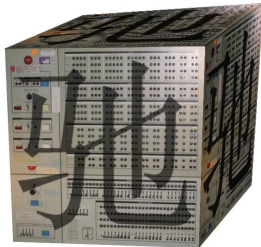


GENERALIZED CHR MACHINES

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 of CHR

3/27

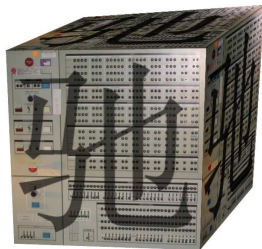
- ▶ Complexity-wise completeness result for CHR (see [CHR'05] and [TOPLAS])
- ▶ “Every algorithm can be implemented in CHR with the optimal time and space complexity”
- ▶ CHR machine: one ω_t step = one machine step
- ▶ In [CHR'05] and [TOPLAS]: compare CHR machine with RAM machine

(Deterministic abstract) CHR machine

4/27

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 $\mathcal{VG} \subseteq \mathcal{G}_{\mathcal{P}}^{\mathcal{H}}$ is a set of *valid goals*, such that \mathcal{P} is a $\Delta_{\omega_t}^{\mathcal{H}}$ -deterministic CHR program for input \mathcal{VG} . The machine takes an input query $\mathbb{G} \in \mathcal{VG}$ and executes a derivation $d \in \Delta_{\omega_t}^{\mathcal{H}} |_{\mathbb{G}}$.



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

Execution strategies

6/27

- ▶ 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)

Strategy classes

7/27

- ▶ set of all execution strategies for a program \mathcal{P} : $\Omega_t^{\mathcal{H}}(\mathcal{P})$
- ▶ a *strategy class* $\Omega(\mathcal{P}) \subseteq \Omega_t^{\mathcal{H}}(\mathcal{P})$ 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 $\Omega_r^{\mathcal{H}}$ is computable
- ▶ so the abstract semantics strategy class $\Omega_t^{\mathcal{H}}$ is computable

Generalized confluence

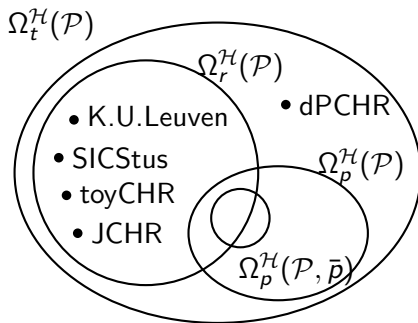
8/27

Definition (Ω -confluence)

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

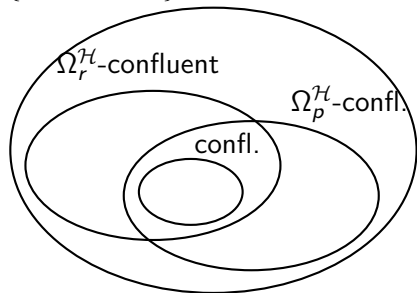
$$\left. \begin{array}{l} \sigma \rightsquigarrow_{\xi_1} \langle \mathbb{G}_1, \mathbb{S}_1, \mathbb{B}_1, \mathbb{T}_1 \rangle_{n_1} \\ \quad \wedge \\ \sigma \rightsquigarrow_{\xi_2} \langle \mathbb{G}_2, \mathbb{S}_2, \mathbb{B}_2, \mathbb{T}_2 \rangle_{n_2} \end{array} \right\} \Rightarrow \mathcal{D}_{\mathcal{H}} \models \bar{\exists}_{\mathbb{G}}(\mathbb{S}_1 \wedge \mathbb{B}_1) \leftrightarrow \bar{\exists}_{\mathbb{G}}(\mathbb{S}_2 \wedge \mathbb{B}_2)$$

Note that a CHR program \mathcal{P} is $\Omega_t^{\mathcal{H}}(\mathcal{P})$ -confluent if and only if it is confluent (in the usual sense)



Execution strategies

{K.U.Leuven}-confluent programs



CHR programs

More instantiated strategy, weaker confluence

10/27

Theorem

For all strategy classes Ω_1 and Ω_2 : if $\Omega_1 \subseteq \Omega_2$, then every Ω_2 -confluent program is also Ω_1 -confluent.

(follows directly from the definitions)

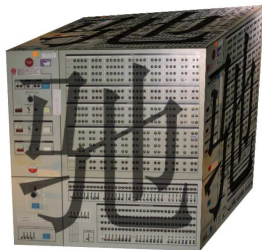
Definition (general CHR machine)

A *CHR machine* is a tuple $\mathcal{M} = (\mathcal{H}, \Omega, \mathcal{P}, \mathcal{W})$ where the host-language \mathcal{H} defines a built-in constraint theory $\mathcal{D}_{\mathcal{H}}$, \mathcal{P} is a CHR program, $\mathcal{W} \subseteq \mathcal{G}_{\mathcal{P}}^{\mathcal{H}}$ 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 $\mathbb{G} \in \mathcal{W}$, picks any execution strategy $\xi \in \Omega$, and executes a derivation $d \in \xi|_{\mathbb{G}}$.

Note that we no longer require the program to be $\Delta_{\omega_t}^{\mathcal{H}}$ -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

Definition

14/27

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

Definition

A *non-deterministic CHR machine* (NCHR machine) is a tuple $\mathcal{M} = (\mathcal{H}, \Omega, \mathcal{P}, \mathcal{W})$, where \mathcal{H} , Ω , \mathcal{P} , and \mathcal{W} are defined as before. The machine takes an input query $G \in \mathcal{W}$ and considers all execution strategies $\xi \in \Omega$. 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.

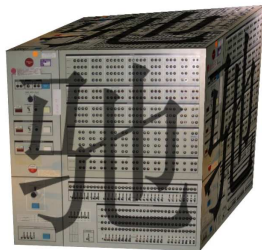
- ▶ Consider NCHR program \mathcal{P}_{3SAT} :

```
clause(A,_,_) <=> true(A).  
clause(_,B,_) <=> true(B).  
clause(_,_,C) <=> true(C).  
true(X), true(not(X)) <=> fail.
```

- ▶ Corresponding NCHR machine $(\emptyset, \Omega_t^{\mathcal{H}}, \mathcal{P}_{3SAT}, 3SATCLAUSES)$ decides 3SAT in linear time
- ▶ 3SAT is NP-complete, so $NP \subseteq NP_{\Omega_t^{\mathcal{H}}}$

$$P = P_{\Omega_t^{\mathcal{H}}} \subseteq P_{\Omega_r^{\mathcal{H}}} \subseteq P_{\{K.U.Leuven\}} = NP_{\{K.U.Leuven\}} \subseteq NP_{\Omega_r^{\mathcal{H}}} \subseteq NP_{\Omega_t^{\mathcal{H}}} = NP$$

- ▶ most of the inclusions collapse to equalities:
 - ▶ $P_{\Omega_t^{\mathcal{H}}} = P_{\{K.U.Leuven\}}$ 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

18/27

- ▶ 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?

Encoding a CHR program in CHR

19/27

- ▶ encode CHR program using “reserved keyword” constraints
- ▶ rule/1, khead/2, rhead/2, guard/2, body/2
- ▶ Example:

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

would be encoded as

```
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)
```

- ▶ 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

Hamiltonian paths in CHRSP

22/27

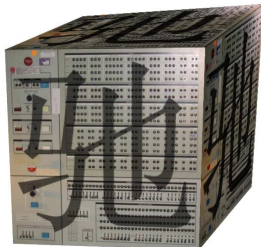
- ▶ language of graphs without Hamiltonian path is co-NP-complete
- ▶ consider this CHRSP program:

```

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: $\text{edge}/2$, $\text{node}/1$, $\text{size}(n)$ ($n = \# \text{nodes}$)
- ▶ program makes rule: `find_path @ node(A1),node(A2),...,node(An), edge(A1,A2),edge(A2,A3),...,edge(An-1,An) ==> 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 $\text{co-NP} \subseteq P$, 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 $P_{RASP} = P$), although exact bounds are still an open problem:

$$coNP \subseteq P_{\Omega_t}^{sp} \subseteq PSPACE$$

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

▶ *Questions?*