Generative Design Systems
Rob Saunders

web: http://web.arch.usyd.edu.au/~rob

e-mail: rob@arch.usyd.edu.au

office: Room 274, Wilkinson Building
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 Monday and the time is between 2pm and 5pm
  - Then the place to be is room 262
Facts

‣ A fact is a statement of knowledge about something in the world, e.g.:
‣ The day is Wednesday
‣ The time is 2:20pm
Inference Algorithms

- 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
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: $A :- B$. 
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

\[
\text{room}(D, W) \leftarrow \text{direction}(D), \\
\quad \text{direction}(W), \\
\quad D \neq W, \\
\quad W \neq \text{north}.
\]
A Problem-Specific Definition of a Frontroom

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

\[
\text{frontroom}(\text{FD}, \text{D}, \text{W}) \leftarrow \text{room}(\text{D}, \text{W}), \\
\quad \text{direction}(\text{FD}), \\
\quad \text{FD} \neq \text{W}.
\]
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 frontdoor facing west:

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

D1 = east,
W1 = south,
D2 = west,
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).

eyes.
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
Rule-Based Systems in Mass Customisation

- Mass customisation systems based on product configuration often use rule based systems, for example:
  - Facts may represent components, behaviours
  - Rules describe the valid ways that components can be put together and the effects of doing so
  - Expert systems can guide through the complex configuration process and/or help diagnose problems with existing configurations
Summary

- Rule-Based Systems
  - Rules + Facts + Inference Algorithm
- Mass Customisation
  - Help guide through configuration process
  - Diagnose problems with a configuration
Tutorial Exercise

- Sign up for an account with Ning at:

- Find some mass customisation web sites and go through the process of configuring a product, for each configurator
  - Post snapshots of your configurations
  - Post a link to the configurator on your blog together with a brief review