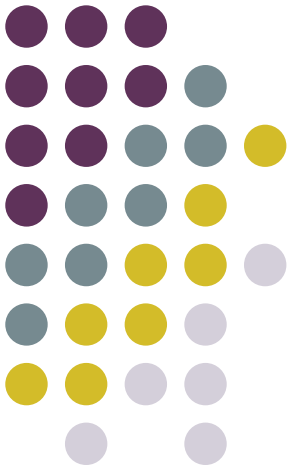
An Introduction to Expert Systems

Kristian Guillaumier 2006, 07



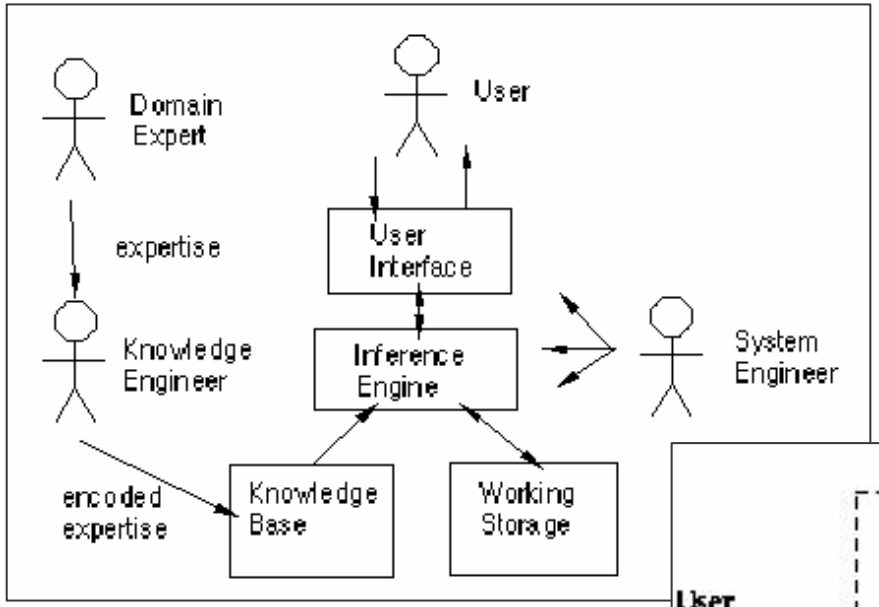
What is an Expert System?

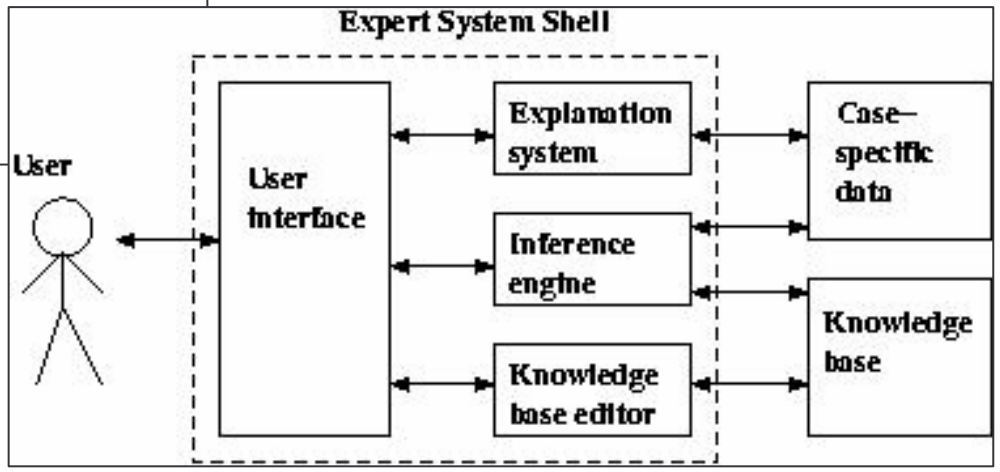
Portions from http://www.macs.hw.ac.uk/~alison/ai3notes/chapter2_5.html

- Informally, a computer system used to provide advice, suggestions and diagnosis in a similar way a domain expert would.
- Involves (the task of a *Knowledge Engineer*):
  - Gathering/Extracting knowledge (*Knowledge Acquisition*).
    - This knowledge is mostly heuristic rather than undeniable facts.
    - E.g. Interviewing a group of medical specialists.
    - Can be very difficult to express certain ideas (gut feeling???).
    - Iterative refinement.
    - Knowledge must be inputted in such a way that eventually the system can explain its reasoning process.
    - Rules are commonly used. If-Then-Else with probabilities on outcomes.
  - These rules are used during an inference process – Deriving a conclusion from stored knowledge.

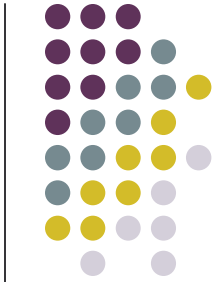
Architecture

Portions from http://www.amzi.com/ExpertSystemsInProlog/01introduction.htm
Portions from http://www.macs.hw.ac.uk/~alison/ai3notes/chapter2_5.html



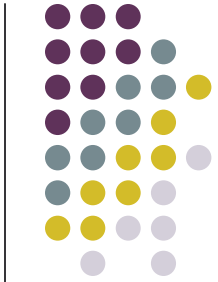


Architecture



- User interface.
- Explanation.
- Working memory (facts and derived facts).
- Inference engine.

An Example (1)



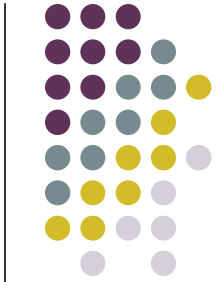
1. If (EngineGettingPetrol) AND (EngineTurns)
   Then (ProblemWithSparkPlugs)
2. If (NOT EngineTurns) AND (NOT LightsTurnOn)
   Then (ProblemWithBattery)
3. If (NOT EngineTurns) AND (LightsTurnOn)
   Then (ProblemWithStarter)
4. If (PetrolInFuelTank)
   Then (EngineGettingPetrol)

- Propositional logic is used to describe rules.
- 3 diagnosable problems: Spark plugs, battery, starter.
- In a simple goal-directed approach, we would try to find out whether we can prove any of the problems.

An Example (2)

- First we try to prove that there is a problem with the spark plugs.
  - Now the goal is to determine if the engine is getting fuel (1st part) and whether the engine turns (2nd part).
  - To prove the 1st part we need to use rule 4.
  - To prove rule 4 we need to know if there is petrol in the fuel tank.
  - There is no rule (1 to 4) that can help to determine this so the system asks the user “Is there petrol in the fuel tank?”.
  - If the answer is yes this new bit of information is stored in working memory (there is petrol in the tank) so that the user is not asked this again.
  - Rule 4 is now proven (the engine is getting petrol) and we now have to prove the 2nd part (whether the engine turns).
  - There is no rule that can help is to determine this so we ask the user “Does the engine turn?”. Again, the answer is recorded.
  - If the answer is no (1st part == true, 2nd part == false) then the AND clause will invalidate the first possibility (i.e. spark plug problem == false).
  - The next possible problem is examined (problem with battery) and so on…

An Example (3)



- Example transcript:

  System: Is there petrol in the fuel tank?

  User: Yes.

  System: Does the engine turn?

  User: No.

  System: Do the lights turn on?

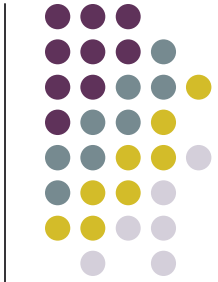
  User: No.

  System: The problem is the battery.

  User: Why?

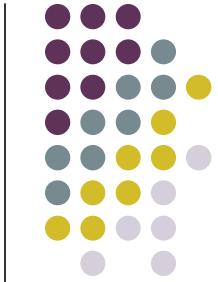
  System: This follows from rule 2 because the engine does not turn and the lights don't turn…

Case Study: Mycin



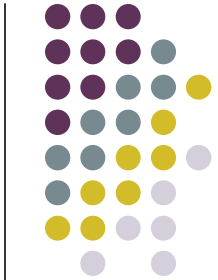
- Developed in early 1970's at Stanford.
- Diagnose and recommend treatment for blood infections.
- A quasi fool-proof diagnosis involves growing a culture of the infecting bacteria and studying them. This takes days and the patient can die in the meantime.
- Specialists came up with rules of thumb about likely problems from the information they gathered and provide a "covering" (broad range) treatment.
- Mycin was developed to understand how experts come to these conclusions and to help junior, non-specialist doctors to come to these conclusions too.

Chaining



- **Forward chaining**
  (Data → Rules → Conclusion):
  - A=1, B=2.
  - If A==1 AND B==2 Then C=3.
  - If C==3 Then D=4.
  - Therefore D==4.
- Backward chaining:
  (Goal → Rules → Sub goals)
  - Is D==4?
  - It is the conclusion of ***If C==3 Then D=4***.
  - Is C==3? becomes a sub goal.
  - It is the conclusion of ***If A==1 AND B==2 Then C=3***.
  - Is A==1? and Is B==2? become sub goals.
  - These are given as facts so C==3? becomes a fact too.
  - Since C=3 is a fact, then D==4 becomes a fact to – ready.

Uncertainty and Priorities

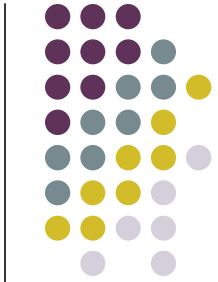


- Sometimes the conclusion might not be 100% certain from the antecedents.
- This could stem from vagueness on the knowledge engineer's part or the problem itself – impossible to be definite.
- One way is to associate a numeric value that scores the certainty of a conclusion.
- "If Fritz is green then he is probably a frog".
- Priorities:
  - If Fire Then Run
  - If Fire AND YourClothesBurning Then PutOutFire

Inference Engine

- Tries to derive conclusions from a rule base.
- Architecture:
  - Rule base.
  - Working memory.
- Sequence:
  - Match rules.
  - Select rules.
  - Execute rules.
- Match rules:
  - Find all the rules that are satisfied by the working memory.
  - These rules are the *conflict set*.
  - The conflict set is transferred to the “select rules” state.

Some Propositional Logic Rules Modus Ponens



- If it is raining then I am wet. It is raining. Therefore I am wet.
- P → Q, P; Q
- Expressed as a tautology (P → Q AND P) → Q.

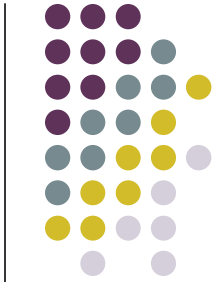
P	Q	P → Q	~ AND P	Result
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

Some Propositional Logic Rules Modus Tollens

- P → Q, !Q; !P
- (P → Q AND !Q) → !P

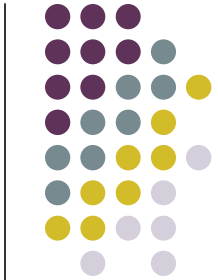
P	Q	P → Q	~ AND !Q	Result
T	T	T	F	T
T	F	F	F	T
F	T	T	F	T
F	F	T	T	T

Other Algebraic Properties



- Double negation: !(!P) == P.
- Chaining: A → B, B → C; A → C.
- De Morgan's Laws:
  - !(P AND Q) is equivalent to (!P OR !Q).
  - !(P OR Q) is equivalent to (!P AND !Q).

Resolution Preliminaries

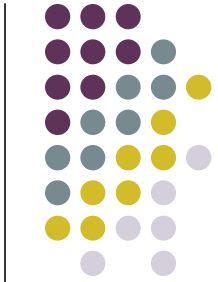


- Based on proof by refutation.
  - Deny the conclusion.
  - Show that there is a contradiction.
- Preliminaries:
  - A formula in conjunctive normal form (CNF) is defined as a conjunction of disjunctions that are literals:

    (A OR B OR C) AND (P OR Q OR R)
  - A literal cannot contain connectives. It is an atom or a negated atom.

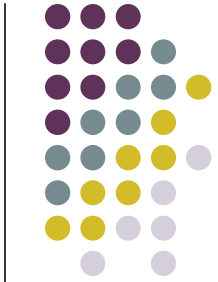
    (A OR B) AND (!P OR Q)
  - … is valid.
  - The terms in parenthesis are called clauses.
  - So a clause is true if any one literal is true.
  - A clause (A OR B) is sometimes written {A,B}

Resolution Preliminaries



- The goal of resolution is to infer a new clause (resolvent) from two other clauses (parent clauses).
- The resolvent is simpler than the parents.
- By continuing the process, a contradiction will be obtained (premise is proven) or the algorithm will terminate because no progress can be made.

Resolution



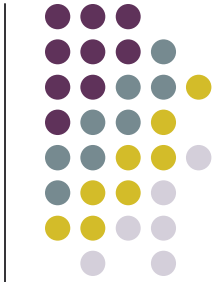
- Resolution is based on the following argument:

  A OR B

  A OR !B

  Therefore, A
- Proof:
  - Write the premises as (A OR B) AND (A OR !B).
  - The axiom of distribution says that P OR (Q AND R) is equivalent to (P OR Q) AND (P OR R).
  - Using this axiom, (A OR B) AND (A OR !B) becomes A OR (B AND !B) which is equivalent to A.
- In English:
  - Either B or !B must be false, so A must be true to satisfy the two clauses (A OR B), (A OR !B).

Resolution



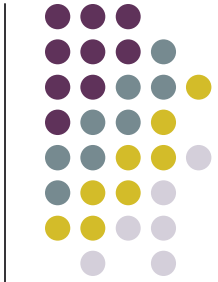
- Similarly it can be argued that

  P OR B

  Q OR !B

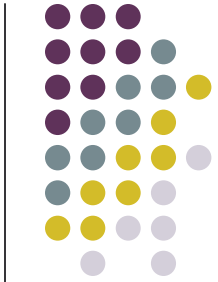
  Therefore, P OR Q
- Again,
  - Either B or !B must be false.
  - So either P or Q must be true.

Resolvents



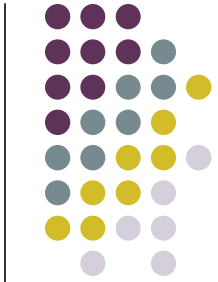
- Let C1 and C2 be two clauses.
- D is a resolvent of C1 and C2 iff:
  - A literal L exists in C1 and a literal !L exists in C2.
- The resolvent is the merging of
  C1-L and C2-!L.

Resolution Example



- Consider the proof for:
  A → B
  B → C
  C → D
  Therefore, A → D
- Convert A → D to disjunctive form:
  !A OR D
- Consider its negation:
  !(!A OR D)
- Which is equivalent to:
  A AND !D

Resolution Example

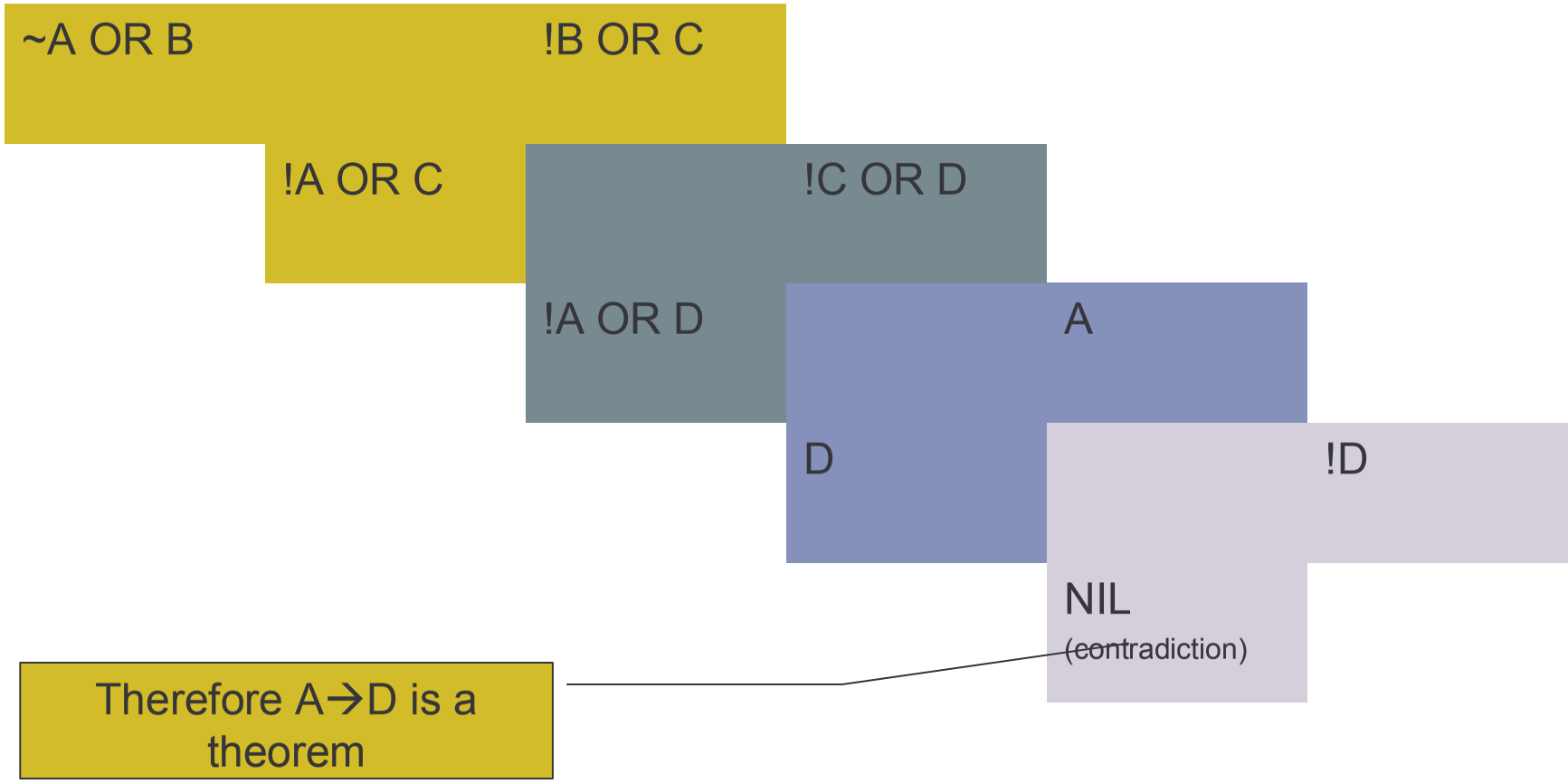


- Convert the premises to CNF:

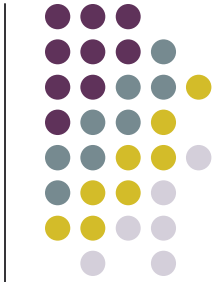
  (!A OR B) AND (!B OR C) AND (!C OR D)
- Add the negated conclusion:

  (!A OR B) AND (!B OR C) AND (!C OR D) AND A AND !D
- This is in CNF so it is suitable to be processed by resolution.
- Resolution can be illustrates with a resolution refutation tree diagram

Resolution Example



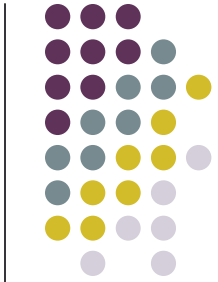
Another Example



- Socrates is a man.
- All men are mortal.
- Therefore, Socrates is a frog.

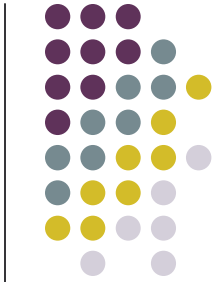
- Man(Socrates)
- Man(?) → Mortal(?)
- Frog(Socrates)
- Abbreviate to:
  - M
  - M → T
  - F

Another Example



- Clauses:
  - M
  - !M OR T
  - !F (negation of conclusion).
- Apply resolution:
  - M + (!M OR T) == T
- New KB:
  - M, !M OR T, !F, T
- Nothing left to resolve – theorem is false.

Resolution Algorithm



- Convert the knowledge base to clause form.
- We are trying to prove a theorem.
- Add the negation of the theorem (in clause form) to the clauses in the KB.
- LOOP: Search for 2 resolvable clauses in the KB.
  - If not found: theorem to be proven is false.
  - If found: resolve and add the resolvent to the KB.
- If a contradiction is reached:
  - Theorem is proven.
- Else:
  - Goto LOOP.

A Simple Expert System Cycle

- Create an agenda (matched rules) by seeing with rule LHS match the working memory.
- Select the rule in the agenda with the highest priority.
- Perform RHS action (update working memory), and remove the rule from the agenda.
- If working memory changed and a new rule fires, add it to the agenda.
- If not ready loop back.

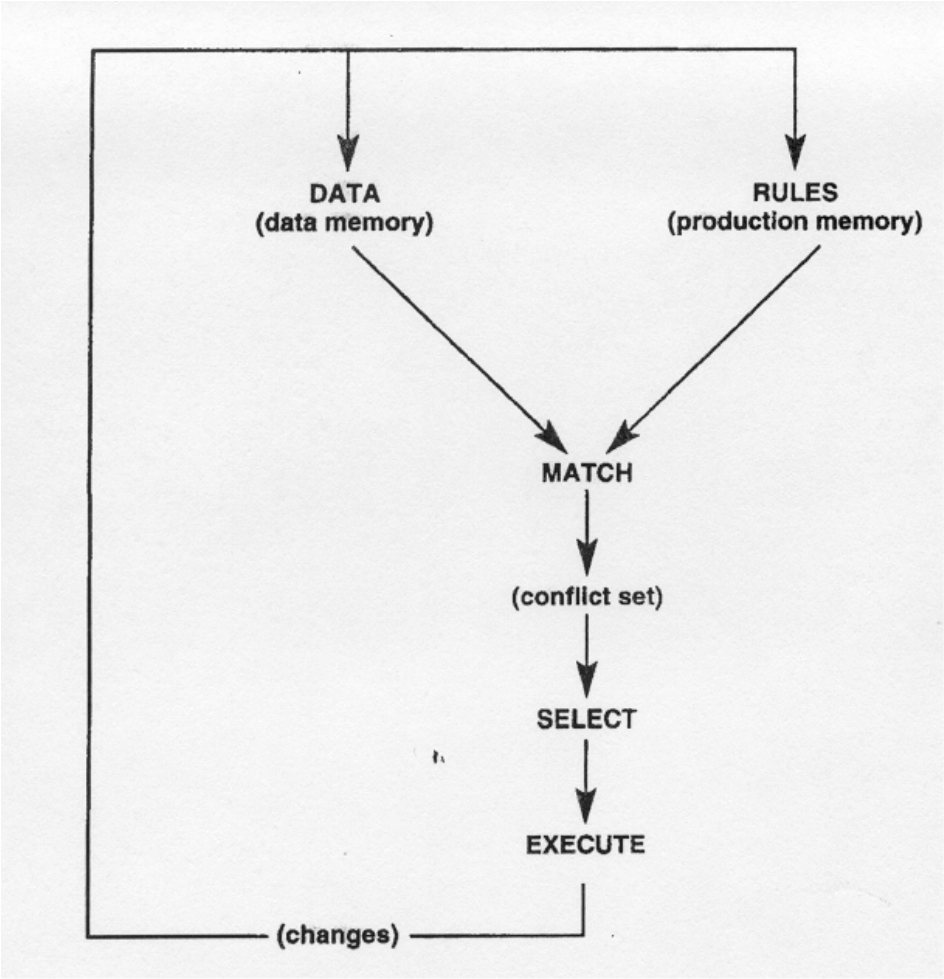
RETE

Portions from http://www.cis.temple.edu/~ingargio/cis587/readings/rete.html
Portions from http://www.cse.unsw.edu.au/~cs9416/04-RETE/rete.html

- A naïve expert system will iteratively try to match all rules against all facts to see which one matches. This is too slow.
- It reorganises rules to allow for more efficient matching.
- Creates a decision tree that combines all patterns (rule conditions) in the knowledge base.

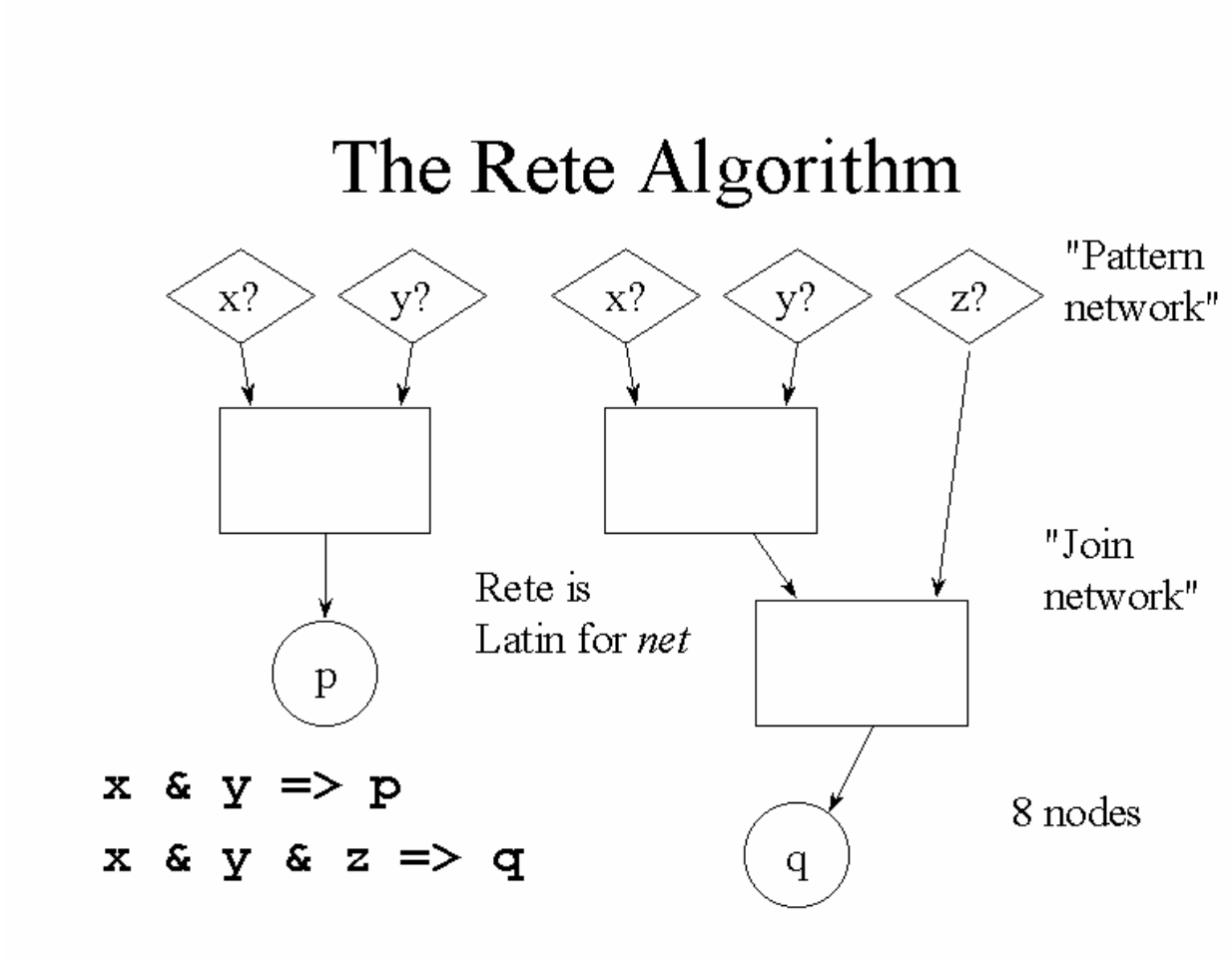
Simple Match

From Dr Dobbs



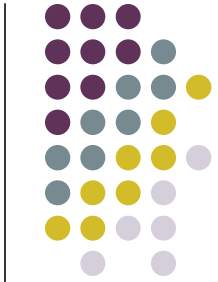
RETE Example (1)

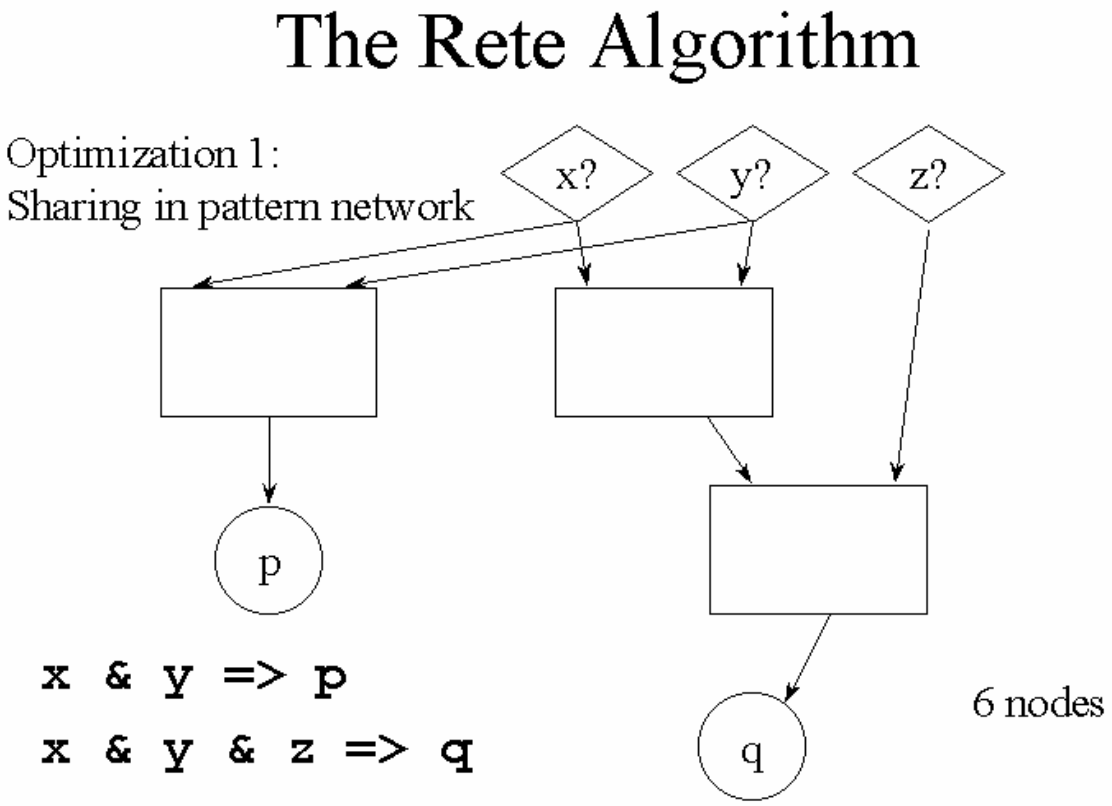
From http://aaaprod.gsfc.nasa.gov/teas/Jess/JessUMBC/sld010.htm



RETE Example (2)

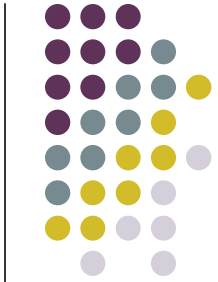
From http://aaaprod.gsfc.nasa.gov/teas/Jess/JessUMBC/sld010.htm

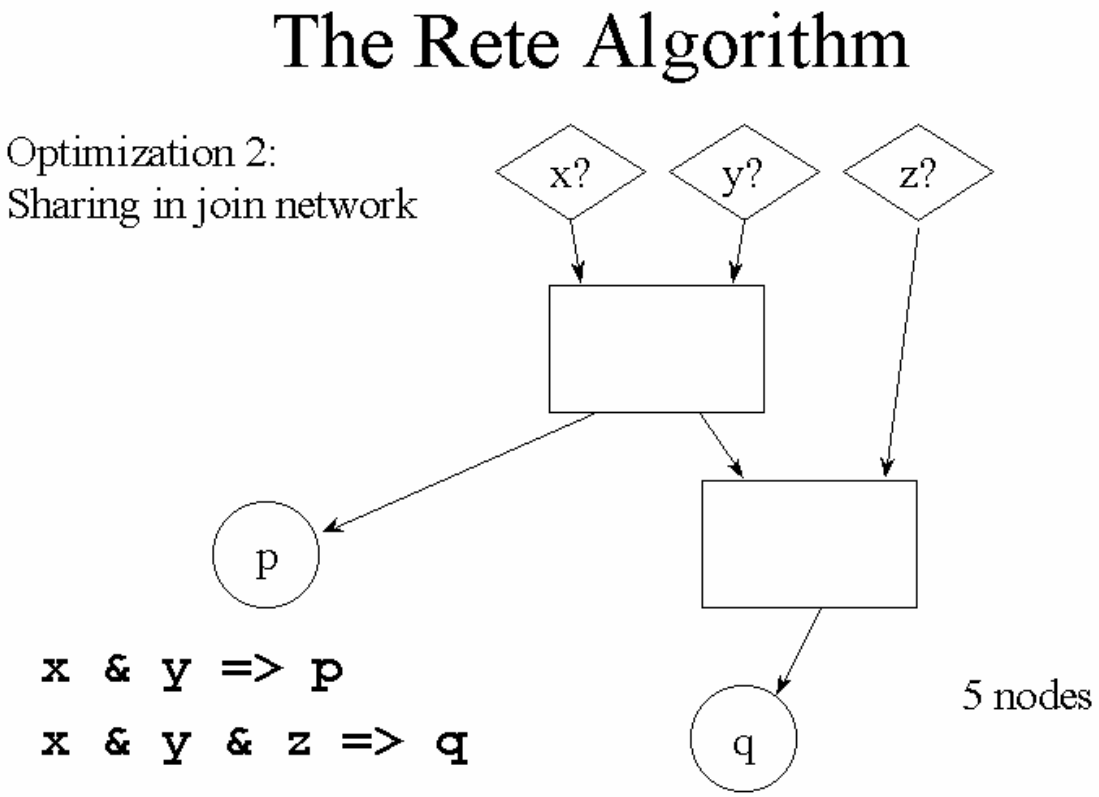




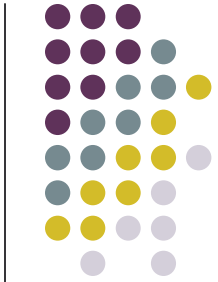
RETE Example (3)

From http://aaaprod.gsfc.nasa.gov/teas/Jess/JessUMBC/sld010.htm



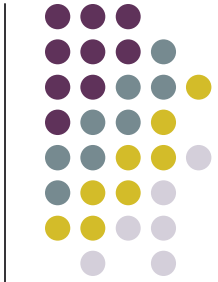


RETE



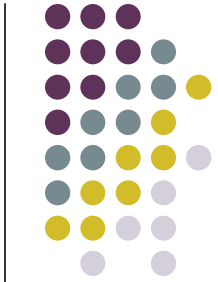
- Store (cache) list of rules that match or partially match the working memory.
- Matched rules (true) and unmatched rules (false) are not checked again – until the working memory is changed and only against the added or removed rules from working memory.

Some Advantages



- Availability and permanence.
- Cost.
- Explanation.
- Speed.

Some Disadvantages



- Who is liable in case of error (Mycin)?
- No scope for intuition.
- Domain knowledge may not be easy to extract to represent.