# Artificial Intelligence II

**Dr Sean Holden**

Computer Laboratory, Room FC06

Telephone extension 63725

Email: sbh11@cl.cam.ac.uk

http://www.cl.cam.ac.uk/users/sbh11/

## Syllabus

Fun new things to be looked at include some more advanced material based on the **logical** approach to AI:

- **Further Symbolic Knowledge Representation:** Representing knowledge using First Order Logic (FOL). The situation calculus. [1 lecture]
- **Further Planning:** Incorporating heuristics into partial-order planning. Planning graphs. The GRAPHPLAN algorithm. Planning using propositional logic. [2 lectures]

(Note: there is no warranty attached to the stated lecture timings!)

## Syllabus

We then delve into some more modern material which takes account of **uncertainty**.

- **Uncertainty and Bayesian Networks:** Review of probability as applied to AI. Bayesian networks. Inference in Bayesian networks using both exact and approximate techniques. Other ways of dealing with uncertainty. [3 lectures]
- **Utility and Decision-making:** Maximising expected utility, decision networks, the value of information. [1 lecture]
- **Uncertain Reasoning Over Time:** Markov processes, transition and sensor models. Inference in temporal models: filtering, prediction, smoothing and finding the most likely explanation. Hidden Markov models. [2 lectures]

## Syllabus

Finally, we look in more depth at **Machine Learning:**

- **Further Supervised Learning:** Bayes theorem as applied to supervised learning. The maximum likelihood and maximum a posteriori hypotheses. Applying the Bayesian approach to neural networks. [4 lectures]
- **Reinforcement Learning:** Learning from rewards and punishments. [2 lectures]

## Books

Once again, the main single text book for the course is:

- *Artificial Intelligence: A Modern Approach*. Stuart Russell and Peter Norvig, Second edition (important!), Prentice Hall, 2002.

There is an accompanying web site at

`http://aima.cs.berkeley.edu/`

## Books

For some of the new material on neural networks you might also like to take a look at:

- *Neural Networks for Pattern Recognition*. Christopher M. Bishop, Oxford University Press, 1995.

And for some of the new material on reinforcement learning you might like to consult:

- *Machine Learning*. Tom Mitchell. McGraw Hill, 1997.

## Dire Warning

!!!DIRE WARNING!!!

This course contains large dollops of:

1. Probability
2. Matrix algebra
3. Calculus.

As I am an evil and vindictive person I will assume that you know everything on these subjects that was covered in earlier courses.

If you don't it is essential that you re-visit your old notes and make sure that you're at home with that material.

YOU HAVE BEEN WARNED. (Cue evil laughter *etc*.)

## How's your maths?

To see if you're up to speed on the maths, have a go at the following:

1. Evaluate the integral
$$\int_{-\infty}^{\infty} \exp(-x^2)\, dx$$
(Hint: square it and change to polar coordinates.)
2. Following on from $1$, evaluate the integral
$$\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \exp\left(-\frac{1}{2}\left(\mathbf{x}^T\mathbf{X}\mathbf{x} + \mathbf{x}^T\mathbf{y} + z\right)\right) dx_1 \cdots dx_p$$
where $\mathbf{X}$ is a symmetric $p \times p$ matrix with real elements, $\mathbf{y} \in \mathbb{R}^p$, $z \in \mathbb{R}$ and
$$\mathbf{x}^T = \left[\begin{array}{cccc} x_1 & x_2 & \cdots & x_p \end{array}\right] \in \mathbb{R}^p$$

(The second one is a bit tricky. Don't worry, I'll show you the answers somewhere around lecture 10.)

## Knowledge representation and reasoning I: FOL

We now look at how an agent might **represent** knowledge about its environment using first order logic (FOL), and **reason** with this knowledge to achieve its goals.

**Aims:**

- to show how FOL can be used to represent knowledge about a simple environment in the form of both background knowledge and knowledge derived from percepts;
- to show how this knowledge can be used to derive non-perceived knowledge about the environment;
- to introduce the situation calculus and demonstrate its application in a simple environment.

## Interesting reading

**Reading:** Russell and Norvig, chapters 7 to 10.

Knowledge representation based on logic is a vast subject and can't be covered in full in the lectures.

In particular:

- techniques for representing further kinds of knowledge;
- techniques for moving beyond the idea of a situation;
- reasoning systems based on categories;
- reasoning systems using default information;
- truth maintenance systems.

Happy reading :-)

## Knowledge representation and reasoning

A quick look at the world tell us that simply reacting to the environment is a very poor (non-“intelligent”) way of behaving.

All of us—and all intelligent agents—should use logical reasoning to help us interact successfully with the world.

Any intelligent agent should:

- possess knowledge about the environment and about how its actions affect the environment;
- use some form of logical reasoning to maintain its knowledge as percepts arrive;
- use some form of logical reasoning to deduce actions to perform in order to achieve goals.

## Knowledge representation and reasoning

This leads to a modified format for a potential intelligent agent:

- it can accept tasks to perform in the form of goals;
- it can be told about its environment;
- it can learn about its environment;
- it can update its knowledge to adapt to changes in its environment.

## Knowledge representation and reasoning

This shift in our viewpoint hides considerable subtlety.

- How do we describe the current state of the world?
- How do we infer from our percepts, knowledge of unseen parts of the world?
- How does the world change as time passes?
- How does the world stay the same as time passes?
- How do we know what we actually want to achieve?
- How do we know the effects of our actions?

## Knowledge-based agent design

Central to our new approach will be the concept of a Knowledge Base, generally abbreviated to KB .

- A KB is a collection of sentences, each of which represents a fact about the world.
- In order to represent facts we need a knowledge representation language.

## Knowledge-based agent design

Once we have a KB , we also need to be able to interact with it. There are two standard ways of doing this: `ask` and `tell`.

- `tell` adds new sentences to the KB .
- `ask` performs inference. It should produce a result that logically follows from what has been told to the KB .

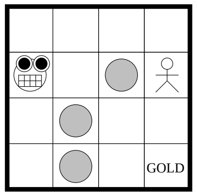
`ask` and `tell` hide, for the time being, an inference algorithm.

A KB can initially contain background knowledge about the world.

An important thing is missing from this description: namely, intelligent beings such as ourselves can also modify our KB by learning.

## Wumpus world

As a simple test scenario for a knowledge-based agent we will make use of the Wumpus World.



The Wumpus World is a 4 by 4 grid-based cave. Our intrepid hero wants to enter the cave, find some gold, and get out again unscathed.

## Wumpus world

The rules of Wumpus World.

- Unfortunately the cave contains a number of pits, which our hero can fall into and die. (You don't want to know what lives in the pits. It will give you nightmares.)
- The cave also contains the Wumpus, who will eat him. (There are few known vegetarian Wumpuses.)
- The Wumpus, having munched too many heroes, is too fat to fall into a pit.

## Wumpus world

Our hero can move around the cave at will and can perceive the following:

- In a position adjacent to the Wumpus, a stench is perceived. (Wumpuses are famed for their lack of personal hygiene.)
- In a position adjacent to a pit, a breeze is perceived. (Like I said: just don't ask what lives in the pits.)
- In the position where the gold is, a glitter is perceived.
- On trying to move into a wall, a bump is perceived.
- On killing the Wumpus a scream is perceived.

In addition, our hero has a single arrow, with which to try to kill the Wumpus.

"Adjacent" in the following does not include diagonals.

## Wumpus world

So we have:

**Percepts:** stench, breeze, glitter, bump, scream

**Actions:** forward, rotate left, rotate right, grab, shoot, climb

Of course, our aim now is not just to design an agent that can perform well in a single cave layout.

We want to design an agent that can usually perform well regardless of the layout of the cave.

## Logic for knowledge representation

What are the requirements for a knowledge representation language?

- It should be expressive.
- It should be concise.
- It should be unambiguous.
- It should be independent of context.
- It should have an inference procedure.

Our choice of language tends to lead to two important commitments:

- Ontological commitments: what does the world consist of?
- Epistemological commitments: what are the allowable states of knowledge?

## Logic for knowledge representation

A knowledge representation requires:

- syntax - what are the allowable patterns of symbols used to represent facts as sentences?
- semantics - what facts about the world do the sentences describe.

We say that an agent believes the facts described by the sentences. (Warning: “belief” has a minefield all of its own awaiting...)

## Logic for knowledge representation

When both syntax and semantics are defined in a sufficiently precise manner we have a logic.

Our central aim is to generate sentences that are true, IF the sentences in the KB are true. This process is based on concepts familiar from your introductory logic courses:

- Entailment: KB $\models \alpha$ means that the KB entails $\alpha$.
- Proof: KB $\vdash_i \alpha$ means that $\alpha$ is derived from the KB using $i$. If $i$ is sound then we have a proof.
- $i$ is sound if it can generate only entailed $\alpha$.
- $i$ is complete if it can find a proof for any entailed $\alpha$.

## Logic for knowledge representation

Our central strategy is to ask whether the sentence

$$\text{KB} \rightarrow \alpha$$

is valid. This makes sense, because the computer has no idea what your interpretation is for the symbols used to make sentences.

## Representing and reasoning with first-order logic (FOL)

Propositional logic is useful for introducing some fundamental ideas, but its ontological commitment—that the world consists of facts—makes it far too limited for further use.

FOL has a different ontological commitment—the world consists of:

- facts;
- objects (with properties distinguishing them from other objects), and;
- relations that hold between objects (some of which can be functions).

## Representing and reasoning with first-order logic (FOL)

FOL has no ontological commitment to:

- time;
- events;
- categories *etc*.

This is part of the reason for its success: there are too many ways of dealing with such concepts and you don't want to fix a single one. FOL can do it in a way matched to the environment.

FOL can express anything at all that can be programmed. Subsets have limited application while logics with extras are debated but not fully understood.

## Representing and reasoning with first-order logic (FOL)

The precise definitions for $\forall$, $\exists$, and $\wedge$ and $\rightarrow$ require that we're a little careful.

If we try to express "all wombats are hairy" as

$$\forall x\ (\texttt{Wombat}(x) \wedge \texttt{Hairy}(x))$$

we are essentially saying

$$\texttt{Wombat}(x_1) \wedge \texttt{Hairy}(x_1) \wedge \texttt{Wombat}(x_2) \wedge \texttt{Hairy}(x_2) \wedge \cdots$$

This is a very strong statement: it says that everything is a hairy wombat. This is almost certainly not what we want to express.

Much better is

$$\forall x\ (\texttt{Wombat}(x) \rightarrow \texttt{Hairy}(x))$$

This works, because $A \rightarrow B$ is always true when $A$ is false.

## Representing and reasoning with first-order logic (FOL)

Similarly, if you try to express "there is a hairy wombat" as

$$\exists x\ (\texttt{Wombat}(x) \rightarrow \texttt{Hairy}(x))$$

you are effectively saying

$$(\texttt{Wombat}(x_1) \rightarrow \texttt{Hairy}(x_1)) \vee (\texttt{Wombat}(x_2) \rightarrow \texttt{Hairy}(x_2)) \vee \cdots$$

and as $A \rightarrow B$ is true when $A$ is false the overall expression is true if anything at all is not a wombat. Here the correct expression is

$$\exists x\ (\texttt{Wombat}(x) \wedge \texttt{Hairy}(x))$$

## Using FOL in AI: `ask` and `tell`

To add a sentence $\alpha$ to our KB we use the notation

$$\texttt{tell}(\text{KB}, \alpha)$$

This is often referred to as an assertion. For example

$$\texttt{tell}(\text{KB}, \forall x, y\ (\texttt{wife}(x,y) \iff \texttt{female}(x) \wedge \texttt{married}(x,y)))$$
$$\texttt{tell}(\text{KB}, \texttt{female}(\texttt{Violet}))$$
$$\texttt{tell}(\text{KB}, \texttt{married}(\texttt{Violet}, \texttt{Bill}))$$

To perform inference we `ask` a question. For example

$$\texttt{ask}(\text{KB}, \texttt{wife}(\texttt{Violet}, \texttt{Bill}))$$

should result in an indication that, yes, Violet is indeed the wife of Bill. This is often known as a query or goal.

## Using FOL in AI: `ask` and `tell`

More usually we `ask` a question that involves an $\exists$. For example

$$\texttt{ask}(\texttt{KB}, \exists a\ \texttt{married}(\texttt{Violet}, a))$$

and expect the answer to include a substitution or binding list of the form

$$\{a/\texttt{Bill}\}$$

or several if appropriate.

## Using FOL in AI: the triumphant return of the Wumpus

Armed with FOL, the design of an agent for the Wumpus world is feasible.

There are at least three potential approaches:

- a reflex agent, which acts directly on the basis of its percepts;
- a model-based agent, which constructs and applies a model of the world;
- a goal-based agent, which formulates goals and attempts to achieve them.

Representing time using the integers, a percept is for example

$$\texttt{Percept}([\texttt{None}, \texttt{Breeze}, \texttt{None}, \texttt{None}, \texttt{None}], 1)$$

which indicates that only a breeze is perceived at time $1$.

## Using FOL in AI: the triumphant return of the Wumpus

The percept list is: `Stench`, `Breeze`, `Glitter`, `Bump`, `Scream`.

The actions are: `Forward`, `Shoot`, `Grab`, `Climb`, `Release`, `Turn(right)`, `Turn(left)`.

To find an action we make a query

$$\texttt{ask}(\texttt{KB}, \exists x\ \texttt{Action}(x, 1))$$

to obtain a substitution such as

$$\{x/\texttt{Forward}\}$$

## The triumphant return of the Wumpus: reflex agent

It is straightforward to devise background knowledge that can convert percepts to actions

$$\forall x, y, z, a, b\ \texttt{Percept}([x, y, \texttt{Glitter}, z, a], b) \rightarrow \texttt{Action}(\texttt{Grab}, b)$$

and we can if necessary introduce slightly more sophisticated rules for perception

$$\forall x, y, z, a, b\ \texttt{Percept}([\texttt{Stench}, x, y, z, a], b) \rightarrow \texttt{Stench}(b)$$
$$\forall x, y, z, a, b\ \texttt{Percept}([x, \texttt{Breeze}, y, z, a], b) \rightarrow \texttt{ByPit}(b)$$
$$\forall x, y, z, a, b\ \texttt{Percept}([x, y, \texttt{Glitter}, z, a], b) \rightarrow \texttt{FoundGold}(b)$$
$$\vdots$$

followed by further rules for actions such as

$$\forall x\ \texttt{FoundGold}(x) \rightarrow \texttt{Action(Grab,x)}$$

BUT, such agents are not very impressive.

Even in this simple world it's hard to see how to make a successful one, as there is no representation of the world.

## FOL - representing the world

Let's assume all percepts are added to the KB.

- A percept history is not useful in itself.
- For example, I know I have socks on and I don't need to search today's percept history to come to that conclusion - it exists in my representation of the world.
- A Wumpus world agent should not need to search its percept history to know it has no arrow - this should be part of its world representation.

Any system of rules based on past percepts can alternatively be implemented using a representation of the world, provided this representation is updated in response to percepts and actions.

Diachronic rules describe how the world changes, or stays the same.

## Situation calculus

Ideally we want to be able to speculate freely about the past and about possible futures. The need to be able to compare different situations means that something better than search is needed.

**Solution:**

- include situations in the logical language used by our KB;
- include axioms in our KB that relate to situations.

The oldest and simplest approach to this idea is situation calculus.

## Situation calculus

In **situation calculus**:

- the world consists of sequences of situations;
- over time, an agent moves from one situation to another;
- situations are changed as a result of actions;
- a situation argument is added to the predicate of any relation or property that can change over time. For example

  $$\texttt{At}(\texttt{Wumpus}, [2,4], S_2) \wedge \texttt{At}(\texttt{Agent}, [2,3], S_5)$$

  suggests our hero might nearly have popped his clogs.
- things that can change over time are sometimes called fluents;
- a situation argument is not needed for things that don't change. These are sometimes referred to as eternal or atemporal.

## Representing change as a result of actions

Situation calculus uses a function

$$\texttt{Result}(\texttt{action}, \texttt{situation})$$

to denote the new situation arising as a result of performing the specified action in the specified situation.

$$\begin{aligned}
&\texttt{Result}(\texttt{Grab}, S_1) = S_2\\
&\texttt{Result}(\texttt{Turn(left)}, S_2) = S_3\\
&\texttt{Result}(\texttt{Shoot}, S_3) = S_4\\
&\texttt{Result}(\texttt{Forward}, S_4) = S_5\\
&\qquad\vdots
\end{aligned}$$

## Effects of actions

Given that an action results in a new situation, we then specify the properties of the new situation.

For example, to keep track of whether our hero has the gold we need effect axioms to describe the effect of picking it up:

$\texttt{Portable}(\texttt{Gold})$
$\forall s\ \texttt{SeeGold}(s) \rightarrow \texttt{Available}(\texttt{Gold}, s)$
$\forall x, s\ \texttt{Available}(x, s) \wedge \texttt{Portable}(x) \rightarrow \texttt{Have}(x, \texttt{Result}(\texttt{Grab}, s))$

Effect axioms describe the way in which the world changes.

## Effects of actions

We need frame axioms to describe the effect of having something but not dropping it:

$$\forall a, x, s\ \texttt{Have}(x, s) \wedge (a \neq \texttt{Release}) \rightarrow \texttt{Have}(x, \texttt{Result}(a, s))$$

or of not having something and not picking it up:

$$\forall a, x, s\ \neg\texttt{Have}(x, s) \wedge (a \neq \texttt{Grab} \vee \neg(\texttt{Available}(x, s) \wedge (\texttt{Portable}(x)$$
$$\rightarrow \neg\texttt{Have}(x, \texttt{Result}(a, s))$$

Frame axioms describe the way in which the world stays the same.

## The Frame problem, and other problems

The frame problem has historically been associated with situation calculus.

**Representational frame problem:** a large number of frame axioms are required to represent the many things in the world which will not change as the result of an action.

We will see how to solve this in a moment.

**Inferential frame problem:** when reasoning about a sequence of situations within the situation calculus, all the unchanged properties still need to be carried through all the steps.

This can be alleviated using planning systems that allow us to reason efficiently when actions change only a small part of the world.

## The Frame problem, and other problems

**Qualification problem:** we are in general never completely certain what conditions are required for an action to be effective.

Consider for example turning the key to start your car.

This will lead to problems if important conditions are omitted from axioms.

**Ramification problem:** actions tend to have implicit consequences that are large in number.

For example, if I pick up a sandwich in a dodgy sandwich shop, I will also be picking up all the bugs that live in it. I don't want to model this explicitly.

## Successor-state axioms

Effect axioms and frame axioms can be combined into successor-state axioms.

One is needed for each predicate that can change over time.

```
true in new situation ⟺ you did something to
                          make it true
                        ∨ it was already true and
                          and you didn't make it
                          false
```

For example

$$\begin{aligned}\forall a, x, s\ \text{Have}(x, \text{Result}(a, s)) \iff & (\mathtt{Portable}(x)\\ & \wedge \mathtt{Available}(x, s)\\ & \wedge a = \mathtt{Grab})\\ & \vee (\mathtt{Have}(x, s) \wedge a \neq \mathtt{Release})\end{aligned}$$

## Knowing where you are

If $S_0$ is the initial situation we know that

$$\mathtt{At}(\mathtt{Agent}, [1, 1], S_0)$$

We need to keep track of what way we're facing. Say east is $0$, north is $90$, west is $180$ and south is $270$.

$$\mathtt{Facing}(\mathtt{Agent}, S_0) = 0$$

We need to know how motion affects location

$$\forall x, y\ \mathtt{ForwardResult}([x, y], 0) = [x + 1, y]$$
$$\forall x, y\ \mathtt{ForwardResult}([x, y], 270) = [x, y - 1]$$
$$\vdots$$

and

$$\forall l, a, s\ \mathtt{At}(a, l, s) \rightarrow \mathtt{GoForward}(a, s) = \mathtt{ForwardResult}(l, \mathtt{Facing}(a,$$

The concept of adjacency is very important in the Wumpus world

$$\forall l_1, l_2\ \mathtt{Adjacent}(l_1, l_2) \iff \exists d\ \mathtt{ForwardResult}(l_1, d) = l_2$$

## Knowing where you are

We also know that the cave is $4$ by $4$ and surrounded by walls

$$\forall x, y\ \mathtt{WallHere}([x, y]) \iff (x = 0 \vee y = 0 \vee x = 5 \vee y = 5)$$

It is only possible to change location by moving, and this only works if you're not facing a wall. We need a successor-state axiom

$$\begin{aligned}\forall a, b, s, l\ \mathtt{At}(b, l, \mathtt{Result}(a, s)) \iff & (l = \mathtt{GoForward}(b, s)\\ & \wedge a = \mathtt{Forward}\\ & \wedge \neg \mathtt{WallHere}(l))\\ & \vee (\mathtt{At}(b, l, s) \wedge a \neq \mathtt{Forward})\end{aligned}$$

It is only possible to change orientation by turning. Again, we need a successor-state axiom

$$\begin{aligned}&\forall b, a, s, o\ \mathtt{Facing}(b, \mathtt{Result}(a, s)) = o \iff\\ &\quad a = \mathtt{Turn(right)} \wedge o = \mathtt{Mod}(\mathtt{Facing}(b, s) - 90, 360)\\ &\quad \vee a = \mathtt{Turn(left)} \wedge o = \mathtt{Mod}(\mathtt{Facing}(b, s) + 90, 360)\\ &\quad \vee (\mathtt{Facing}(b, s) = o \wedge a \neq \mathtt{Turn(right)} \wedge a \neq \mathtt{Turn(left)})\end{aligned}$$

and so on...

## Deducing other properties of the world

If you know where you are, then you can think about places rather than just situations.

$$\forall l, s\ \mathtt{At}(\mathtt{Agent}, l, s) \wedge \mathtt{Breeze}(s) \rightarrow \mathtt{BreezeAt}(l)$$
$$\forall l, s\ \mathtt{At}(\mathtt{Agent}, l, s) \wedge \mathtt{Stench}(s) \rightarrow \mathtt{StenchAt}(l)$$

This is important.

Different actions will lead to different situations, but what we want to know is where breezes, stenches *etc* are so we can deduce properties such as where the pits are, where it's safe to move, and so on.

Synchronic rules relate properties shared by a single state of the world.

There are two kinds: causal and diagnostic.

## Deducing other properties of the world

Causal rules: some properties of the world will produce percepts.

$$\forall s, l_1, l_2\ \texttt{At}(\texttt{Wumpus}, l_1, s) \wedge \texttt{Adjacent}(l_1, l_2) \rightarrow \texttt{StenchAt}(l_2)$$

$$\forall s, l_1, l_2\ \texttt{At}(\texttt{Pit}, l_1, s) \wedge \texttt{Adjacent}(l_1, l_2) \rightarrow \texttt{BreezeAt}(l_2)$$

Systems reasoning with such rules are known as model-based reasoning systems.

## Deducing other properties of the world

Diagnostic rules: infer properties of the world from percepts.

For example, we have already seen

$$\forall l, s\ \texttt{At}(\texttt{Agent}, l, s) \wedge \texttt{Breeze}(s) \rightarrow \texttt{BreezeAt}(l)$$

$$\forall l, s\ \texttt{At}(\texttt{Agent}, l, s) \wedge \texttt{Stench}(s) \rightarrow \texttt{StenchAt}(l)$$

These may not be very strong.

For example, knowing that the absence of a smell or breeze means adjacent squares are safe is unlikely to be as powerful as knowing that the absence of a Wumpus or a pit means a square is safe.

(Squares can be safe even when there are smells and breezes!)

The difference between model-based and diagnostic reasoning can be important. For example, medical diagnosis can be done based on symptoms or based on a model of disease.

## Preferring some actions to others

In general we need to know the relative desirability of different actions.

- For example, it is initially good to explore, but having found the gold it is better to pick it up.
- As usual, we want to maintain modularity. In this context: we don't want a change in an agent's beliefs about one thing to require a change in the rules governing others.
- We should therefore separate facts about goals from facts about actions.

## Preferring some actions to others

For example, actions can be ranked: `Great`, `Good`, `Medium`, `Risky`, `Deadly`

$$\forall a, s\ \texttt{Great}(a, s) \rightarrow \texttt{Action}(a, s)$$
$$\forall a, s\ \texttt{Good}(a, s) \wedge \neg\exists a_2\ \texttt{Great}(a_2, s) \rightarrow \texttt{Action}(a, s)$$
$$\forall a, s\ \texttt{Medium}(a, s) \wedge \neg\exists a_2\ (\texttt{Good}(a_2, s) \vee \texttt{Great}(a_2, s)) \rightarrow \texttt{Actions}(a, s)$$
$$\vdots$$

This produces an action-value system.

## Preferring some actions to others

We now need to describe the action values. For example:

- great actions are picking up the gold or climbing out of the cave after it's found.
- good actions are...

Also, as our priorities change once the gold has been found we need to change our agent's goal:

$$\forall s\ \texttt{Have}(\texttt{Gold}, s) \rightarrow \texttt{Goal}([1,1], s)$$

## Preferring some actions to others

Once a goal is defined there are at least three ways to achieve it:

- inference: `ask` the KB for a sequence of actions. (Computational complexity!)
- search: best first search. This requires us to change the KB into operators and a state representation.
- planning.

## Problems

Assume the existence of a constant `nil` denoting the empty list and and function `cons` as well as the usual shorthand: $[]$ for the empty list, $[a]$ for the list containing only $a$, $[a, b]$ for the list containing $a$ and $b$, $[a|l]$ for the list obtained by adding $a$ to the list $l$, and so on.

1. Write the definition of a FOL function `Action_sequence` that can be used in conjunction with the situation calculus as follows

   $$s_{\text{result}} = \texttt{Action\_sequence}([a_1, a_2, \ldots, a_n], s_{\text{start}})$$

   to map a starting situation $s_{\text{start}}$ and a list $[a_1, a_2, \ldots, a_n]$ of actions to the situation $s_{\text{result}}$ that results if the given list of actions is carried out beginning in the specified situation.
2. Explain how this function might be used in conjunction with `ask` to ask the knowledge base for a sequence of actions required to achieve a goal, rather than just a single action to try at a given time.

## Problems

Making correct use of the situation calculus, write the sentences in FOL required to implement the following actions in Wumpus World:

1. `Climb`
2. `Shoot`

## Problems

Paper 9, Question 8, 2003 - part 2

You wish to construct a robotic pet cat for the purposes of entertainment. One purpose of the cat is to scratch valuable objects when the owner is not present. Give a brief general description of *situation calculus* and describe how it might be used for knowledge representation by the robot. Include in your answer one example each of a *frame axiom*, an *effect axiom*, and a *successor-state axiom*, along with example definitions of suitable predicates and functions. [12 marks]

## Planning II: heuristics, planning graphs, and planning with propositional

We now examine:

- the way in which basic heuristics might be defined for use in planning problems;
- the construction of planning graphs and their use in obtaining more sensible heuristics;
- planning graphs as the basis of the GraphPlan algorithm;
- planning using propositional logic.

**Reading:** Russell and Norvig, chapter 11, sections 11.3, 11.4 and 11.5.

## Planning so far: a brief review

We used the following simple example problem.

The intrepid little scamps in the Cambridge University Roof-Climbing Society wish to attach an inflatable gorilla to the spire of a famous College. To do this they need to leave home and obtain:

- An inflatable gorilla: these can be purchased from all good joke shops.
- Some rope: available from a hardware store.
- A first-aid kit: also available from a hardware store.

They need to return home after they've finished their shopping.

How do they go about planning their jolly escapade?

## The STRIPS language

STRIPS: “Stanford Research Institute Problem Solver” (1970).

**States:** are conjunctions of ground literals with no functions.

$$\begin{aligned}\texttt{At}(\texttt{Home}) &\wedge \neg\texttt{Have}(\texttt{Gorilla})\\ &\wedge \neg\texttt{Have}(\texttt{Rope})\\ &\wedge \neg\texttt{Have}(\texttt{Kit})\end{aligned}$$

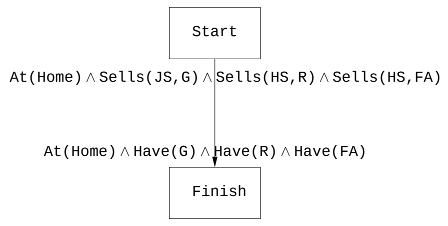
**Goals:** are conjunctions of literals where variables are assumed existentially quantified.

$$\texttt{At}(x) \wedge \texttt{Sells}(x, \texttt{Gorilla})$$

A planner finds a sequence of actions that makes the goal true when performed.

## An example of partial-order planning

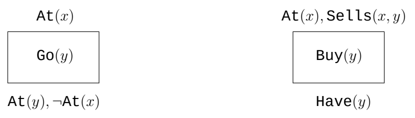
Here is the initial plan:


Start
At(Home) ∧ Sells(JS,G) ∧ Sells(HS,R) ∧ Sells(HS,FA)
At(Home) ∧ Have(G) ∧ Have(R) ∧ Have(FA)
Finish

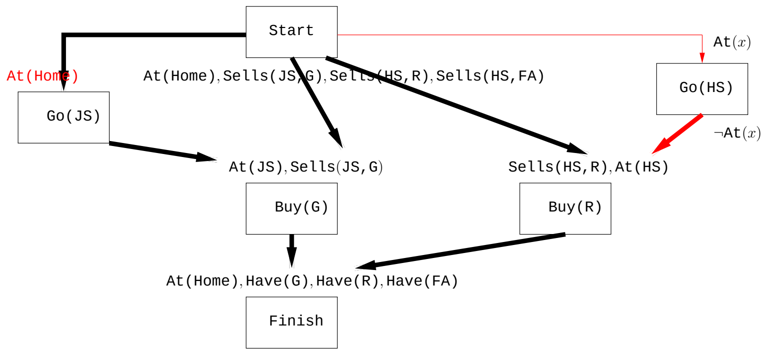
Thin arrows denote ordering.

## An example of partial-order planning

There are two actions available:



## An example of partial-order planning

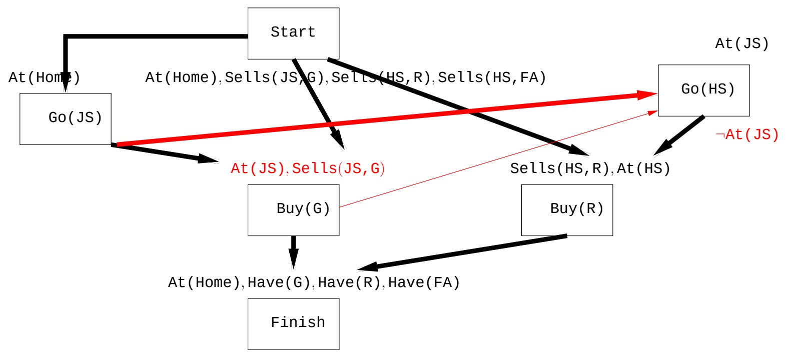


The `At(HS)` precondition is easy to achieve.

*But if we introduce a causal link from* `Start` *to* `Go(HS)` *then we risk invalidating the precondition for* `Go(JS)`.

## An example of partial-order planning

The planner could backtrack and try to achieve the `At`$(x)$ precondition using the existing `Go(JS)` step.



This involves a threat, but one that can be fixed using promotion.

## Using heuristics in planning

We found in looking at search problems that heuristics were a helpful thing to have.

Note that now:

- there is no simple representation of a state;
- consequently it is harder to measure the distance to a goal.

Defining heuristics for planning is therefore more difficult than it was for search problems.

## Using heuristics in planning

We can quickly suggest some possibilities.

For example

$$h = \text{number of unsatisfied preconditions}$$

or

$$h = \text{number of unsatisfied preconditions} - \text{number satisfied by the start state}$$

These can lead to underestimates or overestimates:

- underestimates if actions can affect one another in undesirable ways;
- overestimates if actions achieve many preconditions.

## Using heuristics in planning

We can go a little further by learning from Constraint Satisfaction Problems and adopting the most constrained variable heuristic:

- prefer the precondition satisfiable in the smallest number of ways.

This can be computationally demanding but two special cases are helpful:

- choose preconditions for which no action will satisfy them;
- choose preconditions that can only be satisfied in one way.

## Planning graphs

Planning graphs can be used:

- to compute more sensible heuristics;
- to generate entire plans.

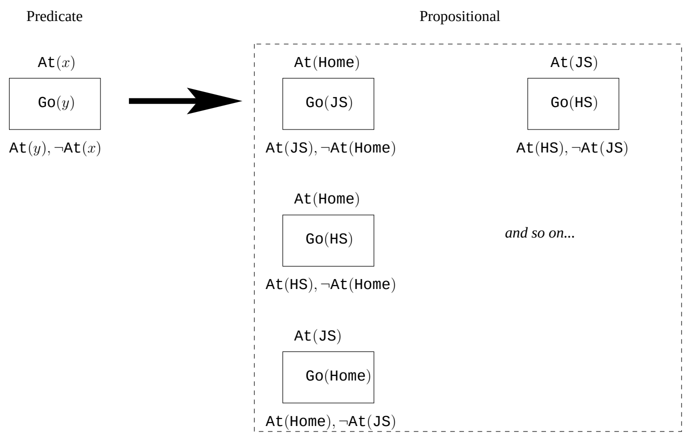
Also, planning graphs are easy to construct.

They apply only when it is possible to work entirely using propositional representations of plans.

Luckily, STRIPS can always be propositionalized...

## Planning graphs

For example: the triumphant return of the gorilla-purchasing roof-climbers...



## Planning graphs

A planning graph is constructed in levels:

- level $0$ corresponds to the start state;
- at each level we keep approximate track of all things that could be true at the corresponding time;
- at each level we keep approximate track of what actions could be applicable at the corresponding time.

The approximation is due to the fact that not all conflicts between actions are tracked. So:

- the graph can underestimate how long it might take for a particular proposition to appear, and therefore;
- a heuristic can be extracted.

## Planning graphs: a simple example

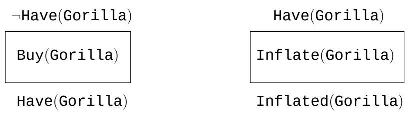
Our intrepid student adventurers will of course need to inflate their gorilla before attaching it to a distinguished roof. It has to be purchased before it can be inflated.

Start state: empty.

We assume that anything not mentioned in a state is false. So the state is actually

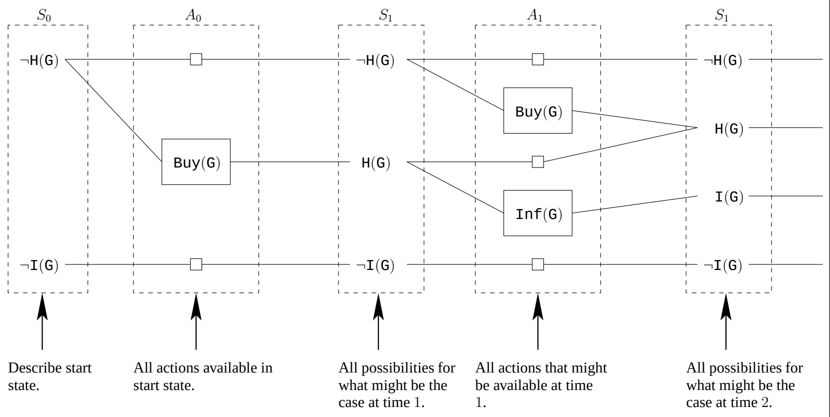
$$\neg\texttt{Have}(\texttt{Gorilla}) \text{ and } \neg\texttt{Inflated}(\texttt{Gorilla})$$

Actions:



Goal: Have(Gorilla) and Inflated(Gorilla).

## Planning graphs

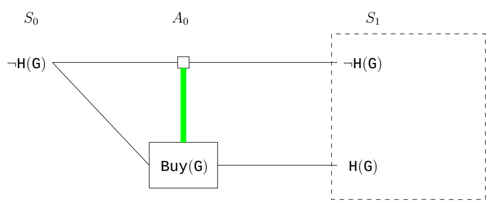

$S_0$
$A_0$
$S_1$
$A_1$
$S_1$
¬H(G)
¬I(G)
Buy(G)
¬H(G)
H(G)
¬I(G)
Buy(G)
Inf(G)
¬H(G)
H(G)
I(G)
¬I(G)
Describe start state.
All actions available in start state.
All possibilities for what might be the case at time 1.
All actions that might be available at time 1.
All possibilities for what might be the case at time 2.

□ = a *persistence action*—what happens if no action is taken.
An action level $A_i$ contains **all** actions that **could** happen given the propositions in $S_i$.

## Mutex links

We also record, using mutual exclusion (mutex) links which pairs of actions could not occur together.

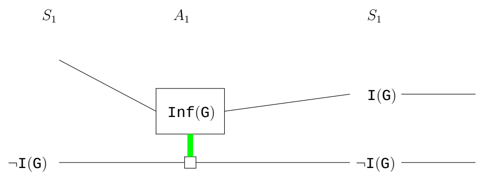
Mutex links 1: Effects are inconsistent.



The effect of one action negates the effect of another.

## Mutex links

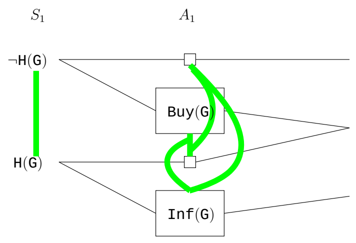
Mutex links 2: The actions interfere.



The effect of an action negates the precondition of another.

## Mutex links

Mutex links 3: Competing for preconditions.



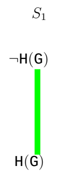
The precondition for an action is mutually exclusive with the precondition for another. (See next slide!)

## Mutex links

A state level $S_i$ contains all propositions that could be true, given the possible preceding actions.

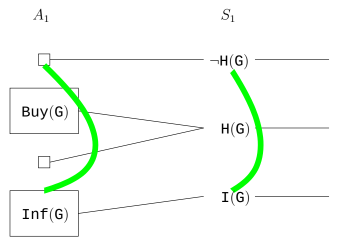
We also use mutex links to record pairs that can not be true simultaneously:

Possibility 1: pair consists of proposition and its negation.



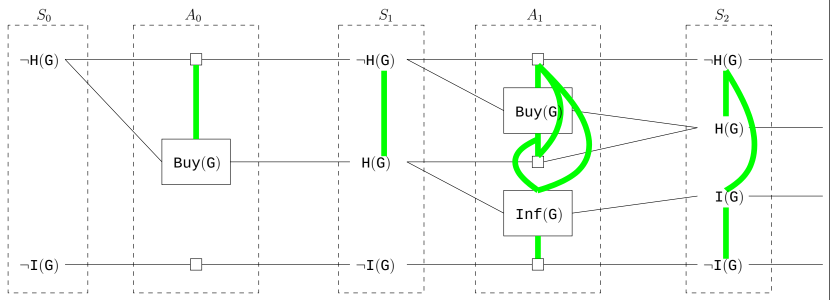
## Mutex links

Possibility 2: all pairs of actions that could achieve the pair of propositions are mutex.



The construction of a planning graph is continued until two identical levels are obtained.

## Planning graphs



## Obtaining heuristics from a planning graph

To estimate the cost of reaching a single proposition:

- any proposition not appearing in the final level has infinite cost and can never be reached;
- the level cost of a proposition is the level at which it first appears but this may be inaccurate as several actions can apply at each level and this cost does not count the number of actions. (It is however admissible.)
- a serial planning graph includes mutex links between all pairs of actions except persistence actions;

Level cost in serial planning graphs can be quite a good measurement.

## Obtaining heuristics from a planning graph

How about estimating the cost to achieve a collection of propositions?

- Max-level: use the maximum level in the graph of any proposition in the set. Admissible but can be inaccurate.
- Level-sum: use the sum of the levels of the propositions. Inadmissible but sometimes quite accurate if goals tend to be decomposable.
- Set-level: use the level at which all propositions appear with none being mutex. Can be accurate if goals tend not to be decomposable.

## Other points about planning graphs

A planning graph guarantees that:

1. if a proposition appears at some level, there may be a way of achieving it;
2. if a proposition does not appear, it can not be achieved.

The first point here is a loose guarantee because only pairs of items are linked by mutex links.

Looking at larger collections can strengthen the guarantee, but in practice the gains are outweighed by the increased computation.

## Graphplan

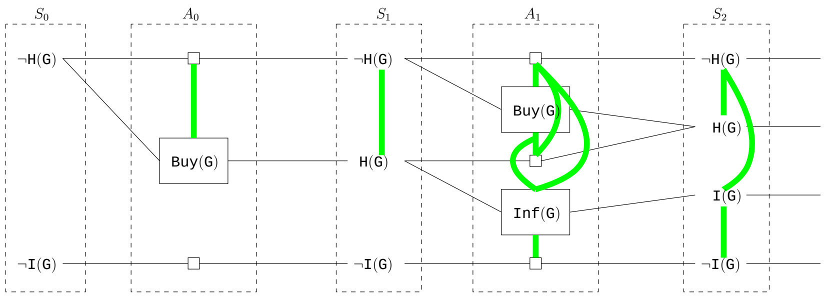
The GraphPlan algorithm goes beyond using the planning graph as a source of heuristics.

```
Start at level 0;
while(true)
{
  if (all goal propositions appear in the current level
      AND no pair has a mutex link)
  {
    attempt to extract a plan;
    if (a solution is obtained)
      return the solution;
    else if (graph indicates there is no solution)
      return fail;
    else
      expand the graph to the next level;
  }
}
```

We extract a plan directly from the planning graph. Termination can be proved but will not be covered here.

## Graphplan in action

Here, at levels $S_0$ and $S_1$ we do not have both `H(G)` and `I(G)` available with no mutex links, and so we expand first to $S_1$ and then to $S_2$.



At $S_2$ we try to extract a solution (plan).

## Extracting a plan from the graph

Extraction of a plan can be formalised as a search problem.

States contain a level, and a collection of unsatisfied goal propositions.

Start state: the current final level of the graph, along with the relevant goal propositions.

Goal: a state at level $S_0$ containing the initial propositions.

## Extracting a plan from the graph

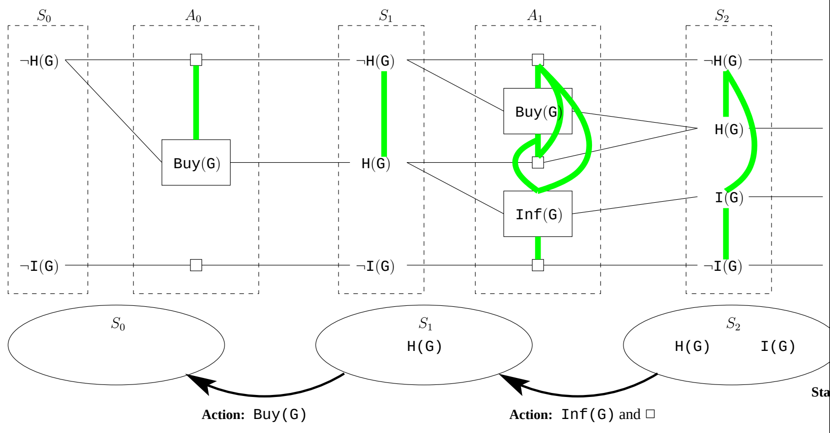
Actions: For a state $S$ with level $S_i$, a valid action is to select any set $X$ of actions in $A_{i-1}$ such that:

1. no pair has a mutex link;
2. no pair of their preconditions has a mutex link;
3. the effects of the actions in $X$ achieve the propositions in $S$.

The effect of such an action is a state having level $S_{i-1}$, and containing the preconditions for the actions in $X$.

Each action has a cost of $1$.

## Graphplan in action



## Heuristics for plan extraction

We can of course also apply heuristics to this part of the process.

For example, when dealing with a set of propositions:

- choose the proposition having maximum level cost first;
- for that proposition, attempt to achieve it using the action for which the maximum/sum level cost of its preconditions is minimum.

## Propositional logic for planning

Earlier we saw that plans might be extracted from a knowledge base via theorem proving, using first order logic (FOL) and situation calculus.

BUT: this might be computationally infeasible for realistic problems.

Sophisticated techniques are available for testing satisfiability in propositional logic, and these have also been applied to planning.

The basic idea is to attempt to find a model of a sentence having the form

description of start state
$\wedge$ descriptions of the possible actions
$\wedge$ description of goal

## Propositional logic for planning

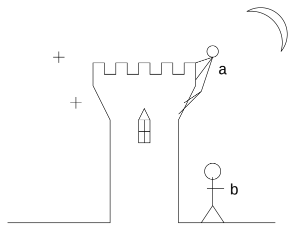
We attempt to construct this sentence such that:

- if $M$ is a model of the sentence then $M$ assigns $\top$ to a proposition if and only if it is in the plan;
- any assignment denoting an incorrect plan will not be a model as the goal description will not be $\top$.
- the sentence is unsatisfiable if no plan exists.

## Propositional logic for planning

Start state:

$$\begin{aligned} S = & \texttt{At}^0(\texttt{a},\texttt{spire}) \wedge \texttt{At}^0(\texttt{b},\texttt{ground}) \\ & \wedge \neg\texttt{At}^0(\texttt{a},\texttt{ground}) \wedge \neg\texttt{At}^0(\texttt{b},\texttt{spire}) \end{aligned}$$



The two climbers want to swap places...

Remember that an expression such as $\texttt{At}^0(\texttt{a},\texttt{spire})$ is a proposition. The superscripted number now denotes time.

## Propositional logic for planning

Goal:

$$\begin{aligned} G = & \texttt{At}^i(\texttt{a},\texttt{ground}) \wedge \texttt{At}^i(\texttt{b},\texttt{spire}) \\ & \wedge \neg\texttt{At}^i(\texttt{a},\texttt{spire}) \wedge \neg\texttt{At}^i(\texttt{b},\texttt{ground}) \end{aligned}$$

Actions: can be introduced using the equivalent of successor-state axioms

$$\begin{aligned} & \texttt{At}^1(\texttt{a,ground}) \leftrightarrow \\ & \quad (\texttt{At}^0(\texttt{a},\texttt{ground}) \wedge \neg(\texttt{At}^0(\texttt{a},\texttt{ground}) \wedge \texttt{Move}^0(\texttt{a},\texttt{ground},\texttt{spire}))) \\ & \quad \vee (\texttt{At}^0(\texttt{a},\texttt{spire}) \wedge \texttt{Move}^0(\texttt{a},\texttt{spire},\texttt{ground})) \end{aligned} \tag{1}$$

Denote by $A$ the collection of all such axioms.

## Propositional logic for planning

We will now find that $S \wedge A \wedge G$ has a model in which $\texttt{Move}^0(\texttt{a},\texttt{spire},\texttt{gro}$ and $\texttt{Move}^0(\texttt{b},\texttt{ground},\texttt{spire})$ are $\top$ while all remaining actions are $\perp$.

In more realistic planning problems we will clearly not know in advance at what time the goal might expect to be achieved.

We therefore:

- loop through possible final times $T$;
- generate a goal for time $T$ and actions up to time $T$;
- try to find a model and extract a plan;
- until a plan is obtained or we hit some maximum time.

## Propositional logic for planning

Unfortunately there is a problem—we may, if considerable care is not applied, also be able to obtain less sensible plans.

In the current example

$$\begin{aligned}\texttt{Move}^0(\texttt{b},\texttt{ground},\texttt{spire}) &= \top\\ \texttt{Move}^0(\texttt{a},\texttt{spire},\texttt{ground}) &= \top\\ \boxed{\texttt{Move}^0(\texttt{a},\texttt{ground},\texttt{spire})} &= \top\end{aligned}$$

is a model, because the successor-state axiom (1) does not in fact preclude the application of $\texttt{Move}^0(\texttt{a},\texttt{ground},\texttt{spire})$.

We need a precondition axiom

$$\texttt{Move}^i(\texttt{a},\texttt{ground},\texttt{spire}) \rightarrow \texttt{At}^i(\texttt{a},\texttt{ground})$$

and so on.

## Propositional logic for planning

Life becomes more complicated still if a third location is added: hospita

$$\texttt{Move}^0(\texttt{a},\texttt{spire},\texttt{ground}) \wedge \texttt{Move}^0(\texttt{a},\texttt{spire},\texttt{hospital})$$

is perfectly valid and so we need to specify that he can't move to two places simultaneously

$$\begin{gathered}\neg(\texttt{Move}^i(\texttt{a},\texttt{spire},\texttt{ground}) \wedge \texttt{Move}^i(\texttt{a},\texttt{spire},\texttt{hospital}))\\ \neg(\texttt{Move}^i(\texttt{a},\texttt{ground},\texttt{spire}) \wedge \texttt{Move}^i(\texttt{a},\texttt{ground},\texttt{hospital}))\\ \vdots\end{gathered}$$

and so on.

These are action-exclusion axioms.

Unfortunately they will tend to produce totally-ordered rather than partially-ordered plans.

## Propositional logic for planning

Alternatively:

1. prevent actions occurring together if one negates the effect or precondition of the other, or;
2. specify that something can't be in two places simultaneously

$$\forall x, i, \texttt{l1}, \texttt{l2} \quad \texttt{l1} \neq \texttt{l2} \rightarrow \neg(\texttt{At}^i(x,\texttt{l1}) \wedge \texttt{At}^i(x,\texttt{l2}))$$

This is an example of a state constraint.

Clearly this process can become very complex, but there are techniques to help deal with this.

## Problems

An undergraduate, Mr. R. J. Compsci, has turned up at this term's Big Party, only to find that it is in the home of his arch rival, who has turned him away. He spies in the driveway a large box and a ladder, and hatches a plan to gatecrash by getting in through a second floor window. Party on!

Here is the planning problem. He needs to move the box to the house, the ladder onto the box, then climb onto the box himself and at that point he can climb the ladder to the window.

## Problems

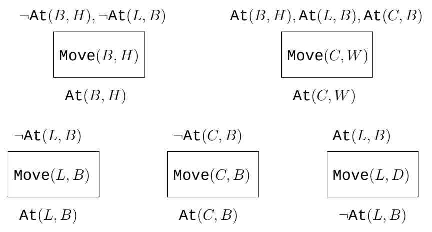
Using the abbreviations

- $B$ - Box
- $L$ - Ladder
- $H$ - House
- $C$ - Mr Compsci
- $W$ - Window

The start state is $\neg$At$(B,H)$, $\neg$At$(L,B)$, $\neg$At$(C,W)$ and $\neg$At$(C,B)$. The goal is At$(C,W)$.

## Problems

The available actions are



Construct the planning graph for this problem (you should probably start by finding a nice big piece of paper) and use the Graphplan algorithm to obtain a plan.