驰

Constraint Handling Rules -
Syntax and Semantics of CHR

Prof. Dr. Thom Frühwirth | 2009 | University of Ulm, Germany

Table of Contents

Constraint Handling Rules (CHR)

驰

CHR logo

- CHR is both: logical and practical
  - related to subset of first-order logic and linear logic
  - general-purpose programming like Prolog and Haskell
- Rules are descriptive and executable

Constraint Handling Rules (CHR)

- no distinction between data and operations
  - constraints cover both
- CHR is a language extension
  - Implementations available for Prolog, Haskell, C, Java, ...
  - in host language CHR constraints can be posted/inspected
  - in CHR rules host language statements can be used
- CHR is synthesis of
  - propagation rules
  - multiset transformation
  - logical variables
  - built-in constraints

with a formal foundation in logic and methods for powerful program analysis

CHR programming language

- for theorem proving and computational logic, integrating
  - forward and backward chaining
  - (integrity) constraints
  - deduction and abduction
  - tabulation
- as flexible production rule system with constraints
- as general-purpose concurrent constraint language

Available Distributions

More than a dozen free libraries to

- Prolog: SICStus, Yap, Eclipse, XSB, hProlog, HAL, SWI,...
- Java, also C
- Haskell, also parallel

Most advanced implementations from K.U. Leuven

Highlight Properties of CHR

Complexity

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

Algorithmic properties

Any CHR program will automatically implement a concurrent anytime (approximation) and online (incremental) algorithm.

Decidability

For terminating CHR programs confluence of rule applications and operational equivalence are decidable.

Overview

- **Syntax**: describes how constituents of a formal language are combined to form valid expressions
- **Semantics**:
  - **Operational**: Description of what it means to execute a statement (as transition system)
  - **Declarative**: Description of the meaning without referring to execution (in logic)
  - Goal: Corresponding operational and declarative semantics
- **Soundness**: Result of computation according to operational semantics is correct regarding declarative semantics
- **Completeness**: Everything proven by declarative semantics can be computed

Preliminaries

Syntactic expressions (I)

- **Signature**:
  - Set of variables $\mathcal{V}$
  - Set of function symbols Σ
  - Set of predicate symbols Π
- Function and predicate symbols have arity (number of arguments they take)
- **Functor** f/n: symbol f with arity n
- **Constants**: function symbols with arity zero
- **Propositions**: predicate symbols with arity zero

Syntactic expressions (II)

- **Term**: variable or function term $f(t_1, \ldots, t_n)$ ($f/n \in \Sigma$, t_i terms)
- **Atomic formula (atom)**: $p(t_1, \ldots, t_n)$ ($p/n \in \Pi$, t_i terms)
- **(Logical) expressions**: Terms and atoms; sets, multisets, and sequences (lists) of logical expressions

Substitution, instance and matching

Definition (Substitution)

Substitution $\theta : \mathcal{V} \to \mathcal{T}(\Sigma, \mathcal{V}')$: finite function from variables to terms
$\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ where each $X_i \neq t_i$
Identity substitution $\epsilon = \emptyset$
Extension to terms, $\theta : \mathcal{T}(\Sigma, \mathcal{V}) \to \mathcal{T}(\Sigma, \mathcal{V}')$
defined by implicit homomorphic extension,
$f(t_1, \ldots, t_n)\theta := f(t_1\theta, \ldots, t_n\theta)$

Substitution θ obtained by replacing each X_i in E with t_i at once.
Subsitutions written as postfix operators, applied from left to right.

Example – Substitution

Example

- $\theta = \{X/2, Y/5\}$: $(X * (Y + 1))\theta = 2 * (5 + 1)$
- $\theta = \{X/Y, Z/5\}$: $(X * (Z + 1))\theta = Y * (5 + 1)$
- $\theta = \{X/Y, Y/Z\}$: $p(X)\theta = p(Y) \neq p(X)\theta\theta = p(Z)$
- $\theta = \{X/Y\}, \tau = \{Y/2\}$:
  - $(X * (Y + 1))\theta\tau = (Y * (Y + 1))\tau = (2 * (2 + 1))$
  - $(X * (Y + 1))\tau\theta = (X * (2 + 1))\theta = (Y * (2 + 1))$

Instance, Renaming, Variants

Definition (Instance)

$E\theta$ is **instance** of E.

$E\theta$ **matches** E with **matching substitution** θ.

($\theta = \{X_1/t_1, \ldots, X_n/t_n\}$, E expression)

Definition (Variant, Variable Renaming)

If E and F are instances of each other then E and F are **variants** of each other.

Substitution θ is a **variable renaming** in $E = F\theta$.

Variable renaming θ is bijective, maps variables to variables.

- **Renamed apart**: Variants with no variables in common
- **Fresh variant**: Variant containing only new variables

Groundness

- Variables either **free** or **bound** (instantiated) to term
- **Ground, fixed (determined) variable**: bound or equivalent to ground term (variable is indistinguishable from the term it is bound to)
- **Ground expression**: Expression not containing (nonground) variables

Unification and syntactic equality

Unification: making expressions *syntactically equivalent* by substituting variables with terms.

Definition (Unifier)

Substitution θ is **unifier** of E and F if $E\theta = F\theta$.

E, F **unifiable**: unifier exists.

$\{p_1, \ldots, p_n\} = \{q_1, \ldots, q_m\}$ shorthand for $p_1 = q_1 \wedge \ldots \wedge p_n = q_n$ if $n = m$ and for false otherwise

Most General Unifier

Definition (Most General Unifier (MGU))

θ is **MGU** for E, F: every unifier τ for E, F is instance of θ, i.e., $\tau = \theta\rho$ for some ρ

(E, F expressions, θ, τ, ρ, θ_i substitutions)

Example – Most General Unifier

Example

$$f(X,a) \quad = \quad f(g(U),Y) \quad = \quad Z$$

MGU:

$$\theta = \{X/g(U), Y/a, Z/f(g(U),a)\}$$

Proof: $f(X,a)\theta = f(g(U),Y)\theta = Z\theta = f(g(U),a)$ one element.

Unifier, but not MGU:

$$\theta' = \{X/g(h(b)), U/h(b), Y/a, Z/f(g(h(b)),a)\}$$

Proof: $\theta' = \theta\{U/h(b)\}$.

Computing Most General Unifier

- Start with empty substitution ϵ
- scan terms simultaneously from left to right according to their structure
- check the syntactic equivalence of the terms encountered repeat
  - *different function symbols*: halt with failure
  - *identical function symbols*: continue
  - *one is unbound variable and other term*:
    - variable *occurs* in other term: halt with failure
    - apply the new substitution to the logical expressions

    add corresponding substitution
  - *variable is bound*: replace it by applying substitution

Example – Most General Unifier (2)

Example

Computing the MGU:

to unify	current substitution, remarks
$p(X,f(a)) = p(a,f(X))$	ϵ, start
$X = a$	$\{X/a\}$, substitution added
$f(a) = f(X)$	continue
$a = X$	$\{X/a\}$, variable is not unbound
$a = a$	continue

MGU is $\{X/a\}$

What about $p(X,f(b)) = p(a,f(X))$?

Example – Most General Unifier (3)

Example

s	t	θ
f	g	failure
a	a	ϵ
X	a	$\{X/a\}$
X	Y	$\{X/Y\}$, but also $\{Y/X\}$
$f(a,X)$	$f(Y,b)$	$\{Y/a, X/b\}$
$f(g(a,X),Y)$	$f(c,X)$	failure
$f(g(a,X),h(c))$	$f(g(a,b),Y)$	$\{X/b, Y/h(c)\}$
$f(g(a,X),h(Y))$	$f(g(a,b),Y)$	failure

Example – Most General Unifier (4)

Example

Examples involving cyclicity:

$X = X$ is unifiable but *not*:

- $X = f(X)$
- $X = p(A, f(X, a))$
- $X = Y \wedge X = f(Y)$

Clark's Equality Theory (CET)

Reflexivity	$(true \rightarrow X = X)$
Symmetry	$(X=Y \rightarrow Y=X)$
Transitivity	$(X=Y \wedge Y=Z \rightarrow X=Z)$
Compatibility	$(X_1=Y_1 \wedge \ldots \wedge X_n=Y_n \rightarrow f(X_1,\ldots,X_n)=f(Y_1,\ldots,Y_n))$
Decomposition	$(f(X_1,\ldots,X_n)=f(Y_1,\ldots,Y_n) \rightarrow X_1=Y_1 \wedge \ldots \wedge X_n=Y_n)$
Contradiction (Clash)	$(f(X_1,\ldots,X_n)=g(Y_1,\ldots,Y_m) \rightarrow false)$ if $f \neq g$ or $n \neq m$
Acyclicity	$(X=t \rightarrow false)$ if t is function term and X appears in t

(Σ signature with infinitely many functions, including at least one constant)

Theorems equality and matching

Theorem (Equality)

Expressions E and F are unifiable if and only if

$$CET \models \exists(E = F).$$

Theorem (Matching)

For expressions E, F and substitution $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$

$$CET \models \forall(E = F\theta \leftrightarrow (X_1 = t_1 \wedge \cdots \wedge X_n = t_n \rightarrow E = F)).$$

E matches F with substitution θ.

($\forall F$ denotes universal closure of formula F)

Constraint systems

- Constraints are distinguished predicates of first-order-logic
- Constraint systems take data types and operations and interpret expressions as constraints
- Data types: typically numbers are used to represent scalars, terms to represent structures

Definition constraint system

- Set of **constraint symbols**
- Set of values called **domain**
- Logical theory $\mathcal{CT}$ called **constraint theory**
    - consists of universally closed formulas (axioms)
    - must be nonempty and consistent
    - must include axiomatization for syntactic equality $=$ (CET) and the propositions $true$ (always holds) and $false$ (never holds)
    - **Complete**: for all constraints c either $\mathcal{CT} \models \forall c$ or $\mathcal{CT} \models \forall \neg c$ holds

Terminology constraint system

- **Atomic constraint**: atomic formula whose predicate symbol is constraint symbol
- **Constraint**: conjunction of atomic constraints
- **Solution**: substitution θ s.t. $C\theta$ holds ($\mathcal{CT} \models C\theta$)
- **Satisfiable (consistent) constraint**: solution exists, otherwise unsatisfiable (inconsistent)
- **Equivalent constraints** C_1, C_2: have the same solutions ($\mathcal{CT} \models \forall(C_1 \leftrightarrow C_2)$)

Reasoning problems

- **Satisfaction problem**: existence of a solution
  - Solved by algorithm called **decision procedure**
- **Solution problem**: Finding a solution
  - Algorithm for solution is called (constraint) **solver**
  - Solver typically also simplifies constraints.

Transition systems (*)

- Most abstract way to capture essence of computation
- Basically a binary relation over states
- Transition relation describes how one can proceed from one state to another

States and transitions

Definition (Transition system)

- Transition system T is pair $T = (\mathcal{S}, \mapsto)$
  - $\mathcal{S}$ is set of states (configurations)
  - Transition $\mapsto$ is binary relation on states, $\mapsto \subseteq \mathcal{S} \times \mathcal{S}$

- TS **deterministic**: at most one transition from every state, otherwise **nondeterministic**
- **Reachability relation** $\mapsto *$: reflexive transitive closure of $\mapsto$
- **Initial, final states**: Nonempty subsets of $\mathcal{S}$.

Derivations and computations

Definition (Derivation)

Derivation: Sequence of states $s_0 \longmapsto s_1 \longmapsto \dots$ where $s_0 \longmapsto s_1 \wedge s_1 \longmapsto s_2 \wedge \dots$.

- **Finite** (terminating) if sequence is finite.
- **Length**: number of transitions in derivation.

Computation: derivation that start with initial state s_0 and ends with final state or is infinite.

Remarks

- $\mathcal{S}$ may be finite, countably infinite, or infinite
- Initial and final states not necessarily disjoint
- If no initial states given, all states initial
- Final states must include states which have no successor
- Final states can include states which have successor
- Transition (reduction) also called derivation/computation step

Example

Example (Soccer)

$\mathcal{S} = \{(t, p, a, b) \mid 0 \leq t, a, b \leq 90, p \in \{A, B\}\}$

Initial states: $\{(0, A, 0, 0), (0, B, 0, 0)\}$

Final states: $(90, p, a, b) \in \mathcal{S}$

$$
\begin{array}{ll}
(t, A, a, b) \mapsto (t+1, A, a{+}1, b) & (t, B, a, b) \mapsto (t+1, B, a, b{+}1) \\
(t, A, a, b) \mapsto (t+1, A, a, b) & (t, B, a, b) \mapsto (t+1, B, a, b) \\
(t, A, a, b) \mapsto (t+1, B, a, b) & (t, B, a, b) \mapsto (t+1, A, a, b)
\end{array}
$$

- Models progression of goal count
  - t: counter for minutes
  - Second component models possession
  - a and b: goal counters
  - Scoring, keeping ball, or loosing ball possible

Induction

Definition (Induction Principle)

Property P defined over states is called **invariant**:

If **base case** $P(s_0)$ holds and **induction hypothesis** "$P(s_n)$ implies $P(s_{n+1})$" holds for all $s_n \mapsto s_{n+1}$ then P holds for all s in derivation

Example (Soccer Invariant)

Score in soccer game always less or equal 90:

- Let $P((t,p,a,b))$ be $t \leq 90$
- P holds for initial states
- In all other states: $0 < t \leq 90$, final states $t = 90$
- All transition increment $t < 90$ by one
  $\Rightarrow$ Induction hypothesis holds $\Rightarrow$ claim holds

Abstract syntax

Two kinds of constraints: CHR (user-defined) constraints and built-in (predefined) constraints.

- **Built-in constraints**:
  - Arbitrary logical relations (solved and simplified effectively)
  - Constraint theory for built-ins is denoted by $\mathcal{CT}$
  - Built-ins true, false, and syntactic equality $=$
  - Allow embedding and utilization of given constraint solvers
  - Allow for side-effect free host language statements
  - Considered as black boxes (correct, terminating confluent)
- **User-defined constraints**:
  - Defined by rules of a CHR program

CHR program

Definition (CHR program)

Built-in Constraint:	C, D	::=	$c(t_1, \ldots, t_n) \mid C \wedge D,\ n \geq 0$
CHR Constraint:	E, F	::=	$e(t_1, \ldots, t_n) \mid E \wedge F,\ n \geq 0$
Goal:	G, H	::=	$C \mid E \mid G \wedge H$
Simplification Rule:	SR	::=	$r\ @\ E \Leftrightarrow C \,\vert\, G$
Propagation Rule:	PR	::=	$r\ @\ E \Rightarrow C \,\vert\, G$
Simpagation Rule:	SPR	::=	$r\ @\ E_1 \backslash E_2 \Leftrightarrow C \,\vert\, G$
CHR Rule:	R	::=	$SR \mid PR \mid SPR$
CHR Program:	P	::=	$\{R_1 \ldots R_m\}, m \geq 0$

- r **name**, optional unique identifier
- E, E_1, E_2 **head**, nonempty conjunction of CHR constraints
- C optional **guard**, conjunction of built-ins
- G **body**, conjunction of built-ins and CHR constraints

Definition (II)

Definition (Additional concepts)

- **Removed constraints**: head constraints of simplification rule and head constraints E_2 of simpagation rule
- **Kept constraints**: other head constraints
- **Defined constraint**: occurs in head of rule
- **Used constraint**: occurs in body of rule
- **Local variable of rule**: does not occur in rule head
- **Range-restricted rule**: No local variables
(Program range-restricted if all rules range-restricted)

Multiset and sequence notation

- Use of first-order logic conjunction emphasizes close ties of CHR to logic
- Should be understood purely syntactically
- Conjunction interpreted as logical operator, multiset or sequence forming operator
- Operator $\uplus$ used for multiset union
- When multisets treated as sequences, order chosen at random
- List notation ($[H|T]$ or $[]$) for sequences
- Operator $+$ denotes sequence concatenation

Generalized simpagation rule notation

- Simplification, propagation and simpagation rules as special case of **Generalized simpagation rule**

$$E_1 \setminus E_2 \Leftrightarrow C \mid G$$

- E_1 kept, E_2 removed constraints, C guard, G body
- If E_1 empty rule equivalent to simplification rule $E_2 \Leftrightarrow C \mid G$
- If E_2 empty rule equivalent to propagation rule $E_1 \Rightarrow C \mid G$
- At least on of E_1 and E_2 must be nonempty

Operational semantics

- Describes how program is executed
- Defined by transitions system
  - States are conjunctions of CHR and built-in constraints
  - Transitions correspond to rule applications
- Starting from initial state rules are applied until exhaustion or contradiction
  - Simplification rule replaces CHR constraints matching its head by its body if guard holds
  - Propagation rule adds its body without removal
  - Simpagation rule removes part of the matched constraints

Very abstract semantics (*)

States

Definition (States)

- **State**: conjunction of built-in and CHR constraints
- **Initial state**: arbitrary state
- **Final state**: no transitions possible anymore

- Conjunction as multiset forming operator:
  - Conjunction is associative and commutative, but not idempotent
  - Multiplicity of conjuncts matters, permutation and grouping allowed
- Built-ins allow for computations with possibly infinitely many ground instances
- States can be understood as set comprehension
  - State $E \wedge D$ (E CHR constraints, D built-ins) stands for potentially infinite set of ground instances E, $\{E|D\}$

Transitions

Definition (Transition Apply)

$$(H_1 \land H_2 \land G) \mapsto_r (H_1 \land C \land B \land G)$$

if there is an instance of a rule in the program with new local variables $\bar{x}$

$$r \;@\; H_1 \backslash H_2 \Leftrightarrow C \mid B$$

and $\mathcal{CT} \models \forall\, (G \rightarrow \exists \bar{x} C)$

- Rule r generalised simpagation rule in **head normal form**: Arguments of the head constraints are distinct variables.
- H_1, H_2, C, B, G denote possibly empty conjunctions of constraints

Ask and Tell

Built-in constraints

- *tell*: *producer* adds/places constraint to the constraint store
- *ask*: *consumer* checks entailment (implication) of constraints from the store (but does not remove any constraint)

Example:

Operation		Constraint Store
tell	$X \leq Y$	$X \leq Y$
tell	$Y \leq Z$	$X \leq Y \land Y \leq Z$
ask	$X \leq Z$	$X \leq Y \land Y \leq Z$
ask	$Y \leq X$	$X \leq Y \land Y \leq Z$
tell	$Z \leq X$	$X = Y \land Y = Z$
ask	$Y \leq X$	$X = Y \land Y = Z$
ask	$X > Z$	$X = Y \land Y = Z$

Applicability condition

- Instance of rule (with new local variables $\bar{x}$) **applicable** if
  - Head constraints appear in the state
  - **Applicability condition (AC)** $\mathcal{CT} \models \forall\ (G \rightarrow \exists \bar{x} C)$ holds
- Actually, AC only considers built-in constraints of G

Rule application (I)

- When rule applied
  - CHR head constraints H_1 kept, H_2 removed from state
  - Guard C and body B is added (C may contain variables not contained in body or head)
- When more than one rule applicable, one is chosen nondeterministically
  - Choice cannot be undone (committed-choice)

Rule application (II)

- CHR constraints can be added and removed by rule application
- CHR constraints behave nonmonotonically in general
- Built-in constraints can only be added but not removed
- Built-ins monotonically accumulate information

Example GCD

Example (Greatest common divisor)

```
gcd1 @ \ gcd(I) ⇔ I=0 | true.
gcd2 @ gcd(I) \ gcd(J) ⇔ J>=I ∧ I>0 | gcd(J−I).
```

($true$, $=$, $\geq$, $>$: built-in constraints)

Example computation

$$
\begin{array}{rl}
 & \underline{\mathtt{gcd}(6) \wedge \mathtt{gcd}(9)} \\
\mapsto_{gcd1} & \underline{\mathtt{gcd}(6) \wedge \mathtt{gcd}(3)} \\
\mapsto_{gcd1} & \underline{\mathtt{gcd}(3) \wedge \mathtt{gcd}(3)} \\
\mapsto_{gcd1} & \underline{\mathtt{gcd}(0)} \wedge \mathtt{gcd}(3) \\
\mapsto_{gcd2} & \mathtt{gcd}(3)
\end{array}
$$

Example – Partial Order Relation

Example (Program)

```
reflexivity @ X leq Y ⇔ X=Y | true (r1)
antisymmetry @ X leq Y ∧ Y leq X ⇔ X=Y (r2)
transitivity @ X leq Y ∧ Y leq Z ⇒ X leq Z (r3)
idempotency @ X leq Y ∧ X leq Y ⇔ X leq Y (r4)
```

(*true* and $=$: built-in constraints)

Example – Partial Order Relation (2)

Example computation

$$
\begin{array}{ll}
 & \underline{\texttt{A leq B}} \wedge \underline{\texttt{C leq A}} \wedge \texttt{B leq C} \\
\mapsto_{\textbf{apply}} (r3) & \texttt{A leq B} \wedge \texttt{C leq A} \wedge \underline{\texttt{B leq C}} \wedge \underline{\texttt{C leq B}} \\
\mapsto_{\textbf{apply}} (r2) & \underline{\texttt{A leq B}} \wedge \underline{\texttt{C leq A}} \wedge \texttt{B=C} \\
\mapsto_{\textbf{apply}} (r2) & \texttt{A=B} \wedge \texttt{B=C}
\end{array}
$$

Example (Program)

$\texttt{X leq Y} \Leftrightarrow \texttt{X=Y | true}$ $(r1)$

$\texttt{X leq Y} \wedge \texttt{Y leq X} \Leftrightarrow \texttt{X=Y}$ $(r2)$

$\texttt{X leq Y} \wedge \texttt{Y leq Z} \Rightarrow \texttt{X leq Z}$ $(r3)$

$\texttt{X leq Y} \wedge \texttt{X leq Y} \Leftrightarrow \texttt{X leq Y}$ $(r4)$

Example – Min

Example (Program)

$\texttt{min(X,Y,Z)} \Leftrightarrow \texttt{X}\leq\texttt{Y} \mid \texttt{Z}=\texttt{X} \quad (r1)$

$\texttt{min(X,Y,Z)} \Leftrightarrow \texttt{Y}\leq\texttt{X} \mid \texttt{Z}=\texttt{Y} \quad (r2)$

$\texttt{min(X,Y,Z)} \Leftrightarrow \texttt{Z}<\texttt{X} \mid \texttt{Y}=\texttt{Z} \quad (r3)$

$\texttt{min(X,Y,Z)} \Leftrightarrow \texttt{Z}<\texttt{Y} \mid \texttt{X}=\texttt{Z} \quad (r4)$

$\texttt{min(X,Y,Z)} \Rightarrow \texttt{Z}\leq\texttt{X} \wedge \texttt{Z}\leq\texttt{Y} \quad (r5)$

($=$, $\leq$ and $<$ built-in constraint symbols)

Example – Min (2)

Example computation

$$
\begin{array}{ll}
 & \mathtt{min(1,2,M)} \\
\mapsto_{\mathbf{apply}}\ (r1) & \mathtt{M{=}1} \\
 & \mathtt{min(A,A,M)} \\
\mapsto_{\mathbf{apply}}\ (r1) & \mathtt{M{=}A} \ \wedge\ \mathtt{A} \leq \mathtt{A} \\
 & \mathtt{min(A,B,M)} \ \wedge\ \mathtt{A} \leq \mathtt{B} \\
\mapsto_{\mathbf{apply}}\ (r1) & \mathtt{M{=}A} \ \wedge\ \mathtt{A} \leq \mathtt{B}
\end{array}
$$

Example (Program)

$\mathtt{min(X,Y,Z)} \Leftrightarrow \mathtt{X{\leq}Y} \mid \mathtt{Z{=}X} \quad (r1)$

...

Example – Min (3)

Example computation

$$\begin{array}{ll} & \texttt{min}(\texttt{A},\texttt{2},\texttt{2}) \\ \mapsto_{\textbf{apply}}\ (r5) & \texttt{min}(\texttt{A},\texttt{2},\texttt{2}) \wedge \texttt{2} \leq \texttt{A} \wedge \texttt{2} \leq \texttt{2} \\ \mapsto_{\textbf{apply}}\ (r2) & \texttt{2}=\texttt{2} \ \wedge\ \texttt{2} \leq \texttt{A} \wedge \texttt{2} \leq \texttt{2} \\ \equiv & \texttt{2} \leq \texttt{A} \end{array}$$

Example (Program)

$\texttt{min(X,Y,Z)} \Leftrightarrow \texttt{X}\leq\texttt{Y} \mid \texttt{Z}=\texttt{X} \quad (r1)$

$\texttt{min(X,Y,Z)} \Leftrightarrow \texttt{Y}\leq\texttt{X} \mid \texttt{Z}=\texttt{Y} \quad (r2)$

...

$\texttt{min(X,Y,Z)} \Rightarrow \texttt{Z}\leq\texttt{X} \ \wedge\ \texttt{Z}\leq\texttt{Y} \quad (r5)$

Example – Min (4)

Example computation

$$
\begin{array}{ll}
 & \mathtt{min}(\mathtt{A},\mathtt{B},\mathtt{M}) \ \wedge\ \mathtt{A}{=}\mathtt{M} \\
\mapsto_{\textbf{apply}}\ (r5) & \mathtt{min}(\mathtt{A},\mathtt{B},\mathtt{M}) \wedge \mathtt{M} \leq \mathtt{A} \wedge \mathtt{M} \leq \mathtt{B} \ \wedge\ \mathtt{A}{=}\mathtt{M} \\
\mapsto_{\textbf{apply}}\ (r1) & \mathtt{A}{=}\mathtt{M} \ \wedge\ \mathtt{A} \leq \mathtt{B} \wedge \mathtt{M} \leq \mathtt{A} \wedge \mathtt{M} \leq \mathtt{B} \wedge \mathtt{A}{=}\mathtt{M} \\
\equiv & \mathtt{M} \leq \mathtt{B} \wedge \mathtt{A}{=}\mathtt{M}
\end{array}
$$

Example (Program)

$\mathtt{min(X,Y,Z)} \Leftrightarrow \mathtt{X}{\leq}\mathtt{Y} \mid \mathtt{Z}{=}\mathtt{X}\ \ (r1)$

...

$\mathtt{min(X,Y,Z)} \Rightarrow \mathtt{Z}{\leq}\mathtt{X} \ \wedge\ \mathtt{Z}{\leq}\mathtt{Y}\ \ (r5)$

Example – Min (5)

Example computation

- `min(A,2,1)` $\mapsto_{\mathbf{apply}\ (r4)} \mapsto^{*}$ `A=1`
- `min(A,2,3)` $\mapsto_{\mathbf{apply}\ (r5)} \mapsto^{*}$ *false*

Example (Program)

...

`min(X,Y,Z)` $\Leftrightarrow$ `Z<Y | X=Z` $(r4)$

`min(X,Y,Z)` $\Rightarrow$ `Z≤X` $\wedge$ `Z≤Y` $(r5)$

CHR with disjunction (*)

Nondeterminisms

- **Don't-care nondeterminism**
  - Choice should not matter for result, it is enough to know one result
  - In CHR, for choice of constraints from a state and for choice of rule to apply
- **Don't-know nondeterminism**
  - Trying out different choices
  - In CHR, usually provided by host-language of CHR library
  - E.g. disjunction of Prolog can be used in rule body
  - Disjunction formalized in $\mathrm{CHR}^{\vee}$

Syntax and states

Extension of syntax of CHR. Disjunction in goals and for states.

Definition (CHR$^\vee$ extended syntax)

Goal:	G, H	$::=$	$C \mid E \mid G \wedge H \mid G \vee H$
Configuration:	S, T	$::=$	$S \mid S \vee T$

- **Configuration** $s_1 \vee s_2 \vee \ldots \vee s_n$: Disjunction of CHR states
- Each state represents independent branch in search tree
- **Initial** configuration: initial state
- **Final** configuration: consists of final states only
- **Failed** configuration: all states have inconsistent built-ins

Transitions (I)

Two additional transitions for configurations

Definition (Split transition in CHR$^\vee$)

Split

$$((H_1 \vee H_2) \wedge G) \vee S \mapsto_\vee (H_1 \wedge G) \vee (H_2 \wedge G) \vee S$$

- Can always be applied when state contains disjunction
- Branching the derivation: splitting into disjunction of two states
- Each state will be processed independently
- Constructs tree of states rather than sequence (search tree)

Transitions (II)

Definition (Apply transition in CHR$^\vee$)

Apply

$$(H_1 \wedge H_2 \wedge G) \vee S \mapsto_r (H_1 \wedge C \wedge B \wedge G) \vee S$$

if there is an instance of a rule in the program with fresh variables $\bar{x}$,

$$r \;@\; H_1 \backslash H_2 \Leftrightarrow C \mid B$$

and $\mathcal{CT} \models \forall \, (G \rightarrow \exists \bar{x} C)$

- Applies to disjunct, i.e. state, inside configuration

Example – Maximum

Example (Maximum in CHR$^\vee$)

`max(X,Y,Z)` $\Leftrightarrow$ `(X`$\leq$`Y` $\wedge$ `Y=Z)` $\vee$ `(Y`$\leq$`X` $\wedge$ `X=Z)`

- `max` constraint in query (initial goal) will reduce to disjunct
- `max(1,2,M)`: first disjunct leads to `M=2`, second fails
- `max(1,2,3)`: both disjuncts fail $\Rightarrow$ failed configuration
- `max(1,1,M)`: both disjuncts reduce to `M=1`

Abstract semantics ω_t

- Abstract operational semantics of CHR
  - Refinement of very abstract semantics
  - Distinguishes between yet unprocessed constraints, CHR and built-in constraints
  - Avoids trivial nontermination
  - Uses matching for rule heads
- Also called standard, theoretical, or high-level operational semantics
- We adopt ω_t version of abstract operational semantics

Trivial nontermination

Very abstract semantics does not care much about termination.

- Failed states do not terminate
  - In failed state any rule is applicable
  - Failed state can only lead to failed state (monotonic accumulation of built-ins)
  - Solution: declare failed states as final states
- Propagation rules do not terminate
  - Can be applied again and again
  - Solution 1: Fair rule selection strategy (not ignoring applicable rule infinitely often)
  - Solution 2: Do not apply propagation rule twice to same constraints (need to keep a propagation history)

Rules and constraints

- Head and body of rule become multisets of atomic constraints
- Guard remains a conjunction of built-in constraints
- CHR constraints with unique identifier to distinguish multiple occurrences
  - **Numbered constraint** c_i consisting of constraint c and identifier i
  - Auxiliary notation $(c_i) = c$ and function $id(c_i) = i$ (with pointwise extension to sequences and sets of constraints)

States (I)

Definition (ω_t state)

A ω_t state is a tuple $\langle G, S, B, T \rangle_n^{\mathcal{V}}$

- **Goal** G: multiset of all constraints to be processed
- **CHR store** S: (multi)set of numbered CHR constraints that can be matched with rules
- **Built-in store** B: conjunction of built-in constraint that has been passed to the built-in solver
- **Propagation history** T: set of tuples (r, I) (r rule name, I sequence of identifiers that matched head constraints of r)
- **Counter** n: next free integer to be used as identifier
- $\mathcal{V}$ variables of initial goal (query) (the global variables of a state)

States (II)

Definition (Kinds of states)

- **Initial state**: $\langle G, \emptyset, true, \emptyset\rangle_1^{\mathcal{V}}$ (G initial goal (**query**, problem, call), $\mathcal{V}$ its variables)
- **Failed state**: $\langle G, S, B, T\rangle_n^{\mathcal{V}}$ with inconsistent built-ins ($\mathcal{CT} \models \neg\exists B$)
- **Successful state**: Consistent built-ins and empty goal store ($G = \emptyset$)
- **Final state**: Successful state with no transition possible or failed state
- (Conditional or qualified) **Answer** (solution, result): $\exists\bar{y}((S) \wedge B)$ from final state $\langle G, S, B, T\rangle_n^{\mathcal{V}}$ ($\bar{y}$ variables *not in* $\mathcal{V}$)

Transitions (I)

Definition (Solve transition)

Solve

$$\langle \{c\} \uplus G, S, B, T \rangle_n \mapsto_{solve} \langle G, S, B', T \rangle_n$$

where c is a built-in constraint and $\mathcal{CT} \models \forall((c \wedge B) \leftrightarrow B')$

- Built-in solver adds built-in from G to B
- $C \wedge B$ is simplified to B' (how far is left unspecified)

Transitions (II)

Definition (Introduce transition)

Introduce

$$\langle \{c\} \uplus G, S, B, T \rangle_n \mapsto_{introduce} \langle G, \{c_n\} \cup S, B, T \rangle_{(n+1)}$$

where c is a CHR constraint

- Adds a CHR constraint c to S and numbers it with n
- Counter n is incremented

Transitions (III)

Definition (Apply transition)

Apply

$$\langle G, H_1 \cup H_2 \cup S, B, T\rangle_n \mapsto_{apply\ r}$$
$$\langle C \uplus G, H_1 \cup S, (H_1){=}H_1' \wedge (H_2){=}H_2' \wedge N \wedge B, T \cup \{(r, id(H_1){+}id(H_2))\}\rangle_n$$

if there is a fresh variant of a rule in the program with variables $\bar{x}$,

$$r \,@\, H_1' \backslash H_2' \;\Leftrightarrow\; N \mid C$$

where $\mathcal{CT} \models \exists(B) \;\wedge\; \forall(B \rightarrow \exists\bar{x}((H_1){=}H_1' \wedge (H_2){=}H_2' \wedge N))$ and $(r, id(H_1){+}id(H_2)) \notin T$.

Operator $+$ denotes sequence concatenation.

- Choses rule r from P
  - for which CHR constraints matching its head exist in S
  - whose guard N is logically implied by B under this matching
- Applies that rule (rule fires, is executed)
  - By replacing matched removed constraints with body

Applicability condition

Definition (Applicability condition)

$$\mathcal{CT} \models \exists(B) \;\wedge\; \forall(B \rightarrow \exists\bar{x}((H_1){=}H_1' \wedge (H_2){=}H_2' \wedge N))$$

for fresh variant $r \;@\; H_1' \backslash H_2' \;\Leftrightarrow\; N \mid C$ of a rule with variables $\bar{x}$

- Ensures that B is satisfiable
- Checks whether H_1 and H_2 match H_1' and H_2' $((H_1){=}H_1' \wedge (H_2){=}H_2')$
  - $\{p_1, \ldots, p_n\} = \{q_1, \ldots, q_m\}$ shorthand for $p_1{=}q_1 \wedge \ldots \wedge p_n{=}q_n$ if $n = m$ and for false otherwise
- Checks if N together with matching is entailed by B under $\mathcal{CT}$
- Checks that propagation history does not contain identifier of CHR constraints matching head of chosen rule $((r, id(H_1){+}id(H_2)) \notin T)$

Example – Matching

Example (Head matching)

$\exists(H{=}H')$, H from state, H' from rule head

- $\exists X(p(a){=}p(X))$
- $\forall Y \exists X(p(Y){=}p(X))$

but not

- $\forall Y \exists X(p(Y){=}p(a))$

Example (Applicability condition)

- $CT \models \exists Y{=}a \wedge \forall Y(Y{=}a \rightarrow (p(Y){=}p(a)))$
- $CT \models \exists Y{=}a \wedge \forall Y(Y{=}a \rightarrow \exists X(p(Y){=}p(X)) \wedge X{=}a)$
- $CT \not\models \exists Y{=}a \wedge \forall Y, Z(Y{=}a \rightarrow (p(Z){=}p(a)))$

Rule application

- When applicable rule is applied
  - Head H_1 is kept, H_2 is removed from CHR store
  - $(H_1)=H_1' \wedge (H_2)=H_2'$ and N are added to the built-in store (N may share variables with C)
  - Body C is added to the goal store
  - Propagation history is updated by adding $(r, id(H_1)+id(H_2))$
- Propagation history entries can be garbage-collected if involved CHR constraints have been removed

Computations

Definition (Computation)

- Finite computation is **successful** if final state is successful
- Finite computation is **failed** if final state is failed
- Computation is **nonterminating** if it has no final state

Example (GCD for abstract operational semantics)

```
gcd1 @ ∅ \ {gcd(0)} ⇔ true | true.
gcd2 @ {gcd(I)} \ {gcd(J)} ⇔ J>=I | {K is J-I, gcd(K)}.
```

Example computation

$$
\begin{array}{ll}
 & \langle\{\underline{\mathtt{gcd}(6)}, \mathtt{gcd}(9)\}, \emptyset\rangle_1 \\
\mapsto_{\textit{introduce}} & \langle\{\underline{\mathtt{gcd}(9)}\}, \{\mathtt{gcd}(6)_1\}\rangle_2 \\
\mapsto_{\textit{introduce}} & \langle\emptyset, \{\underline{\mathtt{gcd}(6)_1, \mathtt{gcd}(9)_2}\}\rangle_3 \\
\mapsto_{\textit{apply}\ \mathtt{gcd2}} & \langle\{\underline{K_1\ \mathtt{is}\ 9-6}, \mathtt{gcd}(K_1)\}, \{\mathtt{gcd}(6)_1\}\rangle_3 \\
\mapsto_{\textit{solve}} & \langle\{\underline{\mathtt{gcd}(3)}\}, \{\mathtt{gcd}(6)_1\}\rangle_3 \\
\mapsto_{\textit{introduce}} & \langle\emptyset, \{\underline{\mathtt{gcd}(6)_1, \mathtt{gcd}(3)_3}\}\rangle_4 \\
\mapsto_{\textit{apply}\ \mathtt{gcd2}} & \langle\{\underline{K_2\ \mathtt{is}\ 6-3}, \mathtt{gcd}(K_2)\}, \{\mathtt{gcd}(3)_3\}\rangle_4 \\
\mapsto_{\textit{solve}} & \langle\{\underline{\mathtt{gcd}(3)}\}, \{\mathtt{gcd}(3)_3\}\rangle_4 \\
\mapsto_{\textit{introduce}} & \langle\emptyset, \{\underline{\mathtt{gcd}(3)_3, \mathtt{gcd}(3)_4}\}\rangle_5 \\
\mapsto_{\textit{apply}\ \mathtt{gcd2}} & \langle\{\underline{K_3\ \mathtt{is}\ 3-3}, \mathtt{gcd}(K_3)\}, \{\mathtt{gcd}(3)_3\}\rangle_5 \\
\mapsto_{\textit{solve}} & \langle\{\underline{\mathtt{gcd}(0)}\}, \{\mathtt{gcd}(3)_3\}\rangle_5 \\
\mapsto_{\textit{introduce}} & \langle\emptyset, \{\mathtt{gcd}(3)_3, \underline{\mathtt{gcd}(0)_5}\}\rangle_6 \\
\mapsto_{\textit{apply}\ \mathtt{gcd1}} & \langle\emptyset, \{\mathtt{gcd}(3)_3\}\rangle_6
\end{array}
$$

Refined operational semantics ω_r

Motivation

- Nondeterminism in abstract operational semantics
  - Order of processing constraints in goal
  - Order of rule applications
- Current sequential CHR implementations
  - execute constraints in goals from left to right
  - execute constraints like a procedure call
  - apply rules in textual order of program

Refined operational semantics ω_r

- Refined semantics
  - formalizes behavior of current implementations
  - is a refinement of the abstract operational semantics
  - allows for more programming idioms and for maximizing performance
  - can cause loss of logical properties and declarative concurrency

Rules and constraints

- CHR program is sequence of rules
- Head and body are sequences of atomic constraints
- **Occurrence**: number for every head constraint (top-down, left-to-right, starting with 1)
  - But removed head constraints in simpagation rule numbered before kept ones
- **Active constraint** c_i^j: numbered constraint only to match with occurrence j of (constraint symbol of) c in some rule head
- Auxiliary notation $(.)$ and function id extended to remove occurrence: $(c_i^j) = c$, $id(c_i^j) = i$

Example GCD

Example (GCD for refined operational semantics)

gcd1 @ [] \ [gcd(0):1] $\Leftrightarrow$ *true* | *true*.
gcd2 @ [gcd(I):3] \ [gcd(J):2] $\Leftrightarrow$ J>=I | [K is J−I, gcd(K)].

States

Definition (ω_r state)

A ω_r state is a tuple $\langle A, S, B, T\rangle_n^{\mathcal{V}}$

- A, S, B, T, n like in abstract semantics
- But goal A redefined into stack
  - Sequence of built-in and CHR constraints, numbered CHR constraints, and active CHR constraints
  - Numbered constraint may appear simultaneously in A and S
- Initial, final, successful, and failed states as well as computations as for abstract semantics

Transitions (I)

- Constraints in goal executed from left to right
- Atomic CHR constraints basically executed like procedure calls
- Constraint under execution is called **active**, tries all rules in textual order of program
  - Active constraint is matched against head constraint of rule with same constraint symbol
  - If matching found, guard check succeeds, and propagation history permits it then rule fires

Transitions (II)

- Rule firing like procedure call
  - Constraints in body are executed left to right
  - When they finish, execution returns to active constraint
- If active constraint still present after all rules tried or executed, it will be removed from stack, kept in CHR store
- Constraints from store will be reconsidered (woken) when new built-ins are added that affect it

Transitions (III)

- Wake-up policy is implementation of $wakeup(S, c, B)$
  - Defines which constraints from S are woken if c is added to built-in store B
  - Ground constraints are never woken
  - Only wake CHR constraints which potentially cause rule firing (those whose variables are further constraint by newly added constraint)
  - No second waking if constraint added a second time

Solve+Wake

Definition (Solve+Wake transition)

Solve+Wake

$$\langle [c|A], S, B, T\rangle_n \mapsto_{solve+wake} \langle wakeup(S, c, B)+A, S, B', T\rangle_n$$

where c is a built-in constraint and $\mathcal{CT} \models \forall((c \wedge B) \leftrightarrow B')$

- Moves built-in c into built-in store (Solve)
- Reconsiders CHR constraints according to wake-up policy by adding them on top of goal stack (Wake)
  - They will eventually become active again

Activate

Definition (Activate transition)

Activate

$$\langle [c|A], S, B, T\rangle_n \mapsto_{activate} \langle [c_n{}^1|A], \{c_n\} \cup S, B, T\rangle_{(n+1)}$$

where c is a CHR constraint

- CHR constraint becomes active for the first time and is added to CHR constraint store
- Counter n is incremented
- Corresponds to **Introduce** from abstract semantics

Reactivate

Definition (Reactivate transition)

Reactivate

$$\langle [c_i|A], S, B, T \rangle_n \mapsto_{reactivate} \langle [c_i{}^1|A], S, B, T \rangle_n$$

where c is a CHR constraint

- Numbered CHR constraint c: Woken and re-added by **Solve**$+$**Wake** and now becomes active again
- Reconsider all rules in whose heads a potential match for c occurs

Apply

Definition (Apply transition)

Apply

$$\langle [c(\bar{t})_i^j | A], H_1 \cup H_2 \cup S, B, T\rangle_n \mapsto_{apply\ r}$$
$$\langle C{+}H{+}A, H_1 \cup S, (H_1){=}H_1' \wedge (H_2){=}H_2' \wedge B, T \cup \{(r, id(H_1){+}id(H_2))\}\rangle_n$$

if there is a fresh variant of a rule in the program with variables $\bar{x}$,

$$r\ @\ H_1' \setminus H_2' \Leftrightarrow N \mid C$$

where the j^{th} occurrence of a constraint c is in the rule head $H_1' \backslash H_2'$ and where $\mathcal{CT} \models \exists(B)\ \wedge\ \forall(B \rightarrow \exists\bar{x}((H_1){=}H_1' \wedge (H_2){=}H_2' \wedge N))$ and $(r, id(H_1){+}id(H_2)) \notin T$.

Let $H{=}[c(\bar{t})_i^j]$ if the occurrence for c is in H_1' and $H{=}[]$ if the occurrence is in H_2'

- Active constraint matches against head constraint of rule with same occurrence number j
- Active constraint either kept or removed in H depending on matched occurrence in rule head

Default

Definition (Default transition)

Default

$$\langle [c_i^j|A], S, B, T\rangle_n \longmapsto_{default} \langle [c_i^{j+1}|A], S, B, T\rangle_n$$

if no other transition is possible in the current state

- No matching of active constraint against rule with occurrence j
- Proceed to next, $j+1$-th occurrence in rules of program

Drop

Definition (Drop transition)

Drop

$$\langle [c_i^{\,j}|A], S, B, T\rangle_n \mapsto_{drop} \langle A, S, B, T\rangle_n$$

where there is no occurrence j for c in P

- Removes active constraint from stack if no more occurrences
- Numbered constraint c_i stays in CHR constraint store

Example (GCD for refined operational semantics)

```
gcd1 @ [] \ [gcd(0)^1] ⇔ true | true.
gcd2 @ [gcd(I)^3] \ [gcd(J)^2] ⇔ J>=I | [K is J-I, gcd(K)].
```

Example computation

$$
\begin{array}{ll}
 & \langle[\mathtt{gcd}(6), \mathtt{gcd}(9)]\ ,\ \emptyset\rangle_1 \\
\mapsto_{activate} & \langle[\mathtt{gcd}(6)^1_1, \mathtt{gcd}(9)]\ ,\ \{\mathtt{gcd}(6)_1\}\rangle_2 \\
\mapsto_{default} & \langle[\mathtt{gcd}(6)^2_1, \mathtt{gcd}(9)]\ ,\ \{\mathtt{gcd}(6)_1\}\rangle_2 \\
\mapsto_{default} & \langle[\mathtt{gcd}(6)^3_1, \mathtt{gcd}(9)]\ ,\ \{\mathtt{gcd}(6)_1\}\rangle_2 \\
\mapsto_{default} & \langle[\mathtt{gcd}(6)^4_1, \mathtt{gcd}(9)]\ ,\ \{\mathtt{gcd}(6)_1\}\rangle_2 \\
\mapsto_{drop} & \langle[\mathtt{gcd}(9)]\ ,\ \{\mathtt{gcd}(6)_1\}\rangle_2 \\
\mapsto_{activate} & \langle[\mathtt{gcd}(9)^1_2]\ ,\ \{\mathtt{gcd}(6)_1, \mathtt{gcd}(9)_2\}\rangle_3 \\
\mapsto_{default} & \langle[\mathtt{gcd}(9)^2_2]\ ,\ \{\mathtt{gcd}(6)_1, \mathtt{gcd}(9)_2\}\rangle_3 \\
\mapsto_{apply\ \mathtt{gcd2}} & \langle[K_1\ \mathtt{is}\ 9-6, \mathtt{gcd}(K_1)]\ ,\ \{\mathtt{gcd}(6)_1\}\rangle_3 \\
\mapsto_{solve+wake} & \langle[\mathtt{gcd}(3)]\ ,\ \{\mathtt{gcd}(6)_1\}\rangle_3 \\
\mapsto_{activate} & \langle[\mathtt{gcd}(3)^1_3]\ ,\ \{\mathtt{gcd}(6)_1, \mathtt{gcd}(3)_3\}\rangle_4 \\
\mapsto_{default} & \langle[\mathtt{gcd}(3)^2_3]\ ,\ \{\mathtt{gcd}(6)_1, \mathtt{gcd}(3)_3\}\rangle_4
\end{array}
$$

Example GCD (II)

Example computation (continued)

$$
\begin{array}{ll}
\mapsto_{\mathit{default}} & \langle[\mathtt{gcd}(3)^3_3]\ ,\ \{\mathtt{gcd}(6)_1, \mathtt{gcd}(3)_3\}\rangle_4 \\
\mapsto_{\mathit{apply}\ \mathtt{gcd2}} & \langle[K_2\ \mathtt{is}\ 6-3, \mathtt{gcd}(K_2), \mathtt{gcd}(3)^3_3]\ ,\ \{\mathtt{gcd}(3)_3\}\rangle_4 \\
\mapsto_{\mathit{solve+wake}} & \langle[\mathtt{gcd}(3), \mathtt{gcd}(3)^3_3]\ ,\ \{\mathtt{gcd}(3)_3\}\rangle_4 \\
\mapsto_{\mathit{activate}} & \langle[\mathtt{gcd}(3)^1_4, \mathtt{gcd}(3)^3_3]\ ,\ \{\mathtt{gcd}(3)_3, \mathtt{gcd}(3)_4\}\rangle_5 \\
\mapsto_{\mathit{default}} & \langle[\mathtt{gcd}(3)^2_4, \mathtt{gcd}(3)^3_3]\ ,\ \{\mathtt{gcd}(3)_3, \mathtt{gcd}(3)_4\}\rangle_5 \\
\mapsto_{\mathit{apply}\ \mathtt{gcd2}} & \langle[K_3\ \mathtt{is}\ 3-3, \mathtt{gcd}(K_3), \mathtt{gcd}(3)^3_3]\ ,\ \{\mathtt{gcd}(3)_3\}\rangle_5 \\
\mapsto_{\mathit{solve+wake}} & \langle[\mathtt{gcd}(0), \mathtt{gcd}(3)^3_3]\ ,\ \{\mathtt{gcd}(3)_3\}\rangle_5 \\
\mapsto_{\mathit{activate}} & \langle[\mathtt{gcd}(0)^1_0, \mathtt{gcd}(3)^3_3]\ ,\ \{\mathtt{gcd}(3)_3, \mathtt{gcd}(0)_5\}\rangle_6 \\
\mapsto_{\mathit{apply}\ \mathtt{gcd1}} & \langle[\mathtt{gcd}(3)^3_3]\ ,\ \{\mathtt{gcd}(3)_3\}\rangle_6 \\
\mapsto_{\mathit{default}} & \langle[\mathtt{gcd}(3)^4_3]\ ,\ \{\mathtt{gcd}(3)_3\}\rangle_6 \\
\mapsto_{\mathit{drop}} & \langle[]\ ,\ \{\mathtt{gcd}(3)_3\}\rangle_6
\end{array}
$$

Relating abstract and refined semantics (I)

- ω_r is an instance of ω_t
- Abstraction that maps states and derivations of ω_r to ω_t

Definition (Abstraction function)

For states:

$$\alpha(\langle A, S, B, T\rangle_n^{\mathcal{V}}) = \langle G, S, B, T\rangle_n^{\mathcal{V}},$$

where G contains all atomic constraints of A expect active and numbered CHR constraints.

For derivations:

$$\alpha(s_1 \mapsto s_2 \mapsto \ldots) = \begin{cases} \alpha(s_1) \mapsto \alpha(\ldots) & \text{if } \alpha(s_1){=}\alpha(s_2) \\ \alpha(s_1) \mapsto \alpha(s_2) \mapsto \alpha(\ldots) & \text{otherwise} \end{cases}$$

Relating abstract and refined semantics (II)

Theorem

For all ω_r derivations D, $\alpha(D)$ is a ω_t derivation.
If D is a terminating computation, then $\alpha(D)$ is a terminating computation.

Termination, confluence under abstract semantics preserved in refined semantics (but not the other way round)

Nondeterminism

Refined semantics is still nondeterministic

- In **Solve+Wake** transition, order of constraints added by wake-up-policy function not defined
- Matching order in **Apply** transition: not known which partner constraint from store is chosen

Declarative semantics

- Declarative semantics associates program with logical theory
- This logical reading should coincide with intended meaning of program
- Declarative semantics facilitates nontrivial program analysis (e.g. correctness for program transformation and composition)
- Logical reading of CHR program consists of logical reading of its rules and built-ins

First-order logic declarative semantics

Logical reading of rules

- Rule logically relates head and body provided the guard is true
- Simplification rule means head is true iff body is true
- Propagation rule means body is true if head is true

Definition (Logical reading)

Simplification rule:	$H \Leftrightarrow C \mid B$	$\forall\,(C \rightarrow (H \leftrightarrow \exists \bar{y}\; B))$
Propagation rule:	$H \Rightarrow C \mid B$	$\forall\,(C \rightarrow (H \rightarrow \exists \bar{y}\; B))$
Simpagation rule:	$H_1 \backslash H_2 \Leftrightarrow C \mid B$	$\forall\,(C \rightarrow ((H_1 \land H_2) \leftrightarrow (H_1 \land \exists \bar{y}\; B)))$

($\bar{y}$ contains all variables only appearing in B)

Example

Example (Partial order relation program)

```
duplicate     @ X leq Y \ X leq Y <=> true.
reflexivity   @ X leq X <=> true.
antisymmetry @ X leq Y , Y leq X <=> X=Y.
transitivity @ X leq Y , Y leq Z ==> X leq Z.
```

Example (Logical reading of partial order program)

(duplicate)	$\forall$ X,Y	$(X \leq Y \land X \leq Y \Leftrightarrow X \leq Y)$
(reflexivity)	$\forall$ X	$(X \leq X \Leftrightarrow \mathit{true})$
(antisymmetry)	$\forall$ X,Y	$(X \leq Y \land Y \leq X \Leftrightarrow X = Y)$
(transitivity)	$\forall$ X,Y,Z	$(X \leq Y \land Y \leq Z \Rightarrow X \leq Z)$

Logical reading and equivalence of programs

Meaning of built-ins has to be considered, too

Definition (Logical reading)

Logical reading of program P is $\mathcal{P}, \mathcal{CT}$
($\mathcal{P}$ conjunction of logical reading of rules in P, $\mathcal{CT}$ constraint theory defining built-ins)

Definition (Logical equivalence)

Programs P_1 and P_2 logically equivalent iff

$$\mathcal{CT} \models \mathcal{P}_1 \leftrightarrow \mathcal{P}_2$$

Logical correctness

Specification can be used to formally verify correctness of program

Definition (Logical correctness)

Logical specification $\mathcal{T}$ of program P is a consistent theory for the CHR constraints in P.

P is logically correct with respect to $\mathcal{T}$ iff

$$\mathcal{T}, \mathcal{CT} \models \mathcal{P}$$

$\mathcal{P}$ does not need to cover all consequences of $\mathcal{T}$

Logical reading of states

Definition (Logical reading of states)

Logical reading of ω_t or ω_r state is the formula

$$\exists \bar{y}\ (G \wedge (S) \wedge B)$$

($\bar{y}$ local variables of the state, those not in $\mathcal{V}$)

- Empty sequences, sets or multisets are interpreted as *true*
- Variables in $\mathcal{V}$ are not quantified
- Local variables in states come from variables of applied rules

Equivalence of states

- Declarative Semantics: Logical equivalence of states if their logical reading is equivalent
- Operational Semantics: Operational equivalence of states if the same rules can be applied to them

Operational equivalence is stricter than logical equivalence

- Take multiset character of CHR constraints into account
- Take propagation history into account

Operational equivalence of states

Definition (Operational state equivalence)

Given two states s_i $(i{=}1,2)$, with

- B_i built-in constraints of state s_i
- In very abstract semantics, C_i are CHR constraints of state
- In ω_t, ω_r operational semantics, C_i is pair of
  - CHR constraints of state with proper renaming of identifiers
  - set of tuple entries in propagation history that only contain (renamed) identifiers from the CHR constraints of the state
- Local variables $\bar{y}_i$ of state renamed apart

$s_1 \equiv s_2$ iff $\mathcal{CT} \models \forall(B_1 \rightarrow \exists\bar{y}_2(C_1{=}C_2)\wedge B_2) \wedge \forall(B_2 \rightarrow \exists\bar{y}_1(C_1{=}C_2)\wedge B_1)$

Note analogy to rule applicability conditition of operational semantics

Examples – operational equivalence of states

Example (Operational equivalence of states)

- The two states with logical reading $q(X) \wedge X = a$ and $\exists Y\ q(a) \wedge X = Y \wedge Y = a$ are equivalent
- The state $q(a)$ is not equivalent to those states
- If X is not a global variable then $\exists X\ q(X) \wedge X = a$, $\exists X, Y\ q(a) \wedge X = Y \wedge Y = a$ and $q(a)$ are equivalent
- The state $q(a) \wedge q(a)$ is not equivalent to these states

Soundness and completeness (I)

Operational and declarative semantics should coincide

- **Soundness**: Result of computation according to operational semantics is correct regarding to declarative semantics
- **Completeness**: Everything proven by declarative semantics can be computed
  - But: logic of declarative semantics too powerful
  - Additional conditions necessary to improve completeness
- Theorems show that for CHR, semantics are strongly related
- Because all states in a derivation are equivalent

Soundness and completeness (II)

Lemma (Equivalence of States in Derivation)

If C logical reading of state appearing in derivation of G then

$$\mathcal{P}, \mathcal{CT} \models \forall\, (C \leftrightarrow G)$$

For logical reading C_1, C_2 of two states in computation of G

$$\mathcal{P}, \mathcal{CT} \models \forall\, (C_1 \leftrightarrow C_2)$$

Soundness and completeness (III)

Theorem (Soundness)

If G has a computation with answer C then

$$\mathcal{P}, \mathcal{CT} \models \forall\,(C \leftrightarrow G)$$

Theorem (Completeness)

G a goal with at least one finite computation, C a goal.
If $\mathcal{P}, \mathcal{CT} \models \forall\,(C \leftrightarrow G)$ then G has finite computation with answer C' such that

$$\mathcal{P}, \mathcal{CT} \models \forall\,(C \leftrightarrow C')$$

Soundness and completeness (IV)

Completeness theorem does not hold if G has no finite computations

Example

Let P be $\mathrm{p} \;\Leftrightarrow\; \mathrm{p}$ and G be p

It holds that $\mathcal{P}, \mathcal{CT} \models \mathrm{p} \leftrightarrow \mathrm{p}$ since $\mathcal{P}$ is $\{\mathrm{p} \leftrightarrow \mathrm{p}\}$

but G has only infinite computations

Failed computations

Try to specialize theorems for failed computations

Theorem (Soundness of failed computations)

If G has a failed computation then

$$\mathcal{P}, \mathcal{CT} \models \neg\exists\, G$$

No analogous completeness result for failed computations

Example

$\mathtt{p} \;\Leftrightarrow\; \mathtt{q}.$

$\mathtt{p} \;\Leftrightarrow\; \mathit{false}\,.$

$\mathcal{P}, \mathcal{CT} \models \neg\mathtt{q}$ holds, but $\mathtt{q}$ has no failed computation

Soundness and completeness (VI)

Discrepancy between operational and declarative semantics comes from additional reasoning power of first-order logic

Example

```
a <=> b          a <=> c
```

- From $\mathcal{P}, \mathcal{CT}$ follows for example $a \leftrightarrow b$, $a \leftrightarrow c$, but also $b \leftrightarrow a$, $b \leftrightarrow c$, $a \leftrightarrow b \wedge c$
- In fact, logical equivalence between any nonempty conjunctions of a, b, c holds
- Only possible computations are $a \mapsto b$, $a \mapsto c$, as well as $b \mapsto^0 b$, and $c \mapsto^0 c$

Rules are directional, logical equivalence is not.

Soundness and completeness (VII)

Stronger completeness result for programs with consistent logical reading and data-sufficient goals

Definition (Data-sufficiency)

Goal is data-sufficient if it has a computation ending in a final state without CHR constraints.

Theorem (Stronger completeness of failed computations)

P with consistent logical reading, G data-sufficient.
If $\mathcal{P}, \mathcal{CT} \models \neg\exists G$ then G has a failed computation.

Even stronger results for *confluent* programs

Linear logic declarative semantics

- Classical logic declarative semantics not always sufficient if CHR used as general purpose language
  - Simplification rules remove and add CHR constraints (nonmonotonic), can model dynamic updates
  - But first-order logic cannot directly express change
- Alternative declarative semantics
  - Based on linear logic
  - Models resource consumption
  - Stronger theorems for soundness and completeness

Syntax (I)

Definition (Syntax of intuitionistic linear logic)

$$L ::= p(\bar{t}) \mid L \multimap L \mid L \otimes L \mid L \& L \mid L \oplus L \mid !L \mid \exists x.L \mid \forall x.L \mid \top \mid 1 \mid 0$$

- Atoms represent resources, may be consumed during reasoning

Syntax (II)

- **Linear implication** $\multimap$ (“lollipop”) different from classical logic
  - $A \multimap B$ (“consuming A yielding B”) means A can be replaced by B
  - A and $A \multimap B$ yields B (implication also consumed)
- **Conjunction** $\otimes$ (“times”) similar to classical logic
  - $A \otimes B$ available iff A and B available
  - $A \otimes A$ not equivalent to A
  - Neutral element 1, corresponds to true

Syntax (III)

- **Modality** ! ("bang") marks stable facts and resources that are not consumed
- **Conjunction** $\&$ ("with") represents internal choice (don't-care)
  - $A \& B$ ("either A or B) implies A or B but not $A \otimes B$
  - Neutral element $\top$ ("top")
- **Disjunction** $\oplus$ expresses external choice (don't-know, similar to classical disjunction)
  - $A \oplus B$ neither implies A nor B alone
  - Neutral element 0, expresses failure

Linear logic declarative semantics (I)

First-order logic (FOL) vs. linear logic semantics

- CHR constraints as linear resources
- Built-ins still in FOL as embedded intuitionistic formulas
- CHR rules as linear implication instead of logical equivalence

Linear logic declarative semantics (II)

Definition (Semantics P^L of CHR$^\vee$ program part 1)

Built-in Constraints:	$true^L$	$::=$	1
	$false^L$	$::=$	0
	$c(\bar{t})^L$	$::=$	$!c(\bar{t})$
CHR Constraints:	$e(\bar{t})^L$	$::=$	$e(\bar{t})$
Goals:	$(G \wedge H)^L$	$::=$	$G^L \otimes H^L$
	$(G \vee H)^L$	$::=$	$G^L \oplus H^L$
Configuration:	$(S \vee T)^L$	$::=$	$S^L \oplus T^L$

- Constraints mapped to $\otimes$ conjunctions of their atomic constraints
- Atomic built-ins banged (treated as unlimited resources)
- $\mathcal{CT}$ translated according to the Girard Translation
- Disjunctions mapped to $\oplus$ disjunctions

Linear logic declarative semantics (III)

Definition (Semantics P^L of CHR$^\vee$ program part 2)

Simpagation Rule:	$(E \backslash F \Leftrightarrow C \vert G)^L$	::=	$!(\forall\,(C^L \multimap (E^L \otimes F^L \multimap E^L \otimes \exists \bar{y} G^L)))$
CHR Program:	$\{R_1 \ldots R_m\}^L$	::=	$R_1^L \otimes \ldots \otimes R_m^L$

- Rules mapped to linear implications
  - Consuming part of head produces body
  - Directional, not commutative (cannot be reversed)
- Formula for rule banged (to be used more than once)
- Program translated into $\otimes$ conjunction of translated rules

Example (I)

Example (Coin throw)

- Coin throw simulator program

```
throw(Coin) ⇔ Coin = head
throw(Coin) ⇔ Coin = tail
```

- Classical declarative FOL semantics

$$(throw(Coin) \leftrightarrow (Coin{=}head)) \wedge (throw(Coin) \leftrightarrow (Coin{=}tail))$$

- Leads to $(Coin{=}head) \leftrightarrow (Coin{=}tail)$ and therefore $head{=}tail$

Example (II)

Example (Coin throw continued)

```
throw(Coin) ⇔ Coin = head
throw(Coin) ⇔ Coin = tail
```

Linear logic reading

$$!\forall(\mathit{throw}(\mathit{Coin}) \multimap !(\mathit{Coin}{=}\mathit{head})) \otimes !\forall(\mathit{throw}(\mathit{Coin}) \multimap !(\mathit{Coin}{=}\mathit{tail}))$$

This is logically equivalent to:

$$!\forall\,(\mathit{throw}(\mathit{Coin}) \multimap !(\mathit{Coin}{=}\mathit{head}) \,\&\, !(\mathit{Coin}{=}\mathit{tail}))$$

Reads as "Of course, consuming $\mathit{throw}(\mathit{Coin})$ produces: Choose from $\mathit{Coin} = \mathit{head}$ and $\mathit{Coin} = \mathit{tail}$" (committed choice)

Another example (I)

Example (Destructive assignment)

`assign(Var,New) ∧ cell(Var,Old) ⇔ cell(Var,New)`

FOL reading:

$$\forall(assign(Var,New) \wedge cell(Var,Old) \Leftrightarrow cell(Var,New))$$

which is logically equivalent to

$$\forall(assign(Var,New) \wedge cell(Var,Old) \Leftrightarrow cell(Var,Old) \wedge cell(Var,New))$$

Means that `Var` holds old and new value simultaneously

Another example (II)

Example (Destructive assignment continued)

`assign(Var,New)` ∧ `cell(Var,Old)` ⇔ `cell(Var,New)`

Linear logic reading

$$!\forall(assign(Var,New) \otimes cell(Var,Old) \multimap cell(Var,New))$$

Reads as "Of course, consuming `assign(Var,New)` and `cell(Var,Old)` produces `cell(Var,New)`."

Yet another example

Example (Prime sieve)

`prime(I) ∧ prime(J) ⇔ J mod I = 0 | prime(I)`

$$\text{FOL:} \quad \forall((M \text{ mod } N = 0) \rightarrow (prime(M) \wedge prime(N) \leftrightarrow prime(N)))$$

"A number is prime when it is multiple of another prime".

$$\text{LL:} \quad !\forall(!(M \text{ mod } N = 0) \multimap (prime(M) \otimes prime(N) \multimap prime(N)))$$

"Of course, consuming `prime(M)` and `prime(N)` where `(M mod N = 0)` produces `prime(N)`"

And even more examples

Example (Birds and penguins)

`bird` $\Leftrightarrow$ `albatross` $\vee$ `penguin.`

`penguin` $\wedge$ `flies` $\Leftrightarrow$ *false*.

$$\text{FOL}: \quad (bird \leftrightarrow albatross \vee penguin) \wedge (penguin \wedge flies \leftrightarrow false)$$

This is correct, but more than can be computed, e.g. $albatros \rightarrow bird$.

$$\text{LL}: \quad !(bird \multimap albatross \oplus penguin) \otimes \; !(penguin \otimes flies \multimap 0)$$

implies only computable implications

$$bird \otimes flies \multimap albatross \otimes flies$$

"`bird` and `flies` can be mapped to `albatross` and `flies`"

Soundness and completeness (I)

- Approach for soundness analogous to classical framework
- In the following:
  - P a CHR$^\vee$ program
  - P^L its logical reading and $!\mathcal{CT}^L$ constraint theory for built-ins
  - S_0 initial configuration, S_m, S_n configurations
  - $\vdash$ denotes deducability

Any configuration in derivation is linearly implied by logical reading of initial configuration

Lemma (Linear implication of states)

If S_n appears in derivation of S_0 then

$$P^L, !\mathcal{CT}^L \vdash \forall(S_0^L \multimap S_n^L)$$

Soundness and completeness (II)

Theorem (Soundness)

If S_0 has computation with final configuration S_n^L then

$$P^L, !\mathcal{CT}^L \vdash \forall\,(S_0^L \multimap S_n^L)$$

Theorem (Completeness)

If

$$P^L, !\mathcal{CT}^L \vdash \forall\,(S_0^L \multimap S_n^L)$$

then there is S_m in a finite prefix of derivation of S_0 with

$$!\mathcal{CT}^L \vdash S_m^L \multimap S_n^L$$