Real World Planning and Acting

CS4246/CS5446

AI Planning and Decision Making

Sem 1, AY2021-22

Topics

- Planning as Search
  - Soundness and completeness
  - Complexity of planning (RN 11, Bibliographical and Historical Notes)
- Heuristics for planning (RN 11.3)
  - Problem relaxation and admissible heuristics
  - Domain independent planning (RN 11.3.1)
  - State abstraction in planning (RN 11..3.2)
- Hierarchical planning (RN 11.4)
  - Hierarchical task networks and HTN Planning
  - High-level actions (RN 11.4.1)
  - Searching for primitive solutions (RN 11.4.2)
  - Searching for abstract solutions (RN 11.4.3)

Solving Planning Problems

Heuristics: for classical planning
HTN: New planning model

- Planning Problem or Model
  - Appropriate abstraction of states, actions, effects, and goals (and costs and values)
- Planning Algorithm
  - Input: a problem
  - Output: a solution in the form of an action sequence
- Planning Solution
  - A plan or path from the initial state(s) to the goal state(s)
    - Any path; OR
    - An optimal path wrt to costs or values
  - A goal state that satisfies certain properties

Planning as Search

Properties and complexity analysis

Recall: Planning as Search

Source: Dana Nau: Lecture slides for *Automated Planning*

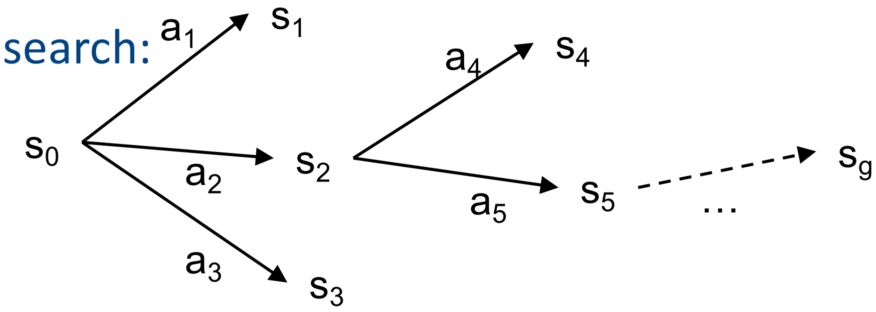
- Nearly all planning procedures are search procedures
  - Different planning procedures have different search spaces
- State-space planning
  - Each node represents a state of the world
  - A plan is a path through the space
- Forward Search
  - Prone to exploring irrelevant actions
  - Need to traverse large state-spaces
  - Average branching factor is huge!
  - Need domain-specific or domain-independent heuristics that can be derived automatically
- Backward search
  - Keep branching factor lower than forward search
  - Using state sets for searching makes it harder to come up with heuristics

Planning Algorithm Properties

- A planning algorithm or planner is sound
  - For any plan generated by the planning algorithm, this plan is guaranteed to be a solution
- A planning algorithm or planner is complete
  - If a solution exists then the planning algorithm will return a solution
- Question: How "good" are the solutions?
  - For any plan generated by the planning algorithm, can the plan be guaranteed to be an optimal solution?

Deterministic Forward Search

- Some deterministic implementations of forward search:
  - breadth-first search
  - depth-first search
  - best-first search (e.g., A*)
  - greedy search



- Breadth-first and best-first search are sound and complete
  - Worst-case memory requirement is exponential in the length of the solution
- In practice, more likely to use depth-first search (DFS) or greedy search
  - Worst-case memory requirement is linear in the length of the solution
  - In general, sound but not complete
  - But classical planning has finite no. of states, DFS can be made complete by doing loop-checking

Source: Dana Nau: Lecture slides for *Automated Planning*

Complexity of Planning

- Two decision problems:
  - PlanSAT –whether there exists any plan that solves a planning problem
  - Bounded PlanSAT –whether there is a solution of length k or less
    - For classical planning, both problems are decidable as number of states is finite
- In general:
  - Both problems are in PSPACE: solvable with polynomial amount of space
  - In many domains, Bounded PlanSAT is NP-complete; PlanSAT is in P
    - Optimal planning is usually hard, but sub-optimal planning is sometimes easy
- In real world planning:
  - Agents usually asked to find plans in specific domains, not in worst-case instances
  - To do well, need good heuristics or alternate planning models

Heuristics for Planning

Improving Efficiency

Heuristics for Planning

- Heuristic function
  - $h(s)$ estimates distance from a state s to the goal g
- Main idea
  - Find admissible heuristic for distance – one that does not overestimate
  - A* or other heuristic search can then find optimal solutions
- Approach
  - Define a relaxed problem that is easier to solve
  - Exact cost of solution to easier problem is heuristic for original problem

Types of Heuristics

- Definition and application
  - Facilitated by factored representation for states and action schemas
- Search problem
  - A graph with states as nodes and actions as edges
  - Find a path connecting initial state to goal state
- Domain independent heuristics –Why?
  - Two ways to relax the problem (make it strictly easier):
    - Add more **edges** – easier to find path
    - Group multiple **nodes** together – abstract with fewer states and easier to search
  - Prune away irrelevant branches of search tree

Heuristics of Adding Edges

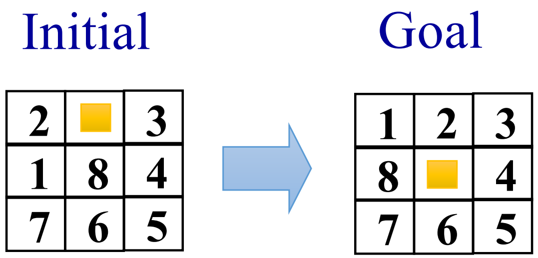
- Ignore preconditions heuristic
  - Drop all preconditions from actions
- Ignore selected preconditions heuristic
  - Derive simpler measures of distance from goal
- Ignore delete list heuristic
  - Allow monotonic progress toward goal
- Caveats:
  - Finding solution to relaxed problem is still NP-hard
  - Trade-off in optimality or admissibility sometimes required
  - Expensive to calculate heuristics
  - Do not reduce state-space size

How to Define an Edge-Based Heuristic?

- Approach:
  - Relax actions by removing all preconditions and all effects except goal literals
  - Count minimum no. of actions required for union effects to satisfy the goal
- Note:
  - An instance of the set cover problem – NP-hard
- Potential solution:
  - Simple greedy algorithm guaranteed to return a set covering whose size is within a factor of $\log(n)$ of the true minimum covering
    - where n is the number of literals in the goal
  - Loses guarantee of admissibility

Example: 8-Puzzle as Planning

- Planning as path search



- Game objective:
  - To rearrange a given initial configuration (state) of eight numbered tiles arranged on a 3×3 board into given final or goal configuration (state)

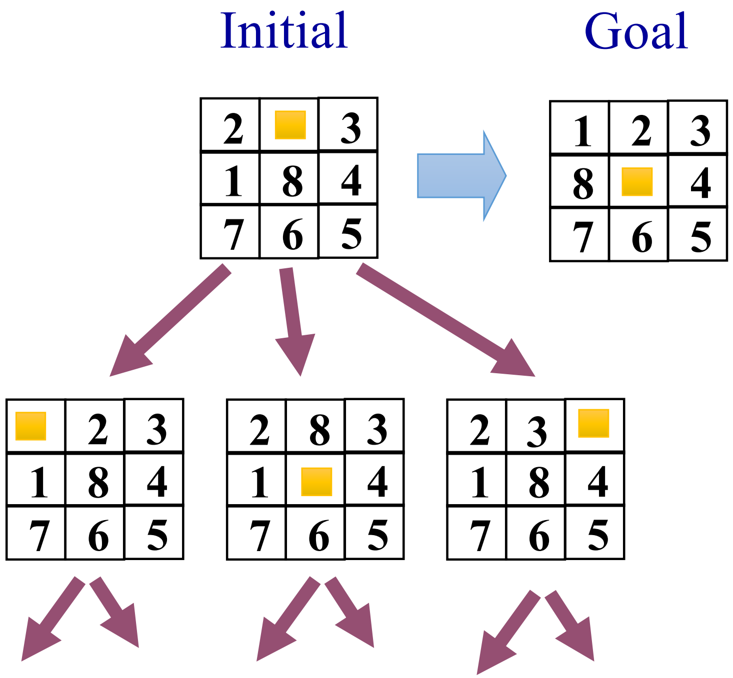
$Action(Slide(t, s_1, s_2)$

Precond: $On(t, s_1) \ \wedge \ Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$

Effect: $On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2)$

Example: 8-Puzzle as Planning

- States
  - A state description specifies the location of each of the eight tiles in one of the nine squares
- Actions or operators
  - Tile slides left, right, up, or down
- Goal test
  - State matches the goal configuration
- Path cost
  - Each step cost 1, so path cost is just the length of the path



$Action(Slide(t, s_1, s_2)$

Precond: $On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$

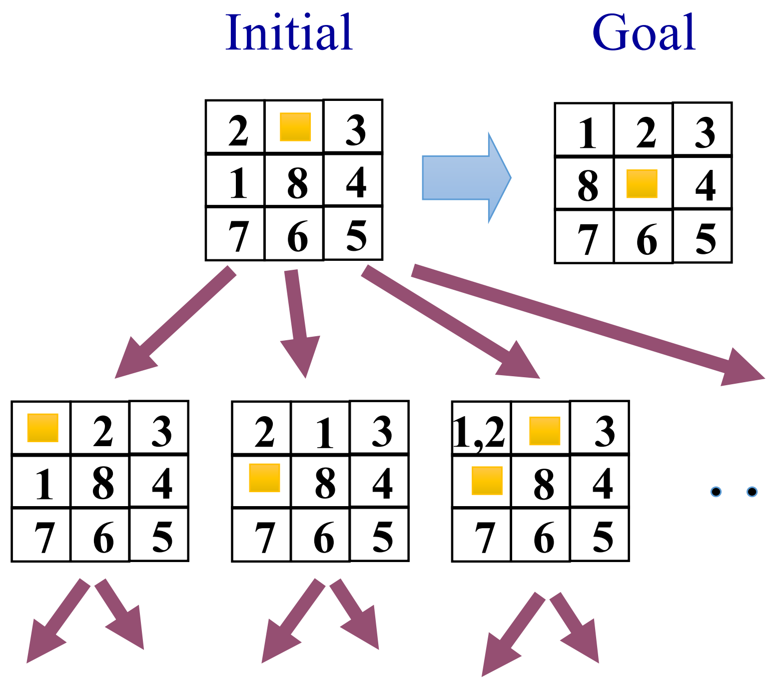
Effect: $On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2)$

Ignore Preconditions Heuristic

- Main idea:
  - All actions become applicable in all states
  - Any single goal fluent can be achieved in one step
    (if there is an applicable action)
  - No. of steps required to solve relaxed problem $\approx$ no. of unsatisfied goals
    1. Some actions may achieve multiple goals
    2. Some actions may undo the effects of others
  - Possible accurate heuristic – consider 1 and ignore 2

Ignore Selected Preconditions Heuristic

- Remove Preconditions
  - $Blank(s_2) \wedge Adjacent(s_1, s_2)$
- What does the heuristic do?
- What are the estimates?



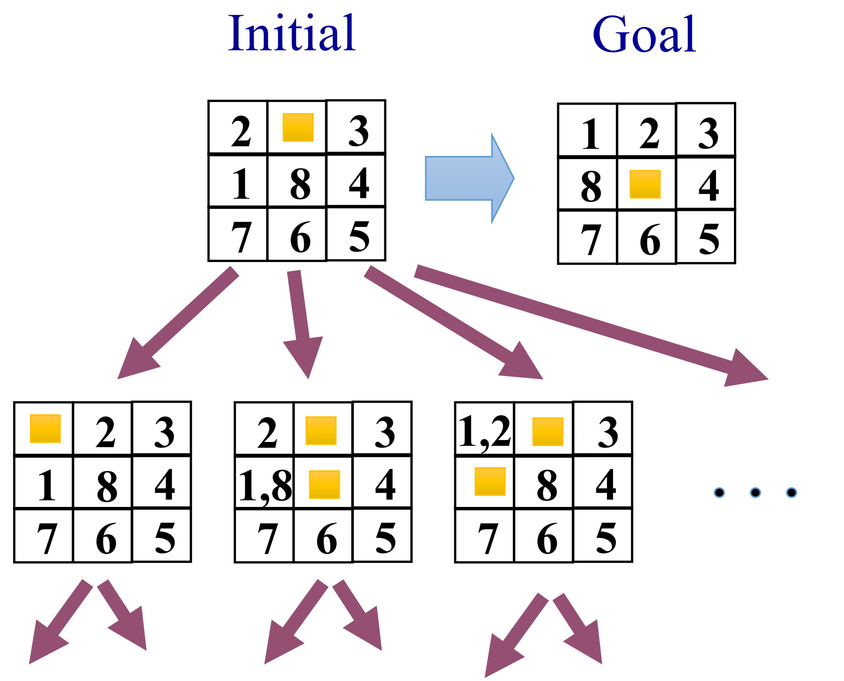
$Action(Slide(t, s_1, s_2)$
Precond: $On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$
Effect: $On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2)$

Ignore Selected Preconditions Heuristic

- Remove Preconditions
  - $Blank(s_2)$
- What does the heuristic do?

- What are the estimates?

- Caveat:
  - Unclear which preconditions can be selectively ignored in general



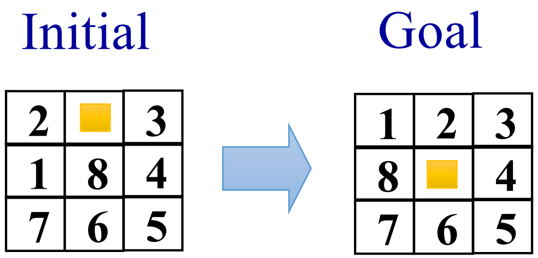
$Action(Slide(t, s_1, s_2)$
  Precond: $On(t, s_1) \land Tile(t) \land Blank(s_2) \land Adjacent(s_1, s_2)$
  Effect: $On(t, s_2) \land Blank(s_1) \land \neg On(t, s_1) \land \neg Blank(s_2)$

Ignore Delete List Heuristic

- Main idea:
  - Assume only positive literals in all plans and actions
  - Remove delete list from all actions (all negative literals from effects) to make monotonic progression toward goal
  - Create a relaxed version of original problem that is easier to solve, where solution length will serve as good heuristics
  - Approximate solution can be found in polynomial time by hill-climbing

Ignore Delete Lists Heuristic

- Ignore delete lists
  - How does the problem become easier when ignore delete lists heuristic is applied to this schema?



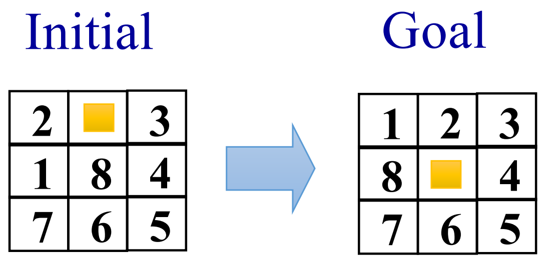
$Action(Slide(t, s_1, s_2)$

  Precond: $On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$

  Effect: $On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2)$

Can We Do Better?

- What would YOU do?
  - Can planning system emulate human moves?
  - Would that help?



$Action(Slide(t, s_1, s_2)$

Precond: $On(t, s_1) \wedge Tile(t) \wedge Blank(s_2) \wedge Adjacent(s_1, s_2)$

Effect: $On(t, s_2) \wedge Blank(s_1) \wedge \neg On(t, s_1) \wedge \neg Blank(s_2)$

Domain-independent Pruning

- Symmetry reduction
  - Prune all symmetric branches of the search tree except for one
  - For many domains, efficiently solve intractable
- Forward pruning
  - Accept risk of pruning away an optimal solution, in order to focus search on promising branches
- Rule-out negative interactions
  - A problem has serializable subgoals if there exists an order of subgoals such that the planner can achieve them in that order without having to undo any of the previously achieved subgoals

Serializable Subgoals

- Planning examples in the real world:
  - Build a tower of blocks on Table, in any order
  - Switch on all n lights with independent switches, in any order
  - Remote Agent Planner that commands NASA's Deep Space One spacecraft (1998) – serializable by design – able to command spacecraft in real-time

Heuristics of State Abstraction

- State abstraction:
  - Many-to-one mapping from states in the ground representation of the problem to the abstract representation
- Main approach:
  - Ignore some fluents
  - Solution in abstract state space will be shorter than a solution in the original space – admissible heuristic
  - Abstract solution extensible to solution for original problem

Example: Air Cargo Transportation

- Original problem:
  - 10 airports, 50 planes and 200 cargos
  - Total no. of states =
- Assumption:
  - All cargos are just in 5 airports, instead of 10, and all cargos in the same airport have the same destination.
- Reformulation:
  - Drop all the irrelevant At fluents (What are the relevant ones?)
  - Total no. of states =
- Solution:
  - Shorter than that for original problem (admissible heuristic)
  - Extensible by adding relevant actions

Decomposition

- Main idea:
  - Divide problem into parts, solving each part independently, and then combining the parts - Key idea in defining heuristics
- How to choose the right abstraction to reduce total cost?
  - Defining abstraction, doing abstract search, mapping abstraction back to original problem
  - Can the cost be less than original planning cost?
  - Example:
    - Pattern databases – cost of creation amortized over many problem instances

Planning Cost with Abstraction

- Problem definition:
  - Suppose the goal is a set of fluents G, divided into disjoint subsets $G_1, \dots, G_n$.
  - Find plans $P_1, \dots, P_n$ that solve the respective subgoals
  - What is the estimated cost of the plan for achieving all of G?
- Heuristic estimation:
  - Think of each $Cost(P_\mathrm{i})$ as a heuristic estimate
  - If each subproblem uses an admissible heuristic, taking Max is admissible
- Assuming subgoal independence:
  - Sum the cost of solving each independent subgoal; if admissible, Sum is better than Max – Why?
  - Solution optimistic when there are negative interactions between subplans for each subgoal; e.g.,action in one subplan deletes goals in another subproblem.
  - Solution pessimistic (not admissible) when there are positive interactions; e.g., actions in one subplan achieves goals in another subproblem

Summary

- Classical or deterministic planning
  - No consensus on the best approach; competition and cross-fertilization induce progress
- Using domain-independent heuristics
  - Transform planning problems into relaxed problem spaces
  - Effective heuristics derived with subgoal independence assumptions by:
    - relaxation of planning problem
    - pruning repeated or irrelevant branches
  - Solve with efficient algorithms

Example: FF- FastForward

- Characteristics:
  - Forward state-space searcher making use of effective heuristics
  - Ignore-delete-list heuristic with graph plan for heuristic estimation
  - Hill-climbing search (modified to keep track of plan) with heuristic to find a solution
    - Non-standard hill-climbing algorithm: avoids local maxima by running a breadth-first search from the current state until a better one is found
    - If this fails, FF switches to greedy best-first search instead
  - [Hoffmann, 2001]

Classical Planning Today

- Classical Planning research: Examples:
  - Families of systems in use – Heuristic Search Planner (HSP), Fast Forward (FF), Fast Downward
  - https://planning.wiki/ref/planners/
- International Planning Competitions
  - https://www.icaps-conference.org/competitions/
  - FastForward (FF) [Hoffmann, 2001] was the winner in the 2002 International Planning Competition (IPC)
  - SATPlan was the winner in the 2004 and 2006 IPC
  - IPC 2014 was won by SymBA*, based on bidirectional search using heuristics and abstraction.
  - IPC 2018 was won by Delfi, using deep learning to select a planner from a collection of planners including Fast Downward (uses forward search) with various heuristics and SymBA*
  - IPC 2020 – Hierarchical planning!
- Classical Planning in practice: Examples
  - Commercial video games [Neufeld, X., et al., 2019]
  - AI planning for enterprise [Sohrabi,S, 2019]
  - Space exploration [Estlin, T., et al. 07] [Rabideau, G., et al., 2020]

Hierarchical Planning and Acting

Manage complexity

Examples

- Example 1:

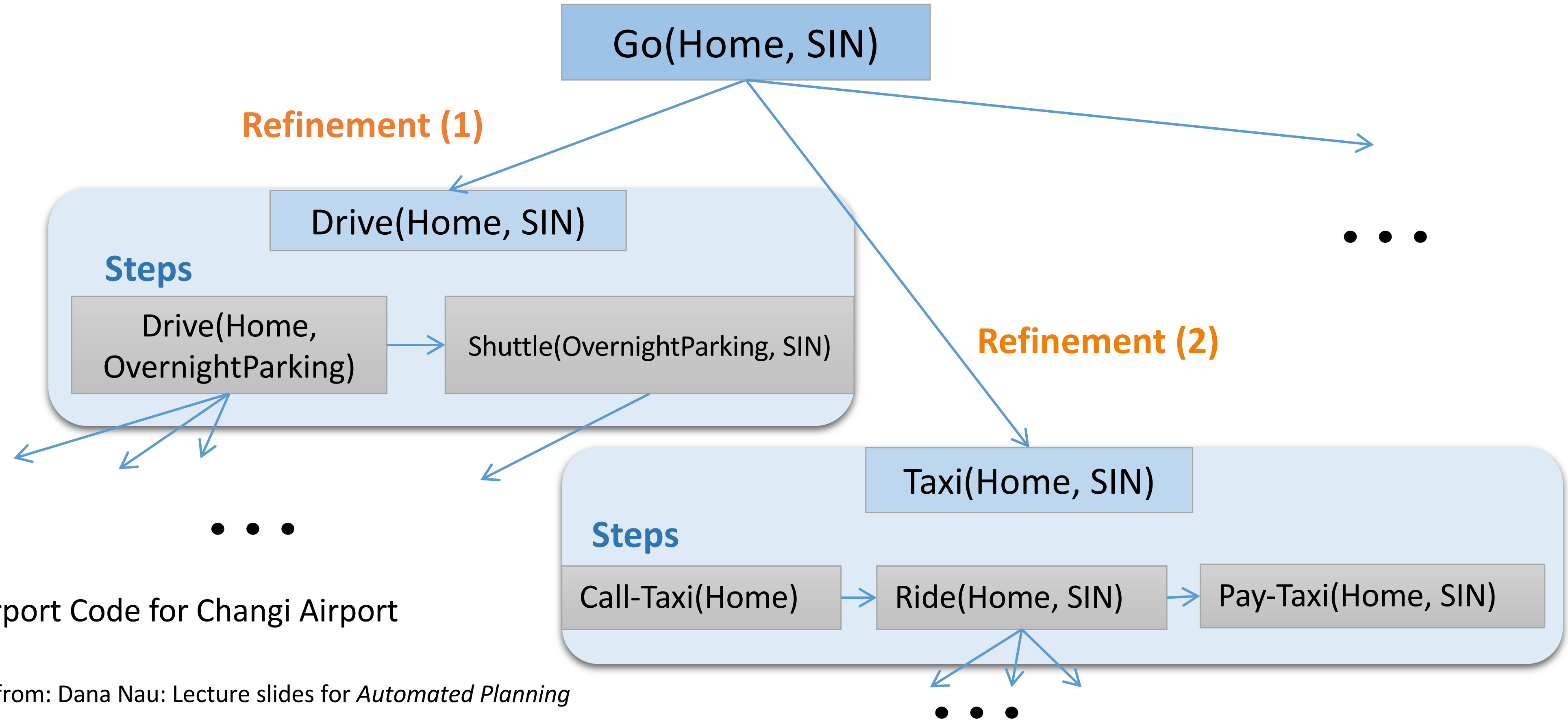
  How to go to CS4246 lecture from home?

  - Solution: Go to COM1 from home, find lecture hall
  - Solution: take MRT, change to internal bus, get off, find SR 2.
  - Solution: Switch on laptop, launch Zoom, join lecture

- Example 2:

  How to land on the Jezero Crater of Mars from X space station?

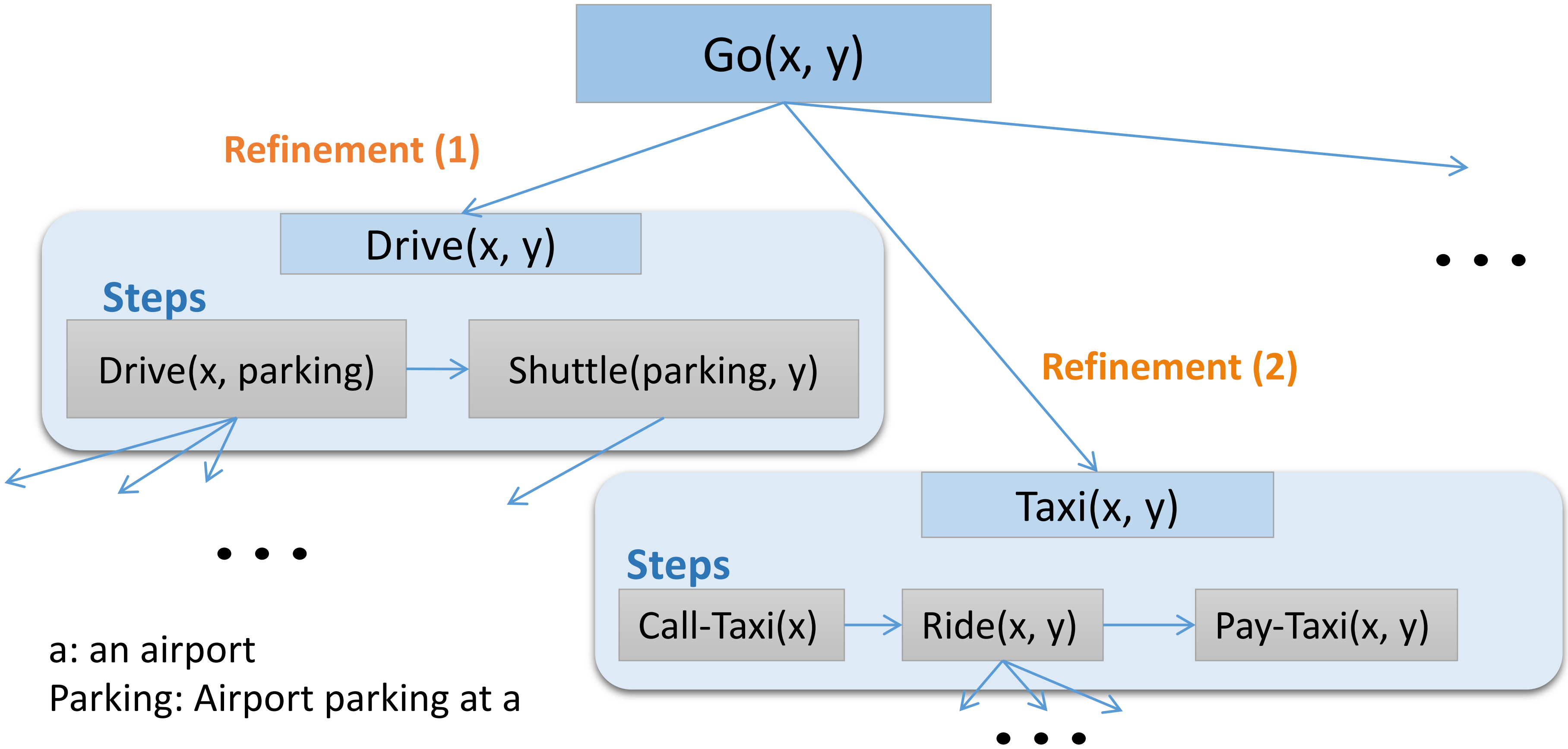
Example: Going to Changi Airport



SIN: Airport Code for Changi Airport

Adapted from: Dana Nau: Lecture slides for *Automated Planning*

Example: Going to an airport



a: an airport
Parking: Airport parking at a

Managing Complexity in Planning

- Hierarchical decomposition
  - Division of tasks into different subtasks at next level
  - At each level focus only a small number of tasks
- Deferred planning
  - Planning can occur before and during plan execution
  - Particular action can remain at an abstract level prior to the execution phase

Hierarchical Decomposition

- Key benefits
  - At each level of hierarchy, a task is reduced to a small number of subtasks or activities at the next lower level
  - Computational cost of finding the correct way to arrange activities for current problem is small
- Examples
  - Software components, subroutines
  - Military, government, and corporations

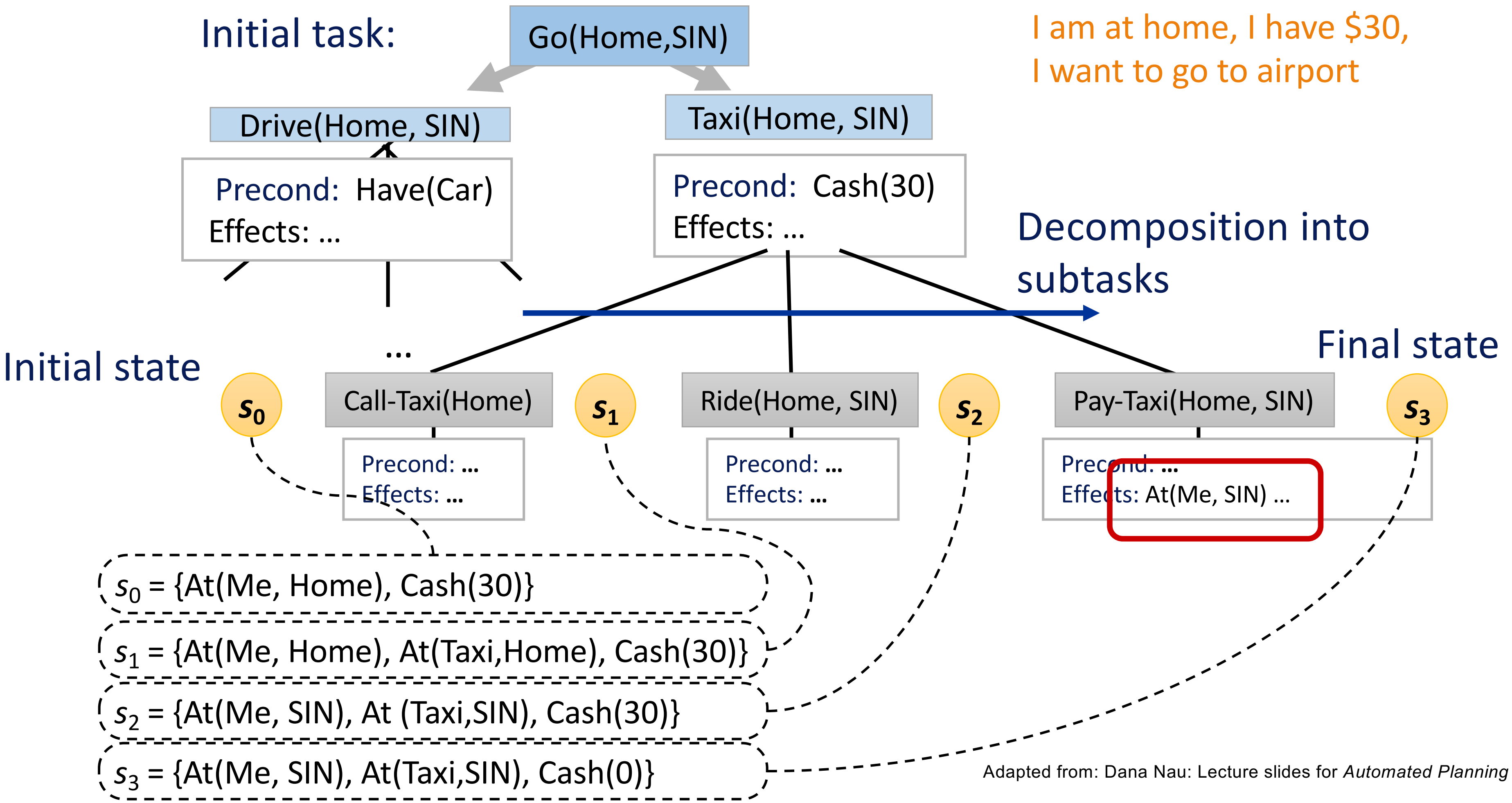
Hierarchical Task Networks

- Hierarchical task networks (HTNs)
  - A formalism to help understand hierarchical decomposition
  - A planning model that manages complexity through task abstractions
- Key concept
  - High-level actions (HLAs)
- Assumptions
  - Full observability
  - Deterministic
  - Availability of primitive actions with standard precondition-effect schemas
  - Main ideas are general in problem solving and planning and decision making

HTN Planning

- Planning Problem or Model
  - HLAs, action schemas, initial state, task list
- Planning Algorithm – How?
  - Input: a problem
  - Output: a solution in the form of an action sequence
- Planning Solution
  - Any executable plan generated by recursively applying:
    - HLA to nonprimitive tasks
    - Actions to primitive tasks
  - A goal state that satisfies certain properties

Example: Going to Changi Airport



Adapted from: Dana Nau: Lecture slides for *Automated Planning*

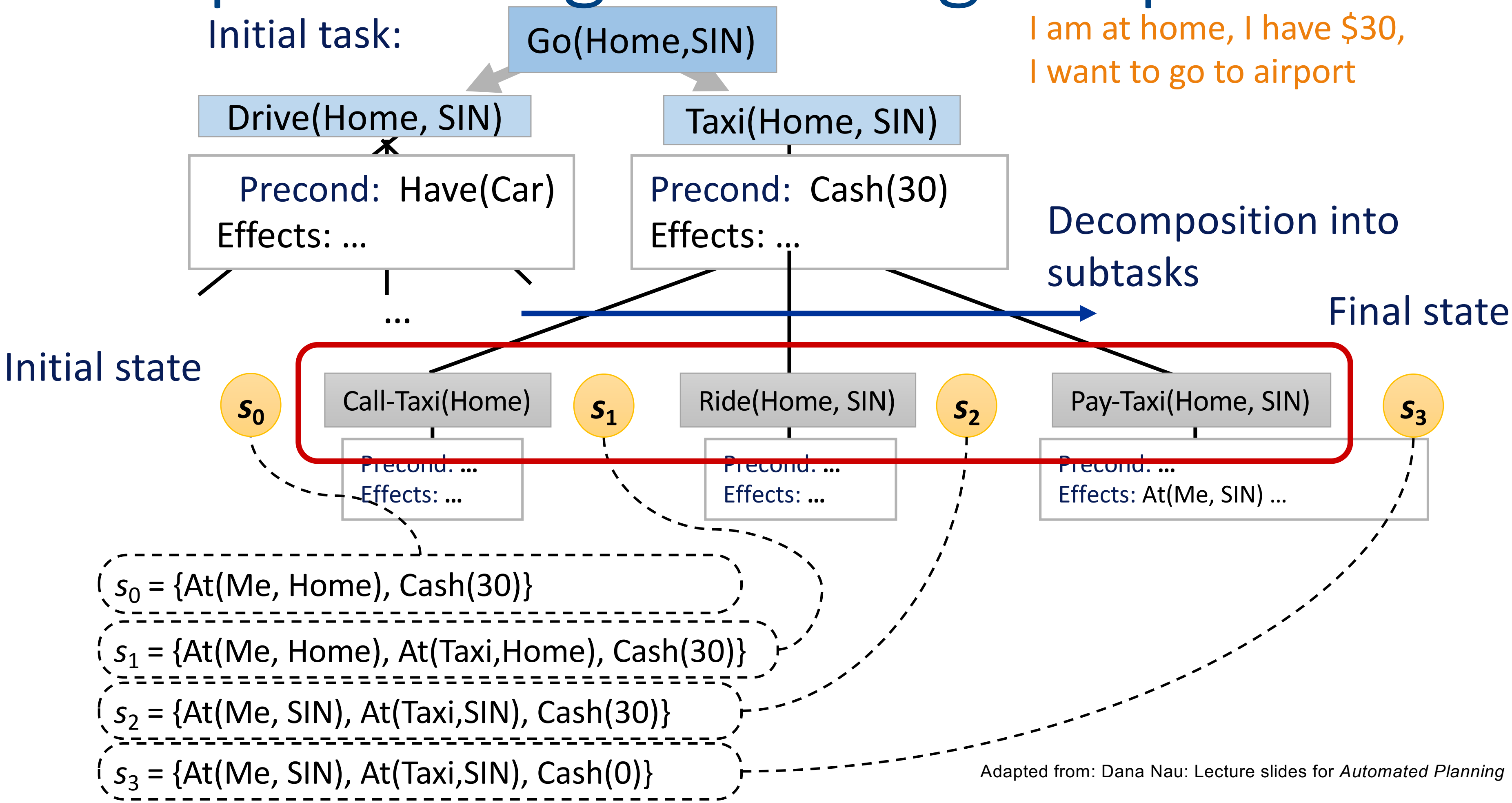
High-Level Actions (HLAs)

- Definition
  - Each HLA has one or more refinements into a sequence of actions
  - Each (refined) action can be an HLA or a primitive action
  - Recursive refinement may be needed
- Meaning
  - HLAs and their refinements embody knowledge about how to do things
    e.g., Go(Home, SIN) – drive or take a taxi

Implementation

- HLA implementation
  - An HLA refinement that contains only primitive actions
- High-level plan implementation
  - High-level plan – a sequence of HLAs
  - Concatenate implementations of each HLA in the sequence
- Observation
  - Given the precondition-effect definitions of each primitive action, can directly determine whether any given implementation of a high-level plan achieves the goal.

Example: Going to Changi Airport



Adapted from: Dana Nau: Lecture slides for *Automated Planning*

Planning with HLAs

- Definition
  - Achieves the goal from a given state if at least one of its implementations achieves the goal from that state
- Note
  - Not all implementations need to achieve the goal
  - The agent decides which implementation to execute
- Question:
  - How is this different from nondeterministic planning?

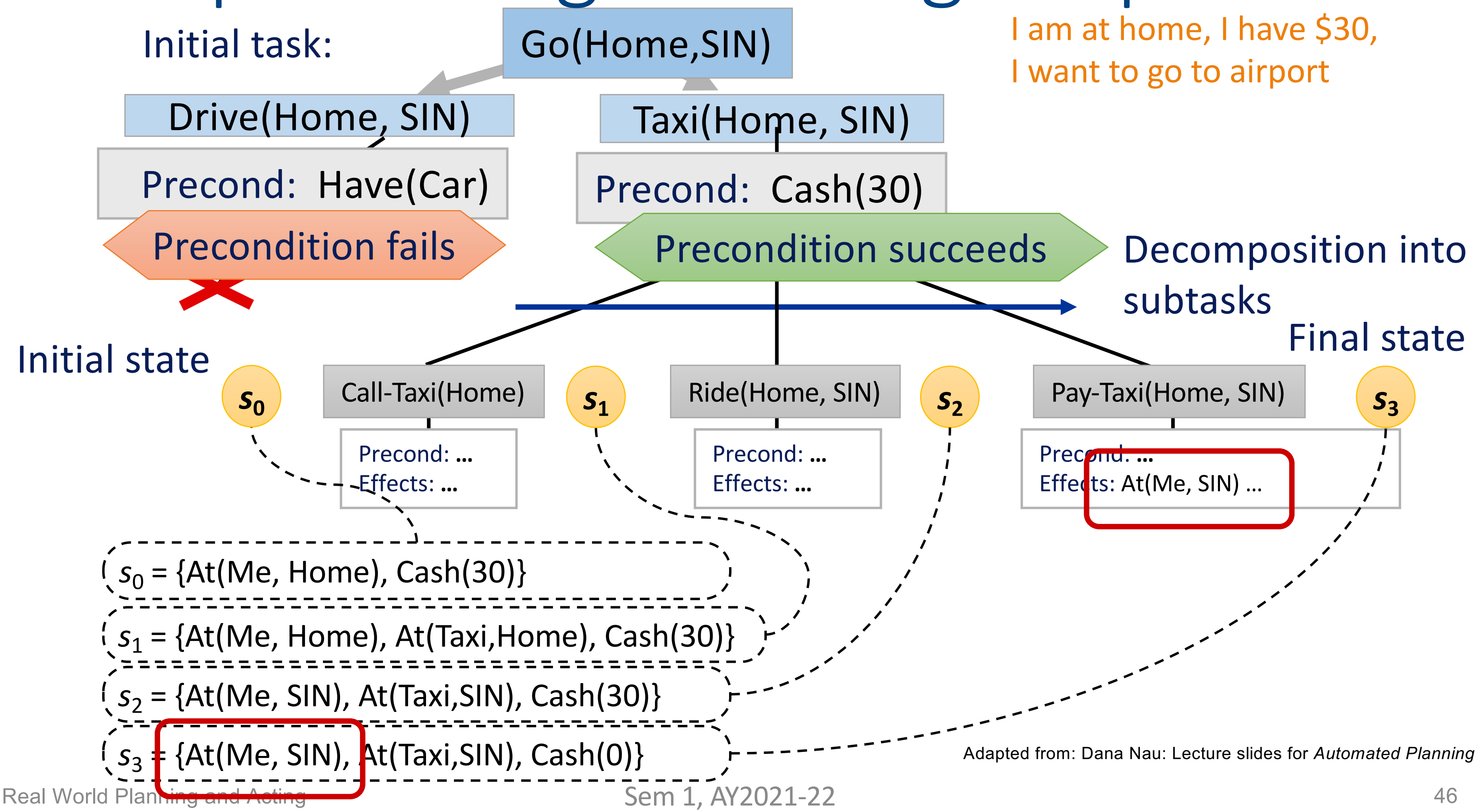
Planning with HLAs

- With one HLA implementation
  - Compute preconditions and effects of HLA from those of the implementation
  - Treat HLA exactly as if it were a primitive action
- Observation
  - Right collection of HLAs can reduce time complexity of (blind) search from exponential to linear in solution depth
  - Devising an appropriate collection of HLAs is HARD!
- With multiple HLA implementations
  - Search among implementations for one that works; OR
  - Reason directly about the HLAs - enables derivation of provably correct abstract plans, without having to consider their implementations

Hierarchical Planning as Search

Searching for Primitive Solutions

Example: Going to Changi Airport



Adapted from: Dana Nau: Lecture slides for *Automated Planning*

Searching for Primitive Solutions

- HTN Planning
  - Start with top level action Act
  - Find an implementation of Act that achieves the goal
- Hierarchical planning algorithm
  - Repeatedly choose an HLA in current plan and replace with refinement
  - Until the plan achieves the goal
- Example:
  - Breadth-first search tree
  - Plans are considered in order of depth of nesting of the refinements, rather than number of primitive steps
  - Can use graph-search, depth-first, and iterative deepening

Generic Planning Framework

- Classical planning definition:
  - For each primitive action ai:
  - Provide one refinement of Act with steps $-$ $[ai, Act]$
  - Create recursive definition of Act to add actions
  - Final refinement:
    - steps – empty, precondition – goal, effect – null
- Algorithm:
  - Repeatedly choose an HLA in the current plan
  - Replace it with one of its refinements
  - Until the plan achieves the goal

Hierarchical Search

```
function HIERARCHICAL-SEARCH(problem, hierarchy) returns a solution or failure
  frontier ← a FIFO queue with [Act] as the only element
  while true do
     if IS-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, prefix)
     if hla is null then          // so plan is primitive and outcome is its result
        if problem.IS-GOAL(outcome)  then return plan
     else for each sequence in REFINEMENTS(hla, outcome, hierarchy) do
        add APPEND(prefix, sequence, suffix) to frontier
```

Source: RN Figure 11.8

Hierarchical Search

- Main idea:
  - Explore space of sequences that conform to knowledge in the HLA library about how things are to be done
  - Knowledge encoded in action sequences in each refinement and in the preconditions of the refinements
- Practical impact:
  - Can generate huge plans with little search
    - e.g., O-PLAN to develop production plans for HITACHI (Bell and Tate 1995)
  - Hierarchically structured – easier for human to understand

Complexity Analysis

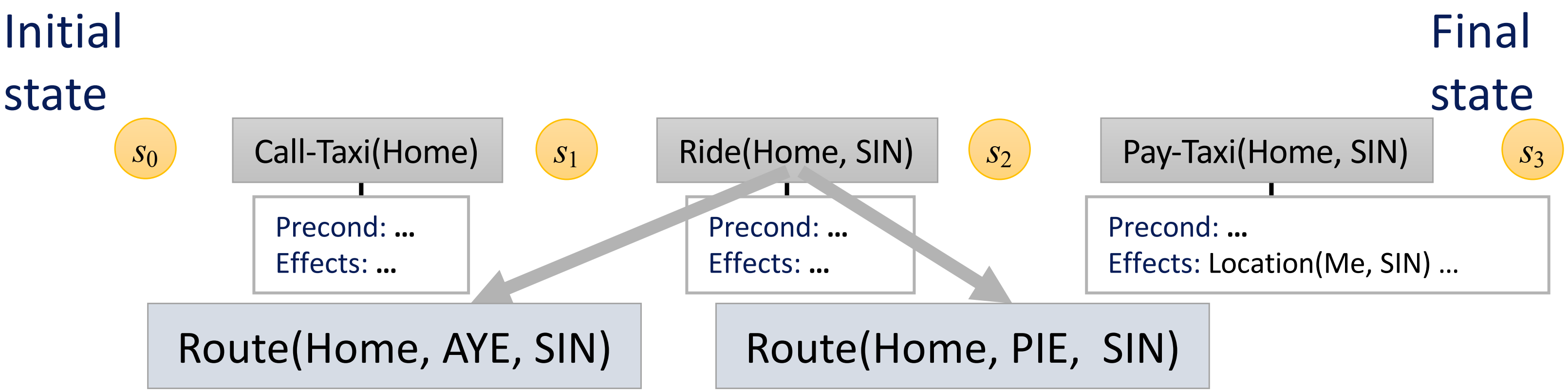
- Assumption
  - A planning problem has a solution with d primitive actions.
- For non-hierarchical, forward state-space planner
  - With b allowable actions at each state, cost is $O(b^d)$
- For HTN planner
  - Suppose each nonprimitive action has r possible refinements, each into k actions at the next lower level
  - So $r^{(d-1)(k-1)}$ possible regular decomposition trees could be constructed (see details in RN 11.4.2)
- Observation
  - Small r and large k - library of HLAs with small number of refinements each yielding a long action sequence - May be hard to construct!

Proving Plan Properties

Searching for Abstract Solutions

Motivation

- Example:
  - We should determine if a high-level plan can get one to the airport, without going through all the specific details like precise route or alighting terminal
    [Call-Taxi(Home), Ride(Home, SIN), Pay-Taxi(SIN)]



Searching for Abstract Solutions

- Approach
  - Write precondition-effect description of the HLAs
  - Prove that the high-level plan achieves the goal
  - Work in small search space of high-level actions
  - Refine committed plan to achieve exponential reduction

Searching for Abstract Solutions

- Downward refinement property (of HLA descriptions)
  - Through description of the steps:
  - Every high-level plan that “claims” to achieve the goal achieves the goal
  - At least one implementation achieves the goal
- Main challenges
  - How to write HLAs with downward refinement property?
  - How to write HLAs with multiple implementations?
  - How to describe effects of an action that can be implemented in many different ways?
- Key idea
  - Determine if reachable sets of a sequence of HLAs in the plan overlap with goals

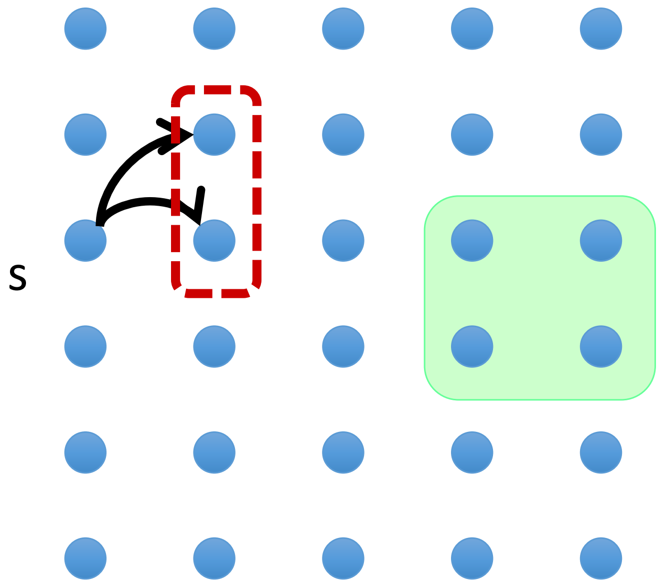
Reachable Set

- Reachable set of an HLA
  - Given a state s and an HLA h: $REACH(s, h)$ is the set of states reachable by any of the HLA's implementations

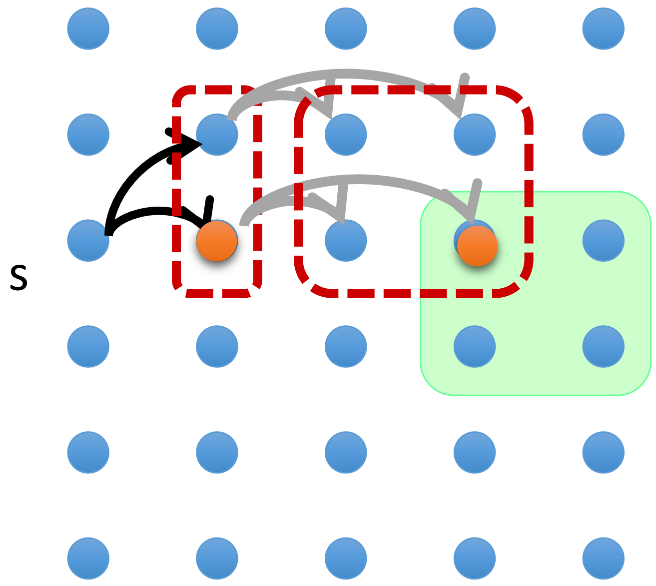
- Reachable set of a sequence of HLAs
  - Reachable set of a sequence of HLAs $[h_1, h_2]$ is the union of all the reachable sets obtained by applying h_2 in each state in the reachable set of h_1:

    $$REACH(s, [h_1, h_2]) = \cup_{s' \in REACH(s,h)} REACH\ (s', h_2)$$

Example



$REACH(s, [h_1])$



$REACH(s, [h_1, h_2])$

Source: RN: Fig 11.6

High-Level Planning

- Practical implications
  - Agent can choose element of the reachable set it ends up in when it executes the HLA
  - HLA with multiple refinements is more “powerful” than the same with fewer refinements
- High-level plan
  - A sequence of HLAs
  - Achieves goal if its reachable set intersects set of goal states
  - Otherwise, the plan does not work
- Search algorithm
  - Search among high-level plans
  - Look for one whose reachable set intersects goal
  - Once that happens, commit to that abstract plan
  - Focus on refining the plan further

Representing HLA Effects

- Effects as reachable sets
  - As reachable set for each possible initial state
  - Represent changes made to each fluent or state variable
- Recall: Primitive action
  - Can add or delete a fluent or variable or leave it unchanged
- HLA
  - Can also control variable value, depending on implementation chosen
  - Description derivable, in principle, from descriptions of its refinements, such that the downward refinement property holds

Representing Reachable Set

- Notations:
  - $\sim$ means possibly, if the agent chooses
  - E.g., $\tilde{+}A$ means "possibly add A", i.e., either leave A unchanged or make it True
- Questions:
  - What do $\tilde{-}A$ and $\tilde{\pm}A$ mean?
- Example

  $Go(Home, SIN)$ with two refinements
  - $Drive(Home, SIN)$ and $Taxi(Home, SIN)$
  - Possibly delete $Cash$ (if agent decides to take a taxi)
  - So should have effect $\tilde{-}Cash$

Example

- Consider:
  - Schemas for HLAs h_1 and h_2:
  - Action (h_1, Precond: $\neg A$, Effect: $A \wedge \tilde{-} B$)
  - Action (h_2, Precond: $\neg B$, Effect: $\tilde{+}A \wedge \tilde{\pm}C$)
- Meaning:
  - h_1 adds A and possibly delete B
  - h_2 possibly adds A and has full control over C
- Exercise:
  - If only B is true in the initial state and goal is $A \wedge C$
  - What sequence of HLAs will achieve the goal?

Summary

- Hierarchical planning
  - Using abstraction to manage complexity
  - Planning as refinements
  - Planning in abstract space
- HTN Planning
  - Focus on tasks instead of goals
  - Use hierarchical decomposition and delayed planning ideas to manage complexity
- Searching for primitive actions
  - Recursive refinement
- Search for abstract actions
  - Downward refinement property
  - Check if reachable set intersects with goals

Example: PANDA

- The PANDA framework for hierarchical planning
  - https://rdcu.be/cn6Ra
  - Höller, D., et al., *The PANDA Framework for Hierarchical Planning.* KI - Künstliche Intelligenz, 2021.
  - Höller, D., et al., *HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems.* Proceedings of the AAAI Conference on Artificial Intelligence, 2020. **34**(06): p. 9883-9891.

HTN Planning Today

- Key idea
  - Construct plan library (knowledge base) of known methods for implementing complex, HLAs
- Approach
  - Learn planning methods from problem-solving experience
  - Save used plan in library as a method for task-specific HLA implementation
  - Accumulate knowledge over time
  - Generalize methods, eliminating problem-specific details, keeping key elements of the plan
- In practice:
  - Many real-world applications; ideas adopted in modern day planning and reinforcement learning
  - Old HTN planners: Noah, Nonlin, O-Plan, SIPE, SIPE-2, SHOP, SHOP2
  - Fast Downward (Helmert 2006) won 2004 IPC; uses hierarchical decomposition of planning tasks to derive heuristics with delayed evaluation in best first search
  - New research trends in hierarchical planning and hierarchical reinforcement learning

Homework

- Readings:
  - Heuristics: RN: 11.3
  - Hierarchical: RN: 11.4
  - Summary: RN: 11.7
  - Complexity: RN Chapter 11, Bibliographical and Historical Notes (last page)
- Reviews:
  - RN: 3.3, 3.5, 3.6 (Review of search and heuristics in problem solving)

References

- Content:
  - Nau, Dana. Lecture Slides from Automate Planning: Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License: http://creativecommons.org/licenses/by-nc-sa/2.0/
- Classical planning systems: (Journal articles publicly available online or through NUS Library e-Resources)

  - Hoffmann, J. and B. Nebel, *The FF planning system: fast plan generation through heuristic search.* J. Artif. Int. Res., 2001. **14**(1): p. 253–302.
  - Helmert, M., *The fast downward planning system.* J. Artif. Int. Res., 2006. **26**(1): p. 191–246.
  - Georgievski, I. and M. Aiello, *HTN planning: Overview, comparison, and beyond.* Artificial Intelligence, 2015. **222**: p. 124-156.

- Classical planning in the Real World:

  - Neufeld, X., et al., Building a Planner: A Survey of Planning Systems Used in Commercial Video Games. IEEE Transactions on Games, 2019. 11(2): p. 91-108. https://ieeexplore.ieee.org/document/8214211
  - Sohrabi, S., AI Planning for Enterprise: Putting Theory Into Practice, in Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19. 2019, International Joint Conferences on Artificial Intelligence Organization. p. 6408--6410. https://www.ijcai.org/proceedings/2019/0897.pdf
  - Estlin, T., et al. *Increased Mars Rover Autonomy using AI Planning, Scheduling and Execution*. in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. 2007. https://ai.jpl.nasa.gov/public/documents/papers/estlin_icra07_marsrover.pdf
  - Rabideau, G.; Wong, V.; Gaines, D.; Agrawal, J.; Chien, S.; Kuhn, S.; Fosse, E.; and Biehl, J. Onboard Automated Scheduling for the Mars 2020 Rover. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation for Space*, of *i-SAIRAS'2020*, Noordwijk, NL, 2020. European Space Agency https://ai.jpl.nasa.gov/public/documents/papers/M2020_OBP_i-SAIRAS2020_camera.pdf