A category-theoric approach to completion-based theorem proving strategies *

Maria Paola Bonacina **Jieh Hsiang**
Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794-4400, USA
{bonacina,hsiang}@sbcs.sunysb.edu

Abstract

In this paper we apply *category theory* to formalize the basic concepts of automated theorem proving. A *theorem proving strategy* is given by a set of inference rules and a search plan. For the *inference rules*, we propose a new characterization of inference rules as *natural transformations*. Traditionally, inference rules are presented either as functions or as rules to transform sets of sentences. Neither description is completely satisfactory, especially if *contraction inference rules*, i.e. rules that delete sentences, are involved. Our view of inference rules as natural transformation couples the advantages of the two traditional approaches: it applies to both expansion and contraction rules, while still saving the functional nature of inference rules.

The *search plan* selects at each stage of a derivation the inference rule and the premises for the next step. In most theorem proving strategies the search plans are described informally or left altogether to the implementation phase. This is not satisfactory, since the actual performance of a prover depends heavily on the search plan. We give a new, abstract definition of search plan and we define precisely how the inference rules and the search plan cooperate to generate a derivation. To our knowledge, this is the first mathematical definition of search plan.

In our previous work on completion procedures, we characterized completion-based theorem proving strategies by three properties: *monotonicity*, *relevance* and *proof reduction*. In fact, these properties capture the essential aspects of any theorem proving process. We show that they are *functoriality properties*: this result clarifies what part of the structure of a theory a theorem proving derivation is required to preserve. We close the paper with a comparison with related work and a discussion on further extensions of our categorical approach.

Keywords: category theory, theorem proving, simplification rules, search mechanisms.

*Research supported in part by grant CCR-8901322, funded by the National Science Foundation. The first author is also supported by a scholarship of Università degli Studi di Milano, Italy. An abstract of this paper appears in *CATEGORY THEORY 1991*, Mc Gill University, Montréal, Canada, June 1991.

1 Introduction

In this paper we extend the *categorical approach to logic* introduced in [24] to the framework for *automated theorem proving* that we have developed in [8, 5, 6]. The idea of applying category theory to logic appeared in the seminal work of F.W.Lawvere [21] on equational logic (see also [23, 27]). More recently, the growth of interest in automated theorem proving and logic programming in the computer science community has spurred a new wave of attention for the application of this tradition of thought to logic in computer science [12, 17, 24]. In [24], Meseguer presents an axiomatization of logic in category theory, which focuses on those elements of logic that are most relevant to computer science. In Section 2 of this paper we present the elements of the work in [24] that we are going to use in our application to theorem proving.

Our research aims at extending the categorical approach in [24] from a description of logic for computer science to a description of theorem proving methods, with a special interest in *Knuth-Bendix type completion procedures* for theorem proving [20, 16, 2, 15, 3]. This interest is largely justified by the remarkable successes that completion-based theorem provers have obtained in recent years (see for instance [18, 1] and [10] for a survey). In [8, 5] we have given a new abstract framework for these methods. According to this framework, a *theorem proving problem* is specified by a pair $(S;\varphi)$, where S is a set of sentences, the *presentation* of a theory, and φ is the theorem to be proved from S, called the *target*. A *theorem proving strategy* $\mathcal{C}$ has two components, a set of *inference rules* I and a *search plan* Σ. The inference rules determine what consequences can be derived from a given set of sentences. The search plan decides which inference rule to apply to which sentences at each step of a *derivation*

$$(S_0;\varphi_0)\vdash_{\mathcal{C}}(S_1;\varphi_1)\vdash_{\mathcal{C}}\ldots\vdash_{\mathcal{C}}(S_i;\varphi_i)\vdash_{\mathcal{C}}\ldots.$$

At each stage $(S_i;\varphi_i)$ the problem is to prove the target φ_i from the presentation S_i. The key concept in our approach is to regard the problem of proving φ_i from S_i as the problem of *reducing a minimal proof of φ_i in S_i*. A theorem proving derivation is a *process of reduction*: reduction of proofs with respect to some *well-founded proof ordering*. The derivation is successful if it reaches a stage where the proof of φ_i in S_i is *empty*, i.e. φ_i is a trivially true theorem in S_i. This idea of proof reduction is the pivot of our entire approach and all other notions in theorem proving are defined accordingly [8, 5, 6].

In this paper we reformulate our framework in terms of category theory. First we characterize *inference rules as natural transformations*. The notion of natural transformation turns out to be much more flexible than the more basic notion of function. It allows us to cover in an elegant and uniform way both *expansion* inference rules, which derive new consequences, and *contraction* inference rules, which delete or replace given sentences. Next, we turn our attention to the *search plan*. The search plan selects at each stage of a derivation an inference rule and a tuple of premises for the next step. Therefore, the search plan is the *control* that turns a set of non-deterministic inference rules into a deterministic procedure. Traditionally, the inference rules are regarded as the logical component of theorem proving, whereas the search plan is the procedural component. As a consequence, search plans are often described informally or by giving algorithms. We feel that the procedural nature of search mechanisms does not eliminate the need for a rigourous

definition. Rather, search is such an essential component in theorem proving, that the problem of giving a systematic treatment of search plans and their properties is very important. We propose a definition of search plan as a pair of functions to choose an inference rule and a tuple of sentences and we show how the inference rules and the search plan interact to generate a derivation. This definition is the first mathematical definition of search plan that we are aware of and we regard it as a very first step toward a theory of search plans.

A theorem proving derivation has two underlying categories: the category of the *states* of a derivation, i.e. the pairs $(S_i; \varphi_i)$, and the category of the *proofs* that the derivation is reducing. Proof reduction is then expressed naturally as a *functoriality property*: a derivation is proof-reducing if the function that associates to a state $(S_i; \varphi_i)$ a minimal proof of φ_i in S_i is a functor from the category of the states of the derivation to the category of the proofs. Similarly, *monotonicity* and *relevance*, which are the soundness properties of a theorem proving derivation, consist in the existence of functors from the category of the states of the derivation to other appropriately defined categories.

We have used a surprisingly small number of notions of category theory: the reader who is not familiar with categories, functors and natural transformations may find all the necessary background in the first chapter of [22]. For the notation and terminology relative to completion procedures we refer to [10].

2 Elements of a categorical approach to logic

We introduce those elements of the categorical approach to logic in [24] that we are going to use in the following. The interested reader may find in [24] a full account of this approach.

The data of a theorem proving computation are *sentences* made of symbols from some *signature*. For instance in first order logic, a signature Θ is a pair (F, P), where F is a set of function symbols with their arities and P is a set of predicate symbols with their arities. For such signatures, a morphism $h\colon \Theta \to \Theta'$ has two components, two arity-preserving functions, mapping the function symbols of Θ into function symbols of Θ' and the predicate symbols of Θ into predicate symbols of Θ'. Signatures and their morphisms form the category $\underline{Sign}$. If there is a morphism $h\colon \Theta \to \Theta'$, a sentence φ on signature Θ can be translated into a sentence $h(\varphi)$ on signature Θ' by replacing each symbol in Θ by the corresponding symbol in Θ'. It is then possible to define a functor $sen\colon \underline{Sign} \to \underline{Set}$ from the category of signatures to the category of sets: sen associates to a signature Θ the set $sen(\Theta)$ of its sentences and to a morphism $h\colon \Theta \to \Theta'$ the function $sen(h)\colon sen(\Theta) \to sen(\Theta')$ translating sentences according to h.

Given signatures and sentences, the familiar logical notion of *entailment* is introduced as a function $\vdash$, associating to each Θ in $\underline{Sign}$ its entailment relation $\vdash_\Theta \subseteq \mathcal{P}(sen(\Theta)) \times sen(\Theta)$, where $\mathcal{P}$ is the power set functor $\mathcal{P}\colon \underline{Set} \to \underline{Set}$. For a set Γ of sentences and a sentence φ, both on signature Θ, $\Gamma \vdash_\Theta \varphi$ means that φ is derivable from Γ. The category $\underline{Sign}$, the functor sen and the function $\vdash$ define what is called in [24] an *entailment system* $(\underline{Sign}, sen, \vdash)$. Basically, an entailment system contains all that is necessary to state assertions of the form $\Gamma \vdash_\Theta \varphi$.

The next step is to define a category $\underline{Th}$ of *theories*: its objects are pairs (Θ, Γ), where Θ

is a signature and Γ is a set of sentences, $\Gamma \subset sen(\Theta)$. A morphism of theories, $h: S \to S'$, for $S = (\Theta, \Gamma)$ and $S' = (\Theta', \Gamma')$ is a morphism of signatures $h: \Theta \to \Theta'$ with the property that $\Gamma' \vdash_{\Theta'} h(\Gamma)$. In other words, the new theory entails the old one. If the old axioms are preserved, i.e. $h(\Gamma) \subseteq \Gamma'$, the morphism is said to be *axiom-preserving*. The subcategory that has all the objects of $\underline{Th}$ and only axiom-preserving morphisms is denoted by $\underline{Th}_0$. The functor $sign: \underline{Th} \to \underline{Sign}$, defined by $sign(\Theta, \Gamma) = \Theta$, extracts the signature of a given theory. The functor $sen: \underline{Sign} \to \underline{Set}$ can be extended to a functor $sen: \underline{Th} \to \underline{Set}$ by defining $sen(S) = sen(sign(S))$. The function $\vdash$ can also be extended to theories: it associates to a theory $S = (\Theta, \Gamma)$ the relation $\vdash_S$ defined by $\Delta \vdash_S \varphi$ if and only if $\Delta \cup \Gamma \vdash_\Theta \varphi$.

One of the motivations for this definition of theory morphisms in [24] is its application to the definition of programs in a declarative language. In equational programming languages such as those of the OBJ family [11, 13, 19], a program is a collection of modules and a module is an equational theory, that is a signature and a set of equations. The process of defining modules proceeds by extensions: one may start with a small set of symbols and equations and extend it with new symbols and equations as the need of adding new operators arises. Such an enlargement is justified as a theory morphism mapping the old module into the new, larger one. Also, axiom-preserving morphisms are often sufficient in this context, since new, larger modules simply include the previously defined ones. Interestingly, we shall see in the following that this notion of theory morphism and especially of axiom-preserving morphisms does not have an immediate application in theorem proving. The intuitive reason is that the incremental definition of programs is a process of enlarging theories, whereas theorem proving is inherently a process of reduction. Theory morphisms are defined to describe extensions of theories rather than reductions. However, category theory allows us to express both kinds of processes by using the notion of *opposite category*. We shall show that as the category $\underline{Th}$ is well suited for programming, its opposite $\underline{Th}^{op}$, i.e. the category which is identical to $\underline{Th}$ except that all arrows are inverted, is better suited for theorem proving.

The model theoretic counterpart of an entailment system is an *institution* [12]. Since our following treatment deals exclusively with proof theory and not with model theory, we introduce just one component of an institution, v.i.z. the function $\models$ that associates to each signature Θ its *satisfaction* relation $\models_\Theta$. Similar to entailment, satisfaction can be extended to theories: $\Delta \models_S \varphi$ for $S = (\Theta, \Gamma)$, if and only if $\Delta \cup \Gamma \models_\Theta \varphi$. An entailment system and an institution define a *logic*, provided the following *soundness* property holds: for all $\Theta \in \underline{Sign}$, $\Gamma \subset sen(\Theta)$ and $\varphi \in sen(\Theta)$, $\Gamma \vdash_\Theta \varphi$ implies $\Gamma \models_\Theta \varphi$. If the opposite implication also holds, the logic is said to be *sound* and *complete*. In the following we shall always assume that the logic underlying our theorem proving problem is sound and complete.

An entailment system defines entailment, but it does not provide any information about *the structure of proofs*. The structure of proofs is determined by giving a *proof calculus*. A *proof calculus* is presented as a 6-tuple $\mathcal{PC} = (\underline{Sign}, sen, \vdash, P, Pr, theorem)$ [1]. The first three components form an entailment system. The fourth component is a functor $P: \underline{Th}_0 \to \underline{Struct}_P$

[1] A proof calculus is denoted by $\mathcal{P}$ in [24], but we use $\mathcal{PC}$ in order to reserve $\mathcal{P}$ for the power set functor. Also the element that we call *theorem* is called π in [24], but we use *theorem* to avoid any potential confusion with the projection functions π_{ni}.

that associates to a theory its *proof structure*. The category $\underline{Struct}_P$ is replaced by a concrete category when a specific proof calculus is given. For instance a proof calculus for equational logic is obtained in [24] by instantiating $\underline{Struct}_P$ to $\underline{Cat}$, the category of all categories. Then for an equational theory (Θ, E), $P((\Theta, E))$ is the category whose objects are all the terms on signature Θ and whose morphisms $p\colon s \to t$ are the proofs $s \leftrightarrow_E^* t$ made of steps of equational replacement by the equations in E.

The fifth component of a proof calculus is a functor $Pr\colon \underline{Struct}_P \to \underline{Set}$ such that $Pr \circ P\colon \underline{Th}_0 \to \underline{Set}$ associates to a theory S the set of all the proofs in S. The composition $Pr \circ P$ is called $proofs$, so that $proofs(S)$ is the set of all the proofs in S. Finally, $theorem\colon proofs \Rightarrow sen$ is a natural transformation from the functor $proofs$ to the functor sen: for all theories S, there exists a function $theorem_S\colon proofs(S) \to sen(S)$ that extracts from a proof its theorem. It is important to remark [24] that P and therefore $proofs$ are functors on $\underline{Th}_0$ but not on $\underline{Th}$. In other words, they can preserve axiom-preserving morphisms only. This is explained as follows: in order to be a functor on $\underline{Th}$, $proofs$ should associate to each theory morphism $h\colon S \to S'$ a function $proofs(h)\colon proofs(S) \to proofs(S')$ mapping proofs in S into proofs in S'. It can be easily observed that if h is not axiom-preserving, $proofs(h)$ is not well defined. Let $h\colon S \to S'$ be a theory morphism which is not axiom-preserving, ψ be an axiom of S such that $h(\psi) \notin S'$ and Υ be a proof of a theorem φ in S, where ψ is used as axiom. Furthermore we can assume for simplicity that ψ is the only axiom of S used in Υ such that $h(\psi) \notin S'$. The natural choice for $proofs(h)(\Upsilon)$ is the proof which is identical to Υ, with all sentences in Υ translated according to h and the occurrence of ψ replaced by a proof of $h(\psi)$ in S'. The definition of theory morphism ensures that a proof of $h(\psi)$ in S' exists. However, the proof of $h(\psi)$ in S' is *not unique* in general and therefore $proofs(h)(\Upsilon)$ is not uniquely defined. In the following, we use $proofs$ as a function $proofs\colon \underline{Th} \to \underline{Set}$, keeping in mind that only its restriction to $\underline{Th}_0$ is a functor.

This categorical approach allows us to present the concepts of our framework for theorem proving abstracting from any specific logic. We do not have to choose a logic a priori. We simply use those categories, such as $\underline{Sign}$ and $\underline{Th}$, that capture those elements that are common to all logics. This generality is especially valuable to us, since our framework for theorem proving is independent of the logic.

3 A categorical approach to theorem proving

In this section we define in terms of basic concepts of category theory the components of a theorem proving strategy $\mathcal{C} =< I; \Sigma >$, where I is the set of *inference rules* and Σ is the *search plan*, together with their properties.

3.1 The inference rules

Inference rules are usually written in the form

$$f\colon \frac{\psi_1 \ldots \psi_n}{\psi}$$

which says that given premises $\psi_1 \dots \psi_n$, the rule f yields ψ. A classical way of interpreting this notation is to regard f as a *function* that, applied to n elements of types $\psi_1 \dots \psi_n$, gives as result an element of type ψ, written as $f\colon \psi_1 \dots \psi_n \to \psi$ [14, 17].

The functional nature of inference rules is not the only aspect that is relevant in theorem proving. At each step of a theorem proving derivation an inference rule f is applied to transform the current data base $(S; \varphi)$. The tuple of premises $\psi_1 \dots \psi_n$ is selected from $(S; \varphi)$. Then f infers ψ from $\psi_1 \dots \psi_n$ and adds ψ to $(S; \varphi)$. The above functional notation does not show explicitly that ψ is added to the data base. More importantly, the effect of f does not necessarily consists in adding ψ: f may *replace* by ψ one or more of the premises $\psi_1 \dots \psi_n$. For instance the rule *Simplification* [26] applies an equation to reduce a sentence to a simpler form: given premises[2] $\psi[s]$ and $s \simeq t$, $\psi[s]$ is deleted and replaced by $\psi[t]$ if $\psi[s] \succ \psi[t]$ for some well-founded ordering $\succ$. If we write simplification in the functional form, it says that $\psi[t]$ is derived from $\psi[s]$ and $s \simeq t$, but it does not describe the full effect of the inference, that is the replacement of $\psi[s]$ by $\psi[t]$. Since the replacement of expressions by simpler and equivalent ones is the essence of simplification, the functional formalism does not convey the meaning of simplification. Even worse, there are inference rules that do not add any sentence, but simply delete one: for example *Subsumption* [9] deletes a clause which is logically implied by another clause in the set. The functional description does not apply to such rules.

We have called the inference rules that add new sentences *expansion inference rules* and those that may delete sentences *contraction inference rules* [8]. Simplification and subsumption are contraction inference rules. *Resolution* [25] is a classical example of an expansion inference rule. The functional view of inference rules applies naturally to expansion rules, but not to contraction rules. Since contraction rules, and especially simplification, are the most important rules in completion based methods, a different way of describing inference rules has been adopted for completion based strategies [2]. Inference rules are presented as rules to transform sets:

$$f\colon \frac{S}{S'}$$

where S and S' are sets of sentences. The rule says that given S, the set S' can be inferred. Then, the general forms of expansion and contraction inference rules are respectively:

$$f\colon \frac{S}{S'} \text{ where } S \subset S'$$

and

$$f\colon \frac{S}{S'} \text{ where } S \not\subseteq S'.$$

For example simplification can be written as

$$\frac{S \cup \{\psi[l\sigma], l \simeq r\}}{S \cup \{\psi[r\sigma], l \simeq r\}}\ \psi[l\sigma] \succ \psi[r\sigma],$$

where σ denotes a substitution. An inference step on $(S; \varphi)$ may expand or contract S or derive a new target φ' from φ and S.

[2]The notation $\psi[s]$ indicates a sentence ψ where s appears as subterm.

The main advantage of displaying inferences as transformations of sets is to show the global effect of an inference step on the data base. It emphasizes whether the data base is expanded or contracted. However, in this formalism the functional aspect of inference rules is not visible, because inference rules are not functions of sets. Given a set of sentences and an inference rule, the inference rule can be applied to different selections of premises in the given set, yielding different sets as result. Only after the premises have been selected the result is uniquely determined. Category theory allows us to give a definition of inference rules that captures both the functional and sets transformation aspects. We start by observing what inference rules have in common. An inference rules has a tuple of premises and it indicates a set of new elements to be added and a set of elements to be deleted. For example, in binary resolution the tuple of premises is the pair of parents, the resolvent is added and no clause is deleted. Simplification also has a pair of premises, an equation $s \simeq t$ and a sentence $\psi[s]$: the effect of the step is to delete $\psi[s]$ and to add $\psi[t]$. Thus, we would like inference rules to be functions from tuples of sentences, the premises, to pairs of sets of sentences, the set to be added and the set to be deleted. Furthermore, inference rules are independent from the signature: for every signature there is the appropriate instance of the inference rule. In order to abstract from the signature, we define inference rules as *natural transformations from tuples of sentences to pairs of sets of sentences*:

Definition 3.1 *Given an entailment system* $(\underline{Sign}, sen, \vdash)$*, an inference rule* f^n *of arity* n *is a natural transformation*

$f^n \colon \mathcal{L}^n \circ sen \Rightarrow (\mathcal{P} \circ sen) \times (\mathcal{P} \circ sen)$,

where $\mathcal{L}^n \colon \underline{Set} \to \underline{Set}$ *is the functor that maps a set into the set of its tuples.*

By definition of natural transformation [22], this means that for all signatures Θ we have a function

$f^n_\Theta \colon \mathcal{L}^n \circ sen(\Theta) \to (\mathcal{P} \circ sen(\Theta)) \times (\mathcal{P} \circ sen(\Theta))$

from tuples of sentences to pairs of sets of sentences of Θ. For an input tuple $\bar{x}$, the first component[3] of the output $\pi_{21}(f^n_\Theta(\bar{x}))$ is the set of sentences to be added and the second component $\pi_{22}(f^n_\Theta(\bar{x}))$ is the set of sentences to be deleted. It is trivial to verify that the commutativity property required by the definition of natural transformation is satisfied, since an inference step commutes with a signature morphism $h \colon \Theta \to \Theta'$:

$$\begin{array}{ccc} \mathcal{L}^n \circ sen(\Theta) & \xrightarrow{\quad f^n_\Theta \quad} & (\mathcal{P} \circ sen)^2(\Theta) \\ {\scriptstyle \mathcal{L}^n \circ sen(h)} \downarrow & /\!/\!/ & \downarrow {\scriptstyle (\mathcal{P} \circ sen)^2(h)} \\ \mathcal{L}^n \circ sen(\Theta') & \xrightarrow[\quad f^n_{\Theta'} \quad]{} & (\mathcal{P} \circ sen)^2(\Theta') \end{array}$$

The following example shows how this scheme applies to simple steps of resolution, simplification and subsumption:

Example 3.1 $resolution(P(0), \neg P(x) \vee P(s(x))) = (\{P(s(0))\}, \emptyset)$,

$simplification(x + 0 \simeq x, P(s(x) + 0)) = (\{P(s(x))\}, \{P(s(x) + 0)\})$,

[3] We use π_{ni} to denote the projection that extracts the i-th element of a tuple.

$$subsumption(P(x), P(s(y))) = (\emptyset, \{P(s(y))\}).$$

The distinction between expansion and contraction rules can be formalized as follows:

Definition 3.2 *An inference rule* f^n *is an* expansion inference rule *if for all signatures* Θ *and inputs* $\bar{x} \in \mathcal{L}^n \circ sen(\Theta)$, $\pi_{22}(f^n_\Theta(\bar{x}))$ *is* empty. *It is a* contraction inference rule *otherwise.*

An inference rule may not apply to a given input sentence. For instance resolution does not apply to two parents such that no pair of their literals unify. Our definition covers this case without resorting to partial functions: if the inference rule does not apply to the given input, both output sets are empty. This reflects what happens in a theorem proving derivation: if the search plan has selected an inference rule and a tuple of premises such that the inference rule does not apply to the premises, the attempted application fails and the data base remains unchanged.

To summarize, the above description of inference rules as natural transformations has three basic advantages: it is independent of the signature, it expresses both expansion and contraction and it covers also the situation where the inference rule does not apply to the selected premises. These three properties are all very important in the theorem proving context. Abstraction from the signature is obviously desirable. Expansion versus contraction is a key issue in theorem proving. In logic it is sufficient to describe an inference rule by specifying how and under which conditions the inferred sentences are constructed from the premises. From the point of view of logic, one is mainly concerned with the existence of proofs and their structure. All inferences can be regarded as expansion inferences; the distinction between expansion and contraction is not relevant. In automated theorem proving we are also concerned with how much memory space and how much time a procedure needs to prove theorems. Given that a proof exists, the problem usually bogs down to whether the strategy will halt with a proof or will run out of memory. Since expansion steps consume space while contraction steps free space, it is important in theorem proving to classify them separately.

Finally, the problem of incorporating in the definition of inference rules the case where the inference rule fails to apply to the chosen premises is also related to the concrete experience of theorem proving. Each successful step in a derivation may be preceded in general by many failed attempts: for instance the program may have selected simplification as inference rule and it may have tried many pairs of premises before it finds one such that simplfication does indeed apply. Therefore it is important that inference rules are well defined even when their applicability conditions are not satisfied by the selected premises.

3.2 The derivations

A theorem proving derivation by $\mathcal{C} =< I; \Sigma >$ is a sequence

$$(S_0; \varphi_0) \vdash_{\mathcal{C}} (S_1; \varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \varphi_i) \vdash_{\mathcal{C}} \ldots$$

where at each stage i the problem is to prove φ_i from S_i. The pair $(S_i; \varphi_i)$ is the *state* of the derivation at stage i. Since the states of a derivation are pairs $(S; \varphi)$, we introduce a set *States*, whose elements are all the pairs $(S; \varphi)$, with $S \in \underline{Th}$ and $\varphi \in sen(S)$. The set *States* contains the

states of all possible derivations. During a derivation, a state is transformed by inference steps. Therefore, inference steps are a natural choice for *morphisms of states*. If a set of inference rules I is given, the set *States* becomes a category $\underline{States}$ with morphisms induced by I:

Definition 3.3 *Given an entailment system* $(\underline{Sign}, sen, \vdash)$ *and a set of inference rules* I*, the category* $\underline{States}$ *has as objects all the pairs* $(S;\varphi)$*, with* $S \in \underline{Th}$ *and* $\varphi \in sen(S)$*. For all* $(S;\varphi)$ *and* $(S';\varphi')$ *in* $\underline{States}$*, there is a morphism* $d\colon (S;\varphi) \to (S';\varphi')$ *if and only if there exists a derivation* $(S;\varphi) \vdash_I^* (S';\varphi')$ *by* I*. A morphism* $d\colon (S;\varphi) \to (S';\varphi')$ *is a function such that*

$d(\varphi) = \varphi'$ *and*

$$\text{for all } \psi \text{ in } S,\ d(\psi) = \begin{cases} \psi & \text{if } \psi \in S' \\ \varphi' & \text{otherwise.} \end{cases}$$

A morphism of states $d\colon (S;\varphi) \to (S';\varphi')$ preserves the target by mapping φ into φ'. For the presentation, d is identity everywhere except on those sentences of S that have been deleted by contraction steps in the process of deriving S' from S. These sentences have been deleted because they are not necessary to prove the target φ'. Thus the morphism d maps them into φ', which is the "justification" of their deletion. Transitivity and reflexivity of $\vdash_I^*$ induce identity and composition of morphisms: the identity morphism maps every state into itself by an empty derivation and composition of morphisms corresponds to the concatenation of inference steps.

The category $\underline{States}$ describes all the derivations by the inference mechanism I. If a specific theorem proving problem $(S_0;\varphi_0)$ is given, the application of I to $(S_0;\varphi_0)$ defines a *subcategory* of $\underline{States}$. This subcategory contains all and only the states that can be derived from $(S_0;\varphi_0)$ by I and their morphisms. In [8, 5] we have represented this subcategory by the *I-tree rooted at* $(S_0;\varphi_0)$. The nodes of the tree are labeled by states, with the initial state $(S_0;\varphi_0)$ at the root. A node $(S;\varphi)$ has a child $(S';\varphi')$ if $(S';\varphi')$ can be derived from $(S;\varphi)$ in one step by I. Thus branches in the tree correspond to single inference steps and paths in the tree, called *I-paths*, correspond to derivations. If a search plan Σ is also given, the *unique* derivation computed by $\mathcal{C} =< I;\Sigma >$ starting from $(S_0;\varphi_0)$ is determined. A search plan Σ is a mechanism to select a path in an I-tree: the derivation from input $(S_0;\varphi_0)$ controlled by Σ is the I-path selected by Σ in the I-tree rooted at $(S_0;\varphi_0)$. This specific derivation $\underline{Der}$

$$(S_0;\varphi_0) \vdash_{\mathcal{C}} (S_1;\varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i;\varphi_i) \vdash_{\mathcal{C}} \ldots$$

is the *subcategory* of $\underline{States}$ that has as elements all the states $(S_i;\varphi_i)$, $i \geq 0$, generated during the derivation, and as morphisms all the sequences of inferences steps that are subsequences of the derivation. To summarize, a set of inference rules I induces the morphisms of $\underline{States}$; the set I and an initial state $(S_0;\varphi_0)$ specify an I-tree; if I, Σ and $(S_0;\varphi_0)$ are all given, a unique derivation is determined. In the next subsection we describe how a search plan selects this unique derivation.

3.3 The search plan

Given an input problem $(S_0;\varphi_0)$, let $(S;\varphi)$ be a node in the I-tree rooted at $(S_0;\varphi_0)$. The node $(S;\varphi)$ has many children in general, because I can be applied to modify $(S;\varphi)$ in many ways,

depending on which inference rule in I is used and which tuple of premises in $(S;\varphi)$ it is applied to. If $(S;\varphi)$ is the state $(S_i;\varphi_i)$ reached at stage i of a derivation, anyone of its children is a candidate to be the successor $(S_{i+1};\varphi_{i+1})$. The search plan selects only one of them to be $(S_{i+1};\varphi_{i+1})$ by selecting the inference rule to be applied and the premises for the application of the rule. Thus a search plan has two components, $\Sigma =< \zeta, \xi >$, a function ζ to choose the inference rule and a function ξ to choose the tuple of premises. The domains of ζ and ξ contain the elements based on which the choice is made. It is very natural to require that the choice depends on the current *state* and therefore ζ and ξ are functions of the state. However, the actual state of the derivation represents only local information, it does not say anything about the past history of the derivation. For instance a derivation may reach a state $(S;\varphi)$ after 100 steps, whereas another derivation may reach the same state $(S;\varphi)$ after $10,000$ steps. If ζ and ξ were functions of the state only, it would be impossible to define a search plan that makes different choices in state $(S;\varphi)$ depending on whether $(S;\varphi)$ has been reached after 100 steps or $10,000$ steps. Therefore, it is desirable to have in the domain of ζ and ξ a component representing some global information about the derivation. For instance, this component may be the *length of the derivation* generated so far. Different choices may be possible. However, it is not realistic to assume that a strategy may save in memory a large amount of information about the history of the derivation. A search plan which requires the strategy to do so is likely to increase its chances of running out of memory. Therefore it is sufficient to define ζ to be

$$\zeta\colon \underline{States} \times \mathbb{N} \to I$$

meaning that $\zeta((S;\varphi),m) = f^n$ is the inference rule selected in state $(S;\varphi)$ after m steps of derivation. Since ξ selects a tuple of sentences, its domain has an additional element n, which indicates the number of sentences to be chosen. Thus the domain of ξ is $\underline{States} \times \mathbb{N} \times \mathbb{N}$ and $\xi((S;\varphi),m,n)$ indicates the choice of n elements from $(S;\varphi)$ at stage m of the derivation. The codomain of ξ cannot be simply a set of tuple of sentences, because we want ξ to be independent from the signature. As codomain of ξ we give a set of natural transformation:

$$\xi\colon \underline{States} \times \mathbb{N} \times \mathbb{N} \to \{\alpha^n \mid \alpha^n\colon \mathcal{P} \circ sen \Rightarrow \mathcal{L}^n \circ sen, \forall n \geq 1\}.$$

By the definition of natural transformation applied to $\alpha^n\colon \mathcal{P} \circ sen \Rightarrow \mathcal{L}^n \circ sen$, for all signatures $\Theta \in \underline{Sign}$ there is a function

$$\alpha^n_\Theta\colon \mathcal{P} \circ sen(\Theta) \to \mathcal{L}^n \circ sen(\Theta)$$

that associates to a set of sentences a tuple of sentences. If $\xi((S;\varphi),m,n) = \alpha^n$ and $S = (\Theta,\Gamma)$, then in state $(S;\varphi)$ after m steps, the search plan selects the tuple $\alpha^n_\Theta(\Gamma \cup \{\varphi\})$. The following definition of search plan summarizes these elements:

Definition 3.4 *Given an entailment system $(\underline{Sign}, sen, \vdash)$ and a set of inference rules I, a search plan Σ for I is a pair of functions $\Sigma =< \zeta, \xi >$ of types*

- $\zeta\colon \underline{States} \times \mathbb{N} \to I$ *and*
- $\xi\colon \underline{States} \times \mathbb{N} \times \mathbb{N} \to \{\alpha^n \mid \alpha^n\colon \mathcal{P} \circ sen \Rightarrow \mathcal{L}^n \circ sen, \forall n \geq 1\}$.

We now have all the elements to define how the inference rules and the search plan cooperate to determine the steps of a derivation. First, we extend the functors *sign* and *sen* on $\underline{Th}$ to functions on $\underline{States}$ by defining $sign((S;\varphi)) = \Theta$ for $S = (\Theta, \Gamma)$ and $sen((S;\varphi)) = sen(sign(S))$.

Definition 3.5 *For all $S_0 \in \underline{Th}$ and $\varphi_0 \in sen(S_0)$, the derivation computed by a theorem proving strategy $\mathcal{C} =< I, \Sigma >$, with inference rules I and search plan $\Sigma =< \zeta, \xi >$, on input $(S_0;\varphi_0)$ is the sequence*

$$(S_0;\varphi_0) \vdash_{\mathcal{C}} (S_1;\varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i;\varphi_i) \vdash_{\mathcal{C}} \ldots$$

where, for all $i \geq 0$, $sign((S_i;\varphi_i)) = \Theta_0$ and, if

- $\zeta((S_i;\varphi_i), i) = f^n$,
- $\xi((S_i;\varphi_i), i, n) = \alpha^n$ *and*
- $\alpha^n_{\Theta_0}(\Gamma_i \cup \{\varphi_i\}) = \bar{x}$, *for* $S_i = (\Theta_0, \Gamma_i)$,

then

$$(S_{i+1};\varphi_{i+1}) = \begin{cases} (S_i;\varphi') & \text{if } \varphi_i \text{ is in } \bar{x},\ \pi_{21}(f^n_{\Theta_0}(\bar{x})) = \{\varphi'\} \text{ and} \\ & \pi_{22}(f^n_{\Theta_0}(\bar{x})) = \{\varphi_i\}, \\ (S_i \cup \pi_{21}(f^n_{\Theta_0}(\bar{x})) - \pi_{22}(f^n_{\Theta_0}(\bar{x})); \varphi_i) & \text{otherwise.} \end{cases}$$

This definition requires that the signature does not change during a derivation: we assume that the initial signature Θ_0 provides the derivation with denumerable sets of predicate symbols, function symbols and variable symbols, so that all the symbols which may be needed during a derivation are available. The core of the definition is the interaction of I and $\Sigma =< \zeta, \xi >$: if ζ chooses the inference rule f^n and ξ chooses the tuple of premises $\bar{x}$, the application of $f^n_{\Theta_0}$ to $\bar{x}$ gives two sets of sentences, $\pi_{21}(f^n_{\Theta_0}(\bar{x}))$ and $\pi_{22}(f^n_{\Theta_0}(\bar{x}))$, with the meaning that the contents of $\pi_{21}(f^n_{\Theta_0}(\bar{x}))$ should replace the contents of $\pi_{22}(f^n_{\Theta_0}(\bar{x}))$. The replacement can be performed either on the target or on the presentation. In the first case, the current target φ_i is among the premises, $\pi_{22}(f^n_{\Theta_0}(\bar{x}))$ contains φ_i and $\pi_{21}(f^n_{\Theta_0}(\bar{x}))$ contains φ', which becomes the new target. Otherwise, the replacement is performed on the presentation. This distinction between *presentation inference steps* and *target inference steps* should not be regarded as too strict. Indeed there are completion procedures, such as the *Linear Completion* procedure for logic programming [7] and the completion procedures for *disproving inductive theorems* [10], that are more conveniently described by allowing inference steps to modify both the target and the presentation [8, 5]. Such steps are legal, as they may be regarded as the composition of a target step and a presentation step.

3.4 Monotonicity and relevance

A theorem proving derivation is required to satisfy two basic *soundness* requirements, that we have called in [8, 5] *monotonicity* and *relevance*. First of all we introduce a functor [4] $th{:}\underline{Th} \to \underline{Set}$, such that for all theories $S = (\Theta, \Gamma)$, $th(S) = \{\psi \mid \Gamma \models_\Theta \psi, \psi \in sen(\Theta)\}$. In other words, $th(S)$ is

[4] The functoriality of $th{:}\underline{Th} \to \underline{Set}$ is an immediate consequence of the definition of theory morphism, under the assumption that the logic is sound and complete.

the set of true theorems of S. *Monotonicity* expresses the requirement that inferences do not add new elements which are not true in the theory:

Definition 3.6 *A derivation*

$(S_0;\varphi_0)\vdash_{\mathcal{C}}(S_1;\varphi_1)\vdash_{\mathcal{C}}\ldots\vdash_{\mathcal{C}}(S_i;\varphi_i)\vdash_{\mathcal{C}}\ldots$

is monotonic *if and only if* $\forall i\geq 0$, $th(S_{i+1})\subseteq th(S_i)$.

Since inference steps also transform the target, we need to require that if the target is modified, the new target is *relevant* to solving the original problem:

Definition 3.7 *A derivation*

$(S_0;\varphi_0)\vdash_{\mathcal{C}}(S_1;\varphi_1)\vdash_{\mathcal{C}}\ldots\vdash_{\mathcal{C}}(S_i;\varphi_i)\vdash_{\mathcal{C}}\ldots$

has the relevance *property if and only if* $\forall i\geq 0$, $\varphi_{i+1}\in Th(S_{i+1})$ *if and only if* $\varphi_i\in Th(S_i)$.

Relevance ensures that a target inference step replaces the target by a new target in such a way that proving the latter is equivalent to proving the former. Monotonicity and relevance indicate the parts of the structure of a theory that theorem proving derivations need to preserve. Therefore we are interested in rephrasing our definitions of monotonicity and relevance as *functoriality properties*:

Theorem 3.1 *A derivation* $\underline{Der}$

$(S_0;\varphi_0)\vdash_{\mathcal{C}}(S_1;\varphi_1)\vdash_{\mathcal{C}}\ldots\vdash_{\mathcal{C}}(S_i;\varphi_i)\vdash_{\mathcal{C}}\ldots$

is monotonic if and only if the projection $pres\colon \underline{States}\to\underline{Th}$, *defined by* $pres((S;\varphi))=S$, *is a functor* $pres\colon \underline{Der}\to\underline{Th}^{op}$.

Proof:

$\Rightarrow$) In order to prove that $pres$ is a functor, we need to show that it associates to every morphism d in $\underline{Der}$ a morphism $pres(d)$ in $\underline{Th}^{op}$:

$$\begin{array}{ccc} (S_j;\varphi_j) & \xrightarrow{\quad d\quad} & (S_{j+k};\varphi_{j+k}) \\ {\scriptstyle pres}\downarrow & & \downarrow{\scriptstyle pres} \\ S_j & \xrightarrow[pres(d)]{} & S_{j+k} \end{array}$$

A morphism $d\colon (S_j;\varphi_j)\to(S_{j+k};\varphi_{j+k})$ in $\underline{Der}$, is a subsequence $(S_j;\varphi_j)\vdash_{\mathcal{C}}^{k}(S_{j+k};\varphi_{j+k})$ of the derivation. Let S_j be (Θ_0,Γ_j) and S_{j+k} be (Θ_0,Γ_{j+k}). By monotonicity, $th(S_{j+k})\subseteq th(S_j)$ holds. Then $\Gamma_{j+k}\subseteq th(S_j)$, i.e. $S_j\models_{\Theta_0}\Gamma_{j+k}$, follows. By completeness of the underlying logic, we can replace satisfaction by entailment and have $S_j\vdash_{\Theta_0}\Gamma_{j+k}$. According to the definition of theory morphism, this is equivalent to say that there is a morphism $pd\colon S_{j+k}\to S_j$ in $\underline{Th}$. For $pres$ to be a functor, the image $pres(d)$ must have the same direction as d, whereas pd has the opposite direction. Thus we define $pres(d)=pd^{op}\colon S_j\to S_{j+k}$ and $pres$ is a functor $pres\colon \underline{Der}\to\underline{Th}^{op}$.

$\Leftarrow$) If *pres* is a functor $pres\colon \underline{Der} \to \underline{Th}^{op}$, then for every morphism $d\colon (S_j;\varphi_j) \to (S_{j+k};\varphi_{j+k})$ in $\underline{Der}$ there is a morphism $pres(d)\colon S_j \to S_{j+k}$ in $\underline{Th}^{op}$. By definition of opposite category, $pres(d)$ is the opposite pd^{op} of some morphism $pd\colon S_{j+k} \to S_j$ in $\underline{Th}$. By definition of theory morphism, this implies that $S_j \vdash_{\Theta_0} \Gamma_{j+k}$. Then we have $S_j \models_{\Theta_0} \Gamma_{j+k}$ by soundness of the underlying logic and finally $th(S_{j+k}) \subseteq th(S_j)$. □

Two remarks apply here. First we observe that *pres* cannot be a functor from $\underline{Der}$ to $\underline{Th}_0$, the subcategory of $\underline{Th}$ that has only axiom-preserving morphisms. The steps of a derivation do not induce axiom-preserving morphisms in general, because a derivation may include contraction steps that delete axioms. It is certainly possible to define a function $closure\colon \underline{States} \to \underline{Th}$ such that $closure((S;\varphi)) = (\Theta, th(S))$, where Θ is the signature of S. Then monotonicity is trivially equivalent to saying that *closure* is a functor $closure\colon \underline{Der} \to \underline{Th}_0^{op}$. However, we prefer to use *pres* and $\underline{Th}$ rather than *closure* and $\underline{Th}_0$, because we want to emphasize the distinction between S and $th(S)$: S is the specific, finite presentation which is actually recorded in the data base during the derivation, whereas $th(S)$ is the unknown and often infinite set of all true theorems in the theory. As we mentioned in Section 2, the restriction to axiom-preserving morphisms may be desirable when theory morphisms are used to describe extensions of theories in the incremental definition of equational programs [11, 13, 19, 24]. On the other hand, it does not seem to be ideal in theorem proving because of contraction steps.

The second remark is that *pres* is a functor from $\underline{Der}$ to $\underline{Th}^{op}$ and not to $\underline{Th}$. This expresses the fact that monotonicity requires that for all $i \geq 0$, $th(S_{i+1}) \subseteq th(S_i)$, but not vice versa. In a theorem proving derivation the theory may become smaller, provided the theorem we are interested in is preserved. Indeed, monotonicity is completed by relevance, to guarantee that $\varphi_{i+1} \in Th(S_{i+1})$ if and only if $\varphi_i \in Th(S_i)$. In theorem proving one is interested only in proving a specific target and therefore it is not necessary to preserve the other theorems as long as the intended theorem is preserved.

For completion-based theorem proving strategies it is worth recalling that completion procedures can also be used for a different purpose than theorem proving. Namely, they can *generate decision procedures* [20]. In this kind of application no target is given and a derivation has the form

$$(S_0;\emptyset) \vdash_{\mathcal{C}} (S_1;\emptyset) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i;\emptyset) \vdash_{\mathcal{C}} \ldots.$$

The derivation proceeds transforming the presentation until it obtains a presentation S such that $\varphi \in th(S)$ is decidable by applying $\mathcal{C}$ itself to $(S;\varphi)$. We have called a presentation with this property a *decision procedure* [8, 5]. For instance, *confluent sets of rewrite rules* are decision procedures for equational theories, since they allow to decide the truth of an equation by rewriting its sides. Knuth-Bendix type completion procedures may be used to generate a confluent set of rewrite rules from a given presentation of an equational theory. For most theories, though, a finite decision procedure does not exist and the derivation which tries to generate it does not halt. We refer to [10] for a survey on these topics.

If the purpose of a derivation is to generate a decision procedure, monotonicity and relevance are replaced by the following *strong monotonicity* property:

Definition 3.8 *A derivation*

$(S_0;\emptyset)\vdash_{\mathcal{C}}(S_1;\emptyset)\vdash_{\mathcal{C}}\ldots\vdash_{\mathcal{C}}(S_i;\emptyset)\vdash_{\mathcal{C}}\ldots$

is strongly monotonic *if and only if* $\forall i\geq 0$, $th(S_{i+1})=th(S_i)$.

Strong monotonicity is necessary to ensure that if the derivation halts, the generated decision procedure is a decision procedure for the input theory.

Corollary 3.1 *A derivation* $\underline{Der}$

$(S_0;\varphi_0)\vdash_{\mathcal{C}}(S_1;\varphi_1)\vdash_{\mathcal{C}}\ldots\vdash_{\mathcal{C}}(S_i;\varphi_i)\vdash_{\mathcal{C}}\ldots$

is strongly monotonic if and only if the projection $pres\colon \underline{States}\rightarrow\underline{Th}$ *is a functor* $pres\colon \underline{Der}\rightarrow\underline{Th}$ *that associates to every morphism in* $\underline{Der}$ *an isomorphism of theories.*

Proof: it follows from Theorem 3.1 and the definition of strong monotonicity. □

We close this section by giving the characterization of *relevance* as a functoriality property. In order to do so we extend the function $\models$ from $\underline{Th}$ to $\underline{States}$. The function $\models\colon \underline{Th}\rightarrow\underline{Set}$ associates to a theory its satisfaction relation. Given a state $(S;\varphi)$ in a theorem proving derivation, we are only interested in knowing whether the target φ is a theorem of S. Therefore, we define $\models$ on $\underline{States}$ in such a way that $\models((S;\varphi))$, for $S=(\Theta,\Gamma)$, tells only whether $\Gamma\models_\Theta\varphi$:

Theorem 3.2 *A derivation* $\underline{Der}$

$(S_0;\varphi_0)\vdash_{\mathcal{C}}(S_1;\varphi_1)\vdash_{\mathcal{C}}\ldots\vdash_{\mathcal{C}}(S_i;\varphi_i)\vdash_{\mathcal{C}}\ldots$

has the relevance property if and only if the function $\models\colon \underline{States}\rightarrow\underline{Set}$ *defined by*

$$\models(((\Theta,\Gamma);\varphi))=\begin{cases}\{(\Gamma,\varphi)\} & \text{if } \Gamma\models_\Theta\varphi\\ \emptyset & \text{otherwise}\end{cases}$$

is a functor $\models\colon \underline{Der}\rightarrow\underline{Set}$ *associating to each morphism in* $\underline{Der}$ *an isomorphism in* $\underline{Set}$.

Proof:

$\Rightarrow$) Let $d\colon (S_j;\varphi_j)\rightarrow(S_{j+k};\varphi_{j+k})$ be a morphism in $\underline{Der}$, where S_j is (Θ_0,Γ_j) and S_{j+k} is (Θ_0,Γ_{j+k}). By relevance, $\varphi_j\in th(S_j)$ if and only if $\varphi_{j+k}\in th(S_{j+k})$. Then

- either both $\varphi_j\in th(S_j)$ and $\varphi_{j+k}\in th(S_{j+k})$ are false and
  $\models((S_j;\varphi_j)) = \models((S_{j+k};\varphi_{j+k}))=\emptyset$
- or both $\varphi_j\in th(S_j)$ and $\varphi_{j+k}\in th(S_{j+k})$ are true,
  $\models((S_j;\varphi_j)) = \{(\Gamma_j,\varphi_j)\}$ and $\models((S_{j+k};\varphi_{j+k})) = \{(\Gamma_{j+k},\varphi_{j+k})\}$.

In the first case the two sets are equal and therefore trivially isomorphic. In the second case the two sets are two singletons and the isomorphism $\models(d)$ simply associates their two elements.

$\Leftarrow$) By the stated functoriality property, if there is a morphism $d\colon (S_j;\varphi_j)\rightarrow(S_{j+k};\varphi_{j+k})$ in $\underline{Der}$ then the two sets $\models((S_j;\varphi_j))$ and $\models((S_{j+k};\varphi_{j+k}))$ are isomorphic. Therefore it cannot be that one is empty whereas the other one is not. By definition of $\models$, it follows that $\varphi_j\in th(S_j)$ if and only if $\varphi_{j+k}\in th(S_{j+k})$. □

3.5 Proof reduction

In this section we give the characterization of a theorem proving derivation as a *process of target-oriented proof reduction*, which is the core of our approach to completion-based theorem proving. First, we assume an underlying proof calculus $\mathcal{PC}$. The choice of $\mathcal{PC}$ depends on the logic and the theorem proving strategy one is interested in. Since in this work we are not considering any specific logic or any specific strategy, we simply assume a proof calculus that describes the structure of the proofs in the theories of $\underline{Th}$. Second, we assume a *proof ordering* $>_p$ on the proofs of $\mathcal{PC}$. A proof ordering is a *well-founded, partial ordering on proofs*. Proof orderings were introduced first in [2, 4] and have represented since then a fundamental element in several works on completion procedures. Proof orderings play a key role in our approach to completion: we refer to [8, 5] for comparisons and further references. Most proof orderings are based on well-founded orderings on terms, atoms and clauses: several examples of such proof orderings may be found in the referenced papers.

Definition 3.9 *Given a proof calculus $\mathcal{PC} = (\underline{Sign}, sen, \vdash, P, Pr, theorem)$, we denote by $Proofs$ the set of all the proofs of $\mathcal{PC}$ and by $\underline{ProofsSet}$ the subcategory of $\underline{Set}$ whose set of objects is $\mathcal{P}(Proofs)$. Given in addition a proof ordering $>_p$, we denote by $\underline{Proofs}$ the category whose set of objects is $Proofs$ and whose morphisms are induced by $>_p$ as follows: for $P, Q \in \underline{Proofs}$, there is a morphism $h\colon P \to Q$ if and only if $P <_p Q$.*

We assume that $\underline{Proofs}$ has a bottom element, *the empty proof*, that we denote by ε. The proof calculus $\mathcal{PC}$ provides us with the natural transformation *theorem* to extract theorems from proofs. We introduce the symbol *true* to denote a trivially true theorem and we define $theorem_S(\varepsilon) = true$ for all theories S. It is worth emphasizing that the introduction of the symbol *true* is not justified only by the need for defining the unique image $theorem_S(\varepsilon)$, it also has a more subtle meaning. For instance in equational logic a theorem $s \simeq s$ is trivially true. However, it is not so in equational theorem proving, since a procedure needs to check that the two sides of the equation are identical before stating that the theorem is true. This is the purpose of an inference rule like *Deletion*,

$$\frac{(E; \hat{s} \simeq \hat{s})}{(E; true)}$$

that appears in completion procedures for equational logic. Similarly, even the most trivial propositional tautology needs to be checked by a tautology checker before it can be declared to be a tautology. Indeed, theorem proving itself consists in showing that a given theorem is valid, i.e. it is a tautology, in the given theory. Thus we need the symbol *true* to represent the state where the validity of the given target has been proved. A state $(S; true)$ is a *successful state*.

The key concept of our proof reduction based approach lies on the observation that proving φ from S is a process of reducing φ to *true* and a proof of φ in S to ε. Since the ultimate goal is to obtain the bottom element ε, at every stage $(S_i; \varphi_i)$ of a theorem proving derivation, we focuse on the *minimal* proofs of φ_i in S_i:

Definition 3.10 *The function $\Pi\colon \underline{States} \to \underline{ProofsSet}$ associates to any state $(S; \varphi)$, the set $\Pi(S; \varphi)$ of the* minimal proofs *of φ in S, according to the ordering $>_p$.*

Clearly, if φ is not in $th(S)$, $\Pi(S;\varphi) = \emptyset$. Note that we have defined $\underline{ProofsSet}$ to be a subcategory of $\underline{Set}$ rather than a subcategory of the category of partially ordered sets. It is because the images of Π are not ordered, since they are sets of minimal, un-orderable proofs. At every stage $(S_i;\varphi_i)$ of a theorem proving derivation, the goal is to prove φ_i from S_i or equivalently to reduce some proof in $\Pi(S_i;\varphi_i)$. The theorem is proved when the derivation reaches a successful state $(S_k;true)$, where $\Pi(S_k;true) = \{\varepsilon\}$. Therefore the inferences in a derivation need to satisfy *proof reduction* requirements. In the following, we recall such requirements and we show that they are equivalent to *functoriality properties* for the function Π. We start by observing that since the purpose of a derivation is to reach ε, an inference should not increase the complexity of proofs:

Definition 3.11 *An inference step $(S;\varphi) \vdash (S';\varphi')$ is* proof-reducing on φ *if for all $P \in \Pi(S,\varphi)$, either $P \in \Pi(S',\varphi')$ or there exists a $Q \in \Pi(S',\varphi')$ such that $P >_p Q$. If the latter holds for some $P \in \Pi(S,\varphi)$, then the step is* strictly proof-reducing.

In other words, every proof which is minimal at a certain stage of the derivation can be replaced only by a smaller proof. A target inference step modifies the target and therefore we require that it is proof-reducing on the target itself:

Definition 3.12 *A target inference step $(S;\varphi) \vdash (S;\varphi')$ is* (strictly) proof-reducing *if it is (strictly) proof-reducing on φ.*

In order to characterize Π as a functor, we use the following:

Definition 3.13 *Given a partially ordered set $(U,<)$ and two subsets $A, B \subseteq U$, a function $f\colon A \to B$ is* reducing *if for all x in A,*

$$f(x) = \begin{cases} x & \text{if } x \in B \\ y & \text{for some } y < x, \text{ otherwise.} \end{cases}$$

Let $\underline{S}$ be a category and $\underline{\mathcal{P}(U)}$ be the subcategory of $\underline{Set}$ whose set of objects is $\mathcal{P}(U)$, i.e. the set of subsets of U. We say that a functor $F\colon \underline{S} \to \underline{\mathcal{P}(U)}$ has reducing images *if for all morphisms h in $\underline{S}$, $F(h)$ is a reducing function.*

We can then show that Π is a functor, which associates to proof-reducing steps in a derivation reducing functions on the corresponding sets of minimal proofs:

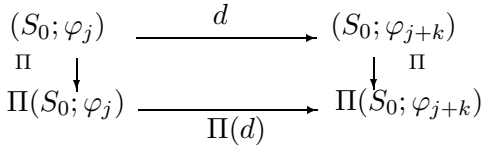
Theorem 3.3 *Let $\underline{Der}$ be a derivation made only of target inference steps. All the steps in $\underline{Der}$ are proof-reducing if and only if the restriction of the function $\Pi\colon \underline{States} \to \underline{ProofsSet}$ to $\underline{Der}$ is a functor $\Pi\colon \underline{Der} \to \underline{ProofsSet}$ which has reducing images.*

Proof:

$\Rightarrow$) If the derivation is proof-reducing, then for all morphisms $d\colon (S_0;\varphi_j) \to (S_0;\varphi_{j+k})$ in $\underline{Der}$ it is possible to define a function $\Pi(d)\colon \Pi(S_0;\varphi_j) \to \Pi(S_0;\varphi_{j+k})$ such that for all P in $\Pi(S_0;\varphi_j)$,

$$\Pi(d)(P) = \begin{cases} P & \text{if } P \in \Pi(S_0;\varphi_j) \\ Q & \text{where } Q <_p P, \text{ otherwise.} \end{cases}$$

The existence of a proof Q such that $Q <_p P$, if $P \notin \Pi(S_0;\varphi_j)$, is guaranteed by proof reduction. Then Π is a functor with reducing images:



$\Leftarrow$) If for all morphisms $d\colon (S_0;\varphi_j) \to (S_0;\varphi_{j+k})$ in $\underline{Der}$, $\Pi(d)\colon \Pi(S_0;\varphi_j) \to \Pi(S_0;\varphi_{j+k})$ is reducing, then for all $P \in \Pi(S_0;\varphi_j)$ either $P \in \Pi(S_0;\varphi_{j+k})$ or there exists a $Q \in \Pi(S_0;\varphi_{j+k})$ such that $P >_p Q$, i.e. all steps in $\underline{Der}$ are proof-reducing. □

For a presentation inference step we need to allow more flexibility, because an inference step on the presentation may not immediately decrease the proof of the target and still be necessary to decrease it eventually. We extend our attention from the given target φ and its proof to a larger set of sentences and proofs. Namely, we define a subfunctor *dom* of *sen*, i.e. a functor $dom\colon \underline{Sign} \to \underline{Set}$, such that for all signatures Θ, $dom(\Theta) \subseteq sen(\Theta)$. If Θ_0 is the signature of a derivation, we call $dom(\Theta_0)$ the *domain* of the derivation. The set $dom(\Theta_0)$ is the set of sentences whose proofs may be reduced by the steps in the derivation. For example, $dom(\Theta_0) = sen(\Theta_0)$ for the Knuth-Bendix completion procedure [20] and $dom(\Theta_0)$ is the subset of ground elements in $sen(\Theta_0)$ for the Unfailing Knuth-Bendix completion procedure [15]. Then, we establish that a presentation step which reduces a proof of the target is proof-reducing, regardless of its effects on the other theorems in the domain. Otherwise, a presentation step is proof-reducing if it does not increase any proof of a theorem in the domain and strictly decreases at least one:

Definition 3.14 *A presentation inference step $(S;\varphi) \vdash (S';\varphi)$, where $sign(S) = sign(S') = \Theta$, is* proof-reducing *on $dom(\Theta)$ if*

1. *either it is strictly proof-reducing on φ*
2. *or*
   - *(a) $\Pi(S,\varphi) = \Pi(S',\varphi)$,*
   - *(b) $\forall \psi \in dom(\Theta)$, $(S;\psi) \vdash (S',\psi)$ is proof-reducing on ψ and*
   - *(c) $\exists \psi \in \mathcal{T}$ such that $(S;\psi) \vdash (S',\psi)$ is strictly proof-reducing on ψ.*

In order to extend Theorem 3.3 to derivations containing both target steps and presentation steps, we introduce *the subcategory induced by a presentation step*:

Definition 3.15 *Let $(S;\varphi) \vdash (S';\varphi)$ be a presentation inference step where $sign(S) = sign(S') = \Theta$ and let $d\colon (S;\varphi) \to (S';\varphi)$ be the corresponding morphism in $\underline{States}$. The* subcategory induced by *d is the subcategory $\underline{Sides}_d \subset \underline{States}$ whose set of objects is*

$$\{(S;\psi) \mid \psi \in dom(\Theta)\} \bigcup \{(S';\psi) \mid \psi \in dom(\Theta)\}$$

and whose morphisms are all the arrows

$$d_\psi\colon (S;\psi) \to (S';\psi) \text{ for } \psi \in dom(\Theta).$$

In other words, $\underline{Sides}_d$ contains all the inference steps that are identical to d except for the choice of a different target in the domain. The purpose of $\underline{Sides}_d$ is to represent the effect of a presentation inference step d on the other theorems in the domain.

Theorem 3.4 *Let* $\underline{Der}$ *be a derivation with signature* Θ_0*. All steps in* $\underline{Der}$ *are proof-reducing on* $dom(\Theta_0)$ *if and only if the function* $\Pi\colon \underline{States} \to \underline{ProofsSet}$ *has the following properties:*

1. *the restriction of* Π *to* $\underline{Der}$ *is a functor* $\Pi\colon \underline{Der} \to \underline{ProofsSet}$ *with reducing images,*
2. *for all one-step morphisms in* $\underline{Der}$*,* $d\colon (S_i;\varphi_i) \to (S_{i+1};\varphi_{i+1})$*, such that* $\varphi_i = \varphi_{i+1}$*, if* $\Pi(d)$ *is the identity function, then*
   (a) *the restriction of* Π *to the subcategory* $\underline{Sides}_d$ *induced by* d *is also a functor* $\Pi\colon \underline{Sides}_d \to \underline{ProofsSet}$ *with reducing images and*
   (b) *there exists at least a morphism* d_ψ *in* $\underline{Sides}_d$*, for* $\psi \in dom(\Theta_0)$*, such that* $\Pi(d_\psi)$ *is not the identity function.*

Proof: the proof of part 1, i.e. the functoriality of Π, is as in Theorem 3.3, keeping into account Conditions 1 and 2a in Definition 3.14. Part 2 mirrors Condition 2 of Definition 3.14. It focuses on those morphisms $d\colon (S_i;\varphi_i) \to (S_{i+1};\varphi_{i+1})$ of $\underline{Der}$ which correspond to presentation inference steps where no proof of the target is reduced. The condition $\varphi_i = \varphi_{i+1}$ says that d is a presentation step. The condition that $\Pi(d)$ is identity is equivalent to $\Pi(S_i;\varphi_i) = \Pi(S_{i+1};\varphi_{i+1})$. These steps are proof-reducing by Condition 2 of Definition 3.14. Condition 2b is equivalent to say that Π is a functor $\Pi\colon \underline{Sides}_d \to \underline{ProofsSet}$:

$$\begin{array}{ccc} (S_i;\psi) & \xrightarrow{\quad d_\psi \quad} & (S_{i+1};\psi) \\ {\scriptstyle\Pi}\downarrow & & \downarrow{\scriptstyle\Pi} \\ \Pi(S_i;\psi) & \xrightarrow[\quad \Pi(d_\psi) \quad]{} & \Pi(S_{i+1};\psi) \end{array}$$

Condition 2c is equivalent to the additional property that $\Pi(d_\psi)$ is not identity for at least one ψ in the domain. □

The notion of proof reduction developed so far applies to presentation inference steps that are either expansion steps or contraction steps which replace some sentences by others. A contraction step which deletes sentences does not modify any minimal proof. In order to characterize these steps, we have introduced a notion of *redundancy*:

Definition 3.16 *A sentence* φ *is* redundant *in* S *on* ψ *if* $\Pi(S,\psi) = \Pi(S \cup \{\varphi\},\psi)$*; it is redundant in* $S = (\Theta,\Gamma)$ *on* $dom(\Theta)$ *if it is redundant on all* $\psi \in dom(\Theta)$*.*

A sentence is redundant in a presentation on a specific target, if adding it to the presentation does not affect any minimal proof of the target. If this holds on the entire domain, the sentence is said to be redundant on the domain. Redundancy can be captured in categorical terms by using the subcategory $\underline{Sides}_d$:

Theorem 3.5 *Let* $d\colon (S \cup \{\varphi\}; \psi) \to (S; \psi)$ *be the morphism corresponding to a presentation step in a derivation* $\underline{Der}$. *Then* φ *is redundant in* S *on* ψ *if and only if* $\Pi(d)$ *is the identity function. It is redundant in* $S = (\Theta, \Gamma)$ *on* $dom(\Theta)$ *if and only if the restriction of* Π *to* $\underline{Sides}_d$ *is a functor* $\Pi\colon \underline{Sides}_d \to \underline{ProofsSet}$ *which associates to every morphism in* $\underline{Sides}_d$ *the identity function.*

Proof: it follows immediately from the definitions of $\underline{Sides}_d$ and redundancy. □

The following notion of reduction covers both proof-reduction and deletion of redundant sentences:

Definition 3.17 *An inference step* $(S; \varphi) \vdash (S'; \varphi')$, *where* $sign(S) = sign(S') = \Theta$, *is* reducing *on* $dom(\Theta)$ *(on* φ*) if either it is proof-reducing on* $dom(\Theta)$ *(on* φ*) or it deletes a sentence which is redundant in* S *on* $dom(\Theta)$ *(on* φ*).*

Our definition of a *completion procedure* summarizes all the properties of theorem proving derivations described so far:

Definition 3.18 *A theorem proving strategy* $\mathcal{C} =< I; \Sigma; dom >$ *is a* completion procedure *if for all* $(S_0; \varphi_0) \in \underline{States}$, *where* $sign(S_0) = \Theta_0$ *and* $\varphi_0 \in dom(\Theta_0)$, *the derivation*

$$(S_0; \varphi_0) \vdash_{\mathcal{C}} (S_1; \varphi_1) \vdash_{\mathcal{C}} \ldots \vdash_{\mathcal{C}} (S_i; \varphi_i) \vdash_{\mathcal{C}} \ldots$$

is monotonic, has the relevance property and $\forall i \geq 0$, *the step* $(S_i; \varphi_i) \vdash_{\mathcal{C}} (S_{i+1}; \varphi_{i+1})$ *is reducing on* $dom(\Theta_0)$ *or on* φ_i.

We cannot conclude our presentation of theorem proving strategies in category theory without mentioning the notion of *completeness* of a strategy. A strategy $\mathcal{C}$ is complete if whenever φ is indeed a theorem of S, $\mathcal{C}$ derives from $(S; \varphi)$ a successful state $(S_k; true)$. In turn, completeness is equivalent to the following two conditions, one for the inference mechanism I and one for the search plan Σ: whenever φ is a theorem of S, the I-tree rooted at $(S; \varphi)$ contains successful nodes and whenever the I-tree rooted at $(S; \varphi)$ contains successful nodes, the search plan Σ is able to reach one such node. We have called these two properties *refutational completeness* of the inference rules and *fairness* of the search plan. The definitions of completeness and fairness in terms of proof reduction have been given in [8, 5, 6].

4 Discussion

We have applied category theory to formalize some fundamental ideas in our approach to theorem proving. Category theory has turned out to be an extremely valuable tool in analyzing concepts and in refining their definitions to a higher level of detail. Such refinement brings the theory closer to the practice of theorem proving. We feel that our analysis of inference rules and search plan contributes to fill the enormous gap between specifying a theorem proving strategy as a set of inference rules and implementing a theorem prover. We can see this contribution both at the level of the inference rules and at the level of the search plan.

Inference rules were traditionally described as *functions*. This is not practical, because it does not cover *contraction* inference rules, which are crucial in keeping the size of the search

space manageable. To solve this problem, the characterization of inference rules as *rules to transform sets* has been proposed and has become a standard for completion-based theorem proving strategies. However, this characterization is non-deterministic, because inference rules are not functions of sets. This non-determinism is useful and elegant at the theoretical level, but it is disappointing in practice, since it contrasts with the obvious experience that once the premises have been selected, the inference rules behave indeed as functions, i.e. deterministically. Since the search plan is the component that selects the premises, this problem can be solved only by defining carefully the inference rules, the search plan and their interaction in determining the relation $\vdash_{\mathcal{C}}$.

For the inference rules, we have given a characterization of inference rules as *natural transformations*. It is more practical than the characterizations of inference rules as functions or as rules to transform sets, because it applies to contraction inference rules, while still maintaining the functional character of inference rules. Also, it makes sense even when the applicability conditions of inference rules fail, as it happens in any real execution of a theorem prover. For the search plan, we have replaced the usually informal definition of search plan by the rigorous definition of the two choice function ζ and ξ. For the relation $\vdash_{\mathcal{C}}$, we have decomposed the notion of inference step $(S_i;\varphi_i)\vdash_{\mathcal{C}}(S_{i+1};\varphi_{i+1})$, usually considered an atomic one, into finer steps. First, ζ is applied to choose an inference rule f^n, second, ξ is applied to choose a tuple of premises $\bar{x}$, third, $f^n_{\Theta_0}$ is applied to $\bar{x}$ to generate the two sets $\pi_{21}(f^n_{\Theta_0}(\bar{x}))$ and $\pi_{22}(f^n_{\Theta_0}(\bar{x}))$ and finally $\pi_{21}(f^n_{\Theta_0}(\bar{x}))$ is replaced by $\pi_{22}(f^n_{\Theta_0}(\bar{x}))$ in the data base. If inference rules are presented as rules to transform sets, this replacement is specified in the inference rules themselves. In our approach, the replacement of $\pi_{21}(f^n_{\Theta_0}(\bar{x}))$ by $\pi_{22}(f^n_{\Theta_0}(\bar{x}))$ belongs to the definition of $\vdash_{\mathcal{C}}$ rather than to the definition of inference rule. Therefore, we no longer need to define inference rules on sets and we can restore their functional nature by defining them as natural transformation.

In the second part of the paper we have shown that the properties of *monotonicity*, *relevance*, *proof reduction* and *redundancy* are functoriality properties. In particular, proof reduction and redundancy consist in the existence of a functor Π from the category $\underline{Der}$ of the states of a derivation to the underlying category $\underline{ProofsSet}$. In order to obtain this result, we have introduced the elegant notion of the *subcategory* $\underline{Sides}_d$ *induced by a presentation inference step* d. This subcategory allows us to treat a single, concrete inference step $(S_i;\varphi_i)\vdash_{\mathcal{C}}(S_{i+1};\varphi_{i+1})$ as possibly infinitely many inferences $(S_i;\psi)\vdash(S_{i+1};\psi)$, one for every element ψ in the domain of the completion procedure. Therefore, we can analyze the effect of the derivation of S_{i+1} from S_i with respect to all the potential targets in the domain. This is necessary for proof reduction of inferences on the presentation, because a presentation inference step is not guaranteed to reduce the proof of the given target and therefore its proof reducing effect must be analyzed on $\underline{Sides}_d$.

To summarize, the categories involved with a theorem proving derivation are $\underline{States}$, with its subcategories such as $\underline{Der}$ and $\underline{Sides}_d$, and, at the proofs level, $\underline{Proofs}$ and $\underline{ProofsSet}$. Our study of inference rules, search plans and derivation relations $\vdash_{\mathcal{C}}$ ultimately characterizes the structure of $\underline{States}$. The only structure imposed on $\underline{Proofs}$ is the proof ordering $>_p$. The connection between $\underline{States}$ and $\underline{ProofsSet}$, and therefore $\underline{Proofs}$, is given by the functor Π. It remains open the problem of how to choose the proof calculus $\mathcal{PC}$ whose proofs are the objects of $\underline{Proofs}$. More precisely, one may investigate whether there exists a relation between the inference rules of

the strategy and the proof calculus which provides the underlying proofs. Another, challenging direction for further research is to start from our definition of search plan to propose an entire theory of search plans for theorem proving.

We conclude this discussion with a comparison with related work. Our framework for completion-based theorem proving is closely related to the research of many other authors. Since a comparison of approaches to completion procedures is beyond the scope of this paper, we refer to [8, 5, 6] for complete references and comparisons in the area of completion procedures and we limit ourselves to some comparison with [24].

We are indebted to [24] for several notions reported in this paper, such as those of entailment system, institution and especially proof calculus. Even more importantly, the reading of [24] has inspired us the idea of using category theory in the first place. This paper continues the work in [24] in the direction of theorem proving. Indeed, most of the key issues in theorem proving do not appear yet in [24]. The procedural part of theorem proving is absent, since there is no notion of search plan and therefore no notion of derivation generated by a search plan. Also, the notion of inference seems to be limited to expansion inference rules only. A tentative characterization of Unfailing Knuth-Bendix completion as a functor $KB: \underline{Th}_0 \to \underline{Th}_0$ is sketched in Example 19 of [24]. This attempt is not satisfactory because it does not take into account contraction inference rules such as simplification. Given an equational theory (Θ, E), KB is not guaranteed to preserve the axioms in E, because they can be simplified. Thus, KB cannot preserve the axiom-preserving morphisms $H: (\Theta, E) \to (\Theta', E')$ of $\underline{Th}_0$ and it cannot be a functor of theories. A theorem proving strategy cannot be required to preserve theory morphisms: the properties of monotonicity, relevance and proof reduction establish all that is preserved by a theorem proving derivation. As we have already pointed out in the previous sections, some elements of the categorical approach in [24] seem to be motivated by the applications to declarative programming languages and abstract data types in the tradition of [11, 13, 19], rather than to theorem proving, and indeed they are not ideal for theorem proving: see for instance the definition of theory morphisms and the use of $\underline{Th}_0$. However, the generality of the approach and the power of category theory allow one to choose how to apply the elements of the categorical framework to programming languages or to theorem proving according to his/her motivation.

5 Acknowledgements

We would like to thank José Meseguer for the fruitful discussions we had on the categorical approach to logic.

References

[1] S.Anantharaman and J.Hsiang, Automated Proofs of the Moufang Identities in Alternative Rings, *Journal of Automated Reasoning*, Vol. 6, No. 1, 76–109, 1990.

[2] L.Bachmair, N.Dershowitz and J.Hsiang, Orderings for Equational Proofs, in *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science*, 346–357, Cambridge, Massachussets, June 1986.

[3] L.Bachmair and H.Ganzinger, On Restrictions of Ordered Paramodulation with Simplification, in M.E.Stickel (ed.), *Proceedings of the Tenth International Conference on Automated Deduction*, Kaiserslautern, Germany, July 1990, Springer Verlag, Lecture Notes in Artificial Intelligence 449, 427–441, 1990.

[4] L.Bachmair and N.Dershowitz, Equational inference, canonical proofs and proof orderings, *Journal of the ACM*, to appear.

[5] M.P.Bonacina, Sulla dimostrazione di teoremi per completamento, (in Italian, English version: On completion theorem proving), Thesis of "Dottorato di Ricerca", Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Milano, Italy, January 1991.

[6] M.P.Bonacina and J.Hsiang, On fairness of completion-based theorem proving strategies, in R.V.Book (ed.), *Proceedings of the Fourth International Conference on Rewriting Techniques and Applications*, Como, Italy, April 1991, Springer Verlag, Lecture Notes in Computer Science 488, 348–360, 1991.

[7] M.P.Bonacina and J.Hsiang, On Rewrite Programs: Semantics and Relationship with Prolog, *Journal of Logic Programming*, to appear.

[8] M.P.Bonacina and J.Hsiang, Completion procedures as Semidecision procedures, in M.Okada, S.Kaplan (eds.), *Proceedings of the Second International Workshop on Conditional and Typed Rewriting Systems*, Montréal, Canada, June 1990, Springer Verlag, Springer Verlag, Lecture Notes in Computer Science 516, 206–232, 1991.

[9] C.L.Chang and R.C.Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York, 1973.

[10] N.Dershowitz and J.-P.Jouannaud, Rewrite Systems, Chapter 15 of Volume B of *Handbook of Theoretical Computer Science*, North-Holland, 1989.

[11] K.Futatsugi, J.A.Goguen, J.P.Jouannaud and J.Meseguer, Principles of OBJ2, in B.Reid (ed.) *Proceedings of the Twelveth Annual ACM Symposium on Principles of Programming Languages*, New Orleans, Louisiana, 1985.

[12] J.A.Goguen and R.Burstall, Introducing institutions, in E.Clarke, D.Kozen (eds.) *Logic of Programs*, Springer Verlag, Lecture Notes in Computer Science 164, 221–256, 1984.

[13] J.A.Goguen and J.Meseguer, Equality, types, modules and (why not?) generics for logic programming, *Journal of Logic Programming*, Vol. 1, No. 2, 179–210, 1984.

[14] W.A.Howard, The formulae-as-types notion of construction, unpublished manuscript, 1969, reprinted in J.P.Seldin and J.R.Hindley (eds.), *To H.B.Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, 1980.

[15] J.Hsiang and M.Rusinowitch, On word problems in equational theories, in Th.Ottman (ed.), *Proceedings of the Fourteenth International Conference on Automata, Languages and Programming*, Karlsruhe, Germany, July 1987, Springer Verlag, Lecture Notes in Computer Science 267, 54–71, 1987.

[16] G.Huet, A Complete Proof of Correctness of the Knuth-Bendix Completion Algorithm, *Journal of Computer and System Sciences*, Vol. 23, 11–21, 1981.

[17] G.Huet, Deduction and Computation, INRIA, Rapport de Recherche No. 513, April 1986.

[18] D.Kapur and H.Zhang, An Overview of Rewrite Rule Laboratory (RRL), in N.Dershowitz (ed.), *Proceedings of the Third International Conference on Rewriting Techniques and Applications*, Chapel Hill, North Carolina, April 1989, Springer Verlag, Lecture Notes in Computer Science 355, 559–563, 1989.

[19] C.Kirchner, H.Kirchner and J.Meseguer, Operational semantics of OBJ3, in *Proceedings of the Nineth International Conference on Automata, Languages and Programming*, Springer Verlag, Lecture Notes in Computer Science 241, 1988.

[20] D.E.Knuth and P.B.Bendix, Simple Word Problems in Universal Algebras, in J.Leech (ed.), *Proceedings of the Conference on Computational Problems in Abstract Algebras*, Oxford, England, 1967, Pergamon Press, Oxford, 263–298, 1970.

[21] F.W.Lawvere, Functorial semantics of algebraic theories, in *Proceedings of the National Academy of Sciences*, 50, 1963, Summary of Ph.D. Thesis, Columbia University.

[22] S.MacLane, *Categories for the working mathematician*, Springer Verlag, 1971.

[23] E.G.Manes, *Algebraic Theories*, Springer Verlag, 1976.

[24] J. Meseguer, General Logics, in H.-D. Ebbinghaus et al. (eds.) *Proceedings of Logic Colloquium '87*, Granada, Spain, 1987, Elsevier Science Publishers (North Holland), 1989.

[25] J.A.Robinson, A Machine Oriented Logic Based on the Resolution Principle, *Journal of the ACM*, Vol. 12, No. 1, 23–41, 1965.

[26] M.Rusinowitch, Theorem Proving with Resolution and Superposition, *Journal of Symbolic Computation*, Vol. 11, No. 1 & 2, 21–50, January/February 1991.

[27] M.E.Szabo, *Algebra of Proofs*, North Holland, 1978.