Artificial Intelligence

14. Planning Formalisms

How to Describe Problems, and What is a “Problem” Anyway?

Prof Sara Bernardini
bernardini@diag.uniroma1.it
www.sara-bernardini.com

Sapienza Università di Roma

Autumn Term

Agenda

1. Introduction
2. Transition Systems
3. STRIPS Planning
4. Finite-Domain Representation (FDR) Planning
5. STRIPS vs. FDR
6. Extended Planning Frameworks [for Reference]
7. Conclusion

Reminder: Planning = General Problem Solving

(some new problem)

describe problem in planning language $\mapsto$ use off-the-shelf solver

(its solution)

- Any problem that can be formulated as a planning problem.
- Don't write the C++ code, just describe the problem!
- Don't maintain the C++ code, maintain the description!

What is a Planning Problem?

Given a planning task:

- A description of the initial state.
- A description of the goal condition.
- A description of a set of possible actions.

→ Find a schedule of actions (a plan) that brings us from the initial state to a state in which the goal condition holds.

Classical Planning

. . . makes **Simplifying Assumptions:**

- Initial situation unique and completely known, environment deterministic, static, discrete, single-agent.
- Actions executed one-by-one, plans are sequences.

This is often not the case in practice! Examples? Handling uncertainty (robot control), temporal/parallel execution (transportation), . . .

So why do we do this?

- Clean framework to study planning problems. (Simplicity is a virtue!)
- Where most influential ideas were conceived.
- Successful applications using classical planning.
- We can successfully compile many extended paradigms into classical planning.

→ We focus entirely on classical planning in this course.

Algorithmic Problems in Planning

Satisficing Planning

Input: A planning task Π.
Output: A plan for Π, or **unsolvable** if no plan for Π exists.

Optimal Planning

Input: A planning task Π.
Output: An optimal plan for Π, or **unsolvable** if no plan for Π exists.

$\rightarrow$ The techniques successful for either one of these are almost disjoint!
$\rightarrow$ Satisficing planning is *much* more effective in practice.

$\rightarrow$ Programs solving these problems are called (optimal) planners, planning systems, or planning tools.

Computational Complexity in Planning

Why? From this course's point of view, it's simply one technical tool we need.

→ To get a heuristic h, we map the planning problem into a simpler (abstract/ relaxed) planning problem, from whose solution we compute h. To compute h efficiently, the "simpler" problem must be solvable in polynomial time.

Definition (PlanEx and PlanOpt). *PlanEx is the problem of deciding, given a (STRIPS or FDR) planning task Π, whether or not there exists a plan for Π. PlanOpt is the problem of deciding, given Π and $B \in \mathbb{R}_0^+$, whether or not there exists a plan for Π whose cost is at most B.*

→ PlanEx ≈ satisficing planning, PlanOpt ≈ optimal planning.

Theorem (Planning is Hard). *Each of PlanEx and PlanOpt is* **PSPACE**-*complete.*

Proof. See Chapter 13 and Bylander (1994), if interested.

Reminder: **NP** and **PSPACE**

Def Turing machine: Works on a tape consisting of tape cells, across which its R/W head moves. The machine has internal states. There are transition rules specifying, given the current cell content and internal state, what the subsequent internal state will be, how what the R/W head does (write a symbol and/or move). Some internal states are accepting.

Def NP: Decision problems for which there exists a *non-deterministic* Turing machine that runs in *time* polynomial in the size of its input. Accepts if *at least one* of the possible runs accepts.

Def PSPACE: Decision problems for which there exists a *deterministic* Turing machine that runs in *space* polynomial in the size of its input.

Relation: Non-deterministic polynomial space can be simulated in deterministic polynomial space. Thus **PSPACE** = **NPSPACE**, and hence (trivially) **NP** $\subseteq$ **PSPACE**. It is commonly believed that **NP** $\not\supseteq$ **PSPACE** (similar to **P** $\subseteq$ **NP**).

→ For comprehensive details, please see a text book. A good one is [Garey and Johnson (1979)]. (On the first 3 pages, they explain why knowing about **NP**-hardness will help you talk to your future boss.)

Our Agenda for This Chapter

2. **Transition Systems:** The basic framework we'll be moving in; forms the basis for both STRIPS and FDR. (= state space)
3. **STRIPS Planning:** STRIPS is by far the most wide-spread planning formalism. It is also the simplest possible reasonably expressive planning formalism, and thus a canonical subject to study.
4. **Finite-Domain Representations (FDR):** FDR is only slightly more general than STRIPS, but as we shall see can be quite useful.
5. **STRIPS vs. FDR:** The two formalisms can be compiled into each other. Such compilations are wide-spread in practice, and we will use them at some points during the course.
6. **Extended Planning Frameworks:** To at least give you a brief glimpse beyond classical planning.

Transition Systems

→ State space of planning task = a transition system.

Definition (Transition System). *A transition system is a 6-tuple* $\Theta = (S, L, c, T, I, S^G)$ *where:*

- S *is a finite set of states.*
- L *is a finite set of transition labels.*
- $c : L \mapsto \mathbb{R}_0^+$ *is the cost function.*
- $T \subseteq S \times L \times S$ *is the transition relation.*
- $I \in S$ *is the initial state.*
- $S^G \subseteq S$ *is the set of goal states.*

The size of Θ *is its number of states, size*$(\Theta) := |S|$.

We say that Θ *has the transition* (s, l, s') *if* $(s, l, s') \in T$. *We also write this* $s \xrightarrow{l} s'$, *or* $s \rightarrow s'$ *when not interested in* l.

We say that Θ *is deterministic if, for all states* s *and labels* l, *there is at most one state* s' *with* $s \xrightarrow{l} s'$.

We say that Θ *has unit costs if, for all* $l \in L$, $c(l) = 1$.

Transition Systems, ctd.

Terminology: $\Theta = (S, A, c, T, I, S^G)$; $s, s', s_i \in S$

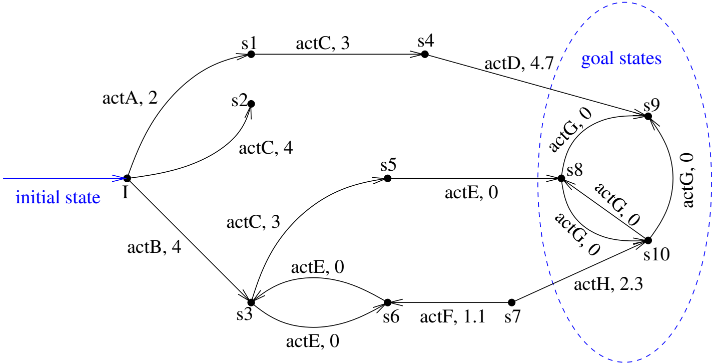
- s' successor of s if $s \to s'$; s predecessor of s' if $s \to s'$.
- s' reachable from s if there exists a sequence of transitions:

$$s = s_0 \xrightarrow{l_1} s_1, \ldots, s_{n-1} \xrightarrow{l_n} s_n = s'$$

  - $n = 0$ possible; then $s = s'$.
  - $l_1, \ldots, l_n$ is called path from s to s'.
  - $s_0, \ldots, s_n$ is also called path from s to s'.
  - The cost of that path is $\sum_{i=1}^{n} c(l_i)$.
- s' reachable (without reference state) means reachable from I.
- Solution for s: path from s to some $s' \in S^G$; optimal if cost is minimal among all solutions for s.
- s is solvable if it has a solution; else, s is a dead end.
- Solution for I is called solution for Θ; Θ is solvable if it has a solution.

Transition Systems: Illustration

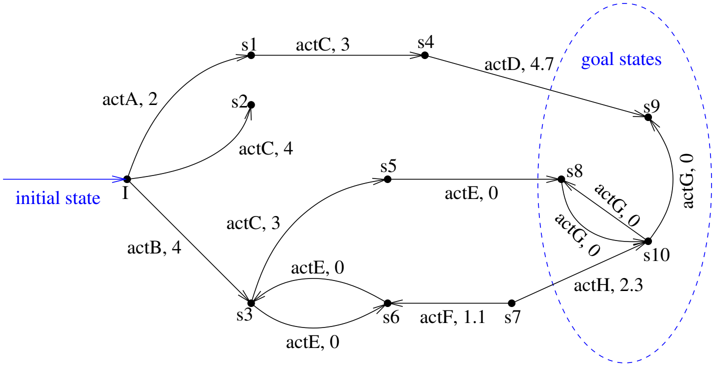
Directed labeled graphs + mark-up for initial state and goal states:



- Are all states in Θ reachable? No: s_7
- Are all states in Θ solvable? No: s_2
- Is this Θ deterministic? No: On two of the goal states (s_8, s_{10}), actG labels more than one outgoing transition.

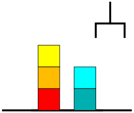
Transition Systems: Illustration, ctd.

Directed labeled graphs + mark-up for initial state and goal states:



- What are the optimal solutions for Θ? Any path that starts with actB, applies actE $n \in \{0, 2, 4, \dots\}$ times, then applies actC then actE and then no action other than actG.

Why don't we simply use Dijkstra? Example Blocksworld



- n blocks, 1 hand.
- A single action either takes a block with the hand or puts a block we're holding onto some other block/the table.

blocks	states
1	1
2	3
3	13
4	73
5	501
6	4051
7	37633
8	394353
9	4596553

blocks	states
10	58941091
11	824073141
12	12470162233
13	202976401213
14	3535017524403
15	65573803186921
16	1290434218669921
17	26846616451246353
18	588633468315403843

$\rightarrow$ We are interested in solving **huge** transition systems, represented in a **compact** way as planning tasks (up next).

STRIPS Planning: Syntax

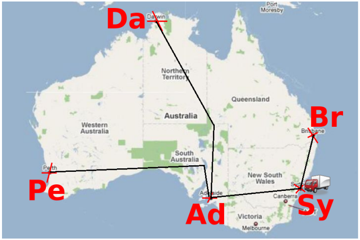
Definition (STRIPS Planning Task). *A STRIPS planning task is a 5-tuple $\Pi = (P, A, c, I, G)$ where:*

- *P is a finite set of facts, also propositions.*
- *A is a finite set of actions; each $a \in A$ is a triple $a = (pre_a, add_a, del_a)$ of subsets of P referred to as the action's precondition, add list, and delete list respectively; we require that $add_a \cap del_a = \emptyset$.*
- *$c : A \mapsto \mathbb{R}_0^+$ is the cost function.*
- *$I \subseteq P$ is the initial state.*
- *$G \subseteq P$ is the goal.*

We say that Π has unit costs if, for all $a \in A$, $c(a) = 1$. We will often give each action $a \in A$ a name (a string), and identify a with that name.

$\rightarrow$ Why do we allow 0-cost actions? Negligible cost (e.g. switch light on, take photo with smartphone), asking questions about only one kind of actions (e.g. Mars rover *take-picture* only).

STRIPS Encoding of "TSP" in Australia



- Propositions P: $\{at(x), visited(x) \mid x \in \{Sydney, Adelaide, Brisbane, Perth, Darwin\}\}$.
- Initial state I: $\{at(Sydney), visited(Sydney)\}$.
- Goal G: $\{at(Sydney)\} \cup \{visited(x) \mid x \in \{Sydney, Adelaide, Brisbane, Perth, Darwin\}\}$.
- Actions $a \in A$: $drive(x, y)$ where x, y have a road.
  - Precondition pre_a: $\{at(x)\}$.
  - Add list add_a: $\{at(y), visited(y)\}$.
  - Delete list del_a: $\{at(x)\}$.
- Cost function c:

$$c(drive(x,y)) = \begin{cases} 1 & \{x,y\} = \{Sydney, Brisbane\} \\ 1.5 & \{x,y\} = \{Sydney, Adelaide\} \\ 3.5 & \{x,y\} = \{Adelaide, Perth\} \\ 4 & \{x,y\} = \{Adelaide, Darwin\} \end{cases}$$

STRIPS Planning: Semantics

Definition (STRIPS State Space). *Let $\Pi = (P, A, c, I, G)$ be a STRIPS planning task. The state space of Π is the labeled transition system $\Theta_\Pi = (S, L, c, T, I, S^G)$ where:*

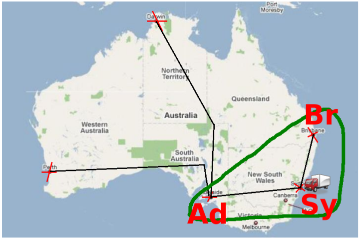
- *The states (also world states) $S = 2^P$ are the subsets of P.*
- *The labels $L = A$ are Π's actions; the cost function c is that of Π.*
- *The transitions are $T = \{s \xrightarrow{a} s' \mid a \in A[s], s' = s[\![a]\!]\}$, where $A[s] := \{a \in A \mid pre_a \subseteq s\}$ are the actions applicable in s; for $a \in A[s]$, $s[\![a]\!] := (s \setminus del_a) \cup add_a$; for $a \notin A[s]$, $s[\![a]\!]$ is undefined, $s[\![a]\!] := \bot$.*
- *The initial state I is identical to that of Π.*
- *The goal states $S^G = \{s \in S \mid G \subseteq s\}$ are those that satisfy Π's goal.*

An (optimal) plan for $s \in S$ is an (optimal) solution for s in Θ_Π. A solution for I is called a plan for Π. Π is solvable if a plan for Π exists.

For $\vec{a} = \langle a_1, \dots, a_n \rangle$, $s[\![\vec{a}]\!] := \begin{cases} s & n = 0 \\ s[\![\langle a_1, \dots, a_{n-1} \rangle]\!][\![a_n]\!] & n > 0 \end{cases}$

$\rightarrow$ Is Θ_Π deterministic? Yes: the successor state s' in $s \xrightarrow{a} s'$ is uniquely determined by s and a, through $s' = s[\![a]\!]$.

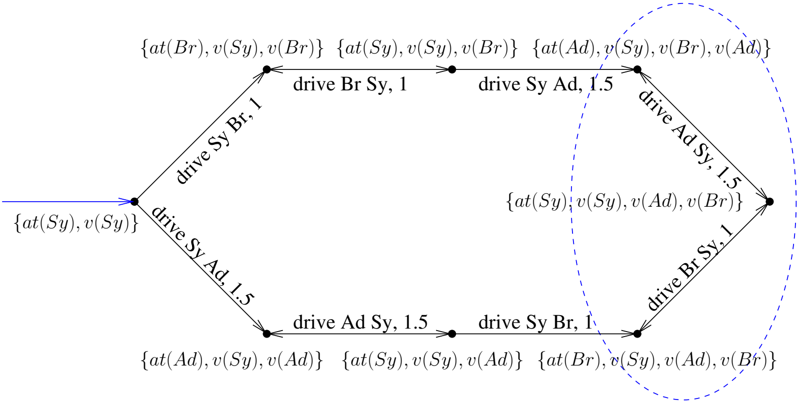
STRIPS Encoding of Simplified "TSP"



- Propositions P: $\{at(x), visited(x) \mid x \in \{Sydney, Adelaide, Brisbane\}\}$.
- Initial state I: $\{at(Sydney), visited(Sydney)\}$.
- Goal G: $\{visited(x) \mid x \in \{Sydney, Adelaide, Brisbane\}\}$. (Note: no "$at(Sydney)$".)
- Actions $a \in A$: $drive(x, y)$ where x, y have a road.
  - Precondition pre_a: $\{at(x)\}$.
  - Add list add_a: $\{at(y), visited(y)\}$.
  - Delete list del_a: $\{at(x)\}$.
- Cost function c:

$$c(drive(x,y)) = \begin{cases} 1 & \{x,y\} = \{Sydney, Brisbane\} \\ 1.5 & \{x,y\} = \{Sydney, Adelaide\} \end{cases}$$

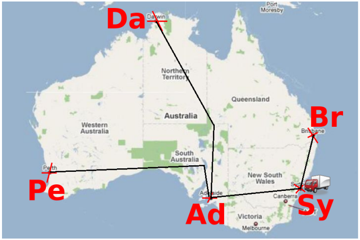
STRIPS Encoding of Simplified "TSP": State Space



→ Exactly one optimal plan: drive Sy Br, drive Br Sy, drive Sy Ad.

→ Is this actually the state space? No, only the reachable part. E.g., Θ_Π also includes the states $\{v(Sy)\}$ and $\{at(Sy), at(Br)\}$.

Questionnaire



- Propositions P: $\{at(x), visited(x) \mid x \in \{Sydney, Adelaide, Brisbane, Perth, Darwin\}\}$.
- Initial state I: $\{at(Sydney), visited(Sydney)\}$.

How many states are there in the "TSP in Australia" task?

$\rightarrow$: $2^{10} = 1024$. But only a small portion of them are reachable (less than $5 \cdot 2^4 = 80$)!

FDR Planning: Syntax

Definition (FDR Planning Task). *A finite-domain representation planning task, short FDR planning task, is a 5-tuple $\Pi = (V, A, c, I, G)$ where:*

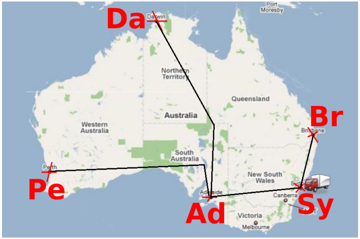
- *V is a finite set of state variables, each $v \in V$ with a finite domain D_v.*
  We refer to (partial) functions on V, mapping each $v \in V$ into a member of D_v, as (partial) variable assignments.
- *A is a finite set of actions; each $a \in A$ is a pair (pre_a, eff_a) of partial variable assignments referred to as the action's precondition and effects.*
- *$c : A \mapsto \mathbb{R}_0^+$ is the cost function.*
- *I is a complete variable assignment called the initial state.*
- *G is a partial variable assignment called the goal.*

We say that Π has unit costs if, for all $a \in A$, $c(a) = 1$.

→ In FDR, a (partial) variable assignment represents a state in I, a condition in pre_a and G, and an effect instruction in eff_a.

Notation: Pairs (v, d) are facts, also written $v = d$. We identify partial variable assignments p with fact sets. We write $V[p] := \{v \in V \mid p(v) \text{ is defined}\}$.

FDR Encoding of "TSP"



- Variables V: $at : \{Sydney, Adelaide, Brisbane, Perth, Darwin\}$; $visited(x) : \{T, F\}$ for $x \in \{Sydney, Adelaide, Brisbane, Perth, Darwin\}$.
- Initial state I: $at = Sydney, visited(Sydney) = T, visited(x) = F$ for $x \neq Sydney$.
- Goal G: $at = Sydney, visited(x) = T$ for all x.
- Actions $a \in A$: $drive(x, y)$ where x, y have a road.
  - Precondition pre_a: $at = x$.
  - Effect eff_a: $at = y, visited(y) = T$.
- Cost function c:

$$c(drive(x,y)) = \begin{cases} 1 & \{x,y\} = \{Sydney, Brisbane\} \\ 1.5 & \{x,y\} = \{Sydney, Adelaide\} \\ 3.5 & \{x,y\} = \{Adelaide, Perth\} \\ 4 & \{x,y\} = \{Adelaide, Darwin\} \end{cases}$$

FDR Planning: Semantics

Definition (FDR State Space). *Let $\Pi = (V, A, c, I, G)$ be an FDR planning task. The state space of Π is the labeled transition system $\Theta_\Pi = (S, L, c, T, I, S^G)$ where:*

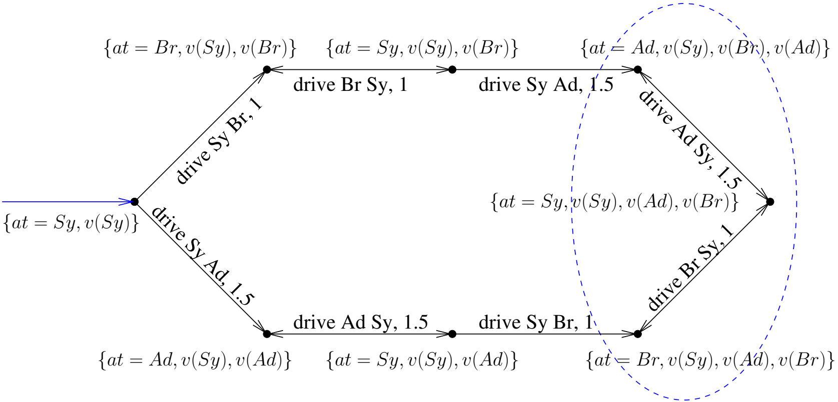
- *The states (also world states) S are the complete variable assignments.*
- *The labels $L = A$ are Π's actions; the cost function c is that of Π.*
- *The transitions are $T = \{s \xrightarrow{a} s' \mid a \in A[s], s' = s[\![a]\!]\}$, where $A[s] := \{a \in A \mid pre_a \subseteq s\}$ are the actions applicable in s; for $a \notin A[s]$, $s[\![a]\!] := \bot$; for $a \in A[s]$, $s[\![a]\!](v) := \begin{cases} eff_a(v) & v \in V[eff_a] \\ s(v) & v \notin V[eff_a] \end{cases}$*
- *The initial state I is identical to that of Π.*
- *The goal states $S^G = \{s \in S \mid G \subseteq s\}$ are those that satisfy Π's goal.*

$\rightarrow$ In $s[\![a]\!]$, instead of "adding/deleting" facts, we overwrite the previous variable values by eff_a.

$\rightarrow$ Plan, optimal plan, $s[\![\vec{a}]\!]$ for action sequence $\vec{a}$: as before (slide 20).

FDR Encoding of Simplified "TSP": State Space

(using "$v(x)$" as shorthand for $visited(x) = T$)



$\rightarrow$ This is only the reachable part of the state space: E.g., Θ_Π also includes the state $\{at = Sy, v(Br)\}$. (But neither $\{v(Sy)\}$ nor $\{at = Sy, at = Br\}$, compare slide 22.)

Questionnaire

Question!

How many STRIPS state variables are needed to encode the problem of finding a path in a graph with n vertices?

(A): 1 (B): n

(C): $\lceil \log_2 n \rceil$ (D): $2 * \lceil \log_2 n \rceil$

→ (D): We need to encode our current position in the graph. This can be done with n propositions of the form "$at(p)$", but it can be done more compactly by: numbering the positions $ID(p)$; representing $ID(p)$ in the binary system using $\lceil \log_2 n \rceil$ bits bit_i; and representing each bit_i with two STRIPS facts $True(bit_i)$ and $False(bit_i)$.

Question!

How many FDR state variables are needed for this?

(A): 1 (B): n

(C): $\lceil \log_2 n \rceil$ (D): $2 * \lceil \log_2 n \rceil$

→ (A): We need 1 variable with n values, encoding our current position in the graph.

STRIPS vs. FDR in Practice

How do people use FDR?

- Our surface language is PDDL, which corresponds to STRIPS.
- Most implemented planning tools are based on Fast Downward (FD) [Helmert (2009)], which reads PDDL input, then internally uses a "clever" STRIPS-2-FDR translation (see next).
- That translation involves a **PSPACE**-complete sub-problem.

Why??? Practical Efficiency!

- Regression: FDR avoids myriads of unreachable states. → **Chapter 17**
- Causal Graphs: Capture variable dependencies; have a much clearer structure for clever FDR (e.g., acyclic vs. cyclic). → **Chapter 16**
- Complexity Analysis: Better with clearer causal graph. → **Chapter 16**
- Construction of Heuristic Functions: Better with multiple-valued variables and clearer causal graph. → **Chapters 18**
- Modeling: Anyway, FDR is more natural! (It's just one truck, after all.)

Why does anybody use STRIPS? It's a legacy system.

→ We should be modeling in FDR. For historical reasons, we aren't.

STRIPS vs. FDR Conversions

Conversions:

(I) FDR-2-STRIPS: For each variable v with domain $\{d_1, \dots, d_k\}$, make k STRIPS facts "$v = d_1$", …, "$v = d_k$".

(II) STRIPS-2-FDR: Naïve vs. clever variants, see slides 36 – 39.

What role does all this play here?

- Both STRIPS and FDR are used in practice. The **programming exercises** focus on the planner Fast Downward, which uses FDR.
- Some techniques in the remainder of the course are easier to introduce in STRIPS, some are easier in FDR, so we will keep both around.
- Specific relevance of (I): If the course introduces a technique A in STRIPS, then A in FDR (and hence your FD code!) is equivalent to "convert-FDR-2-STRIPS-then-do-A".
- Specific relevance of (II): So you get an understanding of how FD processes the PDDL/STRIPS input to FDR.

Isomorphism

Definition (Isomorphism). *Let* $\Theta = (S, L, c, T, I, S^G)$ *and* $\Theta' = (S', L', c', T', I', S'^G)$ *be transition systems. We say that* Θ *is* *isomorphic* *to* Θ'*, written* $\Theta \sim \Theta'$*, if there exist bijective functions* $\varphi : S \mapsto S'$ *and* $\psi : L \mapsto L'$ *such that:*

1. $\varphi(I) = I'$.
2. $s \in S^G$ *iff* $\varphi(s) \in S'^G$.
3. $(s, l, t) \in T$ *iff* $(\varphi(s), \psi(l), \varphi(t)) \in T'$.
4. *For all* $l \in L$, $c(l) = c'(\psi(l))$.

→ Isomorphic transition systems are identical modulo renaming states and actions.

→ Isomorphisms typically result from compilations between different formalisms (see later this chapter); we will also sometimes use them as a technical device.

FDR-2-STRIPS: Details

Definition (FDR-2-STRIPS). *Let $\Pi = (V, A, c, I, G)$ be an FDR planning task. The STRIPS conversion of Π is the STRIPS task $\Pi^{\mathsf{STR}} = (P_V, A^{\mathsf{STR}}, c, I, G)$ where:*

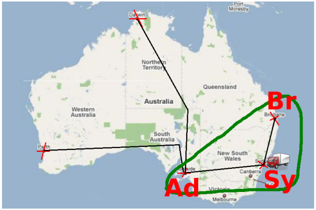
- $P_V = \{v = d \mid v \in V, d \in D_v\}$ *is the set of (STRIPS) facts.*
- $A^{\mathsf{STR}} = \{a^{\mathsf{STR}} \mid a \in A\}$ *where* $pre_{a^{\mathsf{STR}}} = pre_a$, $add_{a^{\mathsf{STR}}} = eff_a$, *and*
  $$del_{a^{\mathsf{STR}}} = \bigcup_{(v=d)\in eff_a} \begin{cases} \{v = pre_a(v)\} & \text{if } pre_a(v) \text{ is defined} \\ \{v = d' \mid d' \in D_v \setminus \{d\}\} & \text{otherwise} \end{cases}$$
- *The cost function c is defined by $c(a^{\mathsf{STR}}) := c(a)$ for all $a^{\mathsf{STR}} \in A^{\mathsf{STR}}$.*
- *I and G are identical to those of Π.*

$\rightarrow$ The adds establish the new variable values of eff_a; the deletes make sure to erase the previous values of those variables.

$\rightarrow$ Take-home message: FDR variable/value pairs $\approx$ STRIPS facts!

Proposition. *Let $\Pi = (V, A, c, I, G)$ be an FDR planning task, and let Π^{STR} be its STRIPS conversion. Then Θ_Π is isomorphic to the sub-system of $\Theta_{\Pi^{\mathsf{STR}}}$ induced by those $s \subseteq P_V$ where, for each $v \in V$, s contains exactly one fact of the form $v = d$. All other states in $\Theta_{\Pi^{\mathsf{STR}}}$ are unreachable.*

FDR-2-STRIPS: Simplified "TSP"



- FDR V: $at : \{Sydney, Adelaide, Brisbane\}; visited(x) : \{T, F\}$ for $x \in \{Sydney, Adelaide, Brisbane\}$.
- STRIPS P: $at(x), visited(x, T), visited(x, F)$ for $x \in \{Sydney, Adelaide, Brisbane\}$.
- FDR $dr(x, y)$: $pre = \{at = x\}$, $eff = \{at = y, v(y) = T\}$.
- STRIPS $dr(x, y)$:
  $pre = \{at(x)\}$, $add = \{at(y), v(y, T)\}$, $del = \{at(x), v(y, F)\}$.

STRIPS-2-FDR: Naïve Translation

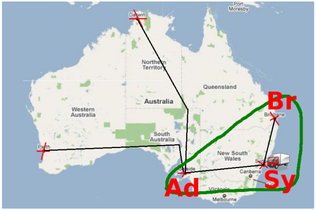
Definition (STRIPS-2-FDR). *Let $\Pi = (P, A, c, I, G)$ be a STRIPS planning task. The FDR conversion of Π is the FDR task $\Pi^{\mathsf{FDR}} = (V_P, A^{\mathsf{FDR}}, c, I^{\mathsf{FDR}}, G^{\mathsf{FDR}})$ where:*

- $V_P = \{v_p \mid p \in P\}$ *is the set of variables, all Boolean.*
- $A^{\mathsf{FDR}} = \{a^{\mathsf{FDR}} \mid a \in A\}$ *where* $pre_{a^{\mathsf{FDR}}} = \{v_p = T \mid p \in pre_a\}$ *and* $eff_{a^{\mathsf{FDR}}} = \{v_p = T \mid p \in add_a\} \cup \{v_p = F \mid p \in del_a\}$.
- *The cost function c is defined by $c(a^{\mathsf{FDR}}) := c(a)$ for all $a^{\mathsf{FDR}} \in A^{\mathsf{STR}}$.*
- $I = \{v_p = T \mid p \in I\}$*; and* $G = \{v_p = T \mid p \in G\}$.

$\rightarrow$ All variables here have two possible values only, so this does not benefit at all from the added expressivity of FDR. Hence the designation "naïve".

Proposition. *Let $\Pi = (P, A, c, I, G)$ be a STRIPS planning task, and let Π^{FDR} be its STRIPS conversion. Then Θ_Π is isomorphic to $\Theta_{\Pi^{\mathsf{STR}}}$.*

STRIPS-2-FDR, Naïve: Simplified "TSP"



- STRIPS P: $at(x), visited(x)$ for $x \in \{Sydney, Adelaide, Brisbane\}$.
- FDR V: $at(x), visited(x) : \{T, F\}$ for $x \in \{Sydney, Adelaide, Brisbane\}$.

- STRIPS $dr(x, y)$: $pre = \{at(x)\}$, $add = \{at(y), v(y)\}$, $del = \{at(x)\}$
- FDR $dr(x, y)$: $pre = \{at(x) = T\}$,
  $eff = \{at(y) = T, v(y) = T, at(x) = F\}$.

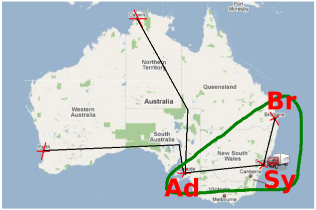
STRIPS-2-FDR: Clever Translation

How to be clever?

- Find sets $\{p_1, \dots, p_k\}$ of STRIPS facts so that every reachable state s makes exactly one p_i true.

  $\rightarrow$ Deciding whether this holds, for a given $\{p_1, \dots, p_k\}$, is **PSPACE**-complete (cf. slide 31). But one can design fast algorithms finding *some* such sets [Helmert (2009)].
- For each set $\{p_1, \dots, p_k\}$ found, make *one* FDR variable v with domain $\{d_1, \dots, d_k\}$.
- This is implemented in the pre-processor of Fast Downward.

STRIPS-2-FDR Naïve vs. Clever: Simplified "TSP"



- STRIPS P: $at(x), visited(x)$ for $x \in \{Sydney, Adelaide, Brisbane\}$.
- Naïve V: $at(x), visited(x) : \{T, F\}$ for $x \in \{Sydney, Adelaide, Brisbane\}$.
- Clever V: $at : \{Sydney, Adelaide, Brisbane\}$; $visited(x) : \{T, F\}$ for $x \in \{Sydney, Adelaide, Brisbane\}$.

→ The naïve version is merely STRIPS in disguise. The clever version is more natural, and is explicit about the "truck position".

Action Description Language (ADL)

Framework Definition: [Pednault (1989); Hoffmann and Nebel (2001)].

Problem: Like STRIPS but with first-order logic (FOL) formulas in pre_a and G, and conditional effects that execute only if their individual effect condition holds.

Plan: Sequence of actions. (Yes, this is still "classical planning".)

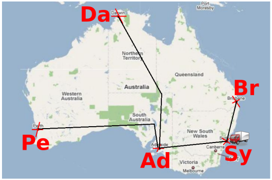
Example: If your action a opens the doors of an elevator, then each passenger gets out iff their individual condition ("Is this my destination floor?") holds. If you want to satisfy complex constraints ("Group A should never meet group B in the elevator") then pre_a gets nasty. (See the file miconic-ADL on Moodle.)

Compilation: FOL formulas: Ground them (the universe is finite) and transform to DNF [Gazen and Knoblock (1997); Koehler and Hoffmann (2000)].

Conditional effects: Either enumerate all combinations of effects, or introduce artificial facts/actions enforcing an "effect evaluation phase" [Nebel (2000)].

State of the art: Get rid of FOL formulas but keep the conditional effects [Hoffmann and Nebel (2001)].

Numeric and Temporal Planning



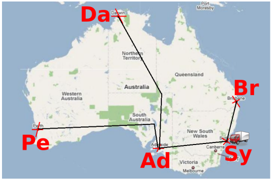
Numeric Planning: [Fox and Long (2003)]
$pre_a : fuelSupply \geq distance(x, y) * fuelConsumption$
$eff_a : fuelSupply := fuelSupply - distance(x, y) * fuelConsumption$

Compilation: Nothing known.

Temporal Planning: [Fox and Long (2003)]
$duration_a : distance(x, y)/speed$
eff_a : at Start $\neg at(x)$, at End $at(y)$.

Compilation: Ignore durations during search, schedule plan as a post-process [Edelkamp (2003)]. Competitive with state of the art!

Soft Goals and Trajectory Constraints



Soft Goals: [Gerevini *et al.* (2009)]

"I don't absolutely have to visit Darwin, but if I do, I get a certain amount R of reward."

Compilation: Artificial actions that allow to forgo each weak goal, at cost R; minimize cost [Keyder and Geffner (2009)]. State of the art!

Trajectory Constraints: [Gerevini *et al.* (2009)]

"I must visit Perth before I visit Darwin."

Compilation: Artificial preconditions/effects, e.g. $visited(Perth)$ into precondition of driving to Darwin [Edelkamp (2006)]. State of the art!

Conformant Planning

Framework Definition: [Smith and Weld (1998); Bonet and Givan (2006)].

Problem: There are many possible initial states (represented as a formula), and each action may have several possible effects. We have no observability during plan execution.

Plan: Sequence of actions that achieves the goal regardless which initial state and action effects occur.

Example: You're in a dark cave but don't know where exactly. The plan is to walk to the right until you reach a wall and can locate yourself. Then navigate to your goal by counting your steps.

Compilation: Artificial "what-if" facts, like "If I was at A initially, then I am now at B" [Palacios and Geffner (2009)]. State of the art!

Contingent Planning

Framework Definition: e.g., [Hoffmann and Brafman (2005)].

Problem: There are many possible initial states (represented as a formula), and each action may have several possible effects. We have partial observability during plan execution.

Plan: Tree of actions that achieves the goal in each of its leaves. ("Plan ahead for all possible contingencies, i.e., situation aspects not known at planning time.")

Example: Solving the Wumpus world: You walk some steps, then use sensing (for breeze and stench), and continue depending on the outcome.

Compilation: Sample initial states, classical planning with artificial facts encoding knowledge yields a plan tree for those; in case a problem is detected during execution, re-plan with the new state of knowledge [Shani and Brafman (2011)]. Competitive with state of the art!

Probabilistic Planning

Framework Definition: e.g., [Younes *et al.* (2005)].

Problem: Each action specifies a probability distribution over its possible effects. We have full observability during plan execution. (Markov Decision Process (MDP) framework.)

Plan: Policy that maps states to actions in a way that maximizes the expected reward.

Example: Controlling a robot: If navigation comes with an imprecision (which it usually does), then the outcome of a "move" operation is uncertain.

Compilation: Make classical problem that acts as if you could *choose* the outcomes; find a plan, and execute; if the plan fails, then re-plan from the current state [Yoon *et al.* (2007)]. State of the art for problems where "reactive behavior" is suitable (things may go wrong, but if they do, they can be easily repaired).

Summary

- Transition systems are a kind of directed graph (typically huge) that encode how the state of the world can change.
- Planning tasks are compact representations for transition systems, based on state variables; they are the input for planning systems.
- In satisficing planning, we must find a solution to planning tasks (or show that no solution exists). In optimal planning, we must additionally guarantee that generated solutions are the cheapest possible.
- Classical planning makes strong simplifying assumptions, but is very successful in practice and can be used by compilation to tackle more expressive planning problems.
- In STRIPS, state variables are Boolean; in FDR, they may have arbitrary finite domains. The two formalisms can be compiled into each other. FDR is preferrable, but current planning technology is based on STRIPS for historical reasons.

  $\rightarrow$ PDDL, see **Next Chapter**.

Remarks

Regarding the name "FDR":

- FDR is not consistently named in the literature.
- It is often referred to as SAS$^+$ because that's what some complexity guys called it, in the first papers considering a formalism equivalent to our FDR [e.g., Bäckström and Nebel (1995)].
- [Helmert (2006)] called it multi-valued planning tasks (MPT) which can still be seen in some papers.
- [Helmert (2009)] finally called it FDR.

Reading

- *Concise Finite-Domain Representations for PDDL Planning Tasks* [Helmert (2009)].

  Available on Moodle

  Content: Describes in detail the "clever" STRIPS-2-FDR conversion implemented in Fast Downward. The sets $\{p_1, \dots, p_k\}$ of STRIPS facts, of which exactly one is true in every reachable state, are found by automatic invariance analysis. Is in wide-spread use, and a basic familiarity with it is relevant for anybody working in planning.

References I

Christer Bäckström and Bernhard Nebel. Complexity results for SAS$^+$ planning. *Computational Intelligence*, 11(4):625–655, 1995.

Blai Bonet and Robert Givan. 5th international planning competition: Non-deterministic track – call for participation. In *Proceedings of the 5th International Planning Competition (IPC'06)*, 2006.

Tom Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204, 1994.

Stefan Edelkamp. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research*, 20:195–238, 2003.

Stefan Edelkamp. On the compilation of plan constraints and preferences. In Derek Long and Stephen Smith, editors, *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS'06)*, pages 374–377, Ambleside, UK, 2006. AAAI Press.

Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

Michael R. Garey and David S. Johnson. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA, 1979.

References II

B. Cenk Gazen and Craig Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In S. Steel and R. Alami, editors, *Proceedings of the 4th European Conference on Planning (ECP'97)*, pages 221–233. Springer-Verlag, 1997.

Alfonso Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

Malte Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173:503–535, 2009.

Jörg Hoffmann and Ronen Brafman. Contingent planning via heuristic forward search with implicit belief states. In Susanne Biundo, Karen Myers, and Kanna Rajan, editors, *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*, pages 71–80, Monterey, CA, USA, 2005. AAAI Press.

References III

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

Emil Keyder and Hector Geffner. Soft goals can be compiled away. *Journal of Artificial Intelligence Research*, 36:547–556, 2009.

Jana Koehler and Jörg Hoffmann. On the instantiation of ADL operators involving arbitrary first-order formulas. In *Proceedings ECAI-00 Workshop on New Results in Planning, Scheduling and Design*, 2000.

Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000.

Hector Palacios and Hector Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research*, 35:623–675, 2009.

Edwin P.D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In R. Brachman, H. J. Levesque, and R. Reiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 1st International Conference (KR-89)*, pages 324–331, Toronto, ON, May 1989. Morgan Kaufmann.

References IV

Guy Shani and Ronen I. Brafman. Replanning in domains with partial information and sensing actions. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, pages 2021–2026. AAAI Press/IJCAI, 2011.

D. E. Smith and D. Weld. Conformant Graphplan. In Jack Mostow and Charles Rich, editors, *Proceedings of the 15th National Conference of the American Association for Artificial Intelligence (AAAI'98)*, pages 889–896, Madison, WI, USA, July 1998. MIT Press.

Sung Wook Yoon, Alan Fern, and Robert Givan. FF-Replan: a baseline for probabilistic planning. In Mark Boddy, Maria Fox, and Sylvie Thiebaux, editors, *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, pages 352–359, Providence, Rhode Island, USA, 2007. Morgan Kaufmann.

Håkan L. S. Younes, Michael L. Littman, David Weissman, and John Asmuth. The first probabilistic track of the international planning competition. *Journal of Artificial Intelligence Research*, 24:851–887, 2005.