Guarded Terms for Rewriting Modulo SMT

Kyungmin Bae[1] and Camilo Rocha[2]

[1] Pohang University of Science and Technology, Pohang, South Korea
[2] Pontificia Universidad Javeriana, Cali, Colombia

Abstract. Rewriting modulo SMT is a novel symbolic technique to model and analyze infinite-state systems that interact with a nondeterministic environment. It seamlessly combines rewriting modulo equational theories, SMT solving, and model checking. One of the main challenges of this technique is to cope with the symbolic state-space explosion problem. This paper presents guarded terms, an approach to deal with this problem for rewriting modulo SMT. Guarded terms can encode many symbolic states into one by using SMT constraints as part of the term structure. This approach enables the reduction of the symbolic state space by limiting branching due to concurrent computation, and the complexity and size of constraints by distributing them in the term structure. A case study of an unbounded and symbolic priority queue illustrates the approach.

1 Introduction

The specification and verification effort in component-based software engineering can be improved using symbolic approaches. They can make available symbolic analysis techniques and tools with the promise of taming the many complexities involved in component-based systems, including real-time and cyber-physical systems. Symbolic techniques can be used to verify the functionality offered by software components for *any* possible input and communication interleaving. Rewriting modulo SMT [19] is a novel symbolic technique to model and analyze infinite-state systems that interact with a nondeterministic environment. It is a symbolic specification and verification method for rewriting logic [14], a general logical framework in which many component-based systems, such as AADL [3] and Ptolemy II [4], can be naturally specified [15].

Rewriting modulo SMT seamlessly combines rewriting modulo equational theories, SMT solving, and model checking. In rewriting modulo SMT, states are represented as symbolic constrained terms $(t\,;\phi)$ with t a term with variables ranging over the *built-ins* (the sorts handled by the SMT solver) and ϕ a SMT-solvable formula. State transitions are symbolic rewrite steps between constrained terms. In one rewrite step from $(t_1\,;\phi_1)$ to $(t_2\,;\phi_2)$, possibly infinitely many instances of t_1 can be rewritten to instances of t_2; namely, those ground instances of t_1 that satisfy the constraint ϕ_1 result in a ground instance of t_2 satisfying the constraint ϕ_2. In general, a n-step symbolic rewrite from $(t_1\,;\phi_1)$ to $(t_n\,;\phi_n)$ captures *all* possible traces with n transitions from ground instances of t_1 satisfying ϕ_1 to *all* ground instances of t_n satisfying ϕ_n. By being complete, the symbolic rewrite relation will capture any ground trace satisfying these conditions, if any exists. This is one of the reasons why rewriting modulo SMT is well-suited for symbolically proving (or disproving) safety properties of rewrite theories.

Rewriting modulo SMT can be used to analyze existential reachability properties of infinite-state systems such as invariant and deadlock freedom properties. Moreover, it can be efficiently implemented by performing matching (instead of the more costly unification) for the term t and querying the SMT solver for the satisfiability of ϕ, for a given symbolic state $(t\,;\phi)$ at each rewrite step. However, the effective application of this technique comes with new challenges in terms of scalability: namely, the symbolic state-space explosion problem. For instance, the symbolic semantics of PLEXIL [18, 19] – a synchronous language developed by NASA to support autonomous spacecraft operations –, which uses rewriting modulo SMT, is nondeterministic despite the fact that its ground counterpart is deterministic [12]. Therefore, in the symbolic rewriting logic semantics of PLEXIL, the state space can grow very large and the constraints become complex, making the formal analysis task time-consuming or unfeasible.

This paper presents *guarded terms*, a technique with the potential to reduce the symbolic state space and the complexity of constraints in the rewriting modulo SMT approach. A guarded term can be seen as a choice operator that is part of the term structure in a constrained term. Guarded terms generalize constrained terms by allowing t in a symbolic state $(t\,;\phi)$ to have, e.g., a guarded term $u_1|_\psi \vee u_2|_{\neg\psi}$ as a subterm. This means that when the constraint $\phi \wedge \psi$ is satisfiable, $(t\,;\phi)$ represents those ground instances of t in which the subterm $u_1|_\psi \vee u_2|_{\neg\psi}$ is replaced by u_1. Analogously, when $\phi \wedge \neg\psi$ is satisfiable, $(t\,;\phi)$ represents those ground instances of t in which the subterm $u_1|_\psi \vee u_2|_{\neg\psi}$ is replaced by u_2. Therefore, the guarded term can actually encode both alternatives without the need for two constrained terms. The greater potential of guarded terms can better be seen when they are composed in parallel or nested, thus enabling the succinct encoding of several constrained terms into one guarded term.

Guarded terms are particularly useful in rewriting modulo SMT in many situations for reducing: (i) the symbolic state space by implicitly encoding branching in the term structure, and (ii) the complexity and size of constraints by distributing them in several parts of the term structure. The effectiveness of the approach is illustrated with a case study of an unbounded and symbolic priority queue that, with the help of guarded terms, enables automatic reachability analysis of the CASH scheduling algorithm [7].

The rest of the paper is organized as follows. Section 2 overviews rewriting logic and rewriting modulo SMT. Section 3 presents guarded terms and their main properties. Section 4 introduces the case study on the CASH algorithm. Finally, Section 5 discusses related work and presents some concluding remarks. The examples used throughout the paper, and the proofs omitted in Section 3 can be found in [5].

2 Rewriting Logic and Rewriting Modulo SMT in a Nutshell

This section briefly explains order-sorted rewriting logic and rewriting modulo SMT, summarizing Sections 2–5 in [19]. Rewriting logic [14] is a semantic framework that unifies a wide range of models of concurrency. Maude [11] is a language and tool to support the formal specification and analysis of concurrent systems in rewriting logic. Rewriting modulo SMT [19] is a symbolic technique to model and analyze reachability properties of infinite-state systems in rewriting logic, that can be executed in Maude by querying decision procedures available from SMT technology.

2.1 Order-Sorted Rewrite Theories

An *order-sorted signature* Σ is a tuple $\Sigma{=}(S, \leq, F)$ with a finite poset of sorts $(S, \leq)$ and set of function symbols F typed with sorts in S, which can be subsort-overloaded. The *set of function symbols of sort* $w \in S^*$ *in* Σ is denoted by Σ_w. The binary relation $\equiv_\leq$ denotes the equivalence relation $(\leq \cup \geq)^+$ generated by $\leq$ on S and its point-wise extension to strings in S^*. The expression $[s]$ denotes the connected component of s, that is, $[s] = [s]_{\equiv_\leq}$. A *top sort* in Σ is a sort $s \in S$ such that for all $s' \in [s]$, $s' \leq s$.

For $X = \{X_s\}_{s \in S}$ an S-indexed family of disjoint variable sets with each X_s countably infinite, the *set of terms of sort* s and the *set of ground terms of sort* s are denoted, respectively, by $T_\Sigma(X)_s$ and $T_{\Sigma,s}$; similarly, $T_\Sigma(X)$ and T_Σ denote, respectively, the set of terms and the set of ground terms.

A *substitution* is an S-indexed mapping $\theta : X \longrightarrow T_\Sigma(X)$ that is different from the identity only for a finite subset of X, and such that $\theta(x) \in T_\Sigma(X)_s$ if $x \in X_s$, for any $x \in X$ and $s \in S$. A substitution θ is called *ground* if and only if $\theta(x) \in T_\Sigma$ or $\theta(x) = x$ for any $x \in X$. The application of a substitution θ to a term t is denoted by θt and the composition (in diagrammatic order) of two substitutions θ_1 and θ_2 is denoted by $\theta_1\theta_2$, so that $\theta_1\theta_2 t$ denotes $\theta_1(\theta_2 t)$.

A *rewrite theory* is a tuple $\mathcal{R} = (\Sigma, E \uplus B, R)$ with: (i) $(\Sigma, E \uplus B)$ an order-sorted equational theory with signature Σ, E a set of equations over T_Σ, and B a set of structural axioms – disjoint from the set of equations E – over T_Σ for which there is a finitary matching algorithm (e.g., associativity, commutativity, and identity, or combinations of them); and (ii) R a finite set of rewrite rules over T_Σ.

Intuitively, $\mathcal{R}$ specifies a concurrent system whose states are elements of the set $T_{\Sigma/E\uplus B}$ of Σ-terms modulo $E \uplus B$ and whose concurrent transitions are axiomatized by the rules R according to the inference rules of rewriting logic [6]. In particular, for $t, u \in T_\Sigma$ representing states of the concurrent system described by $\mathcal{R}$, a transition from t to u is captured by a formula of the form $t \rightarrow_\mathcal{R} u$; the symbol $\rightarrow_\mathcal{R}$ denotes the binary rewrite relation induced by R over $T_{\Sigma/E\uplus B}$ and $\mathcal{T}_\mathcal{R} = (T_{\Sigma/E\uplus B}, \rightarrow_\mathcal{R})$ denotes the *initial reachability model* of $\mathcal{R}$. The expressions $\mathcal{T}_{\Sigma/E\uplus B}$ and $=_{E\uplus B}$ denote, respectively, the *initial algebra* of $(\Sigma, E \uplus B)$ and the congruence induced by $(\Sigma, E \uplus B)$ on Σ-terms.

Example 1. Consider a system with states in the top sort *Conf* of the form $C_1 \parallel C_2$, with C_1 and C_2 multisets of integer numbers. Each state has sort *Conf*, an integer has sort *Int*, and a multiset of integers has sort *Channel*. Multiset union is denoted by juxtaposition, and it is associative, commutative, and has identity *none* (which denotes the empty collection). Integer number addition and the "less-than" total order relation on integers are denoted with the usual function symbols.

The symbol $\overset{n}{=}$ represents the "modulo n congruence" binary relation over integers (i.e., a shorthand for $x \equiv y \mod n$). The system consists of the following three rewrite rules, where I_1, I_2 range over integers and C_1, C_2 over multisets of integers:

$$I_1\ I_2\ C_1 \parallel C_2 \ \rightarrow\ I_1\ C_1 \parallel (I_1 + I_2 + 1)\ C_2 \quad \textbf{if}\ \ 0 < I_1 \wedge 0 < I_2 \wedge (I_1 + I_2 \overset{3}{=} 0)$$

$$I_1\ I_2\ C_1 \parallel C_2 \ \rightarrow\ I_1\ C_1 \parallel C_2 \quad \textbf{if}\ \ 0 < I_1 \wedge 0 < I_2 \wedge \neg(I_1 + I_2 \overset{3}{=} 0)$$

$$I_1 \parallel I_2\ C_2 \ \rightarrow\ \mathit{none} \parallel I_2\ C_2 \quad \textbf{if}\ \ 0 < I_1 \wedge 0 < I_2 \wedge (I_1 + I_2 \overset{17}{=} 0)$$

In this system, integers move from the left channel to the right channel. By the first rule, an intenger I_2 is removed from the left channel and the integer $I_1 + I_2 + 1$ is added to the right channel, for any I_1 and I_2 in the left channel, whenever I_1 and I_2 are at least 1, and $I_1 + I_2$ is a multiple of 3. By the second rule, an integer I_2 is removed from the left channel, for any I_1 and I_2 in the left channel, whenever I_1 and I_2 are at least 1, and $I_1 + I_2$ is not a multiple of 3. By the third rule, an integer I_1 is removed from the left channel, for any I_1 in the left channel and I_2 in the right channel, whenever I_1 is the only number in the left channel, I_1 and I_2 are at least 1, and $I_1 + I_2$ is a multiple of 17.

This system can be executed by rewriting in Maude as follows. Given a rewrite rule $l \to r$ **if** *cond*, with $l, r \in T_\Sigma(X)_{Conf}$, a ground term $t \in T_{\Sigma,Conf}$ rewrites to a ground term $u \in T_{\Sigma,Conf}$ (i.e., $t \to_\mathcal{R} u$) if and only if there is a ground substitution σ such that t and u are respectively substitution instances of l and r modulo $=_{E \uplus B}$ (i.e., $\sigma l =_{E \uplus B} t$ and $\sigma r =_{E \uplus B} u$), and the condition $\sigma cond$ holds. For example, by the first rule and $\sigma = \{I_1 \mapsto 1, I_2 \mapsto 2, C_1 \mapsto 3\ 4, C_2 \mapsto none\}$, follows 1 2 3 4 || *none* $\to_\mathcal{R}$ 1 3 4 || 4.

By being executable in Maude, automatic state-space search capabilities can be used, e.g., to identify potential deadlocks in this system. A Maude search command searches for states that are reachable from a *ground* initial state and match the search pattern and satisfy the search condition. Starting from the initial state 1 2 3 4 || *none*, the following search command checks if there is a deadlock state with a nonempty left channel that can not be further rewritten (and this command finds no solution):

```
search [1]   1 2 3 4 || none   =>!   C1 || C2   such that C1 =/= none .
```

In this command, `[1]` specifies the maximum number (in this case, 1) of solutions that the command should return, and `=>!` means that only non-reducible states with respect to $\to_\mathcal{R}$ are to be considered as solutions to the query.

2.2 Rewriting Modulo SMT

Rewriting modulo SMT is illustrated using a symbolic version of Example 1. In this version, a symbolic state is given by a *constrained term* $(t\,;\phi)$, with $t \in T_\Sigma(X_0)_{Conf}$ and $\phi \in QF_{\Sigma_0}(X_0)$, where $X_0 \subseteq X$ denotes the set of variables ranging over the built-ins, $\Sigma_0 \subseteq \Sigma$ denotes the signature of the built-in sorts, and $QF_{\Sigma_0}(X_0)$ denotes the set of quantifier-free formulas over Σ_0 with variables in X_0.

Example 2. In the symbolic version of the system, the built-in sorts are *Bool* and *Int*, and the non-built-in sorts are *Channel* and *Conf*. There are three rewrite rules with variables I_1, I_2 of sort *Int*, variables C_1, C_2 of sort *Channel*, and ϕ_1, ϕ_2 of sort *Bool*:

$$(I_1\ I_2\ C_1 \parallel C_2\,;\phi_1) \to (I_1\ C_1 \parallel (I_1 + I_2 + 1)\ C_2\,;\phi_1 \wedge \phi_2)$$
$$\textbf{if}\quad \phi_2 := 0 < I_1 \ \wedge\ 0 < I_2 \ \wedge\ (I_1 + I_2 \overset{3}{=} 0) \ \wedge\ sat(\phi_1 \wedge \phi_2)$$

$$(I_1\ I_2\ C_1 \parallel C_2\,;\phi_1) \to (I_1\ C_1 \parallel C_2\,;\phi_1 \wedge \phi_2)$$
$$\textbf{if}\quad \phi_2 := 0 < I_1 \ \wedge\ 0 < I_2 \ \wedge\ \neg(I_1 + I_2 \overset{3}{=} 0) \ \wedge\ sat(\phi_1 \wedge \phi_2)$$

$$(I_1 \parallel I_2\ C_2\,;\phi_1) \to (none \parallel I_2\ C_2\,;\phi_2)$$
$$\textbf{if}\quad \phi_2 := 0 < I_1 \ \wedge\ 0 < I_2 \ \wedge\ (I_1 + I_2 \overset{17}{=} 0) \ \wedge\ sat(\phi_1 \wedge \phi_2)$$

These rules are similar to the ones in Example 1. The key observation in this version is that conditions are treated as constraints, accumulated in the system state, and queried for satisfiability with the help of the function *sat* (an interface to the SMT-solver to check whether ϕ is satisfiable or not). A *matching condition* [11] of the form $t := u$ is a syntactic variant of the equational condition $t = u$ mathematically interpreted as an ordinary equation. Operationally, a matching condition behaves like a 'let' construct in functional programming languages so that $t := u$ introduces t in the rule as the result of reducing u to canonical form with the oriented equations modulo the axioms.

Definition 1. *Given a rewrite rule* $(l\,;\phi_l) \rightarrow (r\,;\phi_r)$ **if** ϕ*, with* $l, r \in T_\Sigma(X)_{Conf}$ *and* $\phi \in QF_{\Sigma_0}(X_0)$*, a constrained term* $(t\,;\phi_t) \in T_\Sigma(X_0)_{Conf} \times QF_{\Sigma_0}(X_0)$ *symbolically rewrites to a constrained term* $(u\,;\phi_u) \in T_\Sigma(X_0)_{Conf} \times QF_{\Sigma_0}(X_0)$ *(denoted by* $(t\,;\phi_t) \rightsquigarrow_{\mathcal{R}} (u\,;\phi_u)$*) if and only if there is a substitution* θ *such that:*

(a) $\theta l =_{E\uplus B} t$ *and* $\theta r =_{E\uplus B} u$*,*
(b) $\mathcal{T}_{\Sigma/E\uplus B} \models (\phi_l \wedge \theta\phi) \Leftrightarrow \phi_u$*, and*
(c) ϕ_u *is* $\mathcal{T}_{\Sigma/E\uplus B}$*-satisfiable.*

The symbolic relation $\rightsquigarrow_{\mathcal{R}}$ is defined as a topmost rewrite relation, where all rewrites take place at the top of the term, induced by R modulo $E \uplus B$ on $T_\Sigma(X_0)$ with extra bookkeeping of constraints.

Condition (a) can be solved by matching as in the definition of $\rightarrow_{\mathcal{R}}$ above. Condition (b) can be met by setting ϕ_u to be $\phi_l \wedge \theta\phi$, as in the above matching conditions. However, Condition (c) cannot – in general – be dealt with by rewriting. The reason is that such a condition can be an inductive theorem of $\mathcal{T}_{\Sigma/E\uplus B}$. Instead, these conditions are checked with the help of decision procedures available from an SMT solver via the function *sat*. Observe that, up to the choice of the semantically equivalent φ_u for which a fixed strategy such as the one suggested above can be assumed, the symbolic relation $\rightsquigarrow_{\mathcal{R}}$ is deterministic in the sense of being determined by the rule and the substitution θ (here it is assumed that variables in the rules are disjoint from the ones in the target terms). The reader is referred to [19] for details about rewriting modulo SMT.

Example 3. Consider the constrained term $(A\ B\ C\ D \parallel \mathit{none}\,;\mathit{true})$, with four variables $A, B, C, D \in X_{Int}$. Notice that $(A\ B\ C\ D \parallel \mathit{none}\,;\mathit{true})$ symbolically rewrites in one step to $(A\ C\ D \parallel A + B + 1; 0 < A \wedge 0 < B \wedge (A + B \overset{3}{=} 0))$.

Rewriting modulo SMT can be used for solving existential reachability goals in the initial model $\mathcal{T}_{\mathcal{R}}$ of a rewrite theory $\mathcal{R}$ modulo built-ins $\mathcal{E}_0$. In general, for any constrained term $(t\,;\phi)$ where t is a state term with sort *Conf* and ϕ is a constraint, $[\![t]\!]_\phi$ is the *denotation* of $(t\,;\phi)$ consisting of all ground instances of t that satisfy ϕ; formally $[\![t]\!]_\phi = \left\{t' \in T_{\Sigma,Conf} \mid\ (\exists\sigma : X \longrightarrow T_\Sigma)\, t' = \sigma t \wedge \mathcal{T}_{\Sigma/E\uplus B} \models \sigma\phi\right\}$. The type of *existential reachability* question that rewriting modulo SMT can solve can now be formulated: are there some states in $[\![t]\!]_\phi$ from which is possible to reach some state in $[\![u]\!]_\psi$? Answering this question can be especially useful for symbolically proving or disproving safety properties of $\mathcal{R}$, such as inductive invariants or deadlock freedom of $\mathcal{T}_{\mathcal{R}}$: when $[\![u]\!]_\psi$ is a set of *bad* states, the idea is to know whether reaching a state in $[\![u]\!]_\psi$ is possible.

Consider the following existential query, where $\mathcal{R}$ is the rewrite theory presented in Example 1 (i.e., the one without symbolic constraints):

$$\mathcal{T}_{\mathcal{R}} \models (\exists I, C_1, C_2)\ I\,(I+1)\,(I+2)\,(I+3) \parallel \mathit{none} \ \rightarrow^*_{\mathcal{R}}\ C_1 \parallel C_2 \ \wedge\ I > 0 \\ \wedge\ C_1 \neq \mathit{none} \ \wedge\ \text{“}C_1 \parallel C_2 \text{ is } \rightarrow_{\mathcal{R}}\text{-irreducible”}.$$

This query asks if it is possible to find an initial state consisting of four consecutive positive integers in the left channel and no number on the right channel that leads to an irreducible state where there are still numbers in the left channel. This is the same type of deadlock freedom property discussed before, namely, the one in which bad states are those irreducible ones in which the channel has at least one number.

Answering this query in the negative proves that $\mathcal{R}$ is deadlock free for any initial state satisfying the initial pattern. Since I ranges over an infinite domain, this question cannot be solved directly via rewriting and would require, e.g., inductive reasoning over $\rightarrow_{\mathcal{R}}$. However, the following Maude *search* command can be issued in Maude to find a proof (or a counterexample) for the symbolic rewrite relation $\rightsquigarrow_{\mathcal{R}}$:

```
search [1] { I  I+1 I+2 I+3 || none , I > 0 }
       =>! { C1 || C2, Phi } such that C1 =/= none .
```

Executed as it is, this command times out after 5 minutes. As shown in the next section, with the help of guarded terms, the exact same search command terminates in less than 1 second without finding a witness, therefore proving the deadlock freedom of $\mathcal{T}_{\mathcal{R}}$ from states satisfying the pattern $I\ (I+1)\ (I+2)\ (I+3) \parallel \mathit{none}$.

3 Guarded Terms

Guarded terms generalize constrained terms with heterogeneous patterns and nested structures. Guarded terms succinctly represent various terms by choices of subterms that are guarded by a constraint. These subterms represent possible realizations, namely, those instances in which the constraints are true. The proofs of lemmas and theorems presented in this section can be found in [5].

3.1 Syntax

Consider the constrained terms $(t_1 ; \phi_1), \ldots, (t_n ; \phi_n)$ with the terms $t_1, \ldots, t_n$ of the same sort. First, a guarded term can be built by combining these constrained terms in parallel as $(t_1 ; \phi_1) \vee \cdots \vee (t_n ; \phi_n)$, semantically representing the union of the sets $[\![t_1]\!]_{\phi_1}, \ldots, [\![t_n]\!]_{\phi_n}$ of ground terms. Second, guarded terms can be *nested* so that the terms t_i may include guarded terms as subterms. For example, if f and g are unary and binary function symbols, respectively, then the term $f((t_1 ; \phi_1) \vee (f(g((t_3 ; \phi_3), t_4)) ; \phi_2))$ is a guarded term. The guarded subterm $g((t_3 ; \phi_3), t_4)$ encodes the ground instances $\theta g(t_3, t_4)$ in which $\theta\phi_3$ is true, for any ground substitution θ.

In order to avoid confusion between the syntax of constrained terms and guarded terms, $(t ; \phi)$ will be written as $t|_\phi$. With this convention, the above-mentioned guarded terms can be written as $t_1|_{\phi_1} \vee \cdots \vee t_n|_{\phi_n}$ and $f(t_1|_{\phi_1} \vee f(g(t_3|_{\phi_3}, t_4))|_{\phi_2})$. The syntax of guarded terms is formally presented in Definition 2.

Definition 2. *Given a signature Σ, the set of* guarded terms of sort s *is the smallest set $\overline{T}_\Sigma(X)_s$ satisfying the following conditions:*

- $T_\Sigma(X)_s \subseteq \overline{T}_\Sigma(X)_s$;
- $f(t_1, \ldots, t_n) \in \overline{T}_\Sigma(X)_s$, *if* $f \in \Sigma_{s_1 \ldots s_n, s}$ *and* $t_i \in \overline{T}_\Sigma(X)_{s_i}$ *for* $1 \leq i \leq n$;
- $t|_\phi \in \overline{T}_\Sigma(X)_s$, *if* $t \in \overline{T}_\Sigma(X)_s$ *and* ϕ *is a constraint; and*
- $t_1 \vee t_2 \in \overline{T}_\Sigma(X)_s$ *for guarded terms* $t_1, t_2 \in \overline{T}_\Sigma(X)_s$ *of sort* s.

A signature $G(\Sigma)$ for guarded terms can be built by adding new sorts and function symbols to the underlying signature Σ as follows:

- a new sort $g(s)$ with a subsort relation $s < g(s)$ for each sort s;
- an operator $f : g(s_1) \cdots g(s_m) \longrightarrow g(s)$ for each operator $f : s_1 \cdots s_m \longrightarrow s$;
- a guard operator $\_|\_ : g(s) \times Bool \longrightarrow g(s)$ for each sort s; and
- an union operator $\_\vee\_ : g(s) \times g(s) \longrightarrow g(s)$ for each sort s,

where constraints have sort *Bool* (in the built-in subsignature $\Sigma_0 \subseteq \Sigma$) and sort $g(s_1)$ is a subsort of $g(s_2)$ whenever s_1 is a subsort of s_2. By construction, a guarded term $u \in \overline{T}_\Sigma(X)_s$ of sort s is a $G(\Sigma)$-term of sort $g(s)$.

Lemma 1. *Given an order-sorted signature Σ, a guarded term u of sort s in $\overline{T}_\Sigma(X)$ is a term of sort $g(s)$ in $T_{G(\Sigma)}(X)$, and vice versa.*

Example 4. In the example in Section 2, a term $I + J + 1$ is added to the right-hand side of the first rule if the condition $I + J \overset{3}{=} 0$ holds. A one-step symbolic rewrite step with this rule involves two cases: either $I + J \overset{3}{=} 0$ holds or not. Thus, n-rewrite steps with this rule can yield 2^n different symbolic states. For example, from the term $A\ B\ C \parallel \mathit{none}$, where A, B, C are integer variables, symbolic rewriting up to three steps generates:

$$
\begin{array}{ll}
A\ B\ C \parallel \mathit{none} & A\ B\ C \parallel (A + B + 1)\ (B + C + 1) \\
A\ B\ C \parallel (A + B + 1) & A\ B\ C \parallel (A + B + 1)\ (C + A + 1) \\
A\ B\ C \parallel (B + C + 1) & A\ B\ C \parallel (B + C + 1)\ (C + A + 1) \\
A\ B\ C \parallel (C + A + 1) & A\ B\ C \parallel (A + B + 1)\ (B + C + 1)\ (C + A + 1).
\end{array}
$$

A set of terms can be succinctly encoded as a guarded term. Consider guarded terms of the form $\mathit{elm}(I, J)$ defined as $\mathit{elm}(I, J) = I + J + 1|_{I+J \overset{3}{=} 0} \vee \emptyset|_{I+J \overset{3}{\neq} 0}$. The guarded term $\mathit{elm}(I_1, J_1)\ \mathit{elm}(I_2, J_2) \cdots \mathit{elm}(I_N, J_N)$ can encode the 2^n symbolic states that can be reached in n-rewrite steps with respect to the satisfaction of the n conditions $I_i + J_i \overset{3}{=} 0$, for $1 \leq i \leq N$. For example, the set of terms reachable from $A\ B\ C \parallel \mathit{none}$ in three steps can be encoded as: $A\ B\ C \parallel \mathit{elm}(A, B)\ \mathit{elm}(B, C)\ \mathit{elm}(C, A)$.

3.2 Semantics

A guarded term u represents a (possibly infinite) set of ground terms, denoted by $[\![u]\!]$. Intuitively, $[\![u|_\phi]\!]$ is the set of all ground instances of u that satisfy the constraint ϕ, and $[\![u_1 \vee u_2]\!]$ is the union of the sets $[\![u_1]\!] \cup [\![u_2]\!]$.

To formally define the semantics of guarded terms, *ground* guarded terms are first considered. The ground semantics in Definition 3 is straightforward, because the constraints in ground guarded terms can be determined as either *true* or *false* in the underlying built-in algebra $\mathcal{T}_{\mathcal{E}_0}$. Specifically, a ground guarded term $u|_\phi$ represents either $[\![u]\!]$ if ϕ is satisfiable in $\mathcal{T}_{\mathcal{E}_0}$ or the empty set if ϕ is not satisfiable in $\mathcal{T}_{\mathcal{E}_0}$.

Definition 3. *Given a ground guarded term $u \in \overline{T}_\Sigma$, the set $[\![u]\!] \subseteq T_\Sigma$ of ground terms represented by u is inductively defined as follows:*

$$
\begin{aligned}
[\![t]\!] &= \{t' \in T_\Sigma \mid t' =_E t\} \ \text{ if } t \in T_\Sigma,\\
[\![f(u_1,\ldots,u_n)]\!] &= \{t \in [\![f(t_1,\ldots,t_n)]\!] \mid t_i \in [\![u_i]\!], 1 \le i \le n\},\\
[\![u|_\phi]\!] &= \begin{cases} [\![u]\!] & \text{if } \mathcal{T}_{\mathcal{E}_0} \models \phi \\ \emptyset & \text{if } \mathcal{T}_{\mathcal{E}_0} \not\models \phi, \end{cases}\\
[\![u_1 \vee u_2]\!] &= [\![u_1]\!] \cup [\![u_2]\!].
\end{aligned}
$$

Example 5. Recall the guarded terms in Example 4. The ground guarded term

$$u \;=\; (3+1+1|_{3+1 \overset{3}{=} 0} \vee \emptyset|_{3+1 \overset{3}{\neq} 0})\,(5+4+1|_{5+4 \overset{3}{=} 0} \vee \emptyset|_{5+4 \overset{3}{\neq} 0})$$

represents the set $[\![u]\!] = \{t' \mid t' =_E \emptyset\,(5+4+1)\}$, because $3+1 \overset{3}{\neq} 0$ and $5+4 \overset{3}{=} 0$. Note that $5+4+1 \in [\![u]\!]$, since $5+4+1 =_E \emptyset\,(5+4+1)$.

The semantics of guarded terms in general, introduced by Definition 4, is based on the semantics of ground guarded terms. Each ground instance θu of a guarded term u under a ground substitution θ defines the set $[\![\theta u]\!]$ of ground terms. The set $[\![u]\!]$ is the union of all the sets given by the ground instances of the guarded term u.

Definition 4. *Given a guarded term $u \in \overline{T}_\Sigma(X)$, the set of all ground terms represented by u is defined as $[\![u]\!] = \{t \in T_\Sigma \mid (\exists \theta : X \longrightarrow T_\Sigma)\ t \in [\![\theta u]\!]\}$.*

It is worth noting that the semantics of non-ground guarded terms cannot be defined in the same way as the case of ground guarded terms. Specifically, the second condition $[\![f(u_1,\ldots,u_n)]\!] = \{t \in [\![f(t_1,\ldots,t_n)]\!] \mid t_i \in [\![u_i]\!], 1 \le i \le n\}$ in Definition 3 for ground guarded terms is generally *not* applicable to non-ground guarded terms. For example, consider an unsorted signature Σ with two different constants c and d. Notice that $[\![f(x|_{x\neq y}, y|_{x=y})]\!] = \emptyset$, but $[\![x|_{x\neq y}]\!] = \{c,d\}$ and $[\![y|_{x=y}]\!] = \{c,d\}$.

Example 6. Consider the nested guarded terms in Example 4. The guarded term

$$(I_1+J_1+1|_{I_1+J_1 \overset{3}{=} 0} \vee \emptyset|_{I_1+J_1 \overset{3}{\neq} 0})\,(I_2+J_2+1|_{I_2+J_2 \overset{3}{=} 0} \vee \emptyset|_{I_2+J_2 \overset{3}{\neq} 0})$$

represents the set of terms for ground substitution $\{I_i \mapsto x_i, J_i \mapsto y_i \mid i = 1,2\}$ as follows:

$$
\begin{aligned}
[\![(x_1+y_1+1)\,(x_2+y_2+1)]\!] &\quad \text{if } x_1+y_1 \overset{3}{=} 0 \ \wedge\ x_2+y_2 \overset{3}{=} 0,\\
[\![\emptyset\,(x_2+y_2+1)]\!] &\quad \text{if } x_1+y_1 \overset{3}{\neq} 0 \ \wedge\ x_2+y_2 \overset{3}{=} 0,\\
[\![(x_1+y_1+1)\,\emptyset]\!] &\quad \text{if } x_1+y_1 \overset{3}{=} 0 \ \wedge\ x_2+y_2 \overset{3}{\neq} 0,\\
[\![\emptyset\,\emptyset]\!] &\quad \text{if } x_1+y_1 \overset{3}{\neq} 0 \ \wedge\ x_2+y_2 \overset{3}{\neq} 0.
\end{aligned}
$$

Each one of these cases includes an infinite number of ground terms because there are an infinite number of ground substitutions that satisfy each guard. It can be easily seen that the number of cases increases exponentially: if the number of elements is n, the number of different cases is 2^n.

The notion of constrained terms in Section 2 can be seen as a special case of guarded terms since the denotation $[\![t]\!]_\phi$ of $(t\,;\phi)$ coincides with the semantics of $t|_\phi$.

Lemma 2. *Given a Σ-term $t \in T_\Sigma(X)$ and a constraint ϕ, $[\![t]\!]_\phi = [\![t|_\phi]\!]$ holds.*

3.3 Equivalence

Guarded terms u_1 and u_2 are called *equivalent*, written $u_1 \equiv u_2$, if they represent the same set of ground terms, that is, $[\![u_1]\!] = [\![u_2]\!]$. For example, guarded terms that are identical up to variable renaming are equivalent, since they have the same set of ground instances. Guarded terms are called *ground equivalent*, written $u_1 \equiv_g u_2$, if their ground instances by the same substitution are equivalent. The ground equivalence $u_1 \equiv_g u_2$ implies $u_1 \equiv u_2$, but the converse may not hold.[3]

Definition 5. *Let $u_1, u_2 \in \overline{T}_\Sigma(X)$:*

1. $u_1 \equiv u_2$ *iff* $[\![u_1]\!] = [\![u_2]\!]$.
2. $u_1 \equiv_g u_2$ *iff* $[\![\theta u_1]\!] = [\![\theta u_2]\!]$, *for every ground substitution* $\theta : X \longrightarrow T_\Sigma$.

The semantics of $\vee$ is associative and commutative with respect to ground equivalence, because its is defined as a set union operation. From now on, the expression $u_1 \vee \cdots \vee u_n$ will be written without parentheses using this fact.

Corollary 1 (Associativity and Commutativity of $\vee$). *If $u_1, u_2, u_3 \in \overline{T}_\Sigma(X)$, then*

1. $u_1 \vee u_2 \equiv_g u_2 \vee u_1$.
2. $(u_1 \vee u_2) \vee u_3 \equiv_g u_1 \vee (u_2 \vee u_3)$.

The ground equivalence $\equiv_g$ between guarded terms is a *congruence* as shown in Lemma 3. On the contrary, the equivalence $\equiv$ violates the congruence rules. Specifically, u and its variable renaming σu satisfy $u \equiv \sigma u$, but $u|_\phi$ and $(\sigma u)|_\phi$ are not equivalent.[4]

Lemma 3 (Congruence of Ground Equivalence). *If $u, v \in \overline{T}_\Sigma(X)$ and $u \equiv_g v$, then:*

- $\sigma u \equiv_g \sigma v$ *for a substitution* $\sigma : X \longrightarrow T_\Sigma(X)$.
- $f(u_1, \ldots, u, \ldots, u_n) \equiv_g f(u_1, \ldots, v, \ldots, u_n)$.
- $u|_\phi \equiv_g v|_\phi$.
- $u_1 \vee \cdots \vee u \vee \cdots \vee u_n \equiv_g u_1 \vee \cdots \vee v \vee \cdots \vee u_n$.

[3] As an example, $0|_{x=0} \equiv 0|_{x=1}$, because $[\![0|_{x=0}]\!] = [\![0|_{x=1}]\!] = \{0\}$. But $0|_{x=0} \not\equiv_g 0_{x=1}$, because $[\![\theta(0|_{x=0})]\!] = \{0\}$ and $[\![\theta(0|_{x=1})]\!] = \emptyset$ for $\theta = \{x \mapsto 0\}$, provided that $0 \neq 1$.

[4] For example, $0|_{x=0} \equiv 0_{y=0}$ but $(0|_{x=0})|_{x=1} \not\equiv (0|_{y=0})|_{x=1}$, because $[\![(0|_{x=0})|_{x=1}]\!] = \emptyset$ and $[\![(0|_{y=0})|_{x=1}]\!] = \{0\}$, provided that 0 and 1 are different.

Guarded terms with different syntactic structures can be ground equivalent (and thus they are also equivalent by definition). Lemma 4 identifies some properties of ground equivalence between guarded terms.

Lemma 4 (Ground Equivalence Rules).

- $(\bigvee_{i=1}^{n} u_i)|_\phi \;\equiv_g\; \bigvee_{i=1}^{n}(u_i|_\phi)$
- $f(u_1, \ldots (\bigvee_{i=1}^{n} u_j^i) \ldots, u_n) \;\equiv_g\; \bigvee_{i=1}^{n} f(u_1, \ldots u_j^i \ldots, u_n)$
- $(u|_\varphi)|_\phi \;\equiv_g\; u|_{\varphi\wedge\phi}$
- $f(u_1, \ldots u_j|_\phi \ldots, u_n) \;\equiv_g\; f(u_1, \ldots u_j \ldots, u_n)|_\phi$

Guarded terms can be simplified using these equivalence rules. For example,

$$f(u_1 \vee u_2, v_1 \vee v_2) \;\equiv_g\; f(u_1, v_1 \vee v_2) \vee f(u_2, v_1 \vee v_2) \;\equiv_g\; \bigvee_{i,j\in\{1,2\}} f(u_i, v_j),$$
$$f(u|_\phi, v|_\psi) \;\equiv_g\; f(u, v)|_\phi|_\psi \;\equiv_g\; f(u, v)|_{\phi\wedge\psi}.$$

By repeatedly applying the equivalences in Lemma 4, a (ground) equivalent guarded term that is a combination of normal constrained terms can be obtained.

Theorem 1 (Standard Form). *Every $u \in \overline{T}_\Sigma(X)$ is ground equivalent to a guarded term in* standard form $t_1|_{\phi_1} \vee t_2|_{\phi_2} \vee \cdots \vee t_n|_{\phi_n}$, *with terms $t_1, \ldots, t_n \in T_\Sigma(X)$.*

A guarded term in standard form has a flat structure in which parallel combinations $\vee$ and constraints ϕ appear only at the top. However, a *nested* guarded term can be exponentially smaller than an equivalent one in the standard form. For example, a nested guarded term $f(u_1 \vee v_1, \ldots, u_n \vee v_n)$ of size $O(n)$ is equivalent to its standard form $\bigvee_{w_i\in\{u_i,v_i\}} f(w_1, \ldots, w_n)$ of size $O(2^n)$. This explains why guarded terms are very useful for succinctly representing the symbolic state space.

Example 7. Consider the nested guarded term in Example 6. The guarded term

$$(I_1 + J_1 + 1|_{I_1+J_1 \overset{3}{=} 0} \vee \emptyset|_{I_1+J_1 \overset{3}{\neq} 0})\,(I_2 + J_2 + 1|_{I_2+J_2 \overset{3}{=} 0} \vee \emptyset|_{I_2+J_2 \overset{3}{\neq} 0})$$

is equivalent to the following guarded term in the standard form:

$$(I_1 + J_1 + 1)\,(I_2 + J_2 + 1)|_{I_1+J_1 \overset{3}{=} 0 \,\wedge\, I_2+J_2 \overset{3}{=} 0} \;\vee\; \emptyset\,(I_2 + J_2 + 1)|_{I_1+J_1 \overset{3}{\neq} 0 \,\wedge\, I_2+J_2 \overset{3}{=} 0} \;\vee$$
$$(I_1 + J_1 + 1)\,\emptyset|_{I_1+J_1 \overset{3}{=} 0 \,\wedge\, I_2+J_2 \overset{3}{\neq} 0} \;\vee\; \emptyset\,\emptyset|_{I_1+J_1 \overset{3}{\neq} 0 \,\wedge\, I_2+J_2 \overset{3}{\neq} 0},$$

where each case is expressed as a single disjunct. As explained in Example 6, since the number of cases increases exponentially, the size of the standard form also increases exponentially with respect to the size of the original one.

3.4 Rewriting with Guarded Terms

In principle, one-step rewrite $u \to u'$ between guarded terms u and u' represents a one-step rewrite $t \to t'$ between the corresponding terms $t \in [\![u]\!]$ and $t' \in [\![u']\!]$. The next task is to consider *guarded rewrite rules* of the form $l \to r$ **if** ϕ, where l and r are guarded terms, in order to define rewriting between guarded terms. Using Lemma 1, it can be assumed that l and r are $G(\Sigma)$-terms in the extended signature $G(\Sigma)$. Also, following the ideas of rewriting modulo SMT, rewriting with guarded terms is topmost.

Definition 6. *A guarded rewrite rule is a triple* $l \to r$ **if** ϕ *with two guarded terms* $l, r \in T_{G(\Sigma)}(X)_{g(State)}$ *of top sort State and a constraint* ϕ*.*

As usual, rewriting by guarded rules happens modulo a set of equations E. To be semantically correct regarding guarded terms, a set of equations E should preserve the semantics of guarded terms (i.e., if $u =_E \theta l$, then $u \equiv \theta l$), which is generally not true.[5] But structural axioms meet this condition as stated in Lemma 5. These structural axioms are very useful to specify component-based systems [11]; e.g., the formal specification of the case study in Section 4 frequently uses these axioms. From this point on it is assumed that E only includes structural axioms.

Lemma 5. *Given a set of equations* B *that only includes identity, associativity, and commutativity,* $u =_B v$ *implies* $u \equiv_g v$ *for guarded terms* $u, v, \in T_{G(\Sigma)}(X)$*.*

Definition 7 introduces the ground rewrite relation $\to_{\mathcal{R}}$ for guarded rewrite rules. A ground rewrite relation $u \to_{\mathcal{R}} u'$ holds by a guarded rule $l \to r$ **if** ϕ, whenever u is an instance of l, u' is an instance of $r|_\phi$, and both $[\![u]\!]$ and $[\![u']\!]$ are not empty.

Definition 7. *For ground guarded terms* $u, u' \in T_{G(\Sigma),g(State)}$*, a ground rewrite relation* $u \to_{\mathcal{R}} u'$ *holds if and only if for a guarded rule* $l \to r$ **if** ϕ *and a ground substitution* $\theta : X \longrightarrow T_{G(\Sigma)}$*, it holds that:* $u =_E \theta l$*,* $u' =_E \theta r|_{\theta\phi}$*,* $[\![u]\!] \neq \emptyset$*, and* $[\![u']\!] \neq \emptyset$*.*

The symbolic rewrite relation $\rightsquigarrow_{\mathcal{R}}$ by guarded rules is introduced in Definition 8. The last condition in this definition states that there is at least one ground instance $\theta u \to_{\mathcal{R}} \theta u'$ and also implies that $\mathcal{T}_{\mathcal{E}_0} \models \sigma\phi$ holds (because $u' =_E \sigma r|_{\sigma\phi}$ includes $\sigma\phi$).

Definition 8. *A symbolic rewrite relation* $u \rightsquigarrow_{\mathcal{R}} u'$ *holds for* $u, u' \in T_{G(\Sigma)}(X)$ *if and only if for a guarded rule* $l \to r$ **if** ϕ *and a substitution* $\sigma : X \longrightarrow T_{G(\Sigma)}(X)$*:* $u =_E \sigma l$*,* $u' =_E \sigma r|_{\sigma\phi}$*, and* $(\exists \theta : X \to T_{G(\Sigma)})\ [\![\theta u]\!] \neq \emptyset \wedge [\![\theta u']\!] \neq \emptyset$*.*

Consider a ground rewrite $u \to_{\mathcal{R}} u'$, which is ground equivalent to its standard expansion $u_1 \vee \cdots \vee u_n \to_{\mathcal{R}} u'_1 \vee \cdots \vee u'_m$, where u's standard form is $u_1 \vee \cdots \vee u_n$ and u''s standard form is $u'_1 \vee \cdots \vee u'_m$. This intuitively represents a set of ground rewrite relations $\{u_i \to u'_j \mid 1 \leq i \leq n, 1 \leq j \leq m\}$. This idea can be used to define a standard expansion of a guarded rewrite rule.

Definition 9 (Standard Expansion). *Consider a guarded rule* $l \to r$ **if** ϕ *such that* l*'s standard form is* $\bigvee_{i=1}^{n} l_i|_{\psi^l_i}$ *and* r*'s standard form is* $\bigvee_{j=1}^{m} r_j|_{\psi^r_j}$*. A* standard expansion *of* $l \to r$ **if** ϕ *is a collection of ordinary rewrite rules*

$$S(l \to r \textbf{ if } \phi) = \{l_i \to r_j \textbf{ if } \phi \wedge \psi^l_i \wedge \psi^r_j \mid 1 \leq i \leq n, 1 \leq j \leq m\}.$$

A guarded rewrite rule $l \to r$ **if** ϕ is related to its standard expansion (in terms of simulation). For each ground rewrite by a guarded rewrite rule, there exists a corresponding ground rewrite by its standard expansion. This means that reachability analysis can be effectively performed using guarded rewrite rules.

[5] Consider a set of equations E that replace any constraint in a guarded term by its negation. Then, $0|_{false} \equiv_E 0|_{\neg false}$, but clearly $[\![0|_{false}]\!] \neq [\![0|_{\neg false}]\!]$.

Theorem 2 (Simulation of Standard Expansion). *Consider a set R of guarded rewrite rules and its standard expansion $\hat{R} = \bigcup_{l \to r \text{ if } \phi \in R} S(l \to r \text{ if } \phi)$. For ground guarded terms $u, u' \in T_{G(\Sigma)}$, $u \to_{\mathcal{R}} u' \implies t \to_{\hat{R}} t'$ holds for any $t \in [\![u]\!]$ and $t' \in [\![u']\!]$.*

However, ground rewriting by a guarded rule may not capture all of ground rewriting by its standard expansion. Consider a guarded term $f(0) \vee g(0)$ and a guarded rewrite rule $f(x) \vee g(s(x)) \to f(x)$ **if** $x < 1$. Clearly, $f(0) \vee g(0)$ cannot be rewritten since it is not equal to any ground instance of $f(x) \vee g(s(x))$. But a ground instance of the rule $f(0) \vee g(s(0)) \to f(0)$ **if** $0 < 1$ suggests that $f(0) \to_{\mathcal{R}} f(0)$ holds. The notion of *admissible guarded terms* with respect to a guarded rewrite rule is introduced.

Definition 10. *A guarded term $u \in T_{G(\Sigma)}(X)$ is admissible for a rule $l \to r$ **if** ϕ if and only if $[\![u]\!] \cap [\![l]\!] \neq \emptyset \iff u =_E \sigma l$ for a substitution $\sigma : X \longrightarrow T_\Sigma(X)$.*

The admissibility requirements can be checked by using the standard form of guarded terms. In order to check $[\![u]\!] \cap [\![l]\!] \neq \emptyset$, u's standard form $\bigvee_i u_i|_{\phi_i}$ and l's standard form $\bigvee_j l_j|_{\psi_j}$ can be both computed, and then check if $\theta u_i =_E \theta l_j$ for a ground substitution θ such that $\mathcal{T}_{\mathcal{E}_0} \models \theta\phi_i \wedge \theta\psi_j$. If no such substitution exists, then clearly $[\![u]\!] \cap [\![l]\!] = \emptyset$. This problem can be determined by E-unification. If $[\![u]\!] \cap [\![l]\!] \neq \emptyset$, it suffices to check if $u =_E \sigma l$ for a substitution σ, which can be determined by E-matching.

For admissible guarded terms, a symbolic rewrite relation exactly captures a ground rewrite relation as stated in Theorem 3. First, the notion of admissible terms is extended to guarded rewrite rules. A guarded rewrite rule $l \to r$ **if** ϕ is admissible for a set R of guarded rewrite rules if and only if the right side r is admissible for every rule in R. A set R of guarded rewrite rules is admissible if and only if every guarded rewrite rule in R is admissible for every rule in R.

Theorem 3 (Symbolic Guarded Rewriting). *Let R be a set of admissible guarded rules, $\theta : X \longrightarrow T_{G(\Sigma)}$ be a ground substitution, and $u, u' \in T_{G(\Sigma)}(X)$ be admissible guarded terms. Then: (i) If $u \rightsquigarrow_{\mathcal{R}} u'$, then $\theta u \to_{\mathcal{R}} \theta u'$ whenever $[\![\theta u]\!] \neq \emptyset$ and $[\![\theta u']\!] \neq \emptyset$. (ii) If $\theta u \to_{\mathcal{R}} w'$ for $w' \in T_{G(\Sigma)}$, then $u \rightsquigarrow_{\mathcal{R}} u'$ and $[\![w']\!] \subseteq [\![u']\!]$ for some u'.*

Example 8. Consider the following guarded rewrite rules:

$$I_1\ I_2\ C_1 \parallel C_2 \to I_1\ C_1 \parallel (I_1 + I_2 + 1|_{(I_1+I_2)\overset{3}{=}0} \vee \emptyset|_{(I_1+I_2)\overset{3}{\neq}0})\ C_2 \text{ if } 0 < I_1 \wedge 0 < I_2$$

$$I_1 \parallel (I_2|_B \vee C_1|_{\neg B})\ C_2 \to \emptyset \parallel (I_2|_B \vee C_1|_{\neg B})\ C_2 \text{ if } B \wedge 0 < I_1 \wedge 0 < I_2 \wedge I_1 + I_2 \overset{17}{=} 0.$$

These rules are admissible, because each state is a pair $D_1 \parallel D_2$, constraints only appear in D_2, and the left-hand side of each rule defines the most general pattern for D_2. By Theorem 2, these rules are related to the standard expansion:

$$I_1\ I_2\ C_1 \parallel C_2 \to I_1\ C_1 \parallel I_1 + I_2 + 1\ C_2 \quad \text{if } 0 < I_1 \wedge 0 < I_2 \wedge (I_1 + I_2) \overset{3}{=} 0$$

$$I_1\ I_2\ C_1 \parallel C_2 \to I_1\ C_1 \parallel \emptyset\ C_2 \quad \text{if } 0 < I_1 \wedge 0 < I_2 \wedge (I_1 + I_2) \overset{3}{\neq} 0$$

$$I_1 \parallel I_2\ C_2 \to \emptyset \parallel I_2\ C_2 \text{ if } B \wedge 0 < I_1 \wedge 0 < I_2 \wedge I_1 + I_2 \overset{17}{=} 0 \wedge B$$

$$I_1 \parallel C_1\ C_2 \to \emptyset \parallel C_1\ C_2 \text{ if } B \wedge 0 < I_1 \wedge 0 < I_2 \wedge I_1 + I_2 \overset{17}{=} 0 \wedge \neg B$$

The last rule can be discarded because its condition is always unsatisfiable (containing both B and $\neg B$). Thanks to theorems 2 and 3, the above guarded rewrite rules can be used to perform reachability analysis from an admissible symbolic state.

The use of guarded terms for the symbolic specification in Section 2.2 results in the search command proving the deadlock-freedom property that had previously timed-out:

```
search [1]  { I  I+1 I+2 I+3 || none, I > 0 }
       =>!  { C1 | C2, Phi } such that C1 =/= none = true .
No solution.
states: 265  rewrites: 37516 in 702ms cpu (704ms real)
```

This means that for *any* collection of 4 consecutive positive integers in the left channel, the system will eventually reach a state without numbers in this channel.

4 A Case Study

This section shows how the number of symbolic states can be dramatically reduced by slightly introducing guarded terms in the existing symbolic specification of CASH in [19]. The CASH algorithm was first formally specified and analyzed in Real-Time Maude using explicit-state methods [17], and symbolically analyzed using rewriting modulo SMT to deal with an infinite number of states [19]. Due to the symbolic state space explosion problem and the complexity of the accumulated constraints, some reachability properties of interest were previously beyond the reach of rewriting modulo SMT in [19], but can now be analyzed using guarded terms.

4.1 Symbolic States Using Guarded Terms

In CASH, each task has a given period and a given worst-case execution time. When an instance of a task is completed before its worst-case execution time, the unused processor time is added to a global queue of unused budgets. Another task can then use these unused execution times as well as its own execution budget, instead of just wasting them. In addition to capacity sharing, tasks are scheduled according to standard preemptive EDF scheduling: the task with the shortest time remaining until its deadline has the priority and can preempt the currently executing task. The reader is referred to [7, 17] for details about the CASH algorithm.

In rewriting logic specification, an object of class C is represented as a term `< O : C |` att_1 `:` v_1 `, ... ,` att_n `:` v_n `>` of sort *Object*, where O is the identifier, and $v_1, \ldots, v_n$ are the values of the attributes $att_1, \ldots, att_n$ [11]. A system state, called a *configuration*, is declared as a multiset of these objects. Multiset union for configurations is given by a juxtaposition operator that is associative and commutative and having the `none` multiset as its identity element.

In the CASH specification, the configuration consists of a number of servers (i.e., tasks) and a global queue. A server state consists of six attributes: the maximum budget, period, internal state (`idle`, `waiting`, or `executing`), time executed, budget time used, and time to deadline. These variables are modeled as nonnegative integers. Specifically, for internal states, 0 denotes `idle`, 1 denotes `waiting`, and 2 denotes `executing`. A server object with name O is modeled as a term of sort *Object* and has the form

```
< O : server | maxBudget : m,  period : p,      state : s,
               timeExecuted : e, usedOfBudget : u, timeToDeadline : d >
```

A global object contains the global queue of unused budgets, the availability of the processor, and a flag to denote whether any deadline has been missed. Both availability and deadline missed flags are modeled as Boolean values. A global object with name G is modeled as a term of sort *Object* and has the form

`< G : global | cq :` *queue*`, available :` a`, deadline-miss :` b `>`

A global queue is a priority queue of unused budgets. Each item in a queue is a pair `(deadline:` i `budget:` i'`)` of deadline i and budget i'. An item with the smallest deadline has the highest priority. In [17, 19], a global queue is specified as a sorted list. Instead, in this work the queue is modeled as a *multiset* of items in order to symbolically define queue operations using only SMT constraints. For example, consider a queue

$$cq \equiv (\texttt{deadline:}\ i_1\ \texttt{budget:}\ i'_1)\ \ldots\ (\texttt{deadline:}\ i_n\ \texttt{budget:}\ i'_n)$$

with n items. The operation to peek the highest-priority element is defined by:

$$peek(cq) = (\texttt{deadline:}\ i_k\ \texttt{budget:}\ i'_k) \iff \textstyle\bigwedge_{j=1}^{n} i_k \leq i_j.$$

Guarded terms are used in the new symbolic specification in a number of ways. First, each variable in a server state or a global object can be guarded by constraints. For example, the guarded term $0_\phi \vee 1_{\neg\phi}$ denotes an internal state of a server which is either 0, if the condition ϕ holds, or 1, otherwise. Second, each item in a server state can be guarded by constraints. In a similar way to the example in Section 2, the guarded term $(\texttt{deadline:}\ i\ \texttt{budget:}\ i')|_\phi \vee emptyQueue|_{\neg\phi}$ denotes an item which is either present if the condition ϕ is satisfiable, or absent otherwise.

Some syntactic sugar is defined to succinctly write these guarded terms as follows: $\phi\ ?\ i : j$ denotes the guarded term $i_\phi \vee j_{\neg\phi}$, for $i, j \in \mathbb{Z}$, and $(e\ \texttt{const:}\ \phi)$ denotes the guarded term $e|_\phi \vee e|_{\neg\phi}$, for each item e in the priority queue. For example, the following expression includes both types of guarded terms:

```
(deadline: I3 >= 1 ? I3 - 1 : 0  budget: I2 >= 1 ? 1 : 0  const: I2 > 1).
```

4.2 Symbolic Transitions Using Guarded Rewrite Rules

The symbolic transitions of the CASH algorithm are specified by 13 conditional rewrite rules in [19], which are adapted from the Real-Time Maude specification in [17]. The rule conditions specify constraints solvable by the SMT decision procedure, together with some extra conditions to generate constraints by rewriting. This section highlights some of the rules to explain how to remove symbolic branching using guarded terms. The reader is referred to [5] for the full description of all (guarded) rewrite rules.

Rule `stopExecuting1`. This rule specifies when one server completes execution while another server is waiting. The waiting server with the least time remaining until its deadline starts executing, and any budget left is added to the global queue. Previously, this behavior was specified by two rewrite rules in [19], depending on whether any budget remains. However, with guarded terms only one rewrite rule is needed:

```
   {< G  : global | cq : CQ, AtSG >
    < O  : server | state : St,                 usedOfBudget : T,
                    maxBudget : NZT,            timeToDeadline : T1,
                    timeExecuted : NZT1,        period : NZT2, AtS >
    < O' : server | state : St', timeToDeadline : T2, AtS' > REST}
=> {< G  : global | cq : (deadline: T1  budget: (NZT monus T)
                          const: (NZT > T)) CQ, AtSG >
    < O  : server | state : idle,               usedOfBudget : NZT,
                    maxBudget : NZT,            timeToDeadline : T1,
                    timeExecuted : NZT1,        period : NZT2, AtS >
    < O' : server | state : executing, timeToDeadline : T2, AtS' > REST}
if ALL[6] := ...  /\  (St === executing) /\  (St' === waiting)  /\  (NZT > 0)
/\ (NZT1 > 0)  /\  (NZT2 > 0)  /\  (T >= 0)  /\  ((NZT monus T) <= T1)[7]
/\ (T1 >= 0)  /\  (T2 >= 0)  /\  nextDeadlineWaiting(ALL,O,T2)[8]
```

The guarded term `(deadline: T1 budget: (NZT monus T) const: (NZT > T))`, which is only present in the priority queue if the condition `NZT > T` is satisfiable, is written in the `global` object in the right hand side of the rewrite rule. Unlike [19], no constraints are yet added regarding the "position" of the newly added item in the queue, because: (i) the item may be absent if its constraint is not satisfiable, and (ii) the priority queue is specified as a multiset with symbolic constraints.

Rule `tickExecutingSpareCapacity`. This rule models time elapse when a server is executing a spare capacity, specified as one guarded rewrite rule below.[9] The left hand side of the rule contains the guarded term `(deadline: I1 budget: I2 const: iB)`. The condition `iB` in the rule condition specifies that the item is present in the queue. The condition `aboveOrEqualDeadline(I1,CQ)` states that all deadlines in `CQ` are at least `I1`, that is, that the item is indeed the one with the highest priority.

```
   {< G : global | cq : (deadline: I1 budget: I2 const: iB) CQ, AtSG >
    < O : server | state : St,
                   timeExecuted : T1,   timeToDeadline : T2, AtS > REST}
=> { delta-global( < G : global | cq : (deadline: I1 budget: (I2 + (- 1))
                     const: (iB and (I2 > 1))) CQ, AtSG >, 1)
     delta-servers(< O : server | state : executing,    timeExecuted : T1,
                     timeToDeadline : T2, AtS > REST, 1, false)}
if ALL := ...  /\  (St === executing)  /\  (T1 >= 0) /\  (T2 >= 0)
/\ iB  /\  (I1 >= 1)  /\  (I2 >= 1)  /\  (T2 >= 1)  /\  (I1 <= T2)
/\ mte-server(ALL,O,1)  /\  noDeadlineMiss(ALL)
/\ aboveOrEqualDeadline(I1,CQ)
```

[6] The specification of `ALL` has been omitted in the rule. `ALL` representsthe entire configuration in the left hand side.

[7] The `monus` operator is defined by: a `monus` $b = \max(a-b, 0)$.

[8] The function `nextDeadlineWaiting(ALL,O,T2)` returns a constraint stating that the `timeToDeadline`of all servers, except `O`, is at least `T2`.

[9] The functions `delta-global` and `delta-servers` model the time lapse; that is, variables in servers and the queue are accordingly increased or decreased (by 1) based on the current state.

4.3 Symbolic Reachability Analysis

The goal is to symbolically analyze if missed deadlines exist for variant of) the CASH algorithm from an infinite set of initial configurations containing two servers s_1 and s_2. Suppose that server s_i (for $i = 1, 2$) has maximum budget b_i and period p_i such that $0 \leq b_i < p_i$. No deadline misses are guaranteed only when the processor utilization is less than or equal to 100% (i.e., $b_1/p_1 + b_2/p_2 \leq 1$) [7]. A counterexample exists if there is no constraint on the initial configurations. Indeed, the previous work [19] can find a symbolic counterexample that violates this constraint using rewriting module SMT.

A more interesting problem is to check the existence of missed deadlines of the *variant* of the CASH algorithm [17], that uses a different scheduling strategy, when $b_1/p_1 + b_2/p_2 \leq 1$ is satisfied. Using explicit-state model checking, it is shown in [17] that the variant can have a counterexample, even though $b_1/p_1 + b_2/p_2 \leq 1$ is satisfied. Since this constraint is non-linear, it is beyond the capabilities of state-of-the-art SMT solvers. But by fixing p_1 and p_2, say $p_1 = 5$ and $p_2 = 7$, the constraint becomes linear and symbolic analysis can be performed using existing SMT solvers.

Here an infinite set of initial configurations with two servers s_1 and s_2 is considered. Server s_i (for $i = 1, 2$) has maximum budget b_i and period p_i such that $0 \leq b_i < p_i$, and has the constraint $b_1/p_1 + b_2/p_2 \leq 1$. A symbolic state can be modeled as term `init(`b_1`,`p_1`,`b_2`,`p_2`)`. The previous work in [19] cannot deal with this problem due to the symbolic state space explosion; it generates nearly one billion symbolic states and does not terminate for a few days. On the other hand, the new specification using guarded terms can successfully find a counterexample from `init(`b_1`,5,`b_2`,7)`:

```
search [1] init(I0,5,I2,7)
       =>* {B:Bool , < g : global | deadline-miss : true, AtS > Cnf} .
Solution 1 (state 590751)
states: 590752  rewrites: 1910935222 in 5513748ms cpu (5513848ms real)
B --> not I0 < 0 and not I2 + -5 < 0 and not 1 + - I0 < 0 and not 7 + -
 I2 < 0 and not 35 + - (I2 * 5) + - (I0 * 7) < 0 and (not 7 + - I2 < 0 or
 I2 + -1 < 0) and (not 12 + - I2 < 0 or I2 < 0) and (not 17 + - I2 < 0 or
 I2 < 0) and - I0 < 0 and I2 + -7 < 0 and I0 + -5 < 0 and 4 + - I2 < 0
Cnf --> < s1 : server | maxBudget : I0, period : 5, state : 2,
              usedOfBudget : 0, timeToDeadline : 0, timeExecuted : 5 >
        < s2 : server | maxBudget : I2, period : 7, state : 0,
              usedOfBudget : I2, timeToDeadline : 7, timeExecuted : 2 >
AtS --> cq : deadline: 7 budget: not I2 < 0 ? I2 + -5 : -5
            const: (not I2 < 0 and 5 + - I2 < 0), available : false
```

The counterexample is found at search depth 23, which could not be reached without guarded terms. A sequence of symbolic rewrites to reach this counterexample can be obtained by Maude's `show path` command. In the counterexample, `B` represents the accumulated constraint whose satisfiable assignment gives a concrete counterexample. The queue `cq` has the single item if the constraint `(not I2 < 0 and 5 + - I2 < 0)` is satisfied; otherwise, `cq` is empty. For example, `B` has the satisfiable assignment `I0` = 1 and `I2` = 5, which is found by an SMT solver, and in this case the queue `cq` is empty. An explicit search with these concrete values in the *ground* rewriting logic semantics of CASH gives an expected result as follows:

```
search [1] init(1,5,5,7)
       =>* {true, < g : global | deadline-miss : true, AtS > Cnf} .
Solution 1 (state 107393)
states: 107394  rewrites: 36507422 in 29543ms cpu (29699ms real)
Cnf --> < s1 : server | maxBudget : 1, period : 5, state : 2,
              usedOfBudget : 0, timeToDeadline : 0, timeExecuted : 5 >
        < s2 : server | maxBudget : 5, period : 7, state : 0,
              usedOfBudget : 5, timeToDeadline : 7, timeExecuted : 2 >
AtS --> cq : emptyQueue, available : false
```

5 Related Work and Concluding Remarks

SMT-based reachability analysis has been used in software component testing and verification of infinite-state systems [9]. SMT-CBMC [1] and Corral [13] use bounded model checking, with unbounded types represented by built-in variables. KLEE [8] is used for symbolic execution and constraint solving, finding possible inputs that will cause a programming artifact to crash. The IC3 SMT-based model checker [10] uses over- and under-approximation techniques to efficiently handle symbolic transitions on infinite sets of states. What all these approaches have in common is the effort in developing advanced techniques and tools to speed up the reachability analysis process based on SMT-solving. Guarded terms are a technique to speed up and often attain convergence of the reachability analysis process for rewriting modulo SMT. It complements narrowing-based reachability analysis [2, 16], another symbolic technique combining narrowing modulo theories and model checking.

This paper presented *guarded terms*, a technique with the potential to reduce the symbolic state space and the complexity of constraints in the rewriting modulo SMT approach. Rewriting modulo SMT [19] is a novel symbolic technique to model and analyze reachability properties of infinite-state systems. This is a technique in the realm of rewriting logic that can greatly improve the specification and verification of reachability properties of open systems such as real-time and cyber-physical systems. Guarded terms generalize the constrained terms of rewriting modulo SMT by allowing a term in a symbolic state to have constrained subterms: these subterms can be seen as a choice operator that is part of the term structure. They can be composed in parallel and be nested, thus enabling the succinct encoding of several constrained terms into one guarded term. The potential of guarded terms for reducing the symbolic state-space, and the complexity and size of constraints has been illustrated by a running example and a case study. The latter is an improvement of a previously developed case study where guarded terms enable the analysis of reachability properties of the CASH scheduling algorithm [7].

As future work, the plan is to explore the use of guarded terms in improving the symbolic rewriting logic semantics of PLEXIL [18, 19], and in specifying the symbolic rewriting logic semantics of other real-time and cyber-physical systems. Other SMT techniques, including state subsumption, backwards reachability, k-induction, and interpolants, should certainly be studied for rewriting modulo SMT. Another perspective is to use guarded terms for improving narrowing-based reachability analysis.

Acknowledgments. The first author was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2016R1D1A1B03935275). The second author has been supported in part by the EPIC project funded by the Administrative Department of Science, Technology and Innovation of Colombia (Colciencias) under contract 233-2017.

References

1. A. Armando, J. Mantovani, and L. Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. *Software Tools for Technology Transfer*, 11(1), 2009.
2. K. Bae, S. Escobar, and J. Meseguer. Abstract logical model checking of infinite-state systems using narrowing. In *RTA*, volume 21 of *LIPIcs*, pages 81–96. Schloss Dagstuhl, 2013.
3. K. Bae, P. Ölveczky, and J. Meseguer. Definition, semantics, and analysis of multirate synchronous aadl. In *FM*, volume 8442 of *LNCS*, pages 94–109. Springer, 2014.
4. K. Bae, P. C. Ölveczky, T. H. Feng, E. A. Lee, and S. Tripakis. Verifying hierarchical Ptolemy II discrete-event models using Real-Time Maude. *Science of Computer Programming*, 77(12):1235–1271, 2012.
5. K. Bae and C. Rocha. A Note on Guarded Terms for Rewriting Modulo SMT. `http://sevlab.postech.ac.kr/~kmbae/rew-smt`, July 2017.
6. R. Bruni and J. Meseguer. Semantic foundations for generalized rewrite theories. *Theoretical Computer Science*, 360(1-3):386–414, Aug. 2006.
7. M. Caccamo, G. Buttazzo, and L. Sha. Capacity sharing for overrun control. In *RTSS*, pages 295–304. IEEE, 2000.
8. C. Cadar, D. Dunbar, and D. R. Engler. KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*, pages 209–224, 2008.
9. C. Cadar and K. Sen. Symbolic execution for software testing: Three decades later. *Communications of the ACM*, 56(2):82–90, Feb. 2013.
10. A. Cimatti and A. Griggio. Software model checking via IC3. In *CAV*, volume 7358 of *LNCS*, pages 277–293. Springer, 2012.
11. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude - a High-performance Logical Framework*, volume 4350. Springer-Verlag, Berlin, Heidelberg, 2007.
12. G. Dowek, C. Muñoz, and C. Rocha. Rewriting Logic Semantics of a Plan Execution Language. *Electronic Proceedings in Theoretical Computer Science*, 18:77–91, Feb. 2010.
13. A. Lal, S. Qadeer, and S. Lahiri. Corral: A solver for reachability modulo theories. Technical Report MSR-TR-2012-9, Microsoft Research, January 2012.
14. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, Apr. 1992.
15. J. Meseguer. Twenty years of rewriting logic. *The Journal of Logic and Algebraic Programming*, 81(7-8):721–781, 2012.
16. J. Meseguer and P. Thati. Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. *Higher-Order and Symbolic Computation*, 20(1-2):123–160, Apr. 2007.
17. P. C. Ölveczky and M. Caccamo. Formal simulation and analysis of the CASH scheduling algorithm in Real-Time Maude. In *FASE*, pages 357–372, Berlin, Heidelberg, 2006. Springer.
18. C. Rocha. *Symbolic Reachability Analysis for Rewrite Theories*. PhD thesis, University of Illinois, Dec. 2012.
19. C. Rocha, J. Meseguer, and C. Muñoz. Rewriting modulo SMT and open system analysis. *Journal of Logical and Algebraic Methods in Programming*, 86(1):269–297, Jan. 2017.