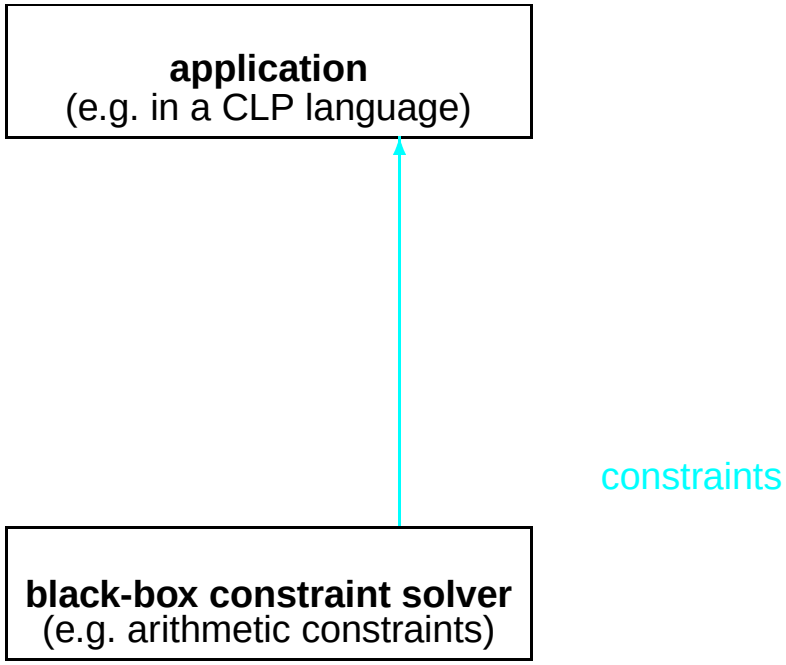
On Completion of Constraint Handling Rules

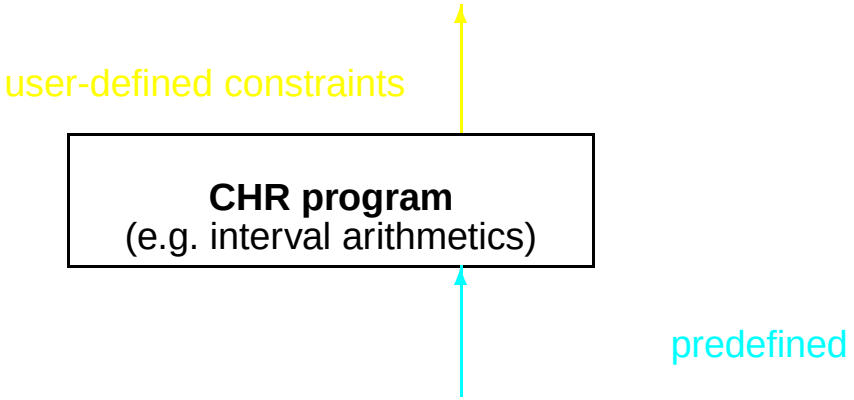
Slim Abdennadher and Thom Frühwirth

Computer Science Department
University of Munich
Oettingenstr. 67, 80538 Munich, Germany

Architecture of Constraint Programs



with CHR


user-defined constraints
CHR program
(e.g. interval arithmetics)
predefined

Remarks

- In general constraint application are written in a CLP language using black box solvers. Nevertheless these solvers are efficient, this approach makes it hard to modify a solver or build a solver over a new domain, let alone reason about and analyze it.
- CHR is a proposal to extend CLP languages to allow the user to define new constraints, together with rules specifying how the new constraints react with the constraint store.
- CHR: declarative
- Black-box solver: non-declarative

CHR: Introductory Example

- defined by CHR: $\leq$
- defined by the blackbox solver: *true*, $=$

$$
\begin{array}{rcll}
X \leq X & \Leftrightarrow & \mathit{true} & \text{(reflexivity)} \\
[X \leq Y & \Leftrightarrow & X = Y \mid \mathit{true} & \text{(reflexivity)}] \\
X \leq Y \wedge Y \leq X & \Leftrightarrow & X = Y & \text{(antisymmetry)} \\
X \leq Y \wedge Y \leq Z & \Rightarrow & X \leq Z & \text{(transitivity)}
\end{array}
$$

$$
\begin{array}{l}
A \leq B \wedge B \leq C \wedge C \leq A \\
\quad \Big| \text{ (transitivity)} \\
A \leq B \wedge B \leq C \wedge C \leq A \wedge A \leq C \\
\quad \Big| \text{ (antisymmetry)} \\
A \leq B \wedge B \leq C \wedge A = C \\
\quad \Big| \text{ (black-box solver)} \\
A \leq B \wedge B \leq A \wedge A = C \\
\quad \Big| \text{ (antisymmetry)} \\
A = B \wedge A = C
\end{array}
$$

Remarks

- CHR consists of guarded rules. We distinguish two kinds of rules. Simplification rule replaces constraints by simpler constraints while preserving logical equivalence. Propagation rule adds new constraints, which are logically redundant but may cause further simplification.

- We define a user-defined constraint for less-than-or-equal. The syntactical equality and `true` are predefined constraints. The CHR program implements reflexivity, antisymmetry, and transitivity in a straightforward way.

- The reflexivity rule states that $X \leq X$ is logically true. Whenever we see a constraint of the form $A \leq A$ we can simplify it with true. The reflexivity rule can be written using a guard (a precondition on the aplicability of the rule).

- The antisymmetry rule means that if we find $X \leq Y$ as well as $Y \leq X$ in the current constraint, we can replace them by the logically equivalent $X=Y$.

- The transitivity rule propagates constraints. It states that the conjunction of $X \leq Y$ and $Y \leq Z$ implies $X \leq Z$. Operationally, we add the logical consequence $X \leq Z$ as a redundant constraint.

- Redundancy from propagation rules is useful, as the following computation shows:

CHR: Syntax and Declarative Semantics

Simplification rule: $H \Leftrightarrow C \mid B \qquad\qquad \forall \bar{x}\ (C \rightarrow (H \leftrightarrow \exists \bar{y}\ B))$

Propagation rule: $H \Rightarrow C \mid B \qquad\qquad \forall \bar{x}\ (C \rightarrow (H \rightarrow \exists \bar{y}\ B))$

($\bar{x}$: variables occurring in H or C; $\bar{y}$: variables occurring only in B;)

Declarative semantics of a CHR program:

- the above logical formulas +
- a constraint theory CT for the predefined constraints.

CHR: Calculus

Solve

If $CT \models \forall^* (G \leftrightarrow G')$

and G' is "simpler" than G

then $\dfrac{G}{G'}$

Simplify

If $(H \Leftrightarrow C \mid B)$ is a fresh variant of a rule with variables $\bar{x}$

and G_{pre} are the predefined constraints in G

and $CT \models G_{pre} \rightarrow \exists \bar{x} (H{=}H' \wedge C)$

then $\dfrac{H' \wedge G}{H{=}H' \wedge B \wedge G}$

Propagate

If $(H \Rightarrow C \mid B)$ is a fresh variant of a rule with variables $\bar{x}$

and G_{pre} are the predefined constraints in G

and $CT \models G_{pre} \rightarrow \exists \bar{x} (H{=}H' \wedge C)$

then $\dfrac{H' \wedge G}{H{=}H' \wedge B \wedge H' \wedge G}$

- "$\wedge$" is AC (or AC1 with unit $\top$)
- "$H = H'$": syntactic equality per component of the conjunctions H and H';
- simplified states and rules;
- Actual states contain information to avoid trivial nontermination of propagation rules.

Confluence

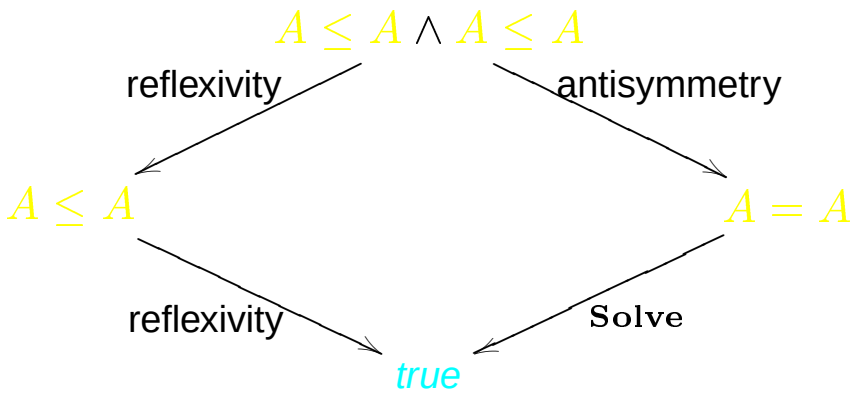
Given a goal, every computation leads to the same result no matter what rules are applied.

A decidable, sufficient and necessary condition for confluence of terminating CHR programs through joinability of critical pairs (Abdennadher, CP97).

Example

$$X \leq X \quad \Leftrightarrow \quad \mathit{true} \qquad \text{(reflexivity)}$$

$$X \leq Y \wedge Y \leq X \quad \Leftrightarrow \quad X = Y \qquad \text{(antisymmetry)}$$



Remarks

- In previous work we introduced a notion of confluence for CHR programs. Confluence is an essential syntactical property of any constraint solver. It ensures that the solver will always compute the same result for a given set of constraints independent of which rules are applied.

- We gave a decidable, sufficient and necessary syntactic condition for confluence of terminating CHR programs. This condition adopts the notion of critical pairs as known from term rewriting systems.

- These critical pairs can be derived from rules with overlapping heads.

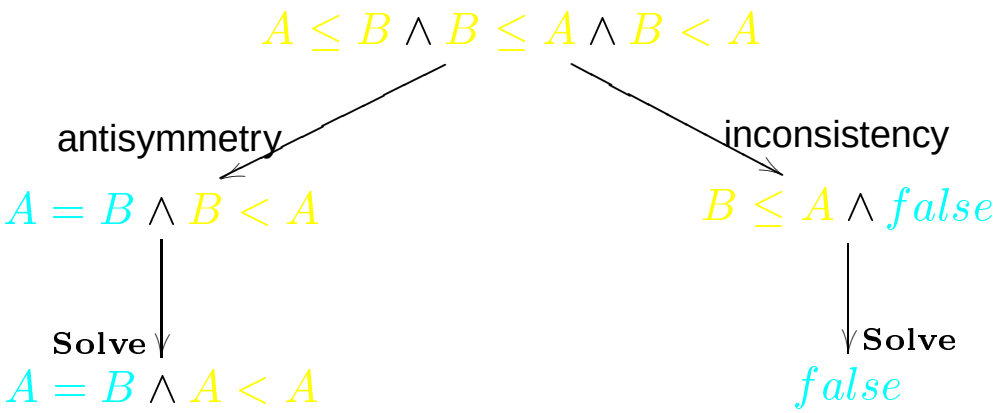
- Consider the program for less-than-or-equal. The critical state stems from unifying the first atom of the head of the reflexivity rule with the first atom of the head of the antisymmetry rule. The critical pair stems from applying the reflexivity and the antisymmetry rule.

Completion

Derive rules from a non-joinable critical pair that would allow a transition from one of the critical states into the other one.

$X \leq Y \wedge Y \leq X \quad \Leftrightarrow \quad X = Y$ (antisymmetry)

$X \leq Y \wedge Y < X \quad \Leftrightarrow \quad false$ (inconsistency)



$X < X \quad \Leftrightarrow \quad X = Y \quad | \quad false$ (irreflexivity)

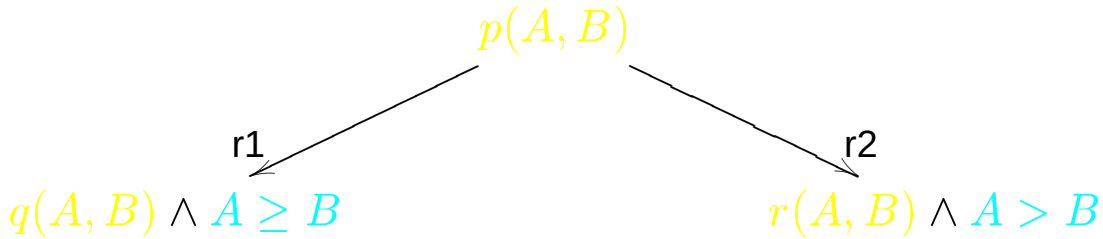
Remarks

- The idea of completion is to derive a rule from a non-joinable critical pair that would allow a transition from one of the critical states into the other one, thus re-introducing confluence.
- This example shows that the completion method can be used - to some extent – to specialize constraints.
- We extend the CHR program for less-than-or-equal by a simplification rule expressing the interaction between less-than-or-equal and less. Then the resulting program loses confluence.
- The completion procedure inserts the following rule ... expressing the irreflexivity of $<$.

Orientation of the Rules

$$p(X,Y) \quad \Leftrightarrow \quad X \geq Y \wedge q(X,Y) \qquad \text{(r1)}$$

$$p(X,Y) \quad \Leftrightarrow \quad X > Y \wedge r(X,Y) \qquad \text{(r2)}$$



$$r(X,Y) \quad \Leftrightarrow \quad X > Y \quad \mid \quad q(X,Y) \wedge X \geq Y \qquad \text{(r3)}$$

$$q(X,Y) \quad \Rightarrow \quad X \geq Y \quad \mid \quad X > Y \qquad \text{(r4)}$$

1¡

Remarks

- In contrast to completion methods for TRS, we need more than one rule to make a critical pair joinable and in general it is not sufficient to insert only simplification rules as in completion for TRS, in order to join a non-joinable critical pair. as the following example shows.

- We write the following CHR program, where `p, q` and `r` are user-defined constraints and $\geq$, $>$ are built-in constraints. The CHR program is not confluent, since the c.p. stemming from `r1` and `r2` is non-joinable.

Inference Rules of Completion

- C: set of critical pairs
- P: set of rules

CP-Deduction:

$$\frac{(C, P) \qquad (S_1, S_2) \text{ is a critical pair of } P}{(C \cup \{(S_1, S_2)\}, P)}$$

CP-Simplification:

$$\frac{(C \cup \{(S_1, S_2)\}, P) \qquad S_1 \mapsto S_1' \qquad S_2 \mapsto S_2'}{(C \cup \{(S_1', S_2)\}, P)}$$

CP-Deletion:

$$\frac{(C \cup \{(S_1, S_2)\}, P) \qquad S_1 \text{ and } S_2 \text{ are joinable}}{(C, P)}$$

CP-Orientation:

$$\frac{(C \cup \{(S_1, S_2)\}, P) \qquad R = orient_{\gg}(S_1, S_2)}{(C, P \cup R)}$$

Remarks

- Our completion algorithm maintains a set C of critical pairs and a set P of rules.
- The rule *CP-Deduction* permits to add critical pairs to C.
- The rule *CP-Simplification* replaces state in a critical pair by its successor state.
- The rule *CP-Deletion* removes a joinable critical pair.
- *CP-Orientation* removes a critical pair from C and adds new rules to P, provided the critical pair can be oriented with respect to the termination ordering $\gg$.

Properties of Completion (I)

1. The algorithm stops successfully
2. The algorithm aborts unsuccessfully
3. The algorithm does not terminate

Range-restricted Rule:
Every variable in the body or the guard appears also in the head.

Correctness Theorem

If P is a range-restricted terminating CHR program and the completion procedure is successful

Then: Output P'

- confluent
- terminating
- logically equivalent to P

Remarks

- As is the case for TRS our completion procedure cannot be always successful. We distinguish three cases:

  1. The algorithm stops successfully and returns a program P'.
  2. The algorithm aborts unsuccessfully, if a critical pair cannot be transformed into rules for one of three reasons:
     - The program remains terminating if new rules are added but the termination ordering is too weak to detect this.
     - The program loses termination if new rules are added.
     - The critical pair consists exclusively of built-in constraints.
  3. The algorithm does not terminate, because new rules produce new critical pairs, which require again new rules, and so on.

- we showed that when the algorithm stops successfully, the returned program is confluent and terminating.

Properties of Completion (II)

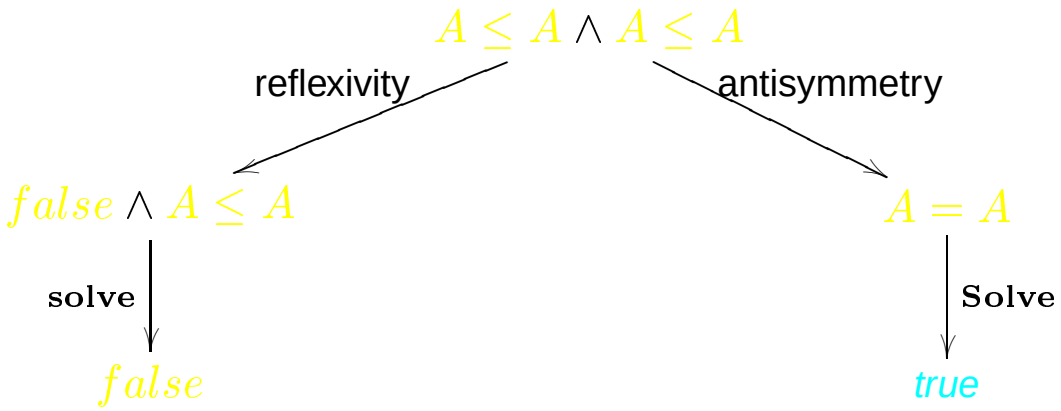
Consistency Theorem

If the completion procedure aborts unsuccessfully because the final states consist only of differing predefined constraints

Then The logical meaning of P is inconsistent.

$$X \leq X \quad \Leftrightarrow \quad \mathit{false} \qquad \text{(reflexivity)}$$

$$X \leq Y \wedge Y \leq X \quad \Leftrightarrow \quad X = Y \qquad \text{(antisymmetry)}$$



Remarks

- Another property of the completion procedure is that it can exhibit inconsistency of the program to complete.
- The logical meaning of this program is not a consistent theory.

Conclusion

- Completion method for Constraint Handling Rules to make a non-confluent CHR program confluent by adding new rules.
- Completion helps the CHR programmer to extend, modify and specialize existing solvers instead of having to write them from scratch.

Current Work: The relationship of completion to partial evaluation.

Remarks

- Partial evaluation is a particular program transformation for specializing programs. One interesting direction for future work is to investigate the relationship of completion to partial evaluation.