Formalisation and Analysis of Component Dependencies

Maria Spichkova

February 23, 2021

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

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 *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 ⇒ chanID set* in Isabelle) and $\mathbb{O}(S)$ (function *OUT :: CSet ⇒ chanID set* in Isabelle). The set of local variables of components is defined in Isabelle by VAR, and the function to map component identifiers to the corresponding variables is defined by *VAR :: CSet ⇒ 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 *DataDependenciesConcreteValues.thy* (specification of the system architecture on several abstraction levels) and *DataDependenciesCaseStudy.thy* (proofs of system architectures' properties).

Function *subcomp :: CSet ⇒ CSet set* maps components to a (possibly empty) set of its subcomponents.

We specify the components' dependencies by the function

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

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

$$\begin{aligned}&Sources^L(C) = \\ &DSources^L(C) \;\cup\; \bigcup_{S \in DSources^L(C)} \{S_1 \mid S_1 \in Sources^L(S)\}\end{aligned}$$

Direct data dependencies are defined by the function

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

$$DSources^L(C) = \{S \mid \exists x \in \mathbb{I}(C) \land x \in \mathbb{O}(S)\}$$

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.

$\mathbb{I}^{\mathcal{D}}(C, y)$ 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 vial 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 $\mathbb{I}^{\mathcal{D}}(C, y) = \{x\}$. In Isabelle, $\mathbb{I}^{\mathcal{D}}(C, y)$ 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 $\mathbb{O}^{\mathcal{D}}$ returns the corresponding set $\mathbb{O}^{\mathcal{D}}(C, x)$ 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: $\forall x \in \mathbb{I}(C).\ \mathbb{O}^{\mathcal{D}}(C, x) \neq \emptyset$.

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 $\mathbb{I}^{\mathcal{D}}$ of data dependencies between the components are defined in the theory *DataDependenciesConcreteValues.thy*. We represent the dependencies graphically using dashed lines over the component box.

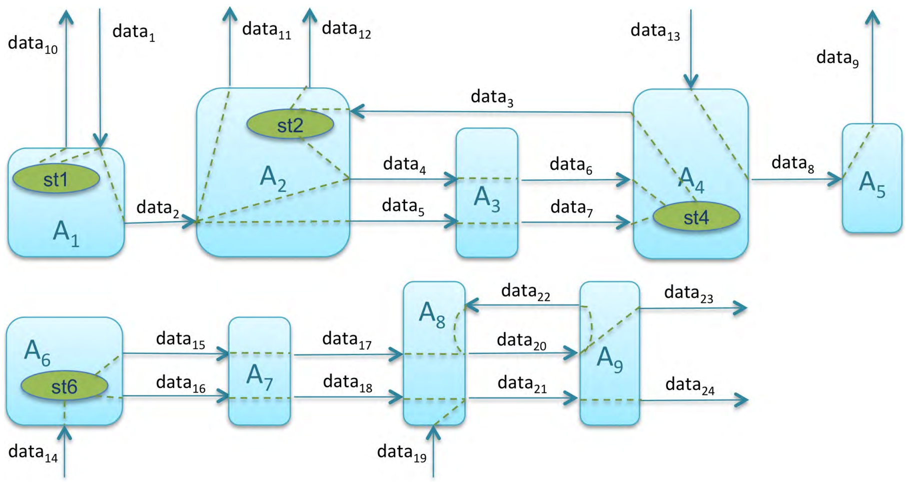

Figure 1: System S: Data dependencies and $\mathbb{I}^{\mathcal{D}}$ sets

Now we can decompose the system's components according to the given $\mathbb{I}^{\mathcal{D}}$ 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 subcomponent of a system computing the corresponding results. We represent this (flat) architecture as a directed graph (components become vertices and channels become edges) and apply one of the existing distributed algorithms for the decomposition into its strongly connected components, e.g. FB [2], OBF [1], or the colouring algorithm [4]. Fig. 3 presents the result of the architecture optimisation.

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

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

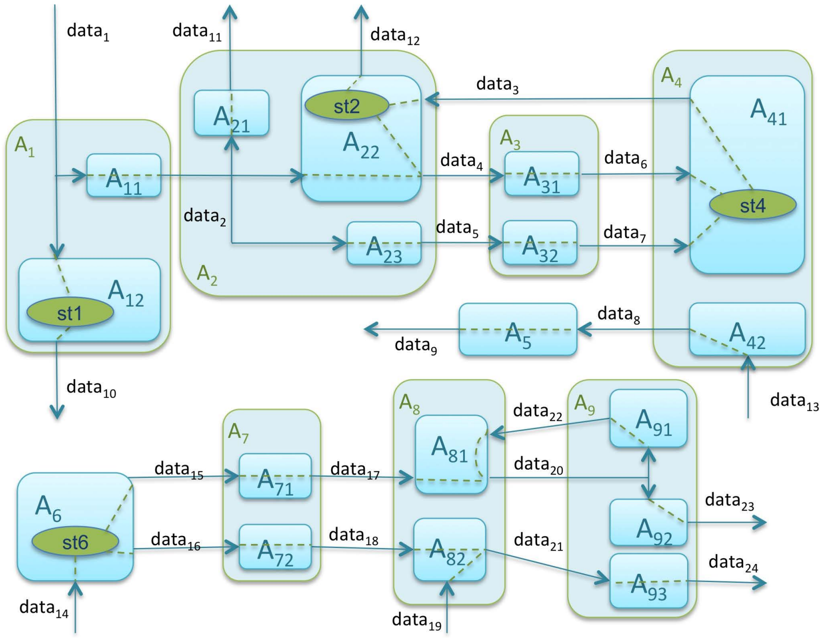

Figure 2: Components' decomposition (level L_1)

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

- measure for costs of the data transfer/ upload to the cloud *UplSize(f)*: size of messages (data packages) within a data flow f and frequency they are produced. This measure can be defined on the level of logical modelling, where we already know the general type of the data and can also analyse the corresponding component (or environment) model to estimate the frequency the data are produced;
- measure for requirement of using high-performance computing and cloud virtual machines, *Perf(X)*: complexity of the computation within a component X, which can be estimated on the level of logical modelling as well.

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

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

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

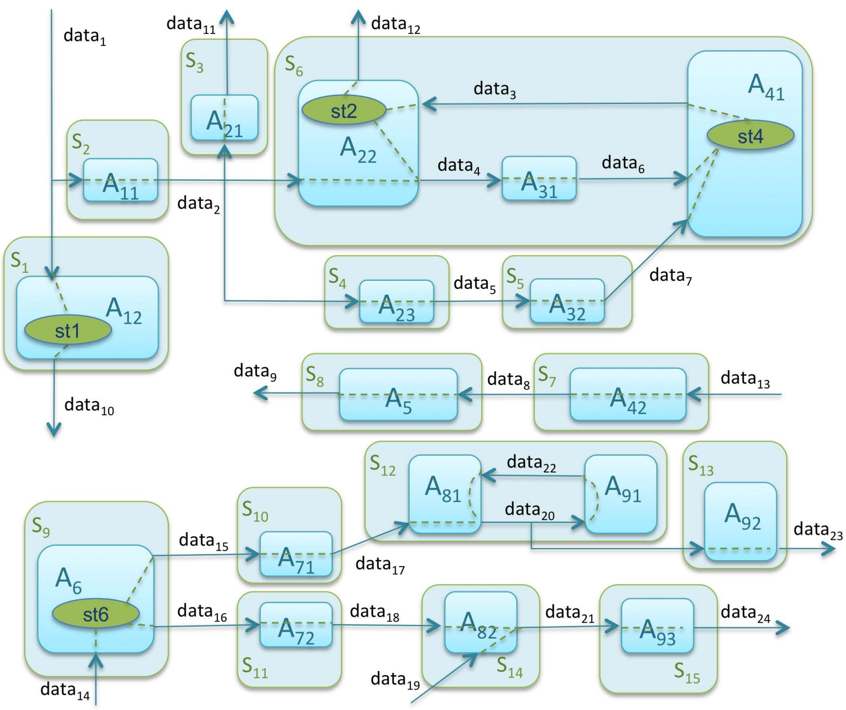

Figure 3: Architecture of S (level L_2)

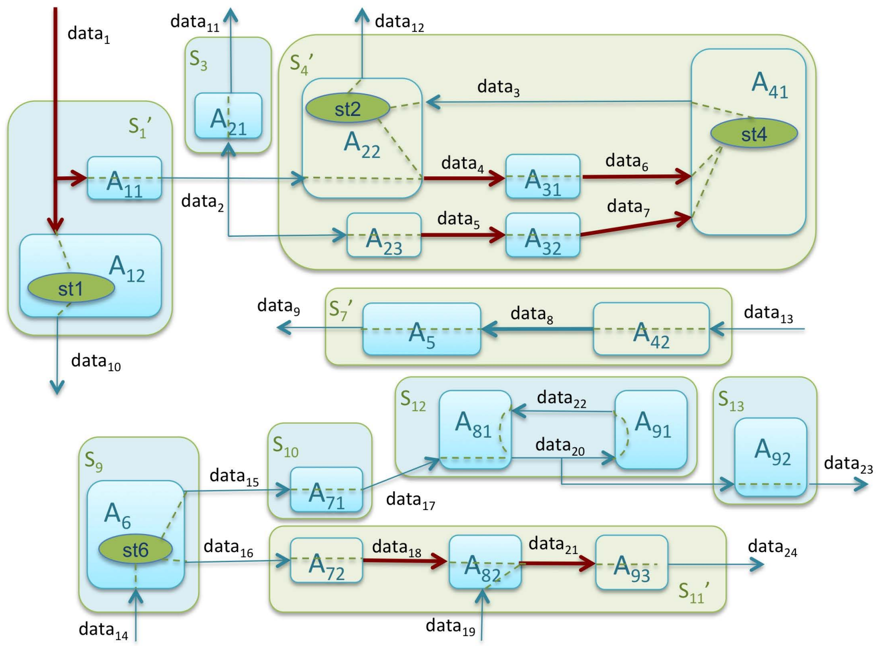

Figure 4: Optimised architecture of S (Level L_3)

2 Case Study: Definitions

theory *DataDependenciesConcreteValues*
 imports *Main*
begin

datatype *CSet* = *sA1*| *sA2*| *sA3*| *sA4*| *sA5*| *sA6*| *sA7*| *sA8*| *sA9*|
sA11| *sA12*| *sA21*| *sA22*| *sA23*| *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* }

| *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* ⇒ *chanID set*
where
OUT sA1 = { *data2*, *data10* }
| *OUT sA2* = { *data4*, *data5*, *data11*, *data12* }

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


— function VAR maps component IDs to the set of its local variables
fun VAR :: CSet ⇒ varID set
where
  VAR sA1 = { stA1 }
| VAR sA2 = { stA2 }
```

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

— function subcomp maps component ID to the set of its subcomponents
fun *subcomp* :: $CSet \Rightarrow CSet\ set$
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

axiomatization
  AbstrLevel :: *AbstrLevelsID* ⇒ *CSet set*
where
AbstrLevel0:

AbstrLevel level0 = {*sA1*, *sA2*, *sA3*, *sA4*, *sA5*, *sA6*, *sA7*, *sA8*, *sA9*}
and
AbstrLevel1:
AbstrLevel level1 = {*sA11*, *sA12*, *sA21*, *sA22*, *sA23*, *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 derectly;
— 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* = {}
| *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 $OUTfromV :: chanID \Rightarrow varID\ set$
where
$OUTfromV\ data1 = \{\}$
$|\ OUTfromV\ data2 = \{\}$
$|\ OUTfromV\ data3 = \{stA4\}$
$|\ OUTfromV\ data4 = \{stA2\}$
$|\ OUTfromV\ data5 = \{\}$
$|\ OUTfromV\ data6 = \{\}$
$|\ OUTfromV\ data7 = \{\}$
$|\ OUTfromV\ data8 = \{\}$
$|\ OUTfromV\ data9 = \{\}$
$|\ OUTfromV\ data10 = \{stA1\}$
$|\ OUTfromV\ data11 = \{\}$
$|\ OUTfromV\ data12 = \{stA2\}$
$|\ OUTfromV\ data13 = \{\}$
$|\ OUTfromV\ data14 = \{\}$
$|\ OUTfromV\ data15 = \{stA6\}$
$|\ OUTfromV\ data16 = \{stA6\}$
$|\ OUTfromV\ data17 = \{\}$
$|\ OUTfromV\ data18 = \{\}$
$|\ OUTfromV\ data19 = \{\}$
$|\ OUTfromV\ data20 = \{\}$
$|\ OUTfromV\ data21 = \{\}$
$|\ OUTfromV\ data22 = \{\}$
$|\ OUTfromV\ data23 = \{\}$
$|\ OUTfromV\ data24 = \{\}$

— Set of channels channels which have UplSize measure greather that the predifined value $HighLoad$
definition
$UplSizeHighLoad :: chanID\ set$
where
$UplSizeHighLoad \equiv \{data1,\ data4,\ data5,\ data6,\ data7,\ data8,\ data18,\ data21\}$

— Set of components from the abstraction level 1 for which the Perf measure is greather that the predifined value $HighPerf$

definition

HighPerfSet :: *CSet set*

where

HighPerfSet ≡ {*sA22*, *sA23*, *sA41*, *sA42*, *sA72*, *sA93*}

end

3 Inter-/Intracomponent dependencies

theory *DataDependencies*
imports *DataDependenciesConcreteValues*
begin

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

definition

correctCompositionDiffLevels :: *CSet* ⇒ *bool*

where

correctCompositionDiffLevels S ≡
∀ *C* ∈ *subcomp S*. ∀ *i*. *S* ∈ *AbstrLevel i* ⟶ *C* ∉ *AbstrLevel i*

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

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

definition

correctCompositionDiffLevelsSYSTEM :: *bool*

where

correctCompositionDiffLevelsSYSTEM ≡
(∀ *S*::*CSet*. (*correctCompositionDiffLevels S*))

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

definition

correctCompositionVAR :: *CSet* ⇒ *bool*

where

correctCompositionVAR S ≡
∀ *C* ∈ *subcomp S*. ∀ *v* ∈ *VAR C*. *v* ∈ *VAR S*

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

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

definition

correctCompositionVARSYSTEM :: *bool*

where

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

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

definition
$correctDeCompositionVAR ::\ CSet \Rightarrow bool$
where
$correctDeCompositionVAR\ S \equiv$
$\forall\ v \in VAR\ S.\ \forall\ C1 \in\ subcomp\ S.\ \forall\ C2 \in\ subcomp\ S.\ v \in VAR\ C1 \wedge v \in VAR\ C2 \longrightarrow C1 = C2$

— General system's property: for all abstraction levels and all components should hold
— after correct decomposition of a component each of its local variable can belong only to one of its subcomponents

definition
$correctDeCompositionVARSYSTEM ::\ bool$
where
$correctDeCompositionVARSYSTEM \equiv$
$(\forall\ S::CSet.\ (correctDeCompositionVAR\ S))$

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

definition
$correctCompositionOUT ::\ chanID \Rightarrow bool$
where
$correctCompositionOUT\ x \equiv$
$\forall\ C\ i.\ x \in OUT\ C \wedge C \in AbstrLevel\ i \longrightarrow\ (\forall\ S \in AbstrLevel\ i.\ x \notin OUT\ S)$

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

definition
$correctCompositionOUTSYSTEM ::\ bool$
where
$correctCompositionOUTSYSTEM \equiv (\forall\ x.\ correctCompositionOUT\ x)$

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

definition
$correctCompositionSubcomp ::\ CSet \Rightarrow bool$
where
$correctCompositionSubcomp\ X \equiv$
$\forall\ C\ i.\ X \in subcomp\ C \wedge C \in AbstrLevel\ i \longrightarrow\ (\forall\ S \in AbstrLevel\ i.\ (S \neq C \longrightarrow X \notin subcomp\ S))$

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

definition
$correctCompositionSubcompSYSTEM ::\ bool$

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

— If a component belongs is defined in the set CSet, it should belong to at least one abstraction level
definition
$allComponentsUsed :: bool$
where
$allComponentsUsed \equiv \forall\ C.\ \exists\ i.\ C \in AbstrLevel\ i$

— if a component does not have any local variables, none of its subcomponents has any local variables
lemma *correctDeCompositionVARempty*:
assumes $correctCompositionVAR\ S$
and $VAR\ S = \{\}$
shows $\forall\ C \in subcomp\ S.\ VAR\ C = \{\}$
$\langle proof \rangle$
definition $OUTfrom :: chanID \Rightarrow chanID\ set$
where
$OUTfrom\ x \equiv (OUTfromCh\ x) \cup \{y.\ \exists\ v.\ v \in (OUTfromV\ x) \wedge y \in (VARfrom\ v)\}$

— if x depends from some input channel(s) directly, then exists
— a component which has them as input channels and x as an output channel
definition
$OUTfromChCorrect :: chanID \Rightarrow bool$
where
$OUTfromChCorrect\ x \equiv$
$(OUTfromCh\ x \neq \{\} \longrightarrow$
$(\exists\ Z\ .\ (x \in (OUT\ Z) \wedge (\forall\ y \in (OUTfromCh\ x).\ y \in IN\ Z)\ )))$

— General system's property: for channels in the system should hold:
— if x depends from some input channel(s) directly, then exists
— a component which has them as input channels and x as an output channel
definition
$OUTfromChCorrectSYSTEM :: bool$
where
$OUTfromChCorrectSYSTEM \equiv (\forall\ x{::}chanID.\ (OUTfromChCorrect\ x))$

— if x depends from some local variables, then exists a component
— to which these variables belong and which has x as an output channel
definition
$OUTfromVCorrect1 :: chanID \Rightarrow bool$
where
$OUTfromVCorrect1\ x \equiv$
$(OUTfromV\ x \neq \{\} \longrightarrow$
$(\exists\ Z\ .\ (x \in (OUT\ Z) \wedge (\forall\ v \in (OUTfromV\ x).\ v \in VAR\ Z)\ )))$

— General system's property: for channels in the system should hold the above property:
definition
$OUTfromVCorrect1SYSTEM :: bool$
where
$OUTfromVCorrect1SYSTEM \equiv (\forall\ x::chanID.\ (OUTfromVCorrect1\ x))$

— if x does not depend from any local variables, then it does not belong to any set VARfrom
definition
$OUTfromVCorrect2 :: chanID \Rightarrow bool$
where
$OUTfromVCorrect2\ x \equiv$
$(OUTfromV\ x = \{\} \longrightarrow (\forall\ v::varID.\ x \notin (VARto\ v)))$

— General system's property: for channels in the system should hold the above property:
definition
$OUTfromVCorrect2SYSTEM :: bool$
where
$OUTfromVCorrect2SYSTEM \equiv (\forall\ x::chanID.\ (OUTfromVCorrect2\ x))$

— General system's property:
— definitions OUTfromV and VARto should give equivalent mappings
definition
$OUTfromV\text{-}VARto :: bool$
where
$OUTfromV\text{-}VARto \equiv$
$(\forall\ x::chanID.\ \forall\ v::varID.\ (v \in OUTfromV\ x \longleftrightarrow x \in (VARto\ v)))$

— 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
$VARfromCorrectSYSTEM :: bool$
where
$VARfromCorrectSYSTEM \equiv$
$(\forall\ v::varID.\ \forall\ Z \in ((AbstrLevel\ level0) \cup (AbstrLevel\ level1)).$
$((v \in VAR\ Z) \longrightarrow (\forall\ x \in VARfrom\ v.\ x \in IN\ Z)))$

— 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
$VARtoCorrectSYSTEM \equiv$
$(\forall\ v::varID.\ \forall\ Z \in ((AbstrLevel\ level0) \cup (AbstrLevel\ level1)).$
$((v \in VAR\ Z) \longrightarrow (\forall\ x \in VARto\ v.\ x \in OUT\ Z)))$

— to detect local variables, unused for computation of any output
definition
VARusefulSYSTEM :: *bool*
where
$VARusefulSYSTEM \equiv (\forall\ v{::}varID.\ (VARto\ v \neq \{\}))$

lemma
OUTfromV-VARto-lemma:
assumes $OUTfromV\ x \neq \{\}$ **and** *OUTfromV-VARto*
shows $\exists\ v{::}varID.\ x \in (VARto\ v)$
⟨*proof*⟩

3.1 Direct and indirect data dependencies between components

— The component C should be defined on the same abstraction
— level we are seaching for its direct or indirect sources,
— otherwise we get an empty set as result
definition
DSources :: *AbstrLevelsID* ⇒ *CSet* ⇒ *CSet set*
where
$DSources\ i\ C \equiv \{Z.\ \exists\ x.\ x \in (IN\ C) \wedge x \in (OUT\ Z) \wedge Z \in (AbstrLevel\ i) \wedge C \in (AbstrLevel\ i)\}$

lemma *DSourcesLevelX*:
$(DSources\ i\ X) \subseteq (AbstrLevel\ i)$
⟨*proof*⟩
definition
DAcc :: *AbstrLevelsID* ⇒ *CSet* ⇒ *CSet set*
where
$DAcc\ i\ C \equiv \{Z.\ \exists\ x.\ x \in (OUT\ C) \wedge x \in (IN\ Z) \wedge Z \in (AbstrLevel\ i) \wedge C \in (AbstrLevel\ i)\}$

axiomatization
Sources :: *AbstrLevelsID* ⇒ *CSet* ⇒ *CSet set*
where
SourcesDef:
$(Sources\ i\ C) = (DSources\ i\ C) \cup (\bigcup\ S \in (DSources\ i\ C).\ (Sources\ i\ S))$
and
SourceExistsDSource:
$S \in (Sources\ i\ C) \longrightarrow (\exists\ Z.\ S \in (DSources\ i\ Z))$
and
NDSourceExistsDSource:
$S \in (Sources\ i\ C) \wedge S \notin (DSources\ i\ C) \longrightarrow$
$(\exists\ Z.\ S \in (DSources\ i\ Z) \wedge Z \in (Sources\ i\ C))$
and
SourcesTrans:
$(C \in Sources\ i\ S \wedge S \in Sources\ i\ Z) \longrightarrow C \in Sources\ i\ Z$

and
SourcesLevelX:
$(Sources\ i\ X) \subseteq (AbstrLevel\ i)$
and
SourcesLoop:
$(Sources\ i\ C) = (XS \cup (Sources\ i\ S)) \wedge (Sources\ i\ S) = (ZS \cup (Sources\ i\ C))$
$\longrightarrow (Sources\ i\ C) = XS \cup ZS \cup \{ C, S\}$
— if we have a loop in the dependencies we need to cut it for counting the sources

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

lemma *Acc-SourcesNOT*: $(X \notin Acc\ i\ C) = (C \notin Sources\ i\ X)$
⟨*proof*⟩
definition
$isNotDSource :: AbstrLevelsID \Rightarrow CSet \Rightarrow bool$
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
$isNotDSourceX\ i\ S\ C \equiv (\forall\ x \in (OUT\ S).\ (C \notin (AbstrLevel\ i) \vee (x \notin (IN\ C))))$

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

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

lemma *DAcc-DSourcesNOT*:
$(X \notin DAcc\ i\ C) = (C \notin DSources\ i\ X)$

⟨*proof*⟩

lemma *DSource-level*:
assumes $S \in (DSources\ i\ C)$
shows $C \in (AbstrLevel\ i)$
⟨*proof*⟩

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

lemma *Sources-DSources*:
$(DSources\ i\ C) \subseteq (Sources\ i\ C)$
⟨*proof*⟩

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

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

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

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

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

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

lemma *allNotDSource-NotSource*:
assumes $\forall\ C\ .\ S \notin (DSources\ i\ C)$
shows $\forall\ Z.\ S \notin (Sources\ i\ Z)$

⟨*proof*⟩

lemma *NotDSource-NotSource*:
assumes $\forall\ C \in (AbstrLevel\ i).\ S \notin (DSources\ i\ C)$
shows $\forall\ Z \in (AbstrLevel\ i).\ S \notin (Sources\ i\ Z)$
⟨*proof*⟩

lemma *isNotSource-Sources*:
assumes *isNotDSource i S*
shows $\forall\ C \in (AbstrLevel\ i).\ S \notin (Sources\ i\ C)$
⟨*proof*⟩

lemma *SourcesAbstrLevel*:
assumes $x \in Sources\ i\ S$
shows $x \in AbstrLevel\ i$
⟨*proof*⟩

lemma *DSourceIsSource*:
assumes $C \in DSources\ i\ S$
shows $C \in Sources\ i\ S$
⟨*proof*⟩

lemma *DSourceOfDSource*:
assumes $Z \in DSources\ i\ S$
and $S \in DSources\ i\ C$
shows $Z \in Sources\ i\ C$
⟨*proof*⟩

lemma *SourceOfDSource*:
assumes $Z \in Sources\ i\ S$
and $S \in DSources\ i\ C$
shows $Z \in Sources\ i\ C$
⟨*proof*⟩

lemma *DSourceOfSource*:
assumes $cDS{:}C \in DSources\ i\ S$
and $sS{:}S \in Sources\ i\ Z$
shows $C \in Sources\ i\ Z$
⟨*proof*⟩

lemma *Sources-singleDSource*:
assumes $DSources\ i\ S = \{C\}$
shows $Sources\ i\ S = \{C\} \cup Sources\ i\ C$
⟨*proof*⟩

lemma *Sources-2DSources*:
assumes $DSources\ i\ S = \{C1,\ C2\}$
shows $Sources\ i\ S = \{C1,\ C2\} \cup Sources\ i\ C1\ \cup Sources\ i\ C2$
⟨*proof*⟩

lemma *Sources-3DSources*:
assumes *DSources i S* = {*C1*, *C2*, *C3*}
shows *Sources i S* = {*C1*, *C2*, *C3*} ∪ *Sources i C1* ∪ *Sources i C2* ∪ *Sources i C3*
⟨*proof*⟩

lemma *singleDSourceEmpty4isNotDSource*:
assumes *DAcc i C* = {*S*}
and *Z* ≠ *S*
shows *C* ∉ (*DSources i Z*)
⟨*proof*⟩

lemma *singleDSourceEmpty4isNotDSourceLevel*:
assumes *DAcc i C* = {*S*}
shows ∀ *Z* ∈ (*AbstrLevel i*). *Z* ≠ *S* ⟶ *C* ∉ (*DSources i Z*)
⟨*proof*⟩

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

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

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

lemma *singleDSourceEmpty4isNotSource*:
assumes *DAcc i C* = {*S*}
and *nSourcS*:*isNotDSource i S*
and *Z* ≠ *S*
shows *C* ∉ (*Sources i Z*)
⟨*proof*⟩

lemma *singleDSourceEmpty4isNotSourceLevel*:
assumes *DAcc i C* = {*S*}
and *nSourcS*:*isNotDSource i S*
shows ∀ *Z* ∈ (*AbstrLevel i*). *Z* ≠ *S* ⟶ *C* ∉ (*Sources i Z*)
⟨*proof*⟩

lemma *singleDSourceLoop*:
assumes $DAcc\ i\ C = \{S\}$
and $DAcc\ i\ S = \{C\}$
shows $\forall\ Z \in (AbstrLevel\ i).\ (Z \neq S \wedge Z \neq C \longrightarrow C \notin (Sources\ i\ Z))$
⟨*proof*⟩

3.2 Components that are elementary wrt. data dependencies

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

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

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

— the set (outSetCorelated x) is empty if x does not depend from any variable
lemma *outSetCorelatedEmpty1*:
assumes $OUTfromV\ x = \{\}$
shows $outSetCorelated\ x = \{\}$
⟨*proof*⟩
lemma *outSetCorelatedNonemptyX*:
assumes $OUTfromV\ x \neq \{\}$ **and** *correct3*:*OUTfromV-VARto*
shows $x \in outSetCorelated\ x$
⟨*proof*⟩
lemma *outSetCorelatedEmpty2*:
assumes $outSetCorelated\ x = \{\}$ **and** *correct3*:*OUTfromV-VARto*
shows $OUTfromV\ x = \{\}$
⟨*proof*⟩

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 chSet

definition
$inSetOfComponents :: AbstrLevelsID \Rightarrow chanID\ set \Rightarrow CSet\ set$
where
$inSetOfComponents\ i\ chSet \equiv$
$\{X.\ (((IN\ X) \cap chSet \neq \{\})\ \wedge X \in (AbstrLevel\ i))\}$

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

definition
$outSetOfComponents :: AbstrLevelsID \Rightarrow chanID\ set \Rightarrow CSet\ set$
where
$outSetOfComponents\ i\ chSet \equiv$
$\{Y.\ (((OUT\ Y) \cap chSet \neq \{\}) \wedge Y \in (AbstrLevel\ i))\}$

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

definition
$minSetOfComponents :: AbstrLevelsID \Rightarrow chanID\ set \Rightarrow CSet\ set$
where
$minSetOfComponents\ i\ chSet \equiv$
$(outSetOfComponents\ i\ chSet) \cup$
$(\bigcup\ S \in (outSetOfComponents\ i\ chSet).\ (Sources\ i\ S))$

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

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

definition $systemIN :: chanID \Rightarrow AbstrLevelsID \Rightarrow bool$
where
$systemIN\ x\ i \equiv (\exists\ C1 \in (AbstrLevel\ i).\ x \in (IN\ C1)) \wedge (\forall\ C2 \in (AbstrLevel\ i).\ x \notin (OUT\ C2))$

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

definition $systemOUT :: chanID \Rightarrow AbstrLevelsID \Rightarrow bool$
where
$systemOUT\ x\ i \equiv (\forall\ C1 \in (AbstrLevel\ i).\ x \notin (IN\ C1)) \wedge (\exists\ C2 \in (AbstrLevel\ i).\ x \in (OUT\ C2))$

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

definition $systemLOC :: chanID \Rightarrow AbstrLevelsID \Rightarrow bool$
where
$systemLOC\ x\ i \equiv (\exists\ C1 \in (AbstrLevel\ i).\ x \in (IN\ C1)) \wedge (\exists\ C2 \in (AbstrLevel\ i).\ x \in (OUT\ C2))$

lemma *systemIN-noOUT*:
assumes $systemIN\ x\ i$
shows $\neg\ systemOUT\ x\ i$

⟨*proof*⟩

lemma *systemOUT-noIN*:
assumes *systemOUT x i*
shows ¬ *systemIN x i*
⟨*proof*⟩

lemma *systemIN-noLOC*:
assumes *systemIN x i*
shows ¬ *systemLOC x i*
⟨*proof*⟩

lemma *systemLOC-noIN*:
assumes *systemLOC x i*
shows ¬ *systemIN x i*
⟨*proof*⟩

lemma *systemOUT-noLOC*:
assumes *systemOUT x i*
shows ¬ *systemLOC x i*
⟨*proof*⟩

lemma *systemLOC-noOUT*:
assumes *systemLOC x i*
shows ¬ *systemOUT x i*
⟨*proof*⟩

definition
noIrrelevantChannels :: *AbstrLevelsID* ⇒ *chanID set* ⇒ *bool*
where
noIrrelevantChannels i chSet ≡
∀ *x* ∈ *chSet*. ((*systemIN x i*) ⟶
(∃ *Z* ∈ (*minSetOfComponents i chSet*). *x* ∈ (*IN Z*)))

definition
allNeededINChannels :: *AbstrLevelsID* ⇒ *chanID set* ⇒ *bool*
where
allNeededINChannels i chSet ≡
(∀ *Z* ∈ (*minSetOfComponents i chSet*). ∃ *x* ∈ (*IN Z*). ((*systemIN x i*) ⟶ (*x* ∈ *chSet*)))

— the set (outSetOfComponents i chSet) should be a subset of all components specified on the abstraction level i
lemma *outSetOfComponentsLimit*:
outSetOfComponents i chSet ⊆ *AbstrLevel i*
⟨*proof*⟩
lemma *inSetOfComponentsLimit*:
inSetOfComponents i chSet ⊆ *AbstrLevel i*

⟨*proof*⟩
lemma *SourcesLevelLimit*:
$(\bigcup\ S \in (\mathit{outSetOfComponents}\ i\ \mathit{chSet}).\ (\mathit{Sources}\ i\ S)) \subseteq \mathit{AbstrLevel}\ i$
⟨*proof*⟩

lemma *minSetOfComponentsLimit*:
$\mathit{minSetOfComponents}\ i\ \mathit{chSet} \subseteq \mathit{AbstrLevel}\ i$
⟨*proof*⟩

3.4 Additional properties: Remote Computation

— The value of $UplSizeHighLoad\ x$ is True if its $UplSize$ measure is greather that a predifined value
definition *UplSizeHighLoadCh* :: *chanID* ⇒ *bool*
where
  $\mathit{UplSizeHighLoadCh}\ x \equiv (x \in \mathit{UplSizeHighLoad})$

— if the $Perf$ measure of at least one subcomponent is greather than a predifined value,
— the $Perf$ measure of this component is greather than $HighPerf$ too
axiomatization *HighPerfComp* :: *CSet* ⇒ *bool*
where
HighPerfComDef:
  $\mathit{HighPerfComp}\ C =$
  $((C \in \mathit{HighPerfSet}) \vee (\exists\ Z \in \mathit{subcomp}\ C.\ (\mathit{HighPerfComp}\ Z)))$

end

4 Case Study: Verification of Properties

theory *DataDependenciesCaseStudy*
  imports *DataDependencies*
begin

4.1 Correct composition of components

— the lemmas AbstrLevels X Y with corresponding proofs can be composend
— and proven automatically, their proofs are identical
lemma *AbstrLevels-A1-A11*:
  assumes $sA1 \in \mathit{AbstrLevel}\ i$
  shows $sA11 \notin \mathit{AbstrLevel}\ i$
⟨*proof*⟩

lemma *AbstrLevels-A1-A12*:
  assumes $sA1 \in \mathit{AbstrLevel}\ i$
  shows $sA12 \notin \mathit{AbstrLevel}\ i$⟨*proof*⟩

lemma *AbstrLevels-A2-A21*:
  assumes $sA2 \in \mathit{AbstrLevel}\ i$

shows *sA21* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-A2-A22*:
assumes *sA2* ∈ *AbstrLevel i*
shows *sA22* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-A2-A23*:
assumes *sA2* ∈ *AbstrLevel i*
shows *sA23* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-A3-A31*:
assumes *sA3* ∈ *AbstrLevel i*
shows *sA31* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-A3-A32*:
assumes *sA3* ∈ *AbstrLevel i*
shows *sA32* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-A4-A41*:
assumes *sA4* ∈ *AbstrLevel i*
shows *sA41* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-A4-A42*:
assumes *sA4* ∈ *AbstrLevel i*
shows *sA42* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-A7-A71*:
assumes *sA7* ∈ *AbstrLevel i*
shows *sA71* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-A7-A72*:
assumes *sA7* ∈ *AbstrLevel i*
shows *sA72* ∉ *AbstrLevel i*⟨*proof*⟩
lemma *AbstrLevels-A8-A81*:
assumes *sA8* ∈ *AbstrLevel i*
shows *sA81* ∉ *AbstrLevel i*⟨*proof*⟩
lemma *AbstrLevels-A8-A82*:
assumes *sA8* ∈ *AbstrLevel i*
shows *sA82* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-A9-A91*:
assumes *sA9* ∈ *AbstrLevel i*
shows *sA91* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-A9-A92*:
assumes *sA9* ∈ *AbstrLevel i*
shows *sA92* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-A9-A93*:

assumes *sA9* ∈ *AbstrLevel i*
shows *sA93* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S1-A12*:
assumes *sS1* ∈ *AbstrLevel i*
shows *sA12* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S2-A11*:
assumes *sS2* ∈ *AbstrLevel i*
shows *sA11* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S3-A21*:
assumes *sS3* ∈ *AbstrLevel i*
shows *sA21* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S4-A23*:
assumes *sS4* ∈ *AbstrLevel i*
shows *sA23* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S5-A32*:
assumes *sS5* ∈ *AbstrLevel i*
shows *sA32* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S6-A22*:
assumes *sS6* ∈ *AbstrLevel i*
shows *sA22* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S6-A31*:
assumes *sS6* ∈ *AbstrLevel i*
shows *sA31* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S6-A41*:
assumes *sS6* ∈ *AbstrLevel i*
shows *sA41* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S7-A42*:
assumes *sS7* ∈ *AbstrLevel i*
shows *sA42* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S8-A5*:
assumes *sS8* ∈ *AbstrLevel i*
shows *sA5* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S9-A6*:
assumes *sS9* ∈ *AbstrLevel i*
shows *sA6* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S10-A71*:
assumes *sS10* ∈ *AbstrLevel i*

shows *sA71* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S11-A72*:
assumes *sS11* ∈ *AbstrLevel i*
shows *sA72* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S12-A81*:
assumes *sS12* ∈ *AbstrLevel i*
shows *sA81* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S12-A91*:
assumes *sS12* ∈ *AbstrLevel i*
shows *sA91* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S13-A92*:
assumes *sS13* ∈ *AbstrLevel i*
shows *sA92* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S14-A82*:
assumes *sS14* ∈ *AbstrLevel i*
shows *sA82* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S15-A93*:
assumes *sS15* ∈ *AbstrLevel i*
shows *sA93* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S1opt-A11*:
assumes *sS1opt* ∈ *AbstrLevel i*
shows *sA11* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S1opt-A12*:
assumes *sS1opt* ∈ *AbstrLevel i*
shows *sA12* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S4opt-A23*:
assumes *sS4opt* ∈ *AbstrLevel i*
shows *sA23* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S4opt-A32*:
assumes *sS4opt* ∈ *AbstrLevel i*
shows *sA32* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S4opt-A22*:
assumes *sS4opt* ∈ *AbstrLevel i*
shows *sA22* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S4opt-A31*:
assumes *sS4opt* ∈ *AbstrLevel i*
shows *sA31* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S4opt-A41*:
assumes *sS4opt* ∈ *AbstrLevel i*
shows *sA41* ∉ *AbstrLevel i*⟨*proof*⟩

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

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

lemma *AbstrLevels-S11opt-A72*:
assumes *sS11opt* ∈ *AbstrLevel i*
shows *sA72* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S11opt-A82*:
assumes *sS11opt* ∈ *AbstrLevel i*
shows *sA82* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *AbstrLevels-S11opt-A93*:
assumes *sS11opt* ∈ *AbstrLevel i*
shows *sA93* ∉ *AbstrLevel i*⟨*proof*⟩

lemma *correctCompositionDiffLevelsA1*: *correctCompositionDiffLevels sA1*⟨*proof*⟩

lemma *correctCompositionDiffLevelsA2*: *correctCompositionDiffLevels sA2*⟨*proof*⟩

lemma *correctCompositionDiffLevelsA3*: *correctCompositionDiffLevels sA3*⟨*proof*⟩

lemma *correctCompositionDiffLevelsA4*: *correctCompositionDiffLevels sA4*⟨*proof*⟩

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

lemma *correctCompositionDiffLevelsA81*: *correctCompositionDiffLevels sA81*⟨*proof*⟩
lemma *correctCompositionDiffLevelsA82*: *correctCompositionDiffLevels sA82*⟨*proof*⟩
lemma *correctCompositionDiffLevelsA91*: *correctCompositionDiffLevels sA91*⟨*proof*⟩
lemma *correctCompositionDiffLevelsA92*: *correctCompositionDiffLevels sA92*⟨*proof*⟩
lemma *correctCompositionDiffLevelsA93*: *correctCompositionDiffLevels sA93*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS1*: *correctCompositionDiffLevels sS1*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS2*: *correctCompositionDiffLevels sS2*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS3*: *correctCompositionDiffLevels sS3*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS4*: *correctCompositionDiffLevels sS4*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS5*: *correctCompositionDiffLevels sS5*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS6*: *correctCompositionDiffLevels sS6*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS7*: *correctCompositionDiffLevels sS7*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS8*: *correctCompositionDiffLevels sS8*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS9*: *correctCompositionDiffLevels sS9*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS10*: *correctCompositionDiffLevels sS10*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS11*: *correctCompositionDiffLevels sS11*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS12*: *correctCompositionDiffLevels sS12*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS13*: *correctCompositionDiffLevels sS13*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS14*: *correctCompositionDiffLevels sS14*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS15*: *correctCompositionDiffLevels sS15*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS1opt*: *correctCompositionDiffLevels sS1opt*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS4opt*: *correctCompositionDiffLevels sS4opt*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS7opt*: *correctCompositionDiffLevels sS7opt*⟨*proof*⟩
lemma *correctCompositionDiffLevelsS11opt*: *correctCompositionDiffLevels sS11opt*⟨*proof*⟩
lemma *correctCompositionDiffLevelsSYSTEM-holds*:
correctCompositionDiffLevelsSYSTEM⟨*proof*⟩
lemma *correctCompositionVARSYSTEM-holds*:
correctCompositionVARSYSTEM
⟨*proof*⟩

lemma *correctDeCompositionVARSYSTEM-holds*:
correctDeCompositionVARSYSTEM
⟨*proof*⟩

4.2 Correct specification of the relations between channels

lemma *OUTfromChCorrect-data1*: *OUTfromChCorrect data1*
⟨*proof*⟩

lemma *OUTfromChCorrect-data2*: *OUTfromChCorrect data2*
⟨*proof*⟩

lemma *OUTfromChCorrect-data3*: *OUTfromChCorrect data3*
⟨*proof*⟩

lemma *OUTfromChCorrect-data4*: *OUTfromChCorrect data4*
⟨*proof*⟩

lemma *OUTfromChCorrect-data5*: *OUTfromChCorrect data5*

⟨*proof*⟩

lemma *OUTfromChCorrect-data6*: *OUTfromChCorrect data6*
⟨*proof*⟩

lemma *OUTfromChCorrect-data7*: *OUTfromChCorrect data7*
⟨*proof*⟩

lemma *OUTfromChCorrect-data8*: *OUTfromChCorrect data8*
⟨*proof*⟩

lemma *OUTfromChCorrect-data9*: *OUTfromChCorrect data9*
⟨*proof*⟩

lemma *OUTfromChCorrect-data10*: *OUTfromChCorrect data10*
⟨*proof*⟩

lemma *OUTfromChCorrect-data11*: *OUTfromChCorrect data11*
⟨*proof*⟩

lemma *OUTfromChCorrect-data12*: *OUTfromChCorrect data12*
⟨*proof*⟩

lemma *OUTfromChCorrect-data13*: *OUTfromChCorrect data13*
⟨*proof*⟩

lemma *OUTfromChCorrect-data14*: *OUTfromChCorrect data14*
⟨*proof*⟩

lemma *OUTfromChCorrect-data15*: *OUTfromChCorrect data15*
⟨*proof*⟩

lemma *OUTfromChCorrect-data16*: *OUTfromChCorrect data16*
⟨*proof*⟩

lemma *OUTfromChCorrect-data17*: *OUTfromChCorrect data17*
⟨*proof*⟩

lemma *OUTfromChCorrect-data18*: *OUTfromChCorrect data18*
⟨*proof*⟩

lemma *OUTfromChCorrect-data19*: *OUTfromChCorrect data19*
⟨*proof*⟩

lemma *OUTfromChCorrect-data20*: *OUTfromChCorrect data20*
⟨*proof*⟩

lemma *OUTfromChCorrect-data21*: *OUTfromChCorrect data21*

⟨*proof*⟩

lemma *OUTfromChCorrect-data22*: *OUTfromChCorrect data22*
⟨*proof*⟩

lemma *OUTfromChCorrect-data23*: *OUTfromChCorrect data23*
⟨*proof*⟩

lemma *OUTfromChCorrect-data24*: *OUTfromChCorrect data24*
⟨*proof*⟩

lemma *OUTfromChCorrectSYSTEM-holds*: *OUTfromChCorrectSYSTEM*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data1*: *OUTfromVCorrect1 data1*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data2*: *OUTfromVCorrect1 data2*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data3*: *OUTfromVCorrect1 data3*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data4*: *OUTfromVCorrect1 data4*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data5*: *OUTfromVCorrect1 data5*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data6*: *OUTfromVCorrect1 data6*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data7*: *OUTfromVCorrect1 data7*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data8*: *OUTfromVCorrect1 data8*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data9*: *OUTfromVCorrect1 data9*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data10*: *OUTfromVCorrect1 data10*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data11*: *OUTfromVCorrect1 data11*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data12*: *OUTfromVCorrect1 data12*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data13*: *OUTfromVCorrect1 data13*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data14*: *OUTfromVCorrect1 data14*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data15*: *OUTfromVCorrect1 data15*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data16*: *OUTfromVCorrect1 data16*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data17*: *OUTfromVCorrect1 data17*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data18*: *OUTfromVCorrect1 data18*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data19*: *OUTfromVCorrect1 data19*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data20*: *OUTfromVCorrect1 data20*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data21*: *OUTfromVCorrect1 data21*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data22*: *OUTfromVCorrect1 data22*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data23*: *OUTfromVCorrect1 data23*
⟨*proof*⟩

lemma *OUTfromVCorrect1-data24*: *OUTfromVCorrect1 data24*
⟨*proof*⟩

lemma *OUTfromVCorrect1SYSTEM-holds*: *OUTfromVCorrect1SYSTEM*
⟨*proof*⟩

lemma *OUTfromVCorrect2SYSTEM*: *OUTfromVCorrect2SYSTEM*
⟨*proof*⟩

lemma *OUTfromV-VARto-holds*:
OUTfromV-VARto
⟨*proof*⟩

lemma *VARfromCorrectSYSTEM-holds*:
VARfromCorrectSYSTEM

⟨*proof*⟩

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

lemma *VARusefulSYSTEM-holds*:
VARusefulSYSTEM
⟨*proof*⟩

4.3 Elementary components

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

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

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

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

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

lemma *elementaryCompDD-sA5*: *elementaryCompDD sA5*
⟨*proof*⟩

lemma *elementaryCompDD-sA6*: *elementaryCompDD sA6*
⟨*proof*⟩

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

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

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

lemma *elementaryCompDD-sA11*: *elementaryCompDD sA11*
⟨*proof*⟩

lemma *elementaryCompDD-sA12*: *elementaryCompDD sA12*
⟨*proof*⟩

lemma *elementaryCompDD-sA21*: *elementaryCompDD sA21*
⟨*proof*⟩

lemma *elementaryCompDD-sA22*: *elementaryCompDD sA22*
⟨*proof*⟩

lemma *elementaryCompDD-sA23*: *elementaryCompDD sA23*
⟨*proof*⟩

lemma *elementaryCompDD-sA31*: *elementaryCompDD sA31*
⟨*proof*⟩

lemma *elementaryCompDD-sA32*: *elementaryCompDD sA32*
⟨*proof*⟩

lemma *elementaryCompDD-sA41*: *elementaryCompDD sA41*
⟨*proof*⟩

lemma *elementaryCompDD-sA42*: *elementaryCompDD sA42*
⟨*proof*⟩

lemma *elementaryCompDD-sA71*: *elementaryCompDD sA71*
⟨*proof*⟩

lemma *elementaryCompDD-sA72*: *elementaryCompDD sA72*
⟨*proof*⟩

lemma *elementaryCompDD-sA81*: *elementaryCompDD sA81*
⟨*proof*⟩

lemma *elementaryCompDD-sA82*: *elementaryCompDD sA82*
⟨*proof*⟩

lemma *elementaryCompDD-sA91*: *elementaryCompDD sA91*
⟨*proof*⟩

lemma *elementaryCompDD-sA92*: *elementaryCompDD sA92*
⟨*proof*⟩

lemma *elementaryCompDD-sA93*: *elementaryCompDD sA93*
⟨*proof*⟩

4.4 Source components

— Abstraction level 0

lemma *A5-NotDSource-level0*: *isNotDSource level0 sA5*
⟨*proof*⟩

lemma *DSourcesA1-L0*: *DSources level0 sA1* = {}

⟨*proof*⟩

lemma *DSourcesA2-L0*: *DSources level0 sA2* = { *sA1*, *sA4*}
⟨*proof*⟩

lemma *DSourcesA3-L0*: *DSources level0 sA3* = { *sA2* }
⟨*proof*⟩

lemma *DSourcesA4-L0*: *DSources level0 sA4* = { *sA3* }
⟨*proof*⟩

lemma *DSourcesA5-L0*: *DSources level0 sA5* = { *sA4* }
⟨*proof*⟩

lemma *DSourcesA6-L0*: *DSources level0 sA6* = {}
⟨*proof*⟩

lemma *DSourcesA7-L0*: *DSources level0 sA7* = {*sA6*}
⟨*proof*⟩

lemma *DSourcesA8-L0*: *DSources level0 sA8* = {*sA7*, *sA9*}
⟨*proof*⟩

lemma *DSourcesA9-L0*: *DSources level0 sA9* = {*sA8*}
⟨*proof*⟩

lemma *A1-DAcc-level0*: *DAcc level0 sA1* = { *sA2* }
⟨*proof*⟩

lemma *A2-DAcc-level0*: *DAcc level0 sA2* = { *sA3* }
⟨*proof*⟩

lemma *A3-DAcc-level0*: *DAcc level0 sA3* = { *sA4* }
⟨*proof*⟩

lemma *A4-DAcc-level0*: *DAcc level0 sA4* = { *sA2*, *sA5* }
⟨*proof*⟩

lemma *A5-DAcc-level0*: *DAcc level0 sA5* = {}
⟨*proof*⟩

lemma *A6-DAcc-level0*: *DAcc level0 sA6* = { *sA7* }
⟨*proof*⟩

lemma *A7-DAcc-level0*: *DAcc level0 sA7* = { *sA8* }
⟨*proof*⟩

lemma *A8-DAcc-level0*: *DAcc level0 sA8* = { *sA9* }
⟨*proof*⟩

lemma *A9-DAcc-level0*: $DAcc\ level0\ sA9 = \{\ sA8\ \}$
⟨*proof*⟩

lemma *A8-NSources*:
$\forall\ C \in (AbstrLevel\ level0).\ (C \neq sA9 \wedge C \neq sA8 \longrightarrow sA8 \notin (Sources\ level0\ C))$
⟨*proof*⟩

lemma *A9-NSources*:
$\forall\ C \in (AbstrLevel\ level0).\ (C \neq sA9 \wedge C \neq sA8 \longrightarrow sA9 \notin (Sources\ level0\ C))$
⟨*proof*⟩

lemma *A7-Acc*:
$(Acc\ level0\ sA7) = \{sA8,\ sA9\}$
⟨*proof*⟩

lemma *A7-NSources*:
$\forall\ C \in (AbstrLevel\ level0).\ (C \neq sA9 \wedge C \neq sA8 \longrightarrow sA7 \notin (Sources\ level0\ C))$
⟨*proof*⟩

lemma *A5-Acc*: $(Acc\ level0\ sA5) = \{\}$
⟨*proof*⟩

lemma *A6-Acc*:
$(Acc\ level0\ sA6) = \{sA7,\ sA8,\ sA9\}$
⟨*proof*⟩

lemma *A6-NSources*:
$\forall\ C \in (AbstrLevel\ level0).\ (C \neq sA9 \wedge C \neq sA8 \wedge C \neq sA7 \longrightarrow sA6 \notin (Sources\ level0\ C))$
⟨*proof*⟩

lemma *SourcesA1-L0*: $Sources\ level0\ sA1 = \{\}$
⟨*proof*⟩

lemma *SourcesA2-L0*: $Sources\ level0\ sA2 = \{\ sA1,\ sA2,\ sA3,\ sA4\ \}$
⟨*proof*⟩

lemma *SourcesA3-L0*: $Sources\ level0\ sA3 = \{\ sA1,\ sA2,\ sA3,\ sA4\ \}$
⟨*proof*⟩

lemma *SourcesA4-L0*: $Sources\ level0\ sA4 = \{\ sA1,\ sA2,\ sA3,\ sA4\ \}$
⟨*proof*⟩

lemma *SourcesA5-L0*: $Sources\ level0\ sA5 = \{\ sA1,\ sA2,\ sA3,\ sA4\ \}$
⟨*proof*⟩

lemma *SourcesA6-L0*: $Sources\ level0\ sA6 = \{\}$
⟨*proof*⟩

lemma *SourcesA7-L0*: *Sources level0 sA7* = { *sA6* }
⟨*proof*⟩

lemma *SourcesA8-L0*: *Sources level0 sA8* = { *sA6*, *sA7*, *sA8*, *sA9* }
⟨*proof*⟩

lemma *SourcesA9-L0*: *Sources level0 sA9* = { *sA6*, *sA7*, *sA8*, *sA9* }
⟨*proof*⟩

lemma *A12-NotSource-level1*: *isNotDSource level1 sA12*
⟨*proof*⟩

lemma *A21-NotSource-level1*: *isNotDSource level1 sA21*
⟨*proof*⟩

lemma *A5-NotSource-level1*: *isNotDSource level1 sA5*
⟨*proof*⟩

lemma *A92-NotSource-level1*: *isNotDSource level1 sA92*
⟨*proof*⟩

lemma *A93-NotSource-level1*: *isNotDSource level1 sA93*
⟨*proof*⟩

lemma *A11-DAcc-level1*: *DAcc level1 sA11* = { *sA21*, *sA22*, *sA23* }
⟨*proof*⟩

lemma *A12-DAcc-level1*: *DAcc level1 sA12* = {}
⟨*proof*⟩

lemma *A21-DAcc-level1*: *DAcc level1 sA21* = {}
⟨*proof*⟩

lemma *A22-DAcc-level1*: *DAcc level1 sA22* = {*sA31*}
⟨*proof*⟩

lemma *A23-DAcc-level1*: *DAcc level1 sA23* = {*sA32*}
⟨*proof*⟩

lemma *A31-DAcc-level1*: *DAcc level1 sA31* = {*sA41*}
⟨*proof*⟩

lemma *A32-DAcc-level1*: *DAcc level1 sA32* = {*sA41*}
⟨*proof*⟩

lemma *A41-DAcc-level1*: *DAcc level1 sA41* = {*sA22*}

⟨*proof*⟩

lemma *A42-DAcc-level1*: $DAcc\ level1\ sA42 = \{sA5\}$
⟨*proof*⟩

lemma *A5-DAcc-level1*: $DAcc\ level1\ sA5 = \{\}$
⟨*proof*⟩

lemma *A6-DAcc-level1*: $DAcc\ level1\ sA6 = \{sA71,\ sA72\}$
⟨*proof*⟩

lemma *A71-DAcc-level1*: $DAcc\ level1\ sA71 = \{sA81\}$
⟨*proof*⟩

lemma *A72-DAcc-level1*: $DAcc\ level1\ sA72 = \{sA82\}$
⟨*proof*⟩

lemma *A81-DAcc-level1*: $DAcc\ level1\ sA81 = \{sA91,\ sA92\}$
⟨*proof*⟩

lemma *A82-DAcc-level1*: $DAcc\ level1\ sA82 = \{sA93\}$
⟨*proof*⟩

lemma *A91-DAcc-level1*: $DAcc\ level1\ sA91 = \{sA81\}$
⟨*proof*⟩

lemma *A92-DAcc-level1*: $DAcc\ level1\ sA92 = \{\}$
⟨*proof*⟩

lemma *A93-DAcc-level1*: $DAcc\ level1\ sA93 = \{\}$
⟨*proof*⟩

lemma *A42-NSources-L1*:
$\forall\ C \in (AbstrLevel\ level1).\ C \neq sA5 \longrightarrow sA42 \notin (Sources\ level1\ C)$
⟨*proof*⟩

lemma *A5-NotSourceSet-level1* :
$\forall\ C\ \in (AbstrLevel\ level1).\ sA5 \notin (Sources\ level1\ C)$
⟨*proof*⟩

lemma *A92-NotSourceSet-level1* :
$\forall\ C\ \in (AbstrLevel\ level1).\ sA92 \notin (Sources\ level1\ C)$
⟨*proof*⟩

lemma *A93-NotSourceSet-level1* :
$\forall\ C\ \in (AbstrLevel\ level1).\ sA93 \notin (Sources\ level1\ C)$
⟨*proof*⟩

lemma *DSourcesA11-L1*: *DSources level1 sA11* = {}
⟨*proof*⟩

lemma *DSourcesA12-L1*: *DSources level1 sA12* = {}
⟨*proof*⟩

lemma *DSourcesA21-L1*: *DSources level1 sA21* = {*sA11*}
⟨*proof*⟩

lemma *DSourcesA22-L1*: *DSources level1 sA22* = {*sA11*, *sA41*}
⟨*proof*⟩

lemma *DSourcesA23-L1*: *DSources level1 sA23* = {*sA11*}
⟨*proof*⟩

lemma *DSourcesA31-L1*: *DSources level1 sA31* = { *sA22* }
⟨*proof*⟩

lemma *DSourcesA32-L1*: *DSources level1 sA32* = { *sA23* }
⟨*proof*⟩

lemma *DSourcesA41-L1*: *DSources level1 sA41* = { *sA31*, *sA32* }
⟨*proof*⟩

lemma *DSourcesA42-L1*: *DSources level1 sA42* = {}
⟨*proof*⟩

lemma *DSourcesA5-L1*: *DSources level1 sA5* = { *sA42* }
⟨*proof*⟩

lemma *DSourcesA6-L1*: *DSources level1 sA6* = {}
⟨*proof*⟩

lemma *DSourcesA71-L1*: *DSources level1 sA71* = { *sA6* }
⟨*proof*⟩

lemma *DSourcesA72-L1*: *DSources level1 sA72* = { *sA6* }
⟨*proof*⟩

lemma *DSourcesA81-L1*: *DSources level1 sA81* = { *sA71*, *sA91* }
⟨*proof*⟩

lemma *DSourcesA82-L1*: *DSources level1 sA82* = { *sA72* }
⟨*proof*⟩

lemma *DSourcesA91-L1*: *DSources level1 sA91* = { *sA81* }
⟨*proof*⟩

lemma *DSourcesA92-L1*: *DSources level1 sA92* = { *sA81* }

⟨*proof*⟩

lemma *DSourcesA93-L1*: *DSources level1 sA93* = { *sA82* }
⟨*proof*⟩

lemma *A82-Acc*: (*Acc level1 sA82*) = {*sA93*}
⟨*proof*⟩

lemma *A82-NSources-L1*:
∀ *C* ∈ (*AbstrLevel level1*). (*C* ≠ *sA93* ⟶ *sA82* ∉ (*Sources level1 C*))
⟨*proof*⟩

lemma *A72-Acc*: (*Acc level1 sA72*) = {*sA82*, *sA93*}
⟨*proof*⟩

lemma *A72-NSources-L1*:
∀ *C* ∈ (*AbstrLevel level1*). (*C* ≠ *sA93* ∧ *C* ≠ *sA82* ⟶ *sA72* ∉ (*Sources level1 C*))
⟨*proof*⟩

lemma *A92-Acc*: (*Acc level1 sA92*) = {}
⟨*proof*⟩

lemma *A92-NSources-L1*:
∀ *C* ∈ (*AbstrLevel level1*). (*sA92* ∉ (*Sources level1 C*))
⟨*proof*⟩

lemma *A91-Acc*: (*Acc level1 sA91*) = {*sA81*, *sA91*, *sA92*}
⟨*proof*⟩

lemma *A91-NSources-L1*:
∀ *C* ∈ (*AbstrLevel level1*). (*C* ≠ *sA92* ∧ *C* ≠ *sA91* ∧ *C* ≠ *sA81* ⟶ *sA91* ∉ (*Sources level1 C*))
⟨*proof*⟩

lemma *A81-Acc*: (*Acc level1 sA81*) = {*sA81*, *sA91*, *sA92*}
⟨*proof*⟩

lemma *A81-NSources-L1*:
∀ *C* ∈ (*AbstrLevel level1*). (*C* ≠ *sA92* ∧ *C* ≠ *sA91* ∧ *C* ≠ *sA81* ⟶ *sA81* ∉ (*Sources level1 C*))
⟨*proof*⟩

lemma *A71-Acc*: (*Acc level1 sA71*) = {*sA81*, *sA91*, *sA92*}
⟨*proof*⟩

lemma *A71-NSources-L1*:
∀ *C* ∈ (*AbstrLevel level1*). (*C* ≠ *sA92* ∧ *C* ≠ *sA91* ∧ *C* ≠ *sA81* ⟶ *sA71* ∉ (*Sources level1 C*))

⟨*proof*⟩

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

lemma *A6-NSources-L1Acc*:
$\forall\ C \in (AbstrLevel\ level1).\ (C \notin (Acc\ level1\ sA6) \longrightarrow sA6 \notin (Sources\ level1\ C))$
⟨*proof*⟩

lemma *A6-NSources-L1*:
$\forall\ C \in (AbstrLevel\ level1).\ (C \neq sA93 \wedge C \neq sA92 \wedge C \neq sA91 \wedge C \neq sA82 \wedge C \neq sA81 \wedge C \neq sA72 \wedge C \neq sA71 \longrightarrow sA6 \notin (Sources\ level1\ C))$
⟨*proof*⟩

lemma *A5-Acc-L1*: $(Acc\ level1\ sA5) = \{\}$
⟨*proof*⟩

lemma *SourcesA11-L1*: $Sources\ level1\ sA11 = \{\}$
⟨*proof*⟩

lemma *SourcesA12-L1*: $Sources\ level1\ sA12 = \{\}$
⟨*proof*⟩

lemma *SourcesA21-L1*: $Sources\ level1\ sA21 = \{sA11\}$
⟨*proof*⟩

lemma *SourcesA22-L1*: $Sources\ level1\ sA22 = \{sA11, sA22, sA23, sA31, sA32, sA41\}$
⟨*proof*⟩

lemma *SourcesA23-L1*: $Sources\ level1\ sA23 = \{sA11\}$
⟨*proof*⟩

lemma *SourcesA31-L1*: $Sources\ level1\ sA31 = \{sA11, sA22, sA23, sA31, sA32, sA41\}$
⟨*proof*⟩

lemma *SourcesA32-L1*: $Sources\ level1\ sA32 = \{sA11, sA23\}$
⟨*proof*⟩

lemma *SourcesA41-L1*: $Sources\ level1\ sA41 = \{sA11, sA22, sA23, sA31, sA32, sA41\}$
⟨*proof*⟩

lemma *SourcesA42-L1*: $Sources\ level1\ sA42 = \{\}$

⟨*proof*⟩

lemma *SourcesA5-L1*: *Sources level1 sA5* = {*sA42*}
⟨*proof*⟩

lemma *SourcesA6-L1*: *Sources level1 sA6* = {}
⟨*proof*⟩

lemma *SourcesA71-L1*: *Sources level1 sA71* = {*sA6*}
⟨*proof*⟩

lemma *SourcesA81-L1*: *Sources level1 sA81* = {*sA6*, *sA71*, *sA81*, *sA91*}
⟨*proof*⟩

lemma *SourcesA91-L1*: *Sources level1 sA91* = {*sA6*, *sA71*, *sA81*, *sA91*}
⟨*proof*⟩

lemma *SourcesA92-L1*: *Sources level1 sA92* = {*sA6*, *sA71*, *sA81*, *sA91*}
⟨*proof*⟩

lemma *SourcesA72-L1*: *Sources level1 sA72* = {*sA6*}
⟨*proof*⟩

lemma *SourcesA82-L1*: *Sources level1 sA82* = {*sA6*, *sA72*}
⟨*proof*⟩

lemma *SourcesA93-L1*: *Sources level1 sA93* = {*sA6*, *sA72*, *sA82*}
⟨*proof*⟩

lemma *SourcesS1-L2*: *Sources level2 sS1* = {}
⟨*proof*⟩

lemma *SourcesS2-L2*: *Sources level2 sS2* = {}
⟨*proof*⟩

lemma *SourcesS3-L2*: *Sources level2 sS3* = {*sS2*}
⟨*proof*⟩

lemma *SourcesS4-L2*: *Sources level2 sS4* = {*sS2*}
⟨*proof*⟩

lemma *SourcesS5-L2*: *Sources level2 sS5* = {*sS2*, *sS4*}
⟨*proof*⟩

lemma *SourcesS6-L2*: *Sources level2 sS6* = {*sS2*, *sS4*, *sS5*}
⟨*proof*⟩

lemma *SourcesS7-L2*: *Sources level2 sS7* = {}

⟨*proof*⟩

lemma *SourcesS8-L2*:
Sources level2 sS8 = {*sS7*}
⟨*proof*⟩

lemma *SourcesS9-L2*:
Sources level2 sS9 = {}
⟨*proof*⟩

lemma *SourcesS10-L2*: *Sources level2 sS10* = {*sS9*}
⟨*proof*⟩

lemma *SourcesS11-L2*: *Sources level2 sS11* = {*sS9*}
⟨*proof*⟩

lemma *SourcesS12-L2*: *Sources level2 sS12* = {*sS9*, *sS10*}
⟨*proof*⟩

lemma *SourcesS13-L2*: *Sources level2 sS13* = {*sS9*, *sS10*, *sS12*}
⟨*proof*⟩

lemma *SourcesS14-L2*: *Sources level2 sS14* = {*sS9*, *sS11*}
⟨*proof*⟩

lemma *SourcesS15-L2*: *Sources level2 sS15* = {*sS9*, *sS11*, *sS14*}
⟨*proof*⟩

4.5 Minimal sets of components to prove certain properties

lemma *minSetOfComponentsTestL2p1*:
minSetOfComponents level2 {*data10*, *data13*} = {*sS1*}
⟨*proof*⟩

lemma *NOT-noIrrelevantChannelsTestL2p1*:
¬ *noIrrelevantChannels level2* {*data10*, *data13*}
⟨*proof*⟩

lemma *NOT-allNeededINChannelsTestL2p1*:
¬ *allNeededINChannels level2* {*data10*, *data13*}
⟨*proof*⟩

lemma *minSetOfComponentsTestL2p2*:
minSetOfComponents level2 {*data1*, *data12*} = {*sS2*, *sS4*, *sS5*, *sS6*}
⟨*proof*⟩

lemma *noIrrelevantChannelsTestL2p2*:
noIrrelevantChannels level2 {*data1*, *data12*}
⟨*proof*⟩

lemma *allNeededINChannelsTestL2p2*:
allNeededINChannels level2 {*data1*, *data12*}
⟨*proof*⟩

lemma *minSetOfComponentsTestL1p3*:
minSetOfComponents level1 {*data1*, *data10*, *data11*} = {*sA12*, *sA11*, *sA21*}
⟨*proof*⟩

lemma *noIrrelevantChannelsTestL1p3*:
noIrrelevantChannels level1 {*data1*, *data10*, *data11*}
⟨*proof*⟩

lemma *allNeededINChannelsTestL1p3*:
allNeededINChannels level1 {*data1*, *data10*, *data11*}
⟨*proof*⟩

lemma *minSetOfComponentsTestL2p3*:
minSetOfComponents level2 {*data1*, *data10*, *data11*} = {*sS1*, *sS2*, *sS3*}
⟨*proof*⟩

lemma *noIrrelevantChannelsTestL2p3*:
noIrrelevantChannels level2 {*data1*, *data10*, *data11*}
⟨*proof*⟩

lemma *allNeededINChannelsTestL2p3*:
allNeededINChannels level2 {*data1*, *data10*, *data11*}
⟨*proof*⟩

end

References

[1] J. Barnat, J. Chaloupka, and J. van de Pol. Improved distributed algorithms for scc decomposition. *Electron. Notes Theor. Comput. Sci.*, 198(1):63–77, 2008.

[2] L. Fleischer, B. Hendrickson, and A. Pınar. On identifying strongly connected components in parallel. In J. Rolim, editor, *Parallel and Distributed Processing*, volume 1800 of *LNCS*, pages 505–511. Springer, 2000.

[3] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[4] S. M. Orzan. *On Distributed Verification and Verified Distribution*. PhD thesis, Free University of Amsterdam, 2004.

[5] M. Spichkova. Architecture: Requirements + Decomposition + Refinement. *Softwaretechnik-Trends*, 31:4, 2011.