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 already known facts

Control Schemes

- Given a set of rules like these, 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.

A Simple Example (I)

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

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

A Simple Example (II)

```
R1: IF hot AND smoky THEN ADD fire
R2: IF alarm_beeps THEN ADD smoky
R3: If fire THEN ADD switch_on_sprinklers

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

A Simple Example (III)

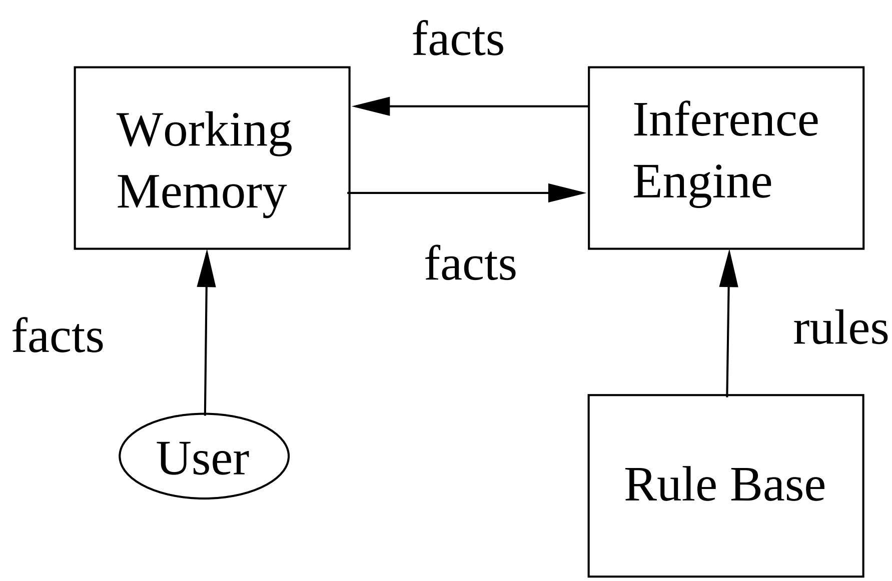
```
R1: IF hot AND smoky THEN ADD fire
R2: IF alarm_beeps THEN ADD smoky
R3: If fire THEN ADD switch_on_sprinklers

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

F3: smoky                 [from F1 by R2]
F4: fire                  [from F2, F4 by R1]
F5: switch_on_sprinklers  [from F4 by R3]
```

A typical *Forward Chaining* example

Forward Chaining

In a forward chaining system:

- Facts are held in a *working memory*
- Condition-action rules represent actions to take when specified facts occur in working memory.
- Typically the actions involve adding or deleting facts from working memory.



Forward Chaining Algorithm (I)

- Repeat
  - Collect the rule whose condition matches a fact in WM.
  - Do actions indicated by the rule
    (add facts to WM or delete facts from WM)
- Until problem is solved or no condition match

Extending the Example

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

F1: alarm_beeps   [Given]
F2: hot           [Given]
F3: dry           [Given]
```

Extending the Example

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

F1: alarm_beeps   [Given]
F2: hot           [Given]
F3: dry           [Given]
```

Now, *two* rules can fire (R2 and R4)

- R4 fires, humidifier is on (then, as before)
- R2 fires, humidifier is off

A conflict!

Forward Chaining Algorithm (II)

- Repeat
  - Collect the *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

Conflict Resolution Strategy (I)

- 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

Conflict Resolution Strategy (II)

- Recency Ordering
  - Data (based on order facts added to WM)
  - Rules (based on rule firings)
- Context Limiting
  - partition rule base into disjoint subsets
  - doing this we can have subsets and we may also have preconditions
- Randomly Selection
- Fire All Application Rules

Meta Knowledge

- Another solution: *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 for solution.
- Explicitly coded in the form of rules, as with “object level” knowledge.

Properties of Forward Chaining

- Note that *all rules which can fire do fire*.
- Can be inefficient — lead to spurious rules firing, 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*.

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.
- Checking *hypothesis*
  - `Should I switch the sprinklers on?`

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.

Example

Rules:

R1: IF hot AND smoky THEN fire

R2: IF alarm_beeps THEN smoky

R3: If fire THEN switch_on_sprinklers

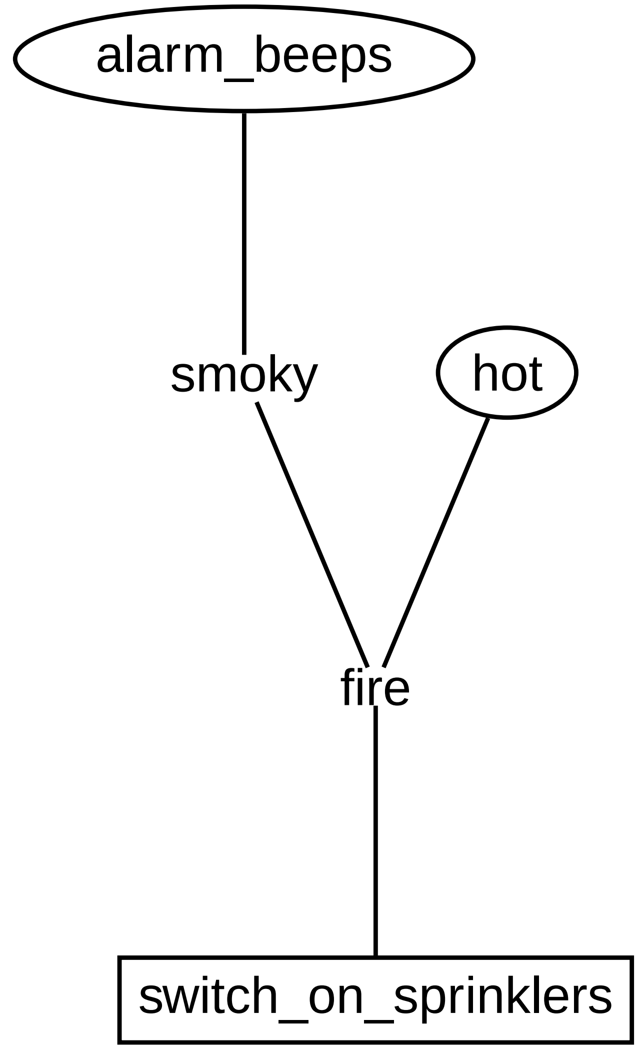
Facts:

F1: hot

F2: alarm_beeps

Goal:

Should I switch sprinklers on?



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 less clear hypothesis and want to see what can be concluded from current situation.
  - Data driven
  - Synthesis systems
    - Design / configuration

Properties of Rules (I)

- 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 a well defined semantics.

Properties of Rules (II)

- They have problems with
  - 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 *closed world assumption*

Example (1)

$$\left.\begin{array}{l} \text{alarm\_beeps} \wedge \text{hot} \\ \quad \wedge(\text{hot} \wedge \text{smoky} \Rightarrow \text{fire}) \\ \quad \wedge(\text{alarm\_beeps} \Rightarrow \text{smoky}) \\ \quad \wedge(\text{fire} \Rightarrow \text{switch\_on\_sprinklers}) \end{array}\right\} \overset{?}{\models} \quad \text{switch\_on\_sprinklers}$$

Example (1)

$$\left.\begin{array}{l} \text{alarm\_beeps} \wedge \text{hot} \\ \boxed{\begin{array}{l} \wedge(\text{hot} \wedge \text{smoky} \Rightarrow \text{fire}) \\ \wedge(\text{alarm\_beeps} \Rightarrow \text{smoky}) \\ \wedge(\text{fire} \Rightarrow \text{switch\_on\_sprinklers}) \end{array}} \end{array}\right\} \stackrel{?}{\models} \quad \text{switch\_on\_sprinklers}$$

Example (2)

$$\left.\boxed{\begin{array}{l}(\text{hot} \wedge \text{smoky} \Rightarrow \text{fire}) \\ \wedge(\text{alarm\_beeps} \Rightarrow \text{smoky}) \\ \wedge(\text{fire} \Rightarrow \text{switch\_on\_sprinklers})\end{array}}\right\} \overset{?}{\models}$$

$$\boxed{\left.\begin{array}{l}\text{alarm\_beeps} \\ \wedge\text{hot}\end{array}\right\} \Rightarrow \text{switch\_on\_sprinklers}}$$

Example (3)

$$\left.\boxed{\begin{array}{l}(\text{hot} \wedge \text{smoky} \Rightarrow \text{fire})\\ \wedge(\text{alarm\_beeps} \Rightarrow \text{smoky})\\ \wedge(\text{fire} \Rightarrow \text{switch\_on\_sprinklers})\end{array}}\right\} \overset{?}{\models}$$

$$\boxed{\neg\text{switch\_on\_sprinklers} \Rightarrow \neg\text{fire}}$$

Example (4)

$$\left.\boxed{\begin{array}{l}(\text{hot} \wedge \text{smoky} \Rightarrow \text{fire}) \\ \wedge(\text{alarm\_beeps} \Rightarrow \text{smoky}) \\ \wedge(\text{fire} \Rightarrow \text{switch\_on\_sprinklers})\end{array}}\right\} \overset{?}{\models}$$

$$\boxed{\left.\begin{array}{l}\neg\text{switch\_on\_sprinklers} \\ \wedge\text{hot}\end{array}\right\} \Rightarrow \neg\text{smoky}}$$

Propositional Logic for KR

- Describe what we know about a particular domain by a propositional formula, KB.
- Formulate a hypothesis, α.
- We want to know whether KB implies α.

Entailment

- *Entailment* means that one thing *follows from* another:

$$KB \models \alpha$$

- Knowledge base KB entails sentence α if and only if α is true in *all worlds* where KB is true
  - E.g., the KB containing "the Giants won" and "the Reds won"
    entails "Either the Giants won or the Reds won"
  - E.g., $x + y = 4$ entails $4 = x + y$
- Entailment is a relationship between sentences (i.e., *syntax*) that is based on *semantics*

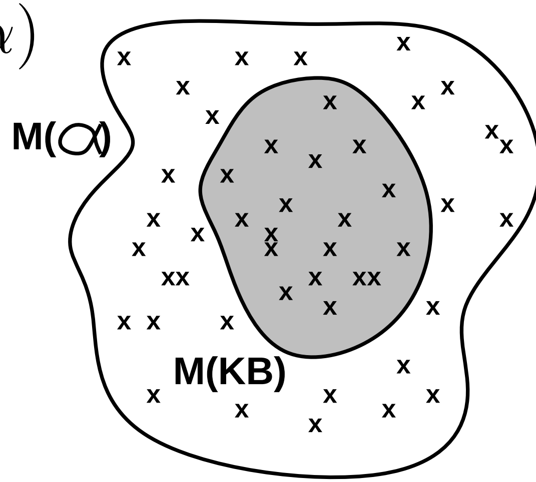
Entailment Test

How do we know that $KB \models \alpha$?

- Models
- Inference

Models

- Logicians typically think in terms of *models*, which are formally structured worlds with respect to which truth can be evaluated
- We say m is a model of a sentence α if α is true in m
- $M(\alpha)$ is the set of all models of α
- Then $KB \models \alpha$ if and only if $M(KB) \subseteq M(\alpha)$
  - E.g. KB = Giants won and Reds won α = Giants won



Boris Konev

Example

$$\left.\begin{array}{l}(\text{hot} \wedge \text{smoky} \Rightarrow \text{fire}) \\ \wedge(\text{alarm\_beeps} \Rightarrow \text{smoky}) \\ \wedge(\text{fire} \Rightarrow \text{switch\_on\_sprinklers})\end{array}\right\} \overset{?}{\models} \neg\text{switch\_on\_sprinklers} \Rightarrow \neg\text{fire}$$

Example

$$\left.\begin{array}{l}(\mathbf{H}\text{ot} \wedge \mathbf{S}\text{moky} \Rightarrow \mathbf{F}\text{ire}) \\ \wedge(\mathbf{A}\text{larm\_beeps} \Rightarrow \mathbf{S}\text{moky}) \\ \wedge(\mathbf{F}\text{ire} \Rightarrow \text{s}\mathbf{W}\text{itch\_on\_sprinklers})\end{array}\right\} \overset{?}{\models} \neg\,\text{s}\mathbf{W}\text{itch\_on\_sprinklers} \Rightarrow \neg\mathbf{F}\text{ire}$$

Example

$$\left.\begin{array}{l} (\mathbf{H}\text{ot} \wedge \mathbf{S}\text{moky} \Rightarrow \mathbf{F}\text{ire}) \\ \wedge(\mathbf{A}\text{larm\_beeps} \Rightarrow \mathbf{S}\text{moky}) \\ \wedge(\mathbf{F}\text{ire} \Rightarrow \text{s}\mathbf{W}\text{itch\_on\_sprinklers}) \end{array}\right\} \models^{?} \neg\, \text{s}\mathbf{W}\text{itch\_on\_sprinklers} \Rightarrow \neg\mathbf{F}\text{ire}$$

Abbreviations:

$$((H \wedge S \Rightarrow F) \wedge (A \Rightarrow S) \wedge (F \Rightarrow W)) \models^{?} (\neg W \Rightarrow \neg F)$$

Truth Table

. . . gives a truth value for all possible interpretations.

H	S	F	A	W	$((H \wedge S \Rightarrow F) \wedge (A \Rightarrow S) \wedge (F \Rightarrow W))$	$\neg W \Rightarrow \neg F$
T	T	T	T	T	T	T
T	T	T	T	F	F	F
T	T	T	F	T	T	T
T	T	T	F	F	F	F
T	T	F	T	T	F	T
T	T	F	T	F	F	T
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$

Truth Table

. . . gives a truth value for all possible interpretations.

H	S	F	A	W	$((H \land S \Rightarrow F) \land (A \Rightarrow S) \land (F \Rightarrow W))$	$\neg W \Rightarrow \neg F$
T	**T**	**T**	**T**	**T**	**T**	**T**
T	*T*	*T*	*T*	*F*	*F*	*F*
T	**T**	**T**	**F**	**T**	**T**	**T**
T	*T*	*T*	*F*	*F*	*F*	*F*
T	*T*	*F*	*T*	*T*	*F*	*T*
T	*T*	*F*	*T*	*F*	*F*	*T*
...	...	...	...	...	...	...

Inference

- $KB \vdash_i \alpha$ = sentence α can be derived from KB by procedure i
- *Soundness*: i is sound if
  whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$
- *Completeness*: i is complete if
  whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$

That is, the procedure will answer any question whose answer follows from what is known by the KB.

Inference Example

$$\frac{\text{fire} \qquad \text{fire} \Rightarrow \text{switch\_on\_sprinklers}}{\text{switch\_on\_sprinklers}}$$

Proof Rules

- Stating that B follows (or is provable) from $A_1, \ldots A_n$ can be written

$$\frac{A_1, \ldots A_n}{B}$$

Modus Ponens

This well known proof rule is called *modus ponens*, i.e. in general

$$\frac{A \Rightarrow B, \;\; A}{B}$$

where A and B are any WFF.

$\wedge$-elimination

- Another common proof rule, known as $\wedge$-elimination is

$$\frac{A \wedge B}{A} \quad \text{or} \quad \frac{A \wedge B}{B}$$

The first of these can be read *if A and B hold (or are provable or true) then A must also hold*.

Example

From $r \wedge s$ and $s \Rightarrow p$ can we prove p, i.e. show $r \wedge s, s \Rightarrow p \vdash p$?

1.	$r \wedge s$	[Given]	
2.	$s \Rightarrow p$	[Given]	
3.	s	[1 $\wedge$ elimination]	$\dfrac{r \wedge s}{s}$
4.	p	[$2, 3$ modus ponens]	$\dfrac{s \Rightarrow p, s}{p}$

$\vee$-introduction

Another proof rule, known as $\vee$-introduction is

$$\frac{A}{A \vee B} \quad \text{or} \quad \frac{A}{B \vee A}$$

The first of these can be read *if A holds (or are provable or true) then $A \vee B$ must also hold*.

Proof Theory

- Reasoning about statements of the logic without considering interpretations is known as *proof theory*.
- *Proof rules* (or inference rules) show us, given true statements how to generate further true statements.
- *Axioms* describe 'universal truths' of the logic.
  - Example $\vdash p \vee \neg p$ is an axiom of propositional logic.
- We use the symbol $\vdash$ denoting *is provable* or *is true*.
- We write $A_1, \ldots A_n \vdash B$ to show that B is provable from $A_1, \ldots A_n$ (given some set of inference rules).

Proofs

- Let $A_1, \ldots, A_m, B$ be well-formed formulae
- There is a proof of B from $A_1, \ldots, A_m$ iff there exists some sequence of formulae

$$C_1, \ldots, C_n$$

such that $C_n = B$, and each formula C_k, for $1 \leq k < n$ is either an axiom or one of the formulae $A_1, \ldots, A_m$, or else is the conclusion of a rule whose premises appeared earlier in the sequence.

Example

From $p \Rightarrow q$, $(\neg r \vee q) \Rightarrow (s \vee p)$, q can we prove $s \vee q$?

1.	$p \Rightarrow q$	[Given]
2.	$(\neg r \vee q) \Rightarrow (s \vee p)$	[Given]
3.	q	[Given]
4.	$s \vee q$	[$3, \vee$ introduction]

- Think how much work we would have had to do to construct a truth table to show $((p \Rightarrow q) \wedge ((\neg r \vee q) \Rightarrow (s \vee p)) \wedge q) \models (s \vee q)$!

Exercise

Show r from $p \Rightarrow (q \Rightarrow r)$ and $p \wedge q$ using the rules we have been given so far. That is prove

$$p \Rightarrow (q \Rightarrow r), p \wedge q \vdash r.$$

Soundness and Completeness

- Let $A_1, \ldots A_n, B$ be well formed formulae and let

$$A_1, \ldots A_n \vdash B$$

denote that B is derivable from $A_1, \ldots A_n$.

- Informally, soundness involves ensuring our proof system gives the *correct* answers.
  - **Theorem**(Soundness) If $A_1, \ldots A_n \vdash B$ then $A_1 \wedge \ldots \wedge A_n \models B$
- Informally, completeness involves ensuring that *all* formulae that should be able to be proved can be.
  - **Theorem**(Completeness) If $A_1 \wedge \ldots \wedge A_n \models B$ then $A_1, \ldots A_n \vdash B$.

More about Soundness & Completeness

- Example: An unsound (bad) inference rule is

$$\frac{A, B}{C}$$

- Using this rule from any p and q we could derive r yet $p \wedge q \models r$ does not hold.

Is Natural Deduction Complete?

- The set of rules modus ponens and $\wedge$ elimination is incomplete:
  Without $\vee$-introduction we cannot do the proof on page 23 yet

$$((p \Rightarrow q) \wedge ((\neg r \vee q) \Rightarrow (s \vee p)) \wedge q) \models (s \vee q).$$

Comments

- We haven't shown a full set of proof rules but just some examples.
- For a full set of proof rules look for *Natural Deduction* in a logic or AI book.
    - More than 10 rules
    - Intricate proofs (indirect proofs, reductio ad absurdum, etc)
- Note, at any step in the proof there may be many rules which could be applied. May need to apply search techniques, heuristics or strategies to find a proof.
- Getting computers to perform proof is an area of AI itself known as *automated reasoning*.

Summary

- We've discussed proof or inference rules and axioms.
- We've given examples of the proof rules $\wedge$-introduction, modus ponens and $\vee$-elimination.
- We've given some example proofs.

Validity, Satisfiability, and Entailment

Implications for Knowledge Representation

- *Deduction Theorem*:
  $KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid
- Or,...
  $KB \models \alpha$ if and only if $(KB \wedge \neg\alpha)$ is unsatisfiable
  - *reductio ad absurdum*

For propositional, predicate and many other logics

Resolution

- Resolution is a proof method for classical propositional and first-order logic.
- Given a formula φ resolution will decide whether the formula is *unsatisfiable* or not.
- Resolution was suggested by Robinson in the 1960s and claimed it to be *machine oriented* as it had only one rule of inference.

Resolution Method

The method involves:-

- translation to a normal form (CNF);
- At each step, a new clause is derived from two clauses you already have
- Proof steps all use the same rule
  - resolution rule;
- repeat until false is derived or no new formulae can be derived.

We first introduce the method for propositional logic and then extend it to predicate logic.

Resolution Rule

- Each A_i is known as a *clause* and we consider the set of clauses $\{A_1, A_2, \ldots, A_k\}$
- The (propositional) resolution rule is as follows.

$$\frac{\begin{array}{c} A \vee p \\ B \vee \neg p \end{array}}{A \vee B}$$

- $A \vee B$ is called the *resolvent*.
- $A \vee p$ and $B \vee \neg p$ are called *parents of the resolvent*.
- p and $\neg p$ are called *complementary literals*.
- Note in the above A or B can be empty.

Resolution applied to Sets of Clauses

Show by resolution that the following set of clauses is unsatisfiable.

$$\{p \vee q, p \vee \neg q, \neg p \vee q, \neg p \vee \neg q\}$$

1.	$p \vee q$	
2.	$p \vee \neg q$	
3.	$\neg p \vee q$	
4.	$\neg p \vee \neg q$	
5.	p	$[1, 2]$
6.	$\neg p$	$[3, 4]$
7.	**false**	$[5, 6]$

Resolution algorithm

Proof by contradiction, i.e. show that $KB \wedge \alpha$ unsatisfiable

```
function PL-RESOLUTION(KB, α) returns true or false
    inputs: KB, the knowledge base, a sentence in propositional logic
            α, the query, a sentence in propositional logic

    clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
    new ← { }
    loop do
        for each C_i, C_j in clauses do
            resolvents ← PL-RESOLVE(C_i, C_j)
            if resolvents contains the empty clause then return true
            new ← new ∪ resolvents
        if new ⊆ clauses then return false
        clauses ← clauses ∪ new
```

Full Circle Example

- Using resolution show

$$((q \wedge p) \Rightarrow r) \models (\neg p \vee \neg q \vee r)$$

- show that

$$((q \wedge p) \Rightarrow r) \wedge \neg(\neg p \vee \neg q \vee r)$$

  is unsatisfiable
  - translate to CNF.
  - apply the resolution algorithm

1. Transformation to CNF

$$
\begin{aligned}
& ((q \wedge p) \Rightarrow r) \wedge \neg(\neg p \vee \neg q \vee r) \\
\equiv \quad & (\neg(q \wedge p) \vee r) \wedge \neg(\neg p \vee \neg q \vee r) \\
\equiv \quad & ((\neg q \vee \neg p) \vee r) \wedge \neg(\neg p \vee \neg q \vee r) \\
\equiv \quad & (\neg q \vee \neg p \vee r) \wedge (\neg\neg p \wedge \neg\neg q \wedge \neg r) \\
\equiv \quad & (\neg q \vee \neg p \vee r) \wedge (p \wedge q \wedge \neg r) \\
\equiv \quad & (\neg q \vee \neg p \vee r) \wedge p \wedge q \wedge \neg r
\end{aligned}
$$

2. Resolution

1. $\neg q \vee \neg p \vee r$
2. p
3. q
4. $\neg r$

Finally apply the resolution rule.

5. $\neg q \vee r$ $\quad [1, 2]$
6. r $\quad [5, 3]$
7. **false** $\quad [4, 6]$

Discussion

- As we have derived false then that means the formula was unsatisfiable.
- Note if we couldn't obtain false that means the original formula was satisfiable.

Comment I

- The resolution rule is derived from a generalisation of the modus ponens inference rule given below.

$$\frac{\begin{array}{l} P \\ P \Rightarrow B \end{array}}{B}$$

- This can be generalised to

$$\frac{\begin{array}{l} A \Rightarrow P \\ P \Rightarrow B \end{array}}{A \Rightarrow B}$$

Comment II

- Resolution restricts the P so it is a proposition, i.e.

$$\frac{\begin{array}{c} A \Rightarrow p \\ p \Rightarrow B \end{array}}{A \Rightarrow B}$$

- Given a set of clauses $A_1 \wedge A_2 \ldots A_k$ to which we apply the resolution rule, if we derive **false** we have obtained $A_1 \wedge \ldots \wedge$ **false** which is equivalent to **false**. Thus the set of clauses is unsatisfiable.

Theoretical Issues

- Resolution is *refutation complete*. That is if given an unsatisfiable set of clauses the procedure is guaranteed to produce **false**.
- Resolution is *sound*. That is if we derive **false** from a set of clauses then the set of clauses is unsatisfiable.
- The resolution method *terminates*. That is we apply the resolution rule until we derive false or no new clauses can be derived and will always stop.

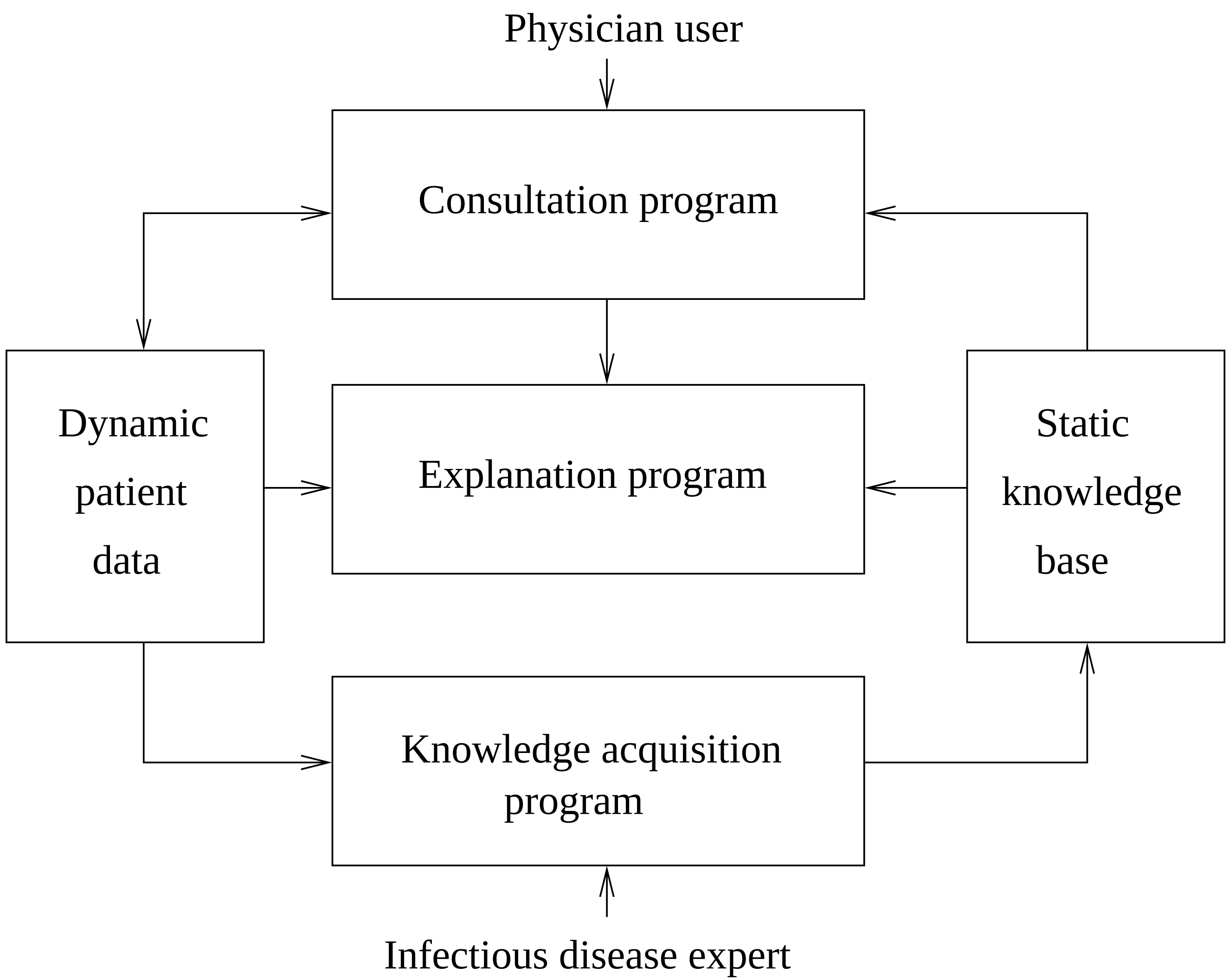
Automated Reasoning

- The resolution proof method may be automated, i.e. carried out by a computer program.
- Theorem provers based on resolution have been developed eg Otter, Spass.
- The topic of automated reasoning lies within the area of AI.
- In the Logic and Computation research group we are interested in automated reasoning, in particular related to resolution.

MYCIN

- One of the most important expert systems developed was MYCIN
- This is a system which diagnoses and treats bacterial infections of the blood.
- The name comes from the fact that most of the drugs used in the treatment of bacterial infections are called:
  - “Something”mycin
- MYCIN is intended to be used by a doctor, to provide advice when treating a patient.
- The idea is that MYCIN can extend the expertise of the doctor in some specific area.

Architecture of MYCIN



Rules in MYCIN

- Rules in MYCIN are of the form:

```
IF
1. The gram stain of the organism is gramneg,
   and
2. The morphology of the organism is rod, and
3. The aerobicity of the organism is anaerobic
THEN
there is suggestive evidence (0.6) that the
identity of the organism is bacteroides.
```

- Note this is not the internal form of the rule!

Another Example

```
IF
1. The identity of the organism is not known
   with certainty, and
2. The gram stain of the organism is gramneg,
   and
3. The morphology of the organism is rod, and
4. The aerobicity of the organism is aerobic
THEN
there is strongly suggestive evidence (0.8)
that the identity of the organism is
enterobactericeae.
```

- The antecedent is allowed to be a mixture of AND and OR conditions.

A Rule with OR Conditions

```
IF
1. The therapy under consideration is:
       cephalothin, or
       clindamycin, or
       erythromycin, or
       lincomycin, or
       vancomycin
   and
2. Meningitis is a diagnosis for the patient
THEN
It is definite that the therapy under
consideration is not a potential therapy.
```

- Note that we have rules about treatment as well as about diagnosis.

OR in the Consequent of a Rule

```
IF
The identity of the organism is bacteroides
THEN
I recommend therapy chosen from among
the following drugs:

1. clindamycin
2. chloramphenicol
3. erythromycin
4. tetracycline
5. carbenecillin
```

Certainty Factors I

- MYCIN uses certainty factors (CFs), values between $+1$ and -1 to show how *certain* its conclusions are, a positive value showing suggestive evidence *for* the conclusion and a negative value *against* the conclusion.
- For example, data of a particular organism relating to its Gram stain, morphology and aerobicity may be as follows.

```
GRAM = (GRMNEG 1.0)
MORPH = (ROD 0.8)
AIR = (ANAEROBIC 0.7)
```

Certainty Factors II

- MYCIN has its own way to calculate further CFs. The certainty of a conjunction is the minimum of individual certainties.
- Applying our first example rule to this data, the certainty of all three premises holding is 0.7 so the conclusion of bacteroides for this data would have a CF of $0.7 \times 0.6 = 0.42$.
- Note CFs do not correspond with probability theory (but computation of CFs more tractable).

How MYCIN Works

- MYCIN has a four stage task:
  - decide which organisms, if any, are causing significant disease.
  - determine the likely identity of the significant organisms.
  - decide which drugs are potentially useful.
  - select the best drug, or set of drugs.
- The control strategy for doing this is coded as meta-knowledge.

The Relevant Rule

```
IF
1. There is an organism which requires
   therapy, and
2. Consideration has been given to possible
   other organisms which require therapy
THEN
1. Compile a list of possible therapies, and
2. Determine the best therapy.
ELSE
Indicate that the patient does not require
therapy.
```

Forward & Backward Chaining

"How do I decide if there is an organism requiring therapy? Well, Rule 90 tells me that organisms associated with significant disease require therapy. But I don't know about any organisms, so I'll ask the user . . . now I can apply RULE 90 to each of these . . . but I need to know if the organism is significant. Now I have a set of rules which tell me whether this is the case. Let's see, RULE 38 says that an organism from a sterile site is significant. I don't have any rules for saying if the site was sterile, so I'll ask the user . . . "

How MYCIN Works II

- So MYCIN starts by trying to apply the control rule, and this generates *sub-goals*.
- The first of these is to determine if there is an organism which needs to be treated.
- This generates another sub-goal; whether the organism is significant.
- This provokes a question to the user.
- The answer allows other rules to be fired, and these lead to further questions.
- Eventually the IF part of the control rule is satisfied, and the THEN part compiles a list of drugs, and chooses from it.

A consultation with MYCIN

- So, MYCIN chains back from its overall goal of deciding what organisms need treatment until it finds it lacks information, and then asks the user for it.
- Using MYCIN is thus an interactive process:
  1. MYCIN starts running.
  2. MYCIN asks a question.
  3. The user answers it.
  4. MYCIN asks another question.
  5. . . .

Example

```
1) Patient's name: (first-last)
** FRED BRAUN
2) Sex
** M
3) Age
** 55
4) Are there any illnesses for Fred Braun
   which may be related to the present illness,
   and from which organisms have been grown
   in the microbiology laboratory?
** Y
-------- CULTURE-1 ----------
5) From what site was the specimen for
   CULTURE-1 taken?
** BLOD
= BLOOD
6) Please give the date and time
   when CULTURE-1 was obtained.
** JUN 20 1977
```

Example (contd.)

```
The first organism isolated from the blood
culture of 20-JUN-77 (CULTURE-1) will be
referred to as:

-------- ORGANISM-1 ---------

7) Enter the laboratory-
reported identity of
ORGANISM-1
** UNKNOWN
8) The stain (gram or Ziehl-
Neelsen acid-fast) of
ORGANISM-1
** NEG
9) Is ORGANISM-1 rod or coccus
** ROD
10) What is the form of the
individual organisms?
** FUSIFORM
...
...
```

Example (contd.)

```
...
It is important to cover for the
following probable infections and
associated organisms:

INFECTION-1 is BACTEREMIA

<ITEM-1> E.COLI
<ITEM-2> KLEBSIELLA
<ITEM-3> ENTEROBACTER
<ITEM-4> KLEBSIELLA-PNEUMONIA
```

Explanation in MYCIN

- An important feature of any expert system is that it can explain how it comes up with conclusions.
- This allows users to understand what the system is doing, and so see that it is doing sensible things.
- One of the advantages of rules is that they make it possible to generate good explanations.
- Essentially the explanation is just a presentation of the *proof tree*.
- We can illustrate this with an example explanation dialogue from MYCIN.

Example

```
34) Was penicillinase added to CULTURE-1?
** Why?

[3.0] This will aid in determining whether
 ORGANISM-1 is a contaminant.
 It has already been established that:
 [3.1] the site of  CULTURE-1 is blood, and
 [3.2] the gram stain of ORGANISM-1 is grampos

Therefore if
 [3.3] penicillinase was added to this
       blood culture
then
 there is weakly suggestive evidence
   that ORGANISM-1 is a contaminant [RULE 39]
```

Example (contd.)

```
** Why?

[4.0] ... in order to determine whether there
 is significant disease associated with
 ORGANISM-1.

 It has already been established that:
  [4.1] this blood culture was
    taken from a sterile source

 Therefore if
  [4.2] ORGANISM-1 is not a contaminant
 then
  there is strongly suggestive evidence
  that there is significant disease
  associated with ORGANISM-1
```

How Explanation Works

- When asked “WHY”, MYCIN:
  - uses the current rule.
- If asked “WHY” again, MYCIN
  - uses the rule which caused the current rule to fire
- Remember that MYCIN is chaining backwards from the overall goal of showing significant disease.
- This explanation can be continued by asking more “WHY”s.

Evaluation of MYCIN

- Evaluated by comparing its performance to 8 members of Stanford medical school: 5 faculty members, one research fellow in infectious diseases, one physician and one student.
- They were given 10 randomly selected case histories and asked to come up with diagnoses and recommendations.
- These given to 8 independent experts in infectious disease to evaluate (scoring as acceptable or not).

Evaluation of MYCIN (contd.)

- MYCIN performed as well as any of the Stanford medical team and considerably better than the physician or student.
- MYCIN has never been used in clinical practice due to:
  - expense of computing power required at the time
  - the amount of time and typing required for a session
  - incompleteness of the knowledge base.