G64FAI: Foundations of Artificial Intelligence

Lecture 9: Expert Systems

Brian Logan
School of Computer Science
bsl@cs.nott.ac.uk

Outline of this lecture

- knowledge representation: facts & rules
- reasoning: forward chaining inference
- examples of expert systems: MYCIN & XCON
- implementing inference: rule matching and conflict resolution
- example: conflict resolution in CLIPS
- applying expert systems
- rule-based systems redux: business rules and the semantic web

Expert systems

- expert systems are one of the simplest applications of knowledge representation and reasoning
- consists of a set of facts, a set of rules and an implementation of the inference procedure
- support reasoning about a particular (narrow) domain in a tightly controlled way
- wide range of applications including medical diagnosis, mini-computer configuration, camera lens design, loan approvals, fault diagnosis etc.—often as part of a larger system
- one of the first commercial applications of AI

Advantages of expert systems

- order or magnitude increases in the speed with which complex tasks can be performed
- increased quality of decisions or solutions (or reduction in the number of errors)
- reduction in cost and number of personnel required and/or reduced training time (tasks are de-skilled)
- formalisation and retention of organisational or business knowledge

Facts & rules

- in an expert system, knowledge is represented as facts and rules using a simplified form of *predicate calculus*
- *facts* are ground (usually atomic) formulas, e.g., *Man*(*Socrates*), stored in *working memory*
- some facts are generic, while others are specific to a particular problem instance
- *rules* are universally quantified clauses—often with single negated literal (definite clauses), e.g., $\neg p \vee q \vee r \equiv (q \wedge r) \rightarrow p$
- a rule $C_1 \wedge \ldots \wedge C_n \rightarrow A$ consists of one or more *conditions* $C_1 \ldots C_n$ and a single *action A*
- all variables appearing in $C_1 \ldots C_n$ are assumed to be *universally quantified*

Inference

- usually forward chaining (modus ponens) with variable substitution

$$\frac{P(a), \quad \forall x\,(P(x) \rightarrow Q(x))}{Q(a)}$$

- for example, from the fact $Man(socrates)$ and the rule $\forall x\,(Man(x) \rightarrow Mortal(x))$ we can derive that

$$\frac{Man(socrates), \quad \forall x\,(Man(x) \rightarrow Mortal(x))}{Mortal(socrates)}$$

  using the substitution $\theta = \{x/socrates\}$

- inference is *sound* but (typically) *not complete*

Rule syntax

- to simplify development, rules are often written in a simplified form of English which omits the quantifiers
- for example, the rule (definite clause)

  $\forall x, \forall y$ (*PremiumCustomer*(*x*) ∧ *LuxuryProduct*(*y*) → *Discount*(*x*, *y*, *7.5%*))

  might be written

  IF *PremiumCustomer*(*x*) **AND** *LuxuryProduct*(*y*) **THEN** *Discount*(*x*, *y*, *7.5%*)

- sometimes referred to as '*production rules*', '*if-then rules*' or '*condition action rules*'

Inference cycle

- at each cycle, the LHS of each rule (its antecedent) is matched against the facts currently in working memory
- matching involves comparing *each* condition in the LHS of a rule in turn with *each* fact in working memory to see if a unifying substitution can be found which satisfies all the conditions
- rules where all the conditions can be matched (and all the variables bound) are said to be *applicable*
- the set of applicable *rule instances* is called the *conflict set*

Inference cycle 2

- one (or more) members of the conflict set are '*fired*' which performs the action on the RHS of the rule, e.g.:
  - adds a fact (or facts) to working memory
  - deletes a fact (or facts) from working memory
  - produces a side effect, e.g., prints some output
- cycle then repeats until one or more facts (representing a solution to the problem) appear in working memory or until no more rules can be fired

Example

- given the following facts and rules

F1 $mother(Mary, Bob)$
F2 $mother(Mary, Alice)$
F3 $father(Bob, Chris)$

R1 $mother(x, z) \wedge parent(z, y) \rightarrow grandmother(x, y)$
R2 $mother(x, y) \rightarrow parent(x, y)$
R3 $father(x, y) \rightarrow parent(x, y)$

Example: inference cycle 1

WM = [*mother*(*Mary*, *Bob*), *mother*(*Mary*, *Alice*), *father*(*Bob*, *Chris*)]

R1 no match
R2 {*x/Mary*, *y/Bob*}, {*x/Mary*, *y/Alice*}
R3 {*x/Bob*, *y/Chris*}

Conflict set = [**R2**:{*x/Mary*, *y/Bob*}, **R2**:{*x/Mary*, *y/Alice*},
R3:{*x/Bob*, *y/Chris*}]

Example: inference cycle 2

WM = [*mother*(*Mary*, *Bob*), *mother*(*Mary*, *Alice*), *father*(*Bob*, *Chris*),
parent(*Mary*, *Bob*)]

R1 no match
R2 {*x/Mary*, *y/Alice*}
R3 {*x/Bob*, *y/Chris*}

Conflict set = [**R2**:{*x/Mary*, *y/Alice*}, **R3**:{*x/Bob*, *y/Chris*}]

Example: inference cycle 3

WM = [*mother*(*Mary*, *Bob*), *mother*(*Mary*, *Alice*), *father*(*Bob*, *Chris*), *parent*(*Mary*, *Bob*), *parent*(*Mary*, *Alice*)]

R1 no match
R2 no match
R3 {*x/Bob*, *y/Chris*}

Conflict set = [**R3**:{*x/Bob*, *y/Chris*}]

Example: inference cycle 4

WM = [*mother*(*Mary*, *Bob*), *mother*(*Mary*, *Alice*), *father*(*Bob*, *Chris*), *parent*(*Mary*, *Bob*), *parent*(*Mary*, *Alice*), *parent*(*Bob*, *Chris*)]

R1 {*x/Mary*, *z/Bob*, *y/Chris*}
R2 no match
R3 no match

Conflict set = [**R1**:{*x/Mary*, *z/Bob*, *y/Chris*}]

Example: inference cycle 5

WM = [*mother*(*Mary*, *Bob*), *mother*(*Mary*, *Alice*), *father*(*Bob*, *Chris*),
parent(*Mary*, *Bob*), *parent*(*Mary*, *Alice*), *parent*(*Bob*, *Chris*),
grandmother(*Mary*, *Chris*)]

R1 no match
R2 no match
R3 no match

Conflict set = [ ]

Exercise: inference

- what happens if we add the fact

F1 $mother(Mary, Bob)$
F2 $mother(Mary, Alice)$
F3 $father(Bob, Chris)$
F4 $\mathbf{mother(Alice, Dan)}$

R1 $mother(x, z) \wedge parent(z, y) \rightarrow grandmother(x, y)$
R2 $mother(x, y) \rightarrow parent(x, y)$
R3 $father(x, y) \rightarrow parent(x, y)$

WM = [*mother*(*Mary*, *Bob*), *mother*(*Mary*, *Alice*), *father*(*Bob*, *Chris*),
parent(*Mary*, *Bob*), *parent*(*Mary*, *Alice*), *parent*(*Bob*, *Chris*),
grandmother(*Mary*, *Chris*)]

Example: inference cycle 6

WM = [*mother*(*Mary*, *Bob*), *mother*(*Mary*, *Alice*), *father*(*Bob*, *Chris*),
parent(*Mary*, *Bob*), *parent*(*Mary*, *Alice*), *parent*(*Bob*, *Chris*),
grandmother(*Mary*, *Chris*)]

R1 no match
R2 {*x/Alice*, *y/Dan*}
R3 no match

Conflict set = [**R2**:{*x/Alice*, *y/Dan*}]

Example: inference cycle 7

WM = [*mother*(*Mary*, *Bob*), *mother*(*Mary*, *Alice*), *father*(*Bob*, *Chris*),
parent(*Mary*, *Bob*), *parent*(*Mary*, *Alice*), *parent*(*Bob*, *Chris*),
grandmother(*Mary*, *Chris*), *parent*(*Alice, Dan*)]

R1 {*x/Mary*, *z/Alice, y/Dan*}
R2 no match
R3 no match

Conflict set = [**R1**:{*x/Mary*, *z/Alice*, *y/Dan*}]

Example: inference cycle 8

WM = [*mother*(*Mary*, *Bob*), *mother*(*Mary*, *Alice*), *father*(*Bob*, *Chris*),
parent(*Mary*, *Bob*), *parent*(*Mary*, *Alice*), *parent*(*Bob*, *Chris*),
grandmother(*Mary*, *Chris*), *parent*(*Alice, Dan*),
grandmother(*Mary*, *Dan*)]

R1 no match
R2 no match
R3 no match

Conflict set = [ ]

MYCIN

- developed in the 1970s, Stanford University (Shortliffe & Buchanan)
- diagnosis of bacterial infections
- based on interviews with experts on infectious diseases
- expert knowledge was reformulated as rules (mostly by system developers)
- discovered that knowledge acquisition is a non-trivial process—human experts find it hard to state all the knowledge required to solve a problem
- approximately 500 rules

Example MYCIN rule

```
RULE035
  PREMISE: ($ AND (SAME CNTXT GRAM GRAMNEG)
                  (SAME CNTXT MORPH ROD)
                  (SAME CNTXT AIR ANAEROBIC))
  ACTION: (CONCLUDE CNTXT IDENTITY BACTEROIDES TALLY .6)
```

- which can be translated as:

IF:
the gram stain of the organism is gramneg, and
the morphology of the organism is rod, and
the aerobicity of the organism is anaerobic
THEN:
there is suggestive evidence (.6) that the identity of the organism is bacteroides

More MYCIN

- some facts and some conclusions of the rules (as above) are not absolutely certain
- MYCIN uses numerical certainty factors between -1 and 1
- certainty factors of premises were combined with the tally in the rule (e.g., 0.6) to give a certainty factor for the conclusions
- later it turned out that MYCIN's recommendations would have been the same if it used only 4 values for certainty factors
- MYCIN was never used in practice due to ethical and legal issues
- when tested on real cases, did as well or better than the members of the Stanford medical school

XCON

- developed by McDermott at CMU (1978)
- system for configuring VAX (mini) computers—used by sales personnel to select system components based on customer requirements
- written using OPS5 (language for implementing production systems, written in LISP)
- 2,500 rules
- used commercially—by 1986 had processed 80,000 orders and was claimed to be saving DEC $25m pa.

Implementing inference

- an *expert system shell* defines a format for specifying facts and rules and an implementation of the inference procedure
- with many rules and many facts, there are two main problems
  - determining which inferences are *possible* at each cycle
  - determining which of those inferences should actually be made

Example: rule matching

- given the following facts and rules

  F1 $son(Mary, Joe)$
  F2 $son(Bill, Bob)$
  F3 $son(Bob, Charles)$
  F4 $daughter(Mary, Alice)$

  R1 $son(x, y) \wedge son(y, z) \rightarrow grandparent(x, z)$

- how many matching attempts will there be?
- what is the size of the conflict set?

Example: rule matching

- there are 16 matching attempts:
- **F1** matches with the first condition of **R1**, {*x/Mary*, *y/Joe*}
- then an attempt is made to match the second condition of **R1** *son*(*Joe*, *z*) with each of **F1**, **F2**, **F3** and **F4** (all of which fail)
- we then *backtrack* and match **F2** to the first condition of **R1**, {*x/Bill*, *y/Bob*}
- then an attempt is made to match the second condition of **R1** *son*(*Bob*, *z*) with each of **F1**, **F2**, **F3** and **F4** (one of which succeeds, **F3,** with *z/Charles*)

Example: rule matching

- we then backtrack and match **F3** to the first condition of **R1**, {*x/Bob*, *y/Charles*}
- then an attempt is made to match the second condition of **R1** *son*(*Charles*, *z*) with each of **F1**, **F2**, **F3** and **F4** (all of which fail)
- we then backtrack and try to match **F4** to the first condition of **R1** (which fails)
- there is only one rule instance in the conflict set **R1**: {*x/Bill*, *y/Bob, z/Charles*}

Rule matching

- rule firing is usually *refractory*, i.e., each rule instance fires at most once
- however each rule may match against many combinations of facts in WM—potentially exponential in the number of facts
- for typical problems, the number of matches is much smaller, but still needs to be recomputed at each cycle
- many systems use a *Rete network* to efficiently determine which rules match the current contents of working memory
- based on the assumption that firing a single rule makes only a few changes to WM, and that these changes typically only affect the applicability of a few rules

Conflict resolution

- firing all the applicable rules can lead to an explosion of possible inferences at later cycles
- most systems fire a *single rule* at each cycle, based on, e.g.:
  - **lexicographic order**: rules are tried in the order they appear in the program and the first matching rule is fired
  - **recency**: facts in WM are tagged with the inference cycle at which they were derived—rules that matched more recent facts are preferred
  - **specificity**: prefer more specific rules, i.e., rules with more conditions or more complex conditions
  - **weighting**: rules are assigned weights or importance values by the system developer—more important rules are preferred

Conflict resolution in CLIPS

- in CLIPS each rule has a *salience* reflecting its importance in problem solving
- new rule instances are placed above all rule instances of lower salience and below all rules of higher salience
- if rule instances have equal salience, ties are broken by the conflict resolution strategy
- CLIPS supports a variety of conflict resolution strategies including *depth*, *breadth*, *simplicity*, *complexity*, *lex*, *mea*, and *random*
- default strategy, *depth*, gives preference to new rule instances; *breadth* places older rule instances higher
- once the conflict set has been computed, CLIPS fires the highest ranking rule instance in the conflict set

Effects of conflict resolution strategy

- consider a CLIPS reasoner with the following set of rules using the *depth* conflict resolution strategy:

  R1: *tiger*(*x*) → *large-carnivore*(*x*)
  R2: *large-carnivore*(*x*) → *dangerous*(*x*)

  R1 has greater salience than R2

- If the agent's working memory contains the following facts:

  WM = [*tiger*(*tigger*), *tiger*(*sherekhan*)]

- then at the next cycle the agent would derive

  WM = [*tiger*(*tigger*), *tiger*(*sherekhan*), *large-carnivore*(*tigger*)]

Effects of conflict resolution strategy

- instances of R1 have greater salience than instances of R2, so on the following cycle the agent will derive

  WM = [*tiger*(*tigger*), *tiger*(*sherekhan*), *large-carnivore*(*tigger*), *large-carnivore*(*sherekhan*)]

- both *large-carnivore*(*tigger*), and *large-carnivore*(*sherekhan*) match R2, but *large-carnivore*(*sherekhan*) will be preferred since it is a more recent instance of R2 than *large-carnivore*(*tigger*)

- on the following cycle the agent therefore derives

  WM = [*tiger*(*tigger*), *tiger*(*sherekhan*), *large-carnivore*(*tigger*), *large-carnivore*(*sherekhan*), *dangerous*(*sherekhan*)]

- finally the agent derives

  WM = [*tiger*(*tigger*), *tiger*(*sherekhan*), *large-carnivore*(*tigger*), *large-carnivore*(*sherekhan*), *dangerous*(*sherekhan*), *dangerous*(*tigger*)]

Example: travel advice

- imagine that we want to give advice about travel destinations
- *far* destinations are *Chile* or *Kenya* $Far \rightarrow Chile \vee Kenya$
- *far* destinations are *international* $Far \rightarrow Int$
- *far* destinations are *expensive* $Far \rightarrow Exp$
- in *Kenya*, *yellow fever* vaccination is strongly recommended, and there is a risk of *malaria* when staying in *lodges*

  $Kenya \rightarrow YellowFever \quad Logde \wedge Kenya \rightarrow Malaria$
- accommodation in *Kenya* is in *lodges* and in *Chile* is in *hotels*

  $Kenya \rightarrow Lodge \quad Chile \rightarrow Hotel$
- when there is a risk of *malaria*, mosquito *nets* are recommended

  $Malaria \rightarrow Nets$

Example: travel advice clauses

- Clauses (1) $\neg Far \vee Chile \vee Kenya$
  (2) $\neg Far \vee Int$
  (3) $\neg Far \vee Exp$
  (4) $\neg Kenya \vee YellowFever$
  (5) $\neg Lodge \vee \neg Kenya \vee Malaria$
  (6) $\neg Kenya \vee Lodge$
  (7) $\neg Chile \vee Hotel$
  (8) $\neg Malaria \vee Nets$
- Prove that Far and $\neg Hotel$ entails $Kenya$
- and that Far and $\neg Hotel$ entails $Nets$

Example: travel advice clauses

- Clauses (1) $\neg Far \vee Chile \vee Kenya$
  (2) $\neg Far \vee Int$
  (3) $\neg Far \vee Exp$
  (4) $\neg Kenya \vee YellowFever$
  (5) $\neg Lodge \vee \neg Kenya \vee Malaria$
  (6) $\neg Kenya \vee Lodge$
  (7) $\neg Chile \vee Hotel$
  (8) $\neg Malaria \vee Nets$
- Premises (9) Far
  (10) $\neg Hotel$
- Goal (11) $\neg Kenya$

Example: travel expert system

- we can reformulate the travel advice problem as a set of rules and facts

(R1a) $Far(x) \wedge \neg Accommodation(x, lodge) \rightarrow Destination(x, chile)$
(R1b) $Far(x) \wedge \neg Accommodation(x, hotel) \rightarrow Destination(x, kenya)$
(R2) $Far(x) \rightarrow Int(x)$
(R3) $Far(x) \rightarrow Exp(x)$
(R4) $Destination(x, kenya) \rightarrow Risk(x, yellowFever)$
(R5) $Accommodation(x, lodge) \wedge Destination(x, kenya) \rightarrow$
$Risk(x, malaria)$
(R6) $Destination(x, kenya) \rightarrow Accommodation(x, lodge)$
(R7) $Destination(x, chile) \rightarrow Accommodation(x, hotel)$
(R8) $Risk(x, malaria) \rightarrow Advised(x, nets)$

- note that the reformulation has *changed the problem*—without information about *Accommodation*, we can't say anything about the *Destination*

Example: travel expert system

- given the facts

  (F1) $Far(johnsHoliday)$
  (F2) $\neg Accommodation(johnsHoliday, hotel)$

- we can derive first

  (F3) $Destination(johnsHoliday, kenya)$

- and then on subsequent inference cycles

  (F4) $Int(johnsHoliday)$,
  (F5) $Exp(johnsHoliday)$,
  (F6) $Risk(johnsHoliday, yellowFever)$,
  (F7) $Accommodation(johnsHoliday, lodge)$,
  (F8) $Risk(johnsHoliday, malaria)$,
  (F9) $Advised(johnsHoliday, nets)$

Comparison with theorem proving

- unlike resolution theorem proving, we don't need to know the clause (fact) we want to derive in advance
- inference is *data-driven*, chaining forward from statements about the problem
- if the rules are not carefully chosen, this can result in the derivation of a large number of irrelevant facts
- some things are hard to represent or reason about using definite clauses, e.g.
  - disjunctions—"Chile *or* Kenya"
  - negations—hard to infer from negative information "I don't want to stay in a hotel"

Problem characteristics

- for the successful development of an expert system, the problem should
- be solvable by a human in 3-180 minutes
- be primarily cognitive, requiring analysis and synthesis
- be well defined and confined to a narrow domain
- not involve a great deal of common sense reasoning
- task knowledge and case studies which are reasonably complete must be available

Application areas

- assistants to human operators
  - generating candidate solutions to difficult design, synthesis or analysis problems
  - to evaluate candidate solutions (produced by humans)
- autonomous decision making components of complex systems
- monitoring the implementation and execution of designs, plans and schedules

Business rules

- a *business rule* is a statement that defines or constrains some aspect of a business
- aim is to separate dynamically changing business procedures, policies and logic from the application source code
- rules are declarative and can be easily modified in response to business needs by non-programmers
- e.g., on-line retail or rental business (see http://www.businessrulesgroup.org/egsbrg.shtml)
- rules are executed by a *rule engine*

Java Rule Engine API

- Java Rule Engine API (JSR-94) is a lightweight programming interface that defines a standard API for acquiring and using a rule engine
- aims to "*to reduce the cost associated with incorporating business logic within applications and … the need to reduce the cost associated with implementing platform-level business logic tools and services*"
- see `javax.rules` and `javax.rules.admin` packages

JESS

- Jess is an expert system shell—a rule engine to which users add their own facts and rules
- written in Java
- uses Rete for efficient incremental rule matching
- reference implementation of the Java Rule Engine API

JESS syntax

- facts are specified as a predicate followed by a list of slots

```
(person (name "Bob Smith") (age 34) (gender Male))
```

- rules can be specified in a LISP-like Jess rule language (or XML):

```
(defrule young-persons-discount
 (person (name ?name) {age < 21})
  =>
 (assert (eligible-for-discount (name ?n)))
```

- LHS is a pattern (if a person is less than 21 years old) and RHS is an action (function call which can add or delete facts or produce output)

The next lecture

Reasoning about actions

Suggested reading:

- Russell & Norvig (2003), chapter 11, section 11.1, 11.2, 11.3 and 11.5