Planning as Model Checking

Paolo Traverso

IRST, Istituto per la Ricerca Scientifica e Tecnologica,
38050 Povo, Trento, Italy

Index

Model Checking

The Model Checking problem is
the problem of determining whether a formula is true in a model.

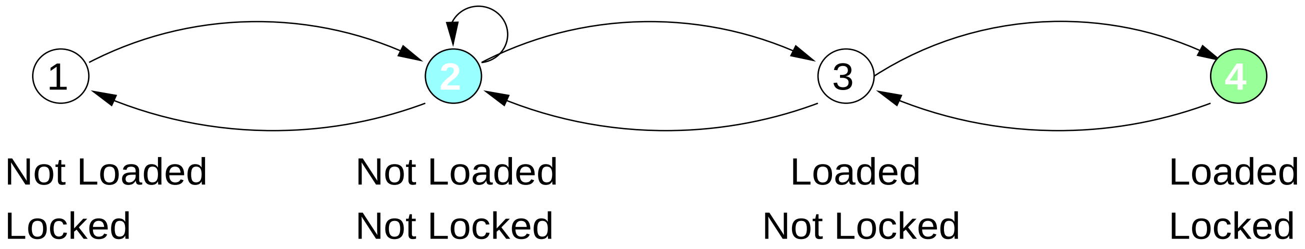
1. A domain of interest is described by a semantic model.
2. A desired property of the domain is described by a logical formula.
3. The fact that a domain satisfies a desired property is determined by checking whether the formula is true in the model.

Model Checking: Domains as Kripke Structures

A Kripke Structure K is a 4-tuple $\langle W, W_0, T, L \rangle$, where

1. W is a finite set of $states$.
2. $W_0 \subseteq W$ is a set of initial states.
3. $T \subseteq W \times W$ is a binary relation on W, the $transition\ relation$, which gives the possible transitions between states. We require T to be total, i.e., for each state $w \in W$ there exists a state $w' \in W$ such that $(w, w') \in T$.
4. $L : W \mapsto 2^{\mathcal{P}}$ is a $labeling\ function$, where $\mathcal{P}$ is a set of atomic propositions. L assigns to each state the set of atomic propositions true in that state.

Domains as Kripke Structures: An Example



1. $W = \{1, 2, 3, 4\}$
2. $W_0 = \{2\}$
3. $T = \{(1, 2), (2, 1), (2, 2), (2, 3), (3, 2), (3, 4), (4, 3)\}$
4. $L(1) = \{Locked\}$, $L(2) = \emptyset$, $L(3) = \{Loaded\}$, $L(4) = \{Loaded, Locked\}$.

Note: *infinite* evolutions of the domain encoded as *paths* $w_0 w_1 w_2 \ldots$ such that, for each i, $(w_i, w_{i+1}) \in T$.

Model Checking: Properties as CTL formulas

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 $\mathbf{E}(p\mathbf{U}q)$)

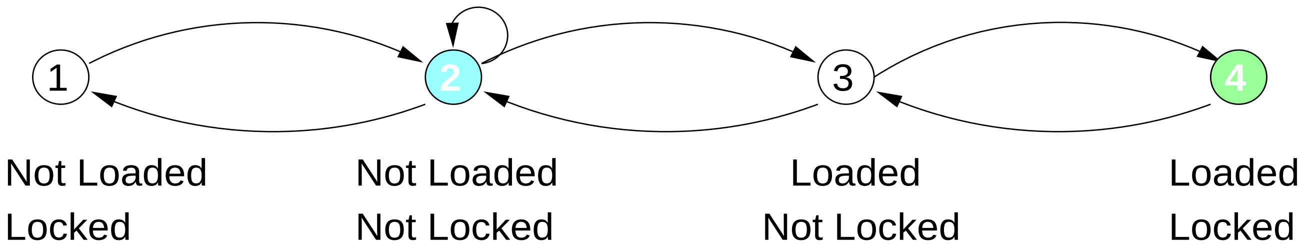
– $\mathbf{AX}p$ ($\mathbf{EX}p$): p holds in every (in some) "next" state.

– $\mathbf{A}(p\mathbf{U}q)$ ($\mathbf{E}(p\mathbf{U}q)$): 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 $\neg\mathbf{AF}\neg p$ ($\neg\mathbf{EF}\neg p$)

Properties as CTL formulas: An Example



$\mathbf{EF}\,Loaded$ holds in state 2

$\mathbf{AF}\,Loaded$ does not hold in state 2

$\mathbf{EG}\,Loaded$ holds in state 2

$\mathbf{AG}\,Loaded$ does not hold in state 2

The Model Checking Problem

CTL semantics (Kripke Structure K, CTL formula p):

- $K, w \models p$ iff $p \in L(w)$, $p \in \mathcal{P}$
- $K, w \models \neg p$ iff $K, w \not\models p$
- $K, w \models p \vee q$ iff $K, w \models p$ or $K, w \models q$
- $K, w \models \mathbf{AX}p$ iff for all paths $\pi = w_0 w_1 \ldots$, with $w = w_0$, $K, w_1 \models p$
- $K, w \models \mathbf{EX}p$ iff there exists a path $\pi = w_0 w_1 \ldots$, with $w = w_0$, such that $K, w_1 \models p$
- $K, w \models \mathbf{A}(p\mathbf{U}q)$ iff for all paths $\pi = w_0 w_1 \ldots$, with $w = w_0$, there exists $i \geq 0$ such that $K, w_i \models q$ and, for all $0 \leq j < i$, $K, w_j \models p$
- $K, w \models \mathbf{E}(p\mathbf{U}q)$ iff there exists a path $\pi = w_0 w_1 \ldots$, with $w = w_0$, and $i \geq 0$ such that $K, w_i \models q$ and, for all $0 \leq j < i$, $K, w_j \models p$

Truth: p is true in K ($K \models p$) if $K, w \models p$ for each $w \in W_0$.

The Model Checking Problem for a CTL formula p and a Kripke Structure K is the problem of determining whether p is true in K

Algorithms for Model Checking

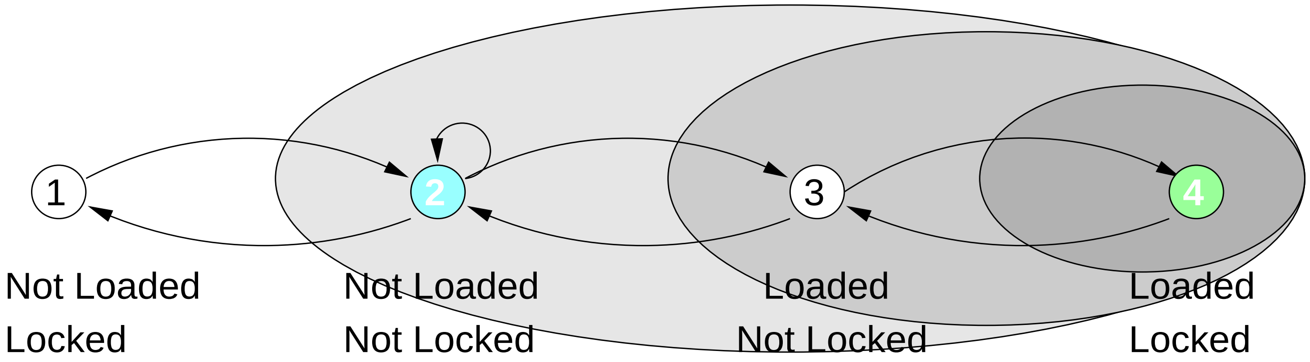
Algorithms for model checking exploit the structure of CTL formulas.

For instance:

- ▷ An atomic formula p is model checked by verifying that $p \in L(s)$ for all $s \in W_0$.
- ▷ Model checking $\mathbf{AX}p$ ($\mathbf{EX}p$) is performed by model checking p in all states (in some state) s' such that $(s, s') \in T$, for each $s \in W_0$.
- ▷ $\mathbf{A}(p\mathbf{U}q)$ or $\mathbf{E}(p\mathbf{U}q)$, can be model checked by exploiting the fact that

$$\begin{aligned} p\mathbf{U}q = \quad & q \vee \\ & (p \wedge \mathbf{X}q) \vee \\ & (p \wedge \mathbf{X}p \wedge \mathbf{XX}q) \vee \\ & \ldots \end{aligned}$$

An example: Model Checking $\mathbf{EF}p$



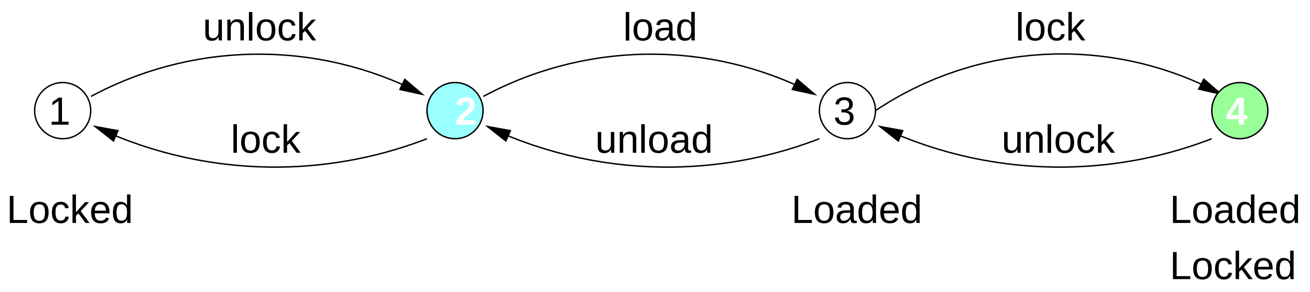
function $\text{MC}\textsc{heck}\text{EF}(p,K)$
  $CurrentStates := \emptyset$;
  $NextStates := \textsc{States}(p,K)$;
  while $NextStates \neq CurrentStates$ **do**
    if $(W_0 \subseteq NextStates)$
      then return $True$;
    $CurrentStates := NextStates$;
    $NextStates := NextStates \cup \textsc{OneStepMCheck}(NextStates,K)$;
  endwhile
    return $False$;

$$\textsc{OneStepMCheck}(States, K) = \{s \in W : \exists s'. \ (s' \in States \ \wedge \ T(s, s'))\}$$

Planning as Model Checking

Classical (Strips-like) Planning: a simple example

action: load	action: unload	action: lock
prec: ¬ Locked, ¬ Loaded	prec: ¬ Locked, Loaded	prec: ¬ Locked
post:	post:	post:
add-list: Loaded	add-list:	add-list: Locked
del-list:	del-list: Loaded	del-list:

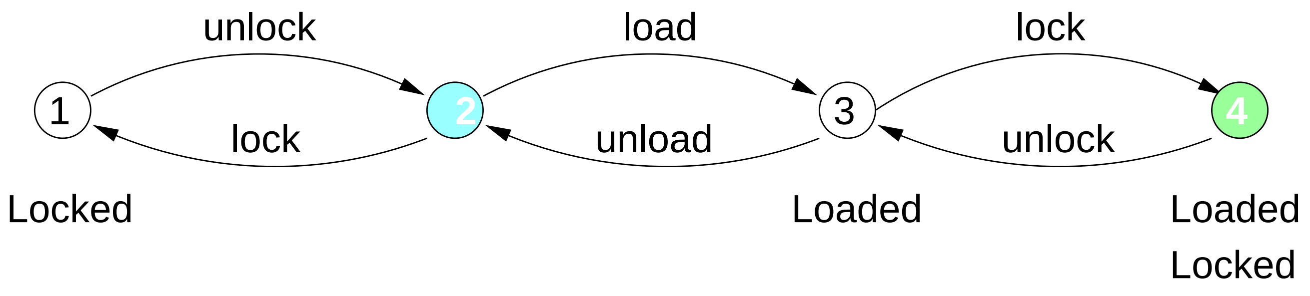


plan = load ; lock

Classical Planning

Classical Planning Assumptions:

- The initial situation is completely specified
- Actions have deterministic effects
- Plans are sequences of actions

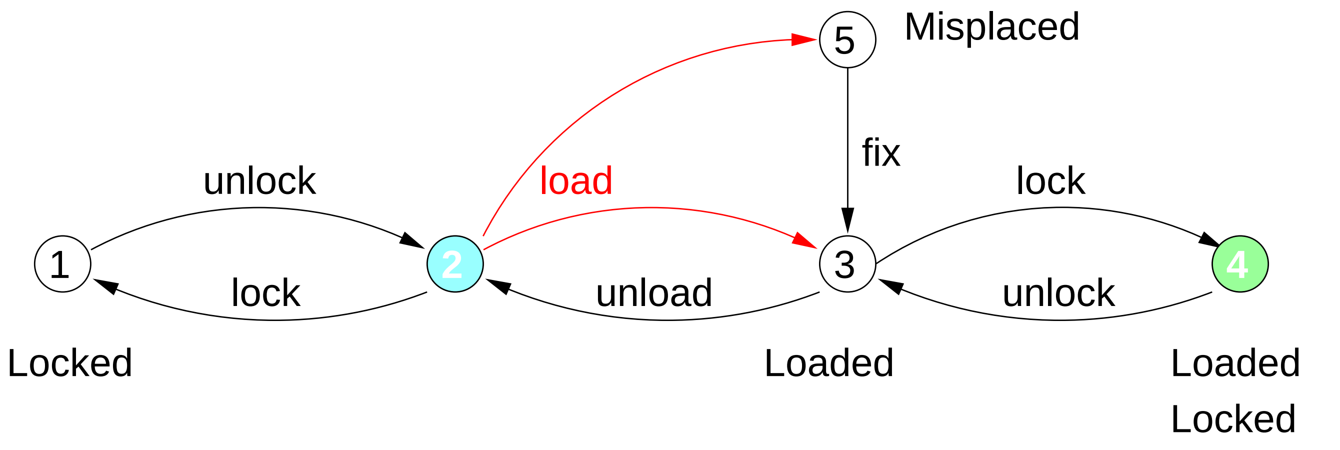


plan = load ; lock

Planning in non-deterministic domains: Motivations

In several practical domains:

- The initial situation is partially specified
- Actions have non-deterministic effects
- Plans as sequences of actions are bound to failure

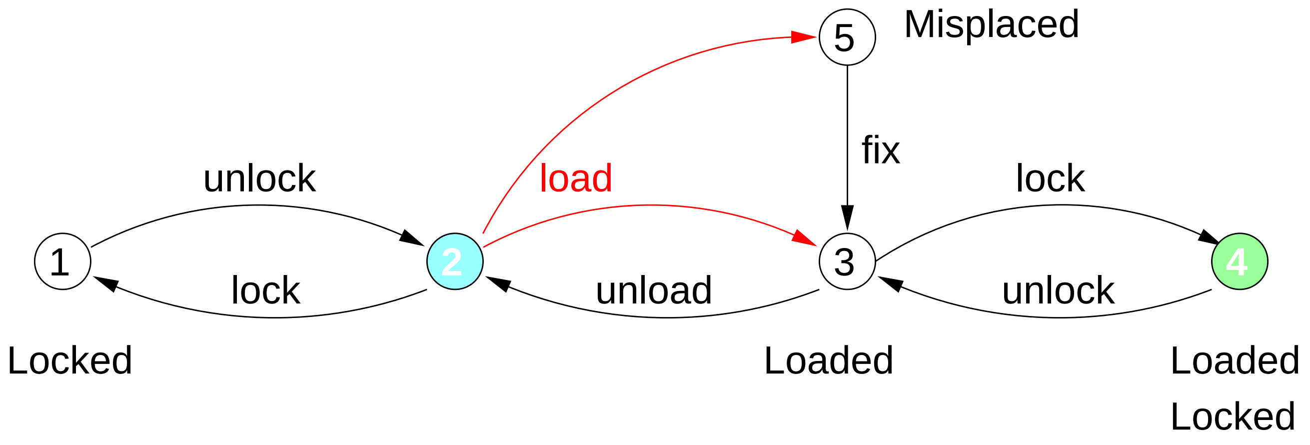


plan = load ; lock

Planning in non-deterministic domains: Motivations

Several practical domains require:

- Conditional Plans
- Strong Plans, i.e., plans that are guaranteed to achieve the goal in spite of non determinism.

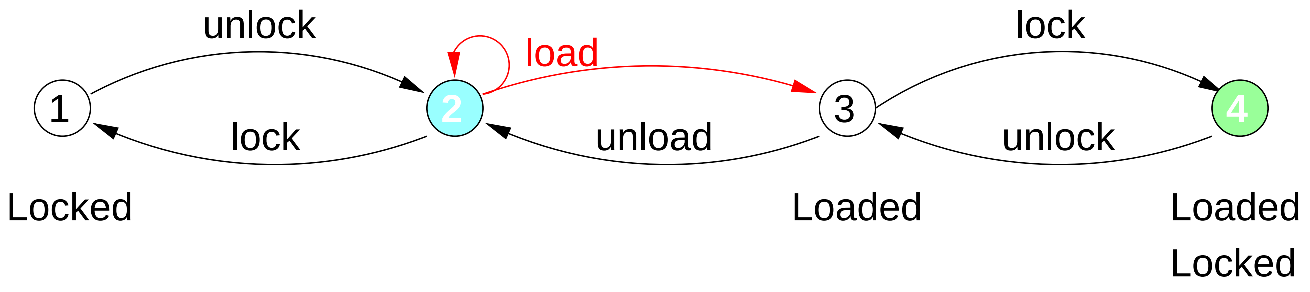


plan = load ; if Misplaced then fix ; lock

Planning in non-deterministic domains: Motivations

Several practical domains require:

- Iterative Plans, encoding trial-and-error strategies
- Strong Cyclic Plans, i.e., plans whose executions
  - ▷ always have a possibility of terminating and, ...
  - ▷ when they do, they are guaranteed to achieve the goal



plan = repeat load until Loaded ; lock

Planning with non-determinism: State of the Art

- Reactive Systems (focus on execution, no plan generation)
- Contingency Planning (lack of efficient automatic planners)
- Conformant Planning (solutions are sequences of actions)
- Stochastic Planning (focus on probability distributions)

Automatic and efficient generation of plans in non-deterministic domains is still an open problem

Planning as Model Checking: The idea

A model-based approach to planning:

- Planning domains are formalized as semantic models.
- Planning Problems are formalized via temporal formulas.
- Planning is done by constructing sets of states where a temporal formula is true.

Some Features:

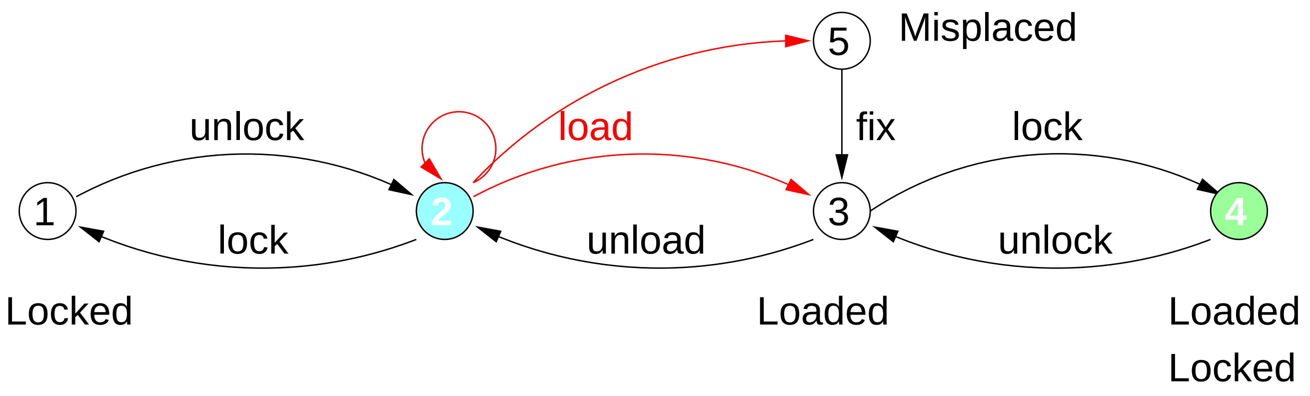
- Well-founded (clear and intuitive (semantic) formalization).
- General (e.g. non-deterministic planning).
- Practical (automatic and efficient planning for large size problems).

Planning Domains

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

1. F is a finite set of *fluents*,
2. $S \subseteq 2^F$ is a finite set of *states*,
3. A is a finite set of *actions*,
4. $R \subseteq S \times A \times S$ is a *transition relation*.

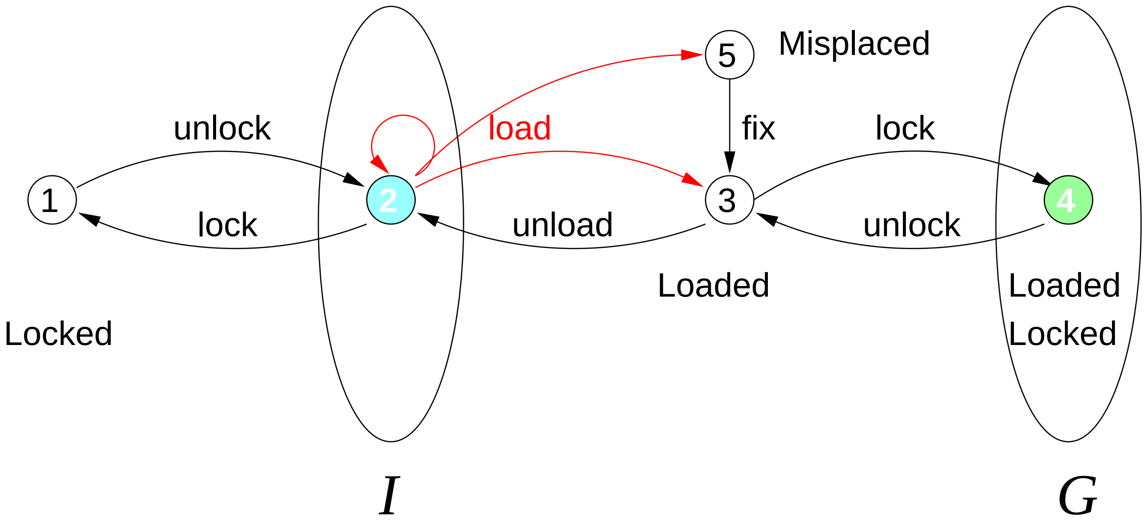
$a \in A$ is executable in $s \in S$ if there exists $s' \in S$ such that $R(s, a, s')$



Planning Problems

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

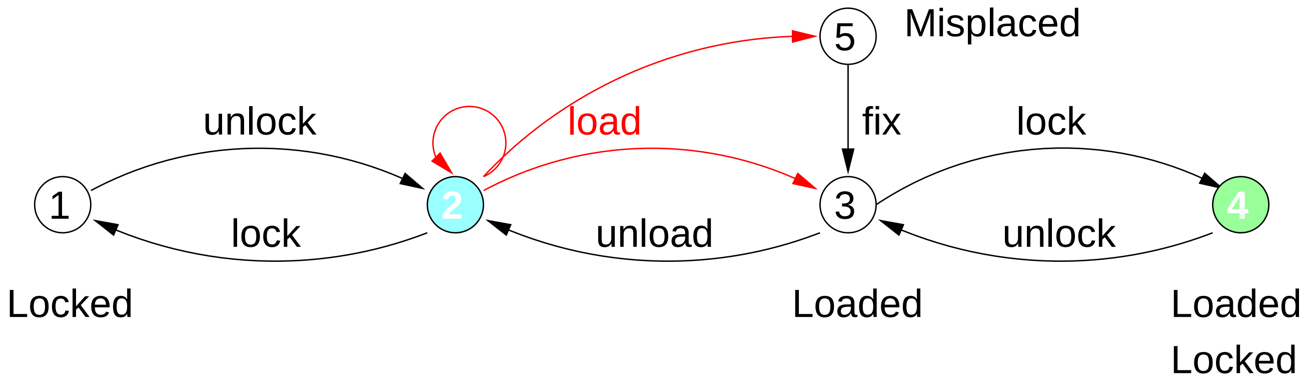
1. $I \subseteq S$ is the set of initial states,
2. $G \subseteq S$ is the set of goal states.



Plans

A *plan* for a planning domain $D = \langle F, S, A, R \rangle$ is a set of pairs
$\{\langle s, a \rangle : \ s \in S, a \in A, a$ executable in $s\}$

Set of states of a plan π: $States(\pi) = \{s : \exists a \in A. \langle s, a \rangle \in \pi\}$.

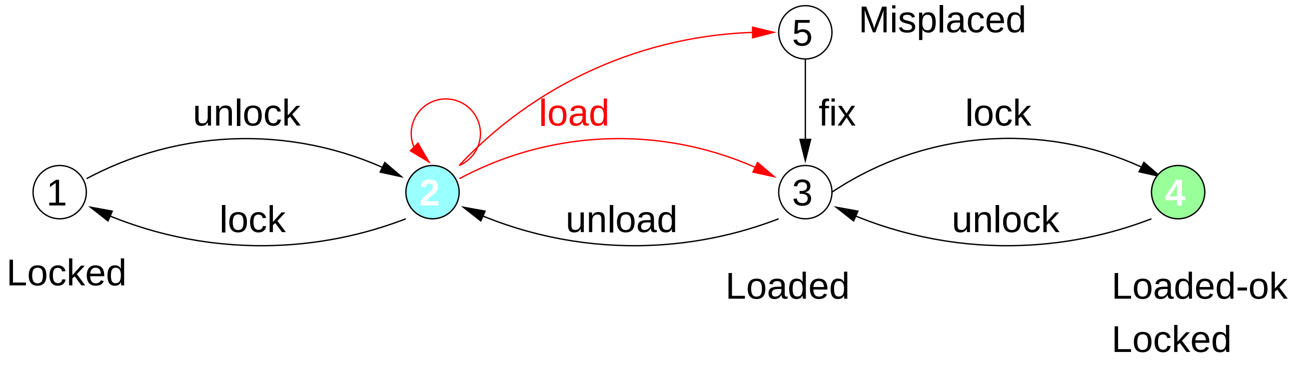


plan = { <2,load> , <3,lock> , <5,fix> }

Plan Executions

An *execution* of a plan π from the state $s_0 \in S$ is an infinite sequence $s_0, s_1, \ldots$ of states in S such that, for all $i \geq 0$,

1. if $s_i \in States(\pi)$ then, for some $a \in A$, $\langle s_i, a \rangle \in \pi$ and $s_{i+1} \in R(s_i, a)$.
2. if $s_i \notin States(\pi)$ then $s_i = s_{i+1}$,



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

executions = 1 , 1 , 1 , 1 , ...
2 , 2 , 2 , 2 , ...
2 , 2 , 3 , 4 , ...
2 , 3 , 4 , 4 , ...
2 , 5 , 3 , 4 , ...

Background: Kripke Structures

A Kripke Structure K is a 4-tuple $\langle W, W_0, T, L \rangle$, where

1. W is a finite set of $states$.
2. $W_0 \subseteq W$ is a set of initial states.
3. $T \subseteq W \times W$ is a binary relation on W, the $transition\ relation$, which gives the possible transitions between states. We require T to be total, i.e., for each state $w \in W$ there exists a state $w' \in W$ such that $(w, w') \in T$.
4. $L : W \mapsto 2^{\mathcal{P}}$ is a $labeling\ function$, where $\mathcal{P}$ is a set of atomic propositions. L assigns to each state the set of atomic propositions true in that state.

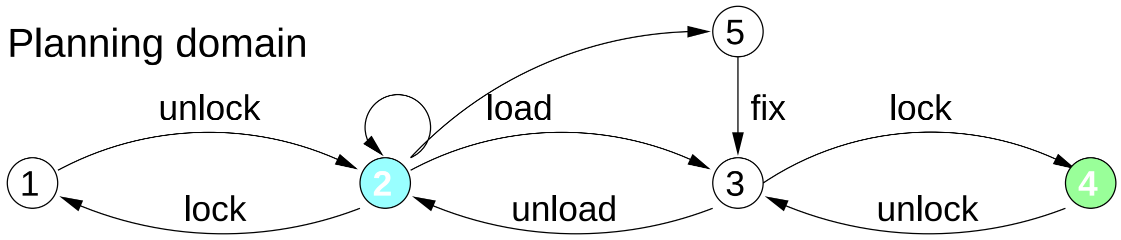
Induced Kripke Structures

Let π be a plan for the planning domain D.

The Kripke structure $K = \langle W, T, L \rangle$ induced by π is defined as

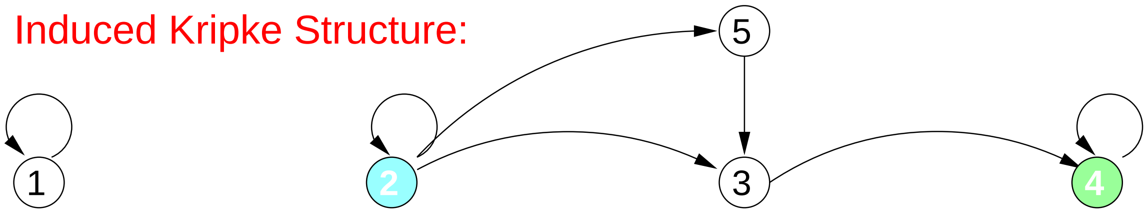
1. $W = S$;
2. $T(s, s')$ iff $\langle s, a \rangle \in \pi$ and $s' \in R(s, a)$, or $s \notin States(\pi)$ and $s = s'$;
3. $L(s) = s$.

Planning domain



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

Induced Kripke Structure:



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 $\mathbf{E}(p\mathbf{U}q)$)

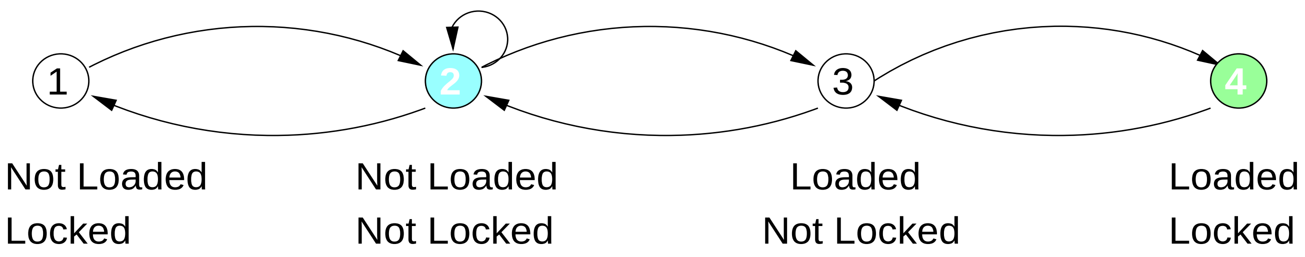
– $\mathbf{AX}p$ ($\mathbf{EX}p$): p holds in every (in some) "next" state.

– $\mathbf{A}(p\mathbf{U}q)$ ($\mathbf{E}(p\mathbf{U}q)$): 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 $\neg\mathbf{AF}\neg p$ ($\neg\mathbf{EF}\neg p$)

Computation Tree Logic (CTL): Examples



$\mathbf{EF}\, Loaded$ holds in state 2

$\mathbf{AF}\, Loaded$ does not hold in state 2

$\mathbf{EG}\, Loaded$ holds in state 2

$\mathbf{AG}\, Loaded$ does not hold in state 2

The Model Checking Problem

CTL semantics (Kripke Structure K, CTL formula p):

- $K, w \models p$ iff $p \in L(w)$, $p \in \mathcal{P}$
- $K, w \models \neg p$ iff $K, w \not\models p$
- $K, w \models p \vee q$ iff $K, w \models p$ or $K, w \models q$
- $K, w \models \mathbf{AX}p$ iff for all paths $\pi = w_0 w_1 \ldots$, with $w = w_0$, $K, w_1 \models p$
- $K, w \models \mathbf{EX}p$ iff there exists a path $\pi = w_0 w_1 \ldots$, with $w = w_0$, such that $K, w_1 \models p$
- $K, w \models \mathbf{A}(p\mathbf{U}q)$ iff for all paths $\pi = w_0 w_1 \ldots$, with $w = w_0$, there exists $i \geq 0$ such that $K, w_i \models q$ and, for all $0 \leq j < i$, $K, w_j \models p$
- $K, w \models \mathbf{E}(p\mathbf{U}q)$ iff there exists a path $\pi = w_0 w_1 \ldots$, with $w = w_0$, and $i \geq 0$ such that $K, w_i \models q$ and, for all $0 \leq j < i$, $K, w_j \models p$

Truth: p is true in K ($K \models p$) if $K, w \models p$ for each $w \in W_0$.

The Model Checking Problem for a CTL formula p and a Kripke Structure K is the problem of determining whether p is true in K

Weak Solutions

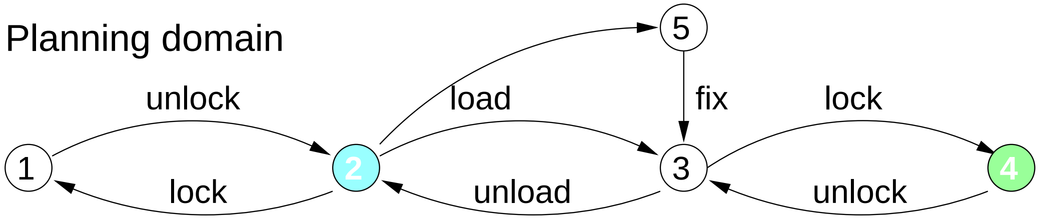
Given: A planning domain D and problem P

A plan π and th Induced Kripke Structure K

The plan π is a weak solution to the planning problem P if

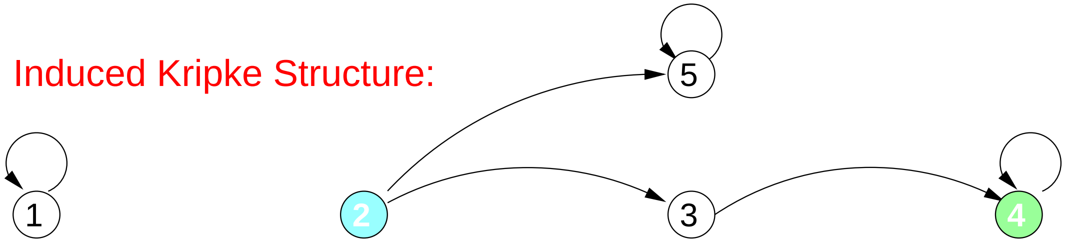
$$\text{for all } s \in I, \quad K, s \models \texttt{EF}\ G.$$

Planning domain



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

Induced Kripke Structure:



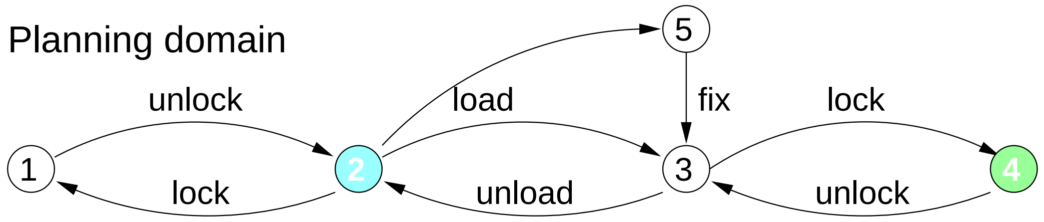
Strong Solutions

Given: A planning domain D and problem P
A plan π and th Induced Kripke Structure K

The plan π is a strong solution to the planning problem P if

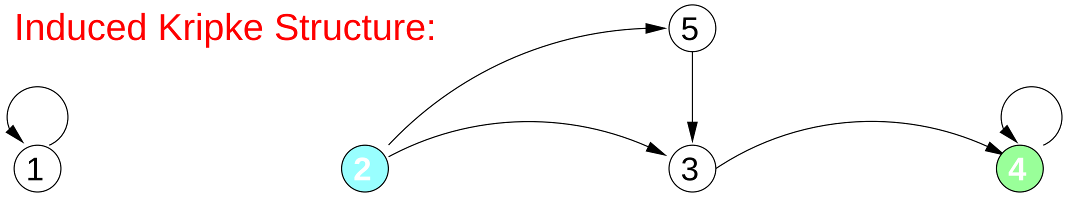
$$\text{for all } s \in I, \quad K, s \models \mathtt{AF}\ G.$$

Planning domain



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

Induced Kripke Structure:



Strong Cyclic Solutions

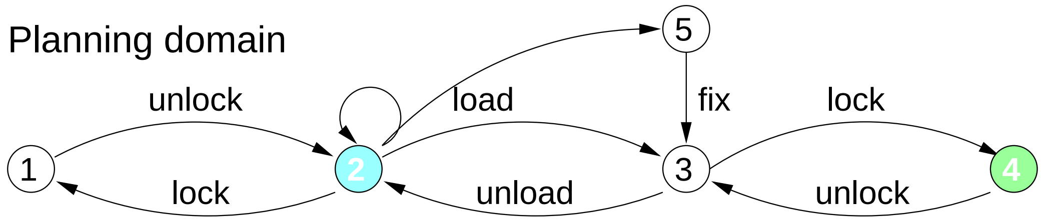
Given: A planning domain D and problem P

A plan π and th Induced Kripke Structure K

The plan π is a strong cyclic solution to the planning problem P if

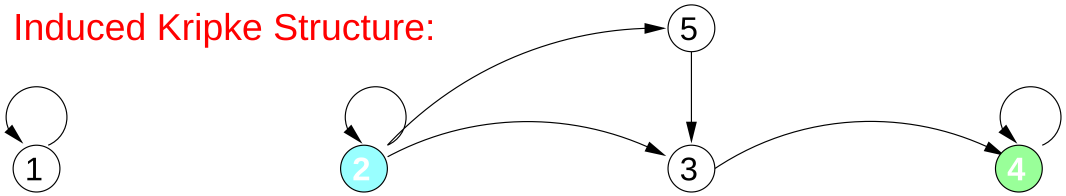
$$\text{for all } s \in I, \quad K, s \models \mathtt{AGEF}\ G.$$

Planning domain



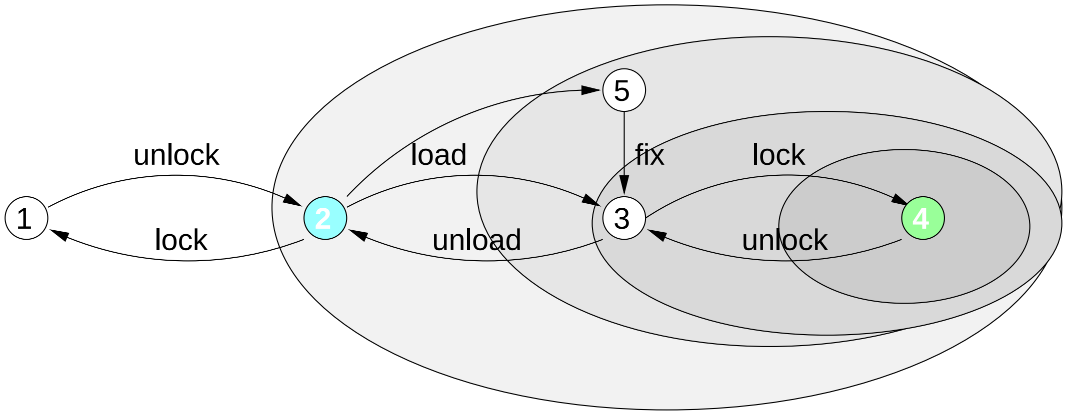
plan = { <2,load> , <3,lock> , <5,fix> }

Induced Kripke Structure:



Planning Algorithms

- Sets of states manipulated during the search
- OBDD based implementation



Weak Planning

. **function** $\textsc{WeakPlan}(P)$

. $States := \emptyset$; $Plan := \emptyset$; ...

. **while** $I \not\subseteq States$ **and** not all states visited so far **do**

. $\pi := \pi \cup \textsc{PruneStates}(\textsc{WeakPreImg}(States, D), States)$.

. $States := States \cup \textsc{ProjectActions}(\textsc{WeakPreImg}(States, D))$

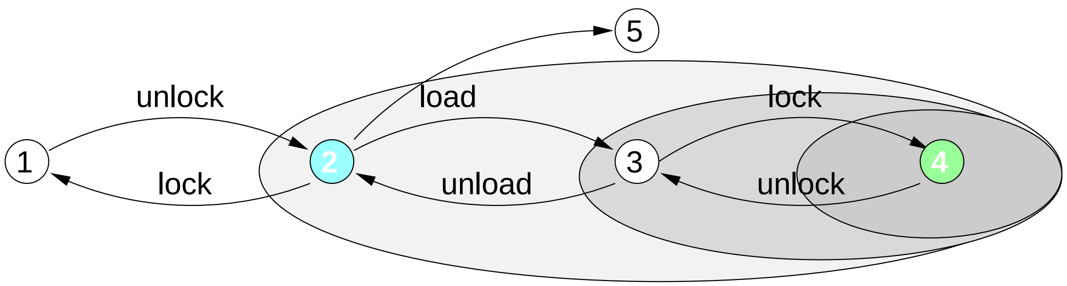
. **if** $I \not\subseteq States$ **then return** π **else return** $Fail$

$\textsc{WeakPreImg}(States, \pi)$:

returns all the $\langle s, a\rangle$ possibly leading to states of $States$

$\textsc{PruneStates}(\pi, States)$:

eliminates from π the $\langle s, a\rangle$ whose states are already in $States$



Weak Planning

$$
\begin{array}{l}
\textbf{function } \textsc{Plan}(P);\\
\quad CurrentStates := \emptyset;\ NextStates := G;\ Plan := \emptyset;\\
\quad \textbf{while } (NextStates \neq CurrentStates) \textbf{ do}\\
\qquad\quad \textbf{if } I \subseteq NextStates\\
\qquad\qquad\quad \textbf{then return } Plan;\\
\qquad\quad OneStepPlan := \textsc{WeakPreImg}(NextStates, D);\\
\qquad\quad Plan := Plan \cup \textsc{PruneStates}(OneStepPlan, NextStates);\\
\qquad\quad CurrentStates := NextStates;\\
\qquad\quad NextStates := NextStates \cup \textsc{ProjectActions}(OneStepPlan);\\
\quad \textbf{return } Fail;
\end{array}
$$

$$
\begin{aligned}
&\textsc{WeakPreImg}(States, D) = \\
&\qquad \{\langle s, a\rangle : s \in S, a \in A, \exists s'.\ (s' \in States \wedge R(s, a, s'))\}
\end{aligned} \tag{1}
$$

$$
\textsc{PruneStates}(\pi, States) = \{\langle s, a\rangle \in \pi : s \notin States\} \tag{2}
$$

$$
\textsc{ProjectActions}(\pi) = \{s : \langle s, a\rangle \in \pi\} \tag{3}
$$

Strong Planning

. **function** $\textsc{StrongPlan}(P)$

. $\quad States := \emptyset;\ Plan := \emptyset;\ \ldots$

. $\quad$ **while** $I \not\subseteq States$ **and** not all states visited so far **do**

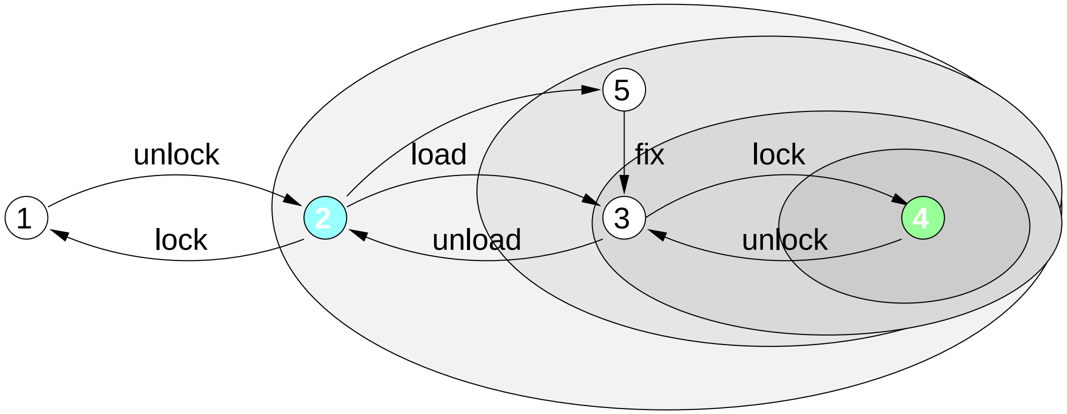
. $\quad\quad \pi := \pi \cup \textsc{PruneStates}(\textsc{StrongPreImg}(States, D), States).$

. $\quad\quad States := States \cup \textsc{ProjectActions}(\textsc{StrongPreImg}(States, D))$

. $\quad$ **if** $I \not\subseteq States$ **then return** π **else return** $Fail$

$\textsc{StrongPreImg}(States, \pi)$:

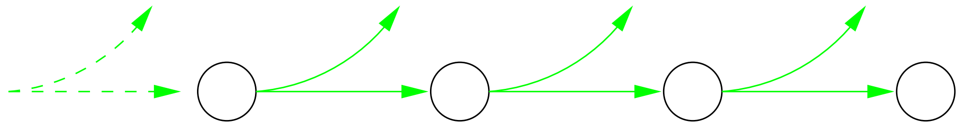
returns all the $\langle s, a \rangle$ necessarily leading to states of $States$



The strong cyclic planning algorithm: intuitions

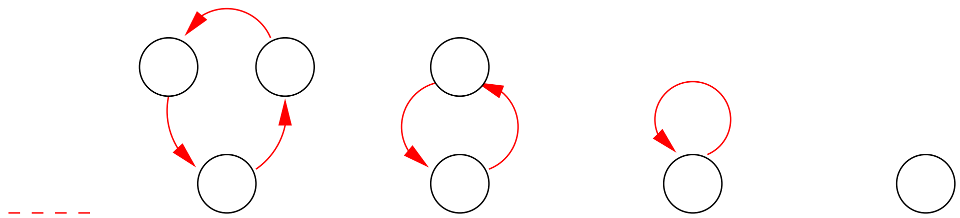
Strong cyclic plans: plans whose executions satisfy **AGEF** G

EFp = p or EXp or EXEXp or EXEXEXp or ...



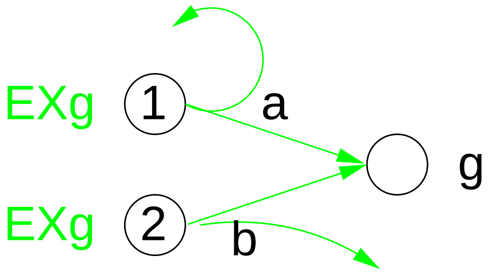
EXEXEXp EXEXp EXp p

AGq = q and AXq and AXAXq and AXAXAXq and ...

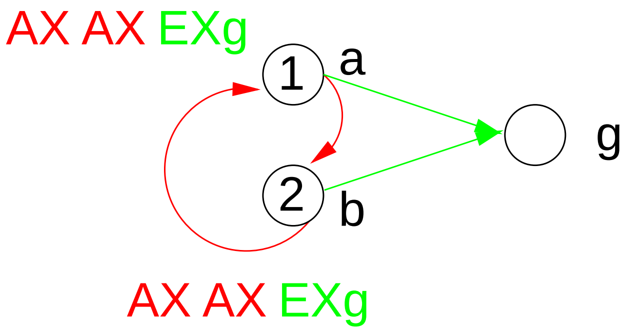


AXAXAXq AXAXq AXq q

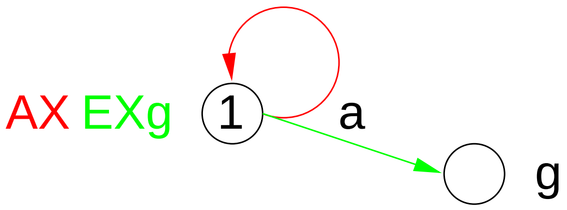
Strong Cyclic Planning: intuitions (cont.)



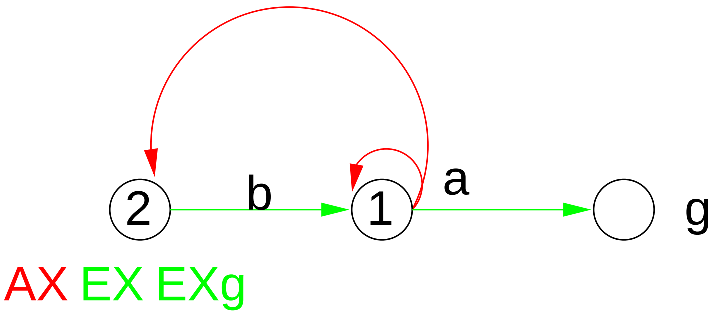
plan = {<1,a>,<2,b>}



plan = {<1,a>,<2,b>}



plan = {<1,a>}



plan = {<1,a>,<2,b>}

Strong Cyclic Planning

function $\textsc{StrongCyclicPlan}(P)$
 $\pi :=$ emptyplan; ...
 while $I \not\subseteq$ states of π **and** not all states visited so far **do**
  $\pi := \textsc{OneStepBack}(P,\ \pi)$
  while π has state-action pairs to be pruned away **do**
   $\pi := \textsc{PruneUnconnected}(P,\ \textsc{PruneOutgoing}(P,\ \pi))$
 if $I \not\subseteq$ states of π **then return** π **else return** Fail

$\textsc{OneStepBack}(P,\ \pi)$:

returns all the $\langle s, a\rangle$ possibly leading to states of π or G

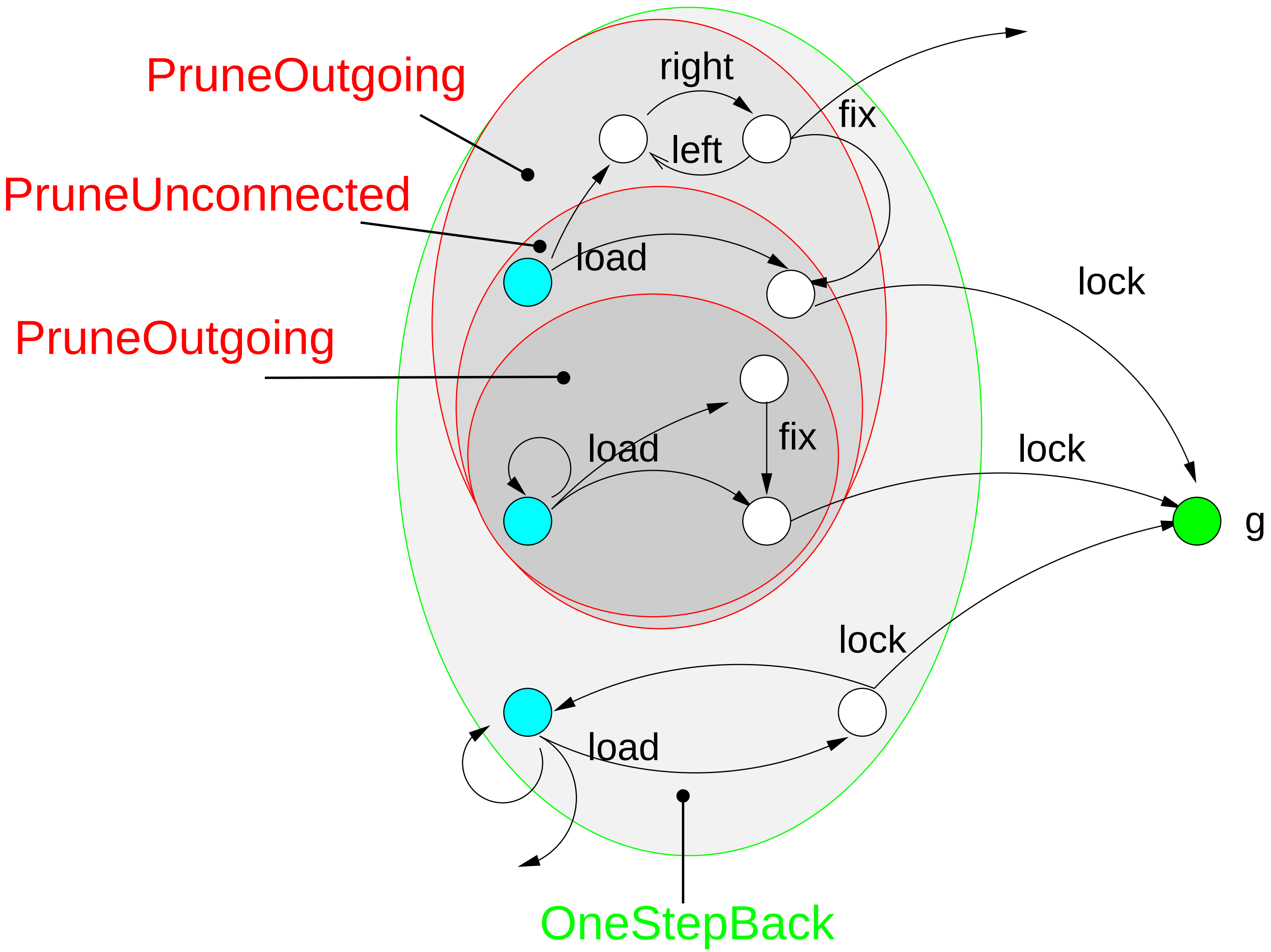
$\textsc{PruneOutgoing}(P,\ \pi)$:

removes all $\langle s, a\rangle$ that may lead to non-goal states not in π

$\textsc{PruneUnconnected}(P,\ \pi)$:

removes loops with no possibility of termination

The strong cyclic planning algorithm: intuitions (cont.)



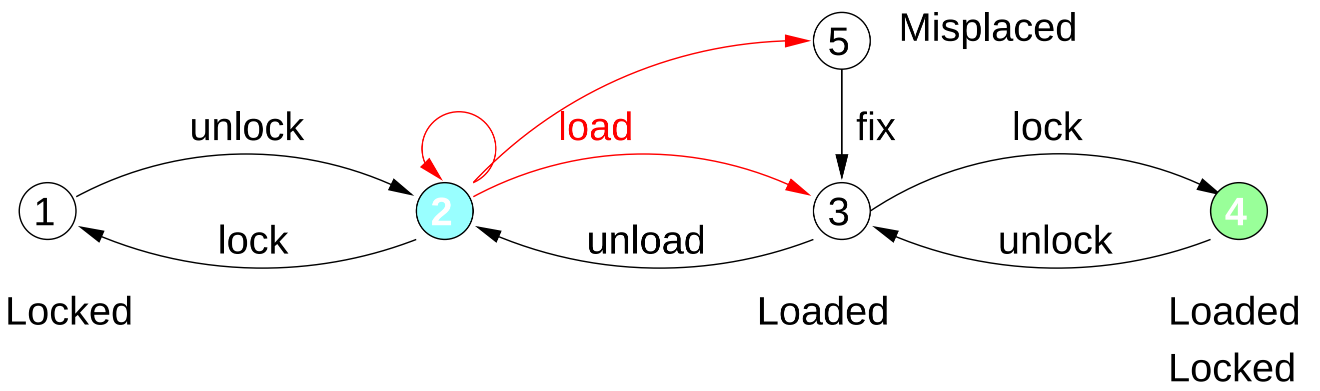
Planning via Symbolic Model Checking

- The problem: realistic planning domains have often large state spaces.
- The idea: exploit the work on *symbolic model checking* based on *Ordered Binary Decision Diagrams* (OBDD's)
- The results: the Model Based Planner (MBP), built on top of NuSMV, a state of the art OBDD based symbolic model checker
- The approach:
  - Sets (e.g. sets of states, plans) as propositional and quantified boolean formulae (QBF)
  - Propositional formulas as OBDD's

Planning Domains via Symbolic Model Checking

Let D be a planning domain $\langle F, S, A, R \rangle$:

- Fluents as boolean variables (state variables) $\mathbf{x}$
- A state corresponds to a complete assignment to the variables in $\mathbf{x}$.
- A propositional formula $\phi(\mathbf{x})$ encodes the set of the states corresponding to the assignments that make ϕ true
- Actions as boolean variables (action variables) α
- A formula $\psi(\mathbf{x}, \alpha)$ encodes a relation between states and actions (e.g., a plan)

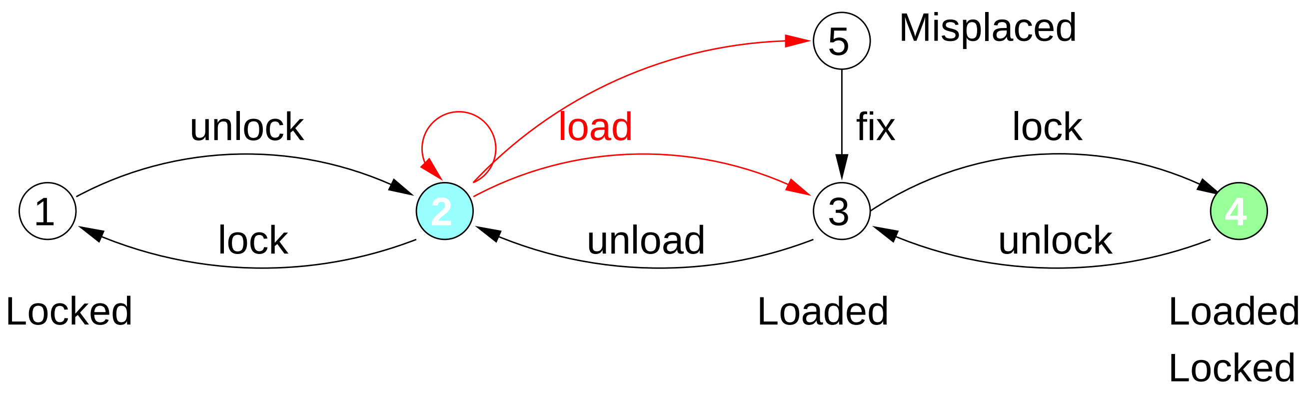


Planning Domains via Symbolic Model Checking (cont.)

Let D be a planning domain $\langle F, S, A, R \rangle$:

- Transition relations are formulas in state variables $\mathbf{x}$, action variables α, and *next* state variables $\mathbf{x}'$: $\mathcal{R}(\mathbf{x}, \alpha, \mathbf{x}')$
- Example:

$$(\neg Loaded \wedge \neg Locked \wedge \neg Misplaced \wedge Act = load) \supset \neg Locked' \wedge \neg(Loaded' \wedge Misplaced')$$

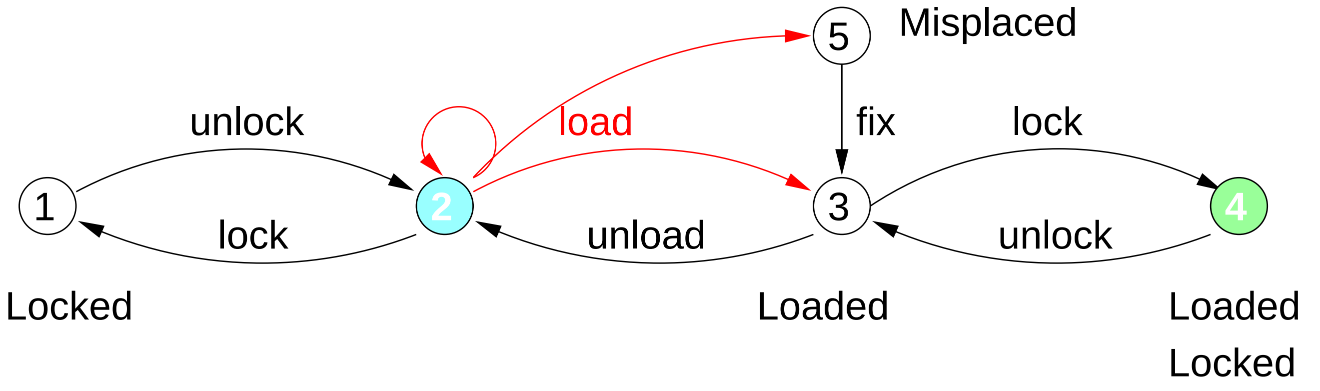


Planning Domains via Symbolic Model Checking (cont.)

Let D be a planning domain $\langle F, S, A, R \rangle$:

- Transition relations are formulas in state variables $\mathbf{x}$, action variables α, and *next* state variables $\mathbf{x}'$: $\mathcal{R}(\mathbf{x}, \alpha, \mathbf{x}')$
- Example:

$$(\neg Loaded \wedge \neg Locked \wedge \neg Misplaced \wedge Act = load) \supset \neg Locked'$$



Planning via Symbolic Model Checking: Algorithms

Some examples:

$\textsc{ProjectActions}(\pi)$ represented by the formula

$$\phi(\mathbf{x}) \quad \doteq \quad \exists\alpha(\pi(\mathbf{x},\alpha))$$

$\textsc{WeakPreImg}(States, D)$ represented by the formula

$$\psi(\mathbf{x},\alpha) \quad \doteq \quad \exists\mathbf{x}'(States(\mathbf{x}') \wedge \mathcal{R}(\mathbf{x},\alpha,\mathbf{x}'))$$

Planning via Symbolic Model Checking: Algorithms

Some examples:

$\textsc{WeakPreImg}(States, D)$ represented by the formula

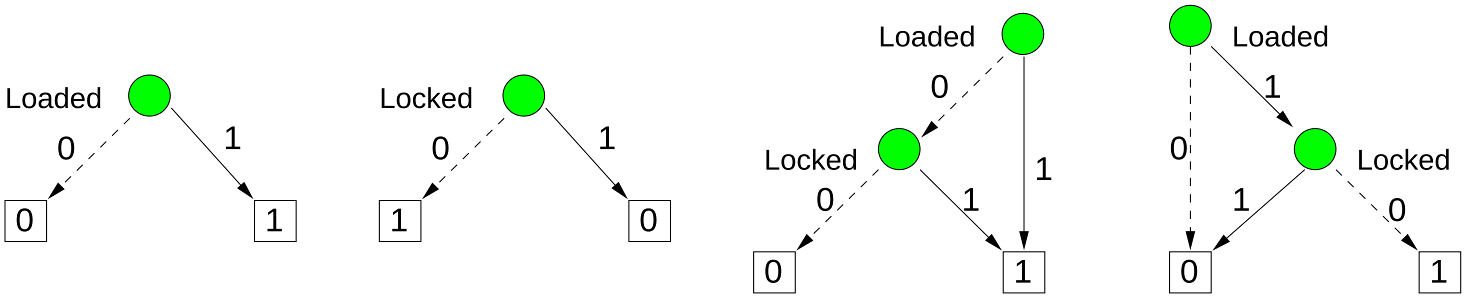
$$\psi(\mathbf{x}, \alpha) \quad \doteq \quad \exists \mathbf{x}' (States(\mathbf{x}') \wedge \mathcal{R}(\mathbf{x}, \alpha, \mathbf{x}'))$$

$\textsc{StrongPreImg}(States, D)$ represented by the formula

$$\phi(\mathbf{x}, \alpha) \quad \doteq \quad \neg \exists \mathbf{x}' (\neg States(\mathbf{x}') \wedge \mathcal{R}(\mathbf{x}, \alpha, \mathbf{x}'))$$

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.

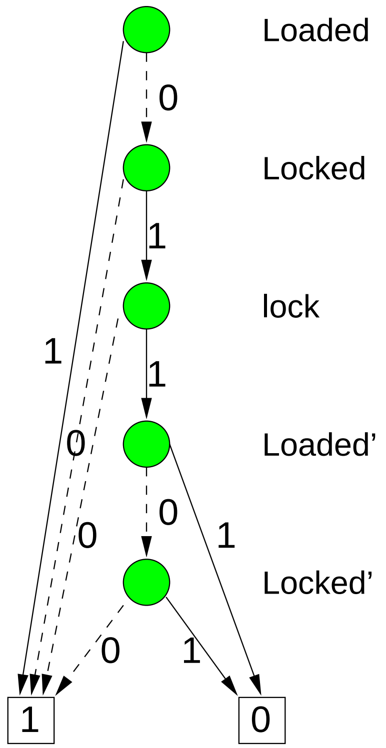


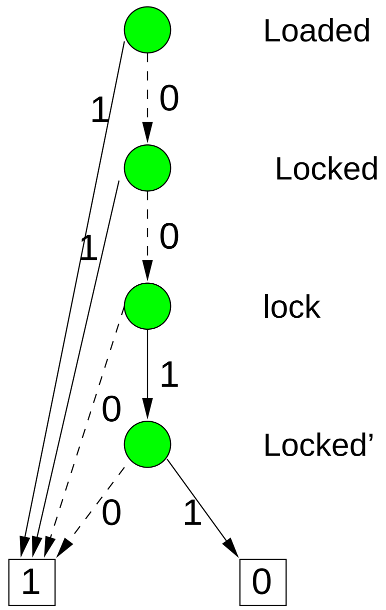
Loaded *Not Locked* *Loaded or Locked* *Loaded and not Locked*

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)



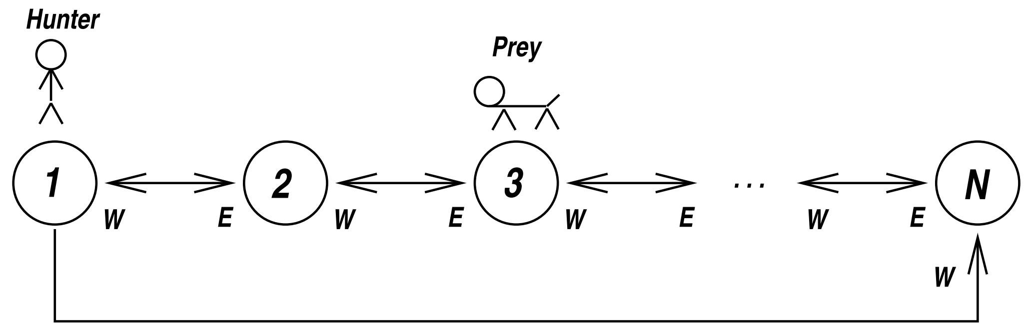


$(\neg Loaded \wedge Locked \wedge lock) \supset$
$\neg Loaded' \wedge \neg Locked'$

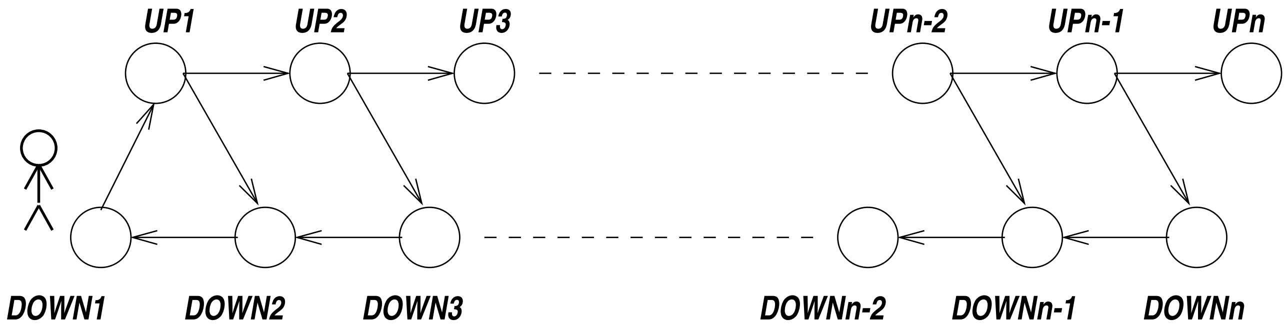
$(\neg Loaded \wedge \neg Locked \wedge lock) \supset \neg Locked'$

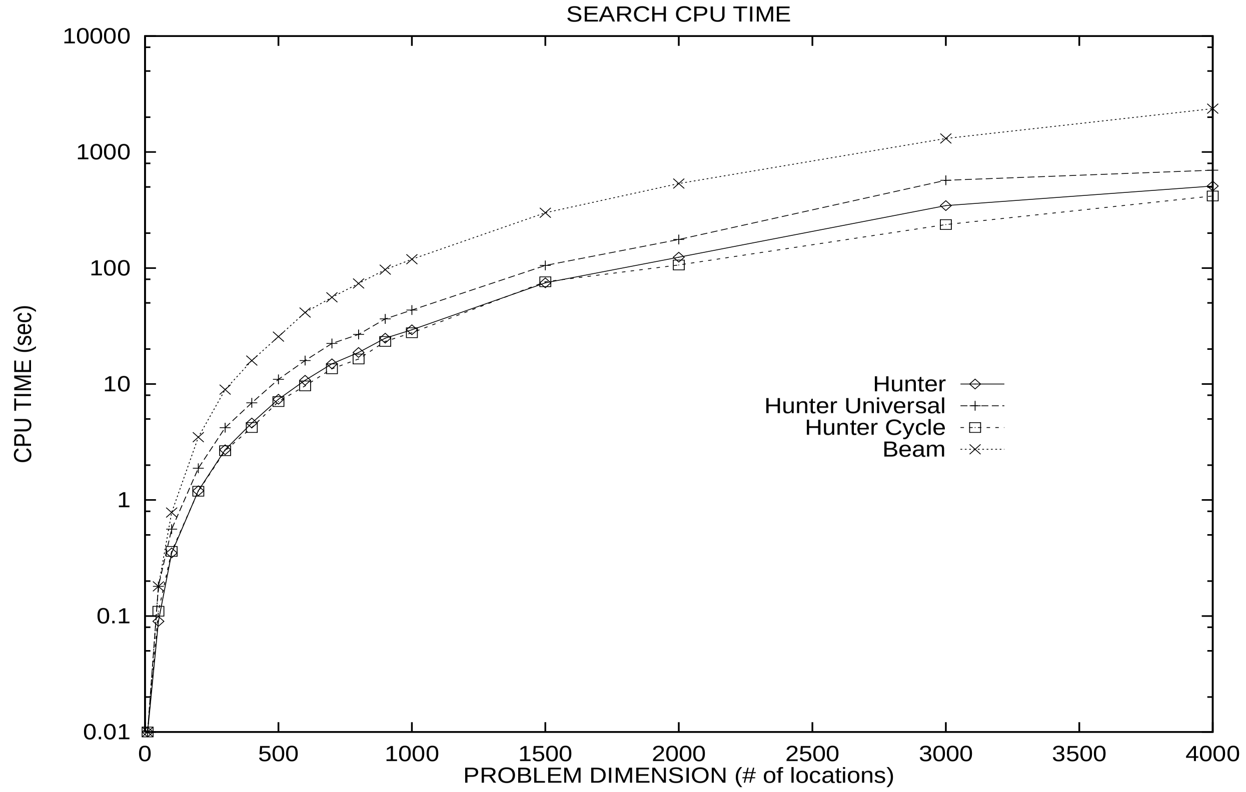
Experimental Analysis: the problems

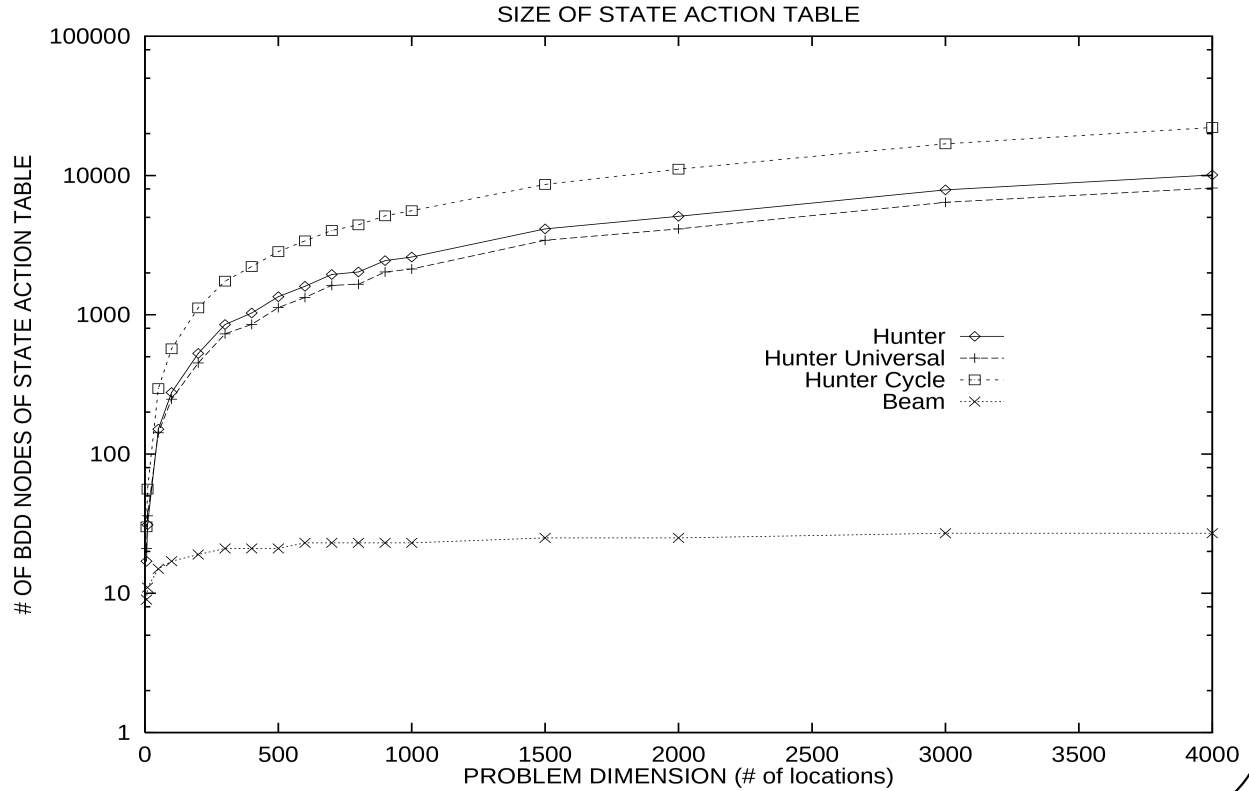
- the Hunter and Prey problem (extensions: universal; cyclic)



- the Beam problem (strong cyclic solutions)




SEARCH CPU TIME
CPU TIME (sec)
10000
1000
100
10
1
0.1
0.01
0
500
1000
1500
2000
2500
3000
3500
4000
PROBLEM DIMENSION (# of locations)
Hunter
Hunter Universal
Hunter Cycle
Beam


SIZE OF STATE ACTION TABLE
OF BDD NODES OF STATE ACTION TABLE
100000
10000
1000
100
10
1
0
500
1000
1500
2000
2500
3000
3500
4000
PROBLEM DIMENSION (# of locations)
Hunter
Hunter Universal
Hunter Cycle
Beam

Future Work

- Partial observability.
- Efficient algorithms for planning in deterministic domains.
- Foundations of Planning as Model Checking (expressiveness).
- Reactive Planning (planning and execution)

Conclusions

Planing as Model Checking is:

- a novel approach to planning;
- a solution to some problems in planning with non-determinism;
- well founded (correct, complete algorithms);
- practical (fast algorithms);
- a step towards planning as synthesis from temporal specifications;

Bibliography

A. Cimatti, E. Giunchiglia, F. Giunchiglia, P. Traverso. *Planning via Model Checking: A Decision Procedure for $\mathcal{AR}$*, ECP97

Cimatti, Roveri, Traverso: *Strong Planning in Non-Deterministic Domains via Model Checking*, AIPS98

Cimatti, Roveri, Traverso. *Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains*, AAAI98

Cimatti,Roveri: *Conformant planning via model checking*, ECP99

Daniele,Traverso,Vardi. *Strong Cyclic Planning Revisited*, ECP99

F. Giunchiglia, P. Traverso. *Planning as Model Checking*, ECP99

Acknowledgements

Piergiorgio Bertoli (IRST)

Alessandro Cimatti (IRST)

Marco Daniele (IRST)

Enrico Giunchiglia (University of Genoa)

Fausto Giunchiglia (University of Trento and IRST)

Marco Roveri (IRST)

Marco Pistore (IRST)

Moshe Y. Vardi (Rice University)