Complexity of a CHR Solver for Existentially Quantified Conjunctions of Equations over Trees

Marc Meister, Khalil Djelloul[*], and Thom Frühwirth

Fakultät für Ingenieurwissenschaften und Informatik
Universität Ulm, Germany

Abstract. Constraint Handling Rules (CHR) is a concurrent, committed-choice, rule-based language. One of the first CHR programs is the classic constraint solver for syntactic equality of rational trees that performs unification. We first prove its exponential complexity in time and space for non-flat equations and deduce from this proof a quadratic complexity for flat equations. We then present an extended CHR solver for solving existentially quantified conjunctions of non-flat equations in the theory of finite or infinite trees. We reach a quadratic complexity by first flattening the equations and introducing new existentially quantified variables, then using the classic solver, and finally eliminating particular equations and quantified variables.

1 Introduction

Constraint Handling Rules (CHR) [3, 5, 14] is a concurrent committed-choice constraint logic programming language consisting of guarded rules that transform multi-sets of constraints (atomic formulas) into simpler ones until they are solved. CHR was initially developed for writing constraint solvers, but has matured into a general-purpose concurrent constraint language over the last decade. Its main features are a kind of multi-set rewriting combined with propagation rules. The clean logical semantics of CHR facilitates non-trivial program analysis and transformation.

One of the first CHR programs is the classic constraint solver for syntactic equality of rational trees (RT) that performs unification [3, 5]. Unification is concerned with making first order logic terms syntactically equivalent by substituting terms for variables. For example, the terms $h(a, f(Y))$ and $h(Y, f(a))$ can be made syntactically equivalent by substituting the constant a for the variable Y. In 1930, Herbrand [6] gave an informal description of a unification algorithm. Robinson [12] rediscovered a similar algorithm when he introduced the resolution procedure for first-order logic in 1965. Since the late 1970s, there are quasi-linear time algorithms for unification. For finite trees (Herbrand terms), see [9] and [11]. For rational trees, see [7]. These algorithms can be considered as extensions of the union-find algorithm [15] from constants to trees.

[*] Funded by the DFG research project GLOB-CON.

Contributions. In this paper we present two new contributions:

(1) The classic RT solver relies on a term order and its complexity (in time and space) was an open problem for a decade. We first prove its *exponential time and space complexity* for non-flat equations using any term order[1] and then show its *quadratic time and space complexity* for flat equations, i.e. each equation contains at most one function symbol.

(2) We show that *existentially quantified conjunctions of non-flat equations*[2] *can be solved in Maher's theory $\mathcal{T}$ of finite or infinite trees [8] with a quadratic complexity* using an extension of this classic RT solver as well as a notion of *reachable variables and equations.* To the best of our knowledge, this is the first CHR solver for existentially quantified conjunctions of non-flat equations in $\mathcal{T}$ with *quadratic complexity.*

Organisation of the Paper. We recall the basics of Constraint Handling Rules (CHR) and present Maher's theory $\mathcal{T}$ of finite or infinite trees [8] in Section 2.

In Section 3 we first introduce the CHR rules of the classic RT solver which is parametrised by an order of terms and prove its termination for any term order. We then show exponential worst-case time and space complexity of RT for any term order in the case of non-flat equations. We end this section showing its quadratic complexity for flat equations.

Finally, in Section 4, we extend the classic RT solver, so that it can solve in $\mathcal{T}$ existentially quantified conjunctions of non-flat equations in quadratic complexity. For that, we first show that any existentially quantified conjunction of non-flat equations can be transformed into an equivalent existentially quantified conjunction of flat equations in linear time and space complexity. For example, the formula $\exists X\, h(X, f(Y))\ \texttt{eq}\ h(Y, f(X))$ is equivalent in $\mathcal{T}$ to $\exists ABCX\, A\ \texttt{eq}\ h(X,B) \wedge B\ \texttt{eq}\ f(Y) \wedge A\ \texttt{eq}\ h(Y,C) \wedge C\ \texttt{eq}\ f(X)$. Then, we define the notion of reachable variables and equations and use it to remove particular quantified variables and equations. This extension is done using only a few CHR rules.

2 Preliminaries

Readers familiar with CHR and the theory of finite or infinite trees can skip this section.

2.1 Constraint Handling Rules

Constraint Handling Rules (CHR) [3, 5, 14] is a concurrent, committed-choice, rule-based logic programming language. We distinguish between two different kinds of constraints: *built-in (pre-defined) constraints* which are solved by a given

[1] We pay the elegance of the solver, which consists of just four rules and so is more concise than most formal specifications of unification, by an exponential complexity.

[2] For example, the equation $\exists X\, h(X, f(Y))\ \texttt{eq}\ h(Y, f(X))$ is an existentially quantified non-flat equation with the free variable Y.

constraint solver and *CHR (user-defined) constraints* which are defined by the rules in a CHR program. This distinction allows one to embed and utilise existing constraint solvers as well as side-effect-free host language statements. Built-in constraint solvers are considered as black-box whose behaviour is trusted and that do not need to be modified or inspected.

Definition 1. *There are two main kinds of rules:*

$$\begin{array}{ll} \textit{Simplification rule} & \textit{Name} \ @\ H \Leftrightarrow G \mid B \\ \textit{Propagation rule} & \textit{Name} \ @\ H \Rightarrow G \mid B \end{array}$$

Name is an optional, unique identifier of a rule, the head H *is a non-empty conjunction of CHR constraints, the* guard G *is a conjunction of built-in constraints, and the* body B *is a goal. A* goal (query, problem) *is a conjunction of built-in and CHR constraints. A trivial guard expression "true |" can be omitted from a rule.*

Since we do not use propagation rules in this paper, it suffices to say that they are equivalent (in the standard semantics) to a simplification rule of the form *Name* @ $H \Leftrightarrow G \mid (H \wedge B)$.

The *standard operational semantics* of CHR is given by a transition system where states are conjunctions of constraints. To the constraints in the store, rules are applied until a fix-point is reached. Note that conjunctions in CHR are considered as *multi-sets* of atomic constraints. Any rule that is applicable can be applied and rule application cannot be undone since CHR is a committed-choice language. A simplification rule $H \Leftrightarrow G \mid B$ is applicable in state $(H' \wedge C)$, if the built-in constraints C_b of C imply that H' matches the head H and imply the guard G (cf. Figure 1).

$$\begin{array}{ll} \text{IF} & H \Leftrightarrow G \mid B \text{ is a copy of a rule } H \Leftrightarrow G \mid B \text{ with new variables } \bar{X} \\ \text{AND} & CT \models \forall (C_b \rightarrow \exists \bar{X} (H = H' \wedge G)) \\ \text{THEN} & (H' \wedge C) \rightarrowtail (B \wedge G \wedge H = H' \wedge C) \end{array}$$

Fig. 1. State transition for simplification rules

If applied, a simplification rule *replaces* the matched CHR constraints in the state by the body of the rule. The number of rule applications in a computation is called *derivation length*. A computation terminates in the final state when the constraint store becomes inconsistent or no rule is applicable[3].

2.2 Theory $\mathcal{T}$ of Finite or Infinite Trees

The theory $\mathcal{T}$ of finite or infinite trees, which is built on an signature containing an infinite set $\mathcal{F}$ of distinct function symbols, has as axioms the infinite set of propositions of one of the three following forms:

[3] To avoid trivial non-termination, propagation is applied to the same constraints only once.

$$\begin{array}{lll}\forall \bar{X} \forall \bar{Y} & \neg(f(\bar{X}) \ \texttt{eq} \ g(\bar{Y})) & [\mathrm{A1}] \\ \forall \bar{X} \forall \bar{Y} & f(\bar{X}) \ \texttt{eq} \ f(\bar{Y}) \rightarrow \bigwedge_i X_i \ \texttt{eq} \ Y_i & [\mathrm{A2}] \\ \forall \bar{X} \exists! \bar{Z} & \bigwedge_i Z_i \ \texttt{eq} \ T_i[\bar{X}\bar{Z}] & [\mathrm{A3}]\end{array}$$

where f and g are distinct function symbols taken from $\mathcal{F}$, $\bar{X}$ 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, and $T_i[\bar{X}\bar{Z}]$ is a term which begins with an element of $\mathcal{F}$ followed by variables taken from $\bar{X}$ or $\bar{Z}$.

The forms [A1], [A2], and [A3] are also called *schemas of axioms* of the theory $\mathcal{T}$. Proposition [A1] – called *conflict of symbols* – shows that two distinct operations produce two distinct individuals. Proposition [A2] – called *explosion* – shows that the same operation on two distinct individuals produces two distinct individuals. Proposition [A3] – called *unique solution* – shows that for a particular form of conjunction of equations, a unique set of solutions exists in $\mathcal{T}$, e.g. the formula $\exists Z \, Z = f(Z)$ has a unique solution which is the infinite tree $f(f(f(...)))$.

Maher has axiomatised the theory $\mathcal{T}$ and shown its completeness using a decision procedure which transforms any first-order formula into a Boolean combination of quantified conjunctions of atomic formulas [8]. A more general decision procedure was recently given by Djelloul in the frame of decomposable theories [2]. Maher has also shown that the structure of finite or infinite trees and the structure of the rational trees are models of $\mathcal{T}$. A rational tree is a finite or infinite tree whose set of subtrees is finite, e.g. the infinite tree $f(f(f(...)))$ is rational as its set of subtrees $\{f(f(f(...)))\}$ is finite.

Note that $\mathcal{T}$ does not accept full elimination of quantifiers. For example, in the formula $\exists X \, Y \ \texttt{eq} \ f(X)$ we cannot remove or eliminate the quantifier $\exists X$. This is due to the fact that for each model M of $\mathcal{T}$ there exist instantiations $\hat{Y}$ of the free variable Y which satisfy the instantiated formula $\exists X \, \hat{Y} \ \texttt{eq} \ f(X)$ (for example $f(1)$) and others which contradict the instantiated formula $\exists X \, \hat{Y} \ \texttt{eq} \ f(X)$ (for example $g(1)$). As a consequence, the formula $\exists X \, Y \ \texttt{eq} \ f(X)$ is neither true nor false in $\mathcal{T}$ and the quantifier $\exists X$ cannot be eliminated. This makes solving existentially quantified conjunctions of equations non-trivial. We show in Section 4, using the notion of *reachable variables*, how to detect whereas a quantification can be eliminated.

3 Rational Tree Equation Solver

The CHR rational tree equation solver (RT solver) given in Subsection 3.1 is one of the first published CHR programs. However, this elegant solver depends on an order of terms. In Subsection 3.2 we first show its termination for any term order. We then prove its exponential worst-case time and space complexity for any term order in the case of non-flat equations (cf. Subsection 3.3), which was an open problem for a decade, and deduce from this proof its quadratic complexity for flat equations in Subsection 3.4.

3.1 The CHR Rational Tree Solver

The CHR program in Figure 2 solves rational tree equations [3, 5]. This solver dates back to late 1993 and was revised in 1998 [14]. The underlying algorithm is similar to the one in [1], but unlike this and most other unification algorithms it uses variable elimination (substitution) only in a very limited way, if it cannot be avoided. As a consequence, the algorithm has to rely on an order on terms for termination. However, this makes termination and complexity analysis considerably harder.

```
reflexivity   @ X eq X           <=> var(X) | true.
orientation   @ T eq X           <=> var(X), X≺T | X eq T.
decomposition @ T1 eq T2         <=> nonvar(T1), nonvar(T2) |
                                     same_functions(T1,T2).
confrontation @ X eq T1, X eq T2 <=> var(X), X≺T1, T1⪯T2 |
                                     X eq T1, T1 eq T2.
```

Fig. 2. CHR Rational tree equation solver (RT solver)

We describe the RT solver where T, T_1, T_2 are meta-variables that range over arbitrary terms: *Auxiliary* built-ins allows the solver to be independent of the representation of terms. Besides *true* and *false*, we have var(T) iff T is a variable and nonvar(T) iff T is a *function term*. We rely on a total pre-order $\preceq$ on terms[4] which fulfils three properties (defined in Subsection 3.2). As usual, we write $T_1 \prec T_2$, iff $T_1 \preceq T_2$ and $T_2 \not\preceq T_1$. The auxiliary same_functions(T_1, T_2) leads to *false* if T_1 and T_2 have not the same function symbol and the same arity (this is called clash), otherwise a constraint `lists2eq`(L_1, L_2) pairwise equates the lists of arguments L_1 and L_2 of the two terms using a simple recursion:

```
lists2eq([HL1|TL1],[HL2|TL2]) <=> HL1 eq HL2, lists2eq(TL1,TL2).
lists2eq([],[])               <=> true.
```

We now explain application of each *CHR rule* of the solver:

`reflexivity` removes trivial equations between identical variables.

`orientation` reverses the arguments of an equation so that the (smaller) variable comes first.

`decomposition` applies to equations between two function terms. When there is a clash, same_functions leads to *false*. Otherwise, the initial equation is replaced by equations between the corresponding arguments of the terms.

`confrontation` replaces the variable X in the second equation X `eq` T_2 by T_1 from the first equation X `eq` T_1. It performs a limited amount of variable elimination (substitution) by only considering the l.h.s.' of equations. This rule duplicates the term T_1 and the guard makes sure that T_1 is not larger than T_2.

[4] A pre-order is reflexive and transitive, however it may not be antisymmetric, i.e. from $T_1 \preceq T_2$ and $T_2 \preceq T_1$ we cannot conclude that T_1 and T_2 are equal. A pre-order becomes an order on the classes of indifferent terms w.r.t. $\preceq$.

Due to the `confrontation` rule, the complexity of the solver is worse than linear. The intricate interaction between the `decomposition` rule and the `confrontation` rule in the case of infinite terms (cyclic terms) makes it hard to prove termination (cf. Subsection 3.2) and to determine the worst-case time complexity of the solver (cf. Subsection 3.3).

Example 1. Here is the derivation for a simple example involving infinite rational trees that shows that one of the equations is redundant, for $\texttt{X} \prec \texttt{f(X)} \prec \texttt{f(f(X))}$.

$$\begin{array}{ll}
 & \underline{\texttt{X eq f(X)}},\ \underline{\texttt{X eq f(f(X))}} \\
\rightarrowtail_{\texttt{confrontation}} & \texttt{X eq f(X)},\ \underline{\texttt{f(X) eq f(f(X))}} \\
\rightarrowtail_{\texttt{decomposition}}\rightarrowtail^{*} & \underline{\texttt{X eq f(X)}},\ \underline{\texttt{X eq f(X)}} \\
\rightarrowtail_{\texttt{confrontation}} & \texttt{X eq f(X)},\ \underline{\texttt{f(X) eq f(X)}} \\
\rightarrowtail_{\texttt{decomposition}}\rightarrowtail^{*} & \texttt{X eq f(X)},\ \underline{\texttt{X eq X}} \\
\rightarrowtail_{\texttt{reflexivity}} & \texttt{X eq f(X)}
\end{array}$$

3.2 Term Order and Termination

We define a generic *term order* $\preceq$ and prove our conjecture from [10] that the RT solver terminates when used with $\preceq$.

Definition 2 (Term order). *A* term order $\preceq$ *is a total pre-order on terms which has the following three properties*[5]*:*
(i) *For different variables X and Y, either $X \prec Y$ or $Y \prec X$.*
(ii) *Any variable is smaller than any function term.*
(iii) *Subterms are smaller than the terms that properly contain them.*

This term order subsumes orders found in the literature, e.g. *term-size order* which is based on *term size.*

Definition 3 (Term size and term-size order). *The* term size $\#T$ *of a term T is the number of occurrences of variables and function symbols. A* term-size order $\preceq_s$ *must respect properties (i) and (ii) of Definition 2 and is based on term size by $S \preceq_s T$ iff $\#S \leq \#T$ for two function terms.*

Similarly to a term-size order $\preceq_s$, we can define a *term-depth order* $\preceq_d$ which is based on the nesting depths of the terms. Note that term-size and term-depth order are not compatible, e.g. $f(f(a)) \prec_s f(a,a,a)$ in term-size order but $f(a,a,a) \prec_d f(f(a))$ in term-depth order. In previous work [10] we introduced the *term-measure order* $\preceq_m$ which is also not compatible to term-size order $\preceq_s$.

A conjunction of atomic constraints is *solved* (or in *solved normal form*) if it is either *false* or if it is of the form $\bigwedge_{i=1}^{n} X_i\ \texttt{eq}\ T_i$ with pairwise distinct variables $X_1, \ldots, X_n$ and arbitrary terms $T_1, \ldots, T_n$ for $n \in \mathbf{N}$. We require X_i to be different to T_j for $1 \leq i \leq j \leq n$, i.e. if a variable occurs on the l.h.s. of an equation, it does neither occur as its r.h.s. nor as the l.h.s. or r.h.s. of

[5] We write $T_1 \prec T_2$, i.e. term T_1 is *smaller than* term T_2, iff $T_1 \preceq T_2$ and $T_2 \not\preceq T_1$. Recall, that a term that is not a variable is a *function term.*

any subsequent equation. By Definition 2, we can restate the conditions for the solved normal form to $X_i \prec X_{i+1}$ (for $1 \leq i < n$) and $X_i \prec T_i$ (for $1 \leq i \leq n$).

The RT solver computes the solved form, as can be shown by contradiction: As long as a conjunction of constraints is not in solved from, at least one rule is applicable. If it is in solved form, no rule is applicable.

We now show termination of the RT solver for any term order.

Theorem 1 (Termination). *The derivation length of the RT solver, used with any term order $\preceq$, and for any given conjunction of equations is finite.*

Proof. We abstract constraints into five disjunct sorts and study the effects of rule applications.

Sort `bi` for the built-ins $false$ or $true$,
Sort `vv` for equations X `eq` Y with two variables X and Y,
Sort `vt` for equations X `eq` T with variable X and function term T,
Sort `tv` for equations T `eq` X with function term T and variable X, and
Sort `tt` for equations T_1 `eq` T_2 with two function terms T_1 and T_2.

We give the *sort transition graph* of the RT solver in Figure 3: Each arrow visualises the effect of a rule application by removing one equation of a given sort and introducing constraints of other sorts. As the solved normal form contains only constraints of the Sorts `bi`, `vv`, or `vt` we indicate this with doubled-rimmed boxes. The built-in constraints of Sort `bi` are treated by the host language, they are never removed, and there is no arrow from Sort `bi`. We study the effects of each rule application in turn.

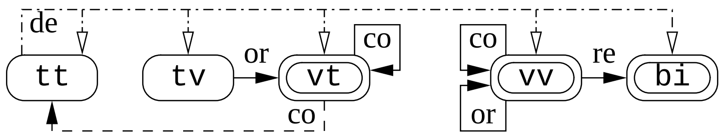

Fig. 3. Sort transition graph for the RT solver for non-flat terms

Application of `reflexivity` replaces the redundant equation X `eq` X by *true*. We visualise this by the arrow from Sort `vv` to Sort `bi`, labelled with **re**, in the right part of Figure 3.

Application of `orientation` is possible to equations of two different sorts and hence two arrows are labelled with **or**. First, an equation of Sort `tv` can be changed into an equation of Sort `vt`, i.e. T `eq` $X \rightarrowtail X$ `eq` T for a variable X and a function term T. Second, when replacing Y `eq` X by X `eq` Y for two variables $X \prec Y$, both the removed and the inserted equation are of Sort `vv`. This is visualised by the self-loop labelled **or** attached to Sort `vv`.

Application of `decomposition` produces equations between the arguments of the function terms of the initial equations (or *false* for different functors). By the subterm property (*iii*) of $\preceq$, the arguments or the new equations are smaller than the initial arguments. We visualise this with dashed-dotted arrows, labelled **de**, from Sort `tt` to all five Sorts. The non-solid arrow-heads indicate that more than one constraint can be inserted.

Application of `confrontation` replaces one occurrence of the variable X by the value of T_1. The guard ensures that $X \prec T_1 \preceq T_2$. As long as T_1 is a variable, it gets closer from below to T_2 but can never exceed it.

- For two variables $T_1 \preceq T_2$, application of `confrontation` removes the equation X `eq` T_2 of Sort `vv` and inserts the equation T_1 `eq` T_2 of Sort `vv`. This is indicated by a self-loop of Sort `vv` labelled by **co**.
- Similarly, for a variable T_1 and a function term T_2, `confrontation` removes and inserts an equation of Sort `vt`, indicated by a self-loop of Sort `vt` labelled by **co**.

If T_1 is a function term, then so must be T_2, and we replace equation X `eq` T_2 of Sort `vt` with T_1 `eq` T_2 of Sort `tt`. As the guard requires $T_1 \preceq T_2$ we visualise this with a dashed arrow, labelled **co**.

Because Figure 3 contains all possible sort transitions, we show that all possible derivation paths, which are given by chaining the sort transitions for all equations in a given problem, are finite. First note, that the number of variables is bounded by the initial number of variables v of the problem because the solver introduces no new variables throughout the derivation.

Clearly, rule `reflexivity` applies at most *once* to a given equation. Also, rule `orientation` can apply at most once for a given equation by properties (*i*) and (*ii*) of $\preceq$. By property (*i*) of $\preceq$, rule `confrontation` can apply v times when term T_1 is a variable for a given equation. It remains to prove that no infinite derivation exists along the loop through Sort `tt` and Sort `vt`, respectively the number of (interleaved) rule applications of `decomposition` and `confrontation` are limited: Equations T_1 `eq` T_2 of Sort `tt`, that are created by `confrontation`, must be eventually decomposed because there is no other possible sort transition and the solved normal form contains no equations of Sort `tt`. As we have $T_1 \preceq T_2$ for equations of Sort `tt` which are generated by application of `confrontation`, subsequent application of `decomposition` on T_1 `eq` T_2 produces terms which are smaller than T_2.

Since both the number of subterms in a given problem and the number of variables v are finite, and since the term order is thus well-founded, the number of `confrontation/decomposition` cycles is bounded. □

3.3 Exponential Complexity for Non-flat Equations

A closer look at Figure 3 allows us to prove that the derivation length is (at most) exponential in the problem size for any term order. By giving an exponential witness query, we also show that this result is tight for (at least) the term-size order $\preceq_s$, term-depth order $\preceq_d$, and the measure-order $\preceq_m$ of [10]. Therefore, worst-case space and (hence) time complexity of the classic CHR constraint solver for unification is *exponential.*

Definition 4. *The* problem size $\#C$ *of a problem* $C = \bigwedge_{i=1}^{n} S_i$ `eq` T_i *is given by* $\#C := \sum_{i=1}^{n} \#S_i + \#T_i$.

Before proving our first main result, we present a simple, yet basic insight on the number of occurrences of *function subterms.*

Property 1. A problem containing n occurrences of function symbols contains n occurrences of function subterms.

We now study the *multi-set of function subterms* for a given problem: Application of `decomposition` removes two function subterms and `confrontation` adds function subterms when replacing a variable X by a function term T_1. Application of `confrontation` in the case of a variable T_1 as well as application of `reflexivity` or `orientation` are invariant to the multi-set of function subterms. As the RT solver only decomposes and copies terms, we have:

Property 2. All function subterms during a computation are function subterms of the initial problem.

Let $S_1 \preceq \cdots \preceq S_n$ be an ascending chain of all occurrences of function subterms of a problem C. The *multiplicities vector* $\langle \#S_1, \ldots, \#S_n \rangle$ is defined by the current number of occurrences of subterms during the computation, e.g. $a \preceq a \preceq f(a) \preceq g(a)$ is an ascending chain for the problem X `eq` $f(a) \wedge X$ `eq` $g(a)$ and $\langle 1, 1, 1, 1 \rangle$, $\langle 2, 0, 1, 1 \rangle$, and $\langle 0, 2, 1, 1 \rangle$ are valid initial multiplicities vectors.

Property 3. The number of `confrontation/decomposition` cycles, i.e. applications of `confrontation` with generation of an equation of Sort `tt` with subsequent removal by `decomposition` (along the dashed arrow in Figure 3), for a problem with n occurrences of function symbols is bounded by 2^n.

Replacing variable X of an equation X `eq` T_2 by a function term T_1 by application of `confrontation` adds one copy of each function subterm occurrence of term T_1 to the problem. Subsequent application of `decomposition` on equation T_1 `eq` T_2 removes one occurrence of both T_1 and T_2 from the problem.

Because the newly created equation T_1 `eq` T_2 of Sort `tt` can only be removed by `decomposition`, no occurrences of function subterms of T_1 `eq` T_2 are effected by intermediate applications of other rules and we do not need to manifest the temporarily addition of T_1 to the problem and can restrict ourselves to *all proper* function subterms of T_1. Summarising, one `confrontation/decomposition` cycle *adds one copy of each proper function subterm occurrence of term* T_1 and *removes one occurrence of the function subterm* T_2 from the problem, hence some entries of the multiplicities vector which are left to T_1's position increase by one and T_2's entry decreases by one.

Starting from the (canonical) initial multiplicities vector $\langle 1, \ldots, 1 \rangle$, the number of cycles is bounded by the number required to make $\langle 1, \ldots, 1 \rangle$ equal to

$\langle 0, \ldots, 0\rangle$ (as then neither `confrontation` nor `decomposition` can apply). Furthermore, we use an upper bound for each cycle by increasing *all* multiplicities to the left of T_2*'s entry* by one (and not only the effected entries of *proper subterms of* T_1) and reduce T_2's entry by one. As we increase entries to the left of a given position in the multiplicities vector, starting form the right side yields the maximal number of possible cycles: From $\langle 1, \ldots, 1\rangle$, we arrive at $\langle 2, \ldots, 2, 0\rangle$ after one cycle and another *two* cycles bring us (via $\langle 3, \ldots, 3, 1, 0\rangle$) to $\langle 4, \ldots, 4, 0, 0\rangle$. Altogether at most $\sum_{i=0}^{n-1} 2^i \leq 2^n$ many cycles suffice.

Theorem 2. *The derivation length of a problem C with problem size $\#C$ of the RT solver using any term order is bounded by $O(2^{\#C})$.*

Proof. Consider a problem C of initial size $\#C = v + n$ with v (occurrences of) variables and n (occurrences of) function symbols.

As only the transition from Sort `tv` to Sort `tt` by application of `confrontation` (along the dashed arrow in Figure 3) increases the problem size, the maximal number of applications is bounded by $O(2^n) = O(2^{\#C})$, cf. Property 3. Each time a variable is replaced by a term, the problem size increases by less than $\#C$. Hence, the maximal problem size during the computation is bounded by $O(\#C\,2^{\#C}) = O(2^{\#C})$ and the `confrontation/decomposition` cycle produces no more than $O(2^{\#C})$ many equations.

As application of `decomposition` strictly decreases the problem size, it can apply at most $O(2^{\#C})$ many times. For each equation, the self-loops can apply at most v times, altogether $O(3v\,2^{\#C}) = O(2^{\#C})$ many rule applications, and the transition from Sort `tv` to Sort `vt` and from Sort `vv` to Sort `bi` can happen only once for each equation, altogether $O(2^{\#C})$ many rule applications.

The derivation length is hence bounded by $O(4\,2^{\#C}) = O(2^{\#C})$. □

We now present a witness query with exponential space complexity (for details see [10]): We apply `confrontation` between selected equations exhaustively before application `decomposition` to yield a maximal number or generated constraints. For mutually recursively defined terms

$$\mathcal{U}_i := \begin{cases} X & \text{if } i = 0 \\ f(\mathcal{L}_{i-1}, X) & \text{otherwise} \end{cases} \qquad \mathcal{L}_i := \begin{cases} X & \text{if } i = 0 \\ f(X, \mathcal{U}_{i-1}) & \text{otherwise} \end{cases}$$

the problem $C(n) = (\bigwedge_{i=1}^{n} X \;\texttt{eq}\; \mathcal{L}_i) \wedge X \;\texttt{eq}\; \mathcal{U}_n \wedge X \;\texttt{eq}\; \mathcal{L}_n$ has quadratic size $\#C(n) = O(n^2)$. For any term order which satisfies that $\mathcal{L}_i$ and $\mathcal{U}_i$ are indifferent (this includes $\preceq_s$, $\preceq_d$, and $\preceq_m$) there exists a derivation which produces exponentially many equations. Precisely 2^{n+1} many equations $X \;\texttt{eq}\; X$ are produced.

Our first main result is that the RT solver using any term order has exponential space and (hence) exponential time complexity. The derivation length is (at most) exponential for any term order (and this result is even tight, e.g. for the standard term-size order).

3.4 Quadratic Complexity for Flat Equations

We can improve the worst-case time and space complexity of the CHR rational tree solver from exponential to quadratic by simply requiring that equations are in flat form when the problem is given. For flat terms, property (*ii*) automatically implies (*iii*) of the term order.

Definition 5. *A conjunction of equations is in* flat form *if each equation contains at most one function symbol.*

For a conjunction of equations in flat form, application of `decomposition` yields equations of Sort `vv` (or *false* for different functors) as all proper subterms for flat terms are variables. Hence we can remove the arrows from Sort `tt` to Sorts `tt`, `tv`, and `vt` from Figure 3 and the sort transition graph for the flat problem, given in Figure 4, *lacks the intricate interaction of* `confrontation` *and* `decomposition`.

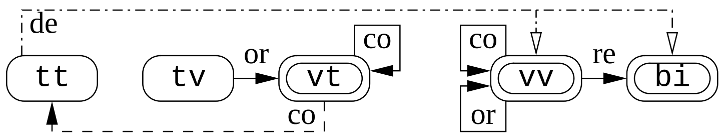

Fig. 4. Sort transition graph for the RT solver for flat terms

Theorem 3. *The derivation length of the RT solver using any term order for a* flat *problem C with problem size $\#C$ is bounded by $O(\#^2C)$.*

Proof. For each initial equations of Sort `vt`, rule `confrontation` can apply at most $\#C$ times (as the number of variables is also bounded by #C) and is hence bounded by $\#^2C$. The sort transition from Sort `vt` to Sort `tt` at most doubles the size of each equation and subsequent application of `decomposition` on the at most $\#C$ equations generates at most $\#C$ equations of Sort `vv`. Together with initial equations of Sort `vv`, rule `confrontation` applies at most $\#C$ times for each of the at most $\#C$ times many equations between two variables. □

4 Solving Existentially Quantified Conjunctions of Non-flat Equations in $\mathcal{T}$

We extend the preceding RT solver for solving existentially quantified conjunctions of non-flat equations in the theory $\mathcal{T}$ of finite or infinite trees with a quadratic complexity. Solving a quantified conjunction φ of non-flat equations in $\mathcal{T}$ means to transform φ into an equivalent existentially quantified conjunction ϕ of flat equations such that ϕ is either the formula *true* or the formula *false* or a formula having at least one free variable, being neither *true* nor *false* in $\mathcal{T}$ and where the solutions of the free variables are expressed in a clear and explicit way. In particular, if φ has no free variables then ϕ is either the formula *true* or the formula *false*. Due to lack of space we could not present in this paper all the CHR rules of our extended solver. Our full implementation is available online at `http://www.informatik.uni-ulm.de/pm/index.php?id=139`.

4.1 Flattening Non-flat Equations in $\mathcal{T}$

The following property is easily shown in $\mathcal{T}$.

Property 4. For a conjunction of constraints $\bigwedge_{i=1}^{n} S_i \;\texttt{eq}\; T_i$ and new quantified variables $X_1, \ldots, X_n$ we have

$$\mathcal{T} \models \left(\bigwedge_{i=1}^{n} S_i \;\texttt{eq}\; T_i \right) \leftrightarrow \exists X_1 ... \exists X_n \bigwedge_{i=1}^{n} \big(X_i \;\texttt{eq}\; S_i \wedge X_i \;\texttt{eq}\; T_i \big) \enspace .$$

For an atomic constraint $X \;\texttt{eq}\; T$, with a function term $T = f(T_1, \ldots, T_n)$ and new quantified variables $X_1, \ldots, X_n$ we have

$$\mathcal{T} \models X \;\texttt{eq}\; T \leftrightarrow \exists X_1 ... \exists X_n \, X \;\texttt{eq}\; f(X_1, \ldots, X_n) \wedge \left(\bigwedge_{i=1}^{n} X_i \;\texttt{eq}\; T_i \right) \enspace .$$

This property shows that a conjunction of non-flat equations can be transformed into an existentially quantified conjunction of flat equations by adding new existentially quantified variables. In our CHR implementation, we traverse the equations of the problem once and replace each nested function symbol by a new existentially quantified variable and a new equation with that variable. For example, the formula $\exists X \, h(X, f(Y)) \;\texttt{eq}\; h(Y, f(X))$ is flattened to $\exists ABCX \, A \;\texttt{eq}\; h(X, B) \wedge B \;\texttt{eq}\; f(Y) \wedge A \;\texttt{eq}\; h(Y, C) \wedge C \;\texttt{eq}\; f(X)$.

In previous work [10] we showed:

Property 5. The size of the flattened problem $\#[C]$ is linear in the problem size, i.e. $\#[C] = O(\#C)$. The number of new existentially quantified variables and the number of new equations is linear in the problem size. The flattening of a problem C can be done in linear time and space w.r.t. the problem size $\#C$.

4.2 Reachable Variables and Equations

The theory $\mathcal{T}$ does not accept full elimination of quantifiers. Hence elimination of existentially quantified variables from a conjunction of equations is not evident. We present the notion of *reachable variables* and use it to detect if a quantified variable can be eliminated or not.

Definition 6. *A* basic formula *is a conjunction of equations in flat form in which all the left hand sides of the equations are variables. Let $\bar{X}$ be a vector of variables*[6] *and let α be a basic formula. The formula $\exists \bar{X} \, \alpha$ is called* formatted *if*

[6] This includes the empty vector ε. Recall also that an empty conjunction of equations is always reduced to *true*.

(i) α does not contain equations of the form Z `eq` Z or Y `eq` X with Z a variable, X an element of $\bar{X}$, and Y a free variable of $\exists \bar{X}\, \alpha$;

(ii) all the left hand sides of the equations of α are distinct variables.

Let us now introduce the notion of *reachable variable*:

Definition 7. *Let $\exists \bar{X}\, \alpha$ be a formatted formula. The* reachable variables *and equations of α from a variable X_0 (the variable X_0 can possibly belong to $\bar{X}$) 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 the variable 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 $\exists \bar{X}\, \alpha$.*

Example 2. In the following formatted formula with free variable Z

$$\exists UVWX\, Z \texttt{ eq } f(U,V) \wedge V \texttt{ eq } g(V) \wedge W \texttt{ eq } f(U,V,X) \ , \tag{1}$$

the equations Z `eq` $f(U,V)$ and V `eq` $g(V)$ and the variables Z, U, and V are reachable. The equation W `eq` $f(U,V,X)$ and the variables W and X are not reachable. Note that the quantifications $\exists UV$ cannot be eliminated since the existence of valid instantiations of U and V in any model M of $\mathcal{T}$ depends on the instantiations of the free variable Z. In fact, if Z is instantiated by $g(0,0)$ then the preceding formula is false in M and if Z is instantiated by $f(1, g(g(g(...))))$ then the preceding formula is true in M. On the other hand, the quantification $\exists WX$ can be removed. In fact, according to axiom [A3] of $\mathcal{T}$ we have $\mathcal{T} \models \exists W\, W \texttt{ eq } f(U,V,X)$. Thus, (1) is equivalent in $\mathcal{T}$ to $\exists UVX\, Z \texttt{ eq } f(U,V) \wedge V \texttt{ eq } g(V)$, which is equivalent to $\exists UV\, Z \texttt{ eq } f(U,V) \wedge V \texttt{ eq } g(V)$.

Example 2 can help the reader to understand the following property:

Property 6. Let $\exists \bar{X}\, \alpha$ be a formatted formula. We have

$$\mathcal{T} \models (\exists \bar{X}\, \alpha) \leftrightarrow (\exists \bar{X}'\, \alpha')$$

where (i) $\bar{X}'$ is the vector of the variables of $\bar{X}$ which are reachable in $\exists \bar{X}\, \alpha$ and (ii) α' is the conjunction of the reachable equations of $\exists \bar{X}\, \alpha$.

This property simply states that non-reachable variables and equations of $\exists \bar{X}\, \alpha$ can be eliminated while the other quantified variables are linked to the instantiations of the free variables. The formatted formula $\exists \bar{X}'\, \alpha'$ is called *final solved form* of $\exists \bar{X}\, \alpha$.

4.3 Reachability in CHR

The CHR implementation of Property 6 consists of the following CHR rules.

```
r0 @ free(X), X eq T  ==> reach(X).
r1 @ reach(X), X eq T <=> nonvar(T) | reachargs(T), reach(X), X sol_eq T.
r2 @ reach(X), X eq Y <=> var(Y)    | reach(Y), reach(X), X sol_eq Y.
r3 @ reach(X), exists(X) <=> sol_exists(X), reach(X).
```

We create CHR constraints `sol_exists` and `sol_eq` for the reachable existentially quantified variables and the reachable equations of any formatted formula $\exists \bar{X}\, \alpha$. Recall that $\exists \bar{X}\, \alpha$ is a formatted formula (cf. Definition 6) for termination and correctness.

Initially, free variables of the formatted formula $\exists \bar{X}\, \alpha$ are stored in CHR constraints `free` and existentially quantified ones in `exists`. All free variables which occur as l.h.s. of an equation of the formatted formula $\exists \bar{X}\, \alpha$ are marked as reachable by rule `r0`. Rules `r1` and `r2` mark the reachable equations. For a flat term T, the built-in reachargs(T) of rule `r1` marks all arguments of T as reachable, by a simple recursion for an auxiliary constraint with rules similar to the ones for `lists2eq` of the RT solver. Rule `r3` marks the reachable quantified variables.

The complexity of this algorithm is bounded by $O(vq)$ where v is the number of distinct variables in the formatted formula $\exists \bar{X}\, \alpha$ and q is the number of equations of α. Since the left hand sides of α are distinct variables (cf. Definition 6) we have $q \leq v$ from which we deduce the following property:

Property 7. The derivation length of *reachability* is bounded by $O(v^2)$, where v is the number of distinct variables in the formatted formula.

4.4 The Solving Algorithm

To solve an existentially quantified conjunction $\exists \bar{X}\, \alpha$ of non-flat equations we apply the following algorithm.

(1) Transform $\exists \bar{X}\, \alpha$ into an equivalent existentially quantified conjunction $\exists \bar{Y}\, \beta$ of flat equations.

(2) Apply the RT solver on β using a *term-order where the variables of* $\bar{Y}$ *are smaller than the free variables of* $\exists \bar{Y}\, \beta$. Let δ be the obtained formula.

(3) If δ is different from *false*[7] then $\exists \bar{Y}\, \delta$ is a formatted formula whose final solved form is obtained using our CHR reachability rules.

[7] Note that CHR terminates immediately when *false* is inserted in the store.

Example 3. Let us solve the following formula with free variables X and V

$$\exists YZ\, f(X)\ \mathtt{eq}\ f(g(X,Y)) \land Z\ \mathtt{eq}\ f(V) \land Z\ \mathtt{eq}\ f(f(Y))$$

After flattening we get

$$\exists YZABCD\, A\ \mathtt{eq}\ f(X) \land B\ \mathtt{eq}\ f(D) \land A\ \mathtt{eq}\ B \land D\ \mathtt{eq}\ g(X,Y) \land \\ Z\ \mathtt{eq}\ f(V) \land Z\ \mathtt{eq}\ f(C) \land C\ \mathtt{eq}\ f(Y)\ .$$

The RT-solver returns the following formatted formula

$$\exists YZABCD\, B\ \mathtt{eq}\ f(X) \land D\ \mathtt{eq}\ X \land A\ \mathtt{eq}\ B \land X\ \mathtt{eq}\ g(X,Y) \land C\ \mathtt{eq}\ V \land \\ Z\ \mathtt{eq}\ f(C) \land V\ \mathtt{eq}\ f(Y)\ .$$

We now compute the reachable variables X, V, and Y. We can then eliminate the quantifications $\exists ABCDZ$ and the equations $B\ \mathtt{eq}\ f(X)$, $D\ \mathtt{eq}\ X$, $A\ \mathtt{eq}\ B$, $C\ \mathtt{eq}\ V$, $Z\ \mathtt{eq}\ f(C)$. The final solved form of the preceding formatted formula is

$$\exists Y\, X\ \mathtt{eq}\ g(X,Y) \land V\ \mathtt{eq}\ f(Y)\ .$$

Note that the solutions of the free variables X and V are expressed in clear and explicit way. Moreover the quantifier $\exists Y$ could not be eliminated since Y is a reachable variable, i.e. it depends on the instantiations of X and V.

From Theorem 3, Property 7, and Property 5 we have

Theorem 4. *The derivation length of an existentially quantified non-flat problem C with problem size $\#C$ of the extended RT solver is bounded by $O(\#^2C)$.*

5 Conclusion

The complexity of the classic CHR rational tree equation solver [3, 5, 14] was an open problem for more than a decade. We showed in this paper its termination and exponential complexity in time and space for any term order when handling non-flat equations, as well as its quadratic complexity for flat equations. This part of our new results extends those given in [10], which were limited to an artificial term-measure order.

Moreover we extended the solver to handle existentially quantified conjunctions of non-flat equations in quadratic time and space complexity. For that, we first flatten the equations by introducing new quantified variables, then solve the flat problem by the classic RT solver, and finally remove particular quantifiers and equations. Our new results extend those given in [10] by introducing *existentially quantified variables* and removing unnecessary quantified variables and equations using *reachability*. We are now able to express the solutions of any existentially quantified conjunction of non-flat equations in a short, clear, and explicit way in all models of the theory $\mathcal{T}$.

To the best of our knowledge, this is the first CHR solver for existentially quantified conjunctions of non-flat equations in $\mathcal{T}$ with *quadratic complexity*. Future work aims to reach a *linear complexity* solver by combining both the RT solver and our reachability computation with the union-find algorithm in CHR [13, 4].

Acknowledgements. We thank the anonymous reviewers for their comments which helped us to improve this paper.

References

1. A. Colmerauer. Prolog and infinite trees. In K. L. Clark and S.-A. Tärnlund, editors, *Logic Programming*, pages 231–251. Academic Press, London, 1982.
2. K. Djelloul. Decomposable theories. *J. Theory and Practice of Logic Programming*, to appear.
3. T. Frühwirth. Theory and Practice of Constraint Handling Rules. *J. Logic Programming*, 37(1-3):95–138, 1998.
4. T. Frühwirth. Parallelizing union-find in Constraint Handling Rules using confluence. In *ICLP 2005*, volume 3668 of *LNCS*, pages 113–127. Springer, 2005.
5. T. Frühwirth and S. Abdennadher. *Essentials of Constraint Programming.* Springer, 2003.
6. J. Herbrand. *Recherches sur la théorie de la demonstration.* PhD thesis, Université de Paris, France, 1930.
7. G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *J. ACM*, 27(4):797–821, 1980.
8. M. J. Maher. Complete axiomatizations of the algebras of finite, rational, and infinite trees. In *LICS'88*, pages 348–357, Los Alamitos (CA), USA, 1988.
9. A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. Program. Lang. Syst.*, 4(2):258–282, 1982.
10. M. Meister and T. Frühwirth. Complexity of the CHR rational tree equation solver. In *CHR 2006*, volume 452 of *Report CW*, pages 77–92. K.U. Leuven, Belgium, 2006.
11. M. S. Paterson and M. N. Wegman. Linear unification. *J. Computer and System Sciences*, 16(2):158–167, 1978.
12. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.
13. T. Schrijvers and T. Frühwirth. Optimal union-find in Constraint Handling Rules. *J. Theory and Practice of Logic Programming*, 6(1&2):213–224, 2006.
14. T. Schrijvers et al. Constraint Handling Rules (CHR) web page, 2007. `http://www.cs.kuleuven.ac.be/~dtai/projects/CHR/`.
15. R. E. Tarjan and J. Van Leeuwen. Worst-case analysis of set union algorithms. *J. ACM*, 31(2):245–281, 1984.