ANML Overview

Action Notation Modeling Language

M. Maratea

Automated Planning
Master in Computer Science and Artificial Intelligence
DeMaCS, Università della Calabria

June 24, 2025

Contents

PROFMAT

What is ANML?

- ANML (Action Notation Modeling Language) is a planning domain modeling language developed by NASA Ames Research Center.
- Designed to describe complex planning problems with durative actions, temporal constraints, and object-oriented structures.
- Offers more expressiveness and readability than PDDL, especially for temporal planning and action intervals.

PROFMAT

Advantages over PDDL 2.1

- Durative conditions on custom intervals (e.g., $[start + x, end - y]$).
- Intermediate effects at specified time points during an action (e.g., $[start + x]$).
- Object-oriented constructs: classes, objects, functions, and actions.
- Intuitive syntax closer to standard programming languages.

PROFMAT

ANML File Structure

- PDDL: separate domain and problem files.
- ANML: single .anml file contains domain, initial state, and goals.

PROFMAT

From PDDL to ANML – Predicates (1/2)

For declaring a predicate in ANML, we use the keyword *fluent* along with the *boolean* built-in datatype like this:

```
// predicate with parameters
fluent boolean predicateName(TYPE param1, TYPE param2, ...);

// basic predicate without parameters
fluent boolean predicateName;
```

PROFMAT

From PDDL to ANML – Predicates (2/2)

PDDL – Domain File (Predicates)

```
(:predicates
 (at-robot ?r - robot ?l - room)
 (at-obj ?b - obj ?r - room)
 (free ?r - robot)
 (carry ?o - obj ?r - robot)
 (allowed ?r - robot ?l - room))
```

ANML – Domain File (Predicates)

```
fluent boolean atRobot(robot r, room l);
fluent boolean atObj(obj b, room r);
fluent boolean free(robot r);
fluent boolean carry(obj o, robot r);
fluent boolean allowed(robot r, room l);
```

PROFMAT

From PDDL to ANML – Functions (1/2)

- to declare *:functions* predicates it's the same thing, just changing the datatype to *integer*.
- Note that, if you want to instantiate a constant predicate or function, you can use the keyword *constant* instead of fluent (this is valid for both predicate and functions, and for all datatypes).

PROFMAT

From PDDL to ANML – Functions (2/2)

PDDL

(:functions(move-time ?r - robot)(battery ?r - robot))

- We assume that the moving time for a robot is the same every time. So we can declare it as a constant in ANML
- Since the battery can either be charged or loose energy from the movements. So we can declare it as a fluent in ANML

ANML

constant integer moveTime(robot r);
fluent integer battery(robot r);

PROFMAT

From PDDL to ANML – Action General Structure

```
action action_name(TYPE parameters) {
  duration := DURATION;
  [start]   start_condition;
  [start + x] intermediate_effect;
  [end - x] intermediate_effect;
  [start + x, end - y] durative_condition;
  [end] end_effect;
 };
```

PROFMAT

Move Action: PDDL 2.1

```
(:durative-action move
 :parameters (?r - robot ?a - room ?b - room)
 :duration (= ?duration (move-time ?r))
 :condition (and (over all (allowed ?r ?b))
                 (at start (at-robot ?r ?a))
                 (at start (> (battery ?r) 20)))
 :effect (and (at start (not (at-robot ?r ?a)))
              (at end   (at-robot ?r ?b))
              (at end   (decrease (battery ?r) 20))))
```

PROFMAT

Move Action: ANML

```
action move(robot r, room a, room b) {
  duration := moveTime(r);

  [all]   allowed(r, b) == true;
  [start] atRobot(r, a) == true;
  [start] battery(r) > 20;

  [start] atRobot(r, a) := false;
  [end]   atRobot(r, b) := true;
  [end]   battery(r) := battery(r) - 20;
};
```

PROFMAT

Move Action Explanation

- *duration* is trivial, we just assign the moving time of the robot. Be careful that the duration must be constant (e.g. you could also use a number directly).

- The qualifiers are used to specify the temporal moment or interval where a condition must hold or an effect should be applied:
  1. `[start]`: at the beginning of the action
  2. `[end]`: at the end of the action
  3. `[all]`: the condition must hold for the entire duration of the action

- conditions are expressed as logical formulas, very similar to our modern programming languages. Note that you can concatenate more conditions with `and`, `or`, and other boolean operators.

PROFMAT

Intermediate Condition and Effects (ICE)

- Another thing you can do, which is not showed in our example, is that you can make a durative condition by just specifying the interval in which such condition should be valid:

Durative Condition

```
[start + k, end - t] somePred(x, y, z) == true;
```

- For effect is the same reasoning, you can assign values to your fluents, and you can also specify the exact time during the action where such fluent should change value:

Intermediate Effects

```
[start + k] somePred(x, y, z) := true;
[end - k] somePred(x, y, z) := true;
```

PROFMAT

Frame Title

- Now you have all you need to convert the other actions
- If your actions are istanteaneous, like in our case, you can omit the duration and all the qualifiers inside the action.

PROFMAT

Problem Definition: Initial State and Goal (1/3)

- Now we start defining the initial state of the problem. Keep in mind that also here you have to use the qualifiers before assigning values, but just if we're speaking about fluents

- To initialize predicates in the initial state we just use `[start]` , for defining the goal predicates we use `[end]`

- If a certain predicate is set to `true` for a specific instance $x = a$, then it must be explicitly set to `false` for all other possible instances $x \neq a$, as the solver does not enforce this behavior by default

PROFMAT

Problem Definition: Initial State and Goal (2/3)

```
instance Room roomA, roomB, roomC;
instance Obj ball1, ball2;
instance Robot eve, wally;

[start] at_robot(wally, roomA) := true;
[start] at_robot(eve, roomB)   := true;
[start] at_robot(wally, roomB) := false;
[start] at_robot(eve, roomA)   := false;
[start] at_robot(wally, roomC) := false;
[start] at_robot(eve, roomC)   := false;
[start] free(wally)            := true;
[start] free(eve)              := true;
[start] at_obj(ball1, roomA)   := true;
[start] at_obj(ball2, roomA)   := true;
[start] at_obj(ball1, roomB)   := false;
[start] at_obj(ball2, roomB)   := false;
```

PROFMAT

Problem Definition: Initial State and Goal (3/3)

```
[start] at_obj(ball1, roomC)    := false;
[start] at_obj(ball2, roomC)    := false;
[start] allowed(eve, roomA)     := false;
[start] allowed(eve, roomB)     := true;
[start] allowed(eve, roomC)     := true;
[start] allowed(wally, roomA)   := true;
[start] allowed(wally, roomB)   := true;
[start] allowed(wally, roomC)   := false;
[start] forall(Obj o, Robot r) { carry(o, r) := false; };
move_time(wally) := 20;
move_time(eve)   := 10;
[start] battery(wally)    := 100;
[start] battery(eve)      := 100;

[end] at_obj(ball1, roomC) == true;
[end] at_obj(ball2, roomC) == true;
```

PROFMAT

Generated Plan

```
;; A plan has been found:
0.5: pick(ball1, roomA, wally) [1]
1.5: move(wally, roomA, roomB) [20]
21.5: move(eve, roomB, roomB) [10]
31.5: drop(ball1, roomB, wally) [1]
32.5: move(wally, roomB, roomA) [20]
52.5: pick(ball1, roomB, eve) [1]
53.5: move(eve, roomB, roomC) [10]
63.5: pick(ball2, roomA, wally) [1]
64.5: drop(ball1, roomC, eve) [1]
64.5: move(wally, roomA, roomB) [20]
84.5: move(eve, roomC, roomB) [10]
93.5: drop(ball2, roomB, wally) [1]
104.5: pick(ball2, roomB, eve) [1]
105.5: move(eve, roomB, roomC) [10]
116.5: drop(ball2, roomC, eve) [1]
```

PROFMAT

ANML Solver: TAMER

- To run and test your ANML models, you can use the TAMER solver, developed by FBK. You can download it **here**. The package includes:
  1. The solver executable
  2. A folder `examples` containing various ready-to-run ANML examples, which are a great starting point to explore ANML syntax and modeling patterns

PROFMAT