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

驰

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

$\Rightarrow$ **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 costumers
- **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 @ LHS1 \ LHS2 ⇔ LHS3 | RHS'
```

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

- `LHS1`: patterns of `LHS` for facts not modified in `RHS`
- `LHS2`: patterns of `LHS` for facts modified in `RHS`
- `LHS3`: conditions of `LHS`
- `RHS'`: `RHS` without removal (for `LHS2` 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
    {(euclidean-pair ^first <first>
                     ^second { <second> > <first>} )
    <e-pair>} -->
    (modify <e-pair> ^first <second>
                     ^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)

```
(p reduce-pair
    {(euclidean-pair ^first <first>
                     ^second { <second> < <first> } )
    <e-pair>} -->
    (modify <e-pair> ^first (compute <first>-<second>))
    (write <first> -- <second> (crlf)) )
```

Example (CHR)

```
reduce-pair @ euclidean_pair(First, Second)) <=> Second < First |
    euclidean_pair(First-Second, Second),
    write(First--Second), nl.
```

Negation-as-absence

Negated pattern in production rules

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

Example (Minimum in OPS5)

```
(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

$\mathtt{N\ @\ LHS1\ \backslash\ LHS2\ -(NEG1,NEG2)}\ \Leftrightarrow\ \mathtt{LHS3}\ |\ \mathtt{RHS}$

translates to CHR rules

$\mathtt{N1\ @\ LHS1}\ \wedge\ \mathtt{LHS2}\ \Rightarrow\ \mathtt{LHS3}\ |\ \mathtt{check(LHS1,LHS2)}$

$\mathtt{N2\ @\ NEG1\ \backslash\ check(LHS1,LHS2)}\ \Leftrightarrow\ \mathtt{NEG2}\ |\ \mathit{true}$

$\mathtt{N3\ @\ LHS1\ \backslash\ LHS2}\ \wedge\ \mathtt{check(LHS1,LHS2)}\ \Leftrightarrow\ \mathtt{RHS}$

- `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)

```
num(X) ==> check(num(X)).
num(Y) \ check(num(X)) <=> Y<X | true.
num(X) \ check(num(X)) <=> min(X).
```

Example (Termination: Transitive closure)

```
e(X,Y) ==> check(e(X,Y)).
p(X,Y) \ check(e(X,Y)) <=> true.
e(X,Y) \ check(e(X,Y)) <=> p(X,Y).
```

Example (Default Reasoning: Marital status)

```
person(X) ==> check(person(X)).
married(X) \ check(person(X)) <=> true.
person(X)  \ check(person(X)) <=> single(X).
```

CHR propagation rules with negation-as-absence

Assume negative part holds, otherwise repair later

- Use `RHS` directly instead of auxiliary `check`
- Works if `RHS` nonempty, no built-ins, contains head variables

Definition (Rule scheme for CHR propagation rule with negation)

CHR propagation rule

```
N @ LHS1 -(NEG1,NEG2) ⇒ LHS3 | RHS
```

translates to CHR rules

```
N2 @ NEG1 \ RHS ⇔ NEG2 | true
N1 @ LHS1 ⇒ LHS3 | RHS
```

Rules are ordered: `N2` rules have to come before `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)

```
min(Y) \ min(X) <=> Y<X | true.
num(X) ==> min(X).
```

Example (Transitive closure in CHR)

```
p(X,Y) \ p(X,Y) <=> true.
e(X,Y) ==> p(X,Y).
```

Example (Marital Status in CHR)

```
married(X) \ single(X) <=> true.
person(X) ==> single(X).
```

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`)

`H1 \ H2 ⇔ LHS3 | RHS : P`

translates to CHR rules

```
delay ∧ H1 ∧ H2 ⇒ LHS3 | rule(P,H1,H2)
choose @ rule(P1,_,_) \ rule(P2,_,_) ⇔ P1≥P2 | true
apply ∧ H1 \ H2 ∧ rule(P,H1,H2) ∧ delay ⇔ RHS ∧ delay ∧ apply
```

- Phase constraint `delay`: finds applicable rules
- Rule `choose`: finds rule with maximum weight
- Phase constraint `apply`: executes chosen rule

Phase constraints `delay` ∧ `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 propertional 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 ative 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) ⇒ R
del @ delete(P) \ R ⇔ match(P,R) | true
upd @ update(R,R1) \ R ⇔ R1
```

(R$= r(x_1, \ldots, x_n)$, R1$= r(y_1, \ldots, y_n)$, x_i, y_j 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 happends (by rule `upd`)

Example

```
update(emp(Name,S1), emp(Name,S2)) <=> S2>S1*(1+0.1) |
              update(emp(Name,S1),emp(Name,S1*1.1)).
```

- *After* update happends (by rule `upd`)

Example

```
update(emp(Name,S1), emp(Name,S2)) <=> S2>S1*(1+0.1) |
              update(emp(Name,S2),emp(Name,S1*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

$$r \,@\, A_1 \Rightarrow A_2 \mid C : p$$

($A1$: atoms of A, $A2$: comparisons of A, C: atoms, p: priority)

Priorities by CHR extension or conflict resolution

Embedding Logical Algorithms in CHR (II)

LA is set-based and has explicit deletion by special atom $del(A)$

Definition (Rule scheme for LA predicate)

n-ary LA predicate a generates simpagation rules

`A \ A` $\Leftrightarrow$ *true*   `del(A) \ del(A)` $\Leftrightarrow$ *true*   `del(A) \ A` $\Leftrightarrow$ *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

$$H \wedge H_1 \wedge H_2 \Rightarrow G \mid B$$

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

$$H \wedge H_1 \Rightarrow H_1{=}H_2 \wedge G \mid B$$

Example

```
a(1,Y), a(X,2) ==> b(X,Y).
```

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

```
a(1,2) ==> b(1,2).
```

LA example (Dijkstra's shortest paths)

Example (Dijkstra in LA)

```
d1 @   1: source(X) ⟶ dist(X,0)
d2 @   1: dist(X,N) ∧ dist(X,M) ∧ N<M ⟶ del(dist(X,M))
dn @ N+2: dist(X,N) ∧ edge(X,Y,M) ⟶ dist(Y,N+M)
```

Example (Dijkstra in CHR)

```
dist(X,N) \ dist(X,N) <=> true.
del(dist(X,N)) \ del(dist(X,N)) <=> true.
del(dist(X,N)) \ dist(X,N) <=> true.

d1 @ source(X) ==> dist(X,0) :1.
d2 @ dist(X,N), dist(X,M) ==> N<M | del(dist(X,M)) :1.
dn @ dist(X,N), edge(X,Y,M) ==> dist(Y,N+M) :N+2.
```

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)

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

(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  -> Y.
s(X)+Y  -> s(X+Y).
```

Example (Translation to CHR)

```
T eq T1+T2, T1 eq 0, T2 eq Y <=> T eq Y.
T eq T1+T2, T1 eq s(T3), T3 eq X, T2 eq Y <=>
        T eq s(T4), T4 eq T5+T6, T5 eq X, T6 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

$$X \;\mathtt{eq}\; S \Leftrightarrow G \mid [X \;\mathtt{eq}\; T]$$

Additional generic rules for data and auxiliary functions

```
X eq T ⇔ datum(T) | X=T.
X eq T ⇔ builtin(T) | call(T,X).
```

(`call(T,X)` calls built-in function `T`, returns result in `X`)

Example (Fibonacci Numbers)

Example (Fibonacci in FP)

```
fib(0) -> 1.
fib(1) -> 1.
fib(N) -> N>=2 | fib(N-1)+fib(N-2).
```

Example (Fibonacci in CHR)

```
T eq fib(0) <=> T=1.
T eq fib(1) <=> T=1.
T eq fib(N) <=> N>=2 | call(F1+F2,T),
                        F1 eq fib(N1), call(N-1,N1),
                        F2 eq fib(N2), call(N-2,N2).
```

(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 witch 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

$$d(x_1), \ldots, d(x_n) \Leftrightarrow c(x_1, \ldots, x_n) \mid f(x_1, \ldots, x_n),$$

where f is defined by rules of the form

$$f(x_1, \ldots, x_n) \Leftrightarrow G \mid D, d(y_1), \ldots, d(y_m),$$

(d wraps molecules, c built-in, G guard, D auxiliary built-ins)

Can unfold f, and optimize to simpagation rules

GAMMA examples and translation into CHR

Example (Minimum)

```
min = (X<Y,(X,Y)\{X})            d(X) \ d(Y) <=> X<Y | true.
```

Example (Greatest Common Divisor)

```
gcd = (X<Y,(X,Y)\{X,Y-X})        d(X) \ d(Y) <=> X<Y | d(Y-X).
```

Example (Prime sieve)

```
prime = (X div Y,(X,Y)\{X})      d(X) \ d(Y) <=> X div Y | 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$^\vee$ simplification rule

Example (Append in Prolog)

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

Example (Append in CHR$^\vee$)

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

Example (Prime sieve)

Comparison between Prolog and CHR by example

Example (Prime sieve in 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)

```
       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.

$$
\begin{array}{lll}
\text{Declarations} & D ::= & p(\tilde{t}) \leftarrow A \mid D, D \\
\text{Agents} & A ::= & \mathit{true} \mid c \mid \sum_{i=1}^{n} c_i \rightarrow A_i \mid A \| A \mid p(\tilde{t})
\end{array}
$$

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

Translation

Definition (Rule scheme for CC expressions)

$$D ::= \quad p(\tilde{t}) \Leftrightarrow A \mid D, D$$
$$A ::= \quad \mathit{true} \mid c \mid \mathtt{ask}(\textstyle\sum_{i=1}^{n} c_i \to A_i) \mid A \wedge A \mid p(\tilde{t})$$

For each Ask $A = \sum_{i=1}^{n} c_i \to A_i$ of CC program generate n simplification rules for `ask` constraint

$$\mathtt{ask}(A) \Leftrightarrow c_i \,|\, A_i \;(1 \leq i \leq n).$$

Example (Maximum)

Example (Maximum in CC)

max(X,Y,Z) $\leftarrow$ (X$\leq$Y $\rightarrow$ Y=Z) + (Y$\leq$X $\rightarrow$ X=Z)

Example (Maximum in CHR)

max(X,Y,Z) $\Leftrightarrow$ ask((X$\leq$Y $\rightarrow$ Y=Z) + (Y$\leq$X $\rightarrow$ X=Z)).

ask((X$\leq$Y $\rightarrow$ Y=Z) + (Y$\leq$X $\rightarrow$ X=Z)) $\Leftrightarrow$ X$\leq$Y | Y=Z.
ask((X$\leq$Y $\rightarrow$ Y=Z) + (Y$\leq$X $\rightarrow$ X=Z)) $\Leftrightarrow$ Y$\leq$X | X=Z.

To simplify ask rules, replace generic `ask` by `ask_max(X,Y,Z)`

Example (Simplified maximum in CHR)

ask_max(X,Y,Z) $\Leftrightarrow$ X$\leq$Y | Y=Z.
ask_max(X,Y,Z) $\Leftrightarrow$ Y$\leq$X | X=Z.

Summary: embedding rule-based approaches into CHR

- Logic- and constraint-based languages in **single-headed CHR$^\vee$ 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
  $\Rightarrow$ CHR with negation-as-absence
- **conflict resolution** of rule-based systems
  $\Rightarrow$ CHR with priorities
- ignorance of **duplicates** in rule-based formalisms
  $\Rightarrow$ CHR with set-based semantics
- built-in **search** of Prolog, constraint logic programming
  $\Rightarrow$ $\mathrm{CHR}^{\vee}$ 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.