Planification hiérarchique

Langages, résolution et apprentissage

Humbert Fiorino
Humbert.Fiorino@imag.fr
http://marvin.imag.fr/doku.php?id=members:fiorino:fiorino

18 janvier 2024

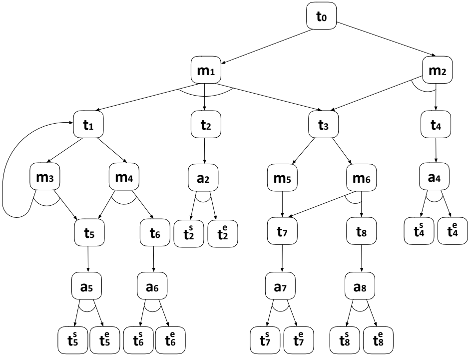
Laboratoire d'Informatique de Grenoble
Université Grenoble Alpes

Intérêts des langages hiérarchiques

- Expressivité :
  - PDDL : uniquement des actions –> grounding + méthodes heurisitques efficaces (Fast Forward etc.)
  - Hiérarchie : notions de tâches et de méthodes
  - Récursivité
- Avantages :
  - Répresentation de ”recettes”/procédures telles qu’établies par les experts du domaine
  - Explicabilité, gestion de niveaux d’abstraction, planification d’initiative mixte
  - Guidage de la recherche
- Inconvénients :
  - Quid du grounding et des heurisitiques ?

Grounding & Heuristics

- How it starts: a grounding procedure... propositional logic, optimizations, compact binary representation etc.
- Task Decomposition Graphs: AND/OR graph
- Heuristics example: mandatory tasks = tasks that will unquestionably be included in a partial plan when a given task is decomposed. For instance, $M(t_o) = \{t_3, t_7, t_7^s, t_7^e\}$, $M(t_1) = \{t_5, t_5^s, t^e5\}$, $M(t_3) = \{t_7, t_7^s, t_7^e\}$, $M(t_4) = \{t_4^s, t_4^e\}$, $M(t_5) = \{t_5^s, t_5^e\}$, $M(t_6) = \{t_6^s, t_6^e\}$, $M(t_7) = \{t_7^s, t_7^e\}$ and $M(t_8) = \{t_8^s, t_8^e\}$.



HDDL

- Need of language standardization
- Extension of PDDL vs. chronicle approaches (e.g. ANML etc.)

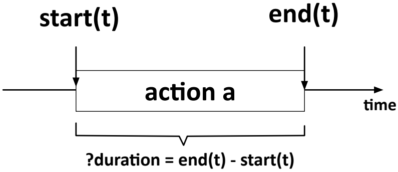
```
(:action pick—up
:parameters (?x — block)
:precondition (and (clear ?x) (ontable ?x) (handempty))
:effect (and (not (ontable ?x)) (not (clear ?x)) (not (handempty))
             (holding ?x)))

(:method do—move
 :parameters (?x — block ?y — block)
 :task (do—move ?x ?y)
 :precondition (and (clear ?x) (clear ?y)
                    (handempty) (ontable ?x))
 :ordered—subtasks(and (t1 (pick—up ?x)) (t2 (stack ?x ?y))) )
```

Durative Actions in HDDL

- Durative actions in HDDL
  1. same formalism as in PDDL2.1
  2. can be represented with two instantaneous events **start** and **end**
  3. have durations
- Example of a simple durative action

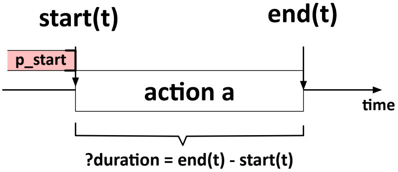
```
(:durative-action action
  :parameters (?x1 - t1 ?x2 - t2)
  :duration (= ?duration 1.00)
  :condition (and
    (at start (p_start ?x1 ?x2))
    (at end (p_end ?x1 ?x2))
    (over all (p_inv ?x1 ?x2)))
  :effect (and
    (at start (e_start ?x1 ?x2))
    (at end (e_end ?x1 ?x2))) )
```



Durative Actions in HDDL

- Durative actions in HDDL
  1. same formalism as in PDDL2.1
  2. can be represented with two instantaneous events **start** and **end**
  3. have durations
- Example of a simple durative action

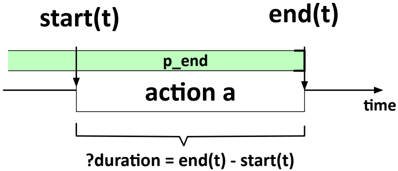
```
(:durative-action action
  :parameters (?x1 - t1 ?x2 - t2)
  :duration (= ?duration 1.00)
  :condition (and
    (at start (p_start ?x1 ?x2))
    (at end (p_end ?x1 ?x2))
    (over all (p_inv ?x1 ?x2)))
  :effect (and
    (at start (e_start ?x1 ?x2))
    (at end (e_end ?x1 ?x2))) )
```



Durative Actions in HDDL

- Durative actions in HDDL
  1. same formalism as in PDDL2.1
  2. can be represented with two instantaneous events **start** and **end**
  3. have durations
- Example of a simple durative action

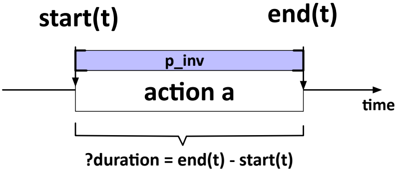
```
(:durative-action action
  :parameters (?x1 - t1 ?x2 - t2)
  :duration (= ?duration 1.00)
  :condition (and
    (at start (p_start ?x1 ?x2))
    (at end (p_end ?x1 ?x2))
    (over all (p_inv ?x1 ?x2)))
  :effect (and
    (at start (e_start ?x1 ?x2))
    (at end (e_end ?x1 ?x2))) )
```



Durative Actions in HDDL

- Durative actions in HDDL
  1. same formalism as in PDDL2.1
  2. can be represented with two instantaneous events **start** and **end**
  3. have durations
- Example of a simple durative action

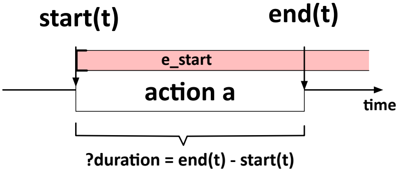
```
(:durative-action action
  :parameters (?x1 - t1 ?x2 - t2)
  :duration (= ?duration 1.00)
  :condition (and
    (at start (p_start ?x1 ?x2))
    (at end (p_end ?x1 ?x2))
    (over all (p_inv ?x1 ?x2)))
  :effect (and
    (at start (e_start ?x1 ?x2))
    (at end (e_end ?x1 ?x2))) )
```



Durative Actions in HDDL

- Durative actions in HDDL
  1. same formalism as in PDDL2.1
  2. can be represented with two instantaneous events **start** and **end**
  3. have durations
- Example of a simple durative action

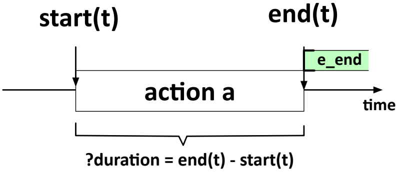
```
(:durative-action action
  :parameters (?x1 - t1 ?x2 - t2)
  :duration (= ?duration 1.00)
  :condition (and
    (at start (p_start ?x1 ?x2))
    (at end (p_end ?x1 ?x2))
    (over all (p_inv ?x1 ?x2)))
  :effect (and
    (at start (e_start ?x1 ?x2))
    (at end (e_end ?x1 ?x2))) )
```



Durative Actions in HDDL

- Durative actions in HDDL
  1. same formalism as in PDDL2.1
  2. can be represented with two instantaneous events **start** and **end**
  3. have durations
- Example of a simple durative action

```
(:durative-action action
  :parameters (?x1 - t1 ?x2 - t2)
  :duration (= ?duration 1.00)
  :condition (and
    (at start (p_start ?x1 ?x2))
    (at end (p_end ?x1 ?x2))
    (over all (p_inv ?x1 ?x2)))
  :effect (and
    (at start (e_start ?x1 ?x2))
    (at end (e_end ?x1 ?x2))) )
```



Toward Durative Methods

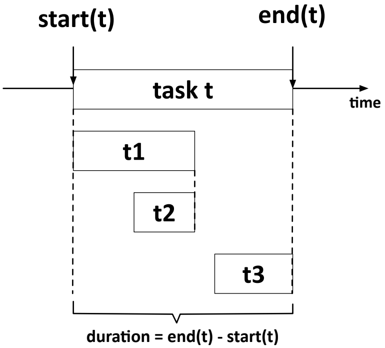
- Guiding idea: keep it as close as possible to PDDL syntax and semantics
  - A durative method has two dummy non durative actions that represent the start and the end of the task achieved by the method
- To cope with time, we propose to extend method definition with:
  1. precondition tagged by time specifier
  2. extending the ordering constraints
  3. duration constraints on method decomposition
  4. constraints on method decomposition from PDDL 3.0

Toward Durative Methods

An abstract simple example

- Durative method preconditions –> same semantics as in durative actions
- Ordering constraints are extended to deal with $<, >, \leq, \geq$ and $=$

```
(:durative-method m
  :parameters (?x1 ?x2 - type)
  :task (t ?x1 ?x2 ?x3)
  :condition (and
    (at start (p_start ?x1 ?x2))
    (at end (p_end ?x1 ?x2))
    (over all (p_inv ?x1 ?x2)))
  :subtasks (and
    (t1 (t1 ?x1 ?x2))
    (t2 (t2 ?x1 ?x2))
    (t3 (t3 ?x1 ?x2))))
  :ordering (and
    (< end(t1) start(t3)
    (< end(t2) start(t3))
    (= end(t1) end(t2)))))
```

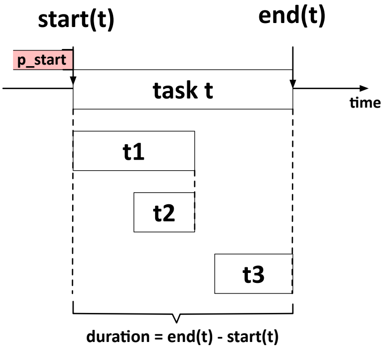


Toward Durative Methods

An abstract simple example

- Durative method preconditions –> same semantics as in durative actions
- Ordering constraints are extended to deal with $<, >, \leq, \geq$ and $=$

```
(:durative-method m
  :parameters (?x1 ?x2 - type)
  :task (t ?x1 ?x2 ?x3)
  :condition (and
    (at start (p_start ?x1 ?x2))
    (at end (p_end ?x1 ?x2))
    (over all (p_inv ?x1 ?x2)))
  :subtasks (and
    (t1 (t1 ?x1 ?x2))
    (t2 (t2 ?x1 ?x2))
    (t3 (t3 ?x1 ?x2))))
  :ordering (and
    (< end(t1) start(t3)
    (< end(t2) start(t3))
    (= end(t1) end(t2)))))
```

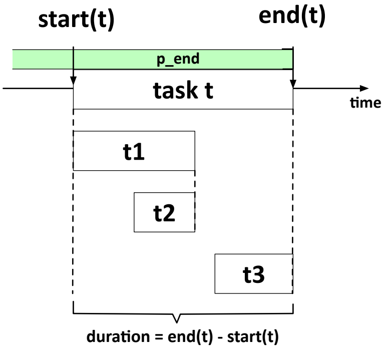


Toward Durative Methods

An abstract simple example

- Durative method preconditions –> same semantics as in durative actions
- Ordering constraints are extended to deal with $<, >, \leq, \geq$ and $=$

```
(:durative-method m
  :parameters (?x1 ?x2 - type)
  :task (t ?x1 ?x2 ?x3)
  :condition (and
    (at start (p_start ?x1 ?x2))
    (at end (p_end ?x1 ?x2))
    (over all (p_inv ?x1 ?x2)))
  :subtasks (and
    (t1 (t1 ?x1 ?x2))
    (t2 (t2 ?x1 ?x2))
    (t3 (t3 ?x1 ?x2))))
  :ordering (and
    (< end(t1) start(t3)
    (< end(t2) start(t3))
    (= end(t1) end(t2)))))
```

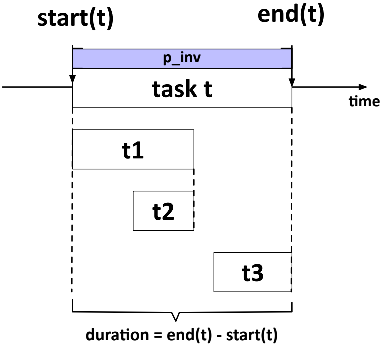


Toward Durative Methods

An abstract simple example

- Durative method preconditions –> same semantics as in durative actions
- Ordering constraints are extended to deal with $<, >, \leq, \geq$ and $=$

```
(:durative-method m
  :parameters (?x1 ?x2 - type)
  :task (t ?x1 ?x2 ?x3)
  :condition (and
    (at start (p_start ?x1 ?x2))
    (at end (p_end ?x1 ?x2))
    (over all (p_inv ?x1 ?x2)))
  :subtasks (and
    (t1 (t1 ?x1 ?x2))
    (t2 (t2 ?x1 ?x2))
    (t3 (t3 ?x1 ?x2))))
  :ordering (and
    (< end(t1) start(t3)
    (< end(t2) start(t3))
    (= end(t1) end(t2)))))
```



Toward Durative Methods

Adding durative constraints to decompositions

- Durative constraints may relate to the duration of the task or to a particular subtask
- Durative constraints on particular subtasks can be rewritten in terms of ordering constraints
- Example:

```
(:durative-method grab_image
  :parameters (?s - satellite ?d1 ?d2 - image_direction)
  :task (grab_image ?s ?d1 ?d2 ?i ?m)
  :duration (and (<= ?duration (* (turn-time (?d1 ?d2)) 2))
    (<= duration(t2) duration(t3)))
  :subtasks (and (t1 (turn_to ?s ?d1 ?d2))
    (t2 (calibrate ?s ?i ?m))
    (t3 (take_image ?s ?d2 ?i ?m)))
  :ordering (< end(t1) start(t3))
  :constraints (and (not (= ?d1 ?d2))) )
```

Toward Durative Methods

Generalizing PDDL 3.0 constraints to method decompositions

- Why not using PDDL 3.0 trajectory constraints to define constraints on method decompositions?
- Constraints on method decompositions are limited to the method scope
- Example:

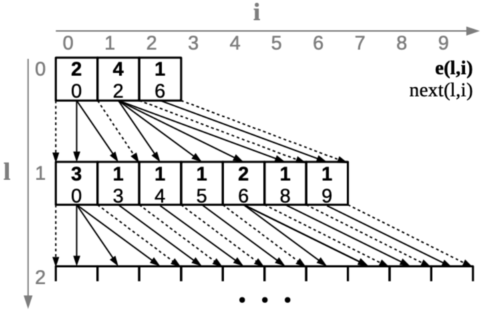
```
(:durative-method method_observe
  :parameters (?d1 ?d2 - image_direction
    ?s - satellite ?i - instrument   ?m - mode)
  :task (do_observation ?d2 ?m)
  :duration (< (duration t1) (calib-time ?i))
  :subtasks (and (t0 (activate_instrument ?s ?i))
    (t1 (turn_to ?s ?d1 ?2))
    (t2 (take_image ?s ?d ?i ?m)))
  :ordering (and (< t0 t2) (< t1 t2))
  :constraints (and (not (= ?d1 ?d2))
    (at-most-once (pointing ?s ?d2))) )
```

Open issues

1. What is the semantics of an empty durative method?
2. Is it interesting to enrich the ordering constraints to express deadlines for the start and end of tasks?
3. Any other points?

Résolution par méthodes d'encodage

- Tree-REX: non-temporal totally-ordered HDDL with an incremental SAT solver
- We have been investigating different approaches: STRIPS, SAT, CSP, SMT, deordering etc. for temporal HDDL
- Example: thanks to HDDL grounding...



Rules of Encoding

The initial state holds at the initial layer 0 at position 0:

$$\bigwedge_{p \in s_0} holds(p, 0, 0) \wedge \bigwedge_{p \notin s_0} \neg holds(p, 0, 0) \tag{1}$$

At each position j of the initial layer, the respective initial task reductions are possible. Let $T = \langle t_0, \ldots, t_j, \ldots, t_{k-1} \rangle$:

$$\bigwedge_{j=0}^{k-1} \bigvee_{r \in R(t_j)} element(r, 0, j) \tag{2}$$

The last position of the initial layer contains a *blank* element:

$$element(blank, 0, k) \tag{3}$$

At the last position of the initial layer, all goal facts hold:

$$\bigwedge_{p \in g} holds(p, 0, k) \tag{4}$$

The presence of an action at some position i implies its preconditions at position i and its effects at position $i+1$:

$$element(a, l, i) \Rightarrow \bigwedge_{p \in pre(a)} holds(p, l, i) \tag{5}$$

$$element(a, l, i) \Rightarrow \bigwedge_{p \in eff^{+}(a)} holds(p, l, i+1)$$

$$element(a, l, i) \Rightarrow \bigwedge_{p \in eff^{-}(a)} \neg holds(p, l, i+1)$$

A reduction at some position i implies its preconditions at that position:

$$element(r, l, i) \Rightarrow \bigwedge_{p \in pre(r)} holds(p, l, i) \tag{6}$$

Each action is primitive, and each reduction is non-primitive. The following rules eliminate the possibility of an action and a reduction to co-occur:

$$\begin{aligned} element(a, l, i) &\Rightarrow primitive(l, i) \\ element(r, l, i) &\Rightarrow \neg primitive(l, i) \end{aligned} \tag{7}$$

If a fact changes, then either this position does not contain an action yet or it contains an action which supports this fact change. Such constraints are also called "frame axioms".

$$\begin{aligned} &holds(p, l, i) \ \wedge \ \neg holds(p, l, i+1) \Rightarrow \\ &\Rightarrow \neg primitive(l, i) \vee \bigvee_{p \in eff^-(a)} element(a, l, i) \\ &\neg holds(p, l, i) \ \wedge \ holds(p, l, i+1) \Rightarrow \\ &\Rightarrow \neg primitive(l, i) \vee \bigvee_{p \in eff^+(a)} element(a, l, i) \end{aligned} \tag{8}$$

At each position, all possibly occurring actions are mutually exclusive. (Note that this also includes the *blank* action variable.) For each pair of actions a_1, a_2, we have:

$$\neg element(a_1, l, i) \vee \neg element(a_2, l, i) \tag{9}$$

A fact p holds at some position i if and only if it also holds at its first child position at the next hierarchical layer.

$$holds(p, l, i) \Leftrightarrow holds(p, l+1, next(l, i)) \tag{10}$$

If an action occurs at some position i, then it will also occur at its first child position at the next hierarchical layer.

$$element(a, l, i) \Rightarrow element(a, l+1, next(l, i)) \tag{11}$$

If an action occurs at some position i, then all further child positions at the next layer will contain a *blank* element.

$$\bigwedge_{0<j<e(l,i)} element(a, l, i) \Rightarrow \\ \Rightarrow element(blank, l+1, next(l, i) + j) \tag{12}$$

If a reduction occurs at some position i, then it implies some valid combination of its subtasks at the next layer. Let $\textit{subtasks}(r) = \langle t_0, \ldots, t_{k-1} \rangle$ and $0 \leq j < k$. If t_j is primitive and accomplished by an action a:

$$\textit{element}(r, l, i) \Rightarrow \textit{element}(a, l + 1, \textit{next}(l, i) + j) \quad (13)$$

If t_j is non-primitive and $R(t_j)$ are its possible reductions:

$$\textit{element}(r, l, i) \Rightarrow \bigvee_{r' \in R(t_j)} \textit{element}(r', l + 1, \textit{next}(l, i) + j) \quad (14)$$

Any positions j at the next layer which remain undefined by an occurring reduction are filled with *blank* symbols.

$$\bigwedge_{k \leq j < e(l,i)} \textit{element}(r, l, i) \Rightarrow \textit{element}(\textit{blank}, l + 1, i + j) \quad (15)$$

To find a plan after n layers, we must ensure that all the positions of the last (i.e. the current) hierarchical layer n must be primitive. Let s_n be the size of the array at layer n:

$$\bigwedge_{0 \leq i < s_n} \textit{primitive}(n, i) \quad (16)$$

- TEP: hybrid planning –> temporal HDDL as input –> partially-ordered plan with timestamps as output
- Solve all Cushing's categories
- First temporal + partially-ordered HDDL –> Non-temporal partially-ordered HDDL solved by a planner with heuristics
- Then timestamps are computed with a CSP solver

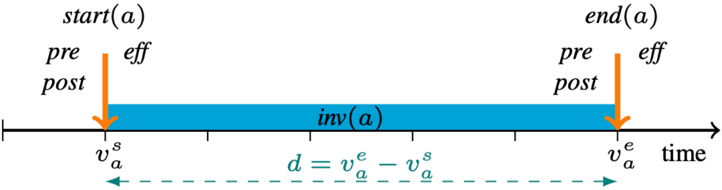
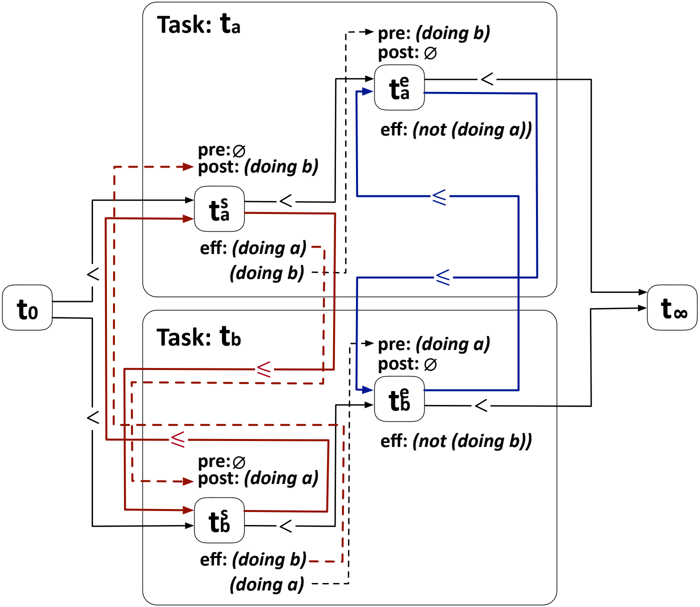


Figure 1: Timeline of a durative action *a*.


Task: ta
pre: (doing b)
post: ∅
eff: (not (doing a))
pre:∅
post: (doing b)
eff: (doing a)
(doing b)
t0
t∞
Task: tb
pre: (doing a)
post: ∅
eff: (not (doing b))
pre:∅
post: (doing a)
eff: (doing b)
(doing a)

Méthodes d'apprentissage automatique

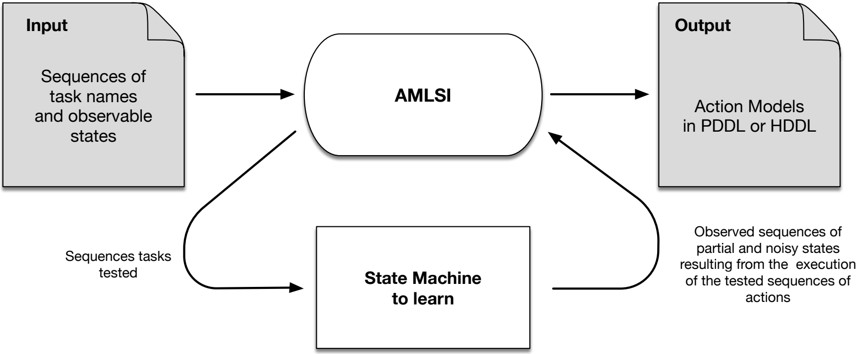
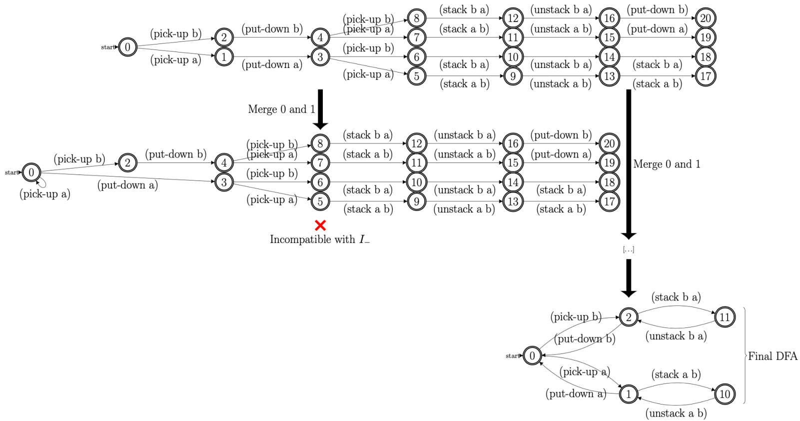


Figure 2: AMLSI: Action Model Learning with State machine Interaction.

Méthodes d’apprentissage automatique

- AMLSI is based on grammar induction techniques (RPNI) + lifting
- Deal with noisy and partial observations (tabu search)
- Accuracy: ability to solve new problems



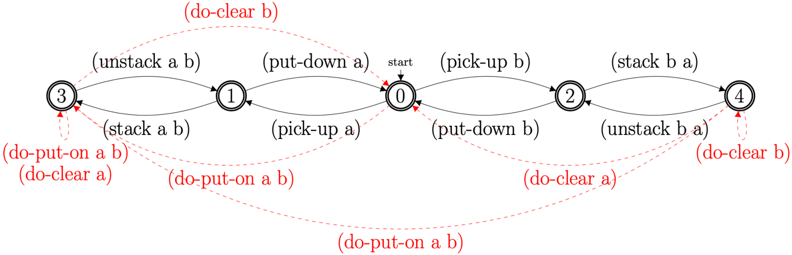


Figure 4: Apprentissage d'automates finis avec méthodes.

Perspectives

- Encodages pour HDDL temporel
- Validation/certification de domaines HDDL

Bibliographie

- Nicolas Cavrel, Damien Pellier, Humbert Fiorino. Efficient HTN to STRIPS Encodings for Concurrent Planning. ICTAI 2023, p. 962-969.
- Maxence Grand, Damien Pellier, Humbert Fiorino. An Accurate PDDL Domain Learning Algorithm from Partial and Noisy Observations. ICTAI 2022, p. 734-738.
- D. Höller, G. Behnke, P. Bercher, S. Biundo, H. Fiorino, D. Pellier, R. Alford. HDDL - A Language to Describe Hierarchical Planning Problems. In the proceedings of the Conference on Artificial Intelligence (AAAI), 2020.
- D. Schreiber, D. Pellier, H. Fiorino, T. Balyo. Tree-REX: SAT-Based Tree Exploration for Efficient and High-Quality HTN Planning. In the proceedings of ICAPS 2019, p. 382-390.
- D. Pellier, H. Fiorino. PDDL4J: A Planning Domain Description Library for Java. Journal of Experimental & Theoretical Artificial Intelligence, pages 143-176, volume 30(1), 2018.
- D. Ramoul, D. Pellier, H. Fiorino and S. Pesty. HTN Planning Approach Using Fully Instantiated Problems. In the proceedings of the International Conference on Tools in Artificial Intelligence (ICTAI), 2016. (Best Student Paper award)