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

Images are subject to copyright of the respective owners

Citations may be incomplete for space reasons
Overview

CHR...

1. Constraint Handling Rules (CHR)
2. Program Analysis
3. Constraint Solvers
4. Language Issues
5. From Computational Logic to Logical Computations
6. Classical Applications
7. Trends in Applications
Part I

CHR...

1. Constraint Handling Rules (CHR)
   - Example Partial Order
   - Syntax and Declarative Semantics
   - Operational Semantics
   - Operational Properties

2. Program Analysis
   - Termination and Complexity
   - Confluence and Completion
   - Operational Equivalence

3. Constraint Solvers
   - Boolean Constraints
   - Linear Polynomial Equations
   - Syntactic Unification of Rational Trees
   - Finite Domains
   - Scheduling
   - Global Lexicographic Order Constraint
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: Implementations 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,...
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,...

CHR-Constraints

CHR Solver

Built-in Constr.

Blackbox
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)
Constraint Handling Rules (CHR)
Program Analysis
Constraint Solvers

Example Partial Order
Syntax and Declarative Semantics
Operational Semantics
Operational Properties

Example Partial Order Constraint

\[
\begin{align*}
X \leq X & \iff \text{true} \quad \text{(reflexivity)} \\
X \leq Y \land Y \leq X & \iff X = Y \quad \text{(antisymmetry)} \\
X \leq Y \land Y \leq Z & \implies X \leq Z \quad \text{(transitivity)}
\end{align*}
\]

\[
\begin{align*}
A \leq B \land B \leq C \land C \leq A \\
\downarrow \\
A \leq B \land B \leq C \land C \leq A \land A \leq C \quad \text{(transitivity)} \\
\downarrow \\
A \leq B \land B \leq C \land A = C \quad \text{(antisymmetry)} \\
\downarrow \\
A \leq B \land B \leq A \land A = C \quad \text{(built-in solver)} \\
\downarrow \\
A = B \land A = C \quad \text{(antisymmetry)}
\end{align*}
\]
Example Partial Order Constraint

\[
\begin{align*}
X \leq X & \iff \text{true} \quad \text{(reflexivity)} \\
X \leq Y \land Y \leq X & \iff X = Y \quad \text{(antisymmetry)} \\
X \leq Y \land Y \leq Z & \implies X \leq Z \quad \text{(transitivity)}
\end{align*}
\]

\[
\begin{align*}
A \leq B \land B \leq C \land C & \leq A \\
\downarrow
\end{align*}
\]

\[
\begin{align*}
A \leq B \land B \leq C \land C \leq A \land A & \leq C \\
\downarrow
\end{align*}
\]

\[
\begin{align*}
A \leq B \land B \leq C \land A = C \\
\downarrow
\end{align*}
\]

\[
\begin{align*}
A \leq B \land B \leq A \land A = C \\
\downarrow
\end{align*}
\]

\[
\begin{align*}
A = B \land A = C
\end{align*}
\]

(antisymmetry)

(antisymmetry)

(built-in solver)
Example Partial Order Constraint

\[ X \leq X \iff \text{true} \] (reflexivity)
\[ X \leq Y \land Y \leq X \iff X = Y \] (antisymmetry)
\[ X \leq Y \land Y \leq Z \implies X \leq Z \] (transitivity)

\[
\begin{align*}
A \leq B \land B \leq C \land C \leq A \\
\Downarrow
A \leq B \land B \leq C \land C \leq A \land A \leq C
\end{align*}
\] (transitivity)

\[
\begin{align*}
A \leq B \land B \leq C \land A = C \\
\Downarrow
A = B \land A = C
\end{align*}
\] (antisymmetry)
Example Partial Order Constraint

\[ X \leq X \iff true \quad \text{(reflexivity)} \]
\[ X \leq Y \land Y \leq X \iff X = Y \quad \text{(antisymmetry)} \]
\[ X \leq Y \land Y \leq Z \implies X \leq Z \quad \text{(transitivity)} \]

\[ \overbrace{A \leq B} \land \overbrace{B \leq C} \land \overbrace{C \leq A} \quad \text{(transitivity)} \]
\[ A \leq B \land B \leq C \land \overbrace{C \leq A} \land \overbrace{A \leq C} \quad \text{(antisymmetry)} \]
\[ A \leq B \land B \leq C \land \overbrace{A = C} \quad \text{(built-in solver)} \]
\[ A \leq B \land B \leq A \land A = C \quad \text{(antisymmetry)} \]
\[ A = B \land A = C \]
Example Partial Order Constraint

\[
\begin{align*}
X \leq X & \iff true \quad \text{(reflexivity)} \\
X \leq Y \land Y \leq X & \iff X = Y \quad \text{(antisymmetry)} \\
X \leq Y \land Y \leq Z & \Rightarrow X \leq Z \quad \text{(transitivity)}
\end{align*}
\]

\[
\begin{align*}
A \leq B \land B \leq C \land C \leq A \\
\downarrow \\
A \leq B \land B \leq C \land C \leq A \land A \leq C \\
\downarrow \\
A \leq B \land B \leq C \land A = C \\
\downarrow \\
A \leq B \land B \leq A \land A = C \\
\downarrow \\
A = B \land A = C
\end{align*}
\]

(antisymmetry)

(antisymmetry)

(built-in solver)
Example Partial Order Constraint

\[ X \leq X \iff \text{true} \quad \text{(reflexivity)} \]
\[ X \leq Y \land Y \leq X \iff X = Y \quad \text{(antisymmetry)} \]
\[ X \leq Y \land Y \leq Z \implies X \leq Z \quad \text{(transitivity)} \]

\[ \bar{A \leq B} \land \bar{B \leq C} \land \bar{C \leq A} \]
\[ \bar{A \leq B} \land \bar{B \leq C} \land \bar{C \leq A} \land \bar{A \leq C} \]
\[ \bar{A \leq B} \land \bar{B \leq C} \land \bar{A = C} \]
\[ \bar{A \leq B} \land \bar{B \leq A} \land \bar{A = C} \]
\[ A = B \land A = C \]
Example Partial Order Constraint

\[ X \leq Y \iff X = Y \mid \text{true} \]  \hspace{1cm} \text{(reflexivity)}

\[ X \leq Y \land Y \leq X \iff X = Y \]  \hspace{1cm} \text{(antisymmetry)}

\[ X \leq Y \land Y \leq Z \implies X \leq Z \]  \hspace{1cm} \text{(transitivity)}

\[ \overline{A \leq B} \land \overline{B \leq C} \land \overline{C \leq A} \]
\[ \downarrow \]
\[ \overline{A \leq B} \land \overline{B \leq C} \land \overline{C \leq A} \land \overline{A \leq C} \]
\[ \downarrow \]
\[ \overline{A \leq B} \land \overline{B \leq C} \land \overline{A \leq C} \]
\[ \downarrow \]
\[ A = B \land A = C \]  \hspace{1cm} \text{(transitivity)}

\[ \overline{A \leq B} \land \overline{B \leq A} \land \overline{A = C} \]
\[ \downarrow \]
\[ A = B \land A = C \]  \hspace{1cm} \text{(antisymmetry)}

\[ \overline{A \leq B} \land \overline{B \leq A} \land \overline{A = C} \]
\[ \downarrow \]
\[ A = B \land A = C \]  \hspace{1cm} \text{(built-in solver)}

\[ A = B \land A = C \]  \hspace{1cm} \text{(antisymmetry)}
Syntax and Declarative Semantics

Logical Reading

**Simplification rule:** \( H \iff C \mid B \forall \bar{x} \ (C \rightarrow (H \leftrightarrow \exists \bar{y} B)) \)

**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 \(CT \models G_{builtin} \rightarrow \exists \bar{x}(H = H' \land C)\)
then \(H' \land G \leftrightarrow G \land H = H' \land B\)

**Propagate**

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

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 \(CT \models G_{builtin} \rightarrow \exists \bar{x}(H=H' \land C)\)
then \(H' \land G \leftrightarrow G \land H=H' \land B\)

**Propagate**

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

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 \((H \iff C \mid B)\) rule with renamed fresh variables \(\bar{x}\)
and \(CT \models G_{builtin} \rightarrow \exists \bar{x}(H=H' \land C)\)
then \(H' \land G \leftrightarrow G \land H=H' \land B\)

**Propagate**

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

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 \((H \iff C | B)\) rule with renamed fresh variables \(\bar{x}\)
and \(CT \models G_{builtin} \rightarrow \exists \bar{x}(H \equiv H' \land C)\)
then \(H' \land G \leftrightarrow G \land H \equiv H' \land B\)

**Propagate**

If \((H \Rightarrow C | B)\) rule with renamed fresh variables \(\bar{x}\)
and \(CT \models G_{builtin} \rightarrow \exists \bar{x}(H \equiv H' \land C)\)
then \(H' \land G \leftrightarrow H' \land G \land H \equiv H' \land B\)

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.

\[
\begin{align*}
A \leq B & \land B \leq C & \land C \leq A \\
A \leq B & \land B \leq C & \land C \leq A & \land A \leq C \\
A \leq B & \land B \leq C & \land A = C \\
A \leq B & \land B \leq A & \land A = C \\
A = B & \land A = C
\end{align*}
\]

(transitivity)

(antisymmetry)

(built-in solver)

(antisymmetry)
Anytime Algorithm

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

\[
\begin{align*}
A \leq B & \land B \leq C \land C \leq A \\
\rightarrow & \\
A \leq B & \land B \leq C \land C \leq A & \land A \leq C \\
\rightarrow & \\
A \leq B & \land B \leq C & \land A \leq C \\
\rightarrow & \\
A \leq B & \land B \leq A & \land A \leq C \\
\rightarrow & \\
A = B & \land A = C
\end{align*}
\]

(transitivity)  
(antisymmetry)  
(built-in solver)  
(antisymmetry)
Anytime Algorithm

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

\[
A \leq B \land B \leq C \land C \leq A \\
\quad \downarrow \\
A \leq B \land B \leq C \land C \leq A \land A \leq C \\
\quad \downarrow \\
A \leq B \land B \leq C \land A = C \\
\quad \downarrow \\
A \leq B \land B \leq A \land A = C \\
\quad \downarrow \\
A = B \land A = C
\]

(transitivity)

(antisymmetry)

(built-in solver)

(antisymmetry)
Online Algorithm

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

Monotonicity and Incrementality

If \( G \rightarrow G' \)
then \( G \land C \rightarrow G' \land C \)

Applicable rules cannot become inapplicable in larger context.

\[
\begin{align*}
A \leq B \land & B \leq C \land C \leq A \\
\downarrow & \\
A \leq B \land B \leq C \land A \leq C \land C \leq A & \text{(transitivity)} \\
\downarrow & \\
A \leq B \land B \leq C \land A = C & \text{(antisymmetry)} \\
\downarrow & \\
\ldots
\end{align*}
\]
Online Algorithm

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

**Monotonicity and Incrementality**

If \( G \xrightarrow{} G' \)
then \( G \land C \xrightarrow{} G' \land C \)

Applicable rules cannot become inapplicable in larger context.

\[
\begin{align*}
A \leq B \land B \leq C \land C \leq A & \quad \text{(transitivity)} \\
A \leq B \land B \leq C \land A \leq C \land C \leq A & \\
A \leq B \land B \leq C \land A = C & \quad \text{(antisymmetry)} \\
\ldots & 
\end{align*}
\]
Online Algorithm

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

Monotonicity and Incrementality

If \( G \overset{\rightarrow}{\Rightarrow} G' \)

then \( G \land C \overset{\rightarrow}{\Rightarrow} G' \land C \)

Applicable rules cannot become inapplicable in larger context.

\[
\begin{align*}
\frac{A \leq B \land B \leq C \land C \leq A}{A \leq B \land B \leq C \land A \leq C \land C \leq A} & \quad \text{(transitivity)} \\
\frac{A \leq B \land B \leq C \land A = C}{A \leq B \land B \leq C \land A = C} & \quad \text{(antisymmetry)}
\end{align*}
\]
Online Algorithm

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

Monotonicity and Incrementality

If \[ G \rightarrow G' \]
then \[ G \land C \rightarrow G' \land C \]

Applicable rules cannot become inapplicable in larger context.

\[
\begin{align*}
A \leq B & \land B \leq C & \land C \leq A & \quad \text{(transitivity)} \\
A \leq B & \land B \leq C & \land A \leq C & \land C \leq A & \quad \text{(antisymmetry)} \\
A \leq B & \land B \leq C & \land A = C \\
\end{align*}
\]

\[ \ldots \]
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.

If $A \rightarrow B$ and $C \rightarrow D$ then $A \land C \rightarrow B \land D$

\[
\begin{align*}
A \leq B \land B \leq C \quad &\implies \quad A \leq C \\
&
\end{align*}
\]
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.

If \( A \longrightarrow B \) and \( C \longrightarrow D \) then \( A \land C \longrightarrow B \land D \)

\[
\begin{align*}
A \leq B & \land B \leq C \\
\downarrow & \\
A \leq B & \land B \leq C & \land & A \leq C \\
\downarrow & \\
A \leq B & \land B \leq C & \land & A \leq C \\
\downarrow & \\
C \leq D & \land D \leq A \\
\downarrow & \\
C \leq D & \land D \leq A & \land & C \leq A \\
\downarrow & \\
\ldots & A = C & \ldots
\end{align*}
\]
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.

If $A \longrightarrow B$ and $C \longrightarrow D$ then $A \land C \longrightarrow B \land D$

$A \leq B \land B \leq C \land A \leq C \land C \leq D \land D \leq A \land C \leq A \land A = C \ldots$

Thom Frühwirth
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.

If \( A \xrightarrow{} B \) and \( C \xrightarrow{} D \) then \( A \land C \xrightarrow{} B \land D \)

\[
\begin{align*}
A \leq B \land B \leq C & \land C \leq D \land D \leq A \\
A \leq B \land B \leq C \land A \leq C & \land C \leq D \land D \leq A \land C \leq A \\
\therefore A = C
\end{align*}
\]
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.

\[
\begin{align*}
\text{If } & A \land E \quad \longrightarrow \quad B \land E \\
\text{and } & C \land E \quad \longrightarrow \quad D \land E \\
\text{then } & A \land C \land E \quad \longrightarrow \quad B \land D \land E
\end{align*}
\]

\[
\begin{align*}
A \leq B & \land \quad B \leq C & \land \quad C \leq A \\
A \leq B \land A \leq C & \land \quad C \leq A \land B \leq A \\
\ldots & A = C \ldots
\end{align*}
\]
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.

If \( A \land E \rightarrow B \land E \) and \( C \land E \rightarrow D \land E \), then \( A \land C \land E \rightarrow B \land D \land E \)

\[
\begin{align*}
A \leq B & \land B \leq C & \land C \leq A \\
\downarrow & & \downarrow \\
A \leq B \land A \leq C & \land B \leq C & \land C \leq A \land B \leq A \\
\downarrow & & \\
\ldots & & A = C & \ldots
\end{align*}
\]

Introduces CHR machines, analogous to RAM and Turing machines. These machines can simulate each other in polynomial time.

⇒ CHR is Turing-complete
⇒ 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]
A *ranking* || maps terms into natural numbers. For all simplification rules

\[ H_1 \land \ldots \land H_n \Leftrightarrow C \mid D \land B_1 \land \ldots \land B_m \]

it holds that

\[ C \land D \rightarrow |H_1| + \ldots + |H_n| > |B_1| + \ldots + |B_m| \]

For all propagation rules

\[ H_1 \land \ldots \land H_n \Rightarrow C \mid D \land B_1 \land \ldots \land B_m \]

it holds that

\[ C \land D \rightarrow |H_i| > |B_j| \text{ for all } i, j \]

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

[Frühwirth, KR’02]
For each rule, there is a minimal, most general state to which it is applicable.

Rule: \( H \iff C \mid B \) or \( H \Rightarrow C \mid B \)

Minimal State: \( H \land C \)

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: \[ H \iff C | B \] or \[ H \Rightarrow C | B \]

Minimal State: \[ H \land C \]

Every other state to which the rule is applicable contains the minimal state (cf. Monotonicity/Incrementality).
For each rule, there is a minimal, most general state to which it is applicable.

Rule: \( H \iff C \mid B \) or \( H \Rightarrow C \mid B \)

Minimal State: \( H \land C \)

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.

\[
\begin{align*}
A & \mapsto B \\
A & \mapsto C \\
\hline
B & \mapsto^* D \\
C & \mapsto^* D
\end{align*}
\]

⇒ Independence from the order in which constraints processed.
⇒ 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.

\[ \begin{align*}
X \leq X & \iff true \quad \text{(reflexivity)} \\
X \leq Y \land Y \leq X & \iff X = Y \quad \text{(antisymmetry)}
\end{align*} \]

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.

\[ X \leq X \iff \text{true} \]  
(reflexivity)

\[ X \leq Y \land Y \leq X \iff X = Y \]  
(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.

\[ X \leq X \iff \text{true} \quad \text{(reflexivity)} \]
\[ X \leq Y \land Y \leq X \iff X = Y \quad \text{(antisymmetry)} \]

Start from overlapping minimal states

\[
\begin{align*}
A &\leq A \land A \leq A \\
& \text{reflexivity} & \text{antisymmetry} \\
A &\leq A \\
& \text{reflexivity} & \text{true} \\
& \text{built-in} & A = A \\
& \text{true}
\end{align*}
\]
Completion

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

\[ X \leq Y \land Y \leq X \iff X = Y \quad \text{(antisymmetry)} \]
\[ X \leq Y \land Y < X \iff \text{false} \quad \text{(inconsistency)} \]

\[ A \leq B \land B \leq A \land B < A \]

antisymmetry

\[ A = B \land B < A \]
\[ A = B \land A < A \]
\[ \text{false} \]

inconsistency

\[ B \leq A \land \text{false} \]
\[ \text{false} \]

\[ X < X \iff \text{false} \quad \text{(irreflexivity)} \]
Completion

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

\[ X \leq Y \land Y \leq X \iff X = Y \quad \text{(antisymmetry)} \]
\[ X \leq Y \land Y < X \iff false \quad \text{(inconsistency)} \]

\[ A \leq B \land B \leq A \land B < A \]

antisymmetry
\[ A = B \land B < A \]
\[ A = B \land A < A \]
\[ false \]

inconsistency
\[ B \leq A \land false \]
\[ false \]

\[ X < X \iff false \quad \text{(irreflexivity)} \]
Derive rules from a non-joinable critical pair for transition from one of the critical states into the other one.

\[
\begin{align*}
X \leq Y \land Y \leq X & \iff X = Y \quad \text{(antisymmetry)} \\
X \leq Y \land Y < X & \iff false \quad \text{(inconsistency)}
\end{align*}
\]

\[
\begin{align*}
A \leq B \land B \leq A \land B < A \\
\text{antisymmetry} & \iff A = B \land B < A \\
\text{inconsistency} & \iff B \leq A \land false
\end{align*}
\]

\[
\begin{align*}
A = B \land A < A \\
false
\end{align*}
\]

\[
X < X \iff false \quad \text{(irreflexivity)}
\]
Completion

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

\[ X \leq Y \land Y \leq X \iff X = Y \quad \text{(antisymmetry)} \]
\[ X \leq Y \land Y < X \iff \text{false} \quad \text{(inconsistency)} \]

\[
\begin{align*}
A \leq B & \land B \leq A \land B < A \\
\text{antisymmetry} & \quad \text{inconsistency} \\
A = B & \land B < A \\
\quad & \\
A = B & \land A < A \\
\quad & \\
B \leq A & \land \text{false} \\
\quad & \\
\text{false} & \\
\quad & \\
X < X & \iff \text{false} \quad \text{(irreflexivity)}
\end{align*}
\]
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.

\[
P_1 \quad \max(X, Y, Z) \Leftrightarrow X < Y \mid Z = Y.
\]
\[
\max(X, Y, Z) \Leftrightarrow X \geq Y \mid Z = X.
\]

\[
P_2 \quad \max(X, Y, Z) \Leftrightarrow X \leq Y \mid Z = Y.
\]
\[
\max(X, Y, Z) \Leftrightarrow X > Y \mid Z = X.
\]
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.

\[ P_1 \quad \max(X, Y, Z) \Leftrightarrow X < Y \mid Z = Y. \]
\[ \max(X, Y, Z) \Leftrightarrow X \geq Y \mid Z = X. \]

\[ P_2 \quad \max(X, Y, Z) \Leftrightarrow X \leq Y \mid Z = Y. \]
\[ \max(X, Y, Z) \Leftrightarrow X > Y \mid Z = X. \]
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.

\[ P_1 \]
\[ \max(X, Y, Z) \iff X < Y \mid Z = Y. \]
\[ \max(X, Y, Z) \iff X \geq Y \mid Z = X. \]

\[ P_2 \]
\[ \max(X, Y, Z) \iff X \leq Y \mid Z = Y. \]
\[ \max(X, Y, Z) \iff X > Y \mid Z = X. \]

\[ \max(X, Y, Z) \land X \geq Y \]
\[ \downarrow P_1 \]
\[ Z = X \land X \geq Y \]
\[ \max(X, Y, Z) \land X \geq Y \]
\[ \downarrow P_2 \]
\[ Z = X \land X \geq Y \]
Local consistency algorithm simplifies one atomic Boolean constraint at a time into syntactic equalities.

\[
\begin{align*}
    \text{and}(X, X, Z) & \iff X = Z. \\
    \text{and}(X, Y, 1) & \iff X = 1 \land Y = 1. \\
    \text{and}(X, 1, Z) & \iff X = Z. \\
    \text{and}(X, 0, Z) & \iff Z = 0. \\
    \text{and}(1, Y, Z) & \iff Y = Z. \\
    \text{and}(0, Y, Z) & \iff Z = 0. \\
    \text{imp}(0, X) & \iff \text{true}. \\
    \text{imp}(X, 0) & \iff X = 0. \\
    \text{imp}(1, X) & \iff X = 1. \\
    \text{imp}(X, 1) & \iff \text{true}. 
\end{align*}
\]
Bridge rules relate constraints from different programs for their cooperation and communication.

\[ \text{and}(X, Y, X) \iff \text{imp}(X, Y). \]

Non-confluent: E.g.

\[
\begin{array}{c}
\text{and}(X, X, X) \\
\text{true} \quad \rightarrow \\
\text{imp}(X, X) \\
\end{array}
\]

Completion adds the rules:

\[
\begin{align*}
\text{imp}(X, X) & \iff \text{true}. \\
\text{imp}(X, Y) \land \text{imp}(X, Y) & \iff \text{imp}(X, Y). \\
\text{imp}(X, Y) \land \text{and}(X, Y, Z) & \iff \text{imp}(X, Y) \land X=Z.
\end{align*}
\]
Solver Union and Cooperation by Completion

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

\[ \text{and}(X, Y, X) \Leftrightarrow \text{imp}(X, Y). \]

Non-confluent: E.g.

\[ \begin{align*}
\text{and}(X, X, X) & \quad \text{true} \\
& \quad \text{imp}(X, X)
\end{align*} \]

Completion adds the rules:

\[ \begin{align*}
\text{imp}(X, X) & \Leftrightarrow \text{true}. \\
\text{imp}(X, Y) \land \text{imp}(X, Y) & \Leftrightarrow \text{imp}(X, Y). \\
\text{imp}(X, Y) \land \text{and}(X, Y, Z) & \Leftrightarrow \text{imp}(X, Y) \land X=Z.
\end{align*} \]
Solver Union and Cooperation by Completion

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

\[ \text{and}(X, Y, X) \iff \text{imp}(X, Y). \]

Non-confluent: E.g.

\[ \text{and}(X, X, X) \]

\[ \text{true} \]

\[ \text{imp}(X, X) \]

**Completion** adds the rules:

\[ \text{imp}(X, X) \iff \text{true}. \]

\[ \text{imp}(X, Y) \land \text{imp}(X, Y) \iff \text{imp}(X, Y). \]

\[ \text{imp}(X, Y) \land \text{and}(X, Y, Z) \iff \text{imp}(X, Y) \land X=Z. \]
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., $\neg x \lor y \lor z$ as $cl([-x,+y,+z])$.

```plaintext
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).
```

Thom Frühwirth
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., $\neg x \lor y \lor z$ as $\text{cl}([-x,+y,+z])$.

- empty_clause @ $\text{cl}([]) \Leftrightarrow \text{false}$.
- tautology @ $\text{cl}(L) \Leftrightarrow \text{in}(+X,L) \land \text{in}(-X,L) \mid \text{true}$.

- resolution @ $\text{cl}(L1) \land \text{cl}(L2) \Rightarrow$
  
  $\text{find}(+X,L1,L3) \land \text{find}(-X,L2,L4) \mid$
  
  $\text{merge}(L3,L4,L) \land$
  
  $\text{cl}(L)$. 
Linear Polynomial Equations

**Equations** $a_1 x_1 + \ldots + a_n x_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**.

\[
A1 \times X + P1 = 0 \land XP = 0 \iff \\
\text{find}(A2 \times X, XP, P2) \mid \\
\text{compute}(P2 - (P1 / A1) \times A2, P3) \land \\
A1 \times X + P1 = 0 \land P3 = 0.
\]

\[
B = 0 \iff \text{number}(B) \mid \text{zero}(B).
\]

\[
1 \times X + 3 \times Y + 5 = 0 \land 3 \times X + 2 \times Y + 8 = 0 \\
\text{compute}((2 \times Y + 8) - ((3 \times Y + 5) / 1) \times 3, P3) \quad \% \ P3 = -7 \times Y + -7 \\
1 \times X + 3 \times Y + 5 = 0 \land -7 \times Y + -7 = 0 \quad \% \ Y = -1 \\
\text{compute}((1 \times X + 5) - ((-7) / -7) \times 3, P3') \quad \% \ P3' = 1 \times X + 2 \\
1 \times X + 2 = 0 \land -7 \times Y + -7 = 0 \quad \% \ X = -2
\]
Linear Polynomial Equations

**Equations** \( a_1 x_1 + \ldots + a_n x_n + b = 0 \), \( a_i \neq 0, x_i \text{ ordered} \).

**Solved form:** leftmost variable of each equation occurs only once. Reach solved normal form by **variable elimination**.

\[
A1*X+P1=0 \land XP=0 \iff \\
\text{find}(A2*X,XP,P2) \land \\
\text{compute}(P2-(P1/A1)*A2,P3) \land \\
A1*X+P1=0 \land P3=0.
\]

\[
B=0 \iff \text{number}(B) \mid \text{zero}(B).
\]

\[
1*X+3*Y+5=0 \land 3*X+2*Y+8=0
\]

\[
\text{compute}((2*Y+8) - ((3*Y+5)/1)*3,P3) \% P3=-7*Y+ -7
\]

\[
1*X+3*Y+5=0 \land -7*Y+ -7=0 \% Y=-1
\]

\[
\text{compute}((1*X+5) - ((-7)/-7)*3,P3') \% P3'=1*X+2
\]

\[
1*X+2=0 \land -7*Y+ -7=0 \% X=-2
\]
Linear Polynomial Equations

Equations $a_1 x_1 + \ldots + a_n x_n + b = 0$, $a_i \neq 0$, $x_i, ordered$.

Solved form: leftmost variable of each equation occurs only once. Reach solved normal form by variable elimination.

\begin{align*}
A1*X+P1&=0 \land XP=0 \iff \\
&\text{find}(A2*X,XP,P2) \mid \\
&\text{compute}(P2-(P1/A1)*A2,P3) \land \\
&A1*X+P1=0 \land P3=0.
\end{align*}

\begin{align*}
B=0 \iff \text{number}(B) \mid \text{zero}(B).
\end{align*}

\begin{align*}
1*X+3*Y+5&=0 \land 3*X+2*Y+8=0 \\
&\text{compute}((2*Y+8)-((3*Y+5)/1)*3,P3) \% P3=-7*Y+ -7 \\
1*X+3*Y+5&=0 \land -7*Y+-7=0 \% Y=-1 \\
&\text{compute}((1*X+5)-((-7)/-7)*3,P3') \% P3'=1*X+2 \\
1*X+2&=0 \land -7*Y+-7=0 \% X=-2
\end{align*}
Linear Polynomial Equations

Equations $a_1x_1 + \ldots + a_nx_n + b = 0$, $a_i \neq 0, x_i \text{ordered}$.  
Solved form: leftmost variable of each equation occurs only once.  
Reach solved normal form by variable elimination.

\[
A1*X+P1=0 \land XP=0 \iff \\
\text{find}(A2*X,XP,P2) \mid \\
\text{compute}(P2-(P1/A1)*A2,P3) \land \\
A1*X+P1=0 \land P3=0.
\]

B=0 $\iff$ number(B) $\mid$ zero(B).

\[
1*X+3*Y+5=0 \land 3*X+2*Y+8=0 \\
\text{compute}((2*Y+8) - ((3*Y+5)/1)*3,P3) \% P3=-7*Y+ -7 \\
1*X+3*Y+5=0 \land -7*Y+ -7=0 \% Y=-1 \\
\text{compute}((1*X+5) - ((-7)/-7)*3,P3') \% P3' = 1*X+2 \\
1*X+2=0 \land -7*Y+ -7=0 \% X=-2
\]
Fourier’s Algorithm

\[ A1 \cdot X + P1 \geq 0 \land XP \geq 0 \Rightarrow \]
\[ \text{find}(A2 \cdot X, XP, P2) \land \text{opposite\_sign}(A1, A2) \]
\[ \text{compute}(P2 - (P1/A1) \cdot A2, P3) \land \]
\[ P3 \geq 0. \]

\[ B \geq 0 \iff \text{number}(B) \mid \text{non\_negative}(B). \]
Solver Cooperation, Combination of Algorithms

**Gaussian Elimination for \(=\)**

\[
A_1 \cdot X + P_1 = 0 \land X \cdot P = 0 \iff
\text{find}(A_2 \cdot X, X \cdot P, P_2) \land
\text{compute}(P_2 - (P_1/A_1) \cdot A_2, P_3) \land A_1 \cdot X + P_1 = 0 \land P_3 = 0.
\]

**Fouriers Algorithm for \(\geq\)**

\[
A_1 \cdot X + P_1 \geq 0 \land X \cdot P \geq 0 \Rightarrow
\text{find}(A_2 \cdot X, X \cdot P, P_2) \land \text{opposite\_sign}(A_1, A_2) \land
\text{compute}(P_2 - (P_1/A_1) \cdot A_2, P_3) \land P_3 \geq 0.
\]

**Bridge Rule for \(= \) and \(\geq\)**

\[
A_1 \cdot X + P_1 = 0 \land X \cdot P \geq 0 \iff
\text{find}(A_2 \cdot X, X \cdot P, P_2) \land
\text{compute}(P_2 - (P_1/A_1) \cdot A_2, P_3) \land A_1 \cdot X + P_1 = 0 \land P_3 \geq 0.
\]
Solver Cooperation, Combination of Algorithms

**Gaussian Elimination for** \( = \)

\[
A1\times X + P1 = 0 \land XP = 0 \iff \\
\quad \text{find}(A2\times X, XP, P2) \ |
\quad \text{compute}(P2 - (P1/A1)\times A2, P3) \land A1\times X + P1 = 0 \land P3 = 0.
\]

**Fouriers Algorithm for** \( \geq \)

\[
A1\times X + P1 \geq 0 \land XP \geq 0 \Rightarrow \\
\quad \text{find}(A2\times X, XP, P2) \land \text{opposite_sign}(A1, A2) \ |
\quad \text{compute}(P2 - (P1/A1)\times A2, P3) \land P3 \geq 0.
\]

**Bridge Rule for** \( = \) and \( \geq \)

\[
A1\times X + P1 = 0 \land XP \geq 0 \iff \\
\quad \text{find}(A2\times X, XP, P2) \ |
\quad \text{compute}(P2 - (P1/A1)\times A2, P3) \land A1\times X + P1 = 0 \land P3 \geq 0.
\]
Solver Cooperation, Combination of Algorithms

Gaussian Elimination for =

\[ A1 \cdot X + P1 = 0 \land XP = 0 \iff \]
\[ \text{find}(A2 \cdot X, XP, P2) \land \]
\[ \text{compute}(P2 - (P1/A1) \cdot A2, P3) \land A1 \cdot X + P1 = 0 \land P3 = 0. \]

Fouriers Algorithm for ≥

\[ A1 \cdot X + P1 \geq 0 \land XP \geq 0 \Rightarrow \]
\[ \text{find}(A2 \cdot X, XP, P2) \land \text{opposite\_sign}(A1, A2) \land \]
\[ \text{compute}(P2 - (P1/A1) \cdot A2, P3) \land P3 \geq 0. \]

Bridge Rule for = and ≥

\[ A1 \cdot X + P1 = 0 \land XP \geq 0 \iff \]
\[ \text{find}(A2 \cdot X, XP, P2) \land \]
\[ \text{compute}(P2 - (P1/A1) \cdot A2, P3) \land A1 \cdot X + P1 = 0 \land P3 \geq 0. \]
**Syntactic Unification**

**Rational tree** (possibly infinite) tree with finite set of subtrees, e.g. $X = f(X)$.

**Solved normal form** $X_1 = t_1 \land \ldots \land X_n = t_n$ ($n \geq 0$)

where $X_i$ is different to $X_j$ and $t_j$, if $i \leq j$

- **reflexivity** @ $X=X \iff \text{var}(X) \mid \text{true}$.
- **orientation** @ $T=X \iff \text{var}(X) \land X@<T \mid X=T$.
- **decomposition** @ $T1=T2 \iff \text{nonvar}(T1) \land \text{nonvar}(T2) \mid \text{same_functor}(T1,T2) \land 
    \text{same_args}(T1,T2)$.
- **confrontation** @ $X=T1 \land X=T2 \iff \text{var}(X) \land X@<T1 \land T1@=<T2 \mid X=T1 \land T1=T2$. 
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))

 makeover decomposition makeover unity

h(f(U),Y,g(h(Y),U))

 makeover orientation

Y=f(U) ∧ Y=f(a) ∧ g(X,a)=g(h(Y),U)

 makeover confrontation

Y=f(U) ∧ U=a ∧ X=h(Y) ∧ a=a

 makeover decomposition makeover unity

Y=f(U) ∧ U=a ∧ X=h(Y)
Syntactic Unification II

reflexivity  @  X=X ⇔ \text{var}(X) \mid \text{true}.

orientation  @  T=X ⇔ \text{var}(X) \land X@<T \mid X=T.

decomposition  @  T1=T2 ⇔ \text{nonvar}(T1) \land \text{nonvar}(T2) \mid

\quad \text{same_functor}(T1,T2) \land

\quad \text{same_args}(T1,T2).

confrontation  @  X=T1 \land X=T2 ⇔ \text{var}(X) \land X@<T1 \land T1@=<T2 \mid

\quad X=T1 \land T1=T2.

\[ h(Y,f(a),g(X,a)) = h(f(U),Y,g(h(Y),U)) \]

\[ \xrightarrow{\text{decomposition}} \]

\[ Y=f(U) \land f(a)=Y \land g(X,a)=g(h(Y),U) \]

\[ \xrightarrow{\text{orientation}} \]

\[ Y=f(U) \land Y=f(a) \land g(X,a)=g(h(Y),U) \]

\[ \xrightarrow{\text{confrontation}} \]

\[ Y=f(U) \land U=a \land X=h(Y) \land U=a \]

\[ \xrightarrow{\text{decomposition}} \]

\[ Y=f(U) \land U=a \land X=h(Y) \land a=a \]

\[ \xrightarrow{\text{decomposition}} \]

\[ Y=f(U) \land U=a \land X=h(Y) \]
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))

\[ \begin{align*}
\rightarrow \text{decomposition} & \quad \rightarrow^{*}
Y &= f(U) \land f(a) = Y \land g(X,a) = g(h(Y),U) \\
\rightarrow \text{orientation} & \\
\rightarrow^{*} & \\
\rightarrow \text{confrontation} & \\
\rightarrow \text{decomposition} & \quad \rightarrow^{*}
U &= a \land X = h(Y) \land U = a \\
Y &= f(U) \land U = a \land X = h(Y) \land a = a \\
Y &= f(U) \land U = a \land X = h(Y) 
\end{align*} \]
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))
\[ \xrightarrow{decomposition} \]
\[ Y=f(U) \land f(a)=Y \land g(X,a)=g(h(Y),U) \]
\[ \xrightarrow{orientation} \]
\[ Y=f(U) \land Y=f(a) \land g(X,a)=g(h(Y),U) \]
\[ \xrightarrow{*} \]
\[ Y=f(U) \land U=a \land X=h(Y) \land U=a \]
\[ \xrightarrow{confrontation} \]
\[ Y=f(U) \land U=a \land X=h(Y) \land a=a \]
\[ \xrightarrow{decomposition} \]
\[ Y=f(U) \land U=a \land X=h(Y) \land U=a \]
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 @ \( X=X \iff \text{var}(X) \mid \text{true} \).

orientation @ \( T=X \iff \text{var}(X) \land X@<T \mid X=T \).

decomposition @ \( T1=T2 \iff \text{nonvar}(T1) \land \text{nonvar}(T2) \mid \text{same}\_functor}(T1,T2) \land \text{same}\_args(T1,T2) \).

confrontation @ \( X=T1 \land X=T2 \iff \text{var}(X) \land X@<T1 \land T1@=<T2 \mid X=T1 \land T1=T2 \).

\[
\begin{align*}
\text{thyme}(Y,f(a),g(X,a)) &= \text{h}(f(U),Y,g(h(Y),U)) \\
\iff & \downarrow \text{decomposition} \iff \ast \\
& Y = f(U) \land f(a) = Y \land g(X,a) = g(h(Y),U) \\
\iff & \downarrow \text{orientation} \\
& Y = f(U) \land Y = f(a) \land g(X,a) = g(h(Y),U) \\
\iff & \downarrow \ast \\
& Y = f(U) \land U = a \land X = h(Y) \land U = a \\
\iff & \downarrow \text{confrontation} \\
& Y = f(U) \land U = a \land X = h(Y) \land a = a \\
\iff & \downarrow \text{decomposition} \iff \ast \\
& Y = f(U) \land U = a \land X = h(Y)
\end{align*}
\]
Syntactic Unification II

reflexivity @ \( X=X \) ⇔ \( \text{var}(X) \mid \text{true} \).

orientation @ \( T=X \) ⇔ \( \text{var}(X) \land X\@<T \mid X=T \).

decomposition @ \( T_1=T_2 \) ⇔ \( \text{nonvar}(T_1) \land \text{nonvar}(T_2) \mid \)
\[
\text{same_functor}(T_1,T_2) \land \\
\text{same_args}(T_1,T_2).
\]

confrontation @ \( X=T_1 \land X=T_2 \) ⇔ \( \text{var}(X) \land X\@<T_1 \land T_1\@=<T_2 \mid \\
X=T_1 \land T_1=T_2 \).

\[
\begin{align*}
Y=f(U) & \land f(a)=Y \land g(X,a)=g(h(Y),U) \\
Y=f(U) & \land Y=f(a) \land g(X,a)=g(h(Y),U) \\
Y=f(U) & \land U=a \land X=h(Y) \land U=a \\
Y=f(U) & \land U=a \land X=h(Y) \land a=a \\
Y=f(U) & \land U=a \land X=h(Y)
\end{align*}
\]
Finite Domains / Interval Arithmetic

\[ X :: A..B \implies A \leq B \]
\[ X :: A1..B1 \land X :: A2..B2 \iff \max(A1, A2, A) \land \min(B1, B2, B) \land X :: A..B \]
\[ X \leq Y \land X :: AX..BX \land Y :: AY..BY \implies BX > BY \mid X :: AX..BY \]
\[ X \leq Y \land X :: AX..BX \land Y :: AY..BY \implies AX > AY \mid Y :: AX..BY \]

\[ A :: 2..3 \land B :: 1..2 \land A \leq B \]
\[ \Downarrow \quad A :: 2..3 \land B :: 1..2 \land A \leq B \land A :: 2..2 \]
\[ \Downarrow \quad B :: 1..2 \land A \leq B \land A :: 2..2 \]
\[ \Downarrow \quad B :: 1..2 \land A \leq B \land A :: 2..2 \land B :: 2..2 \]
\[ \Downarrow \quad A \leq B \land A :: 2..2 \land B :: 2..2 \]
Scheduling

Taming Disjunction. [M. Wallace, Eclipse User Manual]

Two tasks cannot be done at the same time on a single resource.  
⇒ 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.

\[
\begin{align*}
\text{chrTaskResource} (S1, E1, R1, S2, E2, R2) & \iff \\
R1 & = R2, \ E1 > S2 \ | \ E2 <= S1. \\
\text{chrTaskResource} (S1, E1, R1, S2, E2, R2) & \iff \\
R1 & = R2, \ E2 > S1 \ | \ E1 <= S2. \\
\text{chrTaskResource} (S1, E1, R1, S2, E2, R2) & \iff \\
E1 & > S2, \ E2 > S1 \ | \ R1 <> R2.
\end{align*}
\]

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 \( \text{lex} \) constraint

- Non-incremental pointer-based imperative pseudo code, 45 lines
  \textit{Frisch/Hnich/Kiziltan et. al., CP 2002.}

- Finite automata, interpreted, 16 lines
  \textit{Beldiceanu/Carlsson/Petit, CP 2004.}

Non-intuitive code. Hard to analyse.

\[
\begin{align*}
\text{l}_1 \leq_{\text{lex}} \text{l}_2 & \iff (\text{l}_1 = [] \land \text{l}_2 = []) \lor \\
& (\text{l}_1 = [x|\text{l}_1'] \land \text{l}_2 = [y|\text{l}_2'] \land x < y) \lor \\
& (\text{l}_1 = [x|\text{l}_1'] \land \text{l}_2 = [y|\text{l}_2'] \land x = y \land \text{l}_1' \leq_{\text{lex}} \text{l}_2')
\end{align*}
\]
Lexicographic Order Constraint Solver

\[
\begin{align*}
[\ ] \ lex \ [\ ] & \iff \text{true.} \\
[X|L_1] \ lex \ [Y|L_2] & \iff X < Y \ | \ \text{true.} \\
[X|L_1] \ lex \ [Y|L_2] & \iff X = Y \ | \ L_1 \ lex \ L_2. \\
[X|L_1] \ lex \ [Y|L_2] & \implies X = < Y. \\
[X,U|L_1] \ lex \ [Y,V|L_2] & \iff U > V \ | \ X < Y. \\
[X,U|L_1] \ lex \ [Y,V|L_2] & \iff U = V, \ L_1 = [\_|\_] \ | \\
& \quad [X,U] \ lex \ [Y,V], \ [X|L_1] \ lex \ [Y|L_2].
\end{align*}
\]

Executable specification: short, concise
using recursive decomposition and propagation
Lexicographic Order Constraint Solver

\[
[] \text{ lex } [] \iff \text{true.}
\]
\[
[X|L1] \text{ lex } [Y|L2] \iff X < Y \lor \text{true.}
\]
\[
[X|L1] \text{ lex } [Y|L2] \iff X = Y \land L1 \text{ lex } L2.
\]
\[
[X|L1] \text{ lex } [Y|L2] \implies X = \leq Y.
\]
\[
[X,U|L1] \text{ lex } [Y,V|L2] \iff U > V \lor X < Y.
\]
\[
[X,U|L1] \text{ lex } [Y,V|L2] \iff U = V, L1 = [\_|\_] \lor
\]
\[
[X,U] \text{ lex } [Y,V], [X|L1] \text{ lex } [Y|L2].
\]

**Incremental** and concurrent: by nature of CHR

**Efficient**: Optimal linear worst-case time complexity
Lexicographic Order Constraint Solver

\[
[] \text{ lex } [] \iff \text{true.}
\]
\[
[X|L1] \text{ lex } [Y|L2] \iff X<Y \text{ or } \text{true.}
\]
\[
[X|L1] \text{ lex } [Y|L2] \iff X=Y \text{ and } L1 \text{ lex } L2.
\]
\[
[X|L1] \text{ lex } [Y|L2] \implies X=<Y.
\]
\[
[X,U|L1] \text{ lex } [Y,V|L2] \iff U>V \text{ and } X<Y.
\]
\[
[X,U|L1] \text{ lex } [Y,V|L2] \iff U>=V, L1=\_\_ \text{ and }
\]
\[
[X,U] \text{ lex } [Y,V], [X|L1] \text{ lex } [Y|L2].
\]

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.


**Confluence**: proven by CHR confluence checker
**Correctness**: shown by standard CHR analysis
Part II

...Around the World

4. Language Issues
   - Implementations
   - More Semantics
   - Program Generation and Transformation
   - Language Extensions

5. From Computational Logic to Logical Computations

6. Classical Applications

7. Trends in Applications
   - Spatio-Temporal Reasoning
   - Agents and Actions
   - Types and Security
   - Testing and Verification
   - Semantic Web
   - Computational Linguistics

8. Application Projects
   - JMMSSolve Java Memory Machine
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
- **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
- **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

Hard Core CHR People

Slim Abdennadher

Tom Schrijvers

Peter Stuckey

Christian Holzbaur

Armin Wolf
More Semantics


  
  ```
  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.
Example Rule Generation

Intensional definition of minimum:

\[
\begin{align*}
\text{min}(A, B, C) & \leftarrow A \leq B, \quad C = A. \\
\text{min}(A, B, C) & \leftarrow B \leq A, \quad C = B.
\end{align*}
\]

Automatically generate propagation rules:

\[
\begin{align*}
\text{min}(A, B, C) & \Rightarrow C \leq A, \quad C \leq B. \\
\text{min}(A, B, C), \quad C \neq B & \Rightarrow C = A. \\
\text{min}(A, B, C), \quad C \neq A & \Rightarrow C = B. \\
\text{min}(A, B, C), \quad B \leq A & \Rightarrow C = B. \\
\text{min}(A, B, C), \quad A \leq B & \Rightarrow C = A.
\end{align*}
\]
Example Rule Generation

Intensional definition of minimum:

\[
\begin{align*}
\text{min}(A, B, C) & \leftarrow A \leq B, \quad C = A. \\
\text{min}(A, B, C) & \leftarrow B \leq A, \quad C = B.
\end{align*}
\]

Derived constraint handling rules:

\[
\begin{align*}
\text{min}(A, B, C) & \Rightarrow C \leq A \land C \leq B. \\
\text{min}(A, B, C) & \Leftrightarrow C \neq B \mid C = A. \\
\text{min}(A, B, C) & \Leftrightarrow C \neq A \mid C = B. \\
\text{min}(A, B, C) & \Leftrightarrow B \leq A \mid C = B. \\
\text{min}(A, B, C) & \Leftrightarrow A \leq B \mid C = A.
\end{align*}
\]
Program Transformation

- **Specialization of Concurrent Guarded Multi-Set Transformation Rules**, T. Frühwirth, LOPSTR 2004, LNCS.
  *Multiple heads make partial evaluation hard.*

  *Are termination and confluence modular?*

  *To implement extensions and optimizations.*
Union of Constraint Programs

**Well-Behaved CHR program**: terminating and confluent.

**Modularity**: Is well-Behavedness preserved?

- \( P_1 = \{a \iff b\} \)
- \( P_2 = \{b \iff a\} \)
  The union of \( P_1 \) and \( P_2 \) is non-terminating.
  Remedy: Turn into propagation rules or rename constraints.

- \( P_1 = \{a \iff b\} \)
- \( P_3 = \{a \iff c\} \)
  The union of \( P_1 \) and \( P_3 \) is non-confluent.
  Remedy: Completion adds \( \{b \iff c\} \).
Union of Constraint Programs

*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.

- $P_3 = \{a \leftrightarrow c\}$
  
The union of $P_1$ and $P_3$ is non-confluent.
  Remedy: Completion adds $\{b \leftrightarrow c\}$. 
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 \( \{b \Leftrightarrow c\} \).

E.g. Fuzzy Constraints:

\[ X \leq Y : A, \ Y \leq Z : B \implies X \leq Z : A \ast B \]

\[ E \leq F : 0.5, \ F \leq G : 0.4 \implies E \leq F : 0.5, \ F \leq G : 0.4, \ E \leq G : 0.2 \]
Randomized Algorithms


Random walk

\[
\text{walked}_\text{to}(0) \leftrightarrow \text{true}
\]

\[
\text{walked}_\text{to}(N) \leftrightarrow 0.5 \cdot \text{walked}_\text{to}(N+1)
\]

\[
\text{walked}_\text{to}(N) \leftrightarrow 0.5 \cdot \text{walked}_\text{to}(N-1)
\]

*Probabilistic termination.*
Reasoning Services


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

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.

• 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\(^{\lor}\) program for append of two lists

\[
\text{append}(X,Y,Z) \Leftrightarrow \left( X=[], Y=L \land Z=L \lor X=[H|L1] \land Y=L2 \land Z=[H|L3] \land \text{append}(L1,L2,L3) \right).
\]

can be improved by adding the following rule

\[
\text{append}(X,[],Z) \Leftrightarrow X = Z.
\]
Don’t-care and Don’t-know Nondeterminism

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

\[
\text{append}(X,Y,Z) \Leftrightarrow \\
\quad \left( \begin{array}{l}
X=[] \wedge Y=L \wedge Z=L \\
\lor \quad X=[H\vert L_1] \wedge Y=L_2 \wedge Z=[H\vert L_3] \wedge \text{append}(L_1, L_2, L_3) 
\end{array} \right).
\]

can be improved by adding the following rule

\[
\text{append}(X, [], Z) \Leftrightarrow X = Z.
\]
Don’t-care and Don’t-know Nondeterminism

The CHR\(^\vee\) program for append of two lists

\[
\text{append}(X,Y,Z) \Leftrightarrow (X=[] \land Y=L \land Z=L \\
\lor X=[H|L1] \land Y=L2 \land Z=[H|L3] \land \text{append}(L1, L2, L3)).
\]

can be improved by adding the following rule

\[
\text{append}(X,[],Z) \Leftrightarrow X = Z.
\]
Bottom-up and Top-down Evaluation with Tabling

$\text{fib}(N,M)$ is true if $M$ is the $N$th Fibonacci number.

**Top-down Evaluation**

$\text{fib}(0,M) \iff M = 1.$

$\text{fib}(1,M) \iff M = 1.$

$\text{fib}(N,M) \iff N \geq 2 \mid \text{fib}(N-1,M1) \land \text{fib}(N-2,M2) \land M = M1 + M2.$
Bottom-up and Top-down Evaluation with Tabling

\[ \text{fib}(N,M) \ \text{is true if} \ M \ \text{is the} \ N\text{th Fibonacci number.} \]

Top-down Evaluation with Tabling

\[ \text{fib}(N,M_1) \land \text{fib}(N,M_2) \Leftrightarrow M_1 = M_2 \land \text{fib}(N,M_1). \]

\[ \text{fib}(0,M) \Rightarrow M = 1. \]
\[ \text{fib}(1,M) \Rightarrow M = 1. \]
\[ \text{fib}(N,M) \Rightarrow N \geq 2 \land \text{fib}(N-1,M_1) \land \text{fib}(N-2,M_2) \land M = M_1 + M_2. \]
fib(N,M) is true if M is the Nth Fibonacci number.

Bottom-up Evaluation

\[
\begin{align*}
\text{fib} & \iff \text{fib}(0,1) \land \text{fib}(1,1). \\
\text{fib}(N1,M1) \land \text{fib}(N2,M2) & \Rightarrow N1=N2+1 | \\
& \quad N=N1+1 \land M=M1+M2 \land \text{fib}(N,M).
\end{align*}
\]
fib(N,M) is true if M is the Nth Fibonacci number.

Bottom-up Evaluation with Termination

\[
\begin{align*}
\text{fib}(\text{Max}) & \implies \text{fib}(0,1) \land \text{fib}(1,1). \\
\text{fib}(\text{Max}) \land \text{fib}(\text{N1},\text{M1}) \land \text{fib}(\text{N2},\text{M2}) & \implies \text{Max}>\text{N1} \land \text{N1}=\text{N2}+1 \mid \\
& \quad \text{N}=\text{N1}+1 \land \text{M}=\text{M1}+\text{M2} \land \text{fib}(\text{N},\text{M}).
\end{align*}
\]
fib(N,M) is true if M is the Nth Fibonacci number.

Bottom-up Evaluation with Termination, Last Two Results Only

\[ \text{fib(Max)} \Rightarrow \text{fib}(0,1) \land \text{fib}(1,1). \]
\[ \text{fib(Max)} \land \text{fib}(N1,M1) / \text{fib}(N2,M2) \Rightarrow \text{Max}>N1 \land N1=N2+1 \mid N=N1+1 \land M=M1+M2 \land \text{fib}(N,M). \]
Abduction

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

A bird is either an albatros or a penguin.

\[
\text{bird}(X) \iff \text{albatros}(X) \lor \text{penguin}(X).
\]

Penguins can’t fly.

\[
\text{penguin}(X) \land \text{flies}(X) \iff \text{false}.
\]

The query \(\text{bird}(X) \land \text{flies}(X)\) leads to the only answer \(\text{albatros}(X) \land \text{flies}(X)\).
Abduction

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

A bird is either an albatros or a penguin.

\[
bird(X) \Leftrightarrow albatros(X) \lor penguin(X).
\]

Penguins can’t fly.

\[
penguin(X) \land flies(X) \Leftrightarrow false.
\]

The query \(bird(X) \land flies(X)\) leads to the only answer \(albatros(X) \land flies(X)\).
Bottom-up Evaluation of Logic Programs

\[ p(X, Y) \leftarrow e(X, Y). \]
\[ p(X, Y) \leftarrow e(X, Z) \land p(Z, Y). \]

is transformed into

\[ e(X, Y) \Rightarrow p(X, Y). \]
\[ e(X, Z) \land p(Z, Y) \Rightarrow p(X, Y). \]

\[ e(a, b) \land e(b, c) \land e(c, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \land p(a, d) \]
Bottom-up Evaluation of Logic Programs

\[ \begin{align*}
  p(X, Y) & \leftarrow e(X, Y). \\
  p(X, Y) & \leftarrow e(X, Z) \land p(Z, Y).
\end{align*} \]

is transformed into

\[ \begin{align*}
  e(X, Y) & \implies p(X, Y). \\
  e(X, Z) \land p(Z, Y) & \implies p(X, Y).
\end{align*} \]

\[ \begin{aligned}
  e(a, b) \land e(b, c) \land e(c, d) \\
  \downarrow \\
  e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \\
  \downarrow \\
  e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \\
  \downarrow \\
  e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \land p(a, d)
\end{aligned} \]
Bottom-up Evaluation of Logic Programs

\[ p(X, Y) \leftarrow e(X, Y). \]
\[ p(X, Y) \leftarrow e(X, Z) \land p(Z, Y). \]

is transformed into

\[ e(X, Y) \Rightarrow p(X, Y). \]
\[ e(X, Z) \land p(Z, Y) \Rightarrow p(X, Y). \]

\[ e(a, b) \land e(b, c) \land e(c, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \land p(a, d) \]
Bottom-up Evaluation of Logic Programs

\[ p(X, Y) \leftarrow e(X, Y). \]
\[ p(X, Y) \leftarrow e(X, Z) \land p(Z, Y). \]

is transformed into

\[ e(X, Y) \Rightarrow p(X, Y). \]
\[ e(X, Z) \land p(Z, Y) \Rightarrow p(X, Y). \]

\[ e(a, b) \land e(b, c) \land e(c, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \land p(a, d) \]
Bottom-up Evaluation of Logic Programs

\[
p(X, Y) \leftarrow e(X, Y).
\]

\[
p(X, Y) \leftarrow e(X, Z) \land p(Z, Y).
\]

is transformed into

\[
e(X, Y) \Rightarrow p(X, Y).
\]

\[
e(X, Z) \land p(Z, Y) \Rightarrow p(X, Y).
\]

\[
e(a, b) \land e(b, c) \land e(c, d)
\]

\[
\downarrow
\]

\[
e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d)
\]

\[
\downarrow
\]

\[
e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d)
\]

\[
\downarrow
\]

\[
e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \land p(a, d)
\]
Bottom-up Evaluation of Logic Programs

\[ p(X, Y) \leftarrow e(X, Y). \]
\[ p(X, Y) \leftarrow e(X, Z) \land p(Z, Y). \]

is transformed into

\[ e(X, Y) \Rightarrow p(X, Y). \]
\[ e(X, Z) \land p(Z, Y) \Rightarrow p(X, Y). \]

\[ e(a, b) \land e(b, c) \land e(c, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \land p(a, d) \]

Thom Frühwirth  
Constraint Handling Rules (CHR)
Bottom-up Evaluation of Logic Programs

\[
p(X, Y) \leftarrow e(X, Y).
\]
\[
p(X, Y) \leftarrow e(X, Z) \land p(Z, Y).
\]
is transformed into
\[
e(X, Y) \Rightarrow p(X, Y).
\]
\[
e(X, Z) \land p(Z, Y) \Rightarrow p(X, Y).
\]
\[
e(a, b) \land e(b, c) \land e(c, d)
\]
\[
\downarrow
\]
\[
e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d)
\]
\[
\downarrow
\]
\[
e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d)
\]
\[
\downarrow
\]
\[
e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \land p(a, d)
\]
Bottom-up Evaluation of Logic Programs

\[ p(X, Y) \leftarrow e(X, Y). \]
\[ p(X, Y) \leftarrow e(X, Z) \land p(Z, Y). \]

is transformed into

\[ e(X, Y) \Rightarrow p(X, Y). \]
\[ e(X, Z) \land p(Z, Y) \Rightarrow p(X, Y). \]

\[ e(a, b) \land e(b, c) \land e(c, d) \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \land p(a, d) \]
Bottom-up Evaluation of Logic Programs

\[ p(X, Y) \leftarrow e(X, Y). \]
\[ p(X, Y) \leftarrow e(X, Z) \land p(Z, Y). \]

is transformed into

\[ e(X, Y) \Rightarrow p(X, Y). \]
\[ e(X, Z) \land p(Z, Y) \Rightarrow p(X, Y). \]

\[ e(a, b) \land e(b, c) \land e(c, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \land p(a, d) \]
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

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

\begin{verbatim}
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;

}
\end{verbatim}
Prime Numbers Sieve


```java
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.
    }
}
```

Thom Frühwirth  Constraint Handling Rules (CHR)
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

candidates(N) \iff N>1 \mid M \text{ is } N-1, \text{factorial}(N), \text{candidates}(M).
candidates(1) \iff \text{true}.

fact @ factorial(I), \text{factorial}(J) \iff \text{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.

\[
\begin{align*}
\text{make}(a), & \quad \text{make}(b), \quad \text{union}(a,b), \quad \text{find}(b,X). \\
| \text{make} & | \text{make} & | \text{union} \\
\text{root}(a), & \quad \text{root}(b), \quad \text{find}(a,U), \quad \text{find}(b,V), \quad \text{link}(U,V), \quad \text{find}(b,X). \\
| \text{findRoot} & | \text{findRoot} & | \text{findRoot} \\
\text{root}(a), & \quad \text{root}(b), \quad U=a, \quad V=b, \quad \text{link}(U,V), \quad X=b. \\
| \text{link} \\
\text{root}(a), \quad U=a, \quad V=b, \quad V \rightarrow U, \quad X=b. 
\end{align*}
\]
Confluence for Parallelism

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

\[
\begin{align*}
\text{make}(a), \text{make}(b), \text{union}(a,b), \text{find}(b,X). \\
| \text{make} | \text{make} | \text{union} \\\n\text{root}(a), \text{root}(b), \text{find}(a,U), \text{find}(b,V), \text{link}(U,V), \text{find}(b,X). \\
| \text{findRoot} | \text{findRoot} \\\n\text{root}(a), \text{root}(b), U=a, V=b, \text{link}(U,V), \text{find}(b,X). \\
| \text{link} \\\n\text{root}(a), U=a, V=b, V \rightarrow U, \text{find}(b,X). \\
| \text{findNode} \\\n\text{root}(a), U=a, V=b, V \rightarrow U, \text{find}(a,X). \\
| \text{findRoot} \\\n\text{root}(a), U=a, V=b, V \rightarrow U, X=a.
\end{align*}
\]

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 $X \rightarrow A_i$ for each unique value $A_i$, $X$ only on left.
Ordered chain of arcs $X \rightarrow A_1$, $A_1 \rightarrow A_2$, ....
Quadratic worst-case time complexity.

sort @ $X \rightarrow A \setminus X \rightarrow B \iff A@<B \mid A \rightarrow B$.

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

Correctness: Sorted?
- $A \rightarrow B$ 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 $0 \Rightarrow A_i$ for each unique value $A_i$, left side is level. Optimal log-linear worst-case time complexity.

\[
\text{sort } @ X \rightarrow A \setminus X \rightarrow B \iff A@<B \mid A \rightarrow B.
\]

\[
\text{level@ } N \Rightarrow A, N \Rightarrow B \iff A@<B \mid M \text{ is } N+1, M \Rightarrow A, A \rightarrow B.
\]

Query $0\Rightarrow2, 0\Rightarrow5, 0\Rightarrow1, 0\Rightarrow7$.
Answer $2\Rightarrow1, 1 >2, 2 >5, 5 >7$.

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).

\[
\begin{align*}
\text{terminal} & @ A \rightarrow T \land \text{word}(T+R) \implies \text{parses}(A,T+R,R). \\
\text{non-term} & @ A \rightarrow B*C \land \text{parses}(B,I,J) \land \text{parses}(C,J,K) \implies \text{parses}(A,I,K). \\
\text{substrng} & @ \text{word}(T+R) \implies \text{word}(R).
\end{align*}
\]
T. Frühwirth, P. Brisset

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,

Most Popular Constraint-Based Internet Application.
University Course Timetabling

S. Abdennadher, M. Saft, S. Will

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 on 3D Orientation Point Objects, QR2002.

Integrates orientation, distance, cardinal directions over points as well as extended objects.

- RCC Reasoning - B. Bennet, A.G. Cohn, Leeds UK.
- PMON logic for dynamical temporal systems - E. Sandewall, Linkoeping Univ.
Agents and Actions


Specification and Verification of Agent Interaction...

- 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, Linköping University, Sweden.
Types and Security

Chameleon Project, Martin Sulzmann, Peter J. Stuckey.

Improving type error diagnosis, Haskell’04, ACM.
Sound and Decidable Type Inference for Functional Dependencies, ESOP’04, LNCS 2968.

Haskell type classes, implication and dependent types.

\[
\begin{align*}
\text{Sub(Int,Float)} & \iff \text{true;} \\
\text{Sub}(a_1\rightarrow a_2, b_1\rightarrow b_2) & \iff \text{Sub}(b_1,a_1), \text{Sub}(a_2,b_2);
\end{align*}
\]
Types and Security

Chameleon Project, Martin Sulzmann, Peter J. Stuckey.

Improving type error diagnosis, Haskell’04, ACM.
Sound and Decidable Type Inference for Functional Dependencies, ESOP’04, LNCS 2968.

- 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,

- 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.
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 Interchange...for the intelligent integration of information, ACM Transactions on Information Systems, 1999.
Computational Linguistics

CHR Grammars, H. Christiansen, TPLP CHR Special Issue 2005. 
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

\[ X_r = (\text{Cond})?Xw1:X_i \]

\[ \text{ite}(\text{true}, X_r, Xw1, X_i) \iff X_r = Xw1. \]
\[ \text{ite}(\text{false}, X_r, Xw1, X_i) \iff X_r = X_i. \]
\[ \text{ite}(\text{Cond}, X_r, X, X) \iff X_r = X. \]
Gregory J. Duck, University of Melbourne.
Scene calculated from interfering 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.

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

\[
\text{age}(X,A), \text{history}(X,\text{smoker}), \\
\text{serum_data}(X,\text{marker_type}) \iff \\
\text{marker}(X,\text{marker_type},P,B), \\
\text{probability}(P,X,B) \mid \\
\text{possible_lung_cancer}(\text{yes},X).
\]
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).

Constraint Handling Rules (CHR)
Further Reading

Essentials of Constraint Programming
Thom Frühwirth, Slim Abdennadher

Constraint-Programmierung Lehrbuch
Thom Frühwirth, Slim Abdennadher
The CHR Logo

[Image of 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


### 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.