Module 8

Other representation formalisms

Lesson 20

Frames - I

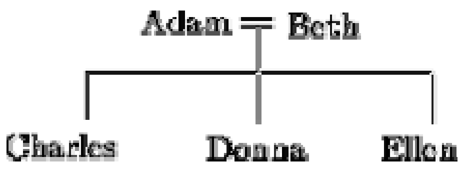
8.4 Frames

Frames are descriptions of conceptual individuals. Frames can exist for ``real" objects such as ``The Watergate Hotel", sets of objects such as ``Hotels", or more ``abstract" objects such as ``Cola-Wars" or ``Watergate".

A Frame system is a collection of objects. Each object contains a number of *slots*. A slot represents an attribute. Each slot has a value. The value of an attribute can be another object. Each object is like a C struct. The struct has a name and contains a bunch of named values (which can be pointers)

Frames are essentially defined by their relationships with other frames. Relationships between frames are represented using *slots*. If a frame `f` is in a relationship `r` to a frame `g`, then we put the *value* `g` in the `r` *slot* of `f`.

For example, suppose we are describing the following genealogical tree:



The frame describing Adam might look something like:

```
Adam:
   sex:     Male
   spouse:  Beth
   child:   (Charles Donna Ellen)
```

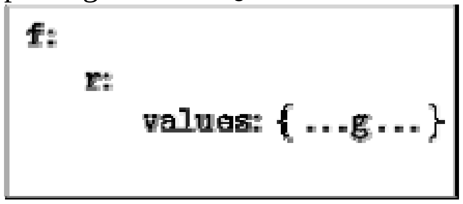
where `sex`, `spouse`, and `child` are slots. Note that a single slot may hold several values (*e.g.* the children of Adam).

The genealogical tree would then be described by (at least) seven frames, describing the following individuals: Adam, Beth, Charles, Donna, Ellen, Male, and Female.

A frame can be considered just a convenient way to represent a set of predicates applied to constant symbols (*e.g. ground instances* of predicates.). For example, the frame above could be written:

```
  sex(Adam,Male)
  spouse(Adam,Beth)
  child(Adam,Charles)
  child(Adam,Donna)
  child(Adam,Ellen)
```

More generally, the ground predicate *r*(*f*,*g*) is represented, in a frame based system, by placing the value `g` in the `r` slot of the frame `f` : *r*(*f*,*g*) ≡



```
f:
    r:
        values: { ...g...}
```

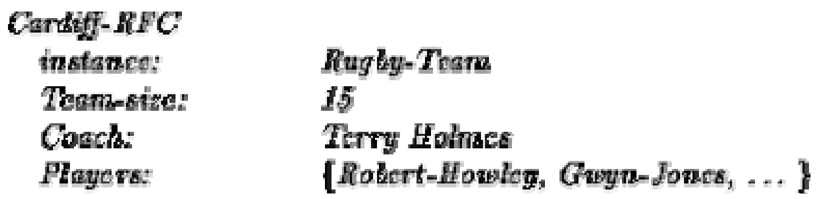
Frames can also be regarded as an extension to Semantic nets. Indeed it is not clear where the distinction between a semantic net and a frame ends. Semantic nets initially we used to represent labelled connections between objects. As tasks became more complex the representation needs to be more structured. The more structured the system it becomes more beneficial to use frames. A *frame* is a collection of attributes or slots and associated values that describe some real world entity. Frames on their own are not particularly helpful but frame systems are a powerful way of encoding information to support reasoning. Set theory provides a good basis for understanding frame systems. Each frame represents:

- a class (set), or
- an instance (an element of a class).

Consider the example below.

```
Person

                 isa:                    Mammal

                 Cardinality:            ...

Adult-Male

                isa:            Person

                 Cardinality:            ...

Rugby-Player

                 isa:            Adult-Male

                 Cardinality:            ...

                 Height:

                Weight:
```

```
                Position:

                Team:

               Team-Colours:

Back


                isa:           Rugby-Player

                Cardinality:            ...

                Tries:

Mike-Hall

                instance:               Back

                Height:                 6-0

                Position:               Centre

                Team:           Cardiff-RFC

                Team-Colours:           Black/Blue

Rugby-Team

                isa:            Team

                Cardinality:            ...

                Team-size:              15

                Coach:
```



```
Cardiff-RFC
   instance:          Rugby-Team
   Team-size:         15
   Coach:             Terry Holmes
   Players:           {Robert-Howley, Gwyn-Jones, ... }
```

Here the frames *Person, Adult-Male, Rugby-Player and Rugby-Team* are all **classes** and the frames *Robert-Howley* and *Cardiff-RFC* are instances.

Note

- The *isa* relation is in fact the subset relation.
- The *instance* relation is in fact *element of*.
- The *isa* attribute possesses a transitivity property. This implies: *Robert-Howley* is a *Back* and a *Back* is a *Rugby-Player* who in turn is an *Adult-Male* and also a *Person*.
- Both *isa* and *instance* have inverses which are called subclasses or all instances.
- There are attributes that are associated with the class or set such as cardinality and on the other hand there are attributes that are possessed by each member of the class or set.

DISTINCTION BETWEN SETS AND INSTANCES

It is important that this distinction is clearly understood.

Cardiff-RFC can be thought of as a set of players or as an instance of a *Rugby-Team*.

If *Cardiff-RFC* were a *class* then

- its instances would be players
- it could not be a subclass of *Rugby-Team* otherwise its elements would be members of Rugby-Team which we do not want.

Instead we make it a subclass of *Rugby-Player* and this allows the players to inherit the correct properties enabling us to let the *Cardiff-RFC* to inherit information about teams.

This means that *Cardiff-RFC* is an instance of *Rugby-Team*.

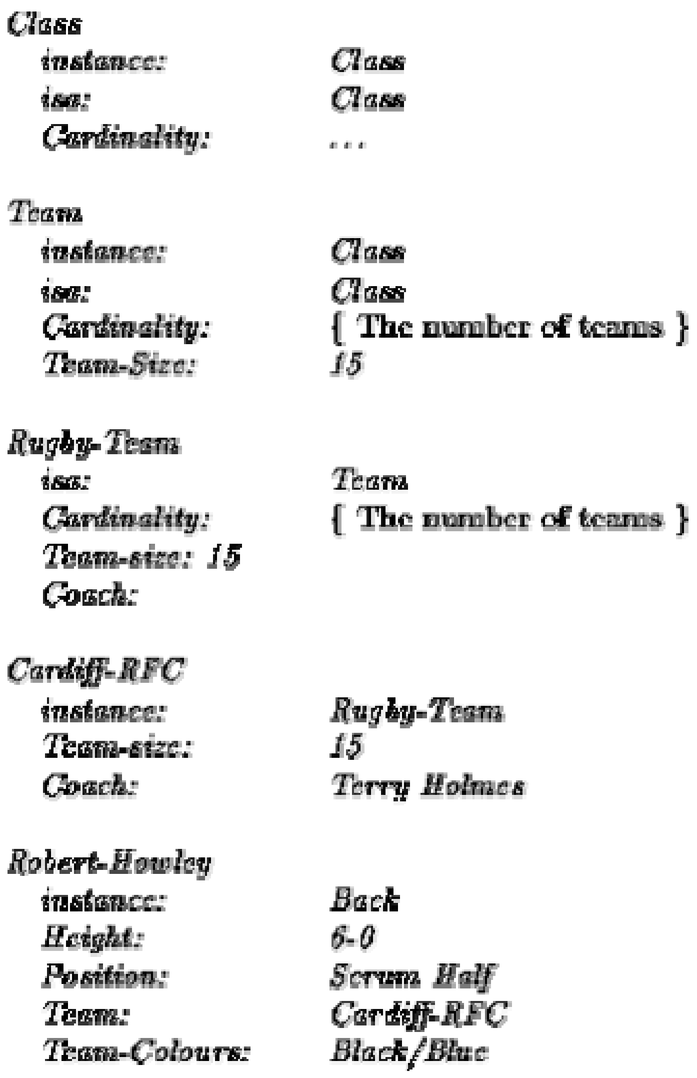
BUT There is a problem here:

- A class is a set and its elements have properties.
- We wish to use inheritance to bestow values on its members.
- But there are properties that the set or class itself has such as the manager of a team.

This is why we need to view *Cardiff-RFC* as a subset of one class players and an instance of teams. We seem to have a CATCH 22. *Solution*: **MetaClasses**

A metaclass is a special class whose elements are themselves classes.

Now consider our rugby teams as:



Class	
instance:	*Class*
isa:	*Class*
Cardinality:	. . .
Team	
instance:	*Class*
isa:	*Class*
Cardinality:	{ The number of teams }
Team-Size:	*15*
Rugby-Team	
isa:	*Team*
Cardinality:	{ The number of teams }
Team-size: 15	
Coach:	
Cardiff-RFC	
instance:	*Rugby-Team*
Team-size:	*15*
Coach:	*Terry Holmes*
Robert-Howley	
instance:	*Back*
Height:	*6-0*
Position:	*Scrum Half*
Team:	*Cardiff-RFC*
Team-Colours:	*Black/Blue*

The basic metaclass is *Class*, and this allows us to

- define classes which are instances of other classes, and (thus)
- inherit properties from this class.

Inheritance of default values occurs when one element or class is an instance of a class.