Safety Signatures for First-order Languages and Their Applications

Arnon Avron
aa@math.tau.ac.il
School of Computer Science
Tel Aviv University
Tel Aviv 69978, Israel

1 Introduction

In several areas of Mathematical Logic and Computer Science one would ideally like to use the set $Form(L)$ of all formulas of some first-order language L for some goal, but this cannot be done safely. In such a case it is necessary to select a subset of $Form(L)$ that can safely be used. Three main examples of this phenomenon are:

- The main principle of naive set theory is the *comprehension* schema:

  $$\exists Z(\forall x.x \in Z \Leftrightarrow A)$$

  where A is a formula in which Z is not free (but may contain other parameters). Ideally, every formula A should be used in this schema. Unfortunately, it is well known that this would lead to paradoxes. What the various axiomatic set theories do is to replace the general comprehension schema by "safer" versions. Thus most of the axioms of ZF, the most famous axiomatic set theory, are just particular instances of the comprehension schema. Historically, the guiding line behind the choice of these instances has been the "limitation of size doctrine" ([8, 10]).

Hendricks et al. (eds.):
First-Order Logic Revisited
Logos Verlag Berlin (2004), 37–58

However, the criterion provided by this doctrine is not constructive, so ZF uses some constructive substitutes to select formulas which seem to meet it. These principles are usually explained and justified ([18]) on *semantic* ground, using certain general ontological assumptions. (some of which, like the "cofinality principle", may be debatable).

- A main goal of computability theory is to characterize the *decidable* relations. Now the most straightforward method of defining relations is by using formulas of an appropriate formal language L (like the language of Peano Arithmetics PA in the case of arithmetical relations). However, usually not every formula of L defines a decidable relation. Hence a major problem here is: what are the "safe" formulas which do? A strongly related problem of crucial importance for proof theory and the foundations of Mathematics (especially Gödel theorems) is: what formulas of L binumerate relations within a given theory T? [1] Again it is well known that in the case of PA no constructive general solution can be given for either problem. Therefore some constructively defined classes of "safe" formulas, broad enough for the various applications, have been selected in its language. Two major examples are the class of primitive recursive (p.r.) formulas ([9, 14]) and the class of bounded formulas ([21]).

- A query language for a database ([24, 2]) is an ordinary first-order language with equality, the signature of which includes predicate symbols for the database relations. Ideally, every formula ψ of a query language can serve as a query. If ψ is closed then the answer to the query is either "yes" or "no". If ψ has free variables then the answer to ψ is the set of tuples which satisfy it in the intended structure. However, an answer to a query should be finite and computable, even if the intended domain is infinite. Hence only "safe" formulas, the answers to which always have these properties, should be used as queries. Unfortunately, it is again undecidable which formulas are "safe". Therefore all commercial query languages (like SQL) allow to use as queries only formulas from some syntactically defined class of safe formulas.

In all these examples the same pattern repeats: a certain undecidable class of f.o. formulas, originally characterized by some semantic criterion, is

[1] If T is r.e. then such a relation is necessarily decidable.

singled out for some fundamental application. Then an effective, syntactically defined subclass that can serve as a sufficient substitute is found. In what follows we show that despite the different purposes and intuitions, the principles which have been used in all these areas in order to secure safety are similar (although they have independently been developed), and are directly based on the role of the first-order logical constants. By merging them we will be able to develop a unified, purely logical framework for dealing with "safety". The key feature of this framework is the use of a generalized concept of a f.o. signature. The idea is that a generalized signature for a language can contain more than just the arity of the possible interpretations of the primitive symbols of the language. It can contain e.g. also information about the size and/or the computability of their *intended* interpretations (reducing by this the class of allowed models).

Three concrete applications of our framework are:

- In set theory it provides a new, concise presentation (and in our opinion, a new understanding) of ZF. This presentation is based on purely *syntactic* criteria concerning the role the f.o. connectives and quantifiers have in defining legitimate new sets.
- In Computability Theory it provides a general framework for analyzing relative computability of both extensional and intensional relations and functions, on arbitrary (or at least countable) f.o. structures.
- In database theory it provides a simple syntactical notion of safety, which allows to use properties of relations and functions which do not belong to the database scheme. This notion is adequate not only for conventional databases, but also for databases in which there is only a partial access to some of the relations (like in the world wide web).

2 The General Framework

In the examples above *two* different factors were involved in questions of "safety": *size* (of the class of tuples which satisfy a given formula) and *computability* (of this class). Now of the three example above only safety in databases is connected with both. It is reasonable therefore to take database theory as our starting point. Another reason for this choice is that many explicit proposals of decidable, syntactically defined classes of safe formulas

have been made in this theory. Examples are: "range separable formulas" ([5]), "range restricted formulas" ([16]), "evaluable formulas" ([6]), "allowed formulas" ([23]), and "range safe formulas" ([2]) [2]. The simplest among them (and the closer to what has actually been implemented) is perhaps the following class $SS(D)$ ("syntactically safe" formulas for a database scheme D) from [24] (originally designed for languages with no function symbols) [3]:

1. $p_i(t_1, \ldots, t_{n_i}) \in SS(D)$ in case p_i (of arity n_i) is in D, and each t_i is either a variable or a constant.
2. $x = c$ and $c = x$ are in $SS(D)$ (where x is a variable and c is a constant).
3. $A \vee B \in SS(D)$ if $A \in SS(D)$, $B \in SS(D)$, and they have the same free variables.
4. $\exists x A \in SS(D)$ if $A \in SS(D)$.
5. If $A = A_1 \wedge A_2 \wedge \ldots \wedge A_k$ then $A \in SS(D)$ if both of the following conditions are met:
   (a) For each $1 \leq i \leq k$, either A_i is atomic, or A_i is in $\mathcal{SS}(\mathcal{D})$, or A_i is a negation of a formula of either type.
   (b) Every free variable x of A is limited in A. This means that there exists $1 \leq i \leq k$ s.t. x is free in A_i, and either $A_i \in SS(D)$, or $A_i \in \{x = y, y = x\}$, where y is already limited in A.

There is one clause in this definition which is somewhat strange: the last one, which treats conjunction. The reason why this clause does not simply tell us (like in the case of disjunction) when a conjunction of *two* formulas is in $SS(D)$, is the desire to take into account the fact that once the value of y (say) is known, the formula $x = y$ becomes safe. One of the crucial observations on which our framework is based is that in order to find

[2] In our opinion there is a mistake in the definition of the last one. According to this definition a formula like $x = c \wedge (\neg \exists y (y \neq x))$ is range safe, although it is clearly not domain independent (despite a theorem to the converse which is proved in [2]). We believe that the source of the problem is a mistake in the way negation is handled there, and that it should be corrected along the lines this is done below.

[3] What we present below is both a generalization and a simplification of Ullman's original definition. It should be noted that Ullman's main concern is the stronger property of domain-independence that we discuss in subsection 3.2.

a common generalization of the various notions described above one should indeed consider *partial* safety. In other words: safety should be viewed as a *relation* between formulas and (finite) sets of variables rather than as a *property* of formulas [4]. Since two different issues are involved here (size and computability), this observation leads to the following two generalizations of $SS(D)$ (where $Fv(E)$ denotes the set of free variables of E):

Definition 1. *A relation $\succ$ between formulas A of a first-order language L in a signature σ and subsets of $Fv(A)$ is a* size-safety (s-safety) relation *if it satisfies the following conditions:*

(1) $A \succ \emptyset$ *for all* A.

(2) *If* $x \notin Fv(t)$ *then* $x = t \succ \{x\}$ *and* $t = x \succ \{x\}$.

(3) *If* $A \succ X$ *and* $B \succ X$ *then* $A \vee B \succ X$.

(4) *If* $y \notin X$ *and* $A \succ X \cup \{y\}$ *then* $\exists y A \succ X$.

(5) *If* $A \succ X$, $B \succ Y$, *and* $X \cap Fv(B) = \emptyset$ *or* $Y \cap Fv(A) = \emptyset$, *then* $A \wedge B \succ X \cup Y$.

Definition 2. *A* c-safety relation *between formulas of a language L and finite sets of variables is defined like in Definition 1, except that condition* (1) *is replaced by the following weaker conditions:*

(1a) $p(x_1, \ldots, x_n) \succ \emptyset$ *in case p is a primitive n-ary predicate symbol of σ.*

(1b) *If* $A \succ \emptyset$ *then* $\neg A \succ \emptyset$.

Our standard interpretation of s-safety is that $A(x_1, \ldots, x_n, y_1, \ldots, y_k)$ is s-safe w.r.t. $\{x_1, \ldots, x_n\}$ in a given structure S, iff either $n = 0$, or for any assignment $c_1 \ldots, c_k$ of values from S for $y_1, \ldots, y_k$, the set of tuples $\langle d_1, \ldots, d_n \rangle$, which together with $c_1 \ldots, c_k$ satisfy A in S, is finite. It is easy to prove that this interpretation indeed defines an s-safety relation (see section 4). To get an intuition concerning definition 2, think of $A(x_1, \ldots, x_n, y_1, \ldots, y_k)$ as a query with parameters $y_1, \ldots, y_k$, and interpret "$A(x_1, \ldots, x_n, y_1, \ldots, y_k) \succ \{x_1, \ldots, x_n\}$" as: "The answer to the query A

[4] This may be compared with Tarski's definition of the validity *property* of formulas in structures via the satisfaction *relation* between formulas and assignments in structures.

is finite and effectively computable for any values of the parameters". *Intuitively* (see again section 4), this defines a c-safety relation, provided that the interpretations of the primitive function symbols of σ are all effectively computable, and the interpretations of the primitive predicate symbols of σ are all effectively decidable (This cannot be rigorously proved, though, since we do not have a precise definition of an "effectively computable answer to a query". We shall return to this point in section 4).

Note 1. For the present framework it is preferable to take $\wedge, \vee, \neg$ and $\exists$ as primitive, and $\rightarrow$ and $\forall$ as defined in terms of them. Moreover: we take $\neg(A \rightarrow B)$ as an abbreviation for $A \wedge \neg B$, and $\forall x_1 \ldots x_k A$ as an abbreviation for $\neg\exists x_1 \ldots x_k \neg A$. This entails the following important property of "bounded quantification": *If $\succ$ is a c-safety relation, $A \succ \{x_1, \ldots, x_n\}$, and $B \succ \emptyset$, then $\exists x_1 \ldots x_n . A \wedge B \succ \emptyset$ and $\forall x_1 \ldots x_n . A \rightarrow B \succ \emptyset$.*

Note 2. In all examples we know, whenever a safety relation $\succ$ is defined by some semantic property, it obeys the following principle: If $A \succ X$, B is logically equivalent to A, and $Fv(A) = Fv(B)$, then $B \succ X$. s-safety is usually closed under the even stronger principle: If $A \succ X$ where $X = \{x_1 \ldots x_k\}$, $\exists y_1 \ldots y_n \forall x_1 \ldots x_k (A \leftrightarrow B)$ is logically valid, and $\{y_1 \ldots y_n\} \cap Fv(B) = \emptyset$, then $B \succ X$. The reason we have not included these principles in the definitions above is that we want to be able to define *decidable* safety relations that can serve in applications as good substitutes for the undecidable, semantically defined ones. Still, for convenience one may incorporate into the definitions useful special cases of these properties, like standard boolean identities, and the following facts concerning substitutions (which follow from the equivalence between $A(t/y)$ and $\exists z \exists y (z = t \wedge y = z \wedge A)$, where $z \notin Fv(t) \cup Fv(A)$):

- If $y \notin X$, $A \succ X \cup \{y\}$, $Y \subseteq Fv(t)$, $Y \cap Fv(A) \subseteq \{y\}$, and $z = t \succ Y$ for $z \notin Fv(t) \cup Fv(A)$, then $A(t/y)$ is equivalent to some B s.t. $B \succ X \cup Y$.
- If $y \notin X$, $A \succ X$, and $X \cap Fv(t) = \emptyset$, then $A(t/y)$ is equivalent to some B such that $B \succ X$.

The straightforward way of defining a reasonable syntactical substitute for a given semantical safety relation is to use definitions 1 or 2 as a basis for an inductive definition. In most cases this amounts to specifying what atomic formulas (other than those of the form $x = t$ or $t = x$) are taken as safe w.r.t. what variables. For this it is usually best to use the following generalization of the notion of a signature for a language (see the introduction for the motivation):

Definition 3. *A* safety-signature *is a pair (σ, F), where σ is an ordinary first-order signature and F is a function which assigns to every n-ary symbol s from σ (other than equality) a nonempty subset of $\mathcal{P}(\{1, \ldots, n\})$, so that if $I \in F(s)$ and $J \subset X$ then $J \in F(s)$.*

Definition 4. *Let (σ, F) be a safety-signature. $\succ_{(\sigma,F)}$ ($\succ^s_{(\sigma,F)}$) is the (inductively defined) minimal c-safety (s-safety) relation $\succ$ (in the first order language induced by σ) which satisfies the following conditions:*

1. *If p is an n-ary predicate symbol of σ; $x_1, \ldots, x_n$ are n distinct variables, and $\{i_1, \ldots, i_k\}$ is in $F(p)$, then $p(x_1, \ldots, x_n) \succ \{x_{i_1}, \ldots, x_{i_k}\}$.*

2. *If f is an n-ary function symbol of σ; $y, x_1, \ldots, x_n$ are $n+1$ distinct variables, and $\{i_1, \ldots, i_k\} \in F(f)$, then $y = f(x_1, \ldots, x_n) \succ \{x_{i_1}, \ldots, x_{i_k}\}$.*

Proposition 1. *Both $\succ_{(\sigma,F)}$ and $\succ^s_{(\sigma,F)}$ satisfy the following conditions:*

1. *If $A \succ X$ then $X \subseteq Fv(A)$.*

2. *If $A \succ X$ and $Z \subseteq X$, then $A \succ Z$.*

In the coming sections we shall see several applications of these notions.

3 Safety in Databases

From a logical point of view, a database of scheme $D = \{p_1, \ldots, p_n\}$ is just a given set of *finite* interpretations of $p_1, \ldots, p_n$. As noted in the introduction, a corresponding query language is an ordinary first-order language with equality, the signature of which contains D. A query is "safe" if its answer is finite and computable for all interpretations in which $p_1, \ldots, p_n$ are finite (and given), while the interpretations of all other predicate symbols are decidable, and function symbols (if any) are interpreted by computable functions. Our framework leads in this case to the following syntactical counterpart:

Definition 5. *Let D be a subset of σ such that each $q \in D$ is a predicate symbol of arity k_q.*

1. *The safety signature (σ, F_D) corresponding to σ and D is defined by:*

$$F_D(q) = \begin{cases} \mathcal{P}(\{1, \ldots, k_q\}) & \text{if } q \in D \\ \{\emptyset\} & \text{otherwise} \end{cases}$$

2. A formula A is called (σ, F_D)-safe if $A \succ_{(\sigma,F_D)} Fv(A)$.

It is easy to show inductively (using the intuitive meaning of c-safety given in the previous section) that each (σ, F_D)-safe formula can safely be used as a query for any database of scheme D, and that in a language without function symbols every formula in Ullman's $SS(D)$ is logically equivalent to some (σ, F_D)-safe formula[5]. It is important to note that our notion can in fact be used even if function symbols *are* allowed (provided that their intended interpretations are computable). Moreover: it is very easy to extend it *in a natural way* in order to be able to take into account safety properties that other functions and relations (not in the database scheme) might have in the intended domain(s). Suppose for example that "$<$" is in the language, and that its intended interpretation is the usual order relation of $\mathcal{N}$, or the substring relation on strings. In such a case the set $\{x \mid x < a\}$ is finite and computable for every a in the intended domain. This fact can be exploited by taking $F_D(<) = \{\emptyset, \{1\}\}$. An example of a query that will become then (σ, F_D)-safe is $\exists x \exists y(p_1(x, y) \wedge z < x + y)$.

3.1 An Application: Querying the Web

An interesting application of c-safety has implicitly been made in [15]. There the web is modeled as an ordinary database augmented with three more special relations: [6] $N(id, title, \ldots)$, $L(source, destination, \ldots)$, $C(node, value)$. The intuitive interpretations of these relations are the following:

- The relation N contains the Web objects which are identified by a Uniform Resource Locator (URL). *id* represents the URL and is a key.
- The relation L holds between two nodes *source* and *destination* if there is a hypertext link from the first to the second.
- The meaning of the relation C is that the string which is represented by its second argument occurs within the body of the document in the URL which is represented by its first argument.

The question investigated in [15] is: what queries should be taken as safe, if we assume that what is practically possible in the case of N and L is to

[5] If we strengthen $\succ_{(\sigma,F_D)}$ as suggested in Note 2, then $SS(D)$ becomes a proper subset of the set of (σ, F_D)-safe formulas.

[6] For simplicity, we ignore other special relations which are used there.

list all their tuples which *correspond to a given first argument*, while C is only assumed to be decidable. A special "Safe Web Calculus" based on these assumptions is then introduced. It is not difficult to see that the notion of safety which is defined by this calculus is in fact equivalent to (σ_{web}, F_{web})-c-safety in our sense, where $\{L, N, C\} \subseteq \sigma_{web}$, and F is defined like in ordinary databases, except that $F(L) = \mathcal{P}(\{2, \ldots, m\})$ (where m is the arity of L), $F(N) = \mathcal{P}(\{2, \ldots, k\})$ (where k is the arity of N), and $F(C) = \{\emptyset\}$.

3.2 Domain-independence

Another property of queries to databases which is considered to be crucial ([12, 24, 2]) is *domain-independence* (d.i.). Its definition (in the case of ordinary databases) is the following:

Definition 6. [7] *Let σ be a signature which includes $\overrightarrow{P} = \{P_1, \ldots, P_n\}$, and optionally constants and other predicate symbols (but no function symbols). A query in σ is called* $\overrightarrow{P}-$d.i. ($\overrightarrow{P}-$domain-independent) *if it has the same answer in S_1 and S_2, whenever S_1 is a substructure of S_2, and the interpretations of $\{P_1, \ldots, P_n\}$ in S_1 and S_2 are identical.*

Our next goal is to show that domain-independence can also successfully be handled within our framework (using safety-signatures as our main tool). For this we generalize first the ordinary notion of an extension of a structure (for a signature σ) to structures for safety-signatures:

Definition 7. *Let (σ, F) be a safety-signature with no function symbols (other than constants). Let S_1 and S_2 be two structures for σ s.t. $S_1 \subseteq S_2$. S_2 is called a* $(\sigma, F)-$extension *of S_1 if the following condition is satisfied: If $p \in \sigma$ is of arity n, $I \in F(p)$, and $a_1, \ldots, a_n$ are elements of S_2 such that $a_i \in S_1$ in case $i \notin I$, then $S_2 \models p(a_1, \ldots, a_n)$ iff $a_i \in S_1$ for* all *i and $S_1 \models p(a_1, \ldots, a_n)$.*

Note 3. Since $\emptyset \in F(P)$ for all $P \in \sigma$, S_1 is a substructure of S_2 whenever S_2 is a $(\sigma, F)-$extension of S_1.

Examples:

[7] This is a slight generalization of the definition in [Su98], which in turn is a generalization of the usual one ([Ki88,Ul88]). The latter applies only to free Herbrand structures which are generated by adding to σ some new set of constants.

1. Let $\sigma = \{=, <\}$, and let $F(<) = \{\emptyset, \{1\}\}$. Obviously, a structure for σ is a $(\sigma, F)-$extension of its initial segments (and only of them).

2. Let $\sigma_{AS} = \{=, \in\}$ and let $F_{AS}(\in) = \{\emptyset, \{1\}\}$. In this case the "universe" is a $(\sigma_{AS}, F_{AS})-$ extension of what are known in Set Theory as "transitive sets".

Definition 8. *Let (σ, F) be as in Definition 7. A formula A of σ is called* $(\sigma, F)-$d.i. w.r.t. X *($A \succ^{di}_{(\sigma,F)} X$) if whenever S_2 is a $(\sigma, F)-$extension of S_1, and A^* resulted from A by substituting values from S_1 for the free variables of A that are not in X, then the sets of tuples which satisfy A^* in S_1 and in S_2 are identical.* [8] *A formula A of σ is called* $(\sigma, F)-$d.i. *if $A \succ^{di}_{(\sigma,F)} Fv(A)$.*

It can easily be proved that $\succ^{di}_{(\sigma,F)}$ is a c-safety relation. It follows that if $A \succ_{(\sigma,F)} X$ (Definition 4) then $A \succ^{di}_{(\sigma,F)} X$. In particular: if $A \succ_{(\sigma,F)} Fv(A)$ then A is $(\sigma, F)-$d.i. It is also obvious that if D is a database scheme in a signature σ, then a formula A is $(\sigma, F_D)-$d.i. iff it is d.i. for D in the usual sense (of Definition 6). Since already in this case the notion of d.i. is undecidable ([7]), the class of $(\sigma, F_D)-$safe formulas is again a good syntactical substitute.

Note 4. Despite the close connection, safety of queries (in the sense of being "effectively finite") and domain independence of them are in general independent notions. Thus every logically valid sentence (or a logical contradiction) is d.i. w.r.t. $\emptyset$, but not necessarily safe w.r.t. $\emptyset$. On the other hand $\forall x.x = c$ is (effectively) safe w.r.t. $\emptyset$ (it is true if the domain is a singleton, false otherwise), but not d.i. w.r.t. $\emptyset$ (for precisely the same reason).

4 Safety in Computability Theory and in Metamathematics

We have followed up to now the intuition that a query is safe iff its answer is finite and computable, but we have not defined what "computable" means. The intuitive notion we have in mind is *not* identical to that investigated in Classical Computability Theory (CCT). CCT provides answers to the questions "what *extensional* relations are decidable?" and "what *extensional*

[8] A^* is a formula only in a generalized sense, but the intention should be clear.

functions are computable?". Queries define however *intensional* relations, and CCT provides only necessary conditions (like finiteness and decidability) for the computability of such relations and functions. Thus every extensional finite relation is "computable" according to CCT, but in reality we might not know how to actually compute an intensional relation even if its extension is finite. However, it is precisely the question of computing intensional relations and functions that we encounter in practice. A particularly delicate question in this context is what is the interpretation of "The answer to the query φ is computable" in case φ is a sentence (i.e.: a query with a "yes" or "no" answer). This is a question to which CCT provides no clue, but is important for database theory, and is also the main point of difference between s-safety and c-safety. It should be noted that this question is crucial for constructive computability theory too. Thus Bridges explicitly gives in [4] the following example of a function f from $\mathcal{N}$ to $\mathcal{N}$, which is "computable" according to CCT, but not constructively so: For all n, $f(n)$ is 1 if Goldbach conjecture is true, 0 otherwise.

I am not aware of any precise definition in the literature (or an analogue of Church's thesis) for the concepts of "computable intensional function" and "computable intensional relation". What can therefore be done at present is to provide obvious properties of these notions and use them for developing a useful corresponding general theory.[9] Where should we start? Well, the usual approach to CCT is to characterize first the class of computable functions, and then to define the class of decidable relations as those relations whose characteristic functions are computable. However, in modern mathematics functions are defined as a special type of relations, and so it seems more reasonable from its point of view to go the other way around. This is certainly more natural when a theory of intensional computability is sought, since even intensional relations and functions should be defined in some formal language — and what first-order languages (which are the most natural languages to use for this purpose) directly define are *relations*. [10] Now for *intensional* relations the general framework suggested here does provide a general relative computability theory (though we do not claim it to be the ultimate one). In fact it provides general sufficient criteria for a parametric formula in some first-order language L to define a (finite) computable (intensional) relation

[9] A similar procedure is suggested in [17] as a possible approach for trying to *prove* Church's thesis in the extensional case.

[10] This has indeed been the approach of [21], where the class of "bounded formulas" is used for defining the basic notions of CCT. We will return to this class below.

in all structures for L in which the interpretations of the primitive predicates and functions of L have certain computational properties:

Definition 9. *Let L be a first-order language with equality, and let S be a structure for L.*

1. *A formula A of L which is not a sentence is* $S-$effective *if the number of tuples which satisfy it in S (for some order of its free variables) is finite, and they can effectively be listed. A sentence is* $S-$effective *if its truth-value in S can effectively be computed.*

2. *A formula A of L is* $S-$effective *w.r.t a finite subset X of its free variables if for any substitution of concrete (syntactic names for) elements from S for the free variables of A which are not in X, we get an $S-$effective formula (in the extended language $L(S)$ of S ([17])).*

Note 5. It should again be emphasized that this definition assumes the intuitive notion of "effective computability" which is left here *undefined* (and probably cannot be defined!). Note also that $S-$effectiveness of a formula A w.r.t. $\emptyset$ means that the relation on S which A defines is effectively *decidable*.

Definition 10. *Let (σ, F) be a safety-signature. A structure S for σ is* appropriate *for (σ, F) if it satisfies the following two conditions:*

- *If p is an n-ary predicate symbol of σ; $x_1, \ldots, x_n$ are n distinct variables, and $\{i_1, \ldots, i_k\} \in F(p)$, then $p(x_1, \ldots, x_n)$ is $S-$effective w.r.t. $\{x_{i_1}, \ldots, x_{i_k}\}$.*

- *If f is an n-ary function symbol of σ; $y, x_1, \ldots, x_n$ are $n+1$ distinct variables, and $\{i_1, \ldots, i_k\} \in F(f)$, then $y = f(x_1, \ldots, x_n)$ is $S-$effective w.r.t. $\{y\}$ and w.r.t. $\{x_{i_1}, \ldots, x_{i_k}\}$.*

Theorem 1. *If S is appropriate for (σ, F), and $C \succ_{(\sigma,F)} X$, then the formula C is $S-$effective w.r.t. X.*

Intuitive Proof: By induction on the structure of C. If C is of the form $x = t$ or $t = x$ then the claim is proved by induction on the structure of t (using the assumption that every formula of the form $y = f(x_1 \ldots, x_n)$ is $S-$effective w.r.t. $\{y\}$). The other safety conditions concerning atomic formulas directly follow from the fact that S is appropriate for (σ, F). The induction step splits into four cases. We do here the case where $C = A \wedge B$,

$A \succ X$, $B \succ Y$, and $Y \cap Fv(A) = \emptyset$. To simplify notation, assume that $Fv(A) = \{x, z\}$, $Fv(B) = Fv(C) = \{x, y, z\}$, $X = \{x\}$, $Y = \{y\}$. Let c be an element of S. To compute $\{< x, y > \in S^2 \mid C(c/z)\}$, compute first the set $Z(c) = \{x \in S \mid A(c/z)\}$ ($Z(c)$ is finite and effectively computable by our assumptions on A). Then for each $d \in Z(c)$ compute the set $W(c, d) = \{y \in S \mid B(d/x, c/z)\}$ ($W(c, d)$ is finite and effectively computable by our assumptions on B). The set $\{< x, y > \in S^2 \mid C(c/z)\}$ is the finite union of the sets $\{d\} \times W(c, d)$ ($d \in Z(c)$).

We present now two famous applications of Theorem 1 from the literature on Metamathematics. This is another area in which one needs (especially for the proof of Gödel's second incompleteness theorem) a class of effective *intensional* relations, defined by formulas in some particular f.o. languages.

Bounded safety: Let $\sigma_b = \{=, 0, S, +, \times, <\}$. Define $F_b(<) = \{\emptyset, \{1\}\}$ and $F(f) = \{\emptyset\}$ for any function symbol f. Let $\succ_b = \succ_{(\sigma_b, F_b)}$.

Primitive recursive safety: Let $\sigma_{PA} = \{=, 0, S, +, \times\}$, and let σ_{PR} be σ_{PA} augmented with function symbols for every primitive recursive (p.r.) function. Define $F_{PR}(S) = \mathcal{P}(\{1\})$, $F_{PR}(+) = \mathcal{P}(\{1, 2\})$, and $F_{PR}(f) = \{\emptyset\}$ otherwise. Let $\succ_{PR} = \succ_{(\sigma_{PR}, F_{PR})}$.

Definition 11. *A formula A is* $\succ_b$-effective *if $A \succ_b \emptyset$. A formula A is* $\succ_{PR}$-effective *if $A \succ_{PR} \emptyset$.*

It is easy to see that the structure $\mathcal{N}$ of the natural numbers (with the standard interpretations of the symbols in σ_b and σ_{PR}) is appropriate for (σ_b, F_b) and (σ_{PR}, F_{PR}). Thus $\{1\} \in F_b(<)$ means that $x < y \succ_b \{x\}$ (where x and y are two different variables). This in turn means that given any $n \in \mathcal{N}$, there is only a finite number of k's such that $k < n$, and they can effectively be listed. Similarly, the fact that $\{1, 2\} \in F_{PR}(+)$ means that $y = x_1 + x_2 \succ_{PR} \{x_1, x_2\}$. This in turn means that given any $n \in \mathcal{N}$, there is only a finite number of pairs $\langle k_1, k_2 \rangle$ such that $n = k_1 + k_2$, and they can effectively be listed. Both claims are obvious. It follows from Theorem 1 that if A is $\succ_b$-*effective* or $\succ_{PR}$-effective then A defines a decidable relation on the structure $\mathcal{N}$.

By letting $x < y$ abbreviate in σ_{PA} $\exists w \exists x \exists z. y = x + w \wedge w = S(z)$, we get that $x < y \succ_{PR} \{x\}$ as well. Hence $\succ_b$ is contained in $\succ_{PR}$. Now Note 1

implies that the set of $\succ_b$-effective formulas is closed under bounded quantification (the same applies to the set of $\succ_{PR}$-effective formulas). It follows that this class is an extension of the class of bounded formulas ([21]). It is not difficult to see, in fact, that the two classes define the same class of (extensional) relations on $\mathcal{N}$. Similarly, the class of $\succ_{PR}$-effective formulas [11] is equivalent to Feferman's class of primitive recursive formulas ([9, 14]), which is of crucial importance in Metamathematics. Actually, this importance is easily seen to be due to the following connections between $\succ_{PR}$ and provability:

Theorem 2. *Let Q^* and PA^* be the extensions in σ_{PR} of Q and PA (respectively) with the defining axioms of every p.r. function*[12]*. Let A be a formula in σ_{PR} such that $A \succ_{PR} \{x_1, \ldots, x_k\}$.*

1. *Assume that A' is a closed substitution instance of A. If A' is true (in $\mathcal{N}$) then $\vdash_{Q^*} A'$, while if it false then $\vdash_{Q^*} \neg A'$.*

2. *Let $F_v(A) - X = \{y_1, \ldots, y_n\}$. If $k > 0$ then there exists a p.r. function f_A s.t. $\vdash_{Q^*} A \rightarrow \big(x_1 < f_A(y_1, \ldots, y_n) \wedge \ldots \wedge x_k < f_A(y_1, \ldots, y_n)\big)$.* [13]

3. *$A \rightarrow Pr_{PA^*}(\ulcorner A(\dot{x}_1, \ldots, \dot{x}_n, \dot{y}_1, \ldots, \dot{y}_n) \urcorner)$ is provable in PA^*.*[14]

Proof: The proof of (3) is similar to the usual proofs of such results in the literature, using the fact that by (2), if $A \succ_{PR} \{x\}$ then the existential quantification of A on x can be replaced by bounded quantification on x. The proofs of (1) and (2) are done simultaneously, using an induction on the construction of $\succ_{PR}$. We do here the case $A = \exists y B$ as an example. To simplify notation we assume that $F_v(B) = \{x, y, z\}$, $B \succ_{PR} \{x, y\}$ (and so indeed $A \succ_{PR} \{x\}$). By induction hypothesis there is a p.r. function f_B s.t.:

$$\begin{array}{ll} \text{(I)} & \vdash_{Q^*} B \rightarrow x < f_B(z) \\ \text{(II)} & \vdash_{Q^*} B \rightarrow y < f_B(z) \end{array}$$

[11] More accurately: the class of formulas which result from the $\succ_{PR}$-effective ones by substituting everywhere $\delta_f(x_1, \ldots, x_n, y)$ for $y = f(x_1, \ldots, x_n)$, where $\delta_f(x_1, \ldots, x_n, y)$ is the standard formula in σ_{PA} which binumerates f in PA ([14]).

[12] It is well known ([14]) that Q^* and PA^* are conservative extensions of Q and PA.

[13] For convenience, we use here the same symbol for f_A in both σ_{PR} and in our metalanguage.

[14] The notation here follows that of [20].

It immediately follows from (I) that $\vdash_{Q^*} A \rightarrow x < f_B(z)$, proving (2) for A (take $f_A = f_B$). To prove (1), assume that A' is a closed instance of A. Then there are numbers n and k such that A' is equivalent in Q^* to $\exists y B(\overline{n}, y, \overline{k})$. If A' is true in $\mathcal{N}$ there is a number m such that $B(\overline{n}, \overline{m}, \overline{k})$ is true, and so (by induction hypothesis) provable in Q^*. This entails that A' is provable in Q^*. If, on the other hand, A' is false then $B(\overline{n}, \overline{i}, \overline{k})$ is false for every $i < f_B(k)$. Hence, by induction hypothesis, we have for every $i < f_B(k)$:

$$\text{(III)} \quad \vdash_{Q^*} \neg B(\overline{n}, \overline{i}, \overline{k})$$

On the other hand (II) above implies that

$$\text{(IV)} \quad \vdash_{Q^*} B(\overline{n}, y, \overline{k}) \rightarrow y < \overline{f_B(k)}.$$

Together, (III) and (IV) imply that $\vdash_{Q^*} \neg \exists y B(\overline{n}, y, \overline{k})$, and so $\vdash_{Q^*} \neg A'$.

Note 6. Let $\sigma_p = \{=, 1, S, +, \times\}$. Define $F_p(S) = P(\{1\})$, $F_p(+) = F_p(\times) = P(\{1, 2\})$. Let $\succ_p = \succ_{(\sigma_p, F_p)}$. It is possible to show that the class of $\succ_p$-effective relations is exactly the class of arithmetical relations that can be decided in polynomial time (some other complexity classes can similarly be characterized).

The class of $\succ_b$-effective relations is a proper subclass of the class of $\succ_{PR}$-effective relations. Our general definition allows us, accordingly, to capture different notions of "effectiveness". None of them can exactly capture the intuitive notion of "constructive effectiveness" (by a diagonalization argument). What *does* seem to be robust is the notion of *semi-decidable* relations:

Definition 12. *A formula is called* $\succ_b$-r.e. *if it is of the form* $\exists x A$, *where* A *is a* $\succ_b$*-effective formula. A relation is called* $\succ_b$-r.e. *if it is defined by a* $\succ_b$-r.e. *formula. The classes of* $\succ_p$*-r.e. and* $\succ_{PR}$*-r.e. formulas and relations are defined similarly.*

Proposition 2. *The classes of* $\succ_b$*-r.e. relations,* $\succ_p$*-r.e. relations, and* $\succ_{PR}$*-r.e. relations are all identical (to the usual class of r.e. relations).*

Note 7. From the last proposition it is clear that a possible formulation of Church's Thesis is that a relation R on $\mathcal{N}$ is semi-decidable iff it is definable by a formula of the form $\exists x A$, where A is either a $\succ_b$-effective or $\succ_{PR}$-effective (R is of course decidable iff both R and its complement are semi-decidable). It follows that CCT for extensional relations does not really involve principles that go beyond those that are suggested in our framework for c-safety of f.o. formulas and for intensional computability.

Note 8. Unlike in databases, the interest in CCT and in Metamathematics has been in safety of a formula A w.r.t. $\emptyset$ (rather than w.r.t. $Fv(A)$).

5 Safety in Set Theory

As we have noted in the introduction, what the various axiomatic set theories actually provide are syntactic criteria for classes of formulas which may be assumed to be "safe" for applying the naive comprehension schema. This is evident, e.g., in the case of Quine's NF, in which the notion of a "stratified formula" is used. However, we show in this section that it is true also in the case of ZF, the most famous (and universally accepted) among the axiomatic set theories. Most of the axioms of ZF are indeed just particular instances of the comprehension schema. As noted in the introduction, the guiding line behind the choice of these instances has been the semantic "limitation of size doctrine" ([8, 10]). According to this criterion, only collections which are not "too big" can be accepted as sets. Here "not too big" is an intuitive notion (which encompasses quite large infinite sets). With this intuitive notion in mind, a formula A of set theory may be called "size-safe" ("s-safe") w.r.t x, if $\{x \mid A\}$ determines a collection which is "not too big". The comprehension axioms of ZF lists all the cases which are universally recognized to be "s-safe" in this sense. Now in databases "size-safe" means "finite". In set theory it means something completely different (like "not equipotent with the collection of all sets"). We show now that the principles which have been used in these two disciplines in order to secure limitation of size are nevertheless the same, although they have been developed independently. This, we believe, provides strong support to the claim (recently made, e.g., by H. Friedman) that with the exception of the infinity axiom, all the other comprehension axioms of ZF are obtained by an extrapolation from the finite case to the general one. It also leads to new presentations of ZF which are based on purely *syntactic* considerations — in contrast to the usual semantical justifications (as presented, e.g., in [18]).

To achieve these goals we should use of course an appropriate s-safety relation rather than a c-safety relation (computability is not an issue here!). To be able to present one, we need (because of the Powerset axiom) to assume that $=$, $\in$ and $\subseteq$ are all primitive symbols of ZF [15]. Finally, in order to get a real insight into the nature of ZF, we follow its presentation in [18]

[15] Hence the usual definition of $\subseteq$ in terms of $\in$ should be taken as one of the axioms.

(where the axioms of ZF are explained and an attempt is made to justify them on a semantic ground). Practically this means that we use for ZF a dynamic language which has the means to introduce new symbols for definable functions. In other words: once $\exists! y A(x_1, \ldots, x_n, y)$ is proved, it is possible to introduce a new function symbol F_A together with the axiom $\forall x_1, \ldots, x_n(A(x_1, \ldots, x_n, F_A(x_1, \ldots, x_n)))$ (see [17], section 4.6). Officially we assume that the language includes all these function symbols from the start, and that every instance of the following schema is an axiom:

$$\forall x_1, \ldots, x_n(\exists! y A(x_1, \ldots, x_n, y) \rightarrow A(x_1, \ldots, x_n, F_A(x_1, \ldots, x_n)))$$

By this we obtain a conservative extension of ZF which we denote by ZF^f. We next introduce a corresponding safety-signature:

Definition 13. *Let $\sigma_{ZF} = \{=, \in, \subseteq\}$, and let σ_{ZF^f} be σ_{ZF} augmented with all the function symbols of ZF^f. Define $F_{ZF}(\in) = F_{ZF}(\subseteq) = \{\emptyset, \{1\}\}$. Extend F_{ZF} to σ_{ZF^f} by letting $F_{ZF^f}(g) = \{\emptyset\}$ for every function symbol g.*

In the rest of this section "safety" will mean $(\sigma_{ZF^f}, F_{ZF^f})$-s-safety. For the reader convenience, we recall that this relation is defined here as follows:

(A) Every formula is safe w.r.t $\emptyset$.

(B) If $x \notin Fv(t)$ then $x = t$, $t = x$, $x \in t$, and $x \subseteq t$ are safe w.r.t $\{x\}$.

(C) If A and B are both safe w.r.t. X, then so is $A \vee B$.

(D) If A is safe w.r.t. X, B is safe w.r.t. Y, and $X \cap Fv(B) = \emptyset$ or $Y \cap Fv(A) = \emptyset$, then $A \wedge B$ is safe w.r.t. $X \cup Y$.

(E) If $y \notin X$ and A is safe w.r.t. $X \cup \{y\}$, then $\exists y A$ is safe w.r.t. X.

Theorem 3. *The standard comprehension axioms of ZF^f (Pairing, Powerset, Union, Separation, and Replacement) can be replaced by the following single* safe comprehension schema (SCn^f):

$$\exists Z \forall x. x \in Z \Leftrightarrow A, \text{ where } A \text{ is safe w.r.t. } \{x\}, \text{ and } Z \notin Fv(A).$$

Proof: The comprehension axioms of ZF^f are all instances of SCn^f:

Pairing: $\exists Z \forall x. x \in Z \Leftrightarrow (x = y \vee x = z)$

The formula used here is safe w.r.t. $\{x\}$ by (B) and (C).

Powerset: $\exists Z \forall x. x \in Z \Leftrightarrow (x \subseteq y)$

The formula used here is safe w.r.t. $\{x\}$ by (B). [16]

Union: $\exists Z \forall x. x \in Z \Leftrightarrow (\exists v. x \in v \wedge v \in y)$

The formula used here is safe w.r.t. $\{x\}$ by (B), (D), and (E).

Separation: $\exists Z \forall x. x \in Z \Leftrightarrow (x \in y \wedge B)$

The formulas used in this schema are safe w.r.t. $\{x\}$ by (A), (B), (D).

Replacement: $\exists Z \forall x. x \in Z \Leftrightarrow (\exists v. v \in y \wedge x = t)$

Here x, y, v are 3 distinct variables, and t is a term in which x does not occur free. The formulas used in this scheme are safe w.r.t. $\{x\}$ by (B), (D), and (E).

For the converse, let C be a formula which is safe w.r.t. $\{x_1, \ldots, x_n\}$ (where $\{x_1, \ldots, x_n\}$ are all free in C). Define $Set_{x_1,\ldots,x_n}C$ to be $\neg C \vee C$ in case $n = 0$, and $\exists Z(\forall x_1 \ldots \forall x_n(< x_1, \ldots, x_n >\in Z \Leftrightarrow C))$ in case $n > 0$. We show by induction on the structure of C that $Set_{x_1,\ldots,x_n}C$ is a theorem of ZF^f (the principle we want to show is obtained from this result as the particular case in which $n = 1$). Most of the cases are straightforward. We again do here the case of conjunction (which is the most complicated) as an example. To simplify notation, assume again that $Fv(A) = \{x, z\}$, $Fv(B) = \{x, y, z\}$, A is safe w.r.t. $\{x\}$, B is safe w.r.t. $\{y\}$ (and so $A \wedge B$ is safe w.r.t. $\{x, y\}$). By induction hypothesis, $\vdash_{ZF^f} Set_x A$, and $\vdash_{ZF^f} Set_y B$. We show that $\vdash_{ZF^f} Set_{x,y}(A \wedge B)$. Now the assumptions imply that there are sets Z(z) and W(x,z) such that:

$$\vdash_{ZF^f} x \in Z(z) \Leftrightarrow A \qquad \vdash_{ZF^f} y \in W(x, z) \Leftrightarrow B$$

It follows that $\{< x, y >| A \wedge B\} = \bigcup_{x \in Z(z)} \{< x, y >| y \in W(x, z)\}$. Hence $Set_{x,y}(A \wedge B)$ follows from the axiom of replacement and the axiom of union.

[16] Note that the validity of the Powerset axiom is enforced here by taking $\subseteq$ as primitive, and letting $\{1\} \in F_{ZF}(\subseteq)$.

Example: The existence of the Cartesian product of two sets, U and V, is due to the safety w.r.t. $\{x\}$ of $\exists a \exists b.a \in U \wedge b \in V \wedge x =< a, b >$ (One should here justify first the use of the term $< a, b >$. This is easy.

Already Theorem 3 suffices for supporting the claim that the construction principles behind ZF are nothing more than standard syntactical principles concerning the first-order logical constants which are normally used to secure *finiteness*[17]. However, if one insists on using just standard first-order formulas of the signature σ_{ZF}, then replacement causes a problem. The reason is that unlike the other comprehension axioms of ZF, its official formulation in ZF has the form of a conditional. A possible solution to this problem is to translate into the language of ZF the conditions which define safety, and take these translations as our axioms. For this we assume first that a binary function symbol $<,>$ for forming ordered pairs is added to σ_{ZF}, together with an axiom which corresponds to its usual definition.[18]

Theorem 4. *The various comprehension axioms of ZF can be replaced (in the language with $<,>$) by the following axioms:*

(A) $Set_{x_1,\ldots,x_n}A \Rightarrow Set_{z_1,\ldots,z_n}A$ *where* $z_1, \ldots, z_n$ *is a permutation of* $x_1, \ldots, x_n$.

(B1) $set_x x = y$

(B2) $set_x x \subseteq y$

(C) $(Set_{x_1,\ldots,x_n}A \wedge Set_{x_1,\ldots,x_n}B) \Rightarrow Set_{x_1,\ldots,x_n}A \vee B$

(D) $(Set_{x_1,\ldots,x_n}A \wedge (\forall x_1 \ldots \forall x_n Set_{y_1,\ldots,y_m}B)) \Rightarrow Set_{x_1,\ldots,x_n,y_1,\ldots,y_m}A \wedge B$ *in case* $\{y_1, \ldots, y_m\} \cap Fv(A) = \emptyset$.

(E) $Set_{x_1,\ldots,x_n,y}A \Rightarrow Set_{x_1,\ldots,x_n}\exists yA$

Proof: We shall show here how to prove replacement from the new set of axioms, leaving the rest for the reader. For this it is convenient to use the version of replacement given in [17]. In the present notation, this version can be formulated as follows:

$$\forall y Set_x A \Rightarrow Set_x \exists y.y \in w \wedge A$$

[17] This basing of the axioms of ZF on a syntactically defined notion of "smallness" is similar in spirit to recent works on category-theoretic models of ZF (see [11, 19]).

[18] Alternatively, one may add a function symbol for forming unordered pairs.

So assume $\forall y Set_x A$. Since $Set_y y \in w$ is logically valid, this assumption implies $(Set_y y \in w) \wedge (\forall y Set_x A)$. By axiom (D) we can infer therefore $Set_{x,y} y \in w \wedge A$. From this $Set_x \exists y.y \in w \wedge A$ follows by axiom (E).

Note 9. Still another approach, in which an extra case is added to the notion of safety in ZF (and the explicit use of the abstraction operation is allowed) is outlined in [1]. In that paper also the infinity axiom is presented as a particular case of the safe comprehension schema, but for this one needs to use an extension of first-order logic with a transitive closure operation.

Note 10. We have seen that in database theory the interest is in safety of a formula w.r.t. to its whole set of free variables. Then we saw that in computability theory and in metamathematics the interest is in safety of a formula w.r.t. the empty set of variables. Now we see that in set theory, in contrast, the main interest is in safety of a formula w.r.t. exactly *one* of its free variables! These differences might be the reason why the connection between the three cases has been hidden for so long.

5.1 Absoluteness

It is interesting to note that also an analogue of the concept of domain-independence from database theory (see subsection 3.2) has independently been introduced in the literature on set theory. This is the notion of *absoluteness*, which is crucial for independence proofs (see, e.g., [13]). Indeed, it is easy to see that a formula is $(\sigma_{AS}, F_{AS})-$d.i. with respect to $\emptyset$ (see the second example after Definition 7) iff it is absolute according to the literature on set theory. Again we see here an interesting difference between what has been taken to be important in database theory and in set theory: while in database theory the main interest is in d.i. of a formula w.r.t. its set of free variables, in set theory the interest has been in d.i. of a formula w.r.t. $\emptyset$!.

6 Bibliography

[1] A. Avron, *Transitive Closure and the mechanization of Mathematics*, in F. Kamareddine (ed.), **Thirty Five Years of Automating Mathematics**, Kluwer Academic Publishers, 2003, 149-171.

[2] S. Abiteboul, R. Hull and V. Vianu, **Foundations of Databases**, Addison-Wesley, 1995.

[3] J. Barwise, **Admissible Sets and Structures**, Perspectives in Mathematical Logic, Springer-Verlag, 1975.

[4] D. S. Bridges, *Constructive Truth in Practice*, in H. G. Dales and G. Oliveri (eds.), **Truth in Mathematics**, Calendron Press, 1998, 53-69.

[5] E.F. Codd, *Relational Completeness of Database Sublanguages*, in R. Rustin (ed.), **Database Systems**, Prentice-Hall (1972), pp. 65-98.

[6] R. Demolombe, *Syntactical characterization of a subset of domain independent formulas*. Technical report, ONERA-CERT, Toulouse (1982).

[7] R.A. Di Paola, *The recursive unsolvability of the decision problem for the class of definite formulas*. J. ACM 16, 324-327, 1969.

[8] A. Fraenkel, Y. Bar-Hillel, and A. Levy, **Foundations of Set Theory**, North-Holland, Amsterdam, 1973.

[9] S. Feferman, *Arithmetization of metamathematics in a general setting*, Fundamenta Mathematica 49 (1960) 35-92.

[10] M. Hallett **Cantorian Set Theory and Limitation of Size**, Clarendon Press, 1984.

[11] A. Joyal, and I. Moerdijk, **Algebraic Set Theory**, Cambridge University Press, 1995.

[12] M. Kiffer, *On Safety Domain independence and capturability of database queries*, Proc. International Conference on database and knowledge bases, 405-414, Jerusalem 1988.

[13] K. Kunen, **Set Theory: an Introduction to Independence Proofs**, North-Holland, Amsterdam, 1980.

[14] P. Lindström, **Aspects of Incompleteness**, Lectures Notes in Logic 10, Springer, 1997.

[15] A. O. Mendelzon, and T. Milo, *Formal Models of Web Queries*, Proceedings of the Sixteenth ACM Symposium on Principles of Database Systems, 134-143, 1997.

[16] J.M. Nicolas, *Logic for Improving Integrity Checking in Relational Data Bases*, Acta Informatica, Vol. 18 (1982), pp. 227-253.

[17] J. R. Shoenfield, **Mathematical Logic**, Addison-Wesley, 1967.

[18] J. R. Shoenfield, *Axioms of Set Theory*, in: **Handbook Of Mathematical Logic** (J. Barwise, Ed.), North-Holland, Amsterdam, 1977.

[19] A. Simpson, *Axioms for the Category of Classes*, Proceedings of the Fourteenth IEEE Symposium on Logic in Computer Science, 77-85.

[20] C. Smorynski, *The Incompleteness Theorems*, in **Handbook of Mathematical Logic** (J. Barwise, Ed.), North-Holland, Amsterdam, 1977.

[21] R. M. Smullyan, **The Incompleteness Theorems**, Oxford University Press, 1992.

[22] D. Suciu, *Domain-independent queries on databases with external functions*, Theoretical Computer Science 190, 279-315, 1998.

[23] R.W. Topor, *Domain independence formulas and databases*. Theoretical Computer Science 52 (1987) 281-306.

[24] J.D. Ullman, **Principles of database and knowledge-base systems**, Computer Science Press, 1988