Analysis and Automatic Generation of CHR Programs

驰

Prof. Dr. Slim Abdennadher
German University in Cairo

CHR Summer School

5.9.2011 - 9.9.2011

Motivation of the Course

Given the following program:

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

Questions to be addressed

- Does the program do what the programmer intended?
- Do we get always the same results independent of the order of rules?
- If not, can the program be fixed automatically?
- Does the program always terminate?
- Is the program efficient enough?
- Is there another program that performs the same task?
- Could CHR programs be generated automatically?

Overview of the course

- **Part I: Analysis of CHR Programs**
  - Confluence
  - Completion
  - Operational Equivalence
- Part II: Automatic Generation of CHR Programs
  - from an extensional definition
  - from an intentional definition

CHR: Syntax and Declarative Semantics

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

- H: non-empty conjunction of user-defined constraints (Head)
- G: conjunction of built-in constraints (Guard)
- B: conjunction of user-defined and built-in constraints (Body)

Declarative semantics of a CHR program:

- the above logical formulas
- a constraint theory CT for the built-in constraints.

CHR: Operational Semantics

Simplify

If	$(H \Leftrightarrow G \mid B)$ is a fresh variant of a rule with variables $\bar{x}$
and	$CT \models C_{built} \rightarrow \exists \bar{x}(H{=}H' \wedge G)$
then	$H' \wedge C \quad \mapsto \quad C \wedge H{=}H' \wedge B$

C_{built} represents the built-in constraints in C.

Propagate

If	$(H \Rightarrow G \mid B)$ is a fresh variant of a rule with variables $\bar{x}$
and	$CT \models C_{built} \rightarrow \exists \bar{x}(H{=}H' \wedge G)$
then	$H' \wedge C \quad \mapsto \quad H' \wedge C \wedge H{=}H' \wedge B$

CHR Program Analysis

Termination
Every computation starting from any goal ends.

Consistency
The logical meaning of the rules is consistent.

Confluence
The answer of a query is always the same, no matter which of the applicable rules are applied.

Completion
Make non-confluent programs confluent by adding new rules.

Operational Equivalence
Do two programs have the same behavior?

Complexity
Determine time complexity from structure of rules.

Important Property: Monotonicity and Incrementality

$$
\begin{array}{llll}
\text{If} & G & \longmapsto & G' \\
\text{then} & G \wedge C & \longmapsto & G' \wedge C
\end{array}
$$

Online Algorithm

The complete input is initially unknown.
The input data arrives during computation.
No recomputation from scratch necessary.

$$[1]\ \underline{A{\leq}B} \wedge \underline{B{\leq}C}\ [4] \wedge C{\leq}A$$

$$\downarrow \qquad [2]\ \text{(transitivity)}$$

$$A{\leq}B \wedge B{\leq}C \wedge \underline{A{\leq}C}\ [4] \wedge \underline{C{\leq}A}$$

$$\downarrow \qquad [5]\ \text{(antisymmetry)}$$

$$A{\leq}B \wedge B{\leq}C \wedge \underline{A{=}C}$$

$$\downarrow$$

$$\ldots$$

Minimal States

For each rule, there is a minimal, most general state to which it is applicable.

Rule: $H \Leftrightarrow C \mid B$ or $H \Rightarrow C \mid B$

Minimal State: $H \wedge C$

Every other state to which the rule is applicable contains the minimal state (cf. Monotonicity/Incrementality).

Confluence

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

- Two states are called **joinable** if there exists states S_1', S_2' such that $S_1 \mapsto^* S_1'$ and $S_2 \mapsto^* S_2'$ and S_1' and S_2' are variants.
- A CHR program is called **confluent** if the following holds for all states S, S_1, S_2:

  If $S \mapsto^* S_1, S \mapsto^* S_2$ then S_1 and S_2 are joinable.

- A CHR is called **locally confluent** if the following holds for all states S, S_1, S_2:

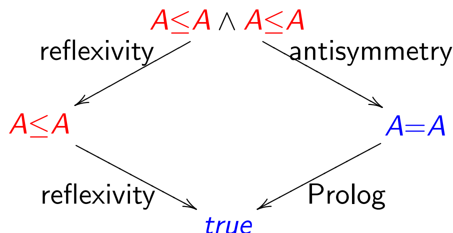
  If $S \mapsto S_1, S \mapsto S_2$ then S_1 and S_2 are joinable.

Confluence: Decidable Criterion

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

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

Start from two overlapping minimal states



Confluence: Critical Ancestor State

- Given a simplification rule R_1 and
- an arbitrary (not necessarily different) rule R_2
- Let $H_i \wedge A_i$ be the head and C_i be the guard of rule R_i ($i = 1, 2$).

A **critical ancestor state** of R_1 and R_2 is

$$(H_1 \wedge A_1 \wedge H_2 \wedge (A_1{=}A_2) \wedge C_1 \wedge C_2),$$

provided A_1 and A_2 are non-empty conjunctions and
$CT \models \exists((A_1{=}A_2) \wedge C_1 \wedge C_2)$.

Confluence: Sufficient and Necessary Condition

- Let S be a critical ancestor state of R_1 and R_2.
- If $S \mapsto S_1$ using rule R_1 and $S \mapsto S_2$ using rule R_2 then the tuple (S_1, S_2) is a **critical pair** of R_1 and R_2.
- A critical pair (S_1, S_2) is **joinable**, if S_1 and S_2 are joinable.

Theorem: A terminating CHR program is **confluent** if and only if all its critical pairs are joinable.

Confluence by Example

```
max(X,Y,Z) ⇔ X≤Y | Z=Y.
max(X,Y,Z) ⇔ Y≤X | Z=X.
max(X,Y,Z) ⇒ X≤Z ∧ Y≤Z.
```

The **critical ancestor state**

$$\mathtt{max}(\mathtt{X},\mathtt{Y},\mathtt{Z}) \wedge \mathtt{X} \leq \mathtt{Y}$$

of the first rule and the third one leads to the following **critical pair**:

$$(S_1, S_2) := (\mathtt{Z{=}Y} \wedge \mathtt{X{\leq}Y} \quad , \quad \mathtt{max(X,Y,Z)} \wedge \mathtt{X{\leq}Y} \wedge \mathtt{X{\leq}Z} \wedge \mathtt{Y{\leq}Z})$$

(S_1, S_2) is joinable since S_1 is a final state and the application of the first rule to S_2 results in S_1.

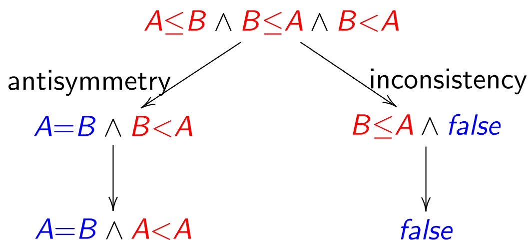
Challenges of the Confluence Proof

- **Challenge 1**: Avoid the **trivial nontermination** caused by the propagation rules
- **Solution**: keep track of the history of the application of the propagation rules.
- **Challenge 2:** The calculus should be **monotonic**, i.e. if a computation can be executed in a context, then the same computation can be executed in any extension of this context.
- **Solution:** Store information about propagation rules which can be possibly applied to a given set of user-defined constraints.

Completion

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

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



$$X < X \quad \Leftrightarrow \quad \mathit{fail} \qquad \text{(irreflexivity)}$$

Completion: Inference 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-Orientation:

$$[2] \quad \frac{(C \cup \{(S_1, S_2)\}, P) \qquad R = \mathit{rulify}_{\gg}(S_1, S_2)}{(C, P \cup R)}$$

CP-Deletion:

$$[3] \quad \frac{(C \cup \{(S_1, S_2)\}, P) \qquad S_1 \text{ and } S_2 \text{ are joinable}}{(C, P)}$$

CP-Simplification:

$$[4] \quad \frac{(C \cup \{(S_1, S_2)\}, P) \qquad S_1 \mapsto S_1'}{(C \cup \{(S_1', S_2)\}, P)}$$

Completion: Example (I)

```
and1 @ and(X,X,Z)  ⇔  X=Z.
and2 @ and(X,Y,X)  ⇔  imp(X,Y).
and3 @ and(X,Y,Z) ∧ and(X,Y,Z1)  ⇔  and(X,Y,Z) ∧ Z=Z1.
imp1 @ imp(X,Y) ∧ imp(Y,X)  ⇔  X=Y.
```

The program is not confluent. The completion procedure inserts the following rules:

```
[2] r1 @ imp(X,X)  ⇔  true.
r2 @ imp(X,Y) ∧ and(X,Y,Z)  ⇔  imp(X,Y) ∧ X=Z.
r3 @ imp(X,Y) ∧ imp(X,Y)  ⇔  imp(X,Y).
```

r1 stems from and2 and and1 [3]r2 stems from and2 and and3 [4]r3 stems from and2 and r2

Completion: Example (II)

```
r1 @ p(X,Y)  ⇔  X ≥ Y ∧ q(X,Y).
r2 @ p(X,Y)  ⇔  X ≤ Y ∧ r(X,Y).
```

P is not confluent, since the critical pair stemming from r1 and r2

$$(\mathtt{q}(\mathtt{X},\mathtt{Y}) \wedge \mathtt{X} \geq \mathtt{Y}, \mathtt{r}(\mathtt{X},\mathtt{Y}) \wedge \mathtt{X} \leq \mathtt{Y})$$

is non-joinable. The corresponding final states are

$$(\mathtt{q}(\mathtt{X},\mathtt{Y}) \wedge \mathtt{X} \geq \mathtt{Y}, \mathtt{r}(\mathtt{X},\mathtt{Y}) \wedge \mathtt{X} \leq \mathtt{Y})$$

Let r(X,Y) $\gg$ q(X,Y). Then the completion procedure inserts the following rules:

```
r3 @ r(X,Y)  ⇔  X ≤ Y | q(X,Y) ∧ X ≥ Y.
r4 @ q(X,Y)  ⇒  X ≥ Y | X ≤ Y.
```

Application of Confluence: Compatibility of Programs

Definition

Let P_1 and P_2 be two confluent and terminating CHR programs and let the union of the two programs, $P_1 \cup P_2$, be terminating. P_1 and P_2 are *compatible* if $P_1 \cup P_2$ is confluent.

Example `max`

$P1$: `max(X,Y,Z)` $\Leftrightarrow$ `X<Y` | `Z=Y.` `max(X,Y,Z)` $\Leftrightarrow$ `X≥Y` | `Z=X.`

$P2$: `max(X,Y,Z)` $\Leftrightarrow$ `X≤Y` | `Z=Y.` `max(X,Y,Z)` $\Leftrightarrow$ `X>Y` | `Z=X.`

3 critical ancestor states coming from one rule in P_1 and one rule in P_2 each:

- `max(X,Y,Z)` $\wedge$ `X<Y` $\wedge$ `X≤Y`
- `max(X,Y,Z)` $\wedge$ `X≥Y` $\wedge$ `X≤Y`
- `max(X,Y,Z)` $\wedge$ `X≥Y` $\wedge$ `X>Y`

All critical pairs stemming from the critical ancestor states are joinable.

Operational Equivalence

Given a goal and two programs, the results of the computation in both programs are the same.

- Let P_1, P_2 be CHR programs. A state S is called **P_1, P_2-joinable** if there are two computations $S \mapsto^*_{P_1} S_1$ and $S \mapsto^*_{P_2} S_2$ and S_1 and S_2 are variants.
- P_1 and P_2 are **operationally equivalent** if all states are P_1, P_2-joinable.

Compatibility vs. Operational Equivalence

Example `max`

$P1$: `max(X,Y,Z)` $\Leftrightarrow$ `X<Y | Z=Y.` `max(X,Y,Z)` $\Leftrightarrow$ `X≥Y | Z=X.`

$P2$: `max(X,Y,Z)` $\Leftrightarrow$ `X≤Y | Z=Y.` `max(X,Y,Z)` $\Leftrightarrow$ `X>Y | Z=X.`

- $P1$ and $P2$ are compatible.
- However not operationally equivalent:

$$\mathtt{max}(\mathtt{X},\mathtt{Y},\mathtt{Z}) \land \mathtt{X} \geq \mathtt{Y} \xrightarrow{P1} \mathtt{Z} = \mathtt{X} \land \mathtt{X} \geq \mathtt{Y}$$

$$\mathtt{max}(\mathtt{X},\mathtt{Y},\mathtt{Z}) \land \mathtt{X} \leq \mathtt{Y} \xrightarrow{P2} \mathtt{Z} = \mathtt{Y} \land \mathtt{X} \leq \mathtt{Y}$$

Operational Equivalence: Decidable Criterion

A decidable, sufficient and necessary condition for operational equivalence of terminating CHR programs through joinability of minimal states.

P_1 defines *max*

$$max(X,Y,Z) \Leftrightarrow X{<}Y \mid Z{=}Y.$$
$$max(X,Y,Z) \Leftrightarrow X{\geq}Y \mid Z{=}X.$$

P_2 defines *max*

$$max(X,Y,Z) \Leftrightarrow X{\leq}Y \mid Z{=}Y.$$
$$max(X,Y,Z) \Leftrightarrow X{>}Y \mid Z{=}X.$$

$$max(X,Y,Z) \wedge X{\geq}Y \quad \xrightarrow{P_1} \quad Z{=}X \wedge X{\geq}Y$$

$$max(X,Y,Z) \wedge X{\geq}Y \quad \xrightarrow{P_2}$$

Operational Equivalence: Sufficient and Necessary Condition

- Let P_1 and P_2 be CHR programs. A state S is **P_1, P_2-joinable**, iff there are two computations $S \mapsto^*_{P_1} S_1$ and $S \mapsto^*_{P_2} S_2$, where S_1 and S_2 are final states, and S_1 and S_2 are variants of each other.
- P_1 and P_2 are **operationally equivalent** if all states are P_1, P_2-joinable.
- The **set of critical states of P_1 and P_2** is defined as follows:

$$\{H \wedge C \mid (H \odot C \mid B) \in P_1 \cup P_2, \text{ where } \odot \in \{\Leftrightarrow, \Rightarrow\}\}$$

- **Theorem:** P_1 and P_2 are **operationally equivalent** iff all critical states of P_1 and P_2 are P_1, P_2-joinable.

Equivalence of Programs and Constraints (I)

P_1:

`p(a)` $\Leftrightarrow$ `s`. `p(b)` $\Leftrightarrow$ `r`. `s` $\wedge$`q` $\Leftrightarrow$ `true`.

P_2:

`p(a)` $\Leftrightarrow$ `s`. `p(b)` $\Leftrightarrow$ `r`.

- Are P_1 and P_2 operationally equivalent? **NO!**
- Are P_1 and P_2 operationally equivalent in `p`? **YES!**

Equivalence of Programs and Constraints (II)

P_1:

`p(a)` $\Leftrightarrow$ `s`. `p(b)` $\Leftrightarrow$ `r`. `s` $\wedge$`r` $\Leftrightarrow$ `true`.

P_2:

`p(a)` $\Leftrightarrow$ `s`. `p(b)` $\Leftrightarrow$ `r`.

- Are P_1 and P_2 operationally equivalent? **NO!**
- Are P_1 and P_2 operationally equivalent in `p`? **NO!**
- **Problem** `p` depends on `s` and `r`.

Equivalence of Constraints
A Sufficient Condition for Operational c-Equivalence

Let c be a CHR symbol. A ***c*-state** is a state where all user-defined constraints have the same CHR symbol c.

Let c be a CHR symbol defined in two CHR programs P_1 and P_2. P_1 and P_2 are **operationally *c*-equivalent** if all c-states are P_1, P_2-joinable.

The set of ***c*-critical states** is defined as follows:

$$\{H \wedge C \mid (H \odot C \mid B) \in P_1 \cup P_2, \text{ where } \odot \in \{\Leftrightarrow, \Rightarrow\} \text{ and } H \text{ contains only } c\text{-dependent CHR symbols}\}$$

Theorem: Let c be a CHR symbol defined in two confluent and terminating CHR programs P_1 and P_2. P_1 and P_2 are operationally c-equivalent if all c-critical states are P_1, P_2-joinable.

Example

P_1:

$\texttt{p(a)} \Leftrightarrow \texttt{s}.\ \ \texttt{p(b)} \Leftrightarrow \texttt{r}.\ \ \texttt{s} \wedge \texttt{r} \Leftrightarrow \texttt{true}.$

P_2:

$\texttt{p(a)} \Leftrightarrow \texttt{s}.\ \ \texttt{p(b)} \Leftrightarrow \texttt{r}.$

p depends on s and r.

The set of *p-critical states*:

$$\{\texttt{p(a)},\ \ \texttt{p(b)},\ \ \texttt{s} \wedge \texttt{r}\}$$

Overview of the course

- Part I: Analysis of CHR Programs
  - Confluence
  - Completion
  - Operational Equivalence
- **Part II: Automatic Generation of CHR Programs**
  - Generate and Test Approach
    - from an extensional definition
    - from an intentional definition
  - Symbolic Approach

Automatic Generation of CHR Programs: Motivation (I)

X	Y	$X \wedge Y$
0	0	0
0	1	0
1	0	0
1	1	1

First Trial:

```
and(0,0,0)  ⇔  true.
and(0,1,0)  ⇔  true.
and(1,0,0)  ⇔  true.
and(1,1,1)  ⇔  true.
```

Question: Is this good enough?

Automatic Generation of CHR Programs: Motivation (II)

X	Y	$X \wedge Y$
0	0	0
0	1	0
1	0	0
1	1	1

Better Solution:

```
and(X,X,Z)  ⇔  X=Z.
and(X,Y,1)  ⇔  X=Y.
and(X,1,Z)  ⇔  X=Z.
and(X,0,Z)  ⇔  Z=0.
and(1,Y,Z)  ⇔  Y=Z.
and(0,Y,Z)  ⇔  Z=0.
```

Automatic Generation of CHR Programs

Motivation

Writing a constraint solver is in general a difficult task

Example: Six-valued logic (ATPG, Van Hentenryck et al.): > 2000 cases

Goal:

Automatic generation of constraint solving algorithms in form of rules, where the user:

- gives extensional or intensional definitions of the constraints
- specifies the admissible syntactic form of the rules

Approach 1: Generate and Test

Step 1: Generation of propagation rules: PropMiner

$$
\begin{aligned}
and(0, Y, Z) &\Rightarrow Z{=}0.\\
and(X, Y, Z),\ neg(X, Y) &\Rightarrow Z{=}0.\\
and(X, Y, Z),\ or(Z, Y, Z1) &\Rightarrow Y{=}Z1.
\end{aligned}
$$

Step 2: Transformation of propagation rules into simplification rules: SimpMiner

$$
\begin{aligned}
and(0, Y, Z) &\Leftrightarrow Z{=}0.\\
and(X, Y, Z),\ neg(X, Y) &\Leftrightarrow neg(X, Y),\ Z{=}0.\\
and(X, Y, Z),\ or(Z, Y, Z1) &\Rightarrow Y{=}Z1.
\end{aligned}
$$

Approach 1: Generation of Propagation Rules

Syntax

$$C_L \quad \Rightarrow \quad C_R$$
$$C_L \quad \Rightarrow \quad \mathit{fail}$$

where C_L and C_R are sets of atomic constraints

PropMiner Algorithm

INPUT

- *Base*: constraints for which rules have to be generated
- $Cand_L$: candidate constraints for lhs
- $Cand_R$: candidate constraints for rhs
- Definition of *Base* and solvers for $Cand_L$ and $Cand_R$

ALGORITHM

$\forall\ C_L$ determine C_R as follows:

if $C_L \models \bot$, then $C_L \Rightarrow fail$

else $C_R = \{C_i \in Cand_R \ \mid C_L \models C_i\}$

if $C_R \neq \emptyset$, then $C_L \Rightarrow C_R$

Generation of Propagation Rules

Example: Boolean Conjunction

X	Y	$X \wedge Y$
0	0	0
0	1	0
1	0	0
1	1	1

- $Base = \{and(X, Y, Z)\}$
- $Cand_L = Cand_R = \{X{=}0, X{=}1, \ldots, Z{=}1, X{=}Y, X{=}Z, Y{=}Z\}$

$and(X, Y, Z)$
$and(X, Y, Z),\ X{=}0 \quad \Rightarrow \quad Z{=}0,\ X{=}Z.$
$and(X, Y, Z),\ X{=}0,\ Y{=}0 \quad \Rightarrow \quad Z{=}0,\ X{=}Z,\ Y{=}Z.$
$\ldots$
$and(X, Y, Z),\ X{=}Y \quad \Rightarrow \quad X{=}Z,\ Y{=}Z.$
$\ldots$

Generation of Propagation Rules

Pruning Strategies (C_L from general to specific)

1. If a rule $C_L \Rightarrow fail$ is generated then do not consider any superset of C_L.
2. If a rule $C_L \Rightarrow C_R$ is generated then do not consider any C such that $C_L \subset C$ and $C \cap C_R \neq \emptyset$.

Example:

$and(X, Y, Z),\ neg(A, B),\ A{=}X,\ B{=}Y \Rightarrow Z{=}0$
$and(X, Y, Z),\ neg(A, B),\ A{=}X,\ B{=}Y,\ B{=}1,\ Z{=}0$ NEGLECT
$and(X, Y, Z),\ neg(A, B),\ A{=}X,\ B{=}Y,\ B{=}1$ LEADS TO
$and(X, Y, Z),\ neg(A, B),\ A{=}X,\ B{=}Y,\ B{=}1 \Rightarrow$
$Z{=}0,\ A{=}0,\ X{=}0,\ Y{=}1.$

Generation of Propagation Rules

For Constraints Defined Intensionally

Having the definition of minimum:

$$
\begin{array}{lcl}
min(A,B,C) & \leftarrow & A \leq B, \quad C = A. \\
min(A,B,C) & \leftarrow & B \leq A, \quad C = B.
\end{array}
$$

How to automatically generate propagation rules:

$$
\begin{array}{rcl}
min(A,B,C) & \Rightarrow & C \leq A, \quad C \leq B. \\
min(A,A,C) & \Rightarrow & A = C. \\
min(A,B,C), \quad C \neq B & \Rightarrow & C = A. \\
min(A,B,C), \quad C \neq A & \Rightarrow & C = B. \\
min(A,B,C), \quad B \leq A & \Rightarrow & C = B. \\
min(A,B,C), \quad A \leq B & \Rightarrow & C = A.
\end{array}
$$

Generation of Propagation Rules

Definition of minimum:

$$\begin{array}{lcll} min(A,B,C) & \leftarrow & A{\leq}B, & C{=}A. \\ min(A,B,C) & \leftarrow & B{\leq}A, & C{=}B. \end{array}$$

$Base = \{min(A,B,C)\}$
$Cand_L = \{A{\leq}B, \ldots, C{\leq}B, A{=}B, \ldots, B{=}C, A{\neq}B, \ldots, B{\neq}C\}$

Goal: $min(A,B,C), \quad A{\leq}B$

Answers: $A{\leq}B, \quad A{=}C$ and $A{=}B, \quad A{=}C$

Least general generalization (lgg): $A{=}C$

The algorithm generates the rule: $min(A,B,C), \quad A{\leq}B \Rightarrow A{=}C$.

Generation of Propagation Rules

INPUT

- *Base*: constraints for which rules have to be generated
- $Cand_L$: candidate constraints for the left hand side
- solver (eventually not complete) for primitive constraints
- a CLP program P defining the constraints of interest

ALGORITHM

generate each possible left hand side C_L wrt. *Base* and $Cand_L$
  for each C_L determine C_R as follows
    let $\mathcal{A}$ be the set of answers for the goal C_L wrt. P
      if $\mathcal{A} = \emptyset$ then $C_L \Rightarrow fail$
      if $\mathcal{A}$ is finite then compute $C_R := lgg(\mathcal{A})$ $lgg \rightarrow$ [plotkin70]
        if $C_R \neq \emptyset$ then $C_L \Rightarrow C_R$

Generation of Propagation Rules

Limit of syntactic lgg

$$
\begin{array}{lcl}
min(A, B, C) & \leftarrow & A \leq B,\ C = A. \\
min(A, B, C) & \leftarrow & B \leq A,\ C = B.
\end{array}
$$

Goal: $min(A, B, C)$

Answers:
$A \leq B,\ A = C$ [2], $A \leq C,\ C \leq A,\ C \leq B$
$B \leq A,\ B = C$ [3], $B \leq C,\ C \leq B,\ C \leq A$

[1] lgg ?
[5] $C \leq A,\ C \leq B$

[6] The algorithm generates the rule

$$min(A, B, C) \Rightarrow C \leq A,\ C \leq B.$$

Generation of Propagation Rules

Interaction of *min* and *max*

Using rules for *min* and rules for *max*, many redundant rules are discarded. 10 propagation rules specific to the interaction of *min* and *max* are generated.

Example

$$
\begin{aligned}
min(A,B,C),\ max(D,E,F),\ C{\neq}E,\ C{\neq}D \quad &\Rightarrow \quad F{\neq}C.\\
min(A,B,C),\ max(D,E,F),\ B{\neq}D,\ A{\neq}D \quad &\Rightarrow \quad D{\neq}C.\\
min(A,B,C),\ max(D,E,F),\ C{\neq}E,\ B{\neq}D,\ A{\neq}F \quad &\Rightarrow \quad F{\neq}C.\\
min(A,B,C),\ max(D,E,F),\ C{\neq}D,\ B{\neq}F,\ A{\neq}E \quad &\Rightarrow \quad F{\neq}C.
\end{aligned}
$$

Generation of Propagation Rules for Recursive Definitions

- Bound the depth of the resolution
- Resolution based on the OLDT scheme

$$
\begin{array}{lcl}
append(X,Y,Z) & \leftarrow & X{=}[\,] \ \wedge \ Y{=}Z. \\
append(X,Y,Z) & \leftarrow & X{=}[H|X1] \ \wedge \ Z{=}[H|Z1] \ \wedge \ append(X1,Y,Z1).
\end{array}
$$

Example of rules (with a bounded resolution depth)

$$
\begin{array}{rcl}
append(A,B,C) \ \wedge \ A{=}B \ \wedge \ C{=}[D] & \Rightarrow & fail. \\
append(A,B,C) \ \wedge \ B{=}C \ \wedge \ C{=}[D] & \Rightarrow & A{=}[\,]. \\
append(A,B,C) \ \wedge \ C{=}[\,] & \Rightarrow & B{=}[\,] \ \wedge \ A{=}[\,]. \\
append(A,B,C) \ \wedge \ A{=}[\,] & \Rightarrow & B{=}C.
\end{array}
$$

Generation of Simplification Rules

Motivation

- Propagation rules do not rewrite constraints but add new ones
- Simplification rules remove constraints from the constraint store

Removing constraints

- allows saving of space
- decreases the cost of constraint solving

Problem

Find criteria to transform some propagation rules into simplification rules

Generation of Simplification Rules

Semantical Criterion: Same Solutions

Example: $and(X, Y, Z),\ neg(X, Y)\ \Rightarrow\ Z{=}0$

1. $and(X, Y, Z),\ neg(X, Y)\ \Leftrightarrow\ Z{=}0$
   Counterexample: $X{=}0, Y{=}0, Z{=}0$
2. $and(X, Y, Z),\ neg(X, Y) \Leftrightarrow and(X, Y, Z),\ Z{=}0$
   Counterexample: $X{=}0, Y{=}0, Z{=}0$
3. $and(X, Y, Z),\ neg(X, Y) \Leftrightarrow neg(X, Y),\ Z{=}0$

Generation of Simplification Rules

SimpMiner Algorithm

INPUT: A set P of propagation rules

OUTPUT: A set P' consisting of propagation and simplification rules

ALGORITHM:

$P' := \emptyset$

for each rule R of the form $H \Rightarrow B$ in P **do**

Find $R' := H \Leftrightarrow B \wedge C$ with $C \subset H$ such that H and $B \wedge C$ have the same solutions

If R' exists

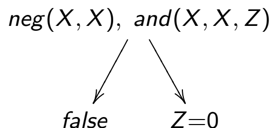
then $P' := P' \cup \{R'\}$

else $P' := P' \cup \{R\}$

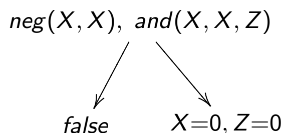
Generation of Simplification Rules

Example: $and(X, Y, Z),\ neg(X, Y)\ \Rightarrow\ Z{=}0$

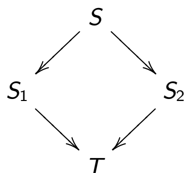
1. $and(X, Y, Z),\ neg(X, Y)\ \Leftrightarrow\ Z{=}0$



2. $and(X, Y, Z),\ neg(X, Y) \Leftrightarrow and(X, Y, Z),\ Z{=}0$



3. $and(X, Y, Z),\ neg(X, Y) \Leftrightarrow neg(X, Y),\ Z{=}0$



Generation of Simplification Rules

SimpMiner Algorithm

INPUT: A set P consisting of propagation rules
OUTPUT: A set P' consisting of propagation and simplification rules
ALGORITHM:

$P' := P$

for each rule R of the form $H \Rightarrow B$ in P **do**

Find $R' := H \Leftrightarrow B \wedge C$ with $C \subset H$ such that $(P' \backslash \{R\}) \cup \{R'\}$ is terminating and confluent.

If R' exists

then $P' := (P' \backslash \{R\}) \cup \{R'\}$

Generation of Simplification Rules

Example

$$\begin{array}{rcl} min(A,B,C) & \Rightarrow & C{\leq}A, \quad C{\leq}B. \\ min(A,A,C) & \Rightarrow & A{=}C. \\ min(A,B,C), \quad C{\neq}B & \Rightarrow & C{=}A. \\ min(A,B,C), \quad C{\neq}A & \Rightarrow & C{=}B. \\ min(A,B,C), \quad B{\leq}A & \Rightarrow & C{=}B. \\ min(A,B,C), \quad A{\leq}B & \Rightarrow & C{=}A. \end{array}$$

is transformed by SimpMiner into

$$\begin{array}{rcl} min(A,B,C) & \Rightarrow & C{\leq}A, \quad C{\leq}B. \\ min(A,A,C) & \Leftrightarrow & A{=}C. \\ min(A,B,C), \quad C{\neq}B & \Rightarrow & C{=}A. \\ min(A,B,C), \quad C{\neq}A & \Rightarrow & C{=}B. \\ min(A,B,C), \; B{\leq}A & \Leftrightarrow & C{=}B, \; B{\leq}A. \\ min(A,B,C), \; A{\leq}B & \Leftrightarrow & C{=}A, \; A{\leq}B. \end{array}$$

Approach 2: Symbolic Construction of CHR Programs: Basic Idea

Given the intentional definition:

$$append(A, B, C) \leftarrow A{=}[] \wedge C{=}B.$$
$$append(A, B, C) \leftarrow A{=}[D|E] \wedge C{=}[F|G] \wedge D{=}F \wedge append(E, B, G).$$

- If the execution of one clause leads to a solution, then the execution of all other clauses will not.
- To construct a rule that simplifies the constraint to the body of one clause: Add the negation of the bodies of all other clauses to the guard.
- Enuring that the rule is applicable only when all other clauses are not valid
- This maintains the consistency with the constraint definition

Approach 2: Symbolic Construction of CHR Programs

Given the intentional definition:

$$append(A, B, C) \leftarrow A{=}[] \wedge C{=}B.$$
$$append(A, B, C) \leftarrow A{=}[D|E] \wedge C{=}[F|G] \wedge D{=}F \wedge append(E, B, G).$$

The following rules are generated by construction:

$$append(A, B, C) \Leftrightarrow A{=}[] \mid C{=}B \wedge A{=}[].$$
$$append(A, B, C) \Leftrightarrow C{=}[] \mid C{=}B \wedge A{=}[] \wedge C{=}[].$$
$$append(A, B, C) \Leftrightarrow A{\neq}[] \mid A{=}[D|E] \wedge C{=}[F|G] \wedge D{=}F \wedge append(E, B, G).$$
$$append(A, B, C) \Leftrightarrow C{\neq}B \mid A{=}[D|E] \wedge C{=}[F|G] \wedge D{=}F \wedge append(E, B, G) \wedge C{\neq}B.$$

ConstructRules Algorithm by Example (I)

$$append(A, B, C) \leftarrow A{=}[] \wedge C{=}B.$$
$$append(A, B, C) \leftarrow A{=}[D|E] \wedge C{=}[F|G] \wedge D{=}F \wedge append(E, B, G).$$

Rules for the first clause are generated as follows:

- Head: $append(A, B, C)$
- Body: $A{=}[] \wedge C{=}B$
- Guard: Negation of the body of the second clause:

$$\neg(A{=}[D|E] \wedge C{=}[F|G] \wedge D{=}F \wedge append(E, B, G))$$

  - The result after simplification:

$$A{=}[] \vee C{=}[] \vee (D{\neq}F \wedge A{=}[D|E] \wedge C{=}[F|G])$$

ConstructRules Algorithm by Example (II)

- Resulting rules for the **first clause**:

$$append(A, B, C) \wedge A{=}[] \Leftrightarrow A{=}[] \wedge C{=}B.$$

$$append(A, B, C) \wedge C{=}[] \Leftrightarrow A{=}[] \wedge C{=}B \wedge C{=}[].$$

$$append(A, B, C) \wedge D{\neq}F \wedge A{=}[D|E] \wedge C{=}[F|G] \Leftrightarrow A{=}[] \wedge C{=}B \wedge D{\neq}F \wedge A{=}[D|E] \wedge C{=}[F|G].$$

- Resulting rules for the **second clause**:

$$append(A, B, C) \wedge A{\neq}[] \Leftrightarrow A{=}[D|E] \wedge C{=}[F|G] \wedge D{=}F \wedge append(E, B, G) \wedge A{\neq}[].$$

$$append(A, B, C) \wedge C{\neq}B \Leftrightarrow A{=}[D|E] \wedge C{=}[F|G] \wedge D{=}F \wedge append(E, B, G) \wedge C{\neq}B.$$

ConstructRules Algorithm: Simplification

- Consider the rule:

$$append(A, B, C) \wedge D{\neq}F \wedge A{=}[D|E] \wedge C{=}[F|G] \Leftrightarrow \underline{A{=}[]} \wedge C{=}B \wedge D{\neq}F \wedge \underline{A{=}[D|E]} \wedge C{=}[F|G].$$

- The existence of the constraints $A{=}[]$ and $A = [D|E]$ leads to a contradiction
- The rule can be simplified to:

$$append(A, B, C) \wedge D{\neq}F \wedge A{=}[D|E] \wedge C{=}[F|G] \Leftrightarrow fail.$$

ConstructRules Algorithm

begin

H: the **head** of the clauses.

B: the set of **bodies** of the clauses.

R: the set of **generated rules** initialized to [].

while B is not empty **do**

Remove from B its first element denoted C_i.

$Other_B$: the set of all clause bodies except C_i.

G_i: the set resulting from negating $Other_B$.

while G_i is not empty **do**

Remove from G_i its first element denoted G_i^j.

Add rule $(H \wedge G_i^j \Leftrightarrow C_i \wedge G_i^j)$ to R.

end while

end while

end

ConstructRules Algorithm: Redundancy

- **Approach**: Using the operational equivalence results of before
- **Idea**: check if the computation step due to the candidate rule that is tested for redundancy can be performed by the remainder of the program
- Execute the minimal states in both programs and comparing the results. If the results are identical, then the rule is obviously redundant and can be removed.

Combining PropMiner with ConstructRules

- First construct valid rules using the ConstructRules algorithm
- Use the generated rules to prune the search tree of the PropMiner algorithm
- Remove the redundant rules using the redundancy algorithms presented before
  - Closure Pruning
  - Redundancy Check

Conclusion

Constraint Handling Rules

- Declarative language for constraint programming
- Executable specification and rapid prototyping
- Good theoretical properties
- Semi-automatic generation of CHR programs using some of the investigated properties

Literature

- Slim Abdennadher: Operational Semantics and Confluence of Constraint Propagation Rules, CP97, LNCS, Springer, 1997.
- Slim Abdennadher and Thom Fruehwirth and Holger Meuss: Confluence and Semantics of Constraint Simplification Rules, Journal Constraints, Kluwer Academic Publishers, 1999.
- Slim Abdennadher, Thom Fruewhwirth: On Completion of Constraint Handling Rules, CP98, Springer LNCS, 1998.
- Slim Abdennadher and Thom Fruehwirth: Operational Equivalence of CHR Programs And Constraints, CP99, Springer LNCS, 1999.
- S. Abdennadher and C. Rigotti: Automatic Generation of Propagation Rules for Finite Domains, CP 2000, 2000.
- Slim Abdennadher and C. Rigotti: Automatic Generation of CHR Constraint Solvers, Journal of Theory and Practice of Logic Programming (TPLP), 2005.
- Ingi Sobhi, Slim Abdennadher, Hariolf Betz: Constructing Rule-based Solvers for Intentionally-defined Constraints, Special Issue on Recent Advances in Constraint Handling Rules, LNAI, 2008.