# CSC 480: Artificial Intelligence


***Franz J. Kurfess***

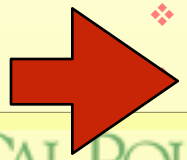
*Visiting Professor*
*Department of Computer Science and Mathematics*
*Munich University of Applied Sciences*
*Germany*

*Professor*
*Computer Science Department*
*California Polytechnic State University*
*San Luis Obispo, CA, U.S.A.*



# Course Overview

- **Introduction**
- **Intelligent Agents**
- **Search**
  - problem solving through search
  - uninformed search
  - informed search
- **Games**
  - games as search problems
- **Knowledge and Reasoning**
  - reasoning agents
  - propositional logic
  - predicate logic
  - knowledge-based systems
- **Learning**
  - learning from observation
  - neural networks
- **Conclusions**



# Knowledge Engineering

- **often performed by a knowledge engineer**
  - intermediary between users, domain experts and programmers
  - must know enough about the domain
    - objects
    - relationships
  - must be comfortable in the representation language
    - encoding of objects and relationships
  - must understand the implementation of the inference mechanism
    - performance issues
    - explanation of results

# Knowledge Engineering Methodology

- **domain**
  - objects, facts
- **vocabulary**
  - predicates, functions, constants in the language of logic
  - ideally results in an ontology
- **background knowledge**
  - general knowledge about the domain
  - specify the axioms about the terms in the ontology
- **specific problem**
  - description of the instances of concepts and objects that determine the problem to be investigated
- **queries**
  - requests answers from the knowledge base/inference mechanism

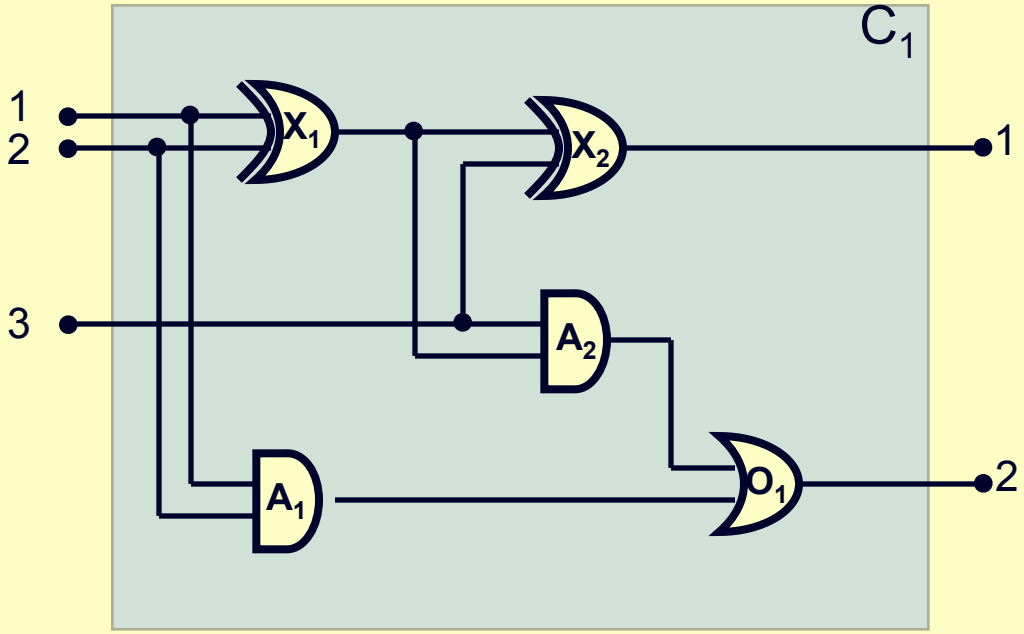
# Example: Electronic Circuits

- **domain**
  - digital circuits, wires, gates, signals, input and output terminals
  - types of gates: *AND, OR, XOR, NOT*
- **vocabulary**
  - constants for naming gates: $X_1$, $X_2$, …
  - functions for gate types
    - $Type(X_1) = XOR$
  - functions for terminals
    - $Out(1, X_1)$ for the only output of $X_1$
    - $In(n, X_1)$ for the input n of $X_1$
  - predicates for connectivity
    - $Connected(Out(1, X_1), In(2, X_2))$
  - two objects and a function for the signal values
    - *On, Off*
    - $Signal(Out(1, X_1))$

# Example: Electronic Circuits (cont.)

- **background knowledge**
  - two connected terminals have the same signal
    - $\forall\ t_1, t_2 \quad Connected(t_1, t_2) \Rightarrow Signal(t_1) = Signal(t_2)$
  - signals must be either on of off, but not both
    - $\forall\ t \quad Signal(t) = On \vee Signal(t) = Off$
      $On \neq Off$
  - connections go both ways (commutative)
    - $\forall\ t_1, t_2 \quad Connected(t_1, t_2) \Leftrightarrow Connected(t_2, t_1)$
  - definition of *OR*
    - $\forall\ g \quad Type(g) = OR \Rightarrow Signal(Out(1,g)) = On \Leftrightarrow \exists\ n\ Signal(In(n,g)) = On$
  - definition of *AND*
    - $\forall\ g \quad Type(g) = AND \Rightarrow Signal(Out(1,g)) = Off \Leftrightarrow \exists\ n\ Signal(In(n,g)) = Off$
  - definition of *XOR* (sometimes denoted by $\oplus$)
    - $\forall\ g \quad Type(g) = XOR \Rightarrow$
      $Signal(Out(1,g)) = On \Leftrightarrow Signal(In(1,g)) \neq Signal(In(2,g))$
  - definition of *NOT*
    - $\forall\ g \quad Type(g) = NOT \Rightarrow Signal(Out(1,g)) \neq Signal(In(1,g))$

# Example: Electronic Circuits (cont.)




- **specific problem: circuit $C_1$ to be modeled**

  *Type($X_1$) = XOR; Type($X_2$) = XOR;*
  *Type($A_1$) = AND; Type($A_2$) = AND;*
  *Type($O_1$) = OR;*
  *Connected(In(1, $C_1$), In(1, $X_1$));*
  *Connected(In(1, $C_1$), In(1, $A_1$));*
  *Connected(In(2, $C_1$), In(2, $X_1$));*
  *Connected(In(2, $C_1$), In(2, $A_1$));*
  *Connected(In(3, $C_1$), In(2, $X_2$));*
  *Connected(In(3, $C_1$), In(1, $A_2$));*
  *Connected(Out(1, $X_1$), In(1, $X_2$));*
  *Connected(Out(1, $X_1$), In(2, $A_2$));*
  *Connected(Out(1, $A_2$), In(1, $O_1$));*
  *Connected(Out(1, $A_1$), In(2, $O_1$));*
  *Connected(Out(1, $X_2$), Out(1, $C_1$));*
  *Connected(Out(1, $O_1$), Out(2, $C_1$));*

# Example: Electronic Circuits (cont.)

- **queries**
  - When is the first output of $C_1$ (sum bit) off and the second one (carry bit) on?

    $\exists\, i_1, i_2, i_3 \;\; Signal(In(1, C_1) = i_1 \wedge Signal(In(2, C_1) = i_2 \wedge$
    $Signal(In(3, C_1) = i_3 \wedge$
    $Signal(Out(1, C_1) = Off \wedge Signal(Out(2, C_1) = On$
  - Give all combinations of the values for the terminals of $C_1$

    $\exists\, i_1, i_2, i_3, o_1, o_2 \quad Signal(In(1, C_1) = i_1 \wedge Signal(In(2, C_1) = i_2 \wedge$
    $Signal(In(3, C_1) = i_3 \wedge$
    $Signal(Out(1, C_1) = o_1 \wedge Signal(Out(2, C_1) = o_2$

# Knowledge Representation (KR)

**Knowledge Types**
**KR Methods**

Production Rules

Semantic Nets

Schemata and Frames

Logic

# Types of Knowledge

- **a priori knowledge**
  - comes before knowledge perceived through senses
  - considered to be universally true
- **a posteriori knowledge**
  - knowledge verifiable through the senses
  - may not always be reliable
- **procedural knowledge**
  - knowing how to do something
- **declarative knowledge**
  - knowing that something is true or false
- **tacit knowledge**
  - knowledge not easily expressed by language

# Knowledge in Expert Systems

- **Conventional Programming**

Data Structures
+ Algorithms
= Programs

N. Wirth

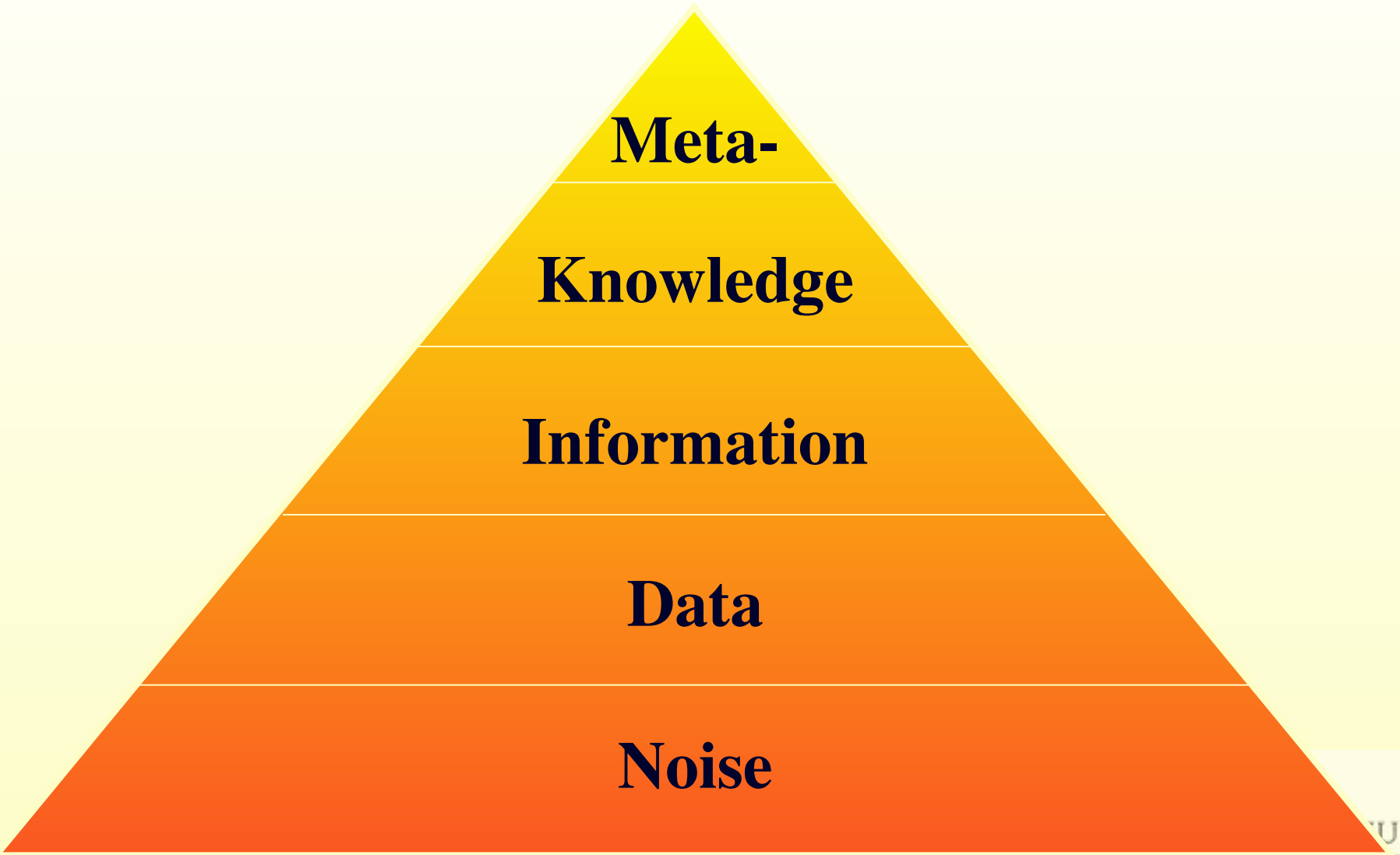
**Knowledge-Based Systems**

Knowledge
+ Inference
= Expert System

# Terminology

- **Data**
  - fixed relations between individual items
  - often arranged as vectors or arrays
  - interpretation is usually provided for the collection of data as a whole
- **Knowledge**
  - separate, possibly dynamic relations between individual items
  - interpretation must be done for individual items
- **Information**
  - generic term, used in a very general sense
  - precisely defined for information theory

# Knowledge Pyramid



# Knowledge Representation Methods

- **Logic**
  - (subset of) first order predicate logic
- **Production Rules**
  - similar to if-then rules
- **Semantic Nets**
  - graph-oriented representation
- **Schemata and Frames**
  - templates

# Logic

- **first-order predicate logic**
  - Horn clauses
  - often augmented with higher-order logic constructs
    - equality
    - math functions
  - procedural constructs
    - performance, program evaluation control
- **fuzzy logic**
  - used mostly for control applications
  - some convenient features for KR
    - “linguistic variables”
    - flexible truth values
      - based on membership in fuzzy sets
      - epistemological commitment

# Production Rules

- **frequently used to formulate the knowledge in expert systems**
- **a formal variation is Backus-Naur form (BNF)**
  - metalanguage for the definition of language syntax
  - a grammar is a complete, unambiguous set of production rules for a specific language
  - a parse tree is a graphic representation of a sentence in that language
  - provides only a syntactic description of the language
    - not all sentences make sense

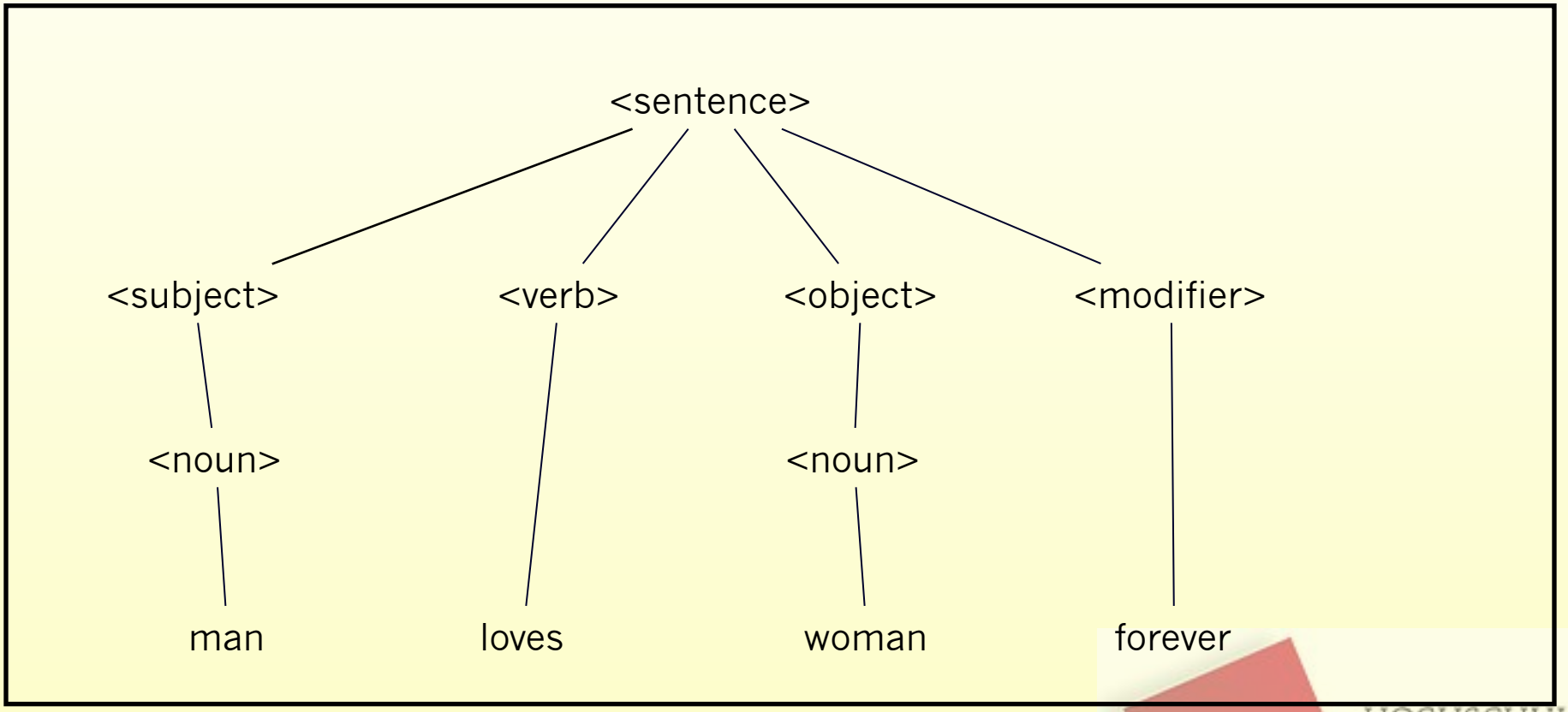
# Example 1 Production Rules

- for a subset of the English language

```
<sentence>   -> <subject> <verb> <object> <modifier>
<subject>    -> <noun>
<object>     -> <noun>
<noun>       -> man | woman
<verb>       -> loves | hates | marries | divorces
<modifier>   -> a little | a lot | forever | sometimes
```

# Example 1 Parse Tree

- **Example sentence:**
  **"man loves woman forever"**



# Example 2 Production Rules

- for a subset of the German language

```
<sentence>        -> <subject phrase> <verb> <object phrase>
<subject phrase>  -> <determiner> <adjective> <noun>
<object phrase>   -> <determiner> <adjective> <noun>

<determiner>      -> der | die | das | den
<noun>            -> Mann | Frau | Kind | Hund | Katze
<verb>            -> mag | schimpft | vergisst|
                     verehrt | verzehrt
<adjective>       -> schoene | starke | laute | duenne
```

# Suitability of Production Rules

- **expressiveness**
  - can relevant aspects of the domain knowledge be stated through rules?
- **computational efficiency**
  - are the computations required by the program feasible?
- **easy to understand?**
  - can humans interpret the rules
- **easy to generate?**
  - how difficult is it for humans to construct rules that reflect the domain knowledge

# Case Studies
# Production Rules

- **sample domains**
  - e.g. theorem proving, determination of prime numbers, distinction of objects (e.g. types of fruit, trees vs. telephone poles, churches vs. houses, animal species)

- **suitability of production rules**
  - basic production rules
    - no salience, certainty factors, arithmetic
  - rules in ES/KBS
    - salience, certainty factors, arithmetic
    - e.g. CLIPS, Jess
  - enhanced rules
    - procedural constructs
      - e.g. loops
    - objects
      - e.g. COOL, Java objects
    - fuzzy logic
      - e.g. FuzzyCLIPS, FuzzyJ

# Trees and Telephone Poles

- **distinguish between stick diagrams of trees and telephone poles**
- **expressiveness**
  - is it possible to specify a set of rules that captures the distinction?
- **computational efficiency**
  - are the computations required by the program feasible?
- **easy to understand?**
  - the rules can be phrased in such a way that humans can understand them with moderate effort
- **easy to generate?**
  - may be difficult; the problem is to identify criteria that are common for trees, but not shared with telephone poles

# Identification and Generation of Prime Numbers

- **identification: for a given number, determine if it is prime**
- **generation: compute the sequence of prime numbers**
- **expressiveness**
  - it is possible to specify identification as well as generation in rules
- **computational efficiency**
  - reasonable if arithmetic is available, very poor if not
- **easy to understand?**
  - the rules can be formulated in an understandable way
- **easy to generate?**
  - may require a good math background

# Advantages of Production Rules

- simple and easy to understand
- straightforward implementation in computers possible
- formal foundations for some variants

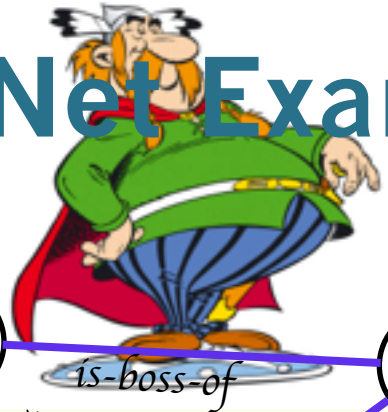
# Problems with Production Rules

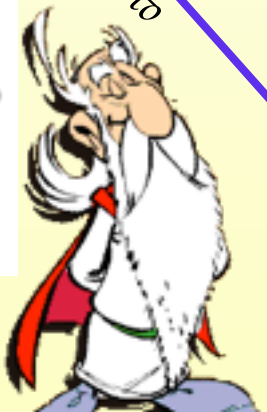
- simple implementations are very inefficient
- some types of knowledge are not easily expressed in such rules
- large sets of rules become difficult to understand and maintain

# Semantic Nets

- **graphical representation for propositional information**
- **originally developed by M. R. Quillian as a model for human memory**
- **labeled, directed graph**
- **nodes represent objects, concepts, or situations**
  - labels indicate the name
  - nodes can be instances (individual objects) or classes (generic nodes)
- **links represent relationships**
  - the relationships contain the structural information of the knowledge to be represented
  - the label indicates the type of the relationship

# Semantix Net Example



Abraracourcix

Astérix

Cétautomatix

Ordralfabetix

Obélix

Idéfix

Panoramix

Gaul

Human

Dog

is-boss-of

is-boss-of

is-a

is-friend-of

is-a

is-a

is-a

is-a

is-a

is-a

buys-from

fights-with

sells-to

AKO

lives-with

takes-care-of

barks-at

[http://www.asterix.tm.fr]





CAL POLY

MÜNCHEN

# Semantix Net Cheats

- **colors**
  - should properly be encoded as separate nodes with relationships to the respective objects
- **font types**
  - implies different types of relationships
  - again would require additional nodes and relationships
- **class relationships**
  - not all dogs live with Gauls
  - AKO (a-kind-of) relationship is special (inheritance)
- **instances**
  - arrows from individual humans to the class Human omitted
    - assumes that AKO allows inheritance
- **directionality**
  - the direction of the arrows matters, not that of the text

# Relationships

- **without relationships, knowledge is an unrelated collection of facts**
  - reasoning about these facts is not very interesting
    - inductive reasoning is possible
- **relationships express structure in the collection of facts**
  - this allows the generation of meaningful new knowledge
    - generation of new facts
    - generation of new relationships

# Types of Relationships

- **relationships can be arbitrarily defined by the knowledge engineer**
  - allows great flexibility
  - for reasoning, the inference mechanism must know how relationships can be used to generate new knowledge
    - inference methods may have to be specified for every relationship
- **frequently used relationships**
  - IS-A
    - relates an instance (individual node) to a class (generic node)
  - AKO (a-kind-of)
    - relates one class (subclass) to another class (superclass)

# Objects and Attributes

- **attributes provide more detailed information on nodes in a semantic network**
  - often expressed as properties
    - combination of attribute and value
  - attributes can be expressed as relationships
    - e.g. has-attribute

# Implementation Questions

- **simple and efficient representation schemes for semantic nets**
  - tables that list all objects and their properties
  - tables or linked lists for relationships
- **conversion into different representation methods**
  - predicate logic
    - nodes correspond variables or constants
    - links correspond to predicates
  - propositional logic
    - nodes and links have to be translated into propositional variables and properly combined with logical connectives

# OAV-Triples

- **object-attribute-value triples**
  - can be used to characterize the knowledge in a semantic net
  - quickly leads to huge tables

| Object | Attribute | Value |
|---|---|---|
| Astérix | profession | warrior |
| Obélix | size | extra large |
| Idéfix | size | petite |
| Panoramix | wisdom | infinite |

# Problems Semantic Nets

- **expressiveness**
  - no internal structure of nodes
  - relationships between multiple nodes
  - no easy way to represent heuristic information
  - extensions are possible, but cumbersome
  - best suited for binary relationships
- **efficiency**
  - may result in large sets of nodes and links
  - search may lead to combinatorial explosion
    - especially for queries with negative results
- **usability**
  - lack of standards for link types
  - naming of nodes
    - classes, instances

# Schemata

- **suitable for the representation of more complex knowledge**
  - causal relationships between a percept or action and its outcome
  - "deeper" knowledge than semantic networks
    - nodes can have an internal structure
  - for humans often tacit knowledge
- **related to the notion of records in computer science**

# Concept Schema

- **abstraction that captures general/typical properties of objects**
  - has the most important properties that one usually associates with an object of that type
    - may be dependent on task, context, background and capabilities of the user, ...
  - similar to stereotypes
- **makes reasoning simpler by concentrating on the essential aspects**
- **may still require relationship-specific inference methods**

# Schema Examples

- **the most frequently used instances of schemata are**
  - frames [Minsky 1975]
  - scripts [Schank 1977]
- **frames consist of a group of slots and fillers to define a stereotypical objects**
- **scripts are time-ordered sequences of frames**

# Frames

- **a frame represents related knowledge about a subject**
  - provides default values for most slots
- **frames are organized hierarchically**
  - allows the use of inheritance
- **knowledge is usually organized according to cause and effect relationships**
  - slots can contain all kinds of items
    - rules, facts, images, video, comments, debugging info, questions, hypotheses, other frames
  - slots can also have procedural attachments
    - procedures that are invoked in specific situations involving a particular slot
      - on creation, modification, removal of the slot value

# Simple Frame Example

| Slot Name | Filler |
|---|---|
| name | Astérix |
| height | small |
| weight | low |
| profession | warrior |
| armor | helmet |
| intelligence | very high |
| marital status | presumed single |

# Overview of Frame Structure

- **two basic elements: slots and facets (fillers, values, etc.);**
- **typically have parent and offspring slots**
  - used to establish a property inheritance hierarchy (e.g., specialization-of)
- **descriptive slots**
  - contain declarative information or data (static knowledge)
- **procedural attachments**
  - contain functions which can direct the reasoning process (dynamic knowledge) (e.g., "activate a certain rule if a value exceeds a given level")
- **data-driven, event-driven ( bottom-up reasoning)**
- **expectation-drive or top-down reasoning**
- **pointers to related frames/scripts - can be used to transfer control to a more appropriate frame**

[Rogers 1999]

# Usage of Frames

- **filling slots in frames**
  - can inherit the value directly
  - can get a default value
  - these two are relatively inexpensive
  - can derive information through the attached procedures (or methods) that also take advantage of current context (slot-specific heuristics)
  - filling in slots also confirms that frame or script is appropriate for this particular situation

[Rogers 1999]

# Restaurant Frame Example

- **generic template for restaurants**
  - different types
  - default values
- **script for a typical sequence of activities at a restaurant**

[Rogers 1999]

# Generic Restaurant Frame

**Generic RESTAURANT Frame**

**Specialization-of**: Business-Establishment

**Types**:

**range**: (Cafeteria, Fast-Food, Seat-Yourself, Wait-To-Be-Seated)

**default**: Seat-Yourself

**if-needed**: IF plastic-orange-counter THEN Fast-Food,
IF stack-of-trays THEN Cafeteria,
IF wait-for-waitress-sign or reservations-made THEN Wait-To-Be-Seated,
OTHERWISE Seat-Yourself.

**Location**:

**range**: an ADDRESS

**if-needed**: (Look at the MENU)

**Name**:

**if-needed**: (Look at the MENU)

**Food-Style**:

**range**: (Burgers, Chinese, American, Seafood, French)

**default**: American

**if-added**: (Update Alternatives of Restaurant)

**Times-of-Operation**:

**range**: a Time-of-Day

**default**: open evenings except Mondays

**Payment-Form**:

**range**: (Cash, CreditCard, Check, Washing-Dishes-Script)

**Event-Sequence**:

**default**: Eat-at-Restaurant Script

**Alternatives**:

**range**: all restaurants with same Foodstyle

# Restaurant Script

**EAT-AT-RESTAURANT Script**

**Props**: (Restaurant, Money, Food, Menu, Tables, Chairs)

**Roles**: (Hungry-Persons, Wait-Persons, Chef-Persons)

**Point-of-View**: Hungry-Persons

**Time-of-Occurrence**: (Times-of-Operation of Restaurant)

**Place-of-Occurrence**: (Location of Restaurant)

**Event-Sequence**:

**first**: Enter-Restaurant Script

**then**: if (Wait-To-Be-Seated-Sign or Reservations)
then Get-Maitre-d's-Attention Script

**then**: Please-Be-Seated Script

**then**: Order-Food-Script

**then**: Eat-Food-Script unless (Long-Wait) when Exit-Restaurant-Angry Script

**then**: if (Food-Quality was better than Palatable)
then Compliments-To-The-Chef Script

**then**: Pay-For-It-Script

**finally**: Leave-Restaurant Script

[Rogers 1999]

# Frame Advantages

- **fairly intuitive for many applications**
  - similar to human knowledge organization
  - suitable for causal knowledge
  - easier to understand than logic or rules
- **very flexible**

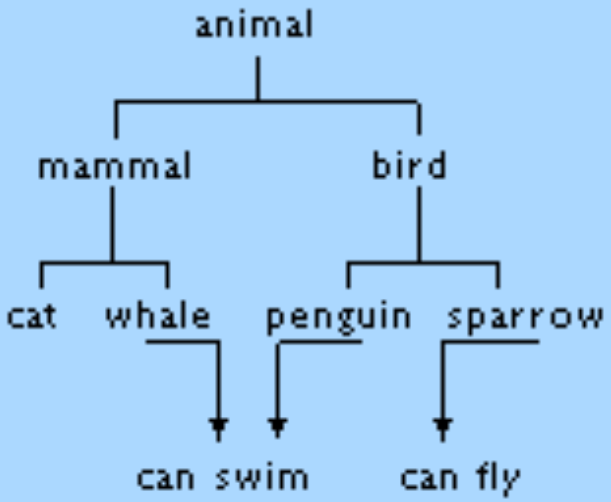
# Frame Problems

- **it is tempting to use frames as definitions of concepts**
  - not appropriate because there may be valid instances of a concept that do not fit the stereotype
  - exceptions can be used to overcome this
    - can get very messy
- **inheritance**
  - not all properties of a class stereotype should be propagated to subclasses
  - alteration of slots can have unintended consequences in subclasses

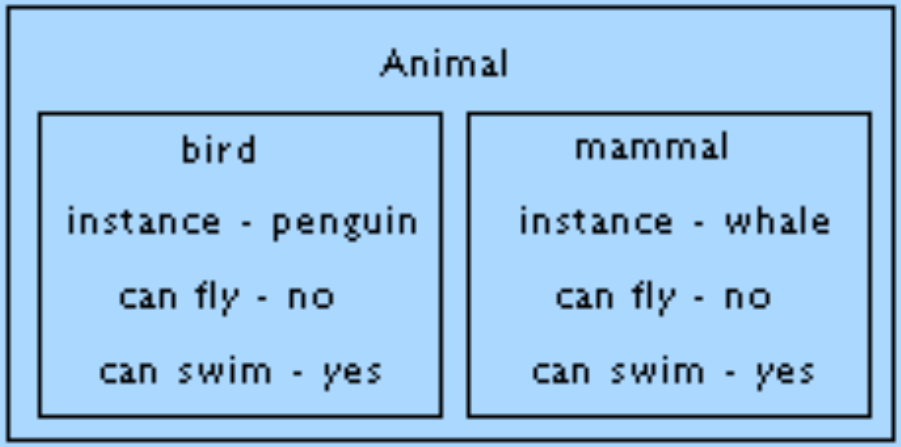
# Example: KR Methods

**STORING INFORMATION**

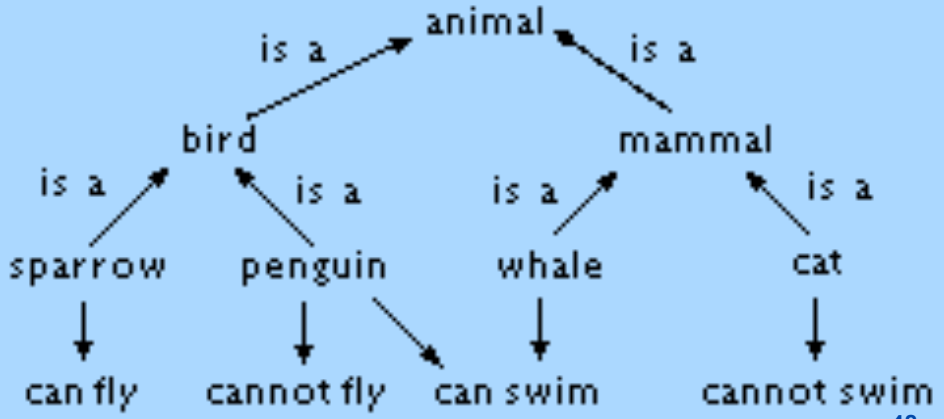
**TREE STRUCTURE**




**FRAME**




**SEMANTIC NET**

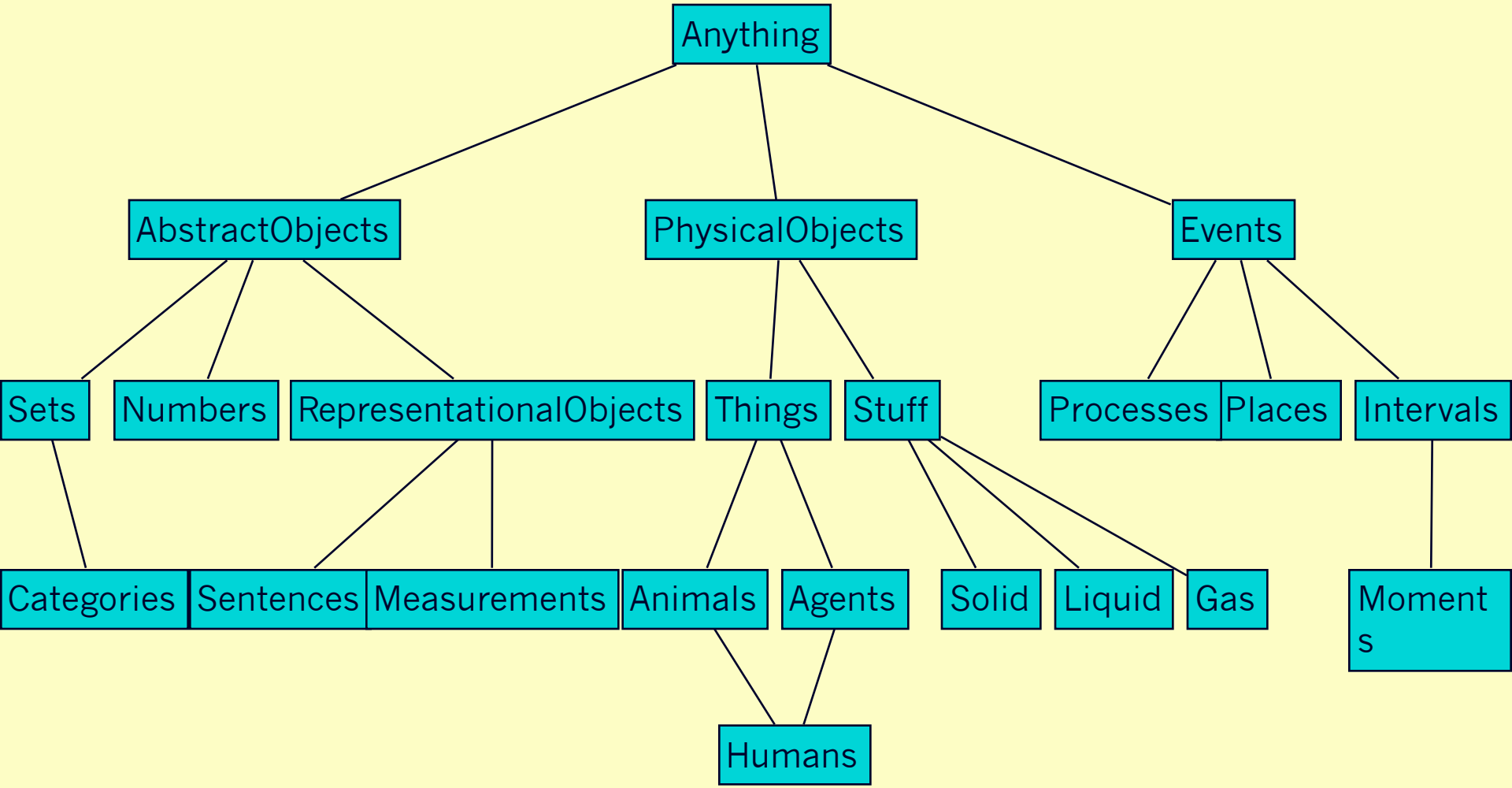



[Guinness 1995]

# Ontologies

- **define the terminology about the objects and their relationships in a systematic way**
  - closely related to taxonomies, classifications
    - ontologies don't have to be hierarchical
    - emphasis on the terms to describe objects, relationships, not on the properties of objects or specific relationships between objects
- **general ontology**
  - convergence of a multitude of special-purpose ontologies
  - should be applicable to any special-purpose domain

# Example General Ontology




*there is no widely agreed upon general (universal) ontology*

# Issues for Ontologies

- categories
- measures
- composite objects
- time, space, and change
- events and processes
- physical objects
- substances
- mental objects and beliefs

# Categories

- **categories are very important for reasoning**
  - almost always organized as taxonomic hierarchies
  - general statements about related objects can be made easily
  - specific properties of instances can either be inferred, or specified explicitly
  - inheritance
    - similar to OO programming

# Measures

- **values for properties of objects**
- **frequently expressed quantitatively**
  - number and unit function
  - allows the use of ordering function on objects

# Composite Objects

- **objects frequently can be decomposed into parts, or composed into larger objects**
- **often expressed through PartOf relation**
  - allows grouping of objects into hierarchies
- **frequently the internal structure of objects is of importance**
  - allows general reasoning about certain aspects of objects
  - in logics, terms can be used to describe the structure of objects

# Time, Space, and Change

- **can be described around the notion of events and processes**
  - events are discrete
  - processes are continuous
    - sometimes also called liquid events

# Events

- **an event is an object with temporal and spatial extent**
  - it occurs somewhere for a certain duration
  - this viewpoint implies some similarities with physical objects
    - also have temporal and spatial extent
  - events may also have internal structure
- **intervals are special events**
  - they include all sub-events during a given time period
- **places are special events**
  - fixed in time, with a temporal extent

# Objects

- **sometimes it is useful to view some types of objects as "fluents"**
  - may change during its existence, but still be considered as an object
    - e.g. the country of Germany
    - the president of the U.S.

# Substances

- **allows the grouping of large numbers of primitive objects into "stuff"**
  - denoted by mass nouns in contrast to count nouns
  - dividing stuff into smaller pieces yields the same type of stuff
    - quantity is different
  - intrinsic properties
    - belong to the substance of an object
    - are retained under subdivisions
  - extrinsic properties
    - come from the specific object as a whole
    - change or get lost under subdivision

# Mental Objects and Beliefs

- **mental objects are "in one's head"**
  - allows the agent to reason about its mental processes
    - some mental processes are the reasoning processes themselves
    - the agent then can perform higher-level reasoning
    - leads to considerable technical and philosophical complications
    - assumed to be a pre-condition for consciousness
- **beliefs are used to make statements about mental objects**

# Semantic Web and RDF

- **Enhancing the World Wide Web**
  - more sophisticated knowledge representation methods
    - meta-data, ontologies
    - commonly used basis: Resource Description Format (RDF)
  - foundation for reasoning

# What is the Resource Description Format (RDF)?

- **Foundation for processing metadata**
  - provides interoperability between applications that exchange machine-understandable information on the Web
- **Arbitrarily expressive language**
- **Syntax-neutral**
  - although it uses XML as the basis

# What can you do with RDF?

- **Resource discovery**
  - better search engine capabilities
- **Cataloging**
  - describing the content and content relationships of a particular Web site, page, or digital library
- **Intelligent software agents**
  - to facilitate knowledge sharing and exchange
- **Content rating**
  - describing collections of pages that represent a single logical "document"
- **Describing intellectual property rights of Web pages**
- **Expressing privacy preferences**
  - user as well
  - privacy policies of a Web site

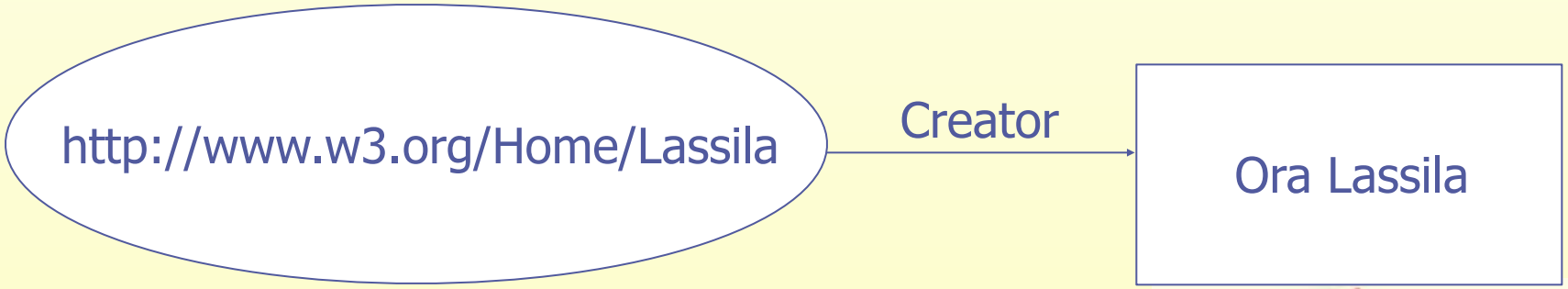
# More RDF

- **RDF should be**
  - domain neutral
  - evolvable
  - capable of acquiring and merging knowledge ("learning")
    - mixing previous knowledge and data with that acquired on the semantic web
- **RDF does not specify a mechanism for reasoning**
  - can be viewed as a simple frame system
  - a reasoning mechanism can be built on top of this frame system

# RDF structure example

Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila

| | |
|---|---|
| Subject (Resource) (*record) | http://www.w3.org/Home/Lassila |
| Predicate (Property) (*column) | Creator |
| Object (literal) (*cell) | "Ora Lassila" |



# The Semantic Web and (vs?) Knowledge Representation

**ENC 2004, September 2004**

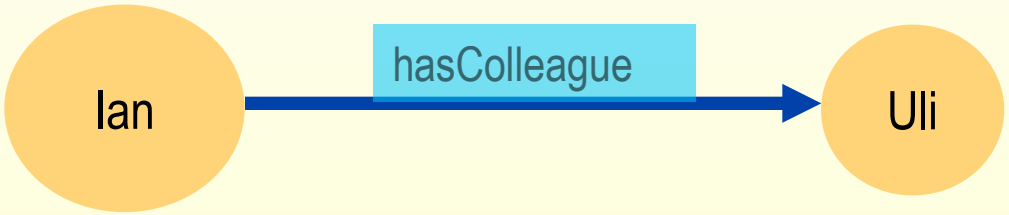
Peter F. Patel-Schneider
Bell Labs Research
Murray Hill, NJ, USA
pfps@lucent.com

# The RDF Data Model

- **Statements are <subject, predicate, object> triples:**

  `<Ian,hasColleague,Uli>`

- **Can be represented as a graph:**



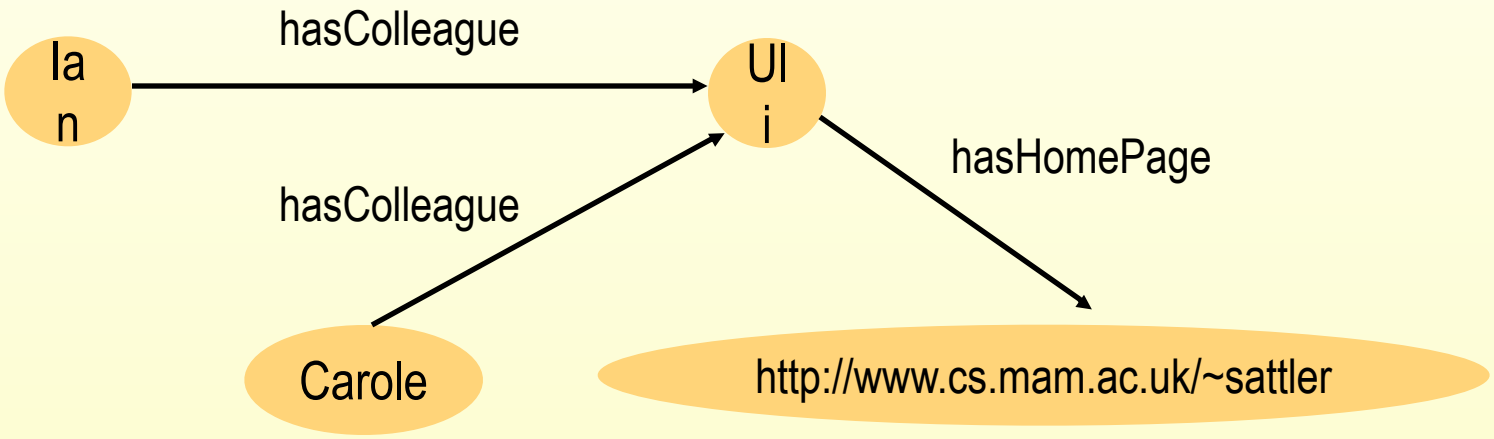

  - Statements describe properties of resources
  - A resource is any object that can be pointed to by a URI:
    - a document, a picture, a paragraph on the Web;
    - http://www.cs.man.ac.uk/index.html
    - a book in the library, a real person (?)
    - isbn://5031-4444-3333
    - …
  - Properties themselves are also resources (URIs)

# URIs

- URI = Uniform Resource Identifier
- "The generic set of all names/addresses that are short strings that refer to resources"
- URLs (Uniform Resource Locators) are a particular type of URI, used for resources that can be accessed on the WWW (e.g., web pages)
- In RDF, URIs typically look like “normal” URLs, often with fragment identifiers to point at specific parts of a document:
  - http://www.somedomain.com/some/path/to/file#fragmentID

# Linking Statements

- The subject of one statement can be the object of another
- Such collections of statements form a directed, labeled graph




- Note that the object of a triple can also be a "literal" (a string)

# RDF Syntax

- RDF has an XML syntax
- Every `Description` element describes a resource
- Every attribute or nested element inside a `Description` is a `property` of that Resource
- **We can refer to resources by using URIs**

```
<Description about="some.uri/person/ian_horrocks">
   <hasColleague resource="some.uri/person/uli_sattler"/>
</Description>
<Description about="some.uri/person/uli_sattler">
   <hasHomePage>http://www.cs.mam.ac.uk/~sattler</hasHomePage>
</Description>
<Description about="some.uri/person/carole_goble">
   <hasColleague resource="some.uri/person/uli_sattler"/>
</Description>
```

# RDF Schema (RDFS)

- **RDF gives a language for meta data annotation, and a way to write it down in XML, but it does not provide any way to structure the annotations**
- **RDF Schema augments RDF to allow you to define vocabulary terms and the relations between those terms**
  - it gives "extra meaning" to particular RDF predicates and resources
  - e.g., Class, subClassOf, domain, range
- **These terms are the RDF Schema building blocks (constructors) used to create vocabularies:**

```
<Person,type,Class>
<hasColleague,type,Property>
<hasColleague,range,Person>
<hasColleague,domain,Person>
<Professor,subClassOf,Person>
<Carole,type,Professor>
<Carole,hasColleague,Ian>
```

# Knowledge-Based Systems and Reasoning

# Shallow and Deep Reasoning

- **shallow reasoning**
  - also called experiential reasoning
  - aims at describing aspects of the world heuristically
  - short inference chains
  - possibly complex rules
- **deep reasoning**
  - also called causal reasoning
  - aims at building a model of the world that behaves like the “real thing”
  - long inference chains
  - often simple rules that describe cause and effect relationships

# Examples Shallow and Deep Reasoning

- shallow reasoning

```
IF a car has
   a good battery
   good spark plugs
   gas
   good tires
  THEN the car can move
```

- deep reasoning

```
IF the battery is good
THEN there is electricity

IF there is electricity AND
good spark plugs
THEN the spark plugs will fire

IF the spark plugs fire AND
   there is gas
THEN the engine will run

IF the engine runs AND
   there are good tires
THEN the car can move
```

# Forward Chaining

- given a set of basic facts, we try to derive a conclusion from these facts
- example: What can we conjecture about Clyde?

```
IF elephant(x) THEN mammal(x)
IF mammal(x) THEN animal(x)
elephant (Clyde)
```

modus ponens:

```
IF p THEN q
p
---------------
q
```

unification:

find compatible values for variables

# Forward Chaining Example

```
IF elephant(x) THEN mammal(x)
IF mammal(x) THEN animal(x)
elephant(Clyde)
```

**unification:**
find compatible values for variables

**modus ponens:**

```
IF p THEN q
p
---------
q
```

```
IF elephant(   x   ) THEN mammal(   x   )
```

```
elephant (Clyde)
```

# Forward Chaining Example

```
IF elephant(x) THEN mammal(x)
IF mammal(x) THEN animal(x)
elephant(Clyde)
```

**unification:**
find compatible values for variables

**modus ponens:**

```
IF p THEN q
p
---------
q
```

```
IF elephant(Clyde) THEN mammal(Clyde)
elephant (Clyde)
```

# Forward Chaining Example

```
IF elephant(x) THEN mammal(x)
IF mammal(x) THEN animal(x)
elephant(Clyde)
```

**unification:**
find compatible values for variables

**modus ponens:**

```
IF p THEN q
p
---------
q
```

```
IF mammal(   x   ) THEN animal(   x   )
```

```
IF elephant(Clyde) THEN mammal(Clyde)
```

```
elephant (Clyde)
```

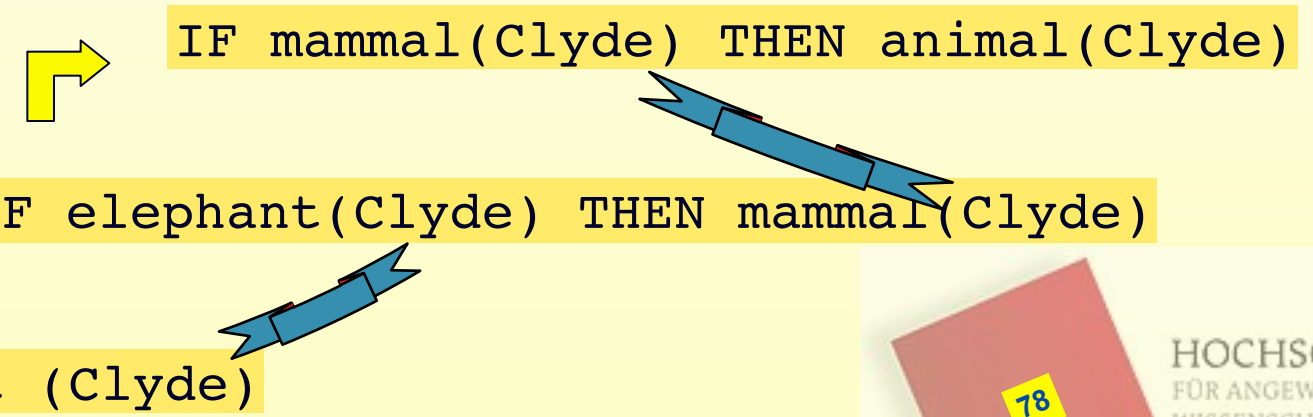
# Forward Chaining Example

```
IF elephant(x) THEN mammal(x)
IF mammal(x) THEN animal(x)
elephant(Clyde)
```

**unification:**
find compatible values for variables

**modus ponens:**

```
IF p THEN q
p
---------
q
```



# Forward Chaining Example

```
IF elephant(x) THEN mammal(x)
IF mammal(x) THEN animal(x)
elephant(Clyde)
```

**unification:**
find compatible values for variables

**modus ponens:**

```
IF p THEN q
p
---
q
```

```
animal(  x  )
```

```
IF mammal(Clyde) THEN animal(Clyde)
```

```
IF elephant(Clyde) THEN mammal(Clyde)
```

```
elephant (Clyde)
```

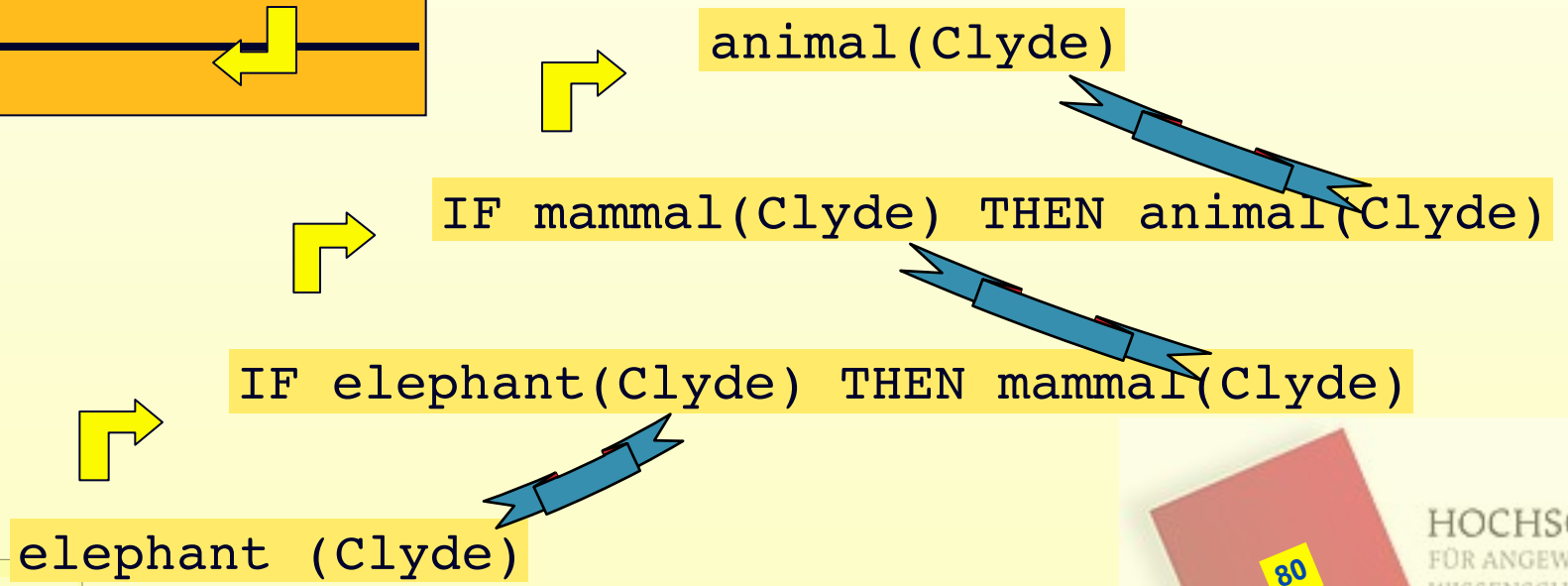
# Forward Chaining Example

```
IF elephant(x) THEN mammal(x)
IF mammal(x) THEN animal(x)
elephant(Clyde)
```

**unification:**
find compatible values for variables

**modus ponens:**

```
IF p THEN q
p
---
q
```



# Backward Chaining

- try to find supportive evidence (i.e. facts) for a hypothesis
- example: Is there evidence that Clyde is an animal?

```
IF elephant(x) THEN mammal(x)
IF mammal(x) THEN animal(x)
elephant (Clyde)
```

modus ponens:

```
IF p THEN q
p
---------------
q
```

unification:

find compatible values for variables

# Backward Chaining Example

```
IF elephant(x) THEN mammal(x)
IF mammal(x) THEN animal(x)
elephant(Clyde)
```

**unification:**
find compatible values for variables

**modus ponens:**

```
IF p THEN q
p
---------
q
```

```
animal(Clyde)
```

```
IF mammal(   x   ) THEN animal(   x   )
```

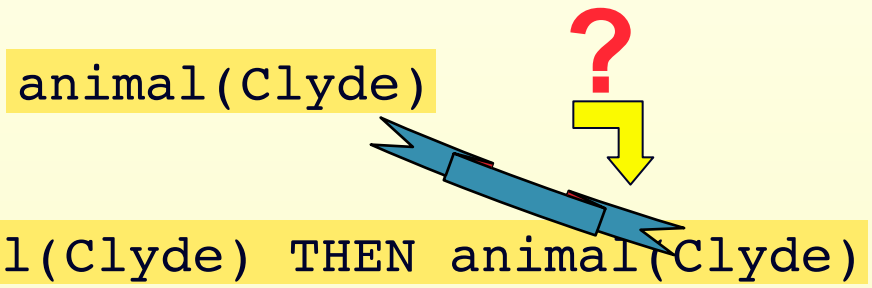
# Backward Chaining Example

```
IF elephant(x) THEN mammal(x)
IF mammal(x) THEN animal(x)
elephant(Clyde)
```

**unification:**
find compatible values for variables

**modus ponens:**

```
IF p THEN q
p
---
q
```



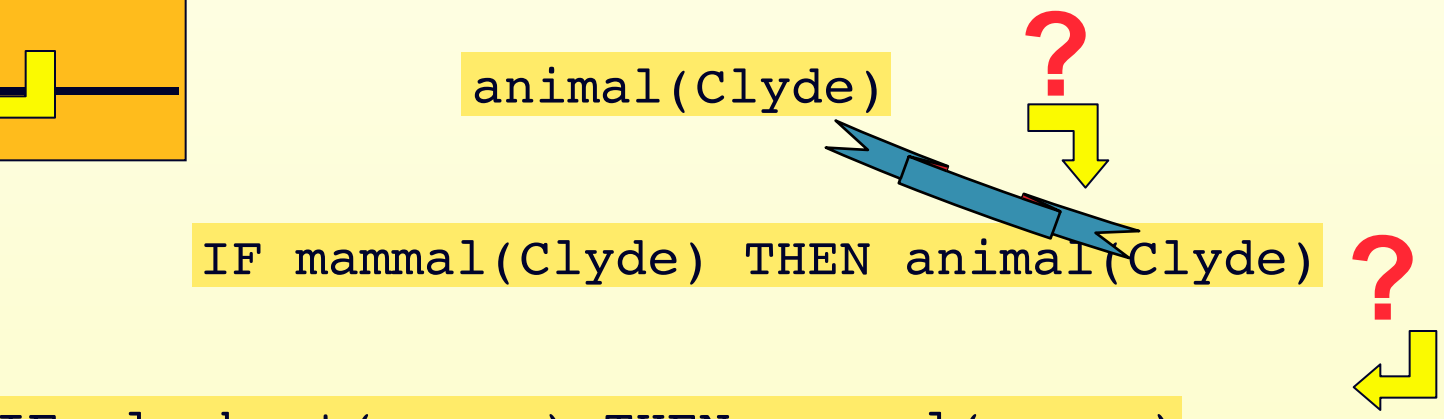
# Backward Chaining Example

```
IF elephant(x) THEN mammal(x)
IF mammal(x) THEN animal(x)
elephant(Clyde)
```

**unification:**
find compatible values for variables

**modus ponens:**

```
IF p THEN q
p
---------
q
```



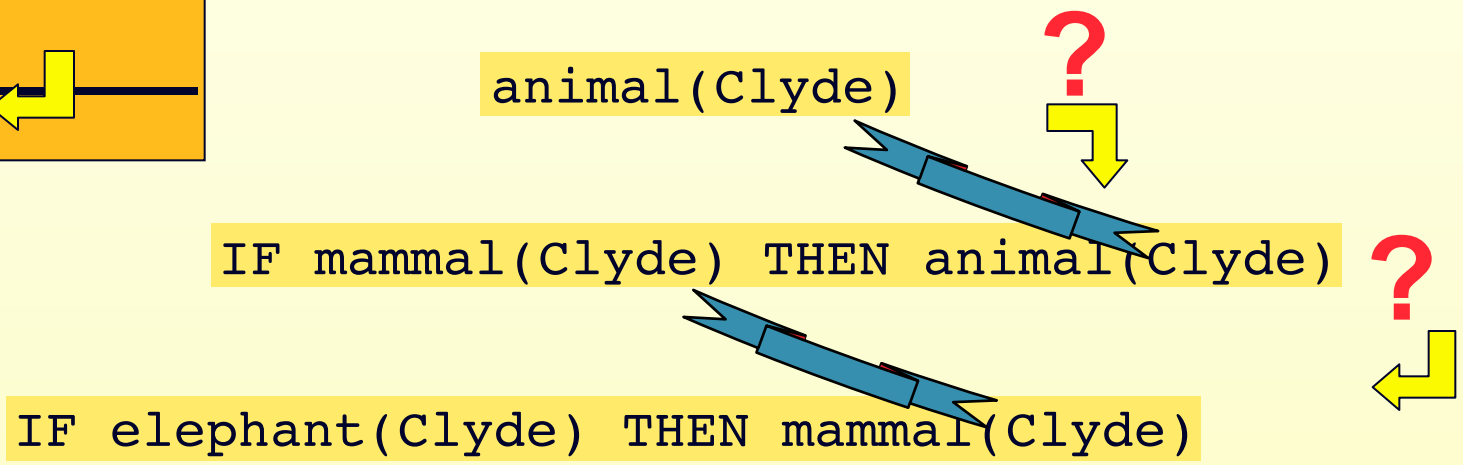
# Backward Chaining Example

```
IF elephant(x) THEN mammal(x)
IF mammal(x) THEN animal(x)
elephant(Clyde)
```

**unification:**
find compatible values for variables

**modus ponens:**

```
IF p THEN q
p
---
q
```



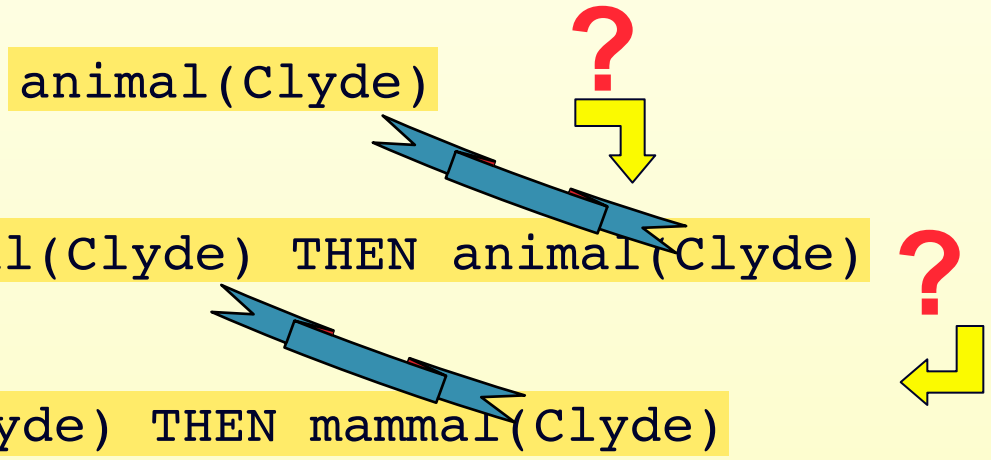
# Backward Chaining Example

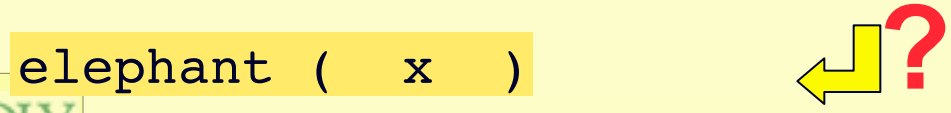
```
IF elephant(x) THEN mammal(x)
IF mammal(x) THEN animal(x)
elephant(Clyde)
```

**unification:**
find compatible values for variables

**modus ponens:**

```
IF p THEN q
p
---------
q
```



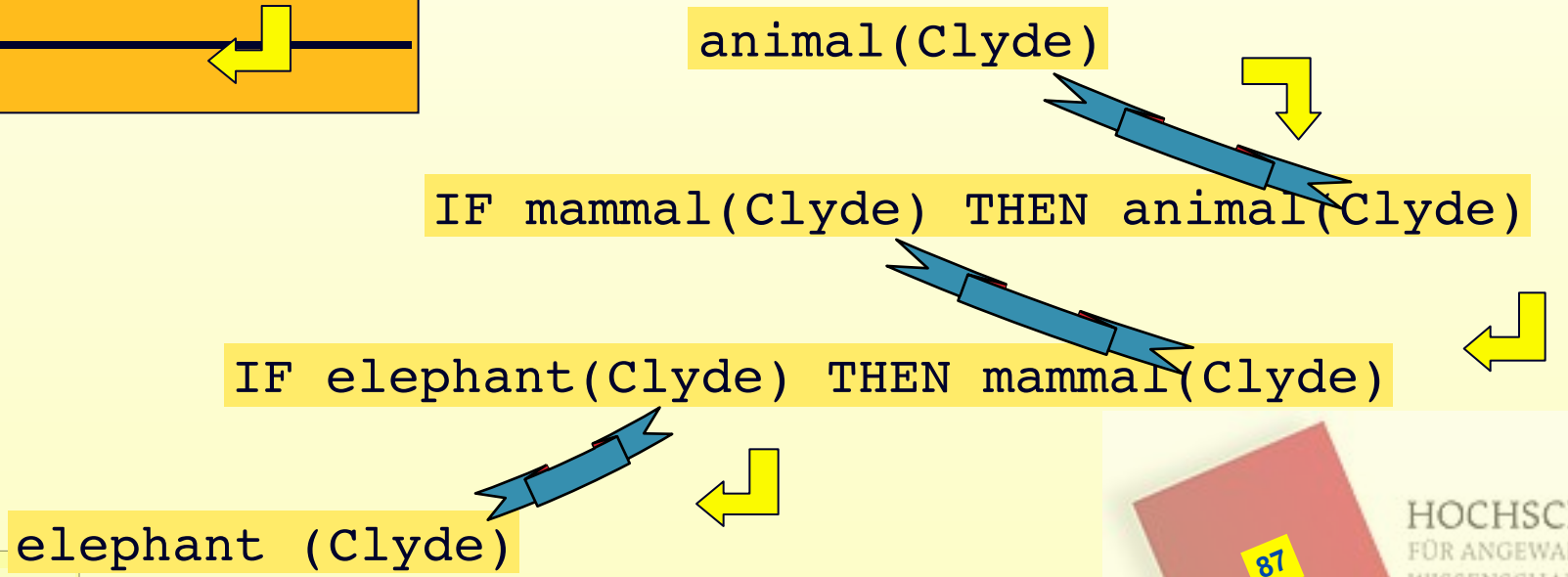
# Backward Chaining Example

```
IF elephant(x) THEN mammal(x)
IF mammal(x) THEN animal(x)
elephant(Clyde)
```

**unification:**
find compatible values for variables

**modus ponens:**

```
IF p THEN q
p
---
q
```



# Forward vs. Backward Chaining

| *Forward Chaining* | *Backward Chaining* |
|---|---|
| planning, control | diagnosis |
| data-driven | goal-driven (hypothesis) |
| bottom-up reasoning | top-down reasoning |
| find possible conclusions supported by given facts | find facts that support a given hypothesis |
| similar to breadth-first search | similar to depth-first search |
| antecedents (LHS) control evaluation | consequents (RHS) control evaluation |

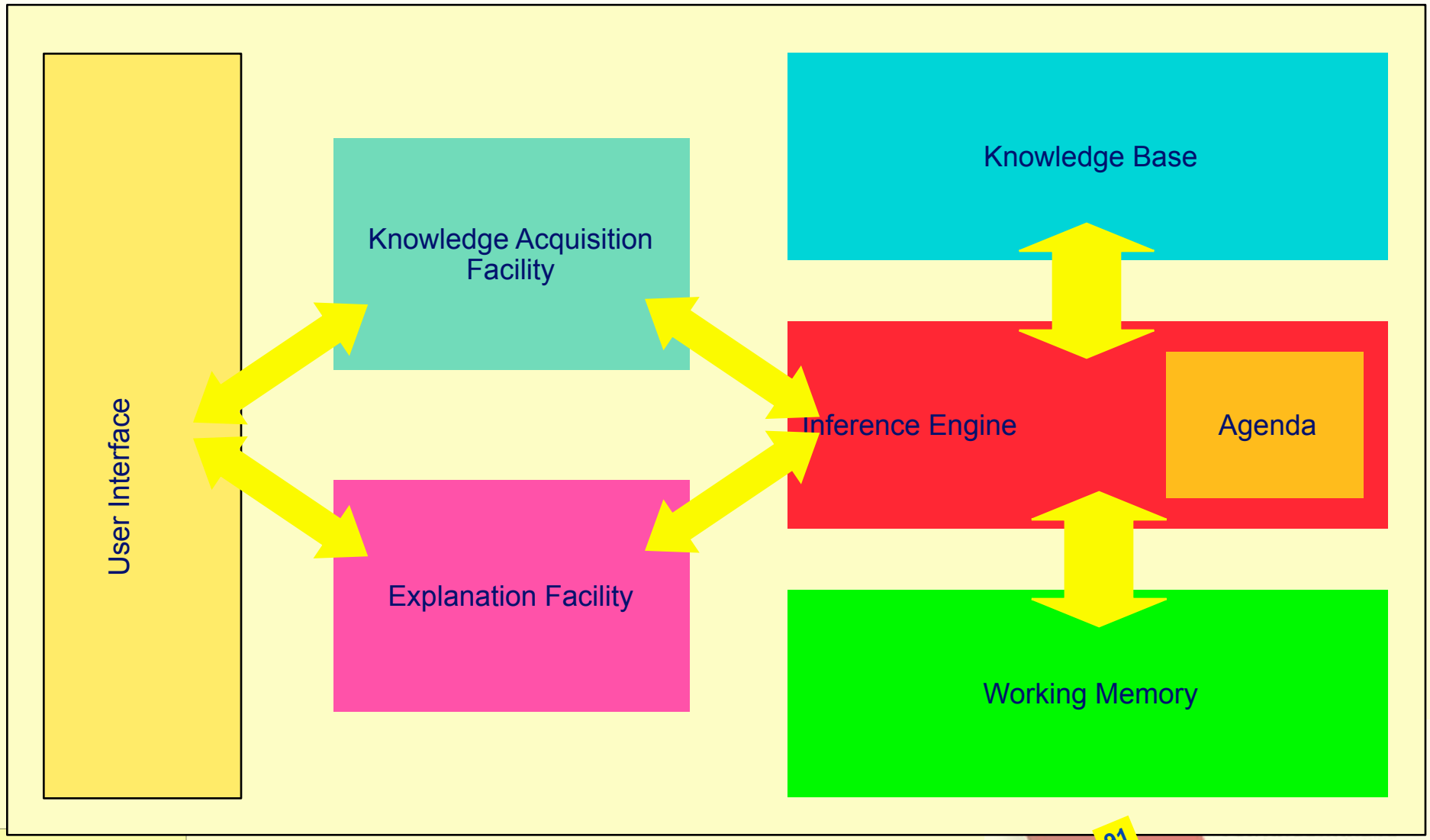
# Expert Systems (ES)

**ES Elements**
**ES Structure**
**Rule-Based ES**
**Inference Engine Cycle**
**Foundations of ES**

# ES Elements

- knowledge base
- inference engine
- working memory
- agenda
- explanation facility
- knowledge acquisition facility
- user interface

# ES Structure



# Rule-Based ES

- **knowledge is encoded as IF … THEN rules**
  - these rules can also be written as production rules
- **the inference engine determines which rule antecedents are satisfied**
  - the left-hand side must “match” a fact in the working memory
- **satisfied rules are placed on the agenda**
- **rules on the agenda can be activated (“fired”)**
  - an activated rule may generate new facts through its right-hand side
  - the activation of one rule may subsequently cause the activation of other rules

# Example Rules

## IF ... THEN Rules

```
Rule: Red_Light
   IF the light is red
      THEN stop
Rule: Green_Light
   IF the light is green
      THEN go
```

antecedent (left-hand-side)

consequent (right-hand-side)

## Production Rules

antecedent (left-hand-side)

```
the light is red  ==> stop
the light is green ==> go
```

consequent (right-hand-side)

# MYCIN Sample Rule

## Human-Readable Format

IF the stain of the organism is gram negative
AND the morphology of the organism is rod
AND the aerobiocity of the organism is gram anaerobic
THEN the there is strongly suggestive evidence (0.8)
that the class of the organism is enterobacteriaceae

## MYCIN Format

```
IF  (AND (SAME CNTEXT GRAM GRAMNEG)
         (SAME CNTEXT MORPH ROD)
         (SAME CNTEXT AIR AEROBIC)
THEN (CONCLUDE CNTEXT CLASS ENTEROBACTERIACEAE
     TALLY .8)
```

[Durkin 94, p. 133]

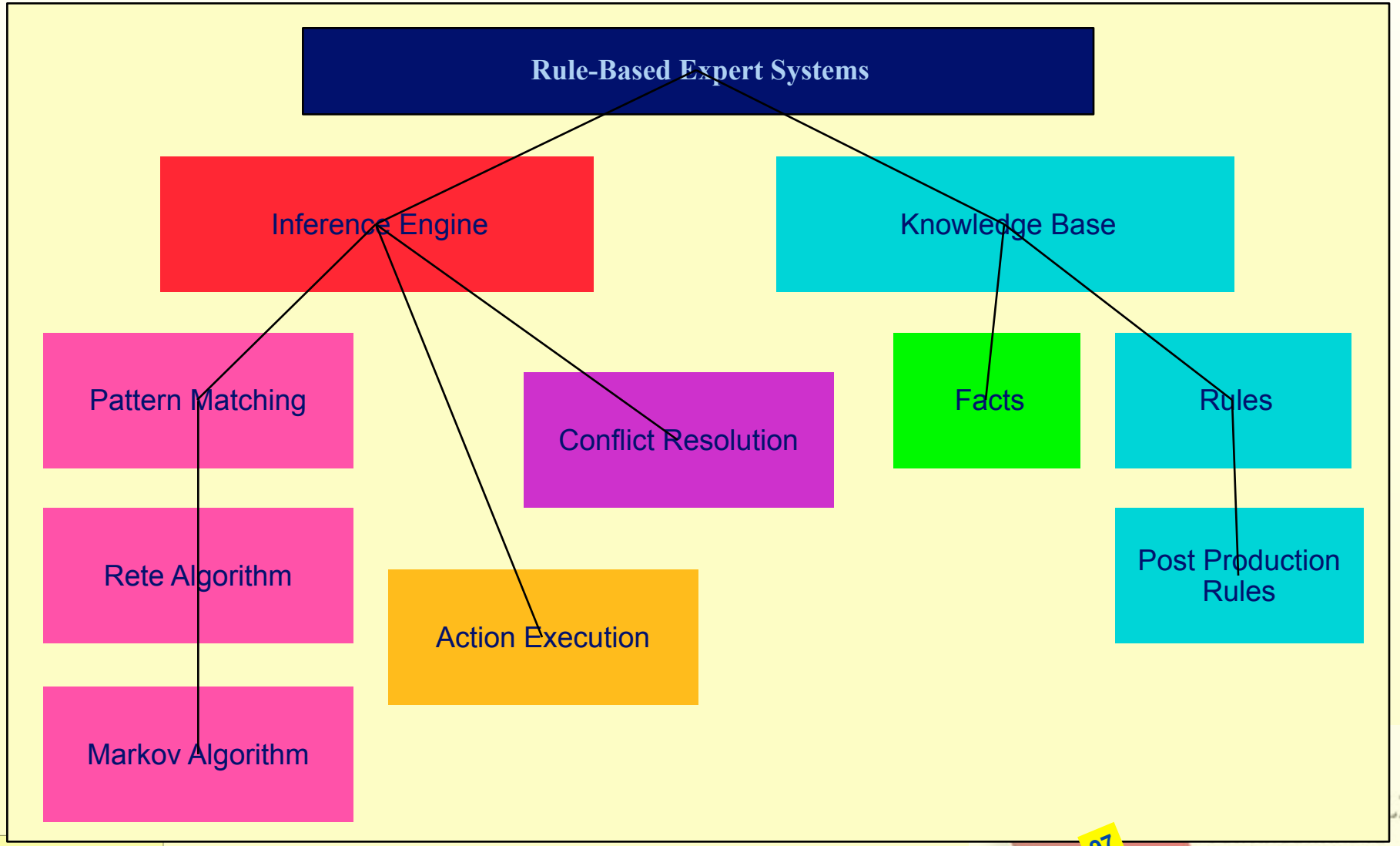
# Inference Engine Cycle

- **describes the execution of rules by the inference engine**
  - conflict resolution
    - select the rule with the highest priority from the agenda
  - execution
    - perform the actions on the consequent of the selected rule
    - remove the rule from the agenda
  - match
    - update the agenda
      - add rules whose antecedents are satisfied to the agenda
      - remove rules with non-satisfied agendas
- **the cycle ends when no more rules are on the agenda, or when an explicit stop command is encountered**

# Forward and Backward Chaining

- **different methods of rule activation**
  - forward chaining (data-driven)
    - reasoning from facts to the conclusion
    - as soon as facts are available, they are used to match antecedents of rules
    - a rule can be activated if all parts of the antecedent are satisfied
    - often used for real-time expert systems in monitoring and control
    - examples: CLIPS, OPS5
  - backward chaining (query-driven)
    - starting from a hypothesis (query), supporting rules and facts are sought until all parts of the antecedent of the hypothesis are satisfied
    - often used in diagnostic and consultation systems
    - examples: EMYCIN

# Foundations of Expert Systems



# Post Production Systems

- **production rules were used by the logician Emil L. Post in the early 40s in symbolic logic**
- **Post's theoretical result**
  - any system in mathematics or logic can be written as a production system
- **basic principle of production rules**
  - a set of rules governs the conversion of a set of strings into another set of strings
    - these rules are also known as rewrite rules
    - simple syntactic string manipulation
    - no understanding or interpretation is required
    - also used to define grammars of languages
      - e.g. BNF grammars of programming languages

# Emil Post

- **20th century mathematician**
- **worked in logic, formal languages**
  - truth tables
  - completeness proof of the propositional calculus as presented in Principia Mathematica
  - recursion theory
    - mathematical model of computation similar to the Turing machine
- **not related to Emily Post ;-)**

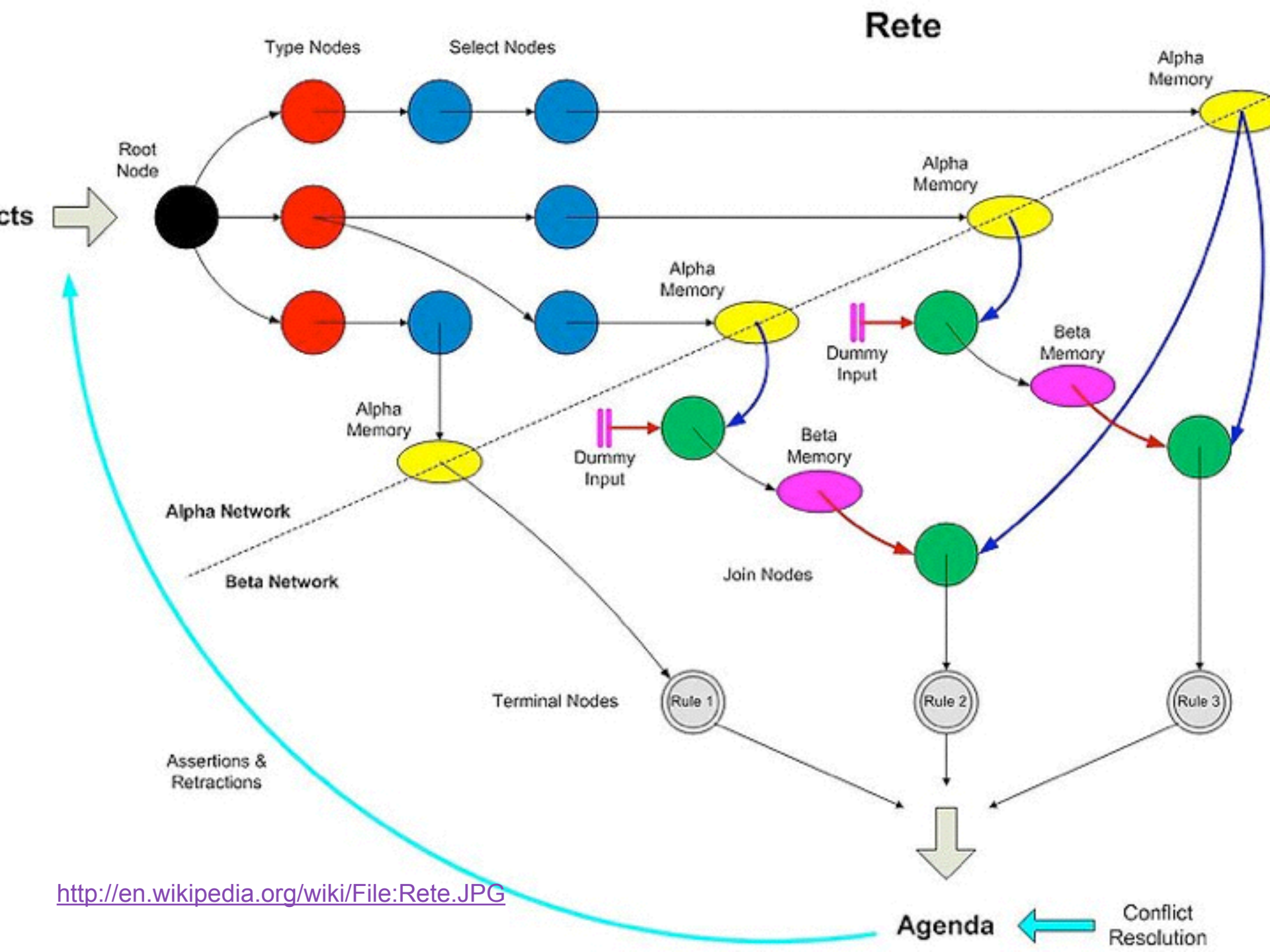
http://en.wikipedia.org/wiki/Emil_Post

# Markov Algorithms

- **in the 1950s, A. A. Markov introduced priorities as a control structure for production systems**
  - rules with higher priorities are applied first
  - allows more efficient execution of production systems
  - but still not efficient enough for expert systems with large sets of rules
  - he is the son of Andrey Markov, who developed Markov chains

# Rete Algorithm

- **developed by Charles L. Forgy in the late 70s for CMU's OPS (Official Production System) shell**
  - stores information about the antecedents in a network
  - in every cycle, it only checks for changes in the networks
  - this greatly improves efficiency




http://en.wikipedia.org/wiki/File:Rete.JPG

# Important Concepts and Terms

- agent
- automated reasoning
- belief
- category
- change
- composite object
- domain
- event
- hierarchy
- inference
- knowledge engineering
- knowledge representation
- logic
- measure
- mental object
- object
- ontology
- physical object
- predicate logic
- process
- propositional logic
- rule
- space
- substance
- time
- vocabulary

# Chapter Summary

- **knowledge representation is a fundamental aspect of reasoning systems**
- **knowledge representation is often done in stages**
  - domain selection
  - vocabulary definition
  - encoding of background knowledge
  - specification of the problem
  - formulation of queries
- **ontologies are used for capturing the terminology of a domain**
- **knowledge engineers act as intermediaries between users, domain experts and programmers**