Semi-Automatic Generation of CHR Solvers for Global Constraints

Frank Raiser

Faculty of Engineering and Computer Sciences, University of Ulm, Germany
Frank.Raiser@uni-ulm.de

Abstract. Constraint programming often involves global constraints, for which various custom filtering algorithms have been published. This work presents a semi-automatic generation of CHR solvers for the subset of global constraints defineable by specific automata. The generation is based on a constraint logic program modelling an automaton and an improved version of the Prim-Miner algorithm. The solvers only need to be generated once and achieve arc-consistency for over 40 global constraints.

1 Introduction

Global constraints are a combination of multiple constraints in order to improve filtering on the domains of involved variables. While it is common knowledge, that specialized filtering algorithms on global constraints achieve better domain pruning than generic approaches, the development and implementation of such algorithms requires a lot of effort. Thus, in [1] a generic method for deriving filtering algorithms from special checker automata is introduced.

Constraint Handling Rules (CHR) [2] is a multi-headed, guarded, and concurrent constraint programming language. CHR was designed for writing constraint solvers and is increasingly being used as a general-purpose programming language. However, there is no direct support for global constraints available in CHR so far.

The aim of this work is to adapt the results from [1] in order to generate CHR solvers for global constraints. To this end, we make use of the Prim-Miner algorithm proposed in [3] to generate rules from a constraint logic program (CLP). The necessary CLP is automatically created from the description of the automaton corresponding to a global constraint, as given by the global constraint catalog (GCC) [4, 5]. This work presents a refined version of the Prim-Miner algorithm that adapts it to the problem at hand, which results in a significant improvement of runtime complexity.

2 Preliminaries

2.1 Automata for Global Constraints

In [1] automata for checking if a global constraint holds are introduced. The underlying idea is to compile a list of *signature arguments* from the arguments

of the global constraint and use this list to iterate through the automaton. These automata can use counters which are initialized to a value in the start state and can be modified at each transition. Additionally, the final state specifies a final value which has to hold for the counter in order for the global constraint to hold. We use $\mathcal{M} = \{1, \ldots, |\mathcal{M}|\}$ for the set of transitions of an automaton.

Example 1. The `among` $(N, [X_1, \ldots, X_k], \overline{V})$ constraint holds if exactly N variables from the set of variables $X_1, \ldots, X_k$ take a value in the set $\overline{V}$. The corresponding automaton consists of two states. In the first state the value of the current variable X_i is checked for inclusion in the set $\overline{V}$ and a counter is incremented in that case. The second state is the final state where the final counter value is compared to N.

[1] further describes an arc-consistent filtering algorithm based on these automata and arc-consistent solvers for the ψ constraints as well as ϕ constraints, which are used to encode the transitions. In this work we generate CHR solvers for those constraints, which in combination with rules generating the necessary ψ and ϕ constraints allow arc-consistent filtering for global constraints. Note, however, that the arc-consistency result only holds for automata which do not involve counters. In all other cases the filtering algorithm and therefore the generated CHR solvers can still be used, but may not achieve arc-consistent filtering.

3 Semi-Automatic Solver Generation

To generate a CHR solver for a global constraint, first the automaton definition is extracted from the global constraint catalog [5]. Then CHR rules for creating signature arguments, ψ and ϕ constraints are written, after which the solvers for ψ constraints and for ϕ constraints are generated. Optionally, the generated ruleset can be optimized in a post-processing step. The following sections present these steps in more detail.

3.1 Generation of ψ and ϕ Constraints

For the filtering algorithm proposed in [1] the generation of ψ and ϕ constraints for the automaton is required. Note that this step cannot be fully automated, as the signature arguments depend on the specific global constraint for which to generate a solver. In some cases, however, the generation of these constraints can be done in a canonical way as detailed in [6].

3.2 Generation of Solver for ψ Constraint

The generation of a solver for the ψ constraints assumes that all transition constraints C_i and their negations are available as arc-consistent built-in constraints. We also require that for all subsets of these constraints their union is available as a arc-consistent built-in constraint. This directly leads to the creation of rules of the following kind $\forall i \in \mathcal{M} : \ \psi(S, \Delta) \Rightarrow \Delta \notin C_i \mid S \in \mathcal{M} \setminus \{i\}$.

Intuitively, these rules make use of each transition i corresponding to a transition constraint C_i and state the fact that if this constraint does not hold the transition cannot be made. Thus, the corresponding identifier for the transition is removed from the possible transitions.

In order to achieve arc-consistency, however, further rules are required. Considering a domain restriction for $D(S) = \{i_1, \ldots, i_k\}$ with $i_1, \ldots, i_{|\mathcal{M}|}$ being a permutation of $\mathcal{M}$ and $0 < k \leq |\mathcal{M}|$, a constraint $C_{i_1} \cup \ldots \cup C_{i_k}$ can be propagated: $\psi(S, \Delta) \wedge S \in \{i_1, \ldots, i_k\} \Rightarrow \Delta \in (C_{i_1} \cup \ldots \cup C_{i_k})$ Such rules are inserted for $D(S) = \{i_1, \ldots, i_k\}$ being any of the possible subsets of $\mathcal{M}$ with the exception of $\emptyset$.

Assuming an automaton with $|\mathcal{M}|$ transition edges where each edge is labeled with a different constraint, $O(|\mathcal{M}|)$ rules of the first kind and $O(2^{|\mathcal{M}|})$ rules of the second kind are added. Using these $O(2^{|\mathcal{M}|})$ rules we have the following result:

Theorem 1. *The generated solver achieves arc-consistency for the ψ constraint.*

Example 2. The generated solver for the ψ constraint of the `among` global constraint consists of these rules:

```
1  ψ(S, Δ, V) ⇒ Δ ∈ V | S ∈ {0, 1} \ {0}
2  ψ(S, Δ, V) ⇒ Δ ∉ V | S ∈ {0, 1} \ {1}
3
4  ψ(S, Δ, V) ∧ S ∈ {0} ⇒ Δ ∉ V
5  ψ(S, Δ, V) ∧ S ∈ {1} ⇒ Δ ∈ V
6  ψ(S, Δ, V) ∧ S ∈ {0, 1} ⇒ ⊤
```

3.3 Generation of Solver for ϕ Constraint

The solver for the ϕ constraint is based on the Prim-Miner algorithm. The basic idea is to encode the automaton in a constraint logic program P for the algorithm to work with and use all possible domains as candidate inputs.

Generation of CLP Creating the CLP P for the ϕ constraints is a straightforward encoding of the automaton's transitions into CLP rules [6]. The generation of these CLPs can be fully automated given the definition of the automaton.

A check performed by the Prim-Miner algorithm against such a CLP consists of a backtracking search. If the given domain restrictions are consistent with one of the automaton's transitions the check succeeds and the check fails if the given domain restrictions do not allow for any of the transitions to fire.

Solver Generation The generation of the solver for the ϕ constraint is performed by a modified version of the Prim-Miner algorithm, which we call the GC-Prim-Miner algorithm. It uses the previously generated CLP P against which to test goals. The resulting ruleset is a CHR solver for the ϕ constraint providing arc-consistency for global constraints whose automaton is free of counters:

Theorem 2. *For automata which do not involve counters the resulting rule set achieves arc-consistency for ϕ.*

As the runtime complexity of the direct application of the Prim-Miner algorithm is insufficient we can make use of the specifics of our application to improve it. By selecting the inputs in an advantageous way we can ensure, that the resulting ruleset still possesses the same propagation power, while at the same time drastically reducing the complexity of the algorithm. The details of this modification can be found in [6], along with the deduction of the runtime complexity now being $O(2^{3n+|\mathcal{M}|} + 2^{2n+2|\mathcal{M}|})$, whereas using the original Prim-Miner algorithm gives a complexity of $O(2^{2^n} * 2^{2^{|\mathcal{M}|}})$.

3.4 Post-Processing of Rule Set

After generating a set of CHR rules with the GC-Prim-Miner algorithm the resulting rule set can be reduced. A large number of the generated rules is redundant, therefore, an additional post-processing of the rule set leads to a more concise solver.

In order to find redundant rules for removal, the results about operational equivalence of CHR programs in [7] can be applied. [7] presents a decidable, sufficient, and necessary syntactic condition to determine operational equivalence of CHR programs that are terminating and confluent [8]. We can apply this condition by removing each rule from the generated rule set successively, and check if the complete rule set and the rule set without that rule are operationally equivalent. If they are, the selected rule is redundant and can be removed.

Example 3. Using the GC-Prim-Miner algorithm on the CLP for the ϕ constraint for the `among` automaton and removing all redundant rules generates the following solver:

1 $\phi(Q, \overline{K}, S, Q', \overline{K'}) \Rightarrow Q \in \{s\}$
2 $\phi(Q, \overline{K}, S, Q', \overline{K'}) \wedge Q' \in \{t\} \Rightarrow Q \in \{s\} \wedge S \in \{\$\}$
3 $\phi(Q, \overline{K}, S, Q', \overline{K'}) \wedge Q' \in \{s\} \Rightarrow Q \in \{s\} \wedge S \in \{0, 1\}$
4 $\phi(Q, \overline{K}, S, Q', \overline{K'}) \wedge S \in \{\$\} \Rightarrow Q \in \{s\} \wedge Q' \in \{t\}$
5 $\phi(Q, \overline{K}, S, Q', \overline{K'}) \wedge S \in \{0\} \Rightarrow Q \in \{s\} \wedge Q' \in \{s\}$
6 $\phi(Q, \overline{K}, S, Q', \overline{K'}) \wedge S \in \{1\} \Rightarrow Q \in \{s\} \wedge Q' \in \{s\}$
7 $\phi(Q, \overline{K}, S, Q', \overline{K'}) \wedge S \in \{0, 1\} \Rightarrow Q \in \{s\} \wedge Q' \in \{s\}$

4 Conclusion

In this paper we have shown a way to semi-automatically generate CHR solvers for the set of automata-describable global constraints. The process is not fully

automated due to the generation of signature arguments and because signature constraints are not available in a suitable format in the global constraint catalog.

We have shown that by the use of the GC-Prim-Miner algorithm, and given the availability of arc-consistent built-in constraint solvers for transition constraints, the generated CHR solvers achieve arc-consistency in those cases the automata-based filtering proposed in [1] allows for it. We have further shown, that the generality of the Prim-Miner algorithm can cause a runtime complexity problem, which can be alleviated by an order of magnitude if specialized for the problem at hand.

For future work the problems associated with the ψ constraint solver [6] need to be tackled. As there are few semantically different signature constraints used in the various automata it might be possible to develop arc-consistent solvers for these, including their negations and unions. Together with a way to automatically extract the signature constraints from the definitions given in the global constraint catalog this would allow for a fully automated generation of the CHR solvers.

References

1. Beldiceanu, N., Carlsson, M., Petit, T.: Deriving filtering algorithms from constraint checkers. In Wallace, M., ed.: Principles and Practice of Constraint Programming (CP'2004). Volume 3258 of Lecture Notes in Computer Science., Springer-Verlag (2004) 107–122
2. 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
3. Abdennadher, S., Rigotti, C.: Automatic generation of CHR constraint solvers. TPLP **5**(4-5) (2005) 403–418
4. Beldiceanu, N., Demassey, S.: Global constraint catalog (2008) http://www.emn.fr/x-info/sdemasse/gccat/.
5. Beldiceanu, N., Carlsson, M., Demassey, S., Petit, T.: Global constraint catalog: Past, present and future. Constraints **12**(1) (2007) 21–62
6. Raiser, F.: Semi-automatic generation of CHR solvers from global constraint automata. Technical Report UIB-2008-03, Ulmer Informatik Berichte, Universität Ulm (February 2008)
7. Abdennadher, S., Frühwirth, T.: Operational equivalence of CHR programs and constraints. In Jaffar, J., ed.: Principles and Practice of Constraint Programming (CP 1999). Volume 1713 of Lecture Notes in Computer Science., Springer-Verlag (1999) 43–57
8. Abdennadher, S.: Operational semantics and confluence of constraint propagation rules. In: Principles and Practice of Constraint Programming. (1997) 252–266