Course "Automated Planning: Theory and Practice"
Chapter 17: Planning in Non Deterministic Domains via Model Checking

Teacher: Marco Roveri - marco.roveri@unitn.it

M.S. Course: Artificial Intelligence Systems (LM)
A.A.: 2025-2026
Where: DISI, University of Trento
URL: https://shorturl.at/A81hf

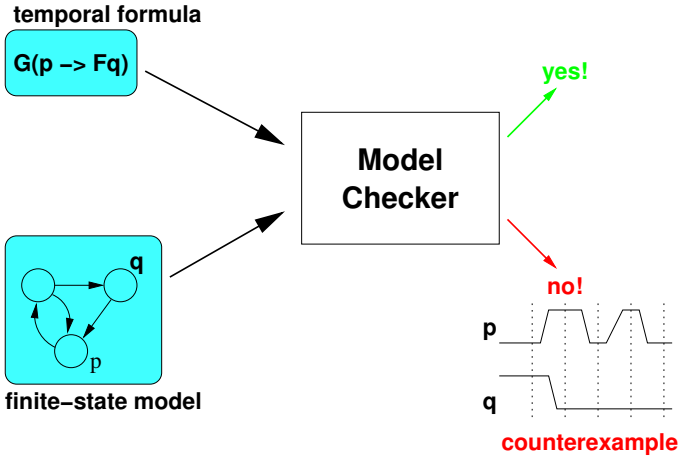
Last updated: Wednesday 12th November, 2025

Terms of Use and Copyright

The material (text, figures) in these slides is authored by Marco Roveri.

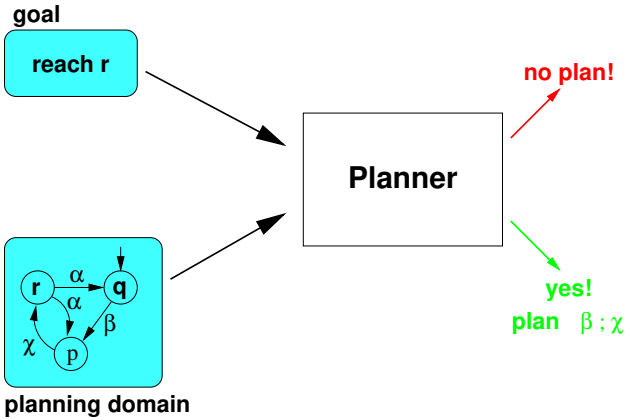
Model Checking

Model Checking: a technique to *validate* a *formal model* of a system against a *logical specification*.



Planning as Model Checking

Planning as Model Checking: a technique to *synthesize* a plan from a *formal model of a domain* and a *logical specification of a goal*.

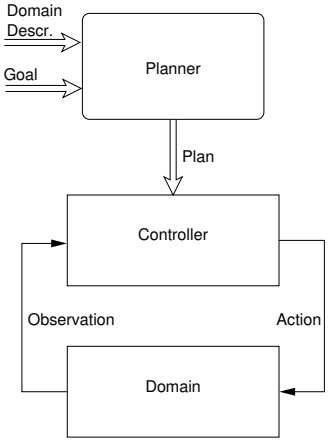


Planning

- "Planning" is the problem of building a suitable *plan* for achieving a given *goal* on a given dynamic domain.
- The plan describes the actions to be executed in the domain in order to achieve the goal.

- Different approaches to planning differ for:
  - the assumptions on the *domain*.
  - the assumptions on the *goals*.
  - the assumptions on the *plans*.
  - the techniques used in the *planning algorithm*.

The conceptual model



- The *domain* evolves according to performed *actions*.
- The *controller* provides the *actions* according to the *observations* on the domain and to a *plan*.
- The *planner* synthesizes a *plan* from a *domain description* and a *goal*.

Classical planning: assumptions

Classical planning relies on simplifying assumptions:

- *finite*: the domain has a finite set of different states.
- *implicit time*: actions are instantaneous state transitions.
- *deterministic*: the initial state is completely specified; each action, if applicable in a state, brings to a single new state.
- *observable*: the state of the system is fully observable to the controller.
- *basic goals*: goals are sets of target states; the objective is to build a plan that leads to one of the goal states.
- *off-line planning*: the plan is built once and for all, before its execution.
- ...

Classical planning

- A plan is a *sequence of actions* whose execution leads from the initial state of the domain to a goal state.
  - the evolution is completely determined by the executed actions (deterministic domain).
  - no feedback is needed from the domain (observation is not necessary).
- Main difficulty:
  - in spite of simplifying assumptions, search space is huge for practical domains.

Non-Determinism

The restriction to *deterministic* domains is unrealistic in many practical cases:

- There may be uncertainty on the *initial state*.
- *Actions* may have *non-nominal outcomes* that are highly critical.
- There may be *exogenous events* or *non-controllable actions*.

In *non-deterministic domains*:

- More than one initial state is possible.
- An action executed in a state may lead to many different states.

Main difficulties:

- Plans cannot simply be sequences of actions.
- Domain models are more complex.
- Lifting most classical planning techniques does not work in practice.

Non-determinism: probabilistic outcomes

It is often possible to associate probabilities to the outcomes of actions:

- non-nominal outcomes of actions often have a very small probability.
- when throwing a fair dice, all the outcomes have equal probability.
- exogenous events occurs with a given probability.

In planning with probabilistic domains (e.g., MDP-based planning):

- probabilities are associated to the outcomes of actions.
- goals are represented by utility, or value functions.
- planning consists in searching for a plan maximizing the utility function.

Main difficulty:

- *optimality* comes into play.

We will not discuss probabilistic models!

Partial observability

In several realistic problems the domain status is only partially visible at run-time to the controller:

- different states of the domain are *indistinguishable* for the controller.
- certain information on the domain is available only after some *sensing actions*.

Particular cases:

- *full observability*.
- *null observability* (conformant planning).

Main difficulty:

- for non-deterministic domains, the controller has a partial knowledge of the domain status.

Extended goals

The assumption that planning goals are sets of final desired states is unrealistic in many cases:

- *safety conditions* (states to be avoided) may complement the main goal.
- *reactive systems* (infinite plans that react to changes).
- *preferences* among plans and *search control rules* can be specified.

Temporally extended goals express conditions on the whole, possibly infinite sequences of states resulting from plan execution.

Main difficulty:

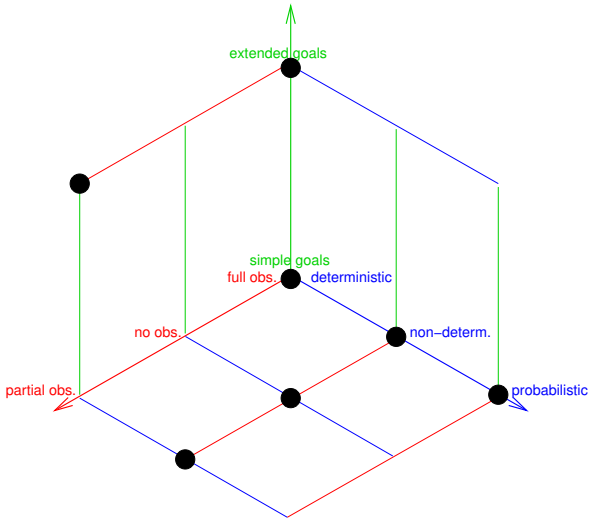
- the plan should trace past history (different "active" goals depending on past events).

Plans

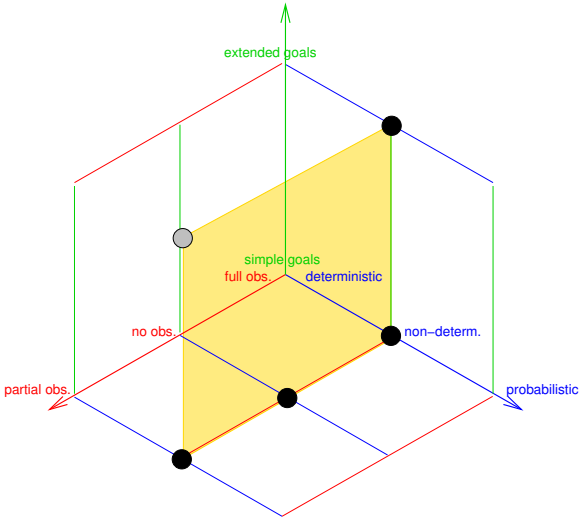
Different kinds of plans are necessary for dealing with different assumptions on domains and goals.

- *sequences of actions* are sufficient for *classical planning*.
- *conditional plans* are necessary for dealing with *non-deterministic* domains under *partial observability*.
- *history-dependent* plans are necessary for *extended goals*.
- *explicit time representation* may be necessary in plans.
- ...

Planning as Model Checking



In this lecture



Remarks on planning system...

...must not be limited to plan synthesis.

- *Plan validation*: verification that a *plan* satisfies a given *property* (not necessarily a goal...).
- *Plan simulation*: interactive simulation of the executions of a *plan* on a given *domain*.
- *Plan synthesis*: automatic generation of the plan from a description of the *domain* and from the *goal*.
- ...

Model Checking

- Model Checking is a widely used automatic technique for verifying design and for discovering possible bugs.

  In this lecture . . .
  - A mild introduction to model checking.
  - Symbolic representation of Finite State Machine (FSM) based on Binary Decision Diagrams (BDDs).
  - An introduction to symbolic model checking algorithms for Computation Tree Logic (CTL).

Model Checking (cont.)

- Model Checking is a verification technique that solves the problem

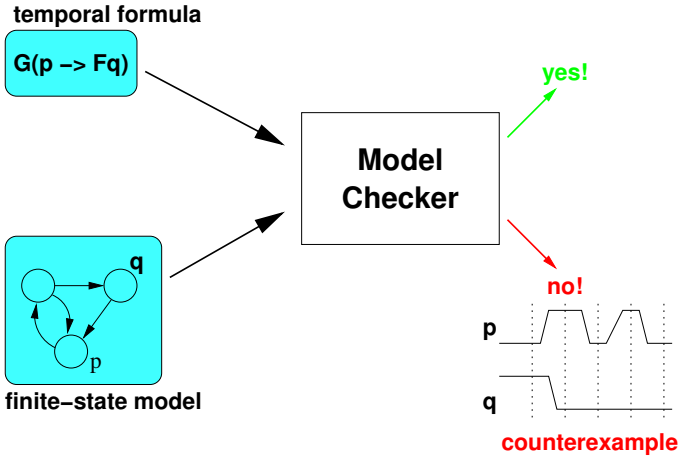
$$\mathcal{M} \models \varphi$$

where...

- Model $\mathcal{M}$ is represented as a FSM.
- Property φ is a temporal logic formula.
- Model checking algorithms exhaustively traverse the model guided by the property.

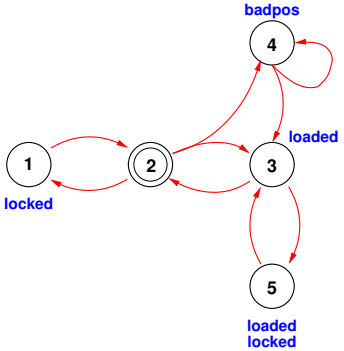
Model Checker

- Software tool for system verification.



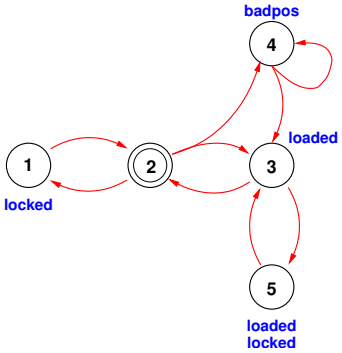
Formal Model: FSM

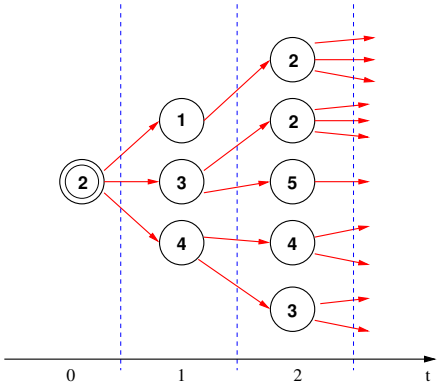
- A model is defined as a tuple $\mathcal{M} = \langle S, S_0, T, P, L \rangle$, where:
  - S is a finite set of states.
  - S_0 is the initial set of states.
  - P is a finite set of atomic propositions.
  - $T \subseteq S \times S$ is the transition relation.
  - $L : P \mapsto 2^S$ assigns to each proposition $p \in P$ the set of states where p holds.



Computation Trees

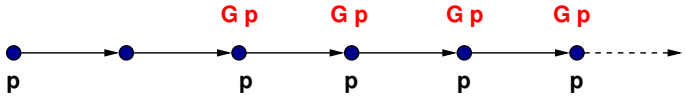
- Computation model is a *tree*:



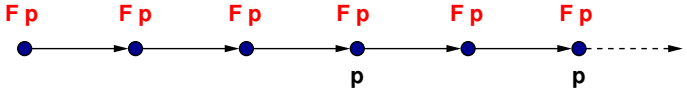


Temporal Logic: LTL

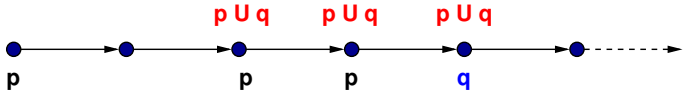
- Express properties of all computation paths.
  - Invariantly p: **G** p



  - Finally p: **F** p



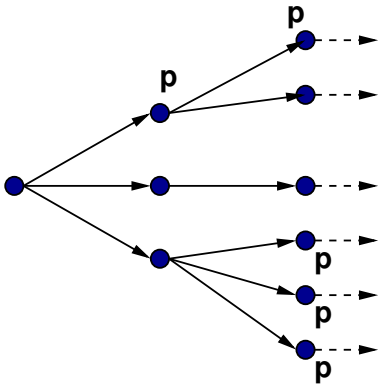
  - p holds until q holds: p **U** q



Temporal Logic: CTL

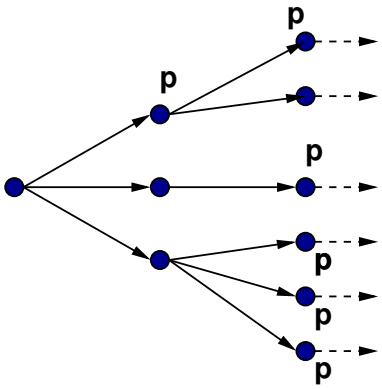
Express properties on computation paths.

- There exists a path where p finally holds: **EF** p



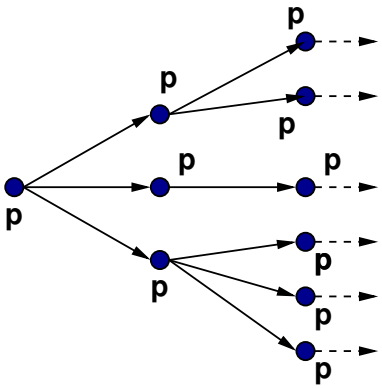
Temporal Logic: CTL (II)

- Property p finally holds for all paths: **AF** p



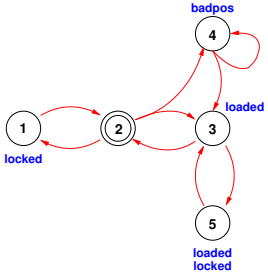
Temporal Logic: CTL (III)

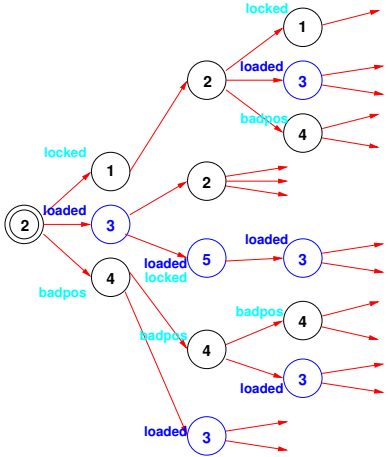
- Invariant p holds along all paths: **AG** p



CTL Examples: AG EF

- It is always possible to reach a state where *loaded* holds: ***AG EF loaded***





Symbolic Model Checking

- Symbolic model checking:
  - Face the *state explosion problem* of explicit state model checking.
  - Use the formula φ to represent the set of states where φ holds.

$$\varphi \rightsquigarrow \{s \mid M, s \models \varphi\}$$

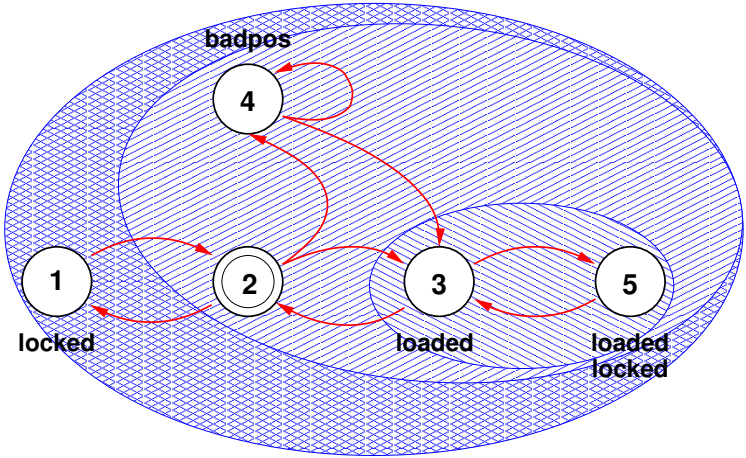
  - Use of Boolean formulas to represent sets and relations.
  - Fix-point characterization of CTL operators.

$$\begin{array}{ll} \mathbf{EF}p \leftrightarrow p \vee \mathbf{EXEF}p & \mathbf{EF}p = lfp_Z\{p \cup \mathbf{EX}Z\} \\ \mathbf{E}[p\mathbf{U}q] \leftrightarrow (q \vee (p \wedge \mathbf{EX}(\mathbf{E}[p\mathbf{U}q])) & \mathbf{E}[p\mathbf{U}q] = lfp_Z\{q \cup (p \cap \mathbf{EX}Z)\} \\ \mathbf{AG}p \leftrightarrow p \wedge \mathbf{AXAG}p & \mathbf{AG}p = gfp_Z\{p \cap \mathbf{AX}Z\} \\ \mathbf{EG}p \leftrightarrow p \wedge \mathbf{EXEG}p & \mathbf{EG}p = gfp_Z\{p \cap \mathbf{EX}Z\} \end{array}$$

$$\ldots\ldots$$

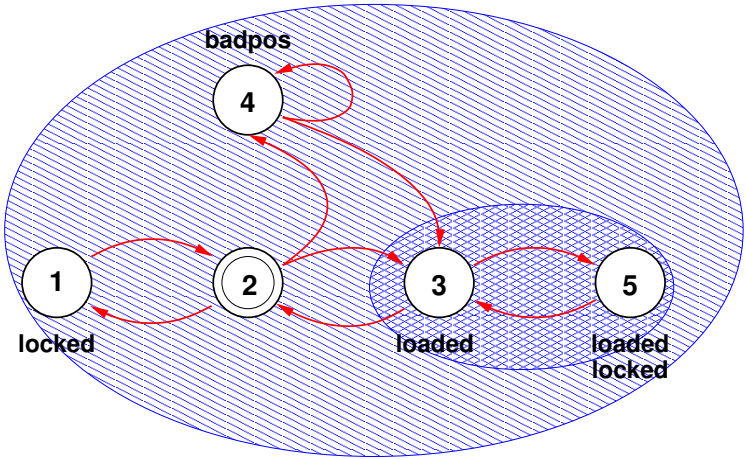
Fix-point characterization of CTL operators

- **EF** *loaded*



Fix-point characterization of CTL operators (II)

- **EG** *loaded*



Symbolic Representation

- A vector $\boldsymbol{x}$ of Boolean variables where each Boolean variable corresponds to an atomic proposition in P.

$$\boldsymbol{x} = \{locked, badpos, loaded\}$$

- A state s is represented with a formula $\xi(s)$ on the propositions:

$$① \rightsquigarrow locked \wedge \neg badpos \wedge \neg loaded$$
$$③ \rightsquigarrow \neg locked \wedge \neg badpos \wedge loaded$$
$$⑤ \rightsquigarrow locked \wedge \neg badpos \wedge loaded$$

- A set of states $Q \subseteq S$ represented symbolically as:

$$\xi(Q) = \bigvee_{s \in Q} \xi(s)$$

Symbolic Representation (II)

- A formula φ represents the set of states where the formula holds:

$$
\begin{aligned}
&\mathit{loaded} && \rightsquigarrow \{③,⑤\}\\
&\neg \mathit{badpos} && \rightsquigarrow \{①,②,③,⑤\}\\
&\mathit{loaded} \wedge \mathit{locked} && \rightsquigarrow \{⑤\}\\
&\mathit{badpos} \wedge \mathit{locked} \wedge \mathit{loaded} && \rightsquigarrow \emptyset\\
&\mathit{loaded} \vee \mathit{locked} && \rightsquigarrow \{①,③,⑤\}
\end{aligned}
$$

- Set theoretic operations map to propositional operations:

$$\xi(Q_1 \setminus Q_2) = \xi(Q_1) \wedge \neg\xi(Q_2) \qquad \xi(Q_1 \cup Q_2) = \xi(Q_1) \vee \xi(Q_2)$$
$$\xi(Q_1 \cap Q_2) = \xi(Q_1) \wedge \xi(Q_2)$$

Symbolic Representation (III)

- A new vector $\boldsymbol{x}'$ of Boolean variables to encode next state:

$$\boldsymbol{x}' = \{loaded', locked', badpos'\}$$

- A transition $t = \langle s_s, s_d \rangle$ encoded as

$$\xi(t) = \xi(\langle s_s, s_d \rangle) = \xi(s_s) \wedge \xi'(s_d)$$

$$\langle ①,② \rangle \rightsquigarrow \left(\begin{array}{ll} (locked \wedge \neg badpos \wedge \neg loaded) & \wedge \\ (\neg locked' \wedge \neg badpos' \wedge \neg loaded') & \end{array} \right)$$

$$\langle ②,③ \rangle \rightsquigarrow \left(\begin{array}{ll} (locked \wedge \neg badpos \wedge \neg loaded) & \wedge \\ (\neg locked' \wedge \neg badpos' \wedge loaded') & \end{array} \right)$$

- Transition relation T represented symbolically as:

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

Symbolic Exploration of the Model

- Images work on sets of states:
  - The forward image $FImg(S)$ of a set S is
    $$FImg(S) = \{s' \mid s \in S \wedge \langle s, s' \rangle \in T\} \quad \rightsquigarrow \quad \exists \boldsymbol{x}.(S(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{x}'))$$
  - The backward image $BImg(S)$ of a set S is
    $$BImg(S) = \{s \mid s' \in S \wedge \langle s, s' \rangle \in T\} \quad \rightsquigarrow \quad \exists \boldsymbol{x}'.(S(\boldsymbol{x}') \wedge T(\boldsymbol{x}, \boldsymbol{x}'))$$

Symbolic CTL model checking

- The set of states where a formula φ holds is represented symbolically.
- $M, S_0 \models \varphi$ reduces to $(\neg\varphi \wedge S_0) = \bot$
- Basic CTL operations represented as

$$\mathbf{EX}(\varphi) = \exists \boldsymbol{x}'.(T(\boldsymbol{x}, \boldsymbol{x}') \wedge \varphi(\boldsymbol{x}'))$$
$$\mathbf{AX}(\varphi) = \forall \boldsymbol{x}'.(T(\boldsymbol{x}, \boldsymbol{x}') \rightarrow \varphi(\boldsymbol{x}'))$$

- Fix-point computations as propositional transformations.

$$\mathbf{AF}\ \varphi = lfp_Z\{\varphi \cup \mathbf{AX}Z\}$$

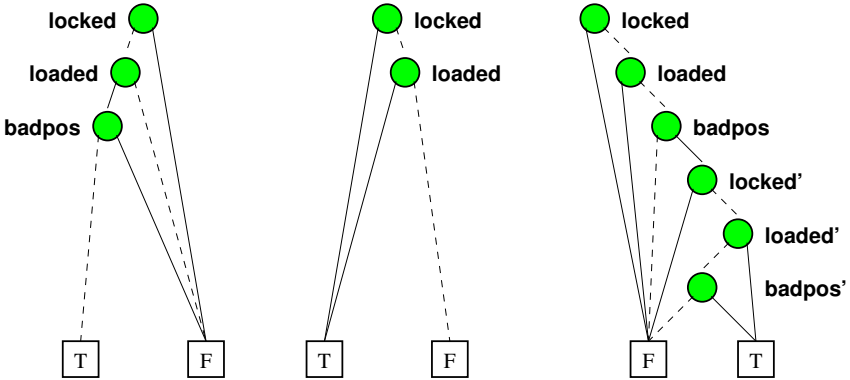
$$\begin{array}{lcl} s_0 & = & \bot \\ s_1 & = & \varphi \vee \mathbf{AX}s_0 = \varphi \\ s_2 & = & \varphi \vee \mathbf{AX}s_1 = \varphi \vee \mathbf{AX}\varphi \\ s_3 & = & \varphi \vee \mathbf{AX}s_2 = \varphi \vee \mathbf{AX}(\varphi \vee \mathbf{AX}\varphi) \\ & & \cdots \\ & & s_i = s_{i-1} \end{array}$$

Binary Decision Diagrams

- The problem in symbolic model checking:
  - The need for efficient and practical representation and manipulation of propositional formulae.
- Binary Decision Diagrams (BDDs).
  - Canonical form for propositional formulae.
  - Efficiently managed by BDD packages.

Binary Decision Diagrams (II)

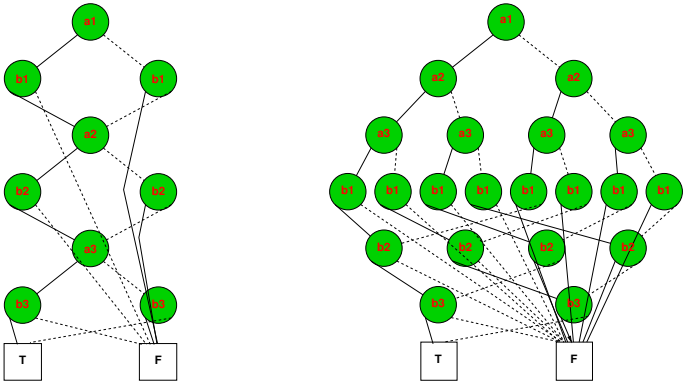
$\{③\}$ $\{①,③,⑤\}$ $\{\langle④,④\rangle,\langle④,③\rangle\}$



Binary Decision Diagrams (III)

- BDDs variable ordering is important.

$$(a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2) \wedge (a_3 \leftrightarrow b_3)$$



Binary Decision Diagrams: Pros and Cons

- Pros
  - Equality test of two BDDs in costant time.
  - Basic propositional operations polynomial on the size of operands.

$$O(\varphi_1 \otimes \varphi_2) = O(|\varphi_1||\varphi_2|)$$

- Cons
  - Quantification exponential in the variables being quantified.
  - BDD size may be exponential in the number of variables.
    - e.g., multiplier.

- They work well in several practical applications.

Introduction

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

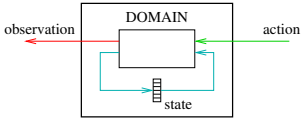
Lecture Overview

In this part of the lecture:

1. General framework
2. Reachability goals
3. Extended goals
4. Partial observability

Planning Domain: Formal Definitions

- A *planning domain* is defined as a tuple $\mathcal{D} = \langle S, S_0, A, T, P, L, O, X \rangle$:
  - S is a finite set of states.
  - S_0 is the initial set of states.
  - A is a finite set of actions.
  - $T \subseteq S \times A \times S$ is the transition relation.
  - P is a finite set of atomic propositions.
  - $L : P \mapsto 2^S$ assigns to each $p \in P$ the set of states where p holds.
  - O is a finite set of possible observations.
  - $X \subseteq S \times O$ is the observation relation.



Planning Domains: Symbolic Representation

- States represented as in symbolic model checking: $\xi(s)$.
- A vector $\boldsymbol{\alpha}$ of Boolean variables, each corresponding to an action in A:

$$\boldsymbol{\alpha} = \{GoWest, GoEast, GoNorth, GoSouth\}$$

- An action represented by assigning *True* to the corresponding Boolean variable:

$$\xi(\texttt{GoWest}) \rightsquigarrow GoWest \qquad \xi(\texttt{GoSouth}) \rightsquigarrow GoSouth$$

- Alternatively, compact logarithmic encoding of actions.
- A transition $t = \langle s_s, a, s_d \rangle$ encoded as

$$\xi(t) = \xi(\langle s_s, a, s_d \rangle) = \xi(s_s) \wedge \xi(a) \wedge \xi'(s_d)$$

Planning Domains: Symbolic Representation (II)

- Observations encoded with a vector $\boldsymbol{o}$ of Boolean variables.

$$\boldsymbol{o} = \{O_{WallN}, O_{WallS}, O_{WallE}, O_{WallO}\}$$

- An observation represented by assigning *True* to the corresponding Boolean variable.

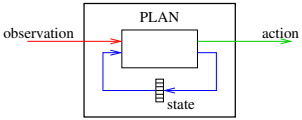
$$\xi(\mathtt{O_{WallW}}) \rightsquigarrow O_{WallW} \qquad \xi(\mathtt{O_{WallS}}) \rightsquigarrow O_{WallS}$$

Symbolic Exploration of the Domains

- Images work on set of states:
  - Forward: set of states reachable from a set of states S
    $$FImg(S) = \{s' \mid s \in S \wedge \langle s, a, s' \rangle \in T\} \quad \rightsquigarrow \quad \exists \boldsymbol{x}.\boldsymbol{\alpha}.(S(\boldsymbol{x}) \wedge T(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{x}'))$$
  - Backward: set of states from which a set of states S is reachable
    $$BImg(S) = \{s \mid s' \in S \wedge \langle s, a, s' \rangle \in T\} \quad \rightsquigarrow \quad \exists \boldsymbol{x}'.\boldsymbol{\alpha}.(S(\boldsymbol{x}') \wedge T(\boldsymbol{x}, \boldsymbol{\alpha}, \boldsymbol{x}'))$$

Plans: Formal Model

- A *plan* Π for a planning domain $\mathcal{D}$ is a tuple $\Pi = \langle \Sigma, \sigma_0, \Psi, \Upsilon \rangle$ where:
  - Σ is a finite set of *plan states*.
  - σ_0 is the *initial* plan state.
  - $\Psi : \Sigma \times O \longmapsto A$ associates to a pair $\langle \sigma, o \rangle$ an action α to execute.
  - $\Upsilon : \Sigma \times O \longmapsto \Sigma$ associates to a pair $\langle \sigma, o \rangle$ a new plan state σ'.



- A configuration for a domain $\mathcal{D}$ and a plan Π is a pair $\langle s, \sigma \rangle \in S \times \Sigma$.
- Plan execution is represented with configuration transitions:

$$\langle s, \sigma \rangle \rightarrow \langle s', \sigma' \rangle$$

Motivations

Reachability goals:

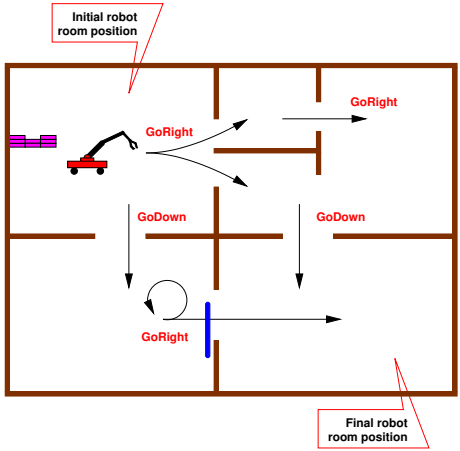
- First and "basic" planning problem:
  - find a plan to achieve a desired final situation.
- When tackled within classical planning:
  - solutions are sequences of actions.
  - solutions are guaranteed to achieve the goal.

Adding non-determinism:

- Plans as sequences of actions are bound to failure.
  - Initial situation might be uncertain.
  - Actions can have non-deterministic effects.
- Plans of different strength, e.g. weak, strong.
- Plans of different structure: sense current state and execute an action.

Example

- The problem . . .
  - . . . find a plan from a (set of) initial state(s) to a goal set of states.



Aim

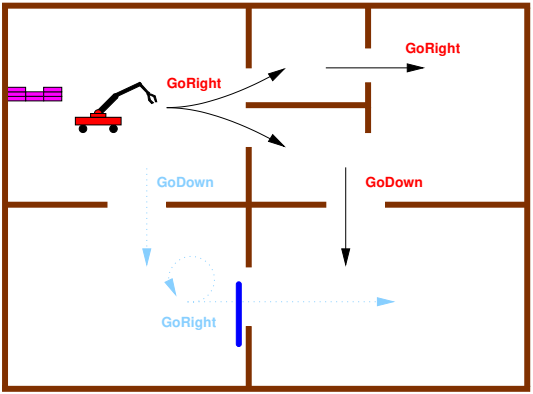
- Objectives:
  - Allow for planning under *full observability* in *non-deterministic* domains.
  - *Synthesis* of plans of different *strength*.
  - Deal *in practice* with *large size domains*.
- Problems:
  - How can we handle non-deterministic actions?
  - Which kind of plans must be generated?
  - Which planning algorithms?
  - How can planning algorithms handle large domains?

The PMC approach for Reachability goals

- Solution plans encode conditional behavior based on domain state.
- Planning algorithms exploit . . .
  - symbolic representation of the domain.
  - symbolic representation of solution plans.

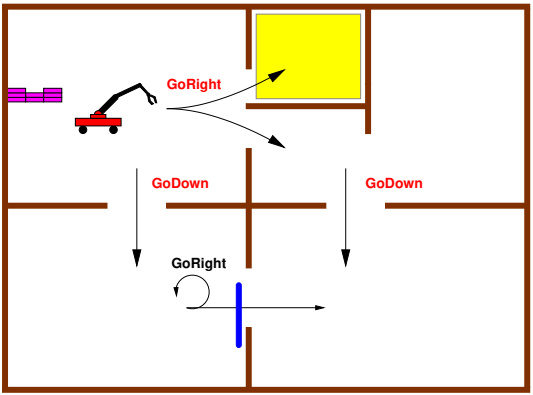
Example

- Strong Solutions: plans that are guaranteed to reach the goal.
  - All executions traces are acyclic and reach the goal.



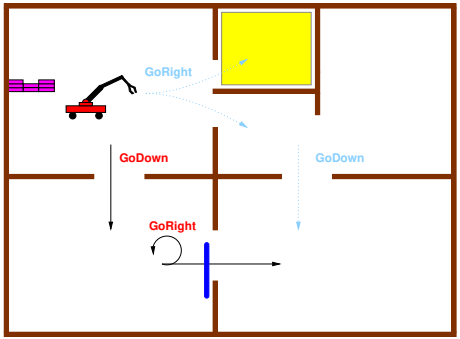
Example

- Weak Solutions: plans that may achieve the goal.
  - At least one execution trace reaches the goal.



Example

- Strong Cyclic Solutions: trial and error strategies.
  - The goal is reachable from all states of the execution traces.
  - Solutions are guaranteed to reach the goal under the assumption of "*fair*" execution.



State-Action Tables

- Solutions are memory-less plans that map states to actions to execute.
- Such solutions can be represented as "state-action tables" (SA).

State	Action
R1	GoUp
R2	GoRight
R3	GoDown
R4	GoRight

- State-action tables map in a general plan. . .
  - . . . without plan states.
  - . . . switching on the current state.

Symbolic Representation of SA Tables

- A state-action pair $p = \langle s, a \rangle$ is represented symbolically as:

$$\xi(p) = \xi(s) \wedge \xi(a)$$

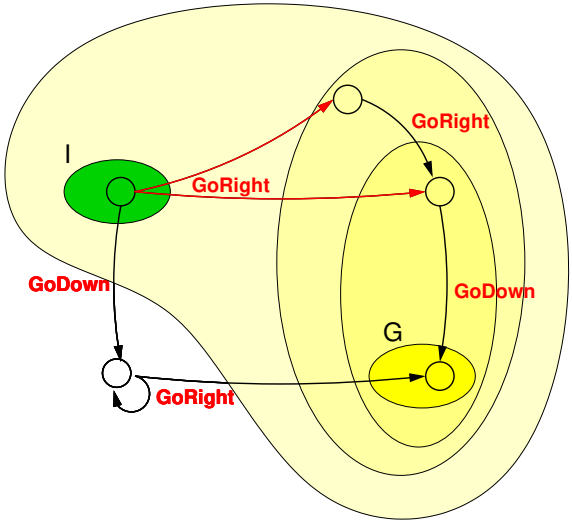
- A state-action table π is encoded symbolically as:

$$\xi(\pi) = \bigvee_{p \in \pi} \xi(p)$$

- Set operations on state-action tables performed as:

$$\xi(\pi_1 \setminus \pi_2) = \xi(\pi_1) \wedge \neg\xi(\pi_2) \qquad \xi(\pi_1 \cup \pi_2) = \xi(\pi_1) \vee \xi(\pi_2)$$
$$\xi(\pi_1 \cap \pi_2) = \xi(\pi_1) \wedge \xi(\pi_2)$$

StrongPlan: Algorithm intuition



StrongPlan: Algorithm

```
function StrongPlan (I, G);
  OSA := Fail; SA := ∅;
  while (OSA ≠ SA) do
    PImg := SPImg(G ∪ StOf(SA));
    NSA := PruneStates(PImg, G ∪ StOf(SA));
    OSA := SA;
    SA := SA ∪ NSA;
  done;
  if (I ⊆ (G ∪ StOf(SA))) then
    return SA;
  else
    return Fail;
end;
```

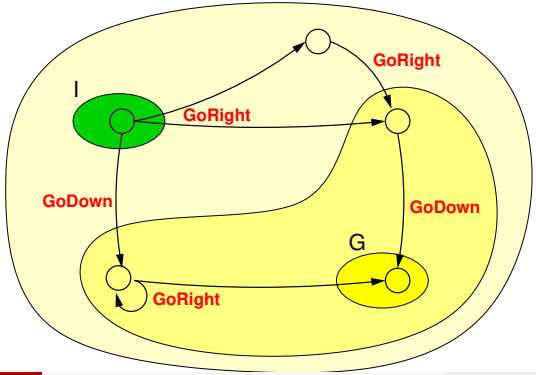
```
function AF (I, G);
  OS := Fail; S := ∅;
  while (OldS ≠ S) do
    PImg := AX(G ∪ S);

    OS := S;
    S := S ∪ PImg;
  done;
  if (I ⊆ (G ∪ S)) then
    return Ok;
  else
    return Fail;
end;
```

StrongPlan: properties

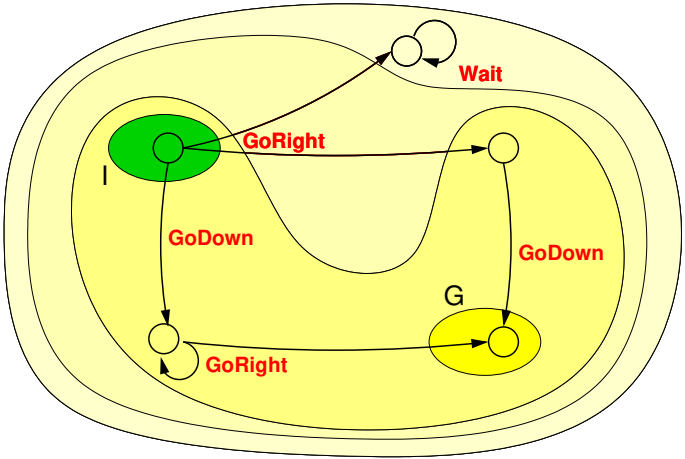
- The algorithm terminates.
- The algorithm is correct and complete.
  - Whenever a strong solution exists it finds a state-action table that is a strong solution.
  - Whenever a strong solution does not exist Fail is returned.
- The algorithm computes “optimal” solutions.
  - Optimality defined on the length of the plan execution traces.

WeakPlan: algorithm intuitions

- WeakPlan is similar to the StrongPlan algorithm.
  - It performs a *weak pre-image* as basic step.
  - Weak pre-image of a set S computes those state-action pairs whose execution may lead to a state $s \in S$.
- WeakPlan similar to the SMC algorithm to compute **EF**.



StrongCyclicPlan: algorithm intuition



StrongCyclicPlan: algorithm

function *StrongCyclicPlan* (I, G)*;*
  $OSA := \emptyset$*;*
  $SA :=$ *UnivSA;*
  while $(OSA \neq SA)$ ***do***
    $OSA := SA$*;*
    $SA :=$ *PruneUnconnected*$($*PruneOutgoing*$(SA, G), G)$*;*
  done*;*
  if $(I \subseteq (G \cup \mathit{StOf}(SA)))$ ***then***
    return *RemoveNonProgress*(SA, G)*;*
  else
    return *Fail;*
end*;*

StrongCyclicPlan vs AG EF

- Similarities:
  - Similar meaning (i.e., from all reached states it is possible to achieve the goal).
  - Plan validation performed by verifying **AG EF**p on the synchronous product of domain and plan.
- Differences:
  - The computation of **AG EF** is performed with two *nested distinct* fix point calculations.
  - StrongCyclicPlan *interleaves* the steps of the two fix point calculations.

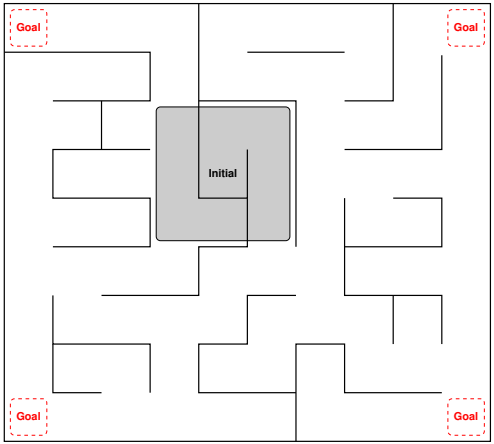
StrongCyclicPlan: properties

- The algorithm terminates.
- The algorithm is correct and complete.
  - Whenever a strong cyclic solution exists it finds a state-action table that is a strong cyclic solution.
  - Whenever a strong cyclic solution does not exist Fail is returned.

Motivations

- Realistic:
  - Environment is seldomly fully observable.
  - Some form of sensing is often available.
- General: full observability, conformancy are special cases.
- Related to relevant problems:
  - Diagnosis.
  - Homing/distinguishing problem.
  - Game theory.

Example: a robot-world with sensing



Aim

Objectives:

- Allow for planning under *partial observability*...
- in *non-deterministic domains*.
- Deal *in practice* with domains of *large size*.

Problems:

- How can we express partial observability?
- Which kind of plans must be generated?
- Planning algorithms?
- How can planning algorithms deal with large domains?

The PMC approach for PO

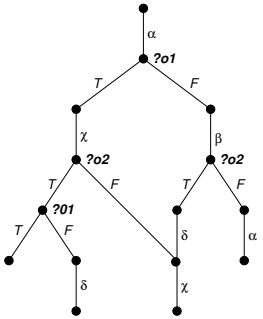
- Partial observability expressed by formulae that encode observation relation.
- Plans encode conditional behavior based on sensing.
- Planning algorithms exploit symbolic representation of observations (as well as actions and states).
- Experimental results show that algorithms work in practice.

Goals

- Due to nondeterminism and partial observability, the controller has a partial knowledge of the state.
- We consider *strong reachability* goals:
  - At the end of plan execution, the executor *knows* that the goal has been reached...
  - ...in spite of possibly not knowing exactly the state.

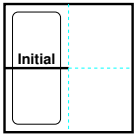
Plans

- Structure of plans:
  - In general: conditional over observations
  - Conformant case: sequences
  - Full observability case: conditional over states
- Expressed as general FSM:
  - Plan state: “plan program counter”.
  - Branching conditions on:
    - “plan program counter”.
    - observations.
- Validated by model-checking.

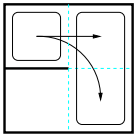


Plan search: Belief States

- Due to nondeterminism and partial observability, status known to controller can be uncertain:
  - ...because the initial situation is uncertain...



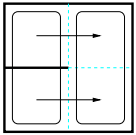
  - ... or because actions may have unpredictable results.



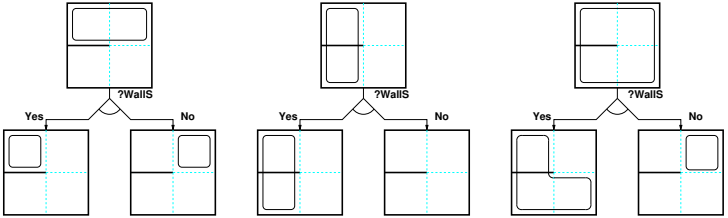
Thus, planning proceeds over sets of states: *belief states (BS)*

Plan Search: base steps

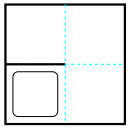
- An action transforms a BS into a BS

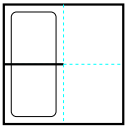


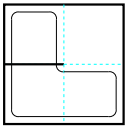
- Observing *may* split a BS

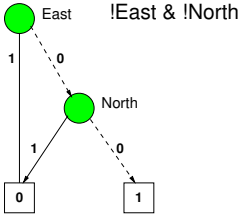


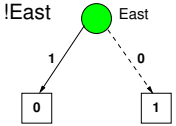
BDD representation of BS

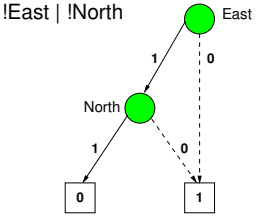




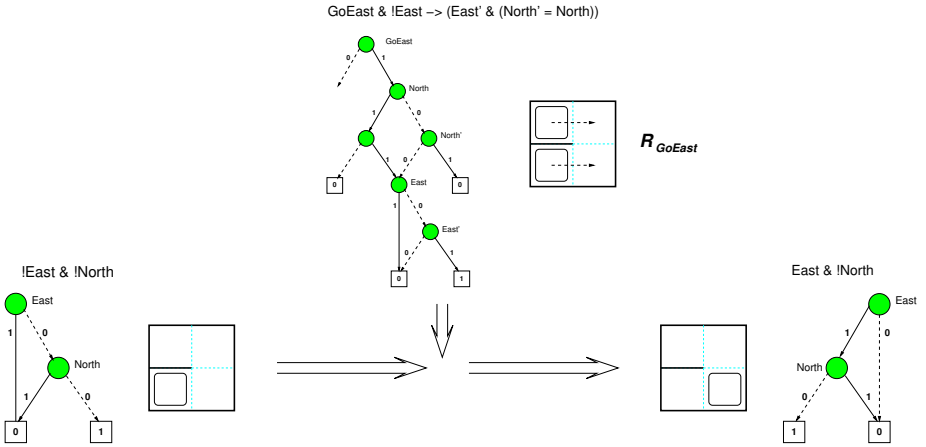




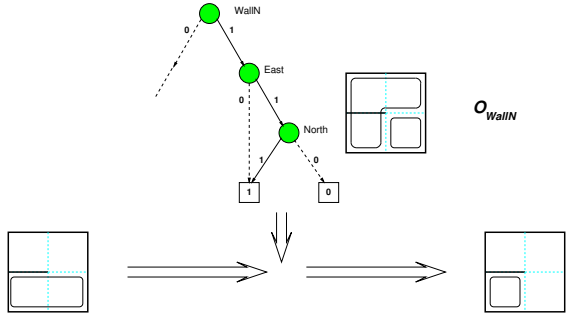




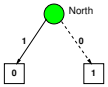
BDD transformation of BS: acting



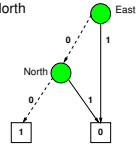
BDD transformation of BS: observing



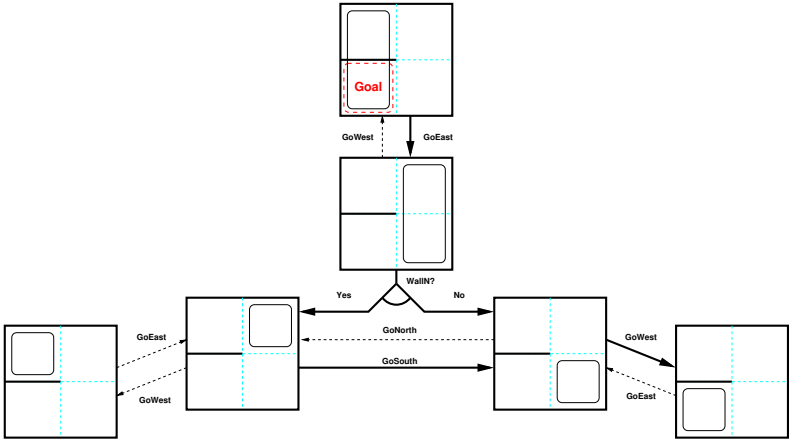
!North



!East & !North

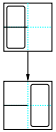


Search space

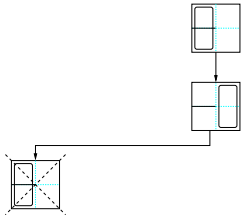


Search algorithm: a DFS search

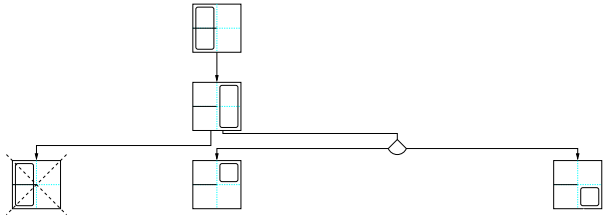
Search algorithm: a DFS search



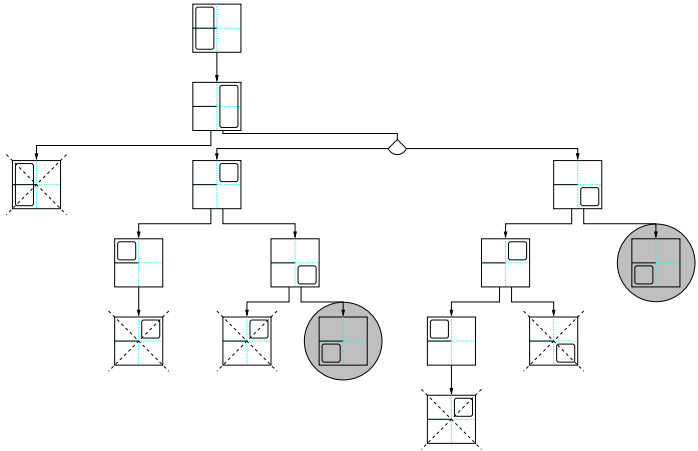
Search algorithm: a DFS search



Search algorithm: a DFS search



Search algorithm: a DFS search



Heuristic search

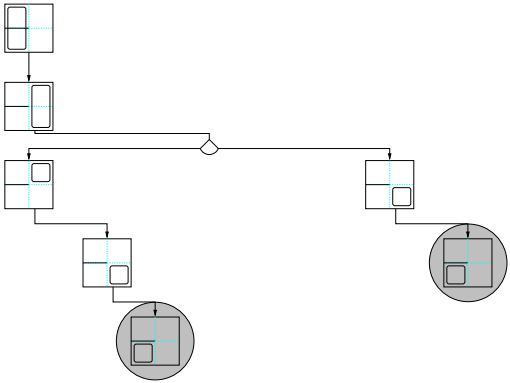
Motivations:

- DFS may produce dumb plans.
- DFS may be not efficient.
- Even simple user-provided heuristics may help a lot.
- Heuristics may be extracted from domain/problem.

Example:

- Favouring observations.
- Avoiding “stepping back”.

Search algorithm: a Heuristic search



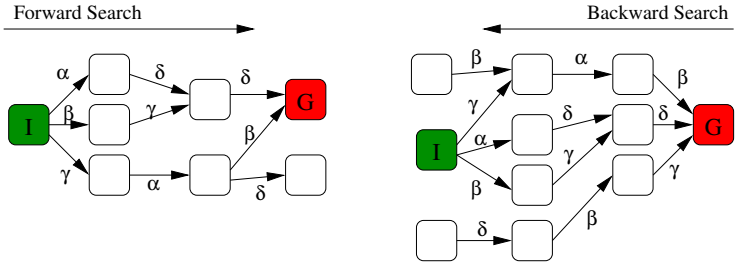
Conformant algorithms

Motivations:

- Relevant applications (e.g. reset sequences).
- Ad-hoc highly efficient planning algorithms.
- Dedicated heuristics and fwd/bwd algorithms, e.g.:
  - breadth-first search in belief space
  - knowledge-driven search
  - ...

Conformant algorithms (II)

- Search space only features “or” branching.
- BFS search in belief space is feasible...
- ...but it’s not the only way.



Properties of algorithms

- The algorithms terminate.
- The algorithms are correct and complete:
  - Whenever a strong solution exists they find a plan which is a strong solution.
  - Whenever a strong solution does not exist they return Fail.

Combining Extended Goals and Partial Observability

- Realistic domains (robot navigation, embedded controllers) combine *partial observability* and *extended goals*.

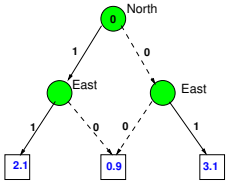
- Challenging problem:
  - knowledge goals: introducing "*knows*" in the goal language.
  - plan synthesis requires combining plan contexts and belief states.
  - no trivial combination of existing PO and EG algorithms.
- Work done so far:
  - a language for expressing temporally-extended knowledge goals:
    - example: $\mathrm{AG}\, p \wedge \mathrm{AF}\, \mathbf{K}\, q$
  - a framework for the validation of plans:
    - a monitor is used for epistemic tracing to trace the belief state.

Planning in "adversarial" environment

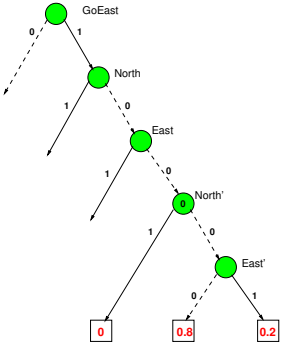
- So far we have assumed a "fair" environment:
  - all the action outcomes have some chance to occur.
- In several applications:
  - the non-determinism is due to uncontrollable, adversarial agents:
    - we cannot assume that they will execute actions in a fair way.
    - they can apply a strategy to prevent goal achievement.
  - the non-determinism is due to lack of information:
    - some action outcomes may never happen in the "real" domain.
- State of the art:
  - some forms of *adversarial* planning as model checking addressed by Jensen and Veloso [6]
  - partial observability and extended goals are still open
  - non-deterministic effects, partial observability and time are still also open

Planning with probabilities

- ADDs (Algebraic Decision Diagrams) can be used for representing:
  - expected rewards:



  - probabilities of outcomes:



- In SPUDD [5], ADDs are exploited for efficient value iteration in MDP planning.

Wrap-up

Planning via Model Checking:

- A single, *well founded framework* for a variety of planning problems.
- Several specialized algorithms available for plan synthesis.
- Works in practice.
- Plan validation "for free": Model Checking of plan against domain.

References I

[1] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013. ISBN 9781608459698. doi: 10.2200/S00513ED1V01Y201306AIM022. URL https://doi.org/10.2200/S00513ED1V01Y201306AIM022.

[2] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004. ISBN 978-1-55860-856-6.

[3] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016. ISBN 978-1-107-03727-4. URL http://www.cambridge.org/de/academic/subjects/computer-science/artificial-intelligence-and-natural-language-processing/automated-planning-and-acting?format=HB.

[4] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2019. doi: 10.2200/S00900ED2V01Y201902AIM042. URL https://doi.org/10.2200/S00900ED2V01Y201902AIM042.

[5] Jesse Hoey, Robert St-Aubin, Alan J. Hu, and Craig Boutilier. SPUDD: stochastic planning using decision diagrams. In Kathryn B. Laskey and Henri Prade, editors, *UAI '99: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, July 30 - August 1, 1999*, pages 279–288. Morgan Kaufmann, 1999. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=178&proceeding_id=15. 87

[6] Rune M. Jensen and Manuela M. Veloso. Obdd-based universal planning for synchronized agents in non-deterministic domains. *J. Artif. Intell. Res.*, 13:189–226, 2000. doi: 10.1613/jair.649. URL https://doi.org/10.1613/jair.649. 86