Modular CHR with *ask* and *tell*

François Fages, Cleyton Mario de Oliveira Rodrigues, Thierry Martinez
Contraintes Project-Team, INRIA Paris-Rocquencourt, France

1. Why CHRat?
2. A Simple Example.
4. Translation of CHRat to flat CHR.
5. Examples of Modular CHRat Solvers.
Programming in CHR.

CHR is a language to define constraint-solvers by multiset rewriting rules which are guarded by built-in constraints.


Example of constraint solver definition.

Let leq(X,Y) token represent the constraint $X \leq Y$.

1. leq(X,X) $\iff$ true.
2. leq(X,Y), leq(Y,X) $\iff$ X = Y. \leftarrow simplifications
3. leq(X,Y), leq(Y,Z) $\Rightarrow$ leq(X,Z). \leftarrow propagation
4. leq(X,Y) \setminus leq(X,Y) $\iff$ true. \leftarrow simpagation

Solved forms are irreflexive and transitively closed.
Programming in CHR is non-modular.

Non-reusability of CHR Constraint-Solvers in Guards

Once a new CHR constraint-solver is defined, the resulting solver cannot become the built-in constraint solver of another CHR program.

Satisfaction and Entailment

- CHR constraint-solvers define satisfiability checkers.
- Guards have to be entailed to fire the associated rule.
Towards a Modular CHR Language

Entailment Checking

Three approaches:

1. **External implementation**

2. **Automatic entailment checking**
   
   \[ C \rightarrow D \iff C \land D \leftrightarrow C \]


3. **Our approach: a discipline for programming entailment checking in CHR with ask and tell.**
min Solver over leq Solver in CHR?

Let $\min(X,Y,Z)$ represent the constraint that $Z$ is the minimum value among $X$ and $Y$.

\[
\text{leq}(X,Y) \setminus \min(X,Y,Z) \iff Z=X.
\]

\[
\text{leq}(Y,X) \setminus \min(X,Y,Z) \iff Z=Y.
\]

\[
\min(X,Y,Z) \implies \text{leq}(Z,X), \text{leq}(Z,Y).
\]

Does not work: $\min(X,X,Z)$ will not be rewritten to $X=Z$ because there is no $\text{leq}(X,X)$ token in the store.
leq Solver Component in CHRat.

File leq_solver.cat

component leq_solver.

export leq /2.

leq(X,X) ⇔ true.

leq(X,Y), leq(Y,X) ⇔ X = Y.

leq(X,Y), leq(Y,Z) ⇒ leq(X,Z).

leq(X,Y) \ leq(X,Y) ⇔ true.

ask(leq(X,X)) ⇔ entailed(leq(X,X)).

leq(X,Y) \ ask(leq(X,Y)) ⇔ entailed(leq(X,Y)).
min Solver Component in CHRat.

File min_solver.cat

component min_solver.
import leq/2 from leq_solver.
export min/3.
min(X,Y,Z) ⇐⇒ leq(X,Y) | Z=X.
min(X,Y,Z) ⇐⇒ leq(Y,X) | Z=Y.
min(X,Y,Z) ⇒ leq(Z,X), leq(Z,Y).

ask(min(X, Y, X)) ⇐⇒ leq(X, Y) | entailed(min(X, Y, X)).
ask(min(X, Y, Y)) ⇐⇒ leq(Y, X) | entailed(min(X, Y, Y)).

min(X,Y,Z)\ask(min(X,Y,Z)) ⇐⇒ entailed(min(X,Y,Z)).
CHRat Syntax.

**component** `<component-name>`: one per file.

**import** `<constraint-declarations>` from `<component-name>`.

Separation is atom-prefix based.

**export** `<constraint-declarations>`.

\[
<\text{rule-name}> @ <\mathcal{H}> \setminus <\mathcal{H}> ⇔ <\mathcal{C}>, <\mathcal{T}> \mid <\mathcal{B}>.
\]

Where:

- \( \mathcal{C} \): built-in constraints
- \( \mathcal{T} \): CHR constraints
- \( \mathcal{H} = \mathcal{T} \cup \text{ask}(\mathcal{T}) \)
- \( \mathcal{B} = \mathcal{C} \cup \mathcal{T} \cup \text{entailed}(\mathcal{T}) \)

**Side condition** Every variable which appears in a CHR guard must appear in the built-in guard or in the heads of the rule.
Configurations \( \langle F, E, D \rangle \& \mathcal{V} \)

where \( \mathcal{V} \) is the set of free variables of the initial query.

Logical meaning \( \exists \vec{y}(\bar{F} \land \bar{E} \land D), \text{ where } \vec{y} \text{ enumerates } \text{fv}(F, E, D) \setminus \mathcal{V}. \)
Solve

\[
\begin{align*}
\forall c \in C \quad \langle \{c\} \cup F, E, D \rangle_V & \rightarrow \langle F, E, c \wedge D \rangle_V
\end{align*}
\]

Introduce

\[
\begin{align*}
\forall t \in \mathcal{T}^* \quad \langle \{t\} \cup F, E, D \rangle_V & \rightarrow \langle F, \{t\} \cup E, D \rangle_V
\end{align*}
\]

where \(\mathcal{T}^* = \mathcal{T} \cup \text{ask}(\mathcal{T}) \cup \text{entailed}(\mathcal{T})\).

Trivial Entailment

\[
\begin{align*}
\forall t \in \mathcal{T} \quad \langle F, \{\text{ask}(t), t\} \cup E, D \rangle_V & \rightarrow \langle \{\text{entailed}(t)\} \cup F, \{t\} \cup E, D \rangle_V
\end{align*}
\]
Ask
\[
\begin{align*}
(H \setminus H' & \iff C_b, C_c \mid B.) \sigma \in P \quad D \vdash_C C_b \\
\langle F, H \cup H' \cup E, D \rangle \nu & \mapsto \langle \text{ask}(C_c) \cup F, H \cup H' \cup E, D \rangle \nu
\end{align*}
\]

Fire
\[
\begin{align*}
(H \setminus H' & \iff C_b, C_c \mid B.) \sigma \in P \quad D \vdash_C C_b \\
\langle F, H \cup H' \cup \text{entailed}(C_c) \cup E, D \rangle \nu & \mapsto \langle B \cup F, H \cup E, D \rangle \nu
\end{align*}
\]
CHRat Declarative Semantics for Rules.

\[(H \ \big| \ H') \iff C_b, C_c \ | \ B.)^{\ddagger} \doteq \]
\[\forall \bar{y}(C_b \rightarrow \overline{H} \land \overline{H'} \rightarrow \overline{\text{ask}(C_c)})\]
\[\land \forall \bar{y}(C_b \rightarrow (\overline{H} \land \overline{H'} \land \overline{\text{entailed}(C_c)} \iff \exists \bar{y}'(\overline{H} \land \overline{B})))\]

**Theorem**

*Operational semantics is sound and complete with respect to declarative semantics.*

If \(D\) is the declarative semantics of a program \(P\) and \(S_1 \leftrightarrow S_2\) two successive configurations in an execution of \(P\), then:

\[D \models_c S_1 \leftrightarrow S_2\]

Adapted from the soundness and completeness theorem of CHR:

Translation to flat CHR.

\[
[H \setminus H' \Leftrightarrow C_b, C_c \mid B.]
\]

\[
= \begin{cases} 
H, H' \Rightarrow C_b \mid \text{ask}(C_c) \\
H \setminus H', \text{entailed}(C_c) \Leftrightarrow C_b \mid B.
\end{cases}
\]

**Theorem**

If:

- \( \mathcal{D} \) is the CHRat declarative semantics of a CHRat program \( P \); and
- \( \mathcal{D}' \) is the CHR declarative semantics of \( \llbracket P \rrbracket \).

then:

\[
\vdash_c \mathcal{D} \leftrightarrow \mathcal{D}'
\]
Example of Translation to flat CHR.

\[
\min(X, Y, Z) \Rightarrow \text{entailed } \min(X, Y, Z).
\]

\[
\min(X, Y, Z) \iff \leq(X, Y) | Z = X.
\]

\[
\min(X, Y, Z) \Rightarrow \text{entailed } \leq(X, Y).
\]

\[
\text{entailed } \leq(X, Y), \min(X, Y, Z) \iff Z = X.
\]

\[
\min(X, Y, Z) \Rightarrow \leq(Z, X), \leq(Z, Y).
\]

\[
\text{ask}(\min(X, Y, X)) \iff \leq(X, Y) | \text{entailed } (\min(X, Y, X)).
\]

\[
\text{ask} \min(X, Y, X) \Rightarrow \text{ask} \leq(X, Y).
\]

\[
\text{entailed } \leq(X, Y), \text{ask} \min(X, Y, X) \iff \text{entailed } \min(X, Y, X).
\]
Union-find Component. (1/3)


File union_find_solver.cat

component union_find.

export make/1, ≃ /2.

make(A) ⟷ root(A, 0).

union(A, B) ⟷ find(A, X), find(B, Y), link(X, Y).

A ≃ B, find(A, X) ⟷ find(B, X), A ≃ X.

root(A, _) \ find(A, X) ⟷ X = A.

link(A, A) ⟷ true.

link(A, B), root(A, N), root(B, M) ⟷ N ≥ M | B ≃ A, N1 is max(M+1, N), root(A, N1).

link(B, A), root(A, N), root(B, M) ⟷ N ≥ M | B ≃ A, N1 is max(M+1, N), root(A, N1).
Union-find Component. (2/3)

File `union_find_solver.cat`

A $\simeq$ B $\implies$ `union(A, B)`.

\( \text{ask}(A \simeq B) \iff \) find(A, X), find(B, Y), check(A, B, X, Y).

\( \text{root}(X) \setminus \text{check}(A, B, X, X) \iff \) entailed(A $\simeq$ B).

![Diagram](attachment://diagram.png)
Union-find Component. (3/3)

\[
\begin{align*}
X \leadsto C \setminus & \text{check}(A, B, X, Y) \iff \\
& \text{find}(A, Z), \text{check}(A, B, Z, Y).
\end{align*}
\]

\[
\begin{align*}
Y \leadsto C \setminus & \text{check}(A, B, X, Y) \iff \\
& \text{find}(B, Z), \text{check}(A, B, X, Z).
\end{align*}
\]
Rational Tree Solver Component.

File rational_tree_solver.cat

component rational_tree_solver.
import \sim /2 from union_find_solver.
export fun/3, arg/3, \sim /2.
fun(X0, F0, N0) \fun(X1, F1, N1) \iff X0 \sim X1 | F0 = F1, N0 = N1.
arg(X0, N, Y0) \arg(X1, N, Y1) \iff X0 \sim X1 | Y0 \sim Y1.

X \sim Y \iff X \sim Y.

Check the paper for the ask-solver!
Conclusion.

Objective

- Generalization of guards.
- Modular definition of solvers.

Proposed Solution.

- Programming discipline to define satisfiability and entailment constraint solvers.

Perspectives.

- Relax the restriction on guard variables.
- Link between declarative semantics of ask and logical implication.
- Modular compilation.