PROGRAM COMPOSITION via UNIFICATION*

Limor Fix **Nissim Francez** **Orna Grumberg**
Computer Science Department, Technion, Haifa 32000, Israel

Abstract

Program composition and compositional proof systems have proven themselves important for simplifying the design and the verification of programs. The paper presents a version of the *jigsaw* program composition operator previously defined in [FFG90, FFG91]. Here, the jigsaw operator is defined as the unification of its components by their most general unifier. The jigsaw operator generalizes and unifies the traditional *sequential* and *parallel* program composition operators and the newly proposed *union* and *superposition* operators. We consider a family of frameworks each consisting of a programming language, a specification language and a compositional syntax-directed proof system. We present syntactic rules to augment any given framework in the family with the jigsaw operator. The augmented framework is syntax-directed and compositional. Moreover, it is sound and complete with respect to the given framework.

1 Introduction

Program composition and compositional proof systems have proven themselves important for simplifying the design and verification of programs. In this approach, small parts of a software system are programmed and verified separately and then composed into a larger system. To develop a system such that its syntactic structure reflects the intuitive structure of the algorithm it implements, program composition operators able to reflect common relations among components of algorithms are required. Traditional program composition operators are the sequential and the parallel composition operators. Recently, other program composition operators have been suggested [BF88, CM88, FF90, KFE90].

In this paper a composition operator, called *jigsaw*, is considered. The jigsaw composition operator generalizes and unifies the *sequential* and the *parallel* operators together with the newly proposed *union* and *superposition* operators [BF88,CM88].

Two versions of the jigsaw operator have already been presented and studied in [FFG90, FFG91]. Both versions present a composition operator which composes *gapped programs*. A gapped program is a program in the language under consideration which includes scattered unspecified statements, to be called *gaps*. The jigsaw operator composes two gapped programs by "filling" gaps in a component with statements taken from the other component. In [FFG90] the jigsaw operator is added to a CSP-like programming language [Hoa78] and a proof rule for

*A preliminary version of this paper appears in [FFG92].

partial correctness of a jigsaw-composed program is presented. In [FFG91] the jigsaw operator is added to a UNITY-like programming language and rules for proving partial correctness and termination of a jigsaw-composed program are presented. In [FFG91] union and superposition are shown to be special cases of the jigsaw composition.

Here, the jigsaw composition of two gapped programs is defined as their **unification** by their most general unifier. A gap in a program is represented by a (meta) variable ranging over programs. A gapped program is a term built over primitive objects, variables and composition operators. The jigsaw composition of two gapped programs T_1 and T_2 is well-defined if and only if those programs are unifiable. The unification procedure simultaneously "fills" gaps of T_1 with sub-programs of T_2 and gaps of T_2 with sub-programs of T_1. The jigsaw supports development of programs which are syntactically structured according to the intuitive structure of the algorithm they implement. Therefore, augmenting an existing framework with the jigsaw is desirable.

We present a method to extend any given compositional syntax-directed proof system with the jigsaw operator. An abstract framework is presented such that any concrete proof system which fits the framework is syntactically extended to include the jigsaw. First, a method to extend the basic programming language with the jigsaw operator is presented. Then, mixed-specifications [Wir71, Bac80, Heh84, Old86] are defined over the basic specifications and constructors. Gapped programs are specified by mixed-specifications. A mixed specification has two syntactic parts, a *promise* and a *conclusion*. The promise introduces assumptions on all gaps of the specified gapped program and the conclusion specifies the behavior of the gapped program under the assumptions imposed by the promise. Finally, a method is presented to extend the basic proof system into a system in which satisfaction of specification can be proved for gapped programs. The extended proof system is syntax-directed and compositional, i.e., a specification for a jigsaw-composed program is verified based on the specifications of its components.

The development of a program in the extended system goes through intermediate stages in which gapped programs are built and verified. The program at the final stage must be a program in which all gaps are bound by the jigsaw operators in the program to gap-free sub-programs. The program at the final stage is therefore always semantically equal to a program in the basic system. The main theorem of the paper states that the extended system is sound and complete with respect to the basic system. More formally, given two programs, T in the basic system and $T^{'}$ in the extended one, such that $T^{'}$ is semantically equal to T, we prove that if a specification S can be verified for $T^{'}$ then S is semantically true for T, and if S is semantically true for T then it can be verified for $T^{'}$.

We present an example of a concrete proof system which fits the abstract framework, and syntactically derive an extended version of it containing the jigsaw operator. The concrete proof system is a system for proving partial correctness for CSP-like programs [ZRB85]. The specification language is a two sorted first order predicate language. In the extended system we develop a small program consisting of a *Producer* and a *Consumer*. Using the jigsaw operator we separate the development and the verification of the communication management part from the development and verification of the other functions of the program.

The rest of the paper is organized as follows. Section 2 presents the abstract basic system. Section 3 introduces the jigsaw-extended system. Section 4 contains an example of extending a concrete system with the jigsaw operator. Finally, Section 5 ends with conclusions.

2 The basic system

We follow [KKZ90] and present an abstract framework for the development and verification of programs. This framework forms the starting point of our work, to be called the *basic* abstract framework. The framework is defined by introducing programs (in a common algebraic setup), specifications and correctness formulas over programs and specifications. In addition, a compositional syntax-directed proof system to derive correctness formulas is defined.

Definition: Let $\mathcal{O}$ be a set of primitive objects and let $\mathcal{C}$ be a set of constructors, where $C \in \mathcal{C}$ is of finite and fixed arity. The set of *programs* (terms), $\mathcal{T}$, defined over $\mathcal{O}$ and $\mathcal{C}$ is the smallest set s.t. $\mathcal{O} \subseteq \mathcal{T}$ and if $T_1, \ldots, T_n \in \mathcal{T}$, $C \in \mathcal{C}$ and C is of arity n then $C(T_1, \ldots, T_n) \in \mathcal{T}$. □

Let $\mathcal{B}$ be a set of semantic objects (behaviors). Let $\widehat{\wp(\mathcal{B})} \subseteq \wp(\mathcal{B})$ be a subset of the power set of $\mathcal{B}$ which has a $\subseteq$-minimal element, denoted by Θ.

Definition: For $\mathcal{T}$ and $\mathcal{B}$ as above, the *semantics* of $\mathcal{T}$ is given by a function $BEH_{\mathcal{T}} : \mathcal{T} \to \widehat{\wp(\mathcal{B})}$. Such a semantics is *compositional* if for every $C \in \mathcal{C}$ there exists a total function $K_C : (\widehat{\wp(\mathcal{B})})^n \to \widehat{\wp(\mathcal{B})}$ s.t. $BEH_{\mathcal{T}}[C(T_1, \ldots, T_n)] = K_C(BEH_{\mathcal{T}}[T_1], \ldots, BEH_{\mathcal{T}}[T_n])$. □

The structure of the basic specifications is not explicitly defined and we assume the existence of a semantic function which maps the specifications into the set $\widehat{\wp(\mathcal{B})}$. Moreover, we assume that a specification does not refer to any program and the association between a specification and a program is made only in the correctness formulas (defined in the sequel).

Definition: Let $\mathcal{S}$ be a set of specifications closed under conjunction, $\wedge$. The *semantics* of $\mathcal{S}$ is given by a function $BEH_{\mathcal{S}} : \mathcal{S} \to \widehat{\wp(\mathcal{B})}$, such that: $BEH_{\mathcal{S}}[S_1 \wedge S_2] = BEH_{\mathcal{S}}[S_1] \bigcap BEH_{\mathcal{S}}[S_2]$. A specification S is a *tautology* if $BEH_{\mathcal{S}}[S] = \mathcal{B}$. □

Since both programs and specifications are mapped to the same set of behaviors, $\widehat{\wp(\mathcal{B})}$, we can relate a program to a specification by relating their respective behaviors.

Definition:

1. The set Φ of *correctness formulas* is the set of formulas of the form "$T\,Sat\,S$", where $T \in \mathcal{T}$ and $S \in \mathcal{S}$.
2. A *structure* is a triple $M =< \mathcal{B}, BEH_{\mathcal{T}}, BEH_{\mathcal{S}} >$, where $\mathcal{B}$ is a set of behaviors and $BEH_{\mathcal{T}}$ and $BEH_{\mathcal{S}}$ are semantic functions for $\mathcal{T}$ and $\mathcal{S}$ over the same domain, $\widehat{\wp(\mathcal{B})}$.
3. A formula "$T\,Sat\,S$"$\in \Phi$ *holds* in a *structure* M, denoted by $M \models T\,Sat\,S$, iff $BEH_{\mathcal{T}}[T] \subseteq BEH_{\mathcal{S}}[S]$. When there is no confusion we write $\models T\ Sat\ S$.

Definition: A *compositional proof system* $\mathcal{AR} =< \mathcal{A}, \mathcal{R}, \mathcal{G} >$ for Φ consists of:

1. A (recursive) set $\mathcal{A} \subseteq \Phi$ of formulas called *axioms*, s.t. if "$T\ Sat\ S$" $\in \mathcal{A}$ then $T \in \mathcal{O}$.
2. A set $\mathcal{G}$ of predicates s.t. for every $C \in \mathcal{C}$ of arity n, there is $G_C \in \mathcal{G}$ on $\mathcal{S}^{n+1}$.
3. A set $\mathcal{R}$ of *proof-rules*, s.t. $\forall C \in \mathcal{C}$ there exists a unique rule $R_C \in \mathcal{R}$ of the form:

$$\frac{T_i\ Sat\ S_i,\ i = 1, n,\ \ G_C(S_1, \ldots, S_n, S)}{C(T_1, \ldots, T_n)\ Sat\ S}$$

4. A way to determine whether a predicate in $G_C \in \mathcal{G}$ of arity n holds for a set of specifications $S_1, \ldots, S_n$.

Definition: A formula $\phi \in \Phi$ is *provable* under no assumption in $\mathcal{AR}$, denoted by $\vdash_{\mathcal{AR}} \phi$, if ϕ can be derived from axioms in $\mathcal{A}$ by applications of rules in $\mathcal{R}$.

Definition: A proof system $\mathcal{AR}$ is *sound* w.r.t. a structure M if for each formula $\phi \in \Phi : \vdash_{\mathcal{AR}} \phi$ implies that $M \models \phi$.

Definition: A proof system $\mathcal{AR}$ is *relatively complete* [1] w.r.t. a structure M if for each formula $\phi \in \Phi : M \models \phi$ implies that $\vdash_{\mathcal{AR}} \phi$.

3 The *jigsaw*-extended system

3.1 The programming language

Next, gapped programs and the jigsaw operator, denoted by $\#$, are presented. Gapped programs are defined by extending the program-algebra with a set of meta variables, $\mathcal{V}$ (with $\mathcal{V} \cap \mathcal{O} = \emptyset$). The program $\#(T_1, T_2)$ is well defined if and only if T_1 and T_2 are *unifiable*, i.e., there exists a substitution θ which binds meta variables to sub-programs, such that applying θ to both T_1 and T_2 results in identical programs, $T_1\theta = T_2\theta$. The meaning of the program $\#(T_1, T_2)$ is defined as the meaning of the program obtained by unifying T_1 and T_2 using their *most general unifier*. A substitution θ is the most general unifier of T_1 and T_2 if and only if $T_1\theta = T_2\theta$ and moreover for every substitution γ, if $T_1\gamma = T_2\gamma$ then there exists a substitution λ such that $\theta o \lambda = \gamma$, where o stands for composition of substitutions (more formally, see Appendix A). Intuitively, the unification of T_1 and T_2 results in a program T which is the *mutual instantiation* of T_1 and T_2, i.e., T can be obtained by a substitution of sub-programs of T_1 for the structurally corresponding meta variables of T_2, and vice versa.

Definition: Let $\mathcal{O}$ be the basic set of primitive objects, let $\mathcal{C}^{\#} = \mathcal{C} \cup \{\#\}$, and let $\mathcal{V}$ be a set of meta variables ranging over programs. The set of *gapped programs*, $\mathcal{T}^{\#}$ (over $\mathcal{O}$ and $\mathcal{C}$), is the smallest set s.t. $\mathcal{O} \subset \mathcal{T}^{\#}$, $\mathcal{V} \subset \mathcal{T}^{\#}$ and if $C \in \mathcal{C}^{\#}$, $T_1, \ldots, T_n \in \mathcal{T}^{\#}$ then $C(T_1, \ldots, T_n) \in \mathcal{T}^{\#}$. [2] $\Box$

Neither a meaning of a meta variable nor a meaning, in the usual way, of a gapped program, are defined. Below, a meaning is assigned to a gapped program in a *context*. A context imposes assumptions on the behaviors of the expected "fillings" of the gaps.

3.2 The specification language and the correctness formulas

Next, a new set of specifications, $\mathcal{S}^{\#}$, and a new set of correctness formulas, are presented. A new correctness formula, of the form "$T\ Sat^{\#}\ S$", binds a gapped program to a specification $S \in \mathcal{S}^{\#}$. In such a formula the specification S defines both a context for T and a characterization of T.

Let T be a gapped program. A meta variable of T is *free* if it is not bound by a jigsaw operator to any sub-program. T can be specified using an *assume-guarantee* [Lam83, Pnu85] type of specification. In such a specification the behavior of T is characterized based on assumptions

[1] We assume, as usual, that all domain properties that are true can be used as additional axioms [Apt81], and that all intermediate assertions needed for the proof can be expressed in $\mathcal{S}$.

[2] Since a variable denotes a gap in a specific location in the program we assume that each variable appears at most once in a given program.

$$
\begin{array}{ll}
T_1 :: C_1(C_2(a_1, a_2), X) & S_1' :: C_1(S_1, \overline{S_2}) \rhd S_3 \\
T_2 :: C_1(Y, a_3) & S_2' :: C_1(\overline{S_4}, S_5) \rhd S_6
\end{array}
$$

Figure 1: Ongoing example

on the behavior of the sub-programs to be substituted for the free variables of T when jigsaw-composed with another program.

We define a set of *assumptions,* $\overline{\mathcal{S}}$, which contains exactly all the specifications of $\mathcal{S}$, upper-lined. The upper-line is used to distinguish an assumption from a basic specification. This distinction is needed to define composition of specifications (more in the sequel).

The new set of specifications, $\mathcal{S}^{\#}$, is a set of *mixed-specifications*. A mixed-specification contains two parts separated by a "$\rhd$" sign. The expression on the right-hand-side of the "$\rhd$" sign is a basic specification, considered as the *conclusion* of the specification. The expression on the left-hand-side of the "$\rhd$" sign is syntactically defined as a member of an algebra with a set $\mathcal{C}^{\#}$ of operators over the sets of basic specifications and assumptions. This expression is considered as the *promise* of the specification.

A gapped program T satisfies a mixed-specification S iff sub-programs of T satisfy the *corresponding* sub-specifications in the promise of S (if any) and T satisfies the conclusion of S under the assumption that the free variables of T (if any) satisfy the corresponding assumptions in the promise of S.

We use the notations in Figure 1 in all the examples presented in this section, where T_1, T_2 are programs and S_1', S_2' are specifications. We start with an example that demonstrates the notion of a gapped program satisfying a mixed-specification.

Example: Let T_1, T_2, S_1' and S_2' be as in Figure 1. The gapped-program T_1 satisfies the mixed-specification S_1' iff $C_2(a_1, a_2)$ satisfies S_1, X is assumed to satisfy $\overline{S_2}$ and moreover T_1 satisfies S_3 under the assumption that X satisfies $\overline{S_2}$.

Next, we exemplify how mixed-specifications enable a compositional verification of a jigsaw-composed program.

Example: Let T_1, T_2, S_1', S_2' be as in Figure 1 and assume the specification S_1' has been verified for T_1 and the specification S_2' has been verified for T_2. The program $\#(T_1, T_2)$ is semantically equal to the program $C_1(C_2(a_1, a_2), a_3)$ obtained by substituting $C_2(a_1, a_2)$ for the meta variable Y and substituting a_3 for the meta variable X. If S_1 and $\overline{S_4}$ are related in such a way that based on the fact that $C_2(a_1, a_2)$ satisfies S_1 we can conclude that $C_2(a_1, a_2)$ also satisfies $\overline{S_4}$, S_1 *confirms* $\overline{S_4}$, then it is possible to conclude that $\#(T_1, T_2)$ satisfies S_6. In a similar way, if S_5 confirms $\overline{S_2}$ then it is possible to conclude that $\#(T_1, T_2)$ satisfies S_3.

We formally define the syntax and the semantics of the new specification language in the following definitions.

Definition:

1. The set $\overline{\mathcal{S}} = \{\overline{S} | S \in \mathcal{S}\}$ is called the *assumption* set. The set $\overline{\mathcal{S}}$ is closed under conjunction and we assume $\overline{S_1} \wedge \overline{S_2} = \overline{S_1 \wedge S_2}$.

2. The set of *mixed-specifications*, $\mathcal{S}^{\#}$, is defined by the following grammar:
   $\mathcal{S}^{\#} ::\ \mathcal{S} \mid \overline{\mathcal{S}} \mid C(\mathcal{S}^{\#}, \ldots, \mathcal{S}^{\#}) \rhd \mathcal{S} \mid \#(\mathcal{S}^{\#}, \mathcal{S}^{\#})$, where $C \in \mathcal{C}$.

3. $S \in \mathcal{S}^{\#}$ is *assumption-free* if it does not contain elements of $\overline{\mathcal{S}}$.

Definition: Let $S_1 \in \mathcal{S}$ and suppose that either $S_2 = \overline{S} \in \overline{\mathcal{S}}$ or $S_2 = S \in \mathcal{S}$. We say that S_1 *confirms* S_2, to be denoted by $S_1 \Rightarrow S_2$, iff $BEH_{\mathcal{S}}[S_1] \subseteq BEH_{\mathcal{S}}[S_2]$. $\Box$

We assume that the basic system provides a way to determine whether $S_1 \Rightarrow S_2$ holds.

Definition: For a mixed-specification $S \in \mathcal{S}^{\#}$, its *conclusion* and *promise*, denoted by $conc(S)$ and $promise(S)$, respectively, are defined as follows:

1. if $S \in \mathcal{S}$ then $conc(S) = S$ and $promise(S)$ is not defined,
2. if $S = \overline{S'} \in \overline{\mathcal{S}}$ then $conc(S) = \overline{S'}$ and $promise(S)$ is not defined,
3. if $S = C(S_1, \ldots, S_n) \rhd S'$ then $conc(S) = S'$ and $promise(S) = C(S_1, \ldots, S_n)$,
4. if $S = \#(S_1, S_2)$ then $conc(S) = conc(S_1) \wedge conc(S_2)$ and $promise(S) = \#(promise(S_1), promise(S_2))$. $\Box$

Example: Let S_1' and S_2' be as in Figure 1 and let $S = \#(S_1', S_2')$, then $promise(S) = \#(C_1(S_1, \overline{S_2}), C_1(\overline{S_4}, S_5))$ and $conc(S) = S_3 \wedge S_6$.

To define the semantics of mixed-specifications, a function $BEH_{\mathcal{S}}^{\#} : \mathcal{S}^{\#} \rightarrow \widehat{\wp(\mathcal{B})}$ is presented. Intuitively, if S' is a "consequence" of $C(S_1, \ldots, S_n)$, i.e., $K_C(BEH_{\mathcal{S}}^{\#}[S_1], \ldots, BEH_{\mathcal{S}}^{\#}[S_n]) \subseteq BEH_{\mathcal{S}}[S']$, then $BEH_{\mathcal{S}}^{\#}[C(S_1, \ldots, S_n) \rhd S']$ is defined as $BEH_{\mathcal{S}}[S']$. Else, it is defined as Θ. If the structures of S_1 and S_2 are unifiable and their assumptions are mutually confirmed then $BEH_{\mathcal{S}}^{\#}[\#(S_1, S_2)]$ is defined as $BEH_{\mathcal{S}}^{\#}[S_1] \cap BEH_{\mathcal{S}}^{\#}[S_2]$. Else, it is defined as Θ. Unification is used to find a match between the assumptions and the specifications that should confirm them. However, here we need to bind specifications to assumptions rather than to variables. We make this precise in the following definitions which lead to the formal definition of the semantics of mixed-specifications.

Definition: Let $\mathcal{V}$ be a set of variables. The set of *skeleton-terms*, $\mathcal{SK}$, is defined by:

$$\mathcal{SK} :: \ \mathcal{S} \mid \mathcal{V} \mid \mathcal{V} \wedge \mathcal{V} \mid C(\mathcal{SK}, \ldots, \mathcal{SK}) \rhd \mathcal{V} \mid \ \#(\mathcal{SK}, \mathcal{SK}), \quad where \ \ C \in \mathcal{C}.$$

Definition: Let $S \in \mathcal{S}^{\#}$ be a mixed-specification. The *skeleton* of S, denoted by $skel(S)$, is a pair in which the first element, denoted by $skel_1(S)$, is a skeleton-term and the second element, denoted by $skel_2(S)$, is a substitution θ which binds basic specifications or assumptions to variables. $skel(S)$ is defined as follows:

1. if $S \in \mathcal{S}$ then $skel(S) =< S, \varepsilon >$,
2. if $S = \overline{S'} \in \overline{\mathcal{S}}$ then $skel(\overline{S'}) =< X, \{X \leftarrow \overline{S'}\} >$, where $X \in \mathcal{V}$,
3. $skel(C(S_1, \ldots, S_n) \rhd S) =< C(skel_1(S_1), \ldots, skel_1(S_n)) \rhd X,\ skel_2(S_1) \bigcup \ldots \bigcup skel_2(S_n) \bigcup \{X \leftarrow S\} >$ [3],
4. $skel(\#(S_1, S_2)) =< \#(skel_1(S_1), skel_1(S_2)), skel_2(S_1) \bigcup skel_2(S_2) >$.

[3]For simplicity assume that $skel_2(S_i)$, $i = 1, n$ have disjoint sets of variables and X is a fresh variable.

Example: Let $S = \#(S_1', S_2')$ where S_1', S_2' are as in Figure 1. Then,

$$\begin{array}{lcl} skel_1(S_1') & = & C_1(S_1, X_2) \rhd X_1 \\ skel_2(S_1') & = & \{X_1 \leftarrow S_3, X_2 \leftarrow \overline{S_2}\} \\ skel_1(S_2') & = & C_1(Y_2, S_5) \rhd Y_1 \\ skel_2(S_2') & = & \{Y_1 \leftarrow S_6, Y_2 \leftarrow \overline{S_4}\} \\ skel_1(\#(S_1', S_2')) & = & \#(C_1(S_1, X_2) \rhd X_1, C_1(Y_2, S_5) \rhd Y_1) \\ skel_2(\#(S_1', S_2')) & = & \{X_1 \leftarrow S_3, X_2 \leftarrow \overline{S_2}, Y_1 \leftarrow S_6, Y_2 \leftarrow \overline{S_4}\} \end{array}$$

Definition: A substitution $\theta_1 = \{X_1 \leftarrow S_1, \ldots, X_n \leftarrow S_n\}$ is *valid* w.r.t. another substitution $\theta_2 = \{Y_1 \leftarrow S_1^{'}, \ldots, Y_m \leftarrow S_m^{'}\}$ if for all $1 \leq i \leq n$, $1 \leq j \leq m$ if $X_i = Y_j$ then $conc(S_i) \Rightarrow conc(S_j^{'})$ holds. □

Next, a transformation on mixed-specifications, $elim : \mathcal{S}^\# \rightarrow \mathcal{S}^\#$, is defined, mapping a mixed-specification containing jigsaw occurrences to a mixed-specification free of jigsaw, with the same meaning (to be defined). The function $elim$ is defined for $\#(S_1, S_2)$ iff S_1 and S_2 are unifiable (see Appendix A) and their assumptions are mutually confirmed. Intuitively, the function $elim$ defines a composition of specifications which is to be used to obtain a specification of the jigsaw-composed program from the specifications of its components.

Definition: Let $S \in \mathcal{S}^\#$ be a mixed-specification. Then $elim(S)$ is defined as follows:

1. if $S \in \mathcal{S} \cup \overline{\mathcal{S}}$ then $elim(S) = S$,
2. $elim(C(S_1, \ldots, S_n) \rhd S^{'})$ is defined to be $C(elim(S_1), \ldots, elim(S_n)) \rhd S^{'}$, if for every $1 \leq i \leq n$, $elim(S_i)$ is defined, and otherwise it is not defined.
3. $elim(\#(S_1, S_2))$ is defined to be $\#(elim(S_1), elim(S_2)) = skel_1(elim(S_1))\theta$, if $skel_1(elim(S_i))$, $i = 1, 2$ are defined and are unifiable [4] and moreover θ is a valid substitution both w.r.t. $skel_2(S_1)$ and w.r.t. $skel_2(S_2)$, where
$\theta = (mgu^*(skel_1(S_1), skel_1(S_2))\; o\; skel_2(S_1)\; o\; skel_2(S_2))$. [5] Otherwise, it is not defined.

If $elim(\#(S_1, S_2))$ is defined, $\#(S_1, S_2)$ is called *well-defined*.

Example: Let $S_1^{'}, S_2^{'}$ be as in Figure 1.

- $mgu^*(skel_1(S_1'), skel_1(S_2')) = \{X_2 \leftarrow S_5, Y_2 \leftarrow S_1, X_1 \leftarrow X_1 \wedge Y_1, Y_1 \leftarrow X_1 \wedge Y_1\}$.
- $\theta = \{X_2 \leftarrow S_5, Y_2 \leftarrow S_1, X_1 \leftarrow S_3 \wedge S_6, Y_1 \leftarrow S_3 \wedge S_6\}$.
- $elim(\#(S_1', S_2')) = skel_1(elim(S_1'))\theta = C_1(S_1, S_5) \rhd S_3 \wedge S_6$, provided that θ is valid for $skel_2(S_1')$, i.e., $S_5 \Rightarrow \overline{S_2}$ and $S_3 \wedge S_6 \Rightarrow S_3$ (which is always true). Moreover, θ is valid for $skel_2(S_2')$, i.e., $S_1 \Rightarrow \overline{S_4}$ and $S_3 \wedge S_6 \Rightarrow S_6$.

Definition: Let $\mathcal{S}^\#$ be the set of mixed-specifications, let $\mathcal{B}$ be the basic set of behaviors and let $BEH_\mathcal{S}$ be the basic semantic function for $\mathcal{S}$. The *semantics* of $\mathcal{S}^\#$ is defined by the function $BEH_\mathcal{S}^\# : S^\# \rightarrow \widehat{\wp(\mathcal{B})}$:

[4] mgu^* is the most general unifier except that if it binds a variable to another variable, for example $X \leftarrow Y$, then this assignment is replaced by $X \leftarrow X \wedge Y$, $Y \leftarrow X \wedge Y$. Intuitively, the assumptions on the corresponding gaps are 'collected'.

[5] The substitution composition in this case is commutative since the substitutions have disjoint sets of variables, therefore, $elim(\#(S_1, S_2)) = elim(\#(S_2, S_1))$.

1. if $S \in \mathcal{S}$ then $BEH^{\#}_{\mathcal{S}}[S] = BEH_{\mathcal{S}}[S]$,

2. if $\overline{S} \in \overline{\mathcal{S}}$ then $BEH^{\#}_{\mathcal{S}}[\overline{S}] = BEH_{\mathcal{S}}[S]$,

3. if $S = C(S_1, \ldots, S_n) \rhd S'$ and $C \neq \#$ then $BEH^{\#}_{\mathcal{S}}[C(S_1, \ldots, S_n) \rhd S'] = BEH^{\#}_{\mathcal{S}}[S']$, if $K_C(BEH^{\#}_{\mathcal{S}}[S_1], \ldots, BEH^{\#}_{\mathcal{S}}[S_n]) \subseteq BEH^{\#}_{\mathcal{S}}[S']$ and the set Θ, otherwise.

4. if $S = \#(S_1, S_2)$ then $BEH^{\#}_{\mathcal{S}}[\#(S_1, S_2)] = BEH^{\#}_{\mathcal{S}}[elim(\#(S_1, S_2))]$, if $\#(S_1, S_2)$ is well-defined and the set Θ, otherwise. □

The meaning of a gapped program T in *context* S is defined by a semantic function $BEH^{\#}_{\mathcal{T}-\mathcal{S}} : \mathcal{T}^{\#} \times \mathcal{S}^{\#} \rightarrow \widehat{\wp(\mathcal{B})}$, to be denoted by $BEH^{\#}_{\mathcal{T}-\mathcal{S}}[T \downarrow_S]$. The meaning of T in context S is defined iff T and S are *identically structured*, i.e., the structural elements in S correspond to the structure of T. Intuitively, if the program T is a basic program then its meaning is defined to be the meaning given to it in the basic system. If T contains a free variable then the meaning of this variable is taken to be the meaning of the corresponding assumption in the promise of S. If T is of the form $\#(T_1, T_2)$ then a meaning is defined for T iff T_1 and T_2 are unifiable. The meaning of $\#(T_1, T_2)$ is defined to be the meaning of the program obtained by unifying T_1 and T_2 using their most general unifier. In order to formally define the meaning of T in context S we need the following definitions.

Definition: Let $T \in \mathcal{T}^{\#}$ and $S \in \mathcal{S}^{\#}$. T and S are *identically structured* iff

1. $T \in \mathcal{T}$ and $S \in \mathcal{S}$, or

2. $T \in \mathcal{V}$ and $S \in \overline{\mathcal{S}}$, or

3. $T = C(T_1, \ldots, T_n)$, $S = C(S_1, \ldots, S_n) \rhd S$ and T_i and S_i are identically structured, $1 \leq i \leq n$, or

4. $T = \#(T_1, T_2)$, $S = \#(S_1, S_2)$ and T_i and S_i are identically structured, $i = 1, 2$.

Example: Let T_1, T_2, S_1' and S_2' be as in Figure 1. T_1 and S_1' are identically structured since $C_2(a_1, a_2)$ and S_1 are identically structured and so are X and $\overline{S_2}$. T_2 and S_2' are identically structured since Y and $\overline{S_4}$ are identically structured and so are a_3 and S_5. Consequently, $\#(T_1, T_2)$ and $\#(S_1', S_2')$ are identically structured.

Definition: Let $T \in \mathcal{T}^{\#}$ be a gapped program. $elim(T)$ is defined by:

1. if $T \in \mathcal{O} \cup \mathcal{V}$ then $elim(T) = T$,

2. $elim(C(T_1, \ldots, T_n)) = C(elim(T_1), \ldots, elim(T_n))$, if $C \neq \#$ and for every $1 \leq i \leq n$, $elim(T_i)$ is defined, and it is not defined, otherwise,

3. $elim(\#(T_1, T_2)) = \#(elim(T_1), elim(T_2)) = elim(T_1)\, mgu(elim(T_1), elim(T_2))$, if $elim(T_i)$, $i = 1, 2$ are defined and $\#(T_1, T_2)$ is well-defined, and it is not defined, otherwise. □

Next the meaning of a term $T \in \mathcal{T}^{\#}$ in a context $S \in \mathcal{S}^{\#}$ is defined by a semantic function $BEH^{\#}_{\mathcal{T}-\mathcal{S}}$. This function is defined only for well-defined terms and specifications and moreover, only for terms and specifications with identical structures.

Definition: Let $T \in \mathcal{T}^{\#}$ and $S \in \mathcal{S}^{\#}$ be such that both $elim(T)$ and $elim(S)$ are defined and moreover, $elim(T)$ and $elim(S)$ are identically structured. The *meaning* of the term T in the context of S, given by the function $BEH^{\#}_{\mathcal{T}-\mathcal{S}} : \mathcal{T} \times \mathcal{S} \rightarrow \widehat{\wp(\mathcal{B})}$, is defined as follows:

1. if $elim(T) \in \mathcal{T}$ then $BEH^{\#}_{\mathcal{T}-\mathcal{S}}[T \downarrow_S] = BEH_{\mathcal{T}}[elim(T)]$,
2. if $elim(T) \in \mathcal{V}$ then $BEH^{\#}_{\mathcal{T}-\mathcal{S}}[T \downarrow_S] = BEH^{\#}_{\mathcal{S}}[S]$,
3. if $elim(T) = C(T_1, \ldots, T_n)$ and $elim(S) = C(S_1, \ldots, S_n) \triangleright S'$ then $BEH^{\#}_{\mathcal{T}-\mathcal{S}}[T \downarrow_S] = K_C(BEH^{\#}_{\mathcal{T}-\mathcal{S}}[T_1 \downarrow_{S_1}], \ldots, BEH^{\#}_{\mathcal{T}-\mathcal{S}}[T_n \downarrow_{S_n}])$. $\square$

In order to define the set of new correctness formulas we need the following definition.

Definition: Let $T \in \mathcal{T}^{\#}$ and $S \in \mathcal{S}^{\#}$. A subterm $T' \in T$ and a subterm $S' \in S$ *correspond* w.r.t. T and S if one of the following conditions holds:

1. $T' = T$ and $S' = S$,
2. There exist subterms T'' and S'' of T and S which correspond w.r.t. T and S and one of the following conditions holds:
   (a) $T'' = C(T_1, \ldots, T_n)$, $S'' = C(S_1, \ldots S_n) \triangleright S^*$, $T' = T_i$ and $S' = S_i$, for some i, $1 \leq i \leq n$,
   (b) $T'' = \#(T_1, T_2)$, $S'' = \#(S_1, S_2)$, $T' = T_i$ and $S' = S_i$, for some i, $1 \leq i \leq 2$.

Definition: Let $T \in \mathcal{T}^{\#}$ and $S \in \mathcal{S}^{\#}$. The formula "$T\ Sat^{\#}\ S$" *holds* in the structure $< \mathcal{B}, BEH^{\#}_{\mathcal{T}-\mathcal{S}}, BEH^{\#}_{\mathcal{S}} >$ iff $BEH^{\#}_{\mathcal{T}-\mathcal{S}}[T \downarrow_S]$ is defined and for each pair of subterms $T' \in T$ and $S' \in S$, if T' and S' correspond w.r.t. T and S then $BEH^{\#}_{\mathcal{T}-\mathcal{S}}[T' \downarrow_{S'}] \subseteq BEH^{\#}_{\mathcal{S}}[S']$.

3.3 The proof system

Next, a proof system for the new correctness formulas is presented.

Definition: Let $\mathcal{AR} =< \mathcal{A}, \mathcal{R}, \mathcal{G} >$ be a basic compositional proof system. The compositional proof system $\mathcal{AR}^{\#} =< \mathcal{A}^{\#}, \mathcal{R}^{\#}, \mathcal{G} >$ for formulas of the form "$T\ Sat^{\#}\ S$" is defined by:

1. The set $\mathcal{G}$ is the basic set of predicates,
2. The set $\mathcal{A}^{\#}$ is an extension of $\mathcal{A}$ with the following formulas:

   The variable-axioms

   $$X\ \ Sat^{\#}\ \overline{S}, \quad where\ X \in \mathcal{V}\ and\ \overline{S} \in \overline{\mathcal{S}}.$$

3. The set $\mathcal{R}^{\#}$ is defined by:
   (a) **The composition rules:** for every rule R_C in $\mathcal{R}$ of the form

   $$\frac{T_i\ Sat\ S_i,\ i = 1, n,\ \ G_C(S_1, \ldots, S_n, S)}{C(T_1, \ldots, T_n)\ Sat\ S}$$

   the following rules are in $\mathcal{R}^{\#}$:

   $$\frac{T_i\ Sat^{\#}\ S_i,\ i = 1, n,\ \ G_C(S_1, \ldots, S_n, S)}{C(T_1, \ldots, T_n)\ Sat^{\#}\ S}$$

   provided that $T_i \in \mathcal{T}$, $S_i \in \mathcal{S}$, for $i = 1, n$, and

   $$\frac{T_i\ Sat^{\#}\ S_i,\ i = 1, n,\ \ G_C(conc(S_1), \ldots, conc(S_n), S)}{C(T_1, \ldots, T_n)\ Sat^{\#}\ C(S_1, \ldots, S_n) \triangleright S}$$

(b) Three additional rules:

i. **The jigsaw rule**

$$\frac{T_i \; Sat^{\#} \; S_i, \; i = 1, 2}{\#(T_1, T_2) \; Sat^{\#} \; \#(S_1, S_2)}$$

Provided that $\#(S_1, S_2)$ *is well − defined*

ii. **The substitution rule**

$$\frac{T \; Sat^{\#} \; S}{T \; Sat^{\#} \; S[skel_1(S_1)\theta \; / \; \#(S_1, S_2)]}$$

For every subterm $\#(S_1, S_2)$ in S, where
$\theta = \; (mgu^*(skel_1(S_1), skel_1(S_2)) \; o \; skel_2(S_1) \; o \; skel_2(S_2))$

iii. **The assumption-freeness rule**

$$\frac{T \; Sat^{\#} \; S}{T \; Sat^{\#} \; S[S_1/(M_1 \rhd S_1)]}$$

For every subterm $M_1 \rhd S_1$ in S, where M_1 *is assumption − free*.

Comments:

1. The variable axioms allow to assume any basic specification for a variable. If the set $\mathcal{S}$ is not recursive neither is the set of additional axioms presented here.

2. A predicate $G \in \mathcal{G}$ is redefined over $(\mathcal{S} \cup \overline{\mathcal{S}})^n$ in the following way: $G(S_1, \ldots, \overline{S_i}, \ldots S_n)$ holds iff $G(S_1, \ldots, S_i, \ldots S_n)$ holds.

3. The composition rules include two types of rules. The simpler rules enable to employ part of the proof in the basic system, provided that the programs and the specifications are of basic types. This postpones the introduction of the more complex rules, required for the jigsaw-extended system, to the point where they are necessary. The composition rules of the second type allow to preserve the component specifications which the conclusion S is based upon.

4. The jigsaw rule allows to derive a mixed specification for a jigsaw-composed program based on the specifications of its constituents. Thus, the compositionality of the basic proof system is preserved. Since only well-defined specifications are allowed, this rule is applicable only when assumptions made on gaps in T_1 are implied (confirmed) by the specifications proved for the corresponding sub-programs in T_2. Similarly, assumptions made on gaps in T_2 are confirmed by the specifications proved for the corresponding sub-programs in T_1.

5. The substitution rule allows to eliminate a jigsaw operator and to reduce the number of assumptions in a specification.

6. The assumption-freeness rule allows to replace a mixed-specification which is free of assumptions by a semantically equivalent basic specification. □

Next the main theorem of the paper is presented. The theorem claims the soundness and completeness of the extended system with respect to the basic one, i.e., given two programs, T in the basic system and T' in the extended one, such that T' is semantically equal to T we prove that any basic specification that can be verified for T' is semantically true for T and every basic specification, semantically true of T, can be verified for T'.

Definition: A predicate $G_C \in \mathcal{G}$ is *monotone* iff

1. If $S_i' \Rightarrow S_i$ and $G_C(S_1, \ldots, S_i, \ldots, S_k, S_{k+1})$ holds then $G_C(S_1, \ldots, S_i', \ldots, S_k, S_{k+1})$ holds, for every $1 \leq i \leq k$.
2. If $S_{k+1} \Rightarrow S_{k+1}'$ and $G_C(S_1, \ldots, S_i, \ldots, S_k, S_{k+1})$ holds then $G_C(S_1, \ldots, S_i, \ldots, S_k, S_{k+1}')$ holds.

Definition: A predicate $G_C \in \mathcal{G}$ is *compositional* iff, if $G_C(S_1, \ldots, S_i, \ldots, S_k, S')$ holds and $G_C(S_1, \ldots, S_i, \ldots, S_k, S'')$ holds then $G_C(S_1, \ldots, S_i, \ldots, S_k, S' \wedge S'')$ holds.

Theorem 3.1: (Soundness and Completeness) Assume all predicates in $\mathcal{G}$ are monotone and compositional. For every $T' \in \mathcal{T}^\#$ and $T \in \mathcal{T}$ such that $T = elim(T')$ and for every basic specification S: $\models T\ Sat\ S$ *iff* $\vdash_{\mathcal{AR}^\#} T'\ Sat^\#\ S$. $\Box$

In Appendix B a proof of Theorem 3.1 is presented.

4 Example

In this section we present an example of a concrete proof system and show that it fits the abstract framework presented in Section 2. Consequently, an extended proof system which allows for jigsaw compositions can be syntactically derived from the basic proof system. The soundness and the completeness of the concrete jigsaw-extended system with respect to the concrete basic system is derived from the soundness and completeness of the abstract framework (Theorem 3.1). The concrete system is a simple version of the system presented in [ZRB85].

4.1 The programming language

Following [ZRB85], a compositional syntax-directed proof system for proving partial correctness for a CSP-like language is presented. The set $\mathcal{T}$ of programs is defined over the following sets of primitive objects, $\mathcal{O}$, and constructors, $\mathcal{C}$:

$$\mathcal{O} = \{skip\} \cup \{\overline{x} := \overline{e}\} \cup \{b\} \cup \{D!e\} \cup \{D?x\}$$
$$\mathcal{C} = \{\|,\ ;,\ \Box,\ While\}$$

The *skip* statement and the multiple assignment statement $\overline{x} := \overline{e}$ have their usual meanings. The statement b (a boolean condition over $\overline{x}$) functions as a guard. Whenever b is evaluated to *true* it can be passed. When b evaluates to *false* the guard cannot be passed and the computation either terminates, if b guards a '*While*' statement or fails, if b does not guard a '*While*' statement. Processes in the network are not allowed to have shared program variables. Processes can communicate with each other only along named, directed channels. Communication along a channel, say D, occurs when an output command $D!e$ and an input command $D?x$ are executed simultaneously by the processes attached to D. The value e is then assigned to the program variable x and both processes continue their execution. The statements $(P_1 \parallel P_2)$[6] and $(P_1; P_2)$ stand for parallel and sequential compositions of P_1 and P_2, respectively. The statement $(P_1 \Box P_2)$ stands for nondeterministic choice between P_1 and P_2. Finally, the statement $(While\ b, P)$ stands for guarded iteration, where the guard is b and the body is P.

4.2 The semantics of the programming language

The set of behaviors $\mathcal{B}$ is next defined.

Definition: A *trace* τ is a finite sequence of pairs consisting of a channel-name and a communication value. A special case is the empty trace $<>$. A trace τ_1 is an initial prefix of τ_2, written as $\tau_1 \leq \tau_2$, iff there exists a trace τ such that the concatenation $\tau_1\tau$ is equal to τ_2. $\Box$

Definition: A state s is a function assigning values to program variables. The set of all states is denoted by $State$. $\Box$

Definition: A state-trace-state triple is a triple (s_1, τ, s_2), where τ is a trace, s_1 is a state and s_2 is a state or a special symbol $\perp$. $\Box$

A triple $(s_1, \tau, \perp)$ indicates an unfinished computation, that is, a computation which started in state s_1 and has already performed the communications in τ, but has not yet terminated.

The set $\mathcal{B}$ is defined as the set of all state-trace-state triples.

Definition: A set of state-trace-state $\mathcal{U}$ is *prefix closed* iff for all triples (s_1, τ_1, s_2), (s_3, τ_2, s_4): if $(s_3, \tau_2, s_4) \in \mathcal{U}$ and $(s_1, \tau_1, s_2) \leq (s_3, \tau_2, s_4)$ then $(s_1, \tau_1, s_2) \in \mathcal{U}$, where $(s_1, \tau_1, s_2) \leq (s_3, \tau_2, s_4)$ iff $(s_1 = s_3 \wedge \tau_1 = \tau_2 \wedge s_2 = s_4)$ or $(s_1 = s_3 \wedge \tau_1 \leq \tau_2 \wedge s_2 = \perp)$. $\Box$

We denote by Θ the set $\{(s, <>, \perp) | s \in States\}$.

Definition: The set $\widehat{\wp(\mathcal{B})}$ is the set of all prefix closed sets of state-trace-state triples which contain the set Θ. $\Box$

Lemma 4.1: The set $\widehat{\wp(\mathcal{B})}$ with the inclusion order $(\subseteq)$ is a cpo (complete partial order) with a minimal element Θ. $\Box$

Proof:

1. The inclusion order on sets is a partial order (reflexive, anti-symmetric, transitive).
2. By definition the minimal element Θ is included in every element of $\widehat{\wp(\mathcal{B})}$.
3. Every chain $X = B_1 \subseteq B_2 \subseteq \ldots$ has a lub (least upper bound) in $\widehat{\wp(\mathcal{B})}$ which is equal to the union of all elements in the chain X. $\Box$

Definition: For a set $\mathcal{U}$ of state-trace-state triples, $close(\mathcal{U})$ is defined as the least nonempty prefix closed set containing all elements of $\mathcal{U}$. $\Box$

The semantics of the programs in $\mathcal{T}$ is next given by defining a semantic function $BEH_{\mathcal{T}} : \mathcal{T} \to \widehat{\wp(\mathcal{B})}$. Let the variables of a program range over the set Val of values. First, $BEH_{\mathcal{T}}$ is

[6] For clarity we use infix notation.

defined for all elements in $\mathcal{O}$:

$$\begin{array}{ll}
BEH_T(skip) & = \{close(\{(s,<>,s)\}) \mid s \in State \wedge s \neq \perp\} \\
BEH_T(x := e) & = \{close(\{(s,<>,s[[e]s/x])\}) \mid s \in State \wedge s \neq \perp\} \\
BEH_T(b) & = \{close(\{if\ [b]s\ then\ (s,<>,s)\ else\ (s,<>,\perp)\}) \mid s \in State \wedge s \neq \perp\} \\
BEH_T(D!e) & = \{close(\{(s,(D,[e]s),s)\}) \mid s \in State \wedge s \neq \perp\} \\
BEH_T(D?x) & = \{close(\{(s,(D,v),s[v/x])\}) \mid v \in Val, s \in State \wedge s \neq \perp\}
\end{array}$$

In order to define the set of functions $\{K_C \mid C \in \mathcal{C}\}$ we adopt the following notations:

1. Let $B_1, B_2 \in \widehat{\wp(\mathcal{B})}$.
2. Let $chan(B_i)$ stands for all channels which appear in the traces of B_i.
3. Let $chan(B_1, B_2) = chan(B_1) \cup chan(B_2)$.
4. Let $\tau|_{chan(B_i)}$ stands for the projection of the trace τ on the set of channels $chan(B_i)$.
5. Let $var(B_i)$ be the set of variables for which the states in B_i assign values.
6. Let $s|_{var(B_i)}$ stands for the projection of the state s on the variables of $var(B_i)$.

The set $\{K_C \mid C \in \mathcal{C}\}$ is defined as follows:

$$\begin{array}{ll}
K_{\Box}(B_1, B_2) = & B_1 \cup B_2 \\
K_{;}(B_1, B_2) = & \{(s,\tau,\perp) \mid (s,\tau,\perp) \in B_1\} \cup \\
& \{(s_1,\tau_1\tau_2,s_2) \mid \exists s^{'} : (s_1,\tau_1,s^{'}) \in B_1 \ \wedge \ (s^{'},\tau_2,s_2) \in B_2\} \\
K_{\|}(B_1, B_2) = & \{(s_1,\tau,s_2) \mid (s_1,\tau|chan(B_1),s^{'}) \in B_1 \ \wedge \ (s_1,\tau|chan(B_2),s^{''}) \in B_2 \wedge \\
& (\tau|_{chan(B_1,B_2)}) = \tau \wedge (if\ s^{'} \neq \perp\ and\ s^{''} \neq \perp\ \ then \\
& s_2|_{var(B_1)} = s^{'} \wedge s_2|_{var(B_2)} = s^{''}\ \ else\ \ s_2 = \perp)\} \\
K_{While}(B_1, B_2) = & \mu \mathcal{J}_{B_1,B_2}(\Theta)
\end{array}$$

Where,

$$\begin{array}{rl}
\mathcal{J}_{B_1,B_2} \stackrel{df.}{=} & \lambda F_{B_1,B_2} \lambda B. \cup_{(s_1,\tau,s_2)\in B} (if\ (s_2,<>,s_2) \notin B_1 \\
& then\ \{(s_1,\tau,s_2)\}\ else\ F_{B_1,B_2}(K_{;}((s_1,\tau,s_2),B_2)))
\end{array}$$

□

Lemma 4.2: For every $T \in \mathcal{T}$, $BEH_{\mathcal{T}}[T] \in \widehat{\wp(\mathcal{B})}$. □
Proof Sketch:

1. By definition for every $O \in \mathcal{O}$, $BEH_{\mathcal{T}}[O] \in \widehat{\wp(\mathcal{B})}$.
2. It is possible to show that for every $C \in \mathcal{C}$, if $B_1, \ldots, B_n \in \widehat{\wp(\mathcal{B})}$ then $K_C(B_1, \ldots, B_n) \in \widehat{\wp(\mathcal{B})}$. □

Conclusion: The least fixed point of $\mathcal{J}_{B_1,B_2}$ is defined since $\widehat{\wp(\mathcal{B})}$ is a cpo and the K_C functions are monotone. Therefore, for every $T \in \mathcal{T}$ the function $BEH_{\mathcal{T}}$ defines a meaning.

4.3 The specification

The set $\mathcal{S}$ of specifications is a set of formula-triples with a typical element $[I,p,q]$, where I,p and q are formulas in a first order predicate language with two sorts of *auxiliary* variables, *integer* variables v and *trace* variables t. A trace is a sequence of communications each of which contains a channel name and a value. Intuitively, a correctness formula "$T\ Sat\,[I,p,q]$" holds if whenever T is started in an initial state and with an initial communication history for which p holds, then the invariant I holds for the communication history of T at any moment during the execution of T. Also, whenever T terminates, q holds for the final state and the final communication history of T. The invariant I does not contain program variables.

The syntax of the formulas I,p,q is defined as:

expression	$e :: 0 \mid 1 \mid \omega \mid v \mid x \mid val(texp[e]) \mid \lvert texp \rvert \mid e_1 + e_2$
channel – name	$c :: D \mid chan(texp[e])$
trace	$texp :: t \mid \pi_{cset} \mid (texp \lvert cset)$
formula	$p :: e_1 = e_2 \mid c_1 = c_2 \mid p_1 \wedge p_2 \mid \neg p \mid \exists v.[p] \mid \exists t.[p]$

Comments: ω is a special value of program variables, standing for "undefined". $val(texp[e])$ is the value of the e-th communication in trace $texp$. $|texp|$ is the length of the trace $texp$. $chan(texp[e])$ is the channel-name of the e-th communication in the trace $texp$. π is a special auxiliary trace variable which must occur free in a formula. π denotes the communication history of the execution up to the current state. Moreover, only a projection of π on a set of channels $cset$, to be denoted π_{cset}, can occur in a formula. $(texp|cset)$ is the projection of trace $texp$ on a set of channels, $cset$. Auxiliary variables, except for π, that occur free in formulas are also called *logical* variables.

We will freely use abbreviations such as $texp_1 = texp_2$ or $p \to q$, defined as usual.

4.4 The semantics of the specifications

An assertion p is interpreted in an auxiliary variables environment γ and a state-trace-state triple (s_1, τ, s_2), where $s_2 \neq \perp$. We write $[p]\gamma(s_1,\tau,s_2)$ for the following interpretation:

1. free occurrences of auxiliary integer variable v or trace variable t, are interpreted as values $\gamma(v)$ and $\gamma(t)$ resp.,
2. program variable x is interpreted as $s_2(x)$,
3. a trace projection π_{cset} is taken to be $(\tau|cset)$.

An assertion p is called *valid* denoted by $\models p$ iff $\forall\gamma\ \forall(s_1,\tau_1,s_2).[p]\gamma(s_1,\tau_1,s_2)$ holds. The semantics of the specifications is defined by the semantic function $BEH_S : \mathcal{S} \to \wp(\mathcal{B})$:

$$BEH_S[[I,p,q]] = \{(s_1,\tau_1,s_2) \mid \forall\gamma\forall(s_0,\tau_0,s_1)[[p]\gamma(s_0,\tau_0,s_1)\ implies$$

$$\forall(s_1,\tau_1^{'},s_2^{'}) \leq (s_1,\tau_1,s_2)\ [[I]\gamma(s_0,\tau_0\tau_1^{'},s_2^{'}) \wedge (s_2^{'} \neq \perp\ implies\ [q]\gamma(s_0,\tau_0\tau_1^{'},s_2^{'}))]\}$$

A formula of the form $T\ \ Sat\ \ S$ is defined to hold iff $BEH_T[T] \subseteq BEH_S[S]$. Intuitively, a program T satisfies $\{I,p,q\}$ if given an initial state-trace-state, (s_0,τ_0,s_1), for which p holds, each execution of T which starts in s_1 either diverges or terminates in a state, s_2, such that

q holds in $(s_0, \tau_0\tau_1, s_2)$, where τ_1 is the trace associated with that execution. Moreover, the assertion I holds for all prefixes of $(s_0, \tau_0\tau_1, s_2)$.

Lemma 4.3: For every specification $[I, p, q]$ if $p \rightarrow I$ is not valid then there does not exist a program $T \in \mathcal{T}$ which satisfies this specification. $\Box$

Proof: First we prove the following Claim.
Claim: Let (s_0, τ_0, s_1) and γ be such that $[p \wedge \neg I]\gamma(s_0, \tau_0, s_1)$ then $(s_1, <>, \perp) \notin BEH_{\mathcal{S}}[[I, p, q]]$.
Proof of claim: Assume $(s_1, <>, \perp) \in BEH_{\mathcal{S}}[[I, p, q]]$. According to the definition of $BEH_{\mathcal{S}}$ if $[p]\gamma(s_0, \tau_0, s_1)$ holds and $(s_1, <>, \perp) \in BEH_{\mathcal{S}}[[I, p, q]]$ then $[I]\gamma(s_0, \tau_0 <>, \perp)$ holds. Therefore, $[I]\gamma(s_0, \tau_0 <>, s_1)$ holds (since I is interpreted only in τ_0), a contradiction. $\Box$

Based on the above Claim if $p \rightarrow I$ is not valid then $BEH_{\mathcal{S}}[[I, p, q]]$ does not contain the set Θ and therefore there does not exist a program which satisfies this specification (since the meaning of every program contains Θ). $\Box$

Conclusion: We can redefine the set of specifications $\mathcal{S}$ to be the set of all specifications for which $p \rightarrow I$ holds. This restriction does not exclude any interesting specification.

Lemma 4.4: For every $[I, p, q]$ if $p \rightarrow I$ is valid then $BEH_{\mathcal{S}}[[I, p, q]] \in \widehat{\wp(\mathcal{B})}$. $\Box$
Proof:

1. Prefix-closed: Let $(s_1, \tau_1, s_2) \in BEH_{\mathcal{S}}[[I, p, q]]$ and let $(s_1, \tau_1^{'}, s_2^{'}) \leq (s_1, \tau_1, s_2)$ then for every γ and for every (s_0, τ_0, s_1) : if $[p]\gamma(s_0, \tau_0, s_1)$ then for every $(s_1, \tau_1^{''}, s_2^{''}) \leq (s_1, \tau_1^{'}, s_2^{'})$ the following holds $(s_1, \tau_1^{''}, s_2^{''}) \leq (s_1, \tau_1, s_2)$ and therefore
$$[I]\gamma(s_0, \tau_0\tau_1^{''}, s_2^{''}) \wedge (s_2^{''} \neq \perp \ \ implies \ \ [q]\gamma(s_0, \tau_0\tau_1^{''}, s_2^{''})).$$
2. Includes Θ: Let $s_1 \in State$. Next we show that $(s_1, <>, \perp) \in BEH_{\mathcal{S}}[[I, p, q]]$.

   For every γ and for every (s_0, τ_1, s_1) it has to be shown that if $[p]\gamma(s_0, \tau_0, s_1)$ holds (we therefore know that $[I]\gamma(s_0, \tau_0, s_1)$ holds) then $[I]\gamma(s_0, \tau_0, \perp)$ holds. We can conclude the above since I is interpreted only in τ_0. $\Box$

Conclusion: For every mixed-specification $S \in \mathcal{S}^{\#}$ the function $BEH_{\mathcal{S}}^{\#}$ (defined as presented in the abstract framework) defines a meaning.

4.5 The basic proof-system

Only some of the axioms and the rule of the system are brought here, the rest can be found in [ZRB85].

Skip	$skip$	Sat	$[I, p \wedge I, p \wedge I]$
Assignment	$\overline{x} := \overline{e}$	Sat	$[I, p[\overline{e}/\overline{x}] \wedge I, p \wedge I]$
Guard	b	Sat	$[I, p \wedge I, p \wedge I \wedge b]$
Output	$D!e$	Sat	$[I, I \wedge I[\pi_D < D, e > /\pi_D] \wedge q[\pi_D < D, e > /\pi_D], I \wedge q]$
Input	$D?x$	Sat	$[I, I \wedge \forall v. (I[\pi_D < D, v > /\pi_D] \wedge q[\pi_D < D, v > /\pi_D, v/x]), I \wedge q]$

Sequential-composition

$$\frac{T_1\ Sat\ S_1,\ \ T_2\ Sat\ S_2,\ \ G_;(S_1,S_2,S)}{(T_1;T_2)\ Sat\ S}$$

Where, $S_i = [I_i, p_i, q_i]$, $i = 1, 2$, $S = [I, p, q]$ and $G_;(S_1, S_2, S)$ holds iff $(I_1 \rightarrow I) \wedge (I_2 \rightarrow I) \wedge (p \rightarrow p_1) \wedge (q_1 \rightarrow p_2) \wedge (q_2 \rightarrow q)$.

Choice-composition

$$\frac{T_1\ Sat\ S_1,\ \ T_2\ Sat\ S_2,\ \ G_\Box(S_1,S_2,S)}{(T_1 \Box T_2)\ Sat\ S}$$

Where, $S_i = [I_i, p_i, q_i]$, $i = 1, 2$, $S = [I, p, q]$ and $G_\Box(S_1, S_2, S)$ holds iff $(I_1 \rightarrow I) \wedge (I_2 \rightarrow I) \wedge (p \rightarrow (p_1 \wedge p_2)) \wedge ((q_1 \vee q_2) \rightarrow q)$.

Parallel-composition

$$\frac{T_1\ Sat\ S_1,\ \ T_2\ Sat\ S_2,\ \ G_\|(S_1,S_2,S)}{(T_1 \| T_2)\ Sat\ S}$$

Where, $S_i = [I_i, p_i, q_i]$, $i = 1, 2$, $S = [I, p, q]$ and $G_\|(S_1, S_2, S)$ holds iff $((I_1 \wedge I_2) \rightarrow I) \wedge (p \rightarrow (p_1 \wedge p_2)) \wedge ((q_1 \wedge q_2) \rightarrow q)$. With the restrictions: $var(p_i, q_i) \cap var(s_j) = \emptyset$ for $(i, j) = (1, 2), (2, 1)$, $chan(I_i, p_i, q_i) \cap chan(S_j) \subset chan(S_i)$ for $(i, j) = (1, 2), (2, 1)$.

$While$ **-composition**

$$\frac{T_1\ Sat\ S_1,\ \ b\ Sat\ S_2,\ \ G_{While}(S_1,S_2,S)}{(While\ b, T_1)\ Sat\ S}$$

Where $S_1 = [I_1, p_1, q_1]$, $S = [I, p, q]$, $S_2 = [true, b, b]$ and $G_{While}(S_1, S_2, S)$ holds iff $(I_1 \rightarrow I) \wedge ((b \wedge p) \rightarrow p_1) \wedge ((q_1 \wedge b) \rightarrow p_1) \wedge ((q_1 \wedge \neg b) \rightarrow q)$. $\Box$

In order to prove that the above system fits the general abstract framework presented in section 2 we have to show that the set of specifications $\mathcal{S}$ is closed under conjunction and that there exists a way to determine if a specification S_1 confirms a specification S_2, $(BEH_{\mathcal{S}}[S_1] \Rightarrow BEH_{\mathcal{S}}[S_2])$.

Lemma 4.5: $BEH_{\mathcal{S}}[[I_1, p_1, q_1]] \Rightarrow BEH_{\mathcal{S}}[[I_2, p_2, q_2]]$ holds iff $(I_1 \rightarrow I_2) \wedge (p_2 \rightarrow p_1) \wedge (q_1 \rightarrow q_2)$ holds.

Proof: Immediate.

To prove closeness under conjunction of $\mathcal{S}$ we have to show that $[I_1, p_1, q_1] \wedge [I_2, p_2, q_2]$ is a specification in $\mathcal{S}$ and that the meaning of this specification is the intersection of the meaning of $[I_1, p_1, q_1]$ with the meaning of $[I_2, p_2, q_2]$. Unfortunately, in [ZRB85] the conjunction operator on specifications of the form $[I, p, q]$ is not defined, however, we show that it is possible to define a specification in $\mathcal{S}$ such that its meaning is the intersection of the meaning of $[I_1, p_1, q_1]$ with the meaning of $[I_2, p_2, q_2]$. Thus, semantically, $\mathcal{S}$ is closed under conjunction even though the conjunction is not represented in the syntax of $\mathcal{S}$ by the $\wedge$ syntactic operator.

Let I, p, q be defined as follows:

$$\begin{aligned}
I = &\quad (L = 1 \rightarrow (I_1 \wedge I_2)) \wedge (L = 2 \rightarrow I_1) \wedge (L = 3 \rightarrow I_2) \\
p = &\quad (p_1 \vee p_2) \wedge ((p_1 \wedge p_2) \rightarrow L = 1) \wedge ((p_1 \wedge \neg p_2) \rightarrow L = 2) \wedge ((\neg p_1 \wedge p_2) \rightarrow L = 3) \\
q = &\quad (L = 1 \rightarrow (q_1 \wedge q_2)) \wedge (L = 2 \rightarrow q_1) \wedge (L = 3 \rightarrow q_2)
\end{aligned}$$

where L is a logical variable which does not appear in the assertions I_i, p_i, q_i, for $i \in \{1, 2\}$.

Let $S_i = [I[i/L], p[i/L], q[i/L]]$. As usual for logical variables (see also [ZRB85]), we interpret $T\ Sat\ [I, p, q]$ as $\forall i:\ \ T\ Sat\ S_i$. By definition this set of correctness formulas hold iff

$$BEH_{\mathcal{T}}[T] \subseteq (\cap_i BEH_{\mathcal{S}}[S_i]).$$

Therefore, the meaning of $[I, p, q]$ is $(\cap_i BEH_{\mathcal{S}}[S_i])$. In our case, for every $i \notin \{1, 2, 3\}$, $S_i = [true, false, true]$, that is, $BEH_{\mathcal{S}}[S_i] = \mathcal{B}$. Thus,

$$(\cap_i BEH_{\mathcal{S}}[S_i]) = (BEH_{\mathcal{S}}[S_1] \cap BEH_{\mathcal{S}}[S_2] \cap BEH_{\mathcal{S}}[S_3]).$$

It is easy to see that

$$\begin{aligned}
S_1 &= [I_1 \wedge I_2, p_1 \wedge p_2, q_1 \wedge q_2] \\
S_2 &= [I_1, p_1 \wedge \neg p_2, q_1] \\
S_3 &= [I_2, \neg p_1 \wedge p_2, q_2]
\end{aligned}$$

To show that the meaning of $[I, p, q]$ is equal to the intersection of the meaning of $[I_1, p_1, q_1]$ with meaning of $[I_2, p_2, q_2]$ we prove the following lemma:

Lemma 4.6:

$$BEH_{\mathcal{S}}[[I_1, p_1, q_1]] \cap BEH_{\mathcal{S}}[[I_2, p_2, q_2]] = BEH_{\mathcal{S}}[S_1] \cap BEH_{\mathcal{S}}[S_2] \cap BEH_{\mathcal{S}}[S_3].$$

Proof:

First direction: We show that $BEH_{\mathcal{S}}[[I_1, p_1, q_1]] \cap BEH_{\mathcal{S}}[[I_2, p_2, q_2]] \subseteq BEH_{\mathcal{S}}[S_1] \cap BEH_{\mathcal{S}}[S_2] \cap BEH_{\mathcal{S}}[S_3]$. Assume, $(s_1, \tau_1, s_2) \in BEH_{\mathcal{S}}[[I_1, p_1, q_1]] \cap BEH_{\mathcal{S}}[[I_2, p_2, q_2]]$.

Four cases:

1. Let γ and (s_0, τ_0, s_1) be such that $[p_1 \wedge p_2]\gamma(s_0, \tau_0, s_1)$ holds. Then for every $(s_1, \tau_1^{'}, s_2^{'}) \leq (s_1, \tau_1, s_2)$, $[I_1 \wedge I_2]\gamma(s_0, \tau_0\tau_1^{'}, s_2^{'})$ holds and if $s_2^{'} \neq \perp$ then $[q_1 \wedge q_2]\gamma(s_0, \tau_0\tau_1^{'}, s_2^{'})$ holds. Therefore, checking every γ and (s_0, τ_0, s_1) as above we can conclude that $(s_1, \tau_1, s_2) \in BEH_{\mathcal{S}}[S_i]$, $i = 1, 3$.

2. Let γ and (s_0, τ_0, s_1) be such that $[\neg p_1 \wedge p_2]\gamma(s_0, \tau_0, s_1)$ holds. Then for every $(s_1, \tau_1^{'}, s_2^{'}) \leq (s_1, \tau_1, s_2)$, $[I_2]\gamma(s_0, \tau_0\tau_1^{'}, s_2^{'})$ holds and if $s_2^{'} \neq \perp$ then $[q_2]\gamma(s_0, \tau_0\tau_1^{'}, s_2^{'})$ holds. Therefore, checking every γ and every (s_0, τ_0, s_1) we can conclude that $(s_1, \tau_1, s_2) \in BEH_{\mathcal{S}}[S_i]$, $i = 1, 3$.

3. The case where $[p_1 \wedge \neg p_2]\gamma(s_0, \tau_0, s_1)$ holds is similar.

4. Let γ and (s_0, τ_0, s_1) be such that $[\neg p_1 \wedge \neg p_2]\gamma(s_0, \tau_0, s_1)$ holds. Then $[p[i/L]]\gamma(s_0, \tau_0, s_1)$ is false for every $i = 1, 3$. As a result, (s_1, τ_1, s_2) is trivially contained in $BEH_{\mathcal{S}}[S_i]$, $i = 1, 3$.

Second direction: To show that $BEH_{\mathcal{S}}[S_1] \cap BEH_{\mathcal{S}}[S_2] \cap BEH_{\mathcal{S}}[S_3] \subseteq BEH_{\mathcal{S}}[[I_1, p_1, q_1]] \cap BEH_{\mathcal{S}}[[I_2, p_2, q_2]]$, the same four cases are checked. The details are omitted. □

Next we show that the predicates $G_{While}, G_{\Box}, G_{\|}, G_{;}$, are monotone and compositional and therefore the syntactic transformation defined in Section 3 on the basic proof system produces a sound and relative complete proof system w.r.t. the basic system.

Lemma 4.7: The predicates $G_{While}, G_{\|}, G_{\Box}, G_{;}$ are monotone and compositional. □
Proof: Only the proof for $G_{;}$ is brought here.

1. Monotone:

   Immediate from the definitions.

2. Compositional:

   Assume $G_{;}([I_1, p_1, q_1], [I_2, p_2, q_2], [I^{'}, p^{'}, q^{'}])$ and $G_{;}([I_1, p_1, q_1], [I_2, p_2, q_2], [I^{''}, p^{''}, q^{''}])$ hold. We know that $(I_1 \rightarrow (I^{'} \wedge I^{''})) \wedge (I_2 \rightarrow (I^{'} \wedge I^{''})) \wedge (p^{'} \rightarrow p_1) \wedge (p^{''} \rightarrow p_1) \wedge (q_1 \rightarrow p_2) \wedge (q_2 \rightarrow (q^{'} \wedge q^{''}))$. The following assertions should be justify:

   (a) $I_1 \rightarrow ((L = 1 \rightarrow (I^{'} \wedge I^{''})) \wedge (L = 2 \rightarrow I^{'}) \wedge (L = 3 \rightarrow I^{''}))$
   (b) $I_2 \rightarrow ((L = 1 \rightarrow (I^{'} \wedge I^{''})) \wedge (L = 2 \rightarrow I^{'}) \wedge (L = 3 \rightarrow I^{''}))$
   (c) $((p^{'} \vee p^{''}) \wedge ((p^{'} \wedge p^{''}) \rightarrow L = 1) \wedge ((p^{'} \wedge \neg p^{''}) \rightarrow L = 2) \wedge ((\neg p^{'} \wedge p^{''}) \rightarrow L = 3)) \rightarrow p_1$
   (d) $(q_1 \rightarrow p_2)$
   (e) $q_2 \rightarrow ((L = 1 \rightarrow (q^{'} \wedge q^{''})) \wedge (L = 2 \rightarrow q^{'}) \wedge (L = 3 \rightarrow q^{''}))$

   All the above can be concluded from the assumption. □

4.6 Application

Showing that the version of the system [ZRB85] presented above fits the basic abstract system, we can develop and verify programs containing jigsaw operators based on the extended system. The jigsaw operator allows the development and verification of a program in a compositional manner in which the units of modularity are not necessarily derived by the top-level constructors of the programming language. To make this point clear we apply the extended concrete system to an example.

Assume we are given the following informal description of a system. The network consists of two nodes. The Producer node is required to produce a sequence of consecutive prime numbers and send them to the other node, called the Consumer. The Consumer node receives the numbers and signals the Producer to halt after receiving a prime number which meets some requirements. In addition, assume that due to the nature of the communication system the values to be communicated should be representable by a binary number of no more than six digits (should not be greater than 63). Thus, some communication management subtask should be added, that handles numbers greater than 63. It is clear that the program should be of the general form $(Producer \,\|\, Consumer)$. Therefore, to develop this program in a compositional syntax-directed manner would traditionally mean to develop and verify the Producer subtask and the Consumer subtask separately and than parallel compose them. However, the program

$Primes :: (Producer^p \,\|\, Consumer^p)$, where

$Producer^p$ ::

```
(While more = 1,
   (While found = 0,
      prod, j, found := prod + 1, 1, 1;
      (While j ≤ max ∧ found ≠ 0,
         found, j := mod(prod, array[j]), j + 1));
   max, array[max + 1], found := max + 1, prod, 0;
   Z)
```

$Consumer^p$ ::

```
(While more = 1,
   V;
   (new ≥ 100; more := 0
   □
    new < 100; skip);
   N)
```

Figure 2: The Primes subtask

$Communication :: (Producer^c \,\|\, Consumer^c)$, where

$Producer^c$ ::

```
(While more = 1,
   Y;
   num := prod;
   (While num ≥ 63,
     D12!63; num := num − 63);
   D12!num; D21?more)
```

$Consumer^c$ ::

```
(While more = 1,
   D12?temp; new := temp;
   (While temp = 63,
    D12?temp; new := new + temp);
   M;
   D21!more)
```

Figure 3: The Communication subtask

consists of two distinct subtasks that one might want to handle separately in a syntax-directed and compositional manner. One subtask includes the production and the acceptance of the prime numbers. The other subtask includes the communication management. The jigsaw operator makes it possible to handle separately those subtasks .

In Figure 2, the Primes subtask is presented. The meta variables Z, V, and N hide the details of the communication management which are not relevant to the development of the Primes subtask. The variable *more* is equal to 1 as long as the Consumer has not yet received the required prime number, that is, a prime number greater or equal to 100. The production of the prime numbers in $Producer^p$ begins in an initial state in which an array variable *array* contains a single value, $array[1] = 2$, and the variable *prod* contains 2. The next prime number is found by repeating the following procedure: The value of *prod* is increased by 1 and then *prod* is divided by all previously found prime numbers (all of them are stored in *array*). If non of them is a divisor of *prod* then *prod* contains a prime number and it is stored in *array*. If the current value of *prod* can be divided by one of the prime numbers stored in *array* then we conclude that *prod* does contain a prime number and repeat the above procedure.

In Figure 3, the Communication subtask is presented. The meta variables Y and M hide the details of the production and the consuming of the prime numbers. In $Producer^c$ a number r greater or equal to 63 is send to $Consumer^c$ by representing r as $m \times 63 + n$, where $0 \leq n < 63$, and sending $m + 1$ numbers to $Consumer^c$. The first m values to be sent are 63 and the last one is n.

The specification that we next verify for the program claims that if the program terminates then the value of *new* at the last state is prime and is greater or equal to 100.

In the following we abbreviate the projection $\pi_{D_{12},D_{21}}$ of π on the channels D_{12} and D_{21} and denote it by π. The length of π is denoted by *last*. We define A_k to be the following set of natural numbers: If the length of π is less than k or the k-th element in π records a communication on channel D_{21} then A_k is the empty set. Else, A_k contains the natural numbers $k, k-1, k-2, \ldots, k-l$ such that the elements $k, k-1, k-2, \ldots k-l$ of π record communications on D_{12} and either the $k-l-1$-th element of π records a communication on D_{21} or $|\pi| = k-l$. Formally,

$$A_k = \begin{cases} \emptyset & \mathit{if}\ \ chan(\pi[k]) = D_{21} \vee |\pi| < k \\ \{k\} \cup A_{k-1} & \mathit{otherwise} \end{cases}$$

Next we present the verification of $Producer^c$:

1. Using the basic guard axiom:

$$more = 1\ \ Sat^{\#}\ \ q_1$$

   where,

$$\begin{array}{lcl} q_1 & \stackrel{df.}{=} & [I, I, more = 1] \\ I & \stackrel{df.}{=} & \forall i : 1 \le i \le last :\ (chan(\pi[i]) = D_{12} \wedge val(\pi[i]) \ne 63) \rightarrow prime(\Sigma_{j \in A_i} val(\pi[j])) \end{array}$$

2. Using a variable axiom:

$$Y\ \ Sat^{\#}\ \ \overline{q_2}$$

   where,

$$\begin{array}{lcl} q_2 & \stackrel{df.}{=} & [I, I \wedge co \wedge pr, I \wedge co \wedge pr \wedge prime(prod)] \\ co & \stackrel{df.}{=} & \pi = <>\ \vee chan(\pi[last]) = D_{21} \\ pr & \stackrel{df.}{=} & \forall k : k \in \mathcal{N} : (prime(k) \wedge k \le prod) \leftrightarrow k \in array[1..max] \end{array}$$

3. Using axioms and proof rules of the basic system:

$$\begin{array}{ll} num := prod; & \\ (While\ num \ge 63, & \\ \ \ D_{12}!63; num := num - 63); & Sat^{\#}\ \ q_3 \\ D_{12}!num; D_{21}?more & \end{array}$$

   where,

$$\begin{array}{lcl} q_3 & \stackrel{df.}{=} & [I, I \wedge co \wedge pr \wedge prime(prod), I \wedge co \wedge pr \wedge prime(prod) \wedge prod = si'] \\ si' & \stackrel{df.}{=} & \Sigma_{j \in A_{last-1}} val(\pi[j]) \end{array}$$

4. Using (2), (3) and the sequential composition rule:

$$\begin{array}{ll} Y; & \\ num := prod; & \\ (While\ num \ge 63, & \\ \ \ D_{12}!63; num := num - 63); & Sat^{\#}\ \ (\overline{q_2}; q_3) \rhd q_4 \\ D_{12}!num; D_{21}?more & \end{array}$$

   where,

$$q_4 \stackrel{df.}{=} [I, I \wedge co \wedge pr, I \wedge co \wedge pr \wedge prime(prod) \wedge prod = si']$$

5. Using (1), (4) and the while composition rule:

$$
\begin{array}{ll}
Producer^c \;\; Sat^{\#} & (While\ q_1, \\
 & \quad (\overline{q_2}; q_3) \rhd q_4 \\
 &) \rhd q_5
\end{array}
$$

where,

$$
\begin{array}{lcl}
q_5 & \stackrel{df.}{=} & [I, I \wedge init, I \wedge prime(si')] \\
init & \stackrel{df.}{=} & found = 0 \wedge max = 1 \wedge array[1] = 2 \wedge prod = 2 \wedge \pi = <> \wedge more = 1
\end{array}
$$

We presented above typical steps of the proof and in Appendix C this example is completed.

5 Conclusions

In [FFG90, FFG91] the jigsaw operator has been introduced into two different frameworks. In both frameworks the jigsaw composition rules are syntax-directed but not compositional. This raises the question whether the non compositionality of those rules is due to the nature of the jigsaw composition or due to the non compositionality of the basic frameworks. Here, we address this question and show that adding the jigsaw to an algebraic and compositional framework results in a compositional rule for jigsaw composition.

The extended proof system is also proved in [Fix92] to be sound and complete, using the common definitions for these notions. In this case, additional properties of the basic framework are assumed.

6 Acknowledgments

We thank the anonymous referees for many useful suggestions to the previous version of this paper. The part of the first and second authors was partially supported by the Israeli National Academy of Sciences (basic research). The second author was also supported by the fund for promotion of research in the Technion. The third author was partially supported by the United States-Israel Bi-national Science Foundation, Jerusalem, Israel.

7 References

[Apt81] K.R. Apt: "Ten years of Hoare's logic: a survey Part I, ACM TOPLAS 3, 1981.

[Bac80] R.J.R. Back: "Correctness preserving program refinements: proof theory and applications," Mathematical Centre Tracts 131, Mathematical Centre, Amsterdam, 1980.

[BF88] L. Bouge, N. Francez: "A Compositional Approach to Superimposition," 15th ACM Symp. on Principles of Programming Languages, San Diego, CA, January 1988.

[CM88] M. Chandy, J. Misra: **"Parallel programs design,"** Addison-Wesly, 1988.

[FF90] N. Francez, I.R. Forman: "Superimposition for interacting processes," CONCUR'90, Amsterdam, August 1990. LNCS 458 J.C.M. Baeten, J.W. Klop (Eds.), Springer-Verlag, 1990.

[FFG90] L. Fix, N. Francez, O. Grumberg: "Semantics-driven decompositions for the verification of distributed programs," Proc. of the IFIP working group 2.2/2.3 working conference on Programming concepts and Methods, Sea of Galilee, Israel, April 1990, North-Holland, pp. 101-123.

[FFG91] L. Fix, N. Francez, O. Grumberg: "Program composition and modular verification," Proc. 18th ICALP, Madrid, July 1991. LNCS 510 J. Leach Albert, B. Monien, M. Rodriguez Artalejo (Eds.), Springer-Verlag, 1991.

[FFG92] L. Fix, N. Francez, O. Grumberg: "Program composition via unification," To appear in the Proc. 19th ICALP, Wien, July 1992.

[Fix92] L. Fix: "Semantics-driven decompositions and compositions for the verification of distributed programs," Ph.D. Thesis, Technion, Haifa, 1992 (written in Hebrew).

[Heh84] E.C.R. Hehner: "Predictive programming," part I and II, Communication ACM 27, 1984.

[Hoa78] C.A.R. Hoare: "Communicating sequential processes," CACM 21, 8, August 1978.

[KFE90] S. Katz, I. Forman, M. Evangelist: "Language constructs for distributed systems," Proc. of the IFIP working group 2.2/2.3 working conference on Programming concepts and Methods, Sea of Galilee, Israel, April 1990, North-Holland, pp. 73-100.

[KKZ90] R. Koymans, R. Kuiper, E. Zijlstra: "Specification specified," in R. Kuiper Ph.D. Thesis University of Eindhoven, 1989.

[Old86] E.R. Olderog: "Process theory: semantics, specification and verification," LNCS 224 J.W. de Bakker, W.P. de Roever, G. Rozenberg (Eds.), Springer-Verlag, 1986.

[Pnu85] A. Pnueli: "In transition from global to modular temporal reasoning about programs," Proc. Logics and Models of Concurrent Systems, Ed. K.R. Apt, Springer-Verlag, 1985.

[Wir71] N. Wirth: "Program development by stepwise refinement," Communication ACM 14, 1971.

[ZRB85] J. Zwiers, W.P. de Roever, P. van Emde Boas: "Compositionality and concurrent networks: soundness and completeness of a proof system," Proc. 12th ICALP, Nafplion, Greece, July 1985, Springer LNCS 194, pp. 509-519.

A Unification

Definition: A *substitution* assigns terms to variables. A substitution is denoted by $\theta = \{X_1 \leftarrow T_1, \ldots, X_n \leftarrow T_n\}$, $n \geq 0$, every T_i is different from X_i and there are no two identical variables. A substitution with no elements is called *empty* substitution and it is denoted by ε. $\Box$

Definition: Let $\theta = \{X_1 \leftarrow T_1, \ldots, X_n \leftarrow T_n\}$ be a substitution and $T \in \mathcal{T}^{\#}$ a term. Then, $T\theta \in \mathcal{T}^{\#}$ is the term obtained from T by simultaneously replacing all occurrences of each X_i in T by T_i. $\Box$

Definition: Let $\theta_1 = \{X_1 \leftarrow T_1, \ldots, X_n \leftarrow T_n\}$, $\theta_2 = \{Y_1 \leftarrow T_1^{'}, \ldots, Y_m \leftarrow T_m^{'}\}$ be two substitutions. The *composition* of θ_1 and θ_2 is the substitution, denoted by $\theta_1 \; o \; \theta_2$, which is obtained from the set $\{X_1 \leftarrow T_1\theta_2, \ldots, X_n \leftarrow T_n\theta_2, Y_1 \leftarrow T_1^{'}, \ldots, Y_m \leftarrow T_m^{'}\}$ by omitting all elements of the form $X_j \leftarrow T_j\theta_2$ for which $X_j = T_j\theta_2$ and all elements of the form $Y_i \leftarrow T_i^{'}$ for which $Y_i \in \{X_1, \ldots, X_n\}$. $\Box$

Definition: A substitution θ is called a *unifier* of two terms T_1, T_2, iff $T_1\theta = T_2\theta$. $\Box$

Definition: Given two terms T_1, T_2 let $\mathcal{U}$ be the set of all unifiers of T_1, T_2. A binary relation

$<_u$ on the elements of $\mathcal{U}$ is defined as follows: $\forall \theta_1, \theta_2 \in \mathcal{U} : \theta_1 <_u \theta_2$ if there exists a substitution δ such that $\theta_1 = \theta_2 \, o \, \delta$. $\Box$

Lemma A.1: The pair $< \mathcal{U}, <_u >$ is a partial pre-order, i.e., reflexive and transitive. $\Box$

Definition: The binary relation R_u, called the renaming relation, on elements of $\mathcal{U}$ is defined as follows: $\theta_1 \ R_u \ \theta_2$ iff $(\theta_1 \ <_u \ \theta_2)$ and $(\theta_2 \ <_u \ \theta_1)$ $\Box$

Lemma A.2: The relation R_u is an equivalence relation, i.e., reflexive, transitive and symmetric. $\Box$

Definition: The partial order $<_{eq}$ on the equivalence classes of R_u is defined as follows: Let E_1, E_2 be two equivalence classes of R_u then $E_1 \ <_{eq} \ E_2$ iff $\exists \gamma \in E_1 \ \exists \theta \in E_2 : \ \gamma \ <_u \ \theta$. $\Box$

Lemma A.3: $E_1 \ <_{eq} \ E_1$ iff $\forall \gamma \in E_1, \ \forall \theta \in E_2 : \ \gamma \ <_u \ \theta$. $\Box$

Lemma A.4: The equivalence classes of the R_u relation are closed under renaming of variables. $\Box$

Lemma A.5: There is only one maximal equivalence class of the R_u relation. $\Box$

Definition: Every element in the maximal equivalence class is called *a most general unifier* of T_1, T_2, denoted by $mgu(T_1, T_2)$. $\Box$

Definition: Let $T_1, T_2 \in \mathcal{T}^{\#}$ be two gapped-programs. Let $T_1^{'} \in T_1$ and $T_2^{'} \in T_2$ be two subprograms of T_1 and T_2. The programs $T_1^{'}$ and $T_2^{'}$ *match with respect to* T_1 and T_2 if one of the following conditions holds:

1. $T_1^{'} = T_1$ and $T_2^{'} = T_2$.
2. There exist subprograms $T_1^{''}$ and $T_2^{''}$ of T_1 and T_2 which match w.r.t. T_1 and T_2 such that $T_1^{''} = C(T_1^*, \ldots, T_n^*)$, $T_2^{''} = C(T_1^{**}, \ldots, T_n^{**})$, $T_1^{'} = T_i^*$ and $T_2^{'} = T_i^{**}$, for some i, $1 \leq i \leq n$. $\Box$

Definition: Programs $T_1, T_2 \in \mathcal{T}^{\#}$ are *unifiable* iff the following two conditions hold:

1. There exists a unifier for T_1 and T_2.
2. Every two subprograms $T_1^{'} \in T_1$ and $T_2^{'} \in T_2$ which match w.r.t. T_1 and T_2 are not syntactically identical. $\Box$

Comment: Condition (2) implies that the jigsaw composition of program with "overlapping" primitive objects is not legal. This restriction is not necessary however it simplifies the technical considerations.

Definition: Let $S_1, S_2 \in \mathcal{SK}$ be two skeleton-terms. Let $S_1^{'} \in S_1$ and $S_2^{'} \in S_2$ be two subterms of S_1 and S_2. The subterms $S_1^{'}$ and $S_2^{'}$ *match with respect to* S_1 and S_2 if one of the following conditions holds:

1. $S_1^{'} = S_1$ and $S_2^{'} = S_2$.
2. There exist subterms $S_1^{''}$ and $S_2^{''}$ of S_1 and S_2 which match w.r.t. S_1 and S_2 such that one of the following holds:
   (a) $S_1^{''} = \#(S_1^*, S_2^*)$, $S_2^{''} = \#(S_1^{**}, S_2^{**})$, $S_1^{'} = S_i^*$ and $S_2^{'} = S_i^{**}$ for some i, $1 \leq i \leq 2$.
   (b) $S_1^{''} = C(S_1^*, \ldots, S_n^*) \triangleright X$, $S_2^{''} = C(S_1^{**}, \ldots, S_n^{**}) \triangleright Y$, $S_1^{'} = S_i^*$ and $S_2^{'} = S_i^{**}$ for some i, $1 \leq i \leq n$. $\Box$

Definition: The skeleton-terms $S_1, S_2 \in \mathcal{SK}$ are *unifiable* iff the following two conditions hold:

1. There exists a unifier for S_1 and S_2.
2. Every two subterms $S_1^{'} \in S_1$ and $S_2^{'} \in S_2$ which match w.r.t. S_1 and S_2 are not both basic specifications. $\Box$

Comment: Condition (2) reflects the corresponding restriction on gapped-programs.

B Soundness and completeness

B.1 Soundness

Lemma B.1: Given $T \in \mathcal{T}^{\#}$ and $S \in \mathcal{S}^{\#}$. If $\vdash_{\mathcal{AR}^{\#}} T\ Sat^{\#}\ S$ then $elim(T)$ and $elim(S)$ are identically structured. $\Box$
Proof: By an induction on rules and axioms.
Lemma B.2: Given $T \in \mathcal{T}^{\#}$ and $S \in \mathcal{S}^{\#}$. If $\vdash_{\mathcal{AR}^{\#}} T\ Sat^{\#}\ S$ then for each pair of matching subterms $T^{'} \in T$ and $S^{'} \in S$ w.r.t. T and S, $\vdash_{\mathcal{AR}^{\#}} T^{'}\ Sat^{\#}\ S^{'}$ holds. $\Box$
Proof: By an induction on rules and axioms.

The next Lemma claims that if it is possible to prove that a mixed-specification S holds for a gapped-program T then it is possible to prove that a mixed-specification $S^{'}$, obtained from S by removing all $jigsaw$ occurrences, holds for the gapped-program $T^{'}$, obtained from T by removing all $jigsaw$ occurrences. This Lemma implicitly states the soundness of the Jigsaw-rule and the Substitution rule.

Lemma B.3: Assume the predicates $G_C \in \mathcal{G}$ are monotone and compositional. Let $T \in \mathcal{T}^{\#}$, $S \in \mathcal{S}^{\#}$. If $\vdash_{\mathcal{AR}^{\#}} T\ Sat^{\#}\ S$ then $\vdash_{\mathcal{AR}^{\#}} elim(T)\ Sat^{\#}\ elim(S)$. $\Box$
Proof: By induction on the structure of T.

1. $T \in \mathcal{V} \cup \mathcal{O}$, according to the definition of $\mathcal{AR}^{\#}$, $S \in \mathcal{S} \cup \overline{\mathcal{S}}$. Therefore according to the definition of $elim$ for programs and specifications $elim(T) = T$ and $elim(S) = S$ and we can conclude that: $\vdash_{\mathcal{AR}^{\#}} elim(T)\ Sat^{\#}\ elim(S)$.
2. $T = C(T_1, \ldots, T_n)$, according to the definition of $\mathcal{AR}^{\#}$ there are two possible forms of S:
   (a) S is of the form $C(S_1, \ldots, S_n) \triangleright S_{n+1}$. According to the rules in $\mathcal{AR}^{\#}$, $\exists S_1^{'}, \ldots S_{n+1}^{'}$ such that $\vdash_{\mathcal{AR}^{\#}} T_i\ Sat^{\#}\ S_i^{'},\ \ i = 1, n$ and
   $G_C(conc(S_1^{'}), \ldots, conc(S_n^{'}), S_{n+1}^{'})$ holds. Using the R_C rule we get
   $\vdash_{\mathcal{AR}^{\#}} C(T_1, \ldots, T_n)\ Sat^{\#}\ C(S_1^{'}, \ldots, S_n^{'}) \triangleright S_{n+1}^{'}$ which leads to
   $\vdash_{\mathcal{AR}^{\#}} C(T_1, \ldots, T_n)\ Sat^{\#}\ C(S_1, \ldots, S_n) \triangleright S_{n+1}$ by using a sequence of rules, l_1, consisting of only Substitution and Assumption-free rules. By the induction hyp.
   $\vdash_{\mathcal{AR}^{\#}} elim(T_i)\ Sat^{\#}\ elim(S_i^{'}),\ \ i = 1, n$, and by definition $conc(S) = conc(elim(S))$ therefore $G_C(conc(elim(S_1^{'})), \ldots, conc(elim(S_n^{'})), S_{n+1}^{'})$ holds. Using the R_C rule we get
   $\vdash_{\mathcal{AR}^{\#}} C(elim(T_1), \ldots, elim(T_n))\ Sat^{\#}\ C(elim(S_1^{'}), \ldots, elim(S_n^{'})) \triangleright S_{n+1}^{'}$. Therefore, according to the definition of $elim$ we get:
   $\vdash_{\mathcal{AR}^{\#}} elim(C(T_1, \ldots, T_n))\ Sat^{\#}\ C(elim(S_1^{'}), \ldots, elim(S_n^{'})) \triangleright S_{n+1}^{'}$. By applying the

Assumption-free rules in l_1 we would get
$\vdash_{\mathcal{AR}^{\#}} elim(C(T_1,\ldots,T_n))\ Sat^{\#}\ C(elim(S_1),\ldots,elim(S_n)) \rhd S'_{n+1}$. The Substitution rule and the Assumption-free rule do not effect the conclusion of the specification and therefore $S_{n+1} = S'_{n+1}$. According to the definition of $elim$ to specifications we get:
$\vdash_{\mathcal{AR}^{\#}} elim(C(T_1,\ldots,T_n))\ Sat^{\#}\ elim(C(S_1,\ldots,S_n)) \rhd S_{n+1}$.

(b) S is of the form $S \in \mathcal{S}$. According to the rules in $\mathcal{AR}^{\#}$, $\exists S_1,\ldots,S_{n+1} \in \mathcal{S}^{\#}$ such that $\vdash_{\mathcal{AR}^{\#}} T_i\ Sat^{\#}\ S_i,\ \ i = 1,n$ and $G_C(conc(S_1),\ldots,conc(S_n),S_{n+1})$ holds. Therefore, $\vdash_{\mathcal{AR}^{\#}}\ C(T_1,\ldots,T_n)\ Sat^{\#}\ C(S_1,\ldots,S_n) \rhd S_{n+1}$ which leads to $(*)\quad \vdash_{\mathcal{AR}^{\#}} C(T_1,\ldots,T_n)\ Sat^{\#}\ S'$ by using only the Substitution and the Assumption-free rules. The formula (*) leads to $\vdash_{\mathcal{AR}^{\#}}\ C(T_1,\ldots,T_n)\ Sat^{\#}\ S$ by using a sequence, l_1, of rules in $\mathcal{R}$. Like in (a) we get $\vdash_{\mathcal{AR}^{\#}}\ elim(C(T_1,\ldots,T_n))\ Sat^{\#}\ elim(S')$, where $elim(S') = S'$ (since $S' \in \mathcal{S}$). Using the sequence l_1 of rules of $\mathcal{R}$ we get
$\vdash_{\mathcal{AR}^{\#}} elim(C(T_1,\ldots,T_n))\ Sat^{\#}\ elim(S)$, where $elim(S) = S$.

3. $T = \#(T_1^*,T_2^*)$, we consider here only the case in which S is of the form $\#(S_1^*,S_2^*)$. By Lemma B.2 and according to the rules of $\mathcal{AR}^{\#}$, $\vdash_{\mathcal{AR}^{\#}} T_i^*\ Sat^{\#}\ S_i^*,\ \ i = 1,2$ and $\#(S_1^*,S_2^*)$ is well-defined.

Claim: If $\vdash_{\mathcal{AR}^{\#}} T_i^*\ Sat^{\#}\ S_i^*,\ i = 1,2$ and $\#(S_1^*,S_2^*)$ is well-defined then
$\vdash_{\mathcal{AR}^{\#}} elim(\#(T_1^*,T_2^*))\ Sat^{\#}\ elim(\#(S_1^*,S_2^*))$ and $conc(elim(\#(S_1^*,S_2^*)) \Rightarrow conc(elim(S_i^*))$, $i = 1,2$. $\Box$

Proof of claim: Let denote $elim(T_1^*)$ by T_1, $elim(T_2^*)$ by T_2, $elim(S_1^*)$ by S_1 and $elim(S_2^*)$ by S_2. By definition, T_1,T_2,S_1,S_2 are #-free. According to Lemma B.1 T_i and S_i are identically structured. We can also conclude that T_1 and T_2 are unifiable (since S_1 and S_2 are unifiable and T_i and S_i are identically structured). By definition,

$$\begin{aligned} elim(\#(T_1^*,T_2^*)) &= T_1 mgu(T_1,T_2) = T_2 mgu(T_1,T_2)\\ elim(\#(S_1^*,S_2^*)) &= skel_1(S_1)mgu^*(skel_1(S_1),skel_1(S_2))oskel_2(S_1)oskel_2(S_2) = \\ &\quad skel_1(S_2)mgu^*(skel_1(S_1),skel_1(S_2))oskel_2(S_1)oskel_2(S_2). \end{aligned}$$

Let the variables in T_1 be denoted by $X_1,\ldots,X_m$ and let the variables in T_2 be denoted by $Y_1,\ldots,Y_q$. Let the variables of $skel_1(S_1)$ be denoted by $W_1,\ldots,W_n$ and let the variables of $skel_1(S_2)$ be denoted by $V_1,\ldots,V_p$.

Let $mgu(T_1,T_2) = \theta_1 \subseteq \{X_1 \leftarrow T'_1,\ldots,X_m \leftarrow T'_m, Y_1 \leftarrow T''_1,\ldots,Y_q \leftarrow T''_q\}$ and $mgu^*(skel_1(S_1),skel_1(S_2))oskel_2(S_1)oskel_2(S_2) = \theta_2 \subseteq \{W_1 \leftarrow S'_1,\ldots,W_n \leftarrow S'_n, V_1 \leftarrow S''_1,\ldots,V_p \leftarrow S''_p\}$.

We can rewrite the conclusion of the claim as:
$\vdash_{\mathcal{AR}^{\#}} T_i\theta_1\ Sat^{\#}\ skel_1(S_i)\theta_2$ and $conc(skel_1(S_i)\theta_2) \Rightarrow conc(S_i),\ i = 1,2$.

We continue by induction on the structure of T_1:

(a) If $T_1 = X \in \mathcal{V}$ and $T_2 = Y \in \mathcal{V}$ then with out lost of generality $\theta_1 = \{X \leftarrow Y\}$ and $\theta_2 = \{W \leftarrow S_1 \wedge S_2, V \leftarrow S_1 \wedge S_2\}$. Therefore, $T_1\theta_1 = Y$ and $skel_1(S_1)\theta_2 = S_1 \wedge S_2$. Using the variable-axiom $\vdash_{\mathcal{AR}^{\#}} Y\ Sat^{\#}\ S_1 \wedge S_2$. Since θ_2 is valid w.r.t. $skel_2(S_1)$ and w.r.t. $skel_2(S_1)$ we get $(S_1 \wedge S_2) \Rightarrow S_i,\ i = 1,2$.

(b) If $T_1 = X \in \mathcal{V}$ and $T_2 \notin \mathcal{V}$ then with out lost of generality $\theta_1 = \{X \leftarrow T_2\}$ and $\theta_2 = \{W \leftarrow S_2\}$ therefore $T_1\theta_1 = T_2$ and $skel_1(S_1)\theta_2 = S_2$. We know that $\vdash_{\mathcal{AR}^\#} T_2^* \; Sat^\# \; S_2^*$ by the induc. hyp. $\vdash_{\mathcal{AR}^\#} elim(T_2^*) \; Sat^\# \; elim(S_2^*)$ which can be rewritten as $\vdash_{\mathcal{AR}^\#} T_2 \; Sat^\# \; S_2$. Moreover, $conc(skel_1(S_1)\theta_2) = conc(S_2) \Rightarrow conc(S_1)$ since θ_2 is valid w.r.t. $skel_2(S_1)$.

(c) If $T_1 \in \mathcal{O}$ then we can conclude that $T_2 \in \mathcal{V}$ since T_1 and T_2 are unifiable. Therefore this case is similar to case (b).

(d) $T_1 = C(T_1^{'}, \ldots, T_n^{'}) \in \mathcal{T}^\#$. Without loss of generality assume that T_1 contains variables. In this case S_2, S_1 and T_2 are of the form $S_1 = C(S_1^{'}, \ldots, S_n^{'}) \rhd S^{'}$, $S_2 = C(S_1^{''}, \ldots, S_n^{''}) \rhd S^{''}$ and $T_2 = C(T_1^{''}, \ldots, T_n^{''})$.

Intuitively, this case is proved as follows:

i. We decompose θ_1, the most general unifier of T_1 and T_2, into n sub-substitutions such that θ_1^i is the most general unifier of $T_i^{'} \in T_1$ and $T_i^{''} \in T_2$. This decomposition is possible since each variable in T_1 and in T_2 appears at most once.

ii. In a similar way, we decompose θ_2 into $n+1$ sub-substitutions such that $skel(S_i^{'})\theta_2^i = skel_1(S_i^{''})\theta_2^i$. Moreover, by definition $S^{'}\theta_2^{n+1} = S^{''}\theta_2^{n+1} = S^{'} \wedge S^{''}$.

iii. By Lemma B.2 and according to the induction hyp. we show that $\vdash_{\mathcal{AR}^\#} T_i^{'}\theta_1^i \; Sat^\# \; skel_1(S_i^{'})\theta_2^i$, $i = 1, n$.

iv. According to the monotonicity and compositionality of G_C we show that $G_C(conc(skel_1(S_1^{'})\theta_2^1, \ldots, conc(skel_1(S_n^{'})\theta_2^n, S^{'} \wedge S^{''})$ hold.

v. Then we compose the subprograms by applying R_C and conclude that $\vdash_{\mathcal{AR}^\#} C(T_1^{'}\theta_1^1, \ldots, T_n^{'}\theta_1^n) \; Sat^\# \; C(skel_1(S_1^{'})\theta_2^1, \ldots, skel_1(S_n^{'})\theta_2^n) \rhd S^{'} \wedge S^{''}$, which can be written as $\vdash_{\mathcal{AR}^\#} elim(\#(T_1^*, T_2^*) \; Sat^\# \; elim(\#(S_1^*, S_2^*))$.

Next we formally present the proof of this case.

It is possible to represent θ_1 as a union of n substitutions $\theta_1 = \{\theta_1^1, \ldots, \theta_1^n\}$ such that $T_1\theta_1 = C(T_1^{'}, \ldots, T_n^{'})\theta_1 = C(T_1^{'}\theta_1^1, \ldots, T_n^{'}\theta_1^n)$ and $T_2\theta_1 = C(T_1^{''}, \ldots, T_n^{''})\theta_1 = C(T_1^{''}\theta_1^1, \ldots, T_n^{''}\theta_1^n)$.

It is possible to represent θ_2 as a union of $n+1$ substitutions $\theta_2 = \{\theta_2^1, \ldots, \theta_2^{n+1}\}$ such that $skel_1(S_1)\theta_2 = skel_1(C(S_1^{'}, \ldots, S_n^{'}) \rhd S^{'})\theta_2 = C(skel_1(S_1^{'})\theta_2^1, \ldots, skel_1(S_n^{'})\theta_2^n) \rhd S^{'} \wedge S^{''}$ (note that $S^{'}\theta_2^{n+1} = S^{'} \wedge S^{''}$) and $skel_1(S_2)\theta_2 = skel_1(C(S_1^{''}, \ldots, S_n^{''}) \rhd S^{''})\theta_2 = C(skel_1(S_1^{''})\theta_2^1, \ldots, skel_1(S_n^{''})\theta_2^n) \rhd S^{'} \wedge S^{''}$.

If $T_1 = C(T_1^{'}, \ldots, T_n^{'})$ then $T_1^* = C(T_1^{'*}, \ldots, T_n^{'*})$ such that $T_i^{'} = elim(T_i^{'*})$ and if $S_1 = C(S_1^{'}, \ldots, S_n^{'}) \rhd S^{'}$ then $S_1^* = C(S_1^{'*}, \ldots, S_n^{'*}) \rhd S^{'*}$ such that $S_i^{'} = elim(S_i^{'*})$. Based on Lemma B.2 $\vdash_{\mathcal{AR}^\#} T_i^{'*} \; Sat^\# \; S_i^{'*}$. By the induc. hyp. $\vdash_{\mathcal{AR}^\#} T_i^{'} \; Sat^\# \; S_i^{'}$. In a similar way we can prove that $\vdash_{\mathcal{AR}^\#} T_i^{''} \; Sat^\# \; S_i^{''}$. Moreover, it can be shown that $\#(S_i^{'}, S_i^{''})$ is well-defined therefore the conditions of the claim hold for $T_i^{'}$, $T_i^{''}$, $S_i^{'}$, $S_i^{''}$, θ_1^i and θ_2^i. By the induction hyp. of the claim $\vdash_{\mathcal{AR}^\#} T_i^{'}\theta_1^i \; Sat^\# \; skel_1(S_i^{'})\theta_2^i$ and $conc(skel_1(S_i^{'})\theta_2^i) \Rightarrow conc(S_i^{'})$. It is easy to prove that $G_C(conc(S_1^{'}), \ldots, conc(S_n^{'}), S^{'})$ holds. By the monotonicity of G_C we get

$(*)$ $G_C(conc(skel_1(S_1^{'})\theta_2^1), \ldots conc(skel_1(S_n^{'})\theta_2^n), S^{'})$ holds. In a similar way we can show that

$(**)$ $G_C(conc(skel_1(S_1^{''})\theta_2^1), \ldots conc(skel_1(S_n^{''})\theta_2^n), S^{''})$ holds. Since θ_2 is a unifier of

S_1 and S_2 we know that $skel_1(S_i^{'})\theta_2^i = skel_1(S_i^{''})\theta_2^i$ therefore by the compositionality of G_C, (*) and (**) we get
$G_C(conc(skel_1(S_1^{''})\theta_2^1), \ldots conc(skel_1(S_n^{''})\theta_2^n), S^{'} \wedge S^{''})$ holds. Applying the R_C rule we get $\vdash_{\mathcal{AR}^\#} T_i\theta_1\ Sat^\#\ skel_1(S_i)\theta_2,\ i = 1, 2$. Since $conc(skel_1(S_i)\theta_2) = S^{'} \wedge S^{''}$ we conclude that $conc(skel_1(S_i)\theta_2) \Rightarrow conc(S_i),\ i = 1, 2$. $\Box$

end of claim's proof.

Theorem B.4: (Soundness:) Assume the predicates $G_C \in \mathcal{G}$ are monotone and compositional. For every $T^{'} \in \mathcal{T}^\#$ and $T \in \mathcal{T}$ such that $T = elim(T^{'})$ and for every $S \in \mathcal{S}$: If $\vdash_{\mathcal{AR}^\#} T^{'}\ Sat^\#\ S$ then $\models T\ Sat\ S$. $\Box$

Proof: By induction on the structure of $T^{'}$.

1. If $T^{'} \in \mathcal{O}$ then $T^{'} = T$ and the rules and axioms used for showing that $\vdash_{\mathcal{AR}^\#} T\ Sat^\# S$ are also in $\mathcal{AR}$ and thus $\vdash_{\mathcal{AR}} T\ Sat\ S$ holds. By the soundness of $\mathcal{AR}$ we get $\models T\ Sat\ S$.

2. If $T^{'} = C(T_1^{'}, \ldots, T_n^{'})$ then by the definition of $elim(T^{'})$ we know that $\exists T_1, \ldots, T_n$ such that $T = C(T_1, \ldots, T_n)$ and $T_i = elim(T_i^{'}),\ i = 1, n$. According to $\mathcal{AR}^\#$ there exist $S_i \in \mathcal{S}^\#,\ i = 1, n$ such that $\vdash_{\mathcal{AR}^\#} T_i^{'}\ Sat^\#\ S_i,\ i = 1, n$ and $G_C(conc(S_1), \ldots, conc(S_n), S)$ holds. Using the R_C rule
$\vdash_{\mathcal{AR}^\#} C(T_1^{'}, \ldots T_n^{'})\ Sat^\#\ C(S_1, \ldots S_n) \rhd S^{'}$ which leads to $\vdash_{\mathcal{AR}^\#} C(T_1^{'}, \ldots T_n^{'})\ Sat^\#\ S$ using a sequence, l_1, of rules.

   For $1 \leq i \leq n$: if $S_i \in \mathcal{S}$ then by the induc. hyp. $\models T_i\ Sat\ S_i$ and by the completeness of $\mathcal{AR}, \vdash_{\mathcal{AR}} T_i\ Sat\ S_i$. If $S_i \notin \mathcal{S}$ then $\vdash_{\mathcal{AR}^\#} T_i^{'}\ Sat^\#\ S_i$ which leads to $\vdash_{\mathcal{AR}^\#} T_i^{'}\ Sat^\#\ elim(S_i)$ using Sub rules. We know that $elim(T_i^{'}) = T_i \in \mathcal{T}$ therefore $elim(S_i)$ is assumption-free (Lemma B.1). By using the Assumption-free rule we get: $\vdash_{\mathcal{AR}^\#} T_i^{'}\ Sat^\#\ conc(elim(S_i))$, $i = 1, n$ which implies that $\vdash_{\mathcal{AR}^\#} T_i^{'}\ Sat^\#\ conc(S_i),\ i = 1, n$. Using Lemma B.3 $\vdash_{\mathcal{AR}^\#} T_i\ Sat^\#\ conc(S_i),\ i = 1, n$. By the induction hyp. $\models T_i\ Sat\ conc(S_i),\ i = 1, n$. By the completeness of the basic system $\vdash_{\mathcal{AR}} T_i\ Sat\ conc(S_i),\ i = 1, n$ and therefore we can prove in $\mathcal{AR}$ that $\vdash_{\mathcal{AR}} T\ Sat\ S$ and by the soundness of $\mathcal{AR}$ we can conclude that $\models T\ Sat\ S$.

3. If $T^{'} = \#(T_1^{'}, T_2^{'})$ then by Lemma B.3 $\vdash_{\mathcal{AR}^\#} elim(T^{'})\ Sat^\#\ S$. By the induction hyp. $\models elim(T^{'})\ Sat\ S$. $\Box$

B.2 Completeness

Lemma B.5: Let $T \in \mathcal{T}^\#$ be #-free and let $S \in \mathcal{S}^\#$. If $\vdash_{\mathcal{AR}^\#} T\ Sat^\#\ S$ then there exists $S^{'} \in \mathcal{S}^\#$ such that:

1. $\vdash_{\mathcal{AR}^\#} T\ Sat^\#\ S^{'}$

2. $conc(S) = conc(S^{'})$

3. T and $S^{'}$ are identically structured

4. for all matching subterms $T^* \in T$ and $S^* \in S^{'}$ w.r.t. T and $S^{'}$, $\vdash_{\mathcal{AR}^\#} T^*\ Sat^\#\ S^*$, holds. $\Box$

Proof: By induction on the structure of T,

1. if $T \in \mathcal{O}$ then according to the definition of $\mathcal{AR}^{\#}$, $S \in \mathcal{S}$, we take $S^{'} = S$.

2. if $T \in \mathcal{V}$ then according to the definition of $\mathcal{AR}^{\#}$, $S \in \overline{\mathcal{S}}$, we take $S^{'} = S$.

3. if $T = C(T_1, \ldots T_n)$ then according to Lemma B.1 we have to consider two cases:

   (a) $S = C(S_1, \ldots, S_n) \triangleright S_{n+1}$. We are given that
   $(*)$ $\vdash_{\mathcal{AR}^{\#}} C(T_1, \ldots, T_n)\ Sat^{\#}\ C(S_1, \ldots S_n) \triangleright S_{n+1}$. To derive (*) in $\mathcal{AR}^{\#}$ the R_C rule must have been applied at least once such that $\exists S_1^{'}, \ldots S_{n+1}^{'}$:
   $\vdash_{\mathcal{AR}^{\#}} T_i\ Sat^{\#}\ S_i^{'}$, $i = 1, n$, $G_C(conc(S_1^{'}), \ldots, conc(S_n^{'}), S_{n+1}^{'})$ holds and $(**)$ $\vdash_{\mathcal{AR}^{\#}}$ $C(T_1, \ldots T_n)\ Sat^{\#} C(S_1^{'}, \ldots S_n^{'}) \triangleright S_{n+1}^{'}$.
   The formula (*) has been derived from (**) by using only the Ass. and Sub. rules. Both rules do not change the conclusion $S_{n+1}^{'}$ and therefore $S_{n+1} = S_{n+1}^{'}$.
   By the induction hyp. for every $1 \leq i \leq n$, $\exists S_i^{''}$ such that:

      i. $\vdash_{\mathcal{AR}^{\#}} T_i\ Sat^{\#}\ S_i^{''}$
      ii. $conc(S_i^{'}) = conc(S_i^{''})$
      iii. T_i and $S_i^{''}$ are identically structured
      iv. for all matching subterms $T^* \in T_i$ and $S^* \in S_i^{''}$ w.r.t. T_i and $S_i^{''}$, $\vdash_{\mathcal{AR}^{\#}}$ $T^*\ Sat^{\#}\ S^*$, holds.

   Therefore, $G_C(conc(S_1^{''}), \ldots, conc(S_n^{''}), S_{n+1})$ holds and we can conclude that:

      i. $\vdash_{\mathcal{AR}^{\#}} C(T_1, \ldots, T_n)\ Sat^{\#}\ C(S_1^{''}, \ldots, S_n^{''}) \triangleright S_{n+1}$,
      ii. $conc(C(S_1, \ldots, S_n) \triangleright S_{n+1}) = conc(C(S_1^{''}, \ldots, S_n^{''}) \triangleright S_{n+1})$,
      iii. $C(T_1, \ldots, T_n)$ and $C(S_1^{''}, \ldots, S_n^{''}) \triangleright S_{n+1}$ are identically structured,
      iv. for all matching subterms $T^* \in T$ and $S^* \in C(S_1^{''}, \ldots, S_n^{''}) \triangleright S_{n+1}$, $\vdash_{\mathcal{AR}^{\#}}$ $T^*\ Sat^{\#}\ S^*$, holds.

   (b) $S \in \mathcal{S}$. In this case, $T \in \mathcal{T}$ and we take $S^{'} = S$. $\Box$

Lemma B.6: Assume the predicates in $\mathcal{G}$ are compositional. Let $T_1, T_2 \in \mathcal{T}^{\#}$ be #-free and let $S \in \mathcal{S}^{\#}$. If $\vdash_{\mathcal{AR}^{\#}} \#(T_1, T_2)\ Sat^{\#}\ S$ then $\exists S_1, S_2$ such that:

1. $\vdash_{\mathcal{AR}^{\#}} T_i\ Sat^{\#}\ S_i$, $i = 1, 2$,

2. $\#(S_1, S_2)$ is well-defined,

3. $conc(S) = conc(S_1) = conc(S_2)$.$\Box$

Proof: We are given that $(*)$ $\vdash_{\mathcal{AR}^{\#}} \#(T_1, T_2)\ Sat^{\#}\ S$. To derive (*) in $\mathcal{AR}^{\#}$ the jigsaw rule must have been applied at least once such that, there exist S_1^*, S_2^*:

1. $\vdash_{\mathcal{AR}^{\#}} T_i\ Sat^{\#}\ S_i^*$, $i = 1, 2$,

2. $\#(S_1^*, S_2^*)$ is well-defined,

3. $conc(S) = conc(S_1^*) \wedge conc(S_2^*)$.

By Lemma B.5 applied for T_1 and T_2 which are #-free and the above there exist $S_1^{'}$ and $S_2^{'}$:

1. $\vdash_{\mathcal{AR}^{\#}} T_i\ Sat^{\#}\ S_i^{'}$, $i = 1, 2$,

2. $\#(S_1', S_2')$ is well-defined,

3. T_i and S_i' are identically structured, $i = 1, 2$,

4. $conc(S) = conc(S_1') \wedge conc(S_2')$.

Next a new specification S_1'' is built for T_1 in the following way:

1. For every meta-variable X in T_1, S_1'' contains the sub-specification of S_2' which corresponds to the subprogram to be substituted for X when unifying T_1 and T_2.

2. For every primitive object in T_1, S_1'' contains the sub-specification of S_1' which corresponds to this object.

3. For every composed sub-program in T_1, S_1'' contains a sub-specification which is the conjunction of the sub-specification of S_1' which corresponds to this sub-program and the sub-specification of S_2' which corresponds to this sub-program.

In a similar way a new specification S_2'' is built for T_2.

Based on the well-definedness of $\#(S_1', S_2')$, the monotonicity and compositionality of the predicates $G_C \in \mathcal{G}$ it is possible to show that $\vdash_{\mathcal{AR}^\#} T_i\ Sat^\#\ S_i''$, $i = 1, 2$. According to the definition of S_1'' and S_2'' the following holds:

1. $\#(S_1'', S_2'')$ is well-defined,

2. $conc(S_1'') = conc(S_2'') = conc(S_1') \wedge conc(S_2') = conc(S)$. $\Box$

Lemma B.7: Let $T \in \mathcal{T}^\#$ and $S \in \mathcal{S}^\#$. If $\vdash_{\mathcal{AR}^\#} elim(T)\ Sat^\#\ S$ then there exists $S' \in \mathcal{S}^\#$ such that $\vdash_{\mathcal{AR}^\#} T\ Sat^\#\ S'$ and $conc(S) = conc(S')$. $\Box$
Proof: By induction on the structure of T.

1. if $T \in \mathcal{O} \cup \mathcal{V}$ then $elim(T) = T$ and therefore $\vdash_{AR^\#} T\ Sat^\#\ S$.

2. if $T = C(T_1, \ldots, T_n)$ then $elim(T) = C(elim(T_1), \ldots, elim(T_n))$. We are given that $\vdash_{\mathcal{AR}^\#} C(elim(T_1), \ldots, elim(T_n))\ Sat^\#\ S$. According to Lemma B.1 there are two cases to consider:

   (a) $S = C(S_1, \ldots, S_n) \rhd S_{n+1}$. We are given that
   $(*)\quad \vdash_{\mathcal{AR}^\#} C(elim(T_1) \ldots, elim(T_n))\ Sat^\# C(S_1, \ldots, S_n) \rhd S_{n+1}$. To derive (*) in $\mathcal{AR}^\#$ the R_C rule has been applied at least once such that $\exists S_1', \ldots, S_{n+1}'$:
   $\vdash_{\mathcal{AR}^\#} elim(T_i)\ Sat^\#\ S_i'$, $i = 1, n$, $G_C(conc(S_1'), \ldots conc(S_n'), S_{n+1}')$ holds and therefore $(**)\quad \vdash_{\mathcal{AR}^\#} C(elim(T_1), \ldots elim(T_n))\ Sat^\#\ C(S_1', \ldots, S_n') \rhd S_{n+1}'$. The formula (*) has been derived from (**) by a sequence of applications of the Sub. and Ass. rules therefore $S_{n+1}' = S_{n+1}$.
   By the induction hyp. there exist S_i'', $i = 1, n$ such that $\vdash_{\mathcal{AR}^\#} T_i\ Sat^\#\ S_i''$ where $conc(S_i'') = conc(S_i')$, $i = 1, n$ therefore we can conclude:
   $(***)\quad \vdash_{\mathcal{AR}^\#} C(T_1, \ldots T_n)\ Sat^\#\ C(S_1'', \ldots, S_n'') \rhd S_{n+1}$.

   (b) $S \in \mathcal{S}$. We are given that
   $(*)\quad \vdash_{\mathcal{AR}^\#} C(elim(T_1), \ldots, elim(T_n))\ Sat^\#\ S$ which implies that $elim(T_i) \in \mathcal{T}$, $i = 1, n$. To derive (*) the R_C rule has been applied at least once such that $\exists S_1', \ldots, S_{n+1}'$:

$\vdash_{\mathcal{AR}^\#} elim(T_i)\ Sat^\#\ S_i^{'},\ i = 1,n,\ G_C(conc(S_1^{'},\ldots,conc(S_n^{'}),S_{n+1}^{'})$ holds and therefore,
$(**)\ C(elim(T_1),\ldots,elim(T_n))\ Sat^\#\ C(S_1^{'},\ldots S_n^{'}) \triangleright S_{n+1}^{'}$. Applying the Ass. rule:
$(***)\ C(elim(T_1),\ldots,elim(T_n))\ Sat^\#\ S_{n+1}^{'}$. The formula (*) can be derived from (***) by using a sequence of rules, l_1, in $\mathcal{R}$.
By the induction hyp. $\exists S_1^{''},\ldots S_n^{''}$ such that $conc(S_i) = conc(S_i^{''})$ and $\vdash_{\mathcal{AR}^\#} T_i\ Sat^\#\ S_i^{''}$ therefore $\vdash_{\mathcal{AR}^\#} C(T_1,\ldots,T_n)\ Sat^\#\ C(S_1^{''},\ldots S_n^{''}) \triangleright S_{n+1}^{'}$. Applying Ass. and Sub. we get $\vdash_{\mathcal{AR}^\#} C(T_1,\ldots,T_n)\ Sat^\#\ S_{n+1}^{'}$. Applying the sequence l_1 would lead to $\vdash_{\mathcal{AR}^\#} C(T_1,\ldots,T_n)\ Sat^\#\ S$.

3. if $T = \#(T_1,T_2)$. We are given that $\vdash_{\mathcal{AR}^\#} elim(\#(T_1,T_2))\ Sat^\#\ S$. We have to consider three cases:

   (a) $elim(\#(T_1,T_2)) \in \mathcal{O}$,
   (b) $elim(\#(T_1,T_2)) \in \mathcal{V}$,
   (c) $elim(\#(T_1,T_2)) = C(T_1^*,\ldots,T_n^*)$.

   We bring here only the third case. If $elim(\#(T_1,T_2)) = C(T_1^*,\ldots,T_n^*)$ then there are two cases: 1) $S = C(S_1,\ldots S_n) \triangleright S_{n+1}$, 2) $S \in \mathcal{S}$. We bring here only the first case. Again there are two cases:

   (a) $elim(T_1) = X \in \mathcal{V}$ and $elim(T_2) = C(T_1^*,\ldots,T_n^*)$,
   (b) $elim(T_1) = C(T_1^{'},\ldots T_n^{'})$ and $elim(T_2) = C(T_1^{''},\ldots T_n^{''})$.

We bring here the proof of case (b). We know that $elim(\#(T_1,T_2)) = C(T_1^*,\ldots,T_n^*)$ and that $elim(\#(C(T_1^{'},\ldots,T_n^{'}),C(T_1^{''},\ldots,T_n^{''}))) = C(T_1^*,\ldots,T_n^*)$. Since every meta variable in T_1 and T_2 appears at most once we can conclude that
$C(elim(\#(T_1^{'},T_1^{''})),\ldots,elim(\#(T_n^{'},T_n^{''}))) = C(T_1^*,\ldots T_n^*)$. Thus we are given that
$\vdash_{\mathcal{AR}^\#} C(elim(\#(T_1^{'},T_1^{''})),\ldots,elim(\#(T_n^{'},T_n^{''})))\ Sat^\#\ C(S_1,\ldots,S_n) \triangleright S_{n+1}$ according to the definition of $\mathcal{AR}^\#$, $\exists S_1^{'},\ldots S_{n+1}^{'}$ such that $\vdash_{\mathcal{AR}^\#} elim(\#(T_i^{'},T_i^{''}))\ Sat^\#\ S_i^{'},\ i = 1,n,$ and $G_C(conc(S_1^{'}),\ldots conc(S_n^{'}),S_{n+1})$ holds. By the induction hyp. there exists $S_i^{''}$ such that $\vdash_{\mathcal{AR}^\#} \#(T_i^{'},T_i^{''})\ Sat^\#\ S_i^{''},\ conc(S_i^{'}) = conc(S_i^{''}),\ i = 1,n$. Therefore,
$\vdash_{\mathcal{AR}^\#} C(\#(T_1^{'},T_1^{''}),\ldots,\#(T_n^{'},T_n^{''}))\ Sat^\#\ C(S_1^{''},\ldots,S_n^{''}) \triangleright S_{n+1}$. Since $T_i^{'}$ and $T_i^{''}$ are part of $elim(T_1)$ and $elim(T_2)$, resp. they are #-free. According to Lemma B.6 there exist specifications:
$S_{1_1}^{'},\ldots S_{n_1}^{'}$ and $S_{1_2}^{'},\ldots S_{n_2}^{'}$ such that

1. $\vdash_{\mathcal{AR}^\#} T_i^{'}\ Sat^\#\ S_{i_1}^{'},\ \vdash_{\mathcal{AR}^\#} T_i^{''}\ Sat^\#\ S_{i_2}^{'},\ i = 1,n.$
2. $\#(S_{i_1}^{'},S_{i_2}^{'})$ is well-defined, $i = 1,n$.
3. $conc(S_i^{''}) = conc(S_{i_1}^{'}) = conc(S_{i_2}^{'})$.

Therefore we can conclude that $\vdash_{\mathcal{AR}^\#} C(T_1^{'},\ldots,T_n^{'})\ Sat^\#\ C(S_{1_1}^{'},\ldots,S_{n_1}^{'}) \triangleright S_{n+1}$,
$\vdash_{\mathcal{AR}^\#} C(T_1^{''},\ldots,T_n^{''})\ Sat^\#\ C(S_{1_2}^{'},\ldots,S_{n_2}^{'}) \triangleright S_{n+1}$.
Based the the induction hyp.
$\vdash_{\mathcal{AR}^\#} T_1\ Sat^\#\ C(S_{1_1}^{'},\ldots,S_{n_1}^{'}) \triangleright S_{n+1}$,
$\vdash_{\mathcal{AR}^\#} T_2\ Sat^\#\ C(S_{1_2}^{'},\ldots,S_{n_2}^{'}) \triangleright S_{n+1}$.

It is possible to prove that
$\#(C(S^{'}_{1_1},\ldots,S^{'}_{n_1}) \rhd S_{n+1}, C(S^{'}_{1_2},\ldots,S^{'}_{n_2}) \rhd S_{n+1})$ is well-defined and thus the jigsaw rule gives $\vdash_{\mathcal{AR}^\#} \#(T_1,T_2)\ Sat^\#\ \#(C(S^{'}_{1_1},\ldots,S^{'}_{n_1}) \rhd S_{n+1}, C(S^{'}_{1_2},\ldots,S^{'}_{n_2}) \rhd S_{n+1})$ and the Substitution rule gives
$\vdash_{\mathcal{AR}^\#} \#(T_1,T_2)\ Sat^\#\ C(S^{''}_1,\ldots,S^{''}_n) \rhd (S_{n+1} \wedge S_{n+1})$. $\Box$

Theorem B.8: (Completeness) Assume the predicates $G_C \in \mathcal{G}$ are monotone. For every $T^{'} \in \mathcal{T}^\#$ and $T \in \mathcal{T}$ such that $T = elim(T^{'})$ and for every $S \in \mathcal{S}$: If $\models T\ Sat\ S$ then $\vdash_{\mathcal{AR}^\#} T^{'}\ Sat^\#\ S$. $\Box$

Proof: By the relative completeness of $\mathcal{AR}$, $\vdash_{\mathcal{AR}} T\ Sat\ S$. It is easy to show that $\vdash_{\mathcal{AR}^\#} T\ Sat\ S$. By replacing T by $elim(T^{'})$ we get $\vdash_{\mathcal{AR}^\#} elim(T^{'})\ Sat\ S$. According to Lemma B.7 we conclude $\vdash_{\mathcal{AR}^\#} T^{'}\ Sat\ S$. $\Box$

C Completing the example of section 4.6

C.1 Verifying $Consumer^c$

1. Using the basic guard axiom:
$$more = 1\ \ Sat^\#\ \ q_6$$
where,
$$q_6 \stackrel{df.}{=} [true, true, more = 1]$$

2. Using axioms and proof rules of the basic system:
$$\begin{array}{ll} D_{12}?temp; new := temp; & \\ (While\ temp = 63, & Sat^\#\ \ q_7 \\ \quad D_{12}?temp; new := new + temp) & \end{array}$$
where,
$$\begin{array}{lll} q_7 & \stackrel{df.}{=} & [true, co \wedge more = 1, new = si \wedge more = 1] \\ si & \stackrel{df.}{=} & \Sigma_{j \in A_{last}} val(\pi[j]) \end{array}$$

3. Using a variable axiom:
$$M\ \ Sat^\#\ \ \overline{q_8}$$
where,
$$q_8 \stackrel{df.}{=} [true, new = si \wedge more = 1, new = si \wedge (more \neq 1 \rightarrow new \geq 100)]$$

4. Using (2), (3) and the sequential composition rule:
$$\begin{array}{ll} D_{12}?temp; new := temp; & \\ (While\ temp = 63, & Sat^\#\ \ (q_7\ ;\ \overline{q_8}) \rhd q_9 \\ \quad D_{12}?temp; new := new + temp); & \\ M & \end{array}$$
where,
$$q_9 \stackrel{df.}{=} [true, co \wedge more = 1, new = si \wedge (more \neq 1 \rightarrow new \geq 100)]$$

5. Using the basic output axiom:

$$D_{21}!more \;\; Sat^{\#} \;\; q_{10}$$

where,

$$\begin{aligned} q_{10} \stackrel{df.}{=} \;& [true, new = si \wedge (more \neq 1 \rightarrow new \geq 100), \\ & new = si' \wedge (more \neq 1 \rightarrow new \geq 100) \wedge chan(\pi[last]) = D_{21}] \end{aligned}$$

6. Using (4), (5) and the sequential composition rule:

$$\begin{array}{ll} D_{12}?temp; new := temp; & \\ (While\; temp = 63, & Sat^{\#} \;\; ((q_7 \; ; \; \overline{q_8}) \rhd q_9 \; ; \; q_{10}) \rhd q_{11} \\ \quad D_{12}?temp; new := new + temp); & \\ M; & \\ D_{21}!more & \end{array}$$

where,

$$q_{11} \stackrel{df.}{=} [true, co \wedge more = 1, new = si' \wedge (more \neq 1 \rightarrow new \geq 100) \wedge chan(\pi[last]) = D_{21}]$$

7. Using (1), (6) and the while composition rule:

$$\begin{array}{ll} Consumer^c \;\; Sat^{\#} & (While\; q_6, \\ & \quad ((q_7 \; ; \; \overline{q_8}) \rhd q_9 \; ; \; q_{10}) \rhd q_{11} \\ &) \rhd q_{12} \end{array}$$

where,

$$q_{12} \stackrel{df.}{=} [true, init, new \geq 100 \wedge new = si']$$

C.2 Verifying *Communication*

Using the parallel composition rule and the specifications S_1 and S_2 verified for $Producer^c$ and $Consumer^c$, respectively:

$$Communication \;\; Sat^{\#} \;\; (S_1 \,\|\, S_2) \rhd q_{13}$$

where,

$$q_{13} \stackrel{df.}{=} [I, I \wedge init, prime(new) \wedge new \geq 100]$$

C.3 Verifying $Producer^p$

1. Using basic guard axiom:

$$more = 1 \;\; Sat^{\#} \;\; p_1$$

where, $p_1 \stackrel{df.}{=} [I, I, more = 1]$

2. Using axioms and proof rules of the basic system:

$$\begin{array}{ll} (While\ found = 0, & \\ \quad prod, j, found := prod + 1, 1, 1; & \\ \quad (while\ j \leq max \wedge found \neq 0, & Sat^{\#}\ \ p_2 \\ \quad\quad found, j := mod(prod, array[j]), j + 1)); & \\ max, array[max + 1], found := max + 1, prod, 0 & \end{array}$$

where,

$$p_2 \stackrel{df.}{=} [I, I \wedge co \wedge pr, I \wedge co \wedge pr \wedge prime(prod)]$$

3. Using a variable axiom:

$$Z\ \ Sat^{\#}\ \ \overline{p_3}$$

where,

$$p_3 \stackrel{df.}{=} [I, I \wedge co \wedge pr \wedge prime(prod), I \wedge co \wedge pr]$$

4. Using (1), (2), (3), the sequential and the while composition rules:

$$\begin{array}{lll} Producer^p & Sat^{\#} & (While\ p_1, \\ & & \quad (p_2; \overline{p_3}) \rhd p_4 \\ & &) \rhd p_5 \end{array}$$

where,

$$\begin{array}{rcl} p_4 & \stackrel{df.}{=} & [I, I \wedge co \wedge pr, I \wedge co \wedge pr] \\ p_5 & \stackrel{df.}{=} & [I, I \wedge init, I] \end{array}$$

C.4 Verifying $Consumer^p$

1. Using the basic guard axiom:

$$more = 1\ \ Sat^{\#}\ \ p_6$$

where

$$p_6 \stackrel{df.}{=} [true, true, more = 1]$$

2. Using a variable axiom:

$$V\ \ Sat^{\#}\ \ \overline{p_7}$$

where,

$$p_7 \stackrel{df.}{=} [true, co \wedge more = 1, new = si \wedge more = 1]$$

3. Using axiom and rules of the basic system:

$$\begin{array}{ll} (new \geq 100; more := 0 & \\ \Box & Sat^{\#}\ \ p_8 \\ new < 100; skip) & \end{array}$$

where,

$$p_8 \stackrel{df.}{=} [true, new = si \wedge more = 1, new = si \wedge (more \neq 1 \rightarrow new \geq 100)]$$

4. Using (2), (3) and the sequential composition rule:

$$\begin{array}{ll} V; & \\ (new \geq 100; more := 0 & \\ \Box & Sat^{\#} \;\; (\; \overline{p_7} \; ; \; p_8 \;) \rhd p_9 \\ new < 100; skip) & \end{array}$$

where,

$$p_9 \stackrel{df.}{=} [true, co \wedge more = 1, new = si \wedge (more \neq 1 \rightarrow new \geq 100)]$$

5. Using a variable axiom:

$$N \;\; Sat^{\#} \;\; \overline{p_{10}}$$

where,

$$p_{10} \stackrel{df.}{=} [true, new = si \wedge (more \neq 1 \rightarrow new \geq 100), co]$$

6. Using (4), (5) and the sequential composition rule:

$$\begin{array}{ll} V; & \\ (new \geq 100; more := 0 & \\ \Box & Sat^{\#} \;\; ((\; \overline{p_7} \; ; \; p_8 \;) \rhd p_9 \; ; \; \overline{p_{10}} \;) \rhd p_{11} \\ new < 100; skip); & \\ N & \end{array}$$

where,

$$p_{11} \stackrel{df.}{=} [true, co \wedge more = 1, co]$$

7. Using (6) and the while composition rule:

$$\begin{array}{lll} Consumer^p & Sat^{\#} & (While \; p_6, \\ & & \quad ((\overline{p_7} \; ; \; p_8) \rhd p_9 \; ; \; \overline{p_{10}}) \rhd p_{11} \\ & &) \rhd p_{12} \end{array}$$

where,

$$p_{12} \stackrel{df.}{=} [true, init, true]$$

C.5 Verifying *Primes*

Using the parallel composition rule and the specifications S_3 and S_4 verified for $Producer^p$ and $Consumer^p$, respectively:

$$Primes \;\; Sat^{\#} \;\; (S_3 \,\|\, S_4) \rhd p_{13}$$

where,

$$p_{13} \stackrel{df.}{=} [I, I \wedge init, I]$$

C.6 Verifying $\#(Primes, Communication)$

1. Using the jigsaw rule and the specifications verified for $Primes$ and $Communication$ we verify that: $p_2 \Rightarrow q_2$, $q_3 \Rightarrow p_3$, $q_7 \Rightarrow p_7$, $p_8 \Rightarrow q_8$ and $q_{10} \Rightarrow p_{10}$ and therefore

$$\#(Primes, Communication)\ \ Sat^{\#}\ \ \#(\ (S_1 \,\|\, S_2) \triangleright q_{13}\ ,\ (S_3 \,\|\, S_4) \triangleright p_{13}\ )$$

2. Using (1) the substitution rule and the assumption-freeness rule:

$$\#(Primes, Communication)\ \ Sat^{\#}\ \ [I, I \wedge init, prime(new) \wedge new \geq 100]$$