More Flexible Term Schematisations via Extended Primal Grammars

Vincent Aravantinos, Ricardo Caferra and Nicolas Peltier
CNRS, INPG
LIG, 46 avenue Félix Viallet
38031 Grenoble Cedex
France
Vincent.Aravantinos@imag.fr, Ricardo.Caferra@imag.fr,
Nicolas.Peltier@imag.fr

Abstract

We propose an extension of primal grammars (Hermann & Galbavý 1997). Primal grammars are term grammars with a high expressive power and good computational properties. The extended grammars have exactly the same properties but are more modular, more concise, and easier to use, as shown by some examples. An algorithm transforming any extended primal grammar into an equivalent primal grammar is provided.

1 Introduction

Schematising sets of objects by abstracting their general structure is a very important, and frequently necessary, task in almost all intelligent activities (e.g. deductive and inductive reasoning, learning, explanation,...). In automated deduction, it is natural to try to schematise terms or formulas yield by theorem provers. Despite the obvious advantages of such capability to improve existing systems (to prevent divergence, extract lemmata, enrich the input languages...) relatively few works have addressed the problem (Chen, Hsiang, & Kong 1990; Salzer 1992; Hermann 1992). This is mainly due to the fact that specifying the structure of (possibly infinite) sets of objects with *mechanizable* formalisms is a very ambitious aim.

Term schematisations are formalisms that allow one to express infinite sets of structurally similar terms in a finite way[1]. These formalisms have been proposed since the 90's, essentially to address divergence problems in symbolic calculi (such as e.g. Knuth-Bendix completion or SLD resolution). Several schematisation languages have been defined with various expressive powers: recurrent terms (Chen, Hsiang, & Kong 1990), terms with integer exponents (Comon 1995), R-terms (Salzer 1992), primal grammars (Hermann & Galbavý 1997). (Hermann 1994) provides a detailed survey of term schematisation. Each of those formalisms permits the construction of term schemas by applying iteratively a certain inductive context over a base term. For instance, the terms sequence $a, f(a), f(f(a)), \ldots, f^n(a)$ can be obtained by iterating n times the context $f(\ldots)$ from the base term a. This is usually denoted by $f(\diamond)^n.a$ (the symbol $\diamond$ denoting a "hole"). It must be observed that n denotes an arithmetic variable, not a fixed natural number. This makes the whole difference with usual first order terms as $f(\diamond)^n.a$ is not a classical term but a schema of terms.

The language of primal grammars is the most expressive formalism for term schematisation. It allows one to iterate arbitrary contexts along several positions, and these contexts may depend on the iteration rank in the sequence, making possible e.g. to denote a list of the form $[1, 2, 3, 4, \ldots, n]$ or $[n, n-1, \ldots, 1]$. This feature is particularly important in the context of formula schematisation. For instance, it allows to express a clause of the form $\bigvee_{i=1}^{n} p_i$ or a set of clauses of the form $\{p_i \vee q_{i+1} \mid i \in [1..n-1]\}$, which is impossible using less expressive formalisms (see (Aravantinos 2007)).

The first order theory of primal grammars is known to be decidable (Peltier 2004). However, this formalism is very cumbersome and difficult to use. This is understandable knowing it has been developed with the goal of being as expressive as possible without loosing the decidability of unification. This imposes conditions on the contexts that are rather strong and not very intuitive (we shall illustrate this point in more details after introducing the necessary definitions: see Section 3 and Examples 2 and 3 on page 3). In this paper, we show that it is possible to relax significantly some of these conditions while preserving the good computational properties of the formalism. More precisely, we introduce a notion of **extended primal grammar**, in which the conditions on the allowed contexts are much weaker. We provide a transformation algorithm reducing any extended primal grammar to a standard one. The obtained grammar is equivalent to the original one, but the latter is in general much simpler, and defined on a less complex signature, which makes it easier to handle for the user.

Why primal grammars ?

We start with *genetic* descriptions of objects and we

[1] This may remind the reader of tree automata (Comon *et al.* 2005) that show many similarities with term schematisations. But the denoted languages are very different.

are looking for their *structural* descriptions, e.g.

$$\left.\begin{array}{r}(\forall x)p(x,a)\\ (\forall x,y)p(f(x),y)\Rightarrow p(x,f(y))\end{array}\right\}\text{: start point}$$

$$(\forall x)p(x,f^n(a))\text{: target representation}$$

Between the start point and the target representation, **arithmetic** appears. How to deal with it ? The most natural way is to enrich term grammars with (decidable fragments of) arithmetic. Primal grammars seem to be a reasonable, though "heavy", solution. The next natural step is to simplify the formalism in order to make it easier to use. Our work is a first technical (partial) solution to this problem.

Designing algorithms allowing to represent (possibly) infinite sets of structurally similar terms can clearly *qualitatively* improve the capabilities of theorem provers and can be also useful in other AI tools. This allows one to handle some parts of the input formula *directly in the unification algorithm* which makes the task of the inference system easier and yields proofs that are shorter and simpler (in particular the proof procedure should terminate more often); this is also useful in *structuring* proofs. It is therefore worth taking steps in this direction.

2 Primal grammars

Our definition is essentially the same (though slightly simpler) as in (Hermann & Galbavý 1997) except that we use a multisorted signature. We refer to (Hermann & Galbavý 1997) for details, additional explanations and examples.

2.1 Syntax

A *signature* is a pair (S,Σ) where S is a set of *sorts* and Σ a set of *function symbols*. Each function symbol is associated to a *profile* of the form $\mathsf{s_1}\times\cdots\times\mathsf{s_n}\rightarrow\mathsf{s_{n+1}}$ where $\mathsf{s_1},\ldots,\mathsf{s_{n+1}}$ are sorts.

Definition 1 (Primal signature).
A primal signature *is a signature*

- *containing the* int *sort;*
- *containing the usual function symbols of Presburger arithmetic:*

$$\begin{array}{rr}\mathbf{0}: & \longrightarrow \mathsf{int}\\ \mathbf{s}: & \mathsf{int}\longrightarrow\mathsf{int}\\ +: & \mathsf{int}\times\mathsf{int}\longrightarrow\mathsf{int}\end{array}$$

- *and such that the profiles of the other function symbols are of the form:*

$$\mathsf{int}^n\times\mathsf{s_1}\times\cdots\times\mathsf{s_k}\longrightarrow\mathsf{s_{k+1}}$$

where $n\geq 0$ *and* $\forall i\in[1..k+1], s_i\neq\mathsf{int}$. *If* $n=0$ *then the symbol is called a* constructor *symbol, otherwise it is called a* defined *symbol.*
Defined symbols will be denoted by $\hat{f},\hat{g},\ldots$ *n is called the* counter arity *of the symbol, denoted by* $\mathcal{A}r_c(\hat{f})$.

Let (S,Σ) be a signature and let $\mathcal{X}$ be a countable set of variables, together with a function mapping each variable x in $\mathcal{X}$ to a sort in S. The set of *terms of sort* s is inductively defined as follows:

- each variable $x\in\mathcal{X}$ of sort s is also a term of sort s;
- for each $f:\mathsf{s_1}\times\cdots\times\mathsf{s_n}\rightarrow\mathsf{s}\in\Sigma$ and for all terms $t_1,\ldots,t_n$ of respective sorts $\mathsf{s_1},\ldots,\mathsf{s_n}$, $f(t_1,\ldots,t_n)$ is a term of sort s.

The whole set of terms is denoted by $\mathcal{A}(S,\Sigma,\mathcal{X})$

A *position* is a finite sequence of natural numbers. ϵ denotes the empty position, $p.q$ denotes the concatenation of p and q and $\leq$ denotes the prefix ordering. The *set of positions in a term* t is defined as follows: ϵ is a position in t and if $t=f(t_1,\ldots,t_n)$, $i\in[1..n]$ and p is a position in t_i, then $i.p$ is a position in t. If A is a set of positions in t, then $t[s]_A$ denotes the term obtained by replacing in t the subterms occurring at any position $p\in A$ by s.

Definition 2 (Primal term algebra).
A primal term *is a term built on a primal signature whose sort is different from* int.

Terms of sort int are called *counter expressions*. Each counter expression will be written in bold face from now on. Variables of sort int are called *counter variables*. The other variables are called *standard variables*.

$\mathcal{CV}ar(t)$ will denote the set of counter variables occurring in the primal term t.

Primal terms not containing defined symbols are called *standard terms*.

We call *redex*[2] *terms* the terms whose root is a defined symbol. They will be written as follows:

$$\hat{f}(\mathbf{c_1},\ldots,\mathbf{c_n};t_1,\ldots,t_k)$$

where:

- $\mathbf{c_1},\ldots,\mathbf{c_n}$ are counter expressions;
- $t_1,\ldots,t_k$ are primal terms;
- ";" is used to distinguish counter arguments from standard ones.

Definition 3 (Regular term).
A primal term t *is called* regular *if for any redex* $\hat{f}(\vec{\mathbf{c}};\vec{u})$ *occurring in* t, $\vec{u}$ *is a vector of standard terms (i.e. without redexes).*

2.2 Semantics

The semantics of defined symbols are specified using a convergent system of rewrite rules, which reduces all ground terms to *standard* terms. This system satisfies some additional properties in order to ensure that the unification problem between primal terms is decidable.

[2]Because they will be *reduced* by a term rewrite system.

Presburger Rewrite Systems

- We assume given a precedence $\prec$ on defined symbols.
- Given two vectors $\vec{t} = (t_1, \ldots, t_n)$ and $\vec{s} = (s_1, \ldots, s_m)$, we say that $\vec{s}$ is a *subvector* of $\vec{t}$ iff $m \leq n$ and there exists an increasing function $\eta : [1..m] \to [1..n]$ *s.t.* $s_i = t_{\eta(i)}$.

The *approximation* of a primal term $\hat{f}(\vec{\mathbf{c}}; \vec{x})$ (denoted by $\mathcal{A}px(\hat{f}(\vec{\mathbf{c}}; \vec{x}))$) is defined as the set:

$$\mathcal{A}px(\hat{f}(\vec{\mathbf{c}}; \vec{x})) = \left\{ \hat{g}(\vec{\mathbf{c}'}; \vec{x'}) \;\middle|\; \begin{array}{l} \hat{f} \succ \hat{g} \\ \vec{\mathbf{c}'} \text{ is a subvector of } \vec{\mathbf{c}} \\ \vec{x'} \text{ is a subvector of } \vec{x} \end{array} \right\}$$

Definition 4.

A rewrite system is called Presburger *if it contains the usual rules of Presburger arithmetics, and for each defined symbol $\hat{f}$ it contains:*

- *a unique* base rule *of the form:*

$$\hat{f}(\mathbf{0}, \vec{\mathbf{c}}; \vec{x}) \to r_1^{\hat{f}}$$

- *a unique* inductive rule *which is either of the form:*
  $\hat{f}(\mathbf{s(i)}, \vec{\mathbf{c}}; \vec{x}) \to r_2^{\hat{f}}[\hat{f}(\mathbf{i}, \vec{\mathbf{c}}; \vec{x})]_A$
  or
  $\hat{f}(\mathbf{s(i)}, \vec{\mathbf{c}}; \vec{x})$
  $\quad \to r_2^{\hat{f}}[\hat{f}(\mathbf{i}, \mathbf{c_1}, \ldots, \mathbf{c_{k-1}}, \mathbf{s(c_k)}, \mathbf{c_{k+1}}, \ldots, \mathbf{c_n}; \vec{x})]_A$

and such that:

1. *A is a finite, non-empty, subset of parallel non empty positions in $r_2^{\hat{f}}$;*
2. *$r_2^{\hat{f}}[\hat{f}(\mathbf{i}, \mathbf{c_1}, \ldots, \mathbf{c_{k-1}}, \mathbf{s(c_k)}, \mathbf{c_{k+1}}, \ldots, \mathbf{c_k}; \vec{x})]_A$ and $r_2^{\hat{f}}[\hat{f}(\mathbf{i}, \vec{\mathbf{c}}; \vec{x})]_A$ are regular terms;*
3. *all $r_1^{\hat{f}}$ and $r_2^{\hat{f}}$ redexes are in $\mathcal{A}px(\hat{f}(\mathbf{i}, \vec{\mathbf{c}}; \vec{x}))$;*

$\mathbf{i}$ is called *principal counter* and, in the inductive rule, $\mathbf{c_k}$ the *auxiliary counter*.

Property 1 (Convergence).

Any Presburger system $\mathcal{R}$ is convergent,
i.e. each term t has a unique normal form denoted by $t{\downarrow_{\mathcal{R}}}$ (or simply by $t{\downarrow}$ if no confusion is possible).

Proof. See (Hermann & Galbavý 1997). □

Primal Grammars

Definition 5.

A primal grammar *is a 3-tuple $(\Sigma, \mathcal{R}, t^\star)$ where:*

- *Σ is a primal signature,*
- *$\mathcal{R}$ is a Presburger rewrite system,*
- *$t^\star$ is a primal term built on Σ, called* start symbol.

Example 1.

The following is a primal grammar:

- a primal signature:

$$\begin{array}{ll} \hat{s} : \mathsf{int} & \longrightarrow \mathsf{t} \\ s : \mathsf{t} & \longrightarrow \mathsf{t} \\ 0 : & \longrightarrow \mathsf{t} \end{array}$$

- the following Presburger system:

$$\hat{s}(\mathbf{0}) \to 0$$
$$\hat{s}(\mathbf{s(i)}) \to s(\hat{s}(\mathbf{i}))$$

- and the start symbol $\hat{s}(\mathbf{n})$.

Intuitively, this example shows how to encode the set of natural numbers with primal grammars. $\hat{s}(\mathbf{n})$ is $s^n(0)$ where n is the value of $\mathbf{n}$, while $\hat{s}(\mathbf{s(s(0))})$ is $s^2(0)$, i.e. the standard term $s(s(0))$ (notice the different typesetting between counters and standard terms).

As this example is often used, $\hat{s}$ hereby denotes the above primal grammar.

3 Motivations for the extension

The requirements in Definition 4 are quite restrictive, in particular Condition 3 which imposes rigid conditions on the redexes occurring in the contexts.

Example 2.

Assume that one wants to denote the list of the natural numbers $[m...n+m]$, in reverse order. The most natural way to proceed would be to define the following rules:

$$\hat{f}(\mathbf{0}, \mathbf{n}) \to [\hat{s}(\mathbf{n})]$$
$$\hat{f}(\mathbf{s(m)}, \mathbf{n}) \to [\hat{s}(\mathbf{n + m + 1}) | \hat{f}(\mathbf{m}, \mathbf{n})]$$

where $\hat{s} \prec \hat{f}$. But the obtained rule is **not** a Presburger rewrite system, because the redex $\hat{s}(\mathbf{n + m + 1})$ is not in the approximation of $\hat{f}(\mathbf{m}, \mathbf{n})$ ($\mathbf{n + m}$ should be $\mathbf{n}$ or $\mathbf{m}$, or the list of arguments should be empty).

Fortunately, in this particular case, it is possible to denote the same list using another function symbol of arity 2:

$$\hat{f}(\mathbf{0}, \mathbf{n}) \to [\hat{s}(\mathbf{n})]$$
$$\hat{f}(\mathbf{s(m)}, \mathbf{n}) \to [\hat{s'}(\mathbf{m}, \mathbf{n}) | \hat{f}(\mathbf{m}, \mathbf{n})]$$

where:

$$\hat{s'}(\mathbf{0}, \mathbf{n}) \to s(\hat{s}(\mathbf{n}))$$
$$\hat{s'}(\mathbf{s(m)}, \mathbf{n}) \to s(\hat{s'}(\mathbf{m}, \mathbf{n}))$$

Here $\hat{s'}(\mathbf{m}, \mathbf{n})$ encodes $\hat{s}(\mathbf{m + n + 1})$.

However, this process is not very natural and the obtained system is bigger and more difficult to understand, due to the presence of additional parameters and defined symbols. Note that a similar behaviour can be observed with auxiliary counters.

We give another example that illustrates a similar, but slightly different aspect.

Example 3.

Assume that we want to denote the list of the n first *even* natural numbers. We construct the following schematisation:

$$\hat{f}(\mathbf{0}, \mathbf{m}) \to [\,]$$
$$\hat{f}(\mathbf{s(n)}, \mathbf{m}) \to [\hat{s}(\mathbf{m + m}) | \hat{f}(\mathbf{n}, \mathbf{s(m)})]$$

Then $\hat{f}(\mathbf{n}, \mathbf{0})$ gives the desired result. Unfortunately the above system is not a Presburger rewrite system due to the fact that $\hat{s}(\mathbf{m + m})$ is not in the approximation of $\hat{f}(\mathbf{n}, \mathbf{m})$.

In order to transform this system into a Presburger rewrite system, we have to replace $\hat{s}(\mathbf{m + m})$ by a redex of the form $\hat{g}(\mathbf{m})$. Obviously, this is possible only if the

context corresponding to $\hat{s}$ is "unfolded", i.e. if one iteration of $\hat{g}$ corresponds to two iterations of $\hat{s}$. More precisely, we get:

$$\hat{g}(\mathbf{s}(\mathbf{m})) \to s(s(\hat{g}(\mathbf{m})))$$
$$\hat{g}(\mathbf{0}) \to 0$$

At this point, a very natural issue arises: **is it possible to *automatize* the above transformations ?** The goal is to provide an algorithm that transforms the systems written above into Presburger rewrite systems, instead of assigning this task to the user.

In case the answer is positive, **what are the conditions** on the redexes that guarantee that this transformation is sound (i.e. yields an equivalent Presburger rewrite system) ? Allowing arbitrary redexes will not be satisfactory, because the good properties of primal grammars will be lost. For instance, let us consider the system:

$$\hat{f}(\mathbf{s}(\mathbf{n}), \mathbf{m}, \mathbf{l}) \to s(\hat{f}(\mathbf{n}, \mathbf{m}+\mathbf{l}, \mathbf{l}))$$
$$\hat{f}(\mathbf{0}, \mathbf{m}, \mathbf{l}) \to \hat{s}(\mathbf{m})$$

The reader can easily check that $\hat{f}(\mathbf{n}, \mathbf{m}, \mathbf{l})$ denotes the term $\hat{s}(\mathbf{m} + (\mathbf{l} \times \mathbf{n}) + \mathbf{n})$. Therefore, multiplication can be easily encoded. Since Peano arithmetic is not decidable, this immediately implies that this system *cannot* be transformed into an equivalent Presburger rewrite system. Consequently, additional conditions are required.

Answering the above questions is the subject of the present paper. Our results significantly extend the scope of primal grammars because some sets of rules that do not satisfy the condition of Definition 4 on the preceding page can now be automatically compiled into a Presburger rewrite system. This is also very useful if the system is generated automatically because in this case, it is difficult to ensure that the conditions of Definition 4 on the previous page are satisfied.

4 Extended Primal Grammars

We now introduce our new definition. It can be viewed as a higher level term schematisation language. We show evidence of this easiness and prove the equivalence with standard primal grammars.

The basic idea is to replace the notion of approximation by a more "high level" and refined notion (and thus more sophisticated): the *potential counters*. Once this is done, Presburger rewrite systems can be extended in a very natural way.

4.1 Potentially Principal Counter

A counter variable is a potentially principal counter if it only occurs in *first parameters* of redex subterms — or does not occur at all. Formally:

Definition 6.

Let t be a primal term. We denote by $\mathcal{CX}$ the set of counter variables.

The set of potentially principal counters *of t, denoted by $\mathcal{PPC}(t)$, is inductively defined in the following way:*

$$\mathcal{PPC}(\hat{f}(\mathbf{c_0}, \ldots, \mathbf{c_k}; \ldots)) \stackrel{def}{=} \mathcal{CX} \setminus \bigcup_{i \neq 0} \mathcal{V}ar(\mathbf{c_i})$$
$$\mathcal{PPC}(f(u_1, \ldots, u_k)) \stackrel{def}{=} \bigcap_i \mathcal{PPC}(u_i) \text{ if } \mathcal{A}r(f) > 0$$
$$\mathcal{PPC}(f) \stackrel{def}{=} \mathcal{CX} \text{ if } \mathcal{A}r(f) = 0$$

Example 4.

Consider the following signature:

$$\hat{f} : \mathsf{int} \times \mathsf{int} \times \mathsf{t} \longrightarrow \mathsf{t}$$
$$\hat{g} : \mathsf{int} \times \mathsf{int} \longrightarrow \mathsf{t}$$
$$h : \mathsf{t} \times \mathsf{t} \longrightarrow \mathsf{t}$$
$$a : \longrightarrow \mathsf{t}$$

x denotes a variable of type t.

Then *(counters that appear in other parameters than the first are marked with a frame)*:

- $h(a, x)$: *any counter variable is a potentially principal counter*;
- $\hat{f}(\mathbf{c_1}, \boxed{\mathbf{c_2}}; x)$ and $h(h(a, x), \hat{f}(\mathbf{c_1}, \boxed{\mathbf{c_2}}; x))$ both have *any counter other than $\mathbf{c_2}$ as a potentially principal counter*;
- $h(\hat{g}(\mathbf{c_1}, \boxed{\mathbf{c_2}}), \hat{g}(\mathbf{c_3}, \boxed{\mathbf{c_4}}))$ has any variable other than $\mathbf{c_2}$ and $\mathbf{c_4}$ as a potentially principal counter;
- $h(\hat{g}(\mathbf{c_1}, \boxed{\mathbf{c_2}}), \hat{g}(\mathbf{c_2}, \boxed{\mathbf{c_2}}))$ has any variable other than $\mathbf{c}_2$ as a potentially principal counter *(even though $\mathbf{c}_2$ also appears as a first parameter)*.

4.2 Potentially Auxiliary Counter

A counter variable is a potentially auxiliary counter if it occurs in *at most* one parameter of a redex subterm. Formally:

Definition 7.

The set of potentially auxiliary counters *of a primal term t denoted by $\mathcal{PAC}(t)$, is inductively defined as follows:*

$$\mathcal{PAC}(\hat{f}(\mathbf{c_0}, \ldots, \mathbf{c_k}; \ldots)) \stackrel{def}{=} \mathcal{CX} \setminus \bigcup_{i \neq j} (\mathcal{V}ar(\mathbf{c_i}) \cap \mathcal{V}ar(\mathbf{c_j}))$$
$$\mathcal{PAC}(f(u_1, \ldots, u_k)) \stackrel{def}{=} \bigcap_i \mathcal{PAC}(u_i) \text{ if } \mathcal{A}r(f) > 0$$
$$\mathcal{PAC}(f) \stackrel{def}{=} \mathcal{CX} \text{ if } \mathcal{A}r(f) = 0$$

Example 5.

With the signature of Example 4,

- $h(a, x)$: any counter variable is a potentially auxiliary counter;
- $\hat{f}(\mathbf{c_1}, \mathbf{c_2})$ and $t = h(a, \hat{f}(\mathbf{c_1}, \mathbf{c_2}))$ both have $\mathbf{c_1}$ and $\mathbf{c_2}$ as potentially auxiliary counters;
- $t = \hat{f}(\mathbf{c_1} + \boxed{\mathbf{c_2}}, \boxed{\mathbf{c_2}} + \mathbf{c_3})$: here, $\mathbf{c_2}$ occurs in *two different parameters* of $\hat{f}$, therefore $\mathbf{c_2} \notin \mathcal{PAC}(t)$; however, $\mathbf{c_1}$ and $\mathbf{c_3}$ are potentially auxiliary counters;
- $h(\hat{g}(\mathbf{c_1}, \mathbf{c_2}), \hat{g}(\boxed{\mathbf{c_2}}, \boxed{\mathbf{c_2}}))$: $\mathbf{c_2}$ is not a potentially auxiliary counter for a subterm, thus neither can it be one for the whole term; but $\mathbf{c_1}$ is a potentially auxiliary counter (as is any counter other than $\mathbf{c_2}$).

4.3 Extended Presburger Systems

We can now define extended Presburger systems, which are intended to replace Presburger systems. The definitions of both notions are almost the same and their differences are highlighted:

Definition 8.

A rewrite system ($\mathcal{R}$) is said extended Presburger *if for each defined symbol $\hat{f}$ it contains:*

- *a unique* base rule *of the form:*

$$\hat{f}(\mathbf{0}, \vec{\mathbf{c}}; \vec{x}) \to r_1^{\hat{f}}$$

- *a unique* inductive rule *which is either of the form:*
  $\hat{f}(\mathbf{s(i)}, \vec{\mathbf{c}}; \vec{x}) \to r_2^{\hat{f}}[\hat{f}(\mathbf{i}, \vec{\mathbf{c}}; \vec{x})]_A$
  or
  $\hat{f}(\mathbf{s(i)}, \vec{\mathbf{c}}; \vec{x})$
  $\quad \to r_2^{\hat{f}}[\hat{f}(\mathbf{i}, \mathbf{c_1}, \ldots, \mathbf{c_{k-1}}, \mathbf{s(c_k)}, \mathbf{c_{k+1}}, \ldots, \mathbf{c_n}; \vec{x})]_A$

and such that

1. *A is a finite subset of parallel non empty positions in $r_2^{\hat{f}}$;*
2. *$r_2^{\hat{f}}[\hat{f}(\mathbf{i}, \mathbf{c_1}, \ldots, \mathbf{c_{k-1}}, \mathbf{s(c_k)}, \mathbf{c_{k+1}}, \ldots, \mathbf{c_k}; \vec{x})]_A$ and $r_2^{\hat{f}}[\hat{f}(\mathbf{i}, \vec{\mathbf{c}}; \vec{x})]_A$ are regular terms;*
3. ***$\mathbf{i}$ is a potentially principal counter of $r_2^{\hat{f}}$;***
4. ***$\mathbf{c_k}$ is a potentially auxiliary counter of $r_1^{\hat{f}}$ and $r_2^{\hat{f}}$;***
5. ***each defined symbol occurring in $r_1^{\hat{f}}$ or $r_2^{\hat{f}}$ is lower than $\hat{f}$;***

It is worth mentioning that the standard terms occurring in the redexes in $r_1^{\hat{f}}$ and $r_2^{\hat{f}}$ are not restricted; this is an important difference with standard primal grammars.

Also notice that the condition on approximation is replaced by conditions 3, 4 and 5 *while the remaining conditions (1 and 2) are not really restrictive.*

Example 6.

We express the informal term schema:

$$[[n, n+1, \ldots, n+k], [n+1, n+2, \ldots, n+k], \ldots, [n+k]]$$

by the following primal grammar ($\hat{s} \prec \hat{g} \prec \hat{f}$):

$$\begin{aligned}
\hat{f}(\mathbf{0}, \mathbf{j}) &\to [\,] \\
\hat{f}(\mathbf{s(i)}, \mathbf{j}) &\to [\hat{g}(\mathbf{s(i)}, \mathbf{j}) | \hat{f}(\mathbf{i}, \mathbf{s(j)})] \\
\hat{g}(\mathbf{0}, \mathbf{j}) &\to [\hat{s}(\mathbf{j})] \\
\hat{g}(\mathbf{s(i)}, \mathbf{j}) &\to [\hat{s}(\mathbf{j}) | \hat{g}(\mathbf{i}, \mathbf{s(j)})]
\end{aligned}$$

with axiom:

$$\hat{f}(\mathbf{k}, \mathbf{n})$$

We check that it is well-defined:

1. *Principal counters*
   - for $\hat{f}$, $\mathbf{i}$ is indeed a potentially principal counter since it occurs only in the first arguments;
   - for $\hat{g}$ and $\hat{s}$ all counters are potentially principal counters (each redex term only has one parameter).
2. *Auxiliary counters*
   - for both $\hat{f}$ and $\hat{g}$, $\mathbf{j}$ occurs only once in each redex term and thus is a potentially auxiliary counter;
   - $\hat{s}$ does not have any auxiliary counter.
3. the other conditions are easily checked (and are the same as those required for usual primal grammars).

4.4 Extended Primal Grammars

Definition 9.

An extended primal grammar *is a 3-tuple $(\Sigma, \mathcal{R}, t^\star)$ where:*

- *Σ is a primal signature,*
- ***$\mathcal{R}$ is an extended Presburger rewrite system,***
- *$t^\star$ is a primal term built on Σ, called* start symbol.

Property 2.

Any primal grammar is an extended primal grammar.

Proof. Immediate : the condition of approximation implies that the principal (resp. auxiliary) counter is indeed a potentially principal (resp. auxiliary) counter. □

Two extended primal grammars $(\Sigma, \mathcal{R}, t^\star)$ and $(\Sigma', \mathcal{R}', t^{\star\prime})$ are said to be *equivalent* if for any ground substitution of the counter variables σ:

$$t^\star\sigma{\downarrow_{\mathcal{R}}} = t^{\star\prime}\sigma{\downarrow_{\mathcal{R}'}}$$

Theorem (Equivalence).

There exists an algorithm transforming any extended primal grammar $(\Sigma, \mathcal{R}, t^\star)$ into an equivalent primal grammar $(\Sigma', \mathcal{R}', t^{\star\prime})$.

Proof. Some basic transformation rules are provided in Figure 1 on the next page in order to give a taste of the algorithm. Section A on page 7 contains a detailled proof and proposes a strategy ensuring its termination and correctness. □

It can be shown that the time complexity of our transformation algorithm is linear in the sizes of all counter expressions plus the number of defined symbols occurring in the *whole* extended primal grammar i.e. the grammar and all the grammars it involves.

The number of defined symbols added by the algorithm is linear in the sizes of counter expressions — each "+" occurring in a counter expression generates a new defined symbol.

4.5 A higher-level language

Here are examples that show what *can be done* with extended primal grammars *but not* with primal grammars:

Example 7.

$$\hat{f}(\mathbf{n}+\mathbf{m},\vec{\mathbf{c}}) \longrightarrow \hat{f}'(\mathbf{n},\mathbf{m},\vec{\mathbf{c}}) \quad \text{where} \begin{cases} \hat{f}'(\mathbf{0},\mathbf{m},\vec{\mathbf{c}}) \to \hat{f}(\mathbf{m},\vec{\mathbf{c}}) \\ \hat{f}'(\mathbf{s}(\mathbf{n}),\mathbf{m},\vec{\mathbf{c}}) \to \hat{f}(\mathbf{s}(\mathbf{n}+\mathbf{m}),\vec{\mathbf{c}}) \end{cases}$$

$$\hat{f}(\mathbf{k}\times\mathbf{n},\vec{\mathbf{c}}) \longrightarrow \hat{f}''(\mathbf{n},\vec{\mathbf{c}}) \quad \text{where} \begin{cases} \hat{f}''(\mathbf{0},\vec{\mathbf{c}}) \to \hat{f}(\mathbf{0},\vec{\mathbf{c}}) \\ \hat{f}''(\mathbf{s}(\mathbf{n}),\vec{\mathbf{c}}) \to \hat{f}(\mathbf{k}+\mathbf{k}\times\mathbf{n}),\vec{\mathbf{c}}) \end{cases}$$

Figure 1: Transforming an extended primal grammar into a standard one: rules are applied on terms occurring in the start axiom *or in the Presburger rewrite system*, and combined with the inductive and base rules of f. f', f'' denote new ("fresh") defined symbols.

1. *Use of any term as a parameter of a redex term.* More precisely, any of the following rules:

$$\hat{f}(\mathbf{0}) \to \hat{g}(\underline{\mathbf{s}(\mathbf{0})})$$
$$\hat{f}(\mathbf{0},\mathbf{j}) \to \hat{g}(\underline{\mathbf{s}(\mathbf{j})})$$
$$\hat{f}(\mathbf{0},\mathbf{j},\mathbf{k}) \to \hat{g}(\underline{\mathbf{j}+\mathbf{k}})$$

   would be allowed in an extended Presburger system but not in a standard Presburger system. Indeed each right member in the above rules involves a redex term whose parameter is any term, not necessarily a variable. Thus none of the right members are in the approximation of its corresponding left member.
   Here is a detailed example:

$$\hat{f}(\mathbf{0},\mathbf{j},\mathbf{k};x) \to \hat{g}(\underline{\mathbf{k},\mathbf{j}+\mathbf{k}};r(x,a))$$
$$\hat{f}(\mathbf{s}(\mathbf{i}),\mathbf{j},\mathbf{k};x) \to r(\hat{g}(\underline{\mathbf{i}+\mathbf{j},\mathbf{s}(\mathbf{k})};a),\hat{f}(\mathbf{i},\mathbf{s}(\mathbf{j}),\mathbf{k};x))$$

   We check that $\mathbf{i}$ (resp. $\mathbf{j}$) is a potentially principal (resp. auxiliary) counter:
   - in both rules, only $\mathbf{k}$ cannot be a principal counter; thus $\mathbf{i}$ is indeed a potentially principal counter.
   - in each redex term in which it appears, $\mathbf{j}$ occurs only in one parameter, and thus is a potentially auxiliary counter.

   We assume that $\hat{g} \prec \hat{f}$, and let the reader check that the other constraints are respected.
2. *The order and duplication of variables does not matter.* i.e. any of the following rule is acceptable in an extended Presburger system but not in a standard one:

$$\hat{f}(\mathbf{0},\underline{\mathbf{j},\mathbf{k}}) \to \hat{g}(\underline{\mathbf{k},\mathbf{j}})$$
$$\hat{f}(\mathbf{0},\mathbf{j},\mathbf{k};\underline{x,y}) \to \hat{g}(\mathbf{i},\mathbf{j},\mathbf{k};\underline{y,x})$$
$$\hat{f}(\mathbf{s}(\mathbf{i}),\underline{\mathbf{j}}) \to r(\hat{g}(\underline{\mathbf{j},\mathbf{j}}),\hat{f}(\mathbf{i},\mathbf{j}))$$

   The underlined terms show clearly that each redex term on the right side is not in the approximation of its corresponding left term.
   Here is a fully worked out example:

$$\hat{f}(\mathbf{0},\mathbf{j},\mathbf{k};x,y) \to \hat{g}(\mathbf{k},\mathbf{j};y,y)$$
$$\hat{f}(\mathbf{s}(\mathbf{i}),\mathbf{j},\mathbf{k};x,y) \to r(\hat{g}(\mathbf{i},\mathbf{k};y,x),\hat{f}(\mathbf{i},\mathbf{j},\mathbf{s}(\mathbf{k});x,y))$$

   Once again we check the extended Presburger system is well defined:
   - $\mathbf{i}$ does not occur in the base rule (of course!) and is the first parameter of each redex term of the induction rule; hence it is a potentially principal counter.
   - $\mathbf{j}$ only occurs in the base rule and only once, thus it is a potentially auxiliary counter.

Why those changes make the system easier to use ? The considerations below are usual considerations for judging the quality of a programming language. They do apply here.

The conditions are easier to check. Even though the notions of potentially principal and auxiliary counters can be seen as more intricate than the one of approximation, they are easier to check. Indeed they need looking at only one variable: the one intended to be the principal/auxiliary counter of the defined symbol we are currently defining. It allows the user to focus only on the variables that are useful for a particular definition whereas the notion of approximation implies to also take care of variables that are only present for other redex terms.

More modular. Imagine we change the profile of a particular symbol, for instance if we realize that the definition of this symbol did not match the intended meaning, and have to change it. Then we can check quickly if the other symbols still respect the constraints of an extended Presburger system. Indeed, we have seen that modifying the order of parameters or duplicating them is possible with extended primal grammars whereas it is not with standard ones. Furthermore, since the defined symbols are less dependent, this makes the definitions easier to reuse.

Higher level. The notions of potential counters are more elaborated and complex than the one of approximation, but their semantics better reflect their actual use: when we construct the definition of a symbol we focus on the principal/auxiliary counter; thus we want to know if the candidates are usable as such. The new definitions capture this notion more accurately.

Less verbose. The fact that we can use counter terms and not only variables in the extended Presburger system allows to express in few defined symbols some definitions that would usually need much more defined symbols. Furthermore it reduces the number of parameters.

5 Conclusion and ongoing research

This paper can be seen as a term schematisation manifesto as primal grammars are the most powerful of these formalisms but are not quite used. This fact is a natural consequence of the intrinsic complexity of this formalism. The main technical result of the paper allows more flexible use of term schematisation based on (decidable) fragments of arithmetic, without additional efforts of the user since a translation algorithm into standard primal grammars is provided. This is potentially useful in

all aplications where easiness of human/computer interaction is capital (as in A.I. and computer-aided mathematics). Hopefully, extended primal grammars will help to popularize the use of term schematisation in Artificial Intelligence.

As an immediate application, we are currently using extended primal grammars to schematise infinite sets of first order formulae produced by resolution theorem provers, a task for which flexible schematisation formalisms are mandatory (Aravantinos 2007).

A Proof of the equivalence

The goal of this section is to demonstrate the equivalence between extended and standard primal grammars.

We already have one implication of the equivalence by Property 2, i.e. for any primal grammar there is an equivalent extended primal grammar.

We now show that for every extended primal grammar, there is an equivalent primal grammar.

Lemma 1.

Let $(\Sigma, \mathcal{R}, t)$ be a primal grammar where $t = \hat{f}(\vec{p}; \vec{x})$ is a redex term, $\vec{p}$ is a vector of *counter expressions*.

There is an equivalent standard primal grammar $(\Sigma', \mathcal{R}', t')$ such that every sub-term of type int is a *variable*.

Furthermore, given a potentially principal (resp. auxiliary) counter i (and only one) for t, we can impose that i also be a potentially principal (resp. auxiliary) counter of t'.

Proof. (sketch) For any i, we note: $\{c'_{i,1}, \dots, c'_{i,m_i}\} = \mathcal{CV}ar(p_i)$ *(arbitrarily ordered)* and $\{c'_{0,1}, \dots, c'_{0,m_0}\} = \mathcal{CV}ar(n)$; we are going to construct a $\hat{f}'$ such that:

$$\hat{f}'(c'_{0,1}, \dots, c'_{0,m_0}, c'_{1,1}, \dots, c'_{1,m_1}, \dots ; \vec{x}) \equiv \hat{f}(p_1, \dots, p_k; \vec{x})$$

Or, diagramatically:

$$\begin{array}{lccccc} \hat{f}(& & n, & \dots & p_k & ; \vec{x}) \\ & & \downarrow & & \downarrow & \\ \hat{f}'(& \overbrace{c'_{0,1}, \dots, c'_{0,m_0}}, & & \dots & \overbrace{c'_{k,1}, \dots, c'_{k,m_k}} & ; \vec{x}) \end{array}$$

The structure of every counter expression is encoded in the *structure* of $\hat{f}'$. e.g. every addition in a counter expression is replaced by a defined symbol whose structure encodes the corresponding addition.

Proof. (formal) We define $n = p_0$.

- $\hat{f}$'s Presburger system is written:

$$\hat{f}(\mathbf{0}, \vec{c}; \vec{x}) \to r_1^{\hat{f}}$$
$$\hat{f}(\mathbf{s}(i), \vec{c}; \vec{x}) \to r_2^{\hat{f}}[\hat{f}(i, c_1, \dots, c_{j-1}, \mathbf{s}(c_j), c_{j+1}, \dots)]_A$$

  We consider the most general case where $\hat{f}$ has an auxiliary counter; indeed, the case without auxiliary counter can be reduced to it by adding a "dummy" additional auxiliary counter occurring neither in $r_1^{\hat{f}}$ nor in $r_2^{\hat{f}}$.

- We prove the result by nœtherian induction on the lexicographical order $(\prec, \leq)$, *where:*
  - $\prec$ *is the precedence on defined symbols (see Section 2.2 on page 3);*
  - $\leq$ *is the natural order on the depth of* n.

  $\mathcal{IH}(t)$ denotes the term obtained by applying the $\mathcal{I}$nduction $\mathcal{H}$ypothesis to the term t.

  1. if $n = \mathbf{0}$ or more generally, if n is a closed term:
     Then we can rewrite the term according to the Presburger system of $\hat{f}$. The head of every redex occurring in the obtained term is strictly lower than $\hat{f}$ according to $\prec$, thus we can apply the induction hypothesis to each redex term to get the desired result.
     Furthermore, potential counters are trivially kept through this transformation, by induction.
  2. if n is not a closed term and is of the form $n' + n''$; then by commutativity and associativity of addition we can impose that n' be a variable; we can even group all its occurrences: $n = k \times n' + n''$ where k is the number of occurrences of n' in m (thus n' does not occur in n'').
     If i occurs in n, we also impose that n' be i (not to lose the property of i being a potentially principal counter).
     - Then we denote by:
       * $r_2^{\hat{f}k}$ the term
       $$\underbrace{r_2^{\hat{f}\left\{\substack{i \leftarrow i+k-1 \\ c_j \leftarrow c_j}\right\}}\left[r_2^{\hat{f}\left\{\substack{i \leftarrow i+k-2 \\ c_j \leftarrow c_j+1}\right\}}\left[\dots r_2^{\hat{f}\left\{\substack{i \leftarrow i \\ c_j \leftarrow c_j+k-1}\right\}}\right]_A\right]_A}_{\times k}$$
       * $R_2^{\hat{f}}$ the term obtained from $r_2^{\hat{f}k}$ by replacing every redex $\hat{g_k}(\vec{d_k}; \vec{y_k})$ by:
       $$\mathcal{IH}\left(\hat{g_i}(\vec{d_i}; \vec{y_i})\left\{\substack{i \leftarrow i+n'' \\ c_1 \leftarrow p_1 \\ \dots \\ c_k \leftarrow p_k}\right\}\right)$$
       * $c'_{0,1} \stackrel{\text{def}}{=} n'$.
     - We apply the induction hypothesis on the term $\hat{f}(n'', \vec{p}; \vec{x})$, this is possible since the depth of n'' is strictly lower than the one of n. This gives us a term t' of counter variables $\vec{c}$. Note that i does not occur in $\vec{c}$ (since n' is not in $\vec{c}$ and if i is in $n = n' + n''$ then $n' = i$). Thus we introduce a new defined symbol $\hat{f}'$ associated with the following rewrite rules::
     $$\hat{f}'(\mathbf{0}, c'_{0,2}, \dots, c'_{0,m_0}, c'_{1,1}, \dots, c'_{k,m_k}; \vec{x}) \to \mathcal{IH}\left(\hat{f}(n'', \vec{p}; \vec{x})\right)$$
     $$\hat{f}'(\mathbf{s}(i), c'_{0,2}, \dots, c'_{0,m_0}, c'_{1,1}, \dots, c'_{k,m_k}; \vec{x}) \to R_2^{\hat{f}}[\hat{f}'(i, \vec{c}', \mathbf{s}(c'_{j,1}), \dots ; \vec{x})]_A$$

where we give $\hat{f}'$ the necessary rank such that the defined symbols introduced by induction are in the approximation.

The grouped treatment of all occurrences of a variable ensures that it is not "dispatched" on many parameters. Furthermore, if a variable is a potentially principal counter before the transformation, then we can impose that it still be after *(but if we choose such a potentially principal counter, the other potentially principal counters necessarily can't keep this property; this does not matter)*.

- We easily show the equality of the generated language:

$$\hat{f}'(n', c'_{0,2}, \dots, c'_{k,m_k}; \vec{x}) \equiv \hat{f}(n' + n'', \vec{p}; \vec{x})$$

- Hence we define: $u' \stackrel{\text{def}}{=} \hat{f}'(n', c'_{0,2}, \dots, c'_{k,m_k}; \vec{x})$ which indeed satisfies the property because n' is a variable.

3. if n is a variable, cf. 2 on the preceding page taking $n' = n$ and $n'' = \mathbf{0}$ *(because $n = n + \mathbf{0}$)*;
4. if n is of the form $\mathbf{s}(n')$, cf. 2 on the previous page taking $n'' = \mathbf{s}(\mathbf{0})$ *(because $\mathbf{s}(n) = n + \mathbf{s}(\mathbf{0})$)*;

□

Property 3.
For every extended primal grammar, there is an equivalent primal grammar.

Proof. (sketch) The goal is to construct a primal grammar from the extended one. Considering both definitions, the main point is to modify the extended grammar so as to respect the approximation constraint.

Three constraints could prevent us to reach our aim:

- The approximation is not defined for a term whose parameters are not simple variables. *Lemma 1 on the preceding page especially deals with this problem.*
- The principal counter shall be in first position in the profile of a defined symbol. This condition cannot be satisfied if ever a variable appears *before* it. *The notion of potentially principal counter is especially designed to deal with this.*
- The arguments of every defined symbol must be pairwise distinct. Hence it is impossible that a variable occurs twice as a parameter. *The notion of potentially auxiliary counter is especially designed to deal with this.*

Proof. (formal) Let $(\Sigma, \mathcal{R}, t^\star)$ be an extended primal grammar..

Let $\hat{f}$ be a defined symbol of principal counter i and auxiliary counter j.

We denote by $u_1, \dots, u_m$ the redex subterms of $r_1^{\hat{f}}$ and $r_2^{\hat{f}}$.

By definition of an extended Presburger system, we have for any $k \in 1..m$:

1. i (resp. j) is potentially principal (resp. auxiliary) counter of u_k;
2. $\mathcal{H}ead(u_k) \prec \hat{f}$.

We construct a defined symbol $\hat{f}'$ "equivalent to $\hat{f}$" but with standard Presburger system, by induction on $\prec$ Let $k \in 1..m$, we denote by u_k the term $\hat{g_k}(n_k, \vec{p_k}; \vec{t_k})$.

- We first apply the induction hypothesis to u_k gives a standard Presburger system for $\hat{g_k}$; let's now construct the Presburger system of $\hat{f}$.
- Let's denote by $u'_k = \hat{g}(n, \vec{p}; \vec{x})$ the term such that $u_k = u'_k\left\{\begin{smallmatrix} x_1 \leftarrow t_1 \\ x_2 \leftarrow t_2 \\ \dots \end{smallmatrix}\right\}$; as we now have a standard Presburger system for u'_k. Indeed, this transformation is needed to apply Lemma 1 on the previous page which requires that the non counter parameters be variables. Applying this lemma to u'_k gives us a term $u''_k = \hat{g'}(c_0, c_1, \dots; \vec{x})$ where $c_0, c_1, \dots$ are *counter variables*.
- i is a potentially principal counter of u_k, therefore by Lemma 1 on the preceding page, we can impose that i is also a potentially principal counter of u''_k; hence, either it is the first parameter of u''_k or it does not occur in it. This ensures that i does not cause issues for u''_k being in the approximation of $\hat{f}(i, \dots)$.
- Seemingly j being a potentially auxiliary counter of $r_1^{\hat{f}}$ and of $r_2^{\hat{f}}$, it occurs at most once in u_k.
- Hence we can distinguish two types of counter variables:
  - the ones that appear between i and j (thus *before* j), denoted by $\vec{c_1}$;
  - the ones that appear *after* j, denoted by $\vec{c_2}$.

  A variable may appear several times and this causes trouble for the approximation. However, those variables don't have any particular role in the Presburger system (i.e. are neither principal nor auxiliary). So we can rename them in order to make them distinct; equalities between variables will be restored outside the Presburger system.
- Then we define:

$$\hat{f}'(\mathbf{0}, \vec{c_1}, j, \vec{c_2}; \vec{x}) \to r_1^{\hat{f}} \text{ where each } u_k \text{ is replaced by } u''_k.$$

$$\hat{f}'(\mathbf{s}(i), \vec{c_1}, j, \vec{c_2}; \vec{x}) \to r_2^{\hat{f}} \text{ where each } u_k \text{ is replaced by } u''_k.$$

  By construction, it is indeed a Presburger system (i.e. each $u''_k \in \mathcal{A}px(\hat{f}'(i, \vec{c_1}, j, \vec{c_2}))$).
- Eventually we define $t'^\star = t^\star$ in which each subterm of the form $\hat{f}(n, \vec{p}; \vec{x})$ is replaced by $\hat{f}'(n, \vec{p}; \vec{t})$; then $(\Sigma, \mathcal{R}', t'^\star)$ is indeed a primal grammar equivalent to $(\Sigma, \mathcal{R}, t^\star)$.

□

References

Aravantinos, V. 2007. *Schémas de preuves et de formules : vers plus de structure et d'efficacité en déduction automatique.* Master thesis, Institut National Polytechnique de Grenoble & Université Joseph Fourier. Available on: `http://www-leibniz.imag.fr/~varavant`.

Chen, H.; Hsiang, J.; and Kong, H. 1990. On finite representations of infinite sequences of terms. In *Conditional and Typed Rewriting Systems*, 100–114. Springer, LNCS 516.

Comon, H.; Dauchet, M.; Gilleron, R.; Jacquemard, F.; Lugiez, D.; Tison, S.; and Tommasi, M. 2005. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`. release October, 12th 2007.

Comon, H. 1995. On unification of terms with integer exponents. *Mathematical System Theory* 28:67–88.

Hermann, M., and Galbavý, R. 1997. Unification of Infinite Sets of Terms schematized by Primal Grammars. *Theoretical Computer Science* 176(1–2):111–158.

Hermann, M. 1992. On the relation between primitive recursion, schematization, and divergence. In *Proceeding 3rd Conference on Algebraic and Logic Programming*, 115–127. Springer, LNCS 632.

Hermann, M. 1994. *Divergence des systèmes de réécriture et schématisation des ensembles infinis de termes.* Habilitation, Université Henri Poincaré Nancy 1.

Peltier, N. 2004. The First Order Theory of Primal Grammars is Decidable. *Theoretical Computer Science* 323:267–320.

Salzer, G. 1992. The unification of infinite sets of terms and its applications. In *Logic Programming and Automated Reasoning (LPAR'92)*, 409–429. Springer, LNAI 624.