Constraint Programming with CHR

Thom Frühwirth

Faculty of Computer Science
University of Ulm, Germany

April 2005

Citations may be incomplete for space reasons

Overview

Constraint Reasoning and Programming

Part I

Constraint Programming

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.

Eugene C. Freuder, Inaugural issue of the *Constraints Journal*, 1997.

Constraint Reasoning

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:
  $x \in \{0, 1, \ldots, 9\} \ \wedge \ x \geq 5 \ \wedge \ \mathrm{prime}(x)$
- *Constraint propagation and simplification* reduce search space:
  $x \in \{0, 1, \ldots, 9\} \wedge x \geq 5 \ \rightarrow \ x \in \{5, 6, 7, 8, 9\}$

Constraint Reasoning Everywhere

Combination

Simplification

Contradiction

Redundancy

Terminology

Language is first-order logic with equality.

- Constraint:
  Conjunction of atomic constraints (predicates)
  E.g., $4X + 3Y = 10 \ \wedge \ 2X - Y = 0$
- 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., $X = 1 \wedge Y = 2$

In general, a normal/solved form of, e.g., the problem
$4X + 3Y + Z = 10 \ \wedge \ 2X - Y = 0$ simplifies into
$Y + Z = 10 \ \wedge \ 2X - Y = 0$

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

```
mortgage(D, T, I, R, S) <=>
       T = 0,
       D = S
         ;
       T > 0,
       T1 = T - 1,
       D1 = D + D*I - R,
       mortgage(D1, T1, I, R, S).
```

Mortgage II

```
mortgage(D, T, I, R, S) <=>
        T = 0, D = S
          ;
        T > 0, T1 = T - 1, D1 = D + D*I - R,
        mortgage(D1, T1, I, R, S).
```

- mortgage(100000,360,0.01,1025,S) yields S=12625.90.
- mortgage(D,360,0.01,1025,0) yields D=99648.79.
- mortgage(100000,T,0.01,1025,S), S=<0 yields T=374, S=-807.96.
- mortgage(D,360,0.01,R,0) yields R=0.0102861198*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, Biochemestry, 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

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

```
solve(S,E,N,D,M,O,R,Y) :-
         [S,E,N,D,M,O,R,Y] in 0..9,
        S≠0, M ≠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

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

```
solve(S,E,N,D,M,O,R,Y) :-
       [S,E,N,D,M,O,R,Y] in 0..9,
      S≠0, M ≠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

		S	E	N	D
+		M	O	R	E
=	M	O	N	E	Y

```
solve(S,E,N,D,M,O,R,Y) :-
         [S,E,N,D,M,O,R,Y] in 0..9,
        S≠0, M ≠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

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

```
solve(S,E,N,D,M,O,R,Y) :-
         [S,E,N,D,M,O,R,Y] in 0..9,
        S≠0, M ≠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

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

```
solve(S,E,N,D,M,O,R,Y) :-
         [S,E,N,D,M,O,R,Y] in 0..9,
        S≠0, M ≠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

		S	E	N	D
+		M	O	R	E
=	M	O	N	E	Y

```
solve(S,E,N,D,M,O,R,Y) :-
         [S,E,N,D,M,O,R,Y] in 0..9,
        S≠0, M ≠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

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

```
solve(S,E,N,D,M,O,R,Y) :-
          [S,E,N,D,M,O,R,Y] in 0..9,
        S≠0, M ≠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

		S	E	N	D
+		M	O	R	E
=	M	O	N	E	Y

```
solve(S,E,N,D,M,O,R,Y) :-
         [S,E,N,D,M,O,R,Y] in 0..9,
        S≠0, M ≠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

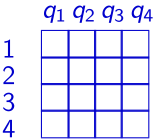
		S	E	N	D
+		M	O	R	E
=	M	O	N	E	Y

```
solve(S,E,N,D,M,O,R,Y) :-
         [S,E,N,D,M,O,R,Y] in 0..9,
        S≠0, M ≠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.



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

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

- 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

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.

	q_1	q_2	q_3	q_4
1				
2				
3				
4				

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

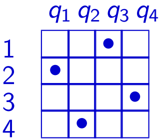
$$\forall\ i \neq j.\ q_i \neq q_j \wedge |q_i - q_j| \neq |i - j|$$

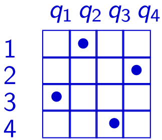
```
solve(N,Qs)       <=> makedomains(N,Qs), queens(Qs), enum(Qs).
queens([Q|Qs])    <=> safe(Q,Qs,1), queens(Qs).
safe(X,[Y|Qs],N)  <=> noattack(X,Y,N), safe(X,Qs,N+1).
noattack(X,Y,N)   <=> X ne Y, X+N ne Y, Y+N ne X.
```

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...

驰

Part II

CHR...

驰

Transcribed as **CHR**, means horse, but also

Part II

CHR...

驰

Transcribed as **CHR**, means horse, but also
to speed, to propagate, to be famous

CHR...

Constraint Handling Rules (CHR)

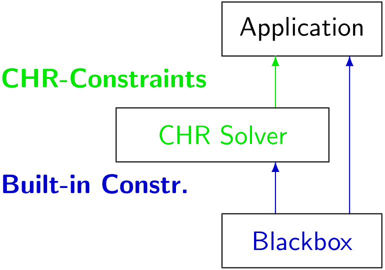
Concurrent committed-choice guarded rules with ask and tell constraints for computational logic and more... (100+ applications)

- 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,...
Extensions: Disjunction/Search, Dynamic and Soft Constraints, Probabilistic Rules, Program Transformation, Literate Programming.

Constraint Handling Rules (CHR)

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



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

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

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Syntax and Declarative Semantics

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

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

Propagate

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

Operational Semantics

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

Simplify

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

Propagate

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

Operational Semantics

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

Simplify

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

Propagate

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

Anytime Algorithm

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

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

Anytime Algorithm

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

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

Anytime Algorithm

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

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

Online Algorithm

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

Monotonicity and Incrementality

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

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

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

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

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

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

$$\downarrow$$

$$\ldots$$

Online Algorithm

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

Monotonicity and Incrementality

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

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

Online Algorithm

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

Monotonicity and Incrementality

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

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

Online Algorithm

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

Monotonicity and Incrementality

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

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

Concurrency

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

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

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

Concurrency

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

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

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

Concurrency

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

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

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

Concurrency

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

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

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

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]

Termination

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

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

it holds that

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

For all propagation rules

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

it holds that

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

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

[Frühwirth, KR'02]

Minimal States

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

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

Minimal State: $H \wedge C$

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

Minimal States

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

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

Minimal State: $H \wedge C$

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

Minimal States

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

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

Minimal State: $H \wedge C$

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

Confluence

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

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

Start from overlapping minimal states

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

Confluence

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

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

Start from overlapping minimal states

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

Confluence

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

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

Start from overlapping minimal states

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

Completion

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

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

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

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

Completion

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

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

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

$$\begin{array}{llll} X{<}X & \Leftrightarrow & \mathit{false} & \text{(irreflexivity)} \end{array}$$

Completion

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

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

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

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

Completion

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

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

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

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

Operational Equivalence

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

$$
\begin{array}{ll}
P1 & max(X,Y,Z) \Leftrightarrow X{<}Y \mid Z{=}Y. \\
 & max(X,Y,Z) \Leftrightarrow X{\geq}Y \mid Z{=}X. \\
 & \\
P2 & max(X,Y,Z) \Leftrightarrow X{\leq}Y \mid Z{=}Y. \\
 & max(X,Y,Z) \Leftrightarrow X{>}Y \mid Z{=}X.
\end{array}
$$

$$
\begin{array}{ccc}
max(X,Y,Z) \wedge X{\geq}Y & \quad & max(X,Y,Z) \wedge X{\geq}Y \\
\downarrow{\scriptstyle P_1} & & \downarrow{\scriptstyle P_2} \\
Z{=}X \wedge X{\geq}Y & &
\end{array}
$$

Operational Equivalence

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

$$
\begin{array}{ll}
P1 & max(X,Y,Z) \Leftrightarrow X{<}Y \mid Z{=}Y. \\
 & max(X,Y,Z) \Leftrightarrow X{\geq}Y \mid Z{=}X. \\
 & \\
P2 & max(X,Y,Z) \Leftrightarrow X{\leq}Y \mid Z{=}Y. \\
 & max(X,Y,Z) \Leftrightarrow X{>}Y \mid Z{=}X.
\end{array}
$$

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

Operational Equivalence

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

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

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

$$max(X, Y, Z) \wedge X{\geq}Y \xrightarrow{P_1} Z{=}X \wedge X{\geq}Y \qquad\qquad max(X, Y, Z) \wedge X{\geq}Y \xrightarrow{P_2}$$

Boolean Constraints

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

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

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

Solver Union and Cooperation by Completion

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

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

Non-confluent: E.g.

$$\begin{tikzcd}
 & and(X,X,X) \arrow[dl] \arrow[dr] & \\
true & & imp(X,X)
\end{tikzcd}$$

Completion adds the rules:

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

Solver Union and Cooperation by Completion

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

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

Non-confluent: E.g.

$$\begin{tikzcd} & and(X, X, X) \arrow[dl] \arrow[dr] & \\ true & & imp(X, X) \end{tikzcd}$$

Completion adds the rules:

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

Solver Union and Cooperation by Completion

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

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

Non-confluent: E.g.

$$\begin{tikzcd}
 & and(X,X,X) \arrow[dl] \arrow[dr] & \\
\mathit{true} & & imp(X,X)
\end{tikzcd}$$

Completion adds the rules:

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

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])`.

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

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

Propositional Resolution

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

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

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

Linear Polynomial Equations

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

B=0  ⇔  number(B) |  zero(B).
```

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

Linear Polynomial Equations

Equations of the form $a_1 x_1 + \ldots + a_n x_n + b = 0$.
Solved form: leftmost variable occurs only once.
Reach solved normal form by **variable elimination**.

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

B=0  ⇔  number(B) | zero(B).

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

Linear Polynomial Equations

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

B=0  ⇔  number(B) |  zero(B).

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

Linear Polynomial Equations

Equations of the form $a_1 x_1 + \ldots + a_n x_n + b = 0$.
Solved form: leftmost variable occurs only once.
Reach solved normal form by **variable elimination**.

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

B=0  ⇔  number(B) |  zero(B).

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

Fourier's Algorithm

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

B≥0  ⇔  number(B) | non_negative(B).
```

Solver Cooperation, Combination of Algorithms

Gaussian Elimination for $=$

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

Fouriers Algorithm for $\geq$

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

Bridge Rule for $=$ and $\geq$

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

Solver Cooperation, Combination of Algorithms

Gaussian Elimination for =

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

Fouriers Algorithm for ≥

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

Bridge Rule for = and ≥

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

Solver Cooperation, Combination of Algorithms

Gaussian Elimination for $=$

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

Fouriers Algorithm for $\geq$

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

Bridge Rule for $=$ and $\geq$

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

Syntactic Unification

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

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

Syntactic Unification II

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

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

Syntactic Unification II

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

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

Syntactic Unification II

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

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

Syntactic Unification II

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

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

Syntactic Unification II

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

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

Syntactic Unification II

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

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

Syntactic Unification II

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

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

Part III

...Around the World

...Around the World

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

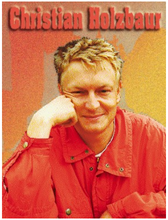
- Tom Schrijvers, David S. Warren, CHR and Tabled Execution, 20th ICLP 2004. Best Technical Paper Award.
- Gregory J. Duck, Christian Holzbaur, Maria Garcia de la Banda, Peter J. Stuckey, Optimizing Compilation of CHR in HAL, TPLP CHR Special Issue 2005.
  Extending Arbitrary Solvers with CHR, 5th ACM SIGPLAN PPDP'03.
- Armin Wolf, Adaptive Constraint Handling with CHR in Java, CP 2001, LNCS 2239.
  Intelligent Search Strategies Based on Adaptive CHR, TPLP CHR Special Issue 2005.
- Christian Holzbaur, Thom Frühwirth, A Prolog CHR Compiler and Runtime System, Applied Artificial Intelligence Vol 14(4), 2000.
- Slim Abdennadher et. al. JCK: A Java Constraint Kit, ENTCS Vol 64, 2000.

Hard Core CHR People

Slim Abdennadher

Tom Schrijvers

Peter Stuckey



Christian Holzbaur

Armin Wolf

More Semantics

- The Refined Operational Semantics of CHR, Gregory J. Duck, Peter J. Stuckey, Maria Garcia de la Banda, Christian Holzbaur, ICLP'04. *Textual rule order. Left-to-right execution of queries.*
- A Linear Logic Semantics for CHR, Hariolf Betz, Master Thesis, Ulm, 2005. *Model change: dynamic resources, actions and states.*
  ```
  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.
- Schedulers and Redundancy for a Class of Constraint Propagation Rules, Sebastian Brand, Krzysztof Apt, TPLP CHR Special Issue 2005.
- Automatic Rule Generation, Eric Monfroy, Valparaiso, Chile.

Program Transformation

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

Soft Constraints

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

E.g. Fuzzy Constraints:
X≤Y:A, Y≤Z:B ==> X≤Z:A*B

E≤F:0.5, F≤G:0.3 ↦
E≤F:0.5, F≤G:0.3, E≤G:0.6

Randomized Algorithms

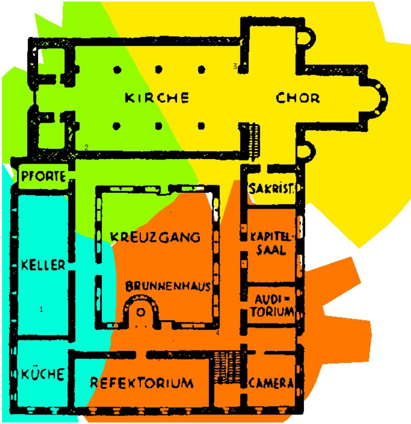
Probabilistic Constraint Handling Rules, T. Frühwirth, A. Di Pierro, H. Wiklicky, WFLP 2002, ENTCS. *CHR with randomized rule choice.*

Random walk

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

Probabilistic termination.

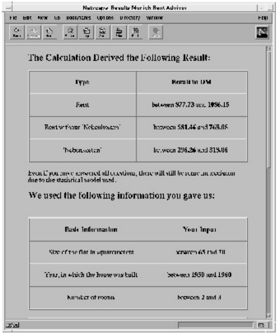
POPULAR - Planning Cordless Communication



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

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

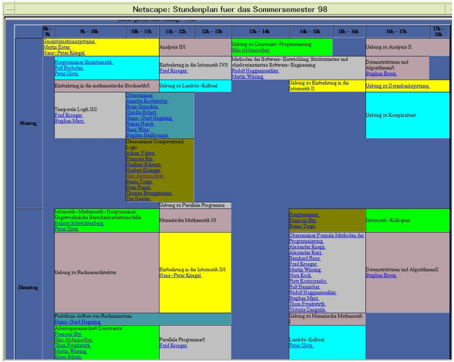
MRA - The Munich Rent Advisor



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

Most Popular Constraint-Based Internet Application.

University Course Timetabling



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

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

Reasoning Services

...System for Generation and Confirmation of Hypotheses,
Alberti, Chesani, Gavanelli, Lamma, W(C)LP 2005.
Extensions of Fung/Kowalksi 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.
$\text{CHR}^{\vee}$: A Flexible Query Language,
S. Abdennadher, H. Schütz, FQAS'98, LNCS.
CHR + disjunction = abduction, bottom-up/top-down evalution...

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

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

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

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

can be improved by adding the following rule

```
append(X,[],Z) ⇔ X = Z.
```

Top-down Evaluation with Tabling

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

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

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

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)` $\vee$ `penguin(X).`

Penguins can't fly.

`penguin(X)` $\wedge$ `flies(X)` $\Leftrightarrow$ `false.`

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

Bottom-up evaluation of logic programs

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

is transformed into

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

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

Bottom-up evaluation of logic programs

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

is transformed into

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

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

Bottom-up evaluation of logic programs

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

is transformed into

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

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

$$\downarrow$$

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

$$\downarrow$$

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

$$\downarrow$$

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

Bottom-up evaluation of logic programs

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

is transformed into

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

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

Bottom-up evaluation of logic programs

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

is transformed into

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

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

Bottom-up evaluation of logic programs

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

is transformed into

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

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

Bottom-up evaluation of logic programs

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

is transformed into

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

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

$$\downarrow$$

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

$$\downarrow$$

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

$$\downarrow$$

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

Bottom-up evaluation of logic programs

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

is transformed into

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

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

$$\downarrow$$

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

$$\downarrow$$

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

$$\downarrow$$

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

Bottom-up evaluation of logic programs

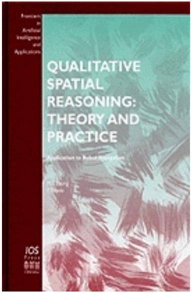
$$
\begin{array}{rcl}
p(X,Y) & \leftarrow & e(X,Y). \\
p(X,Y) & \leftarrow & e(X,Z) \wedge p(Z,Y).
\end{array}
$$

is transformed into

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

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

Spatio-Temporal Reasoning



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

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

Agents and Actions

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

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

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

Logical Algorithms

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).
```

Logical Algorithms

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)).
```

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 $\wedge$ word(T+R) $\Rightarrow$ parses(U,T+R,R).

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

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

Types and Security

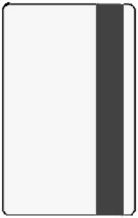
Chameleon Project, Martin Sulzmann, Peter J. Stuckey.

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

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

TypeTool - A Type Inference Visualization Tool, Sandra Alves, Mario Florido, WF(C)LP 2004; Type Inference with CHR, WF(C)LP 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. Kriebel,
TU Munich,
Journal on Software Tools for Technology Transfer
(STTT) 5:2-3, Springer 2004.

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

Semantic Web

The COntext INterchange *Project*

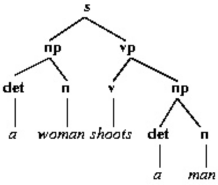
COIN Context Interchange Project,
Stuart E. Madnick, MIT Cambridge.
Reasoning About Temporal Context Using Ontology and Abductive CLP,
PPSWR 2004 LNCS 3208.

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

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

Computational Linguistics

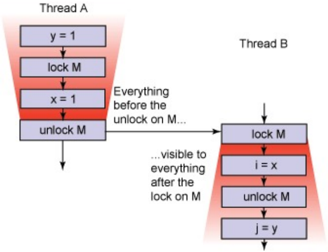
Coordination Revisited: A CHR Approach, Veronica Dahl et. al., LNCS 3315, 2004.
CHR Grammars, Henning Christiansen, TPLP CHR Special Issue 2005.
Assumptions and Abduction in Prolog, H. Christiansen, V. Dahl, WLPE 2004.
Abduction, Assumption Grammars.
Topological Parsing, Gerald Penn et. al., EACL'03.
HPSG, 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



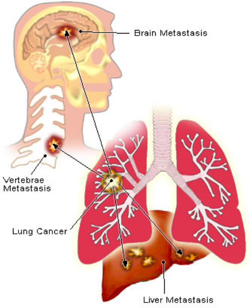
Conditional Read

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

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

Lung Cancer Diagnosis

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



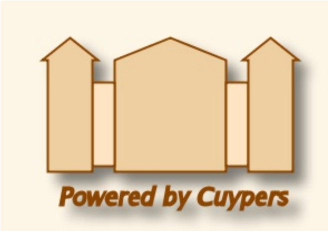
Lung Cancer and Metastasis

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

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

Multimedia Transformation Engine for Web Presentations

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

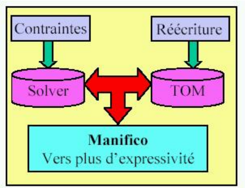


Business Rules for Optimization

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

RÉPUBLIQUE FRANÇAISE MINISTÈRE DE L'ÉCONOMIE DES FINANCES ET DE L'INDUSTRIE — ILOG — Loria — INRIA — RÉPUBLIQUE FRANÇAISE MINISTERE DE LA RECHERCHE

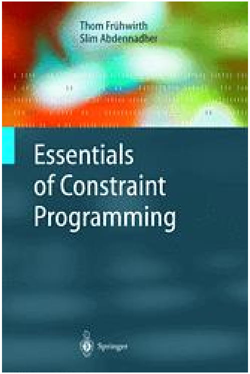
Business Rule: defines or constrains behavior or structure of business.
"A car must be available to be assigned to a rental agreement".



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

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

Further Reading



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

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