Technion - Israel Institute of Technology
Computer Science Department

Program Composition via Unification

by
L. Fix, N. Francez and O. Grumberg

Technical Report #722
May 1992
Program composition and compositional proof systems have proven themselves important for simplifying the design and the 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 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 \( T_1 \). In many cases, 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 which does not support such a development 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 $\mathcal{C} \in \mathcal{C}$ 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 $\mathcal{C}(T_1, \ldots, T_n) \in \mathcal{T}$. \( \square \)

Let $B$ be a set of semantic objects (behaviors). Let $\varphi(B) \subseteq \wp(B)$ be a subset of the power set of $B$ which has a $\subseteq$-minimal element, denoted by $\Theta$.

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

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 $\varphi(B)$. 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, $\land$. The semantics of $\mathcal{S}$ is given by a function $\mathcal{BEH}_\mathcal{S} : \mathcal{S} \rightarrow \varphi(B)$, such that: $\mathcal{BEH}_\mathcal{S}[S_1 \land S_2] = \mathcal{BEH}_\mathcal{S}[S_1] \land \mathcal{BEH}_\mathcal{S}[S_2]$.
A specification $S$ is a tautology if $BEH_S[S] = B$. \(\square\)

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

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

1. A (recursive) set $\mathcal{A} \subseteq \Phi$ of formulas called axioms, s.t. if "$T \text{ Sat } S$" $\in \mathcal{A}$ 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 $S^{n+1}$.
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:

   $$\begin{array}{c}
   T_i \text{ Sat } S_i, \ i = 1,n, \ G_C(S_1,\ldots,S_n,S) \\
   \hline
   C(T_1,\ldots,T_n) \text{ Sat } S \\
   \end{array}$$

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 $\mathcal{A} \mathcal{R}$, denoted by $\vdash_{\mathcal{A} \mathcal{R}} \phi$, if $\phi$ can be derived from axioms in $\mathcal{A}$ by applications of rules in $\mathcal{R}$.

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

**Definition:** A proof system $\mathcal{A} \mathcal{R}$ is relatively complete \footnote{We assume, as usual, that all domain properties that are true can be used as additional axioms \cite{Apt81}, and that all intermediate assertions needed for the proof can be expressed in $\mathcal{S}$.} w.r.t. a structure $M$ if for each formula $\phi \in \Phi : M \models \phi$ implies that $\vdash_{\mathcal{A} \mathcal{R}} \phi$. 

$BEH_S[S_1] \cap BEH_S[S_2]$. A specification $S$ is a tautology if $BEH_S[S] = B$. \(\square\)
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 unifyable, i.e., there exists a substitution \( \theta \) which binds meta variables to sub-programs, such that applying \( \theta \) 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 \( \theta \) 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 \( \gamma \), if \( T_1 \gamma = T_2 \gamma \) then there exists a substitution \( \lambda \) such that \( \theta \circ \lambda = \gamma \), where \( \circ \) 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. \( \mathcal{O} \subseteq \mathcal{T}^\# \), \( \mathcal{V} \subseteq \mathcal{T}^\# \) and if \( C \in \mathcal{C}^\# \), \( T_1, \ldots, T_n \in \mathcal{T}^\# \) then \( C(T_1, \ldots, T_n) \in \mathcal{T}^\# \). \( ^2 \)

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 "\( TSat^\#S \)", 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 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, \( \bar{S} \), which contains exactly all the specifications of \( S \), 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).

---

2Since 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.
The new set of specifications, $S^*$, is a set of mixed-specifications. A mixed-specification contains two parts separated by a “$\triangleright$” sign. The expression on the right-side of the “$\triangleright$” sign is a basic specification, considered as the conclusion of the specification. The expression on the left-side of the “$\triangleright$” sign is syntactically defined as a member of an algebra with a set $C^*$ 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 $T$. For example, the gapped-program $T_1 :: C_1(C_2(a_1, a_2), X)$ satisfies the mixed-specification $C_1(S_1, S_2) \triangleright S_3$ iff $C_2(a_1, a_2)$ satisfies $S_1$, $X$ is assumed to satisfy $S_2$ and moreover $T_1$ satisfies $S_3$ under the assumption that $X$ satisfies $S_2$.

Next, we exemplify how mixed-specifications enable a compositional verification of a jigsaw-composed program. Let $T_1$ be the above program and $T_2 :: C_1(Y, a_3)$. Assume the specification $C_1(S_4, S_5) \triangleright S_6$ has been verified for $T_1$ and the specification $C_1(S_4, S_5) \triangleright S_6$ 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 $S_4$ and $S_5$ 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 $S_4$, $S_1$ confirms $S_4$, then it is possible to conclude that $#(T_1, T_2)$ satisfies $S_6$. In a similar way, if $S_5$ confirms $S_2$ then it is possible to conclude that $#(T_1, T_2)$ satisfies $S_3$.

Next, we formally define the new specification language.

Definition:

1. The set $\mathcal{S} = \{ S \mid S \in \mathcal{S} \}$ is called the assumption set. The set $\mathcal{S}$ is closed under conjunction, and we assume $\mathcal{S}_1 \land \mathcal{S}_2 = \mathcal{S}_1 \land \mathcal{S}_2$.

2. The set of mixed-specifications, $S^*$, is defined by the following grammar:

$$S^* ::= \mathcal{S} \mid C(S^*, \ldots, S^*) \triangleright S \mid #(S^*, S^*)$$

where $C \in C$.

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

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

We assume that the basic system provides a way to determine whether $S_1 \Rightarrow S_2$ holds. It is possible to redefine $S_1 \Rightarrow S_2$ to hold iff $BEH_S[S_1] = BEH_S[S]$. Using inclusion eases the use of the proof system. However, in some concrete systems there might not be a way to determine inclusion while equality can be determined.

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

1. if $S \in S$ then $conc(S) = S$ and $promise(S)$ is not defined,
2. if $S = \overline{S} \in \mathcal{S}$ then $\text{conc}(S) = \overline{S}$ and $\text{promise}(S)$ is not defined,

3. if $S = C(S_1, \ldots, S_n) \triangleright S'$ then $\text{conc}(S) = S'$ and $\text{promise}(S) = C(S_1, \ldots, S_n)$,

4. if $S = \#(S_1, S_2)$ then $\text{conc}(S) = \text{conc}(S_1) \land \text{conc}(S_2)$ and $\text{promise}(S) = \#(\text{promise}(S_1), \text{promise}(S_2))$. □

To define the semantics of mixed-specifications, a function $BEH_s^\# : S^\# \to \wp(\mathcal{B})$ is presented. Intuitively, if $S'$ is a "consequence" of $C(S_1, \ldots, S_n)$, i.e., $K_C(BEH_s^\#[S_1], \ldots, BEH_s^\#[S_n]) \subseteq BEH_s^\#[S']$, then $BEH_s^\#[C(S_1, \ldots, S_n) \triangleright S']$ is defined as $BEH_s^\#[S']$. Else, it is defined as $\Theta$. If the structures of $S_1$ and $S_2$ are unifiable and their assumptions are mutually confirmed then $BEH_s^\#[\#(S_1, S_2)]$ is defined as $BEH_s^\# \cap BEH_s^\#[S_2]$. Else, it is defined as $\Theta$. 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, $SK$, is defined by:

$$SK :: S \mid \mathcal{V} \mid \mathcal{V} \land \mathcal{V} \mid C(SK_1, \ldots, SK_n) \triangleright \mathcal{V} \mid \#(SK, SK),$$

where $C \in \mathcal{C}$.

Definition: Let $S \in S^\#$ 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 $\theta$ which binds basic specifications or assumptions to variables. $skel(S)$ is defined as follows:

1. if $S \in S$ then $skel(S) = < S, \varepsilon >$,

2. if $S = \overline{S} \in \mathcal{S}$ then $skel(\overline{S}) = < X, \{X \leftarrow \overline{S} \} >$, where $X \in \mathcal{V}$,

3. $skel(C(S_1, \ldots, S_n) \triangleright S) = < C(skel_1(S_1), \ldots, skel_1(S_n)) \triangleright X, skel_2(S_1) \cup \ldots \cup skel_2(S_n) \cup \{X \leftarrow S \} >$,

4. $skel(\#(S_1, S_2)) = < \#(skel_1(S_1), skel_1(S_2)), skel_2(S_1) \cup skel_2(S_2) >$.

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 $\text{conc}(S_i) \Rightarrow \text{conc}(S'_j)$ holds. □

Next, a transformation on mixed-specifications, $\text{elim} : S^\# \to S^\#$, 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 $\text{elim}$ is defined for $\#(S_1, S_2)$ iff $S_1$ and $S_2$ are unifiable (see Appendix A) and their assumption are mutually confirmed. Intuitively, the function $\text{elim}$ 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 $S \in S^\#$ be a mixed-specification. Let $\text{elim}(S)$ be:

For simplicity assume that $skel_i(S_i), i = 1, n$ have disjoint sets of variables and $X$ is a fresh variable.
1. if \( S \in \mathcal{S} \cup \mathcal{S} \) then \( \text{elim}(S) = S \),

2. \( \text{elim}(C(S_1, \ldots, S_n) \triangleright S') \) is defined to be \( C(\text{elim}(S_1), \ldots, \text{elim}(S_n)) \triangleright S' \), if for every \( 1 \leq i \leq n \), \( \text{elim}(S_i) \) is defined, and otherwise it is not defined.

3. \( \text{elim}(\#(S_1, S_2)) \) is defined to be \( \#(\text{elim}(S_1), \text{elim}(S_2)) = \text{skel}_1(\text{elim}(S_1))\theta \), if \( \text{skel}_1(\text{elim}(S_i)), i = 1, 2 \) are defined and are unifiable \(^4\) and moreover \( \theta \) is a valid substitution both w.r.t. \( \text{skel}_2(S_1) \) and w.r.t. \( \text{skel}_2(S_2) \), where \( \theta = (\text{mgu}^\ast(\text{skel}_1(S_1), \text{skel}_1(S_2)) \circ \text{skel}_2(S_1) \circ \text{skel}_2(S_2)) \). \(^5\) Otherwise, it is not defined.

If \( \text{elim}(\#(S_1, S_2)) \) is defined, \( \#(S_1, S_2) \) is called \textit{well-defined}.

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

1. if \( S \in \mathcal{S} \) then \( \text{BEH}^\#(S) = \text{BEH}_S[S] \),

2. if \( \mathcal{S} \in \mathcal{S}^\# \) then \( \text{BEH}^\#(\mathcal{S}) = \text{BEH}_S[S] \),

3. if \( S = C(S_1, \ldots, S_n) \triangleright S' \) and \( C \neq \#' \) then \( \text{BEH}^\#[C(S_1, \ldots, S_n) \triangleright S'] = \text{BEH}^\#[S'] \), if \( K_C(\text{BEH}^\#[S_1], \ldots, \text{BEH}^\#(S_n)) \subseteq \text{BEH}^\#[S'] \) and the set \( \Theta \), otherwise.

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

The meaning of a gapped program \( T \) in \textit{context} \( S \) is defined by a semantic function \( \text{BEH}_{T-S}^\#: T^\# \times \mathcal{S}^\# \rightarrow \wp(\mathcal{B}) \), to be denoted by \( \text{BEH}_{T-S}^\#[T \downarrow S] \). The meaning of \( T \) in context \( S \) is defined iff \( T \) and \( S \) are \textit{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 \( T \in T^\# \) and \( S \in \mathcal{S}^\# \). \( T \) and \( S \) are \textit{identically structured} iff

1. \( T \in T \) and \( S \in S \), or

\(^4\) \( \text{mgu}^\ast \) 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 \( X \leftarrow X \land Y \), \( Y \leftarrow X \land Y \). 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, \( \text{elim}(\#(S_1, S_2)) = \text{elim}(\#(S_2, S_1)) \).
2. $T \in \mathcal{V}$ and $S \in \mathcal{S}$, or

3. $T = C(T_1, \ldots, T_n)$, $S = C(S_1, \ldots, S_n) \triangleright S$ 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, $i = 1, 2$.

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

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

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

3. $\text{elim}(\#(T_1, T_2)) = \#(\text{elim}(T_1), \text{elim}(T_2)) \equiv \text{elim}(T_1) \text{mgu}(\text{elim}(T_1), \text{elim}(T_2))$, if $\text{elim}(T_1), i = 1, 2$ 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 $BEH^\#_{-S}$. 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 $\text{elim}(T)$ and $\text{elim}(S)$ are defined and moreover $\text{elim}(T)$ and $\text{elim}(S)$ are identically structured. The meaning of the term $T$ in the context of $S$ is next given by the function $BEH^\#_{-S} : T \times S \rightarrow \mathcal{P}(\mathcal{B})$:

1. if $\text{elim}(T) \in T$ then $BEH^\#_{T-S}[T \downarrow S] = BEH^\#_{T}[\text{elim}(T)]$,

2. if $\text{elim}(T) \in \mathcal{V}$ then $BEH^\#_{T-S}[T \downarrow S] = BEH^\#_{S}[S]$,

3. if $\text{elim}(T) = C(T_1, \ldots, T_n)$ and $\text{elim}(S) = C(S_1, \ldots, S_n) \triangleright S'$ then $BEH^\#_{T-S}[T \downarrow S] = KC(BEH^\#_{T-S}[T_1 \downarrow S_1], \ldots, BEH^\#_{T-S}[T_n \downarrow S_n])$. □

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 $S' = S_i$, $1 \leq i \leq n$,

   (b) $T'' = \#(T_1, T_2)$, $S'' = \#(S_1, S_2)$, $T' = T_i$ and $S' = S_i$, $1 \leq i \leq 2$.

Definition: Let $T \in \mathcal{T}^\#$ and $S \in \mathcal{S}^\#$. The formula "$T \text{ Sat}^\# S''$ holds in the structure $< B, BEH^\#_{T-S}, BEH^\#_{S}>$ iff $BEH^\#_{T-S}[T \downarrow S]$ 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 $BEH^\#_{T-S}[T' \downarrow S'] \subseteq BEH^\#_{S}[S']$. □
3.3 The proof system

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

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

1. The set $G$ is the basic set of criteria,
2. The set $A^*$ is an extension of $A$ with the following formulas:
   - The variable-axioms $\forall X Sat^# S$, where $X \in V$ and $S \in S$.
3. The set $\mathcal{R}^*$ is defined as the basic set of rules, $\mathcal{R}$, except that:
   - (a) The composition rules: each rule $R_c$ in $\mathcal{R}$ of the form $\forall T_i Sat S_j$, $i = 1, n, G_c(S_1, \ldots, S_n, S) \Rightarrow C(T_1, \ldots, T_n) Sat S$
     is transformed into $\forall T_i Sat^# S_j$, $i = 1, n, G_c(conc(S_1), \ldots, conc(S_n), S) \Rightarrow C(T_1, \ldots, T_n) Sat^# C(S_1, \ldots, S_n) \triangleright S$
   - (b) Three additional rules:
     i. The jigsaw rule
        $\forall T_i Sat^# S_i$, $i = 1, 2 \Rightarrow #(T_1, T_2) Sat^# #(S_1, S_2)$
        Provided that #(S_1, S_2) is well-defined
     ii. The substitution rule
        $T Sat^# S \Rightarrow T Sat^# S[skel_1(S_1)\theta / #(S_1, S_2)]$
        For every subterm #(S_1, S_2) in $S$, where
        $\theta = (mgv^*(skel_1(S_1), skel_1(S_2)) o skel_2(S_1) o skel_2(S_2))$
     iii. The assumption-freeness rule
        $T Sat^# M \triangleright S \Rightarrow T Sat^# S[S/M, S_1]$ for every subterm $M \triangleright S_1$ in $S$, where $M$ is assumption-free.

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

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

3. The composition rules 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.

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. $\square$

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 semantically true specification of $T$ can be verified for $T'$.

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

1. If $S_i \Rightarrow S_j$ 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 $G_C(S_1, \ldots, S_i, \ldots, S_k, S' \land S'')$ holds.

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

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 $T$ of programs is defined over the following sets of primitive objects, $O$, and constructors, $C$:

$$O = \{ \text{skip} \} \cup \{ x := e \} \cup \{ b \} \cup \{ D!e \} \cup \{ D?x \}$$

$$C = \{ ||, ;, \Box, \text{While} \}$$

The skip statement and the multiple assignment statement $x := e$ have their usual meanings. The statement $b$ (a boolean condition over $x$) 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 $(P_1 || P_2)$ 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 $(\text{While } b, P)$ 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 $B$ is next defined.

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

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 $(s_1, \tau, s_2)$, where $\tau$ is a trace, $s_1$ is a state and $s_2$ is a state or a special symbol $. $ $\Box$

A triple $(s_1, \tau, .)$ indicates an unfinished computation, that is, a computation

\[ \text{For clarity we use infix notation.} \]
which started in state $s_1$ and has already performed the communications in $\tau$, but has not yet terminated.

The set $B$ is defined as the set of all state-trace-state triples.

Definition: A set of state-trace-state $U$ is prefix closed iff for all triples $(s_1, \tau_1, s_2), (s_3, \tau_2, s_4)$: if $(s_3, \tau_2, s_4) \in U$ and $(s_1, \tau_1, s_2) \leq (s_3, \tau_2, s_4)$ then $(s_1, \tau_1, s_2) \in U$, where $(s_1, \tau_1, s_2) \leq (s_3, \tau_2, s_4)$ iff $(s_1 = s_3 \land \tau_1 = \tau_2 \land s_2 = s_4)$ or $(s_1 = s_3 \land \tau_1 \leq \tau_2 \land s_2 = \bot)$. □

We denote by $\Theta$ the set $\{(s, <>, \bot) | s \in \text{States}\}$.

Definition: The set $\overline{\varphi(B)}$ is the set of all prefix closed sets of state-trace-state triples which contain the set $\Theta$. □

Lemma 4.1: The set $\overline{\varphi(B)}$ with the inclusion order $\subseteq$ is a cpo (complete partial order) with a minimal element $\Theta$. □

Proof:

1. The inclusion order on sets is a partial order (reflexive, anti-symmetric, transitive).

2. By definition the minimal element $\Theta$ is included in every element of $\overline{\varphi(B)}$.

3. Every chain $X = B_1 \subseteq B_2 \subseteq \ldots$ has a lub (least upper bound) in $\overline{\varphi(B)}$ which is equal to the union of all elements in the chain $X$. □

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

The semantics of the programs in $T$ is next given by defining a semantic function $\text{BEH}_T : T \rightarrow \overline{\varphi(B)}$. Let the variables of a program range over the set $\text{Val}$ of values.

First, $\text{BEH}_T$ is defined for all elements in $\mathcal{O}$:

$$\begin{align*}
\text{BEH}_T(\text{skip}) & = \text{close} \{(s, <>, s) \} | s \in \text{State} \land s \neq \bot \\
\text{BEH}_T(x := e) & = \text{close} \{(s, <>, s[[e]/x]) \} | s \in \text{State} \land s \neq \bot \\
\text{BEH}_T(b) & = \text{close} \{ \text{if } [b]s \text{ then } (s, <>, s \text{ else } (s, <>, \bot)) \} | s \in \text{State} \land s \neq \bot \\
\text{BEH}_T(D!e) & = \text{close} \{(s, (D, [e]s), s) \} | s \in \text{State} \land s \neq \bot \\
\text{BEH}_T(D?x) & = \text{close} \{(s, (D, v), s[v/x]) \} | v \in \text{Val}, s \in \text{State} \land s \neq \bot
\end{align*}$$

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

1. Let $B_1, B_2 \in \overline{\varphi(B)}$.

2. Let $\text{chan}(B_i)$ stands for all channels which appear in the traces of $B_i$.

3. Let $\text{chan}(B_1, B_2) = \text{chan}(B_1) \cup \text{chan}(B_2)$.

4. Let $\tau|_{\text{chan}(B_i)}$ stands for the projection of the trace $\tau$ on the set of channels $\text{chan}(B_i)$.

5. Let $\text{var}(B_i)$ be the set of variables for which the states in $B_i$ assign values.
4.3 The specification

The set $S$ 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 "$T \text{ Sat } [I,p,q]$" 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**
  - $e :: 0 | 1 | \omega | v | x | \text{val}(\text{texp}[e]) | \text{texp} | e_1 + e_2$

- **channel - name**
  - $c :: D | \text{chan}(\text{texp}[e])$

- **trace**
  - $\text{texp} :: t [\pi_{\text{cset}}] (\text{texp}[\text{cset}])$

- **formula**
  - $p :: e_1 = e_2 | c_1 = c_2 | c_1 \land c_2 | \neg p | \exists v. [p] | \exists t. [p]$
Comments: \( \omega \) is a special value of program variables, standing for "undefined". \( \text{val}(texp[e]) \) is the value of the \( e \)-th communication in trace \( texp \). \( |texp| \) is the length of the trace \( texp \). \( \text{chan}(texp[e]) \) is the channel-name of the \( e \)-th communication in the trace \( texp \). \( \pi \) is a special auxiliary trace variable which must occur free in a formula. \( \pi \) denotes the communication history of the execution up to the current state. Moreover, only a projection of \( \pi \) on a set of channels \( \text{cset} \), to be denoted \( \pi|_{\text{cset}} \), can occur in a formula. \( (texp|_{\text{cset}}) \) is the projection of trace \( texp \) on a set of channels, \( \text{cset} \). Auxiliary variables, except for \( \pi \), that occur free in formulas are interpreted as if they were universally quantified.

We will freely use abbreviations such as \( texp_1 = texp_2 \) or \( p \rightarrow q \), defined as usual.

### 4.4 The semantics of the specifications

An assertion \( p \) is interpreted in a logical variables environment \( \gamma \) and a state-trace-state triple \( (s_1, \tau, s_2) \), where \( s_2 \neq \bot \). We write \( \langle p \rangle \gamma(s_1, \tau, s_2) \) for the following interpretation:

1. Free occurrences of logical 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 \( \pi|_{\text{cset}} \) is taken to be \( (\tau|_{\text{cset}}) \).

An assertion \( p \) is called valid denoted by \( \models p \) if \( \forall \gamma \forall (s_1, \tau, s_2).[p] \gamma(s_1, \tau, s_2) \) holds. The semantics of the specifications is defined by the semantic function \( \text{BEH}_S : S \rightarrow \wp(B) \):

\[
\text{BEH}_S[I, p, q] = \{ (s_1, \tau_1, s_2) | \forall \gamma \forall (s_0, \tau_0, s_1).[[p] \gamma(s_0, \tau_0, s_1) \text{ implies } [\forall \gamma \forall (s_0, \tau_0, s_1).[[p] \gamma(s_0, \tau_0, s_1) \text{ implies } [q] \gamma(s_0, \tau_0, s_1)] (s_1, \tau_1, s_2) \} \}
\]

A formula of the form \( T \models S \) is defined to hold if \( \text{BEHT}[T] \subseteq \text{BEHS}[S] \). Intuitively, a program \( T \) satisfies \( [I, p, q] \) if given an initial state-trace-state, \( (s_0, \tau_0, s_1) \), 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 \( \tau_1 \) 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 \( T \in T \) that satisfies this specification. \( \square \)

**Proof:** First we prove the following Claim.

**Claim:** Let \( (s_0, \tau_0, s_1) \) and \( \gamma \) be such that \( [p] \wedge \neg I] \gamma(s_0, \tau_0, s_1) \) then \( (s_1, \triangleleft, \bot) \notin \text{BEHS}[I, p, q] \).

**Proof of claim** Assume \( (s_1, \triangleleft, \bot) \in \text{BEHS}[I, p, q] \). According to the definition of \( \text{BEHS} \) if \( \langle p \rangle \gamma(s_0, \tau_0, s_1) \) holds and \( (s_1, \triangleleft, \bot) \in \text{BEHS}[I, p, q] \) then \( [I] \gamma(s_0, \tau_0 \triangleleft, \bot) \) holds. Therefore, \( [I] \gamma(s_0, \tau_0 \triangleleft, \bot) \) holds (since \( I \) is interpreted only in \( \tau_0 \)), a contradiction. \( \square \)
Based on the above Claim if $p \rightarrow I$ is not valid then $BEH_S[[I, p, q]]$ does not contain the set $\Theta$ and therefore there does not exist a program which satisfies this specification (since the meaning of every program contains $\Theta$). □

Conclusion: We can redefine the set of specifications $S$ 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 $BEH_S[[I, p, q]] \in \mathcal{B}$. □

Proof:

1. Prefix-closed: Let $(s_1, \tau_1, s_2) \in BEH_S[[I, p, q]]$ and let $(s_1, \tau_1', s_2') \leq (s_1, \tau_1, s_2)$ then for every $\gamma$ and for every $(s_0, \tau_0, s_1)$ 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 $[I]^{\gamma}(s_0, \tau_0 \tau_1'', s_2'') \wedge (s_2'' \neq \bot$ implies $[q]^{\gamma}(s_0, \tau_0 \tau_1'', s'')$).

2. Includes $\Theta$: Let $s_1 \in State$. Next we show that $(s_1, <>, \bot) \in BEH_S[[I, p, q]]$.

For every $\gamma$ and for every $(s_0, \tau_1, s_1)$ 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 $[I]^{\gamma}(s_0, \tau_0, \bot)$ holds. We can conclude the above since $I$ is interpreted only in $\tau_0$. □

Conclusion: For every mixed-specification $S \in S^*$ the function $BEH^*_S$ (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].

<table>
<thead>
<tr>
<th>Expression</th>
<th>Sat</th>
<th>Expression</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skip</td>
<td>$skip$</td>
<td>$[I, p \land I, p \land I]$</td>
<td>Assignment</td>
</tr>
<tr>
<td>Guard</td>
<td>$b$</td>
<td>$[I, p \land I, p \land I]$</td>
<td>Output</td>
</tr>
<tr>
<td>Input</td>
<td>$D?e$</td>
<td>$[I, I[p_D &lt; D, e &gt; /p_D] \land q[p_D &lt; D, e &gt; /p_D], I \land q]$</td>
<td></td>
</tr>
</tbody>
</table>

For every $v$.

Sequential-composition

\[ T_1 Sat S_1, \; T_2 Sat S_2, \; G_i(S_1, S_2, S) \]

\[ (T_1; T_2) Sat S \]

Where, $S_i = [I_i, p_i, q_i], \; i = 1, 2, \; S = [I, p, q]$ and $G_i(S_1, S_2, S)$ holds iff $(I_1 \rightarrow I) \land (I_2 \rightarrow I) \land (p \rightarrow p_1) \land (q_1 \rightarrow q_2)$. 
Choice-composition
\[
T_1 \text{ Sat } S_1, \ T_2 \text{ Sat } S_2, \ G_{\cap}(S_1, S_2, S)
\]

\[(T_1 \cap T_2) \text{ Sat } S\]

Where, \(S_i = [I, p_i, q_i], \ i = 1, 2, \ S = [I, p, q]\) and \(G_{\cap}(S_1, S_2, S)\) holds iff \((I_1 \rightarrow I) \land (I_2 \rightarrow I) \land (p \rightarrow (p_1 \land p_2)) \land ((q_1 \lor q_2) \rightarrow q)\).

Parallel-composition
\[
T_1 \text{ Sat } S_1, \ T_2 \text{ Sat } S_2, \ G_{||}(S_1, S_2, S)
\]

\[(T_1 \parallel T_2) \text{ Sat } S\]

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

While-composition
\[
T_1 \text{ Sat } S_1, \ b \text{ Sat } S_2, \ G_{\text{While}}(S_1, S_2, S)
\]

\[(\text{While } b, T_1) \text{ Sat } S\]

Where \(S_i = [I, p_i, q_i], \ S = [I, p, q], \ S_2 = [true, b, b]\) and \(G_{\text{While}}(S_1, S_2, S)\) holds iff \((I_1 \rightarrow I) \land ((b \land p) \rightarrow p_1) \land ((q_1 \land b) \rightarrow p_1) \land ((q_1 \lor \neg b) \rightarrow q)\). □

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 \(S\) is closed under conjunction and that there exists a way to determine if a specification \(S_1\) confirms a specification \(S_2\), \((BEH_S[S_1] = BEH_S[S_2]\) in this case).

**Lemma 4.5:** \(BEH_S[[I_1, p_1, q_1]] = BEH_S[[I_2, p_2, q_2]]\) holds iff \((I_1 \leftrightarrow I_2) \land (p_1 \leftrightarrow p_2) \land (q_1 \leftrightarrow q_2)\) holds.

**Proof:** Trivial.

**Lemma 4.6:** The set \(S\) is closed under conjunction. □

**Proof:** In order to prove closedness under conjunction, we use a logical variable in the common way, to be formally defined next.

**Definition:** Let \(L\) be a logical variable\(^7\) which ranges over the set of values \(\text{Dom}\). Let \(S\) be a specification in which the variable \(L\) appears free.

\[
BEH_S[S] \overset{df}{=} \bigcup_{d \in \text{Dom}} BEH_S[S[d/L]],
\]

where \(\bigcup\) is a disjoint union of sets. □

**Definition:** Let \(B_i = \bigcup_{d \in \text{Dom}} B_{i,d}, \ i = 1, 2\). Define,
1. \( B_1 \preceq B_2 \) iff \( \forall b \in \text{Dom}_2 \exists a \in \text{Dom}_1 : B_{1,a} \subseteq B_{2,b} \).

2. \( B_1 \uplus B_2 \overset{d.f.}{=} \bigcup_{d_1,d_2 \in \text{Dom}_1 \times \text{Dom}_2} (B_{1,d_1} \cap B_{2,d_2}) \). □

Comment: For every set \( A, A = \{ a \in \text{Dom}_a \} \) holds, where \( \text{Dom}_a = A \).

Definition: For a program \( T \) and a specification \( S \) we redefine “a formula holds in a structure” as: \( \mathcal{M} \models T \) Sat \( S \) iff \( \text{BEH}_T[T] \subseteq \text{BEH}_S[S] \), where \( \text{BEH}_T[T] = \bigcup_{a \in \text{Dom}_a} \text{BEH}[T]_a \) and \( \text{BEH}_T[T]_a = \text{BEH}_T[T] \). □

Definition: Let \( B_1, B_2 \) be two sets, define \( B_1 \uplus B_2 \) iff \( B_1 \not\subseteq B_2 \) and \( B_2 \not\subseteq B_1 \). □

Next, we prove the closedness under conjunction. By definition, the meaning of the specification \( I_1, p_1, q_1 \land I_2, p_2, q_2 \) is: \( \text{BEH}_S[I_1, p_1, q_1 \land I_2, p_2, q_2] = \text{BEH}_S[I_1, p_1, q_1] \cap \text{BEH}_S[I_2, p_2, q_2] \). The above meaning is equal to the meaning of the specification \( I, p, q \) where

\[
I = (L = 1 \rightarrow (I_1 \land I_2)) \land (L = 2 \rightarrow I_1) \land (L = 3 \rightarrow I_2) \\
p = (p_1 \lor p_2) \land ((p_1 \land p_2) \rightarrow L = 1) \land ((p_1 \land \neg p_2) \rightarrow L = 2) \land ((\neg p_1 \land p_2) \rightarrow L = 3) \\
q = (L = 1 \rightarrow (q_1 \land q_2)) \land (L = 2 \rightarrow q_1) \land (L = 3 \rightarrow q_2)
\]

Where \( L \) is a logical variable, which does not appear in the assertions \( I, p, q \), and ranges over the values \( \{1, 2, 3\} \).

It can be seen that the set \( \text{BEH}_S[I, p, q] \) is defined as:

\[
B_1 \uplus B_2 \uplus B_3
\]

where,

\[
B_1 = \text{BEH}_S[I_1 \land I_2, p_1 \land p_2, q_1 \land q_2] \\
B_2 = \text{BEH}_S[I_1, p_1 \land \neg p_2, q_1] \\
B_3 = \text{BEH}_S[I_2, \neg p_1 \land p_2, q_2]
\]

First direction: Next we show that \( \text{BEH}_S[I_1, p_1, q_1 \land I_2, p_2, q_2] \subseteq \text{BEH}_S[I, p, q] \).

Assume, \((s_1, r_1, s_2) \in \text{BEH}_S[I_1, p_1, q_1] \cap \text{BEH}_S[I_2, p_2, q_2] \). Assume, \((s_1, r_1, s_2) \in \text{BEH}_S[I_1, p_1, q_1] \cap \text{BEH}_S[I_2, p_2, q_2] \).

Four cases:

1. Let \( \gamma \) and \( (s_0, r_0, s_1) \) be such that \( [p_1 \land p_2] \gamma(s_0, r_0, s_1) \) holds. Then for every \( (s_1, r_1, s_2) \leq (s_1, r_1, s_2), [I_1 \land I_2] \gamma(s_0, r_0 r_1, s_2') \) holds and if \( s_2' \neq \bot \) then \( [q_1 \land q_2] \gamma(s_0, r_0 r_1, s_2') \) holds. Therefore, checking every \( \gamma \) and \((s_0, r_0, s_1)\) as above we can conclude that \((s_1, r_1, s_2) \in B_i, i = 1, 3, \).

2. Let \( \gamma \) and \((s_0, r_0, s_1)\) be such that \( [\neg p_1 \land p_2] \gamma(s_0, r_0, s_1) \) holds. Then for every \((s_1, r_1, s_2) \leq (s_1, r_1, s_2), [I_2] \gamma(s_0, r_0 r_1, s_2') \) holds and if \( s_2' \neq \bot \) then \( [q_2] \gamma(s_0, r_0 r_1, s_2') \) holds. Therefore, checking every \( \gamma \) and every \((s_0, r_0, s_1)\) we can conclude that \((s_1, r_1, s_2) \in B_i, i = 1, 3, \).

3. The two other cases are similar and therefore not brought here.
Second direction: Next we show that $BEH_S[[I, p, q]] \subseteq BEH_S[[I, p_1, q_1] \land [I_2, p_2, q_2]]$. It is enough to show that there exists $a \in \{1, 2, 3\}$ such that $B_a \subseteq BEH_S[[I_1, p_1, q_1] \land [I_2, p_2, q_2]]$. Choose, $a = 1$, we have to show that $BEH_S[[I_1 \land I_2, p_1 \land p_2, q_1 \land q_2]] \subseteq BEH_S[[I_1, p_1, q_1] \land [I_2, p_2, q_2]]$.

Assume $(s_1, n_1, s_2) \in BEH_S[[I_1 \land I_2, p_1 \land p_2, q_1 \land q_2]]$, we have to consider here four cases as in the first direction, not brought here. □

Next we show that the predicates $G_{White}, G_{||}, G_i$, 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 $G_{White}, G_{||}, G_i, G'$ are monotone and compositional. □

Proof: Only the proof for $G_i$ is brought here.

1. Monotone:
   It can be concluded from Lemma 4.2.

2. Compositional:
   Assume $G_i([I_1, p_1, q_1], [I_2, p_2, q_2], [I', q']])$ and $G_i([I_1, p_1, q_1], [I_2, p_2, q_2], [I'', q''])$ hold. We know that $(I_1 \rightarrow (I' \land I'')) \land (I_2 \rightarrow (I' \land I'')) \land (p' \rightarrow p_1) \land (p' \rightarrow p_1) \land (q_1 \rightarrow p_2) \land (q_2 \rightarrow (q' \land q''))$. The following assertions should be justify:

   (a) $I_1 \rightarrow ((L = 1 \rightarrow (I' \land I'')) \land (L = 2 \rightarrow I') \land (L = 3 \rightarrow I''))$

   (b) $I_2 \rightarrow ((L = 1 \rightarrow (I' \land I'')) \land (L = 2 \rightarrow I') \land (L = 3 \rightarrow I''))$

   (c) $((p' \land q'') \land ((p' \land q'')) \land (L = 1) \land (p' \land q'') \land L = 2) \land ((p' \land q'') \land ((p' \land q'') \land L = 3)) \rightarrow p_1$

   (d) $(q_1 \rightarrow p_2)$

   (e) $q_2 \rightarrow ((L = 1 \rightarrow (q' \land q'')) \land (L = 2 \rightarrow q') \land (L = 3 \rightarrow q''))$

   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
The system is defined as \(#(\text{Communication}, \text{Primes})\), where \(\text{Communication} \equiv (\text{Producer} \ || \ \text{Consumer})\) and

\[\begin{align*}
\text{Producer} &::=
\text{Consumer} \\
(\text{While more} = 1, Y; \\
val := \text{prod}; \\
(\text{While val} \geq 63, \\
D_{12!}63; \text{val} := \text{val} - 63); \\
D_{12!}\text{val}; D_{21!}\text{more})
\end{align*}\]

\[\begin{align*}
\text{Consumer} &::=
\text{Producer} \\
(\text{While more} = 1, \\
D_{12!}\text{temp}; \text{new} := \text{temp}; \\
(\text{While temp} = 63, \\
D_{12!}\text{temp}; \text{new} := \text{new} + \text{temp}); \\
M; \\
D_{21!}\text{more})
\end{align*}\]

\[\begin{align*}
\text{Primes} &::= (\text{Producer} \ || \ \text{Consumer}), \text{where}
\text{Producer} &::=
\text{Consumer} \\
(\text{While more} = 1, \\
(\text{While found} = 0, \\
\text{prod}, j, \text{found} := \text{prod} + 1, 1, 1; \\
(\text{While } j \leq \text{max} \land \text{found} \neq 0, \\
\text{found}, j := \text{mod}(\text{prod}, \text{array}[j]), j + 1); \\
\text{max}, \text{array}[\text{max} + 1], \text{found} := \text{max} + 1, \text{prod}, 0; \\
Z)
\end{align*}\]

\[\begin{align*}
\text{Consumer} &::=
\text{Producer} \\
(\text{While more} = 1, \\
V; \\
(\text{new} \geq 100; \text{more} := 0 \\
\square) \\
\text{new} < 100; \text{skip}); \\
N)
\end{align*}\]

Figure 1: The Communication and the Primes subtasks

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 \((\text{Producer} \ || \ \text{Consumer})\). 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 then parallel compose them. However, the program 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 (Figure 1). Mixed specifications which can be proved for
Producer\textsuperscript{c} and Consumer\textsuperscript{c}, are:

\[
\begin{align*}
S_{\text{producer}} &:: (\text{While}[\text{true}, \text{more} = 1, \text{more} = 1], \text{[I, Ass}_{1}, \text{Ass}_{1}] ; \text{[true, Ass}_{1}, \text{true}] ) \triangleright [I, \text{Initial}, \text{true}] \\
S_{\text{consumer}} &:: (\text{While}[\text{true}, \text{more} = 1, \text{more} = 1], \text{([I, prime(new) \land more = 1]; \text{[true, more = 1, Ass}_{2}] ) \triangleright [I, \text{more} = 1, \text{true}] ) \triangleright [I, \text{Initial}, \text{true}] 
\end{align*}
\]

Mixed specifications which can be proved for Producer\textsuperscript{p} and Consumer\textsuperscript{p}, are:

\[
\begin{align*}
S_{\text{producer}} &:: \text{While}_{\text{true, more} = 1, \text{more} = 1}, \text{[I, Ass}_{1}, \text{Ass}_{1}] ; \text{[true, Ass}_{1}, \text{true}] ) \triangleright [I, \text{Initial, true}] \\
S_{\text{consumer}} &:: \text{While}_{\text{true, more} = 1, \text{more} = 1}, \text{([I, prime(new) \land more = 1]; \text{[true, more = 1, Ass}_{2}] ) \triangleright [I, \text{true}] \triangleright [I, \text{Initial, true}] 
\end{align*}
\]

Where \( I \) is defined as \( (\forall u. \text{val}(\pi D_{12}D_{21}[v]) \leq 63) \land (\forall u. \text{val}(\pi D_{12}[v]) \neq 63 \rightarrow \text{prime}(\Sigma j E A \pi D_{12}[j]) \) and \( A = \{v', v' + 1, \ldots, v\} \) where \( (\forall j \in A. j \neq v \rightarrow \text{val}(\pi D_{12}[j]) = 63) \land \text{val}(\pi D_{12}[v' - 1]) \neq 63. \)

\[
\begin{align*}
\text{Initial} &:: \text{found} = 0 \land \text{prod} = 2 \land \text{array}[1] = 2 \land \text{max} = 1 \\
\text{Ass}_{1} &:: \text{prod} \in \text{array}[1..\text{max}] \land (\forall l \leq l \leq \text{max. prime(array}[l])] \land (\forall k. \text{prime}(k) \land \\
\text{Ass}_{2} &:: \text{prime}(new) \land (\text{more} \neq 1 \rightarrow \text{new} \geq 100)
\end{align*}
\]

For the Communication subtask we can conclude using the rule for parallel composition that it satisfies: \( S_{\text{communication}} :: (S_{\text{producer}} \parallel S_{\text{consumer}}) \triangleright [I, \text{Initial, true}] \).

For the Primes subtask we can conclude using the rule for parallel composition that it satisfies: \( S_{\text{primes}} :: (S_{\text{producer}} \parallel S_{\text{consumer}}) \triangleright [I, \text{Initial, prime(new) \land new} \geq 100] \).

For the jigsaw composition \#(Communication, Primes) we can conclude by using the jigsaw rule, the substitution rule and the assumption freeness rule that it satisfies \( [I, \text{Initial, prime(new) \land new} \geq 100] \).

### 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

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 Binational Science Foundation, Jerusalem, Israel.

7 References


Lemma A.1: The pair \( \langle \mathcal{U}, \prec \rangle \) is a partial pre-order, i.e., reflexive and transitive.
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 $<_e$ on the equivalence classes of $R_u$ is defined as follows: Let $E_1, E_2$ be two equivalence classes of $R_u$ then $E_1 <_e E_2$ iff $\exists \gamma \in E_1 \exists \theta \in E_2 : \gamma <_u \theta$. $\Box$

Lemma A.3: $E_1 <_e E_1$ 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 $T_1, T_2 \in T^\#$ 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 $1 \leq i \leq n$. $\Box$

Definition: Programs $T_1, T_2 \in T^\#$ 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 a program with "overlapping" primitive objects is not legal. This restriction is not necessary however it simplifies the technical considerations.

Definition: Let $S_1, S_2 \in SK$ 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) $S_1'' = \#(S_1^{**}, S_2^{**})$, $S_2'' = \#(S_1^{**}, S_2^{**})$, $S_1' = S_1^*$ and $S_2' = S_2^*$, $1 \leq i \leq 2$. 

24
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 \( \mathcal{AR}^\# \), \( \vdash_{\mathcal{AR}^\#} T_i \ \text{Sat}^\# S_i \), \( i = 1, 2 \) and \( \#(S_1^*, S_2^*) \) is well-defined.

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

Proof of claim: Let denote \( \text{elim}(T_i^*) \) by \( T_1 \), \( \text{elim}(T_2^*) \) by \( T_2 \), \( \text{elim}(S_1^*) \) by \( S_1 \) and \( \text{elim}(S_2^*) \) 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,
\[
\text{elim}(\#(T_1^*, T_2^*)) = T_1 \text{mgu}(T_1, T_2) = T_2 \text{mgu}(T_1, T_2)
\]
\[
\text{elim}(\#(S_1^*, S_2^*)) = \text{skel}_1(S_1) \text{mgu}^*(\text{skel}_1(S_1), \text{skel}_1(S_2)) \text{oskel}_2(S_1) \text{oskel}_2(S_2) = \text{skel}_1(S_2) \text{mgu}^*(\text{skel}_1(S_1), \text{skel}_1(S_2)) \text{oskel}_2(S_1) \text{oskel}_2(S_2).
\]

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 \( \text{skel}_1(S_1) \) be denoted by \( W_1, \ldots, W_n \) and let the variables of \( \text{skel}_1(S_2) \) be denoted by \( V_1, \ldots, V_p \).
Let $mgv(T_1, T_2) = \theta_1 \subseteq \{X_1 \leftarrow T_1', \ldots, X_m \leftarrow T_m', Y_1 \leftarrow T_1'', \ldots, Y_q \leftarrow T_q''\}$ and

$mgu(\text{ske1}(S_1), \text{ske1}(S_2)) \circ \text{ske1}(S_1) \circ \text{ske1}(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: $\vdash_{AR^*} T_1 \theta_1 \text{Sat*} \text{ske1}(S_1) \theta_2$ and

$\text{conc}(\text{ske1}(S_i) \theta_2) \Rightarrow \text{conc}(S_i), \ i = 1, 2$.

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 $\theta_2 = \{W \leftarrow S_1 \land S_2, V \leftarrow S_1 \land S_2\}$. Therefore, $T_1 \theta_1 = Y$ and $\text{ske1}(S_1) \theta_2 = S_1 \land S_2$. Using the variable-axiom $\vdash_{AR^*} Y \text{Sat*} S_1 \land S_2$. Since $\theta_2$ is valid w.r.t. $\text{ske1}(S_1)$ and w.r.t. $\text{ske1}(S_1)$ we get $(S_1 \land S_2) \Rightarrow S_i, \ i = 1, 2$.

(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 $\text{ske1}(S_1) \theta_2 = S_2$. We know that $\vdash_{AR^*} T_2 \text{Sat*} S_2$ by the induc. hyp. $\vdash_{AR^*} \text{elim}(T_2) \text{Sat*} \text{elim}(S_2)$ which can be rewritten as $\vdash_{AR^*} T_2 \text{Sat*} S_2$. Moreover, $\text{conc}(\text{ske1}(S_1) \theta_2) = \text{conc}(S_2) = \text{conc}(S_1)$ since $\theta_2$ is valid w.r.t. $\text{ske1}(S_2)$.

(c) If $T_1 \in \mathcal{O}$ then we can conclude that $T_2 \notin \mathcal{V}$ 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 $S_1, S_1$ and $T_2$ are of the form $S_1 = C(S_1', \ldots, S_n'), \ T_1 = C(S_1'', \ldots, S_n'') \Rightarrow S'$, $S_2 = C(S_1', \ldots, S_n') \Rightarrow S''$ and $T_2 = C(T_1', \ldots, T_n')$.

Intuitively, this case is proved as follows:

i. We decompose $\theta_1$, the most general unifier of $T_1$ and $T_2$, into $n$ sub-substitutions such that $\theta_1^i$ 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 $\theta_2$ into $n+1$ sub-substitutions such that $\text{ske1}(S_i') \theta_2^i = \text{ske1}(S_i'') \theta_2^i$. Moreover, by definition $S_i' \theta_2^{n+1} = S_i'' \theta_2^{n+1} = S' \land S''$.

iii. By Lemma B.2 and according to the induction hyp. we show that $\vdash_{AR^*} T_1' \theta_1^i \text{Sat*} \text{ske1}(S_i') \theta_2^i, \ i = 1, n$.

iv. According to the monotonicity and compositionality of $G_C$ we show that $G_C(\text{conc}(\text{ske1}(S_i') \theta_1^i), \ldots, \text{conc}(\text{ske1}(S_n') \theta_1^n, S' \land S''))$ hold.

v. Then we compose the subprograms by applying $R_C$ and conclude that $\vdash_{AR^*} C(T_1' \theta_1^i, \ldots, T_n' \theta_1^i) \text{Sat*} C(\text{ske1}(S_1') \theta_2^i, \ldots, \text{ske1}(S_n') \theta_2^n) \Rightarrow S' \land S''$, which can be written as $\vdash_{AR^*} \text{elim}(\#(T_1', T_2') \text{Sat*} \text{elim}(\#(S_1', S_2')))$. Next we formally present the proof of this case.

It is possible to represent $\theta_1$ as a union of $n$ substitutions $\theta_1 = \{\theta_1^1, \ldots, \theta_1^n\}$ such that
Proof: By induction on the structure of $T'$. For every $T' \in T^*$ and $T \in T$ such that $T = \text{elim}(T')$ and for every $S \in S$ if $\vdash_{\text{AR}} T' Sat^# S$ then $\vdash T Sat^# S$. □

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

**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 $\vdash_{\text{AR}} T Sat^# S$ are also in $\text{AR}$ and thus $\vdash_{\text{AR}} T Sat^# S$ holds. By the soundness of $\text{AR}$ we get $\vdash T Sat^# S$.

2. If $T' = C(T_1', \ldots, T_n')$ then by the definition of $\text{elim}(T')$ we know that $\exists T_1, \ldots, T_n$ such that $T = C(T_1, \ldots, T_n)$ and $T_i = \text{elim}(T_i')$, $i = 1, n$. According to $\text{AR}^#$ there exist $S_i \in S^*$, $i = 1, n$ such that $\vdash_{\text{AR}^#} T_i Sat^# S_i$, $i = 1, n$ and $G_C(\text{conc}(S_1), \ldots, \text{conc}(S_n), S)$ holds. Using the $R_C$ rule $\vdash_{\text{AR}^#} C(T_1', \ldots, T_n') Sat^# C(S_1, \ldots, S_n) \Rightarrow S'$ which leads to $\vdash_{\text{AR}^#} C(T_1', \ldots, T_n') Sat^# S$ using a sequence, $l_1$, of rules.
For $1 \leq i \leq n$: if $S_i \in S$ then by the induc. hyp. $\vdash T_i \text{ Sat } S_i$ and by the completeness of $\mathcal{AR}$, $\vdash_{\mathcal{AR}} T_i \text{ Sat } S_i$. If $S_i \not\in S$ then $\vdash_{\mathcal{AR}} T'_i \text{ Sat}^# S_i$ which leads to $\vdash_{\mathcal{AR}} T'_i \text{ Sat}^# \text{ elim}(S_i)$ using Sub rules. We know that $\text{elim}(T'_i) = T_i \in T$ therefore $\text{elim}(S_i)$ is assumption-free (Lemma B.1). By using the Assumption-free rule we get: $\vdash_{\mathcal{AR}} T'_i \text{ Sat}^# \text{ conc(elim}(S_i)), i = 1, n$ which implies that $\vdash_{\mathcal{AR}} T'_i \text{ Sat}^# \text{ conc}(S_i)$, $i = 1, n$. Using Lemma B.3 $\vdash_{\mathcal{AR}} T_i \text{ Sat}^# \text{ conc}(S_i)$, $i = 1, n$. By the induction hyp. $\vdash T_i \text{ Sat conc}(S_i)$, $i = 1, n$. By the completeness of the basic system $\vdash_{\mathcal{AR}} T_i \text{ Sat conc}(S_i)$, $i = 1, n$ and therefore we can prove in $\mathcal{AR}$ that $\vdash_{\mathcal{AR}} T \text{ Sat } S$ and by the soundness of $\mathcal{AR}$ we can conclude that $\vdash T \text{ Sat } S$.

3. If $T' = \#(T'_1, T'_2)$ then by Lemma B.3 $\vdash_{\mathcal{AR}} \text{ elim}(T') \text{ Sat}^# S$. By the induction hyp. $\vdash \text{ elim}(T') \text{ Sat } S$. □

B.2 Completeness

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

1. $\vdash_{\mathcal{AR}} T \text{ Sat}^# S'$
2. $\text{conc}(S) = \text{conc}(S')$
3. $T$ and $S'$ are identically structured
4. for all matching subterms $T^* \in T$ and $S^* \in S^*$ w.r.t. $T$ and $S'$, $\vdash_{\mathcal{AR}} T^* \text{ Sat}^# S^*$, holds. □

Proof: By induction on the structure of $T$,

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

\begin{align*}
(*) \quad & \vdash_{\mathcal{AR}} C(T_1, \ldots, T_n) \text{ Sat}^# C(S_1, \ldots, S_n) \triangleright S_{n+1}. \text{ To derive (*) in } \mathcal{AR}^*
\end{align*}

the RC rule must have been applied at least once such that $\exists S'_1, \ldots, S'_{n+1}:$

\begin{align*}
& \vdash_{\mathcal{AR}} T_i \text{ Sat}^# S'_i, \quad i = 1, n, \quad G_C(\text{conc}(S'_1), \ldots, \text{conc}(S'_n), S'_{n+1}) \text{ holds and}
\end{align*}

\begin{align*}
& \vdash_{\mathcal{AR}} C(T_1, \ldots, T_n) \text{ Sat}^# C(S'_1, \ldots, S'_n) \triangleright S'_{n+1}.
\end{align*}

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. $\vdash_{\mathcal{AR}} T_i \text{ Sat}^# S''_i$
The above implies that:

ii. $\text{conc}(S_i') = \text{conc}(S_i'')$

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 $S_i''$, $\vdash_{AR^*} T^* \text{ Sat}^# S^*$, holds.

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

i. $\vdash_{AR^*} C(T_1, \ldots, T_n) \text{ Sat}^# C(S_1'', \ldots, S_n'') \triangleright S_{n+1}$,

ii. $\text{conc}(C(S_1, \ldots, S_n) \triangleright S_{n+1}) = \text{conc}(C(S_1'', \ldots, S_n'') \triangleright S_{n+1})$,

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 $S^* \in C(S_1'', \ldots, S_n'') \triangleright S_{n+1}$, $\vdash_{AR^*} T^* \text{ Sat}^# S^*$, holds.

(b) $S \in S$. In this case, $T \in T$ and we take $S' = S$. □

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

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

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

3. $\text{conc}(S) = \text{conc}(S_1) = \text{conc}(S_2)$.

Proof: By Lemma B.5, $\exists S'$ such that:

1. $\vdash_{AR^*} \#(T_1, T_2) \text{ Sat}^# S'$,

2. $\text{conc}(S) = \text{conc}(S')$,

3. $\#(T_1, T_2)$ and $S'$ are identically structured,

4. for all matching subterms $T^* \in \#(T_1, T_2)$ and $S^* \in S'$ w.r.t. $\#(T_1, T_2)$ and $S'$, $\vdash_{AR^*} T^* \text{ Sat}^# S^*$, holds.

The above implies that:

1. $S' = \#(S_1', S_2')$,

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

3. $(S_1', S_2')$ is well-defined,

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

5. $\text{conc}(S) = \text{conc}(S_1') \land \text{conc}(S_2')$.

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

According to the definition of $S_1''$ and $S_2''$ the following holds:

1. $\#(S_1'', S_2'')$ is well-defined,
2. $\text{conc}(S_1') = \text{conc}(S_2') = \text{conc}(S_1') \land \text{conc}(S_2') = \text{conc}(S)$. \hfill $\square$

**Lemma B.7**: Let $T \in T^*$ and $S \in S^*$. If $\vdash_{AR^*} \text{elim}(T) \text{ Sat}^* S$ then there exists $S' \in S^*$ such that $\vdash_{AR^*} T \text{ Sat}^* S'$ and $\text{conc}(S) = \text{conc}(S')$. \hfill $\square$

**Proof**: By induction on the structure of $T$.

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

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

(a) $S = C(S_1', \ldots, S_n') \triangleright S_{n+1}$. We are given that

(*) $\vdash_{AR^*} C(\text{elim}(T_1), \ldots, \text{elim}(T_n)) \text{ Sat}^* C(S_1', \ldots, S_n) \triangleright S_{n+1}$. To derive (*) in $AR^*$ the $R_G$ rule has been applied at least once such that $\exists S_1', \ldots, S_{n+1}':$

$\vdash_{AR^*} \text{elim}(T_i) \text{ Sat}^* S_i', \ i = 1, n, \ G_C(\text{conc}(S_1'), \ldots, \text{conc}(S_n'), S_{n+1}')$ holds and therefore (**) $\vdash_{AR^*} C(\text{elim}(T_1), \ldots, \text{elim}(T_n)) \text{ Sat}^* C(S_1', \ldots, S_n') \triangleright S_{n+1}$. 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 $\vdash_{AR^*} T_i \text{ Sat}^* S_i''$ where $\text{conc}(S_i'') = \text{conc}(S_i')$, $i = 1, n$ therefore we can conclude:

(***) \hfill $\vdash_{AR^*} C(T_1, \ldots, T_n) \text{ Sat}^* C(S_1', \ldots, S_n') \triangleright S_{n+1}$.

(b) $S \in S$. We are given that

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

B.5 and Lemma B.6 there exist specifications:

1. \( \forall \alpha \in \mathcal{P} \) s.t. \( \alpha \in \mathcal{P} \) \( \Rightarrow \) \( \alpha \in \mathcal{P} \) \( \Rightarrow \)

2. \( \exists \mathcal{P} \) such that \( \mathcal{P} \) \( \Rightarrow \) \( \mathcal{P} \) \( \Rightarrow \)

3. if \( T = \#(T_1, T_2) \). We are given that \( \vdash_{\mathcal{A} \mathcal{R}} \) elim(\( \#(T_1, T_2) \)) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \)

We bring here only the third case. If elim(\( \#(T_1, T_2) \)) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) then there are two cases: 1) \( S = C(S_1, \ldots, S_n) \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \), 2) \( S \in \mathcal{P} \). We bring here only the first case. Again there are two cases:

(a) elim(\( T_1 \)) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \)

(b) elim(\( T_2 \)) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \)

We bring here the proof of case (b). We know that elim(\( \#(T_1, T_2) \)) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) which means that elim(\( \#(C(T_1, \ldots, T_n), C(T_1, \ldots, T_n)) \)) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \). Since every meta variable in \( T_1 \) and \( T_2 \) appears at most once we can conclude that \( C(\text{elim}(\#(T_1, T_1))), \ldots, C(\text{elim}(\#(T_n, T_n))) \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \). Thus we are given that \( \vdash_{\mathcal{A} \mathcal{R}} \) C(\( \text{elim}(\#(T_1, T_1))), \ldots, C(\text{elim}(\#(T_n, T_n))) \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \). Therefore, \( \vdash_{\mathcal{A} \mathcal{R}} \) C(\( \#(T_1, T_1)), \ldots, \#(T_n, T_n) \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \). According to Lemma B.5 and Lemma B.6 there exist specifications:

1. \( \vdash_{\mathcal{A} \mathcal{R}} \) \( T_i \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \)

2. \( \#(S_i', S_i') \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \) \( \Rightarrow \)

3. \( \text{conc}(S_i') = \text{conc}(S_i') = \text{conc}(S_i') \).
Therefore we can conclude that \( \vdash_{\mathcal{AR}^*} C(T_1', \ldots, T_n') \text{ Sat}# C(S_{1}'_1, \ldots, S_{n}'_1) \triangleright S_{n+1} \),

\( \vdash_{\mathcal{AR}^*} C(T_1', \ldots, T_n') \text{ Sat}# C(S_{1}'_2, \ldots, S_{n}'_2) \triangleright S_{n+1} \).

Based on the induction hyp.

\( \vdash_{\mathcal{AR}^*} T_1 \text{ Sat}# C(S_{1}'_1, \ldots, S_{n}'_1) \triangleright S_{n+1} \),

\( \vdash_{\mathcal{AR}^*} T_2 \text{ Sat}# C(S_{1}'_2, \ldots, S_{n}'_2) \triangleright S_{n+1} \).

It is possible to prove that 

\( #(C(S_{1}'_1, \ldots, S_{n}'_1) \triangleright S_{n+1}, C(S_{1}'_2, \ldots, S_{n}'_2) \triangleright S_{n+1}) \) is well-defined and thus the jigsaw rule gives

\( \vdash_{\mathcal{AR}^*} #(T_1, T_2) \text{ Sat}# #(C(S_{1}'_1, \ldots, S_{n}'_1) \triangleright S_{n+1}, C(S_{1}'_2, \ldots, S_{n}'_2) \triangleright S_{n+1}) \) and the Substitution rule gives

\( \vdash_{\mathcal{AR}^*} #(T_1, T_2) \text{ Sat}# C(S_{1}'', \ldots, S_{n}'') \triangleright (S_{n+1} \land S_{n+1}) \). □

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

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