Constraint Handling Rules
The Story So Far

Thom Frühwirth

Faculty of Computer Science
University of Ulm, Germany

PPDP'06 Venice
July 2006

Citations may be not recent and incomplete for space reasons

Introduction

Rules' Renaissance

- Business Rules
- Semantic Web
- Data Mining
- Verification/Security

Overview

- The CHR Language, Properties, Analysis
- Small Example Programs, Constraint Solvers
- Classical Applications, Trends, Projects

Part I

The CHR Language

Example Partial Order Constraint

$X \leq X$	$\Leftrightarrow$	$true$	(reflexivity)
$X \leq Y \wedge Y \leq X$	$\Leftrightarrow$	$X = Y$	(antisymmetry)
$X \leq Y \wedge Y \leq Z$	$\Rightarrow$	$X \leq Z$	(transitivity)

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

$$\downarrow \quad \text{(transitivity)}$$

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

$$\downarrow \quad \text{(antisymmetry)}$$

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

$$\| \quad \text{[built-in solver]}$$

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

$$\downarrow \quad \text{(antisymmetry)}$$

$$A = B \wedge A = C$$

Example Partial Order Constraint

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

$$
\begin{array}{cl}
\underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A & \\
\downarrow & \text{(transitivity)} \\
A \leq B \wedge B \leq C \wedge \underline{C \leq A} \wedge \underline{A \leq C} & \\
\downarrow & \text{(antisymmetry)} \\
A \leq B \wedge B \leq C \wedge \underline{A = C} & \\
\| & \text{[built-in solver]} \\
\underline{A \leq B} \wedge \underline{B \leq A} \wedge A = C & \\
\downarrow & \text{(antisymmetry)} \\
A = B \wedge A = C &
\end{array}
$$

Example Partial Order Constraint

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

$$
\begin{array}{cl}
\underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A & \\
\downarrow & \text{(transitivity)} \\
A \leq B \wedge B \leq C \wedge \underline{C \leq A} \wedge \underline{A \leq C} & \\
\downarrow & \text{(antisymmetry)} \\
A \leq B \wedge B \leq C \wedge \underline{A = C} & \\
\| & \text{[built-in solver]} \\
\underline{A \leq B} \wedge \underline{B \leq A} \wedge A = C & \\
\downarrow & \text{(antisymmetry)} \\
A = B \wedge A = C &
\end{array}
$$

Example Partial Order Constraint

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

$$
\begin{array}{cl}
\underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A & \\
\downarrow & \text{(transitivity)} \\
A \leq B \wedge B \leq C \wedge \underline{C \leq A} \wedge \underline{A \leq C} & \\
\downarrow & \text{(antisymmetry)} \\
A \leq B \wedge B \leq C \wedge \underline{A = C} & \\
\| & \text{[built-in solver]} \\
\underline{A \leq B} \wedge \underline{B \leq A} \wedge A = C & \\
\downarrow & \text{(antisymmetry)} \\
A = B \wedge A = C &
\end{array}
$$

Example Partial Order Constraint

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

$$
\begin{array}{cl}
\underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A & \\
\downarrow & \text{(transitivity)} \\
A \leq B \wedge B \leq C \wedge \underline{C \leq A} \wedge \underline{A \leq C} & \\
\downarrow & \text{(antisymmetry)} \\
A \leq B \wedge B \leq C \wedge \underline{A = C} & \\
\| & \text{[built-in solver]} \\
\underline{A \leq B} \wedge \underline{B \leq A} \wedge A = C & \\
\downarrow & \text{(antisymmetry)} \\
A = B \wedge A = C &
\end{array}
$$

Example Partial Order Constraint

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

$$\begin{array}{cl} \underline{A{\leq}B} \wedge \underline{B{\leq}C} \wedge C{\leq}A & \\ \downarrow & \text{(transitivity)} \\ A{\leq}B \wedge B{\leq}C \wedge \underline{C{\leq}A} \wedge \underline{A{\leq}C} & \\ \downarrow & \text{(antisymmetry)} \\ A{\leq}B \wedge B{\leq}C \wedge \underline{A{=}C} & \\ \| & \text{[built-in solver]} \\ \underline{A{\leq}B} \wedge \underline{B{\leq}A} \wedge A{=}C & \\ \downarrow & \text{(antisymmetry)} \\ A{=}B \wedge A{=}C & \end{array}$$

Example Partial Order Constraint

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

$$
\begin{array}{cl}
\underline{A{\leq}B} \wedge \underline{B{\leq}C} \wedge C{\leq}A & \\
\downarrow & \text{(transitivity)} \\
A{\leq}B \wedge B{\leq}C \wedge \underline{C{\leq}A} \wedge \underline{A{\leq}C} & \\
\downarrow & \text{(antisymmetry)} \\
A{\leq}B \wedge B{\leq}C \wedge \underline{A{=}C} & \\
\| & \text{[built-in solver]} \\
\underline{A{\leq}B} \wedge \underline{B{\leq}A} \wedge A{=}C & \\
\downarrow & \text{(antisymmetry)} \\
A{=}B \wedge A{=}C &
\end{array}
$$

Syntax and Declarative Semantics

		Declarative Semantics
Simplification rule:	$H \Leftrightarrow C \mid B$	$\forall \bar{x}\,(C \rightarrow (H \leftrightarrow \exists \bar{y}\ B))$
Propagation rule:	$H \Rightarrow C \mid B$	$\forall \bar{x}\,(C \rightarrow (H \rightarrow \exists \bar{y}\ B))$
		Constraint Theory for Built-Ins

- Head H: non-empty conjunction of CHR constraints
- Guard C: conjunction of built-in constraints
- Body B: conjunction of CHR and built-in constraints (goal)

Soundness and Completeness based on logical equivalence of states in a computation.

Operational Semantics

Apply rules until exhaustion in any order (fixpoint computation).
Initial goal (query) $\mapsto^*$ result (answer).

Simplify

If	$(H \Leftrightarrow C \mid B)$ rule with renamed fresh variables $\bar{x}$
and	$CT \models G_{builtin} \rightarrow \exists\bar{x}(H{=}H' \wedge C)$
then	$H' \wedge G \mapsto G \wedge H{=}H' \wedge B$

Propagate

If	$(H \Rightarrow C \mid B)$ rule with renamed fresh variables $\bar{x}$
and	$CT \models G_{builtin} \rightarrow \exists\bar{x}(H{=}H' \wedge C)$
then	$H' \wedge G \mapsto H' \wedge G \wedge H{=}H' \wedge B$

Refined operational semantics [Duck+, ICLP 2004]: Similar to procedure calls, CHR constraints evaluated depth-first from left to right and rules applied top-down in program text order. *Active vs. Partner constraint.*

Operational Semantics

Apply rules until exhaustion in any order (fixpoint computation).
Initial goal (query) $\mapsto^*$ result (answer).

Simplify

If	$(H \Leftrightarrow C \mid B)$ rule with renamed fresh variables $\bar{x}$
and	$CT \models G_{builtin} \rightarrow \exists \bar{x}(H{=}H' \wedge C)$
then	$H' \wedge G \mapsto G \wedge H{=}H' \wedge B$

Propagate

If	$(H \Rightarrow C \mid B)$ rule with renamed fresh variables $\bar{x}$
and	$CT \models G_{builtin} \rightarrow \exists \bar{x}(H{=}H' \wedge C)$
then	$H' \wedge G \mapsto H' \wedge G \wedge H{=}H' \wedge B$

Refined operational semantics [Duck+, ICLP 2004]: Similar to procedure calls, CHR constraints evaluated depth-first from left to right and rules applied top-down in program text order. *Active vs. Partner constraint.*

Operational Semantics

Apply rules until exhaustion in any order (fixpoint computation).
Initial goal (query) $\mapsto^*$ result (answer).

Simplify

If	$(H \Leftrightarrow C \mid B)$ rule with renamed fresh variables $\bar{x}$
and	$CT \models G_{builtin} \rightarrow \exists\bar{x}(H{=}H' \wedge C)$
then	$H' \wedge G \mapsto G \wedge H{=}H' \wedge B$

Propagate

If	$(H \Rightarrow C \mid B)$ rule with renamed fresh variables $\bar{x}$
and	$CT \models G_{builtin} \rightarrow \exists\bar{x}(H{=}H' \wedge C)$
then	$H' \wedge G \mapsto H' \wedge G \wedge H{=}H' \wedge B$

Refined operational semantics [Duck+, ICLP 2004]: Similar to procedure calls, CHR constraints evaluated depth-first from left to right and rules applied top-down in program text order. *Active vs. Partner constraint.*

Operational Semantics

Apply rules until exhaustion in any order (fixpoint computation).
Initial goal (query) $\mapsto^*$ result (answer).

Simplify

If	$(H \Leftrightarrow C \mid B)$ rule with renamed fresh variables $\bar{x}$
and	$CT \models G_{builtin} \rightarrow \exists \bar{x}(H{=}H' \wedge C)$
then	$H' \wedge G \mapsto G \wedge H{=}H' \wedge B$

Propagate

If	$(H \Rightarrow C \mid B)$ rule with renamed fresh variables $\bar{x}$
and	$CT \models G_{builtin} \rightarrow \exists \bar{x}(H{=}H' \wedge C)$
then	$H' \wedge G \mapsto H' \wedge G \wedge H{=}H' \wedge B$

Refined operational semantics [Duck+, ICLP 2004]: Similar to procedure calls, CHR constraints evaluated depth-first from left to right and rules applied top-down in program text order. *Active vs. Partner constraint.*

Anytime Algorithm - Approximation

Computation can be interrupted and restarted at any time.
Intermediate results approximate final result.

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A$$
$$\downarrow \qquad \text{(transitivity)}$$
$$A \leq B \wedge B \leq C \wedge \underline{C \leq A} \wedge \underline{A \leq C}$$
$$\downarrow \qquad \text{(antisymmetry)}$$
$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge A = C$$
$$\downarrow \qquad \text{(antisymmetry)}$$
$$A = B \wedge A = C$$

Anytime Algorithm - Approximation

Computation can be interrupted and restarted at any time.
Intermediate results approximate final result.

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A$$
$$\downarrow \qquad \text{(transitivity)}$$
$$A \leq B \wedge B \leq C \wedge \underline{C \leq A} \wedge \underline{A \leq C}$$
$$\downarrow \qquad \text{(antisymmetry)}$$
$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge A = C$$
$$\downarrow \qquad \text{(antisymmetry)}$$
$$A = B \wedge A = C$$

Anytime Algorithm - Approximation

Computation can be interrupted and restarted at any time.
Intermediate results approximate final result.

$$\underline{A{\leq}B} \wedge \underline{B{\leq}C} \wedge C{\leq}A$$
$$\downarrow \qquad \text{(transitivity)}$$
$$A{\leq}B \wedge B{\leq}C \wedge \underline{C{\leq}A} \wedge \underline{A{\leq}C}$$
$$\downarrow \qquad \text{(antisymmetry)}$$
$$\underline{A{\leq}B} \wedge \underline{B{\leq}C} \wedge A{=}C$$
$$\downarrow \qquad \text{(antisymmetry)}$$
$$A{=}B \wedge A{=}C$$

Online Algorithm - Incremental

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

Monotonicity and Incrementality

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

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

$$\downarrow \qquad \text{(transitivity)}$$

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

$$\downarrow \qquad \text{(antisymmetry)}$$

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

$$\downarrow$$

$$\ldots$$

Online Algorithm - Incremental

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

Monotonicity and Incrementality

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

$$\begin{array}{c} \underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A \\ \downarrow \\ A \leq B \wedge B \leq C \wedge \underline{A \leq C} \;\; \wedge \underline{C \leq A} \\ \downarrow \\ A \leq B \wedge B \leq C \wedge \underline{A = C} \\ \downarrow \\ \ldots \end{array}$$

(transitivity)

(antisymmetry)

Online Algorithm - Incremental

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

Monotonicity and Incrementality

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

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A$$
$$\downarrow \quad \text{(transitivity)}$$
$$A \leq B \wedge B \leq C \wedge \underline{A \leq C} \wedge \underline{C \leq A}$$
$$\downarrow \quad \text{(antisymmetry)}$$
$$A \leq B \wedge B \leq C \wedge \underline{A = C}$$
$$\downarrow$$
$$\ldots$$

Online Algorithm - Incremental

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

Monotonicity and Incrementality

$$
\begin{array}{llcl}
\text{If} & G & \longmapsto & G' \\
\text{then} & G \land C & \longmapsto & G' \land C
\end{array}
$$

$$
\begin{array}{cl}
\underline{A{\leq}B} \land \underline{B{\leq}C} \land C{\leq}A & \\
\downarrow & \text{(transitivity)} \\
A{\leq}B \land B{\leq}C \land \underline{A{\leq}C} \land \underline{C{\leq}A} & \\
\downarrow & \text{(antisymmetry)} \\
A{\leq}B \land B{\leq}C \land \underline{A{=}C} & \\
\downarrow & \\
\ldots &
\end{array}
$$

Concurrency - Weak Parallelism

Rules can be applied in parallel to different parts of the problem.

$$
\begin{array}{llll}
\text{If} & A & \longmapsto & B \\
\text{and} & C & \longmapsto & D \\
\text{then} & A \wedge C & \longmapsto & B \wedge D
\end{array}
$$

$$
\begin{array}{ccc}
\underline{A{\leq}B} \wedge \underline{B{\leq}C} & \wedge & C{\leq}D \wedge D{\leq}A \\
\downarrow & & \downarrow \\
A{\leq}B \wedge B{\leq}C \wedge \underline{A{\leq}C} & \wedge & C{\leq}D \wedge D{\leq}A \wedge \underline{C{\leq}A} \\
 & \downarrow & \\
 & \ldots\ A{=}C\ \ldots &
\end{array}
$$

Concurrency - Weak Parallelism

Rules can be applied in parallel to different parts of the problem.

$$\begin{array}{llll} \text{If} & A & \longmapsto & B \\ \text{and} & C & \longmapsto & D \\ \text{then} & A \wedge C & \longmapsto & B \wedge D \end{array}$$

$$\begin{array}{ccc} \underline{A{\leq}B} \wedge \underline{B{\leq}C} & \wedge & C{\leq}D \wedge D{\leq}A \\ \downarrow & & \downarrow \\ A{\leq}B \wedge B{\leq}C \wedge \underline{A{\leq}C} & \wedge & C{\leq}D \wedge D{\leq}A \wedge \underline{C{\leq}A} \\ & \downarrow & \\ & \ldots\ A{=}C\ \ldots & \end{array}$$

Concurrency - Weak Parallelism

Rules can be applied in parallel to different parts of the problem.

$$\begin{array}{llcl} \text{If} & A & \longmapsto & B \\ \text{and} & C & \longmapsto & D \\ \text{then} & A \wedge C & \longmapsto & B \wedge D \end{array}$$

$$\begin{array}{ccc} \underline{A{\leq}B} \wedge \underline{B{\leq}C} & \wedge & C{\leq}D \wedge D{\leq}A \\ \downarrow & & \downarrow \\ A{\leq}B \wedge B{\leq}C \wedge \underline{A{\leq}C} & \wedge & C{\leq}D \wedge D{\leq}A \wedge \underline{C{\leq}A} \\ & \downarrow & \\ & \ldots\ A{=}C\ \ldots & \end{array}$$

Concurrency - Weak Parallelism

Rules can be applied in parallel to different parts of the problem.

$$
\begin{array}{llcl}
\text{If} & A & \longmapsto & B \\
\text{and} & C & \longmapsto & D \\
\text{then} & A \wedge C & \longmapsto & B \wedge D
\end{array}
$$

$$
\begin{array}{ccc}
\underline{A \leq B} \wedge \underline{B \leq C} & \wedge & C \leq D \wedge D \leq A \\
\downarrow & & \downarrow \\
A \leq B \wedge B \leq C \wedge \underline{A \leq C} & \wedge & C \leq D \wedge D \leq A \wedge \underline{C \leq A} \\
 & \downarrow & \\
 & \ldots\ A = C\ \ldots &
\end{array}
$$

Concurrency - Strong Parallelism

Interleaving semantics: Parallel computation step can be simulated by a sequence of sequential computation steps.

Rules can be applied in parallel to **overlapping parts** of a goal, **if** overlap is not removed.

$$
\begin{array}{llcl}
\text{If} & A \wedge E & \longmapsto & B \wedge E \\
\text{and} & C \wedge E & \longmapsto & D \wedge E \\
\text{then} & A \wedge C \wedge E & \longmapsto & B \wedge D \wedge E
\end{array}
$$

$$
\begin{array}{ccccc}
\underline{A \leq B} & \wedge & \underline{B \leq C} & \wedge & \underline{C \leq A} \\
\downarrow & & & & \downarrow \\
A \leq B \wedge \underline{A \leq C} & \wedge & B \leq C & \wedge & C \leq A \wedge \underline{B \leq A} \\
 & & \downarrow & & \\
 & & \ldots \; A = C \; \ldots & &
\end{array}
$$

Concurrency - Strong Parallelism

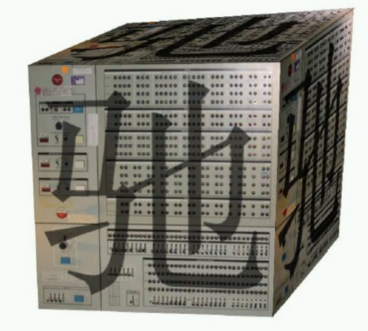
Interleaving semantics: Parallel computation step can be simulated by a sequence of sequential computation steps.

Rules can be applied in parallel to **overlapping parts** of a goal, **if** overlap is not removed.

$$\begin{array}{llcl}
\text{If} & A \wedge E & \longmapsto & B \wedge E \\
\text{and} & C \wedge E & \longmapsto & D \wedge E \\
\text{then} & A \wedge C \wedge E & \longmapsto & B \wedge D \wedge E
\end{array}$$

$$\begin{array}{ccccc}
\underline{A \leq B} & \wedge & \underline{B \leq C} & \wedge & \underline{C \leq A} \\
\downarrow & & & & \downarrow \\
A \leq B \wedge \underline{A \leq C} & \wedge & B \leq C & \wedge & C \leq A \wedge \underline{B \leq A} \\
 & & \downarrow & & \\
 & & \ldots \; A = C \; \ldots & &
\end{array}$$

Optimal Time and Space Complexity



c Jon Sneyers, K.U. Leuven

The CHR Machine
Sublanguage of CHR.
Can be mapped to Turing machines and vice versa.
CHR is Turing-complete.
Can be mapped to RAM machines and vice versa.
Every algorithm can be implemented in CHR with best known time and space complexity.
[Sneyers,Schrijvers,Demoen, CHR'05]
Practical Evidence: Union-Find, Shortest Paths, Fibonacci Heap Algorithms.

CHR Program Analysis

Prove that... [Abdennadher,Frühwirth]

Termination
Every computation starting from any goal ends. [LNAI 1865, 2000]

Complexity
Worst-case time complexity follows from structure of rules. [KR'02]

Consistency and Correctness
Logical reading of the rules is consistent and follows from a specification. [Constraints Journal 2000]

Decidable **Confluence**
The answer of a query is always the same, no matter which of the applicable rules are applied. [CP'96, CP'97, Constraints Journal 2000]

Completion
Non-confluent programs made confluent by adding rules. [CP'98]

Decidable **Operational Equivalence**
Two programs have the same results for any given query. [CP'99]

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).

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).

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

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

Start from overlapping minimal states

$$
\begin{tikzcd}
 & A \leq A \wedge A \leq A \arrow[dl, "\text{reflexivity}"'] \arrow[dr, "\text{antisymmetry}"] & \\
A \leq A \arrow[dr, "\text{reflexivity}"'] & & A = A \arrow[dl, "\text{built-in}"] \\
 & \mathit{true} &
\end{tikzcd}
$$

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.

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

Start from overlapping minimal states

$$
\begin{tikzcd}
 & A{\leq}A \wedge A{\leq}A \arrow[dl, "\text{reflexivity}"'] \arrow[dr, "\text{antisymmetry}"] & \\
A{\leq}A \arrow[dr, "\text{reflexivity}"'] & & A{=}A \arrow[dl, "\text{built-in}"] \\
 & \mathit{true} &
\end{tikzcd}
$$

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.

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

Start from overlapping minimal states

$$
\begin{tikzcd}
 & A{\leq}A \wedge A{\leq}A \arrow[dl, "\text{reflexivity}"'] \arrow[dr, "\text{antisymmetry}"] & \\
A{\leq}A \arrow[dr, "\text{reflexivity}"'] & & A{=}A \arrow[dl, "\text{built-in}"] \\
 & \mathit{true} &
\end{tikzcd}
$$

Completion

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

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

$$\begin{tikzcd}
 & A \leq B \wedge B \leq A \wedge B < A \arrow[dl, "\text{antisymmetry}"'] \arrow[dr, "\text{inconsistency}"] & \\
A = B \wedge B < A \arrow[d] & & B \leq A \wedge \mathit{false} \arrow[d] \\
A = B \wedge A < A & & \mathit{false}
\end{tikzcd}$$

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

Completion

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

$X{\leq}Y \wedge Y{\leq}X$	$\Leftrightarrow$	$X{=}Y$	(antisymmetry)
$X{\leq}Y \wedge Y{<}X$	$\Leftrightarrow$	*false*	(inconsistency)

$$A{\leq}B \wedge B{\leq}A \wedge B{<}A$$

antisymmetry: $A{\leq}B \wedge B{\leq}A \wedge B{<}A \;\longmapsto\; A{=}B \wedge B{<}A \;\longmapsto\; A{=}B \wedge A{<}A$

inconsistency: $A{\leq}B \wedge B{\leq}A \wedge B{<}A \;\longmapsto\; B{\leq}A \wedge \mathit{false} \;\longmapsto\; \mathit{false}$

$X{<}X$	$\Leftrightarrow$	*false*	(irreflexivity)

Completion

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

$$
\begin{array}{llll}
X \leq Y \wedge Y \leq X & \Leftrightarrow & X = Y & \text{(antisymmetry)} \\
X \leq Y \wedge Y < X & \Leftrightarrow & \mathit{false} & \text{(inconsistency)}
\end{array}
$$

$$
\begin{tikzcd}
 & A \leq B \wedge B \leq A \wedge B < A \arrow[dl, "\text{antisymmetry}"'] \arrow[dr, "\text{inconsistency}"] & \\
A = B \wedge B < A \arrow[d] & & B \leq A \wedge \mathit{false} \arrow[d] \\
A = B \wedge A < A & & \mathit{false}
\end{tikzcd}
$$

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

Completion

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

$$
\begin{array}{llll}
X{\leq}Y \wedge Y{\leq}X & \Leftrightarrow & X{=}Y & \text{(antisymmetry)} \\
X{\leq}Y \wedge Y{<}X & \Leftrightarrow & \mathit{false} & \text{(inconsistency)}
\end{array}
$$

$$
\begin{tikzcd}
 & A{\leq}B \wedge B{\leq}A \wedge B{<}A \arrow[dl, "\text{antisymmetry}"'] \arrow[dr, "\text{inconsistency}"] & \\
A{=}B \wedge B{<}A \arrow[d] & & B{\leq}A \wedge \mathit{false} \arrow[d] \\
A{=}B \wedge A{<}A & & \mathit{false}
\end{tikzcd}
$$

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

Operational Equivalence

Given a goal and two programs, computations in both programs leads to the same result.
A decidable, sufficient and necessary condition for operational equivalence of terminating CHR programs through joinability of minimal states.

$$
\begin{array}{lllll}
P1 & min(X,Y,Z) \Leftrightarrow & X \leq Y & | & Z = X. \\
 & min(X,Y,Z) \Leftrightarrow & X > Y & | & Z = Y.
\end{array}
$$

$$
\begin{array}{lllll}
P2 & min(X,Y,Z) \Leftrightarrow & X < Y & | & Z = X. \\
 & min(X,Y,Z) \Leftrightarrow & X \geq Y & | & Z = Y.
\end{array}
$$

$$
\begin{tikzcd}
min(X,Y,Z) \wedge X \leq Y \arrow[d, "P_1"] & min(X,Y,Z) \wedge X \leq Y \arrow[d, "P_2"] \\
Z = X \wedge X \leq Y &
\end{tikzcd}
$$

Operational Equivalence

Given a goal and two programs, computations in both programs leads to the same result.
A decidable, sufficient and necessary condition for operational equivalence of terminating CHR programs through joinability of minimal states.

$$
\begin{array}{lllll}
P1 & min(X,Y,Z) \Leftrightarrow & X{\le}Y & | & Z{=}X . \\
 & min(X,Y,Z) \Leftrightarrow & X{>}Y & | & Z{=}Y .
\end{array}
$$

$$
\begin{array}{lllll}
P2 & min(X,Y,Z) \Leftrightarrow & X{<}Y & | & Z{=}X . \\
 & min(X,Y,Z) \Leftrightarrow & X{\ge}Y & | & Z{=}Y .
\end{array}
$$

$$
\begin{array}{ccc}
min(X,Y,Z) \wedge X{\le}Y & \qquad & min(X,Y,Z) \wedge X{\le}Y \\
\downarrow P_1 & & \downarrow P_2 \\
Z{=}X \wedge X{\le}Y & &
\end{array}
$$

Operational Equivalence

Given a goal and two programs, computations in both programs leads to the same result.
A decidable, sufficient and necessary condition for operational equivalence of terminating CHR programs through joinability of minimal states.

$$
\begin{array}{ll}
P1 & min(X,Y,Z) \Leftrightarrow X \leq Y \mid Z = X. \\
 & min(X,Y,Z) \Leftrightarrow X > Y \mid Z = Y.
\end{array}
$$

$$
\begin{array}{ll}
P2 & min(X,Y,Z) \Leftrightarrow X < Y \mid Z = X. \\
 & min(X,Y,Z) \Leftrightarrow X \geq Y \mid Z = Y.
\end{array}
$$

$$
\begin{array}{c}
min(X,Y,Z) \wedge X \leq Y \\
\downarrow P_1 \\
Z = X \wedge X \leq Y
\end{array}
\qquad\qquad
\begin{array}{c}
min(X,Y,Z) \wedge X \leq Y \\
\downarrow P_2 \\

\end{array}
$$

Part II

Example Programs

Chemical Abstract Machine Style

One constraint. One Simpagation rule.

```
min(N) \ min(M) ⇔ N=<M | true.
```

```
gcd(N) \ gcd(M) ⇔ 0<N,N=<M | gcd(M-N).

fib(N) \ fib(M) ⇔ 0<N,M=<N | fib(M+N).
```

```
prime(I) \ prime(J) ⇔ J mod I = 0 | true.
```

Chemical Abstract Machine Style

One constraint. One Simpagation rule.

```
min(N) \ min(M) <=> N=<M | true.

gcd(N) \ gcd(M) <=> 0<N,N=<M | gcd(M-N).
fib(N) \ fib(M) <=> 0<N,M=<N | fib(M+N).

prime(I) \ prime(J) <=> J mod I = 0 | true.
```

Chemical Abstract Machine Style

One constraint. One Simpagation rule.

```
min(N) \ min(M) ⇔ N=<M | true.
```

```
gcd(N) \ gcd(M) ⇔ 0<N,N=<M | gcd(M-N).
```

```
fib(N) \ fib(M) ⇔ 0<N,M=<N | fib(M+N).
```

```
prime(I) \ prime(J) ⇔ J mod I = 0 | true.
```

Chemical Abstract Machine Style

One constraint. One Simpagation rule.

```
min(N) \ min(M) ⇔ N=<M | true.

gcd(N) \ gcd(M) ⇔ 0<N,N=<M | gcd(M-N).

fib(N) \ fib(M) ⇔ 0<N,M=<N | fib(M+N).

prime(I) \ prime(J) ⇔ J mod I = 0 | true.
```

Computational Logic Programming

`fib(N,M)` is true if `M` is the `N`th Fibonacci number.

Top-down Goal-Driven Evaluation

$$\texttt{fib(0,M)} \Leftrightarrow \texttt{M = 1.}$$
$$\texttt{fib(1,M)} \Leftrightarrow \texttt{M = 1.}$$
$$\texttt{fib(N,M)} \Leftrightarrow \texttt{N}\geq\texttt{2 | fib(N-1,M1)} \wedge \texttt{fib(N-2,M2)} \wedge \texttt{M = M1 + M2.}$$

Computational Logic Programming

`fib(N,M)` is true if `M` is the `N`th Fibonacci number.

Top-down Goal-Driven Evaluation with Tabling (Memoisation)

```
fib(N,M1) ∧ fib(N,M2)  ⇔  M1 = M2 ∧ fib(N,M1).

fib(0,M) ⇒ M = 1.
fib(1,M) ⇒ M = 1.
fib(N,M) ⇒ N≥2 | fib(N-1,M1) ∧ fib(N-2,M2) ∧ M = M1 + M2.
```

Computational Logic Programming

`fib(N,M)` is true if `M` is the `N`th Fibonacci number.

Bottom-up Data-Driven Evaluation

```
fib ⇔ fib(0,1) ∧ fib(1,1).
fib(N1,M1) ∧ fib(N2,M2) ⇒ N1=N2+1 |
                      N=N1+1 ∧ M=M1+M2 ∧ fib(N,M).
```

Computational Logic Programming

`fib(N,M)` is true if `M` is the `N`th Fibonacci number.

Bottom-up Data-Driven Evaluation with Termination

```
fib(Max) ⇒ fib(0,1) ∧ fib(1,1).
fib(Max) ∧ fib(N1,M1) ∧ fib(N2,M2) ⇒ Max>N1 ∧ N1=N2+1 |
                          N=N1+1 ∧ M=M1+M2 ∧ fib(N,M).
```

Computational Logic Programming

`fib(N,M)` is true if `M` is the `N`th Fibonacci number.

Bottom-up Data-Driven Evaluation, Two Results Only

```
fib(Max) ⇒ fib(0,1) ∧ fib(1,1).
fib(Max) ∧ fib(N1,M1) \ fib(N2,M2) ⇒ Max>N1 ∧ N1=N2+1 |
                         N=N1+1 ∧ M=M1+M2 ∧ fib(N,M).
```

Paths in a Graph

$$\begin{array}{rcl} e(X,Y) & \Rightarrow & p(X,Y). \\ e(X,Z) \wedge p(Z,Y) & \Rightarrow & p(X,Y). \end{array}$$

$$e(a,b) \wedge e(b,c) \wedge e(c,d)$$
$$\downarrow\downarrow$$
$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d)$$
$$\downarrow\downarrow$$
$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d)$$
$$\downarrow\downarrow$$
$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d) \wedge p(a,d)$$

Paths in a Graph

$$\begin{array}{rcl} e(X,Y) & \Rightarrow & p(X,Y). \\ e(X,Z) \wedge p(Z,Y) & \Rightarrow & p(X,Y). \end{array}$$

$$e(a,b) \wedge e(b,c) \wedge e(c,d)$$
$$\downarrow\downarrow$$
$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d)$$
$$\downarrow\downarrow$$
$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d)$$
$$\downarrow\downarrow$$
$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d) \wedge p(a,d)$$

Paths in a Graph

$$
\begin{array}{rcl}
e(X,Y) & \Rightarrow & p(X,Y). \\
e(X,Z) \wedge p(Z,Y) & \Rightarrow & p(X,Y).
\end{array}
$$

$$
\begin{array}{c}
e(a,b) \wedge e(b,c) \wedge e(c,d) \\
\downarrow\downarrow \\
e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \\
\downarrow\downarrow \\
e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d) \\
\downarrow\downarrow \\
e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d) \wedge p(a,d)
\end{array}
$$

Paths in a Graph

$$
\begin{array}{rcl}
e(X,Y) & \Rightarrow & p(X,Y). \\
e(X,Z) \wedge p(Z,Y) & \Rightarrow & p(X,Y).
\end{array}
$$

$$
\begin{array}{c}
e(a,b) \wedge e(b,c) \wedge e(c,d) \\
\downarrow\downarrow \\
e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \\
\downarrow\downarrow \\
e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d) \\
\downarrow\downarrow \\
e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d) \wedge p(a,d)
\end{array}
$$

Paths in a Graph

$$
\begin{aligned}
e(X,Y) \quad &\Rightarrow \quad p(X,Y).\\
e(X,Z) \wedge p(Z,Y) \quad &\Rightarrow \quad p(X,Y).
\end{aligned}
$$

$$e(a,b) \wedge e(b,c) \wedge e(c,d)$$
$$\downarrow\downarrow$$
$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d)$$
$$\downarrow\downarrow$$
$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d)$$
$$\downarrow\downarrow$$
$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d) \wedge p(a,d)$$

Paths in a Graph

$$
\begin{array}{rcl}
e(X,Y) & \Rightarrow & p(X,Y). \\
e(X,Z) \wedge p(Z,Y) & \Rightarrow & p(X,Y).
\end{array}
$$

$$e(a,b) \wedge e(b,c) \wedge e(c,d)$$

$$\downarrow\downarrow$$

$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d)$$

$$\downarrow\downarrow$$

$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d)$$

$$\downarrow\downarrow$$

$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d) \wedge p(a,d)$$

Paths in a Graph

$$\begin{array}{rcl} e(X,Y) & \Rightarrow & p(X,Y). \\ e(X,Z) \wedge p(Z,Y) & \Rightarrow & p(X,Y). \end{array}$$

$$e(a,b) \wedge e(b,c) \wedge e(c,d)$$
$$\downarrow\downarrow$$
$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d)$$
$$\downarrow\downarrow$$
$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d)$$
$$\downarrow\downarrow$$
$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d) \wedge p(a,d)$$

Paths in a Graph

$$\begin{array}{rcl} e(X,Y) & \Rightarrow & p(X,Y). \\ e(X,Z) \wedge p(Z,Y) & \Rightarrow & p(X,Y). \end{array}$$

$$\begin{array}{c} e(a,b) \wedge e(b,c) \wedge e(c,d) \\ \downarrow\downarrow \\ e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \\ \downarrow\downarrow \\ e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d) \\ \downarrow\downarrow \\ e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b) \wedge p(b,c) \wedge p(c,d) \wedge p(a,c) \wedge p(b,d) \wedge p(a,d) \end{array}$$

Shortest Paths in a Graph

$$
\begin{array}{rcl}
p(X,Y,N) \setminus p(X,Y,M) & \Leftrightarrow & N \leq M \mid true. \\
e(X,Y) & \Rightarrow & p(X,Y,1). \\
e(X,Z) \wedge p(Z,Y,N) & \Rightarrow & p(X,Y,N+1).
\end{array}
$$

$$e(a,b) \wedge e(b,c) \wedge e(c,d)$$

$$\downarrow\downarrow$$

$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b,1) \wedge p(b,c,1) \wedge p(c,d,1)$$

Shortest Paths in a Graph

$$
\begin{array}{rcl}
p(X,Y,N) \setminus p(X,Y,M) & \Leftrightarrow & N \leq M \mid true. \\
e(X,Y) & \Rightarrow & p(X,Y,1). \\
e(X,Z) \wedge p(Z,Y,N) & \Rightarrow & p(X,Y,N+1).
\end{array}
$$

$$e(a,b) \wedge e(b,c) \wedge e(c,d)$$

$$\downarrow\downarrow$$

$$e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b,1) \wedge p(b,c,1) \wedge p(c,d,1)$$

Shortest Paths in a Graph

$$
\begin{array}{rcl}
p(X,Y,N) \setminus p(X,Y,M) & \Leftrightarrow & N \leq M \mid true. \\
e(X,Y) & \Rightarrow & p(X,Y,1). \\
e(X,Z) \wedge p(Z,Y,N) & \Rightarrow & p(X,Y,N+1).
\end{array}
$$

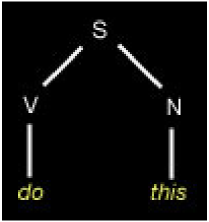
$$
\begin{array}{c}
e(a,b) \wedge e(b,c) \wedge e(c,d) \\
\downarrow\downarrow \\
e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b,1) \wedge p(b,c,1) \wedge p(c,d,1)
\end{array}
$$

Shortest Paths in a Graph

$$
\begin{array}{rcl}
p(X,Y,N) \setminus p(X,Y,M) & \Leftrightarrow & N{\leq}M \mid \mathit{true}. \\
e(X,Y) & \Rightarrow & p(X,Y,1). \\
e(X,Z) \wedge p(Z,Y,N) & \Rightarrow & p(X,Y,N{+}1).
\end{array}
$$

$$
\begin{array}{c}
e(a,b) \wedge e(b,c) \wedge e(c,d) \\
\downarrow\downarrow \\
e(a,b) \wedge e(b,c) \wedge e(c,d) \wedge p(a,b,1) \wedge p(b,c,1) \wedge p(c,d,1)
\end{array}
$$

Dynamic Programming: Parsing



The Cocke-Younger-Kasami (CYK) Algorithm
for grammars in *Chomsky normal form*:
Grammar rules = `A->T` or `A->B*C`,
`A, B, C` nonterminal, `T` terminal symbol.

Word `w` = graph chain of terminal symbols.
Parse `p`= restricted transitive closure over word.

```
terminal    @ A->T, w(T,I,J) ==> p(T,I,J).
nonterminal @ A->B*C, p(B,I,J), p(C,J,K) ==> p(A,I,K).
```

Sorting

One-rule sort related to merge sort and tree sort.
Query Arc `X->Ai` for each unique value `Ai`, `X` only on left of arc.
Answer Ordered chain of arcs `X->A1, A1->A2,...`

```
sort @ X->A \ X->B <=> A<B | A->B.
```

Query `0->2, 0->5, 0->1, 0->7.`
Answer `0->1, 1->2, 2->5, 5->7.`

Complexity: Given n values/arcs.
Each value can move $O(n)$ times to the left.
Quadratic worst-case time complexity.

Sorting

One-rule sort related to merge sort and tree sort.
Arc `0=>Ai` for each unique value `Ai`, left side is level (log of chain length).

```
sort @ X->A \ X->B <=> A<B | A->B.

level@ N=>A , N=>B <=> A<B | N+1=>A, A->B.
```

Query `0=>2, 0=>5, 0=>1, 0=>7.`
Answer `2=>1, 1->2, 2->5, 5->7.`

Complexity: Optimal *log-linear* worst-case time complexity.

Example Rule Generation

Intensional definition of minimum:

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

Derived constraint handling rules:

$$
\begin{array}{lcll}
min(A,B,C) \Rightarrow & & & C{\leq}A \wedge C{\leq}B. \\
min(A,B,C) \Leftrightarrow & C{\neq}B & | & C{=}A. \\
min(A,B,C) \Leftrightarrow & C{\neq}A & | & C{=}B. \\
min(A,B,C) \Leftrightarrow & B{\leq}A & | & C{=}B. \\
min(A,B,C) \Leftrightarrow & A{\leq}B & | & C{=}A.
\end{array}
$$

[Abdennadher,Rigotti, TPLP 2005]   [Apt,Brand,Monfroy]

Linear Polynomial Equations

Equations of the form $a_1x_1 + \ldots + a_nx_n + b = 0$.
Solved form: leftmost variable occurs only once.
Reach solved normal form by Gaussian-style **variable elimination**.

```
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).
```

Fourier's Algorithm

```
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).
```

Combination of Gauss' and Fouriers Algorithms

Gaussian Elimination for $=$

A1*X+P1=0 ∧ XP=0 ⇔
  find(A2*X,XP,P2) |
  compute(P2-(P1/A1)*A2,P3) ∧ A1*X+P1=0 ∧ P3=0.

Fouriers Algorithm for $\geq$

A1*X+P1≥0 ∧ XP≥0 ⇒
  find(A2*X,XP,P2) ∧ opposite_sign(A1,A2) |
  compute(P2-(P1/A1)*A2,P3) ∧ P3≥0.

Bridge Rule for $=$ and $\geq$

A1*X+P1=0 ∧ XP≥0 ⇔
  find(A2*X,XP,P2) |
  compute(P2-(P1/A1)*A2,P3) ∧ A1*X+P1=0 ∧ P3≥0.

Combination of Gauss' and Fouriers Algorithms

Gaussian Elimination for =

```
A1*X+P1=0 ∧ XP=0  ⇔
   find(A2*X,XP,P2) |
   compute(P2-(P1/A1)*A2,P3) ∧  A1*X+P1=0 ∧ P3=0.
```

Fouriers Algorithm for ≥

```
A1*X+P1≥0 ∧ XP≥0  ⇒
   find(A2*X,XP,P2) ∧ opposite_sign(A1,A2) |
   compute(P2-(P1/A1)*A2,P3) ∧  P3≥0.
```

Bridge Rule for = and ≥

```
A1*X+P1=0 ∧ XP≥0  ⇔
   find(A2*X,XP,P2) |
   compute(P2-(P1/A1)*A2,P3) ∧  A1*X+P1=0 ∧ P3≥0.
```

Combination of Gauss' and Fouriers Algorithms

Gaussian Elimination for $=$

```
A1*X+P1=0 ∧ XP=0  ⇔
   find(A2*X,XP,P2) |
   compute(P2-(P1/A1)*A2,P3) ∧  A1*X+P1=0 ∧ P3=0.
```

Fouriers Algorithm for $\geq$

```
A1*X+P1≥0 ∧ XP≥0  ⇒
   find(A2*X,XP,P2) ∧ opposite_sign(A1,A2) |
   compute(P2-(P1/A1)*A2,P3) ∧  P3≥0.
```

Bridge Rule for $=$ and $\geq$

```
A1*X+P1=0 ∧ XP≥0  ⇔
   find(A2*X,XP,P2) |
   compute(P2-(P1/A1)*A2,P3) ∧  A1*X+P1=0 ∧ P3≥0.
```

Syntactic Unification

Rational tree (in)finite tree with finite set of subtrees, e.g. $X = f(X)$.
Solved normal form $X_1{=}t_1 \wedge \ldots \wedge X_n{=}t_n$ $(n \geq 0)$,
where X_i is different to X_j and t_j for all $i \leq j$.

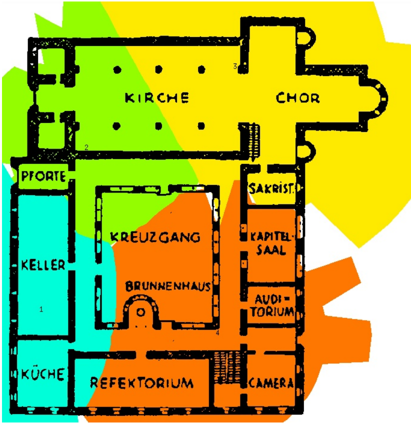
```
reflexivity    @   X=X  ⇔  isvar(X) |  true.
orientation    @   T=X  ⇔  isvar(X) ∧  X≺T |  X=T.
decomposition  @  T1=T2 ⇔   notvar(T1) ∧  notvar(T2) |
                    same_functor(T1,T2) ∧
                    same_args(T1,T2).
confrontation  @  X=T1 ∧  X=T2  ⇔   isvar(X)∧ X≺T1∧ T1⪯T2 |
                    X=T1 ∧  T1=T2.
```

Direct implementation of *Clark's equality theory*.
Quadratic time complexity possible.

Part III

Applications

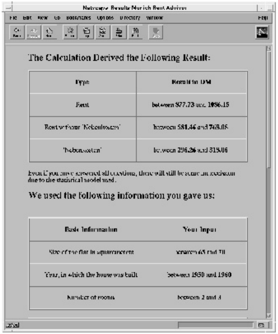
POPULAR - Planning Cordless Communication



T. Frühwirth, P. Brisset
Optimal Placement of Base Stations in Wireless Indoor Communication Networks, IEEE Intelligent Systems Magazine 15(1), 2000.

Voted Among Most Innovative Telecom Applications of the Year by IEEE Expert Magazine, Winner of CP98 Telecom Application Award.

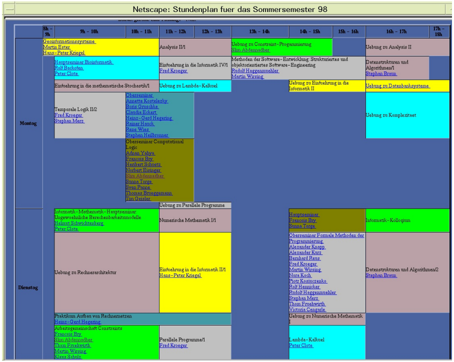
MRA - The Munich Rent Advisor



T. Frühwirth,
S. Abdennadher
The Munich Rent Advisor,
Journal of Theory and
Practice of Logic
Programming, 2000.

Most Popular
Constraint-Based Internet
Application.

University Course Timetabling



S. Abdennadher, M. Saft, S. Will
Classroom Assignment using Constraint Logic Programming, PACLP 2000.

Operational at University of Munich. Room-Allocation for 1000 Lectures a Week.

Reasoning Services

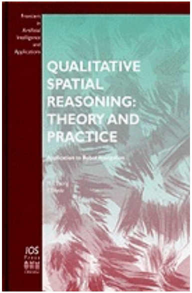
Constraint Abduction, M. Sulzmann, J. Wazny, P. J. Stuckey, CHR 2005.

...System for Generation and Confirmation of Hypotheses,
Alberti, Chesani, Gavanelli, Lamma, W(C)LP 2005.
Interpreting Abduction in CLP, M. Gavanelli et. al., AGP'03.

HYPROLOG:...Assumptions and Abduction,
H. Christiansen, V. Dahl, LNCS 3668, ICLP 2005.
An Experimental CLP Platform for Integrity Constraints and Abduction,
S. Abdennadher, H. Christiansen, FQAS2000, LNCS.
CHR$^\vee$: A Flexible Query Language,
S. Abdennadher, H. Schütz, FQAS'98, LNCS.

- DemoII: Meta-Logic Programming System, Henning Christiansen.
- Terminological Logic Decision Algorithm, Liviu Badea, Bucharest, Romania.
- Description Logic Constraint System, Philip Hanschke, DFKI Kaiserslautern.
- Ordered Resolution Theorem Prover, A. Frisch, Univ. of York, UK.
- PROTEIN+ Theorem Prover, F.Stolzenburg, P. Baumgartner, Univ. Koblenz.

Spatio-Temporal Reasoning



M. T. Escrig, F. Toledo,
Universidad Jaume I, Castellun, Spain.
Qualitative Spatial Reasoning: Theory and Practice, Application to Robot Navigation, IOS Press, 1998.
Qualitative Spatial Reasoning on 3D Orientation Point Objects, QR2002.
Integrates orientation, distance, cardinal directions over points as well as extended objects.

- Spatio-Temporal Annotated CLP - A. Raffaeta, Univ. Venice.
- Diagrammatic Reasoning - B. Meyer, Monash Melbourne.
- RCC Reasoning - B. Bennet, A.G. Cohn, Leeds UK.
- PMON logic for dynamical temporal systems - E. Sandewall, Linkoeping Univ.
- GRF Temporal Reasoning - G. Dondossola, E. Ratto, CISE Milano.

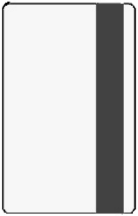
Types and Security

Chameleon Project, Martin Sulzmann, Peter J. Stuckey.

A Theory of Overloading, ACM TOPLAS, 2005.
Improving type error diagnosis, Haskell'04, ACM.
Sound and Decidable Type Inference for Functional Dependencies, ESOP'04, LNCS 2968.
Enforcing Security Policies using Overloading Resolution, Melbourne TR 2001.

- Constraint-Based Polymorphic Type Inference for Functional and Logic Programs, T. Schrijvers, M. Bruynooghe, IFL 2005.
- TypeTool - A Type Inference Visualization Tool, S. Alves, M. Florido, WF(C)LP 2004; Type Inference with CHR, WF(C)LP 2001.
- Subtyping Constraints in Quasi-lattices, E. Coquery, F. Fages, LNCS 2914, 2003; TCLP tool for Type Checking; Type System for CHR, CHR 2005.
- Typed Interfaces to Compose CHR Programs, G. Ringwelski, H. Schlenker.

Testing and Verification



Model Based Testing for Real:
The Inhouse Card Case Study,
A. Pretschner, O. Slotosch, E. Aiglstorfer, S. Kriebel, TU Munich,
Journal on Software Tools for Technology Transfer (STTT) 5:2-3, Springer 2004.

- Automatic Generation of Test Data - J. Harm, University Rostock, Germany.
- Executable Z-Specifications - P. Stuckey, Ph. Dart, University Melbourne.

Agents and Actions

FLUX: A Logic Programming Method for Reasoning Agents,
Michael Thielscher, TPLP CHR Special Issue 2005.
Fluent Calculus, Reasoning about Actions, Robotics.

Specification and Verification of Agent Interaction...
Alberti, Chesani, Gavanelli, Lamma, Mello, Torroni, ACM SAC 2004.
Social integrity constraints on agent behaviour.

- Multi Agent Systems Using Constrains Handling Rules, IC-AI 2002 - B. Bauer, M. Berger, Siemens Munich, Germany - S. Hainzer, Uni Linz, Austria.
- PMON logic for dynamical temporal systems with actions and change - M. Bjgareland, E. Sandewall, Linkoeping University, Sweden.

Semantic Web

The COntext INterchange *Project*

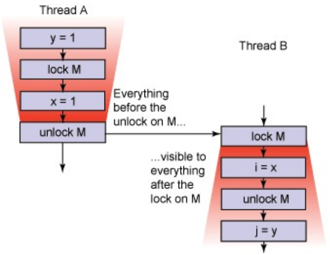
COIN Context Interchange Project,
Stuart E. Madnick, MIT Cambridge.
Reasoning About Temporal Context Using Ontology and Abductive CLP,
PPSWR 2004 LNCS 3208.

Semantic Web Reasoning for Ontology-Based Integration of Resources,
Liviu Badea, Doina Tilivea and Anca Hotaran, PPSWR 2004 LNCS 3208.

- S. Bressan, C.H. Goh, S. Madnick, M. Siegel et. al.
Context Knowledge Representation and Reasoning in the Context Interchange System, Applied Intelligence, Vol 13:2, 2000;
Context Interchange...for the intelligent integration of information, ACM Transactions on Information Systems, 1999.

Java Memory Machine

JMM by Vijay Saraswat, IBM TJ Watson Research and Penn State Univ.
Implementation JMMSolve by Tom Schrijvers, K.U. Leuven, Belgium



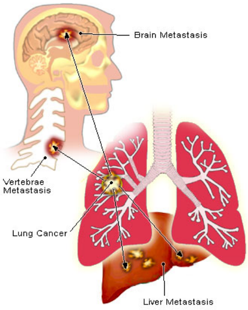
Conditional Read

```
Xr = (Cond)?Xw1:Xi

ite(true,Xr,Xw1,Xi)  <=> Xr = Xw1.
ite(false,Xr,Xw1,Xi) <=> Xr = Xi.
ite(Cond,Xr,X,X)     <=> Xr = X.
```

Lung Cancer Diagnosis

Veronica Dahl, Simon Fraser University, Vancouver, Canada.
Lung cancer is leading cause of cancer death, very low survival rate.
Use bio-markers indicating gene mutations to diagnose lung cancer.



Lung Cancer and Metastasis

Concept Formation Rules (CFR) in CHR.
Retractable constraints.

```
age(X,A),history(X,smoker),
serum_data(X,marker_type) <=>
marker(X,marker_type,P,B),
probability(P,X,B) |
possible_lung_cancer(yes,X).
```

Multimedia Transformation Engine for Web Presentations

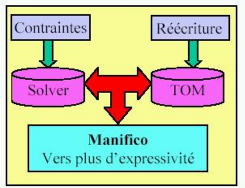
Joost Geurts, University of Amsterdam.
Automatic generation of interactive, time-based and media centric WWW presentations from semi-structured multimedia databases.

Powered by Cuypers

Business Rules for Optimization

MANIFICO - Francois Fages, Claude Kirchner, Hassan Ait-Kaci,...France

Business Rule: defines or constrains behavior or structure of business.
"A car must be available to be assigned to a rental agreement".



DERBY EU Car Rent Case in CHR, O. Bouissou.

```
reservation(Renter,Group,From,To),
available(car(Id,Group,...),From) <=>...
        rentagreement(Renter,Id,From,To).
```

Finally...

Google "Constraint Handling Rules" for the CHR website

驰

Finally...

Google "Constraint Handling Rules" for the CHR website

驰

Transcribed as **CHR**, means

Finally...

Google "Constraint Handling Rules" for the CHR website

驰

Transcribed as **CHR**, means
to speed, to propagate, to be famous

Summary Constraint Handling Rules (CHR)

Essential pure declarative relational language

- Constraint programming language for Computational Logic
- Multi-headed guarded committed-choice rules
  transform multi-set of constraints until exhaustion
- Ideal for concise executable specifications and rapid prototyping
- Any algorithm implementable with optimal time+space complexity
- Any-time (approximation), on-line (incrementality), concurrent algorithms for free.
- Logical and operational semantics coincide strongly
- High-level supports program analysis and transformation:
  Confluence/completion, termination/time complexity, correctness...
- Language extension: Implemenations in most Prologs, Java, Haskell
- 100s of applications from types, time tabling to cancer diagnosis

Conclusions

CHR - From computational logic to logical computations.

High-level abstract approach
Pros: conciseness, properties, analysis...
Cons: Learning, constant time factor overhead.
Try it yourself and find out!

Active research area, many topics, open-ended...

- implementation: CHR in Java,
- environment: confluence checker, debugging,
- analysis: termination and complexity,
- automatic rule generation,
- classical algorithms revisited,
- semantics: linear logic.
- application: software engineering UML.

Conclusions

CHR - From computational logic to logical computations.

High-level abstract approach
Pros: conciseness, properties, analysis...
Cons: Learning, constant time factor overhead.
Try it yourself and find out!

Mailing List CHR@LISTSERV.CC.KULEUVEN.AC.BE
Constraint Handling Rules discussion and announcements

`http://www.cs.kuleuven.ac.be/~dtai/projects/CHR/`
Download. News. Examples. Top Authors. Research Topics. Projects. Applications. 600 Papers. WebCHR Online.

TPLP journal special issue on CHR, vol. 4+5, September 2005.

CHR Presentations at Sitges Conferences 2005

SAT Oct. 1

BeyondFD	16:05	A Constraint Solver for Sequences, N. Kosmatov.

SUN Oct. 2

CP	14:05	CHR Tutorial, T. Frühwirth.
ICLP	17:00	Hyprolog, H. Christiansen.

MON Oct. 3

ICLP	14:45	Guard Optimization, J. Sneyer et. al.
ICLP	14:45	Parallel Union-Find, T. Frühwirth.

TUE Oct. 4

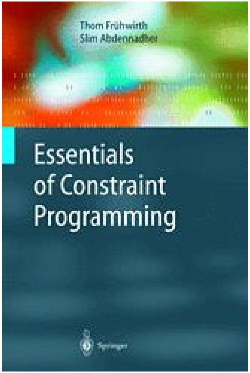
CP	10:30	Linear Logic Semantics, H. Betz.
CP		Implication/Universal Quantification Constr., M. Thielscher.

WED, Oct. 5

CHR 2005	9:00	Full-day workshop.
CSLP'05	11.45	Extracting Selected Phrases..., V. Dahl, Ph. Blache.
WCB'05	16:40	RNA Secondary Structure Design, M. Bavarian, V. Dahl.

References

Google "constraint handling rules"



Essentials of Constraint Programming
Thom Frühwirth,
Slim Abdennadher
Springer, 2003.

Constraint-Programmierung
Lehrbuch
Thom Frühwirth,
Slim Abdennadher
Springer, 1997.

Lexicographic Order Constraint Solver

```
[] lex [] <=> true.
[X|L1] lex [Y|L2] <=> X<Y | true.

[X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.

[X|L1] lex [Y|L2] ==> X=<Y.

[X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
[X,U|L1] lex [Y,V|L2] <=> U>=V, L1=[_|_] |
                          [X,U] lex [Y,V], [X|L1] lex [Y|L2].
```

Executable specification: short, concise
using recursive decomposition and propagation

Lexicographic Order Constraint Solver

```
[] lex [] <=> true.
[X|L1] lex [Y|L2] <=> X<Y | true.

[X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.

[X|L1] lex [Y|L2] ==> X=<Y.

[X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
[X,U|L1] lex [Y,V|L2] <=> U>=V, L1=[_|_] |
                          [X,U] lex [Y,V], [X|L1] lex [Y|L2].
```

Incremental and concurrent: by nature of CHR
Efficient: Optimal linear worst-case time complexity

Lexicographic Order Constraint Solver

```
[] lex [] <=> true.
[X|L1] lex [Y|L2] <=> X<Y | true.

[X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.

[X|L1] lex [Y|L2] ==> X=<Y.

[X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
[X,U|L1] lex [Y,V|L2] <=> U>=V, L1=[_|_] |
                        [X,U] lex [Y,V], [X|L1] lex [Y|L2].
```

Independent of underlying constraint system
Complete: propagates as much as possible

Lexicographic Order Constraint Solver

```
[] lex [] <=> true.
[X|L1] lex [Y|L2] <=> X<Y | true.

[X|L1] lex [Y|L2] <=> X=Y | L1 lex L2.

[X|L1] lex [Y|L2] ==> X=<Y.

[X,U|L1] lex [Y,V|L2] <=> U>V | X<Y.
[X,U|L1] lex [Y,V|L2] <=> U>=V, L1=[_|_] |
                              [X,U] lex [Y,V], [X|L1] lex [Y|L2].
```

Confluence: proven by CHR confluence checker
Correctness: logical reading consequence of specification

Basic Union-Find

[Schrijvers,Frühwirth, TPLP Programming Pearl 2006]

```
make     @ make(X) <=> root(X).
union    @ union(X,Y) <=> find(X,A), find(Y,B), link(A,B).

findNode @ X -> PX \ find(X,R) <=> find(PX,R).
findRoot @ root(X) \ find(X,R) <=> R=X.

linkEq   @ link(X,X) <=> true.
link     @ link(X,Y), root(X), root(Y) <=> Y -> X, root(X).
```

Optimal Union-Find

[Schrijvers,Frühwirth, TPLP Programming Pearl 2006]

```
make      @ make(X) <=> root(X,0).
union     @ union(X,Y) <=> find(X,A), find(Y,B), link(A,B).

findNode @ X -> PX , find(X,R) <=> find(PX,R), X -> R.
findRoot @ root(X) \ find(X,R) <=> R=X.

linkEq    @ link(X,X) <=> true.
linkLeft @ link(X,Y), root(X,RX), root(Y,RY) <=> RX >= RY |
              Y -> X, root(X,max(RX,RY+1)).
linkRight@ link(X,Y), root(Y,RY), root(X,RX) <=> RY >= RX |
              X -> Y, root(Y,max(RY,RX+1)).
```