Decision Making: Rule-Based Systems

2016-06-23

OOB

- Decision Making:
  - N+1: Fuzzy
  - N-0: Production/Rule-based Systems
  - N-1: Planning
  - N-2: Trees
  - N-4: FSMs

Questions

1. How can we describe decision making?
2. What do the algorithms we’ve seen share?
3. What are the dimensions we tend to assess?
4. FSMs/Btrees: _____ :: Planning : ______
5. For the 2nd blank, we need m______s.
6. When is reactive appropriate? Deliberative?
7. What is the ‘hot-potato’ passed around (KE)?
8. H_______ have helped in most approaches.
9. Which approach should you use?

Questions

1. What are the 2 most “complex” decision making techniques we’ve seen?
2. What are their strengths? Weaknesses?
3. What is the key (insight) to their success?
4. What is typically necessary to support this insight (hint: used in Planning + RBS)?
5. What does Planning have that (forward chaining) RBS do not?
6. When do we need a communication mechanism?

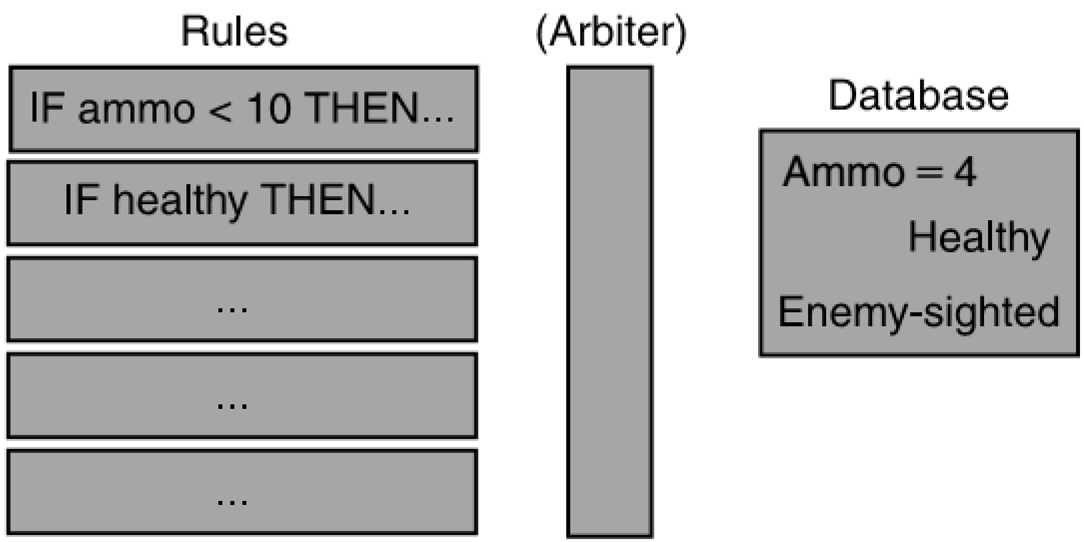
RULE-BASED SYSTEMS

Background

- Symbolic AI, “Expert Systems”
- Vanguard of AI research 70s + early 80s
- Used in some games, but not as common as FSMs or decision trees
  - Reputation for inefficiency + challenge to impl.
  - Similar behaviors achievable using Dtree/FSMs
- More robust than decision trees when worlds are unpredictable
- A form of reactive planning

Production/Rule System

- 2 part structure:
  - Facts (database of knowledge)
  - Rules (if/then constructs, with Boolean ops)
- Like a FSM, but triggers/effects are more general
- Basic idea:
  - Match: facts to if-part of rules ("pattern matching")
    - Rules with matching if's become activated ("triggered")
  - Arbitrate: Choose an active rule to "fire"
    - Can make change to facts or to world
  - Repeat



Millington Figure 5.46

```
If enemiesInSight > 0 and patrolling THEN
    remove( patrolling )
    add( attackNearest )
```

Comments

- It is like writing a program and then allowing the computer to decide which functions to call and when
- Forward vs. Backward chaining
  - B: theorem proving + planning
  - Authors never saw backward in games
- DB rewriting rules vs Condition-Action Rules
  - Rewrite Rules can change DB (+/- facts)
  - Typically only for AI specific knowledge (e.g. patrol)
  - Bias in GAI for condition-action rules (no rewrites)

Declarative Knowledge

- Stateable facts about the world
- (*attribute value*)
  - (Captain-weapon rifle)
- *value* can be nested knowledge
  - (Captain-weapon (rifle (ammo 36)))

DK: Facts

- Health(captain, 51)
- Health(Johnson, 38)
- Health(Sale, 42)
- Health(Whisker, 15)
- Holding(whisker, radio)
- Weapon(whisker, rifle)
- Weapon(johnson, pistol)
- Ammo(whisker, 36)

- Whisker
  - Health: 51
  - Holding: radio

- (captain
  (weapon (rifle (ammo 36) (clips 2)))
  (health 51)
  (position …)
)

  (radio (held-by whisker))

Procedural Knowledge

- Knowledge about how we *do* the things we do
- IF (some facts about the world) THEN do (some action)

PK: Rules

- IF whisker’s health < 15 AND Whisker holding radio THEN Whisker: Radio-call “help!” on radio

- IF whisker’s health = 0 AND whisker holding radio THEN
  - Remove(whisker holding radio)
  - Add(radio on ground)

- IF ?anyone health < 15 THEN …

Components

- Declarative knowledge (facts/KB)
- Procedural knowledge (actions)
- Selection knowledge (conditions, arbiter)
- Arbiter
  - First applicable (FIFO on input order)
  - Least recently used (LIFO on use order)
  - Random
  - Priority / Most specific conditions
  - Dynamic Priority system

Unification

- Binding of vars in logical statements
  – Same problem as in Planning
  – (?persn health 0-15) AND (?radio (heldby ?persn)
- Allows rules to match in many situations
  – See Russell & Norvig, Millington 5.8.7
- N is number of items in DB, M is number of clauses in pattern to match: O(nm), or maybe O(m $\log_2$ n), but generally O(n^m)

Simple Algorithm

```
def ruleBasedIteration( database, rules ):
  for rule in rules:
    bindings = []
    if rule.ifClause.matches( database, bindings ):
      rule.action( bindings ) #fire rule
      return #exit; we’re done for this iteration
  #if we get here, there’s no match; can do default
  #or do nothing
  return
```

RETE

- AI industry standard for rule matching
- Rule patterns represented in DAG
  - Pattern nodes, Join nodes, Rule Nodes
- Each path represents set of patterns for one rule
  - Fast matching (share evaluation)
  - Graceful updates (add/remove facts)
  - Determines which rules are active (all)
  - Millington & Funge cover very well

Rete Example

Swap Radio Rule:

IF

(?p1 (health < 15)) &&

(?p2 (health > 45)) &&

(radio (held-by ?p1))

THEN

remove(radio (held-by ?p1))

add(radio (held-by ?p2)

Change Backup Rule:

IF

(?p1 (health < 15)) &&

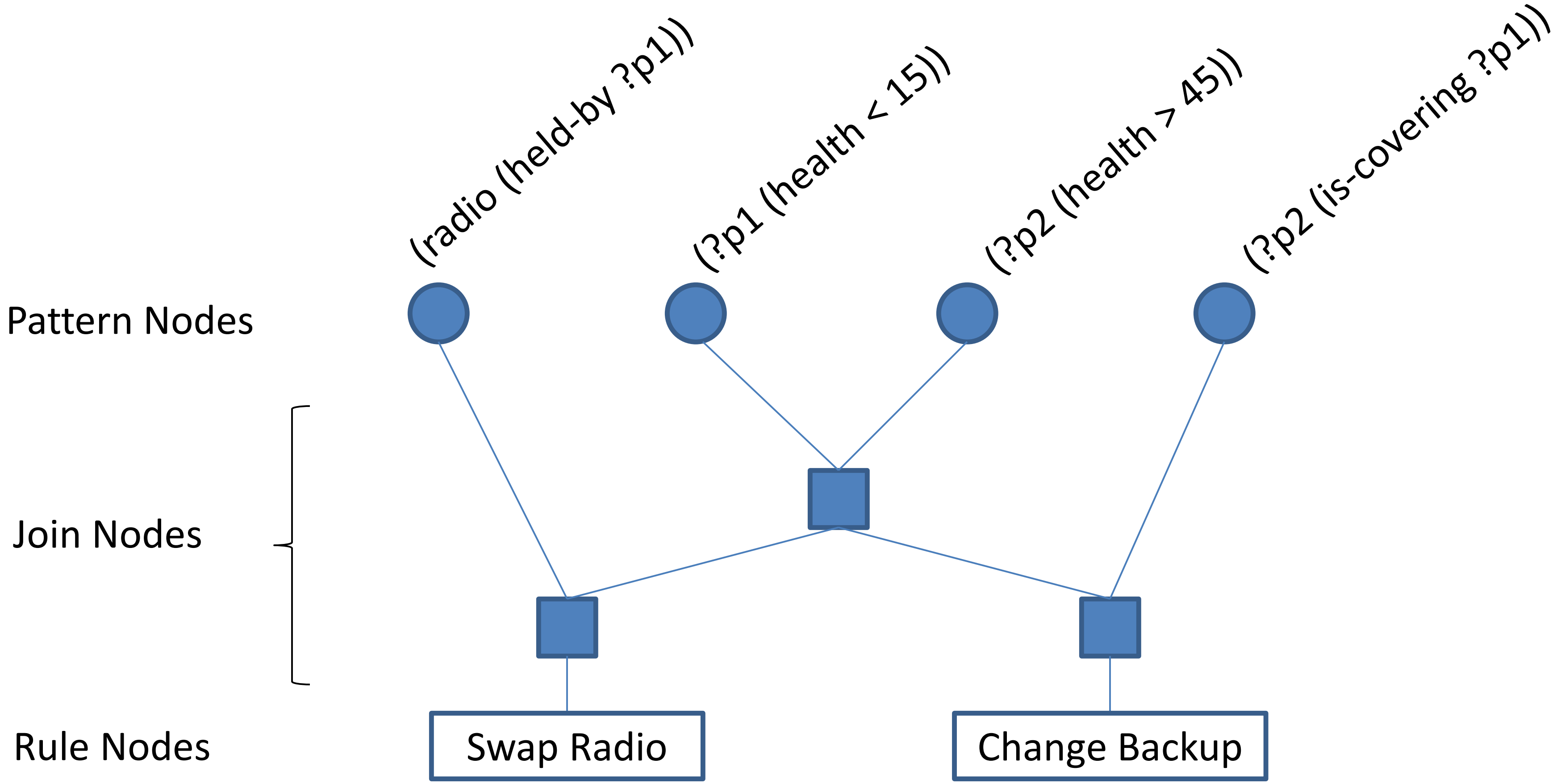
(?p2 (health > 45)) &&

(?p2 (is-covering ?p1))

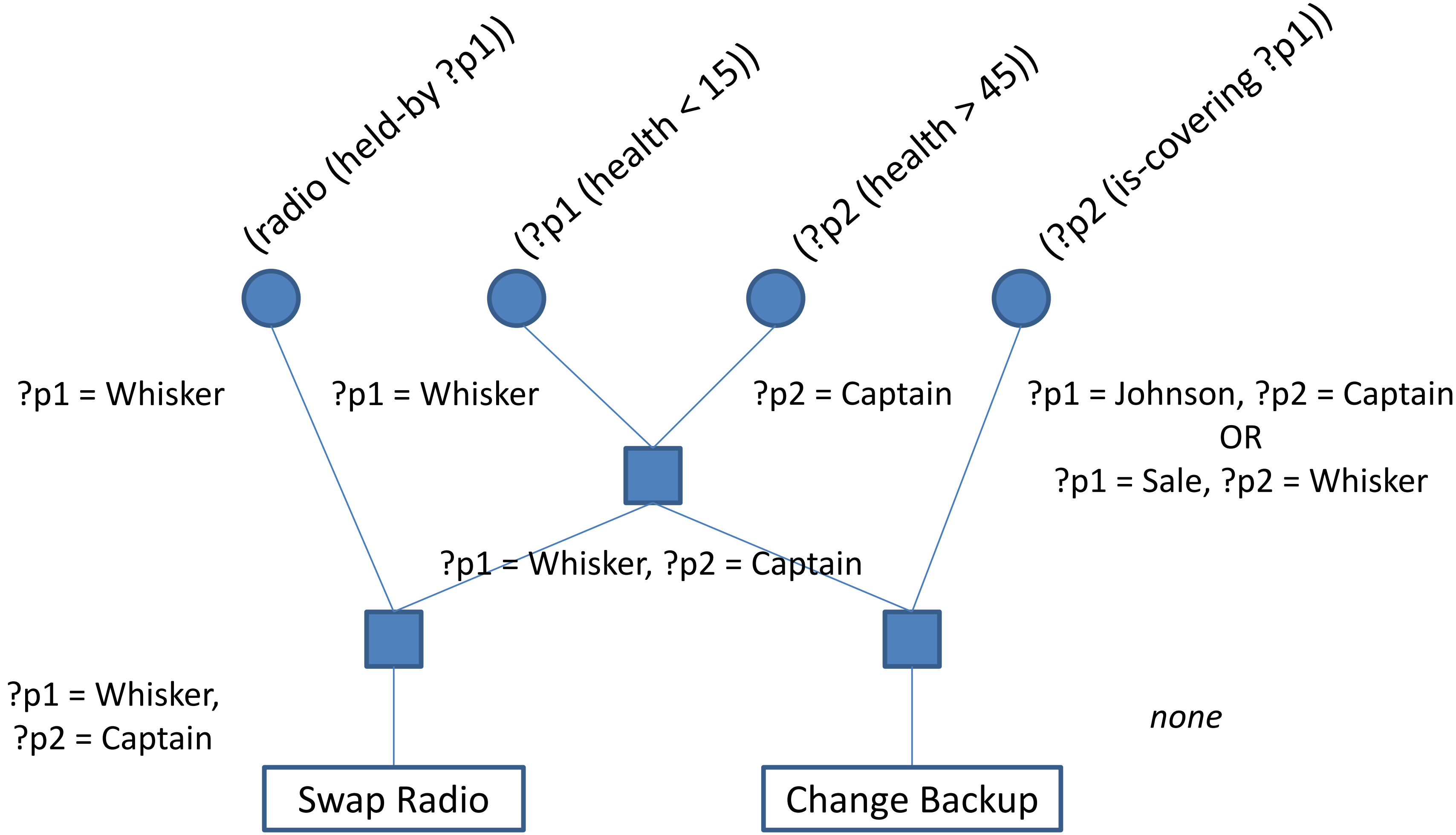
THEN

remove(?p2 (is-covering ?p1))

add(?p1 (is-covering ?p2))


(radio (held-by ?p1))
(?p1 (health < 15))
(?p2 (health > 45))
(?p2 (is-covering ?p1))
Pattern Nodes
Join Nodes
Rule Nodes
Swap Radio
Change Backup

(Captain (health 57) (is-covering Johnson))
(Johnson (health 38))
(Sale (health 42))
(Whisker (health 15) (is-covering Sale))
(Radio (held-by Whisker))



Pattern Nodes

- Database fed into top of network
- Pattern nodes find matches in database and pass them down to join nodes
  - When wildcards are used, variable bindings are also passed down

Pattern Nodes

- Pattern nodes keep record of matching facts for incremental updating
- Find *all* matches instead of *any* match
  - …and all variable-bindings
- E.g.
  - ?person1 could be Whisker or Captain
  - Not at the same time, but we pass both since we don’t know which is useful

Join Nodes

- Make sure that both inputs have matched and any variables agree
- When variable-bindings are used, join nodes identify all acceptable combinations of bindings
- Not necessarily AND
  - AND and XOR need extra support for unification

Rule Nodes

- All rules that receive input at bottom of network are triggered
- Arbiter determines which triggered rule goes on to fire

Updating the Network

- Could re-run each time with new database
  - But usually, data changes minimally between iterations
- Nodes store data, so only need to process changes to database.
  - Only update nodes that need it! Need remove/add.
  - Effects are handled by walking down the network
- Removing facts from database:
  - Request sent to pattern nodes
  - If node has stored match, remove it and pass request down.
  - Adding is basically the same.

Large Rule Sets

- Series of 2D turn-based war games
  - Large rule set
  - Each game in series required addition of many new rules: new features, player requests, bug fixes
  - Eventually, even RETE barfs
- Solution?
  - Group rules, and make activation hierarchy
  - Only rules in active sets are triggered
  - Disabled rules have no chance to trigger
- See “agenda groups” in Drools

Justification in Expert Systems

- Common extension is audit trail
- Capture rule firing information
  - The rule that fired
  - The data that the rule matched
  - Time stamp
- This information can be recursive
- Useful for debugging and justifying behavior

Rete Efficiency

- O(*nmp*) time efficiency
  - n = # rules
  - m = # clauses per rule
  - p = # facts in database
- Unifying wildcards can take over if wildcard matches are large
- More memory usage → faster performance

Resources

- Jess
  - http://www.jessrules.com/
- Drools (/OptaPlanner)
  - http://www.jboss.org/drools/
  - http://www.optaplanner.org/
  - http://www.javacodegeeks.com/2013/04/life-beyond-rete-r-i-p-rete-2013.html
- Aima-java, under FOL (see Unifier.java)
  - https://code.google.com/p/aima-java/

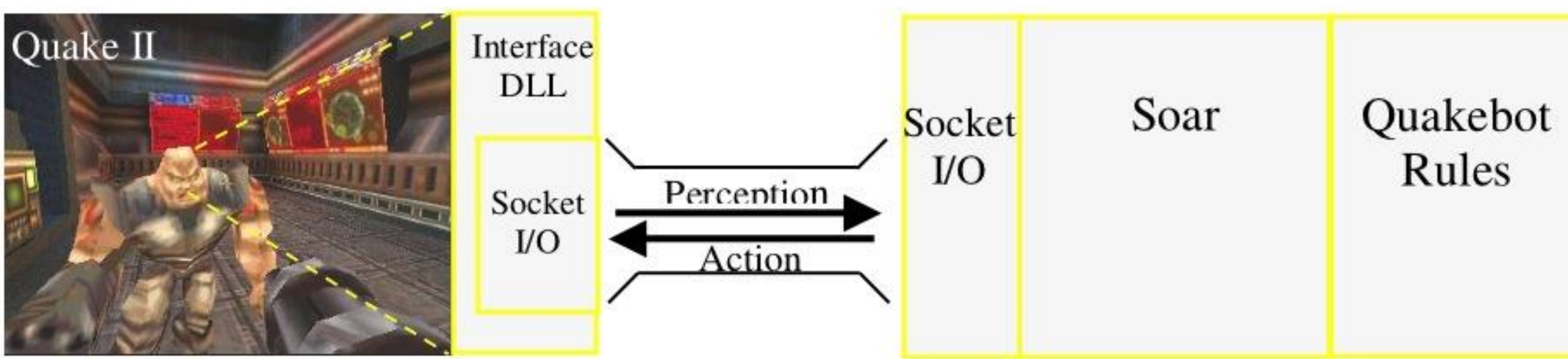
Jess

```
(defrule change-backup
        (< (health ?person1) 15)
        (> (health ?person2) 45)
        ?cover <- (is-covering (?person2 ?person1))
        ==>
        (//make a call to java//)
        (retract ?cover)
        (add (is-covering (?person1 ?person2)))
)
```

Soar

- A production system based on a theory of human cognition
- Production system with fancy arbitration
  - If two rules are active, Soar breaks the tie by firing more rules to figure out which is better
  - Forward mental simulation

```
sp {hello-world
     (state <s> ^type state)
     -->
     (write |Hello World|)
     (halt)}
```



Soar

- Newell, Laird, & Rosenbloom (CMU)
- Represents Newell's *Unified Theory of Cognition*
- Several decades in development
- Used in academic and military applications
- Previous Cognitive Psychology use
- Largest system: 8,000 rules