
Planning as model checking, (OBDDs)

José Luis Ambite*

[* Based in part on Paolo Traverso's (<http://sra.itc.it/people/traverso/>)
tutorial: <http://prometeo.ing.unibs.it/sschool/slides/traverso/traverso-slides.ps.gz>
Some slides from <http://www-2.cs.cmu.edu/~mmv/planning/handouts/BDDplanning.pdf>]
by Rune Jensen <http://www.itu.dk/people/rmj>

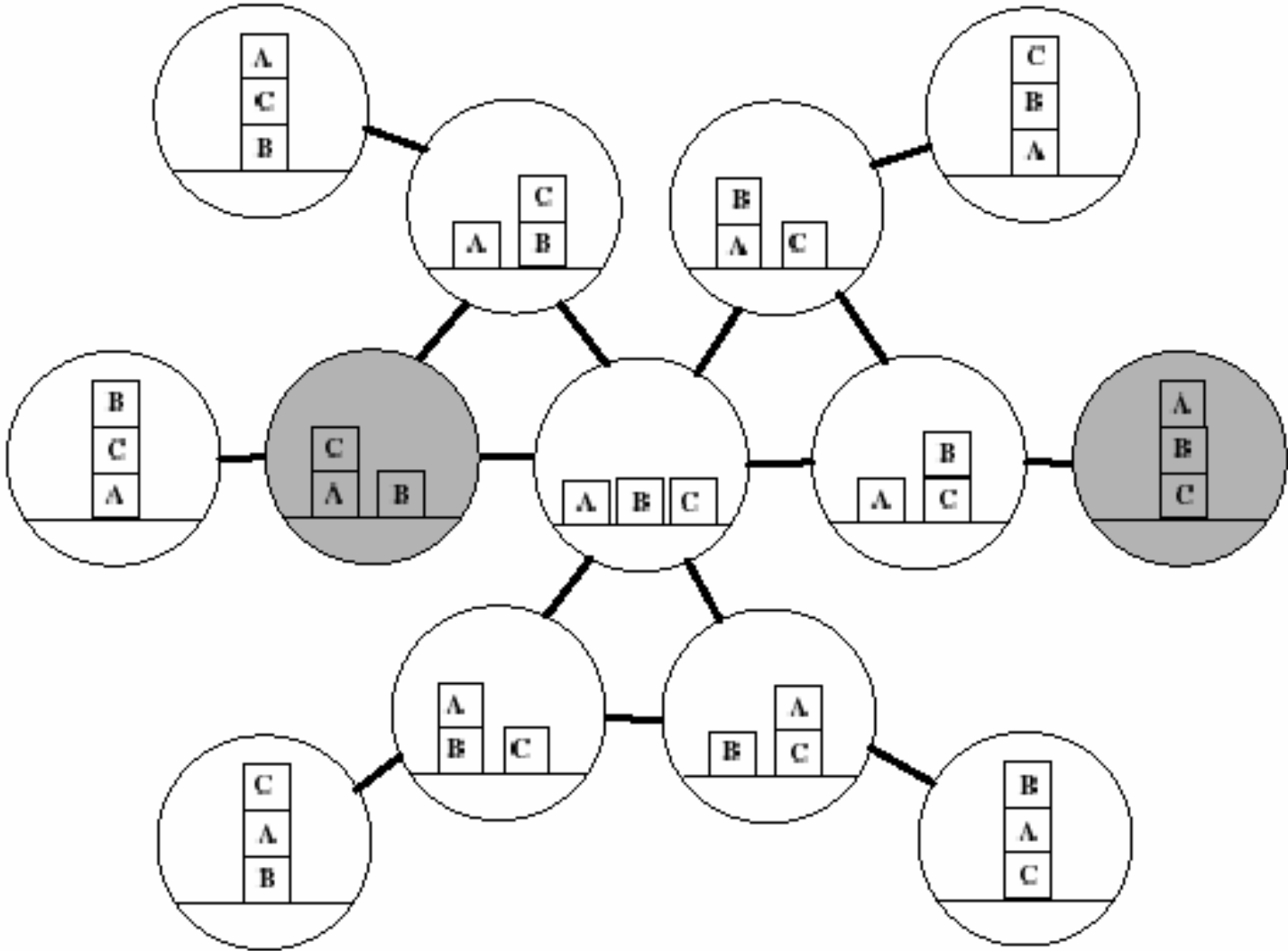
The Model Checking Problem

Determine whether a formula is true in a model

1. A domain of interest is described by a semantic model
2. A desired property of the domain is described by a logical formula
3. Check if the domain satisfy the desired property by checking whether the formula is true in the model

Motivation: Formal verification of dynamic systems

State Space: Blocks World



Classical (Strips-like) Planning: a simple example

action: load

prec: \neg Locked, \neg Loaded

post:

add-list: Loaded

del-list:

action: unload

prec: \neg Locked, Loaded

post:

add-list:

del-list: Loaded

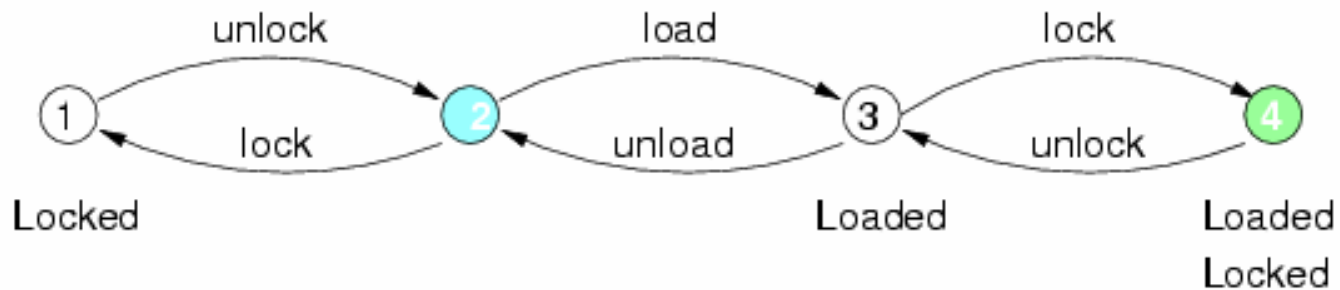
action: lock

prec: \neg Locked

post:

add-list: Locked

del-list:



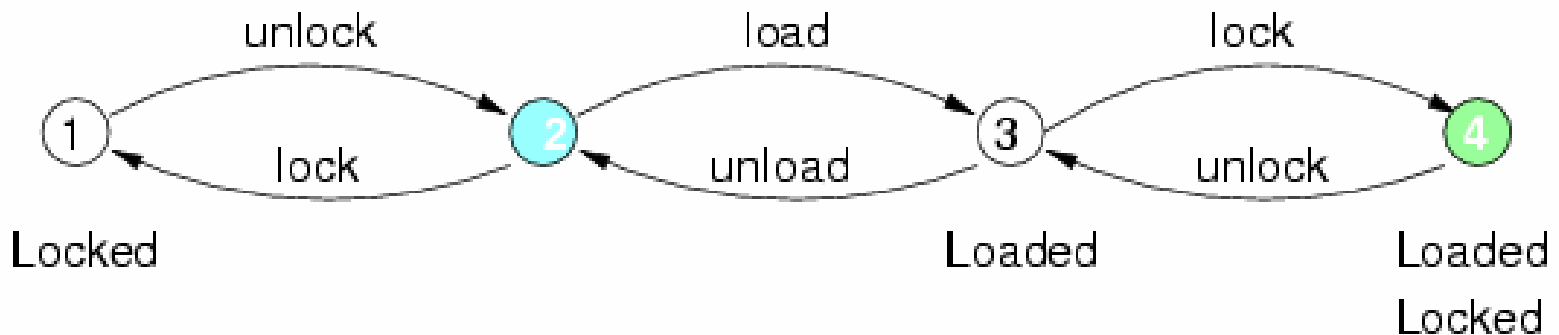
plan = load ; lock

State-Transition Systems: Planning Domains

A planning domain D is a 4-tuple $\langle F, S, A, R \rangle$:

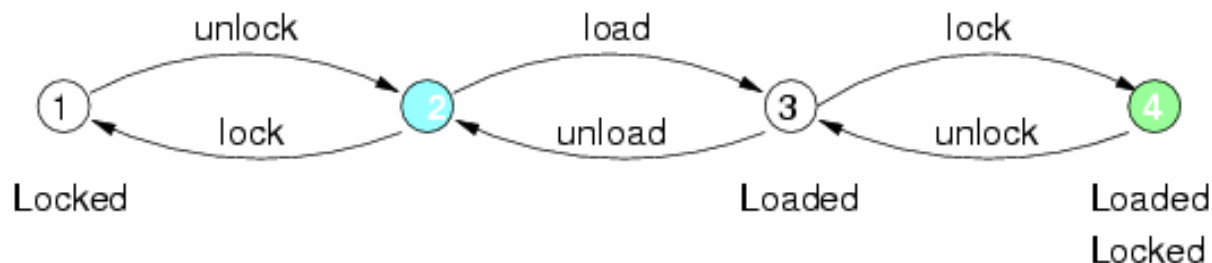
- F is a finite set of Fluents
- $S \subseteq 2^F$ is a finite set of states
- A is a finite set of actions
- $R \subseteq S \times A \times S$ is a transition relation

Action $a \in A$ is executable in $s \in S$ if $\exists s' R(s, a, s')$



State-Transition Systems: (Deterministic) Planning Domain Example:

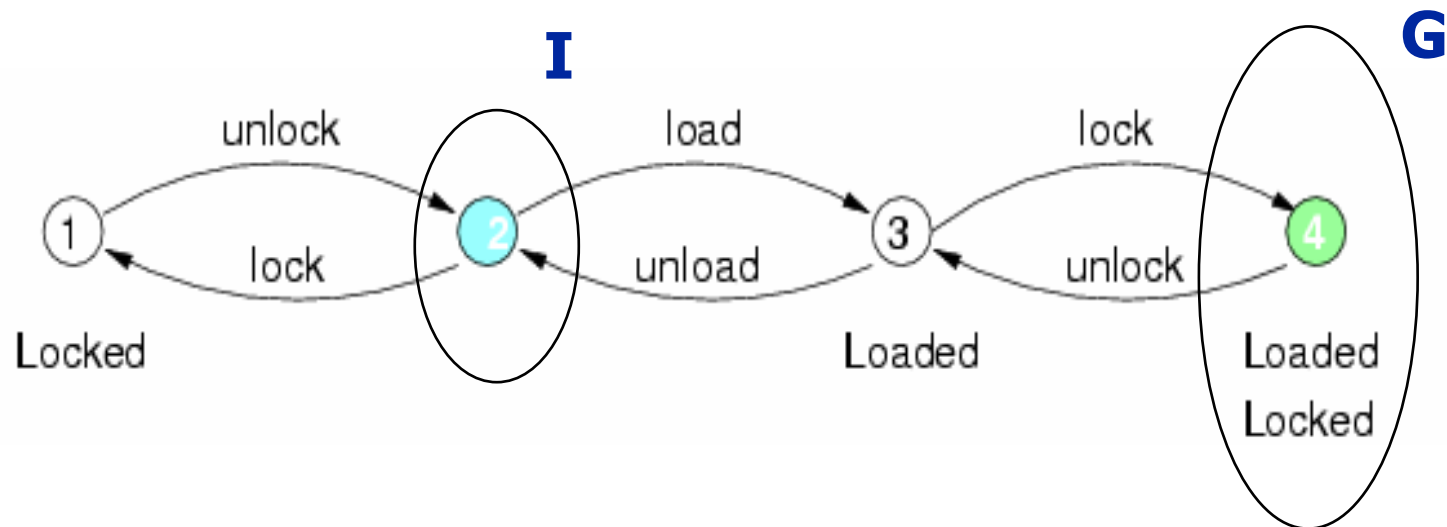
- $F = \{\text{loaded, locked}\}$
- $S = \{(\neg\text{loaded } \text{locked}), (\neg\text{loaded } \neg\text{locked}), (\text{loaded } \neg\text{locked}), (\text{loaded } \text{locked})\}$
- $A = \{\text{lock, unlock, load, unload}\}$
- $R = \{ [(\neg\text{loaded } \text{locked}) \text{ unlock } (\neg\text{loaded } \neg\text{locked})], [(\neg\text{loaded } \neg\text{locked}) \text{ lock } (\neg\text{loaded } \text{locked})], [(\neg\text{loaded } \neg\text{locked}) \text{ load } (\text{loaded } \neg\text{locked})], [(\text{loaded } \neg\text{locked}) \text{ unload } (\neg\text{loaded } \neg\text{locked})], [(\text{loaded } \neg\text{locked}) \text{ lock } (\text{loaded } \text{locked})], [(\text{loaded } \text{locked}) \text{ unlock } (\text{loaded } \neg\text{locked})] \}$



State-Transition Systems: Planning Problem

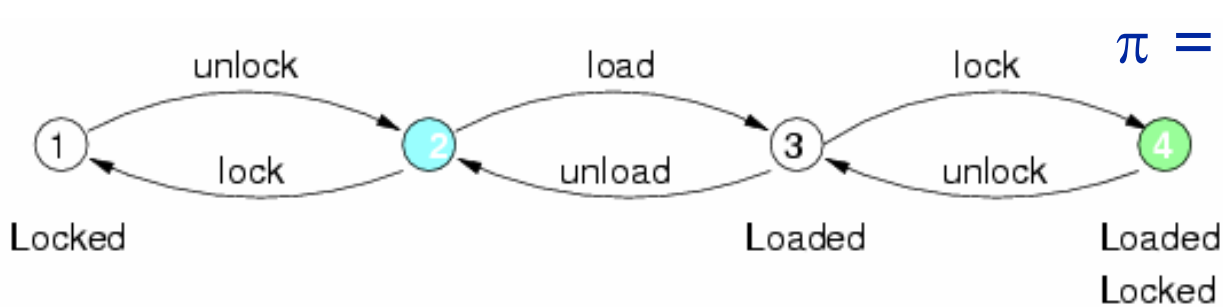
A planning problem P for a planning Domain $D = \langle F S A R \rangle$ is a 3-tuple $\langle D, I, G \rangle$:

- $I \subseteq S$ is the set of initial states
- $G \subseteq S$ is the set of goal states



State-Transition Systems: Plan

- A plan π for a planning problem $P = \langle I, G, D \rangle$ in a planning domain $D = \langle F, S, A, R \rangle$ is a set of state-action pairs:
 - $\{(s, a) : s \in S, a \in A, a \text{ executable in } s\}$
 - at least one (s, a) with $s \in I$
- Goal achieving plan (informally):
 - for each state-action pairs (s, a) , either a leads from s to the goal, $R(s, a) \in G$, or a leads from s to a state s' such that $(s', a') \in \pi$ and a' leads from s' to the goal $R(s', a') \in G$, and so on.



$\pi = \{(2, \text{load}), (3, \text{lock})\}$

Planning Algorithm (Regression)

```
1. function PLAN( $P$ )
2.    $CurrentStates := \emptyset$ ;
3.    $NextStates := G$ ;
4.    $Plan := \emptyset$ ;
5.   while ( $NextStates \neq CurrentStates$ ) do
6.     if  $I \subseteq NextStates$ 
7.       then return  $Plan$ ;
8.      $OneStepPlan := ONESTEPPLAN(NextStates, D)$ ;
9.      $Plan := Plan \cup PRUNESTATES(OneStepPlan, NextStates)$ ;
10.     $CurrentStates := NextStates$ ;
11.     $NextStates := NextStates \cup PROJECTACTIONS(OneStepPlan)$ ;
12.  return  $Fail$ ;
```

Backward image
(regress set of states)

Remove Visited

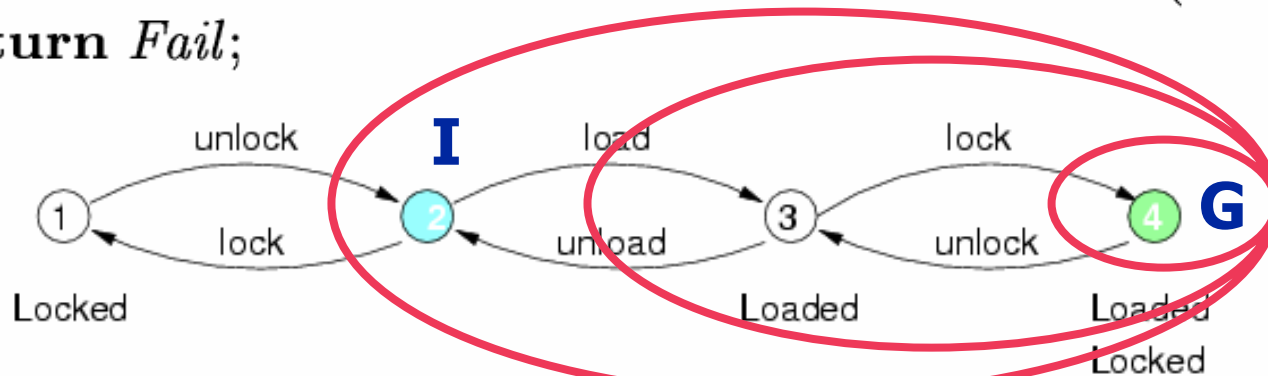
$ONESTEPPLAN(States, D) =$
 $\{\langle s, a \rangle : s \in S, a \in A, \exists s'. (s' \in States \wedge s' = R(s, a))\}$

$PRUNESTATES(\pi, States) = \{\langle s, a \rangle \in \pi : s \notin States\}$

$PROJECTACTIONS(\pi) = \{s : \langle s, a \rangle \in \pi\}$

Planning Algorithm (Regression)

1. **function** PLAN(P)
2. $CurrentStates := \emptyset$;
3. $NextStates := G$;
4. $Plan := \emptyset$;
5. **while** ($NextStates \neq CurrentStates$) **do**
6. **if** $I \subseteq NextStates$
7. **then return** $Plan$;
8. $OneStepPlan := ONESTEPPLAN(NextStates, D)$;
9. $Plan := Plan \cup PRUNESTATES(OneStepPlan, NextStates)$;
10. $CurrentStates := NextStates$;
11. $NextStates := NextStates \cup PROJECTACTIONS(OneStepPlan)$;
12. **return** $Fail$;



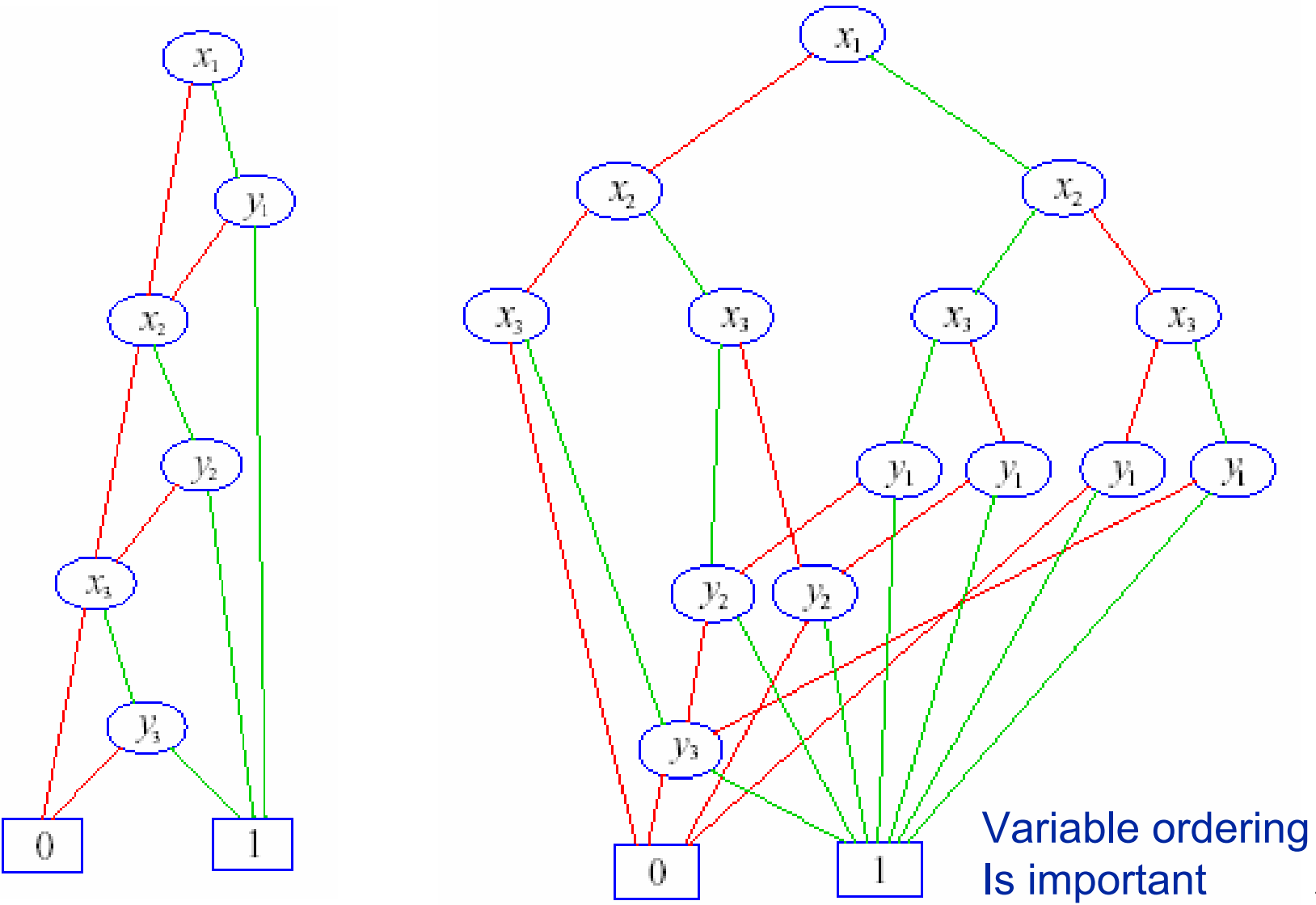
Backward image
(regress set of states)

Remove Visited

Planning via Symbolic Model Checking

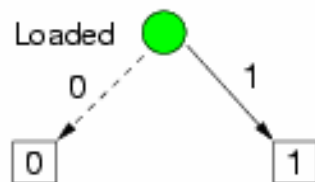
- Problem: Realistic planning domains often have large state spaces
- Idea: exploit the work on symbolic model checking based on Ordered Binary Decision Diagrams (OBDD's)
- OBDD's:
 - Canonical form for propositional formulas
 - Efficient!
 - Polynomial boolean operations: $O(\varphi_1 \otimes \varphi_2) = O(|\varphi_1| |\varphi_2|)$
 - Constant time equality

OBDD example: $(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee (x_3 \wedge y_3)$

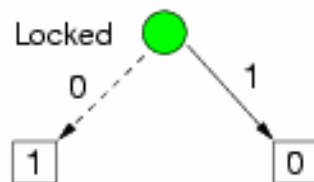


Planning via Symbolic Model Checking with OBDDs

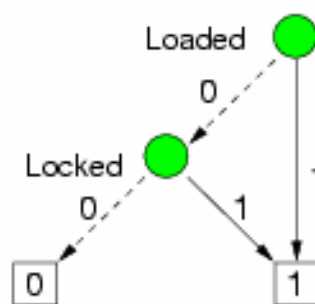
- OBDD's are a compact representation of the assignments satisfying (and falsifying) a given boolean formula.
- Operations over sets of states (e.g. union, intersection) are implemented as boolean operations (e.g. conjunction, disjunction) over OBDDs.



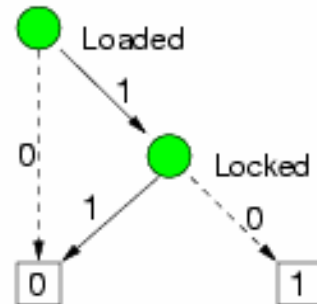
Loaded



Not Locked



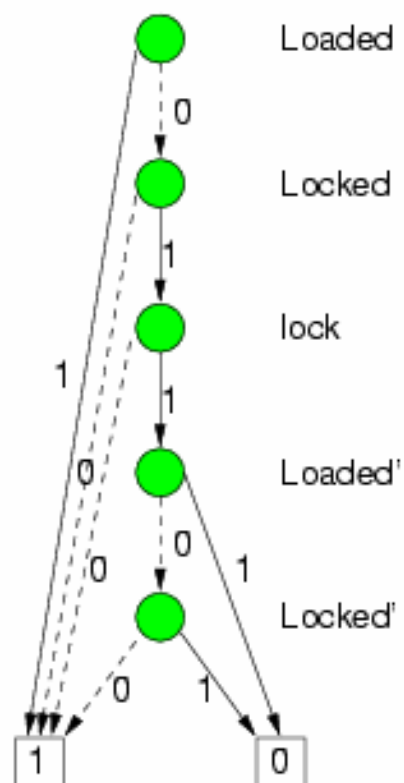
Loaded or Locked



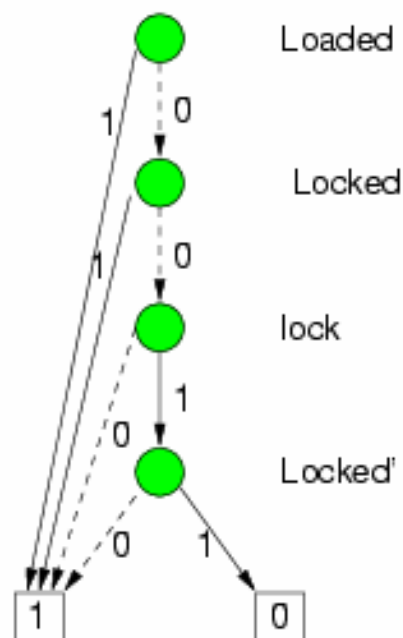
Loaded and not Locked

Planning via Symbolic Model Checking with OBDDs

Transitions are represented by OBDD's in the variables
 \underline{x} (current state), \underline{a} (actions), \underline{x}' (next states)



$$(\neg \text{Loaded} \wedge \text{Locked} \wedge \text{lock}) \supset \neg \text{Loaded}' \wedge \neg \text{Locked}'$$



$$(\neg \text{Loaded} \wedge \neg \text{Locked} \wedge \text{lock}) \supset \neg \text{Locked}'$$

Planning via Model Checking

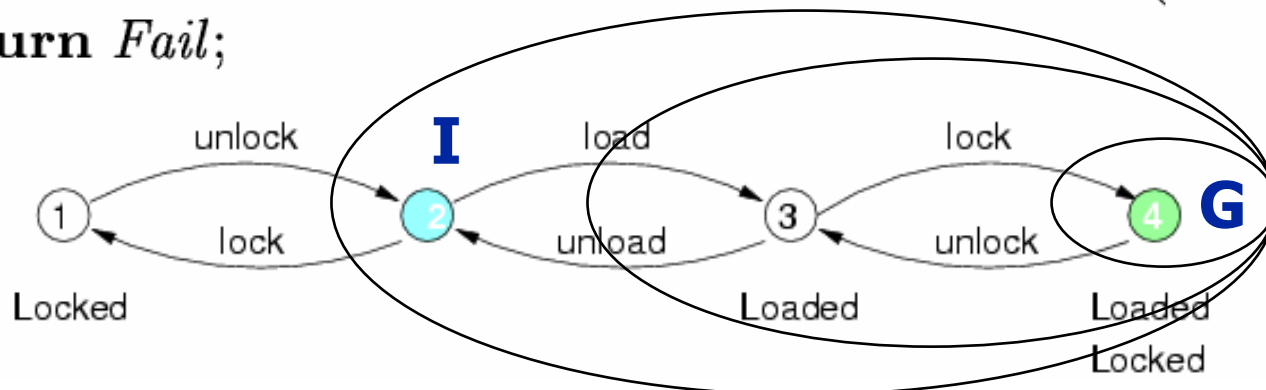
Symbolic Representation

- Action represented by assigning true to the corresponding variable
- Transition $t = \langle s \ a \ s' \rangle$ encoded as
$$\xi(t) = \xi(s) \wedge \xi(a) \wedge \xi(s')$$
- Transition relation T encoded as disjunction of all the transitions

$$\xi(T) = \bigvee_{t \in T} \xi(t)$$

Planning Algorithm (regression)

1. **function** PLAN(P)
2. $CurrentStates := \emptyset$;
3. $NextStates := G$;
4. $Plan := \emptyset$;
5. **while** ($NextStates \neq CurrentStates$) **do**
6. **if** $I \subseteq NextStates$
7. **then return** $Plan$;
8. $OneStepPlan := ONESTEPPLAN(NextStates, D)$;
9. $Plan := Plan \cup PRUNESTATES(OneStepPlan, NextStates)$;
10. $CurrentStates := NextStates$;
11. $NextStates := NextStates \cup PROJECTACTIONS(OneStepPlan)$;
12. **return** $Fail$;



Planning Algorithm (regression)

- OneStepPlan(S) in the regression algorithm is the backward image of the set of states S.
- Can be computed as the QBF formula:

$$\exists \underline{x}' (\text{States}(\underline{x}') \wedge R(\underline{x}, a, \underline{x}'))$$

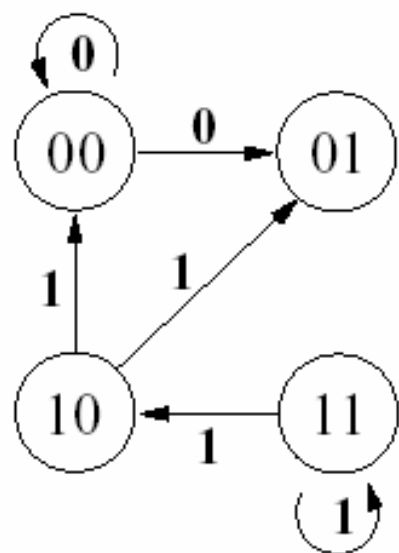
- Quantified Boolean Formula (QBF):

$$\exists x \phi(x, y) = \phi(0, y) \vee \phi(1, y)$$

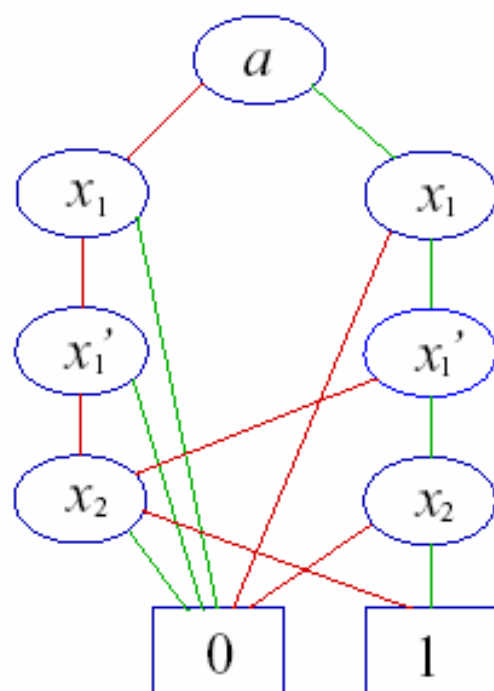
$$\forall x \phi(x, y) = \phi(0, y) \wedge \phi(1, y)$$

Non-deterministic Domains represented by OBDDs

FSM

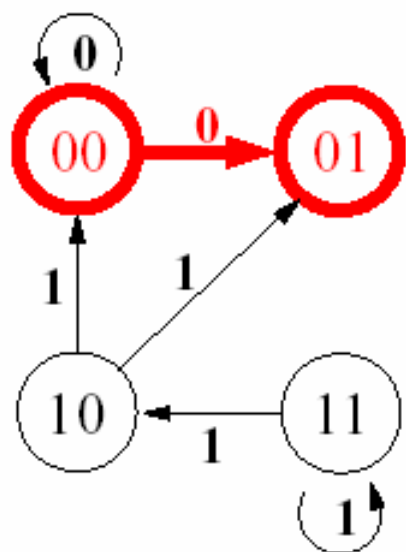


$T(a, x_1, x'_1, x_2, x'_2)$

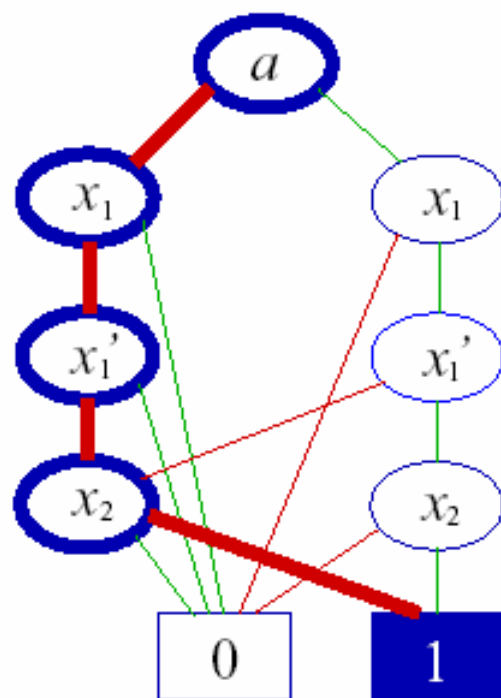


Non-deterministic Domains represented by OBDDs

FSM



$T(a, x_1, x'_1, x_2, x'_2)$

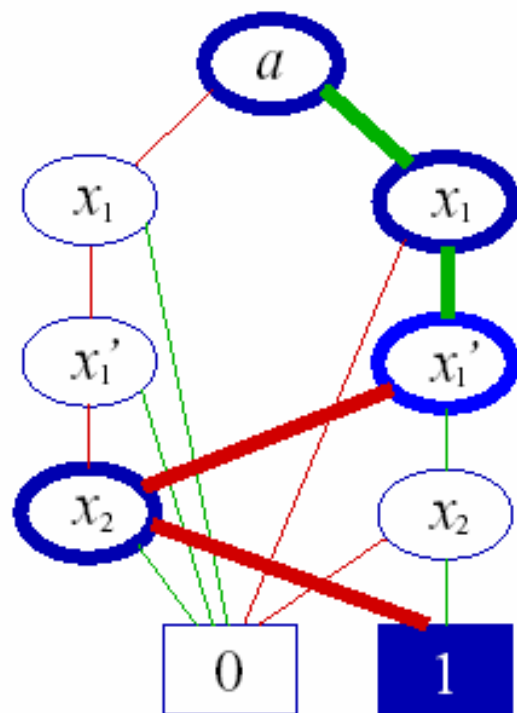
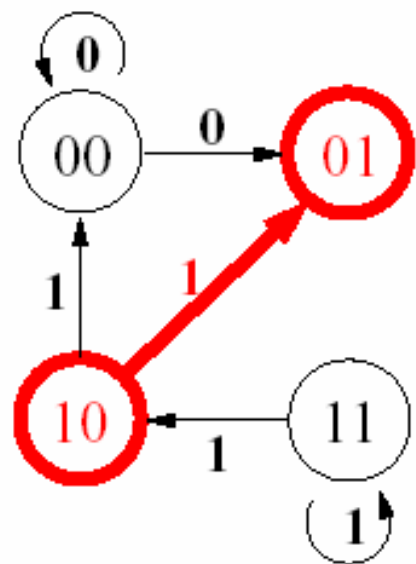


$$T(0,0,0,0,1) = 1$$

Non-deterministic Domains represented by OBDDs

FSM

$T(a, x_1, x'_1, x_2, x'_2)$



$$T(1, 1, 0, 0, 1) = 1$$

OBDD-based Planning

Action 0

Pre: $\neg x_1 \wedge \neg x_2$

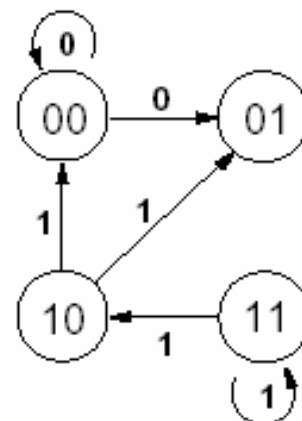
Eff: $\neg x'_1$

Action 1

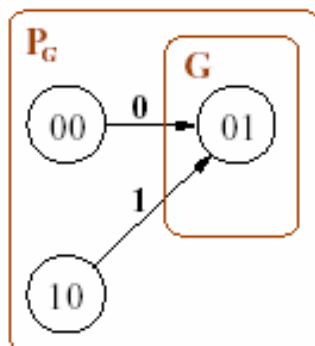
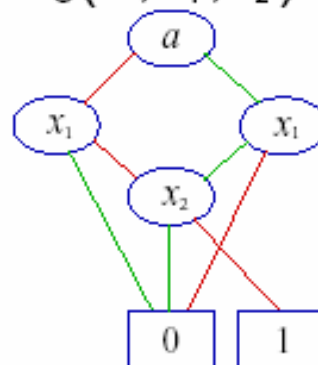
Pre: x_1

Eff: $(x_1 \wedge \neg x_2 \rightarrow \neg x'_1) \wedge$
 $(x_1 \wedge x_2 \rightarrow x'_1)$

FSM



$P_G(a, x_1, x_2)$



$$P_V(a, x_1, x_2) = \exists x'_1, x'_2. T(a, x_1, x'_1, x_2, x'_2) \wedge V(x'_1, x'_2)$$

$$P_G(a, x_1, x_2) = \exists x'_1, x'_2. T(a, x_1, x'_1, x_2, x'_2) \wedge (\neg x'_1 \wedge x'_2)$$

Planning as model checking

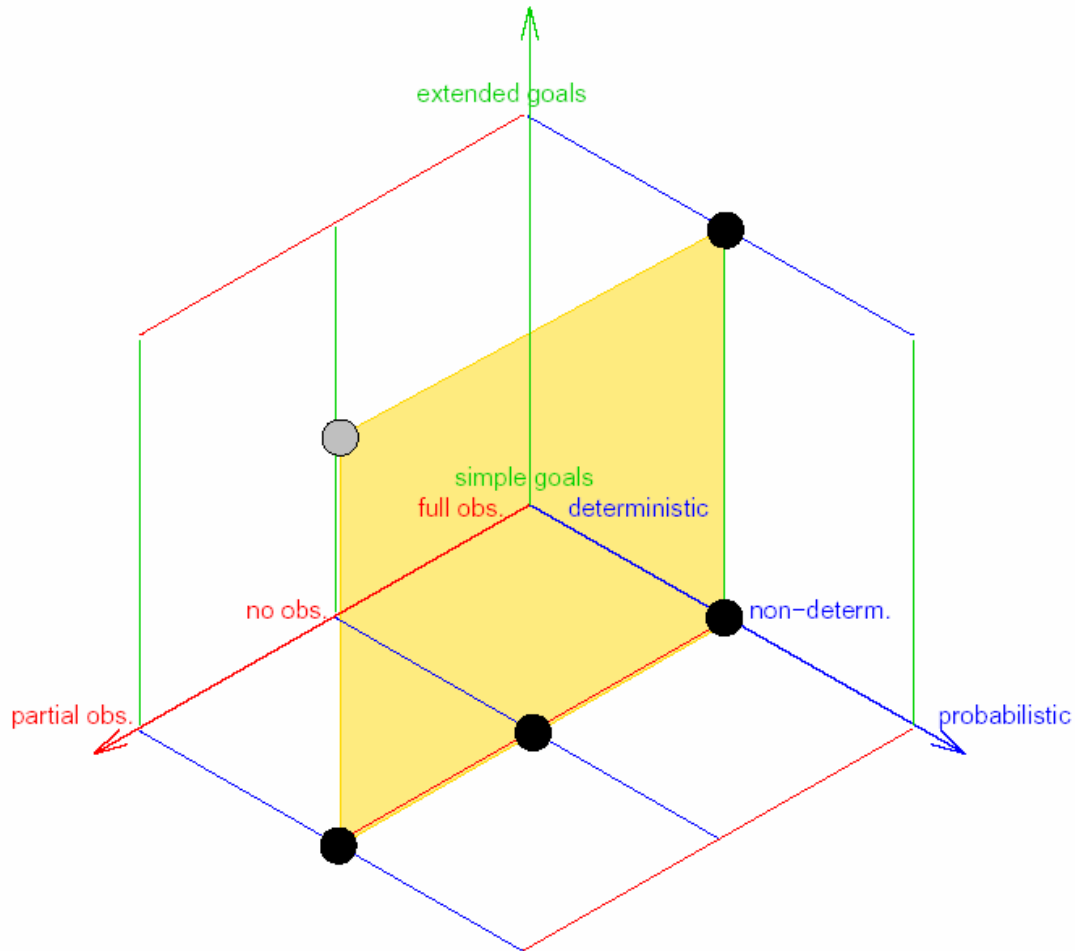
The “*Planning as Model Checking*” approach defines a general framework for dealing with:

- non-deterministic domains...
- extended goals...
- partial observability...
- ...and their combination

In “*Planning as Symbolic Model Checking*” BDD-based symbolic techniques are used for:

- a compact representation of domains and plans
- an efficient search in the domain in the planning algorithm

Planning as Model Checking



Goal language: CTL

We propose CTL as the language for expressing extended goals.

The “Reach *dep* and avoid *lab*” example:

- Do reach *dep* and do avoid *lab* → $AF\ dep \wedge AG\ \neg lab$
- Do reach *dep* and try avoid *lab* → $AF\ dep \wedge EG\ \neg lab$
- Try reach *dep* and do avoid *lab* → $EF\ dep \wedge AG\ \neg lab$

“Reachability” goals:

- “weak” planning → EF
- “strong” planning → AF
- “strong cyclic” planning → $AG\ EF$

Computation Tree Logic (CTL)

Given a finite set \mathcal{P} of atomic propositions, CTL formulas are inductively defined as follows:

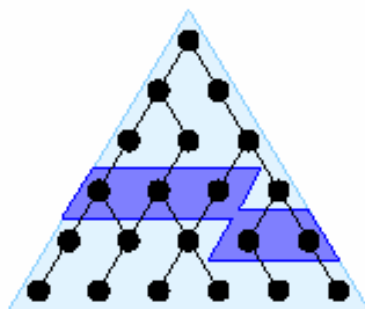
1. Every atomic proposition $p \in \mathcal{P}$ is a CTL formula;
2. If p and q are CTL formulas, then so are
 - (a) $\neg p$, $p \vee q$,
 - (b) $\mathbf{AX}p$, $\mathbf{EX}p$,
 - (c) $\mathbf{A}(p\mathbf{U}q)$, and $\mathbf{E}(p\mathbf{U}q)$

-
- $\mathbf{AX}p$ ($\mathbf{EX}p$): p holds in every (in some) “next” state.
 - $\mathbf{A}(p\mathbf{U}q)$ ($\mathbf{E}(p\mathbf{U}q)$): for every path (for some path) “ p holds until q holds”
 - $\mathbf{AF}p$ ($\mathbf{EF}p$) abbreviation of $\mathbf{A}(\top\mathbf{U}p)$ ($\mathbf{E}(\top\mathbf{U}p)$)
 - $\mathbf{EG}p$ ($\mathbf{AG}p$) abbreviation of $\neg\mathbf{AF}\neg p$ ($\neg\mathbf{EF}\neg p$)

CTL

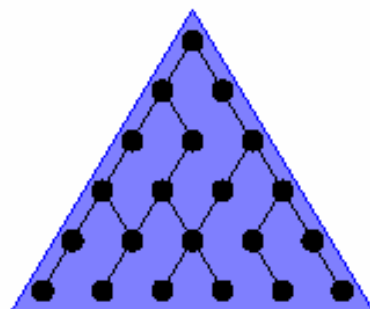


finally P



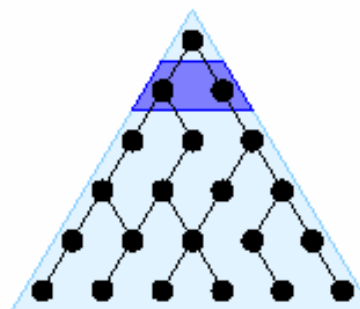
$AF P$

globally P



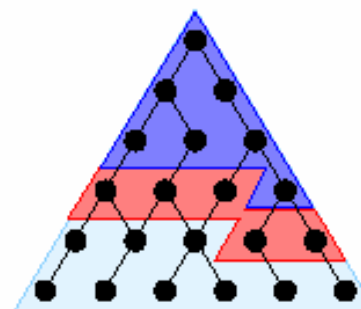
$AG P$

next P

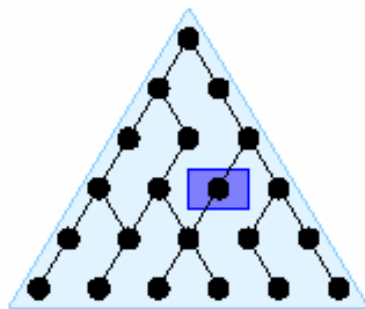


$AX P$

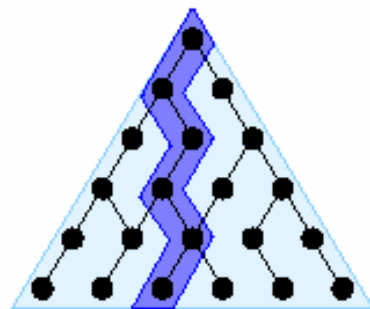
P until q



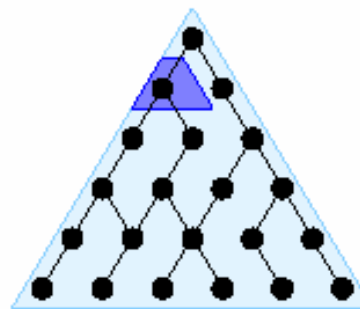
$A [P U q]$



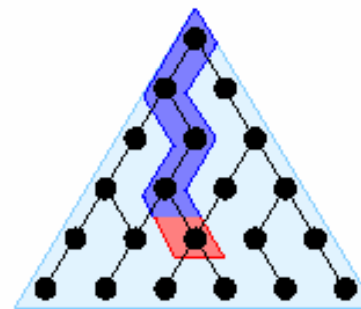
$EF P$



$EG P$

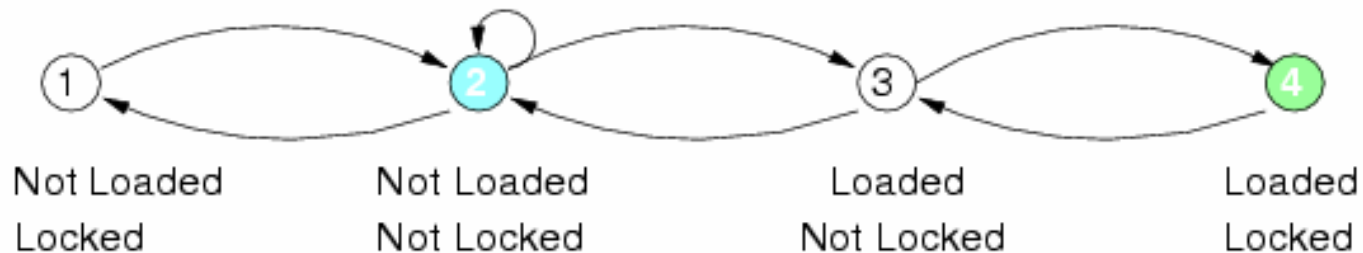


$EX P$



$E [P U q]$

Computation Tree Logic (CTL): Examples



EF *Loaded* holds in state 2

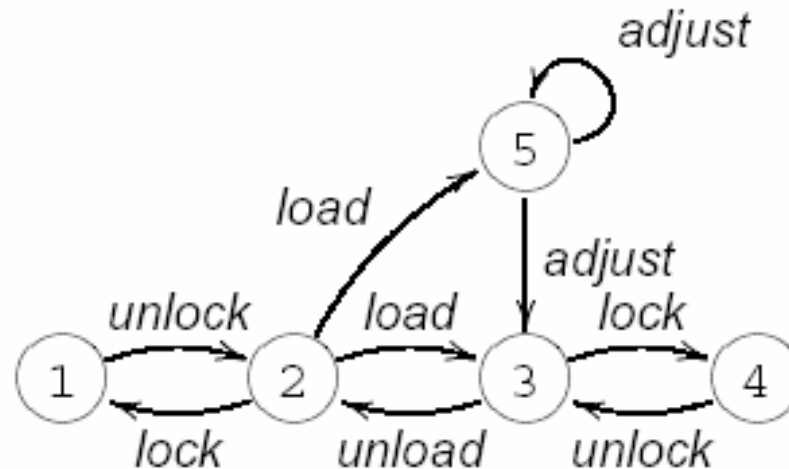
AF *Loaded* does not hold in state 2

EG *Loaded* holds in state 2

AG *Loaded* does not hold in state 2

Weak, Strong and Strong Cyclic Reachability Goals

1. **Weak Solutions:** plans that may achieve the goal
2. **Strong Solutions:** plans that are guaranteed to achieve the goal
3. **Strong Cyclic solutions:** iterative trial-and-error strategies whose executions always have a possibility of terminating and, when they do, they are guaranteed to achieve the goal.



Weak Solutions

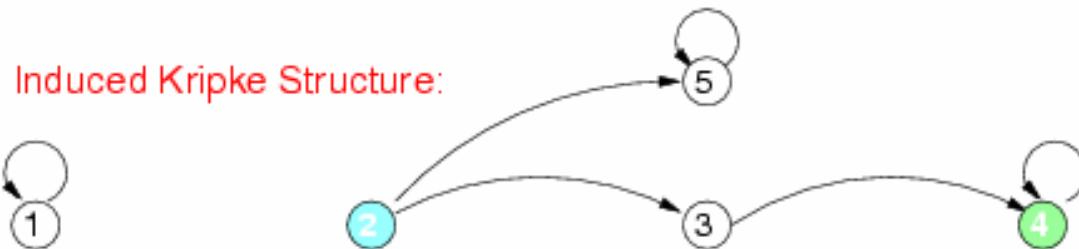
Given: A planning domain D and problem P

A plan π and the Induced Kripke Structure K

The plan π is a weak solution to the planning problem P if
for all $s \in I$, $K, s \models \text{EF } G$.



plan = { <2,load> , <3,lock> }

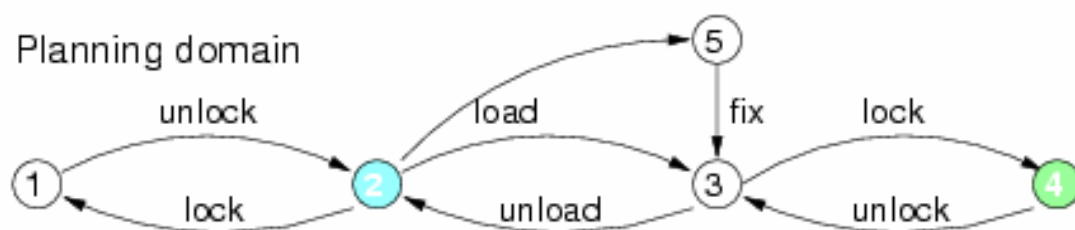


Strong Solutions

Given: A planning domain D and problem P

A plan π and the Induced Kripke Structure K

The plan π is a strong solution to the planning problem P if
for all $s \in I$, $K, s \models \text{AF } G$.



plan = { <2,load> , <3,lock> , <5,fix> }

Induced Kripke Structure:

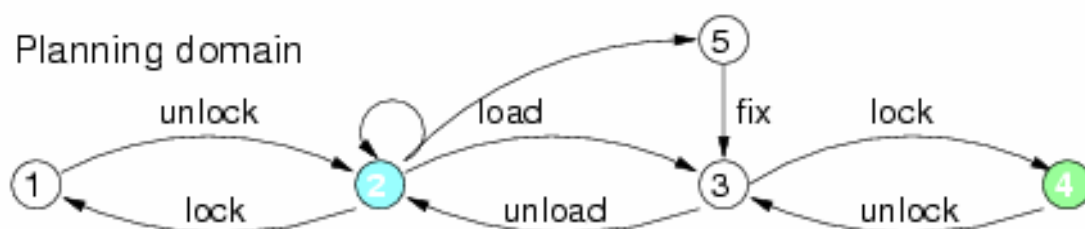


Strong Cyclic Solutions

Given: A planning domain D and problem P

A plan π and the Induced Kripke Structure K

The plan π is a strong cyclic solution to the planning problem P if
for all $s \in I$, $K, s \models \text{AGEF } G$.



plan = { <2,load> , <3,lock> , <5,fix> }

Induced Kripke Structure:

