Creative Systems
DESC9176
Rule-Based Systems
Rule-Based Design Systems

- A rule-based design system is comprised of the following parts:
  1. A set of rules
  2. A database of knowledge
  3. An algorithm for applying the rules to the knowledge
Rules

- A rule is a statement that is made up of two parts, an if-clause and a then-clause, e.g.:
  - If the day is Wednesday and the time is between 6pm and 9pm
  - Then the place to be is room 268
Facts

- A fact is a statement of knowledge about something in the world, e.g.:
  - The day is Wednesday
  - The time is between 6:30pm
Inference Algorithms

- Reasoning algorithms can infer new knowledge from existing knowledge.
- Two common algorithms used in rule-based design systems are:
  - Forward Chaining
  - Backward Chaining
Forward Chaining

- Forward chaining starts with the available facts and uses the rules to conclude new facts
  - Each rule is checked to determine whether the if-clause is true according to the known facts
  - If a rule is found, then the statements in the then-clause are added to the knowledge base
  - The process is repeated until no more rules can be fired, or until a goal state is reached
Forward Chaining

- Given the following two facts and the single rule, forward chaining can be used to conclude that Kermit is a frog:
  - If X is green and X can hop then X is a frog
  - Kermit is green
  - Kermit can hop
Backward Chaining

- Backward chaining starts with a goal and works backwards to determine if there are facts to support the conclusion that the goal is true.
  - Each rule is searched until one is found that has a then-clause that matches a desired goal.
  - If the if-clause of a matching rule is not known to be true, then it is added to the list of goals.
  - The process is repeated until one of the goals can be shown to be true.
Backward Chaining

Given the following two facts and two rules, backward chaining can be used to answer the question “Can Kermit croak?”

- If X is a frog then X can croak
- If X is green and X can hop then X is a frog
- Kermit is green
- Kermit can hop
Of course the reasoning is only as good as the knowledge base...
Programming Languages

- Logical programming languages allow a programmer to naturally express rules and facts and have a built-in inference engine.
- Prolog is a logical programming language with a simple syntax:
  - A fact is written as: A.
  - A rule is written as: A :- B.
Production Systems

- Production systems use forward chaining to produce new facts from a knowledge base.
- Production systems in design can be used to synthesise structure from behaviour.
- Production systems have been used in cognitive modelling to discover the effects of knowledge.
Expert Systems

Expert systems use forward and backward chaining to:

- analyse information supplied by a user as facts about a specific problem
- provide analysis of the problem
- recommend a course of action to the user in order to solve the problem
A Simple Design Example

- A program for designing an architectural unit obeying the following specifications:
  - Two rectangular rooms
  - Each room has a window and interior door
  - Rooms are connected by interior door
  - The front room also has an exterior door
  - No window can face north
  - Window and door cannot be on same wall
  - Windows cannot be on opposite sides
General Facts

direction(north).
direction(south).
direction(east).
direction(west).

opposite(north, south).
opposite(south, north).
opposite(east, west).
opposite(west, east).

not_opposite(D1, D2) :- opposite(D1, D3),
                      D2 =\= D3.
A Problem-Specific Definition of a Room

- Each room has a window and interior door
- No window can face north
- Window and door cannot be on same wall

```
room(D, W) :- direction(D),
            direction(W),
            D =\= W,
            W =\= north.
```
A Problem-Specific Definition of a Frontroom

- (The frontroom is a room)
- The frontroom also has an exterior door
- Window and door cannot be on the same wall

\[
\text{frontroom}(\text{FD}, \text{D}, \text{W}) :\text{ room}(\text{D}, \text{W}),
\text{direction}(\text{FD}),
\text{FD} =\text{W}. \]

A Problem-Specific Room Planner

› Two rectangular rooms
› Rooms are connected by interior door
› Windows cannot be on opposite sides

plan(FD, D1, W1, D2, W2) :- frontroom(FD, D1, W1),
room(D2, W2),
opposite(D1, D2),
not_opposite(W1, W2).
Planning a Unit Entered from the West

We can request that the planner produce a design with the requirement that the unit have a front door facing west:

?- plan(west, D1, W1, D2, W2).

D1 = west,
W1 = south,
D2 = east,
W2 = south
Checking a Design Against Requirements

We can check a design against the requirements by entering a fully specified call to plan.

?- plan(west, north, south, north, south).
no.

?- plan(north, north, west, south, west).
yes.
Design Grammars
Design Grammars

- Design grammars are production systems that can generate designs according to a set of rules (the grammar).

- A well-defined design grammar will generate designs that adhere to design constraints.
Advantages of Using a Design Grammar

- Design grammars give the designer the potential to evaluate a large number of alternative designs without laborious work.
- The designs produced by a grammar can contain designs that might have been overlooked without the aid of a grammar.
- Paving the way for possible innovative designs.
The Language of Grammars

- A set of non-terminal symbols
- A set of terminal symbols
- A set of rules, where a rule is of the form:
  - LHS → RHS
    - LHS and RHS contain terminal or non-terminal symbols
    - LHS must contain at least one non-terminal symbol
An Example Grammar

- Design Grammar
  - $S \rightarrow aSb$
  - $S \rightarrow ba$

- Produces:
  - $abab$ $aababb$ $aaababbb$ $aaaaababbbb$
Interpreting a Production

- In a design grammar, symbols produced by the grammar are interpreted as design elements
- E.g., commands to a drawing program:

  - a = ↓
  - b = →

  \[ \text{aababb} = \text{aabab} \]
Interpreting a Production

- The production of a design grammar can be interpreted in different ways
- E.g., components for a mechanical system:

  - a = spring
  - b = mass
  - abab =  
  - aababb = 
Example of a Real Design
Grammar
Context-Free vs Context-Sensitive

- Grammars can be written to be either context-free or context-sensitive
  - Context-free grammars do not represent the context of a symbol in the rules
  - Context-sensitive grammars represent the context of a symbol using a prefix and suffix
Applying the Rules of a Grammar

• How the rules are applied has a significant impact on what can be produced
  • Rules applied in sequence
    • One symbol replaced at a time
      • Chomsky Grammars
  • Rules applied in parallel
    • All symbols replaced at once
      • Lindenmayer Grammars
Lindenmayer Grammars

- Lindenmayer Grammars were developed to model the growth of natural forms
- Lindenmayer Systems are a very good way to produce organic forms, e.g. fractals, trees, flowers
Deterministic vs Non-Deterministic

- Some rule interpretation systems can support non-deterministic firing of rules, e.g. L-Systems
  - Multiple rules can be specified for the same LHS and the rule fired is chosen at random
  - Some systems allow the different rules to be weighted so that they fired with different probabilities