# Hierarchical Task Network Planning

AI 20-21 Section 1, Planning - 3

Thanks to Francesco Riccio, Guglielmo Gemignani

Sapienza Università di Roma

1

## Hierarchical Task Networks HTN

In real problems, search at the level of primitive actions can be very inefficient.

Often, the actions make very small changes wrt the goal.

- Some steps of the plan may remain abstract until execution.

Examples:
- planning a holiday,
- a mobile robot cleaning the table in a home after dinner.

2

## HTN principles

Basic Idea: create a hierarchical decomposition such that:

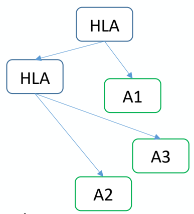
- The operators of classical planning are preserved and and a more abstract concept of action is introduced:

  High Level Actions (HLA)

- Planning proceeds by decomposing nonprimitive tasks recursively into smaller subtasks, until primitive tasks are obtained.

3

## HTN Action Descriptions




Two types of actions:

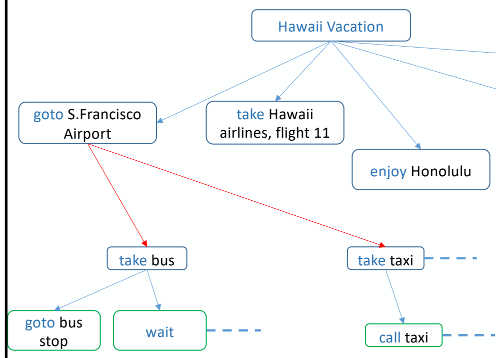
- Primitive actions A that (as in classical planning) can be executed by the agent
- High-Level Actions HLA that cannot be executed by the agent, but have one or more refinements into *sequences of actions.*
  (HLA and their refinements embody a plan)

These sequences of actions can contain both high-level and primitive actions

4

## HTN Example

The solution is to create a hierarchical decomposition




Notice:
red arrows represent alternative refinements (i.e. different methods to implement the HLA)
blu arrows represent sequences of HLA within a single refinement

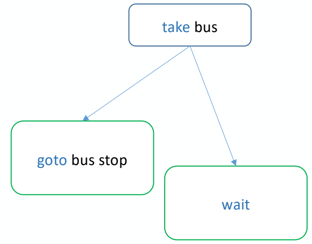
5

5

## High Level Actions (HLA)

HLA refinements containing only primitives are called implementations of the HLA.

An implementation of an High level plan HLP is the concatenation of the implementations of each HLA in the sequence.




6

## Reaching Goals

Given the precondition-effect definitions of each primitive, one can determine whether any given implementation of a HLP achieves the goal!

*An High-level plan achieves the goal from a given state if* ***at least one*** *of its implementations does.*

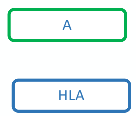
7

## Search for plans

**Base case**:
If an HLA has exactly one implementation, then we can compute the preconditions-effects of the HLA exactly as if it were a primitive action.

**What If an HLA has multiple different implementations?**

- Search among the implementations
- Search for abstract plans
  (Reasoning on HLAs)




8

## Searching for primitive solutions (one implementation for HLA)

A

The hierarchical search refines HLA from the initial abstract specification to a sequence of primitives to determine whether a plan is workable.

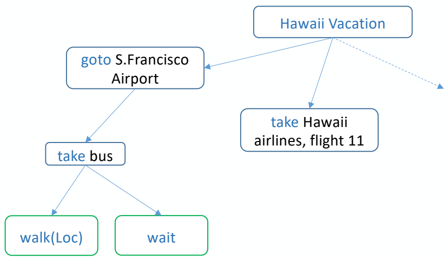
- Repeatedly choose an HLA in the current plan and replace it with one of its refinements until goal is reached.
- The goal check can be performed when the plan is refined to a sequence of primitive actions.

- Complexity b^d -> r^(d-1/k-1),
  b=number of actions, d=length of the solution,
  r = number of refinements, k = length of refinements

9

## Searching for primitive solutions

A




10

## HPlanning algorithm

A breadth-first implementation hierarchical forward planning search

```
function HIERARCHICAL-SEARCH(problem, hierarchy) returns a solution, or failure
  frontier ← a FIFO queue with [Act] as the only element
  loop do
    if EMPTY?(frontier) then return failure
    plan ← POP(frontier)   /* chooses the shallowest plan in frontier */
    hla ← the first HLA in plan, or null if none
    prefix,suffix ← the action subsequences before and after hla in plan
    outcome ← RESULT(problem.INITIAL-STATE, prefix)
    if hla is null then   /* so plan is primitive and outcome is its result */
       if outcome satisfies problem.GOAL then return plan
    else for each sequence in REFINEMENTS(hla, outcome, hierarchy) do
       frontier ← INSERT(APPEND(prefix, sequence, suffix), frontier)
```

11

## Searching for abstract plans

HLA

To really take advantage of the power of hierarchical decomposition we need to guarantee that:

abstract plans achieve the goal.

i.e.: The planner should be able to determine that, eventually, the HLAs **[Drive(Home, Airport), Fly(Airport, Honolulu)]** get the agent to the goal without looking for their refinements.

NOTE: If an High-level plan achieves the goal, working in a small search space of HLAs, then we can let the planner refine each step of the plan.

12

## Downward refinement

HLA

=> Condition for success: a HLP that achieves the goal, if **at least one of its implementation does**.

**Downward refinement property for HLAs**

Approach: define precondition-effect descriptions of the HLAs as in the case of primitives.

How can we model the effects of a high-level action with multiple refinements?

13

## Conservative approach

HLA

Include **ONLY** the positive effects that are achieved by every implementation of the HLA and the negative effects of any implementation.

Ex: Without Cash, can't get to the airport …

- Demonic non-determinism
- Angelic non-determinism

14

## Reachable Sets

HLA

**REACH(s,h)** of an HLA h from state s, is the set of states reachable by any of the HLA's implementations.

The reachable set of a sequence of HLAs is the union of all the reachable sets obtained by applying the n-th HLA in each state in the reachable set of the (n-1)-th HLA, i.e.

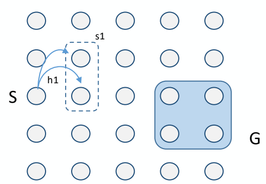
$$REACH(s_n, [h_0, \dots, h_n]) = \bigcup_{\substack{i=1 \\ s_i \in REACH(s_{i-1}, [h_{i-1}, \dots, h_0])}}^{n} REACH(s_i, [h_i, \dots, h_0])$$

15

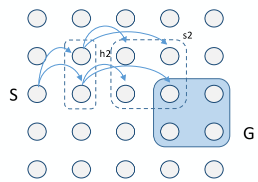
## Reachable sets: example

HLA

Given a sequence of HLAs, it achieves the goal if its reachable set intersects the set of possible goal states.
**NB**: If it does not intersect, then there is no plan.




REACH(s1,h1)




REACH(s2,h2)

16

## Searching for abstract solutions

HLA

Does the HLA reach the goal?

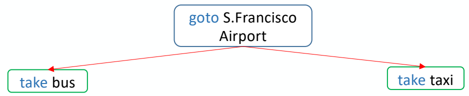
1. Search among high-level plans;

2. If one reachable set intersects the goal:
   Commit to that abstract plan
   (knowing that it works)
   return yes;

3. Else refining the plan further.
   if refinement returns yes, then return yes;
   else return no;

17

## Preconditions and effects of HLA




Action (bus(Home,Airport),
Prec: at(Home)
Eff: at(Airport) Λ state(Tired))

Action (taxi(Home,Airport),
Prec: at(Home)
Eff: at(Airport) Λ -have(Cash))

Action (go(Home,Airport),
Prec: at(Home)
Eff: at(Airport) Λ state(Tired) Λ -have(Cash) )

Both will never be true

18

## Representing preconditions and effects

HLA

Primitive actions can add or delete variables or leave them unchanged.
An HLA can do more, it can control the value of a variable.

+ the add symbol
- the delete symbol
± the add-or-delete symbol
~ the possibly symbol

+~ at(airport) means that possibly adds at(airport)

Action (bus(Home,Airport),
Prec: at(Home)
Eff: at(Airport) Λ state(Tired))

Action (taxi(Home,Airport),
Prec: at(Home)
Eff: at(Airport) Λ -have(Cash))

Action (go(Home,Airport),
Prec: at(Home)
Eff: at(Airport) Λ -~have(Cash) Λ +~ state(Tired) )

19

## Approximated representation

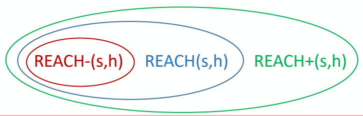
With infinitely many refinements possible, effects should be be **approximated**

Action (go(Home,Airport),
Prec: at(Home)
Eff: at(Airport), -~ have(Cash) Λ +~ state(Tired) )

The REACH(s,h) needs to be redefined with approximate effects:

- Optimistic description, overstate the reachable set REACH+(s,h)
  Possible effects as definite!

- Pessimistic description, understate the reachable set REACH-(s,h)
  Possible effects do not happen!

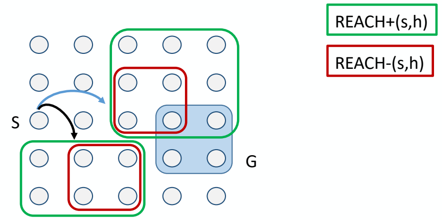



20

## Plan existence

If the optimistic reachable set does not intersect the goal
=> the plan does not work

If the pessimistic reachable set intersects the goal
=> the plan works

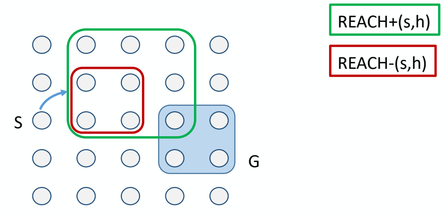



21

## Not always possible to determine existence!

If the optimistic reachable set does intersect the goal
=> the plan might work

If the pessimistic reachable set does not intersect the goal
=> the plan might work




22

## Hplanning for HLPlans

Basis hierarchical refinement search + approximate effects.

**function** ANGELIC-SEARCH(*problem, HTN, initialPlan*) **returns** a solution, or failure
  *frontier* ← a FIFO queue with *InitialPlan* as the only element
  **loop do**
    **if** EMPTY?(*frontier*) **then return** failure
    *plan* ← POP(*frontier*) /* chooses the shallowest node in *frontier* */
    **if** $\text{REACH}^+$ (*problem*.INITIAL-STATE,*plan*) intersects *problem*.GOAL **then**
      **if** *plan* is primitive **then return** *plan*
      *guaranteed* ← $\text{REACH}^-$(*problem*.INITIAL-STATE,*plan*) ∩ *problem*.GOAL
      **if** *guaranteed* ≠ { } and MAKING-PROGRESS(*plan, initialPlan*) **then**
        *finalState* ← any element of *guaranteed*
        **return** DECOMPOSE(*HTN, problem*.INITIAL-STATE, *plan, finalState*)
      *hla* ← some HLA in *plan*
      *prefix, suffix* ← the action subsequences before and after *hla* in *plan*
      **for each** *sequence* **in** REFINEMENTS(*hla, outcome, HTN*) **do**
        *frontier* ← INSERT(APPEND(*prefix, sequence, suffix*),*frontier*)

23

## HLP Decomposition

**function** DECOMPOSE($HTN, s_0, plan, s_f$) **returns** a solution
  *solution* ← an empty plan
  **while** *plan* is not empty **do**
    *action* ← REMOVE-LAST(*plan*)
    $s_i$ ← a state in $\text{REACH}^-(s_0, plan)$ such that $s_f \in \text{REACH}^-(s_i, action)$
    *problem* ← a problem with INITIAL-STATE=$s_i$ and GOAL=$s_f$
    *solution* ← APPEND(ANGELIC-SEARCH(*problem, HTN, action*),*solution*)
    $s_f \leftarrow s_i$
  **return** *solution*

MAKING-PROGRESS checks that the planner is not in an infinite loop recursively applying the same refinement.

24

## Visit Grandpa

Primitives:

Action (takeBus(From, To, Money),
Prec: has(Money) Λ at(From)
Eff: -has(Money) Λ -at(From) Λ at(To))

Action (walk(From, To),
Prec: at(From)
Eff: -at(From) Λ at(To))

Action (talk(),
Eff: state(happy))

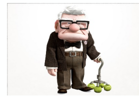
Action (getIcecream(Money),
Prec: has(Money)
Eff: state(happy)) Λ -has(Money) )

Initial state: has(money1) Λ has(money2) Λ at(home)

Goal state: state(happy)

25

## Visit Grandpa



HLAs:

HLA (goto(From,To,Money),
Prec: at(From)
Eff: -at(From) Λ at(To) Λ -~has(money)
Refinement: [takeBus(),walk()])

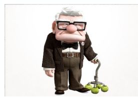
HLA(spendtime(Money),
Prec: at(granpa)
Eff: state(happy) Λ -~has(Money)
Refinement: [getIcecream(),talk()])

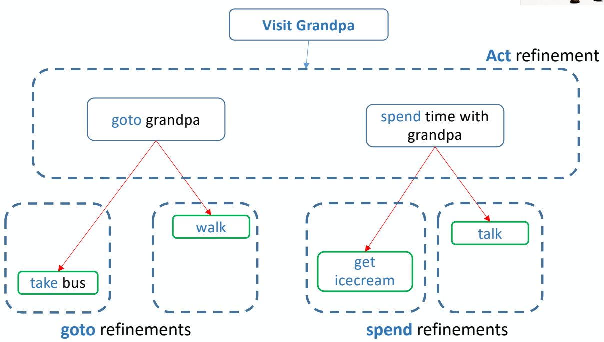
Initial state: has(money1) Λ has(money2) Λ at(home)

Goal state: state(happy)

26

## Visit Grandpa





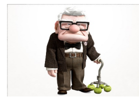

27

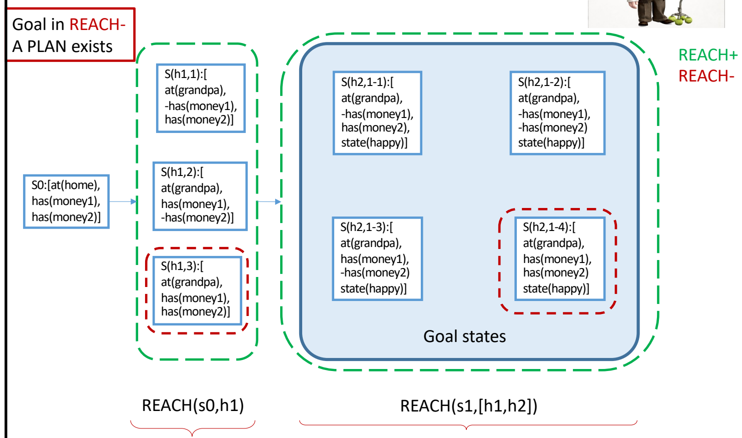
27

## Visit Grandpa

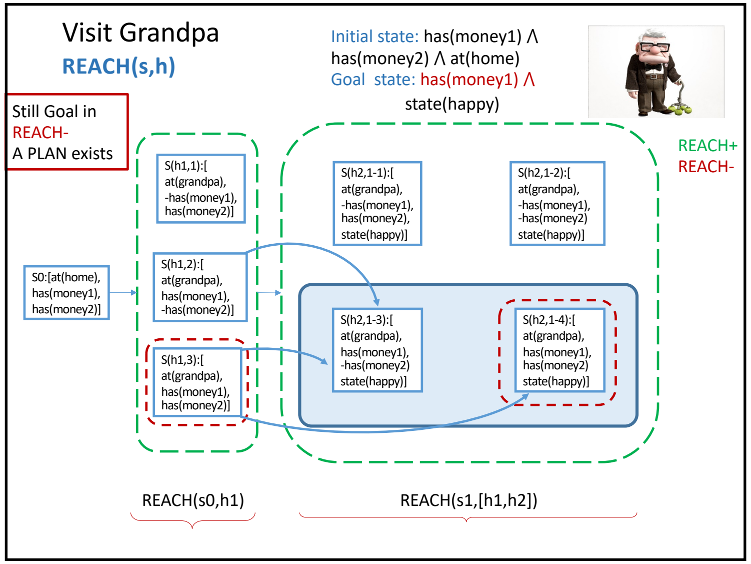
**REACH(s,h) ±**

Initial state: has(money1) Λ has(money2) Λ at(home)
Goal state: state(happy)

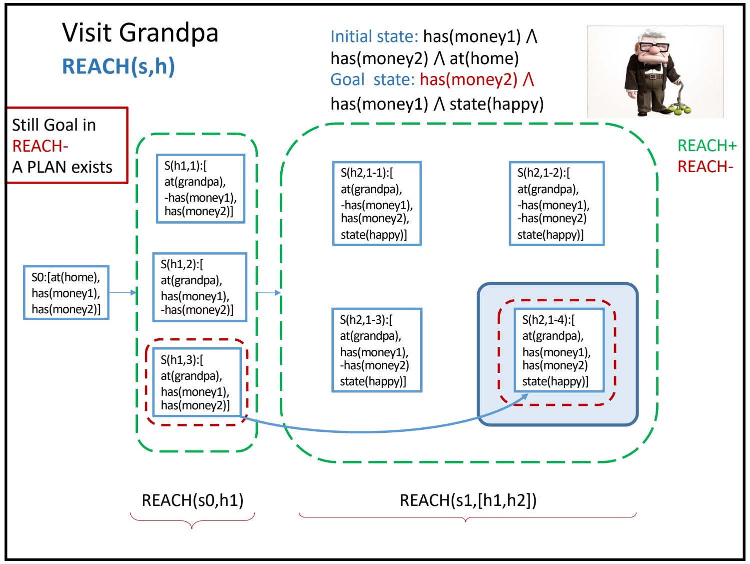




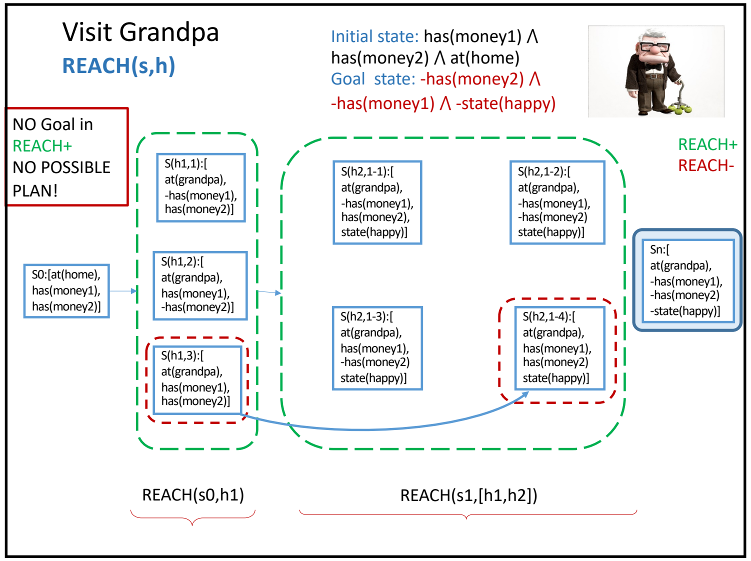

28




29




30




31

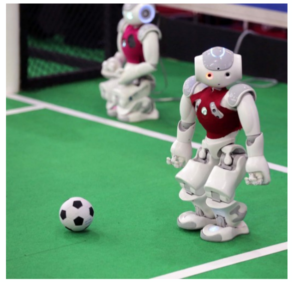
## Soccer Striker

Primitives:

| | |
|---|---|
| turn(left); | turn(right); |
| turnAround(left); | turnAround(right); |
| moveForward(); | kick(); |

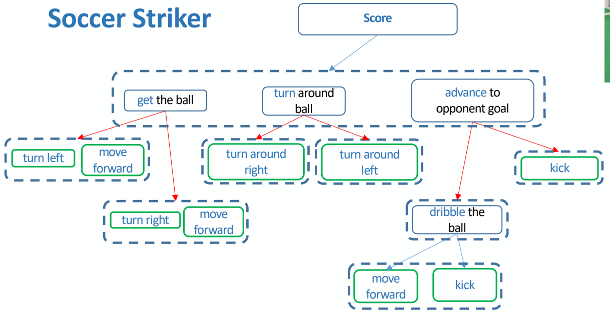
HLAs:

| | |
|---|---|
| get the ball; | turn to goal; |
| advance; | dribble; |



32

## Soccer Striker




35

33

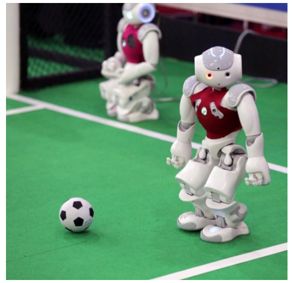
## Soccer Striker

Primitives:

| | |
|---|---|
| turn(left); | turn(right); |
| turnAround(left); | turnAround(right); |
| moveForward(); | kick(); |
| backKick(); | |

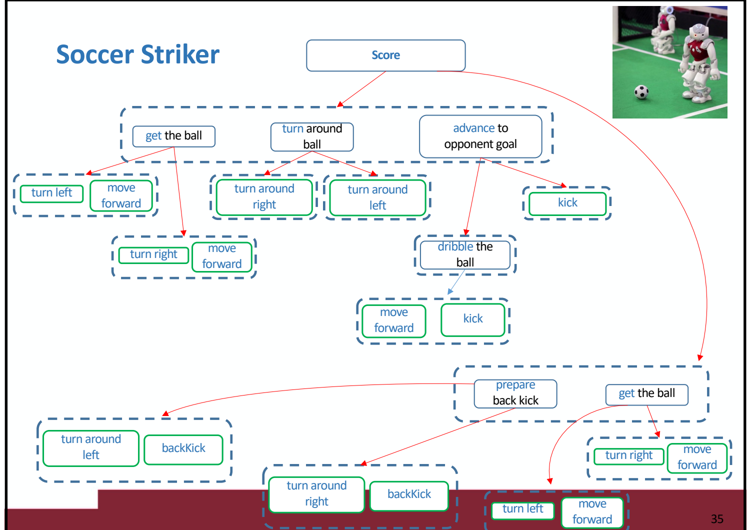
HLAs:

| | |
|---|---|
| get the ball; | turn to goal; |
| advance; | dribble; |
| prepare back kick; | |



34

## Soccer Striker




35

35

## Recursive HTN Plans: example

Write the HLA ClearStack(Block) that takes all the blocks from the stack and puts them on the table.
Use it to create a plan for this problem:

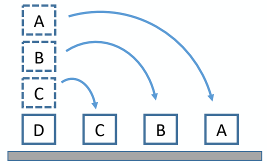
HLA

Refinement (ClearStack(Block),
PREC: on(Block, table )
STEPS: [] )

Refinement (ClearStack(Block),
PREC: -clear(Block) ∧ on(BlockAbove, Block )
STEPS: [ClearStack(BlockAbove), putOnTable(Block)] )

Init: Clear(A) ∧ On(A,B)∧ On(B,C) ∧ On(C,D) ∧ On(D,Table)

Goal: On(A,Table)∧ On(B,Table) ∧ On(C,Table) ∧ On(D,Table)




Action (putOnTable(Block),
Prec: clear(Block) ∧ on(Block, BlockBelow )
Eff: clear(BlockBelow) ∧ on(Block,table) ∧ -on(Block, BlockBelow))

A

36