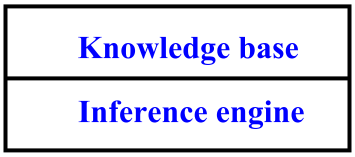
CS 2740 Knowledge Representation
Lecture 11

Production systems

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

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

- **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
- **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
- **Frames:**
  - object oriented representation, some procedural control of inference

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)**

Rules: a special type of if – then rule

$$p_1 \wedge p_2 \wedge \ldots p_n \Rightarrow a_1, a_2, \ldots, a_k$$

Antecedent:
A conjunction of conditions

Consequent:
a sequence of actions

Basic operation:

- Check if the antecedent of a rule is satisfied
- Decide which rule to execute (if more than one rule is satisfied)
- Execute actions in the consequent of the rule

Working memory

- Consists of a set of facts – statements about the world but also can represent various data structures
- The exact syntax and representation of facts may differ across different systems
- **Examples:**
  - **predicates**
  - such as Red(car12)
  - but only ground statements

  or
  - **(type attr1:value1 attr2:value2 …) objects**

    such as: (person age 27 home Toronto)

    The type, attributes and values are all atoms

Rules

$$p_1 \wedge p_2 \wedge \ldots p_n \Rightarrow a_1, a_2, \ldots, a_k$$

- **Antecedents: conjunctions of conditions**
- **Examples:**
  - a conjunction of literals $A(x) \wedge B(x) \wedge C(y)$
  - simple negated or non-negated statements in predicate logic
- or
  - conjunctions of conditions on objects/object
  - (type attr1 spec1 attr2 spec2 …)
  - Where specs can be an atom, a variable, expression, condition

    (person age [n+4] occupation x)

    (person age $\{< 23 \wedge > 6\}$)

Production systems

$$p_1 \wedge p_2 \wedge \dots p_n \Rightarrow a_1, a_2, \dots, a_k$$

- **Consequent:** a sequence of actions
- An action can be:
  - **ADD** the fact to the working memory (WM)
  - **REMOVE** the fact from the WM
  - **MODIFY** an attribute field
  - **QUERY** the user for input, etc …
- **Examples:**

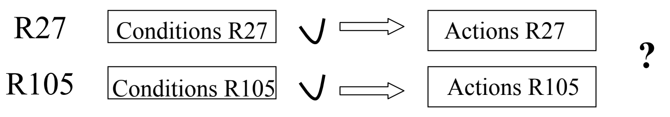
$$A(x) \wedge B(x) \wedge C(y) \Rightarrow add \;\; D(x)$$

- **Or**

(Student name x) $\Rightarrow$ ADD (Person name x)

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?



- 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: $A(x) \wedge B(x) \wedge C(y) \Rightarrow add\ \ D(x)$

R2: $A(x) \wedge B(x) \wedge E(z) \Rightarrow delete\ \ A(x)$

- What can happen if rules are triggered in different order?

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: $A(x) \wedge B(x) \wedge C(y) \Rightarrow add\ \ D(x)$

R2: $A(x) \wedge B(x) \wedge E(z) \Rightarrow delete\ \ A(x)$

- 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)

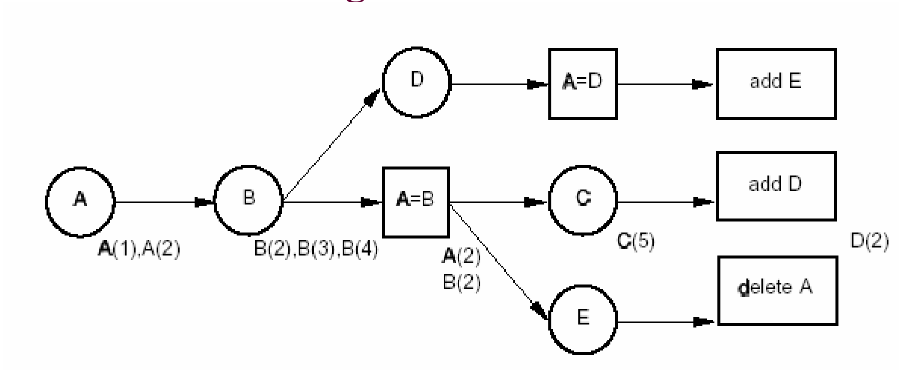
Rete algorithm

- Assume a set of rules:

$$A(x) \wedge B(x) \wedge C(y) \Rightarrow add \;\; D(x)$$
$$A(x) \wedge B(y) \wedge D(x) \Rightarrow add \;\; E(x)$$
$$A(x) \wedge B(x) \wedge E(z) \Rightarrow delete \;\; A(x)$$

- 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: $A(x) \wedge B(x) \wedge C(y) \Rightarrow add \;\; D(x)$

$A(x) \wedge B(y) \wedge D(x) \Rightarrow add \;\; E(x)$

$A(x) \wedge B(x) \wedge E(z) \Rightarrow delete \;\; A(x)$

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.

OPS-5

OPS5 (R1):

- A production system – a programming language
- Used to build commercial expert systems like XCON for configuration of the DEC computers

OPS/R2: (Production Systems Technologies inc.)

- Support for forward and backward chaining
- Improved Rete algorithm
- Object oriented-rules (with inheritance)
- Multiple WM
- User-defined control

End

System developed at CMU (as R1) and used extensively at DEC (now owned by Compaq) to configure early Vax computers

Nearly 10,000 rules for several hundred component types

Major stimulus for commercial interest in rule-based expert systems ★

```
IF
    the context is doing layout and assigning a power supply
    an sbi module of any type has been put in a cabinet
    the position of the sbi module is known
    there is space available for the power supply
    there is no available power supply
    the voltage and the frequency of the components are known
THEN
    add an appropriate power supply
```