F. Dignum and R.P. van de Riet. Knowledge base modelling based on linguistics and founded in logic. *Data & Knowledge Engineering*, 1991.

Knowledge Base Modelling Based on Linguistics and Founded in Logic

F.Dignum†
R.P. van de Riet

Free University, Amsterdam

ABSTRACT

Knowledge Base Systems should in principle be able to store and manipulate any sort of knowledge, including vague (indefinite) knowledge, knowledge about events and obligations and knowledge about temporal aspects.
In this paper a language, CPL, is introduced in which all these different kinds of knowledge can be expressed. This language is based on the linguistic theory of Functional Grammar. The semantics of this language is based on a logical interpretation of the language structures, for which we use five different forms of logic: predicate, modal, deontic, dynamic and temporal logic.
Inferences, based on these logics, that can be made within this language, are shown.

Keywords: Knowledge Bases, Linguistics, Modal logic, Deontic logic, Dynamic Logic, Temporal Logic.

1. Introduction

The word "knowledge base" has been introduced in computer science in the beginning of the eighties. There are mainly two fields in which the word occurs. The first is databases and the second is artificial intelligence. Although the word is widely used at this moment, there is, unfortunately, no consensus on the definition of this term.
Before we give our definition of knowledge bases and knowledge base systems, we will first give an overview of some different views on knowledge bases.

In the field of databases , a knowledge base is usually seen as an extension of a database. The first definition of the term was probably given by G. Wiederhold in [43]. Here a knowledge base is defined as "a system where information or knowledge about data is used together with the data themselves". The difference between knowledge and data is not made formal, but a litmus test for distinguishing knowledge and data is given.

† Current address: University of Swaziland, P/B Kwaluseni, Swaziland.

If you would permit a clerk to update something it is probably data.
If you look for an expert to update it then it is probably knowledge.

As an example, the comparison with Prolog is made. The ground rules found in a Prolog program are usually data, while the inferencing rules are usually knowledge.

This comparison with Prolog indicates that the knowledge base, in this view, is a further step to enhance the intelligence of the database systems.
This viewpoint of knowledge bases is very similar to that of deductive databases [18]. In deductive database systems the database theory is combined with the theory of logic programming [4, 12, 24, 23]. In a deductive database both data and simple rules can be storedU, usually in the form of logical formulas. By using the inference mechanisms developed in logic programming (e.g. resolution), new data can be inferred from the data that is stored in the database.

In the above approaches to knowledge bases the emphasis is put on what information can be stored explicitly and what information can be derived, with the implicit assumption that the knowledge consists of facts and rules (about the facts).
In the Artificial Intelligence approach the emphasis is put on the knowledge representation aspect. That is, what is knowledge and how can it be represented? A knowledge base system is then a system in which knowledge is represented (therefore a system containing a knowledge base is sometimes called a knowledge *based* system).

A good definition of what a knowledge representation system should embody is given by Smith in [39]. In this paper he states the knowledge representation hypothesis:

> Any mechanically embodied intelligent process will be comprised of structural ingredients that a) we as external observers naturally take to represent a *propositional* account of the knowledge that the overall process exhibits, and b) independent of such external semantic attribution, play a formal but *causal* and essential role in engendering the behaviour that manifests that knowledge.

Bachman and Levesque explain in [5] this hypothesis further for knowledge bases as follows.
There are two major properties that the structures in a knowledge base should satisfy. First, it should be possible to interpret the structures as propositions. That is, they have to have a truth value that reflects some property of how the world that is modelled by the knowledge base looks like.
The importance of this requirement resides in the fact that it demands that the knowledge base structures have some sort of underlying truth theory.
The second property of the structures is that they must play a causal role in the system as opposed to, for instance, comments in a program. The system must behave in a way which reflects that certain knowledge is present. Or, more simply, the system must be able to *use* the knowledge.

Note that nothing is said about what constitutes "knowledge". Using the above hypothesis, a database system is also a knowledge base system, although a very simple one, in which the knowledge consists of simple data.
However, from the database perspective, we would like to know what distinguishes a knowledge base from a normal database.

Reiter combines the two views. He defines a knowledge base as follows:

> A knowledge base is a representation of the truths about the specific and general facts of the world being modelled [33].

This definition of a knowledge base complies with the knowledge representation hypothesis stated above. It also distinguishes the knowledge base from a database by specifying that it also contains the truths about general facts. (whereas a database contains only truths about specific facts.)

The above definition comes closest to our own view on knowledge bases. We will, however, be a bit more specific about what we see as "specific" and as "general" facts.

The definition of a knowledge base and a knowledge base system, as they will be used in this paper, is given in the next section. Also some consequences for the knowledge representation language that follow from this definition will be discussed in that section.

In section 3, the language for modelling knowledge bases, CPL, which is an abbreviation for Conceptual Prototyping Language, is introduced with an example. The example is meant to show that different kinds of knowledge can indeed be described using CPL. It is however not meant to serve as a full introduction to all aspects of the language. The language itself is the subject of [13], in which the underlying linguistic ideas are described together with the syntax of the language. In [15], the language is extended and given a formal semantics in terms of (five different forms of) logic. The essence of this paper is to describe these semantics, but in a much more concise form, without proofs.

Some aspects of CPL deserve special attention. The first is the fact that the traditional "constraints" of the knowledge base get a deontic meaning. That is, if the constraints are violated the knowledge base is not inconsistent, but only in an undesirable state. It is, furthermore, possible to define how such an undesirable state can be left again. The semantics of the deontic constraints and other modalities are described in section 4.

In section 5, we discuss the dynamic properties of CPL, in particular the combination of the dynamic and deontic features. This leads to permitted and obliged actions, which are important for any flexible information system.

In section 6, the temporal aspects are discussed. In section 7, we show the logical interpretations of the examples of section 3 and show that the inference made in CPL in that section is sound. Finally, some conclusions are drawn.

2. Knowledge Bases

2.1. The Knowledge Base System

We define a knowledge base system as follows

Definition:

> A knowledge base system is a system that maintains a source of information (the knowledge base) in a way such that a user can communicate with the system as if he communicates with another user having access to that information.

Of course, this definition is so general that it does not give a clear idea about what components a knowledge base system should consist of. Therefore, in the following figure, an overview is given of the components that (in our view) a knowledge base system (KBS) should (minimally) contain to fulfil the above definition.

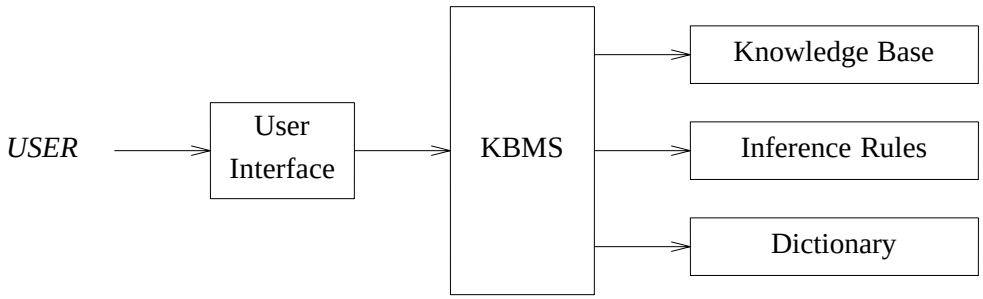

figure 1.

We will now describe each of the components of the KBS indicated in the figure.

The User Interface

Although in the figure the user interface is indicated as only one component, in practice different user interfaces for different users are possible.

The user interface is not further discussed in this paper. However, it should preferably contain a natural language interface, which could be combined with a graphical interface.

The natural language interface ensures that a natural form of communication between the user and the KBS is established. The graphical interface is then used to provide an abstract overview of some part of the knowledge present in the KBS.

Although the user interface is not a part of this paper, we described in [10] that it is possible to construct a natural language parser that takes English as input and that gives structures of the language described in this paper as output.

The Knowledge Base Management System

Just like in database systems, the KBMS manages the other components. The following activities can be distinguished. First, it handles the queries of the user (interface) by using the knowledge base and the inference rules. Secondly, it handles the updates on the knowledge base. Finally, it maintains the integrity of the knowledge base, the dictionary and the inference rules. In multi-user environments it would, of course, also take care of security and of concurrency control.

The inference rules

The inference rules are actually a set of meta-rules, which determine which inferences can be made using the knowledge present in the knowledge base. If the knowledge base were to be represented in first-order logic, the inference rules would contain e.g. the Modus Ponens, but might also contain the rules that govern the resolution process. These rules determine how the knowledge from the knowledge base can be *used* by the system.

It is important to note that these rules are different from the ones meant in the definition of knowledge bases from the database perspective. There the rules have the function of constraints or formulas, which can be used in the inferencing process.

Because we must be able to make inferences about the knowledge in the knowledge base using the inference rules, we have to show that the inferences can be defined in terms of the language which describes the knowledge in the knowledge base.

Although the inference rules are not a main topic of this paper, we will show that this is indeed

possible with the language described in this paper.

The Dictionary

In the dictionary, definitions are given about the elements of the language with which the knowledge base is described. These definitions limit the possible expressions to meaningful ones.
For instance, in the dictionary it will be stipulated that the "agent" of the action "sell" is always a person, while the "goal" of this action is always non-human. That is, the thing that sells something is always a person, while the thing that is being sold is non-human. (We admit that, in a real dictionary, this might be a somewhat simple definition, but it will serve our purpose).
Also some special relations are given here. E.g. a student is a person.

The dictionary is not described in this paper. A possible way to describe the full dictionary is given in the Ph.D. thesis of H.Weigand [42]. In that thesis the lexicon is described as a taxonomically ordered set of F-structures. Scott's theory of information systems [36], forms the basis for the reasoning capability with these structures.

The Knowledge Base

The knowledge base is, of course, the most important part of the knowledge base system. The knowledge base contains the "knowledge" that is maintained by the knowledge base system. Before a formal definition is given of the knowledge base, we will say a few words about what we understand to be "knowledge".
Already Aristotle mentioned two main characteristics of knowledge: knowledge is knowledge of the *causes* and knowledge can be *communicated*. We will come back to the second characteristic in the next section, where the contribution of linguistics to this paper is discussed. It is of more importance for the knowledge representation than for the definition of what is knowledge.
The first characteristic states that knowledge contains knowledge of the *causes*. That is, we must be able to ask questions about WHY the things are as they are described in the knowledge base. So, the knowledge base should not only contain knowledge about facts, but also knowledge about the "essence" of these facts. E.g. we must not only be able to ask which persons borrowed a particular book, but also what *is* a person. Aristotle calls this the "formal cause". In the same way knowledge must contain the "effective cause". That is, how have the facts become as they are at this moment. What were the events that changed the facts, etc.. This implies that the knowledge base should also contain information about events.
The two other causes that are distinguished are the "material cause" and the "final cause". The first defines what a thing is made of, e.g. a string consists of characters. The second is more important and indicates the purposes of the events that are performed. Although the importance of this knowledge is recognized, also for a knowledge base system, we did not include it in the definition as we used it in this paper. This was mainly because we wanted to avoid the complexities that would result from the inclusion of this type of knowledge in the knowledge base. However, in future extensions it should certainly be included as well.

The general definition of a knowledge base, as used in this paper, is as follows:

Definition:

> A knowledge base is a set of statements that describe the *knowledge* about the truths of the actual world plus a set of constraints that describe statements that must be true in all

possible worlds and statements that ought to be true (in all possible worlds).

In this definition the knowledge about the truths of the actual world includes the above mentioned *causes* of these truths.

Note the difference with Reiter's definition. We do not require the knowledge base to consist of statements about the truth of facts in the world, but statements about the *knowledge* of the truth of facts in the world. Because the knowledge of the world may be incomplete, the statements about this knowledge may also be indefinite (i.e. contain disjunctions and/or "null values"). For instance, we might know that there is a person, who returned the book with booknumber 1081, without knowing who it was. Then we can still have the following statement in the knowledge base:

The book with booknumber 1081 has been returned by someone.

Note also that we do not make a difference between specific and general facts, but between facts and constraints. Usually the general facts are seen as constraints, but the following example shows that this is not always correct.

Suppose the following statement is true of the world:

All employees of the Free University of Amsterdam live in Holland.

Then this is certainly a general fact, but that does not make it a constraint. That is, it might just be accidentally true for the present world, but it need not be required. Some day an employee might decide to movw to Belgium, after which the statement would no longer be true.

The constraints are divided in constraints that are always true and constraints that ought to be true. In our view, this is an important distinction. It can be made clear by the following example:

(1) All persons have an age which is non-negative.
(2) All persons that borrow a book must return it (within 3 weeks).

Clearly, that the first constraint will always be true. It is what in [28] is called an *analytical* constraint. The second constraint, however, may be violated. This kind of constraint is called a *deontic* constraint.

The above constraints also show that we have both static and dynamic constraints. That is, we have constraints about the facts and constraints about the actions.

Implicitly, it is also indicated that the knowledge base contains temporal information. Thus, it might contain the information that John was a student in 1986 or that Peter borrowed a book 3 weeks ago.

In the description of a knowledge base, within a knowledge base system as defined above, two different theories play an important role. They are the theory of linguistics and the theory of logic.

In the following sections we will explain the role of each of these theories in our language for describing knowledge bases.

2.2. Linguistics

Because the knowledge in the knowledge base should be accessible and updatable in the same way as the data in a database, we require that the knowledge in the knowledge base be represented in a uniform way. This makes it possible to define uniform update and query routines such that also large knowledge bases are manageable and have fast access possibilities.

This requirement is not fulfilled by most of the modelling languages existing today. Usually, the static and dynamic component are described using different techniques. For instance, the processes

are defined by some graphical language (Petri-nets), while the constraints are defined using first-order logic.
The factual knowledge should, of course, also be represented in the same format as the rest of the knowledge.

In the previous section, it has already been remarked in the definition of knowledge that it should also be communicable. That is, something can only be knowledge if it is possible to communicate it to other people. This led several philosophers, like Wittgenstein, Quine and Gadamer, to the proposition that all knowledge should be expressable in (natural) language and even that only natural language can be used to express all knowledge.
Although we do not support the thesis that knowledge can only be communicated by natural language, we do think that natural language is the best candidate to represent all the different kinds of knowledge (in a uniform way).

Of course, natural language is notoriously ambiguous. Also, the representation would then depend very much on the natural language chosen for the specification.
These problems, however, can be avoided by not using a natural language itself, but the semantic representation of natural language as given by a (well behaved) linguistic theory. The semantic representation will not contain any ambiguities, because it represents the exact meaning of the sentences. Also, it is less dependent on the natural language, because the structure of this representation should be the same for all languages (such that a sentence with the same meaning in different languages would have the same semantic representation, except for the lexical items.)

Because natural language can very well be seen as a knowledge representation language, it s obvious that some of the problems encountered in the theory of knowledge representation also occur in the theory of the semantics of natural language.
Since the semantics of natural language has already been studied for some centuries by philosophers and linguists, it is worthwhile to use this knowledge when a representation language for a knowledge base is designed.
Another advantage of taking the semantic representation of natural language as a basis for the knowledge representation is that this, of course, makes it easy to communicate with the knowledge base system in a natural way.

Some of the more prominent linguistic theories that could be used are: Transformational Generative Grammar [11], Montague-Grammar [17, 19] and Functional Grammar [16].
We will not explain these theories and their differences in this paper, but just state that we opted for the theory of Functional Grammar as the basis of our knowledge representation language. In this theory, the language is seen as a means of communication. This fits in very well with our definition of a knowledge base system. It means that the representation of a sentence should indicate how that representation can be used in a communication process. Because communication also involves reasoning about the information received, it means that it should be possible to use these linguistic structures in an inferencing process.

In Functional Grammar the representation of a sentence is called a predication. The predications are built from *frames*, which are stored in a dictionary.
For instance, the sentence:

John gives a book to Bill.

is represented by the following structure:

$$give_V\,(dx_i{:}John(x_i))_{Ag}\,(ix_j{:}book(x_j))_{Go}\,(dx_k{:}Bill(x_k))_{Rec}$$

which means that some definite Agent called "John" gives some indefinite Goal called "book" to a definite Recipient called "Bill".
This predication is built by filling in the frame for the verb "give", which looks as follows:

$$give_V\,(x_1{:}human(x_1))_{Ag}\,(x_2)_{Go}\,(x_3{:}animate(x_3))_{Rec}$$

This frame specifies that the agent of the action "give" must always be a human and the recipient must always be animate.
It may seem obvious that these frames also form a good basis for the dictionary of the knowledge base system. That this is indeed the case is shown in [42].

The theory of parsing and generating natural language from Functional Grammar predications can also be used to translate the natural language into the knowledge representation language and vice versa. That this is indeed possible is shown in [10]. In that Master's thesis a parser is described that takes natural language sentences as its input and produces structures of our knowledge representation language. In [14] it is shown how Functional Grammar can be used to assist in the parsing process of a natural language sentence.
It does not need saying that the basis of our language CPL is Functional Grammar. The syntax of CPL is given in [13].

2.3. Logic

The other component that plays an important role in the knowledge representation language is logic.
From the definition of a knowledge base it follows that the structures that describe the knowledge in the knowledge base should state something about the truth values of the knowledge of the world. This implies that there is an underlying truth theory, which determines the truth value of each structure.
Traditionally, First Order Logic, in the form of Predicate Logic, has been used to describe the meaning of a database and its constraints. By giving a logical representation of our language it is easy to define inference rules and also to check the consistency of the knowledge base.

It may be obvious that to give the interpretation of all the different kinds of knowledge we need more than first-order logic. In addition we will use four kinds of (modal) logics to describe the interpretation of the knowledge representations. They are:

modal logic (expressing necessity)
deontic logic
dynamic logic
temporal logic

The necessity operator of modal logic is used to express the constraints that always have to be true. Deontic logic is used to define the constraints that ought to be true. The interpretation of the actions is given, using the dynamic logic as described in [29].
Finally, temporal logic is used to describe the temporal aspects of the knowledge representation language.

3. Example

Before introducing CPL in section 4 we now give some examples, showing how knowledge can be modelled.

The first example describes the fact that if a person borrows a book he must return it within 3 weeks.
In CPL, this is described as follows:

(1) MUST:
ACTION:return(<1> ag=A **in** person)(<1> go=B **in** book)(<1> tmp=U **in** time)
(**id:** U before T+3*week)
(**sit:** ACTION:
borrow(<1> ag=A **in** person)(<1> go=B **in** book)(<1> tmp=T **in** time))

The above CPL structure is called a "predication". A predication can be divided in three parts: the modality, the Situation and the Destination. The modality in the above example is MUST, which means that it is deontic and thus the actions are obligatory.
The Situation of the example is the part indicated by **sit**. This part describes the "situation" of the world. If the situation holds for the world then the Destination should be true also. So, it functions as the antecedent of an implication. In the example, the Situation is:

(**sit:** ACTION:
borrow(<1> ag=A **in** person)(<1> go=B **in** book)(<1> tmp=T **in** time))

which describes the situation in which a person borrows a book at a time T.
The word ACTION indicates that the situation describes an action and not a state. The action is a "borrow" action with three arguments. The first argument is the "agent" of the action and is of the type person. The "<1>" indicates the cardinality of the argument. In this case, that it is one person (not a group of several persons).The second argument of the action is the "goal" of the action and is of the type "book". The third argument indicates that the action is performed at time T.
The Destination of the example is:

ACTION:return(<1> ag=A **in** person)(<1> go=B **in** book)(<1> tmp=U **in** time)
(**id:** U before T+3*week)

It describes the fact that the person returns the book within 3 weeks. By using the same variables ('A' and 'B') in the arguments of "return" it is indicated that the person and the book in the Destination are the same as those in the Situation. The time of the returning of the book, however, is different. The time of the returning is further specified by an identifier on the time argument, indicated by **id**.
In this identifier it is stated that the time of returning lies within 3 weeks after the time of the borrowing (indicated by T+3*week). In principle, each argument can be further identified by one or more identifiers. These identifiers themselves may be predications again, so that a nesting of identifications can be given.

That it is possible to reason with this kind of structures is shown by the following example:
If John is obliged to return a book with booknr. 1081 after 3 weeks and John is obliged not to return any book within 4 weeks then John can never fulfil all obligations. This is indicated in CPL by the fact that there is an obligation to perform Fail after 3 weeks. Fail stands for the impossible action. It

leads to no new world. Therefore, the obligation to perform Fail will always lead to a violation of this obligation.

(2) MUST:
ACTION:return(<1> ag=A **in** person)(<1> go=B **in** book)(<1> tmp=T **in** time)
(**id**:PERF:has(<1> zero=A **in** person)(<1> pat=C **in** name)(**id:** C=John))
(**id**:PERF:has(<1> zero=B **in** book)(<1> pat=D **in** booknr)(**id:** D=1081))
(**id**: T=NOW+3*week)

(3) MUST:
ACTION:return(<1> ag=A **in** person)(<~> go=B **in** book)(<1> tmp=T **in** time)
(**id**:PERF:has(<1> zero=A **in** person)(<1> pat=C **in** name)(**id:** C=John))
(**id**: T before NOW+4*week)

⊃

(4) MUST:ACTION:Fail(<1> tmp=T **in** time)
(**id**: T=NOW+3*week)

In the CPL predication of (2), all the arguments of "return" are further identified. The person (A) is further identified by the predication:

(**id**:PERF:has(<1> zero=A **in** person)(<1> pat=C **in** name)(**id:** C=John))

which indicates that the person has a name (C), which is "John". Instead of ACTION, this predication is preceded by PERF. This means that the predication describes a state, which holds at the present time. To indicate something that held in the past (but is no longer true) we use PRET. To describe a future state in CPL, FUT is used.
In (3), the second argument of the action "return" has cardinality "<~>". This indicates that the set of books that are the goal of this "return" action is empty. If this argument was further identified with e.g. the following identifier:

(**id**:PERF:has(<~> zero=B **in** book)(<1> pat=D **in** author)
(**id**:PERF:has(<1> zero=D **in** author)(<1> pat=E **in** name)
(**id:** E=Tanenbaum)))

then the predication would describe the fact that no books written by Tanenbaum may be returned by John within 4 weeks.
In the same way the cardinality of an argument can be "<∗>", which means "all". So, had the cardinality of the second argument of the "return" action been <∗> then the predication would have meant that John returns all books within 4 weeks.

That the above inference in CPL is legitimate (or sound) can only be proven if we can give the semantics of these CPL predications and show that the inference also follows in the semantics. As already said in section 2.3, the semantics of the CPL predications is given by first translating them into logic and then giving a semantics to that logical interpretation.
Before we give the logical interpretation of the examples we will explain in some more detail the dynamic, modal and temporal aspects of the CPL predications (and their logical interpretation) in the next sections. Only then is it possible to see that the inference can also be made in the logical

interpretation. That this is indeed the case is shown in section 7.

4. Modalities

Usually, the knowledge base (or database) is divided into two parts:

1. Data consisting of facts
2. General world knowledge

See e.g. [32, 34, 18, 23] for descriptions of this division. In CPL the facts are defined as follows:

Definition:

The facts of the knowledge base are CPL predications without Situations and with modality FACTUAL.

Besides the facts, the static part of the knowledge base consists of what is often referred to as "general world knowledge". This general knowledge is often divided into two parts, the general rules and the constraints.(See [18]) The decision whether a formula is a rule or a constraint is often taken on the basis of practical reasons. Usually only certain kinsd of rules are allowed, and therefore, each formula that does not comply with the restrictions of the rules should be a constraint.
Although from a practical point of view this might be a valid decision, from a theoretical point of view it is hardly defensible.
The decision whether a formula is a rule or a constraint should be taken on the basis of what is the intention of the formula and not on the basis of its syntax.
The intention of the formula can be seen from the modality of the formula. Therefore, the distinction between the different kinds of general world knowledge in CPL is made on the basis of the modality of the formulas, while the syntax of the formulas is not distinguishing.

Another deviation from the usual definitions of the general world knowledge is that we believe that this "general world knowledge" consists of three parts! (which are therefore distinguished by three different modalities) These three parts are:

1. **Factual knowledge**. These are general rules that can have both a Situation and a Destination and which have modality FACTUAL. These rules are true at a certain moment in time, but they can be changed by updates on the knowledge base. They can also be used to derive new facts from the knowledge base. So, they are comparable with the rules in deductive databases.
   An example is:

   FACTUAL: publish(<1> ag=A **in** publisher)(<1> go=B **in** book)
   (**id:** has(<1> zero=A **in** publisher)(<1> pat=C **in** name)(**id:** C=Prentice-Hall)))
   (**sit:** write(<1> ag=D **in** person)(<1> go=B **in** book)
   (**id:** has(<1> zero=D **in** person)(<1> pat=E **in** name)(**id:** E=Tanenbaum)))

   which means that if Tanenbaum writes a book then it is published by Prentice-Hall.

2. **Knowledge of necessity**. This knowledge is necessarily true in the knowledge base. It coincides with the analytical and empirical rules from [28]. Since these rules are necessarily true, they cannot be changed by any update and moreover they have to be still true after any update. Like the Factual rules, they can, of course, be used to derive new facts from the knowledge base.
   An example is:

NEC: has(<1> zero=A **in** person)(<1> pat=C **in** grandfather)
(**sit:** has(<1> zero=A **in** person)(<1> pat=B **in** father))
(**sit:** has(<1> zero=B **in** person)(<1> pat=C **in** father))

which states that the father of the father of a person is the grandfather of that person.

3. **Deontic knowledge**. This knowledge consists of the classical "constraints" of the knowledge base. They are used to check if the agents in the UoD behave in a permissible way. That is, if the state of the knowledge base is a model of a permissible state in the UoD. (Here the word "model" is used in a descriptive non-mathematical sense.)
   It is not possible to derive new facts from the knowledge base with these rules. Also, they cannot be changed by updates on the knowledge base.
   Both of the examples in the previous section are of this kind.

In the next sections, the logical interpretation of each of the above modalities is explained.

4.1. Modality FACTUAL

4.1.1. Facts

By definition, the facts consist of CPL predications without Sit's. That is, the facts consist of the Destination only.
The modality FACTUAL is interpreted here by the absence of a modality in the logical interpretation (which traditionally means that the formula is a fact).
Some examples of facts are:

(1) FACTUAL:
has(<1> zero=A **in** employee)(<1> pat=B **in** empnr)(**id:** B=16700)
(**id:** has(<1> zero=A **in** employee)(<1> pat=C **in** name)(**id:** C=Brown))
(**id:** has(<1> zero=A **in** employee)(<1> pat=D **in** address)(**id:** D=Amsterdam))
(**id:** has(<1> zero=A **in** employee)(<1> pat=E **in** function)(**id:** E=teacher))

This represents an employee entity, with properties empnr,name,address and function, where the value of the properties is "16700", "Brown", "Amsterdam" and "teacher", respectively.

(2) FACTUAL:
employ(<1> ag=A **in** university)(<1> pat=B **in** employee)
(**id:** has(<1> zero=A **in** university)(<1> pat=C **in** name)(**id:** C=VU))
(**id:** has(<1> zero=B **in** employee)(<1> pat=D **in** name)(**id:** D=Dignum))

This represents a relation "employ", between a university with the name "VU" and an employee with the name "Dignum".

(3) FACTUAL:
order(<2,d> ag= A **in** university)(<3,d> go=B **in** computer)
(**id:** has(<2,d> zero= A **in** university)(<1> pat= C **in** address)(**id:** C=Amsterdam))
(**id:** has(<3,d> zero= B **in** computer)(<1> pat= D **in** type)(**id:** D=Sun3/50))

This fact represents a relation "order" between a set of two universities that have as address "Amsterdam" and a set of three computers with type "Sun3/50". (Where each university orders 3 computers.)

(4) FACTUAL:

give(<1> don=A **in** gvmnt)(<*,d> rec=B **in** univ)(<1> go=C **in** subsidy)
(**id:** has(<1> zero=A **in** gvmnt)(<1> pat=D **in** nat)(**id:** D=Dutch))
(**id:** has(<1> zero=C **in** subsidy))

This fact represents a relation "give" between the Dutch government, all universities and a set of subsidies, where the Dutch government gives each university a subsidy.

From example (4) it can be seen that some CPL facts can be represented by logical rules. That is, these facts can be used to deduce new facts from the knowledge base. From the fact of example (4) and the fact that the VU is a university it can be concluded that the VU gets a subsidy from the Dutch government.
So, facts are not only the occurrences of entities and relationships, but can also express logical rules.

4.1.2. Factual knowledge

The factual knowledge consists of CPL structures with modality FACTUAL that contain both a Destination and a Situation. Therefore, they can always be used as deduction rules. The logical interpretation of these CPL structures is as described in the section on the relation between Situation and Destination. Just as with the Facts there is no modality in the interpretation of these structures.
The fact that they can be changed by an event or do not hold any longer after an event, means that these rules are in fact not general rules that are supposed to always hold, like "the father of my father is my grandfather".
The FACTUAL rules are used for practical reasons. If certain facts can be deduced from other facts at a certain moment it is a waste of space to store all the facts that can also be deduced.
For example, suppose that all computer factories that produce PC's get the terminals from another company that sells terminals. Then each time a factory produces a PC it has to order a terminal from that company.
This can then be represented by the following CPL structure:

FACTUAL:
order(<1> ag=A **in** factory)(<1> go=D **in** terminal)(<1> ben=E **in** company)
(**id:** sell(<1> ag=E **in** company)(<1> go=D **in** terminal))
(**sit:** produce(<1> ag=A **in** factory)(<1> go=B **in** comp)
(**id:** has(<1> zero=B **in** comp)(<1> pat=C **in** type)(**id:** C=PC)))

Now it is not necessary to store the separate Facts for each factory, but it is possible to use the rule to derive the Facts. However, if a factory starts producing the terminals of the PC's itself, the rule is no longer valid and should be retracted.

4.2. Knowledge of necessity

The name "necessary rules" already suggests a strong relationship with the modal necessity operator "□".
The necessary rules can be interpreted by this logical modal operator. As a consequence, the rules with this modality are not only true in the present state, but also in all states "reachable" from the present state. A state is reachable from the present state if there is a (sequence of) (trans)action(s) that transforms the present state into that state. This means that the necessary rules are always true,

whatever update takes place.
It will suffice at this place to give another example of a necessary rule.

NEC: has(<∗> zero=A **in** person)(<1> pat=B **in** age)(**id:** B≥0)

4.3. Deontic knowledge

Besides the modalities FACTUAL and NEC that have been treated in the previous sections there is one other modality concerning the static part of the knowledge base, namely the modality MUST. This modality is used to model the Deontic rules. Since these rules coincide with what are generally known as constraints, we will also refer to these rules as "constraints". However, it is important to note that the "constraint" is the whole rule, including the modality! In the literature on logic databases [18, 23, 34] the constraints are usually ordinary logical formulas without any modality.

The intuitive meaning of constraints is that they should be checked against a particular set of facts (The formulas with modality FACTUAL and NEC). They are not used to derive new facts.
So, from the fact that *MUST*ψ is true it does not follow that ψ is true in the knowledge base. It only expresses a kind of *obligation* for ψ to be true. Hence, the reason to call the formulas with this modality "deontic" ("what should be").

There are two aspects of the *MUST* rules that require our attention. The first is the truth value of these rules. This aspect will be handled by a truth valuation function (see [15]). The second aspect is the relation between the constraints and the facts and necessary rules from the knowledge base. In the literature [34] the emphasis in this relation is placed on the question whether the "constraint" (without modality) should be consistent with the database or should be entailed by the database.
In the second view, a formula containing a set of facts and a constraint that contains a formula that is not "derivable" from the facts are not "consistent", while a formula containing the same set of facts and a constraint that contains the same formula are "consistent" in the first view. With "consistent" is meant that the database "satisfies" the constraints.
In the following very simple (propositional) example the difference between the two views is made clear. Consider a knowledge base consisting of the following fact: *a* (an atom standing for, say, John is married) and the constraint: *MUST(b)* (standing for, say, Mary should be married)

a andsign MUST(b)

That the fact *a* is consistent with the constraint *b* can be checked easily, by making a model based on *a* and *b*. So, in the first view the fact is consistent with the constraint and therefore the database is consistent.
According to the "entailment" view, the database is not consistent because the database, consisting of the single fact: *a* does not "satisfy" the constraint.

In the "consistency" view it could be argued to say that a database is inconsistent if the facts and the constraints are not consistent. However, in the "entailment" view it is clear that the relation between the facts and constraints is NOT the same as the consistency relation. That is, if the database does not satisfy the constraints, that does not necessarily mean that it is possible to derive *false* from the facts and the constraints.

The introduction of the *MUST* operator makes it possible to make a distinction between the consistency of the knowledge base and the satisfaction of the constraints.

The general idea is that the formulas with modality *MUST* are always true unless they are internally inconsistent.(i.e. of the form *MUST*(*aandsign*¬*a*)).
So, the following might be always true:

MUST:possess(<1> pos=A **in** comp)(<1> pat=B **in** product)
(**sit:** sell(<1> ag=A **in** comp)(<1> go=B **in** product))

That is, the fact that a company MUST possess a product if it sells that product is always true. However, this does NOT mean that the following

FACTUAL:possess(<1> pos=A **in** comp)(<1> pat=B **in** product)
(**sit:** sell(<1> ag=A **in** comp)(<1> go=B **in** product))

is always true. If this predication is not true then we say that the knowledge base *violates* the constraint. The *violation* relation determines the relation between the facts and necessary rules on one side and the constraints on the other side. This relation is modeled according to the "entailment" view. That is, the knowledge base *violates* the constraints if they are not entailed by the facts and necessary rules.
If the knowledge base violates the constraints then a special CPL predication Violation is true. This predication is comparable with the special predicate V_i:α as introduced in [28]. It is, however, less specific than this predicate which also states by which action the violation state is reached. We have the following rule concerning the relation between facts and deontic rules:

Theorem:

Let

FACTUAL:P_1 *andsign...andsign FACTUAL:*P_n ⊃ *FACTUAL:P*
MUST:Q_1 *andsign...andsign MUST:*Q_m ⊃ *MUST:Q*
and
$P \equiv \sim Q$

then

FACTUAL:P_1 *andsign...andsign FACTUAL:*P_n *andsign*
MUST:Q_1 *andsign...andsign MUST:*Q_m
⊃ *Violation*

Proof: See [15]. □.

An example of this rule is:
If a person is not allowed to borrow more than three books at the same time and the student John borrowed four novels then this will lead to a violation of the constraints.

MUST:~borrow(<1> pos=A **in** person)(<3+> go=B **in** book)

FACTUAL:borrow(<1> pos=A **in** student)(<4> go=B **in** novel)
(**id:**has(<1> zero=A **in** student)(<1> pat=C **in** name)
(**id:** C=John))

⊃

Violation

At the end of this section we discuss the question whether we could have introduced the MUST operator with the standard deontic operator *O* (for obligation). [20]
The introduction of *O* would have raised questions about the status of the other deontic operators (*P* and *F*) and their relation to the facts. Since we do not need the full power of the *standard* deontic logic, because we only need the obligation to express the constraints, we chose not to interpret the CPL MUST operator by the *O* operator.
The modality from CPL is instead interpreted by an operator with the same name in the logical interpretation.
In [15] it is proven that the axioms that hold for the *O* operator in standard deontic logic (see also [20]) also hold for this operator.

5. Dynamics

5.1. Introduction

In this section, some of the dynamic aspects of CPL are described.
The logical interpretation of events is described using dynamic logic. A first attempt to describe updates in databases using dynamic logic is made in [22]. In [28], dynamic and deontic constraints are described using dynamic logic. The theory of dynamic logic as used for the description of the events in CPL is mainly based on the latter. We will describe the semantics of the events as used in dynamic logic as far as relevant for the aspects of CPL that are described in this paper.

This section describes two different aspects of events:

1. The event as FACTUAL information in the knowledge base. That is, the knowledge base contains the information that a certain event has taken place.
2. The interpretation of the CPL structures involving events (i.e. pre- and postconditions, etc.) in (dynamic) logic.

In the next section we will describe those parts of dynamic logic that play a crucial part in the logical interpretation of the dynamic CPL predications. In section 5.3, the logical interpretation of an event in dynamic logic is described. In this case the event is seen as something that happened in the UoD. That is, it is an instance of the event class. These events are stored in the knowledge base as FACTUAL information. In this section, it is also described how some deontic information on events is expressed in CPL and interpreted in logic. The connection of the pre- and postconditions of an event class with the events are described in section 5.4. There it is possible to formalize when an event is "permissible" and "obliged".

5.2. Dynamic and Deontic logic

We will start this section by describing the set of all action expressions, i.e. all ways an action can be denoted. The set of all action expressions ***Act*** can be determined by the following BNF for its elements (α):

$$\alpha ::- \underline{a} \mid \alpha_1 \cup \alpha_2 \mid \alpha_1 \& \alpha_2 \mid \bar{\alpha} \mid \textbf{any} \mid \textbf{fail}$$

$\underline{a}$ denotes an atomic action. The meaning of $\alpha_1 \cup \alpha_2$ is a choice between α_1 and α_2. $\alpha_1 \& \alpha_2$ stands for the parallel execution of α_1 and α_2 and $\bar{\alpha}$ stands for the negation of action α.
The **any** action is a universal or "don't care which" action. Finally, the **fail** action is the action that always fails. After this action the system stops and nothing can be done any more.

Dynamic assertions

The language in which the actions as described above play a part is a variant of what is called PDL (Propositional Dynamic Logic) in the literature [29, 30] and is almost equal to the one defined in [28].

Definition:

The language L_{dyn} of dynamic expressions, with typical elements Φ and Ψ, is given by the BNF:

$$\Phi ::= \phi \mid \Phi_1 \mathit{andsign} \Phi_2 \mid \Phi_1 \mathit{orsign} \Phi_2 \mid \neg\Psi \mid \forall x \Psi \mid [\alpha]\Psi \mid \mathit{DONE}{:}\alpha \mid \mathit{PERF}{:}\alpha$$

where ϕ is a formula from first-order logic.

End Definition

Without giving the formal definition of the valuation function and the semantics of this language, we will say a few words about the meaning of the dynamic constructs from L_{dyn}.
The formula $[\alpha]\Psi$ denotes the fact that Ψ is true in the world that is reached after α is performed.
DONE:α expresses the fact that α has just been performed. As a consequence it holds in those worlds that are reached by performing α.
PERF:α is true iff the action α has been performed. However, unlike the *DONE* operator, the action does not have to be the last action that was performed. Therefore, it may very well be that both *PERF*:α and *PERF*:$\bar{\alpha}$ are true. (The operator was already briefly introduced in the appendix of [28].)

To be able to reason with the language L_{dyn} we introduce the following derivation rules:

Rules:

(N) $\quad \Phi \vdash [\alpha]\Phi$

(S) $\quad \alpha =_C \alpha' \vdash \Phi(\alpha) \leq> \Phi(\alpha')$

where $\Phi(\alpha)$ stands for a formula containing the action α and $\alpha =_C \alpha'$
indicates that the semantics of the action α is equal to the semantics of action α'.

Note that the antecedent of rule S is not really an expression from L_{dyn}. It would, however, not be difficult to axiomatize the equations given on the actions and incorporate them into our logic. Without formally doing so we consider this to be done and will not give all the axioms and the derivation rules that are involved in this axiomatization.

With the language as introduced above we can now express formally when an action is

forbidden, permitted or obliged. To express these notions we use the following abbreviations:

Definition:

The fact that α is forbidden is expressed by: $F(\alpha)$ which is an abbreviation for $[\alpha]Violation$

The fact that α is permitted is expressed by: $P(\alpha)$ which is an abbreviation for $[\alpha]\neg Violation$

The fact that α is obliged is expressed by: $O(\alpha)$ which is an abbreviation for $[\bar{\alpha}]Violation$

End Definition

In our version of dynamic logic it holds that $P(\alpha) <\!\!\rightarrow \neg F(\alpha)$ and that $F(\alpha)=O(\bar{\alpha})$.

With the dynamic logic as introduced in this section it is now possible to specify the dynamic features of the knowledge base. In the next sections we will show how the CPL structures that describe the dynamic features can be interpreted using the dynamic logic introduced above.

5.3. Actions as facts, permissions and obligations

As said in the introduction of this section, it should be possible to describe the fact that an event has occurred. That is, store it in the knowledge base as FACTUAL information.
We will first describe how the fact that an event has occurred is expressed in CPL. Given this description, a translation to our language L_{dyn} is described afterwards.

Suppose that John has borrowed a book. Then this fact is expressed in CPL as follows:

FACTUAL:ACTION:borrow(<1> ag=A **in** person)(<1> go=B **in** book)
(**id**: has(<1> zero=A **in** person)(<1> pat=C **in** name)(**id:** C=John))

If the above CPL predication is a fact in the present knowledge base, it means that somewhere in the past this event has occurred.
This intuitive interpretation is made more formal by the following translation of the CPL structure to L_{dyn}:

∃A'(person(A') andsign ∃C' (name(A',C') andsign C'=John)
andsign ∃B'(book(B') andsign PERF:borrow(A',B')))

As can be seen from the example, the use of modality FACTUAL and tense ACTION only indicates that the event occurred at some time in the past. To indicate that the event was the last event that was performed we will use DONE, which has, of course, a direct representation in L_{dyn}. So, to describe the fact that John has just borrowed a book, we use the following CPL structure:

FACTUAL:DONE:borrow(<1> ag=A **in** person)(<1> go=B **in** book)
(**id**: has(<1> zero=A **in** person)(<1> pat=C **in** name)(**id:** C=John))

which has as its logical interpretation:

∃A'(person(A') andsign ∃C' (name(A',C') andsign C'=John)
andsign ∃B'(book(B') andsign DONE:borrow(A',B')))

In general, the fact that an event has occurred is represented by a CPL predication without Situation, where the modality is FACTUAL and the "TENSE" is ACTION or DONE.

To indicate in CPL that a certain action is permitted or obliged we use the modalities PERMIT and MUST. So, to indicate that John is permitted to borrow a book, we use the following CPL predication:

PERMIT:ACTION:borrow(<1> ag=A **in** person)(<1> go=B **in** book)
(**id**: has(<1> zero=A **in** person)(<1> pat=C **in** name)(**id:** C=John))

The fact that John must return a book is described in CPL as follows:

MUST:ACTION:return(<1> ag=A **in** person)(<1> go=B **in** book)
(**id**: has(<1> zero=A **in** person)(<1> pat=C **in** name)(**id:** C=John))

The logical interpretation of these examples is straightforward:

$\exists A'(\ person(A')\ andsign\ \exists C'\ (name(A',C')\ andsign\ C'=John)$
$andsign\ \exists B'(\ book(B')\ andsign\ P(borrow(A',B'))\ ))$

$\exists A'(\ person(A')\ andsign\ \exists C'\ (name(A',C')\ andsign\ C'=John)$
$andsign\ \exists B'(\ book(B')\ andsign\ O(return(A',B'))\ ))$

Note that we do not need a modality for the forbidden actions. These prohibitions can always be expressed as the obligation of the negated action.

5.4. Pre- and Postconditions

In this section we describe how the pre- and postconditions, as they are defined in CPL, can be interpreted in L_{dyn}. We will start with the preconditions in the next section. The postconditions and the "triggers" will be handled in the last section.

5.4.1. Preconditions

In [13], it has been observed that there are two kinds of preconditions, the necessary preconditions and the sufficient preconditions. The difference between the two types of preconditions can be described as follows. The necessary preconditions have to be fulfilled if an action is permitted; while, if the sufficient preconditions are fulfilled, the action is permitted. The causal relations between the precondition and the action are in opposite directions for the two types of preconditions. The two types of preconditions will be handled in the following two sections.

5.4.1.1. Necessary preconditions

As said before, a necessary precondition of an action should be true if the action is permitted. They can be described in L_{dyn} by the following schema:

$$P(\alpha) \rightarrow \Phi$$

If the logical interpretation of the CPL structures, denoting necessary preconditions, complies with this schema, the action should be in the Situation and the necessary precondition in the Destination. How this can be done in CPL is shown by the following example and its logical interpretation:

PERMIT:ACTION:reserve(<1> ag=A **in** person)(<1> go=B **in** book)
(**dest**: possess(<1> pos=A **in** person)(<1> pat=C **in** member_nr))
(**dest**: has(<1> zero=B **in** book)(<1> pat=D **in** availability)
(**id**: D<>present))

which means that if a person can reserve a book then he must be a member and the book must not be present.
The logical interpretation of this CPL structure is:

$\forall A' \forall B'(\ person(A')\ andsign\ book(B')\ andsign\ P(reserve(A',B'))$
$\rightarrow$
$\exists C'(\ member\_nr(C')\ andsign\ possess(A',C')\ andsign$
$\exists D'(\ availability(B',D')\ andsign\ D' \neq 'present'))\ )$

5.4.1.2. Sufficient preconditions

In this section we will describe how the second type of preconditions (the sufficient preconditions) can be described in CPL and what the logical interpretation of these preconditions is in L_{dyn}.
The general schema of these preconditions in L_{dyn} is:

$\Phi \rightarrow P(\alpha)$

That is, if the precondition is fulfilled then the action is permitted.
The following example will make this clear:
If a person is a member and the book is present, then the person can borrow the book.

PERMIT: ACTION: borrow(<1> ag=A **in** person)(<1> go=B **in** book))
(**sit**: possess(<1> pos=A **in** person)(<1> pat=C **in** member_nr))
(**sit**: has(<1> zero=B **in** book)(<1> pat=D **in** availability)
(**id**: D=present))

The logical interpretation in L_{dyn} is obvious and is as follows:

$\forall A'\ \forall B'(\ person(A')\ andsign\ book(B')\ andsign$
$\exists C'(member\_nr(C')\ andsign\ possess(A',C')\ andsign$
$\exists D'(availability(B',D')\ andsign\ D' = 'present'))$
$\rightarrow$
$P(borrow(A',B'))\ )$

Note that it follows immediately from the definitions that the sufficient preconditions (*SP*) always have to imply the necessary preconditions (*NP*). ($SP \rightarrow P(\alpha) \rightarrow NP$).

5.4.2. Postconditions

The postconditions of an action can be divided into two types as well. We will distinguish static and dynamic postconditions.
The static postconditions are the postconditions as they are usually used in the literature (see e.g. [3]).
The dynamic postconditions are the postconditions that in their turn involve an action. They can be obligating another action or they can trigger another action. An example of an obligation following an action is that one is obliged to return a book when one has borrowed it. An example of a trigger is that

if one pushes the button of a doorbell the bell will start ringing (provided it works, of course).
We will start in the next section with the description of the static postconditions. We will then say more about the dynamic postconditions and the difference between the triggers and "pending actions" in the succeeding section.

5.4.2.1. Static postconditions.

For the static postconditions it holds that they are true directly after the action. The general schema of the postconditions is written in L_{dyn} in the following way:

$[\alpha]\Phi$

For the formula Φ we can distinguish two cases.

1. Φ is a non-modal formula.
   In this case the postcondition (Φ) indicates an immediate effect of the action. The CPL predications that describe this type of postcondition will have modality FACTUAL.
   This type of postconditions is sometimes used to describe the actions. The actions are then completely defined by their effects (expressed as postconditions)(see e.g. [40, 9, 31, 41]).
   The reason that we will not define the action completely by its postconditions is that they usually underdefine the exact changes that have to be made to the knowledge base. That is, the postconditions can be fulfilled by more than one set of possible changes.
2. Φ is of a deontic nature.
   In this case the postcondition (Φ) indicates that a certain constraint is added to the set of deontic constraints. In CPL, the predications that describe this type of postcondition have modality MUST.

An example of each type of postcondition in CPL can be given as follows:

FACTUAL: possess(<1> zero=A **in** person)(<1> pos=B **in** book)
(**sit:** ACTION: borrow(<1> ag=A **in** person)(<1> go=B **in** book))

MUST: ~possess(<1> zero=A **in** person)(<3+> pos=C **in** book)
(**sit:** ACTION: borrow(<1> ag=A **in** person)(<1> go=B **in** book))

The first example states that the immediate effect of borrowing a book is that one possesses it.
The second example states that it is not allowed to possess more than 3 books after one borrows a book.
The translation of these examples in L_{dyn} is as follows:

$$\forall A' \forall B' (\, person(A') \; andsign \; book(B') \rightarrow [borrow(A',B')] possess(A',B') \,)$$

$$\forall A' \forall B' (\, person(A') \; andsign \; book(B') \rightarrow [borrow(A',B')] MUST(\neg \exists C(\, card(C) \geq 3 \; andsign \; \forall X \in C(book(X) \; andsign \; possess(A',X)) \,)))$$

It is worthwhile to mention at this point that the immediate effects of each action should be consistent with the necessary facts. If this were not the case then we would arrive at an inconsistent state after the action, because in that state both the immediate effects of the action and the necessary facts

hold.
In the same way, the deontic postconditions should always be consistent with the static deontic constraints (which hold in every state).
Of course, this requirement is not necessary if the sufficient preconditions of an action can never be fulfilled (e.g. because they are inconsistent with the necessary facts). In that case the action will never be permitted and the postcondition will never have to be fulfilled.

5.4.2.2. Dynamic postconditions

In this section we will describe the postconditions of events that in themselves contain another event. The dynamic postconditions also have two different types.
The first type are the "pending actions". That is, actions that still have to be done. In other words, an action can cause a situation in which it is obliged to perform another action (the pending action).
The second type of dynamic postconditions is usually referred to as triggered actions.

Triggered actions are always performed, while the pending or obliged actions should be performed, but might not be performed (without the system entering an inconsistent state).
In CPL, this difference is expressed by using a different modality for the two types. The structure of the dynamic postconditions in CPL is the same as that of the static postconditions. The main difference is that the Destination now also contains an action.
The following examples illustrate the above points:

MUST:ACTION:return(<1> ag=A **in** person)(<1> go=B **in** book)
(**sit**: ACTION:borrow(<1> ag=A **in** person)(<1> go=B **in** book))

NEC:ACTION:ring(<1> ag=A **in** bell)
(**sit**: ACTION:push(<1> ag=B **in** person)(<1> go=A **in** bell))

The first example means that if a person borrows a book then he should also return it. The second example means that if a person pushes a bell, the consequence will be that the bell rings.
The translation of these examples in L_{dyn} is as follows:

$$\forall A' \forall B' (\, person(A') \; andsign \; book(B') \rightarrow [borrow(A',B')] O(return(A',B')) \,)$$

$$\forall A' (\, (\exists B' (person(B') \; andsign \; bell(A') \,)) \rightarrow [push(A',B')][\overline{ring}(A')] false \,)$$

Although the translations seem to differ quite a bit, they are not so different. To see this one should remember that $O(return(A',B'))$ is actually an abbreviation of $[\overline{return}(A',B')]Violation$.
If this is substituted in the logical expression again, it is clear that the main difference between the two examples is that not returning the book leads to a state of *Violation* while the not ringing of the bell leads to an inconsistent state.
This is exactly the difference between an action that is triggered and an action that is obliged. A triggered action is bound to happen, it is "caused" by a previous event. Therefore, if it does not happen the system is in an inconsistent state. An action that is obliged should happen, but it is not certain that it will really happen. Therefore, the system will be in a *Violation* state if the action does not happen.

We conclude this section with two remarks on the obliged actions.

The first remark is about the element of time. Usually, the actions that are obliged are required to be performed within a certain time. For instance, one has to return a book within three weeks after it is borrowed. Because this time element is not yet introduced in L_{dyn}, the CPL predication expressing this constraint cannot be interpreted in L_{dyn}. In the next section the logical language will be extended with some features of temporal logic which make it possible to interpret this constraint in our logic too. For now, if an action is obliged, it means that it should happen right away (as the first action that is to be performed).

The second remark on the obliged actions is about what should be done if the obliged action is not performed. For every obliged action one should specify what should be done if the action is not performed.

This can be expressed in L_{dyn} by using the following schema:

$$O(\alpha_i) \rightarrow [\overline{\alpha_i}]O(\alpha_j)$$

This schema indicates that if an action is obliged and not performed then a new obligation is created. This schema can be used indefinitely, creating new obligations every time an obliged action is not performed.

Usually, however, this sequence of new obligation is closed by the fact that the non-performance of a certain action triggers a (system) action. After this action is performed the system is (usually) in a state in which *violation* is not true. If the action is not performed, the system is in an inconsistent state.

The schema of this trigger is expressed in L_{dyn} as follows:

$$O(\alpha_i) \rightarrow [\overline{\alpha_i}][\overline{\alpha_j}]false$$

The following is an example of a sequence of new obligations:

If a person borrows a book he has to return it.
If a person does not return it he has to pay a fine.
If a person does not pay the fine he is expelled.

With the expulsion of the person, the system comes back to a state which does not violate the constraints any more.

The above example can be expressed in CPL in the following way:

MUST:ACTION:return(<1> ag=A **in** person)(<1> go=B **in** book)
(**sit**: ACTION:borrow(<1> ag=A **in** person)(<1> go=B **in** book))

MUST:ACTION:pay(<1> ag=A **in** person)(<1> go=B **in** fine)
(**sit**: ACTION:~return(<1> ag=A **in** person)(<1> go=C **in** book))
(**sit**: MUST:ACTION:return(<1> ag=A **in** person)(<1> go=C **in** book))

NEC:ACTION:expel(<1> go=A **in** person)
(**sit**: ACTION:~pay(<1> ag=A **in** person)(<1> go=B **in** fine))
(**sit**: MUST:ACTION:pay(<1> ag=A **in** person)(<1> go=B **in** fine))

The logical interpretation of the above example is as follows:

$\forall A' \forall B'(\ person(A')\ andsign\ book(B')$
$\rightarrow [borrow(A',B')] O(return(A',B'))\ )$

$\forall$A'$\forall$C'(person(A') *andsign* book(C') *andsign* O(return(A',C'))
$\rightarrow [\overline{return}(A',C')] \exists B'(\ fine(B')\ andsign\ O(pay(A',B')))\ )$

$\forall$A'$\forall$B'(person(A') *andsign* fine(B') *andsign* O(pay(A',B'))
$\rightarrow [\overline{pay}(A',B')][\overline{expel}(A')] false$

Actually, in the current version of CPL the above is not allowed, because it is not allowed to have a modality in the Situation of the Predication. (this would raise the problem of how to combine the different modalities in a predication, which we wanted to avoid as much as possible).

6. Time

6.1. Introduction

In [8] a survey is given on the subject of "time" in information processing.
We did not attempt to formulate a new theory on time. Rather, we use the elements of existing theories in as far as they are applicable to the knowledge base domain (as described here).

To add a temporal aspect to the knowledge base we have to describe the following points:

1. The structure of the time domain.
2. The syntax of the temporal logic that serves as logical interpretation of the CPL predications and its interpretation in the (extended) Kripke-model.
3. The translation of the temporal CPL predications to the temporal logic.

The structure of the time domain was discussed in philosophy for a long time before it became an issue in computer science. See e.g. [7]. The structure of time first became important for computer science when the people in AI started to make planning schemas for robots. For the planning of future actions one has to reason about time and then the structure of time becomes an important issue. See [38] and [1]. The reasoning about some situation in the future brought about the now famous *frame* problem. See [26]. We will see later that this problem does not occur in the knowledge base domain, because we assume that the domain is closed and therefore that all the (relevant) aspects of a situation are described. Therefore nothing unexpected can happen and also all things remain the same unless they are said to change.
More about the structure of the time domain is said in [15]. Although the structure of the time domain is of course important for the interpretation of the temporal aspects, it does not have any influence on the examples used in this paper and thus it is left out here.

In section 2, we describe the temporal logic and its interpretation. In [38] it is argued that there are basically two approaches to extend the logic with a temporal aspect. The first is to append a time(interval) to each term. This leads to a syntax as described in [2] and [27]. The alternative is to introduce some modal time operators. This is described in various places like [38, 35] and [7]. In [19], a direct relation is drawn between the modal operators and the tenses in natural language.
In our temporal logic we will use a combination of the "classical" and the modal approach.

The reason for this approach can be found in the syntax of the temporal CPL predications.

Because CPL is based on the linguistic theory of FG, some of the ideas about time representation in natural language are incorporated in CPL.
In linguistic theories three different time aspects are recognized in each sentence. They are: the time of utterance, the reference time and the event time. The time of utterance indicates the time at which the sentence is used. The reference time is usually indicated by a modifier in the sentence. It is used to indicate a time relative to which the event should be evaluated. The event time finally indicates the time at which the proposition should hold. The relation between the reference time and event time is given by the tense of the sentence.
An example will clarify the above.

On August 3rd, 1987, John borrowed a book.

The utterance time is "now", the time this sentence was read.
The reference time is August 3th, 1987, at which time John had already borrowed a book. Therefore, the event time is placed before the reference time.

In CPL only the reference time and the event time are important for the evaluation of the predication. The time of utterance is only important in sentences involving believes and knowledge. Someone may believe at a certain moment that a certain proposition is true at a certain time, while it is not in fact true.
The reference time is indicated in CPL by a term with semantic function tmp, while the relation between the reference time and event time is indicated by `TENSE`.
How these temporal aspects are interpreted in the temporal logic is described further in the next section.

Before we introduce the time aspects of our knowledge base model, two remarks have to be made.
First, by introducing the time aspect, we can not identify the implementation of the knowledge base with the current state in the knowledge base model any more. Somehow, we have to keep track of the information of past states that does not hold any more in the present state. That is, we cannot remove the old information from the knowledge base altogether, because in a later state we must be able to infer that that information held at some time in the past.
In this paper, we are only concerned with the knowledge base model. How the implementation of the knowledge base should be managed to reflect this model is not within the scope of this paper (although it is an interesting research topic). See also Sernadas [37].

The second remark concerns the kind of information that can be added to the knowledge base. In principle, no temporal information about the past states of the knowledge base can be added. The reason is that each state consists of non-temporal information. The temporal aspect of the information is inferred only from the states in which the information holds.

6.2. Temporal logic

As described in [38], there are two different approaches to describe temporal aspects in logic. The first is the "classical" approach, the second is the modal approach. In the "classical" approach, a time(interval) is assigned to every assertion. So, every logical formula becomes a pair of a first-order formula and a time. In the modal approach, some temporal modal operators are introduced. With these operators, the time at which an assertion is to be evaluated can be determined.
In [38], it is shown that the modal temporal logic can be expressed in the "classical" approach.

Although, in terms of expressibility, we could thus use only the "classical" approach, we will still make use of some modal operators as well. This combination of the two approaches follows from the linguistic basis of CPL.
As already said, in linguistics every assertion has a reference time and a event time. The reference time is usually given in a modifier of the assertion and can be easily expressed using the "classical" approach. The event time is given (relative to the reference time) by the tense of the verb. It seems natural to interpret the tense by a modal tense operator.

In the rest of this section we give the syntax of L_{temp}, which is the temporal extension of L_{dyn}. The semantics of this language is given in terms of the knowledge base model as it was already developed for L_{dyn} extended with the time domain.
The syntax of L_{temp} is given in two steps. First the language L_{dyn} is extended with two temporal modal operators *PAST* and *FUT*. The resulting language is called L_{mdyn}. In the second step L_{mdyn} is extended with temporal intervals to form L_{temp}.

The two temporal operators have their usual intuitive meaning. That is, $PAST(\psi)$ is true iff there is a time interval before the present time such that ψ is true during that interval. The meaning of $FUT(\psi)$ is the same, but now for a time interval in the future.
The definition of L_{mdyn} is the same as for L_{dyn}, except that it is not derived from first-order logic but from first-order temporal logic. That is, the temporal operators are introduced at the lowest level.

The language L_{temp} is defined inductively as follows:

Definition:

(1a) Every interval variable is a temporal term

(1b) If t_1 and t_2 are temporal variables or constants, then (t_1,t_2) is a temporal term

(1c) *NOW* is a temporal term

(1d) If f is a temporal function (like intersection etc.) and $tmp_1,...tmp_n$ are temporal terms then $f(tmp_1,...,tmp_n)$ is a temporal term.

(2a) If tmp_1 and tmp_2 are temporal terms then $before(tmp_1,tmp_2)$, $meets(tmp_1,tmp_2)$, $in(tmp_1,tmp_2)$, $eq(tmp_1,tmp_2)$, $over(tmp_1,tmp_2)$, $starts(tmp_1,tmp_2)$ and $finish(tmp_1,tmp_2)$ are elements of L_{temp}.

(2b) If *tmp* is a temporal term and *iexp* is an expression denoting an integer then $length(tmp)=iexp$ is an element of L_{temp}.

(2c) If t is a temporal variable or constant, *cal* is a calendar function and n is an integer then $cal(t)=n$ is an element of L_{temp}.

(2d) If *tmp* is a temporal term and $\psi \in L_{mdyn}$ then $<tmp,\psi>$ is an element of L_{temp}.

(2e) If *tmp* is a temporal term, $\Psi \in L_{temp}$ and β a transaction then $<tmp,\Psi>$ and $<tmp,[\beta]\Psi>$ are elements of L_{temp}.

(2f) If Φ and Ψ are elements of L_{temp} then $\Phi andsign \Psi$, $\Phi orsign \Psi$ and $\neg\Phi$ are elements of L_{temp}.

(2g) If $\Phi \in L_{temp}$ and z is a (time)variable then $\forall z \Phi \in L_{temp}$.

(2h) If $\Phi \in L_{temp}$ and Φ does not contain *MUST*, $\Box$, of L_{temp}.

End Definition

1a-1d actually define how intervals can be expressed in the logic. That is, they can be expressed by variables or by a tuple consisting of a starting point and an end point.
2a-2c express some relations and functions on time intervals (and points). In 2b, for the expression denoting an integer, we can use just an integer, but also the product of an integer and a calendar-type. This product denotes the product of the integer and the number of time points that fit into an interval of the length of the calendar-type. E.g., if a day consists of 86400 time points then the expression "5*day" denotes the integer 432000.
2d and 2e define the relation between the time intervals and the rest of the formula. A tuple $<tmp,\psi>$ denotes a formula ψ that is true at a certain time(interval) *tmp*.
The last point in the definition is needed to express temporal formulas within an epistemic and deontic context. We need these formulas, because in the interpretation of the static CPL predications, the epistemic or deontic operator has scope over the complete (temporal) formula.
In the axioms, which are given at the end of this section, it is shown that the deontic operator can be moved inwards, over the temporal parts of the formula, to get an equivalent formula where this operator has scope only over a non-temporal part of the formula.
In the syntax of L_{temp} we did not formally allow the use of relative time intervals, like "yesterday" or "next year". However, we will use them in the following as abbreviations of temporal terms. We will not give the logical definition of all these relative time intervals. However, to show that they can be formally defined we will give the definition of *yesterday* as an example for all these relative intervals.

Definition:

yesterday is the interval *I* such that $length(I)=1*day\ andsign\ meets(T,today)$
today is the interval (t_1,t_2) such that
$hour(t_1)=0\ \ andsign\ \ minute(t_1)=0\ \ andsign\ \ second(t_1)=0\ \ andsign\ \ length((t_1,t_2))=1*day\ andsign\ in(NOW,(t_1,t_2))$

End Definition

In this definition it is assumed that "second" is the calendar equivalent of the smallest possible interval.

We will not give the definition of the valuation function, because this involves quite a few technicalities and does not add much to the understanding of the logical interpretation.(The interested reader is referred to [15]).

We conclude this section by giving the following axioms that hold for L_{mdyn} and L_{temp}.

Axioms:

(1) $MUST(FUT(\Psi)) <\!\!\rightarrow FUT(MUST(\Psi))$
(2) $MUST(PAST(\Psi)) <\!\!\rightarrow PAST(MUST(\Psi))$
(3) $\neg PAST(\Box\Psi) <\!\!\rightarrow true$
(4) $\Box\Psi \rightarrow FUT(\Box\Psi)$

The first two axioms express the fact that the temporal and deontic operators are independent. That is, they do not influence each other.
It should be noted that this does not hold for the temporal and epistemic operators. The operator "□" could in a temporal context be expressed as
"always, from now on".
The following axioms hold for L_{temp}.

Axioms:

(1) $MUST(<tmp,\Psi>) <\rightarrow <tmp,MUST(\Psi)>$
(2) $MUST(\forall z\Psi) <\rightarrow \forall z MUST(\Psi)$ (z a time-variable)
(3) $\forall x<tmp,\Phi> <\rightarrow <tmp,\forall x\Phi>$ (if x is not a time variable)
(4) $\Box<tmp,\Psi> <\rightarrow <tmp,\Psi>$ (if $tmp \neq NOW$)
(5) $<tmp,\Psi> andsign <tmp,\Phi> <\rightarrow <tmp,\Psi andsign \Phi>$
(6) $<tmp,\neg\Psi> \rightarrow \neg<tmp,\Psi>$
(7) $\forall t((before(tmp,t) orsigneq(tmp,t)) andsign <t,\Psi>) <\rightarrow <tmp,\Box\Psi>$
(8) $<tmp,\Psi> \rightarrow (\forall I: in(I,tmp) \rightarrow <I,\Psi>)$
(9) $<tmp,[\alpha]<tmp+1,DONE:\alpha>>$

The first two axioms again show that the deontic operator *MUST* does not influence the time aspects. That is, if it is obliged that something is true at a certain time then at that time it is obliged to be true.
Axiom 4 is a consequence of the fact that the truth value of formulas with a reference time that is not relative, does not depend on the world in which they are evaluated.
Note that axiom 6 is only a one way implication!
In axiom 7 it is stated that if a formula is true at every time in the future starting at a certain point *tmp*, then the formula is also necessarily true at *tmp*.
Axiom 9 states that at every time *tmp* it holds that if we perform an action α then at time *tmp*+1 it holds that *DONE*:α.

6.3. Temporal CPL predications

In this section the temporal constructions in CPL are described together with their interpretation in L_{temp}.
We start with the description of the `TENSE` and its interpretation into temporal modal operators.

6.3.1. The event time

As can be seen from the syntax of CPL, the `TENSE` can be specified with every simple predication. The interpretation of this case is very straightforward if there are no identifiers present. As is shown in the following examples, the logical interpretation of `MOD:TENSE:SPred` in this cases is (`MOD`)((`TENSE`)((`SPred`))). Where (PRET)=*PAST*, (PROSP)=*FUT* and (PERF) is the identity operator.
The CPL predication:

FACTUAL:PRET:borrow(<1> ag=A **in** person)(<1> go=B **in** book)

has as its logical interpretation:

PAST(∃A'(person(A') andsign ∃B'(book(B') andsign borrow(A',B'))))

When identifiers are present, the interpretation is adjusted in such a way as to avoid nested temporal operators. This can be seen from the following example:

FACTUAL:
PROSP:give(<1> ag=A **in** person)(<1> go=B **in** book)(<1> rec=C **in** person)
(**id:** PRET:has(<1> zero=A **in** person)(<1> pat=D **in** address)(**id:** D=A'dam))
(**id:** PRET:has(<1> zero=C **in** person)(<1> pat=E **in** address)(**id:** E=R'dam))

In the correct interpretation the scope of the temporal operators does not exceed that of the simple predication to which it belongs and they are not nested.
Therefore, the intended interpretation of the above CPL predication is:

$$\exists A'(person(A') \text{ andsign } PAST(\exists D'(address(A',D') \text{andsign} D'=A'dam)) \text{ andsign } \exists B'(book(B') \text{ andsign } \exists C'(person(C') \text{ andsign } PAST(\exists E'(address(C',E') \text{andsign} E'=R'dam)) \text{ andsign } FUT(give(A',B',C'))))))$$

It should be noted that this interpretation may not be completely accurate, because we require the existence of a book at this moment while it only has to exist at the moment it will be given. These kind of problems are dealt with in [15].

We conclude this section with an example of a temporal CPL predication that contains a negation and its logical interpretation. Consider the following CPL predication:

FACTUAL:PROSP:borrow(<1> ag=A **in** person)(<1> go=B **in** book)
(**id:** PRET:~has(<1> zero=A **in** person)(<1> pat=C **in** address)(**id:** C=A'dam))

The predication means that there is a person, that, at some time in the past, did not live in Amsterdam and who will borrow a book (in the future).
Note that the negation does not get scope over the temporal aspect. That is, we do not mean that the person never lived in Amsterdam. But that there is a time in the past such that the person did not live there.
To describe that the person never lived in Amsterdam, we will make use of the reference time, which will be described in the next section.
The logical interpretation of the above predication is:

$$\exists A'(person(A') \text{ andsign } PAST(\forall C'(\neg address(A',C') \text{orsign} C' \neq A'dam)) \text{ andsign } \exists B'(book(B') \text{ andsign } FUT(borrow(A',B'))))$$

6.3.2. The reference time

In this section, we show how the reference time of a predication is interpreted in our temporal logic.
The reference time of a predication is given by the terms with semantic function tmp. An example is:

FACTUAL:
PERF:borrow(<1> ag=A **in** person)(<1> go=B **in** book)(<1> tmp=T **in** day)
(**id:** T=yesterday)

Which means that some person borrowed a book yesterday.
The type of the tmp term determines the unit of the length of the time interval of the reference time. In the example, this interval has a length of 1 day. The minimum length of the time intervals is the time it takes for an action to be performed. According to the application this time can be a second, an

hour, or any other "natural" time interval. The type of this minimal interval is time.
If the type of the temporal term is time, then the unit of the length of the time interval is unspecified in the logical interpretation.
The logical interpretation of the example is:

$\exists T'$: T'=*yesterday andsign*
$\langle T', \exists A'(person(A') \text{ andsign } \exists B'(book(B') \text{ andsign } borrow(A',B'))\rangle$

As can be seen from the logical interpretation, the quantifier of the time variable gets scope over the rest of the quantifiers of the predication. Another difference with the logical interpretation of the other terms is that the `type` of the temporal term has no direct interpretation into logic. It is only used when the length of the time interval is specified. E.g. Last week, a person borrowed a book for (a period of) two days.
This is described in CPL in the following way:

FACTUAL:
PERF:borrow(<1> ag=A **in** person)(<1> go=B **in** book)(<1> tmp=T **in** day)
(**id:** length(T)=2)
(**id:** T in lastweek)

The logical interpretation of this example is:

$\exists T'$: *length(T')=2*day andsign in(T',lastweek) andsign*
$\langle T', \exists A'(person(A') \text{ andsign } \exists B'(book(B') \text{ andsign } borrow(A',B'))\rangle$

So, although the reference time appears as a normal term in the CPL predication, its logical translation is different from the other terms.

It should be noted that the above example may not be true next week, because the reference time is given relative to the present time. If we want to avoid this, we have to give an absolute time as reference time.

FACTUAL:
PERF:borrow(<1> ag=A **in** person)(<1> go=B **in** book)(<1> tmp=T **in** day)
(**id:** T=(16-07-1989,17-07-1989))

which is interpreted as:

$\exists T'$: $T'=(t_1,t_2)$ *andsign*
$date(t_1)$=16-07-1989 *andsign* $\forall t \neq t_1$(*date(t)=16-07-1989* $\rightarrow$ *before*$(t_1,t))$ *andsign*
$date(t_2)$=17-07-1989 *andsign* $\forall t \neq t_2$(*date(t)=17-07-1989* $\rightarrow$ *before*$(t,t_1))$ *andsign*
$\langle T', \exists A'(person(A') \text{ andsign } \exists B'(book(B') \text{ andsign } borrow(A',B'))\rangle$

So, the logical interpretation of the interval (16-07-1989,17-07-1989) is an interval that starts with the first time point of July 16th and ends with the last time point in July 17th.

It is also possible to quantify over the reference time, because it has the same structure as the other arguments of the predication. The following is a good example of such a quantification:

FACTUAL:
PERF:pay(<1> ag=A **in** company)(<1> go=B **in** salary)(<*> tmp=T **in** day)
(**id:** has(<*> zero=T **in** day)(<1> pat=N **in** name)(**id:** N=Friday))

Which means that the salaries are being paid every Friday.
The logical interpretation of this example is as follows:

$$\exists T: \forall I(\exists N'(name(I,N')\ andsign\ N'=Friday) <\!\!\rightarrow I \in T)\ andsign\ \forall I \in T$$
$$<I, \exists A'(company(A')\ andsign\ \exists B'(salary(B')\ andsign\ pay(A',B'))>$$

From the above example it can be seen that the time variable also is a variable which can be instantiated by a set of intervals, just like the other variables can be instantiated by sets of objects.
The logical interpretation of the identifiers on the temporal term is also similar to that of the other terms when the identifiers consist of a predication.

Using the reference time, it is now also possible to express the fact that a person that never lived in Amsterdam borrows a book. (In contrast with the previous section, where we only could express the fact that a person did not live in Amsterdam at some time in the past.)
In CPL, the predication describing this fact is the following:

FACTUAL:
PERF:borrow(<1> ag=A **in** person)(<1> go=B **in** book)(<1> tmp=U **in** day)
(**id:** U=today)
(**id:** PERF:~has(<1> zero=A **in** person)(<1> pat=C **in** city)(<1> tmp=T **in** time)
(**id:** T before U)
(**id:** C=Amsterdam))

The logical interpretation is:

$$\exists U': U'=today\ andsign$$
$$<U', \exists A'(person(A')\ andsign\ \forall T': \neg before(T',U')\ orsign$$
$$<T', \neg\exists C'(city(A',C')\ andsign\ C'=Amsterdam)>\ andsign$$
$$\exists B'(book(B')\ andsign\ borrow(A',B'))>$$

Of course, the temporal term can occur with every simple predication (just like TENSE). Therefore the reference times can also be nested. Note that this makes the specification a "temporal" one in the sense of Kung in [25]. This means that we can refer to different states in the same constraint.
The following example shows this for a predication which has identifiers with a different reference time than that of the Head of the predication.

FACTUAL:
PERF:borrow(<1> ag=A **in** person)(<1> go=B **in** book)(<1> tmp=T **in** day)
(**id:** PERF:work(<1> ag=A **in** person)(<1> pos=C **in** in univ.)(<1> tmp=U **in** year)
(**id:** U=1989))
(**id:** T=16-07-1989)

Which means that there is a person, who works at a university in 1989 and who borrowed a book on July 16th 1989.
The logical interpretation is as follows:

$\exists T'$: $T'=(t_1,t_2)$ andsign
$date(t_1)=16\text{-}07\text{-}1989$ andsign $\forall t \neq t_1$($date(t)=16\text{-}07\text{-}1989 \rightarrow before(t_1,t)$) andsign
$date(t_2)=16\text{-}07\text{-}1989$ andsign $\forall t \neq t_2$($date(t)=16\text{-}07\text{-}1989 \rightarrow before(t,t_2)$) andsign
<T', $\exists A'$(person(A') andsign
$\exists U'$: $U'=(u_1,u_2)$ andsign
$year(u_1)=1989$ andsign $\forall t \neq u_1$($year(t)=1989 \rightarrow before(u_1,t)$) andsign
$year(u_2)=1989$ andsign $\forall t \neq u_2$($year(t)=1989 \rightarrow before(t,u_2)$) andsign
<U',$\exists C'$(univ.(C')andsignwork(A',C'))>
andsign $\exists B'$(book(B') andsign borrow(A',B')))>

Of course, it is also possible to refer to a reference time of another simple predication within the same CPL predication. This is illustrated by the first example that was given in section 3, which describes the fact that if a person borrows a book he must return it within 3 weeks.

MUST:
ACTION:return(<1> ag=A **in** person)(<1> go=B **in** book)(<1> tmp=U **in** time)
(**id:** U before T+3*week)
(**sit:** ACTION:
borrow(<1> ag=A **in** person)(<1> go=B **in** book)(<1> tmp=T **in** time)

The logical interpretation of this example is:

$\forall T'$: <T', $\forall A' \forall B'$(person(A') andsign book(B') andsign
[$borrow(A',B')$]$\exists U'$:$before(U',T+3*week)$ andsign <U',$O(return(A',B'))$>)>

The last example, which concludes this section, is about the special temporal term *NOW*. This temporal term has an equivalent term in CPL. However, it is also used as the reference time of the simple predications that have no explicit reference time (as a term with semantic role tmp).

PERMIT:ACTION:borrow(<1> ag=A **in** person)(<1> go=B **in** book)
(**sit:** PERF:has(<1> zero=B **in** book)(<1> pat=C **in** availability)(<1> tmp=T **in** time)
(**id:** C=present))

The predication states that if a book is present at a certain time then a person is permitted to borrow that book (at that time).
The logical interpretation of this CPL predication is:

$\forall T'$: <T', $\forall B'$(book(B') andsign $\exists C'$(availability(B',C')andsignC'=present)
$\rightarrow$
<*NOW*, $\exists A'$(person(A') andsign P(borrow(A',B')))> >

The temporal term *NOW* is used to refer to the reference time of the formula in which it is embedded. It can easily be seen that the above interpretation is equivalent to:

$\forall T'$: <T', $\forall B'$(book(B') andsign $\exists C'$(availability(B',C')andsignC'=present)
$\rightarrow$
<T', $\exists A'$(person(A') andsign P(borrow(A',B')))> >

7. Conclusions

In this paper a new view on knowledge bases has been described. Although there are already numerous definitions of knowledge bases this view is essentially different in that it starts from linguistics. It is shown that, starting from linguistics, a broad range of different types of knowledge can be described.

By indicating the logical interpretations of the CPL predications, we have shown that the language has a sound semantics. Furthermore, it is possible to define some inference rules within CPL and show that they are sound. That the inference that was shown in section 2 is sound can be seen from the following logical interpretation ((2) stands for the logical interpretation of (2)):

(2) andsign (3) =

*∃T': T'=NOW+3*week andsign*
<T', ∃A'(person(A') andsign <NOW,∃C'(name(A',C')andsignC'=John)> andsign
∃B'(book(B') andsign <NOW,∃D'(booknr(A',D')andsignD'=1081)> andsign
O(return(A',B'))))>
andsign
*∀T': before(T',NOW+4*week) andsign*
<T', ∃A'(person(A') andsign <NOW,∃C'(name(A',C')andsignC'=John)> andsign
∀B'(¬book(B') orsign
O(return(A',B'))))>

⇒

*∃T': T'=NOW+3*week andsign*
<T', ∃A'(person(A') andsign <NOW,∃C'(name(A',C')andsignC'=John)> andsign
∃B'(book(B') andsign <NOW,∃D'(booknr(A',D')andsignD'=1081)> andsign
O(return(A',B')) andsign O(return(A',B'))))>

⇒

*∃T': T'=NOW+3*week andsign <T',O(***fail***)>*

Which is the logical interpretation of (4).

Of course, it is also possible to show that a given knowledge base is consistent by proving the consistency of its logical interpretation. This is not done in this paper, but can be found in [15].

Finally, it should be remarked that not much attention has been given to implementation of the language CPL in this paper. It was considered important to establish a firm theoretical basis for the language first, before considering implementation issues. These could then be tackled starting with a subset of CPL. In [13], it is described how a subset of CPL can be implemented, but a lot of work remains to be done in this respect.

8. Acknowledgements

The authors are grateful to Hans Weigand and Willem Meijs for helpful suggestions.

9. References

1. J. Allen, ‘‘Maintaining knowledge about temporal intervals,’’ *Comm. of the ACM* **26**(11), pp. 832-843 (November 1983).
2. J. Allen, ‘‘Towards a general theory of action and time,’’ *Artificial Intelligence*(23), pp. 123-154 (1984).
3. V. de Antonellis and B. Zonta, ‘‘Modelling events in database applications design,’’ pp. 23-31 in *Proc. of 7th Intl. Conf on VLDB*, Cannes (1981).
4. K. Apt and M. van Emden, ‘‘Contributions to the theory of logic programming,’’ *Journal of the ACM* **29**(3) (1982).
5. R. Bachman and H. Levesque, ‘‘What makes a Knowledge Base Knowledgeable? A View of the database from the knowledge level,’’ in *Proceedings of the 1st Int Workshop on Expert Database Systems*, ed. L. Kerschberg, Benjamin Cummings Publishing Comp. inc. (1986).
6. J. Baeten, *Procesalgebra,* Kluwer, Dordrecht (1986).
7. J. van Benthem, *The logic of time,* Reidel, Dordrecht (1983).
8. A. Bolour, T. Anderson, L. Dekeyser, and H. Wong, ‘‘The role of time in information processing: A survey,’’ *ACM-SIGMOD review* **12**(3), pp. 27-50 (April 1982).
9. M. Brodie, ‘‘Active and passive component modelling: ACM/PCM,’’ in *Information systems design methodologies: A comparative review*, ed. T. Olle, H. Sol, A. Verrijn-Stuart , North-Holland, Amsterdam (1982).
10. C. Capel and D. Westra, ‘‘LIKE,’’ MsC Thesis, VU Amsterdam (1987).
11. N. Chomsky, *Aspects of the Theory of Syntax,* MIT Press, Cambridge Mass. (1965).
12. K. Clark and S. Tarnlund, *Logic Programming,* Academic Press, New York (1982).
13. F. Dignum, H. Weigand, W. Kreuzen, T. Kemme, and R. van de Riet, ‘‘Constraint Modelling using a Conceptual Prototyping Language,’’ *Data & Knowledge Engineering* **2**(3), pp. 213-254 (1987).
14. F. Dignum, ‘‘Parsing an English Text using Functional Grammar,’’ pp. 109-134 in *Functional Grammar and the Computer*, ed. S. Dik and J. Connolly, Foris, Dordrecht (1989).
15. F. Dignum, ‘‘A Language for Modelling Knowledge Bases. Based on Linguistics, Founded in Logic,’’ PhD. Thesis, Centrale Huisdrukkerij VU, Amsterdam (1989).
16. S. Dik , *Functional Grammar,* North-Holland, Amsterdam (1978).
17. D. Dowty, *Word meaning and Montague Grammar,* Reidel, Dordrecht (1979).
18. H. Gallaire, J. Minker, and J. Nicolas, ‘‘Logic and databases: a deductive approach,’’ *Computing surveys (ACM)* **16**(1), pp. 153-185 (June 1984).
19. L.T.F. Gamut, *Logica, taal en betekenis 2 (Intensionele logica en logische grammatica),* Uitgeverij Het Spectrum, Utrecht (1982).

20. R. Hilpinen(ed), *Deontic Logic: Introductory and Systematic Readings*, Reidel, Dordrecht (1988).

21. T. Jansen and P. van Emde Boas, ‘‘Some Observations on Compositional Semantics,’’ pp. 137-149 in *Proceedings workshop on Logic of Programming, Springer Lecture Notes 131*, ed. D. Kozen, Yorktown Heights (1981).

22. S. Khosla, T. Maibaum, and M. Sadler, ‘‘Database specification,’’ in *IFIP TC2 working conference (Database Semantics)*, ed. R. Meersman, J. Steel, North-Holland, Amsterdam (1985).

23. R. Kowalski, ‘‘Logic for data description,’’ pp. 77-103 in *Logic and data bases*, ed. H. Gallaire, J. Minker, Plenum-Press, New-York (1978).

24. R. Kowalski(ed), *Logic Programming, Proc. IFIP Congress 1983* , North Holland, Amsterdam (1983).

25. C. Kung, ‘‘A temporal framework for information systems specification and verification,’’ PhD. Thesis and Report 14/84, University of Trondheim (1984).

26. J. McCarthy and P. Hayes, ‘‘Some philosophical problems from the standpoint of artificial intelligence,’’ pp. 463-500 in *Machine intelligence 4*, ed. Meltzer, Michie, Edinburgh university press (1969).

27. D. McDermott, ‘‘A temporal logic for reasoning about processes and plans,’’ *Cognitive Science*(6), pp. 101-155 (1982).

28. J-J. Meyer, H. Weigand, and R. Wieringa, ‘‘Modeling Dynamic and Deontic Integrity Constraints in Knowledge Bases,’’ *Data & Knowledge Engineering* **4**(4) (1989).

29. J-J. Meyer, ‘‘A Different Approach to Deontic Logic: Deontic Logic Viewed as a Variant of Dynamic Logic,’’ *Notre Dame Journal of Formal Logic* **29**, pp. 109-136 (winter 1988).

30. J-J. Meyer, ‘‘Using Programming Concepts in Deontic Reasoning,’’ in *Semantics and Contextual Expression*, ed. R. Bartsch, J. van Benthem, P. van Emde Boas, Foris , Dordrecht (to appear).

31. A. Olivé, ‘‘DADES: A Methodology for Specification and Design of Information Systems,’’ in *Information systems design methodologies: A comparative review*, ed. T. Olle, H. Sol, A. Verrijn-Stuart , North-Holland , Amsterdam (1982).

32. R. Reiter, ‘‘Towards a logical reconstruction of relational database theory,’’ pp. 191-233 in *On conceptual modelling*, ed. M. Brodie et. al., Springer, Berlin (1984).

33. R. Reiter, ‘‘Foundations for Knowledge-based Systems,’’ in *Information Processing*, ed. H. Kugler, Elsevier Science Publishers B.V., Amsterdam (1986).

34. R. Reiter, ‘‘On Integrity Constraints,’’ pp. 97-111 in *Proc. of the Second Conf. on Theoretical Aspects of Reasoning about Knowledge*, ed. M. Vardi, Morgan Kaufmann (1988).

35. P. Roper, ‘‘Intervals and tenses,’’ *Journal of Philosophical Logic*(9) (1980).

36. D. Scott, ‘‘Domains for Denotational Semantics,’’ in *Springer, Lecture notes in Computer Science, 140* (1982).

37. A. Sernadas, ‘‘Temporal aspects of logical procedure definition,’’ *Information Systems* **5**(3), pp. 167-187 (1980).

38. Y. Shoham, *Reasoning about change*, MIT-Press, Cambridge, Mass. (1988).

39. B. Smith, ‘‘Reflection and Semantics in a Procedural Language,’’ PhD Thesis and Report MIT/LCS/TR-272, MIT, Cambridge, Mass. (1982).

40. A. Solvberg and C. Kung, ‘‘On structural and behavioral modelling of reality,’’ in *IFIP TC2 working conference (Database Semantics)*, ed. R. Meersman, J. Steel, North-Holland, Amsterdam (1985).

41. G. Verheijen and J. van Bekkum, ‘‘NIAM: an information analysis method,’’ in *Information systems design methodologies: A comparative review*, ed. T. Olle, H. Sol, A. Verrijn-Stuart , North-Holland , Amsterdam (1982).

42. H. Weigand, ‘‘Linguistically motivated principles of knowledgebase systems,’’ PhD Thesis, Foris, Dordrecht (1989).

43. G. Wiederhold, ‘‘Knowledge Bases,’’ *Int. Symp. on Fifth Generation and Super Computers*, Rotterdam (Dec. 1984).

44. T. Winograd , *Language as a Cognitive Process,* Addison-Wesley, Amsterdam (1983).