Constraint Handling Rules - The Story So Far

Thom Frühwirth
University of Ulm, Germany
www.informatik.uni-ulm.de/pm/mitarbeiter/fruehwirth/

Abstract

Rule-based programming experiences renaissance due to its applications in areas such as Business Rules, Semantic Web, Computational Biology, Verification and Security. Executable rules are used in declarative programming languages, in program transformation and analysis, and for reasoning in artificial intelligence applications.

Constraint Handling Rules (CHR) [6, 8, 11] is a concurrent committed-choice constraint logic programming language consisting of guarded rules that transform multi-sets of atomic formulas (constraints) into simpler ones until exhaustion. CHR was initially developed for solving constraints, but has matured into a general-purpose concurrent constraint language over the last decade, because it can embed many rule-based formalisms and describe algorithms in a declarative way. The clean semantics of CHR facilitates non-trivial program analysis and transformation.

Categories and Subject Descriptors D.1.6 [*Programming Techniques*]: Logic Programming; D.3.2 [*Programming Languages*]: Language Classifications—Constraint and logic languages, Constraint Handling Rules; D.3.3 [*Programming Languages*]: Language Constructs and Features—Constraints

General Terms Algorithms, Languages, Theory

Keywords Constraint Programming, Constraint Solving, Computational Logic, Concurrency, Executable Specification, Rule-Based Programming, Program Analysis, Applications

CHR in a Nutshell

CHR programs consist of two main kinds of rules: **Simplification rules** replace constraints by simpler constraints while preserving logical equivalence, e.g., $X \leq Y, Y \leq X \Leftrightarrow X = Y$. **Propagation rules** add new constraints that are logically redundant but may cause further simplification, e.g., $X \leq Y, Y \leq Z \Rightarrow X \leq Z$. Together with $X \leq X \Leftrightarrow \texttt{true}$, these three rules encode the axioms of a partial order relation. The rules compute its transitive closure and replace $\leq$ by equality $=$ whenever possible. For example, $A \leq B, B \leq C, C \leq A$ will be simplified into $A=B, A=C$.

Direct **ancestors of CHR** are logic programming, constraint logic programming [9] and concurrent committed-choice logic programming [10] languages. Like these languages, CHR has an operational semantics and a declarative semantics that are closely related. Other influences were the chemical abstract machine [4], term rewriting systems, and, of course, production rule systems.

CHR is appealing for **computational logic**, because logical theories are usually specified by implications and logical equivalences, corresponding to propagation and simplificiation rules. On the meta-level, given the transformation rules for deduction in a calculus, inference rules map to propagation rules and replacement rules to simplification rules.

The use of CHR as a **general purpose programming language** is justified by the following observation: Given a state transition system, its transition rules can readily be expressed with simplification rules. In this way, dynamics and changes (e.g., updates) can be modelled, possibly triggered by events and handled by actions (that are all represented by atomic constraints). In such applications, conjunctions of constraints are best regarded as interacting collections of concurrent agents or processes. The standard declarative semantics based on first order predicate logic is likely to break down, but linear logic does the job [5].

Rule-based programming languages have the stigma of inefficiency. The paper [12] introduces CHR machines, analogous to RAM and Turing machines. It shows that these machines can simulate each other in polynomial time, thus establishing that CHR is Turing-complete and, more importantly, that every algorithm can be implemented in CHR with **best known time and space complexity**, something that is not known to be possible in other pure declarative programming languages like Prolog. These results hold in practice, as optimal and elegant implementations of algorithms like union-find, shortest paths and Fibonacci heaps have shown.

CHR libraries exist for most Prolog systems, several for Java and Haskell. Standard constraint systems as well as novel ones such as temporal, spatial, or description logic constraints have been implemented, many programs are available online. Besides constraint solvers, **applications of CHR** can be found in computational logic[1], in agent programming, multi-set rewriting and production rule systems. The several hundred publications [11] mentioning CHR cover such diverse applications as type system design for Haskell, time tabling for universities, optimal sender placement, computational linguistics, spatio-temporal reasoning, chip card verification, semantic web information integration, and decision support for cancer diagnosis.

One advantage of a declarative programming language is the ease of **program analysis**. Techniques for termination and time complexity, as well as confluence and operational equivalence of CHR have been investigated.

Since CHR is Turing-complete, **termination** is undecidable. For simplification rules, techniques from term rewriting can be adapted, for propagation rules from deductive databases. Both kinds of rules in one program can make termination proofs hard. From a termination order, an upper bound for the time **complex-**

PPDP'06 July 10–12, 2006, Venice, Italy.
ACM 1-59593-388-3/06/0007.

[1] Integrating deduction and abduction, bottom-up and top-down execution, forward and backward chaining, tabulation and integrity constraints.

ity of simplification rules [7] can automatically be derived, but in general problem-specific methods that account for compiler optimizations are necessary.

Confluence of a program guarantees that any computation for a goal results in the same final state no matter which of the applicable rules are applied. Similar to term rewriting systems, there is a decidable, sufficient and necessary condition for confluence of terminating programs [1]. Any terminating and confluent program has a consistent logical reading and will automatically implement a **concurrent** any-time (approximation) and on-line (incremental) algorithm, where rules can be applied in parallel to different parts of the problem.

Surprisingly, there is also a decidable, sufficient and necessary syntactic condition for **operational equivalence** of terminating and confluent programs [2] (we do not know of any other programming language in practical use with this property).

Some Simple Small CHR Programs

For details, consult the CHR website [11].

Chemical Abstract Machine Programming Style

Compute minimum of a set of `min` candidates
```
min(I), min(J) ⇔ J>=I | min(I).
```
Compute primes, given `prime(2),...,prime(n)`
```
prime(I), prime(J) ⇔ J mod I=:=0 | prime(I).
```
Sort array by swapping positions
```
a(I,V), a(J,W) ⇔ I>J, V<W | a(I,W), a(J,V).
```

These three simplificaiton rules have guards (preconditions on the applicability of a rule): `J>=I`, `J mod I=:=0`, and `I>J,V<W`.

Minimum Constraint - The minimum of `X` and `Y` is `Z`

```
min(X,Y,Z) ⇔ X=<Y | Z=X.
min(X,Y,Z) ⇔ Y=<X | Z=Y.
min(X,Y,Z) ⇔ Z<X  | Y=Z.
min(X,Y,Z) ⇔ Z<Y  | X=Z.
min(X,Y,Z) ⇒ Z=<X, Z=<Y.
```

Also computes backwards, e.g. `min(A,2,2)` yields `2=<A` thanks to the last rule, a propagation rule (which adds the right hand side without removing the left hand side). Such rules can be automatically generated from specifications [3].

Fibonacci Variations - `M` is the `N`th Fibonacci number

Top-down Evaluation
```
fi(0,M) ⇔ M=1.                fi(1,M) ⇔ M=1.
fi(N,M) ⇔ N>=2 | fi(N-1,M1),fi(N-2,M2), M=M1+M2.
```
Matching is used on left hand sides of rules.

Top-down Evaluation with Tabling (in first rule)
```
fi(N,M1), fi(N,M2) ⇔ M1=M2, fi(N,M1).
fi(0,M) ⇒ M=1.                fi(1,M) ⇒ M=1.
fi(N,M) ⇒ N>=2 | fi(N-1,M1),fi(N-2,M2), M=M1+M2.
```
Turned simplification into propagation and merge duplicates.

Bottom-up Evaluation (finite version left as excercise)
```
fib ⇔ fi(0,1), fi(1,1).
fi(N1,M1), fi(N2,M2) ⇒ N2=N1+1 | fi(N2+1,M1+M2).
```
Basically, original simplification rules have been reversed.

Dynamic Program: Parsing with CYK Logical Algorithm

Grammar rules are in Chomsky normal form `A->T` or `A->B*C`.
Word is a +-separated sequence of terminal symbols.
```
terminal @ A->T, word(T+R) ⇒ parses(U,T+R,R).
non-term @ A->B*C, parses(B,I,J),parses(C,J,K) ⇒
                              parses(A,I,K).
substring@ word(T+R) ⇒ word(R).
```

Solving Linear Polynomial Equations and Inequations

Gaussian-like Concurrent Incremental Variable Elimination
Equations of the form `A1*X1+A2*X2+...An*Xn+B=0`.
Solved form: leftmost variable occurs only once.
Polynom for `X` computed from 1st equation replaces `X` in 2nd.
```
A1*X+P1=0, XP=0 ⇔
   find(A2*X,XP,P2) |
   compute(P2-(P1/A1)*A2,P3),
   A1*X+P1=0, P3=0.
B=0 ⇔ number(B) | zero(B).
```
`find`: removing `A2*X` from polynom `XP` is polynom `P1`.
`compute` normalizes expression into linear polynom.

Fourier's Algorithm for inequations is quite similar
```
A1*X+P1≥0, XP≥0 ⇒
   find(A2*X,XP,P2), opposite_sign(A1,A2) |
   compute(P2-(P1/A1)*A2,P3),
   P3≥0.
B≥0 ⇔ number(B) | non_negative(B).
```

References

[1] S. Abdennadher. Operational semantics and confluence of constraint propagation rules. In *3rd Intl. Conf. on Principles and Practice of Constraint Programming*, LNCS 1330. Springer, 1997.

[2] S. Abdennadher and T. Frühwirth. Operational equivalence of constraint handling rules. In *Fifth Intl. Conf. on Principles and Practice of Constraint Programming*, LNCS 1713. Springer, 1999.

[3] S. Abdennadher and C. Rigotti. Automatic generation of chr constraint solvers. *Theory Pract. Log. Program.*, 5(4-5):403–418, 2005.

[4] J.-P. Banatre, A. Coutant, and D. L. Metayer. A parallel machine for multiset transformation and its programming style. *Future Generation Computer Systems*, 4:133–144, 1988.

[5] H. Betz and T. Frühwirth. A linear-logic semantics for constraint handling rules. In *11th Intl. Conf. on Principles and Practice of Constraint Programming CP 2005*, LNCS 3709. Springer, 2005.

[6] T. Frühwirth. Theory and practice of constraint handling rules, Special issue on constraint logic programming. *Journal of Logic Programming*, 37(1-3):95–138, 1998.

[7] T. Frühwirth. As Time Goes By: Automatic Complexity Analysis of Simplification Rules. In *8th Intl. Conf. on Principles of Knowledge Representation and Reasoning*, Toulouse, France, 2002.

[8] T. Frühwirth and S. Abdennadher. *Essentials of Constraint Programming*. Springer, 2003.

[9] K. Marriott and P. J. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, Cambridge, Mass., 1998.

[10] V. Saraswat. *Concurrent Constraint Programming*. MIT Press, Cambridge, Mass., 1993.

[11] T. Schrijvers and T. Frühwirth. CHR Website, `www.cs.kuleuven.ac.be/~dtai/projects/CHR/`, 2006.

[12] J. Sneyers, T. Schrijvers, and B. Demoen. The Computational Power and Complexity of Constraint Handling Rules. In *Second Workshop on Constraint Handling Rules*, Sitges, Spain, October 2005.