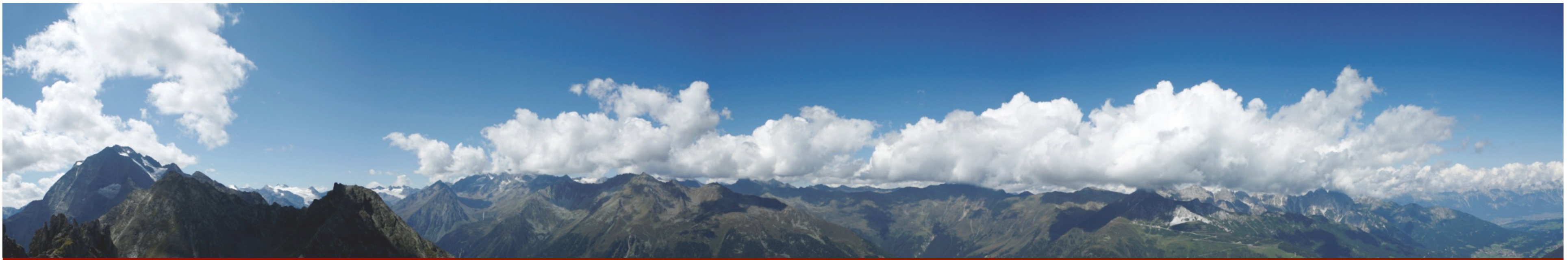
Intelligent Systems

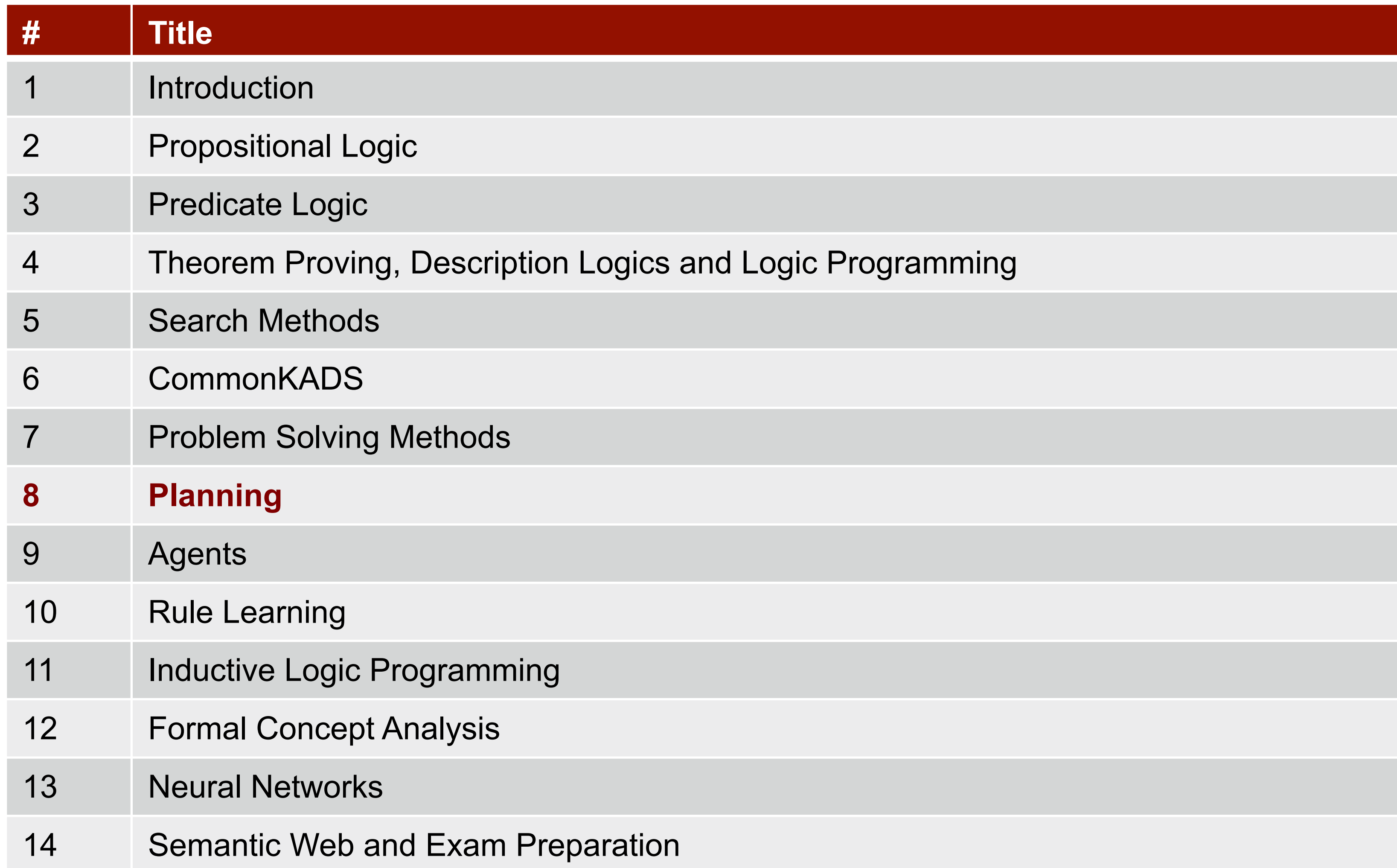
Planning



© Copyright 2009 Dieter Fensel and Barry Norton

Where are we?

STI · INNSBRUCK



#	Title
1	Introduction
2	Propositional Logic
3	Predicate Logic
4	Theorem Proving, Description Logics and Logic Programming
5	Search Methods
6	CommonKADS
7	Problem Solving Methods
8	**Planning**
9	Agents
10	Rule Learning
11	Inductive Logic Programming
12	Formal Concept Analysis
13	Neural Networks
14	Semantic Web and Exam Preparation

Agenda

- Motivation
- Technical Solution
- Illustration by a Larger Example
- Extensions
- Summary
- References

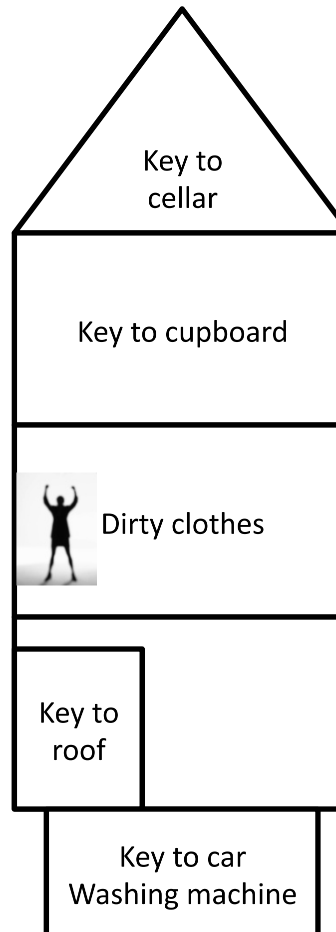
MOTIVATION

Motivation

- What is Planning?
  - *"Planning is the process of thinking about the activities required to create a desired goal on some scale"* [Wikipedia]
- We take a more pragmatic view – *planning is a flexible approach for taking complex decisions:*
  - decide about the schedule of a production line;
  - decide about the movements of an elevator;
  - decide about the flow of paper through a copy machine;
  - decide about robot actions.
- By "flexible" we mean:
  - the problem is described to the planning system in some generic language;
  - a (good) solution is found fully automatically;
  - if the problem changes, all that needs to be done is to change the description.
- We look at methods to solve any problem that can be described in the language chosen for the particular planning system.

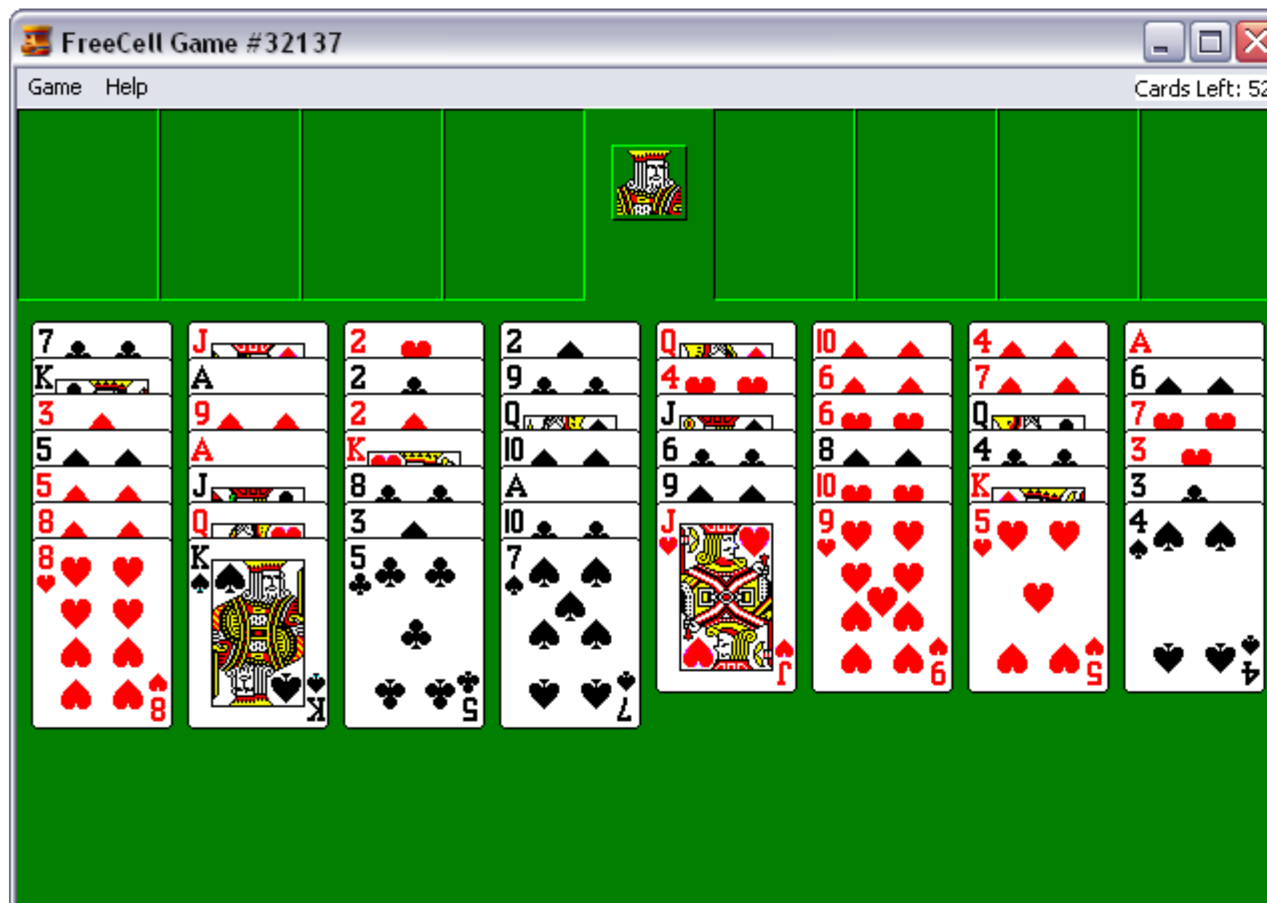
An ‘Easy’ Example

STI · INNSBRUCK



A ‘Hard’ Example

STI · INNSBRUCK



TECHNICAL SOLUTIONS

- This lecture will consider planning as state-space search, for which there are several options:
  - **Forward** from I (initial state) until G (goal state) is satisfied;
  - **Backward** from G until I is reached;
  - Use a **Heuristic** function to estimate G- or I-distance, respectively – prefer states that have a lower heuristic value.
- It will also introduce partial-order planning as a more flexible approach
- In all cases there are three implicit concerns to consider:
  - **Syntax** – how the problem/state space and goal is defined;
  - 'Semantics'/**Algorithm** – e.g. which (combination) of these approaches is used;
  - **Complexity** and **decidability** – the feasibility of the approach.

Syntax – STRIPS as Planning Formalism

STI · INNSBRUCK

- Stanford Research Institute Problem Solver [Nilsson & Fikes, 1970]
- Has only boolean variables: propositional facts
- Restricts:
  - condition and goal formulas to conjunctions of positive atoms/literals
  - effect instructions to atomic updates making atoms true or false
- Boolean variables are called facts
- Transition rules are called actions

Definition - Actions

Let P be a set of facts. A STRIPS action a is a triple a = (pre(a), add(a), del(a)) of subsets of P, where add(a) \ del(a) = Ø

pre(a), add(a), and del(a) are called the action's precondition, add list, and delete list, respectively

Syntax – STRIPS as Planning Formalism

Definition - Task

A STRIPS task is a tuple (P,A, I,G) where P is a (finite) set of facts, A is a (finite) set of STRIPS actions, and I and G are subsets of P.

Definition – Semantics I

Let P be a set of facts, $s \subseteq P$ a world state, and a a STRIPS action. The result result(s, $\langle a \rangle$) of applying (the action sequence consisting only of) a to s is:

result(s, $\langle a \rangle$) = $s \cup$ add(a)) \ del(a) iff pre(a) $\subseteq$ s; undefined otherwise

In the first case, the action is said to be applicable in s. The result of applying an action sequence $\langle a_1, \dots, a_n \rangle$ of arbitrary length to s is defined by result(s,$\langle a_1, \dots, a_n \rangle$) = result(result(s, $\langle a_1, \dots, a_{n-1} \rangle$), $\langle a_n \rangle$) and result(s, $\langle\rangle$) = s.

Definition – Plans

Let (P,A, I,G) be a STRIPS task. An action sequence $\langle a_1, \ldots, a_n \rangle$ is a plan for the task iff $G \subseteq result(I, \langle a_1, \ldots, a_n \rangle)$.

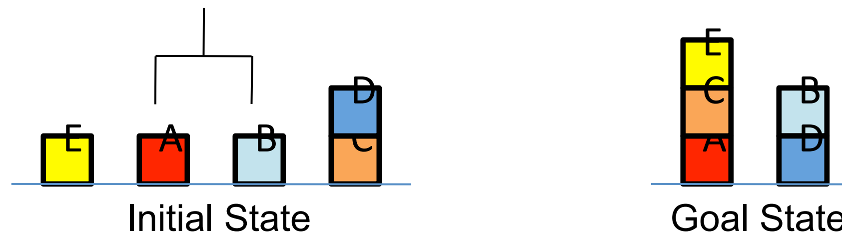
- We also call plans solutions
- The task is called solvable if there is a plan, unsolvable otherwise

Definition – Minimal and Optimal Plans

Let (P,A, I,G) be a STRIPS task. A plan $\langle a_1, \ldots, a_n \rangle$ for the task is minimal if $\langle a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n \rangle$ is **not** a plan for any i. The plan is optimal if there is **no** plan with less than n actions.

Syntax – STRIPS as Planning Formalism

Blocks World Example:



- State variables: on(x) : {NIL, Table,A,B,C,D,E}, clear (x) : {0, 1}, holding () : {NIL,A,B,C,D,E}
- Boolean variables: on(x, y), clear (x), holding(x)
- Actions: stack(x, y), unstack(x, y), putdown(x), pickup(x)
- Initial state: {on(E, Table), clear (E), . . . , on(C, Table), on(D,C), clear (D), holding(NIL)}
- stack(x, y) : ({holding(x), clear (y)}, {on(x, y), holding(nil), clear (x)}, {holding(x), clear (y)})

Syntax – PDDL as Planning Formalism

- Planning Domain Description Language
- Based on STRIPS with various extensions
- Created, in its first version, by Drew McDermott and others
- Used in the biennial International Planning Competition (IPC) series
- The representation is lifted, i.e., makes use of variables these are instantiated with objects
- Actions are instantiated *operators*
- Facts are instantiated *predicates*
- A task is specified via two files: the *domain* file and the *problem* file
- The problem file gives the objects, the initial state, and the goal state
- The domain file gives the predicates and the operators; these may be re-used for different problem files
- The domain file corresponds to the transition system, the problem files constitute instances in that system

Blocks World Example domain file:

```
(define (domain blocksworld)
(:predicates (clear ?x)
(holding ?x)
(on ?x ?y))
(    :action stack
     :parameters (?ob ?underob)
     :precondition (and (clear ?underob) (holding ?ob))
     :effect (and (holding nil) (on ?ob ?underob)
                  (not (clear ?underob)) (not (holding ?ob)))
)
…
```

Syntax – PDDL as Planning Formalism

Blocks World Example problem file:

```
(define (problem bw-xy)
    (:domain blocksworld)
    (:objects nil table a b c d e)
    (:init (on a table) (clear a)
           (on b table) (clear b)
           (on e table) (clear e)
           (on c table) (on d c) (clear d)
           (holding nil)
    )
    (:goal (and (on e c) (on c a) (on b d))))
```

- We now present particular algorithms related to (direct) state-space search:
  - Progression-based Search
  - Regression-based Search
  - Heuristic Functions and Informed Search Algorithms

Definition – Search Scheme

A search scheme is a tuple $(S, s_0, succ, Sol)$:

1. the set S of all search states $s \in S$,
2. the start state $s_0 \in S$,
3. the successor state function $succ : S \rightarrow 2^S$, and
4. the solution states $Sol \subseteq S$.

Note:

- $succ(s)$ is finite for all s
- Solution paths $s_0 \rightarrow, \ldots, \rightarrow s_n \in Sol$ (easily) correspond to solutions to our problem
- The search space is the directed graph implicitly given by s_0 and $succ$

Semantics – Progression versus Regression

Progression:

- Also called *forward search*
- Start at the initial state, apply transition rules, stop when solution state is reached

Regression:

- Also called *backward search*
- Start at the goal formula, step backwards over transition rules that can make the formula TRUE, stop when initial state satisfies the formula

Semantics – Progression Algorithms

- State space: all world states that are reachable from the initial state by applying transition rules
- In progression, search space coincides with state space
- Progression explores only world states that are reachable from *I*; they may not be relevant for *G*

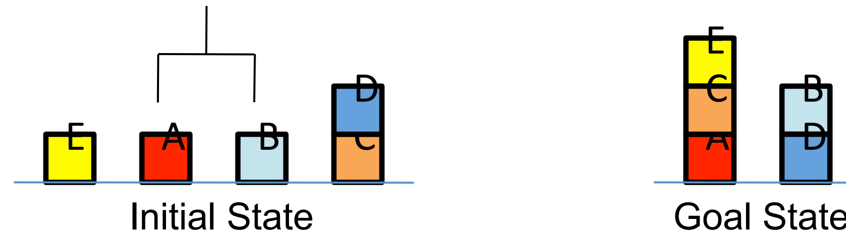
Definition – Progression in STRIPS

Let (P,A,I,G) be a STRIPS task. Progression is the tuple (S, s_0, succ, Sol):

1. $S = 2^P$ is the set of all subsets of P,
2. $s_0 = I$,
3. succ : $S \rightarrow 2^S$ is defined by $succ(s) = \{s_0 \in S \mid \exists a \in A: pre(a) \subseteq s, s_0 = result(s, \langle a \rangle)\}$
4. $Sol = \{s \in S \mid G \subseteq s\}$

Semantics – Progression in STRIPS

Blocks World Example:



- s_0 = I = {on(E, Table), clear(E), . . . , on(C, Table), on(D,C), clear(D), holding(NIL)}
- Applicable actions: pickup(E), pickup(A), pickup(B), unstack(D,C)) |succ (s_0)| = 4
- E.g., result(s_0, ⟨unstack(D,C)⟩) = {on(E, Table), clear(E), . . . , on(C, Table), clear(C), holding(D)}
- Initially, only ??? is relevant

Semantics – Regression Algorithms

- Search states are formulas ϕ; start with the goal formula
- For search state ϕ, for all transition rules t, generate a new formula ψ such that ϕ holds when applying t to a world state in which ψ holds
- "If I have ψ, then, using t, I can achieve ϕ"
- "Regress ϕ through t"
- When ψ holds in the initial state, we can stop
- Regression explores only world states that are relevant for G; they may not be reachable from I
- Regression is not as non-natural as you might think: if you plan a trip to Thailand, do you start with thinking about the train to Frankfurt?
- In general, regression involves dealing with arbitrary formulas, and that can be hard...

Definition – Regression in STRIPS

Let P be a set of facts, $s \subseteq P$, and a a STRIPS action. The regression regress(s, a) of s through a is:

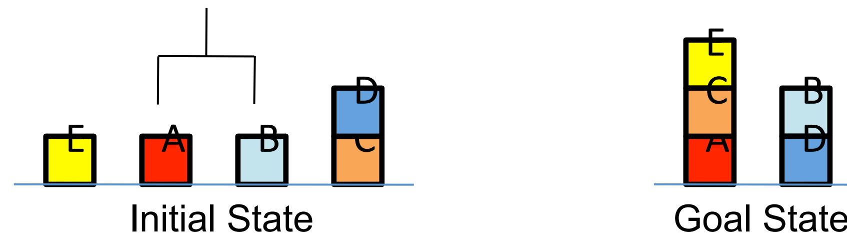
regress(s, a) = $(s \setminus add(a)) \cup pre(a)$ if $add(a) \cap s \neq \emptyset$, $del(a) \cap s = \emptyset$; undefined otherwise

In the first case, s is said to be regressable through a.

- If $s \setminus add(a) = \emptyset$; then a contributes nothing
- If $s \setminus del(a) \neq \emptyset$; then we can't use a to achieve $\bigwedge_{p \in s} p$

Semantics – Regression in STRIPS

Blocks World Example:



- G = {on(E,C), on(C,A), on(B,D)}
- stack(B,D) : ({holding(B), clear(D)}, {on(B,D), holding(nil), clear(B)}, {holding(B), clear(D)})
- regress(G, stack(B,D)) =???

Definition – Heuristic Function

Let $(S, s_0, succ, Sol)$ be a search scheme. A heuristic function is a function $h : S \rightarrow N_0 \cup \{\infty\}$ from search states into the natural numbers including 0 and the ∞ symbol. Heuristic functions, or heuristics, are efficiently computable functions that estimate a state's "solution distance".

Definition – Solution Distance

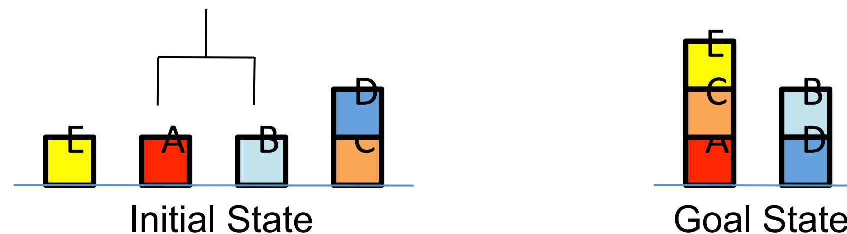
Let $(S, s_0, succ, Sol)$ be a search scheme. The solution distance sd(s) of a state $s \in S$ is the length of a shortest succ path from s to a state in Sol, or 1 if there is no such path. Computing the real solution distance is as hard as solving the problem itself.

Definition – Admissable Heuristic Function

Let $(S, s_0, succ, Sol)$ be a search scheme. A heuristic function h is admissible if $h(s) \leq sd(s)$ for all $s \in S$.

Semantics – Blocksworld Heuristic Example

Blocks World Solution Distance Example:



Progression search

sd(I) = 8, sd(result(I, ⟨unstack(D,C)⟩)) = 7, sd(s) = ??? for all other s ∈ succ (I)

Regression search:

sd(G) = 8, sd(regress(G, stack(E,C))) = 7, sd(regress(G, stack(B,D))) = 7, sd(regress(G, stack(C,A))) =???

Trade Offs in Informed Search

Search time:

- Reasonable quality heuristic: greedy faster
- Very bad quality heuristic: A faster
- Usually, greedy is faster

Solution quality:

- The greedier, the (potentially) worse

So: usually, trade-off between runtime and solution quality

Sometimes the heuristic is so bad that any global search is just like blind search; use a local search instead

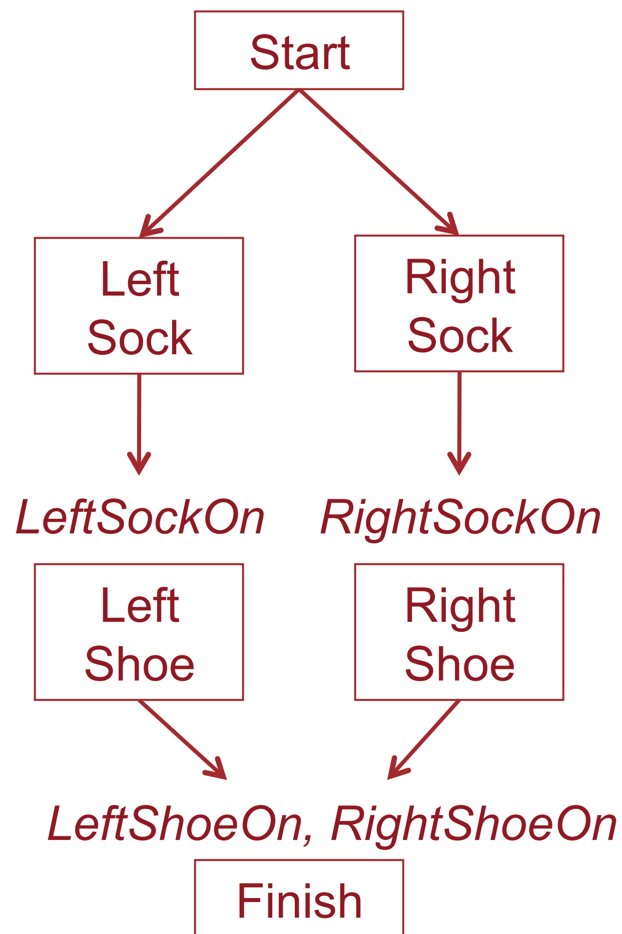
Partial-Order Planning

- Forward and backward state-space search are both forms of *totally-ordered* plan search – explore linear sequences of actions from initial state or goal.
- As such they cannot accomodate *problem decomposition* and work on subgraphs separately.
- A *partial-order planner* depends on:
  - A set of **ordering constraints**, wherein A < B means that A is executed some time before B (not necessarily directly before)
  - A set of **causal links**, A –p-> B, meaning A achieves p for B
  - A set of **open preconditions** not achieved by any action in the current plan

Partial-Order Plans

STI · INNSBRUCK

- Partial Order Plan:



- Total Order Plans:

- Its *time complexity*: in the worst case, how many search states are expanded?

- Its *space complexity*: in the worst case, how many search states are kept in the open list at any point in time?

- Is it *complete*, i.e. is it guaranteed to find a solution if there is one?

- Is it *optimal*, i.e. is it guaranteed to find an optimal solution?

ILLUSTRATION BY A LARGER EXAMPLE

Partial-Order Flat Tyre Example

STI · INNSBRUCK

We consider the application of partial-order planning to the 'spare tyre problem', characterised as follows:

Init(At(Flat, Axle) ∧ At(Spare, Boot))

Goal(At(Spare, Axle))

Action(Remove(Spare, Boot)),
PRECOND: At(Spare, Boot)
EFFECT: ¬ At(Spare, Book) ∧
At(Spare, Ground))

Action(Remove(Flat, Axle)),
PRECOND: At(Flat, Axle)
EFFECT: ¬ At(Flat, Axle) ∧
At(Flat, Ground))

Action(PutOn(Spare, Axle)),
PRECOND: At(Spare, Ground) ∧
¬ At(Flat, Axle)
EFFECT: ¬ At(Spare, Ground) ∧
At(Spare, Axle))

Action(LeaveOvernight,
PRECOND:
EFFECT: ¬ At(Spare, Ground) ∧
¬ At(Spare, Axle) ∧
¬ At(Spare, Boot) ∧
¬ At(Flat, Ground) ∧
¬ At(Flat, Axle))

Partial-Order Flat Tyre Step 1

Step 1:

a) Pick the only open precondition (At(Spare, Axle)) due to the goal;

b) Choose the only applicable action:

Action(PutOn(Spare, Axle)),
PRECOND: At(Spare, Ground) ∧
¬ At(Flat, Axle)
EFFECT: ¬ At(Spare, Ground) ∧
At(Spare, Axle))

Partial-Order Flat Tyre Step 2

STI · INNSBRUCK

Step 1:

a) Pick the only open precondition (At(Spare, Axle)) due to the goal;

b) Choose the only applicable action:

Action(PutOn(Spare, Axle)),
PRECOND: At(Spare, Ground) ∧
¬ At(Flat, Axle)
EFFECT: ¬ At(Spare, Ground) ∧
At(Spare, Axle))

Step 2:

a) Pick the At(Spare, Ground) precondition of PutOn(Spare, Axle)

b) Choose the only applicable action:

Action(Remove(Spare, Boot)),
PRECOND: At(Spare, Boot)
EFFECT: ¬ At(Spare, Book) ∧
At(Spare, Ground))

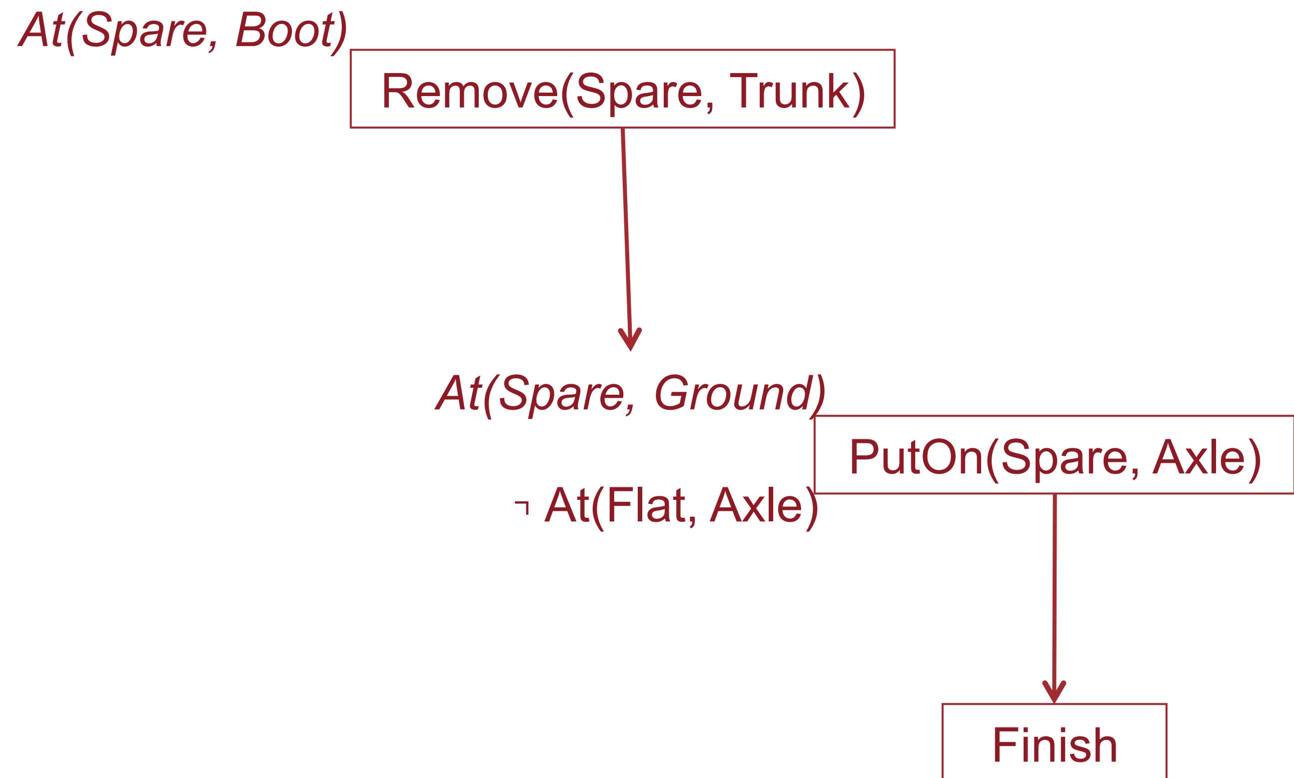
Partial-Order Plan for Flat Tyre after Step 2

STI · INNSBRUCK

Start

At(Spare, Boot)

At(Flat, Axle)



Partial-Order Flat Tyre Step 3

STI · INNSBRUCK

Step 3:

a) Pick the ¬ At(Flat, Axle) precondition of PutOn(Spare, Axle).

b) Choose the LeaveOvernight action, but notice that it has the effect ¬ At(Spare, Ground), which conflicts with the causal link

Remove(Spare, Boot) —At(Spare, Ground)→ PutOn(Spare, Axle)

To resolve this requires the addition of an ordering constraint so that LeaveOvernight preceeds Remove(Spare, Boot).

However, since At(Spare, Boot) is the only remaining precondition, the only further step would be to connect the initial state directly to Remove(Spare, Boot) but then this constraint would be violated.

Therefore we backtrack and try another Step 3…

Partial-Order Flat Tyre Step 3'

Step 3':

a) Consider again the ¬ At(Flat, Axle) precondition of PutOn(Spare, Axle).
b) This time, choose the action:

Action(Remove(Flat, Axle)),
PRECOND: At(Flat, Axle)
EFFECT: ¬ At(Flat, Axle) ∧
At(Flat, Ground))

Partial-Order Flat Tyre Step 4

Step 3':

a) Consider again the ¬ At(Flat, Axle) precondition of PutOn(Spare, Axle).
b) This time, choose the action:

Action(Remove(Flat, Axle)),
    PRECOND: At(Flat, Axle)
    EFFECT: ¬ At(Flat, Axle) ∧
            At(Flat, Ground))

Step 4:

a) Pick the At(Spare, Boot) precondition of PutOn(Spare, Axle)
b) Choose Start to achieve it

Partial-Order Flat Tyre Step 5

Step 3':

a) Consider again the ¬ At(Flat, Axle) precondition of PutOn(Spare, Axle).
b) This time, choose the action:

Action(Remove(Flat, Axle)),
PRECOND: At(Flat, Axle)
EFFECT: ¬ At(Flat, Axle) ∧
At(Flat, Ground))

Step 4:

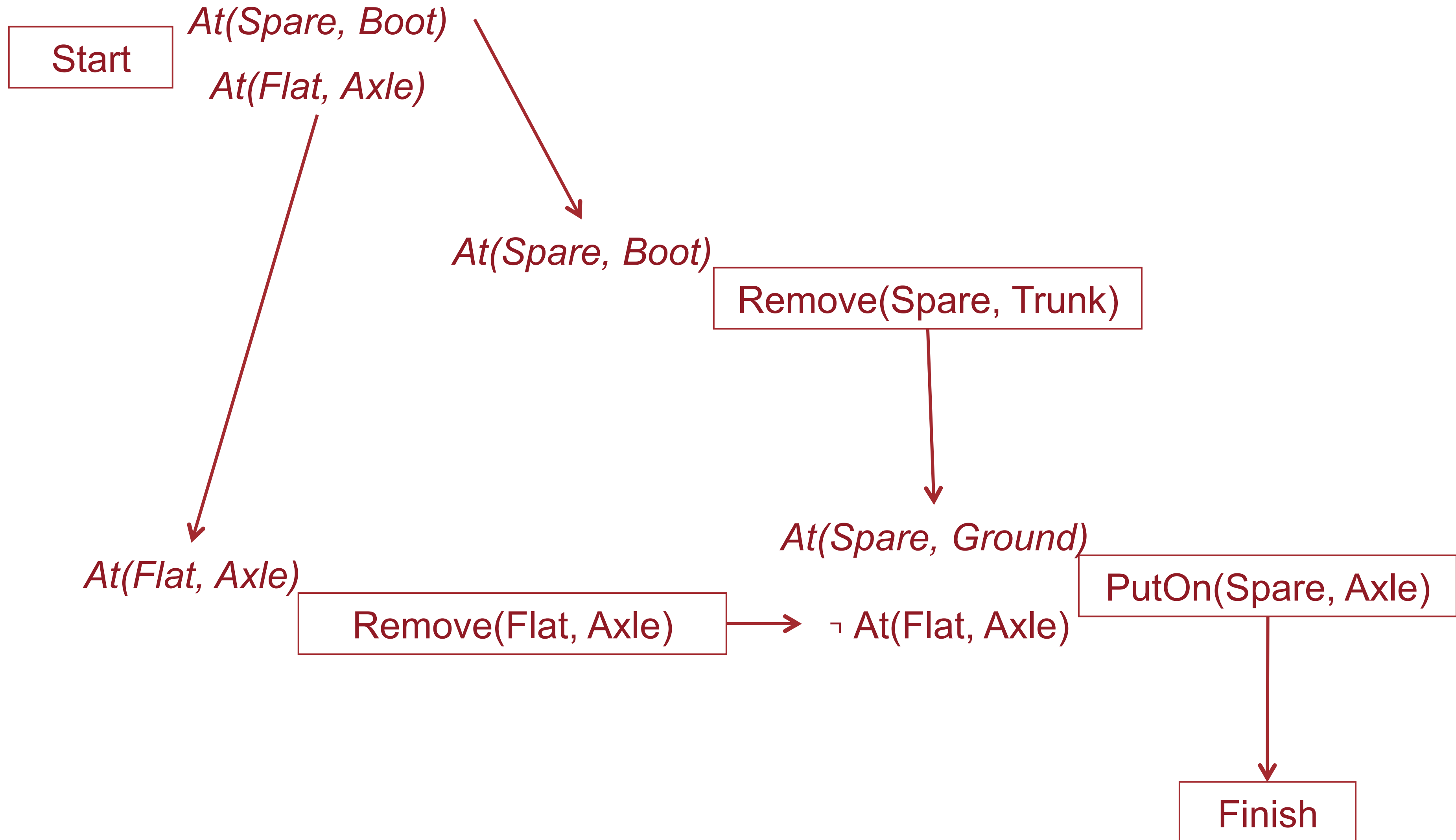
a) Pick the At(Spare, Boot) precondition of PutOn(Spare, Axle)
b) Choose Start to achieve it

Step 5:

a) Pick the remaining At(Flat, Axle) precondition of Remove(Flat, Axle)
b) Choose Start to achieve this (leaving no preconditions)

Final Partial-Order Plan for Flat Tyre

STI · INNSBRUCK



EXTENSIONS

- C*onstrain* the task by a time horizon, *b*
- Formulate this as a *constraint satisfaction problem* and use *backtracking* methods to solve it: “do *x* at time *t* yes/no”, “do *y* at time *t’* yes/no”,
- If there is no solution, increment *b*

- Forward or backward *breadth-first* search
- Represent set of reached states (by time t) through its *characteristic function*
- Most wide-spread: BDDs (Binary Decision Diagrams)
- Rooted DAG, nodes either:
  - 0 sons and labelled 0/1, or
  - 2 sons – 0 and 1 – and labelled with a variable

SUMMARY

Planning Summary

- Planning is a flexible approach for taking complex decisions:
  - the problem is described to the planning system in some generic language;
  - a (good) solution is found fully automatically;
  - if the problem changes, all that needs to be done is to change the description.
- Following concerns must be addressed:
  - Syntax - we have examined representations in PDDL and STRIPS
  - Semantics – we have examined progression, regression and the use of heuristics in algorithms
  - Complexity and decidability

REFERENCES

Links

[McDermott, 1998] “PDDL – the planning domain definition language”, D. McDermott, Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[Nilsson & Fikes, 1970]

“STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving”, N. Nilsson and R. Fikes, SRI Technical Note 43, 1970

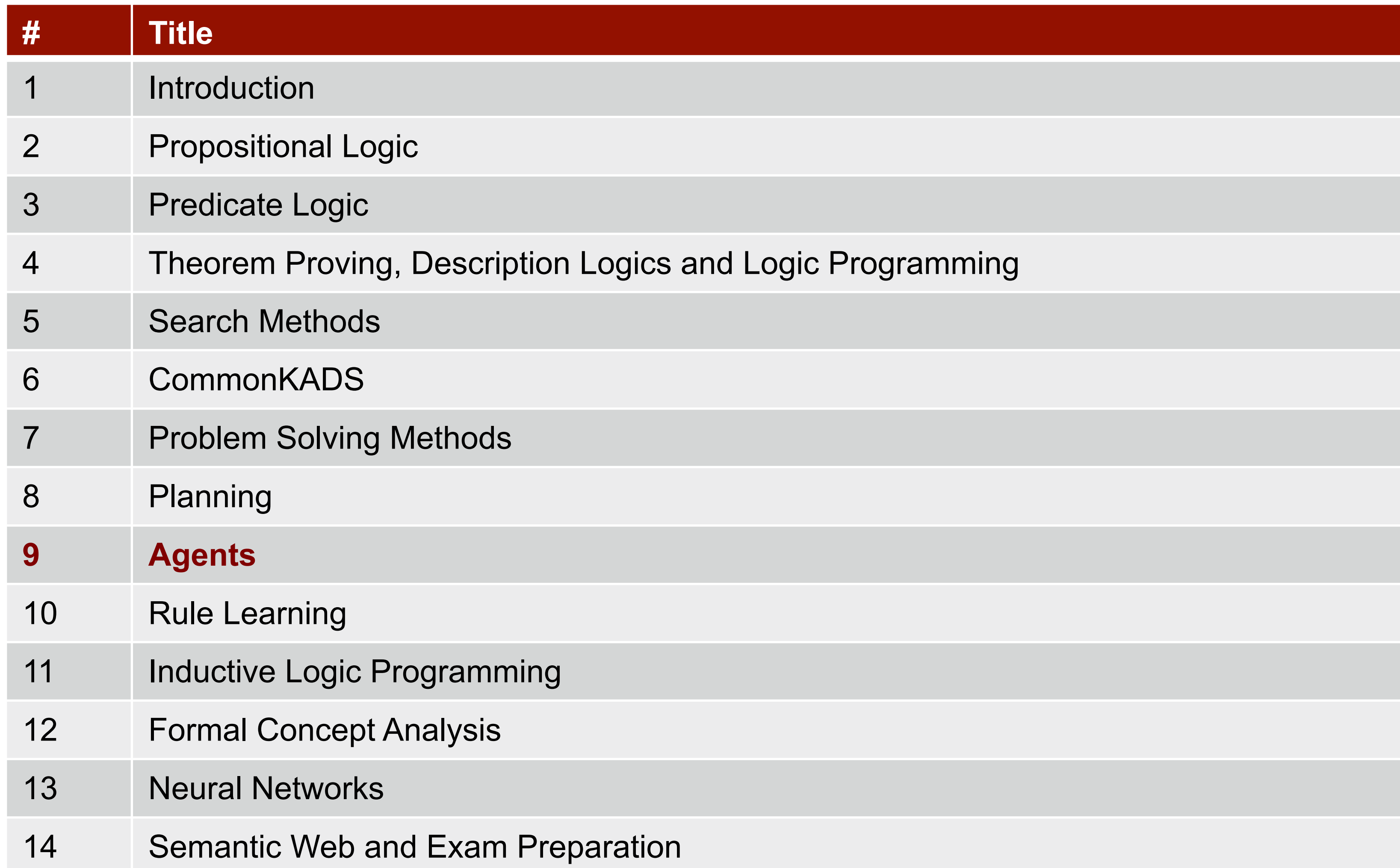
http://www.ai.sri.com/pubs/files/tn043r-fikes71.pdf

[Russell & Norvig, 2002]

“AI: A Modern Approach” (2nd Edition), S. Russell and P. Norvig, Prentice Hall, 2002

Next Lecture

STI · INNSBRUCK



#	Title
1	Introduction
2	Propositional Logic
3	Predicate Logic
4	Theorem Proving, Description Logics and Logic Programming
5	Search Methods
6	CommonKADS
7	Problem Solving Methods
8	Planning
9	**Agents**
10	Rule Learning
11	Inductive Logic Programming
12	Formal Concept Analysis
13	Neural Networks
14	Semantic Web and Exam Preparation

Questions?