Constraint Handling Rules (CHR)

Projects, People, Papers about Programming with CHR

Thom Frühwirth

Faculty of Computer Science
University of Ulm, Germany
www.informatik.uni-ulm.de/pm/mitarbeiter/fruehwirth/

CP 2005 Tutorial, Sitges, October 2005

Citations may be incomplete for space reasons

Overview

CHR...

1. Constraint Handling Rules (CHR)
2. Program Analysis
3. Constraint Solvers

...Around the World

4. Language Issues
5. From Computational Logic to Logical Computations
6. Classical Applications
7. Trends in Applications

Part I

CHR...

Constraint Handling Rules (CHR)

Essential 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, operational equivalence, termination/time complexity, correctness...
- Language extension: Implemenations in most Prologs, Java, Haskell
- 100s of applications from types, time tabling to cancer diagnosis

Constraint Handling Rules (CHR)

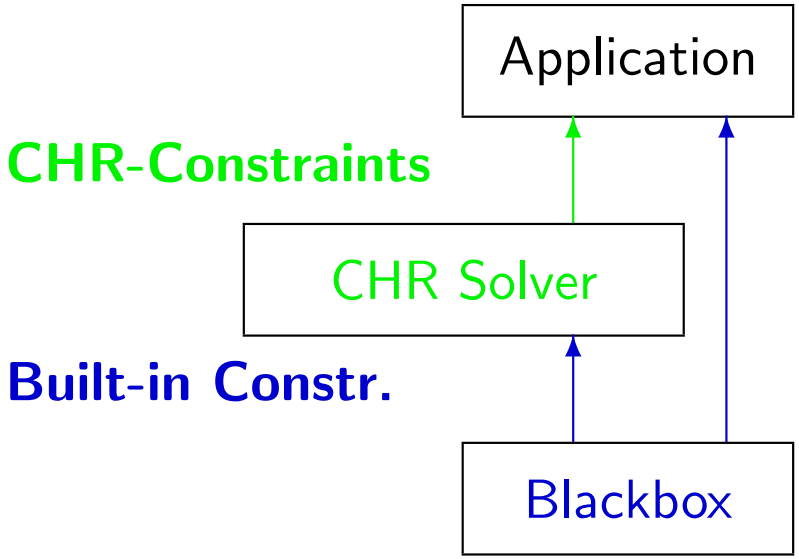
Concurrent committed-choice guarded rules with ask and tell constraints for computational logic and more... (100+ applications)

- Computational Logic
  - combining forward and backward chaining
  - combining deduction and abduction with constraints
  - bottom-up evaluation with integrity constraints
  - top-down evaluation with tabulation
  - automated theorem proving with constraints
  - *simplification and propagation of constraints*
- production rule systems
- multi-set rewriting and transformation
- event-condition-action (ECA) rules

15+ Implementations: Prolog, Java, Haskell,...
Extensions: Disjunction/Search, Dynamic and Soft Constraints, Probabilistic Rules, Program Transformation, Literate Programming.

Constraint Handling Rules (CHR)

Concurrent committed-choice guarded rules with ask and tell constraints for computational logic and more... (100+ applications)



15+ Implementations: Prolog, Java, Haskell,...
Extensions: Disjunction/Search, Dynamic and Soft Constraints, Probabilistic Rules, Program Transformation, Literate Programming.

Roots

Influences Around 1990:

- (Concurrent) constraint logic programming:
M. Maher, E. Shapiro, V. Saraswat, P. V. Hentenryck,... - M. Dincbas's, P.V. Hentenryck's,...Demons and forward rules of CHIP
- J.-M. Andreoli's, R. Pareschi's Logical Objects (LO)
- G. Berry's, G. Boudol's Chemical Abstract Machine (ChAM)
- Term Rewriting Systems Concepts (Leler's BERTRAND, Goguen's OBJ)
- Event-condition-action (ECA) rules of Active Databases, Deductive DB
- Production rule systems like OPS5

1991 Frühwirth's CHR: Inference Rules for Constraint Logic Computation

Related but independent:

- Smolka's Guarded-Rules and OZ
- J.-P. Banatre's, D. Le Metayer's Multi-set Transformation GAMMA
- Meseguer's Rewriting Logic (basis of MAUDE and Kirchner's ELAN)

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Syntax and Declarative Semantics

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

- H: non-empty conjunction of CHR constraints
- C: conjunction of built-in constraints
- B: conjunction of CHR and built-in constraints

Constraint Theory CT for Built-In Constraints

Operational Semantics

Apply rules until exhaustion in any order (fixpoint computation).

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 Prolog, CHR constraints evaluated depth-first from left to right and rules applied top-down in program text order.

Operational Semantics

Apply rules until exhaustion in any order (fixpoint computation).

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 Prolog, CHR constraints evaluated depth-first from left to right and rules applied top-down in program text order.

Operational Semantics

Apply rules until exhaustion in any order (fixpoint computation).

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 Prolog, CHR constraints evaluated depth-first from left to right and rules applied top-down in program text order.

Operational Semantics

Apply rules until exhaustion in any order (fixpoint computation).

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 Prolog, CHR constraints evaluated depth-first from left to right and rules applied top-down in program text order.

Anytime Algorithm

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

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A$$
$$\downarrow \quad \text{(transitivity)}$$
$$A \leq B \wedge B \leq C \wedge \underline{C \leq A} \wedge \underline{A \leq C}$$
$$\downarrow \quad \text{(antisymmetry)}$$
$$A \leq B \wedge B \leq C \wedge \underline{A = C}$$
$$\downarrow \quad \text{(built-in solver)}$$
$$\underline{A \leq B} \wedge \underline{B \leq A} \wedge A = C$$
$$\downarrow \quad \text{(antisymmetry)}$$
$$A = B \wedge A = C$$

Anytime Algorithm

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

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

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

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

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

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

$$\downarrow \quad \text{(built-in solver)}$$

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

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

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

Anytime Algorithm

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

$$\underline{A \leq B} \wedge \underline{B \leq C} \wedge C \leq A$$
$$\downarrow \quad \text{(transitivity)}$$
$$A \leq B \wedge B \leq C \wedge \underline{C \leq A} \wedge \underline{A \leq C}$$
$$\downarrow \quad \text{(antisymmetry)}$$
$$A \leq B \wedge B \leq C \wedge \underline{A = C}$$
$$\downarrow \quad \text{(built-in solver)}$$
$$\underline{A \leq B} \wedge \underline{B \leq A} \wedge A = C$$
$$\downarrow \quad \text{(antisymmetry)}$$
$$A = B \wedge A = C$$

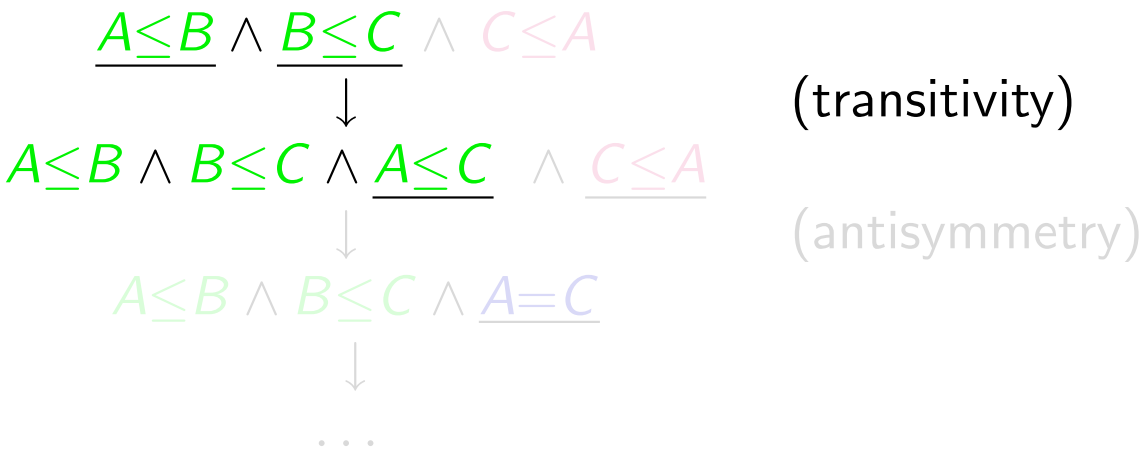
Online Algorithm

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

Monotonicity and Incrementality

If $G \longmapsto G'$
then $G \wedge C \longmapsto G' \wedge C$

Applicable rules cannot become inapplicable in larger context.



Online Algorithm

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

Monotonicity and Incrementality

If $G \longmapsto G'$
then $G \wedge C \longmapsto G' \wedge C$

Applicable rules cannot become inapplicable in larger context.

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

Online Algorithm

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

Monotonicity and Incrementality

If $G \longmapsto G'$
then $G \wedge C \longmapsto G' \wedge C$

Applicable rules cannot become inapplicable in larger context.

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

Online Algorithm

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

Monotonicity and Incrementality

If $G \longmapsto G'$
then $G \wedge C \longmapsto G' \wedge C$

Applicable rules cannot become inapplicable in larger context.

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

Concurrency / Weak Parallelism

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

Rules can be applied in parallel to different parts of a goal.

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

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

$$\downarrow \qquad\qquad\qquad \downarrow$$

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

$$\wedge$$

$$\downarrow$$

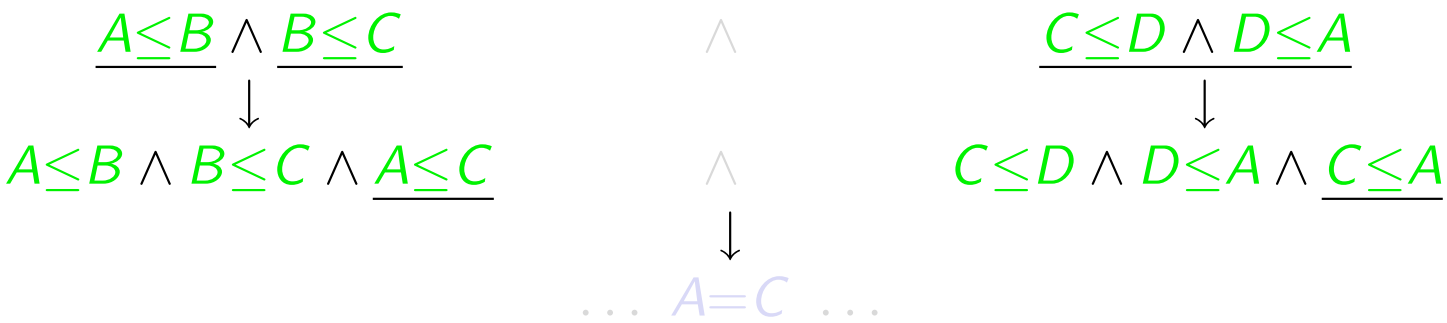
$$\ldots A = C \ldots$$

Concurrency / Weak Parallelism

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

Rules can be applied in parallel to different parts of a goal.

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



Concurrency / Weak Parallelism

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

Rules can be applied in parallel to different parts of a goal.

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

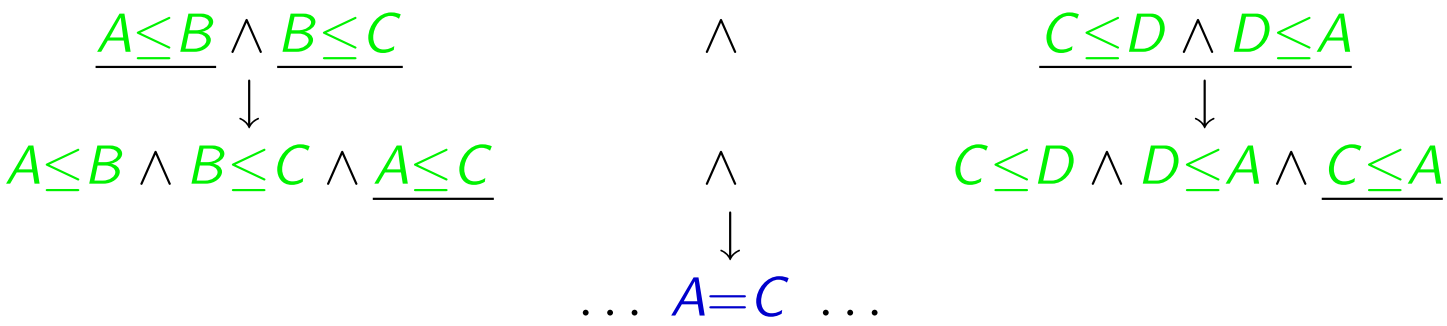
$$\begin{array}{ccccc} \underline{A \leq B} \wedge \underline{B \leq C} & & \wedge & & \underline{C \leq D} \wedge \underline{D \leq A} \\ \downarrow & & & & \downarrow \\ A \leq B \wedge B \leq C \wedge \underline{A \leq C} & & \wedge & & C \leq D \wedge D \leq A \wedge \underline{C \leq A} \\ & & \downarrow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

Concurrency / Weak Parallelism

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

Rules can be applied in parallel to different parts of a goal.

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



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.

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

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.

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

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

Optimal Time and Space Complexity

The Computational Power and Complexity of Constraint Handling Rules, Jon Sneyers, Tom Schrijvers, Bart Demoen. CHR 2005.

Introduces CHR machines, analogous to RAM and Turing machines.
These machines can simulate each other in polynomial time.
$\Rightarrow$ CHR is Turing-complete
$\Rightarrow$ every algorithm can be implemented in CHR with best known time and space complexity.

CHR Program Analysis

Termination

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

Consistency

Logical reading of the rules is consistent. [Constraints Journal 2000]

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

Make non-confluent programs confluent by adding rules. [CP'98]

Operational Equivalence

Do two programs have the same behavior? [CP'99]

Complexity

Determine time complexity from structure of rules. [KR'02]

[Slim Abdennadher and Thom Frühwirth]

Termination

A *ranking* $||$ maps terms into natural numbers.
For all simplification rules

$$H_1 \wedge \ldots \wedge H_n \Leftrightarrow C \mid D \wedge B_1 \wedge \ldots \wedge B_m$$

it holds that

$$C \wedge D \;\rightarrow\; |H_1| + \ldots + |H_n| > |B_1| + \ldots + |B_m|$$

For all propagation rules

$$H_1 \wedge \ldots \wedge H_n \Rightarrow C \mid D \wedge B_1 \wedge \ldots \wedge B_m$$

it holds that

$$C \wedge D \;\rightarrow\; |H_i| > |B_j| \text{ for all } i, j$$

Then the CHR program *terminates* for all queries whose ranking is bounded from above.

[Frühwirth, KR'02]

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

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

$$\begin{array}{lll} A & \mapsto & B \\ A & \mapsto & C \\ \hline B & \mapsto^* & D \\ C & \mapsto^* & D \end{array}$$

$\Rightarrow$ Independence from the order in which constraints processed.
$\Rightarrow$ Consistency of logical reading of the program.
Decidable, sufficient and necessary condition for confluence of terminating CHR programs through joinability of critical pairs.

Critical pair results from applying two rules to an overlap.
Overlap takes both rule heads and guards, shares some CHR constraints.

Confluence

Given a goal, every computation leads to the same result no matter which rules are applied.
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 overlapping minimal states

Confluence

Given a goal, every computation leads to the same result no matter which rules are applied.
A decidable, sufficient and necessary condition for confluence of terminating CHR programs through joinability of critical pairs.

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

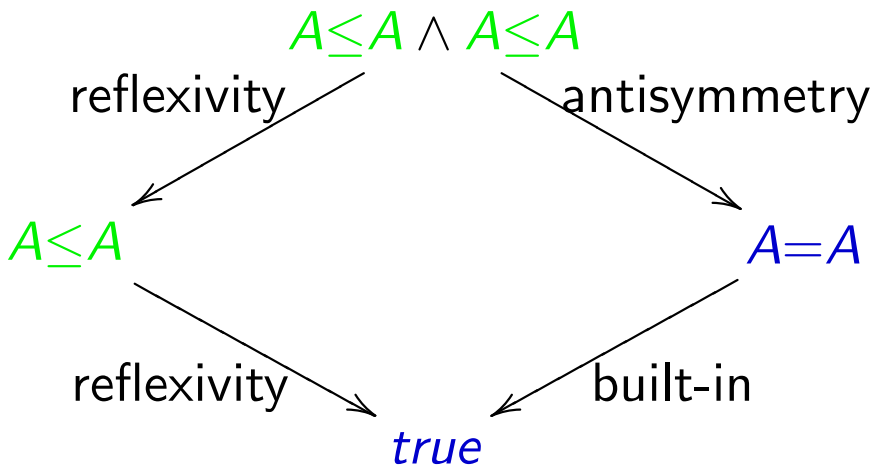
Start from overlapping minimal states

Confluence

Given a goal, every computation leads to the same result no matter which rules are applied.
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 overlapping minimal states



Completion

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

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

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

Completion

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

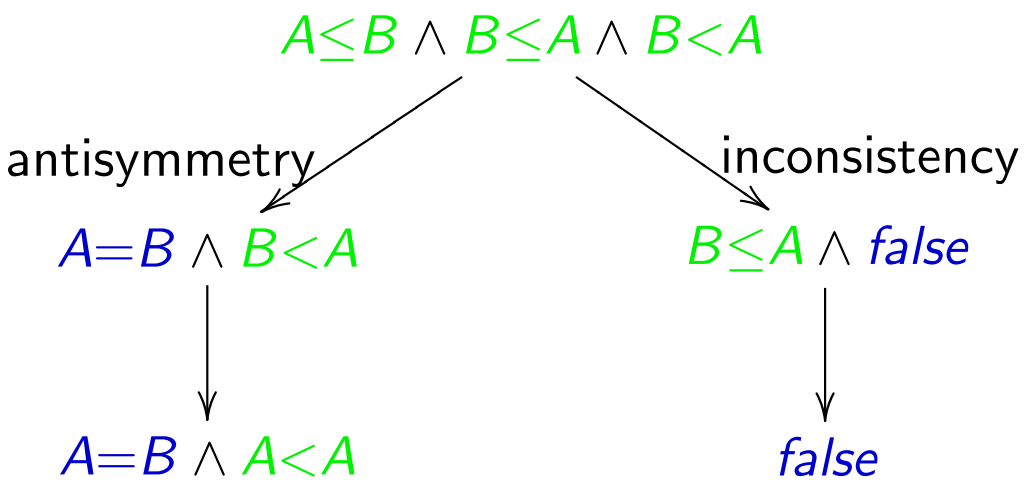
$$X \leq Y \wedge Y \leq X \quad \Leftrightarrow \quad X = Y \qquad \text{(antisymmetry)}$$
$$X \leq Y \wedge Y < X \quad \Leftrightarrow \quad \mathit{false} \qquad \text{(inconsistency)}$$

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

Completion

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

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

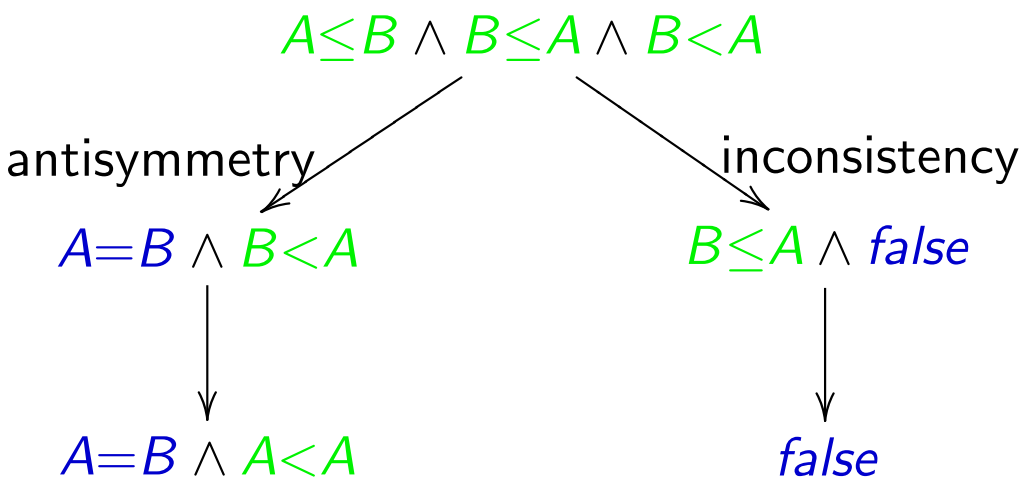


$$X < X \quad \Leftrightarrow \quad \mathit{false} \qquad \text{(irreflexivity)}$$

Completion

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

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



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

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.

$P1$ $max(X,Y,Z) \Leftrightarrow X<Y \mid Z=Y.$
$max(X,Y,Z) \Leftrightarrow X \geq Y \mid Z=X.$

$P2$ $max(X,Y,Z) \Leftrightarrow X \leq Y \mid Z=Y.$
$max(X,Y,Z) \Leftrightarrow X>Y \mid Z=X.$

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.

$P1$ $max(X,Y,Z) \Leftrightarrow X<Y \mid Z=Y.$
$max(X,Y,Z) \Leftrightarrow X \geq Y \mid Z=X.$

$P2$ $max(X,Y,Z) \Leftrightarrow X \leq Y \mid Z=Y.$
$max(X,Y,Z) \Leftrightarrow X>Y \mid Z=X.$

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.

$P1$ $max(X, Y, Z) \Leftrightarrow X<Y \mid Z=Y.$
$max(X, Y, Z) \Leftrightarrow X \geq Y \mid Z=X.$

$P2$ $max(X, Y, Z) \Leftrightarrow X \leq Y \mid Z=Y.$
$max(X, Y, Z) \Leftrightarrow X>Y \mid Z=X.$

$max(X, Y, Z) \wedge X \geq Y$
$\downarrow P_1$
$Z=X \wedge X \geq Y$

$max(X, Y, Z) \wedge X \geq Y$
$\downarrow P_2$

Boolean Constraints

Local consistency algorithm simplifies one atomic Boolean constraint at a time into syntactic equalities.

$$
\begin{aligned}
and(X,X,Z) &\Leftrightarrow X{=}Z.\\
and(X,Y,1) &\Leftrightarrow X{=}1 \wedge Y{=}1.\\
and(X,1,Z) &\Leftrightarrow X{=}Z.\\
and(X,0,Z) &\Leftrightarrow Z{=}0.\\
and(1,Y,Z) &\Leftrightarrow Y{=}Z.\\
and(0,Y,Z) &\Leftrightarrow Z{=}0.
\end{aligned}
$$

$$
\begin{aligned}
imp(0,X) &\Leftrightarrow true.\\
imp(X,0) &\Leftrightarrow X{=}0.\\
imp(1,X) &\Leftrightarrow X{=}1.\\
imp(X,1) &\Leftrightarrow true.
\end{aligned}
$$

Solver Union and Cooperation by Completion

Bridge rules relate constraints from different programs for their cooperation and communication.

$$and(X, Y, X) \Leftrightarrow imp(X, Y).$$

Non-confluent: E.g.

Completion adds the rules:

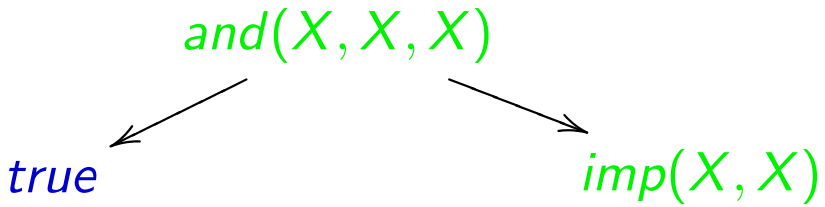
$$imp(X, X) \Leftrightarrow true.$$
$$imp(X, Y) \wedge imp(X, Y) \Leftrightarrow imp(X, Y).$$
$$imp(X, Y) \wedge and(X, Y, Z) \Leftrightarrow imp(X, Y) \wedge X{=}Z.$$

Solver Union and Cooperation by Completion

Bridge rules relate constraints from different programs for their cooperation and communication.

$$and(X, Y, X) \quad \Leftrightarrow \quad imp(X, Y).$$

Non-confluent: E.g.



Completion adds the rules:

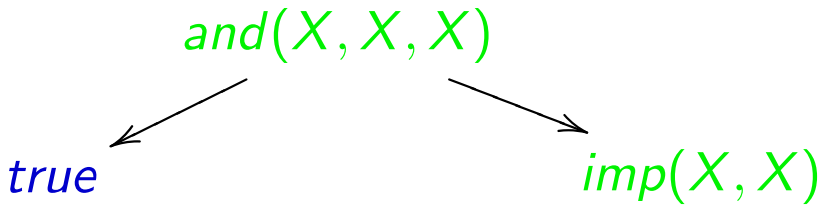
$$imp(X, X) \quad \Leftrightarrow \quad true.$$
$$imp(X, Y) \wedge imp(X, Y) \quad \Leftrightarrow \quad imp(X, Y).$$
$$imp(X, Y) \wedge and(X, Y, Z) \quad \Leftrightarrow \quad imp(X, Y) \wedge X{=}Z.$$

Solver Union and Cooperation by Completion

Bridge rules relate constraints from different programs for their cooperation and communication.

$$and(X, Y, X) \quad \Leftrightarrow \quad imp(X, Y).$$

Non-confluent: E.g.



Completion adds the rules:

$$imp(X, X) \quad \Leftrightarrow \quad true.$$
$$imp(X, Y) \wedge imp(X, Y) \quad \Leftrightarrow \quad imp(X, Y).$$
$$imp(X, Y) \wedge and(X, Y, Z) \quad \Leftrightarrow \quad imp(X, Y) \wedge X{=}Z.$$

Boolean Cardinality

`#(L,U,BL,N)` if between `L` and `U` `1`'s in the list `BL`.

Boolean cardinality can express
- negation `#(0,0,[C],1)`,
- exclusive or `#(1,1,[C1,C2],2)`,
- conjunction `#(N,N,[C1,...,Cn],N)`
- disjunction `#(1,N,[C1,...,Cn],N)`.

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

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

Propositional Resolution

Boolean CSP in CNF: Conjunction of clauses
Clause: Disjunction of Literals
Literal: Positive or negative atomic proposition
Clause as *ordered* list of signed variables.
E.g., $\neg x \vee y \vee z$ as `cl([-x,+y,+z])`.

```
empty_clause @ cl([])  ⇔  false.
tautology    @ cl(L)   ⇔  in(+X,L) ∧ in(-X,L) | true.

resolution   @ cl(L1) ∧ cl(L2) ⇒
                  find(+X,L1,L3) ∧ find(-X,L2,L4) |
                  merge(L3,L4,L) ∧
                  cl(L).
```

Propositional Resolution

Boolean CSP in CNF: Conjunction of clauses
Clause: Disjunction of Literals
Literal: Positive or negative atomic proposition
Clause as *ordered* list of signed variables.
E.g., $\neg x \vee y \vee z$ as `cl([-x,+y,+z])`.

```
empty_clause @  cl([])  ⇔  false.
tautology    @  cl(L)   ⇔  in(+X,L) ∧ in(-X,L) | true.

resolution   @  cl(L1) ∧ cl(L2) ⇒
                   find(+X,L1,L3) ∧ find(-X,L2,L4) |
                   merge(L3,L4,L) ∧
                   cl(L).
```

Linear Polynomial Equations

Equations $a_1x_1 + \ldots + a_nx_n + b = 0, \quad a_i \neq 0, x_i ordered.$
Solved form: leftmost variable of each equation occurs only once.
Reach solved normal form by **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).
```

```
1*X+3*Y+5=0 ∧  3*X+2*Y+8=0
 compute((2*Y+8) - ((3*Y+5)/1)*3,P3)  % P3=-7*Y+ -7
1*X+3*Y+5=0 ∧  -7*Y+ -7=0  % Y=-1
 compute((1*X+5) - ((-7)/-7)*3,P3')   % P3'=1*X+2
1*X+2=0 ∧      -7*Y+ -7=0  % X=-2
```

Linear Polynomial Equations

Equations $a_1x_1 + \ldots + a_nx_n + b = 0, \quad a_i \neq 0, x_i ordered.$
Solved form: leftmost variable of each equation occurs only once.
Reach solved normal form by **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).
```

```
1*X+3*Y+5=0 ∧  3*X+2*Y+8=0
 compute((2*Y+8) - ((3*Y+5)/1)*3,P3)  % P3=-7*Y+ -7
1*X+3*Y+5=0 ∧  -7*Y+ -7=0  % Y=-1
 compute((1*X+5) - ((-7)/-7)*3,P3')   % P3'=1*X+2
1*X+2=0 ∧      -7*Y+ -7=0  % X=-2
```

Linear Polynomial Equations

Equations $a_1x_1 + \ldots + a_nx_n + b = 0, \quad a_i \neq 0, x_i ordered.$
Solved form: leftmost variable of each equation occurs only once.
Reach solved normal form by **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).
```

```
1*X+3*Y+5=0 ∧  3*X+2*Y+8=0
 compute((2*Y+8) - ((3*Y+5)/1)*3,P3)  % P3=-7*Y+ -7
1*X+3*Y+5=0 ∧  -7*Y+ -7=0  % Y=-1
 compute((1*X+5) - ((-7)/-7)*3,P3')   % P3'=1*X+2
1*X+2=0 ∧      -7*Y+ -7=0  % X=-2
```

Linear Polynomial Equations

Equations $a_1x_1 + \ldots + a_nx_n + b = 0, \quad a_i \neq 0, x_i ordered.$
Solved form: leftmost variable of each equation occurs only once.
Reach solved normal form by **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).
```

```
1*X+3*Y+5=0 ∧  3*X+2*Y+8=0
 compute((2*Y+8) - ((3*Y+5)/1)*3,P3)   % P3=-7*Y+ -7
1*X+3*Y+5=0 ∧  -7*Y+ -7=0   % Y=-1
 compute((1*X+5) - ((-7)/-7)*3,P3')    % P3'=1*X+2
1*X+2=0 ∧      -7*Y+ -7=0   % X=-2
```

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).
```

Solver Cooperation, Combination of 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.
```

Solver Cooperation, Combination of 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.
```

Solver Cooperation, Combination of 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 (possibly infinite) 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, if $i \leq j$

```
reflexivity    @   X=X  ⇔   var(X) |  true.
orientation    @   T=X  ⇔   var(X) ∧  X@<T |  X=T.
decomposition  @ T1=T2  ⇔   nonvar(T1) ∧  nonvar(T2) |
                    same_functor(T1,T2) ∧
                    same_args(T1,T2).
confrontation  @ X=T1 ∧  X=T2  ⇔   var(X)∧ X@<T1∧ T1@=<T2 |
                    X=T1 ∧  T1=T2.
```

Syntactic Unification II

```
reflexivity    @   X=X  ⇔   var(X) |  true.
orientation    @   T=X  ⇔   var(X) ∧  X@<T |  X=T.
decomposition  @ T1=T2  ⇔  nonvar(T1) ∧  nonvar(T2) |
                    same_functor(T1,T2) ∧
                    same_args(T1,T2).
confrontation  @ X=T1 ∧  X=T2  ⇔  var(X)∧ X@<T1∧ T1@=<T2 |
                    X=T1 ∧  T1=T2.
```

```
                    h(Y,f(a),g(X,a))=h(f(U),Y,g(h(Y),U))
↦decomposition↦*    Y=f(U) ∧ f(a)=Y ∧ g(X,a)=g(h(Y),U)
↦orientation        Y=f(U) ∧ Y=f(a) ∧ g(X,a)=g(h(Y),U)
↦*                  Y=f(U) ∧ U=a ∧ X=h(Y) ∧ U=a
↦confrontation      Y=f(U) ∧ U=a ∧ X=h(Y) ∧ a=a
↦decomposition↦*    Y=f(U) ∧ U=a ∧ X=h(Y)
```

Syntactic Unification II

```
reflexivity    @   X=X  ⇔   var(X) |  true.
orientation    @   T=X  ⇔   var(X) ∧  X@<T |  X=T.
decomposition  @ T1=T2  ⇔   nonvar(T1) ∧  nonvar(T2) |
                    same_functor(T1,T2) ∧
                    same_args(T1,T2).
confrontation  @ X=T1 ∧  X=T2  ⇔   var(X)∧ X@<T1∧ T1@=<T2 |
                    X=T1 ∧  T1=T2.
```

h(Y,f(a),g(X,a))=h(f(U),Y,g(h(Y),U))

↦decomposition↦* Y=f(U) ∧ f(a)=Y ∧ g(X,a)=g(h(Y),U)

↦orientation Y=f(U) ∧ Y=f(a) ∧ g(X,a)=g(h(Y),U)

↦* Y=f(U) ∧ U=a ∧ X=h(Y) ∧ U=a

↦confrontation Y=f(U) ∧ U=a ∧ X=h(Y) ∧ a=a

↦decomposition↦* Y=f(U) ∧ U=a ∧ X=h(Y)

Syntactic Unification II

```
reflexivity    @   X=X  ⇔   var(X) |  true.
orientation    @   T=X  ⇔   var(X) ∧  X@<T |  X=T.
decomposition  @ T1=T2  ⇔  nonvar(T1) ∧  nonvar(T2) |
                    same_functor(T1,T2) ∧
                    same_args(T1,T2).
confrontation  @ X=T1 ∧  X=T2  ⇔  var(X)∧ X@<T1∧ T1@=<T2 |
                    X=T1 ∧  T1=T2.
```

$\underline{\texttt{h(Y,f(a),g(X,a))=h(f(U),Y,g(h(Y),U))}}$

$\mapsto_{\texttt{decomposition}}\mapsto^*$ $\texttt{Y=f(U) ∧ } \underline{\texttt{f(a)=Y}} \texttt{ ∧ g(X,a)=g(h(Y),U)}$

$\mapsto_{\texttt{orientation}}$ $\texttt{Y=f(U) ∧ Y=f(a) ∧ } \underline{\texttt{g(X,a)=g(h(Y),U)}}$

$\mapsto^*$ $\texttt{Y=f(U) ∧ } \underline{\texttt{U=a}} \texttt{ ∧ X=h(Y) ∧ } \underline{\texttt{U=a}}$

$\mapsto_{\texttt{confrontation}}$ $\texttt{Y=f(U) ∧ U=a ∧ X=h(Y) ∧ } \underline{\texttt{a=a}}$

$\mapsto_{\texttt{decomposition}}\mapsto^*$ $\texttt{Y=f(U) ∧ U=a ∧ X=h(Y)}$

Syntactic Unification II

```
reflexivity    @   X=X  ⇔   var(X) |  true.
orientation    @   T=X  ⇔   var(X) ∧  X@<T |  X=T.
decomposition  @ T1=T2  ⇔  nonvar(T1) ∧  nonvar(T2) |
                    same_functor(T1,T2) ∧
                    same_args(T1,T2).
confrontation  @ X=T1 ∧  X=T2  ⇔  var(X)∧ X@<T1∧ T1@=<T2 |
                    X=T1 ∧  T1=T2.
```

$\mapsto_{\texttt{decomposition}}\mapsto^*$ $\mapsto_{\texttt{orientation}}$ $\mapsto^*$ $\mapsto_{\texttt{confrontation}}$ $\mapsto_{\texttt{decomposition}}\mapsto^*$

```
h(Y,f(a),g(X,a))=h(f(U),Y,g(h(Y),U))
Y=f(U) ∧ f(a)=Y ∧ g(X,a)=g(h(Y),U)
Y=f(U) ∧ Y=f(a) ∧ g(X,a)=g(h(Y),U)
Y=f(U) ∧ U=a ∧ X=h(Y) ∧ U=a
Y=f(U) ∧ U=a ∧ X=h(Y) ∧ a=a
Y=f(U) ∧ U=a ∧ X=h(Y)
```

Syntactic Unification II

```
reflexivity    @   X=X  ⇔   var(X) |  true.
orientation    @   T=X  ⇔   var(X) ∧  X@<T |  X=T.
decomposition  @ T1=T2  ⇔  nonvar(T1) ∧  nonvar(T2) |
                    same_functor(T1,T2) ∧
                    same_args(T1,T2).
confrontation  @ X=T1 ∧  X=T2  ⇔  var(X)∧ X@<T1∧ T1@=<T2 |
                    X=T1 ∧  T1=T2.
```

$\mapsto_{\text{decomposition}}\mapsto^*$
$\mapsto_{\text{orientation}}$
$\mapsto^*$
$\mapsto_{\text{confrontation}}$
$\mapsto_{\text{decomposition}}\mapsto^*$

```
h(Y,f(a),g(X,a))=h(f(U),Y,g(h(Y),U))
Y=f(U) ∧ f(a)=Y ∧ g(X,a)=g(h(Y),U)
Y=f(U) ∧ Y=f(a) ∧ g(X,a)=g(h(Y),U)
Y=f(U) ∧ U=a ∧ X=h(Y) ∧ U=a
Y=f(U) ∧ U=a ∧ X=h(Y) ∧ a=a
Y=f(U) ∧ U=a ∧ X=h(Y)
```

Syntactic Unification II

```
reflexivity    @   X=X  ⇔   var(X) |  true.
orientation    @   T=X  ⇔   var(X) ∧  X@<T |  X=T.
decomposition  @ T1=T2  ⇔  nonvar(T1) ∧  nonvar(T2) |
                    same_functor(T1,T2) ∧
                    same_args(T1,T2).
confrontation  @ X=T1 ∧  X=T2  ⇔  var(X)∧ X@<T1∧ T1@=<T2 |
                    X=T1 ∧  T1=T2.
```

$\mapsto_{\text{decomposition}}\mapsto^*$ $\mathtt{h(Y,f(a),g(X,a))=h(f(U),Y,g(h(Y),U))}$

$\mathtt{Y=f(U) \wedge \underline{f(a)=Y} \wedge g(X,a)=g(h(Y),U)}$

$\mapsto_{\text{orientation}}$ $\mathtt{Y=f(U) \wedge Y=f(a) \wedge \underline{g(X,a)=g(h(Y),U)}}$

$\mapsto^*$ $\mathtt{Y=f(U) \wedge \underline{U=a} \wedge X=h(Y) \wedge \underline{U=a}}$

$\mapsto_{\text{confrontation}}$ $\mathtt{Y=f(U) \wedge U=a \wedge X=h(Y) \wedge \underline{a=a}}$

$\mapsto_{\text{decomposition}}\mapsto^*$ $\mathtt{Y=f(U) \wedge U=a \wedge X=h(Y)}$

Syntactic Unification II

```
reflexivity    @   X=X  ⇔   var(X) |  true.
orientation    @   T=X  ⇔   var(X) ∧  X@<T |  X=T.
decomposition  @ T1=T2  ⇔  nonvar(T1) ∧  nonvar(T2) |
                    same_functor(T1,T2) ∧
                    same_args(T1,T2).
confrontation  @ X=T1 ∧  X=T2  ⇔  var(X)∧ X@<T1∧ T1@=<T2 |
                    X=T1 ∧  T1=T2.
```

$\mapsto_{\texttt{decomposition}}\mapsto^*$ $\mapsto_{\texttt{orientation}}$ $\mapsto^*$ $\mapsto_{\texttt{confrontation}}$ $\mapsto_{\texttt{decomposition}}\mapsto^*$

```
h(Y,f(a),g(X,a))=h(f(U),Y,g(h(Y),U))
Y=f(U) ∧ f(a)=Y ∧ g(X,a)=g(h(Y),U)
Y=f(U) ∧ Y=f(a) ∧ g(X,a)=g(h(Y),U)
Y=f(U) ∧ U=a ∧ X=h(Y) ∧ U=a
Y=f(U) ∧ U=a ∧ X=h(Y) ∧ a=a
Y=f(U) ∧ U=a ∧ X=h(Y)
```

Finite Domains / Interval Arithmetic

$$
\begin{array}{rcl}
X :: A..B & \Rightarrow & A \leq B \\
X :: A1..B1 \wedge X :: A2..B2 & \Leftrightarrow & max(A1, A2, A) \wedge \\
 & & min(B1, B2, B) \wedge \\
 & & X :: A..B \\
X \leq Y \wedge X :: AX..BX \wedge Y :: AY..BY & \Rightarrow & BX > BY \mid \\
 & & X :: AX..BY \\
X \leq Y \wedge X :: AX..BX \wedge Y :: AY..BY & \Rightarrow & AX > AY \mid \\
 & & Y :: AX..BY
\end{array}
$$

$$
\begin{array}{rl}
 & A :: 2..3 \wedge B :: 1..2 \wedge A \leq B \\
\mapsto & A :: 2..3 \wedge B :: 1..2 \wedge A \leq B \wedge A :: 2..2 \\
\mapsto & B :: 1..2 \wedge A \leq B \wedge A :: 2..2 \\
\mapsto & B :: 1..2 \wedge A \leq B \wedge A :: 2..2 \wedge B :: 2..2 \\
\mapsto & A \leq B \wedge A :: 2..2 \wedge B :: 2..2
\end{array}
$$

Scheduling

Taming Disjunction. [M. Wallace, Eclipse User Manual]
Two tasks cannot be done at the same time on a single resource.
$\Rightarrow$ Tasks use distinct resource or one task ends before other task starts.

Variables: start times S, end times E and resources R of the two tasks.

```
chrTaskResource(S1,E1,R1,S2,E2,R2) <==>
              R1 = R2, E1 > S2  |  E2 =< S1.
chrTaskResource(S1,E1,R1,S2,E2,R2) <==>
              R1 = R2, E2 > S1  |  E1 =< S2.
chrTaskResource(S1,E1,R1,S2,E2,R2) <==>
              E1 > S2, E2 > S1  |  R1 <> R2.
```

If tasks use same resource and one task cannot precede the other, constrain task not to start until other task ended.

If tasks guaranteed to overlap, constrain them to use distinct resources.

Lexicographic Order

Applications: Termination analysis, Symmetry breaking, Preferences.

Different algorithms for the `lex` constraint

- Non-incremental pointer-based imperative pseudo code, 45 lines
  Frisch/Hnich/Kiziltan et. al., CP 2002.
- Finite automata, interpreted, 16 lines
  Beldiceanu/Carlsson/Petit, CP 2004.

Non-intuitive code. Hard to analyse.

Definition

Given two sequences l_1 and l_2 of length n, $[x_1, \ldots, x_n]$ and $[y_1, \ldots, y_n]$, then $l_1 \preceq_{lex} l_2$ iff $n{=}0$ or $x_1 < y_1$ or $x_1 = y_1$ and $[x_2, \ldots, x_n] \preceq_{lex} [y_2, \ldots, y_n]$.

Lexicographic Order

Applications: Termination analysis, Symmetry breaking, Preferences.

Different algorithms for the `lex` constraint

- Non-incremental pointer-based imperative pseudo code, 45 lines
  Frisch/Hnich/Kiziltan et. al., CP 2002.
- Finite automata, interpreted, 16 lines
  Beldiceanu/Carlsson/Petit, CP 2004.

Non-intuitive code. Hard to analyse.

Logical Specification

$$
\begin{aligned}
l_1 \preceq_{lex} l_2 \quad \leftrightarrow \quad & (l_1{=}[] \wedge l_2{=}[]) \vee \\
& (l_1{=}[x|l_1'] \wedge l_2{=}[y|l_2'] \wedge x{<}y) \vee \\
& (l_1{=}[x|l_1'] \wedge l_2{=}[y|l_2'] \wedge x{=}y \wedge l_1' \preceq_{lex} l_2')
\end{aligned}
$$

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: shown by standard CHR analysis

Part II

...Around the World

Public Domain Implementations

- SWI Prolog (free), XSB Prolog (tabling), hProlog (on request), Tom Schrijvers, K.U.Leuven, 2004
- HAL, ToyCHR (any Prolog), Gregory Duck, Melbourne, 2004
- SICStus Prolog (reference, free trial), Christian Holzbaur, Vienna, 1998
  YAP Prolog (free port), Vitor Santos Costa, 2000
- ECLiPSe Prolog (2), Sepia Prolog (older), Pascal Brisset, Toulouse, 1994; Kish Shen, IC-Parc, London, 1998
- Haskell, Gregory Duck, Jeremy Wazny, Melbourne, 2004; Martin Sulzmann, Singapore, 2004
- K.U.Leuven JCHR: CHR System for Java,
  P. Van Weert, T. Schrijvers, B. Demoen, CHR 2005
- CHREK/KJCHR/DCHR CHR with disjunction in Java, Eclipse Plug-in, L. Menezes, J. Vitorino, M. Aurelio, Brazil 2005
- Java Constraint Kit (JCK), Slim Abdennadher, Cairo, 2002

Public Domain Implementations

- SWI Prolog (free), XSB Prolog (tabling), hProlog (on request), Tom Schrijvers, K.U.Leuven, 2004
- HAL, ToyCHR (any Prolog), Gregory Duck, Melbourne, 2004
- SICStus Prolog (reference, free trial), Christian Holzbaur, Vienna, 1998
  YAP Prolog (free port), Vitor Santos Costa, 2000
- ECLiPSe Prolog (2), Sepia Prolog (older), Pascal Brisset, Toulouse, 1994; Kish Shen, IC-Parc, London, 1998
- Haskell, Gregory Duck, Jeremy Wazny, Melbourne, 2004; Martin Sulzmann, Singapore, 2004
- K.U.Leuven JCHR: CHR System for Java,
  P. Van Weert, T. Schrijvers, B. Demoen, CHR 2005
- CHREK/KJCHR/DCHR CHR with disjunction in Java, Eclipse Plug-in, L. Menezes, J. Vitorino, M. Aurelio, Brazil 2005
- Java Constraint Kit (JCK), Slim Abdennadher, Cairo, 2002

Some Implementation Papers

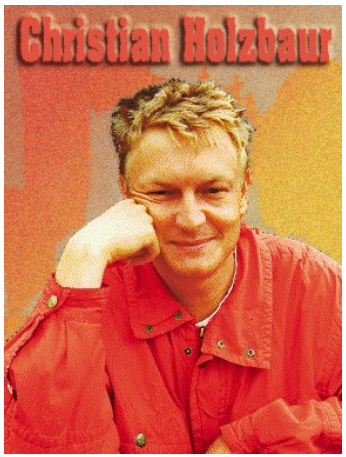
- T. Schrijvers, Analyses, Optimizations and Extensions of CHR, Ph.D. Thesis, KU Leuven, 2005. Poster ICLP 2005. + D.S. Warren, CHR and Tabled Execution, ICLP 2004. Best Paper. + J. Sneyers, B. Demoen, Guard and Continuation Optimization..., ICLP 2005.
- G. J. Duck, C. Holzbaur, M. G. de la Banda, P. J. Stuckey, Optimizing Compilation of CHR in HAL, TPLP CHR Issue 2005. Extending Arbitrary Solvers with CHR, PPDP'03.
- Armin Wolf, Adaptive Constraint Handling with CHR in Java, CP 2001, LNCS 2239. Intelligent Search Strategies Based on Adaptive CHR, TPLP CHR Special Issue 2005.
- C. Holzbaur, T. Frühwirth, A Prolog CHR Compiler and Runtime System, Applied Artificial Intelligence Vol 14(4), 2000.
- S. Abdennadher et. al. JCK: A Java Constraint Kit, ENTCS Vol 64, 2000.

Hard Core CHR People

Slim Abdennadher

Tom Schrijvers

Peter Stuckey



Christian Holzbaur

Armin Wolf

More Semantics

- The Refined Operational Semantics of CHR, Gregory J. Duck, Peter J. Stuckey, Maria Garcia de la Banda, Christian Holzbaur, ICLP'04.
  Textual rule order. Left-to-right execution of queries.
- A Linear Logic Semantics for CHR, Hariolf Betz, CP 2005.
  Model change: dynamic resources, actions and states.
  `switch(on), light(_) <=> light(on).`
  `switch(off), light(_) <=> light(off).`
- A Compositional Semantics for CHR, Giorgio Delzanno, Maurizio Gabbrielli, Maria Chiaria Meo, PPDP 2005.
  Multiple heads are challenging.
- Abstract Interpretation for CHR, Tom Schrijvers, Peter Stuckey, Gregory Duck, PPDP 2005; CHR 2005.
  Framework supporting refined semantics; e.g. functional dependence.

Program Generation

- Automatic Generation of CHR Constraint Solvers, Slim Abdennadher, Christophe Rigotti, TPLP CHR Special Issue 2005.
- Schedulers and Redundancy for a Class of Constraint Propagation Rules, Sebastian Brand, Krzysztof Apt, TPLP CHR Special Issue 2005.
- Constraint programming viewed as rule-based programming Krzysztof Apt, Eric Monfroy, TPLP 2001.

Example Rule Generation

Intensional definition of minimum:

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

Automatically generate propagation rules:

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

Example Rule Generation

Intensional definition of minimum:

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

Derived constraint handling rules:

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

Program Transformation

- Specialization of Concurrent Guarded Multi-Set Transformation Rules, T. Frühwirth, LOPSTR 2004, LNCS.
  Multiple heads make partial evaluation hard.
- Integration and Optimization of Rule-Based Constraint Solvers, S. Abdennadher, T. Frühwirth, LOPSTR 2003, LNCS 3018.
  Are termination and confluence modular?
- Source-to-Source Transformation for a Class of Expressive Rules, T. Frühwirth, C. Holzbaur, AGP 2003.
  To implement extensions and optimizations.

Union of Constraint Progams

Well-Behaved CHR program: terminating and confluent.
Modularity: Is well-Behavedness preserved?

- $P_1 = \{a \Leftrightarrow b\}$
  $P_2 = \{b \Leftrightarrow a\}$
  The union of P_1 and P_2 is non-terminating.
  Remedy: Turn into propagation rules or rename constraints.
- $P_1 = \{a \Leftrightarrow b\}$
  $P_3 = \{a \Leftrightarrow c\}$
  The union of P_1 and P_3 is non-confluent.
  Remedy: Completion adds $\{b \Leftrightarrow c\}$.

Union of Constraint Progams

Well-Behaved CHR program: terminating and confluent.
Modularity: Is well-Behavedness preserved?

- $P_1 = \{a \Leftrightarrow b\}$
  $P_2 = \{b \Leftrightarrow a\}$
  The union of P_1 and P_2 is non-terminating.
  Remedy: Turn into propagation rules or rename constraints.
- $P_1 = \{a \Leftrightarrow b\}$
  $P_3 = \{a \Leftrightarrow c\}$
  The union of P_1 and P_3 is non-confluent.
  Remedy: Completion adds $\{b \Leftrightarrow c\}$.

Union of Constraint Progams

Well-Behaved CHR program: terminating and confluent.
Modularity: Is well-Behavedness preserved?

- $P_1 = \{a \Leftrightarrow b\}$
  $P_2 = \{b \Leftrightarrow a\}$
  The union of P_1 and P_2 is non-terminating.
  Remedy: Turn into propagation rules or rename constraints.
- $P_1 = \{a \Leftrightarrow b\}$
  $P_3 = \{a \Leftrightarrow c\}$
  The union of P_1 and P_3 is non-confluent.
  Remedy: Completion adds $\{b \Leftrightarrow c\}$.

Soft Constraints

Soft Constraint Propagation and Solving in CHR, S. Bistarelli, T. Frühwirth, M. Marte, F. Rossi, Computational Intelligence 20(2), 2004.
Semi-ring constraint algorithms easy in CHR.

E.g. Fuzzy Constraints:

`X≤Y:A, Y≤Z:B ==> X≤Z:A*B`

`E≤F:0.5, F≤G:0.4` $\mapsto$
`E≤F:0.5, F≤G:0.4, E≤G:0.2`

Randomized Algorithms

Probabilistic Constraint Handling Rules, T. Frühwirth, A. Di Pierro, H. Wiklicky, WFLP 2002, ENTCS. *CHR with randomized rule choice.*

Random walk

```
walked_to(0) <=> true
walked_to(N) <=>0.5 walked_to(N+1)
walked_to(N) <=>0.5 walked_to(N-1)
```

Probabilistic termination.

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.

Don't-care and Don't-know Nondeterminism

The CHR$^\vee$ program for append of two lists

```
append(X,Y,Z) ⇔
    (     X=[] ∧ Y=L ∧ Z=L
    ∨     X=[H|L1] ∧ Y=L2 ∧ Z=[H|L3] ∧ append(L1, L2, L3) ).
```

can be improved by adding the following rule

```
append(X,[],Z) ⇔ X = Z.
```

Don't-care and Don't-know Nondeterminism

The CHR$^\vee$ program for append of two lists

```
append(X,Y,Z) ⇔
    (     X=[] ∧ Y=L ∧ Z=L
    ∨     X=[H|L1] ∧ Y=L2 ∧ Z=[H|L3] ∧ append(L1, L2, L3) ).
```

can be improved by adding the following rule

```
append(X,[],Z) ⇔ X = Z.
```

Don't-care and Don't-know Nondeterminism

The CHR$^\vee$ program for append of two lists

```
append(X,Y,Z) ⇔
    (     X=[] ∧ Y=L ∧ Z=L
    ∨     X=[H|L1] ∧ Y=L2 ∧ Z=[H|L3] ∧ append(L1, L2, L3) ).
```

can be improved by adding the following rule

```
append(X,[],Z) ⇔ X = Z.
```

Bottom-up and Top-down Evaluation with Tabling

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

Top-down Evaluation

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

Bottom-up and Top-down Evaluation with Tabling

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

Top-down Evaluation with Tabling

$$\texttt{fib(N,M1)} \wedge \texttt{fib(N,M2)} \Leftrightarrow \texttt{M1 = M2} \wedge \texttt{fib(N,M1).}$$

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

Bottom-up and Top-down Evaluation with Tabling

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

Bottom-up 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).
```

Bottom-up and Top-down Evaluation with Tabling

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

Bottom-up 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).
```

Bottom-up and Top-down Evaluation with Tabling

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

Bottom-up Evaluation with Termination, Last 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).
```

Abduction

Abducibles: predicates only partially defined by integrity constraints.
Abducibles as CHR constraints.

A bird is either an albatros or a penguin.

$\texttt{bird(X)} \Leftrightarrow \texttt{albatros(X)} \vee \texttt{penguin(X)}.$

Penguins can't fly.

$\texttt{penguin(X)} \wedge \texttt{flies(X)} \Leftrightarrow \mathit{false}.$

The query $\texttt{bird(X)} \wedge \texttt{flies(X)}$ leads to the only answer $\texttt{albatros(X)} \wedge \texttt{flies(X)}$.

Abduction

Abducibles: predicates only partially defined by integrity constraints.
Abducibles as CHR constraints.

A bird is either an albatros or a penguin.

$\texttt{bird(X)} \Leftrightarrow \texttt{albatros(X)} \vee \texttt{penguin(X)}.$

Penguins can't fly.

$\texttt{penguin(X)} \wedge \texttt{flies(X)} \Leftrightarrow \mathit{false}.$

The query $\texttt{bird(X)} \wedge \texttt{flies(X)}$ leads to the only answer $\texttt{albatros(X)} \wedge \texttt{flies(X)}$.

Bottom-up Evaluation of Logic Programs

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

is transformed into

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

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

Bottom-up Evaluation of Logic Programs

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

is transformed into

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

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

Bottom-up Evaluation of Logic Programs

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

is transformed into

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

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

$$\downarrow$$

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

$$\downarrow$$

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

$$\downarrow$$

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

Bottom-up Evaluation of Logic Programs

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

is transformed into

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

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

Bottom-up Evaluation of Logic Programs

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

is transformed into

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

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

$$\downarrow$$

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

$$\downarrow$$

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

$$\downarrow$$

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

Bottom-up Evaluation of Logic Programs

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

is transformed into

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

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

Bottom-up Evaluation of Logic Programs

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

is transformed into

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

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

Bottom-up Evaluation of Logic Programs

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

is transformed into

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

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

Bottom-up Evaluation of Logic Programs

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

is transformed into

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

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

Prime Numbers Sieve

In Prolog (or in a concurrent constraint language)

```
primes(N,Ps):- candidates(2,N,Ns), sift(Ns,Ps).

candidates(F,T,[]):- F>T.
candidates(F,T,[F|Ns1]):- F=<T, F1 is F+1, candidates(F1,T,Ns1).

sift([],[]).
sift([P|Ns],[P|Ps1]):- filter(Ns,P,Ns1), sift(Ns1,Ps1).

filter([],P,[]).
filter([X|In],P,Out):- 0 =:= X mod P, filter(In,P,Out).
filter([X|In],P,[X|Out1]):- 0 =\= X mod P, filter(In,P,Out1).
```

Prime Numbers Sieve

In CHR

```
candidates(N) <=> N>1 | M is N-1, prime(N), candidates(M).
candidates(1) <=> true.

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

Less code: no explicit loops, easy analysis. Concurrent, incremental.

```
prime(7), prime(6), prime(5), prime(4), prime(3), prime(2)
|
prime(7), prime(5), prime(4), prime(3), prime(2)
|
prime(7), prime(5), prime(3), prime(2)
```

Prime Numbers Sieve

JCHR, first CHR in Java, part of the Java Constraint Kit JCK [Abdennadher+, ENTCS Vol 64, 2000].

```
handler primes { class java.lang.Integer; class IntUtil;

constraint prime(java.lang.Integer);
constraint candidates(java.lang.Integer);

rules { variable java.lang.Integer N, M, I, J;

if (IntUtil.gt(N,1)) {candidates(N)} <=>
         {M=IntUtil.dec(N) && prime(N) && candidates(M)};
{candidates(1)} <=> {true} ;

if (IntUtil.modNull(J,I)){prime(I) && prime(J)} <=> {true} sift;
      }
}
```

Prime Numbers Sieve

K.U.Leuven JCHR system [P. Van Weert, T. Schrijvers, B. Demoen, 2005].

```
import util.arithmetics.primitives.intUtil;

handler primes {
  constraint candidates(int);
  constraint prime(int);

  rules { variable int N, X, Y;

    candidates(1) <=> true.
    candidates(N) <=> prime(N), candidates(intUtil.dec(N)).

    sift @ prime(Y) \ prime(X) <=> intUtil.modZero(X, Y) | true.
  }
}
```

Factorial

Primes

```
candidates(N) <=> N>1 | M is N-1, prime(N), candidates(M).
candidates(1) <=> true.

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

Factorial

Factorial

```
candidates(N) <=> N>1 | M is N-1, factorial(N), candidates(M).
candidates(1) <=> true.

fact @ factorial(I) , factorial(J) <=> factorial(I*J).
```

Basic Union-Find

Tom Schrijvers, Thom Frühwirth, TPLP Programming Pearl, to appear.

```
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

Tom Schrijvers, Thom Frühwirth, TPLP Programming Pearl, to appear.

```
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)).
```

Confluence for Parallelism

Maximize independence, non-interference of rule applications.
Avoid waiting and deadlocks. Robust against local failures.

```
make(a), make(b), union(a,b), find(b,X).
| make    | make    | union
root(a), root(b), find(a,U),  find(b,V), link(U,V), find(b,X).
                  | findRoot  | findRoot            | findRoot
root(a), root(b), U=a,        V=b,        link(U,V), X=b.
                                          | link
root(a),          U=a,        V=b,        V -> U,    X=b.
```

Confluence for Parallelism

Maximize independence, non-interference of rule applications.
Avoid waiting and deadlocks. Robust against local failures.

```
make(a), make(b), union(a,b), find(b,X).
| make    | make    | union
root(a), root(b), find(a,U),  find(b,V), link(U,V), find(b,X).
                  | findRoot  | findRoot
root(a), root(b), U=a,        V=b,        link(U,V), find(b,X).
                                          | link
root(a),          U=a,        V=b,        V -> U,    find(b,X).
                                                     | findNode
root(a),          U=a,        V=b,        V -> U,    find(a,X).
                                                     | findRoot
root(a),          U=a,        V=b,        V -> U,    X=a.
```

Note `X=a` was `X=b` before. Result depends on order of rule applications.

Sorting

One-rule sort related to merge sort and tree sort.
Arc `X -> Ai` for each unique value `Ai`, `X` only on left.
Ordered chain of arcs `X -> A1, A1 -> A2,...`.
Quadratic worst-case time complexity.

```
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.`

Correctness: Sorted?
- `A -> B` implies `A@<B`.
- Values can only move from right to left.
- Graph always stays connected.
- Final graph is a chain.

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

Sorting

One-rule sort related to merge sort and tree sort.
Arc `0 => Ai` for each unique value `Ai`, left side is level.
Optimal log-linear worst-case time complexity.

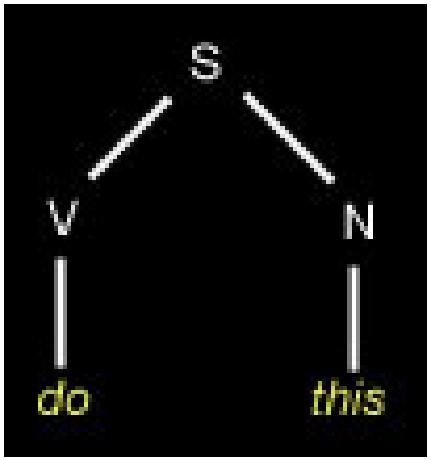
```
sort @ X -> A \ X -> B <=> A@<B | A -> B.
```

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

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

Complexity Analysis left as an exercise...

Dynamic Programming: Parsing



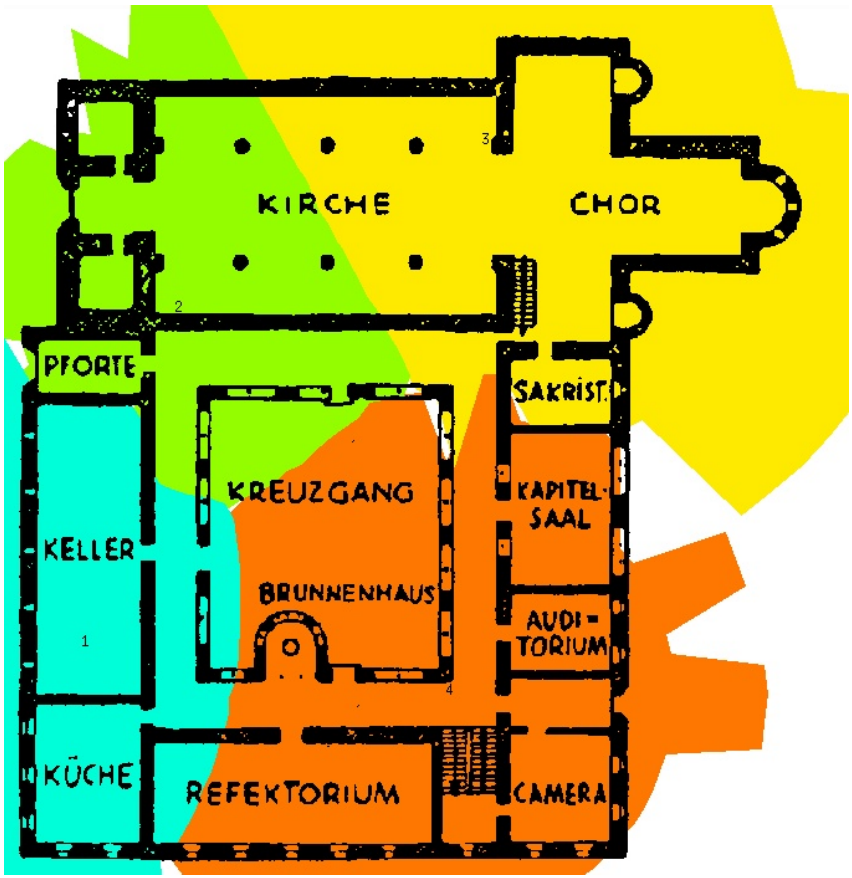
The Cocke-Younger-Kasami Algorithm
for grammars in *Chomsky normal form*:
Grammar rules = `A->T` or `A->B*C`.
Word = Sequence of tokens (terminal symbols).

`terminal@` `A->T` $\wedge$ `word(T+R)` $\Rightarrow$ `parses(A,T+R,R).`

`non-term@` `A->B*C` $\wedge$ `parses(B,I,J)`$\wedge$`parses(C,J,K)` $\Rightarrow$ `parses(A,I,K)`

`substrng@` `word(T+R)` $\Rightarrow$ `word(R).`

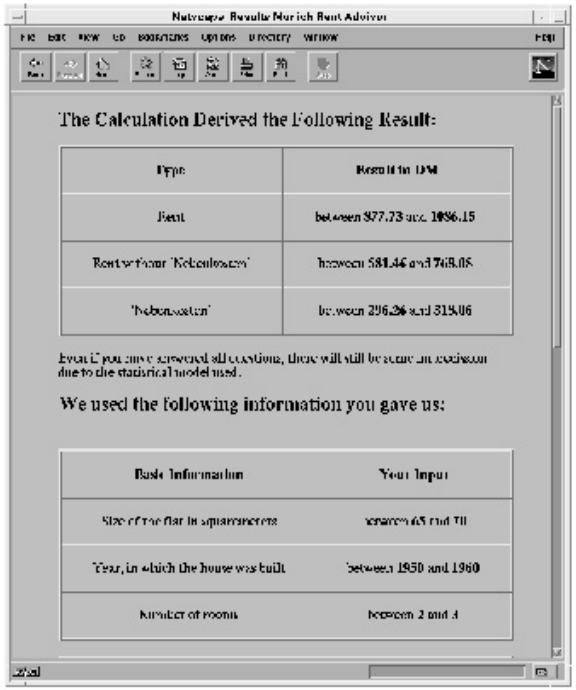
POPULAR - Planning Cordless Communication



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.

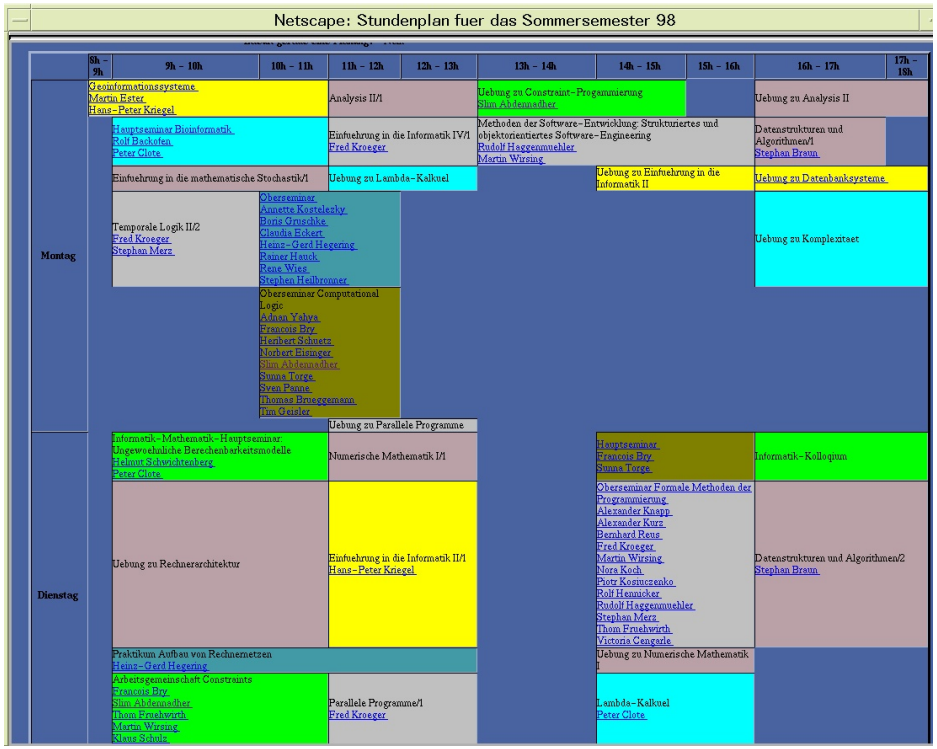
MRA - The Munich Rent Advisor



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

Most Popular Constraint-Based Internet Application.

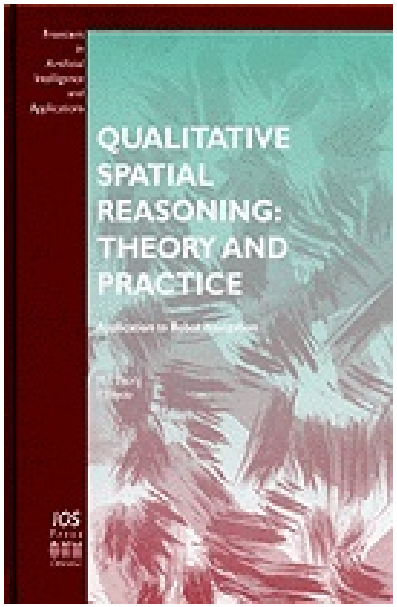
University Course Timetabling



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.

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.

Agents and Actions

Implication and Universal Quantification Constraints in FLUX, CP 2005.
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.

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.

Haskell type classes, implication and dependent types.

```
Sub(Int,Float)          <=> true;
Sub(a1->a2, b1->b2)     <=> Sub(b1,a1), Sub(a2,b2);
```

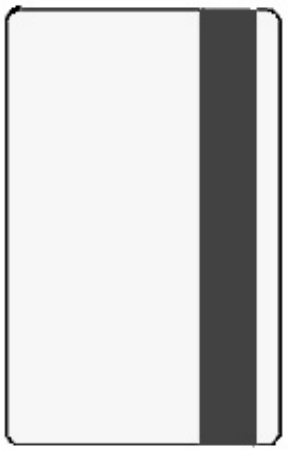
Types and Security

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 Test Case Generation for OCL, P. A. P. Salas, B. K. Aichernig, IIST, UN Uni, 2005.
- Test Cases for the Java Card Virtual Machine, S.-D. Gouraud, A. Gotlieb, JFPC05, France.
- Automatic Generation of Test Data - J. Harm, University Rostock, Germany.
- Executable Z-Specifications - P. Stuckey, Ph. Dart, University Melbourne.

Semantic Web

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.

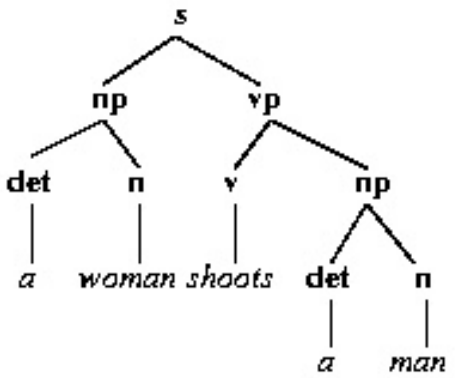
- S. Bressan, C.H. Goh, S. Madnick, M. Siegel et. al.
Context Knowledge Representation and Reasoning in the Context Interchange System, Applied Intelligence, Vol 13:2, 2000;
Context Interchange...for the intelligent integration of information, ACM Transactions on Information Systems, 1999.

Computational Linguistics

Coordination Revisited: A CHR Approach, V. Dahl+, LNCS 3315, 2004.
CHR Grammars, H. Christiansen, TPLP CHR Special Issue 2005.
Meaning in Context, H. Christiansen, V. Dahl, LNCS 3554, CONTEXT'05, 2005.
Assumptions and Abduction in Prolog, H. Christiansen, V. Dahl, WLPE 2004.

Extracting Selected Phrases through Constraint Satisfaction, V. Dahl, Ph. Blache, CSLP 2005.

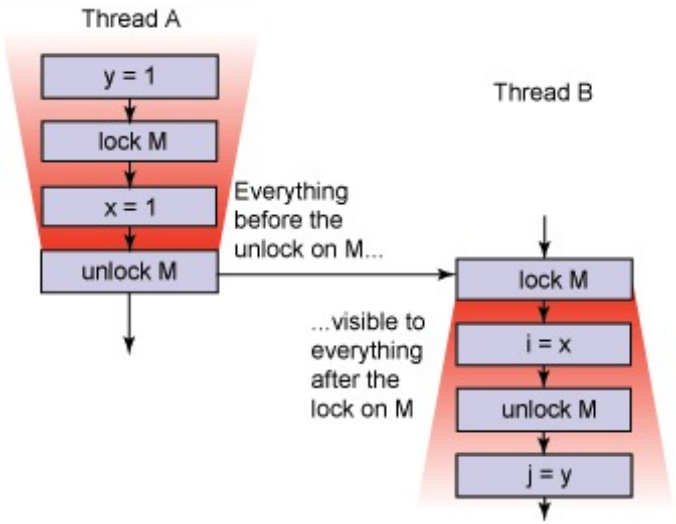
Topological Parsing, Gerald Penn et. al., EACL'03.



- Property Grammars, HPSG, Philippe Blache, Aix; Frank Morawietz, Tuebingen.
- Morphological Analysis, Juergen Oesterle, Univ. Munich, CIS.

Java Memory Machine

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



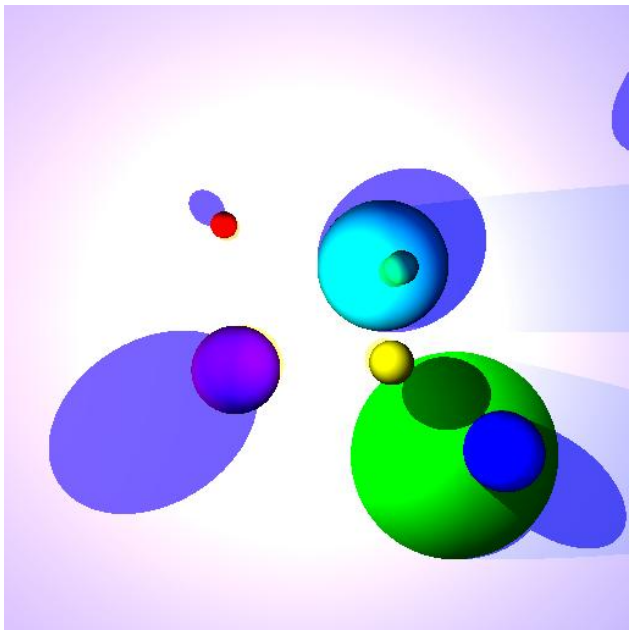
Conditional Read

```
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.
```

ToyCHR Ray Tracing

Gregory J. Duck, University of Melbourne.
Scene calculated from interferring objects and rays.



Cast a ray for a pixel.
Calculate intersection.

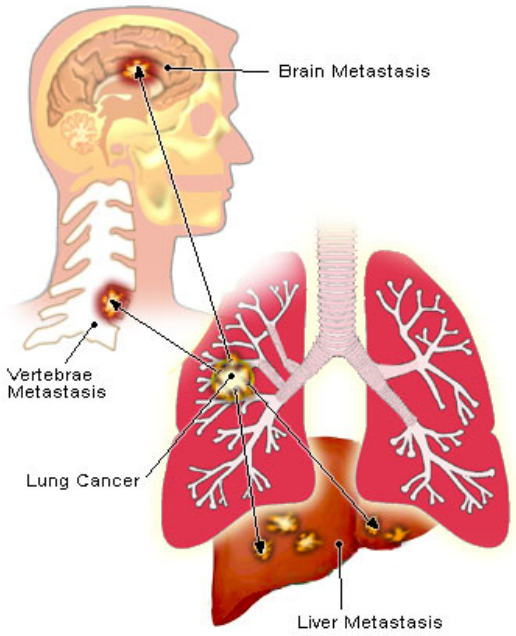
```
sphere(I,XYZ,R,Col), ray(XYZs) ==>
        ...ray(XYZs,Color,I).
```

Calculate shadows or blend colors.

```
sphere(I,XYZ,R,_) \ ray(XYZs,_,J) <=>
        I\=J, block...|...
sphere(I,XYZ,_,C1) \ ray(XYZs,C2,J) <=>
        I=J | ...
```

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.



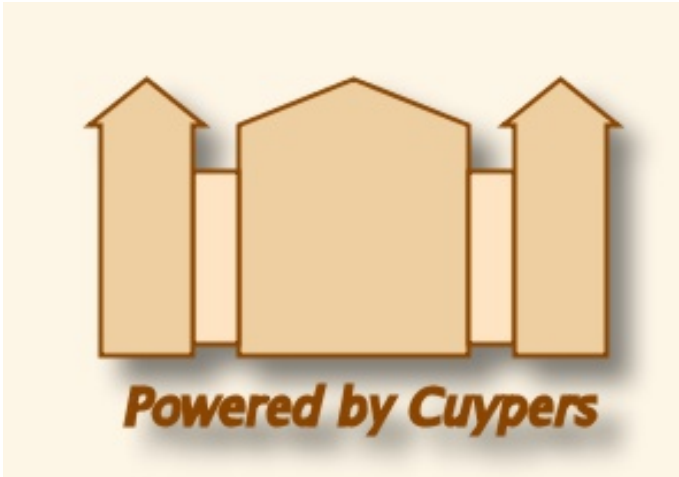
Lung Cancer and Metastasis

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

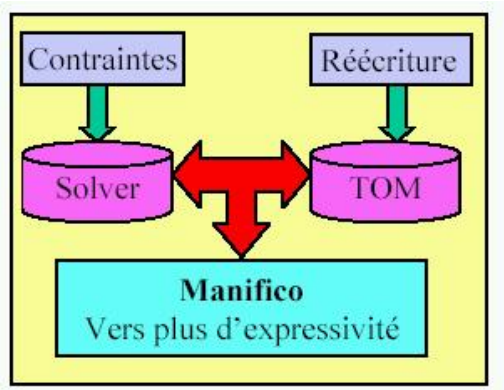
Joost Geurts, University of Amsterdam.
Automatic generation of interactive, time-based and media centric WWW presentations from semi-structured multimedia databases.



Business Rules for Optimization

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

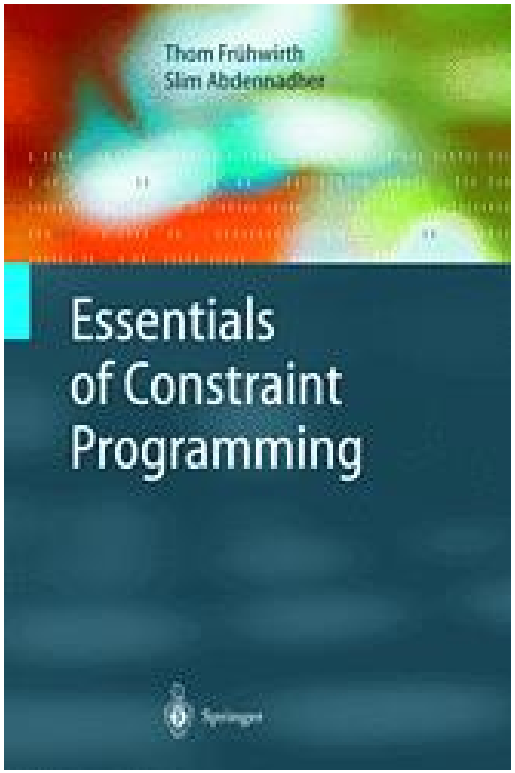
Business Rule: defines or constrains behavior or structure of business.
"A car must be available to be assigned to a rental agreement".



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

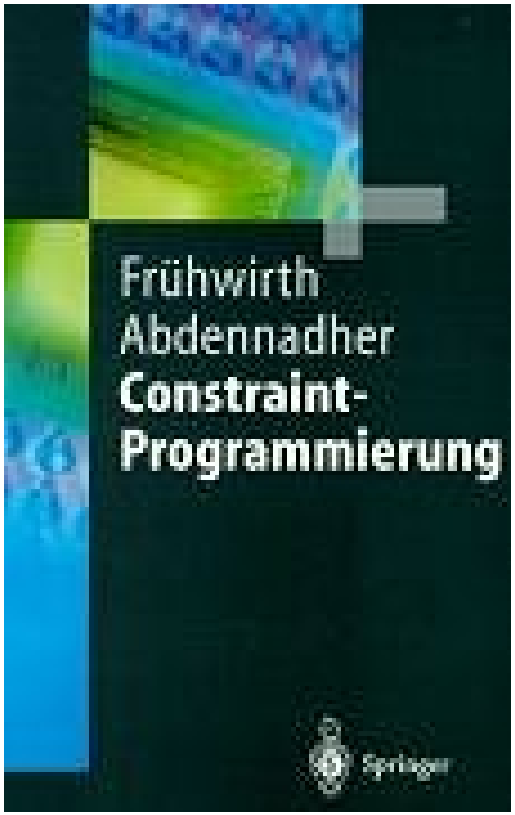
```
reservation(Renter,Group,From,To),
available(car(Id,Group,...),From) <=>...
        rentagreement(Renter,Id,From,To).
```

Further Reading



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

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



The CHR Logo

The CHR Logo

Transcribed as **CHR**, means

The CHR Logo

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.