4 rue Léonard de Vinci
BP 6759
F-45067 Orléans Cedex 2
FRANCE
http://www.univ-orleans.fr/lifo

Rapport de Recherche

Synchronized-ContextFree Tree-tuple Languages

Jacques Chabin, Jing Chen, Pierre Réty
LIFO, Université d'Orléans

Rapport n^o **RR-2006-13**

Synchronized-ContextFree Tree-tuple Languages

Jacques Chabin Jing Chen Pierre Réty

LIFO - Université d'Orléans, B.P. 6759, 45067 Orléans cedex 2, France
E-mail : {chabin, chen, rety}@univ-orleans.fr
http://www.univ-orleans.fr/SCIENCES/LIFO/Members/rety

Abstract. Using logic programs, we define a new class of tree-(tuple) languages, that combines the synchronized-regular tree-(tuple) languages with the context-free tree languages. Thus, this class can express an unbounded counting both in several independent branches (like in a synchronized-regular language), and at two nested positions (like in a context-free language). Despite its bigger expressivity, the new class has the same useful properties as the former ones : closure under union, closure under intersection of one tuple-component with a regular tree language, decidability of membership and emptiness. Therefore, it should give rise to many applications.

Keywords : tree languages, logic programming, rewriting.

1 Introduction

Tree languages allow to express, manipulate, decide properties over infinite sets of first-order terms. Tree-tuple languages allow in addition to deal with relations over terms. This is why they are very useful in many domains of theoretical computer science, like automated deduction (term rewriting [2], one-step rewriting logic [8], equational unification), concurrency [12, 5] (process algebra), protocol verification [4].

Various classes of tree-(tuple) languages have been defined. The simplest class is the class of regular tree languages, defined by finite tree automata. It can be easily extended to tuples in the following way : a tuple language is regular (called *regular relation*) if the tree language obtained by overlapping the tuple components to get single terms, is regular (see [2] for a survey).

Several extensions of regular tree-(tuple) languages, i.e. classes that strictly contain regular languages, have been proposed, in particular (see Example 1 below) :

- The class of *context-free tree languages* [15]. Context-free tree languages have the same properties as context-free string languages : closure under union, closure under intersection with a regular language, decidability of membership and emptiness. They can express an unbounded counting at two nested positions.

- By introducing synchronization constraints when deriving a regular grammar. Such languages are called *synchronized-regular tree-tuple languages*, and were first presented using a grammar formalism, called Tree-Tuple Synchronized Grammars (TTSG) [7] (see also [13]), then improved using a constraint system formalism [5, 9], and finally extended with the possibility of copying a given subterm into several branches, using a logic programming formalism [11, 10]. This class is closed under union, under intersection of one tuple-component with a regular language, membership and emptiness are decidable. It can express an unbounded counting in several independent branches.

Note that context-free tree languages and synchronized-regular tree-tuple languages have the same properties, in the case of tuples of arity one. On the other hand, the class defined by primal grammars [6] is expressive, but does not contain regular languages.

In this paper, we define a new class of tree-(tuple) languages, called *synchronized-contextfree*[1] *tree-tuple languages*, by mixing the synchronized-regular tree-tuple languages up with the context-free tree languages. It can express an unbounded counting both in several independent branches and at two nested positions. We prove[2] that the properties of the former classes are preserved despite the bigger expressivity. Proving decidability of emptiness is not trivial, but the biggest difficulty is for proving closure under intersection of one tuple-component with a regular tree language. Then decidability of membership is a consequence, and closure under union is trivial. All these properties are necessary to have interesting applications. As in [11, 10], we use logic programs instead of grammars.

An application of the new class to equational unification is outlined in [14].

Example 1. (for $n \in \mathbb{N}$, $h^n(a)$ is the term $h(h(...h(a)))$ where h occurs n times)

$$L_1 = \left\{ \begin{array}{cc} \multicolumn{2}{c}{c} \\ / & \backslash \\ f^n & g^k \\ | & | \\ h^n & i^k \\ | & | \\ a & b \end{array} \;\middle|\; n, k \in \mathbb{N} \right\} \qquad L_2 = \left\{ \begin{array}{cc} \multicolumn{2}{c}{c} \\ / & \backslash \\ f^n & g^n \\ | & | \\ h^k & i^k \\ | & | \\ a & b \end{array} \;\middle|\; n, k \in \mathbb{N} \right\} \qquad L_3 = \left\{ \begin{array}{cc} \multicolumn{2}{c}{c} \\ / & \backslash \\ f^n & g^n \\ | & | \\ h^n & i^n \\ | & | \\ a & b \end{array} \;\middle|\; n \in \mathbb{N} \right\}$$

L_1 is context-free, L_2 is synchronized-regular, L_3 is synchronized-contextfree.

2 Definitions

Definition 1. *A* logic program with modes *is a logic program such that a mode-tuple $\vec{m} \in \{I, O\}^n$ is associated to each predicate symbol P (n is the arity of P). In other words, each predicate argument has mode I (Input) or O (Output). To distinguish them,* **output arguments will be covered by a hat**.

[1] The new class is not called synchronized context-free, because in the general case, it is not context-free (nor synchronized-regular). On the other hand synchronized-context-free is misleading. This is why we write synchronized-contextfree.

[2] In a constructive way, i.e. we give algorithms.

Notation : Let P be a predicate symbol. $ArIn(P)$ is the number of input arguments of P, and $ArOut(P)$ is the number of output arguments.

Notation : Let B be a list of atoms (possibly containing only one atom). $Input(B)$ is the input part of B, i.e. the tuple composed of the input arguments of B. $ArIn(B)$ is the arity of $Input(B)$. $VarIn(B)$ is the set of variables that appear in $Input(B)$.
$Output(B)$, $ArOut(B)$, and $VarOut(B)$ are defined in a similar way.
We also define $Var(B) = VarIn(B) \cup VarOut(B)$.

Definition 2. *Let $B = A_1, \ldots, A_n$ be a list of atoms. We say that $A_j > A_k$ (possibly $j = k$) if $\exists y \in VarIn(A_j) \cap VarOut(A_k)$. In other words an input of A_j depends on an output of A_k.*
We say that B has a cycle *if $A_j >^+ A_j$ for some A_j ($>^+$ is the transitive closure of $>$).*
We say that the clause $H \leftarrow B$ has a cycle *if B has a cycle.*

Example 2. $Q(\widehat{x}, s(y))$, $R(\widehat{y}, s(x))$ (x, y are variables) has a cycle because $Q > R > Q$.

Definition 3. *A* Synchronized-ContextFree program (S-CF program) *Prog is a logic program with modes, whose clauses $H \leftarrow B$ satisfy :*

- *$Input(H).Output(B)$ (. is the tuple concatenation) is a linear tuple of variables, i.e. each tuple-component is a variable, and each variable occurs only once,*
- *and B does not have a cycle.*

Given a predicate symbol P without input arguments, the tree-(tuple) language generated by P is $L(P) = \{\vec{t} \in (T_\Sigma)^{ArOut(P)} \mid P(\vec{t}) \in Mod(Prog)\}$, where T_Σ is the set of ground terms over the signature Σ and $Mod(Prog)$ is the least Herbrand model of Prog. $L(P)$ is called Synchronized-ContextFree language (S-CF language).

If in addition every clause $H \leftarrow B$ of Prog satisfies :

- *$Output(H).Input(B)$ (. is the tuple concatenation) is a linear tuple, i.e. each variable of $Output(H).Input(B)$ occurs only once,*

then Prog is said to be non-copying.

Remark 4. - If a predicate symbol P has no input argument, this means that the outputs of P do not depend on a given input value (like in a regular language). Therefore, a S-CF program whose predicate symbols do not have input arguments, is called *synchronized-regular program*. It generates a synchronized-regular tree-(tuple) language. This class of languages has already been studied in [11, 10][3].

[3] For historical reasons, synchronized-regular programs are called cs-programs in those papers.

- If a predicate symbol P has several output arguments, this means that these output arguments are linked (synchronized) together. Therefore, a S-CF program whose predicate symbols have only one output argument, is called *context-free program* (without synchronization). It generates a context-free tree language.
- A non-copying context-free program whose predicate symbols have no input arguments, is called *regular program.* It generates a regular tree language.

Example 3. (x, y are variables)

$$Prog = \{P(\widehat{x}, y) \leftarrow P(\widehat{s(x)}, y)\}$$

is not a S-CF program because $Input(H).Output(B) = (y, s(x))$ is not a tuple of variables.

$$Prog' = \{P'(\widehat{s(x)}, y) \leftarrow P'(\widehat{x}, s(y))\}$$

is a S-CF program because $Input(H).Output(B) = (y, x)$ is a linear tuple of variables, and there is no cycle. $Prog'$ is not copying because $Output(H).Input(B) = (s(x), s(y))$ is a linear tuple.

Example 4.

$$Prog = \{S(\widehat{c(x, y)}) \leftarrow P(\widehat{x}, \widehat{y}, a, b).\;\; P(\widehat{f(x)}, \widehat{g(y)}, x', y') \leftarrow P(\widehat{x}, \widehat{y}, h(x'), i(y')).\;\; P(\widehat{x}, \widehat{y}, x, y) \leftarrow\}$$

is a non-copying S-CF program. Note that $Prog$ is neither a synchronized-regular program, nor a context-free program. The language generated by S is:

$$L(S) = \{\, c(f^n(h^n(a)), g^n(i^n(b))) \mid n \in \mathbb{N}\}$$

Actually, this language is neither a synchronized-regular language, nor a context-free language.

Example 5.

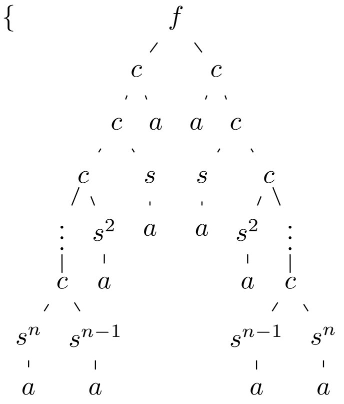
$$Prog = \{S(\widehat{f(x, x)}) \leftarrow S(\widehat{x}).\;\; S(\widehat{a}) \leftarrow\}$$

is a copying context-free program. The language $L(S)$ generated by S is the set of binary balanced terms (all leaves occur at the same depth) over the signature $\Sigma = \{f, a\}$. Actually, $L(S)$ cannot be generated by a non-copying S-CF program.

Here, the notion of "copying" looks like the equality constraints between subterms introduced within finite tree automata [1, 3]. The following example shows that it is more powerful.

Example 6.
$$Prog = \{P(\widehat{f(x,y)}) \leftarrow Q(\widehat{x},\widehat{y},a). \quad Q(\widehat{c(y,x)}, \widehat{c(x,z)}, x) \leftarrow Q(\widehat{y},\widehat{z},s(x)). \quad Q(\widehat{x},\widehat{x},x) \leftarrow\}$$

is a copying S-CF program. The language generated by P is:

$L(P) = \{$  $\mid n \in N\}$

Note that in the leftmost branch, as well as in the rightmost branch, c occurs n times. This language is neither a synchronized-regular language, nor a (even copying) context-free language.

Definition 5. *Let $H \leftarrow B$ be a clause of a S-CF program. The variable x is* undefined *in $H \leftarrow B$ if*
$$x \in (VarOut(H) \cup VarIn(B)) \setminus (VarIn(H) \cup VarOut(B))$$
A clause that has undefined variables is said to be undefined*.*
A S-CF program is completely defined *if it has no undefined clause.*

It is always possible to make a S-CF program completely defined, by adding a new predicate P_Σ and the set of clauses
$$\{P_\Sigma(\widehat{f(x_1,\dots,x_n)}) \leftarrow P_\Sigma(\widehat{x_1}),\dots,P_\Sigma(\widehat{x_n}) \mid f \in \Sigma\}$$
and replacing every undefined clause $H \leftarrow B$ by $H \leftarrow B, P_\Sigma(\widehat{x_1}),\dots,P_\Sigma(\widehat{x_n})$ where $x_1,\dots,x_n$ are the undefined variables of $H \leftarrow B$.

Definition 6. *A regular program Prog is* complete *over the signature Σ if for each $t \in T_\Sigma$, there exists a predicate P s.t. $P(\widehat{t}) \in Mod(Prog)$.*

In a regular program, a clause $P(\widehat{t}) \leftarrow B$ is elementary *if t contains at most one function symbol (the other symbols are variables). $P(\widehat{t}) \leftarrow B$ is* empty *if t is a variable. $P(\widehat{t}) \leftarrow B$ is* without extra variables *if $Var(t) = Var(B)$.*
A regular program is in simple form *if every clause is elementary, not empty, and without extra variables.*

It is always possible to make a regular program complete, by adding a new predicate P_Σ and the set of clauses

$$\{P_\Sigma\left(\begin{array}{c}\widehat{f}\\ \swarrow \ \ \downarrow \ \ \searrow\\ x_1 \ \ \cdots \ \ x_n\end{array}\right) \leftarrow P_\Sigma(\widehat{x_1}),\ldots,P_\Sigma(\widehat{x_n}) \mid f \in \Sigma\}$$

In a regular program, we can make every clause elementary by adding new predicate symbols. Then we can make every clause non-empty by short-circuiting empty clauses.

In a complete regular program, we can remove extra variables in the following way. Let $H \leftarrow B$ be a clause. If $\exists x \in Var(H)\backslash Var(B)$, we replace the clause by $H \leftarrow B, P_\Sigma(\widehat{x})$. If $B = Q_1(\widehat{x_1}),\ldots,Q_n(\widehat{x_n})$ and $x_i \notin Var(H)$, either $L_{Prog}(Q_i)=\emptyset$ and we remove the clause from $Prog$, or we replace the clause by

$$H \leftarrow Q_1(\widehat{x_1}),\ldots,Q_{i-1}(\widehat{x_{i-1}}),Q_{i+1}(\widehat{x_{i+1}}),\ldots,Q_n(\widehat{x_n})$$

Therefore it is always possible to make a regular program complete and in simple form.

3 Intersection with a Regular Language

The intersection of the tree-tuple language E with the tree language E', over the i-th component of E, is denoted and defined by :

$$E \cap_i E' = \{(t_1,\ldots,t_n) \mid (t_1,\ldots,t_n) \in E \wedge t_i \in E'\}$$

Consider a completely defined S-CF program $Prog = (\Sigma, Pred, C)$ where Σ is a signature, $Pred$ is a set of predicate symbols, and C a set of clauses, and consider a complete regular program in simple form $Prog' = (\Sigma, Pred', C')$ over the same signature. We will define the intersection program $Prog'' = (\Sigma, Pred'', C'')$.

3.1 The predicates of the intersection program

The set $Pred''$ of the predicate symbols of the intersection program is defined in the following way. For each $P \in Pred$, for each set $\widehat{S'} \subseteq Pred' \times \{1,\ldots,ArOut(P)\}$, and for each tuple $\vec{Q'} \in Pred'^{ArIn(P)}$, we define the predicate symbol

$$P'' = (P,\, \widehat{S'},\, \vec{Q'})$$

in $Pred''$, that has the same number of output arguments and the same number of input arguments as P.

Remark 7. $Pred''$ is finite since $Pred$ and $Pred'$ are.

Note that P'' is a triple of the form

$$P'' = (P,\, \{(P'_1,i_1),\ldots,(P'_m,i_m)\},\, (Q'_1,\ldots,Q'_{ArIn(P)}))$$

where $P'_1,\ldots,P'_m,Q'_1,\ldots,Q'_{ArIn(P)} \in Pred'$ and $i_1,\ldots,i_m \in \{1,\ldots,ArOut(P)\}$. Intuitively, in an atom $A = P''(\ldots)$ whose predicate symbol is P'' :

- P expresses that the arguments of A should be generated in $Prog$ by predicate P.
- $\{(P'_1, i_1), \ldots, (P'_m, i_m)\}$ expresses that for each $j \in \{1, \ldots, m\}$, the i_j-th output argument of A should be generated in $Prog'$ by predicate P'_j. If $i_j = i_k$ and $j \neq k$, this means that the i_j-th output argument of A should be generated by several predicates of $Prog'$ (P'_j and P'_k). If $n \in \{1, \ldots, ArOut(P)\}$ and $n \notin \{i_1, \ldots, i_m\}$, this means that no constraint in $Prog'$ ranges over the n-th output argument of A.
- $(Q'_1, \ldots, Q'_{ArIn(P)})$ expresses hypotheses (in $Prog'$) over all input arguments of A: for each $j \in \{1, \ldots, ArIn(P)\}$, the j-th input argument of A is supposed to be generated in $Prog'$ by predicate Q'_j.

Notation : $P''|^{Prog}$ is the part of P'' that refers to $Prog$, i.e. $P''|^{Prog} \stackrel{def}{=} P$.

3.2 The clauses of the intersection program

Definition 8. *Given a logic program Prog and a list of atoms L, we say that*

- L *derives into the list of atoms* L' *by a* resolution *step if there exist a clause*[4] $H \leftarrow B$ *in Prog and an atom* $A \in L$ *such that* A *and* H *are unifiable by the most general unifier* σ *(then* $A\sigma = H\sigma$*) and* $L' = (L\sigma)[A\sigma \leftarrow B\sigma]$. *It is written* $L \leadsto^{\sigma} L'$.
- L rewrites *into* L' *if there exist a clause* $H \leftarrow B$ *in Prog, an atom* $A \in L$, *and a substitution* σ, *such that* $A = H\sigma$ *(A is not instantiated by* σ*) and* $L' = L[A \leftarrow B\sigma]$. *It is written* $L \rightarrow^{\sigma} L'$.

We consider the reflexive-transitive closure $\leadsto^*$ *of* $\leadsto$ *and* $\rightarrow^*$ *of* $\rightarrow$.

It is well known that resolution is complete.

Theorem 9. *Let A be a ground atom.* $A \in Mod(Prog)$ *iff* $A \leadsto^*_{Prog} \emptyset$, *where* $\emptyset$ *denotes an empty list of atoms.*

Note that for any atom A, if $A \rightarrow_{Prog} B$ then $A \leadsto_{Prog} B$.
On the other hand, if A is ground, $A \leadsto_{Prog} B$ implies $A \rightarrow_{Prog} B$. In $Prog'$, which is a regular program in simple form, clauses do not have extra-variables. Then A is ground and $A \rightarrow_{Prog'} B$ implies B is ground. Therefore $A \leadsto^*_{Prog'} B$ is equivalent to $A \rightarrow^*_{Prog'} B$. Consequently:

Corollary 10. *Let* $Prog'$ *be a regular program in simple form, and A be a ground atom.* $A \in Mod(Prog')$ *iff* $A \rightarrow^*_{Prog'} \emptyset$.

Notation : $A|^{out}_i$ denotes the i-th output argument of the atom A, and $A|^{in}_i$ denotes the i-th input argument of A, and $A|_\epsilon$ denotes the predicate symbol of A. For example if $A = P(\widehat{s(x)}, y)$, then $A|^{out}_1 = s(x)$, $A|^{in}_1 = y$, and $A|_\epsilon = P$.

[4] We assume that the clause and L have distinct variables.

Notation : Given $P'' \in Pred''$ and A an atom such that $P''|^{Prog} = A|_\epsilon$. We define the new atom $P''[A]$ by

$$P''[A] \; = \; P''(A|_1^{out}, \ldots, A|_{ArOut(A)}^{out}, A|_1^{in}, \ldots, A|_{ArIn(A)}^{in})$$

In other words, $P''[A]$ is obtained from A by replacing the predicate symbol by P''.

Definition 11. *We define the set of clauses PC'' (PC for Potential Clauses) by*

$$PC'' \; = \; \{P_0''[A_0] \leftarrow P_1''[A_1], \ldots, P_p''[A_p] \mid A_0 \leftarrow A_1, \ldots, A_p \in C, \\ P_0'', \ldots, P_p'' \in Pred'', \forall i \, (P_i''|^{Prog} = A_i|_\epsilon)\}$$

PC'' is finite since C and $Pred''$ are. Moreover $(\Sigma, Pred'', PC'')$ is a synchronized-contextfree program. Indeed, clauses in PC'' have the required form, and if a clause of PC'' has a cycle, then the corresponding clause in C also has a cycle, which is impossible.

Definition 12. *Consider an atom A appearing in a clause of PC''. $A|_\epsilon$ is of the form*

$$A|_\epsilon = (P, \, \{(P_1', i_1), \ldots, (P_m', i_m)\}, \, (Q_1', \ldots, Q_{ArIn(P)}'))$$

The list of atoms

$$ConsOut(A) \stackrel{def}{=} P_1'(A|_{i_1}^{out}), \ldots, P_m'(A|_{i_m}^{out})$$

is called output constraint *of A.*
The list of atoms

$$HypIn(A) \stackrel{def}{=} Q_1'(A|_1^{in}), \ldots, Q_{ArIn(P)}'(A|_{ArIn(P)}^{in})$$

is called input hypothesis *of A.*

Notation : If some atom A occurs in a list of atom B, we write $A \in B$.
Consider a list of atoms B_i for each $i \in \{0, \ldots, p\}$. The list of atoms obtained by concatening B_0 and $\ldots$ and B_p is denoted $B_0, \ldots, B_p$.

Definition 13. *Let $C'' \subseteq PC''$ be the set composed of all clauses $H \leftarrow A_1, \ldots, A_p \in PC''$ (poss. $p = 0$) that satisfy the following three properties :*

- $ConsOut(H) \rightarrow_{Prog'}^* B_0$ *(B_0 may be empty)*
- $\forall i \in \{1, \ldots, p\}$, $HypIn(A_i) \rightarrow_{Prog'}^* B_i$ *(B_i may be empty)*
- $\forall A' \in B_0, \ldots, B_p \, (A' \in HypIn(H) \vee \exists j \in \{1, \ldots, p\}, A' \in ConsOut(A_j))$[5].

Remark 14. Since $Prog'$ is a regular program in simple form, clauses of $Prog'$ are not empty and do not have extra variables. And note that in a regular program, arguments of bodies are variables. Therefore $\rightarrow_{Prog'}$ terminates. Thus, the conditions required in the above definition are decidable.

[5] Consequently, the argument of A' is a variable because $(\Sigma, Pred'', PC'')$ is a synchronized-contextfree program.

Remark 15. For readability, predicates of $Pred''$

$$(P,\ \{(P'_1, i_1), \ldots, (P'_m, i_m)\},\ (Q'_1, \ldots, Q'_{ArIn(P)}))$$

will be sometimes denoted by $P^{\{P'_1:i_1,\ldots,P'_m:i_m\}}_{(Q'_1,\ldots,Q'_{ArIn(P)})}$.

Example 7. Let $Prog$ and $Prog'$ be defined be the sets of predicates $Pred = \{P, Q, R\}$ and $Pred' = \{P', Q'\}$, and the sets of clauses:

$$C = \{P(\widehat{x}) \leftarrow Q(\widehat{x}, y), R(\widehat{y}).\quad Q(\widehat{c(y,x)}, x) \leftarrow Q(\widehat{y}, s(x)).\quad Q(\widehat{x}, x) \leftarrow .\quad R(\widehat{a}) \leftarrow\}$$

$$C' = \{P'(\widehat{c(y,x)}) \leftarrow P'(\widehat{y}), Q'(\widehat{x}).\quad P'(\widehat{c(y,x)}) \leftarrow Q'(\widehat{y}), Q'(\widehat{x}).$$

$$Q'(\widehat{s(x)}) \leftarrow Q'(\widehat{x}).\quad Q'(\widehat{a}) \leftarrow\}$$

Note that $Prog$ is copying. Then ($s^*(a)$ is the term $s(s(\ldots s(a)))$ where s occurs arbitrarily many times)

$$L(P) = \{\ c(c(c(\cdots c(s^n(a), s^{n-1}(a)) \cdots, s^2(a)), s(a)), a)\ \mid n \in N\}$$

$$L(P') = \{\ c(c(c(\cdots c(s^*(a), s^*(a)) \cdots, s^*(a)), s^*(a)), s^*(a))\ \mid c \text{ occurs at least once}\}$$

$$Pred'' = \{P^{\{P':1\}}_{()},\ P^{\{Q':1\}}_{()},\ P^{\{P':1,Q':1\}}_{()},\ Q^{\{P':1\}}_{(P')},\ Q^{\{Q':1\}}_{(P')},\ Q^{\{P':1,Q':1\}}_{(P')}$$
$$Q^{\{P':1\}}_{(Q')},\ Q^{\{Q':1\}}_{(Q')},\ Q^{\{P':1,Q':1\}}_{(Q')},\ R^{\{P':1\}}_{()},\ R^{\{Q':1\}}_{()},\ R^{\{P':1,Q':1\}}_{()}\}$$

The clauses of C'' needed to compute $L(P) \cap L(P')$ are:

- $P^{\{P':1\}}_{()}(\widehat{x}) \leftarrow Q^{\{P':1\}}_{(Q')}(\widehat{x}, y),\ R^{\{Q':1\}}_{()}(\widehat{y})$ because
  $ConsOut(P^{\{P':1\}}_{()}(\widehat{x})) = P'(\widehat{x}) \in ConsOut(Q^{\{P':1\}}_{(Q')}(\widehat{x}, y))$,
  $HypIn(Q^{\{P':1\}}_{(Q')}(\widehat{x}, y)) = Q'(\widehat{y}) \in ConsOut(R^{\{Q':1\}}_{()}(\widehat{y}))$.
- $Q^{\{P':1\}}_{(Q')}(\widehat{c(y,x)}, x) \leftarrow Q^{\{P':1\}}_{(Q')}(\widehat{y}, s(x))$ because
  $ConsOut(Q^{\{P':1\}}_{(Q')}(\widehat{c(y,x)}, x)) = P'(\widehat{c(y,x)}) \rightarrow_{Prog'} P'(\widehat{y}), Q'(\widehat{x})$

and $P'(\widehat{y}) \in ConsOut(Q^{\{P':1\}}_{(Q')}(\widehat{y}, s(x)))$ and $Q'(\widehat{x}) \in HypIn(Q^{\{P':1\}}_{(Q')}(\widehat{c(y,x)}, x))$,

$$HypIn(Q^{\{P':1\}}_{(Q')}(\widehat{y}, s(x))) = Q'(\widehat{s}(x)) \to_{Prog'} Q'(\widehat{x}) \in HypIn(Q^{\{P':1\}}_{(Q')}(\widehat{c(y,x)}, x)).$$

- In the same way $Q^{\{P':1\}}_{(Q')}(\widehat{c(y,x)}, x) \leftarrow Q^{\{Q':1\}}_{(Q')}(\widehat{y}, s(x))$ because

$$ConsOut(Q^{\{P':1\}}_{(Q')}(\widehat{c(y,x)}, x)) = P'(\widehat{c(y,x)}) \to_{Prog'} Q'(\widehat{y}), Q'(\widehat{x})$$

and $Q'(\widehat{y}) \in ConsOut(Q^{\{Q':1\}}_{(Q')}(\widehat{y}, s(x)))$ and $Q'(\widehat{x}) \in HypIn(Q^{\{P':1\}}_{(Q')}(\widehat{c(y,x)}, x))$,

$$HypIn(Q^{\{Q':1\}}_{(Q')}(\widehat{y}, s(x))) = Q'(\widehat{s}(x)) \to_{Prog'} Q'(\widehat{x}) \in HypIn(Q^{\{P':1\}}_{(Q')}(\widehat{c(y,x)}, x)).$$

- $Q^{\{Q':1\}}_{(Q')}(\widehat{x}, x) \leftarrow .$ because
  $ConsOut(Q^{\{Q':1\}}_{(Q')}(\widehat{x}, x)) = Q'(\widehat{x}) \in HypIn(Q^{\{Q':1\}}_{(Q')}(\widehat{x}, x)).$
- $R^{\{Q':1\}}_{()}(\widehat{a}) \leftarrow .$ because
  $ConsOut(R^{\{Q':1\}}_{()}(\widehat{a})) = Q'(\widehat{a}) \to_{Prog'} \emptyset.$

Note that $Q^{\{P':1\}}_{(Q')}(\widehat{x}, x) \leftarrow . \notin C''$ because

$$ConsOut(Q^{\{P':1\}}_{(Q')}(\widehat{x}, x)) = P'(\widehat{x}) \notin HypIn(Q^{\{P':1\}}_{(Q')}(\widehat{x}, x))$$

Consequently $L(P) \cap L(P')$ is the same language as $L(P)$, except that c occurs at least once, i.e. $a \notin L(P) \cap L(P')$.

We extend the definition of the language generated by a predicate symbol P, to the case where P has input arguments, by giving initial values to them.

Definition 16. *Consider a logic program with modes Prog and a signature Σ. The language generated by the predicate symbol P with input values: $t_1, \dots, t_{ArIn(P)} \in T_\Sigma$, is defined by*

$$L^\Sigma_{Prog}(P, (t_1, \dots, t_{ArIn(P)})) = \{(s_1, \dots, s_{ArOut(P)}) \in (T_\Sigma)^{ArOut(P)} \mid P(\widehat{s_1, \dots, s_{ArOut(P)}}, t_1, \dots, t_{ArIn(P)}) \in Mod(Prog)\}$$

Lemma 17. *Recall that $Prog'' = (\Sigma, Pred'', C'')$.*
Let $P'' = (P, \{(P'_1, i_1), \dots, (P'_m, i_m)\}, \vec{Q'}) \in Pred''$, and let $\vec{t} \in (T_\Sigma)^{ArIn(P)}$ that satisfies the input hypothesis of P'', i.e. such that $HypIn(P''(\widehat{\dots}, \vec{t})) \to^*_{Prog'} \emptyset$. Then

$$L^\Sigma_{Prog''}(P'', \vec{t}) = (\dots (L^\Sigma_{Prog}(P, \vec{t}) \cap_{i_1} L^\Sigma_{Prog'}(P'_1)) \cap_{i_2} \dots) \cap_{i_m} L^\Sigma_{Prog'}(P'_m)$$

A synchronized-contextfree language is generated by a predicate P without input arguments. If we want to compute $L^\Sigma_{Prog}(P) \cap_i L^\Sigma_{Prog'}(P')$, the previous lemma gives:

Corollary 18. *Let* $P'' = (P, \{(P', i)\}) \in Pred''$ *such that* P *has no input arguments. Then*

$$L^{\Sigma}_{Prog''}(P'') = L^{\Sigma}_{Prog}(P) \cap_i L^{\Sigma}_{Prog'}(P')$$

3.3 Proof of lemma 17

The proof is just outlined here. See the appendix for the full proof.

Soundness We have to prove:

$$L^{\Sigma}_{Prog''}(P'', \vec{t}) \subseteq$$
$$(\dots(L^{\Sigma}_{Prog}(P, \vec{t}) \cap_{i_1} L^{\Sigma}_{Prog'}(P'_1)) \cap_{i_2} \dots) \cap_{i_m} L^{\Sigma}_{Prog'}(P'_m)$$

Let $\vec{s} = (s_1, \dots, s_{ArOut(P)}) \in (T_\Sigma)^{ArOut(P)}$. From Theorem 9 and Corollary 10, we have to prove that

$$P''(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow^*_{Prog''} \emptyset \ \wedge \ HypIn(P''(\widehat{\vec{s}}, \vec{t})) \rightarrow^*_{Prog'} \emptyset$$

implies $\quad P(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow^*_{Prog} \emptyset \wedge P'_1(\widehat{s_{i_1}}) \rightarrow^*_{Prog'} \emptyset \wedge \dots \wedge P'_m(\widehat{s_{i_m}}) \rightarrow^*_{Prog'} \emptyset$

i.e. $\quad P(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow^*_{Prog} \emptyset \wedge ConsOut(P''(\widehat{\vec{s}}, \vec{t})) \rightarrow^*_{Prog'} \emptyset$.

The proof is done by induction on the length n of the rewrite derivation

$$P''(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow^*_{Prog''} \emptyset$$

This derivation necessarily contains at least one step.

- If it contains exactly one step, the clause used in this step is a fact.
- Otherwise, the derivation contains several steps. Let us detail the first step:

$$P''(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow_{Prog''} P''_1(\widehat{\vec{s^1}}, \vec{t^1}), \dots, P''_k(\widehat{\vec{s^k}}, \vec{t^k}) \ \rightsquigarrow^*_{Prog''} \emptyset$$

and let $H \leftarrow A_1, \dots, A_k$ be the clause and σ the substitution used during this step. Since there is no cycle in the clause body, we can suppose without loss of generality that $\forall i, j\, (i \leq j \implies A_i \not> A_j)$. The clause is of the form $P''(\widehat{\vec{s'}}, \vec{t'}) \leftarrow P''_1(\widehat{\vec{s'^1}}, \vec{t'^1}), \dots, P''_k(\widehat{\vec{s'^k}}, \vec{t'^k})$ with $P''(\widehat{\vec{s}}, \vec{t})\sigma = H\sigma = P''(\widehat{\vec{s'}}, \vec{t'})\sigma$ and $\forall i \in \{1, \dots, k\}\ P''_i(\widehat{\vec{s^i}}, \vec{t^i}) = A_i\sigma = P''_i(\widehat{\vec{s'^i}}, \vec{t'^i})\sigma$.

The proof consists in: for $i = 1$ to k (in this order),

1. we prove that the input hypothesis of $P''_i(\widehat{\vec{s^i}}, \vec{t^i})$ is satisfied, i.e.

$$HypIn(P''_i(\widehat{\vec{s^i}}, \vec{t^i})) \rightarrow^*_{Prog'} \emptyset$$

2. from $P''_1(\widehat{\vec{s^1}}, \vec{t^1}), \dots, P''_k(\widehat{\vec{s^k}}, \vec{t^k}) \rightsquigarrow^*_{Prog''} \emptyset$, we can obviously extract a subderivation $P''_i(\widehat{\vec{s^i}}, \vec{t^i}) \rightsquigarrow^*_{Prog''} \emptyset$ whose length is strictly less than n. We can then apply the induction hypothesis on this subderivation.

Completeness We have to prove:

$$L^{\Sigma}_{Prog''}(P'', \vec{t}) \supseteq$$
$$(\ldots(L^{\Sigma}_{Prog}(P, \vec{t}) \cap_{i_1} L^{\Sigma}_{Prog'}(P'_1)) \cap_{i_2} \ldots) \cap_{i_m} L^{\Sigma}_{Prog'}(P'_m)$$

Let $\vec{s} = (s_1, \ldots, s_{ArOut(P)}) \in (T_{\Sigma})^{ArOut(P)}$. From Theorem 9, we have to prove that

$$P(\widehat{\vec{s}, \vec{t}}) \rightsquigarrow^*_{Prog} \emptyset \,\wedge\, P'_1(\widehat{s_{i_1}}) \rightarrow^*_{Prog'} \emptyset \,\wedge\, \ldots \,\wedge\, P'_m(\widehat{s_{i_m}}) \rightarrow^*_{Prog'} \emptyset$$
$$\wedge\, HypIn(P''(\widehat{\vec{s}, \vec{t}})) \rightarrow^*_{Prog'} \emptyset$$

implies

$$P''(\widehat{\vec{s}, \vec{t}}) \rightsquigarrow^*_{Prog''} \emptyset$$

Let us write $\vec{t} = (t_1, \ldots, t_{ArIn(P)})$.
Since $HypIn(P''(\widehat{\vec{s}, \vec{t}})) = Q'_1(t_1), \ldots, Q'_{ArIn(P)}(t_{ArIn(P)})$, we have

$$HypIn(P''(\widehat{\vec{s}, \vec{t}})) \rightarrow^*_{Prog'} \emptyset \iff$$
$$Q'_1(t_1) \rightarrow^*_{Prog'} \emptyset \,\wedge\, \ldots \,\wedge\, Q'_{ArIn(P)}(t_{ArIn(P)}) \rightarrow^*_{Prog'} \emptyset$$

The proof is done by induction on the length n of the rewrite derivation

$$P(\widehat{\vec{s}, \vec{t}}) \rightsquigarrow^*_{Prog} \emptyset$$

• If it contains exactly one step, the clause used in this step is a fact.

• Otherwise, the derivation contains several steps. Let us detail the first step:

$$P(\widehat{\vec{s}, \vec{t}}) \rightsquigarrow_{Prog} P_1(\widehat{\vec{s^1}, \vec{t^1}}), \ldots, P_k(\widehat{\vec{s^k}, \vec{t^k}}) \rightsquigarrow^*_{Prog} \emptyset$$

So there exist in C a clause $P(\widehat{\vec{s'}, \vec{t'}}) \leftarrow P_1(\widehat{\vec{s'^1}, \vec{t'^1}}), \ldots, P_k(\widehat{\vec{s'^k}, \vec{t'^k}})$ (also noted $H \leftarrow A_1, \ldots, A_k$) and a substitution σ such that $P(\widehat{\vec{s}, \vec{t}})\sigma = P(\widehat{\vec{s'}, \vec{t'}})\sigma$ and $\forall i \in \{1, \ldots, k\}$, $P_i(\widehat{\vec{s'^i}, \vec{t'^i}})\sigma = P_i(\widehat{\vec{s^i}, \vec{t^i}})$. Since there is no cycle in the clause body, we can suppose without loss of generality that $\forall i, j\, (i \leq j \implies A_i \not> A_j)$.

4 Decidability of Emptiness

Lemma 19. *Let $P(\widehat{\vec{s}, \vec{t}}) \leftarrow P_1(\widehat{\vec{s^1}, \vec{t^1}}), \ldots, P_n(\widehat{\vec{s^n}, \vec{t^n}})$ be a clause (also denoted $H \leftarrow A_1, \ldots, A_n$) of a synchronized-contextfree language Prog.*
If $\forall i \in \{1, \ldots, n\}$, $\forall \vec{t'^i} \in T_{\Sigma}^{ArIn(P_i)}$, $\exists \vec{s'^i} \in T_{\Sigma}^{ArOut(P_i)}$, $P_i(\widehat{\vec{s'^i}, \vec{t'^i}}) \in Mod(Prog)$ then $\forall \vec{t'} \in T_{\Sigma}^{ArIn(P)}$, $\exists \vec{s'} \in T_{\Sigma}^{ArOut(P)}$, $P(\widehat{\vec{s'}, \vec{t'}}) \in Mod(Prog)$.

Proof. Since there is no cycle in the clause body, we can suppose without loss of generality that $\forall i, j\, (i \leq j \implies A_i \not> A_j)$. Let $\vec{t'} \in T_{\Sigma}^{ArIn(P)}$ be a vector of ground terms and t_0 an arbitrary ground term of T_{Σ}.

• If $n = 0$, the clause is the fact $P(\widehat{\vec{s}, \vec{t}}) \leftarrow$ and all components of $\vec{t}$ are variables (input part of head of clause) then there exists σ such that $\vec{t}\sigma = \vec{t'}$. Let us consider the substitution σ_0 defined $\forall x \in Var(\vec{s}) \setminus Var(\vec{t})$ by $x\sigma_0 = t_0$. Then $\vec{s'} = \vec{s}\sigma\sigma_0$ is a ground vector of $T_\Sigma^{ArOut(P)}$ such that $P(\widehat{\vec{s'}, \vec{t'}}) \in Mod(Prog)$.

• Otherwise we build a ground substitution by scanning the clause body, atom by atom, from left to right.

- For A_1, we define the substitution σ_1 for all $x \in Var(A_1)$ by :
  - if $x \in VarIn(A_1)$ then $x \in VarIn(H)$ (otherwise x would not be defined), suppose it occurs as the p^{th} component of $\vec{t}$. We define $x\sigma_1 = \vec{t'}|_p$.
  - else $x \in VarOut(A_1)$, from the hypothesis, it exists a ground term $\vec{s'^1}$ such that $P_1(\widehat{\vec{s'^1}, t^1\sigma_1}) \in Mod(Prog)$. Since $VarIn(A_1) \cap VarOut(A_1) = \emptyset$ (no cycle), we can extend σ_1 by $\vec{s^1}\sigma_1 = \vec{s'^1}$ ($\vec{s^1} = VarOut(A_1)$ is a linear tuple of variables).

  Thus, we have built a substitution σ_1 such that $P_1(\widehat{\vec{s^1}, t^1})\sigma_1 \in Mod(Prog)$.
- For A_2, we define the substitution σ_2 for all $x \in Var(A_2)$ by:
  - if $x \in VarIn(A_2) \cap VarOut(A_1)$, $x\sigma_2 = x\sigma_1$,
  - else if $x \in VarIn(A_2)$ then $x \in VarIn(H)$, (otherwise x would not be defined), suppose it occurs as the p^{th} component of $\vec{t}$. We define $x\sigma_2 = \vec{t'}|_p$,
  - else $x \in VarOut(A_2)$, from definition of the language $VarOut(A_1, \ldots, A_n)$ is a linear tuple of variables, so $x \notin VarOut(A_1)$. Moreover $x \notin VarIn(A_1)$ (no cycle). Then $P_2(\widehat{\vec{s^2}, \vec{t^2}})\sigma_1\sigma_2 = P_2(\widehat{\vec{s^2}, \vec{t^2}\sigma_1\sigma_2})$, from the hypothesis, it exists a ground term $\vec{s'^2}$ such that $P_2(\widehat{\vec{s'^2}, \vec{t^2}\sigma_1\sigma_2}) \in Mod(Prog)$. We can extend σ_2 by $\vec{s^2}\sigma_2 = \vec{s'^2}$ ($\vec{s^2} = VarOut(A_2)$ is a linear tuple of variables).

  Thus, we have built a substitution σ_2 such that $P_2(\widehat{\vec{s^2}, t^2})\sigma_1\sigma_2 \in Mod(Prog)$. Since $P_1(\widehat{\vec{s^1}, t^1})\sigma_1$ is ground, we also have $P_1(\widehat{\vec{s^1}, t^1})\sigma_1\sigma_2 \in Mod(Prog)$.
- For A_3 until A_n we apply the same reasoning.

Therefore we obtain $\forall j \in \{1, \ldots, n\}$ $A_j\sigma_1 \ldots \sigma_n \in Mod(Prog)$.

Finally let θ be a substitution defined for all $x \in Var(H) \setminus \{Var(A_1) \cup \ldots \cup Var(A_n)\}$ by

- if $x \in VarIn(H) \setminus \{Var(A_1) \cup \ldots \cup Var(A_n)\}$, suppose it occurs as the p^{th} component of $\vec{t}$. We define $x\theta = \vec{t'}|_p$.
- otherwise $x \in VarOut(H) \setminus \{Var(A_1) \cup \ldots \cup Var(A_n)\}$, which is impossible because $Prog$ is completely defined.

We conclude that

$$\forall j \in \{1, \ldots, n\}\ A_j\sigma_1 \ldots \sigma_n\theta \in Mod(Prog)$$

Therefore $H\sigma_1 \ldots \sigma_n\theta \in Mod(Prog)$ since $H \leftarrow A_1, \ldots, A_n$ is in $Prog$. By construction we have $\vec{t}\sigma_1 \ldots \sigma_n\theta = \vec{t'}$ and $\vec{s}\sigma_1 \ldots \sigma_n\theta = \vec{s'}$ is ground. Then the lemma is proved.

Corollary 20. *Emptiness for S-CF programs, is decidable by the following algorithm:*

- *Stamp "non-empty" onto all predicates used in the facts.*
- *For all clause $H \leftarrow A_1, \ldots, A_n$ in Prog, if $\forall j \in \{1, \ldots, n\}$, $A_j|_\epsilon$ is stamped by "non-empty", stamp the predicate $H|_\epsilon$.*
- *Repeat the last step until no more stamp "non-empty" is added.*

Remark 21. As $Mod(Prog)$ is the least Herbrand model of $Prog$, if a predicate P is not stamped "non-empty" then $Mod(Prog)|_P = \emptyset$.

5 Decidability of Membership

Let $(t_1, \ldots, t_n)$ be a tuple of terms and L be a S-CF language. Then

$$(t_1, \ldots, t_n) \in L \iff (\cdots(L \cap_1 \{t_1\}) \cap_2 \{t_2\}) \ldots\ldots \cap_n \{t_n\}) \neq \emptyset$$

For each i, the singleton $\{t_i\}$ is a regular language (because it is a finite set). Thanks to closure under intersection with a regular language and decidability of emptiness, membership is decidable.

6 Conclusion

The class of synchronized-contextfree tree-tuple languages has both a big expressivity and nice properties. It remains :

- to develop applications to equational unification, to rewriting, to process algebra, to protocol verification, and perhaps to other domains of theoretical computer science.
- to optimize and implement the algorithms.
- to make a precise comparison with primal grammars [6].

References

1. B. Bogaert and S. Tison. Equality and Disequality Constraints on Direct Subterms in Tree Automata. In Proceedings of 9th Annual Symposium on Theoretical Aspects of Computer Science, volume 577 of *LNCS*. Springer-Verlag, 1992.
2. H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications* (TATA) http://www.grappa.univ-lille3.fr/tata.
3. M. Dauchet, A.C. Caron, and J.L. Coquidé. Reduction Properties and Automata with Constraints. *Journal of Symbolic Computation*, pages 20:215–233, 1995.
4. T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *Proceedings 17th International CADE, Pittsburgh (Pen., USA)*, volume 1831 of *LNAI*. Springer-Verlag, 2000. (extended version in Technical Report RR-3921, Inria 2000).
5. V. Gouranton, P. Réty, and H. Seidl. Synchronized Tree Languages Revisited and New Applications. In *Proceedings of 6th Conference on Foundations of Software Science and Computation Structures, Genova (Italy)*, LNCS. Springer, 2001.

6. M. Hermann and R. Galbavý. Unification of Infinite Sets of Terms Schematized by Primal Grammars. *Theoretical Computer Science*, 176, 1997.
7. S. Limet and P. Réty. E-Unification by Means of Tree Tuple Synchronized Grammars. *Discrete Mathematics and Theoritical Computer Science (http://www.dmtcs.loria.fr)*, 1:69–98, 1997.
8. S. Limet and P. Réty. A New Result about the Decidability of the Existential One-Step Rewriting Theory. In *Proceedings of 10th Conference on Rewriting Techniques and Applications, Trento (Italy)*, number 1631 in LNCS. Springer, 1999.
9. S. Limet, P. Réty, and H. Seidl. Weakly Regular Relations and Applications. In *Proceedings of 12th Conference on Rewriting Techniques and Applications, Utrecht (The Netherlands)*, LNCS. Springer, 2001.
10. S. Limet and G. Salzer. Proving properties of term rewrite systems via logic programs. In *proceedings of RTA 2004*, volume 3091 of *LNCS*, pages 170–184. Springer Verlag, 2004.
11. Sébastien Limet and Gernot Salzer. Manipulating tree tuple languages by transforming logic programs. In Ingo Dahn and Laurent Vigneron, editors, *Electronic Notes in Theoretical Computer Science*, volume 86. Elsevier, 2003.
12. D. Lugiez and P. Schnoebelen. The Regular Viewpoint on PA-Processes. *Theoretical Computer Science*, 2000.
13. J.C. Raoult. Rational Tree Relations. *Bulletin of the Belgian Mathematical Society Simon Stevin*, 4:149–176, 1997.
14. P. Réty, J. Chabin, and J. Chen. R-Unification thanks to Synchronized-Contextfree Tree Languages. In *Proceedings of the International Workshop on Unification*, Nara, Japan, 2005.
15. W.-C. Rounds. Mappings and Grammars on Trees. *Mathematical Systems Theory*, 3(4):257–287, 1970.

Appendix
Full Proof of Lemma 17

A Soundness

We have to prove:

$$L^{\Sigma}_{Prog''}(P'', \vec{t}) \subseteq$$
$$(\ldots(L^{\Sigma}_{Prog}(P, \vec{t}) \cap_{i_1} L^{\Sigma}_{Prog'}(P'_1)) \cap_{i_2} \ldots) \cap_{i_m} L^{\Sigma}_{Prog'}(P'_m)$$

Let $\vec{s} = (s_1, \ldots, s_{ArOut(P)}) \in (T_\Sigma)^{ArOut(P)}$. From Theorem 9 and Corollary 10, we have to prove that

$$P''(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow^*_{Prog''} \emptyset \ \wedge \ HypIn(P''(\widehat{\vec{s}}, \vec{t})) \rightarrow^*_{Prog'} \emptyset$$

implies $\quad P(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow^*_{Prog} \emptyset \wedge P'_1(\widehat{s_{i_1}}) \rightarrow^*_{Prog'} \emptyset \wedge \ldots \wedge P'_m(\widehat{s_{i_m}}) \rightarrow^*_{Prog'} \emptyset$

i.e. $\quad P(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow^*_{Prog} \emptyset \wedge ConsOut(P''(\widehat{\vec{s}}, \vec{t})) \rightarrow^*_{Prog'} \emptyset$.

The proof is done by induction on the length n of the rewrite derivation

$$P''(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow^*_{Prog''} \emptyset$$

This derivation necessarily contains at least one step.

• If it contains exactly one step, the clause used in this step is a fact. Then there exist in C'' a clause $P''(\widehat{\vec{s'}}, \vec{t'}) \leftarrow$ and a substitution σ such that $P''(\widehat{\vec{s'}}, \vec{t'})\sigma = P''(\widehat{\vec{s}}, \vec{t})\sigma$. The clause is in C'', from definition 13 we have

$$ConsOut(P''(\widehat{\vec{s'}}, \vec{t'})) \rightarrow^*_{Prog'} B_0 \ \wedge \ \forall A' \in B_0,\, A' \in HypIn(P''(\widehat{\vec{s'}}, \vec{t'}))$$

because the clause is a fact (no body). The rewriting is stable under substitution, so we deduce $ConsOut(P''(\widehat{\vec{s'}}, \vec{t'})\sigma) \rightarrow^*_{Prog'} B_0\sigma$ and $\forall A' \in B_0\sigma, A' \in HypIn(P''(\widehat{\vec{s'}}, \vec{t'})\sigma) = HypIn(P''(\widehat{\vec{s}}, \vec{t})\sigma)$. By hypothesis of the lemma, we know that $HypIn(P''(\widehat{\vec{s}}, \vec{t})) \rightarrow^*_{Prog'} \emptyset$, thus $HypIn(P''(\widehat{\vec{s}}, \vec{t})\sigma) \rightarrow^*_{Prog'} \emptyset$, we have $A' \rightarrow^*_{Prog'} \emptyset$ and by extension

$$ConsOut(P''(\widehat{\vec{s}}, \vec{t})\sigma) = ConsOut(P''(\widehat{\vec{s'}}, \vec{t'})\sigma) \rightarrow^*_{Prog'} \emptyset$$

As s and t are ground we conclude $ConsOut(P''(\widehat{\vec{s}}, \vec{t})) \rightarrow^*_{Prog'} \emptyset$. From the derivation step $P''(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow_{Prog''} \emptyset$ and definition 11 we deduce $P(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow_{Prog} \emptyset$, so the lemma is proved.

• Otherwise, the derivation contains several steps. Let us detail the first step:

$$P''(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow_{Prog''} P''_1(\widehat{\vec{s^1}}, \vec{t^1}), \ldots, P''_k(\widehat{\vec{s^k}}, \vec{t^k}) \ \rightsquigarrow^*_{Prog''} \emptyset$$

and let $H \leftarrow A_1, \ldots, A_k$ be the clause and σ the substitution used during this step. Since there is no cycle in the clause body, we can suppose without loss of generality that $\forall i, j\, (i \leq j \implies A_i \not> A_j)$. The clause is of the form $P''(\widehat{\vec{s'}}, \vec{t'}) \leftarrow P''_1(\widehat{\vec{s'^1}}, \vec{t'^1}), \ldots, P''_k(\widehat{\vec{s'^k}}, \vec{t'^k})$ with $P''(\widehat{\vec{s}}, \vec{t})\sigma = H\sigma = P''(\widehat{\vec{s'}}, \vec{t'})\sigma$ and $\forall i \in \{1, \ldots, k\}\, P''_i(\widehat{\vec{s^i}}, \vec{t^i}) = A_i\sigma = P''_i(\widehat{\vec{s'^i}}, \vec{t'^i})\sigma$.

The proof consists in : for $i = 1$ to k (in this order),

1. we prove that the input hypothesis of $P''_i(\widehat{\vec{s^i}}, \vec{t^i})$ is satisfied, i.e.
$$HypIn(P''_i(\widehat{\vec{s^i}}, \vec{t^i})) \rightarrow^*_{Prog'} \emptyset$$
2. from $P''_1(\widehat{\vec{s^1}}, \vec{t^1}), \ldots, P''_k(\widehat{\vec{s^k}}, \vec{t^k}) \rightsquigarrow^*_{Prog''} \emptyset$, we can obviously extract a subderivation $P''_i(\widehat{\vec{s^i}}, \vec{t^i}) \rightsquigarrow^*_{Prog''} \emptyset$ whose length is strictly less than n. We can then apply the induction hypothesis on this subderivation.

- For $i = 1$
  1. The clause used for derivation is in C'', due to definition 13
  $$HypIn(A_1) \rightarrow^*_{Prog'} B_1 \;\wedge$$
  $$\forall A' \in B_1\, (A' \in HypIn(H) \vee \exists j \in \{1, \ldots, k\}\, A' \in ConsOut(A_j))$$
  Actually, $\exists j \in \{1, \ldots, k\}\, A' \in ConsOut(A_j)$ is impossible, because the argument of A' is a variable (one output argument of A_j), so this variable must be in $HypIn(A_1)$ because $Prog'$ is in simple form (no extra variables). We deduce $A_1 > A_j$ which is not possible (no cycle). So $A' \in HypIn(H)$ then $A'\sigma \in HypIn(H\sigma)$. By hypothesis of the lemma we have $HypIn(P''(\widehat{\vec{s}}, \vec{t})) \rightarrow^*_{Prog'} \emptyset$, then $HypIn(H\sigma) = HypIn(P''(\widehat{\vec{s'}}, \vec{t'})\sigma) = HypIn(P''(\widehat{\vec{s}}, \vec{t})\sigma) \rightarrow^*_{Prog'} \emptyset$, therefore we deduce $A'\sigma \rightarrow^*_{Prog'} \emptyset$. This property holds for all $A' \in B_1$, then $HypIn(A_1\sigma) \rightarrow^*_{Prog'} \emptyset$, and finally
  $$HypIn(P''_1(\widehat{\vec{s^1}}, \vec{t^1})) \rightarrow^*_{Prog'} \emptyset$$
  2. Since $HypIn(P''_1(\widehat{\vec{s^1}}, \vec{t^1})) \rightarrow^*_{Prog'} \emptyset$ and $P''_1(\widehat{\vec{s^1}}, \vec{t^1}) \rightsquigarrow^*_{Prog''} \emptyset$ whose length is strictly less than n, we can then apply the induction hypothesis on this subderivation and conclude
  $$P_1(\widehat{\vec{s^1}}, \vec{t^1}) \rightsquigarrow^*_{Prog} \emptyset \;\wedge\; ConsOut(P''_1(\widehat{\vec{s^1}}, \vec{t^1})) \rightarrow^*_{Prog'} \emptyset$$

- For $i = 2$
  1. The clause used for derivation is in C'', due to definition 13
  $$HypIn(A_2) \rightarrow^*_{Prog'} B_2 \;\wedge$$
  $$\forall A' \in B_2\, (A' \in HypIn(H) \vee \exists j \in \{1, \ldots, k\}\, A' \in ConsOut(A_j))$$
  If $A' \in HypIn(H)$, we do the same proof as the case $i = 1$. Otherwise $A' \in ConsOut(A_j)$ for some $j \in \{1, \ldots, k\}$, here the unique possibility is $j = 1$ (no cycle), and since the conclusion of the previous case $i = 1$ is $ConsOut(P''_1(\widehat{\vec{s^1}}, \vec{t^1})) \rightarrow^*_{Prog'} \emptyset$. Consequently

$$ConsOut(A_1\sigma) = ConsOut(P''_1(\widehat{\vec{s'^1}}, \vec{t'^1})\sigma) \rightarrow^*_{Prog'} \emptyset$$

We deduce that $A'\sigma \rightarrow^*_{Prog'} \emptyset$, and finally $HypIn(P''_2(\widehat{\vec{s^2}}, \vec{t^2})) \rightarrow^*_{Prog'} \emptyset$.

2. This part is similar to the case $i = 1$ by replacing 1 by 2.

- We do the same proof for $i = 3$ until $i = k$.

Finally with the first step of the derivation and Definition 11 we prove that $P(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow_{Prog} P_1(\widehat{\vec{s^1}}, \vec{t^1}), \ldots, P_k(\widehat{\vec{s^k}}, \vec{t^k})$. With the conclusion of the previous steps of the proof, we have $\forall i \in \{1, \ldots, k\}\ P_i(\widehat{\vec{s^i}}, \vec{t^i}) \rightsquigarrow^*_{Prog} \emptyset$ then

$$P(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow^*_{Prog} \emptyset$$

On the other hand, $ConsOut(P''(\widehat{\vec{s}}, \vec{t})) = ConsOut(H\sigma)$ ($\vec{s}$ and $\vec{t}$ are ground), and $\forall A' \in ConsOut(H\sigma)$, $A' \in HypIn(H\sigma)$ or $A' \in ConsOut(A_j\sigma)$ for some $j \in \{1, \ldots, k\}$. In the two cases we have $A' \rightarrow^*_{Prog'} \emptyset$ (due to the hypothesis of the lemma or the conclusions of the previous steps of the proof). Then

$$ConsOut(P''(\widehat{\vec{s}}, \vec{t})) \rightarrow^*_{Prog'} \emptyset$$

B Completeness

We have to prove:

$$L^{\Sigma}_{Prog''}(P'', \vec{t}) \supseteq$$
$$(\ldots (\, L^{\Sigma}_{Prog}(P, \vec{t}) \cap_{i_1} L^{\Sigma}_{Prog'}(P'_1)\,) \cap_{i_2} \ldots) \cap_{i_m} L^{\Sigma}_{Prog'}(P'_m)$$

Let $\vec{s} = (s_1, \ldots, s_{ArOut(P)}) \in (T_\Sigma)^{ArOut(P)}$. From Theorem 9, we have to prove that

$$P(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow^*_{Prog} \emptyset \;\wedge\; P'_1(\widehat{s_{i_1}}) \rightarrow^*_{Prog'} \emptyset \;\wedge\; \ldots \;\wedge\; P'_m(\widehat{s_{i_m}}) \rightarrow^*_{Prog'} \emptyset$$
$$\wedge\; HypIn(P''(\widehat{\vec{s}}, \vec{t})) \rightarrow^*_{Prog'} \emptyset$$

implies

$$P''(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow^*_{Prog''} \emptyset$$

Let us write $\vec{t} = (t_1, \ldots, t_{ArIn(P)})$.
Since $HypIn(P''(\widehat{\vec{s}}, \vec{t})) = Q'_1(t_1), \ldots, Q'_{ArIn(P)}(t_{ArIn(P)})$, we have

$$HypIn(P''(\widehat{\vec{s}}, \vec{t})) \rightarrow^*_{Prog'} \emptyset \iff$$
$$Q'_1(t_1) \rightarrow^*_{Prog'} \emptyset \;\wedge\; \ldots \;\wedge\; Q'_{ArIn(P)}(t_{ArIn(P)}) \rightarrow^*_{Prog'} \emptyset$$

The proof is done by induction on the length n of the rewrite derivation

$$P(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow^*_{Prog} \emptyset$$

• If it contains exactly one step, the clause used in this step is a fact. We have $P(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow_{Prog} \emptyset$ and there exist in C the fact $P(\widehat{\vec{s'}}, \vec{t'}) \leftarrow$ and a substitution σ such that $P(\widehat{\vec{s'}}, \vec{t'})\sigma = P(\widehat{\vec{s}}, \vec{t})\sigma = P(\widehat{\vec{s}}, \vec{t})$ because $\vec{s}$ and $\vec{t}$ are ground.

By hypothesis we know that $P'_1(\widehat{s_{i_1}}) \rightarrow^*_{Prog'} \emptyset \wedge \ldots \wedge P'_m(\widehat{s_{i_m}}) \rightarrow^*_{Prog'} \emptyset$ and $Q'_1(t_1) \rightarrow^*_{Prog'} \emptyset \wedge \ldots \wedge Q'_{ArIn(P)}(t_{ArIn(P)}) \rightarrow^*_{Prog'} \emptyset$. Due to definition 11, for each $R'_1, \ldots R'_{ArIn(P)} \in Pred'$, the fact $H = P^{\{P'_1:i_1,\ldots P'_m:i_m\}}_{R'_1,\ldots R'_{ArIn(P)}}(\widehat{\vec{s'}}, \vec{t'}) \leftarrow$ is in PC''. Now, we have to choose predicates $R'_1, \ldots R'_{ArIn(P)}$ so that this fact is in C'' ? We need to prove that $ConsOut(H) \rightarrow^*_{Prog'} B_0$ and $\forall A' \in B_0, A' \in HypIn(H)$. Recall that $ConsOut(H) = P'_1(\widehat{s'_{i_1}}), \ldots P'_m(\widehat{s'_{i_m}})$. We know that

$$\forall j \in \{1, \ldots m\}\ P'_j(\widehat{s'_{i_j}})\sigma = P'_j(\widehat{s_{i_j}})\sigma \ \wedge \ P'_j(\widehat{s_{i_j}})\sigma \rightarrow^*_{Prog'} \emptyset$$

Then $P'_j(\widehat{s'_{i_j}})\sigma \rightarrow^*_{Prog'} \emptyset$. Due to the fact that $Prog'$ is in simple form, we deduce $P'_j(\widehat{s'_{i_j}}) \rightarrow^*_{Prog'} S'_1(\widehat{x_1}), \ldots, S'_k(\widehat{x_k})$ where $S'_1, \ldots S'_k$ are some predicates of $Pred'$ and $x_1, \ldots x_k$ some variables of s'_{i_j}. Since $P(\widehat{\vec{s'}}, \vec{t'}) \leftarrow$ is in C and $Prog$ is completely defined (definition 5), we deduce that all variables of $\vec{s'}$ are in $\vec{t'}$ (recall that all components of $\vec{t'}$ are variables). Then, all variables $x_1, \ldots x_k$ are components of $\vec{t'}$. Suppose $x_1 = t_{j_1}, \ldots x_k = t_{j_k}$, so by replacing all R'_{j_i} by S'_i in the predicate $P^{\{P'_1:i_1,\ldots P'_m:i_m\}}_{R'_1,\ldots R'_{ArIn(P)}}$, we obtain a new fact $H' = P^{\{P'_1:i_1,\ldots P'_m:i_m\}}_{V'_1,\ldots V'_{ArIn(P)}}(\widehat{\vec{s'}}, \vec{t'}) \leftarrow$ which is obviously in PC'' and by construction $ConsOut(H') \rightarrow^*_{Prog'} B_0$ and $\forall A' \in B_0, A' \in HypIn(H')$. We conclude that this new fact is in C'' and by using it we have

$$P''(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow_{Prog''} \emptyset$$

- Otherwise, the derivation contains several steps. Let us detail the first step :

$$P(\widehat{\vec{s}}, \vec{t}) \rightsquigarrow_{Prog} P_1(\widehat{\vec{s^1}}, \vec{t^1}), \ldots, P_k(\widehat{\vec{s^k}}, \vec{t^k}) \rightsquigarrow^*_{Prog} \emptyset$$

So there exist in C a clause $P(\widehat{\vec{s'}}, \vec{t'}) \leftarrow P_1(\widehat{\vec{s'^1}}, \vec{t'^1}), \ldots, P_k(\widehat{\vec{s'^k}}, \vec{t'^k})$ (also noted $H \leftarrow A_1, \ldots, A_k$) and a substitution σ such that $P(\widehat{\vec{s}}, \vec{t})\sigma = P(\widehat{\vec{s'}}, \vec{t'})\sigma$ and $\forall i \in \{1, \ldots, k\}$, $P_i(\widehat{\vec{s'^i}}, \vec{t'^i})\sigma = P_i(\widehat{\vec{s^i}}, \vec{t^i})$. Since there is no cycle in the clause body, we can suppose without loss of generality that $\forall i, j\, (i \leq j \implies A_i \not> A_j)$. By hypothesis of the lemma, we also assume that there exist predicates in $Pred'$, $P'_1, \ldots, P'_m, Q'_1, \ldots, Q'_{ArIn(P)}$ and positions $i_1, \ldots, i_m$ such that

$$\begin{array}{ll} & P'_1(\widehat{s_{i_1}}) \rightarrow^*_{Prog'} \emptyset \wedge \ldots \wedge P'_m(\widehat{s_{i_m}}) \rightarrow^*_{Prog'} \emptyset \\ \text{and} & Q'_1(\widehat{t_1}) \rightarrow^*_{Prog'} \emptyset \wedge \ldots \wedge Q'_{ArIn(P)}(\widehat{t_{ArIn(P)}}) \rightarrow^*_{Prog'} \emptyset \end{array}$$

We deduce that $\forall l \in \{1, \ldots, m\}$, $P'_l(\widehat{s'_{i_l}})\sigma \rightarrow^*_{Prog'} \emptyset$. Since $Prog'$ is in simple form, we obtain : $P'_l(\widehat{s'_{i_l}}) \rightarrow^*_{Prog'} R'_{l_1}(\widehat{x_{i_{l_1}}}), \ldots, R'_{l_n}(\widehat{x_{i_{l_n}}})$ and $R'_{l_1}(\widehat{x_{i_{l_1}}})\sigma \rightarrow^*_{Prog'} \emptyset, \ldots, R'_{l_n}(\widehat{x_{i_{l_n}}})\sigma \rightarrow^*_{Prog'} \emptyset$. The variables $x_{i_{l_?}}$ are in the component s'_{i_l} of $\vec{s'}$. $Prog$ is completely defined (definition 5), then for all variables we have $x_{i_{l_?}} \in Var(\vec{t'}) \cup Var(\vec{s'^1}) \cup \ldots \cup Var(\vec{s'^k})$. Due to definition 11 we have in PC'', all clauses

$$P^{\{P'_1:i_1\dots P'_m:i_m\}}_{Q'_1\dots Q'_{ArIn(P)}}(\widehat{\vec{s'}},\vec{t'}) \leftarrow P_1{}^{\{P'^1_1:i^1_1\dots P'^1_{m_1}:i^1_{m_1}\}}_{Q'^1_1\dots Q'^1_{ArIn(P_1)}}(\widehat{\vec{s'^1}},\vec{t'^1})\dots P_k{}^{\{P'^k_1:i^k_1\dots P'^k_{m_k}:i^k_{m_k}\}}_{Q'^k_1\dots Q'^k_{ArIn(P_k)}}(\widehat{\vec{s'^k}},\vec{t'^k})$$

(also noted $H'' \leftarrow A''_1,\dots,A''_k$) where $Q'*$ and $P'*$ in the body are arbitrary predicates of $Pred'$ and i^j_* arbitrary component positions in the tuple $\vec{s'^j}$. Now, we have to choose all $Q'*$, $P'*$, i^j_* so that this clause is in C''. First, we remove the $ConsOut$ part of all A''_i, and we keep the arbitrary predicates for $HypIn(A''_i)$. We know

$$\forall l \in \{1,\dots,m\}\ P'_l(\widehat{s'_{i_l}}) \rightarrow^*_{Prog'} R'_{l_1}(\widehat{x_{i_{l_1}}}),\dots,R'_{l_n}(\widehat{x_{i_{l_n}}})$$

For each variable, $x_{i_{l_j}} \in HypIn(H)$ or $\exists A_i,\ x_{i_{l_j}} \in ConsOut(A_i)$.
If $x_{i_{l_j}}$ is in $HypIn(H)$, suppose it's the p^{th} component of $\vec{t'}$. We replace Q'_p by R'_{l_j} in the head of the clause $H'' \leftarrow A''_1,\dots,A''_k$.
If $x_{i_{l_j}}$ is in $ConsOut(A_i)$ (only one A_i is possible), suppose it's the p^{th} component of $\vec{s'^i}$. We add $R'_{l_j}:p$ in the $ConsOut$ part of A''_i. Thus $ConsOut(H'') \rightarrow^*_{Prog'} B_0$ and $\forall A' \in B_0\,(A' \in HypIn(H'') \vee \exists j \in \{1,\dots,k\}\ A' \in ConsOut(A''_j))$. Now we scan all A''_i.

- For A''_1. $HypIn(A''_1) = Q'^1_1(\widehat{\vec{t'^1_1}}),\dots Q'^1_{ArIn(P_1)}(\widehat{\vec{t'^1_{ArIn(P_1)}}})$. For all t'^1_j we suppose that predicate Q'^1_j in $Pred'$ is chosen such that

$$Q'^1_j(\widehat{t'^1_j}) \rightarrow^*_{Prog'} R'^1_1(\widehat{x^1_{j_1}}),\dots,R'^1_n(\widehat{x^1_{j_n}})$$

  (it's always possible because $Prog'$ is complete). If $n=0$ we do nothing, else $x^1_{j_l}$ is in $HypIn(H'')$ since $\forall j,\ A_1 \not> A_j$. Suppose it's the i^{th} component of $\vec{t'}$, we replace[6] Q'_i by R'^1_l in H''. So we obtain $HypIn(A''_1) \rightarrow^*_{Prog'} B_1$ and $\forall A' \in B_1,\ A' \in HypIn(H'')$.
- For A''_2, we can do the same reasoning as in the previous case but we have two possibilities for variables $x^2_{j_l}$. They are either in $HypIn(H'')$ or in $ConsOut(A''_1)$. For all l, if the variable $x^2_{j_l}$ is in $HypIn(H'')$ we do the same thing as previously. Else $x^2_{j_l}$ is in $ConsOut(A''_1)$, suppose it's the i^{th} component of $\vec{s'^1}$. We add $R'^2_l:i$ in the $ConsOut$ part of A''_1. So we obtain $HypIn(A''_2) \rightarrow^*_{Prog'} B_2$ and $\forall A' \in B_2,\ A' \in HypIn(H'') \cup ConsOut(A''_1)$.
- We do the same proof for $A''_3,\dots,A''_k$.

By construction we obtain a new clause $H'' \leftarrow A''_1,\dots A''_k$ which is obviously in PC'' and also in C''. By using this clause we obtain :

$$P''(\widehat{\vec{s}},\vec{t}) \rightsquigarrow_{Prog''} P''_1(\widehat{\vec{s^1}},\vec{t^1}),\dots,P''_k(\widehat{\vec{s^k}},\vec{t^k})$$

For each $i \in \{1,\dots,k\}$, the length of the derivation $P_i(\widehat{\vec{s^i}},\vec{t^i}) \rightsquigarrow^*_{Prog} \emptyset$ is strictly less than n, and the other hypotheses required to apply the lemma are satisfied. By induction hypothesis $P''_i(\widehat{\vec{s^i}},\vec{t^i}) \rightsquigarrow^*_{Prog''} \emptyset$. Consequently $P''(\widehat{\vec{s}},\vec{t}) \rightsquigarrow^*_{Prog''} \emptyset$.

[6] The Q'_i that is replaced, doesn't provide from a previous replacing because $Input(H).Output(B)$ is a linear tuple of variables due to Definition 3.