Knowledge Representation & Reasoning

Shyamanta M Hazarika
CSE, SoE
Tezpur University

Conflict resolution

Sometimes with data-directed reasoning, we want to fire *all* applicable rules

With goal-directed reasoning, we may want a *single* rule to fire

- arbitrary
- first rule in order of presentation (as in Prolog)
- specificity, as in

  IF (bird) **THEN ADD** (can-fly)

  IF (bird weight {> 100}) **THEN ADD** (cannot-fly)

  IF (bird) (penguin) **THEN ADD** (cannot-fly)
- recency
  - fire on rule that uses most recent WME
  - fire on least recently used rule
- refractoriness
  - never use same rule for same value of variables (called rule instance)
  - only use a rule/WME pair once (will need a "refresh" otherwise)

Conflict combinations

OPS5:

1. discard rule instances that have already been used
2. order remaining instances in terms of recency of WME matching 1st condition (and then of 2nd condition, etc.)
3. if still no single rule, order rules by number of conditions
4. select arbitrarily among those remaining

SOAR:

system that attempts to find a way to move from a start state to a goal state by applying productions

selecting what rule to fire

≡

deciding what to do next

if unable to decide, SOAR sets up the selection as a new (meta-)goal to solve, and the process iterates

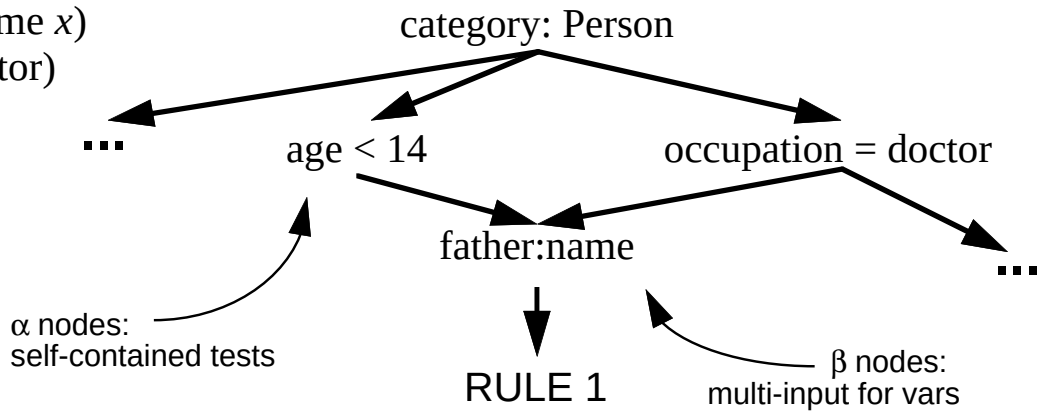
Rete procedure

Early systems spent 90% of their time matching, even with indexing and hashing.

But:
- WM is modified only slightly on each cycle
- many rules share conditions

So:
- incrementally pass WME through network of tests
- tokens that make it through satisfy all conditions and produce conflict set
- can calculate new conflict set in terms of old one and change to WM

IF (Person father *y* age {< 14} name *x*)
(Person name *y* occupation doctor)
THEN ...



Organizing procedures

With the move to put control of inference into the user's hands, we're focusing on more procedural representations

knowing facts by executing code

Even production systems are essentially programming languages.

Note also that everything so far is *flat*, i.e., sentence-like representations

- information about an object is scattered in axioms
- procedure fragments and rules have a similar problem

With enough procedures / sentences in a KB, it could be critical to *organize* them

- production systems might have rule sets, organized by context of application
- but this is not a natural, *representational* motivation for grouping

Object-centered representation

Most obvious organizational technique depends on our ability to see the world in terms of objects

- physical objects:
  - a desk has a surface-material, # of drawers, width, length, height, color, procedure for unlocking, etc.
  - some variations: no drawers, multi-level surface, built-in walls (carrel)
- also, *situations* can be object-like:
  - a class: room, participants, teacher, day, time, seating arrangement, lighting, procedures for registering, grading, etc.
  - leg of a trip: destination, origin, conveyance, procedures for buying ticket, getting through customs, reserving hotel room, locating a car rental etc.

Suggests clustering procedures for determining properties, identifying parts, interacting with parts, as well as constraints between parts, all of *objects*

- legs of desk connect to and support surface
- beginning of a travel leg and destination of prior one

object-centered constraints

Situation recognition

Focus on objects as an organizational / chunking mechanism to make some things easier to find

Suggests a different kind of reasoning than that covered so far

basic idea originally proposed by Marvin Minsky

- recognize (guess) situation; activate relevant object representations
- use those object representations to set up expectations

  some for verification; some make it easier to interpret new details
- flesh out situation once you've recognized

Wide applicability, but typical applications include

- relationship recognition e.g., story understanding
- data monitoring
- propagation and enforcement of constraints for planning tasks

  this latter is most doable and understandable,
  so we will concentrate on it

Basic frame language

Let's call our object structures frames

note wide variety of interpretations in literature

Two types:

- individual frames

  represent a single object like a person, part of a trip

- generic frames

  represent categories of objects, like students

An individual frame is a named list of buckets called slots. What goes in the bucket is called a filler of the slot. It looks like this:

(*frame-name*
  <*slot-name1* *filler1*>
  <*slot-name2* *filler2* > …)

where frame names and slot names are atomic, and fillers are either numbers, strings or the names of other individual frames.

Notation: individual frames: toronto
slot names: :Population (note ":" at start)
generic frames: CanadianCity

Instances and specializations

Individual frames have a special slot called :INSTANCE-OF whose filler is the name of a generic frame:

(toronto
<:**INSTANCE-OF** CanadianCity>
<:Province ontario>
<:Population 4.5M>…)

(tripLeg123-1
<:**INSTANCE-OF** TripLeg>
<:Destination toronto>…)

Generic frames have a syntax that is similar to that of individual frames, except that they have a slot called :IS-A whose filler is the name of another generic frame

(CanadianCity
<:**IS-A** City>
<:Province CanadianProvince>
<:Country canada>…)

We say that the frame toronto is an instance of the frame CanadianCity and that the frame CanadianCity is a specialization of the frame City

Procedures and defaults

Slots in generic frames can have associated procedures

1. computing a filler (when no slot filler is given)

   (Table
   <:Clearance [**IF-NEEDED** `computeClearanceFromLegs`]> …)

2. propagating constraints (when a slot filler is given)

   (Lecture
   <:DayOfWeek WeekDay>
   <:Date [**IF-ADDED** `computeDayOfWeek`]> …)

If we create an instance of Table, the :Clearance will be calculated as needed. Similarly, the filler for :DayOfWeek will be calculated when :Date is filled.

For instances of CanadianCity, the :Country slot will be filled automatically. But we can also have

(city135
<:**INSTANCE-OF** CanadianCity>
<:Country holland>)

The filler canada in CanadianCity is considered a default value.

IS-A and inheritance

Specialization relationships imply that procedures and fillers of more general frame are applicable to more specific frame: inheritance.

For example, instances of MahoganyCoffeeTable will inherit the procedure from Table (via CoffeeTable)

(CoffeeTable
<:**IS-A** Table> ...)

(MahoganyCoffeeTable
<:**IS-A** CoffeeTable> ...)

Similarly, default values are inheritable, so that Clyde inherits a colour from RoyalElephant, not Elephant

(Elephant
<:**IS-A** Mammal>
<:Colour gray> ...)

(RoyalElephant
<:**IS-A** Elephant>
<:Colour white>)

(clyde
<:**INSTANCE-OF** RoyalElephant>)