Towards Term Rewriting Systems in Constraint Handling Rules

Coming to terms with jungles

Frank Raiser and Thom Frühwirth

Faculty of Engineering and Computer Sciences, University of Ulm, Germany
{Frank.Raiser|Thom.Fruehwirth}@uni-ulm.de

Abstract. Term rewriting systems are a formalism in widespread use, often implemented by means of term graph rewriting. In this work we present preliminary results towards an elegant embedding of term graph rewriting in Constraint Handling Rules with rule priorities (CHR^{rp}). As term graph rewriting is well-known to be incomplete with respect to term rewriting, we aim for sound jungle evaluation in CHR^{rp}. Having such an embedding available allows to benefit from CHR's online property and parallelization potential.

1 Introduction

Term rewriting is an important branch of computer science with applications in algebra, recursion theory, software engineering, and programming languages [1]. There is a wealth of known results available concerning term rewriting systems (TRSs).

Constraint handling rules (CHR) is a concurrent committed-choice constraint logic programming language consisting of guarded rules, which transform multisets of atomic formulas (constraints) into simpler ones until exhaustion [2]. Initially created for the development of constraint solvers [3] it has meanwhile grown to a general-purpose programming language [4, 5].

As CHR shares the basic property with TRSs of replacing left-hand sides by right-hand sides, several properties of TRSs have also been investigated in the context of CHR. The most important of these being confluence and termination. However, up to now there is no existing work on embedding a TRS in CHR. Despite their similarities there is a major difference in the way a TRS and a CHR program work, which makes this embedding non-trivial: a TRS can replace subterms of a term independent of how deeply nested the subterm is, whereas CHR replaces multisets of top-level constraints.

Many practical implementations of term rewriting actually perform term graph rewriting [6] instead, which is a sound, but incomplete, alternative to pure term rewriting. The lack of completeness is made up for with the efficiency of the rewriting process. As term graph rewriting can perform multiple term rewrite steps in one step there are even examples of exponential speedups, like the computation of Fibonacci numbers [7]. The reason for these speedups is that

term graph rewriting makes use of structure sharing, such that equal subterms only exist once in a term graph and the need for equality checking, therefore, is avoided. Considering, that term graph rewriting is based on graph transformations for which an embedding in CHR exists [8], this work focuses directly on embedding term graph rewriting into CHR as a means to achieve sound term rewriting.

The theory for term graph rewriting is based on jungles which are introduced in Sect. 2. We also introduce CHR^{rp} [9] there, which is a variant of CHR assigning priorities to rules. It is used in this work instead of plain CHR, as it greatly simplifies the process of updating structure sharing in term graphs without being as restrictive as the refined semantics for CHR. Section 2 further details the correspondence between term rewriting and term graph rewriting, before Sect. 3 presents our approach to embed term graphs with structure sharing in CHR^{rp}. It is shown there, that our proposed CHR^{rp} encoding of term graphs ensures a terminating and confluent computation of term graphs with a maximal amount of shared structures. We plan to use these normal form term graphs as a basis for performing term graph rewriting in CHR^{rp}, which is outlined in Sect. 4. In that section future work regarding jungle evaluation and properties of the CHR^{rp} implementation of term graph rewriting are outlined as well, before Sect. 5 concludes this work.

2 Preliminaries

The following preliminaries are taken from [7]:

Strings A^* denotes the set of all strings over some set A, including the empty string ε. $f^* : A^* \to B^*$ denotes the homomorphic extension of a function $f : A \to B$.

Abstract Reductions Let $\to$ be a binary relation on some set A.

We write $\to^+$ and $\to^*$ for the transitive and transitive-reflexive closure of $\to$, respectively. The n-fold composition of $\to$ ($n \geq 0$) is denoted by $\to^n$; in particular, $\to^0$ is the equality on A.

Some $a \in A$ is a *normal form* (w.r.t. $\to$) if there is no $b \in A$ with $a \to b$. A normal form a is called a *normal form of* $b \in A$ if $b \to^* a$.

We say $\to$ is *terminating* if there is no infinite chain $a_0 \to a_1 \to a_2 \to \ldots$

The relation $\to$ is *confluent* if for all $a, b_1, b_2 \in A$, ${b_1}^* \leftarrow a \to^* b_2$ implies $b_1 \to^* c^* \leftarrow b_2$ for some $c \in A$.

Terms and Substitution $T(X)$ denotes the set of all terms over the set X of variables. Terms can contain *function symbols* from the set Σ. Each function symbol f is associated with an arity $\mathtt{arity}(f) \geq 0$. A function symbol c with $\mathtt{arity}(c) = 0$ is called a *constant*.

A *substitution* $\sigma : T(X) \to T(Y)$, *rewrite rules*, and *term rewriting systems* are defined as usual.

2.1 Constraint Handling Rules with Rule Priorities

This section presents the syntax and operational semantics of constraint handling rules [2, 3]. Constraints are first-order predicates which we separate into *built-in constraints* and *user-defined constraints*. Built-in constraints are provided by the constraint solver while user-defined constraints are defined by a CHR program. A CHR program consists of CHR rules, for which three variants exist: simplification, propagation, and simpagation rules. As simpagation rules are the most general and can directly simulate the other two variants we consider only simpagation rules in this work.

There are different operational semantics available for CHR [4]. We chose to use CHR with rule priorities (CHR^{rp}) for this work, as it is most suitable to the underlying idea of establishing maximal term structure sharing before applying term graph rules. The remaining operational semantics are not as suitable for our work:

refined semantics The operational semantics found in most common CHR implementations is the so-called *refined semantics*. It is geared towards implementation issues and its major drawback in our case is that the application order of rules is fixed by their order of occurrence in the program text. In order to be able to generally embed term graph rewriting, however, we require a non-deterministic rule selection as it is available for term graph rewriting.

abstract semantics The abstract (or standard) operational semantics is the default operational semantics for CHR, which includes non-deterministic rule selection. However, as we want to ensure, that term graphs use maximal structure sharing before term graph rules are applied to them, additional effort would be required: it has to be guaranteed, that despite the non-deterministic rule selection term graph rules can only be applied after the corresponding term graphs provide for maximal structure sharing.

CHR^{rp} extends the abstract semantics with priorities for rules, such that rules with the same priority are still selected non-deterministically, but only when no other rules of higher priority can be applied. This allows us to split our rules for the term graph embedding into two classes: a high-priority class of rules responsible for ensuring maximal structure sharing and a low-priority class of rules corresponding to the embedded term graph rewriting rules.

In CHR^{rp} simpagation rules are of the form

$$\mathit{priority} :: \mathit{Rulename} \ @ \ H_1 \setminus H_2 \Leftrightarrow g \mid C$$

where *priority* is the *priority* of the rule, *Rulename* is an optional unique *identifier* of a rule, the *head* $H_1 \setminus H_2$ is a non-empty conjunction of user-defined constraints, the *guard* g is a conjunction of built-in constraints and the *body* C is a conjunction of built-in and user-defined constraints. Note that with respect to H_1, H_2, and C we mix the use of the terms conjunction, sequence, and multiset.

The operational semantics is based on an underlying *constraint theory* $\mathcal{D}$ for the built-in constraints and a *state*, which is a pair $\langle G, S, B, T \rangle$ where G is a *goal*,

i.e. a multiset of user-defined and built-in constraints, S is the CHR *constraint store*, B is the *built-in store*, and T is the propagation history [4]. Table 1 shows the possible state transitions for CHR^{rp} under the operational semantics of CHR with rule priorities, denoted as ω_p.

1. **Solve** $\langle \{c\} \uplus G, S, B, T\rangle_n \overset{\omega_p}{\mapsto}_P \langle G, S, c \wedge B, T\rangle_n$ where c is a built-in constraint.
2. **Introduce** $\langle \{c\} \uplus G, S, B, T\rangle_n \overset{\omega_p}{\mapsto}_P \langle G, \{c\#n\} \cup S, B, T\rangle_{n+1}$ where c is a CHR constraint.
3. **Apply** $\langle \emptyset, H_1 \cup H_2 \cup S, B, T\rangle_n \overset{\omega_p}{\mapsto}_P \langle \Theta(C), H_1 \cup S, B, T \cup \{t\}\rangle_n$ where P contains a rule of priority p of the form

$$p :: r \ @ \ H_1' \setminus H_2' \Leftrightarrow g \mid C$$

and a matching substitution Θ such that $\mathtt{chr}(H_1) = \Theta(H_1'), \mathtt{chr}(H_2) = \Theta(H_2'), \mathcal{D} \models B \rightarrow \exists_B(\Theta \wedge g)$, $\Theta(p)$ is a ground arithmetic expression and $t = \langle r, \mathtt{id}(H_1) + + \mathtt{id}(H_2)\rangle \notin T$. Furthermore, no rule of priority p' and substitution Θ' exists with $\Theta'(p') < \Theta(p)$ for which the above conditions hold.

Table 1. Transitions of ω_p

2.2 Jungle Evaluation and Term Rewriting

A survey on term graph rewriting can be found in [6], with additional details, especially considering jungle evaluation, in [7]. The following definitions and facts are taken from those two works.

It is well-known, how a term can be represented as a tree. The sharing of equal subterms, however, is not allowed in the usual tree structure. To this end, jungles are used, which are a specialization of hypergraphs:

Definition 1 (Hypergraph). *A* hypergraph $G = (V_G, E_G, \mathtt{att}_G, \mathtt{lab}_G)$ *consists of a finite set* V_G *of* nodes*, a finite set* E_G *of* hyperedges *(or* edges *for short), and a mapping* $\mathtt{lab}_G : E_G \to \Sigma$*, labeling hyperedges with function symbols and a mapping* $\mathtt{att}_G : E_G \to V_G^+$ *such that* $|\mathtt{att}_G(e)| = 1 + \mathtt{arity}(\mathtt{lab}_G(e))$.

Given $e \in E_G$ *with* $\mathtt{att}_G(e) = v_0 v_1 \ldots v_n$, $\mathtt{res}(e) = v_0$ *is called the* result node *and* $\mathtt{arg}(e) = v_1, \ldots, v_n$ *are called the* argument nodes*. We define* $\mathtt{indegree}_G(v) = |\{e \mid v \in \arg(e)\}|$ *and* $\mathtt{outdegree}_G(v) = |\{e \mid v = \mathtt{res}(e)\}|$

Let v_1, v_2 *be two nodes in a hypergraph* G*. Then* $v_1 >_G v_2$ *denotes that there is a non-empty path from* v_1 *to* v_2 *in* G*;* $v_1 \geq_G v_2$ *means* $v_1 >_G v_2$ *or* $v_1 = v_2$*.* G *is* acyclic *if there is no node* $v \in V_G$ *such that* $v >_G v$*.*

This allows us to define a jungle:

Definition 2 (Jungle). *A hypergraph* $G = (V_G, E_G, \mathtt{att}_G, \mathtt{lab}_G)$ *is a* jungle *if*

1. $\mathtt{outdegree}_G(v) \leq 1 \ \forall v \in V_G$,
2. *G is acyclic.*

When we consider ground terms represented as trees, then all leafs are constants. For jungles these constants become hyperedges with arity 1, i.e. hyperedges which are attached to a result node, but have no argument nodes. Conversely, if a node in a jungle is not a result node of an edge we treat it like a variable. It is easy to see that non-linear terms of a term rewriting system, i.e. terms in which a variable occurs multiple times, can be represented by jungles with one shared variable node per variable of the TRS.

Notation $\mathrm{VAR}_G = \{v \in V_G \mid \mathtt{outdegree}_G(v) = 0\}$ denotes the set of variables associated with a jungle G.

Example 1. Figure 1 shows an exemplary jungle used in a rule for computing Fibonacci numbers. Nodes are shown as black dots and hyperedges as rectangles. For an edge, the associated operation symbol is written inside the rectangle and the result node is given by a line without an arrow, whereas the argument nodes are given by arrows. In general, we assume that the order of arguments coincides with the left-to-right order of arrows in a figure.

In Fig. 1 the node r is a root node of the jungle and the node n is a variable node.

Also see the jungles in Fig. 3 for how jungles allow the sharing of common substructures.

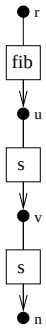

Fig. 1. Exemplary jungle G

To associate jungles with terms we define a mapping assigning terms to each node of a jungle:

Definition 3 (Term Representation Function). *Let G be a jungle. Then*

$$\mathtt{term}_G(v) = \begin{cases} v & \text{if } v \in \mathrm{VAR}_G, \\ \mathtt{lab}_G(e)(\mathtt{term}_G(v_1), \ldots, \mathtt{term}_G(v_n)) & \text{for the unique edge } e \text{ such} \\ & \text{that } \mathtt{att}_G(e) = vv_1 \ldots v_n \end{cases}$$

defines a function $\mathtt{term}_G : V_G \to T(\mathrm{VAR}_G)$.

The set $\mathtt{term}_G(V_G)$ *of all terms represented by a jungle G is denoted by* TERM_G.

Example 2. The terms represented by Fig. 1 are:

node	$\mathtt{term}_G$
r	`fib(s(s(n)))`
u	`s(s(n))`
v	`s(n)`
n	`n`

All terms represented by G, hence, are:
$\mathrm{TERM}_G = \{\mathtt{fib(s(s(n)))}, \mathtt{s(s(n))}, \mathtt{s(n)}, \mathtt{n}\}$

For the remainder of this work we require various morphisms between jungles, which have to satisfy the following definition:

Definition 4 (Jungle Morphism). *Let G, H be jungles. A* jungle morphism *$f : G \to H$ is a pair of mappings $f = (f_V : V_G \to V_H, f_E : E_G \to E_H)$ which preserves sources, targets, and labels, i.e. $\mathtt{att}_H \circ f_E = f_V^* \circ \mathtt{att}_G$ and $\mathtt{lab}_H \circ f_V = \mathtt{lab}_G$.*

A jungle morphism $f = (f_V, f_E)$ is injective (surjective) if and only if f_V and f_E are both injective (surjective).

Notation $\mathrm{ROOT}_G = \{v \in V_G \mid \mathtt{indegree}_G(v) = 0\}$ denotes the set of *roots* of a jungle G.

Analogous to [6] we equate a node v with the set of paths from a root node to v in order to get standard term graphs and avoid the usual isomorphism details. As we allow for multiple root nodes only the path from a specific root node to v is unique, and thus, we include paths from all root nodes to get a unique standard term graph.

As jungles can contain variables, which are represented as nodes that are not result nodes of an edge, every jungle morphism assigning such a node to a node in the target jungle induces a substitution:

Definition 5 (Induced Substitution). *Let $f : G \to H$ be a jungle morphism. Then the* induced substitution *$\sigma : T(\mathrm{VAR}_G) \to T(\mathrm{VAR}_H)$ is defined for all $x \in \mathrm{VAR}_G$ by*

$$\sigma(x) = \mathtt{term}_H(f_V(x))$$

Example 3. Figure 2 shows a jungle morphism g between two jungles G and H. The morphism is depicted by dotted arrows. The jungle H represents the ground term `fib(s(s(s(0))))` which is used to compute the third Fibonacci number. The jungle morphism g, which maps n to $g(n) = n'$, induces a substitution σ with $\sigma(n) = \mathtt{term}_H(n') = \mathtt{s(0)}$.

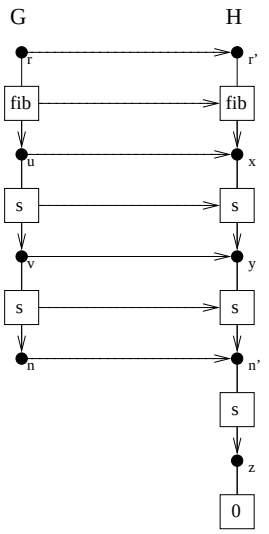

Fig. 2. Jungle morphism $g : G \to H$

3 Structure sharing in CHR^{rp}

This section explains how to encode jungles in CHR^{rp} and introduces a set of rules which implement structure sharing on these jungles. It is shown, that the rules ensure that the maximal amount of structures is shared.

3.1 Jungle Encoding in CHR^{rp}

Definition 6 (Jungle Encoding). *Let $G = (V_G, E_G, \mathtt{att}_G, \mathtt{lab}_G)$ be a jungle. Then G is encoded in CHR^{rp} as follows:*

1. *for all $v \in V_G$ introduce a unique variable X_v.*
2. *For each edge $e \in E_G$ with $\mathtt{res}(e) = v$ and $\mathtt{arg}(e) = v_1, \ldots, v_n$ add the constraint $\mathtt{Eq}(X_v, \mathtt{lab}_G(e)(X_{v_1}, \ldots, X_{v_n}))$*

Let $\mathtt{encode}(G)$ denote the set of $\mathtt{Eq}$ constraints for the CHR^{rp} encoding of G and let X_{V_G} denote the set of variables introduced for the encoding of G[1]*.*

Let X_v be a variable used in $\mathtt{encode}(G)$. Then

$$\mathtt{term}(X_v) = \begin{cases} v & \text{if } \nexists\, \mathtt{Eq}(X_v, \ldots) \in \mathtt{encode}(G) \\ op(\mathtt{term}(X_{v_1}), \ldots, \mathtt{term}(X_{v_k}))) & \text{if } \exists\, \mathtt{Eq}(X_v, op(X_{v_1}, \ldots, X_{v_k})) \\ & \quad \in \mathtt{encode}(G) \end{cases}$$

defines a function $\mathtt{term} : X_{V_G} \to T(X_{V_G})$.

[1] Note that for each variable $X_v \in X_{V_G}$ there is at most one $\mathtt{Eq}(X_v, \_) \in \mathtt{encode}(G)$ due to $\mathtt{outdegree}(v) \leq 1$ (Def. 2)

Example 4. Consider again the jungle H from Fig. 2. It's encoding in CHR^{rp} is: $\mathtt{Eq}(X_{r'}, \mathtt{fib}(X_x)), \mathtt{Eq}(X_x, \mathtt{s}(X_y)), \mathtt{Eq}(X_y, \mathtt{s}(X_{n'})), \mathtt{Eq}(X_{n'}, \mathtt{s}(X_z)), \mathtt{Eq}(X_z, \mathtt{0})$.

The following lemma ensures, that the set of terms represented by $\mathtt{encode}(G)$ via $\mathtt{term}$ is the same as the set of terms represented by G via $\mathtt{term}_G$:

Lemma 1 (Encoding preserves terms). *For an encoding* $\mathtt{encode}(G)$ *of a jungle* $G = (V_G, E_G, \mathtt{att}_G, \mathtt{lab}_G)$ *it holds that:*

$$\forall X \in X_{V_G} : \mathtt{term}(X) \in \mathrm{TERM}_G \tag{1}$$

$$\forall t \in \mathrm{TERM}_G \;\; \exists X \in X_{V_G} : t = \mathtt{term}(X) \tag{2}$$

Proof. Proof for (1) by structural induction:
if $\nexists\, \mathtt{Eq}(X_v, \ldots) \in \mathtt{encode}(G)$ *this implies by Definition 6(1) that* X_v *corresponds to a node* $v \in \mathrm{VAR}_G$*, and thus,* $\mathtt{term}(X_v) = v = \mathtt{term}_G(v) \in \mathrm{TERM}_G$.
if $\exists\, \mathtt{Eq}(X_v, op(X_{v_1}, \ldots, X_{v_k})) \in \mathtt{encode}(G)$ *this implies the existence of an edge* $e \in E_G$ *with* $res_G(e) = v$, $\mathtt{lab}_G(e) = op$ *and* $arg_G(e) = v_1, \ldots, v_k$*. The term* $op(\mathtt{term}(X_{v_1}), \ldots, \mathtt{term}(X_{v_k}))$*, thus, equals the term (Def. 3)* $\mathtt{lab}_G(e)(\mathtt{term}_G(v_1), \ldots, \mathtt{term}_G(v_k))$ *and is, therefore, contained in* TERM_G.

Each term in TERM_G *corresponds to a node* $v \in V_G$ *to which a variable* $X_v \in X_{V_G}$ *is associated. Hence, the proof of (2) is another structural induction analogous to the above.* □

3.2 Structure sharing

The idea of structure sharing is that identical subterms can share the same nodes and edges in a jungle. This cuts down on the space usage of an encoded term, as well as allowing to apply a term rewriting rule to all occurrences of the shared subterm in one step. Based on a lemma from [7] this leads us to the basic idea of how to embed term graph rewriting in CHR^{rp}: Every jungle G is first transformed into a jungle $\overline{G}$ representing the same terms, but for which its subterm structures are maximally shared. It is then known, that for each application of a term graph rewriting rule to the jungle G the rule also applies to the transformed jungle $\overline{G}$.

Using this property of structure sharing we can provide for a CHR^{rp} embedding which avoids the previously mentioned problem of having to detect subterm equality. Whenever two subterms are equal this is trivially seen from the corresponding jungle nodes and edges being shared. The remaining part of this section, therefore, details how structure sharing can be enforced in CHR^{rp}, with the following definitions being adapted from [6]:

Definition 7 (Collapsing). *Given two jungles* G *and* H*,* G collapses *to* H *if there is a jungle morphism* $f : G \to H$ *such that* $f_V(\mathrm{ROOT}_G) = \mathrm{ROOT}_H$ *and* $\mathtt{term}_G(f_V(\mathrm{ROOT}_G)) = \mathtt{term}_H(\mathrm{ROOT}_H))$*. This is denoted by* $G \succeq H$*, or if the morphism is non-injective, by* $G \succ H$*. The latter kind of collapsing is said to be* proper.

Given the notion of collapsing jungles we can further identify the jungles which are fully collapsed, i.e. to which no more proper collapsing steps are applicable:

Definition 8 (Tree, Fully Collapsed). *A jungle G is a* tree *if there is no H with $G \prec H$, while G is* fully collapsed *if there is no H with $G \succ H$.*

The following alternative definition of a fully collapsed jungle is given in [7] and is independent from the notion of collapsing via a jungle morphism:

Definition 9 (Fully Collapsed, Alternative Definition). *A jungle G is called* fully collapsed *if* $\mathtt{term}_G$ *is injective.*

Example 5. Figure 3 shows a jungle which occurs during the computation of $fib(3)$ representing the recursive computation $fib(3) = (fib(1)+fib(0))+fib(1)$. The duplicate occurrence of $fib(1)$ can then be optimized by structure sharing. The collapse step shown in Fig. 3 eliminates one hyperedge representing the term `fib(1)` by reusing another hyperedge which represents the same term. The corresponding jungle morphism is the identity morphism, except for the mapping depicted by the dashed arrows. Overall, structure sharing leads to a linear computation of Fibonacci numbers, as opposed to the naive exponential computation.

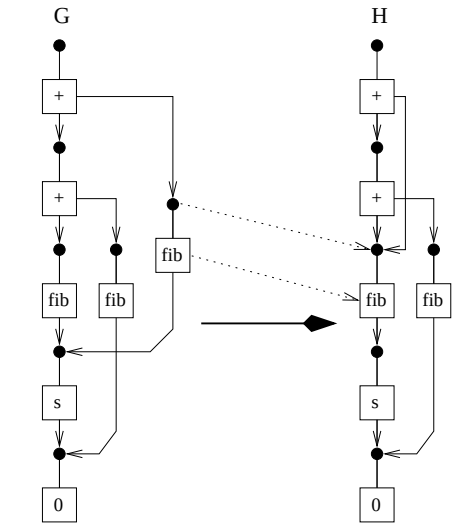

Fig. 3. Collapsing of a jungle

The process of collapsing a jungle (called *Folding* in [7]) is instrumented via a set of folding rules according to the following definition:

Definition 10 (Folding Rule). *Let $op \in \Sigma$ be a function symbol such that* $\texttt{arity}(op) = k \geq 0$.

The folding rule *for op is given by a pair $(L \hookleftarrow K \xrightarrow{b} R)$ of jungle morphisms as depicted in Fig. 4 ("x = y" indicates that b identifies the roots of K; note that L and R have no variables if op is a constant, $\hookleftarrow$ denotes an inclusion morphism). $\mathcal{F}$ denotes the set of folding rules for Σ.*

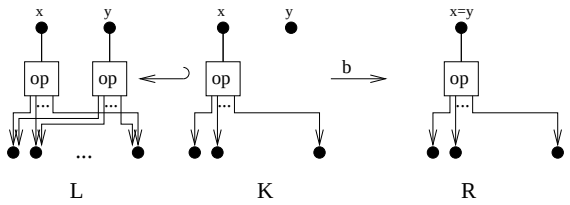

Fig. 4. Folding rule for $op \in \Sigma$

We now port this instrumentation to jungles encoded in CHR by `encode`. The generated folding rules are assigned a static priority of 1. This enforces our basic idea of fully collapsing a jungle, before applying rules of the actual TRS to it.

Definition 11 (CHR folding rule). *Let $op \in \Sigma$ be a function symbol such that* $\texttt{arity}(op) = k \geq 0$. *Then we define the following CHR folding rule:*

$$1 :: fold\_op \ @ \ \texttt{Eq}(X, op(X_1, \ldots, X_k)) \setminus \texttt{Eq}(Y, op(X_1, \ldots, X_k)) \Leftrightarrow X = Y.$$

$\mathcal{P}^{\mathcal{F}}$ denotes the CHR program consisting of all CHR folding rules for Σ.

Note that due to CHR's built-in support for syntactic equivalence we can also use the following single fold rule instead:

$$1 :: fold \ @ \ \texttt{Eq}(X, Term) \setminus \texttt{Eq}(Y, Term) \Leftrightarrow X = Y$$

A single folding step is defined in [7] as follows. In CHR such a folding step coincides with the application of a folding rule, as the following theorem shows.

Definition 12 (Folding Step). *Let G be a jungle. A* folding step $G \underset{\mathcal{F}}{\Rightarrow} H$ *from G to a hypergraph H is constructed as follows:*

- *Find a morphism $g : L \to G$ for some folding rule $(L \hookleftarrow K \xrightarrow{b} R)$ such that g_E is injective.*
- *Obtain D from G by removing $g_E(e)$, where e is the unique edge in $L \setminus K$.*
- *Obtain H from D by identifying $g_V(x)$ and $g_V(y)$, where x and y are the roots in L.*

Fact 1 (Folding Steps Preserve Jungles [7]) *Given a jungle G and a folding step $G \underset{\mathcal{F}}{\Rightarrow} H$, H is a jungle, too.*

Theorem 1 (CHR folding is sound and complete). *For a $op \in \Sigma$ and a jungle G the following steps are equivalent:*

1. $G \underset{\mathcal{F}}{\Rightarrow} H$
2. $\mathtt{encode}(G) \overset{\omega_p}{\rightarrowtail}_{\mathcal{P}^{\mathcal{F}}} \mathtt{encode}(H)$

Proof. (1) ⇒ (2)*:*
Let $g : L \to G$ be the required morphism for the folding rule corresponding to the function symbol $op \in \Sigma$ with g_E being injective. This morphism extends to a matching for the head of the corresponding CHR folding rule for op. The two edges in L directly correspond to the two Eq *constraints in the head of the CHR rule. As g_E is injective there exist two edges e_1 and e_2 in $g_E(E_L)$ with $\mathtt{lab}_G(e_1) = \mathtt{lab}_G(e_2) = op$, $\mathtt{res}_G(e_1) = v, \mathtt{res}_G(e_2) = w$, and $\mathtt{att}_G(e_1) = \mathtt{att}_G(e_2) = v_1, \ldots, v_k$. By Definition 6 there also exist corresponding constraints $\mathtt{Eq}(X_v, op(X_{v_1}, \ldots, X_{v_k}))$ and $\mathtt{Eq}(X_w, op(X_{v_1}, \ldots, X_{v_k}))$. Therefore, these two constraints match the two* Eq *constraints in the head of the corresponding CHR folding rule and the rule can be applied.*

D is obtained from G by removing $g_E(e)$ with e being the unique edge in $L \backslash K$. Let w.l.o.g. $g_E(e) = e_2$ and $X_w = Y$ be the substitution used for the matching of the CHR folding rule's head. Then the application of the simpagation rule removes the Eq *constraint corresponding to e_2, as demanded for the generation of D.*

Finally, H is obtained by identifying $g_V(x)$ and $g_V(y)$ where x and y are the roots in L. As defined by the edges e_1 and e_2 it follows that $g_V(x) = v$ and $g_V(y) = w$. By the implied matching the variables X and Y in the head of the CHR folding rule are bound to the variables X_v and X_w. Therefore, the application of the rule unifies X_v with X_w as required by Definition 12.
(2) ⇒ (1)*:*
This proof is mostly analogous to the previous argumentation: The CHR matching induces the required morphism g where the injectivity is guaranteed by the multiset semantics of CHR. Additionally, we have to show that applying a CHR folding rule actually results in a state which is an encoding of a jungle. This can, however, be seen from what such a rule does. The encoded graph has to contain two edges with the same label and argument nodes and different result nodes. As one of these edges is removed and it's result node identified with the result node of the other edge the result is again a jungle with the resulting state being its encoding modulo variable renaming. □

Example 6. Consider again the folding step depicted in Fig. 3 and let
$\mathtt{encode}(G) = \mathtt{Eq}(X_r, +(X_{v_1}, X_{v_2})),$
$\mathtt{Eq}(X_{v_1}, +(X_{v_3}, X_{v_4})), \mathtt{Eq}(X_{v_2}, \mathtt{fib}(X_{v_5})),$
$\mathtt{Eq}(X_{v_3}, \mathtt{fib}(X_{v_5})), \mathtt{Eq}(X_{v_4}, \mathtt{fib}(X_{v_6})),$
$\mathtt{Eq}(X_{v_5}, \mathtt{s}(X_{v_6})), \mathtt{Eq}(X_{v_6}, \mathtt{0}).$

The CHR folding rule is defined as
$1 :: fold \ @ \ \texttt{Eq}(X, Term) \setminus \texttt{Eq}(Y, Term) \Leftrightarrow X = Y$
and can be applied to the Eq constraints for X_{v_2} and X_{v_3} leading to the following CHR state:
$\texttt{encode}(H) = \texttt{Eq}(X_r, +(X_{v_1}, X_{v_2})),$
$\texttt{Eq}(X_{v_1}, +(X_{v_2}, X_{v_4})), \texttt{Eq}(X_{v_2}, \texttt{fib}(X_{v_5})),$
$\texttt{Eq}(X_{v_4}, \texttt{fib}(X_{v_6})), \texttt{Eq}(X_{v_5}, \texttt{s}(X_{v_6})), \texttt{Eq}(X_{v_6}, 0).$

Using Theorem 1 several properties of jungle folding can be transferred to $\mathcal{P}^{\mathcal{F}}$:

Fact 2 (Folding Steps Preserve Terms [7]) *Let $G \underset{\mathcal{F}}{\Rightarrow} H$ be a folding step. Then $\mathrm{TERM}_G = \mathrm{TERM}_H$.*

Corollary 1 (CHR folding preserves terms). *The application of a CHR folding rule $\texttt{encode}(G) \overset{\omega_p}{\rightarrowtail}_{\mathcal{P}^{\mathcal{F}}} \texttt{encode}(H)$ preserves the terms represented by the encoded jungle G.*

Proof. A direct consequence of Fact 2 and Theorem 1. □

Fact 3 ([7]) $\underset{\mathcal{F}}{\Rightarrow}$ *is terminating and confluent.*

Corollary 2 (CHR folding is terminating and confluent). $\overset{\omega_p}{\rightarrowtail}_{\mathcal{P}^{\mathcal{F}}}$ *is terminating and confluent.*

Proof. This follows directly from the soundness and completeness result in Theorem 1 and Fact 3. □

Following the idea of fully collapsing jungles by the application of folding rules, we transfer the following fact to CHR:

Fact 4 ([7]) *A jungle G is fully collapsed if and only if there is no folding step $G \underset{\mathcal{F}}{\Rightarrow} H$.*

Corollary 3 (fully collapsed with CHR folding). *Let G be a jungle with encoding $\texttt{encode}(G)$. G is fully collapsed if and only if there is no rule in $\mathcal{P}^{\mathcal{F}}$ applicable to $\texttt{encode}(G)$.*

Proof. This is a direct consequence of Thm. 1 and Fact 4.

Corollary 4 (CHR folding fully collapses). *Let G be a jungle with encoding $\texttt{encode}(G)$. Then there exists a terminating computation $\texttt{encode}(G) \overset{\omega_p{}^*}{\rightarrowtail}_{\mathcal{P}^{\mathcal{F}}} \texttt{encode}(H)$, such that the jungle H is fully collapsed.*

Proof. This is a direct consequence of Corollary 2 and Corollary 3. □

4 Discussion and Future Work

Targeting term graph rewriting instead of term rewriting allows us to avoid equivalence problems occurring with non-linear TRSs. A non-linear TRS allows the usage of the same variable multiple times on the left-hand side in order to express equal subterms. Considering a direct approach of flattening a term into a linear number of CHR constraints and associating a variable to each subterm has shown to be problematic in terms of these non-linear equalities.

One possible approach is to compute equality of subterms eagerly, similar to the structure sharing presented in this work. However, when the structures are not shared, but the constraint store only knows that two structures are equal a rewrite rule could change only one of the structures. This leads to the constraint store still modeling equivalence between the structures, and thus, a recomputation is required to regain a consistent store.

Another possibility is to include checking equivalent subterms in guards for non-linear rules resulting in the following kind of rules:

$$c(\ldots, X, \ldots, Y, \ldots), H \Leftrightarrow test\_eq(X, Y) \wedge G \mid B$$

Technically however, these constraints are not built-in, as they have to inspect the constraint store and CHR only allows built-in constraints in guards. Hence, the computation of these equality checks can be triggered by additional propagation rules according to the following scheme:

$$\begin{aligned} c(\ldots, X, \ldots, Y, \ldots), H &\Rightarrow test\_eq(X, Y, R) \\ c(\ldots, X, \ldots, Y, \ldots), test\_eq(X, Y, 1), H &\Leftrightarrow G \mid B \end{aligned}$$

This approach, however, is targeted towards the refined semantics of CHR, as a non-deterministic execution model resembles eager computation – as all propagation rules may fire first leading to the computation of all possible equalities.

Using Adaptive CHR [10] for eagerly computing equivalent subterms is another approach we plan to investigate in the future. With Adaptive CHR the equivalence of subterms also contains a justification, such that in the case of the replacement of a term in only one of the subterms the justification is violated and equivalence is recomputed on demand. While term graph rewriting is incomplete w.r.t. pure term rewriting an embedding in Adaptive CHR can achieve completeness at the cost of the efficiency granted by term graph rewriting.

The collapsing process detailed above is a necessary prerequisite for embedding term graph rewriting in CHR. The next step is the application of jungle evaluation rules to a jungle in order to simulate one or more term rewriting steps. This application is based on general graph transformations using the double pushout approach [11]. An embedding for graph transformations in CHR has already been investigated in [8].

However, the matching of left-hand sides of jungle evaluation rules to host jungles has to be injective in CHR due to its multiset semantics. Consider the jungle evaluation rule in Fig. 5, which represents the term rewriting rule $+(0, 0) \to 0$. As we take care that host jungles are fully collapsed, the left-hand side of the rule

has to use a non-injective matching for the two 0-edges to a shared occurrence of such a 0-edge.

We plan to investigate the possibility of collapsing the jungles occurring in a jungle evaluation rule in order to enforce an injective matching. Figure 6 shows how the resulting jungle evaluation rule for the rule in Fig. 5 looks like. In CHR this can easily be realized, by using each of the jungles as input to $\mathcal{P}^{\mathcal{F}}$ and use the collapsed output jungle for the construction of the corresponding CHR^{rp} rule. However, additional investigations are required to ensure, that using these collapsed rules is sound and complete with respect to the original rules being applied to fully collapsed jungles.

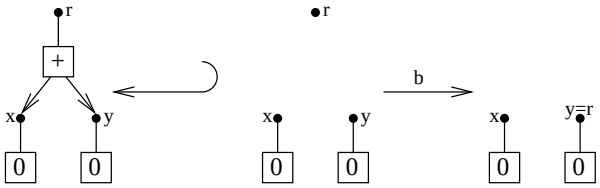

Fig. 5. Jungle evaluation rule

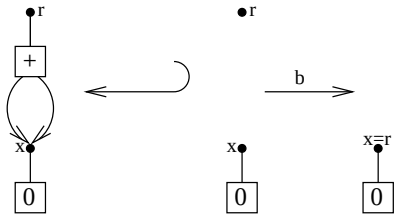

Fig. 6. The collapsed jungle evaluation rule

With a guaranteed injective matching the results from [8] can then be reused in order to perform term graph rewriting in CHR^{rp}. Slight adjustments will be necessary to account for the possible non-injectivity of b and due to the structure sharing idea no edges – except for the one representing the topmost term which is replaced by the rule – must be removed. This could result in a possibly large proportion of the constraint store being garbage left over from rule applications, and thus, requires the addition of cleanup rules. This garbage problem also conflicts with confluence, which is solved by considering *pointed reductions* in [6]. We expect to get a cleaner result for confluence modulo garbage by applying the results on observable confluence [12] in CHR.

5 Conclusion

In this paper we provide a basis for embedding term graph rewriting in Constraint handling rules with rule priorities (CHR^{rp}). We presented how jungles

are representing term graphs, and how these jungles can be encoded in CHR^{rp} such that the encoding is term preserving. We then introduced the concepts of structure sharing and fully collapsing a jungle, for which we proposed means to achieve them in a sound and complete, as well as terminating and confluent, way in CHR^{rp}. Furthermore, we outlined initial ideas for the remaining part of the embedding of term graph rewriting in CHR^{rp}.

References

1. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press, New York, NY, USA (1998)
2. Frühwirth, T., Abdennadher, S.: Essentials of Constraint Programming. Springer-Verlag (2003)
3. Frühwirth, T.: Theory and practice of constraint handling rules. Journal of Logic Programming, Special Issue on Constraint Logic Programming **37**(1-3) (October 1998) 95–138
4. Sneyers, J., Weert, P.V., Schrijvers, T., De Koninck, L.: As time goes by: Constraint Handling Rules — A survey of CHR research between 1998 and 2007. Submitted to Journal of Theory and Practice of Logic Programming (2008)
5. Frühwirth, T.: Constraint handling rules. to appear (2008)
6. Plump, D.: Term graph rewriting. In: Handbook of Graph Grammars and Computing by Graph Transformations. Volume 1., World Scientific (1997) 3–61
7. Hoffmann, B., Plump, D.: Implementing term rewriting by jungle evaluation. Informatique Théorique et Applications **25** (1991) 445–472
8. Raiser, F.: Graph Transformation Systems in CHR. In Dahl, V., Niemelä, I., eds.: Logic Programming, 23rd International Conference, ICLP 2007. Volume 4670 of Lecture Notes in Computer Science., Porto, Portugal, Springer-Verlag (September 2007) 240–254
9. De Koninck, L., Schrijvers, T., Demoen, B.: User-definable rule priorities for CHR. In: PPDP '07: Proceedings of the 9th ACM SIGPLAN international conference on Principles and practice of declarative programming, New York, NY, USA, ACM (2007) 25–36
10. Wolf, A.: Adaptive constraint handling with CHR in java. In: Principles and Practice of Constraint Programming, 7th International Conference, CP 2005. (2001) 256–270
11. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer-Verlag (2006)
12. Duck, G.J., Stuckey, P.J., Sulzmann, M.: Observable confluence for constraint handling rules. In Dahl, V., Niemelä, I., eds.: Logic Programming, 23rd International Conference, ICLP 2007. Volume 4670 of Lecture Notes in Computer Science., Porto, Portugal, Springer-Verlag (September 2007) 224–239