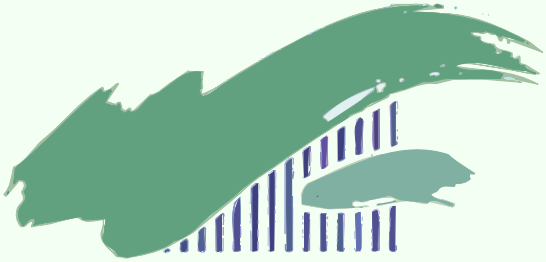

UNIVERSITE D'ORLEANS
Faculté des Sciences
LIFO

Laboratoire d'Informatique Fondamentale d'Orléans
4, rue Léonard de Vinci, BP 6759
F-45067 Orléans Cedex 2
FRANCE

Rapport de Recherche

www : http://www.univ-orleans.fr/SCIENCES/LIFO/

On Second Order Formulae of Pseudo-Regular Theory

Sébastien Limet et Pierre Pillot Université d'Orléans, LIFO

Rapport N°**2006-04**

Abstract

This report deals with a class of second order formulae where the only predicate is the joinability modulo a conditional term rewrite system, first order variables are ground terms and second order variables are relations on ground terms (i.e. sets of tuples of ground terms). We use a technique based on a representation of regular relations by logic programs to decide the existence of solutions of a class of second order pseudo-regular formulae and to exhibit one solution when it exists. Our algorithm produces conditional rewrite rules that define the value of the second order variables. Our technique may be useful to automatically synthetize a program that defines a relation from its specification.

1 Introduction

Second order theories have been widely studied for a long time because of their practical applications in several fields of computer sciences. The most studied class is monadic second order logic. Many variants of this logic have been proved decidable using automata techniques (see [17] for a survey). The solutions of formulae in such a logic is represented by automata on strings or trees e.g. the weak second order logic with k successors WSkS is solved using finite tree automata [16] that defines regular relations. Applications of monadic second order logic are numerous from circuit verification [5] which is the historical use, to program verification [8], or more recently queries in semi-structured databases [7].

In this paper, we study a class of formulae based on the predicate $\downarrow^?_R$ (i.e. the joinability modulo a terms rewrite systems). In the first order case, a solution of an equation $s \downarrow^?_R t$ is a substitution σ such that $s\sigma$ and $t\sigma$ rewrite into the same term. The term rewrite systems we consider are conditional term rewrite systems (CTRS for short). A CTRS defines a relation on terms for some function symbols by means of conditional rewrite rules of the form $l \to r \Leftarrow C$ (e.g. the CTRS of Example 1 above defines the relation + on positive integers). In the second order formulae studied in the present paper, second order variables represent such a relation. For example, a solution of the formula $\forall x X(x) \downarrow^?_R x + x$ where x is a first order variable and X a second order one, is a relation (t_1, t_2) such that $t_2 = 2 \times t_1$. Our aim is to automatically build the rewrite rules that define this relation. It is obvious that in the general case, solving second order formulae is undecidable since the joinability problem in the first order case is undecidable, so we need some restrictions on the CTRS.

In a previous paper [10], we defined the class of first order pseudo-regular formulae and showed that the solutions of such a formula can be represented by a regular relation. Some restrictions are imposed to the rewrite system to insure that the relations it defines are regular relations (those rewrite systems are called pseudo-regular TRS). The closure properties of regular relations is used to prove first that the solutions of a single equation are a regular relation, and then that the solutions of a formula are also regular. The technique used to represent and manipulate relations in that paper is based on logic programs and comes from [12].

The main contributions of this paper are the following. First, we extend the results of [10] to the case of CTRS. CTRS are syntactically much more expressive than TRS and are widely used in functional logic programming as the basis of languages like Babel or Curry. The second contribution is a decision procedure for a class of second order formulae, called positive second pseudo-regular formulae, that includes the first order theory. When a formula is satisfiable, we compute a CTRS that defines one possible instance for the second order variables. Therefore, our technique can be used to automatically generate a CTRS (which can be considered as a program) from the specification of the intended results.

Example 1 *The following CTRS is a pseudo-regular one. It represents the addition for positive integers represented by binary digit strings. In our encoding, the units are the leftmost symbols. For example the term* $0(1(\bot))$ *represents the binary number* 10 *i.e. the number* 2. $+$ *is the addition and* $\oplus$ *the addition with a carry over. The rules with* n_1 *or* n_2 *represents several rules where* n_1 *and* n_2 *can be replaced by* 0 *or* 1.

$\bot + n_1(y) \to n_1(y)$ $\quad$ $n_1(y) + \bot \to n_1(y)$
$0(x) + 0(y) \to 0(x + y)$ $\quad$ $1(x) + 1(y) \to 0(x \oplus y)$
$n_1(x) + n_2(y) \to 1(x + y)$ *if* $n_1 \neq n_2$ $\quad$ $\bot + \bot \to \bot$
$s(\bot) \to 1(\bot)$ $\quad$ $s(0(x)) \to 1(x)$
$s(1(x)) \to 0(s(x))$
$\bot \oplus 0(y) \to 1(y)$ $\quad$ $\bot \oplus 1(y) \to 0(s(y))$
$0(y) \oplus \bot \to 1(y)$ $\quad$ $1(y) \oplus \bot \to 0(s(y))$
$0(x) \oplus 0(y) \to 1(x + y)$ $\quad$ $1(x) \oplus 1(y) \to 1(x \oplus y)$
$n_1(x) \oplus n_2(y) \to 0(x \oplus y)$ *if* $n_1 \neq n_2$ $\quad$ $\bot \oplus \bot \to 1(\bot)$

This CTRS is not really conditional since there are no conditions in the rewrite rules. Let us consider the two following rules that defines the subtraction element by element of two lists of integers.

$sl(\bot, \bot) \to \bot$
$sl(c(x_1, y_1), c(x_2, y_2)) \to c(x_3, sl(y_1, y_2)) \Leftarrow x_2 + x_3 \downarrow_R x_1$

In the context of TRS without condition, the function sl would need the explicit definition of the subtraction between two integers.

$\forall x X(x) = x + x$ *is a positive second order formula, our procedure constructs automatically the following CTRS that defines the only possible instance for* X*.*

$f_X(\bot) \to \bot$ $\quad$ $f'_X(\bot) \to \bot$
$f_X(0(x)) \to 0(f_X(x))$ $\quad$ $f'_X(0(x)) \to 1(f_X(x))$
$f_X(1(x)) \to 0(f'_X(x))$ $\quad$ $f'_X(1(x)) \to 1(f'_X(x))$

Notice that f_X *introduces a* 0 *in the units and shift the digits of the number which corresponds to double a integer represented by binary digits.*

Section 2 states the basic notations and definitions used in this paper. Section 3 describes how we use logic programs to represent regular relations and recall some results of [10]. Section 4 defines the class of pseudo regular CTRS and describe how to translate from CTRS to logic programs and vice et versa. Section 5 defines the class of positive second order pseudo-regular formulae and describes the algorithm to decide them. Finally Section 6 concludes this paper.

2 Preliminaries

We recall some basic notions and notations concerning terms, conditional term rewrite systems and logic programming; for details see [2, 14].

First Order Terms and Relations. Let Σ be a finite set of symbols with arity, *Var* be an infinite set of variables, and $\mathcal{T}(\Sigma, Var)$ be the first-order term algebra over Σ *Var*. A term is *linear* if no variable occurs more than once in it and a term without variable is called a *ground term*. In this paper, Σ consists of three disjoint subsets: the set $\mathcal{F}$ of *defined function symbols*, the set $\mathcal{C}$ of *constructor symbols* and the set Pr of *predicate symbols*. The terms of $\mathcal{T}(\mathcal{C}, Var)$ are called *data-terms* and those of the form $P(\vec{t})$ where P is a predicate symbol of arity n and $\vec{t}$ is a vector of $\mathcal{T}(\mathcal{F} \cup \mathcal{C}, Var)^n$ are called *atoms*.

A *position* p is a string of integers whose length is denoted by $|p|$. For a term t, $Pos(t)$ denotes the set of *positions* in t, and $t|_u$ the *subterm* of t at position u. The term $t[u \leftarrow s]$ is obtained from t by replacing the subterm at position u by s. $Var(t)$ is the set of variables occurring in t. The set $\Sigma Pos(t) \subseteq Pos(t)$ denotes the set of non-variable positions, i.e., $t|_u \notin Var$ for $u \in \Sigma Pos(t)$ and $t|_u \in Var$ for $u \in Pos(t) \setminus \Sigma Pos(t)$. The *depth of a term* $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{C}, Var)$ denoted $Depth(t)$ is 0 if $t \in Var$ and $Max(\{\, |p| \mid p \in \Sigma Pos(t) \,\})$ otherwise. A term of depth 1 is said *flat*. The *depth of an atom* $P(\vec{t})$ denoted $Depth(P(\vec{t}))$ is $Max(\{\, Depth(s) \mid s \in \vec{t} \,\})$. An atom of depth 0 is said *flat*.

A substitution is a mapping from *Var* to $\mathcal{T}(\Sigma, Var)$ where $x\sigma \neq x$ for a finite set of variables. The *domain* of a substitution σ, $Dom(\sigma)$, is the set $\{\, x \in Var \mid x\sigma \neq x \,\}$. For $V \subseteq Var$, $\sigma|_V$ denotes the *restriction of* σ to the variables in V, i.e., $x\sigma|_V = x\sigma$ for $x \in V$ and $x\sigma|_V = x$ otherwise. If $\forall x \in Dom(\sigma)$, $x\sigma$ is a data-term then σ is called a *data substitution*. If a term t is an *instance* of a term s, i.e. $t = s\sigma$, we say that t *matches* s and s *subsumes* t.

Let $CVar = \{ \Box_i \mid i \geq 1 \}$ be the set of *context variables* distinct from Var, a *n-context* is a term t in $T(\Sigma, Var \cup CVar)$ such that each $\Box_i$ $1 \leq i \leq n$ occurs once and only once in t and no other element of $CVar$ occurs in t. $\Box_1$ (also denoted $\Box$) is called the *trivial context*, if C a term of $T(\mathcal{C}, Var \cup CVar)$ then C is a *constructor context*. For an n-context C, the expression $C[t_1, \ldots, t_n]$ denotes the term $C\{ \Box_i \mapsto t_i \mid 1 \leq i \leq n \}$.

This paper mainly deals with relations on ground data-terms. A *data-relation* $\tilde{r}$ of *arity* n is a subset of $\mathcal{T}^{n+1}(\mathcal{C})$. Notice that what we call arity for a relation is its number of components minus 1 because in our context, a relation $\tilde{r}$ with n components models in fact a relation from $\mathcal{T}^{n-1}(\mathcal{C})$ to $T(\mathcal{C})$. The notation $\tilde{r}(t_1, \ldots, t_{n-1})$ denotes the following set of ground data-terms $\{ t_n \mid (t_1, \ldots, t_{n-1}, t_n) \in \tilde{r} \}$. The set of all the possible data-relations is recursively enumerable, so we associate to each data-relation of arity n a unique *relation symbol* of arity n different from $(\Sigma \cup Var)$ and called the *name* of the relation. We denote $\mathcal{R}$ the set of relation names.

Logic programs. If $H, A_1, \ldots, A_n$ are atoms then $H \Leftarrow A_1, \ldots, A_n$ is a *Horn clause*, H is said to be the *head* of the clause and $A_1, \ldots, A_n$ is said to be the body. The elements of $Var(A_1, \ldots, A_n) \setminus Var(H)$ are called *existential variables*. A *logic program* is a set of Horn clauses. The *Herbrand domain* is the set of all ground atoms. The body of the clause $H \Leftarrow \mathcal{B}$ is said linear iff every variable occurs at most once in $\mathcal{B}$. A clause is said to be *linear* if both the head and the body are linear. A set of ground atoms S is an *Herbrand model* of the clause $H \Leftarrow \mathcal{B}$ iff $\forall \sigma$ such that $\mathcal{B}\sigma \subset S$, $H\sigma \in S$. S is an Herbrand model of the logic program $\mathcal{P}$ if it is a model of all clauses of $\mathcal{P}$. For a logic program $\mathcal{P}$ and a ground atom A we write $\mathcal{P} \models A$ if A belongs to the least Herbrand model of $\mathcal{P}$ (denoted $\mathcal{M}(\mathcal{P})$). The *language* described by a n-ary predicate symbol P w.r.t. a program $\mathcal{P}$ is the set $\{ (t_1, \ldots, t_n) \mid \mathcal{P} \models P(t_1, \ldots, t_n) \}$ of n-tuples of ground terms.

Conditional Term Rewrite Systems. A *term rewrite system* (TRS) is a set of oriented equations built over $T(\mathcal{F} \cup \mathcal{C}, Var)$ and called *rewrite rules*. Lhs and rhs are shorthands for the left-hand and right-hand side of a rule, respectively. For a TRS R, the *rewrite relation* is denoted by $\rightarrow_R$ and is defined by $t \rightarrow_R s$ iff there exists a rule $l \rightarrow r$ in R, a non-variable position u in t, and a substitution σ, such that $t|_u = l\sigma$ and $s = t[u \leftarrow r\sigma]$. Such a step is written as $t \rightarrow_{[u, l \rightarrow r]} s$. If σ is a data-substitution then the step is called a *data-step*. If a term t cannot be reduced by any rewriting rule, it is said to be *irreducible*. The reflexive-transitive closure of $\rightarrow_R$ is denoted by $\rightarrow_R^*$. The *joinability relation* $\downarrow_R$ is defined by $t \downarrow_R s$ iff $t \rightarrow_R^* u$ and $s \rightarrow^* u$ for some term u. Notice that R-joinability is equivalent to R-unifiability for confluent rewrite systems.

A *conditional term rewrite system*, CTRS for short, is a finite set of rewrite rules of the form $l \rightarrow r \Leftarrow \mathcal{B}$ where $\mathcal{B}$ is a finite conjunction of conditions that must be checked before rewriting. A rewrite step, $t \rightarrow_R s$ iff there exists a conditional rule $l \rightarrow r \Leftarrow \mathcal{B}$ in R, a non-variable position u in t, and a substitution σ, such that $t|_u = l\sigma$ and $s = t[u \leftarrow r\sigma]$ and $\mathcal{B}\sigma$ is true. The notion of data-step extends trivially to the conditional case.

In this paper, we consider *join CTRS* i.e. the conditions are pairs of terms $s \downarrow_R t$ which are verified for a substitution σ if $s\sigma$ and $t\sigma$ are R-joinable. Moreover, we focus on *constructor based CTRS*, i.e. CTRS which lhs of rewrite rules are of the form $f(t_1, \ldots, t_n)$ where f is defined function symbol and each t_i $(1 \leq i \leq n)$ is a data-term.

In the context of constructor based CTRS, a *data-solution* of a joinability equation $s \downarrow_R^? t$ is a data-substitution σ such that $s\sigma \rightarrow_R^* u$ and $t\sigma \rightarrow_R^* u$ where u is a data term and all rewriting steps are data-steps. For a CTRS R and a defined function f, we denote $\tilde{f}$ the following data-relation $\tilde{f} = \{ (t_1, \ldots, t_n, t) \in T^{n+1}(\mathcal{C}) \mid f(t_1, \ldots, t_n) \rightarrow_R^* t$ with a data only data-steps $\}$.

3 Pseudo-regular relations

This section first presents the logic programming formalism we use to represent the tree tuple languages. Then, it presents results we use in the section 5. All the results presented here come from [10] and [12].

A *cs-program* is simply a logic program consisting of Horn clauses which bodies are linear and without function symbols. [12] introduces two subclasses of cs-programs, called respectively regular and pseudo-regular programs. The first class corresponds to the regular relations of [4], and the second one weakens the syntax of regular programs but [10] shows that regular and pseudo-regular programs define the same class of relations. The next definition introduces regular and pseudo-regular programs as well as a more general class that still describes regular relations.

Definition 1 *A Horn clause is called* non-Greibach *(NG for short) if at least one of the arguments of the head is of depth more than one. Let $H \Leftarrow \mathcal{B}$ be a clause such that $\mathcal{B}$ contains no function symbols.*

- *$H \Leftarrow \mathcal{B}$ is called* NGPR-like *iff none of the arguments of H are variables and there exists a mapping π: $Var \mapsto \mathbb{N}^+$ such that $\pi(x) = u$ then all occurrences of x in the arguments of H are at position u and $\pi(x) = \pi(y)$ for all variables x and y occurring in the same body atom.*
- *$H \Leftarrow \mathcal{B}$ is called* NGSPR *iff it is NGPR-like and it contains no existential variable*
- *$H \Leftarrow \mathcal{B}$ is called* pseudo-regular *(PR for short) iff it is NGPR-like and $\mathcal{B}$ is linear and $Depth(H) = 1$.*
- *$H \Leftarrow \mathcal{B}$ is called* regular *(R for short) iff it is PR and H is linear*

A program is PR-like, PR, R, NGSPR if all its clauses are of the corresponding type.

Example 2 *The clause $P(d(y_1, x_1), c(x_1, y_2)) \Leftarrow P_1(x_1), P_2(y_1, y_2)$ is not NG-PR-like since x_1 is at position 2 in the first argument of the clause head and at position 1 in the second argument.*

The clause $P(d(x_1, y_1), c(x_1, y_2)) \Leftarrow P_1(y_1), P_2(x_1, y_2)$ is also not NG-PR-like since x_1 and y_2 occur in the same body atom but they do not occur at the same position in the head of the clause.

The clause $P(c(s(x), y), s(s(z))) \Leftarrow P(x, z), Q(y)$ is NGSPR since x and z occur both at occurrence 1.1 in the arguments of the head.

The clauses $P_{sl}(c(x_1, y_1), c(x_2, y_2), c(x_3, y_3)) \Leftarrow P_+(x_2, x_3, x_1), P_{sl}(y_1, y_2, y_3)$ and $P_{sl}(\bot, \bot, \bot) \Leftarrow$ are both regular.

Lemma 1 *[10] Any NGSPR program $\mathcal{P}$ can be transformed into an equivalent finite PR program. Any PR program can be transformed into an equivalent finite regular program.*

This lemma is proved in [10] using a logic program transformation technique presented in [12].

Decidability of membership and emptiness tests as well as closure under intersection of pseudo-regular relations have been shown in [12]. An algorithm to compute the closure under complement is specified in [10].

4 Pseudo-regular CTRS

This section specifies the class of CTRS we consider and shows how to translate from CTRS to logic programs and vice versa. The translation from CTRS to logic programs is an extension of [13] while the reverse translation is a new contribution of this paper.

Definition 2 *A conditional constructor based CTRS R is said pseudo-regular if all its rewrite rules are of the form $f(t_1, \ldots, t_n) \to C[f_1(\vec{x}_1), \ldots, f_m(\vec{x}_m)] \Leftarrow f'_1(\vec{x}'_1) \downarrow_R x'_1 \ldots f'_k(\vec{x}'_k) \downarrow_R x'_k$ where*

- *C is a constructor context, $f, f_1, \ldots, f_m, f'_1, \ldots, f'_k$ are defined function symbols*
- *$\bigcup_{1 \leq i \leq m}(\vec{x}_1) \bigcup_{1 \leq i \leq k}(\vec{x'_i}) \bigcup_{1 \leq i \leq k}(x'_k) \subseteq Var(f(t_1, \ldots, t_n)) \cup Var(C)$*
- *there exists a mapping π: $Var \mapsto \mathbb{N}^+$, such that $\pi(x) = u$ implies that all occurrences of x in $t_1, \ldots, t_n$ and in C are at position u,*

$$\frac{\top}{v \rightsquigarrow \langle v, \emptyset \rangle} \quad \text{if } v \in \mathit{Var}$$

$$\frac{s_1 \rightsquigarrow \langle t_1, \mathcal{G}_1 \rangle \ldots s_n \rightsquigarrow \langle t_n, \mathcal{G}_n \rangle}{f(s_1, \ldots, s_n) \rightsquigarrow \langle f(t_1, \ldots, t_n), \bigcup_i \mathcal{G}_i \rangle} \quad \text{if } f \in \mathcal{C}$$

$$\frac{s_1 \rightsquigarrow \langle t_1, \mathcal{G}_1 \rangle \ldots s_n \rightsquigarrow \langle t_n, \mathcal{G}_n \rangle}{f(s_1, \ldots, s_n) \rightsquigarrow \langle x, \bigcup_i \mathcal{G}_i \bigcup \{P_f(t_1, \ldots, t_n, x)\} \rangle} \quad \text{if } f \in \mathcal{F}$$

$$\frac{s_1 \rightsquigarrow \langle t_1, \mathcal{G}_1 \rangle \qquad s_2 \rightsquigarrow \langle t_2, \mathcal{G}_2 \rangle}{s_1 \downarrow_R s_2 \rightsquigarrow \langle \varepsilon, \mathcal{G}_1 \cup \mathcal{G}_2 \cup P_{id}(t_1, t_2) \rangle}$$

$$\frac{s \rightsquigarrow \langle t, \mathcal{G} \rangle \quad c_1 \rightsquigarrow \langle \varepsilon, \mathcal{G}_1 \rangle \ldots c_k \rightsquigarrow \langle \varepsilon, \mathcal{G}_k \rangle}{f(s_1, \ldots, s_n) \rightarrow s \Leftarrow c_1 \ldots c_k \rightsquigarrow P_f(s_1, \ldots, s_n, t) \Leftarrow \mathcal{G} \cup \mathcal{G}_1 \cup \ldots \cup G_k}$$

Note that the variables x introduced by the third rule are new fresh variables.

Table 1: Converting CTRS rules to Horn clauses

- *all the variables of $\vec{x}'_i$ have the same image by π as x'_i $(1 \leq i \leq k)$.*
- *the image by π of all the variables of $\vec{x}_i$ is u, the position of $f_i(\vec{x}_i)$ in $C[f_1(\vec{x}_1), \ldots, f_m(\vec{x}_m)]$ $(1 \leq i \leq m)$.*

C is said the irreducible part *of $C[f_1(\vec{x}_1), \ldots, f_m(\vec{x}_m)]$, the term $f_i(\vec{x}_i)$ are called* possible redexes *and the positions of the f_i in C the* possible redex positions *of $C[f_1(\vec{x}_1), \ldots, f_m(\vec{x}_m)]$*

Example 3 *$R = \{f(s(c(x,y)), s(c(x,z))) \rightarrow s(c(g(x), f(y,y))) \Leftarrow g(z) \downarrow_R y, f(s(0), s(0)) \rightarrow 0, g(s(x)) \rightarrow s(x)\}$ is pseudo-regular. The irreducible part of the lhs of the first rule is $s(c(\Box_1, \Box_2))$ and it contains two possible redex positions namely 1.1 and 1.2 corresponding to the possible redexes $g(x)$ and $f(y,y)$. Notice that the definition 2 does not forbid duplicated variables in a single possible redex.*

We extend the technique presented in [13] that encodes the rewrite relation by a logic program in order to be able to deal with CTRS. This translation intends to obtain logic programs that preserve as best as possible syntactic properties of the TRS. The obtained logic program encodes the rewrite relation with data-steps.

Table 1 specifies the rules that transform terms and conditional rewrite rules to Horn clauses. P_{id} is a pseudo-regular predicate that defines the equality between data terms. Its set of clauses is denotes $\mathcal{P}_{id}$ and is

$$\mathcal{P}_{id} = \{\, P_{id}(c(x_1, \ldots, x_n), c(y_1, \ldots, y_n)) \Leftarrow P_{id}(x_1, y1), \ldots, P_{id}(x_n, y_n) \mid c \in \mathcal{C} \,\}$$

For a CTRS R, let $\mathcal{LP}(R)$ denotes the logic program consisting of $\mathcal{P}_{id}$ union the set of clauses obtained by applying the fourth rule to all rewrite rules in R.

Remark 1 *In the context of pseudo-regular CTRS, the conditions are of the form $f(x_1, \ldots, x_n) \downarrow_R y$. The transformation gives $P_f(x_1, \ldots, x_n, z), P_{id}(y, z)$ which we simplify by $P_f(x_1, \ldots, x_n, y)$.*

For example, the first rewrite rule of Example 3 is transformed into
$P_f(s(c(x,y)), s(c(x,z)), s(c(x_1, x_2))) \Leftarrow P_g(x, x_1), P_f(y, y, x_2), P_g(z, y)$.
The rewrite rules that defines function *sl* of Example 1 are transformed into the two clauses headed by P_{sl} in Example 2
The following theorem states the relation between a constructor based CTRS R and $\mathcal{LP}(R)$.

Theorem 1 *Let R be a CTRS, s a term such that $s \rightsquigarrow \langle s', \mathcal{G} \rangle$. $s \rightarrow^* t$ iff t is a data-term $\mathcal{LP}(R) \models \mathcal{G}\mu$ and $t = s'\mu$ where μ is a data substitution.*

The proof of this theorem is essentially the same as the equivalent one for pseudo-regular TRS. It can be found in [11].

Lemma 2 *Let R be a pseudo-regular CTRS then $\mathcal{LP}(R)$ is a NGSPR logic program.*

Proof 1 *Let $l \to r \Leftarrow C$ a rule of the pseudo-regular CTRS R. $l \to r$ is a pseudo-regular rule according to [10], therefore it is sufficient to prove that all variables of each atom A of $\mathcal{G}$ (with $C \rightsquigarrow \langle \varepsilon, \mathcal{G} \rangle$) have the same image by π which is by definition the case since each atom A is of the form $P_f(\vec{x}, x)$ when it comes from the condition $f(\vec{x}) \downarrow_R^? x$.*

Now, we give the transformation from a NGSPR program into a pseudo-regular CTRS.

For each flat atom A of the form $P(x_1, \ldots, x_{n-1}, x_n)$ we define $term(A) = f_P(x_1, \ldots, x_{n-1})$ and $equat(A) = f_P(x_1, \ldots, x_{n-1}) \downarrow_R x_n$. We extend *equat* to a sets of flat atoms $\mathcal{G}$ in the natural way i.e. $equat(\mathcal{G}) = \{\, equat(A) \mid A \in \mathcal{G} \,\}$.

Definition 3 *Let $\mathcal{C} = P(t_1, \ldots, t_{n-1}, C[x_1, \ldots, x_m]) \Leftarrow \mathcal{G}, \mathcal{G}'$ be a NGSPR clause where $\mathcal{G} = \{\, P(\vec{x}, x) \mid x \in \{x_1, \ldots, x_m\}$ and x occurs once in $\mathcal{G}, \mathcal{G}' \,\}$*

$\mathcal{RR}(\mathcal{C}) = f_P(t_1, \ldots, t_{n-1}) \to C[x_1, \ldots, x_m]\sigma \Leftarrow equat(\mathcal{G}')$ where $\sigma = \{\, x \mapsto term(P(\vec{x}, x)) \mid P(\vec{x}, x) \in \mathcal{G} \,\}$.

For a NGSPR logic program $\mathcal{P}$, let $\mathcal{RR}(\mathcal{P})$ denote the CTRS consisting of the rules $\{\, \mathcal{RR}(H \Leftarrow \mathcal{G}) \mid H \Leftarrow \mathcal{G} \in \mathcal{P} \,\}$.

For example, $P_f(s(x_1, y), s(x_2, y), s(z_1, z_2)) \Leftarrow P_f(x_2, x_1, z_1) P_f(x_1, x_1.x_2)$ is transformed into $f(s(x_1, y), s(x_2, y)) \to s(f(x_2, x_1), z_2) \Leftarrow f(x_1, x_1) \downarrow_R x_2$ and the two clauses headed by P_{sl} in Example 2 are transformed into the two rewrite rules for the function *sl* in the CTRS of Example 1.

Lemma 3 *Let $\mathcal{P}$ be a NGSPR logic program, $\mathcal{RR}(\mathcal{P})$ is a pseudo-regular CTRS.*

Proof 2 *Let us consider a clause $\mathcal{C} = H \Leftarrow \mathcal{G}, \mathcal{G}'$ of the NGSPR program $\mathcal{P}$ where $H = P(t_1, \ldots, t_{n-1}, C[x_1, \ldots, x_m])$, $\mathcal{G}$ is the set of atoms $P(\vec{x}, x)$ such that $x \in \{x_1, \ldots, x_m\}$ and x occurs only once in $\mathcal{G}, \mathcal{G}'$. The corresponding rewrite rules is $f_P(t_1, \ldots, t_{n-1}) \to C[x_1, \ldots, x_m]\sigma \Leftarrow equat(\mathcal{G}')$ where $\sigma = \{\, x \mapsto term(P(\vec{x}, x)) \mid P(\vec{x}, x) \in \mathcal{G} \,\}$. From the definition of NGSPR programs, we know that*

- *No existential variables occurs in $\mathcal{C}$ which means that $Var(\mathcal{G}, \mathcal{G}') \subseteq Var(H)$ hence $\bigcup_{1 \le i \le m} Var(x_i\sigma) \cup Var(equat(\mathcal{G}') \subseteq Var(f_P(t_1, \ldots, t_{n-1}) \cup Var(C)$.*
- *There exists a mapping π from Var to $\mathbb{N}^+$ such that if $\pi(x) = p$ then all occurrences of x are at position p in $t_1, \ldots, t_n$ and C*
- *All the variables of a given atom A of $\mathcal{G}, \mathcal{G}'$ have the same image by π therefore all the variables of each equation of $equat(\mathcal{G}')$ have the same image by π. Moreover all the variables of $x\sigma$ have the same image as x by π i.e. if $x \in \{x_1, \ldots, x_n\}$, the position of $x\sigma$ in $C[x_1, \ldots, x_m]\sigma$.*

So we can conclude that $\mathcal{RR}(\mathcal{C})$ is a NGSPR conditional rewrite rule.

Lemma 4 *Let $\mathcal{P}$ be a NGSPR logic program. $\mathcal{LP}(\mathcal{RR}(\mathcal{P})) = \mathcal{P}$.*

Proof 3 *In this proof we consider that P_{f_P} is a notation for the predicate symbol P. Let $\mathcal{P}$ be a NGSPR logic program and $\mathcal{C}$ be a clause of $\mathcal{P}$. $\mathcal{C}$ is of the form $P(t_1, \ldots, t_{n-1}, C[x_1, \ldots, x_m]) \Leftarrow \mathcal{G}, \mathcal{G}'$.*

$\mathcal{RR}(\mathcal{C}) = f_P(t_1, \ldots, t_{n-1}) \to C[x_1, \ldots, x_m]\sigma \Leftarrow equat(\mathcal{G}')$ where $\sigma = \{\, x \mapsto term(P(\vec{x}, x)) \mid P(\vec{x}, x) \in \mathcal{G} \,\}$. For each atom $P(\vec{x}, x)$ of $\mathcal{G}'$, $equat(P(\vec{x}, x)) = f_P(\vec{x}) \downarrow_R x$. From Remark 1, each equation $f_P(\vec{x}) \downarrow_R x$ is transformed into $P_{f_P}(\vec{x}, x)$ (i.e $P(\vec{x}, x)$).

Each atom $P(\vec{x}, x)$ of $\mathcal{G}$ is transformed into the term $f_P(\vec{x})$. From Table 1 $f_P(\vec{x}) \rightsquigarrow \langle x, P_{f_P}(\vec{x}, x) \rangle$, therefore $C[x_1, \ldots, x_m]\sigma \rightsquigarrow \langle C[x_1, \ldots, x_m], \mathcal{G} \rangle$. Hence $f_P(t_1, \ldots, t_{n-1}) \to C[x_1, \ldots, x_m]\sigma \Leftarrow equat(\mathcal{G}') \rightsquigarrow \mathcal{C}$.

Theorem 2 *Let $\mathcal{P}$ be a NGSPR logic program and $P(\vec{t}, t)$ a ground atom. $\mathcal{P} \models P(\vec{t}, t)$ iff $f_P(\vec{t}) \to^* t$.*

Proof 4 *From Lemma 4, we know that $\mathcal{LP}(\mathcal{RR}(\mathcal{P})) = \mathcal{P}$. The term $f_P(\vec{t}) \rightsquigarrow \langle x, P(\vec{t}, x) \rangle$ because all the terms of $\vec{t}$ are ground data terms and we consider P_{f_P} as a notation for P. From Theorem 1 we know that $f_P(\vec{t}) \to^* t$ iff t is a ground data-term, $\mathcal{LP}(\mathcal{RR}(\mathcal{P})) \models P(\vec{t}, x)\mu$ and $t = x\mu$, in other words iff $\mathcal{P} \models P(\vec{t}, t)$.*

5 Second order pseudo-regular formulae

In this section, we present the kind of formulae we deal with and the way we solve them. We first define the second order algebra we consider.

5.1 Second order

Let $\mathcal{X}$ be a set of second order variables with arity, different from $\Sigma \cup \mathit{Var} \cup \mathcal{R}$. The set of second order terms built over the signature $\mathcal{T}(\Sigma, \mathit{Var}, \mathcal{R}, \mathcal{X})$ is the smallest set such that

- Var is included in this set,
- if $t_1, \ldots, t_n$ are second order terms and f is either a symbol of arity n of Σ or a variable of arity n of $\mathcal{X}$ or a relation symbol of arity n of $\mathcal{R}$, then $f(t_1, \ldots, t_n)$ is a second order term.

For a second order term $Var(t)$ denotes the set of first order and second order variables of t. A second order term t is said *ground* if $Var(t)$ is empty. Let R be a CTRS, t be a ground second order term, the *model* of t is denoted by $\mathcal{M}_R(t)$ and is the smallest set of ground data terms such that

- $\{\, c(t_1, \ldots, t_n) \mid \forall 1 \leq i \leq n, t_i \in \mathcal{M}_R(s_i) \,\}$ if $t = c(s_1, \ldots, s_n)$ and $c \in \mathcal{C}$,
- $\{\, \tilde{f}(t_1, \ldots, t_n) \mid \forall 1 \leq i \leq n, t_i \in \mathcal{M}_R(s_i) \,\}$ if $t = f(s_1, \ldots, s_n)$ and $f \in \mathcal{F}$,
- $\{\, \tilde{r}(t_1, \ldots, t_n) \mid \forall 1 \leq i \leq n, t_i \in \mathcal{M}_R(s_i) \,\}$ if $t = \tilde{r}(s_1, \ldots, s_n)$ and $\tilde{r} \in \mathcal{R}$.

Notice that for a first order term ground term t, $\mathcal{M}_R(t)$ is the set of ground data terms s such that $t \to_R^* s$ with a data-derivation. This is easily proved by induction on the height of the term t.

A *second order substitution* σ is a mapping from $\mathit{Var} \cup \mathcal{X}$ to $\mathcal{T}(\mathcal{F} \cup \mathcal{C}, \mathit{Var}) \cup \mathcal{R} \cup \mathcal{X}$ such that $x\sigma \neq x$ for a finite subset of $\mathit{Var} \cup \mathcal{X}$ and

- if $x \in \mathit{Var}$, $x\sigma \in \mathcal{T}(\mathcal{F} \cup \mathcal{C}, \mathit{Var})$
- if $x \in \mathcal{X}$ and x is of arity n, $x\sigma$ is a data-relation of arity n or is x itself.

The domain of σ is the set of variables such that $x\sigma \neq x$. A second order substitution is called *data ground* if all $x \in \mathit{Dom}(\sigma) \cap \mathit{Var}$, $x\sigma$ is a ground data term. We extend σ to $\mathcal{T}(\Sigma, \mathit{Var}, \mathcal{R}, \mathcal{X})$ trivially.

Example 4 *Let us consider the CTRS of Example 1, the relation $\tilde{r} = \{(t_1, 0(t_1))\}$, the second order term $X(x) + y$ and the substitution $\sigma = \{X \mapsto \tilde{r}, x \mapsto 1(\bot), y \mapsto 1(0(\bot))\}$. $\mathit{Var}(t) = \mathit{Dom}(\sigma) = \{X, x, y\}$. σ is data-ground. $t\sigma = \tilde{r}(1(\bot)) + 1(0(\bot))$ is a ground term and $\mathcal{M}_R(t\sigma) = \{0(0(1(\bot)))\}$ since $\mathcal{M}_R(x\sigma) = \{1(\bot)\}$ so $\mathcal{M}_R(\tilde{r}(x\sigma)) = \{0(1(\bot))\}$ and finally $\mathcal{M}_R(t\sigma)$ is the union $\tilde{+}(t_1, t_2)$ such that $t_1 \in \mathcal{M}_R(\tilde{r}(x\sigma))$ and $t_2 \in \mathcal{M}_R(y\sigma)$ i.e. the set $\{\tilde{+}(0(1(\bot)), 0(1(\bot)))\} = \{0(0(1(\bot)))\}$.*

Notice that the model of the ground term $c(\bot, \bot) + 0(\bot)$ is empty since the first component of $\tilde{+}$ is never headed by the constructor symbol c.

5.2 Pseudo-regular equations and formulae

Definition 4 *Let R be a pseudo regular CTRS. A* second order pseudo-regular joinability equation *$s \downarrow_R^? t$ is an equation such that s and t are second order terms. A* first order pseudo-regular joinability equation *is an equation $s \downarrow_R^? t$ such that neither s nor t contain second order variables.*

Notice that pseudo-regular joinability equations do not contain any constructor symbols. This restriction may be weaken by allowing the sides of the equations to be of the form $C[t_1, \ldots, t_n]$ where C is a context that does not contain any constructor symbols and $t_1, \ldots, t_n$ are ground data-terms. Indeed, in this case we can introduce to the CTRS the rules $\{\, f_{t_i} \to t_i \mid 1 \leq i \leq n \,\}$ where f_{t_i} are new defined function symbols of arity 0. It is obvious that these new rewriting rules are pseudo-regular since they do not contain any variables. Moreover $\tilde{f_{t_i}} = \{t_i\}$ thus $\mathcal{M}_R(C[t_1, \ldots, t_n]) = \mathcal{M}_R(C[f_1, \ldots, f_n])$.

Definition 5 *Let R be a CTRS, and $s \downarrow^?_R t$ be a second order pseudo-regular joinability equation. A ground data substitution σ is a solution of $s \downarrow^?_R t$ iff $s\sigma$ and $t\sigma$ are ground terms and $\mathcal{M}_R(s\sigma) \cap \mathcal{M}_R(t\sigma) \neq \emptyset$.*

Notice that the definition given in [10] for first order joinability equations states that σ is a solution of $s \downarrow^?_R t$, iff $s\sigma \to^*_R u$ and $t\sigma \to^*_R u$ where u is a data term and all rewriting steps are data-steps. Since $\mathcal{M}_R(s\sigma)$ is the set of ground data terms reachable from $s\sigma$ by a data-derivation and $\mathcal{M}_R(t\sigma)$ is the equivalent set for $t\sigma$, the two definitions coincide.

Let R be a pseudo-regular CTRS, first order pseudo-regular R-formulae are defined by the following grammar:

$$e ::= s \downarrow^?_R t | \neg e | e \vee e | e \wedge e | \forall x e | \exists x e$$

where $s \downarrow^?_R t$ is a second order pseudo regular equation and x is a first order variable.

The set solutions of such a formula is defined as follows

- $SOL(s \downarrow^?_R t) = \{ \sigma \mid \mathcal{M}_R(s\sigma) \cap \mathcal{M}_R(t\sigma) \neq \emptyset \}$
- $SOL(\neg e) = \{ \sigma \mid \sigma \notin SOL(e) \}$
- $SOL(e_1 \wedge e_2) = \{ \sigma \mid Dom(\sigma) = Var(e_1 \wedge e_2), \sigma|_{Var(e_1)} \in SOL(e_1), \sigma|_{Var(e_2)} \in SOL(e_2) \}$
- $SOL(e_1 \vee e_2) = \{ \sigma \mid Dom(\sigma) = Var(e_1 \vee e_2), \sigma|_{Var(e_1)} \in SOL(e_1)$ or $\sigma|_{Var(e_2)} \in SOL(e_2) \}$
- $SOL(\exists x e) = \{ \sigma \mid Dom(\sigma) = Var(e) \setminus \{x\}, \exists \sigma' \in SOL(e), \sigma = \sigma'|_{Var(e)\setminus\{x\}} \}$
- $SOL(\forall x e) = \{ \sigma \mid Dom(\sigma) = Var(e) \setminus \{x\}, \forall \sigma' \in SOL(e), \sigma = \sigma'|_{Var(e)\setminus\{x\}} \}$

Example 5 *Let us consider once more the CTRS of Example 1. The following formula is pseudo-regular*

$$\forall x \exists y, \neg y + y \downarrow^?_R x \vee X(x) \downarrow^?_R true$$

The smallest solution of this equation instanciates the variable X to the relation $\tilde{e} = \{ (n, true) \mid n$ is an even number$\}$

Unfortunately, we cannot solve any second order pseudo-regular formulae since most of their solutions instanciate second order variables with non regular relations. Even if we restrict ourselves to regular solutions, they are usually infinitely many and incomparable. Since our tools cannot represent such infinite sets, we restrict ourselves to the class of *positive second formulae* i.e. formulae where second order equations are never under a negation. Thus, we restrict the rule of the grammar $e\ e ::= \neg e$ if the formulae e is a first order one. Moreover, the result of our algorithm is either the empty set if the formula as no model or the set of solutions corresponding to one particular instance of the second order variables. For the example above, our algorithm does not compute the smallest solution.

5.3 Solving Second Order Pseudo-regular Formulae

The main result of [10] is an algorithm to solve any first order pseudo-regular formula when the pseudo-regular TRS is not a conditional one. This algorithm can be used in the conditional case since it is based on the ability to represent the rewrite relation by a NGSPR logic program which is the case for pseudo-regular CTRS (see Theorem 1). The idea of the resolution algorithm of [10] is the following. The solutions of an first order joinability equation $s \downarrow^?_R t$ can be represented by the PR program computed from $\mathcal{LP}(R)$ and the clause $P(\vec{x}) \Leftarrow \mathcal{G}_s, G_t, P_{Id}(x_s, x_t)$ when $s \rightsquigarrow \langle x_s, \mathcal{G}_s \rangle$ and $t \rightsquigarrow \langle x_t, \mathcal{G}_t \rangle$ and $\vec{x} = Var(s \downarrow^?_R t)$. Then the resolution of a first order formula is based on the algorithms that compute the different operations on regular relations represented by PR logic programs.

$$\frac{\top}{\delta(v) = \langle v, true\rangle} \text{ if } v \in Var$$

$$\frac{\delta(t_1) = \langle x_1, e_1\rangle \dots \delta(t_n) = \langle x_n, e_n\rangle}{\delta(f(t_1, \dots, t_n)) = \langle x, \exists x_1 \dots x_n f(x_1, \dots, x_n) \downarrow^?_R x, \bigwedge_{1\leq i\leq n} e_i\rangle}$$
$$\text{if } f \in \mathcal{F} \cup \mathcal{X} \text{ and } x \notin \bigcup_{1\leq i\leq n} Var(e_i)$$

$$\frac{\delta(s) = \langle x_s, e_s\rangle \qquad \delta(t)\langle x_t, e_t\rangle}{\delta(s \downarrow^?_R t) = \exists x_s, x_t x_s \downarrow^?_R x_t \wedge e_s \wedge e_t}$$

Table 2: Flattening second order equations

In the second order case, we cannot compute the set of solutions therefore our aim is to find a particular instance for the second order variables and the corresponding set of solutions for the first order variables. To reach this goal we need to transform the initial formula in two steps: first, we flatten the equations of the formula and then we put the result in prenex disjunctive normal form.

The rules of table 2 are used to transform any second order pseudo regular joinability equation to an equivalent formula which equations are of the form $f(\vec{x}) \downarrow^?_R y$ where $f \in \mathcal{F} \cup \mathcal{X}$ and $\vec{x}, y$ are first order variables.

Lemma 5 *Let R be a pseudo-regular CTRS and $s \downarrow^?_R t$ a second order equation, $SOL(s \downarrow^?_R t) = SOL(\delta(s \downarrow^?_R t))$*

Proof 5 *Let s be a second order term without constructor symbols and $\langle x, e\rangle = \delta(s)$. We prove by induction on the depth of the term s that for all data ground substitution σ of domain $Var(s)$ such that $s\sigma$ is a data ground term, $t \in \mathcal{M}(s\sigma)$ iff $t \in SOL(e)$. If s is a first order variable, it is true since $e = true$ therefore $SOL(e) = \mathcal{T}(\mathcal{C})$.*

Consider now a term s of depth greater than 0, s can be written $f(s_1, \dots, s_n)$. $\delta(s) = \langle x, \exists x_1 \dots x_n f(x_1, \dots, x_n) \downarrow^?_R x, \bigwedge_{1\leq i\leq n} e_i\rangle$ if $\delta(s_i) = \langle x_i, e_i\rangle$. Let σ a ground data substitution of domain $Var(s)$ and $\sigma_i = \sigma|_{Var(s_i)}$ for $1 \leq i \leq n$. By induction hypothesis, we know that $t_i \in \mathcal{M}(s_i\sigma_i)$ iff $s_i \in SOL(e_i)$. Therefore $t \in SOL(\exists x_1 \dots x_n f(x_1, \dots, x_n) \downarrow^?_R x, \bigwedge_{1\leq i\leq n} e_i)$ iff $t \in \tilde{f}(t_1, \dots, t_n)$, thus, iff $t \in \mathcal{M}(f(s_1, \dots, s_n))$.

We consider now a conjunctive pseudo-regular formula F, of the form $fo(\vec{y}) \wedge X_1(\vec{x}_1) \downarrow^?_R x_1 \wedge \dots \wedge X_m(\vec{x}_m) \downarrow^?_R x_m$ where $fo(\vec{y})$ is a conjunction of first order pseudo-regular equations and $\vec{y}$ the vector of the variables occurring in those equations. Les $P_{fo}(\vec{y})$ be the pseudo-regular logic program that represents the set of solutions of $fo(\vec{y})$. Let $SO(F)$ be the following set of clauses $\{ P_{X_i}(\vec{x}_i, x_i) \Leftarrow P_{fo}(\vec{y}) \mid 1 \leq i \leq m \}$ and $\sigma_{SO(F)} = \{X_i \mapsto \tilde{r}_i\}$ where $\tilde{r}_i = \{ (\vec{t}, t) \mid SO(F) \models P_{X_i}(\vec{t}, t) \}$.

Lemma 6 *Let F be a conjunctive pseudo-regular formula, F is satisfiable iff $F\sigma_{SO(F)}$ is satisfiable.*

Proof 6 *It is obvious that if $F\sigma_{SO(F)}$ is satisfiable then F is satisfiable. Now consider that F is satisfiable, this means that $fo(\vec{y})$ is satisfiable, therefore the model of $P_{fo}(\vec{y})$ is not empty. This means that $SO(F)$ instanciates each second order variable of F by a non-empty relation, thus $F\sigma_{SO(F)}$ is satisfiable.*

The following lemma helps to caracterize the instance of the second order variables we compute. It is used to prove that our algorithm gives a solution iff the whole formula is satisfiable. It is also important when one wants to synthetize a CTRS from a pseudo-regular formula.

Lemma 7 *Let $F = fo(\vec{y}) \wedge X_1(\vec{x}_1) \downarrow^?_R x_1 \wedge \dots \wedge X_m(\vec{x}_m) \downarrow^?_R x_m$ be a conjunctive pseudo-regular formula, $\sigma_{SO(F)}$ is the smallest solution of the following second pseudo regular formula $\forall \vec{z} fo(\vec{y}) \Rightarrow X_1(\vec{x}_1) \downarrow^?_R x_1 \wedge \dots \wedge X_m(\vec{x}_m) \downarrow^?_R x_m$ where $\vec{z}$ is a vector composed of the union the variables of $\vec{y}$, of $\vec{x}_i$ and x_i $(1 \leq i \leq m)$.*

Proof 7 *The proof of this lemma is obvious since $\sigma_{SO(F)}$ is computed from the set of clauses $X_i(\vec{x}_i) \Leftarrow fo(\vec{y})$ $(1 \leq i \leq n)$.*

We are now ready to describe the algorithm that decides the satisfiability of a positive pseudo-regular formula and gives one instance of the second order variables and the set of corresponding solutions for the first order variables when the formula is satisfiable.

Let R be a pseudo-regular CTRS and F a positive second-order pseudo-regular formula in prenex disjunctive normal form. $SO(F)$ is the following set of clauses $\{ SO(C) \mid C$ is a conjunctive factor of $F \}$, $\sigma_{SO(F)}$ is the substitution of the second order variables of F defined by $\{X_i \mapsto \tilde{r_i}\}$ where $\tilde{r_i} = \{ (\vec{t}, t) \mid SO(F) \models P_{X_i}(\vec{t}, t) \}$. Finally we call $\mathcal{LP}(F)$ the pseudo-regular logic program equivalent to $SO(F)$ obtained by the transformation of [10].

Algorithm 1 *Let R be a pseudo-regular CTRS and F a positive second-order pseudo-regular formula.*

1. *Compute F' equivalent to F and in prenex disjunctive normal form.*
2. *Compute $\mathcal{LP}(R)$ and $\mathcal{LP}(F')$.*
3. *Solve the pseudo-regular formula $F'\sigma_{SO(F')}$, using the algorithm of [10] considering each second order variable symbols as defined function symbols and $\mathcal{LP}(R) \cup \mathcal{LP}(F')$ as the representation of the relations. The result is a pseudo-regular logic program called $\mathcal{P}_{SOL(F)}$.*
4. *Compute the CTRS $\mathcal{RR}(()\mathcal{P}_{SOL(F)})$.*

Theorem 3 *Let R be a pseudo-regular CTRS and F a positive second-order pseudo-regular formula. $\mathcal{M}(\mathcal{P}_{SOL(F)})$ is not empty iff F is satisfiable.*

Proof 8 *Let us call F' the prenex disjunctive normal form of F. F' is of the form $qtfo_1(\vec{y}_1) \wedge so_1(\vec{x}_1, \vec{X}_1) \vee \ldots fo_n(\vec{y}_n) \wedge so_n(\vec{x}_n, \vec{X}_n)$ where qt are the quantifications of the formula, each $fo_i(\vec{y}_i)$ is a conjunction of first order joinability equations which variables are those of $\vec{y}_i$ and $so_i(\vec{x}_i, \vec{X}_i)$ is a conjunction of flat second order joinability equations of form $X(\vec{x}) \downarrow^?_R x$ where $X \in X_i$ and the variables of $\vec{x}$ and x occur in $\vec{x}_i$.*

$\mathcal{M}(\mathcal{P}_{SOL(F)})$ represents the set of solutions of the formula $F'\sigma_{SO(F')}$, therefore if $\mathcal{M}(\mathcal{P}_{SOL(F)})$ is not empty $F\sigma_{SO(F')}$ is satisfiable thus F is also satisfiable.

If F is satisfiable, F' is also statisfiable therefore $qtfo_1(\vec{y}_1) \vee \ldots fo_n(\vec{y}_n)$ as well as $fo_1(\vec{y}_1) \vee \ldots fo_n(\vec{y}_n)$ have at least one model. This means that $\mathcal{M}(SO(F'))$ is not empty so $\sigma_{SO(F')}$ instanciates second order variables to non empty relations. Moreover we know from Lemma 7 that $\forall \vec{z_i}, fo_i(\vec{y}_i) \Rightarrow so_i(\vec{x}_i, X_i)\sigma_{SO(F')}$ which means that $\forall \vec{z_i}, fo_i(\vec{y}_i) \Rightarrow fo_i(\vec{y}_i) \wedge so_i(\vec{x}_i, X_i)\sigma_{SO(F')}$ hence $qtfo_1(\vec{y}_1) \wedge so_1(\vec{x}_1, \vec{X}_1) \vee \ldots fo_n(\vec{y}_n) \wedge so_n(\vec{x}_n, \vec{X}_n)$ is satisfiable.

Example 6 *Let us consider again the CTRS of Example 1 and the following formula $\exists y,\ X(y + y) \downarrow^?_R y \vee X(s(y + y)) \downarrow^?_R y$*

The flattening of this formula gives $\exists y,\ \exists x_1 (y + y \downarrow^?_R x_1 \wedge X(x_1) \downarrow^?_R y) \vee \exists x_2, x_3 s(x_3) \downarrow^?_R x_2 \wedge y + y \downarrow^?_R x_3 \wedge X(x_2) \downarrow^?_R y)$ which prenex form F' is $\exists y, x_1, x_2, x_3 (y + y \downarrow^?_R x_1 \wedge X(x_1) \downarrow^?_R y) \vee (s(x_3) \downarrow^?_R x_2 \wedge y + y \downarrow^?_R x_3 \wedge \mathcal{X}(x_3) \downarrow^?_R y)$

$SO(F') = \{P_X(x_1) \Leftarrow P_+(y, y, x_1),\ P_X(x_2) \Leftarrow P_s(x_3, x_2), P_+(y, y, x_3)\}$

It can be easily seen that relation defined by P_X in $\mathcal{M}(SO(F'))$ is $\{ (n_1, n_2) \mid n_1/2 = n_2 \}$. For lake of space, we cannot give the CTRS computed by our algorithm.

To specify a relation, we want to synthetize it is important to know Lemma 7. The typical formula to give as input is $\exists \vec{z} fo(\vec{y}) \wedge X(\vec{x}) \downarrow^?_R x$ where $fo(\vec{y})$ is a conjunction of first order equations over the free variables $\vec{y}$ and the variable of $\vec{z}$ are those of $\vec{y}$ union $\vec{x}, x$. Indeed if $fo(\vec{y})$ is satisfiable the computed instance for X is $\forall \vec{z} fo(\vec{y}) \wedge X(\vec{x}) \downarrow^?_R x$. The existential quantification being there only to avoid the generation of the set of solutions of free first order variables.

6 Conclusion

In this paper, we have extended the results of [10] to conditional term rewrite systems. We have also define algorithm to decide the positive second order pseudo-regular formulae. When the formula is satisfiable, the algorithm expresses the instances of second order variables by a CTRS. This result provides a synthesis mecanism of CTRS which can be considered as functional logic programs.

Second order theories with second order variables occurring in the terms have been studied in the context of second order unification (see e.g. [9, 15]). Our joinability equations are unifiability equations when the CTRS is a confluent therefore resolution of pseudo-regular second order equations remains to solve second order unification modulo a CTRS.

The synthesis of programs from a specification has been already investigated. In the context of functional programs for example [6, 1] use term rewriting systems to provide an computational model for functional programming and the specification of the function to be synthetized is a set of equations that can be viewed as a positive conjunctive formulae. Higher order logic has been used in [3] for specification in order to synthesize logic programs but some heuristics are used and the result is partially correct whereas our method is exact (i.e. we obtain a correct instance for second order variables). In most of the cases synthesis of programs (functional or not) use inductive with deductive methods to find partially correct result. As a consequence such methods generate more general program than ours. In our framework, such partial solutions may be generated using some approximations during the computation of the operations on the regular relations.

References

[1] M. Alpuente, D. Ballis, F. J. Correa, and M. Falaschi. Automated Correction of Functional Logic Programs. In P. Degano, editor, *Proc. of the European Symp. on Programming, ESOP 2003*, volume 2618 of *LNCS*, pages 54–68. Springer, 2003.

[2] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, United Kingdom, 1998.

[3] P. Bostrom, H. Idestam-Alquist. Induction of logic programs by example-guided unfolding. *Journal of Logic Programming*, 40:159–183, 1999.

[4] H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications (TATA)*. `http://www.grappa.univ-lille3.fr/tata`, 1997.

[5] D. A. Basin and N. Klarlund. Hardware verification using monadic second-order logic. In *Proceedings of the 7th International Conference On Computer Aided Verification*, volume 939, pages 31–41. Springer Verlag, 1995.

[6] N. Dershowitz and E. Pinchover. Inductive synthesis of equationnal programs. In *Proc. of the Eighth National Conference on Artificial Intelligence*, pages 234–239. AAAI press, 1990.

[7] Georg Gottlob and Christoph Koch. Monadic queries over tree-structured data. In *Proceedings of the 17th IEEE Symposium on Logic in Computer Science*, pages 189 – 202. IEEE Computer Society, 2002.

[8] J. L. Jensen, M. E. Jorgensen, N. Klarlund, and M. I. Schwartzbach. Automatic verification of pointer programs using monadic second-order logic. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 226–236, 1997.

[9] L. Levy and M. Villaret. Linear second-order unification and context unification with tree-regular constraints. In *Proc. of the 11th Int. Conf. on Rewriting Techniques and Applications, RTA'00*, volume 1833 of *LNCS*, pages 156–171. Springer, 2000.

[10] S. Limet and P. Pillot. Solving first order formulae of pseudo-regular theory. In *Proc. 2th Int. Conf. on Thoerical Aspect of Computing (ICTAC'05)*, volume 3091 of *LNCS*, pages 110–124. Springer, 2005.

[11] S. Limet and P. Pillot. On second order formulae of pseudo-regular theory. Technical report, LIFO, Université d'Orléans, 2006.

[12] S. Limet and G. Salzer. Manipulating tree tuple languages by transforming logic programs. In Ingo Dahn and Laurent Vigneron, editors, *Electronic Notes in Theoretical Computer Science*, volume 86. Elsevier, 2003. Accepted for publication in Journal of Automated Reasoning.

[13] S. Limet and G. Salzer. Proving properties of term rewrite systems via logic programs. In V. van Oostrom, editor, *Proc. 15th Int. Conf. on Rewriting Techniques and Applications (RTA'04)*, volume 3091 of *LNCS*, pages 170–184. Springer, 2004.

[14] J.W. Lloyd. *Foundations of Logic Programming*. Springer, 1984.

[15] Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. On equality up-to constraints over finite trees, context unification and one-step rewriting. In *14th International Conference on Automated Deduction*, volume 1249 of *LNAI*, pages 34–48. Springer Verlag, 1997.

[16] J. Thatcher and J. Wright. Generalized finite tree automata theory with an application to a descision problem of second-order logic. *Mathematical System Theory*, 2(1):57–81, 1968.

[17] W. Thomas. *Handbook of Formal Language*, volume 3, chapter 7, pages 389–455. Springer Verlag, 1997.

7 Appendix

The relation $\rightarrow_R^n$ denotes n steps of the rewrite relation. A sequence $t_0 \rightarrow_R t_1 \rightarrow_R \ldots \rightarrow_R t_n$ is called a derivation.

Theorem 4 *1 Let R be a pseudo-regular CTRS, s be a term, and $s \rightsquigarrow \langle s', \mathcal{G} \rangle$. Then $s \rightarrow^* t$ with t data-term holds iff there is a substitution μ such that $t = s'\mu$ and $\mathcal{LP}(R) \models \mathcal{G}\mu$.*

Before proving the theorem in Lemma 8 and 9 we introduce some auxiliary notions. For the body $\mathcal{G}$ of a clause generated by the transformation rules from a rule $l \rightarrow r$, we define

- $O_{\mathcal{G}} = \{ u \mid A_u \in \mathcal{G} \}$
- $O^0_{\mathcal{G}} = \{ u \in O_{\mathcal{G}} \mid u \not< v \text{ for all } v \in O_{\mathcal{G}} \}$
- $O^{i+1}_{\mathcal{G}} = \{ u \in O_{\mathcal{G}} \setminus \bigcup_i O^i_{\mathcal{G}} \mid u \not< v \text{ for all } v \in O_{\mathcal{G}} \setminus \bigcup_i O^i_{\mathcal{G}} \}$
- $\mathit{Height}(\mathcal{G}) = \max\{ i \mid O^i_{\mathcal{G}} \neq \emptyset \}$
- $\mathcal{G}^i = \{ A_u \in \mathcal{G} \mid u \in O^i_{\mathcal{G}} \}$

Note that $O_{\mathcal{G}}$ is the same as $\mathit{PRedPos}(r)$, $\mathit{Height}(\mathcal{G})$ is the maximal number of nested possible redex positions in r, and $O^i_{\mathcal{G}}$ is the set of possible redex positions which are nested below $\mathit{Height}(\mathcal{G}) - i$ redexes. A sequence $t_0 \rightarrow_R t_1 \rightarrow_R \ldots \rightarrow_R t_n$ is called a derivation. In the next definition we note $s \leftarrow s' \Leftarrow C$ if s can be rewrite in s' if it needs all the conditions in C to be satisfied. The relation *cpt* which count the number of necessary rewriting step is recursively defined as follow:

- $\mathit{cpt}(s \rightarrow t \Leftarrow \emptyset) = 1$
- $\mathit{cpt}(s \rightarrow s' \Leftarrow \{\mathit{bigcup}_{s\downarrow^?_R t \in C} s\sigma \rightarrow^* u t\sigma \rightarrow^* u\} \rightarrow^* t) = 1 + \mathit{cpt}(s' \rightarrow^* t) + \mathit{cpt}(\{\mathit{bigcup}_{s\downarrow^?_R t \in C} s\sigma \rightarrow^* u t\sigma \rightarrow^* u\})$.
- $\mathit{cpt}(\{s' \rightarrow^* t\} \cup r) = \mathit{cpt}(s' \rightarrow^* t) + \mathit{cpt}(r)$

Lemma 8 *Let R be a conditionnal term rewriting system, s be a term, and $s \rightsquigarrow \langle s', \mathcal{G} \rangle$. If $\mathcal{LP}(R) \models \mathcal{G}\mu$ for some substitution μ, then $s \rightarrow^* s'\mu$.*

Proof 9 *The unconditionnal part is the same as in [10]. For the conditionnal part we have to prove if $\mathcal{LP}(R) \models \mathcal{G}\mu$ for some substitution μ, then $\exists n$ $\mathit{cpt}(s \rightarrow_R s'\mu) = n$.*

$\mathcal{LP}(R) \models \mathcal{G}\mu$ iff there exists a successful proof tree $\mathcal{T}$ headed by $\mathcal{G}\mu$. We use induction on the height h of the proof trees.

$h = 0$: This means that $s \rightsquigarrow \langle s, \emptyset \rangle$, so s is irreducible so $s \rightarrow^* s$ with 0 steps.

$h > 0$: Let A_u be the chosen atom and $P_f(t_1, \ldots, t_n, t) \Leftarrow \mathcal{B}$ be the top clause used to build the proof tree. Let $f(s_1, \ldots, s_n) \rightarrow r \Leftarrow c$ the conditionnal rewrite rule that produced this clause. $(\mathcal{G} \setminus A_u)\mu \cup \mathcal{B}\sigma$ is the next goal in the proof tree. Let $s'' = s[u \leftarrow \sigma r \Leftarrow \sigma c]$, $s'' \rightsquigarrow \langle s', \mathcal{G} \setminus \{ A_v \mid u \leq v \}) \cup \mathcal{B}\sigma \rangle$. $(\mathcal{G} \setminus \{ A_v \mid u \leq v \})\mu \cup \mathcal{B}\sigma \subseteq (\mathcal{G} \setminus A_u)\mu \cup \mathcal{B}\sigma$ therefore $\mathcal{LP}(R) \models (\mathcal{G} \setminus \{ A_v \mid u \leq v \})\mu \cup \mathcal{B}\sigma$ with a proof of height inferior to h, so $s[u \leftarrow \sigma r \Leftarrow c] \rightarrow^* s'\mu \uplus \sigma = s'\mu$ and no rules have been applied on occurrences of $r\sigma$ brought by σ. Now, let us consider the term $s|_u$. $s|_u \rightsquigarrow \langle x_u, \bigcup_{u \leq v, v \in \mathit{PRedPos}(s|_u)} A_v \rangle$. $\mathcal{LP}(R) \models \{ A_v\mu \mid u \leq v \}$ with a proof of height inferior to h, so $s|_u \rightarrow^* x_u\mu = r\sigma$. Therefore $s \rightarrow^* s[u \leftarrow r\sigma \Leftarrow c\sigma] \rightarrow^* s'\mu$. The whole derivation is data.

For the proof of Lemma 9, we introduce the following notations.

Definition 6 *For a data derivation $s \rightarrow^* t$ we define the term $(s \overset{bas}{\rightarrow}^* t) \setminus u$ as follows:*

- $(s \overset{bas}{\rightarrow}^* t) \setminus u = s|_u$ *if* $\mathit{cpt}(s \rightarrow^* t) = 0$

- *if the derivation is of the form* $s \to s[v \leftarrow r\sigma \Leftarrow c\sigma] \to^* t$, $(s \overset{bas}{\to}^* t) \setminus u = s|_u$ *if* $v < u$ *and* $(s \overset{bas}{\to}^* t) \setminus u = (s[v \leftarrow r\sigma \overset{bas}{\to}^* t) \setminus u$ *otherwise.*

Intuitively, $(s \overset{bas}{\to}^* t) \setminus u$ denotes the term obtained by applying rewrite steps of $s \to^* t$ to $s|_u$.

Lemma 9 *Let R be a conditionnal term rewriting system, s be a term, and $s \rightsquigarrow \langle s', \mathcal{G} \rangle$. If $s \to^* t$ with t data-term then $\mathcal{LP}(R) \models \mathcal{G}\delta$ where $\delta = \{ x_v \mapsto (s \overset{bas}{\to}^* t) \setminus v \mid v \in PRedPos(s) \}$ and $s'\delta = t$.*

Proof 10 *By induction on the number n of necessary rewrite step. For the unconditionnal part and for $(n \leq 1)$ that is the same as in [10]. For the conditionnal part we have to prove if $\exists n\ cpt(s \to_R t) = n$ then $\mathcal{LP}(R) \models \mathcal{G}\delta$ where $\delta = \{ x_v \mapsto (s \overset{bas}{\to}^* t) \setminus v \mid v \in PRedPos(s) \}$ and $s'\delta = t$.*

$n > 0$: $s \to_{[u, l \to r \Leftarrow c, \sigma]} t' \to^m t$. By property of data rewriting $u \in PRedPos(s)$ therefore $u \in O_{\mathcal{G}}$. Let $A_u = P_f(s_1, \ldots, s_n, x_u)$ and $P_f(s'_1, \ldots, s'_n, s'') \Leftarrow \mathcal{B}$ be the clause of $\mathcal{LP}(R)$ produced for $l \to r \Leftarrow c$. We have $t' \rightsquigarrow \mathcal{G} \setminus \{ A_v \mid v > u \} \cup \{ A_v \mid v \in PRedPos(r\sigma) \}$ and $cpt(t' \to^m t) < n$ so $\mathcal{LP}(R) \models (\mathcal{G} \setminus \{ A_v \mid v > u \} \cup \{ A_v \mid v \in PRedPos(r\sigma) \} \cup \{ C_i \mid i \in c \})\delta'$ where $\delta' = \{ x_v \mapsto (t' \overset{bas}{\to}^* t) \setminus v \mid v \in PRedPos(t') \}$. By definition, for all $v \not> u$, $(s \overset{bas}{\to}^* t) \setminus v = (t' \overset{bas}{\to}^* t) \setminus v$, By induction hypothesis,for all $i \in c$,$\mathcal{LP}(R) \models (\{ C_i \mid i \in c \})\delta'$. hence for all $v \not> u$, $x_v\delta' = x_v\delta$ and $\mathcal{LP}(R) \models (\mathcal{G} \setminus \{ A_v \mid v > u \})\delta$.

Now, we have to prove that $\mathcal{LP}(R) \models \{ A_v \in \mathcal{G} \mid v \geq u \}\delta$. By definition, for all $v > u$, $(s \overset{bas}{\to}^* t) \setminus v = s|_v$ so, as in the case $k = 0$, $s_i\delta = s|_{u.i}$ therefore $f(s_1, \ldots, s_n)\delta = l\sigma = f(s'_1, \ldots, s'_n)\sigma$. Moreover $x_u\delta = (s \overset{bas}{\to}^* t) \setminus u = (t' \overset{bas}{\to}^* t) \setminus u$ and $s'' = Irr(r)$ so $(t' \overset{bas}{\to}^* t) \setminus u = Irr(r)[v \leftarrow (s \overset{bas}{\to}^* t) \setminus u|_v, v \in PRedPos(r)]$. Let $\sigma' = \sigma \cup \{x_v \mapsto (s \overset{bas}{\to}^* t) \setminus u|_v\}$. We have $P_f(s_1, \ldots, s_n, x_u)\delta = P_f(s'_1, \ldots, s'_n, s'')\sigma'$ and therefore $\mathcal{G}\delta$ is a logical consequence of $(\mathcal{G} \setminus \{A_u\})\delta \cup \mathcal{B}\sigma'$. $\mathcal{B}\sigma' \subseteq (\{ A_v \mid v \in PRedPos(r\sigma) \})\delta'$ and therefore $\mathcal{LP}(R) \cup \models \mathcal{B}\sigma'$.

It remains to prove that $s'\delta = t$. By induction hypothesis $Irr(t')\delta' = t$. $s' = Irr(s)$ and $s \to_{[u, l \to r \Leftarrow c, \sigma]} t'$. Hence, either $x_u \notin Var(s')$ and in this case $s' = Irr(t')$ or $x_u \in Var(s')$ and in this case $Irr(t') = s'[u \leftarrow Irr(r\sigma) \Leftarrow c\sigma]$. In the first case, since $x_v\delta = x_v\delta'$ for all $v \not> u$, we obtain $s'\delta = Irr(t')\delta' = t$. In the second case, $x_u\delta = (s \overset{bas}{\to}^* t) \setminus u = (t' \overset{bas}{\to}^* t) \setminus u = t|_u$, therefore $s'\delta = Irr(t')\delta'[u \leftarrow x_u\delta] = Irr(t')\delta'[u \leftarrow t|_u] = t[u \leftarrow t|_u] = t$.

Definition 7 *Let R be a term rewrite system and $\mathcal{P}$ be a logic program. We say that $\mathcal{P}$* encodes *$\to^*_R$ if for all terms s and t, the relation $s \to^*_R t$ holds iff for some substitution μ, $\mathcal{P} \models \mathcal{G}\mu$ and $t = s'\mu$ where $s \rightsquigarrow \langle s', \mathcal{G} \rangle$.*

Lemma 10 *2 Let R be a pseudo-regular CTRS then $\mathcal{LP}(()R)$ is a NGSPR logic program.*

Proof 11 *The proof concerning the transformation of the non conditionnal part of the pseudo-regular CTRS by $\rightsquigarrow$ in a NGSPR shared logic program is the same proof as in [10] for the transformation of a pseudo-regular TRS R' in a NG-shared-PR logic program given by $\mathcal{LP}(R')$. Secondly we show that the conditionnal parts add only pseudo-regular atom in the body of the clause. For each $f(x_1, \ldots, x_n) \leftarrow z$ of the conditionnal part we have a define function symbol f. Therefore by applying $\rightsquigarrow$ we create only one atom $P_f(x_1, \ldots, x_n, z)$ wich is added in the boddy of the clause. Futhermore all the variables of $f(x_1, \ldots, x_n) \leftarrow z$ have the same image u by pi and $\rightsquigarrow$ keep the image u by pi for the non conditionnal part hence all the variables of $P_f(x_1, \ldots, x_n, z)$ have the same image u by pi and the clause stay NGSPR.*