

COMP219: Artificial Intelligence

Lecture 16: Forward and Backward Chaining

1

Rule-Based System Architecture

- A collection of **rules**
- A collection of **facts**
- An **inference engine**
- We might want to
 - See what new facts can be *derived*
 - *Ask* whether a fact is implied by the knowledge base and facts already known

3

Overview

- **Last time**
 - Introduced the reasons for explicit knowledge representation
 - Discussed properties of knowledge representation schemes
 - Introduced rules as a form of knowledge representation
- **Today**
 - Introduce algorithms for reasoning with rules
 - Discuss some of the problems of rule-based representations
- Learning outcome covered today:
Distinguish the characteristics, and advantages and disadvantages, of the major knowledge representation paradigms that have been used in AI, such as production rules, semantic networks, propositional logic and first-order logic;

2

Control Schemes



- Given a set of rules, there are essentially two ways we can use them to generate new knowledge
 - **forward chaining**
 - starts with the facts, and sees what rules apply (and hence what should be done) given the facts
 - data driven
 - **backward chaining**
 - starts with something to find out, and looks for rules that will help in answering it
 - goal driven

4

Fire Alarm Example



```
R1: IF hot AND smoky THEN fire
R2: IF alarm_beeps THEN smoky
R3: IF fire THEN sprinklers_on
```

```
F1: alarm_beeps [Given]
F2: hot [Given]
```

- We need to make the consequents **actions**

5

Fire Alarm Example



```
R1: IF hot AND smoky THEN ADD fire
R2: IF alarm_beeps THEN ADD smoky
R3: IF fire THEN DO switch_sprinklers_on
      ADD sprinklers_on
```

```
F1: alarm_beeps [Given]
F2: hot [Given]
```

6

Fire Alarm Example



```
R1: IF hot AND smoky THEN ADD fire
R2: IF alarm_beeps THEN ADD smoky
R3: IF fire THEN DO switch_sprinklers_on
      ADD sprinklers_on
```

```
F1: alarm_beeps [Given]
F2: hot [Given]
```

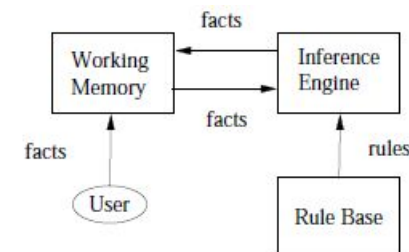
Forward Chaining

Use F1 and R2 to get F3 smoky
Use F2 and F3 and R1 to get F4 fire
Use F4 and R3 to get F5 sprinklers_on

7

Forward Chaining

- In a forward chaining system
 - Facts are held in a **working memory (WM)**
 - Condition-action rules represent actions to take when specified facts occur in working memory
 - Often the actions involve adding or deleting facts from working memory



8

Extending the Example



```
R1: IF hot AND smoky THEN ADD fire
R2: IF alarm_beeps THEN ADD smoky
R3: IF fire THEN DO switch_sprinklers_on
      ADD sprinklers_on
R4: IF dry THEN DO switch_on_humidifier
      ADD humidifier_on
R5: IF sprinklers_on THEN DELETE dry

F1: alarm_beeps;    F2: hot;    F3: dry
```

Now *two* rules match: R2 *and* R4

9

Which rule to use?



- Use R2:
 - Add **smoky**: now R1 and R4 match
- Use R1:
 - Add **fire**: now R3 and R4 match
- Use R3:
 - Add **sprinklers_on**: R4 and R5 match
- Use R5:
 - Delete **dry**: now R4 does **not** match
- Note that R4 is **never** used in this sequence; so the choice **can affect the result**
- We have a **conflict**: we need a **conflict resolution strategy** to select the **right** rule

10

Forward Chaining Algorithm

Repeat

Collect rules whose conditions match facts in WM.

If more than one rule matches:

Use **conflict resolution strategy** to eliminate all but one.

Do actions indicated by the rules (add facts to WM or delete facts from WM).

Until problem is solved or no condition match.

11

Conflict Resolution Strategy

- There are a number of approaches
 - Physically **order the rules**
 - hard to add rules to these systems
 - **Data** ordering
 - arrange problem elements in priority queue
 - use rule dealing with highest priority elements
 - **Specificity** or maximum specificity
 - based on number of conditions matching
 - choose the one with the most matches



12

More Strategies

- **Recency** ordering
 - Data (based on order facts added to WM)
 - Rules (based on rule firings)
- **Context** limiting
 - partition rule base into disjoint subsets
 - we may order the subsets and we may also have preconditions
- **Random** selection
- Can also have combinations to break ties

13



Properties of Forward Chaining

- Can be inefficient - lead to spurious rules firing, and unfocused problem solving (cf. breadth-first search)
- Set of rules that can fire known as **conflict set**
- Decision about which rule to fire - **conflict resolution**

- Different conflict resolutions may give different behaviour and different results

15

Meta Knowledge

- Another solution: use **meta-knowledge** (i.e. knowledge about knowledge) to guide search
- Example of meta-knowledge

```
IF
    conflict set contains any rule (c,a)
    such that a = ``animal is mammal``
THEN
    fire (c,a)
```
- So meta-knowledge encodes knowledge about how to guide search to solve the problem
- Explicitly coded in the form of rules, as with “object level” knowledge

14

Application Areas

- Computer system configuration
 - Many possible set ups: forward chain from user needs
- Reactive robots
 - Get facts from environment and respond appropriately
- Conversational agents
 - Decide on the meaning of natural language input to give an appropriate response

16



Backward Chaining

- Same rules/facts may be processed differently, using backward chaining interpreter
- Backward chaining means reasoning from **goals back to facts**
- The idea is that this **focuses the search**
- Starts from a *goal* or *hypothesis*
 - Should I switch the sprinklers on?

17

Fire Alarm Example



```

R1: IF hot AND smoky THEN ADD fire
R2: IF alarm_beeps THEN ADD smoky
R3: IF fire THEN DO switch_sprinklers_on
      ADD sprinklers_on
F1: alarm_beeps;    F2: hot

```

- Goal: switch_sprinklers_on

Backward Chaining
R3 justifies goal if fire
R1 justifies fire if hot and smoky
Hot is a fact: R2 justifies smoky if alarm beeps
Alarm beeps is a fact

19

Backward Chaining Algorithm

To prove goal G:

If G is in the initial facts, it is proven.

Otherwise, find a rule which can be used to conclude G, and try to prove each of that rule's conditions (make conditions **sub-goals**).

- We add **goals**, not **facts** to working memory

18

Using Prolog

- Prolog supports backward chaining directly:

```

alarm_beeps.
hot.

```

```

fire :- hot, smoky.
smoky :- alarm_beeps.
switch_on_sprinklers :- fire.

```

Conflict resolution is handled by clause order

20

Forward Chaining in Prolog

```
go(X):-member(sprinklers_on,X).
go(X):-member(fire,X), write([switching,sprinklers,on]),
      go([sprinklers_on | X]).
go(X):-member(hot,X), member(smoky,X), go([fire | X]).
go(X):-member(alarm_beeps,X), go([smoky | X]).

?- go([hot,alarm_beeps]).
```

- Argument (the maintained list) acts as working memory
- Member succeeds if fact in working memory
- Conflict resolution through ordering of clauses

21

Forward vs Backward Chaining

- Depends on problem, and on properties of rule set
- If you have clear hypotheses, backward chaining is likely to be better
 - Goal driven
 - Diagnostic problems or classification problems
 - Medical expert systems
- Forward chaining may be better if you have no clear hypothesis and want to see what can be concluded from current situation
 - Data driven
 - Synthesis systems
 - Configuration
 - Reactive systems

23

Exercise

Consider the following financial advice scenario:

```
R1: IF NOT savings_adequate THEN DO invest_savings
      ADD savings_invested
R2: IF savings_adequate AND income_adequate THEN
      DO invest_stocks ADD stocks_invested
R3: IF NOT has_children THEN ADD savings_adequate
R4: IF has_partner AND partner_employed THEN
      ADD income_adequate

F1: has_children
F2: has_partner
F3: partner_employed
```

Should I invest in stocks?

22

Properties of Rules



- Rules are a natural representation
- They are inferentially adequate
- They are representationally adequate for some types of information/environments
- They can be inferentially inefficient (basically doing unconstrained search)
- They can have a well-defined syntax, but lack well-defined semantics
 - Conflict resolution can change their meaning

24

Problems for Rules

- Inaccurate or incomplete information (inaccessible environments)
- Uncertain inference (non-deterministic environments)
- Non-discrete information (continuous environments)
- Default values
 - Anything that is not stated or derivable is false: they make the *closed world assumption*

25

Summary

- We have looked at rules, which have often been used as a form of knowledge representation
- They can be used in either a **data** driven or a **goal** driven manner
 - Forward vs backward chaining
- **Next time**
 - We will look at a different form of knowledge representation: **structured objects**

26