in pddl ci sono vari livelli
level 1=pddl1.0
level 2=level 1+Numbers
level 3=level 2+time

PDDL2.1

Dealing with Temporal and Numerical Planning

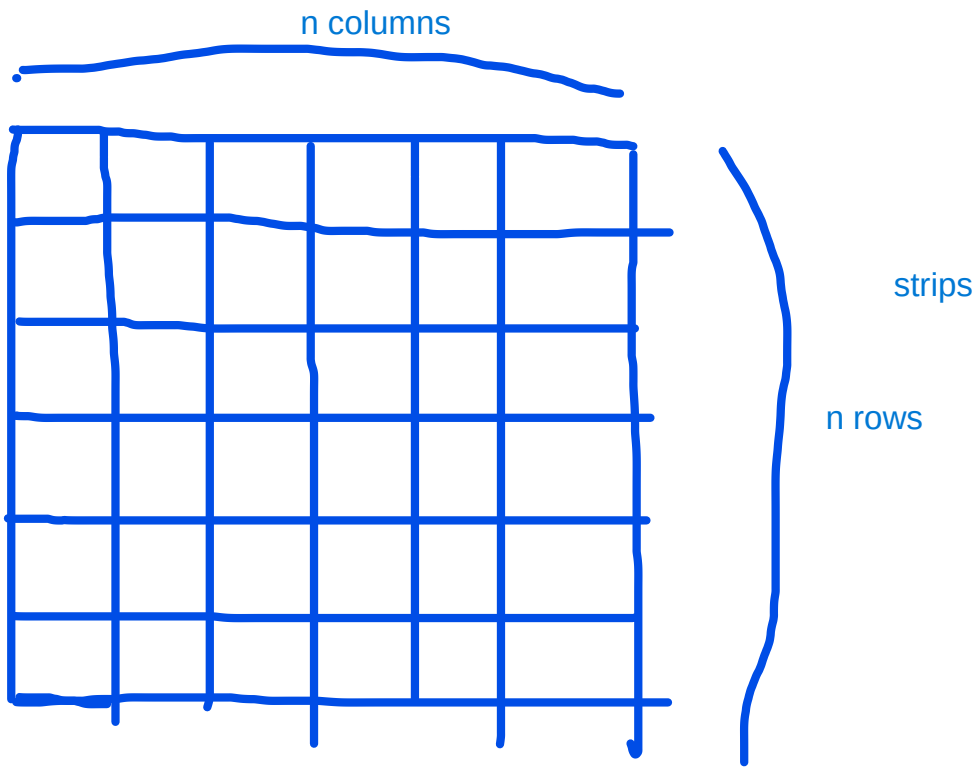
Matteo Cardellini

Università degli Studi di Genova

model problem
in pddl <-> in strips
pero in pddl
i problemi di livello 1 si possono fare in livello 2 che a loro volta possono essere fatti in level 3 ma non al contrario

Motivating Example

In classical PDDL, we can only model *instantaneous* actions (i.e., actions with no duration) which handle only boolean predicates. In the real world, instead, sometimes actions have a *duration* (*Temporal*) or need to manage numerical variables (*Numerical*).



Motivating Example

In classical PDDL, we can only model *instantaneous* actions (i.e., actions with no duration) which handle only boolean predicates. In the real world, instead, sometimes actions have a *duration* (*Temporal*) or need to manage numerical variables (*Numerical*).

Let's consider this example:

> *Wally and Eve are two robots which needs to cooperate to move two balls from Room A to Room C. Wally is allowed in room A and in a passage room B. Eve, instead, is allowed in room B and room C.*

Let's see how this would be modelled in classical PDDL (from now on, PDDL1.0) and then we will extend the domain to a point in which it can no longer be modelled with PDDL1.0, and we will introduce PDDL2.1.

Motivating Example - Domain File

```
(define (domain robot)
    (:requirements :strips :typing) (:types room obj robot)
    (: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))

    (:action move
        :parameters     (?r - robot ?a - room ?b - room)
        :precondition   (and  (at-robot ?r ?a) (allowed ?r ?b))
        :effect         (and  (at-robot ?r ?b) (not (at-robot ?r ?a))))

    (:action pick
        :parameters     (?o - obj ?l - room ?r - robot)
        :precondition   (and  (at-obj ?o ?l) (at-robot ?r ?l) (free ?r))
        :effect         (and (carry ?o ?r) (not (at-obj ?o ?l)) (not (free ?r))))

    (:action drop
        :parameters     (?o - obj ?l - room ?r - robot)
        :precondition   (and  (carry ?o ?r) (at-robot ?r ?l))
        :effect         (and (at-obj ?o ?l) (free ?r) (not (carry ?o ?r)))))
```

Motivating Example - Problem File

```
(define (problem pb1)
    (:domain robot)
    (:objects
        roomA - room roomB - room roomC - room
        ball1 - obj ball2 - obj eve - robot
        wally - robot)
    (:init
        (at-robot wally roomA)
        (at-robot eve roomB)
        (free wally)
        (free eve)
        (at-obj ball1 roomA)
        (at-obj ball2 roomA)
        (allowed wally roomA)
        (allowed wally roomB)
        (allowed eve roomB)
        (allowed eve roomC)
    )
    (:goal (and (at-obj ball1 roomC) (at-obj ball2 roomC)))
)
```

Motivating Example - Plan Found

```
(pick ball2 roomA wally)
(move wally roomA roomB)
(drop ball2 roomB wally)
(pick ball2 roomB eve)
(move eve roomB roomC)
(drop ball2 roomC eve)
(move wally roomB roomA)
(pick ball1 roomA wally)
(move wally roomA roomB)
(drop ball1 roomB wally)
(move eve roomC roomB)
(pick ball1 roomB eve)
(move eve roomB roomC)
(drop ball1 roomC eve)
```

Motivating Example - Plan Found

```
(pick ball2 roomA wally)
(move wally roomA roomB)
(drop ball2 roomB wally)
(pick ball2 roomB eve)
(move eve roomB roomC)
(drop ball2 roomC eve)
(move wally roomB roomA)
(pick ball1 roomA wally)
(move wally roomA roomB)
(drop ball1 roomB wally)
(move eve roomC roomB)
(pick ball1 roomB eve)
(move eve roomB roomC)
(drop ball1 roomC eve)
```

As it can be seen, the generated plan consists of a sequence of *instantaneous* actions that brings the *world* from an initial state (i.e., the two balls are in Room A) to the goal state (i.e., the two balls are in Room C)

Motivating Example - Real World Complications

Since the two robots have not been equipped with teleportation, we could imagine that the two robots take some time to move between two rooms.

Motivating Example - Real World Complications

Since the two robots have not been equipped with teleportation, we could imagine that the two robots take some time to move between two rooms.

In some cases, even for actions with durations, it is still possible to use PDDL1.0 to find the sequence of actions and then execute them in order, waiting for every action to complete before starting the next one. In this example, though, some actions could be executed in parallel: for example, `wally` shouldn't wait for `eve` to move from `roomB` to `roomC` in order to move back to `roomA`.

Motivating Example - Real World Complications

Since the two robots have not been equipped with teleportation, we could imagine that the two robots take some time to move between two rooms.

In some cases, even for actions with durations, it is still possible to use PDDL1.0 to find the sequence of actions and then execute them in order, waiting for every action to complete before starting the next one. In this example, though, some actions could be executed in parallel: for example, `wally` shouldn't wait for `eve` to move from `roomB` to `roomC` in order to move back to `roomA`.

Moreover, the two robots could have different speeds, making it difficult to compute the real interleaving of actions.

PDDL2.1 - Durative Actions

To model actions with durations, PDDL2.1 introduces the concept of `durative-actions`. The syntax is as follows:

```
(:durative-action <action_name>
    :parameters (<arguments>)
    :duration (= ?duration <duration_number>)
    :condition (<logical_expression>)
    :effect (<logical_expression>)
)
```

The durative action has an additional field called `:duration` which defines its duration in terms of the value of the reserved variable *?duration*. The number specified is expressed in *time units (t.u.)* which changes based on the domain.

N.B.: the field's name for the precondition has changed from `:precondition` to `:condition`.

PDDL2.1 - Temporal Operators

PDDL2.1 introduces three different operators for the `<logical_expression>` inside the conditions and effect's fields: these are `at start`, `at end` and `over all`. These operators help to model *when* an action preconditions will be checked and effects will be applied.

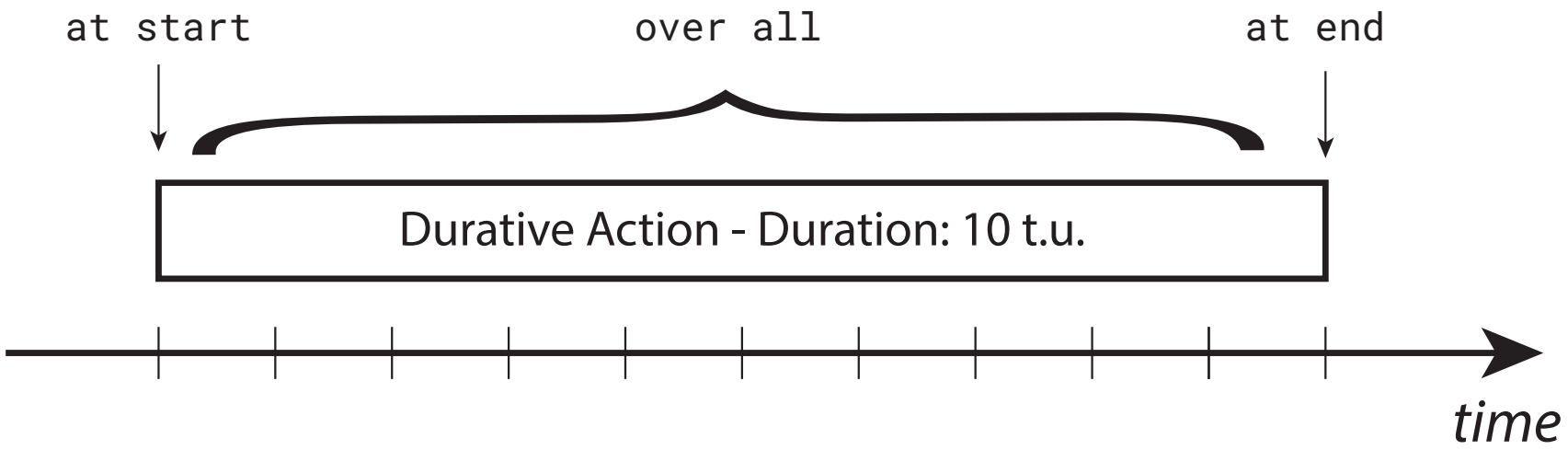


Figure: Representation of a durative action during 10 time units

PDDL2.1 - Numeric Fluents I

In PDDL2.1 the possibility to specify numerical values (*fluents*) was introduced together with some operators to reason upon them and to change their value. These fluents have to be specified in the domain file with the following syntax:

```
(:functions
    (<variable_name> <parameter_name> - <object_type>)
    ...
    (<variable_name> <parameter_name> - <object_type>)
)
```

PDDL2.1 - Numeric Fluents I

In PDDL2.1 the possibility to specify numerical values (*fluents*) was introduced together with some operators to reason upon them and to change their value. These fluents have to be specified in the domain file with the following syntax:

```
(:functions
    (<variable_name> <parameter_name> - <object_type>)
    ...
    (<variable_name> <parameter_name> - <object_type>)
)
```

Let's suppose that, in our domain, we would like to define a numeric variable expressing the time it takes to move between rooms for every robot (assuming the room are equidistant with each other). Moreover, in our domain the robots have a limited battery duration and every `move` action consumes a fixed amount of battery.

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

PDDL2.1 - Numeric Fluents II

In the problem file, it is then possible to initialize the fluents with the = operator in the `:init` section.

```
(:init
    ...
    (allowed eve roomB)
    (allowed eve roomC)
    (= (move-time wally) 20)
    (= (move-time eve) 10)
    (= (battery wally) 100)
    (= (battery eve) 100)
)
```

The changed durative-action

Here is how the `move` action has changed:

```
(: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))))
```

The changed durative-action

Here is how the `move` action has changed:

```
(: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))))
```

In the preconditions, we check that the robot is in the correct room and has enough battery for the trip. We also check that during the action, the robot is allowed to move to the destination room.
In the effects, at the beginning of the action, we remove the robot from the departure room and, at the end of the action, we decrease the battery, and we signal that the robot has arrived at the destination room.

PDDL2.1 - Numeric Fluents III

In the example, we have seen the operator `(decrease ?var <amount>)` in the effects, which subtracted the specified amount from a variable. Similarly, the operator `(increase ?var <amount>)` is also available.

PDDL2.1 - Numeric Fluents III

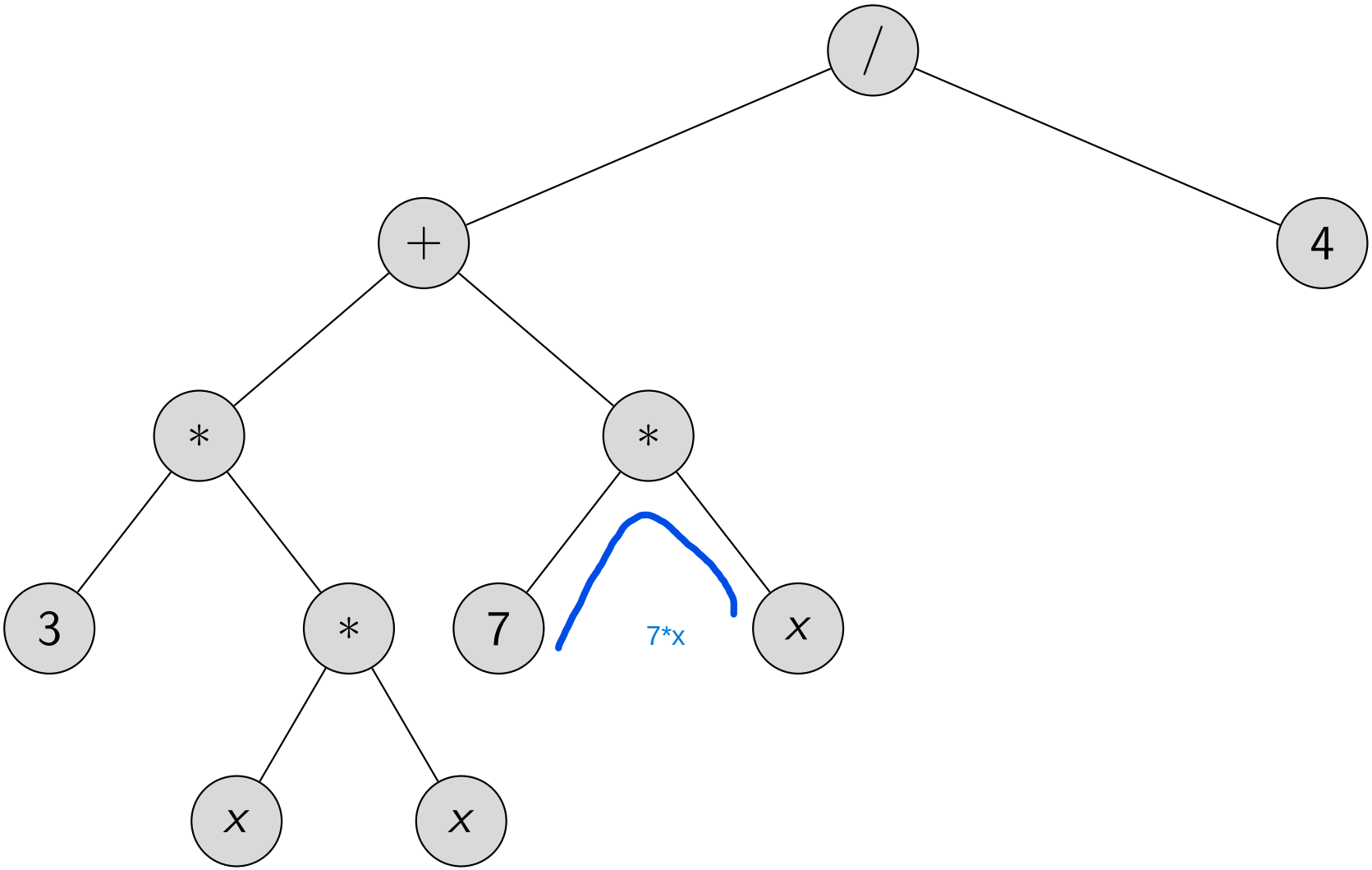
In the example, we have seen the operator `(decrease ?var <amount>)` in the effects, which subtracted the specified amount from a variable. Similarly, the operator `(increase ?var <amount>)` is also available.

PDDL2.1 also introduced numeric operators to deal with arithmetic operations. For example:

```
<exp> = (+ <exp> <exp>) | //Addition
        (- <exp> <exp>) | //Subtraction
        (/ <exp> <exp>) | //Division
        (* <exp> <exp>) | //Multiplication
        (<var>) |
        <const>
```

Example of Expression

$$\frac{3x^2 + 7x}{4} =$$



Example of Expression

$$\frac{3x^2 + 7x}{4}$$

```
(/
  (+
    (*
      3
      (* (x)(x))
    )
    (* 7 (x))
  )
  4
)
```

Complete PDDL2.1 Domain File

```
(define (domain robot)
    (:requirements  :strips :typing :numeric-fluents :durative-actions)
    (:types         room obj robot)
    (:functions     (move-time ?r - robot) (battery ?r - robot))
    (: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))

    (:action pick
        :parameters     (?o - obj ?l - room ?r - robot)
        :precondition   (and  (at-obj ?o ?l) (at-robot ?r ?l) (free ?r))
        :effect         (and (carry ?o ?r) (not (at-obj ?o ?l)) (not (free ?r))))

    (:action drop
        :parameters     (?o - obj ?l - room ?r - robot)
        :precondition   (and  (carry ?o ?r) (at-robot ?r ?l))
        :effect         (and (at-obj ?o ?l) (free ?r)
                        (not (carry ?o ?r))))

    (: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)))))
```

Complete PDDL2.1 Problem File

```
(define (problem pb1)
    (:domain robot)
    (:objects
        roomA - room roomB - room roomC - room
        ball1 - obj ball2 - obj eve - robot
        wally - robot)
    (:init
        (at-robot wally roomA) (at-robot eve roomB)
        (free wally) (free eve)
        (at-obj ball1 roomA) (at-obj ball2 roomA)
        (allowed wally roomA) (allowed wally roomB)
        (allowed eve roomB) (allowed eve roomC)
        (= (move-time wally) 20) (= (move-time eve) 10)
        (= (battery wally) 100) (= (battery eve) 100))
    (:goal (and (at-obj ball1 roomC)(at-obj ball2 roomC))))
```

The domain and problem files for PDDL1.0 and PDDL2.1 are available at `https://github.com/matteocarde/unige-aai`.

PDDL2.1 Plan

Now the plan is a *timestamped* ordered sequence of actions.
The presented domain and problem files produce the following plan:

```
$ lpg++ -o domain.pddl -f problem.pddl -n 1 -seed 1234
```

```
    Time:    (ACTION)                    [action Duration;   action Cost]
    00.0000: (PICK BALL1 ROOMA WALLY)    [D:0.00;            C:1.00]
    00.0000: (MOVE WALLY ROOMA ROOMB)    [D:20.00;           C:1.00]
    20.0000: (DROP BALL1 ROOMB WALLY)    [D:0.00;            C:1.00]
    20.0000: (MOVE WALLY ROOMB ROOMA)    [D:20.00;           C:1.00]
    40.0000: (PICK BALL2 ROOMA WALLY)    [D:0.00;            C:1.00]
    40.0000: (MOVE WALLY ROOMA ROOMB)    [D:20.00;           C:1.00]
    60.0000: (DROP BALL2 ROOMB WALLY)    [D:0.00;            C:1.00]
    60.0000: (PICK BALL2 ROOMB EVE)      [D:0.00;            C:1.00]
    60.0000: (MOVE EVE ROOMB ROOMC)      [D:10.00;           C:1.00]
    70.0000: (DROP BALL2 ROOMC EVE)      [D:0.00;            C:1.00]
    70.0000: (MOVE EVE ROOMC ROOMB)      [D:10.00;           C:1.00]
    80.0000: (PICK BALL1 ROOMB EVE)      [D:0.00;            C:1.00]
    80.0000: (MOVE EVE ROOMB ROOMC)      [D:10.00;           C:1.00]
    90.0000: (DROP BALL1 ROOMC EVE)      [D:0.00;            C:1.00]
```

The solver used is LPG (`https://lpg.unibs.it/lpg/`) for Linux.
NB: Parallelism is not yet achieved! Why ?

PDDL2.1 - Metric

In PDDL2.1, the possibility to specify a metric to evaluate the quality of a plan. In the problem file, we can specify which value we would like to optimize. After the *:goal* field we can write:

```
(:metric minimize total-time)
```

total-time is a reserved word which corresponds to the total planning time (in the plan just found it corresponds to 90) but we could specify any fluent.

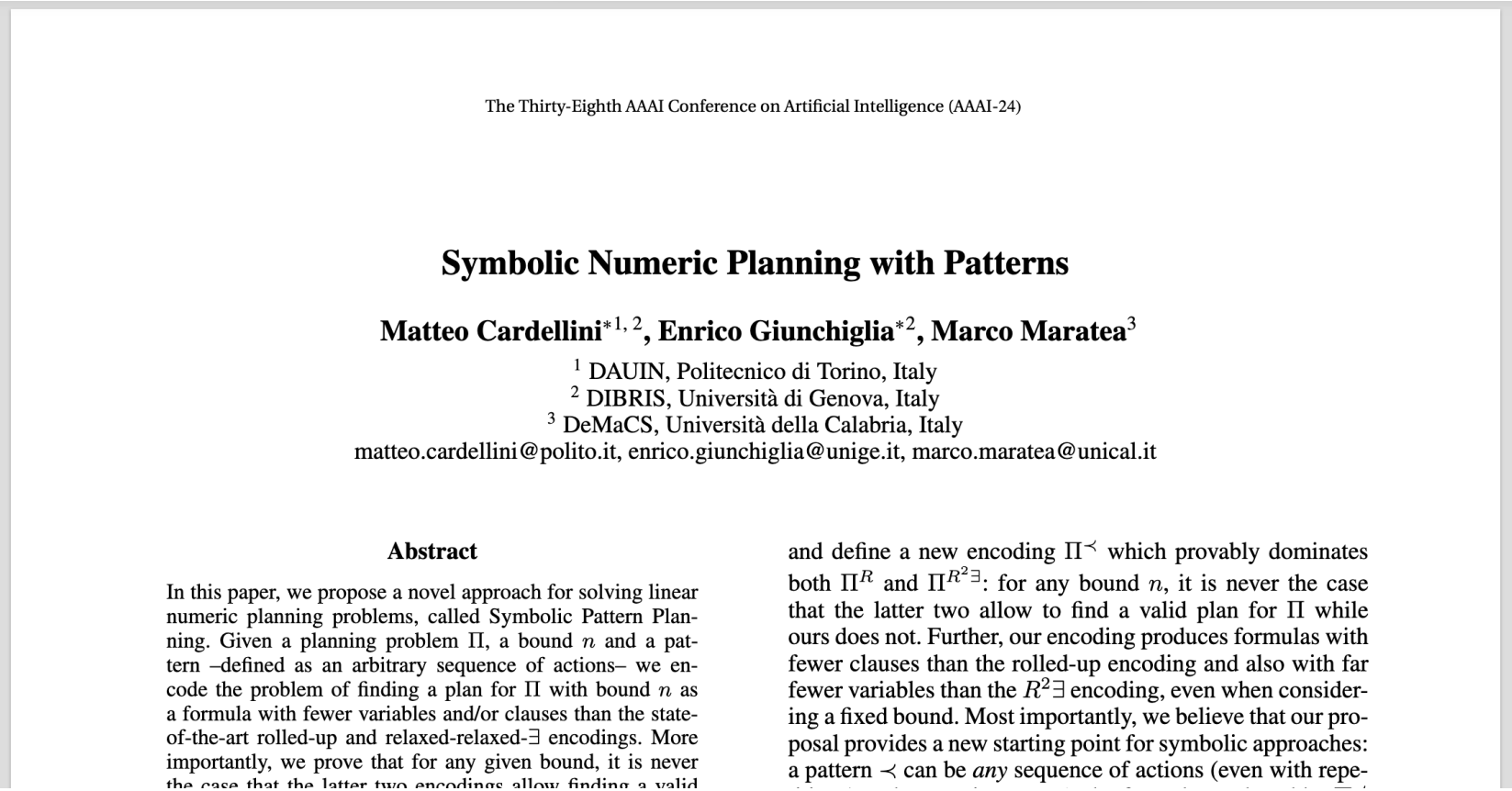
Planner of higher quality exists

Searching for more solutions with the LPG planner outputs the following plan:

```
$ lpg++ -o domain.pddl -f problem.pddl -n 2 -seed 1234
```

```
Time:    (ACTION)                    [action Duration;   action Cost]
00.0000: (PICK BALL1 ROOMA WALLY)    [D:0.00;            C:0.10]
00.0000: (MOVE WALLY ROOMA ROOMB)    [D:20.00;           C:0.10]
20.0000: (DROP BALL1 ROOMB WALLY)    [D:0.00;            C:0.10]
20.0000: (PICK BALL1 ROOMB EVE)      [D:0.00;            C:0.10]
20.0000: (MOVE EVE ROOMB ROOMC)      [D:10.00;           C:0.10]
20.0000: (MOVE WALLY ROOMB ROOMA)    [D:20.00;           C:0.10]
30.0000: (DROP BALL1 ROOMC EVE)      [D:0.00;            C:0.10]
30.0000: (MOVE EVE ROOMC ROOMB)      [D:10.00;           C:0.10]
40.0000: (PICK BALL2 ROOMA WALLY)    [D:0.00;            C:0.10]
40.0000: (MOVE WALLY ROOMA ROOMB)    [D:20.00;           C:0.10]
60.0000: (DROP BALL2 ROOMB WALLY)    [D:0.00;            C:0.10]
60.0000: (PICK BALL2 ROOMB EVE)      [D:0.00;            C:0.10]
60.0000: (MOVE EVE ROOMB ROOMC)      [D:10.00;           C:0.10]
70.0000: (DROP BALL2 ROOMC EVE)      [D:0.00;            C:0.10]
```

Numeric and Temporal are not fully solved!



The Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24)

Symbolic Numeric Planning with Patterns

Matteo Cardellini[*1, 2], **Enrico Giunchiglia**[*2], **Marco Maratea**[3]

[1] DAUIN, Politecnico di Torino, Italy
[2] DIBRIS, Università di Genova, Italy
[3] DeMaCS, Università della Calabria, Italy
matteo.cardellini@polito.it, enrico.giunchiglia@unige.it, marco.maratea@unical.it

Abstract

In this paper, we propose a novel approach for solving linear numeric planning problems, called Symbolic Pattern Planning. Given a planning problem Π, a bound n and a pattern –defined as an arbitrary sequence of actions– we encode the problem of finding a plan for Π with bound n as a formula with fewer variables and/or clauses than the state-of-the-art rolled-up and relaxed-relaxed-$\exists$ encodings. More importantly, we prove that for any given bound, it is never the case that the latter two encodings allow finding a valid

and define a new encoding $\Pi^{\prec}$ which provably dominates both Π^{R} and $\Pi^{R^2\exists}$: for any bound n, it is never the case that the latter two allow to find a valid plan for Π while ours does not. Further, our encoding produces formulas with fewer clauses than the rolled-up encoding and also with far fewer variables than the $R^2\exists$ encoding, even when considering a fixed bound. Most importantly, we believe that our proposal provides a new starting point for symbolic approaches: a pattern $\prec$ can be *any* sequence of actions (even with repe-

Figure: Paper Published at the 38th Conference for the Advancement of Artificial Intelligence held in Vancouver, Canada on February 2024

Numeric and Temporal are not fully solved!

Temporal Numeric Planning with Patterns

Anonymous Submission

Abstract

We consider the task of solving temporal numeric planning problems expressed in PDDL2.1. Given a temporal numeric planning problem Π and a bound n, we show how it is possible to produce an SMT formula (i) whose models correspond to valid plans of Π, and (ii) which extends the recently proposed planning with patterns approach from the numeric to the temporal case, thus allowing finding valid plans for Π with smaller values of n compared to standard approaches based on reduction to an SMT formula. The experimental analysis, which considered all the publicly available temporal planners, testifies to the effectiveness of our approach.

1 Introduction

We consider temporal numeric planning problems expressed in PDDL2.1 [Fox and Long, 2003]. Differently from the classical Bofill *et al.*, 2017]) and the action rolling R encoding [Scala *et al.*, 2016b]), and allows finding valid plans for Π with smaller values of n compared to standard approaches with effect and explanatory frame axioms. Consider, for instance, the problem with a set $\{1, \ldots, q\}$ of bottles, the first p of which containing l_i litres of liquid ($i \in [1, p]$), and the action of pouring from the i-th bottle in $[1, p]$ into the j-th bottle in $(p, q]$, one litre every $d_{i,j}$ seconds. Assuming the bottles are big enough, with a standard symbolic encoding, the goal of emptying the bottles in $[1, p]$ can take up to $n = \sum_{i=1}^{p} l_i$ steps, how many depending also on the specific $d_{i,j}$ values, and will always take $\sum_{i=1}^{p} l_i$ steps when either $p = 1$ or $q = p + 1$, due to the conflicting effects of pouring from or to a single bottle. Our encoding allows to solve such problem with $n = 1$ independently of the actions' durations and the values of p and q.

Experimental analysis conducted on various temporal problems, some of which introduced by us to cover different cases of "required concurrency" [Cushing *et al.*, 2007], high-

Figure: Under Review

Additional topics

- Valid PDDL1.0 instances are also valid PDDL2.1 instances (PDDL2.1 is *strictly more expressive* than PDDL1.0). A PDDL1.0 instance, run with a PDDL2.1 planner, will output a plan in which every action has duration 0.
- More information about PDDL2.1 can be found in the seminal paper of M. Fox, D. Long - `https://arxiv.org/pdf/1106.4561.pdf`
- More information about the planner LPG (`https://www.aaai.org/Papers/AIPS/2002/AIPS02-002.pdf`)