Formalisation and Analysis of Component Dependencies

Maria Spichkova

April 19, 2020

Abstract

This set of theories presents a formalisation in Isabelle/HOL [3] of data dependencies between components. The approach allows to analyse system structure oriented towards efficient checking of system: it aims at elaborating for a concrete system, which parts of the system (or system model) are necessary to check a given property.

Contents

1 Introduction 2

2 Case Study: Definitions 7

3 Inter-/Intracomponent dependencies 14
   3.1 Direct and indirect data dependencies between components . 18
   3.2 Components that are elementary wrt. data dependencies . . 23
   3.3 Set of components needed to check a specific property . . . 24
   3.4 Additional properties: Remote Computation . . . . . . . . . 26

4 Case Study: Verification of Properties 26
   4.1 Correct composition of components . . . . . . . . . . . . . 26
   4.2 Correct specification of the relations between channels . . . 31
   4.3 Elementary components . . . . . . . . . . . . . . . . . . . . 35
   4.4 Source components . . . . . . . . . . . . . . . . . . . . . . . 36
   4.5 Minimal sets of components to prove certain properties . . . 45
1 Introduction

In general, we don’t need complete information about the system as to check its certain property. An additional information about the system can slow the whole process down or even make it infeasible. In this theory we define constraints that allow to find/check the minimal model (and the minimal extent of the system) needed to verify a specific property. Our approach focuses on data dependencies between system components. Dependencies’ analysis results in a decomposition that gives rise to a logical system architecture, which is the most appropriate for the case of remote monitoring, testing and/or verification.

Let $CSet$ be a set of components on a certain abstraction level $L$ of logical architecture (i.e. level of refinement/decomposition, data type $\text{AbstrLevelsID}$ in our Isabelle formalisation). We denote the sets of input and output streams of a component $S$ by $I(S)$ (function $\text{IN} :: CSet \Rightarrow \text{chanID set}$ in Isabelle) and $O(S)$ (function $\text{OUT} :: CSet \Rightarrow \text{chanID set}$ in Isabelle). The set of local variables of components is defined in Isabelle by $\text{VAR}$, and the function to map component identifiers to the corresponding variables is defined by $\text{VAR} :: CSet \Rightarrow \text{varID set}$.

Please note that concrete values for these functions cannot be specified in general, because they strongly depend on a concrete system. In this paper we present a small case study in the theories $\text{DataDependenciesConcreteValues.thy}$ (specification of the system architecture on several abstraction levels) and $\text{DataDependenciesCaseStudy.thy}$ (proofs of system architectures’ properties).

Function $\text{subcomp} :: CSet \Rightarrow CSet set$ maps components to a (possibly empty) set of its subcomponents.

We specify the components’ dependencies by the function

$$\text{Sources}^L :: CSet^L \rightarrow (CSet^L)^*$$

which returns for any component identifier $A$ the corresponding (possibly empty) list of components (names) $B_1, \ldots, B_N$ that are the sources for the input data streams of $A$ (direct or indirect):

$$\text{Sources}^L(C) = D\text{Sources}^L(C) \cup \bigcup_{S \in D\text{Sources}^L(C)} \{S_1 \mid S_1 \in \text{Sources}^L(S)\}$$

Direct data dependencies are defined by the function

$$D\text{Sources}^L :: CSet^L \rightarrow (CSet^L)^*$$

$$D\text{Sources}^L(C) = \{S \mid \exists x \in I(C) \land x \in O(S)\}$$
For example, $C_1 \in DSources^L(C_2)$ means that at least one of the output channels of $C_1$ is directly connected to some of input channels of $C_2$.

$I\!D(C, y)$ denotes the subset of $I(C)$ that output channel $y$ depends upon, directly (specified in Isabelle by function $OUT\!from\!Ch:: chanID \Rightarrow chanID$ set or via local variables (specified by function $OUT\!from\!V:: chanID \Rightarrow varID$ set). For example, let the values of the output channel $y$ of component $C$ depend only on the value of the local variable $st$ that represents the current state of $C$ and is updated depending to the input messages the component receives via the channel $x$, then $I\!D(C, y) = \{x\}$. In Isabelle, $I\!D(C, y)$ is specified by function $OUT\!from:: chanID \Rightarrow varID$ set.

Based on the definition above, we can decompose system’s components to have for each component’s output channel the minimal subcomponent computing the corresponding results (we call them elementary components). An elementary component either

- should have a single output channel (in this case this component can have no local variables), or

- all it output channels are correlated, i.e. mutually depend on the same local variable(s).

If after this decomposition a single component is too complex, we can apply the decomposition strategy presented in [5].

For any component $C$, the dual function $O\!D$ returns the corresponding set $O\!D(C, x)$ of output channels depending on input $x$. This is useful for tracing, e.g., if there are some changes in the specification, properties, constraints, etc. for $x$, we can trace which other channels can be affected by these changes.

If the input part of the component’s interface is specified correctly in the sense that the component does not have any “unused” input channels, the following relation will hold: $\forall x \in I(C). \ O\!D(C, x) \neq \emptyset$.

We illustrate the presented ideas by a small case study: we show how system’s components can be decomposed to optimise the data dependencies within each single component, and after that we optimise architecture of the whole system. System $S$ (cf. also Fig. 1) has 5 components, the set $CSet$ on the level $L_0$ is defined by $\{A_1, \ldots, A_9\}$. The sets $I\!D$ of data dependencies between the components are defined in the theory $Data\!Dependencies\!Concrete\!Values.thy$. We represent the dependencies graphically using dashed lines over the component box.
Now we can decompose the system’s components according to the given $I^D$ specification. This results into the next abstraction level $L_1$ of logical architecture (cf. Fig. 2), on which all components are elementary. Thus, we obtain a (flat) architecture of system. The main feature of this architecture is that each output channel (within the system) belongs the minimal sub-component of a system computing the corresponding results. We represent this (flat) architecture as a directed graph (components become vertices and channels become edges) and apply one of the existing distributed algorithms for the decomposition into its strongly connected components, e.g. FB [2], OBF [1], or the colouring algorithm [4]. Fig. 3 presents the result of the architecture optimisation.

After optimisation of system’s architecture, we can find the minimal part of the system needed to check a specific property (cf. theory DataDependencies). A property can be represented by relations over data flows on the system’s channels, and first of all we should check the property itself, whether it reflect a real relation within a system. Let for a relation $r$, $I_r$, $O_r$ be the sets of input and output channels of the system used in this relation. For each channel from $O_r$ we recursively compute all the sets of the dependent components and corresponding input channels. Their union, restricted to the input channels of the system, should be equal to $I_r$, otherwise we should check whether the property was specified correctly.

Thus, from $O_r$ we obtain the set $outSetOfComponents$ of components having these channels as outputs, and compute the union of corresponding sources’ sets. This union together with $outSetOfComponents$ give us the minimal part of the system needed to check the property $r$: we formalise it in Isabelle by the predicate $minSetOfComponents$. 

![Figure 1: System S: Data dependencies and $I^D$ sets](image-url)
Figure 2: Components’ decomposition (level $L_1$)

For each channel and elementary component (i.e. for any component on the abstraction level $L_1$) we specify the following measures:

- measure for costs of the data transfer/upload to the cloud $UplSize(f)$: size of messages (data packages) within a data flow $f$ and frequency they are produced. This measure can be defined on the level of logical modelling, where we already know the general type of the data and can also analyse the corresponding component (or environment) model to estimate the frequency the data are produced;

- measure for requirement of using high-performance computing and cloud virtual machines, $Perf(X)$: complexity of the computation within a component $X$, which can be estimated on the level of logical modelling as well.

On this basis, we build a system architecture, optimised for remote computation. The $UplSize$ measure should be analysed only for the channels that aren’t local for the components on abstraction levels $L_2$ and $L_3$. 

Using graphical representation, we denote the channels with *UplSize* measure higher than a predefined value by thick red arrows (cf. also set *UplSizeHighLoad* in Isabelle theory *DataDependenciesConcreteValues.thy*), and the components with *Perf* measure higher than a predefined value by light green colour (cf. also set *HighPerfSet* in Isabelle theory *DataDependenciesConcreteValues.thy*), where all other channel and components are marked blue.

Fig. 4 represents a system architecture, optimised for remote computation: components from the abstraction level $L_2$ are composed together on the abstraction level $L_3$, if they are connected by at least one channel with *UplSize* measure higher than a predefined value. The components $S'_4$ and $S'_7$ have *Perf* measure higher than a predefined value, i.e. using high-performance computing and cloud virtual machines is required.

Figure 3: Architecture of $S$ (level $L_2$)
2 Case Study: Definitions

theory DataDependenciesConcreteValues
imports Main
begin

datatype CSet = sA1 | sA2 | sA3 | sA4 | sA5 | sA6 | sA7 | sA8 | sA9 |
              sA11 | sA12 | sA21 | sA12 | sA22 | sA31 | sA32 | sA41 | sA42 |
              sA71 | sA72 | sA81 | sA82 | sA91 | sA92 | sA93 |
              sS1 | sS2 | sS3 | sS4 | sS5 | sS6 | sS7 | sS8 | sS9 | sS10 | sS11 |
              sS12 | sS13 | sS14 | sS15 | sS1opt | sS4opt | sS7opt | sS11opt

datatype chanID = data1 | data2 | data3 | data4 | data5 | data6 | data7 |
                   data8 | data9 | data10 | data11 | data12 | data13 | data14 | data15 |
                   data16 | data17 | data18 | data19 | data20 | data21 | data22 | data23 | data24

datatype varID = stA1 | stA2 | stA4 | stA6

datatype AbstrLevelsID = level0 | level1 | level2 | level3

— function IN maps component ID to the set of its input channels
fun IN :: CSet ⇒ chanID set
where
  IN sA1 = { data1 }
\begin{verbatim}
| IN sA2 = { data2, data3 }
| IN sA3 = { data4, data5 }
| IN sA4 = { data6, data7, data13 }
| IN sA5 = { data8 }
| IN sA6 = { data14 }
| IN sA7 = { data15, data16 }
| IN sA8 = { data17, data18, data19, data22 }
| IN sA9 = { data20, data21 }
| IN sA11 = { data1 }
| IN sA12 = { data1 }
| IN sA21 = { data2 }
| IN sA22 = { data2, data3 }
| IN sA23 = { data2 }
| IN sA31 = { data4 }
| IN sA32 = { data5 }
| IN sA41 = { data6, data7 }
| IN sA42 = { data13 }
| IN sA71 = { data15 }
| IN sA72 = { data16 }
| IN sA81 = { data17, data22 }
| IN sA82 = { data18, data19 }
| IN sA91 = { data20 }
| IN sA92 = { data20 }
| IN sA93 = { data21 }
| IN sS1 = { data1 }
| IN sS2 = { data1 }
| IN sS3 = { data2 }
| IN sS4 = { data2 }
| IN sS5 = { data5 }
| IN sS6 = { data2, data7 }
| IN sS7 = { data13 }
| IN sS8 = { data8 }
| IN sS9 = { data14 }
| IN sS10 = { data15 }
| IN sS11 = { data16 }
| IN sS12 = { data17 }
| IN sS13 = { data20 }
| IN sS14 = { data18, data19 }
| IN sS15 = { data21 }
| IN sS1opt = { data1 }
| IN sS4opt = { data2 }
| IN sS7opt = { data13 }
| IN sS11opt = { data16, data19 }

— function OUT maps component ID to the set of its output channels
\textbf{fun} OUT :: CSet \Rightarrow \text{chanID set}
\textbf{where}

\textbf{OUT sA1} = \{ data2, data10 \}
| \textbf{OUT sA2} = \{ data4, data5, data11, data12 \}
\end{verbatim}
| OUT sA3 = { data6, data7 } |
| OUT sA4 = { data3, data8 } |
| OUT sA5 = { data9 } |
| OUT sA6 = { data15, data16 } |
| OUT sA7 = { data17, data18 } |
| OUT sA8 = { data20, data21 } |
| OUT sA9 = { data22, data23, data24 } |
| OUT sA11 = { data2 } |
| OUT sA12 = { data10 } |
| OUT sA21 = { data11 } |
| OUT sA22 = { data4, data12 } |
| OUT sA23 = { data5 } |
| OUT sA31 = { data6 } |
| OUT sA32 = { data7 } |
| OUT sA41 = { data3 } |
| OUT sA42 = { data8 } |
| OUT sA71 = { data17 } |
| OUT sA72 = { data18 } |
| OUT sA81 = { data20 } |
| OUT sA82 = { data21 } |
| OUT sA91 = { data22 } |
| OUT sA92 = { data23 } |
| OUT sA93 = { data24 } |
| OUT sS1 = { data10 } |
| OUT sS2 = { data2 } |
| OUT sS3 = { data11 } |
| OUT sS4 = { data5 } |
| OUT sS5 = { data7 } |
| OUT sS6 = { data12 } |
| OUT sS7 = { data8 } |
| OUT sS8 = { data9 } |
| OUT sS9 = { data15, data16 } |
| OUT sS10 = { data17 } |
| OUT sS11 = { data18 } |
| OUT sS12 = { data20 } |
| OUT sS13 = { data23 } |
| OUT sS14 = { data21 } |
| OUT sS15 = { data24 } |
| OUT sS1opt = { data2, data10 } |
| OUT sS4opt = { data12 } |
| OUT sS7opt = { data9 } |
| OUT sS11opt = { data24 } |

---

function VAR maps component IDs to the set of its local variables

fun VAR :: CSet ⇒ varID set

where
VAR sA1 = { stA1 }
| VAR sA2 = { stA2 }
VAR sA3 = {}
VAR sA4 = { stA4 }
VAR sA5 = {}
VAR sA6 = { stA6 }
VAR sA7 = {}
VAR sA8 = {}
VAR sA9 = {}
VAR sA11 = {}
VAR sA12 = { stA1 }
VAR sA21 = {}
VAR sA22 = { stA2 }
VAR sA23 = {}
VAR sA31 = {}
VAR sA32 = {}
VAR sA41 = { stA4 }
VAR sA42 = {}
VAR sA71 = {}
VAR sA72 = {}
VAR sA81 = {}
VAR sA82 = {}
VAR sA91 = {}
VAR sA92 = {}
VAR sA93 = {}
VAR sS1 = { stA1 }
VAR sS2 = {}
VAR sS3 = {}
VAR sS4 = {}
VAR sS5 = {}
VAR sS6 = { stA2, stA4 }
VAR sS7 = {}
VAR sS8 = {}
VAR sS9 = { stA6 }
VAR sS10 = {}
VAR sS11 = {}
VAR sS12 = {}
VAR sS13 = {}
VAR sS14 = {}
VAR sS15 = {}
VAR sS1opt = { stA1 }
VAR sS4opt = { stA2, stA4 }
VAR sS7opt = {}
VAR sS11opt = {}

— function subcomp maps component ID to the set of its subcomponents

fun subcomp :: CSet ⇒ CSet set

where
  subcomp sA1 = { sA11, sA12 }
  subcomp sA2 = { sA21, sA22, sA23 }
subcomp sA3 = \{ sA31, sA32 \}
subcomp sA4 = \{ sA41, sA42 \}
subcomp sA5 = {}
subcomp sA6 = {}
subcomp sA7 = \{ sA71, sA72 \}
subcomp sA8 = \{ sA81, sA82 \}
subcomp sA9 = \{ sA91, sA92, sA93 \}
subcomp sA11 = {}
subcomp sA12 = {}
subcomp sA21 = {}
subcomp sA22 = {}
subcomp sA23 = {}
subcomp sA31 = {}
subcomp sA32 = {}
subcomp sA41 = {}
subcomp sA42 = {}
subcomp sA71 = {}
subcomp sA72 = {}
subcomp sA81 = {}
subcomp sA82 = {}
subcomp sA91 = {}
subcomp sA92 = {}
subcomp sA93 = {}
subcomp sS1 = \{ sA12 \}
subcomp sS2 = \{ sA11 \}
subcomp sS3 = \{ sA21 \}
subcomp sS4 = \{ sA23 \}
subcomp sS5 = \{ sA32 \}
subcomp sS6 = \{ sA22, sA31, sA41 \}
subcomp sS7 = \{ sA42 \}
subcomp sS8 = \{ sA5 \}
subcomp sS9 = \{ sA6 \}
subcomp sS10 = \{ sA71 \}
subcomp sS11 = \{ sA72 \}
subcomp sS12 = \{ sA81, sA91 \}
subcomp sS13 = \{ sA92 \}
subcomp sS14 = \{ sA82 \}
subcomp sS15 = \{ sA93 \}
subcomp sS1opt = \{ sA11, sA12 \}
subcomp sS4opt = \{ sA22, sA23, sA31, sA32, sA41 \}
subcomp sS7opt = \{ sA42, sA5 \}
subcomp sS11opt = \{ sA72, sA82, sA93 \}

<table>
<thead>
<tr>
<th>AbstrLevel</th>
<th>maps abstraction level ID to the corresponding set of components</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Axiomatization</strong></td>
<td></td>
</tr>
<tr>
<td>AbstrLevel :: AbstrLevelsID =&gt; CSet set</td>
<td></td>
</tr>
<tr>
<td><strong>where</strong></td>
<td></td>
</tr>
<tr>
<td>AbstrLevel0:</td>
<td></td>
</tr>
<tr>
<td>AbstrLevel level0 = { sA1, sA2, sA3, sA4, sA5, sA6, sA7, sA8, sA9 }</td>
<td></td>
</tr>
</tbody>
</table>
and
AbstrLevel1:
\( \text{AbstrLevel level1} = \{ sA11, sA12, sA21, sA22, sA31, sA32, sA41, sA42, sA5, sA6, sA71, sA72, sA81, sA82, sA91, sA92, sA93 \} \)

and
AbstrLevel2:
\( \text{AbstrLevel level2} = \{ sS1, sS2, sS3, sS4, sS5, sS6, sS7, sS8, sS9, sS10, sS11, sS12, sS13, sS14, sS15 \} \)

and
AbstrLevel3:
\( \text{AbstrLevel level3} = \{ sS1opt, sS3, sS4opt, sS7opt, sS9, sS10, sS11opt, sS12, sS13 \} \)

— function \( \text{VARfrom} \) maps variable ID to the set of input channels it depends from
\[
\text{fun } \text{VARfrom} :: \text{varID } \Rightarrow \text{ chanID set}
\]
\[
\text{where}
\begin{align*}
\text{VARfrom } sA1 &= \{ \text{data1} \} \\
\text{VARfrom } sA2 &= \{ \text{data3} \} \\
\text{VARfrom } sA4 &= \{ \text{data6, data7} \} \\
\text{VARfrom } sA6 &= \{ \text{data14} \}
\end{align*}
\]

— function \( \text{VARto} \) maps variable ID to the set of output channels depending from this variable
\[
\text{fun } \text{VARto} :: \text{varID } \Rightarrow \text{ chanID set}
\]
\[
\text{where}
\begin{align*}
\text{VARto } sA1 &= \{ \text{data10} \} \\
\text{VARto } sA2 &= \{ \text{data4, data12} \} \\
\text{VARto } sA4 &= \{ \text{data3} \} \\
\text{VARto } sA6 &= \{ \text{data15, data16} \}
\end{align*}
\]

— function \( \text{OUTfromCh} \) maps channel ID to the set of input channels
— from which it depends directly;
— an empty set means that the channel is either input of the system or
— its values are computed from local variables or are generated
— within some component independently
\[
\text{fun } \text{OUTfromCh} :: \text{chanID } \Rightarrow \text{ chanID set}
\]
\[
\text{where}
\begin{align*}
\text{OUTfromCh } \text{data1} &= \{} \\
\text{OUTfromCh } \text{data2} &= \{ \text{data1} \} \\
\text{OUTfromCh } \text{data3} &= \{} \\
\text{OUTfromCh } \text{data4} &= \{ \text{data2} \} \\
\text{OUTfromCh } \text{data5} &= \{ \text{data2} \} \\
\text{OUTfromCh } \text{data6} &= \{ \text{data4} \} \\
\text{OUTfromCh } \text{data7} &= \{ \text{data5} \} \\
\text{OUTfromCh } \text{data8} &= \{ \text{data13} \} \\
\text{OUTfromCh } \text{data9} &= \{ \text{data8} \} \\
\text{OUTfromCh } \text{data10} &= \{} \\
\text{OUTfromCh } \text{data11} &= \{ \text{data2} \} \\
\text{OUTfromCh } \text{data12} &= \{}
\end{align*}
\]
| OUTfromCh data13 = {} |
| OUTfromCh data14 = {} |
| OUTfromCh data15 = {} |
| OUTfromCh data16 = {} |
| OUTfromCh data17 = {data15} |
| OUTfromCh data18 = {data16} |
| OUTfromCh data19 = {} |
| OUTfromCh data20 = {data17, data22} |
| OUTfromCh data21 = {data18, data19} |
| OUTfromCh data22 = {data20} |
| OUTfromCh data23 = {data21} |
| OUTfromCh data24 = {data20} |

— function OUTfromV maps channel ID to the set of local variables it depends from

fun OUTfromV :: chanID ⇒ varID set

where

| OUTfromV data1 = {} |
| OUTfromV data2 = {} |
| OUTfromV data3 = {stA4} |
| OUTfromV data4 = {stA2} |
| OUTfromV data5 = {} |
| OUTfromV data6 = {} |
| OUTfromV data7 = {} |
| OUTfromV data8 = {} |
| OUTfromV data9 = {} |
| OUTfromV data10 = {stA1} |
| OUTfromV data11 = {} |
| OUTfromV data12 = {stA2} |
| OUTfromV data13 = {} |
| OUTfromV data14 = {} |
| OUTfromV data15 = {stA6} |
| OUTfromV data16 = {stA6} |
| OUTfromV data17 = {} |
| OUTfromV data18 = {} |
| OUTfromV data19 = {} |
| OUTfromV data20 = {} |
| OUTfromV data21 = {} |
| OUTfromV data22 = {} |
| OUTfromV data23 = {} |
| OUTfromV data24 = {} |

— Set of channels channels which have UplSize measure greather that the predifined value HighLoad

definition

UplSizeHighLoad :: chanID set

where

UplSizeHighLoad ≡ {data1, data4, data5, data6, data7, data8, data18, data21}
— Set of components from the abstraction level 1 for which the Perf measure is greater than the predefined value $HighPerf$

**definition**

$HighPerfSet :: CSet$ set

**where**

$HighPerfSet \equiv \{ sA22, sA23, sA41, sA42, sA72, sA93 \}$

end

3 Inter-/Intracomponent dependencies

**theory** DataDependencies

**imports** DataDependenciesConcreteValues

**begin**

— component and its subcomponents should be defined on different abstraction levels

**definition**

$correctCompositionDiffLevels :: CSet \Rightarrow bool$

**where**

$correctCompositionDiffLevels S \equiv$

$\forall C \in \text{subcomp } S. \forall i. S \in \text{AbstrLevel } i \rightarrow C \notin \text{AbstrLevel } i$

— General system’s property: for all abstraction levels and all components should hold

— component and its subcomponents should be defined on different abstraction levels

**definition**

$correctCompositionDiffLevelsSYSTEM :: bool$

**where**

$correctCompositionDiffLevelsSYSTEM \equiv$

$(\forall S :: CSet. (correctCompositionDiffLevels S))$

— if a local variable belongs to one of the subcomponents, it also belongs to the composed component

**definition**

$correctCompositionVAR :: CSet \Rightarrow bool$

**where**

$correctCompositionVAR S \equiv$

$\forall C \in \text{subcomp } S. \forall v \in \text{VAR } C. v \in \text{VAR } S$

— General system’s property: for all abstraction levels and all components should hold

— if a local variable belongs to one of the subcomponents, it also belongs to the composed component

**definition**

$correctCompositionVARSYSTEM :: bool$

**where**

$correctCompositionVARSYSTEM \equiv$
∀ S::CSet. (correctCompositionVAR S)

— after correct decomposition of a component each of its local variable can belong
only to one of its subcomponents

**definition**
correctDeCompositionVAR :: CSet ⇒ bool

where

correctDeCompositionVAR S ≡
∀ v ∈ VAR S. ∀ C1 ∈ subcomp S. ∀ C2 ∈ subcomp S. v ∈ VAR C1 ∧ v ∈ VAR C2 → C1 = C2

— General system’s property: for all abstraction levels and all components should hold

— after correct decomposition of a component each of its local variable can belong
only to one of its subcomponents

**definition**
correctDeCompositionVARSYSTEM :: bool

where

correctDeCompositionVARSYSTEM ≡
(∀ S::CSet. (correctDeCompositionVAR S))

— if x is an output channel of a component C on some anstraction level, it cannot
be an output of another component on the same level

**definition**
correctCompositionOUT :: chanID ⇒ bool

where

correctCompositionOUT x ≡
∀ C i. x ∈ OUT C ∧ C ∈ AbstrLevel i → (∀ S ∈ AbstrLevel i. x \notin OUT S)

— General system’s property: for all abstraction levels and all channels should hold

**definition**
correctCompositionOUTSYSTEM :: bool

where

correctCompositionOUTSYSTEM ≡ (∀ x. correctCompositionOUT x)

— if X is a subcomponent of a component C on some anstraction level, it cannot
be a subcomponent of another component on the same level

**definition**
correctCompositionSubcomp :: CSet ⇒ bool

where

correctCompositionSubcomp X ≡
∀ C i. X ∈ subcomp C ∧ C ∈ AbstrLevel i → (∀ S ∈ AbstrLevel i. (S \neq C → X \notin subcomp S))

— General system’s property: for all abstraction levels and all components should hold

**definition**
correctCompositionSubcompSYSTEM :: bool

where
correctCompositionSubcompSYSTEM \equiv (\forall \ X. \ correctCompositionSubcomp X)

— If a component belongs is defined in the set CSet, it should belong to at least one abstraction level

**definition**

allComponentsUsed :: bool

**where**

allComponentsUsed \equiv \forall \ C. \ \exists \ i. \ C \in \text{AbstrLevel} \ i

— if a component does not have any local variables, none of its subcomponents has any local variables

**lemma** correctDeCompositionVARempty:

**assumes** correctCompositionVAR \ S

**and** \ VAR \ S = {}  

**shows** \ \forall \ C \in \text{subcomp} \ S. \ VAR \ C = {}

**proof**

**definition** OUTfrom :: chanID \Rightarrow \text{chanID set}

**where**

OUTfrom \ x \equiv (OUTfromCh \ x) \cup \{y. \ \exists \ v. \ v \in (OUTfromV \ x) \land y \in (VARfrom \ v)\}

— if \ x \ depends \ from \ some \ input \ channel(s) \ directly, \ then \ exists

— a component which has them as input channels and \ x \ as an output channel

**definition**

OUTfromChCorrect :: chanID \Rightarrow \text{bool}

**where**

OUTfromChCorrect \ x \equiv

(OUTfromCh \ x \neq \{} \rightarrow

(\exists \ Z. \ (x \in (OUT \ Z) \land (\forall \ y \in (OUTfromCh \ x). \ y \in IN \ Z)))

— General system’s property: for channels in the system should hold:

— if \ x \ depends \ from \ some \ input \ channel(s) \ directly, \ then \ exists

— a component which has them as input channels and \ x \ as an output channel

**definition**

OUTfromChCorrectSYSTEM :: \text{bool}

**where**

OUTfromChCorrectSYSTEM \equiv (\forall \ x::\text{chanID}. \ (OUTfromChCorrect \ x))

— if \ x \ depends \ from \ some \ local \ variables, \ then \ exists \ a \ component

— to which \ these \ variables \ belong \ and \ which \ has \ x \ as \ an \ output \ channel

**definition**

OUTfromVCorrect1 :: \text{chanID} \Rightarrow \text{bool}

**where**

OUTfromVCorrect1 \ x \equiv

(OUTfromV \ x \neq \{} \rightarrow

(\exists \ Z. \ (x \in (OUT \ Z) \land (\forall \ v \in (OUTfromV \ x). \ v \in VAR \ Z)))

— General system’s property: for channels in the system should hold the above
property:

**definition**

\[ \text{OUTfromVCorrect1SYSTEM} :: \text{bool} \]

**where**

\[ \text{OUTfromVCorrect1SYSTEM} \equiv (\forall x::\text{chanID}. \ (\text{OUTfromVCorrect1} \ x)) \]

— if \( x \) does not depend from any local variables, then it does not belong to any set

**definition**

\[ \text{OUTfromVCorrect2} :: \text{chanID} \Rightarrow \text{bool} \]

**where**

\[ \text{OUTfromVCorrect2} \ x \equiv (\text{OUTfromV} \ x = \{\} \rightarrow (\forall v::\text{varID}. \ x \notin (\text{VARto} \ v))) \]

— General system’s property: for channels in the system should hold the above property:

**definition**

\[ \text{OUTfromVCorrect2SYSTEM} :: \text{bool} \]

**where**

\[ \text{OUTfromVCorrect2SYSTEM} \equiv (\forall x::\text{chanID}. \ (\text{OUTfromVCorrect2} \ x)) \]

— General system’s property:
— definitions OUTfromV and VARto should give equivalent mappings

**definition**

\[ \text{OUTfromV-VARto} :: \text{bool} \]

**where**

\[ \text{OUTfromV-VARto} \equiv (\forall x::\text{chanID}. \ (\forall v::\text{varID}. \ (v \in \text{OUTfromV} \ x \iff x \in (\text{VARto} \ v)))) \]

— General system’s property for abstraction levels 0 and 1
— if a variable \( v \) belongs to a component, then all the channels \( v \)
— depends from should be input channels of this component

**definition**

\[ \text{VARfromCorrectSYSTEM} :: \text{bool} \]

**where**

\[ \text{VARfromCorrectSYSTEM} \equiv (\forall v::\text{varID}. \ (\forall Z \in ((\text{AbstrLevel level0}) \cup (\text{AbstrLevel level1})). \ (v \in \text{VAR} \ Z) \rightarrow (\forall x \in \text{VARfrom} \ v. \ x \in \text{IN} \ Z))) \]

— General system’s property for abstraction levels 0 and 1
— if a variable \( v \) belongs to a component, then all the channels \( v \)
— provides value to should be input channels of this component

**definition**

\[ \text{VARtoCorrectSYSTEM} :: \text{bool} \]

**where**

\[ \text{VARtoCorrectSYSTEM} \equiv (\forall v::\text{varID}. \ (\forall Z \in ((\text{AbstrLevel level0}) \cup (\text{AbstrLevel level1})). \ (v \in \text{VAR} \ Z) \rightarrow (\forall x \in \text{VARto} \ v. \ x \in \text{OUT} \ Z))) \]
— to detect local variables, unused for computation of any output

definition
VARusefulSYSTEM :: bool
where
VARusefulSYSTEM ≡ (∀ v::varID. (VARto v ≠ {}))

lemma
OUTfromV-VARto-lemma:
assumes OUTfromV x ≠ {} and OUTfromV-VARto
shows ∃ v::varID. x ∈ (VARto v)
⟨proof⟩

3.1 Direct and indirect data dependencies between components

— The component C should be defined on the same abstraction
— level we are searching for its direct or indirect sources,
— otherwise we get an empty set as result

definition
DSources :: AbstrLevelsID ⇒ CSet ⇒ CSet set
where
DSources i C ≡ {Z. ∃ x. x ∈ (IN C) ∧ x ∈ (OUT Z) ∧ Z ∈ (AbstrLevel i) ∧ C ∈ (AbstrLevel i)}

lemma DSourcesLevelX:
(DSources i X) ⊆ (AbstrLevel i)
⟨proof⟩

definition
DAcc :: AbstrLevelsID ⇒ CSet ⇒ CSet set
where
DAcc i C ≡ {Z. ∃ x. x ∈ (OUT C) ∧ x ∈ (IN Z) ∧ Z ∈ (AbstrLevel i) ∧ C ∈ (AbstrLevel i)}

axiomatization
Sources :: AbstrLevelsID ⇒ CSet ⇒ CSet set
where
SourcesDef:
(Sources i C) = (DSources i C) ∪ (⋃ S ∈ (DSources i C). (Sources i S))
and
SourceExistsDSources:
S ∈ (Sources i C) −→ (∃ Z. S ∈ (DSources i Z))
and
NDSourceExistsDSources:
S ∈ (Sources i C) ∧ S ∉ (DSources i C) −→
(∃ Z. S ∈ (DSources i Z) ∧ Z ∈ (Sources i C))
and
SourcesTrans:
(C ∈ Sources i S ∧ S ∈ Sources i Z) −→ C ∈ Sources i Z
and
SourcesLevelX:
\[(Sources \ i \ X) \subseteq (AbstrLevel \ i)\]
and
SourcesLoop:
\[(Sources \ i \ C) = (XS \cup (Sources \ i \ S)) \land (Sources \ i \ S) = (ZS \cup (Sources \ i \ C))\]
\[
\rightarrow (Sources \ i \ C) = XS \cup ZS \cup \{ \ C, S \}
\]
— if we have a loop in the dependencies we need to cut it for counting the sources

axiomatization
Acc :: AbstrLevelsID \Rightarrow \text{CSet} \Rightarrow \text{CSet set}
where
AccDef:
\[(Acc \ i \ C) = (DAcc \ i \ C) \cup (\bigcup S \in (DAcc \ i \ C). (Acc \ i \ S))\]
and
Acc-Sources:
\[(X \in Acc \ i \ C) = (C \in Sources \ i \ X)\]
and
AccSigeLoop:
\[DAcc \ i \ C = \{ S \} \land DAcc \ i \ S = \{ C \} \rightarrow Acc \ i \ C = \{ C, S \}\]
and
AccLoop:
\[(Acc \ i \ C) = (XS \cup (Acc \ i \ S)) \land (Acc \ i \ S) = (ZS \cup (Acc \ i \ C))\]
\[
\rightarrow (Acc \ i \ C) = XS \cup ZS \cup \{ \ C, S \}
\]
— if we have a loop in the dependencies we need to cut it for counting the accessors

lemma Acc-SourcesNOT: \((X \notin Acc \ i \ C) = (C \notin Sources \ i \ X)\)
⟨proof⟩
definition
isNotDSource :: AbstrLevelsID \Rightarrow \text{CSet} \Rightarrow \text{bool}
where
\[isNotDSource \ i \ S \equiv (\forall x \in (OUT S). (\forall Z \in (AbstrLevel \ i). (x \notin (IN Z))))\]
— component \( S \) is not a source for a component \( Z \) on the abstraction level \( i \)
definition
isNotDSourceX :: AbstrLevelsID \Rightarrow \text{CSet} \Rightarrow \text{CSet} \Rightarrow \text{bool}
where
\[isNotDSourceX \ i \ S \ C \equiv (\forall x \in (OUT S). (C \notin (AbstrLevel \ i) \lor (x \notin (IN C))))\]

lemma isNotSource-isNotSourceX:
isNotDSource \ i \ S \ C \equiv (\forall \ C. isNotDSourceX \ i \ S \ C)
⟨proof⟩

lemma DAcc-DSources:
\[(X \in DAcc \ i \ C) = (C \in DSources \ i \ X)\]
⟨proof⟩

lemma DAcc-DSourcesNOT:
\[(X \notin DAcc \ i \ C) = (C \notin DSources \ i \ X)\]
⟨proof⟩
lemma DSource-level:
  assumes $S \in (\text{DSources } i \ C)$
  shows $C \in (\text{AbstrLevel } i)$
  ⟨proof⟩

lemma SourceExistsDSource-level:
  assumes $S \in (\text{Sources } i \ C)$
  shows $\exists Z \in (\text{AbstrLevel } i). (S \in (\text{DSources } i \ Z))$
  ⟨proof⟩

lemma Sources-DSources:
  $(\text{DSources } i \ C) \subseteq (\text{Sources } i \ C)$
  ⟨proof⟩

lemma NoDSourceNoSource:
  assumes $S \notin (\text{Sources } i \ C)$
  shows $S \notin (\text{DSources } i \ C)$
  ⟨proof⟩

lemma DSourcesEmptySources:
  assumes $\text{DSources } i \ C = \{\}$
  shows $\text{Sources } i \ C = \{\}$
  ⟨proof⟩

lemma DSource-Sources:
  assumes $S \in (\text{DSources } i \ C)$
  shows $(\text{Sources } i \ S) \subseteq (\text{Sources } i \ C)$
  ⟨proof⟩

lemma SourcesOnlyDSources:
  assumes $\forall X. (X \in (\text{DSources } i \ C) \rightarrow (\text{DSources } i \ X) = \{\})$
  shows $\text{Sources } i \ C = \text{DSources } i \ C$
  ⟨proof⟩

lemma SourcesEmptyDSources:
  assumes $\text{Sources } i \ C = \{\}$
  shows $\text{DSources } i \ C = \{\}$
  ⟨proof⟩

lemma NotDSource:
  assumes $\forall x \in (\text{OUT } S). (\forall Z \in (\text{AbstrLevel } i). (x \notin (\text{IN } Z)))$
  shows $\forall C \in (\text{AbstrLevel } i). S \notin (\text{DSources } i \ C)$
  ⟨proof⟩

lemma allNotDSource-NotSource:
  assumes $\forall C . S \notin (\text{DSources } i \ C)$
  shows $\forall Z. S \notin (\text{Sources } i \ Z)$
  ⟨proof⟩
lemma NotDSource-NotSource:
assumes ∀ C ∈ (AbstrLevel i). S ∉ (DSources i C)
shows ∀ Z ∈ (AbstrLevel i). S ∉ (Sources i Z)
⟨proof⟩

lemma isNotSource-Sources:
assumes isNotDSource i S
shows ∀ C ∈ (AbstrLevel i). S ∉ (Sources i C)
⟨proof⟩

lemma SourcesAbstrLevel:
assumes x ∈ Sources i S
shows x ∈ AbstrLevel i
⟨proof⟩

lemma DSourceIsSource:
assumes C ∈ DSources i S
shows C ∈ Sources i S
⟨proof⟩

lemma DSourceOfDSource:
assumes Z ∈ DSources i S
and S ∈ DSources i C
shows Z ∈ Sources i C
⟨proof⟩

lemma SourceOfDSource:
assumes Z ∈ Sources i S
and S ∈ DSources i C
shows Z ∈ Sources i C
⟨proof⟩

lemma DSourceOfSource:
assumes Z ∈ Sources i S
and S ∈ DSources i C
shows C ∈ Sources i Z
⟨proof⟩

lemma Sources-singleDSource:
assumes DSources i S = {C}
shows Sources i S = {C} ∪ Sources i C
⟨proof⟩

lemma Sources-2DSources:
assumes DSources i S = {C1, C2}
shows Sources i S = {C1, C2} ∪ Sources i C1 ∪ Sources i C2
⟨proof⟩
lemma Sources-3DSources:
assumes DSources i S = \{C_1, C_2, C_3\}
shows Sources i S = \{C_1, C_2, C_3\} \cup Sources i C_1 \cup Sources i C_2 \cup Sources i C_3
⟨proof⟩

lemma singleDSourceEmpty4isNotDSource:
assumes DAcc i C = \{S\}
and Z \neq S
shows C \notin (DSources i Z)
⟨proof⟩

lemma singleDSourceEmpty4isNotDSourceLevel:
assumes DAcc i C = \{S\}
shows \forall Z \in (AbstrLevel i). Z \neq S \rightarrow C \notin (DSources i Z)
⟨proof⟩

lemma isNotDSource-EmptyDAcc:
assumes isNotDSource i S
shows DAcc i S = \{
⟨proof⟩

lemma isNotDSource-EmptyAcc:
assumes isNotDSource i S
shows Acc i S = \{
⟨proof⟩

lemma singleDSourceEmpty-Acc:
assumes DAcc i C = \{S\}
and isNotDSource i S
shows Acc i C = \{S\}
⟨proof⟩

lemma singleDSourceEmpty4isNotSource:
assumes DAcc i C = \{S\}
and nSourcS:isNotDSource i S
and Z \neq S
shows C \notin (Sources i Z)
⟨proof⟩

lemma singleDSourceEmpty4isNotSourceLevel:
assumes DAcc i C = \{S\}
and nSourcS:isNotDSource i S
shows \forall Z \in (AbstrLevel i). Z \neq S \rightarrow C \notin (Sources i Z)
⟨proof⟩

lemma singleDSourceLoop:
assumes $\text{DAcc~i~} C = \{S\}$
and $\text{DAcc~i~} S = \{C\}$
sows $\forall Z \in (\text{AbstrLevel ~i}), (Z \neq S \land Z \neq C \rightarrow C \notin (\text{Sources ~i~} Z))$
(proof)

3.2 Components that are elementary wrt. data dependencies

— two output channels of a component $C$ are corelated, if they mutually depend on the same local variable(s)
definition
outPairCorelated :: $\text{CSet} \Rightarrow \text{chanID} \Rightarrow \text{chanID} \Rightarrow \text{bool}$
where
outPairCorelated $C$ $x$ $y$ ≡
($x \in \text{OUT } C$) $\land$ ($y \in \text{OUT } C$) $\land$
($\text{OUTfromV } x \cap \text{OUTfromV } y \neq \{\}$)

— We call a set of output channels of a component correlated to it output channel $x$,
— if they mutually depend on the same local variable(s)
definition
outSetCorelated :: $\text{chanID} \Rightarrow \text{chanID set}$
where
outSetCorelated $x$ ≡
{ $y$::$\text{chanID}$ . $\exists v$::$\text{varID}$. ($v \in \text{OUTfromV } x$) $\land$ ($y \in \text{VARto } v$) }

— Elementary component according to the data dependencies.
— This constraint should hold for all components on the abstraction level 1
definition
elementaryCompDD :: $\text{CSet} \Rightarrow \text{bool}$
where
elementaryCompDD $C$ ≡
((∃ $x$. (OUT $C$) = $\{x\}$ ) $\lor$
($\forall x \in (\text{OUT } C). \forall y \in (\text{OUT } C). ((\text{outSetCorelated } x) \cap (\text{outSetCorelated } y)$
$\neq \{\}) )

— the set (outSetCorelated $x$) is empty if $x$ does not depend from any variable
lemma outSetCorelatedEmpty1:
assumes $\text{OUTfromV } x = \{\}$
sows outSetCorelated $x = \{\}$
(proof)

lemma outSetCorelatedNonemptyX:
assumes $\text{OUTfromV } x \neq \{\}$ and correct3:$\text{OUTfromV-VARto}$
sows $x \in \text{outSetCorelated } x$
(proof)

lemma outSetCorelatedEmpty2:
assumes outSetCorelated $x = \{\}$ and correct3:$\text{OUTfromV-VARto}$
sows $\text{OUTfromV } x = \{\}$
(proof)
3.3 Set of components needed to check a specific property

— set of components specified on abstraction level i, which input channels belong to the set chSet

**definition**
\[\text{inSetOfComponents} :: \text{AbstrLevelsID} \Rightarrow \text{chanID set} \Rightarrow \text{CSet set}\]
**where**
\[\text{inSetOfComponents} i \text{ chSet} \equiv \{\ Y. (((\text{OUT} Y) \cap \text{chSet} \neq \{\}) \land Y \in (\text{AbstrLevel i}))\}\]

— Set of components from the abstraction level i, which output channels belong to the set chSet

**definition**
\[\text{outSetOfComponents} :: \text{AbstrLevelsID} \Rightarrow \text{chanID set} \Rightarrow \text{CSet set}\]
**where**
\[\text{outSetOfComponents} i \text{ chSet} \equiv \{\ X. (((\text{IN} X) \cap \text{chSet} \neq \{\}) \land X \in (\text{AbstrLevel i}))\}\]

— Set of components from the abstraction level i, which have output channels from the set chSet or are sources for such components

**definition**
\[\text{minSetOfComponents} :: \text{AbstrLevelsID} \Rightarrow \text{chanID set} \Rightarrow \text{CSet set}\]
**where**
\[\text{minSetOfComponents} i \text{ chSet} \equiv (\text{outSetOfComponents} i \text{ chSet}) \cup (\bigcup S \in (\text{outSetOfComponents} i \text{ chSet}). (\text{Sources} i S))\]

— Please note that a system output cannot beat the same time a local channel.

— channel x is a system input on an abstraction level i

**definition** \[\text{systemIN} :: \text{chanID} \Rightarrow \text{AbstrLevelsID} \Rightarrow \text{bool}\]
**where**
\[\text{systemIN} x \ i \equiv (\exists \ C1 \in (\text{AbstrLevel i}). x \in (\text{IN} C1)) \land (\forall \ C2 \in (\text{AbstrLevel i}). x \notin (\text{OUT} C2))\]

— channel x is a system input on an abstraction level i

**definition** \[\text{systemOUT} :: \text{chanID} \Rightarrow \text{AbstrLevelsID} \Rightarrow \text{bool}\]
**where**
\[\text{systemOUT} x \ i \equiv (\forall \ C1 \in (\text{AbstrLevel i}). x \notin (\text{IN} C1)) \land (\exists \ C2 \in (\text{AbstrLevel i}). x \in (\text{OUT} C2))\]

— channel x is a system local channel on an abstraction level i

**definition** \[\text{systemLOC} :: \text{chanID} \Rightarrow \text{AbstrLevelsID} \Rightarrow \text{bool}\]
**where**
\[\text{systemLOC} x \ i \equiv (\exists \ C1 \in (\text{AbstrLevel i}). x \in (\text{IN} C1)) \land (\exists \ C2 \in (\text{AbstrLevel i}). x \in (\text{OUT} C2))\]

**lemma** \[\text{systemIN-noOUT}:\]
**assumes** \[\text{systemIN} x \ i\]
**shows** \[\neg \text{systemOUT} x \ i\]
\begin{proof}

\textbf{lemma systemOUT-noIN}:
\begin{itemize}
\item \textbf{assumes} \text{systemOUT } x \ i
\item \textbf{shows} \neg \text{systemIN } x \ i
\end{itemize}
\end{proof}

\textbf{lemma systemIN-noLOC}:
\begin{itemize}
\item \textbf{assumes} \text{systemIN } x \ i
\item \textbf{shows} \neg \text{systemLOC } x \ i
\end{itemize}
\end{proof}

\textbf{lemma systemLOC-noIN}:
\begin{itemize}
\item \textbf{assumes} \text{systemLOC } x \ i
\item \textbf{shows} \neg \text{systemIN } x \ i
\end{itemize}
\end{proof}

\textbf{lemma systemOUT-noLOC}:
\begin{itemize}
\item \textbf{assumes} \text{systemOUT } x \ i
\item \textbf{shows} \neg \text{systemLOC } x \ i
\end{itemize}
\end{proof}

\textbf{lemma systemLOC-noOUT}:
\begin{itemize}
\item \textbf{assumes} \text{systemLOC } x \ i
\item \textbf{shows} \neg \text{systemOUT } x \ i
\end{itemize}
\end{proof}

\textbf{definition}
\begin{itemize}
\item \text{noIrrelevantChannels} :: \text{AbstrLevelsID} \Rightarrow \text{chanID set} \Rightarrow \text{bool}
\item \textbf{where}
\item noIrrelevantChannels i chSet \equiv \forall x \in chSet. ((\text{systemIN } x \ i) \rightarrow (\exists Z \in (\text{minSetOfComponents } i \ chSet). x \in (IN Z)))
\end{itemize}

\textbf{definition}
\begin{itemize}
\item \text{allNeededINChannels} :: \text{AbstrLevelsID} \Rightarrow \text{chanID set} \Rightarrow \text{bool}
\item \textbf{where}
\item allNeededINChannels i chSet \equiv (\forall Z \in (\text{minSetOfComponents } i \ chSet). \exists x \in (IN Z). ((\text{systemIN } x \ i) \rightarrow (x \in chSet)))
\end{itemize}

— the set \text{(outSetOfComponents } i \ chSet) should be a subset of all components specified on the abstraction level \text{i}

\textbf{lemma outSetOfComponentsLimit}:
\text{outSetOfComponents } i \ chSet \subseteq \text{AbstrLevel } i
\end{proof}

\textbf{lemma inSetOfComponentsLimit}:
\text{inSetOfComponents } i \ chSet \subseteq \text{AbstrLevel } i
proof

lemma SourcesLevelLimit:
(⋃ S ∈ (outSetOfComponents i chSet). (Sources i S)) ⊆ AbstrLevel i

lemma minSetOfComponentsLimit:
minSetOfComponents i chSet ⊆ AbstrLevel i

3.4 Additional properties: Remote Computation
— The value of $UplSizeHighLoad x$ is True if its $UplSize$ measure is greater than a predefined value

definition $UplSizeHighLoadCh :: \text{chanID} \Rightarrow \text{bool}$
where
$UplSizeHighLoadCh x \equiv (x \in UplSizeHighLoad)$

— if the $Perf$ measure of at least one subcomponent is greater than a predefined value,
— the $Perf$ measure of this component is greater than $HighPerf$ too

axiomatization $HighPerfComp :: CSet \Rightarrow \text{bool}$
where
$HighPerfComDef :: HighPerfComp C = ((C \in HighPerfSet) \lor (\exists Z \in subcomp C. (HighPerfComp Z)))$

end

4 Case Study: Verification of Properties

theory DataDependenciesCaseStudy
import DataDependencies
begin

4.1 Correct composition of components
— the lemmas AbstrLevels X Y with corresponding proofs can be composed and proven automatically, their proofs are identical

lemma AbstrLevels-A1-A11:
assumes $sA1 \in AbstrLevel i$
shows $sA11 \notin AbstrLevel i$

lemma AbstrLevels-A1-A12:
assumes $sA1 \in AbstrLevel i$
shows $sA12 \notin AbstrLevel i$

lemma AbstrLevels-A2-A21:
assumes $sA2 \in AbstrLevel i$
\begin{verbatim}
shows \(sA21 \notin \text{AbstrLevel } i\)\{proof\}

lemma \text{AbstrLevels-A2-A22}:
assumes \(sA2 \in \text{AbstrLevel } i\)
shows \(sA22 \notin \text{AbstrLevel } i\)\{proof\}

lemma \text{AbstrLevels-A2-A23}:
assumes \(sA2 \in \text{AbstrLevel } i\)
shows \(sA23 \notin \text{AbstrLevel } i\)\{proof\}

lemma \text{AbstrLevels-A3-A31}:
assumes \(sA3 \in \text{AbstrLevel } i\)
shows \(sA31 \notin \text{AbstrLevel } i\)\{proof\}

lemma \text{AbstrLevels-A3-A32}:
assumes \(sA3 \in \text{AbstrLevel } i\)
shows \(sA32 \notin \text{AbstrLevel } i\)\{proof\}

lemma \text{AbstrLevels-A4-A41}:
assumes \(sA4 \in \text{AbstrLevel } i\)
shows \(sA41 \notin \text{AbstrLevel } i\)\{proof\}

lemma \text{AbstrLevels-A4-A42}:
assumes \(sA4 \in \text{AbstrLevel } i\)
shows \(sA42 \notin \text{AbstrLevel } i\)\{proof\}

lemma \text{AbstrLevels-A7-A71}:
assumes \(sA7 \in \text{AbstrLevel } i\)
shows \(sA71 \notin \text{AbstrLevel } i\)\{proof\}

lemma \text{AbstrLevels-A7-A72}:
assumes \(sA7 \in \text{AbstrLevel } i\)
shows \(sA72 \notin \text{AbstrLevel } i\)\{proof\}

lemma \text{AbstrLevels-A8-A81}:
assumes \(sA8 \in \text{AbstrLevel } i\)
shows \(sA81 \notin \text{AbstrLevel } i\)\{proof\}

lemma \text{AbstrLevels-A8-A82}:
assumes \(sA8 \in \text{AbstrLevel } i\)
shows \(sA82 \notin \text{AbstrLevel } i\)\{proof\}

lemma \text{AbstrLevels-A9-A91}:
assumes \(sA9 \in \text{AbstrLevel } i\)
shows \(sA91 \notin \text{AbstrLevel } i\)\{proof\}

lemma \text{AbstrLevels-A9-A92}:
assumes \(sA9 \in \text{AbstrLevel } i\)
shows \(sA92 \notin \text{AbstrLevel } i\)\{proof\}

lemma \text{AbstrLevels-A9-A93}:
\end{verbatim}
assumes $s_{A9} \in \text{AbstrLevel } i$
shows $s_{A93} \notin \text{AbstrLevel } i$\proof

lemma \text{AbstrLevels-S1-A12}:  
assumes $s_{S1} \in \text{AbstrLevel } i$
shows $s_{A12} \notin \text{AbstrLevel } i$\proof

lemma \text{AbstrLevels-S2-A11}:  
assumes $s_{S2} \in \text{AbstrLevel } i$
shows $s_{A11} \notin \text{AbstrLevel } i$\proof

lemma \text{AbstrLevels-S3-A21}:  
assumes $s_{S3} \in \text{AbstrLevel } i$
shows $s_{A21} \notin \text{AbstrLevel } i$\proof

lemma \text{AbstrLevels-S4-A23}:  
assumes $s_{S4} \in \text{AbstrLevel } i$
shows $s_{A23} \notin \text{AbstrLevel } i$\proof

lemma \text{AbstrLevels-S5-A32}:  
assumes $s_{S5} \in \text{AbstrLevel } i$
shows $s_{A32} \notin \text{AbstrLevel } i$\proof

lemma \text{AbstrLevels-S6-A22}:  
assumes $s_{S6} \in \text{AbstrLevel } i$
shows $s_{A22} \notin \text{AbstrLevel } i$\proof

lemma \text{AbstrLevels-S6-A31}:  
assumes $s_{S6} \in \text{AbstrLevel } i$
shows $s_{A31} \notin \text{AbstrLevel } i$\proof

lemma \text{AbstrLevels-S6-A41}:  
assumes $s_{S6} \in \text{AbstrLevel } i$
shows $s_{A41} \notin \text{AbstrLevel } i$\proof

lemma \text{AbstrLevels-S7-A42}:  
assumes $s_{S7} \in \text{AbstrLevel } i$
shows $s_{A42} \notin \text{AbstrLevel } i$\proof

lemma \text{AbstrLevels-S8-A5}:  
assumes $s_{S8} \in \text{AbstrLevel } i$
shows $s_{A5} \notin \text{AbstrLevel } i$\proof

lemma \text{AbstrLevels-S9-A6}:  
assumes $s_{S9} \in \text{AbstrLevel } i$
shows $s_{A6} \notin \text{AbstrLevel } i$\proof

lemma \text{AbstrLevels-S10-A71}:  
assumes $s_{S10} \in \text{AbstrLevel } i$
shows \( sA71 \notin \text{AbstrLevel} \langle \text{proof} \rangle \)

**lemma AbstrLevels-S11-A72:**
assumes \( sS11 \in \text{AbstrLevel} \langle \text{proof} \rangle \)
shows \( sA72 \notin \text{AbstrLevel} \langle \text{proof} \rangle \)

**lemma AbstrLevels-S12-A81:**
assumes \( sS12 \in \text{AbstrLevel} \langle \text{proof} \rangle \)
shows \( sA81 \notin \text{AbstrLevel} \langle \text{proof} \rangle \)

**lemma AbstrLevels-S12-A91:**
assumes \( sS12 \in \text{AbstrLevel} \langle \text{proof} \rangle \)
shows \( sA91 \notin \text{AbstrLevel} \langle \text{proof} \rangle \)

**lemma AbstrLevels-S13-A92:**
assumes \( sS13 \in \text{AbstrLevel} \langle \text{proof} \rangle \)
shows \( sA92 \notin \text{AbstrLevel} \langle \text{proof} \rangle \)

**lemma AbstrLevels-S14-A82:**
assumes \( sS14 \in \text{AbstrLevel} \langle \text{proof} \rangle \)
shows \( sA82 \notin \text{AbstrLevel} \langle \text{proof} \rangle \)

**lemma AbstrLevels-S15-A93:**
assumes \( sS15 \in \text{AbstrLevel} \langle \text{proof} \rangle \)
shows \( sA93 \notin \text{AbstrLevel} \langle \text{proof} \rangle \)

**lemma AbstrLevels-S1opt-A11:**
assumes \( sS1opt \in \text{AbstrLevel} \langle \text{proof} \rangle \)
shows \( sA11 \notin \text{AbstrLevel} \langle \text{proof} \rangle \)

**lemma AbstrLevels-S1opt-A12:**
assumes \( sS1opt \in \text{AbstrLevel} \langle \text{proof} \rangle \)
shows \( sA12 \notin \text{AbstrLevel} \langle \text{proof} \rangle \)

**lemma AbstrLevels-S4opt-A23:**
assumes \( sS4opt \in \text{AbstrLevel} \langle \text{proof} \rangle \)
shows \( sA23 \notin \text{AbstrLevel} \langle \text{proof} \rangle \)

**lemma AbstrLevels-S4opt-A32:**
assumes \( sS4opt \in \text{AbstrLevel} \langle \text{proof} \rangle \)
shows \( sA32 \notin \text{AbstrLevel} \langle \text{proof} \rangle \)

**lemma AbstrLevels-S4opt-A22:**
assumes \( sS4opt \in \text{AbstrLevel} \langle \text{proof} \rangle \)
shows \( sA22 \notin \text{AbstrLevel} \langle \text{proof} \rangle \)

**lemma AbstrLevels-S4opt-A31:**
assumes \( sS4opt \in \text{AbstrLevel} \langle \text{proof} \rangle \)
shows \( sA31 \notin \text{AbstrLevel} \langle \text{proof} \rangle \)
lemma AbstrLevels-S4opt-A41:
assumes sS4opt ∈ AbstrLevel i
shows sA41 ∉ AbstrLevel i ⟨proof⟩

lemma AbstrLevels-S7opt-A42:
assumes sS7opt ∈ AbstrLevel i
shows sA42 ∉ AbstrLevel i ⟨proof⟩

lemma AbstrLevels-S7opt-A5:
assumes sS7opt ∈ AbstrLevel i
shows sA5 ∉ AbstrLevel i ⟨proof⟩

lemma AbstrLevels-S11opt-A41:
assumes sS11opt ∈ AbstrLevel i
shows sA41 ∉ AbstrLevel i ⟨proof⟩

lemma AbstrLevels-S11opt-A42:
assumes sS11opt ∈ AbstrLevel i
shows sA42 ∉ AbstrLevel i ⟨proof⟩

lemma correctCompositionDiffLevelsA1: correctCompositionDiffLevels sA1 ⟨proof⟩

lemma correctCompositionDiffLevelsA2: correctCompositionDiffLevels sA2 ⟨proof⟩

lemma correctCompositionDiffLevelsA3: correctCompositionDiffLevels sA3 ⟨proof⟩

lemma correctCompositionDiffLevelsA4: correctCompositionDiffLevels sA4 ⟨proof⟩

lemma correctCompositionDiffLevelsA5: correctCompositionDiffLevels sA5 ⟨proof⟩
lemma correctCompositionDiffLevelsA6: correctCompositionDiffLevels sA6 ⟨proof⟩
lemma correctCompositionDiffLevelsA7: correctCompositionDiffLevels sA7 ⟨proof⟩
lemma correctCompositionDiffLevelsA8: correctCompositionDiffLevels sA8 ⟨proof⟩
lemma correctCompositionDiffLevelsA9: correctCompositionDiffLevels sA9 ⟨proof⟩
lemma correctCompositionDiffLevelsA11: correctCompositionDiffLevels sA11 ⟨proof⟩
lemma correctCompositionDiffLevelsA12: correctCompositionDiffLevels sA12 ⟨proof⟩
lemma correctCompositionDiffLevelsA21: correctCompositionDiffLevels sA21 ⟨proof⟩
lemma correctCompositionDiffLevelsA22: correctCompositionDiffLevels sA22 ⟨proof⟩
lemma correctCompositionDiffLevelsA23: correctCompositionDiffLevels sA23 ⟨proof⟩
lemma correctCompositionDiffLevelsA31: correctCompositionDiffLevels sA31 ⟨proof⟩
lemma correctCompositionDiffLevelsA32: correctCompositionDiffLevels sA32 ⟨proof⟩
lemma correctCompositionDiffLevelsA41: correctCompositionDiffLevels sA41 ⟨proof⟩
lemma correctCompositionDiffLevelsA42: correctCompositionDiffLevels sA42 ⟨proof⟩
lemma correctCompositionDiffLevelsA71: correctCompositionDiffLevels sA71 ⟨proof⟩
lemma correctCompositionDiffLevelsA72: correctCompositionDiffLevels sA72 ⟨proof⟩
4.2 Correct specification of the relations between channels

lemma OUTfromChCorrect-data1: OUTfromChCorrect data1 (proof)

lemma OUTfromChCorrect-data2: OUTfromChCorrect data2 (proof)

lemma OUTfromChCorrect-data3: OUTfromChCorrect data3 (proof)

lemma OUTfromChCorrect-data4: OUTfromChCorrect data4 (proof)

lemma OUTfromChCorrect-data5: OUTfromChCorrect data5 (proof)
lemma OUTfromChCorrect-data6: OUTfromChCorrect data6
  ⟨proof⟩

lemma OUTfromChCorrect-data7: OUTfromChCorrect data7
  ⟨proof⟩

lemma OUTfromChCorrect-data8: OUTfromChCorrect data8
  ⟨proof⟩

lemma OUTfromChCorrect-data9: OUTfromChCorrect data9
  ⟨proof⟩

lemma OUTfromChCorrect-data10: OUTfromChCorrect data10
  ⟨proof⟩

lemma OUTfromChCorrect-data11: OUTfromChCorrect data11
  ⟨proof⟩

lemma OUTfromChCorrect-data12: OUTfromChCorrect data12
  ⟨proof⟩

lemma OUTfromChCorrect-data13: OUTfromChCorrect data13
  ⟨proof⟩

lemma OUTfromChCorrect-data14: OUTfromChCorrect data14
  ⟨proof⟩

lemma OUTfromChCorrect-data15: OUTfromChCorrect data15
  ⟨proof⟩

lemma OUTfromChCorrect-data16: OUTfromChCorrect data16
  ⟨proof⟩

lemma OUTfromChCorrect-data17: OUTfromChCorrect data17
  ⟨proof⟩

lemma OUTfromChCorrect-data18: OUTfromChCorrect data18
  ⟨proof⟩

lemma OUTfromChCorrect-data19: OUTfromChCorrect data19
  ⟨proof⟩

lemma OUTfromChCorrect-data20: OUTfromChCorrect data20
  ⟨proof⟩

lemma OUTfromChCorrect-data21: OUTfromChCorrect data21
  ⟨proof⟩
lemma OUTfromChCorrect-data22: OUTfromChCorrect data22
(proof)

lemma OUTfromChCorrect-data23: OUTfromChCorrect data23
(proof)

lemma OUTfromChCorrect-data24: OUTfromChCorrect data24
(proof)

lemma OUTfromChCorrectSYSTEM-holds: OUTfromChCorrectSYSTEM
(proof)

lemma OUTfromVCorrect1-data1: OUTfromVCorrect1 data1
(proof)

lemma OUTfromVCorrect1-data2: OUTfromVCorrect1 data2
(proof)

lemma OUTfromVCorrect1-data3: OUTfromVCorrect1 data3
(proof)

lemma OUTfromVCorrect1-data4: OUTfromVCorrect1 data4
(proof)

lemma OUTfromVCorrect1-data5: OUTfromVCorrect1 data5
(proof)

lemma OUTfromVCorrect1-data6: OUTfromVCorrect1 data6
(proof)

lemma OUTfromVCorrect1-data7: OUTfromVCorrect1 data7
(proof)

lemma OUTfromVCorrect1-data8: OUTfromVCorrect1 data8
(proof)

lemma OUTfromVCorrect1-data9: OUTfromVCorrect1 data9
(proof)

lemma OUTfromVCorrect1-data10: OUTfromVCorrect1 data10
(proof)

lemma OUTfromVCorrect1-data11: OUTfromVCorrect1 data11
(proof)

lemma OUTfromVCorrect1-data12: OUTfromVCorrect1 data12
(proof)
lemma OUTfromVCorrect1-data13: OUTfromVCorrect1 data13  
(proof)

lemma OUTfromVCorrect1-data14: OUTfromVCorrect1 data14  
(proof)

lemma OUTfromVCorrect1-data15: OUTfromVCorrect1 data15  
(proof)

lemma OUTfromVCorrect1-data16: OUTfromVCorrect1 data16  
(proof)

lemma OUTfromVCorrect1-data17: OUTfromVCorrect1 data17  
(proof)

lemma OUTfromVCorrect1-data18: OUTfromVCorrect1 data18  
(proof)

lemma OUTfromVCorrect1-data19: OUTfromVCorrect1 data19  
(proof)

lemma OUTfromVCorrect1-data20: OUTfromVCorrect1 data20  
(proof)

lemma OUTfromVCorrect1-data21: OUTfromVCorrect1 data21  
(proof)

lemma OUTfromVCorrect1-data22: OUTfromVCorrect1 data22  
(proof)

lemma OUTfromVCorrect1-data23: OUTfromVCorrect1 data23  
(proof)

lemma OUTfromVCorrect1-data24: OUTfromVCorrect1 data24  
(proof)

lemma OUTfromVCorrect1SYSTEM-holds: OUTfromVCorrect1SYSTEM  
(proof)

lemma OUTfromV-VARto-holds:  
OUTfromV-VARto  
(proof)

lemma VARfromCorrectSYSTEM-holds:  
VARfromCorrectSYSTEM
lemma VARtoCorrectSYSTEM-holds: VARtoCorrectSYSTEM
⟨proof⟩

lemma VARusefulSYSTEM-holds: VARusefulSYSTEM
⟨proof⟩

4.3 Elementary components
— On the abstraction level 0 only the components sA5 and sA6 are elementary

lemma NOT-elementaryCompDD-sA1: ¬ elementaryCompDD sA1
⟨proof⟩

lemma NOT-elementaryCompDD-sA2: ¬ elementaryCompDD sA2
⟨proof⟩

lemma NOT-elementaryCompDD-sA3: ¬ elementaryCompDD sA3
⟨proof⟩

lemma NOT-elementaryCompDD-sA4: ¬ elementaryCompDD sA4
⟨proof⟩

lemma elementaryCompDD-sA5: elementaryCompDD sA5
⟨proof⟩

lemma elementaryCompDD-sA6: elementaryCompDD sA6
⟨proof⟩

lemma NOT-elementaryCompDD-sA7: ¬ elementaryCompDD sA7
⟨proof⟩

lemma NOT-elementaryCompDD-sA8: ¬ elementaryCompDD sA8
⟨proof⟩

lemma NOT-elementaryCompDD-sA9: ¬ elementaryCompDD sA9
⟨proof⟩

lemma elementaryCompDD-sA11: elementaryCompDD sA11
⟨proof⟩

lemma elementaryCompDD-sA12: elementaryCompDD sA12
⟨proof⟩

lemma elementaryCompDD-sA21: elementaryCompDD sA21
⟨proof⟩
lemma elementaryCompDD-sA22: elementaryCompDD sA22 ⟨proof⟩
lemma elementaryCompDD-sA23: elementaryCompDD sA23 ⟨proof⟩
lemma elementaryCompDD-sA31: elementaryCompDD sA31 ⟨proof⟩
lemma elementaryCompDD-sA32: elementaryCompDD sA32 ⟨proof⟩
lemma elementaryCompDD-sA41: elementaryCompDD sA41 ⟨proof⟩
lemma elementaryCompDD-sA42: elementaryCompDD sA42 ⟨proof⟩
lemma elementaryCompDD-sA71: elementaryCompDD sA71 ⟨proof⟩
lemma elementaryCompDD-sA72: elementaryCompDD sA72 ⟨proof⟩
lemma elementaryCompDD-sA81: elementaryCompDD sA81 ⟨proof⟩
lemma elementaryCompDD-sA82: elementaryCompDD sA82 ⟨proof⟩
lemma elementaryCompDD-sA91: elementaryCompDD sA91 ⟨proof⟩
lemma elementaryCompDD-sA92: elementaryCompDD sA92 ⟨proof⟩
lemma elementaryCompDD-sA93: elementaryCompDD sA93 ⟨proof⟩

4.4 Source components
— Abstraction level 0

lemma A5-NotDSource-level0: IsNotDSource level0 sA5 ⟨proof⟩
lemma DSourcesA1-L0: DSources level0 sA1 = {}
proof

lemma DSourcesA2-L0: DSources level0 sA2 = { sA1, sA4 }
(proof)

lemma DSourcesA3-L0: DSources level0 sA3 = { sA2 }
(proof)

lemma DSourcesA4-L0: DSources level0 sA4 = { sA3 }
(proof)

lemma DSourcesA5-L0: DSources level0 sA5 = { sA4 }
(proof)

lemma DSourcesA6-L0: DSources level0 sA6 = {}
(proof)

lemma DSourcesA7-L0: DSources level0 sA7 = {sA6}
(proof)

lemma DSourcesA8-L0: DSources level0 sA8 = {sA7, sA9}
(proof)

lemma DSourcesA9-L0: DSources level0 sA9 = {sA8}
(proof)

lemma A1-DAcc-level0: DAcc level0 sA1 = { sA2 }
(proof)

lemma A2-DAcc-level0: DAcc level0 sA2 = { sA3 }
(proof)

lemma A3-DAcc-level0: DAcc level0 sA3 = { sA4 }
(proof)

lemma A4-DAcc-level0: DAcc level0 sA4 = { sA2, sA5 }
(proof)

lemma A5-DAcc-level0: DAcc level0 sA5 = {}
(proof)

lemma A6-DAcc-level0: DAcc level0 sA6 = { sA7 }
(proof)

lemma A7-DAcc-level0: DAcc level0 sA7 = { sA8 }
(proof)

lemma A8-DAcc-level0: DAcc level0 sA8 = { sA9 }
(proof)
lemma A9-DAcc-level0: \( \text{DAcc level0 } sA9 = \{ sA8 \} \)

lemma A8-NSources:
\( \forall C \in (\text{AbstrLevel level0}). (C \neq sA9 \land C \neq sA8 \rightarrow sA8 \notin (\text{Sources level0 } C)) \)

lemma A9-NSources:
\( \forall C \in (\text{AbstrLevel level0}). (C \neq sA9 \land C \neq sA8 \rightarrow sA9 \notin (\text{Sources level0 } C)) \)

lemma A7-Acc:
\( \text{(Acc level0 } sA7) = \{ sA8, sA9 \} \)

lemma A7-NSources:
\( \forall C \in (\text{AbstrLevel level0}). (C \neq sA9 \land C \neq sA8 \rightarrow sA7 \notin (\text{Sources level0 } C)) \)

lemma A5-Acc: \( \text{(Acc level0 } sA5) = \{ \} \)

lemma A6-Acc:
\( \text{(Acc level0 } sA6) = \{ sA7, sA8, sA9 \} \)

lemma A6-NSources:
\( \forall C \in (\text{AbstrLevel level0}). (C \neq sA9 \land C \neq sA8 \land C \neq sA7 \rightarrow sA6 \notin (\text{Sources level0 } C)) \)

lemma SourcesA1-L0: \( \text{Sources level0 } sA1 = \{ \} \)

lemma SourcesA2-L0: \( \text{Sources level0 } sA2 = \{ sA1, sA2, sA3, sA4 \} \)

lemma SourcesA3-L0: \( \text{Sources level0 } sA3 = \{ sA1, sA2, sA3, sA4 \} \)

lemma SourcesA4-L0: \( \text{Sources level0 } sA4 = \{ sA1, sA2, sA3, sA4 \} \)

lemma SourcesA5-L0: \( \text{Sources level0 } sA5 = \{ sA1, sA2, sA3, sA4 \} \)

lemma SourcesA6-L0: \( \text{Sources level0 } sA6 = \{ \} \)
lemma SourcesA7-L0: Sources level0 sA7 = { sA6 }
(proof)

lemma SourcesA8-L0: Sources level0 sA8 = { sA6, sA7, sA8, sA9 }
(proof)

lemma SourcesA9-L0: Sources level0 sA9 = { sA6, sA7, sA8, sA9 }
(proof)

lemma A12-NotSource-level1: isNotDS source level1 sA12
(proof)

lemma A21-NotSource-level1: isNotDS source level1 sA21
(proof)

lemma A5-NotSource-level1: isNotDS source level1 sA5
(proof)

lemma A92-NotSource-level1: isNotDS source level1 sA92
(proof)

lemma A93-NotSource-level1: isNotDS source level1 sA93
(proof)

lemma A11-DAcc-level1: DAcc level1 sA11 = { sA21, sA22, sA23 }
(proof)

lemma A12-DAcc-level1: DAcc level1 sA12 = {}
(proof)

lemma A21-DAcc-level1: DAcc level1 sA21 = {}
(proof)

lemma A22-DAcc-level1: DAcc level1 sA22 = {sA31}
(proof)

lemma A25-DAcc-level1: DAcc level1 sA23 = {sA32}
(proof)

lemma A31-DAcc-level1: DAcc level1 sA31 = {sA41}
(proof)

lemma A32-DAcc-level1: DAcc level1 sA32 = {sA41}
(proof)

lemma A41-DAcc-level1: DAcc level1 sA41 = {sA22}
\begin{proof}

\textbf{lemma \ A42-DAcc-level1:} \ \text{DAcc level1} \ sA42 = \{sA5\}
\end{proof}

\begin{proof}

\textbf{lemma \ A5-DAcc-level1:} \ \text{DAcc level1} \ sA5 = \{\}
\end{proof}

\begin{proof}

\textbf{lemma \ A6-DAcc-level1:} \ \text{DAcc level1} \ sA6 = \{sA71, \ sA72\}
\end{proof}

\begin{proof}

\textbf{lemma \ A71-DAcc-level1:} \ \text{DAcc level1} \ sA71 = \{sA81\}
\end{proof}

\begin{proof}

\textbf{lemma \ A72-DAcc-level1:} \ \text{DAcc level1} \ sA72 = \{sA82\}
\end{proof}

\begin{proof}

\textbf{lemma \ A81-DAcc-level1:} \ \text{DAcc level1} \ sA81 = \{sA91, \ sA92\}
\end{proof}

\begin{proof}

\textbf{lemma \ A82-DAcc-level1:} \ \text{DAcc level1} \ sA82 = \{sA93\}
\end{proof}

\begin{proof}

\textbf{lemma \ A91-DAcc-level1:} \ \text{DAcc level1} \ sA91 = \{sA81\}
\end{proof}

\begin{proof}

\textbf{lemma \ A92-DAcc-level1:} \ \text{DAcc level1} \ sA92 = \{\}
\end{proof}

\begin{proof}

\textbf{lemma \ A93-DAcc-level1:} \ \text{DAcc level1} \ sA93 = \{\}
\end{proof}

\begin{proof}

\textbf{lemma \ A42-NSources-L1:}
\forall \ C \in (\text{AbstrLevel level1}). \ C \neq sA5 \rightarrow sA42 \notin (\text{Sources level1} \ C)
\end{proof}

\begin{proof}

\textbf{lemma \ A5-NotSourceSet-level1 :}
\forall \ C \in (\text{AbstrLevel level1}). \ sA5 \notin (\text{Sources level1} \ C)
\end{proof}

\begin{proof}

\textbf{lemma \ A92-NotSourceSet-level1 :}
\forall \ C \in (\text{AbstrLevel level1}). \ sA92 \notin (\text{Sources level1} \ C)
\end{proof}

\begin{proof}

\textbf{lemma \ A93-NotSourceSet-level1 :}
\forall \ C \in (\text{AbstrLevel level1}). \ sA93 \notin (\text{Sources level1} \ C)
\end{proof}
lemma $\text{DSourcesA11-L1}$: $\text{DSources level1 sA11} = \{\}$
(proof)

lemma $\text{DSourcesA12-L1}$: $\text{DSources level1 sA12} = \{\}$
(proof)

lemma $\text{DSourcesA21-L1}$: $\text{DSources level1 sA21} = \{sA11\}$
(proof)

lemma $\text{DSourcesA22-L1}$: $\text{DSources level1 sA22} = \{sA11, sA41\}$
(proof)

lemma $\text{DSourcesA23-L1}$: $\text{DSources level1 sA23} = \{sA11\}$
(proof)

lemma $\text{DSourcesA31-L1}$: $\text{DSources level1 sA31} = \{ sA22 \}$
(proof)

lemma $\text{DSourcesA32-L1}$: $\text{DSources level1 sA32} = \{ sA23 \}$
(proof)

lemma $\text{DSourcesA41-L1}$: $\text{DSources level1 sA41} = \{ sA31, sA32 \}$
(proof)

lemma $\text{DSourcesA42-L1}$: $\text{DSources level1 sA42} = \{\}$
(proof)

lemma $\text{DSourcesA5-L1}$: $\text{DSources level1 sA5} = \{ sA42 \}$
(proof)

lemma $\text{DSourcesA6-L1}$: $\text{DSources level1 sA6} = \{\}$
(proof)

lemma $\text{DSourcesA71-L1}$: $\text{DSources level1 sA71} = \{ sA6 \}$
(proof)

lemma $\text{DSourcesA72-L1}$: $\text{DSources level1 sA72} = \{ sA6 \}$
(proof)

lemma $\text{DSourcesA81-L1}$: $\text{DSources level1 sA81} = \{ sA71, sA91 \}$
(proof)

lemma $\text{DSourcesA82-L1}$: $\text{DSources level1 sA82} = \{ sA72 \}$
(proof)

lemma $\text{DSourcesA91-L1}$: $\text{DSources level1 sA91} = \{ sA81 \}$
(proof)

lemma $\text{DSourcesA92-L1}$: $\text{DSources level1 sA92} = \{ sA81 \}$
\[ \text{proof} \]

**Lemma DSourcesA93-L1**: \( \text{DSources level1 sA93} = \{ \text{sA82} \} \)

\[ \text{proof} \]

**Lemma A82-Acc**: \( \text{Acc level1 sA82} = \{ \text{sA93} \} \)

\[ \text{proof} \]

**Lemma A82-NSources-L1**: \( \forall C \in (\text{AbstrLevel level1}). (C \neq \text{sA93} \rightarrow \text{sA82} \notin \text{Sources level1 C}) \)

\[ \text{proof} \]

**Lemma A72-Acc**: \( \text{Acc level1 sA72} = \{ \text{sA82}, \text{sA93} \} \)

\[ \text{proof} \]

**Lemma A72-NSources-L1**: \( \forall C \in (\text{AbstrLevel level1}). (C \neq \text{sA93} \land C \neq \text{sA82} \rightarrow \text{sA72} \notin \text{Sources level1 C}) \)

\[ \text{proof} \]

**Lemma A92-Acc**: \( \text{Acc level1 sA92} = \{ \} \)

\[ \text{proof} \]

**Lemma A92-NSources-L1**: \( \forall C \in (\text{AbstrLevel level1}). (\text{sA92} \notin \text{Sources level1 C}) \)

\[ \text{proof} \]

**Lemma A91-Acc**: \( \text{Acc level1 sA91} = \{ \text{sA81}, \text{sA91}, \text{sA92} \} \)

\[ \text{proof} \]

**Lemma A91-NSources-L1**: \( \forall C \in (\text{AbstrLevel level1}). (C \neq \text{sA92} \land C \neq \text{sA91} \land C \neq \text{sA81} \rightarrow \text{sA91} \notin \text{Sources level1 C}) \)

\[ \text{proof} \]

**Lemma A81-Acc**: \( \text{Acc level1 sA81} = \{ \text{sA81}, \text{sA91}, \text{sA92} \} \)

\[ \text{proof} \]

**Lemma A81-NSources-L1**: \( \forall C \in (\text{AbstrLevel level1}). (C \neq \text{sA92} \land C \neq \text{sA91} \land C \neq \text{sA81} \rightarrow \text{sA81} \notin \text{Sources level1 C}) \)

\[ \text{proof} \]

**Lemma A71-Acc**: \( \text{Acc level1 sA71} = \{ \text{sA81}, \text{sA91}, \text{sA92} \} \)

\[ \text{proof} \]

**Lemma A71-NSources-L1**: \( \forall C \in (\text{AbstrLevel level1}). (C \neq \text{sA92} \land C \neq \text{sA91} \land C \neq \text{sA81} \rightarrow \text{sA71} \notin \text{Sources level1 C}) \)
proof

lemma A6-Acc-L1:
(\text{Acc level1 } sA6) = \{sA71, sA72, sA81, sA82, sA91, sA92, sA93\}
(proof)

lemma A6-NSources-L1Acc:
\forall C \in (\text{AbstrLevel level1}). (C \notin (\text{Acc level1 } sA6) \rightarrow sA6 \notin (\text{Sources level1 } C))
(proof)

lemma A6-NSources-L1:
\forall C \in (\text{AbstrLevel level1}). (C \neq sA93 \land C \neq sA92 \land C \neq sA91 \land C \neq sA82 \land C \neq sA81 \land C \neq sA72 \land C \neq sA71
\rightarrow sA6 \notin (\text{Sources level1 } C))
(proof)

lemma A5-Acc-L1: (\text{Acc level1 } sA5) = \{}
(proof)

lemma SourcesA11-L1: Sources level1 sA11 = \{}
(proof)

lemma SourcesA12-L1: Sources level1 sA12 = \{}
(proof)

lemma SourcesA21-L1: Sources level1 sA21 = \{sA11\}
(proof)

lemma SourcesA22-L1: Sources level1 sA22 = \{sA11, sA22, sA23, sA31, sA32, sA41\}
(proof)

lemma SourcesA23-L1: Sources level1 sA23 = \{sA11\}
(proof)

lemma SourcesA31-L1: Sources level1 sA31 = \{sA11, sA22, sA23, sA31, sA32, sA41\}
(proof)

lemma SourcesA32-L1: Sources level1 sA32 = \{sA11, sA23\}
(proof)

lemma SourcesA41-L1: Sources level1 sA41 = \{sA11, sA22, sA23, sA31, sA32, sA41\}
(proof)

lemma SourcesA42-L1: Sources level1 sA42 = \{}
(proof)

43
lemma SourcesA5-L1: Sources level1 sA5 = {sA42}
(proof)

lemma SourcesA6-L1: Sources level1 sA6 = {}
(proof)

lemma SourcesA71-L1: Sources level1 sA71 = {sA6}
(proof)

lemma SourcesA81-L1: Sources level1 sA81 = {sA6, sA71, sA81, sA91}
(proof)

lemma SourcesA91-L1: Sources level1 sA91 = {sA6, sA71, sA81, sA91}
(proof)

lemma SourcesA92-L1: Sources level1 sA92 = {sA6, sA71, sA81, sA91}
(proof)

lemma SourcesA72-L1: Sources level1 sA72 = {sA6}
(proof)

lemma SourcesA82-L1: Sources level1 sA82 = {sA6, sA72}
(proof)

lemma SourcesA93-L1: Sources level1 sA93 = {sA6, sA72, sA82}
(proof)

lemma SourcesS1-L2: Sources level2 sS1 = {}
(proof)

lemma SourcesS2-L2: Sources level2 sS2 = {}
(proof)

lemma SourcesS3-L2: Sources level2 sS3 = {sS2}
(proof)

lemma SourcesS4-L2: Sources level2 sS4 = {sS2}
(proof)

lemma SourcesS5-L2: Sources level2 sS5 = {sS2, sS4}
(proof)

lemma SourcesS6-L2: Sources level2 sS6 = {sS2, sS4, sS5}
(proof)

lemma SourcesS7-L2: Sources level2 sS7 = {}

proof

lemma SourcesS8-L2: Sources level2 sS8 = \{sS7\}
(proof)

lemma SourcesS9-L2: Sources level2 sS9 = \{}
(proof)

lemma SourcesS10-L2: Sources level2 sS10 = \{sS9\}
(proof)

lemma SourcesS11-L2: Sources level2 sS11 = \{sS9\}
(proof)

lemma SourcesS12-L2: Sources level2 sS12 = \{sS9, sS10\}
(proof)

lemma SourcesS13-L2: Sources level2 sS13 = \{sS9, sS10, sS12\}
(proof)

lemma SourcesS14-L2: Sources level2 sS14 = \{sS9, sS11\}
(proof)

lemma SourcesS15-L2: Sources level2 sS15 = \{sS9, sS11, sS14\}
(proof)

4.5 Minimal sets of components to prove certain properties

lemma minSetOfComponentsTestL2p1:
minSetOfComponents level2 \{data10, data13\} = \{sS1\}
(proof)

lemma NOT-noIrrelevantChannelsTestL2p1:
\neg \text{noIrrelevantChannels level2 } \{data10, data13\}
(proof)

lemma NOT-allNeededINChannelsTestL2p1:
\neg \text{allNeededINChannels level2 } \{data10, data13\}
(proof)

lemma minSetOfComponentsTestL2p2:
minSetOfComponents level2 \{data1, data12\} = \{sS2, sS4, sS5, sS6\}
(proof)

lemma noIrrelevantChannelsTestL2p2:
\text{noIrrelevantChannels level2 } \{data1, data12\}
(proof)
lemma allNeededINChannelsTestL2p2:
allNeededINChannels level2 {data1, data12}
(proof)

lemma minSetOfComponentsTestL1p3:
minSetOfComponents level1 {data1, data10, data11} = {sA12, sA11, sA21}
(proof)

lemma noIrrelevantChannelsTestL1p3:
noIrrelevantChannels level1 {data1, data10, data11}
(proof)

lemma allNeededINChannelsTestL1p3:
allNeededINChannels level1 {data1, data10, data11}
(proof)

lemma minSetOfComponentsTestL2p3:
minSetOfComponents level2 {data1, data10, data11} = {sS1, sS2, sS3}
(proof)

lemma noIrrelevantChannelsTestL2p3:
noIrrelevantChannels level2 {data1, data10, data11}
(proof)

lemma allNeededINChannelsTestL2p3:
allNeededINChannels level2 {data1, data10, data11}
(proof)

end

References


