Knowledge Representation Techniques

Knowledge Engineering Course

Andrea Bonarini
Department of Electronics - Politecnico di Milano
http://www.dei.polimi.it/people/bonarini

Academic Year 2010-2011

Declarations or procedures...

- Declarative description
  Set of clauses describing properties.
  E.g.: A circle of radius *r* is the place of the points of a plane having distance *r* from a point called center
- Procedural description
  Set of procedures to describe how to use knowledge or how to increment knowledge
  E.g., To obtain a circle of radius *r* you can take a glass of radius *r* and go around it with a pencil

What is better?

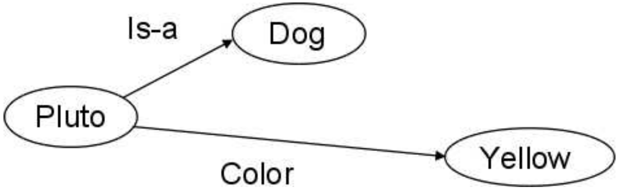
In the 70's and 80's there have been discussions about this topic

- A declaration can be used in principle in different ways, but needs something else to be used (e.g. an inferential mechanism, or programs that operate on it)
- Procedural knowledge cannot usually be re-used, nor it is usually possible to reason on it, but it is ready to be used for what it has been designed

We will see some knowledge representation techniques belonging to the two categories and eventual hybridizations

Semantic Networks

Nodes represent atomic concepts
Links are named and represent relationships



Problems with Semantic Networks

- No representation of semantics of links
- No classification of nodes
- No procedural description

Frames

A way to frame knowledge in terms of properties related to entities.

$< frame > ::= < frame\_name > (< is\_a >< slot >^*)$
$< is\_a > ::= \ (is\_aframe\_name >^*)$
$< slot > ::= \ (< slot\_name >< slot\_value >)$
$< slot\_value > ::= < frame\_name > | < string > | < number >$

E.g.,

$(\#dog \ \ (is\_a \ \ \#mammal)$
$\ (number\_of\_legs \ \ 4)$
$\ (eats \ \ “meat''))$

Characteristics and problems with frames

What can we represent with frames?

- Properties of entities
- A generic is_a hierarchy

Problems with frames

- No procedural description
- The is_a hierarchy is used both for specialization and instances

How to represent frames?

A definition CLIPS-like:

$(defframe\ < frame\_name >$
$\quad (is\_a < frame\_name >^{*})$
$\quad (< slot\_name >< slot\_value >)^{*}$
$)$

E.g.,

$(defframe\ \#Pippo\ (is\_a\ \#dog\ \#cartoon)$
$(number\_of\_legs\ \ 4)$
$(color\ \text{“}yellow\text{”})$
$(owned\_by\ \#Mickey)$
$(eats\ \text{"}meat\text{“})$
$)$

Test #1: frames

Let's represent with frames the knowledge contained in this text

- An engineer is a person
- Aristides is an engineer
- Engineers have a master degree
- Aristides got its degree on July, 25 2009
- Aristides likes dancing
- Engineers like computers and do not like dancing

Test #2: frames

Let's represent with frames the knowledge contained in this text.

- Pump P342 output is linked to pipe C32.
- Pipe C32 is linked to tank S321.
- Tank S321 is empty.
- The flow of pipe C32 is 2 m^3/h.
- The output flow from pump P342 is 1 m^3/h
- Pump P342 is not working properly

Demons

Procedures associated to data.

Data-driven programming

Three types of demons:

- **IfNeeded** is triggered when the datum is read
- **IfAdded** is triggered when the datum is changed
- **Notifier** is triggered if the datum takes a value

Demon definition

For instance, as procedures associated to frames:

$(\mathit{defframe} \ < \mathit{frame\_name} >$
$\quad (\mathit{is\_a} < \mathit{frame\_name} >^*)$
$\quad (< \mathit{slot\_name} >< \mathit{slot\_value} >$
$\quad ((< \mathit{demon} - \mathit{type} >< \mathit{demon} - \mathit{ref} >< \mathit{arguments} >)^*))^*$
$)$

E.g.,

$(\mathit{defframe} \ \#\mathit{login\_env} \ (\mathit{is\_a} \ \#\mathit{security\_env})$
$\ (\mathit{owner} \ \#\mathit{user})$
$\ (\mathit{last\_login} \ \#\mathit{date}$
$\quad ((\mathit{IfAdded} \ \#\mathit{InformSysManager} \ (\mathit{last\_login} \ \mathit{owner}))))$
$)$

Characteristics and problems of demons

What can we represent with demons?

- Procedures associated with data

Problems with demons

- They need data to be attached to
- It is difficult to control the execution flow

What is an Object?

Data structure able to exchange messages with others and to produce an action to answer a message

- Classes and instances
- Generalization and instance-class inheritance
- Procedures as methods to answer messages
- Typization of properties

Class definition

$$
\begin{array}{l}
(defclass \;\; < class\_name > \\
\quad (is\_a < nome\_frame >^*) \\
\quad (slot < slot\_name >< facet >^*)^* \\
)
\end{array}
$$

E.g.,

$$
\begin{array}{l}
(defclass \;\; \#CAR \\
\quad (is\_a \;\; \#USER) \\
\quad (slot \;\; max\_speed \;\; (access \;\; read - only) \;\; (default \;\; 200)) \\
\quad (slot \;\; (speed \;\; (default \;\; 0)) \\
)
\end{array}
$$

Instance definition

$$
\begin{array}{l}
(make - instance \quad < instance\_name > \quad of \quad < class\_name > \\
\quad (< slot\_name > \ < slot\_value >)^* \\
)
\end{array}
$$

E.g.,

$$
\begin{array}{l}
(make - instance \ \ MyCar \ \ of \ \ CAR \\
\quad (speed \ \ 100) \\
)
\end{array}
$$

Method definition (I)

$$
\begin{aligned}
&(defmessage - handler \;\; < class\_name > \; < method\_name > \\
&\qquad\qquad\qquad ((< var\_name > \; < var\_type >)^{*}) \\
&\quad < body >)
\end{aligned}
$$

E.g.,

$$
\begin{aligned}
&(defmessage - handler \;\; CAR \;\; accellerate \\
&\quad ((?percentual - speed \;\; ZERO - ONE)) \\
&\quad (bind \;\; ?self : speed \;\; (+ \;\; ?self : speed \\
&\qquad\quad (* \;\; ?self : speed \;\; ?percentual - speed))) \\
&)
\end{aligned}
$$

Method example (I)

```
CLIPS > (make-instance my-car of CAR
                  (speed 100))
[mycar]

CLIPS > (send [mycar] accellerate 0.1)
110

CLIPS > (send [mycar] print)
[mycar] of CAR
        (max-speed 200)
        (speed 110)
```

Method definition (II)

$(defmethod \ \ < method\_name > \ \ ((< var\_name > \ < var\_type >)^{*})$
$\quad < body >)$

E.g.,

$(defmethod \ \ sum \ \ ((?a \ \ INTEGER)\,(?b \ \ INTEGER))$
$\quad (+ \ \ ?a \ \ ?b))$

$(defmethod \ \ sum \ \ ((?a \ \ STRING)\,(?b \ \ STRING))$
$\quad (str - cat \ \ ?a \ \ ?b))$

$CLIPS >$ (sum 2 3)
5

$CLIPS >$ (sum "Man" "drake")
Mandrake

Assignment and reading of slot values

Methods defined by default at class definition to provide access to slots.

$(send \;\; < instance\_name > \;\; put- < slot\_name > \; < new\_value >)$

$(send \;\; < instance\_name > \;\; get- < slot\_name >)$

```
CLIPS > (send [mycar] put-speed 180)
180

CLIPS > (send [mycar] get-speed)
180
```

Test #3: objects

Let's represent with objects the knowledge contained in this text.

- Pump P342 output is linked to pipe C32.
- Pipe C32 is linked to tank S321.
- Tank S321 is empty.
- The flow of pipe C32 is 2 m^3/h.
- The output flow from pump P342 is 1 m^3/h
- Pump P342 is not working properly
- When a tank is empty an alarm is risen
- Indicator PIRCA321 provides the pressure in tank S231 on demand
- Indicator LEV321 provides level in tank S321 at any time

Logic

Logic is the formal systematic study of the principles of valid inference and correct reasoning.
It has been established as a discipline by Aristotle in the western world and used since thousands of years to represent formally knowledge and its use.

Logic is often divided into two parts, inductive reasoning and deductive reasoning. The first draws general conclusions from specific examples, the second draws logical conclusions from definitions and axioms.

Among the main elements we have in logic: axioms (what it is assumed to be true), theorems (to define relationships), operators to combine truth values, quantifiers ("it exists...“, "for all...“) to operate on variables.

Potential problems with logic

- The procedural aspects can only be represented by inferential mechanisms
- Once triggered, an inferential mechanism will generate all all it can draw, which is the same as making explicit all the knowledge coded in a compact way in theorems, and may lead to memory problems
- First order logic is often not enough, and more powerful logics are needed

Rules

Rules are used to represent inferential mechanisms.
Their general form is:

$$
\begin{aligned}
< rule > ::= \ & If(< composite\_clause >) \\
& Then(< composite\_clause >) \\
< clause > ::= & < fact > \ | \ < pattern > \\
< composite\_clause > ::= & < clause > \ | NOT \ < composite\_clause > \\
& | < composite\_clause > \ AND \ < composite\_clause > \\
& | < composite\_clause > \ OR \ < composite\_clause > \\
< fact > ::= & < string > \\
< pattern > ::= & "A \ sequence \ of \ variables \ and \ strings" \\
< variable > ::= & ?< string >
\end{aligned}
$$

Forward and backward rules

Rules can be used to represent either induction or deduction.

In the two cases they are used in different ways: **forward chaining** to deduce consequences, or **backward chaining** to induce possible causes.

E.g., in the case of a car:

- IF ("headlight circuit open")
  THEN ("headlights cannot produce any light")
- IF ("headlight fuse burnt")
  THEN ("headlight circuit open")
- IF ("short circuit on headlight circuit")
  THEN ("headlight fuse burnt")

We might use these rules either backward, to diagnose the fact that the headlights are not working, or forward, to deduce the consequences of a short circuit.

Pattern matching

A pattern is a sequence of strings and variables, such as those in this rule:

IF (?a ($Color #Red)) AND (?a ($InstanceOf #Apple))
THEN (?a ($ReadyP True))

Pattern-matching compares a pattern with a sequence of strings (e.g., a fact), assigning to the variables the corresponding values.

E.g., if we have the rule above, and the facts:
(#Apple1 ($Color #Red))
(#Apple1 ($InstanceOf #Apple))

we can match ?a with #Apple1 and keep this unification for this rule:
(?a #Apple1)

By triggering the rule, we might infer (#Apple1 (ReadyP True))

Conflict set

At any time we have facts describing the state of the world, listed in the so-called **fact base**.

We can match the set of rules (**rule base**) with the fact base, and obtain a list of rules that might be triggered to increment the fact base.

This list is called **conflict set**, since, in most systems, only one rule at a time can be triggered before repeating the pattern matching process on the updated fact base.

Conflict resolution

More than one rule can be found in the conflict set. Different policies can be adopted to resolve this conflict and select the rule to be triggered:

- the first rule entering in the conflict set (lexicographic order): the designer gives the priority of the rules by ordering them in the rule base
- the rule using the fact entered most recently, among the ones matching rules, in the fact base: this helps to follow a reasoning line
- the rule using more facts, to exploit at best the available knowledge

Using rules

Rules are used to represent heuristic knowledge about the use of knowledge.
Possible problems:

- the inferential mechanism is usually pre-defined for a rule base
- they do not represent naturally declarative knowledge, they use it

Test #5: KR Techniques

Let's represent with objects and rules the knowledge needed to produce the set of actions that CAT, the robot, has to take to go from one place to another.

- CAT can:
  - move forward and backward
  - turn left and right by 30°
  - perceive obstacles and doors (with different sensors) at a close distance forward, backward, on the left and on the right.
- CAT is in Andrew's office
- Andrew's office is North of the Corridor
- CAT should come to the Secretary' office
- the Secretary's office is on the South side of the corridor
- the Secretary's office is the second door left to the door in front of Andrew's office