The Computational Power and Complexity of Constraint Handling Rules

CHR 2005 presentation

Jon Sneyers, Tom Schrijvers, Bart Demoen
{jon,toms,bmd}@cs.kuleuven.be

Dept. Computer Science, K.U.Leuven, Belgium
Overview

1. Models of computation
2. CHR machines
3. Computational power of CHR
4. Complexity of CHR
5. Conclusion
6. Future work
1. Models of computation: TM

- Turing Machine \( M = \langle Q, \Sigma, s_0, b, F, \delta \rangle \)
  
  - \( Q \): finite set of states
  - \( \Sigma \): tape alphabet
  - \( s_0 \in Q \): initial state
  - \( b \in \Sigma \): blank symbol
  - \( F \subseteq Q \): set of accepting final states
  - \( \delta : Q \setminus F \times \Sigma \mapsto Q \times \Sigma \times \{L, R\} \)
    transition function (L: left shift; R: right shift)

- Operates on an infinite tape, every tape position contains one symbol of \( \Sigma \)
1. Models of computation: RAM

- Random Access Memory (RAM) machine: program (list of instructions) + random-access memory
- Peano-Arithmetic RAM: inc, dec, clr, jump, cjump, halt
- Standard-Arithmetic RAM: PA-RAM + indirection + arithmetic
- Memory cells contain $n$-bit integers
- SA-RAM is realistic model of typical pc
1. Models of computation: PA-RAM

Peano-Arithmetic RAM:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>inc $A$</td>
<td>Increment the value of register $A$ by one.</td>
</tr>
<tr>
<td>dec $A$</td>
<td>Decrement the value of register $A$ by one.</td>
</tr>
<tr>
<td>clr $A$</td>
<td>Set the value of register $A$ to zero.</td>
</tr>
<tr>
<td>jump $L$</td>
<td>Set the program counter to $L$.</td>
</tr>
<tr>
<td>cjump $A$ $L$</td>
<td>If the content of register $A$ is zero, set the program counter to $L$; otherwise continue.</td>
</tr>
<tr>
<td>halt</td>
<td>Halt execution of the RAM.</td>
</tr>
</tbody>
</table>
1. Models of computation: SA-RAM

Standard-Arithmetic RAM: PA-RAM +

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>const</strong></td>
<td>(B) (A_1)</td>
</tr>
<tr>
<td><strong>add</strong></td>
<td>(A_2) (A_1)</td>
</tr>
<tr>
<td><strong>sub</strong></td>
<td>(A_2) (A_1)</td>
</tr>
<tr>
<td><strong>mult</strong></td>
<td>(A_2) (A_1)</td>
</tr>
<tr>
<td><strong>div</strong></td>
<td>(A_2) (A_1)</td>
</tr>
<tr>
<td><strong>move</strong></td>
<td>(A_2) (A_1)</td>
</tr>
<tr>
<td><strong>i_move</strong></td>
<td>(A_2) (A_1)</td>
</tr>
<tr>
<td><strong>move_i</strong></td>
<td>(A_2) (A_1)</td>
</tr>
</tbody>
</table>
1. Models of computation: TM vs RAM

- PA-RAM and SA-RAM are both Turing complete: you can simulate RAM on TM and TM on RAM.
- RAM and TM are polynomially related: you can simulate a $T$-time TM on a RAM in $O(T)$ time.
  - you can simulate a $T$-time PA-RAM on a TM in $O(T)$ time.
  - you can simulate a $T$-time SA-RAM on a TM in $O(T^8)$ time (tighter bounds can be shown).
2. CHR machines
2. CHR machines: motivation

- When studying the complexity of CHR, we want to separate the implementation-dependent from the implementation-independent.

  Implementation-independent part: CHR machine: CHR program + operational semantics

  Implementation-dependent part: CHR compiler + runtime
  ~ simulates a CHR machine on a RAM machine

- Compiler/runtime optimizations
  ~ better simulators of CHR on RAM
2. CHR machines

- CHR machine: CHR program + operational semantics

- Two kinds of CHR programs:
  - CHR-only machine: no host-language built-ins allowed except syntactic (in)equality
  - Minimal host-language CHR machine: arithmetic built-ins allowed (e.g. in Prolog: \texttt{is/2})

- Computation:
  
  input query $\Rightarrow$ transitions of op. sem. $\Rightarrow$ solved form = output
2. CHR machines

CHR

TM

RAM

$O(T)$

$O(T^8)$

$O(T)$

The Computational Power and Complexity of Constraint Handling Rules – p.11
3. Computational power of CHR

- CHR(-only) machines are Turing complete:

![Diagram showing the computational power of CHR](image)
3. Computational power of CHR

- CHR(-only) machines are Turing complete:
  \[ \Rightarrow \] CHR compiler compiles CHR program to (ultimately) RAM machine program
3. Computational power of CHR

- CHR(-only) machines are Turing complete:
  \[ \Rightarrow \] : CHR compiler compiles CHR program to (ultimately) RAM machine program, which can be simulated on a TM.
3. Computational power of CHR

- CHR(-only) machines are Turing complete:
  \[ \Rightarrow \text{CHR compiler compiles CHR program to (ultimately) RAM machine program, which can be simulated on a TM} \]
3. Computational power of CHR

CHR(-only) machines are Turing complete:

⇒: CHR compiler compiles CHR program to (ultimately) RAM machine program, which can be simulated on a TM

⇐: every TM can be simulated on a CHR-only machine
3. Computational power of CHR

Simulating Turing Machines in CHR:

- **input query**: 
  - delta/5 constraints: transition function
  - current_state/1 constraint: initial state
  - cell/4 constraints: initial tape
  - head/1 constraint: initial tape position

- **CHR transitions** map to TM transitions (+ some tape extension overhead)

- **solved form**: remaining cell/4 constraints correspond to TM output
3. Computational power of CHR

Turing Machine Simulator

trans @ delta(S,G,Sp,Gp,Dir) \ current_state(S), head(Cell),
cell(Cell,G,Left,Right)
<= current_state(Sp),
cell(Cell,Gp,Left,Right),
move(Dir,Cell,Left,Right).

move(l,Cell,L,_) <= L \= null | head(L).
move(l,Cell,null,R), cell(Cell,G,null,R)
<= cell(Cell,G,L,R), cell(L,b,null,Cell), head(L).
move(r,Cell,_,R) <= R \= null | head(R).
move(r,Cell,L,null), cell(Cell,G,_,_) 
<= cell(Cell,G,L,R), cell(R,b,Cell,null), head(R).
4. Complexity of CHR: definition

- Time complexity of TM: # steps
- Time complexity of RAM: # cycles
- Time complexity of CHR machine: # transitions

\(
\leadsto \text{lower bound for the realistic time complexity!}
\)

(partner constraint lookup cannot always be done in constant time)

- Realistic complexity of CHR program = \( S(C) \)
  
  \( C \) : complexity of CHR machine (depends only on CHR program)
  
  \( S(T) \) : complexity for simulating a \( T \)-time CHR machine on a
  RAM machine (depends on the CHR compiler/runtime)
4. Complexity of CHR: RAM $\Rightarrow$ CHR

- $O(T)$
- $O(T^8)$
- ?

The Computational Power and Complexity of Constraint Handling Rules – p.16
Results:

- **CHR-only machine can simulate** $T$-time PA-RAM in $O(T)$ time
- Minimal host-language CHR machine can simulate $T$-time SA-RAM in $O(T)$ time
4. Complexity of CHR: RAM \(\Rightarrow\) CHR

Simulating RAM machines on CHR machines:

- **input query**:
  - \(\text{prog(Label,Next,Instruction [,args])}\) constraints: RAM program
  - \(m/2\) constraints: encode memory cells:
    - PA-RAM simulator: successor constraint \(s/2\):
      - \(A=0 : m(A,\text{zero})\)
      - \(A=1 : m(A,B), s(B,\text{zero})\)
      - \(A=2 : m(A,B), s(B,C), s(C,\text{zero})\) etc.
    - SA-RAM simulator: simply \(m(\text{Address},\text{Value})\)
  - \(pc/1\) constraint: program counter initialization

- **CHR transitions** map to RAM machine cycles

- **solved form**: remaining \(m/2\) (and \(s/2\)) constraints correspond to RAM output
4. Complexity of CHR: PA-RAM $\Rightarrow$ CHR

Peano-arithmetic RAM simulator

\[ \text{prog}(L, L_1, \text{inc}, A) \setminus m(A, X), \text{pc}(L) \leftrightarrow m(A, Z), s(Z, X), \text{pc}(L_1). \]
\[ \text{prog}(L, L_1, \text{dec}, A) \setminus m(A, X), s(X, Y), \text{pc}(L) \leftrightarrow m(A, Y), \text{pc}(L_1). \]
\[ \text{prog}(L, L_1, \text{clr}, A) \setminus m(A, X), \text{pc}(L) \leftrightarrow m(A, \text{zero}), \text{pc}(L_1). \]
\[ \text{prog}(L, L_1, \text{jump}, A) \setminus \text{pc}(L) \leftrightarrow \text{pc}(A). \]
\[ \text{prog}(L, L_1, \text{cjump}, A, L_2), m(A, \text{zero}) \setminus \text{pc}(L) \leftrightarrow \text{pc}(L_2). \]
\[ \text{prog}(L, L_1, \text{cjump}, A, L_2), m(A, X), s(X, \_ ) \setminus \text{pc}(L) \leftrightarrow \text{pc}(L_1). \]
\[ \text{prog}(L, L_1, \text{halt}) \setminus \text{pc}(L) \leftrightarrow \text{true}. \]
4. Complexity of CHR: SA-RAM ⇒ CHR

Standard-arithmetic RAM simulator

\[
\begin{align*}
\text{prog}(L, L1, \text{const}, B, A) \ & \ m(A,\_), \ pc(L) \iff m(A, B), \ pc(L1). \\
\text{prog}(L, L1, \text{add}, B, A), \ & \ m(B, Y) \ \ m(A, X), \ pc(L) \iff Z \text{ is } X+Y, \ m(A, Z), \ pc(L1). \\
\text{prog}(L, L1, \text{sub}, B, A), \ & \ m(B, Y) \ \ m(A, X), \ pc(L) \iff Z \text{ is } X-Y, \ m(A, Z), \ pc(L1). \\
\text{prog}(L, L1, \text{mult}, B, A), \ & \ m(B, Y) \ \ m(A, X), \ pc(L) \iff Z \text{ is } X*Y, \ m(A, Z), \ pc(L1). \\
\text{prog}(L, L1, \text{div}, B, A), \ & \ m(B, Y) \ \ m(A, X), \ pc(L) \iff Z \text{ is } X//Y, \ m(A, Z), \ pc(L1). \\
\text{prog}(L, L1, \text{move}, B, A), \ & \ m(B, X) \ \ m(A,\_), \ pc(L) \iff m(A, X), \ pc(L1). \\
\text{prog}(L, L1, \text{i\_move}, B, A), \ & \ m(B, C), \ m(C, X) \ \ m(A,\_), \ pc(L) \iff m(A, X), \ pc(L1). \\
\text{prog}(L, L1, \text{move\_i}, B, A), \ & \ m(B, X), \ m(A, C) \ \ m(C,\_), \ pc(L) \iff m(C, X), \ pc(L1). \\
\text{prog}(L, L1, \text{jump}, A) \ & \ pc(L) \iff pc(A). \\
\text{prog}(L, L1, \text{cjump}, R, A), \ & \ m(R, 0) \ \ pc(L) \iff pc(A). \\
\text{prog}(L, L1, \text{cjump}, R, A), \ & \ m(R, X) \ \ pc(L) \iff X =\&= 0 \ | \ pc(L1). \\
\text{prog}(L, L1, \text{halt}) \ & \ pc(L) \iff \text{true}.
\end{align*}
\]
4. Complexity of CHR: CHR $\Rightarrow$ RAM

The Computational Power and Complexity of Constraint Handling Rules – p.21
4. Complexity of CHR: $\text{CHR} \Rightarrow \text{RAM}$

The diagram illustrates the computational power and complexity of Constraint Handling Rules (CHR) compared to Turing Machines (TM) and RAM machines. The complexity is expressed as $O(T)$, $O(T^8)$, and $O(T^8)$, indicating the time complexity for these computational models.
4. Complexity of CHR: CHR $\Rightarrow$ RAM

Simulating CHR machines on RAM machines:

- CHR compilers basically transform a CHR program to a RAM program

- Assume all host-language statements take constant time (e.g. CHR-only or minimal host-language CHR)

- In general, a SA-RAM can simulate a $T$-time CHR machine in $O(T^m)$ time where $m$ is the maximal number of head constraints in a rule

- Better bounds can be shown for a given program if partner lookup can be done faster…
4. Complexity of CHR: $\text{RAM} \Rightarrow \text{CHR} \Rightarrow \text{RAM}$
4. Complexity of CHR: \( \text{RAM} \Rightarrow \text{CHR} \Rightarrow \text{RAM} \)
4. Complexity of CHR: $\text{RAM} \Rightarrow \text{CHR} \Rightarrow \text{RAM}$
4. Complexity of CHR: $\text{RAM} \Rightarrow \text{CHR} \Rightarrow \text{RAM}$

- Big question: *Given a $T$-time RAM program, does an equivalent CHR program exist which can be compiled (using an existing compiler) to an $O(T)$-time RAM program?*

- SA-RAM simulator has rules with 5 heads $\Rightarrow$ naive compiler results in $O(T^5)$-time RAM program $\sim$ not good enough!

- Using hash-table constraint stores and intelligent scheduling of partner constraint lookups, we get an $O(T)$-time RAM program!

- K.U.Leuven CHR system (e.g. in SWI-Prolog) can do this

- Same for space complexity
Every algorithm can be implemented in CHR with the best known time/space complexity!

* implicit constant factor might be paralyzingly huge; CHR program might be disgustingly ugly
5. Conclusion: summary

The Computational Power and Complexity of Constraint Handling Rules – p.26
6. Future work

- Investigate constant factor / elegance
- Linear speedup theorem for CHR machines? (combining rules to shorten derivations by a constant factor)
- What kind of CHR rules allow constant-time partner constraint lookup?
- Alternative semantics for non-deterministic CHR machines?
- Parallel CHR machines?
- Self-modifying CHR machines??