Constraint Programming with CHR

Thom Frühwirth

Faculty of Computer Science
University of Ulm, Germany

April 2005

Images are subject to copyright of the respective owners

Citations may be incomplete for space reasons
## Overview

### Constraint Programming

1. Constraint Reasoning
2. Constraint Programming
3. Background
4. More Examples

### CHR...

5. Constraint Handling Rules (CHR)
6. Program Analysis
7. Constraint Solvers

### ...Around the World

8. Language Issues
9. Classical Applications
Part I

Constraint Programming

1. Constraint Reasoning
2. Constraint Programming
3. Background
4. More Examples
The Holy Grail

**Constraint Programming** represents one of the closest approaches computer science has yet made to the **Holy Grail** of programming: the user states the problem, the computer solves it.

The Idea

- **Combination Lock Example**
  
  0 1 2 3 4 5 6 7 8 9

  Greater or equal 5.
  Prime number.

- **Declarative problem** representation by variables and constraints:
  
  \[ x \in \{0, 1, \ldots, 9\} \land x \geq 5 \land \text{prime}(x) \]

- **Constraint propagation and simplification** reduce search space:
  
  \[ x \in \{0, 1, \ldots, 9\} \land x \geq 5 \rightarrow x \in \{5, 6, 7, 8, 9\} \]
Constraint Reasoning Everywhere

Combination

Simplification

Contradiction

Redundancy
Constraint Reasoning
Constraint Programming
Background
More Examples

Terminology

Language is first-order logic with equality.

- **Constraint:**
  Conjunction of atomic constraints (predicates)
  E.g., $4X + 3Y = 10 \land 2X - Y = 0$

- **Constraint Problem (Query):**
  A given, initial constraint

- **Constraint Solution (Answer):**
  A valuation for the variables in a given constraint problem that satisfies all constraints of the problem. E.g., $X = 1 \land Y = 2$

In general, a normal/solved form of, e.g., the problem

$4X + 3Y + Z = 10 \land 2X - Y = 0$ simplifies into

$Y + Z = 10 \land 2X - Y = 0$
Mortgage

D: Amount of Loan, Debt, Principal
T: Duration of loan in months
I: Interest rate per month
R: Rate of payments per month
S: Balance of debt after T months

\[ \text{mortgage}(D, T, I, R, S) \leftrightarrow \]
\[ \begin{align*}
T &= 0, \\
D &= S \\
\quad ; \\
T &> 0, \\
T_1 &= T - 1, \\
D_1 &= D + D*I - R, \\
\text{mortgage}(D_1, T_1, I, R, S). 
\end{align*} \]
Mortgage II

\[
\text{mortgage}(D, T, I, R, S) \iff \\
T = 0, D = S \\
; \\
T > 0, T1 = T - 1, D1 = D + D \times I - R, \\
\text{mortgage}(D1, T1, I, R, S). \\
\]

- \text{mortgage}(100000, 360, 0.01, 1025, S) yields \( S=12625.90 \).
- \text{mortgage}(D, 360, 0.01, 1025, 0) yields \( D=99648.79 \).
- \text{mortgage}(100000, T, 0.01, 1025, S), S=<0 yields \( T=374, S=-807.96 \).
- \text{mortgage}(D, 360, 0.01, R, 0) yields \( R=0.0102861198 \times D \).
Advantages of Constraint Logic Programming

**Theoretical**
Logical Foundation – First-Order Logic

**Conceptual**
Sound Modeling

**Practical**
Efficient Algorithms/Implementations
Combination of different Solvers
Early Commercial Applications (in the 90s)

- **Lufthansa**: Short-term staff planning.
- **Hongkong Container Harbor**: Resource planning.
- **Renault**: Short-term production planning.
- **Nokia**: Software configuration for mobile phones.
- **Airbus**: Cabin layout.
- **Siemens**: Circuit verification.
- **Caisse d’epargne**: Portfolio management.

In *Decision Support Systems for Planning and Configuration*, for Design and Analysis.
**Constraint Reasoning and Programming**

**Generic Framework** for
- **Modeling**
  - with partial information
  - with infinite information
- **Reasoning**
  - with new information
- **Solving**
  - combinatorial problems
Early History of Constraint Programming

60s, 70s Constraint networks in artificial intelligence.
70s Logic programming (Prolog).
80s Constraint logic programming.
80s Concurrent logic programming.
90s Concurrent constraint programming.
90s Commercial applications.
Constraint Reasoning Algorithms

Adaption and combination of existing efficient algorithms from

- **Mathematics**
  - Operations research
  - Graph theory
  - Algebra

- **Computer Science**
  - Finite automata
  - Automatic proving

- **Economics**

- **Linguistics**
Application Domains

- Modeling
- Executable Specifications
- Solving Combinatorial Problems
- Scheduling, Planning, Timetabling
- Configuration, Layout, Placement, Design
- Analysis: Simulation, Verification, Diagnosis
  of software, hardware and industrial processes.
Application Domains II

- Artificial Intelligence
  - Machine Vision
  - Natural Language Understanding
  - Temporal and Spatial Reasoning
  - Theorem Proving
  - Qualitative Reasoning
  - Robotics
  - Agents
  - Bioinformatics
Applications in Research

- **Computer Science:** Program Analysis, Robotics, Agents
- **Molecular Biology, Biochemistry, Bioinformatics:** Protein Folding, Genomic Sequencing
- **Economics:** Scheduling
- **Linguistics:** Parsing
- **Medicine:** Diagnosis Support
- **Physics:** System Modeling
- **Geography:** Geo-Information-Systems
Early Commercial Applications

- **Lufthansa**: Short-term staff planning.
- **Hongkong Container Harbor**: Resource planning.
- **Renault**: Short-term production planning.
- **Nokia**: Software configuration for mobile phones.
- **Airbus**: Cabin layout.
- **Siemens**: Circuit verification.
- **Caisse d'epargne**: Portfolio management.

In *Decision Support Systems for Planning and Configuration*, for *Design and Analysis*.
Crypto-Arithmetic Problem

\[
\begin{array}{cccccccc}
S & E & N & D \\
+ & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}
\]

solve(S,E,N,D,M,O,R,Y) :-
[S,E,N,D,M,O,R,Y] in 0..9,
S \neq 0, M \neq 0,
alldifferent([S,E,N,D,M,O,R,Y]),
1000*S + 100*E + 10*N + D
+ 1000*M + 100*O + 10*R + E
= 10000*M + 1000*O + 100*N + 10*E + Y,
labeling([S,E,N,D,M,O,R,Y]).

S=9, E in 4..7, N in 5..8, M=1, O=0, [D,R,Y] in 2..8
With Search: S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2
Crypto-Arithmetic Problem

\[
\begin{array}{ccc}
& S & E & N & D \\
+ & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}
\]

```
solve(S,E,N,D,M,O,R,Y) :-
    [S,E,N,D,M,O,R,Y] in 0..9,
    S \neq 0, M \neq 0,
    alldifferent([S,E,N,D,M,O,R,Y]),
    1000*S + 100*E + 10*N + D
    + 1000*M + 100*O + 10*R + E
    = 10000*M + 1000*O + 100*N + 10*E + Y,
    labeling([S,E,N,D,M,O,R,Y]).
```

S=9, E in 4..7, N in 5..8, M=1, O=0, [D,R,Y] in 2..8

With Search: S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2
### Crypto-Arithmetic Problem

\[ \begin{array}{c}
S & E & N & D \\
+ & M & O & R & E \\
\hline
M & O & N & E & Y
\end{array} \]

\[
\text{solve}(S,E,N,D,M,O,R,Y) : - \\
[S,E,N,D,M,O,R,Y] \text{ in } 0..9, \\
S \neq 0, \ M \neq 0, \\
alldifferent([S,E,N,D,M,O,R,Y]), \\
1000* S + 100* E + 10* N + D \\
+ 1000 * M + 100 * O + 10 * R + E \\
= 10000 * M + 1000 * O + 100 * N + 10 * E + Y, \\
labeling([S,E,N,D,M,O,R,Y]).
\]

\[ S=9, \ E \text{ in } 4..7, \ N \text{ in } 5..8, \ M=1, \ O=0, \ [D,R,Y] \text{ in } 2..8 \]

With Search: \[ S=9, \ E=5, \ N=6, \ D=7, \ M=1, \ O=0, \ R=8, \ Y=2 \]
Crypto-Arithmetic Problem

\[
\begin{array}{c}
\text{S} & \text{E} & \text{N} & \text{D} \\
+ & \text{M} & \text{O} & \text{R} & \text{E} \\
\hline
\text{M} & \text{O} & \text{N} & \text{E} & \text{Y}
\end{array}
\]

\[\text{solve}(S,E,N,D,M,O,R,Y) \leftarrow\]
\[\text{[S,E,N,D,M,O,R,Y]} \text{ in } 0..9,\]
\[S \neq 0, M \neq 0,\]
\[\text{alldifferent([S,E,N,D,M,O,R,Y])},\]
\[1000*S + 100*E + 10*N + D\]
\[+\]
\[1000*M + 100*O + 10*R + E\]
\[= 10000*M + 1000*O + 100*N + 10*E + Y,\]
\[\text{labeling([S,E,N,D,M,O,R,Y])}.\]

S=9, E in 4..7, N in 5..8, M=1, O=0, [D,R,Y] in 2..8
With Search: S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2
Crypto-Arithmetic Problem

\[
\begin{array}{c}
S \\
E \\
N \\
D \\
+ \\
M \\
O \\
R \\
E \\
= \\
M \\
O \\
N \\
E \\
Y \\
\end{array}
\]

\[
\text{solve}(S,E,N,D,M,O,R,Y) :- \\
[S,E,N,D,M,O,R,Y] \text{ in } 0..9, \\
S \neq 0, M \neq 0, \\
\text{alldifferent}([S,E,N,D,M,O,R,Y]), \\
1000*S + 100*E + 10*N + D \\
+ 1000*M + 100*O + 10*R + E \\
= 10000*M + 1000*O + 100*N + 10*E + Y, \\
\text{labeling}([S,E,N,D,M,O,R,Y]).
\]

S=9, E in 4..7, N in 5..8, M=1, O=0, [D,R,Y] in 2..8
With Search: S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2
Crypto-Arithmetic Problem

\[
\begin{array}{c}
S \quad E \quad N \quad D \\
+ \quad M \quad O \quad R \quad E \\
= \quad M \quad O \quad N \quad E \quad Y
\end{array}
\]

solve(S,E,N,D,M,O,R,Y) :-
[S,E,N,D,M,O,R,Y] in 0..9,
S \neq 0, M \neq 0,
alldifferent([S,E,N,D,M,O,R,Y]),
1000*S + 100*E + 10*N + D
+ 1000*M + 100*O + 10*R + E
= 10000*M + 1000*O + 100*N + 10*E + Y,
labeling([S,E,N,D,M,O,R,Y]).

S=9, E in 4..7, N in 5..8, M=1, O=0, [D,R,Y] in 2..8
With Search: S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2
Crypto-Arithmetic Problem

\[
\begin{array}{c}
S \\
E \\
N \\
D \\
+ \\
M \\
O \\
R \\
E \\
\hline \\
= \\
M \\
O \\
N \\
E \\
Y
\end{array}
\]

\[
\text{solve}(S,E,N,D,M,O,R,Y) :- \\
[S,E,N,D,M,O,R,Y] \text{ in } 0..9, \\
S \neq 0, M \neq 0, \\
\text{alldifferent}([S,E,N,D,M,O,R,Y]), \\
1000*S + 100*E + 10*N + D \\
+ 1000*M + 100*O + 10*R + E \\
= 10000*M + 1000*O + 100*N + 10*E + Y, \\
\text{labeling}([S,E,N,D,M,O,R,Y]).
\]

S=9, E in 4..7, N in 5..8, M=1, O=0, [D,R,Y] in 2..8

With Search: S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2
Crypto-Arithmetic Problem

\[
\begin{array}{cccc}
S & E & N & D \\
+ & M & O & R & E \\
\hline
= & M & O & N & E & Y
\end{array}
\]

\[
solve(S,E,N,D,M,O,R,Y) :- \\
[S,E,N,D,M,O,R,Y] in 0..9, \\
S \neq 0, M \neq 0, \\
\text{alldifferent}([S,E,N,D,M,O,R,Y]), \\
1000*S + 100*E + 10*N + D \\
+ 1000*M + 100*O + 10*R + E \\
= 10000*M + 1000*O + 100*N + 10*E + Y, \\
\text{labeling}([S,E,N,D,M,O,R,Y]).
\]

S=9, E in 4..7, N in 5..8, M=1, O=0, [D,R,Y] in 2..8
With Search: S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2
solve(S,E,N,D,M,O,R,Y) :-
    [S,E,N,D,M,O,R,Y] in 0..9,
    S \neq 0, M \neq 0,
    alldifferent([S,E,N,D,M,O,R,Y]),
    1000*S + 100*E + 10*N + D
    + 1000*M + 100*O + 10*R + E
    = 10000*M + 1000*O + 100*N + 10*E + Y,
    labeling([S,E,N,D,M,O,R,Y]).

S=9, E in 4..7, N in 5..8, M=1, O=0, [D,R,Y] in 2..8
With Search: S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2
**n-Queens Problem**

Place $n$ queens $q_1, \ldots, q_n$ on an $n \times n$ chess board, such that they do not attack each other.

- no two queens on same row, column or diagonal
  - each row and each column with exactly one queen
  - each diagonal at most one queen
- $q_i$: row position of the queen in the $i$-th column

$$q_1, \ldots, q_n \in \{1, \ldots, n\}$$

\[ \forall \ i \neq j. \ q_i \neq q_j \land |q_i - q_j| \neq |i - j| \]
n-Queens Problem II

Place $n$ queens $q_1, \ldots, q_n$ on an $n \times n$ chess board, such that they do not attack each other.

\[
\begin{array}{cccc}
q_1 & q_2 & q_3 & q_4 \\
1 & & & \\
2 & & & \\
3 & & & \\
4 & & & \\
\end{array}
\]

$q_1, \ldots, q_n \in \{1, \ldots, n\}$

$\forall \ i \neq j. \ q_i \neq q_j \land |q_i - q_j| \neq |i - j|$

\[
\text{solve}(N,Qs) \iff \text{makedomains}(N,Qs), \text{queens}(Qs), \text{enum}(Qs).
\]

\[
\text{queens}([Q|Qs]) \iff \text{safe}(Q,Qs,1), \text{queens}(Qs).
\]

\[
\text{safe}(X,[Y|Qs],N) \iff \text{noattack}(X,Y,N), \text{safe}(X,Qs,N+1).
\]

\[
\text{noattack}(X,Y,N) \iff X \neq Y, \ X+N \neq Y, \ Y+N \neq X.
\]
n-Queens Problem III

solve(4, [Q1, Q2, Q3, Q4])

- **makedomains** produces
  
  Q1 in [1,2,3,4], Q2 in [1,2,3,4]
  Q3 in [1,2,3,4], Q4 in [1,2,3,4]

- **safe** adds noattack producing ne constraints

- **enum** called for labeling

- [Q1, Q2, Q3, Q4] = [2,4,1,3], [Q1, Q2, Q3, Q4] = [3,1,4,2]
Part II

CHR…
Part II

CHR...

Transcribed as CHR, means horse, but also 飈
Transcribed as **CHR**, means horse, but also to speed, to propagate, to be famous
Constraint Handling Rules (CHR)

5

- Example Partial Order
- Syntax and Declarative Semantics
- Operational Semantics
- Operational Properties

Program Analysis

6

- Termination and Complexity
- Confluence
- Completion
- Operational Equivalence

Constraint Solvers

7

- Boolean Constraints
- Linear Polynomial Equations
- Syntactic Unification
- Finite Domains
Constraint Handling Rules (CHR)

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

- theorem proving with constraints
- combining forward and backward chaining
- manipulating attributed variables
- combining deduction and abduction
- bottom-up evaluation with integrity constr.
- top-down evaluation with tabulation
- production rule systems
- event-condition-action (ECA) rules
- simplification and propagation of constraints

15+ Implementations: Prolog, Java, Haskell,...

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

15+ Implementations: Prolog, Java, Haskell,...
CHR in Numbers

Constraint Handling Rules:
Concurrent committed-choice guarded rules with ask and tell constraints for computational logic and more...

1 language
2 semantics
3 kinds or rules
4 main implementors
5 host languages
15+ implementations
100+ projects use CHR
200+ citations of main paper
500+ references to CHR
1991 year of creation of CHR
CHR in Numbers

Constraint Handling Rules:
Concurrent committed-choice guarded rules with ask and tell constraints for computational logic and more...

1 language
2 semantics
3 kinds or rules
4 main implementors
5 host languages
15+ implementations
100+ projects use CHR
200+ citations of main paper
500+ references to CHR
1991 year of creation of CHR
CHR in Numbers

Constraint Handling Rules:
Concurrent committed-choice guarded rules with ask and tell constraints for computational logic and more...

1 language
2 semantics
3 kinds or rules
4 main implementors
5 host languages
15+ implementations
100+ projects use CHR
200+ citations of main paper
500+ references to CHR
1991 year of creation of CHR
CHR in Numbers

Constraint Handling Rules:
Concurrent committed-choice guarded rules with ask and tell constraints for computational logic and more...

1 language
2 semantics
3 kinds or rules
4 main implementors
5 host languages
15+ implementations
100+ projects use CHR
200+ citations of main paper
500+ references to CHR
1991 year of creation of CHR
CHR in Numbers

Constraint Handling Rules:
Concurrent committed-choice guarded rules with ask and tell constraints for computational logic and more...

1 language
2 semantics
3 kinds or rules
4 main implementors
5 host languages
15+ implementations
100+ projects use CHR
200+ citations of main paper
500+ references to CHR
1991 year of creation of CHR
CHR in Numbers

Constraint Handling Rules:
Concurrent committed-choice guarded rules with ask and tell constraints for computational logic and more...

1 language
2 semantics
3 kinds or rules
4 main implementors
5 host languages
15+ implementations
100+ projects use CHR
200+ citations of main paper
500+ references to CHR
1991 year of creation of CHR
CHR in Numbers

Constraint Handling Rules:
Concurrent committed-choice guarded rules with ask and tell constraints for computational logic and more...

1 language
2 semantics
3 kinds or rules
4 main implementors
5 host languages
15+ implementations
100+ projects use CHR
200+ citations of main paper
500+ references to CHR
1991 year of creation of CHR
CHR in Numbers

Constraint Handling Rules:
Concurrent committed-choice guarded rules with ask and tell constraints for computational logic and more...

1 language
2 semantics
3 kinds or rules
4 main implementors
5 host languages
15+ implementations
100+ projects use CHR
200+ citations of main paper
500+ references to CHR
1991 year of creation of CHR
CHR in Numbers

Constraint Handling Rules:
Concurrent committed-choice guarded rules with ask and tell constraints for computational logic and more...

1 language
2 semantics
3 kinds or rules
4 main implementors
5 host languages
15+ implementations
100+ projects use CHR
200+ citations of main paper
500+ references to CHR
1991 year of creation of CHR
 CHR in Numbers

Constraint Handling Rules:
Concurrent committed-choice guarded rules with ask and tell constraints for computational logic and more...

1 language
2 semantics
3 kinds or rules
4 main implementors
5 host languages
15+ implementations
100+ projects use CHR
200+ citations of main paper
500+ references to CHR
1991 year of creation of CHR
Example Partial Order Constraint

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

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

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

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

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

\[ X \leq Y \land Y \leq X \iff X = Y \] (antisymmetry)

\[ X \leq Y \land Y \leq Z \Rightarrow X \leq Z \] (transitivity)

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

(transitivity)

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

(antisymmetry)

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

(built-in solver)

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

(antisymmetry)
Example Partial Order Constraint

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

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

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

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

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

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

\text{(transitivity)}
\text{(antisymmetry)}
\text{(built-in solver)}
\text{(antisymmetry)}
**Example Partial Order Constraint**

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

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

(antisymmetry)  
(built-in solver)  
(antisymmetry)
Syntax and Declarative Semantics

Declarative Semantics

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

**Propagation rule:** \( H \Rightarrow C \mid B \) \( \forall \overline{x} (C \rightarrow (H \rightarrow \exists \overline{y} B)) \)

*Constraint Theory for Built-Ins*

- \( H \): non-empty conjunction of CHR constraints
- \( C \): conjunction of built-in constraints
- \( B \): conjunction of CHR and built-in constraints
Operational Semantics

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

Simplify

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

Propagate

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

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

\[
\begin{align*}
A \leq B & \land B \leq C & C \leq A \\
\downarrow & & (\text{transitivity}) \\
A \leq B & \land B \leq C & C \leq A & \land A \leq C \\
\downarrow & & (\text{antisymmetry}) \\
A \leq B & \land B \leq C & A = C \\
\downarrow & & (\text{built-in solver}) \\
A \leq B & \land B \leq A & A = C \\
\downarrow & & (\text{antisymmetry}) \\
A = B & \land A = C
\end{align*}
\]
Anytime Algorithm

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

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

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

\[
A \leq B \land B \leq C \land C \leq A
\]

↓

\[
A \leq B \land B \leq C \land C \leq A \land A \leq C
\]

(transitivity)

↓

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

(antisymmetry)

↓

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

(built-in solver)

↓

\[
A = B \land A = C
\]

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

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

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

(transitivity)

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

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

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

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

Monotonicity and Incrementality

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

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

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

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

\[
\frac{A \leq B \land B \leq C}{A \leq B \land B \leq C \land A \leq C}
\]

\[
\frac{C \leq D \land D \leq A}{C \leq D \land D \leq A \land C \leq A}
\]

\[
\ldots A = C \ldots
\]
Concurrency

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

If \( A \mapsto B \) and \( C \mapsto D \), then \( A \land C \mapsto B \land D \).

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

\[
\begin{align*}
C \leq D \land D \leq A \land C \leq A \\
C \leq D \land D \leq A \land C \leq A
\end{align*}
\]

\[
\begin{align*}
\ldots \quad A = C \quad \ldots
\end{align*}
\]
Concurrency

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

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

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

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

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

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

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

\[\vdots \quad A = C \quad \vdots\]
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]
A *ranking* $\|\|$ maps terms into natural numbers. For all simplification rules

$$H_1 \land \ldots \land H_n \iff C \mid D \land B_1 \land \ldots \land B_m$$

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]
Minimal States

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

Rule: \[ H \iff C \mid B \quad \text{or} \quad H \Rightarrow C \mid B \]

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: \( H \leftrightarrow C \mid B \) 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: \[ H \iff C \mid B \quad \text{or} \quad H \Rightarrow C \mid B \]

Minimal State: \[ H \land C \]

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

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

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

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

Start from overlapping minimal states

- Reflexivity: \[ A \leq A \land A \leq A \]
- Antisymmetry: \[ A = A \]
- Reflexivity: \[ A \leq A \]
- Built-in: \[ \text{true} \]
Confluence

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

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

Start from overlapping minimal states

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

\[
\begin{align*}
A \leq A \land A \leq A & \iff A = A \quad \text{(antisymmetry)} \\
A \leq A & \iff \text{true} \quad \text{(reflexivity)} \\
A \leq A & \iff A = A \quad \text{(built-in)}
\end{align*}
\]
Confluence

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

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

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

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

Start from overlapping minimal states

\[
A \leq A \land A \leq A
\]

reflexivity

\[
A \leq A
\]

reflexivity

\[
A = A
\]

antisymmetry

\[
true
\]

built-in
Completion

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

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

\[ A \leq B \land B \leq A \land B < A \]
\[ \quad \text{antisymmetry} \rightarrow A = B \land B < A \]
\[ \quad \text{inconsistency} \rightarrow B \leq A \land \text{false} \]

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

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

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

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

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

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

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

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

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

antisymmetry

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

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

inconsistency

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

\[ \text{false} \]

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

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

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

\[ P_1 \quad \text{max}(X, Y, Z) \iff X < Y \quad | \quad Z = Y. \]
\[ \text{max}(X, Y, Z) \iff X \geq Y \quad | \quad Z = X. \]

\[ P_2 \quad \text{max}(X, Y, Z) \iff X \leq Y \quad | \quad Z = Y. \]
\[ \text{max}(X, Y, Z) \iff X > Y \quad | \quad Z = X. \]

\[ \text{max}(X, Y, Z) \land X \geq Y \quad \downarrow \quad \text{max}(X, Y, Z) \land X \geq Y \]
\[ P_1 \quad \downarrow \quad P_2 \]
\[ Z = X \land X \geq Y \]
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.

\[ P1 \quad \max(X, Y, Z) \iff X < Y \land Z = Y. \]
\[ \max(X, Y, Z) \iff X \geq Y \land Z = X. \]

\[ P2 \quad \max(X, Y, Z) \iff X \leq Y \land Z = Y. \]
\[ \max(X, Y, Z) \iff X > Y \land Z = X. \]
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.

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

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

\[
\text{max}(X, Y, Z) \land X \geq Y \quad \text{max}(X, Y, Z) \land X \geq Y
\]

\( P_1 \)
\[
\rightarrow Z = X \land X \geq Y
\]

\( P_2 \)
Boolean Constraints

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

\[\begin{align*}
\text{and}(X, X, Z) & \iff X = Z. \\
\text{and}(X, Y, 1) & \iff X = 1 \land Y = 1. \\
\text{and}(X, 1, Z) & \iff X = Z. \\
\text{and}(X, 0, Z) & \iff Z = 0. \\
\text{and}(1, Y, Z) & \iff Y = Z. \\
\text{and}(0, Y, Z) & \iff Z = 0. \\
\text{imp}(0, X) & \iff \text{true}. \\
\text{imp}(X, 0) & \iff X = 0. \\
\text{imp}(1, X) & \iff X = 1. \\
\text{imp}(X, 1) & \iff \text{true}. 
\end{align*}\]
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.

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

Completion adds the rules:

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

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

Non-confluent: E.g.

\[ and(X, X, X) \]

Completion adds the rules:

\[ \text{imp}(X, X) \Leftrightarrow \text{true}. \]
\[ \text{imp}(X, Y) \land \text{imp}(X, Y) \Leftrightarrow \text{imp}(X, Y). \]
\[ \text{imp}(X, Y) \land \text{and}(X, Y, Z) \Leftrightarrow \text{imp}(X, Y) \land X=Z. \]
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.

\[
\text{and}(X, X, X) \\
\text{true} \quad \text{imp}(X, X)
\]

Completion adds the rules:

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

\[
\begin{align*}
\text{empty clause} @ & \quad \text{cl}([]) \iff \text{false}. \\
\text{tautology} @ & \quad \text{cl}(L) \iff \text{in}(+X,L) \land \text{in}(-X,L) \mid \text{true}.
\end{align*}
\]

\[
\begin{align*}
\text{resolution} @ & \quad \text{cl}(L1) \land \text{cl}(L2) \Rightarrow \\
& \quad \text{find}(+X,L1,L3) \land \text{find}(-X,L2,L4) \mid \\
& \quad \text{merge}(L3,L4,L) \land \text{cl}(L).
\end{align*}
\]
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., \( \lnot x \lor y \lor z \) as \( \text{cl}([-x,+y,+z]) \).

\[
\begin{align*}
\text{empty_clause} & \quad @ \quad \text{cl}([]) \iff \text{false}. \\
\text{tautology} & \quad @ \quad \text{cl}(L) \iff \text{in}(+X,L) \land \text{in}(-X,L) | \text{true}. \\
\text{resolution} & \quad @ \quad \text{cl}(L_1) \land \text{cl}(L_2) \Rightarrow \\
& \quad \text{find}(+X,L_1,L_3) \land \text{find}(-X,L_2,L_4) | \\
& \quad \text{merge}(L_3,L_4,L) \land \\
& \quad \text{cl}(L).
\end{align*}
\]
Linear Polynomial Equations

**Equations** of the form $a_1x_1 + \ldots + a_nx_n + b = 0$.

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

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

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

$$1*X+3*Y+5=0 \land 3*X+2*Y+8=0$$
$$\text{compute}((2*Y+8) - ((3*Y+5)/1)*3,P3) \ % \ P3=-7*Y+ -7$$
$$1*X+3*Y+5=0 \land -7*Y+ -7=0 \ % \ Y=-1$$
$$\text{compute}((1*X+5) - ((-7)/-7)*3,P3') \ % \ P3'=1*X+2$$
$$1*X+2=0 \land -7*Y+ -7=0 \ % \ X=-2$$
Equations of the form $a_1x_1 + \ldots + a_nx_n + b = 0$.

**Solved form:** leftmost variable occurs only once.

Reach solved normal form by **variable elimination**.

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

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

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

**Equations** of the form \( a_1x_1 + \ldots + a_nx_n + b = 0 \).

**Solved form:** Leftmost variable occurs only once.

Reach solved normal form by **variable elimination**.

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

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

\[
1 \times X + 3 \times Y + 5 = 0 \land 3 \times X + 2 \times Y + 8 = 0 \\
\text{compute}\left((2 \times Y + 8) - \left((3 \times Y + 5)/1\right) \times 3, P_3\right) \% P_3 = -7 \times Y + -7 \\
1 \times X + 3 \times Y + 5 = 0 \land -7 \times Y + -7 = 0 \% Y = -1 \\
\text{compute}\left((1 \times X + 5) - \left((-7)/-7\right) \times 3, P_3'\right) \% P_3' = 1 \times X + 2 \\
1 \times X + 2 = 0 \land -7 \times Y + -7 = 0 \% X = -2
\]
Linear Polynomial Equations

Equations of the form $a_1x_1 + \ldots + a_nx_n + b = 0$.

Solved form: leftmost variable occurs only once.

Reach solved normal form by variable elimination.

\[ A_1X + P_1 = 0 \land XP = 0 \iff \]
\[ \text{find}(A_2X, XP, P_2) \land \]
\[ \text{compute}(P_2 - (P_1/A_1)*A_2, P_3) \land \]
\[ A_1X + P_1 = 0 \land P_3 = 0. \]

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

\[ 1X + 3Y + 5 = 0 \land 3X + 2Y + 8 = 0 \]
\[ \text{compute}((2Y + 8) - ((3Y + 5)/1)*3, P_3) \]
\[ \text{compute}((1X + 5) - ((-7)/-7)*3, P_3') \]
\[ 1X + 2 = 0 \land -7Y + -7 = 0 \]
\[ 1X + 2 = 0 \land -7Y + -7 = 0 \]

Thom Fröhlich 
Constraint Programming with CHR
Fourier’s Algorithm

\begin{align*}
A_1X + P_1 & \geq 0 \land XP \geq 0 \Rightarrow \\
& \text{find}(A_2X, XP, P_2) \land \text{opposite\_sign}(A_1, A_2) \land \\
& \text{canon}(P_2 - (P_1/A_1)A_2, P_3) \land \\
& P_3 \geq 0. \\
B \geq 0 & \iff \text{number}(B) \parallel \text{non\_negative}(B).
\end{align*}
Solver Cooperation, Combination of Algorithms

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

\[
A_1^*X + P_1 = 0 \land XP = 0 \iff \\
\text{find}(A_2^*X, XP, P_2) \\
canon(P_2 - (P_1/A_1)^*A_2, P_3) \land A_1^*X + P_1 = 0 \land P_3 = 0.
\]

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

\[
A_1^*X + P_1 \geq 0 \land XP \geq 0 \Rightarrow \\
\text{find}(A_2^*X, XP, P_2) \land \text{opposite\_sign}(A_1, A_2) \\
canon(P_2 - (P_1/A_1)^*A_2, P_3) \land P_3 \geq 0.
\]

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

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

**Gaussian Elimination for** $=$

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

**Fouriers Algorithm for** $\geq$

\[
A_1 X + P_1 \geq 0 \land XP \geq 0 \implies \\
\text{find}(A_2 X, XP, P_2) \land \text{opposite_sign}(A_1, A_2) \mid \\
\text{canon}(P_2 - (P_1/A_1)A_2, P_3) \land P_3 \geq 0.
\]

**Bridge Rule for** $=$ and $\geq$

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

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

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

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

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

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

\[
A1 \times X + P1 = 0 \land XP \geq 0 \iff \\
\text{find}(A2 \times X, XP, P2) \\
\text{canon}(P2 - (P1/A1) \times A2, P3) \land A1 \times X + P1 = 0 \land P3 \geq 0.
\]
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 @ $T_1=T_2 \iff \text{nonvar}(T_1) \land \text{nonvar}(T_2) \mid$

\hspace{1cm} \text{same_functor}(T_1,T_2) \land

\hspace{2cm} \text{same_args}(T_1,T_2).

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

\hspace{1cm} X=T_1 \land T_1=T_2.$
reflexivity @ \quad X=X \iff \text{var}(X) \mid \text{true}.

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

decomposition @ \quad T1=T2 \iff \text{nonvar}(T1) \land \text{nonvar}(T2) \mid \text{same_functor}(T1,T2) \land \text{same_args}(T1,T2).

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

\[
h(Y,f(a),g(X,a)) = h(f(U),Y,g(h(Y),U))
\]
\(\xrightarrow{\text{decomposition}}\)
\[
Y=f(U) \land f(a)=Y \land g(X,a)=g(h(Y),U)
\]
\(\xrightarrow{\text{orientation}}\)
\[
Y=f(U) \land Y=f(a) \land g(X,a)=g(h(Y),U)
\]
\(\xrightarrow{*}\)
\[
Y=f(U) \land U=a \land X=h(Y) \land U=a
\]
\(\xrightarrow{\text{confrontation}}\)
\[
Y=f(U) \land U=a \land X=h(Y) \land a=a
\]
\(\xrightarrow{\text{decomposition}}\)
\[
Y=f(U) \land U=a \land X=h(Y)
\]
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.

\[
\begin{align*}
\text{h}(Y,f(a),g(X,a)) &= \text{h}(f(U),Y,g(h(Y),U)) \\
Y &= f(U) ∧ f(a)=Y ∧ g(X,a)=g(h(Y),U) \\
\end{align*}
\]

\[
\begin{align*}
\text{Y} &= f(U) ∧ Y=f(a) ∧ g(X,a)=g(h(Y),U) \\
\text{Y} &= f(U) ∧ U=a ∧ X=h(Y) ∧ U=a \\
\text{Y} &= f(U) ∧ U=a ∧ X=h(Y) ∧ a=a \\
\text{Y} &= f(U) ∧ U=a ∧ X=h(Y) \\
\end{align*}
\]
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.

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

\[ \rightarrow \text{decomposition} \rightarrow^* \]

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

\[ \rightarrow \text{orientation} \]

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

\[ \rightarrow^* \]

\[ Y=f(U) ∧ U=a ∧ X=h(Y) ∧ U=a \]

\[ \rightarrow \text{confrontation} \]

\[ Y=f(U) ∧ U=a ∧ X=h(Y) ∧ a=a \]

\[ \rightarrow \text{decomposition} \rightarrow^* \]

\[ Y=f(U) ∧ U=a ∧ X=h(Y) \]
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 @ \( X=T1 \land X=T2 \iff \text{var}(X) \land X<T1 \land T1<T2 \mid X=T1 \land T1=T2 \).

\[
\begin{align*}
\text{decomposition} & \rightarrow \text{orientation} \\
\text{orientation} & \rightarrow \text{confrontation} \\
\text{confrontation} & \rightarrow \text{decomposition}
\end{align*}
\]

\[
\begin{align*}
h(Y,f(a),g(X,a)) &= h(f(U),Y,g(h(Y),U)) \\
Y &= f(U) \land f(a) = Y \land g(X,a) = g(h(Y),U) \\
Y &= f(U) \land Y = f(a) \land g(X,a) = g(h(Y),U) \\
Y &= f(U) \land U = a \land X = h(Y) \land U = a \\
Y &= f(U) \land U = a \land X = h(Y) \land a = a \\
Y &= f(U) \land U = a \land X = h(Y)
\end{align*}
\]
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** @ \( X=T1 \land X=T2 \iff \text{var}(X) \land X@<T1 \land T1@=<T2 \mid X=T1 \land T1=T2. \)

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

\[
\overset{\rightarrow \text{decomposition}}{\Rightarrow}^{\rightarrow \ast}
\]

\[
\overset{\rightarrow \text{orientation}}{\Rightarrow}
\]

\[
\overset{\rightarrow \ast}{\Rightarrow}
\]

\[
\overset{\rightarrow \text{confrontation}}{\Rightarrow}
\]

\[
\overset{\rightarrow \text{decomposition}}{\Rightarrow}^{\rightarrow \ast}
\]
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 \( @ \) \( X=T1 \land X=T2 \iff \text{var}(X) \land X@<T1 \land T1@=<T2 \mid \\
X=T1 \land T1=T2 \).

\[
\begin{align*}
\text{h}(Y,f(a),g(X,a)) = & h(f(U),Y,g(h(Y),U)) \\
\text{\textarrow{decomposition}} \text{\textarrow{}} \iff & Y=f(U) \land f(a)=Y \land g(X,a)=g(h(Y),U) \\
\text{\textarrow{orientation}} \iff & Y=f(U) \land Y=f(a) \land g(X,a)=g(h(Y),U) \\
\text{\textarrow{}} \iff & Y=f(U) \land U=a \land X=h(Y) \land U=a \\
\text{\textarrow{confrontation}} \iff & Y=f(U) \land U=a \land X=h(Y) \land a=a \\
\text{\textarrow{decomposition}} \text{\textarrow{}} \iff & Y=f(U) \land U=a \land X=h(Y)
\end{align*}
\]
Part III

...Around the World

8 Language Issues

9 Classical Applications

10 Trends in Applications

11 Application Projects
Language Issues
- Implementations
- More Semantics
- Program Generation and Transformation
- Language Extensions

Classical Applications

Trends in Applications
- Reasoning Services
- Spatio-Temporal Reasoning
- Agents and Actions
- Logical Algorithms
- Types and Security
- Testing and Verification
- Semantic Web
- Computational Linguistics

Application Projects
- JMMSolve Java Memory Machine
- Lung Cancer Diagnosis
Public Domain Implementations

- **SWI Prolog (new, 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 (2)**, Gregory Duck, Jeremy Wazny, Melbourne, 2004; Martin Sulzmann, Singapore
- **Java Constraint Kit (JCK) (pre-release)**, Slim Abdennadher, Cairo, 2002
Public Domain Implementations

- **SWI Prolog** (new, 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** (2), Gregory Duck, Jeremy Wazny, Melbourne, 2004; Martin Sulzmann, Singapore
- **Java Constraint Kit (JCK)** (pre-release), 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).
  Logical Algorithms, e.g. Union-Find.
  ```

- **A Compositional Semantics for CHR**, Maurizio Gabbrielli, Maria Chiaria Meo, CILC 2004. *Multiple heads are challenging.*
Program Generation

- **Automatic Generation of CHR Constraint Solvers**, Slim Abdennadher, Christophe Rigotti, TPLP CHR Special Issue 2005.


- **Automatic Rule Generation**, Eric Monfroy, Valparaiso, Chile.
Program Transformation

  *Multiple heads make partial evaluation hard.*

  *Are termination and confluence modular?*

  *To implement extensions and optimizations.*
Soft Constraints


E.g. Fuzzy Constraints:
\[ X \leq Y : A, \ Y \leq Z : B \implies X \leq Z : A \ast B \]
\[ E \leq F : 0.5, \ F \leq G : 0.3 \implies E \leq F : 0.5, \ F \leq G : 0.3, \ E \leq G : 0.6 \]
Randomized Algorithms


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.*
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.
S. Abdennadher, M. Saft, S. Will

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

...System for Generation and Confirmation of Hypotheses, Alberti, Chesani, Gavanelli, Lamma, W(C)LP 2005.

*Extensions of Fung/Kowalski IFF proof procedure*

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*: A Flexible Query Language,

S. Abdennadher, H. Schütz, FQAS’98, LNCS.

*CHR + disjunction = abduction, bottom-up/top-down evaluation...*

Don’t-care and Don’t-know Nondeterminism

The CHR^\lor program for append of two lists

append(X,Y,Z) \iff
( X=[] \land Y=L \land Z=L \\
\lor X=[H|L1] \land Y=L2 \land Z=[H|L3] \land append(L1, L2, L3) )

can be improved by adding the following rule

append(X, [], Z) \iff X = Z.
Top-down Evaluation with Tabling

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

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

\[ \text{fib}(0, M) \Rightarrow M = 1. \]
\[ \text{fib}(1, M) \Rightarrow M = 1. \]
\[ \text{fib}(N, M) \Rightarrow N \geq 2 \mid \text{fib}(N-1, M_1) \land \text{fib}(N-2, M_2) \land M = M_1 + M_2. \]
Abduction

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

A bird is either an albatros or a penguin.

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

Penguins can’t fly.

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

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

\[ e(a, b) \land e(b, c) \land e(c, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \]
\[ \downarrow \]
\[ e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \land p(a, d) \]
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) & \Rightarrow p(X, Y). \\
e(X, Z) \land p(Z, Y) & \Rightarrow p(X, Y).
\end{align*}
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

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

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

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

is transformed into

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

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

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

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

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

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

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

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

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

\[
\downarrow
\]

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

\[
\downarrow
\]

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

\[
\downarrow
\]

\[
e(a, b) \land e(b, c) \land e(c, d) \land p(a, b) \land p(b, c) \land p(c, d) \land p(a, c) \land p(b, d) \land p(a, d)
\]
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, Linköping Univ.
Agents and Actions

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

Specification and Verification of Agent Interaction...
*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, Linkoping University, Sweden.
Naive 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.

\[
\text{make} @ \quad \text{make}(X) \iff \text{root}(X,0).
\]
\[
\text{union} @ \quad \text{union}(X,Y) \iff \text{find}(X,A), \text{find}(Y,B), \text{link}(A,B).
\]
\[
\text{findNode} @ \quad X \sim P X, \text{find}(X,R) \iff \text{find}(P X,R), X \sim R.
\]
\[
\text{findRoot} @ \quad \text{root}(X) \setminus \text{find}(X,R) \iff R=X.
\]
\[
\text{linkEq} @ \quad \text{link}(X,X) \iff \text{true}.
\]
\[
\text{linkLeft} @ \quad \text{link}(X,Y), \text{root}(X,R X), \text{root}(Y,R Y) \iff R X \geq R Y \mid Y \sim X, \text{root}(X,\text{max}(R X, R Y+1)).
\]
\[
\text{linkRight} @ \quad \text{link}(X,Y), \text{root}(Y,R Y), \text{root}(X,R X) \iff R Y \geq R X \mid X \sim Y, \text{root}(Y,\text{max}(R Y, R X+1)).
\]
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).

term @ A→T \land word(T+R) \Rightarrow parses(U,T+R,R).

non-term @ A→B*C \land parses(B,I,J) \land parses(C,J,K) \Rightarrow parses(A,I,K).

substr @ word(T+R) \Rightarrow word(R).
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.


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

TypeTool - A Type Inference Visualization Tool, Sandra Alves, Mario Florido, WF(C)L P 2004; Type Inference with CHR, WF(C)L P 2001.

Subtyping Constraints in Quasi-lattices, Emmanuel Coquery, Francois Fages, LNCS 2914, 2003; TCLP tool for Type Checking CHR.

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. Kriegel,
TU Munich,

- 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.

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

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

Abduction, Assumption Grammars.
Topological Parsing, Gerald Penn et. al., EACL’03.

HPDG, Attribute Logic.

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

\[ X_r = (\text{Cond})?X_{w1}:X_i \]

\[ \text{ite(true,Xr,Xw1,Xi)} \iff X_r = X_{w1}. \]
\[ \text{ite(false,Xr,Xw1,Xi)} \iff X_r = X_i. \]
\[ \text{ite(Cond,Xr,X,X)} \iff X_r = X. \]
Lung Cancer Diagnosis

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

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).

Thom Frühwirth
Constraint Programming with CHR
Further Reading

**Essentials of Constraint Programming**
Thom Frühwirth, Slim Abdennadher

**Constraint-Programmierung Lehrbuch**
Thom Frühwirth, Slim Abdennadher