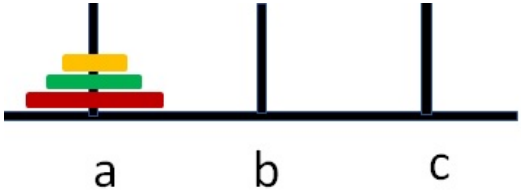
Model checking of CTL formulae

FMSS, 2019-2020

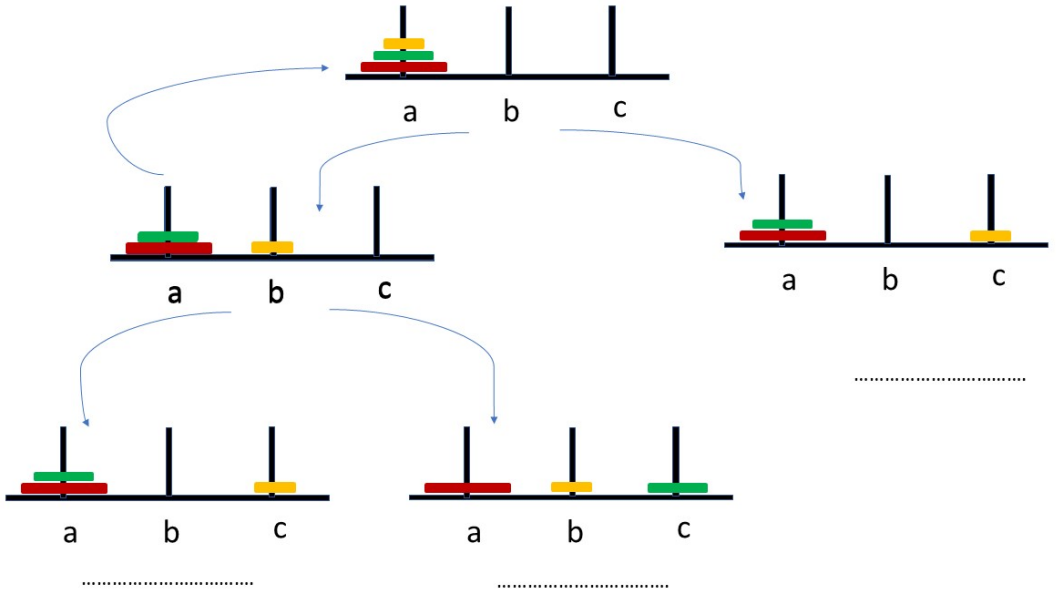
Example: Hanoi Tower



- Three rods (a, b, c) and three disks with different size in order on rod a. The largest disk at the bottom of a.
- Move the entire stack of disks from rod a to rod c, assuming the following rules:
  - only one disk can be moved at a time
  - no larger disk may be placed on top of a smaller disk

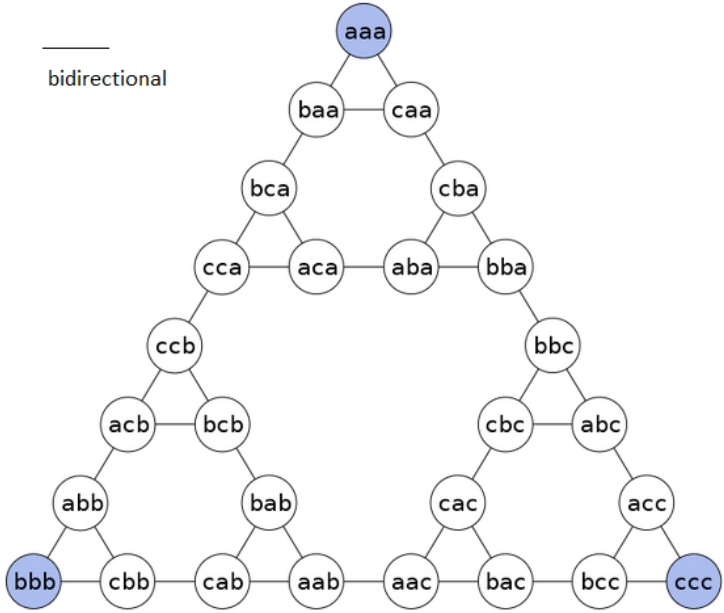
https://en.wikipedia.org/wiki/Tower_of_Hanoi

Hanoi Tower: steps



Transition System

State: list of positions in disk increasing size ($pos_{small}, pos_{medium}, pos_{large}$)



Transition System

$TS = (S, I, \rightarrow, AP, L)$ where:
- S sequences of three letters chosen among `a`, `b`, `c` ;
- $I =$ `aaa`;
- $\rightarrow$ is the transition relation;
- $AP = S$ atomic propositions same as the set of states;
- $L(s) = s$

We can check if along every path it is always possible to reach the configuration "all disks on rod `c`". Let $\phi = ccc$.

A state formula ϕ holds on a TS if it holds for all initial states: $\forall \sigma \in I : \sigma \models \phi$

CTL

STATE FORMULAE

$$\phi ::= tt \mid ap \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid E\Psi \mid A\Psi$$

PATH FORMULAE

$$\Psi ::= X\phi \mid F\phi \mid G\phi \mid \phi_1 U \phi_2$$

$\phi_1 U \phi_2$
path formula which requires that exists a state s such that ϕ_2 holds and ϕ_1 holds in all states up to the state s

Other formulae

ff for $\neg tt$
$\phi_1 \vee \phi_2$ for $\neg((\neg\phi_1) \wedge (\neg\phi_2))$
$\phi_1 \implies \phi_2$ for $((\neg\phi_1) \vee \phi_2)$

Semantics of state formulae

$$
\begin{array}{lll}
\sigma \models \mathit{tt} & \text{iff} & \text{true} \\
\sigma \models \mathit{ap} & \text{iff} & \text{ap} \in L(\sigma) \\
\sigma \models \phi_1 \wedge \phi_2 & \text{iff} & (\sigma \models \phi_1) \wedge (\sigma \models \phi_2) \\
\sigma \models \neg\phi & \text{iff} & \sigma \not\models \phi \\
\sigma \models E\Psi & \text{iff} & \pi : \pi \in \mathit{Path}(\sigma) \wedge \pi \models \Psi \\
\sigma \models A\Psi & \text{iff} & \forall\pi : \pi \in \mathit{Path}(\sigma) \implies \pi \models \Psi
\end{array}
$$

Semantics of path formulae

$$
\begin{array}{lll}
\sigma_0\sigma_1 \cdots \sigma_n \cdots \models X\phi & \text{iff} & \sigma_1 \models \phi \wedge n > 0 \\
\sigma_0\sigma_1 \cdots \sigma_n \cdots \models F\phi & \text{iff} & \sigma_n \models \phi \wedge n \geq 0 \\
\sigma_0\sigma_1 \cdots \sigma_n \cdots \models G\phi & \text{iff} & \forall i : \sigma_i \models \phi \\
\sigma_0\sigma_1 \cdots \sigma_n \cdots \models \phi_2 \bigcup \phi_2 & \text{iff} & ((\sigma_n \models \phi_2 \wedge n \geq 0) \\
 & & \wedge(\forall i \in \{0, , n-1\} : \sigma_i \models \phi_1))
\end{array}
$$

Definitions

- state s is *stuck* if there are no transitions leaving s.
- a path Π in a transition system is a sequence of states $\sigma_0\sigma_1 \cdots \sigma_n \cdots$ such that $\forall i > 0 : \sigma_{i-1} \rightarrow \sigma_i$ and the path is as long as possible.
  $Path(\sigma_0)$ denotes the set of paths $\Pi = \sigma_0\sigma_1 \cdots \sigma_n \cdots$, starting in σ_0.
  If σ' is a stuck state, then $Path(\sigma') = \sigma'$.

Definitions

- Given $S_0 \subseteq S$, $Reach_1(S_0) = \{\sigma_1 \mid \sigma_0\sigma_1 \cdots \sigma_n \cdots \in Path(\sigma_0)$ and $\sigma_0 \in S_0\}$
  states reachable from a state in S_0 in one step.
- Given $S_0 \subseteq S$, $Reach(S_0) = \{\sigma_n \mid \sigma_0\sigma_1 \cdots \sigma_n \cdots \in Path(\sigma_0)$ and $\sigma_0 \in S_0$ and $n \geq 0\}$
  states reachable from a state in S_0 in zero or more steps.
- $Reach(I)$ is the set of reachable states.

Model checker

Model checker: an automatic program that can check if a property holds on the system.
The complexity of model checking for CTL depends on the product of the size of the transition system (TS) and the size of the formula ϕ.

size (TS) = number of states plus number of transitions plus sum over all states of the number of atomic propositions

size(ϕ) = number of symbols

The complexity increases when the TS is constructed by a program graph and the memory represents a set of variables.
Assumptions:

- S is a finite set of states
- there are transitions leaving all reachable states (this reduces the complexity, for example: $AX\phi$ is the same as $\neg(EX(\neg\phi))$) We add self-loops on stuck states.

A model checking algorithm

$$Sat(\phi) = \{\sigma \mid \sigma \models \phi\}$$

Simple cases:

- $Sat(tt) = S$
- $Sat(ap) = \{\sigma \in S \mid ap \in L(\sigma)\}$
- $Sat(\phi_1 \wedge \phi_2) = Sat(\phi_1) \cup Sat(\phi_2)$
- $Sat(\neg\phi) = S - Sat(\phi)$
- $Sat(EX\phi) = \{\sigma \mid Reach_1(\{\sigma\}) \cap Sat(\phi) \neq \{\}\}$
- $Sat(AX\phi) = \{\sigma \mid Reach_1(\{\sigma\}) \subseteq Sat(\phi)\}$

A model checking algorithm

- $Sat(EF\phi) = \{\sigma \mid Reach(\{\sigma\}) \cap Sat(\phi) \neq \{\}\}$
- $Sat(AG\phi) = \{\sigma \mid Reach(\{\sigma\}) \subseteq Sat(\phi)\}$

More complex cases (not shown):

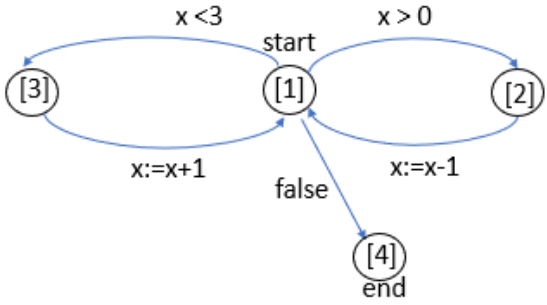
- $Sat(EG\phi)$
- $Sat(AF\phi)$
- $Sat(E(\phi_1 \bigcup \phi_2))$

Analysis of programs

Program = Control Flow Graph (CFG) + Data

- CFG represents the control structure of the program.
- memory represents the data structure on which the program operates.
- the semantics of the programs is based on the memory and the program point. When we execute an instruction we move from a pair (pp, m) to another pair (pp', m'). The value of pp' and m' depends on the values pp and m and the semantics of the instruction that is executed.

An example



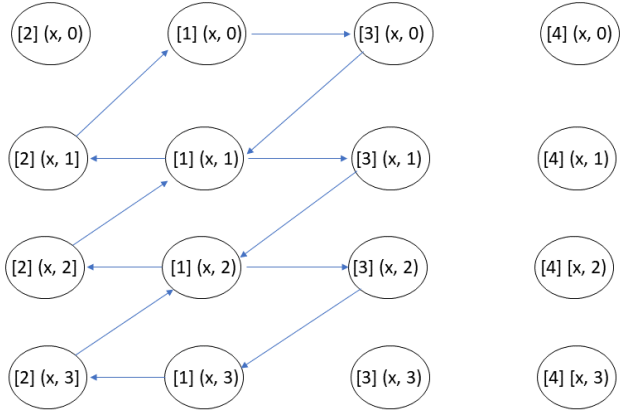
Assume x can take value : 0, 1, 2, 3
Memory m: $(x, 0)$,$(x, 1)$, $(x, 2)$, $(x, 3)$
Program point pp: [1], [2], [3], [4]

Transition System TS

- $S = \{(pp, m) \mid pp \in \{[i], i = 1, 2, 3, 4\} \wedge$
  $m \in \{(x, i), i = 0, 1, 2, 3\}\}$
- $I = \{([1], (x, i)), i = 0, 1, 2, 3\}$
- $\rightarrow$ .....
- $AP = \{@[i], i = 1, 2, 3, 4\} \cup \{@(x, i), i = 0, 1, 2, 3\} \cup$
  $\{start\} \cup \{end\}$
- $L([1], (x, i)) = \{@1, @(x, i), start\}$
  $L([2], (x, i)) = \{@2, @(x, i)\}$
  $L([3], (x, i)) = \{@3, @(x, i)\}$
  $L([4], (x, i)) = \{@4, @(x, i), end\}$

Transition System TS

transition system obtained by the program control flow graph



Analysis of programs

How many states? How many reachable states?

Consider the set of states where the following formula holds:
@[3]
@$(x, 2)$
@$[1] \wedge$ @$(x, 2)$

Termination of the system

$$start \implies EF\ end$$

for each initial state it is possible to terminate

$$start \implies AF\ end$$

for each initial state it is certain to terminate

Analysis of programs

When the transition system is built by the program control flow graph, and the memory has k variables taking values in $\{0, \cdots, n-1\}$, the complexity of the model checking is exponential in the number of variables.
The complexity depends on the product of the size of the program points, the size of the formula and n^k.