Toward a first-order extension of Prolog's unification using CHR

Khalil Djelloul, Thi-Bich-Hanh Dao, Thom Fruehwirth

Toward a First-Order Extension of Prolog's Unification using CHR

A CHR First-Order Constraint Solver Over Finite or Infinite Trees

Khalil Djelloul
University of Ulm
Germany
khalil.djelloul@uni-ulm.de

Thi-Bich-Hanh Dao
Laboratoire d'Informatique
Fondamentale d'Orleans
Orleans, France
Dao@lifo.univ-orleans.fr

Thom Fruehwirth
University of Ulm
Germany
thom.fruehwirth@uni-ulm.de

ABSTRACT

Prolog, which stands for PROgramming in LOGic, is the most widely used language in the logic programming paradigm. One of its main concepts is unification. It represents the mechanism of binding the contents of variables and can be seen as solving conjunctions of equations over finite or infinite trees. We present in this paper an idea of a first-order extension of Prolog's unification by giving a general algorithm for solving any first-order constraint in the theory T of finite or infinite trees, extended by a relation which allows to distinguish between finite and infinite trees. The algorithm is given in the form of 16 rewriting rules which transform any first-order formula φ into an equivalent disjunction ϕ of simple formulas in which the solutions of the free variables are expressed in a clear and explicit way. We end this paper describing a CHR implementation of our algorithm. CHR (Constraint Handling Rules) has originally been developed for writing constraint solvers, but the constraints here go much beyond implicitly quantified conjunctions of atomic constraints and are considered as arbitrary first-order formulas built on the signature of T. We discuss how we implement nested local constraint stores and what programming patterns and language features we found useful in the CHR implementation of our algorithm.

Categories and Subject Descriptors

F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Logic and Constraints Programming*; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*Deduction, Logic Programming*

General Terms

Algorithms, Theory

Keywords

Prolog, Unification, CHR (Constraint Handling Rules), Theory of finite or infinite trees, First-order constraints solving

1. INTRODUCTION

Prolog, which stands for PROgramming in LOGic [17, 4], is the most widely used language in the logic programming paradigm. Prolog is based on first-order predicate calculus over Horn clauses and the execution of a Prolog program is an application of theorem proving by first-order resolution. The fundamental concepts behind Prolog are unification, tail recursion and backtracking.

The core of Prolog, namely Alan Robinson's unification algorithm [16] is a mechanism of binding the contents of variables and can be viewed as a one-time assignment. This operation is represented in Prolog by the symbol "=". For example, the unification $f(g(a)) = f(x)$ is possible by binding the variable x to the term $g(a)$. A. Colmerauer has then incorporated into the first extension of Prolog, known as Prolog II, the unification of infinite terms [3, 5] and has shown that unifications of the form $x = f(x)$ are possible by binding the variable x to the infinite term $f(f(f(...)))$.

One of the most significant innovations in Prolog was done in Prolog III and IV [6, 2] where the concept of unification was replaced by the concept of constraint solving over finite or infinite trees. However, the internal solvers of Prolog III and IV are not able to solve arbitrary quantified first-order constraints over finite or infinite trees.

In this paper, we present two contributions:

(1) *A theoretical contribution*: First of all, we extend the signature of Maher's theory of finite or infinite trees [14] by the relation *finite*(t) which forces the term t to be a finite tree. Then, we extend Maher's axiomatization by two new axioms and show its completeness by giving, not only a decision procedure, but a full first-order constraint solver which for every first-order formula gives an equivalent solved formula in which the solutions of the free variables are expressed in a clear and explicit way. The main ideas behind this solver come from the works of T. Dao [8] on the theory of finite or infinite trees.

(2) *A practical contribution* which consists in a full CHR implementation of our algorithm. **Constraint Handling Rules (CHR)** [11, 12, 20] is a concurrent committed-choice constraint logic programming language consisting of

guarded rules that transform multi-sets of atomic formulas (constraints) into simpler ones until exhaustion. CHR was initially developed for solving constraints, but has matured into a general-purpose concurrent constraint language over the last decade, because it can embed many rule-based formalisms and describe algorithms in a declarative way. Moreover, the clean semantics of CHR facilitates non-trivial program analysis and transformation. The power and complexity of the 16 rules of our solver posed a challenge for CHR and its programmer. We show how to implement nesting of local constraint stores in CHR, as well as the debated negation-as-absence (the correspondence to negation-as-failure in Prolog). We distinguish between operation and data constraints and encode necessary control (phases of the algorithm) in constraints relying on the refined operational semantics of CHR systems.

The paper is organized in four sections followed by a conclusion. This introduction is the first section. In Section 2, we present our extended theory T of finite or infinite trees built on a signature containing not only an infinite set of function symbols, but also a relation *finite*(t) which enables to distinguish between finite or infinite trees. In section 3, we present structured formulas that we call *working formulas* and show some of their properties. We end this section by a general algorithm solving any first-order constraint in T. This algorithm handles our working formulas and is given in the form of 16 rewriting rules. It transforms any first-order formula into a disjunction of simple formulas in which the solutions of the free variables are expressed in a clear and explicit way. The correctness of our algorithm implies the completeness of T. Finally, we present in Section 4 our CHR implementation and show how to implement the complex nested structure of our working formulas as well as the different controls which enable the algorithm to move from one phase to another.

2. THE THEORY T OF FINITE OR INFINITE TREES

2.1 Syntax

We are given once and for all, an infinite countable set $\mathbf{V}$ of *variables* and the set L of *logical* symbols: $=, true, false, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists, (,)$. We are also given once and for all, a *signature* S, i.e. a set of symbols partitioned into two subsets: the set of *function* symbols and the set of *relation* symbols. To each element s of S is linked a non-negative integer called *arity* of s. An n-ary symbol is a symbol with arity n. A 0-ary function symbol is called *constant.*

As usual, an *expression* is a word on $L \cup S \cup V$ which is either a *term*, i.e. of one of the two forms:

$$x,\ f(t_1,\dots,t_n), \tag{1}$$

or a *formula*, i.e. of one of the eleven forms:

$$\begin{gathered} s=t,\ r(t_1,\dots,t_n),\ true,\ false, \\ \neg\varphi,\ (\varphi\wedge\psi),\ (\varphi\vee\psi),\ (\varphi\rightarrow\psi),\ (\varphi\leftrightarrow\psi),(\forall x\,\varphi),\ (\exists x\,\varphi). \end{gathered} \tag{2}$$

In (1), x is taken from $\mathbf{V}$, f is an n-ary function symbol taken from S and the t_i's are shorter terms. In (2), s, t and the t_i's are terms, r is an n-ary relation symbol taken from S and φ and ψ are shorter formulas. The formulas of the first line of (2) are known as *atomic*, and *flat* if they are of one of the following forms: $true$, $false$, $x_0 = x_1, x_0 = f(x_1, ..., x_n)$, $r(x_1, ..., x_n)$, where all the x_i's are possibly non-distinct variables taken from $\mathbf{V}$, f is an n-ary function symbol taken from S and r is an n-ary relation symbol taken from S. We will also use the quantifiers $\exists?$ (at most one) and $\exists!$ (one and only one).

2.2 Axioms

Let F be a set of function symbols containing an infinity of distinct n-ary function symbols which are not constants and at least one constant. Let *finite* be an 1-ary relation symbol. The theory T of finite or infinite trees built on the signature $S = F \cup \{finite\}$ has as axioms the infinite set of propositions of one of the five following forms:

$$\begin{array}{ll}
\forall\bar{x}\forall\bar{y} & \neg(f(\bar{x})=g(\bar{y}))\\
\forall\bar{x}\forall\bar{y} & f(\bar{x})=f(\bar{y})\rightarrow \bigwedge_i x_i=y_i\\
\forall\bar{x}\exists!\bar{z} & \bigwedge_i z_i=t_i[\bar{x}\bar{z}]\\
\forall\bar{x}\forall u & \neg(u=t[u,\bar{x}]\wedge \mathit{finite}(u))\\
\forall\bar{x}\forall u & (u=f(\bar{x})\wedge \mathit{finite}(u))\leftrightarrow(u=f(\bar{x})\wedge\bigwedge_i \mathit{finite}(x_i))
\end{array}$$

where f and g are distinct function symbols taken from F, $\bar{x}$ is a vector of possibly non-distinct variables x_i, $\bar{y}$ is a vector of possibly non-distinct variables y_i, $\bar{z}$ is a vector of distinct variables z_i, $t_i[\bar{x}\bar{z}]$ is a term which begins with an element of F followed by variables taken from $\bar{x}$ or $\bar{z}$, and $t[u, \bar{x}]$ is a term containing at least one occurrence of an element of F and the variable u and possibly other variables taken from $\bar{x}$.

This theory is an extension of the basic theory of finite or infinite trees given by M. Maher in [14] and built on a signature containing an infinite set of function symbols. Maher's theory is composed of the three first axioms of T and its completeness was shown using a decision procedure which transforms each proposition into a Boolean combination of existentially quantified conjunctions of atomic formulas. A more general decision procedure was recently proposed by K. Djelloul in the frame of decomposable theories [9].

3. SOLVING FIRST-ORDER CONSTRAINTS IN T

3.1 Normalized and working formulas

Let us assume that the infinite set $\mathbf{V}$ is ordered by a strict linear dense order relation without endpoints denoted by $\succ$. Starting from this section, we impose the following discipline to every formula φ in T: The quantified variables of φ are renamed so that: (1) The quantified variable of φ have distinct names that are different from those of the free variables. (2) For all variables x, y and all sub-formulas φ_i of φ, if y has a free occurrence in φ_i and x has a bound occurrence in φ_i then $x \succ y$. We show that we can always transform any formula φ into an equivalent formula ϕ, which respects the discipline of the formulas in T, only by renaming the quantified variables of φ.

DEFINITION 3.1.1. *Let $v_1, ..., v_n, u_1, ..., u_m$ be variables. A* basic formula *is a formula of the form*

$$(\bigwedge_{i=1}^{n} v_i = t_i) \wedge (\bigwedge_{i=1}^{m} \mathit{finite}(u_i)) \tag{3}$$

in which all the equations $v_i = t_i$ are flat. Note that if $n = m = 0$ then (3) is reduced to true. The basic formula

(3) is called solved *if all the variables $v_1, ..., v_n, u_1, ..., u_m$ are distinct and for each equation of the form $x = y$ we have $x \succ y$. If α is a basic formula then we denote by: (i) $Lhs(\alpha)$ the set of the variables which occur in the left hand sides of the equations of α. (ii) $FINI(\alpha)$ the set of the variables which occur in a sub-formula of α of the form finite(x).*

Definition 3.1.2. *Let α be a basic formula and $\bar{x}$ a vector of variables. The reachable variables and equations of α from the variable x_0 are those which occur in a sub-formula of α of the form:*

$$x_0 = t_0(x_1) \wedge x_1 = t_1(x_2) \wedge ... \wedge x_{n-1} = t_{n-1}(x_n),$$

where x_{i+1} occurs in the term $t_i(x_{i+1})$. The reachable variables and equations of $\exists\bar{x}\,\alpha$ are those which are reachable in α from the free variables of $\exists\bar{x}\,\alpha$. A sub-formula of α of the form finite(u) is called reachable in $\exists\bar{x}\,\alpha$ if $u \notin \bar{x}$ or u is a reachable variable of $\exists\bar{x}\,\alpha$.

Definition 3.1.3. *A normalized formula φ of depth $d \geq 1$ is a formula of the form $\neg(\exists\bar{x}\,\alpha \wedge \bigwedge_{i\in I} \varphi_i)$, with I a finite (possibly empty) set, α a basic formula and the φ_i's normalized formulas of depth d_i with $d = 1 + \max\{0, d_1, ..., d_n\}$.*

Property 3.1.4. *Every formula φ is equivalent in T to a normalized formula.*

Definition 3.1.5. *A* general solved *formula is a normalized formula of the form $\neg(\exists\bar{x}\,\alpha \wedge \bigwedge_{i=1}^{n} \neg(\exists\bar{y}_i\,\beta_i))$, with $n \geq 0$ and such that:*

1. *α and all the β_i, with $i \in \{1, ..., n\}$, are solved basic formulas.*
2. *If α' is the conjunction of the equations of α then all the conjunctions $\alpha' \wedge \beta_i$, with $i \in \{1, ..., n\}$, are solved basic formulas.*
3. *All the variables of $\bar{x}$ are reachable in $\exists\bar{x}\,\alpha$.*
4. *For all $i \in \{1, ..., n\}$, all the variables of $\bar{y}_i$ are reachable in $\exists\bar{y}_i\,\beta_i$.*
5. *If finite(u) is a sub-formula of α then for all $i \in \{1, ..., n\}$, the formula β_i contains either finite(u), or finite(v) where v is a reachable variable from u in $\alpha\wedge\beta_i$ and does not occur in a left hand side of an equation of $\alpha \wedge \beta_i$.*
6. *For all $i \in \{1, ..., n\}$, the formula β_i contains at least one atomic formula which does not occur in α.*

Property 3.1.6. *Let φ be a general solved formula. If φ has no free variables then φ is the formula $\neg(\exists\varepsilon\,$ true$)$ else neither $T \models \neg\varphi$ nor $T \models \varphi$.*

Definition 3.1.7. *A working formula is a normalized formula in which all the occurrences of $\neg$ are replaced by $\neg^k$ with $k \in \{0, ..., 5\}$ and such that each occurrence of a sub-formula of the form*

$$p = \neg^k(\exists\bar{x}\,\alpha \wedge q), \quad \text{with} \ \ k > 0, \tag{4}$$

***satisfies the** k **first conditions** of the condition list bellow. In (4) α is a basic formula, q is a conjunction of working formulas of the form $\bigwedge_{i=1}^{n} \neg^{k_i}(\exists\bar{y}_i\,\beta_i \wedge q_i)$, with $n \geq 0$, β_i a basic formula, q_i a conjunction of working formulas, and in the below condition list α' is the basic formula of the immediate top-working formula p' of p if it exists.*

1. *If p' exists then $T \models \alpha \rightarrow \alpha'$ and $T \models \alpha_{eq} \rightarrow \alpha'_{eq}$ where α_{eq} and α'_{eq} are the conjunctions of the equations of α respectively α'. Moreover, the set of the variables of $Lhs(\alpha') \cup FINI(\alpha')$ is included in those of $Lhs(\alpha) \cup FINI(\alpha)$.*
2. *The left hand sides of the equations of α are distinct and for all equations of the form $u = v$ we have $u \succ v$.*
3. *α is a basic solved formula.*
4. *If p' exists then the set of the equations of α' is included in those of α.*
5. *The variables of $\bar{x}$, the equations of α and the constraints of the form finite(x) of α are reachable in $\exists\bar{x}\,\alpha$. Moreover, if $n > 0$ then for all $i \in \{1, ..., n\}$ the conjunction β_i contains at least one atomic formula which does not occur in α.*

An initial *working formula is a working formula which begins with $\neg^4$ and such that $k = 0$ for all the other occurrences of $\neg^k$. A* final *working formula is a working formula of depth less or equal to 2 with $k = 5$ for all the occurrences of $\neg^k$.*

Property 3.1.8. *Let p be the final formula $\neg^5(\exists\bar{x}\,\alpha \wedge \bigwedge_{i=1}^{n} \neg^5(\exists\bar{y}_i\,\beta_i)$ The formula $\neg(\exists\bar{x}\,\alpha \wedge \bigwedge_{i=1}^{n} \neg(\exists\bar{y}_i\,\beta_i^*))$ is a general solved formula equivalent to p in T, where β_i^* is the basic formula β_i from which we have removed all the equations which occur also in α.*

3.2 The Rules

We present in Fig 1. the rewriting rules which transform an initial working formula of any depth d to a conjunction of final formulas, equivalent in T. In these rules the letters u, v and w represent variables, the letters $\bar{x}$, $\bar{y}$ and $\bar{z}$ represent vectors of variables, the letters a, b and c represent basic formulas, the letter q represents a conjunction of working formulas, the letter r represents a conjunction of flat equations, formulas of the form *finite*(x) and working formulas. All these letters can be subscripted or have primes. Moreover, $u \succ v$, f and g are two distinct function symbols taken from F. In rule (3), t is a flat term, i.e. either a variable or a term of the form $f(x_1, ..., x_n)$ with f an n-ary function symbol taken from F. In rule (6), the equations of a have distinct left hand sides and for each equation of the form $u = v$ we have $u \succ v$. In rule (9), the variable u is reachable from u in a. In rule (10), the variable u is non-reachable from u in a. Moreover, if f is a constant then $n = 0$. In rule (11), a is a solved basic formula. In rule (13), a and a'' are conjunctions of equations having the same left hand sides, and a' is a conjunction of formulas of the form *finite*(u). In rule (15), $n \geq 0$ and for all $i \in \{1, ..., n\}$ the formula b_i is different from the formula a. The pairs $(\bar{x}', a')$ and $(\bar{y}_i', b_i')$ are obtained by a decomposition of $\bar{x}$ and a into $\bar{x}'\bar{x}''\bar{x}'''$ and $a' \wedge a'' \wedge a'''$ as follows:

- a' is the conjunction of the equations and the formulas of the form *finite*(x) which are reachable in $\exists\bar{x}\,a$.
- $\bar{x}'$ is the vector the variables of $\bar{x}$ which are reachable in $\exists\bar{x}\,a$.
- a'' is the conjunction of the formulas of the form *finite*(x) which are non-reachable in $\exists\bar{x}\,a$.

$$
\begin{array}{cccc}
(1) & \neg^1(\exists\bar{x}\, u = u \wedge r) & \Longrightarrow & \neg^1(\exists\bar{x}\, r) \\
(2) & \neg^1(\exists\bar{x}\, v = u \wedge r) & \Longrightarrow & \neg^1(\exists\bar{x}\, u = v \wedge r) \\
(3) & \neg^1(\exists\bar{x}\, u = v \wedge u = t \wedge r) & \Longrightarrow & \neg^1(\exists\bar{x}\, u = v \wedge v = t \wedge r) \\
(4) & \neg^1(\exists\bar{x}\, u = fv_1...v_n \wedge u = gw_1...w_m \wedge r) & \Longrightarrow & \mathit{true} \\
(5) & \neg^1(\exists\bar{x}\, u = fv_1...v_n \wedge u = fw_1...w_n \wedge r) & \Longrightarrow & \neg^1(\exists\bar{x}\, u = fv_1...v_n \wedge \bigwedge_{i=1}^n v_i = w_i \wedge r) \\
(6) & \neg^1(\exists\bar{x}\, a \wedge q) & \Longrightarrow & \neg^2(\exists\bar{x}\, a \wedge q) \\
(7) & \neg^2(\exists\bar{x}\, \mathit{finite}(u) \wedge \mathit{finite}(u) \wedge r) & \Longrightarrow & \neg^2(\exists\bar{x}\, \mathit{finite}(u) \wedge r) \\
(8) & \neg^2(\exists\bar{x}\, u = v \wedge \mathit{finite}(u) \wedge r) & \Longrightarrow & \neg^2(\exists\bar{x}\, u = v \wedge \mathit{finite}(v) \wedge r) \\
(9) & \neg^2(\exists\bar{x}\, \mathit{finite}(u) \wedge a \wedge q) & \Longrightarrow & \mathit{true} \\
(10) & \neg^2(\exists\bar{x}\, u = f(v_1, ..., v_n) \wedge \mathit{finite}(u) \wedge r) & \Longrightarrow & \neg^2(\exists\bar{x}\, u = f(v_1, ..., v_n) \wedge \bigwedge_{i=1}^n \mathit{finite}(v_i) \wedge r) \\
(11) & \neg^2(\exists\bar{x}\, a \wedge q) & \Longrightarrow & \neg^3(\exists\bar{x}\, a \wedge q) \\
(12) & \neg^4(\exists\bar{x}\, a \wedge q \wedge \neg^0(\exists\bar{y}\, r)) & \Longrightarrow & \neg^4(\exists\bar{x}\, a \wedge q \wedge \neg^1(\exists\bar{y}\, a \wedge r)) \\
(13) & \neg^4(\exists\bar{x}\, a \wedge a' \wedge q \wedge \neg^3(\exists\bar{y}\, a'' \wedge r)) & \Longrightarrow & \neg^4(\exists\bar{x}\, a \wedge a' \wedge q \wedge \neg^4(\exists\bar{y}\, a \wedge r)) \\
(14) & \neg^4(\exists\bar{x}\, a \wedge q \wedge \neg^5(\exists\bar{y}\, a)) & \Longrightarrow & \mathit{true} \\
(15) & \neg^4(\exists\bar{x}\, a \wedge \bigwedge_{i=1}^n \neg^5(\exists\bar{y}_i\, b_i)) & \Longrightarrow & \neg^5(\exists\bar{x}'\, a' \wedge \bigwedge_{i\in K} \neg^5(\exists\bar{y}'_i\, b'_i)^*) \\
(16) & \neg^4\left[\begin{array}{l} \exists\bar{x}\, a \wedge q \wedge \\ \neg^5\left[\begin{array}{l} \exists\bar{y}\, b \wedge \\ \bigwedge_{i=1}^n \neg^5(\exists\bar{z}_i\, c_i) \end{array}\right] \end{array}\right] & \Longrightarrow & \left[\begin{array}{l} \neg^4(\exists\bar{x}\, a \wedge q \wedge \neg^5(\exists\bar{y}\, b)) \wedge \\ \bigwedge_{i=1}^n \neg^4(\exists\bar{x}\bar{y}\bar{z}_i\, c_i \wedge q_0)^* \end{array}\right]
\end{array}
$$

Fig 1. Transformation of an initial working formula into a final working formula.

- $\bar{x}''$ is the vector the variables of $\bar{x}$ which are non-reachable in $\exists\bar{x}\, a$ and do not occur in the left hand sides of the equations of a.
- $\bar{x}'''$ is the vector the variables of $\bar{x}$ which are non-reachable in $\exists\bar{x}\, a$ and occur in the left hand sides of the equations of a.
- a''' is the conjunction of the equations which are non-reachable in $\exists\bar{x}\, a$.
- b_i^* is the formula obtained by removing from b_i the formulas of the form $\mathit{finite}(u)$ which occur also in a''
- $\bar{y}'_i$ is the vector of the variables of $\bar{y}_i\bar{x}'''$ which are reachable in $\exists\bar{y}_i\bar{x}'''\, b_i^*$.
- b'_i is the conjunction of the equations and the formulas of the form $\mathit{finite}(x)$ which are reachable in $\exists\bar{y}_i\bar{x}'''\, b_i^*$.
- $K \subseteq \{1, ..., n\}$ is the set of the indices i such that $i \in K$ if and only if no variable of $\bar{x}''$ occurs in b'_i.
- The formula $\bigwedge_{i\in K} \neg^5(\exists\bar{y}'_i\, b'_i)^*$ is the formula $\bigwedge_{i\in K} \neg^5(\exists\bar{y}'_i\, b'_i)$ in which we have renamed the quantified variables so that they satisfy the discipline of the formulas in T.

In rule (16), $n > 0$ and q_0 is the formula q in which all the occurrences of $\neg^k$ have been replaced by $\neg^0$. The formula $\bigwedge_{i=1}^n \neg^4(\exists\bar{x}\bar{y}\bar{z}_i\, c_i \wedge q_0)^*$ is the formula $\bigwedge_{i=1}^n \neg^4(\exists\bar{x}\bar{y}\bar{z}_i\, c_i \wedge q_0)$ in which we have renamed the quantified variables so that they satisfy the discipline of the formulas of T.

The main idea behind this algorithm consists in (i) a top-down simplification and propagation of constraints. In each level, quantified basic formulas are locally solved by the rules (1)...(11) and propagated to the embedded sub-formulas by rule (12). Finiteness is checked by rule (9), inconsistent sub-formulas are removed by rule (14) and basic formulas are restored by rule (13). (ii) a bottom-up elimination of quantifiers and reduction of the depth of working formulas done by the rules (15) and (16).

Property 3.2.1. *Every repeated application of our rewriting rules on an initial working formula p terminates and produces a conjunction of final formulas equivalent to p in T.*

3.3 The Solving Algorithm

Let p any formula. Solving p in T proceeds as follows:

(1) Transform the formula $\neg p$ (the negation of p) into a normalized formula p_1 equivalent to $\neg p$ in T.

(2) Transform p_1 into the following working formula p_2

$$p_2 = \neg^4(\exists\varepsilon\, \mathit{true} \wedge \neg^0(\exists\varepsilon\, \mathit{true} \wedge p_1)),$$

where all the occurrences of $\neg$ in p_1 are replaced by $\neg^0$.

(3) Apply the preceding rewriting rules on p_2 as many time as possible. According to Property 3.2.1 we obtain at the end a wnfv conjunction p_3 of final working formulas of the form $\bigwedge_{i=1}^n \neg^5(\exists\bar{x}_i\, \alpha_i \wedge \bigwedge_{j=1}^{n_i} \neg^5(\exists\bar{y}_{ij}\, \beta_{ij}))$. According to Property 3.1.8, the formula p_3 is equivalent in T to the following wnfv conjunction p_4 of general solved formulas $\bigwedge_{i=1}^n \neg(\exists\bar{x}_i\, \alpha_i \wedge \bigwedge_{j=1}^{n_i} \neg(\exists\bar{y}_{ij}\, \beta_{ij}^*))$, where β_{ij}^* is the formula β_{ij} from which we have removed all the equations which occur also in α_i. Since p_4 is equivalent to $\neg p$ in T, then p is equivalent in T to $\neg \bigwedge_{i=1}^n \neg(\exists\bar{x}_i\, \alpha_i \wedge \bigwedge_{j=1}^{n_i} \neg(\exists\bar{y}_{ij}\, \beta_{ij}^*))$, which is equivalent to the following disjunction p_5

$$\bigvee_{i=1}^n (\exists\bar{x}_i\, \alpha_i \wedge \bigwedge_{j=1}^{n_i} \neg(\exists\bar{y}_{ij}\, \beta_{ij}^*)).$$

This is the solved form of the formula p where the solutions of the free variables are expressed in a clear and explicit way. Moreover, according to the form of p_2 and using the properties 3.1.6 and 3.1.8, we show that if $T \models p$ then p_5 is the formula $\exists\varepsilon\, \mathit{true}$, and if $T \models \neg p$ then p_5 is the formula *false* (obtained with $n = 0$, i.e. the empty disjunction). We can now present our main theorem, from which we deduce the completeness of T:

Theorem 3.3.1. *Every formula is equivalent in T either to true, or to false, or to a disjunction of general solved formulas which has at least one free variable, which is equivalent neither to true nor to false, and where the solutions of the free variables are expressed in a clear and explicit way.*

4. IMPLEMENTATION OF OUR ALGORITHM

Our CHR implementation was done using Christian Holzbaur's CHR library of Sicstus Prolog 3.11.0. It consists of 18 CHR constraints and 73 CHR rules, almost all of them belonging to computations for the complicated rules (15) and (16) of our algorithm. We will be able to quickly prototype optimizations and variations of our algorithm and to parallelize it. For CHR, the implementation of this complex solver helps to understand what programming patterns and language features can be useful. Indeed for code size and degree of abstraction it seems only possible and interesting to describe the CHR implementation, and we do so in the following. The reader can find our full CHR implementation at `http://khalil.djelloul.free.fr/solver.txt` and can experiment with it online using webchr at `http://chr.informatik.uni-ulm.de/ webchr/`.

4.1 Constraint Handling Rules (CHR) Implementation

4.1.1 Execution of CHR

CHR [11, 12, 20] manipulates constraints that reside in a constraint store. It contains all information necessary for computation. Let H, C and B denote conjunctions of constraints. A simplification rule $H \Leftrightarrow C \mid B$ *replaces* instances of the CHR constraints H by B provided the guard C holds. A propagation rule $H \Rightarrow C \mid B$ instead just *adds* B to H without removing anything. The hybrid simpagation rules will come handy in the implementation: $H_1 \backslash H_2 \Leftrightarrow C \mid B$ removes macthed constraints H_2 but keeps constraints H_1.

The constraints of the store comprise the state of an execution. Starting from an arbitrary *initial store/state (also called goal, query, problem)*, CHR rules are applied exhaustively, until a fixpoint is reached. If new constraints arrive, rules are reconsidered for application. A *final store/state (also called answer, solution)* is one where either no rule application is possible anymore or where the constraints are inconsistent.

In more detail, a rule is *applicable*, if its head constraints are matched by constraints in the current goal one-by-one and if, under this matching, the guard of the rule is logically implied by the constraints in the store. Any of the applicable rules can be applied, and the application cannot be undone, it is committed-choice. Trivial non-termination of propagation rule applications is avoided by applying a propagation rule at most once to the same constraints [1].

As in Prolog, almost all CHR implementations execute queries from left to right and apply rules top-down in the textual order of the program [10]. A CHR constraint in a query can be understood as a procedure that goes efficiently through the rules of the program in the order they are written, and when it matches a head constraint of a rule, it will look for the other, *partner constraints* of the head in the *constraint store* and check the guard until an applicable rule is found. We consider such a constraint to be *active*. If the active constraint has not been removed after trying all rules, it will be put into the constraint store. Constraints from the store will be reconsidered (woken) if newly added constraints constrain variables of the constraint, because then rules may become applicable since their guards now hold.

4.1.2 CHR Constraints

The implementation consists of 18 constraints: 2 main constraints that encode the tree data structure of the working formulas and the atomic formulas, 9 auxiliary constraints that perform reachability analysis, variable renaming and copying of formulas, and 7 constraints that encode execution control information, mainly for rules (15) and (16).

The two main constraints are nf/4 for the negated quantified basic formulas and of/2 for the equations and the relation *finite*. In more detail, `nf(ParentId,Id,Phase,ExVars)` describes a **n**egated quantified basic **f**ormula with its Id and the Id of its parent node, the phase it is in and the list of existentially quantified variables. `Var=FlatTerm of Id` denotes an equation between a variable and a flat term (a variable or a function symbol applied to variables) that belongs to the negated sub-formula with the identifier Id. `finite(U) of Id` denotes the relation *finite*(U) that belongs to the negated sub-formula with the identifier Id.

For identifiers and problem variables, logical variables are used. The main reason is that most CHR implementations index on logical variables and that new logical variables can be introduced by just using them. The order on the problem variables is implemented by using a built-in ordering mechanism of CHR in Sicstus Prolog, that will order the logical variables chronologically in the order of their introduction. This implies that the nf/4 constraints have to be introduced in a top-down manner.

It is easy to represent any working formula φ using conjunctions of nf/4 and of/2 constraints. For that, it is enough to create one nf/4 constraint for each quantified basic formula of φ and to use a conjunction of of/2 constraints to enumerate the atomic formulas linked to each quantified basic formula. An example is given in Fig 2.

4.1.3 CHR Rules

In our algorithm, we can distinguish two main parts that are iterated until a solved normal form is reached: (i) In the first part, given a formula (local store), the equations of its parent are copied into it by rule (12), then the local store is simplified by rules (1) to (5), finiteness is checked by rules (6) to (11), and finally rule (13) replaces certain local equations by their counterpart equations from the parent. (ii) In the second part, rule (14), (15) and (16) apply and change the structure of the formula in a significant way by copying sub-formulas.

In part 1, when we consider the phase labels of the nf/4 CHR constraint, in the initial formula, the top-most formula will have phase 4 and all sub-formulas the level 0. The computation is triggered by the presence of the top-most formula together with a direct sub-formula. When rules (1) to (13) have been applied to it, the sub-formula will also be at phase 4 and can now in turn trigger the simplification of the equations in its direct sub-formulas. This means that in part 1, computation proceeds top-down in the tree repre-

Fig 2. An example of a working formula expressed using the constraints of/2 and nf/4.

$$\neg^4 \left[\begin{array}{l} \exists u\, u = 1 \wedge \\ \left[\begin{array}{l} \neg^0(\exists \varepsilon\, u = s(v)) \wedge \\ \neg^0(\exists w_1\, u = s(w_1) \wedge w_1 = s(v)) \wedge \\ \neg^5(\exists \varepsilon\, v = s(u) \wedge u = 1 \wedge \left[\begin{array}{l} \neg^5(\exists \varepsilon\, v = s(u) \wedge u = 1 \wedge \mathit{finite}(w_1)) \wedge \\ \neg^5(\exists w_3\, v = s(u) \wedge u = 1 \wedge w_2 = s(w_3) \wedge \mathit{finite}(w_3)) \end{array} \right]) \end{array} \right] \end{array} \right].$$

The preceding working formula can be expressed using the following conjunction of constraints:

$$\begin{array}{l} \mathtt{nf(Q,P1,4,[U]),U = 1\ of\ P1,} \\ \mathtt{nf(P1,P2,0,[\,]),U = S(V)\ of\ P2,} \\ \mathtt{nf(P1,P3,0,[W1]),U = S(W1)\ of\ P3,W1 = S(V)\ of\ P3,} \\ \mathtt{nf(P1,P4,5,[\,]),V = S(U)\ of\ P4,U = 1\ of\ P4} \\ \mathtt{nf(P4,P5,5,[\,]),V = S(U)\ of\ P5,U = 1\ of\ P5,finite(W1)\ of\ P5} \\ \mathtt{nf(P4,P6,5,[W3]),V = S(U)\ of\ P6,U = 1\ of\ P6,W2 = S(W3)\ of\ P6,finite(W3)\ of\ P6} \end{array}$$

senting the structure of the nested formulas. At the end of part 1, all formulas will be in phase 4.

The rules of part 1, i.e. (1) to (13), have a rather direct translation into CHR rules.

```
% 1 Locally simplify equations
(1) @ nf(Q,P,1,Xs) \ U=U of P <=> true.
(2) @ nf(Q,P,1,Xs) \ V=U of P <=> gt(U,V) | U=V of P.
(3) @ nf(Q,P,1,Xs), U=V of P \ U=G of P <=> gt(U,V) | V=G of P
(4) @ nf(Q,P,1,Xs), U=F of P,  U=G of P <=>
          notsamefunctor(F,G) | P=true.
(5) @ nf(Q,P,1,Xs), U=F of P \ U=G of P <=>
          samefunctor(F,G) |  same_args(F,G,P).
% 1-2 enter next phase after solving
(6) @ nf(Q,P,1,Xs) <=> nf(Q,P,2,Xs).

% rules for finite/2
(7) @ nf(P0,P,2,Xs), finite(U) of P \ finite(U) of P <=> true.
(8) @ nf(P0,P,2,Xs), U=V of P \ finite(U) of P <=>
          var(V) | finite(V) of P.
(9+10)@nf(P0,P,2,Xs),U=T of P \ finite(U) of P <=>
          nonvar(T) |   reach_args(U,T,P), finite_args(U,T,P).

% 2-3 enter next phase after finiteness check
(11) @ nf(Q,P,2,Xs) <=> nf(Q,P,3,Xs).

% 4/0-4/1 copy down before solving
(12a) @ nf(Q,P,4,Xs), A of P, nf(P,P1,0,Ys) ==> A of P1.
(12b) @ nf(Q,P,4,Xs)        \ nf(P,P1,0,Ys) <=> nf(P,P1,1,Ys).

% 4/3-4/4 replace down after solving
(13a) @ nf(Q,P,4,Xs),U=V of P, nf(P,P1,3,Ys)\ U=G of P1 <=>
      V\==G | U=V of P1.

(13b)@   nf(Q,P,4,Xs) \ nf(P,P1,3,Ys)   <=> nf(P,P1,4,Ys).
```

Note that rules (1) to (5) are a direct implementation of an equation solver for flat rational trees [12, 15]. These rules are similar to the more general rational tree solver that is a classical example for CHR code. By applying results of [15], we can show that the worst-case time complexity of the rules in the first part of the algorithm is quadratic in the size of the equations in the local formula.

In the rules (2) and (3), the predicate `gt(U,V)` is used to check if we have $\mathtt{U} \succ \mathtt{V}$. In the rules (4) and (5), the predicates `samefunctor(F,G)` and `notsamefunctor(F,G)` are used to check if the terms F and G start by same or distinct function symbols. Note also that in rule (4) we let `P=true`. Binding the identifier `P` removes all equations from the local store associated with it using a simplification rule `E of true <=> true`. Instead of relying on Prolog's built-in syntactic equality `=`, we could also have used a CHR constraint, say `remove(P)` to the same effect.

In rule (9+10) `reach_args(U,T,P)` checks reachability of `U` from itself in `P`. If so, `P` wil be set to `true` and thus the local store will be removed, implementing rule (9). Otherwise, if `P` is still free, the subsequent `finite_args(U,T,P)` will propagate down the `finite` relation from `U` to its arguments, implementing rule (10).

Part 2, on the contrary to part 1, proceeds bottom-up, starting at the leaves (with $n = 0$ in rule (15)). Leaf formulas go from phase 4 to 5. Each formula with a phase 4 formula as the top one and only phase 5 formulas as sub-formulas are rewritten in part 2 (with $n \neq 0$ in rule (15)) into formula with phase 5 formula as top-formula and only phase 5 formulas as sub-formulas. In rule (16), formulas may be copied and the new formulas are introduced with phase 4 and 0 (all the n$\neg^k$ of q are changed into $\neg^0$) which causes a local application of part 1 of the algorithm and so on. The phase change continues going up in the tree structure of the working formulas until all working formulas are in phase 5 and have a depth less or equal to 2.

A more or less direct translation of rules of the algorithm was sufficient so far, but it is by far not sufficient for the rules (14) to (16) of part 2 of the algorithm. For rule (14):

$$\neg^4(\exists \bar{x}\, a \wedge q \wedge \neg^5(\exists \bar{y}\, a)) \Longrightarrow \mathit{true},$$

the implementation is easy when nested negation-as-absence is used to verify that there is no constraint in the sub-formula that is not in the main formula. Negation-as-absence can be directly encoded in CHR implementations, but then it requires two additional rules per negation. Instead, we have chosen to use the CHR library built-in `find_constraint(Var,Pattern,Match)` of the Sicstus-Prolog CHR library that returns on backtracking all constraints `Match` that match the `Pattern` and that are indexed on the variable `Var` together with negation-as-failure provided by the Prolog built-in \+.

```
4/5-true trivial satisfaction-each A of P1  occurs as A of P
(14) @ nf(Q,P,4,Xs), nf(P,P1,5,Ys) <=>
                \+ (find_constraint(P1,(A of P1),_),
                \+ find_constraint(P,(A of P),_) )
                | P=true.
```

For the rules (15) and (16), all primed expressions have to be computed, some quantified variables must be renamed and many nf/4 constraints and their equations should be copied and updated. Simpagation rules and auxiliary constraints collect the nested nf/4 constraints, compute the reachable variables and atomic formulas, rename the quantified variables and produce updated nf/4 and of/2 constraints. This is by far the most complicated and extensive part of our CHR implementation which was finally done using 40 CHR rules. In order no to overburden the reader with technical details, we omit the description of those 40 rules in this article.

5. CONCLUSION

We presented in this paper two contributions : (1) A general algorithm solving any first-order constraint in an extended theory T of finite or infinite trees. The algorithm is in the form of 16 rewriting rules and its correctness implies the completeness of T. (2) A full CHR implementation of our algorithm. We discussed programming patterns for maintaining a hierarchy of local stores and for control using phases by constraints, and desirable program features like the debated negation-as-absence. Simpagation rules proved useful to perform iterations that modify the constraint store.

A. Colmerauer and T. Dao [7] have shown that the complexity of all algorithms solving first-order constraints over finite or infinite trees has a non-elementary complexity. This is why we have used two strategies in the algorithm: a top down propagation of constraints and a bottom-up elimination of quantifiers and distribution. This technique enables us to remove the inconsistent working formulas and to not lose time with solving huge working formulas which contradict their top-working formulas.

Future work will aim at an even more declarative implementation that is directly amenable to established CHR analysis techniques for termination, complexity and confluence properties. We think that we can minimize the use of the debated negation-as-absence [18] by introducing reference counters for the two main constraints. This should also give us the possibility to get a parallel implementation that is derived from the existing one with little modification, similar to what has been done for parallelizing the union-find algorithm in CHR [13].

So far, the CHR implementation has not been optimized. To improve efficiency we could enforce the data-only nature of the equations (the constraint of/2) by declaring it passive in each rule. This compiler pragma of CHR in Sicstus Prolog avoids to generate code for the occurrence of a passive constraint in the head of the rule. Another direction would be to enable input and output of formulas in latex, based on the work of literate CHR. This will enable us to test easily our algorithm on big examples and compare the performances with those obtained using a decision procedure for decomposable theories [9].

6. REFERENCES

[1] Abdennadher, S. 1997. Operational Semantics and Confluence of Constraint Propagation Rules. In Proc of the third International Conference on Principles and Practice of Constraint Programming. LNCS 1330.

[2] Benhamou, F., Colmerauer, A., Garetta, H., Pasero, R. and Van-caneghem, M. 1996. Le manuel de Prolog IV. PrologIA, Marseille, France.

[3] Colmerauer, A. 1982. Prolog and infinite trees. In K.L. Clark and S-A. Tarnlund, editors, Logic Programming. Academic Press. pp. 231–251.

[4] Colmerauer, A., Kanoui, H. and Van-caneghem,M. 1983. Prolog, Theoretical Basis and Current Developments. TSI 2(4):271–311.

[5] Colmerauer, A. 1984. Equations and inequations on finite and infinite trees. Proceeding of the International conference on the fifth generation of computer systems, pp. 85–99.

[6] Colmerauer, A. 1990. An introduction to Prolog III. *Communication of the ACM*, 33(7):68–90.

[7] Colmerauer, A. and Dao, T. 2003. Expressiveness of full first-order formulas in the algebra of finite or infinite trees, Constraints, 8(3): 283–302.

[8] Dao, T. 2000. Resolution de contraintes du premier ordre dans la theorie des arbres finis ou infinis. These d'informatique, Universite de la mediterranee, France.

[9] Djelloul, K. 2006. Decomposable theories. Journal of Theory and practice of logic programming TPLP. (to appear)

[10] Duck, G., Stuckey, P., Banda, M. and Holzbaur, C. 2004. The Refined Operational Semantics of Constraint Handling Rules. In Proc of the 20th International Conference on Logic Programming. LNCS 3132, pp. 105-119.

[11] Fruehwirth, T. 1998. Theory and Practice of Constraint Handling Rules. Special Issue on Constraint Logic Programming. Journal of Logic Programming. 37(1–3): 95-138.

[12] Fruehwirth, T. and Abdennadher, S. 2003. Essentials of Constraint Programming. Springer.

[13] Fruehwirth, T. 2005. Parallelizing Union-Find in Constraint Handling Rules Using Confluence. In proc of the 21st International Conference of Logic Programming. LNCS, Vol 3668. pp: 113-127.

[14] Maher M. Complete axiomatization of the algebra of finite, rational and infinite trees. *Technical report, IBM*, 1988.

[15] Meister, M. and Fruehwirth, T. 2006. Complexity of the CHR Rational Tree Equation Solver. In Proc of the third Workshop on Constraint Handling Rules.

[16] Robinson, J.A. 1965. A machine-oriented logic based on the resolution principle. JACM, 12(1):23–41.

[17] Roussel, F. 1975. Prolog : Manuel de Reference et d'Utilisation, Groupe d'Intelligence Artificielle, Marseille-Luminy.

[18] Van Weert, P., Sneyers, J., Schrijvers, T. and Demoen, B. 2006. Constraint Handling Rules with Negations as Absence. In Proc of the third Workshop on Constraint Handling Rules.

[19] Schrijvers, T., Demoen, B., Duck, G., Stuckey, P. and Fruehwirth, T. 2006. Automatic implication checking for CHR constraints. In Proc of the 6th International Workshop on Rule-Based Programming. ENTC, vol 147, pp. 93-111.

[20] Schrijvers, T. and Fruehwirth. CHR Website, www.cs.kuleuven.ac.be/ dtai/projects/CHR/

Acknowledgments Khalil Djelloul thanks the DFG research project GLOB-CON for funding and supporting his research. Particular thanks also to Marc Meister and Hariolf Betz for their kind reviews of this article.