Constraint Handling Rules - What Else?

Thom Frühwirth

University of Ulm, Germany

What Researchers Say About CHR

- *One of the most powerful multiset rewriting languages.*
  Kazunori Ueda, Waseda University, Japan
- *Consistently outperforms Rete-based rule-based systems.*
  Peter Van Weert, K.U. Leuven
- *Significant speed up when executed on multi-core systems.*
  Edmund S. L. Lam, National University of Singapore
- *Lingua franca, a hub which collects and dispenses research efforts from and to the various related fields.*
  Jon Sneyers, K.U. Leuven

Minimum I

Minimum program

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

- Computing minimum of multiset of numbers n_i
- Numbers given as query `min(`n_1`)`, `min(`n_2`)`, ..., `min(`n_k`)`
- `min(`n_i`)` means n_i is potential minimum
- Simpagation rule takes two `min` constraints and removes the one representing the larger value.
- Program continues until only one `min` constraint left
- This `min` constraint represents smallest value

Minimum II

Minimum program

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

- Rule corresponds to intuitive algorithm:
  “Cross out larger numbers until one, the minimum remains”
- Illustrates use of multi-headed rule to iterate over data
  - No explicit loops or recursion needed
  - Keeps program code compact
  - Makes program easier to analyze

Greatest common divisor (I)

GCD program

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

- Computes greatest common divisor of natural number represented as `gcd(N)`
- Result is remaining nonzero `gcd` constraint

Example computation

```
gcd(12),gcd(8)
gcd(8), gcd(4)
gcd(4), gcd(4)
gcd(4), gcd(0)
```

Prime sieve

Prime sieve (I)

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

- Rule removes multiples of each of the numbers
- Query: Prime number candidates from `2` to up to `N`
  i.e. `prime(2),prime(3),prime(4),...prime(N)`
- Each number absorbs multiples of itself, eventually only prime numbers remain

Example computation

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

Constraint Handling Rules (CHR)

Concurrent declarative programming language and versatile computational formalism as well

- Semantic foundation in classical and linear logic
- Efficient sequential and parallel execution model
- Guaranteed properties such as anytime and online algorithm properties
- Powerful analysis methods for deciding e.g. program equivalence

Part I

The CHR Language

1. The CHR Language
2. Operational Properties
3. Program Analysis

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Example Partial Order Constraint

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

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

Syntax and Declarative Semantics

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

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

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

Operational Semantics

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

Simplify

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

Propagate

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

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

Operational Semantics

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

Simplify

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

Propagate

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

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

Operational Semantics

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

Simplify

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

Propagate

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

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

Operational Semantics

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

Simplify

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

Propagate

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

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

Properties of CHR programs

Guaranteed properties

- Anytime approximation algorithm
- Online incremental algorithm
- Concurrent/Parallel execution

Analyzable properties

- Termination/Time Complexity (semi-automatic)
- Determinism/Confluence (decidable)
- Program Equivalence (decidable!)

Anytime Algorithm - Approximation

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

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

Anytime Algorithm - Approximation

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

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

Anytime Algorithm - Approximation

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

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

Online Algorithm - Incremental

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

Monotonicity and Incrementality

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 - Incremental

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 - Incremental

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 - Incremental

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	$G \wedge C$	$\longmapsto$	$G' \wedge C$

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

Concurrency - Strong Parallelism

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

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

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

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

Concurrency - Strong Parallelism

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

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

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

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

Concurrency - Strong Parallelism

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

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

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

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

Optimal Time and Space Complexity

c

Jon Sneyers, K.U. Leuven

The CHR Machine

Sublanguage of CHR.

Can be mapped to Turing machines and vice versa.

CHR is Turing-complete.

Can be mapped to RAM machines and vice versa.

Every algorithm can be implemented in CHR with best known time and space complexity.

[Sneyers,Schrijvers,Demoen, CHR'05]

Practical Evidence: Union-Find, Shortest Paths, Fibonacci Heap Algorithms.

Efficiency - Better Time and Space Complexity

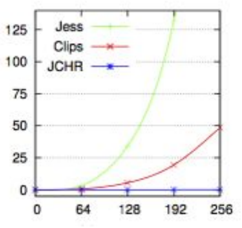
©pixabay

- CHR with mode declarations has optimal time and space complexity.
- JESS too, but 10-100 times slower.
- Prolog, Maude, Haskell not optimal if pure.
- CHR within one order of magnitude of best implementations in any other language (C,. . .).

Sneyers et.al., The computational power and complexity of Constraint Handling Rules, ACM TOPLAS 31(2) 2009.

Efficiency - The orders of magnitude

Up to one Million rules per second with CHR in C



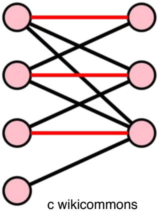
©Van Weert

JESS, CLIPS — hours
x 10-100
JCHR 2.0, CCHR — minutes
x 10-100
CHR in FPGA Hardware — seconds

Van Weert, Efficient lazy evaluation of rule-based programs, IEEE TKDE 2010.
Triossi, Compiling CHR to parallel hardware, ACM PPDP 2012.
Wuille, CCHR: the fastest CHR implementation, CHR'07.

Efficiency - Superior Implementation Techniques

Faster and faster Algorithms for matching facts to rules



c wikicommons

Eager Matching

- 1982 RETE: with join indexing.
- 1987 TREAT: without join indexing.

Lazy Matching

- 1990 LEAPS: with shadowing.
- 2000 CHR: with propagation history.

Van Weert, Efficient lazy evaluation of rule-based programs, IEEE TKDE 2010.

CHR Basic Compilation Scheme

`r: H1,...,Hm\...,Hn <=> G1,...,Gl | B1,...,Bk.`
`Hi` is one of `H1,...,Hn` and the j-th occurrence in the program

```
procedure occurrence_Hi_j(Hi,IDi)
 foreach (H1,ID1) in lookup(H1)
  :                    // except Hi
  foreach (Hn,IDn) in lookup(Hn)
   if alive(ID1) and...and alive(IDn)
    if all_different(ID1,...,IDn)
     if G1 and...and Gl
      if not in_history(r,ID1,...,IDn)
       add_to_history(r,ID1,...,IDn);
       kill(ID1);...;kill(IDm);
       create(B1,IDB1);...;create(Bk,IDBk);
       activate(B1,IDB1);...;activate(Bk,IDBk);
       if not alive(IDi) return true
      end
  :
 end
```

Common Compiler Optimizations

c wikicommons

Fact Invariants
- Set semantics
- Functional Dependencies

Join Computation
- Fact indexing
- Backjumping
- Loop-invariant code motion
- Non-robust iterators
- Join ordering

Fact Base
- Late indexing
- In-place modifications

Fact Activation
- Scheduling
- Passive occurrences
- Retraction preference
- Reapplication prevention

Program Specialization
- Class specialization
- Guard simplification

CHR Program Analysis

Prove Program Properties

Termination
Every computation starting from any goal ends.

Semi-Automatic **Complexity**
Worst-case time complexity follows from structure of rules.

Consistency and Correctness
Logical reading of rules is consistent, follows from a specification.

Decidable **Confluence**
The answer of a query is always the same, no matter which of the applicable rules are applied.

Completion Algorithm
Non-confluent programs made confluent by adding rules.

Decidable **Operational Equivalence**
Two programs have the same results for any given query.

Minimal States

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

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

Minimal State: $H \wedge C$

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

Minimal States

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

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

Minimal State: $H \wedge C$

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

Minimal States

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

Rule:	$H \Leftrightarrow C \mid B$ or $H \Rightarrow C \mid B$
Minimal State:	$H \wedge C$

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

Confluence

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

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

Start from overlapping minimal states

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

Confluence

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

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

Start from overlapping minimal states

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.

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

Start from overlapping minimal states

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

Completion

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

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

$$A \leq B \wedge B \leq A \wedge B < A$$

antisymmetry $\swarrow$ $\searrow$ inconsistency

$$A = B \wedge B < A \qquad\qquad B \leq A \wedge \mathit{false}$$
$$\downarrow \qquad\qquad\qquad \downarrow$$
$$A = B \wedge A < A \qquad\qquad \mathit{false}$$

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

Completion

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

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

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

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

Completion

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

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

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

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

Completion

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

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

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

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

Operational Equivalence

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

$$
\begin{array}{lllll}
P1 & min(X,Y,Z) \Leftrightarrow & X \leq Y & | & Z = X. \\
 & min(X,Y,Z) \Leftrightarrow & X > Y & | & Z = Y. \\
 & & & & \\
P2 & min(X,Y,Z) \Leftrightarrow & X < Y & | & Z = X. \\
 & min(X,Y,Z) \Leftrightarrow & X \geq Y & | & Z = Y.
\end{array}
$$

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

Operational Equivalence

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

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

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

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

Operational Equivalence

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

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

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

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

Part II

Example Programs

Mergers and acquisitions

- Sum up values:

```
sum(Value1), sum(Value2) <=>
    sum(Value1+Value2).
```

Mergers and acquisitions

- Sum up values:

```
sum(Value1), sum(Value2) <=>
    sum(Value1+Value2).
```

- CHR constraint `company(Name,Value)` represents company with market value `Value`
- Larger company buys smaller company:

```
company(Name1,Value1), company(Name2,Value2) <=>
    Value1>Value2 | company(Name1,Value1+Value2).
```

Computational Logic Programming

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

Top-down Goal-Driven Evaluation

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

Computational Logic Programming

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

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

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

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

Computational Logic Programming

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

Bottom-up Data-Driven Evaluation

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

Computational Logic Programming

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

Bottom-up Data-Driven Evaluation with Termination

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

Computational Logic Programming

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

Bottom-up Data-Driven Evaluation, Two Results Only

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

Sorting

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

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

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

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

Sorting

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

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

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

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

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

Combination of Gauss' and Fouriers Algorithms

Gaussian Elimination for $=$

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

Fouriers Algorithm for $\geq$

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

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

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

Combination of Gauss' and Fouriers Algorithms

Gaussian Elimination for $=$

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

Fouriers Algorithm for $\geq$

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

Bridge Rule for $=$ and $\geq$

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

Combination of Gauss' and Fouriers Algorithms

Gaussian Elimination for $=$

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

Fouriers Algorithm for $\geq$

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

Bridge Rule for $=$ and $\geq$

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

Description Logic with Rules in CHR

Straightforward integration of DL, rules and constraints.

`and`:	**If**	$x : C_1 \sqcap C_2 \in \mathcal{A}$ and $\{x : C_1, x : C_2\} \not\subseteq \mathcal{A}$
	then	$\mathcal{A} \rightarrow_{\sqcap} \mathcal{A} \cup \{x : C_1, x : C_2\}$
`or`:	**If**	$x : C_1 \sqcup C_2 \in \mathcal{A}$ and $\{x : C_1, x : C_2\} \cap \mathcal{A} = \emptyset$
	then	$\mathcal{A} \rightarrow_{\sqcup} \mathcal{A} \cup \{x : D\}$ for some $D \in \{C_1, C_2\}$
`some`:	**If**	$x : \exists R.D \in \mathcal{A}$ and there is no y with $\{(x, y) : R, y : D\} \subseteq \mathcal{A}$
	then	$\mathcal{A} \rightarrow_{\exists} \mathcal{A} \cup \{(x, y) : R,\ y : D\}$ for a fresh individual y
`all`:	**If**	$x : \forall R.D \in \mathcal{A}$ and there is a y with $(x, y) : R \in \mathcal{A}$ and $y : D \notin \mathcal{A}$
	then	$\mathcal{A} \rightarrow_{\forall} \mathcal{A} \cup \{y : D\}$

Figure: The completion rules for $\mathcal{ALC}$

Description Logic with Rules in CHR

Straightforward integration of DL, rules and constraints.
DL in CHR: shorter than formal specification!
Correct, confluent, concurrent, anytime, online algorithm.

```
and   @ I:S1 and S2 <=> I:S1, I:S2
or    @ I:S1 or S2 <=> (I:S1 ; I:S2)
some  @ I:some R is S <=> (I,J):R, J:S
all   @ I:all R is S, (I,J):R ==> J:S
```

Figure: CHR Rules for $\mathcal{ALC}$

Description Logic with Rules in CHR

Straightforward integration of DL, rules and constraints.
DL in CHR: shorter than formal specification!
Correct, confluent, concurrent, anytime, online algorithm.

```
and   @ I:S1 and S2 <=> I:S1, I:S2
or    @ I:S1 or S2 <=> (I:S1 ; I:S2)
some  @ I:some R is S <=> (I,J):R, J:S
all   @ I:all R is S, (I,J):R ==> J:S
```

Figure: CHR Rules for $\mathcal{ALC}$

Easily combine DL with CHR rules (like SWRL)
E.g. the `uncle` role (male sibling of person's father):
`Z:male, (Y,Z):hassibling, (X,Y):hasparent ==> (X,Z):hasuncle.`

Part III

Applications

CHR Research Application Domains

- **Programming** type systems, algorithm design, verification and testing,
- **Constraints** constraint solving and reasoning,
- **Time** scheduling and planning, spatial and temporal reasoning,
- **Logic** logical reasoning, abduction, probabilistic reasoning,
- **Agents** agent-based systems, semantic web reasoning,
- **Languages** computational linguistics, grammars,
- **and many more** legal reasoning, cognitive system modelling, automatic music generation, game playing, bio-informatics, data mining, ...

Embedding Formalisms and Languages in CHR

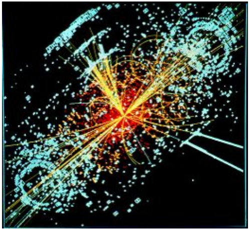
Embedding by straightforward source-to-source transformation:

- Term Rewriting Systems (TRS). Uses Equational Logic.
- Functional Programming (FP),
- General Abstract Model for Multiset Manipulation (GAMMA),
- Graph Transformation Systems (GTS),
- (Colored) Petri Nets (PN),
- Logical Algorithms (LA). Only known implementation. *Achieves the tight optimal time complexity.*
- Production Rules and Business Rules,
- Event-Condition-Action (ECA) Rules,
- Deductive Database languages like DATALOG,
- Description Logic (DL) with SWRL-style rules,
- Prolog and Constraint Logic Programming (CLP). Uses Clark's Completion.
- Concurrent Constraint Programming (CC) languages.

Online tool `http://pmx.informatik.uni-ulm.de/chr/translator`.

Logical Parallelism and Declarative Concurrency

c wikicommons



c wikicommons

Confluent programs can be executed in parallel without modification.

Often optimal linear speedup by parallelization (superlinear speedup e.g. for gcd algorithm).

Constant time sorting with CHR ultra-parallelism.

Implementations: Haskell, C++ on Nvidia CUDA, on FPGA Hardware.

Classical Algorithms: Union-Find, Preflow-Push.

Application: Particle collider data filtering with trigger rules at CERN.

Testing and Verification

Conditions, Assigments, Memory Locations modelled as CHR constraints:

- Symbolic execution along control-flow graphs
- Feasible paths computation and generalisation
- Automatic test data generation with heuristics

c wikicommons

Applications

Reasoning with data structures, e.g. arrays and heaps. Separation Logic for heap reasoning using SMCHR (Satisfiability Modulo Theories with CHR).

Verification of business processes, agents, web services.

Commercial Users: BSSE, Agitar, Logicblox. BSSE found mission-critical bug in satellite software.

Probabilistic Legal Reasoning

Legal argumentation: both parties make claims and use legal rules

c wikicommons

- Judge can accept claims and rules to be applicable or not
- Given probabilities of acceptance, what is the chance to win the case?
- Expressed in Probabilistic Argumentation Logic (a defeasible logic)
- Implemented in CHRISM (CHR with PRISM for probabilisitic reasoning and learning).

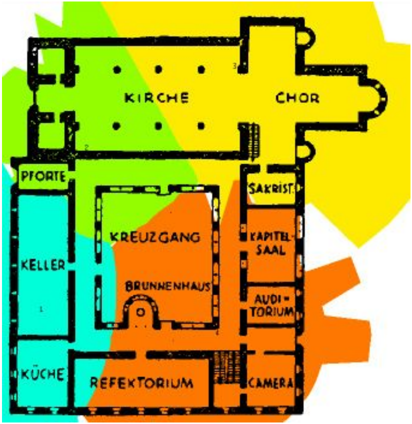
Sneyers et. al. Probabilistic legal reasoning in CHRiSM, TPLP 2013.

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.

c Joost Geurts

POPULAR - Planning Cordless Communication



c Thom Frühwirth, Pascal Brisset

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.

University Course Timetabling

c Slim Abdennadher

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.

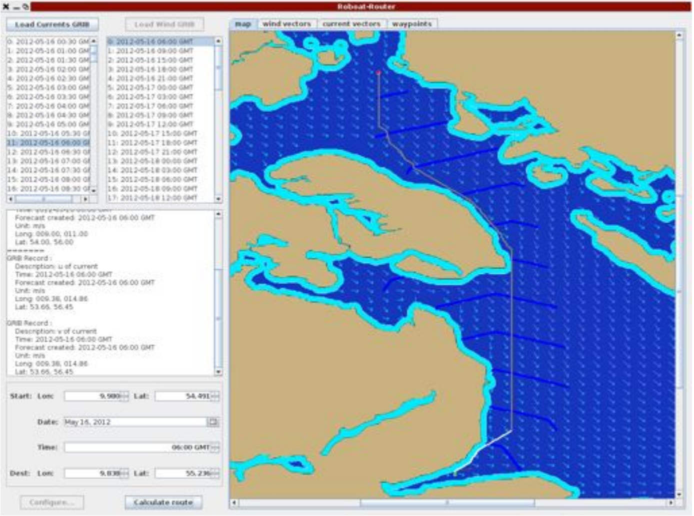
Multi-touch-enabled Music Generation&Manipulation

Uses probabilistic CHR (CHIRSM) for automatic music composition while learning musical styles.

Long-term Routing for Autonomous Sailboats

We work with the world-champion in robot sailing to set new world record in unassisted sailing: Planning in continuous domain with uncertain weather, sea currents.

Long-Term Routing using Weather and Current Forcests



Langbein, Stelzer, Frühwirth. Robotic Sailing 2011, Springer LNCS.

Industrial CHR Users

Stock Broking, New Zealand

Injection Mold Design, Canada

Optical Network Routing, USA

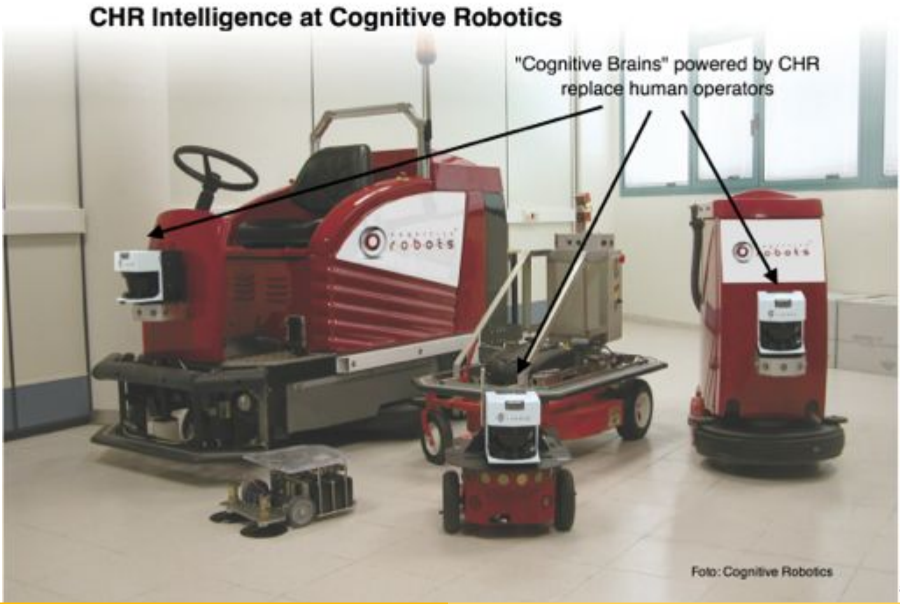
Test Case Generation, Germany

Unit Testing, USA

Knowledge Management, USA

Robotic Vehicle Control, Spain

Autonomous Vehicle Control



SecuritEase Stock-Broking Software

c wikicommons

- New Zealand (NZX) over 50% market share
- Australia (ASX) up to US $ 250 million, 50,000 trades per day
- CHR used for automatic trading, order acceptance checker, forms compiler, SQL query compiler

Constraint Handling Rules

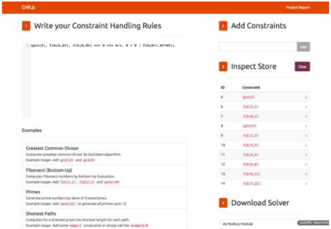
CHR Logo

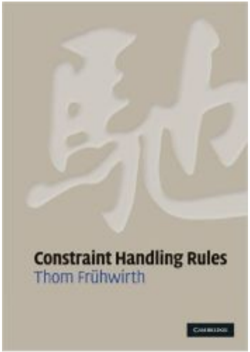
- Ultra-high-level formalism and programming language
- Integrated into host languages like Prolog, Java, C...
- Dozens of open-source implementations
- Naturally supports parallelism
- Online and anytime algorithm properties for free
- Analysis tools (termination, complexity, confluence)
- Faster than commercial rule-based systems
- Lingua Franca for Computation

Getting Started with Constraint Handling Rules

Search the Internet/Web for "Constraint Handling Rules"

Play with CHR at
`http://chrjs.net/`





Make your own rules.

馳

Constraint Handling Rules

Supplementary Material

Google "Constraint Handling Rules" for the CHR website

驰

Finally...

Google "Constraint Handling Rules" for the CHR website

驰

Transcribed as **CHR**, means

Finally...

Google "Constraint Handling Rules" for the CHR website

驰

Transcribed as **CHR**, means

to speed, to propagate, to be famous

Chemical Abstract Machine Style

One constraint. One Simpagation rule.

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

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

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

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

Chemical Abstract Machine Style

One constraint. One Simpagation rule.

```
min(N) \ min(M) ⇔ N=<M | true.

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

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

Chemical Abstract Machine Style

One constraint. One Simpagation rule.

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

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

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

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

Chemical Abstract Machine Style

One constraint. One Simpagation rule.

```
min(N) \ min(M) ⇔ N=<M | true.

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

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

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

Paths in a Graph

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

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

$$\downarrow\downarrow$$

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

$$\downarrow\downarrow$$

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

$$\downarrow\downarrow$$

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

Paths in a Graph

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

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

$$\downarrow\downarrow$$

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

$$\downarrow\downarrow$$

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

$$\downarrow\downarrow$$

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

Paths in a Graph

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

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

Paths in a Graph

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

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

Paths in a Graph

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

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

Paths in a Graph

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

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

Paths in a Graph

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

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

Paths in a Graph

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

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

Shortest Paths in a Graph

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

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

Shortest Paths in a Graph

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

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

$$\downarrow\downarrow$$

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

Shortest Paths in a Graph

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

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

Shortest Paths in a Graph

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

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

Exchange sort (I)

Exchange sort program

```
a(I,V), a(J,W) <=> I>J, V<W | a(I,W), a(J,V).
```

- Rule sorts array by exchanging values which are in wrong order
- Array is sequence of constraints `a(Index,Value)` i.e. `a(1, A1),...,a(n,An)`

Example computation

```
a(0,1), a(1,7), a(2,5), a(3,9), a(4,2)
a(0,1), a(1,5), a(2,7), a(3,2), a(4,9)
a(0,1), a(1,5), a(2,2), a(3,7), a(4,9)
a(0,1), a(1,2), a(2,5), a(3,7), a(4,9)
```

Linear Polynomial Equations

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

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

B=0  ⇔  number(B) |  zero(B).
```

Fourier's Algorithm

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

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

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.