Generalized CHR machines

Jon Sneyers and Thom Frühwirth
K.U.Leuven, Belgium          Universität Ulm, Germany

CHR’08 Workshop
Hagenberg, Austria, July 2008

驰
1 Introduction
   - Complexity-wise completeness
   - CHR machines

2 Strategy classes

3 Non-deterministic CHR machines

4 Self-modifying CHR machines

5 Conclusion
Complexity-wise completeness result for CHR (see [CHR’05] and [TOPLAS])

“Every algorithm can be implemented in CHR with the optimal time and space complexity”

CHR machine: one \( \omega_t \) step = one machine step

In [CHR’05] and [TOPLAS]: compare CHR machine with RAM machine
(Deterministic abstract) CHR machine

Definition

A deterministic abstract CHR machine is a tuple $\mathcal{M} = (\mathcal{H}, \mathcal{P}, \mathcal{VG})$. The host language $\mathcal{H}$ defines a built-in constraint theory $\mathcal{D}_\mathcal{H}$, $\mathcal{P}$ is a CHR program, and $\mathcal{VG} \subseteq \mathcal{G}^\mathcal{H}_\mathcal{P}$ is a set of valid goals, such that $\mathcal{P}$ is a $\Delta^\mathcal{H}_{\omega_t}$-deterministic CHR program for input $\mathcal{VG}$. The machine takes an input query $\mathcal{G} \in \mathcal{VG}$ and executes a derivation $d \in \Delta^\mathcal{H}_{\omega_t}|\mathcal{G}$. 
1 Introduction

2 Strategy classes
   - Execution strategies
   - Strategy classes
   - Generalized confluence
   - Generalized CHR machines

3 Non-deterministic CHR machines

4 Self-modifying CHR machines

5 Conclusion
An execution strategy is a set of derivations such that there is exactly one derivation for every initial state.

A computable execution strategy is an execution strategy that can be implemented (i.e., for which an underlying state transition system exists whose transition function is computable).
set of all execution strategies for a program $\mathcal{P}$: $\Omega^H_t(\mathcal{P})$

- a strategy class $\Omega(\mathcal{P}) \subseteq \Omega^H_t(\mathcal{P})$ is a set of execution strategies

- a computable strategy class is a strategy class which contains at least one computable execution strategy.

- clearly, the strategy class corresponding to the K.U.Leuven CHR system is computable

- so the refined semantics strategy class $\Omega^H_r$ is computable

- so the abstract semantics strategy class $\Omega^H_t$ is computable
Definition (Ω-confluence)

A CHR program \( P \) is \( \Omega(P) \)-confluent if, for every initial state \( \langle G, \emptyset, \text{true}, \emptyset \rangle_1 = \sigma \in \Sigma^{\text{init}} \) and execution strategies \( \xi_1, \xi_2 \in \Omega(P) \), the following holds:

\[
\sigma \xrightarrow{\xi_1} \langle G_1, S_1, B_1, T_1 \rangle_{n_1} \quad \land \quad \sigma \xrightarrow{\xi_2} \langle G_2, S_2, B_2, T_2 \rangle_{n_2}
\]

\[
\Rightarrow D_H \models \exists G (S_1 \land B_1) \leftrightarrow \exists G (S_2 \land B_2)
\]

Note that a CHR program \( P \) is \( \Omega^t_H(P) \)-confluent if and only if it is confluent (in the usual sense).
Overview

Execution strategies

\[ \Omega^H_t(\mathcal{P}) \]

- K.U. Leuven
- SICStus
- toyCHR
- JCHR

\[ \Omega^H_r(\mathcal{P}) \]

- dPCHR

\[ \Omega^H_p(\mathcal{P}, \bar{p}) \]

CHR programs

\{K.U. Leuven\}-confluent programs

\[ \Omega^H_r \text{-confluent} \]

\[ \Omega^H_p \text{-confl.} \]

confl.
Theorem

For all strategy classes $\Omega_1$ and $\Omega_2$: if $\Omega_1 \subseteq \Omega_2$, then every $\Omega_2$-confluent program is also $\Omega_1$-confluent.

(follows directly from the definitions)
Definition (general CHR machine)

A CHR machine is a tuple $\mathcal{M} = (\mathcal{H}, \Omega, \mathcal{P}, \mathcal{VG})$ where the host-language $\mathcal{H}$ defines a built-in constraint theory $\mathcal{D}_\mathcal{H}$, $\mathcal{P}$ is a CHR program, $\mathcal{VG} \subseteq \mathcal{G}_\mathcal{P}^{\mathcal{H}}$ is a set of valid goals, and $\Omega \subseteq \Omega_t^\mathcal{H}(\mathcal{P})$ is a strategy class. The machine takes an input query $\mathcal{G} \in \mathcal{VG}$, picks any execution strategy $\xi \in \Omega$, and executes a derivation $d \in \xi|_\mathcal{G}$.

Note that we no longer require the program to be $\Delta^{\mathcal{H}}_{\omega_t}$-deterministic for valid input, and we allow it to use any strategy class.
Do generalized CHR machines have more power than abstract deterministic CHR machines?
- yes, for exotic strategy classes
- not really for the usual (refined, priority):

\[ P_{\Omega_t} = P_{\Omega_r} = P_{\Omega_p} = P \]
1 Introduction

2 Strategy classes

3 Non-deterministic CHR machines
   - Definition
   - Example
   - Complexity

4 Self-modifying CHR machines

5 Conclusion
We define non-deterministic CHR machines similarly to the way non-deterministic Turing machines are defined.

A non-deterministic CHR machine (NCHR machine) is a tuple $\mathcal{M} = (\mathcal{H}, \mathcal{Q}, \mathcal{P}, \mathcal{V}_G)$, where $\mathcal{H}$, $\mathcal{Q}$, $\mathcal{P}$, and $\mathcal{V}_G$ are defined as before. The machine takes an input query $G \in \mathcal{V}_G$ and considers all execution strategies $\xi \in \mathcal{Q}$. If there are strategies that result in a successful derivation $d \in \xi|_G$, any of those is returned. Otherwise, any failure derivation is returned. If all derivations are infinite, any infinite derivation is returned.
Consider NCHR program $\mathcal{P}_{3\text{SAT}}$:

\[
\begin{align*}
&\text{clause}(A,_,_) \iff \text{true}(A). \\
&\text{clause}(_,B,_) \iff \text{true}(B). \\
&\text{clause}(_,_,C) \iff \text{true}(C). \\
&\text{true}(X), \text{true}(\text{not}(X)) \iff \text{fail}.
\end{align*}
\]

Corresponding NCHR machine

$(\emptyset, \Omega_t^\mathcal{H}, \mathcal{P}_{3\text{SAT}}, 3\text{SATCLAUSES})$ decides 3SAT in linear time.

3SAT is NP-complete, so $NP \subseteq NP_{\Omega_t^\mathcal{H}}$. 
\[ P = P_{\Omega_t^\mathcal{H}} \subseteq P_{\Omega_r^\mathcal{H}} \subseteq P_{\{\text{K.U.Leuven}\}} = NP_{\{\text{K.U.Leuven}\}} \subseteq NP_{\Omega_r^\mathcal{H}} \subseteq NP_{\Omega_t^\mathcal{H}} = NP \]

- most of the inclusions collapse to equalities:
  - \( P_{\Omega_t^\mathcal{H}} = P_{\{\text{K.U.Leuven}\}} \) because the RAM-simulator program of [CHR’05] is \((\Omega_t^\mathcal{H})\)-confluent
1 Introduction
2 Strategy classes
3 Non-deterministic CHR machines
4 Self-modifying CHR machines
   - RASP machines
   - Definition
   - Complexity
5 Conclusion
RASP machine

- RASP machine = RAM machine with Stored Program
- much like real computers; von Neumann architecture
- data and program instructions in the same memory space
- so programs can be self-modifying
- in terms of complexity, RASP machine = RAM machine
  - you can simulate a RASP on a RAM with constant factor overhead
- what about self-modifying CHR machines?
encode CHR program using “reserved keyword” constraints

- rule/1, khead/2, rhead/2, guard/2, body/2

Example:

```prolog
foo @ bar ==> baz.
qux @ blarg \ wibble <=> flob | wobble.
```

would be encoded as

```prolog
rule(foo), khead(foo,bar), guard(foo,true), body(foo,baz),
rule(qux), khead(qux,blarg), rhead(qux,wibble),
     guard(qux,flob), body(qux,wobble)
```
CHRSP machines

- program in constraint store
- program is put in the query (or in the initial store)
- rule can only fire if there are no pending rule components to be introduced
CHRSP machine decides co-NP-complete languages in only linear time

Example: Hamiltonian paths in directed graphs
language of graphs without Hamiltonian path is co-NP-complete

consider this CHRSP program:

```prolog
size(N) <= rule(find_path), size(N,A).
size(N,A) <= N>1 |
    khead(find_path,node(A)), khead(find_path,edge(A,B)),
    size(N-1,B).
size(1,A) <=
    khead(find_path,node(A)), body(find_path,fail).
```

input constraints: edge/2, node/1, size(n) (n = #nodes)

program makes rule: find_path @
    node(A1),node(A2),...,node(A_n),
    edge(A1,A2),edge(A2,A3),...,edge(A_{n-1},A_n) ==> fail.

this rule fires (rejecting the input graph) if and only if the graph has a Hamiltonian path
If a regular CHR machine exists that can simulate CHRSP machines with only polynomial overhead, then co-NP ⊆ P, and thus P = NP.

So if \( P \neq NP \), then CHRSP machines are strictly more powerful than regular CHR machines.
1 Introduction

2 Strategy classes

3 Non-deterministic CHR machines

4 Self-modifying CHR machines

5 Conclusion
   - Summary
   - Future work
In polynomial time, a regular CHR machine can do what a Turing machine (or a RAM machine) can do:

\[ P_{\Omega_t} = P \]

The same holds for non-deterministic CHR machines:

\[ NP_{\Omega_t} = NP \]

Self-modifying CHR machines are more powerful than regular ones (even though \( P_{\text{RASP}} = P \)), although exact bounds are still an open problem:

\[ coNP \subseteq P^{sp}_{\Omega_t} \subseteq \text{PSPACE} \]
Future work

- precise bounds on the complexity of CHRSP machines
- influence of strategy class on computational power
- non-deterministic self-modifying programs
Questions?