First Steps towards a Lingua Franca for Computing: Rule-based Approaches in CHR

Thom Frühwirth | July 2009 | CHR 2009, Pasadena, CA, USA
The success of Constraint Handling Rules

CHR - an essential unifying computational formalism?

- CHR is a logic and a programming language
- CHR can express any algorithm with optimal complexity
- CHR supports reasoning and program analysis
- CHR programs are efficient and very fast
- CHR programs are anytime, online and concurrent algorithms
- CHR has many applications from academia to industry

⇒ CHR - a Lingua Franca for computer science:
* The first formalism and the first language for students
* Reasoning formalism and programming language for research
Renaissance of rule-based approaches

Results on rule-based system re-used and re-examined for

- Business rules and Workflow systems
- Semantic Web (e.g. validating forms, ontology reasoning, OWL)
- Aspect Oriented Programming
- UML (e.g. OCL invariants) and extensions (e.g. ATL)
- Computational Biology (e.g. protein folding, genome analysis)
- Medical Diagnosis
- Software Verification and Security (e.g. access policies)
Embedding rule-based approaches in CHR and vice versa

Conceptual level: source-to-source transformation (no interpreter, no compiler)

- Rule-based systems
  - Production Rules
  - Event-Condition-Action Rules
  - (Business Rules)
  - Logical Algorithms

- Rewriting- and graph-based formalisms
  - Term Rewriting and Functional Programming
  - (Graph Transformation Systems) and Multiset Transformation
  - (Colored Petri Nets)

- Logic- and constraint-based programming languages
  - Prolog and Constraint Logic Programming
  - Concurrent Constraint Programming
Embeddings in CHR

Advantages

CHR as *lingua franca* has to embed other approaches

- **Advantages of CHR for execution**
  - Efficiency, also optimal complexity possible
  - Abstract execution by constraints, even when arguments unknown
  - Incremental, anytime, online algorithms for free
  - Concurrent, parallel for confluent programs

- **Advantages of CHR for analysis**
  - Decidable confluence and operational equivalence
  - Estimating complexity semi-automatically
  - Logic-based declarative semantics for correctness

- *Embedding means comparison, cross-fertilization, transfer of ideas*
Rule-based systems

Use ground representation, no declarative semantics

- **Production rule systems (1980s)**
  - First rule-based systems
  - Imperative, destructive assignment

- **Event-Condition-Action (ECA) rules (mid 1990s)**
  - Extension of production rules for active database systems
  - Some aspects standardized in SQL-3

- **Business rules (since end of 1990s)**
  - Constrain structure and behavior of business
  - Describe operation of company and interaction with customers

- **Logical Algorithms formalism (early 2000)**
  - Hypothetical declarative production rule language
  - Overshadowing information instead of removal
OPS5 Translation

**Definition (Rule scheme for production rule)**

**OPS5 production rule**

\[(p \ N \ LHS \ --> \ RHS)\]

translates to CHR generalized simpagation rule

\[N @ LHS_1 \ \backslash \ \ LHS_2 \ \iff \ \ LHS_3 \ | \ RHS'\]

**LHS** left hand side (if-clause), **RHS** right hand side (then-clause)

- **LHS\(_1\):** patterns of LHS for facts not modified in RHS
- **LHS\(_2\):** patterns of LHS for facts modified in RHS
- **LHS\(_3\):** conditions of LHS
- **RHS\(_'\):** RHS without removal (for LHS\(_2\) facts)
Example (Greatest common divisor) (I)

Example (OPS5)

(p done-no-divisors
  (euclidean-pair ^first <first> ^second 1) -->
  (write GCD is 1) (halt) )

(p found-gcd
  (euclidean-pair ^first <first> ^second <first>) -->
  (write GCD is <first>) (halt) )

Example (CHR)

done-no-divisors @ euclidean_pair(First, 1) <=> write(GCD is 1).

found-gcd @ euclidean_pair(First, First) <=> write(GCD is First).
Example (Greatest common divisor) (II)

Example (OPS5)

\[(p\ switch-pair\n\{(euclidean-pair\ ^first\ <first>
\quad ^second\ \{\ <second>\ >\ <first>\}\ )\n\}<e-pair>\} -->
(modify\ <e-pair>\ ^first\ <second>
\quad ^second\ <first>)
(write\ <first>\ --\ <second>\ (crlf))\)

Example (CHR)

switch-pair @ euclidean_pair(First, Second) <= Second > First |
euclidean_pair(Second, First),
write(First--Second), nl.
Example (Greatest common divisor) (III)

Example (OPS5)

\[
\begin{align*}
(p \text{ reduce-pair} & \\
\{ (\text{euclidean-pair} \ ^\text{first} <\text{first}> \\
\ ^\text{second} \{ <\text{second}> < <\text{first}> \} ) \} & \rightarrow \\
\text{modify} <\text{e-pair}> \ ^\text{first} (\text{compute} <\text{first}>-<\text{second}>)) \\
\text{write} <\text{first}> -- <\text{second}> (\text{crlf}) \}
\end{align*}
\]

Example (CHR)

\[\text{reduce-pair} @ \text{euclidean\_pair}(\text{First}, \text{Second}) \iff \text{Second} < \text{First} \mid \text{euclidean\_pair}(\text{First}-\text{Second}, \text{Second}), \text{write}(\text{First}--\text{Second}), \text{nl.}\]
Negation-as-absence

Negated pattern in production rules

- Satisfied if no fact satisfies condition
- Violates monotonicity
- Semantics unclear

Example (Minimum in OPS5)

```prolog
(p minimum
  (num ^val <x>)
  -(num ^val < <x>)
  --> (make min ^val <x>) )
```

Negation-as-absence also used for ensuring termination and for default reasoning
CHR rules with negation-as-absence

Definition (Rule scheme for CHR rule with negation-as-absence)

CHR generalised simpagation rule

\[ N @ LHS1 \setminus LHS2 \rightarrow (\text{NEG1}, \text{NEG2}) \Leftrightarrow LHS3 \setminus \text{RHS} \]

translates to CHR rules

\[ N1 @ LHS1 \land LHS2 \Rightarrow LHS3 \setminus \text{check}(LHS1, LHS2) \]
\[ N2 @ \text{NEG1} \setminus \text{check}(LHS1, LHS2) \Leftrightarrow \text{NEG2} \setminus \text{true} \]
\[ N3 @ LHS1 \setminus LHS2 \land \text{check}(LHS1, LHS2) \Leftrightarrow \text{RHS} \]

- **NEG1** CHR constraints, **NEG2** built-in constraints
- **check**: auxiliary CHR constraint
- **Rule N2** must be tried before rule **N3** (refined semantics)
Embeddings of production rules with negation in CHR

Example (Minimum)
\[
\begin{align*}
\text{num}(X) & \implies \text{check}(\text{num}(X)). \\
\text{num}(Y) \setminus \text{check}(\text{num}(X)) & \implies Y < X \lor \text{true}. \\
\text{num}(X) \setminus \text{check}(\text{num}(X)) & \implies \text{min}(X).
\end{align*}
\]

Example (Termination: Transitive closure)
\[
\begin{align*}
\text{e}(X,Y) & \implies \text{check}(\text{e}(X,Y)). \\
\text{p}(X,Y) \setminus \text{check}(\text{e}(X,Y)) & \implies \text{true}. \\
\text{e}(X,Y) \setminus \text{check}(\text{e}(X,Y)) & \implies \text{p}(X,Y).
\end{align*}
\]

Example (Default Reasoning: Marital status)
\[
\begin{align*}
\text{person}(X) & \implies \text{check}(\text{person}(X)). \\
\text{married}(X) \setminus \text{check}(\text{person}(X)) & \implies \text{true}. \\
\text{person}(X) \setminus \text{check}(\text{person}(X)) & \implies \text{single}(X).
\end{align*}
\]
CHR propagation rules with negation-as-absence

Assume negative part holds, otherwise repair later

- Use \texttt{RHS} directly instead of auxiliary check
- Works if \texttt{RHS} nonempty, no built-ins, contains head variables

Definition (Rule scheme for CHR propagation rule with negation)

CHR propagation rule

\[ \begin{align*}
N & \@ \text{LHS}_1 - (\text{NEG}_1, \text{NEG}_2) \Rightarrow \text{LHS}_3 \mid \text{RHS} \\
\end{align*} \]

translates to CHR rules

\[ \begin{align*}
N_2 & \@ \text{NEG}_1 \ \backslash \ \text{RHS} \Leftrightarrow \text{NEG}_2 \mid \text{true} \\
N_1 & \@ \text{LHS}_1 \Rightarrow \text{LHS}_3 \mid \text{RHS} \\
\end{align*} \]

Rules are ordered: \texttt{N2} rules have to come before \texttt{N1} rules
Consequences and examples

- Shorter, more concise programs
- Often incremental, concurrent, declarative \(\Rightarrow\) easier analysis
- Negation often not needed (if we have propagation rules)

**Example (Minimum in CHR)**

\[
\begin{align*}
\text{min}(Y) \ \backslash \ \text{min}(X) & \iff Y < X \ | \ \text{true}.
\text{num}(X) & \implies \text{min}(X).
\end{align*}
\]

**Example (Transitive closure in CHR)**

\[
\begin{align*}
\text{p}(X,Y) \ \backslash \ \text{p}(X,Y) & \iff \text{true}.
\text{e}(X,Y) & \implies \text{p}(X,Y).
\end{align*}
\]

**Example (Marital Status in CHR)**

\[
\begin{align*}
\text{married}(X) \ \backslash \ \text{single}(X) & \iff \text{true}.
\text{person}(X) & \implies \text{single}(X).
\end{align*}
\]
Conflict resolution

Fire applicable rule with largest weight.

**Definition (Rule scheme for CHR rule with static or dynamic weight)**

Generalised simpagation rule (with weight, priority or probability \( P \))

\[
H_1 \setminus H_2 \Leftrightarrow LHS_3 | RHS : P
\]

translates to CHR rules

\[
\text{delay} \land H_1 \land H_2 \Rightarrow LHS_3 | \text{rule}(P,H_1,H_2)
\]

\[
\text{choose} @ \text{rule}(P_1,\_\_,\_\_) \setminus \text{rule}(P_2,\_\_,\_\_) \Leftrightarrow P_1 \geq P_2 | \text{true}
\]

\[
\text{apply} \land H_1 \setminus H_2 \land \text{rule}(P,H_1,H_2) \land \text{delay} \Leftrightarrow RHS \land \text{delay} \land \text{apply}
\]

- Phase constraint \text{delay}: finds applicable rules
- Rule \text{choose}: finds rule with maximum weight
- Phase constraint \text{apply}: executes chosen rule

Phase constraints \text{delay} \land \text{apply} present at end of query
Summary production rule systems in CHR

- **Negation-as-absence** and **conflict resolution** use similar translation scheme
- **Propagation and simpagation rules** come handy
- **Special case** of negation-as-absence avoids absence check

- **Phase constraints** avoid rule firing before conflict resolution
- Phase constraints rely on left-to-right evaluation order of queries
- Alternatively, rely on order of rules (CHR refined semantics)

- **Program sizes** are roughly proportional to each other
- **CHR complexity** with proper conflict resolution roughly as production rule program
Event Condition Action rules in CHR

Extension of production rules for active databases, generalise features like integrity constraints, triggers and view maintenance

ECA rules

on Event if Condition then Action

Definition (Rule scheme for database relation)

\( n \)-ary relation \( r \) generates CHR rules for database update events

ins @ insert(R) \( \Rightarrow \) R

del @ delete(P) \( \setminus \) R \( \Leftrightarrow \) match(P,R) \( | \) true

upd @ update(R,R1) \( \setminus \) R \( \Leftrightarrow \) R1

\((R= r(x_1, \ldots, x_n), R1= r(y_1, \ldots, y_n), x_i, y_j \text{ distinct variables})\)

match(P,R) holds if tuple R matches tuple pattern P

Additional generic rules to remove events (at end of program)
Example (Salary increase)

Limit employee’s salary increase by 10 %

▶ *Before* update happens (by rule upd)

Example

\[
\text{update}(\text{emp}(\text{Name}, S1), \text{emp}(\text{Name}, S2)) \iff S2 > S1 \times (1 + 0.1) \quad | \quad \text{update}(\text{emp}(\text{Name}, S1), \text{emp}(\text{Name}, S1 \times 1.1)).
\]

▶ *After* update happens (by rule upd)

Example

\[
\text{update}(\text{emp}(\text{Name}, S1), \text{emp}(\text{Name}, S2)) \iff S2 > S1 \times (1 + 0.1) \quad | \quad \text{update}(\text{emp}(\text{Name}, S2), \text{emp}(\text{Name}, S1 \times 1.1)).
\]

▶ Difference: first argument of *update* in the body
LA formalism

- Hypothetical bottom-up logic programming language
- Features explicit deletion of atoms and rule priorities
- Declarative production rule language, deductive database language, inference rules with deletion
- Designed to derive tight complexity results
- **The only implementation is in CHR**
- *It achieves the theoretically postulated complexity results!*
Embedding Logical Algorithms in CHR

Range-restricted ground bottom-up formalism with first-order logic syntax for rules, with rule priorities and explicit deletion of atoms

Definition (Rule scheme for LA rule)

LA rule

\[ r @ p : A \rightarrow C \]

translates to CHR propagation rule with priority

\[ r @ A_1 \Rightarrow A_2 \mid C : p \]

\((A_1: \text{atoms of } A, A_2: \text{comparisons of } A, C: \text{atoms}, p: \text{priority})\)

Priorities by CHR extension or conflict resolution
LA is set-based and has explicit deletion by special atom $\text{del}(A)$

**Definition (Rule scheme for LA predicate)**

$n$-ary LA predicate $a$ generates simpagation rules

\[
A \setminus A \iff \text{true} \quad \text{del}(A) \setminus \text{del}(A) \iff \text{true} \quad \text{del}(A) \setminus A \iff \text{true}
\]

($A = a(x_1, \ldots, x_n), x_i$ distinct variables)

But set-basedness needs more work...
Ensuring set-based semantics

Generation of new rule variants by unifying head constraints

Definition (Rule scheme for set-based semantics)

To CHR propagation rules

\[ H \land H_1 \land H_2 \Rightarrow G \mid B \]

add rules (if guard does not imply that \( H \land H_1 \) contains \( B \))

\[ H \land H_1 \Rightarrow H_1 = H_2 \land G \mid B \]

Example

\( a(1,Y), a(X,2) \Rightarrow b(X,Y) \).

Additional rule from unifying \( a(1,Y) \) and \( a(X,2) \)

\( a(1,2) \Rightarrow b(1,2) \).
LA example (Dijkstra’s shortest paths)

Example (Dijktra in LA)

\[
\begin{align*}
    d1 & @ 1: \text{source}(X) \rightarrow \text{dist}(X,0) \\
    d2 & @ 1: \text{dist}(X,N) \land \text{dist}(X,M) \land N<M \rightarrow \text{del}(\text{dist}(X,M)) \\
    dn & @ N+2: \text{dist}(X,N) \land \text{edge}(X,Y,M) \rightarrow \text{dist}(Y,N+M)
\end{align*}
\]

Example (Dijkstra in CHR)

\[
\begin{align*}
    \text{dist}(X,N) \backslash \text{dist}(X,N) & \Leftrightarrow \text{true}. \\
    \text{del}(\text{dist}(X,N)) \backslash \text{del}(\text{dist}(X,N)) & \Leftrightarrow \text{true}. \\
    \text{del}(\text{dist}(X,N)) \backslash \text{dist}(X,N) & \Leftrightarrow \text{true}. \\
    d1 & @ \text{source}(X) \Rightarrow \text{dist}(X,0) :1. \\
    d2 & @ \text{dist}(X,N), \text{dist}(X,M) \Rightarrow N<M \mid \text{del}(\text{dist}(X,M)) :1. \\
    dn & @ \text{dist}(X,N), \text{edge}(X,Y,M) \Rightarrow \text{dist}(Y,N+M) :N+2.
\end{align*}
\]
Rewriting-based and graph-based formalisms (I)

- Term rewriting systems (TRS)
  - Replace subterms given term according to rules until exhaustion
  - Formally based on equational logic
  - TRS analysis inspired CHR analysis (termination, confluence)
- Functional Programming (FP)
  - Basically syntactic fragment of TRS extended with built-ins
- Graph transformation systems (GTS)
  - Generalise TRS: graphs are rewritten under matching morphism

Translate to positive ground range-restricted CHR simplification rules over binary (and unary) CHR constraints
Term rewriting systems (TRS) and CHR

Principles

- Rewriting rules: directed equations between ground terms
- Rule application: Given a term, replace subterms that match lhs. of rule with rhs. of rule
- Rewriting until no further rule application is possible

Comparison to CHR

- TRS locally rewrite subterms at fixed position in one ground term (functional notation)
- CHR globally manipulates several constraints in multisets of constraints (relational notation)
- TRS rules: no built-ins, no guards, no logical variables
- TRS rules: restrictions on occurrences of pattern variables
Flattening

Transformation forms basis for embedding TRS (and FP) in CHR

- Opposite of variable elimination, introduce new variables
- Flattening function transforms atomic equality constraint $eq$ between nested terms into conjunction of flat equations

**Definition (Flattening function)**

$$[X \ eq \ T] := \begin{cases} X \ eq \ T & \text{if } T \text{ is a variable} \\ X \ eq \ f(X_1, \ldots, X_n) \land \bigwedge_{i=1}^{n} [X_i \ eq \ T_i] & \text{if } T = f(T_1, \ldots, T_n) \end{cases}$$

($X$ variable, $T$ term, $X_1 \ldots X_n$ new variables)
Embedding TRS in CHR

Definition (Rule scheme for term rewriting rule)

\[ S \rightarrow T \]

translates to CHR simplification rule

\[ [X \text{ eq } S] \Leftrightarrow [X \text{ eq } T] \]

Example (TRS Addition of natural numbers)

0+Y \rightarrow Y.

s (X) + Y \rightarrow s (X+Y).

Example (Translation to CHR)

T eq T1+T2, T1 eq 0, T2 eq Y \Leftrightarrow T eq Y.

T eq T1+T2, T1 eq s(T3), T3 eq X, T2 eq Y \Leftrightarrow

\[ T \text{ eq } s(T4), \ T4 \text{ eq } T5+T6, \ T5 \text{ eq } X, \ T6 \text{ eq } Y. \]
Functional programming (FP)

- FP can be seen as programming language based on TRS
  - Extended by built-in functions and guard tests
  - Matching only at outermost redex of lhs. of rewrite rule

**Definition (Rule scheme for functional program rule)**

\[ S \rightarrow G \mid T \]

translates to CHR simplification rule

\[ X \text{ eq } S \iff G \mid [X \text{ eq } T] \]

Additional generic rules for data and auxiliary functions

\[
\begin{align*}
X \text{ eq } T &\iff \text{datum}(T) \mid X=T. \\
X \text{ eq } T &\iff \text{builtin}(T) \mid \text{call}(T,X).
\end{align*}
\]

*(call(T,X) calls built-in function T, returns result in X)*
Example (Fibonacci Numbers)

Example (Fibonacci in FP)

\[
\begin{align*}
\text{fib}(0) & \rightarrow 1. \\
\text{fib}(1) & \rightarrow 1. \\
\text{fib}(N) & \rightarrow N \geq 2 \mid \text{fib}(N-1)+\text{fib}(N-2).
\end{align*}
\]

Example (Fibonacci in CHR)

\[
\begin{align*}
T & \equiv \text{fib}(0) \iff T=1. \\
T & \equiv \text{fib}(1) \iff T=1. \\
T & \equiv \text{fib}(N) \iff N \geq 2 \mid \text{call}(F1+F2,T), \\
& \quad F1 \equiv \text{fib}(N1), \text{call}(N-1,N1), \\
& \quad F2 \equiv \text{fib}(N2), \text{call}(N-2,N2).
\end{align*}
\]

(Generic rules for datum and built-in already applied in bodies)
GAMMA

- Based solely on multiset rewriting
- Chemical metaphor: molecules in solution react according to reaction rules
- Basis of Chemical Abstract Machine (CHAM)
- Reaction in parallel on disjoint sets of molecules

Definition (GAMMA)

- GAMMA program: pairs $(c/n, f/n)$ (predicate $c$, function $f$)
- $f$ applied to molecules for which $c$ holds
- Result $f(x_1, \ldots, x_n) = \{y_1, \ldots, y_m\}$ replaces $\{x_1, \ldots, x_n\}$ in $S$
GAMMA Translation

Molecules modeled as unary CHR constraints, reactions as rules

Definition (Rule scheme for GAMMA pair)

GAMMA pair \((c/n,f/n)\) translated to simplification rule

\[
d(x_1), \ldots, d(x_n) \iff c(x_1, \ldots, x_n) \mid f(x_1, \ldots, x_n),
\]

where \(f\) is defined by rules of the form

\[
f(x_1, \ldots, x_n) \iff G \mid D, d(y_1), \ldots, d(y_m),
\]

\((d\) wraps molecules, \(c\) built-in, \(G\) guard, \(D\) auxiliary built-ins)
GAMMA examples and translation into CHR

Example (Minimum)

\[
\text{min} = (X < Y, (X, Y) \setminus \{X\}) \quad d(X) \setminus d(Y) \iff X < Y \mid \text{true}.
\]

Example (Greatest Common Divisor)

\[
\text{gcd} = (X < Y, (X, Y) \setminus \{X, Y-X\}) \quad d(X) \setminus d(Y) \iff X < Y \mid d(Y-X).
\]

Example (Prime sieve)

\[
\text{prime} = (X \div Y, (X, Y) \setminus \{X\}) \quad d(X) \setminus d(Y) \iff X \div Y \mid \text{true}.
\]
Constraint logic programming translation to CHR

For pure Prolog and CLP without cut and negation-as-failure

Definition (Rule scheme for pure (C)LP clauses)

- CLP predicate $p/n$ is considered as CHR constraint
- For each predicate $p/n$ Clark’s completion of $p/n$ added as CHR\(\lor\) simplification rule

Example (Append in Prolog)

```prolog
append([], L, L) ← true.
append([H|L1], L2, [H|L3]) ← append(L1, L2, L3).
```

Example (Append in CHR\(\lor\))

```prolog
append(X, Y, Z) ⇔
(  X=[] \land Y=Z=L
\lor
  X=\[H\mid L1\] \land Y=L2 \land Z=[H\mid L3] \land append(L1, L2, L3) ).
```
Example (Prime sieve)

Comparison between Prolog and CHR by example

Example (Prime sieve in Prolog)

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

upto(F,T,[]):- F>T, !.
upto(F,T,[F|Ns1]):- F1 is F+1, upto(F1,T,Ns1).

sift([],[]).
sift([P|Ns],[P|Ps1]):- filter(Ns,P,Ns1), sift(Ns1,Ps1).
filter([],P,[]).
filter([X|In],P,Out):- X mod P =:= 0, !, filter(In,P,Out).
filter([X|In],P,[X|Out1]):- filter(In,P,Out1).
```

Prolog uses nonlogical cut operator.

Example (Prime sieve in CHR)

```prolog
upto(N) <=> N>1 | M is N-1, upto(M), prime(N).
sift @ prime(I) \ prime(J) <=> J mod I =:= 0 | true.
```
Example (Shortest path)

Comparison between Prolog and CHR by example

Example (Shortest path in Prolog)

```
p(From,To,Path,N) :- e(From,To,N).
p(From,To,Path,N) :- e(From,Via,1),
    not member(Via,Path),
    p(Via,To,[Via|Path],N1),
    N is N1+1.
shortestp(From,To,N) :- p(From,To,[],N),
    not (p(From,To,[],N1),N1<N).
```

Prolog uses nonlogical negation-as-failure.

Example (Shortest path in CHR)

```
p(X,Y,N) \ p(X,Y,M) <=> N=<M | true.
e(X,Y) ==> p(X,Y,1).
e(X,Y), p(Y,Z,N) ==> p(X,Z,N+1).
```
Concurrent constraint programming (CC)

- One of the frameworks closest to CHR
- CC permits don’t care and don’t know nondeterminisms
- We concentrate on the committed-choice fragment of CC (Based on don’t-care nondeterminism like CHR)

Definition (Abstract syntax of CC program)

CC program is a finite sequence of declarations.

- Declarations: \( D ::= p(\tilde{t}) \leftarrow A \mid D, D \)
- Agents: \( A ::= true \mid c \mid \sum_{i=1}^{n} c_i \rightarrow A_i \mid A || A \mid p(\tilde{t}) \)

(p user-defined predicate symbol, \( \tilde{t} \) sequence of terms, \( c \) and \( c_i \)’s constraints)
Translation

Definition (Rule scheme for CC expressions)

\[ D ::= p(\bar{t}) \iff A \mid D, D \]
\[ A ::= \text{true} \mid c \mid \text{ask}(\sum_{i=1}^{n} c_i \rightarrow A_i) \mid A \land A \mid p(\bar{t}) \]

For each Ask \( A = \sum_{i=1}^{n} c_i \rightarrow A_i \) of CC program generate \( n \) simplification rules for \text{ask} constraint

\[ \text{ask}(A) \iff c_i \mid A_i \ (1 \leq i \leq n). \]
Example (Maximum)

Example (Maximum in CC)
\[
\text{max}(X, Y, Z) \leftarrow (X \leq Y \rightarrow Y = Z) + (Y \leq X \rightarrow X = Z)
\]

Example (Maximum in CHR)
\[
\text{max}(X, Y, Z) \iff \text{ask}\left((X \leq Y \rightarrow Y = Z) + (Y \leq X \rightarrow X = Z)\right).
\]
\[
\text{ask}\left((X \leq Y \rightarrow Y = Z) + (Y \leq X \rightarrow X = Z)\right) \iff X \leq Y \mid Y = Z.
\]
\[
\text{ask}\left((X \leq Y \rightarrow Y = Z) + (Y \leq X \rightarrow X = Z)\right) \iff Y \leq X \mid X = Z.
\]

To simplify ask rules, replace generic \text{ask} by \text{ask}\_\text{max}(X, Y, Z)

Example (Simplified maximum in CHR)
\[
\text{ask}\_\text{max}(X, Y, Z) \iff X \leq Y \mid Y = Z.
\]
\[
\text{ask}\_\text{max}(X, Y, Z) \iff Y \leq X \mid X = Z.
\]
Summary: embedding rule-based approaches into CHR

- Logic- and constraint-based languages in **single-headed CHR**
  - Simplification rules

- Rule-based systems and formalisms in **positive ground range-restricted CHR**
  - *ground*: queries ground
  - *positive*: no built-ins in body of rule
  - *range-restricted*: variables in guard and body also in head

- These conditions imply
  - Every state in a computation is ground
  - CHR constraints do not delay and wake up
  - Guard entailment check is just test
  - Computations cannot fail
Useful CHR features to support embeddings

We used source-to-source transformation for features. Could also use dynamic CHR (justifications) or extensions of CHR for:

- **built-in “negation”** of rule-based systems
  ⇒ CHR with negation-as-absence

- **conflict resolution** of rule-based systems
  ⇒ CHR with priorities

- ignorance of **duplicates** in rule-based formalisms
  ⇒ CHR with set-based semantics

- **built-in search** of Prolog, constraint logic programming
  ⇒ $\text{CHR}^\lor$ with disjunction or search library

Most available in upcoming K.U. Leuven CHR.
Distinguishing CHR features to be embedded

*Unique combination of features makes embedding of CHR in other approaches hard*

- **Multiple Head Atoms** not in other programming languages
- **Propagation rules** only in Logical Algorithms
- **Constraints** only in constraint-based programming
  - Logical variables instead of ground representation
  - Constraints are reconsidered when new information arrives
  - Notion of failure due to built-in constraints
- **Logical Declarative Semantics** only in Prolog, CLP
  - CHR computations justified by logic reading of program
Embedding fragments of CHR in other rule-based approaches

Possibilities are rather limited (without interpreter or compiler)

- **Positive ground range-restricted fragment** embeddable into
  - Rule-based systems with negation and Logical Algorithms
  - Only simplification rules in Rewriting- and Graph-based formalisms

- **Single-headed rules with constraints** embeddable into
  - Concurrent constraint programming languages

Typically, no (efficient) implementations exist for those approaches
Conclusions

CHR as *lingua franca* can indeed embed rule-based approaches from theoretical to practical: systems, formalism, programming languages

▶ Positive ground range-restricted CHR fragment sufficient?
▶ What is the right syntax and terminology for CHR?
▶ Which features are good, which are bad?
▶ What are the issues, what are just technicalities?
▶ What about comparison and cross-fertilisation of embeddings?

*If you answer these questions, CHR has a exciting and bright future.*