A Formal Library of Set Relations and Its Application to Synchronous Languages[☆]

Camilo Rocha[1]

Department of Computer Science, University of Illinois, Urbana IL 61801, USA

César Muñoz

NASA Langley Research Center, MS. 130, Hampton, VA 23681, USA

Gilles Dowek

École Polytechnique and INRIA, LIX, École Polytechnique, 91128 Palaiseau Cedex, France

Abstract

Set relations are particularly suitable for specifying the small-step operational semantics of synchronous languages. In this paper, a formal library of set relations for the definition, verification of properties, and execution of binary set relations is presented. The formal library consists of a set of theories written in the *Prototype Verification System* (PVS) that contains definitions and proofs of properties, such as determinism and compositionality, for synchronous relations. The paper also proposes a serialization procedure that enables the simulation of synchronous set relations via set rewriting systems. The library and the serialization procedure are illustrated with the rewriting logic semantics of the *Plan Execution Interchange Language* (PLEXIL), a rich synchronous plan execution language developed by NASA to support autonomous spacecraft operations.

Keywords: set relations, synchronous languages, small-step semantics, rewriting logic semantics, plan execution

[☆] This document is a collaborative effort; authors are listed in reversed alphabetical order. This work was supported in part by the National Aeronautics and Space Administration at Langley Research Center under the Research Cooperative Agreement No. NCC-1-02043 awarded to the National Institute of Aerospace while the first and third authors were visiting the institute and the second author was resident there. The first author was partially supported by NSF Grant CNS 08-34709.

Email addresses: `hrochan2@cs.illinois.edu` (Camilo Rocha), `cesar.a.munoz@nasa.gov` (César Muñoz), `gilles.dowek@polytechnique.fr` (Gilles Dowek)

[1] Corresponding author.

1. Introduction

Defining the semantic relation of a language and formally reasoning about this relation is generally difficult, time consuming, and error-prone. Hence, a formal framework supporting both verification and execution of the operational semantics of a programming language is useful, for it would reduce the amount of work needed to define its semantic relation and formally prove its properties. Moreover, since programming languages tend to evolve constantly, tools supporting rapid yet correct prototyping of their execution are highly valuable.

The definition of such a framework would be a major endeavor if it had to be done from scratch for each language. As a minimum, the framework should include a formal library of binary relations general enough to support the definitions of useful operations, their execution, and formal proofs of their properties. Fortunately, semantic relations are, in general, built from simple relations with a limited number of operations, such as reflexive-transitive closure, reduction to normal form, parallel closure, etc., which have been studied extensively [2].

Operations such as the synchronous relation, which is at the basis of the operational semantics [27] of synchronous languages such as Esterel [4], Lustre [6], and Signal [15], have been less formally studied in an abstract setting. Synchronous languages were introduced in the 1980s to program *reactive systems*, i.e., systems whose behavior is determined by their continuous reaction to the environment where they are deployed. Nowadays, synchronous languages are extensively used in embedded applications and automatic control software of critical nature. The family of synchronous languages is characterized by the *synchronous hypothesis*, which states that a reactive system is arbitrarily fast and able to react immediately in no time to stimuli from the external environment. One of the main consequences of the synchronous hypothesis is that components running in parallel are perfectly synchronized and cannot arbitrarily interleave. The implementation of a synchronous language in a sequential machine requires the simulation of the synchronous semantics into an asynchronous computation model and must ensure the validity of the synchronous hypothesis.

This paper presents a formal library of binary set relations for the specification, proof of properties, and execution of synchronous relations. The formal library consists of a set of theories written in the higher-order specification language of the Prototype Verification System (PVS) [26]. The PVS theories contain formal definitions of abstract binary relations on sets, and proofs of sufficient conditions of properties such as determinism and compositionality, which are of particular interest in the development of synchronous languages [9, 20].

The paper also proposes a serialization procedure that correctly and completely simulates asynchronously a given synchronous relation on sets. Since the constructed relation is built on top of an asynchronous one, it can be specified in asynchronous rewriting engines that support set rewriting such as Maude [7].

The application of the formal library and the serialization procedure are illustrated with the specification of a small-step semantics of the *Plan Execution Interchange Language* (PLEXIL) [14] in PVS and its rewriting logic specification in Maude. Plan execution is a cornerstone in systems involving intelligent software agents such as robotics, unmanned vehicles, and habitats. PLEXIL[2] is a synchronous plan execution language developed by NASA to support spacecraft automation, and it has been used on midsize autonomy applications at NASA such as robotic rovers, a prototype of a Mars drill, and to demonstrate automation for the International Space Station.

Outline. Section 2 presents the definitions of set relations included in the formal library. Sufficient conditions for determinism and compositionality of these relations are given in Section 3. Section 4 illustrates how the formalism presented in this paper can be used to define the small-step semantics of a simple synchronous language. Section 5 describes the serialization procedure for synchronous relations. Section 6 gives an overview of the small-step semantics of PLEXIL in PVS and the rewriting logic specification of this semantics in Maude. Section 7 discusses related work and concludes this work.

The mathematical development presented in sections 2, 3, and 6.1 have been formalized and mechanically verified in PVS. The rewriting logic specification discussed in Section 6.2 has been implemented in Maude. The mathematical developments in PVS and the implementation in Maude are all publicly available from `http://shemesh.larc.nasa.gov/people/cam/PLEXIL`.

2. Set Relations

The proposed library includes several definitions related to abstract set relations. All the definitions presented in this section are formally specified in PVS. However, for readability, this paper uses standard mathematical notation.

Let $\mathcal{U}$ be a set whose elements are denoted $A, B, \ldots$ and let $\rightarrow$ be a binary relation on $\mathcal{U}$. An element $A \in \mathcal{U}$ is called a $\rightarrow$-*redex* if it is a reducible element for $\rightarrow$, i.e., if there exist $B \in \mathcal{U}$ such that the *pair* $\langle A;B\rangle \in \rightarrow$. As usual, the fact that the pair $\langle A;B\rangle$ is in $\rightarrow$ will be denoted $A \rightarrow B$. Moreover, $A \not\rightarrow B$ denotes the fact that the pair $\langle A;B\rangle$ is not in $\rightarrow$.

The *identity* relation, *n-fold composition*, and *reflexive-transitive closure* of $\rightarrow$ are defined as usual, e.g, see [2], and denoted by $\rightarrow^0$, $\rightarrow^n$, and $\rightarrow^*$, respectively. Observe that $\rightarrow$ and $\rightarrow^1$ denote the same binary relation. If $A \rightarrow^* B$ and B is not reducible for $\rightarrow$, then B is called a $\rightarrow$-*normal form* and, furthermore, B is said to be a $\rightarrow$-normal form of A. The *normalized relation* $\rightarrow^\downarrow$ of $\rightarrow$ relates elements to their normal forms and it is formally defined as follows.

Definition 1 (Normalized relation)**.** *The relation $\rightarrow^\downarrow$ denotes the set of pairs $\langle A;B\rangle$ in $\mathcal{U}\times\mathcal{U}$ such that $A\rightarrow^\downarrow B$ if and only if $A\rightarrow^* B$ and B is a $\rightarrow$-normal form.*

Note that $\rightarrow^\downarrow$-normal forms do not have a $\rightarrow$-normal form. Indeed, assume that A is a $\rightarrow^\downarrow$-normal form and that B is one its $\rightarrow$-normal forms. By definition, it holds that $A\rightarrow^* B$. Since B is a $\rightarrow$-normal form of A, it holds that $A\rightarrow^\downarrow B$, which contradicts the fact that A is $\rightarrow^\downarrow$-normal form. Hence, the only elements in $\mathcal{U}$ that are $\rightarrow^\downarrow$-normal forms are those that do not have a $\rightarrow$-normal form.

[2] PLEXIL is available from `http://shemesh.larc.nasa.gov/people/cam/PLEXIL`.

Example 1. *Let $\mathcal{U}$ be the set containing the distinct elements A, B, C, D, and the relation $\rightarrow$ defined on $\mathcal{U}$ as follows:*

$$\begin{array}{rcl} A & \rightarrow & B, \\ C & \rightarrow & C. \end{array}$$

The elements A and C are $\rightarrow$-redexes, while the elements B and D are $\rightarrow$-normal forms. Moreover, the relation $\rightarrow^{\downarrow}$ consists of the pairs $\langle A;B \rangle$, $\langle B;B \rangle$ and $\langle D;D \rangle$. The element C is the only $\rightarrow^{\downarrow}$-normal form.

For the rest of this paper, it is assumed that $\mathcal{U}$ is the family of all finite sets over an abstract, possibly infinite, set T, i.e., $\mathcal{U} \subseteq 2^T$ and, therefore, $\rightarrow$ is a binary relation on finite sets of T. The elements of T will be denoted by lowercase letters $a, b, c, \ldots$ and will be called *terms*. When it is clear from the context, curly brackets are omitted from set notation, e.g., $a, b \rightarrow b$ denotes $\{a, b\} \rightarrow \{b\}$. Because of this abuse of notation, the symbol ',' is overloaded to denote set union, e.g., if A denotes the set $\{a, b\}$, B denotes the set $\{c, d\}$, and D denotes the set $\{d, e\}$, the notation $A, B \rightarrow B, D$ denotes $\{a, b, c, d\} \rightarrow \{c, d, e\}$.

The *asynchronous* relation $\rightarrow^{\Box}$ of $\rightarrow$ is the relation defined as the congruence closure of $\rightarrow$.

Definition 2 (Asynchronous relation)**.** *The relation $\rightarrow^{\Box}$ denotes the set of pairs $\langle A;B \rangle$ in $\mathcal{U} \times \mathcal{U}$ such that $A \rightarrow^{\Box} B$ if and only if there exist sets A' and B' such that $A' \subseteq A$, $A' \neq \emptyset$, $A' \rightarrow B'$ and $B = (A \setminus A') \cup B'$.*

The *parallel* relation $\rightarrow^{\parallel}$ of $\rightarrow$ is the relation defined as the parallel closure of $\rightarrow$.

Definition 3 (Parallel relation)**.** *The relation $\rightarrow^{\parallel}$ denotes the set of pairs $\langle A;B \rangle$ in $\mathcal{U} \times \mathcal{U}$ such that $A \rightarrow^{\parallel} B$ if and only if there exist $A_1, \ldots, A_n$, nonempty, pairwise disjoint subsets of A, and sets $B_1, \ldots, B_n$ such that $A_i \rightarrow B_i$ and $B = (A \setminus \bigcup_{1 \leq i \leq n} A_i) \cup \bigcup_{1 \leq i \leq n} B_i$.*

Example 2. *Let T be the set of distinct terms a, b, c, d, e, and the relation $\rightarrow$ defined on $\mathcal{U}$ as follows*

$$\begin{array}{rcl} a, b & \rightarrow & b, d, \\ c & \rightarrow & d, \\ a, e & \rightarrow & d. \end{array}$$

It holds that $a, b, c, e \rightarrow^{\Box} b, c, d, e$, because $\{b, c, d, e\} = (\{a, b, c, e\} \setminus \{a, b\}) \cup \{b, d\}$. Moreover, $a, b, c, e \rightarrow^{\parallel} b, d, e$, because $\{b, d, e\} = (\{a, b, c, e\} \setminus (\{a, b\} \cup \{c\})) \cup \{b, d\} \cup \{d\}$.

The synchronous relation of $\rightarrow$ is defined as a subset of the parallel closure of $\rightarrow$, where a given strategy selects the $\rightarrow$-redexes.

Definition 4 (Strategy)**.** *A* strategy *for $\rightarrow$ is a function that maps elements $A \in \mathcal{U}$ into a set of parts of A such that if $A' \in s(A)$ then A' is a $\rightarrow$-redex.*

Since $\mathcal{U}$ is the family of all finite sets in T, if $A \in \mathcal{U}$, then set $s(A)$ is also a family of finite sets. Therefore, without loss of generality, it can be considered that for $A \in \mathcal{U}$, $s(A) = \{A_1, \ldots, A_n\}$, where for $1 \leq i < j \leq n$, $A_i \neq \emptyset$, A_i is a $\rightarrow$-redex, $A_i \subseteq A$, and $A_i \cap A_j = \emptyset$.

Definition 5 (Synchronous relation)**.** *Let s be a strategy for $\rightarrow$, the relation $\rightarrow^s$ denotes the set of pairs $\langle A;B\rangle$ in $\mathcal{U} \times \mathcal{U}$ such that $A \rightarrow^s B$ if and only if there exist $B_1, \ldots, B_n$ such that $s(A) = \{A_1, \ldots, A_n\}$, $A_i \rightarrow B_i$, and $B = (A \setminus \bigcup_{1 \leq i \leq n} A_i) \cup \bigcup_{1 \leq i \leq n} B_i$.*

Example 3. *Let $\rightarrow$ be the binary set relation defined in Example 2 and s_1, s_2, and s_3 be strategies for $\rightarrow$ such that:*

$$\begin{array}{rcl} s_1(\{a,b,c,e\}) & = & \{\{a,b\},\{c\}\}, \\ s_2(\{a,b,c,e\}) & = & \{\{a,e\},\{c\}\}, \\ s_3(\{a,b,c,e\}) & = & \{\{c\}\}. \end{array}$$

It holds that

$$\begin{array}{lcl} a,b,c,e & \rightarrow^{s_1} & b,d,e, \\ a,b,c,e & \rightarrow^{s_2} & b,d, \\ a,b,c,e & \rightarrow^{s_3} & a,b,d,e. \end{array}$$

A natural way of defining strategies is via priorities. A *priority* for a relation $\rightarrow$ is a function p that maps $\rightarrow$-redexes into numbers.

Definition 6 (Maximal redex)**.** *Let $A \in \mathcal{U}$ and p be a priority function for $\rightarrow$. A nonempty subset B of A is said to be a* maximal $\rightarrow$-redex *of A if B is a $\rightarrow$-redex and for all $\rightarrow$-redex A' such that $A' \subseteq A$, $A' \neq \emptyset$, $A' \neq B$, and $A' \cap B \neq \emptyset$, it holds that $p(B) > p(A')$.*

The maximal $\rightarrow$-redexes of $A \in \mathcal{U}$ are all pairwise disjoint by construction.

Definition 7. *Given a priority function p for $\rightarrow$, the* maximal $\rightarrow$-redexes strategy *for p is the function that maps elements $A \in \mathcal{U}$ into the set of its maximal $\rightarrow$-redexes.*

Example 4. *Let $\rightarrow$ be the binary set relation defined in Example 2 and p be a priority for $\rightarrow$ such that $p(\{a,b\}) > p(\{a,e\})$. The maximal $\rightarrow$-redexes strategy for p is s_1, as defined in Example 3.*

3. Determinism and Compositionality

In addition to the definition of the binary relations presented in Section 2, the proposed library includes formal proofs of several properties related to determinism and compositionality of the relations $\rightarrow^n$, $\rightarrow^\downarrow$, and $\rightarrow^s$, based on the determinism and compositionality of the relation $\rightarrow$. All lemmas presented in this section are formally proved in PVS. A summary of the PVS results presented in this section is electronically available from `http://shemesh.larc.nasa.gov/people/cam/PLEXIL/library_summary.txt`.

3.1. Determinism

A relation is deterministic if each element is in relation with at most one element.

Definition 8 (Determinism)**.** *A binary relation $\to$ on $\mathcal{U}$ is said to be* deterministic *if for all A, B_1, and B_2 in $\mathcal{U}$, it holds that $A \to B_1$ and $A \to B_2$ implies $B_1 = B_2$.*

Determinism is preserved by the n-fold relation, the normalized relation, and the synchronous relation.

Lemma 1 (Determinism of $\to^n$)**.** *$\to^n$ is deterministic if $\to$ is deterministic.*

Lemma 2 (Determinism of $\to^\downarrow$)**.** *$\to^\downarrow$ is deterministic if $\to$ is deterministic.*

Lemma 3 (Determinism of $\to^s$)**.** *$\to^s$ is deterministic if $\to$ is deterministic and s is a strategy for $\to$.*

In general, the relations $\to^*$, $\to^\square$ and $\to^\parallel$ are not deterministic, not even in the case where the relation $\to$ is deterministic.

Example 5. *Let T be the set of distinct terms a, b, c, d, e and $\to$ be the binary relation on $\mathcal{U}$ defined as follows.*

$$\begin{array}{rcl} a, b & \to & b, \\ b & \to & d, \\ a, c & \to & e. \end{array}$$

The relation $\to$ is deterministic because every element of $\mathcal{U}$ is related to at most one element. The relation $\to^*$ is non-deterministic because $a, b \to^* b$ and $a, b \to^* d$, but $\{b\} \neq \{d\}$. The relation $\to^\square$ is non-deterministic because $a, b \to^\square b$ and $a, b \to^\square a, d$, but $\{b\} \neq \{a, d\}$. The relation $\to^\parallel$ is non-deterministic because $a, b, c \to^\parallel b, c$ and $a, b, c \to^\parallel d, e$, but $\{b, c\} \neq \{d, e\}$. Assume that the strategy s is defined such that

$$s(\{a, b, c\}) = \{\{a, c\}, \{b\}\},$$

the relation $\to^s$ is deterministic. In particular, the only element related to $\{a, b, c\}$ in $\to^s$ is $\{d, e\}$.

3.2. Compositionality

Intuitively, a relation $\to$ is compositional if $A \to A'$ and $B \to B'$ implies that $A \mid B \to A' \mid B'$, where $\mid$ is a binary operation that maps elements in $\mathcal{U} \times \mathcal{U}$ into elements in $\mathcal{U}$. Since $\mathcal{U} \subseteq 2^T$, the operation $\mid$ can be simply defined as the set union operation. However, most of the results presented in this section hold for an abstract definition of the binary operation $\mid$.

The compositionality property is not stated for all elements A and B, but for pairs $\langle A;B \rangle$ in a given set of elements $\mathcal{E} \subseteq \mathcal{U} \times \mathcal{U}$.

Definition 9 (Compositionality)**.** *The relation $\to$ is said to be* compositional for a set $\mathcal{E} \subseteq \mathcal{U} \times \mathcal{U}$ *if for all A, A', B, and B' in $\mathcal{U}$, $\langle A;B \rangle \in \mathcal{E}$, $A \to A'$, and $B \to B'$ implies $A \mid B \to A' \mid B'$.*

3.2.1. Compositionality of n-fold relation

In general, it is not true that the relation $\to^n$ is compositional for $\mathcal{E}$ when $\to$ is compositional for $\mathcal{E}$, as illustrated by the following example.

Example 6. *Define the operation | as set union. Let T be the set of distinct terms a, a', a'', b, b', b'', and $\to$ be the binary relation on $\mathcal{U}$ defined as follows.*

$$\begin{array}{rcl} a & \to & a', \\ a' & \to & a'', \\ b & \to & b', \\ b' & \to & b'', \\ a, b & \to & a', b'. \end{array}$$

The relation $\to$ is compositional for $\mathcal{E} = \{\langle a;b \rangle\}$, but $\to^2$ is not: it holds that $a \to^2 a''$ and $b \to^2 b''$, but $a, b \not\to^2 a'', b''$.

The counter-example above suggests that to achieve compositionality for $\to^n$ the set $\mathcal{E}$ must be *closed* under the relation $\to$.

Definition 10 (Closed)**.** *A set $\mathcal{E} \subseteq \mathcal{U} \times \mathcal{U}$ is said to be* closed *under $\to$, if and only if*

- *$\langle A;B \rangle \in \mathcal{E}$ and $A \to A'$ implies $\langle A';B \rangle \in \mathcal{E}$, and*
- *$\langle A;B \rangle \in \mathcal{E}$ and $B \to B'$ implies $\langle A;B' \rangle \in \mathcal{E}$.*

A weaker condition is the following.

Definition 11 (Weakly closed)**.** *A set of pairs $\mathcal{E} \subseteq \mathcal{U} \times \mathcal{U}$ is said to be* weakly closed *under $\to$, if $A \to A'$, $B \to B'$, and $\langle A;B \rangle \in \mathcal{E}$, implies $\langle A';B' \rangle \in \mathcal{E}$.*

Lemma 4. *If $\mathcal{E}$ is closed under $\to$, then it is weakly closed.*

This weak closure condition happens to be sufficient to prove compositionality of $\to^n$ as shown by the following lemma.

Lemma 5 (Compositionality of $\to^n$)**.** *If $\to$ is compositional for $\mathcal{E}$, and $\mathcal{E}$ is weakly closed under $\to$, then $\to^n$ is compositional for $\mathcal{E}$.*

3.2.2. Compositionality of the normalizing relation

In general, it is not true that the relation $\to^{\downarrow}$ is compositional for $\mathcal{E}$ when $\to$ is compositional for $\mathcal{E}$. It turns out that a stronger notion of compositionality is needed, one which requires necessarily the set $\mathcal{E}$ to be closed under $\to$. However, this closure condition happens to be insufficient as shown by the following counter-example.

Example 7. *Define the operation $|$ as set union. Let T be the set of distinct terms a, a', a'', b, b', b'' and $\rightarrow$ be the binary relation on $\mathcal{U}$ defined as follows.*

$$\begin{array}{rcl} a & \rightarrow & a', \\ a' & \rightarrow & a'', \\ b & \rightarrow & b', \\ a, b & \rightarrow & a', b', \\ a', b & \rightarrow & a'', b'. \end{array}$$

Let $\mathcal{E}$ be the set of pairs $\{\langle a;b\rangle, \langle a;b'\rangle, \langle a';b\rangle, \langle a';b'\rangle, \langle a'';b\rangle, \langle a'';b'\rangle\}$. The relation $\rightarrow$ is compositional for the elements of $\mathcal{E}$ and $\mathcal{E}$ is closed under $\rightarrow$. However, the relation $\rightarrow^{\downarrow}$ is not compositional for $\mathcal{E}$, because it holds that $a \rightarrow^{\downarrow} a''$ and $b \rightarrow^{\downarrow} b'$, but $a, b \not\rightarrow^{\downarrow} a'', b'$.

In this case, a stronger notion of compositionality is needed.

Definition 12 (Strong compositionality)**.** *The relation $\rightarrow$ is said to be* strongly compositional *for a set $\mathcal{E} \subseteq \mathcal{U} \times \mathcal{U}$ if it is compositional for $\mathcal{E}$ and, moreover, for all A, A', B, and B' in $\mathcal{U}$,*

1. *if $\langle A;B\rangle \in \mathcal{E}$, $A \rightarrow A'$, and B is a $\rightarrow$-normal form, then $A \mid B \rightarrow A' \mid B$,*
2. *if $\langle A;B\rangle \in \mathcal{E}$, A is a $\rightarrow$-normal form and $B \rightarrow B'$, then $A \mid B \rightarrow A \mid B'$, and*
3. *if A and B are $\rightarrow$-normal forms, then $A \mid B$ is a $\rightarrow$-normal form.*

Lemma 6 (Compositionality of $\rightarrow^{\downarrow}$)**.** *If $\mathcal{E}$ is closed under $\rightarrow$, and $\rightarrow$ is strongly compositional for $\mathcal{E}$, then $\rightarrow^{\downarrow}$ is compositional for $\mathcal{E}$.*

In general, the relation $\rightarrow^{\downarrow}$ is not strongly compositional for $\mathcal{E}$. Indeed this would not make much sense since, as explained in Section 2, elements in $\rightarrow^{\downarrow}$-normal form are non-normalizable for $\rightarrow$.

The compositionality property of relations that are built on top of $\rightarrow^{\downarrow}$ may require the closure of $\mathcal{E}$ under $\rightarrow^{\downarrow}$. The following lemma states that this closure property holds when $\mathcal{E}$ is closed under $\rightarrow$.

Lemma 7 (Closure of $\rightarrow^{\downarrow}$)**.** *If a set $\mathcal{E} \subseteq \mathcal{U} \times \mathcal{U}$ is closed under $\rightarrow$, then it is closed under $\rightarrow^{\downarrow}$.*

3.2.3. Compositionality of the synchronous relation

Let s be a strategy and the operation $|$ be the set union.

Definition 13 (Set of disjoints pairs)**.** *A set $\mathcal{E} \subseteq \mathcal{U} \times \mathcal{U}$ is said to be a* set of disjoint pairs *if for all $\langle A;B\rangle \in \mathcal{E}$, $A \cap B = \emptyset$.*

Definition 14 (Commutation with union)**.** *A strategy s is said to* commute with union *for elements of $\mathcal{E}$ if for all $\langle A;B\rangle \in \mathcal{E}$, $s(A \cup B) = s(A) \cup s(B)$.*

Lemma 8 (Strong compositionality of $\rightarrow^{s}$)**.** *If $\mathcal{E}$ is a set of disjoint pairs and the strategy s commutes with union for elements of $\mathcal{E}$, then $\rightarrow^{s}$ is strongly compositional for $\mathcal{E}$.*

When the strategy s is the maximal $\rightarrow$-redex strategy, Lemma 8 requires this strategy to commute with union for $\mathcal{E}$. The following lemma provides sufficient conditions to prove this commutation property.

Lemma 9. *Let $\mathcal{E}$ be a set of disjoint pairs, the maximal $\rightarrow$-redexes strategy commutes with union for elements of $\mathcal{E}$ if for all $\langle A;B\rangle \in \mathcal{E}$ and for all $\rightarrow$-redex C such that $C \subseteq A \cup B$, it holds that $C \subseteq A$ or $C \subseteq B$.*

The compositionality of relations that are built on top of $\rightarrow^s$, such as $(\rightarrow^s)^{\downarrow}$, may require the closure of $\mathcal{E}$ under $\rightarrow^s$. The following lemma provides sufficient conditions for this closure property to hold.

Lemma 10 (Closure of $\rightarrow^s$)**.** *If*

1. *the set $\mathcal{E} \subseteq \mathcal{U} \times \mathcal{U}$ is closed under $\rightarrow$,*
2. *$\langle A;B\rangle \in \mathcal{E}$ implies $\langle B;A\rangle \in \mathcal{E}$,*
3. *$\langle A;B\rangle \in \mathcal{E}$ and $A' \subseteq A$ implies $\langle A';B\rangle \in \mathcal{E}$, and*
4. *for any set C and all $a \in C$, $\langle a;B\rangle \in \mathcal{E}$ implies $\langle \bigcup_{a\in C}\{a\};B\rangle \in \mathcal{E}$,*

then $\mathcal{E}$ is closed under $\rightarrow^s$.

4. Small-Step Semantics of a Simple Synchronous Language

This section illustrates the concepts presented in the previous sections by giving the small-step semantics of a simple synchronous language with arithmetic expressions. The small-step semantic relation of the language will be defined as the synchronous relation $\rightarrow^s$ of a relation $\rightarrow$ for a given strategy s. The relation $\rightarrow$ is called the *atomic* relation of the language.

Consider a language that consists of two kinds of elements: *memory terms* $\texttt{Mem}(m, v)$ and *assignment terms* $l\texttt{:=}u$, where m, l denote memory names and v, u denote numerical values. In this case, the set T consists of all ground terms having the form $\texttt{Mem}(m, v)$ or $m\texttt{:=}v$. As defined in Section 2, the set $\mathcal{U}$ is the family of all finite sets over T.

The atomic relation $\rightarrow$ is defined such that $A \rightarrow B$ if and only if $A = \{\texttt{Mem}(m, v), m\texttt{:=}u\}$ and $B = \{\texttt{Mem}(m, u)\}$, for some memory name m and values v, u. For instance, it holds that

$$\begin{aligned} \texttt{Mem(x,3)},\ \texttt{x:=4} &\quad\rightarrow\quad \texttt{Mem(x,4)}, \\ \texttt{Mem(y,4)},\ \texttt{y:=3} &\quad\rightarrow\quad \texttt{Mem(y,3)}. \end{aligned} \tag{1}$$

The small-step semantic relation of the language can be defined as the synchronous relation $\rightarrow^s$, where s is the maximal $\rightarrow$-redexes strategy for a priority function p that satisfies

$$p(\{\texttt{Mem}(m, v), m\texttt{:=}u\}) = u.$$

This priority function gives a higher priority to $\to$-redexes that assign a higher value. For instance, since $p(\{\texttt{Mem(y,4)},\texttt{y:=3}\}) = 3 > 0 = p(\{\texttt{Mem(y,4)},\texttt{y:=0}\})$ and $\{\texttt{Mem(x,3)},\texttt{x:=4}\} \cap \{\texttt{Mem(y,4)},\texttt{y:=3}\} = \emptyset$, it holds that

$$\begin{aligned} &s(\{\texttt{Mem(x,3)},\texttt{Mem(y,4)},\texttt{x:=4},\texttt{y:=3},\texttt{y:=0}\}) = \\ &\qquad \{\{\texttt{Mem(x,3)},\texttt{x:=4}\},\{\texttt{Mem(y,4)},\texttt{y:=3}\}\}. \end{aligned}$$

Therefore,

$$\texttt{Mem(x,3)},\ \texttt{x:=4},\ \texttt{Mem(y,4)},\ \texttt{y:=3},\ \texttt{y:=0} \quad \to^s \quad \texttt{Mem(x,4)},\ \texttt{Mem(y,3)},\ \texttt{y:=0},$$

and

$$\texttt{Mem(x,3)},\ \texttt{x:=4},\ \texttt{Mem(y,4)},\ \texttt{y:=3},\ \texttt{y:=0} \quad (\to^s)^\downarrow \quad \texttt{Mem(x,4)},\ \texttt{Mem(y,0)}.$$

Since the atomic relation $\to$ is deterministic, the synchronous relation $\to^s$ is also deterministic (Lemma 3). Therefore, by lemmas 1 and 2, the n-fold relation $(\to^s)^n$ and the normalized relation $(\to^s)^\downarrow$ are also deterministic.

Let $\mid$ be the set union operation and $\mathcal{E}$ be a set in $\mathcal{U} \times \mathcal{U}$. In general, the relation $\to$ is not compositional. For instance, although the set of pairs in Formula (1) are in $\to$,

$$\texttt{Mem(x,3)},\ \texttt{x:=4},\ \texttt{Mem(y,4)},\ \texttt{y:=3} \quad \not\to \quad \texttt{Mem(x,4)},\ \texttt{Mem(y,3)}.$$

The relation $\to^s$ is not compositional for an arbitrary set $\mathcal{E}$ either. For instance, it holds that

$$\begin{aligned} \texttt{Mem(y,4)},\ \texttt{y:=3} \quad &\to^s \quad \texttt{Mem(y,3)}, \\ \texttt{Mem(y,4)},\ \texttt{y:=0} \quad &\to^s \quad \texttt{Mem(y,0)}, \end{aligned}$$

but

$$\texttt{Mem(y,4)},\ \texttt{y:=3},\ \texttt{y:=0} \quad \not\to^s \quad \texttt{Mem(y,3)},\ \texttt{Mem(y,0)}.$$

On the other hand, the relation $\to^s$ is strongly compositional, and therefore compositional, for the set $\mathcal{E}$ of *independent* pairs, i.e., pairs $\langle A;B\rangle$ where A and B have no memory names in common. In that case, it can be shown, using Lemma 9, that the maximal $\to$-redexes strategy s commutes with union for the elements of $\mathcal{E}$. Therefore, by Lemma 8, the relation $\to^s$ is strongly compositional for the set $\mathcal{E}$ of independent pairs. Moreover, using lemmas 5 and 6, it can also be verified that the n-fold relation $(\to^s)^n$ and the normalized relation $(\to^s)^\downarrow$ are compositional for $\mathcal{E}$.

The synchronous language defined in this section can be extended with arithmetic expressions recursively formed using memory names, numerical values, and expressions of the form $e_1 \texttt{+} e_2$, where e_1 and e_2 denote ground arithmetic expressions. Of course, other type of expressions can be considered, but the ones defined here suffice for the discussion that follows. Terms in the extended language are of the form $\mathtt{Mem}(m,v)$ or $m\texttt{:=}e$, where e is a ground arithmetic expression.

The small-step semantics of the extended language requires the definition of an evaluation function *eval* that takes as inputs an element $A \in \mathcal{U}$, which is a set of ground terms, and a ground expression e. It returns a numerical value. This function is inductively defined on e as follows.

$$\mathit{eval}(A,e) = \begin{cases} v & \text{if } e \text{ is the numerical value } v, \\ v & \text{if } e \text{ is the memory name } m \text{ and } \mathtt{Mem}(m,v) \in A, \\ v_1 + v_2 & \text{if } e \text{ has the form } e_1 \texttt{+} e_2, \\ & v_1 = \mathit{eval}(A,e_1), \text{ and } v_2 = \mathit{eval}(A,e_2). \end{cases}$$

The function *eval* is not well-defined when A has two terms $\mathtt{Mem}(m,v)$ and $\mathtt{Mem}(m,u)$ such that $v \neq u$, or when there is a memory name m that occurs in e but such that there is no a term $\mathtt{Mem}(m,v)$ in A.

Definition 15. *An element $A \in \mathcal{U}$ is called a* context *if*

1. *for all memory terms $\mathtt{Mem}(m,v) \in A$ and $\mathtt{Mem}(m,u) \in A$, it holds that $v = u$, and*
2. *for all memory names m occurring in an expression e in A, there exists a term $\mathtt{Mem}(m,u) \in A$.*

Henceforth, the upper case Greek letter Γ denotes an arbitrary context.

For all context Γ and ground expression e occurring in Γ, the term $\mathit{eval}(\Gamma,e)$ is well-defined. For example, let Γ be the following context.

$$\Gamma \quad = \quad \{\mathtt{Mem(x,3)},\ \mathtt{Mem(y,4)},\ \mathtt{x:=y},\ \mathtt{y:=x+10},\ \mathtt{y:=x}\}. \tag{2}$$

Then, $\mathit{eval}(\Gamma,\mathtt{x}) = \mathtt{3}$, $\mathit{eval}(\Gamma,\mathtt{x+10}) = \mathtt{13}$, and $\mathit{eval}(\Gamma,\mathtt{y}) = \mathtt{4}$.

For the extended language, the atomic relation $\xrightarrow{\Gamma}$, which depends on a given context Γ, is defined such that $A \xrightarrow{\Gamma} B$ if and only if $A \subseteq \Gamma$, $A = \{\mathtt{Mem}(m,v), m\texttt{:=}e\}$, $B = \{\mathtt{Mem}(m,u)\}$, and $u = \mathit{eval}(\Gamma,e)$, for some memory name m, values v and u, and ground expression e.

For instance, let Γ be defined as in Formula (2). Therefore,

$$\begin{aligned} \mathtt{Mem(x,3)},\ \mathtt{x:=y} &\xrightarrow{\Gamma} \mathtt{Mem(x,4)}, \\ \mathtt{Mem(y,4)},\ \mathtt{y:=x+10} &\xrightarrow{\Gamma} \mathtt{Mem(y,13)}, \\ \mathtt{Mem(y,4)},\ \mathtt{y:=x} &\xrightarrow{\Gamma} \mathtt{Mem(y,3)}. \end{aligned}$$

The synchronous relation $(\xrightarrow{\Gamma})^{s}$ is defined for a strategy s that is the maximal $\rightarrow$-redexes strategy for a priority function p that satisfies

$$p(\{\mathtt{Mem}(m,v), m\texttt{:=}e\}) = w,$$

where $w = \mathit{eval}(\Gamma,e)$. Therefore,

$$\mathtt{Mem(x,3)},\ \mathtt{x:=y},\ \mathtt{Mem(y,4)},\ \mathtt{y:=x+10},\ \mathtt{y:=x}\ \ (\xrightarrow{\Gamma})^{s}\ \ \mathtt{Mem(x,4)},\ \mathtt{Mem(y,13)},\ \mathtt{y:=x}.$$

The relation $\xrightarrow{\Gamma}$ is deterministic for any context Γ. Therefore, by Lemma 3, the synchronous relation $(\xrightarrow{\Gamma})^s$ is also deterministic for any Γ. Using Lemma 8, it can be verified that for all contexts Γ, the synchronous relation $(\xrightarrow{\Gamma})^s$ is strongly compositional for the set $\mathcal{E}$ of independent pairs.

It is tempting to define the semantic relation of the extended language as the synchronous relation $(\xrightarrow{\Gamma})^s$. However, this is not a good idea because this relation depends on a given context Γ, which is fixed. For example, if Γ is defined as in Formula (2), it holds that

$$\texttt{Mem(x,4), Mem(y,13), y:=x} \quad (\xrightarrow{\Gamma})^s \quad \texttt{Mem(x,4),Mem(y,3)}.$$

A better semantic relation, which does not depend on a fixed context Γ, is given by the relation $\Rightarrow_{\rightarrow,s}$ defined such that $A \Rightarrow_{\rightarrow,s} B$ if and only $A\ (\xrightarrow{A})^s\ B$. In this case,

$$\begin{aligned}\texttt{Mem(x,3),x:=y,Mem(y,4),y:=x+10,y:=x} \quad &\Rightarrow_{\rightarrow,s} \quad \texttt{Mem(x,4),Mem(y,13),y:=x},\\ \texttt{Mem(x,4), Mem(y,13), y:=x} \quad &\Rightarrow_{\rightarrow,s} \quad \texttt{Mem(x,4),Mem(y,4)}.\end{aligned}$$

Since $(\xrightarrow{\Gamma})^s$ is deterministic for any context Γ, the semantic relation $\Rightarrow_{\rightarrow,s}$ is deterministic. Therefore, by lemmas 1 and 2, the n-fold relation $\Rightarrow^n_{\rightarrow,s}$ and the normalized relation $\Rightarrow_{\rightarrow,s}{\downarrow}$ are also deterministic. Since $(\xrightarrow{\Gamma})^s$ is strongly compositional for the set $\mathcal{E}$ of independent pairs for any context Γ, the semantic relation $\Rightarrow_{\rightarrow,s}$ is also strongly compositional, therefore compositional, for the same set $\mathcal{E}$. Finally, using lemmas 5 and 6, it can be shown that the n-fold relation $\Rightarrow^n_{\rightarrow,s}$ and the normalized relation $\Rightarrow_{\rightarrow,s}{\downarrow}$ are also compositional.

5. Serialization Procedure for Synchronous Relations

Rewriting systems are a computational way of defining binary relations and therefore provide a suitable framework for specifying and executing operational semantics of programming languages [24]. Specifying semantic relations, as the ones presented in Section 4, by using rewriting systems poses a practical challenge. The semantic relations $\rightarrow^s$ and $\Rightarrow_{\rightarrow,s}$ in Section 4 are defined on top of synchronous relations. However, rewriting systems typically implement the maximal concurrency of rewrite rules by asynchronous rewriting. This section proposes a procedure, called *serialization*, that enables the specification of synchronous relations by means of asynchronous ones.

Let R_1 and R_2 be two binary relations on $\mathcal{U}$. If $R_1 \subseteq R_2$, then the relation R_1 is said to *correctly simulate* R_2 and, conversely, the relation R_2 is said to *completely simulate* R_1. It is noted that if R_1 correctly and completely simulates R_2 then $R_1 = R_2$.

For instance, it is easy to see that the synchronous relation $\rightarrow^s$, where $\rightarrow$ is a binary relation on $\mathcal{U}$ and s is a strategy for $\rightarrow$, is included in the relation $(\rightarrow^{\square})^*$, i.e., if $A \rightarrow^s B$, then $A\ (\rightarrow^{\square})^*\ B$ for $A, B \in \mathcal{U}$. Hence, the transitive closure of the asynchronous relation of →, i.e., $(\to^{\square})^*$, completely simulates the synchronous relation $\to^s$. Since the opposite inclusion is not true, the transitive closure of the asynchronous relation does not correctly simulate the synchronous relation.

The serialization procedure has as parameters an arbitrary set T, a binary relation → on $\mathcal{U}$ (the family of all finite sets over T), and a strategy s for →, and it outputs a relation $R_{\to,s}$ that correctly and completely simulates $\to^s$.

The procedure is defined by the following steps.

1. A new set $T^\bullet$ is defined such that there is a bijection *mark* that maps terms in T into terms in $T^\bullet$. The families of finite sets over $T^\bullet$ and $T \cup T^\bullet$ are denoted, respectively, $\mathcal{U}^\bullet$ and $\mathcal{U}^\odot$. The bijection *mark* extends naturally to elements in $\mathcal{U}$, i.e., it extends to a bijection that maps elements in $\mathcal{U}$ into elements in $\mathcal{U}^\bullet$.
2. The binary relation $\to^\bullet$ on $\mathcal{U}^\odot$ is defined such that $A \to^\bullet C$ if and only if $A \to B$ and $C = \mathit{mark}(B)$ (observe that $A \in \mathcal{U}$ and $C \in \mathcal{U}^\bullet$).
3. The relation $R_{\to,s}$ on $\mathcal{U}$ is defined such that $\langle A;B\rangle \in R_{\to,s}$ if and only if $B = (A \setminus A') \cup \mathit{mark}^{-1}(B')$, where

$$\begin{array}{ccc} s(A) & = & \{A_1 \ldots A_n\}, \\ A' & = & \bigcup\limits_{1\le i\le n} A_i, \\ A' & ((\to^\bullet)^\square)^\downarrow & B'. \end{array}$$

Theorem 1. *Given a binary relation → on $\mathcal{U}$ and a strategy s for →, the relation $R_{\to,s}$, constructed by the serialization procedure, correctly and completely simulates the synchronous relation $\to^s$, i.e., $R_{\to,s} = \to^s$.*

Proof. It suffices to prove that for all $A, B \in \mathcal{U}$, $\langle A;B\rangle \in R_{\to,s}$ if and only $A \to^s B$. Let $A \in \mathcal{U}$ be such that $s(A) = \{A_1, \ldots, A_n\}$, where the elements $A_i \subseteq A$ are non-empty, pairwise disjoint, →-redexes. Define $A' = \bigcup_{1\le i\le n} A_i$.

Simulation correctness. Assume that $\langle A;B\rangle \in R_{\to,s}$. By definition of $R_{\to,s}$, $B = (A \setminus A') \cup \mathit{mark}^{-1}(B')$, where $A'\ ((\to^\bullet)^\square)^\downarrow\ B'$. Since the elements A_i are pairwise disjoint →-redexes, they are also pairwise disjoint $\to^\bullet$-redexes. Moreover, since $\to^\bullet$ relates elements in $\mathcal{U}$ into elements in $\mathcal{U}^\bullet$, the normalized reduction $((\to^\bullet)^\square)^\downarrow$ is exactly the n-fold relation $((\to^\bullet)^\square)^n$, where n is the size of $s(A)$. Therefore, $B' = \bigcup_{1\le i\le n} \mathit{mark}(B_i)$, where $A_i \to B_i$. Since $\mathit{mark}^{-1}(B') = \bigcup_{1\le i\le n} B_i$, it holds that $A \to^s B$.

Simulation completeness. Assume that $A \to^s B$. By definition of $\to^s$, $B = (A \setminus A') \cup \bigcup_{1\le i\le n} B_i$, where $A_i \to B_i$. By definition of $\to^\bullet$, it holds that $A_i \to^\bullet \mathit{mark}(B_i)$. Since the elements A_i are pairwise disjoint $\to^\bullet$-redexes, it holds that $A'\ ((\to^\bullet)^\square)^*\ B'$, where $B' = \bigcup_{1\le i\le n} \mathit{mark}(B_i)$. However, B' is a $(\to^{\bullet\square})$-normal form. Therefore, it holds that $A'\ ((\to^\bullet)^\square)^\downarrow\ B'$. Since $\mathit{mark}^{-1}(B') = \bigcup_{1\le i\le n} B_i$, it holds that $\langle A;B\rangle \in R_{\to,s}$.

□

6. Small-Step Semantics of PLEXIL

This section gives an overview of the small-step semantics of PLEXIL, a plan execution language developed by NASA to support space operations. PLEXIL is a synchronous language for specifying actions to be executed by an autonomous system. These actions can be part of normal spacecraft operations or they can respond to unexpected changes in the environment.

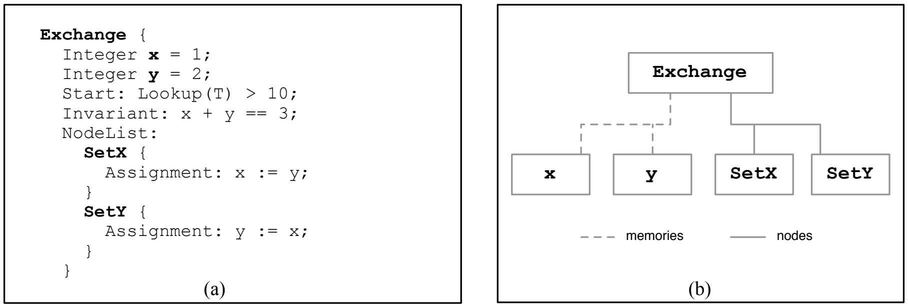
A PLEXIL program, called a *plan*, is a tree of *nodes* representing a hierarchical decomposition of tasks. Interior nodes, called *list nodes*, provide control structure and naming scope for local variables. The primitive actions of a plan are specified in the leaf nodes. Leaf nodes can be *assignment nodes*, which assign values to local memories, *command nodes*, which call external commands, or *empty nodes*, which do nothing. PLEXIL plans interact with a functional layer that provides the interface with the external environment. This functional layer executes the external commands and communicate the status and result of their execution to the plan through external variables.

Nodes have an *execution state*, which can be `Inactive`, `Waiting`, `Executing`, `Finishing`, or `Finished`, and an *execution outcome*, which can be `None`, `Success`, or `Failure`. They can declare local variables, accessible to the node in which they are declared and all its descendants. In contrast to local variables, which have a hierarchical scope, the execution state and the execution outcome of a node are available to all nodes in the plan. Assignment nodes have also a *priority* that is used to solve race conditions. The *internal state* of a node consist of the current values of its execution state, execution outcome, and local variables.

Each node is equipped with a set of *gate conditions* and *check conditions* that govern the execution of a plan. The gate conditions are *start condition*, which specifies when a node starts its execution, *end condition*, which specifies when a node ends its execution, *repeat condition*, which specifies when a node can repeat its execution, and *skip condition*, which specifies when the execution of a node can be skipped. The check conditions signal abnormal execution states of a node and they are *pre-condition*, *post-condition*, and *invariant*. The language includes basic Boolean, arithmetic, and string expressions. It also includes *lookup expressions* that read the value of *external variables* provided to the plan by the functional layer. Expressions appear in conditions, assignments, and arguments of commands.

The execution in PLEXIL is driven by external events that trigger changes in the gate conditions. All nodes affected by a change in a gate condition synchronously respond to the event by modifying their internal state. These internal modifications may trigger more changes in gate conditions, which in turn are synchronously processed until quiescence by all nodes involved. External events are considered in the order in which they are received. An external event and all its cascading effects are processed before the next event is considered. This behavior is known as run-to-completion semantics.

Consider the PLEXIL plan in Figure 1. The plan consists of a root node `Exchange` of type list, and leaf nodes `SetX` and `SetY` of type *assignment*. The node `Exchange` declares to local variables `x` and `y`. The values of these variables



```
Exchange {
  Integer x = 1;
  Integer y = 2;
  Start: Lookup(T) > 10;
  Invariant: x + y == 3;
  NodeList:
    SetX {
      Assignment: x := y;
    }
    SetY {
      Assignment: y := x;
    }
}
```

(a)

(b)

Figure 1: A PLEXIL Plan

are exchanged by the synchronous execution of the node assignments `SetX` and `SetY`. The node `Exchange` also declares a start condition and an invariant condition. The start condition states that the node waits for an external variable `T` to be greater than 0 before starting its execution. The invariant condition states that at any execution step the values of `x` and `y` add to 3.

The small-step semantics of PLEXIL has been specified in PVS' higher-order logic and Maude's rewriting logic. Both formalizations are available from `http://shemesh.larc.nasa.gov/people/cam/PLEXIL`. This paper focuses on the synchronous and run-to-completion aspects of the semantics. For a more complete presentation of the formal semantics, the reader is referred to [11, 12].

6.1. Synchronous and Run-To-Completion Semantics of PLEXIL

The mathematical development presented in this section has been written and mechanically verified in PVS. It is based on the formal library of set relations presented in sections 2 and 3.

Although PLEXIL is much more complex than the synchronous language with arithmetic expressions presented in Section 4, the development of PLEXIL's small-step semantics follows a similar approach to that of the simple language. First, a set of ground terms T is defined. Then, an atomic relation $\xrightarrow{\Gamma}$, for a given context Γ, is defined on $\mathcal{U}$, the family of all finite sets over T. Next, a semantic relation $\Rightarrow_{\to,s}$ is defined on top of the synchronous relation $(\xrightarrow{\Gamma})^{s}$, for a given strategy s for $\to$. More complex relations, such the normalized relation $\Rightarrow_{\to,s}{}^{\downarrow}$, are built on top of the semantic relation as needed. Finally, properties such as determinism and compositionality are proved for each one of these relations in a hierarchical way.

In the case of PLEXIL, the set T of terms consists of *nodes*, written $\mathtt{Node}(N)$, *local variables*, written $\mathtt{Mem}(N, m, v)$, and *external variables*, written $\mathtt{Ext}(m, v)$, where N denotes qualified names, m denotes names, and v denotes literal values. A qualified name uniquely identifies a node in a PLEXIL plan and it consists of the name of the node concatenated to the name of its ancestors. The term $\mathtt{Node}(N)$ represents a record $[\mathtt{Name} = N, \mathtt{Type} = t, \ldots, \mathtt{State} = s, \mathtt{Outcome} = o]$

that contains the complete definition of the node N of a PLEXIL plan, its current execution state s, and its current execution outcome o. The term $\texttt{Mem}(N, m, v)$ represents a local variable m, which is declared in the node N of a PLEXIL plan, with a current value v. The term $\texttt{Ext}(m, v)$ represents an external variable m with a current value v.

Dot-notation is used to access the fields of the record represented by a node term, e.g., $\texttt{Node}(N).\texttt{Invariant}$ refers to the Boolean expression specifying the invariant condition of the node N in PLEXIL plan and $\texttt{Node}(N).\texttt{State}$ refers to the current execution state of the same node. Furthermore, the notation

$$\texttt{Node}(N)\ \texttt{with}\ [f_0 = v_0, \ldots, f_n = v_n]$$

is used to represent a record that is exactly as $\texttt{Node}(N)$ except in the fields f_i, where it has the value v_i, for $0 \leq i \leq n$.

A *context* is a set $A \in \mathcal{U}$, where $\mathcal{U}$ is the family of all finite sets over T, if:

1. $\texttt{Node}(N_1) \in A$, $\texttt{Node}(N_2) \in A$, and $N_1 = N_2$ implies that $\texttt{Node}(N_1)$ and $\texttt{Node}(N_2)$ are field by field equal,
2. $\texttt{Mem}(N, m, v_1) \in A$ and $\texttt{Mem}(N, m, v_2) \in A$ implies that $v_1 = v_2$,
3. $\texttt{Ext}(m, v_1) \in A$ and $\texttt{Ext}(m, v_2) \in A$ implies $v_1 = v_2$, and
4. local and external variables occurring in expressions in A are terms in A.

Contexts are denoted by uppercase Greek letters Γ, Σ, etc.

An *environment* is a context that consists only of external variables. Given a syntactically correct PLEXIL plan that only refers to external variables in an environment Σ, an initial context Γ_0 can be constructed such that $\Sigma \subseteq \Gamma_0$ and there is a term in Γ_0 for each node and local variable declared in the plan. Node and memory names are fully qualified in Γ_0. Furthermore, the initial execution state and execution outcome of a node term in Γ_0 are set to `Inactive` and `None`, respectively. Node memories in Γ_0 have as value the initial value declared in the plan (or a default value if none is declared).

Given Γ and a PLEXIL expression e that occurs in Γ, $\mathit{eval}(\Gamma, e)$ is a function defined in a similar way to the evaluation function presented in Section 4.

6.1.1. Atomic Relation

The atomic relation $\overset{\Gamma}{\to}$, for a given context Γ, defines the changes in the internal state of nodes as consequence of changes in their gate conditions. It is formally defined by 42 individual rules (see [11, 12]). For instance, the rule that updates a memory by an assignment node whose execution state is `Executing` is defined such that $A \overset{\Gamma}{\to} B$ if

$$\begin{array}{rcl}
A & \subseteq & \Gamma, \\
A & = & \{\texttt{Node}(N), \texttt{Mem}(M, m, v)\}, \\
\texttt{Node}(N)\texttt{.State} & = & \texttt{Executing}, \\
\texttt{Node}(N)\texttt{.Type} & = & \texttt{Assignment}, \\
\texttt{Node}(N)\texttt{.Body} & = & (M, m)\texttt{:=}e, \\
u & = & \mathit{eval}(\Gamma, e), \mathit{and} \\
B & = & \{\texttt{Node}(N)\ \texttt{with}\ [\texttt{State} = \texttt{IterationEnded}, \\
 & & \ \texttt{Outcome} = \texttt{Success}], \texttt{Mem}(M, m, u)\}.
\end{array}$$

The following theorem is proved by considering all 42 rules that define the atomic relation $\overset{\Gamma}{\to}$.

Lemma 11. *PLEXIL's atomic relation* $\overset{\Gamma}{\to}$ *is deterministic for any context* Γ.

6.1.2. Micro Relation

The synchronous aspect of PLEXIL's execution is captured by the semantic relation $\Rightarrow_{\to,s}$, called *micro*. This relation is defined such that $A \Rightarrow_{\to,s} B$ if and only if $A(\overset{A}{\to})^{s} B$, where s is the maximal redexes strategy for a priority function p that satisfies

$$p(A) = \begin{cases} \max\{\texttt{Node}(N)\texttt{.Priority} \mid \texttt{Node}(N) \in A'\} & \text{if } A' \neq \emptyset, \\ 0 & \text{otherwise,} \end{cases}$$

where $A' = \{\texttt{Node}(N) \in A \mid \texttt{Node}(N)\texttt{.Type} = \texttt{Assignment}\}$.

Lemma 12. *PLEXIL's micro relation* $\Rightarrow_{\to,s}$ *is deterministic.*

Proof. By lemmas 3 and 11, the relation $\to^{s}$ is deterministic for all Γ. Therefore, the relation $\Rightarrow_{\to,s}$ is deterministic. □

Since $\Rightarrow_{\to,s}$ is deterministic, given an environment Σ and an initial context Γ_0 that includes Σ, there are unique contexts Γ_1, ..., Γ_n such that $\Gamma_0 \Rightarrow_{\to,s} \Gamma_1 \Rightarrow_{\to,s} \ldots \Rightarrow_{\to,s} \Gamma_n$. Furthermore, as none of the atomic rules modify the external environment, it hold that $\Sigma \subseteq \Gamma_i$, for $i \leq n$. Each one of these execution steps is called a micro-step.

Let $\mathcal{E}$ be the set of independent pairs of $\mathcal{U}$, i.e., $\langle A;B \rangle \in \mathcal{E}$ if and only if A and B have no names in common. The following theorem is proved using Lemma 9.

Lemma 13. *The strategy* s *commutes with union for the elements of* $\mathcal{E}$.

Lemma 14. *PLEXIL's micro relation* $\Rightarrow_{\to,s}$ *is strongly compositional, and therefore compositional, for the set* $\mathcal{E}$ *of independent pairs.*

Proof. The result follows from lemmas 8 and 13. □

Using Lemma 10, it can be proved that the set $\mathcal{E}$ is closed under $\rightarrow^{s}$, for all Γ. The following lemma follows from that result.

Lemma 15. *The set $\mathcal{E}$ is closed under $\Rightarrow_{\rightarrow,s}$.*

6.1.3. Quiescence Relation

The run-to-completion aspect of PLEXIL's execution is captured by the *quiescence* relation, which is formally defined as the normalized relation $\Rightarrow_{\rightarrow,s}{\downarrow}$.

Lemma 16. *PLEXIL's quiescence relation $\Rightarrow_{\rightarrow,s}{\downarrow}$ is deterministic.*

Proof. By lemmas 2 and 12. □

Lemma 17. *PLEXIL's quiescence relation $\Rightarrow_{\rightarrow,s}{\downarrow}$ is compositional for the set $\mathcal{E}$ of independent pairs.*

Proof. The result follows from lemmas 6, 15, and 14. □

6.2. Rewriting Logic Semantics of PLEXIL

This section presents the rewrite theory $\mathcal{R}_{\mathrm{PXL}}$ that specifies the rewriting logic semantics of PLEXIL at the level of the atomic and micro relations. The construction of $\mathcal{R}_{\mathrm{PXL}}$ follows the serialization procedure presented in Section 5 for a correct and complete simulation of PLEXIL's micro relation $\Rightarrow_{\rightarrow,s}$.

Rewriting logic [22] is a general semantic framework that unifies a wide range of models of concurrency. Rewriting logic specifications can be executed in Maude, a high-performance rewriting logic implementation, and benefit from formal analysis tools available to it, such as Maude's LTL model checker.

A *rewriting logic specification* is a tuple $\mathcal{R} = (\Sigma, E \cup A, R)$, where

- $(\Sigma, E \cup A)$ is an order-sorted equational logic theory with Σ a signature of disjoint sets of sorts partially ordered and a family of sets of operators; A a set of structural axioms (typically associativity, commutativity, and identity) such that there exists a *matching algorithm modulo A* producing a finite number of A-matching substitutions; and E a set of universally quantified *conditional equations* of the form

  $$(\forall X)\; l : t = u \text{ if } \bigwedge_i t_i = u_i,$$

  where X is a set of sorted variables, t, u, t_i, and u_i are Σ-terms with variables among those in X.

- R is a set of universally quantified *conditional rewrite rules* (with equational conditions) of the form

  $$(\forall X)\; l : t \longrightarrow u \text{ if } \bigwedge_i t_i = u_i.$$

Intuitively, $\mathcal{R}$ specifies a *concurrent system*, whose states are elements of the initial algebra $T_{\Sigma/E\cup A}$ specified by the equational theory $(\Sigma, E \cup A)$ and whose *concurrent transitions* are specified by the rules R.

The binary relation $\rightarrow_{\mathcal{R}}$ on $T_{\Sigma/E\cup A}$ is defined such that $[a]_{E\cup A} \rightarrow_{\mathcal{R}} [b]_{E\cup A}$ iff there is a term $a' \in [a]_{E\cup A}$ such that a' can be rewritten to b', using some rule in R, and $b' \in [b]_{E\cup A}$. For arbitrary E and R, whether $[a]_{E\cup A} \rightarrow_{\mathcal{R}} [b]_{E\cup A}$ holds is in general *undecidable*, even when the equations in E are confluent and terminating modulo A. Therefore, the most useful rewrite theories satisfy additional executability conditions under which the relation $\rightarrow_{\mathcal{R}}$ can be reduced to simpler forms of rewriting just modulo A, where both equality modulo A and matching modulo A are decidable. This is the case when E is *ground strongly normalizing* and *ground confluent* modulo A [10], and, furthermore, R is *ground coherent* [31] relative to the equations E modulo A. Intuitively, ground coherence means that the strategy of first simplifying a term to a canonical form with E modulo A and then applying a rule with R modulo A achieves the effect of rewriting with R modulo $E \cup A$.

6.2.1. The Rewrite Theory $\mathcal{R}_{\mathrm{PXL}}$

PLEXIL's rewriting logic specification $\mathcal{R}_{\mathrm{PXL}} = (\Sigma_{\mathrm{PXL}}, E_{\mathrm{PXL}} \cup A_{\mathrm{PXL}}, R_{\mathrm{PXL}})$ is defined as follows.

The signature Σ_{PXL} defines the sets T and $\mathcal{U}$, which are specified by sorts `Object` and `Configuration`, respectively. These sorts are predefined in Maude's `CONFIGURATION` module for object-oriented specifications. The set T consists of objects $\langle N : \mathit{type} \mid \mathtt{attr_1} = \mathtt{val_1}, \ldots, \mathtt{attr_n} = \mathtt{val_n} \rangle$ representing node terms, objects $\langle m.N : \mathtt{Mem} \mid \mathtt{Value} = v \rangle$ representing local variables, and objects $\langle m : \mathtt{Ext} \mid \mathtt{Value} = v \rangle$ representing external variables. The sort `Object` is extended to allow marked versions of objects.

The set of structural axioms A_{PXL} consists of structural axioms for function symbols in Σ_{PXL}. For instance, it defines the laws of associativity (A), commutativity (C), identity (U), and idempotency (I) for elements in $\mathcal{U}$. Therefore, a term $A \in \mathcal{U}$ is actually an equivalence class of terms, i.e., the set of terms representing all contexts congruent to A modulo ACUI.

The equational theory $(\Sigma_{\mathrm{PXL}}, E_{\mathrm{PXL}} \cup A_{\mathrm{PXL}})$ specifies the functions *eval* and *mark*. The relation $R_{\overset{\Gamma}{\rightarrow},s}$, as constructed by the serialization procedure in Section 5, is given by a function `R` equationally defined in E_{PXL}. The function `R` has as type `Configuration` $\times$ `Configuration` $\longrightarrow$ `Configuration`. It is defined by conditional equations that implement the marked version of PLEXIL's atomic relation, e.g., $\overset{\Gamma}{\rightarrow}^{\bullet}$. The first argument of `R` corresponds to the context Γ, the second argument correspond to the configuration that is synchronously reduced. Since PLEXIL's atomic relation is deterministic (Lemma 11), the relation $\overset{\Gamma}{\rightarrow}^{\bullet}$ is also deterministic, and hence confluent.

Finally, the set of rules R_{PXL}, which specifies PLEXIL's micro relation $\Rightarrow_{\rightarrow,s}$, is defined as the single rewrite rule

$$\Gamma \longrightarrow \mathtt{unmark}(\mathtt{R}(\Gamma, \Gamma)).$$

The run-to-completion semantics of Exchange is depicted at the level of the micro relation in Figure 6.2.1. Since the start condition of Exchange is true in the initial context Γ_0, a series of synchronous reactions occurs until quiescence is reached at the ninth micro-step. The values of memories x and y are effectively exchanged exactly in the fifth micro step. For all contexts Γ_i, $0 \leq i \leq 9$, the invariant condition of `x+y = 3` is satisfied.

	Execution State				
	Exchange	SetX	SetY	x	y
Γ_0	Inactive	Inactive	Inactive	1	2
Γ_1	Waiting	Inactive	Inactive	1	2
Γ_2	Executing	Inactive	Inactive	1	2
Γ_3	Executing	Waiting	Waiting	1	2
Γ_4	Executing	Executing	Executing	1	2
Γ_5	Executing	IterationEnded	IterationEnded	2	1
Γ_6	Executing	Finished	Finished	2	1
Γ_7	Finishing	Finished	Finished	2	1
Γ_8	IterationEnded	Finished	Finished	2	1
Γ_9	Finished	Finished	Finished	2	1

Table 1: Summarized execution trace of Exchange

7. Conclusion

Synchronous languages provide a natural model for synchronous reactive systems. They were first proposed in the early 1980's as a technique for modeling and constructing process control systems [18]. Because of their importance to critical embedded systems, significant work has taken place in the application of formal methods to the design and implementation of synchronous languages (see, for example, [11, 28, 8]). State of the art synchronous languages include Esterel [4], Lustre [6], Signal [15], and NASA's *Plan Execution Interchange Language* (PLEXIL) [14]. The Coq system is used in [4] to give the formal foundations of the synchronous language Esterel. The determinism property for Esterel was formally proven by O. Tardieu in [30]. Based on ideas borrowed from process algebras, Lüttgen proposes *step algebras* [21] to model the synchronous step reactions of Harel's Statecharts [17]. All these efforts have led to solid foundations of the respective languages. However, they rely on the specifics of their language of interest. The work presented in this paper has been applied to the PLEXIL language [12, 13]. However, it is based on mathematical constructs that are common to several synchronous languages.

The relationship between synchronous languages and rewriting techniques can be tracked down to *Synchronous Rewriting Systems* [16, 25]. Semantics of synchronous and parallel languages have been already considered and specified in rewriting logic (see, for example, [1, 23]). Synchronous rewriting is often related to *parallel rewriting* because of their similar nature. The notions presented in this paper draw a clear distinction between them, in which synchronous relations can be seen as parallel ones parametrized by a reduction strategy. Rewriting logic has been used previously as a testbed for specifying and animating the semantics of synchronous languages. M. AlTurki and J. Meseguer [1] have studied the rewriting logic semantics of the language Orc, which includes a synchronous reduction relation. T. Serbanuta *et al.* [29] define the execution of P-systems with structured data with continuations. The focus of the former is to use rewriting logic to study the (mainly) non-deterministic behavior of Orc programs, while the focus of the latter is to study the relationship between P-systems and the existing continuation framework for enriching each with the strong features of the other. P. Lucanu [19] studies the problem of the interleaving semantics of concurrency in rewriting logic for synchronous systems from the perspective of P-systems. The contribution of this paper to rewriting logic research is the definition of serialization procedure that enables the simulation of arbitrary synchronous relations on sets via set rewriting systems. Future work on this area will study the development of an extension of Maude in which rewrite rules can be executed synchronously for set rewrite systems given user-defined reduction strategies.

This paper proposed a formalism, which is based on binary set relations, for specifying the small-step semantics of synchronous languages. The basic construct is an atomic set relation on top of which more complex relations, such as the synchronous relation, can be constructed. Properties of these complex relations are proved in a hierarchical way. Moreover, a serialization procedure is provided for the specification of the semantic relation of a synchronous language in the asynchronous computational model supported by traditional rewriting systems. This simple, yet powerful, approach is used to define the semantics of a complex synchronous language such a PLEXIL, to prove properties such as determinism and compositionality in a modular way, and to provide a formal executable interpreter of the language.

Set rewriting systems is at the basis of models of concurrent computations such at the Gamma model [3] and the Chemical Abstract Machine [5]. The formalism presented here closely follows these models but focuses on the synchronous relation and the mechanical verification of its properties. It rests to be studied how this formalism corresponds to higher level mathematical constructs such as Moore and Mealy automata, which are often used for modeling synchronous languages.

The concrete contribution of this paper is a library in PVS that formalizes the mathematical development presented here. The library consist of definitions of six operations that construct relations from simpler relations: the n-fold iteration, the reflexive-transitive closure, the reduction to a normal form, and the asynchronous, parallel, and synchronous relations. For these relations, properties related to determinism and compositionality have been formally proved. An apparent limitation of the library is that it only deals with relations constructed on the same set $\mathcal{U} \subseteq 2^T$. In principle, this is not a major drawback as the set $\mathcal{U}$ can always be extended to include other terms (as it was done for example with the definition of $\to^\bullet$ in Section 5). However, for technical reasons it may be desirable to keep relations on different sets separated and to provide operations, such as Cartesian product, that combine relations defined on different types. A need for such operations was noted when defining higher-level relations on top of PLEXIL's quiescence. These cases were handled appropriately in the PLEXIL semantics, but they were not properly isolated in the formal library. Making these operations explicit in the library will be considered for future work.

Although the formalism presented here has only been used to define the formal semantics of PLEXIL and to prove its properties, it is believed that this formalism can be applied to other deterministic synchronous languages. To the best of the authors' knowledge, there was no mechanized library of abstract set relations suitable for the definition and verification of synchronous relations; neither was there a correctness proof of a serialization procedure for the simulation of synchronous relations by rewrite systems.

Acknowledgements

The authors would like to thank the members of the NASA's Automation for Operation (A4O) project and, especially, the PLEXIL development team led by Michael Dalal at NASA Ames.The authors would also like to thank the anonymous referees for all their comments that helped to improve this paper.

References

[1] AlTurki, M., Meseguer, J., 2008. Reduction semantics and formal analysis of Orc programs. Electr. Notes Theor. Comput. Sci. 200 (3), 25–41.

[2] Baader, F., Nipkow, T., 1998. Term Rewriting and all that. Cambridge University Press, Cambridge.

[3] Banâtre, J.-P., Métayer, D. L., 1990. The GAMMA model and its discipline of programming. Science of Computer Programming 15 (1), 55–77.

[4] Berry, G., 2000. The foundations of Esterel. In: Proof, Language and Interaction: Essays in Honour of Robin Milner. MIT Press, Cambridge, MA, USA, pp. 425–454.

[5] Berry, G., Boudol, G., 1992. The Chemical Abstract Machine. Theoretical Computer Science 96 (1), 217–248.

[6] Caspi, P., Pilaud, D., Halbwachs, N., Plaice, J. A., 1987. Lustre: a declarative language for real-time programming. In: POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. ACM, New York, NY, USA, pp. 178–188.

[7] Clavel, M., Durán, F., Eker, S., Meseguer, J., Lincoln, P., Martí-Oliet, N., Talcott, C., 2007. All About Maude - A High-Performance Logical Framework, 1st Edition. Springer LNCS Vol. 4350.

[8] Dajani-Brown, S., Cofer, D., Bouali, A., 2004. Formal verification of an avionics sensor voter using SCADE. In: Lakhnech, Y., Yovine, S. (Eds.), FORMATS/FTRTFT. Vol. 3253 of Lecture Notes in Computer Science. Springer, pp. 5–20.

[9] De Simone, R., Ressouche, A., 1994. Compositional semantics of Esterel and verification by compositional reductions. Lecture Notes in Computer Science 818, 441–454.

[10] Dershowitz, N., Jouannaud, J. P., 1990. Rewrite systems. In: Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B). The MIT Press, pp. 243–320.

[11] Dowek, G., Muñoz, C., Păsăreanu, C., September 2007. A formal analysis framework for PLEXIL. In: Proceedings of 3rd Workshop on Planning and Plan Execution for Real-World Systems. pp. 45–51.

[12] Dowek, G., Muñoz, C., Păsăreanu, C., 2008. A small-step semantics of PLEXIL. Technical Report 2008-11, National Institute of Aerospace, Hampton, VA.

[13] Dowek, G., Muñoz, C., Rocha, C., 2010. Rewriting logic semantics of a plan execution language. CoRR abs/1002.2872.

[14] Estlin, T., Jónsson, A., Păsăreanu, C., Simmons, R., Tso, K., Verna, V., 2006. Plan Execution Interchange Language (PLEXIL). Technical Memorandum TM-2006-213483, NASA.

[15] Guernic, P. L., Gautier, T., Borgne, M. L., Maire, C. L., 1991. Programming real-time applications with SIGNAL. In: Proceedings of the IEEE, Volume 79(9). pp. 1321–1336.

[16] Harbusch, K., Poller, P., 1994. Structural translation with synchronous rewriting systems. Technical Report RT-94-01, Université Paris 7, Paris, France.

[17] Harel, D., June 1987. Statecharts: A visual formalism for complex systems. Science of Computer Programming 8 (3), 231–274.

[18] Heninger, K., 1980. Specifying software requirements for complex systems: New techniques and their application. IEEE Transactions on Software Engineering 6 (1), 2–13.

[19] Lucanu, D., 2009. Strategy-based rewrite semantics for membrane systems preserves maximal concurrency of evolution rule actions. Electronic Notes in Theoretical Computer Science 237, 107–125.

[20] Lüttgen, G., Beeck, M., Cleaveland, R., November 2000. A compositional approach to statecharts semantics. In: ACM SIGSOFT 8th Intl. Symposium on Foundations of Software Engineering (FSE 2000). Vol. 25,6 of

ACM Software Engineering Notes. ACM Press, San Diego, California, pp. 120–129.

[21] Lüttgen, G., Mendler, M., 2002. Axiomatizing an algebra of step reactions for synchronous languages. In: Brim, L., Jancar, P., Kretínský, M., Kucera, A. (Eds.), CONCUR. Vol. 2421 of Lecture Notes in Computer Science. Springer, pp. 386–401.

[22] Meseguer, J., 1992. Conditional rewriting logic as a unified model of concurrency. Theoretical Computer Science 96 (1), 73–155.

[23] Meseguer, J., 2000. Rewriting logic and Maude: a wide-spectrum semantic framework for object-based distributed systems. In: Smith, S. F., Talcott, C. L. (Eds.), FMOODS. Vol. 177 of IFIP Conference Proceedings. Kluwer, pp. 89–.

[24] Meseguer, J., Rosu, G., 2007. The rewriting logic semantics project. Theoretical Computer Science 373 (3), 213–237.

[25] Owen, R., Giorgio, S., 1996. Synchronous models of language. In: Proceedings of the 34th annual meeting on Association for Computational Linguistics. Association for Computational Linguistics, Morristown, NJ, USA, pp. 116–123.

[26] Owre, S., Rushby, J., Shankar, N., Jun. 1992. PVS: A prototype verification system. In: Kapur, D. (Ed.), 11th International Conference on Automated Deduction (CADE). Vol. 607 of Lecture Notes in Artificial Intelligence. Springer-Verlag, Saratoga, NY, pp. 748–752.

[27] Plotkin, G. D., 2004. A structural approach to operational semantics. Journal of Logic and Algebraic Programming 60-61, 17–139.

[28] Sampath, P., Rajeev, A. C., Ramesh, S., Shashidhar, K. C., 2007. Testing model-processing tools for embedded systems. In: IEEE Real-Time and Embedded Technology and Applications Symposium. IEEE Computer Society, pp. 203–214.

[29] Serbanuta, T., Stefanescu, G., Rosu, G., 2008. Defining and executing P systems with structured data in k. In: Corne, D. W., Frisco, P., Paun, G., Rozenberg, G., Salomaa, A. (Eds.), Workshop on Membrane Computing. Vol. 5391 of Lecture Notes in Computer Science. Springer, pp. 374–393.

[30] Tardieu, O., 2007. A deterministic logical semantics for pure esterel. ACM Transactions on Programming Languages and Systems 29 (2), 8.

[31] Viry, P., 2002. Equational rules for rewriting logic. Theoretical Computer Science 285, 487–517.