1 Introduction

In [1], we have described a technique for computing non-regular approximations using synchronized tree languages. This technique can handle the reachability problem of [2]. These synchronized tree languages [4, 3] are recognized using CS-programs [5], i.e. a particular class of Horn clauses. From an initial CS-program $\text{Prog}$ and a left-linear term rewrite system (TRS) $R$, another CS-program $\text{Prog}'$ is computed in such a way that its language represents an over-approximation of the set of terms (called descendants) reachable by rewriting using $R$, from the terms of the language of $\text{Prog}$. This algorithm is called completion. However, the assumptions of the result showing that all the descendants are obtained, i.e. Theorem 14 in [1], are not correct. Actually, preserving should be replaced by non-copying (a variable cannot occur several times in the head of a clause). However, the non-copying nature of a CS-program is not preserved by completion as soon as the given TRS is not right-linear. Consequently, the final result presented in [1] holds for completely linear TRS, and not for just left-linear TRS.

In this paper, we propose a correction of [1], assuming that the initial CS-program is non-copying, and the TRS is completely linear (see Section 3).

2 Preliminaries

Consider two disjoint sets, $\Sigma$ a finite ranked alphabet and $\text{Var}$ a set of variables. Each symbol $f \in \Sigma$ has a unique arity, denoted by $\text{ar}(f)$. The notions of first-order term, position and substitution are defined as usual. Given two substitutions $\sigma$ and $\sigma'$, $\sigma \circ \sigma'(x) = \sigma(\sigma'(x))$. $T_{\Sigma}$ denotes the set of ground terms (without variables) over $\Sigma$. For a term $t$, $\text{Var}(t)$ is the set of variables of $t$, $\text{Pos}(t)$ is the set of positions of $t$. For $p \in \text{Pos}(t)$, $t(p)$ is the symbol of $\Sigma \cup \text{Var}$ occurring at position $p$ in $t$, and $t|_p$ is the subterm of $t$ at position $p$. The term $t$ is linear if each variable of $t$ occurs only once in $t$. The term $t[t'|_p$ is obtained from $t$ by replacing the subterm at position $p$ by $t'$. \(\text{PosVar}(t) = \{p \in \text{Pos}(t) \mid t(p) \in \text{Var}\}\), \(\text{PosNonVar}(t) = \{p \in \text{Pos}(t) \mid t(p) \notin \text{Var}\}\).

A rewrite rule is an oriented pair of terms, written $l \rightarrow r$. We always assume that $l$ is not a variable, and $\text{Var}(r) \subseteq \text{Var}(l)$. A rewrite system $R$ is a finite set of rewrite rules. lhs stands for left-hand-side, rhs for right-hand-side. The rewrite relation $\rightarrow_R$ is defined as follows: $t \rightarrow_R t'$ if there exist a position $p \in$
Prog predicate symbols ofProg is a logic program composed of CS-clauses.

A CS-clause is a Horn-clause \( H \leftarrow B \) s.t. \( B \) is flat and linear. A CS-program \( \text{Prog} \) is a logic program composed of CS-clauses. \( \text{Pred}(\text{Prog}) \) denotes the set of predicate symbols of \( \text{Prog} \). Given a predicate symbol \( P \) of arity \( n \), the tree-(tuple) language generated by \( P \) is \( L(P) = \{ t \in (T_\Sigma)^n \mid P(t) \in \text{Mod}(\text{Prog}) \} \), where \( T_\Sigma \) is the set of ground terms over the signature \( \Sigma \) and \( \text{Mod}(\text{Prog}) \) is the least Herbrand model of \( \text{Prog} \). \( L(P) \) is called Synchronized language.

The following definition describes syntactic properties over CS-clauses.

**Definition 2.** A CS-clause \( P(t_1, \ldots, t_n) \leftarrow B \) is:

- empty if \( \forall i \in \{1, \ldots, n\} \), \( t_i \) is a variable.
- normalized if \( \forall i \in \{1, \ldots, n\} \), \( t_i \) is a variable or contains only one occurrence of function-symbol.
- preserving if \( \text{Var}(P(t_1, \ldots, t_n)) \subseteq \text{Var}(B) \).
- non-copying if \( P(t_1, \ldots, t_n) \) is linear.

A CS-program is normalized, preserving, non-copying if all its clauses are.

**Example 1.** The CS-clause \( P(x, y, z) \leftarrow Q(x, y, z) \) is empty, normalized, preserving, and non-copying (\( x, y, z \) are variables).

The CS-clause \( P(f(x), y, g(x, z)) \leftarrow Q_1(x), Q_2(y, z) \) is normalized, preserving, and copying. \( P(f(g(x)), y) \leftarrow Q(x) \) is not normalized and not preserving.

Given a CS-program, we focus on two kinds of derivations.

---

1 In former papers, synchronized tree-tuple languages were defined thanks to sorts of grammars, called constraint systems. Thus "CS" stands for Constraint System.
Definition 3. Given a logic program \( \text{Prog} \) and a sequence of atoms \( G \),

- \( G \) derives into \( G' \) by a resolution step if there exist a clause\(^2\) \( H \leftarrow B \) in \( \text{Prog} \) and an atom \( A \in G \) such that \( A \) and \( H \) are unifiable by the most general unifier \( \sigma \) (then \( \sigma(A) = \sigma(H) \)) and \( G' = \sigma(G)[\sigma(A) \leftarrow \sigma(B)] \). It is written \( G \rightarrow_\sigma G' \).

- \( G \) rewrites into \( G' \) if there exist a clause \( H \leftarrow B \) in \( \text{Prog} \), an atom \( A \in G \), and a substitution \( \sigma \), such that \( A = \sigma(H) \) (\( A \) is not instantiated by \( \sigma \)) and \( G' = \text{Prog}[A \leftarrow \sigma(B)] \). It is written \( G \rightarrow \sigma G' \).

Sometimes, we will write \( G \sim_{[H \leftarrow B, \sigma]} G' \) or \( G \rightarrow_{[H \leftarrow B, \sigma]} G' \) to indicate the clause used by the step.

Example 2. Let \( \text{Prog} = \{P(x_1, g(x_2)) \leftarrow P'(x_1, x_2). P(f(x_1), x_2) \leftarrow P''(x_1, x_2)\} \), and consider \( G = P(f(x), y) \). Thus, \( P(f(x), y) \sim_{\sigma_1} P'(f(x), x_2) \) with \( \sigma_1 = [x_1/f(x), y/g(x_2)] \) and \( P(f(x), y) \rightarrow_{\sigma_2} P''(x, y) \) with \( \sigma_2 = [x_1/x, x_2/y] \).

Note that for any atom \( A \), if \( A \rightarrow B \) then \( A \sim B \). If in addition \( \text{Prog} \) is preserving, then \( \text{Var}(A) \subseteq \text{Var}(B) \). On the other hand, \( A \sim \sigma B \) implies \( \sigma(A) \rightarrow B \). Consequently, if \( A \) is ground, \( A \sim B \) implies \( A \rightarrow B \).

We consider the transitive closure \( \sim^+ \) and the reflexive-transitive closure \( \sim^* \) of \( \sim \).

For both derivations, given a logic program \( \text{Prog} \) and three sequences of atoms \( G_1, G_2 \) and \( G_3 \):

- if \( G_1 \sim_{\sigma_1} G_2 \) and \( G_2 \sim_{\sigma_2} G_3 \) then one has \( G_1 \sim_{\sigma_2 \circ \sigma_1} G_3 \);
- if \( G_1 \rightarrow_{\sigma_1} G_2 \) and \( G_2 \rightarrow_{\sigma_2} G_3 \) then one has \( G_1 \rightarrow_{\sigma_2 \circ \sigma_1} G_3 \).

In the remainder of the paper, given a set of CS-clauses \( \text{Prog} \) and two sequences of atoms \( G_1 \) and \( G_2 \), \( G_1 \sim^*_{\text{Prog}} G_2 \) (resp. \( G_1 \rightarrow^*_{\text{Prog}} G_2 \)) also denotes that \( G_2 \) can be derived (resp. rewritten) from \( G_1 \) using clauses of \( \text{Prog} \).

It is well known that resolution is complete.

Theorem 1. Let \( A \) be a ground atom. \( A \in \text{Mod}(\text{Prog}) \) iff \( A \sim^*_{\text{Prog}} \emptyset \).

2.2 Computing descendants

We just give the main ideas using an example. See [1] for a formal description.

Example 3. Let \( R = \{f(x) \rightarrow g(h(x))\} \) and let \( I = \{f(a)\} \) generated by the CS-program \( \text{Prog} = \{P(f(x)) \leftarrow Q(x). Q(a) \leftarrow\} \). Note that \( R^*(I) = \{f(a), g(h(a))\} \).

To simulate the rewrite step \( f(a) \rightarrow g(h(a)) \), we consider the rewrite-rule’s left-hand side \( f(x) \). We can see that \( P(f(x)) \rightarrow_{\text{Prog}} Q(x) \) and \( P(f(x)) \rightarrow_R P(g(h(x))) \). Then the clause \( P(g(h(x))) \leftarrow Q(x) \) is called critical pair\(^3\). This

\(^2\) We assume that the clause and \( G \) have distinct variables.

\(^3\) In former work, a critical pair was a pair. Here it is a clause since we use logic programs.
critical pair is not convergent (in $\text{Prog}$) because $\neg(P(g(h(x))) \rightarrow_{\text{Prog}}^{\ast} Q(x))$.
To get the descendants, the critical pairs should be convergent. Let $\text{Prog}' = \text{Prog} \cup \{P(g(h(x))) \leftarrow Q(x)\}$. Now the critical pair is convergent in $\text{Prog}'$, and note that the predicate $P$ of $\text{Prog}'$ generates $R^*(I)$.

For technical reasons$^4$, we consider only normalized CS-programs, and $\text{Prog}'$ is not normalized. The critical pair can be normalized using a new predicate symbol, and replaced by normalized clauses $P(g(y)) \leftarrow Q_1(y)$. $Q_1(h(x)) \leftarrow Q(x)$. This is the role of Function $\text{norm}$ in the completion algorithm below.

In general, adding a critical pair (after normalizing it) into the CS-program may create new critical pairs, and the completion process may not terminate. To force termination, two bounds $\text{predicate-limit}$ and $\text{arity-limit}$ are fixed. If $\text{predicate-limit}$ is reached, Function $\text{norm}$ should re-use existing predicates instead of creating new ones. If a new predicate symbol is created whose arity$^5$ is greater than $\text{arity-limit}$, then this predicate has to be cut by Function $\text{norm}$ into several predicates whose arities do not exceed $\text{arity-limit}$. On the other hand, for a fixed$^6$ CS-program, the number of critical pairs may be infinite. Function $\text{removeCycles}$ modifies some clauses so that the number of critical pairs is finite.

**Definition 4 (comp as in [1]).** Let arity-limit and predicate-limit be positive integers. Let $R$ be a left-linear rewrite system, and $\text{Prog}$ be a finite, normalized and preserving CS-program. The completion process is defined by:

Function $\text{comp}_R(\text{Prog})$

$\text{Prog} = \text{removeCycles}(\text{Prog})$

while there exists a non-convergent critical pair $H \leftarrow B$ in $\text{Prog}$ do

$\text{Prog} = \text{removeCycles}(\text{Prog} \cup \text{norm}_{\text{Prog}}(H \leftarrow B))$

end while

return $\text{Prog}$

The following results show that an over-approximation of the descendants is computed.

**Theorem 2 ([1]).** Let $\text{Prog}$ be a normalized and preserving CS-program and $R$ be a left-linear rewrite system.
If all critical pairs are convergent, then $\text{Mod}(\text{Prog})$ is closed under rewriting by $R$, i.e. $(A \in \text{Mod}(\text{Prog}) \land A \rightarrow_{R}^{\ast} A') \implies A' \in \text{Mod}(\text{Prog})$.

**Theorem 3 ([1]).** Let $R$ be a left-linear TRS and $\text{Prog}$ be a normalized preserving CS-program. Function $\text{comp}$ always terminates, and all critical pairs are convergent in $\text{comp}_R(\text{Prog})$. Moreover, $R^*(\text{Mod}(\text{Prog})) \subseteq \text{Mod}(\text{comp}_R(\text{Prog}))$.

### 3 Fixing [1]

The hypotheses mentioned in Definition 4 and Theorems 2 and 3 are not sufficient to ensure the computation of an over-approximation.

---

$^4$ Critical pairs are computed only at root positions.

$^5$ The number of arguments.

$^6$ i.e. without adding new clauses in the CS-program.
Indeed, let us describe two counter-examples that are strongly connected.

**Example 4.** Let $	ext{Prog} = \{ P(f(x), f(x)) \leftarrow Q(x) \cdot Q(a) \leftarrow . \cdot Q(b) \leftarrow . \}$ and $R = \{ a \rightarrow b \}$. $\text{Prog}$ is preserving and normalized as required in Theorem 2. $R$ is ground and consequently, left-linear. There is only one critical pair $Q(b) \leftarrow .$, which is convergent. $P(f(a), f(a)) \in \text{Mod}(\text{Prog})$ and $P(f(a), f(a)) \rightarrow_R P(f(b), f(a))$. However $P(f(b), f(a)) \notin \text{Mod}(\text{Prog})$.

The copying nature of $\text{Prog}$ is problematic in Example 4 in the sense that it prevents the predicate symbol $P$ from having two different terms right under. Another problem is that a non-right-linear rewrite rule (but left-linear) may generate a copying critical pair, i.e. copying CS-clauses. Consequently, even if the starting program is not copying, it may become copying during the completion algorithm. Example 5 illustrates this problem.

**Example 5.** $\text{Prog} = \{ P(f(x)) \leftarrow Q(x) \cdot Q(a) \leftarrow . \}$ and $R = \{ f(x) \rightarrow g(x, x) \}$. There is one critical pair $P(g(x, x)) \leftarrow Q(x)$, which is copying. Thus $\text{comp}_R(\text{Prog})$ is copying.

Actually, the hypotheses of Definition 4 have to be stronger i.e. the TRS must be linear in order to prevent the introduction of copying clauses by the completion process, and the starting program must be non-copying. On the other hand, the preserving assumption is not needed anymore.

**Definition 5 (New comp).** Let arity-limit and predicate-limit be positive integers. Let $R$ be a linear rewrite system, and $\text{Prog}$ be a finite, normalized and non-copying CS-program. The completion process is defined by:

Function $\text{comp}_R(\text{Prog})$

- $\text{Prog} = \text{removeCycles}(\text{Prog})$
- while there exists a non-convergent critical pair $H \leftarrow B$ in $\text{Prog}$ do
  - $\text{Prog} = \text{removeCycles}(\text{Prog} \cup \text{norm}_{\text{Prog}}(H \leftarrow B))$
- end while
- return $\text{Prog}$

Thus, a new version of Theorem 2 is given below:

**Theorem 4.** Let $R$ be a left-linear\(^7\) rewrite system and $\text{Prog}$ be a normalized non-copying CS-program.

If all critical pairs are convergent, then $\text{Mod}(\text{Prog})$ is closed under rewriting by $R$, i.e. $(A \in \text{Mod}(\text{Prog}) \land A \rightarrow_R A') \implies A' \in \text{Mod}(\text{Prog})$.

Proof. Let $A \in \text{Mod}(\text{Prog})$ s.t. $A \rightarrow_{\rightarrow_R} A'$. Then $A_i = C[\sigma(l)]$ for some $i \in \mathbb{N}$ and $A' = A[i \leftarrow C[\sigma(r)]]$.

Since resolution is complete, $A \sim^* \emptyset$. Since $\text{Prog}$ is normalized, resolution consumes symbols in $C$ one by one, thus $G_0 = A \sim^* G_k \sim^* \emptyset$ and there exists

\(^7\)From a theoretical point of view, left-linearity is sufficient when every critical pair is convergent. However, to make every critical pair convergent by completion, full linearity is necessary (see Theorem 5).
an atom $A'' = P(t_1, \ldots, t_n)$ in $G_k$ and $j$ s.t. $t_j = \sigma(l)$ and the top symbol of $t_j$ is consumed (or $t_j$ disappears) during the step $G_k \rightsquigarrow G_{k+1}$. Since $Prog$ is non-copying, $t_j = \sigma(l)$ admits only one antecedent in $A$.

Consider new variables $x_1, \ldots, x_n$ s.t. $\{x_1, \ldots, x_n\} \cap Var(l) = \emptyset$, and let us define the substitution $\sigma'$ by $\forall i, \sigma'(x_i) = t_i$ and $\forall x \in Var(l), \sigma'(x) = \sigma(x)$.

Then $\sigma'(P(x_1, \ldots, x_{j-1}, l, x_{j+1}, \ldots, x_n)) = A''$, and according to resolution (or narrowing) properties $P(x_1, \ldots, l, \ldots, x_n) \sim_{\emptyset}^* \emptyset$ and $\theta \leq \sigma'$.

This derivation can be decomposed into: $P(x_1, \ldots, l, \ldots, x_n) \sim_{\emptyset}^* G' \sim_{\theta_2} G \sim_{\emptyset}^* \emptyset$ where $\theta = \theta_3 \circ \theta_2 \circ \theta_1$, and s.t. $G'$ is not flat and $G$ is flat\(^8\). $P(x_1, \ldots, l, \ldots, x_n) \sim_{\emptyset}^* G' \sim_{\theta_2} G$ can be commuted into $P(x_1, \ldots, l, \ldots, x_n) \sim_{\emptyset}^* B' \sim_{\gamma_2} G$ s.t. $B$ is flat, $B'$ is not flat, and within $P(x_1, \ldots, l, \ldots, x_n) \sim_{\emptyset}^* B' \sim_{\gamma_2} B$ resolution is applied only on non-flat atoms, and we have $\gamma_3 \circ \gamma_2 \circ \gamma_1 = \theta_2 \circ \theta_1$. Then $\gamma_2 \circ \gamma_1 P(x_1, r, \ldots, x_n))$ $B$ is a critical pair. By hypothesis, it is convergent, then $\gamma_2 \circ \gamma_1 P(x_1, r, \ldots, x_n)) \rightarrow^* B$. Note that $\gamma_3(B) \rightarrow^* G$ and recall that $\theta_3 \circ \gamma_2 \circ \gamma_1 = \theta_3 \circ \theta_2 \circ \theta_1 = \theta$. Then $\theta(P(x_1, r, \ldots, x_n)) \rightarrow^* \theta_3(G) \rightarrow^* 0$, and since $\theta \leq \sigma'$ we get $P(t_1, \ldots, \sigma(r), \ldots, t_n) = \sigma(P(x_1, \ldots, r, \ldots, x_n)) \rightarrow^* 0$. Recall that $\sigma(l)$ has only one antecedent in $A$. Therefore $A' \sim^* G_k A'' \rightarrow^* P(t_1, \ldots, \sigma(r), \ldots, t_n) \sim^* \emptyset$, hence $A' \in Mod(Prog)$.

By trivial induction, the proof can be extended to the case of several rewrite steps.

Consequently, one can update Theorem 3 as follows:

**Theorem 5.** Let $R$ be a linear rewrite system and $Prog$ be a normalized non-copying CS-program. Function $\text{comp}_R$ always terminates, and all critical pairs are convergent in $\text{comp}_R(Prog)$. Moreover, $R^*(\text{Mod}(Prog)) \subseteq \text{Mod}(\text{comp}_R(Prog))$.

**References**


---

\(^8\) Since $\emptyset$ is flat, a flat goal can always be reached, i.e. in some cases $G = \emptyset$. 