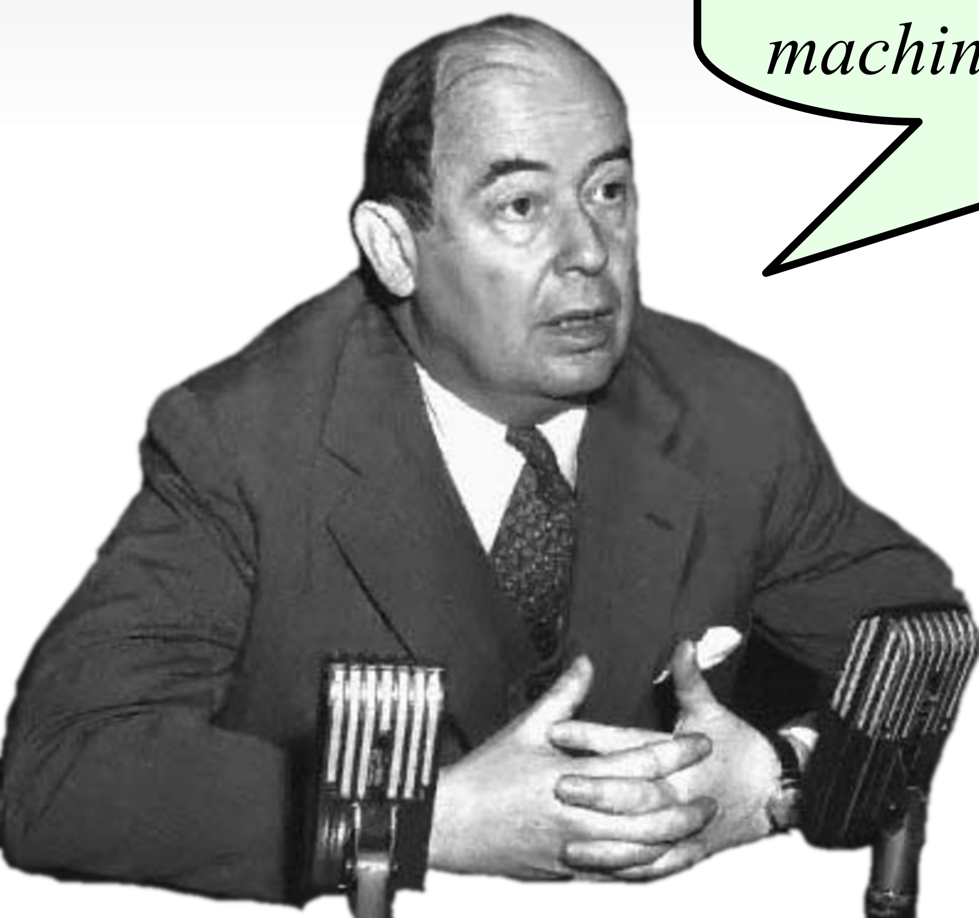
Computability and Complexity of CONSTRAINT HANDLING RULES

Jon Sneyers

August 2010

von Neumann quote

"You insist that there is something that a machine can't do. If you will tell me ***precisely*** *what it is that a machine cannot do, then I can always make a machine which will do just that."*



John von Neumann (1903-1957)

Hungarian-American mathematician,
pioneer of computer science

Overview

- **complexity** (and complexity-wise completeness) of CHR
    - Lecture one (today): the big picture
    - Lecture two (Thursday): the nasty details
- **computability** of (fragments of) CHR
    - Lecture three (Friday)

PART ONE

Complexity-wise completeness

The big picture

Theory topics (3)

- Program analysis
  - Confluence (Abdennadher, Duck et al, Raiser&Tacchella, Haemmerlé&Fages, ...)
  - Operational equivalence (Abdennadher&Frühwirth)
  - Termination (Frühwirth, Paolo Pilozzi, Dean Voets)
  - Complexity (Frühwirth&Schrijvers, Sneyers, De Koninck)
  - Abstract interpretation (Schrijvers, Stuckey, Duck)
  - ...

Remember the shortest path problem

- How long does it take?
  - **It depends...**

- which algorithm is used ?
- how is it implemented ?
- how large is the map (graph) ?

Computational Complexity Theory

- How does an algorithm **scale** with the input size?

		algorithm A	*algorithm B*
		log-linear	quadratic
	input size (*n*)	$O(n \log n)$	$O(n^2)$
Leuven	5000	2 ms	25 ms
Brussels	50000	23 ms	2.5 seconds
New York City	277863	151 ms	1 min 17 seconds
Florida	1228116	747 ms	25 min, 8 seconds
North America	29883886	22 seconds	10 days, 8 hours, 4 min

Some asymptotic time complexities

Function	Name
O(1)	constant
O(log n)	logarithmic
O(n)	linear
O(n log n)	loglinear, quasilinear
$O(n^2)$	quadratic
$O(n^3)$	cubic
$O(n^k)$ (fixed k)	polynomial
$O(c^n)$ $(c > 1)$	exponential
O(n!)	factorial

What about Dijkstra's algorithm?

- Dijkstra's algorithm is $O(n \log n)$
  - for sparse graphs (in general: $O(m + n \log n)$)
  - if implemented in a good way, e.g. using Fibonacci-heaps
- This is optimal: you cannot do better
- Dijkstra's algorithm can be implemented in CHR (with the optimal complexity)

Some other examples...

Edsger Dijkstra (1930-2002)
Dutch computer scientist

Dijkstra's algorithm

can be implemented efficiently in CHR

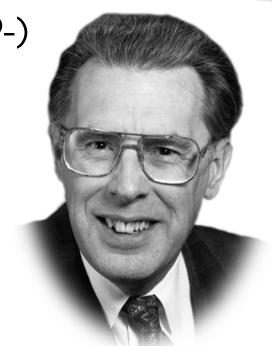
Robert E. Tarjan (1948-)
American computer scientist

Jan van Leeuwen (1946-)
Dutch computer scientist

The Union-Find algorithm

can be implemented efficiently in CHR

John E. Hopcroft (1939-)
American computer scientist



Hopcroft's algorithm

can be implemented efficiently in CHR

?

... can **everything** be implemented efficiently in CHR?

Can **everything** be implemented efficiently in CHR?

Yes!

Complexity-wise completeness result for CHR

Can **everything** … ciently in CHR?

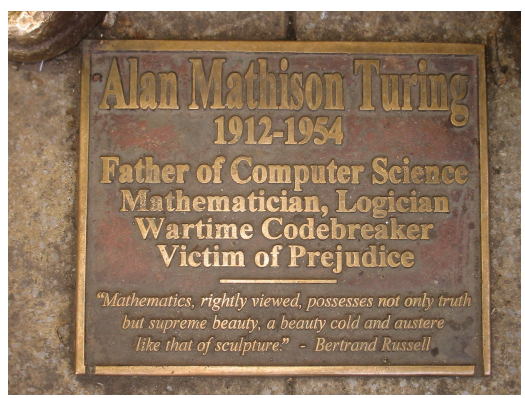
What can be computed ?

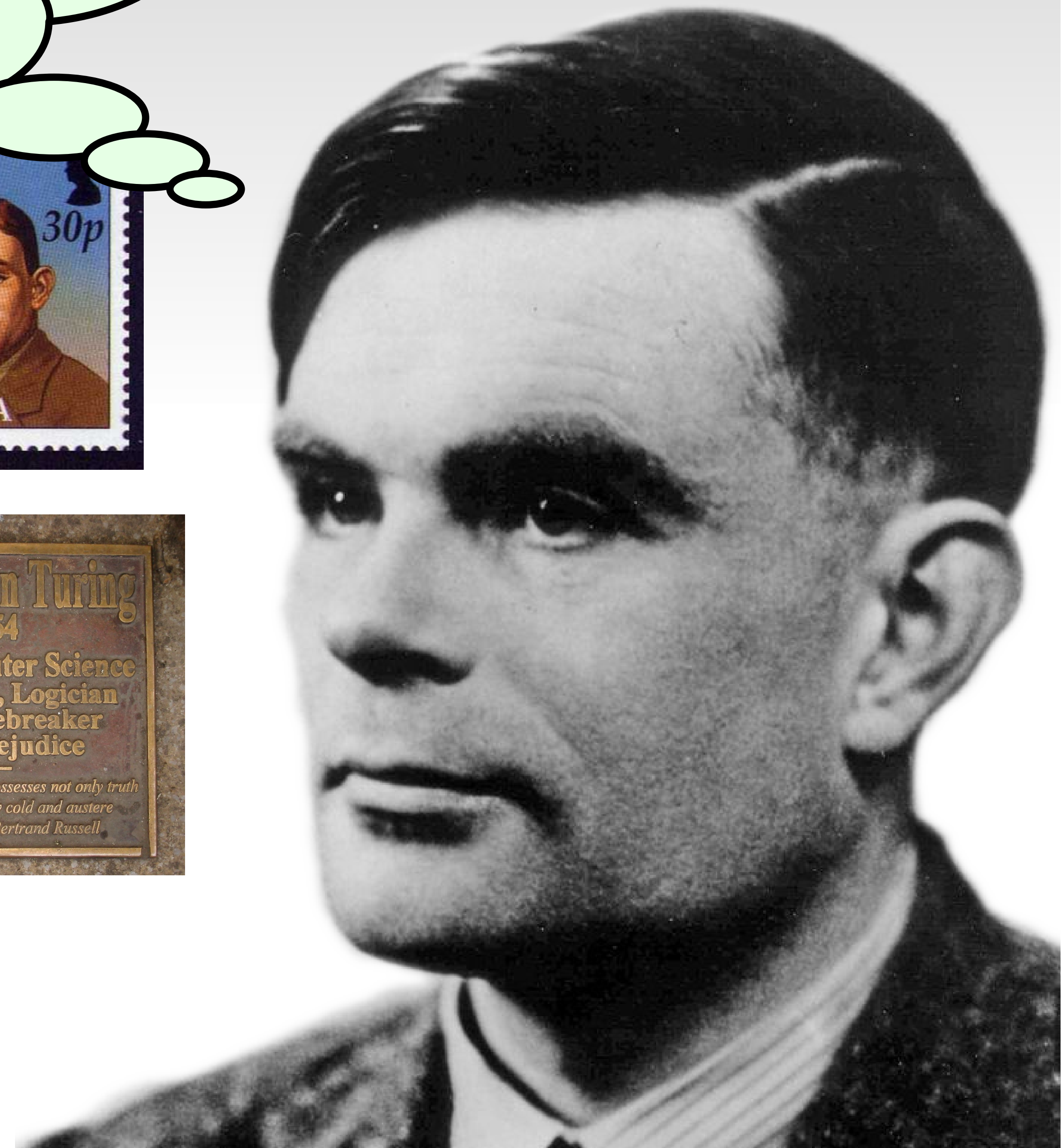
Yes!

Complexity-wise completeness result for CHR

How can you **prove** this?

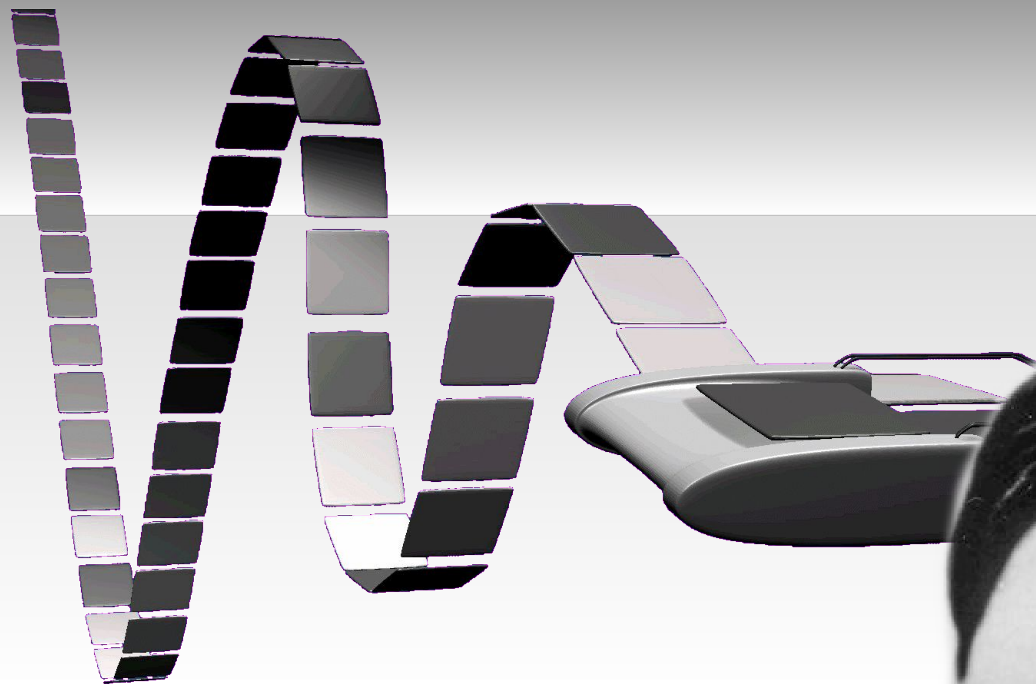
What can be computed ?





Alan Turing (1912-1954)

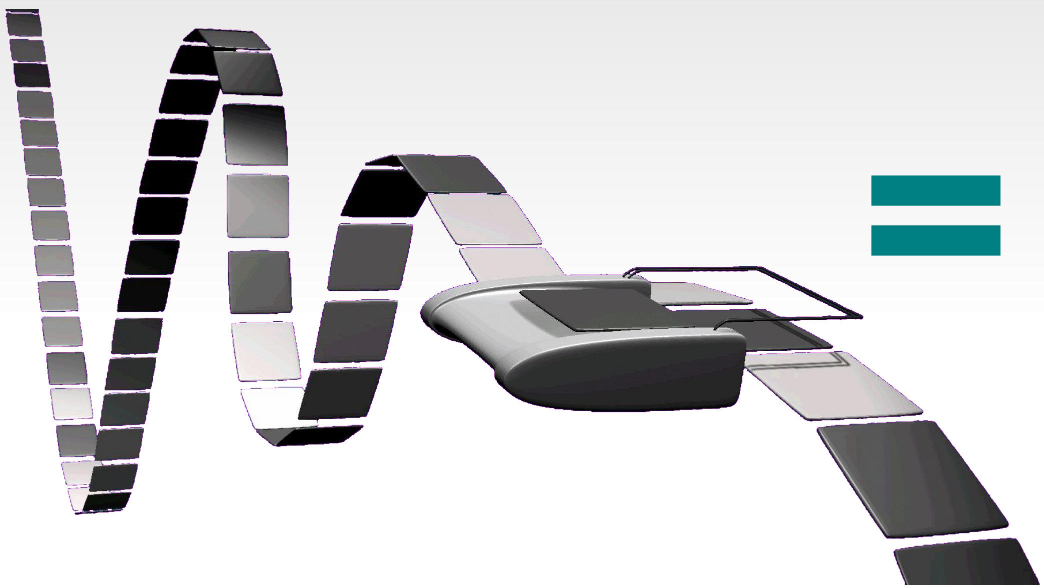
English mathematician,
pioneer of computer science



Turing Machine
model of computation

Alan Turing (1912-1954)
English mathematician,
pioneer of computer science

Models of computation



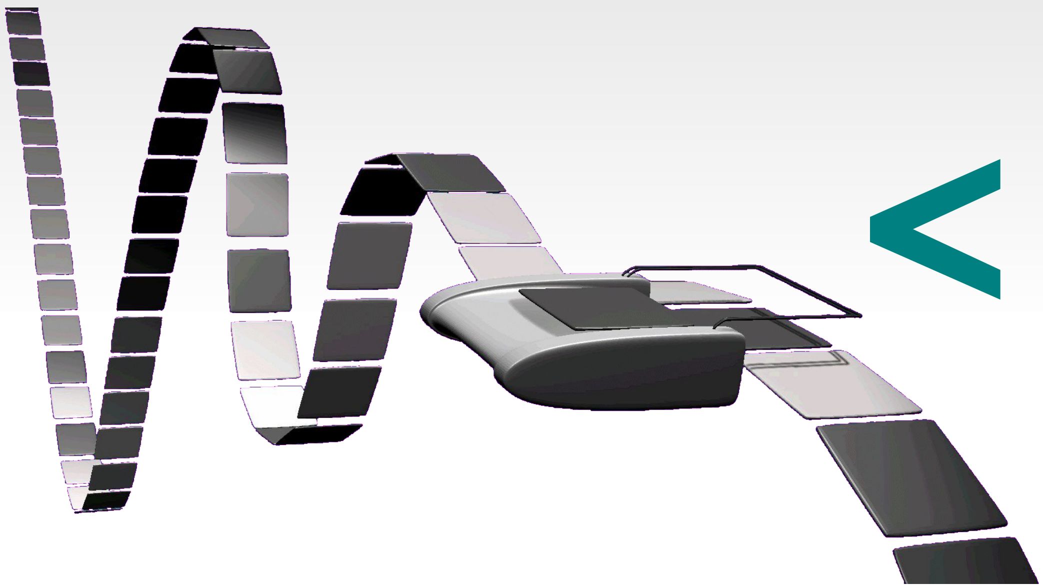
=

Turing machine

RAM machine

what can be computed

Models of computation



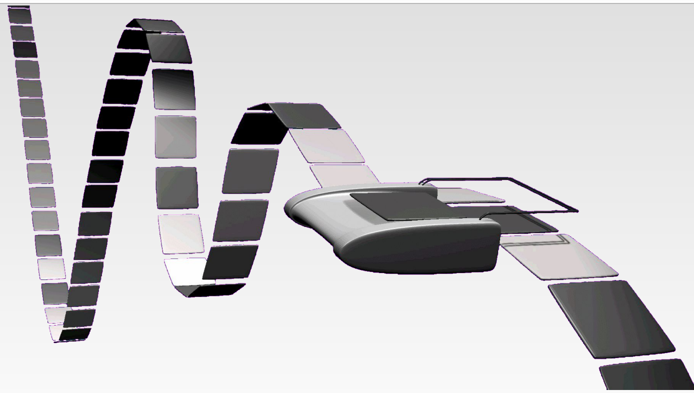
<

Turing machine

RAM machine

how efficiently can things be computed

New model of computation



Turing machine

RAM machine

CHR machine

The CHR machine

CHR machine

RAM machine

what can be computed

The CHR machine

?

CHR machine

RAM machine

how efficiently can things be computed

RAM can simulate CHR

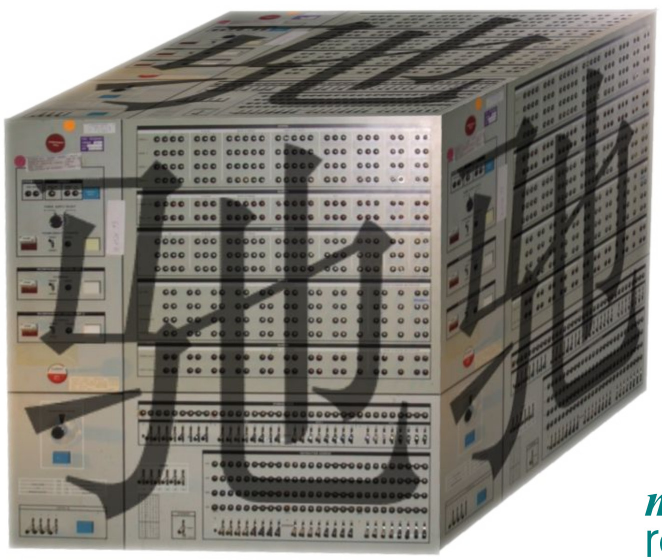
time T, space S

CHR program

```
i(L,init,A), m(A,B), maxm(M) \ c(L) <=> initm(M+1,B,L).
  initm(A,B,L) <=> A =< B | m(A,0), initm(A+1,B,L).
  initm(A,B,L), m(B,X) <=> A > B | m(B,0), maxm(B), c(L+1).
i(L,cnst,B,A) \ m(A,X), c(L) <=> m(A,B), c(L+1).
i(L,add,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X+Y), c(L+1).
i(L,sub,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X-Y), c(L+1).
...
```

RAM program
to simulate CHR
programs

Leuven
CHR
system



CHR machine

time

$$O(TS^{m+1})$$

m = maximum dependency rank of the (non-passive) occurrences in the rules of the CHR program

RAM machine

CHR can simulate RAM

CHR program
to simulate RAM programs

```
i(L,init,A), m(A,B), maxm(M) \ c(L) <=> initm(M+1,B,L).
  initm(A,B,L) <=> A =< B | m(A,0), initm(A+1,B,L).
  initm(A,B,L), m(B,X) <=> A > B | m(B,0), maxm(B), c(L+1).
i(L,cnst,B,A) \ m(A,X), c(L) <=> m(A,B), c(L+1).
i(L,add,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X+Y), c(L+1).
i(L,sub,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X-Y), c(L+1).
i(L,mul,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X*Y), c(L+1).
i(L,div,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X//Y), c(L+1).
i(L,mov,B,A), m(B,Y) \ m(A,_), c(L) <=> m(A,Y), c(L+1).
i(L,imv,B,A), m(B,C), m(C,Y) \ m(A,_), c(L) <=> m(A,Y), c(L+1).
i(L,mvi,B,A), m(B,Y), m(A,C) \ m(C,_), c(L) <=> m(C,Y), c(L+1).
i(L,jmp,A) \ c(L) <=> c(A).
i(L,cjmp,A,J), m(A,0) \ c(L) <=> c(J).
i(L,cjmp,A,J), m(A,X) \ c(L) <=> X =\= 0 | c(L+1).
i(L,halt) \ c(L) <=> true.
```

CHR machine

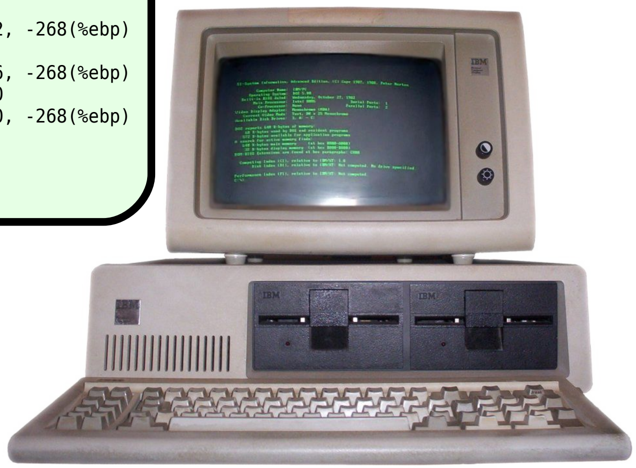
time

O(T)

RAM program

```
.L3:
   cmpl  $100, -268(%ebp)
   je    .L7
   cmpl  $100, -268(%ebp)
   jg    .L11
   cmpl  $97, -268(%ebp)
   je    .L6
   cmpl  $97, -268(%ebp)
   jg    .L12
   cmpl  $0, -268(%ebp)
   je    .L2
   cmpl  $10, -268(%ebp)
   je    .L2
   jmp   .L4
.L12:
   cmpl  $99, -268(%ebp)
   je    .L2
   jmp   .L4
.L11:
   cmpl  $112, -268(%ebp)
   je    .L9
   cmpl  $116, -268(%ebp)
   je    .L10
   cmpl  $110, -268(%ebp)
   je    .L8
...
```

time *T*



RAM machine

Complexity-wise completeness

CHR program
to simulate RAM programs

```
i(L,init,A), m(A,B), maxm(M) \ c(L) <=> initm(M+1,B,L).
  initm(A,B,L) <=> A =< B | m(A,0), initm(A+1,B,L).
  initm(A,B,L), m(B,X) <=> A > B | m(B,0), maxm(B), c(L+1).
i(L,cnst,B,A) \ m(A,X), c(L) <=> m(A,B), c(L+1).
i(L,add,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X+Y), c(L+1).
i(L,sub,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X-Y), c(L+1).
i(L,mul,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X*Y), c(L+1).
i(L,div,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X//Y), c(L+1).
i(L,mov,B,A), m(B,Y) \ m(A,_), c(L) <=> m(A,Y), c(L+1).
i(L,imv,B,A), m(B,C), m(C,Y) \ m(A,_), c(L) <=> m(A,Y), c(L+1).
i(L,mvi,B,A), m(B,Y), m(A,C) \ m(C,_), c(L) <=> m(C,Y), c(L+1).
i(L,jmp,A) \ c(L) <=> c(A).
i(L,cjmp,A,J), m(A,0) \ c(L) <=> c(J).
i(L,cjmp,A,J), m(A,X) \ c(L) <=> X =\= 0 | c(L+1).
i(L,halt) \ c(L) <=> true.
```

RAM program

```
.L3:
   cmpl   $100, -268(%ebp)
   je     .L7
   cmpl   $100, -268(%ebp)
   jg     .L11
   cmpl   $97, -268(%ebp)
   je     .L6
   cmpl   $97, -268(%ebp)
   jg     .L12
   cmpl   $0, -268(%ebp)
   je     .L2
   cmpl   $10, -268(%ebp)
   je     .L2
   jmp    .L4
.L12:
   cmpl   $99, -268(%ebp)
   je     .L2
   jmp    .L4
.L11:
   cmpl   $112, -268(%ebp)
   je     .L9
   cmpl   $116, -268(%ebp)
   je     .L10
   cmpl   $110, -268(%ebp)
   je     .L8
...
```

time T

time $O(T)$

CHR machine

Leuven CHR system

time $O(TS^{m+1})$

RAM machine

Complexity-wise completeness

CHR program
to simulate RAM programs

```
i(L,init,A), m(A,B), maxm(M) \ c(L) <=> initm(M+1,B,L).
  initm(A,B,L) <=> A =< B | m(A,0), initm(A+1,B,L).
  initm(A,B,L), m(B,X) <=> A > B | m(B,0), maxm(B), c(L+1).
i(L,cnst,B,A) \ m(A,X), c(L) <=> m(A,B), c(L+1).
i(L,add,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X+Y), c(L+1).
i(L,sub,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X-Y), c(L+1).
i(L,mul,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X*Y), c(L+1).
i(L,div,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X//Y), c(L+1).
i(L,mov,B,A), m(B,Y) \ m(A,_), c(L) <=> m(A,Y), c(L+1).
i(L,imv,B,A), m(B,C), m(C,Y) \ m(A,_), c(L) <=> m(A,Y), c(L+1).
i(L,mvi,B,A), m(B,Y), m(A,C) \ m(C,_), c(L) <=> m(C,Y), c(L+1).
i(L,jmp,A) \ c(L) <=> c(A).
i(L,cjmp,A,J), m(A,0) \ c(L) <=> c(J).
i(L,cjmp,A,J), m(A,X) \ c(L) <=> X =\= 0 | c(L+1).
i(L,halt) \ c(L) <=> true.
```

RAM program

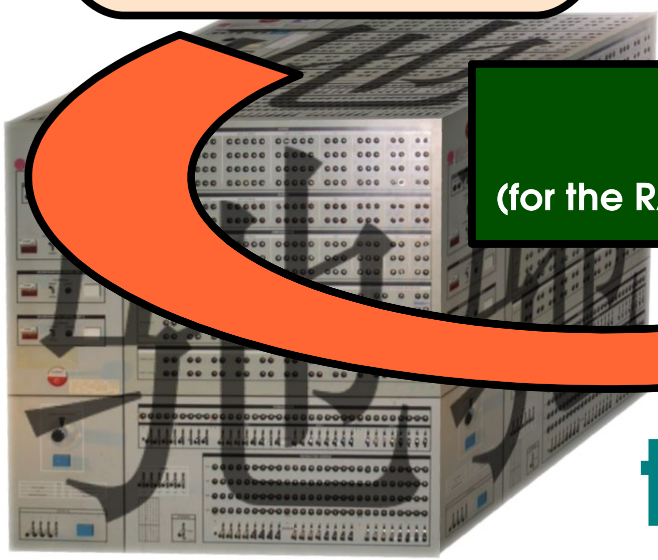
```
.L3:
   cmpl  $100, -268(%ebp)
   je    .L7
   cmpl  $100, -268(%ebp)
   jg    .L11
   cmpl  $97, -268(%ebp)
   je    .L6
   cmpl  $97, -268(%ebp)
   jg    .L12
   cmpl  $0, -268(%ebp)
   je    .L2
   cmpl  $10, -268(%ebp)
   je    .L2
   jmp   .L4
.L12:
   cmpl  $99, -268(%ebp)
   je    .L2
   jmp   .L4
.L11:
   cmpl  $112, -268(%ebp)
   je    .L9
   cmpl  $116, -268(%ebp)
   je    .L10
   cmpl  $110, -268(%ebp)
   je    .L8
...
```

time T

time $O(T)$

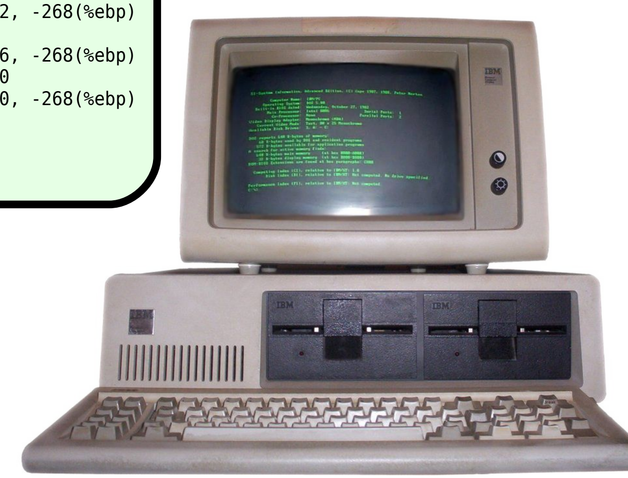
$m = 0$
(for the RAM simulator program)

Leuven CHR system



CHR machine

time $O(T$[redacted]$)$



ground program
(no triggering)

chine

Complexity-wise completeness

CHR program
to simulate RAM programs

```
i(L,init,A), m(A,B), maxm(M) \ c(L) <=> initm(M+1,B,L).
  initm(A,B,L) <=> A =< B | m(A,0), initm(A+1,B,L).
  initm(A,B,L), m(B,X) <=> A > B | m(B,0), maxm(B), c(L+1).
i(L,cnst,B,A) \ m(A,X), c(L) <=> m(A,B), c(L+1).
i(L,add,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X+Y), c(L+1).
i(L,sub,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X-Y), c(L+1).
i(L,mul,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X*Y), c(L+1).
i(L,div,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X//Y), c(L+1).
i(L,mov,B,A), m(B,Y) \ m(A,_), c(L) <=> m(A,Y), c(L+1).
i(L,imv,B,A), m(B,C), m(C,Y) \ m(A,_), c(L) <=> m(A,Y), c(L+1).
i(L,mvi,B,A), m(B,Y), m(A,C) \ m(C,_), c(L) <=> m(C,Y), c(L+1).
i(L,jmp,A) \ c(L) <=> c(A).
i(L,cjmp,A,J), m(A,0) \ c(L) <=> c(J).
i(L,cjmp,A,J), m(A,X) \ c(L) <=> X =\= 0 | c(L+1).
i(L,halt) \ c(L) <=> true.
```

RAM program

```
.L3:
   cmpl  $100, -268(%ebp)
   je    .L7
   cmpl  $100, -268(%ebp)
   jg    .L11
   cmpl  $97, -268(%ebp)
   je    .L6
   cmpl  $97, -268(%ebp)
   jg    .L12
   cmpl  $0, -268(%ebp)
   je    .L2
   cmpl  $10, -268(%ebp)
   je    .L2
   jmp   .L4
.L12:
   cmpl  $99, -268(%ebp)
   je    .L2
   jmp   .L4
.L11:
   cmpl  $112, -268(%ebp)
   je    .L9
   cmpl  $116, -268(%ebp)
   je    .L10
   cmpl  $110, -268(%ebp)
   je    .L8
...
```

time T

time

Leuven
CHR
system

CHR machine

$O(T)$

RAM machine

Can **everything** be implemented efficiently in CHR?

Yes!

Complexity-wise completeness result for CHR

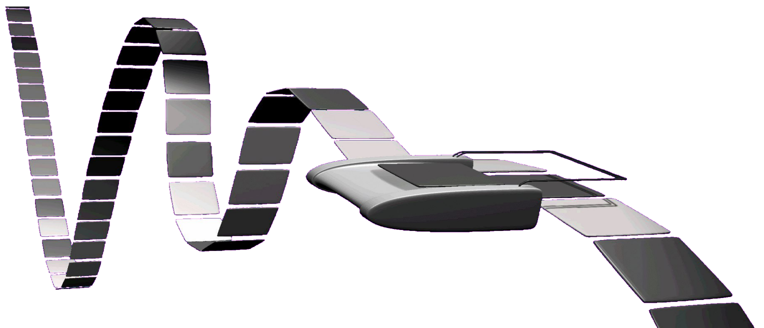
PART TWO

Complexity-wise completeness

The nasty details

Turing machine definition

- Turing machine $\mathbf{M} = \langle \mathbf{Q}, \mathbf{\Sigma}, \mathbf{q_0}, \mathbf{b}, \mathbf{F}, \mathbf{\delta} \rangle$, where
  - $\mathbf{Q}$ is a finite set of states
  - $\mathbf{\Sigma}$ is a finite set of symbols (the tape alphabet)
  - $\mathbf{q_0} \in \mathbf{Q}$ is the initial state
  - $\mathbf{b} \in \mathbf{\Sigma}$ is the blank symbol
  - $\mathbf{F} \subseteq \mathbf{Q}$ are the accepting final states
  - $\mathbf{\delta} : \mathbf{Q} \times \mathbf{\Sigma} \rightarrow \mathbf{Q} \times \mathbf{\Sigma} \times \{\texttt{left}, \texttt{right}\}$
    is the transition function



RAM machine definition

- CPU + RAM memory

address:	0	1	2	3	...
value	[0]	[1]	[2]	[3]	...

- Instruction set:
  - `cnst B,A : [A] := B`
  - `add B,A : [A] := [A]+[B]`
  - `mov B,A : [A] := [B]`
  - `imv B,A : [A] := [[B]]`
  - `mvi B,A : [[A]] := [B]`
  - `cjmp A,L : if [A]=0 goto L`
  - `...`

CHR machine definition

- CHR machine $\mathcal{M} = \langle \mathcal{H}, \Omega, \mathcal{P}, \mathcal{VG} \rangle$
  - Host language $\mathcal{H}$ (e.g. Prolog or "none": Φ)
  - Strategy class Ω (e.g. Ω_t or Ω_r)
  - CHR program $\mathcal{P}$
  - Valid goals $\mathcal{VG}$
- Given an input goal from $\mathcal{VG}$, the program $\mathcal{P}$ is executed according to an execution strategy in Ω and according to the host language $\mathcal{H}$

Turing machine representation in CHR

- Encode an input tape as follows:

…	b	s0	s1	**s2**	s3	s4	s5	b	…

head(C)

		cell (A,s0)	**cell (B,s1)**	**cell (C,s2)**	**cell (D,s3)**	**cell (E,s4)**	**cell (F,s5)**		

	adj (null,A)	**adj (A,B)**	**adj (B,C)**	**adj (C,D)**	**adj (D,E)**	**adj (E,F)**	**adj (F,null)**	

Turing machine representation in CHR

- Encode a TM $\langle \mathbf{Q}, \mathbf{\Sigma}, \mathbf{q_0}, \mathbf{b}, \mathbf{F}, \boldsymbol{\delta} \rangle$ as follows:
  - For each $(\mathbf{q}, \mathbf{s}) \in \mathbf{Q} \times \mathbf{\Sigma}$:
  - If $\boldsymbol{\delta}(\mathbf{q},\mathbf{s}) = (\mathbf{q}',\mathbf{s}',\mathbf{d})$, then add **delta(q,s,q',s',d)**
  - If $\boldsymbol{\delta}(\mathbf{q},\mathbf{s})$ is undefined then add **nodelta(q,s)**
  - For each $\mathbf{q} \in \mathbf{Q} \setminus \mathbf{F}$: add **reject(q)**
  - Add **state($\mathbf{q_0}$)**

Turing machine simulator in CHR

- CHR machine $\mathcal{M}_{\mathcal{TM}} = \langle \Phi, \Omega_t, \textbf{TMSIM}, \mathcal{VG} \rangle$
- **TMSIM** is the following program:

```
r1 @ delta(Q,S,Q2,T,left), adj(L,C) \ state(Q), cell(C,S), head(C)
  <=> L \== null | state(Q2), cell(C,T), head(L).

r2 @ delta(Q,S,Q2,T,right), adj(C,R) \ state(Q), cell(C,S), head(C)
  <=> R \== null | state(Q2), cell(C,T), head(R).

r3 @ delta(Q,S,Q2,T,left) \ adj(null,C), state(Q), cell(C,S), head(C)
  <=> cell(L,b), adj(null,L), adj(L,C), state(Q2), cell(C,T), head(L).

r4 @ delta(Q,S,Q2,T,right) \ adj(C,null), state(Q), cell(C,S), head(C)
  <=> cell(R,b), adj(C,R), adj(R,null), state(Q2), cell(C,T), head(R).

fail @ nodelta(Q,S), reject(Q), state(Q), cell(C,S), head(C) <=> fail.
```

Turing machine simulator in CHR

- Given a TM and an input tape, we can construct an input goal for **TMSIM**
- The TM terminates iff **TMSIM** terminates
- The TM output corresponds to the **TMSIM** output

- Conclusion: CHR machine is Turing complete
  - Actually we've only shown that CHR is **at least** as powerful as Turing machines
  - Since we can execute CHR on a real computer (which is Turing complete), TM are also **at least** as powerful as CHR machines

RAM machine representation in CHR

- **RAMSIMUL** simulates RAM machines in CHR
- We assume a host language that has basic arithmetic (+,−,*,/)
- RAM memory: **m(Address,Value)**
- RAM program: **i(Label,Instruction,Operands)**
- Current instruction: **c(Label)**

RAMSIMUL

```
i(L,init,A), m(A,B), maxm(M) \ c(L) <=> initm(M+1,B,L).
  initm(A,B,L) <=> A =< B | m(A,0), initm(A+1,B,L).
  initm(A,B,L), m(B,X) <=> A > B | m(B,0), maxm(B), c(L+1).
i(L,cnst,B,A) \ m(A,X), c(L) <=> m(A,B), c(L+1).
i(L,add,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X+Y), c(L+1).
i(L,sub,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X-Y), c(L+1).
i(L,mul,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X*Y), c(L+1).
i(L,div,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X//Y), c(L+1).
i(L,mov,B,A), m(B,Y) \ m(A,_), c(L) <=> m(A,Y), c(L+1).
i(L,imv,B,A), m(B,C), m(C,Y) \ m(A,_), c(L) <=> m(A,Y), c(L+1).
i(L,mvi,B,A), m(B,Y), m(A,C) \ m(C,_), c(L) <=> m(C,Y), c(L+1).
i(L,jmp,A) \ c(L) <=> c(A).
i(L,cjmp,A,J), m(A,0) \ c(L) <=> c(J).
i(L,cjmp,A,J), m(A,X) \ c(L) <=> X =\= 0 | c(L+1).
i(L,halt) \ c(L) <=> true.
```

Everything can be done in CHR

- But what about the time/space complexity?

1. What is lost when we simulate a RAM machine on a CHR machine?

2. How fast can a CHR machine be implemented in reality? (i.e., on a RAM machine)

Time complexity definition (TM)

- Definition:

  The time complexity of a TM is a function

  - Given an input size ***n*** (the number of non-blank cells on the input tape)
  - Gives the maximal derivation length for inputs of size ***n*** (the derivation length is the number of transition steps)

- We are typically only interested in asymptotic time complexities (big-O notation)

Time complexity definition (RAM)

- Similar definition for RAM machines
- Number of instructions executed is what counts

Time complexity definition (CHR)

- Similar definition for CHR machines
- Number of ω_t transitions is what counts

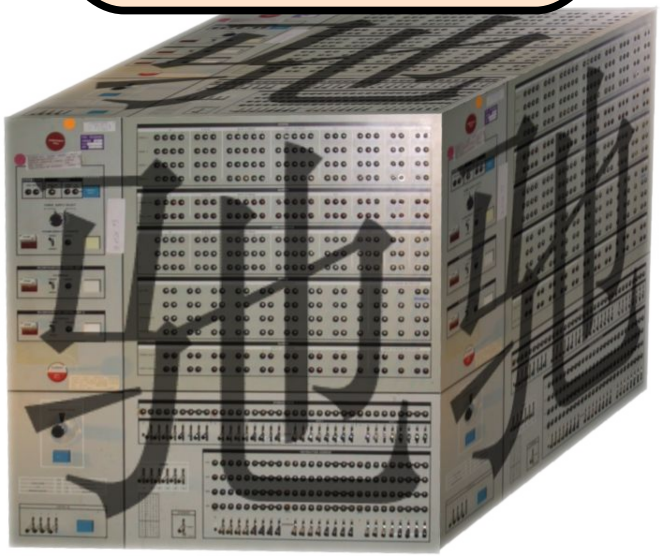
Space complexity definitions

- Space used by a TM is the maximal number of tape cells used during execution
- Space used by a RAM machine is the number of memory cells it uses multiplied by the number of bits needed to represent the largest memory value (often assumed constant, e.g. 64 bit)
- Space used by a CHR machine is the maximal space needed to represent an execution state (constraint store, built-in store, propagation history)

CHR can simulate RAM efficiently

RAMSIMUL

```
i(L,init,A), m(A,B), maxm(M) \ c(L) <=> initm(M+1,B,L).
  initm(A,B,L) <=> A =< B | m(A,0), initm(A+1,B,L).
  initm(A,B,L), m(B,X) <=> A > B | m(B,0), maxm(B), c(L+1).
i(L,cnst,B,A) \ m(A,X), c(L) <=> m(A,B), c(L+1).
i(L,add,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X+Y), c(L+1).
i(L,sub,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X-Y), c(L+1).
i(L,mul,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X*Y), c(L+1).
i(L,div,B,A), m(B,Y) \ m(A,X), c(L) <=> m(A,X//Y), c(L+1).
i(L,mov,B,A), m(B,Y) \ m(A,_), c(L) <=> m(A,Y), c(L+1).
i(L,imv,B,A), m(B,C), m(C,Y) \ m(A,_), c(L) <=> m(A,Y), c(L+1).
i(L,mvi,B,A), m(B,Y), m(A,C) \ m(C,_), c(L) <=> m(C,Y), c(L+1).
i(L,jmp,A) \ c(L) <=> c(A).
i(L,cjmp,A,J), m(A,0) \ c(L) <=> c(J).
i(L,cjmp,A,J), m(A,X) \ c(L) <=> X =\= 0 | c(L+1).
i(L,halt) \ c(L) <=> true.
```



CHR machine

time $O(T)$

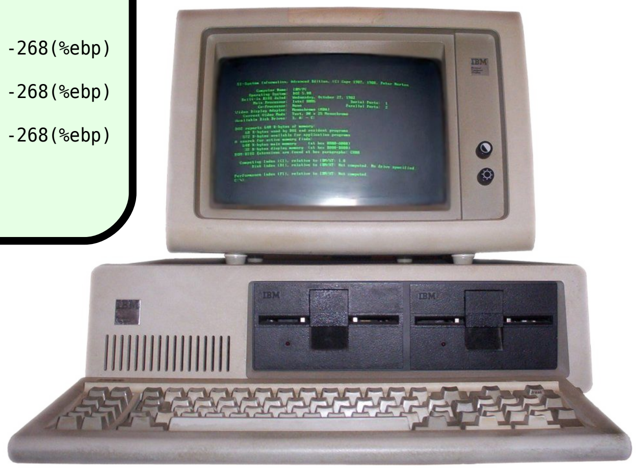
space $O(S)$

RAM program

```
.L3:
   cmpl   $100, -268(%ebp)
   je     .L7
   cmpl   $100, -268(%ebp)
   jg     .L11
   cmpl   $97, -268(%ebp)
   je     .L6
   cmpl   $97, -268(%ebp)
   jg     .L12
   cmpl   $0, -268(%ebp)
   je     .L2
   cmpl   $10, -268(%ebp)
   je     .L2
   jmp    .L4
.L12:
   cmpl   $99, -268(%ebp)
   je     .L2
   jmp    .L4
.L11:
   cmpl   $112, -268(%ebp)
   je     .L9
   cmpl   $116, -268(%ebp)
   je     .L10
   cmpl   $110, -268(%ebp)
   je     .L8
...
```

time T

space S



RAM machine

Can RAM machines simulate CHR machines?

- This is what a CHR compiler does!
- See Peter Van Weert's lectures on optimizing compilation
- Refined semantics compilation:
  - Active constraint seeks partner constraints
  - If there are S constraints in the store, and there are *p* partner heads, this can take $O(S^p)$ time
  - After rule application, constraints can be triggered and reactivated, which can take $O(S^{p+1})$ time

Meta-complexity result (1)

- Given a CHR machine $\mathcal{M}$ which takes time T and space S, and all rules have at most *n* heads, then $\mathcal{M}$ can be simulated on a RAM machine using $O(T\,S^n)$ time and O(S) space.

  (if the refined semantics can be used and the host language built-ins take constant time to evaluate)

- **RAMSIMUL** has rules with 5 heads, so it can be executed in $\mathbf{O(TS^5)}$ and space O(S)

No triggering

- If there is no triggering (for example when all constraints are always ground), then the $O(T S^{n})$ is reduced to $O(T S^{n-1})$
- **RAMSIMUL** uses only ground constraints, so it can be executed in $\mathbf{O(T S^{4})}$

Determined partners

- There can be functional dependencies between constraint arguments
  - For example in **RAMSIMUL**: **m(Address,Value)**
    - Given an **Address**, there is only one **m**/2 constraint
- A **determined partner** is a partner constraint that is uniquely determined by the active constraint or recursively by already determined partners (w.r.t. some join ordering)
  - Using efficient constraint store indexing, a determined partner can be found in constant time

Example of determined partners

i(L,imv,B,A), m(B,C), m(C,Y) \ m(A,_), c(L) <=> m(A,Y), c(L+1).

- **c(L)** is the active constraint
  - **L** is given, so **i(L,_,_,_)** is determined (only one instruction per label)
  - Now given **A**, we can find **m(A,_)**, and given **B**, we can find **m(B,_)**
  - Now given **C** we can find **m(C,_)**
- So for this occurrence of **c**/1 and given this join ordering, all partner constraints are determined

Dependency rank

- The **dependency rank** of a constraint occurrence (w.r.t. some join ordering) is the number of partner constraints that are *not* determined
- E.g. in the previous example, the dependency rank of **c**/1 is zero.
- Trivial bound:

  the dependency rank ≤ the number of partners (which is ≤ $n-1$)

Meta-complexity result (2)

- Given a CHR machine $\mathcal{M}$ which takes time T and space S, and (w.r.t. some join ordering) **the maximal dependency rank of all (non-passive) occurrences is *m***, then $\mathcal{M}$ can be simulated on a RAM machine using $O(T\,S^{m+1})$ time.
- If there is no triggering (e.g. ground program), then the time improves to $O(T\,S^{m})$
- **RAMSIMUL** has maximal dependency rank m=0

Q.E.D.

PART THREE

Computability of fragments of CHR

What language features are really needed?

CHR is Turing complete

- **TMSIM** shows that CHR is Turing complete, even
  - Without host language
  - Only variables and constants (no complex terms)
  - Without propagation rules
- What about syntactic fragments of CHR?
  - Restricted kind of rules (e.g. #heads)
  - Restricted constraint arguments (arity, data types)
  - Restricted host language

Only propagation rules

- **TMSIM** used simpagation rules to *update* simulated tape cells (delete old, insert new)
- Only propagation rules: nothing can be deleted
- Possible solution: add "kill flag" argument
  - Add one argument to every constraint
  - Initially a variable, instantiate it to a constant to "delete" the constraint, add guards in every rule
  - Requires host language built-ins
- Other solution: add "timestamp" argument

TMSIM-PROP (1)

```
% add timestamps
head(C) ==> inittime(T), head(T,C).
inittime(T), state(Q) ==> state(T,Q).
inittime(T), cell(C,S) ==> cell(T,C,S).
inittime(T), adj(L,R) ==> adj(T,L,R).

% compute next step
r13 @ state(T,Q), head(T,C), cell(T,C,S), delta(Q,S,Q2,S2,left)
  ==> next(T,U), state(U,Q2), cell(U,C,S2), mleft(T,C,U), cright(T).

r24 @ state(T,Q), head(T,C), cell(T,C,S), delta(Q,S,Q2,S2,right)
  ==> next(T,U), state(U,Q2), cell(U,C,S2), mright(T,C,U), cleft(T).

state(T,Q), head(T,C), cell(T,C,S), nodelta(Q,S), reject(Q) ==> fail.
```

TMSIM-PROP (2)

```
% move head, extending tape if needed
mleft(T,C,U), adj(T,L,C) ==> L \== null | head(U,L), cleft(T).
mleft(T,C,U), adj(T,null,C) ==> head(U,L), adj(U,null,L), adj(U,L,C).

mright(T,C,U), adj(T,C,R) ==> R \== null | head(U,R), cright(T).
mright(T,C,U), adj(T,C,null) ==> head(U,R), adj(U,C,R), adj(U,R,null).


% copy non-modified tape to next timestamp
cell(T,C,S), next(T,U), head(T,C2) ==> C \== C2 | cell(U,C,S).
adj(T,L,R), next(T,U) ==> L \== null, R \== null | adj(U,L,R).

cleft(T), next(T,U), adj(T,X,null) ==> adj(U,X,null).
cright(T), next(T,U), adj(T,null,X) ==> adj(U,null,X).
```

How many rules are needed?

- **TMSIM** has 5 rules; can we make a TM simulator using less rules?
- Yes, it turns out 1 rule is enough
- We use a slightly different tape representation
  - At the tape ends we add little loops:
    - adj(null,C) ➔ adj(L,L), adj(L,C), cell(L,b).
    - adj(C,null) ➔ adj(R,R), adj(C,R), cell(R,b).
  - We use redundant adj/3 constraints (one for each direction):
    - adj(A,B) ➔ adj(A,B,left), adj(B,A,right).

One monster rule: **TMSIM-1R**

```
% adj(null,C) <=> adj(L,L), adj(L,C), cell(L,b).
% adj(C,null) <=> adj(R,R), adj(C,R), cell(R,b).
% adj(A,B) <=> adj(A,B,left), adj(B,A,right).

r1234 @ delta(Q,S,Q2,S2,D), state(Q), head(C)
      \ adj(A,C,D), adj(C,B,D), cell(C,S), adj(C,A,E),adj(B,C,E)
      <=> adj(A,C2,D), adj(C2,B,D), adj(C,C,D),
            adj(C2,A,E), adj(B,C2,E), adj(C,C,E),
            cell(C,b), cell(C2,S2), state(Q2), head(A).
```

How many heads are needed?

- Every program can be transformed to a program with only 2-headed rules
  - Consider e.g. a rule of the form A, B, C, D ==> E
  - This rule can be written in three 2-headed rules:
    - A, B ==> X
    - C,D ==> Y
    - X, Y ==> E

  using new auxiliary constraints (X and Y)
- n-headed CHR (n≥2) has the same power as 2-headed CHR (you just need more rules)

Single-headed CHR

- 1-headed CHR is weaker than 2-headed CHR
- If complex terms are allowed (e.g. functors with arbitrary nesting, or numbers with arithmetic), it is still Turing complete
- Otherwise it is not Turing complete

Overview

Host language / data types	1-headed	2-headed (or more)
No arguments (propositional CHR)	Not Turing complete	(Betz 2007)
Only variables and constants, **range-restricted** rules only		(Mauro+ 2010)
Only variables and constants **without** unification	(Sneyers 2008)	(Sneyers+ 2005)
Only variables and constants **with** unification	(Mauro+ 2010)	
Complex arguments (functors and/or arithmetic)	(Di Giusto+ 2008)	Turing complete

Overview

Host language / data types	1-headed		≥ 2-headed	
	Prop	Simp	Prop	Simp
Propositional CHR	Not TC			(Betz 2007)
Propositional CHR, **refined operational semantics**			(Sneyers 2008)	(Sneyers 2008)
Only variables and constants, **range-restricted** rules only				(Mauro+ 2010)
Only variables and constants **without** unification		(Sneyers 2008)	(Sneyers 2008)	(Sneyers + 2005)
Only variables and constants **with** unification		(Mauro+ 2010)		
Complex arguments (functors and/or arithmetic)	(Di Giusto+ 2008)	(Di Giusto+ 2008)		TC