Demand-driven Normalisation for ACD Term Rewriting

Leslie De Koninck[1], Gregory J. Duck[2], and Peter J. Stuckey[2]

[1] Department of Computer Science, K.U.Leuven, Belgium
Leslie.DeKoninck@cs.kuleuven.be
[2] NICTA Victoria Laboratory, Australia
{gjd,pjs}@cs.mu.oz.au

Abstract. ACD Term Rewriting (ACDTR) is term rewriting modulo associativity, commutativity, and a limited form of distributivity called conjunctive context. Previous work presented an implementation for ACDTR based on bottom-up eager normalisation, extended to support the conjunctive context. This paper investigates the possibility of using a demand-driven normalisation strategy for ACDTR. Again, dealing with the conjunctive context proves to be challenging. The alternative normalisation strategy is compared with the current form of eager normalisation and potential further improvements on the strategy are investigated.

1 Introduction

ACD Term rewriting (ACDTR) [1] is term rewriting modulo the equational theory E consisting of the following equivalences:

$$\begin{array}{rc} (\textit{associativity}) & (X \circ Y) \circ Z \approx_E X \circ (Y \circ Z) \\ (\textit{commutativity}) & X \circ Y \approx_E Y \circ X \\ (\textit{distributivity}) & P \wedge f(Q_1, \ldots, Q_i, \ldots, Q_n) \approx_E P \wedge f(Q_1, \ldots, P \wedge Q_i, \ldots, Q_n) \end{array}$$

for any *associative commutative* (AC) operator $\circ$, functor f/n and $i \in [1..n]$. ACDTR *simplification* rules are of the form "$H \iff B$" where H and B are terms. These rules rewrite terms matching H into B. In [1], rules may have guards; these are not considered in this paper to simplify the presentation. The distributivity property is used by *simpagation* rules of the form "$C \setminus H \iff B$" in which C is matched with terms in the *conjunctive context* (CC) of H, that is, the terms that appear conjoined with a superterm of H.

ACDTR is implemented in the Cadmium system [2] which is used in the G12 project [3] to map high-level constraint models onto low-level executable ones. ACDTR subsumes Constraint Handling Rules (CHR) [4] and so it inherits the latter's applications. The Cadmium system uses bottom-up eager normalisation, which is common for (AC) term rewriting, but incomplete because of the conjunctive context. Indeed, when normalising from the bottom up, the CC of a term may not be in normal form. In [2], this problem is dealt with by an event mechanism that causes conjuncts to be renormalised in case their CC changes

in a relevant way. This event mechanism is an improvement over an earlier naive renormalisation policy described in [1]. This paper presents a top-down, demand-driven (lazy) normalisation strategy for ACDTR. First, an example:

Example 1. Consider the rule "$\mathtt{f}(X) \iff \mathtt{a}$" and *goal* term $\mathtt{f}(\mathtt{a}(\mathtt{complex}, \mathtt{term}))$. The argument of the $\mathtt{f}/1$ term is not relevant for the above rule. However, when using eager bottom-up normalisation, this argument is first normalised, but finds itself being discarded later on. Lazy normalisation avoids this. □

When compared to the Cadmium system [2], the present work differs as follows: (1) normalisation proceeds from the top down and in a demand-driven way; (2) rules that rewrite AC terms are considered as multi-headed rules that apply to any term matching one of the heads, provided terms matching the remaining heads can be found amongst its siblings. E.g. the rule "$\mathtt{a} + \mathtt{b} \iff \mathtt{c}$" is considered to apply to $\mathtt{a}/0$ and $\mathtt{b}/0$ terms rather than to $+$ terms; (3) we use an *active* conjunctive context as an alternative for the events of [2]. The idea is that the active term (i.e. the term being normalised) looks for terms that can be rewritten given that it is part of their conjunctive context.

While the implementation in [2] is based on techniques common in (AC) term rewriting, extended to support the conjunctive context, the principles behind the implementation presented in this paper are largely inspired by implementation techniques for CHR. In particular, the active CC and active AC operands are inspired by how applicable rule instances are found in CHR by means of an active constraint. The main contribution of this paper is that it shows the feasibility of a demand-driven normalisation policy for ACD term rewriting, with a discussion of the challenges such a policy introduces, in particular w.r.t. the CC.

2 Preliminaries

We assume some familiarity with term rewriting, see [5]. We consider AC terms to be flattened, e.g. $A \circ (B \circ C)$ with $\circ$ an AC operator, is represented as $\circ(A, B, C)$ and similar for any nesting of AC terms. We say that a rule "$H_1 \wedge \ldots \wedge H_m \setminus H_{m+1} \circ \ldots \circ H_n \iff B$" *applies* to a term S, subterm of the goal term T, if S matches with one of the heads $H_1, \ldots, H_n$ (modulo AC) and terms matching the remaining heads can be found among the subterms of T, such that the following conditions hold: (1) all terms matching the heads are different, (2) if $m + 1 < n$, the terms that match the heads $H_{m+1}, \ldots, H_n$ have a common parent which is a $\circ$ term, and (3) the terms matching the heads $H_1 \ldots, H_m$ appear in conjunction with $H_{m+1}, \ldots, H_n$ or one of their ancestors. We define that a rule *rewrites* a term S if the above conditions hold, and S matches one of the heads $H_{m+1}, \ldots, H_n$. In analogy with CHR terminology, we refer to the heads $H_1, \ldots, H_m$ as the *kept* heads, and to $H_{m+1}, \ldots, H_n$ as the *removed* heads.

A term S, subterm of the goal term T, may depend on this goal term in that some rules only apply to S or its subterms because it is part of T. The *conjunctive context* of a term T is the set of all terms that appear in conjunction with (a superterm of) T. The *relevant context* of a term T is its conjunctive context,

plus its siblings if T's parent is an AC term. For example, given the goal term $\mathtt{f}(\mathtt{a} \wedge \mathtt{g}(\mathtt{b} + \mathtt{c}) \wedge \mathtt{d})$, the conjective context of subterm $\mathtt{b}$ is $\{\mathtt{a}, \mathtt{d}\}$ and the relevant context is $\{\mathtt{a}, \mathtt{c}, \mathtt{d}\}$. The relevant context contains exactly those terms that are relevant for enabling rule applications. In [2], only the CC of a term is important. Here, we consider rules of the form "$H_1 \wedge \ldots \wedge H_m \setminus H_{m+1} \circ \ldots \circ H_n \iff B$" with $\circ$ an AC operator to rewrite any term matching one of the heads $H_{m+1}, \ldots, H_n$. In contrast, in [2], such a rule is considered to rewrite a $\circ$ term. The view we take here allows for more fine-grained on-demand rewrites during matching. Finally, we define the *inverse conjunctive context* of a term T as the set of terms whose conjunctive context contains T.

3 Demand-driven Normalisation

Now we define what it means for a term to be in *normal form* and *toplevel normal form*. The toplevel normal form roughly corresponds to the (weak) head normal form in the lambda calculus and is less restrictive than the normal form.

Definition 1 (Normal form). *A term S, subterm of the goal term T, is in normal form if no rule rewrites S or its subterms.*

Definition 2 (Toplevel normal form). *A term $T = f(T_1, \ldots, T_n)$ has toplevel functor f/n. A term is in toplevel normal form if further normalisation of the term does not change its toplevel functor.*

The *status* of a term is either *normalised*, *toplevel normalised* or *unnormalised*. If a term's relevant context changes, its status may also change. E.g. given the rules "$\mathtt{f}(X) \wedge \mathtt{u} \iff X$" and "$\mathtt{t} \setminus \mathtt{a} \iff \mathtt{b}$" then term $\mathtt{f}(\mathtt{a})$ has status *normalised* as part of the goal term $\mathtt{f}(\mathtt{a}) \wedge \mathtt{n}$ whereas it has status *unnormalised* as part of the goal term $\mathtt{f}(\mathtt{a}) \wedge \mathtt{u}$ and status *toplevel normalised* as part of the goal term $\mathtt{f}(\mathtt{a}) \wedge \mathtt{t}$. We use the normalisation status of a term to decide if a rule can apply to a given term without having to normalise it completely. For example, a rule "$\mathtt{f}(X) \iff \mathtt{a}$" cannot be applied to a term $\mathtt{g}(\mathtt{b})$ if this term is in toplevel normal form. In particular, we do not need to normalise its argument ($\mathtt{b}$).

We now propose a rewrite strategy that ensures a term is in toplevel normal form. After applying it (sequentially) to all operands of an AC term, these operands are all in toplevel normal form. This is non-trivial as rewriting an AC operand changes the relevant context of its siblings, and so a term that was in toplevel normal form before, may no longer be so after such a context change.

We first describe how a given subject term is matched with a pattern term. We distinguish between the case that the pattern is a *linear* variable, a *nonlinear* variable, and a non-variable term. Variables are linear if they appear only once in the rule heads. In case of a linear variable, a match is trivially found and we assert a binding between the variable and the term with which it is matched. In case of a nonlinear variable, we first check if we already have a binding for this variable. If so, we match the term in question with the term bound to the variable (modulo AC). Otherwise, we normalise the term because at that point, it is not yet known

which terms the variable need to be matched with further on. Therefore, we take a safe approach and normalise the term. While this deviates from lazy matching, there is no unique lazy way in general to match with nonlinear variables anyway: we can apply rewrites to any of the terms being matched in case they are not equal. Finally, in case of matching with a non-variable term, we check whether the functors of subject and pattern correspond. If so, we continue by matching the arguments of subject and pattern. Otherwise, we try to rewrite the subject term and if this succeeds, we try to match it with the pattern again.

To normalise a term, we first ensure it is in toplevel normal form by exhaustively applying rules to it. Next, we traverse all the term's arguments and recursively normalise them. A rule application starts with a matching phase, followed by a rewrite step if successful. First, we match the *active* term with a rule head, applying rules to its subterms if necessary. If the rule head is a kept head, we continue by looking for terms matching the rule's removed heads, and then for terms matching the remaining kept heads. The latter are in the CC of the terms matching the removed heads. Otherwise, we first match with the remaining removed heads, and then with the kept heads. During matching, no rules are applied to any term other than the subterms of the active term.

4 Optimisation

We have a prototype implementation of the described ideas, consisting of a Prolog front-end responsible for program analysis and preprocessing, and a Java back-end that interprets an internal representation of the program rules. We now present some optimisations that have been implemented.

Variables in a rule's body that also appear in its head, are bound to terms during matching. After a rule firing, these terms may appear in a new context which affects their normalisation status. Sometimes we can keep the normalisation status, namely if no terms are added to their relevant context. This optimisation is a generalisation of the conjunction collector optimisation of [2].

When terms are duplicated, we can either copy their representation, or use a shared representation for the duplicates. In standard term rewriting using a bottom-up normalisation strategy, the duplicates are already in normal form, and so we can easily share their representation as we do not need to perform rewrites on them. The Cadmium system [2] also uses a form of sharing, but rewrites are not performed on multiple occurrences of a shared term simultaneously.

Sharing causes some problems. Firstly, a rule may only apply to a term because of its context and different occurrences of a term may have a different context. Also, because we use a flattened representation for AC terms, the *result* of a rule application may be context dependent. We support sharing with simultaneous rewrites of all occurrences of a shared term. However, if a rewrite depends on a shared term's context, its representation is copied first, and the rewrite takes place on this non-shared copy. We allow AC terms to be in a non-flattened form temporarily, and flatten such terms on demand while matching.

Example 2. As an example of sharing, let there be given the following program:

$$\begin{aligned} \mathtt{f}(X) &\iff \mathtt{g}(X,X) & \qquad \mathtt{c} \setminus \mathtt{b} &\iff \mathtt{a} \\ \mathtt{g}(\mathtt{a} \wedge X, Y) &\iff \mathtt{h}(X,Y) & \qquad \mathtt{a} \setminus \mathtt{c} &\iff \mathtt{d} \end{aligned}$$

and goal term $\mathtt{f}(\mathtt{b} \wedge \mathtt{c})$, which is first rewritten into $\mathtt{g}(\mathtt{b} \wedge \mathtt{c}, \mathtt{b} \wedge \mathtt{c})$ using a shared representation for the $\wedge$ term. While matching this term with the rule for $\mathtt{g}/2$, we rewrite $\mathtt{b}$ into $\mathtt{a}$, which results in $\mathtt{g}(\mathtt{a} \wedge \mathtt{c}, \mathtt{a} \wedge \mathtt{c})$. Since $\mathtt{b}$ is only shared via its parent, we can perform this rewrite for all occurrences of $\mathtt{b}$ simultaneously. Next, we rewrite the goal term into $\mathtt{h}(\mathtt{c}, \mathtt{a} \wedge \mathtt{c})$ where the $\mathtt{c}$ terms are shared. Finally, we rewrite the second occurrence of $\mathtt{c}$ into $\mathtt{d}$. This rewrite depends on a shared term's context, so we copy its representation first. The result is $\mathtt{h}(\mathtt{c}, \mathtt{a} \wedge \mathtt{d})$. □

A final optimisation concerns inverse conjunctive context lookups. The inverse conjunctive context of a term T is computed in a demand-driven way and from the top down, i.e. a term is always considered before its subterms. We use indexing on the functor symbols appearing in a term to reduce the number of terms that are considered. We further optimise our approach by only indexing those terms that have already been considered while matching.

5 Conclusion

This work is strongly related to work on lazy evaluation in functional languages. It is known that lazy evaluation leads to better termination behaviour. It may also reduce the number of rule applications if subterms of a term being rewritten are discarded. In [1], it was shown that a bottom-up eager normalisation strategy for ACD term rewriting is incomplete because a term's conjunctive context might not be in normal form. In the approach we take here, it holds that if a conjunction term is in toplevel normal form, then so are all of its conjuncts. This means that often, terms in the conjunctive context are (at least) in toplevel normal form.

Acknowledgements Research funded by a PhD grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen). NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence Program.

References

1. Duck, G.J., Stuckey, P.J., Brand, S.: ACD term rewriting. In: ICLP 2006. LNCS, vol. 4079, Springer (2006) 117–131
2. Duck, G.J., De Koninck, L., Stuckey, P.J.: Cadmium: An implementation of ACD term rewriting. In: ICLP 2008. LNCS, vol. 5366, Springer (2008) 531–545
3. Stuckey, P.J., et al.: The G12 project: Mapping solver independent models to efficient solutions. In: ICLP 2005. LNCS, vol. 3668, Springer (2005) 9–13
4. Frühwirth, T.: Theory and practice of Constraint Handling Rules. Journal of Logic Programming **37**(1-3) (1998) 95–138
5. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge Univ. Press (1998)