Principles of AI Planning 17. Strong cyclic planning

Albert-Ludwigs-Universität Freiburg

Bernhard Nebel and Robert Mattmüller February 5th, 2018





Strong cyclic plans

Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

Summary

Strong cyclic plans

- The simplest objective for nondeterministic planning is the one we have considered in the previous lecture: reach a goal state with certainty.
- With this objective the nondeterminism can also be understood as an opponent like in 2-player games.
 The plan guarantees reaching a goal state no matter what the opponent does: plans are winning strategies.

Strong cyclic plans

Motivation

DRD

Nested Fixpoint Algorithm

> icremental Ianning Algorithm

Maintenance

Planning objectives Limitations of strong plans

- In strong plans, goal states can be reached without visiting any state twice.
- This property guarantees that the length of executions is bounded by some constant (which is smaller than the number of states.)
- Some solvable problems are not solvable this way.
 - Action may fail to have any effect. Hit a coconut to break it.
 - Action may fail and take us away from the goals.
 Build a house of cards.

Consequences:

- 1 It is impossible to avoid visiting some states several times.
- 2 There is no finite upper bound on execution length.

Motivation

Nested Fixpoint Algorithm

BURG

Incremental Planning Algorithm

Maintenance

Planning objectives

When strong cyclic plans make sense

Fairness assumption

For any nondeterministic operator $\langle \chi, \{e_1, \dots, e_n\} \rangle$, the "probability" of every effect e_i , $i = 1, \dots, n$, is greater than 0.

Alternatively: For each $s' \in img_o(s)$ the "probability" of reaching s' from s by o is greater than 0.

This assumption guarantees that a strong cyclic plan reaches the goal almost certainly (with probability 1).

This is not compatible with viewing nondeterminism as an opponent in a 2-player game: the opponent's strategy might rule out some of the choices e_1, \ldots, e_n .

Strong ov

plans

Motivation

Nested Fixpoint Algorithm

> ncremental Planning Algorithm

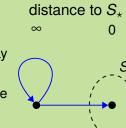
Maintenance

Need for strong cyclic plans Example

Example (Breaking a coconut)

- Initial state: coconut is intact.
- Goal state: coconut is broken.
- On every hit the coconut may or may not break.
- There is no finite upper bound on the number of hits.

This is equivalent to coin tossing.



Strong cyclic

plans

Motivation

Nested Fixpoint Algorithm

> ncremental Planning Algorithn

Maintenance

Need for strong cyclic plans Example

Example (Build a house of cards)

- Initial state: all cards lie on the table.
- Goal state: house of cards is complete.
- At every construction step the house may collapse.



distance to S_{+}

UNI FREIBUR

Strong cyclic plans

Motivation

Nested Fixpoint Algorithm

> Incremental Planning Algorithn

Maintenance

Algorithms for strong cyclic planning

We present two algorithms for strong cyclic planning:

- The nested fixpoint algorithm is conceptually simpler, but typically very costly, especially if not implemented symbolically.
 - Historically older
 - Uninformed
 - Considers entire state space
- The determinization-based incremental planning algorithm is a bit more complicated, but typically more efficient.
 - Historically newer, state of the art
 - Can use informed classical planner as sub-procedure
 - Often only considers small portion of state space

Strong cyclic

plans

2

Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

- Finds plans that may loop (strong cyclic plans).
- The algorithm is rather tricky in comparison to the algorithm for strong plans.
- Every state covered by a plan satisfies two properties:
 - The state is good: there is at least one execution (= path in the graph defined by the plan) leading to a goal state.
 - 2 Every successor state is either a goal state or good.
- The algorithm repeatedly eliminates states that are not good.

Strong cyclic plans

Motivation

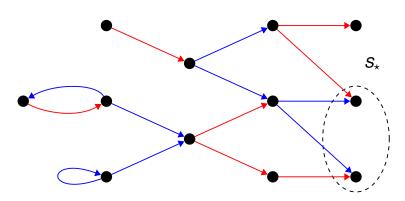
BURG

Nested Fixpoint Algorithm

> ncremental Planning Algorithm

Maintenance





Strong cyclic plans

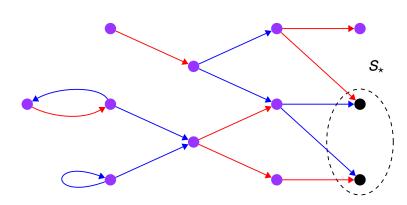
Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

All states are candidates for being good.



UNI FREIBURG

Strong cyclic plans

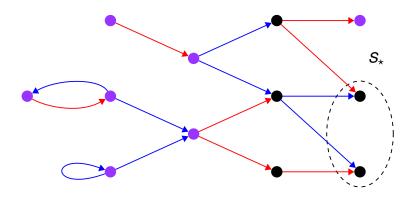
Motivation

Nested Fixpoint Algorithm

> Incremental Planning Algorithm

Maintenance

States from which goals are reachable in \leq 1 steps so that all immediate successors are possibly good.



Strong cycli

BURG

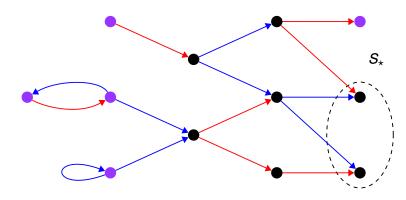
Motivation

Nested Fixpoint Algorithm

> Incremental Planning Algorithm

Maintenance

States from which goals are reachable in \leq 2 steps so that all immediate successors are possibly good.



LUN FREIBURG

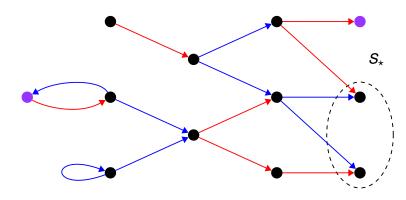
Motivation

Nested Fixpoint Algorithm

> Incremental Planning Algorithm

Maintenance

States from which goals are reachable in \leq 3 steps so that all immediate successors are possibly good.



Strong cycli

plans

BURG

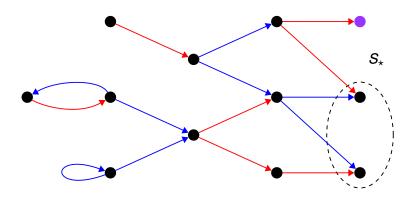
Motivation

Nested Fixpoint Algorithm

> Incremental Planning Algorithm

Maintenance

States from which goals are reachable in \leq 4 steps so that all immediate successors are possibly good.



FREIBURG

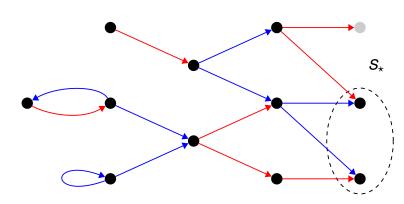
. Motivation

Nested Fixpoint Algorithm

> Incremental Planning Algorithm

Maintenance

Eliminate states that turned out not to be good.



UNI FREIBURG

Strong cyclic plans

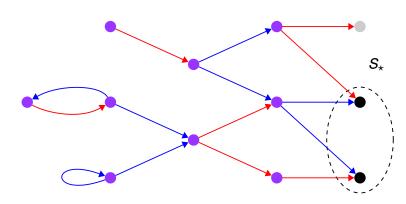
Motivation

Nested Fixpoint Algorithm

> Incremental Planning Algorithm

Maintenance

The set of possibly good states is now smaller.



UNI FREIBURG

Strong cyclic plans

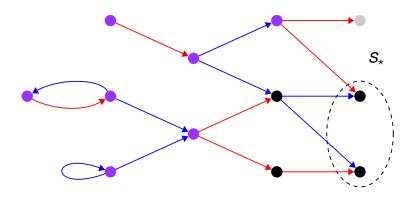
Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

States from which goals are reachable in \leq 1 steps so that all immediate successors are possibly good.



Strong cyclic

Motivation

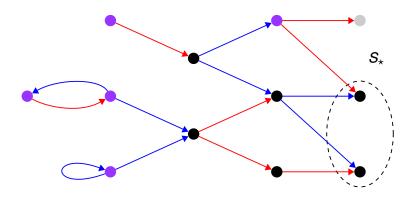
BURG

Nested Fixpoint Algorithm

Incremental Planning Algorithn

Maintenance

States from which goals are reachable in \leq 2 steps so that all immediate successors are possibly good.



Strong cyclic

Motivation

BURG

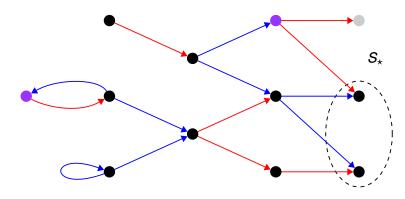
ZW

Nested Fixpoint Algorithm

> Incremental Planning Algorithm

Maintenance

States from which goals are reachable in \leq 3 steps so that all immediate successors are possibly good.



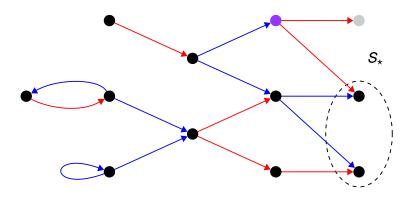
Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

States from which goals are reachable in \leq 4 steps so that all immediate successors are possibly good.



Strong cvg

BURG

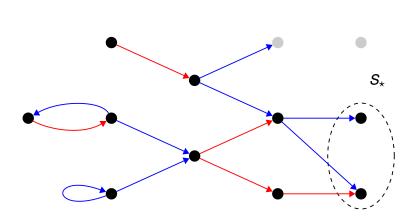
Motivation

Nested Fixpoint Algorithm

> Incremental Planning Algorithm

Maintenance

Eliminate states that turned out not to be good.



FREIBURG

Strong cyclic plans

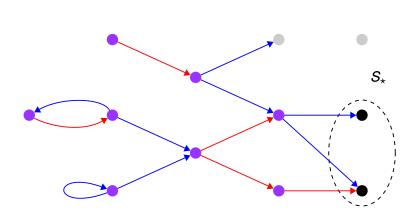
Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

The set of possibly good states is now smaller.



FREIBURG

Strong cyclic plans

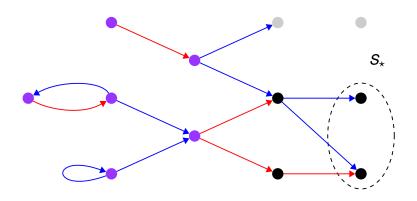
Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

States from which goals are reachable in \leq 1 steps so that all immediate successors are possibly good.



Strong cyclic

Motivation

BURG

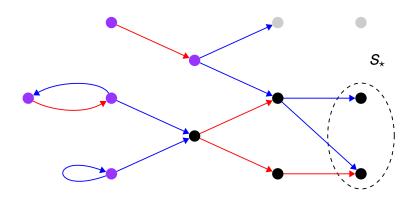
ZW

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

States from which goals are reachable in \leq 2 steps so that all immediate successors are possibly good.



Strong cyclic

Motivation

BURG

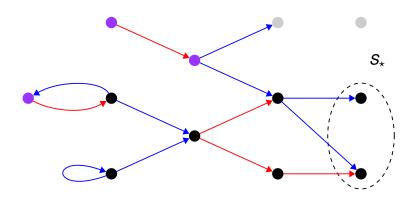
ZW

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

States from which goals are reachable in \leq 3 steps so that all immediate successors are possibly good.



Strong cyclic

Motivation

BURG

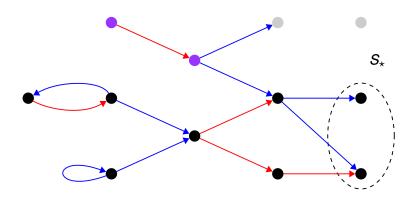
ZW

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

States from which goals are reachable in \leq 4 steps so that all immediate successors are possibly good.



Strong cyclic

Motivation

BURG

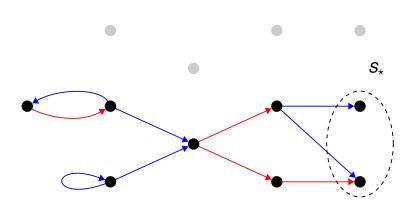
ZW

Nested Fixpoint Algorithm

> Incremental Planning Algorithm

Maintenance

Remaining states are all good. A further iteration would not eliminate more states.



Strong cycli

plans

URG

æ

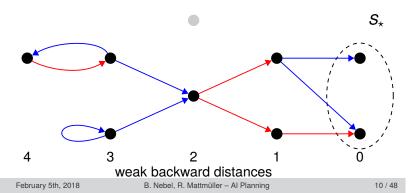
Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

Assign each state an operator so that the successor states are goal states or good, and some of them are closer to goal states. Use weak distances computed with weak preimages. For this example this is trivial.



Strong cyclic

Motivation

DRD

> Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

Recall the definition of cyclic strong plans:

Definition (strong cyclic plan)

Let *S* be the set of states of a planning task Π . Then a strong cyclic plan for Π is a function $\pi : S_{\pi} \to O$ for some subset $S_{\pi} \subseteq S$ such that

- $\pi(s)$ is applicable in s for all $s \in S_{\pi}$,
- $S_{\pi}(s_0) \subseteq S_{\pi} \cup S_{\star}$ (π is closed), and
- $\blacksquare S_{\pi}(s') \cap S_{\star} \neq \emptyset \text{ for all } s' \in S_{\pi}(s_0) \ (\pi \text{ is proper}).$

FREB

Strong cyclic plans

Motivation

Nested Fixpoint Algorithm

> incremental Planning Algorithm

Maintenance

Procedure prune

- The procedure prune finds a maximal set of states for which reaching goals with looping is possible.
- It consists of two nested loops:
 - The outer loop iterates through i = 0, 1, 2, ... and produces a shrinking sequence of candidate good state sets $C_0, C_1, ..., C_n$ until $C_i = C_{i+1}$.
 - **2** The inner loop identifies growing sets W_j of states from which a goal state can be reached with *j* steps without leaving the current set of candidate good states C_j . The union of all W_0, W_1, \ldots will be C_{j+1} .

Strong cyclic plans

Motivation

> Nested Fixpoint Algorithm

> > Incremental Planning Algorithm

Maintenance

Procedure prune

 $\begin{array}{l} \text{def prune}(S,\,O,\,S_{\star}):\\ C_{0}:=S\\ \text{for each }i\in\mathbb{N}_{1}:\\ W_{0}:=S_{\star}\\ \text{for each }j\in\mathbb{N}_{1}:\\ W_{j}:=W_{j-1}\cup\bigcup_{o\in O}(wpreimg_{o}(W_{j-1})\cap spreimg_{o}(C_{i-1}))\\ \text{ if }W_{j}=W_{j-1}:\\ \text{ break}\\ C_{i}:=W_{j}\\ \text{ if }C_{i}=C_{i-1}:\\ \text{ return }\langle C_{i},\langle W_{0},\ldots,W_{i-1}\rangle\rangle \end{array}$



Strong cyclic plans

Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithn

Maintenance

Lemma (Procedure prune)

Let *S* and $S_* \subseteq S$ be sets of states and *O* a set of operators. Then prune(*S*, *O*, *S*_{*}) terminates after a finite number of steps and returns $C \subseteq S$ such that there is a strategy $\pi : C \setminus S_* \to O$ that is a strong cyclic plan (for the states for which it is defined) and maximal in the sense that there is no set $C' \supseteq C$ and a strong cyclic plan $\pi' : C' \setminus S_* \to O$.

The sets W_j also returned by *prune* encode weak distances and can be used to define the strong cyclic plan π.

FREIBUR

Strong cyclic plans

Motivation

Nested Fixpoint Algorithm

> Incremental Planning Algorithm

Maintenance

Nested Fixpoint Algorithm

Main algorithm

The planning algorithm

def strong-cyclic-plan($\langle V, I, O, \gamma \rangle$): S := set of states over V $S_{\star} := \{s \in S \mid s \models \gamma\}$ $\langle C, (W_j)_{j=0,1,2,...} \rangle = prune(S, O, S_{\star})$ if $I \notin C$: return no solution

 $\begin{array}{l} \text{for each } s \in C: \\ \delta(s) \coloneqq \min\{j \in \mathbb{N}_0 \mid s \in W_j\} \\ \text{for each } s \in C \setminus S_\star: \\ \pi(s) \coloneqq \text{some operator } o \in O \text{ with } img_o(s) \subseteq C \\ \text{ and } \min\{\delta(s') \mid s' \in img_o(s)\} < \delta(s) \end{array}$

return π

Strong cyclic plans

Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithn

Maintenance

- The procedure prune runs in polynomial time in the number of states because the number of iterations of each loop is at most n – hence there are O(n²) iterations – and computation on each iteration takes polynomial time in the number of states.
- Finding strong cyclic plans for full observability is in the complexity class EXPTIME.
- The problem is also EXPTIME-hard.
- Similar to strong planning, we can speed up the algorithm in many practical cases by using a symbolic implementation (e.g. with BDDs).

Strong cyclic plans

Motivation

DRG

Nested Fixpoint Algorithm

> ncremental Planning Algorithm

Maintenance

Idea [Kuter/Nau/Reisner/Goldman, 2008; Fu/Ng/Bastiani/Yen, 2011]:

- Pretend the planning task was deterministic: Turn each action *o* = ⟨*χ*,*E*⟩ with *E* = {*e*₁,...,*e*_n} into *n* actions *o_i* = ⟨*χ*,*e_i*⟩ for *i* = 1,...,*n*. Obtain classical problem Π'.
- 2. Find classical plan *P* in Π' . Add state-action mapping corresponding to *P* to π .
- 3. For each operator o_i used in P (in state s), identify original nondeterministic operator o and states $S' = img_o(s)$.
- 4. For each "open" state $s' \in S'$, go to 2.

Remark: May require backtracking, if some state used in a classical plan turns out not to admit a strong cyclic plan.

Strong cyclic

Motivation

2

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

Definition (all-outcomes determinization)

Let $\Pi = \langle V, I, O, \gamma \rangle$ be a nondeterministic planning task. The all-outcomes determinization of Π is the deterministic planning task $\Pi_{det} = \langle V, I, O_{det}, \gamma \rangle$, where $O_{det} = \bigcup_{o \in O} o_{det}$, and $\langle \chi, E \rangle_{det} = \{ \langle \chi, e \rangle \mid e \in E \}.$

Strong cyclic plans

Motivation

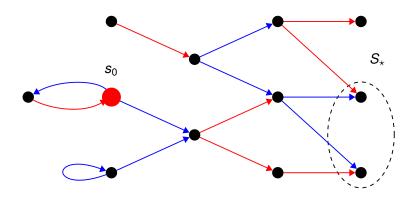
2

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

List of states to solve: $\{s_0\}$



UNI FREIBURG

Strong cyclic plans

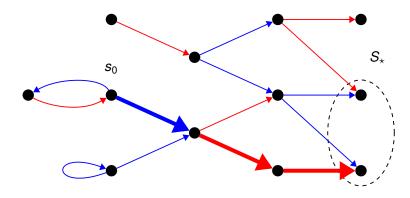
Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

Plan for s_0 in determinization: $blue_2$, red_2 , red_2



FREIBURG

Strong cyclic plans

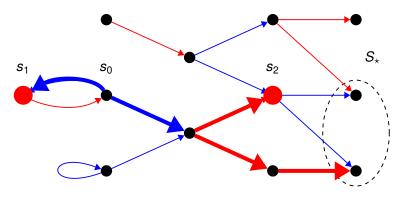
Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

"Undesired" outcomes of $blue_1$ and red_1 lead to new list of states to solve: $\{s_1, s_2\}$



Strong cycli

plans

BURG

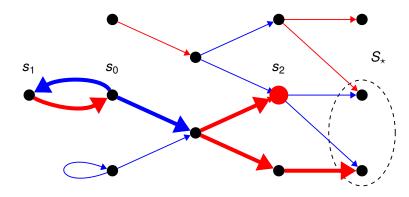
Motivation

Nested Fixpoint Algorithm

> Incremental Planning Algorithm

Maintenance

Plan for s_1 in determinization: red, $blue_2$, red_2 , red_2



FREIBURG

Strong cyclic plans

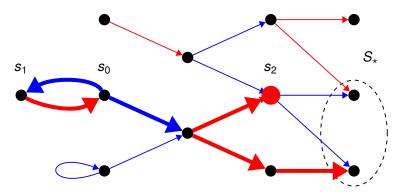
Motivation

Nested Fixpoint Algorithm

> Incremental Planning Algorithm

Maintenance

No new "undesired" outcomes. List of states to solve: $\{s_2\}$



UNI FREIBURG

Strong cyclic plans

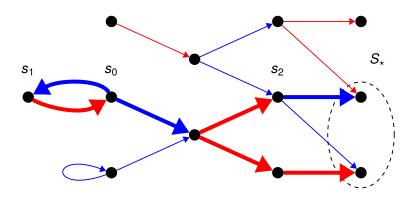
Motivation

Nested Fixpoint Algorithm

> Incremental Planning Algorithm

Maintenance

Plan for s₂ in determinization: blue₁



Strong cyclic plans

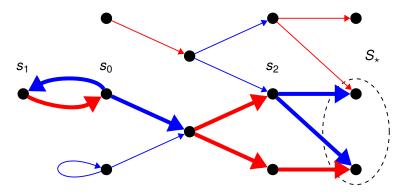
Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

"Undesired" outcome of $blue_2$ in s_2 leads to goal state, too. List of states to solve: \emptyset . Strong cyclic plan found.



Strong cyclic

Motivation

DRD

Ē

Nested Fixpoint Algorithm

> Incremental Planning Algorithm

Maintenance

Pseudocode

Procedure incremental-strong-cyclic-plan

```
def incremental-strong-cyclic-plan(\langle V, I, O, \gamma \rangle):
   \pi \leftarrow \emptyset; fail \leftarrow \{I\}
    while fail \neq \emptyset:
          s ← SelectAndRemoveFrom(fail)
          \pi' \leftarrow \mathsf{DetSearch}(\langle V, s, O_{\det}, \gamma \rangle)
          if \pi' = \text{FAILUBE}:
                  if s = /: return FAILUBE
                  else: Backtrack(s, \pi, \langle V, I, O, \gamma \rangle)
          else:
                  \pi \leftarrow \pi \cup \pi'
           fail \leftarrow {s \in S | s nongoal state reachable from I
                                 following \pi, but \pi(s) undefined
   return \pi
```

DRG

æ

Incremental Planning Algorithm

Maintenance

If a deterministic search fails, the state *s* from which it started cannot be part of a strong cyclic plan.

- If s = l, the whole given planning problem is unsolvable and the algorithm returns FAILURE.
- Otherwise, state *s*, which has already been added to the constructed policy π , has to be removed from π , and the algorithm has to ensure that *s* will never be reconsidered again. This is accomplished by the procedure BACKTRACK.

Strong cyclic plans

Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

Procedure backtrack

def backtrack($s, \pi, \langle V, I, O, \gamma \rangle$): update π by deleting all entries that would immediately lead to s, i.e. $\pi \leftarrow \pi \setminus \{(s', \pi(s')) | s \in img_{\pi(s')}(s')\}$ add all states s' removed from π to the set of fail-states fail permanently mark all formerly assigned actions $\pi(s')$ removed from π at s' as inapplicable in s' to avoid running into the same dead end again.

Strong cyclic plans

Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

- Iteratively solves all-outcomes determinizations of Π with "fail-states" as initial states.
- Planner can choose desired outcome of each action.
- Deterministic plans are added to policy under construction.
- Corresponding undesired outcomes have to be added to the set of "fail-states" fail.
- Deterministic plans for "fail-states" are constructed until no more "fail-states" remain.
- Eventually, the algorithm either returns a strong cyclic plan or FAILURE if no such plan exists.

Strong cyclic plans

Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

Correctness

Theorem

Procedure incremental-strong-cyclic-plan, called with task Π , returns a strong cyclic plan for Π iff such a plan exists, and FAILURE, otherwise.

Strong cyclic plans

Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance

Underlying classical planner

- Can use any classical planner for deterministic searches.
- Can benefit from heuristics etc. used there.
- Classical planner can be configured to prefer short solutions or solutions using deterministic actions induced by nondeterministic actions with few different outcomes (likely fewer new "fail-states").

Strong cyclic plans

Motivation

> Nested Fixpoint Algorithm

> > Incremental Planning Algorithm

Maintenance

Improvements

- When to terminate a deterministic sub-search?
 - At goal states?
 - At states currently part of the partial solution?
 - At parent of currently solved "fail-state"?

This can make a huge differnce.

- Similarly: Where should the heuristic guide the classical planner? Goals, partial solution, parent node?
- Additional marking of nodes as definitely solved if this can be detected.
- State reuse between subsequent classical planner calls.
- Generalization of solved states by regression search from goal along weak (deterministic) plan (cf. [Muise/McIlraith/Beck, 2012]).

M

Strong cyclic plans

Motivation

Nested Fixpoint Algorithm

Incremental Planning Algorithm

Maintenance



Strong cyclic plans

Maintenance

Definition

Example

Algorithm

Summary

Maintenance goals

- In this lecture, we usually limit ourselves to the problem of finding plans that reach a goal state.
- In practice, planning is often about more general goals, where execution cannot be terminated.
 - 1 An animal: find food, eat, sleep, find food, eat, sleep, ...
 - 2 Cleaning robot: keep the building clean.
- These problems cannot be directly formalized in terms of reachability because infinite (unbounded) plan execution is needed.
- We do not discuss this topic in full detail. However, to give at least a little impression of planning for temporally extended goals, we will discuss the simplest objective with infinite plan executions: maintenance.

Strong cyclic plans

Maintenance

Definition

DRG

2

Example

Algorith



Strong cyclic plans

Maintenance

Definition

Example

Algorithm

Summary

Definition

Let $\mathscr{T} = \langle V, I, O, \gamma \rangle$ be a planning task with state set *S* and set of goal states $S_* = \{s \in S | s \models \gamma\}$. A strategy π for \mathscr{T} is called a plan for maintenance for \mathscr{T} iff

• $\pi(s)$ is applicable in s for all $s \in S_{\pi}$,

•
$$S_{\pi}(s_0) \subseteq S_{\pi}$$
, and

$$\blacksquare S_{\pi}(s_0) \subseteq S_{\star}.$$

- The state of an animal is determined by three state values: hunger (0, 1, 2), thirst (0, 1, 2) and location (river, pasture, desert). There is also a special state called death.
- Thirst grows when not at river; at river it is 0.
- Hunger grows when not on pasture; on pasture it is 0.
- If hunger or thirst exceeds 2, the animal dies.
- The goal of the animal is to avoid death.

Strong cyclic plans

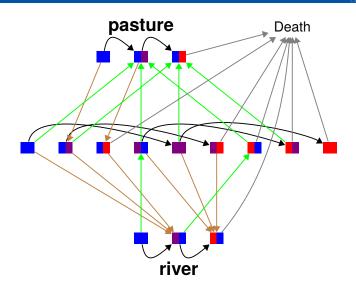
Maintenance

Definition

Example

Algorithr

Transition system for the example



Strong cyclic

BURG

plans

Maintenance

Definition

Example

Algorithm

We can infer rules backwards starting from the death condition.

- If in desert and thirst = 2, must go to river.
- 2 If in desert and hunger = 2, must go to pasture.
- If on pasture and thirst = 1, must go to desert.
- 4 If at river and hunger = 1, must go to desert.

If the above rules conflict, the animal will die.



DRG

Strong cyclic plans

Maintenance

Definition

Example

Algorithm

Algorithm for maintenance goals Idea

Summary of the algorithm idea

Repeatedly eliminate from consideration those states that in one or more steps unavoidably lead to a non-goal state.

- A state is *i*-safe iff there is a plan that guarantees "survival" for the next *i* actions.
- A state is safe (or ∞ -safe) iff it is *i*-safe for all $i \in \mathbb{N}_0$.
- The 0-safe states are exactly the goal states: maintenance objective is satisfied for the current state.
- Given all *i*-safe states, compute all *i* + 1-safe states by using strong preimages.
- For some *i* ∈ N₀, *i*-safe states equal *i* + 1-safe states because there are only finitely many states and at each step and *i* + 1-safe states are a subset of *i*-safe states. Then *i*-safe states are also ∞-safe.

Strong cyclic

Maintenance

Definitio

DRG

2

Example

Algorithm

Summa

Algorithm for maintenance goals

Planning for maintenance goals

```
def maintenance-plan(\langle V, I, O, \gamma \rangle):
   S = set of states over V
   Safe<sub>0</sub> := {s \in S \mid s \models \gamma}
   for each i \in \mathbb{N}_1:
          Safe_i := Safe_{i-1} \cap \bigcup_{o \in O} spreimg_o(Safe_{i-1})
          if Safe_i = Safe_{i-1}:
                break
   if I \notin Safe_i:
          return no solution
   for each s \in Safe_i:
          \pi(s) := some operator o \in O with img_o(s) \subseteq Safe_i
   return \pi
```

Strong cyclic

Maintenance

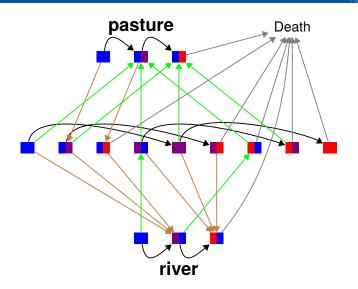
Definition

æ

Example

Algorithm

Transition system for the example



Strong cyclic

strong cyclic plans

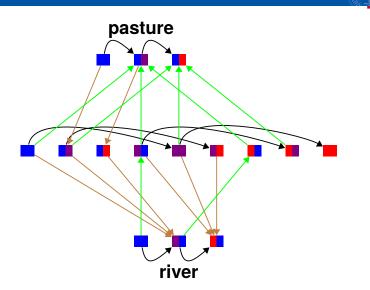
Maintenance

Definition

Example

Algorithm

0-safe states



Strong cyclic plans

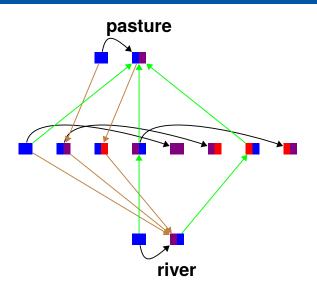
Maintenance

Definition

Example

Algorithm

1-safe states



UNI FREIBURG

> Strong cyclic plans

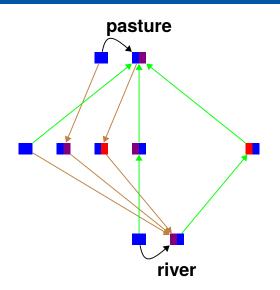
Maintenance

Definition

Example

Algorithm

Maintenance goals *i*-safe states for all $i \ge 2$



UNI FREIBURG

Strong cyclic plans

Maintenance

Definition

Example

Algorithm



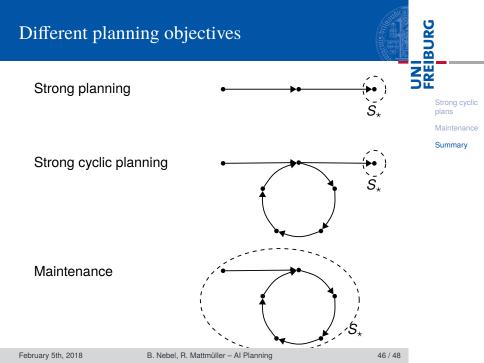
Strong cyclic plans

Maintenance

Summary

Summary

February 5th, 2018



Outlook: Computational tree logic

- We have considered different classes of solutions for planning tasks by defining different planning problems.
 - strong planning problem: find a strong plan
 - strong cyclic planning problem: find a strong cyclic plan
 ...
- Alternatively, we could allow specifying goals in a modal logic like computational tree logic to directly express the type of plan we are interested in using modalities such as A (all), E (exists), G (globally), and F (finally).
 - Weak planning: EFφ
 - Strong planning: $AF\phi$
 - Strong cyclic planning: $AGEF\phi$
 - Maintenance: $AG\phi$

Strong cyclic plans

DRD

æ

Summary

strong plans to strong cyclic plans.The story does not end there: When considering infinitely

We have extended our earlier planning algorithm from

- executing plans, many more types of goals are feasible.
- We considered maintenance as a simple example of a temporally extended goal.
- In general, temporally extended goals be expressed in modal logics such as computational tree logic (CTL).
- We presented dynamic programming (backward search) algorithms for strong cyclic and maintenance planning.
- In practice, one might implement both algorithms by using binary decision diagrams (BDDs) as a data structure for state sets.

Strong cyclic plans Maintenance

Summary

2