ASH: The Answer Set Programming based Hierarchical Abstraction Planner for Robots

RSMG 5

Oliver Michael Kamperis

Marco Castellani

Yongjing Wang

Background and Motivation

Planning requires reasoning about and making a series of related decisions regarding action and state change into an often large future temporal horizon prior to, or during, execution.

- Classical “flat” planning is intractable for non-trivial problems;
  - Search space exponential in description size,
  - Number of solutions exponential in plan length.
- In real-world, offline complete planning is non-feasible and futile.

- Methods to overcome this include;
  - Specialise planners with application specific heuristics (manual or learnt),
  - Simplify or apply constraints to the problem that reduce search space,
  - Partial-plan online, split original problem into independently solvable sub-problems.

Planning over an Abstraction Hierarchy

- The human approach to overcome complexity of real-world is to plan online by applying abstraction in reasoning;
  - Only decide upon the abstract stages of a plan initially,
  - Deal with the details of each stage as they occur,
  - If plan fails, only have to re-plan small portion of existing plan.

- ASH has the following novel capabilities inspired by this;
  - Plan over a hierarchy of abstractions of a planning domain,
    - Abstract problems exponentially faster to solve,
    - Abstract solutions guide search in more concrete space,
  - Divide and sub-divide problem and solve each sub-problem independently,
    - Allows online partial planning but rarely leads to optimal plans,
    - Multiple types, standard propagation, or look-back propagation.

Planning over an Abstraction Hierarchy

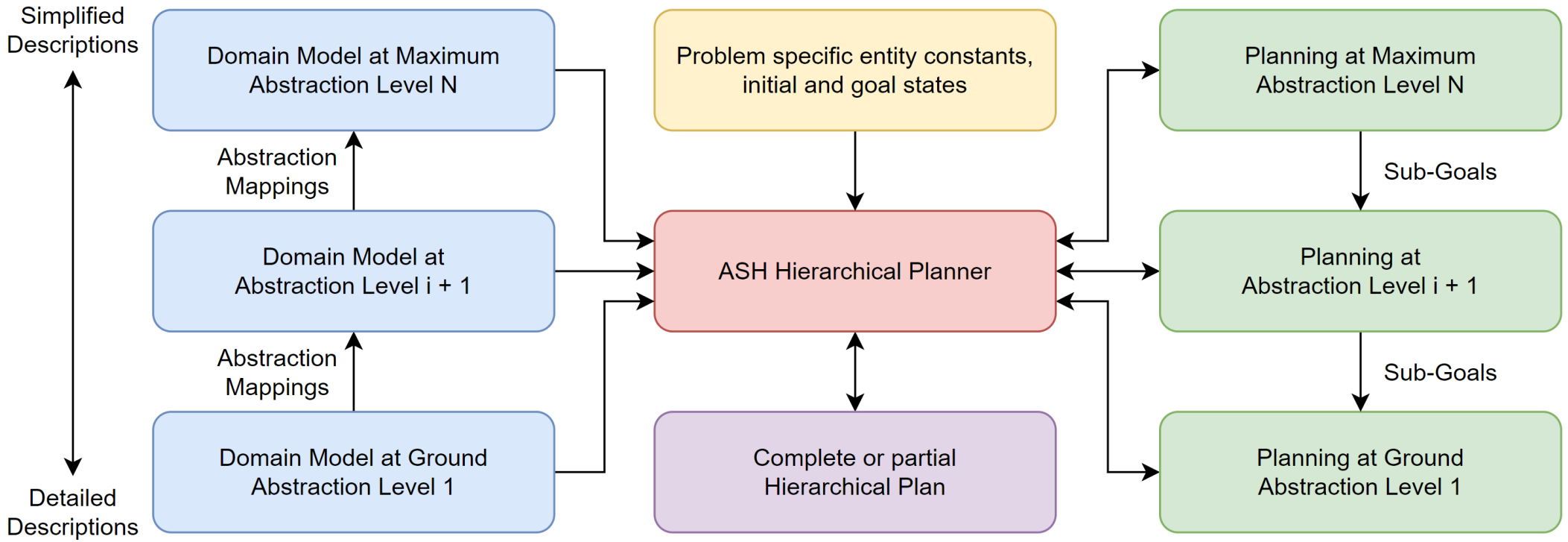
- Planning domain it represented by multiple domain models at different abstraction levels, arranged in a hierarchy.

- Iteratively solve the planning problems defined by the domain model at each abstraction is descending order.
  - Only at the most abstraction level does a complete plan have to be generated,
  - At lower abstractions, solution from the previous level is “skeleton” solution at the next, this provides guidance to search and allows problem splitting.

- Skeleton solution is formed by effects of actions passed as sub-goals;
  - Sub-goals apply **conformance constraint** which limits number of solutions,
  - Constraint simply enforces that plans at all levels **achieve the same effects**,
  - Requires the ability to reason about state representation at different levels.

Planning over an Abstraction Hierarchy

- Design sufficiently expressive ground model to deal with the application.
- Abstractions are then created through various methods.
- **Abstraction mappings** define relationship between state over abstractions.



State Transition Systems

- A three-tuple $\boldsymbol{\lambda^L = <\Sigma^L, A^L, T^L>}$ where;
  - **Σ** is the set of valid states,
  - **A** is the set of actions,
  - **T** is the set of legal transitions,
  - **A** transition of $\boldsymbol{\lambda^L}$ is $\boldsymbol{\tau_t^L = <\sigma_t, \alpha_{t+1}, \sigma_{t+1}> : \tau_t^L \in T^L, \sigma \in \Sigma^L, \alpha \subseteq A^L}$.

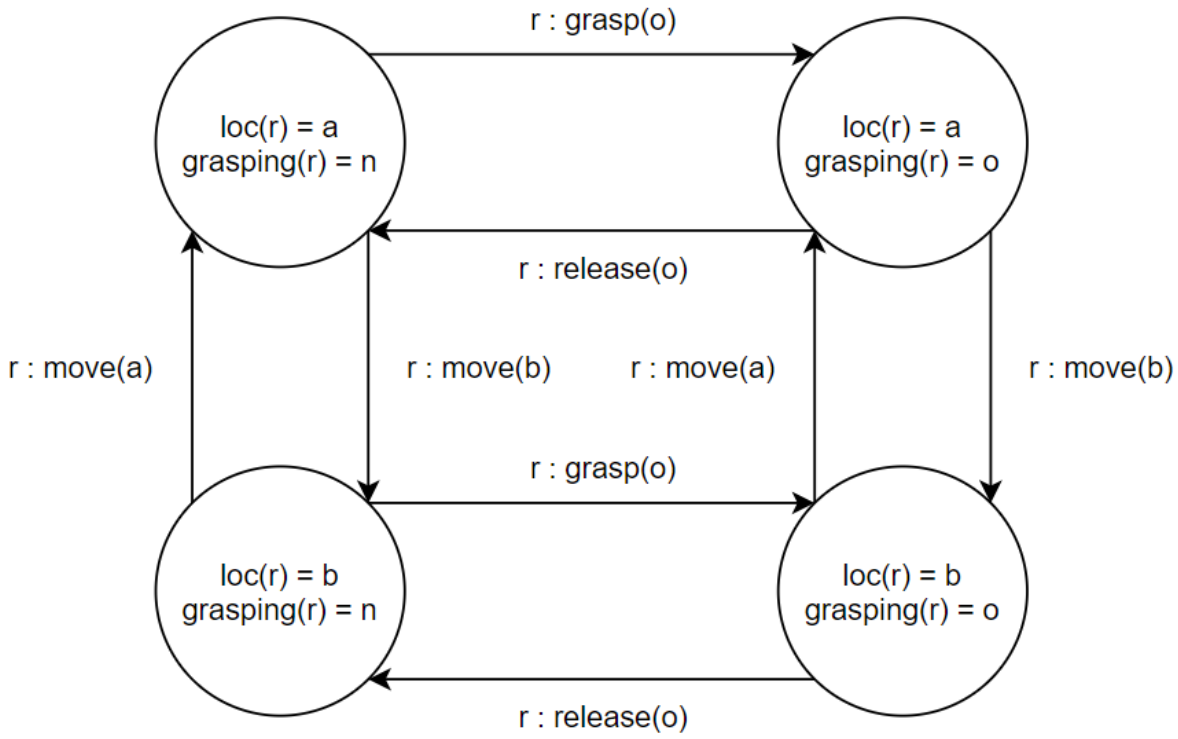
- A directed graph defining a dynamic system with;
  - Discrete deterministic dynamics, action effects Markovian and instantaneous,
  - Similar to finite state automaton, except no fixed initial or final state,
  - Planning requires finding sequence of transitions of this graph.

- Each domain model defines a unique transition system for that level.
- Enforcing conformance between plans at different levels requires the state representation to be mapped between levels, **a function $\boldsymbol{M^L : \Sigma^L \rightarrow \Sigma^{L+1}}$**

State Transition Systems

- Systems are defined under answer-set semantics.
- Each state and transition is unique answer set of program **Π** containing;
  - The domain independent part of ASH,
  - The domain dependent planning domain model, and…

- A valid state is unique answer set of **Π** plus encoding of a complete set of state variables.
- A legal transition is unique answer of **Π** plus an arbitrarily large set of concurrent actions.



Reductions

- Simplest type of abstraction, based on assumptive reasoning;
  - Obtained by removing a condition prohibiting or penalising an action,
  - Assume the condition can be met by a sequence of actions at the next level,
  - Adds or reduces cost of arcs in a state transition system,
  - Result is large reduction in solution length.

- Reductions are well defined and have the upward-solution property;
  - Existence of concrete solution implies existence of abstract solution,
  - This is because concrete solution is also solution to the abstract problem.

- State space is identical, so state mapping is direct and one-to-one;
  - **in^{L+1}(object) = location ← in^{L}(object) = location**

Refinements

- More complex, based on “ancestry” relations between classes and entities;
  - Obtained by relating entities to their components and removing a class type,
  - Based on concepts of resolution and granularity in explanation,
  - Reduces the number of entity constants and state space of transition diagram,
  - Result is large reduction in state space and solution length.

- Refinements also well defined and have the downward-solution property;
  - Existence of abstract solution implies existence of concrete solution,
  - This is because concrete solution is also solution to the abstract problem.

- State space reduces, so state mapping is direct but many-to-one;
  - $\mathbf{in^{L+1}(object) = location_1 \leftarrow in^{L}(object) = location_2, desce\_of(l, location_1, location_2)}$
  - State of refined model seen as the union of the non-refined states that map to it.

Redefinitions

- No limits, anything goes!
  - Removal or replacement of state variables,
  - Replacement of actions with abstract task descriptions.
- Not so well defined, no clear general way to define the mapping between levels, needs to be designed carefully by the engineer.
- Intended only at the highest level for abstract task scheduling;
  - Determine a sequence of high-level tasks that need to be completed,
  - Each task is itself its own planning problem.
- Redefinitions are difficult but potentially very powerful due to the ability to form task schedules form abstract goal descriptions.

Hierarchical Plans

- A four-tuple $H_{j,k}^{L,N}$ = <$I^{L,N}$, $G^{L,N}$, {$\Delta_t^{L,N}$ | t ϵ [j,k)}, {P^m | m ϵ [L,N)} >;
  - $I^{L,N}$ is the initial state,
  - $G^{L,N}$ is the goal state,
  - $\Delta_t^{L,N}$ is a sequence of decomposition trees,
  - $P^{L,N-1}$ is the "trailing" plan.

- A single level plan is simply a sequence of state transitions.
- A hierarchical plan is conceptually a "pyramid" of stacked single plans.
- Decomposition trees define the relations between state transitions at different abstraction levels, i.e. which sequence of low level transitions achieve the same effect as the higher level transition.

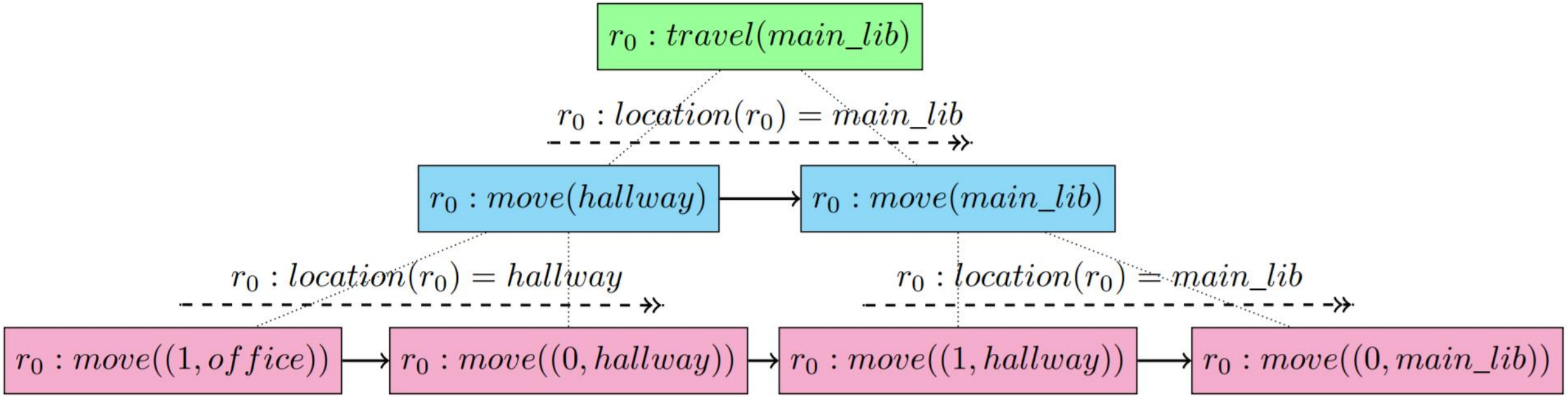
Decomposition Trees

- A three-tuple $\boldsymbol{\Delta_t^{L,N}} = < \boldsymbol{\tau_t^L}, \boldsymbol{\rho(\tau_t^L)}, \{\boldsymbol{\Delta_i^{L,N-1}} \mid \boldsymbol{i \in [j,k) : j >= t, k >= j}\}>$ where;
  - $\boldsymbol{\tau_t^L}$ is the head transition,
  - $\boldsymbol{\rho(\tau_t^L)}$ is the sub-goal set,
  - $\{\boldsymbol{\Delta_i^{L,N-1}} \mid \boldsymbol{i \in [j,k)}\}$ are its child trees.

- The sub-goal set is the set of direct effects of the action set of the head transition.
- The children are a sequence of transitions at the next adjacent abstraction whose final state satisfies these sub-goals.
- Each action set at the maximum abstraction level defines a decomposition tree, **constructing each tree is a sub-problem**.

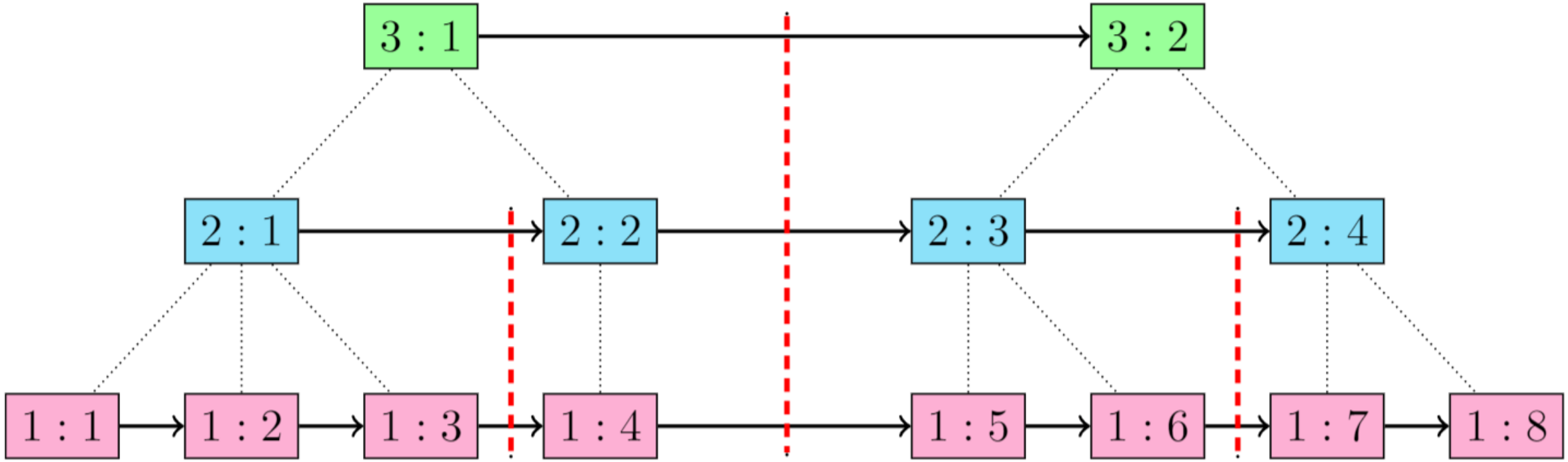
Decomposition Trees

- Each node is the head of its own tree.
- Tree is **complete** if it is either at the ground level or is conformant.
- Tree is **absolute** if is complete and all of its children are absolute.
- Each tree is **independently solvable**.



Decomposition Trees in Hierarchical Plans

- Nodes are of the form **[** abstraction level **:** time step **]**.
- Red lines are possible divisions or sub-divisions;
  - Each division is a tree or sequence of trees,
  - Divisions are solved iteratively from left to right.



Problems Encountered

- Where do we choose to divide a hierarchical plan?
- Do we allow revision of previously completed trees?
- If so how many do we allow to be revised?
- Complexity of constructed tree not indicative of complexity of solving it.