preceq_o \vartheta \circ_s \delta$ "

<proof>

lemma subst_mgt_comp': " $\vartheta \circ_s \delta \preceq_o \sigma \implies \vartheta \preceq_o \sigma$ "

<proof>

lemma var_self: " $(\lambda w. \text{if } w = v \text{ then Var } v \text{ else Var } w) = \text{Var}$ "

<proof>

lemma var_same[simp]: " $\text{Var}(v := t) = \text{Var} \longleftrightarrow t = \text{Var } v$ "

<proof>

lemma subst_eq_if_eq_vars: " $(\bigwedge v. (\text{Var } v) \cdot \vartheta = (\text{Var } v) \cdot \sigma) \implies \vartheta = \sigma$ "

<proof>

lemma subst_all_empty[simp]: " $\{\} \cdot_{set} \vartheta = \{\}$ "

<proof>

lemma subst_all_insert: " $(\text{insert } t \ M) \cdot_{set} \delta = \text{insert } (t \cdot \delta) \ (M \cdot_{set} \delta)$ "

<proof>

lemma subst_apply_fv_subset: " $\text{fv } t \subseteq V \implies \text{fv } (t \cdot \delta) \subseteq \text{fv}_{set} (\delta \setminus V)$ "

<proof>

lemma subst_apply_fv_empty:

assumes " $\text{fv } t = \{\}$ "

shows " $\text{fv } (t \cdot \sigma) = \{\}$ "

<proof>

lemma subst_compose_fv:

assumes " $\text{fv } (\vartheta \ x) = \{\}$ "

shows " $\text{fv } ((\vartheta \circ_s \sigma) \ x) = \{\}$ "

<proof>

lemma subst_compose_fv':

fixes $\vartheta \ \sigma :: ('a, 'b) \text{ subst}$

assumes " $y \in \text{fv } ((\vartheta \circ_s \sigma) \ x)$ "

shows " $\exists z. z \in \text{fv } (\vartheta \ x)$ "

<proof>

lemma subst_apply_fv_unfold: " $\text{fv } (t \cdot \delta) = \text{fv}_{set} (\delta \setminus \text{fv } t)$ "

<proof>

lemma subst_apply_fv_unfold': " $\text{fv } (t \cdot \delta) = (\bigcup v \in \text{fv } t. \text{fv } (\delta \ v))$ "

<proof>

lemma subst_apply_fv_union: " $\text{fv}_{set} (\delta \setminus V) \cup \text{fv } (t \cdot \delta) = \text{fv}_{set} (\delta \setminus (V \cup \text{fv } t))$ "

<proof>

lemma subst_elimI[intro]: " $(\bigwedge t. v \notin \text{fv } (t \cdot \sigma)) \implies \text{subst_elim } \sigma \ v$ "

<proof>

```

lemma subst_elimI'[intro]: "( $\bigwedge w. v \notin \text{fv } (\text{Var } w \cdot \vartheta)$ )  $\implies$  subst_elim  $\vartheta$  v"
<proof>

lemma subst_elimD[dest]: "subst_elim  $\sigma$  v  $\implies$  v  $\notin$  fv (t  $\cdot$   $\sigma$ )"
<proof>

lemma subst_elimD'[dest]: "subst_elim  $\sigma$  v  $\implies$   $\sigma$  v  $\neq$  Var v"
<proof>

lemma subst_elimD''[dest]: "subst_elim  $\sigma$  v  $\implies$  v  $\notin$  fv ( $\sigma$  w)"
<proof>

lemma subst_elim_rm_vars_dest[dest]:
  "subst_elim ( $\sigma :: ('a, 'b)$  subst) v  $\implies$  v  $\notin$  vs  $\implies$  subst_elim (rm_vars vs  $\sigma$ ) v"
<proof>

lemma occs_subst_elim: " $\neg$ Var v  $\sqsubseteq$  t  $\implies$  subst_elim (Var(v := t)) v  $\vee$  (Var(v := t)) = Var"
<proof>

lemma occs_subst_elim': " $\neg$ Var v  $\sqsubseteq$  t  $\implies$  subst_elim (Var(v := t)) v"
<proof>

lemma subst_elim_comp: "subst_elim  $\vartheta$  v  $\implies$  subst_elim ( $\delta \circ_s \vartheta$ ) v"
<proof>

lemma var_subst_idem: "subst_idem Var"
<proof>

lemma var_upd_subst_idem:
  assumes " $\neg$ Var v  $\sqsubseteq$  t" shows "subst_idem (Var(v := t))"
<proof>
    
```

2.3.2 Lemmata: Domain and Range of Substitutions

```

lemma range_vars_alt_def: "range_vars s  $\equiv$  fvset (subst_range s)"
<proof>

lemma subst_dom_var_finite[simp]: "finite (subst_domain Var)" <proof>

lemma subst_range_Var[simp]: "subst_range Var = {}" <proof>

lemma range_vars_Var[simp]: "range_vars Var = {}" <proof>

lemma finite_subst_img_if_finite_dom: "finite (subst_domain  $\sigma$ )  $\implies$  finite (range_vars  $\sigma$ )"
<proof>

lemma finite_subst_img_if_finite_dom': "finite (subst_domain  $\sigma$ )  $\implies$  finite (subst_range  $\sigma$ )"
<proof>

lemma subst_img_alt_def: "subst_range s = {t.  $\exists v. s v = t \wedge t \neq \text{Var } v$ }"
<proof>

lemma subst_fv_img_alt_def: "range_vars s = ( $\bigcup t \in \{t. \exists v. s v = t \wedge t \neq \text{Var } v\}. \text{fv } t$ )"
<proof>

lemma subst_domI[intro]: " $\sigma$  v  $\neq$  Var v  $\implies$  v  $\in$  subst_domain  $\sigma$ "
<proof>

lemma subst_imgI[intro]: " $\sigma$  v  $\neq$  Var v  $\implies$   $\sigma$  v  $\in$  subst_range  $\sigma$ "
<proof>

lemma subst_fv_imgI[intro]: " $\sigma$  v  $\neq$  Var v  $\implies$  fv ( $\sigma$  v)  $\subseteq$  range_vars  $\sigma$ "
<proof>
    
```

lemma *subst_domain_subst_Fun_single[simp]*:

"subst_domain (Var(x := Fun f T)) = {x}" (is "?A = ?B")

<proof>

lemma *subst_range_subst_Fun_single[simp]*:

"subst_range (Var(x := Fun f T)) = {Fun f T}" (is "?A = ?B")

<proof>

lemma *range_vars_subst_Fun_single[simp]*:

"range_vars (Var(x := Fun f T)) = fv (Fun f T)"

<proof>

lemma *var_renaming_is_Fun_iff*:

assumes "subst_range $\delta \subseteq$ range Var"

shows "is_Fun t = is_Fun (t · δ)"

<proof>

lemma *subst_fv_dom_img_subset*: "fv t \subseteq subst_domain $\vartheta \implies$ fv (t · ϑ) \subseteq range_vars ϑ "

<proof>

lemma *subst_fv_dom_img_subset_set*: "fv_{set} M \subseteq subst_domain $\vartheta \implies$ fv_{set} (M ·_{set} ϑ) \subseteq range_vars ϑ "

<proof>

lemma *subst_fv_dom_ground_if_ground_img*:

assumes "fv t \subseteq subst_domain s" "ground (subst_range s)"

shows "fv (t · s) = {}"

<proof>

lemma *subst_fv_dom_ground_if_ground_img'*:

assumes "fv t \subseteq subst_domain s" " $\bigwedge x. x \in$ subst_domain s \implies fv (s x) = {}"

shows "fv (t · s) = {}"

<proof>

lemma *subst_fv_unfold*: "fv (t · s) = (fv t - subst_domain s) \cup fv_{set} (s \setminus (fv t \cap subst_domain s))"

<proof>

lemma *subst_fv_unfold_ground_img*: "range_vars s = {} \implies fv (t · s) = fv t - subst_domain s"

<proof>

lemma *subst_img_update*:

" $\llbracket \sigma v = \text{Var } v; t \neq \text{Var } v \rrbracket \implies$ range_vars ($\sigma(v := t)$) = range_vars $\sigma \cup$ fv t"

<proof>

lemma *subst_dom_update1*: "v \notin subst_domain $\sigma \implies$ subst_domain ($\sigma(v := \text{Var } v)$) = subst_domain σ "

<proof>

lemma *subst_dom_update2*: "t \neq Var v \implies subst_domain ($\sigma(v := t)$) = insert v (subst_domain σ)"

<proof>

lemma *subst_dom_update3*: "t = Var v \implies subst_domain ($\sigma(v := t)$) = subst_domain $\sigma - \{v\}$ "

<proof>

lemma *var_not_in_subst_dom[elim]*: "v \notin subst_domain s \implies s v = Var v"

<proof>

lemma *subst_dom_vars_in_subst[elim]*: "v \in subst_domain s \implies s v \neq Var v"

<proof>

lemma *subst_not_dom_fixed*: " $\llbracket v \in$ fv t; v \notin subst_domain s $\rrbracket \implies$ v \in fv (t · s)" <proof>

lemma *subst_not_img_fixed*: " $\llbracket v \in$ fv (t · s); v \notin range_vars s $\rrbracket \implies$ v \in fv t"

<proof>

lemma `ground_range_vars[intro]`: "ground (subst_range s) \implies range_vars s = {}"
 <proof>

lemma `ground_subst_no_var[intro]`: "ground (subst_range s) \implies x \notin range_vars s"
 <proof>

lemma `ground_img_obtain_fun`:
 assumes "ground (subst_range s)" "x \in subst_domain s"
 obtains f T where "s x = Fun f T" "Fun f T \in subst_range s" "fv (Fun f T) = {}"
 <proof>

lemma `ground_term_subst_domain_fv_subset`:
 "fv (t \cdot δ) = {} \implies fv t \subseteq subst_domain δ "
 <proof>

lemma `ground_subst_range_empty_fv`:
 "ground (subst_range ϑ) \implies x \in subst_domain $\vartheta \implies$ fv (ϑ x) = {}"
 <proof>

lemma `subst_Var_notin_img`: "x \notin range_vars s \implies t \cdot s = Var x \implies t = Var x"
 <proof>

lemma `fv_in_subst_img`: "[|s v = t; t \neq Var v|] \implies fv t \subseteq range_vars s"
 <proof>

lemma `empty_dom_iff_empty_subst`: "subst_domain ϑ = {} \longleftrightarrow ϑ = Var" <proof>

lemma `subst_dom_cong`: " $(\bigwedge v t. \vartheta v = t \implies \delta v = t) \implies$ subst_domain $\vartheta \subseteq$ subst_domain δ "
 <proof>

lemma `subst_img_cong`: " $(\bigwedge v t. \vartheta v = t \implies \delta v = t) \implies$ range_vars $\vartheta \subseteq$ range_vars δ "
 <proof>

lemma `subst_dom_elim`: "subst_domain s \cap range_vars s = {} \implies fv (t \cdot s) \cap subst_domain s = {}"
 <proof>

lemma `subst_dom_insert_finite`: "finite (subst_domain s) = finite (subst_domain (s(v := t)))"
 <proof>

lemma `trm_subst_disj`: "t \cdot ϑ = t \implies fv t \cap subst_domain ϑ = {}"
 <proof>

lemma `trm_subst_ident[intro]`: "fv t \cap subst_domain ϑ = {} \implies t \cdot ϑ = t"
 <proof>

lemma `trm_subst_ident'[intro]`: "v \notin subst_domain $\vartheta \implies$ (Var v) \cdot ϑ = Var v"
 <proof>

lemma `trm_subst_ident''[intro]`: " $(\bigwedge x. x \in$ fv t $\implies \vartheta$ x = Var x) \implies t \cdot ϑ = t"
 <proof>

lemma `set_subst_ident`: "fv_{set} M \cap subst_domain ϑ = {} \implies M \cdot _{set} ϑ = M"
 <proof>

lemma `trm_subst_ident_subterms[intro]`:
 "fv t \cap subst_domain ϑ = {} \implies subterms t \cdot _{set} ϑ = subterms t"
 <proof>

lemma `trm_subst_ident_subterms'[intro]`:
 "v \notin fv t \implies subterms t \cdot _{set} Var(v := s) = subterms t"
 <proof>

lemma *const_mem_subst_cases*:

assumes "Fun c [] $\in M \cdot_{set} \vartheta$ "
shows "Fun c [] $\in M \vee$ Fun c [] $\in \vartheta \setminus fv_{set} M$ "

<proof>

lemma *const_mem_subst_cases'*:

assumes "Fun c [] $\in M \cdot_{set} \vartheta$ "
shows "Fun c [] $\in M \vee$ Fun c [] $\in subst_range \vartheta$ "

<proof>

lemma *fv_subterms_substI[intro]*: "y $\in fv t \implies \vartheta y \in subterms t \cdot_{set} \vartheta$ "

<proof>

lemma *fv_subterms_subst_eq[simp]*: "fv_{set} (subterms (t · ϑ)) = fv_{set} (subterms t ·_{set} ϑ)"

<proof>

lemma *fv_subterms_set_subst*: "fv_{set} (subterms_{set} M ·_{set} ϑ) = fv_{set} (subterms_{set} (M ·_{set} ϑ))"

<proof>

lemma *fv_subterms_set_subst'*: "fv_{set} (subterms_{set} M ·_{set} ϑ) = fv_{set} (M ·_{set} ϑ)"

<proof>

lemma *fv_subst_subset*: "x $\in fv t \implies fv (\vartheta x) \subseteq fv (t \cdot \vartheta)$ "

<proof>

lemma *fv_subst_subset'*: "fv s $\subseteq fv t \implies fv (s \cdot \vartheta) \subseteq fv (t \cdot \vartheta)$ "

<proof>

lemma *fv_subst_obtain_var*:

fixes $\delta::('a, 'b) subst$
assumes "x $\in fv (t \cdot \delta)$ "
shows " $\exists y \in fv t. x \in fv (\delta y)$ "

<proof>

lemma *set_subst_all_ident*: "fv_{set} (M ·_{set} ϑ) $\cap subst_domain \delta = \{\}$ $\implies M \cdot_{set} (\vartheta \circ_s \delta) = M \cdot_{set} \vartheta$ "

<proof>

lemma *subterms_subst*:

"subterms (t · d) = (subterms t ·_{set} d) \cup subterms_{set} (d \setminus (fv t \cap subst_domain d))"

<proof>

lemma *subterms_subst'*:

fixes $\vartheta::('a, 'b) subst$
assumes " $\forall x \in fv t. (\exists f. \vartheta x = Fun f []) \vee (\exists y. \vartheta x = Var y)$ "
shows "subterms (t · ϑ) = subterms t ·_{set} ϑ "

<proof>

lemma *subterms_subst''*:

fixes $\vartheta::('a, 'b) subst$
assumes " $\forall x \in fv_{set} M. (\exists f. \vartheta x = Fun f []) \vee (\exists y. \vartheta x = Var y)$ "
shows "subterms_{set} (M ·_{set} ϑ) = subterms_{set} M ·_{set} ϑ "

<proof>

lemma *subterms_subst_subterm*:

fixes $\vartheta::('a, 'b) subst$
assumes " $\forall x \in fv a. (\exists f. \vartheta x = Fun f []) \vee (\exists y. \vartheta x = Var y)$ "
and "b $\in subterms (a \cdot \vartheta)$ "
shows " $\exists c \in subterms a. c \cdot \vartheta = b$ "

<proof>

lemma *subterms_subst_subset*: "subterms t ·_{set} $\sigma \subseteq subterms (t \cdot \sigma)$ "

<proof>


```

lemma subterms_subst_subset': "subtermsset M ·set σ ⊆ subtermsset (M ·set σ)"
⟨proof⟩

lemma subtermsset_subst:
  fixes ϑ::('a,'b) subst
  assumes "t ∈ subtermsset (M ·set ϑ)"
  shows "t ∈ subtermsset M ·set ϑ ∨ (∃x ∈ fvset M. t ∈ subterms (ϑ x))"
⟨proof⟩

lemma rm_vars_dom: "subst_domain (rm_vars V s) = subst_domain s - V"
⟨proof⟩

lemma rm_vars_dom_subset: "subst_domain (rm_vars V s) ⊆ subst_domain s"
⟨proof⟩

lemma rm_vars_dom_eq':
  "subst_domain (rm_vars (UNIV - V) s) = subst_domain s ∩ V"
⟨proof⟩

lemma rm_vars_img: "subst_range (rm_vars V s) = s ` subst_domain (rm_vars V s)"
⟨proof⟩

lemma rm_vars_img_subset: "subst_range (rm_vars V s) ⊆ subst_range s"
⟨proof⟩

lemma rm_vars_img_fv_subset: "range_vars (rm_vars V s) ⊆ range_vars s"
⟨proof⟩

lemma rm_vars_fv_obtain:
  assumes "x ∈ fv (t · rm_vars X ϑ) - X"
  shows "∃y ∈ fv t - X. x ∈ fv (rm_vars X ϑ y)"
⟨proof⟩

lemma rm_vars_apply: "v ∈ subst_domain (rm_vars V s) ⇒ (rm_vars V s) v = s v"
⟨proof⟩

lemma rm_vars_apply': "subst_domain δ ∩ vs = {} ⇒ rm_vars vs δ = δ"
⟨proof⟩

lemma rm_vars_ident: "fv t ∩ vs = {} ⇒ t · (rm_vars vs ϑ) = t · ϑ"
⟨proof⟩

lemma rm_vars_fv_subset: "fv (t · rm_vars X ϑ) ⊆ fv t ∪ fv (t · ϑ)"
⟨proof⟩

lemma rm_vars_fv_disj:
  assumes "fv t ∩ X = {}" "fv (t · ϑ) ∩ X = {}"
  shows "fv (t · rm_vars X ϑ) ∩ X = {}"
⟨proof⟩

lemma rm_vars_ground_supports:
  assumes "ground (subst_range ϑ)"
  shows "rm_vars X ϑ supports ϑ"
⟨proof⟩

lemma rm_vars_split:
  assumes "ground (subst_range ϑ)"
  shows "ϑ = rm_vars X ϑ ◦s rm_vars (subst_domain ϑ - X) ϑ"
⟨proof⟩

lemma rm_vars_fv_img_disj:
  assumes "fv t ∩ X = {}" "X ∩ range_vars ϑ = {}"
  shows "fv (t · rm_vars X ϑ) ∩ X = {}"

```

<proof>

lemma *subst_apply_dom_ident*: " $t \cdot \vartheta = t \implies \text{subst_domain } \delta \subseteq \text{subst_domain } \vartheta \implies t \cdot \delta = t$ "

<proof>

lemma *rm_vars_subst_apply_ident*:

assumes " $t \cdot \vartheta = t$ "

shows " $t \cdot (\text{rm_vars } vs \ \vartheta) = t$ "

<proof>

lemma *rm_vars_subst_eq*:

" $t \cdot \delta = t \cdot \text{rm_vars } (\text{subst_domain } \delta - \text{subst_domain } \delta \cap \text{fv } t) \ \delta$ "

<proof>

lemma *rm_vars_subst_eq'*:

" $t \cdot \delta = t \cdot \text{rm_vars } (\text{UNIV} - \text{fv } t) \ \delta$ "

<proof>

lemma *rm_vars_comp*:

assumes " $\text{range_vars } \delta \cap vs = \{\}$ "

shows " $t \cdot \text{rm_vars } vs \ (\delta \circ_s \vartheta) = t \cdot (\text{rm_vars } vs \ \delta \circ_s \text{rm_vars } vs \ \vartheta)$ "

<proof>

lemma *rm_vars_fv_set_subst*:

assumes " $x \in \text{fv_set } (\text{rm_vars } X \ \vartheta \ \backslash Y)$ "

shows " $x \in \text{fv_set } (\vartheta \ \backslash Y) \vee x \in X$ "

<proof>

lemma *disj_dom_img_var_notin*:

assumes " $\text{subst_domain } \vartheta \cap \text{range_vars } \vartheta = \{\}$ " " $\vartheta \ v = t$ " " $t \neq \text{Var } v$ "

shows " $v \notin \text{fv } t$ " " $\forall v \in \text{fv } (t \cdot \vartheta). v \notin \text{subst_domain } \vartheta$ "

<proof>

lemma *subst_sends_dom_to_img*: " $v \in \text{subst_domain } \vartheta \implies \text{fv } (\text{Var } v \cdot \vartheta) \subseteq \text{range_vars } \vartheta$ "

<proof>

lemma *subst_sends_fv_to_img*: " $\text{fv } (t \cdot s) \subseteq \text{fv } t \cup \text{range_vars } s$ "

<proof>

lemma *ident_comp_subst_trm_if_disj*:

assumes " $\text{subst_domain } \sigma \cap \text{range_vars } \vartheta = \{\}$ " " $v \in \text{subst_domain } \vartheta$ "

shows " $(\vartheta \circ_s \sigma) \ v = \vartheta \ v$ "

<proof>

lemma *ident_comp_subst_trm_if_disj'*: " $\text{fv } (\vartheta \ v) \cap \text{subst_domain } \sigma = \{\} \implies (\vartheta \circ_s \sigma) \ v = \vartheta \ v$ "

<proof>

lemma *subst_idemI[intro]*: " $\text{subst_domain } \sigma \cap \text{range_vars } \sigma = \{\} \implies \text{subst_idem } \sigma$ "

<proof>

lemma *subst_idemI'[intro]*: " $\text{ground } (\text{subst_range } \sigma) \implies \text{subst_idem } \sigma$ "

<proof>

lemma *subst_idemE*: " $\text{subst_idem } \sigma \implies \text{subst_domain } \sigma \cap \text{range_vars } \sigma = \{\}$ "

<proof>

lemma *subst_idem_rm_vars*: " $\text{subst_idem } \vartheta \implies \text{subst_idem } (\text{rm_vars } X \ \vartheta)$ "

<proof>

lemma *subst_fv_bounded_if_img_bounded*: " $\text{range_vars } \vartheta \subseteq \text{fv } t \cup V \implies \text{fv } (t \cdot \vartheta) \subseteq \text{fv } t \cup V$ "

<proof>

lemma *subst_fv_bound_singleton*: " $\text{fv } (t \cdot \text{Var}(v := t')) \subseteq \text{fv } t \cup \text{fv } t'$ "

<proof>

lemma *subst_fv_bounded_if_img_bounded'*:

assumes "range_vars $\vartheta \subseteq \text{fv}_{\text{set}} M$ "

shows " $\text{fv}_{\text{set}} (M \cdot_{\text{set}} \vartheta) \subseteq \text{fv}_{\text{set}} M$ "

<proof>

lemma *ground_img_if_ground_subst*: " $(\bigwedge v t. s v = t \implies \text{fv } t = \{v\}) \implies \text{range_vars } s = \{v\}$ "

<proof>

lemma *ground_subst_fv_subset*: "ground (subst_range ϑ) $\implies \text{fv } (t \cdot \vartheta) \subseteq \text{fv } t$ "

<proof>

lemma *ground_subst_fv_subset'*: "ground (subst_range ϑ) $\implies \text{fv}_{\text{set}} (M \cdot_{\text{set}} \vartheta) \subseteq \text{fv}_{\text{set}} M$ "

<proof>

lemma *subst_to_var_is_var[elim]*: " $t \cdot s = \text{Var } v \implies \exists w. t = \text{Var } w$ "

<proof>

lemma *subst_dom_comp_inI*:

assumes " $y \notin \text{subst_domain } \sigma$ "

and " $y \in \text{subst_domain } \delta$ "

shows " $y \in \text{subst_domain } (\sigma \circ_s \delta)$ "

<proof>

lemma *subst_comp_notin_dom_eq*:

" $x \notin \text{subst_domain } \vartheta_1 \implies (\vartheta_1 \circ_s \vartheta_2) x = \vartheta_2 x$ "

<proof>

lemma *subst_dom_comp_eq*:

assumes " $\text{subst_domain } \vartheta \cap \text{range_vars } \sigma = \{v\}$ "

shows " $\text{subst_domain } (\vartheta \circ_s \sigma) = \text{subst_domain } \vartheta \cup \text{subst_domain } \sigma$ "

<proof>

lemma *subst_img_comp_subset[simp]*:

"range_vars $(\vartheta_1 \circ_s \vartheta_2) \subseteq \text{range_vars } \vartheta_1 \cup \text{range_vars } \vartheta_2$ "

<proof>

lemma *subst_img_comp_subset'*:

assumes " $t \in \text{subst_range } (\vartheta_1 \circ_s \vartheta_2)$ "

shows " $t \in \text{subst_range } \vartheta_2 \vee (\exists t' \in \text{subst_range } \vartheta_1. t = t' \cdot \vartheta_2)$ "

<proof>

lemma *subst_img_comp_subset''*:

"subterms_{set} (subst_range $(\vartheta_1 \circ_s \vartheta_2)$) \subseteq

subterms_{set} (subst_range ϑ_2) $\cup ((\text{subterms}_{\text{set}} (\text{subst_range } \vartheta_1)) \cdot_{\text{set}} \vartheta_2)$ "

<proof>

lemma *subst_img_comp_subset'''*:

"subterms_{set} (subst_range $(\vartheta_1 \circ_s \vartheta_2)$) - range Var \subseteq

subterms_{set} (subst_range ϑ_2) - range Var $\cup ((\text{subterms}_{\text{set}} (\text{subst_range } \vartheta_1) - \text{range Var}) \cdot_{\text{set}} \vartheta_2)$ "

<proof>

lemma *subst_img_comp_subset_const*:

assumes " $\text{Fun } c [] \in \text{subst_range } (\vartheta_1 \circ_s \vartheta_2)$ "

shows " $\text{Fun } c [] \in \text{subst_range } \vartheta_2 \vee \text{Fun } c [] \in \text{subst_range } \vartheta_1 \vee$

$(\exists x. \text{Var } x \in \text{subst_range } \vartheta_1 \wedge \vartheta_2 x = \text{Fun } c [])$ "

<proof>

lemma *subst_img_comp_subset_const'*:

fixes $\delta \tau :: ('f, 'v) \text{subst}$

assumes " $(\delta \circ_s \tau) x = \text{Fun } c []$ "

shows " $\delta x = \text{Fun } c [] \vee (\exists z. \delta x = \text{Var } z \wedge \tau z = \text{Fun } c [])$ "

<proof>

lemma subst_img_comp_subset_ground:
 assumes "ground (subst_range $\vartheta 1$)"
 shows "subst_range ($\vartheta 1 \circ_s \vartheta 2$) \subseteq subst_range $\vartheta 1 \cup$ subst_range $\vartheta 2$ "
<proof>

lemma subst_fv_dom_img_single:
 assumes " $v \notin \text{fv } t$ " " $\sigma v = t$ " " $\bigwedge w. v \neq w \implies \sigma w = \text{Var } w$ "
 shows "subst_domain $\sigma = \{v\}$ " "range_vars $\sigma = \text{fv } t$ "
<proof>

lemma subst_comp_upd1:
 " $\vartheta (v := t) \circ_s \sigma = (\vartheta \circ_s \sigma)(v := t \cdot \sigma)$ "
<proof>

lemma subst_comp_upd2:
 assumes " $v \notin \text{subst_domain } s$ " " $v \notin \text{range_vars } s$ "
 shows " $s(v := t) = s \circ_s (\text{Var}(v := t))$ "
<proof>

lemma ground_subst_dom_iff_img:
 "ground (subst_range σ) $\implies x \in \text{subst_domain } \sigma \iff \sigma x \in \text{subst_range } \sigma$ "
<proof>

lemma finite_dom_subst_exists:
 "finite $S \implies \exists \sigma :: ('f, 'v) \text{ subst. subst_domain } \sigma = S$ "
<proof>

lemma subst_inj_is_bij_betw_dom_img_if_ground_img:
 assumes "ground (subst_range σ)"
 shows "inj $\sigma \iff \text{bij_betw } \sigma (\text{subst_domain } \sigma) (\text{subst_range } \sigma)$ " (is "?A \iff ?B")
<proof>

lemma bij_finite_ground_subst_exists:
 assumes "finite ($S :: 'v \text{ set}$)" "infinite ($U :: ('f, 'v) \text{ term set}$)" "ground U "
 shows " $\exists \sigma :: ('f, 'v) \text{ subst. subst_domain } \sigma = S$
 $\wedge \text{bij_betw } \sigma (\text{subst_domain } \sigma) (\text{subst_range } \sigma)$
 $\wedge \text{subst_range } \sigma \subseteq U$ "
<proof>

lemma bij_finite_const_subst_exists:
 assumes "finite ($S :: 'v \text{ set}$)" "finite ($T :: 'f \text{ set}$)" "infinite ($U :: 'f \text{ set}$)"
 shows " $\exists \sigma :: ('f, 'v) \text{ subst. subst_domain } \sigma = S$
 $\wedge \text{bij_betw } \sigma (\text{subst_domain } \sigma) (\text{subst_range } \sigma)$
 $\wedge \text{subst_range } \sigma \subseteq ((\lambda c. \text{Fun } c []) \setminus (U - T))$ "
<proof>

lemma bij_finite_const_subst_exists':
 assumes "finite ($S :: 'v \text{ set}$)" "finite ($T :: ('f, 'v) \text{ terms}$)" "infinite ($U :: 'f \text{ set}$)"
 shows " $\exists \sigma :: ('f, 'v) \text{ subst. subst_domain } \sigma = S$
 $\wedge \text{bij_betw } \sigma (\text{subst_domain } \sigma) (\text{subst_range } \sigma)$
 $\wedge \text{subst_range } \sigma \subseteq ((\lambda c. \text{Fun } c []) \setminus U) - T$ "
<proof>

lemma bij_betw_iteI:
 assumes "bij_betw $f A B$ " "bij_betw $g C D$ " " $A \cap C = \{\}$ " " $B \cap D = \{\}$ "
 shows "bij_betw ($\lambda x. \text{if } x \in A \text{ then } f x \text{ else } g x$) ($A \cup C$) ($B \cup D$)"
<proof>

lemma subst_comp_split:
 assumes "subst_domain $\vartheta \cap \text{range_vars } \vartheta = \{\}$ "
 shows " $\vartheta = (\text{rm_vars } (\text{subst_domain } \vartheta - V) \vartheta) \circ_s (\text{rm_vars } V \vartheta)$ " (is ?P)

and " $\vartheta = (\text{rm_vars } V \ \vartheta) \circ_s (\text{rm_vars } (\text{subst_domain } \vartheta - V) \ \vartheta)$ " (is ?Q)
 <proof>

lemma subst_comp_eq_if_disjoint_vars:
 assumes " $(\text{subst_domain } \delta \cup \text{range_vars } \delta) \cap (\text{subst_domain } \gamma \cup \text{range_vars } \gamma) = \{\}$ "
 shows " $\gamma \circ_s \delta = \delta \circ_s \gamma$ "
 <proof>

lemma subst_eq_if_disjoint_vars_ground:
 fixes $\xi \ \delta :: ('f, 'v) \text{ subst}$
 assumes " $\text{subst_domain } \delta \cap \text{subst_domain } \xi = \{\}$ " "ground (subst_range ξ)" "ground (subst_range δ)"
 shows " $t \cdot \delta \cdot \xi = t \cdot \xi \cdot \delta$ "
 <proof>

lemma subst_img_bound: " $\text{subst_domain } \delta \cup \text{range_vars } \delta \subseteq \text{fv } t \implies \text{range_vars } \delta \subseteq \text{fv } (t \cdot \delta)$ "
 <proof>

lemma subst_all_fv_subset: " $\text{fv } t \subseteq \text{fv}_{\text{set}} M \implies \text{fv } (t \cdot \vartheta) \subseteq \text{fv}_{\text{set}} (M \cdot_{\text{set}} \vartheta)$ "
 <proof>

lemma subst_support_if_mgt_subst_idem:
 assumes " $\vartheta \preceq_o \delta$ " "subst_idem ϑ "
 shows " ϑ supports δ "
 <proof>

lemma subst_support_iff_mgt_if_subst_idem:
 assumes "subst_idem ϑ "
 shows " $\vartheta \preceq_o \delta \longleftrightarrow \vartheta$ supports δ "
 <proof>

lemma subst_support_comp:
 fixes $\vartheta \ \delta \ \mathcal{I} :: ('a, 'b) \text{ subst}$
 assumes " ϑ supports \mathcal{I} " " δ supports \mathcal{I} "
 shows " $(\vartheta \circ_s \delta)$ supports \mathcal{I} "
 <proof>

lemma subst_support_comp':
 fixes $\vartheta \ \delta \ \sigma :: ('a, 'b) \text{ subst}$
 assumes " ϑ supports δ "
 shows " ϑ supports $(\delta \circ_s \sigma)$ " " σ supports $\delta \implies \vartheta$ supports $(\sigma \circ_s \delta)$ "
 <proof>

lemma subst_support_comp_split:
 fixes $\vartheta \ \delta \ \mathcal{I} :: ('a, 'b) \text{ subst}$
 assumes " $(\vartheta \circ_s \delta)$ supports \mathcal{I} "
 shows " $\text{subst_domain } \vartheta \cap \text{range_vars } \vartheta = \{\} \implies \vartheta$ supports \mathcal{I} "
 and " $\text{subst_domain } \vartheta \cap \text{subst_domain } \delta = \{\} \implies \delta$ supports \mathcal{I} "
 <proof>

lemma subst_idem_support: " $\text{subst_idem } \vartheta \implies \vartheta$ supports $\vartheta \circ_s \delta$ "
 <proof>

lemma subst_idem_iff_self_support: " $\text{subst_idem } \vartheta \longleftrightarrow \vartheta$ supports ϑ "
 <proof>

lemma subterm_subst_neq: " $t \sqsubset t' \implies t \cdot s \neq t' \cdot s$ "
 <proof>

lemma fv_Fun_subst_neq: " $x \in \text{fv } (\text{Fun } f \ T) \implies \sigma \ x \neq \text{Fun } f \ T \cdot \sigma$ "
 <proof>

lemma subterm_subst_unfold:
 assumes " $t \sqsubset s \cdot \vartheta$ "

shows " $(\exists s'. s' \sqsubseteq s \wedge t = s' \cdot \vartheta) \vee (\exists x \in \text{fv } s. t \sqsubseteq \vartheta x)$ "
 <proof>

lemma *subterm_subst_img_subterm*:
assumes " $t \sqsubseteq s \cdot \vartheta$ " " $\bigwedge s'. s' \sqsubseteq s \implies t \neq s' \cdot \vartheta$ "
shows " $\exists w \in \text{fv } s. t \sqsubseteq \vartheta w$ "
 <proof>

lemma *subterm_subst_not_img_subterm*:
assumes " $t \sqsubseteq s \cdot \mathcal{I}$ " " $\neg(\exists w \in \text{fv } s. t \sqsubseteq \mathcal{I} w)$ "
shows " $\exists f T. \text{Fun } f T \sqsubseteq s \wedge t = \text{Fun } f T \cdot \mathcal{I}$ "
 <proof>

lemma *subst_apply_img_var*:
assumes " $v \in \text{fv } (t \cdot \delta)$ " " $v \notin \text{fv } t$ "
obtains w **where** " $w \in \text{fv } t$ " " $v \in \text{fv } (\delta w)$ "
 <proof>

lemma *subst_apply_img_var'*:
assumes " $x \in \text{fv } (t \cdot \delta)$ " " $x \notin \text{fv } t$ "
shows " $\exists y \in \text{fv } t. x \in \text{fv } (\delta y)$ "
 <proof>

lemma *nth_map_subst*:
fixes $\vartheta::('f, 'v) \text{ subst}$ **and** $T::('f, 'v) \text{ term list}$ **and** $i::\text{nat}$
shows " $i < \text{length } T \implies (\text{map } (\lambda t. t \cdot \vartheta) T) ! i = (T ! i) \cdot \vartheta$ "
 <proof>

lemma *subst_subterm*:
assumes " $\text{Fun } f T \sqsubseteq t \cdot \vartheta$ "
shows " $(\exists S. \text{Fun } f S \sqsubseteq t \wedge \text{Fun } f S \cdot \vartheta = \text{Fun } f T) \vee$
 $(\exists s \in \text{subst_range } \vartheta. \text{Fun } f T \sqsubseteq s)$ "
 <proof>

lemma *subst_subterm'*:
assumes " $\text{Fun } f T \sqsubseteq t \cdot \vartheta$ "
shows " $\exists S. \text{length } S = \text{length } T \wedge (\text{Fun } f S \sqsubseteq t \vee (\exists s \in \text{subst_range } \vartheta. \text{Fun } f S \sqsubseteq s))$ "
 <proof>

lemma *subst_subterm''*:
assumes " $s \in \text{subterms } (t \cdot \vartheta)$ "
shows " $(\exists u \in \text{subterms } t. s = u \cdot \vartheta) \vee s \in \text{subterms}_{\text{set}} (\text{subst_range } \vartheta)$ "
 <proof>

2.3.3 More Small Lemmata

lemma *funs_term_subst*: " $\text{funs_term } (t \cdot \vartheta) = \text{funs_term } t \cup (\bigcup x \in \text{fv } t. \text{funs_term } (\vartheta x))$ "
 <proof>

lemma *fv_set_subst_img_eq*:
assumes " $X \cap (\text{subst_domain } \delta \cup \text{range_vars } \delta) = \{\}$ "
shows " $\text{fv}_{\text{set}} (\delta \cdot (Y - X)) = \text{fv}_{\text{set}} (\delta \cdot Y) - X$ "
 <proof>

lemma *subst_Fun_index_eq*:
assumes " $i < \text{length } T$ " " $\text{Fun } f T \cdot \delta = \text{Fun } g T' \cdot \delta$ "
shows " $T ! i \cdot \delta = T' ! i \cdot \delta$ "
 <proof>

lemma *fv_exists_if_unifiable_and_neq*:
fixes $t t'::('a, 'b) \text{ term}$ **and** $\delta \vartheta::('a, 'b) \text{ subst}$
assumes " $t \neq t'$ " " $t \cdot \vartheta = t' \cdot \vartheta$ "
shows " $\text{fv } t \cup \text{fv } t' \neq \{\}$ "

<proof>

lemma `const_subterm_subst`: " $\text{Fun } c \ [] \sqsubseteq t \implies \text{Fun } c \ [] \sqsubseteq t \cdot \sigma$ "

<proof>

lemma `const_subterm_subst_var_obtain`:

assumes " $\text{Fun } c \ [] \sqsubseteq t \cdot \sigma$ " " $\neg \text{Fun } c \ [] \sqsubseteq t$ "

obtains x **where** " $x \in \text{fv } t$ " " $\text{Fun } c \ [] \sqsubseteq \sigma x$ "

<proof>

lemma `const_subterm_subst_cases`:

assumes " $\text{Fun } c \ [] \sqsubseteq t \cdot \sigma$ "

shows " $\text{Fun } c \ [] \sqsubseteq t \vee (\exists x \in \text{fv } t. x \in \text{subst_domain } \sigma \wedge \text{Fun } c \ [] \sqsubseteq \sigma x)$ "

<proof>

lemma `fv_pairs_subst_fv_subset`:

assumes " $x \in \text{fv_pairs } F$ "

shows " $\text{fv } (\vartheta x) \subseteq \text{fv_pairs } (F \cdot_{\text{pairs}} \vartheta)$ "

<proof>

lemma `fv_pairs_step_subst`: " $\text{fv_set } (\delta \ ` \text{fv_pairs } F) = \text{fv_pairs } (F \cdot_{\text{pairs}} \delta)$ "

<proof>

lemma `fv_pairs_subst_obtain_var`:

fixes δ :: "('a, 'b) subst"

assumes " $x \in \text{fv_pairs } (F \cdot_{\text{pairs}} \delta)$ "

shows " $\exists y \in \text{fv_pairs } F. x \in \text{fv } (\delta y)$ "

<proof>

lemma `pair_subst_ident[intro]`: " $(\text{fv } t \cup \text{fv } t') \cap \text{subst_domain } \vartheta = \{\} \implies (t, t') \cdot_p \vartheta = (t, t')$ "

<proof>

lemma `pairs_substI[intro]`:

assumes " $\text{subst_domain } \vartheta \cap (\bigcup (s, t) \in M. \text{fv } s \cup \text{fv } t) = \{\}$ "

shows " $M \cdot_{\text{pset}} \vartheta = M$ "

<proof>

lemma `fv_pairs_subst`: " $\text{fv_pairs } (F \cdot_{\text{pairs}} \vartheta) = \text{fv_set } (\vartheta \ ` (\text{fv_pairs } F))$ "

<proof>

lemma `fv_pairs_subst_subset`:

assumes " $\text{fv_pairs } (F \cdot_{\text{pairs}} \delta) \subseteq \text{subst_domain } \sigma$ "

shows " $\text{fv_pairs } F \subseteq \text{subst_domain } \sigma \cup \text{subst_domain } \delta$ "

<proof>

lemma `pairs_subst_comp`: " $F \cdot_{\text{pairs}} \delta \circ_s \vartheta = ((F \cdot_{\text{pairs}} \delta) \cdot_{\text{pairs}} \vartheta)$ "

<proof>

lemma `pairs_substI'[intro]`:

" $\text{subst_domain } \vartheta \cap \text{fv_pairs } F = \{\} \implies F \cdot_{\text{pairs}} \vartheta = F$ "

<proof>

lemma `subst_pair_compose[simp]`: " $d \cdot_p (\delta \circ_s \mathcal{I}) = d \cdot_p \delta \cdot_p \mathcal{I}$ "

<proof>

lemma `subst_pairs_compose[simp]`: " $D \cdot_{\text{pset}} (\delta \circ_s \mathcal{I}) = D \cdot_{\text{pset}} \delta \cdot_{\text{pset}} \mathcal{I}$ "

<proof>

lemma `subst_apply_pair_pair`: " $(t, s) \cdot_p \mathcal{I} = (t \cdot \mathcal{I}, s \cdot \mathcal{I})$ "

<proof>

lemma `subst_apply_pairs_nil[simp]`: " $[] \cdot_{\text{pairs}} \delta = []$ "

<proof>

lemma `subst_apply_pairs_singleton[simp]`: " $[(t,s)] \cdot_{pairs} \delta = [(t \cdot \delta, s \cdot \delta)]$ "
 $\langle proof \rangle$

lemma `subst_apply_pairs_Var[iff]`: " $F \cdot_{pairs} Var = F$ " $\langle proof \rangle$

lemma `subst_apply_pairs_pset_subst`: " $set (F \cdot_{pairs} \vartheta) = set F \cdot_{pset} \vartheta$ "
 $\langle proof \rangle$

2.3.4 Finite Substitutions

inductive_set `fsubst`: " (a, b) subst set" where
`fvar`: " $Var \in fsubst$ "
`FUpdate`: " $\llbracket \vartheta \in fsubst; v \notin subst_domain \vartheta; t \neq Var v \rrbracket \implies \vartheta(v := t) \in fsubst$ "

lemma `finite_dom_iff_fsubst`:
" $finite (subst_domain \vartheta) \longleftrightarrow \vartheta \in fsubst$ "
 $\langle proof \rangle$

lemma `fsubst_induct[case_names fvar FUpdate, induct set: finite]`:
`assumes` " $finite (subst_domain \delta)$ " " $P Var$ "
`and` " $\bigwedge \vartheta v t. \llbracket finite (subst_domain \vartheta); v \notin subst_domain \vartheta; t \neq Var v; P \vartheta \rrbracket \implies P (\vartheta(v := t))$ "
`shows` " $P \delta$ "
 $\langle proof \rangle$

lemma `fun_upd_fsubst`: " $s(v := t) \in fsubst \longleftrightarrow s \in fsubst$ "
 $\langle proof \rangle$

lemma `finite_img_if_fsubst`: " $s \in fsubst \implies finite (subst_range s)$ "
 $\langle proof \rangle$

2.3.5 Unifiers and Most General Unifiers (MGUs)

abbreviation `Unifier`: " (f, v) subst $\implies (f, v)$ term $\implies (f, v)$ term $\implies bool$ " where
" $Unifier \sigma t u \equiv (t \cdot \sigma = u \cdot \sigma)$ "

abbreviation `MGU`: " (f, v) subst $\implies (f, v)$ term $\implies (f, v)$ term $\implies bool$ " where
" $MGU \sigma t u \equiv Unifier \sigma t u \wedge (\forall \vartheta. Unifier \vartheta t u \longrightarrow \sigma \preceq_o \vartheta)$ "

lemma `MGUI[intro]`:
`shows` " $\llbracket t \cdot \sigma = u \cdot \sigma; \bigwedge \vartheta :: (f, v) \text{ subst}. t \cdot \vartheta = u \cdot \vartheta \rrbracket \implies MGU \sigma t u$ "
 $\langle proof \rangle$

lemma `UnifierD[dest]`:
`fixes` $\sigma :: (f, v) \text{ subst}$ and $f g :: f$ and $X Y :: (f, v) \text{ term list}$
`assumes` " $Unifier \sigma (Fun f X) (Fun g Y)$ "
`shows` " $f = g$ " " $length X = length Y$ "
 $\langle proof \rangle$

lemma `MGUD[dest]`:
`fixes` $\sigma :: (f, v) \text{ subst}$ and $f g :: f$ and $X Y :: (f, v) \text{ term list}$
`assumes` " $MGU \sigma (Fun f X) (Fun g Y)$ "
`shows` " $f = g$ " " $length X = length Y$ "
 $\langle proof \rangle$

lemma `MGU_sym[sym]`: " $MGU \sigma s t \implies MGU \sigma t s$ " $\langle proof \rangle$

lemma `Unifier_sym[sym]`: " $Unifier \sigma s t \implies Unifier \sigma t s$ " $\langle proof \rangle$

lemma `MGU_nil`: " $MGU Var s t \longleftrightarrow s = t$ " $\langle proof \rangle$

lemma `Unifier_comp`: " $Unifier (\vartheta \circ_s \delta) t u \implies Unifier \delta (t \cdot \vartheta) (u \cdot \vartheta)$ "
 $\langle proof \rangle$

lemma *Unifier_comp'*: "Unifier δ (t · ϑ) (u · ϑ) \implies Unifier ($\vartheta \circ_s \delta$) t u"
 <proof>

lemma *Unifier_excludes_subterm*:
 assumes ϑ : "Unifier ϑ t u"
 shows " $\neg t \sqsubset u$ "
 <proof>

lemma *MGU_is_Unifier*: "MGU σ t u \implies Unifier σ t u" <proof>

lemma *MGU_Var1*:
 assumes " $\neg \text{Var } v \sqsubset t$ "
 shows "MGU (Var(v := t)) (Var v) t"
 <proof>

lemma *MGU_Var2*: " $v \notin \text{fv } t \implies$ MGU (Var(v := t)) (Var v) t"
 <proof>

lemma *MGU_Var3*: "MGU Var (Var v) (Var w) \longleftrightarrow v = w" <proof>

lemma *MGU_Const1*: "MGU Var (Fun c []) (Fun d []) \longleftrightarrow c = d" <proof>

lemma *MGU_Const2*: "MGU ϑ (Fun c []) (Fun d []) \implies c = d" <proof>

lemma *MGU_Fun*:
 assumes "MGU ϑ (Fun f X) (Fun g Y)"
 shows "f = g" "length X = length Y"
 <proof>

lemma *Unifier_Fun*:
 assumes "Unifier ϑ (Fun f (x#X)) (Fun g (y#Y))"
 shows "Unifier ϑ x y" "Unifier ϑ (Fun f X) (Fun g Y)"
 <proof>

lemma *Unifier_subst_idem_subst*:
 "subst_idem r \implies Unifier s (t · r) (u · r) \implies Unifier (r \circ_s s) (t · r) (u · r)"
 <proof>

lemma *subst_idem_comp*:
 "subst_idem r \implies Unifier s (t · r) (u · r) \implies
 ($\wedge q$. Unifier q (t · r) (u · r) \implies s \circ_s q = q) \implies
 subst_idem (r \circ_s s)"
 <proof>

lemma *Unifier_mgt*: "[Unifier δ t u; $\delta \preceq_s \vartheta$] \implies Unifier ϑ t u" <proof>

lemma *Unifier_support*: "[Unifier δ t u; δ supports ϑ] \implies Unifier ϑ t u"
 <proof>

lemma *MGU_mgt*: "[MGU σ t u; MGU δ t u] \implies $\sigma \preceq_s \delta$ " <proof>

lemma *Unifier_trm_fv_bound*:
 "[Unifier s t u; v \in fv t] \implies v \in subst_domain s \cup range_vars s \cup fv u"
 <proof>

lemma *Unifier_rm_var*: "[Unifier ϑ s t; v \notin fv s \cup fv t] \implies Unifier (rm_var v ϑ) s t"
 <proof>

lemma *Unifier_ground_rm_vars*:
 assumes "ground (subst_range s)" "Unifier (rm_vars X s) t t"
 shows "Unifier s t t"
 <proof>

```

lemma Unifier_dom_restrict:
  assumes "Unifier s t" "fv t  $\cup$  fv t'  $\subseteq$  S"
  shows "Unifier (rm_vars (UNIV - S) s) t t'"
<proof>

```

2.3.6 Well-formedness of Substitutions and Unifiers

```

inductive_set wf_subst_set::('a,'b) subst set where
  Empty[simp]: "Var  $\in$  wf_subst_set"
| Insert[simp]:
  "[ $\vartheta \in$  wf_subst_set; v  $\notin$  subst_domain  $\vartheta$ ;
  v  $\notin$  range_vars  $\vartheta$ ; fv t  $\cap$  (insert v (subst_domain  $\vartheta$ )) = {}]"
   $\implies$   $\vartheta(v := t) \in$  wf_subst_set"

```

```

definition wf_subst::('a,'b) subst  $\Rightarrow$  bool where
  "wf_subst  $\vartheta \equiv$  subst_domain  $\vartheta \cap$  range_vars  $\vartheta = \{\}$   $\wedge$  finite (subst_domain  $\vartheta$ )"

```

```

definition wf_MGU::('a,'b) subst  $\Rightarrow$  ('a,'b) term  $\Rightarrow$  ('a,'b) term  $\Rightarrow$  bool where
  "wf_MGU  $\vartheta$  s t  $\equiv$  wf_subst  $\vartheta \wedge$  MGU  $\vartheta$  s t  $\wedge$  subst_domain  $\vartheta \cup$  range_vars  $\vartheta \subseteq$  fv s  $\cup$  fv t"

```

```

lemma wf_subst_subst_idem: "wf_subst  $\vartheta \implies$  subst_idem  $\vartheta$ " <proof>

```

```

lemma wf_subst_properties: " $\vartheta \in$  wf_subst_set = wf_subst  $\vartheta$ "
<proof>

```

```

lemma wf_subst_induct[consumes 1, case_names Empty Insert]:
  assumes "wf_subst  $\delta$ " "P Var"
  and " $\wedge \vartheta$  v t. [wf_subst  $\vartheta$ ; P  $\vartheta$ ; v  $\notin$  subst_domain  $\vartheta$ ; v  $\notin$  range_vars  $\vartheta$ ;
  fv t  $\cap$  insert v (subst_domain  $\vartheta$ ) = {}]"
   $\implies$  P ( $\vartheta(v := t)$ )"
  shows "P  $\delta$ "
<proof>

```

```

lemma wf_subst_fsubst: "wf_subst  $\delta \implies \delta \in$  fsubst"
<proof>

```

```

lemma wf_subst_nil: "wf_subst Var" <proof>

```

```

lemma wf_MGU_nil: "MGU Var s t  $\implies$  wf_MGU Var s t"
<proof>

```

```

lemma wf_MGU_dom_bound: "wf_MGU  $\vartheta$  s t  $\implies$  subst_domain  $\vartheta \subseteq$  fv s  $\cup$  fv t" <proof>

```

```

lemma wf_subst_single:
  assumes "v  $\notin$  fv t" " $\sigma$  v = t" " $\wedge w. v \neq w \implies \sigma w =$  Var w"
  shows "wf_subst  $\sigma$ "
<proof>

```

```

lemma wf_subst_reduction:
  "wf_subst s  $\implies$  wf_subst (rm_var v s)"
<proof>

```

```

lemma wf_subst_compose:
  assumes "wf_subst  $\vartheta1$ " "wf_subst  $\vartheta2$ "
  and "subst_domain  $\vartheta1 \cap$  subst_domain  $\vartheta2 = \{\}$ "
  and "subst_domain  $\vartheta1 \cap$  range_vars  $\vartheta2 = \{\}$ "
  shows "wf_subst ( $\vartheta1 \circ_s \vartheta2$ )"
<proof>

```

```

lemma wf_subst_append:
  fixes  $\vartheta1$   $\vartheta2$ ::('f,'v) subst"
  assumes "wf_subst  $\vartheta1$ " "wf_subst  $\vartheta2$ "
  and "subst_domain  $\vartheta1 \cap$  subst_domain  $\vartheta2 = \{\}$ "

```

```

    and "subst_domain  $\vartheta_1 \cap \text{range\_vars } \vartheta_2 = \{\}$ "
    and "range_vars  $\vartheta_1 \cap \text{subst\_domain } \vartheta_2 = \{\}$ "
    shows "wf_subst ( $\lambda v. \text{if } \vartheta_1 v = \text{Var } v \text{ then } \vartheta_2 v \text{ else } \vartheta_1 v$ )"
  <proof>

lemma wf_subst_elim_append:
  assumes "wf_subst  $\vartheta$ " "subst_elim  $\vartheta v$ " " $v \notin \text{fv } t$ "
  shows "subst_elim ( $\vartheta(w := t)$ )  $v$ "
  <proof>

lemma wf_subst_elim_dom:
  assumes "wf_subst  $\vartheta$ "
  shows " $\forall v \in \text{subst\_domain } \vartheta. \text{subst\_elim } \vartheta v$ "
  <proof>

lemma wf_subst_support_iff_mgt: "wf_subst  $\vartheta \implies \vartheta \text{ supports } \delta \iff \vartheta \preceq_o \delta$ "
  <proof>

```

2.3.7 Interpretations

```

abbreviation interpretation_subst:: "('a, 'b) subst  $\implies$  bool" where
  "interpretation_subst  $\vartheta \equiv \text{subst\_domain } \vartheta = \text{UNIV} \wedge \text{ground } (\text{subst\_range } \vartheta)$ "

lemma interpretation_substI:
  " $(\bigwedge v. \text{fv } (\vartheta v) = \{\}) \implies \text{interpretation\_subst } \vartheta$ "
  <proof>

lemma interpretation_grounds[simp]:
  "interpretation_subst  $\vartheta \implies \text{fv } (t \cdot \vartheta) = \{\}$ "
  <proof>

lemma interpretation_grounds_all:
  "interpretation_subst  $\vartheta \implies (\bigwedge v. \text{fv } (\vartheta v) = \{\})$ "
  <proof>

lemma interpretation_grounds_all':
  "interpretation_subst  $\vartheta \implies \text{ground } (M \cdot_{\text{set}} \vartheta)$ "
  <proof>

lemma interpretation_comp:
  assumes "interpretation_subst  $\vartheta$ "
  shows "interpretation_subst ( $\sigma \circ_s \vartheta$ )" "interpretation_subst ( $\vartheta \circ_s \sigma$ )"
  <proof>

lemma interpretation_subst_exists:
  " $\exists \mathcal{I}:: ('f, 'v) \text{ subst. interpretation\_subst } \mathcal{I}$ "
  <proof>

lemma interpretation_subst_exists':
  " $\exists \vartheta:: ('f, 'v) \text{ subst. subst\_domain } \vartheta = X \wedge \text{ground } (\text{subst\_range } \vartheta)$ "
  <proof>

lemma interpretation_subst_idem:
  "interpretation_subst  $\vartheta \implies \text{subst\_idem } \vartheta$ "
  <proof>

lemma subst_idem_comp_upd_eq:
  assumes " $v \notin \text{subst\_domain } \mathcal{I}$ " "subst_idem  $\vartheta$ "
  shows " $\mathcal{I} \circ_s \vartheta = \mathcal{I}(v := \vartheta v) \circ_s \vartheta$ "
  <proof>

lemma interpretation_dom_img_disjoint:
  "interpretation_subst  $\mathcal{I} \implies \text{subst\_domain } \mathcal{I} \cap \text{range\_vars } \mathcal{I} = \{\}$ "

```

<proof>

2.3.8 Basic Properties of MGUs

lemma *MGU_is_mgu_singleton*: "MGU ϑ s t = is_mgu ϑ $\{(s,t)\}$ "

<proof>

lemma *Unifier_in_unifiers_singleton*: "Unifier ϑ s t \longleftrightarrow $\vartheta \in$ unifiers $\{(s,t)\}$ "

<proof>

lemma *subst_list_singleton_fv_subset*:

" $(\bigcup x \in$ set (subst_list (subst v t) E). fv (fst x) \cup fv (snd x))
 \subseteq fv $t \cup (\bigcup x \in$ set E . fv (fst x) \cup fv (snd x))"

<proof>

lemma *subst_of_dom_subset*: "subst_domain (subst_of L) \subseteq set (map fst L)"

<proof>

lemma *wf_MGU_is_imgu_singleton*: "wf_{MGU} ϑ s t \implies is_imgu ϑ $\{(s,t)\}$ "

<proof>

lemma *mgu_subst_range_vars*:

assumes "mgu s t = Some σ " shows "range_vars $\sigma \subseteq$ vars_term $s \cup$ vars_term t "

<proof>

lemma *mgu_subst_domain_range_vars_disjoint*:

assumes "mgu s t = Some σ " shows "subst_domain $\sigma \cap$ range_vars $\sigma = \{\}$ "

<proof>

lemma *mgu_same_empty*: "mgu (t::('a,'b) term) t = Some Var"

<proof>

lemma *mgu_var*: assumes " $x \notin$ fv t " shows "mgu (Var x) t = Some (Var($x := t$))"

<proof>

lemma *mgu_gives_wellformed_subst*:

assumes "mgu s t = Some ϑ " shows "wf_{subst} ϑ "

<proof>

lemma *mgu_gives_wellformed_MGU*:

assumes "mgu s t = Some ϑ " shows "wf_{MGU} ϑ s t "

<proof>

lemma *mgu_vars_bounded[dest?]*:

"mgu M N = Some $\sigma \implies$ subst_domain $\sigma \cup$ range_vars $\sigma \subseteq$ fv $M \cup$ fv N "

<proof>

lemma *mgu_gives_subst_idem*: "mgu s t = Some $\vartheta \implies$ subst_idem ϑ "

<proof>

lemma *mgu_always_unifies*: "Unifier ϑ M $N \implies \exists \delta. \text{mgu } M \ N = \text{Some } \delta$ "

<proof>

lemma *mgu_gives_MGU*: "mgu s t = Some $\vartheta \implies$ MGU ϑ s t "

<proof>

lemma *mgu_eliminate[dest?]*:

assumes "mgu M N = Some σ "

shows " $(\exists v \in$ fv $M \cup$ fv N . subst_elim σ v) \vee $\sigma =$ Var"

(is "P M N σ ")

<proof>

lemma *mgu_eliminate_dom*:

```

assumes "mgu x y = Some  $\vartheta$ " "v  $\in$  subst_domain  $\vartheta$ "
shows "subst_elim  $\vartheta$  v"
<proof>

lemma unify_list_distinct:
  assumes "Unification.unify E B = Some U" "distinct (map fst B)"
  and " $\bigcup x \in \text{set } E. \text{fv } (fst\ x) \cup \text{fv } (snd\ x) \cap \text{set } (\text{map } fst\ B) = \{\}$ "
  shows "distinct (map fst U)"
<proof>

lemma mgu_None_is_subst_neq:
  fixes s t::('a,'b) term and  $\delta$ ::('a,'b) subst"
  assumes "mgu s t = None"
  shows "s  $\cdot$   $\delta \neq$  t  $\cdot$   $\delta$ "
<proof>

lemma mgu_None_if_neq_ground:
  assumes "t  $\neq$  t'" "fv t =  $\{\}$ " "fv t' =  $\{\}$ "
  shows "mgu t t' = None"
<proof>

lemma mgu_None_commutates:
  "mgu s t = None  $\implies$  mgu t s = None"
<proof>

lemma mgu_img_subterm_subst:
  fixes  $\delta$ ::('f,'v) subst" and s t u::('f,'v) term"
  assumes "mgu s t = Some  $\delta$ " "u  $\in$  subtermsset (subst_range  $\delta$ ) - range Var"
  shows "u  $\in$  ((subterms s  $\cup$  subterms t) - range Var)  $\cdot$ set  $\delta$ "
<proof>

lemma mgu_img_consts:
  fixes  $\delta$ ::('f,'v) subst" and s t::('f,'v) term" and c::'f and z::'v
  assumes "mgu s t = Some  $\delta$ " "Fun c []  $\in$  subtermsset (subst_range  $\delta$ )"
  shows "Fun c []  $\in$  subterms s  $\cup$  subterms t"
<proof>

lemma mgu_img_consts':
  fixes  $\delta$ ::('f,'v) subst" and s t::('f,'v) term" and c::'f and z::'v
  assumes "mgu s t = Some  $\delta$ " " $\delta$  z = Fun c []"
  shows "Fun c []  $\sqsubseteq$  s  $\vee$  Fun c []  $\sqsubseteq$  t"
<proof>

lemma mgu_img_composed_var_term:
  fixes  $\delta$ ::('f,'v) subst" and s t::('f,'v) term" and f::'f and Z::"v list"
  assumes "mgu s t = Some  $\delta$ " "Fun f (map Var Z)  $\in$  subtermsset (subst_range  $\delta$ )"
  shows " $\exists Z'. \text{map } \delta\ Z' = \text{map } \text{Var } Z \wedge \text{Fun } f\ (\text{map } \text{Var } Z') \in \text{subterms } s \cup \text{subterms } t$ "
<proof>

```

2.3.9 Lemmata: The "Inequality Lemmata"

Subterm injectivity (a stronger injectivity property)

definition subterm_inj_on where

"subterm_inj_on f A $\equiv \forall x \in A. \forall y \in A. (\exists v. v \sqsubseteq f\ x \wedge v \sqsubseteq f\ y) \implies x = y$ "

lemma subterm_inj_on_imp_inj_on: "subterm_inj_on f A \implies inj_on f A"

<proof>

lemma subst_inj_on_is_bij_betw:

"inj_on ϑ (subst_domain ϑ) = bij_betw ϑ (subst_domain ϑ) (subst_range ϑ)"

<proof>

```

lemma subterm_inj_on_alt_def:
  "subterm_inj_on f A  $\longleftrightarrow$ 
  (inj_on f A  $\wedge$  ( $\forall s \in f \cdot A. \forall u \in f \cdot A. (\exists v. v \sqsubseteq s \wedge v \sqsubseteq u) \longrightarrow s = u$ ))"
  (is "?A  $\longleftrightarrow$  ?B")
<proof>

lemma subterm_inj_on_alt_def':
  "subterm_inj_on  $\vartheta$  (subst_domain  $\vartheta$ )  $\longleftrightarrow$ 
  (inj_on  $\vartheta$  (subst_domain  $\vartheta$ )  $\wedge$ 
  ( $\forall s \in \text{subst\_range } \vartheta. \forall u \in \text{subst\_range } \vartheta. (\exists v. v \sqsubseteq s \wedge v \sqsubseteq u) \longrightarrow s = u$ ))"
  (is "?A  $\longleftrightarrow$  ?B")
<proof>

lemma subterm_inj_on_subset:
  assumes "subterm_inj_on f A"
  and "B  $\subseteq$  A"
  shows "subterm_inj_on f B"
<proof>

lemma inj_subst_unif_consts:
  fixes  $\mathcal{I} \vartheta \sigma :: ('f, 'v) \text{ subst}$  and  $s \ t :: ('f, 'v) \text{ term}$ 
  assumes  $\vartheta$ : "subterm_inj_on  $\vartheta$  (subst_domain  $\vartheta$ )" " $\forall x \in (fv \ s \cup fv \ t) - X. \exists c. \vartheta \ x = \text{Fun } c \ []$ "
  "subtermsset (subst_range  $\vartheta$ )  $\cap$  (subterms s  $\cup$  subterms t) = {}" "ground (subst_range  $\vartheta$ )"
  "subst_domain  $\vartheta \cap X = \{\}$ "
  and  $\mathcal{I}$ : "ground (subst_range  $\mathcal{I}$ )" "subst_domain  $\mathcal{I} = \text{subst\_domain } \vartheta$ "
  and unif: "Unifier  $\sigma$  (s  $\cdot$   $\vartheta$ ) (t  $\cdot$   $\vartheta$ )"
  shows " $\exists \delta. \text{Unifier } \delta$  (s  $\cdot$   $\mathcal{I}$ ) (t  $\cdot$   $\mathcal{I}$ )"
<proof>

lemma inj_subst_unif_comp_terms:
  fixes  $\mathcal{I} \vartheta \sigma :: ('f, 'v) \text{ subst}$  and  $s \ t :: ('f, 'v) \text{ term}$ 
  assumes  $\vartheta$ : "subterm_inj_on  $\vartheta$  (subst_domain  $\vartheta$ )" "ground (subst_range  $\vartheta$ )"
  "subtermsset (subst_range  $\vartheta$ )  $\cap$  (subterms s  $\cup$  subterms t) = {}"
  " $(fv \ s \cup fv \ t) - \text{subst\_domain } \vartheta \subseteq X$ "
  and tfr: " $\forall f \ U. \text{Fun } f \ U \in \text{subterms } s \cup \text{subterms } t \longrightarrow U = [] \vee (\exists u \in \text{set } U. u \notin \text{Var } ` X)$ "
  and  $\mathcal{I}$ : "ground (subst_range  $\mathcal{I}$ )" "subst_domain  $\mathcal{I} = \text{subst\_domain } \vartheta$ "
  and unif: "Unifier  $\sigma$  (s  $\cdot$   $\vartheta$ ) (t  $\cdot$   $\vartheta$ )"
  shows " $\exists \delta. \text{Unifier } \delta$  (s  $\cdot$   $\mathcal{I}$ ) (t  $\cdot$   $\mathcal{I}$ )"
<proof>

context
begin
private lemma sat_ineq_subterm_inj_subst_aux:
  fixes  $\mathcal{I} :: ('f, 'v) \text{ subst}$ 
  assumes "Unifier  $\sigma$  (s  $\cdot$   $\mathcal{I}$ ) (t  $\cdot$   $\mathcal{I}$ )" "ground (subst_range  $\mathcal{I}$ )"
  " $(fv \ s \cup fv \ t) - X \subseteq \text{subst\_domain } \mathcal{I}$ " "subst_domain  $\mathcal{I} \cap X = \{\}$ "
  shows " $\exists \delta :: ('f, 'v) \text{ subst}. \text{subst\_domain } \delta = X \wedge \text{ground (subst\_range } \delta) \wedge s \cdot \delta \cdot \mathcal{I} = t \cdot \delta \cdot \mathcal{I}$ "
<proof>

The "inequality lemma": This lemma gives sufficient syntactic conditions for finding substitutions  $\vartheta$  under which terms  $s$  and  $t$  are not unifiable.

This is useful later when establishing the typing results since we there want to find well-typed solutions to inequality constraints / "negative checks" constraints, and this lemma gives conditions for protocols under which such constraints are well-typed satisfiable if satisfiable.

lemma sat_ineq_subterm_inj_subst:
  fixes  $\vartheta \mathcal{I} \delta :: ('f, 'v) \text{ subst}$ 
  assumes  $\vartheta$ : "subterm_inj_on  $\vartheta$  (subst_domain  $\vartheta$ )"
  "ground (subst_range  $\vartheta$ )"
  "subst_domain  $\vartheta \cap X = \{\}$ "
  "subtermsset (subst_range  $\vartheta$ )  $\cap$  (subterms s  $\cup$  subterms t) = {}"
  " $(fv \ s \cup fv \ t) - \text{subst\_domain } \vartheta \subseteq X$ "
  and tfr: " $(\forall x \in (fv \ s \cup fv \ t) - X. \exists c. \vartheta \ x = \text{Fun } c \ []) \vee$ 
  ( $\forall f \ U. \text{Fun } f \ U \in \text{subterms } s \cup \text{subterms } t \longrightarrow U = [] \vee (\exists u \in \text{set } U. u \notin \text{Var } ` X)$ )"

```

```

and I: "∀δ::('f,'v) subst. subst_domain δ = X ∧ ground (subst_range δ) → s · δ · I ≠ t · δ · I"
      "(fv s ∪ fv t) - X ⊆ subst_domain I" "subst_domain I ∩ X = {}" "ground (subst_range I)"
      "subst_domain I = subst_domain ∅"
and δ: "subst_domain δ = X" "ground (subst_range δ)"
shows "s · δ · ∅ ≠ t · δ · ∅"
⟨proof⟩
end

```

```

lemma ineq_subterm_inj_cond_subst:
  assumes "X ∩ range_vars ∅ = {}"
  and "∀f T. Fun f T ∈ subterms_set S → T = [] ∨ (∃u ∈ set T. u ∉ Var ` X)"
  shows "∀f T. Fun f T ∈ subterms_set (S ·_set ∅) → T = [] ∨ (∃u ∈ set T. u ∉ Var ` X)"
⟨proof⟩

```

2.3.10 Lemmata: Sufficient Conditions for Term Matching

Injective substitutions from variables to variables are invertible

definition *subst_var_inv* where

```
"subst_var_inv δ X ≡ (λx. if Var x ∈ δ ` X then Var ((inv_into X δ) (Var x)) else Var x)"
```

lemma *inj_var_ran_subst_is_invertible*:

```

assumes δ_inj_on_t: "inj_on δ (fv t)"
  and δ_var_on_t: "δ ` fv t ⊆ range Var"
shows "t = t · δ ∘_s subst_var_inv δ (fv t)"
⟨proof⟩

```

Sufficient conditions for matching unifiable terms

lemma *inj_var_ran_unifiable_has_subst_match*:

```

assumes "t · δ = s · δ" "inj_on δ (fv t)" "δ ` fv t ⊆ range Var"
shows "t = s · δ ∘_s subst_var_inv δ (fv t)"
⟨proof⟩

```

end

2.4 Dolev-Yao Intruder Model (Intruder_Deduction)

```

theory Intruder_Deduction
imports Messages More_Unification
begin

```

2.4.1 Syntax for the Intruder Deduction Relations

```

consts INTRUDER_SYNT::('f,'v) terms ⇒ ('f,'v) term ⇒ bool" (infix "⊢c" 50)
consts INTRUDER_DEDUCT::('f,'v) terms ⇒ ('f,'v) term ⇒ bool" (infix "⊢" 50)

```

2.4.2 Intruder Model Locale

The intruder model is parameterized over arbitrary function symbols (e.g, cryptographic operators) and variables. It requires three functions: - *arity* that assigns an arity to each function symbol. - *public* that partitions the function symbols into those that will be available to the intruder and those that will not. - *Ana*, the analysis interface, that defines how messages can be decomposed (e.g., decryption).

```

locale intruder_model =
  fixes arity :: "'fun ⇒ nat"
  and public :: "'fun ⇒ bool"
  and Ana :: "('fun,'var) term ⇒ (('fun,'var) term list × ('fun,'var) term list)"
  assumes Ana_keys_fv: "∧t K R. Ana t = (K,R) ⇒ fv_set (set K) ⊆ fv t"
  and Ana_keys_wf: "∧t k K R f T.
    Ana t = (K,R) ⇒ (∧g S. Fun g S ⊆ t ⇒ length S = arity g)
    ⇒ k ∈ set K ⇒ Fun f T ⊆ k ⇒ length T = arity f"
  and Ana_var[simp]: "∧x. Ana (Var x) = ([], [])"
  and Ana_fun_subterm: "∧f T K R. Ana (Fun f T) = (K,R) ⇒ set R ⊆ set T"

```

and Ana_subst: " $\bigwedge t \delta K R. \llbracket \text{Ana } t = (K,R); K \neq [] \vee R \neq [] \rrbracket \implies \text{Ana } (t \cdot \delta) = (K \cdot_{list} \delta, R \cdot_{list} \delta)$ "
begin

lemma Ana_subterm: assumes "Ana t = (K,T)" shows "set T \subseteq subterms t"
<proof>

lemma Ana_subterm': "s \in set (snd (Ana t)) \implies s \sqsubseteq t"
<proof>

lemma Ana_vars: assumes "Ana t = (K,M)" shows "fv_{set} (set K) \subseteq fv t" "fv_{set} (set M) \subseteq fv t"
<proof>

abbreviation \mathcal{V} where " $\mathcal{V} \equiv UNIV::\text{'var set}$ "

abbreviation Σ_n (" $\Sigma-$ ") where " $\Sigma^n \equiv \{f::\text{'fun. arity } f = n\}$ "

abbreviation Σ_{pub} (" $\Sigma_{pub}-$ ") where " $\Sigma_{pub}^n \equiv \{f. \text{public } f\} \cap \Sigma^n$ "

abbreviation Σ_{priv} (" $\Sigma_{priv}-$ ") where " $\Sigma_{priv}^n \equiv \{f. \neg \text{public } f\} \cap \Sigma^n$ "

abbreviation Σ_{pub} where " $\Sigma_{pub} \equiv (\bigcup n. \Sigma_{pub}^n)$ "

abbreviation Σ_{priv} where " $\Sigma_{priv} \equiv (\bigcup n. \Sigma_{priv}^n)$ "

abbreviation Σ where " $\Sigma \equiv (\bigcup n. \Sigma^n)$ "

abbreviation \mathcal{C} where " $\mathcal{C} \equiv \Sigma^0$ "

abbreviation \mathcal{C}_{pub} where " $\mathcal{C}_{pub} \equiv \{f. \text{public } f\} \cap \mathcal{C}$ "

abbreviation \mathcal{C}_{priv} where " $\mathcal{C}_{priv} \equiv \{f. \neg \text{public } f\} \cap \mathcal{C}$ "

abbreviation Σ_f where " $\Sigma_f \equiv \Sigma - \mathcal{C}$ "

abbreviation Σ_{fpub} where " $\Sigma_{fpub} \equiv \Sigma_f \cap \Sigma_{pub}$ "

abbreviation Σ_{fpriv} where " $\Sigma_{fpriv} \equiv \Sigma_f \cap \Sigma_{priv}$ "

lemma disjoint_fun_syms: " $\Sigma_f \cap \mathcal{C} = \{\}$ " <proof>

lemma id_union_univ: " $\Sigma_f \cup \mathcal{C} = UNIV$ " " $\Sigma = UNIV$ " <proof>

lemma const_arity_eq_zero[dest]: "c \in $\mathcal{C} \implies$ arity c = 0" <proof>

lemma const_pub_arity_eq_zero[dest]: "c \in $\mathcal{C}_{pub} \implies$ arity c = 0 \wedge public c" <proof>

lemma const_priv_arity_eq_zero[dest]: "c \in $\mathcal{C}_{priv} \implies$ arity c = 0 \wedge \neg public c" <proof>

lemma fun_arity_gt_zero[dest]: "f \in $\Sigma_f \implies$ arity f > 0" <proof>

lemma pub_fun_public[dest]: "f \in $\Sigma_{fpub} \implies$ public f" <proof>

lemma pub_fun_arity_gt_zero[dest]: "f \in $\Sigma_{fpub} \implies$ arity f > 0" <proof>

lemma Σ_f _unfold: " $\Sigma_f = \{f::\text{'fun. arity } f > 0\}$ " <proof>

lemma \mathcal{C} _unfold: " $\mathcal{C} = \{f::\text{'fun. arity } f = 0\}$ " <proof>

lemma \mathcal{C}_{pub} _unfold: " $\mathcal{C}_{pub} = \{f::\text{'fun. arity } f = 0 \wedge \text{public } f\}$ " <proof>

lemma \mathcal{C}_{priv} _unfold: " $\mathcal{C}_{priv} = \{f::\text{'fun. arity } f = 0 \wedge \neg \text{public } f\}$ " <proof>

lemma Σ_{pub} _unfold: " $(\Sigma_{pub}^n) = \{f::\text{'fun. arity } f = n \wedge \text{public } f\}$ " <proof>

lemma Σ_{priv} _unfold: " $(\Sigma_{priv}^n) = \{f::\text{'fun. arity } f = n \wedge \neg \text{public } f\}$ " <proof>

lemma Σ_{fpub} _unfold: " $\Sigma_{fpub} = \{f::\text{'fun. arity } f > 0 \wedge \text{public } f\}$ " <proof>

lemma Σ_{fpriv} _unfold: " $\Sigma_{fpriv} = \{f::\text{'fun. arity } f > 0 \wedge \neg \text{public } f\}$ " <proof>

lemma Σ_n _m_eq: " $\llbracket (\Sigma^n) \neq \{\}; (\Sigma^n) = (\Sigma^m) \rrbracket \implies n = m$ " <proof>

2.4.3 Term Well-formedness

definition "wf_{trm} t \equiv $\forall f T. \text{Fun } f T \sqsubseteq t \longrightarrow \text{length } T = \text{arity } f$ "

abbreviation "wf_{trms} T \equiv $\forall t \in T. \text{wf}_{trm} t$ "

lemma Ana_keys_wf': "Ana t = (K,T) \implies wf_{trm} t \implies k \in set K \implies wf_{trm} k"
<proof>

lemma wf_trm_Var[simp]: "wf_{trm} (Var x)" <proof>

lemma wf_trm_subst_range_Var[simp]: "wf_{trms} (subst_range Var)" <proof>

lemma wf_trm_subst_range_iff: " $(\forall x. \text{wf}_{trm} (\vartheta x)) \longleftrightarrow \text{wf}_{trms} (\text{subst_range } \vartheta)$ "
<proof>

lemma wf_trm_subst_ranged: "wf_{trms} (subst_range ϑ) \implies wf_{trm} (ϑx)"
<proof>


```

lemma wf_trm_subst_rangeI[intro]:
  "(\x. wf_trm (\delta x)) \implies wf_trms (subst_range \delta)"
<proof>

lemma wf_trmI[intro]:
  assumes "\t. t \in set T \implies wf_trm t" "length T = arity f"
  shows "wf_trm (Fun f T)"
<proof>

lemma wf_trm_subterm: "[wf_trm t; s \sqsubseteq t] \implies wf_trm s"
<proof>

lemma wf_trm_subtermeq:
  assumes "wf_trm t" "s \sqsubseteq t"
  shows "wf_trm s"
<proof>

lemma wf_trm_param:
  assumes "wf_trm (Fun f T)" "t \in set T"
  shows "wf_trm t"
<proof>

lemma wf_trm_param_idx:
  assumes "wf_trm (Fun f T)"
  and "i < length T"
  shows "wf_trm (T ! i)"
<proof>

lemma wf_trm_subst:
  assumes "wf_trms (subst_range \delta)"
  shows "wf_trm t = wf_trm (t \cdot \delta)"
<proof>

lemma wf_trm_subst_singleton:
  assumes "wf_trm t" "wf_trm t'" shows "wf_trm (t \cdot Var(v := t'))"
<proof>

lemma wf_trm_subst_rm_vars:
  assumes "wf_trm (t \cdot \delta)"
  shows "wf_trm (t \cdot rm_vars X \delta)"
<proof>

lemma wf_trm_subst_rm_vars': "wf_trm (\delta v) \implies wf_trm (rm_vars X \delta v)"
<proof>

lemma wf_trms_subst:
  assumes "wf_trms (subst_range \delta)" "wf_trms M"
  shows "wf_trms (M \cdot_{set} \delta)"
<proof>

lemma wf_trms_subst_rm_vars:
  assumes "wf_trms (M \cdot_{set} \delta)"
  shows "wf_trms (M \cdot_{set} rm_vars X \delta)"
<proof>

lemma wf_trms_subst_rm_vars':
  assumes "wf_trms (subst_range \delta)"
  shows "wf_trms (subst_range (rm_vars X \delta))"
<proof>

lemma wf_trms_subst_compose:
  assumes "wf_trms (subst_range \vartheta)" "wf_trms (subst_range \delta)"

```

shows "wf_{trms} (subst_range (ϑ ∘_s δ))"
 ⟨proof⟩

lemma wf_trm_subst_compose:
 fixes δ::('fun, 'v) subst"
 assumes "wf_{trm} (ϑ x)" "∧x. wf_{trm} (δ x)"
 shows "wf_{trm} ((ϑ ∘_s δ) x)"
 ⟨proof⟩

lemma wf_trms_Var_range:
 assumes "subst_range δ ⊆ range Var"
 shows "wf_{trms} (subst_range δ)"
 ⟨proof⟩

lemma wf_trms_subst_compose_Var_range:
 assumes "wf_{trms} (subst_range ϑ)"
 and "subst_range δ ⊆ range Var"
 shows "wf_{trms} (subst_range (δ ∘_s ϑ))"
 and "wf_{trms} (subst_range (ϑ ∘_s δ))"
 ⟨proof⟩

lemma wf_trm_subst_inv: "wf_{trm} (t · δ) ⇒ wf_{trm} t"
 ⟨proof⟩

lemma wf_trms_subst_inv: "wf_{trms} (M ·_{set} δ) ⇒ wf_{trms} M"
 ⟨proof⟩

lemma wf_trm_subterms: "wf_{trm} t ⇒ wf_{trms} (subterms t)"
 ⟨proof⟩

lemma wf_trms_subterms: "wf_{trms} M ⇒ wf_{trms} (subterms_{set} M)"
 ⟨proof⟩

lemma wf_trm_arity: "wf_{trm} (Fun f T) ⇒ length T = arity f"
 ⟨proof⟩

lemma wf_trm_subterm_arity: "wf_{trm} t ⇒ Fun f T ⊆ t ⇒ length T = arity f"
 ⟨proof⟩

lemma unify_list_wf_trm:
 assumes "Unification.unify E B = Some U" "∀(s,t) ∈ set E. wf_{trm} s ∧ wf_{trm} t"
 and "∀(v,t) ∈ set B. wf_{trm} t"
 shows "∀(v,t) ∈ set U. wf_{trm} t"
 ⟨proof⟩

lemma mgu_wf_trm:
 assumes "mgu s t = Some σ" "wf_{trm} s" "wf_{trm} t"
 shows "wf_{trm} (σ v)"
 ⟨proof⟩

lemma mgu_wf_trms:
 assumes "mgu s t = Some σ" "wf_{trm} s" "wf_{trm} t"
 shows "wf_{trms} (subst_range σ)"
 ⟨proof⟩

2.4.4 Definitions: Intruder Deduction Relations

A standard Dolev-Yao intruder.

inductive intruder_deduct::('fun,'var) terms ⇒ ('fun,'var) term ⇒ bool"

where

Axiom[simp]: "t ∈ M ⇒ intruder_deduct M t"
 | Compose[simp]: "[length T = arity f; public f; ∧t. t ∈ set T ⇒ intruder_deduct M t]"

```

    ⇒ intruder_deduct M (Fun f T)"
| Decompose: "⟦intruder_deduct M t; Ana t = (K, T); ∧k. k ∈ set K ⇒ intruder_deduct M k;
              ti ∈ set T⟧
              ⇒ intruder_deduct M ti"

```

A variant of the intruder relation which limits the intruder to composition only.

```

inductive intruder_synth: "('fun, 'var) terms ⇒ ('fun, 'var) term ⇒ bool"
where
  AxiomC[simp]: "t ∈ M ⇒ intruder_synth M t"
| ComposeC[simp]: "⟦length T = arity f; public f; ∧t. t ∈ set T ⇒ intruder_synth M t⟧
                  ⇒ intruder_synth M (Fun f T)"

```

```

adhoc_overloading INTRUDER_DEDUCT intruder_deduct
adhoc_overloading INTRUDER_SYNTX intruder_synth

```

```

lemma intruder_deduct_induct[consumes 1, case_names Axiom Compose Decompose]:
  assumes "M ⊢ t" "∧t. t ∈ M ⇒ P M t"
          "∧T f. ⟦length T = arity f; public f;
                ∧t. t ∈ set T ⇒ M ⊢ t;
                ∧t. t ∈ set T ⇒ P M t⟧ ⇒ P M (Fun f T)"
          "∧t K T ti. ⟦M ⊢ t; P M t; Ana t = (K, T); ∧k. k ∈ set K ⇒ M ⊢ k;
                ∧k. k ∈ set K ⇒ P M k; ti ∈ set T⟧ ⇒ P M ti"
  shows "P M t"
  <proof>

```

```

lemma intruder_synth_induct[consumes 1, case_names AxiomC ComposeC]:
  fixes M: "('fun, 'var) terms" and t: "('fun, 'var) term"
  assumes "M ⊢c t" "∧t. t ∈ M ⇒ P M t"
          "∧T f. ⟦length T = arity f; public f;
                ∧t. t ∈ set T ⇒ M ⊢c t;
                ∧t. t ∈ set T ⇒ P M t⟧ ⇒ P M (Fun f T)"
  shows "P M t"
  <proof>

```

2.4.5 Definitions: Analyzed Knowledge and Public Ground Well-formed Terms (PGWTs)

```

definition analyzed: "('fun, 'var) terms ⇒ bool" where
  "analyzed M ≡ ∀t. M ⊢ t ↔ M ⊢c t"

```

```

definition analyzed_in where

```

```

  "analyzed_in t M ≡ ∀K R. (Ana t = (K, R) ∧ (∀k ∈ set K. M ⊢c k)) → (∀r ∈ set R. M ⊢c r)"

```

```

definition decomp_closure: "('fun, 'var) terms ⇒ ('fun, 'var) terms ⇒ bool" where
  "decomp_closure M M' ≡ ∀t. M ⊢ t ∧ (∃t' ∈ M. t ⊆ t') ↔ t ∈ M'"

```

```

inductive public_ground_wf_term: "('fun, 'var) term ⇒ bool" where

```

```

  PGWT[simp]: "⟦public f; arity f = length T;
                ∧t. t ∈ set T ⇒ public_ground_wf_term t⟧
                ⇒ public_ground_wf_term (Fun f T)"

```

```

abbreviation "public_ground_wf_terms ≡ {t. public_ground_wf_term t}"

```

```

lemma public_const_deduct:
  assumes "c ∈ Cpub"
  shows "M ⊢ Fun c []" "M ⊢c Fun c []"
  <proof>

```

```

lemma public_const_deduct'[simp]:
  assumes "arity c = 0" "public c"
  shows "M ⊢ Fun c []" "M ⊢c Fun c []"
  <proof>

```

```

lemma private_fun_deduct_in_ik:

```

```

    assumes t: "M ⊢ t" "Fun f T ∈ subterms t"
    and f: "¬public f"
    shows "Fun f T ∈ subtermsset M"
⟨proof⟩

lemma private_fun_deduct_in_ik':
  assumes t: "M ⊢ Fun f T"
  and f: "¬public f"
  and M: "Fun f T ∈ subtermsset M ⇒ Fun f T ∈ M"
  shows "Fun f T ∈ M"
⟨proof⟩

lemma pgwt_public: "[public_ground_wf_term t; Fun f T ⊆ t] ⇒ public f"
⟨proof⟩

lemma pgwt_ground: "public_ground_wf_term t ⇒ fv t = {}"
⟨proof⟩

lemma pgwt_fun: "public_ground_wf_term t ⇒ ∃ f T. t = Fun f T"
⟨proof⟩

lemma pgwt_arity: "[public_ground_wf_term t; Fun f T ⊆ t] ⇒ arity f = length T"
⟨proof⟩

lemma pgwt_wellformed: "public_ground_wf_term t ⇒ wftrm t"
⟨proof⟩

lemma pgwt_deducible: "public_ground_wf_term t ⇒ M ⊢c t"
⟨proof⟩

lemma pgwt_is_empty_synth: "public_ground_wf_term t ⇔ {} ⊢c t"
⟨proof⟩

lemma ideduct_synth_subst_apply:
  fixes M:: "('fun, 'var) terms" and t:: "('fun, 'var) term"
  assumes "{} ⊢c t" "∧v. M ⊢c v"
  shows "M ⊢c t · v"
⟨proof⟩

```

2.4.6 Lemmata: Monotonicity, deduction private constants, etc.

```

context
begin
lemma ideduct_mono:
  "[M ⊢ t; M ⊆ M'] ⇒ M' ⊢ t"
⟨proof⟩

lemma ideduct_synth_mono:
  fixes M:: "('fun, 'var) terms" and t:: "('fun, 'var) term"
  shows "[M ⊢c t; M ⊆ M'] ⇒ M' ⊢c t"
⟨proof⟩

lemma ideduct_reduce:
  "[M ∪ M' ⊢ t; ∧t'. t' ∈ M' ⇒ M ⊢ t'] ⇒ M ⊢ t"
⟨proof⟩

lemma ideduct_synth_reduce:
  fixes M:: "('fun, 'var) terms" and t:: "('fun, 'var) term"
  shows "[M ∪ M' ⊢c t; ∧t'. t' ∈ M' ⇒ M ⊢c t'] ⇒ M ⊢c t"
⟨proof⟩

lemma ideduct_mono_eq:
  assumes "∀t. M ⊢ t ⇔ M' ⊢ t" shows "M ∪ N ⊢ t ⇔ M' ∪ N ⊢ t"

```

<proof>

lemma *deduct_synth_subterm*:

fixes $M::('fun, 'var) \text{ terms}$ and $t::('fun, 'var) \text{ term}$
 assumes " $M \vdash_c t$ " " $s \in \text{subterms } t$ " " $\forall m \in M. \forall s \in \text{subterms } m. M \vdash_c s$ "
 shows " $M \vdash_c s$ "

<proof>

lemma *deduct_if_synth*[intro, dest]: " $M \vdash_c t \implies M \vdash t$ "

<proof> **lemma** *ideduct_ik_eq*: assumes " $\forall t \in M. M' \vdash t$ " shows " $M' \vdash t \iff M' \cup M \vdash t$ "

<proof> **lemma** *synth_if_deduct_empty*: " $\{\} \vdash t \implies \{\} \vdash_c t$ "

<proof> **lemma** *ideduct_deduct_synth_mono_eq*:

assumes " $\forall t. M \vdash t \iff M' \vdash_c t$ " " $M \subseteq M'$ "
 and " $\forall t. M' \cup N \vdash t \iff M' \cup N \cup D \vdash_c t$ "
 shows " $M \cup N \vdash t \iff M' \cup N \cup D \vdash_c t$ "

<proof>

lemma *ideduct_subst*: " $M \vdash t \implies M \cdot_{\text{set}} \delta \vdash t \cdot \delta$ "

<proof>

lemma *ideduct_synth_subst*:

fixes $M::('fun, 'var) \text{ terms}$ and $t::('fun, 'var) \text{ term}$ and $\delta::('fun, 'var) \text{ subst}$
 shows " $M \vdash_c t \implies M \cdot_{\text{set}} \delta \vdash_c t \cdot \delta$ "

<proof>

lemma *ideduct_vars*:

assumes " $M \vdash t$ "
 shows " $\text{fv } t \subseteq \text{fv}_{\text{set}} M$ "

<proof>

lemma *ideduct_synth_vars*:

fixes $M::('fun, 'var) \text{ terms}$ and $t::('fun, 'var) \text{ term}$
 assumes " $M \vdash_c t$ "
 shows " $\text{fv } t \subseteq \text{fv}_{\text{set}} M$ "

<proof>

lemma *ideduct_synth_priv_fun_in_ik*:

fixes $M::('fun, 'var) \text{ terms}$ and $t::('fun, 'var) \text{ term}$
 assumes " $M \vdash_c t$ " " $f \in \text{funs_term } t$ " " $\neg \text{public } f$ "
 shows " $f \in \bigcup (\text{funs_term } ` M)$ "

<proof>

lemma *ideduct_synth_priv_const_in_ik*:

fixes $M::('fun, 'var) \text{ terms}$ and $t::('fun, 'var) \text{ term}$
 assumes " $M \vdash_c \text{Fun } c []$ " " $\neg \text{public } c$ "
 shows " $\text{Fun } c [] \in M$ "

<proof>

lemma *ideduct_synth_ik_replace*:

fixes $M::('fun, 'var) \text{ terms}$ and $t::('fun, 'var) \text{ term}$
 assumes " $\forall t \in M. N \vdash_c t$ "
 and " $M \vdash_c t$ "
 shows " $N \vdash_c t$ "

<proof>

end

2.4.7 Lemmata: Analyzed Intruder Knowledge Closure

lemma *deducts_eq_if_analyzed*: " $\text{analyzed } M \implies M \vdash t \iff M \vdash_c t$ "

<proof>

lemma *closure_is_superset*: " $\text{decomp_closure } M M' \implies M \subseteq M'$ "

<proof>

```
lemma deduct_if_closure_deduct: "[[M' ⊢ t; decomp_closure M M']] ⇒ M ⊢ t"
⟨proof⟩
```

```
lemma deduct_if_closure_synt: "[[decomp_closure M M'; M' ⊢c t]] ⇒ M ⊢ t"
⟨proof⟩
```

```
lemma decomp_closure_subterms_composable:
  assumes "decomp_closure M M'"
  and "M' ⊢c t'" "M' ⊢ t" "t ⊆ t'"
  shows "M' ⊢c t"
⟨proof⟩
```

```
lemma decomp_closure_analyzed:
  assumes "decomp_closure M M'"
  shows "analyzed M'"
⟨proof⟩
```

```
lemma analyzed_if_all_analyzed_in:
  assumes M: "∀t ∈ M. analyzed_in t M"
  shows "analyzed M"
⟨proof⟩
```

```
lemma analyzed_is_all_analyzed_in:
  "(∀t ∈ M. analyzed_in t M) ⇔ analyzed M"
⟨proof⟩
```

```
lemma ik_has_synt_ik_closure:
  fixes M :: "('fun, 'var) terms"
  shows "∃M'. (∀t. M ⊢ t ⇔ M' ⊢c t) ∧ decomp_closure M M' ∧ (finite M → finite M')"
⟨proof⟩
```

2.4.8 Intruder Variants: Numbered and Composition-Restricted Intruder Deduction Relations

A variant of the intruder relation which restricts composition to only those terms that satisfy a given predicate Q .

```
inductive intruder_deduct_restricted::
  "('fun, 'var) terms ⇒ ((('fun, 'var) term ⇒ bool) ⇒ ('fun, 'var) term ⇒ bool)"
  ("⟨_; _⟩ ⊢r _" 50)
where
  AxiomR[simp]: "t ∈ M ⇒ ⟨M; Q⟩ ⊢r t"
  | ComposeR[simp]: "[[length T = arity f; public f; ∧t. t ∈ set T ⇒ ⟨M; Q⟩ ⊢r t; Q (Fun f T)]]
    ⇒ ⟨M; Q⟩ ⊢r Fun f T"
  | DecomposeR: "[[⟨M; Q⟩ ⊢r t; Ana t = (K, T); ∧k. k ∈ set K ⇒ ⟨M; Q⟩ ⊢r k; ti ∈ set T]]
    ⇒ ⟨M; Q⟩ ⊢r ti"
```

A variant of the intruder relation equipped with a number representing the height of the derivation tree (i.e., $\langle M; k \rangle \vdash_n t$ iff k is the maximum number of applications of the compose and decompose rules in any path of the derivation tree for $M \vdash t$).

```
inductive intruder_deduct_num::
  "('fun, 'var) terms ⇒ nat ⇒ ('fun, 'var) term ⇒ bool"
  ("⟨_; _⟩ ⊢n _" 50)
where
  AxiomN[simp]: "t ∈ M ⇒ ⟨M; 0⟩ ⊢n t"
  | ComposeN[simp]: "[[length T = arity f; public f; ∧t. t ∈ set T ⇒ ⟨M; steps t⟩ ⊢n t]]
    ⇒ ⟨M; Suc (Max (insert 0 (steps ` set T)))⟩ ⊢n Fun f T"
  | DecomposeN: "[[⟨M; n⟩ ⊢n t; Ana t = (K, T); ∧k. k ∈ set K ⇒ ⟨M; steps k⟩ ⊢n k; ti ∈ set T]]
    ⇒ ⟨M; Suc (Max (insert n (steps ` set K)))⟩ ⊢n ti"
```

```
lemma intruder_deduct_restricted_induct[consumes 1, case_names AxiomR ComposeR DecomposeR]:
  assumes "⟨M; Q⟩ ⊢r t" "∧t. t ∈ M ⇒ P M Q t"
```

```

"∧T f. [[length T = arity f; public f;
  ∧t. t ∈ set T ⇒ ⟨M; Q⟩ ⊢r t;
  ∧t. t ∈ set T ⇒ P M Q t; Q (Fun f T)
]] ⇒ P M Q (Fun f T)"
"∧t K T ti. [[⟨M; Q⟩ ⊢r t; P M Q t; Ana t = (K, T); ∧k. k ∈ set K ⇒ ⟨M; Q⟩ ⊢r k;
  ∧k. k ∈ set K ⇒ P M Q k; ti ∈ set T]] ⇒ P M Q ti"

```

shows "P M Q t"

⟨proof⟩

lemma intruder_deduct_num_induct[consumes 1, case_names AxiomN ComposeN DecomposeN]:

assumes "⟨M; n⟩ ⊢_n t" "∧t. t ∈ M ⇒ P M 0 t"

"∧T f steps.

```

[[length T = arity f; public f;
  ∧t. t ∈ set T ⇒ ⟨M; steps t⟩ ⊢n t;
  ∧t. t ∈ set T ⇒ P M (steps t) t]]
⇒ P M (Suc (Max (insert 0 (steps ` set T)))) (Fun f T)"

```

"∧t K T t_i steps n.

```

[[⟨M; n⟩ ⊢n t; P M n t; Ana t = (K, T);
  ∧k. k ∈ set K ⇒ ⟨M; steps k⟩ ⊢n k;
  ti ∈ set T; ∧k. k ∈ set K ⇒ P M (steps k) k]]
⇒ P M (Suc (Max (insert n (steps ` set K)))) ti"

```

shows "P M n t"

⟨proof⟩

lemma ideduct_restricted_mono:

"[[⟨M; P⟩ ⊢_r t; M ⊆ M'] ⇒ ⟨M'; P⟩ ⊢_r t"

⟨proof⟩

2.4.9 Lemmata: Intruder Deduction Equivalences

lemma deduct_if_restricted_deduct: "⟨M;P⟩ ⊢_r m ⇒ M ⊢ m"

⟨proof⟩

lemma restricted_deduct_if_restricted_ik:

assumes "⟨M;P⟩ ⊢_r m" "∀m ∈ M. P m"

and P: "∀t t'. P t → t' ⊆ t → P t'"

shows "P m"

⟨proof⟩

lemma deduct_restricted_if_synth:

assumes P: "P m" "∀t t'. P t → t' ⊆ t → P t'"

and m: "M ⊢_c m"

shows "⟨M; P⟩ ⊢_r m"

⟨proof⟩

lemma deduct_zero_in_ik:

assumes "⟨M; 0⟩ ⊢_n t" shows "t ∈ M"

⟨proof⟩

lemma deduct_if_deduct_num: "⟨M; k⟩ ⊢_n t ⇒ M ⊢ t"

⟨proof⟩

lemma deduct_num_if_deduct: "M ⊢ t ⇒ ∃k. ⟨M; k⟩ ⊢_n t"

⟨proof⟩

lemma deduct_normalize:

assumes M: "∀m ∈ M. ∀f T. Fun f T ⊆ m → P f T"

and t: "⟨M; k⟩ ⊢_n t" "Fun f T ⊆ t" "¬P f T"

shows "∃l ≤ k. (⟨M; l⟩ ⊢_n Fun f T) ∧ (∀t ∈ set T. ∃j < l. ⟨M; j⟩ ⊢_n t)"

⟨proof⟩

lemma deduct_inv:

assumes "⟨M; n⟩ ⊢_n t"

```

shows "t ∈ M ∨
  (∃ f T. t = Fun f T ∧ public f ∧ length T = arity f ∧ (∀ t ∈ set T. ∃ l < n. ⟨M; l⟩ ⊢n t)) ∨
  (∃ m ∈ subtermsset M.
    (∃ l < n. ⟨M; l⟩ ⊢n m) ∧ (∀ k ∈ set (fst (Ana m)). ∃ l < n. ⟨M; l⟩ ⊢n k) ∧
    t ∈ set (snd (Ana m)))"
(is "?P t n ∨ ?Q t n ∨ ?R t n")
⟨proof⟩

```

```

lemma restricted_deduct_if_deduct:
  assumes M: "∀ m ∈ M. ∀ f T. Fun f T ⊆ m → P (Fun f T)"
  and P_subterm: "∀ f T t. M ⊢ Fun f T → P (Fun f T) → t ∈ set T → P t"
  and P_Ana_key: "∀ t K T k. M ⊢ t → P t → Ana t = (K, T) → M ⊢ k → k ∈ set K → P k"
  and m: "M ⊢ m" "P m"
  shows "⟨M; P⟩ ⊢r m"
⟨proof⟩

```

```

lemma restricted_deduct_if_deduct':
  assumes "∀ m ∈ M. P m"
  and "∀ t t'. P t → t' ⊆ t → P t'"
  and "∀ t K T k. P t → Ana t = (K, T) → k ∈ set K → P k"
  and "M ⊢ m" "P m"
  shows "⟨M; P⟩ ⊢r m"
⟨proof⟩

```

```

lemma private_const_deduct:
  assumes c: "¬public c" "M ⊢ (Fun c []::('fun,'var) term)"
  shows "Fun c [] ∈ M ∨
    (∃ m ∈ subtermsset M. M ⊢ m ∧ (∀ k ∈ set (fst (Ana m)). M ⊢ m) ∧
    Fun c [] ∈ set (snd (Ana m)))"
⟨proof⟩

```

```

lemma private_fun_deduct_in_ik'':
  assumes t: "M ⊢ Fun f T" "Fun c [] ∈ set T" "∀ m ∈ subtermsset M. Fun f T ∉ set (snd (Ana m))"
  and c: "¬public c" "Fun c [] ∉ M" "∀ m ∈ subtermsset M. Fun c [] ∉ set (snd (Ana m))"
  shows "Fun f T ∈ M"
⟨proof⟩

```

end

2.4.10 Executable Definitions for Code Generation

```

fun intruder_synth' where
  "intruder_synth' pu ar M (Var x) = (Var x ∈ M)"
| "intruder_synth' pu ar M (Fun f T) = (
  Fun f T ∈ M ∨ (pu f ∧ length T = ar f ∧ list_all (intruder_synth' pu ar M) T))"

```

```

definition "wftrm' ar t ≡ (∀ s ∈ subterms t. is_Fun s → ar (the_Fun s) = length (args s))"

```

```

definition "wftrms' ar M ≡ (∀ t ∈ M. wftrm' ar t)"

```

```

definition "analyzed_in' An pu ar t M ≡ (case An t of
  (K,T) ⇒ (∀ k ∈ set K. intruder_synth' pu ar M k) → (∀ s ∈ set T. intruder_synth' pu ar M s))"

```

```

lemma (in intruder_model) intruder_synth'_induct[consumes 1, case_names Var Fun]:
  assumes "intruder_synth' public arity M t"
  "∧x. intruder_synth' public arity M (Var x) ⇒ P (Var x)"
  "∧f T. (∧z. z ∈ set T ⇒ intruder_synth' public arity M z ⇒ P z) ⇒
  intruder_synth' public arity M (Fun f T) ⇒ P (Fun f T) "
  shows "P t"
⟨proof⟩

```

```

lemma (in intruder_model) wftrm_code[code_unfold]:
  "wftrm t = wftrm' arity t"

```


<proof>

lemma (in intruder_model) wf_trms_code[code_unfold]:

"wf_trms M = wf_trms' arity M"

<proof>

lemma (in intruder_model) intruder_synth_code[code_unfold]:

"intruder_synth M t = intruder_synth' public arity M t"

(is "?A \longleftrightarrow ?B")

<proof>

lemma (in intruder_model) analyzed_in_code[code_unfold]:

"analyzed_in t M = analyzed_in' Ana public arity t M"

<proof>

end

3 The Typing Result for Non-Stateful Protocols

In this chapter, we formalize and prove a typing result for “stateless” security protocols. This work is described in more detail in [2] and [1, chapter 3].

3.1 Strands and Symbolic Intruder Constraints (Strands_and_Constraints)

```
theory Strands_and_Constraints
imports Messages More_Unification Intruder_Deduction
begin
```

3.1.1 Constraints, Strands and Related Definitions

```
datatype poscheckvariant = Assign ("assign") | Check ("check")
```

A strand (or constraint) step is either a message transmission (either a message being sent *Send* or being received *Receive*) or a check on messages (a positive check *Equality*—which can be either an “assignment” or just a check—or a negative check *Inequality*)

```
datatype (funsstp: 'a, varsstp: 'b) strand_step =
  Send      "('a,'b) term" ("send⟨_⟩st" 80)
| Receive   "('a,'b) term" ("receive⟨_⟩st" 80)
| Equality  poscheckvariant "('a,'b) term" "('a,'b) term" ("⟨_⟩: ⟨_⟩ ≐ ⟨_⟩st" [80,80])
| Inequality (bvarsstp: "'b list") ("⟨('a,'b) term × ('a,'b) term⟩ list" ("∀_⟨_⟩≠: ⟨_⟩st" [80,80]))
where
  "bvarsstp (Send _) = []"
| "bvarsstp (Receive _) = []"
| "bvarsstp (Equality _ _ _) = []"
```

A strand is a finite sequence of strand steps (constraints and strands share the same datatype)

```
type_synonym ('a,'b) strand = "('a,'b) strand_step list"
```

```
type_synonym ('a,'b) strands = "('a,'b) strand set"
```

```
abbreviation "trmspairs F ≡ ⋃ (t,t') ∈ set F. {t,t'}"
```

```
fun trmsstp :: "('a,'b) strand_step ⇒ ('a,'b) terms" where
  "trmsstp (Send t) = {t}"
| "trmsstp (Receive t) = {t}"
| "trmsstp (Equality _ t t') = {t,t'}"
| "trmsstp (Inequality _ F) = trmspairs F"
```

```
lemma varsstp_unfold[simp]: "varsstp x = fvset (trmsstp x) ∪ set (bvarsstp x)"
⟨proof⟩
```

The set of terms occurring in a strand

```
definition trmsst where "trmsst S ≡ ⋃ (trmsstp ` set S)"
```

```
fun trms_liststp :: "('a,'b) strand_step ⇒ ('a,'b) term list" where
  "trms_liststp (Send t) = [t]"
| "trms_liststp (Receive t) = [t]"
| "trms_liststp (Equality _ t t') = [t,t']"
| "trms_liststp (Inequality _ F) = concat (map (λ(t,t'). [t,t']) F)"
```

The set of terms occurring in a strand (list variant)

3 The Typing Result for Non-Stateful Protocols

definition $trms_list_{st}$ **where** $"trms_list_{st} S \equiv remdups (concat (map trms_list_{stp} S))"$

The set of variables occurring in a sent message

definition $fv_{snd}::('a,'b) strand_step \Rightarrow 'b set$ **where**
 $"fv_{snd} x \equiv case x of Send t \Rightarrow fv t \mid _ \Rightarrow \{\}"$

The set of variables occurring in a received message

definition $fv_{rcv}::('a,'b) strand_step \Rightarrow 'b set$ **where**
 $"fv_{rcv} x \equiv case x of Receive t \Rightarrow fv t \mid _ \Rightarrow \{\}"$

The set of variables occurring in an equality constraint

definition $fv_{eq}::"poscheckvariant \Rightarrow ('a,'b) strand_step \Rightarrow 'b set$ **where**
 $"fv_{eq} ac x \equiv case x of Equality ac' s t \Rightarrow if ac = ac' then fv s \cup fv t else \{\} \mid _ \Rightarrow \{\}"$

The set of variables occurring at the left-hand side of an equality constraint

definition $fv_{leq}::"poscheckvariant \Rightarrow ('a,'b) strand_step \Rightarrow 'b set$ **where**
 $"fv_{leq} ac x \equiv case x of Equality ac' s t \Rightarrow if ac = ac' then fv s else \{\} \mid _ \Rightarrow \{\}"$

The set of variables occurring at the right-hand side of an equality constraint

definition $fv_{req}::"poscheckvariant \Rightarrow ('a,'b) strand_step \Rightarrow 'b set$ **where**
 $"fv_{req} ac x \equiv case x of Equality ac' s t \Rightarrow if ac = ac' then fv t else \{\} \mid _ \Rightarrow \{\}"$

The free variables of inequality constraints

definition $fv_{ineq}::('a,'b) strand_step \Rightarrow 'b set$ **where**
 $"fv_{ineq} x \equiv case x of Inequality X F \Rightarrow fv_{pairs} F - set X \mid _ \Rightarrow \{\}"$

fun $fv_{stp}::('a,'b) strand_step \Rightarrow 'b set$ **where**
 $"fv_{stp} (Send t) = fv t"$
 $\mid "fv_{stp} (Receive t) = fv t"$
 $\mid "fv_{stp} (Equality _ t t') = fv t \cup fv t'"$
 $\mid "fv_{stp} (Inequality X F) = (\bigcup (t,t') \in set F. fv t \cup fv t') - set X"$

The set of free variables of a strand

definition $fv_{st}::('a,'b) strand \Rightarrow 'b set$ **where**
 $"fv_{st} S \equiv \bigcup (set (map fv_{stp} S))"$

The set of bound variables of a strand

definition $bvars_{st}::('a,'b) strand \Rightarrow 'b set$ **where**
 $"bvars_{st} S \equiv \bigcup (set (map (set \circ bvars_{stp}) S))"$

The set of all variables occurring in a strand

definition $vars_{st}::('a,'b) strand \Rightarrow 'b set$ **where**
 $"vars_{st} S \equiv \bigcup (set (map vars_{stp} S))"$

abbreviation $wfrestrictedvars_{stp}::('a,'b) strand_step \Rightarrow 'b set$ **where**
 $"wfrestrictedvars_{stp} x \equiv$
 $case x of Inequality _ _ \Rightarrow \{\} \mid Equality Check _ _ \Rightarrow \{\} \mid _ \Rightarrow vars_{stp} x"$

The variables of a strand whose occurrences might be restricted by well-formedness constraints

definition $wfrestrictedvars_{st}::('a,'b) strand \Rightarrow 'b set$ **where**
 $"wfrestrictedvars_{st} S \equiv \bigcup (set (map wfrestrictedvars_{stp} S))"$

abbreviation $wfvarsoccs_{stp}$ **where**
 $"wfvarsoccs_{stp} x \equiv case x of Send t \Rightarrow fv t \mid Equality Assign s t \Rightarrow fv s \mid _ \Rightarrow \{\}"$

The variables of a strand that occur in sent messages or as variables in assignments

definition $wfvarsoccs_{st}$ **where**
 $"wfvarsoccs_{st} S \equiv \bigcup (set (map wfvarsoccs_{stp} S))"$

The variables occurring at the right-hand side of assignment steps

fun $assignment_rhs_{st}$ **where**

```

"assignment_rhs_st [] = {}"
| "assignment_rhs_st (Equality Assign t t'#S) = insert t' (assignment_rhs_st S)"
| "assignment_rhs_st (x#S) = assignment_rhs_st S"

```

The set function symbols occurring in a strand

```

definition funs_st::('a,'b) strand  $\Rightarrow$  'a set" where
  "funs_st S  $\equiv$   $\bigcup$  (set (map funs_stp S))"

```

```

fun subst_apply_strand_step::('a,'b) strand_step  $\Rightarrow$  ('a,'b) subst  $\Rightarrow$  ('a,'b) strand_step"
  (infix ".stp" 51) where
  "Send t .stp  $\vartheta$  = Send (t .  $\vartheta$ )"
| "Receive t .stp  $\vartheta$  = Receive (t .  $\vartheta$ )"
| "Equality a t t' .stp  $\vartheta$  = Equality a (t .  $\vartheta$ ) (t' .  $\vartheta$ )"
| "Inequality X F .stp  $\vartheta$  = Inequality X (F .pairs rm_vars (set X)  $\vartheta$ )"

```

Substitution application for strands

```

definition subst_apply_strand::('a,'b) strand  $\Rightarrow$  ('a,'b) subst  $\Rightarrow$  ('a,'b) strand"
  (infix ".st" 51) where
  "S .st  $\vartheta$   $\equiv$  map ( $\lambda$ x. x .stp  $\vartheta$ ) S"

```

The semantics of inequality constraints

```

definition
  "ineq_model ( $\mathcal{I}$ ::('a,'b) subst) X F  $\equiv$ 
    ( $\forall \delta$ . subst_domain  $\delta$  = set X  $\wedge$  ground (subst_range  $\delta$ )  $\longrightarrow$ 
      list_ex ( $\lambda$ f. fst f . ( $\delta$   $\circ_s$   $\mathcal{I}$ )  $\neq$  snd f . ( $\delta$   $\circ_s$   $\mathcal{I}$ )) F)"

```

```

fun simple_stp where
  "simple_stp (Receive t) = True"
| "simple_stp (Send (Var v)) = True"
| "simple_stp (Inequality X F) = ( $\exists \mathcal{I}$ . ineq_model  $\mathcal{I}$  X F)"
| "simple_stp _ = False"

```

Simple constraints

```

definition simple where "simple S  $\equiv$  list_all simple_stp S"

```

The intruder knowledge of a constraint

```

fun ik_st::('a,'b) strand  $\Rightarrow$  ('a,'b) terms" where
  "ik_st [] = {}"
| "ik_st (Receive t#S) = insert t (ik_st S)"
| "ik_st (_#S) = ik_st S"

```

Strand well-formedness

```

fun wf_st::'b set  $\Rightarrow$  ('a,'b) strand  $\Rightarrow$  bool" where
  "wf_st V [] = True"
| "wf_st V (Receive t#S) = (fv t  $\subseteq$  V  $\wedge$  wf_st V S)"
| "wf_st V (Send t#S) = wf_st (V  $\cup$  fv t) S"
| "wf_st V (Equality Assign s t#S) = (fv t  $\subseteq$  V  $\wedge$  wf_st (V  $\cup$  fv s) S)"
| "wf_st V (Equality Check s t#S) = wf_st V S"
| "wf_st V (Inequality _ #S) = wf_st V S"

```

Well-formedness of constraint states

```

definition wf_constr::('a,'b) strand  $\Rightarrow$  ('a,'b) subst  $\Rightarrow$  bool" where
  "wf_constr S  $\vartheta$   $\equiv$  (wf_subst  $\vartheta$   $\wedge$  wf_st {} S  $\wedge$  subst_domain  $\vartheta$   $\cap$  vars_st S = {}  $\wedge$ 
    range_vars  $\vartheta$   $\cap$  bvars_st S = {}  $\wedge$  fv_st S  $\cap$  bvars_st S = {})"

```

```

declare trms_st_def[simp]
declare fv_snd_def[simp]
declare fv_rcv_def[simp]
declare fv_eq_def[simp]
declare fv_leq_def[simp]
declare fv_req_def[simp]
declare fv_ineq_def[simp]

```

```

declare fvst_def[simp]
declare varsst_def[simp]
declare bvarsst_def[simp]
declare wfrestrictedvarsst_def[simp]
declare wfvarsoccsst_def[simp]

lemmas wfst_induct = wfst.induct[case_names Nil ConsRcv ConsSnd ConsEq ConsEq2 ConsIneq]
lemmas ikst_induct = ikst.induct[case_names Nil ConsRcv ConsSnd ConsEq ConsIneq]
lemmas assignment_rhsst_induct = assignment_rhsst.induct[case_names Nil ConsEq2 ConsSnd ConsRcv ConsEq ConsIneq]

```

Lexicographical measure on strands

```

definition sizest:: "('a, 'b) strand ⇒ nat" where
  "sizest S ≡ size_list (λx. Max (insert 0 (size ` trmsstp x))) S"

definition measurest:: "((('a, 'b) strand × ('a, 'b) subst) × ('a, 'b) strand × ('a, 'b) subst) set"
where
  "measurest ≡ measures [λ(S, ϑ). card (fvst S), λ(S, ϑ). sizest S]"

```

```

lemma measurest_alt_def:
  "((s, x), (t, y)) ∈ measurest =
    (card (fvst s) < card (fvst t) ∨ (card (fvst s) = card (fvst t) ∧ sizest s < sizest t))"
⟨proof⟩

```

```

lemma measurest_trans: "trans measurest"
⟨proof⟩

```

Some lemmata

```

lemma trms_listst_is_trmsst: "trmsst S = set (trms_listst S)"
⟨proof⟩

```

```

lemma subst_apply_strand_step_def:
  "s ·stp ϑ = (case s of
    Send t ⇒ Send (t · ϑ)
  | Receive t ⇒ Receive (t · ϑ)
  | Equality a t t' ⇒ Equality a (t · ϑ) (t' · ϑ)
  | Inequality X F ⇒ Inequality X (F ·pairs rm_vars (set X) ϑ))"
⟨proof⟩

```

```

lemma subst_apply_strand_nil[simp]: "[ ] ·st δ = [ ]"
⟨proof⟩

```

```

lemma finite_funsstp[simp]: "finite (funsstp x)" ⟨proof⟩
lemma finite_funsst[simp]: "finite (funsst S)" ⟨proof⟩
lemma finite_trmspairs[simp]: "finite (trmspairs x)" ⟨proof⟩
lemma finite_trmsstp[simp]: "finite (trmsstp x)" ⟨proof⟩
lemma finite_varsstp[simp]: "finite (varsstp x)" ⟨proof⟩
lemma finite_bvarsstp[simp]: "finite (set (bvarsstp x))" ⟨proof⟩
lemma finite_fvsnd[simp]: "finite (fvsnd x)" ⟨proof⟩
lemma finite_fvrcv[simp]: "finite (fvrcv x)" ⟨proof⟩
lemma finite_fvstp[simp]: "finite (fvstp x)" ⟨proof⟩
lemma finite_varsst[simp]: "finite (varsst S)" ⟨proof⟩
lemma finite_bvarsst[simp]: "finite (bvarsst S)" ⟨proof⟩
lemma finite_fvst[simp]: "finite (fvst S)" ⟨proof⟩

```

```

lemma finite_wfrestrictedvarsstp[simp]: "finite (wfrestrictedvarsstp x)"
⟨proof⟩

```

```

lemma finite_wfrestrictedvarsst[simp]: "finite (wfrestrictedvarsst S)"
⟨proof⟩

```

```

lemma finite_wfvarsoccsstp[simp]: "finite (wfvarsoccsstp x)"
<proof>

lemma finite_wfvarsoccsst[simp]: "finite (wfvarsoccsst S)"
<proof>

lemma finite_ikst[simp]: "finite (ikst S)"
<proof>

lemma finite_assignment_rhsst[simp]: "finite (assignment_rhsst S)"
<proof>

lemma ikst_is_rcv_set: "ikst A = {t. Receive t ∈ set A}"
<proof>

lemma ikstD[dest]: "t ∈ ikst S ⇒ Receive t ∈ set S"
<proof>

lemma ikstD'[dest]: "t ∈ ikst S ⇒ t ∈ trmsst S"
<proof>

lemma ikstD''[dest]: "t ∈ subtermsset (ikst S) ⇒ t ∈ subtermsset (trmsst S)"
<proof>

lemma ikst_subterm_exD:
  assumes "t ∈ ikst S"
  shows "∃x ∈ set S. t ∈ subtermsset (trmsstp x)"
<proof>

lemma assignment_rhsstD[dest]: "t ∈ assignment_rhsst S ⇒ ∃t'. Equality Assign t' t ∈ set S"
<proof>

lemma assignment_rhsstD'[dest]: "t ∈ subtermsset (assignment_rhsst S) ⇒ t ∈ subtermsset (trmsst S)"
<proof>

lemma bvarsst_split: "bvarsst (S@S') = bvarsst S ∪ bvarsst S'"
<proof>

lemma bvarsst_singleton: "bvarsst [x] = set (bvarsstp x)"
<proof>

lemma strand_fv_bvars_disjointD:
  assumes "fvst S ∩ bvarsst S = {}" "Inequality X F ∈ set S"
  shows "set X ⊆ bvarsst S" "fvpairs F - set X ⊆ fvst S"
<proof>

lemma strand_fv_bvars_disjoint_unfold:
  assumes "fvst S ∩ bvarsst S = {}" "Inequality X F ∈ set S" "Inequality Y G ∈ set S"
  shows "set Y ∩ (fvpairs F - set X) = {}"
<proof>

lemma strand_subst_hom[iff]:
  "(S@S') ·st ∅ = (S ·st ∅)@(S' ·st ∅)" "(x#S) ·st ∅ = (x ·stp ∅)#(S ·st ∅)"
<proof>

lemma strand_subst_comp: "range_vars δ ∩ bvarsst S = {} ⇒ S ·st δ ∘s ∅ = ((S ·st δ) ·st ∅)"
<proof>

lemma strand_substI[intro]:
  "subst_domain ∅ ∩ fvst S = {} ⇒ S ·st ∅ = S"
  "subst_domain ∅ ∩ varsst S = {} ⇒ S ·st ∅ = S"
<proof>
    
```

3 The Typing Result for Non-Stateful Protocols

lemma `strand_substI'`:

"fv_{st} S = {} \implies S ·_{st} ϑ = S"

"vars_{st} S = {} \implies S ·_{st} ϑ = S"

<proof>

lemma `strand_subst_set`: "(set (S ·_{st} ϑ)) = ((λ x. x ·_{stp} ϑ) ` (set S))"

<proof>

lemma `strand_map_inv_set_snd_rcv_subst`:

assumes "finite (M::('a,'b) terms)"

shows "set ((map Send (inv set M)) ·_{st} ϑ) = Send ` (M ·_{set} ϑ)" (is ?A)

"set ((map Receive (inv set M)) ·_{st} ϑ) = Receive ` (M ·_{set} ϑ)" (is ?B)

<proof>

lemma `strand_ground_subst_vars_subset`:

assumes "ground (subst_range ϑ)" shows "vars_{st} (S ·_{st} ϑ) \subseteq vars_{st} S"

<proof>

lemma `ik_union_subset`: " \bigcup (P ` ik_{st} S) \subseteq (\bigcup x \in (set S). \bigcup (P ` trms_{stp} x))"

<proof>

lemma `ik_snd_empty[simp]`: "ik_{st} (map Send X) = {}"

<proof>

lemma `ik_snd_empty'[simp]`: "ik_{st} [Send t] = {}" *<proof>*

lemma `ik_append[iff]`: "ik_{st} (S@S') = ik_{st} S \cup ik_{st} S'" *<proof>*

lemma `ik_cons`: "ik_{st} (x#S) = ik_{st} [x] \cup ik_{st} S" *<proof>*

lemma `assignment_rhs_append[iff]`: "assignment_rhs_{st} (S@S') = assignment_rhs_{st} S \cup assignment_rhs_{st} S'"

<proof>

lemma `eqs_rcv_map_empty`: "assignment_rhs_{st} (map Receive M) = {}"

<proof>

lemma `ik_rcv_map`: assumes "t \in set L" shows "t \in ik_{st} (map Receive L)"

<proof>

lemma `ik_subst`: "ik_{st} (S ·_{st} δ) = ik_{st} S ·_{set} δ "

<proof>

lemma `ik_rcv_map'`: assumes "t \in ik_{st} (map Receive L)" shows "t \in set L"

<proof>

lemma `ik_append_subset[simp]`: "ik_{st} S \subseteq ik_{st} (S@S'" "ik_{st} S' \subseteq ik_{st} (S@S'"

<proof>

lemma `assignment_rhs_append_subset[simp]`:

"assignment_rhs_{st} S \subseteq assignment_rhs_{st} (S@S'"

"assignment_rhs_{st} S' \subseteq assignment_rhs_{st} (S@S'"

<proof>

lemma `trmsst_cons`: "trms_{st} (x#S) = trms_{stp} x \cup trms_{st} S" *<proof>*

lemma `trm_strand_subst_cong`:

"t \in trms_{st} S \implies t · δ \in trms_{st} (S ·_{st} δ)

\vee (\exists X F. Inequality X F \in set S \wedge t · rm_vars (set X) δ \in trms_{st} (S ·_{st} δ))"

(is "t \in trms_{st} S \implies ?P t δ S")

"t \in trms_{st} (S ·_{st} δ) \implies (\exists t'. t = t' · δ \wedge t' \in trms_{st} S)

\vee (\exists X F. Inequality X F \in set S \wedge (\exists t' \in trms_{pairs} F. t = t' · rm_vars (set X) δ))"

(is "t \in trms_{st} (S ·_{st} δ) \implies ?Q t δ S")

<proof>

3.1.2 Lemmata: Free Variables of Strands

lemma *fv_trm_snd_rcv[simp]*: "fv_{set} (trms_{stp} (Send t)) = fv t" "fv_{set} (trms_{stp} (Receive t)) = fv t"
 ⟨proof⟩

lemma *in_strand_fv_subset*: "x ∈ set S ⇒ vars_{stp} x ⊆ vars_{st} S" ⟨proof⟩

lemma *in_strand_fv_subset_snd*: "Send t ∈ set S ⇒ fv t ⊆ ⋃ (set (map fv_{snd} S))" ⟨proof⟩

lemma *in_strand_fv_subset_rcv*: "Receive t ∈ set S ⇒ fv t ⊆ ⋃ (set (map fv_{rcv} S))" ⟨proof⟩

lemma *fv_{snd}E*:

assumes "v ∈ ⋃ (set (map fv_{snd} S))"

obtains t where "send⟨t⟩_{st} ∈ set S" "v ∈ fv t"

⟨proof⟩

lemma *fv_{rcv}E*:

assumes "v ∈ ⋃ (set (map fv_{rcv} S))"

obtains t where "receive⟨t⟩_{st} ∈ set S" "v ∈ fv t"

⟨proof⟩

lemma *vars_{stp}I[intro]*: "x ∈ fv_{stp} s ⇒ x ∈ vars_{stp} s"

⟨proof⟩

lemma *vars_{st}I[intro]*: "x ∈ fv_{st} S ⇒ x ∈ vars_{st} S" ⟨proof⟩

lemma *fv_{st}_subset_vars_{st}[simp]*: "fv_{st} S ⊆ vars_{st} S" ⟨proof⟩

lemma *vars_{st}_is_fv_{st}_bvars_{st}*: "vars_{st} S = fv_{st} S ∪ bvars_{st} S"

⟨proof⟩

lemma *fv_{stp}_is_subterm_trms_{stp}*: "x ∈ fv_{stp} a ⇒ Var x ∈ subterms_{set} (trms_{stp} a)"

⟨proof⟩

lemma *fv_{st}_is_subterm_trms_{st}*: "x ∈ fv_{st} A ⇒ Var x ∈ subterms_{set} (trms_{st} A)"

⟨proof⟩

lemma *vars_{st}_snd_map*: "vars_{st} (map Send X) = fv (Fun f X)" ⟨proof⟩

lemma *vars_{st}_rcv_map*: "vars_{st} (map Receive X) = fv (Fun f X)" ⟨proof⟩

lemma *vars_snd_rcv_union*:

"vars_{stp} x = fv_{snd} x ∪ fv_{rcv} x ∪ fv_{eq} assign x ∪ fv_{eq} check x ∪ fv_{ineq} x ∪ set (bvars_{stp} x)"

⟨proof⟩

lemma *fv_snd_rcv_union*:

"fv_{stp} x = fv_{snd} x ∪ fv_{rcv} x ∪ fv_{eq} assign x ∪ fv_{eq} check x ∪ fv_{ineq} x"

⟨proof⟩

lemma *fv_snd_rcv_empty[simp]*: "fv_{snd} x = {} ∨ fv_{rcv} x = {}" ⟨proof⟩

lemma *vars_snd_rcv_strand[iff]*:

"vars_{st} (S::('a,'b) strand) =
 (⋃ (set (map fv_{snd} S))) ∪ (⋃ (set (map fv_{rcv} S))) ∪ (⋃ (set (map (fv_{eq} assign) S)))
 ∪ (⋃ (set (map (fv_{eq} check) S))) ∪ (⋃ (set (map fv_{ineq} S))) ∪ bvars_{st} S"

⟨proof⟩

lemma *fv_snd_rcv_strand[iff]*:

"fv_{st} (S::('a,'b) strand) =
 (⋃ (set (map fv_{snd} S))) ∪ (⋃ (set (map fv_{rcv} S))) ∪ (⋃ (set (map (fv_{eq} assign) S)))
 ∪ (⋃ (set (map (fv_{eq} check) S))) ∪ (⋃ (set (map fv_{ineq} S)))"

⟨proof⟩

lemma *vars_snd_rcv_strand2[iff]*:

"wfrestrictedvars_{st} (S::('a,'b) strand) =

3 The Typing Result for Non-Stateful Protocols

$(\bigcup(\text{set}(\text{map } fv_{snd} S)) \cup (\bigcup(\text{set}(\text{map } fv_{rcv} S))) \cup (\bigcup(\text{set}(\text{map} (fv_{eq} \text{ assign}) S))))$
 $\langle \text{proof} \rangle$

lemma $fv_{snd_rcv_strand_subset[simp]}$:

" $\bigcup(\text{set}(\text{map } fv_{snd} S)) \subseteq fv_{st} S$ " " $\bigcup(\text{set}(\text{map } fv_{rcv} S)) \subseteq fv_{st} S$ "
" $\bigcup(\text{set}(\text{map} (fv_{eq} \text{ ac}) S)) \subseteq fv_{st} S$ " " $\bigcup(\text{set}(\text{map } fv_{ineq} S)) \subseteq fv_{st} S$ "
" $wfvarsoccs_{st} S \subseteq fv_{st} S$ "

$\langle \text{proof} \rangle$

lemma $vars_{snd_rcv_strand_subset2[simp]}$:

" $\bigcup(\text{set}(\text{map } fv_{snd} S)) \subseteq wfrestrictedvars_{st} S$ " " $\bigcup(\text{set}(\text{map } fv_{rcv} S)) \subseteq wfrestrictedvars_{st} S$ "
" $\bigcup(\text{set}(\text{map} (fv_{eq} \text{ assign}) S)) \subseteq wfrestrictedvars_{st} S$ " " $wfvarsoccs_{st} S \subseteq wfrestrictedvars_{st} S$ "

$\langle \text{proof} \rangle$

lemma $wfrestrictedvars_{st_subset_vars_{st}}$: " $wfrestrictedvars_{st} S \subseteq vars_{st} S$ "

$\langle \text{proof} \rangle$

lemma $subst_sends_strand_step_fv_to_img$: " $fv_{stp} (x \cdot_{stp} \delta) \subseteq fv_{stp} x \cup \text{range_vars } \delta$ "

$\langle \text{proof} \rangle$

lemma $subst_sends_strand_fv_to_img$: " $fv_{st} (S \cdot_{st} \delta) \subseteq fv_{st} S \cup \text{range_vars } \delta$ "

$\langle \text{proof} \rangle$

lemma $ineq_apply_subst$:

assumes " $\text{subst_domain } \delta \cap \text{set } X = \{\}$ "
shows " $(\text{Inequality } X F) \cdot_{stp} \delta = \text{Inequality } X (F \cdot_{pairs} \delta)$ "

$\langle \text{proof} \rangle$

lemma $fv_strand_step_subst$:

assumes " $P = fv_{stp} \vee P = fv_{rcv} \vee P = fv_{snd} \vee P = fv_{eq} \text{ ac} \vee P = fv_{ineq}$ "
and " $\text{set}(\text{bvars}_{stp} x) \cap (\text{subst_domain } \delta \cup \text{range_vars } \delta) = \{\}$ "
shows " $fv_{set} (\delta \setminus (P x)) = P (x \cdot_{stp} \delta)$ "

$\langle \text{proof} \rangle$

lemma fv_strand_subst :

assumes " $P = fv_{stp} \vee P = fv_{rcv} \vee P = fv_{snd} \vee P = fv_{eq} \text{ ac} \vee P = fv_{ineq}$ "
and " $\text{bvars}_{st} S \cap (\text{subst_domain } \delta \cup \text{range_vars } \delta) = \{\}$ "
shows " $fv_{set} (\delta \setminus (\bigcup(\text{set}(\text{map } P S)))) = \bigcup(\text{set}(\text{map } P (S \cdot_{st} \delta)))$ "

$\langle \text{proof} \rangle$

lemma fv_strand_subst2 :

assumes " $\text{bvars}_{st} S \cap (\text{subst_domain } \delta \cup \text{range_vars } \delta) = \{\}$ "
shows " $fv_{set} (\delta \setminus (wfrestrictedvars_{st} S)) = wfrestrictedvars_{st} (S \cdot_{st} \delta)$ "

$\langle \text{proof} \rangle$

lemma fv_strand_subst' :

assumes " $\text{bvars}_{st} S \cap (\text{subst_domain } \delta \cup \text{range_vars } \delta) = \{\}$ "
shows " $fv_{set} (\delta \setminus (fv_{st} S)) = fv_{st} (S \cdot_{st} \delta)$ "

$\langle \text{proof} \rangle$

lemma $fv_trms_{pairs_is_fv_{pairs}}$:

" $fv_{set} (\text{trms}_{pairs} F) = fv_{pairs} F$ "

$\langle \text{proof} \rangle$

lemma $fv_{pairs_in_fv_trms_{pairs}}$: " $x \in fv_{pairs} F \implies x \in fv_{set} (\text{trms}_{pairs} F)$ "

$\langle \text{proof} \rangle$

lemma $trms_{st_append}$: " $\text{trms}_{st} (A \circ B) = \text{trms}_{st} A \cup \text{trms}_{st} B$ "

$\langle \text{proof} \rangle$

lemma $trms_{pairs_subst}$: " $\text{trms}_{pairs} (a \cdot_{pairs} \vartheta) = \text{trms}_{pairs} a \cdot_{set} \vartheta$ "

$\langle \text{proof} \rangle$

```

lemma trms_pairs_fv_subst_subset:
  "t ∈ trms_pairs F ⇒ fv (t · ∅) ⊆ fv_pairs (F ·_pairs ∅)"
⟨proof⟩

lemma trms_pairs_fv_subst_subset':
  fixes t::('a,'b) term and ∅::('a,'b) subst
  assumes "t ∈ subterms_set (trms_pairs F)"
  shows "fv (t · ∅) ⊆ fv_pairs (F ·_pairs ∅)"
⟨proof⟩

lemma trms_pairs_funs_term_cases:
  assumes "t ∈ trms_pairs (F ·_pairs ∅)" "f ∈ funs_term t"
  shows "(∃ u ∈ trms_pairs F. f ∈ funs_term u) ∨ (∃ x ∈ fv_pairs F. f ∈ funs_term (∅ x))"
⟨proof⟩

lemma trm_stp_subst:
  assumes "subst_domain ∅ ∩ set (bvars_stp a) = {}"
  shows "trms_stp (a ·_stp ∅) = trms_stp a ·_set ∅"
⟨proof⟩

lemma trms_st_subst:
  assumes "subst_domain ∅ ∩ bvars_st A = {}"
  shows "trms_st (A ·_st ∅) = trms_st A ·_set ∅"
⟨proof⟩

lemma strand_map_set_subst:
  assumes δ: "bvars_st S ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "⋃(set (map trms_stp (S ·_st δ))) = (⋃(set (map trms_stp S))) ·_set δ"
⟨proof⟩

lemma subst_apply_fv_subset_strand_trm:
  assumes P: "P = fv_stp ∨ P = fv_rcv ∨ P = fv_snd ∨ P = fv_eq ac ∨ P = fv_ineq"
  and fv_sub: "fv t ⊆ ⋃(set (map P S)) ∪ V"
  and δ: "bvars_st S ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "fv (t · δ) ⊆ ⋃(set (map P (S ·_st δ))) ∪ fv_set (δ ` V)"
⟨proof⟩

lemma subst_apply_fv_subset_strand_trm2:
  assumes fv_sub: "fv t ⊆ wfrestrictedvars_st S ∪ V"
  and δ: "bvars_st S ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "fv (t · δ) ⊆ wfrestrictedvars_st (S ·_st δ) ∪ fv_set (δ ` V)"
⟨proof⟩

lemma subst_apply_fv_subset_strand:
  assumes P: "P = fv_stp ∨ P = fv_rcv ∨ P = fv_snd ∨ P = fv_eq ac ∨ P = fv_ineq"
  and P_subset: "P x ⊆ ⋃(set (map P S)) ∪ V"
  and δ: "bvars_st S ∩ (subst_domain δ ∪ range_vars δ) = {}"
  "set (bvars_stp x) ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "P (x ·_stp δ) ⊆ ⋃(set (map P (S ·_st δ))) ∪ fv_set (δ ` V)"
⟨proof⟩

lemma subst_apply_fv_subset_strand2:
  assumes P: "P = fv_stp ∨ P = fv_rcv ∨ P = fv_snd ∨ P = fv_eq ac ∨ P = fv_ineq ∨ P = fv_req ac"
  and P_subset: "P x ⊆ wfrestrictedvars_st S ∪ V"
  and δ: "bvars_st S ∩ (subst_domain δ ∪ range_vars δ) = {}"
  "set (bvars_stp x) ∩ (subst_domain δ ∪ range_vars δ) = {}"
  shows "P (x ·_stp δ) ⊆ wfrestrictedvars_st (S ·_st δ) ∪ fv_set (δ ` V)"
⟨proof⟩

lemma strand_subst_fv_bounded_if_img_bounded:
  assumes "range_vars δ ⊆ fv_st S"
  shows "fv_st (S ·_st δ) ⊆ fv_st S"
⟨proof⟩
    
```

3 The Typing Result for Non-Stateful Protocols

lemma strand_fv_subst_subset_if_subst_elim:

assumes "subst_elim δ v " and " $v \in \text{fv}_{st} S \vee \text{bvars}_{st} S \cap (\text{subst_domain } \delta \cup \text{range_vars } \delta) = \{\}$ "
 shows " $v \notin \text{fv}_{st} (S \cdot_{st} \delta)$ "

<proof>

lemma strand_fv_subst_subset_if_subst_elim':

assumes "subst_elim δ v " " $v \in \text{fv}_{st} S$ " " $\text{range_vars } \delta \subseteq \text{fv}_{st} S$ "
 shows " $\text{fv}_{st} (S \cdot_{st} \delta) \subseteq \text{fv}_{st} S$ "

<proof>

lemma fv_ik_is_fv_rcv: " $\text{fv}_{set} (\text{ik}_{st} S) = \bigcup (\text{set } (\text{map } \text{fv}_{rcv} S))$ "

<proof>

lemma fv_ik_subset_fv_st[simp]: " $\text{fv}_{set} (\text{ik}_{st} S) \subseteq \text{wfrestrictedvars}_{st} S$ "

<proof>

lemma fv_assignment_rhs_subset_fv_st[simp]: " $\text{fv}_{set} (\text{assignment_rhs}_{st} S) \subseteq \text{wfrestrictedvars}_{st} S$ "

<proof>

lemma fv_ik_subset_fv_st'[simp]: " $\text{fv}_{set} (\text{ik}_{st} S) \subseteq \text{fv}_{st} S$ "

<proof>

lemma ik_st_var_is_fv: " $\text{Var } x \in \text{subterms}_{set} (\text{ik}_{st} A) \implies x \in \text{fv}_{st} A$ "

<proof>

lemma fv_assignment_rhs_subset_fv_st'[simp]: " $\text{fv}_{set} (\text{assignment_rhs}_{st} S) \subseteq \text{fv}_{st} S$ "

<proof>

lemma ik_st_assignment_rhs_st_wfrestrictedvars_subset:

" $\text{fv}_{set} (\text{ik}_{st} A \cup \text{assignment_rhs}_{st} A) \subseteq \text{wfrestrictedvars}_{st} A$ "

<proof>

lemma strand_step_id_subst[iff]: " $x \cdot_{stp} \text{Var} = x$ " *<proof>*

lemma strand_id_subst[iff]: " $S \cdot_{st} \text{Var} = S$ " *<proof>*

lemma strand_subst_vars_union_bound[simp]: " $\text{vars}_{st} (S \cdot_{st} \delta) \subseteq \text{vars}_{st} S \cup \text{range_vars } \delta$ "

<proof>

lemma strand_vars_split:

" $\text{vars}_{st} (S@S') = \text{vars}_{st} S \cup \text{vars}_{st} S'$ "

" $\text{wfrestrictedvars}_{st} (S@S') = \text{wfrestrictedvars}_{st} S \cup \text{wfrestrictedvars}_{st} S'$ "

" $\text{fv}_{st} (S@S') = \text{fv}_{st} S \cup \text{fv}_{st} S'$ "

<proof>

lemma bvars_subst_ident: " $\text{bvars}_{st} S = \text{bvars}_{st} (S \cdot_{st} \delta)$ "

<proof>

lemma strand_subst_subst_idem:

assumes "subst_idem δ " " $\text{subst_domain } \delta \cup \text{range_vars } \delta \subseteq \text{fv}_{st} S$ " " $\text{subst_domain } \vartheta \cap \text{fv}_{st} S = \{\}$ "
 " $\text{range_vars } \delta \cap \text{bvars}_{st} S = \{\}$ " " $\text{range_vars } \vartheta \cap \text{bvars}_{st} S = \{\}$ "

shows " $(S \cdot_{st} \delta) \cdot_{st} \vartheta = (S \cdot_{st} \delta)$ "

and " $(S \cdot_{st} \delta) \cdot_{st} (\vartheta \circ_s \delta) = (S \cdot_{st} \delta)$ "

<proof>

lemma strand_subst_img_bound:

assumes " $\text{subst_domain } \delta \cup \text{range_vars } \delta \subseteq \text{fv}_{st} S$ "

and " $(\text{subst_domain } \delta \cup \text{range_vars } \delta) \cap \text{bvars}_{st} S = \{\}$ "

shows " $\text{range_vars } \delta \subseteq \text{fv}_{st} (S \cdot_{st} \delta)$ "

<proof>

lemma strand_subst_img_bound':

```

assumes "subst_domain  $\delta \cup \text{range\_vars } \delta \subseteq \text{vars}_{st} S$ "
and "(subst_domain  $\delta \cup \text{range\_vars } \delta) \cap \text{bvars}_{st} S = \{\}$ "
shows "range_vars  $\delta \subseteq \text{vars}_{st} (S \cdot_{st} \delta)$ "
<proof>

lemma strand_subst_all_fv_subset:
assumes "fv  $t \subseteq \text{fv}_{st} S$ " "(subst_domain  $\delta \cup \text{range\_vars } \delta) \cap \text{bvars}_{st} S = \{\}$ "
shows "fv  $(t \cdot \delta) \subseteq \text{fv}_{st} (S \cdot_{st} \delta)$ "
<proof>

lemma strand_subst_not_dom_fixed:
assumes " $v \in \text{fv}_{st} S$ " and " $v \notin \text{subst\_domain } \delta$ "
shows " $v \in \text{fv}_{st} (S \cdot_{st} \delta)$ "
<proof>

lemma strand_vars_unfold: " $v \in \text{vars}_{st} S \implies \exists S' x S''. S = S'@x\#S'' \wedge v \in \text{vars}_{stp} x$ "
<proof>

lemma strand_fv_unfold: " $v \in \text{fv}_{st} S \implies \exists S' x S''. S = S'@x\#S'' \wedge v \in \text{fv}_{stp} x$ "
<proof>

lemma subterm_if_in_strand_ik:
" $t \in \text{ik}_{st} S \implies \exists t'. \text{Receive } t' \in \text{set } S \wedge t \sqsubseteq t'$ "
<proof>

lemma fv_subset_if_in_strand_ik:
" $t \in \text{ik}_{st} S \implies \text{fv } t \subseteq \bigcup (\text{set } (\text{map } \text{fv}_{rcv} S))$ "
<proof>

lemma fv_subset_if_in_strand_ik':
" $t \in \text{ik}_{st} S \implies \text{fv } t \subseteq \text{fv}_{st} S$ "
<proof>

lemma vars_subset_if_in_strand_ik2:
" $t \in \text{ik}_{st} S \implies \text{fv } t \subseteq \text{wfrestrictedvars}_{st} S$ "
<proof>

3.1.3 Lemmata: Simple Strands

lemma simple_Cons[dest]: "simple  $(s\#S) \implies \text{simple } S$ "
<proof>

lemma simple_split[dest]:
assumes "simple  $(S@S')$ "
shows "simple  $S$ " "simple  $S'$ "
<proof>

lemma simple_append[intro]: " $\llbracket \text{simple } S; \text{simple } S' \rrbracket \implies \text{simple } (S@S')$ "
<proof>

lemma simple_append_sym[sym]: "simple  $(S@S') \implies \text{simple } (S'\@S)$ " <proof>

lemma not_simple_if_snd_fun: " $(\exists S' S'' f X. S = S'\@Send (\text{Fun } f X)\#S'') \implies \neg \text{simple } S$ "
<proof>

lemma not_list_all_elim: " $\neg \text{list\_all } P A \implies \exists B x C. A = B@x\#C \wedge \neg P x \wedge \text{list\_all } P B$ "
<proof>

lemma not_simple_stp_elim:
assumes " $\neg \text{simple}_{stp} x$ "
shows " $(\exists f T. x = \text{Send } (\text{Fun } f T)) \vee$ 
 $(\exists a t t'. x = \text{Equality } a t t') \vee$ 
 $(\exists X F. x = \text{Inequality } X F \wedge \neg (\exists \mathcal{I}. \text{ineq\_model } \mathcal{I} X F))$ "

```

<proof>

lemma *not_simple_elim*:

assumes " \neg simple S"

shows " $(\exists A B f T. S = A@Send (Fun f T)#B \wedge simple A) \vee$
 $(\exists A B a t t'. S = A@Equality a t t'#B \wedge simple A) \vee$
 $(\exists A B X F. S = A@Inequality X F#B \wedge \neg(\exists \mathcal{I}. ineq_model \mathcal{I} X F))$ "

<proof>

lemma *simple_fun_prefix_unique*:

assumes " $A = S@Send (Fun f X)#S'$ " "simple S"

shows " $\forall T g Y T'. A = T@Send (Fun g Y)#T' \wedge simple T \longrightarrow S = T \wedge f = g \wedge X = Y \wedge S' = T'$ "

<proof>

lemma *simple_snd_is_var*: " $[Send t \in set S; simple S] \Longrightarrow \exists v. t = Var v$ "

<proof>

3.1.4 Lemmata: Strand Measure

lemma *measure_{st}_wellfounded*: "*wf* measure_{st}" *<proof>*

lemma *strand_size_append[iff]*: " $size_{st} (S@S') = size_{st} S + size_{st} S'$ "

<proof>

lemma *strand_size_map_fun_lt[simp]*:

" $size_{st} (map Send X) < size (Fun f X)$ "

" $size_{st} (map Send X) < size_{st} [Send (Fun f X)]$ "

" $size_{st} (map Send X) < size_{st} [Receive (Fun f X)]$ "

<proof>

lemma *strand_size_rm_fun_lt[simp]*:

" $size_{st} (S@S') < size_{st} (S@Send (Fun f X)#S')$ "

" $size_{st} (S@S') < size_{st} (S@Receive (Fun f X)#S')$ "

<proof>

lemma *strand_fv_card_map_fun_eq*:

" $card (fv_{st} (S@Send (Fun f X)#S')) = card (fv_{st} (S@(map Send X)@S'))$ "

<proof>

lemma *strand_fv_card_rm_fun_le[simp]*: " $card (fv_{st} (S@S')) \leq card (fv_{st} (S@Send (Fun f X)#S'))$ "

<proof>

lemma *strand_fv_card_rm_eq_le[simp]*: " $card (fv_{st} (S@S')) \leq card (fv_{st} (S@Equality a t t'#S'))$ "

<proof>

3.1.5 Lemmata: Well-formed Strands

lemma *wf_prefix[dest]*: " $wf_{st} V (S@S') \Longrightarrow wf_{st} V S$ "

<proof>

lemma *wf_vars_mono[simp]*: " $wf_{st} V S \Longrightarrow wf_{st} (V \cup W) S$ "

<proof>

lemma *wf_{st}I[intro]*: " $wf_{restrictedvars_{st}} S \subseteq V \Longrightarrow wf_{st} V S$ "

<proof>

lemma *wf_{st}I'[intro]*: " $\bigcup (fv_{rcv} \setminus set S) \cup \bigcup (fv_{req} \setminus assign \setminus set S) \subseteq V \Longrightarrow wf_{st} V S$ "

<proof>

lemma *wf_append_exec*: " $wf_{st} V (S@S') \Longrightarrow wf_{st} (V \cup wfvarsoccs_{st} S) S'$ "

<proof>

lemma *wf_append_suffix*:

" $wf_{st} V S \implies wf_{restrictedvars_{st}} S' \subseteq wf_{restrictedvars_{st}} S \cup V \implies wf_{st} V (S@S')$ "
 <proof>

lemma `wf_append_suffix'`:
 assumes " $wf_{st} V S$ "
 and " $\bigcup (fv_{rcv} \setminus set S') \cup \bigcup (fv_{req} \setminus assign \setminus set S') \subseteq wfvarsoccs_{st} S \cup V$ "
 shows " $wf_{st} V (S@S')$ "
 <proof>

lemma `wf_send_compose`: " $wf_{st} V (S@(map Send X)@S') = wf_{st} V (S@Send (Fun f X)#S')$ "
 <proof>

lemma `wf_snd_append[iff]`: " $wf_{st} V (S@[Send t]) = wf_{st} V S$ "
 <proof>

lemma `wf_snd_append'`: " $wf_{st} V S \implies wf_{st} V (Send t#S)$ "
 <proof>

lemma `wf_rcv_append[dest]`: " $wf_{st} V (S@Receive t#S') \implies wf_{st} V (S@S')$ "
 <proof>

lemma `wf_rcv_append'[intro]`:
 " $\llbracket wf_{st} V (S@S'); fv t \subseteq wf_{restrictedvars_{st}} S \cup V \rrbracket \implies wf_{st} V (S@Receive t#S')$ "
 <proof>

lemma `wf_rcv_append''[intro]`: " $\llbracket wf_{st} V S; fv t \subseteq \bigcup (set (map fv_{snd} S)) \rrbracket \implies wf_{st} V (S@[Receive t])$ "
 <proof>

lemma `wf_rcv_append'''[intro]`: " $\llbracket wf_{st} V S; fv t \subseteq wf_{restrictedvars_{st}} S \cup V \rrbracket \implies wf_{st} V (S@[Receive t])$ "
 <proof>

lemma `wf_eq_append[dest]`: " $wf_{st} V (S@Equality a t t'#S') \implies fv t \subseteq wf_{restrictedvars_{st}} S \cup V \implies wf_{st} V (S@S')$ "
 <proof>

lemma `wf_eq_append'[intro]`:
 " $\llbracket wf_{st} V (S@S'); fv t' \subseteq wf_{restrictedvars_{st}} S \cup V \rrbracket \implies wf_{st} V (S@Equality a t t'#S')$ "
 <proof>

lemma `wf_eq_append''[intro]`:
 " $\llbracket wf_{st} V (S@S'); fv t' \subseteq wfvarsoccs_{st} S \cup V \rrbracket \implies wf_{st} V (S@[Equality a t t']@S')$ "
 <proof>

lemma `wf_eq_append'''[intro]`:
 " $\llbracket wf_{st} V S; fv t' \subseteq wf_{restrictedvars_{st}} S \cup V \rrbracket \implies wf_{st} V (S@[Equality a t t'])$ "
 <proof>

lemma `wf_eq_check_append[dest]`: " $wf_{st} V (S@Equality Check t t'#S') \implies wf_{st} V (S@S')$ "
 <proof>

lemma `wf_eq_check_append'[intro]`: " $wf_{st} V (S@S') \implies wf_{st} V (S@Equality Check t t'#S')$ "
 <proof>

lemma `wf_eq_check_append''[intro]`: " $wf_{st} V S \implies wf_{st} V (S@[Equality Check t t'])$ "
 <proof>

lemma `wf_ineq_append[dest]`: " $wf_{st} V (S@Inequality X F#S') \implies wf_{st} V (S@S')$ "
 <proof>

lemma `wf_ineq_append'[intro]`: " $wf_{st} V (S@S') \implies wf_{st} V (S@Inequality X F#S')$ "
 <proof>

3 The Typing Result for Non-Stateful Protocols

lemma `wf_ineq_append'` [intro]: " $wf_{st} V S \implies wf_{st} V (S@[Inequality X F])$ "
 <proof>

lemma `wf_rcv_fv_single`[elim]: " $wf_{st} V (\text{Receive } t\#S') \implies fv\ t \subseteq V$ "
 <proof>

lemma `wf_rcv_fv`: " $wf_{st} V (S@Receive\ t\#S') \implies fv\ t \subseteq wfvarsoccs_{st}\ S \cup V$ "
 <proof>

lemma `wf_eq_fv`: " $wf_{st} V (S@Equality\ Assign\ t\ t'\#S') \implies fv\ t' \subseteq wfvarsoccs_{st}\ S \cup V$ "
 <proof>

lemma `wf_simple_fv_occurrence`:
 assumes " $wf_{st}\ \{\}\ S$ " " $simple\ S$ " " $v \in wfrestrictedvars_{st}\ S$ "
 shows " $\exists S_{pre}\ S_{suf}. S = S_{pre}@Send\ (Var\ v)\#S_{suf} \wedge v \notin wfrestrictedvars_{st}\ S_{pre}$ "
 <proof>

lemma `Unifier_strand_fv_subset`:
 assumes `g_in_ik`: " $t \in ik_{st}\ S$ "
 and `δ`: " $Unifier\ \delta\ (Fun\ f\ X)\ t$ "
 and `disj`: " $bvars_{st}\ S \cap (subst_domain\ \delta \cup range_vars\ \delta) = \{\}$ "
 shows " $fv\ (Fun\ f\ X \cdot \delta) \subseteq \bigcup (set\ (map\ fv_{rcv}\ (S \cdot_{st}\ \delta)))$ "
 <proof>

lemma `wf_st_induct'` [consumes 1, case_names Nil ConsSnd ConsRcv ConsEq ConsEq2 ConsIneq]:
 fixes `S`: " $(a, b)\ strand$ "
 assumes " $wf_{st}\ V\ S$ "
 " $P\ []$ "
 " $\bigwedge t\ S. \llbracket wf_{st}\ V\ S; P\ S \rrbracket \implies P\ (S@[Send\ t])$ "
 " $\bigwedge t\ S. \llbracket wf_{st}\ V\ S; P\ S; fv\ t \subseteq V \cup wfvarsoccs_{st}\ S \rrbracket \implies P\ (S@[Receive\ t])$ "
 " $\bigwedge t\ t'\ S. \llbracket wf_{st}\ V\ S; P\ S; fv\ t' \subseteq V \cup wfvarsoccs_{st}\ S \rrbracket \implies P\ (S@[Equality\ Assign\ t\ t'])$ "
 " $\bigwedge t\ t'\ S. \llbracket wf_{st}\ V\ S; P\ S \rrbracket \implies P\ (S@[Equality\ Check\ t\ t'])$ "
 " $\bigwedge X\ F\ S. \llbracket wf_{st}\ V\ S; P\ S \rrbracket \implies P\ (S@[Inequality\ X\ F])$ "
 shows " $P\ S$ "
 <proof>

lemma `wf_subst_apply`:
 " $wf_{st}\ V\ S \implies wf_{st}\ (fv_{set}\ (\delta \setminus V))\ (S \cdot_{st}\ \delta)$ "
 <proof>

lemma `wf_unify`:
 assumes `wf`: " $wf_{st}\ V\ (S@Send\ (Fun\ f\ X)\#S')$ "
 and `g_in_ik`: " $t \in ik_{st}\ S$ "
 and `δ`: " $Unifier\ \delta\ (Fun\ f\ X)\ t$ "
 and `disj`: " $bvars_{st}\ (S@Send\ (Fun\ f\ X)\#S') \cap (subst_domain\ \delta \cup range_vars\ \delta) = \{\}$ "
 shows " $wf_{st}\ (fv_{set}\ (\delta \setminus V))\ ((S@S') \cdot_{st}\ \delta)$ "
 <proof>

lemma `wf_equality`:
 assumes `wf`: " $wf_{st}\ V\ (S@Equality\ ac\ t\ t'\#S')$ "
 and `δ`: " $mgu\ t\ t' = Some\ \delta$ "
 and `disj`: " $bvars_{st}\ (S@Equality\ ac\ t\ t'\#S') \cap (subst_domain\ \delta \cup range_vars\ \delta) = \{\}$ "
 shows " $wf_{st}\ (fv_{set}\ (\delta \setminus V))\ ((S@S') \cdot_{st}\ \delta)$ "
 <proof>

lemma `wf_rcv_prefix_ground`:
 " $wf_{st}\ \{\}\ ((map\ Receive\ M)@S) \implies vars_{st}\ (map\ Receive\ M) = \{\}$ "
 <proof>

lemma `simple_wfvarsoccs_st_is_fv_snd`:
 assumes " $simple\ S$ "
 shows " $wfvarsoccs_{st}\ S = \bigcup (set\ (map\ fv_{snd}\ S))$ "
 <proof>


```

lemma wf_st_simple_induct[consumes 2, case_names Nil ConsSnd ConsRcv ConsIneq]:
  fixes S: "('a, 'b) strand"
  assumes "wf_st V S" "simple S"
          "P []"
          " $\bigwedge v S. \llbracket wf\_st\ V\ S; simple\ S; P\ S \rrbracket \implies P (S@[Send\ (Var\ v)])$ "
          " $\bigwedge t S. \llbracket wf\_st\ V\ S; simple\ S; P\ S; fv\ t \subseteq V \cup \bigcup (set\ (map\ fv_{snd}\ S)) \rrbracket \implies P (S@[Receive\ t])$ "
          " $\bigwedge X\ F\ S. \llbracket wf\_st\ V\ S; simple\ S; P\ S \rrbracket \implies P (S@[Inequality\ X\ F])$ "
  shows "P S"
<proof>

lemma wf_trm_stp_dom_fv_disjoint:
  " $\llbracket wf_{constr}\ S\ \vartheta; t \in trms_{st}\ S \rrbracket \implies subst\_domain\ \vartheta \cap fv\ t = \{\}$ "
<proof>

lemma wf_constr_bvars_disj: "wf_constr S  $\vartheta \implies (subst\_domain\ \vartheta \cup range\_vars\ \vartheta) \cap bvars_{st}\ S = \{\}$ "
<proof>

lemma wf_constr_bvars_disj':
  assumes "wf_constr S  $\vartheta$ " "subst_domain  $\delta \cup range\_vars\ \delta \subseteq fv_{st}\ S$ "
  shows "(subst_domain  $\delta \cup range\_vars\ \delta) \cap bvars_{st}\ S = \{\}$ " (is ?A)
  and "(subst_domain  $\vartheta \cup range\_vars\ \vartheta) \cap bvars_{st}\ (S \cdot_{st}\ \delta) = \{\}$ " (is ?B)
<proof>

lemma (in intruder_model) wf_simple_strand_first_Send_var_split:
  assumes "wf_st  $\{\}$  S" "simple S" " $\exists v \in wf\_restrictedvars_{st}\ S. t \cdot \mathcal{I} = \mathcal{I}\ v$ "
  shows " $\exists v\ S_{pre}\ S_{suf}. S = S_{pre}@Send\ (Var\ v)#S_{suf} \wedge t \cdot \mathcal{I} = \mathcal{I}\ v$ "
          " $\wedge \neg(\exists w \in wf\_restrictedvars_{st}\ S_{pre}. t \cdot \mathcal{I} = \mathcal{I}\ w)$ "
  (is "?P S")
<proof>

lemma (in intruder_model) wf_strand_first_Send_var_split:
  assumes "wf_st  $\{\}$  S" " $\forall v \in wf\_restrictedvars_{st}\ S. t \cdot \mathcal{I} \sqsubseteq \mathcal{I}\ v$ "
  shows " $\exists S_{pre}\ S_{suf}. \neg(\exists w \in wf\_restrictedvars_{st}\ S_{pre}. t \cdot \mathcal{I} \sqsubseteq \mathcal{I}\ w)$ "
          " $\wedge ((\exists t'. S = S_{pre}@Send\ t'#S_{suf} \wedge t \cdot \mathcal{I} \sqsubseteq t' \cdot \mathcal{I})$ "
          " $\vee (\exists t'\ t''. S = S_{pre}@Equality\ Assign\ t'\ t''#S_{suf} \wedge t \cdot \mathcal{I} \sqsubseteq t' \cdot \mathcal{I}))$ "
  (is " $\exists S_{pre}\ S_{suf}. ?P\ S_{pre} \wedge ?Q\ S\ S_{pre}\ S_{suf}$ ")
<proof>

```

3.1.6 Constraint Semantics

```

context intruder_model
begin

```

Definitions

The constraint semantics in which the intruder is limited to composition only

```

fun strand_sem_c: "('fun, 'var) terms  $\Rightarrow$  ('fun, 'var) strand  $\Rightarrow$  ('fun, 'var) subst  $\Rightarrow$  bool" (" $\llbracket \_ \rrbracket_c$ ")
where
  " $\llbracket M; [] \rrbracket_c = (\lambda \mathcal{I}. True)$ "
  | " $\llbracket M; Send\ t\ \#S \rrbracket_c = (\lambda \mathcal{I}. M \vdash_c t \cdot \mathcal{I} \wedge \llbracket M; S \rrbracket_c \mathcal{I})$ "
  | " $\llbracket M; Receive\ t\ \#S \rrbracket_c = (\lambda \mathcal{I}. \llbracket insert\ (t \cdot \mathcal{I})\ M; S \rrbracket_c \mathcal{I})$ "
  | " $\llbracket M; Equality\ \_ \ t\ t'\ \#S \rrbracket_c = (\lambda \mathcal{I}. t \cdot \mathcal{I} = t' \cdot \mathcal{I} \wedge \llbracket M; S \rrbracket_c \mathcal{I})$ "
  | " $\llbracket M; Inequality\ X\ F\ \#S \rrbracket_c = (\lambda \mathcal{I}. ineq\_model\ \mathcal{I}\ X\ F \wedge \llbracket M; S \rrbracket_c \mathcal{I})$ "

```

```

definition constr_sem_c (" $\_ \models_c \langle \_, \_ \rangle$ ") where " $\mathcal{I} \models_c \langle S, \vartheta \rangle \equiv (\vartheta\ supports\ \mathcal{I} \wedge \llbracket \{\}; S \rrbracket_c \mathcal{I})$ "
abbreviation constr_sem_c' (" $\_ \models_c \langle \_ \rangle$ " 90) where " $\mathcal{I} \models_c \langle S \rangle \equiv \mathcal{I} \models_c \langle S, Var \rangle$ "

```

The full constraint semantics

```

fun strand_sem_d: "('fun, 'var) terms  $\Rightarrow$  ('fun, 'var) strand  $\Rightarrow$  ('fun, 'var) subst  $\Rightarrow$  bool" (" $\llbracket \_ \rrbracket_d$ ")
where
  " $\llbracket M; [] \rrbracket_d = (\lambda \mathcal{I}. True)$ "

```

3 The Typing Result for Non-Stateful Protocols

```

/ "[M; Send t#S]_d = (λI. M ⊢ t · I ∧ [M; S]_d I)"
/ "[M; Receive t#S]_d = (λI. [insert (t · I) M; S]_d I)"
/ "[M; Equality _ t t'#S]_d = (λI. t · I = t' · I ∧ [M; S]_d I)"
/ "[M; Inequality X F#S]_d = (λI. ineq_model I X F ∧ [M; S]_d I)"

```

definition `constr_sem_d` ("_ ⊢ ⟨_,_⟩") **where** "I ⊢ ⟨S,ϑ⟩ ≡ (ϑ supports I ∧ [{}; S]_d I)"
abbreviation `constr_sem_d'` ("_ ⊢ ⟨_⟩" 90) **where** "I ⊢ ⟨S⟩ ≡ I ⊢ ⟨S,Var⟩"

`lemmas strand_sem_induct = strand_sem_c.induct[case_names Nil ConsSnd ConsRcv ConsEq ConsIneq]`

Lemmata

lemma `strand_sem_d_if_c`: "I ⊢_c ⟨S,ϑ⟩ ⇒ I ⊢ ⟨S,ϑ⟩"
 ⟨proof⟩

lemma `strand_sem_mono_ik`:
 "[M ⊆ M'; [M; S]_c ϑ] ⇒ [M'; S]_c ϑ" (is "[?A'; ?A'] ⇒ ?A")
 "[M ⊆ M'; [M; S]_d ϑ] ⇒ [M'; S]_d ϑ" (is "[?B'; ?B'] ⇒ ?B")
 ⟨proof⟩

context

begin

private lemma `strand_sem_split_left`:

```

"[M; S@S']_c ϑ ⇒ [M; S]_c ϑ"
"[M; S@S']_d ϑ ⇒ [M; S]_d ϑ"

```

⟨proof⟩ **lemma** `strand_sem_split_right`:

```

"[M; S@S']_c ϑ ⇒ [M ∪ (ik_st S ·_set ϑ); S']_c ϑ"
"[M; S@S']_d ϑ ⇒ [M ∪ (ik_st S ·_set ϑ); S']_d ϑ"

```

⟨proof⟩

lemmas `strand_sem_split[dest]` =

```

strand_sem_split_left(1) strand_sem_split_right(1)
strand_sem_split_left(2) strand_sem_split_right(2)

```

end

lemma `strand_sem_Send_split[dest]`:

```

"[ [M; map Send T]_c ϑ; t ∈ set T ] ⇒ [M; [Send t]_c ϑ" (is "[?A'; ?A'] ⇒ ?A")
"[ [M; map Send T]_d ϑ; t ∈ set T ] ⇒ [M; [Send t]_d ϑ" (is "[?B'; ?B'] ⇒ ?B")
"[ [M; map Send T@S]_c ϑ; t ∈ set T ] ⇒ [M; Send t#S]_c ϑ" (is "[?C'; ?C'] ⇒ ?C")
"[ [M; map Send T@S]_d ϑ; t ∈ set T ] ⇒ [M; Send t#S]_d ϑ" (is "[?D'; ?D'] ⇒ ?D")

```

⟨proof⟩

lemma `strand_sem_Send_map`:

```

"(∧t. t ∈ set T ⇒ [M; [Send t]_c I) ⇒ [M; map Send T]_c I"
"(∧t. t ∈ set T ⇒ [M; [Send t]_d I) ⇒ [M; map Send T]_d I"

```

⟨proof⟩

lemma `strand_sem_Receive_map`: "[M; map Receive T]_c I" "[M; map Receive T]_d I"

⟨proof⟩

lemma `strand_sem_append[intro]`:

```

"[ [M; S]_c ϑ; [M ∪ (ik_st S ·_set ϑ); S']_c ϑ ] ⇒ [M; S@S']_c ϑ"
"[ [M; S]_d ϑ; [M ∪ (ik_st S ·_set ϑ); S']_d ϑ ] ⇒ [M; S@S']_d ϑ"

```

⟨proof⟩

lemma `ineq_model_subst`:

```

fixes F::"((('a,'b) term × ('a,'b) term) list"
assumes "(subst_domain δ ∪ range_vars δ) ∩ set X = {}"
and "ineq_model (δ ∘_s ϑ) X F"
shows "ineq_model ϑ X (F ·_pairs δ)"

```

⟨proof⟩

lemma `ineq_model_subst'`:

```

fixes F::"(('a,'b) term × ('a,'b) term) list"
assumes "(subst_domain δ ∪ range_vars δ) ∩ set X = {}"
and "ineq_model ϑ X (F ·pairs δ)"
shows "ineq_model (δ ∘s ϑ) X F"
⟨proof⟩

lemma ineq_model_ground_subst:
fixes F::"(('a,'b) term × ('a,'b) term) list"
assumes "fvpairs F - set X ⊆ subst_domain δ"
and "ground (subst_range δ)"
and "ineq_model δ X F"
shows "ineq_model (δ ∘s ϑ) X F"
⟨proof⟩

context
begin
private lemma strand_sem_subst_c:
assumes "(subst_domain δ ∪ range_vars δ) ∩ bvarsst S = {}"
shows "⟦M; S⟧c (δ ∘s ϑ) ⟹ ⟦M; S ·st δ⟧c ϑ"
⟨proof⟩ lemma strand_sem_subst_c':
assumes "(subst_domain δ ∪ range_vars δ) ∩ bvarsst S = {}"
shows "⟦M; S ·st δ⟧c ϑ ⟹ ⟦M; S⟧c (δ ∘s ϑ)"
⟨proof⟩ lemma strand_sem_subst_d:
assumes "(subst_domain δ ∪ range_vars δ) ∩ bvarsst S = {}"
shows "⟦M; S⟧d (δ ∘s ϑ) ⟹ ⟦M; S ·st δ⟧d ϑ"
⟨proof⟩ lemma strand_sem_subst_d':
assumes "(subst_domain δ ∪ range_vars δ) ∩ bvarsst S = {}"
shows "⟦M; S ·st δ⟧d ϑ ⟹ ⟦M; S⟧d (δ ∘s ϑ)"
⟨proof⟩

lemmas strand_sem_subst =
strand_sem_subst_c strand_sem_subst_c' strand_sem_subst_d strand_sem_subst_d'
end

lemma strand_sem_subst_subst_idem:
assumes δ: "(subst_domain δ ∪ range_vars δ) ∩ bvarsst S = {}"
shows "⟦⟦M; S ·st δ⟧c (δ ∘s ϑ); subst_idem δ⟧ ⟹ ⟦M; S⟧c (δ ∘s ϑ)"
⟨proof⟩

lemma strand_sem_subst_comp:
assumes "(subst_domain ϑ ∪ range_vars ϑ) ∩ bvarsst S = {}"
and "⟦M; S⟧c δ" "subst_domain ϑ ∩ (varsst S ∪ fvset M) = {}"
shows "⟦M; S⟧c (ϑ ∘s δ)"
⟨proof⟩

lemma strand_sem_c_imp_ineqs_neq:
assumes "⟦M; S⟧c I" "Inequality X [(t,t')] ∈ set S"
shows "t ≠ t' ∧ (∀δ. subst_domain δ = set X ∧ ground (subst_range δ)
→ t · δ ≠ t' · δ ∧ t · δ · I ≠ t' · δ · I)"
⟨proof⟩

lemma strand_sem_c_imp_ineq_model:
assumes "⟦M; S⟧c I" "Inequality X F ∈ set S"
shows "ineq_model I X F"
⟨proof⟩

lemma strand_sem_wf_simple_fv_sat:
assumes "wfst {} S" "simple S" "⟦{}; S⟧c I"
shows "∧v. v ∈ wfrestrictedvarsst S ⟹ ikst S ·set I ⊢c I v"
⟨proof⟩

lemma strand_sem_wf_ik_or_assignment_rhs_fun_subterm:
assumes "wfst {} A" "⟦{}; A⟧c I" "Var x ∈ ikst A" "I x = Fun f T"

```

3 The Typing Result for Non-Stateful Protocols

```

      "ti ∈ set T" "¬ikst A ·set I ⊢c ti" "interpretationsubst I"
obtains S where
  "Fun f S ∈ subtermsset (ikst A) ∨ Fun f S ∈ subtermsset (assignment_rhsst A)"
  "Fun f T = Fun f S · I"
⟨proof⟩

lemma strand_sem_not_unif_is_sat_ineq:
  assumes "∄∅. Unifier ∅ t t'"
  shows "[M; [Inequality X [(t,t')]]]c I" "[M; [Inequality X [(t,t')]]]d I"
⟨proof⟩

lemma ineq_model_singleI[intro]:
  assumes "∀δ. subst_domain δ = set X ∧ ground (subst_range δ) ⟶ t · δ · I ≠ t' · δ · I"
  shows "ineq_model I X [(t,t')]"
⟨proof⟩

lemma ineq_model_singleE:
  assumes "ineq_model I X [(t,t')]"
  shows "∀δ. subst_domain δ = set X ∧ ground (subst_range δ) ⟶ t · δ · I ≠ t' · δ · I"
⟨proof⟩

lemma ineq_model_single_iff:
  fixes F::("('a,'b) term × ('a,'b) term) list"
  shows "ineq_model I X F ⟷
    ineq_model I X [(Fun f (Fun c []#map fst F), Fun f (Fun c []#map snd F))]"
  (is "?A ⟷ ?B")
⟨proof⟩

```

3.1.7 Constraint Semantics (Alternative, Equivalent Version)

These are the constraint semantics used in the CSF 2017 paper

```

fun strand_sem_c::("fun,'var) terms ⇒ ('fun,'var) strand ⇒ ('fun,'var) subst ⇒ bool" ("[_;
_]_c'")
  where
    "[M; []]c' = (λI. True)"
  | "[M; Send t#S]c' = (λI. M ·set I ⊢c t · I ∧ [M; S]c' I)"
  | "[M; Receive t#S]c' = [insert t M; S]c'"
  | "[M; Equality _ t t'#S]c' = (λI. t · I = t' · I ∧ [M; S]c' I)"
  | "[M; Inequality X F#S]c' = (λI. ineq_model I X F ∧ [M; S]c' I)"

fun strand_sem_d::("fun,'var) terms ⇒ ('fun,'var) strand ⇒ ('fun,'var) subst ⇒ bool" ("[_;
_]_d'")
  where
    "[M; []]d' = (λI. True)"
  | "[M; Send t#S]d' = (λI. M ·set I ⊢ t · I ∧ [M; S]d' I)"
  | "[M; Receive t#S]d' = [insert t M; S]d'"
  | "[M; Equality _ t t'#S]d' = (λI. t · I = t' · I ∧ [M; S]d' I)"
  | "[M; Inequality X F#S]d' = (λI. ineq_model I X F ∧ [M; S]d' I)"

lemma strand_sem_eq_defs:
  "[M; A]c' I = [M ·set I; A]c I"
  "[M; A]d' I = [M ·set I; A]d I"
⟨proof⟩

lemma strand_sem_split'[dest]:
  "[M; S@S']c' ∅ ⟹ [M; S]c' ∅"
  "[M; S@S']c' ∅ ⟹ [M ∪ ikst S; S']c' ∅"
  "[M; S@S']d' ∅ ⟹ [M; S]d' ∅"
  "[M; S@S']d' ∅ ⟹ [M ∪ ikst S; S']d' ∅"
⟨proof⟩

lemma strand_sem_append'[intro]:
  "[M; S]c' ∅ ⟹ [M ∪ ikst S; S']c' ∅ ⟹ [M; S@S']c' ∅"

```

```

  "[M; S]_d' ∅ ⇒ [M ∪ ikst S; S']_d' ∅ ⇒ [M; S@S']_d' ∅"
  <proof>
end

```

3.1.8 Dual Strands

```

fun dualst::('a,'b) strand ⇒ ('a,'b) strand" where
  "dualst [] = []"
| "dualst (Receive t#S) = Send t#(dualst S)"
| "dualst (Send t#S) = Receive t#(dualst S)"
| "dualst (x#S) = x#(dualst S)"

lemma dualst_append: "dualst (A@B) = (dualst A)@(dualst B)"
  <proof>

lemma dualst_self_inverse: "dualst (dualst S) = S"
  <proof>

lemma dualst_trms_eq: "trmsst (dualst S) = trmsst S"
  <proof>

lemma dualst_fv: "fvst (dualst A) = fvst A"
  <proof>

lemma dualst_bvars: "bvarsst (dualst A) = bvarsst A"
  <proof>

end

```

3.2 The Lazy Intruder (Lazy_Intruder)

```

theory Lazy_Intruder
imports Strands_and_Constraints Intruder_Deduction
begin

context intruder_model
begin

```

3.2.1 Definition of the Lazy Intruder

The lazy intruder constraint reduction system, defined as a relation on constraint states

```

inductive_set LI_rel::
  "(((('fun,'var) strand × (('fun,'var) subst)) ×
    (('fun,'var) strand × (('fun,'var) subst)) set"
  and LI_rel' (infix "↔" 50)
  and LI_rel_trancl (infix "↔+" 50)
  and LI_rel_rtrancl (infix "↔*" 50)
where
  "A ↔ B ≡ (A,B) ∈ LI_rel"
| "A ↔+ B ≡ (A,B) ∈ LI_rel+"
| "A ↔* B ≡ (A,B) ∈ LI_rel*"

| Compose: "[simple S; length T = arity f; public f]
  ⇒ (S@Send (Fun f T)#S',∅) ↔ (S@(map Send T)@S',∅)"
| Unify: "[simple S; Fun f T' ∈ ikst S; Some δ = mgu (Fun f T) (Fun f T')]"
  ⇒ (S@Send (Fun f T)#S',∅) ↔ ((S@S') ·st δ,∅ ∘s δ)"
| Equality: "[simple S; Some δ = mgu t t']"
  ⇒ (S@Equality _ t t'#S',∅) ↔ ((S@S') ·st δ,∅ ∘s δ)"

```

3.2.2 Lemma: The Lazy Intruder is Well-founded

```

context
begin
private lemma LI_compose_measure_lt: " $((S@(\text{map Send } T)@S', \vartheta_1), (S@Send (\text{Fun } f \text{ } T)\#S', \vartheta_2)) \in \text{measure}_{st}$ "
<proof> lemma LI_unify_measure_lt:
  assumes "Some  $\delta = \text{mgu } (\text{Fun } f \text{ } T) \text{ } t$ " "fv  $t \subseteq \text{fv}_{st} S$ "
  shows " $((S@S') \cdot_{st} \delta, \vartheta_1), (S@Send (\text{Fun } f \text{ } T)\#S', \vartheta_2) \in \text{measure}_{st}$ "
<proof> lemma LI_equality_measure_lt:
  assumes "Some  $\delta = \text{mgu } t \text{ } t'$ "
  shows " $((S@S') \cdot_{st} \delta, \vartheta_1), (S@Equality a \text{ } t \text{ } t'\#S', \vartheta_2) \in \text{measure}_{st}$ "
<proof> lemma LI_in_measure: " $(S_1, \vartheta_1) \rightsquigarrow (S_2, \vartheta_2) \implies ((S_2, \vartheta_2), (S_1, \vartheta_1)) \in \text{measure}_{st}$ "
<proof> lemma LI_in_measure_trans: " $(S_1, \vartheta_1) \rightsquigarrow^+ (S_2, \vartheta_2) \implies ((S_2, \vartheta_2), (S_1, \vartheta_1)) \in \text{measure}_{st}$ "
<proof> lemma LI_converse_wellfounded_trans: "wf  $((LI\_rel^+)^{-1})$ "
<proof> lemma LI_acyclic_trans: "acyclic  $(LI\_rel^+)$ "
<proof> lemma LI_acyclic: "acyclic  $LI\_rel$ "
<proof>

lemma LI_no_infinite_chain: " $\neg(\exists f. \forall i. f \text{ } i \rightsquigarrow^+ f (\text{Suc } i))$ "
<proof> lemma LI_unify_finite:
  assumes "finite  $M$ "
  shows "finite  $\{(S@Send (\text{Fun } f \text{ } T)\#S', \vartheta), ((S@S') \cdot_{st} \delta, \vartheta \circ_s \delta) \mid \delta \text{ } T'. \text{simple } S \wedge \text{Fun } f \text{ } T' \in M \wedge \text{Some } \delta = \text{mgu } (\text{Fun } f \text{ } T) (\text{Fun } f \text{ } T')\}$ "
<proof>
end

```

3.2.3 Lemma: The Lazy Intruder Preserves Well-formedness

```

context
begin
private lemma LI_preserves_subst_wf_single:
  assumes " $(S_1, \vartheta_1) \rightsquigarrow (S_2, \vartheta_2)$ " "fvst  $S_1 \cap \text{bvars}_{st} S_1 = \{\}$ " "wfsubst  $\vartheta_1$ "
  and "subst_domain  $\vartheta_1 \cap \text{vars}_{st} S_1 = \{\}$ " "range_vars  $\vartheta_1 \cap \text{bvars}_{st} S_1 = \{\}$ "
  shows "fvst  $S_2 \cap \text{bvars}_{st} S_2 = \{\}$ " "wfsubst  $\vartheta_2$ "
  and "subst_domain  $\vartheta_2 \cap \text{vars}_{st} S_2 = \{\}$ " "range_vars  $\vartheta_2 \cap \text{bvars}_{st} S_2 = \{\}$ "
<proof> lemma LI_preserves_subst_wf:
  assumes " $(S_1, \vartheta_1) \rightsquigarrow^* (S_2, \vartheta_2)$ " "fvst  $S_1 \cap \text{bvars}_{st} S_1 = \{\}$ " "wfsubst  $\vartheta_1$ "
  and "subst_domain  $\vartheta_1 \cap \text{vars}_{st} S_1 = \{\}$ " "range_vars  $\vartheta_1 \cap \text{bvars}_{st} S_1 = \{\}$ "
  shows "fvst  $S_2 \cap \text{bvars}_{st} S_2 = \{\}$ " "wfsubst  $\vartheta_2$ "
  and "subst_domain  $\vartheta_2 \cap \text{vars}_{st} S_2 = \{\}$ " "range_vars  $\vartheta_2 \cap \text{bvars}_{st} S_2 = \{\}$ "
<proof>

lemma LI_preserves_wellformedness:
  assumes " $(S_1, \vartheta_1) \rightsquigarrow^* (S_2, \vartheta_2)$ " "wfconstr  $S_1 \text{ } \vartheta_1$ "
  shows "wfconstr  $S_2 \text{ } \vartheta_2$ "
<proof>

lemma LI_preserves_trm_wf:
  assumes " $(S, \vartheta) \rightsquigarrow^* (S', \vartheta')$ " "wftrms  $(\text{trms}_{st} S)$ "
  shows "wftrms  $(\text{trms}_{st} S')$ "
<proof>
end

```

3.2.4 Theorem: Soundness of the Lazy Intruder

```

context
begin
private lemma LI_soundness_single:
  assumes "wfconstr  $S_1 \text{ } \vartheta_1$ " " $(S_1, \vartheta_1) \rightsquigarrow (S_2, \vartheta_2)$ " " $\mathcal{I} \models_c \langle S_2, \vartheta_2 \rangle$ "
  shows " $\mathcal{I} \models_c \langle S_1, \vartheta_1 \rangle$ "
<proof>

theorem LI_soundness:

```

```

  assumes "wf_constr S1 ϑ1" "(S1, ϑ1) ~* (S2, ϑ2)" "I ⊨c ⟨S2, ϑ2⟩"
  shows "I ⊨c ⟨S1, ϑ1⟩"
⟨proof⟩
end

```

3.2.5 Theorem: Completeness of the Lazy Intruder

```

context
begin
private lemma LI_completeness_single:
  assumes "wf_constr S1 ϑ1" "I ⊨c ⟨S1, ϑ1⟩" "¬simple S1"
  shows "∃ S2 ϑ2. (S1, ϑ1) ~ (S2, ϑ2) ∧ (I ⊨c ⟨S2, ϑ2⟩)"
⟨proof⟩

theorem LI_completeness:
  assumes "wf_constr S1 ϑ1" "I ⊨c ⟨S1, ϑ1⟩"
  shows "∃ S2 ϑ2. (S1, ϑ1) ~* (S2, ϑ2) ∧ simple S2 ∧ (I ⊨c ⟨S2, ϑ2⟩)"
⟨proof⟩
end

```

3.2.6 Corollary: Soundness and Completeness as a Single Theorem

```

corollary LI_soundness_and_completeness:
  assumes "wf_constr S1 ϑ1"
  shows "I ⊨c ⟨S1, ϑ1⟩ ↔ (∃ S2 ϑ2. (S1, ϑ1) ~* (S2, ϑ2) ∧ simple S2 ∧ (I ⊨c ⟨S2, ϑ2⟩))"
⟨proof⟩

end

end

```

3.3 The Typed Model (Typed_Model)

```

theory Typed_Model
imports Lazy_Intruder
begin

  Term types

type_synonym ('f, 'v) term_type = "('f, 'v) term"

  Constructors for term types

abbreviation (input) TAtom::"'v ⇒ ('f, 'v) term_type" where
  "TAtom a ≡ Var a"

abbreviation (input) TComp::"['f, ('f, 'v) term_type list] ⇒ ('f, 'v) term_type" where
  "TComp f T ≡ Fun f T"

```

The typed model extends the intruder model with a typing function Γ that assigns types to terms.

```

locale typed_model = intruder_model arity public Ana
  for arity::"fun ⇒ nat"
  and public::"fun ⇒ bool"
  and Ana::"('fun, 'var) term ⇒ (('fun, 'var) term list × ('fun, 'var) term list)"
+
  fixes Γ::"('fun, 'var) term ⇒ ('fun, 'atom::finite) term_type"
  assumes const_type: "∧c. arity c = 0 ⇒ ∃a. ∀T. Γ (Fun c T) = TAtom a"
  and fun_type: "∧f T. arity f > 0 ⇒ Γ (Fun f T) = TComp f (map Γ T)"
  and infinite_typed_consts: "∧a. infinite {c. Γ (Fun c []) = TAtom a ∧ public c}"
  and Γ_wf: "∧t f T. TComp f T ⊆ Γ t ⇒ arity f > 0"
  "∧x. wfterm (Γ (Var x))"
  and no_private_funs[simp]: "∧f. arity f > 0 ⇒ public f"
begin

```

3.3.1 Definitions

The set of atomic types

abbreviation $\overline{\mathcal{A}}_a \equiv UNIV::('atom\ set)$ "

Well-typed substitutions

definition $w_{t_{subst}}$ **where**

$"w_{t_{subst}}\ \sigma \equiv (\forall v. \Gamma\ (Var\ v) = \Gamma\ (\sigma\ v))"$

The set of sub-message patterns (SMP)

inductive_set $SMP::('fun, 'var)\ terms \Rightarrow ('fun, 'var)\ terms$ for M **where**

$MP[intro]: "t \in M \Longrightarrow t \in SMP\ M"$

$| Subterm[intro]: "[t \in SMP\ M; t' \sqsubseteq t] \Longrightarrow t' \in SMP\ M"$

$| Substitution[intro]: "[t \in SMP\ M; w_{t_{subst}}\ \delta; wf_{trms}\ (subst_range\ \delta)] \Longrightarrow (t \cdot \delta) \in SMP\ M"$

$| Ana[intro]: "[t \in SMP\ M; Ana\ t = (K, T); k \in set\ K] \Longrightarrow k \in SMP\ M"$

Type-flaw resistance for sets: Unifiable sub-message patterns must have the same type (unless they are variables)

definition tfr_{set} **where**

$"tfr_{set}\ M \equiv (\forall s \in SMP\ M - (Var\ \backslash \mathcal{V}). \forall t \in SMP\ M - (Var\ \backslash \mathcal{V}). (\exists \delta. Unifier\ \delta\ s\ t) \longrightarrow \Gamma\ s = \Gamma\ t)"$

Type-flaw resistance for strand steps: - The terms in a satisfiable equality step must have the same types - Inequality steps must satisfy the conditions of the "inequality lemma"

fun tfr_{stp} **where**

$"tfr_{stp}\ (Equality\ a\ t\ t') = ((\exists \delta. Unifier\ \delta\ t\ t') \longrightarrow \Gamma\ t = \Gamma\ t)"$

$| "tfr_{stp}\ (Inequality\ X\ F) = ($

$(\forall x \in fv_{pairs}\ F - set\ X. \exists a. \Gamma\ (Var\ x) = TAtom\ a) \vee$

$(\forall f\ T. Fun\ f\ T \in subterms_{set}\ (trms_{pairs}\ F) \longrightarrow T = [] \vee (\exists s \in set\ T. s \notin Var\ \backslash set\ X)))"$

$| "tfr_{stp}\ _ = True"$

Type-flaw resistance for strands: - The set of terms in strands must be type-flaw resistant - The steps of strands must be type-flaw resistant

definition tfr_{st} **where**

$"tfr_{st}\ S \equiv tfr_{set}\ (trms_{st}\ S) \wedge list_all\ tfr_{stp}\ S"$

3.3.2 Small Lemmata

lemma $tfr_{stp_list_all_alt_def}$:

$"list_all\ tfr_{stp}\ S \longleftrightarrow$

$((\forall a\ t\ t'. Equality\ a\ t\ t' \in set\ S \wedge (\exists \delta. Unifier\ \delta\ t\ t') \longrightarrow \Gamma\ t = \Gamma\ t') \wedge$

$(\forall X\ F. Inequality\ X\ F \in set\ S \longrightarrow$

$(\forall x \in fv_{pairs}\ F - set\ X. \exists a. \Gamma\ (Var\ x) = TAtom\ a)$

$\vee (\forall f\ T. Fun\ f\ T \in subterms_{set}\ (trms_{pairs}\ F) \longrightarrow T = [] \vee (\exists s \in set\ T. s \notin Var\ \backslash set\ X))))"$

$(is\ "?P\ S \longleftrightarrow ?Q\ S")$

$\langle proof \rangle$

lemma Γ_wf' : $"wf_{trm}\ t \Longrightarrow wf_{trm}\ (\Gamma\ t)"$

$\langle proof \rangle$

lemma fun_type_inv : **assumes** $"\Gamma\ t = TComp\ f\ T"$ **shows** $"arity\ f > 0"$ $"public\ f"$

$\langle proof \rangle$

lemma $fun_type_inv_wf$: **assumes** $"\Gamma\ t = TComp\ f\ T"$ $"wf_{trm}\ t"$ **shows** $"arity\ f = length\ T"$

$\langle proof \rangle$

lemma $const_type_inv$: $"\Gamma\ (Fun\ c\ X) = TAtom\ a \Longrightarrow arity\ c = 0"$

$\langle proof \rangle$

lemma $const_type_inv_wf$: **assumes** $"\Gamma\ (Fun\ c\ X) = TAtom\ a"$ **and** $"wf_{trm}\ (Fun\ c\ X)"$ **shows** $"X = []"$

$\langle proof \rangle$


```

lemma const_type': "∀ c ∈ C. ∃ a ∈  $\mathfrak{A}_a$ . ∀ X.  $\Gamma$  (Fun c X) = TAtom a" <proof>
lemma fun_type': "∀ f ∈  $\Sigma_f$ . ∀ X.  $\Gamma$  (Fun f X) = TComp f (map  $\Gamma$  X)" <proof>

lemma infinite_public_consts[simp]: "infinite {c. public c ∧ arity c = 0}"
<proof>

lemma infinite_fun_syms[simp]:
  "infinite {c. public c ∧ arity c > 0} ⇒ infinite  $\Sigma_f$ "
  "infinite C" "infinite  $C_{pub}$ " "infinite (UNIV::'fun set)"
<proof>

lemma id_univ_proper_subset[simp]: " $\Sigma_f \subset UNIV$ " "(∃ f. arity f > 0) ⇒ C ⊂ UNIV"
<proof>

lemma exists_fun_notin_funs_term: "∃ f::'fun. f ∉ funs_term t"
<proof>

lemma exists_fun_notin_funs_terms:
  assumes "finite M" shows "∃ f::'fun. f ∉ ⋃ (funs_term ` M)"
<proof>

lemma exists_notin_funs_st: "∃ f. f ∉ funs_st (S::('fun,'var) strand)"
<proof>

lemma infinite_typed_consts': "infinite {c.  $\Gamma$  (Fun c []) = TAtom a ∧ public c ∧ arity c = 0}"
<proof>

lemma atypes_inhabited: "∃ c.  $\Gamma$  (Fun c []) = TAtom a ∧ wftrm (Fun c []) ∧ public c ∧ arity c = 0"
<proof>

lemma atype_ground_term_ex: "∃ t. fv t = {} ∧  $\Gamma$  t = TAtom a ∧ wftrm t"
<proof>

lemma fun_type_id_eq: " $\Gamma$  (Fun f X) = TComp g Y ⇒ f = g"
<proof>

lemma fun_type_length_eq: " $\Gamma$  (Fun f X) = TComp g Y ⇒ length X = length Y"
<proof>

lemma type_ground_inhabited: "∃ t'. fv t' = {} ∧  $\Gamma$  t =  $\Gamma$  t'"
<proof>

lemma type_wfttype_inhabited:
  assumes "∧ f T. Fun f T ⊆ τ ⇒ 0 < arity f" "wftrm τ"
  shows "∃ t.  $\Gamma$  t = τ ∧ wftrm t"
<proof>

lemma type_pgwt_inhabited: "wftrm t ⇒ ∃ t'.  $\Gamma$  t =  $\Gamma$  t' ∧ public_ground_wf_term t'"
<proof>

lemma pgwt_type_map:
  assumes "public_ground_wf_term t"
  shows " $\Gamma$  t = TAtom a ⇒ ∃ f. t = Fun f []" " $\Gamma$  t = TComp g Y ⇒ ∃ X. t = Fun g X ∧ map  $\Gamma$  X = Y"
<proof>

lemma wt_subst_Var[simp]: "wtsubst Var" <proof>

lemma wt_subst_trm: "(∧ v. v ∈ fv t ⇒  $\Gamma$  (Var v) =  $\Gamma$  (∅ v)) ⇒  $\Gamma$  t =  $\Gamma$  (t · ∅)"
<proof>

lemma wt_subst_trm': "[wtsubst σ;  $\Gamma$  s =  $\Gamma$  t] ⇒  $\Gamma$  (s · σ) =  $\Gamma$  (t · σ)"
<proof>

```

3 The Typing Result for Non-Stateful Protocols

lemma `wt_subst_trm'`: " $wt_{subst} \sigma \implies \Gamma t = \Gamma (t \cdot \sigma)$ "
 $\langle proof \rangle$

lemma `wt_subst_compose`:
assumes " $wt_{subst} \vartheta$ " " $wt_{subst} \delta$ " **shows** " $wt_{subst} (\vartheta \circ_s \delta)$ "
 $\langle proof \rangle$

lemma `wt_subst_TAtom_Var_cases`:
assumes ϑ : " $wt_{subst} \vartheta$ " " $wf_{trms} (subst_range \vartheta)$ "
and x : " $\Gamma (Var x) = TAtom a$ "
shows " $(\exists y. \vartheta x = Var y) \vee (\exists c. \vartheta x = Fun c [])$ "
 $\langle proof \rangle$

lemma `wt_subst_TAtom_fv`:
assumes ϑ : " $wt_{subst} \vartheta$ " " $\forall x. wf_{trm} (\vartheta x)$ "
and " $\forall x \in fv t - X. \exists a. \Gamma (Var x) = TAtom a$ "
shows " $\forall x \in fv (t \cdot \vartheta) - fv_{set} (\vartheta \backslash X). \exists a. \Gamma (Var x) = TAtom a$ "
 $\langle proof \rangle$

lemma `wt_subst_TAtom_subterms_subst`:
assumes " $wt_{subst} \vartheta$ " " $\forall x \in fv t. \exists a. \Gamma (Var x) = TAtom a$ " " $wf_{trms} (\vartheta \backslash fv t)$ "
shows " $subterms (t \cdot \vartheta) = subterms t \cdot_{set} \vartheta$ "
 $\langle proof \rangle$

lemma `wt_subst_TAtom_subterms_set_subst`:
assumes " $wt_{subst} \vartheta$ " " $\forall x \in fv_{set} M. \exists a. \Gamma (Var x) = TAtom a$ " " $wf_{trms} (\vartheta \backslash fv_{set} M)$ "
shows " $subterms_{set} (M \cdot_{set} \vartheta) = subterms_{set} M \cdot_{set} \vartheta$ "
 $\langle proof \rangle$

lemma `wt_subst_subst_upd`:
assumes " $wt_{subst} \vartheta$ "
and " $\Gamma (Var x) = \Gamma t$ "
shows " $wt_{subst} (\vartheta(x := t))$ "
 $\langle proof \rangle$

lemma `wt_subst_const_fv_type_eq`:
assumes " $\forall x \in fv t. \exists a. \Gamma (Var x) = TAtom a$ "
and δ : " $wt_{subst} \delta$ " " $wf_{trms} (subst_range \delta)$ "
shows " $\forall x \in fv (t \cdot \delta). \exists y \in fv t. \Gamma (Var x) = \Gamma (Var y)$ "
 $\langle proof \rangle$

lemma `TComp_term_cases`:
assumes " $wf_{trm} t$ " " $\Gamma t = TComp f T$ "
shows " $(\exists v. t = Var v) \vee (\exists T'. t = Fun f T' \wedge T = map \Gamma T' \wedge T' \neq [])$ "
 $\langle proof \rangle$

lemma `TAtom_term_cases`:
assumes " $wf_{trm} t$ " " $\Gamma t = TAtom \tau$ "
shows " $(\exists v. t = Var v) \vee (\exists f. t = Fun f [])$ "
 $\langle proof \rangle$

lemma `subtermeq_imp_subtermtypreeq`:
assumes " $wf_{trm} t$ " " $s \sqsubseteq t$ "
shows " $\Gamma s \sqsubseteq \Gamma t$ "
 $\langle proof \rangle$

lemma `subterm_funs_term_in_type`:
assumes " $wf_{trm} t$ " " $Fun f T \sqsubseteq t$ " " $\Gamma (Fun f T) = TComp f (map \Gamma T)$ "
shows " $f \in funs_term (\Gamma t)$ "
 $\langle proof \rangle$

lemma `wt_subst_fv_termtyp_subterm`:
assumes " $x \in fv (\vartheta y)$ "

```

  and "wt_subst  $\vartheta$ "
  and "wf_trm ( $\vartheta$  y)"
  shows " $\Gamma$  (Var x)  $\sqsubseteq$   $\Gamma$  (Var y)"
<proof>

```

```

lemma wt_subst_fv_set_termtype_subterm:
  assumes "x  $\in$  fv_set ( $\vartheta$  ` Y)"
  and "wt_subst  $\vartheta$ "
  and "wf_trms (subst_range  $\vartheta$ )"
  shows " $\exists$  y  $\in$  Y.  $\Gamma$  (Var x)  $\sqsubseteq$   $\Gamma$  (Var y)"
<proof>

```

```

lemma funs_term_type_iff:
  assumes t: "wf_trm t"
  and f: "arity f > 0"
  shows "f  $\in$  funs_term ( $\Gamma$  t)  $\longleftrightarrow$  (f  $\in$  funs_term t  $\vee$  ( $\exists$  x  $\in$  fv t. f  $\in$  funs_term ( $\Gamma$  (Var x))))"
  (is "?P t  $\longleftrightarrow$  ?Q t")
<proof>

```

```

lemma funs_term_type_iff':
  assumes M: "wf_trms M"
  and f: "arity f > 0"
  shows "f  $\in$   $\bigcup$  (funs_term `  $\Gamma$  ` M)  $\longleftrightarrow$ 
  (f  $\in$   $\bigcup$  (funs_term ` M)  $\vee$  ( $\exists$  x  $\in$  fv_set M. f  $\in$  funs_term ( $\Gamma$  (Var x))))" (is "?A  $\longleftrightarrow$  ?B")
<proof>

```

```

lemma Ana_subterm_type:
  assumes "Ana t = (K,M)"
  and "wf_trm t"
  and "m  $\in$  set M"
  shows " $\Gamma$  m  $\sqsubseteq$   $\Gamma$  t"
<proof>

```

```

lemma wf_trm_TAtom_subterms:
  assumes "wf_trm t" " $\Gamma$  t = TAtom  $\tau$ "
  shows "subterms t = {t}"
<proof>

```

```

lemma wf_trm_TComp_subterm:
  assumes "wf_trm s" "t  $\sqsubseteq$  s"
  obtains f T where " $\Gamma$  s = TComp f T"
<proof>

```

```

lemma SMP_empty[simp]: "SMP {} = {}"
<proof>

```

```

lemma SMP_I:
  assumes "s  $\in$  M" "wt_subst  $\delta$ " "t  $\sqsubseteq$  s  $\cdot$   $\delta$ " " $\bigwedge$ v. wf_trm ( $\delta$  v)"
  shows "t  $\in$  SMP M"
<proof>

```

```

lemma SMP_wf_trm:
  assumes "t  $\in$  SMP M" "wf_trms M"
  shows "wf_trm t"
<proof>

```

```

lemma SMP_ikI[intro]: "t  $\in$  ikst S  $\implies$  t  $\in$  SMP (trmsst S)" <proof>

```

```

lemma MP_setI[intro]: "x  $\in$  set S  $\implies$  trmsstp x  $\sqsubseteq$  trmsst S" <proof>

```

```

lemma SMP_setI[intro]: "x  $\in$  set S  $\implies$  trmsstp x  $\sqsubseteq$  SMP (trmsst S)" <proof>

```

```

lemma SMP_subset_I:

```

3 The Typing Result for Non-Stateful Protocols

assumes M : " $\forall t \in M. \exists s \delta. s \in N \wedge wt_{subst} \delta \wedge wf_{trms} (subst_range \delta) \wedge t = s \cdot \delta$ "
shows " $SMP\ M \subseteq SMP\ N$ "
 <proof>

lemma SMP_union : " $SMP\ (A \cup B) = SMP\ A \cup SMP\ B$ "
 <proof>

lemma $SMP_append[simp]$: " $SMP\ (trms_{st}\ (S@S')) = SMP\ (trms_{st}\ S) \cup SMP\ (trms_{st}\ S')$ " (is " $?A = ?B$ ")
 <proof>

lemma SMP_mono : " $A \subseteq B \implies SMP\ A \subseteq SMP\ B$ "
 <proof>

lemma SMP_Union : " $SMP\ (\bigcup_{m \in M}. f\ m) = (\bigcup_{m \in M}. SMP\ (f\ m))$ "
 <proof>

lemma $SMP_singleton_ex$:
 " $t \in SMP\ M \implies (\exists m \in M. t \in SMP\ \{m\})$ "
 " $m \in M \implies t \in SMP\ \{m\} \implies t \in SMP\ M$ "
 <proof>

lemma SMP_Cons : " $SMP\ (trms_{st}\ (x\#S)) = SMP\ (trms_{st}\ [x]) \cup SMP\ (trms_{st}\ S)$ "
 <proof>

lemma $SMP_Nil[simp]$: " $SMP\ (trms_{st}\ []) = \{\}$ "
 <proof>

lemma $SMP_subset_union_eq$: **assumes** " $M \subseteq SMP\ N$ " **shows** " $SMP\ N = SMP\ (M \cup N)$ "
 <proof>

lemma $SMP_subterms_subset$: " $subterms_{set}\ M \subseteq SMP\ M$ "
 <proof>

lemma SMP_SMP_subset : " $N \subseteq SMP\ M \implies SMP\ N \subseteq SMP\ M$ "
 <proof>

lemma $wt_subst_rm_vars$: " $wt_{subst}\ \delta \implies wt_{subst}\ (rm_vars\ X\ \delta)$ "
 <proof>

lemma $wt_subst_SMP_subset$:
assumes " $trms_{st}\ S \subseteq SMP\ S'$ " " $wt_{subst}\ \delta$ " " $wf_{trms}\ (subst_range\ \delta)$ "
shows " $trms_{st}\ (S \cdot_{st}\ \delta) \subseteq SMP\ S'$ "
 <proof>

lemma MP_subset_SMP : " $\bigcup (trms_{stp}\ `set\ S) \subseteq SMP\ (trms_{st}\ S)$ " " $trms_{st}\ S \subseteq SMP\ (trms_{st}\ S)$ " " $M \subseteq SMP\ M$ "
 <proof>

lemma $SMP_fun_map_snd_subset$: " $SMP\ (trms_{st}\ (map\ Send\ X)) \subseteq SMP\ (trms_{st}\ [Send\ (Fun\ f\ X)])$ "
 <proof>

lemma $SMP_wt_subst_subset$:
assumes " $t \in SMP\ (M \cdot_{set}\ I)$ " " $wt_{subst}\ I$ " " $wf_{trms}\ (subst_range\ I)$ "
shows " $t \in SMP\ M$ "
 <proof>

lemma $SMP_wt_instances_subset$:
assumes " $\forall t \in M. \exists s \in N. \exists \delta. t = s \cdot \delta \wedge wt_{subst}\ \delta \wedge wf_{trms}\ (subst_range\ \delta)$ "
and " $t \in SMP\ M$ "
shows " $t \in SMP\ N$ "
 <proof>

lemma SMP_consts :
assumes " $\forall t \in M. \exists c. t = Fun\ c\ []$ "

```

    and "∀t ∈ M. Ana t = ([], [])"
    shows "SMP M = M"
  ⟨proof⟩

lemma SMP_subterms_eq:
  "SMP (subtermsset M) = SMP M"
  ⟨proof⟩

lemma SMP_funs_term:
  assumes t: "t ∈ SMP M" "f ∈ funs_term t ∨ (∃x ∈ fv t. f ∈ funs_term (Γ (Var x)))"
    and f: "arity f > 0"
    and M: "wftrms M"
    and Ana_f: "∧s K T. Ana s = (K,T) ⇒ f ∈ ∪ (funs_term ` set K) ⇒ f ∈ funs_term s"
  shows "f ∈ ∪ (funs_term ` M) ∨ (∃x ∈ fvset M. f ∈ funs_term (Γ (Var x)))"
  ⟨proof⟩

lemma id_type_eq:
  assumes "Γ (Fun f X) = Γ (Fun g Y)"
  shows "f ∈ C ⇒ g ∈ C" "f ∈ Σf ⇒ g ∈ Σf"
  ⟨proof⟩

lemma fun_type_arg_cong:
  assumes "f ∈ Σf" "g ∈ Σf" "Γ (Fun f (x#X)) = Γ (Fun g (y#Y))"
  shows "Γ x = Γ y" "Γ (Fun f X) = Γ (Fun g Y)"
  ⟨proof⟩

lemma fun_type_arg_cong':
  assumes "f ∈ Σf" "g ∈ Σf" "Γ (Fun f (X@x#X')) = Γ (Fun g (Y@y#Y'))" "length X = length Y"
  shows "Γ x = Γ y"
  ⟨proof⟩

lemma fun_type_param_idx: "Γ (Fun f T) = Fun g S ⇒ i < length T ⇒ Γ (T ! i) = S ! i"
  ⟨proof⟩

lemma fun_type_param_ex:
  assumes "Γ (Fun f T) = Fun g (map Γ S)" "t ∈ set S"
  shows "∃s ∈ set T. Γ s = Γ t"
  ⟨proof⟩

lemma tfr_stp_all_split:
  "list_all tfrstp (x#S) ⇒ list_all tfrstp [x]"
  "list_all tfrstp (x#S) ⇒ list_all tfrstp S"
  "list_all tfrstp (S@S') ⇒ list_all tfrstp S"
  "list_all tfrstp (S@S') ⇒ list_all tfrstp S'"
  "list_all tfrstp (S@x#S') ⇒ list_all tfrstp (S@S')"
  ⟨proof⟩

lemma tfr_stp_all_append:
  assumes "list_all tfrstp S" "list_all tfrstp S'"
  shows "list_all tfrstp (S@S')"
  ⟨proof⟩

lemma tfr_stp_all_wt_subst_apply:
  assumes "list_all tfrstp S"
    and ϑ: "wtsubst ϑ" "wftrms (subst_range ϑ)"
    "bvarsst S ∩ range_vars ϑ = {}"
  shows "list_all tfrstp (S ·st ϑ)"
  ⟨proof⟩

lemma tfr_stp_all_same_type:
  "list_all tfrstp (S@Equality a t t'#S') ⇒ Unifier δ t t' ⇒ Γ t = Γ t'"
  ⟨proof⟩

```

```

lemma tfr_subset:
  " $\bigwedge A B. \text{tfr}_{set} (A \cup B) \implies \text{tfr}_{set} A$ "
  " $\bigwedge A B. \text{tfr}_{set} B \implies A \subseteq B \implies \text{tfr}_{set} A$ "
  " $\bigwedge A B. \text{tfr}_{set} B \implies \text{SMP } A \subseteq \text{SMP } B \implies \text{tfr}_{set} A$ "
<proof>

lemma tfr_empty[simp]: "tfrset {}"
<proof>

lemma tfr_consts_mono:
  assumes " $\forall t \in M. \exists c. t = \text{Fun } c []$ "
    and " $\forall t \in M. \text{Ana } t = ([], [])$ "
    and "tfrset N"
  shows "tfrset (N  $\cup$  M)"
<proof>

lemma dualst_tfrstp: "list_all tfrstp S  $\implies$  list_all tfrstp (dualst S)"
<proof>

lemma subst_var_inv_wt:
  assumes "wtsubst  $\delta$ "
  shows "wtsubst (subst_var_inv  $\delta$  X)"
<proof>

lemma subst_var_inv_wf_trms:
  "wftrms (subst_range (subst_var_inv  $\delta$  X))"
<proof>

lemma unify_list_wt_if_same_type:
  assumes "Unification.unify E B = Some U" " $\forall (s,t) \in \text{set } E. \text{wf}_{trm} s \wedge \text{wf}_{trm} t \wedge \Gamma s = \Gamma t$ "
  and " $\forall (v,t) \in \text{set } B. \Gamma (\text{Var } v) = \Gamma t$ "
  shows " $\forall (v,t) \in \text{set } U. \Gamma (\text{Var } v) = \Gamma t$ "
<proof>

lemma mgu_wt_if_same_type:
  assumes "mgu s t = Some  $\sigma$ " "wftrm s" "wftrm t" " $\Gamma s = \Gamma t$ "
  shows "wtsubst  $\sigma$ "
<proof>

lemma wt_Unifier_if_Unifier:
  assumes s_t: "wftrm s" "wftrm t" " $\Gamma s = \Gamma t$ "
  and  $\delta$ : "Unifier  $\delta$  s t"
  shows " $\exists \vartheta. \text{Unifier } \vartheta s t \wedge \text{wt}_{subst} \vartheta \wedge \text{wf}_{trms} (\text{subst\_range } \vartheta)$ "
<proof>

end

```

3.3.3 Automatically Proving Type-Flaw Resistance

Definitions: Variable Renaming

abbreviation "max_var t \equiv Max (insert 0 (snd ` fv t))"

abbreviation "max_var_set X \equiv Max (insert 0 (snd ` X))"

definition "var_rename n v \equiv Var (fst v, snd v + Suc n)"

definition "var_rename_inv n v \equiv Var (fst v, snd v - Suc n)"

Definitions: Computing a Finite Representation of the Sub-Message Patterns

A sufficient requirement for a term to be a well-typed instance of another term

definition is_wt_instance_of_cond where

"is_wt_instance_of_cond $\Gamma t s \equiv$ (
 $\Gamma t = \Gamma s \wedge$ (case mgu t s of

```

None ⇒ False
| Some δ ⇒ inj_on δ (fv t) ∧ (∀x ∈ fv t. is_Var (δ x)))"

```

definition has_all_wt_instances_of where

```
"has_all_wt_instances_of Γ N M ≡ ∀t ∈ N. ∃s ∈ M. is_wt_instance_of_cond Γ t s"
```

This function computes a finite representation of the set of sub-message patterns

definition SMP0 where

```
"SMP0 Ana Γ M ≡ let
  f = λt. Fun (the_Fun (Γ t)) (map Var (zip (args (Γ t)) [0..

```

These definitions are useful to refine an SMP representation set

fun generalize_term where

```
"generalize_term _ _ n (Var x) = (Var x, n)"
| "generalize_term Γ p n (Fun f T) = (let τ = Γ (Fun f T)
  in if p τ then (Var (τ, n), Suc n)
  else let (T',n') = foldr (λt (S,m). let (t',m') = generalize_term Γ p m t in (t'#S,m'))
    T ([],n)
  in (Fun f T', n'))"
```

definition generalize_terms where

```
"generalize_terms Γ p ≡ map (fst ∘ generalize_term Γ p 0)"
```

definition remove_superfluous_terms where

```
"remove_superfluous_terms Γ T ≡
let
  f = λS t R. ∃s ∈ set S - R. s ≠ t ∧ is_wt_instance_of_cond Γ t s;
  g = λS t (U,R). if f S t R then (U, insert t R) else (t#U, R);
  h = λS. remdups (fst (foldr (g S) S ([],{})))
in while (λS. h S ≠ S) h T"
```

Definitions: Checking Type-Flaw Resistance

definition is_TComp_var_instance_closed where

```
"is_TComp_var_instance_closed Γ M ≡ ∀x ∈ fv_set (set M). is_Fun (Γ (Var x)) →
  list_ex (λt. is_Fun t ∧ Γ t = Γ (Var x) ∧ list_all is_Var (args t) ∧ distinct (args t)) M"
```

definition finite_SMP_representation where

```
"finite_SMP_representation arity Ana Γ M ≡
  list_all (wf_trm' arity) M ∧
  has_all_wt_instances_of Γ (subterms_set (set M)) (set M) ∧
  has_all_wt_instances_of Γ (∪((set ∘ fst ∘ Ana) ` set M)) (set M) ∧
  is_TComp_var_instance_closed Γ M"
```

definition comp_tfr_set where

```
"comp_tfr_set arity Ana Γ M ≡
  finite_SMP_representation arity Ana Γ M ∧
  (let δ = var_rename (max_var_set (fv_set (set M)))
  in ∀s ∈ set M. ∀t ∈ set M. is_Fun s ∧ is_Fun t ∧ Γ s ≠ Γ t → mgu s (t · δ) = None)"
```

fun comp_tfr_stp where

```
"comp_tfr_stp Γ (⟨_ : t ≐ t'⟩_st) = (mgu t t' ≠ None → Γ t = Γ t)"
| "comp_tfr_stp Γ (∀X(∀≠: F)_st) = (
  (∀x ∈ fv_pairs F - set X. is_Var (Γ (Var x))) ∨
  (∀u ∈ subterms_set (trms_pairs F).
    is_Fun u → (args u = [] ∨ (∃s ∈ set (args u). s ∉ Var ` set X))))"
| "comp_tfr_stp _ _ = True"
```

definition comp_tfr_st where

3 The Typing Result for Non-Stateful Protocols

```
"comp_tfrst arity Ana  $\Gamma$  M S  $\equiv$ 
  list_all (comp_tfrstp  $\Gamma$ ) S  $\wedge$ 
  list_all (wftrm 'arity) (trms_listst S)  $\wedge$ 
  has_all_wt_instances_of  $\Gamma$  (trmsst S) (set M)  $\wedge$ 
  comp_tfrset arity Ana  $\Gamma$  M"
```

Small Lemmata

```
lemma less_Suc_max_var_set:
  assumes z: "z  $\in$  X"
  and X: "finite X"
  shows "snd z < Suc (max_var_set X)"
<proof>
```

```
lemma (in typed_model) finite_SMP_representationD:
  assumes "finite_SMP_representation arity Ana  $\Gamma$  M"
  shows "wftrms (set M)"
  and "has_all_wt_instances_of  $\Gamma$  (subtermsset (set M)) (set M)"
  and "has_all_wt_instances_of  $\Gamma$  ( $\bigcup$  ((set  $\circ$  fst  $\circ$  Ana) ` set M)) (set M)"
  and "is_TComp_var_instance_closed  $\Gamma$  M"
<proof>
```

```
lemma (in typed_model) is_wt_instance_of_condD:
  assumes t_instance_s: "is_wt_instance_of_cond  $\Gamma$  t s"
  obtains  $\delta$  where
  " $\Gamma$  t =  $\Gamma$  s" "mgu t s = Some  $\delta$ "
  "inj_on  $\delta$  (fv t)" " $\delta$  ` (fv t)  $\subseteq$  range Var"
<proof>
```

```
lemma (in typed_model) is_wt_instance_of_condD':
  assumes t_wf_trm: "wftrm t"
  and s_wf_trm: "wftrm s"
  and t_instance_s: "is_wt_instance_of_cond  $\Gamma$  t s"
  shows " $\exists \delta$ . wtsubst  $\delta$   $\wedge$  wftrms (subst_range  $\delta$ )  $\wedge$  t = s  $\cdot$   $\delta$ "
<proof>
```

```
lemma (in typed_model) is_wt_instance_of_condD'':
  assumes s_wf_trm: "wftrm s"
  and t_instance_s: "is_wt_instance_of_cond  $\Gamma$  t s"
  and t_var: "t = Var x"
  shows " $\exists y$ . s = Var y  $\wedge$   $\Gamma$  (Var y) =  $\Gamma$  (Var x)"
<proof>
```

```
lemma (in typed_model) has_all_wt_instances_ofD:
  assumes N_instance_M: "has_all_wt_instances_of  $\Gamma$  N M"
  and t_in_N: "t  $\in$  N"
  obtains s  $\delta$  where
  "s  $\in$  M" " $\Gamma$  t =  $\Gamma$  s" "mgu t s = Some  $\delta$ "
  "inj_on  $\delta$  (fv t)" " $\delta$  ` (fv t)  $\subseteq$  range Var"
<proof>
```

```
lemma (in typed_model) has_all_wt_instances_ofD':
  assumes N_wf_trms: "wftrms N"
  and M_wf_trms: "wftrms M"
  and N_instance_M: "has_all_wt_instances_of  $\Gamma$  N M"
  and t_in_N: "t  $\in$  N"
  shows " $\exists \delta$ . wtsubst  $\delta$   $\wedge$  wftrms (subst_range  $\delta$ )  $\wedge$  t  $\in$  M  $\cdot$ set  $\delta$ "
<proof>
```

```
lemma (in typed_model) has_all_wt_instances_ofD'':
  assumes N_wf_trms: "wftrms N"
  and M_wf_trms: "wftrms M"
  and N_instance_M: "has_all_wt_instances_of  $\Gamma$  N M"
```



```

    and t_in_N: "Var x ∈ N"
    shows "∃y. Var y ∈ M ∧ Γ (Var y) = Γ (Var x)"
  ⟨proof⟩

lemma (in typed_model) has_all_instances_of_if_subset:
  assumes "N ⊆ M"
  shows "has_all_wt_instances_of Γ N M"
  ⟨proof⟩

lemma (in typed_model) SMP_I':
  assumes N_wf_trms: "wf_trms N"
  and M_wf_trms: "wf_trms M"
  and N_instance_M: "has_all_wt_instances_of Γ N M"
  and t_in_N: "t ∈ N"
  shows "t ∈ SMP M"
  ⟨proof⟩

```

Lemma: Proving Type-Flaw Resistance

```

locale typed_model' = typed_model arity public Ana Γ
  for arity::"'fun ⇒ nat"
  and public::"'fun ⇒ bool"
  and Ana::"'(fun, (('fun, 'atom::finite) term_type × nat)) term
    ⇒ (('fun, (('fun, 'atom) term_type × nat)) term list
      × ('fun, (('fun, 'atom) term_type × nat)) term list)"
  and Γ::"'(fun, (('fun, 'atom) term_type × nat)) term ⇒ ('fun, 'atom) term_type"
+
  assumes Γ_Var_fst: "∧τ n m. Γ (Var (τ,n)) = Γ (Var (τ,m))"
  and Ana_const: "∧c T. arity c = 0 ⇒ Ana (Fun c T) = ([], [])"
  and Ana_subst'_or_Ana_keys_subterm:
    "(∀f T δ K R. Ana (Fun f T) = (K,R) → Ana (Fun f T · δ) = (K ·list δ, R ·list δ)) ∨
     (∀t K R k. Ana t = (K,R) → k ∈ set K → k ⊆ t)"
begin

lemma var_rename_inv_comp: "t · (var_rename n ◦s var_rename_inv n) = t"
  ⟨proof⟩

lemma var_rename_fv_disjoint:
  "fv s ∩ fv (t · var_rename (max_var s)) = {}"
  ⟨proof⟩

lemma var_rename_fv_set_disjoint:
  assumes "finite M" "s ∈ M"
  shows "fv s ∩ fv (t · var_rename (max_var_set (fvset M))) = {}"
  ⟨proof⟩

lemma var_rename_fv_set_disjoint':
  assumes "finite M"
  shows "fvset M ∩ fvset (N ·set var_rename (max_var_set (fvset M))) = {}"
  ⟨proof⟩

lemma var_rename_is_renaming[simp]:
  "subst_range (var_rename n) ⊆ range Var"
  "subst_range (var_rename_inv n) ⊆ range Var"
  ⟨proof⟩

lemma var_rename_wt[simp]:
  "wtsubst (var_rename n)"
  "wtsubst (var_rename_inv n)"
  ⟨proof⟩

lemma var_rename_wt':
  assumes "wtsubst δ" "s = m · δ"

```

3 The Typing Result for Non-Stateful Protocols

shows "wt_{subst} (var_rename_inv n ◦_s δ)" "s = m · var_rename n · var_rename_inv n ◦_s δ"
 ⟨proof⟩

lemma var_rename_wf_{trms}_range[simp]:
 "wf_{trms} (subst_range (var_rename n))"
 "wf_{trms} (subst_range (var_rename_inv n))"
 ⟨proof⟩

lemma Fun_range_case:
 "(∀ f T. Fun f T ∈ M → P f T) ↔ (∀ u ∈ M. case u of Fun f T ⇒ P f T | _ ⇒ True)"
 "(∀ f T. Fun f T ∈ M → P f T) ↔ (∀ u ∈ M. is_Fun u → P (the_Fun u) (args u))"
 ⟨proof⟩

lemma is_TComp_var_instance_closedD:
 assumes x: "∃ y ∈ fv_{set} (set M). Γ (Var x) = Γ (Var y)" "Γ (Var x) = TComp f T"
 and closed: "is_TComp_var_instance_closed Γ M"
 shows "∃ g U. Fun g U ∈ set M ∧ Γ (Fun g U) = Γ (Var x) ∧ (∀ u ∈ set U. is_Var u) ∧ distinct U"
 ⟨proof⟩

lemma is_TComp_var_instance_closedD':
 assumes "∃ y ∈ fv_{set} (set M). Γ (Var x) = Γ (Var y)" "TComp f T ⊆ Γ (Var x)"
 and closed: "is_TComp_var_instance_closed Γ M"
 and wf: "wf_{trms} (set M)"
 shows "∃ g U. Fun g U ∈ set M ∧ Γ (Fun g U) = TComp f T ∧ (∀ u ∈ set U. is_Var u) ∧ distinct U"
 ⟨proof⟩

lemma TComp_var_instance_wt_subst_exists:
 assumes gT: "Γ (Fun g T) = TComp g (map Γ U)" "wf_{trm} (Fun g T)"
 and U: "∀ u ∈ set U. ∃ y. u = Var y" "distinct U"
 shows "∃ ∅. wt_{subst} ∅ ∧ wf_{trms} (subst_range ∅) ∧ Fun g T = Fun g U · ∅"
 ⟨proof⟩

lemma TComp_var_instance_closed_has_Var:
 assumes closed: "is_TComp_var_instance_closed Γ M"
 and wf_M: "wf_{trms} (set M)"
 and wf_δx: "wf_{trm} (δ x)"
 and y_ex: "∃ y ∈ fv_{set} (set M). Γ (Var x) = Γ (Var y)"
 and t: "t ⊆ δ x"
 and δ_wt: "wt_{subst} δ"
 shows "∃ y ∈ fv_{set} (set M). Γ (Var y) = Γ t"
 ⟨proof⟩

lemma TComp_var_instance_closed_has_Fun:
 assumes closed: "is_TComp_var_instance_closed Γ M"
 and wf_M: "wf_{trms} (set M)"
 and wf_δx: "wf_{trm} (δ x)"
 and y_ex: "∃ y ∈ fv_{set} (set M). Γ (Var x) = Γ (Var y)"
 and t: "t ⊆ δ x"
 and δ_wt: "wt_{subst} δ"
 and t_Γ: "Γ t = TComp g T"
 and t_fun: "is_Fun t"
 shows "∃ m ∈ set M. ∃ ∅. wt_{subst} ∅ ∧ wf_{trms} (subst_range ∅) ∧ t = m · ∅ ∧ is_Fun m"
 ⟨proof⟩

lemma TComp_var_and_subterm_instance_closed_has_subterms_instances:
 assumes M_var_inst_cl: "is_TComp_var_instance_closed Γ M"
 and M_subterms_cl: "has_all_wt_instances_of Γ (subterms_{set} (set M)) (set M)"
 and M_wf: "wf_{trms} (set M)"
 and t: "t ⊆_{set} set M"
 and s: "s ⊆ t · δ"
 and δ: "wt_{subst} δ" "wf_{trms} (subst_range δ)"
 shows "∃ m ∈ set M. ∃ ∅. wt_{subst} ∅ ∧ wf_{trms} (subst_range ∅) ∧ s = m · ∅"
 ⟨proof⟩

```

context
begin
private lemma SMP_D_aux1:
  assumes "t ∈ SMP (set M)"
    and closed: "has_all_wt_instances_of Γ (subtermsset (set M)) (set M)"
      "is_TComp_var_instance_closed Γ M"
    and wf_M: "wftrms (set M)"
  shows "∀x ∈ fv t. ∃y ∈ fvset (set M). Γ (Var y) = Γ (Var x)"
⟨proof⟩ lemma SMP_D_aux2:
  fixes t::('fun, ('fun, 'atom) term × nat) term"
  assumes t_SMP: "t ∈ SMP (set M)"
    and t_Var: "∃x. t = Var x"
    and M_SMP_repr: "finite_SMP_representation arity Ana Γ M"
  shows "∃m ∈ set M. ∃δ. wtsubst δ ∧ wftrms (subst_range δ) ∧ t = m · δ"
⟨proof⟩ lemma SMP_D_aux3:
  assumes hyps: "t' ⊆ t" and wf_t: "wftrm t" and prems: "is_Fun t'"
    and IH:
      "(∃f. t = Fun f []) ∧ (∃m ∈ set M. ∃δ. wtsubst δ ∧ wftrms (subst_range δ) ∧ t = m · δ) ∨
      (∃m ∈ set M. ∃δ. wtsubst δ ∧ wftrms (subst_range δ) ∧ t = m · δ ∧ is_Fun m)"
    and M_SMP_repr: "finite_SMP_representation arity Ana Γ M"
  shows "(∃f. t' = Fun f []) ∧ (∃m ∈ set M. ∃δ. wtsubst δ ∧ wftrms (subst_range δ) ∧ t' = m · δ) ∨
      (∃m ∈ set M. ∃δ. wtsubst δ ∧ wftrms (subst_range δ) ∧ t' = m · δ ∧ is_Fun m)"
⟨proof⟩

lemma SMP_D:
  assumes "t ∈ SMP (set M)" "is_Fun t"
    and M_SMP_repr: "finite_SMP_representation arity Ana Γ M"
  shows "(∃f. t = Fun f []) ∧ (∃m ∈ set M. ∃δ. wtsubst δ ∧ wftrms (subst_range δ) ∧ t = m · δ) ∨
      (∃m ∈ set M. ∃δ. wtsubst δ ∧ wftrms (subst_range δ) ∧ t = m · δ ∧ is_Fun m)"
⟨proof⟩

lemma SMP_D':
  fixes M
  defines "δ ≡ var_rename (max_var_set (fvset (set M)))"
  assumes M_SMP_repr: "finite_SMP_representation arity Ana Γ M"
    and s: "s ∈ SMP (set M)" "is_Fun s" "∄f. s = Fun f []"
    and t: "t ∈ SMP (set M)" "is_Fun t" "∄f. t = Fun f []"
  obtains σ s0 ∅ t0
  where "wtsubst σ" "wftrms (subst_range σ)" "s0 ∈ set M" "is_Fun s0" "s = s0 · σ" "Γ s = Γ s0"
    and "wtsubst ∅" "wftrms (subst_range ∅)" "t0 ∈ set M" "is_Fun t0" "t = t0 · δ · ∅" "Γ t = Γ t0"
⟨proof⟩

lemma SMP_D'':
  fixes t::('fun, ('fun, 'atom) term × nat) term"
  assumes t_SMP: "t ∈ SMP (set M)"
    and M_SMP_repr: "finite_SMP_representation arity Ana Γ M"
  shows "∃m ∈ set M. ∃δ. wtsubst δ ∧ wftrms (subst_range δ) ∧ t = m · δ"
⟨proof⟩
end

lemma tfrset_if_comp_tfrset:
  assumes "comp_tfrset arity Ana Γ M"
  shows "tfrset (set M)"
⟨proof⟩

lemma tfrset_if_comp_tfrset':
  assumes "let N = SMPO Ana Γ M in set M ⊆ set N ∧ comp_tfrset arity Ana Γ N"
  shows "tfrset (set M)"
⟨proof⟩

lemma tfrstp_is_comp_tfrstp: "tfrstp a = comp_tfrstp Γ a"
⟨proof⟩

```

```

lemma tfrst_if_comp_tfrst:
  assumes "comp_tfrst arity Ana  $\Gamma$  M S"
  shows "tfrst S"
<proof>

lemma tfrst_if_comp_tfrst':
  assumes "comp_tfrst arity Ana  $\Gamma$  (SMPO Ana  $\Gamma$  (trms_listst S)) S"
  shows "tfrst S"
<proof>

```

Lemmata for Checking Ground SMP (GSMP) Disjointness

```

context
begin
private lemma ground_SMP_disjointI_aux1:
  fixes M::('fun, ('fun, 'atom) term  $\times$  nat) term set"
  assumes f_def: "f  $\equiv$   $\lambda$ M. {t  $\cdot$   $\delta$  | t  $\delta$ . t  $\in$  M  $\wedge$  wtsubst  $\delta$   $\wedge$  wftrms (subst_range  $\delta$ )  $\wedge$  fv (t  $\cdot$   $\delta$ ) = {}}"
  and g_def: "g  $\equiv$   $\lambda$ M. {t  $\in$  M. fv t = {}}"
  shows "f (SMP M) = g (SMP M)"
<proof> lemma ground_SMP_disjointI_aux2:
  fixes M::('fun, ('fun, 'atom) term  $\times$  nat) term list"
  assumes f_def: "f  $\equiv$   $\lambda$ M. {t  $\cdot$   $\delta$  | t  $\delta$ . t  $\in$  M  $\wedge$  wtsubst  $\delta$   $\wedge$  wftrms (subst_range  $\delta$ )  $\wedge$  fv (t  $\cdot$   $\delta$ ) = {}}"
  and M_SMP_repr: "finite_SMP_representation arity Ana  $\Gamma$  M"
  shows "f (set M) = f (SMP (set M))"
<proof> lemma ground_SMP_disjointI_aux3:
  fixes A B C::('fun, ('fun, 'atom) term  $\times$  nat) term set"
  defines "P  $\equiv$   $\lambda$ t s.  $\exists$  $\delta$ . wtsubst  $\delta$   $\wedge$  wftrms (subst_range  $\delta$ )  $\wedge$  Unifier  $\delta$  t s"
  assumes f_def: "f  $\equiv$   $\lambda$ M. {t  $\cdot$   $\delta$  | t  $\delta$ . t  $\in$  M  $\wedge$  wtsubst  $\delta$   $\wedge$  wftrms (subst_range  $\delta$ )  $\wedge$  fv (t  $\cdot$   $\delta$ ) = {}}"
  and Q_def: "Q  $\equiv$   $\lambda$ t. intruder_synth' public arity {} t"
  and R_def: "R  $\equiv$   $\lambda$ t.  $\exists$ u  $\in$  C. is_wt_instance_of_cond  $\Gamma$  t u"
  and AB: "wftrms A" "wftrms B" "fvset A  $\cap$  fvset B = {}"
  and C: "wftrms C"
  and ABC: " $\forall$ t  $\in$  A.  $\forall$ s  $\in$  B. P t s  $\longrightarrow$  (Q t  $\wedge$  Q s)  $\vee$  (R t  $\wedge$  R s)"
  shows "f A  $\cap$  f B  $\subseteq$  f C  $\cup$  {m. {}  $\vdash_c$  m}"
<proof>

lemma ground_SMP_disjointI:
  fixes A B::('fun, ('fun, 'atom) term  $\times$  nat) term list" and C
  defines "f  $\equiv$   $\lambda$ M. {t  $\cdot$   $\delta$  | t  $\delta$ . t  $\in$  M  $\wedge$  wtsubst  $\delta$   $\wedge$  wftrms (subst_range  $\delta$ )  $\wedge$  fv (t  $\cdot$   $\delta$ ) = {}}"
  and "g  $\equiv$   $\lambda$ M. {t  $\in$  M. fv t = {}}"
  and "Q  $\equiv$   $\lambda$ t. intruder_synth' public arity {} t"
  and "R  $\equiv$   $\lambda$ t.  $\exists$ u  $\in$  C. is_wt_instance_of_cond  $\Gamma$  t u"
  assumes AB_fv_disj: "fvset (set A)  $\cap$  fvset (set B) = {}"
  and A_SMP_repr: "finite_SMP_representation arity Ana  $\Gamma$  A"
  and B_SMP_repr: "finite_SMP_representation arity Ana  $\Gamma$  B"
  and C_wf: "wftrms C"
  and ABC: " $\forall$ t  $\in$  set A.  $\forall$ s  $\in$  set B.  $\Gamma$  t =  $\Gamma$  s  $\wedge$  mgu t s  $\neq$  None  $\longrightarrow$  (Q t  $\wedge$  Q s)  $\vee$  (R t  $\wedge$  R s)"
  shows "g (SMP (set A))  $\cap$  g (SMP (set B))  $\subseteq$  f C  $\cup$  {m. {}  $\vdash_c$  m}"
<proof>

end

end

end

```

3.4 The Typing Result (Typing_Result)

```
theory Typing_Result
imports Typed_Model
begin
```

3.4.1 The Typing Result for the Composition-Only Intruder

```
context typed_model
begin
```

Well-typedness and Type-Flaw Resistance Preservation

```
context
begin
```

```
private lemma LI_preserves_tfr_stp_all_single:
  assumes "(S,ϑ) ~ (S',ϑ')" "wf_constr S ϑ" "wt_subst ϑ"
  and "list_all tfr_stp S" "tfr_set (trms_st S)" "wf_trms (trms_st S)"
  shows "list_all tfr_stp S'"
<proof> lemma LI_in_SMP_subset_single:
  assumes "(S,ϑ) ~ (S',ϑ')" "wf_constr S ϑ" "wt_subst ϑ"
  "tfr_set (trms_st S)" "wf_trms (trms_st S)" "list_all tfr_stp S"
  and "trms_st S ⊆ SMP M"
  shows "trms_st S' ⊆ SMP M"
<proof> lemma LI_preserves_tfr_single:
  assumes "(S,ϑ) ~ (S',ϑ')" "wf_constr S ϑ" "wt_subst ϑ" "wf_trms (subst_range ϑ)"
  "tfr_set (trms_st S)" "wf_trms (trms_st S)"
  "list_all tfr_stp S"
  shows "tfr_set (trms_st S') ∧ wf_trms (trms_st S)"
<proof> lemma LI_preserves_welltypedness_single:
  assumes "(S,ϑ) ~ (S',ϑ')" "wf_constr S ϑ" "wt_subst ϑ" "wf_trms (subst_range ϑ)"
  and "tfr_set (trms_st S)" "wf_trms (trms_st S)" "list_all tfr_stp S"
  shows "wt_subst ϑ' ∧ wf_trms (subst_range ϑ)"
<proof>
lemma LI_preserves_welltypedness:
  assumes "(S,ϑ) ~* (S',ϑ')" "wf_constr S ϑ" "wt_subst ϑ" "wf_trms (subst_range ϑ)"
  and "tfr_set (trms_st S)" "wf_trms (trms_st S)" "list_all tfr_stp S"
  shows "wt_subst ϑ'" (is "?A ϑ'")
  and "wf_trms (subst_range ϑ)" (is "?B ϑ'")
<proof>
lemma LI_preserves_tfr:
  assumes "(S,ϑ) ~* (S',ϑ')" "wf_constr S ϑ" "wt_subst ϑ" "wf_trms (subst_range ϑ)"
  and "tfr_set (trms_st S)" "wf_trms (trms_st S)" "list_all tfr_stp S"
  shows "tfr_set (trms_st S'" (is "?A S'")
  and "wf_trms (trms_st S'" (is "?B S'")
  and "list_all tfr_stp S'" (is "?C S'")
<proof>
end
```

Simple Constraints are Well-typed Satisfiable

Proving the existence of a well-typed interpretation

```
context
begin
lemma wt_interpretation_exists:
  obtains I::('fun, 'var) subst"
  where "interpretation_subst I" "wt_subst I" "subst_range I ⊆ public_ground_wf_terms"
<proof>
```

lemma `wt_grounding_subst_exists`:

```
" $\exists \vartheta. wt_{subst} \vartheta \wedge wf_{trms} (subst\_range \vartheta) \wedge fv (t \cdot \vartheta) = \{\}$ "
<proof> fun fresh_pgw: "'fun set  $\Rightarrow$  ('fun, 'atom) term_type  $\Rightarrow$  ('fun, 'var) term" where
  "fresh_pgw S (TAtom a) =
    Fun (SOME c. c  $\notin$  S  $\wedge$   $\Gamma$  (Fun c []) = TAtom a  $\wedge$  public c) []"
| "fresh_pgw S (TComp f T) = Fun f (map (fresh_pgw S) T)"
```

private lemma `fresh_pgw_same_type`:

```
assumes "finite S" "wftrm t"
shows " $\Gamma$  (fresh_pgw S ( $\Gamma$  t)) =  $\Gamma$  t"
```

<proof> **lemma** `fresh_pgw_empty_synth`:

```
assumes "finite S" "wftrm t"
shows " $\{\} \vdash_c$  fresh_pgw S ( $\Gamma$  t)"
```

<proof> **lemma** `fresh_pgw_has_fresh_const`:

```
assumes "finite S" "wftrm t"
obtains c where "Fun c []  $\sqsubseteq$  fresh_pgw S ( $\Gamma$  t)" "c  $\notin$  S"
```

<proof> **lemma** `fresh_pgw_subterm_fresh`:

```
assumes "finite S" "wftrm t" "wftrm s" "funs_term s  $\subseteq$  S"
shows "s  $\notin$  subterms (fresh_pgw S ( $\Gamma$  t))"
```

<proof> **lemma** `wt_fresh_pgw_term_exists`:

```
assumes "finite T" "wftrm s" "wftrms T"
obtains t where " $\Gamma$  t =  $\Gamma$  s" " $\{\} \vdash_c$  t" " $\forall s \in T. \forall u \in$  subterms s. u  $\notin$  subterms t"
```

<proof>

lemma `wt_bij_finite_subst_exists`:

```
assumes "finite (S::'var set)" "finite (T::('fun, 'var) terms)" "wftrms T"
shows " $\exists \sigma::$ ('fun, 'var) subst.
  subst_domain  $\sigma$  = S
   $\wedge$  bij_betw  $\sigma$  (subst_domain  $\sigma$ ) (subst_range  $\sigma$ )
   $\wedge$  subtermsset (subst_range  $\sigma$ )  $\subseteq$  {t.  $\{\} \vdash_c$  t} - T
   $\wedge$  ( $\forall s \in$  subst_range  $\sigma. \forall u \in$  subst_range  $\sigma. (\exists v. v \sqsubseteq s \wedge v \sqsubseteq u) \longrightarrow s = u$ )
   $\wedge$  wtsubst  $\sigma$ 
   $\wedge$  wftrms (subst_range  $\sigma$ )"
```

<proof> **lemma** `wt_bij_finite_tatom_subst_exists_single`:

```
assumes "finite (S::'var set)" "finite (T::('fun, 'var) terms)"
and " $\wedge x. x \in S \implies \Gamma$  (Var x) = TAtom a"
shows " $\exists \sigma::$ ('fun, 'var) subst. subst_domain  $\sigma$  = S
   $\wedge$  bij_betw  $\sigma$  (subst_domain  $\sigma$ ) (subst_range  $\sigma$ )
   $\wedge$  subst_range  $\sigma \subseteq ((\lambda c. \text{Fun } c \text{ []}) \cdot \{c. \Gamma (\text{Fun } c \text{ []}) = \text{TAtom } a \wedge$ 
    public c  $\wedge$  arity c = 0}) - T
   $\wedge$  wtsubst  $\sigma$ 
   $\wedge$  wftrms (subst_range  $\sigma$ )"
```

<proof>

lemma `wt_bij_finite_tatom_subst_exists`:

```
assumes "finite (S::'var set)" "finite (T::('fun, 'var) terms)"
and " $\wedge x. x \in S \implies \exists a. \Gamma$  (Var x) = TAtom a"
shows " $\exists \sigma::$ ('fun, 'var) subst. subst_domain  $\sigma$  = S
   $\wedge$  bij_betw  $\sigma$  (subst_domain  $\sigma$ ) (subst_range  $\sigma$ )
   $\wedge$  subst_range  $\sigma \subseteq ((\lambda c. \text{Fun } c \text{ []}) \cdot \mathcal{C}_{pub}) - T$ 
   $\wedge$  wtsubst  $\sigma$ 
   $\wedge$  wftrms (subst_range  $\sigma$ )"
```

<proof>

theorem `wt_sat_if_simple`:

```
assumes "simple S" "wfconstr S  $\vartheta$ " "wtsubst  $\vartheta$ " "wftrms (subst_range  $\vartheta$ )" "wftrms (trmsst S)"
and  $\mathcal{I}'$ : " $\forall X F. \text{Inequality } X F \in \text{set } S \longrightarrow \text{ineq\_model } \mathcal{I}' X F$ "
  "ground (subst_range  $\mathcal{I}'$ )"
  "subst_domain  $\mathcal{I}' = \{x \in \text{vars}_{st} S. \exists X F. \text{Inequality } X F \in \text{set } S \wedge x \in fv_{pairs} F - \text{set } X\}$ "
and tfr_stp_all: "list_all tfrstp S"
shows " $\exists \mathcal{I}. \text{interpretation}_{subst} \mathcal{I} \wedge (\mathcal{I} \models_c \langle S, \vartheta \rangle) \wedge wt_{subst} \mathcal{I} \wedge wf_{trms} (subst\_range \mathcal{I})"$ "
```

<proof>

end

Theorem: Type-flaw resistant constraints are well-typed satisfiable (composition-only)

There exists well-typed models of satisfiable type-flaw resistant constraints in the semantics where the intruder is limited to composition only (i.e., he cannot perform decomposition/analysis of deducible messages).

```

theorem wt_attack_if_tfr_attack:
  assumes "interpretation_subst I"
    and "I ⊨c ⟨S, ϑ⟩"
    and "wf_constr S ϑ"
    and "wt_subst ϑ"
    and "tfrst S"
    and "wf_trms (trmsst S)"
    and "wf_trms (subst_range ϑ)"
  obtains Iτ where "interpretation_subst Iτ"
    and "Iτ ⊨c ⟨S, ϑ⟩"
    and "wt_subst Iτ"
    and "wf_trms (subst_range Iτ)"
⟨proof⟩

```

Contra-positive version: if a type-flaw resistant constraint does not have a well-typed model then it is unsatisfiable

```

corollary secure_if_wt_secure:
  assumes "¬(∃Iτ. interpretation_subst Iτ ∧ (Iτ ⊨c ⟨S, ϑ⟩) ∧ wt_subst Iτ)"
    and "wf_constr S ϑ" "wt_subst ϑ" "tfrst S"
    and "wf_trms (trmsst S)" "wf_trms (subst_range ϑ)"
  shows "¬(∃I. interpretation_subst I ∧ (I ⊨c ⟨S, ϑ⟩))"
⟨proof⟩

```

end

3.4.2 Lifting the Composition-Only Typing Result to the Full Intruder Model

```

context typed_model
begin

```

Analysis Invariance

```

definition (in typed_model) Ana_invar_subst where
  "Ana_invar_subst M ≡
    (∀f T K M δ. Fun f T ∈ (subtermsset M) →
      Ana (Fun f T) = (K, M) → Ana (Fun f T · δ) = (K ·list δ, M ·list δ))"

```

```

lemma (in typed_model) Ana_invar_subst_subset:
  assumes "Ana_invar_subst M" "N ⊆ M"
  shows "Ana_invar_subst N"
⟨proof⟩

```

```

lemma (in typed_model) Ana_invar_substD:
  assumes "Ana_invar_subst M"
  and "Fun f T ∈ subtermsset M" "Ana (Fun f T) = (K, M)"
  shows "Ana (Fun f T · I) = (K ·list I, M ·list I)"
⟨proof⟩

```

end

Preliminary Definitions

Strands extended with "decomposition steps"

```

datatype (funsestp: 'a, varsestp: 'b) extstrand_step =
  Step "('a, 'b) strand_step"
| Decomp "('a, 'b) term"

```

```

context typed_model

```

```

begin

context
begin
private fun trmsestp where
  "trmsestp (Step x) = trmsstp x"
  | "trmsestp (Decomp t) = {t}"

private abbreviation trmsest where "trmsest S ≡ ⋃ (trmsestp ` set S)"

private type_synonym ('a,'b) extstrand = "('a,'b) extstrand_step list"
private type_synonym ('a,'b) extstrands = "('a,'b) extstrand set"

private definition decomp::('fun,'var) term ⇒ ('fun,'var) strand where
  "decomp t ≡ (case (Ana t) of (K,T) ⇒ send⟨t⟩st#map Send K@map Receive T)"

private fun tost where
  "tost [] = []"
  | "tost (Step x#S) = x#(tost S)"
  | "tost (Decomp t#S) = (decomp t)@(tost S)"

private fun toest where
  "toest [] = []"
  | "toest (x#S) = Step x#toest S"

private abbreviation "ikest A ≡ ikst (tost A)"
private abbreviation "wfest V A ≡ wfst V (tost A)"
private abbreviation "assignment_rhsest A ≡ assignment_rhsst (tost A)"
private abbreviation "varsest A ≡ varsst (tost A)"
private abbreviation "wfrestrictedvarsest A ≡ wfrestrictedvarsst (tost A)"
private abbreviation "bvarsest A ≡ bvarsst (tost A)"
private abbreviation "fvest A ≡ fvst (tost A)"
private abbreviation "funsest A ≡ funsst (tost A)"

private definition wfsts'::('fun,'var) strands ⇒ ('fun,'var) extstrand ⇒ bool where
  "wfsts' S A ≡ (∀S ∈ S. wfst (wfrestrictedvarsest A) (dualst S)) ∧
    (∀S ∈ S. ∀S' ∈ S. fvst S ∩ bvarsst S' = {}) ∧
    (∀S ∈ S. fvst S ∩ bvarsest A = {}) ∧
    (∀S ∈ S. fvst (tost A) ∩ bvarsst S = {})"

private definition wfsts::('fun,'var) strands ⇒ bool where
  "wfsts S ≡ (∀S ∈ S. wfst {} (dualst S)) ∧ (∀S ∈ S. ∀S' ∈ S. fvst S ∩ bvarsst S' = {})"

private inductive well_analyzed::('fun,'var) extstrand ⇒ bool where
  Nil[simp]: "well_analyzed []"
  | Step: "well_analyzed A ⇒ well_analyzed (A@[Step x])"
  | Decomp: "[well_analyzed A; t ∈ subtermsset (ikest A ∪ assignment_rhsest A) - (Var ` V)]
    ⇒ well_analyzed (A@[Decomp t])"

private fun subst_apply_extstrandstep (infix ".estp" 51) where
  "subst_apply_extstrandstep (Step x) ϑ = Step (x .stp ϑ)"
  | "subst_apply_extstrandstep (Decomp t) ϑ = Decomp (t . ϑ)"

private lemma subst_apply_extstrandstep'_simps[simp]:
  "(Step (send⟨t⟩st)) .estp ϑ = Step (send⟨t . ϑ⟩st)"
  "(Step (receive⟨t⟩st)) .estp ϑ = Step (receive⟨t . ϑ⟩st)"
  "(Step ((a: t ≐ t')st)) .estp ϑ = Step ((a: (t . ϑ) ≐ (t' . ϑ))st)"
  "(Step (∀X(∀≠: F)st)) .estp ϑ = Step (∀X(∀≠: (F .pairs rm_vars (set X) ϑ))st)"
⟨proof⟩ lemma varsestp_subst_apply_simps[simp]:
  "varsestp ((Step (send⟨t⟩st)) .estp ϑ) = fv (t . ϑ)"
  "varsestp ((Step (receive⟨t⟩st)) .estp ϑ) = fv (t . ϑ)"
  "varsestp ((Step ((a: t ≐ t')st)) .estp ϑ) = fv (t . ϑ) ∪ fv (t' . ϑ)"
  "varsestp ((Step (∀X(∀≠: F)st)) .estp ϑ) = set X ∪ fvpairs (F .pairs rm_vars (set X) ϑ)"

```



```

<proof> definition subst_apply_extstrand (infix ".est" 51) where "S .est  $\vartheta \equiv \text{map } (\lambda x. x .estp \vartheta) S"$ 

private abbreviation updatest::("('fun,'var) strands  $\Rightarrow$  ('fun,'var) strand  $\Rightarrow$  ('fun,'var) strands"
where
  "updatest S S  $\equiv$  (case S of Nil  $\Rightarrow$  S - {S} | Cons _ S'  $\Rightarrow$  insert S' (S - {S}))"

private inductive_set decompest::
  "('fun,'var) terms  $\Rightarrow$  ('fun,'var) terms  $\Rightarrow$  ('fun,'var) subst  $\Rightarrow$  ('fun,'var) extstrands"

for M and N and I where
  Nil: "[ ]  $\in$  decompest M N I"
| Decompose: "[D  $\in$  decompest M N I; Fun f T  $\in$  subtermsset (M  $\cup$  N);
  Ana (Fun f T) = (K,M); M  $\neq$  [ ];
  (M  $\cup$  ikest D) .set I  $\vdash_c$  Fun f T . I;
   $\bigwedge k. k \in \text{set } K \Rightarrow$  (M  $\cup$  ikest D) .set I  $\vdash_c$  k . I]
   $\Rightarrow$  D@[Decompose (Fun f T)]  $\in$  decompest M N I"

private fun decomp_rmest::("('fun,'var) extstrand  $\Rightarrow$  ('fun,'var) extstrand" where
  "decomp_rmest [ ] = [ ]"
| "decomp_rmest (Decompose t#S) = decomp_rmest S"
| "decomp_rmest (Step x#S) = Step x#(decomp_rmest S)"

private inductive semest_d::("('fun,'var) terms  $\Rightarrow$  ('fun,'var) subst  $\Rightarrow$  ('fun,'var) extstrand  $\Rightarrow$  bool"
where
  Nil[simp]: "semest_d M0 I [ ]"
| Send: "semest_d M0 I S  $\Rightarrow$  (ikest S  $\cup$  M0) .set I  $\vdash$  t . I  $\Rightarrow$  semest_d M0 I (S@[Step (send<t>st)])"
| Receive: "semest_d M0 I S  $\Rightarrow$  semest_d M0 I (S@[Step (receive<t>st)])"
| Equality: "semest_d M0 I S  $\Rightarrow$  t . I = t' . I  $\Rightarrow$  semest_d M0 I (S@[Step ((a: t  $\doteq$  t')st)])"
| Inequality: "semest_d M0 I S
   $\Rightarrow$  ineq_model I X F
   $\Rightarrow$  semest_d M0 I (S@[Step ( $\forall X(\forall \neq: F)$ )st])"
| Decompose: "semest_d M0 I S  $\Rightarrow$  (ikest S  $\cup$  M0) .set I  $\vdash$  t . I  $\Rightarrow$  Ana t = (K, M)
   $\Rightarrow$  ( $\bigwedge k. k \in \text{set } K \Rightarrow$  (ikest S  $\cup$  M0) .set I  $\vdash$  k . I)  $\Rightarrow$  semest_d M0 I (S@[Decompose t])"

private inductive semest_c::("('fun,'var) terms  $\Rightarrow$  ('fun,'var) subst  $\Rightarrow$  ('fun,'var) extstrand  $\Rightarrow$  bool"
where
  Nil[simp]: "semest_c M0 I [ ]"
| Send: "semest_c M0 I S  $\Rightarrow$  (ikest S  $\cup$  M0) .set I  $\vdash_c$  t . I  $\Rightarrow$  semest_c M0 I (S@[Step (send<t>st)])"
| Receive: "semest_c M0 I S  $\Rightarrow$  semest_c M0 I (S@[Step (receive<t>st)])"
| Equality: "semest_c M0 I S  $\Rightarrow$  t . I = t' . I  $\Rightarrow$  semest_c M0 I (S@[Step ((a: t  $\doteq$  t')st)])"
| Inequality: "semest_c M0 I S
   $\Rightarrow$  ineq_model I X F
   $\Rightarrow$  semest_c M0 I (S@[Step ( $\forall X(\forall \neq: F)$ )st])"
| Decompose: "semest_c M0 I S  $\Rightarrow$  (ikest S  $\cup$  M0) .set I  $\vdash_c$  t . I  $\Rightarrow$  Ana t = (K, M)
   $\Rightarrow$  ( $\bigwedge k. k \in \text{set } K \Rightarrow$  (ikest S  $\cup$  M0) .set I  $\vdash_c$  k . I)  $\Rightarrow$  semest_c M0 I (S@[Decompose t])"

```

Preliminary Lemmata

```

private lemma wfsts_wfsts':
  "wfsts S = wfsts' S [ ]"
<proof> lemma decomp_ik:
  assumes "Ana t = (K,M)"
  shows "ikst (decompose t) = set M"
<proof> lemma decomp_assignment_rhs_empty:
  assumes "Ana t = (K,M)"
  shows "assignment_rhsst (decompose t) = {}"
<proof> lemma decomp_tfrstp:
  "list_all tfrstp (decompose t)"
<proof> lemma trmsest_ikI:
  "t  $\in$  ikest A  $\Rightarrow$  t  $\in$  subtermsset (trmsest A)"
<proof> lemma trmsest_ik_assignment_rhsI:
  "t  $\in$  ikest A  $\cup$  assignment_rhsest A  $\Rightarrow$  t  $\in$  subtermsset (trmsest A)"
<proof> lemma trmsest_ik_subtermsI:

```

```

assumes "t ∈ subtermsset (ikest A)"
shows "t ∈ subtermsset (trmsest A)"
⟨proof⟩ lemma trmsestD:
  assumes "t ∈ trmsest A"
  shows "t ∈ trmsst (tost A)"
⟨proof⟩ lemma subst_apply_extstrand_nil[simp]:
  "[ ] ·est ∅ = [ ]"
⟨proof⟩ lemma subst_apply_extstrand_singleton[simp]:
  "[Step (receive(t)st)] ·est ∅ = [Step (Receive (t · ∅))]"
  "[Step (send(t)st)] ·est ∅ = [Step (Send (t · ∅))]"
  "[Step (⟨a: t ≐ t'⟩st)] ·est ∅ = [Step (Equality a (t · ∅) (t' · ∅))]"
  "[Decomp t] ·est ∅ = [Decomp (t · ∅)]"
⟨proof⟩ lemma extstrand_subst_hom:
  "(S@S') ·est ∅ = (S ·est ∅)@(S' ·est ∅)" "(x#S) ·est ∅ = (x ·estp ∅)#(S ·est ∅)"
⟨proof⟩ lemma decomp_vars:
  "wfrestrictedvarsst (decomp t) = fv t" "varsst (decomp t) = fv t" "bvarsst (decomp t) = {}"
  "fvst (decomp t) = fv t"
⟨proof⟩ lemma bvarsest_cons: "bvarsest (x#X) = bvarsest [x] ∪ bvarsest X"
⟨proof⟩ lemma bvarsest_append: "bvarsest (A@B) = bvarsest A ∪ bvarsest B"
⟨proof⟩ lemma fvest_cons: "fvest (x#X) = fvest [x] ∪ fvest X"
⟨proof⟩ lemma fvest_append: "fvest (A@B) = fvest A ∪ fvest B"
⟨proof⟩ lemma bvars_decomp: "bvarsest (A@[Decomp t]) = bvarsest A" "bvarsest (Decomp t#A) = bvarsest A"
⟨proof⟩ lemma bvars_decomp_rm: "bvarsest (decomp_rmest A) = bvarsest A"
⟨proof⟩ lemma fv_decomp_rm: "fvest (decomp_rmest A) ⊆ fvest A"
⟨proof⟩ lemma ik_assignment_rhs_decomp_fv:
  assumes "t ∈ subtermsset (ikest A ∪ assignment_rhsest A)"
  shows "fvest (A@[Decomp t]) = fvest A"
⟨proof⟩ lemma wfrestrictedvarsest_decomp_rmest_subset:
  "wfrestrictedvarsest (decomp_rmest A) ⊆ wfrestrictedvarsest A"
⟨proof⟩ lemma wfrestrictedvarsest_eq_wfrestrictedvarsst:
  "wfrestrictedvarsest A = wfrestrictedvarsst (tost A)"
⟨proof⟩ lemma decomp_set_unfold:
  assumes "Ana t = (K, M)"
  shows "set (decomp t) = {send(t)st} ∪ (Send ` set K) ∪ (Receive ` set M)"
⟨proof⟩ lemma ikest_finite: "finite (ikest A)"
⟨proof⟩ lemma assignment_rhsest_finite: "finite (assignment_rhsest A)"
⟨proof⟩ lemma toest_append: "toest (A@B) = toest A@toest B"
⟨proof⟩ lemma tost_toest_inv: "tost (toest A) = A"
⟨proof⟩ lemma tost_append: "tost (A@B) = (tost A)@(tost B)"
⟨proof⟩ lemma tost_cons: "tost (a#B) = (tost [a])@(tost B)"
⟨proof⟩ lemma wfrestrictedvarsest_split:
  "wfrestrictedvarsest (x#S) = wfrestrictedvarsest [x] ∪ wfrestrictedvarsest S"
  "wfrestrictedvarsest (S@S') = wfrestrictedvarsest S ∪ wfrestrictedvarsest S'"
⟨proof⟩ lemma ikest_append: "ikest (A@B) = ikest A ∪ ikest B"
⟨proof⟩ lemma assignment_rhsest_append:
  "assignment_rhsest (A@B) = assignment_rhsest A ∪ assignment_rhsest B"
⟨proof⟩ lemma ikest_cons: "ikest (a#A) = ikest [a] ∪ ikest A"
⟨proof⟩ lemma ikest_append_subst:
  "ikest (A@B ·est ∅) = ikest (A ·est ∅) ∪ ikest (B ·est ∅)"
  "ikest (A@B) ·set ∅ = (ikest A ·set ∅) ∪ (ikest B ·set ∅)"
⟨proof⟩ lemma assignment_rhsest_append_subst:
  "assignment_rhsest (A@B ·est ∅) = assignment_rhsest (A ·est ∅) ∪ assignment_rhsest (B ·est ∅)"
  "assignment_rhsest (A@B) ·set ∅ = (assignment_rhsest A ·set ∅) ∪ (assignment_rhsest B ·set ∅)"
⟨proof⟩ lemma ikest_cons_subst:
  "ikest (a#A ·est ∅) = ikest ([a ·estp ∅]) ∪ ikest (A ·est ∅)"
  "ikest (a#A) ·set ∅ = (ikest [a] ·set ∅) ∪ (ikest A ·set ∅)"
⟨proof⟩ lemma decomp_rmest_append: "decomp_rmest (S@S') = (decomp_rmest S)@(decomp_rmest S)"
⟨proof⟩ lemma decomp_rmest_single[simp]:
  "decomp_rmest [Step (send(t)st)] = [Step (send(t)st)]"
  "decomp_rmest [Step (receive(t)st)] = [Step (receive(t)st)]"
  "decomp_rmest [Decomp t] = [ ]"
⟨proof⟩ lemma decomp_rmest_ik_subset: "ikest (decomp_rmest S) ⊆ ikest S"
⟨proof⟩ lemma decompest_ik_subset: "D ∈ decompest M N I ⇒ ikest D ⊆ subtermsset (M ∪ N)"

```

```

<proof> lemma decompest_decomprmest_empty: "D ∈ decompest M N I ⇒ decomprmest D = []"
<proof> lemma decompest_append:
  assumes "A ∈ decompest S N I" "B ∈ decompest S N I"
  shows "A@B ∈ decompest S N I"
<proof> lemma decompest_subterms:
  assumes "A' ∈ decompest M N I"
  shows "subtermsset (ikest A') ⊆ subtermsset (M ∪ N)"
<proof> lemma decompest_assignmentrhs_empty:
  assumes "A' ∈ decompest M N I"
  shows "assignmentrhsest A' = {}"
<proof> lemma decompest_finiteik_append:
  assumes "finite M" "M ⊆ decompest A N I"
  shows "∃D ∈ decompest A N I. ikest D = (∪m ∈ M. ikest m)"
<proof> lemma decomp_snd_exists[simp]: "∃D. decomp t = send(t)st#D"
<proof> lemma decomp_nonnil[simp]: "decomp t ≠ []"
<proof> lemma tost_nil_inv[dest]: "tost A = [] ⇒ A = []"
<proof> lemma well_analyzedD:
  assumes "well_analyzed A" "Decomp t ∈ set A"
  shows "∃f T. t = Fun f T"
<proof> lemma well_analyzed_inv:
  assumes "well_analyzed (A@[Decomp t])"
  shows "t ∈ subtermsset (ikest A ∪ assignmentrhsest A) - (Var ` V)"
<proof> lemma well_analyzed_split_left_single: "well_analyzed (A@[a]) ⇒ well_analyzed A"
<proof> lemma well_analyzed_split_left: "well_analyzed (A@B) ⇒ well_analyzed A"
<proof> lemma well_analyzed_append:
  assumes "well_analyzed A" "well_analyzed B"
  shows "well_analyzed (A@B)"
<proof> lemma well_analyzed_singleton:
  "well_analyzed [Step (send(t)st)]" "well_analyzed [Step (receive(t)st)]"
  "well_analyzed [Step (<a: t ≐ t'>st)]" "well_analyzed [Step (∀X(∀≠: F)st)]"
  "¬well_analyzed [Decomp t]"
<proof> lemma well_analyzed_decomprmest_fv: "well_analyzed A ⇒ fvest (decomprmest A) = fvest A"
<proof> lemma semest_d_split_left: assumes "semest_d M0 I (A@A'" shows "semest_d M0 I A"
<proof> lemma semest_d_eq_semst: "semest_d M0 I A = [M0; tost A]d' I"
<proof> lemma semest_c_eq_semst: "semest_c M0 I A = [M0; tost A]c' I"
<proof> lemma semest_c_decomprmest_deduct_aux:
  assumes "semest_c M0 I A" "t ∈ ikest A ·set I" "t ∉ ikest (decomprmest A) ·set I"
  shows "ikest (decomprmest A) ∪ M0 ·set I ⊢ t"
<proof> lemma semest_c_decomprmest_deduct:
  assumes "semest_c M0 I A" "ikest A ∪ M0 ·set I ⊢c t"
  shows "ikest (decomprmest A) ∪ M0 ·set I ⊢ t"
<proof> lemma semest_d_decomprmest_ifsemest_c: "semest_c M0 I A ⇒ semest_d M0 I (decomprmest A)"
<proof> lemma semest_c_decompest_append:
  assumes "semest_c {} I A" "D ∈ decompest (ikest A) (assignmentrhsest A) I"
  shows "semest_c {} I (A@D)"
<proof> lemma decompest_preserves_wf:
  assumes "D ∈ decompest (ikest A) (assignmentrhsest A) I" "wfest V A"
  shows "wfest V (A@D)"
<proof> lemma decompest_preserves_model_c:
  assumes "D ∈ decompest (ikest A) (assignmentrhsest A) I" "semest_c M0 I A"
  shows "semest_c M0 I (A@D)"
<proof> lemma decompest_exist_aux:
  assumes "D ∈ decompest M N I" "M ∪ ikest D ⊢ t" "¬(M ∪ (ikest D) ⊢c t)"
  obtains D' where
    "D@D' ∈ decompest M N I" "M ∪ ikest (D@D') ⊢c t" "M ∪ ikest D ⊆ M ∪ ikest (D@D'"
<proof> lemma decompest_ik_max_exist:
  assumes "finite A" "finite N"
  shows "∃D ∈ decompest A N I. ∀D' ∈ decompest A N I. ikest D' ⊆ ikest D"
<proof> lemma decompest_exist:
  assumes "finite A" "finite N"
  shows "∃D ∈ decompest A N I. ∀t. A ⊢ t → A ∪ ikest D ⊢c t"
<proof> lemma decompest_exist_subst:
  assumes "ikest A ·set I ⊢ t · I"

```

and "sem_{est_c} {} \mathcal{I} A" "wf_{est} {} A" "interpretation_{subst} \mathcal{I} "
 and "Ana_invar_subst (ik_{est} A \cup assignment_rhs_{est} A)"
 and "well_analyzed A"
 shows " $\exists D \in \text{decomps}_{est}$ (ik_{est} A) (assignment_rhs_{est} A) \mathcal{I} . ik_{est} (A@D) \cdot_{set} $\mathcal{I} \vdash_c t \cdot \mathcal{I}$ "
 <proof> lemma wf_{sts}'_update_{st}_nil: assumes "wf_{sts}' S A" shows "wf_{sts}' (update_{st} S []) A"
 <proof> lemma wf_{sts}'_update_{st}_snd:
 assumes "wf_{sts}' S A" "send(t)_{st}#S \in S"
 shows "wf_{sts}' (update_{st} S (send(t)_{st}#S)) (A@[Step (receive(t)_{st})])"
 <proof> lemma wf_{sts}'_update_{st}_rcv:
 assumes "wf_{sts}' S A" "receive(t)_{st}#S \in S"
 shows "wf_{sts}' (update_{st} S (receive(t)_{st}#S)) (A@[Step (send(t)_{st})])"
 <proof> lemma wf_{sts}'_update_{st}_eq:
 assumes "wf_{sts}' S A" " $\langle a: t \doteq t' \rangle_{st}\#S \in S$ "
 shows "wf_{sts}' (update_{st} S ($\langle a: t \doteq t' \rangle_{st}\#S$)) (A@[Step ($\langle a: t \doteq t' \rangle_{st}$)])"
 <proof> lemma wf_{sts}'_update_{st}_ineq:
 assumes "wf_{sts}' S A" " $\forall X(\bigvee \neq: F)_{st}\#S \in S$ "
 shows "wf_{sts}' (update_{st} S ($\forall X(\bigvee \neq: F)_{st}\#S$)) (A@[Step ($\forall X(\bigvee \neq: F)_{st}$)])"
 <proof> lemma trms_{st}_update_{st}_eq:
 assumes "x#S \in S"
 shows " \bigcup (trms_{st} \setminus update_{st} S (x#S)) \cup trms_{stp} x = \bigcup (trms_{st} \setminus S)" (is "?A = ?B")
 <proof> lemma trms_{st}_update_{st}_eq_snd:
 assumes "send(t)_{st}#S \in S" "S' = update_{st} S (send(t)_{st}#S)" "A' = A@[Step (receive(t)_{st})]"
 shows " \bigcup (trms_{st} \setminus S) \cup (trms_{est} A) = (\bigcup (trms_{st} \setminus S')) \cup (trms_{est} A')"
 <proof> lemma trms_{st}_update_{st}_eq_rcv:
 assumes "receive(t)_{st}#S \in S" "S' = update_{st} S (receive(t)_{st}#S)" "A' = A@[Step (send(t)_{st})]"
 shows " \bigcup (trms_{st} \setminus S) \cup (trms_{est} A) = (\bigcup (trms_{st} \setminus S')) \cup (trms_{est} A')"
 <proof> lemma trms_{st}_update_{st}_eq_eq:
 assumes " $\langle a: t \doteq t' \rangle_{st}\#S \in S$ " "S' = update_{st} S ($\langle a: t \doteq t' \rangle_{st}\#S$)" "A' = A@[Step ($\langle a: t \doteq t' \rangle_{st}$)]"
 shows " \bigcup (trms_{st} \setminus S) \cup (trms_{est} A) = (\bigcup (trms_{st} \setminus S')) \cup (trms_{est} A')"
 <proof> lemma trms_{st}_update_{st}_eq_ineq:
 assumes " $\forall X(\bigvee \neq: F)_{st}\#S \in S$ " "S' = update_{st} S ($\forall X(\bigvee \neq: F)_{st}\#S$)" "A' = A@[Step ($\forall X(\bigvee \neq: F)_{st}$)]"
 shows " \bigcup (trms_{st} \setminus S) \cup (trms_{est} A) = (\bigcup (trms_{st} \setminus S')) \cup (trms_{est} A')"
 <proof> lemma ik_{st}_update_{st}_subset:
 assumes "x#S \in S"
 shows " \bigcup (ik_{st} \setminus dual_{st} \setminus (update_{st} S (x#S))) \subseteq \bigcup (ik_{st} \setminus dual_{st} \setminus S)" (is ?A)
 " \bigcup (assignment_rhs_{st} \setminus (update_{st} S (x#S))) \subseteq \bigcup (assignment_rhs_{st} \setminus S)" (is ?B)
 <proof> lemma ik_{st}_update_{st}_subset_snd:
 assumes "send(t)_{st}#S \in S"
 "S' = update_{st} S (send(t)_{st}#S)"
 "A' = A@[Step (receive(t)_{st})]"
 shows " \bigcup (ik_{st} \setminus dual_{st} \setminus S') \cup (ik_{est} A') \subseteq
 (\bigcup (ik_{st} \setminus dual_{st} \setminus S)) \cup (ik_{est} A)" (is ?A)
 " \bigcup (assignment_rhs_{st} \setminus S') \cup (assignment_rhs_{est} A') \subseteq
 (\bigcup (assignment_rhs_{st} \setminus S)) \cup (assignment_rhs_{est} A)" (is ?B)
 <proof> lemma ik_{st}_update_{st}_subset_rcv:
 assumes "receive(t)_{st}#S \in S"
 "S' = update_{st} S (receive(t)_{st}#S)"
 "A' = A@[Step (send(t)_{st})]"
 shows " \bigcup (ik_{st} \setminus dual_{st} \setminus S') \cup (ik_{est} A') \subseteq
 (\bigcup (ik_{st} \setminus dual_{st} \setminus S)) \cup (ik_{est} A)" (is ?A)
 " \bigcup (assignment_rhs_{st} \setminus S') \cup (assignment_rhs_{est} A') \subseteq
 (\bigcup (assignment_rhs_{st} \setminus S)) \cup (assignment_rhs_{est} A)" (is ?B)
 <proof> lemma ik_{st}_update_{st}_subset_eq:
 assumes " $\langle a: t \doteq t' \rangle_{st}\#S \in S$ "
 "S' = update_{st} S ($\langle a: t \doteq t' \rangle_{st}\#S$)"
 "A' = A@[Step ($\langle a: t \doteq t' \rangle_{st}$)]"
 shows " \bigcup (ik_{st} \setminus dual_{st} \setminus S') \cup (ik_{est} A') \subseteq
 (\bigcup (ik_{st} \setminus dual_{st} \setminus S)) \cup (ik_{est} A)" (is ?A)
 " \bigcup (assignment_rhs_{st} \setminus S') \cup (assignment_rhs_{est} A') \subseteq
 (\bigcup (assignment_rhs_{st} \setminus S)) \cup (assignment_rhs_{est} A)" (is ?B)
 <proof> lemma ik_{st}_update_{st}_subset_ineq:
 assumes " $\forall X(\bigvee \neq: F)_{st}\#S \in S$ "
 "S' = update_{st} S ($\forall X(\bigvee \neq: F)_{st}\#S$)"

```

    "A' = A@[Step (∀X(∀≠: F)st)]"
  shows "(⋃ (ikst `dualst ` S')) ∪ (ikest A') ⊆
        (⋃ (ikst `dualst ` S)) ∪ (ikest A)" (is ?A)
    "(⋃ (assignment_rhsst ` S')) ∪ (assignment_rhsest A') ⊆
        (⋃ (assignment_rhsst ` S)) ∪ (assignment_rhsest A)" (is ?B)
<proof>

```

Transition Systems Definitions

```

inductive pts_symbolic::
  "((fun,'var) strands × (fun,'var) strand) ⇒
   ((fun,'var) strands × (fun,'var) strand) ⇒ bool"
(infix "⇒•" 50) where
  Nil[simp]:      "[ ] ∈ S ⇒ (S,A) ⇒• (updatest S [ ],A)"
  Send[simp]:    "send⟨t⟩st#S ∈ S ⇒ (S,A) ⇒• (updatest S (send⟨t⟩st#S),A@[receive⟨t⟩st])"
  Receive[simp]: "receive⟨t⟩st#S ∈ S ⇒ (S,A) ⇒• (updatest S (receive⟨t⟩st#S),A@[send⟨t⟩st])"
  Equality[simp]: "⟨a: t ≐ t'⟩st#S ∈ S ⇒ (S,A) ⇒• (updatest S (⟨a: t ≐ t'⟩st#S),A@[⟨a: t ≐ t'⟩st])"
  Inequality[simp]: "∀X(∀≠: F)st#S ∈ S ⇒ (S,A) ⇒• (updatest S (∀X(∀≠: F)st#S),A@[∀X(∀≠: F)st])"

private inductive pts_symbolic_c::
  "((fun,'var) strands × (fun,'var) extstrand) ⇒
   ((fun,'var) strands × (fun,'var) extstrand) ⇒ bool"
(infix "⇒•c" 50) where
  Nil[simp]:      "[ ] ∈ S ⇒ (S,A) ⇒•c (updatest S [ ],A)"
  Send[simp]:    "send⟨t⟩st#S ∈ S ⇒ (S,A) ⇒•c (updatest S (send⟨t⟩st#S),A@[Step (receive⟨t⟩st)])"
  Receive[simp]: "receive⟨t⟩st#S ∈ S ⇒ (S,A) ⇒•c (updatest S (receive⟨t⟩st#S),A@[Step
  (send⟨t⟩st)])"
  Equality[simp]: "⟨a: t ≐ t'⟩st#S ∈ S ⇒ (S,A) ⇒•c (updatest S (⟨a: t ≐ t'⟩st#S),A@[Step (⟨a: t ≐ t'⟩st)])"
  Inequality[simp]: "∀X(∀≠: F)st#S ∈ S ⇒ (S,A) ⇒•c (updatest S (∀X(∀≠: F)st#S),A@[Step (∀X(∀≠:
  F)st)])"
  Decompose[simp]: "Fun f T ∈ subtermsset (ikest A ∪ assignment_rhsest A)
    ⇒ (S,A) ⇒•c (S,A@[Decomp (Fun f T)])"

```

```

abbreviation pts_symbolic_rtrancl (infix "⇒•*" 50) where "a ⇒•* b ≡ pts_symbolic•* a b"
private abbreviation pts_symbolic_c_rtrancl (infix "⇒•c*" 50) where "a ⇒•c* b ≡ pts_symbolic_c•* a b"

```

```

lemma pts_symbolic_induct[consumes 1, case_names Nil Send Receive Equality Inequality]:
  assumes "(S,A) ⇒• (S',A')"
  and "[ ] ∈ S; S' = updatest S [ ]; A' = A" ⇒ P"
  and "∀t S. [send⟨t⟩st#S ∈ S; S' = updatest S (send⟨t⟩st#S); A' = A@[receive⟨t⟩st]] ⇒ P"
  and "∀t S. [receive⟨t⟩st#S ∈ S; S' = updatest S (receive⟨t⟩st#S); A' = A@[send⟨t⟩st]] ⇒ P"
  and "∀a t t' S. [⟨a: t ≐ t'⟩st#S ∈ S; S' = updatest S (⟨a: t ≐ t'⟩st#S); A' = A@[⟨a: t ≐ t'⟩st]]
  ⇒ P"
  and "∀X F S. [∀X(∀≠: F)st#S ∈ S; S' = updatest S (∀X(∀≠: F)st#S); A' = A@[∀X(∀≠: F)st]] ⇒ P"
  shows "P"
<proof> lemma pts_symbolic_c_induct[consumes 1, case_names Nil Send Receive Equality Inequality
Decompose]:
  assumes "(S,A) ⇒•c (S',A')"
  and "[ ] ∈ S; S' = updatest S [ ]; A' = A" ⇒ P"
  and "∀t S. [send⟨t⟩st#S ∈ S; S' = updatest S (send⟨t⟩st#S); A' = A@[Step (receive⟨t⟩st)]] ⇒ P"
  and "∀t S. [receive⟨t⟩st#S ∈ S; S' = updatest S (receive⟨t⟩st#S); A' = A@[Step (send⟨t⟩st)]] ⇒ P"
  and "∀a t t' S. [⟨a: t ≐ t'⟩st#S ∈ S; S' = updatest S (⟨a: t ≐ t'⟩st#S); A' = A@[Step (⟨a: t ≐ t'⟩st)]]
  ⇒ P"
  and "∀X F S. [∀X(∀≠: F)st#S ∈ S; S' = updatest S (∀X(∀≠: F)st#S); A' = A@[Step (∀X(∀≠: F)st)]]
  ⇒ P"
  and "∀f T. [Fun f T ∈ subtermsset (ikest A ∪ assignment_rhsest A); S' = S; A' = A@[Decomp (Fun f T)]]
  ⇒ P"
  shows "P"
<proof> lemma pts_symbolic_c_preserves_wf_prot:
  assumes "(S,A) ⇒•c* (S',A')" "wfsts' S A"

```

```

shows "wf_sts' S' A'"
<proof> lemma pts_symbolic_c_preserves_wf_is:
  assumes "(S,A) ⇒•c* (S',A')" "wf_sts' S A" "wf_st V (to_st A)"
  shows "wf_st V (to_st A)'"
<proof> lemma pts_symbolic_c_preserves_tfr_set:
  assumes "(S,A) ⇒•c* (S',A)'"
    and "tfr_set ((⋃(trms_st ` S)) ∪ (trms_est A))"
    and "wf_trms ((⋃(trms_st ` S)) ∪ (trms_est A))"
  shows "tfr_set ((⋃(trms_st ` S')) ∪ (trms_est A')) ∧ wf_trms ((⋃(trms_st ` S')) ∪ (trms_est A'))"
<proof> lemma pts_symbolic_c_preserves_tfr_stp:
  assumes "(S,A) ⇒•c* (S',A)'" "∀S ∈ S ∪ {to_st A}. list_all tfr_stp S"
  shows "∀S ∈ S' ∪ {to_st A'}. list_all tfr_stp S"
<proof> lemma pts_symbolic_c_preserves_well_analyzed:
  assumes "(S,A) ⇒•c* (S',A)'" "well_analyzed A"
  shows "well_analyzed A'"
<proof> lemma pts_symbolic_c_preserves_Ana_invar_subst:
  assumes "(S,A) ⇒•c* (S',A)'"
    and "Ana_invar_subst (
      ⋃(ik_st ` dual_st ` S) ∪ (ik_est A) ∪
      ⋃(assignment_rhs_st ` S) ∪ (assignment_rhs_est A))"
  shows "Ana_invar_subst (
      ⋃(ik_st ` dual_st ` S') ∪ (ik_est A') ∪
      ⋃(assignment_rhs_st ` S') ∪ (assignment_rhs_est A'))"
<proof> lemma pts_symbolic_c_preserves_constr_disj_vars:
  assumes "(S,A) ⇒•c* (S',A)'" "wf_sts' S A" "fv_est A ∩ bvars_est A = {}"
  shows "fv_est A' ∩ bvars_est A' = {}"
<proof>

```

Theorem: The Typing Result Lifted to the Transition System Level

```

private lemma wf_sts'_decomp_rm:
  assumes "well_analyzed A" "wf_sts' S (decomp_rm_est A)" shows "wf_sts' S A"
<proof> lemma decomp_est_pts_symbolic_c:
  assumes "D ∈ decomp_est (ik_est A) (assignment_rhs_est A) I"
  shows "(S,A) ⇒•c* (S,A@D)"
<proof> lemma pts_symbolic_to_pts_symbolic_c:
  assumes "(S,to_st (decomp_rm_est A_d)) ⇒•* (S',A)'" "sem_est_d {} I (to_est A)'" "sem_est_c {} I A_d"
  and wf: "wf_sts' S (decomp_rm_est A_d)" "wf_est {} A_d"
  and tar: "Ana_invar_subst ((⋃(ik_st ` dual_st ` S) ∪ (ik_est A_d))
    ∪ (⋃(assignment_rhs_st ` S) ∪ (assignment_rhs_est A_d)))"
  and wa: "well_analyzed A_d"
  and I: "interpretation_subst I"
  shows "∃A_d'. A' = to_st (decomp_rm_est A_d') ∧ (S,A_d) ⇒•c* (S',A_d') ∧ sem_est_c {} I A_d'"
<proof> lemma pts_symbolic_c_to_pts_symbolic:
  assumes "(S,A) ⇒•c* (S',A)'" "sem_est_c {} I A'"
  shows "(S,to_st (decomp_rm_est A)) ⇒•* (S',to_st (decomp_rm_est A'))"
    "sem_est_d {} I (decomp_rm_est A)'"
<proof> lemma pts_symbolic_to_pts_symbolic_c_from_initial:
  assumes "(S_0,[]) ⇒•* (S,A)" "I ⊨ ⟨A⟩" "wf_sts' S_0 []"
  and "Ana_invar_subst (⋃(ik_st ` dual_st ` S_0) ∪ ⋃(assignment_rhs_st ` S_0))" "interpretation_subst I"
  shows "∃A_d. A = to_st (decomp_rm_est A_d) ∧ (S_0,[]) ⇒•c* (S,A_d) ∧ (I ⊨_c ⟨to_st A_d⟩)"
<proof> lemma pts_symbolic_c_to_pts_symbolic_from_initial:
  assumes "(S_0,[]) ⇒•c* (S,A)" "I ⊨_c ⟨to_st A⟩"
  shows "(S_0,[]) ⇒•* (S,to_st (decomp_rm_est A))" "I ⊨ ⟨to_st (decomp_rm_est A)⟩"
<proof> lemma to_st_trms_wf:
  assumes "wf_trms (trms_est A)"
  shows "wf_trms (trms_st (to_st A))"
<proof> lemma to_st_trms_SMP_subset: "trms_st (to_st A) ⊆ SMP (trms_est A)"
<proof> lemma to_st_trms_tfr_set:
  assumes "tfr_set (trms_est A)"
  shows "tfr_set (trms_st (to_st A))"
<proof>

```

```

theorem wt_attack_if_tfr_attack_pts:
  assumes "wfsts S0" "tfrset (⋃(trmsst ` S0))" "wftrms (⋃(trmsst ` S0))" "∀S ∈ S0. list_all tfrstp S"
  and "Ana_invar_subst (⋃(ikst ` dualst ` S0) ∪ ⋃(assignment_rhsst ` S0))"
  and "(S0, []) ⇒* (S, A)" "interpretationsubst I" "I ⊨ ⟨A, Var⟩"
  shows "∃Iτ. interpretationsubst Iτ ∧ (Iτ ⊨ ⟨A, Var⟩) ∧ wtsubst Iτ ∧ wftrms (subst_range Iτ)"
⟨proof⟩

```

Corollary: The Typing Result on the Level of Constraints

There exists well-typed models of satisfiable type-flaw resistant constraints

```

corollary wt_attack_if_tfr_attack_d:
  assumes "wfst {A}" "fvst A ∩ bvarsst A = {}" "tfrst A" "wftrms (trmsst A)"
  and "Ana_invar_subst (ikst A ∪ assignment_rhsst A)"
  and "interpretationsubst I" "I ⊨ ⟨A⟩"
  shows "∃Iτ. interpretationsubst Iτ ∧ (Iτ ⊨ ⟨A⟩) ∧ wtsubst Iτ ∧ wftrms (subst_range Iτ)"
⟨proof⟩

```

end

end

end

4 The Typing Result for Stateful Protocols

In this chapter, we lift the typing result to stateful protocols. For more details, we refer the reader to [3] and [1, chapter 4].

4.1 Stateful Strands (Stateful_Strands)

```
theory Stateful_Strands
imports Strands_and_Constraints
begin
```

4.1.1 Stateful Constraints

```
datatype (funssstp: 'a, varssstp: 'b) stateful_strand_step =
  Send (the_msg: "('a,'b) term") ("send⟨_⟩" 80)
| Receive (the_msg: "('a,'b) term") ("receive⟨_⟩" 80)
| Equality (the_check: poscheckvariant) (the_lhs: "('a,'b) term") (the_rhs: "('a,'b) term")
  ("⟨_ : _ ≐ _⟩" [80,80])
| Insert (the_elem_term: "('a,'b) term") (the_set_term: "('a,'b) term") ("insert⟨_,_⟩" 80)
| Delete (the_elem_term: "('a,'b) term") (the_set_term: "('a,'b) term") ("delete⟨_,_⟩" 80)
| InSet (the_check: poscheckvariant) (the_elem_term: "('a,'b) term") (the_set_term: "('a,'b) term")
  ("⟨_ : _ ∈ _⟩" [80,80])
| NegChecks (bvarssstp: "'b list")
  (the_eqs: "((('a,'b) term × ('a,'b) term) list)")
  (the_ins: "((('a,'b) term × ('a,'b) term) list)")
  ("∀_⟨∀≠: _ ∨∉: _⟩" [80,80])
where
  "bvarssstp (Send _) = []"
| "bvarssstp (Receive _) = []"
| "bvarssstp (Equality _ _ _) = []"
| "bvarssstp (Insert _ _) = []"
| "bvarssstp (Delete _ _) = []"
| "bvarssstp (InSet _ _ _) = []"

type_synonym ('a,'b) stateful_strand = "('a,'b) stateful_strand_step list"
type_synonym ('a,'b) dbstatelist = "((('a,'b) term × ('a,'b) term) list)"
type_synonym ('a,'b) dbstate = "((('a,'b) term × ('a,'b) term) set)"

abbreviation
  "is_Assignment x ≡ (is_Equality x ∨ is_InSet x) ∧ the_check x = Assign"

abbreviation
  "is_Check x ≡ ((is_Equality x ∨ is_InSet x) ∧ the_check x = Check) ∨ is_NegChecks x"

abbreviation
  "is_Update x ≡ is_Insert x ∨ is_Delete x"

abbreviation InSet_select ("select⟨_,_⟩") where "select⟨t,s⟩ ≡ InSet Assign t s"
abbreviation InSet_check ("⟨_ in _⟩") where "⟨t in s⟩ ≡ InSet Check t s"
abbreviation Equality_assign ("⟨_ := _⟩") where "⟨t := s⟩ ≡ Equality Assign t s"
abbreviation Equality_check ("⟨_ == _⟩") where "⟨t == s⟩ ≡ Equality Check t s"

abbreviation NegChecks_Inequality1 ("⟨_ != _⟩") where
  "⟨t != s⟩ ≡ NegChecks [] [(t,s)] []"

abbreviation NegChecks_Inequality2 ("∀_⟨_ != _⟩") where
```

4 The Typing Result for Stateful Protocols

" $\forall x(t \neq s) \equiv \text{NegChecks } [x] [(t,s)] []$ "

abbreviation *NegChecks_Inequality3* (" $\forall _ , _ (_ \neq _)$ ") where

" $\forall x,y(t \neq s) \equiv \text{NegChecks } [x,y] [(t,s)] []$ "

abbreviation *NegChecks_Inequality4* (" $\forall _ , _ , _ (_ \neq _)$ ") where

" $\forall x,y,z(t \neq s) \equiv \text{NegChecks } [x,y,z] [(t,s)] []$ "

abbreviation *NegChecks_NotInSet1* (" $_ \text{ not in } _$ ") where

" $\langle t \text{ not in } s \rangle \equiv \text{NegChecks } [] [] [(t,s)]$ "

abbreviation *NegChecks_NotInSet2* (" $\forall _ (_ \text{ not in } _)$ ") where

" $\forall x(t \text{ not in } s) \equiv \text{NegChecks } [x] [] [(t,s)]$ "

abbreviation *NegChecks_NotInSet3* (" $\forall _ , _ (_ \text{ not in } _)$ ") where

" $\forall x,y(t \text{ not in } s) \equiv \text{NegChecks } [x,y] [] [(t,s)]$ "

abbreviation *NegChecks_NotInSet4* (" $\forall _ , _ , _ (_ \text{ not in } _)$ ") where

" $\forall x,y,z(t \text{ not in } s) \equiv \text{NegChecks } [x,y,z] [] [(t,s)]$ "

fun *trms_{sstp}* where

"*trms_{sstp}* (Send t) = {t}"
 | "*trms_{sstp}* (Receive t) = {t}"
 | "*trms_{sstp}* (Equality _ t t') = {t,t'}"
 | "*trms_{sstp}* (Insert t t') = {t,t'}"
 | "*trms_{sstp}* (Delete t t') = {t,t'}"
 | "*trms_{sstp}* (InSet _ t t') = {t,t'}"
 | "*trms_{sstp}* (NegChecks _ F F') = *trms_{pairs}* F \cup *trms_{pairs}* F'"

definition *trms_{sst}* where "*trms_{sst}* S $\equiv \bigcup$ (*trms_{sstp}* ` set S)"

declare *trms_{sst}_def*[simp]

fun *trms_list_{sstp}* where

"*trms_list_{sstp}* (Send t) = [t]"
 | "*trms_list_{sstp}* (Receive t) = [t]"
 | "*trms_list_{sstp}* (Equality _ t t') = [t,t']"
 | "*trms_list_{sstp}* (Insert t t') = [t,t']"
 | "*trms_list_{sstp}* (Delete t t') = [t,t']"
 | "*trms_list_{sstp}* (InSet _ t t') = [t,t']"
 | "*trms_list_{sstp}* (NegChecks _ F F') = concat (map ($\lambda(t,t'). [t,t']$) (F@F'))"

definition *trms_list_{sst}* where "*trms_list_{sst}* S $\equiv \text{remdups } (\text{concat } (\text{map } \textit{trms_list_sstp } S))$ "

definition *ik_{sst}* where "*ik_{sst}* A $\equiv \{t. \text{Receive } t \in \text{set } A\}$ "

definition *bvars_{sst}* :: "('a, 'b) stateful_strand \Rightarrow 'b set" where

"*bvars_{sst}* S $\equiv \bigcup$ (set (map (set \circ *bvars_{sstp}*) S))"

fun *fv_{sstp}* :: "('a, 'b) stateful_strand_step \Rightarrow 'b set" where

"*fv_{sstp}* (Send t) = fv t"
 | "*fv_{sstp}* (Receive t) = fv t"
 | "*fv_{sstp}* (Equality _ t t') = fv t \cup fv t'"
 | "*fv_{sstp}* (Insert t t') = fv t \cup fv t'"
 | "*fv_{sstp}* (Delete t t') = fv t \cup fv t'"
 | "*fv_{sstp}* (InSet _ t t') = fv t \cup fv t'"
 | "*fv_{sstp}* (NegChecks X F F') = *fv_{pairs}* F \cup *fv_{pairs}* F' - set X"

definition *fv_{sst}* :: "('a, 'b) stateful_strand \Rightarrow 'b set" where

"*fv_{sst}* S $\equiv \bigcup$ (set (map *fv_{sstp}* S))"

fun *fv_list_{sstp}* where

"*fv_list_{sstp}* (send(t)) = *fv_list* t"
 | "*fv_list_{sstp}* (receive(t)) = *fv_list* t"

```

| "fv_listsstp (<_ : t ÷ s) = fv_list t@fv_list s"
| "fv_listsstp (insert<t,s) = fv_list t@fv_list s"
| "fv_listsstp (delete<t,s) = fv_list t@fv_list s"
| "fv_listsstp (<_ : t ∈ s) = fv_list t@fv_list s"
| "fv_listsstp (∀X(∀≠: F ∨≠: F')) = filter (λx. x ∉ set X) (fv_listpairs (F@F'))"

```

definition `fv_listsst` **where**

```
"fv_listsst S ≡ remdups (concat (map fv_listsstp S))"
```

declare `bvarssst_def[simp]`

declare `fvsst_def[simp]`

definition `varssst` :: "('a, 'b) stateful_strand ⇒ 'b set" **where**

```
"varssst S ≡ ⋃ (set (map varssstp S))"
```

abbreviation `wfrestrictedvarssstp` :: "('a, 'b) stateful_strand_step ⇒ 'b set" **where**

```

"wfrestrictedvarssstp x ≡
  case x of
    NegChecks _ _ _ ⇒ {}
  | Equality Check _ _ ⇒ {}
  | InSet Check _ _ ⇒ {}
  | Delete _ _ ⇒ {}
  | _ ⇒ varssstp x"

```

definition `wfrestrictedvarssst` :: "('a, 'b) stateful_strand ⇒ 'b set" **where**

```
"wfrestrictedvarssst S ≡ ⋃ (set (map wfrestrictedvarssstp S))"
```

abbreviation `wfvarsoccssstp` **where**

```

"wfvarsoccssstp x ≡
  case x of
    Send t ⇒ fv t
  | Equality Assign s t ⇒ fv s
  | InSet Assign s t ⇒ fv s ∪ fv t
  | _ ⇒ {}"

```

definition `wfvarsoccssst` **where**

```
"wfvarsoccssst S ≡ ⋃ (set (map wfvarsoccssstp S))"
```

fun `wf'sst` :: "'b set ⇒ ('a, 'b) stateful_strand ⇒ bool" **where**

```

"wf'sst V [] = True"
| "wf'sst V (Receive t#S) = (fv t ⊆ V ∧ wf'sst V S)"
| "wf'sst V (Send t#S) = wf'sst (V ∪ fv t) S"
| "wf'sst V (Equality Assign t t'#S) = (fv t' ⊆ V ∧ wf'sst (V ∪ fv t) S)"
| "wf'sst V (Equality Check _ _#S) = wf'sst V S"
| "wf'sst V (Insert t s#S) = (fv t ⊆ V ∧ fv s ⊆ V ∧ wf'sst V S)"
| "wf'sst V (Delete _ _#S) = wf'sst V S"
| "wf'sst V (InSet Assign t s#S) = wf'sst (V ∪ fv t ∪ fv s) S"
| "wf'sst V (InSet Check _ _#S) = wf'sst V S"
| "wf'sst V (NegChecks _ _ _#S) = wf'sst V S"

```

abbreviation `wfsst S ≡ wf'sst {} S ∧ fvsst S ∩ bvarssst S = {}`

fun `subst_apply_stateful_strand_step` ::

```

"('a, 'b) stateful_strand_step ⇒ ('a, 'b) subst ⇒ ('a, 'b) stateful_strand_step"
(infix ".sstp" 51) where
"send<t> .sstp ϑ = send<t · ϑ>"
| "receive<t> .sstp ϑ = receive<t · ϑ>"
| "⟨a : t ÷ s⟩ .sstp ϑ = ⟨a : (t · ϑ) ÷ (s · ϑ)⟩"
| "⟨a : t ∈ s⟩ .sstp ϑ = ⟨a : (t · ϑ) ∈ (s · ϑ)⟩"
| "insert<t,s> .sstp ϑ = insert<t · ϑ, s · ϑ>"
| "delete<t,s> .sstp ϑ = delete<t · ϑ, s · ϑ>"
| "∀X(∀≠: F ∨≠: G) .sstp ϑ = ∀X(∀≠: (F ·pairs rm_vars (set X) ϑ) ∨≠: (G ·pairs rm_vars (set X) ϑ))"

```

definition `subst_apply_stateful_strand::`

```
"('a,'b) stateful_strand ⇒ ('a,'b) subst ⇒ ('a,'b) stateful_strand"
(infix ".sst" 51) where
"S .sst ∅ ≡ map (λx. x .sstp ∅) S"
```

fun `dbupdsst::`"('f,'v) stateful_strand ⇒ ('f,'v) subst ⇒ ('f,'v) dbstate ⇒ ('f,'v) dbstate"
where

```
"dbupdsst [] I D = D"
| "dbupdsst (Insert t s#A) I D = dbupdsst A I (insert ((t,s) .p I) D)"
| "dbupdsst (Delete t s#A) I D = dbupdsst A I (D - {((t,s) .p I})"
| "dbupdsst (_#A) I D = dbupdsst A I D"
```

fun `db'sst::`"('f,'v) stateful_strand ⇒ ('f,'v) subst ⇒ ('f,'v) dbstatelist ⇒ ('f,'v) dbstatelist"
where

```
"db'sst [] I D = D"
| "db'sst (Insert t s#A) I D = db'sst A I (List.insert ((t,s) .p I) D)"
| "db'sst (Delete t s#A) I D = db'sst A I (List.removeAll ((t,s) .p I) D)"
| "db'sst (_#A) I D = db'sst A I D"
```

definition `dbsst where`

```
"dbsst S I ≡ db'sst S I []"
```

fun `setopssstp where`

```
"setopssstp (Insert t s) = {(t,s)}"
| "setopssstp (Delete t s) = {(t,s)}"
| "setopssstp (InSet _ t s) = {(t,s)}"
| "setopssstp (NegChecks _ _ F') = set F'"
| "setopssstp _ = {}"
```

The set-operations of a stateful strand

definition `setopssst where`

```
"setopssst S ≡ ⋃ (setopssstp ` set S)"
```

fun `setops_listsstp where`

```
"setops_listsstp (Insert t s) = [(t,s)]"
| "setops_listsstp (Delete t s) = [(t,s)]"
| "setops_listsstp (InSet _ t s) = [(t,s)]"
| "setops_listsstp (NegChecks _ _ F') = F'"
| "setops_listsstp _ = []"
```

The set-operations of a stateful strand (list variant)

definition `setops_listsst where`

```
"setops_listsst S ≡ remdups (concat (map setops_listsstp S))"
```

4.1.2 Small Lemmata

lemma `trms_listsst_is_trmssst:` "trms_{sst} S = set (trms_list_{sst} S)"
<proof>

lemma `setops_listsst_is_setopssst:` "setops_{sst} S = set (setops_list_{sst} S)"
<proof>

lemma `fv_listsstp_is_fvsstp:` "fv_{sstp} a = set (fv_list_{sstp} a)"
<proof>

lemma `fv_listsst_is_fvsst:` "fv_{sst} S = set (fv_list_{sst} S)"
<proof>

lemma `trmssstp_finite[simp]:` "finite (trms_{sstp} x)"
<proof>

lemma `trmssst_finite[simp]:` "finite (trms_{sst} S)"
<proof>

```

lemma varssstp_finite[simp]: "finite (varssstp x)"
⟨proof⟩

lemma varssst_finite[simp]: "finite (varssst S)"
⟨proof⟩

lemma fvsstp_finite[simp]: "finite (fvsstp x)"
⟨proof⟩

lemma fvsst_finite[simp]: "finite (fvsst S)"
⟨proof⟩

lemma bvarssstp_finite[simp]: "finite (set (bvarssstp x))"
⟨proof⟩

lemma bvarssst_finite[simp]: "finite (bvarssst S)"
⟨proof⟩

lemma substsst_nil[simp]: "[ ] ·sst δ = [ ]"
⟨proof⟩

lemma dbsst_nil[simp]: "dbsst [ ] I = [ ]"
⟨proof⟩

lemma iksst_nil[simp]: "iksst [ ] = {}"
⟨proof⟩

lemma iksst_append[simp]: "iksst (A@B) = iksst A ∪ iksst B"
⟨proof⟩

lemma iksst_subst: "iksst (A ·sst δ) = iksst A ·set δ"
⟨proof⟩

lemma dbsst_set_is_dbupdsst: "set (db'sst A I D) = dbupdsst A I (set D)" (is "?A = ?B")
⟨proof⟩

lemma dbupdsst_no_upd:
  assumes "∀ a ∈ set A. ¬is_Insert a ∧ ¬is_Delete a"
  shows "dbupdsst A I D = D"
⟨proof⟩

lemma dbsst_no_upd:
  assumes "∀ a ∈ set A. ¬is_Insert a ∧ ¬is_Delete a"
  shows "db'sst A I D = D"
⟨proof⟩

lemma dbsst_no_upd_append:
  assumes "∀ b ∈ set B. ¬is_Insert b ∧ ¬is_Delete b"
  shows "db'sst A = db'sst (A@B)"
⟨proof⟩

lemma dbsst_append:
  "db'sst (A@B) I D = db'sst B I (db'sst A I D)"
⟨proof⟩

lemma dbsst_in_cases:
  assumes "(t,s) ∈ set (db'sst A I D)"
  shows "(t,s) ∈ set D ∨ (∃ t' s'. insert(t',s') ∈ set A ∧ t = t' · I ∧ s = s' · I)"
⟨proof⟩

lemma dbsst_in_cases':
  assumes "(t,s) ∈ set (db'sst A I D)"

```

4 The Typing Result for Stateful Protocols

and " $(t,s) \notin \text{set } D$ "
 shows " $\exists B C t' s'. A = B @ \text{insert}(t',s') \# C \wedge t = t' \cdot I \wedge s = s' \cdot I \wedge$
 $(\forall t'' s''. \text{delete}(t'',s'') \in \text{set } C \longrightarrow t \neq t'' \cdot I \vee s \neq s'' \cdot I)$ "

<proof>

lemma `dbsst_filter`:

"`db'sst A I D = db'sst (filter is_Update A) I D`"

<proof>

lemma `subst_sst_cons`: "`a#A .sst δ = (a .sstp δ)#(A .sst δ)`"

<proof>

lemma `subst_sst_snoc`: "`A@[a] .sst δ = (A .sst δ)@[a .sstp δ]`"

<proof>

lemma `subst_sst_append[simp]`: "`A@B .sst δ = (A .sst δ)@(B .sst δ)`"

<proof>

lemma `sst_vars_append_subset`:

"`fvsst A ⊆ fvsst (A@B)`" "`bvarssst A ⊆ bvarssst (A@B)`"

"`fvsst B ⊆ fvsst (A@B)`" "`bvarssst B ⊆ bvarssst (A@B)`"

<proof>

lemma `sst_vars_disj_cons[simp]`: "`fvsst (a#A) ∩ bvarssst (a#A) = {}`" \implies "`fvsst A ∩ bvarssst A = {}`"

<proof>

lemma `fvsst_cons_subset[simp]`: "`fvsst A ⊆ fvsst (a#A)`"

<proof>

lemma `fvsstp_subst_cases[simp]`:

"`fvsstp (send(t) .sstp ϑ) = fv (t · ϑ)`"

"`fvsstp (receive(t) .sstp ϑ) = fv (t · ϑ)`"

"`fvsstp ((c: t ≐ s) .sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)`"

"`fvsstp (insert(t,s) .sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)`"

"`fvsstp (delete(t,s) .sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)`"

"`fvsstp ((c: t ∈ s) .sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)`"

"`fvsstp (∀X(∀≠: F ∨≠: G) .sstp ϑ) =`

`fvpairs (F .pairs rm_vars (set X) ϑ) ∪ fvpairs (G .pairs rm_vars (set X) ϑ) - set X`"

<proof>

lemma `varssstp_cases[simp]`:

"`varssstp (send(t)) = fv t`"

"`varssstp (receive(t)) = fv t`"

"`varssstp ((c: t ≐ s)) = fv t ∪ fv s`"

"`varssstp (insert(t,s)) = fv t ∪ fv s`"

"`varssstp (delete(t,s)) = fv t ∪ fv s`"

"`varssstp ((c: t ∈ s)) = fv t ∪ fv s`"

"`varssstp (∀X(∀≠: F ∨≠: G)) = fvpairs F ∪ fvpairs G ∪ set X`" (is ?A)

"`varssstp (∀X(∀≠: [(t,s)] ∨≠: [])) = fv t ∪ fv s ∪ set X`" (is ?B)

"`varssstp (∀X(∀≠: [] ∨≠: [(t,s)])) = fv t ∪ fv s ∪ set X`" (is ?C)

<proof>

lemma `varssstp_subst_cases[simp]`:

"`varssstp (send(t) .sstp ϑ) = fv (t · ϑ)`"

"`varssstp (receive(t) .sstp ϑ) = fv (t · ϑ)`"

"`varssstp ((c: t ≐ s) .sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)`"

"`varssstp (insert(t,s) .sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)`"

"`varssstp (delete(t,s) .sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)`"

"`varssstp ((c: t ∈ s) .sstp ϑ) = fv (t · ϑ) ∪ fv (s · ϑ)`"

"`varssstp (∀X(∀≠: F ∨≠: G) .sstp ϑ) =`

`fvpairs (F .pairs rm_vars (set X) ϑ) ∪ fvpairs (G .pairs rm_vars (set X) ϑ) ∪ set X`" (is ?A)

"`varssstp (∀X(∀≠: [(t,s)] ∨≠: [])) .sstp ϑ) =`

`fv (t · rm_vars (set X) ϑ) ∪ fv (s · rm_vars (set X) ϑ) ∪ set X`" (is ?B)

"vars_{sstp} ($\forall X(\forall \neq: [] \vee \notin: [(t,s)]) \cdot_{sstp} \vartheta$) =
 fv ($t \cdot rm_vars$ (set X) ϑ) \cup fv ($s \cdot rm_vars$ (set X) ϑ) \cup set X" (is ?C)
 <proof>

lemma bvars_{sst}_cons_subset: "bvars_{sst} A \subseteq bvars_{sst} (a#A)"
 <proof>

lemma bvars_{sstp}_subst: "bvars_{sstp} (a \cdot_{sstp} δ) = bvars_{sstp} a"
 <proof>

lemma bvars_{sst}_subst: "bvars_{sst} (A \cdot_{sst} δ) = bvars_{sst} A"
 <proof>

lemma bvars_{sstp}_set_cases[simp]:
 "set (bvars_{sstp} (send(t))) = {}"
 "set (bvars_{sstp} (receive(t))) = {}"
 "set (bvars_{sstp} ($\langle c: t \doteq s \rangle$)) = {}"
 "set (bvars_{sstp} (insert(t,s))) = {}"
 "set (bvars_{sstp} (delete(t,s))) = {}"
 "set (bvars_{sstp} ($\langle c: t \in s \rangle$)) = {}"
 "set (bvars_{sstp} ($\forall X(\forall \neq: F \vee \notin: G)$)) = set X"
 <proof>

lemma bvars_{sstp}_NegChecks: " $\neg is_NegChecks$ a \implies bvars_{sstp} a = []"
 <proof>

lemma bvars_{sst}_NegChecks: "bvars_{sst} A = bvars_{sst} (filter is_NegChecks A)"
 <proof>

lemma vars_{sst}_append[simp]: "vars_{sst} (A@B) = vars_{sst} A \cup vars_{sst} B"
 <proof>

lemma vars_{sst}_Nil[simp]: "vars_{sst} [] = {}"
 <proof>

lemma vars_{sst}_Cons: "vars_{sst} (a#A) = vars_{sstp} a \cup vars_{sst} A"
 <proof>

lemma fv_{sst}_Cons: "fv_{sst} (a#A) = fv_{sstp} a \cup fv_{sst} A"
 <proof>

lemma bvars_{sst}_Cons: "bvars_{sst} (a#A) = set (bvars_{sstp} a) \cup bvars_{sst} A"
 <proof>

lemma vars_{sst}_Cons'[simp]:
 "vars_{sst} (send(t)#A) = vars_{sstp} (send(t)) \cup vars_{sst} A"
 "vars_{sst} (receive(t)#A) = vars_{sstp} (receive(t)) \cup vars_{sst} A"
 "vars_{sst} ($\langle a: t \doteq s \rangle$ #A) = vars_{sstp} ($\langle a: t \doteq s \rangle$) \cup vars_{sst} A"
 "vars_{sst} (insert(t,s)#A) = vars_{sstp} (insert(t,s)) \cup vars_{sst} A"
 "vars_{sst} (delete(t,s)#A) = vars_{sstp} (delete(t,s)) \cup vars_{sst} A"
 "vars_{sst} ($\langle a: t \in s \rangle$ #A) = vars_{sstp} ($\langle a: t \in s \rangle$) \cup vars_{sst} A"
 "vars_{sst} ($\forall X(\forall \neq: F \vee \notin: G)$ #A) = vars_{sstp} ($\forall X(\forall \neq: F \vee \notin: G)$) \cup vars_{sst} A"
 <proof>

lemma vars_{sstp}_is_fv_{sstp}_bvars_{sstp}:
 fixes x: "('a, 'b) stateful_strand_step"
 shows "vars_{sstp} x = fv_{sstp} x \cup set (bvars_{sstp} x)"
 <proof>

lemma vars_{sst}_is_fv_{sst}_bvars_{sst}:
 fixes S: "('a, 'b) stateful_strand"
 shows "vars_{sst} S = fv_{sst} S \cup bvars_{sst} S"
 <proof>

lemma `varssstp_NegCheck[simp]`:

"vars_{sstp} ($\forall X(\bigvee \neq: F \bigvee \notin: G$) = set $X \cup \text{fv}_{pairs} F \cup \text{fv}_{pairs} G$ "

<proof>

lemma `bvarssstp_NegCheck[simp]`:

"bvars_{sstp} ($\forall X(\bigvee \neq: F \bigvee \notin: G$) = X "

"set (bvars_{sstp} ($\forall [](\bigvee \neq: F \bigvee \notin: G$))) = $\{ \}$ "

<proof>

lemma `fvsstp_NegCheck[simp]`:

"fv_{sstp} ($\forall X(\bigvee \neq: F \bigvee \notin: G$) = $\text{fv}_{pairs} F \cup \text{fv}_{pairs} G - \text{set } X$ "

"fv_{sstp} ($\forall [](\bigvee \neq: F \bigvee \notin: G$) = $\text{fv}_{pairs} F \cup \text{fv}_{pairs} G$ "

"fv_{sstp} ($\langle t \neq s \rangle$) = $\text{fv } t \cup \text{fv } s$ "

"fv_{sstp} ($\langle t \text{ not in } s \rangle$) = $\text{fv } t \cup \text{fv } s$ "

<proof>

lemma `fvsst_append[simp]`: "fv_{sst} ($A@B$) = fv_{sst} $A \cup \text{fv}_{sst} B$ "

<proof>

lemma `bvarssst_append[simp]`: "bvars_{sst} ($A@B$) = bvars_{sst} $A \cup \text{bvars}_{sst} B$ "

<proof>

lemma `fvsstp_is_subterm_trmssstp`:

assumes "x $\in \text{fv}_{sst} a$ "

shows "Var x $\in \text{subterms}_{set} (\text{trms}_{sst} a)$ "

<proof>

lemma `fvsst_is_subterm_trmssst`: "x $\in \text{fv}_{sst} A \implies \text{Var } x \in \text{subterms}_{set} (\text{trms}_{sst} A)$ "

<proof>

lemma `var_subterm_trmssstp_is_varssstp`:

assumes "Var x $\in \text{subterms}_{set} (\text{trms}_{sst} a)$ "

shows "x $\in \text{vars}_{sst} a$ "

<proof>

lemma `var_subterm_trmssst_is_varssst`: "Var x $\in \text{subterms}_{set} (\text{trms}_{sst} A) \implies x \in \text{vars}_{sst} A$ "

<proof>

lemma `var_trmssst_is_varssst`: "Var x $\in \text{trms}_{sst} A \implies x \in \text{vars}_{sst} A$ "

<proof>

lemma `iksst_trmssst_subset`: "ik_{sst} $A \subseteq \text{trms}_{sst} A$ "

<proof>

lemma `var_subterm_iksst_is_varssst`: "Var x $\in \text{subterms}_{set} (\text{ik}_{sst} A) \implies x \in \text{vars}_{sst} A$ "

<proof>

lemma `var_subterm_iksst_is_fvsst`:

assumes "Var x $\in \text{subterms}_{set} (\text{ik}_{sst} A)$ "

shows "x $\in \text{fv}_{sst} A$ "

<proof>

lemma `fv_iksst_is_fvsst`:

assumes "x $\in \text{fv}_{set} (\text{ik}_{sst} A)$ "

shows "x $\in \text{fv}_{sst} A$ "

<proof>

lemma `fv_trmssst_subset`:

"fv_{set} ($\text{trms}_{sst} S$) $\subseteq \text{vars}_{sst} S$ "

"fv_{sst} $S \subseteq \text{fv}_{set} (\text{trms}_{sst} S)$ "

<proof>

lemma *fv_ik_subset_fv_sst'*[simp]: " $fv_{set} (ik_{sst} S) \subseteq fv_{sst} S$ "
 <proof>

lemma *fv_ik_subset_vars_sst'*[simp]: " $fv_{set} (ik_{sst} S) \subseteq vars_{sst} S$ "
 <proof>

lemma *ik_sst_var_is_fv*: " $Var\ x \in subterms_{set} (ik_{sst} A) \implies x \in fv_{sst} A$ "
 <proof>

lemma *vars_sstp_subst_cases'*:
 assumes $x: "x \in vars_{sstp} (s \cdot_{sstp} \vartheta)"$
 shows " $x \in vars_{sstp} s \vee x \in fv_{set} (\vartheta \setminus vars_{sstp} s)"$ "
 <proof>

lemma *vars_sst_subst_cases*:
 assumes " $x \in vars_{sst} (S \cdot_{sst} \vartheta)"$ "
 shows " $x \in vars_{sst} S \vee x \in fv_{set} (\vartheta \setminus vars_{sst} S)"$ "
 <proof>

lemma *subset_subst_pairs_diff_exists*:
 fixes $I::"('a, 'b) subst"$ and $D\ D'::"('a, 'b) dbstate"$
 shows " $\exists Di. Di \subseteq D \wedge Di \cdot_{pset} I = (D \cdot_{pset} I) - D'"$ "
 <proof>

lemma *subset_subst_pairs_diff_exists'*:
 fixes $I::"('a, 'b) subst"$ and $D::"('a, 'b) dbstate"$
 assumes "finite D"
 shows " $\exists Di. Di \subseteq D \wedge Di \cdot_{pset} I \subseteq \{d \cdot_p I\} \wedge d \cdot_p I \notin (D - Di) \cdot_{pset} I"$ "
 <proof>

lemma *stateful_strand_step_subst_inI*[intro]:
 " $send\langle t \rangle \in set\ A \implies send\langle t \cdot \vartheta \rangle \in set\ (A \cdot_{sst} \vartheta)"$
 " $receive\langle t \rangle \in set\ A \implies receive\langle t \cdot \vartheta \rangle \in set\ (A \cdot_{sst} \vartheta)"$
 " $\langle c: t \doteq s \rangle \in set\ A \implies \langle c: (t \cdot \vartheta) \doteq (s \cdot \vartheta) \rangle \in set\ (A \cdot_{sst} \vartheta)"$
 " $insert\langle t, s \rangle \in set\ A \implies insert\langle t \cdot \vartheta, s \cdot \vartheta \rangle \in set\ (A \cdot_{sst} \vartheta)"$
 " $delete\langle t, s \rangle \in set\ A \implies delete\langle t \cdot \vartheta, s \cdot \vartheta \rangle \in set\ (A \cdot_{sst} \vartheta)"$
 " $\langle c: t \in s \rangle \in set\ A \implies \langle c: (t \cdot \vartheta) \in (s \cdot \vartheta) \rangle \in set\ (A \cdot_{sst} \vartheta)"$
 " $\forall X(\forall \neq: F \vee \notin: G) \in set\ A$
 $\implies \forall X(\forall \neq: (F \cdot_{pairs} rm_vars\ (set\ X)\ \vartheta) \vee \notin: (G \cdot_{pairs} rm_vars\ (set\ X)\ \vartheta)) \in set\ (A \cdot_{sst} \vartheta)"$
 " $\langle t \neq s \rangle \in set\ A \implies \langle t \cdot \vartheta \neq s \cdot \vartheta \rangle \in set\ (A \cdot_{sst} \vartheta)"$
 " $\langle t\ not\ in\ s \rangle \in set\ A \implies \langle t \cdot \vartheta\ not\ in\ s \cdot \vartheta \rangle \in set\ (A \cdot_{sst} \vartheta)"$ "
 <proof>

lemma *stateful_strand_step_cases_subst*:
 " $is_Send\ a = is_Send\ (a \cdot_{sstp} \vartheta)"$
 " $is_Receive\ a = is_Receive\ (a \cdot_{sstp} \vartheta)"$
 " $is_Equality\ a = is_Equality\ (a \cdot_{sstp} \vartheta)"$
 " $is_Insert\ a = is_Insert\ (a \cdot_{sstp} \vartheta)"$
 " $is_Delete\ a = is_Delete\ (a \cdot_{sstp} \vartheta)"$
 " $is_InSet\ a = is_InSet\ (a \cdot_{sstp} \vartheta)"$
 " $is_NegChecks\ a = is_NegChecks\ (a \cdot_{sstp} \vartheta)"$
 " $is_Assignment\ a = is_Assignment\ (a \cdot_{sstp} \vartheta)"$
 " $is_Check\ a = is_Check\ (a \cdot_{sstp} \vartheta)"$
 " $is_Update\ a = is_Update\ (a \cdot_{sstp} \vartheta)"$ "
 <proof>

lemma *stateful_strand_step_subst_inv_cases*:
 " $send\langle t \rangle \in set\ (S \cdot_{sst} \sigma) \implies \exists t'. t = t' \cdot \sigma \wedge send\langle t' \rangle \in set\ S"$
 " $receive\langle t \rangle \in set\ (S \cdot_{sst} \sigma) \implies \exists t'. t = t' \cdot \sigma \wedge receive\langle t' \rangle \in set\ S"$
 " $\langle c: t \doteq s \rangle \in set\ (S \cdot_{sst} \sigma) \implies \exists t' s'. t = t' \cdot \sigma \wedge s = s' \cdot \sigma \wedge \langle c: t' \doteq s' \rangle \in set\ S"$
 " $insert\langle t, s \rangle \in set\ (S \cdot_{sst} \sigma) \implies \exists t' s'. t = t' \cdot \sigma \wedge s = s' \cdot \sigma \wedge insert\langle t', s' \rangle \in set\ S"$
 " $delete\langle t, s \rangle \in set\ (S \cdot_{sst} \sigma) \implies \exists t' s'. t = t' \cdot \sigma \wedge s = s' \cdot \sigma \wedge delete\langle t', s' \rangle \in set\ S"$
 " $\langle c: t \in s \rangle \in set\ (S \cdot_{sst} \sigma) \implies \exists t' s'. t = t' \cdot \sigma \wedge s = s' \cdot \sigma \wedge \langle c: t' \in s' \rangle \in set\ S"$ "

4 The Typing Result for Stateful Protocols

$\forall X(\forall \neq: F \vee \notin: G) \in \text{set } (S \cdot_{sst} \sigma) \implies$
 $\exists F' G'. F = F' \cdot_{pairs} \text{rm_vars } (\text{set } X) \sigma \wedge G = G' \cdot_{pairs} \text{rm_vars } (\text{set } X) \sigma \wedge$
 $\forall X(\forall \neq: F' \vee \notin: G') \in \text{set } S$

<proof>

lemma `stateful_strand_step_fv_subset_cases`:

`"send(t) ∈ set S ⇒ fv t ⊆ fvsst S"`
`"receive(t) ∈ set S ⇒ fv t ⊆ fvsst S"`
`"⟨c: t ≐ s⟩ ∈ set S ⇒ fv t ∪ fv s ⊆ fvsst S"`
`"insert(t,s) ∈ set S ⇒ fv t ∪ fv s ⊆ fvsst S"`
`"delete(t,s) ∈ set S ⇒ fv t ∪ fv s ⊆ fvsst S"`
`"⟨c: t ∈ s⟩ ∈ set S ⇒ fv t ∪ fv s ⊆ fvsst S"`
`"∀ X(∀ ≠: F ∨ ∉: G) ∈ set S ⇒ fvpairs F ∪ fvpairs G - set X ⊆ fvsst S"`

<proof>

lemma `trmssst_nil[simp]`:

`"trmssst [] = {}"`

<proof>

lemma `trmssst_mono`:

`"set M ⊆ set N ⇒ trmssst M ⊆ trmssst N"`

<proof>

lemma `trmssst_in`:

`assumes "t ∈ trmssst S"`
`shows "∃ a ∈ set S. t ∈ trmssstp a"`

<proof>

lemma `trmssst_cons`: `"trmssst (a#A) = trmssstp a ∪ trmssst A"`

<proof>

lemma `trmssst_append[simp]`: `"trmssst (A@B) = trmssst A ∪ trmssst B"`

<proof>

lemma `trmssstp_subst`:

`assumes "set (bvarssstp a) ∩ subst_domain ∅ = {}"`
`shows "trmssstp (a ·sstp ∅) = trmssstp a ·set ∅"`

<proof>

lemma `trmssstp_subst'`:

`assumes "¬is_NegChecks a"`
`shows "trmssstp (a ·sstp ∅) = trmssstp a ·set ∅"`

<proof>

lemma `trmssstp_subst''`:

`fixes t::('a,'b) term" and δ::('a,'b) subst"`
`assumes "t ∈ trmssstp (b ·sstp δ)"`
`shows "∃ s ∈ trmssstp b. t = s · rm_vars (set (bvarssstp b)) δ"`

<proof>

lemma `trmssstp_subst'''`:

`fixes t::('a,'b) term" and δ ∅::('a,'b) subst"`
`assumes "t ∈ trmssstp (b ·sstp δ) ·set ∅"`
`shows "∃ s ∈ trmssstp b. t = s · rm_vars (set (bvarssstp b)) δ ∘s ∅"`

<proof>

lemma `trmssst_subst`:

`assumes "bvarssst S ∩ subst_domain ∅ = {}"`
`shows "trmssst (S ·sst ∅) = trmssst S ·set ∅"`

<proof>

lemma `trmssst_subst_cons`:

`"trmssst (a#A ·sst δ) = trmssstp (a ·sstp δ) ∪ trmssst (A ·sst δ)"`

<proof>

lemma (in intruder_model) wf_{trms}_trms_{sstp}_subst:
 assumes "wf_{trms} (trms_{sstp} a ·_{set} δ)"
 shows "wf_{trms} (trms_{sstp} (a ·_{sstp} δ))"
<proof>

lemma trms_{sst}_fv_vars_{sst}_subset: "t ∈ trms_{sst} A ⇒ fv t ⊆ vars_{sst} A"
<proof>

lemma trms_{sst}_fv_subst_subset:
 assumes "t ∈ trms_{sst} S" "subst_domain ϑ ∩ bvars_{sst} S = {}"
 shows "fv (t · ϑ) ⊆ vars_{sst} (S ·_{sst} ϑ)"
<proof>

lemma trms_{sst}_fv_subst_subset':
 assumes "t ∈ subterms_{set} (trms_{sst} S)" "fv t ∩ bvars_{sst} S = {}" "fv (t · ϑ) ∩ bvars_{sst} S = {}"
 shows "fv (t · ϑ) ⊆ fv_{sst} (S ·_{sst} ϑ)"
<proof>

lemma trms_{sstp}_funs_term_cases:
 assumes "t ∈ trms_{sstp} (s ·_{sstp} ϑ)" "f ∈ funs_term t"
 shows "(∃ u ∈ trms_{sstp} s. f ∈ funs_term u) ∨ (∃ x ∈ fv_{sstp} s. f ∈ funs_term (ϑ x))"
<proof>

lemma trms_{sst}_funs_term_cases:
 assumes "t ∈ trms_{sst} (S ·_{sst} ϑ)" "f ∈ funs_term t"
 shows "(∃ u ∈ trms_{sst} S. f ∈ funs_term u) ∨ (∃ x ∈ fv_{sst} S. f ∈ funs_term (ϑ x))"
<proof>

lemma fv_{sst}_is_subterm_trms_{sst}_subst:
 assumes "x ∈ fv_{sst} T"
 and "bvars_{sst} T ∩ subst_domain ϑ = {}"
 shows "ϑ x ∈ subterms_{set} (trms_{sst} (T ·_{sst} ϑ))"
<proof>

lemma fv_{sst}_subst_fv_subset:
 assumes "x ∈ fv_{sst} S" "x ∉ bvars_{sst} S" "fv (ϑ x) ∩ bvars_{sst} S = {}"
 shows "fv (ϑ x) ⊆ fv_{sst} (S ·_{sst} ϑ)"
<proof>

lemma (in intruder_model) wf_{trms}_trms_{sst}_subst:
 assumes "wf_{trms} (trms_{sst} A ·_{set} δ)"
 shows "wf_{trms} (trms_{sst} (A ·_{sst} δ))"
<proof>

lemma fv_{sst}_subst_obtain_var:
 assumes "x ∈ fv_{sst} (S ·_{sst} δ)"
 shows "∃ y ∈ fv_{sst} S. x ∈ fv (δ y)"
<proof>

lemma fv_{sst}_subst_subset_range_vars_if_subset_domain:
 assumes "fv_{sst} S ⊆ subst_domain σ"
 shows "fv_{sst} (S ·_{sst} σ) ⊆ range_vars σ"
<proof>

lemma fv_{sst}_in_fv_trms_{sst}: "x ∈ fv_{sst} S ⇒ x ∈ fv_{set} (trms_{sst} S)"
<proof>

lemma stateful_strand_step_subst_comp:
 assumes "range_vars δ ∩ set (bvars_{sstp} x) = {}"
 shows "x ·_{sstp} δ ∘_s ϑ = (x ·_{sstp} δ) ·_{sstp} ϑ"
<proof>

lemma `stateful_strand_subst_comp`:

`assumes "range_vars $\delta \cap \text{bvars}_{sst} S = \{\}$ "`

`shows "S $\cdot_{sst} \delta \circ_s \vartheta = (S \cdot_{sst} \delta) \cdot_{sst} \vartheta$ "`

`<proof>`

lemma `subst_apply_bvars_disj_NegChecks`:

`assumes "set X $\cap \text{subst_domain } \vartheta = \{\}$ "`

`shows "NegChecks X F G $\cdot_{sstp} \vartheta = \text{NegChecks X (F } \cdot_{pairs} \vartheta) (G \cdot_{pairs} \vartheta)$ "`

`<proof>`

lemma `subst_apply_NegChecks_no_bvars[simp]`:

`" $\forall [] \langle \nabla \neq: F \vee \notin: F' \rangle \cdot_{sstp} \vartheta = \forall [] \langle \nabla \neq: (F \cdot_{pairs} \vartheta) \vee \notin: (F' \cdot_{pairs} \vartheta) \rangle$ "`

`" $\forall [] \langle \nabla \neq: [] \vee \notin: F' \rangle \cdot_{sstp} \vartheta = \forall [] \langle \nabla \neq: [] \vee \notin: (F' \cdot_{pairs} \vartheta) \rangle$ "`

`" $\forall [] \langle \nabla \neq: F \vee \notin: [] \rangle \cdot_{sstp} \vartheta = \forall [] \langle \nabla \neq: (F \cdot_{pairs} \vartheta) \vee \notin: [] \rangle$ "`

`" $\forall [] \langle \nabla \neq: [] \vee \notin: [(t,s)] \rangle \cdot_{sstp} \vartheta = \forall [] \langle \nabla \neq: [] \vee \notin: [(t \cdot \vartheta, s \cdot \vartheta)] \rangle$ " (is ?A)`

`" $\forall [] \langle \nabla \neq: [(t,s)] \vee \notin: [] \rangle \cdot_{sstp} \vartheta = \forall [] \langle \nabla \neq: [(t \cdot \vartheta, s \cdot \vartheta)] \vee \notin: [] \rangle$ " (is ?B)`

`<proof>`

lemma `setopssst_mono`:

`"set M \subseteq set N \implies setopssst M \subseteq setopssst N"`

`<proof>`

lemma `setopssst_nil[simp]`: `"setopssst [] = $\{\}$ "`

`<proof>`

lemma `setopssst_cons[simp]`: `"setopssst (a#A) = setopssstp a \cup setopssst A"`

`<proof>`

lemma `setopssst_cons_subset[simp]`: `"setopssst A \subseteq setopssst (a#A)"`

`<proof>`

lemma `setopssst_append`: `"setopssst (A@B) = setopssst A \cup setopssst B"`

`<proof>`

lemma `setopssstp_member_iff`:

`"(t,s) \in setopssstp x \longleftrightarrow`

`(x = Insert t s \vee x = Delete t s \vee (\exists ac. x = InSet ac t s) \vee`

`(\exists X F F'. x = NegChecks X F F' \wedge (t,s) \in set F'))"`

`<proof>`

lemma `setopssst_member_iff`:

`"(t,s) \in setopssst A \longleftrightarrow`

`(Insert t s \in set A \vee Delete t s \in set A \vee (\exists ac. InSet ac t s \in set A) \vee`

`(\exists X F F'. NegChecks X F F' \in set A \wedge (t,s) \in set F'))"`

`(is "?P \longleftrightarrow ?Q")`

`<proof>`

lemma `setopssstp_subst`:

`assumes "set (bvarssstp a) \cap subst_domain $\vartheta = \{\}$ "`

`shows "setopssstp (a $\cdot_{sstp} \vartheta) = \text{setops}_{sstp} a \cdot_{pset} \vartheta$ "`

`<proof>`

lemma `setopssstp_subst'`:

`assumes " \neg is_NegChecks a"`

`shows "setopssstp (a $\cdot_{sstp} \vartheta) = \text{setops}_{sstp} a \cdot_{pset} \vartheta$ "`

`<proof>`

lemma `setopssstp_subst''`:

`fixes t::"('a,'b) term \times ('a,'b) term" and δ ::"('a,'b) subst"`

`assumes t: "t \in setopssstp (b $\cdot_{sstp} \delta)$ "`

`shows " \exists s \in setopssstp b. t = s \cdot_p rm_vars (set (bvarssstp b)) δ "`

`<proof>`

```

lemma setopssst_subst:
  assumes "bvarssst S ∩ subst_domain ∅ = {}"
  shows "setopssst (S ·sst ∅) = setopssst S ·pset ∅"
⟨proof⟩

lemma setopssst_subst':
  fixes p:: "('a, 'b) term × ('a, 'b) term" and δ:: "('a, 'b) subst"
  assumes "p ∈ setopssst (S ·sst δ)"
  shows "∃ s ∈ setopssst S. ∃ X. set X ⊆ bvarssst S ∧ p = s ·p rm_vars (set X) δ"
⟨proof⟩

```

4.1.3 Stateful Constraint Semantics

```
context intruder_model
```

```
begin
```

```
definition negchecks_model where
```

```

"negchecks_model (I::('a, 'b) subst) (D::('a, 'b) dbstate) X F G ≡
  (∀ δ. subst_domain δ = set X ∧ ground (subst_range δ) ⟶
    (list_ex (λf. fst f · (δ ◦s I) ≠ snd f · (δ ◦s I)) F ∨
     list_ex (λf. f ·p (δ ◦s I) ∉ D) G))"

```

```
fun strand_sem_stateful::
```

```

"('fun, 'var) terms ⟹ ('fun, 'var) dbstate ⟹ ('fun, 'var) stateful_strand ⟹ ('fun, 'var) subst ⟹
bool"
("[[_; _; _]]s")

```

```
where
```

```

"[[M; D; []]]s = (λI. True)"
| "[[M; D; Send t#S]]s = (λI. M ⊢ t · I ∧ [[M; D; S]]s I)"
| "[[M; D; Receive t#S]]s = (λI. [[insert (t · I) M; D; S]]s I)"
| "[[M; D; Equality _ t t'#S]]s = (λI. t · I = t' · I ∧ [[M; D; S]]s I)"
| "[[M; D; Insert t s#S]]s = (λI. [[M; insert ((t, s) ·p I) D; S]]s I)"
| "[[M; D; Delete t s#S]]s = (λI. [[M; D - {(t, s) ·p I}; S]]s I)"
| "[[M; D; InSet _ t s#S]]s = (λI. (t, s) ·p I ∈ D ∧ [[M; D; S]]s I)"
| "[[M; D; NegChecks X F F'#S]]s = (λI. negchecks_model I D X F F' ∧ [[M; D; S]]s I)"

```

```
lemmas strand_sem_stateful_induct =
```

```

strand_sem_stateful.induct[case_names Nil ConsSnd ConsRcv ConsEq
  ConsIns ConsDel ConsIn ConsNegChecks]

```

```
abbreviation constr_sem_stateful (infix "|=s" 91) where "I |=s A ≡ [[{}; {}; A]]s I"
```

```
lemma stateful_strand_sem_NegChecks_no_bvars:
```

```

"[[M; D; [(t not in s)]]]s I ⟹ (t · I, s · I) ∉ D"
"[[M; D; [(t != s)]]]s I ⟹ t · I ≠ s · I"
⟨proof⟩

```

```
lemma strand_sem_ik_mono_stateful:
```

```

"[[M; D; A]]s I ⟹ [[M ∪ M'; D; A]]s I"
⟨proof⟩

```

```
lemma strand_sem_append_stateful:
```

```

"[[M; D; A@B]]s I ⟷ [[M; D; A]]s I ∧ [[M ∪ (iksst A ·set I); dbupdsst A I D; B]]s I"
(is "?P ⟷ ?Q ∧ ?R")
⟨proof⟩

```

```
lemma negchecks_model_db_subset:
```

```

fixes F F':: "('a, 'b) term × ('a, 'b) term) list"
assumes "D' ⊆ D"
and "negchecks_model I D X F F'"
shows "negchecks_model I D' X F F'"

```

<proof>

lemma *negchecks_model_db_subset*:

fixes $F F'::('a, 'b) \text{ term} \times ('a, 'b) \text{ term} \text{ list}$

assumes " $D' \subseteq D$ "

and " $\forall f \in \text{set } F'. \forall \delta. \text{subst_domain } \delta = \text{set } X \wedge \text{ground } (\text{subst_range } \delta) \longrightarrow f \cdot_p (\delta \circ_s \mathcal{I}) \notin D - D'$ "

and "*negchecks_model* $\mathcal{I} D' X F F'$ "

shows "*negchecks_model* $\mathcal{I} D X F F'$ "

<proof>

lemma *negchecks_model_subst*:

fixes $F F'::('a, 'b) \text{ term} \times ('a, 'b) \text{ term} \text{ list}$

assumes " $(\text{subst_domain } \delta \cup \text{range_vars } \delta) \cap \text{set } X = \{\}$ "

shows "*negchecks_model* $(\delta \circ_s \vartheta) D X F F' \longleftrightarrow \text{negchecks_model } \vartheta D X (F \cdot_{\text{pairs}} \delta) (F' \cdot_{\text{pairs}} \delta)$ "

<proof>

lemma *strand_sem_subst_stateful*:

fixes $\delta::('fun, 'var) \text{ subst}$

assumes " $(\text{subst_domain } \delta \cup \text{range_vars } \delta) \cap \text{bvars}_{sst} S = \{\}$ "

shows " $\llbracket M; D; S \rrbracket_s (\delta \circ_s \vartheta) \longleftrightarrow \llbracket M; D; S \cdot_{sst} \delta \rrbracket_s \vartheta$ "

<proof>

end

4.1.4 Well-Formedness Lemmata

lemma *wfvarsoccs_{sst}_subset_wfrestrictedvars_{sst}* [*simp*]:

" $\text{wfvarsoccs}_{sst} S \subseteq \text{wfrestrictedvars}_{sst} S$ "

<proof>

lemma *wfvarsoccs_{sst}_append*: " $\text{wfvarsoccs}_{sst} (S@S') = \text{wfvarsoccs}_{sst} S \cup \text{wfvarsoccs}_{sst} S'$ "

<proof>

lemma *wfrestrictedvars_{sst}_union* [*simp*]:

" $\text{wfrestrictedvars}_{sst} (S@T) = \text{wfrestrictedvars}_{sst} S \cup \text{wfrestrictedvars}_{sst} T$ "

<proof>

lemma *wfrestrictedvars_{sst}_singleton*:

" $\text{wfrestrictedvars}_{sst} [s] = \text{wfrestrictedvars}_{sstp} s$ "

<proof>

lemma *wf_{sst}_prefix* [*dest*]: " $\text{wf}'_{sst} V (S@S') \Longrightarrow \text{wf}'_{sst} V S$ "

<proof>

lemma *wf_{sst}_vars_mono*: " $\text{wf}'_{sst} V S \Longrightarrow \text{wf}'_{sst} (V \cup W) S$ "

<proof>

lemma *wf_{sst}I* [*intro*]: " $\text{wfrestrictedvars}_{sst} S \subseteq V \Longrightarrow \text{wf}'_{sst} V S$ "

<proof>

lemma *wf_{sst}I'* [*intro*]:

assumes " $\bigcup ((\lambda x. \text{case } x \text{ of}$
 Receive $t \Rightarrow \text{fv } t$
 | Equality Assign $_ t' \Rightarrow \text{fv } t'$
 | Insert $t t' \Rightarrow \text{fv } t \cup \text{fv } t'$
 | $_ \Rightarrow \{\}$) ` set $S) \subseteq V$ "

shows " $\text{wf}'_{sst} V S$ "

<proof>

lemma *wf_{sst}_append_exec*: " $\text{wf}'_{sst} V (S@S') \Longrightarrow \text{wf}'_{sst} (V \cup \text{wfvarsoccs}_{sst} S) S'$ "

<proof>

```

lemma wf_sst_append:
  "wf'_{sst} X S  $\implies$  wf'_{sst} Y T  $\implies$  wf'_{sst} (X  $\cup$  Y) (S@T)"
<proof>

lemma wf_sst_append_suffix:
  "wf'_{sst} V S  $\implies$  wfrestrictedvars_{sst} S'  $\subseteq$  wfrestrictedvars_{sst} S  $\cup$  V  $\implies$  wf'_{sst} V (S@S')"
<proof>

lemma wf_sst_append_suffix':
  assumes "wf'_{sst} V S"
  and " $\bigcup$  (( $\lambda$ x. case x of
    Receive t  $\Rightarrow$  fv t
    | Equality Assign _ t'  $\Rightarrow$  fv t'
    | Insert t t'  $\Rightarrow$  fv t  $\cup$  fv t'
    | _  $\Rightarrow$  {}) ` set S')  $\subseteq$  wfvarsoccs_{sst} S  $\cup$  V"
  shows "wf'_{sst} V (S@S')"
<proof>

lemma wf_sst_subst_apply:
  "wf'_{sst} V S  $\implies$  wf'_{sst} (fv_{set} ( $\delta$  ` V)) (S ._{sst}  $\delta$ )"
<proof>

end

```

4.2 Extending the Typing Result to Stateful Constraints (Stateful_Typing)

```

theory Stateful_Typing
imports Typing_Result Stateful_Strands
begin

  Locale setup

  locale stateful_typed_model = typed_model arity public Ana  $\Gamma$ 
  for arity::"'fun  $\Rightarrow$  nat"
  and public::"'fun  $\Rightarrow$  bool"
  and Ana::"'(fun, 'var) term  $\Rightarrow$  (('fun, 'var) term list  $\times$  ('fun, 'var) term list)"
  and  $\Gamma$ ::"'(fun, 'var) term  $\Rightarrow$  ('fun, 'atom::finite) term_type"
  +
  fixes Pair::"'fun"
  assumes Pair_arity: "arity Pair = 2"
  and Ana_subst': " $\backslash$ f T  $\delta$  K M. Ana (Fun f T) = (K, M)  $\implies$  Ana (Fun f T  $\cdot$   $\delta$ ) = (K .list  $\delta$ , M .list  $\delta$ )"
begin

lemma Ana_invar_subst'[simp]: "Ana_invar_subst S"
<proof>

definition pair where
  "pair d  $\equiv$  case d of (t, t')  $\Rightarrow$  Fun Pair [t, t']"

fun tr_pairs::
  "'(('fun, 'var) term  $\times$  ('fun, 'var) term) list  $\Rightarrow$ 
  ('fun, 'var) dbstatelist  $\Rightarrow$ 
  (('fun, 'var) term  $\times$  ('fun, 'var) term) list list"
where
  "tr_pairs [] D = [[]]"
| "tr_pairs ((s, t)#F) D =
  concat (map ( $\lambda$ d. map ((#) (pair (s, t), pair d)) (tr_pairs F D)) D)"

```

A translation/reduction tr from stateful constraints to (lists of) "non-stateful" constraints. The output represents a finite disjunction of constraints whose models constitute exactly the models of the input constraint. The typing result for "non-stateful" constraints is later lifted to the stateful setting through this reduction procedure.

```

fun tr::("fun, 'var) stateful_strand  $\Rightarrow$  ("fun, 'var) dbstatelist  $\Rightarrow$  ("fun, 'var) strand list"
where
  "tr [] D = [[]]"
  | "tr (send⟨t⟩#A) D = map ((#) (send⟨t⟩st)) (tr A D)"
  | "tr (receive⟨t⟩#A) D = map ((#) (receive⟨t⟩st)) (tr A D)"
  | "tr (⟨ac: t  $\doteq$  t'⟩#A) D = map ((#) (⟨ac: t  $\doteq$  t'⟩st)) (tr A D)"
  | "tr (insert⟨t,s⟩#A) D = tr A (List.insert (t,s) D)"
  | "tr (delete⟨t,s⟩#A) D =
      concat (map (λDi. map (λB. (map (λd. ⟨check: (pair (t,s))  $\doteq$  (pair d)⟩st) Di)@
                    (map (λd.  $\forall$  [] (v≠: [(pair (t,s), pair d)]st) [d←D. d  $\notin$  set Di])@B)
                    (tr A [d←D. d  $\notin$  set Di])))
              (subseqs D))"
  | "tr (⟨ac: t  $\in$  s⟩#A) D =
      concat (map (λB. map (λd. ⟨ac: (pair (t,s))  $\doteq$  (pair d)⟩st#B) D) (tr A D))"
  | "tr (⟨ $\forall$ X (v≠: F v≠: F')⟩#A) D =
      map ((@) (map (λG.  $\forall$ X (v≠: (F@G)⟩st) (trpairs F' D))) (tr A D))"

```

Type-flaw resistance of stateful constraint steps

```

fun tfrsstp where
  "tfrsstp (Equality _ t t') = (( $\exists$ δ. Unifier δ t t')  $\longrightarrow$   $\Gamma$  t =  $\Gamma$  t')"
  | "tfrsstp (NegChecks X F F') = (
      (F' = []  $\wedge$  ( $\forall$ x  $\in$  fvpairs F-set X.  $\exists$ a.  $\Gamma$  (Var x) = TAtom a))  $\vee$ 
      ( $\forall$ f T. Fun f T  $\in$  subtermsset (trmspairs F  $\cup$  pair ` set F')  $\longrightarrow$ 
        T = []  $\vee$  ( $\exists$ s  $\in$  set T. s  $\notin$  Var ` set X)))"
  | "tfrsstp _ = True"

```

Type-flaw resistance of stateful constraints

definition tfr_{sst} **where** "tfr_{sst} S \equiv tfr_{set} (trms_{sst} S \cup pair ` setops_{sst} S) \wedge list_all tfr_{sstp} S"

4.2.1 Small Lemmata

lemma pair_in_pair_image_iff:

```

"pair (s,t)  $\in$  pair ` P  $\iff$  (s,t)  $\in$  P"
⟨proof⟩

```

lemma subst_apply_pairs_pair_image_subst:

```

"pair ` set (F ·pairs  $\vartheta$ ) = pair ` set F ·set  $\vartheta$ "
⟨proof⟩

```

lemma Ana_subst_subterms_cases:

```

fixes  $\vartheta$ ::("fun, 'var) subst"
assumes t: "t  $\in$  subtermsset (M ·set  $\vartheta$ )"
and s: "s  $\in$  set (snd (Ana t))"
shows "( $\exists$ u  $\in$  subtermsset M. t = u ·  $\vartheta$   $\wedge$  s  $\in$  set (snd (Ana u)) ·set  $\vartheta$ )  $\vee$  ( $\exists$ x  $\in$  fvset M. t  $\sqsubseteq$   $\vartheta$  x)"
⟨proof⟩

```

lemma tfr_{sstp}_alt_def:

```

"list_all tfrsstp S =
  (( $\forall$ ac t t'. Equality ac t t'  $\in$  set S  $\wedge$  ( $\exists$ δ. Unifier δ t t')  $\longrightarrow$   $\Gamma$  t =  $\Gamma$  t')  $\wedge$ 
  ( $\forall$ X F F'. NegChecks X F F'  $\in$  set S  $\longrightarrow$  (
    (F' = []  $\wedge$  ( $\forall$ x  $\in$  fvpairs F-set X.  $\exists$ a.  $\Gamma$  (Var x) = TAtom a))  $\vee$ 
    ( $\forall$ f T. Fun f T  $\in$  subtermsset (trmspairs F  $\cup$  pair ` set F')  $\longrightarrow$ 
      T = []  $\vee$  ( $\exists$ s  $\in$  set T. s  $\notin$  Var ` set X))))"
(is "P S = Q S")
⟨proof⟩

```

lemma fun_pair_eq[dest]: "pair d = pair d' \implies d = d'"

⟨proof⟩

lemma fun_pair_subst: "pair d · δ = pair (d ·_p δ)"

⟨proof⟩

lemma fun_pair_subst_set: "pair ` M ·_{set} δ = pair ` (M ·_{pset} δ)"

<proof>

lemma fun_pair_eq_subst: "pair d · δ = pair d' · ϑ \longleftrightarrow d ·_p δ = d' ·_p ϑ"

<proof>

lemma setops_{sst}_pair_image_cons[simp]:

"pair ` setops_{sst} (x#S) = pair ` setops_{sstp} x \cup pair ` setops_{sst} S"
 "pair ` setops_{sst} (send(t)#S) = pair ` setops_{sst} S"
 "pair ` setops_{sst} (receive⟨t⟩#S) = pair ` setops_{sst} S"
 "pair ` setops_{sst} (⟨ac: t \doteq t'⟩#S) = pair ` setops_{sst} S"
 "pair ` setops_{sst} (insert⟨t,s⟩#S) = {pair (t,s)} \cup pair ` setops_{sst} S"
 "pair ` setops_{sst} (delete⟨t,s⟩#S) = {pair (t,s)} \cup pair ` setops_{sst} S"
 "pair ` setops_{sst} (⟨ac: t \in s⟩#S) = {pair (t,s)} \cup pair ` setops_{sst} S"
 "pair ` setops_{sst} ($\forall X(\forall \neq: F \vee \notin: G)$ #S) = pair ` set G \cup pair ` setops_{sst} S"

<proof>

lemma setops_{sst}_pair_image_subst_cons[simp]:

"pair ` setops_{sst} (x#S ·_{sst} ϑ) = pair ` setops_{sstp} (x ·_{sstp} ϑ) \cup pair ` setops_{sst} (S ·_{sst} ϑ)"
 "pair ` setops_{sst} (send(t)#S ·_{sst} ϑ) = pair ` setops_{sst} (S ·_{sst} ϑ)"
 "pair ` setops_{sst} (receive⟨t⟩#S ·_{sst} ϑ) = pair ` setops_{sst} (S ·_{sst} ϑ)"
 "pair ` setops_{sst} (⟨ac: t \doteq t'⟩#S ·_{sst} ϑ) = pair ` setops_{sst} (S ·_{sst} ϑ)"
 "pair ` setops_{sst} (insert⟨t,s⟩#S ·_{sst} ϑ) = {pair (t,s) · ϑ} \cup pair ` setops_{sst} (S ·_{sst} ϑ)"
 "pair ` setops_{sst} (delete⟨t,s⟩#S ·_{sst} ϑ) = {pair (t,s) · ϑ} \cup pair ` setops_{sst} (S ·_{sst} ϑ)"
 "pair ` setops_{sst} (⟨ac: t \in s⟩#S ·_{sst} ϑ) = {pair (t,s) · ϑ} \cup pair ` setops_{sst} (S ·_{sst} ϑ)"
 "pair ` setops_{sst} ($\forall X(\forall \neq: F \vee \notin: G)$ #S ·_{sst} ϑ) =
 pair ` set (G ·_{pairs} rm_vars (set X) ϑ) \cup pair ` setops_{sst} (S ·_{sst} ϑ)"

<proof>

lemma setops_{sst}_are_pairs: "t \in pair ` setops_{sst} A \implies \exists s s'. t = pair (s,s)"

<proof>

lemma fun_pair_wf_{trm}: "wf_{trm} t \implies wf_{trm} t' \implies wf_{trm} (pair (t,t'))"

<proof>

lemma wf_{trms}_pairs: "wf_{trms} (trms_{pairs} F) \implies wf_{trms} (pair ` set F)"

<proof>

lemma tfr_{sst}_Nil[simp]: "tfr_{sst} []"

<proof>

lemma tfr_{sst}_append: "tfr_{sst} (A@B) \implies tfr_{sst} A"

<proof>

lemma tfr_{sst}_append': "tfr_{sst} (A@B) \implies tfr_{sst} B"

<proof>

lemma tfr_{sst}_cons: "tfr_{sst} (a#A) \implies tfr_{sst} A"

<proof>

lemma tfr_{sstp}_subst:

assumes s: "tfr_{sstp} s"
 and ϑ: "wt_{subst} ϑ" "wf_{trms} (subst_range ϑ)" "set (bvars_{sstp} s) \cap range_vars ϑ = {}"
 shows "tfr_{sstp} (s ·_{sstp} ϑ)"

<proof>

lemma tfr_{sstp}_all_wt_subst_apply:

assumes S: "list_all tfr_{sstp} S"
 and ϑ: "wt_{subst} ϑ" "wf_{trms} (subst_range ϑ)" "bvars_{sst} S \cap range_vars ϑ = {}"
 shows "list_all tfr_{sstp} (S ·_{sst} ϑ)"

<proof>

lemma tr_{pairs}_empty_case:

assumes "tr_{pairs} F D = []"

shows "D = []" "F ≠ []"
 ⟨proof⟩

lemma $tr_{pairs_elem_length_eq}$:
 assumes "G ∈ set (tr_{pairs} F D)"
 shows "length G = length F"
 ⟨proof⟩

lemma tr_{pairs_index} :
 assumes "G ∈ set (tr_{pairs} F D)" "i < length F"
 shows "∃ d ∈ set D. G ! i = (pair (F ! i), pair d)"
 ⟨proof⟩

lemma tr_{pairs_cons} :
 assumes "G ∈ set (tr_{pairs} F D)" "d ∈ set D"
 shows "(pair (s,t), pair d)#G ∈ set (tr_{pairs} ((s,t)#F) D)"
 ⟨proof⟩

lemma $tr_{pairs_has_pair_lists}$:
 assumes "G ∈ set (tr_{pairs} F D)" "g ∈ set G"
 shows "∃ f ∈ set F. ∃ d ∈ set D. g = (pair f, pair d)"
 ⟨proof⟩

lemma $tr_{pairs_is_pair_lists}$:
 assumes "f ∈ set F" "d ∈ set D"
 shows "∃ G ∈ set (tr_{pairs} F D). (pair f, pair d) ∈ set G"
 (is "?P F D f d")
 ⟨proof⟩

lemma $tr_{pairs_db_append_subset}$:
 "set (tr_{pairs} F D) ⊆ set (tr_{pairs} F (D@E))" (is ?A)
 "set (tr_{pairs} F E) ⊆ set (tr_{pairs} F (D@E))" (is ?B)
 ⟨proof⟩

lemma $tr_{pairs_trms_subset}$:
 "G ∈ set (tr_{pairs} F D) ⇒ trms_{pairs} G ⊆ pair ` set F ∪ pair ` set D"
 ⟨proof⟩

lemma $tr_{pairs_trms_subset'}$:
 "⋃ (trms_{pairs} ` set (tr_{pairs} F D)) ⊆ pair ` set F ∪ pair ` set D"
 ⟨proof⟩

lemma tr_trms_subset :
 "A' ∈ set (tr A D) ⇒ trms_{st} A' ⊆ trms_{sst} A ∪ pair ` setops_{sst} A ∪ pair ` set D"
 ⟨proof⟩

lemma $tr_{pairs_vars_subset}$:
 "G ∈ set (tr_{pairs} F D) ⇒ fv_{pairs} G ⊆ fv_{pairs} F ∪ fv_{pairs} D"
 ⟨proof⟩

lemma $tr_{pairs_vars_subset'}$: "⋃ (fv_{pairs} ` set (tr_{pairs} F D)) ⊆ fv_{pairs} F ∪ fv_{pairs} D"
 ⟨proof⟩

lemma tr_vars_subset :
 assumes "A' ∈ set (tr A D)"
 shows "fv_{st} A' ⊆ fv_{sst} A ∪ (⋃ (t,t') ∈ set D. fv t ∪ fv t')" (is ?P)
 and "bvars_{st} A' ⊆ bvars_{sst} A" (is ?Q)
 ⟨proof⟩

lemma tr_vars_disj :
 assumes "A' ∈ set (tr A D)" "∀ (t,t') ∈ set D. (fv t ∪ fv t') ∩ bvars_{sst} A = {}"
 and "fv_{sst} A ∩ bvars_{sst} A = {}"
 shows "fv_{st} A' ∩ bvars_{st} A' = {}"

<proof>

lemma *wf_fun_pair_ineqs_map:*

assumes "*wf_{st} X A*"

shows "*wf_{st} X (map (λd. ∀Y⟨V≠: [(pair (t, s), pair d)]_{st}) D@A)*"

<proof>

lemma *wf_fun_pair_negchecks_map:*

assumes "*wf_{st} X A*"

shows "*wf_{st} X (map (λG. ∀Y⟨V≠: (F@G)_{st}) M@A)*"

<proof>

lemma *wf_fun_pair_eqs_ineqs_map:*

fixes *A::('fun, 'var) strand*

assumes "*wf_{st} X A*" "*Di ∈ set (subseqs D)*" "*∀(t,t') ∈ set D. fv t ∪ fv t' ⊆ X*"

shows "*wf_{st} X ((map (λd. ⟨check: (pair (t,s)) ≐ (pair d)_{st}⟩ Di)@
(map (λd. ∀[]⟨V≠: [(pair (t,s), pair d)]_{st}) [d←D. d ∉ set Di])@A)*"

<proof>

lemma *trms_{sst}_wt_subst_ex:*

assumes *ϑ: "wt_{subst} ϑ"* "*wf_{trms} (subst_range ϑ)*"

and *t: "t ∈ trms_{sst} (S ·_{sst} ϑ)"*

shows "*∃s δ. s ∈ trms_{sst} S ∧ wt_{subst} δ ∧ wf_{trms} (subst_range δ) ∧ t = s · δ*"

<proof>

lemma *setops_{sst}_wt_subst_ex:*

assumes *ϑ: "wt_{subst} ϑ"* "*wf_{trms} (subst_range ϑ)*"

and *t: "t ∈ pair ` setops_{sst} (S ·_{sst} ϑ)"*

shows "*∃s δ. s ∈ pair ` setops_{sst} S ∧ wt_{subst} δ ∧ wf_{trms} (subst_range δ) ∧ t = s · δ*"

<proof>

lemma *setops_{sst}_wf_{trms}:*

"*wf_{trms} (trms_{sst} A) ⇒ wf_{trms} (pair ` setops_{sst} A)*"

"*wf_{trms} (trms_{sst} A) ⇒ wf_{trms} (trms_{sst} A ∪ pair ` setops_{sst} A)*"

<proof>

lemma *SMP_MP_split:*

assumes "*t ∈ SMP M*"

and *M: "∀m ∈ M. is_Fun m"*

shows "*(∃δ. wt_{subst} δ ∧ wf_{trms} (subst_range δ) ∧ t ∈ M ·_{set} δ) ∨
t ∈ SMP ((subterms_{set} M ∪ ∪((set ∘ fst ∘ Ana) ` M)) - M)*"

(is "?P t ∨ ?Q t")

<proof>

lemma *setops_subterm_trms:*

assumes *t: "t ∈ pair ` setops_{sst} S"*

and *s: "s ⊆ t"*

shows "*s ∈ subterms_{set} (trms_{sst} S)*"

<proof>

lemma *setops_subterms_cases:*

assumes *t: "t ∈ subterms_{set} (pair ` setops_{sst} S)"*

shows "*t ∈ subterms_{set} (trms_{sst} S) ∨ t ∈ pair ` setops_{sst} S*"

<proof>

lemma *setops_SMP_cases:*

assumes "*t ∈ SMP (pair ` setops_{sst} S)*"

and "*∀p. Ana (pair p) = ([], [])*"

shows "*(∃δ. wt_{subst} δ ∧ wf_{trms} (subst_range δ) ∧ t ∈ pair ` setops_{sst} S ·_{set} δ) ∨ t ∈ SMP (trms_{sst} S)*"

<proof>

lemma *tfr_setops_if_tfr_trms:*

```

assumes "Pair  $\notin \bigcup (\text{funs\_term} \setminus \text{SMP} (\text{trms}_{sst} S))"$ 
and " $\forall p. \text{Ana} (\text{pair } p) = ([], [])"$ 
and " $\forall s \in \text{pair} \setminus \text{setops}_{sst} S. \forall t \in \text{pair} \setminus \text{setops}_{sst} S. (\exists \delta. \text{Unifier } \delta s t) \longrightarrow \Gamma s = \Gamma t"$ 
and " $\forall s \in \text{pair} \setminus \text{setops}_{sst} S. \forall t \in \text{pair} \setminus \text{setops}_{sst} S.
  (\exists \sigma \vartheta \varrho. \text{wt}_{subst} \sigma \wedge \text{wt}_{subst} \vartheta \wedge \text{wf}_{trms} (\text{subst\_range } \sigma) \wedge \text{wf}_{trms} (\text{subst\_range } \vartheta) \wedge
  \text{Unifier } \varrho (s \cdot \sigma) (t \cdot \vartheta))
  \longrightarrow (\exists \delta. \text{Unifier } \delta s t)"$ 
and tfr: "tfrset (trmssst S)"
shows "tfrset (trmssst S  $\cup$  pair  $\setminus$  setopssst S)"
<proof>

```

4.2.2 The Typing Result for Stateful Constraints

context

begin

private lemma tr_wf':

```

assumes " $\forall (t, t') \in \text{set } D. (\text{fv } t \cup \text{fv } t') \cap \text{bvars}_{sst} A = \{\}$ "
and " $\forall (t, t') \in \text{set } D. \text{fv } t \cup \text{fv } t' \subseteq X"$ 
and " $\text{wf}'_{sst} X A$ " " $\text{fv}_{sst} A \cap \text{bvars}_{sst} A = \{\}$ "
and " $A' \in \text{set} (\text{tr } A D)"$ 
shows " $\text{wf}_{st} X A'$ "

```

<proof> lemma tr_wf_{trms}:

```

assumes " $A' \in \text{set} (\text{tr } A [])"$ " " $\text{wf}_{trms} (\text{trms}_{sst} A)"$ 
shows " $\text{wf}_{trms} (\text{trms}_{st} A)"$ 

```

<proof>

lemma tr_wf:

```

assumes " $A' \in \text{set} (\text{tr } A [])"$ 
and " $\text{wf}_{sst} A"$ 
and " $\text{wf}_{trms} (\text{trms}_{sst} A)"$ 
shows " $\text{wf}_{st} \{\} A'$ "
and " $\text{wf}_{trms} (\text{trms}_{st} A)"$ 
and " $\text{fv}_{st} A' \cap \text{bvars}_{st} A' = \{\}$ "

```

<proof> lemma tr_tfr_{stp}:

```

assumes " $A' \in \text{set} (\text{tr } A D)"$ " " $\text{list\_all } \text{tfr}_{stp} A"$ 
and " $\text{fv}_{sst} A \cap \text{bvars}_{sst} A = \{\}$ " (is "?P0 A D")
and " $\forall (t, s) \in \text{set } D. (\text{fv } t \cup \text{fv } s) \cap \text{bvars}_{sst} A = \{\}$ " (is "?P1 A D")
and " $\forall t \in \text{pair} \setminus \text{setops}_{sst} A \cup \text{pair} \setminus \text{set } D. \forall t' \in \text{pair} \setminus \text{setops}_{sst} A \cup \text{pair} \setminus \text{set } D.
  (\exists \delta. \text{Unifier } \delta t t') \longrightarrow \Gamma t = \Gamma t'$ " (is "?P3 A D")
shows " $\text{list\_all } \text{tfr}_{stp} A'$ "

```

<proof>

lemma tr_tfr:

```

assumes " $A' \in \text{set} (\text{tr } A [])"$ " and " $\text{tfr}_{sst} A$ " and " $\text{fv}_{sst} A \cap \text{bvars}_{sst} A = \{\}$ "
shows " $\text{tfr}_{st} A'$ "

```

<proof> lemma fun_pair_ineqs:

```

assumes " $d \cdot_p \delta \cdot_p \vartheta \neq d' \cdot_p \mathcal{I}"$ 
shows " $\text{pair } d \cdot \delta \cdot \vartheta \neq \text{pair } d' \cdot \mathcal{I}"$ 

```

<proof> lemma tr_Delete_constr_iff_aux1:

```

assumes " $\forall d \in \text{set } Di. (t, s) \cdot_p \mathcal{I} = d \cdot_p \mathcal{I}"$ 
and " $\forall d \in \text{set } D - \text{set } Di. (t, s) \cdot_p \mathcal{I} \neq d \cdot_p \mathcal{I}"$ 
shows " $\llbracket M; (\text{map } (\lambda d. \langle \text{check: } (\text{pair } (t, s)) \doteq (\text{pair } d) \rangle_{st}) Di) @
  (\text{map } (\lambda d. \forall [] \langle \neq \rangle: [(\text{pair } (t, s), \text{pair } d)]_{st}) [d \leftarrow D. d \notin \text{set } Di]) \rrbracket_d \mathcal{I}"$ 

```

<proof> lemma tr_Delete_constr_iff_aux2:

```

assumes "ground M"
and " $\llbracket M; (\text{map } (\lambda d. \langle \text{check: } (\text{pair } (t, s)) \doteq (\text{pair } d) \rangle_{st}) Di) @
  (\text{map } (\lambda d. \forall [] \langle \neq \rangle: [(\text{pair } (t, s), \text{pair } d)]_{st}) [d \leftarrow D. d \notin \text{set } Di]) \rrbracket_d \mathcal{I}"$ 
shows " $(\forall d \in \text{set } Di. (t, s) \cdot_p \mathcal{I} = d \cdot_p \mathcal{I}) \wedge (\forall d \in \text{set } D - \text{set } Di. (t, s) \cdot_p \mathcal{I} \neq d \cdot_p \mathcal{I})"$ 

```

<proof> lemma tr_Delete_constr_iff:

```

fixes  $\mathcal{I}::('fun, 'var) \text{subst}"$ 
assumes "ground M"
shows " $\text{set } Di \cdot_{pset} \mathcal{I} \subseteq \{(t, s) \cdot_p \mathcal{I}\} \wedge (t, s) \cdot_p \mathcal{I} \notin (\text{set } D - \text{set } Di) \cdot_{pset} \mathcal{I} \longleftrightarrow
  \llbracket M; (\text{map } (\lambda d. \langle \text{check: } (\text{pair } (t, s)) \doteq (\text{pair } d) \rangle_{st}) Di) @$ 

```

```

      (map (λd. ∀ []⟨∀≠: [(pair (t,s), pair d)]⟩st) [d←D. d ∉ set Di])d I"
⟨proof⟩ lemma tr_NotInSet_constr_iff:
  fixes I::('fun,'var) subst"
  assumes "∀ (t,t') ∈ set D. (fv t ∪ fv t') ∩ set X = {}"
  shows "(∀δ. subst_domain δ = set X ∧ ground (subst_range δ) ⟶ (t,s) ·p δ ·p I ∉ set D ·pset I)
  ⟷ [M; map (λd. ∀ X⟨∀≠: [(pair (t,s), pair d)]⟩st) D]d I"
⟨proof⟩

```

```

lemma tr_NegChecks_constr_iff:
  "(∀G∈set L. ineq_model I X (F@G)) ⟷ [M; map (λG. ∀ X⟨∀≠: (F@G)⟩st) L]d I" (is ?A)
  "negchecks_model I D X F F' ⟷ [M; D; [∀ X⟨∀≠: F ∨∉: F'⟩]]s I" (is ?B)
⟨proof⟩

```

```

lemma tr_pairs_sem_equiv:
  fixes I::('fun,'var) subst"
  assumes "∀ (t,t') ∈ set D. (fv t ∪ fv t') ∩ set X = {}"
  shows "negchecks_model I (set D ·pset I) X F F' ⟷
  (∀G ∈ set (tr_pairs F' D). ineq_model I X (F@G))"
⟨proof⟩

```

```

lemma tr_sem_equiv':
  assumes "∀ (t,t') ∈ set D. (fv t ∪ fv t') ∩ bvarssst A = {}"
  and "fvsst A ∩ bvarssst A = {}"
  and "ground M"
  and I: "interpretationsubst I"
  shows "[M; set D ·pset I; A]s I ⟷ (∃A' ∈ set (tr A D). [M; A']d I)" (is "?P ⟷ ?Q")
⟨proof⟩

```

```

lemma tr_sem_equiv:
  assumes "fvsst A ∩ bvarssst A = {}" and "interpretationsubst I"
  shows "I ⊢s A ⟷ (∃A' ∈ set (tr A []). (I ⊢ ⟨A'⟩))"
⟨proof⟩

```

```

theorem stateful_typing_result:
  assumes "wfsst A"
  and "tfrsst A"
  and "wftrms (trmssst A)"
  and "interpretationsubst I"
  and "I ⊢s A"
  obtains Iτ
  where "interpretationsubst Iτ"
  and "Iτ ⊢s A"
  and "wtsubst Iτ"
  and "wftrms (subst_range Iτ)"
⟨proof⟩

```

end

end

4.2.3 Proving type-flaw resistance automatically

definition pair' where

```
"pair' pair_fun d ≡ case d of (t,t') ⇒ Fun pair_fun [t,t']"
```

fun comp_tfr_{sstp} where

```

  "comp_tfrsstp Γ pair_fun (⟨_: t ≐ t'⟩) = (mgu t t' ≠ None ⟶ Γ t = Γ t')"
| "comp_tfrsstp Γ pair_fun (∀ X⟨∀≠: F ∨∉: F'⟩) = (
  (F' = [] ∧ (∀x ∈ fvpairs F - set X. is_Var (Γ (Var x)))) ∨
  (∀u ∈ subtermsset (trmspairs F ∪ pair' pair_fun ` set F').
    is_Fun u ⟶ (args u = [] ∨ (∃s ∈ set (args u). s ∉ Var ` set X))))"
| "comp_tfrsstp _ _ _ = True"

```

```

definition comp_tfrsst where
  "comp_tfrsst arity Ana  $\Gamma$  pair_fun M S  $\equiv$ 
    list_all (comp_tfrsstp  $\Gamma$  pair_fun) S  $\wedge$ 
    list_all (wftrm ' arity) (trms_listsst S)  $\wedge$ 
    has_all_wt_instances_of  $\Gamma$  (trmssst S  $\cup$  pair' pair_fun ` setopssst S) (set M)  $\wedge$ 
    comp_tfrset arity Ana  $\Gamma$  M"

locale stateful_typed_model' = stateful_typed_model arity public Ana  $\Gamma$  Pair
  for arity::"'fun  $\Rightarrow$  nat"
    and public::"'fun  $\Rightarrow$  bool"
    and Ana::"'(fun, (('fun, 'atom::finite) term_type  $\times$  nat)) term
       $\Rightarrow$  (('fun, (('fun, 'atom) term_type  $\times$  nat)) term list
         $\times$  ('fun, (('fun, 'atom) term_type  $\times$  nat)) term list)"
    and  $\Gamma$ ::"'(fun, (('fun, 'atom) term_type  $\times$  nat)) term  $\Rightarrow$  ('fun, 'atom) term_type"
    and Pair::"'fun"
  +
  assumes  $\Gamma$ _Varfst': " $\bigwedge_{\tau} n m. \Gamma$  (Var ( $\tau, n$ )) =  $\Gamma$  (Var ( $\tau, m$ ))"
    and Anaconst': " $\bigwedge_c T. \text{arity } c = 0 \implies \text{Ana} (\text{Fun } c T) = ([], [])"$ 
begin

sublocale typed_model'
  <proof>

lemma pair_code:
  "pair d = pair' Pair d"
  <proof>

lemma tfrsstp_is_comp_tfrsstp: "tfrsstp a = comp_tfrsstp  $\Gamma$  Pair a"
  <proof>

lemma tfrsst_if_comp_tfrsst:
  assumes "comp_tfrsst arity Ana  $\Gamma$  Pair M S"
  shows "tfrsst S"
  <proof>

lemma tfrsst_if_comp_tfrsst':
  assumes "comp_tfrsst arity Ana  $\Gamma$  Pair (SMPO Ana  $\Gamma$  (trms_listsst S@map pair (setops_listsst S))) S"
  shows "tfrsst S"
  <proof>

end

end

```

5 The Parallel Composition Result for Non-Stateful Protocols

In this chapter, we formalize and prove a compositionality result for security protocols. This work is an extension of the work described in [4] and [1, chapter 5].

5.1 Labeled Strands (Labeled_Strands)

```
theory Labeled_Strands
imports Strands_and_Constraints
begin
```

5.1.1 Definitions: Labeled Strands and Constraints

```
datatype 'l strand_label =
  LabelN (the_LabelN: "'l") ("ln _")
| LabelS ("*")
```

Labeled strands are strands whose steps are equipped with labels

```
type_synonym ('a, 'b, 'c) labeled_strand_step = "'c strand_label × ('a, 'b) strand_step"
type_synonym ('a, 'b, 'c) labeled_strand = "('a, 'b, 'c) labeled_strand_step list"
```

```
abbreviation is_LabelN where "is_LabelN n x ≡ fst x = ln n"
abbreviation is_LabelS where "is_LabelS x ≡ fst x = *"
```

```
definition unlabel where "unlabel S ≡ map snd S"
definition proj where "proj n S ≡ filter (λs. is_LabelN n s ∨ is_LabelS s) S"
abbreviation proj_unl where "proj_unl n S ≡ unlabel (proj n S)"
```

```
abbreviation wfrestrictedvarslst where "wfrestrictedvarslst S ≡ wfrestrictedvarsst (unlabel S)"
```

```
abbreviation subst_apply_labeled_strand_step (infix ".lstp" 51) where
  "x .lstp ϑ ≡ (case x of (l, s) ⇒ (l, s .stp ϑ))"
```

```
abbreviation subst_apply_labeled_strand (infix ".lst" 51) where
  "S .lst ϑ ≡ map (λx. x .lstp ϑ) S"
```

```
abbreviation trmslst where "trmslst S ≡ trmsst (unlabel S)"
abbreviation trms_projlst where "trms_projlst n S ≡ trmsst (proj_unl n S)"
```

```
abbreviation varslst where "varslst S ≡ varsst (unlabel S)"
abbreviation vars_projlst where "vars_projlst n S ≡ varsst (proj_unl n S)"
```

```
abbreviation bvarslst where "bvarslst S ≡ bvarsst (unlabel S)"
abbreviation fvlst where "fvlst S ≡ fvst (unlabel S)"
```

```
abbreviation wflst where "wflst V S ≡ wfst V (unlabel S)"
```

5.1.2 Lemmata: Projections

```
lemma is_LabelS_proj_iff_not_is_LabelN:
  "list_all is_LabelS (proj l A) ↔ ¬list_ex (is_LabelN l) A"
⟨proof⟩
```

```
lemma proj_subset_if_no_label:
```

```

assumes "¬list_ex (is_LabelN l) A"
shows "set (proj l A) ⊆ set (proj l' A)"
and "set (proj_unl l A) ⊆ set (proj_unl l' A)"
⟨proof⟩

lemma proj_in_setD:
assumes a: "a ∈ set (proj l A)"
obtains k b where "a = (k, b)" "k = (ln l) ∨ k = ★"
⟨proof⟩

lemma proj_set_mono:
assumes "set A ⊆ set B"
shows "set (proj n A) ⊆ set (proj n B)"
and "set (proj_unl n A) ⊆ set (proj_unl n B)"
⟨proof⟩

lemma unlabel_nil[simp]: "unlabel [] = []"
⟨proof⟩

lemma unlabel_mono: "set A ⊆ set B ⇒ set (unlabel A) ⊆ set (unlabel B)"
⟨proof⟩

lemma unlabel_in: "(l,x) ∈ set A ⇒ x ∈ set (unlabel A)"
⟨proof⟩

lemma unlabel_mem_has_label: "x ∈ set (unlabel A) ⇒ ∃l. (l,x) ∈ set A"
⟨proof⟩

lemma proj_nil[simp]: "proj n [] = []" "proj_unl n [] = []"
⟨proof⟩

lemma singleton_lst_proj[simp]:
  "proj_unl l [(ln l, a)] = [a]"
  "l ≠ l' ⇒ proj_unl l' [(ln l, a)] = []"
  "proj_unl l [(★, a)] = [a]"
  "unlabel [(l'', a)] = [a]"
⟨proof⟩

lemma unlabel_nil_only_if_nil[simp]: "unlabel A = [] ⇒ A = []"
⟨proof⟩

lemma unlabel_Cons[simp]:
  "unlabel ((l,a)#A) = a#unlabel A"
  "unlabel (b#A) = snd b#unlabel A"
⟨proof⟩

lemma unlabel_append[simp]: "unlabel (A@B) = unlabel A@unlabel B"
⟨proof⟩

lemma proj_Cons[simp]:
  "proj n ((ln n,a)#A) = (ln n,a)#proj n A"
  "proj n ((★,a)#A) = (★,a)#proj n A"
  "m ≠ n ⇒ proj n ((ln m,a)#A) = proj n A"
  "l = (ln n) ⇒ proj n ((l,a)#A) = (l,a)#proj n A"
  "l = ★ ⇒ proj n ((l,a)#A) = (l,a)#proj n A"
  "fst b ≠ ★ ⇒ fst b ≠ (ln n) ⇒ proj n (b#A) = proj n A"
⟨proof⟩

lemma proj_append[simp]:
  "proj l (A'@B') = proj l A'@proj l B'"
  "proj_unl l (A@B) = proj_unl l A@proj_unl l B'"
⟨proof⟩

```



```

lemma proj_unl_cons[simp]:
  "proj_unl l ((ln l, a)#A) = a#proj_unl l A"
  "l ≠ l' ⇒ proj_unl l' ((ln l, a)#A) = proj_unl l' A"
  "proj_unl l ((*, a)#A) = a#proj_unl l A"
  <proof>

lemma trms_unlabel_proj[simp]:
  "trmsstp (snd (ln l, x)) ⊆ trmsprojlst l [(ln l, x)]"
  <proof>

lemma trms_unlabel_star[simp]:
  "trmsstp (snd (*, x)) ⊆ trmsprojlst l [(*, x)]"
  <proof>

lemma trmslst_union[simp]: "trmslst A = (⋃ l. trmsprojlst l A)"
  <proof>

lemma trmslst_append[simp]: "trmslst (A@B) = trmslst A ∪ trmslst B"
  <proof>

lemma trmsprojlst l_append[simp]: "trmsprojlst l (A@B) = trmsprojlst l A ∪ trmsprojlst l B"
  <proof>

lemma trmsprojlst l_subset[simp]:
  "trmsprojlst l A ⊆ trmsprojlst l (A@B)"
  "trmsprojlst l B ⊆ trmsprojlst l (A@B)"
  <proof>

lemma trmslst_subset[simp]:
  "trmslst A ⊆ trmslst (A@B)"
  "trmslst B ⊆ trmslst (A@B)"
  <proof>

lemma varslst_union: "varslst A = (⋃ l. varsprojlst l A)"
  <proof>

lemma unlabel_Cons_inv:
  "unlabel A = b#B ⇒ ∃ A'. (∃ n. A = (ln n, b)#A') ∨ A = (*, b)#A'"
  <proof>

lemma unlabel_snoc_inv:
  "unlabel A = B@[b] ⇒ ∃ A'. (∃ n. A = A'@[ln n, b]) ∨ A = A'@[(*, b)]"
  <proof>

lemma proj_idem[simp]: "proj l (proj l A) = proj l A"
  <proof>

lemma proj_ikst_is_proj_rcv_set:
  "ikst (proj_unl n A) = {t. (ln n, Receive t) ∈ set A ∨ (*, Receive t) ∈ set A}"
  <proof>

lemma unlabel_ikst_is_rcv_set:
  "ikst (unlabel A) = {t | ∃ l. (l, Receive t) ∈ set A}"
  <proof>

lemma proj_ik_union_is_unlabel_ik:
  "ikst (unlabel A) = (⋃ l. ikst (proj_unl l A))"
  <proof>

lemma proj_ik_append[simp]:
  "ikst (proj_unl l (A@B)) = ikst (proj_unl l A) ∪ ikst (proj_unl l B)"
  <proof>

```

```

lemma proj_ik_append_subst_all:
  "ikst (proj_unl l (A@B)) ·set I = (ikst (proj_unl l A) ·set I) ∪ (ikst (proj_unl l B) ·set I)"
⟨proof⟩

lemma ik_proj_subset[simp]: "ikst (proj_unl n A) ⊆ trms_projlst n A"
⟨proof⟩

lemma prefix_proj:
  "prefix A B ⇒ prefix (unlabel A) (unlabel B)"
  "prefix A B ⇒ prefix (proj n A) (proj n B)"
  "prefix A B ⇒ prefix (proj_unl n A) (proj_unl n B)"
⟨proof⟩

```

5.1.3 Lemmata: Well-formedness

```

lemma wfvarsoccsst_proj_union:
  "wfvarsoccsst (unlabel A) = (⋃ l. wfvarsoccsst (proj_unl l A))"
⟨proof⟩

lemma wf_if_wf_proj:
  assumes "∀ l. wfst V (proj_unl l A)"
  shows "wfst V (unlabel A)"
⟨proof⟩

end

```

5.2 Parallel Compositionality of Security Protocols (Parallel_Compositionality)

```

theory Parallel_Compositionality
imports Typing_Result Labeled_Strands
begin

```

5.2.1 Definitions: Labeled Typed Model Locale

```

locale labeled_typed_model = typed_model arity public Ana Γ
  for arity::"fun ⇒ nat"
  and public::"fun ⇒ bool"
  and Ana::"('fun, 'var) term ⇒ (('fun, 'var) term list × ('fun, 'var) term list)"
  and Γ::"('fun, 'var) term ⇒ ('fun, 'atom::finite) term_type"
  +
  fixes label_witness1 and label_witness2::"lbl"
  assumes at_least_2_labels: "label_witness1 ≠ label_witness2"
begin

```

The Ground Sub-Message Patterns (GSMP)

```

definition GSMP::"('fun, 'var) terms ⇒ ('fun, 'var) terms" where
  "GSMP P ≡ {t ∈ SMP P. fv t = {}}"
```

```

definition typing_cond where
  "typing_cond A ≡
  wfst {} A ∧
  fvst A ∩ bvarsst A = {} ∧
  tfrst A ∧
  wftrms (trmsst A) ∧
  Ana_invar_subst (ikst A ∪ assignment_rhsst A)"

```

5.2.2 Definitions: GSMP Disjointedness and Parallel Composability

```

definition GSMP_disjoint where
  "GSMP_disjoint P1 P2 Secrets ≡ GSMP P1 ∩ GSMP P2 ⊆ Secrets ∪ {m. {} ⊢c m}"

```

definition $\text{declassified}_{lst}$ where

" $\text{declassified}_{lst} (\mathcal{A}::('fun, 'var, 'lbl) \text{ labeled_strand}) \mathcal{I} \equiv \{t. (\star, \text{Receive } t) \in \text{set } \mathcal{A}\}_{\text{set } \mathcal{I}}$ "

definition par_comp where

" $\text{par_comp} (\mathcal{A}::('fun, 'var, 'lbl) \text{ labeled_strand}) (\text{Secrets}::('fun, 'var) \text{ terms}) \equiv$
 $(\forall l1 \ l2. l1 \neq l2 \longrightarrow \text{GSMP_disjoint} (\text{trms_proj}_{lst} \ l1 \ \mathcal{A}) (\text{trms_proj}_{lst} \ l2 \ \mathcal{A}) \ \text{Secrets}) \wedge$
 $(\forall s \in \text{Secrets}. \forall s' \in \text{subterms } s. \{\} \vdash_c s' \vee s' \in \text{Secrets}) \wedge$
 $\text{ground } \text{Secrets}$ "

definition $\text{strand_leaks}_{lst}$ where

" $\text{strand_leaks}_{lst} \ \mathcal{A} \ \text{Sec} \ \mathcal{I} \equiv (\exists t \in \text{Sec} - \text{declassified}_{lst} \ \mathcal{A} \ \mathcal{I}. \exists l. (\mathcal{I} \models \langle \text{proj_unl } l \ \mathcal{A} @ [\text{Send } t] \rangle))$ "

5.2.3 Definitions: Homogeneous and Numbered Intruder Deduction Variants

definition proj_specific where

" $\text{proj_specific} \ n \ t \ \mathcal{A} \ \text{Secrets} \equiv t \in \text{GSMP} (\text{trms_proj}_{lst} \ n \ \mathcal{A}) - (\text{Secrets} \cup \{m. \{\} \vdash_c m\})$ "

definition $\text{heterogeneous}_{lst}$ where

" $\text{heterogeneous}_{lst} \ t \ \mathcal{A} \ \text{Secrets} \equiv ($
 $(\exists l1 \ l2. \exists s1 \in \text{subterms } t. \exists s2 \in \text{subterms } t.$
 $l1 \neq l2 \wedge \text{proj_specific} \ l1 \ s1 \ \mathcal{A} \ \text{Secrets} \wedge \text{proj_specific} \ l2 \ s2 \ \mathcal{A} \ \text{Secrets}))$ "

abbreviation homogeneous_{lst} where

" $\text{homogeneous}_{lst} \ t \ \mathcal{A} \ \text{Secrets} \equiv \neg \text{heterogeneous}_{lst} \ t \ \mathcal{A} \ \text{Secrets}$ "

definition $\text{intruder_deduct_hom}::$

" $(('fun, 'var) \text{ terms} \Rightarrow ('fun, 'var, 'lbl) \text{ labeled_strand} \Rightarrow ('fun, 'var) \text{ terms} \Rightarrow ('fun, 'var) \text{ term}$
 $\Rightarrow \text{bool})$ (" $_;$; $_;$) $\vdash_{\text{hom}} _$ " 50)

where

" $\langle M; \mathcal{A}; \text{Sec} \rangle \vdash_{\text{hom}} t \equiv \langle M; \lambda t. \text{homogeneous}_{lst} \ t \ \mathcal{A} \ \text{Sec} \wedge t \in \text{GSMP} (\text{trms}_{lst} \ \mathcal{A}) \rangle \vdash_r t$ "

lemma $\text{intruder_deduct_hom_AxiomH}[\text{simp}]$:

assumes " $t \in M$ "
 shows " $\langle M; \mathcal{A}; \text{Sec} \rangle \vdash_{\text{hom}} t$ "
 $\langle \text{proof} \rangle$

lemma $\text{intruder_deduct_hom_ComposeH}[\text{simp}]$:

assumes " $\text{length } X = \text{arity } f$ " "public f " " $\bigwedge x. x \in \text{set } X \Longrightarrow \langle M; \mathcal{A}; \text{Sec} \rangle \vdash_{\text{hom}} x$ "
 and " $\text{homogeneous}_{lst} (\text{Fun } f \ X) \ \mathcal{A} \ \text{Sec}$ " " $\text{Fun } f \ X \in \text{GSMP} (\text{trms}_{lst} \ \mathcal{A})$ "
 shows " $\langle M; \mathcal{A}; \text{Sec} \rangle \vdash_{\text{hom}} \text{Fun } f \ X$ "
 $\langle \text{proof} \rangle$

lemma $\text{intruder_deduct_hom_DecomposeH}$:

assumes " $\langle M; \mathcal{A}; \text{Sec} \rangle \vdash_{\text{hom}} t$ " "Ana $t = (K, T)$ " " $\bigwedge k. k \in \text{set } K \Longrightarrow \langle M; \mathcal{A}; \text{Sec} \rangle \vdash_{\text{hom}} k$ " " $t_i \in \text{set } T$ "
 shows " $\langle M; \mathcal{A}; \text{Sec} \rangle \vdash_{\text{hom}} t_i$ "
 $\langle \text{proof} \rangle$

lemma $\text{intruder_deduct_hom_induct}[\text{consumes } 1, \text{case_names } \text{AxiomH } \text{ComposeH } \text{DecomposeH}]$:

assumes " $\langle M; \mathcal{A}; \text{Sec} \rangle \vdash_{\text{hom}} t$ " " $\bigwedge t. t \in M \Longrightarrow P \ M \ t$ "
 $\bigwedge X \ f. \llbracket \text{length } X = \text{arity } f; \text{public } f;$
 $\bigwedge x. x \in \text{set } X \Longrightarrow \langle M; \mathcal{A}; \text{Sec} \rangle \vdash_{\text{hom}} x;$
 $\bigwedge x. x \in \text{set } X \Longrightarrow P \ M \ x;$
 $\text{homogeneous}_{lst} (\text{Fun } f \ X) \ \mathcal{A} \ \text{Sec};$
 $\text{Fun } f \ X \in \text{GSMP} (\text{trms}_{lst} \ \mathcal{A})$
 $\rrbracket \Longrightarrow P \ M \ (\text{Fun } f \ X)$
 $\bigwedge t \ K \ T \ t_i. \llbracket \langle M; \mathcal{A}; \text{Sec} \rangle \vdash_{\text{hom}} t; P \ M \ t; \text{Ana } t = (K, T);$
 $\bigwedge k. k \in \text{set } K \Longrightarrow \langle M; \mathcal{A}; \text{Sec} \rangle \vdash_{\text{hom}} k;$
 $\bigwedge k. k \in \text{set } K \Longrightarrow P \ M \ k; t_i \in \text{set } T \rrbracket \Longrightarrow P \ M \ t_i$
 shows " $P \ M \ t$ "
 $\langle \text{proof} \rangle$

lemma ideduct_hom_mono :

" $\llbracket \langle M; \mathcal{A}; \text{Sec} \rangle \vdash_{\text{hom}} t; M \subseteq M' \rrbracket \implies \langle M'; \mathcal{A}; \text{Sec} \rangle \vdash_{\text{hom}} t$ "
 <proof>

5.2.4 Lemmata: GSMP

lemma *GSMP_disjoint_empty[simp]*:

"GSMP_disjoint {} A Sec" "GSMP_disjoint A {} Sec"
 <proof>

lemma *GSMP_mono*:

assumes " $N \subseteq M$ "
 shows " $\text{GSMP } N \subseteq \text{GSMP } M$ "
 <proof>

lemma *GSMP_SMP_mono*:

assumes " $\text{SMP } N \subseteq \text{SMP } M$ "
 shows " $\text{GSMP } N \subseteq \text{GSMP } M$ "
 <proof>

lemma *GSMP_subterm*:

assumes " $t \in \text{GSMP } M$ " " $t' \sqsubseteq t$ "
 shows " $t' \in \text{GSMP } M$ "
 <proof>

lemma *GSMP_subterms*: " $\text{subterms}_{\text{set}} (\text{GSMP } M) = \text{GSMP } M$ "

<proof>

lemma *GSMP_Ana_key*:

assumes " $t \in \text{GSMP } M$ " " $\text{Ana } t = (K, T)$ " " $k \in \text{set } K$ "
 shows " $k \in \text{GSMP } M$ "
 <proof>

lemma *GSMP_append[simp]*: " $\text{GSMP } (\text{trms}_{\text{list}} (A @ B)) = \text{GSMP } (\text{trms}_{\text{list}} A) \cup \text{GSMP } (\text{trms}_{\text{list}} B)$ "

<proof>

lemma *GSMP_union*: " $\text{GSMP } (A \cup B) = \text{GSMP } A \cup \text{GSMP } B$ "

<proof>

lemma *GSMP_Union*: " $\text{GSMP } (\text{trms}_{\text{list}} A) = (\bigcup l. \text{GSMP } (\text{trms_proj}_{\text{list}} l A))$ "

<proof>

lemma *in_GSMP_in_proj*: " $t \in \text{GSMP } (\text{trms}_{\text{list}} A) \implies \exists n. t \in \text{GSMP } (\text{trms_proj}_{\text{list}} n A)$ "

<proof>

lemma *in_proj_in_GSMP*: " $t \in \text{GSMP } (\text{trms_proj}_{\text{list}} n A) \implies t \in \text{GSMP } (\text{trms}_{\text{list}} A)$ "

<proof>

lemma *GSMP_disjointE*:

assumes A : " $\text{GSMP_disjoint } (\text{trms_proj}_{\text{list}} n A) (\text{trms_proj}_{\text{list}} m A) \text{ Sec}$ "
 shows " $\text{GSMP } (\text{trms_proj}_{\text{list}} n A) \cap \text{GSMP } (\text{trms_proj}_{\text{list}} m A) \subseteq \text{Sec} \cup \{m. \{\} \vdash_c m\}$ "
 <proof>

lemma *GSMP_disjoint_term*:

assumes " $\text{GSMP_disjoint } (\text{trms_proj}_{\text{list}} l A) (\text{trms_proj}_{\text{list}} l' A) \text{ Sec}$ "
 shows " $t \notin \text{GSMP } (\text{trms_proj}_{\text{list}} l A) \vee t \notin \text{GSMP } (\text{trms_proj}_{\text{list}} l' A) \vee t \in \text{Sec} \vee \{\} \vdash_c t$ "
 <proof>

lemma *GSMP_wt_subst_subset*:

assumes " $t \in \text{GSMP } (M \cdot_{\text{set}} I)$ " " $\text{wt}_{\text{subst}} I$ " " $\text{wf}_{\text{trms}} (\text{subst_range } I)$ "
 shows " $t \in \text{GSMP } M$ "
 <proof>

lemma *GSMP_wt_substI*:

assumes "t ∈ M" "wt_{subst} I" "wf_{trms} (subst_range I)" "interpretation_{subst} I"
 shows "t · I ∈ GSMP M"
 <proof>

lemma GSMP_disjoint_subset:
 assumes "GSMP_disjoint L R S" "L' ⊆ L" "R' ⊆ R"
 shows "GSMP_disjoint L' R' S"
 <proof>

lemma GSMP_disjoint_fst_specific_not_snd_specific:
 assumes "GSMP_disjoint (trms_proj_{lst} l A) (trms_proj_{lst} l' A) Sec" "l ≠ l'"
 and "proj_specific l m A Sec"
 shows "¬proj_specific l' m A Sec"
 <proof>

lemma GSMP_disjoint_snd_specific_not_fst_specific:
 assumes "GSMP_disjoint (trms_proj_{lst} l A) (trms_proj_{lst} l' A) Sec"
 and "proj_specific l' m A Sec"
 shows "¬proj_specific l m A Sec"
 <proof>

lemma GSMP_disjoint_intersection_not_specific:
 assumes "GSMP_disjoint (trms_proj_{lst} l A) (trms_proj_{lst} l' A) Sec"
 and "t ∈ Sec ∨ {} ⊢_c t"
 shows "¬proj_specific l t A Sec" "¬proj_specific l' t A Sec"
 <proof>

5.2.5 Lemmata: Intruder Knowledge and Declassification

lemma ik_proj_subst_GSMP_subset:
 assumes I: "wt_{subst} I" "wf_{trms} (subst_range I)" "interpretation_{subst} I"
 shows "ik_{st} (proj_unl n A) ·_{set} I ⊆ GSMP (trms_proj_{lst} n A)"
 <proof>

lemma declassified_proj_ik_subset: "declassified_{lst} A I ⊆ ik_{st} (proj_unl n A) ·_{set} I"
 <proof>

lemma declassified_proj_GSMP_subset:
 assumes I: "wt_{subst} I" "wf_{trms} (subst_range I)" "interpretation_{subst} I"
 shows "declassified_{lst} A I ⊆ GSMP (trms_proj_{lst} n A)"
 <proof>

lemma declassified_subterms_proj_GSMP_subset:
 assumes I: "wt_{subst} I" "wf_{trms} (subst_range I)" "interpretation_{subst} I"
 shows "subterms_{set} (declassified_{lst} A I) ⊆ GSMP (trms_proj_{lst} n A)"
 <proof>

lemma declassified_secrets_subset:
 assumes A: "∀n m. n ≠ m → GSMP_disjoint (trms_proj_{lst} n A) (trms_proj_{lst} m A) Sec"
 and I: "wt_{subst} I" "wf_{trms} (subst_range I)" "interpretation_{subst} I"
 shows "declassified_{lst} A I ⊆ Sec ∪ {m. {} ⊢_c m}"
 <proof>

lemma declassified_subterms_secrets_subset:
 assumes A: "∀n m. n ≠ m → GSMP_disjoint (trms_proj_{lst} n A) (trms_proj_{lst} m A) Sec"
 and I: "wt_{subst} I" "wf_{trms} (subst_range I)" "interpretation_{subst} I"
 shows "subterms_{set} (declassified_{lst} A I) ⊆ Sec ∪ {m. {} ⊢_c m}"
 <proof>

lemma declassified_proj_eq: "declassified_{lst} A I = declassified_{lst} (proj n A) I"
 <proof>

lemma declassified_append: "declassified_{lst} (A@B) I = declassified_{lst} A I ∪ declassified_{lst} B I"

<proof>

lemma *declassified_prefix_subset*: "prefix A B \implies declassified_{l_{st}} A I \subseteq declassified_{l_{st}} B I"

<proof>

5.2.6 Lemmata: Homogeneous and Heterogeneous Terms

lemma *proj_specific_secrets_anti_mono*:

assumes "proj_specific l t A Sec" "Sec' \subseteq Sec"

shows "proj_specific l t A Sec'"

<proof>

lemma *heterogeneous_secrets_anti_mono*:

assumes "heterogeneous_{l_{st}} t A Sec" "Sec' \subseteq Sec"

shows "heterogeneous_{l_{st}} t A Sec'"

<proof>

lemma *homogeneous_secrets_mono*:

assumes "homogeneous_{l_{st}} t A Sec'" "Sec' \subseteq Sec"

shows "homogeneous_{l_{st}} t A Sec"

<proof>

lemma *heterogeneous_supterm*:

assumes "heterogeneous_{l_{st}} t A Sec" "t \sqsubseteq t'"

shows "heterogeneous_{l_{st}} t' A Sec"

<proof>

lemma *homogeneous_subterm*:

assumes "homogeneous_{l_{st}} t A Sec" "t' \sqsubseteq t"

shows "homogeneous_{l_{st}} t' A Sec"

<proof>

lemma *proj_specific_subterm*:

assumes "t \sqsubseteq t'" "proj_specific l t' A Sec"

shows "proj_specific l t A Sec \vee t \in Sec \vee {} \vdash_c t"

<proof>

lemma *heterogeneous_term_is_Fun*:

assumes "heterogeneous_{l_{st}} t A S" shows " $\exists f T. t = \text{Fun } f T$ "

<proof>

lemma *proj_specific_is_homogeneous*:

assumes A: " $\forall l l'. l \neq l' \implies \text{GSMP_disjoint } (\text{trms_proj}_{l_{st}} l A) (\text{trms_proj}_{l_{st}} l' A) \text{ Sec}$ "

and t: "proj_specific l m A Sec"

shows "homogeneous_{l_{st}} m A Sec"

<proof>

lemma *deduct_synth_homogeneous*:

assumes "{} \vdash_c t"

shows "homogeneous_{l_{st}} t A Sec"

<proof>

lemma *GSMP_proj_is_homogeneous*:

assumes " $\forall l l'. l \neq l' \implies \text{GSMP_disjoint } (\text{trms_proj}_{l_{st}} l A) (\text{trms_proj}_{l_{st}} l' A) \text{ Sec}$ "

and "t \in GSMP (trms_proj_{l_{st}} l A)" "t \notin Sec"

shows "homogeneous_{l_{st}} t A Sec"

<proof>

lemma *homogeneous_is_not_proj_specific*:

assumes "homogeneous_{l_{st}} m A Sec"

shows " $\exists l::'lbl. \neg \text{proj_specific } l m A \text{ Sec}$ "

<proof>

```

lemma secrets_are_homogeneous:
  assumes " $\forall s \in \text{Sec}. P s \longrightarrow (\forall s' \in \text{subterms } s. \{\} \vdash_c s' \vee s' \in \text{Sec})"$ " "s  $\in$  Sec" "P s"
  shows "homogeneouslst s A Sec"
<proof>

lemma GSMP_is_homogeneous:
  assumes A: " $\forall l l'. l \neq l' \longrightarrow \text{GSMP\_disjoint } (\text{trms\_proj}_{lst} l A) (\text{trms\_proj}_{lst} l' A) \text{ Sec}"$ "
  and t: "t  $\in$  GSMP (trmslst A)" "t  $\notin$  Sec"
  shows "homogeneouslst t A Sec"
<proof>

lemma GSMP_intersection_is_homogeneous:
  assumes A: " $\forall l l'. l \neq l' \longrightarrow \text{GSMP\_disjoint } (\text{trms\_proj}_{lst} l A) (\text{trms\_proj}_{lst} l' A) \text{ Sec}"$ "
  and t: "t  $\in$  GSMP (trmslst l A)  $\cap$  GSMP (trmslst l' A)" "l  $\neq$  l'"
  shows "homogeneouslst t A Sec"
<proof>

lemma GSMP_is_homogeneous':
  assumes A: " $\forall l l'. l \neq l' \longrightarrow \text{GSMP\_disjoint } (\text{trms\_proj}_{lst} l A) (\text{trms\_proj}_{lst} l' A) \text{ Sec}"$ "
  and t: "t  $\in$  GSMP (trmslst A)"
  "t  $\notin$  Sec -  $\bigcup \{\text{GSMP } (\text{trms\_proj}_{lst} l_1 A) \cap \text{GSMP } (\text{trms\_proj}_{lst} l_2 A) \mid l_1 l_2. l_1 \neq l_2\}"$ "
  shows "homogeneouslst t A Sec"
<proof>

lemma declassified_secrets_are_homogeneous:
  assumes A: " $\forall l l'. l \neq l' \longrightarrow \text{GSMP\_disjoint } (\text{trms\_proj}_{lst} l A) (\text{trms\_proj}_{lst} l' A) \text{ Sec}"$ "
  and I: "wtsubst I" "wftrms (subst_range I)" "interpretationsubst I"
  and s: "s  $\in$  declassifiedlst A I"
  shows "homogeneouslst s A Sec"
<proof>

lemma Ana_keys_homogeneous:
  assumes A: " $\forall l l'. l \neq l' \longrightarrow \text{GSMP\_disjoint } (\text{trms\_proj}_{lst} l A) (\text{trms\_proj}_{lst} l' A) \text{ Sec}"$ "
  and t: "t  $\in$  GSMP (trmslst A)"
  and k: "Ana t = (K,T)" "k  $\in$  set K"
  "k  $\notin$  Sec -  $\bigcup \{\text{GSMP } (\text{trms\_proj}_{lst} l_1 A) \cap \text{GSMP } (\text{trms\_proj}_{lst} l_2 A) \mid l_1 l_2. l_1 \neq l_2\}"$ "
  shows "homogeneouslst k A Sec"
<proof>

```

5.2.7 Lemmata: Intruder Deduction Equivalences

```

lemma deduct_if_hom_deduct: " $\langle M; A; S \rangle \vdash_{hom} m \implies M \vdash m"$ "
<proof>

lemma hom_deduct_if_hom_ik:
  assumes " $\langle M; A; \text{Sec} \rangle \vdash_{hom} m"$ " " $\forall m \in M. \text{homogeneous}_{lst} m A \text{ Sec} \wedge m \in \text{GSMP } (\text{trms}_{lst} A)"$ "
  shows "homogeneouslst m A Sec  $\wedge$  m  $\in$  GSMP (trmslst A)"
<proof>

lemma deduct_hom_if_synth:
  assumes hom: "homogeneouslst m A Sec" "m  $\in$  GSMP (trmslst A)"
  and m: "M  $\vdash_c$  m"
  shows " $\langle M; A; \text{Sec} \rangle \vdash_{hom} m"$ "
<proof>

lemma hom_deduct_if_deduct:
  assumes A: "par_comp A Sec"
  and M: " $\forall m \in M. \text{homogeneous}_{lst} m A \text{ Sec} \wedge m \in \text{GSMP } (\text{trms}_{lst} A)"$ "
  and m: "M  $\vdash m"$ " "m  $\in$  GSMP (trmslst A)"
  shows " $\langle M; A; \text{Sec} \rangle \vdash_{hom} m"$ "
<proof>

```

5.2.8 Lemmata: Deduction Reduction of Parallel Composable Constraints

```

lemma par_comp_hom_deduct:
  assumes A: "par_comp A Sec"
  and M: "∀l. ∀m ∈ M l. homogeneouslst m A Sec"
        "∀l. M l ⊆ GSMP (trms_projlst l A)"
        "∀l. Discl ⊆ M l"
        "Discl ⊆ Sec ∪ {m. {} ⊢c m}"
  and Sec: "∀l. ∀s ∈ Sec - Discl. ¬(⟨M l; A; Sec⟩ ⊢hom s)"
  and t: "(∪l. M l; A; Sec) ⊢hom t"
  shows "t ∉ Sec - Discl" (is ?A)
        "∀l. t ∈ GSMP (trms_projlst l A) ⟶ ⟨M l; A; Sec⟩ ⊢hom t" (is ?B)
⟨proof⟩

```

```

lemma par_comp_deduct_proj:
  assumes A: "par_comp A Sec"
  and M: "∀l. ∀m ∈ M l. homogeneouslst m A Sec"
        "∀l. M l ⊆ GSMP (trms_projlst l A)"
        "∀l. Discl ⊆ M l"
  and t: "(∪l. M l) ⊢ t" "t ∈ GSMP (trms_projlst l A)"
  and Discl: "Discl ⊆ Sec ∪ {m. {} ⊢c m}"
  shows "M l ⊢ t ∨ (∃s ∈ Sec - Discl. ∃l. M l ⊢ s)"
⟨proof⟩

```

5.2.9 Theorem: Parallel Compositionality for Labeled Constraints

```

lemma par_comp_prefix: assumes "par_comp (A@B) M" shows "par_comp A M"
⟨proof⟩

```

```

theorem par_comp_constr_typed:
  assumes A: "par_comp A Sec"
  and I: "I ⊢ (unlabel A)" "interpretationsubst I" "wtsubst I" "wftrms (subst_range I)"
  shows "(∀l. (I ⊢ ⟨proj_unl l A⟩)) ∨ (∃A'. prefix A' A ∧ (strand_leakslst A' Sec I))"
⟨proof⟩

```

```

theorem par_comp_constr:
  assumes A: "par_comp A Sec" "typing_cond (unlabel A)"
  and I: "I ⊢ (unlabel A)" "interpretationsubst I"
  shows "∃Iτ. interpretationsubst Iτ ∧ wtsubst Iτ ∧ wftrms (subst_range Iτ) ∧ (Iτ ⊢ (unlabel A)) ∧
        ((∀l. (Iτ ⊢ ⟨proj_unl l A⟩)) ∨ (∃A'. prefix A' A ∧ (strand_leakslst A' Sec Iτ)))"
⟨proof⟩

```

5.2.10 Theorem: Parallel Compositionality for Labeled Protocols

Definitions: Labeled Protocols

We state our result on the level of protocol traces (i.e., the constraints reachable in a symbolic execution of the actual protocol). Hence, we do not need to convert protocol strands to intruder constraints in the following well-formedness definitions.

```

definition wflsts :: "('fun, 'var, 'lbl) labeled_strand set ⇒ bool" where
  "wflsts S ≡ (∀A ∈ S. wflst {} A) ∧ (∀A ∈ S. ∀A' ∈ S. fvlst A ∩ bvarslst A' = {})"

```

```

definition wflsts' :: "('fun, 'var, 'lbl) labeled_strand set ⇒ bool"
where
  "wflsts' S A ≡ (∀A' ∈ S. wfst (wfrestrictedvarslst A) (unlabel A')) ∧
        (∀A' ∈ S. ∀A'' ∈ S. fvlst A' ∩ bvarslst A'' = {}) ∧
        (∀A' ∈ S. fvlst A' ∩ bvarslst A = {}) ∧
        (∀A' ∈ S. fvlst A ∩ bvarslst A' = {})"

```

```

definition typing_cond_prot where
  "typing_cond_prot P ≡
  wflsts P ∧
  tfrset (∪(trmslst ` P)) ∧

```



```

wftrms (⋃ (trmslst ` P)) ∧
(∀ A ∈ P. list_all tfrstp (unlabel A)) ∧
Ana_invar_subst (⋃ (ikst ` unlabel ` P) ∪ ⋃ (assignment_rhsst ` unlabel ` P))"

```

definition par_comp_prot where

```

"par_comp_prot P Sec ≡
(∀ l1 l2. l1 ≠ l2 →
  GSMP_disjoint (⋃ A ∈ P. trms_projlst l1 A) (⋃ A ∈ P. trms_projlst l2 A) Sec) ∧
ground Sec ∧ (∀ s ∈ Sec. ∀ s' ∈ subterms s. {} ⊢c s' ∨ s' ∈ Sec) ∧
typing_cond_prot P"
```

Lemmata: Labeled Protocols

lemma wf_{lsts_eqs}_wf_{lsts'} [simp]: "wf_{lsts} S = wf_{lsts'} S []"
<proof>

lemma par_comp_prot_impl_par_comp:
 assumes "par_comp_prot P Sec" "A ∈ P"
 shows "par_comp A Sec"
 <proof>

lemma typing_cond_prot_impl_typing_cond:
 assumes "typing_cond_prot P" "A ∈ P"
 shows "typing_cond (unlabel A)"
 <proof>

Theorem: Parallel Compositionality for Labeled Protocols

definition component_prot where

```
"component_prot n P ≡ (∀ l ∈ P. ∀ s ∈ set l. is_LabelN n s ∨ is_LabelS s)"
```

definition composed_prot where

```
"composed_prot Pi ≡ {A. ∀ n. proj n A ∈ Pi n}"
```

definition component_secure_prot where

```
"component_secure_prot n P Sec attack ≡ (∀ A ∈ P. suffix [(ln n, Send (Fun attack []))] A →
  (∀ Iτ. (interpretationsubst Iτ ∧ wtsubst Iτ ∧ wftrms (subst_range Iτ)) →
    ¬(Iτ ⊢ ⟨proj_unl n A⟩) ∧
    (∀ A'. prefix A' A →
      (∀ t ∈ Sec-declassifiedlst A' Iτ. ¬(Iτ ⊢ ⟨proj_unl n A'@[Send t]⟩))))))"
```

definition component_leaks where

```
"component_leaks n A Sec ≡ (∃ A' Iτ. interpretationsubst Iτ ∧ wtsubst Iτ ∧ wftrms (subst_range Iτ)
  ∧
  prefix A' A ∧ (∃ t ∈ Sec - declassifiedlst A' Iτ. (Iτ ⊢ ⟨proj_unl n A'@[Send t]⟩)))"
```

definition unsat where

```
"unsat A ≡ (∀ I. interpretationsubst I → ¬(I ⊢ ⟨unlabel A⟩))"
```

theorem par_comp_constr_prot:

```

assumes P: "P = composed_prot Pi" "par_comp_prot P Sec" "∀ n. component_prot n (Pi n)"
and left_secure: "component_secure_prot n (Pi n) Sec attack"
shows "∀ A ∈ P. suffix [(ln n, Send (Fun attack []))] A →
  unsat A ∨ (∃ m. n ≠ m ∧ component_leaks m A Sec)"
```

<proof>

end

5.2.11 Automated GSMP Disjointness

```

locale labeled_typed_model' = typed_model' arity public Ana Γ +
  labeled_typed_model arity public Ana Γ label_witness1 label_witness2
  for arity::"fun ⇒ nat"
```

```

and public::"'fun ⇒ bool"
and Ana::"('fun, (('fun, 'atom::finite) term_type × nat)) term
  ⇒ (('fun, (('fun, 'atom) term_type × nat)) term list
  × ('fun, (('fun, 'atom) term_type × nat)) term list)"
and Γ::"('fun, (('fun, 'atom) term_type × nat)) term ⇒ ('fun, 'atom) term_type"
and label_witness1 label_witness2::'lbl
begin

lemma GSMP_disjointI:
fixes A' A B B'::"('fun, ('fun, 'atom) term × nat) term list"
defines "f ≡ λM. {t · δ | t δ. t ∈ M ∧ wtsubst δ ∧ wftrms (subst_range δ) ∧ fv (t · δ) = {}}"
  and "δ ≡ var_rename (max_var_set (fvset (set A)))"
assumes A'_wf: "list_all (wftrm 'arity) A'"
  and B'_wf: "list_all (wftrm 'arity) B'"
  and A_inst: "has_all_wt_instances_of Γ (set A') (set A)"
  and B_inst: "has_all_wt_instances_of Γ (set B') (set (B ·list δ))"
  and A_SMP_repr: "finite_SMP_representation arity Ana Γ A"
  and B_SMP_repr: "finite_SMP_representation arity Ana Γ (B ·list δ)"
  and AB_trms_disj:
    "∀t ∈ set A. ∀s ∈ set (B ·list δ). Γ t = Γ s ∧ mgu t s ≠ None ⟶
      (intruder_synth' public arity {} t ∧ intruder_synth' public arity {} s) ∨
      ((∃u ∈ Sec. is_wt_instance_of_cond Γ t u) ∧ (∃u ∈ Sec. is_wt_instance_of_cond Γ s u))"
  and Sec_wf: "wftrms Sec"
shows "GSMP_disjoint (set A') (set B') ((f Sec) - {m. {} ⊢c m})"
⟨proof⟩

end

end

```

6 The Stateful Protocol Composition Result

In this chapter, we extend the compositionality result to stateful security protocols. This work is an extension of the work described in [4] and [1, chapter 5].

6.1 Labeled Stateful Strands (Labeled_Stateful_Strands)

```
theory Labeled_Stateful_Strands
imports Stateful_Strands Labeled_Strands
begin
```

6.1.1 Definitions

Syntax for stateful strand labels

```
abbreviation Star_step (" $\star$ , _") where
  " $\langle \star, (s :: ('a, 'b) \text{stateful\_strand\_step}) \rangle \equiv (\star, s)$ "
```

```
abbreviation LabelN_step (" $\_$ , _") where
  " $\langle (l :: 'a), (s :: ('b, 'c) \text{stateful\_strand\_step}) \rangle \equiv (ln\ l, s)$ "
```

Database projection

```
abbreviation dbproj where "dbproj l D  $\equiv$  filter ( $\lambda d. \text{fst } d = l$ ) D"
```

The type of labeled stateful strands

```
type_synonym ('a, 'b, 'c) labeled_stateful_strand_step = "'c strand_label  $\times$  ('a, 'b)
stateful_strand_step"
```

```
type_synonym ('a, 'b, 'c) labeled_stateful_strand = "('a, 'b, 'c) labeled_stateful_strand_step list"
```

Dual strands

```
fun duallsstp :: "('a, 'b, 'c) labeled_stateful_strand_step  $\Rightarrow$  ('a, 'b, 'c) labeled_stateful_strand_step"
where
  "duallsstp (l, send(t)) = (l, receive(t))"
| "duallsstp (l, receive(t)) = (l, send(t))"
| "duallsstp x = x"
```

```
definition duallsst :: "('a, 'b, 'c) labeled_stateful_strand  $\Rightarrow$  ('a, 'b, 'c) labeled_stateful_strand"
where
```

```
"duallsst  $\equiv$  map duallsstp"
```

Substitution application

```
fun subst_apply_labeled_stateful_strand_step ::
  "('a, 'b, 'c) labeled_stateful_strand_step  $\Rightarrow$  ('a, 'b) subst  $\Rightarrow$ 
  ('a, 'b, 'c) labeled_stateful_strand_step"
(infix ".lsstp" 51) where
  "(l, s) .lsstp  $\vartheta$  = (l, s .sstp  $\vartheta$ )"
```

```
definition subst_apply_labeled_stateful_strand ::
  "('a, 'b, 'c) labeled_stateful_strand  $\Rightarrow$  ('a, 'b) subst  $\Rightarrow$  ('a, 'b, 'c) labeled_stateful_strand"
```

```
(infix ".lsst" 51) where
  "S .lsst  $\vartheta \equiv$  map ( $\lambda x. x .lsstp  $\vartheta$ ) S"$ 
```

Definitions lifted from stateful strands

```
abbreviation wfrestrictedvarslsst where "wfrestrictedvarslsst S  $\equiv$  wfrestrictedvarssst (unlabel S)"
```

```
abbreviation iklsst where "iklsst S  $\equiv$  iksst (unlabel S)"
```

abbreviation db_{lsst} where " $db_{lsst} S \equiv db_{sst} (\text{unlabel } S)$ "

abbreviation db'_{lsst} where " $db'_{lsst} S \equiv db'_{sst} (\text{unlabel } S)$ "

abbreviation $trms_{lsst}$ where " $trms_{lsst} S \equiv trms_{sst} (\text{unlabel } S)$ "

abbreviation $trms_proj_{lsst}$ where " $trms_proj_{lsst} n S \equiv trms_{sst} (\text{proj_unl } n S)$ "

abbreviation $vars_{lsst}$ where " $vars_{lsst} S \equiv vars_{sst} (\text{unlabel } S)$ "

abbreviation $vars_proj_{lsst}$ where " $vars_proj_{lsst} n S \equiv vars_{sst} (\text{proj_unl } n S)$ "

abbreviation $bvars_{lsst}$ where " $bvars_{lsst} S \equiv bvars_{sst} (\text{unlabel } S)$ "

abbreviation fv_{lsst} where " $fv_{lsst} S \equiv fv_{sst} (\text{unlabel } S)$ "

Labeled set-operations

fun $setops_{lsstp}$ where

" $setops_{lsstp} (i, \text{insert}(t, s)) = \{(i, t, s)\}$ "
 | " $setops_{lsstp} (i, \text{delete}(t, s)) = \{(i, t, s)\}$ "
 | " $setops_{lsstp} (i, \langle _ : t \in s \rangle) = \{(i, t, s)\}$ "
 | " $setops_{lsstp} (i, \forall _ (\forall \neq: _ \vee \notin: F')) = ((\lambda(t, s). (i, t, s)) \ ` \text{set } F')$ "
 | " $setops_{lsstp} _ = \{ \}$ "

definition $setops_{lsst}$ where

" $setops_{lsst} S \equiv \bigcup (setops_{lsstp} \ ` \text{set } S)$ "

6.1.2 Minor Lemmata

lemma $subst_lsst_nil[simp]$: " $[] \cdot_{lsst} \delta = []$ "

$\langle proof \rangle$

lemma $subst_lsst_cons$: " $a \# A \cdot_{lsst} \delta = (a \cdot_{lsstp} \delta) \# (A \cdot_{lsst} \delta)$ "

$\langle proof \rangle$

lemma $subst_lsst_singleton$: " $[(l, s)] \cdot_{lsst} \delta = [(l, s \cdot_{sstp} \delta)]$ "

$\langle proof \rangle$

lemma $subst_lsst_append$: " $A @ B \cdot_{lsst} \delta = (A \cdot_{lsst} \delta) @ (B \cdot_{lsst} \delta)$ "

$\langle proof \rangle$

lemma $subst_lsst_append_inv$:

assumes " $A \cdot_{lsst} \delta = B1 @ B2$ "

shows " $\exists A1 A2. A = A1 @ A2 \wedge A1 \cdot_{lsst} \delta = B1 \wedge A2 \cdot_{lsst} \delta = B2$ "

$\langle proof \rangle$

lemma $subst_lsst_member[intro]$: " $x \in \text{set } A \implies x \cdot_{lsstp} \delta \in \text{set } (A \cdot_{lsst} \delta)$ "

$\langle proof \rangle$

lemma $subst_lsst_unlabel_cons$: " $\text{unlabel } ((l, b) \# A \cdot_{lsst} \vartheta) = (b \cdot_{sstp} \vartheta) \# (\text{unlabel } (A \cdot_{lsst} \vartheta))$ "

$\langle proof \rangle$

lemma $subst_lsst_unlabel$: " $\text{unlabel } (A \cdot_{lsst} \delta) = \text{unlabel } A \cdot_{sst} \delta$ "

$\langle proof \rangle$

lemma $subst_lsst_unlabel_member[intro]$:

assumes " $x \in \text{set } (\text{unlabel } A)$ "

shows " $x \cdot_{sstp} \delta \in \text{set } (\text{unlabel } (A \cdot_{lsst} \delta))$ "

$\langle proof \rangle$

lemma $subst_lsst_prefix$:

assumes " $\text{prefix } B (A \cdot_{lsst} \vartheta)$ "

shows " $\exists C. C \cdot_{lsst} \vartheta = B \wedge \text{prefix } C A$ "

$\langle proof \rangle$

lemma $dual_{lsst_nil[simp]}$: " $dual_{lsst} [] = []$ "

$\langle proof \rangle$

lemma $dual_{l_{sst}}_Cons[simp]$:

" $dual_{l_{sst}} ((l, send\langle t \rangle) \# A) = (l, receive\langle t \rangle) \# (dual_{l_{sst}} A)$ "
" $dual_{l_{sst}} ((l, receive\langle t \rangle) \# A) = (l, send\langle t \rangle) \# (dual_{l_{sst}} A)$ "
" $dual_{l_{sst}} ((l, \langle a: t \dot{=} s \rangle) \# A) = (l, \langle a: t \dot{=} s \rangle) \# (dual_{l_{sst}} A)$ "
" $dual_{l_{sst}} ((l, insert\langle t, s \rangle) \# A) = (l, insert\langle t, s \rangle) \# (dual_{l_{sst}} A)$ "
" $dual_{l_{sst}} ((l, delete\langle t, s \rangle) \# A) = (l, delete\langle t, s \rangle) \# (dual_{l_{sst}} A)$ "
" $dual_{l_{sst}} ((l, \langle a: t \in s \rangle) \# A) = (l, \langle a: t \in s \rangle) \# (dual_{l_{sst}} A)$ "
" $dual_{l_{sst}} ((l, \forall X \langle \forall \neq: F \vee \notin: G \rangle) \# A) = (l, \forall X \langle \forall \neq: F \vee \notin: G \rangle) \# (dual_{l_{sst}} A)$ "

$\langle proof \rangle$

lemma $dual_{l_{sst}}_append[simp]$: " $dual_{l_{sst}} (A @ B) = dual_{l_{sst}} A @ dual_{l_{sst}} B$ "

$\langle proof \rangle$

lemma $dual_{l_{sstp}}_subst$: " $dual_{l_{sstp}} (s \cdot_{l_{sstp}} \delta) = (dual_{l_{sstp}} s) \cdot_{l_{sstp}} \delta$ "

$\langle proof \rangle$

lemma $dual_{l_{sst}}_subst$: " $dual_{l_{sst}} (S \cdot_{l_{sst}} \delta) = (dual_{l_{sst}} S) \cdot_{l_{sst}} \delta$ "

$\langle proof \rangle$

lemma $dual_{l_{sst}}_subst_unlabel$: " $unlabel (dual_{l_{sst}} (S \cdot_{l_{sst}} \delta)) = unlabel (dual_{l_{sst}} S) \cdot_{l_{sst}} \delta$ "

$\langle proof \rangle$

lemma $dual_{l_{sst}}_subst_cons$: " $dual_{l_{sst}} (a \# A \cdot_{l_{sst}} \sigma) = (dual_{l_{sstp}} a \cdot_{l_{sstp}} \sigma) \# (dual_{l_{sst}} (A \cdot_{l_{sst}} \sigma))$ "

$\langle proof \rangle$

lemma $dual_{l_{sst}}_subst_append$: " $dual_{l_{sst}} (A @ B \cdot_{l_{sst}} \sigma) = (dual_{l_{sst}} A @ dual_{l_{sst}} B) \cdot_{l_{sst}} \sigma$ "

$\langle proof \rangle$

lemma $dual_{l_{sst}}_subst_snoc$: " $dual_{l_{sst}} (A @ [a] \cdot_{l_{sst}} \sigma) = (dual_{l_{sst}} A \cdot_{l_{sst}} \sigma) @ [dual_{l_{sstp}} a \cdot_{l_{sstp}} \sigma]$ "

$\langle proof \rangle$

lemma $dual_{l_{sst}}_memberD$:

assumes " $(l, a) \in set (dual_{l_{sst}} A)$ "
shows " $\exists b. (l, b) \in set A \wedge dual_{l_{sstp}} (l, b) = (l, a)$ "
 $\langle proof \rangle$

lemma $dual_{l_{sstp}}_inv$:

assumes " $dual_{l_{sstp}} (l, a) = (k, b)$ "
shows " $l = k$ "
and " $a = receive\langle t \rangle \implies b = send\langle t \rangle$ "
and " $a = send\langle t \rangle \implies b = receive\langle t \rangle$ "
and " $(\nexists t. a = receive\langle t \rangle \vee a = send\langle t \rangle) \implies b = a$ "

$\langle proof \rangle$

lemma $dual_{l_{sst}}_self_inverse$: " $dual_{l_{sst}} (dual_{l_{sst}} A) = A$ "

$\langle proof \rangle$

lemma $vars_{l_{sst}}_unlabel_dual_{l_{sst}}_eq$: " $vars_{l_{sst}} (dual_{l_{sst}} A) = vars_{l_{sst}} A$ "

$\langle proof \rangle$

lemma $fv_{l_{sst}}_unlabel_dual_{l_{sst}}_eq$: " $fv_{l_{sst}} (dual_{l_{sst}} A) = fv_{l_{sst}} A$ "

$\langle proof \rangle$

lemma $bvars_{l_{sst}}_unlabel_dual_{l_{sst}}_eq$: " $bvars_{l_{sst}} (dual_{l_{sst}} A) = bvars_{l_{sst}} A$ "

$\langle proof \rangle$

lemma $vars_{l_{sst}}_unlabel_Cons$: " $vars_{l_{sst}} ((l, b) \# A) = vars_{l_{sstp}} b \cup vars_{l_{sst}} A$ "

$\langle proof \rangle$

lemma $fv_{l_{sst}}_unlabel_Cons$: " $fv_{l_{sst}} ((l, b) \# A) = fv_{l_{sstp}} b \cup fv_{l_{sst}} A$ "

$\langle proof \rangle$

lemma $bvars_{sst_unlabel_Cons}$: " $bvars_{l_{sst}} ((l,b)\#A) = set (bvars_{sstp} b) \cup bvars_{l_{sst}} A$ "
 <proof>

lemma $bvars_{l_{sst_subst}}$: " $bvars_{l_{sst}} (A \cdot_{l_{sst}} \delta) = bvars_{l_{sst}} A$ "
 <proof>

lemma $dual_{l_{sst_member}}$:
 assumes " $(l,x) \in set A$ "
 and " $\neg is_Receive\ x$ " " $\neg is_Send\ x$ "
 shows " $(l,x) \in set (dual_{l_{sst}} A)$ "
 <proof>

lemma $dual_{l_{sst_unlabel_member}}$:
 assumes " $x \in set (unlabel\ A)$ "
 and " $\neg is_Receive\ x$ " " $\neg is_Send\ x$ "
 shows " $x \in set (unlabel (dual_{l_{sst}} A))$ "
 <proof>

lemma $dual_{l_{sst_steps_iff}}$:
 " $(l,send\langle t \rangle) \in set A \longleftrightarrow (l,receive\langle t \rangle) \in set (dual_{l_{sst}} A)$ "
 " $(l,receive\langle t \rangle) \in set A \longleftrightarrow (l,send\langle t \rangle) \in set (dual_{l_{sst}} A)$ "
 " $(l,\langle c: t \dot{=} s \rangle) \in set A \longleftrightarrow (l,\langle c: t \dot{=} s \rangle) \in set (dual_{l_{sst}} A)$ "
 " $(l,insert\langle t,s \rangle) \in set A \longleftrightarrow (l,insert\langle t,s \rangle) \in set (dual_{l_{sst}} A)$ "
 " $(l,delete\langle t,s \rangle) \in set A \longleftrightarrow (l,delete\langle t,s \rangle) \in set (dual_{l_{sst}} A)$ "
 " $(l,\langle c: t \in s \rangle) \in set A \longleftrightarrow (l,\langle c: t \in s \rangle) \in set (dual_{l_{sst}} A)$ "
 " $(l,\forall X(\forall \neq: F \vee \notin: G)) \in set A \longleftrightarrow (l,\forall X(\forall \neq: F \vee \notin: G)) \in set (dual_{l_{sst}} A)$ "
 <proof>

lemma $dual_{l_{sst_unlabel_steps_iff}}$:
 " $send\langle t \rangle \in set (unlabel\ A) \longleftrightarrow receive\langle t \rangle \in set (unlabel (dual_{l_{sst}} A))$ "
 " $receive\langle t \rangle \in set (unlabel\ A) \longleftrightarrow send\langle t \rangle \in set (unlabel (dual_{l_{sst}} A))$ "
 " $\langle c: t \dot{=} s \rangle \in set (unlabel\ A) \longleftrightarrow \langle c: t \dot{=} s \rangle \in set (unlabel (dual_{l_{sst}} A))$ "
 " $insert\langle t,s \rangle \in set (unlabel\ A) \longleftrightarrow insert\langle t,s \rangle \in set (unlabel (dual_{l_{sst}} A))$ "
 " $delete\langle t,s \rangle \in set (unlabel\ A) \longleftrightarrow delete\langle t,s \rangle \in set (unlabel (dual_{l_{sst}} A))$ "
 " $\langle c: t \in s \rangle \in set (unlabel\ A) \longleftrightarrow \langle c: t \in s \rangle \in set (unlabel (dual_{l_{sst}} A))$ "
 " $\forall X(\forall \neq: F \vee \notin: G) \in set (unlabel\ A) \longleftrightarrow \forall X(\forall \neq: F \vee \notin: G) \in set (unlabel (dual_{l_{sst}} A))$ "
 <proof>

lemma $dual_{l_{sst_list_all}}$:
 " $list_all\ is_Receive\ (unlabel\ A) \implies list_all\ is_Send\ (unlabel (dual_{l_{sst}} A))$ "
 " $list_all\ is_Send\ (unlabel\ A) \implies list_all\ is_Receive\ (unlabel (dual_{l_{sst}} A))$ "
 " $list_all\ is_Equality\ (unlabel\ A) \implies list_all\ is_Equality\ (unlabel (dual_{l_{sst}} A))$ "
 " $list_all\ is_Insert\ (unlabel\ A) \implies list_all\ is_Insert\ (unlabel (dual_{l_{sst}} A))$ "
 " $list_all\ is_Delete\ (unlabel\ A) \implies list_all\ is_Delete\ (unlabel (dual_{l_{sst}} A))$ "
 " $list_all\ is_InSet\ (unlabel\ A) \implies list_all\ is_InSet\ (unlabel (dual_{l_{sst}} A))$ "
 " $list_all\ is_NegChecks\ (unlabel\ A) \implies list_all\ is_NegChecks\ (unlabel (dual_{l_{sst}} A))$ "
 " $list_all\ is_Assignment\ (unlabel\ A) \implies list_all\ is_Assignment\ (unlabel (dual_{l_{sst}} A))$ "
 " $list_all\ is_Check\ (unlabel\ A) \implies list_all\ is_Check\ (unlabel (dual_{l_{sst}} A))$ "
 " $list_all\ is_Update\ (unlabel\ A) \implies list_all\ is_Update\ (unlabel (dual_{l_{sst}} A))$ "
 <proof>

lemma $dual_{l_{sst_in_set_prefix_obtain}}$:
 assumes " $s \in set (unlabel (dual_{l_{sst}} A))$ "
 shows " $\exists l\ B\ s'. (l,s) = dual_{l_{sstp}} (l,s') \wedge prefix (B@[l,s']) A$ "
 <proof>

lemma $dual_{l_{sst_in_set_prefix_obtain_subst}}$:
 assumes " $s \in set (unlabel (dual_{l_{sst}} (A \cdot_{l_{sst}} \vartheta)))$ "
 shows " $\exists l\ B\ s'. (l,s) = dual_{l_{sstp}} ((l,s') \cdot_{l_{sstp}} \vartheta) \wedge prefix ((B \cdot_{l_{sst}} \vartheta)@[l,s'] \cdot_{l_{sstp}} \vartheta) (A \cdot_{l_{sst}} \vartheta)$ "
 <proof>

lemma $trms_{sst_unlabel_dual_{l_{sst_eq}}$: " $trms_{l_{sst}} (dual_{l_{sst}} A) = trms_{l_{sst}} A$ "

<proof>

lemma `trmssst_unlabel_subst_cons`:

"`trmslsst ((l,b)#A ·lsst δ) = trmssstp (b ·sstp δ) ∪ trmslsst (A ·lsst δ)`"

<proof>

lemma `trmssst_unlabel_subst`:

assumes "`bvarslsst S ∩ subst_domain ϑ = {}`"

shows "`trmslsst (S ·lsst ϑ) = trmslsst S ·set ϑ`"

<proof>

lemma `trmssst_unlabel_subst'`:

fixes `t::('a,'b) term` and `δ::('a,'b) subst`

assumes "`t ∈ trmslsst (S ·lsst δ)`"

shows "`∃s ∈ trmslsst S. ∃X. set X ⊆ bvarslsst S ∧ t = s · rm_vars (set X) δ`"

<proof>

lemma `trmssst_unlabel_subst''`:

fixes `t::('a,'b) term` and `δ ϑ::('a,'b) subst`

assumes "`t ∈ trmslsst (S ·lsst δ) ·set ϑ`"

shows "`∃s ∈ trmslsst S. ∃X. set X ⊆ bvarslsst S ∧ t = s · rm_vars (set X) δ ∘s ϑ`"

<proof>

lemma `trmssst_unlabel_dual_subst_cons`:

"`trmslsst (duallsst (a#A ·lsst σ)) = (trmssstp (snd a ·sstp σ)) ∪ (trmslsst (duallsst (A ·lsst σ)))`"

<proof>

lemma `duallsst_funs_term`:

"`∪ (funs_term ` (trmssst (unlabel (duallsst S)))) = ∪ (funs_term ` (trmssst (unlabel S)))`"

<proof>

lemma `duallsst_dblsst`:

"`db'lsst (duallsst A) = db'lsst A`"

<proof>

lemma `dbsst_unlabel_append`:

"`db'lsst (A@B) I D = db'lsst B I (db'lsst A I D)`"

<proof>

lemma `dbsst_duallsst`:

"`db'sst (unlabel (duallsst (T ·lsst δ))) I D = db'sst (unlabel (T ·lsst δ)) I D`"

<proof>

lemma `labeled_list_insert_eq_cases`:

"`d ∉ set (unlabel D) ⇒ List.insert d (unlabel D) = unlabel (List.insert (i,d) D)`"

"`(i,d) ∈ set D ⇒ List.insert d (unlabel D) = unlabel (List.insert (i,d) D)`"

<proof>

lemma `labeled_list_insert_eq_ex_cases`:

"`List.insert d (unlabel D) = unlabel (List.insert (i,d) D) ∨`

`(∃j. (j,d) ∈ set D ∧ List.insert d (unlabel D) = unlabel (List.insert (j,d) D))`"

<proof>

lemma `proj_subst`: "`proj l (A ·lsst δ) = proj l A ·lsst δ`"

<proof>

lemma `proj_set_subset[simp]`:

"`set (proj n A) ⊆ set A`"

<proof>

lemma `proj_proj_set_subset[simp]`:

"`set (proj n (proj m A)) ⊆ set (proj n A)`"

"`set (proj n (proj m A)) ⊆ set (proj m A)`"

```

"set (proj_unl n (proj m A)) ⊆ set (proj_unl n A)"
"set (proj_unl n (proj m A)) ⊆ set (proj_unl m A)"
⟨proof⟩

```

```

lemma proj_in_set_iff:
  "(ln i, d) ∈ set (proj i D) ↔ (ln i, d) ∈ set D"
  "(*, d) ∈ set (proj i D) ↔ (*, d) ∈ set D"
⟨proof⟩

```

```

lemma proj_list_insert:
  "proj i (List.insert (ln i,d) D) = List.insert (ln i,d) (proj i D)"
  "proj i (List.insert (*,d) D) = List.insert (*,d) (proj i D)"
  "i ≠ j ⇒ proj i (List.insert (ln j,d) D) = proj i D"
⟨proof⟩

```

```

lemma proj_filter: "proj i [d←D. d ∉ set Di] = [d←proj i D. d ∉ set Di]"
⟨proof⟩

```

```

lemma proj_list_Cons:
  "proj i ((ln i,d)#D) = (ln i,d)#proj i D"
  "proj i ((*,d)#D) = (*,d)#proj i D"
  "i ≠ j ⇒ proj i ((ln j,d)#D) = proj i D"
⟨proof⟩

```

```

lemma proj_duallsst:
  "proj l (duallsst A) = duallsst (proj l A)"
⟨proof⟩

```

```

lemma proj_instance_ex:
  assumes B: "∀ b ∈ set B. ∃ a ∈ set A. ∃ δ. b = a ·lsst δ ∧ P δ"
  and b: "b ∈ set (proj l B)"
  shows "∃ a ∈ set (proj l A). ∃ δ. b = a ·lsst δ ∧ P δ"
⟨proof⟩

```

```

lemma proj_dbproj:
  "dbproj (ln i) (proj i D) = dbproj (ln i) D"
  "dbproj * (proj i D) = dbproj * D"
  "i ≠ j ⇒ dbproj (ln j) (proj i D) = []"
⟨proof⟩

```

```

lemma dbproj_Cons:
  "dbproj i ((i,d)#D) = (i,d)#dbproj i D"
  "i ≠ j ⇒ dbproj j ((i,d)#D) = dbproj j D"
⟨proof⟩

```

```

lemma dbproj_subset[simp]:
  "set (unlabel (dbproj i D)) ⊆ set (unlabel D)"
⟨proof⟩

```

```

lemma dbproj_subseq:
  assumes "Di ∈ set (subseqs (dbproj k D))"
  shows "dbproj k Di = Di" (is ?A)
  and "i ≠ k ⇒ dbproj i Di = []" (is "i ≠ k ⇒ ?B")
⟨proof⟩

```

```

lemma dbproj_subseq_subset:
  assumes "Di ∈ set (subseqs (dbproj i D))"
  shows "set Di ⊆ set D"
⟨proof⟩

```

```

lemma dbproj_subseq_in_subseqs:
  assumes "Di ∈ set (subseqs (dbproj i D))"
  shows "Di ∈ set (subseqs D)"

```


<proof>

lemma *proj_subseq*:

assumes " $Di \in \text{set} (\text{subseqs} (\text{dbproj} (\text{In } j) D))$ " " $j \neq i$ "
 shows " $[d \leftarrow \text{proj } i D. d \notin \text{set } Di] = \text{proj } i D$ "

<proof>

lemma *unlabel_subseqsD*:

assumes " $A \in \text{set} (\text{subseqs} (\text{unlabel } B))$ "
 shows " $\exists C \in \text{set} (\text{subseqs } B). \text{unlabel } C = A$ "

<proof>

lemma *unlabel_filter_eq*:

assumes " $\forall (j, p) \in \text{set } A \cup B. \forall (k, q) \in \text{set } A \cup B. p = q \longrightarrow j = k$ " (is "?P (set A)")
 shows " $[d \leftarrow \text{unlabel } A. d \notin \text{snd } \text{` } B] = \text{unlabel } [d \leftarrow A. d \notin B]$ "

<proof>

lemma *subseqs_mem_dbproj*:

assumes " $Di \in \text{set} (\text{subseqs } D)$ " " $\text{list_all } (\lambda d. \text{fst } d = i) Di$ "
 shows " $Di \in \text{set} (\text{subseqs} (\text{dbproj } i D))$ "

<proof>

lemma *unlabel_subst*: " $\text{unlabel } S \cdot_{\text{sst}} \delta = \text{unlabel } (S \cdot_{\text{lsst}} \delta)$ "

<proof>

lemma *subterms_subst_lsst*:

assumes " $\forall x \in \text{fv}_{\text{set}} (\text{trms}_{\text{lsst}} S). (\exists f. \sigma x = \text{Fun } f []) \vee (\exists y. \sigma x = \text{Var } y)$ "
 and " $\text{bvars}_{\text{lsst}} S \cap \text{subst_domain } \sigma = \{\}$ "
 shows " $\text{subterms}_{\text{set}} (\text{trms}_{\text{lsst}} (S \cdot_{\text{lsst}} \sigma)) = \text{subterms}_{\text{set}} (\text{trms}_{\text{lsst}} S) \cdot_{\text{set}} \sigma$ "

<proof>

lemma *subterms_subst_lsst_ik*:

assumes " $\forall x \in \text{fv}_{\text{set}} (\text{ik}_{\text{lsst}} S). (\exists f. \sigma x = \text{Fun } f []) \vee (\exists y. \sigma x = \text{Var } y)$ "
 shows " $\text{subterms}_{\text{set}} (\text{ik}_{\text{lsst}} (S \cdot_{\text{lsst}} \sigma)) = \text{subterms}_{\text{set}} (\text{ik}_{\text{lsst}} S) \cdot_{\text{set}} \sigma$ "

<proof>

lemma *labeled_stateful_strand_subst_comp*:

assumes " $\text{range_vars } \delta \cap \text{bvars}_{\text{lsst}} S = \{\}$ "
 shows " $S \cdot_{\text{lsst}} \delta \circ_s \vartheta = (S \cdot_{\text{lsst}} \delta) \cdot_{\text{lsst}} \vartheta$ "

<proof>

lemma *sst_vars_proj_subset[simp]*:

" $\text{fv}_{\text{sst}} (\text{proj_unl } n A) \subseteq \text{fv}_{\text{sst}} (\text{unlabel } A)$ "
 " $\text{bvars}_{\text{sst}} (\text{proj_unl } n A) \subseteq \text{bvars}_{\text{sst}} (\text{unlabel } A)$ "
 " $\text{vars}_{\text{sst}} (\text{proj_unl } n A) \subseteq \text{vars}_{\text{sst}} (\text{unlabel } A)$ "

<proof>

lemma *trms_sst_proj_subset[simp]*:

" $\text{trms}_{\text{sst}} (\text{proj_unl } n A) \subseteq \text{trms}_{\text{sst}} (\text{unlabel } A)$ " (is ?A)
 " $\text{trms}_{\text{sst}} (\text{proj_unl } m (\text{proj } n A)) \subseteq \text{trms}_{\text{sst}} (\text{proj_unl } n A)$ " (is ?B)
 " $\text{trms}_{\text{sst}} (\text{proj_unl } m (\text{proj } n A)) \subseteq \text{trms}_{\text{sst}} (\text{proj_unl } m A)$ " (is ?C)

<proof>

lemma *trms_sst_unlabel_prefix_subset*:

" $\text{trms}_{\text{sst}} (\text{unlabel } A) \subseteq \text{trms}_{\text{sst}} (\text{unlabel } (A@B))$ " (is ?A)
 " $\text{trms}_{\text{sst}} (\text{proj_unl } n A) \subseteq \text{trms}_{\text{sst}} (\text{proj_unl } n (A@B))$ " (is ?B)

<proof>

lemma *trms_sst_unlabel_suffix_subset*:

" $\text{trms}_{\text{sst}} (\text{unlabel } B) \subseteq \text{trms}_{\text{sst}} (\text{unlabel } (A@B))$ "
 " $\text{trms}_{\text{sst}} (\text{proj_unl } n B) \subseteq \text{trms}_{\text{sst}} (\text{proj_unl } n (A@B))$ "

<proof>

```

lemma setopslsstpD:
  assumes p: "p ∈ setopslsstp a"
  shows "fst p = fst a" (is ?P)
    and "is_Update (snd a) ∨ is_InSet (snd a) ∨ is_NegChecks (snd a)" (is ?Q)
⟨proof⟩

lemma setopslsst_nil[simp]:
  "setopslsst [] = {}"
⟨proof⟩

lemma setopslsst_cons[simp]:
  "setopslsst (x#S) = setopslsstp x ∪ setopslsst S"
⟨proof⟩

lemma setopssst_proj_subset:
  "setopssst (proj_unl n A) ⊆ setopssst (unlabel A)"
  "setopssst (proj_unl m (proj n A)) ⊆ setopssst (proj_unl n A)"
  "setopssst (proj_unl m (proj n A)) ⊆ setopssst (proj_unl m A)"
⟨proof⟩

lemma setopssst_unlabel_prefix_subset:
  "setopssst (unlabel A) ⊆ setopssst (unlabel (A@B))"
  "setopssst (proj_unl n A) ⊆ setopssst (proj_unl n (A@B))"
⟨proof⟩

lemma setopssst_unlabel_suffix_subset:
  "setopssst (unlabel B) ⊆ setopssst (unlabel (A@B))"
  "setopssst (proj_unl n B) ⊆ setopssst (proj_unl n (A@B))"
⟨proof⟩

lemma setopslsst_proj_subset:
  "setopslsst (proj n A) ⊆ setopslsst A"
  "setopslsst (proj m (proj n A)) ⊆ setopslsst (proj n A)"
⟨proof⟩

lemma setopslsst_prefix_subset:
  "setopslsst A ⊆ setopslsst (A@B)"
  "setopslsst (proj n A) ⊆ setopslsst (proj n (A@B))"
⟨proof⟩

lemma setopslsst_suffix_subset:
  "setopslsst B ⊆ setopslsst (A@B)"
  "setopslsst (proj n B) ⊆ setopslsst (proj n (A@B))"
⟨proof⟩

lemma setopslsst_mono:
  "set M ⊆ set N ⇒ setopslsst M ⊆ setopslsst N"
⟨proof⟩

lemma trmssst_unlabel_subset_if_no_label:
  "¬list_ex (is_LabelN l) A ⇒ trmslsst (proj l A) ⊆ trmslsst (proj l' A)"
⟨proof⟩

lemma setopssst_unlabel_subset_if_no_label:
  "¬list_ex (is_LabelN l) A ⇒ setopssst (proj_unl l A) ⊆ setopssst (proj_unl l' A)"
⟨proof⟩

lemma setopslsst_proj_subset_if_no_label:
  "¬list_ex (is_LabelN l) A ⇒ setopslsst (proj l A) ⊆ setopslsst (proj l' A)"
⟨proof⟩

lemma setopslsstp_subst_cases[simp]:
  "setopslsstp ((l, send(t)) ·lsstp δ) = {}"

```

```

"setopslsstp ((l, receive⟨t⟩) ·lsstp δ) = {}"
"setopslsstp ((l, ⟨ac: s ≐ t⟩) ·lsstp δ) = {}"
"setopslsstp ((l, insert⟨t, s⟩) ·lsstp δ) = {(l, t · δ, s · δ)}"
"setopslsstp ((l, delete⟨t, s⟩) ·lsstp δ) = {(l, t · δ, s · δ)}"
"setopslsstp ((l, ⟨ac: t ∈ s⟩) ·lsstp δ) = {(l, t · δ, s · δ)}"
"setopslsstp ((l, ∀X(∀≠: F ∨ ∉: F')) ·lsstp δ) =
  ((λ(t, s). (l, t · rm_vars (set X) δ, s · rm_vars (set X) δ)) ` set F'" (is "?A = ?B")
⟨proof⟩

lemma setopslsstp_subst:
  assumes "set (bvarssstp (snd a)) ∩ subst_domain ϑ = {}"
  shows "setopslsstp (a ·lsstp ϑ) = (λp. (fst a, snd p ·p ϑ)) ` setopslsstp a"
⟨proof⟩

lemma setopslsstp_subst':
  assumes "set (bvarssstp (snd a)) ∩ subst_domain ϑ = {}"
  shows "setopslsstp (a ·lsstp ϑ) = (λ(i, p). (i, p ·p ϑ)) ` setopslsstp a"
⟨proof⟩

lemma setopslsst_subst:
  assumes "bvarslsst S ∩ subst_domain ϑ = {}"
  shows "setopslsst (S ·lsst ϑ) = (λp. (fst p, snd p ·p ϑ)) ` setopslsst S"
⟨proof⟩

lemma setopslsstp_in_subst:
  assumes p: "p ∈ setopslsstp (a ·lsstp δ)"
  shows "∃ q ∈ setopslsstp a. fst p = fst q ∧ snd p = snd q ·p rm_vars (set (bvarssstp (snd a))) δ"
  (is "∃ q ∈ setopslsstp a. ?P q")
⟨proof⟩

lemma setopslsst_in_subst:
  assumes "p ∈ setopslsst (A ·lsst δ)"
  shows "∃ q ∈ setopslsst A. fst p = fst q ∧ (∃ X ⊆ bvarslsst A. snd p = snd q ·p rm_vars X δ)"
  (is "∃ q ∈ setopslsst A. ?P A q")
⟨proof⟩

lemma setopslsst_duallsst_eq:
  "setopslsst (duallsst A) = setopslsst A"
⟨proof⟩

end

```

6.2 Stateful Protocol Compositionality (Stateful_Compositionality)

```

theory Stateful_Compositionality
imports Stateful_Typing Parallel_Compositionality Labeled_Stateful_Strands
begin

```

6.2.1 Small Lemmata

```

lemma (in typed_model) wt_subst_sstp_vars_type_subset:
  fixes a: "('fun, 'var) stateful_strand_step"
  assumes "wtsubst δ"
  and "∀ t ∈ subst_range δ. fv t = {} ∨ (∃ x. t = Var x)"
  shows "Γ ` Var ` fvsstp (a ·sstp δ) ⊆ Γ ` Var ` fvsstp a" (is ?A)
  and "Γ ` Var ` set (bvarssstp (a ·sstp δ)) = Γ ` Var ` set (bvarssstp a)" (is ?B)
  and "Γ ` Var ` varssstp (a ·sstp δ) ⊆ Γ ` Var ` varssstp a" (is ?C)
⟨proof⟩

lemma (in typed_model) wt_subst_lsst_vars_type_subset:
  fixes A: "('fun, 'var, 'a) labeled_stateful_strand"
  assumes "wtsubst δ"

```

```

    and "∀t ∈ subst_range δ. fv t = {} ∨ (∃x. t = Var x)"
  shows "Γ ` Var ` fvlsst (A ·lsst δ) ⊆ Γ ` Var ` fvlsst A" (is ?A)
    and "Γ ` Var ` bvarslsst (A ·lsst δ) = Γ ` Var ` bvarslsst A" (is ?B)
    and "Γ ` Var ` varslsst (A ·lsst δ) ⊆ Γ ` Var ` varslsst A" (is ?C)
<proof>

```

```

lemma (in stateful_typed_model) fv_pair_fv_pairs_subset:
  assumes "d ∈ set D"
  shows "fv (pair (snd d)) ⊆ fvpairs (unlabel D)"
<proof>

```

```

lemma (in stateful_typed_model) labeled_sat_ineq_lift:
  assumes "[M; map (λd. ∀X(∀≠: [(pair (t,s), pair (snd d))]))st] [d←dbproj i D. d ∉ set Di]]d I"
    (is "?R1 D")
  and "∀(j,p) ∈ {(i,t,s)} ∪ set D ∪ set Di. ∀(k,q) ∈ {(i,t,s)} ∪ set D ∪ set Di.
    (∃δ. Unifier δ (pair p) (pair q)) → j = k" (is "?R2 D")
  shows "[M; map (λd. ∀X(∀≠: [(pair (t,s), pair (snd d))]))st] [d←D. d ∉ set Di]]d I"
<proof>

```

```

lemma (in stateful_typed_model) labeled_sat_ineq_dbproj:
  assumes "[M; map (λd. ∀X(∀≠: [(pair (t,s), pair (snd d))]))st] [d←D. d ∉ set Di]]d I"
    (is "?P D")
  shows "[M; map (λd. ∀X(∀≠: [(pair (t,s), pair (snd d))]))st] [d←dbproj i D. d ∉ set Di]]d I"
    (is "?Q D")
<proof>

```

```

lemma (in stateful_typed_model) labeled_sat_ineq_dbproj_sem_equiv:
  assumes "∀(j,p) ∈ ((λ(t, s). (i, t, s)) ` set F') ∪ set D.
    ∀(k,q) ∈ ((λ(t, s). (i, t, s)) ` set F') ∪ set D.
    (∃δ. Unifier δ (pair p) (pair q)) → j = k"
  and "fvpairs (map snd D) ∩ set X = {}"
  shows "[M; map (λG. ∀X(∀≠: (F@G))st) (trpairs F' (map snd D))]d I ↔
    [M; map (λG. ∀X(∀≠: (F@G))st) (trpairs F' (map snd (dbproj i D)))]d I"
<proof>

```

```

lemma (in stateful_typed_model) labeled_sat_eqs_list_all:
  assumes "∀(j, p) ∈ {(i,t,s)} ∪ set D. ∀(k,q) ∈ {(i,t,s)} ∪ set D.
    (∃δ. Unifier δ (pair p) (pair q)) → j = k" (is "?P D")
  and "[M; map (λd. ⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st) D]]d I" (is "?Q D")
  shows "list_all (λd. fst d = i) D"
<proof>

```

```

lemma (in stateful_typed_model) labeled_sat_eqs_subseqs:
  assumes "Di ∈ set (subseqs D)"
  and "∀(j, p) ∈ {(i,t,s)} ∪ set D. ∀(k, q) ∈ {(i,t,s)} ∪ set D.
    (∃δ. Unifier δ (pair p) (pair q)) → j = k" (is "?P D")
  and "[M; map (λd. ⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st) Di]]d I"
  shows "Di ∈ set (subseqs (dbproj i D))"
<proof>

```

```

lemma (in stateful_typed_model) duallsst_tfrsstp:
  assumes "list_all tfrsstp (unlabel S)"
  shows "list_all tfrsstp (unlabel (duallsst S))"
<proof>

```

```

lemma (in stateful_typed_model) setopssst_unlabel_duallsst_eq:
  "setopssst (unlabel (duallsst A)) = setopssst (unlabel A)"
<proof>

```

6.2.2 Locale Setup and Definitions

```

locale labeled_stateful_typed_model =
  stateful_typed_model arity public Ana Γ Pair

```

```

+ labeled_typed_model arity public Ana  $\Gamma$  label_witness1 label_witness2
  for arity::"fun  $\Rightarrow$  nat"
  and public::"fun  $\Rightarrow$  bool"
  and Ana::"('fun, 'var) term  $\Rightarrow$  (('fun, 'var) term list  $\times$  ('fun, 'var) term list)"
  and  $\Gamma$ ::"('fun, 'var) term  $\Rightarrow$  ('fun, 'atom::finite) term_type"
  and Pair::"fun"
  and label_witness1::"lbl"
  and label_witness2::"lbl"
begin

definition lpair where
  "lpair lp  $\equiv$  case lp of (i,p)  $\Rightarrow$  (i,pair p)"

lemma setopslsstp_pair_image[simp]:
  "lpair ` (setopslsstp (i,send(t))) = {}"
  "lpair ` (setopslsstp (i,receive(t))) = {}"
  "lpair ` (setopslsstp (i,(ac: t  $\doteq$  t')) = {}"
  "lpair ` (setopslsstp (i,insert(t,s))) = {(i, pair (t,s))}"
  "lpair ` (setopslsstp (i,delete(t,s))) = {(i, pair (t,s))}"
  "lpair ` (setopslsstp (i,(ac: t  $\in$  s))) = {(i, pair (t,s))}"
  "lpair ` (setopslsstp (i, $\forall X(\forall \neq: F \vee \notin: F')$ )) = (( $\lambda(t,s). (i, pair (t,s))$ ) ` set F)"
<proof>

definition par_complsst where
  "par_complsst (A::('fun, 'var, 'lbl) labeled_stateful_strand) (Secrets::('fun, 'var) terms)  $\equiv$ 
    ( $\forall$  l1 l2. l1  $\neq$  l2  $\longrightarrow$ 
      GSMP_disjoint (trmssst (proj_unl l1 A)  $\cup$  pair ` setopssst (proj_unl l1 A))
        (trmssst (proj_unl l2 A)  $\cup$  pair ` setopssst (proj_unl l2 A)) Secrets)  $\wedge$ 
      ground Secrets  $\wedge$  ( $\forall s \in$  Secrets.  $\forall s' \in$  subterms s. {}  $\vdash_c$  s'  $\vee$  s'  $\in$  Secrets)  $\wedge$ 
      ( $\forall (i,p) \in$  setopslsst A.  $\forall (j,q) \in$  setopslsst A.
        ( $\exists \delta$ . Unifier  $\delta$  (pair p) (pair q)  $\longrightarrow$  i = j)"

definition declassifiedlsst where
  "declassifiedlsst A  $\mathcal{I} \equiv$  {t. (*, receive(t))  $\in$  set A} .set  $\mathcal{I}$ "

definition strand_leakslsst ("_ leaks _ under _") where
  "(A::('fun, 'var, 'lbl) labeled_stateful_strand) leaks Secrets under  $\mathcal{I} \equiv$ 
    ( $\exists t \in$  Secrets - declassifiedlsst A  $\mathcal{I}$ .  $\exists n$ .  $\mathcal{I} \models_s$  (proj_unl n A@[send(t)]))"

definition typing_condsst where
  "typing_condsst A  $\equiv$  wfsst A  $\wedge$  wftrms (trmssst A)  $\wedge$  tfrsst A"

type_synonym ('a, 'b, 'c) labeleddbstate = "('c strand_label  $\times$  (('a, 'b) term  $\times$  ('a, 'b) term)) set"
type_synonym ('a, 'b, 'c) labeleddbstatelist = "('c strand_label  $\times$  (('a, 'b) term  $\times$  ('a, 'b) term)) list"

For proving the compositionality theorem for stateful constraints the idea is to first define a variant of the reduction technique that was used to establish the stateful typing result. This variant performs database-state projections, and it allows us to reduce the compositionality problem for stateful constraints to ordinary constraints.

fun trpc::
  "('fun, 'var, 'lbl) labeled_stateful_strand  $\Rightarrow$  ('fun, 'var, 'lbl) labeleddbstatelist
   $\Rightarrow$  ('fun, 'var, 'lbl) labeled_strand list"
where
  "trpc [] D = [[]]"
  | "trpc ((i,send(t))#A) D = map ((#) (i,send(t))st) (trpc A D)"
  | "trpc ((i,receive(t))#A) D = map ((#) (i,receive(t))st) (trpc A D)"
  | "trpc ((i,(ac: t  $\doteq$  t'))#A) D = map ((#) (i,(ac: t  $\doteq$  t'))st) (trpc A D)"
  | "trpc ((i,insert(t,s))#A) D = trpc A (List.insert (i,(t,s)) D)"
  | "trpc ((i,delete(t,s))#A) D = (
      concat (map ( $\lambda$ Di. map ( $\lambda$ B. (map ( $\lambda$ d. (i,(check: (pair (t,s))  $\doteq$  (pair (snd d)))st)) Di)@
        (map ( $\lambda$ d. (i, $\forall$  []( $\forall \neq: [(pair (t,s), pair (snd d))]$ )st))
          [d $\leftarrow$ dbproj i D. d  $\notin$  set Di])@B)

```

```

      (trpc A [d←D. d ∉ set Di]))
      (subseqs (dbproj i D)))"
| "trpc ((i,⟨ac: t ∈ s⟩)#A) D =
  concat (map (λB. map (λd. (i,⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩)st)#B) (dbproj i D)) (trpc A D))"
| "trpc ((i,∀X⟨∇≠: F ∨∇≠: F'⟩)#A) D =
  map ((@) (map (λG. (i,∀X⟨∇≠: (F@G)⟩)st)) (trpairs F' (map snd (dbproj i D)))) (trpc A D)"

```

6.2.3 Small Lemmata

lemma par_comp_{l_{sst}}_nil:

```

  assumes "ground Sec" "∀s ∈ Sec. ∀s' ∈ subterms s. {} ⊢c s' ∨ s' ∈ Sec"
  shows "par_complsst [] Sec"

```

⟨proof⟩

lemma par_comp_{l_{sst}}_subset:

```

  assumes A: "par_complsst A Sec"
  and BA: "set B ⊆ set A"
  shows "par_complsst B Sec"

```

⟨proof⟩

lemma par_comp_{l_{sst}}_split:

```

  assumes "par_complsst (A@B) Sec"
  shows "par_complsst A Sec" "par_complsst B Sec"

```

⟨proof⟩

lemma par_comp_{l_{sst}}_proj:

```

  assumes "par_complsst A Sec"
  shows "par_complsst (proj n A) Sec"

```

⟨proof⟩

lemma par_comp_{l_{sst}}_dual_{l_{sst}}:

```

  assumes A: "par_complsst A S"
  shows "par_complsst (duallsst A) S"

```

⟨proof⟩

lemma par_comp_{l_{sst}}_subst:

```

  assumes A: "par_complsst A S"
  and δ: "wtsubst δ" "wftrms (subst_range δ)" "subst_domain δ ∩ bvarslsst A = {}"
  shows "par_complsst (A ·lsst δ) S"

```

⟨proof⟩

lemma wf_pair_negchecks_map':

```

  assumes "wfst X (unlabel A)"
  shows "wfst X (unlabel (map (λG. (i,∀Y⟨∇≠: (F@G)⟩)st)) M@A))"

```

⟨proof⟩

lemma wf_pair_eqs_ineqs_map':

```

  fixes A: "('fun, 'var, 'lbl) labeled_strand"
  assumes "wfst X (unlabel A)"
  "Di ∈ set (subseqs (dbproj i D))"
  "fvpairs (unlabel D) ⊆ X"

```

```

  shows "wfst X (unlabel (
    (map (λd. (i,⟨check: (pair (t,s)) ≐ (pair (snd d))⟩)st)) Di)@
    (map (λd. (i,∀[]⟨∇≠: [(pair (t,s), pair (snd d))]⟩)st)) [d←dbproj i D. d ∉ set Di]@A))"

```

⟨proof⟩

lemma trms_{sst}_setops_{sst}_wt_instance_ex:

```

  defines "M ≡ λA. trmslsst A ∪ pair ` setopssst (unlabel A)"
  assumes B: "∀b ∈ set B. ∃a ∈ set A. ∃δ. b = a ·lsst δ ∧ wtsubst δ ∧ wftrms (subst_range δ)"
  shows "∀t ∈ M B. ∃s ∈ M A. ∃δ. t = s · δ ∧ wtsubst δ ∧ wftrms (subst_range δ)"

```

⟨proof⟩

lemma setops_{l_{sst}}_wt_instance_ex:

assumes $B: \forall b \in \text{set } B. \exists a \in \text{set } A. \exists \delta. b = a \cdot_{lsst} \delta \wedge \text{wt}_{subst} \delta \wedge \text{wf}_{trms} (\text{subst_range } \delta)$
 shows $\forall p \in \text{setops}_{lsst} B. \exists q \in \text{setops}_{lsst} A. \exists \delta. \text{fst } p = \text{fst } q \wedge \text{snd } p = \text{snd } q \cdot_p \delta \wedge \text{wt}_{subst} \delta \wedge \text{wf}_{trms} (\text{subst_range } \delta)$
 <proof>

6.2.4 Lemmata: Properties of the Constraint Translation Function

lemma `tr_par_labeled_rcv_iff`:
 $B \in \text{set } (\text{tr}_{pc} A D) \implies (i, \text{receive}(t)_{st}) \in \text{set } B \longleftrightarrow (i, \text{receive}(t)) \in \text{set } A$
 <proof>

lemma `tr_par_declassified_eq`:
 $B \in \text{set } (\text{tr}_{pc} A D) \implies \text{declassified}_{lst} B I = \text{declassified}_{lsst} A I$
 <proof>

lemma `tr_par_ik_eq`:
 assumes $B \in \text{set } (\text{tr}_{pc} A D)$
 shows $\text{ik}_{st} (\text{unlabel } B) = \text{ik}_{sst} (\text{unlabel } A)$
 <proof>

lemma `tr_par_deduct_iff`:
 assumes $B \in \text{set } (\text{tr}_{pc} A D)$
 shows $\text{ik}_{st} (\text{unlabel } B) \cdot_{set} I \vdash t \longleftrightarrow \text{ik}_{sst} (\text{unlabel } A) \cdot_{set} I \vdash t$
 <proof>

lemma `tr_par_vars_subset`:
 assumes $A' \in \text{set } (\text{tr}_{pc} A D)$
 shows $\text{fv}_{lst} A' \subseteq \text{fv}_{sst} (\text{unlabel } A) \cup \text{fv}_{pairs} (\text{unlabel } D)$ (is ?P)
 and $\text{bvars}_{lst} A' \subseteq \text{bvars}_{sst} (\text{unlabel } A)$ (is ?Q)
 <proof>

lemma `tr_par_vars_disj`:
 assumes $A' \in \text{set } (\text{tr}_{pc} A D)$ $\text{fv}_{pairs} (\text{unlabel } D) \cap \text{bvars}_{sst} (\text{unlabel } A) = \{\}$
 and $\text{fv}_{sst} (\text{unlabel } A) \cap \text{bvars}_{sst} (\text{unlabel } A) = \{\}$
 shows $\text{fv}_{lst} A' \cap \text{bvars}_{lst} A' = \{\}$
 <proof>

lemma `tr_par_trms_subset`:
 assumes $A' \in \text{set } (\text{tr}_{pc} A D)$
 shows $\text{trms}_{lst} A' \subseteq \text{trms}_{sst} (\text{unlabel } A) \cup \text{pair } \text{ ` setops}_{sst} (\text{unlabel } A) \cup \text{pair } \text{ ` snd } \text{ ` set } D$
 <proof>

lemma `tr_par_wf_trms`:
 assumes $A' \in \text{set } (\text{tr}_{pc} A [])$ $\text{wf}_{trms} (\text{trms}_{sst} (\text{unlabel } A))$
 shows $\text{wf}_{trms} (\text{trms}_{lst} A')$
 <proof>

lemma `tr_par_wf'`:
 assumes $\text{fv}_{pairs} (\text{unlabel } D) \cap \text{bvars}_{sst} (\text{unlabel } A) = \{\}$
 and $\text{fv}_{pairs} (\text{unlabel } D) \subseteq X$
 and $\text{wf}'_{sst} X (\text{unlabel } A)$ $\text{fv}_{sst} (\text{unlabel } A) \cap \text{bvars}_{sst} (\text{unlabel } A) = \{\}$
 and $A' \in \text{set } (\text{tr}_{pc} A D)$
 shows $\text{wf}_{lst} X A'$
 <proof>

lemma `tr_par_wf`:
 assumes $A' \in \text{set } (\text{tr}_{pc} A [])$
 and $\text{wf}_{sst} (\text{unlabel } A)$
 and $\text{wf}_{trms} (\text{trms}_{lsst} A)$
 shows $\text{wf}_{lst} \{\} A'$
 and $\text{wf}_{trms} (\text{trms}_{lst} A')$
 and $\text{fv}_{lst} A' \cap \text{bvars}_{lst} A' = \{\}$
 <proof>

lemma tr_par_tfr_{sstp}:

```

  assumes "A' ∈ set (trpc A D)" "list_all tfrsstp (unlabel A)"
  and "fvsst (unlabel A) ∩ bvarssst (unlabel A) = {}" (is "?P0 A D")
  and "fvpairs (unlabel D) ∩ bvarssst (unlabel A) = {}" (is "?P1 A D")
  and "∀t ∈ pair ` setopssst (unlabel A) ∪ pair ` snd ` set D.
      ∀t' ∈ pair ` setopssst (unlabel A) ∪ pair ` snd ` set D.
          (∃δ. Unifier δ t t') → Γ t = Γ t'" (is "?P3 A D")
  shows "list_all tfrstp (unlabel A)"

```

<proof>

lemma tr_par_tfr:

```

  assumes "A' ∈ set (trpc A [])" and "tfrsst (unlabel A)"
  and "fvsst (unlabel A) ∩ bvarssst (unlabel A) = {}"
  shows "tfrst (unlabel A)"

```

<proof>

lemma tr_par_proj:

```

  assumes "B ∈ set (trpc A D)"
  shows "proj n B ∈ set (trpc (proj n A) (proj n D))"

```

<proof>

lemma tr_par_preserves_typing_cond:

```

  assumes "par_complsst A Sec" "typing_condsst (unlabel A)" "A' ∈ set (trpc A [])"
  shows "typing_cond (unlabel A)"

```

<proof>

lemma tr_par_preserves_par_comp:

```

  assumes "par_complsst A Sec" "A' ∈ set (trpc A [])"
  shows "par_comp A' Sec"

```

<proof>

lemma tr_leaking_prefix_exists:

```

  assumes "A' ∈ set (trpc A [])" "prefix B A'" "ikst (proj_unl n B) ·set I ⊢ t · I"
  shows "∃C D. prefix C B ∧ prefix D A ∧ C ∈ set (trpc D []) ∧ (ikst (proj_unl n C) ·set I ⊢ t · I)"

```

<proof>

6.2.5 Theorem: Semantic Equivalence of Translation

context

begin

An alternative version of the translation that does not perform database-state projections. It is used as an intermediate step in the proof of semantic equivalence.

private fun tr'_{pc}:

```

  ("('fun,'var,'lbl) labeled_stateful_strand ⇒ ('fun,'var,'lbl) labeleddbstatelist
   ⇒ ('fun,'var,'lbl) labeled_strand list")

```

where

```

  "tr'pc [] D = [[]]"
  | "tr'pc ((i,send⟨t⟩)#A) D = map ((#) (i,send⟨t⟩st)) (tr'pc A D)"
  | "tr'pc ((i,receive⟨t⟩)#A) D = map ((#) (i,receive⟨t⟩st)) (tr'pc A D)"
  | "tr'pc ((i,⟨ac: t ≐ t'⟩)#A) D = map ((#) (i,⟨ac: t ≐ t'⟩st)) (tr'pc A D)"
  | "tr'pc ((i,insert⟨t,s⟩)#A) D = tr'pc A (List.insert (i,(t,s)) D)"
  | "tr'pc ((i,delete⟨t,s⟩)#A) D = (
      concat (map (λDi. map (λB. (map (λd. (i,⟨check: (pair (t,s)) ≐ (pair (snd d))⟩st)) Di)@
          (map (λd. (i,∀ []⟨≠: [(pair (t,s), pair (snd d))⟩st])
              [d←D. d ∉ set Di])@B)
          (tr'pc A [d←D. d ∉ set Di]))
          (subseqs D)))"
  | "tr'pc ((i,⟨ac: t ∈ s⟩)#A) D =
      concat (map (λB. map (λd. (i,⟨ac: (pair (t,s)) ≐ (pair (snd d))⟩st)#B) D) (tr'pc A D))"
  | "tr'pc ((i,∀X⟨≠: F ∨ ≠: F'⟩)#A) D =
      map ((@) (map (λG. (i,∀X⟨≠: (F@G)⟩st)) (trpairs F' (map snd D)))) (tr'pc A D)"

```


Part 1

```
private lemma tr'_par_iff_unlabel_tr:
  assumes "∀ (i,p) ∈ setopslsst A ∪ set D.
          ∀ (j,q) ∈ setopslsst A ∪ set D.
          p = q → i = j"
  shows "(∃ C ∈ set (tr'pc A D). B = unlabel C) ↔ B ∈ set (tr (unlabel A) (unlabel D))"
  (is "?A ↔ ?B")
⟨proof⟩
```

Part 2

```
private lemma tr_par_iff_tr'_par:
  assumes "∀ (i,p) ∈ setopslsst A ∪ set D. ∀ (j,q) ∈ setopslsst A ∪ set D.
          (∃ δ. Unifier δ (pair p) (pair q)) → i = j"
  (is "?R3 A D")
  and "∀ (l,t,s) ∈ set D. (fv t ∪ fv s) ∩ bvarssst (unlabel A) = {}" (is "?R4 A D")
  and "fvsst (unlabel A) ∩ bvarssst (unlabel A) = {}" (is "?R5 A D")
  shows "(∃ B ∈ set (trpc A D). ⟦M; unlabel B⟧d I) ↔ (∃ C ∈ set (tr'pc A D). ⟦M; unlabel C⟧d I)"
  (is "?P ↔ ?Q")
⟨proof⟩
```

Part 3

```
private lemma tr'_par_sem_equiv:
  assumes "∀ (l,t,s) ∈ set D. (fv t ∪ fv s) ∩ bvarssst (unlabel A) = {}"
  and "fvsst (unlabel A) ∩ bvarssst (unlabel A) = {}" "ground M"
  and "∀ (i,p) ∈ setopslsst A ∪ set D. ∀ (j,q) ∈ setopslsst A ∪ set D.
          (∃ δ. Unifier δ (pair p) (pair q)) → i = j" (is "?R A D")
  and I: "interpretationsubst I"
  shows "⟦M; set (unlabel D) ·pset I; unlabel A⟧s I ↔ (∃ B ∈ set (tr'pc A D). ⟦M; unlabel B⟧d I)"
  (is "?P ↔ ?Q")
⟨proof⟩
```

Part 4

```
lemma tr_par_sem_equiv:
  assumes "∀ (l,t,s) ∈ set D. (fv t ∪ fv s) ∩ bvarssst (unlabel A) = {}"
  and "fvsst (unlabel A) ∩ bvarssst (unlabel A) = {}" "ground M"
  and "∀ (i,p) ∈ setopslsst A ∪ set D. ∀ (j,q) ∈ setopslsst A ∪ set D.
          (∃ δ. Unifier δ (pair p) (pair q)) → i = j"
  and I: "interpretationsubst I"
  shows "⟦M; set (unlabel D) ·pset I; unlabel A⟧s I ↔ (∃ B ∈ set (trpc A D). ⟦M; unlabel B⟧d I)"
  (is "?P ↔ ?Q")
⟨proof⟩
```

end

6.2.6 Theorem: The Stateful Compositionality Result, on the Constraint Level

```
theorem par_comp_constr_stateful:
  assumes A: "par_complsst A Sec" "typing_condsst (unlabel A)"
  and I: "I ⊢s unlabel A" "interpretationsubst I"
  shows "∃ Iτ. interpretationsubst Iτ ∧ wtsubst Iτ ∧ wftrms (subst_range Iτ) ∧ (Iτ ⊢s unlabel A) ∧
          ((∀ n. Iτ ⊢s proj_unl n A) ∨ (∃ A'. prefix A' A ∧ (A' leaks Sec under Iτ)))"
⟨proof⟩
```

6.2.7 Theorem: The Stateful Compositionality Result, on the Protocol Level

```
abbreviation wflsst where
  "wflsst V A ≡ wf'sst V (unlabel A)"
```

We state our result on the level of protocol traces (i.e., the constraints reachable in a symbolic execution of the actual protocol). Hence, we do not need to convert protocol strands to intruder constraints in the following well-formedness definitions.

definition $wf_{l_{ssts}}$:: "('fun, 'var, 'lbl) labeled_stateful_strand set \Rightarrow bool" where
 $wf_{l_{ssts}} S \equiv (\forall A \in S. wf_{l_{sst}} \{A\}) \wedge (\forall A \in S. \forall A' \in S. fv_{l_{sst}} A \cap bvars_{l_{sst}} A' = \{\})"$

definition $wf_{l_{ssts}}'$::
 "('fun, 'var, 'lbl) labeled_stateful_strand set \Rightarrow ('fun, 'var, 'lbl) labeled_stateful_strand \Rightarrow bool"
 where

$wf_{l_{ssts}}' S A \equiv (\forall A' \in S. wf'_{sst} (wf_{restrictedvars_{l_{sst}}} A) (unlabel A')) \wedge$
 $(\forall A' \in S. \forall A'' \in S. fv_{l_{sst}} A' \cap bvars_{l_{sst}} A'' = \{\}) \wedge$
 $(\forall A' \in S. fv_{l_{sst}} A' \cap bvars_{l_{sst}} A = \{\}) \wedge$
 $(\forall A' \in S. fv_{l_{sst}} A \cap bvars_{l_{sst}} A' = \{\})"$

definition $typing_cond_prot_stateful$ where

$typing_cond_prot_stateful P \equiv$
 $wf_{l_{ssts}} P \wedge$
 $tfr_{set} (\bigcup (trms_{l_{sst}} \setminus P) \cup pair \setminus \bigcup (setops_{sst} \setminus unlabel \setminus P)) \wedge$
 $wf_{trms} (\bigcup (trms_{l_{sst}} \setminus P)) \wedge$
 $(\forall S \in P. list_all tfr_{sstp} (unlabel S))"$

definition $par_comp_prot_stateful$ where

$par_comp_prot_stateful P Sec \equiv$
 $(\forall l1 l2. l1 \neq l2 \rightarrow$
 $GSMP_disjoint (\bigcup A \in P. trms_{sst} (proj_unl l1 A) \cup pair \setminus setops_{sst} (proj_unl l1 A))$
 $(\bigcup A \in P. trms_{sst} (proj_unl l2 A) \cup pair \setminus setops_{sst} (proj_unl l2 A)) Sec) \wedge$
 $ground Sec \wedge (\forall s \in Sec. \forall s' \in subterms s. \{\} \vdash_c s' \vee s' \in Sec) \wedge$
 $(\forall (i,p) \in \bigcup A \in P. setops_{l_{sst}} A. \forall (j,q) \in \bigcup A \in P. setops_{l_{sst}} A.$
 $(\exists \delta. Unifier \delta (pair p) (pair q) \rightarrow i = j) \wedge$
 $typing_cond_prot_stateful P"$

definition $component_secure_prot_stateful$ where

$component_secure_prot_stateful n P Sec attack \equiv$
 $(\forall A \in P. suffix [(ln n, Send (Fun attack []))] A \rightarrow$
 $(\forall \mathcal{I}_\tau. (interpretation_{subst} \mathcal{I}_\tau \wedge wt_{subst} \mathcal{I}_\tau \wedge wf_{trms} (subst_range \mathcal{I}_\tau)) \rightarrow$
 $\neg(\mathcal{I}_\tau \models_s (proj_unl n A)) \wedge$
 $(\forall A'. prefix A' A \rightarrow$
 $(\forall t \in Sec\text{-declassified}_{l_{sst}} A' \mathcal{I}_\tau. \neg(\mathcal{I}_\tau \models_s (proj_unl n A'@[Send t])))$))"

definition $component_leaks_stateful$ where

$component_leaks_stateful n A Sec \equiv$
 $(\exists A' \mathcal{I}_\tau. interpretation_{subst} \mathcal{I}_\tau \wedge wt_{subst} \mathcal{I}_\tau \wedge wf_{trms} (subst_range \mathcal{I}_\tau) \wedge prefix A' A \wedge$
 $(\exists t \in Sec - declassified_{l_{sst}} A' \mathcal{I}_\tau. (\mathcal{I}_\tau \models_s (proj_unl n A'@[Send t])))$ "

definition $unsat_stateful$ where

$unsat_stateful A \equiv (\forall \mathcal{I}. interpretation_{subst} \mathcal{I} \rightarrow \neg(\mathcal{I} \models_s unlabel A))"$

lemma $wf_{l_{ssts}}_eqs_wf_{l_{ssts}}'$ [simp]: " $wf_{l_{ssts}} S = wf_{l_{ssts}}' S []$ "

$\langle proof \rangle$

lemma $par_comp_prot_impl_par_comp_stateful$:

assumes " $par_comp_prot_stateful P Sec$ " " $A \in P$ "

shows " $par_comp_{l_{sst}} A Sec$ "

$\langle proof \rangle$

lemma $typing_cond_prot_impl_typing_cond_stateful$:

assumes " $typing_cond_prot_stateful P$ " " $A \in P$ "

shows " $typing_cond_{sst} (unlabel A)$ "

$\langle proof \rangle$

theorem $par_comp_constr_prot_stateful$:

assumes P : " $P = composed_prot Pi$ " " $par_comp_prot_stateful P Sec$ " " $\forall n. component_prot n (Pi n)$ "

and $left_secure$: " $component_secure_prot_stateful n (Pi n) Sec attack$ "

shows " $\forall A \in P. suffix [(ln n, Send (Fun attack []))] A \rightarrow$ "

$unsat_stateful A \vee (\exists m. n \neq m \wedge component_leaks_stateful m A Sec)$ "

$\langle proof \rangle$

end

6.2.8 Automated Compositionality Conditions

definition `comp_GSMP_disjoint` where

```
"comp_GSMP_disjoint public arity Ana  $\Gamma$  A' B' A B C  $\equiv$ 
  let B $\delta$  = B ·list var_rename (max_var_set (fvset (set A)))
  in has_all_wt_instances_of  $\Gamma$  (set A') (set A)  $\wedge$ 
     has_all_wt_instances_of  $\Gamma$  (set B') (set B $\delta$ )  $\wedge$ 
     finite_SMP_representation arity Ana  $\Gamma$  A  $\wedge$ 
     finite_SMP_representation arity Ana  $\Gamma$  B $\delta$   $\wedge$ 
     ( $\forall t \in \text{set } A. \forall s \in \text{set } B\delta. \Gamma t = \Gamma s \wedge \text{mgu } t s \neq \text{None} \rightarrow$ 
       (intruder_synth' public arity {} t  $\wedge$  intruder_synth' public arity {} s)  $\vee$ 
       ( $\exists u \in \text{set } C. \text{is\_wt\_instance\_of\_cond } \Gamma t u) \wedge (\exists u \in \text{set } C. \text{is\_wt\_instance\_of\_cond } \Gamma s u))"$ 
```

definition `comp_par_complsst` where

```
"comp_par_complsst public arity Ana  $\Gamma$  pair_fun A M C  $\equiv$ 
  let L = remdups (map (the_LabelN  $\circ$  fst) (filter (Not  $\circ$  is_LabelS) A));
      MPO =  $\lambda B. \text{remdups } (\text{trms\_list}_{sst} B @ \text{map } (\text{pair}' \text{pair\_fun}) (\text{setops\_list}_{sst} B));$ 
      pr =  $\lambda l. \text{MPO } (\text{proj\_unl } l A)$ 
  in length L > 1  $\wedge$ 
     list_all (wftrm' arity) (MPO (unlabel A))  $\wedge$ 
     list_all (wftrm' arity) C  $\wedge$ 
     has_all_wt_instances_of  $\Gamma$  (subtermsset (set C)) (set C)  $\wedge$ 
     is_TComp_var_instance_closed  $\Gamma$  C  $\wedge$ 
     ( $\forall i \in \text{set } L. \forall j \in \text{set } L. i \neq j \rightarrow$ 
       comp_GSMP_disjoint public arity Ana  $\Gamma$  (pr i) (pr j) (M i) (M j) C)  $\wedge$ 
     ( $\forall (i,p) \in \text{setops}_{l_{sst}} A. \forall (j,q) \in \text{setops}_{l_{sst}} A. i \neq j \rightarrow$ 
       (let s = pair' pair_fun p; t = pair' pair_fun q
         in mgu s (t · var_rename (max_var s)) = None))"
```

locale `labeled_stateful_typed_model'` =

```
stateful_typed_model' arity public Ana  $\Gamma$  Pair
+ labeled_typed_model' arity public Ana  $\Gamma$  label_witness1 label_witness2
for arity::"fun  $\Rightarrow$  nat"
and public::"fun  $\Rightarrow$  bool"
and Ana::"('fun,('atom::finite) term_type  $\times$  nat)) term
   $\Rightarrow$  (('fun,('fun,('atom) term_type  $\times$  nat)) term list
     $\times$  ('fun,('fun,('atom) term_type  $\times$  nat)) term list)"
and  $\Gamma$ ::"('fun,('fun,('atom) term_type  $\times$  nat)) term  $\Rightarrow$  ('fun,('atom) term_type)"
and Pair::"fun"
and label_witness1::"lbl"
and label_witness2::"lbl"
```

begin

sublocale `labeled_stateful_typed_model`

`<proof>`

lemma `GSMP_disjoint_if_comp_GSMP_disjoint`:

```
defines "f  $\equiv$   $\lambda M. \{t \cdot \delta \mid t \delta. t \in M \wedge \text{wt}_{subst} \delta \wedge \text{wf}_{trms} (\text{subst\_range } \delta) \wedge \text{fv } (t \cdot \delta) = \{\}\}"$ 
assumes AB'_wf: "list_all (wftrm' arity) A'" "list_all (wftrm' arity) B'"
       and C_wf: "list_all (wftrm' arity) C"
       and AB'_disj: "comp_GSMP_disjoint public arity Ana  $\Gamma$  A' B' A B C"
shows "GSMP_disjoint (set A') (set B') ((f (set C)) - {m. {}  $\vdash_c$  m})"
<proof>
```

lemma `par_complsst_if_comp_par_complsst`:

```
defines "f  $\equiv$   $\lambda M. \{t \cdot \delta \mid t \delta. t \in M \wedge \text{wt}_{subst} \delta \wedge \text{wf}_{trms} (\text{subst\_range } \delta) \wedge \text{fv } (t \cdot \delta) = \{\}\}"$ 
assumes A: "comp_par_complsst public arity Ana  $\Gamma$  Pair A M C"
shows "par_complsst A ((f (set C)) - {m. {}  $\vdash_c$  m})"
<proof>
```

```

lemma par_complssst_if_comp_par_complssst':
  defines "f ≡ λM. {t · δ | t δ. t ∈ M ∧ wtsubst δ ∧ wftrms (subst_range δ) ∧ fv (t · δ) = {}}"
  assumes a: "comp_par_complssst public arity Ana Γ Pair A M C"
    and B: "∀b ∈ set B. ∃a ∈ set A. ∃δ. b = a ·lssstp δ ∧ wtsubst δ ∧ wftrms (subst_range δ)"
      (is "∀b ∈ set B. ∃a ∈ set A. ∃δ. b = a ·lssstp δ ∧ ?D δ")
  shows "par_complssst B ((f (set C)) - {m. {} ⊢c m})"
  <proof>

end

end

```

7 Examples

In this chapter, we present two examples illustrating our results: In section 7 we show that the TLS example from [2] is type-flaw resistant. In section 7 we show that the keyserver examples from [3, 4] are also type-flaw resistant and that the steps of the composed keyserver protocol from [4] satisfy our conditions for protocol composition.

7.1 Proving Type-Flaw Resistance of the TLS Handshake Protocol (Example_TLS)

```
theory Example_TLS
imports "../Typed_Model"
begin
```

```
declare [[code_timing]]
```

7.1.1 TLS example: Datatypes and functions setup

```
datatype ex_atom = PrivKey | SymKey | PubConst | Agent | Nonce | Bot
```

```
datatype ex_fun =
  clientHello | clientKeyExchange | clientFinished
| serverHello | serverCert | serverHelloDone
| finished | changeCipher | x509 | prfun | master | pmsForm
| sign | hash | crypt | pub | concat | privkey nat
| pubconst ex_atom nat
```

```
type_synonym ex_type = "(ex_fun, ex_atom) term_type"
type_synonym ex_var = "ex_type × nat"
```

```
instance ex_atom::finite
⟨proof⟩
```

```
type_synonym ex_term = "(ex_fun, ex_var) term"
type_synonym ex_terms = "(ex_fun, ex_var) terms"
```

```
primrec arity::"ex_fun ⇒ nat" where
```

```
  "arity changeCipher = 0"
| "arity clientFinished = 4"
| "arity clientHello = 5"
| "arity clientKeyExchange = 1"
| "arity concat = 5"
| "arity crypt = 2"
| "arity finished = 1"
| "arity hash = 1"
| "arity master = 3"
| "arity pmsForm = 1"
| "arity prfun = 1"
| "arity (privkey _) = 0"
| "arity pub = 1"
| "arity (pubconst _ _) = 0"
| "arity serverCert = 1"
| "arity serverHello = 5"
| "arity serverHelloDone = 0"
| "arity sign = 2"
```

7 Examples

```
| "arity x509 = 2"

fun public::"ex_fun ⇒ bool" where
  "public (privkey _) = False"
| "public _ = True"

fun Anacrypt::"ex_term list ⇒ (ex_term list × ex_term list)" where
  "Anacrypt [Fun pub [k],m] = ([k], [m])"
| "Anacrypt _ = ([], [])"

fun Anasign::"ex_term list ⇒ (ex_term list × ex_term list)" where
  "Anasign [k,m] = ([], [m])"
| "Anasign _ = ([], [])"

fun Ana::"ex_term ⇒ (ex_term list × ex_term list)" where
  "Ana (Fun crypt T) = Anacrypt T"
| "Ana (Fun finished T) = ([], T)"
| "Ana (Fun master T) = ([], T)"
| "Ana (Fun pmsForm T) = ([], T)"
| "Ana (Fun serverCert T) = ([], T)"
| "Ana (Fun serverHello T) = ([], T)"
| "Ana (Fun sign T) = Anasign T"
| "Ana (Fun x509 T) = ([], T)"
| "Ana _ = ([], [])"
```

7.1.2 TLS example: Locale interpretation

```
lemma assm1:
  "Ana t = (K,M) ⇒ fvset (set K) ⊆ fv t"
  "Ana t = (K,M) ⇒ (∧g S'. Fun g S' ⊆ t ⇒ length S' = arity g)
    ⇒ k ∈ set K ⇒ Fun f T' ⊆ k ⇒ length T' = arity f"
  "Ana t = (K,M) ⇒ K ≠ [] ∨ M ≠ [] ⇒ Ana (t · δ) = (K ·list δ, M ·list δ)"
⟨proof⟩
```

```
lemma assm2: "Ana (Fun f T) = (K, M) ⇒ set M ⊆ set T"
⟨proof⟩
```

```
lemma assm6: "0 < arity f ⇒ public f" ⟨proof⟩
```

```
global_interpretation im: intruder_model arity public Ana
  defines wftrm = "im.wftrm"
  and wftrms = "im.wftrms"
⟨proof⟩
```

7.1.3 TLS Example: Typing function

```
definition Γv::"ex_var ⇒ ex_type" where
  "Γv v = (if (∀t ∈ subterms (fst v). case t of
    (TComp f T) ⇒ arity f > 0 ∧ arity f = length T
    | _ ⇒ True)
  then fst v else TAtom Bot)"
```

```
fun Γ::"ex_term ⇒ ex_type" where
  "Γ (Var v) = Γv v"
| "Γ (Fun (privkey _) _) = TAtom PrivKey"
| "Γ (Fun changeCipher _) = TAtom PubConst"
| "Γ (Fun serverHelloDone _) = TAtom PubConst"
| "Γ (Fun (pubconst τ _) _) = TAtom τ"
| "Γ (Fun f T) = TComp f (map Γ T)"
```

7.1.4 TLS Example: Locale interpretation (typed model)

```
lemma assm7: "arity c = 0 ⇒ ∃a. ∀X. Γ (Fun c X) = TAtom a" ⟨proof⟩
```

lemma *assm8*: "0 < arity f \implies Γ (Fun f X) = TComp f (map Γ X)" *<proof>*

lemma *assm9*: "infinite {c. Γ (Fun c []) = TAtom a \wedge public c}"
<proof>

lemma *assm10*: "TComp f T \sqsubseteq Γ t \implies arity f > 0"
<proof>

lemma *assm11*: "im.wf_{term} (Γ (Var x))"
<proof>

lemma *assm12*: " Γ (Var (τ , n)) = Γ (Var (τ , m))"
<proof>

lemma *Ana_const*: "arity c = 0 \implies Ana (Fun c T) = ([], [])"
<proof>

lemma *Ana_keys_subterm*: "Ana t = (K,T) \implies k \in set K \implies k \sqsubset t"
<proof>

global_interpretation *tm*: typed_model' arity public Ana Γ
<proof>

7.1.5 TLS example: Proving type-flaw resistance

abbreviation $\Gamma_v_clientHello$ where

" $\Gamma_v_clientHello \equiv$
TComp clientHello [TAtom Nonce, TAtom Nonce, TAtom Nonce, TAtom Nonce, TAtom Nonce]"

abbreviation $\Gamma_v_serverHello$ where

" $\Gamma_v_serverHello \equiv$
TComp serverHello [TAtom Nonce, TAtom Nonce, TAtom Nonce, TAtom Nonce, TAtom Nonce]"

abbreviation Γ_v_pub where

" $\Gamma_v_pub \equiv$ TComp pub [TAtom PrivKey]"

abbreviation Γ_v_x509 where

" $\Gamma_v_x509 \equiv$ TComp x509 [TAtom Agent, Γ_v_pub]"

abbreviation Γ_v_sign where

" $\Gamma_v_sign \equiv$ TComp sign [TAtom PrivKey, Γ_v_x509]"

abbreviation $\Gamma_v_serverCert$ where

" $\Gamma_v_serverCert \equiv$ TComp serverCert [Γ_v_sign]"

abbreviation $\Gamma_v_pmsForm$ where

" $\Gamma_v_pmsForm \equiv$ TComp pmsForm [TAtom SymKey]"

abbreviation Γ_v_crypt where

" $\Gamma_v_crypt \equiv$ TComp crypt [Γ_v_pub , $\Gamma_v_pmsForm$]"

abbreviation $\Gamma_v_clientKeyExchange$ where

" $\Gamma_v_clientKeyExchange \equiv$
TComp clientKeyExchange [Γ_v_crypt]"

abbreviation $\Gamma_v_HSMsigs$ where

" $\Gamma_v_HSMsigs \equiv$ TComp concat [
 $\Gamma_v_clientHello$,
 $\Gamma_v_serverHello$,
 $\Gamma_v_serverCert$,
TAtom PubConst,
 $\Gamma_v_clientKeyExchange$]"

```

abbreviation "T1 n ≡ Var (TAtom Nonce,n)"
abbreviation "T2 n ≡ Var (TAtom Nonce,n)"
abbreviation "RA n ≡ Var (TAtom Nonce,n)"
abbreviation "RB n ≡ Var (TAtom Nonce,n)"
abbreviation "S n ≡ Var (TAtom Nonce,n)"
abbreviation "Cipher n ≡ Var (TAtom Nonce,n)"
abbreviation "Comp n ≡ Var (TAtom Nonce,n)"
abbreviation "B n ≡ Var (TAtom Agent,n)"
abbreviation "Prca n ≡ Var (TAtom PrivKey,n)"
abbreviation "PMS n ≡ Var (TAtom SymKey,n)"
abbreviation "PB n ≡ Var (TComp pub [TAtom PrivKey],n)"
abbreviation "HSMsigs n ≡ Var (Γv_HSMsigs,n)"

```

Defining the over-approximation set

```

abbreviation clientHellotrm where
  "clientHellotrm ≡ Fun clientHello [T1 0, RA 1, S 2, Cipher 3, Comp 4]"

abbreviation serverHellotrm where
  "serverHellotrm ≡ Fun serverHello [T2 0, RB 1, S 2, Cipher 3, Comp 4]"

abbreviation serverCerttrm where
  "serverCerttrm ≡ Fun serverCert [Fun sign [Prca 0, Fun x509 [B 1, PB 2]]]"

abbreviation serverHelloDonetrm where
  "serverHelloDonetrm ≡ Fun serverHelloDone []"

abbreviation clientKeyExchangetrm where
  "clientKeyExchangetrm ≡ Fun clientKeyExchange [Fun crypt [PB 0, Fun pmsForm [PMS 1]]]"

abbreviation changeCiphertrm where
  "changeCiphertrm ≡ Fun changeCipher []"

abbreviation finishedtrm where
  "finishedtrm ≡ Fun finished [Fun prfun [
    Fun clientFinished [
      Fun prfun [Fun master [PMS 0, RA 1, RB 2]],
      RA 3, RB 4, Fun hash [HSMsigs 5]
    ]
  ]]"

definition MTLS::"ex_term list" where
  "MTLS ≡ [
    clientHellotrm,
    serverHellotrm,
    serverCerttrm,
    serverHelloDonetrm,
    clientKeyExchangetrm,
    changeCiphertrm,
    finishedtrm
  ]"

```

7.1.6 Theorem: The TLS handshake protocol is type-flaw resistant

```

theorem "tm.tfrset (set MTLS)"
⟨proof⟩

end

```


7.2 The Keyserver Example (Example_Keyserver)

```
theory Example_Keyserver
imports "../Stateful_Compositionality"
begin
```

```
declare [[code_timing]]
```

7.2.1 Setup

Datatypes and functions setup

```
datatype ex_lbl = Label1 ("1") | Label2 ("2")
```

```
datatype ex_atom =
  Agent | Value | Attack | PrivFunSec
| Bot
```

```
datatype ex_fun =
  ring | valid | revoked | events | beginauth nat | endauth nat | pubkeys | seen
| invkey | tuple | tuple' | attack nat
| sign | crypt | update | pw
| encodingsecret | pubkey nat
| pubconst ex_atom nat
```

```
type_synonym ex_type = "(ex_fun, ex_atom) term_type"
type_synonym ex_var = "ex_type × nat"
```

```
lemma ex_atom_UNIV:
  "(UNIV::ex_atom set) = {Agent, Value, Attack, PrivFunSec, Bot}"
⟨proof⟩
```

```
instance ex_atom::finite
⟨proof⟩
```

```
lemma ex_lbl_UNIV:
  "(UNIV::ex_lbl set) = {Label1, Label2}"
⟨proof⟩
```

```
type_synonym ex_term = "(ex_fun, ex_var) term"
type_synonym ex_terms = "(ex_fun, ex_var) terms"
```

```
primrec arity::"ex_fun ⇒ nat" where
```

```
  "arity ring = 2"
| "arity valid = 3"
| "arity revoked = 3"
| "arity events = 1"
| "arity (beginauth _) = 3"
| "arity (endauth _) = 3"
| "arity pubkeys = 2"
| "arity seen = 2"
| "arity invkey = 2"
| "arity tuple = 2"
| "arity tuple' = 2"
| "arity (attack _) = 0"
| "arity sign = 2"
| "arity crypt = 2"
| "arity update = 4"
| "arity pw = 2"
| "arity (pubkey _) = 0"
| "arity encodingsecret = 0"
| "arity (pubconst _ _) = 0"
```

7 Examples

```

fun public::"ex_fun  $\Rightarrow$  bool" where
  "public (pubkey _) = False"
  | "public encodingsecret = False"
  | "public _ = True"

fun Anacrypt::"ex_term list  $\Rightarrow$  (ex_term list  $\times$  ex_term list)" where
  "Anacrypt [k,m] = ([Fun invkey [Fun encodingsecret [], k]], [m])"
  | "Anacrypt _ = ([], [])"

fun Anasign::"ex_term list  $\Rightarrow$  (ex_term list  $\times$  ex_term list)" where
  "Anasign [k,m] = ([], [m])"
  | "Anasign _ = ([], [])"

fun Ana::"ex_term  $\Rightarrow$  (ex_term list  $\times$  ex_term list)" where
  "Ana (Fun tuple T) = ([], T)"
  | "Ana (Fun tuple' T) = ([], T)"
  | "Ana (Fun sign T) = Anasign T"
  | "Ana (Fun crypt T) = Anacrypt T"
  | "Ana _ = ([], [])"

```

Keyserver example: Locale interpretation

```

lemma assm1:
  "Ana t = (K,M)  $\Longrightarrow$  fvset (set K)  $\subseteq$  fv t"
  "Ana t = (K,M)  $\Longrightarrow$  ( $\bigwedge$ g S'. Fun g S'  $\sqsubseteq$  t  $\Longrightarrow$  length S' = arity g)
     $\Longrightarrow$  k  $\in$  set K  $\Longrightarrow$  Fun f T'  $\sqsubseteq$  k  $\Longrightarrow$  length T' = arity f"
  "Ana t = (K,M)  $\Longrightarrow$  K  $\neq$  []  $\vee$  M  $\neq$  []  $\Longrightarrow$  Ana (t  $\cdot$   $\delta$ ) = (K  $\cdot$ list  $\delta$ , M  $\cdot$ list  $\delta$ )"
  <proof>

```

```

lemma assm2: "Ana (Fun f T) = (K, M)  $\Longrightarrow$  set M  $\subseteq$  set T"
  <proof>

```

```

lemma assm6: "0 < arity f  $\Longrightarrow$  public f" <proof>

```

```

global interpretation im: intruder_model arity public Ana
  defines wftrm = "im.wftrm"
  <proof>

```

```

type_synonym ex_strand_step = "(ex_fun,ex_var) strand_step"
type_synonym ex_strand = "(ex_fun,ex_var) strand"

```

Typing function

```

definition  $\Gamma_v$ ::"ex_var  $\Rightarrow$  ex_type" where
  " $\Gamma_v$  v = (if ( $\forall$ t  $\in$  subterms (fst v). case t of
    (TComp f T)  $\Rightarrow$  arity f > 0  $\wedge$  arity f = length T
    | _  $\Rightarrow$  True)
    then fst v else TAtom Bot)"

```

```

fun  $\Gamma$ ::"ex_term  $\Rightarrow$  ex_type" where
  " $\Gamma$  (Var v) =  $\Gamma_v$  v"
  | " $\Gamma$  (Fun (attack _) _) = TAtom Attack"
  | " $\Gamma$  (Fun (pubkey _) _) = TAtom Value"
  | " $\Gamma$  (Fun encodingsecret _) = TAtom PrivFunSec"
  | " $\Gamma$  (Fun (pubconst  $\tau$  _) _) = TAtom  $\tau$ "
  | " $\Gamma$  (Fun f T) = TComp f (map  $\Gamma$  T)"

```

Locale interpretation: typed model

```

lemma assm7: "arity c = 0  $\Longrightarrow$   $\exists$ a.  $\forall$ X.  $\Gamma$  (Fun c X) = TAtom a" <proof>

```

```

lemma assm8: "0 < arity f  $\Longrightarrow$   $\Gamma$  (Fun f X) = TComp f (map  $\Gamma$  X)" <proof>

```

```

lemma assm9: "infinite {c.  $\Gamma$  (Fun c []) = TAtom a  $\wedge$  public c}"

```

<proof>

lemma *assm10*: "TComp f T \sqsubseteq Γ t \implies arity f > 0"

<proof>

lemma *assm11*: "im.wf_{trm} (Γ (Var x))"

<proof>

lemma *assm12*: " Γ (Var (τ , n)) = Γ (Var (τ , m))"

<proof>

lemma *Ana_const*: "arity c = 0 \implies Ana (Fun c T) = ([], [])"

<proof>

lemma *Ana_subst'*: "Ana (Fun f T) = (K,M) \implies Ana (Fun f T \cdot δ) = (K \cdot list δ , M \cdot list δ)"

<proof>

global_interpretation *tm*: typed_model' arity public Ana Γ

<proof>

Locale interpretation: labeled stateful typed model

global_interpretation *stm*: labeled_stateful_typed_model' arity public Ana Γ tuple 1 2

<proof>

type_synonym *ex_stateful_strand_step* = "(ex_fun,ex_var) stateful_strand_step"

type_synonym *ex_stateful_strand* = "(ex_fun,ex_var) stateful_strand"

type_synonym *ex_labeled_stateful_strand_step* =
"(ex_fun,ex_var,ex_lbl) labeled_stateful_strand_step"

type_synonym *ex_labeled_stateful_strand* =
"(ex_fun,ex_var,ex_lbl) labeled_stateful_strand"

7.2.2 Theorem: Type-flaw resistance of the keyserver example from the CSF18 paper

abbreviation "PK n \equiv Var (TAtom Value,n)"

abbreviation "A n \equiv Var (TAtom Agent,n)"

abbreviation "X n \equiv (TAtom Agent,n)"

abbreviation "ringset t \equiv Fun ring [Fun encodingsecret [], t]"

abbreviation "validset t t' \equiv Fun valid [Fun encodingsecret [], t, t']"

abbreviation "revokedset t t' \equiv Fun revoked [Fun encodingsecret [], t, t']"

abbreviation "eventsset \equiv Fun events [Fun encodingsecret []]"

abbreviation *S_{ks}* :: "(ex_fun,ex_var) stateful_strand_step list" where

"S_{ks} \equiv [
 insert(Fun (attack 0) [], eventsset),
 delete(PK 0, validset (A 0) (A 0)),
 \forall (TAtom Agent,0) (PK 0 not in revokedset (A 0) (A 0)),
 \forall (TAtom Agent,0) (PK 0 not in validset (A 0) (A 0)),
 insert(PK 0, validset (A 0) (A 0)),
 insert(PK 0, ringset (A 0)),
 insert(PK 0, revokedset (A 0) (A 0)),
 select(PK 0, validset (A 0) (A 0)),
 select(PK 0, ringset (A 0)),
 receive(Fun invkey [Fun encodingsecret [], PK 0]),
 receive(Fun sign [Fun invkey [Fun encodingsecret [], PK 0], Fun tuple' [A 0, PK 0]]),
 send(Fun invkey [Fun encodingsecret [], PK 0]),
 send(Fun sign [Fun invkey [Fun encodingsecret [], PK 0], Fun tuple' [A 0, PK 0]])
]"

```
theorem "stm.tfrsst Sks"
  <proof>
```

7.2.3 Theorem: Type-flaw resistance of the keyserver examples from the ESORICS18 paper

```
abbreviation "signmsg t t' ≡ Fun sign [t, t']"
abbreviation "cryptmsg t t' ≡ Fun crypt [t, t']"
abbreviation "invkeymsg t ≡ Fun invkey [Fun encodingsecret [], t]"
abbreviation "updatemsg a b c d ≡ Fun update [a,b,c,d]"
abbreviation "pwmsg t t' ≡ Fun pw [t, t']"

abbreviation "beginauthset n t t' ≡ Fun (beginauth n) [Fun encodingsecret [], t, t']"
abbreviation "endauthset n t t' ≡ Fun (endauth n) [Fun encodingsecret [], t, t']"
abbreviation "pubkeyset t ≡ Fun pubkeys [Fun encodingsecret [], t]"
abbreviation "seenset t ≡ Fun seen [Fun encodingsecret [], t]"

declare [[coercion "Var::ex_var ⇒ ex_term"]]
declare [[coercion_enabled]]
```

```
definition S'ks::"ex_labeled_stateful_strand_step list" where
  "S'ks ≡ [
constructive_steps_from_the_first_protocol/compile_steps_are_ignored

###/R11/1
    <1, send(invkeymsg (PK 0))>,
    <*, <PK 0 in validset (A 0) (A 1)>>,
    <1, receive(Fun (attack 0) [])>,

###/R12/1
    <1, send(signmsg (invkeymsg (PK 0)) (Fun tuple' [A 0, PK 0]))>,
    <*, <PK 0 in validset (A 0) (A 1)>>,
    <*, ∀X 0, X 1<PK 0 not in validset (Var (X 0)) (Var (X 1))>>,
    <1, ∀X 0, X 1<PK 0 not in revokedset (Var (X 0)) (Var (X 1))>>,
    <*, <PK 0 not in beginauthset 0 (A 0) (A 1)>>,

###/R13/1
    <*, <PK 0 in beginauthset 0 (A 0) (A 1)>>,
    <*, <PK 0 in endauthset 0 (A 0) (A 1)>>,

###/R14/1
    <*, receive(PK 0)>,
    <*, receive(invkeymsg (PK 0))>,

###/R15/1
    <1, insert(PK 0, ringset (A 0))>,
    <*, insert(PK 0, validset (A 0) (A 1))>,
    <*, insert(PK 0, beginauthset 0 (A 0) (A 1))>,
    <*, insert(PK 0, endauthset 0 (A 0) (A 1))>,

###/R16/1
    <1, select(PK 0, ringset (A 0))>,
    <1, delete(PK 0, ringset (A 0))>,

###/R17/1
    <*, <PK 0 not in endauthset 0 (A 0) (A 1)>>,
    <*, delete(PK 0, validset (A 0) (A 1))>,
    <1, insert(PK 0, revokedset (A 0) (A 1))>,

###/R18/1
not_a_step
  ]"
```

```

stm/13/1
  ⟨1, send(PK 0)⟩,

stm/13/10/1
  ⟨1, send(Fun (attack 0) [])⟩,

stm/13/12/1
  ⟨2, send(invkeymsg (PK 0))⟩,
  ⟨*, ⟨PK 0 in validset (A 0) (A 1)⟩⟩,
  ⟨2, receive(Fun (attack 1) [])⟩,

stm/13/12/2
  ⟨2, send(cryptmsg (PK 0) (updatemsg (A 0) (A 1) (PK 1) (pwmsg (A 0) (A 1))))⟩,
  ⟨2, select(PK 0, pubkeyset (A 0))⟩,
  ⟨2, ∀X 0(PK 0 not in pubkeyset (Var (X 0)))⟩,
  ⟨2, ∀X 0(PK 0 not in seenset (Var (X 0)))⟩,

stm/13/13/2
  ⟨*, ⟨PK 0 in beginauthset 1 (A 0) (A 1)⟩⟩,
  ⟨*, ⟨PK 0 in endauthset 1 (A 0) (A 1)⟩⟩,

stm/13/14/2
  ⟨*, receive(PK 0)⟩,
  ⟨*, receive(invkeymsg (PK 0))⟩,

stm/13/15/2
  ⟨2, select(PK 0, pubkeyset (A 0))⟩,
  ⟨*, insert(PK 0, beginauthset 1 (A 0) (A 1))⟩,
  ⟨2, receive(cryptmsg (PK 0) (updatemsg (A 0) (A 1) (PK 1) (pwmsg (A 0) (A 1))))⟩,

stm/13/16/2
  ⟨*, ⟨PK 0 not in endauthset 1 (A 0) (A 1)⟩⟩,
  ⟨*, insert(PK 0, validset (A 0) (A 1))⟩,
  ⟨*, insert(PK 0, endauthset 1 (A 0) (A 1))⟩,
  ⟨2, insert(PK 0, seenset (A 0))⟩,

stm/13/17/2
  ⟨2, receive(pwmsg (A 0) (A 1))⟩,

stm/13/18/2
  stm/13/18/1

stm/13/19/2
  ⟨2, insert(PK 0, pubkeyset (A 0))⟩,

stm/13/19/2
  ⟨2, send(Fun (attack 1) [])⟩
]
"

theorem "stm.tfrsst (unlabel S'_{k_s})"
  ⟨proof⟩

```

7.2.4 Theorem: The steps of the keyserver protocols from the ESORICS18 paper satisfy the conditions for parallel composition

theorem

fixes S f

```

defines "S ≡ [PK 0, invkeymsg (PK 0), Fun encodingsecret []]@concat (
  map (λs. [s, Fun tuple [PK 0, s]])
    [validset (A 0) (A 1), beginauthset 0 (A 0) (A 1), endauthset 0 (A 0) (A 1),
     beginauthset 1 (A 0) (A 1), endauthset 1 (A 0) (A 1)])@
  [A 0]"

```

7 Examples

```
    and "f ≡ λM. {t · δ | t δ. t ∈ M ∧ tm.wtsubst δ ∧ im.wftrms (subst_range δ) ∧ fv (t · δ) = {}}"  
    and "Sec ≡ (f (set S)) - {m. im.intruder_synth {} m}"  
    shows "stm.par_complsst S' ks Sec"  
⟨proof⟩  
end
```

Bibliography

- [1] A. V. Hess. *Typing and Compositionality for Stateful Security Protocols*. PhD thesis, 2019. URL <https://orbit.dtu.dk/en/publications/typing-and-compositionality-for-stateful-security-protocols>.
- [2] A. V. Hess and S. Mödersheim. Formalizing and Proving a Typing Result for Security Protocols in Isabelle/HOL. In *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, pages 451–463. IEEE Computer Society, 2017. doi: 10.1109/CSF.2017.27.
- [3] A. V. Hess and S. Mödersheim. A Typing Result for Stateful Protocols. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 374–388. IEEE Computer Society, 2018. doi: 10.1109/CSF.2018.00034.
- [4] A. V. Hess, S. Mödersheim, and A. D. Brucker. Stateful Protocol Composition. In J. López, J. Zhou, and M. Soriano, editors, *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I*, volume 11098 of *Lecture Notes in Computer Science*, pages 427–446. Springer, 2018. doi: 10.1007/978-3-319-99073-6_21.