Automated Planning

PLG Group

Universidad Carlos III de Madrid

AI. 2008-09

Indice

Indice

Indice

Indice

Control Knowledge

- Planning is PSPACE-complete [Bäckström, 1998]: it needs heuristics
- Types:
  - Domain-independent heuristics: HSP, FF, LPG ...
  - Domain-dependent heuristics
- There are many ways for knowledge representation:
  - control rules or rewriting rules
  - hierarchical task network
  - operator preconditions (functions, new predicates, ...)
  - macro-operators, cases, policies, ...

Control Rules

- Declarative if-then statements
- IF Conditions THEN Decisions
- Conditions: on the search meta-state
- Decisions: on the search alternatives
- Decision types: select, reject, prefer
- PRODIGY [Veloso *et al.*, 1995] was the first architecture defining a control-rule language

Ejemplo de regla de control

```
(control-rule selecciona-unload-airplane
   (if (and (current-goal (en <objeto> <sitio1>))
            (true-in-state (en <objeto> <sitio2>))
            (true-in-state (localizada-en <sitio1> <ciudad1>))
            (true-in-state (localizada-en <sitio2> <ciudad2>))
            (different-vars-p)
            (type-of-object <objeto> OBJECT)
            (type-of-object <sitio1> AEROPUERTO))
   (then select operator unload-airplane)))
```

Example of control rule

```
(control-rule select-operators-unload-airplane
    (if (current-goal (at <object> <location1>))
        (true-in-state (at <object> <location2>))
        (true-in-state (loc-at <location1> <city1>))
        (true-in-state (loc-at <location2> <city2>))
        (different-vars-p)
        (type-of-object <object> object)
        (type-of-object <location1 > airport))
    (then select operator unload-airplane))
```

Conditions (meta-predicates)

- **some literal / is true in the current state**
  `(true-in-state (at <object> <location2>))`
- **some literal / is the current goal**
  `(current-goal(at <object> <location1>))`
- **some literal / is a pending goal**
  `(some-candidate-goals /)`
- **some instance is of a given type**
  `(type-of-object <object> object)`

Decisions (decision points)

- state-space planning: goal, operator, binding, apply/subgoal
- plan-space planning: operator, threat, threat resolution
- hierarchical planning (HTN): method
- heuristic search-based planning: instantiated operator (forward) or goal (backward)
- graph-based planning: instantiated operator, 1-phase termination

Types of rules in PRODIGY

Decisions

- Goal: select an unachieved goal (*select/reject/prefer goal*)
- Operator: select an operator to achieve some goal (*select/reject/prefer operators*)
- Binding: select bindings for an operator when trying to achieve a goal (*select/reject/prefer bindings*)
- Decide to apply an operator for achieving a goal or subgoal on an unachieved goal (*sub-goal*)

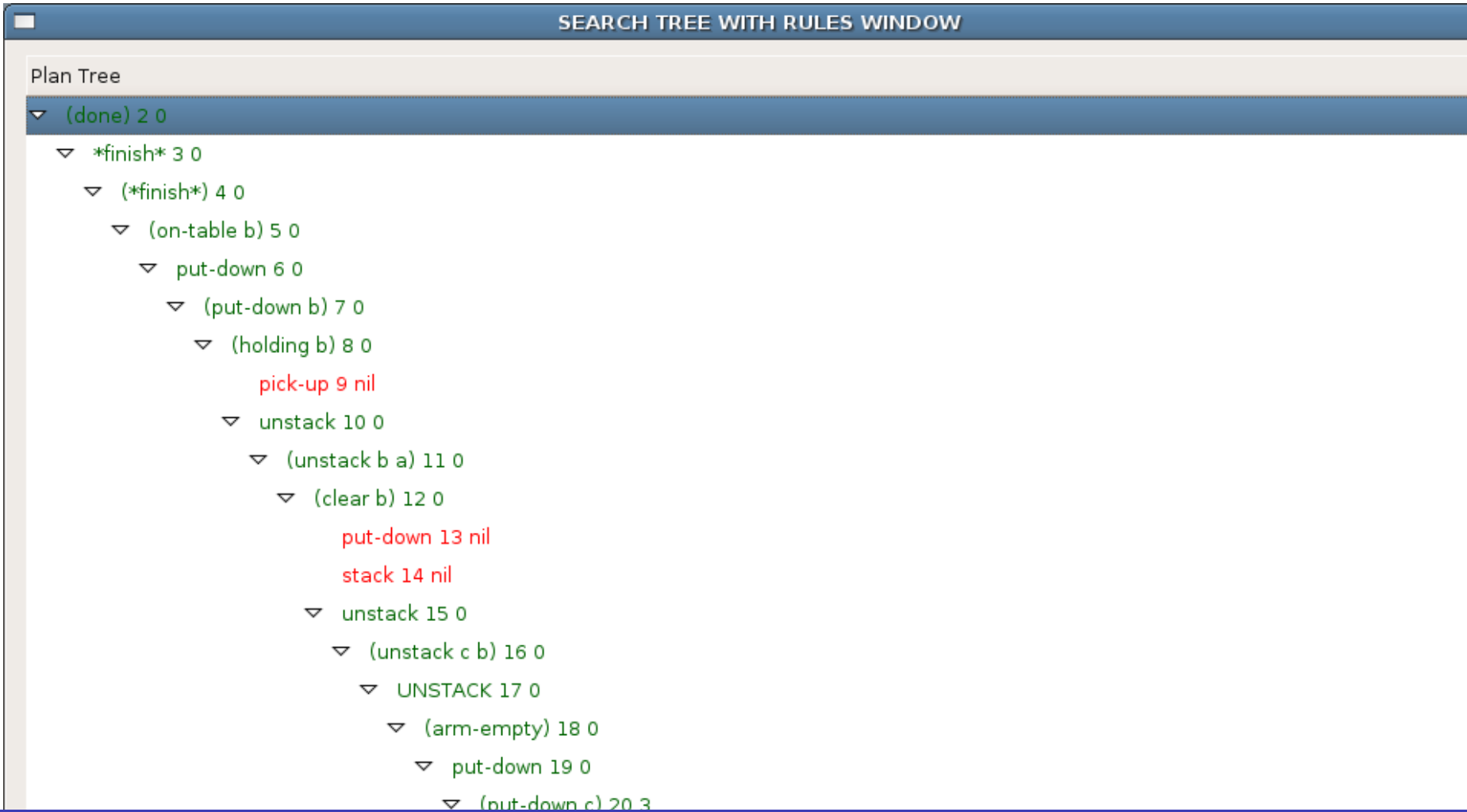
- Decision alternatives: select, reject or prefers

Examples of Control Rules

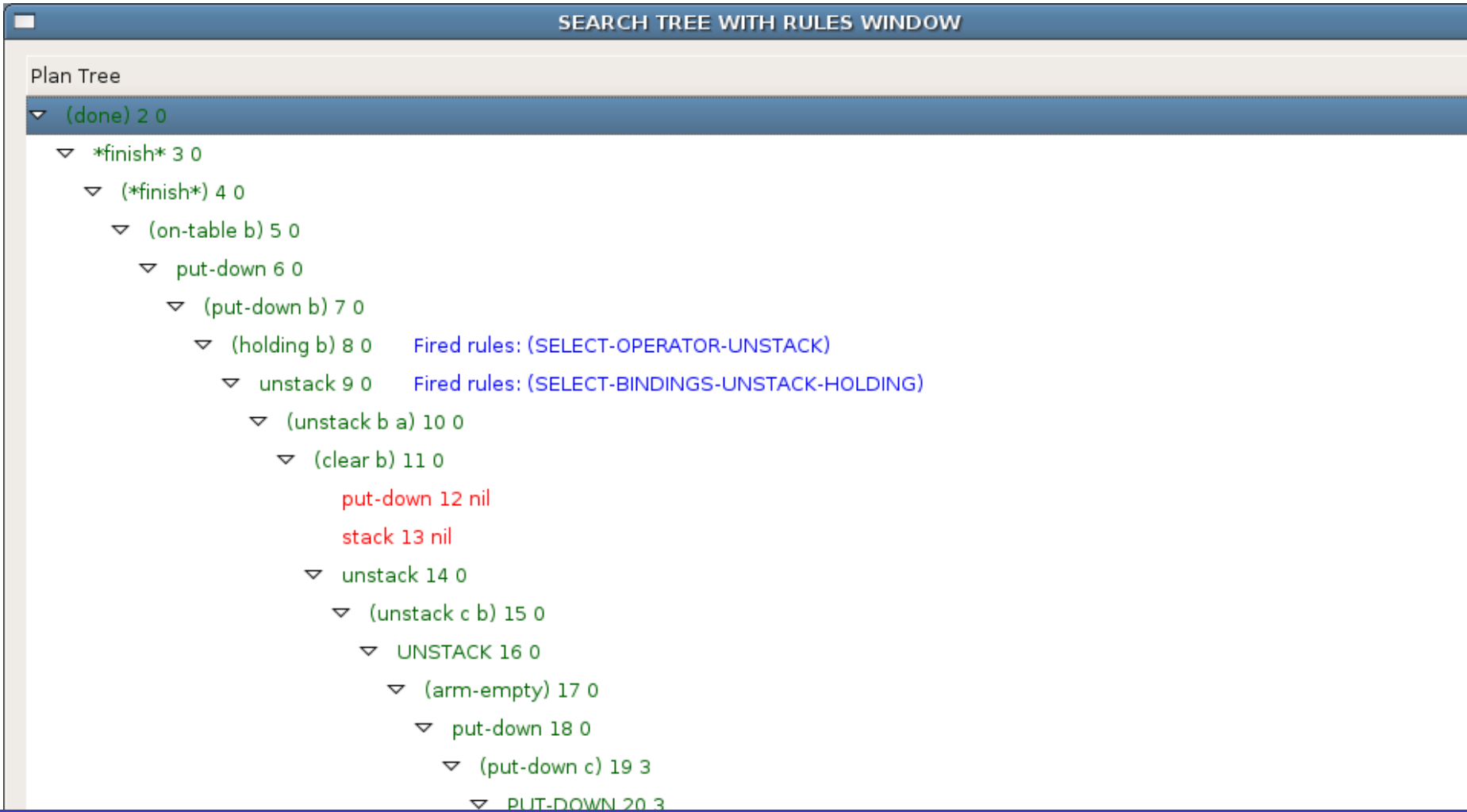
```
(control-rule select-op-unstack-for-holding
    (if (and (current-goal (holding <x>))
             (true-in-state (on <x> <y>))
    (then select operator unstack))
```

```
(control-rule select-bindings-unstack-holding
    (if (and (current-goal (holding <x>))
             (current-operator unstack)
             (true-in-state (on <x> <y>))))
    (then select bindings ((<ob> . <x>) (<underob> . <y>))))
```

Example of tree without rules



Example of tree with rules



Simple Temporal Logics

- Extend first-order logic with modal operators:
  - $\Phi_1 \cup \Phi_2$: Φ_1 has to hold until Φ_2 holds
  - $\Box\Phi$: Φ has to hold on the entire subsequent path
  - $\Diamond\Phi$: Φ eventually has to hold (somewhere on the subsequent path)
  - $\circ\Phi$: Φ has to hold at the next state
- Then, they utilize domain specific search control information to control simple forward chaining search
- Examples: TLPLAN [Bacchus and Kabanza, 2000] and TALPLANNER [Kvarnström, 2002]

Temporal formula

- Natural language:"it should always happen that if an object is inside an airplane in a given state and the airplane is not in the destination city of the object, then in the next state the object should stay in the airplane"
- Formally:
  $\Box\ (\forall\ [p : \text{airplane}(p)]\ \exists\ [l : \text{at}(p, l)]\ \forall\ [o : \text{in-wrong-city}(o, l)]\ \text{in}(o, p) \Rightarrow \circ\ \text{in}(o, p))$
- Program:

```
(always (forall (?p) (airplane ?p)
                     (exists (?l) (at ?p ?l)
                                  (forall (?o) (in-wrong-city ?o ?l)
                                               (implies (in ?o ?p)
                                                        (next (in ?o ?p)))))))
```

Example in blocksworld

```
(def-defined-predicate (above ?x ?y)
    (or (on ?x ?y)
        (exists (?z) (on ?z ?y) (above ?x ?z))))

(def-defined-predicate (in-final-position ?x)
    (or (and (on-table ?x)
             (not (exists (?y) (goal (on ?x ?y)))))
        (exists (?y) (on ?x ?y)
            (and (not (goal (on-table ?x)))
                 (forall (?z) (goal (on ?x ?z)) (= ?z ?y))
                 (forall (?z) (goal (on ?z ?y)) (= ?z ?x))
                 (in-final-position ?y)))))
```

Example in blocksworld

```
(always (forall (?x) (clear ?x)
            (and (implies
                     (in-final-position ?x)
                     (next (and (not (holding ?x))
                                (forall (?y) (on ?y ?x)
                                    (in-final-position ?y)))))
                 (implies
                     (bad-tower ?x)
                     (next (not (exists (?y) (on ?y ?x)))))
                 (implies
                     (and (on-table ?x)
                          (not (exists (?y) (goal (on ?x ?y))
                                       (good-tower ?y))))
                     (next (not (holding ?x)))))))
```

Adding preconditions to operators

```
(def-adl-operator (fly ?p ?from ?to)
    (pre (?p) (airplane ?p) (?from) (at ?p ?from) (?to) (airport ?to)
        (and (not (= ?from ?to))
            (not (exists (?obj) (at ?obj ?from)
                        (move-by-plane ?obj ?from)))
            (not (exists (?obj) (in ?obj ?p)
                        (unload-from-plane ?obj ?from)))
            (or (goal ?obj ?to)
                (exists (?obj) (at ?obj ?to)
                        (move-by-plane ?obj ?to))
                (exists (?obj) (in ?obj ?p)
                        (unload-from-plane ?obj ?to)))))
    (add (at ?p ?to))
    (del (at ?p ?from)))
```

Rewriting rules

```
(define-rule :name avoid-move-twice
     :if (:operators ((?n1 (unstack ?b1 ?b2))
                      (?n2 (stack ?b1 ?b3 Table)))
         :links (?n1 (on ?b1 Table) ?n2)
         :constrains ((possibly-adjacent ?n1 ?n2)
                      (:neq ?b2 ?b3)))
         :replace (:operators (?n1 ?n2))
         :with (:operators (?n3 (stack ?b1 ?b3 ?b2))))
(define-rule :name avoid-undo
         :if (:operators ((?n1 (unstack ?b1 ?b2))
                          (?n2 (stack ?b1 ?b2 Table)))
             :constrains ((possibly-adjacent ?n1 ?n2)))
         :replace (:operators (?n1 ?n2))
         :with nil)
```

Conocimiento de control en funciones

```
(OPERATOR LOAD-TRUCK
  (params <obj> <truck> <loc>)
  (preconds ((<obj> OBJECT)
             (<truck> TRUCK)
             (<loc> (and LOCATION
                         (in-truck-city-p  <truck> <loc>))))
    (and (at-obj <obj> <loc>)
         (at-truck <truck> <loc>)))
  (effects ()
    ((add (inside-truck <obj> <truck>))
     (del (at-obj <obj> <loc>)))))
```

Control knowledge in object definitions

```
(OPERATOR LOAD-TRUCK
    (params <obj> <truck> <loc>)
    (preconds ((<obj> OBJECT)
               (<truck> TRUCK)
               (<loc> (and LOCATION
                           (in-truck-city-p <truck> <loc>))))
       (and (at-obj <obj> <loc>)
            (at-truck <truck> <loc>)))
    (effects ()
       ((add (inside-truck <obj> <truck>))
       (del (at-obj <obj> <loc>)))))
```

(when PRODIGY loads a domain file, it also loads the file named functions.lisp in the same directory)

Conocimiento de control en precondiciones

```
(OPERATOR STACK
 (params <ob> <underob>)
 (preconds ((<ob> OBJECT)
            (<underob> (and OBJECT
                            (diff <ob> <underob>))))
   (and (good-tower <underob>)
        (clear <underob>)
        (holding <ob>)))
 (effects ()
   ((del (holding <ob>))
    (del (clear <underob>))
    (add (arm-empty))
    (add (clear <ob> ))
    (add (on <ob> <underob>)))))
```

Control knowledge in preconditions

```
(OPERATOR STACK
    (params <ob> <underob>)
    (preconds ((<ob> OBJECT)
               (<underob> (and OBJECT
                               (diff <ob> <underob>))))
        (and (good-tower <underob>)
             (clear <underob>)
             (holding <ob>)))
    (effects ()
        ((del (holding <ob>))
        (del (clear <underob>))
        (add (arm-empty))
        (add (clear <ob> ))
        (add (on <ob> <underob>)))))
```

Macro-operators

Sequences of operators

```
(:action load*fly-airplane*unload
        :parameters (?p - airplane ?s - airport
                     ?d - airport ?o - package)
        :precondition (and (at ?p ?s)
                           (at ?o ?s)
                           (not (= ?s ?d)))
        :effect (and (at ?p ?d)
                     (at ?o ?d)
                     (not (at ?o ?s))
                     (not (at ?p ?s))))
```

Policies (MDP)

Q-Table:

What cumulative expected delayed reward when applying action A at state S

	Actions		
States	A_1	...	A_n
S_1	0.7	...	0.2
...	...	...	...
S_m	0.5	...	0.3

Indice

References

Fahiem Bacchus and Froduald Kabanza.

Using temporal logics to express search control knowledge for planning.

Artificial Intelligence, 116:123–191, 2000.

C. Bäckström.

Computational Aspects of Reordering Plans.

In *Journal of Artificial Intelligence Research*, volume 9, pages 99–137. Morgan Kaufmann Pub. Inc., 1998.

Jonas Kvarnström.

Applying domain analysis techniques for domain-dependent control in TALplanner.

In *Proceedings of the Sixth International Conference on Automated Planning and Scheduling*, pages 101–110. AAAI, AAAI Press, 2002.

Manuela M. Veloso, Jaime Carbonell, M. Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe.

Integrating planning and learning: The PRODIGY architecture.

Journal of Experimental and Theoretical Artificial Intelligence, 7(1):81–120, 1995.