Next Up Previous
Next: What to Represent? **Up:** Artificial Intelligence II **Previous:** Exercises

Knowledge Representation

---

dave@cs.cf.ac.uk

Next Up Previous
Next: Using Knowledge **Up:** Knowledge Representation **Previous:** Knowledge Representation

What to Represent?

Let us first consider what kinds of knowledge might need to be represented in AI systems:

Objects
-- Facts about objects in our world domain. *e.g.* Guitars have strings, trumpets are brass instruments.

Events
-- Actions that occur in our world. *e.g.* Steve Vai played the guitar in Frank Zappa's Band.

Performance
-- A behavior like *playing the guitar* involves knowledge about how to do things.

Meta-knowledge
-- knowledge about what we know. *e.g.* Bobrow's Robot who plan's a trip. It knows that it can read street signs along the way to find out where it is.

Thus in solving problems in AI we must represent knowledge and there are two entities to deal with:

Facts
-- truths about the real world and what we represent. This can be regarded as the *knowledge level*

Representation of the facts
which we manipulate. This can be regarded as the *symbol level* since we usually define the representation in terms of symbols that can be manipulated by programs.

We can structure these entities at two levels

the knowledge level
-- at which facts are described

the symbol level
-- at which representations of objects are defined in terms of symbols that can be manipulated in programs (see Fig. 5)

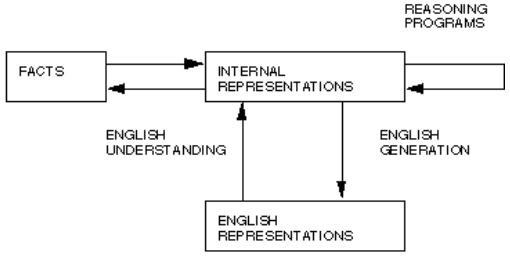


Fig 5 Two Entities in Knowledge Representation

English or natural language is an obvious way of representing and handling facts. Logic enables us to

consider the following fact: *spot is a dog* as *dog(spot)* We could then infer that all dogs have tails with: $\forall x : dog(x) \rightarrow hasatail(x)$ We can then deduce:

hasatail(Spot)

Using an appropriate backward mapping function the English sentence *Spot has a tail can be generated.*

The available functions are not always one to one but rather are many to many which is a characteristic of English representations. The sentences *All dogs have tails* and *every dog has a tail* both say that each dog has a tail but the first could say that each dog has more than one tail try substituting teeth for tails. When an AI program manipulates the internal representation of facts these new representations should also be interpretable as new representations of facts.

Consider the classic problem of the mutilated chess board. Problem In a normal chess board the opposite corner squares have been eliminated. The given task is to cover all the squares on the remaining board by dominoes so that each domino covers two squares. No overlapping of dominoes is allowed, can it be done. Consider three data structures

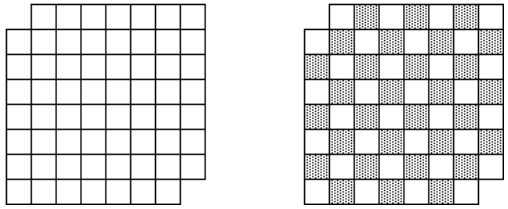


Fig. 3.1 Mutilated Checker

the first two are illustrated in the diagrams above and the third data structure is the number of black squares and the number of white squares. The first diagram loses the colour of the squares and a solution is not east to see; the second preserves the colours but produces no easier path whereas counting the number of squares of each colour giving black as 32 and the number of white as 30 yields an immediate solution of NO as a domino must be on one white square and one black square, thus the number of squares must be equal for a positive solution.

Next Up Previous

Next: Using Knowledge **Up:** Knowledge Representation **Previous:** Knowledge Representation

dave@cs.cf.ac.uk

Next Up Previous

Next: Properties for Knowledge Representation **Up:** Knowledge Representation **Previous:** What to Represent?

Using Knowledge

We have briefly mentioned where knowledge is used in AI systems. Let us consider a little further to what applications and how knowledge may be used.

Learning
-- acquiring knowledge. This is more than simply adding new facts to a knowledge base. New data may have to be *classified* prior to storage for easy *retrieval, etc.*. *Interaction* and *inference* with existing facts to avoid redundancy and replication in the knowledge and and also so that facts can be updated.

Retrieval
-- The representation scheme used can have a critical effect on the *efficiency* of the method. Humans are very good at it.

Many AI methods have tried to model human (see lecture on distributed reasoning)

Reasoning
-- Infer facts from existing data.

If a system on only knows:

- Miles Davis is a Jazz Musician.
- All Jazz Musicians can play their instruments well.

If things like *Is Miles Davis a Jazz Musician?* or *Can Jazz Musicians play their instruments well?* are asked then the answer is readily obtained from the data structures and procedures.

However a question like *Can Miles Davis play his instrument well?* requires reasoning.

The above are all related. For example, it is fairly obvious that learning and reasoning involve retrieval *etc*.

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Approaches to Knowledge Representation **Up:** Knowledge Representation **Previous:** Using Knowledge

Properties for Knowledge Representation Systems

The following properties should be possessed by a knowledge representation system.

Representational Adequacy
-- the ability to represent the required knowledge;
Inferential Adequacy
- the ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original;
Inferential Efficiency
- the ability to direct the inferential mechanisms into the most productive directions by storing appropriate guides;
Acquisitional Efficiency
- the ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

To date no single system optimises all of the above

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Simple relational knowledge **Up:** Knowledge Representation **Previous:** Properties for Knowledge Representation

Approaches to Knowledge Representation

We briefly survey some representation schemes. We will look at some in more detail in further lectures. Also some other course (*e.g. Expert Systems* also deal with related subjects).

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Inheritable knowledge **Up:** Approaches to Knowledge Representation **Previous:** Approaches to Knowledge Representation

Simple relational knowledge

The simplest way of storing facts is to use a relational method where each fact about a set of objects is set out systematically in columns. This representation gives little opportunity for inference, but it can be used as the knowledge basis for inference engines.

- Simple way to store facts.
- Each fact about a set of objects is set out systematically in columns (Fig. 7).
- Little opportunity for inference.
- Knowledge basis for inference engines.

Musician	Style	Instrument	Age
Miles Davis	Jazz	Trumpet	deceased
John Zorn	Avant Garde	Saxophone	35
Frank Zappa	Rock	Guitar	deceased
John Mclaughlin	Jazz	Guitar	47

Figure: Simple Relational Knowledge

We can ask things like:

- Who is dead?
- Who plays Jazz/Trumpet *etc.*?

This sort of representation is popular in database systems.

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Inferential Knowledge **Up:** Approaches to Knowledge Representation **Previous:** Simple relational knowledge

Inheritable knowledge

Relational knowledge is made up of objects consisting of

- attributes
- corresponding associated values.

We extend the base more by allowing inference mechanisms:

- Property inheritance
  - elements inherit values from being members of a class.
  - data must be organised into a hierarchy of classes (Fig. 8).

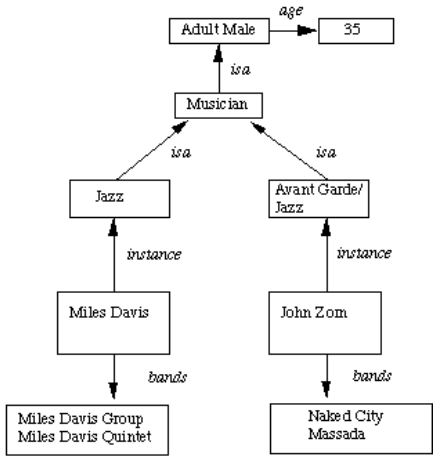


Fig. 8 Property Inheritance Hierarchy

- Boxed nodes -- objects and values of attributes of objects.
- Values can be objects with attributes and so on.
- Arrows -- point from object to its value.
- This structure is known as a slot and filler structure, semantic network or a collection of frames.

The algorithm to retrieve a value for an attribute of an instance object:

1. Find the object in the knowledge base
2. If there is a value for the attribute report it
3. Otherwise look for a value of instance if none fail
4. Otherwise go to that node and find a value for the attribute and then report it
5. Otherwise search through using *isa* until a value is found for the attribute.

---

dave@cs.cf.ac.uk

Next: Procedural Knowledge **Up:** Knowledge Representation **Previous:** Inheritable knowledge

Inferential Knowledge

Represent knowledge as *formal logic*:

All dogs have tails $\forall x$: *dog(x)* $\rightarrow$ *hasatail(x)* Advantages:

- A set of strict rules.
  - Can be used to derive more facts.
  - Truths of new statements can be verified.
  - Guaranteed correctness.
- Many inference procedures available to in implement standard rules of logic.
- Popular in AI systems. *e.g* Automated theorem proving.

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Issue in Knowledge Representation **Up:** Inferential Knowledge **Previous:** Inferential Knowledge

Procedural Knowledge

Basic idea:

- Knowledge encoded in some procedures
  - small programs that know how to do specific things, how to proceed.
  - *e.g* a parser in a natural language understander has the knowledge that a *noun phrase* may contain articles, adjectives and nouns. It is represented by calls to routines that know how to process articles, adjectives and nouns.

Advantages:

- *Heuristic* or domain specific knowledge can be represented.
- *Extended logical inferences*, such as default reasoning facilitated.
- *Side effects* of actions may be modelled. Some rules may become false in time. Keeping track of this in large systems may be tricky.

Disadvantages:

- Completeness -- not all cases may be represented.
- Consistency -- not all deductions may be correct.

  e.g If we know that *Fred is a bird* we might deduce that *Fred can fly*. Later we might discover that *Fred is an emu*.

- Modularity is sacrificed. Changes in knowledge base might have far-reaching effects.
- Cumbersome control information.

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Summary and the way **Up:** Knowledge Representation **Previous:** Procedural Knowledge

Issue in Knowledge Representation

Below are listed issues that should be raised when using a knowledge representation technique:

Important Attributes
-- Are there any attributes that occur in many different types of problem?

There are two *instance* and *isa* and each is important because each supports property inheritance.

Relationships
-- What about the relationship between the attributes of an object, such as, inverses, existence, techniques for reasoning about values and single valued attributes. We can consider an example of an inverse in

band(John Zorn,Naked City)

This can be treated as John Zorn plays in the band *Naked City* or John Zorn's band is *Naked City*.

Another representation is *band = Naked City*

band-members = John Zorn, Bill Frissell, Fred Frith, Joey Barron, . . .

Granularity
-- At what level should the knowledge be represented and what are the primitives. Choosing the Granularity of Representation Primitives are fundamental concepts such as holding, seeing, playing and as English is a very rich language with over half a million words it is clear we will find difficulty in deciding upon which words to choose as our primitives in a series of situations.

If *Tom feeds a dog* then it could become:

feeds(tom, dog)

If *Tom gives the dog a bone* like:

gives(tom, dog,bone) Are these the same?

In any sense does giving an object food constitute feeding?

If $give(x, food) \rightarrow feed(x)$ then we are making progress.

But we need to add certain inferential rules.

In the famous program on relationships *Louise is Bill's cousin* How do we represent this? *louise = daughter (brother or sister (father or mother(bill)))* Suppose it is *Chris* then we do not know if it is *Chris* as a male or female and then *son* applies as well.

Clearly the separate levels of understanding require different levels of primitives and these need many rules to link together apparently similar primitives.

Obviously there is a potential storage problem and the underlying question must be what level of comprehension is needed.

Next Up Previous

Next: Summary and the way **Up:** Knowledge Representation **Previous:** Procedural Knowledge

dave@cs.cf.ac.uk

Next Up Previous

Next: Further Reading **Up:** Knowledge Representation **Previous:** Issue in Knowledge Representation

Summary and the way forward

In this lecture we have hopefully seen the need for knowledge in reasoning programs.

Many issues need to be considered when deciding on the representation scheme for knowledge.

We introduced the concept of a *slot and filler* data structure for inheritable knowledge. In the next two lectures we will be looking at different types of *slot and filler* representations starting with semantic nets and frames and moving onto stronger conceptual dependency based representations.

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Logic Knowledge Representation **Up:** Knowledge Representation **Previous:** Summary and the way

Further Reading

Artificial Intelligence by Rich and Knight covers most topics well.

One book dedicated to this field is *Knowledge Representation: An AI perspective* by H. Reichgelt, Ablex, New York.

The Handbook of Artificial Intelligence (Vol. 1, Ch 3) provides a good concise introduction to most topics

Bobrow and Collins deal with fundamental issues and also their robot planner in *Representation and Understanding*, Academic Press (1975).

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Predicate logic **Up:** Artificial Intelligence II **Previous:** Further Reading

Logic Knowledge Representation

We briefly mentioned how logic can be used to represent simple facts in the last lecture. Here we will highlight major principles involved in knowledge representation. In particular *predicate logic* will be met in other knowledge representation schemes and reasoning methods.

A more comprehensive treatment is given in the third year *Expert Systems* course. **Symbols used** The following standard logic symbols we use in this course are:

`For all`	∀
`There exists`	∃
`Implies`	→
`Not`	¬
`Or`	∨
`And`	∧

Let us now look at an example of how predicate logic is used to represent knowledge. There are other ways but this form is popular.

---

dave@cs.cf.ac.uk

Next Up Previous

Next: An example **Up:** Logic Knowledge Representation **Previous:** Logic Knowledge Representation

Predicate logic

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Isa and instance relationships **Up:** Predicate logic **Previous:** Predicate logic

An example

Consider the following:

- Prince is a mega star.
- Mega stars are rich.
- Rich people have fast cars.
- Fast cars consume a lot of petrol.

and try to draw the conclusion: *Prince's car consumes a lot of petrol*.

So we can translate *Prince is a mega star* into: *mega_star(prince)* and *Mega stars are rich* into: $\forall m: mega\_star(m) \rightarrow rich(m)$

Rich people have fast cars, the third axiom is more difficult:

- Is *cars* a relation and therefore *car(c,m)* says that case *c* is *m's* car. **OR**
- Is *cars* a function? So we may have *car_of(m)*.

Assume *cars* is a relation then axiom 3 may be written: $\forall c,m: car(c,m) \wedge rich(m) \rightarrow fast(c)$.

The fourth axiom is a general statement about *fast cars*. Let *consume(c)* mean that car *c* consumes a lot of petrol. Then we may write: $\forall c:[fast(c) \wedge \exists m:car(c,m) \rightarrow consume(c)]$

Is this enough? NO! -- Does prince have a car? We need the *car_of* function after all (and addition to *car*): $\forall c:car(car\_of(m),m)$. The result of applying *car_of* to *m* is *m*'s car. The final set of predicates is: $mega\_star(prince) \forall m: mega\_star(m) \rightarrow rich(m) \forall c:car(car\_of(m),m). \forall c,m: car(c,m) \wedge rich(m) \rightarrow fast(c). \forall c:[fast(c) \wedge \exists m:car(c,m) \rightarrow consume(c)]$. Given this we could conclude: *consume(car_of(prince))*.

---

dave@cs.cf.ac.uk

Next Up Previous
Next: Applications and extensions **Up:** Predicate logic **Previous:** An example

Isa and *instance* relationships

Two attributes *isa* and *instance* play an important role in many aspects of knowledge representation.

The reason for this is that they support *property inheritance*.

isa
-- used to show class inclusion, *e.g. isa(mega_star,rich)*.

instance
-- used to show class membership, *e.g. instance(prince,mega_star).*

From the above it should be simple to see how to represent these in predicate logic.

---

dave@cs.cf.ac.uk

Next Up Previous
Next: Further reading **Up:** Logic Knowledge Representation **Previous:** Isa and instance relationships

Applications and extensions

- First order logic basically extends predicate calculus to allow:
  - functions -- return *objects* not just TRUE/FALSE.
  - equals predicate added.
- Problem solving and theorem proving -- large application areas.
- STRIPS robot planning system employs a first order logic system to enhance its means-ends analysis (GPS) planning. This amalgamation provided a very powerful heuristic search.
- Question answering systems.

---

dave@cs.cf.ac.uk

Next Up Previous
Next: Exercises **Up:** Logic Knowledge Representation **Previous:** Applications and extensions

Further reading

Artificial Intelligence by Rich and Knight covers this topic well.

Knowledge Representation: An AI perspective by H. Reichgelt, Ablex, New York covers all aspects of this topic with good examples.

The Handbook of Artificial Intelligence (Vol. 1, Ch 3) provides a good concise treatment of this area.

See also the *Expert Systems* course.

---

dave@cs.cf.ac.uk

Next Up Previous
Next: Procedural Knowledge Representations **Up:** Logic Knowledge Representation **Previous:** Further reading

Exercises

1. Assume the following facts:
   - Steve only likes easy courses.
   - Computing courses are hard.
   - All courses in Sociology are easy.
   - ``Society is evil'' is a sociology course.

   Represent these facts in predicate logic and answer the question?

   What course would Steve like?

2. Find out what knowledge representation schemes are used in the STRIPS system.

---

dave@cs.cf.ac.uk

Next Up Previous
Next: Declarative or Procedural? **Up:** Artificial Intelligence II **Previous:** Exercises

Procedural Knowledge Representations

---

dave@cs.cf.ac.uk

Next Up Previous
Next: An Example **Up:** Procedural Knowledge Representations **Previous:** Procedural Knowledge Representations

Declarative or Procedural?

Declarative knowledge representation:

- Static representation -- knowledge about objects, events *etc.* and their relationships and states given.
- Requires a program to know what to do with knowledge and how to do it.

Procedural representation:

- control information necessary to use the knowledge is embedded in the knowledge itself. *e.g.* how to find relevant facts, make inferences *etc.*
- Requires an interpreter to follow instructions specified in knowledge.

---

dave@cs.cf.ac.uk

Next Up Previous
Next: Representing How to Use **Up:** Procedural Knowledge Representations **Previous:** Declarative or Procedural?

An Example

Let us consider what knowledge an alphabetical sorter would need:

- Implicit knowledge that *A* comes before *B etc.*
- This is easy -- really integer comparison of (ASCII) codes for $A, B, \ldots$.
  - All programs contain procedural knowledge of this sort.
- The procedural information here is that knowledge of *how to alphabetise* is represented explicitly in the alphabetisation procedure.
  - A declarative system might have to have explicit facts like *A* comes before *B*, *B* comes before *C etc.*.

---

dave@cs.cf.ac.uk

Next Up Previous
Next: Further reading **Up:** Procedural Knowledge Representations **Previous:** An Example

Representing How to Use Knowledge

Need to represent *how to control* the processing:

direction
-- indicate the direction an implication could be used. *E.g.* To prove something can fly show it is a bird. $fly(x) \rightarrow bird(x)$.

Knowledge to achieve goal
-- specify what knowledge might be needed to achieve a specific goal. For example to prove something is a bird try using two facts *has_wings* and *has_feathers* to show it.

---

dave@cs.cf.ac.uk

Next Up Previous
Next: Exercises **Up:** Procedural Knowledge Representations **Previous:** Representing How to Use

Further reading

Knowledge Representation: An AI perspective by H. Reichgelt, Ablex, New York covers all aspects of this topic with good examples.

The Handbook of Artificial Intelligence (Vol. 1, Ch 3) provides a good concise treatment of this area.

Winograd's article in Bobrow and Collins deal with fundamental issues and also their robot planner in *Representation and Understanding*, Academic Press (1975) deals with declarative verses procedural issues.

---

dave@cs.cf.ac.uk

Next Up Previous
Next: Weak Slot and Filler **Up:** Procedural Knowledge Representations **Previous:** Further reading

Exercises

1. Discuss how procedural methods may be used to solve the following problems:
   - Natural Language Understanding
   - The Games of Nim and Kalah (see last years notes for rules).
   - Path Planning type tasks.

---

dave@cs.cf.ac.uk

Next Up Previous
Next: Why use this data **Up:** Artificial Intelligence II **Previous:** Exercises

Weak Slot and Filler Structures

We have already met this type of structure when discussing inheritance in the last lecture. We will now study this in more detail.

---

dave@cs.cf.ac.uk

Next Up Previous
Next: Semantic Nets **Up:** Weak Slot and Filler **Previous:** Weak Slot and Filler

Why use this data structure?

- It enables attribute values to be retrieved quickly
  - assertions are indexed by the entities
  - binary predicates are indexed by first argument. *E.g. team(Mike-Hall , Cardiff)*.
- Properties of relations are easy to describe .
- It allows ease of consideration as it embraces aspects of object oriented programming.

So called because:

- A *slot* is an attribute value pair in its simplest form.
- A *filler* is a value that a slot can take -- could be a numeric, string (or any data type) value or a pointer to another slot.
- A *weak* slot and filler structure does not consider the *content* of the representation.

We will study two types:

- Semantic Nets.
- Frames.

---

dave@cs.cf.ac.uk

Next Up Previous
Next: Representation in a Semantic **Up:** Weak Slot and Filler **Previous:** Why use this data

Semantic Nets

The major idea is that:

- The meaning of a concept comes from its relationship to other concepts, and that,
- The information is stored by interconnecting nodes with labelled arcs.

---

dave@cs.cf.ac.uk

Next Up Previous
Next: Inference in a Semantic **Up:** Semantic Nets **Previous:** Semantic Nets

Representation in a Semantic Net

The physical attributes of a person can be represented as in Fig. 9.

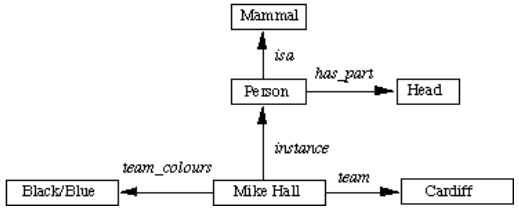


Fig. 9 A Semantic Network

These values can also be represented in logic as: *isa(person, mammal), instance(Mike-Hall, person) team(Mike-Hall, Cardiff)*

We have already seen how conventional predicates such as *lecturer(dave)* can be written as *instance (dave, lecturer)* Recall that *isa* and *instance* represent inheritance and are popular in many knowledge representation schemes. But we have a problem: *How we can have more than 2 place predicates in semantic nets? E.g. score(Cardiff, Llanelli, 23-6)* Solution:

- Create new nodes to represent new objects either contained or alluded to in the knowledge, *game* and *fixture* in the current example.
- Relate information to nodes and fill up slots (Fig: 10).

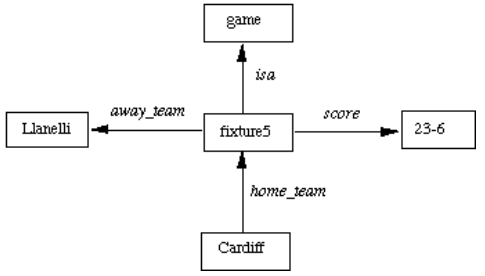


Fig. 10 A Semantic Network for *n*-Place Predicate

As a more complex example consider the sentence: *John gave Mary the book*. Here we have several aspects of an event.

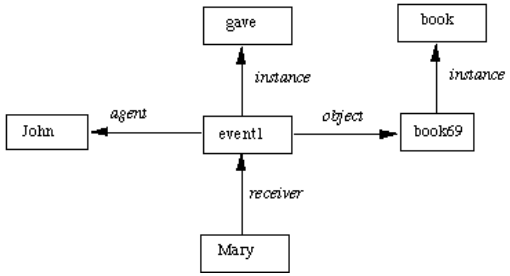


Fig. 11 A Semantic Network for a Sentence

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Extending Semantic Nets **Up:** Semantic Nets **Previous:** Representation in a Semantic

Inference in a Semantic Net

Basic inference mechanism: *follow links between nodes*.

Two methods to do this:

Intersection search

-- the notion that *spreading activation* out of two nodes and finding their intersection finds relationships among objects. This is achieved by assigning a special tag to each visited node.

Many advantages including entity-based organisation and fast parallel implementation. However very structured questions need highly structured networks.

Inheritance

-- the *isa* and *instance* representation provide a mechanism to implement this.

Inheritance also provides a means of dealing with *default reasoning*. *E.g.* we could represent:

- Emus are birds.
- Typically birds fly and have wings.
- Emus run.

in the following Semantic net:

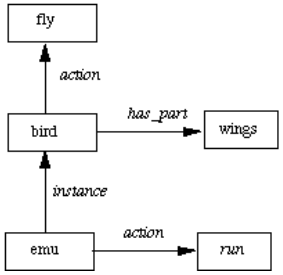


Fig. 12 A Semantic Network for a Default Reasoning

In making certain inferences we will also need to *distinguish between the link that defines a new entity and holds its value and the other kind of link that relates two existing entities*. Consider the example shown where the height of two people is depicted and we also wish to compare them.

We need extra nodes for the concept as well as its value.

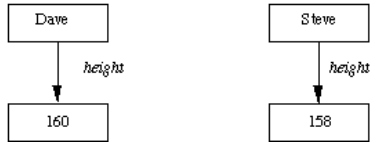


Fig. 12 Two heights

Special procedures are needed to process these nodes, but without this distinction the analysis would be very limited.

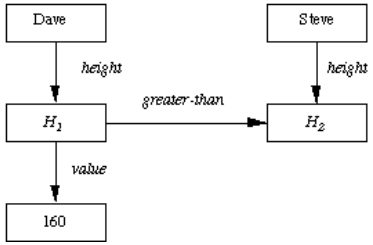


Fig. 12 Comparison of two heights

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Frames **Up:** Semantic Nets **Previous:** Inference in a Semantic

Extending Semantic Nets

Here we will consider some extensions to Semantic nets that overcome a few problems (see Exercises) or extend their expression of knowledge.

Partitioned Networks *Partitioned* Semantic Networks allow for:

- propositions to be made without commitment to truth.
- expressions to be quantified.

Basic idea: *Break network into **spaces** which consist of groups of nodes and arcs and regard each **space** as a node.*

Consider the following: *Andrew believes that the earth is flat.* We can encode the proposition *the earth is flat* in a *space* and within it have nodes and arcs the represent the fact (Fig. 15). We can the have nodes and arcs to link this *space* the the rest of the network to represent Andrew's belief.

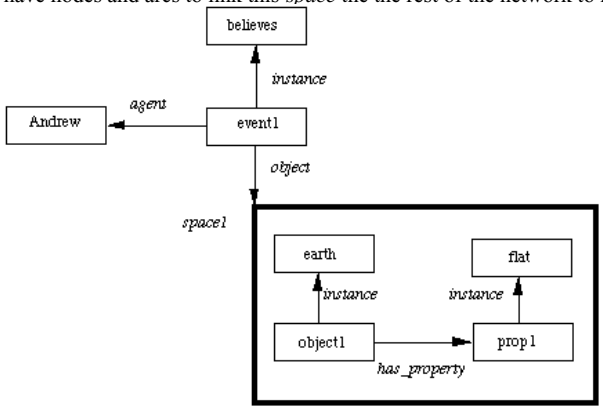


Fig. 12 Partitioned network

Now consider the quantified expression: *Every parent loves their child* To represent this we:

- Create a *general statement*, GS, special class.
- Make node g an instance of GS.
- Every element will have at least 2 attributes:
  - a *form* that states which relation is being asserted.
  - one or more *forall* (∀) or *exists* (∃) connections -- these represent universally quantifiable variables in such statements *e.g. x, y* in $\forall x : parent(x) \rightarrow \exists y: child(y) \wedge loves(x,y)$

Here we have to construct two *spaces one* for each *x,y*. **NOTE:** We can express ∃ variables as *existentially qualified* variables and express the event of *love* having an agent *p* and receiver *b* for every parent *p* which could simplify the network (See Exercises).

Also If we change the sentence to *Every parent loves child* then the node of the object being acted on

(*the child*) lies outside the form of the general statement. Thus it is not viewed as an existentially qualified variable whose value may depend on the agent. (See Exercises and Rich and Knight book for examples of this) So we could construct a partitioned network as in Fig. 16

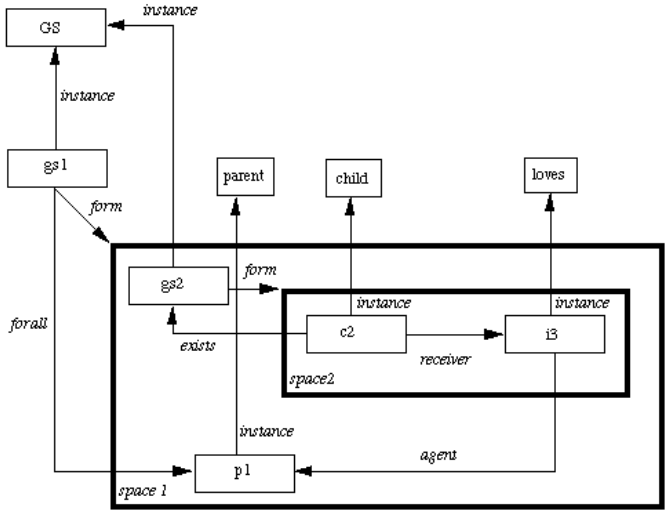


Fig. 12 ∀ Partitioned network

Next Up Previous
Next: Frames **Up:** Semantic Nets **Previous:** Inference in a Semantic

dave@cs.cf.ac.uk

Next Up Previous
Next: Frame Knowledge Representation **Up:** Weak Slot and Filler **Previous:** Extending Semantic Nets

Frames

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).

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Interpreting frames **Up:** Frames **Previous:** Frames

Frame Knowledge Representation

Consider the example first discussed in Semantics Nets (Section 6.2.1):

```
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, ... }
```

Figure: A simple frame system

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*

Figure: A Metaclass frame system

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.

Slots as Objects

How can we to represent the following properties in frames?

- Attributes such as *weight, age* be attached and make sense.
- Constraints on values such as *age* being less than a hundred
- Default values
- Rules for inheritance of values such as children inheriting parent's names
- Rules for computing values
- Many values for a slot.

A slot is a relation that maps from its domain of classes to its range of values.

A relation is a set of ordered pairs so one relation is a subset of another.

Since slot is a set the set of all slots can be represent by a metaclass called *Slot*, say.

Consider the following:

```
SLOT
                isa:            Class
                instance:               Class
                domain:
                range:
                range-constraint:
                definition:
                default:
                to-compute:
                single-valued:
Coach
                instance:               SLOT
                domain:                 Rugby-Team
                range:                  Person
                range-constraint:               λx (experience x.manager)
                default:
                single-valued:                  TRUE


Colour
                instance:               SLOT
                domain:                 Physical-Object
                range:                  Colour-Set
                single-valued:                  FALSE
Team-Colours
                instance:               SLOT
                isa:            Colour
                domain:                 team-player
```

```
                range:                  Colour-Set
                range-constraint:               not Pink
                single-valued:                  FALSE
Position
                instance:               SLOT
                domain:                 Rugby-Player
                range:                  { Back, Forward, Reserve }
                to-compute:             λ x.position
                single-valued:                  TRUE
```

NOTE the following:

- Instances of *SLOT* are slots
- Associated with *SLOT* are attributes that each instance will inherit.
- Each slot has a domain and range.
- Range is split into two parts one the class of the elements and the other is a constraint which is a logical expression if absent it is taken to be true.
- If there is a value for default then it must be passed on unless an instance has its own value.
- The *to-compute* attribute involves a procedure to compute its value. *E.g.* in *Position* where we use the dot notation to assign values to the slot of a frame.
- Transfers through lists other slots from which values can be derived from inheritance.

Next Up Previous
Next: Interpreting frames **Up:** Frames **Previous:** Frames

dave@cs.cf.ac.uk

Next Up Previous
Next: Further Reading **Up:** Frames **Previous:** Frame Knowledge Representation

Interpreting frames

A frame system interpreter must be capable of the following in order to exploit the frame slot representation:

- Consistency checking -- when a slot value is added to the frame relying on the domain attribute and that the value is legal using range and range constraints.
- Propagation of *definition* values along *isa* and *instance* links.
- Inheritance of default. values along *isa* and *instance* links.
- Computation of value of slot as needed.
- Checking that only correct number of values computed.

See *Exercises* for further instances of drawing inferences *etc.* from frames.

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Exercises **Up:** Weak Slot and Filler **Previous:** Interpreting frames

Further Reading

Artificial Intelligence by Rich and Knight covers this topic well.

Knowledge Representation: An AI perspective by H. Reichgelt, Ablex, New York covers all aspects of this topic with good examples.

The Handbook of Artificial Intelligence (Vol. 1, Ch 3) provides a good concise treatment of this area. Volume 3 Chapter 11 also gives good background information on Quinlan's semantic memory system. The HAM, ACT and MEMOD systems are also dealt with.

Artificial Intelligence Programming by Charniak *et al*. gives Example LISP code for constructing Semantic nets and forms.

See also the *Expert Systems* course.

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Strong Slot and Filler **Up:** Weak Slot and Filler **Previous:** Further Reading

Exercises

1. Construct Semantic Net representations of the following:
    1. Dave is Welsh, Dave is a Lecturer.
    2. Paul leant his new Frank Zappa CD to his best friend.
2. Find out how the *SNePS* system aims to improve the expressiveness of semantic nets.
3. Represent the following in a Semantic Net and Comment on how the SNePS system helps.
    1. *Mike and Mary's telephone number is the same.*
    2. *John believes that Mike and Mary's telephone number is the same.*
4. Represent the following in partitioned semantic networks:
    1. Every player kicked a ball.
    2. All players like the referee.
    3. Andrew believes that there is a fish with lungs.
5. Simplify Fig. 16 that represents $\forall x : parent(x) \rightarrow \exists y: baby(y) \wedge loves(x,y)$

    so that we can express $\exists$ variables as *existentially qualified* variables and express the event of *love* having an agent p and receiver b for every parent p.

6. Pick a problem area and represent the knowledge in frame based system.
7. Devise algorithms that enable reasoning with frames. Discuss how
    1. Inference through inheritance can be achieved.
    2. Matching can be achieved.
8. What are the advantages of a frame based knowledge representation?
9. What problems do you envisage a a frame based knowledge representation having? Give examples of knowledge hard to represent in a frame. How could some problems be overcome?

    HINT: address issues to do with semantics, inheritance, expressiveness.

10. What programming languages would be suited to implement a semantic network and frames?

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Conceptual Dependency (CD) **Up:** Artificial Intelligence II **Previous:** Exercises

Strong Slot and Filler Structures

Strong Slot and Filler Structures typically:

- Represent links between objects according to more *rigid* rules.
- Specific notions of what types of object and relations between them are provided.
- Represent knowledge about common situations.

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Scripts **Up:** Strong Slot and Filler **Previous:** Strong Slot and Filler

Conceptual Dependency (CD)

Conceptual Dependency originally developed to represent knowledge acquired from natural language input.

The goals of this theory are:

- To help in the drawing of inference from sentences.
- To be independent of the words used in the original input.
- That is to say: *For any 2 (or more) sentences that are identical in meaning there should be only one representation of that meaning.*

It has been used by many programs that portend to understand English (*MARGIE, SAM, PAM*). CD developed by Schank *et al.* as were the previous examples.

CD provides:

- a structure into which nodes representing information can be placed
- a specific set of primitives
- at a given level of granularity.

Sentences are represented as a series of diagrams depicting actions using both abstract and real physical situations.

- The agent and the objects are represented
- The actions are built up from a set of primitive acts which can be modified by tense.

Examples of Primitive Acts are:

ATRANS
-- Transfer of an abstract relationship. *e.g. give*.
PTRANS
-- Transfer of the physical location of an object. *e.g. go*.
PROPEL
-- Application of a physical force to an object. *e.g. push*.
MTRANS
-- Transfer of mental information. *e.g. tell*.
MBUILD
-- Construct new information from old. *e.g. decide*.
SPEAK
-- Utter a sound. *e.g. say*.

ATTEND
-- Focus a sense on a stimulus. *e.g. listen, watch.*
MOVE
-- Movement of a body part by owner. *e.g. punch, kick*.
GRASP
-- Actor grasping an object. *e.g. clutch*.
INGEST
-- Actor ingesting an object. *e.g. eat*.

EXPEL
-- Actor getting rid of an object from body. *e.g. ????*.

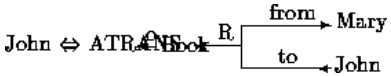
Six primitive conceptual categories provide *building blocks* which are the set of allowable dependencies in the concepts in a sentence:

PP
-- Real world objects.
ACT
-- Real world actions.
PA
-- Attributes of objects.
AA
-- Attributes of actions.
T
-- Times.
LOC
-- Locations.

How do we connect these things together?

Consider the example:

John gives Mary a book



- Arrows indicate the direction of dependency. Letters above indicate certain relationships:
  o
  -- object.
  R
  -- recipient-donor.
  I
  -- instrument *e.g. eat with a spoon*.
  D
  -- destination *e.g. going home.*

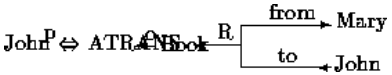
- Double arrows (⇔) indicate *two-way* links between the actor (PP) and action (ACT).
- The actions are built from the set of primitive acts (see above).
  - These can be modified by *tense etc.*

    The use of tense and mood in describing events is extremely important and schank introduced the following modifiers:

    p
    -- past
    f
    -- future
    t
    -- transition
    t_s
    -- start transition
    t_f
    -- finished transition
    k
    -- continuing
    ?
    -- interrogative
    /
    -- negative
    delta
    -- timeless
    c
    -- conditional
    the absence of any modifier implies the *present tense*.

So the *past tense* of the above example:

John gave Mary a book becomes:



The ⇔ has an object (actor), PP and action, ACT. *I.e.* PP ⇔ ACT. The triplearrow (⟺) is also a two link but between an object, PP, and its attribute, PA. *I.e.* PP ⟺ PA.

It represents *isa* type dependencies. *E.g*

Dave ⟺ lecturerDave is a lecturer.

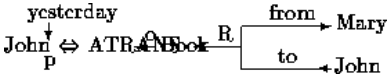
Primitive states are used to describe many state descriptions such as height, health, mental state, physical state.

There are many more physical states than primitive actions. They use a numeric scale.

E.g. John ⟺ height(+10) *John is the tallest* John ⟺ height(< average) *John is short* Frank Zappa ⟺ health(-10) *Frank Zappa is dead* Dave ⟺ mental_state(-10) *Dave is sad* Vase ⟺ physical_state(-10) *The vase is broken*

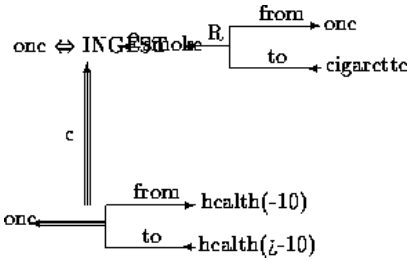
You can also specify things like the time of occurrence in the relation ship.

For Example: *John gave Mary the book yesterday*



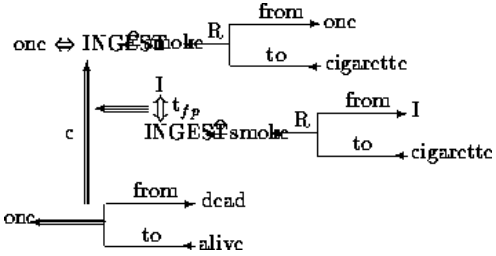
Now let us consider a more complex sentence: *Since smoking can kill you, I stopped* Lets look at how we represent the inference that *smoking can kill*:

- Use the notion of *one* to apply the knowledge to.
- Use the *primitive act* of INGESTing smoke from a cigarette to one.
- Killing is a transition from being alive to dead. We use *triple arrow*s to indicate a transition from one state to another.
- Have a conditional, *c* causality link. The *triple arrow* indicates dependency of one concept on another.



To add the fact that *I stopped smoking*

- Use similar rules to imply that I smoke cigarettes.
- The qualification t_{fp} attached to this dependency indicates that the instance INGESTing smoke has stopped.



Advantages of CD:

- Using these primitives involves fewer inference rules.
- Many inference rules are already represented in CD structure.
- The holes in the initial structure help to focus on the points still to be established.

Disadvantages of CD:

- Knowledge must be decomposed into fairly low level primitives.

- Impossible or difficult to find correct set of primitives.
- A lot of inference may still be required.
- Representations can be complex even for relatively simple actions. Consider:

  Dave bet Frank five pounds that Wales would win the Rugby World Cup.

  Complex representations require a lot of storage

Applications of CD:

MARGIE
(*Meaning Analysis, Response Generation and Inference on English*) -- model natural language understanding.
SAM
(*Script Applier Mechanism*) -- Scripts to understand stories. See next section.
PAM
(*Plan Applier Mechanism*) -- Scripts to understand stories.

Schank *et al.* developed all of the above.

Next Up Previous

Next: Scripts **Up:** Strong Slot and Filler **Previous:** Strong Slot and Filler

dave@cs.cf.ac.uk

Next Up Previous
Next: CYC **Up:** Strong Slot and Filler **Previous:** Conceptual Dependency (CD)

Scripts

A *script* is a structure that prescribes a set of circumstances which could be expected to follow on from one another.

It is similar to a thought sequence or a chain of situations which could be anticipated.

It could be considered to consist of a number of slots or frames but with more specialised roles.

Scripts are beneficial because:

- Events tend to occur in known runs or patterns.
- Causal relationships between events exist.
- Entry conditions exist which allow an event to take place
- Prerequisites exist upon events taking place. *E.g.* when a student progresses through a degree scheme or when a purchaser buys a house.

The components of a script include:

Entry Conditions
-- these must be satisfied before events in the script can occur.
Results
-- Conditions that will be true after events in script occur.
Props
-- Slots representing objects involved in events.
Roles
-- Persons involved in the events.
Track
-- Variations on the script. Different tracks may share components of the same script.
Scenes
-- The sequence of *events* that occur. *Events* are represented in *conceptual dependency* form.

Scripts are useful in describing certain situations such as robbing a bank. This might involve:

- Getting a gun.
- Hold up a bank.
- Escape with the money.

Here the *Props* might be

- Gun, *G*.
- Loot, *L*.
- Bag, *B*
- Get away car, *C*.

The *Roles* might be:

- Robber, *S*.
- Cashier, *M*.

- Bank Manager, *O*.
- Policeman, *P*.

The *Entry Conditions* might be:

- *S* is poor.
- *S* is destitute.

The *Results* might be:

- *S* has more money.
- *O* is angry.
- *M* is in a state of shock.
- *P* is shot.

There are 3 scenes: obtaining the gun, robbing the bank and the getaway.

The full Script could be described in Fig 19.

Script: ROBBERY	*Track: Successful Snatch*
Props: G = Gun, L = Loot, B= Bag, C = Get away car.	*Roles*: R = Robber, M = Cashier, O = Bank Manager, P = Policeman.
Entry Conditions: R is poor. R is destitute.	*Results*: R has more money. O is angry. M is in a state of shock. P is shot.
Scene 1: Getting a gun R PTRANS R into Gun Shop R MBUILD R choice of G R MTRANS choice. R ATRANS buys G (go to scene 2)	
Scene 2 Holding up the bank R PTRANS R into bank R ATTEND eyes M, O and P R MOVE R to M position R GRASP G R MOVE G to point to M R MTRANS "Give me the money or ELSE" to M P MTRANS "Hold it Hands Up" to R R PROPEL shoots G P INGEST bullet from G M ATRANS L to M M ATRANS L puts in bag, B M PTRANS exit O ATRANS raises the alarm (go to scene 3)	
Scene 3: The getaway M PTRANS C	

Fig. 12 Simplified Bank Robbing Script

Some additional points to note on Scripts:

- If a particular script is to be applied it must be activated and the activating depends on its significance.
- If a topic is mentioned in passing then a pointer to that script could be held.
- If the topic is important then the script should be opened.
- The danger lies in having too many active scripts much as one might have too many windows open on the screen or too many recursive calls in a program.
- Provided events follow a known trail we can use scripts to represent the actions involved and use them to answer detailed questions.
- Different trails may be allowed for different outcomes of Scripts (*e.g.* The bank robbery goes wrong).

Advantages of Scripts:

- Ability to predict events.
- A single coherent interpretation may be build up from a collection of observations.

Disadvantages:

- Less general than frames.
- May not be suitable to represent all kinds of knowledge.

Next Up Previous

Next: CYC **Up:** Strong Slot and Filler **Previous:** Conceptual Dependency (CD)

dave@cs.cf.ac.uk

Next Up Previous

Next: Further Reading **Up:** Strong Slot and Filler **Previous:** Scripts

CYC

What is CYC?

- An ambitious attempt to form a very large knowledge base aimed at capturing commonsense reasoning.
- Initial goals to capture knowledge from a hundred randomly selected articles in the En*CYC*lopedia Britannica.
- Both Implicit and Explicit knowledge encoded.
- Emphasis on study of underlying information (assumed by the authors but not needed to tell to the readers.

Example: Suppose we read that *Wellington learned of Napoleon's death*

Then we (humans) can conclude *Napoleon never new that Wellington had died*.

How do we do this?

We require special implicit knowledge or commonsense such as:

- We only die once.
- You stay dead.
- You cannot learn of anything when dead.
- Time cannot go backwards.

Why build large knowledge bases:

Brittleness
-- Specialised knowledge bases are *brittle*. Hard to encode new situations and non-graceful degradation in performance. Commonsense based knowledge bases should have a firmer foundation.

Form and Content
-- Knowledge representation may not be suitable for AI. Commonsense strategies could point out where difficulties in content may affect the form.

Shared Knowledge
-- Should allow greater communication among systems with common bases and assumptions.

How is CYC coded?

- By hand.
- Special CYCL language:
  - LISP like.
  - Frame based
  - Multiple inheritance
  - Slots are fully fledged objects.
  - Generalised inheritance -- any link not just *isa* and *instance*.

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Exercises **Up:** Strong Slot and Filler **Previous:** CYC

Further Reading

Artificial Intelligence by Rich and Knight covers these topic well. In particular see the sections on scripts for more detail on more complicated scripts with varying tracks. And also a more comprehensive treatment of CYC.

Knowledge Representation: An AI perspective by H. Reichgelt, Ablex, New York covers all aspects of this topic with good examples.

The Essentials of AI by Ginsberg provides some useful background information

The Handbook of Artificial Intelligence (Vol. 1, Ch 3) provides a good concise treatment of this area. Details of MARGIE, SAM and PAM may also be found in Vol. 1, Ch 4.

Examples of how MARGIE and SAM is coded in LISP is given in *AI with Common Lisp* by Noyes.

---

dave@cs.cf.ac.uk

Next Up Previous

Next: Reasoning with Uncertainty: Non-Monotonic **Up:** Strong Slot and Filler **Previous:** Further Reading

Exercises

1. Construct CD representation of the following:
    1. John begged Mary for a pencil.
    2. Jim stirred his coffee with a spoon.
    3. Dave took the book off Jim.
    4. On my way home, I stopped to fill my car with petrol.
    5. I heard strange music in the woods.
    6. Drinking beer makes you drunk.
    7. John killed Mary by strangling her.
2. Try capturing the differences between the following in CD:

    1. John slapped Dave, John punched Dave.
    2. Sue likes Prince, Sue adores Prince.
3. Rewrite the script given in the lecture so that the Bank robbery goes wrong.
4. Write a script to allow for both outcome of the Bank robbery: Getaway and going wrong and getting caught.
5. Write a script for enrolling as a student.
6. Find out about how MARGIE, SAM and PAM are implemented. In particular pay attention to their reasoning and inference mechanisms with the knowledge.
7. Find out how the CYCL language represents knowledge.
8. What are the two levels of representation in the constraints of CYC?
9. Find out the relevance of *Meta-Knowledge* in CYC and how it controls the interpretations of knowledge.
10. Find out what levels of concepts CYC has in its *ontology*.
    1. Where should the following concepts be placed in this ontology
        - dog
        - court case
        - South Wales Echo
        - Wales
        - Pint of Brains Dark
        - The Open Golf Championship.

---

dave@cs.cf.ac.uk

Next Up Previous

Next: What is reasoning? **Up:** Artificial Intelligence II **Previous:** Exercises

Reasoning with Uncertainty: Non-Monotonic Reasoning

---

dave@cs.cf.ac.uk