Termination of CHR Constraint Solvers

Thom Frühwirth
Ludwig-Maximilians-Universität München
Oettingenstrasse 67
D-80538 Munich, Germany
fruehwir@informatik.uni-muenchen.de
www.informatik.uni-muenchen.de/~fruehwir/

4th October 1999

Remarks

We want to adapt and use existing approaches to termination in rule-based languages (logic programming and rewriting systems) to prove termination of actually implemented CHR constraint solvers.

Our approach proves termination of many CHR constraint solvers, ranging from Boolean and arithmetic to terminological and path-consistent constraints.

Why Termination?

- Good for users
- Confluence
- Completion
- Operational Equivalence

Prerequisite in all theoretical results about CHR.

Remarks

Related Work

Term Rewriting

- Dershowitz, JSC 1987.

Logic Programming

- Overview in de Schreye/Decorte, JLP 1994.
- Bezem, JLP 1993 and before.
- Apt/Pedreschi, ESPRIT CSL 90.

Logic Programming with Coroutining

- Naish, TR Melbourne 1992.
- Marchiori/Teusink, ILPS 95.

Constraint Logic Programming

- Colussi/Marchiori/Marchiori, PPCP 95.
- Mesnard, JICSLP 96.

Concurrent Logic Programming

- Plümer, FGCS 92.
- Rao/Kapur/Shyamasundar, NGC 1997.

Remarks

The main line of work in termination of logic programs originated from predecessor publications of the author of [Bez93]. Both programs and goals are characterized in terms of *level mappings*, a function from ground atoms to natural numbers. A logic program is *recurrent* if for every ground instance of each rule, the *level* of the head atom is higher than the level of each body atom. A goal is *bounded* if for every ground instance of each atom in the goal there is a maximum level which is not exceeded.

[ApPe90] and successive work refined this approach: Local variables and the specific left-to-right SLD resolution of Prolog are taken into account.

While [Mes96, MaTe95, Plu92] embark on level mappings, the theoretical work [CMM95] provides necessary and sufficient conditions for termination based on dataflow graphs, the practical work [Nai92] discusses informally how terminating procedures can be combined ensuring overall termination, and [KKS97] can use techniques from TRS directly since they translate GHC programs into TRS.

Concurrent Constraint Programming?

Termination problem even harder.

Example. Even Number Constraint:

```
even(X) <=> X=s(Y) | Y=s(Z), even(Z).
```

Queries:

- `even(N)` delays.
- `even(f(N))` delays.
- `even(s(N))` leads to `N=s(N1),even(N1)`.
- `even(N),even(s(N))` does not terminate.

Constraint Handling Rules?

Not only concurrency and constraints, but also propagation rules and multiple heads.

Remarks

Basic Idea

Termination for rule-based languages

Prove that head of the rule is larger than body of the rule in some *well-founded order*.

Polynomial interpretation:
Maps terms to natural numbers.

E.g. `even(X) <=> X=s(Y) | Y=s(Z), even(Z).`

$rank(even(s^n(X))) = n$

Not enough, must take variables into account.

Modes and types.

E.g. `even` cannot be moded.
`even` terminates if ill-typed.

Boundedness: Queries must be sufficiently known.

E.g. `even` with ground (variable-free) argument.

Remarks

The basic idea is to prove that in each rule, the head atom is strictly larger than every atom occurring in the body of the rule. In order to be applicable, programs and queries usually have to be well-moded or queries sufficiently known.

Contents

We will first give syntax and semantics for CHR. In the next section, we introduce useful termination orders for CHR. Then we prove termination of actually implemented CHR constraint solvers ranging from Boolean and arithmetic to terminological and path-consistent constraints.

Syntax

Upper case letters stand for conjunctions of CHR (user-defined) or built-in (predefined) constraints.

A *simplification CHR* is of the form

```
[Name '@'] Head '<=>' [Guard '|'] Body.
```

A *propagation CHR* is of the form

```
[Name '@'] Head '==>' [Guard '|'] Body.
```

`Head` is a conjunction of CHR constraints.
`Guard` is a conjunction of built-in constraints.
`Body` is a conjunction of built-in and CHR constraints.

Remarks

Operational Semantics

A *state (goal)*: conjunction of constraints.

Upper case letters: conjunctions of constraints.

CT: constraint theory for the built-in constraints.

Simplify

$H' \wedge D \longmapsto (H = H') \wedge G \wedge B \wedge D$

if (H `<=>` $G \mid B$) variant of rule in P

$CT \models D_{bi} \rightarrow \exists \bar{x}(H = H' \wedge G)$

Propagate

$H' \wedge D \longmapsto (H = H') \wedge G \wedge B \wedge H' \wedge D$

if (H `==>` $G \mid B$) variant of rule in P

$CT \models D_{bi} \rightarrow \exists \bar{x}(H = H' \wedge G)$

Remarks

An *initial state (query)*. A *final state (answer)*: No fair computation step is possible anymore or the state is inconsistent.

$\bar{x}$ denotes the fresh variables occurring in the variant of the rule choosen from P.

A rule is *applicable* to CHR constraints H' whenever these constraints match the head atoms H of the rule and the guard G is entailed (implied) by the built-in constraint store.

If a simplification rule (`(`H `<=>` G `|` B`)`) appearing in the given CHR program P is applicable to the CHR constraint H', the **Simplify** transition removes H' from the state, adds B and also adds the equation $H = H'$ and the guard G to the state. If a propagation rule (`(`H `==>` G `|` B`)`) is applicable to H', the **Propagate** transition adds B and also adds the equation $H = H'$ and the guard G.

We require that the rules are applied fairly, i.e. that every rule that is applicable is applied eventually. Fairness is respected and trivial non-termination is avoided by applying a propagation rule at most once to the same constraints.

Termination of CHR Constraint Solvers

Termination in all contexts under any scheduling which prefers built-in constraints.

A CHR program P is terminating for a class of goals G, *if there are no infinite derivations starting from any goal in G using rules from P.*

Remarks

CHR Termination Orders

Linear polynomial interpretation: *Rank* of a term or atom is defined by a linear positive combination of the rankings of its arguments.

$$rank(f(t_1, \ldots, t_n)) =$$
$$a_0^f + a_1^f * rank(t_1) + \ldots + a_n^f * rank(t_n)$$

where the a_i^f are natural numbers.
For each variable X, $rank(X) \geq 0$.

Rank orders are stable under substitution and well-founded for ground terms or atoms.

Example. Size of a term.

$size(f(t_1, \ldots, t_n)) = 1 + size(t_1) + \ldots + size(t_n)$

$size(f(a, g(b, c)) = 5$.
$size(f(a, X)) = 2 + size(X)$ with $size(X) \geq 0$.
$size(f(g(X), X)) \geq size(f(a, X))$ since
$2 + 2 * size(X) \geq 2 + size(X)$.

Remarks

CHR Termination Orders 2

- Rank of built-in constraints is zero.
- *Rank constraints*: Built-in constraints may imply order constraints on arguments, e.g. $s = t \rightarrow rank(s) = rank(t)$.
- *Ranking formula of a rule* `H <=> G | B`: $RC(G, B) \rightarrow rank(H) > rank(B)$, where $RC(G, B)$ is the conjunction of the rank constraints derived from the built-in constraints in the guard and body of the rule.
- *Boundedness of a goal G*: The rank of any instance of G is bounded from above by some constant k.

Remarks

Obviously, the rank of a ground (variable-free) term is always bounded.

CHR Termination Theorem

Given a CHR termination order where

$$rank((A \wedge B)) = rank(A) + rank(B).$$

and a CHR program P without propagation rules. If for each rule in P the ranking formula holds, then P is terminating for all bounded goals.

Proof Sketch. Applying a rule (H `<=>` $G \mid B$) to a state $H' \wedge D$ leads to $(H = H') \wedge G \wedge B \wedge D$.

1. Show that
$rank(H' \wedge D) > rank(((H = H') \wedge G \wedge B \wedge D))$.

We know $(H = H') \rightarrow rank(H) = rank(H')$, $rank(G) = 0$, $rank(H = H') = 0$ and $RC(G, B) \rightarrow rank(H) > rank(B)$.

$rank(H' \wedge D) = rank(H') + rank(D) = rank(H) + rank(D)$.

$rank(((H = H') \wedge G \wedge B \wedge D)) = rank(B) + rank(D)$.

2. Show that ranks of states are always bounded.

Remarks

To show that the ranks of all states are bounded, note the following: Any ranking is well-founded and has the stability property. Since goals are bounded, the rank of a state is bounded. Due to the ranking condition, the boundedness of the source state is propagated to target state. Thus no infinite computations are possible, hence P is terminating.

The Theorem also holds for CHR termination orders extended to multi-sets where

$$rank((A \wedge B)) = rank(A) \cup rank(B).$$

Boolean Algebra, Propositional Logic

The Boolean cardinality constraint `#(L,U,BL,N)` holds if between `L` and `U` Boolean variables in the list `BL` are equal to `1`. `N` is the length of `BL`.
Boolean cardinality can express
negation `#(0,0,[C],1)`,
exclusive or `#(1,1,[C1,C2],2)`,
conjunction `#(N,N,[C1,...,Cn],N)`
and disjunction `#(1,N,[C1,...,Cn],N)`.

```
triv_sat@ #(L,U,BL,N) <=> L=<0,N=<U | true.
pos_sat @ #(L,U,BL,N) <=> L=N | all(1,BL).
neg_sat @ #(L,U,BL,N) <=> U=0 | all(0,BL).
pos_red @ #(L,U,BL,N) <=> delete(1,BL,BL1) |
          0<U, #(L-1,U-1,BL1,N-1).
neg_red @ #(L,U,BL,N) <=> delete(0,BL,BL1) |
          L<N, #(L,U,BL1,N-1).
```

Remarks

In the code, all constraints except cardinality `#` are built-in.

`all(T,L)` binds all elements of the list `L` to `T`.

`delete(X,L,L1)` deletes the element `X` from the list `L` resulting in the list `L1`.

When `delete/3` is used in the guard, it will only succeed if the element to be removed actually occurs in the list. E.g. `delete(1,BL,BL1)` will delay if it tries to bind a variable in `BL` to `1`. It will only succeed if there actually is a `1` in the list. It will fail, if all elements of the list are zeros.

Termination

CHR termination order:

$rank(\#(L,U,BL,N)) = length(BL)$

$length([]) = 0$
$length([X|L]) = 1 + length(L)$

$delete(X,L,L1) \rightarrow length(L) = length(L1) + 1$

Termination proof:
From

```
pos_red @ #(L,U,BL,N) <=> delete(1,BL,BL1) |
            0<U, #(L-1,U-1,BL1,N-1).
```

we get to prove

$length(BL) = length(BL1) + 1 \;\rightarrow$
$length(BL) > length(BL1).$

Boundedness: finite closed list `BL`.

Remarks

Since the cardinality constraint is either simplified into a built-in constraint (satisfaction rules) or reduced to a cardinality with a shorter list (reduction rules), this implementation terminates.

Due to the ranking, a goal consisting of built-in and cardinality constraints is bounded if the lengths of the lists in the cardinality constraints are known, i.e. if the lists are closed. If a list was open(-ended), there could be producers of an infinite list, and then the associated cardinality constraint would not necessarily terminate.

Linear Polynomial Equations

Linear polynomial equation:
$a_1 * x_1 + \ldots + a_n * x_n + b = 0$.
Variables appear in strictly descending order.

Variable Elimination

```
empty @ B eq 0 <=> number(B) | B=0.

eliminate @
A1*X+P1 eq 0, A2*X+P2 eq 0 <=>
   compute(P2+P1*A2/A1,P3),
   A1*X+P1 eq 0, P3 eq 0.
```

Remarks

A *linear polynomial equation* is of the form $p + b = 0$ where b is a constant and the polynomial p is the sum of monomials of the form $a_i * x_i$ with coefficient $a_i \neq 0$ and x_i is a variable. Constants and coefficients are numbers.

The `empty` rule says that if the polynomial contains no more variables, the constant `B` must be (approximate to) zero.

The `eliminate` rule performs variable elimination. It takes two equations that start with the same variable. The first equation is left unchanged, it is used to eliminate the occurrence of the common variable in the second equation. The auxiliary built-in constraint `compute` simplifies a polynomial arithmetic expression into a new polynomial.

Note that no variable is made explicit, i.e. no pivoting is performed. Any two equations with the same first variable can react with each other.

The solver can be extended by a few rules to create explicit variable bindings, to make implicit equalities between variables explicit, to deal with inequations using slack variables or fouriers algorithm.

Termination

CHR termination order:

Extend termination order to multi-sets.

$rank((A \wedge B)) = rank(A) \cup rank(B)$
$rank(A) = vars(A)$ if A is a CHR constraint
$rank(A) = \{\}$ if A is a built-in constraint
$a_1 * X_1 \ldots a_n * X_n + b = 0 \rightarrow X_i \succ X_j$ if $i > j$
$rank(compute(E, P) \rightarrow vars(E) \supseteq vars(P)$

Termination proof:

```
A1*X+P1 eq 0, A2*X+P2 eq 0 <=>
   compute(P2+P1*A2/A1,P3),
   A1*X+P1 eq 0, P3 eq 0.
```

Prove $(P_2 \cup P_1 \supseteq P3) \rightarrow$
$(\{X\} \cup P_1 \cup \{X\} \cup P2) > (\{X\} \cup P_1 \cup P3)$.

Boundedness: Order is on sets of variables.

Remarks

For better readability, we write just P instead of $vars(P)$.

The rank constraint (1) says that the monomials in the equations are ordered by their variables. The rank constraint (2) says that the built-in constraint `compute` does not introduce new variables, but may eliminate occurences of some.

Hence the body rank multiset contains only variables from the head rank multiset. Due to (1) we know that the variable X does not occur in P_1, P_2 and P_3, and that it comes before all other variables in P_1, P_2 and P_3 in the variable order.

Therefore the head rank multiset is strictly greater in the multiset order than the body rank multiset, because in the former X occurs twice and in the latter X occurs only once.

Path Consistency

A *disjunctive binary constraint* `c(I,K,{ r1,..., rn})` denotes a finite disjunction $(X\ r_1\ Y) \vee \ldots \vee (X\ r_n\ Y)$.

Path consistency algorithm

```
c(I,K,C1), c(K,J,C2), c(I,J,C3) <=>
      composition(C1,C2,C12),
      intersection(C12,C3,C123),
      C123=\=C3
      |
      c(I,K,C1), c(K,J,C2), c(I,J,C123).
```

Remarks

A *binary constraint network* consists of a set of variables and a set of (disjunctive) binary constraints between them. The network can be represented by a *directed constraint graph*, where the nodes denote variables and the arcs are labeled by binary constraints. Logically, a network is a conjunction of binary constraints.

A *disjunctive binary constraint* c_{xy} between two variables X and Y, also written $X\ \{r_1, \ldots, r_n\}\ Y$, is a finite disjunction $(X\ r_1\ Y) \vee \ldots \vee (X\ r_n\ Y)$, where each r_i is a relation that is applicable to X and Y. The r_i are called *primitive constraints.* They are assumed to be pairwise disjoint.

A network is *path consistent* if for pairs of nodes (i, j) and all paths $i - i_1 - i_2 \ldots i_n - j$ between them, the direct constraint c_{ij} is at least as tight than the indirect constraint along the path, i.e. the composition of constraints $c_{ii_1} \otimes \ldots \otimes c_{i_n j}$ along the path.

Composition of disjunctive constraints can be computed by pairwise composition of its primitive constraints. Intersection for disjunctive constraints can be implemented by set intersection.

Termination

CHR termination order:

$rank(c(I,K,C)) = cardinality(C)$

$rank(A) = 0$ otherwise.

$intersection(C1,C2,C3) \rightarrow$

$rank(C3) \leq rank(C1) \wedge rank(C3) \leq rank(C2)$

$intersection(C1,C2,C3) \wedge C3 \neq C2 \rightarrow$

$rank(C3) \neq rank(C2)$

Termination proof:

In the guard, `C123=\=C3` is checked to make sure the new constraint `C123` is different from the old one `C3`. Hence the cardinality of `C123` must be strictly less than that of `C3`.

Boundedness: `Ci` must be known finite set.

Remarks

To prove termination we rely on the cardinality of the sets representing the disjunctive constraints and the properties of set intersection.

Hence the body is ranked strictly smaller than the head of the rule. Goals are bounded, since C is always a known, finite set of primitive constraints.

Any solver derived from this generic path consistency solver will terminate, too.

Interval Domain Constraints

```
inconsistent @ X in A:B <=> A>B | false.
intersection @ X in A:B, X in C:D <=> A=<B |
               X in max(A,C):min(B,D).

le @ X le Y, X in A:B, Y in C:D <=>
     A=<B,B>D |
     X le Y, X in A:D, Y in C:D.
eq @ X eq Y, X in A:B, Y in C:D <=>
     A=<B,C=<D,A=\=C |
     X eq Y, X in max(A,C):B, Y in max(C,A):D.

add @ add(X,Y,Z), X in A:B, Y in C:D, Z in E:F
      <=>
      A=<B,C=<D,
 \+(A>=E-D,B=<F-C,C>=E-B,D=<F-A,E>=A+C,F=<B+D)
      |
      add(X,Y,Z),
      X in max(A,E-D):min(B,F-C),
      Y in max(C,E-B):min(D,F-A),
      Z in max(E,A+C):min(F,B+D).
```

Remarks

Arc consistency can be seen as special case of path consistency, where all but one constraint is unary instead of binary.

The interval constraint `X in A:B` means that `X` is a number between the bounds `A` and `B`.

Termination

CHR termination order:

$rank(X\ in\ A:B) = B - A + 1$ if $B \geq A$
$rank(C) = 0$ otherwise

Termination proof:
In each rule, at least one interval in the head is strictly larger than the corresponding interval in the body, while the other intervals remain unchanged or will be removed. Proved using the inequalities in the guards.

```
eq @ X eq Y, X in A:B, Y in C:D <=>
     A=<B,C=<D,A=\=C |
     X eq Y, X in max(A,C):B, Y in max(C,A):D.
```

Boundedness: The interval bounds are known.

Remarks

The constraints `A=<B` and `C=<D` in the guard of a rule ensure that the rank of the head of the rule cannot be 0. (In implementations that apply rules in textual order, these guard constraints can be dropped.) The ranking condition for the first rule `inconsistent` also holds, even though its head rank is 0, since its order constraint is inconsistent:

$$(A > B \land \mathit{false}) \rightarrow 0 > 0$$

Even though the ranking is only well-defined for interval bounds that are integers, there is a simple way to allow for floating point numbers and rational numbers as well: First note that each kind of numbers is closed under the interval computations, since they use only the arithmetic operations `max, min` and `+, -`. Note that floating point numbers can be represented by rational numbers. Finally, any problem on rational numbers can be transformed into an equivalent one on integers by multiplying all numbers in the problem with their greatest common divisor.

Conclusions

- First work on proving termination in concurrent constraint logic languages.
- Proved termination for many CHR constraint solvers.
- Linear polynomial interpretations sufficed, contrary to what was feared in the literature.
- Recursion modifies one sufficiently known argument position.
- Limited use of propagation rules.

Open Problem for future work

Propagation rules in path and arc consistency algorithms on incomplete constraint networks:

```
c(I,K,C1), c(K,J,C2) ==>
         composition(C1,C2,C3), c(I,J,C3).
c(I,J,C1), c(I,J,C2) <=>
         intersection(C1,C2,C3), c(I,J,C3).
```

Fairness has to be considered for termination.

Remarks

These solvers have recursion on the same constraint through both simplification and propagation rules. This means that a constraint can be first added and then be removed during the computation.

Future works aims at giving termination proofs also for this kind of solvers. One will have to take into account fairness (which implies that simplification is applied sufficiently often before propagation) and the fact that propagation rules are never applied a second time to the same constraints.

Another interesting line of future work is to strengthen the antecedent of a ranking condition by introducing type constraints (ill-typed goals either delay or fail).