

Automatic Planning

Jörg Hoffmann

University of Innsbruck
DERI

Masterstudium Informatik Vertiefungsmodul (WM 9)

Automatic Planning

2. Planning Formalisms

- ▶ **Why STRIPS Planning?**
- ▶ Preface: Propositional Formulas
- ▶ STRIPS Planning
- ▶ The Complexity of STRIPS Planning
- ▶ PDDL
- ▶ Numeric STRIPS

Why STRIPS Planning?

STRIPS is the simplest possible (reasonably expressive) transition system description language.

- ▶ STRIPS has only Boolean variables: propositional facts
- ▶ We keep in mind the generality of the approaches
- ▶ We also consider some extensions beyond STRIPS

- ▶ STRIPS = Stanford Research Institute Problem Solver

Automatic Planning

2. Planning Formalisms

- ▶ Why STRIPS Planning?
- ▶ Preface: Propositional Formulas
- ▶ STRIPS Planning
- ▶ The Complexity of STRIPS Planning
- ▶ PDDL
- ▶ Numeric STRIPS

Definition

Let P be a set of propositions. A propositional formula ϕ is either:

1. a proposition $p \in P$, or
 2. the **negation** $\neg\phi'$ of a propositional formula ϕ'
 3. the **conjunction** $\phi_1 \wedge \phi_2$ of propositional formulas ϕ_1 and ϕ_2 , or
 4. the **disjunction** $\phi_1 \vee \phi_2$ of propositional formulas ϕ_1 and ϕ_2 .
- ▶ A proposition is either 1 or 0 in an “interpretation” (== value assignment == state)
 - ▶ Formulas of the form p and $\neg p$ are called **literals**

Definition

Let P be a set of propositions, ϕ a propositional formula, i an interpretation of P . ϕ **holds in i** , written $i \models \phi$, iff either:

1. $\phi = p \in P$ and $i(p) = 1$, or
2. $\phi = \neg\phi'$ and ϕ' does not hold in i , or
3. $\phi = \phi_1 \wedge \phi_2$ and both ϕ_1 and ϕ_2 hold in i , or
4. $\phi = \phi_1 \vee \phi_2$ and at least one of ϕ_1 and ϕ_2 holds in i .

- ▶ The interpretation i will be our system state

$P = \{\text{doof}, \text{mittel}, \text{schlau}\}$

$i = \{\text{doof} \mapsto 0, \text{mittel} \mapsto 1, \text{schlau} \mapsto 0\}$

1. $\text{doof} \wedge \neg \text{mittel} \wedge \neg \text{schlau}$
2. $\text{schlau} \vee \neg \text{schlau}$
3. $(\text{doof} \vee \text{mittel} \vee \text{schlau}) \wedge$
 $(\neg \text{doof} \vee \neg \text{mittel}) \wedge$
 $(\neg \text{mittel} \vee \neg \text{schlau}) \wedge$
 $(\neg \text{doof} \vee \neg \text{schlau})$
4. $(\neg \text{doof} \vee \text{mittel}) \wedge (\neg \text{mittel} \vee \text{schlau}) \wedge (\neg \text{schlau} \vee \text{doof}) \wedge$
 $(\text{doof} \vee \text{schlau}) \wedge (\neg \text{mittel} \vee \neg \text{schlau})$

Automatic Planning

2. Planning Formalisms

- ▶ Why STRIPS Planning?
- ▶ Preface: Propositional Formulas
- ▶ **STRIPS Planning**
- ▶ The Complexity of STRIPS Planning
- ▶ PDDL
- ▶ Numeric STRIPS

- ▶ Only Booleans
- ▶ STRIPS restricts
 - ▶ (condition and goal) formulas to conjunctions of positive (non-negated) atoms/literals
 - ▶ effect instructions to atomic updates making an atom true or false
- ▶ We resort to a relatively simple notation suitable for this special case: the notation and terminology used in the AI Planning community
- ▶ Boolean variables are called **facts**
- ▶ Transition rules are called **actions**

Definition

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 $add(a) \cap del(a) = \emptyset$.

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

Definition

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 .

- ▶ I is the initial state, G is commonly referred to as the **goal state** (which is misleading since G describes only a desirable *property* of a state that we want to reach)

STRIPS Planning, Syntax, So?

STRIPS is a shortcut/specialized notation in the following way:

- ▶ The action precondition $pre(a)$ is short for the condition formula $\bigwedge_{p \in pre(a)} p = 1$; likewise for the goal “state” G
- ▶ The action add and delete lists are short for the effect instruction $\{p := 1 \mid p \in add(a)\}, \{p := 0 \mid p \in del(a)\}$
- ▶ The initial state I is short for the value assignment $\{p \mapsto 1 \mid p \in I\} \cup \{p \mapsto 0 \mid p \in P \setminus I\}$; likewise for all **world states** s
- ▶ With that, the \models relation between states and fact conjunctions is notated as the \supseteq relation between the respective fact sets

Definition

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

$$\text{result}(s, \langle a \rangle) = \begin{cases} (s \cup \text{add}(a)) \setminus \text{del}(a) & \text{pre}(a) \subseteq s \\ \text{undefined} & \text{otherwise} \end{cases}$$

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

$$\text{result}(s, \langle a_1, \dots, a_n \rangle) = \text{result}(\text{result}(s, \langle a_1, \dots, a_{n-1} \rangle), \langle a_n \rangle)$$

and

$$\text{result}(s, \langle \rangle) = s.$$

Definition

Let (P, A, I, G) be a STRIPS task. An action sequence $\langle a_1, \dots, a_n \rangle \in A^*$ is a **plan** for the task iff $G \subseteq \text{result}(I, \langle a_1, \dots, a_n \rangle)$.

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

Definition

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

Are these plans minimal? Are they optimal?

1. drive-IBK-Kufstein, drive-Kufstein-Munich
2. drive-IBK-Vienna, buy-map, drive-Vienna-Munich
3. drive-IBK-Vienna, buy-map, buy-map, drive-Vienna-Munich

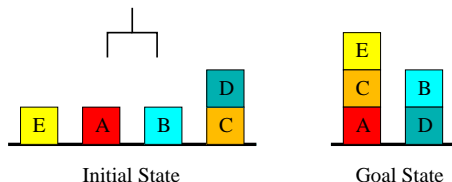
... making plans *minimal* is easy.

- ▶ The reachability problem: given a STRIPS task, determine whether it is solvable
- ▶ As said, we will concentrate mainly on the methods that have been found successful for *finding* plans in *solvable* tasks
- ▶ We would, ideally, be interested in finding optimal plans
- ▶ Unsurprisingly, it will turn out that this is much harder in practice (and in theory) than finding just some plan

STRIPS Planning, Remarks II

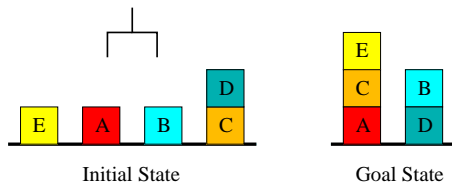
- ▶ A finite-domain (non-Boolean) variable with k different values can be simulated with $\lceil \log_2 k \rceil$ Boolean variables/facts
- ▶ *Logarithmic encoding vs. linear encoding*
- ▶ $dom(v) = \{0, 1, 2, 3\} \implies$ linear: $v-eq-0$, $v-eq-1$, $v-eq-2$, $v-eq-3$; logarithmic: $v-lo-bit$, $v-hi-bit$
- ▶ Linear is more understandable!
- ▶ In practice, linear does *not* necessarily imply an efficiency loss:
 - ▶ ... the state spaces are the same
 - ▶ ... and many heuristic functions yield bad estimates on logarithmic encodings!

(Oh no it's) the Blockworld



- ▶ State variables: $on(x) : \{NIL, Table, A, B, C, D, E\}$,
 $clear(x) : \{0, 1\}$, $holding() : \{NIL, A, B, C, D, E\}$
- ▶ on and $holding$ suffice to describe the world state; $clear$ helps to formulate action conditions
- ▶ Actions: $stack(x, y)$, $unstack(x, y)$, $putdown(x)$, $pickup(x)$
- ▶ E.g.: $stack(x, y) : holding() = x \wedge clear(y) = 1 \longrightarrow$
 $on(x) := y, holding() := NIL, clear(y) := 0, clear(x) := 1$


The Blockworld, Linear Encoding



- ▶ (State variables: $on(x) : \{NIL, Table, A, B, C, D, E\}$, $clear(x) : \{0, 1\}$, $holding() : \{NIL, A, B, C, D, E\}$)
- ▶ Boolean variables: $on(x, y)$, $clear(x)$, $holding(x)$
- ▶ Initial state: $\{on(E, Table), clear(E), \dots, on(C, Table), on(D, C), clear(D), holding(NIL)\}$
- ▶ $stack(x, y) : (\{holding(x), clear(y)\}, \{on(x, y), holding(nil), clear(x)\}, \{holding(x), clear(y)\})$

The 8-Puzzle

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 6 | | 9 |
| 4 | 7 | 8 |




| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | | 6 |
| 7 | 8 | 9 |

- ▶ State variables: $at(t_i)$ for $i \in \{0, 1, \dots, 9\} \setminus \{5\}$ ($0 = blank$),
 $dom = \{p_1, \dots, p_9\}$
- ▶ Actions: e.g., $mv(t_6, p_4, p_5) : at(t_6) = p_4 \wedge at(t_0) = p_5 \longrightarrow$
 $at(t_6) := p_5, at(t_0) := p_4$
- ▶ Initial, goal: $at(t_1) = p_1, \dots$

The 8-Puzzle in STRIPS

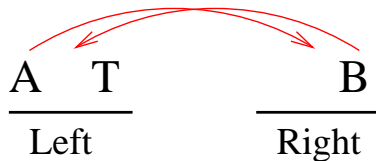
| | | |
|---|---|---|
| 1 | 2 | 3 |
| 6 | | 9 |
| 4 | 7 | 8 |



| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | | 6 |
| 7 | 8 | 9 |

- ▶ (State variables: $at(t_i)$ for $i \in \{0, 1, \dots, 9\} \setminus \{5\}$,
 $dom = \{p_1, \dots, p_9\}$)
- ▶ Boolean variables: $at(t_i, p_j)$ for $i \in \{0, 1, \dots, 9\} \setminus \{5\}$,
 $j \in \{1, \dots, 9\}$
- ▶ Actions: e.g., $mv(t_6, p_4, p_5) : at(t_6, p_4) \wedge at(t_0, p_5) \longrightarrow$
 $at(t_6, p_5), at(t_0, p_4), \neg at(t_6, p_4), \neg at(t_0, p_5)$
- ▶ Initial, goal: $at(t_1, p_1), \dots$

Our Play-Through Example: Logistics



- ▶ Facts: $at(A, Left)$, $at(A, Right)$, $in(A, T)$, $at(B, Left)$, $at(B, Right)$, $in(B, T)$, $at(T, Left)$, $at(T, Right)$
- ▶ Actions: $drive(t, l, l') : (\{at(t, l)\}, \{at(t, l')\}, \{at(t, l)\})$,
 $load(p, t, l) : (\{at(p, l), at(t, l)\}, \{in(p, t)\}, \{at(p, l)\})$,
 $unload(p, t, l) : ?$
- ▶ (Plus typing, i.e. distinguish truck/packages/locations)
- ▶ More generally: location set L , truck set T , package set P
(the official benchmark distinguishes “cities” and “airplanes”)

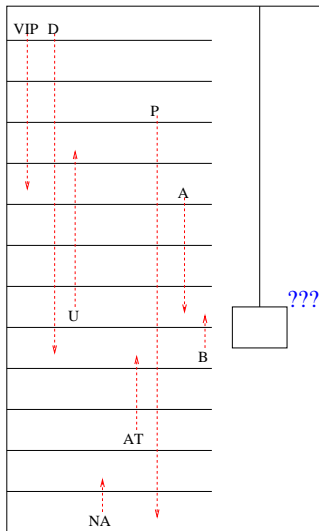
The Freecell Domain



The Freecell Domain in STRIPS

- ▶ *sendtofree*:
((*x*, *y*, *z*, *u*),
 { (card, *x*), (card, *y*), (num, *z*), (num, *u*),
 (clear, *x*), (on, *x*, *y*), (cellspace, *z*), (successor, *z*, *u*)},
 { (incell, *x*), (clear, *y*), (cellspace, *u*)},
 { (on, *x*, *y*), (clear, *x*), (cellspace, *z*)})
- ▶ *sendtofree-b*:
((*x*, *y*, *z*, *u*, *v*),
 { (card, *x*), (num, *y*), (num, *z*), (num, *u*), (num, *v*),
 (clear, *x*), (bottomcol, *x*), (cellspace, *y*),
 (successor, *y*, *z*), (colspace, *u*), (successor, *v*, *u*)},
 { (incell, *x*), (cellspace, *z*), (colspace, *v*)},
 { (bottomcol, *x*), (clear, *x*), (cellspace, *y*), (colspace, *u*)})
- ▶ *sendtonewcol*:
((*x*, *y*, *z*, *u*),
 { (card, *x*), (card, *y*), (num, *z*), (num, *u*),
 (clear, *x*), (on, *x*, *y*), (colspace, *z*), (successor, *z*, *u*)},
 { (bottomcol, *x*), (clear, *y*), (colspace, *u*)},
 { (on, *x*, *y*), (colspace, *z*)})
- ▶ *sendtohome*:
((*x*, *y*, *z*, *u*, *v*, *w*),
 { (card, *x*), (card, *y*), (suit-type, *z*), (num, *u*), (card, *v*),
 (num, *w*), (on, *x*, *y*), (clear, *x*), (home, *v*), (suit, *x*, *z*),
 (suit, *v*, *z*), (value, *x*, *u*), (value, *v*, *w*), (successor, *u*, *w*)},
 { (home, *x*), (clear, *y*)},
 { (on, *x*, *y*), (clear, *x*), (home, *v*)})
- ▶ *sendtohome-b*:
((*x*, *y*, *z*, *u*, *v*, *w*, *q*),
 { (card, *x*), (suit-type, *y*), (num, *z*), (card, *u*),
 (num, *v*), (num, *w*), (num, *q*),
 (bottomcol, *x*), (clear, *x*), (home, *u*),
 (suit, *x*, *v*), (suit, *u*, *v*), (value, *x*, *z*)}

The Miconic Domain



The Miconic Domain: “Stop” Action

```
( (x),  
  (floor, x) ∧ (lift-at, x) ∧  
  (∀y : (not-passenger, y) ∨ (not-no-access, y, x) ∨ (not-boarded, y)) ∧  
  (∀y : ((not-vip, y) ∨ (served, y)) ∨  
    ∃y : (vip, y) ∧ ((origin, y, x) ∨ (destin, y, x))) ∧  
  (∀y : (not-going-nonstop, y) ∨ (not-boarded, y) ∨ (destin, y, x)) ∧  
  (∀y : (not-never-alone, y) ∨  
    ((not-boarded, y) ∨ (destin, y, x)) ∧  
    ((served, y) ∨ (not-origin, y, x))) ∨  
  ∃z : (attendant, z) ∧  
    ((boarded, z) ∧ (not-destin, z, x)) ∨  
    ((not-served, z) ∧ (origin, z, x))) ∧  
  (∀y : (not-conflict-A, y) ∨  
    ((not-boarded, y) ∨ (destin, y, x)) ∧  
    ((served, y) ∨ (not-origin, y, x))) ∨  
  ∀y : (not-conflict-B, y) ∨  
    ((not-boarded, y) ∨ (destin, y, x)) ∧  
    ((served, y) ∨ (not-origin, y, x))),  
{ ( (y),  
  (passenger, y) ∧ (boarded, y) ∧ (destin, y, x),  
  {(served, y), (not-boarded, y)},  
  {(boarded, y), (not-served, y)}},  
  ( (y),  
  (passenger, y) ∧ (not-served, y) ∧ (origin, y, x),  
  {(boarded, y)},  
  {(not-boarded, y)})) }
```

Automatic Planning

2. Planning Formalisms

- ▶ Why STRIPS Planning?
- ▶ Preface: Propositional Formulas
- ▶ STRIPS Planning
- ▶ **The Complexity of STRIPS Planning**
- ▶ PDDL
- ▶ Numeric STRIPS

Definition

Let *PLANSAT* denote the following decision problem. Given a STRIPS task (P, A, I, G) , is the task solvable?

Theorem

PLANSAT is **PSPACE**-complete.

Proof.

Sketch: see Blackboard. Details – see [Bylander,AI-1994]. □

Example of a planning domain with exponentially long plans:
Towers of Hanoi

Definition

Let *Bounded-PLANSAT* denote the following decision problem. Given a STRIPS task (P, A, I, G) and an integer b . Is there a plan with at most b actions?

- ▶ Isn't any easier than PLANSAT: *why?*
- ▶ Isn't any harder than PLANSAT: keep a counter in non-det poly-space algorithm

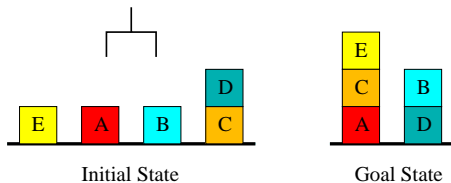
Definition

Let *Poly-Bounded-PLANSAT* denote the following decision problem. Given a STRIPS task (P, A, I, G) and an integer b bounded by a polynomial in the size of (P, A, I, G) . Is there a plan with at most b actions?

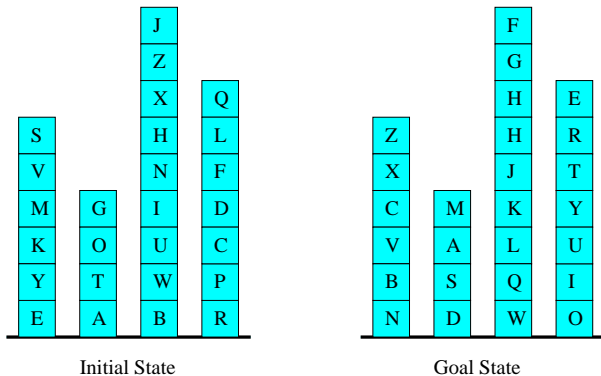
- ▶ **Is a member of NP:** *why?*
- ▶ **NP-hardness:** see exercises

- ▶ *In the general case*, they have the same complexity
- ▶ *Within applications*, bounded plan existence is often harder than plan existence
- ▶ In many benchmarks, Bounded-PLANSAT is **NP**-complete while PLANSAT is in **P**
- ▶ For example: Blocksworld, Logistics!
- ▶ Which means: *optimal* planning is (almost) never “easy” ...

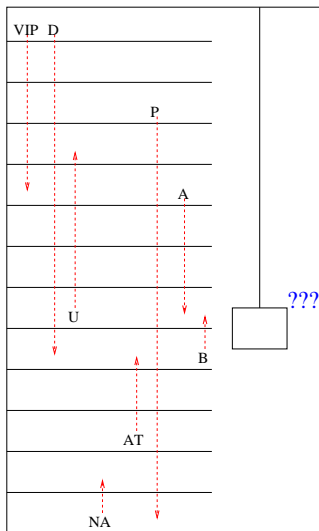
The Blocksworld is Hard?



The Blocksworld is Hard!



Miconic: Both is Hard



Automatic Planning

2. Planning Formalisms

- ▶ Why STRIPS Planning?
- ▶ Preface: Propositional Formulas
- ▶ STRIPS Planning
- ▶ The Complexity of STRIPS Planning
- ▶ **PDDL**
- ▶ Numeric STRIPS

- ▶ The *Planning Domain Description Language*
- ▶ A description language for planning in the STRIPS formalism and various extensions
- ▶ Created, in its first version, by Drew McDermott and others
- ▶ Used in the biennial *International Planning Competition (IPC)* series
- ▶ 1998: PDDL
- ▶ 2000: “PDDL subset for the 2000 competition” [Bacchus]
- ▶ 2002: PDDL2.1, levels 1-3 [Fox&Long,JAIR-2003]
- ▶ 2004: PDDL2.2 [Hoffmann&Edelkamp,JAIR-2005]
- ▶ 2006: PDDL3 [Gerevini&Long,...]

- ▶ 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

PDDL — “Domain File” Example

```
(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))))
)
```

(We will use the word “domain” to refer to different “benchmarks”)

PDDL — “Problem File” Example

```
(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))))
```

- ▶ STRIPS + *ADL*: Action Description Language:
 - ▶ Arbitrary 1st order formulas in action preconditions and the goal
 - ▶ Conditional effects, i.e. effects that only occur if their separate effect condition holds
- ▶ ADL is a real headache to implement
- ▶ The systems that *can* handle ADL *compile* it down to simpler formats [Gazen&Knoblock, ECP-1997] (specifically, STRIPS with conditional effects)
- ▶ E.g. FF: 7000 lines compilation, 2000 lines core planner

Miconic “Stop” – PDDL

```
(:action stop
:parameters (?f - floor)
:precondition (and (lift-at ?f)
  (imply
    (exists
      (?p - conflict-A)
      (or (and (not (served ?p))
              (origin ?p ?f))
          (and (boarded ?p)
              (not (destin ?p ?f))))))
    (forall
      (?q - conflict-B)
      (and (or (destin ?q ?f)
              (not (boarded ?q)))
          (or (served ?q)
              (not (origin ?q ?f))))))
    (imply (exists
      (?p - conflict-B)
      (or (and (not (served ?p))
              (origin ?p ?f))
          (and (boarded ?p)
              (not (destin ?p ?f))))))
      (forall
        (?q - conflict-A)
        (and (or (destin ?q ?f)
                (not (boarded ?q)))
            (or (served ?q)
                (not (origin ?q ?f))))))
      (imply
        (exists
          (?p - never-alone)
          (or (and (origin ?p ?f)
```

Fahiem Bacchus selected a subset of the ADL subset of McDermott's PDDL for the 2000 competition.

(Actually, he first designed a whole new language all of his own, but the IPC-2000 organizing committee didn't like it.)

Maria Fox and Derek Long extended Bacchus's PDDL with various stuff.

- ▶ PDDL2.1 level 1: Bacchus's PDDL
- ▶ PDDL2.1 level 2: level 1 extended with numeric state variables. Comparisons between numeric expressions are allowed as logical atoms, effects can assign the value of an expression to a numeric variable
- ▶ PDDL2.1 level 3: level 2 extended with action durations. Actions take an amount of time given by the value of a numeric expression. Conditions and effects are evaluated at either the start or the end of the action, and several actions can be executed in parallel
- ▶ Here, we will sometimes consider numeric planning; we won't consider durational planning

PDDL2.1 was (and is) still considered a challenge ...

... so Stefan Edelkamp and I made only two relatively minor language extensions for PDDL2.2.

- ▶ Derived predicates: predicates that are not affected by the actions, and whose value is instead derived via a set of derivation rules of the form **if** $\phi(\bar{x})$ **then** $P(\bar{x})$. E.g. the flow of current in an electricity network
- ▶ Timed initial literals: literals that will become true, independently of the actions taken, at some pre-specified point in time (relative to the initial state). E.g. sunrise and sunset

... we don't do this here

Actually, Gerevini&Long thought that PDDL2.2 is still not enough ...

... and extended it with various complex notions of soft goals and preferences to obtain PDDL3

... and this is where I stopped following PDDL in detail

... in particular, we don't do this here

Automatic Planning

2. Planning Formalisms

- ▶ Why STRIPS Planning?
- ▶ Preface: Propositional Formulas
- ▶ STRIPS Planning
- ▶ The Complexity of STRIPS Planning
- ▶ PDDL
- ▶ **Numeric STRIPS**

Numeric STRIPS, Syntax

- ▶ STRIPS extended with numeric (real-valued) state variables
- ▶ In addition to the facts P we consider a (finite) set V of numeric variables
- ▶ An expression is formed by numeric variables v , constants c , and the $+$, $-$, $*$, $/$ connectives
- ▶ Expressions can be compared with $<$, \leq , $=$, \geq , $>$
- ▶ Comparisons of expressions are allowed in action preconditions and the goal
- ▶ Action effects of the form $v := \text{exp}(\bar{v})$ are allowed
- ▶ E.g. $\text{drive}(IBK, M) : \text{at}(IBK) \wedge \text{fuel} \geq 10 \longrightarrow \text{at}(M) := 1, \text{at}(IBK) := 0, \text{fuel} := \text{fuel} - 10$

Numeric STRIPS, PDDL Example

```
(:action build-house
:parameters (?p - place)
:precondition (and (>= (available wood ?p) 1) (>= (available stone ?p) 1))
:effect (and (increase (housing ?p) 1)
             (decrease (available wood ?p) 1)
             (decrease (available stone ?p) 1))))
```

```
(:action build-coal-stack
:parameters (?p - place)
:precondition (>= (available timber ?p) 1)
:effect (and (increase (labour) 2)
             (decrease (available timber ?p) 1)
             (has-coal-stack ?p)))
```

```
(:goal (and (>= (housing location0) 2) (has-coal-stack location0)))
```

- ▶ To evaluate an expression in a world/system state, we insert the values assigned by the state
- ▶ When applying a numeric effect $v := \text{exp}(\bar{v})$ to a state s , the value of v in the result state is the value of exp in s
- ▶ The definitions of plans, solvability, and of minimal/optimal plans don't change

Numeric STRIPS, Complexity

- ▶ PLANSAT is undecidable for this extended formalism
- ▶ E.g., one can simulate Abacus programs, that only increment or decrement the variables, and see if they are greater than 0
- ▶ Abacus program termination is undecidable, therefore numeric STRIPS is undecidable even if expression comparisons are restricted to $v > 0$ and effects are restricted to $v := v + 1$ and $v := v - 1$
- ▶ Details: [Helmert,AIPS-2002]
- ▶ We can still hope to **find a plan** in reasonable time ...

- ▶ S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, section on Planning
- ▶ T. Bylander, *The Computational Complexity of Propositional STRIPS Planning*, Artificial Intelligence Journal, 1994
- ▶ C. Gazen and C. Knoblock, *Combining the Expressivity of UCPOP with the Efficiency of Graphplan*, Proceedings of the 4th European Conference on Planning (ECP-1997)
- ▶ M. Helmert, *Decidability and Undecidability Results for Planning with Numerical State Variables*, Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2002)