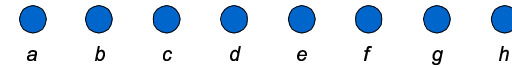


Planning

- Plan Skeletons
- Pruning Techniques
- Planning with Preferences

Example: Coin Game



- Jump with **one** coin over **two** coins onto **one** coin
- After **4** moves, obtain **4** stacks with **2** coins each

Game Description: Initial Situation

$$\text{Holds}(f, S_0) \equiv f = \text{Cell}(a, 1) \vee f = \text{Cell}(b, 1) \vee f = \text{Cell}(c, 1) \vee f = \text{Cell}(d, 1) \vee \\ f = \text{Cell}(e, 1) \vee f = \text{Cell}(f, 1) \vee f = \text{Cell}(g, 1) \vee f = \text{Cell}(h, 1)$$

Game Description: Preconditions

$$\text{Poss}(\text{Jump}(x, y), s, t) \equiv t = \text{Do}(\text{Jump}(x, y), s) \wedge \\ \text{Holds}(\text{Cell}(x, 1), s) \wedge \text{Holds}(\text{Cell}(y, 1), s) \wedge \\ \text{CoinsBetween}(x, y, s) = 2$$

Game Description: Effects

$$\begin{aligned}
 & Poss(Jump(x, y), s, t) \supset \\
 & (\forall f)[f = Cell(x, 0) \vee f = Cell(y, 2) \vee (Holds(f, s) \wedge f \neq Cell(x, 1) \wedge f \neq Cell(y, 1)) \\
 & \quad \supset Holds(f, t)] \\
 & \wedge \\
 & (\forall f)[f = Cell(x, 1) \vee f = Cell(y, 1) \vee (\neg Holds(f, s) \wedge f \neq Cell(x, 0) \wedge f \neq Cell(y, 2)) \\
 & \quad \supset \neg Holds(f, t)]
 \end{aligned}$$

Game Description: Goal

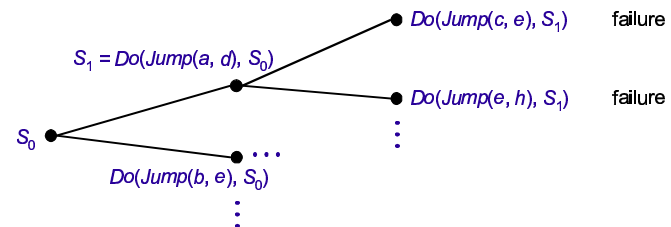
$$Goal(s) \equiv (\forall x, v) (Holds(Cell(x, v), s) \supset v = 0 \vee v = 2)$$

A Simple GOLOG Planning Program

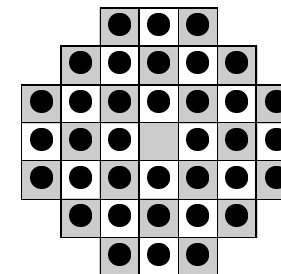
```

while  $\neg$ Goal do
   $\pi a. a$ 
endWhile
    
```

If executed offline, this program defines the whole search space:



A More Difficult Planning Problem



Domain-Independent Heuristics

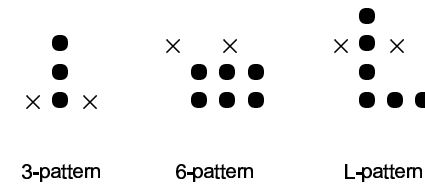
- Search Space for Peg Solitaire too large for brute-force search (Solution depth: 35; branching factor: 4 initially, quickly increasing)

Some domain-independent search heuristics:

- hash tables (aka transposition tables)
- exploiting symmetries

But these techniques usually do not scale up.

Domain-Dependent Heuristics: A Planning Strategy for Peg Solitaire



3-pattern

6-pattern

L-pattern

Plan Skeletons

- A (nondeterministic) GOLOG program can be used to describe a **plan skeleton**.
- If executed offline, the program defines a heuristics which spans a (reduced) search space.
- Two notions of completeness:
 - **strong completeness**: all plans are in the skeleton
 - **weak completeness**: skeleton contains a plan if problem is solvable

A GOLOG Program for Peg Solitaire

```

proc SolvePattern(p)
  while ¬Solved(p) do
    πx.πy.πz. (y ∈ p?; Jump(x, y, z))
  endWhile
endProc;

while ¬Goal do
  πp. (IsPattern(p)?; SolvePattern(p))
endWhile

```

Jump(x, y, z) :-> move with peg in x over y into cell z.

Remark: This heuristics is complete but not strongly complete.

A Generic Plan Skeleton

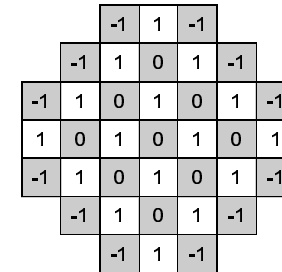
```

while  $\neg$ Goal do
   $\pi a, a; Allowed?$ 
endWhile
    
```

This program describes search with local constraints for action selection.

A branch in the search tree is cut off by the domain-dependent condition *Allowed*.

Example: Resource Count in Peg Solitaire



$Holds(Allowed, s) \stackrel{def}{=} ResourceCount(s) \geq 0$

$ResourceCount(s) : \Leftrightarrow$ weighted sum of all occupied cells in s

Remark: This heuristics is strongly complete.

Pruning Techniques

General Pruning Techniques

Linear Temporal Logic: Classical logic extended by modalities

symbol	meaning
$\bigcirc \phi$	ϕ holds next
$\square \phi$	ϕ always holds
$\diamond \phi$	ϕ eventually holds
$\phi \mathcal{U} \psi$	ϕ holds until ψ

Example: $\square [In(x, y) \supset In(x, y) \mathcal{U} At(y, GoalLocation(x))]$

Semantics

S_0, S_1, S_2, \dots \Rightarrow infinite sequence of situations

A formula ϕ is **true in S_j** , written $S_j \models \phi$, if

- $\text{Holds}(\phi, S_j)$, where ϕ atomic;
- $S_{j+1} \models \psi$, where ϕ is $\text{O}\psi$;
- $S_j \models \psi$ for all $j \geq i$, where ϕ is $\text{O}\Box\psi$;
- $S_j \models \psi$ for some $j \geq i$, where ϕ is $\text{O}\Diamond\psi$;
- $S_j \models \psi_2$ for some $j \geq i$ such that $S_k \models \psi_1$ for all $i \leq k < j$, where ϕ is $\psi_1 \text{U} \psi_2$;
- $S_j \not\models \psi$, where ϕ is $\neg\psi$; likewise for $\wedge, \vee, \supset, \equiv$.

Example

Consider the interpretation

S_0	S_1	S_2	S_3	S_4	
A	$\neg A$	$\neg A$	$\neg A$	$\neg A$	\dots
B	B	$\neg B$	$\neg B$	$\neg B$	

Then

$$S_0 \models \Box(A \supset B)$$

$$S_0 \models \Box A \supset B$$

$$S_0 \models \text{O}\Box \neg A$$

$$S_0 \models \Diamond(\neg A \wedge \neg B)$$

$$S_0 \models \Box(B \text{U} \neg A)$$

A Logistics Domain

Symbol	Type
<i>Load</i>	OBJECT \times VEHICLE \mapsto ACTION
<i>Unload</i>	OBJECT \times VEHICLE \mapsto ACTION
<i>Move</i>	VEHICLE \times LOCATION \mapsto ACTION
<i>Fly</i>	VEHICLE \times LOCATION \mapsto ACTION

Symbol	Type
<i>At</i>	OBJECT \times LOCATION \mapsto FLUENT
<i>In</i>	OBJECT \times VEHICLE \mapsto FLUENT
<i>Location</i>	VEHICLE \times LOCATION \mapsto FLUENT

- 10 objects, 2 locations and 1 truck each in 4 cities, 3 planes
 \rightarrow total state space $6 \cdot 10^{14}$

Control Rules for Logistics Domain

1. Move a vehicle only if there are no objects at its present location which need to be loaded into or unloaded from the vehicle;
2. Move a vehicle only to locations where it needs to pick up or to drop goods;
3. Load objects only into the kind of vehicle (truck or plane) with which they need to be transported;
4. Unload objects only where they need to be unloaded.

Auxiliary Predicates

$MustBeMovedBy(o, v) : \Leftrightarrow$ the goal requires to move object o by the type of vehicle (truck or plane) to which v belongs

$MustBeUnloadedAt(o, l) : \Leftrightarrow$ the goal requires to unload object o at location l (possibly an airport location)

The Control Rules in Linear Temporal Logic

1. $\Box[Location(v, l) \wedge (\exists o) (At(o, l) \wedge MustBeMovedBy(o, v)) \supset OLocation(v, l)]$
 $\Box[Location(v, l) \wedge (\exists o) (In(o, v) \wedge MustBeUnloadedAt(o, l)) \supset OLocation(v, l)]$
2. $\Box[Location(v, l) \wedge OLocation(v, l') \wedge l \neq l' \supset$
 $(\exists o) (At(o, l') \wedge MustBeMovedBy(o, v) \vee In(o, v) \wedge MustBeUnloadedAt(o, l'))]$
3. $\neg MustBeMovedBy(o, v) \supset \Box \neg In(o, v)$
4. $\Box[In(o, v) \wedge Location(v, l) \wedge \neg MustBeUnloadedAt(o, l) \supset OIn(o, v)]$

Progression (1)

- Evaluating a temporal logic formula against an action sequence requires to consider the entire history, which can be costly.
- Control formulas can be **progressed** successively through situations, and then be locally evaluated.
- For example, $\Box \phi$ is valid now if ϕ holds and $\Box \phi$ holds in the next situation.

Progression (2)

Let S be the current situation, then **Progress(ϕ)** is

- **True** if $Holds(\phi, S)$ and ϕ atomic
- **False** if $\neg Holds(\phi, S)$ and ϕ atomic
- $Progress(\psi_1) \wedge Progress(\psi_2)$ if ϕ is $\psi_1 \wedge \psi_2$; likewise for $\vee, \neg, \supset, \equiv$
- $\bigwedge_c Progress(\psi/x/c)$ if ϕ is $(\forall x)\psi$
- $\bigvee_c Progress(\psi/x/c)$ if ϕ is $(\exists x)\psi$

Note: This restricts progression to domains with finitely many ground terms.

Progression (3)

- ψ , if ϕ is $O\psi$
- $\text{Progress}(\psi) \wedge \Box\psi$, if ϕ is $\Box\psi$
- $\text{Progress}(\psi) \vee \Diamond\psi$, if ϕ is $\Diamond\psi$
- $\text{Progress}(\psi_2) \vee (\text{Progress}(\psi_1) \wedge \psi_1 \mathcal{U} \psi_2)$, if ϕ is $\psi_1 \mathcal{U} \psi_2$

Preferences

Planning with Preferences

- In the classical definition of planning, all plans are equal
- Additional preferences allow to distinguish “good” from “bad” plans
- A preference can be given quantitatively (by an objective function) or qualitatively by **preference formulas**

Basic Desire Formulas

Definition 4.3.1

A **basic desire formula** can be

- a fluent
- the atomic $\text{final}(f)$, where f is a fluent
- the atomic $\text{occurs}(a)$, where a is an action
- any combination of the above using the logical connectives and the modalities of linear temporal logic

Atomic Preference Formulas

Definition 4.3.2

An **atomic preference formula** is of the form

$$\varphi_0 < \varphi_1 < \dots < \varphi_n$$

where $n \geq 0$ and the φ_i 's are basic desire formulas.

Note: If $n = 0$, then a preference formula is simply a basic desire formula.

Preference Formulas

Definition 4.3.3

A **preference formula** can be

- an atomic preference formula
- a **conditional** $\varphi \Rightarrow \Phi$
- a **general conjunction** $\Phi_1 \& \dots \& \Phi_n$
- a **general disjunction** $\Phi_1 | \dots | \Phi_n$

φ basic desire; $\Phi, \Phi_1, \dots, \Phi_n$ preference formulas; $n \geq 2$

Evaluation of Preferences

Each plan determines a sequence $S_0, S_1, S_2, \dots, S_n$ of situations.

Consider the infinite sequence $S_0, S_1, S_2, \dots, S_n, S_n, S_n, \dots$

Basic desires are evaluated as temporal logic formulas (cf Slide 16), augmented by the definition

- $S_i \models \text{final}(f)$ if $\text{Holds}(f, S_n)$
- $S_i \models \text{occurs}(a)$ if $S_{i+1} = \text{Do}(a, S_i)$

Weight of Basic Desire

Consider $S_0, S_1, S_2, \dots, S_n, S_n, S_n, \dots$

Let φ be a basic desire, then

$$\omega(\varphi) = \begin{cases} 0 & \text{if } S_0 \models \varphi \\ 1 & \text{otherwise} \end{cases}$$

Weight of Atomic Preference

$$\omega(\varphi_0 < \dots < \varphi_n) = \begin{cases} \min\{i : \omega(\varphi_i) = 0\} & \text{if it exists} \\ n+1 & \text{otherwise} \end{cases}$$

Weight of Preferences

$$\omega(\varphi \Rightarrow \Phi) = \begin{cases} \omega(\Phi) & \text{if } \omega(\varphi) = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\omega(\Phi_1 \& \dots \& \Phi_n) = \sum \omega(\Phi_i)$$

$$\omega(\Phi_1 | \dots | \Phi_n) = \min \omega(\Phi_i)$$

Preferred Plans

- Ideal plans, which satisfy all preferences, have weight 0.
- Plan p_1 is preferred over plan p_2 if the sum of the weights of all preference formulas wrt. p_1 is lower than that wrt. p_2 .

Progression of Preference Formulas (1)

- To avoid evaluation of preferences for each plan separately, preference formulas can be progressed.
- Progression of basic desires as temporal logic formulas (cf Slides 24 - 25) augmented by the definition
 - $\text{Progress}(\text{final}(f))$ is $\text{final}(f)$; and
 - $\text{Progress}(\text{occurs}(a))$ is $\text{next}(a)$, with the auxiliary definition

$$S_i \models \text{next}(a) \text{ if } S_i = \text{Do}(a, s) \text{ for some } s$$

$$\text{Progress}(\text{next}(a)) \text{ is True if } S = \text{Do}(a, s) \text{ for some } s, \text{ else False}$$

Progression of Preference Formulas (2)

- $Progress(\varphi_0 < \varphi_1 < \dots < \varphi_n) = Progress(\varphi_0) < Progress(\varphi_1) < \dots < Progress(\varphi_n)$
- $Progress(\varphi \Rightarrow \Phi) = Progress(\varphi) \Rightarrow Progress(\Phi)$
- $Progress(\Phi_1 \& \dots \& \Phi_n) = Progress(\Phi_1) \& \dots \& Progress(\Phi_n)$
- $Progress(\Phi_1 | \dots | \Phi_n) = Progress(\Phi_1) | \dots | Progress(\Phi_n)$

Correctness of Progression

Consider a sequence $S_0, S_1, S_2, \dots, S_n, S_{n+1}, S_{n+2}, \dots$

A formula Φ has the same weight wrt. S_0, S_1, \dots as $Progress^n(\Phi)$ has wrt. S_n, S_{n+1}, \dots