Section #7: Review of Knowledge Representation and Planning (we only got through representation)

Matthew Tong
May 23, 2007
Slides adapted from Tom Lenaerts
Ontological engineering

• How to create more general and flexible representations.
  – Many universal concepts like actions, time, physical object and beliefs
  – Operates on a bigger scale than K.E.

• Define general framework of concepts
  – Upper ontology

• Limitations of logic representation
  – Red, green and yellow tomatoes: exceptions and uncertainty
The upper ontology of the world
Difference with special-purpose ontologies

• A general-purpose ontology should be applicable in more or less any special-purpose domain.
  – Add domain-specific axioms

• In any sufficiently demanding domain different areas of knowledge need to be unified.
  – Reasoning and problem solving could involve several areas simultaneously

• What do we need to express?
  Categories, Measures, Composite objects, Time, Space, Change, Events, Processes, Physical Objects, Substances, Mental Objects, Beliefs
Categories and objects

- KR requires the organisation of objects into categories
  - Interaction at the level of the object
  - Reasoning at the level of categories
- Categories play a role in predictions about objects
  - Based on perceived properties
Categories and Logic

• Categories can be represented in two ways by FOL
  – Predicates: apple(x)
  – *Reification* of categories into objects: apples
    • Make categories into constant terms
    • Have statements like x ∈ Cats or member(x, Cats)

• Category = set of its members
Category organization

- Relation = *inheritance*:
  - All instance of food are edible, fruit is a subclass of food and apples is a subclass of fruit then an apple is edible.
- Defines a taxonomy
FOL and categories

• An object is a member of a category
  – MemberOf(BB_{12}, Basketballs)
  – BB_{12} ∈ Basketballs

• A category is a subclass of another category
  – SubsetOf(Basketballs, Balls)
  – Basketballs ⊂ Balls

• All members of a category have some properties
  – \( \forall x \ (x ∈ \text{Basketballs}) \Rightarrow \text{Round}(x) \)

• All members of a category can be recognized by some properties
  – \( \forall x \ (\text{Orange}(x) \land \text{Round}(x) \land \text{Diameter}(x)=9.5\text{in} \land \)
    MemberOf(x, Balls) ⇒ MemberOf(x, BasketBalls))

• A category as a whole has some properties
  – MemberOf(Dogs, DomesticatedSpecies)
  – Can have categories of categories
Relations between categories

- Two or more categories are **disjoint** if they have no members in common:
  - Disjoint(s) ⇔ (∀ c₁, c₂ c₁ ∈ s ∧ c₂ ∈ s ∧ c₁ ≠ c₂ ⇒ Intersection(c₁,c₂) ={})
  - Example; Disjoint({animals, vegetables})
- A set of categories s constitutes an **exhaustive decomposition** of a category c if all members of the set c are covered by categories in s:
  - E.D.(s,c) ⇔ (∀ i i ∈ c ⇒ ∃ c₂ c₂ ∈ s ∧ i ∈ c₂)
  - Example: ExhaustiveDecomposition({Americans, Canadian, Mexicans}, NorthAmericans).
Relations between categories

• A *partition* is a disjoint exhaustive decomposition:
  – Partition(s,c) $\iff$ Disjoint(s) $\land$ E.D.(s,c)
  – Example: Partition({Males,Females},Persons).

• Is (\{Americans,Canadian, Mexicans\},NorthAmericans) a partition?

• Categories can be defined by providing necessary and sufficient conditions for membership
  – $\forall x \, Bachelor(x) \iff Male(x) \land Adult(x) \land Unmarried(x)$
Natural kinds

• Many categories have no clear-cut definitions (chair, bush, book).

• Tomatoes: sometimes green, red, yellow, black. Mostly round.

• One solution: category $Typical(Tomatoes)$.
  – $\forall x, x \in Typical(Tomatoes) \Rightarrow \text{Red}(x) \land \text{Spherical}(x)$.
  – We can write down useful facts about categories without providing exact definitions.

• What about “bachelor”? Quine challenged the utility of the notion of *strict definition*. We might question a statement such as “the Pope is a bachelor”.

Physical composition

• One object may be part of another:
  – PartOf(Bucharest,Romania)
  – PartOf(Romania,EasternEurope)
  – PartOf(EasternEurope,Europe)

• The PartOf predicate is transitive (and irreflexive), so we can infer that PartOf(Bucharest,Europe)

• More generally:
  – ∀ x PartOf(x,x)
  – ∀ x,y,z PartOf(x,y) ∧ PartOf(y,z) ⇒ PartOf(x,z)

• Often characterized by structural relations among parts.
  – E.g. Biped(a) ⇒

\[ \exists l_1, l_2, b (\text{Leg}(l_1) \land \text{Leg}(l_2) \land \text{Body}(b) \land \]
\[ \text{PartOf}(l_1, a) \land \text{PartOf}(l_2, a) \land \text{PartOf}(b, a) \land \]
\[ \text{Attached}(l_1, b) \land \text{Attached}(l_2, b) \land \]
\[ l_1 \neq l_2 \land (\forall l_3 (\text{Leg}(l_3) \Rightarrow (l_3 = l_1 \lor l_3 = l_2))) \]
Measurements

- Objects have height, mass, cost, .... Values that we assign to these are measures.
- Combine Unit functions with a number: \( \text{Length}(L_1) = \text{Inches}(1.5) = \text{Centimeters}(3.81) \).
- Conversion between units:
  \( \forall i \, \text{Centimeters}(2.54 \times i) = \text{Inches}(i) \).
- Some measures have no scale: Beauty, Difficulty, etc.
  - Most important aspect of measures: is that they are orderable.
  - Don't care about the actual numbers. (An apple can have deliciousness .9 or .1.)
Actions, events and situations

- Reasoning about outcome of actions is central to KB-agent.
- How can we keep track of location in FOL?
  - Remember the multiple copies in PL.
- Representing time by situations (states resulting from the execution of actions).
  - Situation calculus
Actions, events and situations

- Situation calculus:
  - Actions are logical terms
  - Situations are logical terms consisting of
    - The initial situation $I$
    - All situations resulting from the action on $I$ ($= Result(a,s)$)
  - Fluent are functions and predicates that vary from one situation to the next.
    - E.g. $\neg Holding(G_1, S_0)$
  - Eternal predicates are also allowed
    - E.g. $Gold(G_1)$
Actions, events and situations

- Results of action sequences are determined by the individual actions.
- *Projection task*: an SC agent should be able to deduce the outcome of a sequence of actions.
- *Planning task*: find a sequence that achieves a desirable effect
Actions, events and situations
Describing change

• Simples Situation calculus requires two axioms to describe change:
  – Possibility axiom: when is it possible to do the action
    \[ At(Agent,x,s) \land Adjacent(x,y) \Rightarrow Poss(Go(x,y),s) \]
  – Effect axiom: describe changes due to action
    \[ Poss(Go(x,y),s) \Rightarrow At(Agent,y,Result(Go(x,y),s)) \]

• What stays the same?
  – Frame problem: how to represent all things that stay the same?
  – Frame axiom: describe non-changes due to actions
    \[ At(o,x,s) \land (o \neq Agent) \land \neg Holding(o,s) \Rightarrow \]
    \[ At(o,x,Result(Go(y,z),s)) \]
Representational frame problem

• If there are F fluents and A actions then we need AF frame axioms to describe other objects are stationary unless they are held.
  – We write down the effect of each actions

• Solution; describe how each fluent changes over time
  – Successor-state axiom:

\[
Pos(a,s) \Rightarrow \left( At(Agent,y, Result(a,s)) \iff (a = Go(x,y)) \lor \left( At(Agent,y,s) \land a \neq Go(y,z) \right) \right)
\]

  – Note that next state is completely specified by current state.
  – Each action effect is mentioned only once.
Other problems

• How to deal with secondary (implicit) effects?
  – If the agent is carrying the gold and the agent moves then the gold moves too.
  – Ramification problem

• How to decide EFFICIENTLY whether fluents hold in the future?
  – Inferential frame problem.

• Extensions:
  – Event calculus (when actions have a duration)
  – Process categories
Mental events and objects

• So far, KB agents can have beliefs and deduce new beliefs
• What about knowledge about beliefs? What about knowledge about the inference process?
  – Requires a model of the mental objects in someone’s head and the processes that manipulate these objects.
• Relationships between agents and mental objects: believes, knows, wants, …
  – Believes(Lois, Flies(Superman)) with Flies(Superman) being a function … a candidate for a mental object (reification).
  – Agent can now reason about the beliefs of agents.
The internet shopping world

- A Knowledge Engineering example
- An agent that helps a buyer to find product offers on the internet.
  - IN = product description (precise or ¬precise)
  - OUT = list of webpages that offer the product for sale.
- Environment = WWW
- Percepts = web pages (character strings)
  - Extracting useful information required.
The internet shopping world

- Find relevant product offers
  
  $RelevantOffer(page, url, query) \iff Relevant(page, url, query) \land Offer(page)$

  - Write axioms to define Offer(x)
  - Find relevant pages: Relevant(x,y,z) ?
    - Start from an initial set of stores.
    - What is a relevant category?
    - What are relevant connected pages?
  - Require rich category vocabulary.
    - Synonymy and ambiguity
  - How to retrieve pages: $GetPage(url)$?
    - Procedural attachment

- Compare offers (information extraction).
Reasoning systems for categories

• How to organise and reason with categories?
  – Semantic networks
    • Visualize knowledge-base
    • Efficient algorithms for category membership inference
  – Description logics
    • Formal language for constructing and combining category definitions
    • Efficient algorithms to decide subset and superset relationships between categories.
Semantic Networks

• Logic vs. semantic networks
• Many variations
  – All represent individual objects, categories of objects and relationships among objects.
• Allows for inheritance reasoning
  – Female persons inherit all properties from person.
  – Cfr. OO programming.
• Inference of inverse links
  – SisterOf vs. HasSister
Semantic network example
Semantic networks

- **Drawbacks**
  - Links can only assert binary relations
  - Can be resolved by reification of the proposition as an event

- **Representation of default values**
  - Enforced by the inheritance mechanism.
Description logics

- Are designed to describe definitions and properties about categories
  - A formalization of semantic networks
- Principal inference task is
  - Subsumption: checking if one category is the subset of another by comparing their definitions
  - Classification: checking whether an object belongs to a category.
  - Consistency: whether the category membership criteria are logically satisfiable.
Reasoning with Default Information

• “The following courses are offered: CSE101, CSE102, CSE150, ECE101”
  • Four (db)
    – Assume that this information is complete (not asserted ground atomic sentences are false)
    = CLOSED WORLD ASSUMPTION
    – Assume that distinct names refer to distinct objects
    = UNIQUE NAMES ASSUMPTION
  • FOL
    – Does not make these assumptions
    – Requires completion (aka “Clark Completion”
      » Course(c) ⇔ [c = CSE101 ∨ c = CSE101 ∨ …]
  • Prolog has these assumptions built in
Default logic

• Many of the inferences have default status rather than being absolutely certain
  – Inferred facts can be wrong and need to be retracted = BELIEF REVISION.
  – Assume KB contains sentence P and we want to execute TELL(KB, ¬P)
    • To avoid contradiction: RETRACT(KB,P)
    • But what about sentences inferred from P?
Planning

- The Planning problem
- Planning with State-space search
- Partial-order planning
- Planning graphs
- Planning with propositional logic
- Analysis of planning approaches
What is Planning

• Generate sequences of actions to perform tasks and achieve objectives.
  – States, actions and goals
• Search for solution over abstract space of plans.
• Classical planning environment: fully observable, deterministic, finite, static and discrete.
• Assists humans in practical applications
  – design and manufacturing
  – military operations
  – games
  – space exploration
Difficulty of real world problems

• Assume a problem-solving agent using some search method …
  – Which actions are relevant?
    • Exhaustive search vs. backward search
  – What is a good heuristic functions?
    • Good estimate of the cost of the state?
    • Problem-dependent vs, -independent
  – How to decompose the problem?
    • Most real-world problems are *nearly* decomposable.
Planning language

• What is a good language?
  – Expressive enough to describe a wide variety of problems.
  – Restrictive enough to allow efficient algorithms to operate on it.
  – Planning algorithm should be able to take advantage of the logical structure of the problem.

• STRIPS and ADL
General language features

- **Representation of states**
  - Decompose the world in logical conditions and represent a state as a *conjunction of positive literals*.
    - Propositional literals: $Poor \land Unknown$
    - FO-literals (grounded and function-free): $At(Plane1, Melbourne) \land At(Plane2, Sydney)$
  - Closed world assumption

- **Representation of goals**
  - Partially specified state and represented as a *conjunction of positive ground literals*
  - A goal is *satisfied* if the state contains all literals in goal.
General language features

- Representations of actions
  - Action = PRECOND + EFFECT
    
    \[
    \text{Action(Fly}(p, \text{from}, \text{to}),
    \]
    \[
    \text{PRECOND: } \text{At}(p, \text{from}) \land \text{Plane}(p) \land \text{Airport(from)} \land \text{Airport(to)}
    \]
    \[
    \text{EFFECT: } \neg \text{At}(p, \text{from}) \land \text{At}(p, \text{to})
    \]
    
    = action schema (p, from, to need to be instantiated)
    
    - Action name and parameter list
    - Precondition (conj. of function-free literals)
    - Effect (conj of function-free literals and P is True and not P is false)
    
    - Add-list vs delete-list in Effect
Language semantics?

• How do actions affect states?
  – An action is applicable in any state that satisfies the precondition.
  – For FO action schema applicability involves a substitution $\theta$ for the variables in the PRECOND.

  $\text{At}(P1,\text{JFK}) \land \text{At}(P2,\text{SFO}) \land \text{Plane}(P1) \land \text{Plane}(P2) \land \text{Airport}(\text{JFK}) \land \text{Airport}(\text{SFO})$

  Satisfies : $\text{At}(p,\text{from}) \land \text{Plane}(p) \land \text{Airport}(\text{from}) \land \text{Airport}(\text{to})$

  With $\theta = \{p/P1, \text{from}/\text{JFK}, \text{to}/\text{SFO}\}$

  Thus the action is applicable.
Language semantics?

• The result of executing action $a$ in state $s$ is the state $s'$
  
  – $s'$ is same as $s$ except
    
    • Any positive literal $P$ in the effect of $a$ is added to $s'$
    • Any negative literal $\neg P$ is removed from $s'$

  $EFFECT: \neg AT(p,from) \land AT(p,to):$
  
  $AT(P1,SFO) \land AT(P2,SFO) \land Plane(P1) \land Plane(P2) \land Airport(JFK) \land$
  $Airport(SFO)$

  – STRIPS assumption: (avoids representational frame problem)
    
    every literal NOT in the effect remains unchanged
Expressiveness and extensions

• STRIPS is simplified
  – Important limit: function-free literals
    • Allows for propositional representation
    • Function symbols lead to infinitely many states and actions

• Recent extension: Action Description language (ADL)
  \[
  \text{Action(Fly(p:Plane, from: Airport, to: Airport),}
  \]
  \[
  \text{PRECOND: At(p,from) } \land \ (\text{from } \neq \text{to})
  \]
  \[
  \text{EFFECT: } \neg \text{At(p,from) } \land \text{At(p,to))}
  \]

Standardization: Planning domain definition language (PDDL)
Example: air cargo transport

\textit{Init} (At(C1, SFO) \land At(C2, JFK) \land At(P1, SFO) \land At(P2, JFK) \land Cargo(C1) \land Cargo(C2) \land Plane(P1) \land Plane(P2) \land Airport(JFK) \land Airport(SFO))

\textit{Goal} (At(C1, JFK) \land At(C2, SFO))

\textit{Action}\textbf{(Load}(c, p, a) \\
\quad \textit{PRECOND}: \textit{At}(c, a) \land \textit{At}(p, a) \land \textit{Cargo}(c) \land \textit{Plane}(p) \land \textit{Airport}(a) \\
\quad \textit{EFFECT}: \neg \textit{At}(c, a) \land \neg \textit{In}(c, p))

\textit{Action}\textbf{(Unload}(c, p, a) \\
\quad \textit{PRECOND}: \textit{In}(c, p) \land \textit{At}(p, a) \land \textit{Cargo}(c) \land \textit{Plane}(p) \land \textit{Airport}(a) \\
\quad \textit{EFFECT}: \textit{At}(c, a) \land \neg \textit{In}(c, p))

\textit{Action}\textbf{(Fly}(p, \textit{from}, \textit{to}) \\
\quad \textit{PRECOND}: \textit{At}(p, \textit{from}) \land \textit{Plane}(p) \land \textit{Airport}(\textit{from}) \land \textit{Airport}(\textit{to}) \\
\quad \textit{EFFECT}: \neg \textit{At}(p, \textit{from}) \land \textit{At}(p, \textit{to}))

[\textit{Load}(C1, P1, SFO), \textit{Fly}(P1, SFO, JFK), \textit{Load}(C2, P2, JFK), \textit{Fly}(P2, JFK, SFO)]
Example: Spare tire problem

\[ \text{Init}(\text{At(Flat, Axle)} \land \text{At(Spare, trunk)}) \]
\[ \text{Goal}(\text{At(Spare, Axle)}) \]
\[ \text{Action}(\text{Remove(Spare, Trunk)} \]
\[ \text{PRECOND: } \text{At(Spare, Trunk)} \]
\[ \text{EFFECT: } \neg\text{At(Spare, Trunk)} \land \text{At(Spare, Ground)} \]
\[ \text{Action}(\text{Remove(Flat, Axle)} \]
\[ \text{PRECOND: } \text{At(Flat, Axle)} \]
\[ \text{EFFECT: } \neg\text{At(Flat, Axle)} \land \text{At(Flat, Ground)} \]
\[ \text{Action}(\text{PutOn(Spare, Axle)} \]
\[ \text{PRECOND: } \text{At(Spare, Ground)} \land \neg\text{At(Flat, Axle)} \]
\[ \text{EFFECT: } \text{At(Spare, Axle)} \land \neg\text{At(Spare, Ground)} \]
\[ \text{Action}(\text{LeaveOvernight} \]
\[ \text{PRECOND: } \]
\[ \text{EFFECT: } \neg\text{At(Spare, Ground)} \land \neg\text{At(Spare, Axle)} \land \neg\text{At(Spare, trunk)} \land \neg\text{At(Flat, Ground)} \land \neg\text{At(Flat, Axle)} \]

This example goes beyond STRIPS: negative literal in pre-condition (ADL description)
Example: Blocks world

*Init*(On(A, Table) ∧ On(B, Table) ∧ On(C, Table) ∧ Block(A) ∧ Block(B) ∧ Block(C) ∧ Clear(A) ∧ Clear(B) ∧ Clear(C))

*Goal*(On(A, B) ∧ On(B, C))

*Action*(Move(b, x, y))

  PRECOND: On(b, x) ∧ Clear(b) ∧ Clear(y) ∧ Block(b) ∧ (b ≠ x) ∧ (b ≠ y) ∧ (x ≠ y)

  EFFECT: On(b, y) ∧ Clear(x) ∧ ¬ On(b, x) ∧ ¬ Clear(y))

*Action*(MoveToTable(b, x))

  PRECOND: On(b, x) ∧ Clear(b) ∧ Block(b) ∧ (b ≠ x)

  EFFECT: On(b, Table) ∧ Clear(x) ∧ ¬ On(b, x))

Spurious actions are possible: Move(B, C, C)
Planning with state-space search

- Both forward and backward search possible
- Progression planners
  - forward state-space search
  - Consider the effect of all possible actions in a given state
- Regression planners
  - backward state-space search
  - To achieve a goal, what must have been true in the previous state.
Progression and regression

(a) At(P₁, A) At(P₂, A) Fly(P₁,A,B) At(P₁, B) At(P₂, A)
(b) At(P₁, A) At(P₂, B) Fly(P₁,A,B) At(P₁, B) At(P₂, B)
At(P₁, B) At(P₂, B)
At(P₂, A)
Progression algorithm

- Formulation as state-space search problem:
  - Initial state = initial state of the planning problem
    - Literals not appearing are false
  - Actions = those whose preconditions are satisfied
    - Add positive effects, delete negative
  - Goal test = does the state satisfy the goal
  - Step cost = each action costs 1
- No functions … any graph search that is complete is a complete planning algorithm.
  - E.g. A*
- Inefficient:
  - (1) irrelevant action problem
  - (2) good heuristic required for efficient search
Regression algorithm

• How to determine predecessors?
  – What are the states from which applying a given action leads to the goal?
    Goal state = $At(C1, B) \land At(C2, B) \land \ldots \land At(C20, B)$
    Relevant action for first conjunct: $Unload(C1,p,B)$
    Works only if pre-conditions are satisfied.
    Previous state= $\text{In}(C1, p) \land \text{At}(p, B) \land At(C2, B) \land \ldots \land At(C20, B)$
    Subgoal $\text{At}(C1,B)$ should not be present in this state.

• Actions must not undo desired literals (consistent)
• Main advantage: only relevant actions are considered.
  – Often much lower branching factor than forward search.
Regression algorithm

• General process for predecessor construction
  – Give a goal description G
  – Let A be an action that is relevant and consistent
  – The predecessors is as follows:
    • Any positive effects of A that appear in G are deleted.
    • Each precondition literal of A is added, unless it already appears.

• Any standard search algorithm can be added to perform the search.

• Termination when predecessor satisfied by initial state.
  – In FO case, satisfaction might require a substitution.
Heuristics for state-space search

• Neither progression or regression are very efficient without a good heuristic.
  – How many actions are needed to achieve the goal?
  – Exact solution is NP hard, find a good estimate

• Two approaches to find admissible heuristic:
  – The optimal solution to the relaxed problem.
    • Remove all preconditions from actions
  – The subgoal independence assumption:
    The cost of solving a conjunction of subgoals is approximated by the sum of the costs of solving the subproblems independently.
Partial-order planning

• Progression and regression planning are *totally ordered plan search* forms.
  – They cannot take advantage of problem decomposition.
    • Decisions must be made on how to sequence actions on all the subproblems

• Least commitment strategy:
  – Delay choice during search
Shoe example

Goal(RightShoeOn ∧ LeftShoeOn)
Init()
Action(RightShoe, PRECOND: RightSockOn
    EFFECT: RightShoeOn)
Action(RightSock, PRECOND:
    EFFECT: RightSockOn)
Action(LeftShoe, PRECOND: LeftSockOn
    EFFECT: LeftShoeOn)
Action(LeftSock, PRECOND:
    EFFECT: LeftSockOn)

Planner: combine two action sequences (1) leftsock, leftshoe (2) rightsock, rightshoe
Partial-order planning (POP)

- Any planning algorithm that can place two actions into a plan without which comes first is a PO plan.
POP as a search problem

- States are (mostly unfinished) plans.
  - The empty plan contains only start and finish actions.
- Each plan has 4 components:
  - A set of actions (steps of the plan)
  - A set of ordering constraints: A < B (A before B)
    - Cycles represent contradictions.
  - A set of causal links \( A \xrightarrow{p} B \)
    - The plan may not be extended by adding a new action C that conflicts with the causal link. (if the effect of C is \( \neg p \) and if C could come after A and before B)
  - A set of open preconditions.
    - If precondition is not achieved by action in the plan.
Example of final plan

- Actions={Rightsock, Rightshoe, Leftsock, Leftshoe, Start, Finish}
- Orderings={Rightsock < Rightshoe; Leftsock < Leftshoe}
- Links={Rightsock->Rightsockon -> Rightshoe, Leftsock->Leftsockon-> Leftshoe, Rightshoe->Rightshoeon->Finish, ...}
- Open preconditions={}
POP as a search problem

- A plan is *consistent* iff there are no cycles in the ordering constraints and no conflicts with the causal links.
- A consistent plan with no open preconditions is a *solution*.
- A partial order plan is executed by repeatedly choosing *any* of the possible next actions.
  - This flexibility is a benefit in non-cooperative environments.
Solving POP

• Assume propositional planning problems:
  – The initial plan contains Start and Finish, the ordering constraint Start < Finish, no causal links, all the preconditions in Finish are open.
  – Successor function:
    • picks one open precondition \( p \) on an action \( B \) and
    • generates a successor plan for every possible consistent way of choosing action \( A \) that achieves \( p \).
  – Test goal
Enforcing consistency

• When generating successor plan:
  – The causal link $A \rightarrow p \rightarrow B$ and the ordering constraint $A < B$ is added to the plan.
    • If $A$ is new also add $\text{start} < A$ and $A < B$ to the plan
  – Resolve conflicts between new causal link and all existing actions
  – Resolve conflicts between action $A$ (if new) and all existing causal links.
Process summary

• Operators on partial plans
  – Add link from existing plan to open precondition.
  – Add a step to fulfill an open condition.
  – Order one step w.r.t another to remove possible conflicts
• Gradually move from incomplete/vague plans to complete/correct plans
• Backtrack if an open condition is unachievable or if a conflict is irresolvable.
Example: Spare tire problem

\[
\begin{align*}
\text{Init} & : \text{At}(\text{Flat, Axle}) \land \text{At}(\text{Spare, trunk}) \\
\text{Goal} & : \text{At}(\text{Spare, Axle}) \\
\text{Action} (\text{Remove}(\text{Spare, Trunk}) & : \\
\quad \text{PRECOND: } & \text{At}(\text{Spare, Trunk}) \\
\quad \text{EFFECT: } & \neg \text{At}(\text{Spare, Trunk}) \land \text{At}(\text{Spare, Ground}) \\
\text{Action} (\text{Remove}(\text{Flat, Axle}) & : \\
\quad \text{PRECOND: } & \text{At}(\text{Flat, Axle}) \\
\quad \text{EFFECT: } & \neg \text{At}(\text{Flat, Axle}) \land \text{At}(\text{Flat, Ground}) \\
\text{Action} (\text{PutOn}(\text{Spare, Axle}) & : \\
\quad \text{PRECOND: } & \text{At}(\text{Spare, Groundp}) \land \neg \text{At}(\text{Flat, Axle}) \\
\quad \text{EFFECT: } & \text{At}(\text{Spare, Axle}) \land \neg \text{At}(\text{Spare, Ground}) \\
\text{Action} (\text{LeaveOvernight}) & : \\
\quad \text{PRECOND: } & \\
\quad \text{EFFECT: } & \neg \text{At}(\text{Spare, Ground}) \land \neg \text{At}(\text{Spare, Axle}) \land \neg \text{At}(\text{Spare, trunk}) \land \neg \text{At}(\text{Flat, Ground}) \land \neg \text{At}(\text{Flat, Axle}) \\
\end{align*}
\]
Solving the problem

- Initial plan: Start with EFFECTS and Finish with PRECOND.
Solving the problem

- Initial plan: Start with EFFECTS and Finish with PRECOND.
- Pick an open precondition: \textit{At(Spare, Axle)}
- Only \textit{PutOn(Spare, Axle)} is applicable
- Add causal link: \textit{PutOn(Spare, Axle)} \rightarrow \text{At(Spare, Axle)} \rightarrow \text{Finish}
- Add constraint: \textit{PutOn(Spare, Axle)} < \text{Finish}
Solving the problem

- Pick an open precondition: \( \text{At}(\text{Spare, Ground}) \)
- Only \( \text{Remove}(\text{Spare, Trunk}) \) is applicable
- Add causal link:
  \[
  \text{Remove}(\text{Spare,Trunk}) \xrightarrow{\text{At}(\text{Spare,Ground})} \text{PutOn}(\text{Spare,Axle})
  \]
- Add constraint: \( \text{Remove}(\text{Spare, Trunk}) < \text{PutOn}(\text{Spare, Axle}) \)
Solving the problem

- Pick an open precondition: \( \neg \text{At(Flat, Axle)} \)
- *LeaveOverNight* is applicable
- conflict: *LeaveOverNight* also has the effect \( \neg \text{At(Spare,Ground)} \)

\[
\text{Remove(Spare, Trunk)} \xrightarrow{\text{At(Spare,Ground)}} \text{PutOn(Spare, Axle)}
\]

- To resolve, add constraint: *LeaveOverNight* < Remove(Spare, Trunk)
Solving the problem

- Pick an open precondition: \( \text{At}(\text{Spare, Ground}) \)
- \( \text{LeaveOverNight} \) is applicable
- conflict: \( \text{Remove}(\text{Spare, Trunk}) \rightarrow \text{PutOn}(\text{Spare, Axle}) \)
- To resolve, add constraint: \( \text{LeaveOverNight} < \text{Remove}(\text{Spare, Trunk}) \)
- Add causal link:

\[
\text{LeaveOverNight} \rightarrow \neg \text{At}(\text{Spare, Ground}) \rightarrow \text{PutOn}(\text{Spare, Axle})
\]
Solving the problem

- Pick an open precondition: \( \text{At}(\text{Spare}, \text{Trunk}) \)
- Only \textit{Start} is applicable
- Add causal link: \( \text{Start} \xrightarrow{\text{At}(\text{Spare},\text{Trunk})} \text{Remove}(\text{Spare},\text{Trunk}) \)
- Conflict: of causal link with effect \( \text{At}(\text{Spare},\text{Trunk}) \) in \textit{LeaveOverNight}
  - No re-ordering solution possible.
- backtrack
Solving the problem

- Remove *LeaveOverNight*, *Remove*(Spare, Trunk) and causal links
- Repeat step with *Remove*(Spare, Trunk)
- Add also *RemoveFlatAxle* and finish
Some details …

- What happens when a first-order representation that includes variables is used?
  - Complicates the process of detecting and resolving conflicts.
  - Can be resolved by introducing inequality constraint.

- CSP’s most-constrained-variable constraint can be used for planning algorithms to select a PRECOND.
Planning graphs

• Used to achieve better heuristic estimates.
  – A solution can also directly extracted using GRAPHPLAN.
• Consists of a sequence of levels that correspond to time steps in the plan.
  – Level 0 is the initial state.
  – Each level consists of a set of literals and a set of actions.
    • \textit{Literals} = all those that \textit{could} be true at that time step, depending upon the actions executed at the preceding time step.
    • \textit{Actions} = all those actions that \textit{could} have their preconditions satisfied at that time step, depending on which of the literals actually hold.
Planning graphs

• “Could”?  
  – Records only a restricted subset of possible negative interactions among actions.

• They work only for propositional problems.

• Example:
  
  Init(Have(Cake))
  Goal(Have(Cake) ∧ Eaten(Cake))
  Action(Eat(Cake), PRECOND: Have(Cake)
  EFFECT: ¬Have(Cake) ∧ Eaten(Cake))
  Action(Bake(Cake), PRECOND: ¬Have(Cake)
  EFFECT: Have(Cake))
Cake example

- Start at level S0 and determine action level A0 and next level S1.
  - A0 >> all actions whose preconditions are satisfied in the previous level.
  - Connect precond and effect of actions S0 --> S1
  - Inaction is represented by persistence actions.
- Level A0 contains the actions that could occur
  - Conflicts between actions are represented by mutex links
Cake example

- Level S1 contains all literals that could result from picking any subset of actions in A0
  - Conflicts between literals that can not occur together (as a consequence of the selection action) are represented by mutex links.
  - S1 defines multiple states and the mutex links are the constraints that define this set of states.
- Continue until two consecutive levels are identical: *leveled off*
  - Or contain the same amount of literals (explanation follows later)
Cake example

- A mutex relation holds between **two actions** when:
  - *Inconsistent effects*: one action negates the effect of another.
  - *Interference*: one of the effects of one action is the negation of a precondition of the other.
  - *Competing needs*: one of the preconditions of one action is mutually exclusive with the precondition of the other.

- A mutex relation holds between **two literals** when (*inconsistent support)*:
  - If one is the negation of the other OR
  - If each possible action pair that could achieve the literals is mutex.
PG and heuristic estimation

- PG’s provide information about the problem
  - A literal that does not appear in the final level of the graph cannot be achieved by any plan.
    - Useful for backward search (cost = inf).
  - Level of appearance can be used as cost estimate of achieving any goal literals = level cost.
  - Small problem: several actions can occur
    - Restrict to one action using serial PG (add mutex links between every pair of actions, except persistence actions).
  - Cost of a conjunction of goals? Max-level, sum-level and set-level heuristics.

PG is a relaxed problem.
The GRAPHPLAN Algorithm

- How to extract a solution directly from the PG

```
function GRAPHPLAN(problem) return solution or failure

graph ← INITIAL-PLANNING-GRAPH(problem)
goals ← GOALS[problem]

loop do
  if goals all non-mutex in last level of graph then do
    solution ← EXTRACT-SOLUTION(graph, goals, LENGTH(graph))
    if solution ≠ failure then return solution
  else if NO-SOLUTION-POSSIBLE(graph) then return failure
  graph ← EXPAND-GRAPH(graph, problem)
```
Example: Spare tire problem

\[
\text{Init(At(Flat, Axle) \land At(Spare, trunk))}
\]
\[
\text{Goal(At(Spare, Axle))}
\]
\[
\text{Action(Remove(Spare, Trunk)}
\]
\[
\begin{align*}
\text{PRECOND: } & \text{At(Spare, Trunk)} \\
\text{EFFECT: } & \neg \text{At(Spare, Trunk)} \land \text{At(Spare, Ground)}
\end{align*}
\]
\[
\text{Action(Remove(Flat,Axle)}
\]
\[
\begin{align*}
\text{PRECOND: } & \text{At(Flat,Axle)} \\
\text{EFFECT: } & \neg \text{At(Flat,Axle)} \land \text{At(Flat,Ground)}
\end{align*}
\]
\[
\text{Action(PutOn(Spare,Axle)}
\]
\[
\begin{align*}
\text{PRECOND: } & \text{At(Spare,Groundp) \land \neg At(Flat,Axle)} \\
\text{EFFECT: } & \text{At(Spare,Axle) \land \neg At(Spare,Ground)}
\end{align*}
\]
\[
\text{Action(LeaveOvernight)}
\]
\[
\begin{align*}
\text{PRECOND: } & \\
\text{EFFECT: } & \neg \text{At(Spare,Ground)} \land \neg \text{At(Spare,Axle)} \land \neg \text{At(Spare, trunk)} \land \neg \text{At(Flat,Ground)} \land \neg \text{At(Flat,Axle)}
\end{align*}
\]

This example goes beyond STRIPS: negative literal in pre-condition (ADL description)
Initially the plan consist of 5 literals from the initial state and the CWA literals (S0).
Add actions whose preconditions are satisfied by EXPAND-GRAPH (A0)
Also add persistence actions and mutex relations.
Add the effects at level S1
Repeat until goal is in level S_i
GRAPHPLAN example

- EXPAND-GRAPH also looks for mutex relations
  - Inconsistent effects
    - E.g. Remove(Spare, Trunk) and LeaveOverNight due to At(Spare, Ground) and **not** At(Spare, Ground)
  - Interference
    - E.g. Remove(Flat, Axle) and LeaveOverNight At(Flat, Axle) as PRECOND and **not** At(Flat, Axle) as EFFECT
  - Competing needs
    - E.g. PutOn(Spare,Axle) and Remove(Flat, Axle) due to At(Flat,Axle) and **not** At(Flat, Axle)
  - Inconsistent support
    - E.g. in S2, At(Spare,Axle) and At(Flat,Axle)
• In S2, the goal literals exist and are not mutex with any other
  – Solution might exist and EXTRACT-SOLUTION will try to find it
• EXTRACT-SOLUTION can use Boolean CSP to solve the problem or a search process:
  – Initial state = last level of PG and goal goals of planning problem
  – Actions = select any set of non-conflicting actions that cover the goals in the state
  – Goal = reach level S0 such that all goals are satisfied
  – Cost = 1 for each action.
GRAPHPLAN example

- Termination? YES
- PG are monotonically increasing or decreasing:
  - Literals increase monotonically
  - Actions increase monotonically
  - Mutexes decrease monotonically
- Because of these properties and because there is a finite number of actions and literals, every PG will eventually level off!
Planning with propositional logic

- Planning can be done by proving theorem in situation calculus.
- Here: test the *satisfiability* of a logical sentence:

\[ \text{initial state} \land \text{all possible action descriptions} \land \text{goal} \]

- Sentence contains propositions for every action occurrence.
  - A model will assign true to the actions that are part of the correct plan and false to the others
  - An assignment that corresponds to an incorrect plan will not be a model because of inconsistency with the assertion that the goal is true.
  - If the planning is unsolvable the sentence will be unsatisfiable.
function SATPLAN(problem, $T_{max}$) return solution or failure
  inputs: problem, a planning problem
            $T_{max}$, an upper limit to the plan length
  for $T = 0$ to $T_{max}$ do
    cnf, mapping ← TRANSLATE-TO_SAT(problem, $T$)
    assignment ← SAT-SOLVER(cnf)
    if assignment is not null then
      return EXTRACT-SOLUTION(assignment, mapping)
  return failure
\[ \text{cnf, mapping} \leftarrow \text{TRANSLATE-TO_SAT}(\text{problem}, T) \]

- Distinct propositions for assertions about each time step.
  - Superscripts denote the time step
    \[ At(P_1, SFO) \wedge At(P_2, JFK) \]
  - No CWA thus specify which propositions are not true
    \[ \neg At(P_1, SFO) \wedge \neg At(P_2, JFK) \]
  - Unknown propositions are left unspecified.
- The goal is associated with a particular time-step
  - But which one?
cnf, mapping ← TRANSLATE-TO_SAT(problem, T)

• How to determine the time step where the goal will be reached?
  – Start at $T=0$
    • Assert $At(P1,SFO)^0 \land At(P2,JFK)^0$
  – Failure .. Try $T=1$
    • Assert $At(P1,SFO)^1 \land At(P2,JFK)^1$
  – ...
  – Repeat this until some minimal path length is reached.
  – Termination is ensured by $T_{max}$
cnf, mapping ← TRANSLATE-TO_SAT(problem, T)

• How to encode actions into PL?
  – Propositional versions of successor-state axioms
    \[ \text{At}(P1,JFK)^t \iff \]
    \[
    (\text{At}(P1,JFK)^0 \land \neg(\text{Fly}(P1,JFK,SFO)^0 \land \text{At}(P1,JFK)^0)) \lor \\
    (\text{Fly}(P1,SFO,JFK)^0 \land \text{At}(P1,SFO)^0)
    \]
  – Such an axiom is required for each plane, airport and time step
  – If more airports add another way to travel than additional disjuncts are required
• Once all these axioms are in place, the satisfiability algorithm can start to find a plan.
assignment ← SAT-SOLVER(cnf)

• Multiple models can be found
• They are NOT satisfactory: (for T=1)
  \[ \text{Fly}(P1,SFO,JFK)^0 \land \text{Fly}(P1,JFK,SFO)^0 \land \text{Fly}(P2,JFK,SFO)^0 \]
  The second action is infeasible
  Yet the plan IS a model of the sentence

\[ \text{initial state} \land \text{all possible action descriptions} \land \text{goal}^1 \]
• Avoiding illegal actions: pre-condition axioms
  \[ \text{Fly}(P1,SFO,JFK)^0 \Rightarrow \text{At}(P1,JFK) \]
• Exactly one model now satisfies all the axioms where the goal is achieved at \( T=1 \).
assignment ← SAT-SOLVER(cnf)

- A plane can fly at two destinations at once
- They are NOT satisfactory: (for T=1)
  \[ \text{Fly}(P1, SFO, JFK)^0 \land \text{Fly}(P2, JFK, SFO)^0 \land \text{Fly}(P2, JFK, LAX)^0 \]
  The second action is infeasible
  Yet the plan allows spurious relations

- Avoid spurious solutions: action-exclusion axioms
  \[ \neg(\text{Fly}(P2, JFK, SFO)^0 \land \text{Fly}(P2, JFK, LAX)) \]
  Prevents simultaneous actions

- Lost of flexibility since plan becomes totally ordered: no actions are allowed to occur at the same time.
  - Restrict exclusion to preconditions
Analysis of planning approach

• Planning is an area of great interest within AI
  – Search for solution
  – Constructively prove a existence of solution

• Biggest problem is the combinatorial explosion in states.

• Efficient methods are under research
  – E.g. divide-and-conquer