Planning: situation calculus

Knowledge-based system

- **Knowledge base:**
  - A set of sentences that describe the world in some formal (representational) language (e.g. first-order logic)
  - Domain specific knowledge

- **Inference engine:**
  - A set of procedures that work upon the representational language and can infer new facts or answer KB queries (e.g. resolution algorithm, forward chaining)
  - Domain independent
# Automated reasoning systems

### Examples and main differences:

- **Theorem provers**
  - Prove sentences in the first-order logic. Use inference rules, resolution rule and resolution refutation.

- **Deductive retrieval systems**
  - Systems based on rules (KBs in Horn form)
  - Prove theorems or infer new assertions (forward, backward chaining)

- **Production systems**
  - Systems based on rules with actions in antecedents
  - Forward chaining mode of operation

- **Semantic networks**
  - Graphical representation of the world, objects are nodes in the graphs, relations are various links

---

# Production systems

Based on rules, but different from KBs in the Horn form.

Knowledge base is divided into:

- **A Rule base (includes rules)**
- **A Working memory (includes facts)**

A special type of if – then rule

\[ p_1 \land p_2 \land \ldots \land p_n \Rightarrow a_1, a_2, \ldots, a_k \]

- **Antecedent**: a conjunction of literals
  - facts, statements in predicate logic

- **Consequent**: a conjunction of actions. An action can:
  - **ADD** the fact to the KB (working memory)
  - **REMOVE** the fact from the KB (consistent with logic ?)
  - **QUERY** the user, etc …
Production systems

Based on rules, but different from KBs in the Horn form
Knowledge base is divided into:

- A Rule base (includes rules)
- A Working memory (includes facts)

A special type of if – then rule

\[ p_1 \land p_2 \land \ldots \land p_n \Rightarrow a_1, a_2, \ldots, a_k \]

- **Antecedent**: a conjunction of literals
  - facts, statements in predicate logic
- **Consequent**: a conjunction of actions. An action can:
  - **ADD** the fact to the KB (working memory)
  - **REMOVE** the fact from the KB → !!! Different from logic
  - **QUERY** the user, etc …

Production systems

- Use **forward chaining to do reasoning**:
  - If the antecedent of the rule is satisfied (rule is said to be “active”) then its consequent can be executed (it is “fired”)

- **Problem**: Two or more rules are active at the same time. Which one to execute next?

R27 Conditions R27 ✓ ➔ Actions R27
R105 Conditions R105 ✓ ➔ Actions R105

- Strategy for selecting the rule to be fired from among possible candidates is called **conflict resolution**
**Production systems**

- Why is conflict resolution important? Or, Why do we care about the order?
- Assume that we have two rules and the preconditions of both are satisfied:

  \[ R1: \ A(x) \land B(x) \land C(y) \Rightarrow add \ D(x) \]

  \[ R2: \ A(x) \land B(x) \land E(z) \Rightarrow delete \ A(x) \]

- What can happen if rules are triggered in different order?
  - If \( R1 \) goes first, \( R2 \) condition is still satisfied and we infer \( D(x) \)
  - If \( R2 \) goes first we may never infer \( D(x) \)
Production systems

- **Problems with production systems:**
  - Additions and Deletions can change a set of active rules;
  - If a rule contains variables testing all instances in which the rule is active may require a large number of unifications.
  - Conditions of many rules may overlap, thus requiring to repeat the same unifications multiple times.

- **Solution: Rete algorithm**
  - gives more efficient solution for managing a set of active rules and performing unifications
  - Implemented in the system **OPS-5** (used to implement XCON – an expert system for configuration of DEC computers)

Rete algorithm

- Assume a set of rules:

\[ A(x) \land B(x) \land C(y) \Rightarrow add \ D(x) \]
\[ A(x) \land B(y) \land D(x) \Rightarrow add \ E(x) \]
\[ A(x) \land B(x) \land E(z) \Rightarrow delete \ A(x) \]

- And facts:  
  \[ A(1), A(2), B(2), B(3), B(4), C(5) \]

- **Rete:**
  - Compiles the rules to a network that merges conditions of multiple rules together (avoid repeats)
  - Propagates valid unifications
  - Reevaluates only changed conditions
**Rete algorithm. Network.**

**Rules:**
- \( A(x) \land B(y) \land C(z) \Rightarrow \text{add } D(x) \)
- \( A(x) \land B(y) \land D(x) \Rightarrow \text{add } E(x) \)
- \( A(x) \land B(x) \land E(z) \Rightarrow \text{delete } A(x) \)

**Facts:**
- \( A(1), A(2), B(2), B(3), B(4), C(5) \)

---

**Conflict resolution strategies**

- **Problem:** Two or more rules are active at the same time. Which one to execute next?
- **Solutions:**
  - No duplication (do not execute the same rule twice)
  - Recency. Rules referring to facts newly added to the working memory take precedence
  - Specificity. Rules that are more specific are preferred.
  - Priority levels. Define priority of rules, actions based on expert opinion. Have multiple priority levels such that the higher priority rules fire first.
Semantic network systems

- Knowledge about the world described in terms of graphs. Nodes correspond to:
  - **Concepts or objects** in the domain.

  Links to relations. Three kinds:
  - **Subset links** (isa, part-of links)
  - **Member links** (instance links)
  - **Function links**.

- Can be transformed to the first-order logic language
- Graphical representation is often easier to work with
  - better overall view on individual concepts and relations

---

Semantic network. Example.

```
Water

<table>
<thead>
<tr>
<th>Tree structure</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ship</td>
<td>isa, is-part</td>
</tr>
<tr>
<td>Ocean liner</td>
<td>isa, is-part</td>
</tr>
<tr>
<td>Oil tanker</td>
<td>member</td>
</tr>
<tr>
<td>Engine</td>
<td>member</td>
</tr>
<tr>
<td>Hull</td>
<td>is-part</td>
</tr>
<tr>
<td>Swimming pool</td>
<td>isa, member</td>
</tr>
<tr>
<td>Queen Mary</td>
<td>member</td>
</tr>
<tr>
<td>Exxon Valdez</td>
<td>member</td>
</tr>
<tr>
<td>Boiler</td>
<td>is-part</td>
</tr>
</tbody>
</table>

Inferred properties:  
Queen Mary is a ship
Queen Mary has a boiler
```
Planning: situation calculus

Representation of actions, situations, events

The world is dynamic:
• What is true now may not be true tomorrow
• Changes in the world may be triggered by our activities

Problems:
• Logic (FOL) as we had it referred to a static world. How to represent the change in the FOL?
• How to represent actions we can use to change the world?

Planning problem:
• find a sequence of actions that achieves some goal in this complex world?
• A very complex search problem
Situation calculus

Provides a framework for representing change, actions and reasoning about them

• Situation calculus
  – based on first-order logic,
  – a situation variable models new states of the world
  – action objects model activities
  – uses inference methods developed for FOL to do the reasoning

Situation calculus

• Logic for reasoning about changes in the state of the world
• The world is described by:
  – Sequences of situations of the current state
  – Changes from one situation to another are caused by actions
• The situation calculus allows us to:
  – Describe the initial state and a goal state
  – Build the KB that describes the effect of actions (operators)
  – Prove that the KB and the initial state lead to a goal state
    • extracts a plan as side-effect of the proof
Situation calculus

The language is based on the First-order logic plus:

- **Special variables**: $s,a$ – objects of type situation and action
- **Action functions**: return actions.
  - E.g. $\text{Move}(A, \text{TABLE}, B)$ represents a move action
  - $\text{Move}(x,y,z)$ represents an action schema
- **Two special function symbols of type situation**
  - $s_0$ – initial situation
  - $\text{DO}(a,s)$ – denotes the situation obtained after performing an action $a$ in situation $s$
- **Situation-dependent functions and relations**
  (also called *fluents*)
  - **Relation**: $\text{On}(x,y,s)$ – object $x$ is on object $y$ in situation $s$;
  - **Function**: $\text{Above}(x,s)$ – object that is above $x$ in situation $s$.

Situation calculus. Blocks world example.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

Initial state

- $\text{On}(A,\text{Table}, s_0)$
- $\text{On}(B,\text{Table}, s_0)$
- $\text{On}(C,\text{Table}, s_0)$
- $\text{Clear}(A, s_0)$
- $\text{Clear}(B, s_0)$
- $\text{Clear}(C, s_0)$
- $\text{Clear}($Table, $s_0)$

Goal

Find a state (situation) $s$, such that

- $\text{On}(A,B, s)$
- $\text{On}(B,C, s)$
- $\text{On}(C,\text{Table}, s)$
Blocks world example.

**Initial state**

- On(A, Table, s₀)
- On(B, Table, s₀)
- On(C, Table, s₀)
- Clear(A, s₀)
- Clear(B, s₀)
- Clear(C, s₀)
- Clear(Table, s₀)

**Goal**

- On(A, B, s)
- On(B, C, s)
- On(C, Table, s)

**Note:** It is not necessary that the goal describes all relations

Clear(A, s)

---

Blocks world example.

**Assume a simpler goal** On(A, B, s)

**Initial state**

- On(A, Table, s₀)
- On(B, Table, s₀)
- On(C, Table, s₀)
- Clear(A, s₀)
- Clear(B, s₀)
- Clear(C, s₀)
- Clear(Table, s₀)

**Goal** On(A, B, s)

3 possible goal configurations
Knowledge base: Axioms.

Knowledge base needed to support the reasoning:
• Must represent changes in the world due to actions.

Two types of axioms:
• **Effect axioms**
  – changes in situations that result from actions
• **Frame axioms**
  – things preserved from the previous situation

---

Blocks world example. Effect axioms.

**Effect axioms:**
Moving x from y to z. \(MOVE (x, y, z)\)

Effect of move changes on **On** relations
\[On(x, y, s) \land Clear(x, s) \land Clear(z, s) \rightarrow On(x, z, DO(MOVE(x, y, z), s))\]
\[On(x, y, s) \land Clear(x, s) \land Clear(z, s) \rightarrow \neg On(x, y, DO(MOVE(x, y, z), s))\]

Effect of move changes on **Clear** relations
\[On(x, y, s) \land Clear(x, s) \land Clear(z, s) \rightarrow Clear(y, DO(MOVE(x, y, z), s))\]
\[On(x, y, s) \land Clear(x, s) \land Clear(z, s) \land (z \neq \text{Table}) \rightarrow \neg Clear(z, DO(MOVE(x, y, z), s))\]
Blocks world example. Frame axioms.

- **Frame axioms.**
  - Represent things that remain unchanged after an action.

  **On relations:**
  \[
  On(u, v, s) \land (u \neq x) \land (v \neq y) \rightarrow On(u, v, DO(MOVE(x, y, z), s))
  \]

  **Clear relations:**
  \[
  Clear(u, s) \land (u \neq z) \rightarrow Clear(u, DO(MOVE(x, y, z), s))
  \]

Planning in situation calculus

**Planning problem:**
- find a sequence of actions that lead to a goal

**Planning in situation calculus is converted to the theorem proving problem**

**Goal state:**
\[
\exists s \ On(A, B, s) \land On(B, C, s) \land On(C, Table, s)
\]

- Possible inference approaches:
  - Inference rule approach
  - Conversion to SAT

- **Plan** (solution) is a byproduct of theorem proving.

- **Example:** blocks world
Planning in a blocks world.

Initial state

- On(A, Table, s₀)
- On(B, Table, s₀)
- On(C, Table, s₀)
- Clear(A, s₀)
- Clear(B, s₀)
- Clear(C, s₀)
- Clear(Table, s₀)

Goal

- On(A, B, s)
- On(B, C, s)
- On(C, Table, s)

Planning in the blocks world.

Initial state (s₀)

s₀ =

- On(A, Table, s₀) Clear(A, s₀) Clear(Table, s₀)
- On(B, Table, s₀) Clear(B, s₀)
- On(C, Table, s₀) Clear(C, s₀)

Action: MOVE (B, Table, C)

s₁ = DO(MOVE (B, Table, C), s₀)

- On(A, Table, s₁) Clear(A, s₁)
- On(B, C, s₁) Clear(B, s₁)
- ¬On(B, Table, s₁) Clear(C, s₁)
- On(C, Table, s₁)
Planning in the blocks world.

Initial state (s0) | s1 | s2
---|---|---
A | B | C

\[ s_1 = DO(MOVE(B, Table, C), s_0) \]
\[ On(A, Table, s_1) \]
\[ On(B, C, s_1) \quad Clear(A, s_1) \quad Clear(Table, s_1) \]
\[ \neg On(B, Table, s_1) \quad Clear(B, s_1) \]
\[ On(C, Table, s_1) \quad \neg Clear(C, s_1) \]

**Action:** \( MOVE(A, Table, B) \)
\[ s_2 = DO(MOVE(A, Table, B), s_1) \]
\[ = DO(MOVE(A, Table, B), DO(MOVE(B, Table, C), s_0)) \]
\[ On(A, B, s_2) \quad \neg On(A, Table, s_2) \quad \neg Clear(B, s_2) \]
\[ On(B, C, s_2) \quad \neg On(B, Table, s_2) \quad \neg Clear(C, s_2) \]
\[ On(C, Table, s_2) \quad Clear(A, s_2) \quad Clear(Table, s_2) \]

Planning in situation calculus.

**Planning problem:**
- Find a sequence of actions that lead to a goal
- Is a special type of a search problem
- Planning in situation calculus is converted to theorem proving.

**Problems:**
- Large search space
- Large number of axioms to be defined for one action
- Proof may not lead to the best (shortest) plan.
Planning

Planning problem:
- find a sequence of actions that achieves some goal
- An instance of a search problem

Methods for modeling and solving a planning problem:
- State space search
- Situation calculus based on FOL
  - Inference rules
  - Resolution refutation