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
Linguistic Modelling Laboratory
Bulgarian Academy of Sciences
Acad. G. Bonchev Str. 25A, 1113 Sofia, Bulgaria
kivs@bgcict.acad.bg
naso@sirma.bg400.bg

Handouts

August 25, 2000

1 Knowledge. Reasoning. Knowledge Representation. Knowledge Representation Systems.

Here we try to give answers to the following questions:

- 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?

Material in this section is partially based on [Poole et. al. 1998] and [Davis, et. al 1996].

1.1 Knowledge. Reasoning. Knowledge Representation.

As most fundamental notions, the concept **knowledge** can not be defined rigorously. We usually associate knowledge with human beings and their abilities for intellectual behaviour with respect to the world and to themselves. Our lack of sufficient understanding of the nature and mechanisms of human knowledge makes the borders of the concept unclear. Most people have no doubt that the achievements of contemporary science are part of human knowledge, but quite often we can hear a child say proudly "I know how to ride a bicycle!". On the other hand, a professor of mathematics could say: "Solving quadratic equations is not knowledge." Because of this vagueness of our intuition about knowledge, the definitions of this concept in the literature range from "Everything is knowledge" to "Knowledge is what could be represented in my (formal) system." Here we start with the first definition and we hope to become more specific as the course progresses.

Definition 1 (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.

In order to support the action of agents, knowledge has to correspond in some way to the world. Knowledge has to reflect the entities in the world and the relations between them. Thus knowledge constitutes a model of the world and agents use their knowledge as a model of the world. In this way knowledge allows an agent to determine the consequences of some action by changing its knowledge state rather than acting in the world. In this way we consider reasoning as a change of the knowledge state of an agent.

Definition 2 (Reasoning) *Reasoning is an internally driven change in the knowledge state of an agent.*

One consequence of regarding knowledge as a model and reasoning as changes in the model is that knowledge is, usually, not perfect with respect to the world. In most cases, knowledge is, at its best, perfect with respect to an abstraction of the world. Thus a given piece of knowledge is good in solving only a class of problems that are coherent with an abstraction of the world. This doesn't mean that no knowledge could be true with respect to a world, rather this means that knowledge couldn't be true with respect to all the characteristics of a world.

Another important point is that knowledge is not necessarily homogeneous. It could be modularised in different chunks so that any chunk represents an aspect of the world. This allows us to speak of knowledge about some phenomena or domains. Also this modularisation supports reasoning by using only the relevant chunks of knowledge and changes only in some parts of the knowledge state.

On the base of the two definitions we define knowledge representation as:

Definition 3 (Knowledge Representation) *Knowledge representation is the study of nature of knowledge and reasoning with the goal to simulate them on a computer.*

Knowledge representation has to answer questions like: "What constitutes a model?"; "How does a model respect the world?"; "How does a model change to reflect the changes in the world?"; "How is a model built?"; "How is a model used to predict the changes in the world?". Having in mind that agents are situated in the world similar questions could be asked about knowledge of agents: "How is knowledge about agent's knowledge and reasoning represented?", "How is knowledge about knowledge and reasoning of other agents represented?".

In knowledge representation, we are not interested in the knowledge of arbitrary agents, but a special class of agents called *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.

Let us consider in what way do human beings behave as intelligent agents. The following are some examples of the ways in which human beings are using their knowledge to achieve some goals:

- **Deduction.** A student is proving a theorem in geometry.

  The student selects the premises that are known to be true in geometry (axioms and already proved theorems), defines the theorem that has to be shown to be true, constructs a chains of claims starting with the premises and using rules in such a way that if the premises are true then the resulting claim is also true. The theorem is true if one such chain of true claims contains the theorem. One interesting thing here is that the claims in a chain have the same formal status. All they are true and if they are not axioms then they are theorems. The difference between the theorem that the student wants to prove and the other theorems in the chain is that the chosen theorem is important in some way for the student.

- **Induction.** A scientist is building a theory about a set of observations.

  The scientist determines a set of characteristics that are important for the observations, after this she/he sorts the observations according to their characteristics. At the end she/he creates a theory that predicts the existence of the observations with appropriate characteristics. A problem arises when more that one theory predicting the observation are possible. Then knowledge outside the observation is used to judge which one is appropriate.

- **Abduction.** A doctor is explaining a new symptom to another doctor.

  A new symptom which doesn't follow from the "theory of human health" and previously known symptoms has been observed. The doctor changes the theory in a "minimal" way so as to produce a new theory which predicts the new symptom and does not reject any of the previous known ones, or the doctor assumes the truth of some other fact that wasn't known before the observation of the new symptom. Again, a problem is the existence of more than one explanation of the symptom and usually the choice is made on the base of other knowledge.

- **Commonsense reasoning.** I am driving on the highway with 140 km/h when the restriction is 120 km/h.

  I don't worry about the police because I have an anti-radar device. Of course I am not 100% sure that the police doesn't have a new radar which can not be detected by my anti-radar device. I

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

assume that the existence of such a new radar is so important that if it exist then I have to know about it.

- **Learning.** A child becomes skilled in using natural language.

  What kind of mechanisms is used by a child in order to recognize what constitutes the language, what kind of innate information there is in the mind of the child, how it is combined with the information received by the child.

- **Planning.** A thief is going to rob a bank.

  The thief determines several conditions which are important for the success: which time is appropriate, from where to enter the bank, how to switch off the alarm, where the money is, how to take it out, how to escape. For each step different tools will be necessary, some of these tools could be used for several tasks. The thief has to foresee that some of the steps may not be successful.

- **Instructions carrying out.** A computer scientist is installing a new printer following a list of instructions.

  The user manual lists a sequence of elementary instructions describing the installation process. Each instruction presupposes a certain state of the world and determines an action to change the state in an appropriate way. If one understands the instructions and is able to carry out the action specified by the instruction sheet, then she/he could install the printer even if they don't know exactly how it works.

- **Communication.** Two friends are talking to each other.

  How does one decide what to say to the other? Which words to use? How do these words map to concepts and notions about the world? How do they understand each other?

On the basis of the above examples, we can identify different kinds of knowledge and reasoning along different axes. We could differentiate between *Deduction* and *the carrying out of instructions*. In the first case we use some basic definitions and facts about **what** is true in the world and with the help of a general reasoning mechanism we decide what else must be true in the same world. In the second case we must know what the instructions mean and **how** to carry them out in the world. We call the first kind of knowledge **declarative** and the second kind **procedural**. In our course we will be concerned with the presentation of declarative knowledge. There is another opposition between *Deduction* and *Abduction*. In the case of deduction if our premises are true we can conclude with certainty that the result of the deduction is also true. In the case of abduction there is no such certainty about the explanation because it is possible that several explanations exist. Another dimension is the difference between *represented knowledge* and *derived knowledge*. Represented knowledge can be used without any additional processing because it is stored *explicitly* in the model. Derived knowledge, on the other hand, needs some reasoning in order to be used. We think of derived knowledge as represented *implicitly* in the model. Usually, knowledge is taken to have **intension** and **extension**. Intension is considered as the internal structure of a chunk of knowledge and its relationship to other knowledge. Extension is the part of a world the knowledge is about. One example is given: "Evening star" and "Morning star" are two concepts with different intensions, but with the same extension — the planet Venus. Very often chunks of knowledge are compared on the base of their extensions and two chunks with different intensions but with the same extensions are considered as equivalent.

As one can notice the main reasoning methods (deduction, induction and abduction) presuppose some degree of uncertainty about which derived knowledge is of use in some situation. These usually can not be decided on the characteristics of the derived knowledge alone. Some additional knowledge about the context, the aims of the agent or agent's knowledge about the use of the derived knowledge is necessary. This is why knowledge representation is considered as "a (fragmentary) theory of intelligent reasoning".

There are several paradigms of knowledge representation today. The main division is between **symbolic** and **non-symbolic** approaches. Supporters of non-symbolic knowledge representation consider knowledge and reasoning to be features of an evolutionary developed organ and claim that the simulation of

the architecture of this organ will give us the possibility to represent knowledge. On the other hand, supporters of symbolic knowledge representation accept the hypothesis that on a certain level of abstraction knowledge is represented as symbols and reasoning is a manipulation of these symbols. Let us remind you of the **Church-Turing thesis**:Any symbol manipulation can be carried out on a Turing machine; thus, we could hope that knowledge could be represented symbolically on a computer and reasoning could be done as symbol manipulation over this representation. In the following lectures we will take the term **knowledge representation** to mean both represented knowledge itself and the appropriate reasoning. Now we can formulate:

Definition 4 (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.*

1.2 Knowledge Representation Systems. Architecture and Requirements.

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.

The actual formal languages and inferences of a KRRS are defined with respect to a set of tasks the system has to deal with.

The formal language is defined by a **syntax**: gives the basic categories of symbols (**vocabulary, signature**) and the rules (**grammar**) of combining the basic symbols in complex expressions; and **semantics**: defines the meaning of the basic symbols and the complex expressions.

The syntax is important because it allows writing distinct types of expressions for the distinct categories of things (objects, relations, functions) in the world we want to speak about. For example, if we want to speak about sets of objects we will need some expressions for sets. The semantics allows us to describe the connection between the expressions and the things these expressions represent. In other words, semantics gives us the similarity between the model (knowledge state) and the world we are trying to represent. This similarity is captured by the notion of "truth" which makes it possible to decide whether an expression is a true proposition about the world. One important point about the semantics is that it doesn't depend on the way the expressions of the language are processed (declarative semantics).

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**. A typical example is Kl-One-based system where in the terminological part a restricted language for concept definition is used and in the assertional part more powerful language can be used.

The expressions in the modules of a knowledge base represent the explicit knowledge represented in the knowledge base. The implicit knowledge represented by these expressions needs some inferences in order to be applied. It is possible to have more than one inference defined. There are several 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.

  Each kind of expressions allowed by the syntax has to be treated by a inference method. The result of an inference method has to be clearly defined. The result has to be interpreted according to the semantics of the language in an rigorous manner. Two syntactically different expressions with the same semantics interpretations have to lead to the same result under an inference method.

- **Soundness.** The results of an inference method have to be right.

  We expect a system to give to us only true propositions that follow from a knowledge base. If some inference method violates this condition, we would like to be able to recognise the situations in which it could produce a wrong answer.

- **Completeness.** An inference method has to be able to explicate all correct implicit knowledge.

  We expect a system to be able to find any implicit proposition in a knowledge base. Again, if an inference method doesn't meet this condition we would like to have an characterisation of this gap in the inference.

- **Decidability.** There exists an algorithm that implements the rules of the inference method preserving all characteristics of the method like completeness and soundness.

  We would like to use the inference method in a real computer system and we want the implementation to keep all properties of the method. There are inference methods that are complete and sound but for which there is no an algorithm to implement them.

- **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. Such algorithms exist only for a very limited number of formal languages. Usually we would like to have more expressive languages. This causes usage of algorithms with worse complexity for the worst cases hoping that they behave well in the usual cases. Another way to overcome this problem is to weaken the semantics. In this case the semantics will qualify less propositions as true.

An KRRS can be characterised on several levels of abstractions [Guarino 1994], [Brachman 1979]:

- **Implementational level.** Here the system is described in terms of memory units (cells, pointers) and procedures.

  There is no semantics prescribed to this level. The same data structures and procedures (graph unification, for example) could be used for different KRRSs.

- **Logical level.** Here the system is described in logical terms like propositions, predicates, functions, logical operators and inference rules.

  A general, uniform and neutral formal semantics is given in terms of relations between objects in the world, but these relations are not distinguished by other characteristics apart from their arities.

- **Epistemological level.** Concept types, structuring relations.

  Generic notions of concept and concept forming operations are defined regardless of any represented knowledge. On this level one defines what constitutes a concept, what kinds of operations are allowed on concept structure and etc.

- **Ontological level.** Categories of predicates classified on the basis of formal ontology primitives.

  Here knowledge primitives are interpreted satisfying the restrictions of a formal ontology which distinguishes: (1) among the entities of the world (physical objects, events, processes...); and (2) among the meta-level categories used to model the world (concepts, properties, states, roles, attributes, various kinds of part-of relations...).

- **Conceptual level.** Conceptual relations, primitive objects and actions.

  At the conceptual level, primitives have a definite cognitive interpretation, corresponding to language-independent concepts like elementary actions or thematic roles.

- **Linguistic level.** Linguistic terms.

  Knowledge is associated directly to nouns and verbs and is managed by linguistic mechanisms.

Keep in mind than by far not every KRRS supports all of these levels of representations.

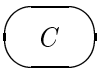
2 Kl-One Family

Kl-One[2] originated from the idea of structured inheritance networks developed in the dissertation of Ronald Brachman as a reply to the criticism of William Woods to the informal in those days knowledge representation systems. Kl-One presents a theory of concept structure description. Here, we first give an introduction to Kl-One based on [BS 1985], and then we present a formal description of $\mathcal{ACLNR}$ language ([BH 1991]) which is one of the descendants of Kl-One and a method for inference in this language

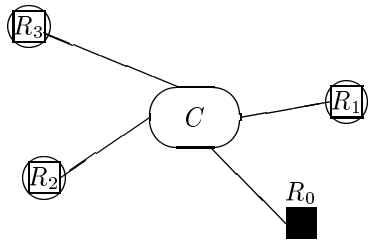
2.1 Introduction

Let us consider the following hypothetical scenario for the process of concept forming by humans:

1. A group of objects in the world are recognised as belonging to a concept, called **generic concept**. We give a name to it. We depict the concept as an ellipse. The concept's name is written inside:

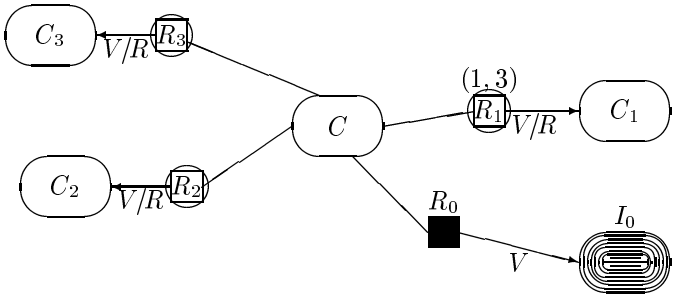


2. In the next step, we determine the properties that characterise the concept: these properties are called **roles**. There are two kinds of properties: **generic roles** for properties that have different values for different representatives of the concept; and **individual roles** for properties that have the same values for all representatives of a concept. Generic roles are called also RoleSets. We depict a generic role as square surrounded by a circle and individual role by a filled square:

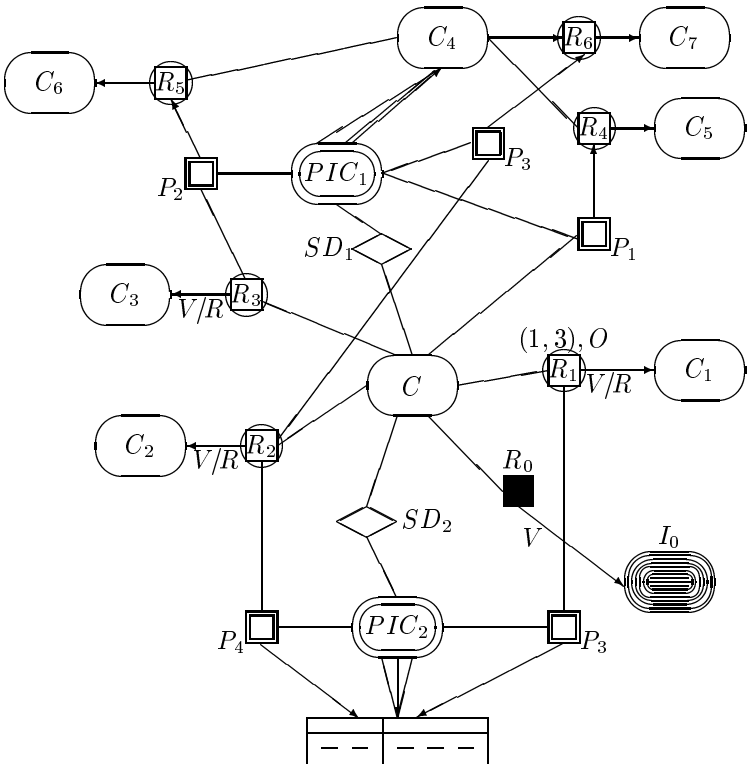


[2] This name stands for a system developed at Bolt Beranek and Newman Inc., the language of this system and a whole family of languages that follow the same knowledge representation philosophy.

3. The next step in the definition of a concept is to state some constraints on the roles of the concept. These constraints are of two kinds: **Value Restriction** (V/R) which describes the potential fillers for a generic role and **Number Restriction** which expresses the cardinality information for a generic role. Value restriction is stated by a generic concept and numeric restriction is given as a pair of numbers, a lower and upper bound, defining the range of cardinalities for the set of values. "NIL" is used for infinity. The value for an individual role is given by an individual concept (filled ellipse). These restrictions are depicted in the following way:

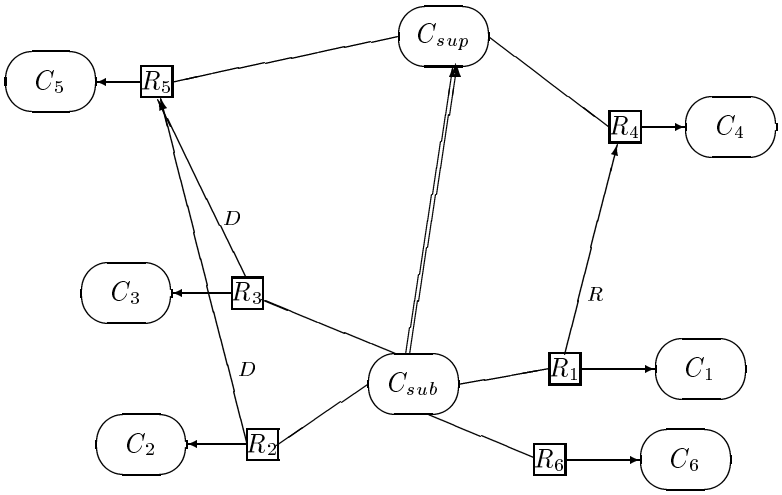


4. Besides constraints on the fillers of a given role, we need to impose some constraints on the fillers of different roles in a concept or on configurations of roles. These general constraints are given by the so called **structural descriptions**, depicted as diamonds. Each structural description connects some parts of a concept definition with the roles of another concept. The meaning is that the corresponding parts of the concept have to meet the conditions imposed by the other concept. A special case of structural description is **Role Value Map**. In this case the equality of the fillers of some roles is stated. Correspondences between the elements of the definition of the current concept and the roles of the constraining concept is given by so called **Parametric Individual Concept** Structural descriptions are also a convenient way to impose constraints defined by external procedures.



This picture demonstrates all Kl-One epistemological constructs which defines a concept locally.

5. Additionally, we would like to compare the concept with the other concepts we have already defined. We want to find similar concepts, concepts that are more general and concepts that are more specific. Also we would like some common characteristics for several concepts to be represented only once and to be inherited by the appropriate concepts. Kl-One provides means for solving all this problems by arranging the concept in concept hierarchy called **taxonomy**. Concepts in this hierarchy are ordered with respect to subsumption relation. A generic concept subsumes another generic concept, if any instance of the second is necessarily an instance of the first. A concept subsuming another concept is called **superconcept** and the subsumed concept is called **subconcept**. The connection of a superconcept to a subconcept is called **specialization**. A concept can have more that one superconcept (multiple inheritance). When a concept inherits definitions of generic roles from a superconcept these definitions may be modified in two ways: **restriction** — "If Concept A with RoleSet Ra subsumes Concept B, and if RoleSet Rb of B restricts Ra, then every set of fillers of Rb satisfies all restrictions on *both* Ra and Rb. Moreover, Ra and Rb designate the same two-place relation"; **differentiation** — "A RoleSet differentiates another when the former denotes a subrelation of the relation denoted by the later." More formally: " If RoleSet Rb differentiates RoleSet Ra, then any pair of individuals that satisfy the relation denotes by Rb also satisfies the relation denoted by Ra. Furthermore, all individuals in the range of the relation denoted by Rb satisfy the Value Restrictions of both Ra and Rb. The maximum (cardinality of the image of the relation for any individual in the domain) specified in the Number Restriction of Ra is also the maximum for Rb, unless a smaller maximum is specified directly at Rb. The minimum for RB is 1, unless a larger minimum is specified directly at Rb."



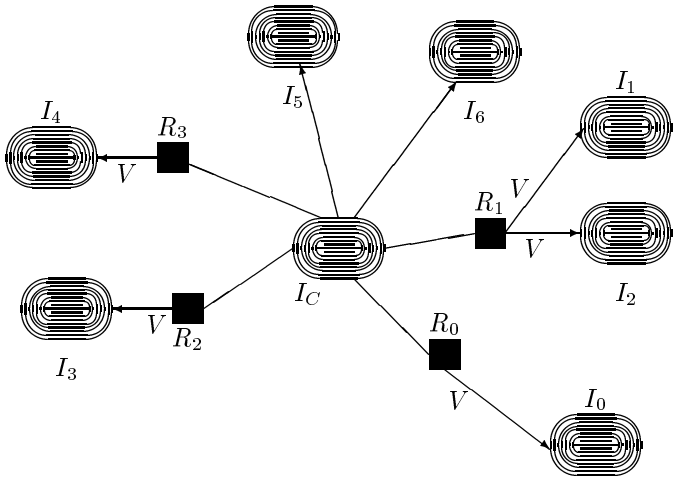
The generic concepts in a taxonomy define the terms in which we can speak for the world. The local and the inherited information impose the constraints on the instances of these concepts. There are two kinds of concepts: **Primitive Concept** — the definition of such a concept impose only necessary conditions that the instances of the concept must satisfy, but not all individuals that satisfy this conditions are instances of the concept (these are not sufficient conditions). Primitive concepts are used to represent of *natural kinds* like `dog` or `lemon` for which one can not provide sufficient conditions, but can specify a rich set of necessary conditions. For example, to completely define the concept `person` it is not sufficient to state that a `person` is an `animal` with two legs and two hands, and that is why that concept is declared as primitive. With such a primitive concept we do not say what distinguishes a `person` from an `animal`; we just say that there is some specialization which makes `person` a subclass of `animal`, and we give it a name. Some times primitive concepts are represented only by their names (atomic concepts). **Defined Concept** — the definition of such a concept impose necessary and sufficient conditions on the instances of the concept. Every

individual that satisfies the conditions is an instance of the concept. A concept has also the name of its superconcepts as a part of its definition, thus one could define a concept only by specifying the exact place of the concept in the taxonomy. Quoting [BS 1985]:

> The components of a concept are
>
> - its subsuming concepts (its superconcepts).
> - and its local internal structure (its component restrictions) expressed in
>   - roles, which describe potential relationships between instances of the concept and those of other closely associated concepts (i.e., its properties, parts, etc.), and
>   - structural descriptions which express the interrelations among the roles.
>
> and
>
> A Kl-One concept is well-formed if
>
> - it has more than one superconcept (if there are no local restrictions, a concept is defined as conjunction of its superconcepts).
> - differs from its superconcept in at least one restriction.
> - it is primitive.

The taxonomy is usually called **terminology** or **T-Box** and states universal propositions about the world, i.e. if in the world there is at least one instance of a concept C then it has to meet all the conditions stated in the definition of the concept, but the definition itself doesn't imply existence of any instances of the concept.

6. When we have a definition of a concept we could use it to recognize instances of the concept. We depict an instance of a concept by an filled ellipse which is connected with individual roles, where each role corresponds to a generic or individual role in the definition of the generic concept. Every individual role is connected with individual concepts that represent the values of this role. Also it is connected with instances of its superconcepts. This fact is depicted by a thick arrow. An individual is an instance of a generic concept either if it is stated as such or if it satisfies all necessary and sufficient conditions. All represented individual concepts in the system make some assertions about the world. This part of the knowledge base in Kl-One is called **assertional component** or **A-Box**.



As it was mentioned above the definition of a primitive concept imposes only necessary conditions on the instances of this concept. Thus, the name of a primitive concept stands for the missing sufficient conditions. Two primitive concepts with the same definitions but with different names are taken to be different. Therefore in order to recognize an individual as an instance of a primitive concept, it is not enough the instance to satisfy the conditions stated by the definition of the

primitive concept, it is necessary the individual to be recognized as an instance of a subconcept of the primitive one or it to be stated explicitly as an instance of the primitive concept. Also it is possible to use some default reasoning based on prototypical information in order to recognize one individual as a typical instance of a primitive concept (see [Franconi, et. al 1992]).

7. Finally, we would like to show how we reason with these representation. The main inference method is **classification**. A generic concept is defined by its local properties: roles, restrictions on the roles and structural descriptions; and some of its superconcepts (not necessary the most specific ones), and the task is to find the right place of this concept in the taxonomy. i.e. the set of its immediate superconcepts and the set of its immediate subconcepts have to be found. The classificational procedure is known as **classifier**. On the base of classifier one can answer several questions about the represented knowledge such as: "Does this definition of a concept make sense?", "Which generic concepts in the T-Box are possible descriptions of an partial individual concept?". For example, individuals in ABox are related with the terms representing their partial descriptions. The set of all the most specific concepts, from the terminology, which describes an individual gives the type of the individual; it is obtained by classifying in the terminology the complete "abstraction" of the individual, i.e., a concept describing only that individual. Types for individuals are maintained by a procedure called *recognizer*. The recognizer identifies the individual with respect to the known concepts by classification of the abstraction of the individual in the concept hierarchy. A comprehensive list of inferences is given below.

The scenario we have just described is not the only possible, but it demonstrates a possible motivation for developing a language similar to Kl-One. Also, it introduced most of the important ideas in Kl-One. Notice that the above introduction is on epistemological level. It describes concept forming operations and relations between concepts without pointing to any logical language and/or to some particular conceptualization of the world. Additionally to these there is an attempt to build corresponding primitives on linguistics level by William Woods. In the paper [BS 1985] each figure with a concept definition is accompanied by a text in restricted English which conveys the same formal meaning as the concept definition itself. For example the following is the text associated with the concept definition in figure 2 on page 183: "A MESSAGE is, among other things, a THING with at least one Sender, all of which are PERSONs, at lease one Recipient, all of which are PERSONs, a Body, which is a TEXT, a SendDate, which is a DATE, and a ReceivedDate, which is a DATE." In the original system there is nothing done on ontological level. One could represent a concept RED in the same way in which represents the concept PERSON. The subsequent developments of the descendants of Kl-One are mainly on logical level. We now turn to one of this descendants - the concept language $\mathcal{ALCNR}$.

2.2 Terminological logics

In this section we describe a common subset of the formal languages, the inferential capabilities, and the common architecture of the descendants of the Kl-One knowledge representation language. These descendants are known under several names as: description logics (DLs), concept languages, term description languages, terminological logics, taxonomic logics. There are several Knowledge Representation Systems built around these languages: KRIPTON [BPL 1985], NIKL [KBR 1986], BACK [QK 1990], LOOM [MB 1987], CLASSIC [BBM+ 1989], YAK [Franconi 1990], $\mathcal{KRIS}$ [BH 1991], and some others. The presentation will follow the syntax and the model-theoretic semantics of the language $\mathcal{ALCNR}$ (see [SSS 1988], [HN 90], and [BDS 1993]).

2.2.1 Description Logics Language $\mathcal{ALCNR}$

Description logics deal with three categories of syntactic primitives: concepts, roles and individuals. Concepts denote classes of objects in the domain under examination, roles denote binary relation on the domain, and individuals stand for the objects in the domain. The following definitions concern the $\mathcal{ALCNR}$ concept language, but they are true for almost all concept languages.

Definition 5 (Signature)
$\Sigma = < CN, RN, IM >$ *is a* $\mathcal{ALCNR}$ *signature if and only if*
CN *is a set of concept names,*
RN *is a set of role names, and*
IN *is a set of individual names.*
The three sets are pairwise disjoint.

Definition 6 (Concept Terms) *C and D are concept terms if and only if they are formed by means of the following syntax:*

$C,D \longrightarrow CN$			*(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)*
$(\geq nR)$	$\mid (\leq nR)$		*(number restrictions)*

where CN is a concept name, C and D are concept terms, and R is a role term (see below). $\top$ and $\bot$ are two special concept names denote the universal concept and the absurd concept respectively.

Definition 7 (Role Terms) *R is a role term if and only if it is formed by means of the following syntax:*

$R \longrightarrow P_1 \sqcap \ldots \sqcap P_k$ *(role conjunction)*

where $k \geq 1$.

Concept and role terms are the expression which are used to define the necessary concepts and roles. Concept names are used to designate concepts in simple manner. The conjunction states that a concept or a role is a specialization of the conjuncts. For instance, a woman is a human and a female. Thus a woman has the properties of a human and properties of a female. The disjunction states that a concept has as subconcepts the disjunct concepts. For instance, a human is a man or a woman. The complement states that a concept is everything that is not something else. Usually, the complement is used in conjunction with some other concepts that restrict the scope of the negation. For instance, a bachelor is a man and is not married. The universal and existential quantification states the value restriction for a role. In the first case, every filler of a role has to be an instance of a given concept, in the second case at least one filler has to satisfy the appropriate concept. For instance, a father is a man who has a child that is a human; a father of sons is a father all children of whom are men. Number restriction state the lower and upper bounds of the numbers of the fillers for a role. For instance, a happy mother is a mother with at least two children and at most three children. This language allows only the specialization operation for roles.

Definition 8 (Interpretation)
$\mathcal{I} = < \mathcal{D}, [\![.]\!] >$ *is an interpretation if and only if*
$\mathcal{D}$ *is a non-empty set of objects,*
$[\![.]\!]$ *is a total function*
from CN to the powerset of $\mathcal{D}$,
from RN to the powerset of the Cartesian product $\mathcal{D} \times \mathcal{D}$, and
from IN to $\mathcal{D}$
(any two distinct individual names are mapped to two different objects — Unique Names Assumption (UNA)) ,
such that the following equations are satisfied:
For concept terms:

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

For role terms:

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

An interpretation connects the concept and role terms with an abstraction of the world. These abstraction is given by a set of objects (called **domain**) and the sets of unary and binary relations over the set of objects. The interpretation connects the individual names with objects in the domain. the concept names with some of the unary relations over the domain and the role names with some of the binary relations over the domain. The connection established by the interpretation must satisfy the conditions for the complex terms. For instance, conjunction is the relation which is an intersection of the relations designated by the conjuncts.

Definition 9 (Terminological Statement)
Let CN be a concept name, and C is a concept term. Any terminological statement has one of the following forms:

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

Definition 10 (Terminological Statement Satisfiability) *An interpretation $\mathcal{I} = < \mathcal{D}, [\![.]\!] >$* satisfies
a concept definition $CN \doteq C$ if and only if $[\![CN]\!] = [\![C]\!]$, and
a primitive concept definition $CN \dot{\leq} C$ if and only if $[\![CN]\!] \subseteq [\![C]\!]$.

A concept definition $CN \doteq C$ states that the concept term C contains the necessary and sufficient conditions one object to be an instance of the concept CN. A primitive concept definition $CN \dot{\leq} C$ states only necessary conditions. These two facts are captured by the definition of satisfiability of terminological statements. All terminological statements in a knowledge base form the TBox. They define the terms in which one can speak about the world. They state only universal propositions in sense that they don't presuppose existence of any objects in the world. But if the world contains instances of some concept these instances must obey the concept definition. The existence of particular objects and their properties are stated in the assertional part by so called **assertional statements**.

Definition 11 (Assertional Statement) *Let C be a concept term, R is a role term, and a, b are individual names. Any assertional statement has one of the following two forms:*

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

Definition 12 (Assertional Statement Satisfiability)
An interpretation $\mathcal{I} = < \mathcal{D}, [\![.]\!] >$ satisfies
the assertional statement $C(a)$ if and only if $[\![a]\!] \in [\![C]\!]$,
the assertional statement $R(a, b)$ if and only if $([\![a]\!], [\![b]\!]) \in [\![R]\!]$.

The assertional statement $C(a)$ expresses that the individual denoted by a satisfies the conditions of the concept term C and the assertional statement $R(a, b)$ expresses that between the individual a and the individual b the relation denoted by R holds. All assertional statements in the knowledge base form the ABox.

2.2.2 Knowledge Base in $\mathcal{ALCNR}$

Definition 13 (Knowledge Base) *Ω is an $\mathcal{ALCNR}$ knowledge base if and only if $\Omega = <\mathcal{T}_{\mathcal{BOX}}, \mathcal{A}_{\mathcal{BOX}}>$, 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.*

Definition 14 (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 $\mathcal{A}_{\mathcal{BOX}}$.*

2.2.3 Inferential Problems

Any system, supporting $\mathcal{ALCNR}$ language, has to provide inferences to solve at least the following problems ([BBH+ 1990],[BDS 1993]):

1. *KB-satisfiability*: Ω is *satisfiable* if and only if it has a model.
2. *Concept Satisfiability*: C is *satisfiable* with respect to Ω if and only if there exists a model $\mathcal{I} = <\mathcal{D}, [\![.]\!]>$ of Ω such that $[\![C]\!] \neq \emptyset$.
3. *Concept Subsumption*: C is *subsumed* by D, written $D \sqsubseteq C$ with respect to Ω if and only if for every model $\mathcal{I} = <\mathcal{D}, [\![.]\!]>$ of Ω it holds that $[\![C]\!] \subseteq [\![D]\!]$.
4. *Concept Equivalence*: C is *equivalence* to D with respect to Ω, written $C \approx_\Omega D$ if and only if for every model $\mathcal{I} = <\mathcal{D}, [\![.]\!]>$ of Ω it holds that $[\![C]\!] = [\![D]\!]$.
5. *Classification in $\mathcal{T}_{\mathcal{BOX}}$*: find all most specific concepts D with respect to the subsumption relation such that $D \sqsubseteq C$ with respect to Ω and all more general concepts E with respect to the subsumption relation such that $C \sqsubseteq E$ with respect to Ω.
6. *Taxonomy construction*: find the smallest relation on the concepts in $\mathcal{T}_{\mathcal{BOX}}$ such that its transitive closure is the subsumption relation (modulo Concept Equivalence, see above).
7. *Deduction*: what facts are deducible from Ω.
8. *Instance Checking*: a is an *instance* of C, written $\Omega \models C(a)$, if and only if the assertion statement $C(a)$ is satisfied in every model of Ω.
9. *Realization*: find the most specific concepts with respect to $\mathcal{T}_{\mathcal{BOX}}$and the subsumption relation such that some individual from the $\mathcal{A}_{\mathcal{BOX}}$ is instance of them.
10. *Retrieval*: which individuals occurring in the $\mathcal{A}_{\mathcal{BOX}}$ are instances of C.

2.2.4 Sub-Languages

Different sub-languages are obtained when some restriction on the forming operators are imposed. Some of more popular ones are ([HN 90]):

$\mathcal{ALCN}$ consists of concepts containing no intersections of roles.

$\mathcal{ALCR}$ consists of concepts containing no number restrictions.

$\mathcal{ALC}$ consists of concepts containing no number restrictions and no intersections of roles.

2.2.5 Example

We use `typewriter font` for concepts, *slanted fonts* for roles and **Bold with initial capital** for individuals.

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

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

`woman` $\doteq$ `human` $\sqcap$ `female`

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

`man` $\doteq$ `human` $\sqcap$ `male`

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

`mother` $\doteq$ `woman` $\sqcap \exists$ *has-child*.`human`

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

`father` $\doteq$ `man` $\sqcap \exists$ *has-child*.`human`

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

`parent` $\doteq$ `father` $\sqcup$ `mother`

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

`grandparent` $\doteq$ (`father` $\sqcup$ `mother`) $\sqcap \exists$ *has-child*.`parent`

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

`grandmother` $\doteq$ `woman` $\sqcap$ `grandparent`

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

`grandfather` $\doteq$ `grandparent` $\sqcap \neg$ `grandmother`

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

`mother-without-daughter` $\doteq$ `mother` $\sqcap \forall$ *has-child*.`male`

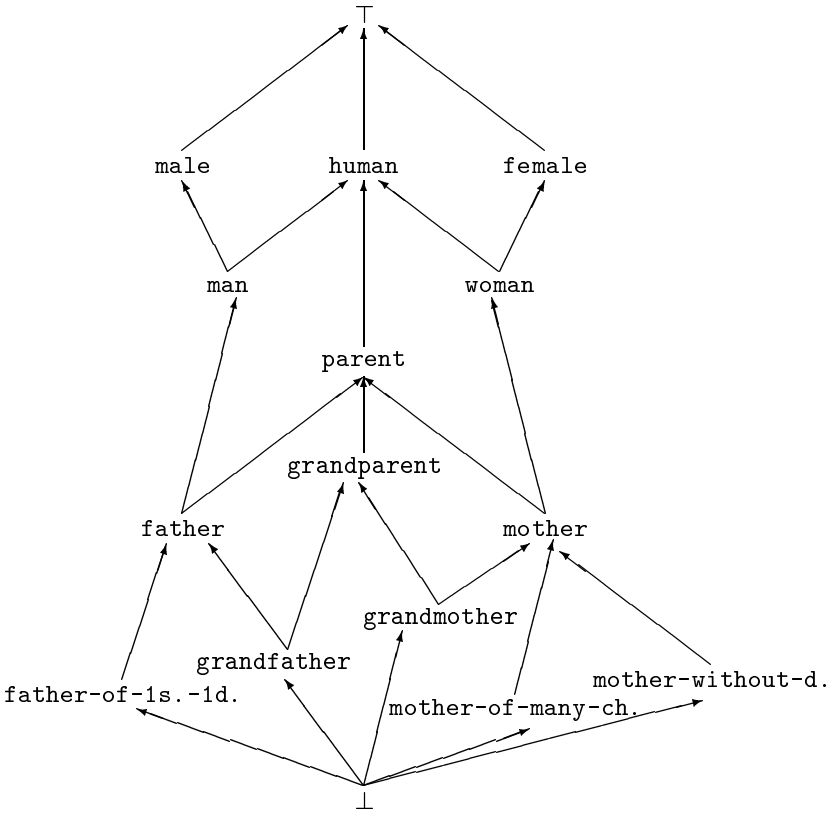
Every mother whose children are male is a mother without daughter.

`mother-of-many-children` $\doteq$ `mother` $\sqcap$ (≥ 3 *has-child*)

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

`father-of-a-son-and-a-daughter` $\doteq$ `father` $\sqcap$ (≤ 2 *has-child*) $\sqcap \exists$ *has-child*.`male` $\sqcap \exists$ *has-child*.`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.



2.3 Deduction

Here, we present some algorithms for deduction in Description logics. We are concerned especially with the following problems: concept satisfiability, concept subsumption and taxonomy construction. First we show how to reduced concept satisfiability to concept subsumption and vice versa. Then we present an algorithm for testing of concept satisfiability. Afterwards several algorithms for taxonomy construction are given. We assume that we are working with respect to acyclic knowledge base. This means that a concept name CN doesn't appear in the definition of CN directly or indirectly. This allows us to assume that the knowledge base is empty in a sense that we could expand the definition of each concept by the definitions of other concepts in order to have only roles and primitive concepts in the definitions. Thus we could work only with concept terms without consulting the knowledge base. One another restriction is that we work with terminological statements of the kind $\texttt{C} \doteq \texttt{CT}$.

For example:

In the definition of concept `grandparent` we substitute concepts `father`, `mother` and `parent` with their definitions and we received the following definition for `grandparent`:

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

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

Here we have to make the substitution again for concepts `man`, `woman`, `father` and `mother`. Afterwards, we will have to make new substitution and so on up to the moment when no more substitutions are possible. We give here the expanded definition. Notice that it contains only primitive concepts (`human`, `male` and `female`):

$\texttt{grandparent} \doteq (((\texttt{human} \sqcap \texttt{male}) \sqcap \exists\ \textit{has-child}.\texttt{human}) \sqcup ((\texttt{human} \sqcap \texttt{female}) \sqcap \exists\ \textit{has-child}.\texttt{human}))$

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

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

Of course this definition looks awful, but keep in mind that it is only internally used for inference purposes

and is invisible to the user.

Let C and D be two concepts. Concept C subsumes concept D if and only if the concept $\neg C \sqcap D$ is unsatisfiable.

Let C be a concept. Concept C is satisfiable if and only if $\bot$ doesn't subsume C.

The reduction of the two problem to each other can be done in linear time.

For example, if one wants to check whether a mother with at least three children is subsumed by a mother with at least two children

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

is equivalent to one to check satisfiability of the following concept:

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

which is equivalent to

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

and which is obviously unsatisfiable. Thus, the initial problem has a positive answer.

Another preprocessing which can be done in linear time and simplifies the presentation of the algorithm is to convert the concept to its **negative normal form**. The following rewriting rules can be used to convert each $\mathcal{ALNR}$ in negative normal form:

$$
\begin{array}{rcl}
\neg\top & \longrightarrow & \bot \\
\neg\bot & \longrightarrow & \top \\
\neg(C \sqcap D) & \longrightarrow & \neg C \sqcup \neg D \\
\neg(C \sqcup D) & \longrightarrow & \neg C \sqcap \neg D
\end{array}
$$

$$
\begin{array}{rcl}
\neg\neg C & \longrightarrow & C \\
\neg(\forall R.C) & \longrightarrow & \exists R.\neg C \\
\neg(\exists R.C) & \longrightarrow & \forall R.\neg C \\
\neg(\leq nR) & \longrightarrow & (\geq n+1R) \\
\neg(\geq nR) & \longrightarrow & \begin{cases} \bot & \text{if } n = 0 \\ (\leq n-1R) & \text{if } n > 0 \end{cases}
\end{array}
$$

Let us consider an example:

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

$\Downarrow$

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

$\Downarrow$

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

The last step in the deduction process is done by satisfiability check for a concept term expanded and converted to negative normal form. The algorithm works on a construction called a constraint system. First, a countable set of variables is assumed. We use x, y and z for variables.

There are two kinds of constraints: $x : C$, xPy. Where C is a concept term expanded and converted to negative normal form and P is a role name. A **constraint system** S is a finite, nonempty set of constraints. In an interpretation $\mathcal{I}$, variables are mapped to objects in the domain of $\mathcal{I}$ by a function α called $\mathcal{I}$-assignment. An $\mathcal{I}$-assignment α satisfies the constraint $x : C$ if and only if $\alpha(x) \in [\![C]\!]$ and it satisfies a constraint xPy if and only if $(\alpha(x), \alpha(y)) \in [\![P]\!]$. A constraint is satisfiable if there exists an interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment such that the $\mathcal{I}$-assignment satisfies the constraint. A constraint system is satisfied by an $\mathcal{I}$-assignment if and only if it satisfies all constraints in the system. At the end, a constraint system is satisfiable if and only if there exists an interpretation $\mathcal{I}$ and an $\mathcal{I}$-assignment such that the $\mathcal{I}$-assignment satisfies the system.

Let S be a constraints system and $R = P_1 \sqcap \ldots \sqcap P_k$ be a role. We say that xRy holds in S if and only if the constraints $xP_1y, \ldots, xP_ky$ are in S. For a role R, a constraint system S and variable x we define a function which returns the number of the variables y such that xRy holds in S:

$$n_{R,S}(x) = \|\{y \mid xRy \text{ holds in } S\}\|^3$$

The satisfiability algorithm is given as a system of 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. $S \longrightarrow_{\exists} \{xP_1y, \ldots, xP_ky, y : C\} \cup S$
   if $x : \exists R.C$ 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

Here $[y/z]S$ is the constraint system obtained from S by replacing each occurrence of y with z and 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 $n_{R,S}(x) > n$.

The above rules are of two kinds: **deterministic**: $\longrightarrow_{\sqcap}$, $\longrightarrow_{\exists}$, $\longrightarrow_{\forall}$, $\longrightarrow_{\geq}$; **nondeterministic**: $\longrightarrow_{\sqcap}$, , $\longrightarrow_{\leq}$.

A **clash** is a constraint system having one of the following forms:

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

A constraint system is complete if and only if no rule is applicable to it and it doesn't contain a clash. Any complete constraint system is satisfiable. All rules above preserve the satisfiability, i.e. if S is satisfiable and S' is obtained from S by application of one of the rules the S' is also satisfiable and if S is not satisfiable then S' is also not satisfiable.

Let C be an expanded concept term converted in negative normal form then the system $\{x : C\}$ is satisfiable if and only if C is satisfiable.

Example:

Let us check whether man is subsumed by father:

`man` $\sqsubseteq$ `father` ?

First, reduction to satisfiability problem:

`man` $\sqcap \neg$ `father`.

[3] For set X, $\|X\|$ is the number of the elements of X.

Next, expansion of the definitions (we give the end result):

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

Afterwards, conversion to negative normal form (the end result):

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

We convert this to a constraint system and apply the rules:

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

$\Downarrow$ 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 $\longrightarrow_{\sqcup}$ and 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 $\longrightarrow_{\sqcup}$ and we choose the left disjunct

$\{x : \texttt{human},\ x : \texttt{male},\ x : \neg\ \texttt{human}\ \}$. **Clash.**

$\Downarrow$ return to point 2, by rule $\longrightarrow_{\sqcup}$ and we choose the right disjunct

$\{x : \texttt{human},\ x : \texttt{male},\ x : \neg\ \texttt{male}\ \}$. **Clash.**

$\Downarrow$ return to point 1, by rule $\longrightarrow_{\sqcup}$ and we choose the right disjunct

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

This constraint system is complete. The rule $\longrightarrow_{\forall}$ is not applicable because the condition xRy to hold for some y is not met. Man is not a subconcept of father because there are men without children.

The algorithm defined on the base of the system of rules is PSPACE-hard and requires nondeterministic exponential time. The hard part is the choice in the case of nondeterministic rules. If equality is included in the language then the satisfiability problem is undecidable.

2.4 Taxonomy construction. Classification.

The goal of taxonomy construction is to build the minimal relation between the concepts in the terminology (in TBox) such that its transitive and reflexive closure to be the subsumption relation between the concepts in TBox. For example, in the TBox given on page 15, `grandmother` is immediate subconcept of `mother` but it is defined by the superconcept `woman`. In the Tbox only the subsumption between `grandmother` and `mother` is stored. The relation between `grandmother` and `woman` will be in the transitive closure of the relations between `grandmother` and `mother` and `mother` and `woman`. The material in this section is based on [BHN+ 1993]. Some of the ideas originated from [Lipkis 1982], [Levinson 1984], [Levinson 1992], [MacGregor 1988] and [Ellis 1991].

Classification is closely related to the problem of taxonomy construction. Let us suppose that we have already built a taxonomy and we have a new concept C which we want to classify with respect to the taxonomy. This is equivalent one to construct a new taxonomy which contains also the new concept and then to extract the immediate superconcept and the immediate subconcepts of C. Thus, one can construct a taxonomy starting with an empty one (only universal and absurd concepts) and then classifying the concepts in the TBox one by one. In this classificational approach one will need to compare every new concept with (some) concepts already in the taxonomy with respect to subsumption relation. As we saw in the previous section for reasonable description languages, subsumption check is a very expensive operation. Thus, in order the taxonomy construction and the classification to be efficient, one would like to do as less subsumption check as possible. The different methods given below improve on this criterion.

On general level the problem is defined as follow: if a partial order $\leq$ over the set X is given a relation of precedence $\prec$ over X to be found such that its transitive and reflexive closure to be equivalent to the partial order. In the case of total order the construction of the relation $\prec$ is called sorting. In the case of partial order it is called identification problem. In the case of description logics, the partial order is

defined by the subsumption relation between concepts.

Some technical notions: first, $x \prec y$ if and only if $x \leq y$ and there is no z different from x and y such that $x \leq z \leq y$. If $x \leq y$, we say that x *succeeds* y or that y *precedes* x. If $x \prec y$ we say that x *immediately succeeds* y or that y *immediately precedes* x.

All presented methods are working incrementally along the lines described above. On a given step, there is a precedence relation $\prec_i$ built for some subset $X_i \subseteq X$, on the next step for an element $c \in X \setminus X_i$ the relation $\prec_{i+1}$ is built over the set $X_{i+1} = X_i \cup \{c\}$. The most important procedure in this step in the construction of two set of elements of X_i. The set of the elements that immediately follows c and the set of the elements that immediately precedes c:

$X_i \downarrow c := \{x \in X \mid c \prec x\}$ — (top search)

$X_i \uparrow c := \{x \in X \mid x \prec c\}$ — (bottom search)

The construction of these sets is equivalent to classification in description logics. The main difference is that subsumption relation on concepts is only a quasi-order because of the equivalence of concepts.

2.4.1 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.

2.4.2 Methods of Simple and Enhanced Traverse

Obviously, in the method of brute force, most of the comparisons between elements are unnecessary. The already constructed part of the hierarchy could be used when the two sets are built for a new element. Here, we present the top search algorithm. Let us suppose we have built already the taxonomy for a set X_i and c is a new concept. Let $x \in X_i$ be such that $c \not\leq x$ then for each other concept $y \in X_i$ such that $y \leq x$ we can conclude without any checks that $y \not\leq c$. That is, if a concept c is not a subconcept of another concept x then c is not a subconcept of any subconcept of x. The top search algorithm starts at the top of the already built taxonomy ($\top$). Then the algorithm follows down the paths in the taxonomy. It stops to follow some paths if their common prefix contains a concept that doesn't precede the new concept c. In this way, some subsumption checks for which is known that will fail are avoided. This method is called *simple traversal method*. The algorithm is given as two functions – one for the top-search itself and one for the subsumption check:

```
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
```

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):

```
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
```

In the simple traversal method we explore the taxonomy in depth first manner. In this, way some unnecessary checks can not be avoid. For example, if the concept c is subconcept of the concept x then it is subconcept of all superconcepts of x. Thus, it makes sense we to check first the superconcepts of x and if these checks succeeded the to check c against x. This can be achieved by modification of the second function given above in such a way that for a given node to check first its immediate predecessors and then if these checks are successful then to call the subsumption checker for the current node. This function is called `enhanced-top-subs?` and is given below. Just described procedure is using negative information on subsumption check (if one of the predecessors doesn't subsume the new concept, then the current node also doesn't subsume the new concept). It is possible we to use also positive information. If one of the successors of the current node subsumes the new concept then the current node also subsumes the new concept. As it is shown in [BHN+ 1993], the two methods can be combined by a new change in `simple-top-subs?` which also propagates the negative information down in the taxonomy. The method using these optimizations is called *enhanced traversal method.*

So far, the top and the bottom search don't interact with each other. If we do, firstly, the top search and then the bottom search we can use the results from subsumption checks already done during the top search. 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
```

2.5 Non-standard Semantics

In the following we show that if one changes the semantics of the language, it will lead to an easier algorithm. The material in this section is based on [Patel-Schneider 198?].

Let us take the following language. The signature this time comprises two disjoint sets of symbols: concept names — CN and role names — RN.

The concept terms are formed according to the following syntax:

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

where CN is a concept name, C and D are concept terms, and R is a role term. And role terms are formed by means of the following syntax:

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

where P_i are role names.

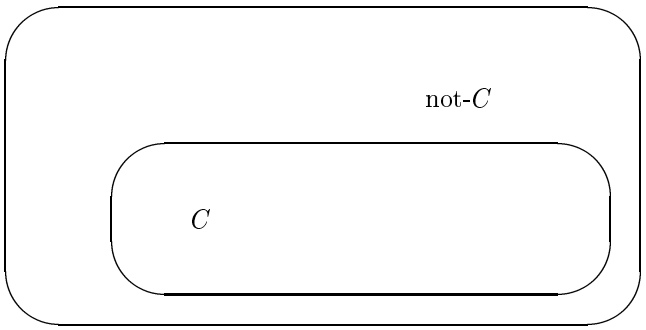
This language is undecidable with respect to the standard semantics (see above). The concept forming operator $(rvm\ R\ S)$ and the role forming operator $(restr\ R\ C)$ have the following standard semantics:

Let $\mathcal{I} = < \mathcal{D}, [\![.]\!] >$ be an interpretation then

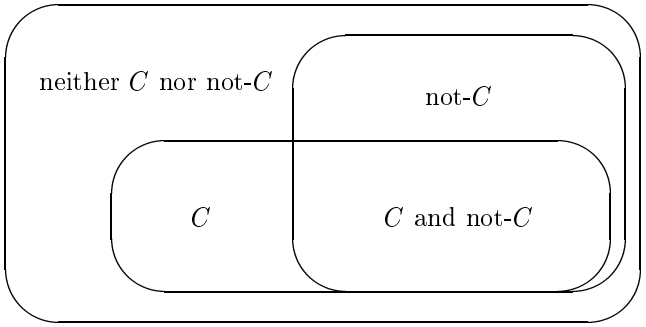
$$\begin{aligned} [\![(rvm\ R\ S)]\!] &= \{d \in \mathcal{D} \mid \{d_r \in \mathcal{D} \mid (d, d_r) \in [\![R]\!]\} = \{d_s \in \mathcal{D} \mid (d, d_s) \in [\![S]\!]\}\} \\ [\![(restr\ R\ C)]\!] &= \{\langle d, e\rangle \in \mathcal{D} \times \mathcal{D} \mid \langle d, e\rangle \in [\![R]\!] \wedge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

Here, we present a new four-valued semantics which allows design of an algorithm which runs in time proportional to the squared the sum of the sizes of the two concept terms. In the standard semantics a given concept divides the domain of each interpretation in two complementary subsets of objects: the set of objects that belong to the concept and the set of objects that don't belong to the concept. These subsets are necessary disjoint. In the new four-valued semantics the interpretation again maps the concept to two sets: one containing the positive representatives of the concept and one containing the negative representatives of the concept. But this time these two sets are not complements to each other and they are not necessarily disjoint. The following picture shows the difference between the standard two-valued semantics and the four-valued semantics:

Standard semantics:



Non-standard semantics:



Similarly, roles are mapped to two sets of ordered pairs of objects. Of course there are new restrictions on the mapping of the complex concepts. For instance, for the concept $C \sqcap D$ the set of the positive examples is the intersection of the sets of the positive examples of C and D and the set of the negative examples is the union of sets of their negative examples.

Here is the definition of these semantics:

A interpretation is a triple $\mathcal{I} = \langle \mathcal{D}, [\![.]\!]^t, [\![.]\!]^f \rangle$, where $\mathcal{D}$ is a set of objects. $[\![.]\!]^t$ is a total function from concepts to subsets of $\mathcal{D}$ and from roles to subsets of $\mathcal{D} \times \mathcal{D}$, mapping concepts and roles to the set of their positive examples. Similarly, $[\![.]\!]^f$ is a total function from concepts to subsets of $\mathcal{D}$ and from roles to subsets of $\mathcal{D} \times \mathcal{D}$, but it maps concepts and roles to the set of their negative examples. The two functions have to satisfy the following equations:

For concept terms:

$$
\begin{array}{rcl}
[\![\top]\!]^t & = & \mathcal{D} \\
[\![\top]\!]^f & = & \emptyset \\
[\![C \sqcap D]\!]^t & = & [\![C]\!]^t \cap [\![D]\!]^t \\
[\![C \sqcap D]\!]^f & = & [\![C]\!]^f \cup [\![D]\!]^f \\
[\![\forall R.C]\!]^t & = & \{d_1 \in \mathcal{D} \mid \forall d_2 : (d_1, d_2) \in [\![R]\!]^f \vee d_2 \in [\![C]\!]^t\} \\
[\![\forall R.C]\!]^f & = & \{d_1 \in \mathcal{D} \mid \exists d_2 : (d_1, d_2) \in [\![R]\!]^t \wedge d_2 \in [\![C]\!]^f\} \\
[\![\exists R.\top]\!]^t & = & \{d_1 \in \mathcal{D} \mid \exists d_2 : (d_1, d_2) \in [\![R]\!]^t\} \\
[\![\exists R.\top]\!]^f & = & \{d_1 \in \mathcal{D} \mid \exists d_2 : (d_1, d_2) \in [\![R]\!]^f\} \\
[\![(rvm\ R\ S)]\!]^t & = & \{d_1 \in \mathcal{D} \mid \forall d_2 : (d_1, d_2) \in [\![R]\!]^f \vee (d_1, d_2) \in [\![S]\!]^t\} \\
[\![(rvm\ R\ S)]\!]^f & = & \{d_1 \in \mathcal{D} \mid \exists d_2 : (d_1, d_2) \in [\![R]\!]^t \wedge (d_1, d_2) \in [\![S]\!]^f\}
\end{array}
$$

For role terms:

$$
\begin{array}{rcl}
[\![P_1 \sqcap \ldots \sqcap P_k]\!]^t & = & [\![P_1]\!]^t \cap \ldots \cap [\![P_k]\!]^t \\
[\![P_1 \sqcap \ldots \sqcap P_k]\!]^f & = & [\![P_1]\!]^f \cup \ldots \cup [\![P_k]\!]^f \\
[\![(restr\ R\ C)]\!]^t & = & \{\langle d, e\rangle \in \mathcal{D} \times \mathcal{D} \mid \langle d, e\rangle \in [\![R]\!]^t \wedge e \in [\![C]\!]^t\} \\
[\![(restr\ R\ C)]\!]^f & = & \{\langle d, e\rangle \in \mathcal{D} \times \mathcal{D} \mid \langle d, e\rangle \in [\![R]\!]^f \vee e \in [\![C]\!]^f\}
\end{array}
$$

The subsumption algorithm is working in two phases. First, a concept is transformed into its canonical form where conjuncts of a concept are not themselves conjuncts, the concept C in $\forall R.C$ is not a conjunct, the second argument S in $(rvmRS)$ is an atomic role and conjuncts of a role are not themselves conjuncts. Here are the rules for transforming roles and concepts in 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. $((restr\ R_1\ C) \sqcap R_2) \longrightarrow (restr\ (R_1 \sqcap R_2)\ C)$

The second phase is the actual subsumption check. The concept $C_1 \sqcap \ldots \sqcap C_n$ subsumes $C'_1 \sqcap \ldots \sqcap C'_m$ if and only if for each i, $1 \leq i \leq n$, there is a j, $1 \leq j \leq m$, such that one of the following holds:

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 $C'_j = (rvmR'S)$ and R subsumes R' (S is a role name).

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

According to this semantics, the law of the excluded middle and modus ponens are not valid (see, [Patel-Schneider 1985]). Thus any reasoning requiring them in the standard two-valued semantics will fail here.

Example:

The soncept:

`human` $\sqcap \forall$ *friend*.`doctor` $\sqcap \forall$ `(restr` *friend* `doctor)`.$(\exists$ *speciality*.$\top)$

is not subsumed by

`human` $\sqcap \forall$ *friend*.$(\exists$ *speciality*.$\top)$

because it is possible that some friend is a doctor and is not a doctor in this semantics, but without a speciality and thus $\forall$ *friend*.$(\exists$ *speciality*.$\top)$ is false, but *friend*.`doctor` and $\forall$ `(restr` *friend* `doctor)`.$(\exists$ *speciality*.$\top)$ are not false.

The algorithm presented above is sound and complete with respect to four-valued semantics and it runs in time proportional to the square of the sum of the sizes of the two expressions. Thus, subsumption in this semantics is easily computable. With respect to the standard semantics this language is undecidable (see [Scmidt-Schauß 1989]).

3 Knowledge Interchange Format

Construction of a knowledge base is a time-consuming, full of errors and hard to reuse business. In most of the cases, knowledge represented in one system is impossible to be used in another system without a considerable effort of restructuring and redesigning. This effort is, sometimes, comparable with the effort of the construction of the original knowledge base itself. Thus, any facilities supporting (even partial) transfer of knowledge between different system and languages would be of great help. In the early 1990s the Knowledge Sharing Effort (KSE) was started in USA. The aim of the KSE was to be developed mechanisms and means to assist the sharing of knowledge bases among different research groups using different knowledge representation systems. Several developments were undertaken, including: (1) invention of a medium for knowledge transfer; (2) establishing of a methodology for knowledge transfer; (3) creation of basic ontological theory in different fields.

Some of the main ideas about the transfer of knowledge were borrowed from the area of machine translation. The central role is played by a knowledge representation language used as an interlingua between different knowledge representation languages and systems. This language is very powerful in order to be able to accommodate different kinds of knowledge and to incorporate different kinds of reasoning. This language is Knowledge Interchange Format (KIF). The language developed significantly with respect to its early versions. The currently actual version is presented in the draft proposed American National Standard (dpANS) NCITS.T2/98-004 — [Genesereth 1998]. This material is also based on [GF 1992] and [Perlis 1986].

KIF has the following essential features (from [Genesereth 1998]):

- The language has declarative semantics. It is possible to understand the meaning of expressions in the language without appeal to an interpreter for manipulating those expressions. In this way, KIF differs from other languages that are based on specific interpreters, such as Emycin and Prolog.
- The language is logically comprehensive — at its most general, it provides means of expression of arbitrary logical sentences. In this way, it differs from relational database languages (like SQL) and logic programming languages (like Prolog).
- The language provides means of representation of knowledge about knowledge. This allows the user to make knowledge representation decisions explicit and introduce new knowledge representation constructs without modifying the language.

There are two additional features taken into consideration when KIF was designed (to the extent possible while preserving the preceding):

- *Implementability.* Although KIF is not intended for use within programs as a representation or communication language, it should be usable for that purpose if desired.
- *Readability.* Although KIF is not intended primarily as a language for interaction with humans, human readability facilitates its use in describing representation language semantics, its use as a publication language for example knowledge bases, its use to assist humans when dealing with knowledge base translation problems, etc.

In order to meet the former requirements KIF was designed to be an extension of first order logic. First, we give some motivation for the different extensions and afterwards we present the language itself.

Self-reference and truth

The knowledge itself is a part of the world the knowledge is about. The expressions of the language used to represent the knowledge are also objects in world. In everyday life we very often use our language to make propositions about our knowledge or about the language itself. In many cases, we need to refer to the expressions of the language we use for knowledge representation and to explicate their properties. The main property of an expression we are interested in is its semantic relation with the other objects

in the world, that is its truth. In order we to be able to do this we need a naming convention for the expressions of the language and a predicate determining the truth of an expression. Again, it is very easy to represent paradoxes if the language allows an unrestricted version of self-reference.

Concrete domains

There are some domains in which there is an agreement about the relevant theories that described them. Some of these domains are very often needed for the representation of knowledge about other domains. Such a domain is number theory, for instance. Although most of the knowledge representation languages are capable to represent the necessary part of the number theory it is much more convenient to have it builtin the language than to represent it. There are several advantages of such a builtin theory. The most important is that the semantic interpretation of the theory is the same in every knowledge base. A minor one is that in an implementation more effective reasoning procedures for the concrete domain can be implemented than the general inference mechanism of the main language is.

KIF can be considered a first-order language in which all relations and functions are themselves objects in the domain of the interpretation, all the expressions of the language are objects in the domain of any KIF interpretation. KIF provides means to represent quoted and unquoted expressions of the language and it is possible to state properties of the expressions and especially their truth. In the last specification of KIF ([Genesereth 1998]) there are just two builtin theories about numbers, lists and strings.

3.1 Syntax

The syntax of KIF is computer-oriented and could be divided into three parts: characters, lexemes and expressions.

KIF uses ASCII characters classified as upper case letters (`A` | `B` | ... | `X` | `Y` | `Z`), lower case letters (`a` | `b` | ... | `x` | `y` | `z`), digits (`0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`), alpha characters (non-alphabetic characters that are used in the same way that letters are used) (`!` | `$` | `%` | `&` | `*` | `+` | `-` | `.` | `/` | `<` | `=` | `>` | `?` | `@` | `_` | `~`), special characters (`"` | `#` | `'` | `(` | `)` | `,` | | `^` | `‘`), white space (`space` | `tab` | `return` | `linefeed` | `page`), and other characters (every ASCII character that is not a member of one of the other categories). A normal character is either an upper case character, a lower case character, a digit, or an alpha character.

There are five types of lexemes in KIF – special lexemes, words, character references, character strings, and character blocks.

Each special character forms its own lexeme. It cannot be combined with other characters to form more complex lexemes (see [Genesereth 1998] for more on this).

A word is a contiguous sequence of (1) normal characters or (2) other characters preceded by the escape character (Escape characters are not discussed here.)

For the purpose of grammatical analysis, it is useful to categorize the class of words a little further, viz. as **variables**, **operators**, and **constants**. This categorization is disjoint and exhaustive. Every word is a member of one and only one category.

A variable is a word in which the first character is ? or @. A variable beginning with ? is called an **individual variable**. A variable beginning with an @ is called a **sequence variable**. Individual variables are used when quantifying over individual objects. Sequence variables are used when quantifying over sequences of objects.

Operators are used to form complex expressions of various sorts. There are three types of operators in KIF – **term operators**, **sentence operators**, and **definition operators**. Term operators are used when forming complex terms. There are five term operators: `value` | `listof` | `quote` | `if` | `cond`. Sentence operators are used when forming complex sentences. The following are the sentence operators: `holds` | `=` | `/=` | `not` | `and` | `or` | `=>` | `<=` | `<=>` | `forall` | `exists`. Definition operators are used in forming definitions. Here are the definition operators: `defobject` | `defunction` | `defrelation` | `deflogical` | `:=` | `:->` | `:<=` | `:=>`.

All other words are called constants. There are four categories of constants in KIF – **object constants**,

function constants, **relation constants**, and **logical constants**. Object constants are used to denote individual objects. Function constants denote functions on those objects. Relation constants denote relations. Logical constants express statements about the world and are either true or false. KIF is unusual among logical languages in that there is no syntactic distinction among these four types of constants; any constant can be used where any other constant can be used. The differences between these categories of constants is entirely semantic. Constants are also divided in **basic constants** and **non-basic constants**. The basic constants have fixed meaning in each KIF knowledge base.

There are three disjoint types of expressions in the language – **terms**, **sentences**, and **definitions**. Terms are used to denote objects in the world being described; sentences are used to express facts about the world; and definitions are used to define constants. Definitions and sentences are called forms. A **knowledge base** is a finite set of forms. The order of forms within the knowledge base is unimportant.

The following are the 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>))
```

In KIF the arity of function and relation constants are not syntactic features. They are treated semantically.

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>)
```

KIF definitions can be **complete**: they specify an expression defining a concept completely, or partial. The The **unrestricted** partial definitions are constraining a concept without necessarily giving a complete equivalence. On the other hand the "conservative" **partial** definitions are restricted in that their addition to a knowledge base does not result in the logical entailment of any additional sentences not containing the ¡constant¿ being defined.

3.2 Semantics

Similarly to the logic, the semantics of KIF is defined in terms of a **conceptualization** of the world and an interpretation connecting the syntax primitives of KIF with the elements of this conceptualization.

3.2.1 Conceptualization

The conceptualization determines the kinds of objects, functions and relations that exist in the world. A **universe of discourse** is the set of all objects presumed or hypothesized to exist in the world. KIF is **conceptually promiscuous** in that it does **not** require every user to share the same universe of discourse. On the other hand, KIF is **conceptually grounded**: every universe of discourse **is** required to include certain **basic** objects.

The following basic objects must occur in every universe of discourse.

- All ASCII Characters.
- Words.
- All complex numbers.
- All finite lists of objects in the universe of discourse.
- $\perp$ (pronounced "bottom") – a distinguished object that occurs as the value of various functions when applied to arguments for which the functions make no sense.

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

Functions and relations are sets of finite lists. Every function is a relation, but not every relation is a function. In a function, there cannot be two lists that disagree on only the last element. For every finite sequence of objects (called the *arguments*), a *function* associates a unique object (called the *value*). KIF allows functions and relations with variable arity, i.e. it is perfectly acceptable for a function or a relation to contain lists of different lengths. For example, the + function contains the lists $\langle 1, 1, 2\rangle$ and $\langle 1, 1, 1, 3\rangle$, reflecting the fact that the sum of 1 and 1 is 2 and the fact that the sum of 1 and 1 and 1 is 3. Similarly, the relation $<$ contains the lists $\langle 1, 2\rangle$ and $\langle 1, 2, 3\rangle$, reflecting the fact that 1 is less than 2 and the fact that 1 is less than 2 and 2 is less than 3.

3.2.2 Interpretation

An **interpretation** is a function i that associates the constants of KIF with the elements of a conceptualization. In order to be an interpretation, a function must satisfy the following two requirements.

First, the function must map constants to concepts of the appropriate type. O is a 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 $i(\sigma) \in \{true, false\}$.

Second, i must "satisfy" the axioms given in the document defining KIF ([Genesereth 1998]). Note that, even with these restrictions, KIF is only a "partially interpreted" language. Although the interpretations of some constants are constrained in the definition of the language, the meanings of other constants are left open.

The following are some of the axioms "built-in" KIF:

Every interpretation must map every numerical constant σ into the corresponding number n (assuming base 10).

$$i(\sigma) = n$$

Every interpretation must map the object constant `bottom` to $\perp$.

$$i(\texttt{bottom}) = \perp$$

Every interpretation must map the logical constant `true` into *true* and the logical constant `false` into *false*.

$$i(\texttt{true}) = true \text{ and } i(\texttt{false}) = false$$

3.2.3 Variable Assignment

A **variable assignment** v is a function that (1) maps individual variables $\mathcal{V}$ into objects in a universe of discourse O and (2) maps sequence variables $\mathcal{W}$ into finite sequences of objects. The notion of a variable assignment is important in defining the meaning of quantified terms and sentences.

3.2.4 Semantic Value

Given an interpretation and a variable assignment, we can assign a *semantic value* to every term in the language. We formalize this as a function s_{iv} from the set $\mathcal{T}$ of terms into the set O of objects in the universe of discourse.

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

$$s_{iv}(\nu) = v(\nu) \qquad \nu \text{ is an individual variable}$$

$s_{iv}(\sigma) = i(\sigma)$ σ is a constant

In the case of functional or relational constant, the semantic value is the corresponding set of lists. The semantic value of a functional term is obtained by applying the function denoted by the function constant in the term to the objects denoted by the arguments.

$s_{iv}((\pi\ \tau_1 \ldots \tau_n)) = i(\pi)[s_{iv}(\tau_1), ..., s_{iv}(\tau_n)]$ without a terminating sequence variable

$s_{iv}((\pi\ \tau_1 \ldots \tau_n\ \omega)) = i(\pi)[s_{iv}(\tau_1), ..., s_{iv}(\tau_n)|s_{iv}(\omega)]$ with a terminating sequence variable

$s_{iv}((\texttt{listof}\ \tau_1\ \ldots\ \tau_k)) = \langle s_{iv}(\tau_1), ..., s_{iv}(\tau_k)\rangle$

$s_{iv}((\texttt{listof}\ \tau_1\ \ldots\ \tau_k\ \omega)) = \langle s_{iv}(\tau_1), ..., s_{iv}(\tau_k)|s_{iv}(\omega)\rangle$

$s_{iv}((\texttt{quote}\ \epsilon)) = \epsilon$

$s_{iv}((\texttt{quote}\ (\epsilon_1\ \ldots\ \epsilon_n))) = s_{iv}((\texttt{listof}\ (\texttt{quote}\ \epsilon_1)\ \ldots\ (\texttt{quote}\ \epsilon_n)))$

$$s_{iv}((\texttt{if}\ \phi\ \tau_1)) = \begin{cases} s_{iv}(\tau_1) & t_{iv}(\phi) = true \\ \bot & \text{otherwise} \end{cases}$$

$$s_{iv}((\texttt{cond}\ (\phi_1\ \tau_1)\ \ldots\ (\phi_n\ \tau_n))) = \begin{cases} s_{iv}(\tau_1) & t_{iv}(\phi_1) = true \\ \ldots & \ldots \\ s_{iv}(\tau_2) & t_{iv}(\phi_n) = true \\ \bot & \text{otherwise} \end{cases}$$

3.2.5 Truth Value

The *truth value* for sentences in the language is defined as a function t_{iv} that maps sentences $\mathcal{S}$ into the truth values $true$ or $false$.

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

$t_{iv}(\lambda) = i(\lambda)$ λ is a logical constant

$$t_{iv}((\texttt{=}\ \tau_1\ \tau_2)) = \begin{cases} true & s_{iv}(\tau_1) = s_{iv}(\tau_2) \\ false & \text{otherwise} \end{cases}$$

$$t_{iv}((\texttt{/=}\ \tau_1\ \tau_2)) = \begin{cases} false & s_{iv}(\tau_1) = s_{iv}(\tau_2) \\ true & \text{otherwise} \end{cases}$$

$$t_{iv}((\rho\ \tau_1\ \ldots\ \tau_n)) = \begin{cases} true & \langle s_{iv}(\tau_1), ..., s_{iv}(\tau_n)\rangle \in i(\rho) \\ false & \text{otherwise} \end{cases}$$

$$t_{iv}((\rho\ \tau_1\ \ldots\ \tau_n\ \omega)) = \begin{cases} true & \langle s_{iv}(\tau_1), ..., s_{iv}(\tau_n)|s_{iv}(\omega)\rangle \in i(\rho) \\ false & \text{otherwise} \end{cases}$$

$$t_{iv}((\texttt{not}\ \phi)) = \begin{cases} true & t_{iv}(\phi) = false \\ false & \text{otherwise} \end{cases}$$

$$t_{iv}((\texttt{and}\ \phi_1\ \ldots\ \phi_n)) = \begin{cases} true & t_{iv}(\phi_j) = true \text{ for all } j\ 1 \leq j \leq n \\ false & \text{otherwise} \end{cases}$$

$$t_{iv}((\texttt{or}\ \phi_1\ \ldots\ \phi_n)) = \begin{cases} true & t_{iv}(\phi_j) = true \text{ for some } j\ 1 \leq j \leq n \\ false & \text{otherwise} \end{cases}$$

$$t_{iv}((\texttt{=>}\ \phi_1\ \ldots\ \phi_n\ \phi)) = \begin{cases} true & \text{for some } j\ t_{iv}(\phi_j) = false \text{ or } t_{iv}(\phi) = true \\ false & \text{otherwise} \end{cases}$$

$$t_{iv}((\texttt{<=}\ \phi\ \phi_1\ \ldots\ \phi_n)) = \begin{cases} true & t_{iv}(\phi) = true \text{ or for some } j\ t_{iv}(\phi_j) = false \\ false & \text{otherwise} \end{cases}$$

$$t_{iv}((\texttt{<=>}\ \phi_1\ \phi_2)) = \begin{cases} true & t_{iv}(\phi_1) = t_{iv}(\phi_2) \\ false & \text{otherwise} \end{cases}$$

A variable assignment v' is a *version* of variable assignment v with respect to variables ν_1, ..., ν_n if and only if v' agrees with v on all variables except for ν_1, ..., ν_n. The assignments for ν_1, ..., ν_n can be the same as those in v or can be completely different. In the next two equations, v' is a version of v with respect to variables in the first argument.

$$t_{iv}(\texttt{(exists (}\nu_1\ \ldots\ \nu_k\ \omega\texttt{)}\ \phi\texttt{)}) = \begin{cases} true & \exists v'\ t_{iv'}(\phi) = true \\ false & \text{otherwise} \end{cases}$$

$$t_{iv}(\texttt{(forall (}\nu_1\ \ldots\ \nu_k\ \omega\texttt{)}\ \phi\texttt{)}) = \begin{cases} true & \forall v'\ t_{iv'}(\phi) = true \\ false & \text{otherwise} \end{cases}$$

3.2.6 Logical Entailment

An interpretation i **logically satisfies** a sentence ϕ if and only if the truth value of the sentence is *true* for all variable assignments. Whenever this is the case, we say that i is a *model* of ϕ. Extending this notion to sets of sentences, we say that an interpretation is a model of a set of sentences if and only if it is a model of every sentence in the set of sentences.

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

3.2.7 Definitions

The definitional operators in KIF allow us to state sentences that are true "by definition" in a way that distinguishes them from sentences that express contingent properties of the world. Definitions have no truth values in the sense described above. They are so because we say that they are so.

On the other hand, definitions have content – sentences that allow us to derive other sentences as conclusions. In KIF, every definition has a corresponding set of sentences, called the *content* of the definition. The rules for determining the content of a definition are slightly complicated and unfortunately are not described anywhere for the current version of the language. The following is a brief outline, sufficient to enable the reader to understand the use of definitional constructs in the intervening chapters.

The `defobject` operator is used to define objects. The two simplest forms are shown below, together with their content. In the first case, the content is the equation involving the object constant in the definition with the defining term. In the second case, the content is the conjunction of the constituent sentences.

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

The `deffunction` operator is used to define functions. Again, the two simplest forms are shown below, together with their defining axioms. In the first case, the content is the equation involving (1) the term formed from the function constant in the definition and the variables in its argument list and (2) the defining term. In the second case, as with object definitions, the content is the conjunction of the constituent sentences.

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)

The `defrelation` operator is used to define relations. The two simplest forms are shown below, together with their defining axioms. In the first case, the content is the equivalence relating (1) the relational sentence formed from the relation constant in the definition and the variables in its argument list and (2) the defining sentence. In the second case, as with object and function definitions, the content is the conjunction of the constituent sentences.

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)

3.3 Lists

A *list* is a finite sequence of objects in a universe of discourse. The term (`listof` τ_1 ... τ_k) is used to denote the list of objects denoted by τ_1, ..., τ_k. For example, the following expression denotes the list of an object named `mary`, a list of objects named `tom`, `dick`, and `harry`, and an object named `sally`.

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

In the original document ([Genesereth 1998]) an comprehensive list of objects, functions and relations defining the theory of lists is given. Here we present only a few of them.

The relation `list` is the type predicate for lists. An object is a list if and only if there is a corresponding expression involving the `listof` operator.

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

The object constant `nil` denotes the empty list. `null` tests whether or not an object is the empty list.

```
(defobject nil := (listof))

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

The value of `subst` is the object or list obtained by substituting the object supplied as first argument for all occurrences of the object supplied as second argument in the object or list supplied as third argument.

```
(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)))|
```

where the function `cons` adds the object specified as its first argument to the beginning of the list specified as its second argument:

```
(deffunction cons (?x ?l) :=
  (if (= ?l (listof @l)) (listof ?x @l)))
```

3.4 Metaknowledge

3.4.1 Naming Expressions

To formalize knowledge about knowledge, KIF uses a conceptualization in which expressions are treated as objects in the universe of discourse and in which there are functions and relations appropriate to these objects. In KIF conceptualization, words are treated as primitive objects (i.e. having no subparts). Complex expressions are conceptualized as lists of sub-expressions. Every complex expression is regarded as a list of its immediate sub-expressions. For example, the sentence `(not (p (+ a b c) d))` is a list consisting of the operator `not` and the sentence `(p (+ a b c) d)`. This sentence is treated as a list consisting of the relation constant `p` and the terms `(+ a b c)` and `d`. The first of these terms is a list consisting of the function constant `+` and the object constants `a`, `b`, and `c`.

In order to assert properties of expressions in the language, one needs a way of referring to those expressions. There are two ways of doing this in KIF.

One way is to use the `quote` operator in front of an expression. For example, the following sentence ascribes to the individual named `john` the belief that the moon is made of a particular kind of blue cheese.

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

Since expressions are first-order objects, one can quantify over them, thereby asserting properties of whole classes of sentences.

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

The second way of referring to expressions is KIF is to use the `listof` function. For example, a 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)`.

The advantage of the `listof` representation over the `quote` representation is that it allows us to quantify over parts of expressions.

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

While the use of `listof` allows us to describe the structure of expressions in arbitrary detail, it is somewhat awkward. Rather than using the `listof` function constant as described above, one writes the expression preceded by the up-arrow character `^` and add a comma character `,` in front of any sub-expression that is not to be taken literally.

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

3.4.2 Formalizing Syntax

In order to facilitate the encoding of knowledge about KIF, the language includes type relations for the various syntactic categories defined in the syntax of the language. We give here only a few of these relations.

For every individual variable ν, there is an axiom asserting that it is indeed an individual variable. Each such axiom is a defining axiom for the `indvar` relation.

`(indvar (quote` ν`))`

For every sequence variable ω, there is an axiom asserting that it is a sequence variable. Each such axiom is a defining axiom for the `seqvar` relation.

`(seqvar (quote` ω`))`

The relations `word` and `sentence` are defined as follow:

```
(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)))
```

3.4.3 Changing Levels of Denotation

This section presents some vocabulary that allows one to change levels of denotation, i.e. to relate expressions about expressions with the expressions they denote.

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. The standard name for an expression τ is `(quote` τ`)`; the standard name for a non-expression is at the discretion of the user.

The final level-crossing vocabulary item is the relation constant `wtr` ("weakly true"). For example, one can say that a sentence of the form `(=> (p ?x) (q ?x))` is true by writing the following sentence.

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

The advantage of this meta notation becomes clear when one needs to quantify over sentences, as in the encoding of axiom schemata. For example, one can say that every sentence of the form `(=>` ϕ ϕ`)` is true with the following sentence.

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

Semantically, one would like to say that a sentence of the form `(wtr '`ϕ`)` is true if and only if the sentence ϕ is true. In other words, for any interpretation and variable assignment, the truth value t_{iv}(`(wtr '`ϕ`)`) is the same as the truth value $t_{iv}(\phi)$. Unfortunately, this causes serious problems. Equating a truth function with the meaning it ascribes to `wtr` quickly leads to paradoxes. The English sentence "This sentence is false." illustrates the paradox. In KIF this sentence can be written as shown below. The sentence, in effect, asserts its own negation.

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

Fortunately, we can circumvent such paradoxes by slightly modifying the proposed definition of `wtr`. In particular, we have the following axiom schema for all p that do not contain any occurrences of `wtr`. For all p that do contain occurrences, `wtr` is false.

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

With this modified definition, the paradox described above disappears, yet we retain the ability to write virtually all useful axiom schemata as metalevel axioms.

Levels of representation

Standard logic considers the propositions about the world to be ordered in levels: on the basic level only atomic propositions can be expressed about individuals in the world; the next level allows representation of functions and predicates over individuals to be expressed (first-order logic), the next level adds functions and predicates over functions and predicate over objects (second-order logics) and so on. In this layered world one can speak on a given level about the elements of the lower levels. This approach insures some very important features of the corresponding logics (especially for first-order logic) and avoids some of well known paradoxes such as Russel paradox about the barber who gives a shave to everybody who doesn't shave himself and the question is whether this barber shaves himself or not. In any case there is a contradiction.

Unfortunately, when one wants to represent knowledge about the knowledge of an agent or agent's knowledge about the knowledge of some other agents there is a problem with the levels. Let us consider the following situation: Kiril is believing everything that Paul is saying. On which level we have to represent this proposition. If Paul expresses propositions only in first order logic, then we have to use second-order logic, but sometimes Paul is making claims in second-order logic then we have to use third-order logic in order to express that Kiril believes these propositions also. Thus, we have to use an infinite chain of propositions in order to express such a simple fact. One solution is to use a logic that considers entities in the world on one level. In such a logic we can do propositions about and quantify over objects, functions and predicates in uniform way regardless of their order. In order to define such a logic one has to include a mechanism that prevent the definitions of paradoxes similar to the paradox described above.

3.5 Conformance

3.5.1 Introduction

KIF is a highly expressive language. For many, this is a desirable feature; but there are disadvantages. One disadvantage is that it complicates the job of building fully conforming systems. Another disadvantage is that the resulting systems tend to be "heavyweight" (i.e. they are larger and in some cases less efficient than systems that employ more restricted languages).

In order to deal with these problems, the KIF committee in the Fall of 1997 voted to augment the basic language specification with a set of "conformance dimensions". These dimensions are not the same as the "conformance levels" of other languages. Rather, each conformance dimension has a variety of levels within that dimension.

A "conformance profile" is a selection of alternatives from each conformance dimension. System builders are expected to make choices for each dimension and and then ensure that their systems adhere to the resulting comformance profile. Systems are expected to use the terminology defined here to share information about their conformance profile with other systems (in a protocol-specific manner).

Although this conformance profile scheme is more complex than one based on conformance levels, it accommodates varying capabilities and/or computational constraints while providing a migration path from more restrictive to more expressive.

3.5.2 Conformance Dimensions

Introduction

A conformance dimension is a classification of KIF sentences into conformance categories on the basis of a single syntactic criterion. (For example, the quantification dimension provides two categories, quantified KIF and unquantified KIF, based on whether or not a conforming knowledge base contains quantifiers.)

Logical Form

The first conformance dimension concerns logical form. There are five basic categories: atomic, conjunctive, positive, logical, and rule-like. Rule-like knowledge bases are further categorized as Horn or non-Horn and recursive or non-recursive.

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.

The dependency graph for a rule system is a graph whose nodes are the constants in relational position. There is an edge from the node for a given relation constant p to the node of relation constant q if and only if p appears in the body of a rule whose head predicate is p.

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.

3.5.3 Common Conformance Profiles

A conformance profile is a selection of alternatives for each conformance dimension. Given the dimensions and categories defined in the preceding section, it is possible to define a large number of profiles. A single system may use different profiles in different types of communication. In particular, it is common to use one profile for assertions and another for queries. The following paragraphs define a few common types of systems with their corresponding 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 Horn system (e.g. pure Datalog) is one in which (1) all assertions are rules that are Horn, unquantified, and baselevel and (2) all queries are positive, non-recursive, unquantified, and baselevel.

A relational system is one in which (1) all assertions are rules that are simple, unquantified (but may be non-Horn and non-recursive), and baselevel and (2) all queries are logical, non-recursive, 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.

3.5.4 Dealing with Differences in Conformance Profiles

The existence of multiple conformance profiles raises the question of what happens when systems with different profiles must communicate.

Whenever the conformance profile of a receiver is known, a sender should avoid sending expressions that fall outside the receiver's conformance profile.

Unfortunately, this rule cannot be enforced in all situations. In some cases, conformance information

about receivers is unavailable; and, even when conformance information is available, it may be desirable to send a message that falls outside a receiver's profile, e.g. it may be most efficient for a sender to broadcast a single knowledge base to a large number of receivers with differing conformance profiles rather than sending different knowledge bases to each receiver.

Whenever a receiver receives a non-conforming expression, it is free to ignore the expression, even though it may be able to make sense of portions of that expression. If the receiver ignores a non-conforming expression and the sender requests a reply, the receiver should report a failure.

4 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 Pratchet, Truckers)

There are many tasks that can be better resolved employing the resources of multiple computers. This is just the point for using more "computational" resources in paralell. In the same time, there is another reason for trying to distribute the computation into multiple, somehow independent processes: just to make the complex things simpler and manageable by separation into multiple modules that can be developed, tested, managed and maintained separately. That is why it is quite popular to have a number of virtual (or real) processes even running on a single machine.

In the area of distributed computation there are many overlaping and ambiguous terms. However, one simple view is that any process (a program running on a computer) can be considered like an *agent*. These agents are being frequently characterized as "computational", "artificial", "intelligent", etc. The part of the AI that studies this point of view is refered like *Distributed AI (DAI)*.

Once having the thinks simplified by distributing the tasks between multiple objects, we should take care for the coordination between them. Following the analogy with the human agents, multy-agent systems can be seen like some kind of society. Here we are presenting KQML like one of the possible means to organize the communication between the agents in the scociety.

4.1 Communication between agents

The interaction between the agents can be studied like a combination of:

- **interaction protocol** — the strategy of the agent about its relations with the society. For example: negotiation strategies, playing strategies, "each time when you do not know something, find somebody else and ask him";
- **communication language** — the expressive mean (or mediator) for message exchange between the agents. It should provide possibilities for characterising the messages according to their pragmatics (or intention). Sample pragmatics: assertion, query, advertisement.
- **transport protocol** — the media of the communication. It can be TCP, SMTP, RMI, etc.

The communication language should provide means for negotiation (or alignment) of:

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

Pragmatics negotiation should care for the following aspects:

- 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?

Communication language is also responsible for the negotiation of:

- the structure of the society, i.e. the architecture of the distributed system;
- the structure of the conversation.

There can be many different approaches and variations related with the organization of the communication. The most general determiner is the compromise between how "open" will be the convention (the communication language) and how complex (and/or slow) will be to support it. Bellow we just present the approach accepted by the creators of KQML that is pretty "open" textual notation for communication.

4.2 KQML history and philosophy

4.2.1 Short history

KQML agent communication language being developed by the external interfaces working group of the DARPA Knowledge Sharing Effort. KQML is intended to be a high-level language to be used by knowledge-based system to share knowledge at run time.

Its specification have been first published in 1993. The last proposal for new specification of the language ([Labrou and Finin,97]) was published in 1997-th. More informa about the language and related materials can be found on the following address: `http://http://www.cs.umbc.edu/kqml/`

4.2.2 Philosophy

KQML is a high-level *communication language*, as well, as a protocol for information exchange that is independent from the syntax and ontology of the messages. It implements the approach to envision the information exchange like *unified access to a* KB. The third important point is that *KQML* gives an *extendable set of pragmatics* for classification of the believes and intentions related with a chunk of information.

The communication language steps on the transport protocol. Here are the assertions of *KQML* for the underlying protocol:

- the agents are related with one-way communication links that can transfer discrete messages (package- rather than stream-oriented);
- the communication can introduce delays;
- the receiver knows the delivery connection for each message;
- First-Sent-First-Received — the messages sent in one and the same direction are being received in the order of sending;
- message delivery is reliable.

KQML primitives are called **performatives**. These are speech acts of some types that determine the actions that the agents can use to communicate. The type of the performative borrows the *message pragmatics*, that determines the intention and the protocol of the communication. Sample pragmatics: ask, tell, achieve, reply, deny, monitor, forward, subscribe, evaluate, etc. Further on, each message is characterised by attribute-value pairs. Such attributes are: content, language, topic, ontology, sender, reply-with.

It is considered that each *KQML* agent has a virtual *KB*with two constituents:

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

So, the most important performatives give basic access two this arrays:

- **tell** — addition to the information array;
- **untell** — removal from the information array;
- **achieve** — goal addition;
- **unachieve** — goal removal.

There are there kinds of performatives: core, standard and extended. The *core* performatives are small set of mandatory performatives. The *standard* (or reserved) performative are those that are defined in the *KQML* specification, but they are not mandatory. Finally, the *extended* performatives are all other performatives that can be defined and implemented in specific scocieties.

4.2.3 Facilitators

In a *KQML* agent there can be (in fact, there should be) agents who's only or main responsibility is to support the communication between the others. Such kind of agents are named *facilitators* and they can provide one or more of the following services:

- mapping between the physical address and the symbolic names of the agents;
- support of a registry of the services that the other agents are willing to provide or use;
- communication services like routing, connecting, etc.

4.2.4 *KQML* is quite "open"

One of the central goals of the creators of *KQML* was to make it as "open" as possible. Here some of the dimensions of this flexibility:

- the transport protocol is not fixed. Any protocol that fulfills the requirements is welcome;
- there are no limitation about the formal language used for encoding of the message content. The only technical requirement is that it should be representable as a plain text with a finite length. The default representation is *KIF*;
- there are no limitation for the ontology. It should be just declared exactly which ontology is used. Even there is no default ontology proposed;
- the architecture of the *KQML* society is not restricted. As well, there are no limitations for the type and behaviour of the agents. The existence of at least one facilitator is suggested, but not mandatory;
- new types of performatives can be added;
- new parameters for the existing performatives can be also added

In both cases (for new performatives and parameters) the limitation is that the *reserved* ones should not be used for other purposes.

4.3 *KQML* syntax

The syntax if the performatives (messages) in *KQML* is a restriction on the

ASCII representation of Common Lisp Polish-prefix notation. The general structure of the performative is following:

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

The full syntax can be found in [Labrou and Finin,97].

There are two important limitations on the syntax of the value of the `:content` parameter. The content should be in the syntax of the language which name is the value of the `:language` parameter. Also the constant symbols used (the names) should be described in the ontology referred by the `:ontology` parameter.

4.4 *KQML* semantics

The semantics of *KQML* is defined in terms of the changes of the state of the VKB of the agent receiving the message. However, there is no formal description of this semantic.

Here follows a list of the *reserved paramater* names and their meanings:

Name	Value	Explanation
`language`	`word`	the name of the language of representation of the content
`ontology`	`word`	the name of the ontology used in the content
`content`	`expression`	the specific information cared by the message
`sender`	`word`	the actual sender
`receiver`	`word`	the actual receiver
`from`	`word`	the starting point of the performative that is a content of a `forward` performative
`to`	`word`	the target of the performative that is a content of a `forward` performative
`in-reply-to`	`expression`	in response query this the value of this parameter should be the same like `:reply-with` of the previous query
`reply-with`	`expression`	the label that will be expected in `in-reply-to` of the response query
`force`	`word`	if the value is `permanent`, the agent promises that he will not cancel this performative. The default is `tentative`. (with `deny`)

The list of reserved performatives can be found in [Labrou and Finin,97].

4.5 Open questions

1. Is it acceptable in one VKB to keep expressions in different languages?

2. If the content of the messages is always translated, what happens with their identity?
Imagine that the agent A have **tell**ed to agent B that he believes in the fact Fa. Let's have the agent B translated the fact to Fa_{Lb} in its own language Lb. What should happen if A stops to support this believe (by **untell**). How should B decide to retract Fa_{Lb} from its VKB?

References

[BBH+ 1990] F. Baader, H.-J. Bürckert, J. Heinsohn, B. Hollunder, J. Müller, B. Nebel, W. Nutt, H.-J. Profitlich. *Terminological Knowledge Representation: A Proposal for a Terminological Logic.* Technical Memo RR-90-04, DFKI, Kaiserslautern, Germany, 1990.

[BH 1991] Franz Baader and Bernhard Hollunder. *A terminological knowledge representation system with complete inference algorithm.* in Proc. of the Workshop on Processing Declarative Knowledge, PDK-91, Lecture Notes in Artificial Intelligence. Springer Verlag, 1991.

[BHN+ 1993] F. Baader, B. Hollunder, B. Nebel, H.-J. Profitlich, E. Franconi. *An Empirical Analysis of Optimization Techniques for Terminological Representation Systems or: 'Making KRIS get a move on'.* Research Report RR-93-03, DFKI, Saarbrücken, Germany, 1993.

[Brachman 1979] Ronald J. Brachman. *On the Epistemological Status of Semantic Networks.* In N. V. Findler (Ed.), Associative Networks: Representation and Use of Knowledge by Computers. Academic Press, 1979.

[BPL 1985] Ronald J. Brachman, Victoria Pigman Gilbert, and Hector J. Levesque. *An essential hybrid reasoning system: Knowledge and symbol level accounts in KRYPTON.* In Proc. of the International Joint Conference on Artificial Intelligence. pages 532-539, Los Angeles, California, 1985.

[BS 1985] Ronald J. Brachman and James G. Schmolze. *An overview of the Kl-One knowledge representation system.* Cognitive Science, 9(2):171-216, 1985.

[BBM+ 1989] Alexander Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. *CLASSIC: a structural data model for objects.* in ACM SIGMOD, 1989.

[BDS 1993] Martin Buchheit, Francesco M. Donini, Andrea Schaerf. *Decidable Reasoning in Terminological Knowledge Representation Systems.* Technical Report RR-93-10, DFKI, Kaiserslautern, Germany, 1993.

[Davis, et. al 1996] Randall Davis, Howard Shrobe and Peter Szolovits. *What is Knowledge Representation?* AI Magazine, 14(1):17-33, 1993.

[Ellis 1991] Gerard Ellis. *Compiled hierarchical retrieval.* In proc. of the 6th Annual Conceptual Graphs Workshop, 1992.

[Franconi 1990] Enrico Franconi. *The YAK (Yet Another Karphagen) manual.* IRST - Manual 9003-01, Povo TN, Italy, 1990.

[Franconi, et. al 1992] Enrico Franconi. Bernardo Magnini and Oliviero Stock *Prototypes in a Hybrid Language with Primitive Descriptions.* Computers Math. Applic. Vol. 23, No. 6-9, pp 543-555, Pergamon Press, 1992.

[Genesereth 1998] Genesereth, Michael R., and Richard Fikes, eds. *Knowledge Interchange Format.* draft proposed American National Standard (dpANS). NCITS.T2/98-004, `http://logic.stanford.edu/kif/`

[GF 1992] Genesereth, B. and Richard E. Fikes. *Knowledge Interchange Format Version 3.0* Reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, June 1992. "Living document" of the Interlingua Working Group of the DARPA Knowledge Sharing Effort.

[Guarino 1994] Nicola Guarino. *The Ontological Level.* in R. Casati, B. Smith and G. White (eds.), Philosophy and the Cognitive Sciences, Vienna: Hülder-Pichler-Tempsky, 1994.

[HN 90] Bernhard Hollunder and Werner Nutt. *Subsumption Algorithms for Concept Languages.* Technical Report RR-90-04, DFKI, Kaiserslautern, Germany, 1990.

[KBR 1986] Thomas S. Kaczmarek, Raymond Bates, and Gabriel Robins. *Recent developments in NIKL.* in Proc. of the 5th National Conference on Artificial Intelligence AAAI-86, pages 978-985, 1986.

[Kramer, Mylopoulos, 1992] Kramer, Bryan and Mylopoulos John, *Knowledge Representation.* In: Encyclopedia of Artificial Intelligence, S.C. Shapiro, ed., John Wiles & Sons, New York, 1992, pp.743-759.

[MB 1987] Robert M. MacGregor and Raymond Bates. *The Loom knowledge representation language.* Technical Report ISI/RS-87-188, University of Southern California, Information Science Institute, Marina del Rey, California, 1987.

[Labrou and Finin,97] Labrou, Yannis and Finin, Tim. *A proposal for a new KQML specification*, TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD 21250, 1997, `http://www.cs.umbc.edu/ jklabrou/publications/tr9703.ps`

[Levinson 1984] Robert Levison. *A self-organizing retrieval system for graphs.* In proc. of the 3rd National Conference of the American Association for Artificial Intelligence, pages 203-206, Austin, Texas, 1984.

[Levinson 1992] Robert Levison. *Pattern associativity and the retrieval of semantic networks.* Jurnal of Computers & Mathematics with Applications, 23(6-9), 573-600, 1992.

[Lipkis 1982] Thomas Lipkis. *A Kl-One classifier.* In J.G.Schmolze and R.J.Brachman, editors, Proc. of the 1981 Kl-One Workshop, pages 128-145, BBN Report No. 4842. Cambridge, MA, 1982.

[MacGregor 1988] Robert M. MacGregor. *A deductive pattern mather.* In proc. of the 7th National Conference of the American Association for Artificial Intelligence, pages 403-408, Saint Paul, MI, August 1988.

[QK 1990] Joachim Quants and Carsten Kindermann. *Implementation of the BACK system version 4.* Technical report KIT-Report 78, FB Informatik, Technische Universität Berlin, Berlin, Germany. 1990.

[Patel-Schneider 198?] Peter F. Patel-Schneider. *A Four-Valued Semantics for Frame-Based Description Languages.*

[Patel-Schneider 1985] Peter F. Patel-Schneider. *A Decidable Firsat-Order Logic for Knowledge Representation.* In Proc. IJCAI-85, pages 455-458, 1985.

[Perlis 1986] Donald Perlis. *Language with Self-Reference I. Foundations (or — We can have everything in first-order logic!)*

[Poole et. al. 1998] David Poole, Alan Mackworth and Randy Goebel. *Computational Intelligence - A Logical Approach.* Oxford University Press, New York, USA, 1998.

[SSS 1988] Manfred Scmidt-Schauß and Gert Smolka. *Attributive Concept Descriptions with Unions and Complements.* IWBS Report 68, IBM - Deutschland GmbH, Stuttgart, Germany, 1988.

[Scmidt-Schauß 1989] Manfred Scmidt-Schauß. *Subsumption in KL-ONE is undecidable.* In proc. of the First International Cobference on Knowledge Representation and Reasoning. 1989.

[von Luck, et. al 1987] Kai von Luck, Bernhard Nebel, Christof Peltason, and Albrecht Schmiedel. *The Anatomy of the BACK System.* Technical report KIT-Report 41, FB Informatik, Technische Universität Berlin, Berlin, Germany. 1987.