Science of Computer Programming 92 (2014) 211–228

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Synchronous set relations in rewriting logic

Camilo Rocha [a,*], César Muñoz [b]

[a] Escuela Colombiana de Ingeniería, Bogotá, D.C., Colombia
[b] NASA Langley Research Center, Hampton, VA, USA

ARTICLE INFO

Article history:
Received 10 April 2012
Received in revised form 3 July 2013
Accepted 17 July 2013
Available online 19 August 2013

Keywords:
Synchronous set relations
Synchronous semantics
Rewriting logic
Formal simulation and verification
PLEXIL

ABSTRACT

This paper presents a mathematical foundation and a rewriting logic infrastructure for the execution and property verification of synchronous set relations. The mathematical foundation is given in the language of abstract set relations. The infrastructure, which is written in the Maude system, enables the synchronous execution of a set relation provided by the user. By using the infrastructure, algorithm verification techniques such as reachability analysis and model checking, already available in Maude for traditional *asynchronous* rewriting, are automatically available to synchronous set rewriting. In this way, set-based synchronous languages and systems such as those built from agents, components, or objects can be naturally specified and simulated, and are also amenable to formal verification in the Maude system. The use of the infrastructure and some of its Maude-based verification capabilities are illustrated with an executable operational semantics of the Plan Execution Interchange Language (PLEXIL), a synchronous language developed by NASA to support autonomous spacecraft operations.

1. Introduction

Synchronous languages are extensively used in embedded and critical applications and synchronous set relations provide a natural model for describing their operational semantics. In a previous work, Rocha et al. [16] have proposed a *serialization procedure* for simulating the execution of synchronous set-based relations by *asynchronous* term rewriting. The synchronous execution of a set relation is a parallel reduction, where the terms to be reduced in parallel are selected according to some strategy. Such a serialization procedure was *manually* implemented to provide the rewriting logic semantics of the Plan Execution Interchange Language (PLEXIL) [7], a synchronous plan execution language developed by NASA to support spacecraft automation [8]. Defining the synchronous semantic relation of a programming language, such as PLEXIL, is generally difficult, time consuming, and error-prone. Moreover, manually implementing the generic serialization procedure proposed in [16] adds complexity to an already challenging task. This paper presents new contributions in the direction of supporting more general synchronous set relations and a computer infrastructure that *automatically* implements a serialization procedure for a general class of synchronous set relations.

In particular, the work presented in this paper extends the development in [16] in two significant ways. First, it generalizes the theoretical foundations of synchronous set relations by extending the notion of strategy to include a larger set of synchronous transformations such as those that enable non-deterministic ways to select elements in a set to be synchronously reduced. Strategies, as defined in [16], only enable deterministic ways to select elements in a set to be synchronously reduced. Deterministic strategies are appropriate in the context of PLEXIL's operational semantics is deterministic. However, they may be too restrictive for encoding the semantics of non-deterministic synchronous languages.

* Corresponding author.
E-mail addresses: camilo.rocha@escuelaing.edu.co (C. Rocha), cesar.a.munoz@nasa.gov (C. Muñoz).

0167-6423/$ – see front matter
http://dx.doi.org/10.1016/j.scico.2013.07.008

Moreover, non-deterministic strategies are fundamental to the development of new techniques for symbolic reachability analysis of synchronous languages [13]. Even in the case of deterministic languages, such as PLEXIL, non-determinism is introduced by the inherent nature of symbolic variables.

The second major contribution with respect to [16] is a computer infrastructure that implements, on-the-fly, a serialization procedure for a synchronous language provided by the user. This infrastructure is written in Maude [4], a high-performance rewriting logic engine. The infrastructure provides syntactic constructs for defining synchronous relations and their, possibly non-deterministic, reduction strategies. Using Maude's reflective capabilities, these constructs are translated into asynchronous rewriting systems that soundly and completely simulate the synchronous relations provided as inputs.

These two contributions allow for simpler and more succinct language specifications, and more general synchronous set relations. Also, since programming languages tend to evolve constantly, this infrastructure can help language designers to focus exclusively on the synchronous semantic design without shifting their attention to details in the serialization procedure implementation.

The rest of this paper is organized as follows. Section 2 presents, in an abstract setting, definitions that are relevant to synchronous set relations. These definitions properly extend those given in [16]. Formally, a synchronous set relation is defined as the *synchronous closure* of an *atomic* relation with a given *strategy*. Two sets are synchronously related for a particular strategy if the first set can be transformed into the second set by parallel atomic transformations according to the strategy. The strategy selects sets of redexes that can be synchronously transformed.

Section 3 describes Maude's rewriting logic, which provides the formal framework for the development presented in this paper. Maude is a reflective language and rewriting logic system that supports both equational and rewrite theories. Maude implements set rewriting, i.e., rewriting modulo axioms such as associativity, commutativity, and identity. These capabilities are well-suited for set-based and multiset-based concurrent systems.

Section 4 describes the proposed infrastructure, which comprises generic sorts and terms, algebraic properties of generic datatypes, and functions that support the on-the-fly implementation of a serialization procedure. The algebraic datatypes in this specification can be instantiated with the specific constructs of a set-based synchronous language provided by the user. These constructs are translated into labeled conditional rewriting rules that, under some conditions, becomes a formal interpreter for the synchronous semantic relation of the language.

Section 5 describes how the proposed infrastructure can be used for defining the executable semantics of a simple synchronous language with arithmetic expressions. Code snippets of the Maude specification are used to illustrate, in a concrete way, how the infrastructure is used.

As a more involved example, Section 6 presents a rewriting logic semantics of PLEXIL implemented on top of the infrastructure developed in Maude. This semantics comprises a significant subset of the language that includes Boolean and arithmetic expressions, and all language specific predicates. As a direct advantage of using the infrastructure, all commands in Maude for rewrite specifications such as its rewrite and search commands, and formal verification tools such as Maude's LTL Model Checker, are available for analyzing properties of programs in PLEXIL.

The infrastructure in Maude and the examples presented in this paper are available from http://shemesh.larc.nasa.gov/people/cam/PLEXIL.

2. Abstract synchronous set relations

This section introduces the concepts of abstract set relations used in this paper.

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 there exists $B \in \mathcal{U}$ such that the *pair* $\langle A; B\rangle \in \rightarrow$. The expressions $A \rightarrow B$ and $A \not\rightarrow B$ denote, respectively, $\langle A; B\rangle \in \rightarrow$ and $\langle A; B\rangle \notin \rightarrow$. The *identity* relation and *reflexive-transitive closure* of $\rightarrow$ are defined as usual and denoted $\rightarrow^0$ and $\rightarrow^*$, respectively.

Henceforth, it is assumed that $\mathcal{U}$ is the family of all *nonempty* finite sets over an abstract and possibly infinite set T, i.e., $\mathcal{U} \subseteq \wp(T)$, $\emptyset \notin \mathcal{U}$, and if $A \in \mathcal{U}$ then $\mathrm{card}(A) \in \mathbb{N}$ (therefore, $\rightarrow$ is a binary relation on finite sets of T). The elements of $\mathcal{U}$ are called *terms* in this section. The elements of T will be denoted by lowercase letters $a, b, \ldots$. 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 $\{d, e\}$, notation $A, B \rightarrow B, D$ denotes $\{a, b, c, d\} \rightarrow \{c, d, e\}$.

The *parallel* relation $\rightarrow^{\parallel}$ of $\rightarrow$ is the relation defined as the parallel closure of $\rightarrow$, i.e., 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 $B = (A \setminus \bigcup_{1 \leqslant i \leqslant n} A_i) \cup \bigcup_{1 \leqslant i \leqslant n} B_i$.

This paper focuses on synchronous set relations. The synchronous relation of an abstract set relation $\rightarrow$ is defined as a subset of the parallel closure of $\rightarrow$, where a given strategy selects elements from $\rightarrow$. Formally, a $\rightarrow$*-strategy* is a function s that maps an element $A \in \mathcal{U}$ into a set $s(A) \subseteq \wp(\rightarrow)$ such that if $\{\langle A_1; B_1\rangle, \ldots, \langle A_n; B_n\rangle\} \in s(A)$, then $A_i \subseteq A$ and $A_i \rightarrow B_i$, for $1 \leqslant i \leqslant n$, and $A_1, \ldots, A_n$ are pairwise disjoint.

Definition 1 *(Synchronous relation).* Let s be a $\rightarrow$-strategy. 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 $B = (A \setminus \bigcup_{1 \leqslant i \leqslant n} A_i) \cup \bigcup_{1 \leqslant i \leqslant n} B_i$, where $\{\langle A_1; B_1\rangle, \ldots, \langle A_n; B_n\rangle\} \in s(A)$.