Advanced Planning

Chapter 12

Outline

- ♢ Scheduling and Planning
- ♢ Hierarchical Planning
- ♢ Planning in Non-Deterministic Domains
- ♢ Conditional Planning
- ♢ Execution Monitoring, Replanning and Repair
- ♢ Continuous Planning
- ♢ Multiagent Planning

Scheduling -vs- Planning

- In Chapter 11, the STRIPS reps only allowed us to decide the **relative ordering** among planning actions.

- In Scheduling, we also need to decide the **absolute**:
  - Time when an event/action will occur.
  - Duration of the event/action

Job-Shop Scheduling Problem (JSSP)

Given: K jobs to do, where job = fixed sequence of actions, which:

- Have quantitative time durations.
- Use resources (which are often shared among the different jobs).

Find: A schedule that:

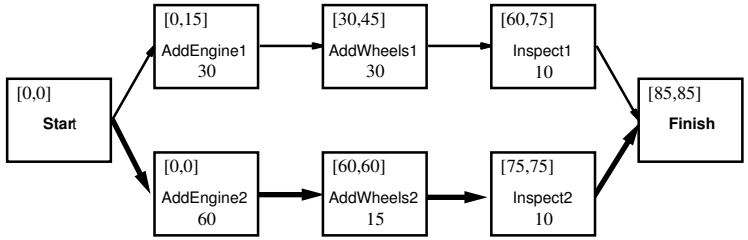
- Determines a start time for each action.
- Obeys all hard constraints - e.g. no temporal overlap between **mutex actions** (i.e., those using same one-act-at-a-time resource,).
- Minimizes the **total time** to perform all actions and jobs.

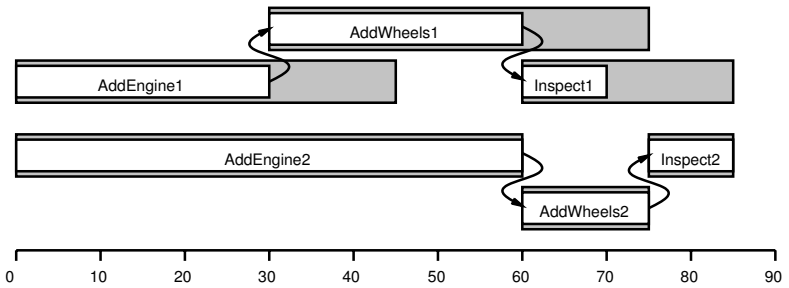
Automobile Assembly Scheduling

- 2 jobs: Assemble cars C1 and C2.
- job = 3 actions = add engine, add wheels, inspect whole car.
- resource constraint = do each act at special one-car-only station.

Planning + Scheduling for Job-Shop Problems

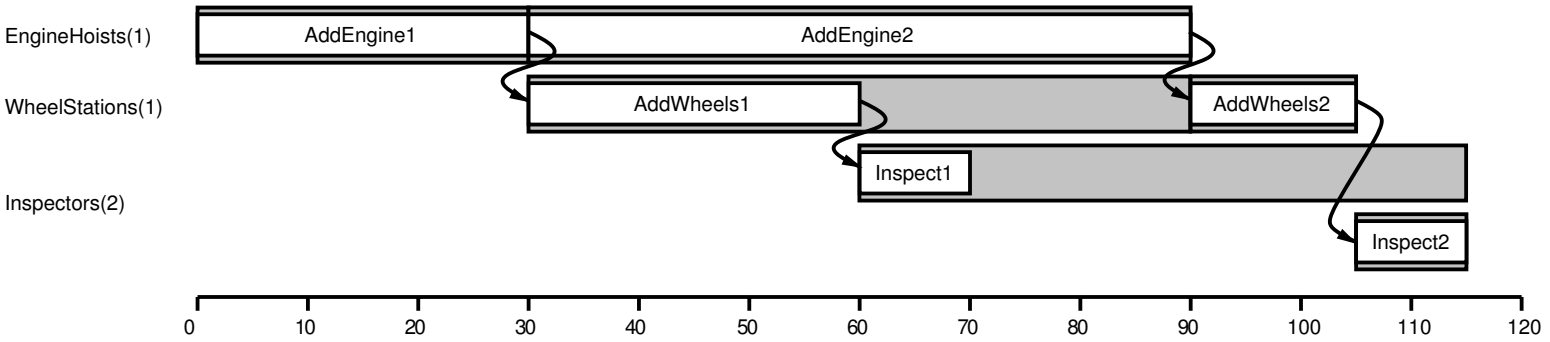
1. Create a partially-ordered plan (Normally given by the JSSP scenario)
2. Use the critical-path method to determine the schedule.





- **Critical path** = longest sequence of actions, each of which has no slack (i.e. it must begin at a particular time, else whole plan delayed).
- This solution assumes no resource constraints. Note that 2 engines are being added simultaneously.

Resource-Constrained JSSP



- Takes longer, since same actions (on different cars) cannot overlap due to resource constraints.
- Critical Path = AddEngine1, AddEngine2, AddWheels2, Inspect2.
- Remaining actions have considerable slack time: they can begin much later without affecting the total plan time.

Planning + Scheduling

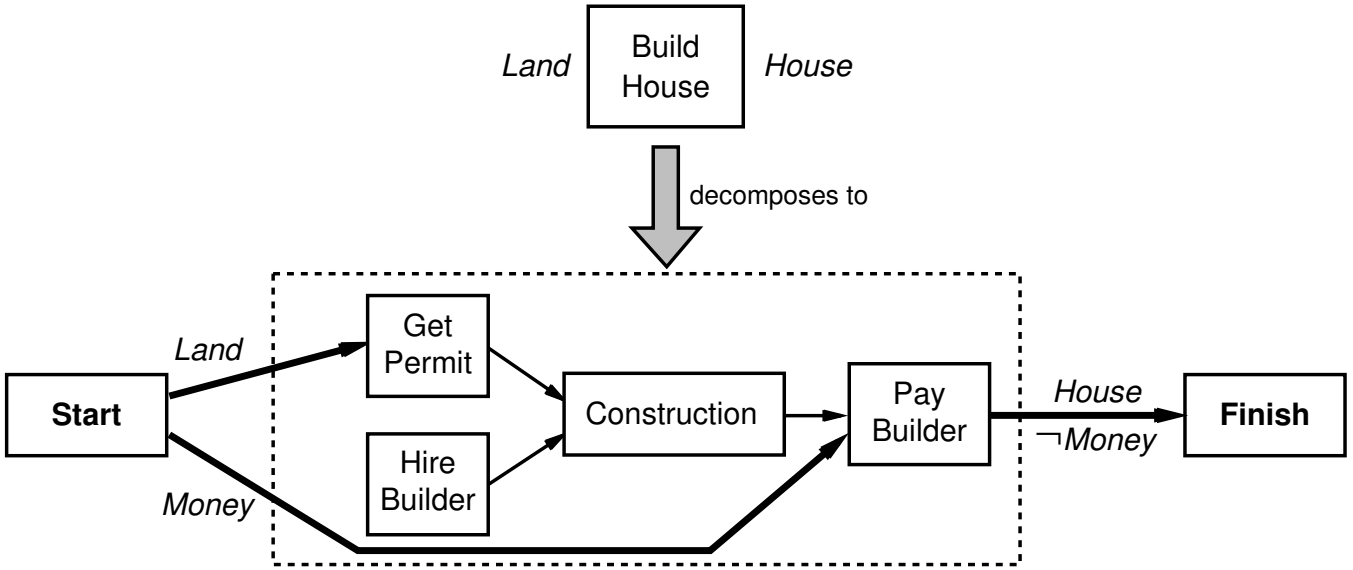
1. Serial: Plan, then Schedule. Use partial- or full-order planners. Then schedule to determine actual start times.
2. Interleaved: Mix planning and scheduling.
   For example, include resource constraints during partial planning.
   These can determine conflicts between actions just as causal links do in POP.

Final Note: We are still working in **classic planning environments** (i.e., observable, deterministic, static and discrete), but for scheduling, we need to consider absolute start times and durations.

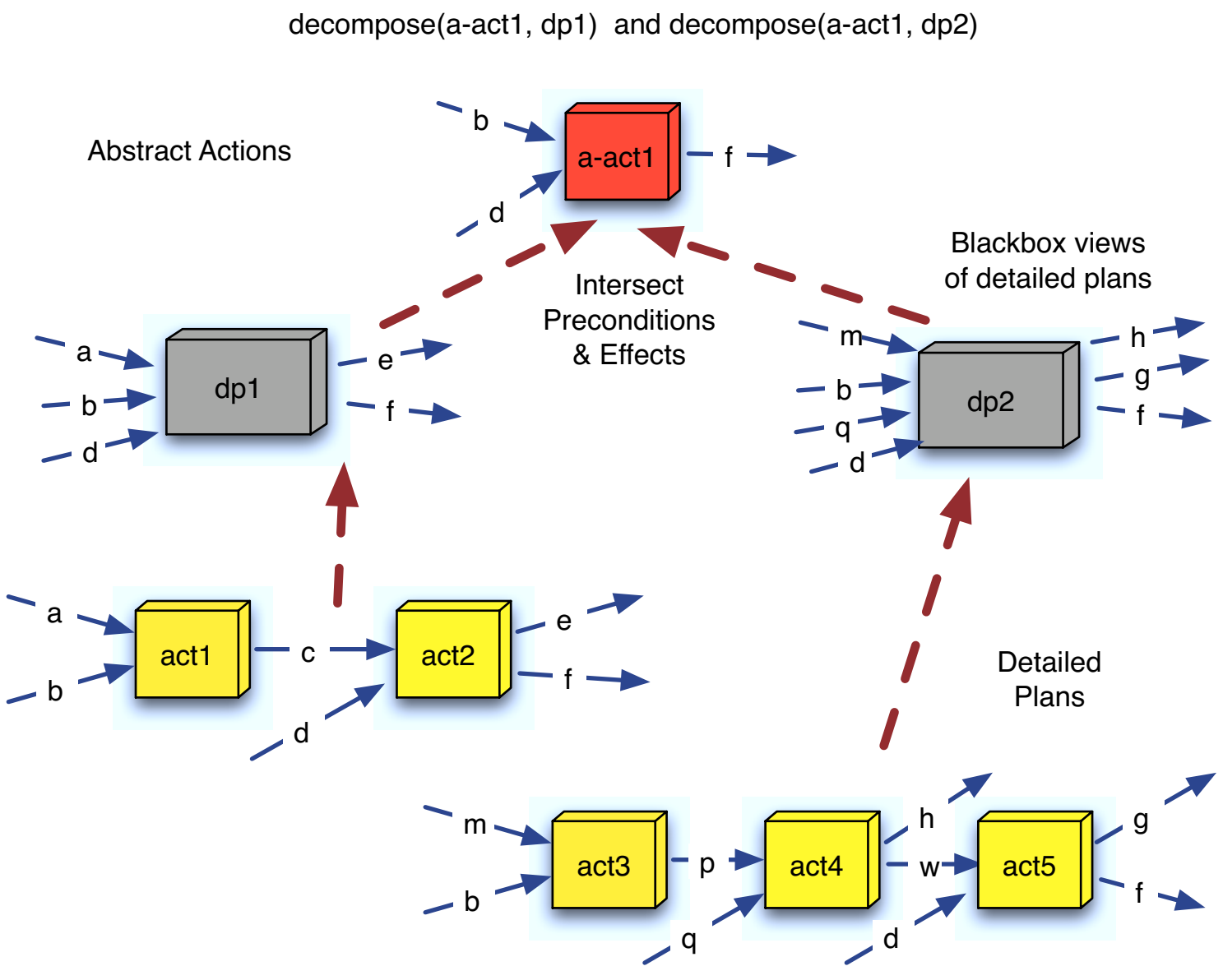
Hierarchical Planning

Hierarchical Problem Decomposition - express actions at one level as **small** sets of actions at the next lower level.
Solve relatively small problems at each level of abstraction.

Hierarchical Task Network (HTN) Planning - begin with abstract plan and continue expanding each activity (i.e., replacing it with its lower-level realization) until plan consists of only low-level actions.



Plan Libraries



Hierarchical POP

In Hierarchical POP, we now have options in the successor function:

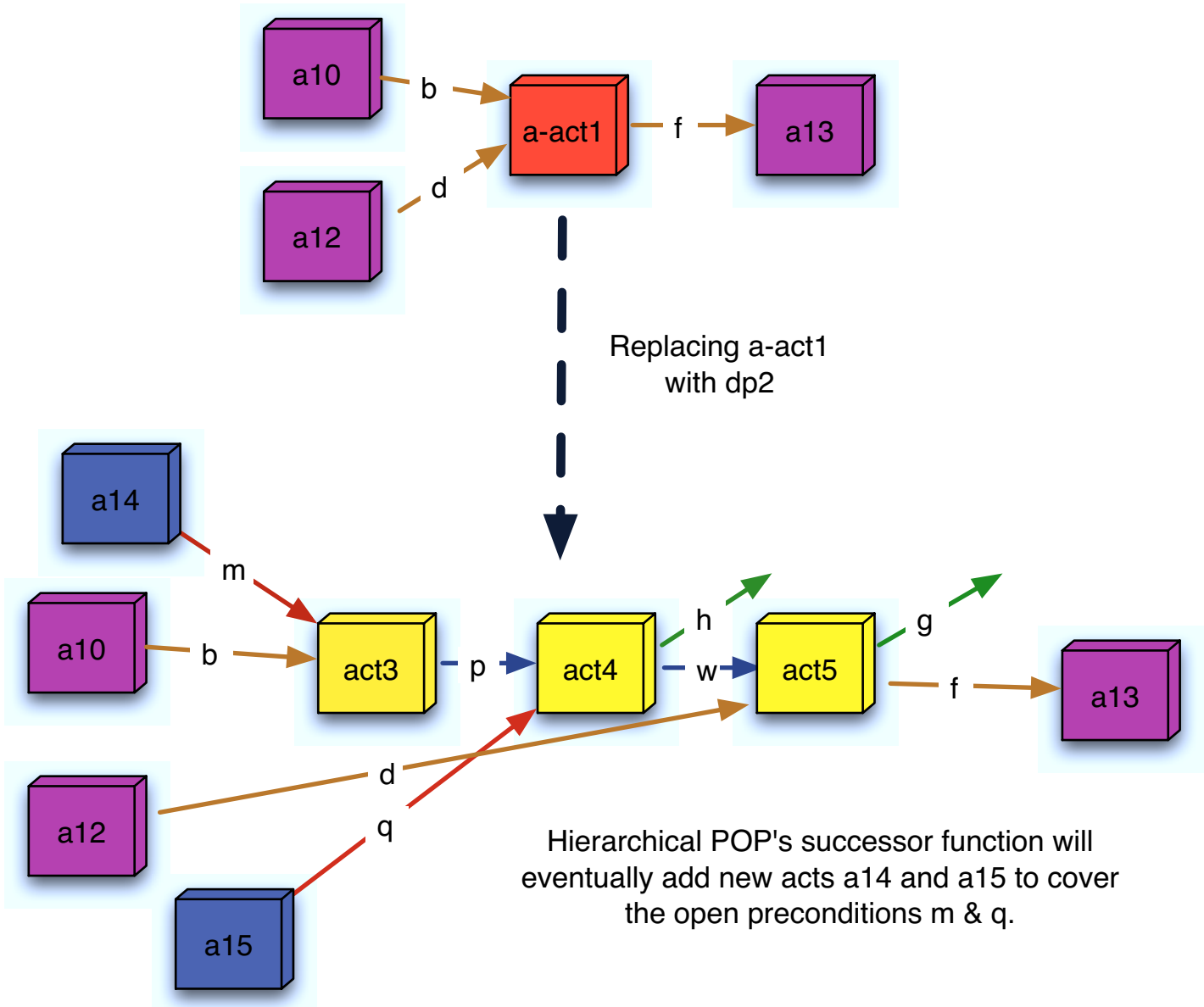
1. Pick an action to fulfill an open precondition
2. Expand an abstract action with one of its decompositions from the plan library.

Expansion via the Decomposition

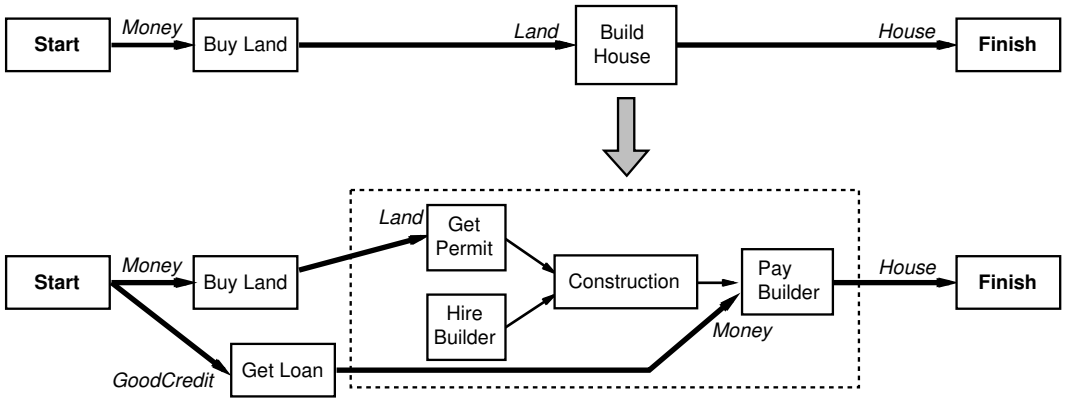
In plan P, to expand abstract-action aa by its decomposition (detailed plan) dp:

1. $\forall a \in acts(dp)$, either add an instance of a to P, or find an existing instance of a that is already in P (subtask sharing).
2. Copy all *internal constraints* (i.e. those between 2 dp actions) into P.
3. $\forall c \in ordering - constraints(P) \ni aa \in c$ Replace c by one or more new ordering constraints that involve acts(dp), but not aa.
   E.g., if $a* \prec aa$, then $\forall b \in acts(dp) : a* \prec b$, but this may be too restrictive. Some acts of dp may be independent of a*.
4. Connect causal links involving aa to the appropriate acts in dp. E.g. satisfied preconditions of aa now become satisfied preconditions of actions in dp.
   But, some preconditions in dp will not be satisfied, i.e. those that were abstracted away in the definition of aa
   Also connect effects of dp to acts in P that aa previously fed into.

Expanding an Abstract Action



Hierarchical POP for House Building



- Replace Buy-Land $\stackrel{Land}{\longrightarrow}$ Build-House with Buy-Land $\stackrel{Land}{\longrightarrow}$ Get-Permit.
- The Money precondition to Pay-Builder was not part of the abstract act (Build-House), so it has no immediate satisfier. Eventually, Hierarchical POP adds the Get-Loan action to cover this.
- Note that although Buy-Land $\prec$ Build-House, it need not precede every action of the decomposition, such as Hire-Builder, which is probably independent of Buy-Land.
- However, when money is a limited resource, then Buy-Land and Hire-Builder will become tightly interdependent, since buying an expensive lot may force you to hire a cheap builder!

Indeterminacy in the Real World

- Classic Planning Environments: Fully observable, deterministic, static, discrete.
- But the world is often unpredictable.
- And our knowledge of both the world and the effects of our behaviors on it are:
  - incorrect - we have a bit of knowledge that is wrong.
  - incomplete - we lack important knowledge about certain things.
- The possibilities for having correct and complete knowledge depend upon the degree of **indeterminacy** in the world:
  - **Bounded Indeterminacy:** The effects of some actions are nondeterministic, but there are only a finite number of possibilities
  - **Unbounded Indeterminacy:** nondeterministic and non-enumerable.
  - $\exists$ different non-classical planning systems, 2 for each type of indeterminacy.

Planning with Bounded Indeterminacy

Task: Get my daughters out of bed in the morning.
Finite set of responses to each action.

1. Sensorless Planning - Ignore the world! Design plan that is guaranteed to work, despite unpredictability.
   Coercion: Use acts that (eventually) force system into a desired state.
   (a) Turn bed upside-down.
   (b) Turn on light, remove blankets, crank radio volume, wait.
2. Conditional Planning - plan = **tree**, with different conditional branches. Use **sensors** to determine real result of a plan step, then act accordingly.
   (a) Shake vigorously. If no response, shake again. ELSE: Say "Good Morning...now get OUT of bed!"
   (b) If respone to "get OUT.." is "No", then explain importance of punctuality, a good education, etc. ELSE: Ask "What do you want for breakfast?"

Planning with Unbounded Indeterminacy

Task: Daughter wake-up, but with unpredictable set of responses.

1. Execution monitoring and replanning - You cannot enumerate all possible consequences of actions, so you cannot generate a plan tree ahead of time.
   Instead, use sensors to monitor results of actions, then modify the plan if reality $\neq$ expectations.
   (a) Response = "Only if I can take all my CDs to school" $\Rightarrow$ Plan a negotiation strategy, then execute.
   (b) Response = "But today school starts an hour late!" $\Rightarrow$ Plan for a return visit in 1 hour.
   (c) Any response $\Rightarrow$ Do cost-benefit analysis of dragging (kicking and screaming) daughters out of bed.

Planning with Unbounded Indeterminacy (2)

2. Continuous planning - system remains in the environment, continuously forming new goals and plans, and (hopefully) learning from experience, e.g. learning probabilities of nondeterministic outcomes to build partial conditional planners, which have cached contingency plans for high-probability (or low probability but highly important) events.
   - Parenthood = Waking children, Putting to bed, Getting them to do homework, etc.
   - All involve plans that undergo continuous modification.
   - Responses are infinite, but often very predictable in character.
   - New goals are always arising. E.g., Convince that boys should be avoided until they are 25!

Cond Planning in Fully Observable Envirs

- The agent cannot predict (with certainty) the outcomes of its actions
- But, when the action is executed in the world, then the agent can observe the results (with certainty).
- So the agent's picture of the world is complete and accurate, but only at execution time.
- E.g. Blocksworld where newly-stacked blocks **sometimes** fall onto table.

BlocksWorld

PRECONDITION: $Clear(x) \wedge Clear(y) \wedge On(x, z)$
ACTION: Stack(x,y)
EFFECT: $(\neg Clear(y) \wedge \neg On(x, z) \wedge On(x, y) \wedge Clear(z))$
$\vee\ (\mathbf{On(x, table)} \wedge (\mathbf{WHEN : z \neq table \rightarrow \neg On(x, z)}))$

- The disjunction in the effects indicates need for conditional planning.
- The preconditions are considered at planning time.
- The condition of the WHEN is only evaluated at plan-execution time.

Conditional Wake-Up Planning

When I try to wake up my daughter by shaking her, she may either wake up happy or not wake up at all. Also, when the light is on, she **sometimes** wakes up and is irritated.

PRECONDITION: $In(daughter, bed) \wedge Asleep(daughter)$
ACTION: Shake(daughter)
EFFECT: $(Awake(daughter) \wedge Happy(daughter)) \vee$
$\{\} \vee$
$(Awake(daughter) \wedge (WHEN : On(light) \rightarrow \neg Happy(daughter))$

Conditional Wake-Up Planning (2)

- At planning time, we cannot predict the result of shake(daughter), even if we know that on(light) is true.
- However, during plan **execution**, we will be able to **accurately observe** the result of shake(daughter), and the planner should then give options for dealing with all 3 possibilities:
  1. $(Awake(daughter) \wedge Happy(daughter)) \Rightarrow Greet$
  2. $\{\} \Rightarrow shake(daughter)$ Again!
  3. $(Awake(daughter) \wedge \neg Happy(daughter)) \Rightarrow turnoff(light)$

Games Against Nature

Conditional Planning $\approx$ playing a game against the world:

- We know exactly what moves we can make.
- But we do not know (with certainty) their effects

- $\approx$ not knowing how an opponent will respond in a 2-person game

- So the planner needs a plan for every possible outcome of its actions.

- This is similar to Minimax, but planner does not assume WORST possible outcomes.
- It assumes ANY of the outcomes could occur, and includes an action to handle each.
- So the plan is an AND-OR graph, with:
  - AND node = Conditional action for all outcomes
  - OR node = choice of agent actions.

Cond Planning in Partially Observable Envirs

- Uncertain actions: The agent cannot predict (with certainty) the outcomes of its actions.
- Imperfect Sensors: When the action is executed in the world, the agent cannot necessarily observe the results with 100% accuracy.
- So the agent's picture of the world is incomplete and/or inaccurate.
- E.g. In Wakeup-World, my daughter can FAKE that she is asleep. Then when the agent's sensors detect *asleep(daughter)*, that may not be the real state of the world.
- So the state of the world needs to be represented as the planner's belief state, which can be disjunctive.
- Thus, a belief state represents k **possible worlds**
- Applying an action to a state $=$ applying to EACH of the k possible worlds.
- Each of the k child states may then be further divided if there are any uncertain observables. $\rightarrow$ combinatorial explosion of states!!

Cond Partially-Observable Wakeup Planning

Assume:

1. Kids often fake being asleep.
2. Kids gets very stubborn if shaken while already awake.
3. Once in a stubborn state, they refuse to respond to further shaking.

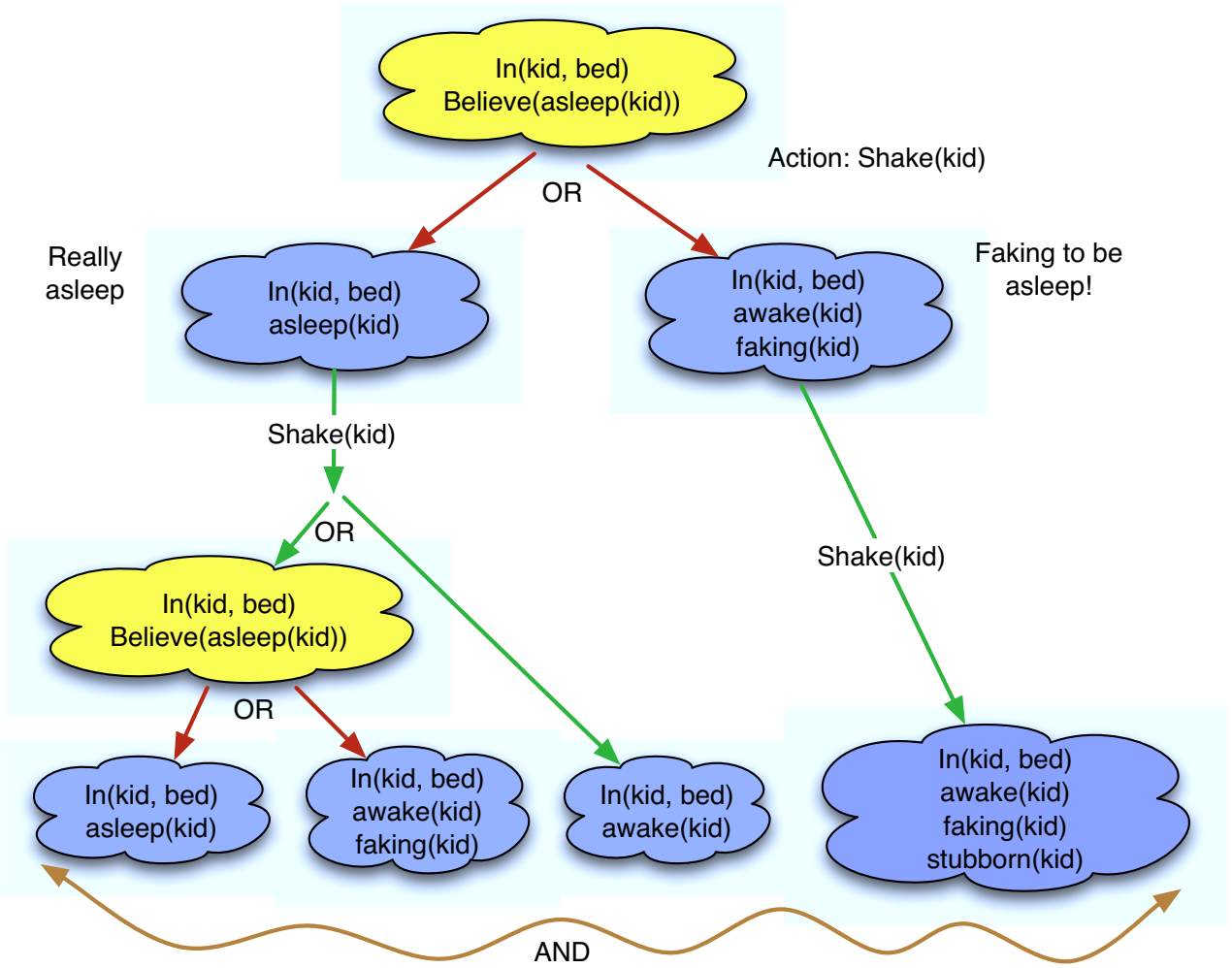
$Believe(asleep(kid)) \rightarrow asleep(kid) \vee (awake(kid) \wedge faking(kid))$

PRECONDITION: $In(kid, bed) \wedge awake(kid)$
ACTION: Shake(kid)
EFFECT: $stubborn(kid)$

PRECONDITION: $stubborn(kid)$
ACTION: Shake(kid)
EFFECT: $\{\}$

Waker Faker

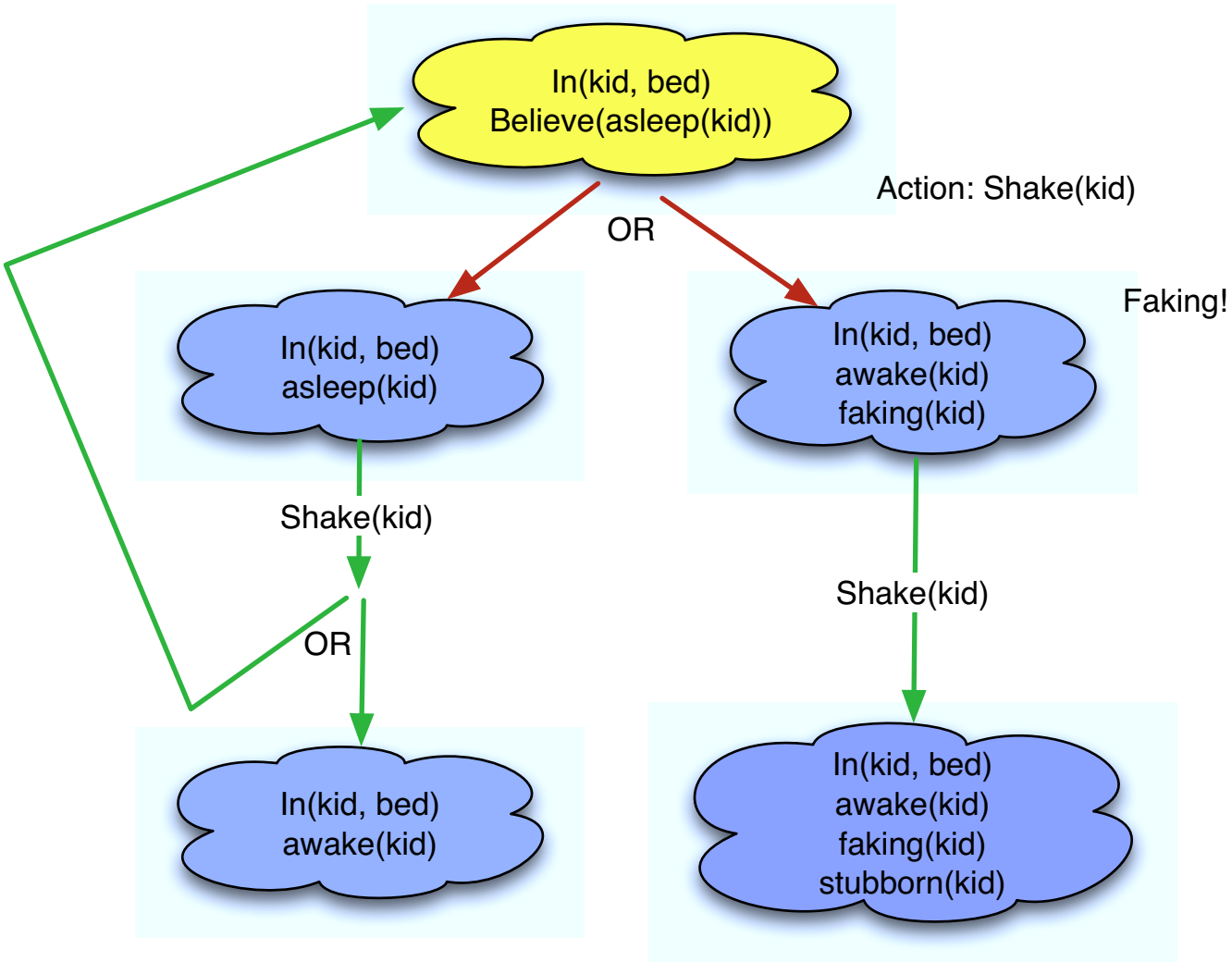
Applying actions to states containing beliefs



4 possible results of shaking a kid whom you believe to be asleep.
The planner needs to have actions for each contingency (AND)

Waker Faker (2)

The AND-OR Tree is really a graph.



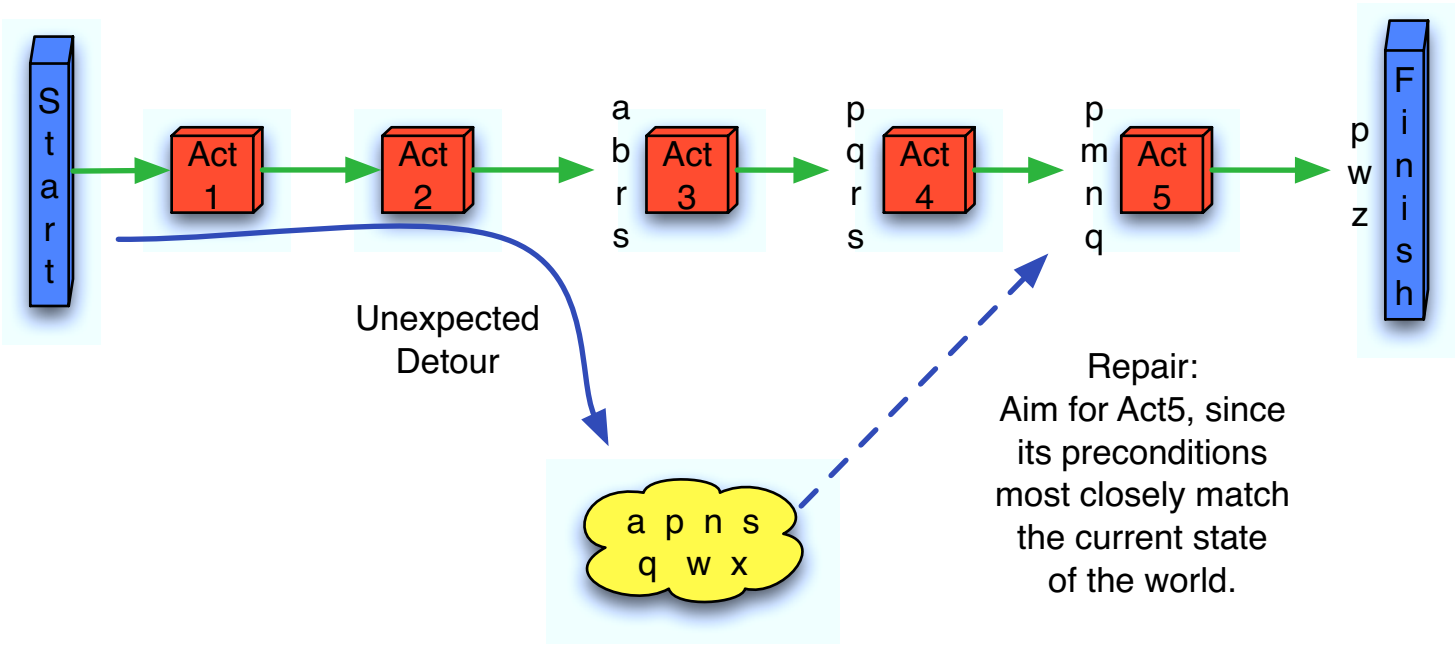
Execution Monitoring and Replanning

- Use sensors to keep track of progress during plan execution.
- When reality $\neq$ expectations (i.e. sensor readings $\neq$ plan states), repair or replan!
- Essential in real-world environments.
- Applicable to full and partially-observable environments
- Works with state-space, partial-order, and conditional plans.
- Monitoring:
  - Action monitoring - check that last action (A) achieved predicted effects.
  - Plan monitoring - check that remainder of plan (post-A) is still possible to execute.
    I.e., preconditions for post-A steps that are satisfied by pre-A steps should still be satisfied.

Repair and Replanning

Fixes when plan, P, achieving goal, G, fails:

- Replan - Keep G, but come up with new plan, P*.
- Repair - get real-world back in a situation from which P can continue. Good heuristic: aim for action (A) in P where preconditions(A) $\approx$ current world state.



Continuous Planning

- Planner **lives** in the world.
- Constantly changing goals in light of new needs.
- Constantly changing plans due to non-static environment, unpredictable action results, etc.
- True literals suddenly become false, solved preconditions become re-opened, etc.

Kid Party

Assume:

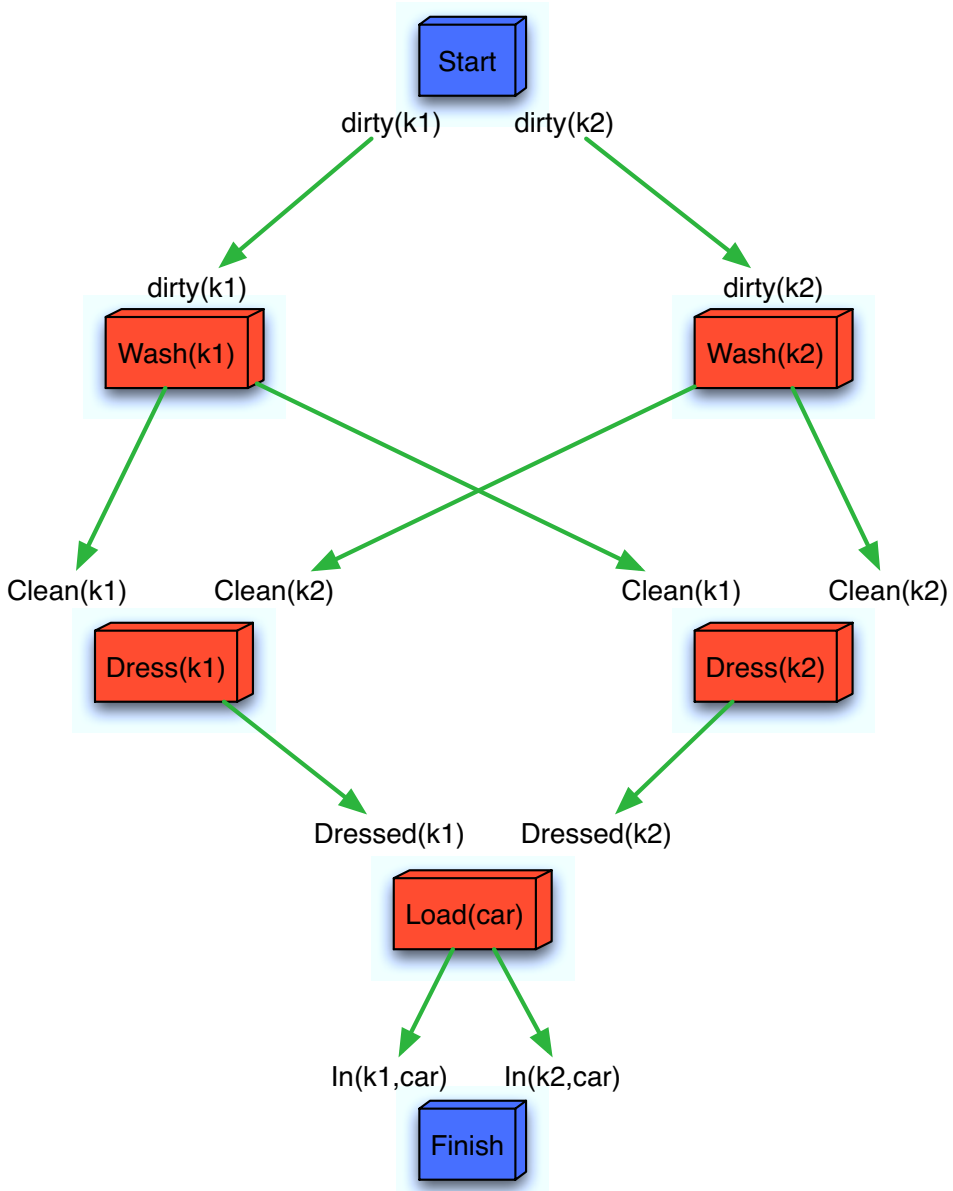
- Prepare kids for party $\Rightarrow$ clean, dress up and put in car.
- A kid cannot be dressed until all the kids are clean. Otherwise, any dirty kid can mess up another kid's clean clothes.
- Only two kids, k1 and k2, in the family.

PRECONDITION: $Dirty(kid)$
ACTION: Wash(kid)
EFFECT: $Clean(kid) \wedge \neg Dirty(kid)$

PRECONDITION: $Clean(k1) \wedge Clean(k2)$
ACTION: Dress(kid)
EFFECT: $Dressed(kid)$

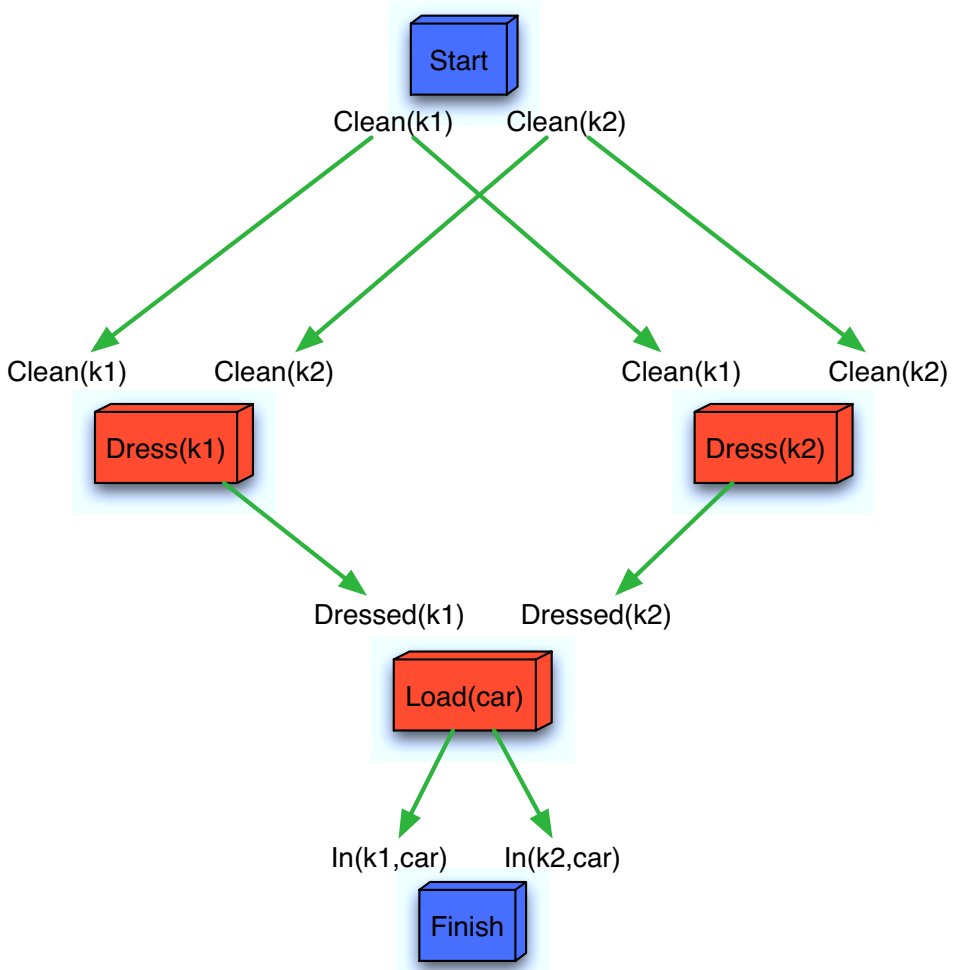
PRECONDITION: $Dressed(k1) \wedge Dressed(k2)$
ACTION: Loadcar
EFFECT: $In(k1, car) \wedge In(k2, car)$

Kid Party Continuous Planning



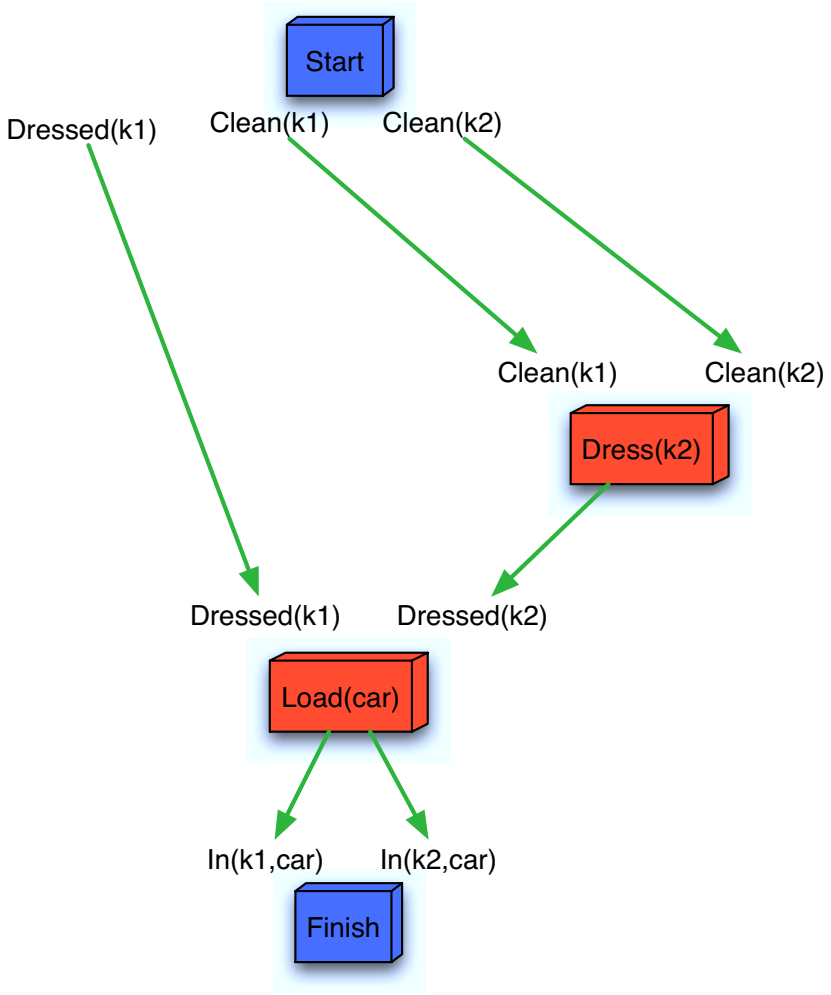
Kid Party Continuous Planning (2)

After washing both kids
Notice that new aspects of the current state (clean(k1) and clean(k2)) are moved to Start.



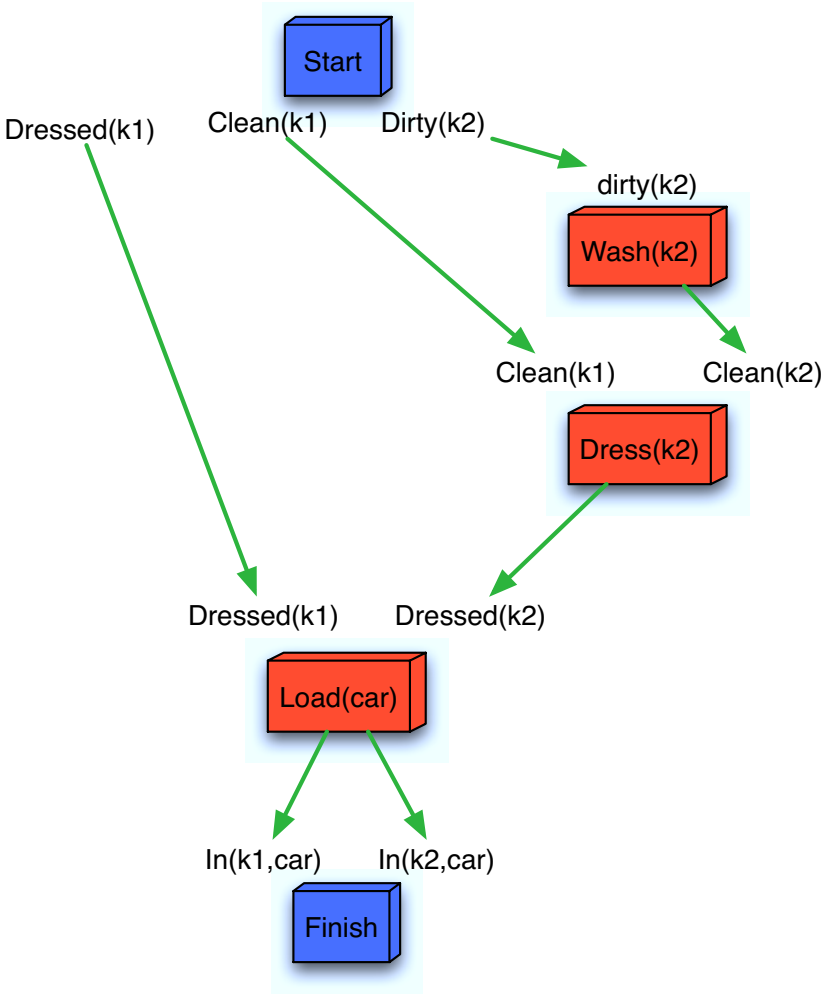
Kid Party Continuous Planning (3)

After dressing K1.
Dressed(k1) moved to Start.



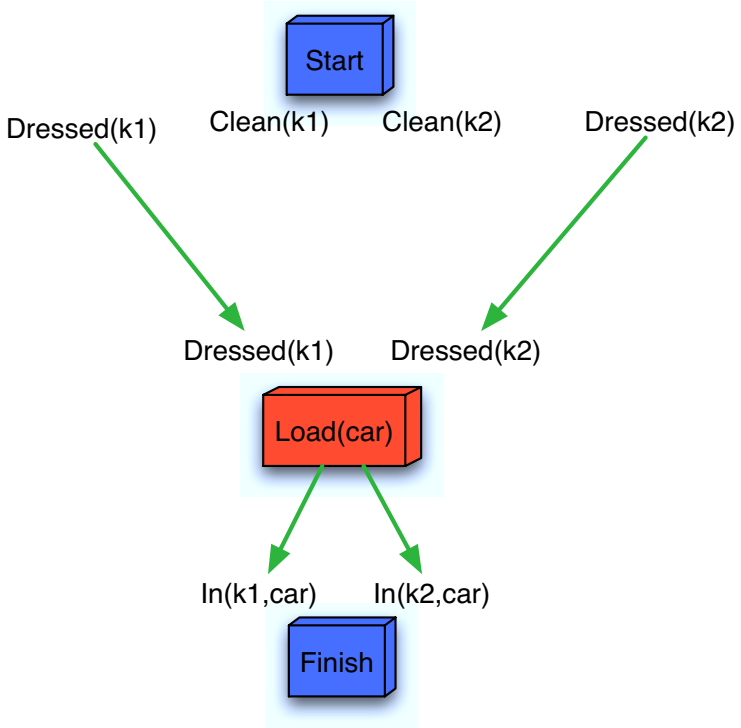
Kid Party Continuous Planning (4)

Unexpected event: K2 gets dirty again!
Modify plan to satisfy open precondition of Dress(K2)



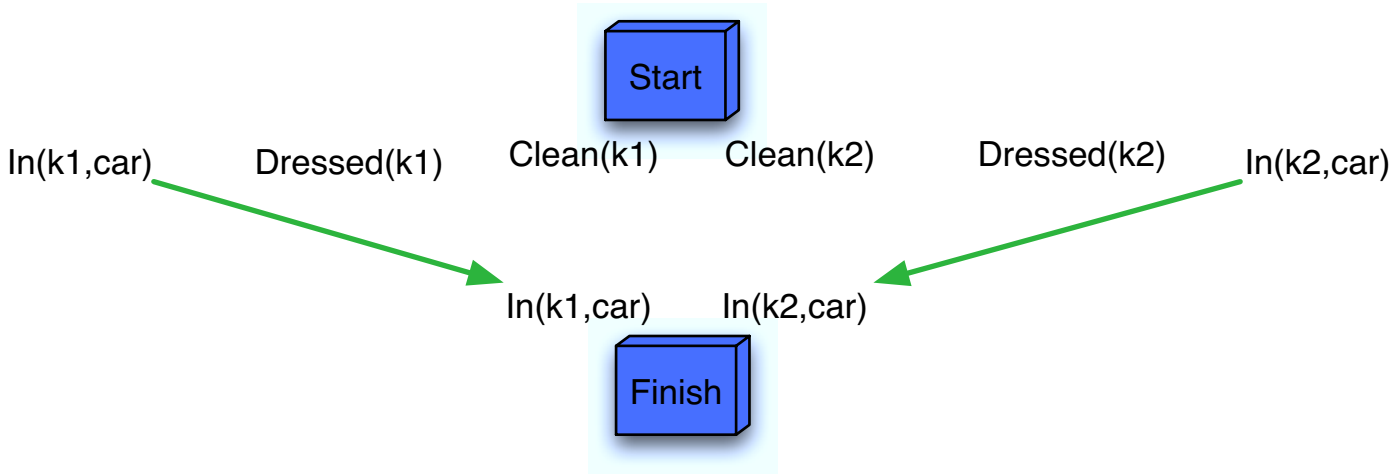
Kid Party Continuous Planning (5)

Wash and Dresss K2.



Kid Party Continuous Planning (6)

- Load car.
- All preconditions/subgoals of goal are now true in present (start) state.
- Plan-execution successfully completed!



Multi-Agent Planning

- While nature is usually indifferent to plans (in *games with nature*).
- Other agents normally **cooperate** or **compete**.

Cooperation

- Coordination - All agents need to work together constructively. They shouldn't be undoing each other's subgoals or otherwise blocking actions.
- Joint Planning - If the total plan is globally written, then the agents can be treated as resources and be properly distributed among tasks.
- Emergent Behavior - If autonomous agents, then each needs strategy (possibly learned) for interacting with envir + agent(s) so that agent group achieves global goal. Difficult! Fun!
  Example: Boids (http://www.red3d.com/cwr/boids/)

Cooperation

No need for complex interactive plans!
Tools: Minimax search, Game theory

Classifying Planning Tasks + Environments

	Obs	Det	Stability	Resol
	Full, Partial	Full, Bd/Unbd Indet	Stat, Dyn	Disc, Cont
Chess				
8-puzzle				
Yahtzee				
Tennis				
Hunting				
Taxi Routing				