Reconstructing almost-linear Tree Equation Solving Algorithms in CHR

Marc Meister and Thom Frühwirth

Fakultät für Ingenieurwissenschaften und Informatik
Universität Ulm, Germany
{`Marc.Meister,Thom.Fruehwirth`}`@uni-ulm.de`

Abstract. Solving equations over trees is an essential problem in symbolic computation. We reconstruct almost-linear tree equation solving algorithms in the high-level and rule-based Constraint Handling Rules (CHR) language. To this end, we combine the available CHR solver for rational trees with the union-find algorithm. We extend the almost-linear CHR rational tree solver to handle existentially quantified conjunctions of equations in the theory of finite or infinite trees in almost-linear time.

1 Introduction

Static unification, i.e. equation solving over trees, is an essential problem in symbolic computing, in particular in theorem proving and declarative programming languages. Logic programming languages like Prolog rely on unification to treat logical variables, term rewriting systems need it for confluence testing, Constraint Handling Rules (CHR) [5] for both, and functional languages like ML for type checking.

Even though *almost-linear time algorithms* based on the essential union-find algorithm [17] for the unification problem over finite and rational trees are known since the 1970ties, e.g. [9] and [12], they are rarely implemented e.g. in Prolog with the argument that they are too complicated and cause significant overhead.

Moreover, in the context of constraint logic programming, one also needs to deal with local, i.e. existentially quantified variables. It is not obvious how to extend the classic algorithms to these cases without giving up on optimal complexity. In this paper we do so in a straightforward way using CHR as an implementation language. This choice is not accidental. The code in CHR is more concise than even theoretical expositions of unification, the extensions are straightforward. CHR guarantees properties like anytime and online algorithm and is concurrent, and it was shown that any algorithm, including thus union-find, can be implemented in CHR with best-known time and space complexity.

For a lucid exposition of unification algorithms see [1] and for a multidisciplinary survey of unification see [10].

Contributions and Overview. We reconstruct an almost-linear tree equation solving algorithm as concise CHR solver and modify it to solve existentially quantified conjunctions of equations in the theory of finite and infinite trees in almost-linear time.

– We recall the basics of Constraint Handling Rules (CHR) [5], Maher's theory $\mathcal{T}$ of finite or infinite trees [11], and Tarjan's union-find algorithm [17] in Section 2.
– We reconstruct Huet's almost-linear tree solving algorithm for finite and infinite trees [9] in CHR by combining the quadratic classic CHR rational tree solver [5, 13] with the almost-linear CHR union-find solver [15]. Our exceptionally concise, high-level, and rule-based CHR solver has optimal almost-linear time complexity. See Section 3.
– We modify the CHR solver to solve existentially quantified conjunctions of non-flat equations in theory $\mathcal{T}$. The finally solved formula is free of equations that are not linked to the instantiations of free variables. See Section 4.

Supplementary Online Information. Our complete implementation is available online at `http://www.informatik.uni-ulm.de/pm/index.php?id=142`.

2 Preliminaries

Readers familiar with CHR, the theory $\mathcal{T}$, or the union-find algorithm can skip the corresponding sub-section(s).

2.1 Constraint Handling Rules

Constraint Handling Rules (CHR) [5, 16] is a concurrent, committed-choice, rule-based logic programming language. We distinguish between two different kinds of constraints: *built-in constraints* which are solved by a given constraint solver, and *user-defined constraints* which are defined by the rules in a CHR program. This distinction allows one to embed and utilise existing constraint solvers.

A *CHR program* P is a finite set of rules R @ $H_1 \setminus H_2 \Leftrightarrow G \mid B$. Each rule has a unique identifier R, the *head* $H_1 \setminus H_2$ is a non-empty multi-set conjunction of user-defined constraints, the *guard* G is a conjunction of built-in constraints, and the *body* B is a goal. A *goal* is a multi-set conjunction of built-in and user-defined constraints. We omit the trivial guard expression "*true* |". A rule R is a *simpagation rule* if both head expressions H_1 and H_2 are non-empty. If expression H_1 is empty, we have a *simplification rule* and write R @ $H_2 \Leftrightarrow G \mid B$. We do not use *propagation rules* with empty head expression H_2 in this paper.

The *operational semantics* of CHR is defined by a state transition system where states are multi-set conjunctions of atomic constraints. Any one of the rules that are applicable can be applied and rule application cannot be undone since CHR is a committed-choice language. A rule R @ $H_1 \setminus H_2 \Leftrightarrow G \mid B$ is applicable in state $\langle H_1' \wedge H_2' \wedge C\rangle$ if the built-in constraints C_b of C imply that H_1' matches H_1, H_2' matches H_2, and the guard G is entailed under this matching, cf. (1). The consistent, predicate logic, built-in constraint theory CT contains at least Clark's syntactic equality $\doteq$.

$$\begin{array}{ll}
\text{IF} & R \text{ @ } H_1 \setminus H_2 \Leftrightarrow G \mid B \text{ is a fresh variant of rule } R \text{ with new variables } \bar{X} \\
\text{AND} & CT \models (\forall)\, C_b \rightarrow \exists \bar{X}\, (H_1 \doteq H_1' \wedge H_2 \doteq H_2' \wedge G) \\
\text{THEN} & \langle H_1' \wedge H_2' \wedge C\rangle \rightarrowtail_R \langle H_1' \wedge G \wedge B \wedge H_1 \doteq H_1' \wedge H_2 \doteq H_2' \wedge C\rangle
\end{array} \tag{1}$$

If applied, a rule *replaces* the matched user-defined constraints of the head expression H_2 in the state by the body of the rule. Rules are applied until exhaustion, i.e. the CHR run-time system computes the reflexive transitive closure $\rightarrowtail_P^*$ of $\rightarrowtail_P$. The derivation $\langle C\rangle \rightarrowtail_P^* \langle C'\rangle$ has initial goal C, answer C', and *derivation length* defined by the number of rule applications. Whenever the conjunction of constraints in a state becomes inconsistent the derivation terminates immediately with answer *false*.

CHR rules have an immediate predicate logic declarative semantics. For a simplification rule, the guard implies a logical equality between the l.h.s. and r.h.s. of the rule. Formally, the logical reading of the simplification rule R @ $H_2 \Leftrightarrow G \mid B$ is $(\forall)\ G \rightarrow (H_2 \leftrightarrow \exists\bar{Y}\ B)$ where $(\forall)$ denotes universal closure and $\bar{Y}$ are the variables that appear only in the body B.

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

The theory $\mathcal{T}$ of finite or infinite trees is equivalent to Clark's equality theory (CET) *without* the occur-check (acyclicity) and one *additional uniqueness axiom*, which handles implied equalities, makes the theory complete [11]. The signature of $\mathcal{T}$ consists of an infinite set of distinct function symbols (written as lower-case letters) and the binary predicate symbol $=$.

Besides the usual axioms for *reflexivity*, *symmetry*, and *transitivity* for variables of CET, theory $\mathcal{T}$ has the following axiom scheme according to [11]:

$$(\forall)\ \neg\big(f(S_1,\ldots,S_n) = g(T_1,\ldots,T_m)\big) \tag{A1}$$

$$(\forall)\ f(S_1,\ldots,S_n) = f(T_1,\ldots,T_n) \rightarrow \bigwedge_{i=1}^{n} S_i = T_i \tag{A2}$$

$$(\forall)\ \exists! X_1 \ldots \exists! X_n\ \bigwedge_{i=1}^{n} X_i = T_i \tag{A3}$$

In (A1), f and g are distinct function symbols. In (A3), $X_1,\ldots,X_n$ are *distinct* variables, $T_1,\ldots,T_n$ are function terms, i.e. no variables, and $\exists! X_i$ denotes that *there exists a unique variable* X_i.

Axiom scheme (A1) is called *contradiction* or *clash* as two distinct function symbols cannot be equal. Axiom scheme (A2) allows to *decompose* an equation by propagating equality to pairwise equality of the arguments. From (A1) and (A2) we see that we can strengthen the implication in (A2) to logical equivalence. The reverse direction is often called *composition.* Axiom scheme (A3) requires that for a particular form of conjunction of equations a *unique* set of solutions exists: For example the formula $\exists X\ X = f(X)$ has a unique solution which is the infinite tree $f(f(f(\ldots)))$. Without (A3), the theory is not complete, e.g. neither does the sentence $\exists X \exists Y\ X = f(X) \wedge Y = f(Y) \wedge \neg(X = Y)$ follow, nor does its negation.

The structure of finite or infinite trees and the structure of the rational trees are models of $\mathcal{T}$. A *rational tree (RT)* is a finite or infinite tree whose set of subtrees is finite, i.e. it has a finite representation as a directed (possibly cyclic) graph by merging all nodes with common subtrees. A rational tree can be represented as conjunction of binary equality constraints, e.g. the infinite tree $f(f(f(\dots)))$ only contains itself as its set of subtrees $\{f(f(f(...)))\}$ is finite and it can be represented by the equation $X = f(X)$.

The theory $\mathcal{T}$ does not accept full elimination of existential quantifiers, e.g. in the formula $\exists X\ Y = f(X)$ we cannot remove or eliminate the quantifier $\exists X$ and the formula is neither true nor false in $\mathcal{T}$ but depends on the instantiation of the free variable Y.

2.3 Union-Find Algorithm

The classic union-find algorithm solves the problem of maintaining a collection of disjoint sets under the operation of union [17]. Each set is represented by a rooted tree, whose nodes are the elements of the set. The root is called the *representative* of the set. The representative may change when the tree is updated by a union operation. With the algorithm come three operations:

make(X) introduces X by creating a new tree with the only node X;
find(X, R) returns the representative R of the tree in which X is contained by following the path from the node X to the root R of the tree;
union(X, Y) joins the two trees in which X and Y are contained by finding their roots R_X and R_Y. If they are different one root node is updated to point to the other (possibly changing the representative).

With the two independent optimisations *path compression* and *union-by-rank* that keep the trees shallow and balanced, the union-find algorithm has logarithmic worst-case and almost constant amortised running time per operation [17]: For n variables and a mixed sequence of u calls to the union operator and f calls to the find operator, the time complexity for an optimal implementation is $O(m\,G(n))$ with $m = 2u + f$ (we allow calls to union with arguments that are from the same tree). Function G is an extremely slow growing inverse of Ackermann's function with $G(n) < 5$ for all practical n.

Accessing the operations of the union-find algorithm as built-in constraints requires to define ask- and tell-versions for find(X, R) and union(X, Y). We define find(X, R) (ask) to be true iff X is not a root variable. Telling the constraint find(X, R), however, returns the representative R of the tree in which X is contained. Similarly union(X, Y) (ask) is true iff X and Y belong to the same tree but only telling union(X, Y) makes X and Y belong to the same tree.

Clearly the *predicate-logical reading* of union(X, Y) for two variables is equality $X = Y$. The constraint union(X, Y) observes the axioms of reflexivity, symmetry, and transitivity of CET for variables: Inserting union(X, X) keeps the equality sets unchanged and asking union(X, X) returns true ($X = X \leftrightarrow \mathit{true}$). We have union$(X, Y)$ iff union(Y, X), hence the *orientation* of variables is invariant to the built-in theory ($X = Y \leftrightarrow Y = X$). Finally, if union$(X, Y) \wedge$ union(Y, Z) holds, then we have union(X, Z) and union is hence transitive ($X = Y \wedge Y = Z \rightarrow X = Z$). However, union$(X, Y)$ or union(Y, X) and the order constraints are told may yield different representatives.

3 Combining the Rational Tree Equation Solver with the Union-Find Algorithm

We reconstruct Huet's almost-linear infinite unification algorithm [9] as a CHR solver accessing Tarjan's union-find algorithm [17] by built-in constraints [2]. We take an *extreme programming* style of development by starting from the classic RT solver [5, 8], add the basic idea to handle equality between variables by the union-find built-in solver, and inspect the necessary changes. We then prove the correctness of our hierarchical solver, show its optimal almost-linear time complexity when using the refined semantics of CHR [3], and briefly explain how to use the optimal CHR union-find implementation [15] as built-in solver.

3.1 Classic CHR Rational Tree Equation Solver

One of the first CHR programs is the classic constraint solver for syntactic equality of rational trees that performs unification [5, 8] where equations $S = T$ between two terms are encoded as CHR constraints $S \texttt{ eq } T$ (cf. Figure 1).

Auxiliary built-ins allow the solver to be independent of the representation of terms. Besides *true* and *false*, we have $\mathrm{v}(T)$ iff T is a variable and $\mathrm{f}(T)$ iff T is a *function term*. Variables are strictly ordered by $\prec$, each variable is smaller than any function term, and function terms are ordered according to term-depth (for details see [13]). The auxiliary $\mathrm{s}(T_1, T_2)$ leads to *false* if T_1 and T_2 have not the same function symbol and the same arity (this is called clash). The auxiliary $\mathrm{a}(T, L)$ returns the arguments of a function term T as a list L.

$$\begin{aligned}
re\ @\ & X \texttt{ eq } X \Leftrightarrow \mathrm{v}(X) \mid true \\
or\ @\ & T \texttt{ eq } X \Leftrightarrow \mathrm{v}(X) \wedge X \prec T \mid X \texttt{ eq } T \\
de\ @\ & T_1 \texttt{ eq } T_2 \Leftrightarrow \mathrm{f}(T_1) \wedge \mathrm{f}(T_2) \mid \mathrm{s}(T_1, T_2) \wedge \mathrm{a}(T_1, L_1) \wedge \mathrm{a}(T_2, L_2) \wedge \texttt{e}(L_1, L_2) \\
co\ @\ & X \texttt{ eq } T_1 \setminus X \texttt{ eq } T_2 \Leftrightarrow \mathrm{v}(X) \wedge X \prec T_1 \preceq T_2 \mid T_1 \texttt{ eq } T_2 \\
aux\ @\ & \texttt{e}([X|L_1], [Y|L_2]) \Leftrightarrow X \texttt{ eq } Y \wedge \texttt{e}(L_1, L_2)
\end{aligned}$$

Fig. 1. Classic rational tree equation solver (RT solver)

We now explain application of each rule of the RT solver:

Reflexivity (*re*) removes trivial equations between identical variables.

Orientation (*or*) reverses the arguments of an equation so that the (smaller) variable comes first.

Decomposition (*de*) applies to equations between two function terms. If there is no clash, the initial equation is replaced by equations between the corresponding arguments of the terms. To this end, the CHR constraint $\texttt{e}(L_1, L_2)$

pairwise equates the lists of arguments L_1 and L_2 of the two terms using the simple recursion of rule *aux*.[1]

Confrontation (*co*) replaces the variable X in the second equation $X \texttt{ eq } T_2$ by T_1 from the first equation $X \texttt{ eq } T_1$. It performs a limited amount of variable elimination (substitution) by only considering the l.h.s.' of equations. The order in the guard ensures termination.

Property 1 ([13]). The classic RT solver terminates and if there is no clash, it returns a conjunction of atomic constraints of the form $\bigwedge_{i=1}^{n} X_i \texttt{ eq } T_i$ in the theory of the rational trees. The variables $X_1, \dots, X_n$ are pairwise distinct and X_i is different to T_j for $1 \leq i \leq j \leq n$. For a conjunction of equations with terms of maximal depth one (flat terms) its time complexity is quadratic.[2]

3.2 CHR Program Specialisation to Strict Flat Form

We specialise the classic RT solver w.r.t. goals that are in *strict flat form.*

Definition 1 (Strict Flat). *A conjunction of equations is in* strict flat form *if each equation contains at most one function symbol and each l.h.s. is a variable.*

We apply program transformation techniques for CHR [7]: Two CHR programs P_1 and P_2 are *operationally equivalent*, iff for all states S, we have $S \rightarrowtail^{*}_{P_1} S_1$, $S \rightarrowtail^{*}_{P_2} S_2$, and the two final states S_i are identical up to renaming of variables and logical equivalence of built-in constraints.

As the solver decomposes terms, all terms have depth zero or one and partitioning the condition of the guards of *or* and *co* yields the following rules with simplified guards. For conjunctions of equations in strict flat form, the classic RT solver is operationally equivalent to program $\{re, or_1, or_2, de, co_1, co_2, co_3, aux\}$:

$$
\begin{aligned}
&or_1 \;@\; Y \texttt{ eq } X \Leftrightarrow \mathrm{v}(X) \wedge \mathrm{v}(Y) \wedge X \prec Y \mid X \texttt{ eq } Y\\
&or_2 \;@\; T \texttt{ eq } X \Leftrightarrow \mathrm{v}(X) \wedge \mathrm{f}(T) \mid X \texttt{ eq } T\\
&co_1 \;@\; X \texttt{ eq } Y \setminus X \texttt{ eq } Z \Leftrightarrow \mathrm{v}(X) \wedge \mathrm{v}(Y) \wedge \mathrm{v}(Z) \wedge X \prec Y \preceq Z \mid Y \texttt{ eq } Z\\
&co_2 \;@\; X \texttt{ eq } Y \setminus X \texttt{ eq } T \Leftrightarrow \mathrm{v}(X) \wedge \mathrm{v}(Y) \wedge \mathrm{f}(T) \wedge X \prec Y \mid Y \texttt{ eq } T\\
&co_3 \;@\; X \texttt{ eq } T_1 \setminus X \texttt{ eq } T_2 \Leftrightarrow \mathrm{v}(X) \wedge \mathrm{f}(T_1) \wedge \mathrm{f}(T_2) \mid T_1 \texttt{ eq } T_2
\end{aligned}
$$

To avoid intermediate equations $T_1 \texttt{ eq } T_2$ with two function terms T_i, we unfold rule *de* into rule co_3 and add the mnemonic rule $de{+}co_3$:

$$
\begin{aligned}
de{+}co_3 \;@\; X \texttt{ eq } T_1 \setminus X \texttt{ eq } T_2 \Leftrightarrow \mathrm{v}(X) \wedge \mathrm{f}(T_1) \wedge \mathrm{f}(T_2) \mid&\\
\mathrm{s}(T_1, T_2) \wedge \mathrm{a}(T_1, L_1) \wedge \mathrm{a}(T_2, L_2) \wedge \texttt{e}(L_1, L_2)&\ .
\end{aligned}
$$

Rule $de{+}co_3$ short-cuts derivations with intermediate equations $T_1 \texttt{ eq } T_2$, so we can remove the redundant rules *de* and co_3. As equations are in strict flat form in all states of the derivation, rule or_2 is redundant and we can remove the condition $\mathrm{v}(X)$ from the guard of each rule.

[1] To remove empty constraints $\texttt{e}([], [])$ one may want to add a rule $\texttt{e}([], []) \Leftrightarrow$ *true*.

[2] The classic RT solver also works with non-flat equations with exponential complexity.

$$\begin{aligned}
re \ @\ & X \mathtt{\ eq\ } X \Leftrightarrow \mathit{true}\\
or_1 \ @\ & Y \mathtt{\ eq\ } X \Leftrightarrow \mathrm{v}(Y) \wedge X \prec Y \mid X \mathtt{\ eq\ } Y\\
co_1 \ @\ & X \mathtt{\ eq\ } Y \setminus X \mathtt{\ eq\ } Z \Leftrightarrow \mathrm{v}(Y) \wedge \mathrm{v}(Z) \wedge X \prec Y \preceq Z \mid Y \mathtt{\ eq\ } Z\\
co_2 \ @\ & X \mathtt{\ eq\ } Y \setminus X \mathtt{\ eq\ } T \Leftrightarrow \mathrm{v}(Y) \wedge \mathrm{f}(T) \wedge X \prec Y \mid Y \mathtt{\ eq\ } T\\
de{+}co_3 \ @\ & X \mathtt{\ eq\ } T_1 \setminus X \mathtt{\ eq\ } T_2 \Leftrightarrow \mathrm{f}(T_1) \wedge \mathrm{f}(T_2) \mid\\
& \qquad \mathrm{s}(T_1,T_2) \wedge \mathrm{a}(T_1,L_1) \wedge \mathrm{a}(T_2,L_2) \wedge \mathtt{e}(L_1,L_2)\\
aux \ @\ & \mathtt{e}([X|L_1],[Y|L_2]) \Leftrightarrow X \mathtt{\ eq\ } Y \wedge \mathtt{e}(L_1,L_2)
\end{aligned}$$

Fig. 2. Rational tree solver for strict flat form (RT solver)

Lemma 1 (RT solver for strict flat form). *For conjunctions of equations in* strict flat form*, the classic RT solver (cf. Figure 1) is operationally equivalent to the RT solver for strict flat form (cf. Figure 2).*

Proof. By program specialisation, properties of the order $\prec$, splitting rules according to a partition of the condition of the guards, unfolding, and removing of redundant rules. □

We use the more accessible RT solver for strict flat form for our extreme programming approach.

3.3 An Extreme Programming Development Style

We now want to improve the time complexity of the RT solver for equations in strict flat form. We employ a union-find built-in solver to handle equations between two variables and adapt the RT solver accordingly.

To this end, consider rule *e2u* which replaces equalities $X = Y$ between two variables – encoded by CHR constraints $X \mathtt{\ eq\ } Y$ – with built-in constraints $\mathrm{union}(X,Y)$:

$$e2u \ @\ X \mathtt{\ eq\ } Y \Leftrightarrow \mathrm{v}(Y) \mid \mathrm{union}(X,Y) \ .$$

Constraint $\mathrm{union}(X,Y)$ observes the axioms of reflexivity, symmetry, and transitivity of CET for variables (cf. Sub-section 2.3). Rules *re*, or_1, and co_1 implement reflexivity, orientation (a limited form of symmetry), and confrontation between variables (a limited form of transitivity). Taking an extreme programming approach we replace the subsumed rules *re*, or_1, and co_1 with rule *e2u*.

Rule co_2 must be adapted to the union-find data-structure as its head constraint $X \mathtt{\ eq\ } Y$ overlaps with the head of rule *e2u*: We replace the *CHR head constraint* $X \mathtt{\ eq\ } Y$ *of rule* co_2 by the *built-in guard constraint* $\mathrm{union}(X,Y)$:

$$co_2{}' \ @\ X \mathtt{\ eq\ } T \Leftrightarrow \mathrm{union}(X,Y) \wedge \mathrm{f}(T) \wedge X \prec Y \mid Y \mathtt{\ eq\ } T \ .$$

In the classic RT solver the strict order of variables $\prec$, guarantees that any function term T is eventually attached to a unique variable Y in the set of equal variables with $X = Y$. The canonical unique representative in the union-find data-structure for a set of equal variables is its root. Recall that the built-in $\mathrm{find}(X, Y)$ (ask) is true if X is not a root variable, and returns a root variable Y with $X = Y$ when told. Hence, we replace rule co_2 with

$$\mathit{root} \ @\ X \ \texttt{eq}\ T \Leftrightarrow \mathrm{f}(T) \wedge \mathrm{find}(X, Y) \mid Y \ \texttt{eq}\ T \ .$$

Note that we dropped $\mathrm{union}(X, Y)$ (ask) from the guard as $\mathrm{find}(X, Y)$ implies $X = Y$. Rules *de+co*$_3$ and *aux* have no `eq` constraints between two variables in the head and are not affected by our transformation. We finally unfold rule *e2u* into rule *aux*. We now show that our UF+RT solver, given in Figure 3 is correct.

$$\begin{aligned}
\mathit{e2u}\ @\ X\ \texttt{eq}\ Y &\Leftrightarrow \mathrm{v}(Y) \mid \mathrm{union}(X, Y)\\
\mathit{root}\ @\ X\ \texttt{eq}\ T &\Leftrightarrow \mathrm{f}(T) \wedge \mathrm{find}(X, Y) \mid Y\ \texttt{eq}\ T\\
\mathit{de{+}co}_3\ @\ X\ \texttt{eq}\ T_1 \setminus X\ \texttt{eq}\ T_2 &\Leftrightarrow \mathrm{f}(T_1) \wedge \mathrm{f}(T_2) \mid\\
&\qquad \mathrm{s}(T_1, T_2) \wedge \mathrm{a}(T_1, L_1) \wedge \mathrm{a}(T_2, L_2) \wedge \texttt{e}(L_1, L_2)\\
\mathit{aux'}\ @\ \texttt{e}([X|L_1], [Y|L_2]) &\Leftrightarrow \mathrm{union}(X, Y) \wedge \texttt{e}(L_1, L_2)
\end{aligned}$$

Fig. 3. Rational tree solver for strict flat form using union-find (UF+RT solver)

Definition 2 (Solved CHR State). *A CHR state for a built-in theory that includes the union-find is* solved *if it is false or if its CHR constraints are in the form* $\bigwedge_{i=1}^{n} X_i\ \texttt{eq}\ T_i$ *with pairwise distinct* root variables $X_1, \ldots, X_n$ *and flat functions terms* $T_1, \ldots, T_n$.

Lemma 2 (Correctness). *For conjunctions of equations in strict flat form the UF+RT solver terminates with a solved state in the theory of the rational trees.*

Proof. The solver terminates as rule *e2u* removes CHR constraints $X\ \texttt{eq}\ Y$ between two variables, rule *root* pushes flat terms equations strictly upwards in the trees, and rule *de+co*$_3$ removes CHR constraints $X\ \texttt{eq}\ T$ for a function term T. As long as a state is not solved, at least one rule is applicable and if it is in solved form, no rule is applicable.

The logical reading of each rule *e2u*, *root*, and *de+co*$_3$ is valid in theory $\mathcal{T}$ because $X\ \texttt{eq}\ Y$, $\mathrm{union}(X, Y)$, and $\mathrm{find}(X, Y)$ are encodings for $X = Y$: For rule *e2u* we have $(\forall)\ X = Y \leftrightarrow X = Y$ and for rule *root* we have $(\forall) X = Y \rightarrow (X = T \leftrightarrow Y = T)$. For rule *de+co*$_3$ we consider two cases: If T_1 and T_2 have different function symbols f and g, then $\mathrm{s}(T_1, T_2)$ fails, i.e. $\neg\big(X = f(\ldots) \wedge X = g(\ldots)\big)$, otherwise we have $\big(X = f(X_1, \ldots, X_n) \wedge X = f(Y_1, \ldots, Y_n)\big) \leftrightarrow \bigwedge_{i=1}^{n} X_i = Y_i$ as $\texttt{e}([X_1, \ldots, X_n], [Y_1, \ldots, Y_n]) \leftrightarrow \bigwedge_{i=1}^{n} X_i = Y_i$ by rule *aux'*. $\square$

Definition 3 (Solved Form). *A conjunction of equations in strict flat form is* solved *if it is false or if it is in the form* $\bigwedge_{i=1}^{n} X_i = T_i$ *with pairwise distinct* *variables $X_1, \ldots, X_n$ and terms $T_1, \ldots, T_n$ for $n \in \mathbf{N}$. We require each term T_i to be different to X_j for $1 \leq j \leq n$.*

The formula $X = Y \wedge Z = Y \wedge Y = f(X)$ is solved but $X = Y \wedge Y = Z \wedge Z = f(X)$ is *not solved* as variable Y appears both on the l.h.s. and r.h.s. of equations between variables. We can convert a solved CHR state to a solved form by adding equations $X = R_X$ for each non-root variable X with root-variable R_X in linear time. Hence, root variables are on the r.h.s. in equations between two variables and on the l.h.s. for equations which contain a function symbol.

Lemma 3 (Conversion to Solved Form). *Consider a solved CHR state that is not false with CHR constraints $\bigwedge_{i=1}^{n} X_i \ \mathtt{eq}\ T_i$ and conjunction C_b of built-ins. Then $\left(\bigwedge_{X: C_b \rightarrow \mathrm{find}(X,R_X)} X = R_X\right) \wedge \left(\bigwedge_{i=1}^{n} X_i = T_i\right)$ is solved. The amortised time complexity for calling* $\mathrm{find}(X, R_X)$ *for each variable X is constant.*

Proof. Calling $\mathrm{find}(X, R_X)$ for each variable X (without intermediate calls to union) touches each node in the trees once due to path compression. □

3.4 Complexity of the UF+RT Solver

As the number of application of rules *e2u*, *de+co*$_3$, and *aux*$'$ is independent of the order rules are applied, we achieve a minimal derivation length when we delay application of rule *root*.

To this end, we use the refined semantics of CHR [3] for scheduling rule and constraint selection. In refined semantics, constraints are inserted sequentially into the store from left-to-right and applicable rules for the constraints in the store are chosen in textual execution order.

$$\begin{aligned}
\mathit{e2u} \ @\ & X \ \mathtt{eq}\ Y \Leftrightarrow \mathrm{v}(Y) \mid \mathrm{union}(X, Y) \\
\mathit{de{+}co}_3 \ @\ & X \ \mathtt{eq}\ T_1 \setminus X \ \mathtt{eq}\ T_2 \Leftrightarrow \mathrm{s}(T_1, T_2) \wedge \mathrm{a}(T_1, L_1) \wedge \mathrm{a}(T_2, L_2) \wedge \mathtt{e}(L_1, L_2) \\
\mathit{aux}' \ @\ & \mathtt{e}([X|L_1], [Y|L_2]) \Leftrightarrow \mathrm{union}(X, Y) \wedge \mathtt{e}(L_1, L_2) \\
\mathit{root} \ @\ & X \ \mathtt{eq}\ T \Leftrightarrow \mathrm{find}(X, Y) \mid Y \ \mathtt{eq}\ T
\end{aligned}$$

Fig. 4. UF+RT solver for refined semantics (refined UF+RT)

Consider the concise UF+RT solver for refined semantics (cf. Figure 4): Compared to the UF+RT solver, rule *root* is last in textual order and guards are simplified. When rule *root* applies, all equalities of constraints are already propagated, i.e. there are neither equations $X \ \mathtt{eq}\ Y$ between variables nor $\mathtt{e}(L_1, L_2)$ constraints in the store. Also, due to rule *de+co*$_3$ there is at most one equation $X \ \mathtt{eq}\ T$ with a function term T for each variable X in the store. We now bound the number of rule applications.

Lemma 4 (Rule Applications). *Consider a conjunction of equations C in strict flat form with $\#C$ occurrences of variable and function symbols. Then (i) $\#e2u + \#aux' \leq \#C$, (ii) $\#de{+}co_3 \leq \#C$, and (iii) $\#root \leq \#C$ where $\#R$ denotes the number of applications of rule R of the refined UF+RT solver.*

Proof. (i) Consider a measure for conjunctions of CHR constraints $\left|\bigwedge_{i=1}^{n} C_i\right| := \sum_{i+1}^{n} |C_i|$ where $|X \texttt{ eq } T|$ is the number of occurrences of variables in T and $|\texttt{e}(L_1, L_2)|$ is the length of list L_1. Because $|.|$ is invariant to reordering of constraints we can treat local replacements of constraints, caused by a rule applications, independently. The measure is not affected by rules *de+co*$_3$ and *root* and each application of *e2u* or *aux'* decreases the measure by one.[3] Hence $\#e2u + \#aux' \leq |C| \leq \#C$.

(ii) Application of rule *de+co*$_3$ decreases the number of occurrences of function symbols by one and hence we have $\#de{+}co_3 \leq \#C$.

(iii) Consider two cases: When *inserting* $X \texttt{ eq } T$ with a function symbol T to the store, rule *root* applies if X is not a root variable and no other constraint $X \texttt{ eq } T'$ is already in the store. For equations $X \texttt{ eq } T$ that are already *in the store*, rule *root* applies when X is no longer root due to linking. The sum of occurrences of function symbols and the number of variables is bounded by $\#C$. □

We can now give our first main result for an efficient CHR system, e.g. the K.U.Leuven system [14], that allows to find partner constraints for rule application of rule *de+co*$_3$ in *constant time* by using an *index* on the shared variable X of the head $X \texttt{ eq } T_1 \setminus X \texttt{ eq } T_2$. For details on constant time rule selection due to combination of matching, partner constraints, and guards, see [15]. We also require that the built-in union-find algorithm is implemented with optimal almost-linear time complexity.

Theorem 1 (Almost-linear Refined Tree Equation Solver). *Consider an efficient CHR system with indexing and an optimal, almost-linear union-find implementation accessible through built-in constraints. Solving conjunctions of equations in strict flat form with the refined UF+RT solver has almost-linear time complexity.*

Proof. By Lemma 2 the refined UF+RT solver is correct as the refined semantics is an instance of the operational semantics [3]. By Lemma 4, both the number of rule applications and the number of calls to the built-in constraints union (tell) and find (tell) is linear. Also the solver does not introduce new variables. Hence the refined UF+RT solver inherits the almost-linear time complexity of the underlying union-find algorithm. Finally, the solved CHR state is converted in linear time to the solved formula by Lemma 3. □

Theorem 1 improves on the quadratic complexity from [13] to solve equations in the theory of rational trees.

[3] If there is a clash the derivation stops immediately.

3.5 Simulating the Hierarchical UF+RT Solver

The union-find algorithm has been implemented in CHR with optimal, almost-linear time complexity [15]. Because stacking one CHR solver on top of another (cf. [2] for details on hierarchical solvers) is (up to now) not supported by any CHR implementation we are aware of, we cannot use the union-find constraint $\mathrm{find}(X, Y)$ as built-in in the guard of rule *root* directly. To reuse the optimal CHR union-find implementation, where $\mathrm{union}(X, Y)$ and $\mathrm{find}(X, Y)$ are CHR constrains, both constraints can only be accessed in tell-mode. We can simulate the necessary wake-up of rule *root* (when X is no longer a root) of the ask-constraint $\mathrm{find}(X, Y)$ by replacing rule $\mathit{root} \mathbin{@} X \texttt{ eq } T \Leftrightarrow \mathrm{find}(X, Y) \mid Y \texttt{ eq } T$ with

$$\mathit{root'} \mathbin{@} \texttt{notroot}(X) \setminus X \texttt{ eq } T \Leftrightarrow \mathrm{find}(X, Y) \wedge Y \texttt{ eq } T \ \ ,$$

where $\mathrm{find}(X, Y)$ is a tell-constraint. We adapt the union-find implementation to insert an CHR constraint $\texttt{notroot}(X)$ when X becomes a non-root variable due to linking.

4 Existential Variables

In [13] the classic CHR RT solver [5, 8] was modified to solve existentially quantified conjunction of equations with quadratic complexity. We modify the refined UF+RT solver (cf. Figure 4) to solve existentially quantified conjunction of equations in *almost-linear* time.

4.1 Purging Unreachable Variables and Equations

To eliminate existentially quantified variables from an existentially quantified conjunction of equations we require that the conjunction is in *oriented* and *representative* form.

Definition 4 (Oriented Form). *An existentially quantified and solved conjunction of equations* $\exists \bar{X}\ \bigwedge_{i=1}^{n} X_i = T_i$ *is* oriented *if it does not contain equations* $X_j = T_j$ *with a free variable* $X_j \notin \bar{X}$ *and an existentially quantified variable* $T_j \in \bar{X}$.

Any non-oriented, existentially quantified, and solved conjunction of equations can be transformed into an equivalent oriented formula:

Property 2. Consider a solved formula $\exists \bar{X}\ \bigwedge_{i=1}^{n} X_i = T_i$ with an equation $X_j = T_j$ between a free variable X_j and an existentially quantified variable T_j. Then

$$\mathcal{T} \models \big(\exists \bar{X} \bigwedge_{i=1}^{n} X_i = T_i\big) \leftrightarrow \big(\exists \bar{X} \bigwedge_{i=1}^{n} E_i\big) \text{ with } E_i \equiv \begin{cases} T_j = X_j & \text{if } i = j \\ X_i[X_j \leftarrow T_j] = T_i & \text{if } i \neq j \wedge \mathrm{f}(T_i) \\ X_i = T_i[X_j \leftarrow T_j] & \text{if } i \neq j \wedge \mathrm{v}(T_i) \end{cases}$$

and the conjunction $\bigwedge_{i=1}^{n} E_i$ is in solved form.

Definition 5 (Representative Form). *An existentially quantified, solved, and oriented conjunction of equations* $\exists \bar{X}\ \bigwedge_{i=1}^{n} X_i = T_i$ *is* representative *if each function term* T_j *does not contain an existentially quantified argument variable* $X_k \in \bar{X}$ *with* $k \neq j$*.*

We can transform an oriented formula to an equivalent representative formula by replacing argument variables by the *representative* variables, i.e. by variables on the r.h.s. of equations between two variables:

Property 3. Consider an oriented formula $\exists \bar{X}\ \bigwedge_{i=1}^{n} X_i = T_i$. Then, we have

$$\mathcal{T} \models \left(\exists \bar{X}\ \bigwedge_{i=1}^{n} X_i = T_i\right) \leftrightarrow \left(\exists \bar{X}\ \bigwedge_{i=1}^{n} X_i = T_i'\right) \text{ with } T_i \equiv \begin{cases} T_i & \text{if } \mathrm{v}(T) \\ T_i\sigma & \text{if } \mathrm{f}(T_i) \end{cases}$$

with $\sigma \equiv \prod_{k:\mathrm{v}(T_k)} [T_k \leftarrow X_k]$ and $\exists \bar{X}\ \bigwedge_{i=1}^{n} X_i = T_i'$ is solved and oriented.

We transform the solved formula $\exists Y\ X = Y \wedge Z = Y \wedge Y = f(Y)$ to the equivalent and oriented formula $Y = X \wedge Z = X \wedge X = f(Y)$. Replacing $X = f(Y)$ by $X = f(X)$ yields the representative formula.

Variables and equations that are linked to to the instantiations of free variables are called *reachable*. Adapting the notion of *reachability* [13] for an existentially quantified conjunction of equations in representative form allows to *purge* non-reachable equations and quantified variables.

Definition 6 (Purged Form). *A formula* $\exists \bar{X}\ \bigwedge_{i=1}^{n} X_i = T_i$ *in representative form is* purged *(or* finally solved*) if all variables in* $\bar{X}$ *and all equations* $X_i = T_i$ *are reachable: A variable* X *is* reachable *if* X *is a free variable or if* X *appears as an argument of a function term* T_i *in a reachable equation* $X_i = T_i$*. An equation* $X_i = T_i$ *is* reachable *if* X_i *is reachable.*

Any non-purged but representative formula can be transformed into an equivalent purged formula by eliminating unreachable equations and variables according to Maher's *uniqueness axiom (A3).*

Property 4. Consider a formatted formula $\exists \bar{X}\ \bigwedge_{i=1}^{n} X_i = T_i$, its sub-vector $\bar{X}'$ consisting of the reachable variables of $\bar{X}$, and its reachable equations $X_{i_j} = T_{i_j}$. Then we have $\mathcal{T} \models \left(\exists \bar{X}\ \bigwedge_{i=1}^{n} X_i = T_i\right) \leftrightarrow \left(\exists \bar{X}'\ \bigwedge_{j=1}^{k} X_{i_j} = T_{i_j}\right)$.

The representative formula $\exists YUW\ Y = X \wedge Z = X \wedge X = f(W) \wedge W = g(X, W) \wedge U = f(W)$ is equivalent to the purged formula $\exists W\ Z = X \wedge X = f(W) \wedge W = g(X, W)$.

4.2 Transforming to Representative Form and Purging in CHR

To transform a solved form to an equivalent oriented form, we apply program $\{o_1, o_2, o_3\}$ where existentially quantified variables are marked with CHR constraints $\texttt{exists}(X)$, free variables with $\texttt{free}(X)$, and equations $=$ are encoded as CHR constraints $\texttt{eq}$.

$$o_1 \;@\; \texttt{replace}(X, Y) \setminus X \;\texttt{eq}\; T \Leftrightarrow \mathrm{v}(T) \mid Y \;\texttt{eq}\; T$$

$$o_2 \;@\; \texttt{replace}(X, Y) \setminus Z \;\texttt{eq}\; X \Leftrightarrow Z \;\texttt{eq}\; Y$$

$$o_3 \;@\; \texttt{free}(Y) \wedge \texttt{exists}(X) \setminus Y \;\texttt{eq}\; X \Leftrightarrow \texttt{replace}(X, Y) \wedge X \;\texttt{eq}\; Y$$

Under *refined semantics*, rules o_1 and o_2 apply exhaustively for any generated CHR constraint $\texttt{replace}(X,Y)$ by rule o_3 which replaces an equation $Y \texttt{ eq } X$ between a free variable Y and an existentially quantified variable X with $X \texttt{ eq } Y$. Rules o_1 and o_2 update the representatives of all affected equalities for both function terms T attached to Y and for equations $Y \texttt{ eq } X$ between two variables.

To substitute non-representative argument variables and purge non-reachable variables and equations we apply program $\{p_1, p_2, p_3, p_4\}$ on the answer of program $\{o_1, o_2, o_3\}$. The purged (or finally solved) form is encoded by CHR constraints $\texttt{eq}'$ and $\texttt{exists}'$.

$$p_1 \ @ \ X \texttt{ eq } Y \setminus \texttt{free}(X) \Leftrightarrow \mathrm{v}(Y) \mid X \texttt{ eq}' \ Y \wedge \texttt{reach}(X)$$
$$p_2 \ @ \ X \texttt{ eq } T \setminus \texttt{free}(X) \Leftrightarrow \mathrm{f}(T) \mid \texttt{reach}(X)$$
$$p_3 \ @ \ \texttt{reach}(X) \setminus X \texttt{ eq } T \Leftrightarrow \mathrm{f}(T) \mid \mathrm{reachargs}(T,T') \wedge X \texttt{ eq}' \ T'$$
$$p_4 \ @ \ \texttt{reach}(X) \setminus \texttt{exists}(X) \Leftrightarrow \texttt{exists}'(X)$$

Rule p_1 saves equations between two free variables to the finally solved from. Both rules p_1 and p_2 mark free variables X that can lead to other reachable variables with a CHR constraint $\texttt{reach}(X)$. For a reachable variable X, rule p_3 propagates reachability to the arguments of the attached function term T by calling the built-in $\mathrm{reachargs}(T,T')$ which returns a function term T' with representative argument variables and marks the equation as reachable. Rule p_4 saves reachable existentially quantified variables X to the finally solved form.

4.3 Solving Algorithm

Our solving algorithm A for existentially quantified conjunction of non-flat equations $\exists \bar{Y} \ \bigwedge_{i=1}^{n} S_i = T_i$ consists of four sequentially executed parts.

(1) Transform $\exists \bar{Y} \ \bigwedge_{i=1}^{n} S_i = T_i$ to an equivalent existentially quantified conjunction of equations $\exists \bar{X} \ C_1$ in strict flat form by adding new existentially quantified variables (cf. [13] for details).
(2) Apply the refined UF+RT solver on C_1. If the solver terminates with *false* stop with an error, otherwise convert the solved CHR state to the solved form C_2 (cf. Section 3 for details).
(3) Transform $\exists \bar{X} \ C_2$ to an equivalent and oriented formula $\exists \bar{X} \ C_3$ by application of program $\{o_1, o_2, o_3\}$ (cf. Subsection 4.1).
(4) Transform $\exists \bar{X} \ C_3$ to an equivalent finally solved formula $\exists \bar{X}' \ C_4$ by application of program $\{p_1, p_2, p_3, p_3\}$ (cf. Subsection 4.1).

We can now state our second main contribution:

Theorem 2 (Almost-linear Tree Equation Solver With Existential Variables). *The time complexity of algorithm A to solve existentially quantified conjunctions of non-flat equations in theory $\mathcal{T}$ is almost-linear.*

Proof. Flattening can be done in linear time and space and both the number of new existentially quantified variables and the number of new equations are linear in the size of the non-flat existentially quantified conjunction of equations [13]. By Theorem 1 the second step takes almost-linear time for the refined UF+RT solver. Each programs $\{o_1, o_2, o_3\}$ and $\{p_1, p_2, p_3, p_4\}$ traverses the formula once in linear time. □

Our extended CHR solver implements algorithm A with optimal almost-linear time complexity and is available online (cf. link in Section 1).

Example 1. We apply algorithm A on the following formula with the free variable X:

$$\exists VWZ\; W = X \land f(X) = f(g(W,Z)) \land f(Z) = f(f(V)) \enspace .$$

(1) Flattening to an equivalent formula with additional existential quantified variables and equations in strict flat form yields

$$\exists VWZABCD\; W = X \land A = f(X) \land B = g(W,Z) \land A = f(B) \land \\ C = f(Z) \land D = f(V) \land C = f(D) \enspace .$$

(2) Application of the refined UF-RT solver on the conjunction of equations which returns a solved form

$$X = W \land W = g(W,V) \land Z = f(V) \land A = f(X) \land \\ B = W \land C = f(Z) \land D = Z \enspace .$$

(3) Orientation on the quantified and solved formula yields

$$\exists VWZABCD\; W = X \land X = g(W,Z) \land Z = f(V) \land A = f(X) \land \\ B = X \land C = f(Z) \land D = Z \enspace .$$

(4) Transforming in representative form and purging of unreachable variables and equations yields the concise final solved formula

$$\exists VZ\; X = g(X,Z) \land Z = f(V) \enspace .$$

5 Conclusion

We reconstructed Huet's tree equation solving algorithm for rational trees as a CHR solver for refined semantics with optimal almost-linear time complexity with improves on the quadratic complexity of [13]. To this end, we optimised the quadratic classic rational tree solver by combination with the almost-linear union-find solver. Our compact and highly concise code is shorter than implementations in other languages and even shorter than most formal expositions.

Moreover, we extended the CHR solver to solve existentially quantified conjunctions of non-flat equations in theory $\mathcal{T}$ in almost-linear time using the notion of reachability [13]. Our new definitions of solved, oriented, representative, and purged form are adapted to the union-find data-structure and yield a more explicit answer, e.g. $\exists X\ Y = f(X)$ instead of the finally solved form $\exists X\ Y = f(X) \wedge X = Y$ from [13]. To the best of our knowledge, this is the first CHR solver for existentially quantified conjunctions of non-flat equations with *almost-linear time complexity*.

As unification is known to be inherently sequential (cf. [4]) future work aims to study the declarative concurrency of the CHR solver when using the parallel CHR union-find implementation [6].

We aim to extend the solver with entailment and disentailment as the basis of an algorithm for solving arbitrary first-order formulas involving equations and inequations in CHR.

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
2. G. J. Duck, P. J. Stuckey, M. J. García de la Banda, and C. Holzbaur. Extending arbitrary solvers with Constraint Handling Rules. In *PPDP'03*, pages 79–90. ACM Press, 2003.
3. G. J. Duck, P. J. Stuckey, M. J. García de la Banda, and C. Holzbaur. The refined operational semantics of Constraint Handling Rules. In *ICLP 2004*, volume 3132 of *LNCS*, pages 90–104. Springer, 2004.
4. C. Dwork, P. C. Kanellakis, and J. C. Mitchell. On the sequential nature of unification. *J. Logic Programming*, 1(1):35–50, 1984.
5. T. Frühwirth. Theory and Practice of Constraint Handling Rules. *J. Logic Programming*, 37(1–3):95–138, 1998.
6. T. Frühwirth. Parallelizing union-find in Constraint Handling Rules using confluence. In *ICLP 2005*, volume 3668 of *LNCS*, pages 113–127. Springer, 2005.
7. T. Frühwirth. Specialization of concurrent guarded multi-set transformation rules. In *LOPSTR 2004*, volume 3573 of *LNCS*, pages 133–148. Springer-Verlag, 2005.
8. T. Frühwirth and S. Abdennadher. *Essentials of Constraint Programming.* Springer, 2003.
9. G. P. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *J. ACM*, 27(4):797–821, 1980.
10. K. Knight. Unification: a multidisciplinary survey. *ACM Comput. Surv.*, 21(1):93–124, 1989.
11. M. J. Maher. Complete axiomatizations of the algebras of finite, rational, and infinite trees. In *3rd Annual IEEE Symposium on Logic in Computer Science, LICS'88*, pages 348–357, Los Alamitos, CA, USA, 1988.
12. A. Martelli and G. Rossi. Efficient unification with infinite terms in logic programming. In *FGCS'84*, pages 202–209. ICOT, 1984.
13. M. Meister, K. Djelloul, and T. Frühwirth. Complexity of a CHR solver for existentially quantified conjunctions of equations over trees. In *Recent Advances in Constraints*, of *LNCS*. Springer, to appear.
14. T. Schrijvers and B. Demoen. The K.U.Leuven CHR system: implementation and application. In *CHR 2004, Selected Contributions*, volume 2004-01 of *Ulmer Informatik-Berichte*. Universität Ulm, Germany, 2004.
15. 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.
16. T. Schrijvers et al. The Constraint Handling Rules (CHR) web page, 2007. `http://www.cs.kuleuven.ac.be/~dtai/projects/CHR/`.
17. R. E. Tarjan and J. Van Leeuwen. Worst-case analysis of set union algorithms. *J. ACM*, 31(2):245–281, 1984.