Knowledge base systems : a formal model

Eindhoven University of Technology

Department of Mathematics and Computing Science

Knowledge Base Systems, a Formal Model

by

A.E. Eiben and R.V. Schuwer

Computing Science Note 91/20
Eindhoven, September 1991

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. F. van Neerven
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

Knowledge Base Systems, a Formal Model

A.E. Eiben [1] and R.V. Schuwer [2]

[1] Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

[2] Department of Industrial Engineering and Management Science
Section Management Information Systems and Automation
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

Knowledge base systems (KBS's) are gaining popularity among software users and developers. It is, however, not clear what a KBS is and which functions it has to fulfill. In this paper we investigate the notion of what could be called a rule based KBS. We wish to answer two main questions. First, we want to determine what is a KBS intended for, i.e. what should it compute. The second question is, which components are needed in KBS to compute what is intended. Based on logic programming and deductive database theory we elaborate a model of KBS's in which we can answer these questions. Applying the model for examining software tools we observe that it serves as a good guide in evaluating and developing KBS's.

1 Introduction

Knowledge base systems (KBS's) can be considered as the link between theoretical AI, deductive databases and practice. There are several good software tools that are classified as a KBS, but it is in no sense standardized what a KBS is, or which functions it has to fulfill. Definitions of a KBS are mostly informal, like the following one from [MAR88]:

"A knowledge base system is a computer program, where application specific knowledge and application independent deduction rules are separated as good as possible."

A closer look on this informal definition discloses that it is too vague. Therefore, it cannot serve as a guide in evaluating software, or in designing a KBS. Another deficiency of this approach is that taking 'application specific' and 'application independent' as a basis of knowledge organization is a risky choice. Namely, the notion of 'application' is quite arbitrary and so is the border between application specific and application independent.

We take a more formal approach. First, we explicitly identify knowledge base systems with rule based systems. Thereafter we classify knowledge by the following hierarchy:

- The Universe of Discourse, eg. the world of printers.
- Factual (situation specific) knowledge, eg. the symptoms of a broken printer.
- Instructive knowledge that applies to more situations of the same type, e.g. fault detection instructions for printers. This knowledge utilizes factual knowledge, and is often in the form of implications, eg. symptoms ⇒ fault.
- Meta knowledge that does not concern the modeled world, but the system itself, in particular, the usage of the formerly mentioned knowledge. Meta knowledge is meant to steer the reasoning process of the system and it is sometimes expressed in the form condition ⇒ action.

The distinction between instructive and meta knowledge is not always made, as it turns out from the next example, cf. [FLV88]:

if	(the amplified of given_line = yes **and**		
	the type_class of a_modem = LINE_DRIVER)	**or**	
	the type_class of a_modem = BASE_BAND		
then	set the value of a_modem to removed	**and**	(1)
	forget all following rules from the selection	**and**	(2)
	try all rules from context FIND_MODEM from this_script		(3)

Hereby (1) leads to a new fact, while (2) and (3) are obviously part of the process control.

In this article we do not consider the 'administrative' functions and components of a KBS, such as user interface, explanation module, etc. We do not consider certainty factors either,

although they play an important role in several knowledge base systems. We regard the use of certainty factors simply as assigning numbers to formulae and coupling a numeric computation to the logical deduction. We concentrate on one single module and we set up a framework that formally describes the 'thinking kernel' of a KBS. Naturally, we do not claim that this is the only possible formal interpretation of KBS's, but it *is* a formal one and it answers our two basic questions, that is

1) What is a KBS intended for, i.e. what should it compute?
2) Which components are needed in a KBS to compute what is intended?

Our approach has a flavor of deductive database theory and logic programming [LLO 87], [MIN 88] and can be characterized as a proof-theoretical view (as opposed to a model-theoretic view cf. [GMN84]).

As the results of Section 6 indicate our answers to the above questions turned out to be very useful when evaluating existing software tools.

2 Basic notions

A **language skeleton** or **database skeleton** is a triple $S = (F, P, ar)$, where F and P are disjoint finite sets of function symbols and relation symbols respectively; $ar : F \cup P \rightarrow \mathbb{N}$ is the arity function. Constants are function symbols with zero arity.

A **term** is defined as follows:

- every constant is a term;
- every variable is a term;
- for $f \in F$ with $ar(f) = n$ and terms $t_1,..,t_n$: $f(t_1,..,t_n)$ is a term.

There are no other terms.

An **atom** is defined as follows:

for $p \in P$ with $ar(p) = n$ and terms $t_1,..,t_n$: $p(t_1,..,t_n)$ is an atom. When none of the terms contains a variable, an atom is called **ground**.

For an atom a, both a and $\neg a$ are **literals**. a is a positive literal, whereas $\neg a$ is a negative literal.

The notations T(S), A(S) and L(S) stand for the set of all terms of S, the set of all atoms of

S and the set of all literals of S, respectively. If E is a set of expressions then [E] denotes the set of all ground instances of the elements of E. Throughout this paper a **rule** means a normal clause with a not empty body. A **rule-base** is a finite set of rules.

Let r be the rule $A \leftarrow L_1 \wedge \ldots \wedge L_n$ and let u stand for a finite set of literals. The **immediate consequences** of u via the rule r are

$T_{\{r\}}(u) = u \cup \{ A\theta \mid \theta \text{ is a substitution, } \{L_1\theta, \ldots, L_n\theta\} \subseteq u \}$,

where T has its standard meaning from [LLO87]. In accordance with the commonly used term "applying a rule to a database" we introduce r(u) as a shorthand for $T_{\{r\}}(u)$.

Example 0

Let $u = \{ p(a), p(b) \}$ and let r be the rule $q(x) \leftarrow p(x)$.

Then $r(u) = u \cup \{q(a), q(b)\}$ □.

3 Definition of a knowledge model

In this section we develop the notion of a knowledge model to answer our first question: what do we want to compute. We follow the terminology of [BRO89], in order to show the similarities between aspects of our model and the database model of [BRO89].

Let S be a database skeleton. The **database universe** (DB universe) belonging to S is:

$U(S) = \{ u \subseteq [L(S)] \mid$ there is no $L \in u$ such that $L \in u$ and $\neg L \in u \}$.

The **database states** (DB states) are the elements of U(S).

Observe the nature of our database notion. A language skeleton S determines the set L(S) as the set of all elementary statements. Database states are consistent ground subsets of L(S), the set of all possible database states is the database universe U(S). Notice that with considering only ground databases we do not impose any serious restriction, since any time a literal L(x) would occur we can replace (represent) it by [L(x)]. A database state $u \in U(S)$ is considered as a collection of true statements, that is if we have $u = \{L_1, \ldots, L_n\}$ then we assume that each $L_1, \ldots, L_n$ is true.

Obviously one might not want to consider all the possible database states, but only those ones that satisfy some conditions. This leads to the following notion. If S is a database skeleton, U(S) is its database universe, then an integrity constraint is a function

$c : U(S) \rightarrow \{TRUE, FALSE, UNDEF\}$.

We assume that every constraint is specified by a first order formula φ (not necessarily in the language of S) in the following way:

$$c_\varphi(u) = \begin{cases} TRUE & \text{if} \quad u \models \varphi \\ FALSE & \text{if} \quad u \models \neg\varphi \\ UNDEF & \text{otherwise} \end{cases},$$

where $u \models \varphi$ and $u \models \neg\varphi$ is decided by identifying u with the conjunction of its literals. Notice that additionally to the usual truth values TRUE and FALSE, we also have UNDEF that stands for undefined, thus 3-valued logic is used. The value UNDEF occurs for instance if u does not contain enough information to tell the truth value of φ.

Example 1

If $U(S) = \{ \{p(a)\},\{p(a),q(b)\},\{p(a),\neg q(b)\},...\}$, φ is $q(b) \leftarrow p(a)$, then

$c_\varphi(\{p(a),q(b)\}) = TRUE$

$c_\varphi(\{p(a),\neg q(b)\}) = FALSE$

$c_\varphi(\{p(a)\}) = UNDEF.$

In the last case q(b) is not an element of the DB state. Therefore, no assertion can be made about the truth of φ w.r.t. $\{p(a)\}$, consequently the value $c_\varphi(\{p(a)\})$ is undefined. □

Example 1 also illustrates the importance of the distinction between rules and constraints. Applying φ as a rule, we obtain q(b) deduced from p(a), even though φ was meant as defining formula of a constraint, that is for control. The crucial difference between rules and constraints is in their purpose: rules are objects to deduce new facts with, while constraints are to filter the DB universe. The reason why constraints are frequently mixed up with rules is twofold:

- the constraint c_φ and its defining formula φ are often not distinguished,
- φ can be an implication in the language of S, in which case it looks like a rule.

Let U(S) be a database universe, C a set of constraints. A $u \in U(S)$ is <u>a feasible database state</u> with respect to C, if

$\forall c \in C : c(u) = TRUE \vee c(u) = UNDEF.$

The <u>feasible database universe</u> corresponding to C is defined as

$U = U(S) \upharpoonright C = \{ u \in U(S) \mid u \text{ is feasible w.r.t. } C \}.$

A knowledge base is usually described as a set of 'facts and rules'. We maintain the same notion in our terminology. We define a knowledge base as a pair (u,R) of explicitly stored

information (u) and some rules (R) to add information implicitly. Given a language skeleton S, and a set of constraints C, a **knowledge base** is a pair (u,R), where
– $u \in U(S) \upharpoonright C$ is a feasible database state,
– R is a rule base, where all the rules are in the language of S.
In the sequel we do not mention the presence of S, C and R; if not stated otherwise, U will denote the feasible DB-state $U(S)\upharpoonright C$.

To obtain the implicit information the rules of R can be used to extend u with deduced facts to get a new (feasible!) database state u'. This can be repeated for u' to get u" etc. until we reach the so called deductive closure of u. If $u \in U$, R is a rule base, then a **deductive closure** of u by R is a set $u \cup v \subseteq [L(S)]$ such that
- $u \cup v \in U$ (is consistent and feasible),
- $L \in v \Rightarrow \exists r_1,..,r_j \in R : L \in r_j(..(r_1(u)))$ and
- $u \cup v$ is maximal within [L(S)].
Observe that $v \subseteq [A(S)]$ always holds since only atoms are allowed in the head of clauses.

At this point we make no assertions about how new facts are derived, nor how consistency and feasibility are preserved.

The purpose of a rule-base is to specify the facts (knowledge) contained in its deductive closure. Roughly speaking we can say that a knowledge base 'means' its deductive closure.

Let us observe again the notion "applying a rule to a DB-state". As was defined before, when a rule is applied to a DB-state, all facts that can be derived then, are actually derived "in one atomic action". One could say that the granularity for deducing facts is the DB-state. The resulting DB-state should be feasible then, otherwise the rule is not allowed to be used at all on the original DB-state. (Actually the original DB-state can be considered to be not feasible, as is elaborated in [SCH91]). In practice however, when applying a rule to a DB-state, some software deduce one fact at a time ("granularity for deduction is fact") and repeat this until the resulting DB-state would not be feasible anymore. This can lead to serious trouble, as can be seen in the following example.

Example 2
Let S = (F, P, ar) a language skeleton, with F = {a, b}, P = {p, q}, ar(a) = ar(b) = 0, ar(p) = ar(q) = 1 and
$C = \{ c_\varphi \}$ where $\varphi \Leftrightarrow \forall x,y \in F : p(x) \wedge p(y) \Rightarrow x = y$. (There is at most one p).

Let $u = \{q(a), q(b)\}$ and $R = \{r\} = \{ p(x) \leftarrow q(x) \}$.
According to the definition of r(u), here $r(u) = \{q(a), q(b), p(a), p(b)\}$, which is not feasible. When the granularity for deduction is fact, then there are two different sets that satisfy the definition of the deductive closure: $\{q(a), q(b), p(a)\}$ and $\{q(a), q(b), p(b)\}$.
Observe that none of these sets can be extended any further; although p(b), respectively p(a) is deducible by R, the resulting DB state is not feasible. So in the latter case the result of r(u) depends on how the system computes the deductive closure. This is not a desirable situation: one has to consider implementation details on a conceptual level. □

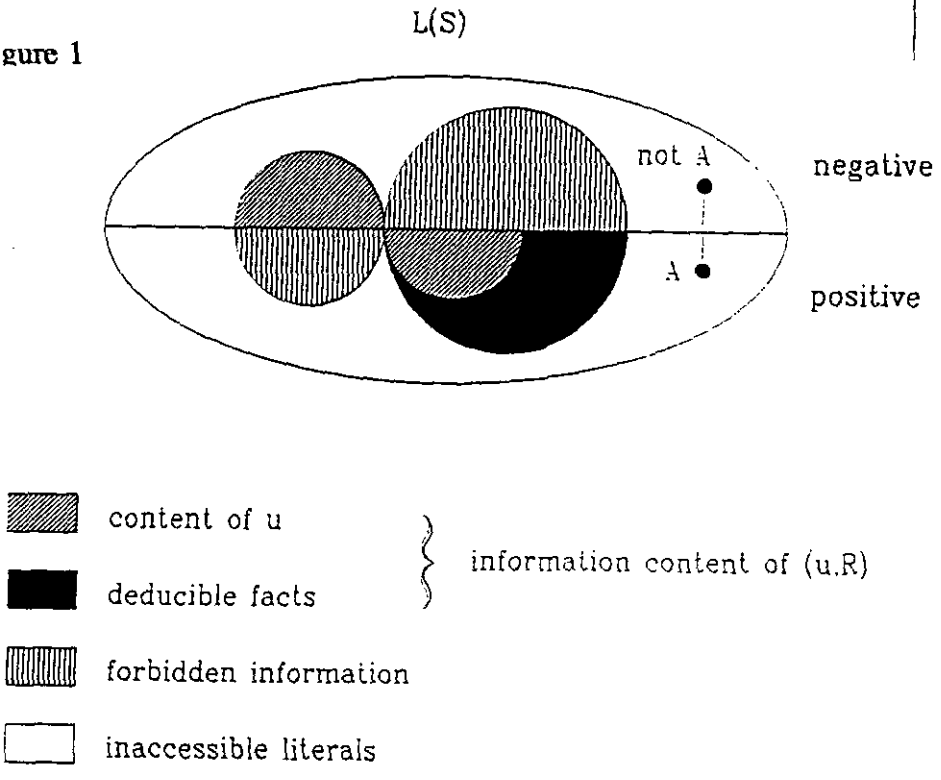
The next objective of this section is to give a characterization of the set of all facts with respect to a rule base. We approach the question 'from outside' and introduce the notion of a knowledge function to embody the concluded-from-the-DB-state relation 'in one go'. A **knowledge function** $k_R : U \rightarrow U$ is a function where for every $u \in U$ $k_R(u)$ is a deductive closure of u via R.

Building a KBS one has to specify a model that describes the possible facts, the possible conclusions (implications) and specifies the conclusions that are actually made. These can be specified by a **knowledge model** that is a triple (U, R, k_R), where
- $U = U(S)\restriction C$ is the feasible database universe,
- R is a rule base, where all the rules are in the language of S,
- $k_R : U \rightarrow U$ is the knowledge function.

Notice that we use U as a primitive although it could be defined as a derivate of a language skeleton S and a set of constraints C. In practice, however, the concept of a DB universe will be more useful.

We look upon a knowledge model as the outside of a KBS that identifies ***what*** we intend to compute by a rule-base within this knowledge model. Formally, if (U, R, k_R) is a knowledge model and (u,R) is a knowledge base ($u \in U$), then the set L(S) can be divided into the following subsets.

Figure 1



- $k_R(u)$ a maximal set of literals that can be deduced from u by R, thus we call it the information content of (u, R); u contains the explicitly stored facts, $k_R(u) \setminus u$ contains the implicit information, stored by R, unfolded by k_R. $k_R(u) \setminus u$ only contains positive facts because of the facts, that only positive literals are in the head of a rule.
- $\{ \neg L \mid L \in k_R(u) \}$ is the set of forbidden information. Due to the consistency of $k_R(u)$, literals from $\{ \neg L \mid L \in k_R(u) \}$ are all false.
- $k_R(u) \cup \{ \neg L \mid L \in k_R(u) \}$ is the range of knowledge of (u, R). This set contains all those literals the truth value of which can be determined by (u,R), k_R and the consistency requirement.
- $L(S) \setminus (k_R(u) \cup \{ \neg L \mid L \in k_R(u) \})$ is the set of inaccessible literals. No statement can be made about these literals based on (u,R) and k_R.

Now we can answer our first question (see Section 1). In our view a knowledge base (u,R) is built to represent its information contents, it 'means' $k_R(u)$. (One could also say that a knowledge base represents its whole range of knowledge, but this would not make much difference.) A KBS is considered as a tool that supports the definition of a knowledge base (u,R) and the computation of $k_R(u)$. Analysis of this latter will be the topic of the next section.

4 Definition of a knowledge base system

Here we integrate dynamic features by not only telling which conclusions can be drawn, but also specifying how to draw the conclusions. This will show the 'inside' of k_R, and at the end we will have answered our second question by having defined the computational components of a knowledge base system.

In the sequel we assume that a knowledge model (U, R, k_R) is given. The working of a knowledge base system can informally be stated as follows. Having a knowledge base (u,R) the system is given a conjunction of literals as a query. To answer the query, the truth value of each literal has to be decided on the basis of (U, R, k_R).

Formally a query is a conjunction $g = L_1 \wedge \ldots \wedge L_n$, its literals will be called hypotheses. If it can not lead to confusion we identify $g = L_1 \wedge \ldots \wedge L_n$ with the set $\{L_1, \ldots, L_n\}$. When given a query the system has to determine in which of the subsets of figure 1 each of the hypotheses fall. If L_i falls in $k_R(u)$ then L_i is true, if it falls in the set of forbidden information then L_i is false and so is g. If no L_i is forbidden and an L_i is inaccessible, then we can not make an assertion about L_i, nor can we about g. We remark that hypotheses may have variables. In this case it is understood that they are quantified existentially and an appropriate substitution is also expected in the answer.

As it turns out from the above, to answer a query the system may have to compute at least a part of $k_R(u)$. This is done by a reasoning process, consisting of elementary inference steps where both the DB state (known literals) and the goal (hypotheses still to be proved) can be changed. Any phase of such a reasoning process can be described by a pair (u,g), where u is the actual DB state, g is the goal. Such a pair (u,g) is called a reasoning state. Considering the actual goal as the set of hypotheses we still need to decide about, we can regard a reasoning process as an attempt to reduce the goal to empty. A reasoning chain within the knowledge model (U, R, k_R), is a sequence $((u_0,g_0),..,(u_k,g_k))$ of reasoning

states such that $k \in \mathbb{N}$ and for every i, $1 \leq i \leq k$:

a) $u_{i-1} \subseteq u_i \subseteq k_R(u_{i-1})$ (no loss of information)

b) 1) $g_i = g_{i-1}$, or

2) $g_i = (g_{i-1} \setminus A)\theta$ such that $A\theta \in k_R(u_{i-1})$ for an atom $A \in g_{i-1}$, or

3) $g_i = (g_{i-1} \cup \{L_1,..,L_n\})\theta$ where $A \leftarrow L_1,..,L_n \in R$ and $A\theta \in g_{i-1}$,

c) $(u_{i-1},g_{i-1}) \neq (u_i,g_i)$,

where θ is a ground substitution.

Next we will have a look on how a transition from (u_{i-1},g_{i-1}) to (u_i,g_i) can be made. Let $\mathcal{R} = \{ (u,g) \mid u \in U, g \text{ is a goal in } S \}$. For every transition in a reasoning chain a hypothesis from g and a rule from R has to be chosen. Therefore we need:

a <u>goal selection function</u> $\gamma : \mathcal{R} \rightarrow L(S)$, such that $\gamma((u,g)) \in g$ and

a <u>rule selection function</u> $\rho : \mathcal{R} \rightarrow R$.

The set of goal selection function will be denoted as G, while RS stands for the set of rule selection functions.

An <u>inference rule</u> (I-rule) is a function $i : \mathcal{R} \rightarrow \mathcal{R}$ that generates a new reasoning state. We can distinguish the following cases where i_1 satisfies (a) and (b_1), i_2 satisfies (a) and (b_2) and i_3 satisfies (a) and (b_3) from the definition of reasoning chain.

1. knowledge base extended

$i_1((u,g)) = (u \cup \{A\theta\}, g)$ where $A\theta \in k_R(u) \setminus u$, $A \leftarrow L_1, \ldots, L_n \in \rho((u,g))$ for a $\rho \in RS$, $\{L_1,...,L_n\}\theta \subseteq u$.

2. hypothesis proved

$i_2((u,g)) = (u, (g \setminus \{A\})\theta)$ if $A \in \gamma((u,g))$ for a $\gamma \in G$ and $A\theta \in u$, or

$i_2((u,g)) = (u \cup \{A\theta\}, (g \setminus \{A\})\theta)$ if $A \in \gamma((u,g))$ for a $\gamma \in G$, $A\theta \in k_R(u) \setminus u$, and $A \leftarrow L_1, \ldots, L_n \in \rho((u,g))$ for a $\rho \in RS$, $\{L_1,...,L_n\}\theta \subseteq u$.

3. goal reformulated

$i_3((u,g)) = (u, (g \cup \{L_1,...,L_n\})\theta)$ where $A \in \gamma((u,g))$ for a $\gamma \in G$, $A \leftarrow L_1, \ldots, L_n \in \rho((u,g))$ for a $\rho \in RS$ and $A\theta \in g$.

It is easy to see that for any inference rule i and reasoning state (u,g), the sequence ((u,g), i((u,g)) is a reasoning chain.

We are now able to give a first provisional notion of a knowledge base system (KBS). Given a knowledge base (u,R), a KBS is a computer program to compute the range of knowledge of (u,R). For this purpose a KBS consists of sets G and RS and an inference procedure, which takes as arguments the sets G and RS and which generates a reasoning

chain that proves g if possible. The following pre- and postconditions determine such an inference procedure IP:

{ (u,R) is a knowledge base, g is a goal, $u_0 = u$, $g_0 = g$ }
IP(G,RS)
{ $\exists\, n \in \mathbb{N}$: $((u_0,g_0),...,(u_n,g_n))$ is a reasoning chain }

Notice that from the pre- and postconditions it is clear to see, that an inference procedure only produces correct reasoning chains (soundness). Conversely, we aim to reach the situation where $g_n = \emptyset$. We therefore define a **perfect** inference procedure IP as follows:

For every input (u_0,g_0):
if there exists a reasoning chain $((u_0,g_0),...,(u_n,\emptyset))$
then the postcondition for IP is: { $\exists\, n \in \mathbb{N}$: $((u_0,g_0),...,(u_n,\emptyset))$ is a reasoning chain }.

In the foregoing we implicitly assumed that u was present. In practice, however, it happens frequently that the system questions the user during a reasoning process. We do not interpret this as a reasoning action but rather as completing $u \in U$ by information that was not stored beforehand.

There are situations where it is not enough to be able to compute only the range of knowledge of (u,R). This is the case when for a hypothesis $L \in g$ holds that $L\theta \in k_R(u)$ or $\neg L\theta \in k_R(u)$ for no substitution θ. In this case L belongs to the set of inaccessible literals w.r.t. $k_R(u)$, thus the truth value of L can not be decided by the knowledge model. In practice one often turns to some rule beyond the knowledge model to obtain an answer for such an L. An **external rule** (E-rule) is a partial function that assigns a truth value to some of the inaccessible literals:

$e : \{ L \in [L(S)] \mid L \notin k_R(u) \wedge \neg L \notin k_R(u) \} \nrightarrow \{\text{TRUE, FALSE, UNDEFINED}\}$.

Example 3
The well-known Closed World Assumption (CWA) is the following E-rule:
$e_{CWA} : \{ L \in [L(S)] \mid L \notin k_R(u) \wedge \neg L \notin k_R(u) \} \rightarrow \{\text{TRUE, FALSE}\}$ such that

$$e_{CWA}(L) = \begin{cases} \text{TRUE} & \text{if } L = \neg A, \text{ for an } A \in A(S) \\ \text{FALSE} & \text{if } L \in A(S) \end{cases};$$

□

Implicitly the "classical" CWA-definition assumes the presence of only positive literals into the database. In that case the first line of the definition is not needed.

Example 4

The Negation as Finite Failure rule can be formulated as follows

e_{NFF} : { $\neg A \in [L(S)] \mid A \in A(S) \setminus k_R(u)$ } $\rightarrow$ {TRUE, FALSE} such that

$$e_{NFF}(\neg A) = \begin{cases} \text{TRUE} & \text{if all the attempts to prove A finitely fail} \\ \text{FALSE} & \text{if otherwise} \end{cases}$$

□

The presence of E-rules has consequences for the earlier given definitions of reasoning chain and I-rules and thus also for the notions of inference procedure and KBS. With E-rules namely it is possible to deduce facts that are not element of the range of knowledge of (u,R). One can say that with the presence of E-rules the range of knowledge of a KBS (the information which can be computed with a KBS) includes the range of knowledge of (u,R). For a KBS with E-rules we therefore need to adapt the definitions of reasoning chain, I-rules and inference procedure slightly.

An extended reasoning chain within the knowledge model (u,R,k_R) is a sequence $((u_0,g_0),...,(u_k,g_k))$ of reasoning states such that condition b) and c) of the definition of reasoning chain holds and the condition, that for every i, $1 \leq i \leq k$: $u_{i-1} \subseteq u_i$.

A function i: $\mathcal{R} \rightarrow \mathcal{R}$ is an extended I-rule if and only if one of the following conditions holds :

- i is an I-rule
- $i((u,g)) = (u \cup \{A\theta\},(g\setminus\{A\})\theta)$ if $A \in \gamma((u,g))$ for a $\gamma \in G$, $A\theta \notin k_R(u)$, $\neg A\theta \notin k_R(u)$ and $\exists\, e \in E : e(A\theta) = \text{TRUE}$, where θ is a ground substitution.

The set of extended I-rules will be denoted as I^*. The function i from the second condition will be denoted as i^*.

When an inference procedure is also able to use the set E as an argument, we will call it an extended inference procedure. It produces extended reasoning chains as can be seen by the pre- and postconditions of an extended inference procedure IP^* :

{ (u,R) is a knowledge base, g is a goal, $u_0 = u$, $g_0 = g$ }

$IP^*(G,RS,E)$

{ $\exists\, n \in \mathbb{N}$: $((u_0,g_0),...,(u_n,g_n))$ is an extended reasoning chain }

The definition of a perfect extended inference procedure is mutatis mutandis the same as that from a perfect inference procedure.

We are now able to give the ultimate definition of a knowledge base system. Let us suppose we have a language skeleton S and a set of constraints C. A **knowledge base system** (or rather, the computational part of a knowledge base system) consists of:

- U a feasible DB universe,
- $u \in U$ a feasible DB-state (set of data)
- R a rule base
- G a set of goal selection functions
- RS a set of rule selection functions
- E a set of E-rules and
- $IP^*(G, RS, E)$ an extended inference procedure.

The knowledge hierarchy of the introduction can be precisely formulated now:

- U contains the knowledge about the Universe of Discourse,
- u contains situation specific knowledge (facts),
- R is the instructive knowledge,
- G, RS, E, and $IP^*(G,RS,E)$ form the meta-knowledge.

The set I^* of extended I-rules gives a framework for a KBS. The reasoning chain, which will be generated by the extended inference procedure can be described by a convolution of elements from I^*. Strategies, with which the extended inference procedure works, can sometimes have an impact on the extended I-rules, which are used to describe the resulting reasoning chain. With forward chaining, for example, only inference rules i_1, i_2 and i^* will be used. Backward chaining uses i_2, i_3 and i^*. In fact, a reasoning chain reflects the way of reasoning which has to be made for arriving at the conclusion (which is a proof of g, if possible). To put it in another way: the reasoning chain reflects the proof for a conclusion to which the system has arrived. This proof may not be the same as the process of the

reasoning. In particular it is possible that during this process one goes into a dead-end direction. One than turns back to an earlier point in the reasoning process to continue into another direction (backtracking). Such sideways will not be found back into the resulting reasoning chain. A consequence of this is that all the facts, that were deduced when going to the sideway, will be removed from the database when turning back to that earlier point. When some of these removed facts will be needed later on in the reasoning process, they will be deduced again and so will be found into the resulting reasoning chain. The example of the next chapter will clarify this.

5 An Example

As an illustration for the former given model and its components consider the following example of a Prolog-like knowledge base system.

Given is a knowledge base system with the following content:

G = {"Select hypotheses according to the textual order"}
RS = {"Select a rule according to the textual order"}
E = { "If the hypothesis is a ground hypothesis, then answer is 'False', else answer is 'No solution'" (1),
"Use the NFF-rule for a negative hypothesis" (2)}

IP^*(G,RS,E) =
{"Select rules from R according to the backward chaining strategy" (1)
"If hypothesis_to_solve is selected
Look into database for unification
If no success
Select rule_to_use
Endif
Endif" (2)
"If no rule_to_use can be found
Backtrack
Endif" (3) }

Suppose the following knowledge base is given (in which rules about family relations and facts about the Dutch Royal Family):

R ={ father(X,Y) <– relation(X,Z),mother(Z,Y). (1)
relation(X,Y) <– divorced(X,Y). (2)
relation(X,Y) <– married(X,Y). (3)
brother(X,Y) <– mother(Z,X), mother(Z,Y), X ≠ Y. (4) }

u ={ mother(beatrix,willem_alexander), (1)
mother(beatrix,johan_friso), (2)
married(claus,beatrix) (3) }

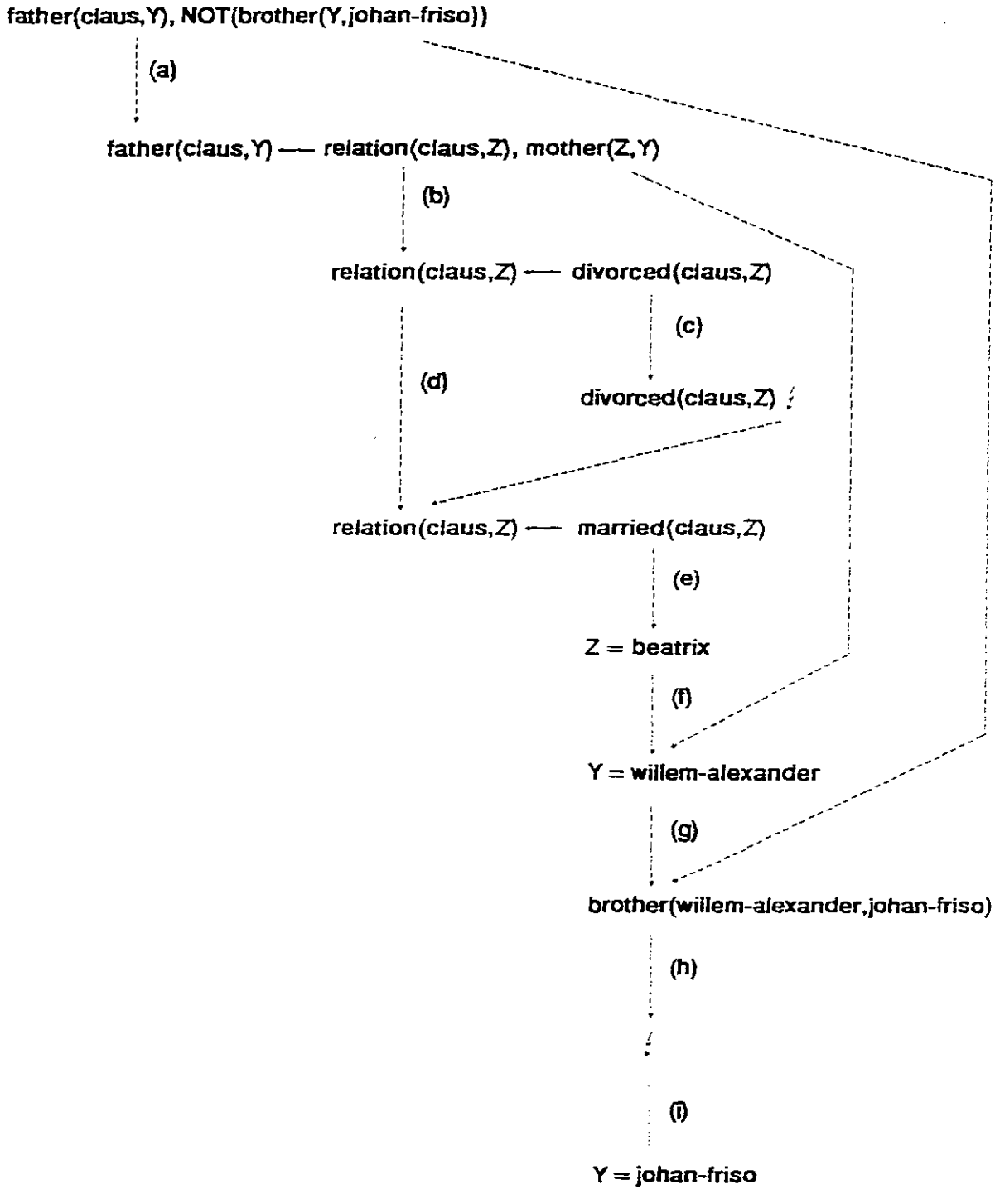
The following goal is offered to the system (with thanks to the Dutch comedians Walden and Muijselaer for their inspiration):

g ={ father(claus,Y), not(brother(Y,johan_friso)) }

(Prince johan_friso asks: "Who is a son of my father (Claus), but is not my brother ?")

Figure 2 shows schematically which procedure will take place to solve the goal.

Figure 2


father(claus,Y), NOT(brother(Y,johan-friso))
(a)
father(claus,Y) — relation(claus,Z), mother(Z,Y)
(b)
relation(claus,Z) — divorced(claus,Z)
(c)
(d)
divorced(claus,Z)
relation(claus,Z) — married(claus,Z)
(e)
Z = beatrix
(f)
Y = willem-alexander
(g)
brother(willem-alexander,johan-friso)
(h)
(i)
Y = johan-friso

Comment: (where "IP" is short for "IP*(G,RS,E))

a. According to the goal-selection rule this hypothesis will be proven first. Because no unification can be found, according to metarule (2) from IP, a rule will be selected from R. Because the process of solving the goal is done according to the backward-chaining strategy (metarule (1) from IP), rule (1) is selected as into this rule the head matches with the hypothesis to be proven. The clauses into the tail of the rule will be put into g (again because of the BW-chaining strategy).
b. Goal-selection gives this hypothesis first. The metarule from RS gives this rule to use as a result.
c. No unification can be found, nor a rule to use (using (2) from IP). Therefore backtracking takes place (metarule (3) from IP).
d. According to RS.
e. According to (2) from IP a unification can be found. The goal-selection metarule gives hypothesis mother(beatrix,Y).
f. (2) from IP gives a unification.
g. Metarule 2 from E gives this hypothesis.
h. Via a similar way of reasoning as before this hypothesis can be proven. Because of this and metarule (2) from E the original hypothesis is FALSE.
i. Backtracking and resuming the process of reasoning at the end gives the result.

Parts of the reasoning chain, which will be built for the above query:

Starting reasoning state:
u: { mother(beatrix,willem_alexander), mother(beatrix,johan_friso),
married(claus,beatrix) }
g: { father(claus,Y), not(brother(Y,johan_friso)) }

Reasoning state after step (a):
I-rule i3:
u: unchanged
g: { relation(claus,Z), mother(Z,Y), father(claus,Y), not(brother(Y,johan_friso)) }

Reasoning state after step (d):
I-rule i3:
u: unchanged
g: { married(claus,Z), relation(claus,Z), mother(Z,Y), father(claus,Y),
not(brother(Y,johan_friso)) }

Reasoning state after step (e):
I-rule i2(b):
u: unchanged
g: { relation(claus,beatrix), mother(beatrix,Y), father(claus,Y),
not(brother(Y,johan_friso)) }

Reasoning state at the end:
u: { father(claus,johan_friso), relation(claus,beatrix),
mother(beatrix,willem_alexander), mother(beatrix,johan_friso),
married(claus,beatrix) }
g: {}

When the whole of the reasoning chain would have been written down, one could have noticed that, due to the fact that only the backward chaining strategy was used, inference rule i1 was not used during the construction of the reasoning chain.

6 Application of the Formal Model

The formal model of a knowledge base system, as was developed in section 4, has been used to evaluate two expert system shells and an implementation of Prolog. This evaluation had two objectives:

1. Verify that all aspects of the model were correct and that all phenomena which occurred in the evaluation could be described with the elements of the model.
2. Investigate how the sets G, RS, and E and the inference procedure $IP^*(G,RS,E)$ were implemented and which possibilities the tool gave to formulate U and R.

The tools under investigation were KES-II (in the sequel called Tool A), Acquaint (Tool B) and Turbo-Prolog. KES-II is a frame-based shell written in C. Acquaint also is a frame-based shell, written in Lisp. All three tools need at least an IBM-XT or compatible. For this evaluation a simple case was developed. Eight queries were offered to the system and the answers of the queries were analyzed. In this experiment we did not use application specific meta knowledge. The results were as follows.

Component: U and R

Tool A: Definition of U(S), R in different sections. It is not possible to define constraints.

Tool B: It is not possible to define constraints.

Prolog: The DB state u is contained in the unit clauses. Obligation to group facts and rules with the same head. Due to the RS-component facts must be written before the rules.

Component: G

Tool A: Selection according to the textual order.

Tool B: Selection according to the textual order in the head of a chosen rule.

Prolog: Selection according to the textual order.

Component: RS

Tool A: Selection according to the textual order in the rule base. Since all rules are evaluated (even if the answer is already known) the order is not important.

Tool B: Every rule is given a weight, which depends on three factors. The rule with the highest weight is chosen first.

Prolog: Selection according to the textual order in the rule base.

Component: E

Tool A: Value "unknown" when no rule is applicable. When an expression from a hypothesis is not contained in any rule, the value is asked.

Tool B: Value "unknown" when no rule is applicable or when an expression from a hypothesis is not contained in any rule.

Prolog: For negative hypotheses the Negation as Failure rule is used. The reasoning process is interrupted when no answer is found within a limit.

Procedure $IP^*(G,RS,E)$

Tool A: Exclusively backward chaining.

Tool B: Alternating use of backward chaining and forward chaining. Goal selection takes place after the rule selection.

Prolog: Backward chaining, depth-first with backtracking.

From this experiment we also concluded that neither Tool A nor Tool B had a complete separation between the rule-base and the set of inference rules. Aspects from the meta knowledge could be programmed together with the rules. Properly using such a system therefore requires a very good insight in the elements of G, RS, E and $IP^*(G,RS,E)$, something that can not be expected from an end user.

For this experiment the model proved to be guiding. It was possible to explain all observations in terms of our model. Therefore, we have confidence that the model is sound and has a great value in evaluating and developing tools and languages of knowledge based character.

7 Conclusions

The aim of this paper was to find a formal answer to the questions what a KBS is intended to compute, and which components are required to compute it. In Section 3 we formally defined knowledge bases, their information content and set the purpose of a KBS as facilitating the computation of the information content of a knowledge base.

In Section 4 we defined the notion of a reasoning chain and determined the necessary components of a procedure that produces a reasoning chain to every query given to the knowledge base. These components were: goal selecting functions, rule selection functions and external rules. A KBS then was identified as a system where all these components are present and integrated in an inference procedure. The framework we obtained gave a good understanding of a KBS. Its components and functions could be clearly distinguished. Using this framework as a reference model for testing software tools, we have observed that it provided a very good insight in the tested systems. We believe that this feature can very well support design and evaluation of knowledge base systems.

References

[BRO89] de Brock, E.O., *Foundations of semantic databases*, Academic Service, Schoonhoven, 1989 (in Dutch).

[FLV88] Florescu, A., van Liempd, E.P.M. and Velthuijsen, H., *A distributed implementation of a network configuration problem*, Memorandum 1844 DNL/88, PTT–Research Neher Laboratorium, Leidschendam, 1988.

[GMN84] Gallaire, H., Minker, J. and Nicholas, H., Logic and Databases: A Deductive Approach, *Computing Surveys*, Vol. 16, nr. 2, 1984, pp. 153–185.

[LLO87] Lloyd, J.W., *Foundations of Logic Programming*, Second Edition, Springer Verlag, Berlin, 1987.

[MAR88] Mars, N., Research at good quality: Knowledge technology in development, *Informatie*, 30, nr. 2, 1988, pp. 84–90, (in Dutch).

[MIN88] Minker, J., Ed., *Deductive Databases and Logic Programming*, Morgan Kaufmann, 1988.

[SCH91] Schuwer, R. and Pels, H.J., *A unified view on data- and knowledge bases*, To appear, 1991.

In this series appeared:

89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a parallel program for Dirichlet convolution.
89/8	E.H.L.Aarts A.E.Eiben K.M. van Hee	A general theory of genetic algorithms.
89/9	K.M. van Hee P.M.P. Rambags	Discrete event systems: Dynamic versus static topology.
89/10	S.Ramesh	A new efficient implementation of CSP with output guards.
89/11	S.Ramesh	Algebraic specification and implementation of infinite processes.
89/12	A.T.M.Aerts K.M. van Hee	A concise formal framework for data modeling.
89/13	A.T.M.Aerts K.M. van Hee M.W.H. Hesen	A program generator for simulated annealing problems.
89/14	H.C.Haesen	ELDA, data manipulatie taal.
89/15	J.S.C.P. van der Woude	Optimal segmentations.
89/16	A.T.M.Aerts K.M. van Hee	Towards a framework for comparing data models.
89/17	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra.

90/1	W.P.de Roever- H.Barringer- C.Courcoubetis-D.Gabbay R.Gerth-B.Jonsson-A.Pnueli M.Reed-J.Sifakis-J.Vytopil P.Wolper	Formal methods and tools for the development of distributed and real time systems, p. 17.
90/2	K.M. van Hee P.M.P. Rambags	Dynamic process creation in high-level Petri nets, pp. 19.
90/3	R. Gerth	Foundations of Compositional Program Refinement - safety properties - , p. 38.
90/4	A. Peeters	Decomposition of delay-insensitive circuits, p. 25.
90/5	J.A. Brzozowski J.C. Ebergen	On the delay-sensitivity of gate networks, p. 23.
90/6	A.J.J.M. Marcelis	Typed inference systems : a reference document, p. 17.
90/7	A.J.J.M. Marcelis	A logic for one-pass, one-attributed grammars, p. 14.
90/8	M.B. Josephs	Receptive Process Theory, p. 16.
90/9	A.T.M. Aerts P.M.E. De Bra K.M. van Hee	Combining the functional and the relational model, p. 15.
90/10	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).
90/11	P. America F.S. de Boer	A proof system for process creation, p. 84.
90/12	P.America F.S. de Boer	A proof theory for a sequential version of POOL, p. 110.
90/13	K.R. Apt F.S. de Boer E.R. Olderog	Proving termination of Parallel Programs, p. 7.
90/14	F.S. de Boer	A proof system for the language POOL, p. 70.
90/15	F.S. de Boer	Compositionality in the temporal logic of concurrent systems, p. 17.
90/16	F.S. de Boer C. Palamidessi	A fully abstract model for concurrent logic languages, p. p. 23.
90/17	F.S. de Boer C. Palamidessi	On the asynchronous nature of communication in logic languages: a fully abstract model based on sequences, p. 29.

90/18	J.Coenen E.v.d.Sluis E.v.d.Velden	Design and implementation aspects of remote procedure calls, p. 15.
90/19	M.M. de Brouwer P.A.C. Verkoulen	Two Case Studies in ExSpect, p. 24.
90/20	M.Rem	The Nature of Delay-Insensitive Computing, p.18.
90/21	K.M. van Hee P.A.C. Verkoulen	Data, Process and Behaviour Modelling in an integrated specification framework, p. 37.
91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.

91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.
91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. .
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.