A Unified Framework to Compute over Tree Synchronized Grammars and Primal Grammars
Frédéric Saubion, Igor Stéphan

To cite this version:

HAL Id: hal-00958984
https://hal.inria.fr/hal-00958984
Submitted on 13 Mar 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
A Unified Framework to Compute over Tree Synchronized Grammars and Primal Grammars

Frédéric Saubion and Igor Stéphan

LERIA, Université d’Angers, 2, Bd Lavoisier
49045 Angers Cedex 01, France

received Mar 27, 2001, accepted Nov 17, 2002.

Tree languages are powerful tools for the representation and schematization of infinite sets of terms for various purposes (unification theory, verification and specification ...). In order to extend the regular tree language framework, more complex formalisms have been developed. In this paper, we focus on Tree Synchronized Grammars and Primal Grammars which introduce specific control structures to represent non regular sets of terms. We propose a common unified framework in order to achieve the membership test for these particular languages. Thanks to a proof system, we provide a full operational framework, that allows us to transform tree grammars into Prolog programs (as it already exists for word grammars with DCG) whose goal is to recognize terms of the corresponding language.

Keywords: Tree Grammars, Proof Systems, Prolog Implementation

1 Introduction

Tree languages [3, 6] have been extensively studied and have many applications in various areas such as term rewriting, term schematization, specification and verification ... Languages can be handled either from the generation point of view (i.e. grammars) or from the recognition point of view (i.e. automata). Regular tree languages [3, 6] are very close to regular word grammar with tree (ie terms)† as basic objects. They have nice closure properties (concerning complementation, union and intersection) and the membership test and the emptiness test are decidable. Moreover they can be defined using a notion of finite automata which can be easily used to test the membership of a term to a given language. Therefore, from a practical point of view, tree automata appear to be easily implementable tools for recognition. But, these notions are not sufficient to describe more complex sets of terms (non regular tree languages).

In this paper, we focus on two particular non regular classes of tree languages defined by their associated notion of tree grammars: Tree Synchronized Grammars and Primal Grammars. The purpose of this paper is to define a uniform and operational framework to describe and implement a membership test procedure for these languages.

† In the following and without any loss of generality, we consider always terms as the basic elements of tree languages

1365–8050 © 2002 Discrete Mathematics and Theoretical Computer Science (DMTCS), Nancy, France
Tree Synchronized Grammars (TSG) \cite{13} and Primal Grammars (PG) \cite{11} are issued from the E-unification framework. E-unification \cite{17} is known to be undecidable in general, but, as explained in \cite{14}, some decidable classes can be characterized by using such tree languages. Due to their specific definitions introducing control in the derivation process, a notion of automata can not be easily derived for these two classes of tree languages. Their expressive power can be highlighted by the two following simple examples.

**Example 1 (TSG)** We want to define the language corresponding to \{ \( f(s^n(z), s^{2n}(z)) \mid n \in \mathbb{N} \) \} which is obviously not a regular language.

The language description problem lies in the fact that, informally, each time a function symbol \( s \) is produced in the first argument of \( f \) one has to produce two in the second argument. In fact, the growth of the first argument has to be synchronized with the second. Tree Synchronized Grammars take this control into account by allowing packs of synchronized production rules instead of usual single grammar productions rules.

**Example 2 (PG)** In this second example, we want to describe the language of all the integer lists \([n, \ldots, 0]\).

Integer are represented with \( z \) (for the zero integer) and successor function \( s \) and lists are built thanks to the binary constructor \( \ast \) and the empty list \( \bot \) (i.e. the list \([2, 1, 0]\) is obtained as \( \ast(s(s(z))), \ast(s(z), \ast(z, \bot)) \)).

This language is also not regular.

The main problem here is the relation between the different arguments of the list you are constructing. Once you have generated an element \( s^n(z) \), next element of the list must be \( s^{n+1}(z) \). It appears that you have to count the depth of successive argument. By introducing counters variables in the production rules, Primal Grammars are able to handle this language.

One has to remark that the language of the first example cannot be generated using PG while the language of the second example cannot be described by a TSG. Due to these two specific control mechanisms (synchronization and counter variables), the standard notion of automata does not appear clearly.

Inspired by the implementation of Definite Clause Grammars \cite{2} that allows to write word grammars as Horn clauses in a Prolog environment, our aim is to propose a similar approach for these particular tree languages. The key idea is to define a proof framework to describe the behavior of tree grammars and therefore to provide a proof theoretical semantics for grammar derivations. In this context, we change from syntactic generation of trees to logical deduction (i.e. productions rules are translated into logical formulas and grammar derivations into logical inferences). Then, the membership test, for a language described by a grammar, just consists in proving a particular formula in a sequent calculus using a proof system. This approach provides a uniform framework for the definition and use of both type of grammars. The method can be briefly described as follows:

Production rules are translated into logical formulas built over linear logic syntax \cite{8} to handle the problem of synchronization (this idea was already investigated in \cite{10}). Then, a sequent calculus proof system, inspired by the proof system of D. Miller \cite{16}, allows us to translate derivations into proof searches. The same job can be adapted for Primal Grammars.

The equivalence between the notion of grammar computation and our notion of proof is established by a correctness and completeness result. The last part of our work consists in refining this initial proof system.
Computing over TSG and PG

in order to introduce a notion of strategy in the proof search and thus to get a goal directed procedure. At
this step, both transformation function and logical inference system can be easily translated into Prolog
Horn clauses. Of course, it should be noticed that our system also works for Regular Tree Languages,
since they are included in TSG. Note that an alternative algorithm has been proposed for the membership
test of a TTSG in [9].

This paper is an extension of the previous work [8] and is organized as follows. Section 2 presents basic
definitions of Tree Synchronized Grammars, Primal Grammars and proof systems. Section 3 described
how the different formalism can be translated into linear logic formulas. The proof system is defined and
refined in Section 4. Section 5 is the implementation issue of our work and we conclude in Section 6.

2 Preliminaries

In this paper, we recall here some notions about tree grammars and linear logic proof systems. We refer
the reader to [3, 6, 7, 8, 13, 16] for more details.

We first introduce the two specific grammar formalisms we deal with in this paper. The original for-
malisms have been adapted to unify and simplify the presentation.

2.1 Tree Synchronized Grammars

Let $L$ be a finite set of symbols (a signature), $T(L)$ denotes the first-order algebra of ground terms over
$L$. $L$ is partitioned in two parts: the set $F$ of terminal symbols, and the set $N$ of non terminal symbols.
Upper-case letters denote elements of $N$. $t|u$ denotes the subterm of $t$ at occurrence $u$ and $t[u ← v]$
denotes the term obtained by replacing in $t$ the subterm $t[u]$ by $v$. $t[v]$ denotes a term containing a subterm
$v$ at a particular occurrence and $O(t)$ denotes the set of occurrences in $t$. $C(X_1, ..., X_n)$ denotes a term with
occurrences: $\{1, ..., n\}$ such that $C(X_1, ..., X_n)|i = X_i$. $C$ will be called a context. We first define the notion
of productions for TSG. We require that $F ∩ N = \emptyset$ and that each element of $F ∪ N$ has a fixed arity. We
refer the reader to [3] for more details on these basic notions.

Definition 1 (Productions) A production is a rule of the form $X ⇒ t$ where $X ∈ N$ and $t ∈ T(L)$. A pack
of productions is a set of productions denoted $\{X_1 ⇒ t_1, ..., X_n ⇒ t_n\}$.

- When the pack is a singleton of the form $\{X_1 ⇒ C(Y_1, ..., Y_n)\}$ where $C$ is a context of terminal
  symbols and $Y_1, ..., Y_n$ non terminals. The production is said free, and is written more simply
  $X_1 ⇒ C(Y_1, ..., Y_n)$.
- When the pack is of the form $\{X_1 ⇒ Y_1, ..., X_n ⇒ Y_n\}$ where $Y_1, ..., Y_n$ are terminals or non ter-
ninals. The productions of the pack are said synchronized.

We can then define the notion of TSG.

Definition 2 (TSG) A TSG is defined by a 4-tuple $(F, N, PP, TI)$ where

- $F$ is the set of terminals,
- $N$ is the finite set of non terminals,
- $PP$ is a finite set of packs of productions,
• \(TI\) is the axiom of the TSG denoted \((I,\#)\) where \(I\) is a non terminal and \(\#\) a new symbol added to the signature.

Note that, the axiom is a pair due to the control of synchronizations. Basically, counters are associated to non terminal symbols to ensure the integrity of the application of synchronized productions.

Back to example 1, we wanted to define the language corresponding to \(\{f(s^n(z),s^{2n}(z)) \mid n \in \mathbb{N}\}\). The corresponding TSG consists of the following production rules:

\[
R_0 : I \Rightarrow f(X,Y) \\
R_1 : X \Rightarrow z \\
R_2 : Y \Rightarrow z \\
P_1 : \{R_3 : X \Rightarrow s(X), R_4 : Y \Rightarrow s(s(Y))\}
\]

where \(X, Y\) are non terminal symbols and \(f, s, z\) terminal symbols. \(P_1 : \{R_3, R_4\}\) denotes that, using the pack of production \(P_1, R_3\) and \(R_4\) have to be applied at the same time (synchronization).

**Definition 3 (Computations of a TSG)** The set of computations of a TSG \(Gr = (\mathcal{F}, \mathcal{N}, PP, TI)\), denoted \(\text{Comp}(Gr)\), is the smallest set defined by:

- \(TI\) is in \(\text{Comp}(Gr)\),

- if \(t\) is in \(\text{Comp}(Gr)\), \(t|_a = (X,c)\) and the free production \(X \Rightarrow C(Y_1, \ldots, Y_n)\) is in \(PP\) then \(t[u \leftarrow C((Y_1, c), \ldots, (Y_n, c))]\) is in \(\text{Comp}(Gr)\),

- if \(t\) is in \(\text{Comp}(Gr)\) and there exists \(n\) pairwise different occurrences \(u_1, \ldots, u_n\) of \(t\) such that \(\forall i \in [1,n] \ t|_{u_i} = (X_i, c_i)\) and \(c_i = a\) and the pack of productions \(\{X_1 \Rightarrow Y_1, \ldots, X_n \Rightarrow Y_n\}\) is in \(PP\), then \(t[u_1 \leftarrow (Y_1, b)] \ldots [u_n \leftarrow (Y_n, b)]\) (where \(b\) is a new symbol) is in \(\text{Comp}(Gr)\).

The symbol \(\Rightarrow\) denoting also the above two deduction steps, a derivation of \(Gr\) is a sequence of computations \(TI \Rightarrow t_1 \Rightarrow \ldots \Rightarrow t_n\).

Always following Example II, one possible derivation of this grammar is thus (where \(a\) is a new symbol):

\[
(I,\#) \Rightarrow_{R_0} f((X,\#),(Y,\#)) \Rightarrow_{P_1} f(s((X,a)),s(s((Y,a)))) \\
\Rightarrow_{R_1} f(s(z),s(s((Y,a)))) \Rightarrow_{R_2} f(s(z),s(s(z)))
\]

As mentioned in the introduction, counters are introduced to control the application of the synchronized production rules. The previous definition imposes that only non terminals having the same control symbol can be used in a synchronized production. It should be noticed that TSG were originally defined using tuple of counters. Here, as we already did, concerning tuples of terms in the definition of the axiom, we consider a single counter to control synchronization in order to simplify the presentation of the basic concepts. The second step of this derivation clearly illustrates the notion of synchronized production rules.

**Definition 4 (Recognized Language)** The language recognized by a TSG \(Gr\), denoted \(\text{Rec}(Gr)\), is the set of trees composed of terminal symbols \(\text{Comp}(Gr) \cap \mathcal{T} (\mathcal{F})\).
2.2 Primal Grammars

In this section, we recall definitions related to primal grammars which have been introduced by M. Hermann and R. Galbavý in [11]. To avoid too long developments, some notations have been modified to fit ours. More details can be found in [11].

The control mechanism introduced by Hermann and Galbavý is based on the notion of counter variables which are variables having integer values.

**Definition 5** A counter expression is an expression built over 0, successor function $s$ and a set of counter variables $\text{Cnt}$. $s(c)$ means $1 + c$ if $c$ is a counter expression. The ground counter expression $s^n(0)$ is interpreted as the integer $n$.

Now, we recall the notion of primal term.

**Definition 6 (Primal Algebra)** The algebra of primal terms is defined over a set of functions $\mathcal{D}$ (whose elements $\tilde{f}$ are called defined functions and have a counter arity $\text{car}(\tilde{f})$ and a standard arity $\text{ar}(\tilde{f})$), a set of constructor symbols $\mathcal{F}$, a set of ordinary variables $X$ and a set of counter variables $\text{Cnt}$. This is the smallest set such that

- each ordinary variable of $X$ is a primal term,
- if $c_1, \ldots, c_k$ are counter expressions, $t_1, \ldots, t_n$ are primal terms and $\tilde{f} \in \mathcal{D}$ such that $\text{car}(\tilde{f}) = k$ and $\text{ar}(\tilde{f}) = k + n$, then $\tilde{f}(c_1, \ldots, c_k; t_1, \ldots, t_n)$ is a primal term,
- if $t_1, \ldots, t_n$ are primal terms and $c$ is a constructor with the arity $\text{ar}(c) = n$ the $c(t_1, \ldots, t_n)$ is a primal term.

The schematization is achieved by introducing a particular type of rewrite system.

**Definition 7 (Presburger rewrite system)** A Presburger rewrite system contains for each defined function $\tilde{f}$ the following pair of Presburger rewrite rules:

- basic rule
  $$\tilde{f}(0, \overline{c}; \overline{x}) \rightarrow_{\text{Pbg}} t_1$$
- and the inductive rule which have one of the following forms:
  $$\tilde{f}(n+1, \overline{c}; \overline{x}) \rightarrow_{\text{Pbg}} t_2[u \leftarrow \tilde{f}(n, \overline{c}; \overline{x})]$$
  $$\tilde{f}(n+1, \overline{c}; \overline{x}) \rightarrow_{\text{Pbg}} t_2[u \leftarrow \tilde{f}(n, c_1, \ldots, c_{i-1}, c_i + 1, c_{i+1}, \ldots, c_n; \overline{x})]$$

where $\overline{c}$ is a vector of counter variables, $\overline{x}$ a vector of ordinary variables and $u \in O(t_2)$.

To simplify the presentation, we omit here some additional conditions that are needed to ensure decidability properties of PG. We refer the reader to [11] for more details.

From the previous definitions, we can now define a primal grammar and its associated language.
We recall basic notations for sequent calculus. atoms and the logic connectives. (T-algebra built over A formula for the precise definitions of these connectives). symbols with fixed arity and \( \Sigma \) predicates symbols and terms. A substitution is a mapping from \( V \) ground integer value to each counter variable of \( t \). Let us consider a first order logic signature \( \Sigma \) with \( \Sigma \) finite set of function symbols with fixed arity and \( \Sigma \) finite set of predicate symbols. \( T(\Sigma, V) \) denotes the first order term algebra built over \( \Sigma \) and \( V \) and \( T(\Sigma, V) \) denotes the set of ground terms. Atoms are, as usual, built over predicates symbols and terms. A substitution is a mapping from \( V \) to \( T(\Sigma, V) \) which is extended to \( T(\Sigma, V) \). A substitution assigning \( t \) to a variable \( x \) will be denoted \( \{x \leftarrow t\} \). We introduce some linear logic\(^{\dagger}\) notations: \( \cdot \) denotes the linear implication and \( \otimes \) denotes the multiplicative conjunction (see \([8, 7]\) for the precise definitions of these connectives). A formula \( A \cdot B_1 \otimes \ldots \otimes B_n \) will be called a clause. The set \( \Sigma_{\text{formula}} \) is the set of \( \Sigma \)-formulas built using atoms and the logic connectives.

2.3 Linear Logic and Sequent Calculus

We recall here some basic notations and notions related to first order and linear logics \([8, 7]\), and proof systems in sequent calculus \([16]\).

Let us consider a first order logic signature \( \Sigma \) with \( V \) a countable set of variables, \( \Sigma \) finite set of function symbols with fixed arity and \( \Sigma \) finite set of predicate symbols. \( T(\Sigma, V) \) denotes the first order term algebra built over \( \Sigma \) and \( V \) and \( T(\Sigma, V) \) denotes the set of ground terms. Atoms are, as usual, built over predicates symbols and terms. A substitution is a mapping from \( V \) to \( T(\Sigma, V) \) which is extended to \( T(\Sigma, V) \). A substitution assigning \( t \) to a variable \( x \) will be denoted \( \{x \leftarrow t\} \). We introduce some linear logic\(^{\dagger}\) notations: \( \cdot \) denotes the linear implication and \( \otimes \) denotes the multiplicative conjunction (see \([8, 7]\) for the precise definitions of these connectives).

A formula \( A \cdot B_1 \otimes \ldots \otimes B_n \) will be called a clause. The set \( \Sigma_{\text{formula}} \) is the set of \( \Sigma \)-formulas built using atoms and the logic connectives.

We recall basic notations for sequent calculus.

\(^{\dagger}\) Intuitively, the key idea of linear logic we use here is that, when performing logical inferences, some hypothesis can be consumed. This means that, along a proof, some formulas are persistent (as in usual first order logic) but some formulas can only be used once.
**Definition 9 (Sequent)** A sequent will be written as $\Sigma : \Delta \rightarrow G$ where $\Sigma$ is the signature, $\Delta$ a multiset of $\Sigma$-formulas and $G$ a $\Sigma$-formula.

From these sequents we can build proof systems.

**Definition 10 (Proof System)** A proof system consists of a set of inference rules between sequents. An inference rule is presented here as:

\[
\Sigma : \Delta \rightarrow G
\]

\[
\Sigma' : \Delta' \rightarrow G'
\]

We use the classical notion of proof (i.e. there is a proof of a sequent in a proof system if this sequent is the root of a proof tree constructed using inference rules with empty leaves).

### 3 From TSG and PG Computation to Proof Search

In this section, we define the translation of grammar production rules into linear logic formulas. Then, designing a particular proof system, we show that usual grammar computations as defined in Definitions 3 and 8 can be reduced to proof searches using this system.

#### 3.1 Transformation of TSG into Linear Logic Formulas

Given a TSG $(F, N, PP, TI)$, the set $PP$ of production rules is partitioned into $PP_{\text{free}}$ the set of free production rules and $PP_{\text{sync}}$ the set of packs of synchronized production rules. We define now a transformation function $\Psi$ which is decomposed into two different mappings (corresponding respectively to the transformation of free and synchronized production rules). A predicate symbol and a variable will be associated to each non terminal.

**Definition 11 (Transformation Function $\Psi_{TSG}$)**

Let $\sigma_N : N \rightarrow \Sigma_N$ be the mapping that translates every non terminal symbol into a predicate symbol (to simplify, we will write $\sigma_N(N) = n$). Let $\sigma_V : N \rightarrow V$ be the mapping that transforms every non terminal symbol into a logical variable (we will write $\sigma_V(N) = N$). For the sake of readability, universal quantifications have been omitted when clear.

Let $\sigma_{PP_{\text{free}}} : PP_{\text{free}} \rightarrow \Sigma_{\text{formula}}$ be the mapping that translates every free production rule into a $\Sigma$-formula and $\sigma_{PP_{\text{sync}}} : PP_{\text{sync}} \rightarrow \Sigma_{\text{formula}}$ the mapping that translates every pack of synchronized production rules into a $\Sigma$-formula.

**Free productions**

Let $N \rightarrow g(N_1, \ldots, N_p)$ and $N \rightarrow t$ in $PP_{\text{free}}$.

\[
\sigma_{PP_{\text{free}}}(N \rightarrow g(N_1, \ldots, N_p)) = n(g(N_1, \ldots, N_p), c) \circ n_1(N_1, c) \otimes \cdots \otimes n_p(N_p, c)
\]

\[
\sigma_{PP_{\text{free}}}(N \rightarrow t) = n(t, c) \circ 1
\]

**Synchronized Productions**

Let $S = \{ R \mid P \in PP_{\text{sync}} \land R \in P \}$ and $\{ R_1; \ldots; R_n \} \in PP_{\text{sync}}$.
\[ \sigma_{PP_{\text{sync}}} \left( \{ R_1, \ldots, R_n \} \right) = \forall c \exists nc (\sigma_S(R_1, c, nc) \otimes \ldots \otimes \sigma_S(R_n, c, nc)) \]

where \( \sigma_S : S \times C \times C \rightarrow \Sigma_{\text{formula}} \) is defined\(^8\) for \( N \rightarrow g(N_1, \ldots, N_p) \) and \( N \rightarrow t \) in \( S \), as:

\[
\begin{align*}
\sigma_S(N \rightarrow g(N_1, \ldots, N_p), c, nc) &= n(g(N_1, \ldots, N_p), c) \circ - n_1(N_1, nc) \otimes \ldots \otimes n_p(N_p, nc) \\
\sigma_S(N \rightarrow t, c, nc) &= n(t, c) \circ - I
\end{align*}
\]

Let \( \Sigma_{\Psi} = \Sigma_{\mathcal{F}} \cup \Sigma_N \) and \( \Psi((\mathcal{F}, \mathcal{N}, PP, TI)) = \sigma_{PP_{\text{free}}} \cup \sigma_{PP_{\text{sync}}} \) (in the following, \( \Psi \) will denote a set of formulas obtained from a TSG and will be called a TSG program). At this step, a particular sequent calculus defines the semantics of \( \Psi \).

A similar transformation function can be defined for primal grammars.

### 3.2 Transformation of PG into Linear Logic Formulas

We suppose in the following that Presburger rewrite systems are such that each definite symbol not appearing in the left hand side of a rule must also appear linearly in the right hand side. This is not in fact a restriction since multiple occurrences of a same symbol can be replaced by a new symbol with the same definition. This transformation does not modify the primal grammar definition.

Given a primal grammar \( (\mathcal{F}, \mathcal{D}, \mathcal{R}, A) \), the function \( \sigma_D : \mathcal{D} \rightarrow \Sigma_D \) maps each defined symbol \( N \) to a predicate symbol (we will use the notation \( \sigma_D(N) = n \)). \( \sigma_V : \mathcal{D} \rightarrow \mathcal{V} \) is the function that maps each defined symbol \( N \) to a logical variable (\( \sigma_V(N) = N \)). We use the notation \( \vec{c} = (c^1, \ldots, c^{ar(N)-1}) \), \( \vec{c} = (c^1, \ldots, c^{i-1}, s(c^i), c^{i+1}, \ldots, c^{ar(N)-1}) \) and \( \forall i \in [1..p], \vec{c}_i \subseteq \vec{c} \).

\( \sigma_R : \mathcal{R} \rightarrow \Sigma_{\text{formula}} \) is the function that maps to each Presburger rewrite rule a formula:

---

\(^8\) The set \( C \) denotes a set of counters (i.e. new symbols which are not in the current signature).
In this section, we define a sequent calculus proof system inspired by the system Forum of D. Miller [16].

3.3 The Proof System FG

As previously defined for $TSG$, we get the notion of $PG$ program $\Psi$ from the above definition. After these syntactic transformations, there is no more need to distinguish the two different types of grammar definition. We propose a uniform proof system which allows us to handle both formalisms at once.

### 3.3 The Proof System FG

In this section, we define a sequent calculus proof system inspired by the system Forum of D. Miller [16]. Our proof system $FG$ (Forum for Grammars) is defined by the following inference rules and defines the basic proof theoretical semantics of a $TSG$ (resp. $PG$) program $\Psi$. In order to simplify the presentation, we have omitted $\Psi$ in the sequents since, of course, it does not change.

**Definition 12 (FG system)**

\[
\Sigma := \emptyset
\]

\[
[\text{Sync}]
\]

\[
\frac{\Sigma, \alpha : C_1 \theta, \ldots, C_r \theta : \Delta \rightarrow G}{\Sigma : \Delta \rightarrow G}
\]

where $\forall c \in \text{nc}(C_1 \otimes \ldots \otimes C_r) \in \Psi$, $\theta = \{ c \leftarrow \beta, nc \leftarrow \alpha \}$, $\beta \in \Sigma$, $\alpha \not\in \Sigma$. 
Frédéric Saubion and Igor Stéphan

\[ \Sigma : \Delta_1 \rightarrow A_1 \sigma \ldots \Sigma : \Delta_p \rightarrow A_p \sigma \]

where \( C = (H \circ - A_1 \otimes \ldots \otimes A_p) \in \Psi \) and \( H \sigma = G \).

\[ \Sigma : \Delta_1 \rightarrow A_1 \sigma \ldots \Sigma : \Delta_p \rightarrow A_p \sigma \]

where \( H \sigma = G \).

**Comments:** It should be noticed that we distinguish free production rules and synchronized production rules (see the definition of the transformation function \( \Psi \) in Section 3.1). Clearly free productions appear as clauses \( H \circ - A_1 \otimes \ldots \otimes A_n \) and packs of synchronized productions appear as formulas \( \forall C \in \operatorname{inc}(C_1 \otimes \ldots \otimes C_n) \) where the \( C_i \)'s are \( H \circ - A_1 \otimes \ldots \otimes A_n \) and are called linear clauses (in the following linear clause will always refer to a clause generated by synchronization rules). Clauses corresponding to free productions in \( \Psi \) are persistent along the proof and are used in the inference [Back!] (this corresponds to a step of the grammar derivation using this free production). The treatment of a pack of synchronization is performed thanks to the rules [Sync] and [Back?]. The first step consists in generating the formula corresponding to this pack in \( \Delta \) (rule [Sync]). The control is ensured by the instantiation of the counter variables with new symbols added to the signature \( \Sigma \) in rule [Sync]. Since \( \Delta \) is the linear logic part of our system, the linear clauses will be consumed when used by rule [Back?] (in the philosophy of linear logic). This ensures the integrity of the synchronization and the control of the simultaneous application of the different productions of this pack. [Back?] and [Back!] lead to branching in the search tree.

As a consequence, our system \( FG \) is a subset of Forum (i.e. our rules can be expressed in Forum).

**Proof:** The rules of FORUM used in this proof can be found in [16]. We introduce the program \( \Psi \) in the proof to keep the initial FORUM presentation.

**Proof of the correctness of \( 1 \) w.r.t. FORUM**

Let \( 1 \equiv \bot \circ - \bot \) in

\[ \Sigma : \Psi ; \bot \rightarrow \| L \]

\[ \Sigma ; \Psi ; \bot \rightarrow \| \text{decide}_1 \]

\[ \Sigma : \Psi ; \bot \rightarrow \| R \]

\[ \Sigma : \Psi ; \rightarrow 1 \]

Proof: The rules of FORUM used in this proof can be found in [16]. We introduce the program \( \Psi \) in the proof to keep the initial FORUM presentation.
Proof of the correctness of [Sync] w.r.t. FORUM
Let \( C \otimes C' \equiv (\bot \circ (\bot \circ C) \not\equiv (\bot \circ C')) \) and \([\otimes L]\) :
\[
\Sigma : \Psi; C, C', \Delta \rightarrow G \\
\Sigma : \Psi; C, C', \Delta \rightarrow \bot, G \quad \Sigma : \Psi; C, C' \rightarrow \bot \circ \bot, G \\
\Sigma : \Psi; C, \Delta \rightarrow \bot, \bot \circ C', G \\
\Sigma : \Psi; \Delta \rightarrow \bot \circ C, \bot \circ C', G \\
\Sigma : \Psi; \Delta \rightarrow (\bot \circ C) \not\equiv (\bot \circ C'), G
g \quad \Sigma : \Psi; \rightarrow G
\]
and \( \exists B \equiv (\forall x B^⊥)^⊥ \equiv \bot \circ (\forall x (\bot \circ B)) \) and \([\exists L]\) (\( y \not\in \Sigma \)):
\[
y, \Sigma : \Psi; \Delta, B \{ \Sigma \} \rightarrow G \\
y, \Sigma : \Psi; \Delta, B \{ \Delta \} \rightarrow \bot, G \\
y, \Sigma : \Psi; \Delta \rightarrow \bot \circ B \{ \Sigma \} , G \quad \Sigma : \Psi; \rightarrow G
\]
in
\[
\Sigma, \beta, \alpha : \Psi; C_1 \{ c \leftarrow \beta, nc \leftarrow \alpha \}, \ldots, C_n \{ c \leftarrow \beta, nc \leftarrow \alpha \}, \Delta \rightarrow G_{[\otimes L]}^{(\otimes L)^{-1}} \\
\Sigma, \beta, \alpha : \Psi; \Delta \rightarrow C_1 \{ c \leftarrow \beta, nc \leftarrow \alpha \}, G_{\forall \exists L} \quad \Sigma, \beta : \Psi; \Delta \rightarrow G_{\forall \exists L} \\
\Sigma, \beta : \Psi; \Delta \rightarrow C_1 \{ c \leftarrow \beta, nc \leftarrow \alpha \}, G_{\forall \exists L} \quad \Sigma, \beta : \Psi; \Delta \rightarrow G_{\forall \exists L}^{(\otimes L)^{-1}}
\]
with \( \alpha(\not\in \beta) \not\in \Sigma \).

Proof of the correctness of [Back !] and [Back ?] w.r.t. FORUM
Let \( t_1, \ldots, t_n \in T(\Sigma \not\equiv \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\} \) with \( H \rho = G \)
and \( \delta_\oplus = \Sigma : \Psi; \Delta_{p-1} \rightarrow A_{p-1} \rho \Sigma : \Psi; \Delta_p \rightarrow A_p \rho \)
\[
\vdots \\
\Sigma : \Psi; \Delta_1 \rightarrow A_1 \rho \Sigma : \Psi; \Delta_2, \ldots, \Delta_p \rightarrow A_2 \rho \otimes \ldots \otimes A_p \rho \\
\Sigma : \Psi; \Delta_1, \ldots, \Delta_p \rightarrow A_1 \rho \otimes \ldots \otimes A_p \rho
\]
and \( C \otimes C' \equiv (\bot \circ (\bot \circ C) \not\equiv (\bot \circ C')) \) and
3.4 Correctness and Completeness of FG

The correctness and completeness of the system FG is ensured by two theorems: one w.r.t. the TSG and the other w.r.t. FG.

3.4.1 Correctness and Completeness of FG w.r.t. TSG

We have to prove the following theorem.

**Theorem 1 (Correctness and Completeness of FG w.r.t. TSG)**

Given a TSG \((F, N', PP, (I, #))\), the corresponding TSG program \(\Psi\) and \(t \in T(\Sigma_I)\), \((\Sigma_{\Psi, #} : \rightarrow \sigma_{\Psi}(I)(t, #))\) has a proof in FG w.r.t. the TSG program \(\Psi\) if and only if \((I, #) \Rightarrow t\).

**Prologue to the proof of Correctness and completeness of FG w.r.t. TSG.**

The set of instantiated productions of a computation of a TSG \(Gr = (F, N', PP, TI)\) denoted \(IP(Gr)\) is the smallest set defined by:
Computing over TSG and PG

- if \( t \) is in \( \text{Comp}(Gr) \), \( t\uvec = (X, \beta) \), and \( R = (X \rightarrow g(Y_1, \ldots, Y_n)) \) is in \( PP_{\text{free}} \) then \( R(\beta) = ((X, \beta) \Rightarrow g(Y_1, \ldots, Y_n)) \) is in \( IP(Gr) \).
- if \( t \) is in \( \text{Comp}(Gr) \) and there exists \( n \) pairwise different occurrences \( u_1, \ldots, u_n \) of \( t \) such that \( \forall i \in [1, n], t\uvec = (X_i, c_i) \) and \( c_i = \beta \) and the pack of productions \( P = \{ R_1 = X_1 \rightarrow g_1(Y_1', \ldots, Y_{n_1}' \}, \ldots, R_r = X_r \rightarrow g_r(Y_r', \ldots, Y_{n_r}') \} \in PP \) (where \( \alpha \) is a new symbol)
  \[ P(\alpha, \beta) = \{ (X_1, \beta) \Rightarrow g_1(Y_1', \alpha, \ldots, Y_{n_1}', \alpha) \}, \ldots, (X_r, \beta) \rightarrow g_r(Y_r', \alpha, \ldots, Y_{n_r}', \alpha) \} \]
  = \{ R_1(\alpha, \beta), \ldots, R_r(\alpha, \beta) \} is included in \( IP(Gr) \).

Let \( \sigma_I : IP(Gr) \rightarrow \Sigma_{\text{formula}} \) be the mapping that translates every instantiated productions into a \( \Sigma_{\text{formula}} \):

\[
\sigma_I((X, \beta) \Rightarrow g((Y_1, \alpha), \ldots, (Y_n, \alpha))) = x(g(Y_1, \ldots, Y_n, \beta) \leftarrow y_1(Y_1, \alpha) \odot \ldots \odot y_n(Y_n, \alpha)
\]

A presynchronised production is a production of the form \((X, \beta) \Rightarrow C((Y_1, \alpha), \ldots, (Y_n, \alpha))\) where \( C \) is a context of terminal symbols, \( \beta, \alpha \) two new symbols and \( Y_1, \ldots, Y_n \) non terminals. The set of computations of a TSG \( Gr \) enlarged by presynchronised productions \( \Phi \) is \( \text{Comp}(Gr) \) plus

- \( t\uvec = ((X, \beta) \Rightarrow C((Y_1, \alpha), \ldots, (Y_n, \alpha))) \) if \( t \) is in the computation of \( Gr \) enlarged with \( \Phi \), \( t\uvec = (X, \beta) \) and \((X, \beta) \Rightarrow g((Y_1, \alpha), \ldots, (Y_n, \alpha)) \) is in \( \Phi \).

If \( s \) is the number of instances of packs of productions used during a TSG derivation \((t \Rightarrow t')\):

- \( \Sigma_{\Phi} \cup \{ \# \} \) plus the set of counters used during the derivation is denoted \( \Sigma(t \Rightarrow t') \) and
- the set of instances of synchronized productions is denoted \( \Phi(t \Rightarrow t') \).

If \( s \) is the number of instances of packs of productions used during a TSG derivation \((I, \# \Rightarrow t)\):

- the \( h^{th} \) pack of productions used during this TSG derivation is denoted \( P_h(\beta_h, \alpha_h), 1 \leq h \leq s \),
- the family of sets \( \{ \Sigma_h \}_{0 \leq h \leq s} \) is defined by induction: \( \Sigma_0 = \Sigma_{\Phi} \cup \{ \# \} \) and \( \forall h, 0 \leq h < s \), \( \Sigma_{h+1} = \Sigma_h \cup \{ \alpha_{h+1} \} \),
- the family of sets \( \{ \Delta^h \}_{0 \leq h \leq s} \) is defined by induction: \( \Delta^0 = \emptyset \) and \( \forall h, 0 \leq h < s \), \( \Delta^{h+1} = \Delta^h \cup \{ R_1, \ldots, R_p / R_1, \ldots, R_p = \sigma_I(P_{h+1}(\beta_{h+1}, \alpha_{h+1}))) \} \) (Remark: \( \Delta = \sigma_I(\Phi((I, \# \Rightarrow t))) \).

Proof of \( FG \Rightarrow TSG \).

First, we prove by structural induction that if there exists a \( FG \) proof (with only instances of the [Back !], [Back ?] and [1] rules) for \((\Sigma : \Delta^t \Rightarrow n(t, \beta))\) then there is a derivation \((N, \beta) \Rightarrow t) \) for the TSG enlarged by the set of presynchronised productions \( \Phi' \) with \( \sigma_I(\Phi') = \Delta^t \).

Base Cases: Only the case [Back ?] + [1] is treated, the case [Back !] + [1] is similar.

\[
\Sigma: \neg !^1 \Rightarrow (n(t, \beta) \leftarrow 1) \Rightarrow n(t, \beta)
\]

with the linear clause \((n(t, \beta) \leftarrow 1) \) from a presynchronised production \((N, \beta) \Rightarrow t) \). Then \( (N, \beta) \Rightarrow t \)
for the TSG enlarged with \{\{((N, \beta) \Rightarrow t)\} \} and \( \sigma_I(\{((N, \beta) \Rightarrow t)\}) \) = \{\{n(t, \beta) \leftarrow 1)\} \).
Induction cases: Only the case [Back ?] is treated, the case [Back !] is similar.

\[
\begin{align*}
\Sigma_i : \Delta_i \rightarrow n_1(t_1, \alpha) \\
\vdots \\
\Sigma_p : \Delta_p \rightarrow n_p(t_p, \alpha)
\end{align*}
\]
\[
\Sigma_1 : \Delta_1, \ldots, \Delta_p, n(g(N_1, \ldots, N_p, \beta) \Rightarrow n_1(N_1, \alpha) \otimes \ldots \otimes n_p(N_p, \alpha) \Rightarrow n(t, \beta))
\]

with \( \alpha \not\in \Sigma_i \).

By induction hypothesis \( \forall i, 1 \leq i \leq p \), there exists a derivation \((N_i, \beta) \Rightarrow t_i \) for the TSG enlarged with the set of presynchronised productions \( \Phi_i \) and \( \Delta_i = \sigma_i(\Phi_i) \). Then there exists a derivation \((N, \beta) \Rightarrow t \) for the TSG enlarged with the presynchronised productions

\[
\bigcup_{i=1}^{p} \Phi_i \cup \{(N, \beta) \rightarrow g((N_1, \alpha), \ldots, (N_p, \alpha))\}
\]

and

\[
\Delta^t = \bigcup_{i=1}^{p} \sigma_i(\Phi_i) \cup \{\sigma_i((N, \beta) \rightarrow g((N_1, \alpha), \ldots, (N_p, \alpha)))\}.
\]

\[\square\]

Synchronization Rule Introduction: We begin with a definition.

Definition 13 (Restricted FG proof) A FG proof is in restricted form if all the instances of the [Sync] rule are at the root of the proof.

First, we prove by induction on \( k \) if there exists a restricted FG proof for \((\Sigma_k, \# : \rightarrow i(t, \#))\):

\[
\begin{align*}
\vdash & \Sigma_i : \Delta^i \rightarrow i(t, \#)_{\text{[Sync]}^i} \\
& \vdots \\
\vdash & \Sigma_k, \# : \rightarrow i(t, \#)_{\text{[Sync]}^k}
\end{align*}
\]

then there exists a restricted FG proof for \((\Sigma_k : \Delta^k \rightarrow i(t, \#))\):

\[
\begin{align*}
\vdash & \Sigma_i : \Delta^i \rightarrow i(t, \#)_{\text{[Sync]}^i} \\
& \vdots \\
\vdash & \Sigma_k : \Delta^k \rightarrow i(t, \#)_{\text{[Sync]}^{k+1}}
\end{align*}
\]

The case \( k = 0 \) is trivial. Now we assume \( 0 < k \leq s \). By induction hypothesis, there exists a FG
there exists a derivation
\[ \gamma \]
with \( k \geq 1 \). Let \( \kappa = \{ \beta \in \Sigma_{k-1} : \beta \not\in \Sigma_k \} \) and \( \Theta = \{ c \leftarrow \beta, nc \leftarrow \alpha \} \) and a derivation
\[
(I, \#) \Rightarrow t
\]
for the TSG enlarged with \( \Phi^{k-1} \), \( \sigma_f(\Phi^{k-1}) = \Delta^{k-1} \).
But \( \Delta^k = \Delta^{k-1} \cup \{ \sigma_s(R_1, c, nc)\theta, \ldots, \sigma_s(R_p, c, nc)\theta \} \) and \( \Sigma_k = \Sigma_{k-1} \cup \{ \alpha \} \) then
\[
\gamma
\]
with \( \Sigma_k \) restricted \( FG \) proof for
\[
(I, \#) \Rightarrow t
\]
for the TSG enlarged with \( \Phi^k \). The case \( k = s \) is a consequence of the previous result. Now we assume that \( 0 \leq k < s \). By induction hypothesis on the proof (with \( P_{k+1} = \{ R_1, \ldots, R_p \}, p > 1 \):
Proof of $TSG \Rightarrow FG$.
First, we prove by induction on the structure of derivation: if there exists a derivation

$$g((N_1, \beta), \ldots, (N_r, \beta)) \Rightarrow g(t_1, \ldots, t_l)$$

for a $TSG$ then there exists for every $k$, $1 \leq k \leq l$, a $FG$ proof for

$$(\Sigma((N_k, \beta)) \Rightarrow t_k) : \sigma_k((\Phi((N_k, \beta)) \Rightarrow t_k)) \rightarrow n_k(t_k, \beta)$$

(2)

**Base cases:** Only the case with a pack of productions is treated, the case with a free production is similar.
Let $\{N_1 \rightarrow t_1, \ldots, N_r \rightarrow t_r\} \in PP_{sync}$ and the derivation ($g((N_1, \beta), \ldots, (N_r, \beta)) \Rightarrow g(t_1, \ldots, t_r)$). Then for every $k$, $1 \leq k \leq r$ there exists a $FG$ proof:

$$\Sigma \vdash (n_k(t_k, \beta) \Rightarrow 1) \rightarrow n_k(t_k, \beta)$$

with

$$\Phi((N_k, \beta) \Rightarrow t_k) = \{(N_k, \beta) \Rightarrow t_k\}$$

$$\sigma_k((\Phi((N_k, \beta) \Rightarrow t_k)) = \{n_k(t_k, \beta) \Rightarrow 1\}$$

$$\Phi(g((N_1, \beta), \ldots, (N_r, \beta)) \Rightarrow g(t_1, \ldots, t_r)) = \{(n_1(t_1, \beta) \Rightarrow 1), \ldots, (n_r(t_r, \beta) \Rightarrow 1)\}.$$**Induction cases:** Only the case with a pack of productions is treated, the case with a free production is similar. Let

$$\{N_1 \rightarrow g_1(N_1^{\alpha_1}, \ldots, N_1^{\alpha_1_{p_1}}), \ldots, N_r \rightarrow g_r(N_r^{\alpha_r}, \ldots, N_r^{\alpha_r_{p_r}})\} \in PP_{sync}$$

and a derivation

$$g((N_1, \beta), \ldots, (N_r, \beta))$$

$$\Rightarrow g(g((N_1^1, \alpha), \ldots, (N_1^{p_1}, \alpha)), \ldots, g_r((N_r^1, \alpha), \ldots, (N_r^{p_r}, \alpha)))$$

$$\Rightarrow g(g(t_1^1, \ldots, t_r^{p_r}), \ldots, g_r(t_1^1, \ldots, t_r^{p_r}))$$

Then by induction hypothesis, for every $k$, $1 \leq k \leq r$ and $i_k, 1 \leq i_k \leq p_k$ there exists a $FG$ proof:

$$\delta_k = \frac{\nabla_k}{\Sigma((N_k^i, \alpha) \Rightarrow i_k^m) : \sigma_k((\Phi((N_k^i, \alpha) \Rightarrow i_k^m)) \rightarrow n_k^i(t_k^i, \alpha)}$$

For every $k$, $1 \leq k \leq r$.

$$\sigma_k((\Phi((N_k, \beta) \Rightarrow g_k(t_1^1, \ldots, t_k^{p_k})))$$

$$= \sigma_k(\bigcup_{i=1}^{p_k} \Phi((N_k^i, \alpha) \Rightarrow i_k^m) \cup \{(N_k, \beta) \Rightarrow g_k((N_k^2, \alpha), \ldots, (N_k^{p_k}, \alpha))\})$$

$$= \bigcup_{i=1}^{p_k} \sigma_k((\Phi((N_k^i, \alpha) \Rightarrow i_k^m)) \cup \{(N_k, \beta) \Rightarrow g_k((N_k^2, \alpha), \ldots, (N_k^{p_k}, \alpha))\})$$

$$= \bigcup_{i=1}^{p_k} \sigma_k((\Phi((N_k^i, \alpha) \Rightarrow i_k^m)) \cup \{n_k(g_k(N_1^i, \ldots, N_k^{p_k}), \beta) \Rightarrow n_k^i(N_k^i, \alpha), \ldots, n_k^{p_k}(N_k^{p_k}, \alpha)\}$$
Then for every \( k, 1 \leq k \leq r \) there exits a FG proof (the signatures of \( \delta^i_k, 1 \leq i_k \leq p_k \), are augmented with \( \beta \)):

\[
\Sigma((N_k, \beta) \xrightarrow{g_k} (t^1_k, \ldots, t^{p_k}_k)) : \sigma_{I}(\Phi((N_k, \beta) \xrightarrow{g_k} (t^1_k, \ldots, t^{p_k}_k))) \rightarrow n_k(g_k(t^1_k, \ldots, t^{p_k}_k), \beta)
\]

\[
\sum_{i=1}^{p_k} \Sigma((N_k, \beta) \xrightarrow{g_k} (t^1_k, \ldots, t^{p_k}_k)) = \bigcup_{i=1}^{p_k} \Sigma((N^i_k, \alpha) = t^i_k) \cup \{ \beta \}
\]

and

\[
\Phi(g((N_1, \beta), \ldots, (N_r, \beta))) \xrightarrow{g} (g_1(t^1_1, \ldots, t^{p_1}_1), \ldots, g_r(t^1_r, \ldots, t^{p_r}_r)) = \bigcup_{k=1}^{r} \Phi((N_k, \beta) \xrightarrow{g_k} (t^1_k, \ldots, t^{p_k}_k))).
\]

By the result above, we have the following corollary:

**Corollary 1** If there exists a derivation \((I, \#) \xrightarrow{t}\) for a TSG then there exists a FG proof for \((\Sigma_i : \Delta^i \rightarrow i(t, \#))\).

Now we prove by induction on \( h \): if there exists a FG proof for \((\Sigma_s : \Delta^s \rightarrow i(t, \#))\) then there exists an FG proof for \((\Sigma_h : \Delta^h \rightarrow i(t, \#))\).

The base case \( h = 0 \) is trivial, now we assume that \( 0 < h \leq s \). \( \Sigma_{h+1} = \Sigma_h \cup \{ \alpha_{h+1} \} \) and \( \Delta^{h+1} = \Delta^h \cup \{ R^{h+1}_1, \ldots, R^{h+1}_p / (R^{h+1}_1, \ldots, R^{h+1}_p) = \sigma_{I}(P_{h+1}(\beta_{h+1}, \alpha_{h+1})) \}. \) By induction hypothesis, there exists a FG proof for:

\[
\forall \\
\Sigma_h, \alpha_{h+1} : \Delta^h, R^{h+1}_1, \ldots, R^{h+1}_p \rightarrow i(t, \#)
\]

so there exists a FG proof:

\[
\forall \\
\Sigma_h, \alpha_{h+1} : \Delta^h, R^{h+1}_1, \ldots, R^{h+1}_p \rightarrow i(t, \#)
\]

\[\mathtt{sync}\]

This achieves the first part of the correctness and completeness of our transformation. We now have to provide a similar theorem for Primal Grammars.

### 3.4.2 Correctness and Completeness of FG w.r.t. PG

**Theorem 2** (Correctness and Completeness of FG w.r.t. PG)

Given a PG \((\mathcal{F}, \mathcal{D}, \mathcal{R}, A)\), the corresponding PG program \(\Psi\) and \(\bar{c} \in \mathbb{N}^{\text{ntc}}(A)\) and \(t \in T(\mathcal{F})\). \((\Sigma_{\Psi} : \Psi; \rightarrow \sigma_{\mathcal{D}}(A)(t, \bar{c}))\) has a proof in FG if and only if \((A(\bar{c}) \Rightarrow t)\).
Proof of \( PG \implies FG \).

We prove by induction on the length of a \( PG \) derivation if there exists a \( PG \) derivation 
\( (\tilde{f}(m^0, \tilde{m})) \rightarrow_{\text{pg}} t \), with \( \tilde{f} \in D \) and \( m^0 \in \mathbb{N} \) and \( \tilde{m} \in N^a(f)^{-1} \) then there exists a \( FG \) proof for \( (\Sigma_\Psi : \Psi; \rightarrow f(t, m^0, \tilde{m})) \).

The right hand-side of the sequent \( (\Sigma_\Psi : \Psi;) \) does not vary and is omitted.

**Case \( l = 1 \):** \( \tilde{f}(m^0, \tilde{m}) \rightarrow_{\text{pg}} t \) then \( m^0 = 0 \) and there exists a basic rule \( (\tilde{f}(0, \tilde{c}) \rightarrow t) \in \mathcal{R} \) and a linear clause \( (\forall \tilde{c}(f(t, 0, \tilde{c}) \lor 1)) \in \Psi \) and a \( FG \) proof:

\[
\begin{array}{c}
\rightarrow t[1] \\
\rightarrow f(t, 0, \tilde{m})[\text{Back}]
\end{array}
\]

**Case \( l > 1 \):** Only the case with the last applied rule as an instance of the second inductive rule \([\text{II}]\) is treated, the case for the first inductive rule is similar. So \( m^0 = s(m^{-1}) \) and there exists a context \( C \) with \( t = C(t_1, \ldots, t_p)[t'] \) and an inductive rule \( (\tilde{f}(s(c^0), \tilde{c}) \rightarrow_{\text{pg}} C(\tilde{f}_1(\tilde{c}_1), \ldots, \tilde{f}_p(\tilde{c}_p)))[\tilde{f}(c^0, \tilde{c}^+)]) \in \mathcal{R} \).

So \( \forall i \in [1..p], \tilde{f}_i(\tilde{m}_i) \rightarrow_{\text{pg}} t_i, \tilde{f}(m^{-1}, \tilde{m}^+) \rightarrow_{\text{pg}} t' \) and \( \Sigma^p_{i=1} k_i + k' = l - 1 \). Then by induction hypothesis, there exist \( FG \) proofs for \( \forall i \in [1..p], (\rightarrow f(t_i, m_i)) \):

\[
\begin{array}{c}
\delta_i \\
\rightarrow f(t_i, m_i)
\end{array}
\]

and there exists a \( FG \) proof for \( (\rightarrow f(t', m^{-1}, \tilde{m}^+)) \):

\[
\begin{array}{c}
\delta' \\
\rightarrow f(t', m^{-1}, \tilde{m}^+)
\end{array}
\]

Moreover,

\[
\begin{aligned}
\sigma_{\mathcal{R}}(f(s(c^0), \tilde{c}) & \rightarrow_{\text{pg}} C(\tilde{f}_1(\tilde{c}_1), \ldots, \tilde{f}_p(\tilde{c}_p))[\tilde{f}(c^0, \tilde{c}^+)]) = \\
& \forall c^0 \forall \tilde{c} F_1 \cdots F_p \forall F(f(C(F_1, \ldots, F_p)[F], s(c^0), \tilde{c})) \\
& \rightarrow f_1(F_1, \tilde{c}_1) \otimes \cdots \otimes f_p(F_p, \tilde{c}_p) \otimes f(F(c^0, \tilde{c}^+))
\end{aligned}
\]

Then there exists a \( FG \) proof for \( (\rightarrow f(C(t_1, \ldots, t_p)[t'], s(m^{-1}), \tilde{m})) \):

\[
\begin{array}{c}
\delta_1 \\
\delta_p \\
\delta'
\end{array}
\]

\[
\begin{array}{c}
\rightarrow f(t_1, m_1) \\
\rightarrow f(t_p, m_p) \\
\rightarrow f(t', m^{-1}, \tilde{m}^+)[\text{Back}]
\end{array}
\]

\[\text{We write } k = ar(f) \text{ and } \tilde{c} = (c^1, \ldots, c^k) \text{ and } \tilde{m} = (m^1, \ldots, m^k) \text{ and } \tilde{c}^+ = (c^1, \ldots, c^{k-1}, s(c^k), c^{k+1}, \ldots, c^p) \text{ and } \tilde{m}^+ = (m^1, \ldots, m^{k-1}, s(m^k), m^{k+1}, \ldots, m^p) \text{ and } \forall i \in [1..p], \tilde{c}_i \text{ is a subsequence of } \tilde{c} \text{ and } \tilde{m}_i \text{ is a subsequence of } \tilde{m} \text{ on the same positions.} \]
Proof of $PG \iff FG$.

We prove by induction on the structure of a $FG$ proof if there exists a $FG$ proof for

$$(\Sigma_\Psi: \Psi_i \rightarrow f(t, m^0, \bar{m})),$$  \hspace{1cm} (4)

with $\bar{f} \in D$ and $m^0 \in \mathbb{N}$ and $\bar{m} \in \mathbb{N}^{ar(\bar{f})-1}$ then there exists a $PG$ derivation $(\bar{f}(m^0, \bar{m}) \Rightarrow_{PG} t)$

The right hand-side of the sequent $(\Sigma_\Psi: \Psi_i)$ does not vary and is omitted.

**Base case:** There exists a proof of $\Rightarrow_{PG} (f(0, \bar{m}) \Rightarrow_{PG} t)$ and a basic rule $\bar{f}(0, \bar{m}) \Rightarrow_{PG} t$.

**Induction cases:** There exists a $FG$ proof for $(\rightarrow f(t, m^0, \bar{m}))$. The only possible inference rule applied at the root of the proof is an instance of the $[\text{Back}!]$ rule. Only the case for a linear clause obtained from the second inductive rule is treated, the cases for the basic rule or the first inductive rule are similar.

We have a $FG$ proof of shape

$$\Delta_1 \ldots \Delta_p \delta' [\text{Back}!]$$

with the linear clause

$$\forall \bar{c} \forall \vec{v} F_1 \ldots F_p \forall F(f(C(F_1, \ldots, F_p)[F], s(\bar{c}), \bar{c}) \Rightarrow$$

$$f_1(F_1, \bar{c}_1) \otimes \ldots \otimes f_p(F_p, \bar{c}_p) \otimes f(F, \bar{c}, \bar{c}^+) \in \Psi$$

(obtained from the rule $\bar{f}(s(\bar{c}), \bar{c}) \Rightarrow C(f_1(\bar{c}_1), \ldots, f_p(\bar{c}_p))[f(\bar{c}, \bar{c}^+)] \in \mathbb{R}$ with $t = C(t_1, \ldots, t_p)[t']$ and $\forall i \in [1..p] \delta_i$ a proof with the root $(\rightarrow f_i(t_i, \bar{m}_i))$ and $\delta'$ a proof with the root $(\rightarrow n(t', m^0, m^+), t')$.

For all proofs $\delta_i, i \in [1..p], \text{ by induction hypothesis, there exist PG derivations } \bar{f}_i(\bar{m}_i) \Rightarrow_{PG} t_i \text{ and for the proof } \delta'$, by induction hypothesis, there exists a PG derivation $\bar{f}(m^0, m^+) \Rightarrow_{PG} t'$. Then by the inductive rule $\bar{f}(s(\bar{c}), \bar{c}) \Rightarrow_{PG} C(\bar{f}_1(\bar{c}_1), \ldots, \bar{f}_p(\bar{c}_p))[N(\bar{c}, \bar{c}^+) \in \mathbb{R}, \bar{f}(m^0, m^+ \Rightarrow_{PG} t$. 

\hspace{1cm} □

### 3.4.3 Application

We address here Example on $TSG$ to illustrate how the transformation and proof system $FG$ work. We build the derivation tree for the $TSG$ program $\Psi$. $\Sigma_\Psi$ is omitted in the left hand side of the sequents of the following proof in order to simplify the notation, we only mention the new symbols added to the signature along the proof and universal quantifiers are omitted in the sequents:

---

\[\begin{align*}
\text{We denote } k = ar(\bar{f}) \text{ and } \bar{c} = (c_1, \ldots, c_p) \text{ and } \bar{m} = (m_1, \ldots, m_q) \text{ and } \bar{c}^+ = (c_1, \ldots, c_p, s(c), c^{p+1}, \ldots, c^+) \text{ and } m^+ = (m_1, \ldots, m_{q+1}, s(m), m_{q+1}, \ldots, m_q) \text{ and } \forall i \in [1..p], \bar{c}_i \text{ is a subsequence of } \bar{c} \text{ and } \bar{m}_i \text{ a subsequence of } \bar{m} \text{ on the same positions.}
\end{align*}\]
\[
\{ F_0 : \forall c (\forall X \forall Y (i(f(X,Y),c) \land x(X,c) \otimes y(Y,c))), \\
F_1 : \forall c (x(z,c) \land 1), \\
F_2 : \forall c (y(z,c) \land 1), \\
F_3 : \forall c \exists nc ((\forall X (x(s(X),c) \land x(X,nc))) \otimes (\forall Y (y(s(Y),c) \land y(Y,nc)))) \} 
\]

\[
\begin{align*}
\delta_1 & \quad \delta_2 \\
\#, \beta : x(s(X),\#) & \land x(X,\#) \land y(s(Y),\#) \land y(Y,\beta) \implies i(f(s(z),s(s(z))),\#) & [\text{Sync}] F_3 \\
\# & \implies i(f(s(z),s(s(z))),\#) & [\text{Sync}] F_3
\end{align*}
\]

where

\[
\delta_1 : \left\{ \begin{array}{l}
1 \\
\#, \beta : x(z,\beta) & [\text{Back !}] F_1 \\
\#, \beta : x(s(X),\#) & [\text{Back ?}] \\
\#, \beta : x(s(X),\#) \land x(X,\#) \implies x(s(z),\#)
\end{array} \right.
\]

and

\[
\delta_2 : \left\{ \begin{array}{l}
1 \\
\#, \beta : y(z,\beta) & [\text{Back !}] F_2 \\
\#, \beta : y(s(Y),\#) & [\text{Back ?}] \\
\#, \beta : y(s(Y),\#) \land y(Y,\beta) \implies y(s(s(z),\#)
\end{array} \right.
\]

4 From Linear Logic to Prolog

From now on, there is no more need to distinguish TSG and PG since they have been embedded into the same formalism (i.e. linear logic formulas). From an operational point of view, it appears clearly that the previous system FG does not provide any strategy for a proof search (especially concerning the use of the rule [\text{Sync}]). Furthermore, a refinement will help us to get the implementation of the system. Therefore, we define a goal directed proof system \( FG^{\text{dir}} \) with the following inference rules.

We define the set operator \( \sqcup \) as \( \sqcup_{1 \leq i \leq n} A_i = \bigcup_{1 \leq i \leq n} A_i \) such that \( \forall 1 \leq i, j \leq n, i \neq j, A_i \cap A_j = \emptyset. \)
Definition 14 (\(FG^{dir}\) system)

\[ \Sigma : \left( (\Delta_1 \rightarrow G_1), \ldots, (\Delta_{i-1} \rightarrow G_{i-1}), \right. \]
\[ \left. (\Delta'_1 \rightarrow A_1 \sigma), \ldots, (\Delta'_p \rightarrow A_p \sigma), \right. \]
\[ \left. (\Delta_{i+1} \rightarrow G_{i+1}), \ldots, (\Delta_n \rightarrow G_n) \right) \]

\( \Sigma : \left( (\Delta_1 \rightarrow G_1), \ldots, (\Delta_n \rightarrow G_n) \right) \)

if \( H \circ - A_1 \otimes \ldots \otimes A_p \in \Psi \), there exists a substitution \( \sigma \) such that \( H \sigma = G_i \) and \( \Delta_i = \{ 1 \leq k \leq p \Delta'_k \} \).

\[ \Sigma : \left( (\Delta_1 \rightarrow G_1), \ldots, (\Delta_n \rightarrow G_n) \right) \]

if \( C = (H \circ - A_1 \otimes \ldots \otimes A_p) \in \Delta_i \), there exists a substitution \( \sigma \) such that \( H \sigma = G_i \) and \( \Delta_i \setminus \{ C \} = \{ 1 \leq k \leq p \Delta'_k \} \).

\[ \Sigma, \alpha : \left( (\Delta_1 \cup \Delta''_1 \rightarrow G_1), \ldots, (\Delta_{i-1} \cup \Delta''_{i-1} \rightarrow G_{i-1}), \right. \]
\[ \left. (\Delta'_1 \cup \Delta''_1 \rightarrow A_1 \sigma), \ldots, (\Delta'_p \cup \Delta''_p \rightarrow A_p \sigma), \right. \]
\[ \left. (\Delta_{i+1} \cup \Delta''_{i+1} \rightarrow G_{i+1}), \ldots, (\Delta_n \cup \Delta''_n \rightarrow G_n) \right) \]

\( \Sigma : \left( (\Delta_1 \rightarrow G_1), \ldots, (\Delta_n \rightarrow G_n) \right) \)

if \( \forall c \exists n(c \otimes \ldots \otimes C_r) \in \Psi, 1 \leq j \leq r \),
\( C_j \theta = (H \circ - A_1 \otimes \ldots \otimes A_p) \),
\( H \sigma = G_i, \Delta_i = \{ 1 \leq k \leq p \Delta'_k \} \),
\( \{ C_1 \theta, \ldots, C_{j-1} \theta, C_{j+1} \theta, \ldots, C_r \theta \} = \{ 1 \leq k \leq p \Delta''_k \} \),
\( \beta \in \Sigma, \alpha \not\in \Sigma \) and \( \theta = \{ c \not\leftarrow \beta, n c \not\leftarrow \alpha \} \).

\[ \Sigma, \alpha : \left( (\Delta_1 \cup \Delta'_1 \rightarrow G_1), \ldots, (\Delta_{i-1} \cup \Delta'_{i-1} \rightarrow G_{i-1}) \right. \]
\[ \left. (\Delta_{i+1} \cup \Delta'_{i+1} \rightarrow G_{i+1}), \ldots, (\Delta_n \cup \Delta'_n \rightarrow G_n) \right) \]

\( \Sigma : \left( (\Delta_1 \rightarrow G_1), \ldots, (\Delta_n \rightarrow G_n) \right) \)
if
\[
\forall c \exists nc(C_1 \otimes \ldots \otimes C_r) \in \Psi, 1 \leq j \leq r; C_j \theta = (H \circ I),
\]
\[
H \sigma = G_i, \Delta_i = \emptyset,
\]
\[
C_1 \theta, \ldots, C_{j-1} \theta, C_{j+1} \theta, \ldots, C_r \theta = (\cup_{1 \leq k \leq n} \Delta'_k),
\]
\[
\beta \in \Sigma, \alpha \not\in \Sigma \text{and } \theta = \{c \leftarrow \beta, nc \leftarrow \alpha\}.
\]

[Axiom \(\text{dir}\)]
\[
\Sigma : (\Delta_1 \rightarrow G_1), \ldots, (\Delta_{i-1} \rightarrow G_{i-1}),
\]
\[
(\Delta_{i+1} \rightarrow G_{i+1}), \ldots, (\Delta_n \rightarrow G_n)
\]
if \(H \circ I \in \Psi, \Delta_i = \emptyset\) and there exists a substitution \(\sigma\) such that \(H \sigma = G_i\).

[Axiom \(\text{dir}^\text{?}\)]
\[
\Sigma : (\Delta_1 \rightarrow G_1), \ldots, (\Delta_{i-1} \rightarrow G_{i-1}),
\]
\[
(\Delta_{i+1} \rightarrow G_{i+1}), \ldots, (\Delta_n \rightarrow G_n)
\]
if \(\Delta_i = \{H \circ I\}, \) there exists a substitution \(\sigma\) such that \(H \sigma = G_i\).

The main difference introduced by this system is that synchronizations and reduction are led by the goal to be solved. We now have to prove that this system is equivalent to our initial system \(FG\). The equivalence of the two systems is ensured by the following theorem:

**Theorem 3 (\(FG\text{dir} \iff FG\))** There exists a proof of a sequent in the system \(FG\) if and only if there exists a proof of this sequent in the system \(FG\text{dir}\).

**Proof:** This proof is inspired by [19].

The transformation of a \(FG\) proof tree into a \(FG\text{dir}\) proof derivation is achieved by recombining the different branches of the proof into a single derivation and then collapsing the synchronization rule with its first corresponding linear clause. The intermediate system \(FG\text{lin}\) is needed to achieve the first part of the proof.

**Definition 15 (\(FG\text{lin}\) System)**

[Back \(\text{lin}\)]
\[
(\Sigma_1 : \Delta_1 \rightarrow G_1), \ldots, (\Sigma_{i-1} : \Delta_{i-1} \rightarrow G_{i-1}),
\]
\[
(\Sigma_i : \Delta'_1 \rightarrow A_1 \sigma), \ldots, (\Sigma_i : \Delta'_p \rightarrow A_p \sigma),
\]
\[
(\Sigma_{i+1} : \Delta_{i+1} \rightarrow G_{i+1}), \ldots, (\Sigma_n : \Delta_n \rightarrow G_n)
\]
\[
(\Sigma_1 : \Delta_1 \rightarrow G_1), \ldots, (\Sigma_n : \Delta_n \rightarrow G_n)
\]
where \((H \circ A_1 \otimes \ldots \otimes A_p) \in \Psi, H \sigma = G_i\) and \(\Delta_i = \cup_{1 \leq k \leq p} \Delta'_k\).
Lemma 1 \((\text{FG} \iff \text{FG}^\text{lin})\)

There exists proofs of the sequents \(\forall i, 1 \leq i \leq n, (\Sigma_i : \Delta_i \rightarrow G_i)\) in the system FG if and only if there exists a proof of the sequent \((\Sigma_1 : \Delta_1 \rightarrow G_1) \ldots (\Sigma_n : \Delta_n \rightarrow G_n)\) in the system \(\text{FG}^\text{lin}\).

**Proof of \(\text{FG} \implies \text{FG}^\text{lin}\).**

We prove by induction on the structure of a FG proof if \(\forall i, 1 \leq i \leq n\) there exists a FG proof of \((\Sigma_i : \Delta_i \rightarrow G_i)\) then there exists a \(\text{FG}^\text{lin}\) proof of \((\Sigma_i : \Delta_i \rightarrow G_i)\).

**Base case** \([\text{Back ?}] + [1]:\)

The FG proof is as follows:

\[
\frac{}{\Sigma : \vdash 1} \quad \Sigma : H \circ \rightarrow 1 \rightarrow G \quad \text{with } H\sigma = G
\]

Then the derivation:

\[
\frac{}{\Sigma : G} \quad \Sigma : \vdash G \quad \text{(Axiom \(\text{lin}\))}
\]

is a \(\text{FG}^\text{lin}\) proof.
Base case \([\textit{Back} !] + [1]\): This case is similar to the \([\textit{Back} ?] + [1]\) base case.

Induction case \([\textit{Sync}]\): The inference at the root of the proof is an instance of the \([\textit{Sync}]\) rule:

\[
\begin{array}{c}
\nabla \\
\Sigma : \Delta \rightarrow G
\end{array}
\]

with \((\forall c \exists nc(C_1 \otimes \ldots \otimes C_p)) \in \Psi, \beta \in \Sigma, \alpha \notin \Sigma \text{ and } \theta = \{c \leftarrow \beta, nc \leftarrow \alpha\}\). By induction hypothesis, for the \(FG\) proof tree:

\[
\begin{array}{c}
\nabla \\
\Sigma : \Delta \rightarrow G
\end{array}
\]

there exists a \(FG\) \(^{lin}\) proof:

\[
\begin{array}{c}
D \\
(\alpha, \Sigma : \Delta \rightarrow G)
\end{array}
\]

Then the following derivation:

\[
\begin{array}{c}
D \\
(\alpha, \Sigma : \Delta \rightarrow G)
\end{array}
\]

is a \(FG\) \(^{lin}\) proof.

Induction case \([\textit{Back} ?]\): The inference at the root of the proof is an instance of the \([\textit{Back} ?]\) rule:

\[
\begin{array}{c}
\nabla_1 \\
\ldots \\
\nabla_p \\
\Sigma : \Delta_i \rightarrow A_i \sigma \\
\Sigma : H \circ A_1 \otimes \ldots \otimes A_p, \Delta_1, \ldots, \Delta_p \rightarrow G
\end{array}
\]

with \(H \sigma = G\). By induction hypothesis for every \(1 \leq i \leq n\) there is a \(FG\) \(^{lin}\) proof:

\[
\overline{S_i}
\]

\[
\{S_j\}_{j \in I_i}
\]

Let give the following \(FG\) \(^{lin}\) derivation \(\forall i, 1 \leq i \leq p\):

\[
D_i \left\{ \begin{array}{c}
\overline{S_i} \cup \{\Sigma : \Delta_k \rightarrow A_k(\sigma)\}_{i < k \leq p} \\
\{S_j\}_{j \in I_i} \cup \{\Sigma : \Delta_k \rightarrow A_k(\sigma)\}_{i < k \leq p} \\
\ldots \\
(\Sigma : \Delta_i \rightarrow A_i(\sigma)) \cup \{\Sigma : \Delta_k \rightarrow A_k(\sigma)\}_{i < k \leq p}
\end{array} \right\}
\]
Then the derivation:

\[
\begin{align*}
& S_l^p \\
& D_p \quad \frac{(\Sigma : \Delta_p \rightarrow A_p \sigma)}{S_{l-1}^p, (\Sigma : \Delta_p \rightarrow A_p \sigma)} \\
& \vdots \\
& \frac{(\Sigma : \Delta_i \rightarrow A_i \sigma), \{(\Sigma : \Delta_k \rightarrow A_k \sigma)\}_{1 \leq i \leq k \leq p}}{S_{l-1}^p, \{(\Sigma : \Delta_k \rightarrow A_k \sigma)\}_{1 \leq i \leq k \leq p}} \\
& \vdots \\
& \frac{(\Sigma : \Delta_1 \rightarrow A_1 \rho), \{(\Sigma : \Delta_k \rightarrow A_k \sigma)\}_{1 < k \leq p}}{(\Sigma : \Delta_1, \ldots, \Delta_p \rightarrow G)}
\end{align*}
\]

is a \(FG^{lin}\) proof.

**Induction case [Back ?]:** This case is similar to the [Back ?] case.

\[\square\]

**Proof of \(FG^{lin} \Rightarrow FG\).**

We prove by induction on the structure of a \(FG^{lin}\) proof if there exists a \(FG^{lin}\) proof of \((\Sigma : \Delta_1 \rightarrow G_1), \ldots, (\Sigma : \Delta_n \rightarrow G_n)\) then \(\forall i, 1 \leq i \leq n\) there exist \(FG\) proofs for \((\Sigma : \Delta_i \rightarrow G_i)\).

**Base case [Axiom \(?^{lin}\):** The \(FG^{lin}\) proof is as follows:

\[
(\Sigma : H \circ{-} 1 \rightarrow G)^{\text{Axiom } ?^{lin} } \quad \text{with } H \sigma = G.
\]

The proof tree:

\[
\begin{align*}
& \Sigma : 1^1 \\
& \Sigma : H \circ{-} 1 \rightarrow G^{\text{Back ? }}
\end{align*}
\]

is a \(FG\) proof.

**Base case [Axiom \(!^{lin}\):**

This case is similar to the [Axiom ?] base case.

**Induction case [Back \(!^{lin}\):** The inference at the root of the proof is an instance of the [Back \(!^{lin}\)] rule (without loss of generality \(i = 1\)):

\[
\begin{align*}
& D \\
& \frac{(\Sigma : \Delta_1 \rightarrow A_1 \sigma), \ldots, (\Sigma : \Delta_p \rightarrow A_p \sigma), (\Sigma : \Delta_2 \rightarrow G_2), \ldots, (\Sigma : \Delta_n \rightarrow G_n)}{(\Sigma : \Delta_1, H \circ{-} A_1 \otimes \ldots \otimes A_p \rightarrow G_1), (\Sigma : \Delta_2 \rightarrow G_2), \ldots, (\Sigma : \Delta_n \rightarrow G_n)}
\end{align*}
\]
with $H\sigma = G_1$. By induction hypothesis, $\forall j, 1 \leq j \leq p$, there exists a FG proof:

$$
\frac{\Sigma_j : \Delta'_j \rightarrow A_j \sigma}{\Sigma_j : \Delta'_j \rightarrow A_j \sigma}
$$

and $\forall i, 2 \leq i \leq n$, there exists a FG proof:

$$
\frac{\Sigma_i : \Delta_i \rightarrow G_i}{\Sigma_i : \Delta_i \rightarrow G_i}
$$

Then the proof tree:

$$
\frac{\Sigma_1 : \Delta'_1 \rightarrow A_1 \sigma \ldots \Sigma_p : \Delta'_p \rightarrow A_p \sigma}{\Sigma_1 : \Delta'_1, \ldots, \Delta'_p \rightarrow G_1}
$$

is a FG proof.

**Induction cases** $[\text{Axiom} \ D_{lin}], \ [\text{Axiom} \ Q_{lin}]$ and $[\text{Back} \ D_{lin}]$: The cases $[\text{Axiom} \ D_{lin}], \ [\text{Axiom} \ Q_{lin}]$ and $[\text{Back} \ D_{lin}]$ are similar to the $[\text{Back} \ Q_{lin}]$ induction case.

**Induction case** $[\text{Sync} \ lin]$: The inference at the root of the proof is an instance of the $[\text{Sync} \ lin]$ rule (without loss of generality $i = 1$):

$$
D \begin{array}{c}
(S_1, \alpha : C_1 \theta, \ldots, C_r \theta, \Delta_1 \rightarrow G_1), (S_2 : \Delta_2 \rightarrow G_2), \ldots, (S_n : \Delta_n \rightarrow G_n) \\
(S_1 : \Delta_1 \rightarrow G_1), \ldots, (S_n : \Delta_n \rightarrow G_n)
\end{array}
$$

with $\forall c \exists nc(C_1 \otimes \ldots \otimes C_r) \in \Psi$, $\beta \in \Sigma_1$, $\alpha \not\in \Sigma_1$ and $\theta = \{c \leftarrow \beta, nc \leftarrow \alpha\}$. By induction hypothesis, there exist $\forall i, 2 \leq i \leq n$, a FG proof:

$$
\frac{\Sigma_i : \Delta_i \rightarrow G_i}{\Sigma_i : \Delta_i \rightarrow G_i}
$$

and a FG proof:

$$
\frac{\Sigma_1, \alpha : C_1 \theta, \ldots, C_r \theta, \Delta_1 \rightarrow G_i}{\Sigma_1, \alpha : C_1 \theta, \ldots, C_r \theta, \Delta_1 \rightarrow G_i}
$$

Then the proof tree:

$$
\frac{\Sigma_1, \alpha : C_1 \theta, \ldots, C_r \theta, \Delta_1 \rightarrow G_i}{\Sigma_1, \alpha : \Delta_1 \rightarrow G_i}
$$

is a FG proof.

$\square$

The following definitions and lemmas are technical ones to establish the equivalence between $FG_{lin}$ and $FG_{dir}$ systems.
Lemma 2 (FG Restriction Lemma)
If there exists a FG proof then there exists a restricted FG proof.

Proof of the FG restriction.
By (a double) induction on the distance between the closer instance of the [Sync] rule to the root and the root, (and the number of instances of [Sync] rule which is straight forward), we prove that if there exists a FG derivation then there exists a restricted FG derivation with the same first and last resolvents.

Base case: An instance of the [Sync] rule is already at the root.

Induction case: The instance of the rule at the root of the derivation is either an instance of [Back !], [Back ?] or [I] rules. The [I] is trivial. The [Back ?] case is only treated, the [Back !] is similar. The derivation is of the form (without loss of generality $i = 1$):

\[
\begin{array}{c}
\Sigma : \Delta_1 \rightarrow A_1 \sigma \ldots \Sigma : \Delta_p \rightarrow A_p \sigma \\
\Sigma : \Delta \rightarrow G
\end{array}
\]

with

\[
C = (H \circ A_1 \otimes \ldots \otimes A_p) \in \Delta,
\]

\[
H \sigma = G,
\]

\[
\Delta \setminus \{C\} = \{\psi_{1 \leq k \leq p} \Delta_k\}.
\]

By induction hypothesis, there exists a FG derivation $\nabla'$ with an instance of the [Sync] rule at the root, the same number of instances of the [Sync] rule and the same first and last resolvents as $\nabla$. A restricted FG derivation is obtained by permuting this instance of the [Sync] rule and the instance of the [Back ?] rule and inserting in the signature of every sequents of the descendants of $(\Sigma : \Delta_k \rightarrow A_k \sigma) \ (\forall k, 1 \leq k \leq p)$ in the derivation $\nabla'$ the constant $\alpha$ inserted by the instance of the [Sync] rule.

At this stage, we need more intermediate definitions to continue the proof.

Definition 16 (Restricted FG lin proof)
A FG lin proof is in restricted form if all the instances of the Sync lin rule are at the root of the proof.

Definition 17 (Initial instance)
The initial instance w.r.t. an instance of the Sync lin rule is the instance of Sync lin or Axiom lin rules which uses for the first time one of the linear clauses introduced by the Sync lin rule.

Definition 18 (Ordered FG lin proof)
A FG lin proof is ordered if the order on the Eigenvariables induced by the instances of the Sync lin rules is the same as the one induced by their initial instances.
Lemma 3 (\(\text{FG}^{\text{lin}}\) Restriction Lemma)
If there exists a \(\text{FG}^{\text{lin}}\) proof then there exists an restricted \(\text{FG}^{\text{lin}}\) proof.

Proof of the restriction lemma for \(\text{FG}^{\text{lin}}\).
By (a double) induction on the distance between the closer instance of the \([\text{Sync}^{\text{lin}}]\) rule to the root and the root, (and the number of instances of \([\text{Sync}^{\text{lin}}]\) rule which is straight forward), we prove that if there exists a \(\text{FG}^{\text{lin}}\) derivation then there exists a restricted \(\text{FG}^{\text{lin}}\) derivation with the same first and last resolvents.

Base case: An instance of the \([\text{Sync}^{\text{lin}}]\) rule is already at the root.

Induction case: The instance of the rule at the root of the derivation is either an instance of \([\text{Back}^{\text{lin}}]\), \([\text{Axiom}^{\text{lin}}]\), \([\text{Back}^{\text{lin}}]\) or \([\text{Axiom}^{\text{lin}}]\) rules. The \([\text{Back}^{\text{lin}}]\) case is only treated, there others are similar. The derivation is of the form (without loss of generality \(i = 1\)):

\[
\forall \ (\Sigma_1 : \Delta'_1 \rightarrow A_1 \sigma), \ldots, (\Sigma_1 : \Delta'_n \rightarrow A_p \sigma), \\
(\Sigma_2 : \Delta_2 \rightarrow G_2), \ldots, (\Sigma_n : \Delta_n \rightarrow G_n) \\
(\Sigma_1 : \Delta_1 \rightarrow G_1), \ldots, (\Sigma_n : \Delta_n \rightarrow G_n)
\]

with \(C = (H \sigma \leftarrow A_1 \otimes \ldots \otimes A_p) \in \Delta_1, H \sigma = G_1 \) and \(\Delta_1 \setminus \{C\} = \cup_{1 \leq k \leq p} \Delta'_p\). By induction hypothesis, there exists a \(\text{FG}^{\text{lin}}\) derivation \(\forall'\) with an instance of the \([\text{Sync}^{\text{lin}}]\) rule at the root, the same number of instances of the \([\text{Sync}^{\text{lin}}]\) rule and the same first and last resolvents as \(\forall\). A restricted \(\text{FG}^{\text{lin}}\) derivation is obtained by permuting this instance of the \([\text{Sync}^{\text{lin}}]\) rule and the instance of the \([\text{Back}^{\text{lin}}]\) rule and inserting in the signature of every sequents of the descendants of \((\Sigma_1 : \Delta'_1 \rightarrow A_1 \sigma)\) \((\forall k, 1 \leq k \leq p)\) in the derivation \(\forall'\) the constant \(\alpha\) inserted by the instance of the \([\text{Sync}^{\text{lin}}]\) rule.

\[\square\]

Lemma 4 (Ordered Lemma)
If there exists a \(\text{FG}^{\text{lin}}\) proof then there exists a restricted, ordered \(\text{FG}^{\text{lin}}\) proof.

Proof of the order lemma for \(\text{FG}^{\text{lin}}\).
By the restriction lemma, for any \(\text{FG}^{\text{lin}}\) proof there exists a restricted \(\text{FG}^{\text{lin}}\) proof. The existential introduction induces a partial order on the Eigenvariables. The order on the initial instances respects this partial order. Two instances of \([\text{Sync}^{\text{lin}}]\) rule can be permuted if this permutation respects the partial order.

The order of the initial instances can be chosen as the order of introduction of the Eigenvariables. \[\square\]

Lemma 5 (\(\text{FG}^{\text{lin}} \iff \text{FG}^{\text{dir}}\))
There exists a proof of the sequent \((\Sigma_1 : \Delta_1 \rightarrow G_1) \ldots (\Sigma_n : \Delta_n \rightarrow G_n)\) in the system \(\text{FG}^{\text{lin}}\) if and only if there exists a proof of the sequent \((\Sigma_1 : \Delta_1 \rightarrow G_1) \ldots (\Sigma_n : \Delta_n \rightarrow G_n)\) in the system \(\text{FG}^{\text{dir}}\).

Proof of \(\text{FG}^{\text{lin}} \Rightarrow \text{FG}^{\text{dir}}\).
We prove by induction on the number of instances of the \([\text{Sync}^{\text{lin}}]\) rule if there exists a restricted, ordered \(\text{FG}^{\text{lin}}\) derivation of \((\# : \rightarrow G)\) then there exists a \(\text{FG}^{\text{dir}}\) derivation of \((\# : \rightarrow G)\) with the same last resolvent.
Case $l = 0$: The $FG^{lin}$ derivation contains no instance of $[Sync^{lin}]$ rule so this derivation is also a $FG^{dir}$ derivation.

Case $l > 0$: The restricted, ordered $FG^{lin}$ derivation is composed of two parts:

- a derivation $R$ containing only instances of the $[Back^{lin}]$, $[Axiom^{lin}]$, $[Back^{*lin}]$ and $[Axiom^{*lin}]$ rules:

$$
(\Sigma, \alpha : \Delta, C_1 \theta, \ldots, C_l \theta \rightarrow G)
$$

- and a derivation $S$ containing only $l$ instances of $[Sync^{lin}]$ rule with the last applied on

$$
\forall c \exists nc(C_1 \otimes \ldots \otimes C_r) \in \Psi, \theta = \{c \leftarrow \beta, nc \leftarrow \alpha\}, \beta \in \Sigma, \alpha \not\in \Sigma : \quad (5)
$$

$$
(\Sigma, \alpha : \Delta, C_1 \theta, \ldots, C_l \theta \rightarrow G) \\
(\Sigma : \Delta \rightarrow G) \\
\ldots \\
(# : \rightarrow G)
$$

By induction hypothesis $R$ contains the initial instance corresponding to the instance of this $[Sync^{lin}]$. It can be an instance of the $[Back^{*lin}]$ or $[Axiom^{*lin}]$ rules. The $[Back^{*lin}]$ case is only treated, the $[Axiom^{*lin}]$ case is similar. The derivation $R'$ is of the form (without loss of generality $i = 1$):

$$
R' = \left\{ \begin{array}{l}
\forall \nabla \\left( (\Sigma, \alpha : \Delta_1', \ldots, \Delta_l' \rightarrow A_1) \ldots (\Sigma, \alpha : \Delta_n', \ldots, \Delta_l' \rightarrow A_p) \sigma, \\
(\Sigma, \alpha : \Delta_1 \rightarrow G_1), \ldots (\Sigma, \alpha : \Delta_n \rightarrow G_n) \\
(\Sigma, \alpha : \Delta_1 \cup (\bigcup_{1 \leq k \leq n} \Delta_k') \cup (H \circ A_1 \otimes \ldots \otimes A_p) \rightarrow G_1), \\
(\Sigma, \alpha : \Delta_2 \cup \Delta_2' \rightarrow G_2), \ldots, (\Sigma, \alpha : \Delta_n \cup \Delta_n' \rightarrow G_n) \\
\end{array} \right\}
$$

with

$$
C_1 \theta, \ldots, C_{j-1} \theta, C_{j+1} \theta, \ldots, C_{l} \theta = (\psi_{1 \leq k \leq n} \Delta_k') \cup (\psi_{1 \leq k \leq p} \Delta_k''), \\
\bigcup_{1 \leq k \leq p} \Delta_k' = \Delta_1, \\
C_j \theta = H \circ A_1 \otimes \ldots \otimes A_p, \\
H \sigma = G_1.
$$

The derivation $R$ is restricted and ordered then $R'$ contains only instances of the $[Back^{lin}]$, $[Axiom^{lin}]$, $[Back^{*lin}]$ and $[Axiom^{*lin}]$ rules (on non initial instances) then the derivation $R'$ is also a $FG^{dir}$ derivation.
Let \( \mathcal{D}'' \) be the same derivation as \( \mathcal{D}' \) without the instances \( C_k \theta, 1 \leq k \leq r \) and \( \alpha \). Since the derivation \( \mathcal{R} \) is ordered the derivation \( \mathcal{D}'' \) is still a \( FG^{lin} \) derivation. So by induction hypothesis on the \( FG^{lin} \) derivation below formed from \( S \) without the last inference and \( \mathcal{D}'' \):

\[
\frac{\left( \Sigma : \Delta \rightarrow G \right)}{\Sigma : \Delta \rightarrow G}
\]

there is a \( FG^{dir} \) derivation \( \mathcal{D}''' \):

\[
\left( \Sigma : \Delta \rightarrow G \right)
\]

So the derivation:

\[
\mathcal{D}''' \begin{cases}
\mathcal{R}' \begin{cases}
\frac{\left( \Sigma, \alpha : \Delta_1' \cup \Delta_1'' \rightarrow A_1 \sigma \right), \ldots, \left( \Sigma, \alpha : \Delta_p' \cup \Delta_p'' \rightarrow A_p \sigma \right),}{\left( \Sigma, \alpha : \Delta_1' \cup \Delta_1'' \rightarrow A_1 \sigma \right), \ldots, \left( \Sigma, \alpha : \Delta_p' \cup \Delta_p'' \rightarrow A_p \sigma \right)}
\end{cases}
\end{cases}
\]

with \( \{ C_1 \theta, \ldots, C_{j-1} \theta, C_{j+1} \theta, \ldots, C_r \theta \} = (\cup_{1 \leq k \leq n} \Delta_k''') \cup (\cup_{1 \leq k \leq p} \Delta_k'') \) and \( \cup_{1 \leq k \leq p} \Delta_k' = \Delta_1 \) in a \( FG^{dir} \) derivation.

\( \square \)

**Proof of** \( FG^{dir} \Rightarrow FG^{lin} \).

We prove by induction on the number of instances of the \( [\text{Sync}^+_{dir}] \) or \( [\text{Sync}1_{dir}] \) rules if there exists a \( FG^{dir} \) derivation of \( \left( \# : \rightarrow G \right) \) then there exists a \( FG^{lin} \) derivation of \( \left( \# : \rightarrow G \right) \) with the same last resolvent.

**Case** \( l = 0 \): The \( FG^{dir} \) derivation contains no instance of \( [\text{Sync}^+_{dir}] \) nor \( [\text{Sync}1_{dir}] \) rule so this derivation is also a \( FG^{lin} \) derivation.

**Case** \( l > 0 \): The \( FG^{dir} \) derivation is composed of two parts:

- a derivation \( \mathcal{R} \) containing only instances of the \( [\text{Back}^+_{dir}], [\text{Axiom}^+_{dir}], [\text{Back}^?_{dir}] \) and \( [\text{Axiom} \ ?_{dir}] \) rules;
- and a derivation \( \mathcal{D} \) containing only \( l \) instances of \( [\text{Sync}^+_{dir}] \) or \( [\text{Sync}1_{dir}] \) rule;
and a link inference of \([\text{Sync}^{+\text{dir}}]\) or \([\text{Sync}^{\text{dir}}]\) between \(R\) and \(D\) applied on \(\forall c \exists n c(C_1 \otimes \ldots \otimes C_r) \in \Psi\) with \(\theta = \{c \leftarrow \beta, nc \leftarrow \alpha\}, \beta \in \Sigma, \alpha \notin \Sigma\).

The \([\text{Sync}^{+\text{dir}}]\) case is only treated, the \([\text{Sync}^{\text{dir}}]\) case is similar.

The derivation \(D\) is of the form (without loss of generality \(i = 1\)):

\[
\begin{align*}
\vdash & \quad \left\{ (\Sigma, \alpha : \Delta'^{i} \rightarrow A_1 \sigma), \ldots, (\Sigma, \alpha : \Delta'^{i} \rightarrow A_p \sigma), \\
& \quad (\Sigma, \alpha : \Delta^{i}_2 \rightarrow G_2), \ldots, (\Sigma, \alpha : \Delta^{i}_n \rightarrow G_n) \right. \\
\end{align*}
\]

with \(\{C_1 \theta, \ldots, C_j \theta, C_{j+1} \theta, \ldots, C_n \theta\} = (\bigcup_{1 \leq k \leq p} \Delta'_{k}) \cup (\bigcup_{1 \leq k \leq p} \Delta''_{k}), \bigcup_{1 \leq k \leq p} \Delta'_{k} = \Delta_{1}, \Delta_{j} \theta = H \circ A_{1} \otimes \ldots \otimes A_{p}, H \delta = G_{1}\). The derivation \(R\) contains only instances of \([\text{Back}^{+\text{dir}}]\), \([\text{Axiom}^{\text{dir}}]\), \([\text{Back}^{-\text{dir}}]\) and \([\text{Axiom}^{-\text{dir}}]\) so it is also a \(FG^{\text{lin}}\) derivation. By induction hypothesis on the \(FG^{\text{dir}}\) derivation \(D'\) containing \(l - 1\) instances of \([\text{Sync}^{+\text{dir}}]\) or \([\text{Sync}^{-\text{dir}}]\) rules there exists a \(FG^{\text{lin}}\) derivation:

\[
\begin{align*}
\vdash & \quad \left\{ (\Sigma : \Delta^{1} \rightarrow G_1), \ldots, (\Sigma : \Delta^{n} \rightarrow G_n) \right. \\
& \quad \left. \vdash (\Sigma : \Delta \rightarrow G) \right\}
\end{align*}
\]

Let \(D''\) be the same derivation as \(D''\) but for all \((\Sigma' : \Delta' \rightarrow G')\) of \(D'\):

- if \(G'\) is an ancestor of \(G_k, 1 < k \leq n\), then \(\Delta''_{k}\) is added to \(\Delta'\);
- if \(G'\) is an ancestor of \(G_1\) then \((\bigcup_{1 \leq k \leq p} \Delta''_{k}) \cup \{C_j \theta\}\) is added to \(\Delta'\);
- \(\alpha\) is added to \(\Sigma\).

Then the derivation:

\[
\begin{align*}
\vdash & \quad \left\{ (\Sigma, \alpha : \Delta'^{i} \rightarrow A_1 \sigma), \ldots, (\Sigma, \alpha : \Delta'^{i} \rightarrow A_p \sigma), \\
& \quad (\Sigma, \alpha : \Delta^{i}_2 \rightarrow G_2), \ldots, (\Sigma, \alpha : \Delta^{i}_n \rightarrow G_n) \right. \\
\end{align*}
\]

\[
\begin{align*}
\vdash & \quad \left\{ (\Sigma, \alpha : \Delta^{i} \cup (\bigcup_{1 \leq k \leq p} \Delta''_{k}) \cup \{C_j \theta\} \rightarrow G_1), \\
& \quad (\Sigma, \alpha : \Delta^{i}_2 \rightarrow G_2), \ldots, (\Sigma, \alpha : \Delta^{i}_n \rightarrow G_n) \right. \\
\end{align*}
\]

is a \(FG^{\text{lin}}\) derivation. \(\Box\)
We have to mention here that this system provides a goal directed approach for the recognition of an element of a tree language. This kind of operation is not practically possible using the initial grammar definition which gives a method to produce elements of the language but does not give any strategy to recognize an element.

The introduction of unification and variable renaming in the system $\text{FG}^{\text{dir}}$ would also ensure the generation of the element of the language. This has to be formally proved thanks to a lifting lemma (as it is done for the SLD resolution of Prolog [15]). As a consequence, the generation is available in the Prolog implementation of our system described in the next section.

Concerning the decidability of our approach, it is clear that problems encountered with DCG’s appear here (mainly due to left recursions and empty transitions). Since Prolog is the underlying framework, the problems related to its depth left first strategy also occur. Thus, termination of our method depends on this search strategy.

5 Implementation Issue

This section briefly describes how the implementation can be achieved from the previous inference system. At this time, a library is able, taking a TSG (resp. a PG) as input, to provide, as output, a predicate $\text{phrase}_{\text{TSG}}$ (resp. $\text{phrase}_{\text{PG}}$) which can recognize or generate a term for the defined language from the axiom. The implementation of the method described along this paper can be divided in two parts: the translation of the grammar into linear logic formulas (described in Section 5.1) and the implementation of the proof system $\text{FG}^{\text{dir}}$ in Prolog. The following presentation of this implementation is related to Example 1.

5.1 Management of signature and linear context

The extension of the signature (i.e. new symbols introduced by existential quantifiers of synchronization) is inspired by the management of essentially universal quantifier (quantifier $\pi$ in the body of clauses) of $\lambda$Prolog [3]: only the cardinality of this extension is needed.

The management of linear clauses is meta-programmed by a linear program continuation LPC (i.e. a set containing the remaining linear clauses introduced by instances of $\text{Sync}^{+\text{dir}}$ or $\text{Sync}^{1}$ which have to be used later). This technique is inspired by the management of intuitionistic implication of $\lambda$Prolog [12, 1].

5.2 Transformation Function

Back to Example 1, the transformation is illustrated here by the implementation of the clause $F_0$ which corresponds to the free production $R_0$.

\begin{verbatim}
i(C, f(X, Y), SigmaIn, SigmaOut, DeltaIn, DeltaOut) :-
    x(C, X, SigmaIn, SigmaInter, DeltaIn, DeltaInter),
    y(C, Y, SigmaInter, SigmaOut, DeltaInter, DeltaOut).
\end{verbatim}

SigmaIn, SigmaInter and SigmaOut represent the cardinality of the extension of the signature. DeltaIn, DeltaInter and DeltaOut represent the LPC.

5.3 Implementation of the Proof System $\text{FG}^{\text{dir}}$ in Prolog

The inference rules $\text{[Back !]}$ and $\text{[Axiom !]}$ are handled by the reduction mechanism of Prolog.
Computing over TSG and PG

The inference rules \([\text{Back}!]\) and \([\text{Axiom}!]\) are implemented by a meta-programming of Prolog by the predicate \text{linear} over the linear program continuation.

\[
\text{linear}(\text{PredicateSymbol}, C, \text{Term}, \text{SigmaIn}, \text{SigmaOut}, [C_j|C_j+1_{-1}|C_m], C_1_{-1}C_j, \Delta\text{Out}) :- \\
C_j = (\text{Head} :- \text{Body}), \\
\text{Head} =.. [\text{PredicateSymbol}, C, \text{Term}, \text{SigmaIn}, \text{SigmaOut}, C_1_{-1}C_j, \Delta\text{Out}], \\
\text{append}(C_1_{-1}C_j, C_j+1_{-1}C_m, C_1_{-1}C_m), \text{call}(\text{Body}).
\]

The linear clause \(C_j\) is meta-interpreted and then discarded from the linear program continuation: \(C_1_{-1}C_m[C_j[C_j+1_{-1}|C_m] \cup C_j+1_{-1}C_m\) (see append predicate).

A recursive prolog clause is added for the program continuation traversal.

The inference rules \([\text{Sync} + \text{dir}]\) and \([\text{Sync} 1]\) are implemented in a similar way as respectively \([\text{Back}!]\) and \([\text{Axiom}!]\) but with an increase of the linear program continuation by the linear clauses introduced by the synchronization.

The prolog clause below corresponds to the implementation of \(x\) for the translation of the pack of production \(\{X \Rightarrow s(X), Y \Rightarrow s(s(Y))\}:

\[
(\forall c \exists nc((\forall X(x(s(X),c) \circ -x(X,nc))) \otimes (\forall Y(y(s(s(Y)),c) \circ -y(Y,nc))))).
\]

\[
x(C, s(X), \text{SigmaIn}, \text{SigmaOut}, \text{DeltaIn}, \text{DeltaOut}) :- \\
NC is \text{SigmaIn} + 1, \\
Cy = y(C, s(s(Y)), SI, SO, DI, DO) :- y(NC, Y, SI, SO, DI, DO), \\
x(NC, X, NC, \text{SigmaOut}, [Cy|\text{DeltaIn}], \text{DeltaOut}).
\]

The linear clause \(\sigma_5(Y \Rightarrow s(s(Y)), 0, 1) = (\forall Y(y(s(s(Y)), 0) \circ -y(Y, 1)))\) is implemented by \(Cy\) added to the linear program continuation.

### 5.4 Reduction strategy

Due to the leftmost selection rule and the depth-first search strategy of Prolog the reduction strategy is a left-outermost strategy as shown on the trace below for the goal \(\text{phrase}_TSG(f(s(z), s(s(z))), i)\) (the clause \(y(0, s(s(Y)), SI, SO, DI, DO) :- y(1, Y, SI, SO, DI, DO))\) is denoted \(Cy\). The first column is the (simplified†† prolog trace, the second one contains the linear program continuation and the third one the term already recognized.

---

† There is of course a symmetrical clause for \(y\).

†† In the real system, variables are denoted as in Prolog (i.e. of the form _1234). For the sake of readability, explicit names have been introduced and meta-programming arguments have been discarded.
phrase_TSG(f(s(z),s(s(z))),i) LPC
i(0,f(s(z),s(s(z))))
\{\}
\{(I,0)\}
f((X, 0), (Y, 0))
x(0,s(z)),y(0,s(s(z)))
\{Cy\}
f(s((X, 0)), (Y, 0))
x(1,z),y(0,s(s(z)))
\{\}
f(s(z), (Y, 0))
y(0,s(s(z)))
\{Cy\}
f(s((X, 0)), (Y, 0))
y(1,z)
\{\}
f(s(z), s(s((Y, 0))))
\{\}
f(s(z), s(s(z)))

5.5 Practical Use of the System

We can now illustrate the complete use of our system on PG's Example 2. The user has only to write as input the following prolog clause:

```
pg_with_trace : -
    name(FileName, "pg_with_trace.pl"),
    open(FileName, write, Stream),
    sigma_pg(pg(f, [f,g], [
        [(f(0) -> z), (f(s(n)) -> g(s(n))*f(n))],
        [(g(0) -> z), (g(s(n)) -> s(g(n)))]
    ]), Stream),
    close(Stream).
```

in order to provide the description of the PG thanks to the predicate `sigma_pg`. The execution of this clause produces file `pg_with_trace.pl`, containing a predicate `phrase_pg`, which can be loaded and used for grammar recognition as:

```
? - phrase_pg(s(z)*z,f,[s(0)]).
g*f
s(g)*f
s(z)*f
s(z)*z
```

yes

This execution shows that `s(z)*z` belongs to the PG and how it can be generated from the PG.

The source code of this implementation is available at:


6 Conclusion

In this paper, we describe an implementation scheme for particular types of tree language including specific control features. The main idea is to provide an uniform framework in order to compute over tree grammars. As it has been done for word grammars with DCG [2], we propose a transformation method that allows us to get a set of Prolog Horn clauses from a grammatical definition of a tree language. This method consists in translating grammar derivation into proof search using a sequent calculus proof system.
Computing over TSG and PG

Based on linear logic. By successive refinements, we get a goal directed procedure implemented in Prolog. Since such tree languages appear as powerful tools for the schematization of sets of terms to represent solutions of symbolic computation problems, it seemed necessary to define a method to use these representations. Moreover, this approach allows us to embed various formalism into a unique framework, were computations can include Regular Languages, TSG and PG.

Acknowledgements

We would like to thank Miki Hermann, Sébastien Limet and Pierre Réty for fruitful discussions on this work.

References


