Disclaimer: I use these notes as a guide rather than a comprehensive coverage of the topic. They are neither a substitute for attending the lectures nor for reading the assigned material.

- "Everything should be as simple as possible, but not simpler." – Einstein
- Occam (of Razor fame – parsimony, economy, succinctness in logic/problem-solving)
  - "Entities should not be multiplied more than necessary"
  - "Of two competing theories or explanations, all other things being equal, the simpler one is to be preferred."
- "All that is complex is not useful. All that is useful is simple." – Mikhail Kalashnikov (of AK-47 fame)

# N-2&3: Decision Making, FSMs

1. How can we describe decision making?
2. What makes FSMs so attractive? What is difficult to do with them?
3. Two drawbacks of FSMs and how to fix?
4. What are the performance dimensions we tend to assess?
5. What are two methods we discussed to learn about changes in the world state?
6. FSMs/Btrees: R___ :: Planning : D____
7. When is the R__ good? When is D__?
8. H_____ have helped in most approaches.
9. What are two methods we discussed to learn about changes in the world state?

# N-1: Decision Making, D/B-trees

1. How many outcomes does a D/B-tree produce?
2. What are advantages of D-Trees?
3. Discuss the effects of tree balance.
4. Must D-trees be a tree?
5. Can D-trees translate into rules? If so how?
6. How can we use d-trees for prediction?
7. What is the notion of overfitting?

Decision trees can represent any Boolean function of the input attributes

More on learning Dtrees:
https://courses.cs.washington.edu/courses/cse573/12sp/lectures/19-dtree.pdf
https://www.cc.gatech.edu/~bboots3/CS4641-Fall2016/Lectures/Lecture2.pdf

# IMPORTANT NOTE

- Any desired set of behaviors that can be represented in a Behavior Tree can be represented in an FSM and vice versa.

- Differences:
  - **Sequences of Actions**: FSMs require extra variable tracking to handle sequences of states
  - **Error Recovery**: FSMs require many more linkages to do what Btrees do naturally

# Btree, Dtree, FSMs: Caveat Emptor

- In principle/theory… **they're equivalent**
  - There is nothing a behavior tree can do that a FSM cannot do
  - There is nothing a FSM can do that a decision tree cannot do
- While FSMs CAN represent sequences, it's awkward. Authoring/Design differences
  - BTrees make it easier to **design plan-like sequences** of actions
  - BTree **hierarchical decomposition** makes it easier to design behavior **modularly**
- Hybrid techniques possible: eg Dtree to evaluate FSM transitions
- Dtrees, Btrees and FSMs (and rules): reactive decision making
  - rely on **the ability of the designer** to create a good structure
  - can NOT respond to novel situations (**designer must anticipate all**)
  - have about same computation time (**quick responses**)
  - create enemies with **reliable patterns** of behavior (**less true for rules**)

# REACTIVE DECISION MAKING ALTERNATIVES

# Reactive Decision Making

- Real-time decision making by performing one action every instant
- Examples
  - State-action table
  - Universal plan
  - Behavior trees
  - Rule systems

# Reactive Planning

- Behavior trees implement a simple form of reactive planning
  - Real-time decision making by performing one action every instant

# Reactive Planning

- Where a state-action table gives us:

  $s_1$ --> $a_1$

  $s_2$ --> $a_2$

  …

  we get this from reactive plans:

  $s_1$ --> $a_{11}$ $a_{12}$ $a_{13}$…

  $s_2$ --> $a_{21}$ $a_{22}$…

  …

# Reactive Planning

- Advantages
  - Try things, fail, and fall back
  - Appearance of goal-driven behavior without a formal definition of goals
  - Fast

# Reactive Planning

- Advantages
  - Try things, fail, and fall back
  - Appearance of goal-driven behavior without a formal definition of goals
  - Fast
- Disadvantages
  - Can’t really think ahead
  - Only as forward-thinking as the designer makes it

# Decision Making: Rule-Based Systems

2018-02-27

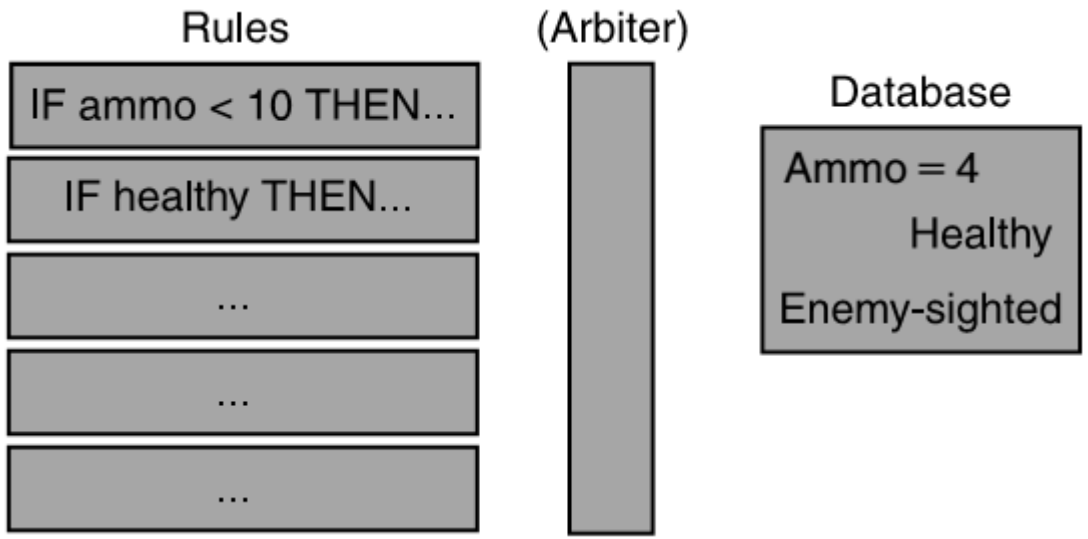
See M&F 5.8

# 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 – improvisation possible
- 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 based on DB/effects are more general
  - Rule-based system may also include arbiter
- Basic process / 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)

DATABASE:
1500 kg fuel remaining
100 km from base
enemies sighted: Enemy 42, Enemy 21
currently patroling

(DB rewrite) RULE:
IF number of sighted enemies > 0
and currently patroling
THEN
remove(currently patroling)
add(attack first sighted enemy)

# 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 [21, 46, 92])
  )

  (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

- The If-clause contains items of data to match joined by any Boolean operators

- 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
- On complexity:
  - N is number of items in DB,
  - M is number of clauses in a 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
```

```
class Rule:
    ifClause
    def action(bindings)

class Match:
    def matches( DB, bindings)
```

# 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

# Warning re IP

“You should also be careful of proprietary algorithms because many are patented. Just because an algorithm is published doesn’t mean it isn’t patented. You could end up having to pay a license fee for your implementation, even if you wrote the source code from scratch. We’re not lawyers, so we’d advise you to see an intellectual property attorney if you have any doubts.” – M&F

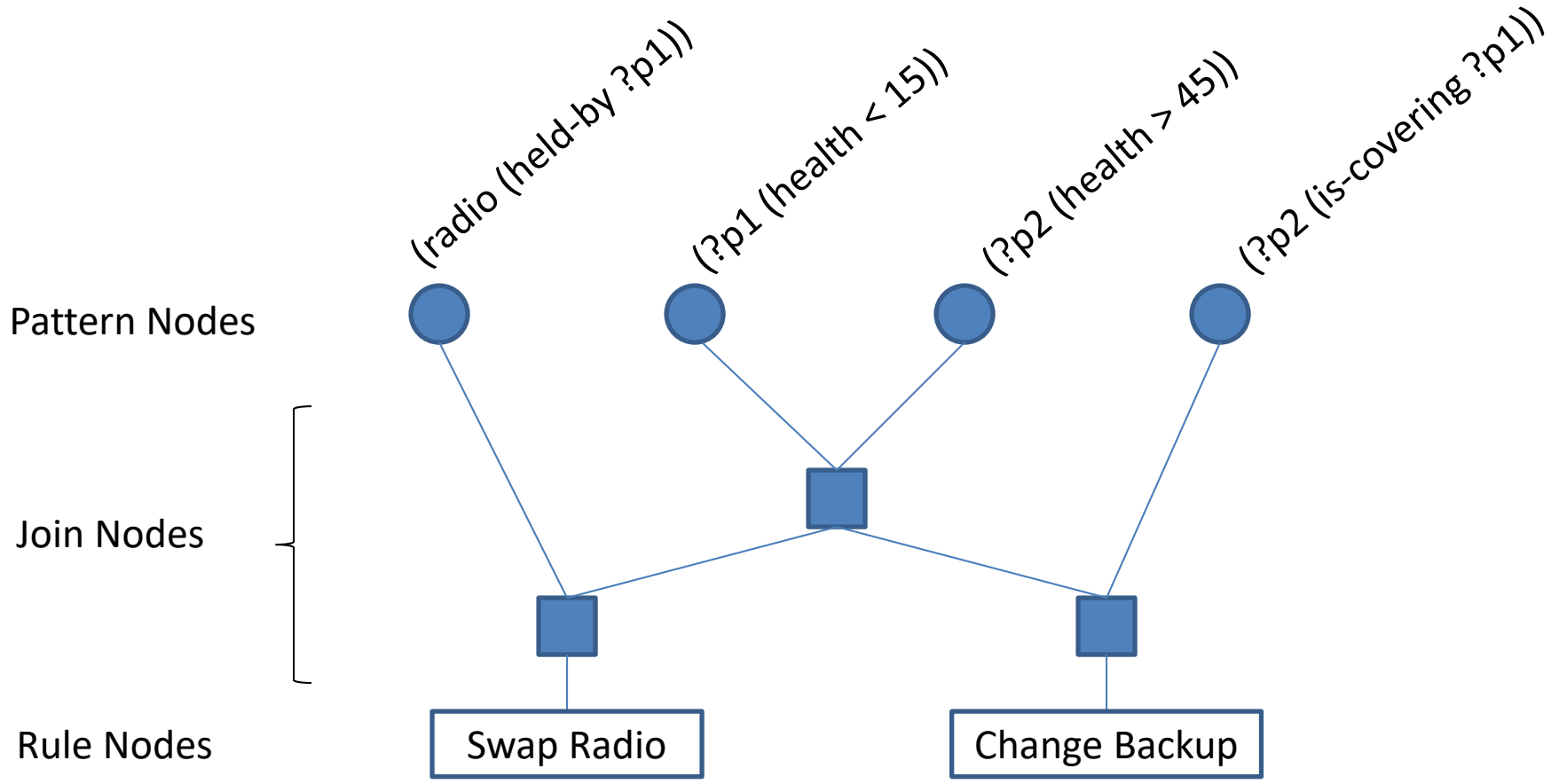
# Rete Example

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)
```

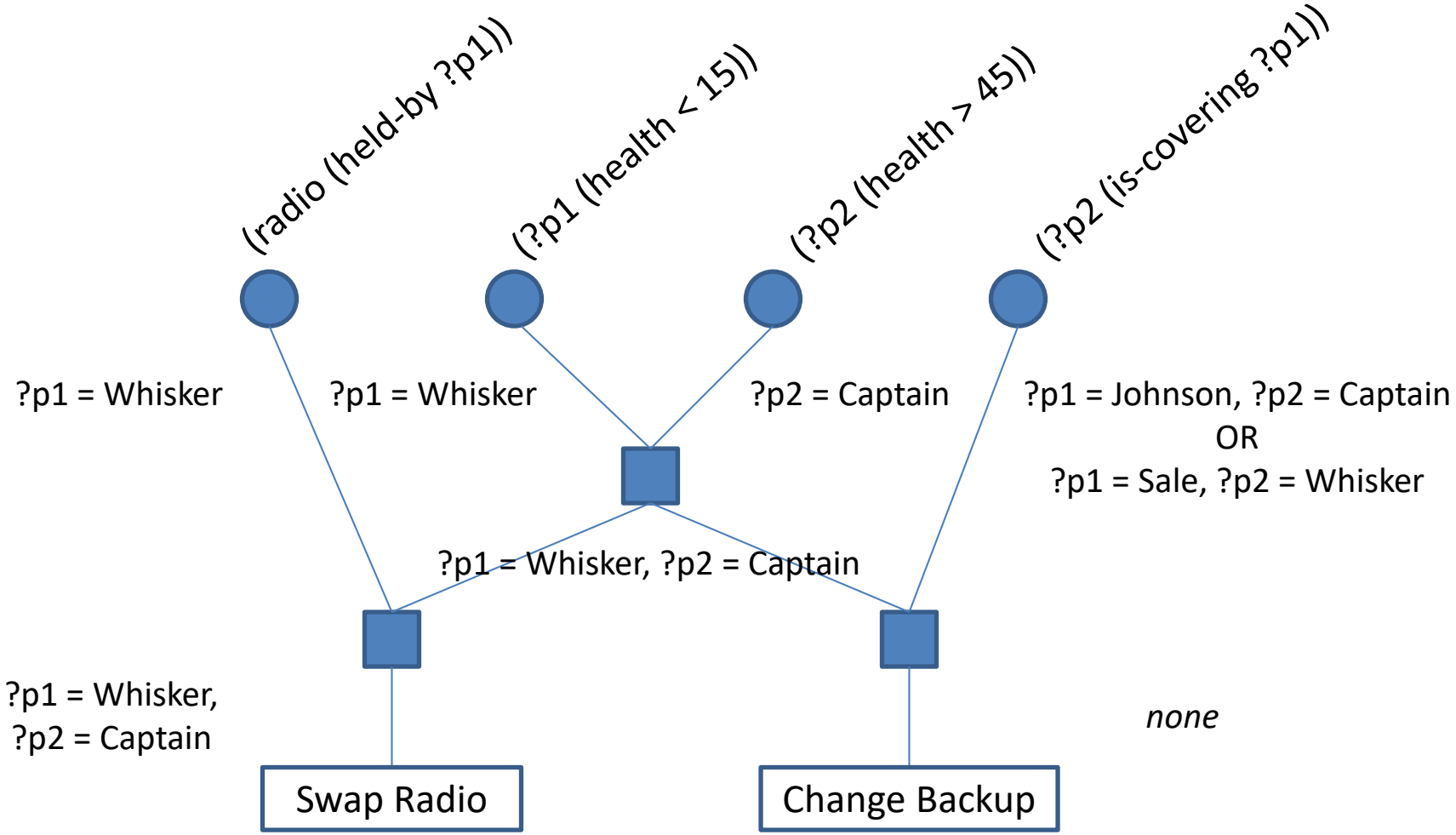
Change Backup Rule:

```
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

M&F Fig 5.48

(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

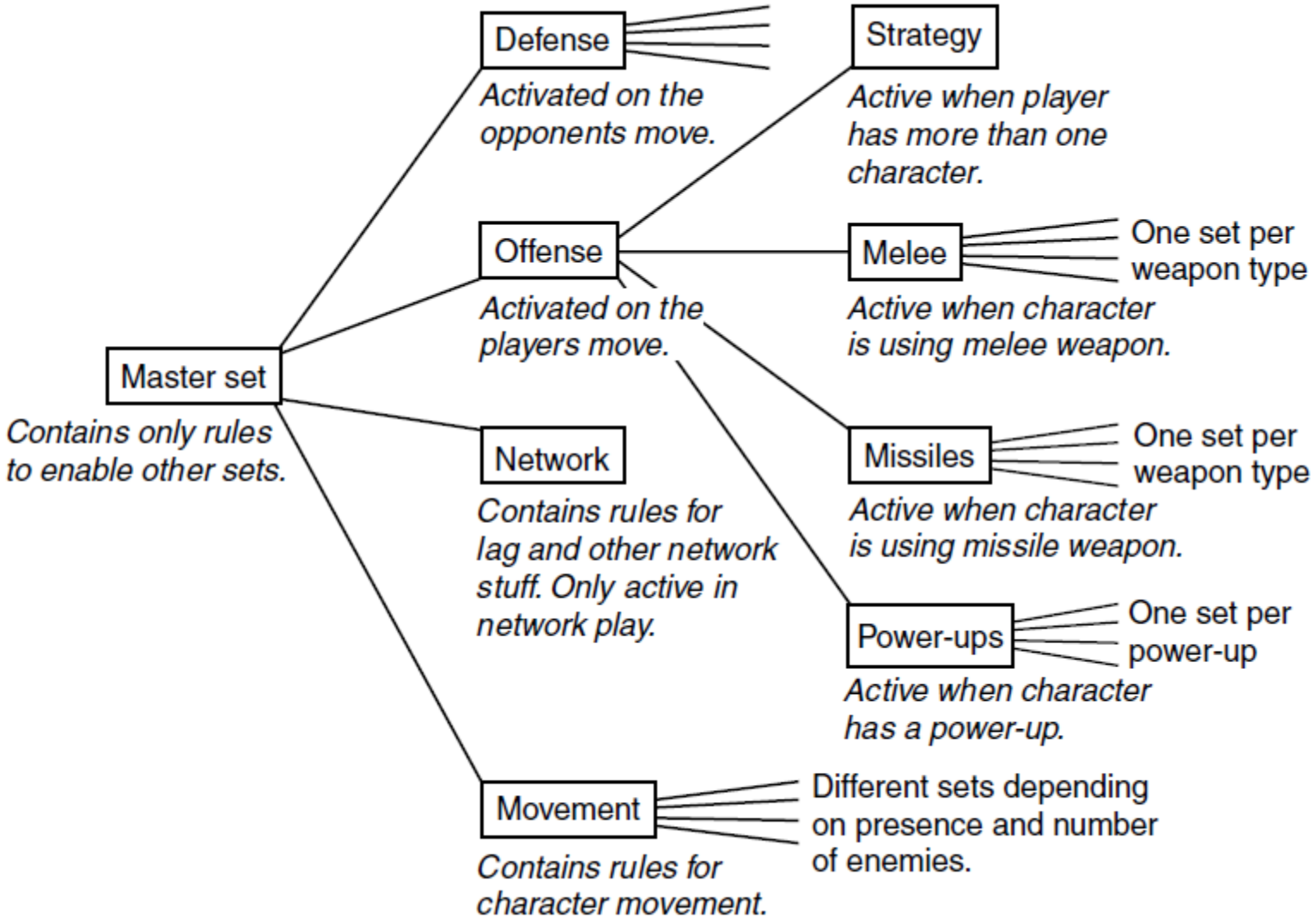
- All rules that receive input at bottom of network are triggered
- Arbiter determines which triggered rule goes on to fire

# 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


Master set
Contains only rules to enable other sets.
Defense
Activated on the opponents move.
Offense
Activated on the players move.
Network
Contains rules for lag and other network stuff. Only active in network play.
Movement
Contains rules for character movement.
Different sets depending on presence and number of enemies.
Strategy
Active when player has more than one character.
Melee
One set per weapon type
Active when character is using melee weapon.
Missiles
One set per weapon type
Active when character is using missile weapon.
Power-ups
One set per power-up
Active when character has a power-up.

M&F 5.53

# 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

# Advantages & Disadvantages

- Pros
  - First technique we've seen that allows for improvisation
  - Another way to achieve reactive behaviors
- Cons
  - Less designer control – possible to have unanticipated reactions
  - Can be difficult to debug
  - Difficult to make sequences of behaviors to achieve a goal
  - Reputation for inefficiency + challenge to impl.
  - Similar behaviors achievable using Dtree/FSMs

# Resources

- A full source code implementation is provided on the M&F website, with lots of comments
- 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/

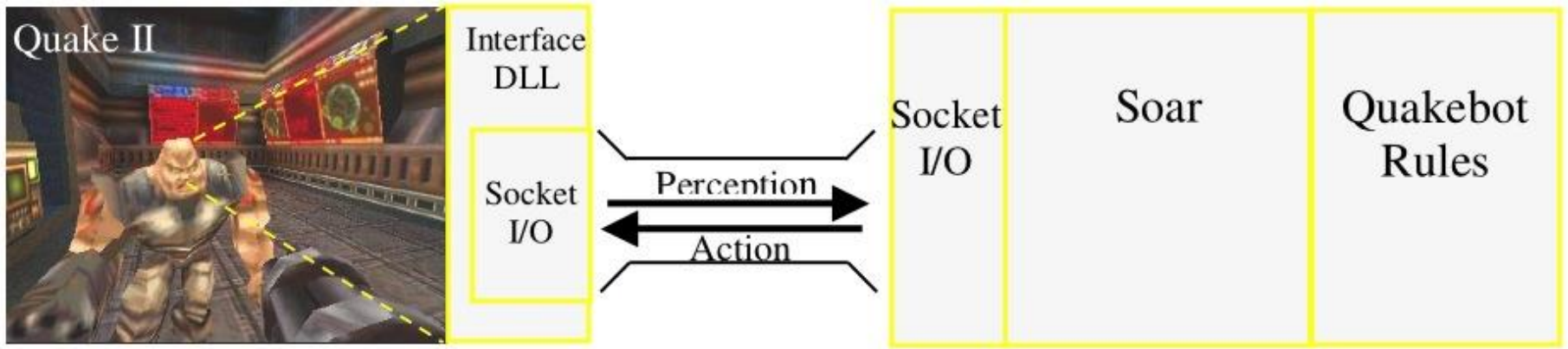
# Jess

```
(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