Constraint Handling Rules -
Getting started
Table of Contents

Getting started

How CHR works
CHR programs and their execution
Overview

- Basic introduction to CHR using examples
- Rule types and their behavior
- Logical variables and built-in constraints
- Concrete syntax
- Informal description of rule execution in CHR
CHR implementations

- Most recent and advanced implementation: K.U. Leuven (recommended)
- Programs also executable with minor changes in other Prolog implementations of CHR
- K.U. Leuven JCHR: CHR implementation in Java
- K.U. Leuven CHR library for C
- CHR code (declarations and rules) and host language statements mixed in programs
Declarations

Declarations introduce CHR constraints we will define by rules

Example (Declarations)

```prolog
:- module(weather, [rain/0]).
:- use_module(library(chr)).
:- chr_constraint rain/0, wet/0, umbrella/0.
```

- **Functor notation** $c/n$: name $c$, number of arguments $n$ of constraint $c(t_1, \ldots, t_n)$

- First line: optional Prolog module declaration: declares module `weather`, where only constraint `rain/0` is exported.

- Second line: loading CHR library

- Third line: Defines CHR constraints `rain`, `wet`, and `umbrella`
  - At least name and arity must be given
Rules (I)

- Parts of a rule:
  - Optional name
  - Left-hand side (l.h.s.) called head, with optional guard
  - Right-hand side (r.h.s) called body
- Head, guard, and body consist of constraints
- Three different kind of rules
Rules (II)

Example (Rules)

\[
\begin{align*}
\text{rain} & \implies \text{wet}. \\
\text{rain} & \implies \text{umbrella}.
\end{align*}
\]

- First rule: “If it rains, then it is wet”
- Second rule: “If it rains, we need an umbrella”
- Head of both rules is \text{rain}
- Bodies: \text{wet} and \text{umbrella}
- No guards
- Also called propagation rules (\implies)
  - Do not remove constraints, only add new ones
Queries

- Posing query initiates computations
- Rules applied to query until exhaustion (no more changes happen)
- Rule applications manipulate query by removing and adding constraints
- Result (called answer) consists of remaining constraints

Example (Query)

rain ==> wet.

rain ==> umbrella.

Posing query \textit{rain results in rain, wet, umbrella} (not necessarily in this order)
Top-down execution

- Rules applied in textual order
- In general: If more than one rule applicable, one rule is chosen
- Rule applications cannot be undone like in Prolog
  \[ \Rightarrow \text{CHR is a committed-choice language} \]

Example (Top-down execution)

Two simplification rules

\[
\text{rain} \iff \text{wet}.
\]
\[
\text{rain} \iff \text{umbrella}.
\]

- Application of first rule removes \text{rain}
- Second rule never applied
Simplification rules

- Propagation rules
  - Drawing conclusions from existing information

- Simplification rules
  - Simplify things
  - Express state change
  - Dynamic behavior
Example

Example (Walk)

► Walk expressed by movements `east, west, south, north`

► Multiplicity of steps matters, order does not matter for walk

► Simplification rules express that steps can cancel out each other (i.e. `east and west`)

  `east, west <= true.
  south, north <= true.``

► Rules simplify walk to one with minimal number of steps

► Query `east, south, west, west, south, south, north, east, east`

  yields answer `east, south, south`
Logical variables

- Featured in declarative languages like CHR
- Similar to mathematical unknowns and variables in logic
- Can be unbound or bound
- Bound variables indistinguishable from value they are bound to
- Bound variables cannot be overridden
- Languages with such variables called single-assignment languages
- Other languages like C and Java feature destructive (multiple) assignments
Example

▶ Two constraints with one argument representing men (e.g. `male(joe)`) and women (e.g. `female(sue)`)

▶ Assigning men and woman for dancing with simplification rule

\[
\text{male}(X), \text{female}(Y) \iff \text{pair}(X,Y).
\]

▶ Variables \(X, Y\) placeholders for values of constraints matching rule head

▶ Scope of variable is rule it appears in

▶ Given query with several men and women, rule pairs them until only people of one sex left
Types of rules

Example (Propagation rule)

- Computing all possible pairs with propagation rule (keeps male and female constraints)

\[
\text{male}(X), \quad \text{female}(Y) \implies \text{pair}(X,Y).
\]

- Number of pairs quadratic in number of people

⇒ Propagation rule can be expensive

Example (Simpagation rule)

- One man dances with several women expressed by simpagation rule

\[
\text{male}(X) \setminus \text{female}(Y) \iff \text{pair}(X,Y).
\]

- Head constraints left of backslash \ kept, head constraints right of backslash removed
Example (Family relationships (I))

- Propagation rule named \texttt{mm} expresses grandmother relationship
  
  \[ \texttt{mm @ mother}(X,Y), \texttt{mother}(Y,Z) \Rightarrow \texttt{grandmother}(X,Z). \]

- **Constraint** \texttt{grandmother}(joe,sue) reads as “Grandmother of Joe is Sue”

- Allows derivation of grandmother relationship from mother relationship

- \texttt{mother}(joe,ann), \texttt{mother}(ann,sue) will propagate \texttt{grandmother}(joe,sue) using rule \texttt{mm}
Built-in constraints

- Two kinds of constraints in CHR
- CHR constraints (user-defined constraints)
  - Declared in current program and defined by CHR rules
- Built-in constraints (built-ins)
  - Predefined in host language or imported CHR constraints from other modules
- On left hand side CHR and built-ins constraints separated into head and guard
- On right hand side freely mixed
Syntactic equality

Example (Family relationships (II))

- Mother of a person is unique, expressed by rule
  \[ \text{dm} @ \text{mother}(X,Y) \backslash \text{mother}(X,Z) \iff Y=Z. \]

- Syntactic equality: Mother relation is function, first argument determines second

- Rule enforces this using built-in syntactic equality =
  - Constraint \( Y=Z \) makes sure that both variables have the same value
  - Occurrences of one variable are replaced by (value of) other variable

- Query \text{mother}(joe,ann), \text{mother}(joe,ann) will lead to \text{mother}(joe,ann)
  - \( \text{ann} = \text{ann} \) simplified away, is always true
Failure

Example (Family relationship (III))

```prolog
dm @ mother(X, Y) \ mother(X, Z) <=> Y=Z.
```

- **Query** `mother(joe, ann), mother(joe, sue)` fails (Joe would have two different mothers)
- **Rule** `dm` will lead to `ann=sue`, which cannot be satisfied
- Built-in acts as test in this case

- Failure aborts computation
- Failure leads to answer `no` in most Prolog systems
Variables in queries and head matching

- Query can contain variables (matching successful as long as they are not bound by matching)

Example (Family relationship (IV))

\[
\text{mm } @ \text{ mother}(X,Y), \text{ mother}(Y,Z) \implies \text{ grandmother}(X,Z).
\]

- **Answer** \(\text{grandmother}(A,C)\) for query \(\text{mother}(A,B),\text{ mother}(B,C)\)

- **No rule applicable to** \(\text{mother}(A,B), \text{ mother}(C,D)\)

- **Answer** \(\text{grandmother}(A,D)\) when built-in added to query:
  \(\text{mother}(A,B), \text{ mother}(C,D), \text{ B=C}\)

- **Adding** \(A=D\) instead leads to \(\text{grandmother}(C,B)\)

- **Adding** \(A=C\) makes rule \(\text{dm}\) applicable, leads to \(\text{mother}(A,B), A=C, B=D\)
Example (I)

Example (Mergers and acquisitions)

- CHR constraint `company(Name, Value)` represents company with market value `Value`

- Larger company buys company with smaller value expressed by rule

  \[
  \text{company}(\text{Name1}, \text{Value1}), \text{company}(\text{Name2}, \text{Value2}) \leftrightarrow \text{Value1} > \text{Value2} \mid \text{company}(\text{Name1}, \text{Value1} + \text{Value2}).
  \]

- Guard `Value1 > Value2` acts as precondition of rule applicability

- Only built-ins allowed in guard
Example (II)

Example (Mergers and acquisitions cont.)

- In line arithmetic expression $\text{Value}_1 + \text{Value}_2$ works for host language Java.

- In Prolog `is` has to be used leading to rule:

```prolog
company(Name1,Value1), company(Name2,Value2) <=>
Value1 > Value2 | Value is Value1 + Value2, company(Name1:Name2,Value).
```

- Rule is applicable to any pair of companies with different value.

- After exhaustive only a few companies will remain (all with the same value).
Concrete Syntax

- CHR-specific part of program consists of declarations and rules
- Declarations are implementation-specific
- In following EBNF grammar:
  - Terminals in single quotes
  - Expressions ins square brackets optional
  - Alternatives separated by |
### Rules

Rule --> [Name '@']
(SimplificationRule | PropagationRule | SimpagationRule) ‘.’

SimplificationRule -->
Head ‘<=’ [Guard ‘|’] Body

PropagationRule -->
Head ‘==’ [Guard ‘|’] Body

SimpagationRule -->
Head ‘\’ Head ‘<=’ [Guard ‘|’] Body

- Three different types of rules in CHR
- ‘|’ separates guard from body of rule
- ‘\’ separates head of simpagation rule into two parts
### Rules

- **Head** → CHRConstraints
- **Guard** → BuiltInConstraints
- **Body** → Goal

#### CHRConstraints →
- CHRConstraint
- CHRConstraint , CHRConstraints

#### BuiltInConstraints →
- BuiltIn
- BuiltIn , BuiltInConstraints

#### Goal →
- CHRConstraint | BuiltIn | Goal , Goal

- **Query** → Goal

- **Head of rule is sequence of CHR constraints**
- **Guard is a sequence of built-ins constraints**
- **Body is a sequence of built-ins and CHR constraints**
Basic built-in constraints (I)

- Using set of predicates from host language Prolog
- Can be used for auxiliary computations in rule body
- Built-ins in guard of rule usually test (succeed or fail)

- Most basic built-ins
  - `true/0` always succeeds
  - `fail/0` never succeeds

- Testing if variables are bound
  - `var/1` tests if argument is unbound variable
  - `nonvar/1` tests if argument is bound variable
Basic built-in constraints (II)

- **Syntactical identity of expressions (infix):**
  - `/2` makes arguments syntactically identical by binding variables (fails if binding not possible)
  - `==/2` tests if arguments syntactically identical
  - `\=/2` tests if arguments syntactically different

- **Computing and comparing arithmetic expressions (infix):**
  - `is/2` binds first argument to value of arithmetic expression in the second argument (fails if not possible)
  - `</2, =</2, >/2, >=/2, =:=/2, =\=/2` test if arguments are arithmetic expressions whose values satisfy comparison
Basic built-in constraints (III)

- $=/2$ and $is/2$ bind first argument
  $\Rightarrow$ should never be used in guards
- Use $==$/$2$ and $=:=/2$ instead
- But some compilers make silent replacement
Informal semantics

- Description of current sequential implementation
- Based on so-called refined operational semantics
- Maybe different rule application in parallel, experimental and future implementations
- Those implementations will still respect so-called abstract operational semantics
Constraints

- Constraint is active operation as well as passive data
- Constraints in goals processed from left to right
- When CHR constraint encountered:
  - Evaluated like procedure call
  - Checks applicability of rules it appears in
  - Called active constraint
- Rules applied in textual order
- If no rule applicable to active constraint it becomes passive and is put in constraint store
- Passive constraints become active again context changes (their variables get bound)
Head matching

- One head constraint of rule is matched against active constraint
- Matching succeeds if constraint serves pattern
- Matching may bind variables in head (not in active constraint)
- If matching succeeds and rule head consists of more than one constraint, constraint store is searched for partner constraints to match other head constraints
Head constraints searched from left to right

Exception: simpagation rule
  - Constraints to be removed searched for before constraints to be kept are searched for

If matching succeeds, guard is checked

If several head constraints match active constraint, rule tried for each matching

If no successful matching exists, active constraint tries next rule
Guard checking

- Guard is precondition on rule applicability
- Test that either succeeds or fails
- If guard succeeds, rule is applied
- If guard fails, active constraint tries next head matching
Body execution

- When rule is applied, we say it fires
- Simplification rule: matching constraints removed, body executed
- Simpagation rule: similar to simplification rule but constraints matching head part preceding \ kept.
- Propagation rule: Body executed without removing any constraints
- Propagation rule will not fire with same constraint again
- According to rule type head constraints either called kept or removed
- Next rule tried when active constraint not removed