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 . . 25
   3.3 Set of components needed to check a specific property . . . 26
   3.4 Additional properties: Remote Computation . . . . . . . . . 29
4 Case Study: Verification of Properties 29
   4.1 Correct composition of components . . . . . . . . . . . . . . 29
   4.2 Correct specification of the relations between channels . . . 34
   4.3 Elementary components . . . . . . . . . . . . . . . . . . . . 39
   4.4 Source components . . . . . . . . . . . . . . . . . . . . . . 42
   4.5 Minimal sets of components to prove certain properties . . 58
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 $\mathbb{I}(S)$ (function $IN :: CSet \Rightarrow \text{chanID set}$ in Isabelle) and $\mathbb{O}(S)$ (function $OUT :: CSet \Rightarrow \text{chanID set}$ 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 $\text{subcomp} :: CSet \Rightarrow CSet \rightarrow \text{CSet set}$ maps components to a (possibly empty) set of its subcomponents.

We specify the components’ dependencies by the function

$$Sources^L : CSet^L \rightarrow (CSet^L)^*$$

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):

$$Sources^L(C) = DSources^L(C) \cup \bigcup_{S \in DSources^L(C)}\{S_1 | S_1 \in Sources^L(S)\}$$

Direct data dependencies are defined by the function

$$DSources^L : CSet^L \rightarrow (CSet^L)^*$$

$$DSources^L(C) = \{S | \exists x \in I(C) \land x \in O(S)\}$$
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 \( C_2 \).

\( \mathbb{I}^D(C, y) \) denotes the subset of \( \mathbb{I}(C) \) that output channel \( y \) depends upon, directly (specified in Isabelle by function \( OUTfromCh:: chanID \Rightarrow chanID set \)) or via local variables (specified by function \( OUTfromV:: chanID \Rightarrow varID set \)). 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 \( \mathbb{I}^D(C, y) = \{ x \} \). In Isabelle, \( \mathbb{I}^D(C, y) \) is specified by function \( OUTfrom:: chanID \Rightarrow varID set \).

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 \( \mathbb{O}^D \) returns the corresponding set \( \mathbb{O}^D(C, x) \) 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: \( \forall x \in \mathbb{I}(C). \mathbb{O}^D(C, x) \neq \emptyset \).

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 \( \mathbb{I}^D \) of data dependencies between the components are defined in the theory \texttt{DataDependenciesConcreteValues.thy}. We represent the dependencies graphically using dashed lines over the component box.
Figure 1: System $S$: Data dependencies and $I^D$ sets

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 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 $minSetOfComponents$. 
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 $\text{UplSize}(f)$: 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, $\text{Perf}(X)$: 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 $\text{UplSize}$ measure should be analysed only for the channels that aren’t local for the components on abstraction levels $L_2$ and $L_3$. 

Figure 2: Components’ decomposition (level $L_1$)
Using graphical representation, we denote the channels with \textit{UplSize} measure higher than a predefined value by thick red arrows (cf. also set \textit{UplSizeHighLoad} in Isabelle theory \textit{DataDependenciesConcreteValues.thy}), and the components with \textit{Perf} measure higher than a predefined value by light green colour (cf. also set \textit{HighPerfSet} in Isabelle theory \textit{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 \textit{UplSize} measure higher than a predefined value. The components $S_4'$ and $S_7'$ have \textit{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 | sA22 | 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
fun OUT :: CSet \Rightarrow chanID set
where
  OUT sA1 = { data2, data10 }
  OUT sA2 = { data4, data5, data11, data12 }
\end{verbatim}
fun VAR :: CSet ⇒ varID set

where
  VAR sA1 = { stA1 }
  | VAR sA2 = { stA2 }

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

— function AbstrLevel maps abstraction level ID to the corresponding set of components

\textbf{axiomatization}

\textit{AbstrLevel} :: AbstrLevelsID \Rightarrow CSet set

\textbf{where}

\textit{AbstrLevel0}:
AbstrLevel level0 = \{sA1, sA2, sA3, sA4, sA5, sA6, sA7, sA8, sA9\}
and
AbstrLevel1:
AbstrLevel level1 = \{sA11, sA12, sA21, sA22, sA31, sA32, sA41, sA42, sA5, sA6, sA71, sA72, sA81, sA82, sA91, sA92, sA93\}
and
AbstrLevel2:
AbstrLevel level2 = \{sS1, sS2, sS3, sS4, sS5, sS6, sS7, sS8, sS9, sS10, sS11, sS12, sS13, sS14, sS15\}
and
AbstrLevel3:
AbstrLevel level3 = \{sS1opt, sS3, sS4opt, sS7opt, sS9, sS10, sS11opt, sS12, sS13\}

— function VARfrom maps variable ID to the set of input channels it depends from

fun VARfrom :: varID ⇒ chanID set
where
  VARfrom stA1 = \{data1\}
  VARfrom stA2 = \{data3\}
  VARfrom stA4 = \{data6, data7\}
  VARfrom stA6 = \{data14\}

— function VARto maps variable ID to the set of output channels depending from this variable

fun VARto :: varID ⇒ chanID set
where
  VARto stA1 = \{data10\}
  VARto stA2 = \{data4, data12\}
  VARto stA4 = \{data3\}
  VARto stA6 = \{data15, data16\}

— function 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

fun OUTfromCh :: chanID ⇒ chanID set
where
  OUTfromCh data1 = \{
  | OUTfromCh data2 = \{data1\}
  | OUTfromCh data3 = \{
  | OUTfromCh data4 = \{data2\}
  | OUTfromCh data5 = \{data2\}
  | OUTfromCh data6 = \{data4\}
  | OUTfromCh data7 = \{data5\}
  | OUTfromCh data8 = \{data13\}
  | OUTfromCh data9 = \{data8\}
  | OUTfromCh data10 = \{

12
| OUTfromCh data11 = \{data2\} |
| OUTfromCh data12 = {} |
| 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 = \{stA1\} |
| 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 greater than the predefined value HighLoad

definition

UplSizeHighLoad :: chanID set

where
UplSizeHighLoad ≡ \{\text{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 \(High\Perf\).

**definition**

\(High\PerfSet :: \text{CSet set} \)

**where**

\(High\PerfSet \equiv \{sA22, sA23, sA41, sA42, sA72, sA93\}\)

end

3 Inter-/Intracomponent dependencies

**theory** \(Data\Dependencies\)

**imports** \(Data\DependenciesConcreteValues\)

**begin**

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

**definition**

\(correct\Composition\DiffLevels :: \text{CSet} \Rightarrow \text{bool} \)

**where**

\(correct\Composition\DiffLevels S \equiv \forall C \in \text{subcomp } S. \forall i. S \in \text{AbstrLevel } i \rightarrow C \notin \text{AbstrLevel } i\)

— 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**

\(correct\Composition\DiffLevels\text{SYSTEM} :: \text{bool} \)

**where**

\(correct\Composition\DiffLevels\text{SYSTEM} \equiv (\forall S :: \text{CSet}. (correct\Composition\DiffLevels\text{SYSTEM} S))\)

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

**definition**

\(correct\Composition\VAR :: \text{CSet} \Rightarrow \text{bool} \)

**where**

\(correct\Composition\VAR S \equiv \forall C \in \text{subcomp } S. \forall v \in \VAR C. \; v \in \VAR S\)

— 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**

\(correct\Composition\VAR\text{SYSTEM} :: \text{bool} \)
where
\[ \text{correctCompositionVARSYSTEM} \equiv (\forall S : \text{CSet}. (\text{correctCompositionVAR} S)) \]

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

definition
\[ \text{correctDeCompositionVAR} :: \text{CSet} \Rightarrow \text{bool} \]
where
\[ \text{correctDeCompositionVAR} S \equiv (\forall v \in \text{VAR} S. \forall C1 \in \text{subcomp} S. \forall C2 \in \text{subcomp} S. v \in \text{VAR} C1 \land v \in \text{VAR} C2 \rightarrow C1 = C2) \]

— 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
\[ \text{correctDeCompositionVARSYSTEM} :: \text{bool} \]
where
\[ \text{correctDeCompositionVARSYSTEM} \equiv (\forall S : \text{CSet}. (\text{correctDeCompositionVAR} S)) \]

— 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
\[ \text{correctCompositionOUT} :: \text{chanID} \Rightarrow \text{bool} \]
where
\[ \text{correctCompositionOUT} x \equiv (\forall C i. x \in \text{OUT} C \land C \in \text{AbstrLevel} i \rightarrow (\forall S \in \text{AbstrLevel} i. x \notin \text{OUT} S) \]

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

definition
\[ \text{correctCompositionOUTSYSTEM} :: \text{bool} \]
where
\[ \text{correctCompositionOUTSYSTEM} \equiv (\forall x. \text{correctCompositionOUT} x) \]

— 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
\[ \text{correctCompositionSubcomp} :: \text{CSet} \Rightarrow \text{bool} \]
where
\[ \text{correctCompositionSubcomp} X \equiv (\forall C i. X \in \text{subcomp} C \land C \in \text{AbstrLevel} i \rightarrow (\forall S \in \text{AbstrLevel} i. (S \neq C \rightarrow X \notin \text{subcomp} S))) \]

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

correctCompositionSubcompSYSTEM :: bool
where
  correctCompositionSubcompSYSTEM ≡ (∀ X. correctCompositionSubcomp X)

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

definition

correctDeCompositionVARempty :: bool
where
  correctDeCompositionVARempty ≡
  correctCompositionVAR S ∧ VAR S = {}

— 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 ∀ C ∈ subcomp S. VAR C = {}
  using assms by (metis all-not-in-conv correctCompositionVAR-def)

— function OUTfrom maps channel ID to the set of input channels it depends from,
  directly (OUTfromCh) or via local variables (VARfrom)
  an empty set means that the channel is either input of the system or
  its values are generated within some component independently

definition

OUTfrom :: chanID ⇒ chanID set
where
  OUTfrom x ≡ (OUTfromCh x) ∪ {y. ∃ v. v ∈ (OUTfromV x) ∧ y ∈ (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 ⇒ bool
where
  OUTfromChCorrect x ≡
  (OUTfromCh x ≠ {} → (∃ Z . (x ∈ (OUT Z) ∧ (∀ y ∈ (OUTfromCh x). y ∈ 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 :: bool
where
  OUTfromChCorrectSYSTEM ≡ (∀ x::chanID. (OUTfromChCorrect x))
— if \( x \) depends on some local variables, then exists a component
to which these variables belong and which has \( x \) as an output channel

**definition**
\[ \text{OUTfromVCorrect1} :: \text{chanID} \Rightarrow \text{bool} \]

**where**
\[ \text{OUTfromVCorrect1} \ x \equiv \]
\[ (\text{OUTfromV} \ x \neq \{\} \rightarrow (\exists \ Z. \ (x \in (\text{OUT} \ Z) \land (\forall \ v \in (\text{OUTfromV} \ x). \ v \in \text{VAR} \ Z)))) \]

— 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 on any local variables, then it does not belong to any set \( \text{VARfrom} \)

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

**where**
\[ \text{OUTfromVCorrect2} \ x \equiv \]
\[ (\text{OUTfromV} \ x = \{\} \rightarrow (\forall \ v::\text{varID}. \ x /\in (\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 \( \text{OUTfromV} \) and \( \text{VARto} \) 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
VARtoCorrectSYSTEM :: bool
where
VARtoCorrectSYSTEM ≡
(∀ v::varID. ∀ Z ∈ ((AbstrLevel level0) ∪ (AbstrLevel level1))
( v ∈ VAR Z) → (∀ x ∈ VARto v. x ∈ 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)
using assms by (simp add: OUTfromV-VARto-def, auto)

3.1 Direct and indirect data dependencies between components
— The component C should be defined on the same abstraction
— level we are seaching 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)
by (simp add: DSources-def, auto)

— The component C should be defined on the same abstraction level we are
— seaching for its direct or indirect acceptors (coponents, for which C is a source),
— otherwise we get an empty set as result
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

\[ \text{Sources} :: \text{AbstrLevelsID} \Rightarrow \text{CSet} \Rightarrow \text{CSet set} \]

where

SourcesDef:
\[(\text{Sources } i \ C) = (\text{DSources } i \ C) \cup (\bigcup S \in (\text{DSources } i \ C). (\text{Sources } i \ S)) \]

and

SourceExistsDSource:
\[S \in (\text{Sources } i \ C) \rightarrow (\exists Z. S \in (\text{DSources } i \ Z))\]

and

NDSourceExistsDSource:
\[S \in (\text{Sources } i \ C) \land S \notin (\text{DSources } i \ C) \rightarrow (\exists Z. S \in (\text{DSources } i \ Z) \land Z \in (\text{Sources } i \ C))\]

and

SourcesTrans:
\[(C \in \text{Sources } i \ S \land S \in \text{Sources } i \ Z) \rightarrow C \in \text{Sources } i \ Z\]

and

SourcesLevelX:
\[(\text{Sources } i \ X) \subseteq (\text{AbstrLevel } i)\]

and

SourcesLoop:
\[(\text{Sources } i \ C) = (\text{Sources } i \ X) \land (\text{Sources } i \ S) = (\text{Sources } i \ Z) \land (\text{Sources } i \ C) \land Z \in (\text{Sources } i \ C)\]

— if we have a loop in the dependencies we need to cut it for counting the sources

axiomatization

\[ \text{Acc} :: \text{AbstrLevelsID} \Rightarrow \text{CSet} \Rightarrow \text{CSet set} \]

where

AccDef:
\[(\text{Acc } i \ C) = (\text{DAcc } i \ C) \cup (\bigcup S \in (\text{DAcc } i \ C). (\text{Acc } i \ S)) \]

and

Acc-Sources:
\[(X \in \text{Acc } i \ C) = (C \in \text{Sources } i \ X)\]

and

AccSigleLoop:
\[(\text{DAcc } i \ C) = \{S\} \land (\text{DAcc } i \ S) = \{C\} \rightarrow \text{Acc } i \ C = \{C, S\}\]

and

AccLoop:
\[(\text{Acc } i \ C) = (\text{Sources } i \ X) \land (\text{Acc } i \ S) = (\text{Sources } i \ Z) \land (\text{Acc } i \ C) \land Z \in (\text{Sources } i \ C)\]

— if we have a loop in the dependencies we need to cut it for counting the accessors

lemma Acc-SourcesNOT: \((X \notin \text{Acc } i \ C) = (C \notin \text{Sources } i \ X)\)

by (metis Acc-Sources)

— component S is not a source for any component on the abstraction level i

definition

\[ \text{isNotDSource} :: \text{AbstrLevelsID} \Rightarrow \text{CSet} \Rightarrow \text{bool} \]

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 \ CSet \Rightarrow \ CSet \Rightarrow \ 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 \ = (\forall \ C. \ isNotDSourceX \ i \ S \ C)
bym (auto, (simp add: isNotDSource-def isNotDSourceX-def)+)

lemma DAcc-DSources:
(X \in \ DAcc \ i \ C) = (C \in \ DSources \ i \ X)
bym (auto, (simp add: DAcc-def DSources-def, auto)+)

lemma DAcc-DSourcesNOT:
(X \notin \ DAcc \ i \ C) = (C \notin \ DSources \ i \ X)
bym (auto, (simp add: DAcc-def DSources-def, auto)+)

lemma DSource-level:
assumes S \in (\DSources \ i \ C)
shows C \in (AbstrLevel \ i)
using asms by (simp add: DSources-def, auto)

lemma SourceExistsDSource-level:
assumes S \in (\Sources \ i \ C)
shows \exists Z \in (AbstrLevel \ i). (S \in (\DSources \ i \ Z))
using asms by (metis DSource-level SourceExistsDSource)

lemma Sources-DSources:
(\DSources \ i \ C) \subseteq (\Sources \ i \ C)
proof –
have (\Sources \ i \ C) = (\DSources \ i \ C) \cup (\union S \in (\DSources \ i \ C). (\Sources \ i \ S))
by (rule SourcesDef)
thus ?thesis by auto
qed

lemma NoDSourceNoSource:
assumes S \notin (\Sources \ i \ C)
shows S \notin (\DSources \ i \ C)
using asms by (metis (full-types) Sources-DSources set-rev-mp)

lemma DSourcesEmptySources:
assumes DSources \ i \ C = {}
shows Sources \ i \ C = {}
proof –
have (\Sources \ i \ C) = (\DSources \ i \ C) \cup (\union S \in (\DSources \ i \ C). (\Sources \ i \ S))
by (rule SourcesDef)
with assms show ?thesis by auto
qed

lemma DSource-Sources:
assumes $S \in (\text{DSources } i \ C)$
shows $(\text{Sources } i \ S) \subseteq (\text{Sources } i \ C)$
proof –
have $(\text{Sources } i \ C) = (\text{DSources } i \ C) \cup (\bigcup S \in (\text{DSources } i \ C). (\text{Sources } i \ S))$
by (rule SourcesDef)
with assms show ?thesis by auto
qed

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 –
have sDef: $(\text{Sources } i \ C) = (\text{DSources } i \ C) \cup (\bigcup S \in (\text{DSources } i \ C). (\text{Sources } i \ S))$
by (rule SourcesDef)
from assms have $\forall X. (X \in (\text{DSources } i \ C) \rightarrow (\text{Sources } i \ X) = \{\})$
by (simp add: DSourcesEmptySources)
hence $(\bigcup S \in (\text{DSources } i \ C). (\text{Sources } i \ S)) = \{\} \text{ by auto}$
with sDef show ?thesis by simp
qed

lemma SourcesEmptyDSources:
assumes Sources $i \ C = \{\}$
shows DSources $i \ C = \{\}$
using assms by (metis Sources-DSources bot.extremum-uniqueI)

lemma NotDSource:
assumes $\forall x \in (\text{OUT } S). (\forall Z \in (\text{AbstrLevel } i). (x \notin (\text{IN } Z)))$
shows $\forall C \in (\text{AbstrLevel } i). S \notin (\text{DSources } i \ C)$
using assms by (simp add: AbstrLevel0 DSources-def)

lemma allNotDSource-NotSource:
assumes $\forall C . S \notin (\text{DSources } i \ C)$
shows $\forall Z . S \notin (\text{Sources } i \ Z)$
using assms by (metis SourceExistsDSource)

lemma NotDSource-NotSource:
assumes $\forall C \in (\text{AbstrLevel } i). S \notin (\text{DSources } i \ C)$
shows $\forall Z \in (\text{AbstrLevel } i). S \notin (\text{Sources } i \ Z)$
using assms by (metis SourceExistsDSource-level)

lemma isNotSource-Sources:
assumes $\text{isNotDSource } i \ S$
shows $\forall C \in (\text{AbstrLevel } i). S \notin (\text{Sources } i \ C)$
using \texttt{assms} \\
by (simp add: isNotDSource-def, metis (full-types) NotDSource NotDSource-NotSource)

\textbf{lemma} SourcesAbstrLevel: \\
\texttt{assumes} \quad x \in \text{Sources} \circ i \ S \\
\texttt{shows} \quad x \in \text{AbstrLevel} \circ i \\
\texttt{using} \quad \texttt{assms} \\
by (metis SourcesLevelX in-mono)

\textbf{lemma} DSourceIsSource: \\
\texttt{assumes} \quad C \in \text{DSources} \circ i \ S \\
\texttt{shows} \quad C \in \text{Sources} \circ i \ S \\
\texttt{proof} – \\
\texttt{have} \quad (\text{Sources} \circ i \ S) = (\text{DSources} \circ i \ S) \cup (\bigcup Z \in (\text{DSources} \circ i \ S). (\text{Sources} \circ i \ Z)) \\
\texttt{by} \quad (\text{rule SourcesDef}) \\
\texttt{with} \quad \texttt{assms} \quad \texttt{show} \quad \texttt{thesis} \quad \texttt{by} \quad \texttt{simp} \\
\texttt{qed}

\textbf{lemma} DSourceOfDSource: \\
\texttt{assumes} \quad Z \in \text{DSources} \circ i \ S \\
\quad \text{and} \quad S \in \text{DSources} \circ i \ C \\
\texttt{shows} \quad Z \in \text{Sources} \circ i \ C \\
\texttt{using} \quad \texttt{assms} \\
\texttt{proof} – \\
\texttt{from} \quad \texttt{assms} \quad \texttt{have} \quad \texttt{src:Sources} \circ i \ S \subseteq \text{Sources} \circ i \ C \quad \texttt{by} \quad (\text{simp add: DSource-Sources}) \\
\texttt{from} \quad \texttt{assms} \quad \texttt{have} \quad Z \in \text{Sources} \circ i \ S \quad \texttt{by} \quad (\text{simp add: DSourceIsSource}) \\
\texttt{with} \quad \texttt{src} \quad \texttt{show} \quad \texttt{thesis} \quad \texttt{by} \quad \texttt{auto} \\
\texttt{qed}

\textbf{lemma} SourceOfDSource: \\
\texttt{assumes} \quad Z \in \text{Sources} \circ i \ S \\
\quad \text{and} \quad S \in \text{DSources} \circ i \ C \\
\texttt{shows} \quad Z \in \text{Sources} \circ i \ C \\
\texttt{using} \quad \texttt{assms} \\
\texttt{proof} – \\
\texttt{from} \quad \texttt{assms} \quad \texttt{have} \quad \texttt{Sources} \circ i \ S \subseteq \text{Sources} \circ i \ C \quad \texttt{by} \quad (\text{simp add: DSource-Sources}) \\
\texttt{thus} \quad \texttt{thesis} \quad \texttt{by} \quad (\text{metis (full-types) assms(1) set-rev-mp}) \\
\texttt{qed}

\textbf{lemma} DSourceOfSource: \\
\texttt{assumes} \quad cDS: C \in \text{DSources} \circ i \ S \\
\quad \text{and} \quad sS:S \in \text{Sources} \circ i \ Z \\
\texttt{shows} \quad C \in \text{Sources} \circ i \ Z \\
\texttt{proof} – \\
\texttt{from} \quad \texttt{cDS} \quad \texttt{have} \quad C \in \text{Sources} \circ i \ S \quad \texttt{by} \quad (\text{simp add: DSourceIsSource}) \\
\texttt{from} \quad \texttt{this} \quad \texttt{and} \quad \texttt{sS} \quad \texttt{show} \quad \texttt{thesis} \quad \texttt{by} \quad (\text{metis (full-types) SourcesTrans}) \\
\texttt{qed}

\textbf{lemma} Sources-singleDSource:

22
assumes $\text{DSources} i S = \{C\}$
shows $\text{Sources} i S = \{C\} \cup \text{Sources} i C$
proof
have sDef: $(\text{Sources} i S) = (\text{DSources} i S) \cup (\bigcup Z \in (\text{DSources} i S). (\text{Sources} i Z))$
  by (rule SourcesDef)
from assms have $(\bigcup Z \in (\text{DSources} i S). (\text{Sources} i Z)) = \text{Sources} i C$
  by auto
with sDef assms show ?thesis by simp
qed

lemma Sources-2DSources:
assumes $\text{DSources} i S = \{C_1, C_2\}$
shows $\text{Sources} i S = \{C_1, C_2\} \cup \text{Sources} i C_1 \cup \text{Sources} i C_2$
proof
have sDef: $(\text{Sources} i S) = (\text{DSources} i S) \cup (\bigcup Z \in (\text{DSources} i S). (\text{Sources} i Z))$
  by (rule SourcesDef)
from assms have $(\bigcup Z \in (\text{DSources} i S). (\text{Sources} i Z)) = \text{Sources} i C_1 \cup \text{Sources} i C_2$
  by auto
with sDef and assms show ?thesis by simp
qed

lemma Sources-3DSources:
assumes $\text{DSources} i S = \{C_1, C_2, C_3\}$
shows $\text{Sources} i S = \{C_1, C_2, C_3\} \cup \text{Sources} i C_1 \cup \text{Sources} i C_2 \cup \text{Sources} i C_3$
proof
have sDef: $(\text{Sources} i S) = (\text{DSources} i S) \cup (\bigcup Z \in (\text{DSources} i S). (\text{Sources} i Z))$
  by (rule SourcesDef)
from assms have $(\bigcup Z \in (\text{DSources} i S). (\text{Sources} i Z)) = \text{Sources} i C_1 \cup \text{Sources} i C_2 \cup \text{Sources} i C_3$
  by auto
with sDef and assms show ?thesis by simp
qed

lemma singleDSourceEmpty4isNotDSource:
assumes $\text{DAcc} i C = \{S\}$
  and $Z \neq S$
shows $C \notin (\text{DSources} i Z)$
proof
from assms have $(Z \notin \text{DAcc} i C)$ by simp
thus ?thesis by (simp add: DAcc-DSourcesNOT)
qed

lemma singleDSourceEmpty4isNotDSourceLevel:
assumes $\text{DAcc} i C = \{S\}$
shows \( \forall Z \in (\text{AbstrLevel } i), \ Z \neq S \rightarrow C \notin (\text{Sources } i \ Z) \)

using assms by (metis singleDSourceEmpty4isNotDSource)

**lemma** isNotDSource-EmptyDAcc:
assumes isNotDSource i S
shows \( \text{DAcc } i \ S = \{\} \)
using assms by (simp add: DAcc-def isNotDSource-def, auto)

**lemma** isNotDSource-EmptyAcc:
assumes isNotDSource i S
shows \( \text{Acc } i \ S = \{\} \)
proof −
  have \((\text{Acc } i \ S) = (\text{DAcc } i \ S) \cup (\bigcup X \in (\text{DAcc } i \ S). \ (\text{Acc } i \ X))\)
    by (rule AccDef)
  thus \(?thesis \) by (metis SUP-empty Un-absorb assms isNotDSource-EmptyDAcc)
qed

**lemma** singleDSourceEmpty-Acc:
assumes \( \text{DAcc } i \ C = \{S\} \)
and isNotDSource i S
shows \( \text{Acc } i \ C = \{S\} \)
proof −
  have \(\text{Acc } C: (\text{Acc } i \ C) = (\text{DAcc } i \ C) \cup (\bigcup S \in (\text{DAcc } i \ C). \ (\text{Acc } i \ S))\)
    by (rule AccDef)
  from assms have \(\text{Acc } i \ S = \{\}\) by (simp add: isNotDSource-EmptyAcc)
  with AccC show \(?thesis \)
    by (metis SUP-empty UN-insert Un-commute Un-empty-left assms(1))
qed

**lemma** singleDSourceEmpty4isNotSource:
assumes \( \text{DAcc } i \ C = \{S\} \)
and isNotDSource i S
and \( Z \neq S \)
shows \( C \notin (\text{Sources } i \ Z) \)
proof −
  from assms have \(\text{Acc } i \ C = \{S\} \) by (simp add: singleDSourceEmpty-Acc)
  with assms have \( Z \notin \text{Acc } i \ C \) by simp
  thus \(?thesis \) by (simp add: Acc-SourcesNOT)
qed

**lemma** singleDSourceEmpty4isNotSourceLevel:
assumes \( \text{DAcc } i \ C = \{S\} \)
and isNotDSource i S
shows \( \forall Z \in (\text{AbstrLevel } i), \ Z \neq S \rightarrow C \notin (\text{Sources } i \ Z) \)
using assms
by (metis singleDSourceEmpty4isNotDSource)
lemma singleDSourceLoop:
assumes \( DA \cdot i \cdot C = \{ S \} \)
and \( DA \cdot i \cdot S = \{ C \} \)
shows \( \forall Z \in (AbstrLevel i). (Z \neq S \land Z \neq C \rightarrow C \notin (Sources i Z)) \)
using assms
by (metis AccSigleLoop Acc-SourcesNOT empty-iff insert-iff)

3.2 Components that are elementary wrt. data dependencies
— two output channels of a component \( C \) are correlated, if they mutually depend on the same local variable(s)
definition outPairCorelated :: CSet ⇒ chanID ⇒ chanID ⇒ bool
where
outPairCorelated \( C \cdot x \cdot y \equiv (x \in OUT C) \land (y \in OUT C) \land (OUTfromV x) \cap (OUTfromV y) \neq \{} \)

— 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 :: chanID ⇒ chanID set
where
outSetCorelated \( x \equiv \{ y :: \text{chanID} . \exists v :: \text{varID}. (v \in (OUTfromV x) \land (y \in VARto v)) \} \)

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

— the set (outSetCorelated \( x \)) is empty if \( x \) does not depend from any variable
lemma outSetCorelatedEmpty1:
assumes \( OUTfromV x = \{ \} \)
shows outSetCorelated \( x = \{ \} \)
using assms by (simp add: outSetCorelated-def)

— if \( x \) depends from at least one variable and the predicates OUTfromV and VARto
are defined correctly,
— the set \( \text{outSetCorrelated} \ x \) contains \( x \) itself

**Lemma** \( \text{outSetCorrelatedNonemptyX} \):

**Assumes** \( \text{OUTfromV} \ x \neq \{\} \) and correct3: \( \text{OUTfromV-VARto} \)

**Shows** \( x \in \text{outSetCorrelated} \ x \)

**Proof** —

from assms have \( \exists v :: \text{varID}. \ x \in (\text{VARto} \ v) \)

by (rule \( \text{OUTfromV-VARto-lemma} \))

from this and assms show \( \) ?thesis

by (simp add: \( \text{outSetCorrelated-def} \) \( \text{OUTfromV-VARto-def} \))

qed

— if the set \( \text{outSetCorrelated} \ x \) is empty, this means that \( x \) does not depend from any variable

**Lemma** \( \text{outSetCorrelatedEmpty2} \):

**Assumes** \( \text{outSetCorrelated} \ x = \{\} \) and correct3: \( \text{OUTfromV-VARto} \)

**Shows** \( \text{OUTfromV} \ x = \{\} \)

**Proof** (rule \( \text{ccontr} \))

assume \( \text{OUTfromVNonempty}; \text{OUTfromV} \ x \neq \{\} \)

from this and correct3 have \( x \in \text{outSetCorrelated} \ x \)

by (rule \( \text{outSetCorrelatedNonemptyX} \))

from this and assms show False by simp

qed

### 3.3 Set of components needed to check a specific property

— set of components specified on abstreaction level \( i \), which input channels belong to the set \( \text{chSet} \)

**Definition**

\( \text{inSetOfComponents} :: \text{AbstrLevelsID} \Rightarrow \text{chanID set} \Rightarrow \text{CSet set} \)

where

\( \text{inSetOfComponents} \ i \ \text{chSet} \equiv \{X. ((\text{IN} \ X) \cap \text{chSet} \neq \{\}) \land X \in (\text{AbstrLevel} \ i))\}\)

— Set of components from the abstraction level \( i \), which output channels belong to the set \( \text{chSet} \)

**Definition**

\( \text{outSetOfComponents} :: \text{AbstrLevelsID} \Rightarrow \text{chanID set} \Rightarrow \text{CSet set} \)

where

\( \text{outSetOfComponents} \ i \ \text{chSet} \equiv \{Y. ((\text{OUT} \ Y) \cap \text{chSet} \neq \{\}) \land Y \in (\text{AbstrLevel} \ i))\}\)

— Set of components from the abstraction level \( i \),

— which have output channels from the set \( \text{chSet} \) or are sources for such components

**Definition**

\( \text{minSetOfComponents} :: \text{AbstrLevelsID} \Rightarrow \text{chanID set} \Rightarrow \text{CSet set} \)

where

\( \text{minSetOfComponents} \ i \ \text{chSet} \equiv \)
\[(\text{outSetOfComponents } i \text{ chSet}) \cup\]
\[((\bigcup S \in (\text{outSetOfComponents } i \text{ chSet})). (\text{Sources } S))\]

— 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

using \text{assms} by \((\text{simp add: systemIN-def systemOUT-def})\)

**lemma** \text{systemOUT-noIN}:

assumes \text{systemOUT } x \ i

shows \neg \text{systemIN } x \ i

using \text{assms} by \((\text{simp add: systemIN-def systemOUT-def})\)

**lemma** \text{systemIN-noLOC}:

assumes \text{systemIN } x \ i

shows \neg \text{systemLOC } x \ i

using \text{assms} by \((\text{simp add: systemIN-def systemLOC-def})\)

**lemma** \text{systemLOC-noIN}:

assumes \text{systemLOC } x \ i

shows \neg \text{systemIN } x \ i

using \text{assms} by \((\text{simp add: systemIN-def systemLOC-def})\)

**lemma** \text{systemOUT-noLOC}:

assumes \text{systemOUT } x \ i

shows \neg \text{systemLOC } x \ i

using \text{assms} by \((\text{simp add: systemOUT-def systemLOC-def})\)

**lemma** \text{systemLOC-noOUT}:
assumes system\text{LOC} \, x \, i
shows ¬ system\text{OUT} \, x \, i
using assms by (simp add: system\text{LOC-def} system\text{OUT-def})

definition
noIrrelevantChannels :: AbstrLevelsID ⇒ chanID set ⇒ bool
where
noIrrelevantChannels \, i \, \text{chSet} ≡
\forall x \in \text{chSet}. \, ((\text{systemIN} \, x \, i) \, \longrightarrow
(\exists Z \in (\text{minSetOfComponents} \, i \, \text{chSet} \, . \, x \in (\text{IN} \, Z))))

definition
allNeededINChannels :: AbstrLevelsID ⇒ chanID set ⇒ bool
where
allNeededINChannels \, i \, \text{chSet} ≡
(\forall Z \in (\text{minSetOfComponents} \, i \, \text{chSet} \, . \, \exists x \in (\text{IN} \, Z) \, . \, ((\text{systemIN} \, x \, i) \, \longrightarrow \, (x \in \text{chSet})))

— the set (\text{outSetOfComponents} \, i \, \text{chSet}) should be a subset of all components
specified on the abstraction level \, i
lemma outSetOfComponentsLimit:
outSetOfComponents \, i \, \text{chSet} \subseteq \text{AbstrLevel} \, i
by (metis (lifting) mem-Collect-eq outSetOfComponents-def subsetI)

— the set (\text{inSetOfComponents} \, i \, \text{chSet}) should be a subset of all components
specified on the abstraction level \, i
lemma inSetOfComponentsLimit:
inSetOfComponents \, i \, \text{chSet} \subseteq \text{AbstrLevel} \, i
by (metis (lifting) inSetOfComponents-def mem-Collect-eq subsetI)

— the set of components, which are sources for the components
— out of (\text{inSetOfComponents} \, i \, \text{chSet}), should be a subset of
— all components specified on the abstraction level \, i
lemma SourcesLevelLimit:
(\bigcup S \in (\text{outSetOfComponents} \, i \, \text{chSet}). \, (\text{Sources} \, i \, S)) \subseteq \text{AbstrLevel} \, i
proof —
have sg1: outSetOfComponents \, i \, \text{chSet} \subseteq \text{AbstrLevel} \, i
by (simp add: outSetOfComponentsLimit)
have \forall S. \, S \in (\text{outSetOfComponents} \, i \, \text{chSet}) \, \longrightarrow \, \text{Sources} \, i \, S \subseteq \text{AbstrLevel} \, i
by (metis SourcesLevelX)
from this and sg1 show ?thesis by auto
qed

lemma minSetOfComponentsLimit:
minSetOfComponents \, i \, \text{chSet} \subseteq \text{AbstrLevel} \, i
proof —
have sg1: outSetOfComponents \, i \, \text{chSet} \subseteq \text{AbstrLevel} \, i
by (simp add: outSetOfComponentsLimit)
have \( (\bigcup S \in \text{outSetOfComponents i chSet}. (\text{Sources } i S)) \subseteq \text{AbstrLevel } i \)
by (simp add: SourcesLevelLimit)
with \(sg1 \)
show \(?thesis\) by (simp add: minSetOfComponents-def)
qed

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

4 Case Study: Verification of Properties

theory DataDependenciesCaseStudy
  imports DataDependencies
begin

4.1 Correct composition of components
— the lemmas \(\text{AbstrLevels X Y}\) with corresponding proofs can be composed
— and proven automatically, their proofs are identical
lemma \(\text{AbstrLevels-A1-A11}:\)
  assumes \(sA1 \in \text{AbstrLevel } i\)
  shows \(sA11 \notin \text{AbstrLevel } i\)
using \(\text{assms}\)
by (induct \(i\), simp add: AbstrLevel0, simp add: AbstrLevel1, simp add: AbstrLevel2, simp add: AbstrLevel3)

lemma \(\text{AbstrLevels-A1-A12}:\)
  assumes \(sA1 \in \text{AbstrLevel } i\)
  shows \(sA12 \notin \text{AbstrLevel } i\)

lemma \(\text{AbstrLevels-A2-A21}:\)
  assumes \(sA2 \in \text{AbstrLevel } i\)
  shows \(sA21 \notin \text{AbstrLevel } i\)
lemma AbstrLevels-A2-A22:
assumes $sA2 \in \text{AbstrLevel } i$
shows $sA22 \notin \text{AbstrLevel } i$

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

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

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

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

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

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

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

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

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

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

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

lemma AbstrLevels-A9-A93:
assumes $sA9 \in \text{AbstrLevel } i$
shows $sA93 \notin AbstrLevel i$

**lemma AbstrLevels-S1-A12:**
assumes $sS1 \in AbstrLevel i$
shows $sA12 \notin AbstrLevel i$

**lemma AbstrLevels-S2-A11:**
assumes $sS2 \in AbstrLevel i$
shows $sA11 \notin AbstrLevel i$

**lemma AbstrLevels-S3-A21:**
assumes $sS3 \in AbstrLevel i$
shows $sA21 \notin AbstrLevel i$

**lemma AbstrLevels-S4-A23:**
assumes $sS4 \in AbstrLevel i$
shows $sA23 \notin AbstrLevel i$

**lemma AbstrLevels-S5-A32:**
assumes $sS5 \in AbstrLevel i$
shows $sA32 \notin AbstrLevel i$

**lemma AbstrLevels-S6-A22:**
assumes $sS6 \in AbstrLevel i$
shows $sA22 \notin AbstrLevel i$

**lemma AbstrLevels-S6-A31:**
assumes $sS6 \in AbstrLevel i$
shows $sA31 \notin AbstrLevel i$

**lemma AbstrLevels-S6-A41:**
assumes $sS6 \in AbstrLevel i$
shows $sA41 \notin AbstrLevel i$

**lemma AbstrLevels-S7-A42:**
assumes $sS7 \in AbstrLevel i$
shows $sA42 \notin AbstrLevel i$

**lemma AbstrLevels-S8-A5:**
assumes $sS8 \in AbstrLevel i$
shows $sA5 \notin AbstrLevel i$

**lemma AbstrLevels-S9-A6:**
assumes $sS9 \in AbstrLevel i$
shows $sA6 \notin AbstrLevel i$

**lemma AbstrLevels-S10-A71:**
assumes $sS10 \in AbstrLevel i$
shows $sA71 \notin AbstrLevel i$
lemma AbstrLevels-S11-A72:
assumes $sS11 \in AbstrLevel \ i$
shows $sA72 \notin AbstrLevel \ i$

lemma AbstrLevels-S12-A81:
assumes $sS12 \in AbstrLevel \ i$
shows $sA81 \notin AbstrLevel \ i$

lemma AbstrLevels-S12-A91:
assumes $sS12 \in AbstrLevel \ i$
shows $sA91 \notin AbstrLevel \ i$

lemma AbstrLevels-S13-A92:
assumes $sS13 \in AbstrLevel \ i$
shows $sA92 \notin AbstrLevel \ i$

lemma AbstrLevels-S14-A82:
assumes $sS14 \in AbstrLevel \ i$
shows $sA82 \notin AbstrLevel \ i$

lemma AbstrLevels-S15-A93:
assumes $sS15 \in AbstrLevel \ i$
shows $sA93 \notin AbstrLevel \ i$

lemma AbstrLevels-S1opt-A11:
assumes $sS1opt \in AbstrLevel \ i$
shows $sA11 \notin AbstrLevel \ i$

lemma AbstrLevels-S1opt-A12:
assumes $sS1opt \in AbstrLevel \ i$
shows $sA12 \notin AbstrLevel \ i$

lemma AbstrLevels-S4opt-A23:
assumes $sS4opt \in AbstrLevel \ i$
shows $sA23 \notin AbstrLevel \ i$

lemma AbstrLevels-S4opt-A32:
assumes $sS4opt \in AbstrLevel \ i$
shows $sA32 \notin AbstrLevel \ i$

lemma AbstrLevels-S4opt-A22:
assumes $sS4opt \in AbstrLevel \ i$
shows $sA22 \notin AbstrLevel \ i$

lemma AbstrLevels-S4opt-A31:
assumes $sS4opt \in AbstrLevel \ i$
shows $sA31 \notin AbstrLevel \ i$
lemma $\text{AbstrLevels-S4opt-A41}$:
assumes $s_{S4\text{opt}} \in \text{AbstrLevel } i$
shows $s_{A41} \notin \text{AbstrLevel } i$

lemma $\text{AbstrLevels-S7opt-A42}$:
assumes $s_{S7\text{opt}} \in \text{AbstrLevel } i$
shows $s_{A42} \notin \text{AbstrLevel } i$

lemma $\text{AbstrLevels-S7opt-A5}$:
assumes $s_{S7\text{opt}} \in \text{AbstrLevel } i$
shows $s_{A5} \notin \text{AbstrLevel } i$

lemma $\text{AbstrLevels-S11opt-A72}$:
assumes $s_{S11\text{opt}} \in \text{AbstrLevel } i$
shows $s_{A72} \notin \text{AbstrLevel } i$

lemma $\text{AbstrLevels-S11opt-A82}$:
assumes $s_{S11\text{opt}} \in \text{AbstrLevel } i$
shows $s_{A82} \notin \text{AbstrLevel } i$

lemma $\text{AbstrLevels-S11opt-A93}$:
assumes $s_{S11\text{opt}} \in \text{AbstrLevel } i$
shows $s_{A93} \notin \text{AbstrLevel } i$

lemma $\text{correctCompositionDiffLevelsA1}$: $\text{correctCompositionDiffLevels } s_{A1}$

lemma $\text{correctCompositionDiffLevelsA2}$: $\text{correctCompositionDiffLevels } s_{A2}$

lemma $\text{correctCompositionDiffLevelsA3}$: $\text{correctCompositionDiffLevels } s_{A3}$

lemma $\text{correctCompositionDiffLevelsA4}$: $\text{correctCompositionDiffLevels } s_{A4}$

— lemmas $\text{correctCompositionDiffLevelsX}$ and corresponding proofs
— are identical for all elementary components, they can be constructed automatically
lemma $\text{correctCompositionDiffLevelsA5}$: $\text{correctCompositionDiffLevels } s_{A5}$
lemma $\text{correctCompositionDiffLevelsA6}$: $\text{correctCompositionDiffLevels } s_{A6}$
lemma $\text{correctCompositionDiffLevelsA7}$: $\text{correctCompositionDiffLevels } s_{A7}$
lemma $\text{correctCompositionDiffLevelsA8}$: $\text{correctCompositionDiffLevels } s_{A8}$
lemma $\text{correctCompositionDiffLevelsA9}$: $\text{correctCompositionDiffLevels } s_{A9}$
lemma $\text{correctCompositionDiffLevelsA11}$: $\text{correctCompositionDiffLevels } s_{A11}$
lemma $\text{correctCompositionDiffLevelsA12}$: $\text{correctCompositionDiffLevels } s_{A12}$
lemma $\text{correctCompositionDiffLevelsA21}$: $\text{correctCompositionDiffLevels } s_{A21}$
lemma $\text{correctCompositionDiffLevelsA22}$: $\text{correctCompositionDiffLevels } s_{A22}$
lemma $\text{correctCompositionDiffLevelsA23}$: $\text{correctCompositionDiffLevels } s_{A23}$
lemma $\text{correctCompositionDiffLevelsA31}$: $\text{correctCompositionDiffLevels } s_{A31}$
lemma $\text{correctCompositionDiffLevelsA32}$: $\text{correctCompositionDiffLevels } s_{A32}$
lemma $\text{correctCompositionDiffLevelsA41}$: $\text{correctCompositionDiffLevels } s_{A41}$
lemma $\text{correctCompositionDiffLevelsA42}$: $\text{correctCompositionDiffLevels } s_{A42}$
lemma \text{correctCompositionDiffLevelsA71}: \text{correctCompositionDiffLevels sA71}

lemma \text{correctCompositionDiffLevelsA72}: \text{correctCompositionDiffLevels sA72}

lemma \text{correctCompositionDiffLevelsA81}: \text{correctCompositionDiffLevels sA81}

lemma \text{correctCompositionDiffLevelsA82}: \text{correctCompositionDiffLevels sA82}

lemma \text{correctCompositionDiffLevelsA91}: \text{correctCompositionDiffLevels sA91}

lemma \text{correctCompositionDiffLevelsA92}: \text{correctCompositionDiffLevels sA92}

lemma \text{correctCompositionDiffLevelsA93}: \text{correctCompositionDiffLevels sA93}

lemma \text{correctCompositionDiffLevelsS1}: \text{correctCompositionDiffLevels sS1}

lemma \text{correctCompositionDiffLevelsS2}: \text{correctCompositionDiffLevels sS2}

lemma \text{correctCompositionDiffLevelsS3}: \text{correctCompositionDiffLevels sS3}

lemma \text{correctCompositionDiffLevelsS4}: \text{correctCompositionDiffLevels sS4}

lemma \text{correctCompositionDiffLevelsS5}: \text{correctCompositionDiffLevels sS5}

lemma \text{correctCompositionDiffLevelsS6}: \text{correctCompositionDiffLevels sS6}

lemma \text{correctCompositionDiffLevelsS7}: \text{correctCompositionDiffLevels sS7}

lemma \text{correctCompositionDiffLevelsS8}: \text{correctCompositionDiffLevels sS8}

lemma \text{correctCompositionDiffLevelsS9}: \text{correctCompositionDiffLevels sS9}

lemma \text{correctCompositionDiffLevelsS10}: \text{correctCompositionDiffLevels sS10}

lemma \text{correctCompositionDiffLevelsS11}: \text{correctCompositionDiffLevels sS11}

lemma \text{correctCompositionDiffLevelsS12}: \text{correctCompositionDiffLevels sS12}

lemma \text{correctCompositionDiffLevelsS13}: \text{correctCompositionDiffLevels sS13}

lemma \text{correctCompositionDiffLevelsS14}: \text{correctCompositionDiffLevels sS14}

lemma \text{correctCompositionDiffLevelsS15}: \text{correctCompositionDiffLevels sS15}

lemma \text{correctCompositionDiffLevelsS1opt}: \text{correctCompositionDiffLevels sS1opt}

lemma \text{correctCompositionDiffLevelsS4opt}: \text{correctCompositionDiffLevels sS4opt}

lemma \text{correctCompositionDiffLevelsS7opt}: \text{correctCompositionDiffLevels sS7opt}

lemma \text{correctCompositionDiffLevelsS11opt}: \text{correctCompositionDiffLevels sS11opt}

lemma \text{correctCompositionDiffLevelsSYSTEM-holds}: \text{correctCompositionDiffLevelsSYSTEM}

lemma \text{correctCompositionVARSYSTEM-holds}: \text{correctCompositionVARSYSTEM}

by (simp add: \text{correctCompositionVARSYSTEM-def}, clarify, case-tac \text{S}, (simp add: \text{correctCompositionVAR-def}))+

lemma \text{correctDeCompositionVARSYSTEM-holds}: \text{correctDeCompositionVARSYSTEM}

by (simp add: \text{correctDeCompositionVARSYSTEM-def}, clarify, case-tac \text{S}, (simp add: \text{correctDeCompositionVAR-def}))+

4.2 Correct specification of the relations between channels

lemma \text{OUTfromChCorrect-data1}: \text{OUTfromChCorrect data1}

by (simp add: \text{OUTfromChCorrect-def})

lemma \text{OUTfromChCorrect-data2}: \text{OUTfromChCorrect data2}

by (metis \text{IN.simps}(27) \text{OUT.simps}(27) \text{OUTfromCh.simps}(2) \text{OUTfromChCorrect-def insertII})

lemma \text{OUTfromChCorrect-data3}: \text{OUTfromChCorrect data3}

by (metis \text{OUTfromCh.simps}(3) \text{OUTfromChCorrect-def})
lemma OUTfromChCorrect-data4: OUTfromChCorrect data4
by (metis IN.simps(2) OUT.simps(2) OUTfromCh.simps(4) OUTfromChCorrect-def insertI1 singleton-iff)

lemma OUTfromChCorrect-data5: OUTfromChCorrect data5
by (simp add: OUTfromChCorrect-def, metis IN.simps(14) OUT.simps(14) insertI1)

lemma OUTfromChCorrect-data6: OUTfromChCorrect data6
by (simp add: OUTfromChCorrect-def, metis IN.simps(15) OUT.simps(15) insertI1)

lemma OUTfromChCorrect-data7: OUTfromChCorrect data7
by (simp add: OUTfromChCorrect-def, metis IN.simps(16) OUT.simps(16) insertI1)

lemma OUTfromChCorrect-data8: OUTfromChCorrect data8
by (simp add: OUTfromChCorrect-def, metis IN.simps(18) OUT.simps(18) insertI1)

lemma OUTfromChCorrect-data9: OUTfromChCorrect data9
by (simp add: OUTfromChCorrect-def, metis IN.simps(33) OUT.simps(33) singleton-iff)

lemma OUTfromChCorrect-data10: OUTfromChCorrect data10
by (simp add: OUTfromChCorrect-def)

lemma OUTfromChCorrect-data11: OUTfromChCorrect data11
by (simp add: OUTfromChCorrect-def, metis (full-types) IN.simps(2) OUT.simps(2) OUT.simps(31) Un-empty-right Un-insert-left Un-insert-right insertI1)

lemma OUTfromChCorrect-data12: OUTfromChCorrect data12
by (simp add: OUTfromChCorrect-def)

lemma OUTfromChCorrect-data13: OUTfromChCorrect data13
by (simp add: OUTfromChCorrect-def)

lemma OUTfromChCorrect-data14: OUTfromChCorrect data14
by (metis OUTfromCh.simps(14) OUTfromChCorrect-def)

lemma OUTfromChCorrect-data15: OUTfromChCorrect data15
by (metis OUTfromCh.simps(15) OUTfromChCorrect-def)

lemma OUTfromChCorrect-data16: OUTfromChCorrect data16
by (metis OUTfromCh.simps(16) OUTfromChCorrect-def)

lemma OUTfromChCorrect-data17: OUTfromChCorrect data17
proof
  have data17 ∈ OUT sA71 ∧ data15 ∈ IN sA71
    by (metis IN.simps(19) OUT.simps(19) insertI1)
  thus ?thesis by (metis IN.simps(19) OUTfromCh.simps(17) OUTfromChCorrect-def)
qed

lemma OUTfromChCorrect-data18: OUTfromChCorrect data18
  by (simp add: OUTfromChCorrect-def, metis IN.simps(20) OUT.simps(20) insertI1)

lemma OUTfromChCorrect-data19: OUTfromChCorrect data19
  by (metis OUTfromCh.simps(19) OUTfromChCorrect-def)

lemma OUTfromChCorrect-data20: OUTfromChCorrect data20
  by (simp add: OUTfromChCorrect-def, metis IN.simps(20) OUT.simps(21) insertI1 insert-subset subset-insertI)

lemma OUTfromChCorrect-data21: OUTfromChCorrect data21
  by (simp add: OUTfromChCorrect-def, metis (full-types) IN.simps(22) OUT.simps(22) insertI1 insert-subset subset-insertI)

lemma OUTfromChCorrect-data22: OUTfromChCorrect data22
  by (simp add: OUTfromChCorrect-def, metis (full-types) IN.simps(23) OUT.simps(23) insertI1)

lemma OUTfromChCorrect-data23: OUTfromChCorrect data23
  by (simp add: OUTfromChCorrect-def, metis (full-types) IN.simps(9) OUT.simps(9) insert-subset subset-insertI)

lemma OUTfromChCorrect-data24: OUTfromChCorrect data24
  by (simp add: OUTfromChCorrect-def, metis IN.simps(9) OUT.simps(9) insertI1 insert-subset subset-insertI)

lemma OUTfromChCorrectSYSTEM-holds: OUTfromChCorrectSYSTEM
  by (simp add: OUTfromChCorrectSYSTEM-def, clarify, case-tac x, simp add: OUTfromChCorrect-data1, simp add: OUTfromChCorrect-data2, simp add: OUTfromChCorrect-data3, simp add: OUTfromChCorrect-data4, simp add: OUTfromChCorrect-data5, simp add: OUTfromChCorrect-data6, simp add: OUTfromChCorrect-data7, simp add: OUTfromChCorrect-data8, simp add: OUTfromChCorrect-data9, simp add: OUTfromChCorrect-data10, simp add: OUTfromChCorrect-data11, simp add: OUTfromChCorrect-data12, simp add: OUTfromChCorrect-data13, simp add: OUTfromChCorrect-data14, simp add: OUTfromChCorrect-data15, simp add: OUTfromChCorrect-data16, simp add: OUTfromChCorrect-data17, simp add: OUTfromChCorrect-data18, simp add: OUTfromChCorrect-data19, simp add: OUTfromChCorrect-data20, simp add: OUTfromChCorrect-data21, simp add: OUTfromChCorrect-data22, simp add: OUTfromChCorrect-data23, simp add: OUTfromChCorrect-data24)
lemma OUTfromVCorrect1-data1: OUTfromVCorrect1 data1
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data2: OUTfromVCorrect1 data2
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data3: OUTfromVCorrect1 data3
proof –
  have data3 ∈ OUT sA41 ∧ stA4 ∈ VAR sA41
    by (metis OUT.simps(17) VAR.simps(17) insertII)
  thus ?thesis by (metis OUTfromV.simps(3) OUTfromVCorrect1-def VAR.simps(17))
qed

lemma OUTfromVCorrect1-data4: OUTfromVCorrect1 data4
by (simp add: OUTfromVCorrect1-def, metis (full-types) OUT.simps(2) VAR.simps(2) insertII)

lemma OUTfromVCorrect1-data5: OUTfromVCorrect1 data5
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data6: OUTfromVCorrect1 data6
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data7: OUTfromVCorrect1 data7
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data8: OUTfromVCorrect1 data8
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data9: OUTfromVCorrect1 data9
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data10: OUTfromVCorrect1 data10
proof –
  have data10 ∈ OUT sA12 ∧ stA1 ∈ VAR sA12
    by (metis OUT.simps(11) VAR.simps(11) insertII)
  thus ?thesis by (metis OUT.simps(26) OUTfromV.simps(10) OUTfromVCorrect1-def VAR.simps(26) insertII)
qed

lemma OUTfromVCorrect1-data11: OUTfromVCorrect1 data11
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data12: OUTfromVCorrect1 data12
proof –
  have data12 ∈ OUT sA22 ∧ stA2 ∈ VAR sA22
    by (metis (full-types) OUT.simps(13) VAR.simps(13) insertII)
  thus ?thesis by (metis OUTfromV.simps(12) OUTfromVCorrect1-def VAR.simps(13))
lemma OUTfromVCorrect1-data13: OUTfromVCorrect1 data13
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data14: OUTfromVCorrect1 data14
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data15: OUTfromVCorrect1 data15
proof –
  have A6ch: data15 ∈ OUT sA6 ∧ stA6 ∈ VAR sA6
    by (metis OUT.simps(6) VAR.simps(6) insertI1)
  thus ?thesis by (simp add: OUTfromVCorrect1-def, metis A6ch)
qed

lemma OUTfromVCorrect1-data16: OUTfromVCorrect1 data16
proof –
  have A6ch: data16 ∈ OUT sA6 ∧ stA6 ∈ VAR sA6
    by (metis (full-types) OUT.simps(6) VAR.simps(6) insertCI)
  thus ?thesis by (simp add: OUTfromVCorrect1-def, metis A6ch)
qed

lemma OUTfromVCorrect1-data17: OUTfromVCorrect1 data17
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data18: OUTfromVCorrect1 data18
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data19: OUTfromVCorrect1 data19
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data20: OUTfromVCorrect1 data20
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data21: OUTfromVCorrect1 data21
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data22: OUTfromVCorrect1 data22
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data23: OUTfromVCorrect1 data23
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1-data24: OUTfromVCorrect1 data24
by (simp add: OUTfromVCorrect1-def)

lemma OUTfromVCorrect1SYSTEM-holds: OUTfromVCorrect1SYSTEM
by (simp add: OUTfromVCorrect1SYSTEM-def, clarify, case-tac x,
simp add: OUTfromVCorrect1-data1, simp add: OUTfromVCorrect1-data2,
simp add: OUTfromVCorrect1-data3, simp add: OUTfromVCorrect1-data4,
simp add: OUTfromVCorrect1-data5, simp add: OUTfromVCorrect1-data6,
simp add: OUTfromVCorrect1-data7, simp add: OUTfromVCorrect1-data8,
simp add: OUTfromVCorrect1-data9, simp add: OUTfromVCorrect1-data10,
simp add: OUTfromVCorrect1-data11, simp add: OUTfromVCorrect1-data12,
simp add: OUTfromVCorrect1-data13, simp add: OUTfromVCorrect1-data14,
simp add: OUTfromVCorrect1-data15, simp add: OUTfromVCorrect1-data16,
simp add: OUTfromVCorrect1-data17, simp add: OUTfromVCorrect1-data18,
simp add: OUTfromVCorrect1-data19, simp add: OUTfromVCorrect1-data20,
simp add: OUTfromVCorrect1-data21, simp add: OUTfromVCorrect1-data22,
simp add: OUTfromVCorrect1-data23, simp add: OUTfromVCorrect1-data24

lemma OUTfromVCorrect2SYSTEM: OUTfromVCorrect2SYSTEM
by (simp add: OUTfromVCorrect2SYSTEM-def, auto, case-tac x,
((simp add: OUTfromVCorrect2-def, auto, case-tac v, auto) |
(simp add: OUTfromVCorrect2-def)))+

lemma OUTfromV-VARto-holds:
OUTfromV-VARto
by (simp add: OUTfromV-VARto-def, auto, (case-tac x, auto), (case-tac v, auto))

lemma VARfromCorrectSYSTEM-holds:
VARfromCorrectSYSTEM
by (simp add: VARfromCorrectSYSTEM-def AbstrLevel0 AbstrLevel1)

lemma VARtoCorrectSYSTEM-holds:
VARtoCorrectSYSTEM
by (simp add: VARtoCorrectSYSTEM-def AbstrLevel0 AbstrLevel1)

lemma VARusefulSYSTEM-holds:
VARusefulSYSTEM
by (simp add: VARusefulSYSTEM-def, auto, case-tac v, auto)

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

lemma NOT-elementaryCompDD-sA1: ¬ elementaryCompDD sA1
proof –
  have outSetCorrelated data2 ∩ outSetCorrelated data10 = {}
  by (metis OUTfromV.simps(2) inf-bot-left outSetCorrelatedEmpty1)
  thus ?thesis by (simp add: elementaryCompDD-def)
qed

lemma NOT-elementaryCompDD-sA2: ¬ elementaryCompDD sA2
proof –
  have outSetCorrelated data5 ∩ outSetCorrelated data11 = {}
  by (metis OUTfromV.simps(5) inf-bot-right inf-commute outSetCorrelatedEmpty1)

39
thus \(?thesis\) by (simp add: elementaryCompDD-def)
qed

lemma \(\text{NOT-elementaryCompDD-sA3}: \neg \text{elementaryCompDD sA3}\)
proof –
  have outSetCorelated data6 \(\cap\) outSetCorelated data7 = {}
    by (metis OUTfromV.simps(7) inf-bot-right outSetCorelatedEmpty1)
  thus \(?thesis\) by (simp add: elementaryCompDD-def)
qed

lemma \(\text{NOT-elementaryCompDD-sA4}: \neg \text{elementaryCompDD sA4}\)
proof –
  have outSetCorelated data3 \(\cap\) outSetCorelated data8 = {}
    by (metis OUTfromV.simps(8) inf-bot-left inf-commute outSetCorelatedEmpty1)
  thus \(?thesis\) by (simp add: elementaryCompDD-def)
qed

lemma \(\text{elementaryCompDD-sA5}: \text{elementaryCompDD sA5}\)
by (simp add: elementaryCompDD-def)

lemma \(\text{elementaryCompDD-sA6}: \text{elementaryCompDD sA6}\)
proof –
  have oSet15: outSetCorelated data15 \(\neq\) {}
    by (simp add: outSetCorelated-def, auto)
  have oSet16: outSetCorelated data16 \(\neq\) {}
    by (simp add: outSetCorelated-def, auto)
  have outSetCorelated data15 \(\cap\) outSetCorelated data16 \(\neq\) {}
    by (simp add: outSetCorelated-def, auto)
  with oSet15 oSet16 show \(?thesis\) by (simp add: elementaryCompDD-def, auto)
qed

lemma \(\text{NOT-elementaryCompDD-sA7}: \neg \text{elementaryCompDD sA7}\)
proof –
  have outSetCorelated data17 \(\cap\) outSetCorelated data18 = {}
    by (metis (full-types) OUTfromV.simps(17) disjoint-iff-not-equal empty-iff outSetCorelatedEmpty1)
  thus \(?thesis\) by (simp add: elementaryCompDD-def)
qed

lemma \(\text{NOT-elementaryCompDD-sA8}: \neg \text{elementaryCompDD sA8}\)
proof –
  have outSetCorelated data20 \(\cap\) outSetCorelated data21 = {}
    by (metis OUTfromV.simps(21) inf-bot-right outSetCorelatedEmpty1)
  thus \(?thesis\) by (simp add: elementaryCompDD-def)
qed

lemma \(\text{NOT-elementaryCompDD-sA9}: \neg \text{elementaryCompDD sA9}\)
proof –
have \(\text{outSetCorelated data}^{23} \cap \text{outSetCorelated data}^{24} = \{\}\)
by (metis (full-types) \text{OUTfromV}.\text{.simps}(23) \text{disjoint-iff-not-equal empty-iff out-SetCorelatedEmpty1))

thus \(?\text{thesis}\) by (simp add: \text{elementaryCompDD-def})

qed

— On the abstraction level 1 all components are elementary

\begin{itemize}
  \item \textbf{lemma} \text{elementaryCompDD-sA11}: \text{elementaryCompDD sA11} by (simp add: \text{elementaryCompDD-def})
  \item \textbf{lemma} \text{elementaryCompDD-sA12}: \text{elementaryCompDD sA12} by (simp add: \text{elementaryCompDD-def})
  \item \textbf{lemma} \text{elementaryCompDD-sA21}: \text{elementaryCompDD sA21} by (simp add: \text{elementaryCompDD-def})
  \item \textbf{lemma} \text{elementaryCompDD-sA22}: \text{elementaryCompDD sA22} proof
    \begin{itemize}
      \item \textbf{have} \text{outSetCorelated data}^{4} \neq \{\}
        by (simp add: \text{outSetCorelated-def, auto})
      \item \textbf{have} \text{outSetCorelated data}^{12} \neq \{\}
        by (simp add: \text{outSetCorelated-def, auto})
      \item \textbf{have} \text{outSetCorelated data}^{4} \cap \text{outSetCorelated data}^{12} \neq \{\}
        by (simp add: \text{outSetCorelated-def, auto})
      \item \textbf{with} \text{outSet}^{4} \text{outSet}^{12} \textbf{show} \(?\text{thesis}\)
        by (simp add: \text{elementaryCompDD-def, auto})
    \end{itemize}
  \item \textbf{qed}
  \item \textbf{lemma} \text{elementaryCompDD-sA23}: \text{elementaryCompDD sA23} by (simp add: \text{elementaryCompDD-def})
  \item \textbf{lemma} \text{elementaryCompDD-sA31}: \text{elementaryCompDD sA31} by (simp add: \text{elementaryCompDD-def})
  \item \textbf{lemma} \text{elementaryCompDD-sA32}: \text{elementaryCompDD sA32} by (simp add: \text{elementaryCompDD-def})
  \item \textbf{lemma} \text{elementaryCompDD-sA41}: \text{elementaryCompDD sA41} by (simp add: \text{elementaryCompDD-def})
  \item \textbf{lemma} \text{elementaryCompDD-sA42}: \text{elementaryCompDD sA42} by (simp add: \text{elementaryCompDD-def})
  \item \textbf{lemma} \text{elementaryCompDD-sA71}: \text{elementaryCompDD sA71} by (simp add: \text{elementaryCompDD-def})
  \item \textbf{lemma} \text{elementaryCompDD-sA72}: \text{elementaryCompDD sA72}
\end{itemize}
by (simp add: elementaryCompDD-def)

lemma elementaryCompDD-sA81: elementaryCompDD sA81
by (simp add: elementaryCompDD-def)

lemma elementaryCompDD-sA82: elementaryCompDD sA82
by (simp add: elementaryCompDD-def)

lemma elementaryCompDD-sA91: elementaryCompDD sA91
by (simp add: elementaryCompDD-def)

lemma elementaryCompDD-sA92: elementaryCompDD sA92
by (simp add: elementaryCompDD-def)

lemma elementaryCompDD-sA93: elementaryCompDD sA93
by (simp add: elementaryCompDD-def)

4.4 Source components
— Abstraction level 0

lemma A5-NotDSource-level0: isNotDSource level0 sA5
by (simp add: isNotDSource-def, auto, case-tac Z, auto)

lemma DSourcesA1-L0: DSources level0 sA1 = {}
by (simp add: DSources-def, auto, case-tac x, auto)

lemma DSourcesA2-L0: DSources level0 sA2 = {sA1, sA4}
by (simp add: DSources-def AbstrLevel0, auto)

lemma DSourcesA3-L0: DSources level0 sA3 = {sA2}
by (simp add: DSources-def AbstrLevel0, auto)

lemma DSourcesA4-L0: DSources level0 sA4 = {sA3}
by (simp add: DSources-def AbstrLevel0, auto)

lemma DSourcesA5-L0: DSources level0 sA5 = {sA4}
by (simp add: DSources-def AbstrLevel0, auto)

lemma DSourcesA6-L0: DSources level0 sA6 = {}
by (simp add: DSources-def, auto, case-tac x, auto)

lemma DSourcesA7-L0: DSources level0 sA7 = {sA6}
by (simp add: DSources-def AbstrLevel0, auto)

lemma DSourcesA8-L0: DSources level0 sA8 = {sA7, sA9}
by (simp add: DSources-def AbstrLevel0, force)

lemma DSourcesA9-L0: DSources level0 sA9 = {sA8}
by (simp add: DSources-def AbstrLevel0, auto)

lemma A1-DAcc-level0: \( D\text{Acc \ level0} \ sA1 = \{ sA2 \} \)
by (simp add: DAcc-def AbstrLevel0, auto)

lemma A2-DAcc-level0: \( D\text{Acc \ level0} \ sA2 = \{ sA3 \} \)
by (simp add: DAcc-def AbstrLevel0, force)

lemma A3-DAcc-level0: \( D\text{Acc \ level0} \ sA3 = \{ sA4 \} \)
by (simp add: DAcc-def AbstrLevel0, auto)

lemma A4-DAcc-level0: \( D\text{Acc \ level0} \ sA4 = \{ sA2, sA5 \} \)
by (simp add: DAcc-def AbstrLevel0, auto)

lemma A5-DAcc-level0: \( D\text{Acc \ level0} \ sA5 = \{ \} \)
by (simp add: DAcc-def AbstrLevel0, auto)

lemma A6-DAcc-level0: \( D\text{Acc \ level0} \ sA6 = \{ sA7 \} \)
by (simp add: DAcc-def AbstrLevel0, auto)

lemma A7-DAcc-level0: \( D\text{Acc \ level0} \ sA7 = \{ sA8 \} \)
by (simp add: DAcc-def AbstrLevel0, auto)

lemma A8-DAcc-level0: \( D\text{Acc \ level0} \ sA8 = \{ sA9 \} \)
by (simp add: DAcc-def AbstrLevel0, auto)

lemma A9-DAcc-level0: \( D\text{Acc \ level0} \ sA9 = \{ sA8 \} \)
by (simp add: DAcc-def AbstrLevel0, force)

lemma A8-NSources:
\( \forall C \in (\text{AbstrLevel \ level0}). \ (C \neq sA9 \land C \neq sA8 \rightarrow sA8 \notin (\text{Sources \ level0} \ C)) \)
by (metis A8-DAcc-level0 A9-DAcc-level0 singleDSourceLoop)

lemma A9-NSources:
\( \forall C \in (\text{AbstrLevel \ level0}). \ (C \neq sA9 \land C \neq sA8 \rightarrow sA9 \notin (\text{Sources \ level0} \ C)) \)
by (metis A8-DAcc-level0 A9-DAcc-level0 singleDSourceLoop)

lemma A7-Acc:
\( (\text{Acc \ level0} \ sA7) = \{ sA8, sA9 \} \)
by (metis A7-DAcc-level0 A8-DAcc-level0 A9-DAcc-level0 AccDef AccSigleLoop insert-commute)

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

lemma A5-Acc: \( (\text{Acc \ level0} \ sA5) = \{ \} \)
by (metis A5-NotDSource-level0 isNotDSource-EmptyAcc)
lemma A6-Acc:
\[(\text{Acc level}_0 \ sA6) = \{sA7, sA8, sA9\}\]
proof
  have daA6: \(\text{DAcc level}_0 \ sA6 = \{sA7\}\) by (rule A6-DAcc-level0)
  hence \(\bigcup S \in (\text{DAcc level}_0 \ sA6). (\text{Acc level}_0 S) = (\text{Acc level}_0 sA7)\) by simp
  hence aA6: \(\bigcup S \in (\text{DAcc level}_0 \ sA6). (\text{Acc level}_0 S) = \{sA8, sA9\}\) by (simp add: A7-Acc)
  have (\(\text{Acc level}_0 sA6\)) = (\(\text{DAcc level}_0 sA6\)) \(\cup \bigcup S \in (\text{DAcc level}_0 sA6). (\text{Acc level}_0 S)\) by simp
  hence aA6: \(\bigcup S \in (\text{DAcc level}_0 sA6). (\text{Acc level}_0 S) = \{sA8, sA9\}\)
proof
  have DA2level0:sA2 \(\in (\text{AbstrLevel level}_0)\) by (simp add: AbstrLevel0)
  have sgA5:sA5 \(\notin \text{Sources level}_0 sA2\)
    by (metis A5-NotDSource-level0 DSourc Level NoDSourceNoSource allNotDSource-NotSource isNotSource-Sources)
  from DA2level0 have sgA6:sA6 \(\notin \text{Sources level}_0 sA2\) by (simp add: A6-NSources)
  from DA2level0 have sgA7:sA7 \(\notin \text{Sources level}_0 sA2\) by (simp add: A7-NSources)
  from DA2level0 have sgA8:sA8 \(\notin \text{Sources level}_0 sA2\) by (simp add: A8-NSources)
  from DA2level0 have sgA9:sA9 \(\notin \text{Sources level}_0 sA2\) by (simp add: A9-NSources)
  have Sources level0 sA2 \(\subseteq \{sA1, sA2, sA3, sA4, sA5, sA6, sA7, sA8, sA9\}\)
    by (metis AbstrLevel0 SourcesLevelX)
  with sgA5 sgA6 sgA7 sgA8 sgA9 show Sources level0 sA2 \(\subseteq \{sA1, sA2, sA3, sA4, sA5, sA6, sA7, sA8, sA9\}\)
    by blast
qed

next
show \(\{sA1, sA2, sA3, sA4\} \subseteq \text{Sources level}_0 sA2\)
proof
  have dsA4: \(\{sA3\} \subseteq \text{Sources level}_0 sA2\)
    by (metis DSourc-Sources DSourcesA2-L0 DSourcesA4-L0 Sources-DSourc insertII insert-commute subset-trans)
  have \(\{sA2\} \subseteq \text{Sources level}_0 sA2\)
    by (metis DSourc-Sources DSourcesA2-L0 DSourcesA3-L0 DSourcesA4-L0 Sources-DSourc insertII
      insert-commute subset-trans)
with \( dsA4 \) show \( \{ sA1, sA2, sA3, sA4 \} \subseteq Sources level0 sA2 \)
by (metis DSourcesA2-L0 Sources-DSources insert-subset)
qed

lemma SourcesA3-L0: Sources level0 sA3 = \( \{ sA1, sA2, sA3, sA4 \} \)
proof
show Sources level0 sA3 \( \subseteq \) \( \{ sA1, sA2, sA3, sA4 \} \)
proof
have \( a2 : Sources level0 sA2 = \{ sA1, sA2, sA3, sA4 \} \) by (simp add: SourcesA2-L0)
have \( \{ sA2 \} \subseteq DSources level0 sA3 \) by (simp add: DSourcesA3-L0)
with \( a2 \) show Sources level0 sA3 \( \subseteq \) \( \{ sA1, sA2, sA3, sA4 \} \)
by (metis DSourse-Sources DSourcesA2-L0 DSourcesA4-L0 insertI1 insert-commute subset-trans)
qed

next
show \( \{ sA1, sA2, sA3, sA4 \} \subseteq Sources level0 sA3 \)
by (metis (full-types) DSource-Sources DSourcesA3-L0 SourcesA2-L0 insertI1)
qed

lemma SourcesA4-L0: Sources level0 sA4 = \( \{ sA1, sA2, sA3, sA4 \} \)
proof
have \( A3s : Sources level0 sA3 = \{ sA1, sA2, sA3, sA4 \} \) by (rule SourcesA3-L0)
have Sources level0 sA4 = \( \{ sA3 \} \cup Sources level0 sA3 \)
by (metis DSourcesA4-L0 Sources-singleDSource)
with \( A3s \) show \(?thesis\) by auto
qed

lemma SourcesA5-L0: Sources level0 sA5 = \( \{ sA1, sA2, sA3, sA4 \} \)
proof
have \( A4s : Sources level0 sA4 = \{ sA1, sA2, sA3, sA4 \} \) by (rule SourcesA4-L0)
have Sources level0 sA5 = \( \{ sA4 \} \cup Sources level0 sA4 \)
by (metis DSourcesA5-L0 Sources-singleDSource)
with \( A4s \) show \(?thesis\) by auto
qed

lemma SourcesA6-L0: Sources level0 sA6 = \( \{} \)
by (simp add: DSourcesA6-L0 DSourcesEmptySources)

lemma SourcesA7-L0: Sources level0 sA7 = \( \{ sA6 \} \)
by (metis DSourcesA7-L0 SourcesA6-L0 SourcesEmptyDSources SourcesOnlyDSources singleton-iff)

lemma SourcesA8-L0: Sources level0 sA8 = \( \{ sA6, sA7, sA8, sA9 \} \)
proof
have \( dA8 : DSources level0 sA8 = \{ sA7, sA9 \} \) by (rule DSourcesA8-L0)
have \( dA9 : DSources level0 sA9 = \{ sA8 \} \) by (rule DSourcesA9-L0)
have \( (Sources level0 sA8) = (DSources level0 sA8) \cup (\bigcup S \in (DSources level0 sA8)) \)

45
\( sA8 \). (Sources level0 S))
   by (rule SourcesDef)
   hence sourcesA8:(Sources level0 sA8) = (\{sA7, sA9, sA6\} \cup (Sources level0 sA9))
      by (simp add: DSourcesA8-L0 SourcesA7-L0, auto)
   have (Sources level0 sA9) = (DSources level0 sA9) \cup (\bigcup S \in (DSources level0 sA9). (Sources level0 S))
      by (rule SourcesDef)
   hence (Sources level0 sA9) = (\{sA8\} \cup (Sources level0 sA8))
      by (simp add: DSourcesA9-L0)
   with sourcesA8 have (Sources level0 sA8) = \{sA7, sA9, sA6\} \cup \{sA8\} \cup \{sA8, sA9\}
      by (metis SourcesA8-L0 Un-insert-right insert-absorb2 insert-is-Un)
   thus ?thesis by auto
qed

lemma SourcesA9-L0: Sources level0 sA9 = \{ sA6, sA7, sA8, sA9 \}
proof –
   have (Sources level0 sA9) = (DSources level0 sA9) \cup (\bigcup S \in (DSources level0 sA9). (Sources level0 S))
      by (rule SourcesDef)
   hence sourcesA9:(Sources level0 sA9) = (\{sA8\} \cup (Sources level0 sA8))
      by (simp add: DSourcesA9-L0)
   thus ?thesis by (metis SourcesA8-L0 Un-insert-right insert-absorb2 insert-is-Un)
qed

— Abstraction level 1

lemma A12-NotSource-level1: isNotDSource level1 sA12
by (simp add: isNotDSource-def, auto, case-tac Z, auto)
lemma A21-NotSource-level1: isNotDSource level1 sA21
by (simp add: isNotDSource-def, auto, case-tac Z, auto)
lemma A5-NotSource-level1: isNotDSource level1 sA5
by (simp add: isNotDSource-def, auto, case-tac Z, auto)
lemma A92-NotSource-level1: isNotDSource level1 sA92
by (simp add: isNotDSource-def, auto, case-tac Z, auto)
lemma A93-NotSource-level1: isNotDSource level1 sA93
by (simp add: isNotDSource-def, auto, case-tac Z, auto)
lemma A11-DAcc-level1: DAcc level1 sA11 = \{ sA21, sA22, sA23 \}
by (simp add: DAcc-def AbstrLevel1, auto)
lemma A12-DAcc-level1: \( \text{DAcc level1 } sA12 = {} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A21-DAcc-level1: \( \text{DAcc level1 } sA21 = {} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A22-DAcc-level1: \( \text{DAcc level1 } sA22 = \{sA31\} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A23-DAcc-level1: \( \text{DAcc level1 } sA23 = \{sA32\} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A31-DAcc-level1: \( \text{DAcc level1 } sA31 = \{sA41\} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A32-DAcc-level1: \( \text{DAcc level1 } sA32 = \{sA41\} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A41-DAcc-level1: \( \text{DAcc level1 } sA41 = \{sA22\} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A42-DAcc-level1: \( \text{DAcc level1 } sA42 = \{sA5\} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A5-DAcc-level1: \( \text{DAcc level1 } sA5 = {} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A6-DAcc-level1: \( \text{DAcc level1 } sA6 = \{sA71, sA72\} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A71-DAcc-level1: \( \text{DAcc level1 } sA71 = \{sA81\} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A72-DAcc-level1: \( \text{DAcc level1 } sA72 = \{sA82\} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A81-DAcc-level1: \( \text{DAcc level1 } sA81 = \{sA91, sA92\} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A82-DAcc-level1: \( \text{DAcc level1 } sA82 = \{sA93\} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A91-DAcc-level1: \( \text{DAcc level1 } sA91 = \{sA81\} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A92-DAcc-level1: \( \text{DAcc level1 } sA92 = {} \)
by (simp add: DAcc-def AbstrLevel1, auto)

lemma A93-DAcc-level1: \( \text{DAcc level1 } sA93 = {} \)
by (simp add: DAcc-def AbstrLevel1, auto)

**Lemma A42-NSources-L1:**
\[
\forall C \in (\text{AbstrLevel level1}). \ C \neq sA5 \rightarrow sA42 \notin (\text{Sources level1 C})
\]
by (metis A42-DAcc-level1 A5-NotSource-level1 singleDSourceEmpty4isNotSource)

**Lemma A5-NotSourceSet-level1 :**
\[
\forall C \in (\text{AbstrLevel level1}). \ sA5 \notin (\text{Sources level1 C})
\]
by (metis A5-NotSource-level1 isNotSource-Sources)

**Lemma A92-NotSourceSet-level1 :**
\[
\forall C \in (\text{AbstrLevel level1}). \ sA92 \notin (\text{Sources level1 C})
\]
by (metis A92-NotSource-level1 isNotSource-Sources)

**Lemma A93-NotSourceSet-level1 :**
\[
\forall C \in (\text{AbstrLevel level1}). \ sA93 \notin (\text{Sources level1 C})
\]
by (metis A93-NotSource-level1 isNotSource-Sources)

**Lemma DSourcesA11-L1:**
\[
\text{DSources level1 sA11} = \{\}
\]
by (simp add: DSources-def, auto, case_tac x, auto)

**Lemma DSourcesA12-L1:**
\[
\text{DSources level1 sA12} = \{\}
\]
by (simp add: DSources-def AbstrLevel1, auto)

**Lemma DSourcesA21-L1:**
\[
\text{DSources level1 sA21} = \{sA11\}
\]
by (simp add: DSources-def AbstrLevel1, auto)

**Lemma DSourcesA22-L1:**
\[
\text{DSources level1 sA22} = \{sA11, sA41\}
\]
by (simp add: DSources-def AbstrLevel1, auto)

**Lemma DSourcesA23-L1:**
\[
\text{DSources level1 sA23} = \{sA11\}
\]
by (simp add: DSources-def AbstrLevel1, auto)

**Lemma DSourcesA31-L1:**
\[
\text{DSources level1 sA31} = \{sA22\}
\]
by (simp add: DSources-def AbstrLevel1, auto)

**Lemma DSourcesA32-L1:**
\[
\text{DSources level1 sA32} = \{sA23\}
\]
by (simp add: DSources-def AbstrLevel1, auto)

**Lemma DSourcesA41-L1:**
\[
\text{DSources level1 sA41} = \{sA31, sA32\}
\]
by (simp add: DSources-def AbstrLevel1, auto)

**Lemma DSourcesA42-L1:**
\[
\text{DSources level1 sA42} = \{\}
\]
by (simp add: DSources-def AbstrLevel1, auto)

**Lemma DSourcesA5-L1:**
\[
\text{DSources level1 sA5} = \{sA42\}
\]
by (simp add: DSources-def AbstrLevel1, auto)
lemma DSourcesA6-L1: DSources level1 sA6 = {}
by (simp add: DSources-def AbstrLevel1, auto)

lemma DSourcesA71-L1: DSources level1 sA71 = { sA6 }
by (simp add: DSources-def AbstrLevel1, auto)

lemma DSourcesA72-L1: DSources level1 sA72 = { sA6 }
by (simp add: DSources-def AbstrLevel1, auto)

lemma DSourcesA81-L1: DSources level1 sA81 = { sA71, sA91 }
by (simp add: DSources-def AbstrLevel1, auto)

lemma DSourcesA82-L1: DSources level1 sA82 = { sA72 }
by (simp add: DSources-def AbstrLevel1, auto)

lemma DSourcesA91-L1: DSources level1 sA91 = { sA81 }
by (simp add: DSources-def AbstrLevel1, auto)

lemma DSourcesA92-L1: DSources level1 sA92 = { sA82 }
by (simp add: DSources-def AbstrLevel1, auto)

lemma DSourcesA93-L1: DSources level1 sA93 = { sA82 }
by (simp add: DSources-def AbstrLevel1, auto)

lemma A82-Acc: (Acc level1 sA82) = {sA93}
by (metis A82-DAcc-level1 A93-NotSource-level1 singleDSourceEmpty-Acc)

lemma A82-NSources-L1:
\forall C \in (AbstrLevel level1). \ (C \neq sA93 \longrightarrow sA82 \notin (Sources level1 C))
by (metis A82-Acc Acc-Sources insert-iff singleton-iff)

lemma A72-Acc: (Acc level1 sA72) = {sA82, sA93}
proof
  have daA72: DAcc level1 sA72 = { sA82 } by (rule A72-DAcc-level1)
  hence (\bigcup S \in (DAcc level1 sA72). (Acc level1 S)) = (Acc level1 sA82) by simp
  hence aA72:(\bigcup S \in (DAcc level1 sA72). (Acc level1 S)) = { sA93 } by (simp add: A82-Acc)
  have (Acc level1 sA72) = (DAcc level1 sA72) \cup (\bigcup S \in (DAcc level1 sA72).
  (Acc level1 S))
  by (rule AccDef)
  with daA72 aA72 show ?thesis by auto
qed

lemma A72-NSources-L1:
\forall C \in (AbstrLevel level1). \ (C \neq sA93 \land C \neq sA82 \longrightarrow sA72 \notin (Sources level1 C))
by (metis A72-Acc Acc-Sources insert-iff singleton-iff)

lemma A92-Acc: (Acc level1 sA92) = {}
by \((\text{metis A92-NotSource-level1 isNotDSource-EmptyAcc})\)

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

**Lemma A91-Acc:** \((\text{Acc level1} sA91) = \{ sA81, sA91, sA92 \} \)
**Proof**
- have \(\text{da91}: \text{DAcc level1} sA91 = \{ sA81 \} \)
  by \((\text{rule A91-DAcc-level1})\)
- hence \(\text{a91}:(\bigcup S \in (\text{DAcc level1} sA91). (\text{Acc level1} S)) = (\text{Acc level1} sA81) \)
  by \(\text{simp}\)
- have \(\text{acc91}: (\text{Acc level1} sA91) = (\text{DAcc level1} sA91) \cup (\bigcup S \in (\text{DAcc level1} sA91). (\text{Acc level1} S)) = (\text{Acc level1} sA81) \)
  by \(\text{simp}\)
- have \(\text{da81}: \text{DAcc level1} sA81 = \{ sA91, sA92 \} \)
  by \((\text{rule A81-DAcc-level1})\)
- hence \(\text{acc81}: (\text{Acc level1} sA81) = (\bigcup S \in (\text{DAcc level1} sA81). (\text{Acc level1} S)) \cup (\text{Acc level1} sA91) \)
  by \(\text{auto}\)
- hence \(\text{acc91}: (\text{Acc level1} sA91) = (\text{DAcc level1} sA91) \cup (\bigcup S \in (\text{DAcc level1} sA91). (\text{Acc level1} S)) \cup (\text{Acc level1} sA91) \)
  by \(\text{simp}\)
- have \(\text{acc91}: (\text{Acc level1} sA91) = \{ sA81 \} \cup (\text{Acc level1} sA91) \)
  by \(\text{simp}\)
- thus ?thesis by \(\text{auto}\)
qed

**Lemma A91-NSources-L1:**
\[ \forall C \in (\text{AbstrLevel level1}). (C \neq sA92 \land C \neq sA91 \land C \neq sA81 \rightarrow sA91 \notin (\text{Sources level1} C)) \]
**Proof**
- have \(\forall C \in (\text{AbstrLevel level1}). (C \neq sA92 \land C \neq sA91 \land C \neq sA81 \rightarrow (C \notin (\text{Acc level1} sA91))) \)
  by \((\text{metis A91-Acc insert-iff singleton-iff})\)
- thus ?thesis by \(\text{metis Acc-SourcesNOT}\)
qed

**Lemma A81-Acc:** \((\text{Acc level1} sA81) = \{ sA81, sA91, sA92 \} \)
**Proof**
- have \(\text{da91}: \text{DAcc level1} sA91 = \{ sA81 \} \)
  by \((\text{rule A91-DAcc-level1})\)
- hence \(\text{a91}:(\bigcup S \in (\text{DAcc level1} sA91). (\text{Acc level1} S)) = (\text{Acc level1} sA81) \)
  by \(\text{simp}\)
- have \(\text{acc91}: (\text{Acc level1} sA91) = (\text{DAcc level1} sA91) \cup (\bigcup S \in (\text{DAcc level1} sA91). (\text{Acc level1} S)) \cup (\text{Acc level1} sA91) \)
  by \(\text{simp}\)
- hence \(\text{acc91}: (\text{Acc level1} sA91) = \{ sA81 \} \cup (\text{Acc level1} sA91) \)
  by \(\text{simp}\)
- have \(\text{da81}: \text{DAcc level1} sA81 = \{ sA91, sA92 \} \)
  by \((\text{rule A81-DAcc-level1})\)
hence \( \bigcup S \in \left( \text{DAcc level1 } sA81 \right) \) \((\text{Acc level1 } S) = \left( \text{Acc level1 } sA92 \right) \cup \left( \text{Acc level1 } sA91 \right) \) by auto

have \((\text{Acc level1 } sA81) = \left( \text{DAcc level1 } sA81 \right) \cup \left( \bigcup S \in \left( \text{DAcc level1 } sA81 \right) \right) \) \((\text{Acc level1 } S)\) by (rule AccDef)

with \( \text{da81 a81} \) have \( \text{acc81} \): \((\text{Acc level1 } sA81) = \left( \text{DAcc level1 } sA81 \right) \cup \left( \bigcup S \in \left( \text{DAcc level1 } sA81 \right) \right) \) \((\text{Acc level1 } S)\)

by (metis A92-Acc sup-bot.left-neutral)

from \( \text{acc81 acc91} \) have \((\text{Acc level1 } sA81) = \left\{ sA91, sA92 \right\} \cup \left\{ sA81 \right\} \cup \left\{ sA81, sA91 \right\} \)

by (metis AccLoop)

thus \(?\)thesis by auto

qed

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

proof –

have \( \forall C \in \left( \text{AbstrLevel level1} \right) . \left( C \neq sA92 \land C \neq sA91 \land C \neq sA81 \rightarrow (C \notin \left( \text{Acc level1 } sA81 \right)) \right) \)

by (metis \( \text{A81-Acc insert-iff singleton-iff} \))

thus \(?\)thesis by (metis \( \text{Acc-SourcesNOT} \))

qed

lemma \( \text{A71-Acc} \): \((\text{Acc level1 } sA71) = \left\{ sA81, sA91, sA92 \right\} \)

proof –

have \( \text{daA6 \text{DAcc level1 } sA71} = \left\{ sA81 \right\} \) by (rule \( \text{A6-DAcc-level1} \))

hence \((\text{Acc level1 } sA71) = \left( \text{DAcc level1 } sA71 \right) \cup \left( \bigcup S \in \left( \text{DAcc level1 } sA71 \right) \right) \) \((\text{Acc level1 } S)\)

by simp

with \( \text{daA6 a71} \) show \(?\)thesis by (metis \( \text{A91-Acc A91-DAcc-level1 AccDef} \))

qed

lemma \( \text{A71-NSources-L1} \):
\( \forall C \in \left( \text{AbstrLevel level1} \right) . \left( C \neq sA92 \land C \neq sA91 \land C \neq sA81 \rightarrow \text{sA81} \notin \left( \text{Sources level1 } C \right) \right) \)

proof –

have \( \forall C \in \left( \text{AbstrLevel level1} \right) . \left( C \neq sA92 \land C \neq sA91 \land C \neq sA81 \rightarrow (C \notin \left( \text{Acc level1 } sA71 \right)) \right) \)

by (metis \( \text{A71-Acc insert-iff singleton-iff} \))

thus \(?\)thesis by (metis \( \text{Acc-SourcesNOT} \))

qed

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

proof –

have \( \text{daA6 \text{DAcc level1 } sA6} = \left\{ sA71, sA72 \right\} \) by (rule \( \text{A6-DAcc-level1} \))

hence \((\bigcup S \in \left( \text{DAcc level1 } sA6 \right) . \left( \text{Acc level1 } S \right) = \left( \text{Acc level1 } sA71 \right) \cup \left( \text{Acc level1 } sA72 \right) \) by simp
hence $\text{A6} : (\bigcup S \in (\text{DAcc level1 sA6}). (\text{Acc level1 } S)) = \{\text{sA81, sA91, sA92}\} \
\cup \{\text{sA82, sA93}\}$
  by (simp add: A71-Acc A72-Acc)
have $(\text{Acc level1 sA6}) = (\text{DAcc level1 sA6}) \cup (\bigcup S \in (\text{DAcc level1 sA6}). (\text{Acc level1 } S))$
  by (rule AccDef)
with $\text{daA6 aA6}$ show $?\text{thesis}$ by auto
qed

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

lemma A6-NSources-L1:
  $\forall C \in (\text{AbstrLevel level1}). (C \neq \text{sA93} \land C \neq \text{sA92} \land C \neq \text{sA91} \land C \neq \text{sA82} \land C \neq \text{sA81} \land C \neq \text{sA72} \land C \neq \text{sA71} \rightarrow sA6 \notin (\text{Sources level1 C}))$
proof
  have $\forall C \in (\text{AbstrLevel level1})$.
  $(C \neq \text{sA93} \land C \neq \text{sA92} \land C \neq \text{sA91} \land C \neq \text{sA82} \land C \neq \text{sA81} \land C \neq \text{sA72} \land C \neq \text{sA71} \rightarrow (C \notin (\text{Acc level1 sA6})))$
    by (metis A6-Acc-L1 empty-iff insert-iff)
  thus $?\text{thesis}$ by (metis Acc-SourcesNOT)
qed

lemma A5-Acc-L1: $(\text{Acc level1 sA5}) = \{}$
by (metis A5-NotSource-level1 isNotDSource-EmptyAcc)

lemma SourcesA11-L1: Sources level1 sA11 = $\{}$
by (simp add: DSourcesA11-L1 DSourcesEmptySources)

lemma SourcesA12-L1: Sources level1 sA12 = $\{}$
by (simp add: DSourcesA12-L1 DSourcesEmptySources)

lemma SourcesA21-L1: Sources level1 sA21 = $\{sA11\}$
by (simp add: DSourcesA21-L1 Sources-singleDSource)

lemma SourcesA22-L1: Sources level1 sA22 = $\{sA11, sA22, sA23, sA31, sA32, sA41\}$
proof
  show Sources level1 sA22 $\subseteq \{sA11, sA22, sA23, sA31, sA32, sA41\}$
  proof
    have $A2level1 : sA22 \in (\text{AbstrLevel level1})$ by (simp add: AbstrLevel1)
    from $A2level1$ have $sgA42 : sA42 \notin $ Sources level1 sA22 by (metis A42-NSources-L1 CSet.distinct(347))
    have $sgA5 : sA5 \notin $ Sources level1 sA22
  qed

52
by (metis A5-NotSource-level1 Acc-Sources all-not-in-conv isNotDSOURCE-EmptyAcc)

have sgA12: sA12 ∉ Sources level1 sA22 by (metis A12-NotSource-level1  A2level1 isNotSource-Sources)
  have sgA21: sA21 ∉ Sources level1 sA22 by (metis A21-NotSource-level1  DACC-DSOURCES NOT NDSourceExistsDSOURCE empty-iff isNotDSource-EmptyDACC)
  from A2level1 have sgA6: sA6 ∉ Sources level1 sA22 by (simp add: A6-NSOURCES-L1)
  from A2level1 have sgA71: sA71 ∉ Sources level1 sA22 by (simp add: A71-NSOURCES-L1)
  from A2level1 have sgA72: sA72 ∉ Sources level1 sA22 by (simp add: A72-NSOURCES-L1)
  from A2level1 have sgA81: sA81 ∉ Sources level1 sA22 by (simp add: A81-NSOURCES-L1)
  from A2level1 have sgA82: sA82 ∉ Sources level1 sA22 by (simp add: A82-NSOURCES-L1)
  from A2level1 have sgA91: sA91 ∉ Sources level1 sA22 by (simp add: A91-NSOURCES-L1)
  from A2level1 have sgA92: sA92 ∉ Sources level1 sA22 by (simp add: A92-NSOURCES-L1)
  from A2level1 have sgA93: sA93 ∉ Sources level1 sA22 by (metis A93-NotSourceSet-level1)

have Sources level1 sA22 ⊆ {sA11, sA12, sA21, sA22, sA23, sA31, sA32, sA41, sA42, sA5, sA6, sA71, sA72, sA81, sA82, sA91, sA92, sA93} by (metis AbstrLevel1 SourcesLevelX)

with sgA5 sgA12 sgA21 sgA42 sgA6 sgA71 sgA72 sgA81 sgA82 sgA91 sgA92 sgA93 show 
  Sources level1 sA22 ⊆ {sA11, sA22, sA23, sA31, sA32, sA41}
  by auto
qed

next

show {sA11, sA22, sA23, sA31, sA32, sA41} ⊆ Sources level1 sA22

proof –

  have sDef: (Sources level1 sA22) = (DSOURCES level1 sA22) ∪ (⋃ S ∈ (DSOURCES level1 sA22). (Sources level1 S))
    by (rule SourcesDef)

  have A11s: sA11 ∈ Sources level1 sA22 by (metis DSOURCESIsSource DSOURCESA22-L1 insertII)

  have A41s: sA41 ∈ Sources level1 sA22 by (metis DSOURCESIsSource DSOURCESA22-L1 insertCI)

  have A31s: sA31 ∈ Sources level1 sA22 by (metis DSOURCESIsSource DSOURCESA41-L1 SourcesTrans insertCI)

  have A32s: sA32 ∈ Sources level1 sA22 by (metis A32-DAcc-level1 A41s DACC-DSOURCES NOT DSOURCEDsSource insertII)

  have A23s: sA23 ∈ Sources level1 sA22 by (metis A32s DSOURCEDsSource DSOURCESA32-L1 insertII)

  have A22s: sA22 ∈ Sources level1 sA22 by (metis A31s DSOURCEDsSource 5
lemma SourcesA23-L1: Sources level1 sA23 = \{sA11\} 
by (simp add: DSourcesA23-L1 SourcesA11-L1 Sources-singleDSource)

lemma SourcesA31-L1: Sources level1 sA31 = \{sA11, sA22, sA23, sA31, sA32, sA41\}  
by (metis DSourcesA31-L1 SourcesA22-L1 Sources-singleDSource Un-insert-right insert-absorb2 insert-is-Un)

lemma SourcesA32-L1: Sources level1 sA32 = \{sA11, sA23\}  
by (metis DSourcesA32-L1 SourcesA23-L1 Sources-singleDSource Un-insert-right insert-is-Un)

lemma SourcesA41-L1: Sources level1 sA41 = \{sA11, sA22, sA23, sA31, sA32, sA41\}  
by (metis DSourcesA41-L1 SourcesA31-L1 SourcesA32-L1 Sources-2DSources Un-absorb Un-commute Un-insert-left)

lemma SourcesA42-L1: Sources level1 sA42 = \{}  
by (simp add: DSourcesA42-L1 DSourcesEmptySources)

lemma SourcesA5-L1: Sources level1 sA5 = \{sA42\}  
by (simp add: DSourcesA5-L1 SourcesA42-L1 Sources-singleDSource)

lemma SourcesA6-L1: Sources level1 sA6 = \{}  
by (simp add: DSourcesA6-L1 DSourcesEmptySources)

lemma SourcesA71-L1: Sources level1 sA71 = \{sA6\}  
by (metis DSourcesA71-L1 SourcesA6-L1 Sources-emptyDSources SourcesOnlyDSources singleton-iff)

lemma SourcesA81-L1: Sources level1 sA81 = \{sA6, sA71, sA81, sA91\}  
proof –  
  have dA81: DSources level1 sA81 = \{sA71, sA91\} by (rule DSourcesA81-L1)  
  have dA91: DSources level1 sA91 = \{sA81\} by (rule DSourcesA91-L1)  
  have (Sources level1 sA81) = (DSources level1 sA81) ∪ (∪ S ∈ (DSources level1 sA81). (Sources level1 S))  
    by (rule SourcesDef)  
    with dA81 have (Sources level1 sA81) = (∪ S ∈ (DSources level1 sA81). (Sources level1 S))  
      by (metis (hide-lams, no-types) SUP-empty UN-insert Un-insert-left sup-bot.left-neutral sup-commute)  
  hence sourcesA81:(Sources level1 sA81) = (∪ S ∈ (DSources level1 sA81))  
    by (metis SourcesA71-L1 insert-is-Un sup-assoc)
have \((\text{Sources level1 } s_{A91}) = (\text{DSources level1 } s_{A91}) \cup (\bigcup S \in (\text{DSources level1 } s_{A91}). (\text{Sources level1 } S))\)
by (rule SourcesDef)
with \(d_{A91}\) have \((\text{Sources level1 } s_{A91}) = (\{ s_{A81} \} \cup (\text{Sources level1 } s_{A81}))\)
by simp
with \(s_{\text{sourcesA81}}\) have \((\text{Sources level1 } s_{A81}) = \{ s_{A71}, s_{A91}, s_{A6} \} \cup \{ s_{A81} \}\)
\(\cup \{ s_{A81}, s_{A91} \}\)
by (metis SourcesLoop)
thus \(\text{thesis}\) by auto
qed

lemma SourcesA91-L1: \(\text{Sources level1 } s_{A91} = \{ s_{A6}, s_{A71}, s_{A81}, s_{A91} \}\)
proof –
have \(\text{DSources level1 } s_{A91} = \{ s_{A81} \}\)
by (rule DSourcesA91-L1)
thus \(\text{thesis}\) by (metis SourcesA81-L1 Sources-singleDSource
Un-empty-left Un-insert-left insert-absorb2 insert-commute)
qed

lemma SourcesA92-L1: \(\text{Sources level1 } s_{A92} = \{ s_{A6}, s_{A71}, s_{A81}, s_{A91} \}\)
by (metis DSourcesA91-L1 DSourcesA92-L1 SourcesA91-L1 Sources-singleDSource)

lemma SourcesA72-L1: \(\text{Sources level1 } s_{A72} = \{ s_{A6} \}\)
by (metis DSourcesA6-L1 DSourcesA72-L1 SourcesOnlyDSources singleton-iff)

lemma SourcesA82-L1: \(\text{Sources level1 } s_{A82} = \{ s_{A6}, s_{A72} \}\)
proof –
have \(d_{A82}:\text{Sources level1 } s_{A82} = \{ s_{A72} \}\)
by (rule DSourcesA82-L1)
have \((\text{Sources level1 } s_{A82}) = (\text{DSources level1 } s_{A82}) \cup (\bigcup S \in (\text{DSources level1 } s_{A82}). (\text{Sources level1 } S))\)
by (rule SourcesDef)
with \(d_{A82}\) have \((\text{Sources level1 } s_{A82}) = \{ s_{A72} \} \cup (\text{Sources level1 } s_{A72})\)
by simp
thus \(\text{thesis}\) by (metis SourcesA72-L1 Un-commute insert-is-Un)
qed

lemma SourcesA93-L1: \(\text{Sources level1 } s_{A93} = \{ s_{A6}, s_{A72}, s_{A82} \}\)
by (metis DSourcesA93-L1 SourcesA82-L1 Sources-singleDSource Un-insert-right insert-is-Un)

— Abstraction level 2

lemma SourcesS1-L2: \(\text{Sources level2 } s_{S1} = \{ \}\)
proof –
have \(\text{DSources level2 } s_{S1} = \{ \}\)
by (simp add: DSources-def AbstrLevel2, auto)
thus \(\text{thesis}\) by (simp add: DSourcesEmptySources)
qed
lemma SourcesS2-L2: Sources level2 sS2 = {}
proof
  have DSources level2 sS2 = {} by (simp add: DSources-def AbstrLevel2, auto)
  thus ?thesis by (simp add: DSourcesEmptySources)
qed

lemma SourcesS3-L2: Sources level2 sS3 = {sS2}
proof
  have DSourcesS3: DSources level2 sS3 = {sS2} by (simp add: DSources-def AbstrLevel2, auto)
  have Sources level2 sS2 = {} by (rule SourcesS2-L2)
  with DSourcesS3 show ?thesis by (simp add: Sources-singleDSource)
qed

lemma SourcesS4-L2: Sources level2 sS4 = {sS2}
proof
  have DSourcesS4: DSources level2 sS4 = {sS2} by (simp add: DSources-def AbstrLevel2, auto)
  have Sources level2 sS4 = {sS2} by (rule SourcesS4-L2)
  with DSourcesS4 show ?thesis by (simp add: Sources-singleDSource)
qed

lemma SourcesS5-L2: Sources level2 sS5 = {sS2, sS4}
proof
  have DSourcesS5: DSources level2 sS5 = {sS4} by (simp add: DSources-def AbstrLevel2, auto)
  have Sources level2 sS4 = {sS2} by (rule SourcesS4-L2)
  with DSourcesS5 show ?thesis by (simp add: Sources-singleDSource)
qed

lemma SourcesS6-L2: Sources level2 sS6 = {sS2, sS4, sS5}
proof
  have DSourcesS6: DSources level2 sS6 = {sS2, sS3} by (simp add: DSources-def AbstrLevel2, auto)
  have Sources level2 sS2 = {} by (rule SourcesS2-L2)
  have Sources level2 sS5 = {sS2, sS4} by (rule SourcesS5-L2)
  with SourcesS2 DSourcesS6 show ?thesis by (simp add: Sources-2DSources, auto)
qed

lemma SourcesS7-L2: Sources level2 sS7 = {}
proof
  have DSources level2 sS7 = {} by (simp add: DSources-def AbstrLevel2, auto)
  thus ?thesis by (simp add: DSourcesEmptySources)
qed

lemma SourcesS8-L2: Sources level2 sS8 = {sS7}
proof 
  have DSourcesS8:DSources level2 sS8 = {sS7} by (simp add: DSources-def AbstrLevel2, auto)
  have Sources level2 sS7 = {} by (rule SourcesS7-L2)
  with DSourcesS8 show ?thesis by (simp add: Sources-singleDSource)
qed

lemma SourcesS9-L2:
Sources level2 sS9 = {}
proof 
  have DSources level2 sS9 = {} by (simp add: DSources-def AbstrLevel2, auto)
  thus ?thesis by (simp add: DSourcesEmptySources)
qed

lemma SourcesS10-L2: Sources level2 sS10 = {sS9}
proof 
  have DSourcesS10:DSources level2 sS10 = {sS9} by (simp add: DSources-def AbstrLevel2, auto)
  have Sources level2 sS9 = {} by (rule SourcesS9-L2)
  with DSourcesS10 show ?thesis by (simp add: Sources-singleDSource)
qed

lemma SourcesS11-L2: Sources level2 sS11 = {sS9}
proof 
  have DSourcesS11:DSources level2 sS11 = {sS9} by (simp add: DSources-def AbstrLevel2, auto)
  have Sources level2 sS9 = {} by (rule SourcesS9-L2)
  with DSourcesS11 show ?thesis by (simp add: Sources-singleDSource)
qed

lemma SourcesS12-L2: Sources level2 sS12 = {sS9, sS10}
proof 
  have DSourcesS12:DSources level2 sS12 = {sS10} by (simp add: DSources-def AbstrLevel2, auto)
  have Sources level2 sS10 = {sS9} by (rule SourcesS10-L2)
  with DSourcesS12 show ?thesis by (simp add: Sources-singleDSource)
qed

lemma SourcesS13-L2: Sources level2 sS13 = {sS9, sS10, sS12}
proof 
  have DSourcesS13:DSources level2 sS13 = {sS12} by (simp add: DSources-def AbstrLevel2, auto)
  have Sources level2 sS12 = {sS9, sS10} by (rule SourcesS12-L2)
  with DSourcesS13 show ?thesis by (simp add: Sources-singleDSource)
qed

lemma SourcesS14-L2: Sources level2 sS14 = {sS9, sS11}
proof 
  have DSourcesS14:DSources level2 sS14 = {sS11} by (simp add: DSources-def
AbstrLevel2, auto)
  have Sources level2 sS11 = {sS9} by (rule SourcesS11-L2)
  with DSourcesS14 show ?thesis by (simp add: Sources-singleDSource)
qed

lemma SourcesS15-L2: Sources level2 sS15 = {sS9, sS11, sS14}
proof
  have DSourcesS15: DSources level2 sS15 = {sS14} by (simp add: DSources-def AbstrLevel2, auto)
  have Sources level2 sS14 = {sS9, sS11} by (rule SourcesS14-L2)
  with DSourcesS15 show ?thesis by (simp add: Sources-singleDSource)
qed

4.5 Minimal sets of components to prove certain properties

lemma minSetOfComponentsTestL2p1:
  minSetOfComponents level2 {data10, data13} = {sS1}
proof
  have outL2: outSetOfComponents level2 {data10, data13} = {sS1} by (simp add: outSetOfComponents-def AbstrLevel2, auto)
  have Sources level2 sS1 = {} by (simp add: SourcesS1-L2)
  with outL2 show ?thesis by (simp add: minSetOfComponents-def)
qed

lemma NOT-noIrrelevantChannelsTestL2p1:
  ¬ noIrrelevantChannels level2 {data10, data13}
by (simp add: noIrrelevantChannels-def systemIN-def minSetOfComponentsTestL2p1 AbstrLevel2)

lemma NOT-allNeededINChannelsTestL2p1:
  ¬ allNeededINChannels level2 {data10, data13}
by (simp add: allNeededINChannels-def systemIN-def minSetOfComponentsTestL2p1 AbstrLevel2)

lemma minSetOfComponentsTestL2p2:
  minSetOfComponents level2 {data1, data12} = {sS2, sS4, sS5, sS6}
proof
  have outL2: outSetOfComponents level2 {data1, data12} = {sS6} by (simp add: outSetOfComponents-def AbstrLevel2, auto)
  have Sources level2 sS6 = {sS2, sS4, sS5} by (simp add: SourcesS6-L2)
  with outL2 show ?thesis by (simp add: minSetOfComponents-def)
qed

lemma noIrrelevantChannelsTestL2p2:
  noIrrelevantChannels level2 {data1, data12}
by (simp add: noIrrelevantChannels-def systemIN-def minSetOfComponentsTestL2p2 AbstrLevel2)

58
lemma allNeededINChannelsTestL2p2:
allNeededINChannels level2 {data1, data12}
by (simp add: allNeededINChannels-def minSetOfComponentsTestL2p2 systemIN-def AbstrLevel2)

lemma minSetOfComponentsTestL1p3:
minSetOfComponents level1 {data1, data10, data11} = {sA12, sA11, sA21}
proof –
  have sg1:outSetOfComponents level1 {data1, data10, data11} = {sA12, sA21}
  by (simp add: outSetOfComponents-def AbstrLevel1, auto)
  have DSources level1 sA12 = {}
  by (simp add: DSources-def AbstrLevel1, auto)
  hence sg2:Sources level1 sA12 = {} by (simp add: DSourcesEmptySources)
  have sg3:DSources level1 sA21 = {sA11}
  by (simp add: DSources-def AbstrLevel1, auto)
  have sg4:Sources level1 sA21 = {sA11}
  by (simp add: SourcesS1-L2)
  with sg1 sg2 show ?thesis by (simp add: minSetOfComponents-def, blast)
qed

lemma noIrrelevantChannelsTestL1p3:
noIrrelevantChannels level1 {data1, data10, data11}
by (simp add: noIrrelevantChannels-def systemIN-def minSetOfComponentsTestL1p3 AbstrLevel1)

lemma allNeededINChannelsTestL1p3:
allNeededINChannels level1 {data1, data10, data11}
by (simp add: allNeededINChannels-def minSetOfComponentsTestL1p3 systemIN-def AbstrLevel1)

lemma minSetOfComponentsTestL2p3:
minSetOfComponents level2 {data1, data10, data11} = {sS1, sS2, sS3}
proof –
  have sg1:outSetOfComponents level2 {data1, data10, data11} = {sS1, sS3}
  by (simp add: outSetOfComponents-def AbstrLevel2, auto)
  have sS1:Sources level2 sS1 = {} by (simp add: SourcesS1-L2)
  have Sources level2 sS3 = {sS2} by (simp add: SourcesS3-L2)
  with sg1 sS1 show ?thesis by (simp add: minSetOfComponents-def, blast)
qed

lemma noIrrelevantChannelsTestL2p3:
oIrrelevantChannels level2 {data1, data10, data11}
by (simp add: noIrrelevantChannels-def systemIN-def minSetOfComponentsTestL2p3)
lemma allNeededINChannelsTestL2p3:
allNeededINChannels level2 \{data1, data10, data11\}
by (simp add: allNeededINChannels-def minSetOfComponentsTestL2p3 systemIN-def AbstrLevel2)