Model checking tecniques

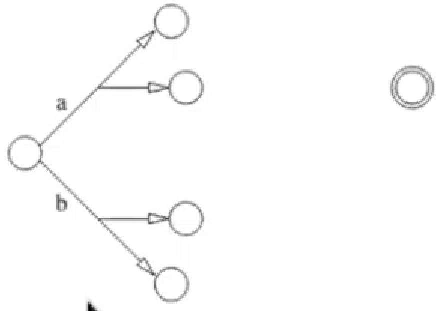
based on backward propagation

use a compact representation for a set of states

backward propagation

base for the BDD-based algorithm

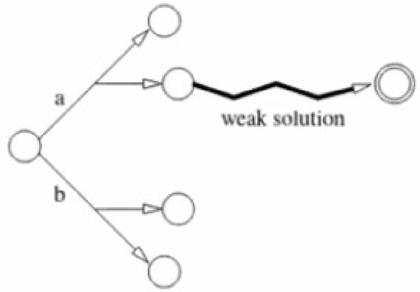
consider a state and its actions
single goal state, somewhere



what if some successor has a path to the goal?

weak policy

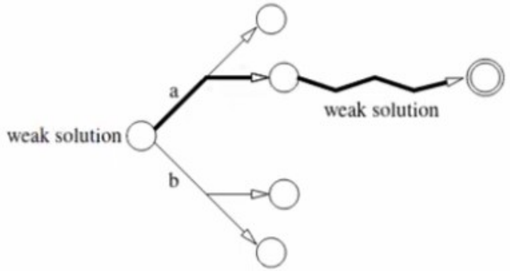
example: goal reachable from a successor



goal reachable from the predecessor

backward expansion of a weak policy

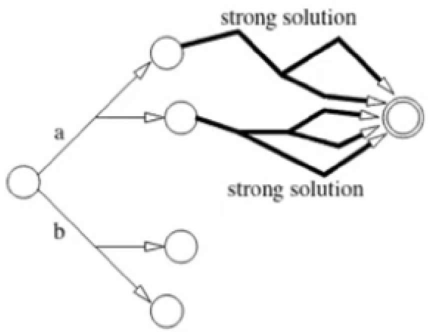
goal reachable from successor ⇒ reachable from predecessor



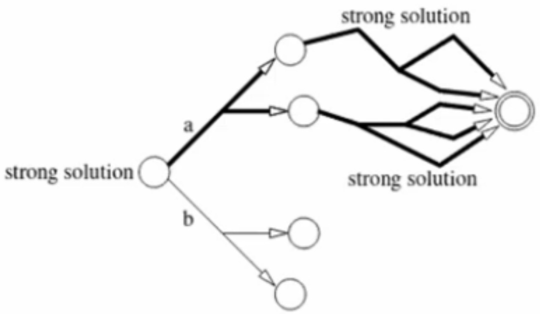
weak policy expanded backwards

strong policy

other example:



expanding a strong policy backwards



requires:

- strong policy from successors
- from **every** successor of **an** action
  choose action a ⇒ action b irrelevant

strong cyclic solution

means:

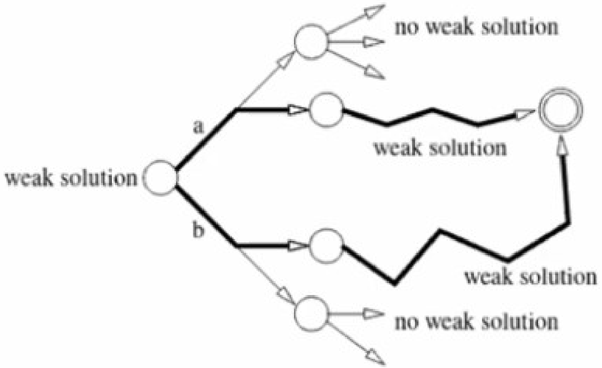
- weak solution
- that is also a weak solution for all states it reaches

first step: expand a weak solution backwards
same as for a weak policy

second step: remove states for which no weak policy exists

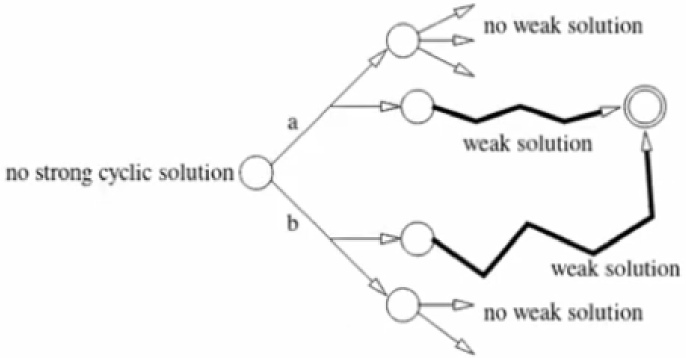
no weak policy

initial condition:



both for a and b:
a successor have no weak solution

backward removal

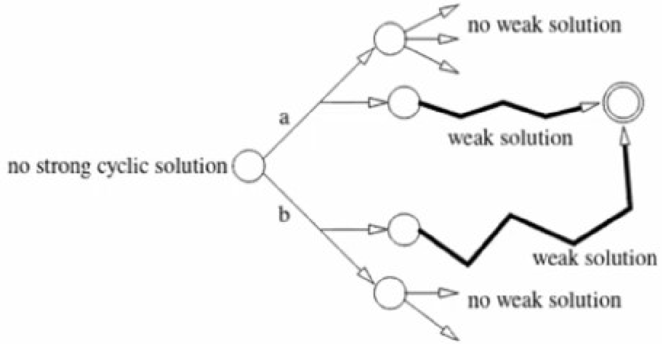


executing a may lead to a state with no weak solution
state reachable, with no weak solution
⇒ no strong cyclic solution for leftmost node, if executing a

holds for b as well

no strong cyclic solution

backward removal, in practice



for **every** action, there exists **at least** a possible a successor with "no weak solution"
⇒ no strong cyclic solution from the state

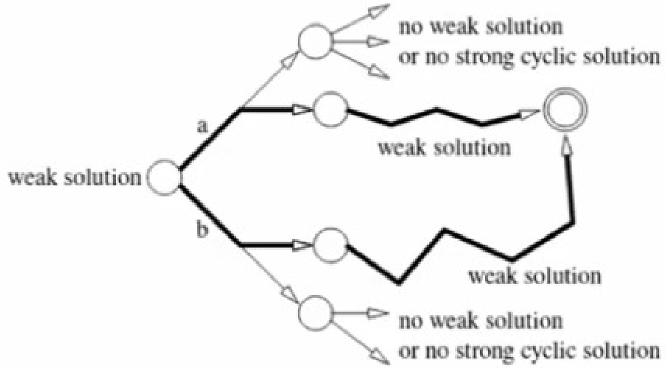
again: the "no weak solution" nodes are the ones that matter

recursion of backward removal

no weak solution for the successor
⇒ no strong cyclic solution for precedecessor

to allow recursion: same condition in successor and predecessor
so that we can repeat from the predecessor, backwards

general condition for backward removal

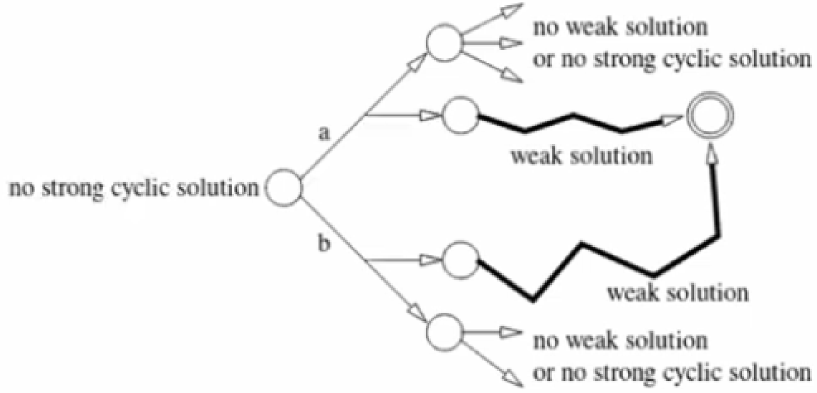


no strong cyclic solution =

- no weak solution for state
  OR
- no weak solution for some reachable state

in both cases: no strong cyclic solution for predecessor

result of backward removal



do it as much as possible
what remains is a strong cyclic solution

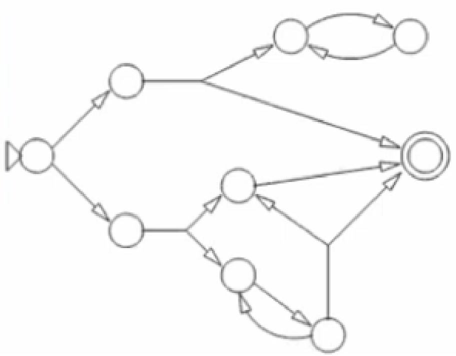
begin and end of backward propagation

start from goal states
they all have weak/strong/strong cyclic solutions

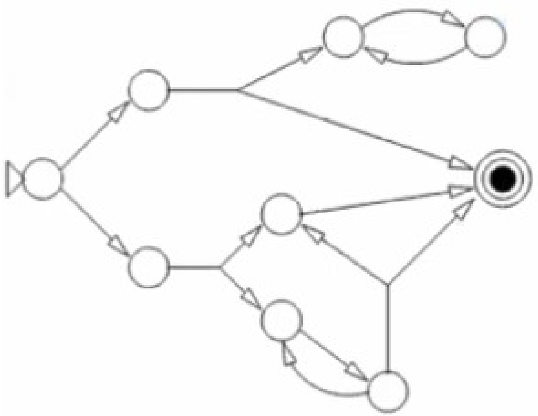
propagate backwards

result: all states that have weak/strong/strong cyclic solutions
check if **initial state** is among them

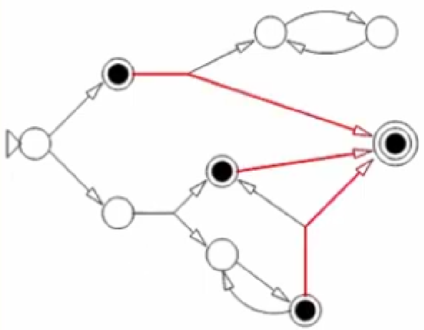
complete example (strong cyclic policy)



goal states: mark weak



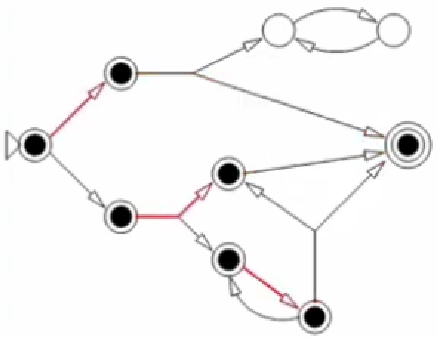
propagate backwards



node marked ⇒ predecessor marked

● = weak policy exists

propagate backwards, again

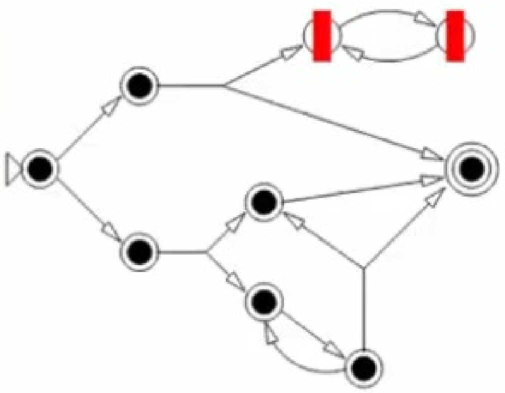


node marked ⇒ predecessor marked

● = weak policy exists

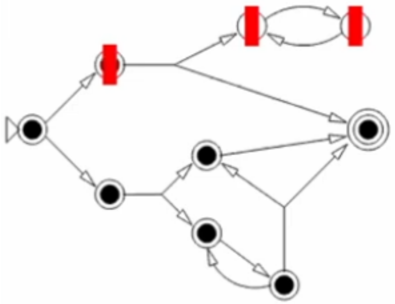
initial state marked: a weak solution exists
is it strong cyclic?

square mark others



node is not marked with a circle
⇒ mark with a square

propagate square marks backwards, effect

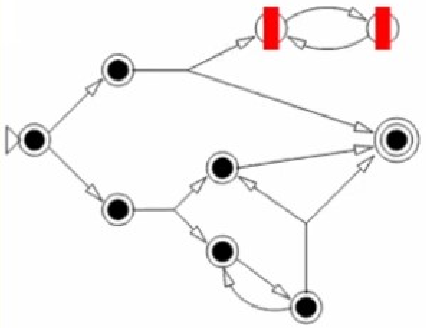


= goal may be unreacheable

another node is marked with a square

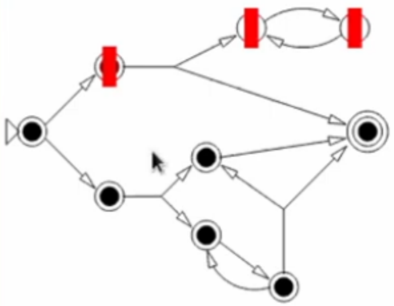
what now?

propagate square marks backwards



for every action at least a square-marked successor
⇒ mark state

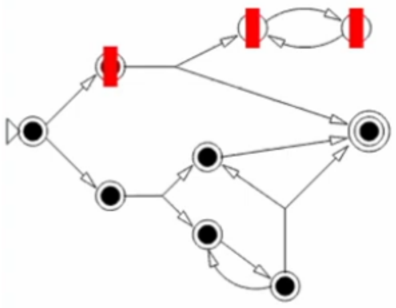
propagate square marks backwards, again?



= goal may be unreacheable

propagate square back to initial state?

choice of action vs. nondeterminism



= goal may be unreacheabl

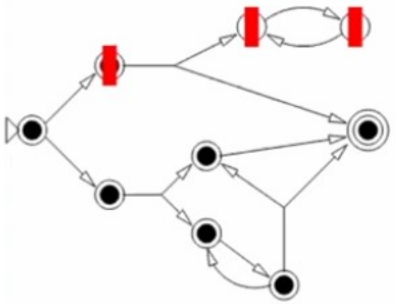
two actions from the intial state
first is not part of a strong cyclic policy
second is

just choose the second action

do not square-mark the initial state

A node is marked with a square only if **every** action leads to **at least** a square-marked successor.

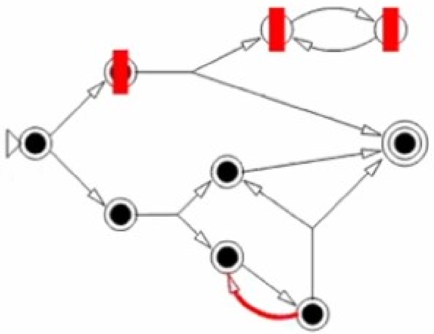
final condition



a strong cyclic policy exists

obtained by going backwards from the goal state
avoiding square-marked states

obtain a policy

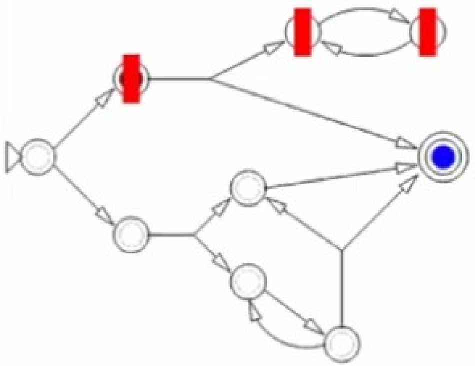


not all actions between marked states are good

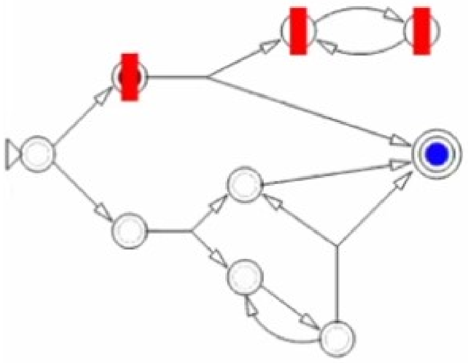
do not include the red arrow (bold) in the policy

how to exclude the likes of it automatically?

find the policy: mark the goal



find the policy: mark the goal

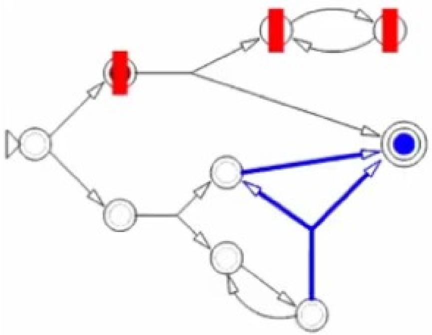


find the policy: proceed backwards

mark actions leading a marked state, but
do not mark actions leading to a square-marked state

why: choose actions leading to states that have a weak policy

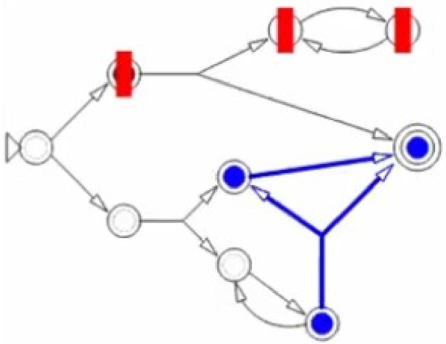
find the policy: proceed backwards



mark actions leading a marked state, but
do not mark actions leading to a square-marked state

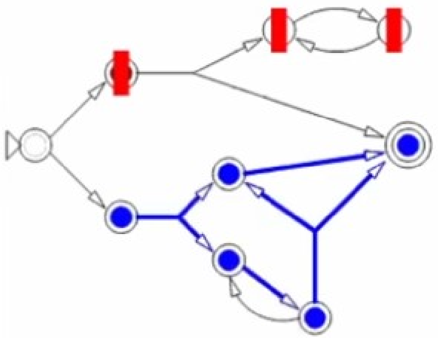
why: choose actions leading to states that have a weak policy

find the policy: mark states

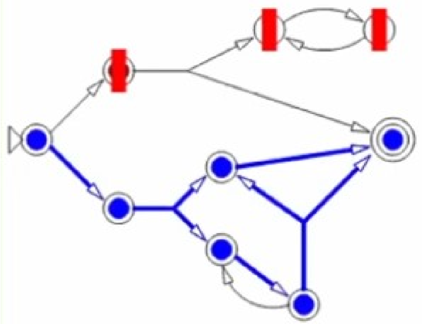


mark states with a marked action starting from them

find the policy: proceed backwards, again



find the policy: proceed backwards once again



strong cyclic policy

summary: proceed by increasing the distance to the goal
avoid states that do not have a weak policy

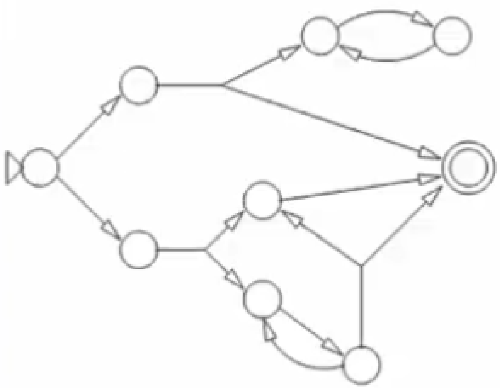
also, in general: choose an action for each state

interleaved method

similar to the previous one

iterate between goal reachability and action removal

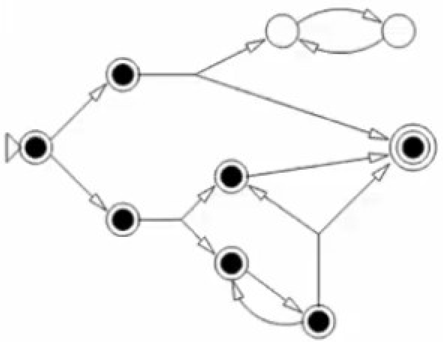
initial condition



as before, mark nodes where the goal is reachable

proceed backwards from the goal

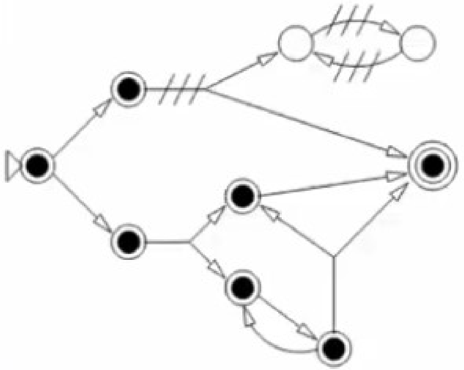
goal reachability



goal is reachable from all marked nodes

now: remove an action if it leads to a non-marked node

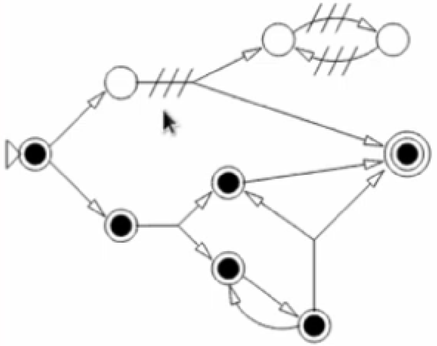
remove actions



some actions are removed

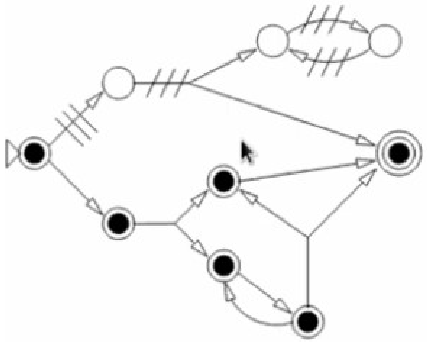
clear the marks and repeat

remove actions, again



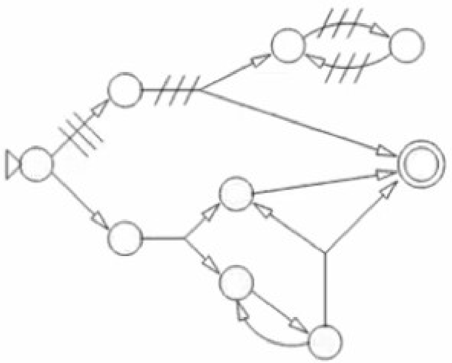
remove an action if it leads to a non-marked state

after the action is removed



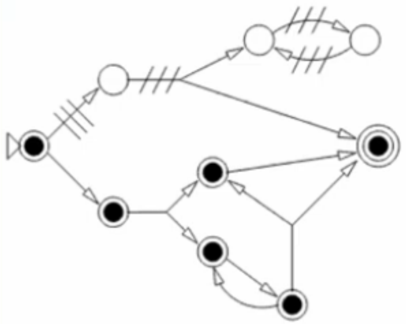
again, clear the marks…

goal reachability, yet another time



from the goal:
mark previous nodes, iteratively

remove actions, yet another time



remove an action if it leads to a non-marked state

this time: nothing removed
⇒ **stop**

interleaving method: summary

interleave:

- backwards from the goal, mark states
- removed actions leading to non-goal states
- repeat if some action removed

initial state has some outgoing action
⇒ strong cyclic policy exists

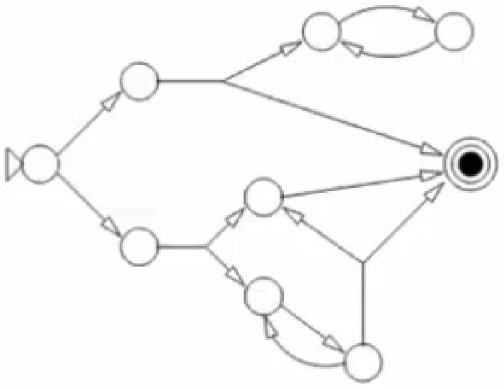
BDD-based representation of states

example: finding a weak policy backwards

initially marked: goal
at each step: new states marked

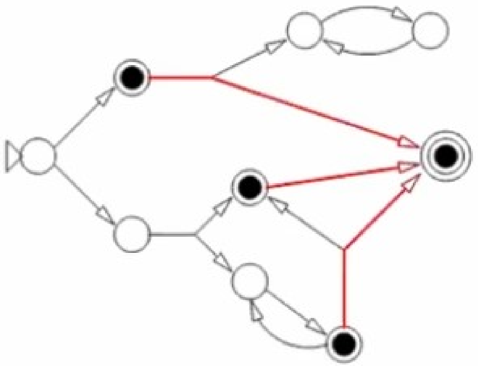
in a different way: a **set of states**
those marked

initial set of states



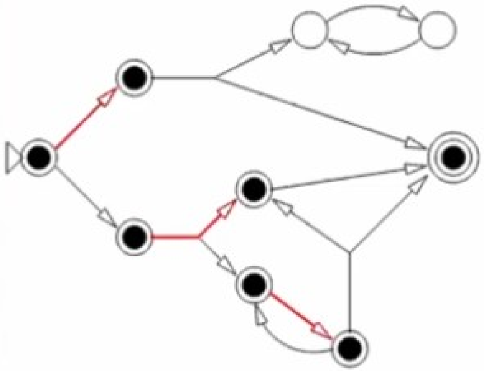
the set only contains the goal states

after the first propagation



the set now contains four states

final condition



the set contains all states but the two upper ones

representation of states

in these figures: a state is a node in a graph

in practice: a possible state of the domain
example:

- initial state=$\{\neg x, y, z\}$
- goal state=$\{x, y, z\}$
- another state=$\{x, \neg y, \neg z\}$
- yet another=$\{\neg x, \neg y, z\}$
- …

an example propositional domain

state variables x, y and z

action a sets x but changes y unpredictably

action b sets y, but changes z unpredictably

initial state $\{\neg x, y, z\}$
goal: $\{x, y, z\}$

meaning of formulae

action a, formula $a \rightarrow x' \wedge (z' \equiv z)$

when executing a, state xyz becomes $x'y'z'$

nondeterministic: $x=0, y=1, z=0$ may become either $x'=1, y'=1, z'=0$ or $x'=1, y'=0, z'=0$

actions as formulae

action a sets x but changes y unpredictably
action b sets y, but changes z unpredictably
initial state $\{\neg x, \neg y, \neg z\}$
goal: $\{x, y, z\}$

action a
    formula $a \rightarrow x' \wedge (z' \equiv z)$
action b
    formula $b \rightarrow y' \wedge (x' \equiv x)$
no parallel execution
    formula $\neg(a \equiv b)$

transition formula

action a
    formula $a \rightarrow x' \wedge (z' \equiv z)$
action b
    formula $b \rightarrow y' \wedge (x' \equiv x)$
no parallel execution
    formula $\neg(a \equiv b)$

transition formula $T = (a \rightarrow x' \wedge (z' \equiv z)) \wedge (b \rightarrow y' \wedge (x' \equiv x)) \wedge \neg(a \equiv b)$

X variables representing the current state
A variables representing the action
X' variables representing the next state

formula is true if X' is a possible successor of X if executing A

goal as a formula

goal: {x,y,z}

formula $G = x \wedge y \wedge z$

formula is true if X is a goal state

predecessors

goal G, variables X
true if X is a goal state
transition T, variables X, A and X'
true if X' is a successor of X when executing A

predecessors of the goal: $\exists A \exists X' . T \wedge (G[X'/X])$
true if one of the successors of X is a goal state

weak solutions, as an evolving formula

1. $F = G$
2. $F' = F[X'/X]$
3. $F'' = \exists A \exists X' . T \wedge F'$
4. if F'' is not equivalent to F, set $F=F''$ and go to 2

iterations

1. $F = G$
2. $F' = F[X'/X]$
3. $F'' = \exists A \exists X' . T \wedge F'$
4. if F'' is not equivalent to F, set $F=F''$ and go to 2

first iteration:

F constraints X to be a goal state
F' is the same, but on X'
F'' is true on X if there exists an action such that one of its successors is a goal state

second iteration: start with the predecessors of the goal
obtain the states that have the goal as a successor of a successor

BDDs

1. $F = G$
2. $F' = F[X'/X]$
3. $F'' = \exists A \exists X' . T \wedge F'$
4. if F'' is not equivalent to F, set $F=F''$ and go to 2

represent all formulae as BDDs

BDDs can be conjoined and existentially quantified

the algorithm, on BDDs

- turn formula T into a BDD
- turn goal formula G into a BDD
- rename the variables of the BDD of G from X to X'
- predecessors: $\exists A \exists X' . T \wedge G$
  compute BDD of $T \wedge G$ given the BDDs of T and G
  then compute the BDD of $\exists A \exists X' . T \wedge G$ given the BDD of $T \wedge G$
- result is a BDD on variables X only

again: rename X to X' in this BDD, conjoin the result with the BDD of T and do $\exists X \exists A$

when the BDD does not change any longer
it represents the set of states for which a weak solution exists

which BDDs? why BDDs?

only keep 2 BDDs:

1. the BDD representing T
2. the BDD representing F:
   (states curently established to have the goal reachable)

discard intermediate ones

a single BDD may represent exponentially many states