HTN Planning with Semantic Attachments

Maurício Cecílio Magnaguagno and Felipe Meneguzzi
Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre - RS, Brazil

Symbolic-Geometric planning

- Usually solved by separate planners/solvers
  - One solver is the main program that is able to call other solvers
  - Constraints discovered by each solver must be transmitted to the other
    - May require replanning (costly)

- Why not solve most of the problem with one planner/solver?
  - Use external solvers not as one big black-box that returns plans
  - Use external solvers as small smart-unification engines

Classical vs Hierarchical Planning

Classical

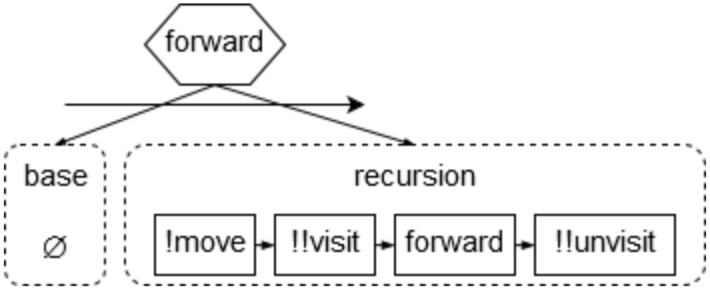
- Actions
  - Easier to modify
- Goal-oriented
- Planner controls plan quality
  - Decisions are built-in
- Speed/memory is limited by planner
  - Better planners are required
- Constant set of objects
  - Easier to optimize (enumerate)

Hierarchical

- Actions + Methods
  - Easier to control
- Task-oriented
- Description controls plan quality
  - Decision are external
- Speed/memory is limited by description
  - Better methods are required
- Dynamic set of objects
  - **Easier to handle continuous/external values**
    - Common in motion/temporal planning

Hierarchical Planning

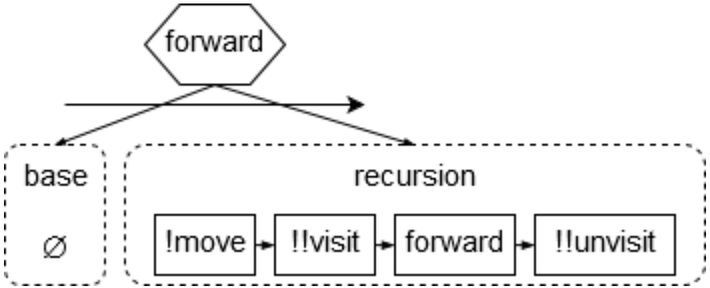
- Mostly symbolic
  - Discretization
  - User provided “recipes”
  - Support numeric operations, external calls
- Less decisions than classical planning
  - More control, easier to extend
  - Follow tasks → methods → subtasks
- Task list



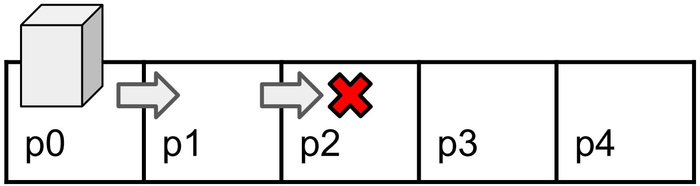

```
(defdomain search (; This is a JSHOP description
  (:operator (!move ?agent ?from ?to)
      ( (at ?agent ?from) (adjacent ?from ?to) )
      ( (at ?agent ?from) )
      ( (at ?agent ?to) ) )
  (:operator (!!visit ?agent ?pos)
      ()
      ()
      ( (visited ?agent ?pos) ) )
  (:operator (!!unvisit ?agent ?pos)
      ()
      ( (visited ?agent ?pos) )
      () )
  (:method (forward ?agent ?goal)
      base
      ( (at ?agent ?goal) )
      ()
      recursion
      (
        (at ?agent ?from)
        (adjacent ?from ?place)
        (not (visited ?agent ?place)) )
      (
        (!move ?agent ?from ?place)
        (!!visit ?agent ?from)
        (forward ?agent ?goal)
        (!!unvisit ?agent ?from) ) ) )
```

Hierarchical Planning

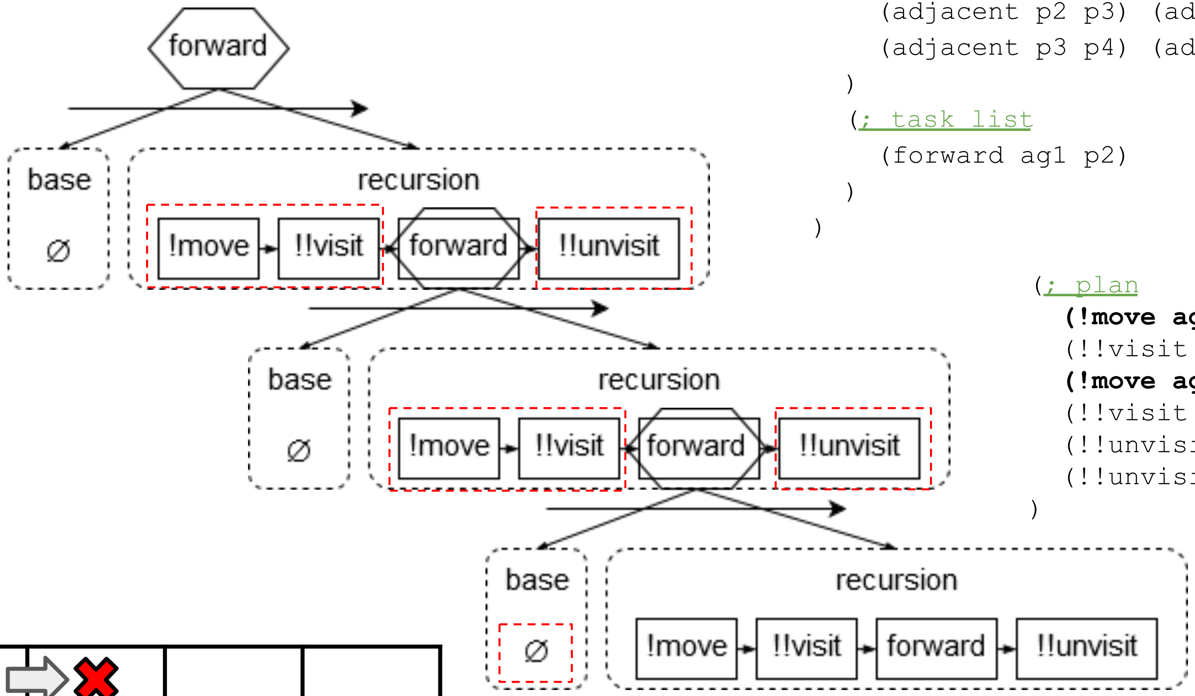
- Mostly symbolic
  - Discretization
  - User provided “recipes”
  - Support numeric operations, external calls
- Less decisions than classical planning
  - More control, easier to extend
  - Follow tasks → methods → subtasks
- Task list

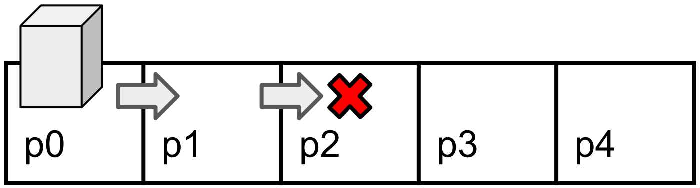



```
(defproblem pb1 search
  (; initial state
    (at ag1 p0)
    (adjacent p0 p1) (adjacent p1 p0)
    (adjacent p1 p2) (adjacent p2 p1)
    (adjacent p2 p3) (adjacent p3 p2)
    (adjacent p3 p4) (adjacent p4 p3)
  )
  (; task list
    (forward ag1 p2)
  )
)
```



Hierarchical Planning



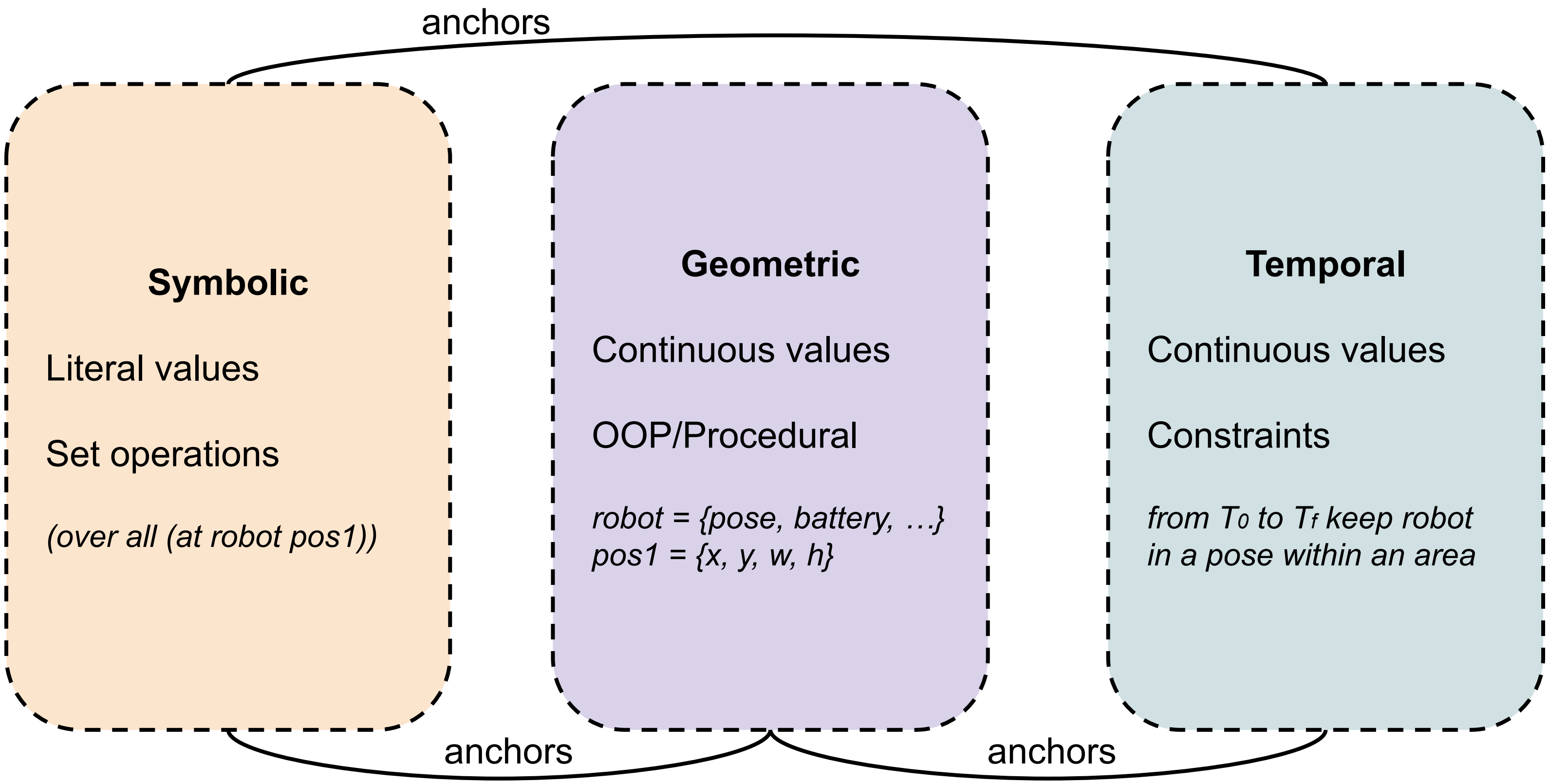



```
(defproblem pb1 search
  (; initial state
    (at ag1 p0)
    (adjacent p0 p1) (adjacent p1 p0)
    (adjacent p1 p2) (adjacent p2 p1)
    (adjacent p2 p3) (adjacent p3 p2)
    (adjacent p3 p4) (adjacent p4 p3)
  )
  (; task list
    (forward ag1 p2)
  )
)
```

```
(; plan
  (!move ag1 p0 p1)
  (!!visit   ag1 p0)
  (!move ag1 p1 p2)
  (!!visit   ag1 p1)
  (!!unvisit ag1 p1)
  (!!unvisit ag1 p0)
)
```

Planning Challenges

- Hard to compare numeric values
  - Discretization or limited exponent/mantissa
  - Numeric error, is 1.00001 = 1 or 100000 = 100001?
- Hard/impossible to access external functions/structures
  - Usually only literals or numeric values
  - No support for objects (OOP) such as points, lines, polygons…
- How to handle geometric/temporal definitions as symbols
  - Can we **anchor** symbols to external structures?


anchors
Symbolic
Literal values
Set operations
(over all (at robot pos1))
Geometric
Continuous values
OOP/Procedural
robot = {pose, battery, …}
pos1 = {x, y, w, h}
Temporal
Continuous values
Constraints
from T_0 to T_f keep robot
in a pose within an area
anchors
anchors

Symbolic ⇐ anchors ⇒ Geometric/Temporal/Object

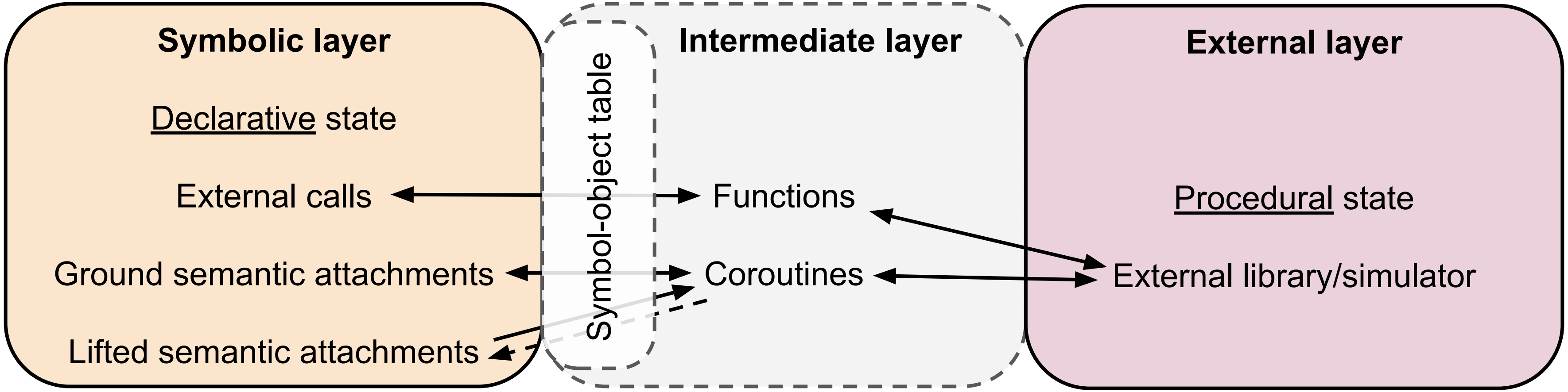
Question: is it possible to perform symbolic-geometric planning efficiently by dynamically generating symbolic anchors to external objects?

Goal: Our main goal is to obtain a symbolic-geometric planning approach that is both competitive and easier to describe domains when compared with other approaches, that either precompute a lot of data or are limited by a fixed number of anchors between the symbolic and geometric layers.

Symbolic-Geometric Planning

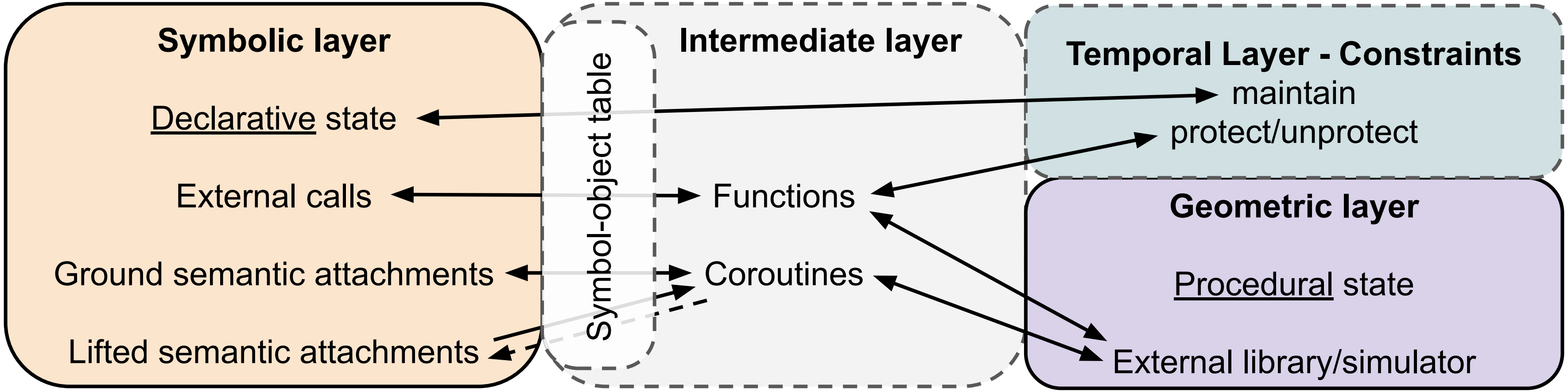
- Extend HTN planning and descriptions
  - More procedural than classical planning/PDDL
  - Better control over which decisions/outside calls are made during planning
- Generate anchors during planning
  - `position1 = (x, y)`
  - `polygon2 = (p1, p2, ..., pn)`
  - `robot = (pose, speed, battery, parts, ...)`
- Support external calls with anchors instead of numeric constructions
  - `(call < (call distance 0 0 10 4) 3)`
  - `(call = (call distance  p1   p2) near)` ⇐ More readable
- Break problem in layers

Layers



Symbolic layer
Declarative state
External calls
Ground semantic attachments
Lifted semantic attachments
Symbol-object table
Intermediate layer
Functions
Coroutines
External layer
Procedural state
External library/simulator

Layers


Symbolic layer
Declarative state
External calls
Ground semantic attachments
Lifted semantic attachments
Symbol-object table
Intermediate layer
Functions
Coroutines
Temporal Layer - Constraints
maintain
protect/unprotect
Geometric layer
Procedural state
External library/simulator

Coroutines / Semi-coroutines / Generators

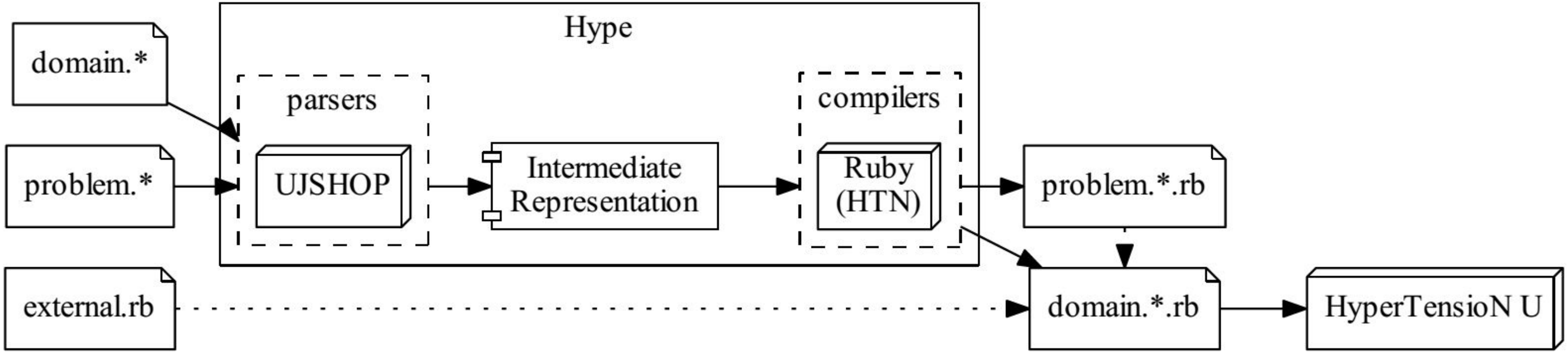
- Subroutines for non-preemptive multitasking
- Execution can be suspended and resumed
- Able to implement
  - Cooperative tasks
  - Iterators
  - Infinite lists

- Semi-coroutines = weaker co-routines
  - Main routine has control
  - Coroutine can save state and resume main routine

```
define consecutive⟨from, n)
  for i ← from to from + n
    yield i, i+1

for ⟨a, b⟩ in consecutive(5, 3)
  print ⟨a, b, a+b⟩
```

```
⟨5, 6, 11⟩
⟨6, 7, 13⟩
⟨7, 8, 15⟩
⟨8, 9, 17⟩
```

Framework



Hype
domain.*
parsers
compilers
problem.*
UJSHOP
Intermediate
Representation
Ruby
(HTN)
problem.*.rb
external.rb
domain.*.rb
HyperTensioN U

Reorder preconditions during compilation phase

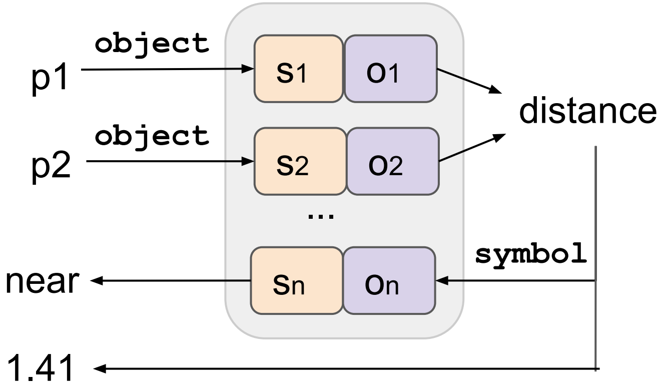
```
(:attachments (sa1 ?a ?b) (sa2 ?a ?b))
(:method (m ?t1 ?t2)
  label
  (; preconditions
    (call != ?t1 ?t2) ; no dependencies
    (call != ?fv1 ?fv2) ; ?fv1 and ?fv2 dependencies
    (sa1 ?t1 ?fv1)   ; no dependencies, ground ?fv1
    (pre1 ?t1 ?t2)   ; no dependencies
    (sa2 ?fv1 ?fv2)  ; ?fv1 dependency, ground ?fv2
    (pre2 ?fv3 ?fv1) ; ?fv1 dependency, ground ?fv3
  )
  (; subtasks
    (subtask ?t1 ?t2 ?fv1 ?fv2)
  )
)
```

```
define m(t1, t2)
  if t1 ≠ t2
    for each fv1,fv3; state ⊂ {⟨pre1,t1,t2⟩, ⟨pre2,fv3,fv1⟩}
      for each sa1(t1, fv1)
        free variable fv2
        for each sa2(fv1, fv2)
          if fv1 ≠ fv2
            decompose([⟨subtask, t1, t2, fv1, fv2⟩])
```

Symbol-object table

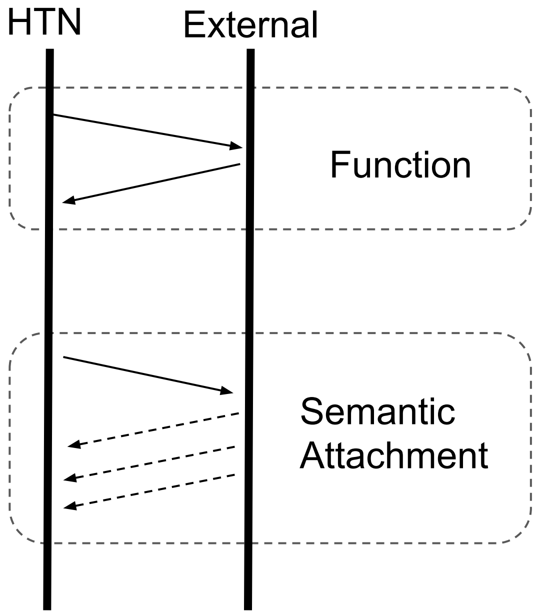
- Convert a symbol to an object and vice-versa
  - position1 ⇒ (x: 20, y: 18)
- Equivalent objects in the geometric layer ⇒ same symbol
  - Easier to compare (table already did the comparison when computed)
  - Easier to debug (user control generated literal names)

```
define distance(p1, p2)
  o1 = object(p1)
  o2 = object(p2)
  return symbol(hypot(x(o1) - x(o2), y(o1) - y(o2)))
```



Semantic attachments

- Avoid complex preconditions and effect descriptions (update state)
- Easier to be computed in a lazy way (iterative)
- Describe them externally to the planner
    - `(:attachments (my-attachment ?param1 ?param2))`
    - Replace by other implementations if necessary
    - Minimal modification over original language (easily reproducible)
- Usage is the same as common predicates
    - Easily replace declarative aspects with procedures

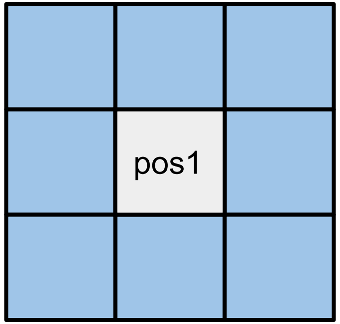


Example - adjacent

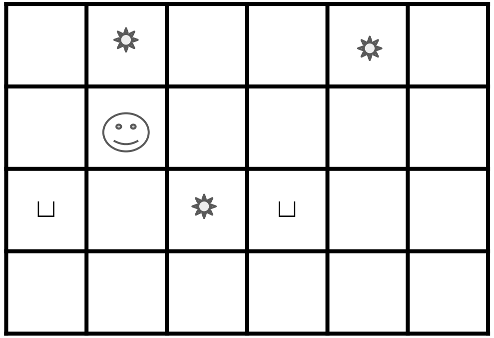
```
constant WIDTH = 5, HEIGHT = 5
constant DIRECTIONS = {⟨-1,-1⟩, ⟨0,-1⟩, ⟨1,-1⟩, ⟨-1,0⟩, ⟨1,0⟩, ⟨-1,1⟩, ⟨0,1⟩, ⟨1,1⟩}

define adjacent(pos1, pos2)
  pos1 ← object(pos1)
  if pos2 is ground
    pos2 ← object(pos2)
    if |x(pos1) - x(pos2)| ≤ 1 ∧ |y(pos1) - y(pos2)| ≤ 1
      yield
  else if pos2 is free
    for each ⟨x, y⟩ ∈ DIRECTIONS
       nx ← x + x(pos1)
       ny ← y + y(pos1)
       if 0 ≤ nx < WIDTH ∧ 0 ≤ ny < HEIGHT
          pos2 ← symbol(⟨nx, ny⟩)
          yield
```

Ground - test and resume

Lifted - unify and resume



Domains and Experiments - Plant Watering / Gardening



```
define adjacent(x, y, nx, ny, gx, gy)
  x ← numeric(x)
  y ← numeric(y)
  gx ← numeric(gx)
  gy ← numeric(gy)
  ; compare returns -1, 0, 1 for <, =, >, respectively
  nx ← symbol(x + compare(gx, x))
  ny ← symbol(y + compare(gy, y))
  yield
```

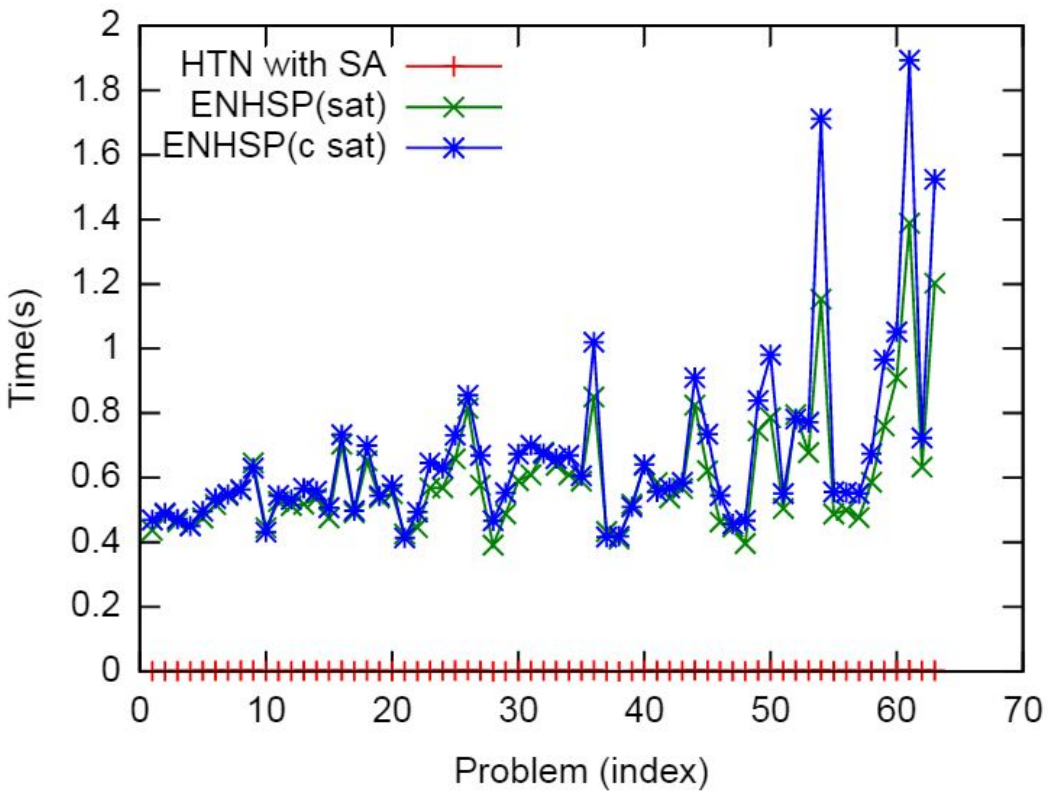
```
(:attachments (adjacent ?x ?y ?nx ?ny ?gx ?gy))
(:method (travel ?a ?gx ?gy)
  base
  (; preconditions
    (call = (call function (x ?a)) ?gx)
    (call = (call function (y ?a)) ?gy)
  )
  () ; empty subtasks
  keep_moving
  (; preconditions
    (adjacent
      (call function (x ?a))
      (call function (y ?a))
      ?nx ?ny
      ?gx ?gy)
  )
  (; subtasks
    (!move ?a ?nx ?ny)
    (travel ?a ?gx ?gy)
  )
)
```

Domains and Experiments - Plant Watering / Gardening

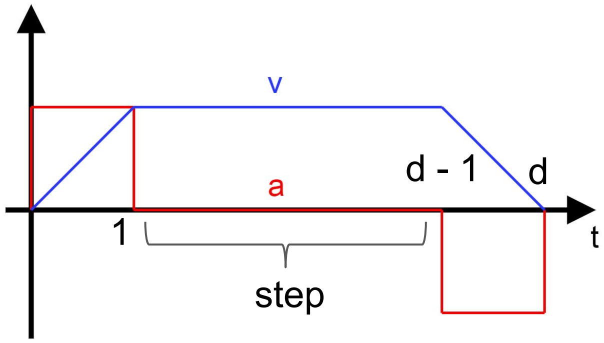
```
(:attachments (adjacent ?x ?y ?nx ?ny ?gx ?gy))
(:method (travel ?a ?gx ?gy)
  base
  (; preconditions
    (call = (call function (x ?a)) ?gx)
    (call = (call function (y ?a)) ?gy)
  )
  () ; empty subtasks
  keep_moving
  (; preconditions
    (adjacent
      (call function (x ?a))
      (call function (y ?a))
      ?nx ?ny
      ?gx ?gy)
  )
  (; subtasks
    (!move ?a ?nx ?ny)
    (travel ?a ?gx ?gy)
  )
```

```
define travel(a, gx, gy)
  if x(a) = gx ∧ y(a) = gy
    decompose([])
  free variables nx, ny
  for each adjacent(x(a), y(a), nx, ny, gx, gy)
    decompose([
      ⟨move, a, nx, ny⟩,
      ⟨travel, a, gx, gy⟩
    ])
```

Domains and Experiments - Plant Watering / Gardening



Domains and Experiments - Car Linear

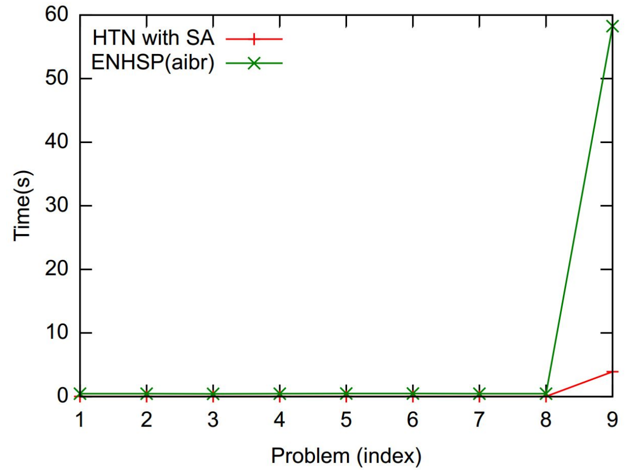



```
(:- (speed_limit ?time)
  (and
    (assign ?vt (call function v ?time))
    (assign ?max (call function max_speed))
    (call >= ?vt (call - 0 ?max))
    (call <= ?vt ?max)
  )
)
```

```
(:attachments (step ?t ?min ?max ?step))
(:method (forward ?min_dest ?max_dest)
  base
  ()
  ((!!test_destination ?min_dest ?max_dest 0))
  keep_moving
  ((step ?deadline 1))
  (
    (!start_car 0 ?deadline)
    (!accelerate 0)
    (!decelerate 1)
    (!decelerate (call - ?deadline 1))
    (!accelerate ?deadline)
    (!stop_car ?deadline)
    (!!test_destination ?min_dest ?max_dest ?deadline)
  )
)
```

Processes: acceleration ⇒ speed ⇒ position

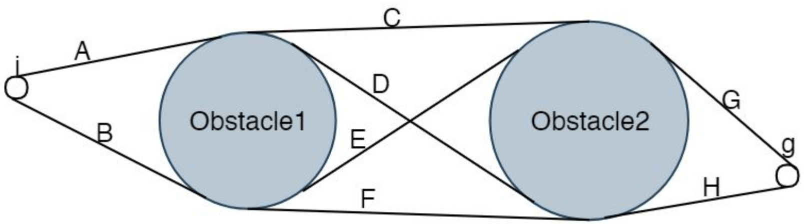
Domains and Experiments - Car Linear



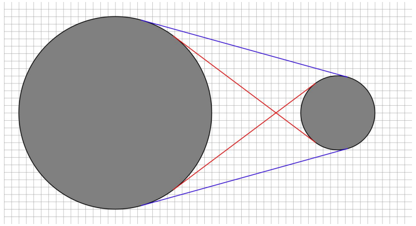
Problem	1	2	3	4	5	6	7	8	9
ENHSP(aibr)	0.461	0.462	0.427	0.461	0.475	0.474	0.443	0.466	58.256
HTN with SA	0.015	0.015	0.015	0.015	0.015	0.015	0.015	0.015	03.920

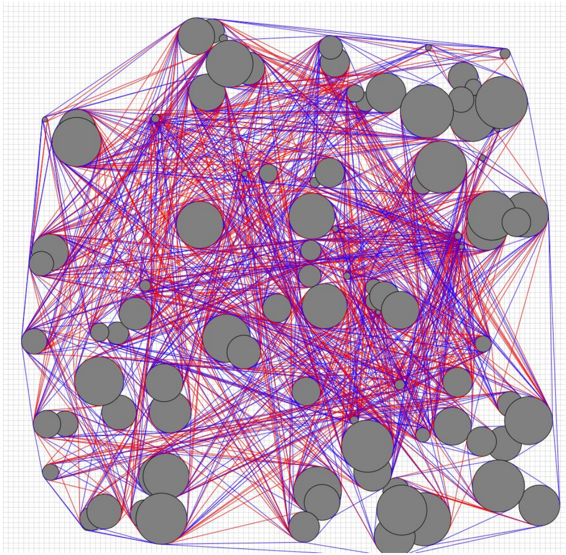
Domains and Experiments - Bitangent movement

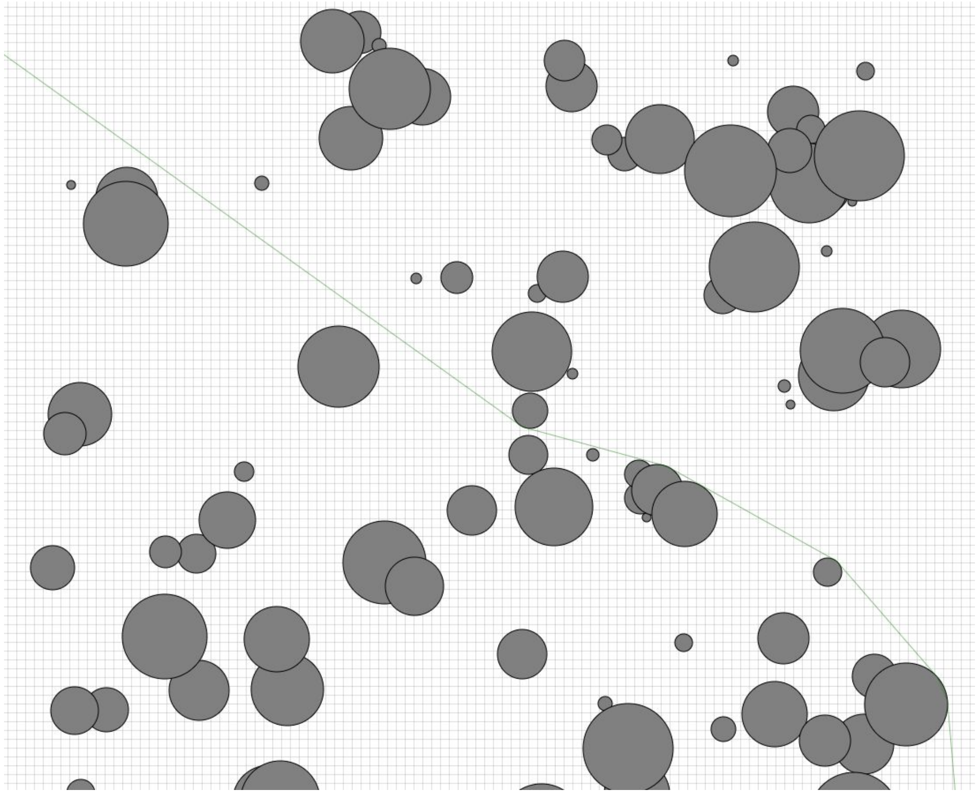
- Use external motion planner vs calculate continuous path during planning
- Bitangent search / Dubins path
  - ACG, ADH, BEG, BFH



Domains and Experiments - Bitangent movement







Domains and Experiments - Bitangent movement

```
(:method (forward ?agent ?goal)
  base
  ((at ?agent ?goal)) ; preconditions
  () ; empty subtasks
  search
  (; preconditions
    (at ?agent ?start)
    (call search-circular ?agent ?start ?goal)
  )
  ; subtasks
  ((apply-plan ?agent ?start 0 (call plan-size)))
)
```

```
(:method (apply-plan ?agent ?from ?index ?size)
  index-equals-size
  ((call = ?index ?size)) ; preconditions
  () ; empty subtasks
  get-next-action
  ; preconditions
  ((assign ?to (call plan-position ?index)))
  (; subtasks
    (!move ?agent ?from ?to)
    (apply-plan ?agent ?to (call + ?index 1) ?size)
  )
)
```

First option: call external motion planner and consume steps

Domains and Experiments - Bitangent movement

```
(:attachments (closest ?circle ?to ?outcircle
?indir ?outdir ?goal))

(:method (forward-attachments ?agent ?goal)
  clockwise
  ((at ?agent ?start)) ; preconditions
  (; subtasks
    (loop ?agent ?start ?start clock ?goal)
  )
  counter-clockwise
  ((at ?agent ?start)) ; preconditions
  (; subtasks
    (loop ?agent ?start ?start counter ?goal)
  )
)
```

```
(:method (loop ?agent ?from ?circle ?indir ?goal)
  base
  ((call visible ?from ?circle ?goal)) ; preconditions
  ((!move ?agent ?from ?goal)) ; subtasks
  recursion
  (; preconditions
    (closest ?circle ?to ?outcircle ?indir ?outdir ?goal)
    (not (visited ?agent ?to))
  )
  (; subtasks
    (!move ?agent ?from ?to)
    (!!visit ?agent ?from)
    (loop ?agent ?to ?outcircle ?outdir ?goal)
    (!!unvisit ?agent ?from)
  )
)
```

Second option: implement motion planner as part of symbolic description

Conclusions

- HTN Planning with Semantic Attachments
  - Flexibility
    - No preprocessing
    - No limited amount of anchors (symbols)
    - Able to describe more problems (realistically)
  - External elements expand possibilities
    - Debug with readable object names
    - Geometry/physics libraries
  - Future work
    - Formalization of semantic attachments
    - Support non DFS-based HTN planners

- Available at https://github.com/Maumagnaguagno/HyperTensioN_U