Declarative Knowledge Representation (Logical Approach)

Summer School
in
Computational Linguistics and Represented Knowledge
Sozopol, Bulgaria
August 25 - September 8, 2000

Kiril Iv. Simov & Atanas K. Kiryakov

August 25, 2000

TOPICS

- Knowledge. Reasoning. Knowledge Representation. Knowledge Representation Systems.
- Kl-One Family. Classification. Deduction. Non-standard Semantics. Prototypes.
- XML Standards for Ontology Exchange.
- Knowledge Interchange Format.
- Multi-agent systems. Formal pragmatics of communication.

Questions we try to answer:

- What is knowledge?
- What is reasoning?
- What is the connection between knowledge and reasoning?
- What kinds of knowledge and reasoning there are?
- What is knowledge representation?
- What does a knowledge representation system look like?

Knowledge: Knowledge is such a state of an agent which allows the agent to act in a real or imaginary world. We call this state *a knowledge state of the agent.*

- Knowledge correspond in some way to the world
- Knowledge constitutes a model of the world
- Thinking rather acting

Reasoning: Reasoning is an internally driven change in the knowledge state of an agent.

- Knowledge is not perfect.
- One knowledge — one problem
- Modularization of knowledge
- Different reasoning in different modules

Knowledge Representation: Knowledge representation is the study of nature of knowledge and reasoning with the goal to simulate them on a computer.

Intelligent agents such that they:

- solve complex[1] tasks in some domain according to some goals and situations;
- acquire new knowledge on the base of their experience;
- are flexible to changing environments and goals;
- act appropriately under perceptual limitation and finite resources;
- use their knowledge to predict the world changes caused by some acting before they actually do it.

Human beings are using their knowledge to achieve some goals:

- **Deduction.** A student is proving a theorem in geometry.
- **Induction.** A scientist is building a theory about a set of observations.
- **Abduction.** A doctor is explaining a new symptom to another doctor.
- **Commonsense reasoning.** I am driving on the highway with 140 km/h when the restriction is 120 km/h.

[1] Criteria for complexity could be defined by comparison with human being.

- **Learning.** A child becomes skilled in using natural language.
- **Planning.** A thief is going to rob a bank.
- **Instructions carrying out.** A computer scientist is installing a new printer following a list of instructions.
- **Communication.** Two friends are talking to each other.

Classification of knowledge:

- Declarative vs. Procedural
- Certain vs Uncertain
- Explicit vs Implicit
- Intensional vs Extensional
- Symbolic vs Non-symbolic

Church-Turing thesis:Any symbol manipulation can be carried out on a Turing machine.

Symbol Knowledge Representation Hypotheses: Relevant knowledge can be represented in a *symbolic* way and relevant reasoning can be carried out over this representation as symbol manipulation.

A knowledge representation and reasoning system (KRRS) comprises the following elements:

- **Formal Language.** Relevant knowledge is represented as a structured set of expressions in a (formal) language, called **knowledge base**. The knowledge base could be modularised in order different kinds of knowledge to be represented in different modules and to support different kinds of reasoning.
- **Reasoning Services.** Reasoning is done through rules for manipulating the formal language expressions in the system, called **inference methods**. Inferences define the ways in which knowledge is used. In a modularised knowledge base different inferences could be applied to the different modules.
- **Maintenance Services.** Mechanisms for adding, deleting and updating of the knowledge. Also, tools for structuring of the knowledge base. These are in strong connection with the reasoning services.

A KRRS is called **balanced** if the following conditions hold:

- The modules are connected in precise way.
- There is a common theory of knowledge representation. This theory explains which kinds of knowledge should be represented in which modules.
- There is a common semantics for all modules. This common semantics defines the meaning of the expressions in the different modules and the meaning of the connections between the modules.
- The inferences have to respect the syntax and the semantics.

If in KRRS the different modules are using different languages such a system is called **hybrid**.

Requirements which an inference method ideally should meet:

- **Syntax and semantics awareness.** An inference method has to respect the syntax and the semantics of the formal language.
- **Soundness.** The results of an inference method have to be right.
- **Completeness.** An inference method has to be able to explicate all correct implicit knowledge.
- **Decidability.** There exists an algorithm that implements the rules of the inference method preserving all characteristics of the method like completeness and soundness.
- **Complexity and Tractability.** If an inference method is decidable, then the question is how much resources are necessary to do the inference. Usually, two measures are used: **worst case complexity** - the most difficult inferences; **average case** - the usual inferences.

  An inference method is **tractable** if a polynomial algorithm exists for the worst case.

An KRRS can be characterised on several levels of abstractions:

- **Implementational level.** Here the system is described in terms of memory units (cells, pointers) and procedures.
- **Logical level.** Here the system is described in logical terms like propositions, predicates, functions, logical operators and inference rules.
- **Epistemological level.** Concept types, structuring relations.
- **Ontological level.** Categories of predicates classified on the basis of formal ontology primitives.
- **Conceptual level.** Conceptual relations, primitive objects and actions.
- **Linguistic level.** Linguistic terms.

Kl-One Family

Kl-One presents a theory of concept structure description.

- **Generic Concept.** A description of a class of instances.
  - **Primitive Concept.** A description of necessary conditions for the instances of the concept.

    Primitive concepts represent the basic like "natural kinds".

    One instance could be regarded as belonging to a primitive concept only this is stated explicitly.
  - **Defined Concepts.** A description of necessary and sufficient conditions for the instances of the concept.

    One instance belongs to a defined concept if it satisfies the conditions of the definition of the concept.
- **Individual Concept.** A (partial) description of an individual.

  An individual concept is regarded as an instance of a generic concept if (1) it is stated explicitly; or (2) it satisfies the conditions of a definition of a defined concept.

Concept Structure

A concept is described on two levels:

- **Locally.** The local structure of a concept is expressed in
  - **Roles.** Generalized and instantiated attributes.
    - **Value Restriction.** Description of a potential instance.
    - **Number Restriction.** Maximum and minimum numbers of instances.
  - **Structural Description.** Description of interrelations between local elements of an instance of the concept.
    - **Role Value Map.** Equality of sets of instances.
    - **Parametric Concepts.** General relations over a set of instances.
- **Globally.** Subsumption relation on concepts.
  - **Superconcepts** and **Subconcepts.**
  - **Inheritance.** Moving of information from Superconcepts to subconcepts.
  - **Taxonomy.** Connection to the immediate superconcepts and subconcepts is calculated.

Reasoning in Kl-One

- **Classification.** Finding the right place of a new concept in a taxonomy.
- **Recognizer.** Finding the most specific concepts in a taxonomy which describe a partially described individual concept. This inference is called sometimes *realization*.
- **KB-satisfiability**.
- **Concept Satisfiability**.
- **Concept Subsumption**.
- **Concept Equivalence**.
- **Taxonomy construction**.
- **Deduction**.
- **Instance Checking**.
- **Retrieval**.

$\mathcal{ALCNR}$ Syntax

Signature:

CN is a set of concept names,
RN is a set of role names, and
IN is a set of individual names.

Concept Terms:

C,$D \longrightarrow CN$			(concept name)
	$\top$		(top)
	$\bot$		(bottom)
	$(C \sqcap D)$		(conjunction)
	$(C \sqcup D)$		(disjunction)
	$\neg C$		(complement)
	$\forall R.C$		(universal quantification)
	$\exists R.C$		(existential quantification)
	$(\geq nR)$	$(\leq nR)$	(number restrictions)

Role Terms:

$$R \longrightarrow P_1 \sqcap \ldots \sqcap P_k \qquad \text{(role conjunction)}$$

where $k \geq 1$.

$\mathcal{ALCNR}$ Semantics

Interpretation: $\mathcal{I} = < \mathcal{D}, [\![.]\!] >$ is an interpretation if and only if

$\mathcal{D}$ is a non-empty set of objects (domain),

$[\![.]\!]$ is a total function

from CN to the powerset of $\mathcal{D}$,

from RN to the powerset of $\mathcal{D} \times \mathcal{D}$, and

from IN to $\mathcal{D}$. (UNA)

Each interpretation satisfies the following equalities:

$$
\begin{aligned}
[\![\top]\!] &= \mathcal{D} \\
[\![\bot]\!] &= \emptyset \\
[\![C \sqcap D]\!] &= [\![C]\!] \cap [\![D]\!] \\
[\![C \sqcup D]\!] &= [\![C]\!] \cup [\![D]\!] \\
[\![\neg C]\!] &= \mathcal{D} \setminus [\![C]\!] \\
[\![\forall R.C]\!] &= \{d_1 \in \mathcal{D} \mid \forall d_2 : (d_1, d_2) \in [\![R]\!] \rightarrow d_2 \in [\![C]\!]\} \\
[\![\exists R.C]\!] &= \{d_1 \in \mathcal{D} \mid \exists d_2 : (d_1, d_2) \in [\![R]\!] \wedge d_2 \in [\![C]\!]\} \\
[\![\geq nR]\!] &= \{d_1 \in \mathcal{D} \mid \|\{d_2 \mid (d_1, d_2) \in [\![R]\!]\}\| \geq n\} \\
[\![\leq nR]\!] &= \{d_1 \in \mathcal{D} \mid \|\{d_2 \mid (d_1, d_2) \in [\![R]\!]\}\| \leq n\}
\end{aligned}
$$

$$
[\![P_1 \sqcap \ldots \sqcap P_k]\!] = [\![P_1]\!] \cap \ldots \cap [\![P_k]\!]
$$

$\mathcal{ALCNR}$ Statements

Terminological Statement:

CN is a concept name and C is a concept term.

$CN \doteq C$ (*concept definition*)

$CN \dot{\leq} C$ (*primitive concept definition*)

Assertional Statement:

C is a concept term, R is a role term, and a, b are individual names:

$$C(a),\ R(a,b)$$

Terminological Statement Satisfiability:

$\mathcal{I} = < \mathcal{D}, [\![.]\!] >$ *satisfies*

$CN \doteq C$ if and only if $[\![CN]\!] = [\![C]\!]$, and

$CN \dot{\leq} C$ if and only if $[\![CN]\!] \subseteq [\![C]\!]$.

Assertional Statement Satisfiability:

$\mathcal{I} = < \mathcal{D}, [\![.]\!] >$ *satisfies*

$C(a)$ if and only if $[\![a]\!] \in [\![C]\!]$,

$R(a,b)$ if and only if $([\![a]\!], [\![b]\!]) \in [\![R]\!]$.

$\mathcal{ALCNR}$ Knowledge Base

Knowledge Base:
$\Omega =< \mathcal{T}_{\mathcal{BOX}}, \mathcal{A}_{\mathcal{BOX}} >$
where

$\mathcal{T}_{\mathcal{BOX}}$ is a set of terminological statements, called TBox, and

$\mathcal{A}_{\mathcal{BOX}}$ is a set of assertional statements, called ABox.

Any concept name CN may appear at most once as left hand side of a terminological statement.

Knowledge Base Model:
An interpretation $\mathcal{I} = < \mathcal{D}, [\![.]\!] >$ is a *model* for a knowledge base $\Omega =< \mathcal{T}_{\mathcal{BOX}}, \mathcal{A}_{\mathcal{BOX}} >$ if and only if the interpretation satisfies all terminological statements in $\mathcal{T}_{\mathcal{BOX}}$ and all assertional statements in $\mathcal{A}_{\mathcal{BOX}}$.

Example

$\texttt{male} \dot{\leq} \neg\texttt{female}$

Female and male are disjoint. Female and male are primitive concepts.

$\texttt{woman} \doteq \texttt{human} \sqcap \texttt{female}$

Every woman is a human and a female. Human is primitive concept.

$\texttt{man} \doteq \texttt{human} \sqcap \texttt{male}$

Every man is a human and a male. Man and woman are disjoint because male and female are disjoint.

$\texttt{mother} \doteq \texttt{woman} \sqcap \exists \textit{has-child}.\texttt{human}$

Every mother is a woman with at least one child who is a human.

$\texttt{father} \doteq \texttt{man} \sqcap \exists \textit{has-child}.\texttt{human}$

Every father is a man with at least one child who is a human.

$\texttt{parent} \doteq \texttt{father} \sqcup \texttt{mother}$

Every parent is either a father or a mother. No parent could be a father and a mother.

$\texttt{grandparent} \doteq (\texttt{father} \sqcup \texttt{mother}) \sqcap \exists \textit{has-child}.\texttt{parent}$

Every grandparent is a parent with at least one child that is a parent.

$\texttt{grandmother} \doteq \texttt{woman} \sqcap \texttt{grandparent}$

Every grandmother is a mother with at least one child that is a parent.

$\texttt{grandfather} \doteq \texttt{grandparent} \sqcap \neg\texttt{grandmother}$

Every grandfather is a father with at least one child that is a parent.

$\texttt{mother-without-daughter} \doteq \texttt{mother} \sqcap \forall \textit{has-child}.\texttt{male}$

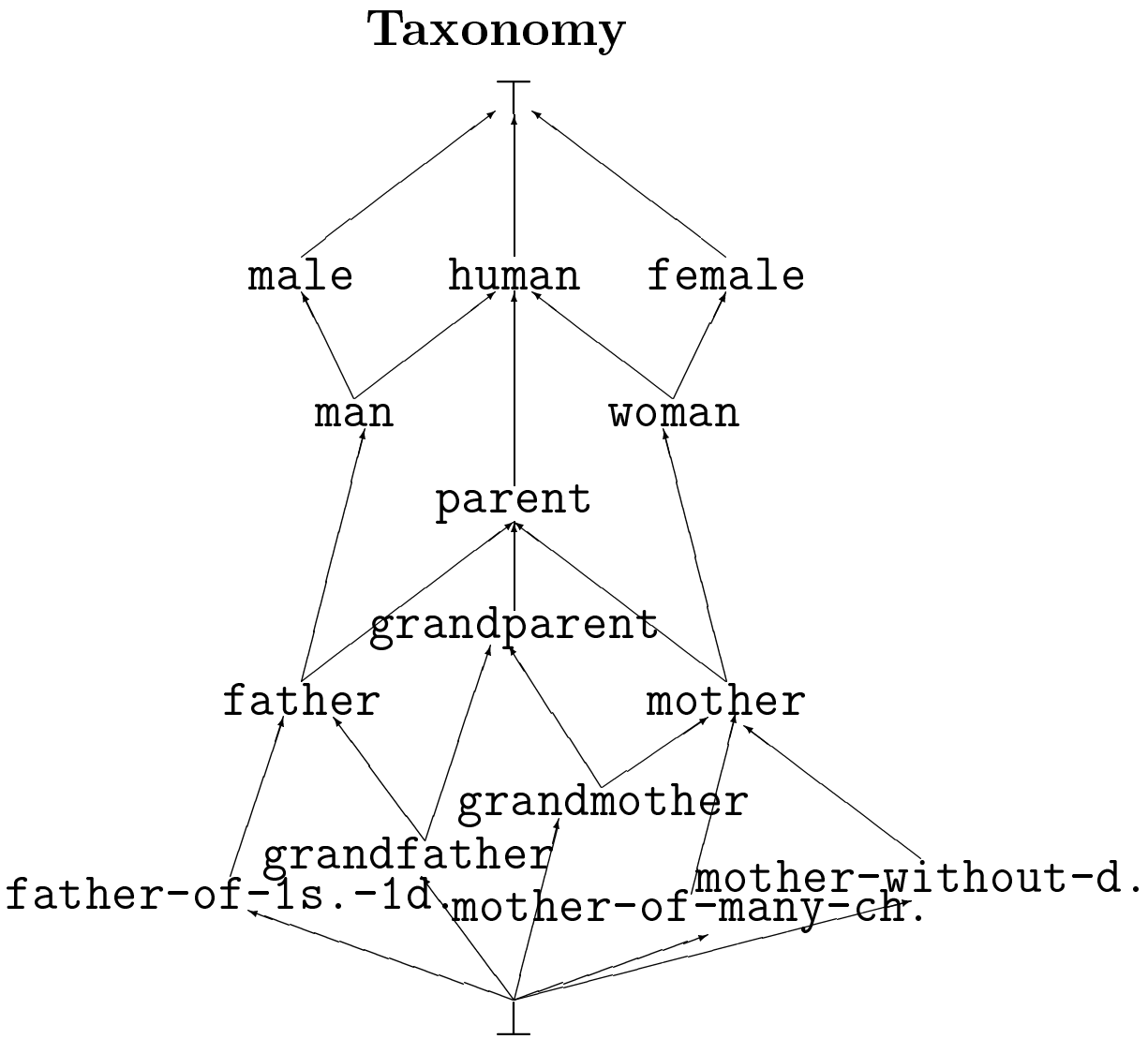
Every mother whose children are male is a mother without daughter.

$\texttt{mother-of-many-children} \doteq \texttt{mother} \sqcap (\geq 3\ \textit{has-child})$

Every mother with more than 2 children is a mother of many children.

$\texttt{father-of-a-son-and-a-daughter} \doteq \texttt{father} \sqcap (\leq 2\ \textit{has-child}) \sqcap \exists \textit{has-child}.\texttt{male} \sqcap \exists \textit{has-child}.\texttt{female}$

Every father with no more than 2 children and at least one male child and at least one female child is a father of a son and a daughter.



Deduction in $\mathcal{ALCNR}$

We are interested in two problems:

- Concept subsumption.
- Concept satisfiability.

Reduction of the problems:

C subsumes D if and only if $\neg C \sqcap D$ is unsatisfiable.

C is satisfiable if and only if $\bot$ doesn't subsume C.

Example:

$$\texttt{mother} \sqcap (\geq 3\ \textit{has-child}) \sqsubseteq \texttt{mother} \sqcap (\geq 2\ \textit{has-child})$$

$$\Downarrow$$

$$\texttt{mother} \sqcap (\geq 3\ \textit{has-child}) \sqcap \neg(\texttt{mother} \sqcap (\geq 2\ \textit{has-child}))$$

$$\Downarrow$$

$$\texttt{mother} \sqcap (\geq 3\ \textit{has-child}) \sqcap (\leq 1\ \textit{has-child})$$

The reduction can be done in linear time.

Simple concepts in $\mathcal{ALCNR}$

The algorithm is working with concept definitions that are:

- **Acyclic.** No concept name appears in its own definition. Directly or indirectly.

  No concepts like:

  $\texttt{man} \doteq \texttt{human} \sqcap \textit{father}.\texttt{man}$

- **Defined.** The TBox contains only terminological statements of the kind: $CN \doteq C$.

  For primitive concepts:

  $\texttt{male} \doteq \neg\texttt{female} \sqcap \texttt{male*}$

- **Unfolded.** Only primitive concepts participate in the right side of a concept definition.

  $\texttt{father} \doteq \texttt{man} \sqcap \exists \textit{has-child}.\texttt{human}$

  $\Downarrow$

  $\texttt{father} \doteq \texttt{human} \sqcap \texttt{male} \sqcap \exists \textit{has-child}.\texttt{human}$

- **Negative Normal Form.** Negation is only in front of concept names.

Negative Normal Form

$$\neg\top \longrightarrow \bot$$

$$\neg\bot \longrightarrow \top$$

$$\neg(C \sqcap D) \longrightarrow \neg C \sqcup \neg D$$

$$\neg(C \sqcup D) \longrightarrow \neg C \sqcap \neg D$$

$$\neg\neg C \longrightarrow C$$

$$\neg(\forall R.C) \longrightarrow \exists R.\neg C$$

$$\neg(\exists R.C) \longrightarrow \forall R.\neg C$$

$$\neg(\leq nR) \longrightarrow (\geq n+1R)$$

$$\neg(\geq nR) \longrightarrow \begin{cases} \bot & \text{if } n = 0 \\ (\leq n-1R) & \text{if } n > 0 \end{cases}$$

$$\neg((\texttt{man} \sqcup \texttt{woman}) \sqcap \exists \textit{has-child}.\texttt{human})$$

$$\Downarrow$$

$$\neg(\texttt{man} \sqcup \texttt{woman}) \sqcup \neg\exists \textit{has-child}.\texttt{human}$$

$$\Downarrow$$

$$(\neg\texttt{man} \sqcap \neg\texttt{woman}) \sqcup \forall \textit{has-child}.(\neg\texttt{human}).$$

Constraint Systems

A countable set of variables is assumed: x, y and z

Two kinds of constraints: $x : C$, xPy.

A **constraint system** S is a finite, nonempty set of constraints.

α is $\mathcal{I}$-assignment if it is function from variables in the object of the domain of $\mathcal{I}$

An $\mathcal{I}$-assignment α satisfies
$x : C$ if and only if $\alpha(x) \in [\![C]\!]$, and
xPy if and only if $(\alpha(x), \alpha(y)) \in [\![P]\!]$

A constraint c is satisfiable if $\exists\ \mathcal{I}$ and $\mathcal{I}$-assignment such that $\mathcal{I}$-assignment satisfies c

A constraint system S is satisfied by an $\mathcal{I}$-assignment if and only if it satisfies all constraints in S

S is satisfiable if and only if $\exists\ \mathcal{I}$ and $\mathcal{I}$-assignment such that $\mathcal{I}$-assignment satisfies S

Technical Notions

For role $R = P_1 \sqcap \ldots \sqcap P_k$,
xRy **holds in** S if and only if
$\{xP_1y, \ldots, xP_ky\} \subseteq S$.

For R, S and x

$$n_{R,S}(x) = \|\{y \mid xRy \text{ holds in } S\}\|$$

returns the number of the variables y such that xRy holds in S

$[y/z]S$ is the constraint system obtained from S by replacing each occurrence of y with z

This replacement is **safe in** S if and only if
for each variable x and for each role R such that
$x : (\geq nR) \in S$ and
xRy and xRz hold in S then
$n_{R,S}(x) > n$.

Rewriting Rules

1. $S \longrightarrow_{\sqcap} \{x : C1, x : C_2\} \cup S$

   if $x : C_1 \sqcap C_2$ is in S, and $x : C1$ and $x : C_2$ are not both in S

2. $S \longrightarrow_{\sqcup} \{x : D\} \cup S$

   if $x : C_1 \sqcup C_2$ is in S, neither $x : C1$ nor $x : C_2$ is in S, and $D = C_1$ or $D = C_2$

3. $S \longrightarrow_{\exists} \{xP_1y, \ldots, xP_ky, y : C\} \cup S$

   f $x : \exists R.C$ is in S, $R = P_1 \sqcap \ldots \sqcap P_k$, there is no z such that xRz holds in S and $z : C$ is in S, and y is a new variable

4. $S \longrightarrow_{\forall} \{y : C\} \cup S$

   if $x : \forall R.C$ is in S, xRy holds in S and $y : C$ is not in S

5. $S \longrightarrow_{\geq} \{xP_1y, \ldots, xP_ky\} \cup S$

   if $x : (\geq nR)$ is in S, $R = P_1 \sqcap \ldots \sqcap P_k$, $n_{R,S}(x) < n$, and y is a new variable

6. $S \longrightarrow_{\leq} [y/z]S$

   if $x : (\leq nR)$ is in S, xRy and xRz holds in S, $n_{R,S}(x) > n$, and the replacement of y by z is safe in S

Rewriting Rules

deterministic rules: $\longrightarrow_{\sqcap}$, $\longrightarrow_{\exists}$, $\longrightarrow_{\forall}$, $\longrightarrow_{\geq}$
nondeterministic rules: $\longrightarrow_{\sqcap}$, , $\longrightarrow_{\leq}$

Clash is:

- $\{x : \bot\}$
- $\{x : CN, x : \neg CN\}$
- $\{x : (\leq 0R), xP_1y, \ldots, xP_ky\}$, where $R = P_1 \sqcap \ldots \sqcap P_k$
- $\{x : (\geq mQ), x : (\leq nR)\}$ where $m > n$ and $Q \sqsubset R$.

S is complete constraint system if and only if no rule is applicable to S and S doesn't contain a clash.

All rules preserve the satisfiability.

Concept C is satisfiable if and only if the constraint system $\{x : C\}$ is satisfiable.

Example

Is `man` subsumed by `father`?

$\texttt{man} \sqsubseteq \texttt{father}$?

Reduction:

$\texttt{man} \sqcap \neg\texttt{father}$

Expansion of the definitions:

$$\texttt{human} \sqcap \texttt{male} \sqcap \neg((\texttt{human} \sqcap \texttt{male}) \sqcap \exists \textit{has-child}.\texttt{human})$$

Negative normal form:

$$\texttt{human} \sqcap \texttt{male} \sqcap ((\neg\texttt{human} \sqcup \neg\texttt{male}) \sqcup \forall \textit{has-child}.(\neg\texttt{human}))$$

Constraint system:

$\{x : ((\texttt{human} \sqcap \texttt{male}) \sqcap$
$((\neg\texttt{human} \sqcup \neg\texttt{male}) \sqcup \forall \textit{has-child}.(\neg\texttt{human}))) \}$

$\Downarrow$ by rule $\longrightarrow_{\sqcap}$

$\{x : (\texttt{human} \sqcap \texttt{male}),$
$x : ((\neg\texttt{human} \sqcup \neg\texttt{male}) \sqcup \forall \textit{has-child}.(\neg\texttt{human})) \}.$

$\Downarrow$ by rule $\longrightarrow_{\sqcap}$

$\{x:\texttt{human}, x:\texttt{male},$
$x:((\neg\texttt{human} \sqcup \neg\texttt{male}) \sqcup \forall \textit{has-child}.(\neg\texttt{human}))\ \}$

Choice point 1

$\Downarrow$ by rule $\longrightarrow_{\sqcup}$

we choose the left disjunct

$\{x:\texttt{human}, x:\texttt{male}, x:(\neg\texttt{human} \sqcup \neg\texttt{male})\ \}$

Choice point 2

$\Downarrow$ by rule $\longrightarrow_{\sqcup}$

we choose the left disjunct

$\{x:\texttt{human}, x:\texttt{male}, x:\neg\texttt{human}\ \}$

Clash

$\Downarrow$ return to point 2,

by rule $\longrightarrow_{\sqcup}$

we choose the right disjunct

$\{x:\texttt{human}, x:\texttt{male}, x:\neg\texttt{male}\ \}$

Clash.

$\Downarrow$ return to point 1,

by rule $\longrightarrow_{\sqcup}$

we choose the right disjunct

$\{x:\texttt{human}, x:\texttt{male}, x:\forall \textit{has-child}.(\neg\texttt{human})\ \}$

The system is complete. The rule $\longrightarrow_{\forall}$ is not applicable because the condition xRy to hold for some y is not met.

Taxonomy construction. Classification.

General problem:

Let $\leq$ be a partial order over the set X.
The smallest relation of precedence $\prec$ over X to be found such that its transitive and reflexive closure to be equivalent to the partial order.

For concepts:
The subsumption relation on concepts is regarded as partial order.

Technical notions:
$x \prec y$ if and only if $x \leq y$ and $\nexists z$, $z \neq x$ and $z \neq y$ such that $x \leq z \leq y$
$x \leq y$, x is *successor* of y and y is *predecessor* of x
$x \prec y$, x is *immediately successor* y and
y is *immediately predecessor* of x

$\prec_i$ is built for $X_i \subseteq X$
$\prec_{i+1}$ to be built for $X_{i+1} = X_i \cup \{c\}$ $(c \in X \setminus X_i)$
$X_i \downarrow c := \{x \in X \mid c \prec x\}$ — (top search)
$X_i \uparrow c := \{x \in X \mid x \prec c\}$ — (bottom search)

Method of "Brute Force"

In this case the "top search" is accomplished in the following way:

1. For each $x \in X_i$, $c \leq x$ is checked.
2. $X_i \downarrow c$ is the set of all $x \in X_i$ such that the check succeeds and $\forall y \in X_i (y \prec_i x)$ the check fails.

For the "bottom search" the dual schemata is used. This method uses $2\times \mid X_i \mid$ comparisons on each step and thus $n \times (n-1)$ comparisons for the whole set of n elements.

Methods of Simple Traverse

During the top search algorithm if $c \not\leq x$ then $\forall y \in X_i$ such that $y \leq x$ then $y \not\leq c$.

The nodes in the taxonomy for X_i are marked by three markers: "visited" if the subsumption check is already done for the node, "positive" if the subsumption check was successful for the node and "negative" if the subsumption check failed for the node. The mark "visited" helps us to avoid the multiple checks for a node. The marks "positive" and "negative" help us to avoid calls to the subsumption checker subs?(a, b).

```
top-search(c, x) =
   mark(x, "visited")
   Pos-Succ ← ∅
   for all y with y ≺_i x do
    if simple-top-subs?(y, c)
     then Pos-Succ ← Pos-Succ ∪{y}
    fi
   od
   if Pos-Succ is empty
    then return {x}
    else Result ← ∅
     for all y ∈ Pos-Succ do
      if not marked?(y, "visited")
       then Result ← Result ∪ top-search(c, y)
      fi
     od
     return Result
   fi
```

```
simple-top-subs?(y,c) =
   if marked?(y,"positive")
    then return true
    elsif marked?(y,"negative")
     then return false
     elsif subs?(y,c)
        then mark(y,"positive")
         return true
        else mark(y,"negative")
         return false
     fi
    fi
   fi
```

Methods of Enhanced Traverse

If the concept c is subconcept of the concept x the it is subconcept of all superconcepts of x.

First, check the superconcepts of x and if these checks succeeded then check c against x.

Let $X_i \downarrow c$ be already constructed then the immediate successors of c are successors of the elements of $X_i \downarrow c$.

Something more, an immediate successor of c is a successor of all element of $X_i \downarrow c$.

```
enhanced-top-subs?(y, c) =
   if marked?(y,"positive")
      then return true
      elsif marked?(y,"negative")
         then return false
         elsif for all z with y ≺_i z
                     enhanced-top-subs?(z, c)
                  and subs?(y, c)
            then mark(y,"positive")
              return true
            else mark(y,"negative")
              return false
         fi
      fi
   fi
```

Non-standard semantics

Concept terms:

$C, D \longrightarrow CN$	|	(concept name)
$\top$	|	(top)
$(C \sqcap D)$	|	(conjunction)
$\forall R.C$	|	(universal quantification)
$\exists R.\top$	|	(existential quantification)
$(rvm\ R\ S)$		(role value map)

Role terms:

$R \longrightarrow P_1 \sqcap \ldots \sqcap P_k$ where $k \geq 1$ |		(role conjunction)
$(restr\ R\ C)$		(role restriction)

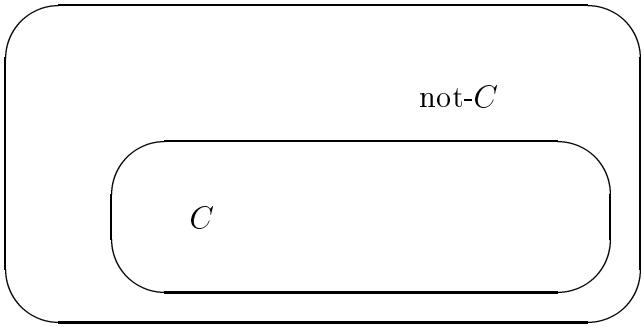
where P_i are role names.

Standard semantics for the new operators:

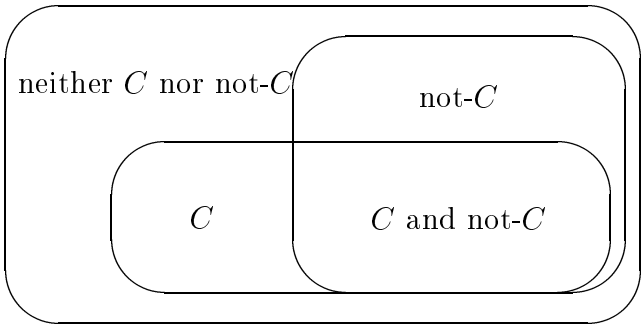
$$[\![(rvm\ R\ S)]\!] =$$
$$\{d \in \mathcal{D} \mid$$
$$\{d_r \in \mathcal{D} \mid (d, d_r) \in [\![R]\!]\} = \{d_s \in \mathcal{D} \mid (d, d_s) \in [\![S]\!]\}\}$$
$$[\![(restr\ R\ C)]\!] =$$
$$\{\langle d, e\rangle \in \mathcal{D} \times \mathcal{D} \mid \langle d, e\rangle \in [\![R]\!] \wedge e \in [\![C]\!]\}$$

Four-valued Semantics

Standard semantics:



Non-standard semantics



Interpretation

$\mathcal{I} = \langle \mathcal{D}, [\![.]\!]^t, [\![.]\!]^f \rangle$, where

$\mathcal{D}$ is a set of objects.

$[\![.]\!]^t$ - positive interpretation function

$[\![.]\!]^f$ - negative interpretation function

Interpretation

For concept terms:

$[\![\top]\!]^t = \mathcal{D}$

$[\![\top]\!]^f = \emptyset$

$[\![C \sqcap D]\!]^t = [\![C]\!]^t \cap [\![D]\!]^t$

$[\![C \sqcap D]\!]^f = [\![C]\!]^f \cup [\![D]\!]^f$

$[\![\forall R.C]\!]^t = \{d_1 \in \mathcal{D} \mid \forall d_2 : (d_1, d_2) \in [\![R]\!]^f \vee d_2 \in [\![C]\!]^t\}$

$[\![\forall R.C]\!]^f = \{d_1 \in \mathcal{D} \mid \exists d_2 : (d_1, d_2) \in [\![R]\!]^t \wedge d_2 \in [\![C]\!]^f\}$

$[\![\exists R.\top]\!]^t = \{d_1 \in \mathcal{D} \mid \exists d_2 : (d_1, d_2) \in [\![R]\!]^t\}$

$[\![\exists R.\top]\!]^f = \{d_1 \in \mathcal{D} \mid \exists d_2 : (d_1, d_2) \in [\![R]\!]^f\}$

$[\![(rvm\ R\ S)]\!]^t =$

$\{d_1 \in \mathcal{D} \mid \forall d_2 : (d_1, d_2) \in [\![R]\!]^f \vee (d_1, d_2) \in [\![S]\!]^t\}$

$[\![(rvm\ R\ S)]\!]^f =$

$\{d_1 \in \mathcal{D} \mid \exists d_2 : (d_1, d_2) \in [\![R]\!]^t \wedge (d_1, d_2) \in [\![S]\!]^f\}$

For role terms:

$[\![P_1 \sqcap \ldots \sqcap P_k]\!]^t = [\![P_1]\!]^t \cap \ldots \cap [\![P_k]\!]^t$

$[\![P_1 \sqcap \ldots \sqcap P_k]\!]^f = [\![P_1]\!]^f \cup \ldots \cup [\![P_k]\!]^f$

$[\![(restr\ R\ C)]\!]^t =$

$\{\langle d, e\rangle \in \mathcal{D} \times \mathcal{D} \mid \langle d, e\rangle \in [\![R]\!]^t \wedge e \in [\![C]\!]^t\}$

$[\![(restr\ R\ C)]\!]^f =$

$\{\langle d, e\rangle \in \mathcal{D} \times \mathcal{D} \mid \langle d, e\rangle \in [\![R]\!]^f \vee e \in [\![C]\!]^f\}$

Algorithms

Canonical form:

1. commutativity and associativity of $\sqcap$
2. $\forall R.(C \sqcap D) \longrightarrow (\forall R.C) \sqcap (\forall R.D)$
3. $(rvm\ R\ (S_1 \sqcap S_2) \longrightarrow (rvm\ R\ S_1) \sqcap (rvm\ R\ S_2)$
4. $(rvm\ R\ (restr\ S\ C)) \longrightarrow (rvm\ R\ S) \sqcap (\forall RC)$
5. $(restr\ (restr\ R\ C_1)\ C_2) \longrightarrow (restr\ R(C_1 \sqcap C_2))$
6. $((restr\ R_1\ C) \sqcap R_2) \longrightarrow (restr\ (R_1 \sqcap R_2)\ C)$

Subsumption check:

$C_1 \sqcap \ldots \sqcap C_n$ subsumes $C'_1 \sqcap \ldots \sqcap C'_m$ if and only if $\forall i$, $1 \leq i \leq n$, $\exists j$, $1 \leq j \leq m$, such that:

1. C_i is a concept name and $C'_j = C_i$,
2. $C_i = (\exists R.\top)$ and $C'_j = (\exists R'.\top)$ and R subsumes R',
3. $C_i = (\forall R.D)$ and $C'_j = (\forall R'.D')$ and D subsumes D' and R subsumes R',
4. $C_i = (rvm\ R\ S)$ and $C'_j = (rvm\ R'\ S)$ and $R' \sqsubseteq R$.

$S_1 \sqcap \ldots \sqcap S_n$ subsumes $S'_1 \sqcap \ldots \sqcap S'_m$ if and only if
$\forall S_i$, $0 \leq i \leq n$, $\exists S'_j$, $0 \leq j \leq m$ such that $S_i = S'_j$
$(restr\ (S_1 \sqcap \ldots \sqcap S_m)\ C)$ is subsumed by $(restr\ (S'_1 \sqcap \ldots \sqcap S'_n)\ C')$ if and only if
$\forall i$, $\exists j$ such that $S_j = S'_i$ and $C \sqsubseteq C'$.

Not valid:

$\phi \vee \neg\phi$

$((\phi \wedge (\phi \rightarrow \psi)) \rightarrow \psi)$

Example:

The soncept:

$\texttt{human} \sqcap \forall \textit{friend}.\texttt{doctor} \sqcap$

$\forall \texttt{(restr}\ \textit{friend}\ \texttt{doctor)}.(\exists \textit{speciality}.\top)$

is not subsumed by

$\texttt{human} \sqcap \forall \textit{friend}.(\exists \textit{speciality}.\top)$

Prototypes in Kl-One

The motivation:

- Most of the concepts are primitive.
- In common life we point to these concepts on the base of their typical features.
- The recognition has to be done on the base of prototypical information.

Tweety flies and has wings.

Is Tweety a bird?

Prototypical reasoning mechanism cooperates with the recodnizer.

Two steps of inference:

- The recognizer is using the concept hierarchy to determine the most specific concepts that describe an individual.
- The prototype reasoner compares the individual's description with the prototypical information connected with these concepts.

Similarity Model

The model comprises three components:

- a prototype representation;
- a procedure for prototype modification;
- a rule for determining the similarity between an object and a prototype.

Prototype Representation

Prototype i a collection of attribute/value pairs.

Attributes can only be roles defined in the terminology and values are concepts.

Each attribute/value pair for a given prototype is provided with two weights:

- **diagnosticity** of the attribute
- **salience** of the value

Prototype formation:

A prototype is considered an abstraction over a set of individuals.

Prototypes refer to primitive concepts in the terminology.

How to determine sufficient typical conditions for a given primitive concept?

This mechanism is based on frequencies over the instance set; diagnosticity and salience are defined in the following way:

- **attribute diagnosticity**: Given a prototype P derived from a primitive concept C, the diagnosticity of an attribute A associated to P is defined as the probability that A describes an instance X, given the fact that X is a member of C.
- **value salience**: given P as the prototype derived from C, the salience of the attribute/value pair A/V is defined as the probability that the A/V pair describes an instance X, given the fact that X is a member of C.

Relations to the terminology.

The prototype is non-contradictory to the definition of primitive concept.

Reasoning with Prototypes

Contrast model is used. Instance categorisation in this model succeeds when a given threshold on the similarity function is reached.

Similarity between an instance I and a prototype P is determined by three feature sets.

- $I \cap P$, the features common to I and P;

- $P - I$, the prototype distinctive features;
- $I - P$, the instance distinctive features.

The additive version of the contrast model is expressed by the equation:

$$SIM(P, I) = \sum i\ [af_i(I \cap P) - bf_i(P - I) - cf_i(I - P)]$$

Where i indexes the relevant attributes, f_i is assumed to multiply the number of votes (salience) for attribute i and a, b, c are parameters that determine the relative contribution of each attribute set.

The contrast model respects the following properties that influence human similarity judgements: focus hypothesis (i.e., common features have a greater weight than distinctive features), asymmetry (i.e., the subject has a larger import than the referent of the judgment), context dependency (i.e., context dependent features are more relevant).

Probability and Similarity

Human behaviour does not strictly follow the probability rules.

Knowledge Interchange Format

Why we need KIF?

- knowledge exchange
- formal description of KR languages

History:

- started as a part of KSE in 1990
- first version in 1992
- currently – draft proposed American National Standard (dpANS) NCITS.T2/98-004

The three directions in KSE:

- medium for knowledge transfer – **KIF**
- methodology for knowledge transfer – **KQML**
- ontology framework and basic ontological theories in different fields – **Ontolingua**

Essential features:

- declarative semantics
- logically comprehensive
- meta-knowledge

Objectives:

- Implementability
- Readability

Few comments:

- Self-reference and truth — especially important for NLP and intensional logics
- Concrete domains: numbers, lists and strings.

  Why are they useful? Why they are just few?

The End of the Beginning

KIF can be considered as:

- First-Order Language
- Functions & Relations are also objects in the domain
- All the Expressions are objects also

Few KIF forms:

```
(listof mary (listof tom dick harry) sally)

(defobject nil := (listof))
(defrelation null (?l) := (= ?l (listof)))
```

```
(defrelation rational (?x) :=
      (exists (?y) (and (integer ?y)
                        (integer (* ?x ?y)))))
```

Syntax

Computer-oriented formal syntax: characters, lexemes and expressions.

KIF characters are the ASCII characters classified as:

- upper case letters
- lower case letters
- digits
- alpha characters (`!` | `$` | `%` | `&` | `*` | `+` | `-` | `.` | `/` | `<` | `=` | `>` | `?` | `@` | `_` | `~`)
- special characters (`"` | `#` | `'` | `(` | `)` | `,` | `\` | `^` | `` ` ``)
- white space (`space` | `tab` | `return` | `linefeed` | `page`)
- other characters - the rest ASCII characters

A **normal** character is each one in the first four categories.

Lexemes

Five types of lexemes– special lexemes, words, character references, character strings, and character blocks.

Word is a sequence of: (1) normal characters or (2) other characters preceded by the escape character \

Words could be **variables**, **operators**, and **constants**

Variables start with ? or @.

Three types of operators in KIF

- **term operators** (say `(+ 1 2)`)
- **sentence operators** (say `(= a b)`)
- **definition operators**
  (say `(defobject Title := "KIF")`)

Five term operators:

`value | listof | quote | if | cond`.

Sentence operators:

`holds | = | /= | not | and |`
`or | => |<= | <=> | forall | exists`.

Definition operators:

`defobject | defunction | defrelation | deflogical |`
`:= | :-> | :<= | :=>`.

All other words are called **constants** which are split in four semantic categories.

Basic constants and **Non-basic constants**

Expressions

Three disjoint types – **terms**, **sentences**, and **definitions**.

Definitions and sentences are called **forms**.

A **knowledge base** is a finite set of forms.

Types of terms in KIF – **individual variables**, **constants**, **functional terms**, **list terms**, **quotations**, and **logical terms**.

```
<funterm>  ::= (<constant> <term>* [<seqvar>]) |
               (value <constant> <term>* [<seqvar>])
<listterm> ::= (listof <term>* [<seqvar>])
<quoterm>  ::= (quote <expression>)

<logterm>  ::= (if <sentence> <term> [<term>])|
               (cond (<sentence> <term>) ...
                     (<sentence> <term>))
```

Forms in KIF are defined according to the following syntax:

Sentences in KIF:

```
 <sentence> ::=
```

```
<logconst>|<equation>|<inequality>|
<relsent>|<logsent>|<quantsent>

<equation> ::= (= <term> <term>)
<inequality> ::= (/= <term> <term>)

<relsent> ::= (<relconst> <term>* [<seqvar>])|
        (holds <relconst> <term>* [<seqvar>])

<logsent> ::= (not <sentence>)|
              (and <sentence>*)|
              (or <sentence>*)|
              (=> <sentence>* <sentence>)|
              (<= <sentence> <sentence>*)|
              (<=> <sentence> <sentence>)

<quantsent> ::= (forall (<varspec>+) <sentence>) |
                (exists (<varspec>+) <sentence>)

<varspec> ::= <variable> |
              (<variable> <constant>)
```

Definitions in KIF:

```
<definition> ::= <unrestricted> | <complete> |
                 <partial>
```

```
<unrestricted> ::=
(defobject <constant> [<string>] <sentence>*) |

(deffunction <constant> [<string>] <sentence>*) |

(defrelation <constant> [<string>] <sentence>*) |

(deflogical <constant>[<string>] <sentence>*)

<complete> ::=
 (defobject <constant> [<string>] := <term>) |
 (deffunction <constant> (<indvar>* [<seqvar>])
                         [<string>] := <term>) |
 (defrelation <constant> (<indvar>* [<seqvar>])
                     [<string>] := <sentence>) |
 (deflogical <constant> [<string>] := <sentence>)

<partial> ::=
 (defobject <constant> [<string>] :->
                     <indvar> :<= <sentence>) |
 (defobject <constant> [<string>] :->
                     <indvar> :=> <sentence>) |
 (deffunction <constant> (<indvar>* [<seqvar>])
     [<string>] :-> <indvar> :<= <sentence>) |
 (deffunction <constant> (<indvar>* [<seqvar>])
     [<string>] :-> <indvar> :=> <sentence>) |
```

```
(defrelation <constant> (<indvar>* [<seqvar>])
                  [<string>] :<= <sentence>) |
(defrelation <constant> (<indvar>* [<seqvar>])
                  [<string>] :=> <sentence>) |
(deflogical <constant> [<string>]
                              :<= <sentence>)
(deflogical <constant> [<string>]
                              :=> <sentence>)
```

Which of the sentences is syntacticaly wrong:

```
[1] (defrelation rational (?x) :=
   (exists (?y integer) (integer (* ?x ?y))))

[2] (/= ?x (and ?x
                (not (exists (?y)
                             (= ?y ?x)))))
```

Semantics

Defined in terms of a **conceptualization** and an **interpretation**

Conceptualization - determine the objects, functions and relations

A **universe of discourse** is the set of all objects.

Every universe of discourse should the **basic** objects:

- All ASCII Characters
- Words (constants, operators, and viariables)
- All complex numbers
- All finite lists of objects
- $\perp$

Of course, any universe of discourse could contain whatever *non-basic* objects in addition.

How to represent functions and relations using lists?
Relations are sets of sequences

```
'< = {<1,2>, <2,3>, <1,3>, ... , <1,2,3>, ...}
```

where `<1,2>` stands for the fact that `(< 1 2)`

Functions too! What is the difference?

Interpretation
An **interpretation** is a function i that associates the constants of KIF with the elements of a conceptualization.
Let's call O the universe of discourse.

- 1. If σ is an object constant, then $i(\sigma) \in O$.
- 2. If σ is a function constant, then $i(\sigma) : O^* \longrightarrow O$.

- 3. If σ is a relation constant, then $i(\sigma) \subseteq O^*$.
- 4. If σ is a logical constant, then $i(\sigma) \in \{true, false\}$.

Some of the axioms "built-in" KIF:

- For each numerical constant σ $i(\sigma) = n$
- $i(\texttt{bottom}) = \bot$
- $i(\texttt{true}) = true$ and $i(\texttt{false}) = false$

Variable Assignment v is a function

- individual variables $\mathcal{V}$ into objects in O
- sequence variables $\mathcal{W}$ into finite sequences of objects.

Semantic Value

$s_{iv} : \mathcal{T} \longrightarrow O$ such that

- $s_{iv}(\nu) = v(\nu)$ $\quad$ ν is a variable
- $s_{iv}(\sigma) = i(\sigma)$ $\quad$ σ is a constant
- What about functional and relational constants?
  $s_{iv}((\pi\tau_1 \ldots \tau_n)) = i(\pi)[s_{iv}(\tau_1), ..., s_{iv}(\tau_n)]$
- $s_{iv}(\texttt{(listof } \tau_1 \texttt{ ... } \tau_k \texttt{ } \omega\texttt{)}) = \langle s_{iv}(\tau_1), ..., s_{iv}(\tau_k) | s_{iv}(\omega) \rangle$

- $s_{iv}(\texttt{(quote } \epsilon\texttt{)}) = \epsilon$
- $s_{iv}(\texttt{(quote (}\epsilon_1 \ \ldots \ \epsilon_n\texttt{))}) =$
  $s_{iv}(\texttt{(listof (quote } \epsilon_1\texttt{) } \ldots \ \texttt{(quote } \epsilon_n\texttt{))})$
- $s_{iv}(\texttt{(if } \phi \ \tau_1\texttt{)}) = \begin{cases} s_{iv}(\tau_1) & t_{iv}(\phi) = true \\ \bot & \text{otherwise} \end{cases}$
- $s_{iv}(\texttt{(cond (}\phi_1 \ \tau_1\texttt{) } \ldots \ \texttt{(}\phi_n \ \tau_n\texttt{))}) =$
  $\begin{cases} s_{iv}(\tau_1) & t_{iv}(\phi_1) = true \\ \ldots & \ldots \\ s_{iv}(\tau_2) & t_{iv}(\phi_n) = true \\ \bot & \text{otherwise} \end{cases}$

Truth Value

A function $t_{iv} : \mathcal{S} \longrightarrow \{true, false\}$

- $t_{iv}(\lambda) = i(\lambda)$ $\quad$ λ is a logical constant
- $t_{iv}(\texttt{(= } \tau_1 \ \tau_2\texttt{)}) = \begin{cases} true & s_{iv}(\tau_1) = s_{iv}(\tau_2) \\ false & \text{otherwise} \end{cases}$
- $t_{iv}(\texttt{(/= } \tau_1 \ \tau_2\texttt{)}) = \begin{cases} false & s_{iv}(\tau_1) = s_{iv}(\tau_2) \\ true & \text{otherwise} \end{cases}$
- $t_{iv}(\texttt{(}\rho \ \tau_1 \ \ldots \ \tau_n\texttt{)}) = \begin{cases} true & \langle s_{iv}(\tau_1), ..., s_{iv}(\tau_n)\rangle \in i(\rho) \\ false & \text{otherwise} \end{cases}$
- $t_{iv}(\texttt{(not } \phi\texttt{)}) = \begin{cases} true & t_{iv}(\phi) = false \\ false & \text{otherwise} \end{cases}$
- similar for `and`, `or`, =>, <=, and <=>

A variable assignment v' is a *version* of variable assignment v with respect to variables ν_1, ..., ν_n iff ...

Next v' is a version of v with respect to variables in the first argument.

$t_{iv}(\texttt{(exists (}\nu_1\ \ldots\ \nu_k\ \omega\texttt{)}\ \phi\texttt{)}) =$

$\begin{cases} true & \exists v'\ t_{iv'}(\phi) = true \\ false & \text{otherwise} \end{cases}$

$t_{iv}(\texttt{(forall (}\nu_1\ \ldots\ \nu_k\ \omega\texttt{)}\ \phi\texttt{)}) =$

$\begin{cases} true & \forall v'\ t_{iv'}(\phi) = true \\ false & \text{otherwise} \end{cases}$

Logical Entailment

An interpretation i **logically satisfies** a sentence ϕ iff ...

i is a *model* of ϕ

If Δ is a set of sentences, we say that Δ *logically entails* a sentence ϕ iff every model of Δ is also a model of ϕ.

Definitions

Definitions have no truth values in the sense described above.

Every definition has a corresponding set of sentences, called the *content*

Definition	Defining Axiom
(defobject σ := τ)	(= σ τ)
(defobject σ ϕ_1 ... ϕ_n)	(and ϕ_1 ... ϕ_n)

Definition	Defining Axiom
(deffunction π (ν_1 ...ν_n) := τ)	(= (π ν_1 ...ν_n) τ)
(deffunction π ϕ_1 ... ϕ_n)	(and ϕ_1 ... ϕ_n)

Definition	Defining Axiom
(defrelation ρ (ν_1 ...ν_n) := ϕ)	(<=> (ρ ν_1 ...ν_n) ϕ)
(defrelation ρ ϕ_1 ... ϕ_n)	(and ϕ_1 ... ϕ_n)

Lists

```
(defrelation list (?x) :=
     (exists (@l) (= ?x (listof @l)))

(defobject nil := (listof))
(defrelation null (?l) := (= ?l (listof)))

(deffunction subst (?x ?y ?z) :=
   (cond ((= ?y ?z) ?x)
         ((null ?z) nil)
         ((list ?z)
```

```
            (cons (subst ?x ?y (first ?z))
                  (subst ?x ?y (rest ?z))))
         (true ?z)))|
(deffunction cons (?x ?l) :=
   (if (= ?l (listof @l)) (listof ?x @l)))
```

Metaknowledge

Naming Expressions

Words are treated as primitive objects

Complex expressions are lists of sub-expressions

There are two ways to refer to expressions in KIF.

- `(believes john '(material moon stilton))`

  `(=> (believes john ?p) (believes mary ?p))`

- Complex expression like `(p a b)` can be denoted by a term of the form `(listof 'p 'a 'b)`, as well as `'(p a b)`

  ```
  (=> (believes john (listof 'material ?x ?y))
     (believes lisa (listof 'material ?x ?y)))
  ```

Another possible syntax

```
(=> (believes john ^(material ,?x ,?y))
    (believes isa ^(material ,?x ,?y)))
```

Formalizing Syntax

`(indvar (quote` ν`))`

`(seqvar (quote` ω`))`

```
(defrelation word (?x) :=
 (or (variable ?x) (operator ?x)
     (constant ?x)))
(defrelation sentence (?x) :=
     (or (logconst ?x) (relsent ?x)
         (equation ?x) (inequality ?x)
         (logsent ?x) (quantsent ?x)))
```

Changing Levels of Denotation

The term `(denotation` τ`)` denotes the object denoted by the object denoted by τ. A quotation denotes the quoted expression; the denotation of any other object is $\perp$.

The term `(name` τ`)` denotes the standard name for the object denoted by the term τ.

Weekly true

```
    (wtr '(=> (p ?x) (q ?x)))

    (=> (sentence ?p) (wtr `(=> ,?p ,?p)))

(wtr (subst
    (name ^(subst (name x) ^x ^(wtr ,x)))
```

```
                    ^x
  ^(not (wtr (subst (name x)
                    ^x ^(not (wtr ,x)))))))
```

Avoiding the paradox

```
(<=> (wtr 'p) p)
```

Conformance

"conformance levels" and "conformance profile" vs. "conformance level"

Logical Form

A knowledge base is **atomic** if and only if it contains no logical operators.

A knowledge base is **conjunctive** if and only if it contains no logical operators except for conjunction.

A knowledge base is **positive** if and only if it contains no logical operators except for conjunction and disjunction.

A knowledge base is **logical** if and only if it contains no logical operators except for conjunction, disjunction, and negation.

A knowledge base is rule-like if and only if every sentence is either atomic or an implication or reverse implication in which all sub-expressions are atomic sentences

or negations of atomic sentences. A rule system is a rule-like knowledge base.

A rule system is **Horn** if and only if every constituent of every rule is atomic (i.e. no negations allowed). Otherwise, the rule system is said to be non-Horn.

Dependency graph for a rule system

A rule system is **recursive** if there is a cycle in its dependency graph. Otherwise, the rule system is said to be **non-recursive**.

Term Complexity

The nature of terms defines a second conformance dimension. There are two categories: simple and complex.

A knowledge base is **simple** if and only if the only terms occurring the knowledge base are constants and variables.

A knowledge base is **complex** if and only if it contains terms other than constants or variables, e.g. functional terms or logical terms.

Order

The third conformance dimension concerns the presence or absence of variables.

A knowledge base is **ground**, or zeroth-order, if and only if it contains no variables. Otherwise, a knowledge base in **nonground**.

A knowledge base is **first-order** if and only if there are no variables in the first argument of any explicit functional term or explicit relational sentence.

A knowledge base is **higher-order** otherwise.

Quantification

For nonground knowledge bases, there are two alternatives – quantified and unquantified.

A nonground knowledge base is **quantified** if and only if it contains at least one explicit quantifier.

A nonground knowledge base is **unquantified** if and only if it contains no explicit quantifiers.

Metaknowledge

The final conformance dimension concerns the ability to express metaknowledge, e.g. to write sentences about sentences.

A knowledge base is **baselevel** if and only if it contains no occurrences of the `quote` operator or the `wtr` relation.

Otherwise, the knowledge base is metalevel.

Common Conformance Profiles

A database system is one in which (1) all assertions are atomic, simple, ground, and baselevel and (2) all queries are positive, simple, unquantified, and baselevel.

A first-order system is one that allows the broadest categories within each conformance dimension except that only first-order expressions are accommodated.

A full KIF system is one that accepts the broadest categories within each conformance dimension, i.e. any KIF knowledge base is acceptable in any context.

Dealing with Differences in Conformance Profiles

Knowledge Query and Manipulation Language. Multi-agent systems

The way to deal with an impossible task is to chop it down into a number of merely very difficult tasks, and break each one of them into a group of horribly hard tasks, and each one of them into tricky jobs, and each one of them...
(Terry Pratchett, Truckers)

What's agent?
Why we need distributed computation?

- to distribute the load
- to manage the complexity
- for distributed tasks

Communication between agents

The interaction is a combination of:

- **interaction protocol** — the strategy of the agent
- **communication language** — the expressive mean
- **transport protocol** — the media of the communication

What needs to be negotiated:

- the **syntax** of the message content;
- the **semantics** of the message content;
- the **pragmatics** of the message;

More about the Pragmatics:

- Who to talk to?
- How to find him?
- How to get his attention?
- How to start and maintain the conversation?
- What to do with the messages of the others?
- How to answer adequately?

Some administrative functions also:

- the structure of the society
- the structure of the conversation.

How "open" the ACL should be?

KQML history and philosophy

First published in 1993.
The last proposal for new specification in 1997th.
`http://http://www.cs.umbc.edu/kqml/`

Philosophy

- *KQML* is a high-level *communication language*
- a protocol for information exchange
- an extendable set of pragmatics

KQML Assertions for the transport protocol:

- one-way communication links;
- discrete messages (packages vs. stream);
- the communication can introduce delays;
- the receiver knows the sender;
- First-Sent-First-Received;
- message delivery is reliable.

Basics

The *KQML* primitives are called **performatives** — determine the intention and the protocol

Sample pragmatics: ask, tell, achieve, reply

Each performative consists of:

- type;
- attribute-value pairs;

Sample attributes: content, language, topic, ontology, sender, reply-with.

Agents as VKB

A virtual *KB* with two constituents:

- an information array — the agents believes;
- a goal array — intentions.

She most basic performatives: **tell**, **untell**, **achieve**, **unachieve**.

Kinds if Performatives: core, standard and extended.

Facilitators

To take care for:

- mapping between the physical address and the symbolic names
- support of a registry of the services;
- routing, connecting, etc.

KQML is quite "open"

The dimensions of this flexibility:

- the transport protocol is not fixed
- the KR language — not fixed. The defualt representation is *KIF* ;
- no limitations for the ontology
- the society architecture is not restricted
- new types of performatives can be added;
- new parameters for the existing performatives allowed

KQML syntax

```
<performative> ::=
   (<word> {<white> :<word>
            <white> <expression>}*)
```

An example:

```
(tell :sender A :receiver B ...
      :language PROLOG :content "father(c,d)")
```

KQML semantics

Defined in terms of the changes of the state of the VKB

Open questions

1. Is it acceptable in one VKB to keep expressions in different languages?
2. What about message identity?

The Future — ACL by FIPA

Foundation for Intelligent Physical Agents (FIPA)
http://www.fipa.org