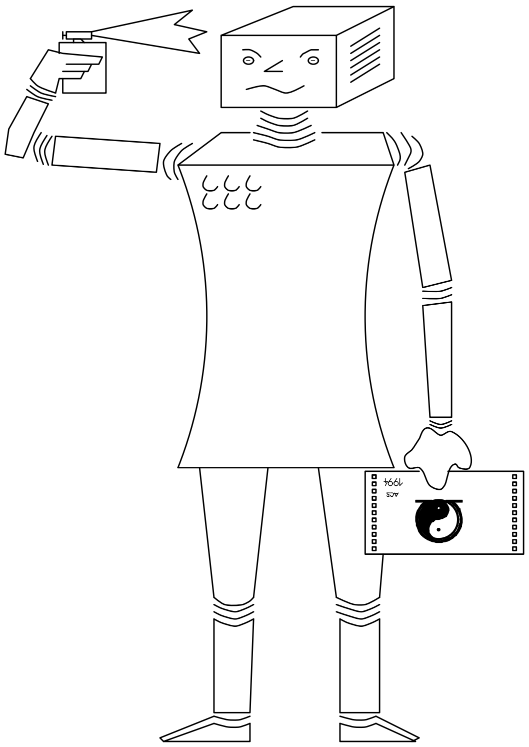

SEKI Report

Useful Properties of a Frame-Based Representation of Mathematical Knowledge

Manfred Kerber

SEKI Report SR-91-06

Useful Properties of a Frame-Based Representation of Mathematical Knowledge

Manfred Kerber
Fachbereich Informatik, Universität des Saarlandes
D-6600 Saarbrücken, Germany
kerber@cs.uni-sb.de

Abstract

To prove difficult theorems in a mathematical field requires substantial knowledge of that field. In this paper a frame-based knowledge representation formalism is presented, which supports a conceptual representation and to a large extent guarantees the consistency of the built-up knowledge bases. We define a semantics of the representation by giving a translation into the underlaying logic.

Keywords: frames, consistency, conservative extension, conceptual representation

Tota methodus consistit in ordine et dispositione eorum ad quae mentis acies est convertenda, ut aliquam veritatem inveniamus.
René Descartes, Regula V
REGULÆ AD DIRECTIONEM INGENII

This work was supported by the Deutsche Forschungsgemeinschaft, SFB 314 (D2,D3)

1 Introduction

In this paper we are going to describe how to represent mathematical "factual" knowledge for automated theorem proving. Guideline is knowledge like that of a mathematical dictionary. In particular we are here not interested in heuristic knowledge as described in [15, 16, 17, 7, 18]. The main means for our representation is higher-order logic. Indeed one may ask why such a powerful logic is not sufficient for the description of the factual knowledge of mathematics, because logic has been developed in the last hundred years for that purpose. The answer is that it is possible to find many extra-logical features in the presentation of mathematics, in mathematical text books for instance, and that these features are essential for mathematical activities like theorem proving.

What are the shortcomings of logic that make a mathematical knowledge representation necessary? One main point is, that the basic notion of logic is that of a formula, but that in mathematics different kinds of formulae are distinguished, namely *axioms*, *definitions*, and *theorems* and we will subdivide these kinds even more. Furthermore in mere logic a knowledge base consists of an unstructured set of formulae, whereas text books are well-structured and mathematicians spend a lot of time in the final presentation of the mathematical content. In addition there are constraints – which are not present in logic – in the procedure of stating theorems or defining concepts. The most important constraint for definitions is, that all concepts which are used in the definition – with the exception of the definiendum of course – must already be known. Analogously all concepts in theorems must be known. But even if all concepts are known, a definition has to fulfil further extra-logical requirements, for example, it is normally given in a form as abstract as possible.

Perhaps the main difference between logic and mathematics can be seen in the *conceptual* representation in mathematics. The standard schema of this procedure in mathematical text books is: "definition", "example", "theorem", "proof". When we look closer at this procedure, we see that the introduction of a concept is not terminated by giving a definition, but that examples, counter-examples, and lemmata about the introduced concept immediately belong to the concept. They do that in such an extent that it is possible to say: you have not understood the concept if you know only the definition, but you have not seen any examples and you do not know the simple properties of it.

Another great difference between logic and mathematics is that in mathematics it is always assumed – even if it cannot be proved – that a knowledge base is consistent, whereas by logic certain formulae are related, but it does not matter whether the preconditions are fulfilled or not. (This mathematical assumption is also the main reason for the completeness of the set-of-support strategy in resolution theorem proving.)

Which requirements should a knowledge representation formalism for mathematics satisfy? There are the following properties we would like to see:

– The knowledge base should be consistent and the representation formalism should support to keep it consistent.

– The representation formalism should reflect the different types of knowledge, that is, axioms, definitions, and theorems should be distinguished.

– The knowledge base should be redundancy free.

– It should not be possible to use unknown concepts.

– It should be possible that the knowledge can be represented in a conceptual, structured way.

– The representation formalism should be powerful enough in order to represent the knowledge easily.

– The formalism should have a clear semantics.

Now we discuss to what extent we can realize the requirements above.

The consistency of a knowledge base is of particular interest, because otherwise it is possible to derive anything of it. Unfortunately it cannot be shown in general because of Gödel's incompleteness result [5], when a representation language such as first-order or higher-order logic is used. But we can restrict the possibilities where inconsistencies may be imported: definitions and theorems should not lead to any inconsistencies, because definitions form conservative extensions and theorems are proved to be consequences. So only axioms can cause any trouble. Of course we cannot guarantee that the definitions of concepts are *correct*, that is, in accordance with the general use of them. We can define the concept "group" as something quite different from the general use, but because we cannot import contradictions by a definition, other parts of the knowledge base not using this concept cannot be concerned with such a non-standard definition. (The importance of this fact for automated theorem proving is already noted by Robert S. Boyer and J Strother Moore in [3, p.13].)

The distinction of the three different kinds of knowledge is very important for consistency: If we guarantee that definitions are really definitions and that theorems are proved, we have to be careful only with the axioms. Therefore we will use in the following three different basic knowledge units, one for axioms, one for definitions, and one for theorems.

The redundancy freeness can partially be guaranteed by preventing that concepts are defined twice, but to some extent it will be left to the user of a system.

If we have a knowledge base Δ and we want to add a new knowledge unit ϑ, a check of the signatures of the knowledge base and the newly introduced unit can guarantee that all concepts used in ϑ, with the exception of a newly defined one, are already in Δ.

When introducing concepts we do not want to spread the knowledge about these concepts all over the whole knowledge base. Hence a concept should not just consist of its actual logical definition, but simple consequences, examples, or alternative definitions should immediately be associated with this concept. Therefore we introduce in the following a formalism to represent mathematical concepts, such that the knowledge associated with an object is representationally attached to that object.

Now we present a frame-based representation of mathematical knowledge, introducing it by examples as we go along. Then we give a formal definition, discuss the properties of such knowledge bases and finish by some considerations on the advantages and disadvantages of the chosen formalism.

2 The Representation Language

In this section we introduce our representational formalism. We use the *frame* representation of Marvin Minsky [13]. The original idea is to represent knowledge in an object-oriented way and to simulate thereby the knowledge organisation as it is presumably organized in the mind (of a mathematician). The frames should structure the knowledge and contain in particular the information how to use this knowledge. Although introduced in opposition to the logicistic wing of the AI community, a frame can (more or less) be viewed as a certain way of arranging predicate logical facts [8], [14, chap.7]. Since our facts are predicate logical expressions, in our case this corresponds to meta-logical facts. A frame consists of a *name*, *slots*, and *fillers*. Slots correspond to certain meta-predicates and the fillers are arguments of these predicates. The syntactical surface is that of a box, as in figure 1.

We will use as the heart of the representation sc Alonzo Church's theory of types [4] with the following three extensions. For further details see [10]. We shortly introduce the extensions now.

Optional Parameters

The first extension is that of optional parameters. In order to define a group we need indeed only two parameters: a set and a binary operation on that set. If these two parameters are given, it is well defined whether or not the pair $\langle G, + \rangle$ is a group, that is, that $group(G, +)$ holds, provided we have the corresponding definitions. We do not need to know the name of the neutral element or the name of the inverse function. But these names may become important later on. In order to express that $\langle G, + \rangle$ is a group with neutral element 0, we want to use the same predicate symbol *group* and say $group(G, +, 0)$. Optional parameters are allowed only in predicate constants. We allow them by a declaration of the arity of a predicate symbol $l \leq \texttt{arity}(P) \leq m$ (with l, m explicit natural numbers). For instance:
$2 \leq \texttt{arity}(group) \leq 4$ or $2 \leq \texttt{arity}(ex\_left\_neutral\_element) \leq 3$.

For the definition of a predicate constant P with arity $l \leq \texttt{arity}(P) \leq m$ every formula φ containing $P(t_1, \ldots, t_k)$ with $l \le k \le m$ is an abbreviation for φ with $P(t_1, \ldots, t_k)$ replaced by $(\exists z_{k+1}, \ldots, z_m \;\; P(t_1, \ldots, t_k, z_{k+1}, \ldots, z_m))$.

Variable Sorts

For the representation of certain concepts it is useful to have variable sorts. For instance if we want to define the concept group, we want to say "$\forall G : (\iota \to o) \;\; \forall + : (G \times G \to G) \;\; \mathit{group}(G, +) \iff \ldots$", that is, we want to use the predicate variable G as a sort symbol. We allow such an expression as abbreviation for the corresponding relativization, that is,
$\forall G : (\iota \to o) \;\; \forall + : (G \times G \to G) \;\; \mathit{group}(G, +) \iff \ldots$ is the abbreviation for $\forall G : (\iota \to o) \;\; \forall + : (\iota \times \iota \to \iota) \;\; \mathit{function}(+, G \times G \to G) \Longrightarrow (\mathit{group}(G, +) \iff \ldots)$. We write $\forall x : C \;\; P(x)$ as abbreviation for $\forall x \;\; C(x) \Longrightarrow P(x)$. In general we write unary predicates in a sort like manner, but we always use them only as abbreviation for the corresponding relativization. By *function* we mean the following predicate:

$$\mathit{function}(f, X_1 \times \cdots \times X_m \to Y) \iff$$
$$(\forall x_1 \;\; X_1(x_1) \Longrightarrow (\cdots \forall x_m \;\; X_m(x_m) \Longrightarrow Y(f(x_1, \ldots, x_m)) \cdots)).$$

Constructors

A common way of introducing concepts is via an inductive definition. Higher-order logic is rich enough to cover this situation, but because of the particular importance of this, we will provide special facilities for defining concepts by mathematical induction. The standard example is that of the PEANO axioms for axiomatizing the natural numbers. Instead of giving these axioms we write shortly the data structure $\mathbb{N}$ (compare e.g. [9]):

data structure	$\mathbb{N} : (\iota \to o)$
constructors	base$(0 : \mathbb{N})$
	step$(s : (\mathbb{N} \to \mathbb{N}))$

In general we can introduce a data structure by:

- $P : (\kappa \to o)$ with $\kappa \neq o$
- $(c_j : P)$ for $1 \le j \le n$, the c_j are called constructor constants.
- $(f_i : (P \times \kappa^i_1 \times \cdots \times \kappa^i_{k_i} \to P))$ for $1 \le i \le m$, the f_i are called constructor functions.

This data structure has the following meaning:

- There is a constant $P : (\kappa \to o)$.
- For all constructor constants c_j, there are constants of sort P and for all constructor functions f_i, there are constants of sort $(P \times \kappa^i_1 \times \cdots \times \kappa^i_{k_i} \to P))$.
- No constructor constant is in the range of any constructor function.
- All constructor functions are injective.

– The ranges of the constructor functions are disjoint.

– The induction principle holds.

Examples

Now we begin the proper part of the description of the representation formalism. We introduce it with the help of examples and begin with the definition of the concept "associative". This definition could be given in an extended sorted higher-order logic as:

$$\forall C\!:\!(\iota \rightarrow o)\ \forall f\!:\!(C \times C \rightarrow C)\ \ associative(C,f) \iff$$
$$\forall x,y,z\!:\!C\ \ f(f(x,y),z) \equiv f(x,f(y,z)).$$

and this is now represented in a frame in figure 1 below. Every frame belongs to one of the three kinds: definition, axiom, or theorem. The kind is here indicated by the keyword "**Definition:**" The name of the introduced concept follows after a colon. In this case it is "**associative**". In the upper right corner we give the type of the definition. The entry "property" means that the whole concept models a property; a standard translation of a "property definition" into predicate logic can be done by mapping it into a predicate symbol. A "property definition" represents the relationship between its parameters. (The other type of a concept definition is that of a "mapping concept". In this case a new object is created, of which an example is given in figure 4 below.)

Now the slots and the slot-fillers of the frames are introduced:

Definition:	**associative**		property
parameters:	C	$:(\iota \rightarrow o)$	
	f	$:(C \times C \rightarrow C)$	
definition:	$\forall x,y,z\!:\!C\ \ f(f(x,y),z) \equiv f(x,f(y,z))$		
context:	basic algebra		

Figure 1

The argument of the binary property **associative** is given in the slot "parameters". The number of parameters corresponds to the arity of the predicate symbol defined in the frame. In the slot "parameters" the *formal* parameters and their sorts are written and when using the defined object elsewhere they are then bound to the *actual* parameters.

In the slot "definition" we find a (higher-order) logical definition of the concept. In the case of a property definition, the entry consists of the part of the definition that follows the equivalence sign "$\iff$".

The slot "context" is provided for structuring the whole knowledge base into different modules. If we introduce a partial order among these contexts, we have the usual taxonomical hierarchy. (Here more details may become necessary, for example, which information may be used by other modules.) For instance we might structure a large knowledge base as in figure 2. This example is taken from the algebra text-book of

Bartel L. van der Waerden [19, p.xi]. Such a structure is standard in all fields of mathematics, therefore we have to provide a possibility for representing it. How this information can be used is discussed below.

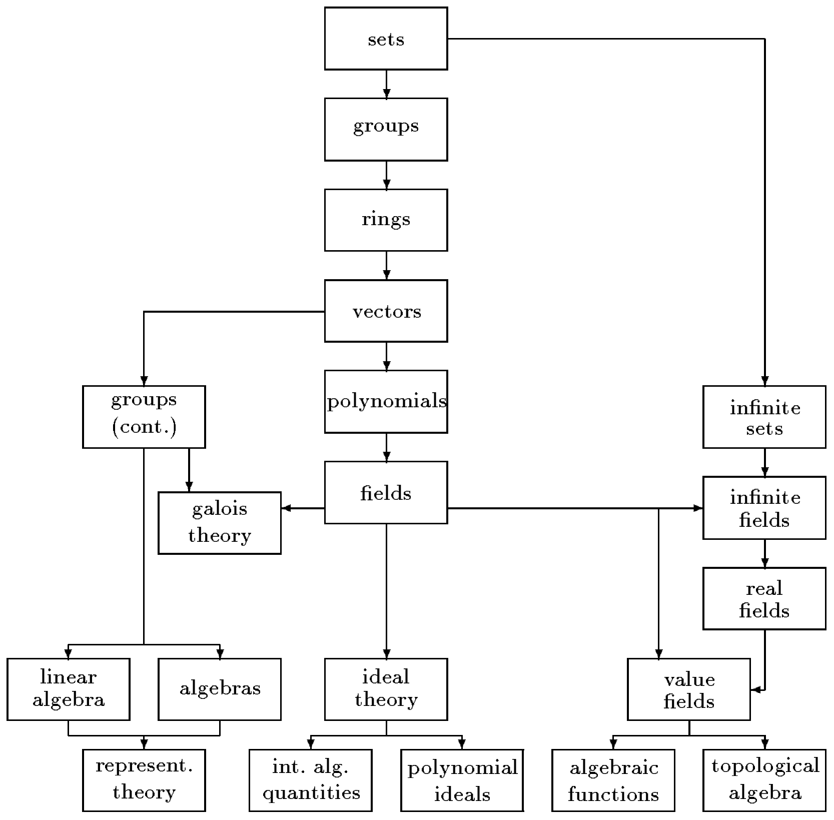

Figure 2

Now we shall introduce the second kind of frame, namely that of an axiom frame. In order to do so we take the first four axioms of Gödel's set axioms [6].

Axiom: set

axioms:		$\forall x{:}Set\ \ Class(x)$	
		$\forall X,Y{:}Class\ \ X \in Y \Longrightarrow Set(X)$	
		$\forall X,Y{:}Class\ \ (\forall u{:}Set\ \ u \in X \iff u \in Y) \Longrightarrow X \equiv Y$	
		$\forall x,y{:}Set\ \ \exists z{:}Set\ \ (\forall u{:}Set\ \ u \in z \iff (u \equiv x \vee u \equiv y))^*$	
consequences:	1)	$\forall x,y{:}Set\ \ y \in \{x\} \iff x \equiv y$	<proof-set-cons-1>
signature_ext:		$Class \sqsubseteq \iota$	
		$Set \sqsubseteq Class$	
		$\in: (Set \times Class \to o)$	
		$\{.,.\} : (Set \times Set \to Set)$	
context:		sets	

Figure 3

*The existence of the variable z of sort Set can be written by the Skolem function $\{x,y\}$. As usual $\{x,x\}$ is abbreviated to $\{x\}$.

While definitions are always of the form "$\forall$*parameters definiendum*(*parameters*) $\iff$ *formula*", the slot "axioms" in contrast can contain arbitrary formulae, stating the properties of one or more function and/or predicate constants. If these constants are newly introduced by the frame, they have to be explicitly summarized in the slot "signature_ext". The slot "consequences" contains lemmata about the concepts introduced by the axioms. Such consequences must be proved; in <proof-set-cons-1> a pointer to such a proof is stored.

The logics $\mathcal{L}$ are expressible enough to give any of the standard axiomatizations of set theory such as ZFC, but for most cases it is sufficient to use sets of a certain sort κ as an abbreviation for a predicate of the sort $(\kappa \to o)$. If one follows this simple notion of a set, it is not possible to have elements of different types in one single set.

Now we give an example for a "mapping concept" and define a "pair".

Definition:	**pair**		mapping
parameters:	x	:Set	
	y	:Set	
definition:	$\{x, \{x, y\}\}$		
sort:	Set		<proof-pair-sort>
main_property:	$\forall x, y, u, v{:}Set\ \ \langle x, y\rangle \equiv \langle u, v\rangle \iff x \equiv u \wedge y \equiv v$		<proof-pair-main_prop>
context:	sets		

Figure 4

We write $\langle x, y\rangle$ instead of $pair(x, y)$. The concept "pair" is an example where the entry in the definition slot itself is less important and indeed it is never used again. The definition is only given in order to have a set theoretical foundation of the concept as $\langle a, b\rangle :\equiv \{a, \{a, b\}\}$. The concept is closely related to a "main_property"; almost the only thing one has to know about the concept "pair" is: they are equal if and only if they agree on all arguments. In order to model main properties we introduce a corresponding new slot with this statement as filler. Another view of a main property is that this is the intrinsic meaning of the concept and that the definition is only an implementation of the concept in logic.

In the next concept we use an "optional" parameter, which corresponds to the optional parameters of predicates as introduced above. The name "*neutral_element*" is a selector of the tuple $(C, f, 0)$.

Definition:	**ex_left_neutral_element**		property
parameters:	C	:$(\iota \to o)$	
	f	:$(C \times C \to C)$	
(optional)	0	:C	(called *neutral_element*)
definition:	$\forall x{:}C\ \ f(0, x) \equiv x$		
context:	basic algebra		

Figure 5

Together with the similarly definable concept "ex_neutral_element" and "ex_inverse" it is now possible to introduce the concept "group".

Definition:		**group**		property
parameters:		G	$:(\iota \rightarrow o)$	(called *carrier*)
		$+$	$:(G \times G \rightarrow G)$	(called *operation*)
(optional)		0	$:G$	(called *neutral_element*)
(optional)		$-$	$:(G \rightarrow G)$	(called *inverse*)
definition:		*TRUE*		
superconcepts:	1)	$associative(G,+)$		
	2)	$ex\_neutral\_element(G,+,0)$		
	3)	$ex\_inverse(G,+,0,-)$		
equivalences:	1)	$associative(G,+) \wedge$ $ex\_left\_neutral\_element(G,+,0) \wedge$ $ex\_left\_inverse(G,+,0,-)$		<proof-group-equ-1>
	2)	$associative(G,+) \wedge$ $ex\_right\_neutral\_element(G,+,0) \wedge$ $ex\_right\_inverse(G,+,0,-)$		<proof-group-equ-2>
examples:	1)	$(Int,+,0,-)$	model Integers	
	2)	$(Rat,+,0,-)$	model Rationals	
	3)	$(Rat\backslash\{0\},\cdot,1,^{-1})$	model Rationals	
context:		basic algebra		

Figure 6

The slot "superconcepts" expects slot-fillers that are generalizations of the actual concept. For instance every group is especially an associative structure. *TRUE* in the definition slot means that the concept is fully defined by the conjunction of its superconcepts.

The slot "equivalences" contains logical expressions that are necessary and sufficient to define the concept, that is, they are logically equivalent to the conjunction of the "definition" slot-filler and the "superconcepts" slot-fillers and form an alternative definition of the concept. In order to guarantee that the filler is really equivalent to the definition of the concept there must be a proof, which can be found in <proof-group-equ-1>. Although the different definitions of a concept are equivalent, they can be of different use in different context. For example, if one wants to prove that a certain object is a group, it is easier to use one of the two equivalent formulations, because then one has less to prove (e.g. $ex\_left\_neutral\_element(G,+,0)$ instead of $ex\_neutral\_element(G,+,0)$). The definition itself may be preferable if it is used as a premis, because then one can immediately use the property $ex\_right\_neutral\_element(G,+,0)$ without proving it first. In principle the frame approach is flexible enough to store such meta knowledge. But that is not integrated, in particular it is necessary to have a formal language to express meta knowledge. We can see an important difference between our epistemological term "property" and the logical term "predicate": The conceptual representation allows to make some assertions *about* the concepts (as for instance to use a certain variant of the definition in some situation). This would be difficult if we had mere predicates and distributed the knowledge in a whole set of formulae. Whether and how this can be used for actually guiding a theorem prover is a difficult question and not yet investigated deeply.

In the slot "examples" we can find a reference to a model of the corresponding concept. How examples can be represented and how it can be proved that an example is really an example is not investigated in this paper.

We could have also given another definition for the concept "group" by writing the following formulae into the "definition" slot:

$$\forall x, y, z{:}G \;\; (x+y)+z \equiv x+(y+z) \;\wedge$$
$$\exists x{:}G \;\; \forall y{:}G \;\; x+y \equiv y \equiv y+x \;\; (\wedge\; x \equiv 0) \;\wedge$$
$$\forall x{:}G \;\; \exists y{:}G \;\; x+y \equiv 0 \equiv y+x \;\; (\wedge\; y \equiv -x)$$

But this is not as appropriate as the above formulation, because it is less structured. We want to formulate the concepts as abstract as possible for several reasons. If we formulate abstractly, we have the chance to find proofs also at a more high, abstract level. There is also the chance to find analogies or to use some special purpose algorithms that are defined for a special concept. For instance when the concept "group" is given, one might want to have a special treatment for the "+" as an associative operation. That could be done by not expanding the definition of associativity but by using a special equality reasoning procedure for associative function symbols, for instance, an A-unification algorithm.

So far we have given only examples for *simply* defined or axiomatized concepts. The next example is about *inductively* defined concepts. To this end we use the constructors introduced above.

We show how one can axiomatize the natural numbers with the constructors 0 and s.

Axiom: Nat

axiom:	base	$(0 : \mathbb{N})$
	step	$(s : (\mathbb{N} \to \mathbb{N}))$
signature_ext:		$\mathbb{N} : (\iota \to o)$
		$0 : \mathbb{N}$
		$s : (\mathbb{N} \to \mathbb{N})$
context:		numbers

Figure 7

The key words "base" and "step" correspond to the induction base and the induction step, respectively. In this example the induction base says that 0 is a natural number. The step case means that all natural numbers are constructed from 0 by the constructor function s.

Of course we could give second-order formulae in order to introduce the natural numbers, but because of its particular importance, we provide special facilities for inductively defined or axiomatized concepts.

Now we can define the function + for natural numbers.

Definition:	$+_{\mathbb{N}}$			mapping
parameters:		n	$:\mathbb{N}$	
		m	$:\mathbb{N}$	
sort:		$\mathbb{N}$		<proof-$+_{\mathbb{N}}$-sort>
definition:	base	$\forall n{:}\mathbb{N}\ \ n +_{\mathbb{N}} 0 \equiv n$		
	step	$\forall n, m{:}\mathbb{N}\ \ n +_{\mathbb{N}} s(m) \equiv s(n +_{\mathbb{N}} m)$		<proof-$+_{\mathbb{N}}$-def>
context:		numbers		

Figure 8

In this definition frame "base" and "step" correspond as in the axiom frame to the induction base and the induction step. For an inductive definition it must be shown that it is a definition indeed, that is, that all cases are covered and that the step case is well-founded (see e.g. [9]).

The next example shows an *implicit* definition. It is the general case of a definition and subsumes all previous cases as special cases. An implicit definition consists of an arbitrary formula set that uniquely characterizes the concept.

Definition:	**exp**		mapping
parameters:	x	$:\mathbb{R}$	
sort:	$\mathbb{R}$		<proof-exp-sort>
definition:	$\forall x, y{:}\mathbb{R}\ \ \exp(x+y) \equiv \exp(x)\cdot\exp(y)$		
	$\exp(1) \equiv e$		
	$continous(\exp, \mathbb{R})$		<proof-exp-def>
context:	real functions		

Figure 9

In order to make sure that it is a valid definition, the existence and uniqueness of the concept must be proved in <proof-exp-def>.

Partial functions can be defined in the following form:

Definition:	**reciprocal**		mapping
parameters:	x	$:\mathbb{R}$	
preconditions:	$x \not\equiv 0$		
sort:	$\mathbb{R}$		<proof-reciprocal-sort>
definition:	$reciprocal(x)\cdot x \equiv 1$		<proof-reciprocal-def>
context:	real functions		

Figure 10

The slot "preconditions" contains formulae, which must be satisfied, for the definition to make sense. A correct treatment of the preconditions requires *partial* functions. However we do not consider them in the following, because they require an essential extension of the logics $\mathcal{L}^n$, as for instance a transition from two-valued to three-valued logics. In particular we cannot translate them adequately into first-order (two-valued) logic. For an overview of partial logics see [2, 11].

The concepts introduced so far are ordered by the fillers of the slot "superconcepts", which induce a transitive network of concepts with inheritance. That is, every concept inherits all definitions, consequences, equivalences, and counter-examples of its superconcepts. A superconcept itself inherits all examples of its subconcepts. A small section of this net can be seen in the following figure:

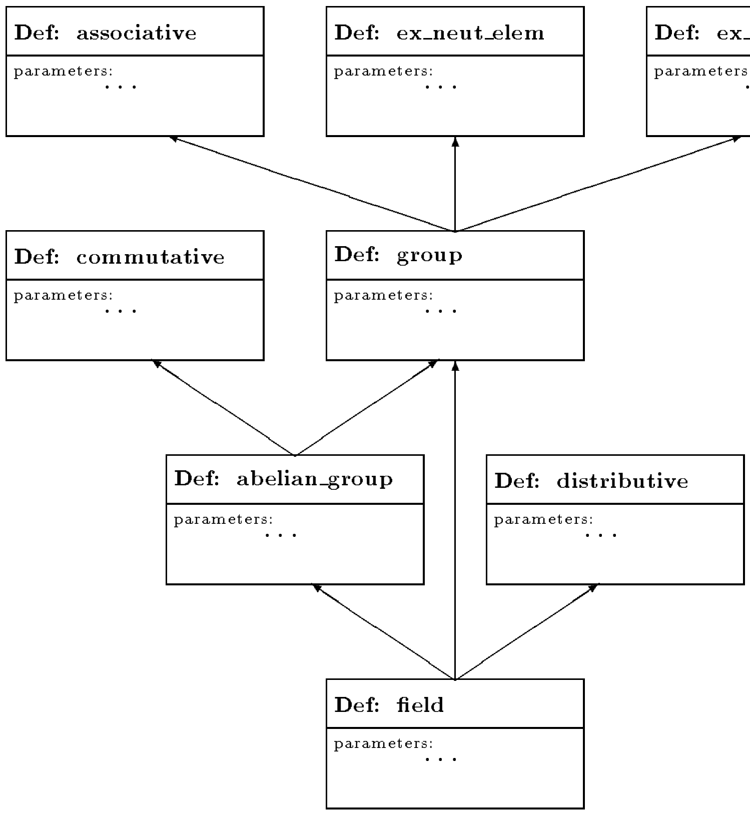

Figure 11

The edges in the network are labeled with the corresponding parameters in the "superconcepts" slot. In figure 11 we have omitted the labels for simplicity reasons. For instance, the edge from "field" to "abelian_group" expresses that a field $(F, +, 0, -, \cdot, 1, ^{-1})$ is an abelian group with respect to the *addition*, that is, $(F, +, 0, -)$ is an abelian group; whereas it is only a group with respect to the *multiplication*, that is, $(F\backslash\{0\}, \cdot, 1, ^{-1})$ is a group.

Finally we give an example for a theorem frame. We use CANTOR'S theorem that the power-set of a set has greater cardinality than the set itself in the formulation of ANDREWS [1, p.184].

Theorem: Cantor	theorem
theorem:	$\forall s:(\iota \to o)\ \neg\exists g:(\iota \to (\iota \to o))\ \forall f:(\iota \to o)\ f \subseteq s \Longrightarrow (\exists j:\iota\ \ s(j) \wedge g(j) = f)$
status:	"proved"
proof:	<proof-Cantor>
context:	sets

Figure 12

The filler of the slot "theorem" is an arbitrary closed formula of $\mathcal{L}^n$.

The "status"-slot records whether the theorem is "proved", "conjectured", or "rejected".

In "proof" a pointer to a proof is given, if the status of the theorem is "proved" or "rejected".

After this informal introduction to our frame-based representation language, we shall now give a formal definition in the next section.

3 Formal Treatment

Now we define a general syntax of the three different kinds of frames. Their semantics is then given via translations into the underlying higher-order logic. The following frame shows all the different slots and possible fillers in a definition frame.

Definition:		**<Name>**		property | mapping
[parameters:		{ <variable_symbol>	:<sort_symbol>	[(called <name>)] }
[{ (optional)		<variable_symbol>	:<sort_symbol>	[(called <name>)] }]]
[preconditions:		{ <formula> }]		
definition:		{ <formula> } |		
		{ <term> } |		
	base	{ <formula> }		
	step	{ <formula> }	<proof-<Name>-def> |	
		{ <formula> }	<proof-<Name>-def>	
[defined_symbols:		{ <Name> }]		
[superconcepts:	{ #)	<concept>({ <variable_symbol> }) }]		
[sort:		<sort_symbol>	<proof-<Name>-sort>]	
[main_property:		<formula>	<proof-<Name>-main_prop>]	
[consequences:	{ #)	<formula>	<proof-<Name>-cons-#> }]	
[equivalences:	{ #)	<formula>	<proof-<Name>-equ-#> }]	
[examples:	{ #)	<pointer_to_model>	<proof-<Name>-ex-#> }]	
[counter_ex:	{ #)	<pointer_to_model>	<proof-<Name>-counter_ex-#> }]	
[used_in:		{ <Name> }]		
[subconcepts:		{ <concept> }]		
context:		<ContextName>		

Figure 13

We use the extended Backus-Naur form (EBNF) in the following way:
[.] stands for no or one occurrence, { . } for one or more repetitions, <.> for non-terminal symbols, and | for "or".

1 Definition (Definition Frame): A *definition frame* (written ϑ) is a list of the following elements:

1. "parameters" is of a list of variable_symbols with sorts and optionally selector names.

2. "preconditions" is a list of $\mathcal{L}$-formulae.

3. "definition" is either a simple definition, an inductive definition, or an implicit definition. If it is a simple definition it consists of an $\mathcal{L}$-formula or an $\mathcal{L}$-term corresponding to the type of the frame (property or mapping). The defined concept must not occur in that formula or term.

   In case of an inductive definition two slots must be filled, one for the base case and another for the step case, both with $\mathcal{L}$-formulae. For every constructor constant c_j the base case contains a formula of the form:
   $\forall x_1, \ldots, x_m$ **<Name>**$(c_j, x_1, \ldots, x_m) \iff \psi_j$, where the defined concept must not occur in ψ (see definition of axiom frame, definition 4).
   For every constructor function f_i the step case contains a formula of the form
   $\forall x_1, \ldots, x_m \forall y, y_1, \ldots, y_{k_i}$ **<Name>**$(f_i(y, y_1, \ldots, y_{k_i}), x_1, \ldots, x_m) \iff \psi_i$ where in ψ_i no constructor function occurs. (In the case of a mapping concept "$\iff \psi$" is replaced by "$\equiv t$".)

   Implicit definitions consist of sets of formulae and a proof that the defined object exists and is unique.

4. "defined_symbols" is the list of symbols that are defined in an implicit definition, if more than one symbol is defined. In all other cases just the symbol **<Name>** is defined.

5. "superconcepts" is a list of atomic $\mathcal{L}$-formulae.

6. "sort" is a sort symbol that shows the sort of a concept of type "mapping". This slot must be filled when a definition of type "mapping" is given.

7. "main_property" is an $\mathcal{L}$-formula. The proof must show that the formula in this slot follows from the definition.

8. "consequences" and "equivalences" are lists of $\mathcal{L}$-formulae. The proofs show that the formulae are consequences or equivalences of the definition, respectively.

9. "examples" and "counter_ex" are lists of $\mathcal{L}$-structures, which are models or no models of the concept, respectively.

10. "used_in" is a list of axioms, definitions, and theorems. "subconcepts" is a list of axioms and definitions. These slots can be filled automatically, since the "used_in"

slot is only a book-keeping slot and "subconcepts" is the inverse slot to the "superconcept" slot.

11. "context" is a name for a theory.

The frame must not contain any free variable except the variables of the slot "parameters". **<Name>** is a constant symbol of sort $(\kappa_1 \times \cdots \times \kappa_m \to \kappa)$, where κ_i is the sort of the i-th parameter and κ is the entry of the "sort" slot for a mapping frame and equal to o for a property frame.

2 Definition (Semantics of a Definition Frame): We fix the *semantics of a definition frame* by giving a logical translation. We begin with the translation of the parts 1 through 5 of definition 1. Let $x_1, \ldots, x_m$ be the variables of the parameter slot, $p_1, \ldots, p_l$ be the list of preconditions, φ be the entry of the definition slot and $s_1, \ldots, s_k$ be the entries of the superconcepts slot. The logical translation of a simple definition of type property is then:
$\forall x_1, \ldots, x_m \;\; p_1 \wedge \ldots \wedge p_l \Longrightarrow (\textbf{<Name>}(x_1, \ldots, x_m) \iff (s_1 \wedge \ldots \wedge s_m \wedge \varphi))$.
The $x_1, \ldots, x_m$ are the only free variable symbols in p_i, s_i, and φ. Optional parameters are treated in the same manner as above. If the slot "preconditions" or "superconcepts" is unfilled it is translated by the same formula. Recall that an empty conjunction evaluates to *TRUE*.

In the case of a mapping frame the translation is:
$\forall x_1, \ldots, x_m \;\; p_1 \wedge \ldots \wedge p_l \Longrightarrow (\textbf{<Name>}(x_1, \ldots, x_m) \equiv s)$,
where s is an $\mathcal{L}$-term. The "sort" slot entry κ is translated as $(s : \kappa)$ or $\kappa(s)$.

In the case of an inductive or an implicit definition the translation is equivalent to:
$\forall x_1, \ldots, x_m \;\; p_1 \wedge \ldots \wedge p_l \Longrightarrow \varphi_1 \wedge \ldots \wedge \varphi_m$,
where the φ_i are the entries of the definition slot.

The other slots contain theorems, models, or structuring information.

The slot "consequences" contains theorems c_i that belong to the concept. The proved theorems are translated by the formulae itself, that is, by c_i for all i.[*]

The semantic status of an entry of the slot "main_property" is the same as that of an entry of the "consequences" slot.

Entries ψ of the slot "equivalences" are translated to:
$\forall x_1, \ldots, x_m \;\; p_1 \wedge \ldots \wedge p_l \Longrightarrow (\textbf{<Name>}(x_1, \ldots, x_m) \iff \psi)$.

"subconcepts" is the inverse slot of "superconcepts". That means, whenever we make an entry in the "superconcepts" slot of concept A, that B is a superconcept of A, the slot "subconcepts" of B is automatically filled by the filler A. (Parameters are neglected in the "subconcepts" slot.) This slot – as well as the "used_in" and "context" slot – is not translated into logic, that is, it has no semantics and is only for pragmatic use.

[*]For instance if the union of sets is defined, we should write into this slot, that the union is associative, commutative, and idempotent.

The semantics of the slots "examples" and "counter_ex" is defined by the logical model relation, that is, $\mathcal{M}$ is an example iff $\mathcal{M} \models$<**Name**>$(x_1, \ldots, x_m)$, and $\mathcal{M}$ is a counter-example iff $\mathcal{M} \not\models$<**Name**>$(x_1, \ldots, x_m)$.

3 Remark: Superconcepts provide an inheritance relation, for instance "group" is a superconcept for "associative". So every example for a group is in particular an example for an associative structure. On the other hand "associative" is a subconcept of "group". So every consequence in the "associative"-frame is also a consequence for the "group"-frame.

Analogously one can define axiom frames and theorem frames. All frame types have a "consequence" and a "context" slot.

In the next figure all different slots of an axiom frame are shown:

Axiom: <**Name**>

axioms:		{ <formula> } |	
	base	{ <term_declaration> }	
	step	{ <term_declaration> }	
[signature_ext:		[{ <constant_symbol>	:<sort_symbol> }]]
[consequences:	{ #)	<formula>	<proof-<Name>-cons-#> }]
[examples:	{ #)	<pointer_to_model>	<proof-<Name>-ex-#> }]
[counter_ex:	{ #)	<pointer_to_model>	<proof-<Name>-counter_ex-#> }]
[used_in:		{ <name> }]	
context:		<ContextName>	

Figure 14

4 Definition (Axiom frame): An *axiom frame* (also written as ϑ) is a list of the following elements: "axioms", "signature_ext", "consequences", "examples", "counter_-ex", "used_in", and "context" with:

1. The slot "axioms" contains either a set of formulae as simple axioms or new constructor symbols (compare page 5) are introduced. In the latter case the name of the axiom is a new predicate symbol. In the "base"-subslot the constructor constants of this new sort are declared in the form of term declarations. In the "step"-subslot the constructor functions are declared in the same manner. These symbols have to occur also in the slot "signature_ext".

2. "signature_ext" contains the axiomatized constants, if they are new.

3. All other slots are similar to the corresponding slots of a definition frame.

5 Definition (Semantics of an Axiom Frame): The *semantics of an axiom frame* is also given by the translation to $\mathcal{L}$. In the case of simple axioms it is translated into the conjunction of the formulae itself. In the case of inductive axioms they are translated into the corresponding data structure, which in turn can be translated as shown on page 5.

Finally we show what a theorem frame looks like in general:

Theorem:		**<Name>**		[<theoremtype>]
[assumptions:		<closed formula>]		
theorem:		<closed formula>		
status:		"proved" | "conjectured" | "rejected"		
[proof:		<proof>]		
[consequences:	{ #)	<formula>	<proof-<Name>-cons-#> }]	
context:		<ContextName>		

Figure 15

6 Definition (Theorem Frame): A *theorem frame* (also written as ϑ) is a list of: "assumptions", "theorem", "status", "proof", "consequences", and "context" with:

1. "assumptions" is a list of formulae, which are preconditions for the theorem.
2. "theorem" is a formula.
3. "status" is either "proved", "conjectured", or "rejected".
4. "proof" is a pointer to a proof in the case that the status of the frame is "proved". It is a pointer to a counterexample in the case that the status is "rejected".
5. the other slots are just as above.

The <theoremtype> is used for the classification of the theorem as "lemma", "theorem", "main theorem", or "corollary".

7 Definition (Semantics of a Theorem Frame): The *semantics of a theorem frame* with entries $\varphi_1, \ldots, \varphi_m$ as assumptions and ψ as theorem is again given by a translation. In the case of the status "proved" it is: $\varphi_1 \wedge \ldots \wedge \varphi_m \Longrightarrow \psi$, otherwise, it is: *TRUE*.

8 Definition (Signature of a Frame): The *signature of a frame* ϑ is the set of all constants in the terms and formulae in ϑ. (Recall that we have no free variables in the frames, except the variables introduced in the "parameters"-slot.)

9 Definition (Extension of a Frame): A *frame-extension* ϑ' of a frame ϑ is a frame with the same name and classification, where all slot-fillers but the slots mentioned below are identical. The fillers of the slots consequences, equivalences, examples, counter_ex, and used_in can differ in the form, that the fillers of ϑ are sublists of the corresponding fillers of ϑ'; the slot main_property can differ, if it is not filled in ϑ. The status of a theorem frame may be changed from "conjectured" to "proved" or "rejected". In both cases the proof slot must be filled.

So far we have considered single frames, in the next section we discuss when many such frames form a knowledge base.

4 Building up a Knowledge Base

In this section we define the notion “knowledge base” and consider the consistency of knowledge bases. In particular we discuss the conditions when a definition forms a conservative extension and therefore does not endanger the consistency of a knowledge base. A knowledge base is defined inductively: to the empty knowledge base we can add frames under certain conditions.

10 Definition (Knowledge Base): A *knowledge base* Δ over $\mathcal{L}$ is defined inductively:

- as the empty knowledge base $\Delta_{\emptyset} = \emptyset$, or
- an immediate extension of a knowledge base.

Δ is called an *immediate extension* of a knowledge base Δ' iff

- $\Delta = \Delta' \cup \{\vartheta\}$ with axiom frame, definition frame, or theorem frame ϑ relative to the knowledge base Δ', or
- it is equal to a knowledge base Δ' for all but one entry and this entry is a frame-extension of the other. Formally $\Delta \backslash \{\vartheta\} = \Delta' \backslash \{\vartheta'\}$ and ϑ is a frame-extension of ϑ'.

The transitive closure of this relation is called an *extension*.

The *signature of a knowledge base* Δ is the union of all signatures of the frames contained in Δ. The logic with this signature is denoted by $\mathcal{L}(\Delta)$.

Now we define *frame relative to* a knowledge base:

- A theorem frame ϑ is called a *theorem frame relative to a knowledge base* Δ if it is a theorem frame, its signature is a subset of the signature of Δ. Furthermore if the status is “proved” or “rejected”, the slot proof must contain a proof for “$\Delta \models$ (assumptions $\Longrightarrow$ theorem)”, respectively a counterexample for this relation. The entries in the “consequences” slot must be proved too.
- An axiom frame ϑ is called an *axiom frame relative to a knowledge base* Δ if it is an axiom frame and all occurring constants in the frame are either in the signature of Δ or in the slot “signature_ext” of ϑ. The signature of Δ must be disjoint from the elements in “signature_ext”. Lemmata in the frame must fulfill the condition for a theorem frame above.
- A frame ϑ is a *definition frame relative to a knowledge base* Δ if it is a definition frame, the defined concept name(s) is (are) not in the signature of Δ, but all other constant symbols are. Lemmata in the frame must fulfill the condition for a theorem frame above.

For the following we need the notion of semantic consequence. A formula φ follows semantically (weakly/strongly) from a knowledge base Δ ($\Delta \models \varphi$ or $\Delta \models\!\!\!\equiv \varphi$), iff the translation of Δ into logic entails φ (weakly/strongly).

11 Definition (Consistency): A knowledge base Δ is called (weakly/strongly) *consistent* iff there is no formula φ so that $\Delta \models \varphi$ and $\Delta \models \neg\varphi$ (or $\Delta \models\!\!\!\equiv \varphi$ and $\Delta \models\!\!\!\equiv \neg\varphi$, respectively).

12 Definition (Conservativity): An extension Δ of a knowledge base Δ' is called (weakly/strongly) *conservative* iff for all formulae φ holds: If $\varphi \in \mathcal{L}(\Delta')$ then $\Delta' \models \varphi$ iff $\Delta \models \varphi$ (or with $\models\!\!\!\equiv$ instead of $\models$, respectively).

13 Remark: In particular by a conservative extension we cannot import any contradiction. If the knowledge base Δ' is consistent and Δ is a conservative extension of Δ', then Δ is also consistent, because otherwise we could deduce from Δ a formula φ and its negation $\neg\varphi$ and by this any formula in $\mathcal{L}(\Delta)$, and hence any in $\mathcal{L}(\Delta')$, so we would have $\Delta \models \varphi$ and $\Delta \models \neg\varphi$.

14 Definition (Definition-Conservativity): A knowledge base is called *definition-conservative* iff every definition is a conservative extension.

15 Remark: We would expect now that the logics $\mathcal{L}^n$ are definition-conservative. Unfortunately due to our definition of higher-order logics and their semantics this is not the case in general as can be seen in the next example.

16 Example: Let Δ consist of the following axioms:

- $a : \iota,\ b : \iota,\ R : (\iota \to o)$
- $\forall P{:}(\iota \times \iota \to o)\ \forall x, y{:}\iota\ \ P(x, y) \iff P(y, x)$
- $R(a) \wedge \neg R(b)$

These axioms are consistent since we can give a weak model: $\mathcal{D}_\iota = \{1, 2\}$, $\mathcal{D}_{(\iota\times\iota\to o)}$ consists only of the binary relations that map everything to $\mathtt{T}$. $\mathcal{J}(a) = 1$, $\mathcal{J}(b) = 2$ $\mathcal{J}(R)(1) = \mathtt{T}$, $\mathcal{J}(R)(2) = \mathtt{F}$. This is of course no longer a model if we "define" a new binary predicate Q by $\forall x, y\ Q(x, y) :\iff R(x) \wedge \neg R(y)$ and add this to our knowledge base. We have $Q(a, b)$ since $R(a)$ and $\neg R(b)$. On the other hand by the commutativity axiom we get $Q(b, a)$, hence $\neg R(a)$ and $R(b)$. That is, now we have a contradiction in our knowledge base. Hence $\mathcal{L}^n$ is not definition-conservative for $n > 1$. This cannot happen if we have all comprehension axioms in the knowledge base (compare definition 18).

17 Definition (Extensionality Axioms): The *extensionality axioms* Ξ are the following formulae:

Ξ^f For all function symbols f, g of type $\tau = (\tau_1 \times \cdots \times \tau_m \to \sigma)$, $\sigma \neq o$:
$\forall f \ \forall g \ (\forall x_{\tau_1} \ldots \forall x_{\tau_m} \ f(x_{\tau_1}, \ldots, x_{\tau_m}) \equiv g(x_{\tau_1}, \ldots, x_{\tau_m})) \Longrightarrow f \equiv g$

Ξ^p For all predicate symbols p, q of type $\tau = (\tau_1 \times \cdots \times \tau_m \to o)$:
$\forall p \ \forall q \ (\forall x_{\tau_1} \ldots \forall x_{\tau_m} \ p(x_{\tau_1}, \ldots, x_{\tau_m}) \iff q(x_{\tau_1}, \ldots, x_{\tau_m})) \Longrightarrow p \equiv q$

18 Definition (Comprehension Axioms): The *comprehension axioms* Υ are the following formulae:

Υ^f For every term t of type $\tau \neq o$ of which the free variables are at most the different variables $x_1, \ldots, x_m, y_1, \ldots, y_k$ of type $\tau_1, \ldots, \tau_m, \sigma_1, \ldots, \sigma_k$:
$\forall y_1 \ldots \forall y_k \ \exists f_{(\tau_1 \times \cdots \times \tau_m \to \tau)} \ \forall x_1 \ldots \forall x_m \ \ (f(x_1, \ldots, x_m) \equiv t)$.

Υ^p For every formula φ of which the free variables are at most the different variables $x_1, \ldots, x_m, y_1, \ldots, y_k$ of type $\tau_1, \ldots, \tau_m, \sigma_1, \ldots, \sigma_k$:
$\forall y_1 \ldots \forall y_k \ \exists p_{(\tau_1 \times \cdots \times \tau_m \to o)} \ \forall x_1 \ldots \forall x_m \ \ (p(x_1, \ldots, x_m) \iff \varphi)$.

19 Lemma: *Every knowledge base Δ is definition-conservative for implicit definitions.*

Proof: This lemma holds trivially because we have required for implicit definitions that they must contain a proof that the defined objects exist and are unique. ■

20 Lemma: *$\mathcal{L}^1$ is definition-conservative.*

Proof: Inductive definitions are not possible in $\mathcal{L}^1$, hence we have to show the property only for simple definitions. Let Δ be given and ϑ, the extending frame of Δ, may be equivalent to $\forall x_1, \ldots, x_m$**<Name>**$(x_1, \ldots, x_m) \iff s_1 \wedge \ldots \wedge s_m \wedge \varphi$ where φ is the formula in the definition slot of the frame and the s_i are the entries of the superconcepts slot. So we can replace any instance of **<Name>**$(x_1, \ldots, x_m)$ by the corresponding instance of $s_1 \wedge \ldots \wedge s_m \wedge \varphi$. Since the two formulations are equivalent, the extension is conservative. In the case of a mapping definition the defined term is substituted instead of the formula. Of course we can make this replacement also, when preconditions are present. ■

21 Lemma: *If a knowledge base Δ contains the comprehension axioms Υ, then it is definition-conservative for simple definitions.*[*]

Proof: Let $\mathcal{S}'$ be the signature of Δ' and $\mathcal{S} = \mathcal{S}' \cup \{$**<Name>**$\}$ be the signature of $\Delta' \cup \{\vartheta\}$. ϑ is equivalent to $\forall x_1, \ldots, x_m$ **<Name>**$(x_1, \ldots, x_m) \iff s_1 \wedge \ldots \wedge s_m \wedge \varphi$

[*]We do not need such axioms if we use for our higher-order logic the λ-calculus, since we get the corresponding constant simply by λ-abstraction. The comprehension axioms cannot explicitly be in our knowledge base, because we have not provided facilities for introducing axiom schemata, but it can be easily tested whether a formula is a comprehension axiom or not.

where φ is the formula in the definition slot of the frame and the s_i are the entries of the superconcepts slot. By a comprehension axiom we get a predicate P so that $\forall x_1, \ldots, x_m P(x_1, \ldots, x_m) \iff s_1 \wedge \ldots \wedge s_m \wedge \varphi$. So we can replace any instance of **<Name>**$(x_1, \ldots, x_m)$ by the corresponding instance of $P(x_1, \ldots, x_m)$.

For definitions of the type "mapping" we can proceed analogously. ■

22 Lemma: *If a knowledge base Δ contains the comprehension axioms Υ, then it is definition-conservative for inductive definitions.*

Proof: Let Δ be given and ϑ be an inductive definition. Let Q be the $m+1$-place predicate $(m \geq 0)$ of sort $(\kappa \times \kappa_1 \times \cdots \times \kappa_m \to o)$ that is defined per induction on the first argument, then the inductive definition is of the form:

base: $\forall x_1, \ldots, x_m\, Q(c_j, x_1, \ldots, x_m) \iff \psi_j$ for $j = 1, \ldots, n$ where c_j are all constructor symbols of sort κ for the inductively introduced concept κ.

step: $\forall x_1, \ldots, x_m \forall y, y_1, \ldots, y_{k_i}\; Q(f_i(y, y_1, \ldots, y_{k_i}), x_1, \ldots, x_m) \iff \psi_i$ for all constructor functions f_i where in the ψ_i no constructor function can occur.

Now we have to show that there is exactly one predicate Q that satisfies this definition. By inductively applying the defining equivalences we get that for all constants d of sort κ, we can equivalently replace $Q(d, x_1, \ldots, x_m)$ by an expression not containing Q. Consequently for all elements d of sort κ there is a simple definition of Q for this element as argument. Hence we can conclude that the general definition of Q forms a conservative extension. ■

Summarizing we have:

23 Lemma: *If a knowledge base Δ contains the comprehension axioms Υ, then it is definition-conservative.*

24 Lemma: *If Δ is a consistent knowledge base and ϑ a theorem relative to Δ, then $\Delta \cup \{\vartheta\}$ is consistent.*

Proof: Because the deductive closures of Δ and $\Delta \cup \{\vartheta\}$ are the same, the theorem holds trivially. ■

25 Lemma: *The empty knowledge base $\Delta_\emptyset$ is consistent.*

Proof: Since for all formulae φ with $\Delta_\emptyset \models \varphi$, φ is a tautology and φ and $\neg\varphi$ cannot be tautologies at once, $\Delta_\emptyset$ must be consistent. ■

26 Theorem: *If Δ is a consistent knowledge base that contains the comprehension axioms Υ and ϑ is a concept definition relative to Δ or a theorem relative to Δ, then $\Delta \cup \{\vartheta\}$ is consistent.*

Proof: This follows immediately from the fact that concept definitions form conservative extensions and by conservative extension no contradiction can be imported, and by lemma 24. ■

27 Remark: In our considerations we have neglected the "context" slot. It could be integrated by defining a partial order on the contexts and by sharpening the definition of a frame relative to a knowledge base (compare definition 10) in that way that the required properties have not to hold only for the whole knowledge base Δ and ϑ, but even for that subpart of the knowledge base that consists of those modules, which are in the reflexive, transitive closure of the module, to which ϑ belongs.

5 Critique of the Frame Approach

In this section we summarize the advantages of frames for the representation of mathematical concepts and state the main disadvantage.

The first reason why we use frames is the *concept oriented* way of representation. Frames enable us to represent the knowledge in such a way that everything what belongs immediately to a concept is represented in the concept. In other words the concept is not given by a mere definition of it, but by a definition plus a set of important properties. Definitions and theorems need not be distributed over the knowledge base, but are structured. If we wrote our definitions and theorems in logic itself, we would have no structure facilities. The frames provide this structure facilities.

By the frame approach we can distinguish so-called *primitives*, that are, axioms, definitions, and theorems. Every primitive has its own status, in logic we only have formulae and do not distinguish between axioms, definitions, and theorems. These are meta-logical features of formulae. Here we have the possibility to require a special form for definitions, so that they are really definitions and cannot import any contradictions into a knowledge base. Theorems can be arbitrary formulae, but in contrast to axioms, they must be proved in order to be used in the proofs of other theorems. Furthermore we can guarantee that we use only defined concepts and exclude the case of "ignotum per ignotum". The consistency of knowledge bases is widely ensured.

Frames have the necessary *strength* to represent the required properties. For instance KL-ONE is not strong enough to represent a concept hierarchy, where an "abelian_group" is a superconcept of "field", since it is necessary to specify parameters, relative to which the hierarchic relation holds. Furthermore we can choose our slots, that is, primitives for describing concepts in an adequate way.

We can give (and have given) to the frames a *clear semantics*, what is important if one compares it to the situation of semantic networks, where years after building up large knowledge bases the untenability of the approach had to be stated, because it was impossible to give a clear semantics for the is-a hierarchy as long as "is-a" was used for

$\in$ *and* $\subseteq$.

Another advantage compared to most other representation formalisms is the *flexibility* of frames, that is, it is easily possible to add new features to the frames. In particular it will be necessary to add meta-knowledge how to use all this knowledge. The usage of the knowledge in the frames is of course a very difficult problem, which will be left to the user for the beginning. Some heuristic information about the usage belongs to the concept. How this knowledge can be represented has to be cleared in the context of higher problem solving methods like proof planning and tactical theorem proving.

The main disadvantage of the proposed representation formalism is, that structuring, modularization, and classifications must be done by the user. Almost no automation can be expected in such a general approach.

Acknowledgement

I like to thank to all persons which supported me writing my Ph.D.thesis, of which this paper is one part. Especially I like to thank Prof. JÖRG H. SIEKMANN for many motivating talks. Many improvements of this thesis are due to his suggestions. For the strenuous work of reading drafts of this part of my thesis I have to thank MICHAEL KOHLHASE and AXEL PRÄCKLEIN. Their suggestions lead to several improvements. Of course, all remaining errors are mine.

DIRK SCHRÖDER and RALF KOERSTEIN worked on building up a knowledge base and used this knowledge for proving theorems with the Markgraf Karl Refutation Procedure [12]. Since at that time the theory was not yet clear this work was not always easy, but helped much in developing our theory.

References

[1] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof.* Academic Press, Orlando, Florida, USA, 1986.

[2] Stephen Blamey. Partial logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, chapter III.1, pages 1–70. D.Reidel Publishing Company, Dodrecht, Netherlands, 1986. Volume III: Alternatives to Classical Logic.

[3] Robert S. Boyer and J Strother Moore. *A Computational Logic.* Academic Press, New York, USA, 1979.

[4] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, **5**:56–68, 1940.

[5] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, **38**:173–198, 1931.

[6] Kurt Gödel. *The Consistency of the Axiom of Choice and of the Generalized Continuum-Hypothesis with the Axioms of Set Theory*, volume 3 of *Annals of Mathematics Studies.* Princeton University Press, Princeton, New Jersey; eighth printing 1970, 1940.

[7] Jacques Hadamard. *The Psychology of Invention in the Mathematical Field.* Dover Publications, New York, USA; edition 1949, 1944.

[8] Patrick J. Hayes. The logic of frames. In Ronald J. Brachman and Hector J. Levesque, editors, *Readings in Knowledge Representation*, chapter 14, pages 287–295. Morgan Kaufmann, 1985, San Mateo, California, USA, 1979. also in: *Frame Conceptions and Text Understanding*, p.46–61, D. Metzing, editor, Berlin, Germany, Walter de Gruyter.

[9] Dieter Hutter. Vollständige Induktion. In Karl Hans Bläsius and Hans-Jürgen Bürckert, editors, *Dedukionssysteme - Automatisierung des logischen Denkens*, chapter V, pages 153–172. Oldenbourg, München, Germany, 1987.

[10] Manfred Kerber. *On the Representation of Mathematical Concepts and their Translation into First Order Logic.* PhD thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, 1992, forthcoming.

[11] Lothar Kreiser, Siegfried Gottwald, and Werner Stelzner, editors. *Nichtklassische Logik*, volume I. Akademie Verlag, Berlin, Germany, 1990.

[12] Karl Mark G Raph. The Markgraf Karl Refutation Procedure. Technical Report Memo-SEKI-MK-84-01, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany, January 1984.

[13] Marvin Minsky. A framework for representing knowledge. In Patrick Henry Winston, editor, *The Psychology of Computer Vision.* McGraw-Hill, New York, USA, 1975. also in: *Mind Design*, pages 95–128, J. Haugeland, editor, Cambridge, Massachusetts, USA, MIT-Press, 1981, and *Readings in Knowledge Representation*, pages 245–262, chapter 12, Ronald J. Brachman and Hector J. Levesque, editors, San Mateo, California, USA, Morgan Kaufmann, 1985.

[14] Nils J. Nilsson. *Principles of Artificial Intelligence.* Morgan Kaufman, San Mateo, California, USA, 1980.

[15] George Pólya. *How to Solve It.* Princeton University Press, Princeton, New Jersey, USA, also as Penguin Book, 1990, London, United Kingdom, 1945.

[16] George Pólya. *Mathematics and Plausible Reasoning.* Princeton University Press, Princeton, New Jersey, USA, 1954. Two volumes, Vol.1: Induction and Analogy in Mathematics, Vol.2: Patterns of Plausible Inference.

[17] George Pólya. *Mathematical Discovery – On understanding, learning, and teaching problem solving.* Princeton University Press, Princeton, New Jersey, USA, 1962/1965. Two volumes, also as combined edition, 1981, John Wiley and Sons, New York, USA.

[18] Bartel L. van der Waerden. Wie der Beweis der Vermutung von Baudet gefunden wurde. *Abh. Math. Sem. Univ. Hamburg*, **28**:6–15, 1964.

[19] Bartel L. van der Waerden. *Algebra I*, volume 12 of *Heidelberger Taschenbücher*. Springer Verlag, Berlin, Germany, eighth edition, 1971.