MAX-PLANCK-INSTITUT
FÜR
INFORMATIK

Reflection in Logical Systems

Seán Matthews

MPI–I–92–250 November 1992

INFORMATIK

Im Stadtwald
W 6600 Saarbrücken
Germany

Author's Address

Max-Planck-Institut für Informatik,
Im Stadtwald, W-6600 Saarbrücken, Germany.
`sean@mpi-sb.mpg.de`

Publication Notes

Also in the proceedings of IMSA'92: International workshop on new models for software architecture, reflection and metalevel architecture, Tokyo, November 1992.

Acknowledgements

I would like to thank Alan Smaill and Alex Simpson for very helpful criticism of these, and other ideas, and Fausto Giunchiglia, who encouraged me to become interested in this area in the first place.

Abstract

I develop some of the theory of self-referential systems. I present the necessary semantic ideas, and combine this with work in proof theory, on the necessary properties of a proof predicate, to develop practical theories for reasoning about such systems. I propose to use this to exploit the idea of a reflection principle as a systematic way to extend such self-referential theories safely. I also try to relate theoretical points to practical concerns.

Keywords

reflection, semantics, proof theory.

§ 1 Introduction

The theory of self-referential systems presents special problems. In this paper I present a possible practical approach to the semantics and proof theory of such systems.

In § 2 I develop a semantics for self-referential systems. This is done by taking the semantics of a non-self-referential system and extending it with the facilities that characterise reflection. I also consider some of the properties of such a system.

In § 3 I look at ways that the proof theory of an ordinary (i.e., not self-referential) system can be safely extended to include the idea of self refence developed in the previous section. In the second part of this section I also look at the idea of a 'reflection principle'; this is a way that the proof theory can be extended with new axioms not provable in the original theory, but that on examining the semantics can be easily seen to be true. Such reflection principles are doubly useful: first, they allow us to reason about the system in ways that are are intuitively reasonable but *not* permissible in the initial theory; second, they extend the theory so that it is able to prove useful new facts about the system, including facts that have nothing to do with self-reference.

In § 4 I extend the discussion in § 3 by considering ways of building *practical* proof theories with self-referential properties and how they can be extended with reflection principles. The previous section considered only what was possible in theory, not what could be done in practice, and the obvious way to extend a proof theory to be self referential is so labour intensive and error prone to implement that it is not remotely practical. Instead I propose that work by Löb on the necessary behaviour of such self-referential proof facilities should be used. It turns out that Löb's work is not quite sufficient for present purposes, so a suitable extension is described.

In the last couple of sections I briefly look at how well my proposed approach actually works, present some conclusions and, finally, discuss limitations of current work and how it might be developed.

§ 2 The idea of reflection

The question of what it means to talk about reflection in a formal theory has to be addressed as a prelude to everything else. An informal definition is that it is the ability of a system to introspect. This is not, however, a useful definition, thus a formal definition is presented here.

§ **2.1 Meaning** We can think of a system $\mathscr{S}$ as consisting of the pair $(\mathcal{L}, \mathfrak{A})$, where $\mathcal{L}$ is a language used to talk about $\mathscr{S}$ and $\mathfrak{A}$ is a structure that gives the meaning of terms and formulae in $\mathcal{L}$[1]. Then we can use $\mathfrak{A}$ to interpret ground terms t in the language $\mathcal{L}$ as referring to particular objects; this is denoted $\mathfrak{A}[\![t]\!]$. For instance in the system of arithmetic $\mathscr{A} \equiv (\mathcal{L}_{\mathscr{A}}, \mathfrak{A}_{\mathscr{A}})$, $1+2$ and $0+3$ might be ground terms of $\mathcal{L}_{\mathscr{A}}$, in which case $\mathfrak{A}_{\mathscr{A}}[\![1+2]\!]$ and $\mathfrak{A}_{\mathscr{A}}[\![0+3]\!]$ would denote same value of *three*. In the same way, a closed formula A of $\mathcal{L}$ will denote either *true* or *false*; for example $\mathfrak{A}_{\mathscr{A}}[\![1+2=0+3]\!]$ is *true*, and $\mathfrak{A}_{\mathscr{A}}[\![\exists x(x=x+1)]\!]$ is *false*. This definition of the meaning of formulae can be extended to apply also to open formulae by saying that such a formula A is *true* (*false*) iff for arbitrary interpretations of the variables as objects in $\mathfrak{A}$ it is *true* (*false*); i.e., an open sentence can be regarded as equivalent to its universal closure. Notice that it is not so easy to give a definition of the meaning of non-ground terms (if x is a variable, then what should $\mathfrak{A}_{\mathscr{A}}[\![x+1]\!]$ mean?), since there is no equivalent, for terms, to universal closure.

A further reasonable property of systems, that will be assumed in future, is that there is no 'junk'; i.e., it is possible using $\mathcal{L}$ to talk about every thing in $\mathfrak{A}$[2].

§ **2.2 Encoding** Now I consider what facilities are needed for a system to refer to itself.

I use the language $\mathcal{L}$ externally to query $\mathscr{S}$. If $\mathscr{S}$ is to be able to query itself it needs a similar language, and the obvious available candidate for this is $\mathcal{L}$. So there has to be some equivalent of the formulae and terms of $\mathcal{L}$ among the objects in $\mathfrak{A}$. Assuming that this is the case, then since there is no junk in $\mathfrak{A}$ there is some way of referring those objects using $\mathcal{L}$; i.e., there are ground terms in $\mathcal{L}$ which mean those objects. At the moment I do not want to extend the language $\mathcal{L}$, so I will define a notational *abbreviation* for this: for any formula A or term t, some ground term denoting the corresponding object in $\mathfrak{A}$ is written in abbreviated form as $\ulcorner A \urcorner$ or $\ulcorner t \urcorner$[3].

[1]In mathematical logic, for instance, we would call $\mathscr{S}$ a model; however we can equally think of it as a computer system, especially one where the programming language is closely allied to the operating system (e.g., a Lisp Machine), then $\mathcal{L}$ is the programming language/system language, and $\mathfrak{A}$ is the software (the interpreter/compiler, the operating system etc.) and hardware together.

[2]To pursue the idea of a system as a computer further, this is equivalent to the reasonable demand that the language of the operating system is sufficient so that a user is able to inquire after any part of the system.

[3]Since to do this I need to be able to map the set of all formulae and terms of $\mathcal{L}$ onto

§ **2.3 Canonical forms** The encoding facility just described is not quite enough by itself. Encoding means that the language of $\mathcal{L}$ is somehow mapped onto the objects in $\mathfrak{A}$; however, it is also necessary to be able to go in the other direction; i.e., some way of mapping objects in $\mathfrak{A}$ onto terms in $\mathcal{L}$.

This is slightly tricky: since there is no junk in $\mathfrak{A}$ there is at least one term in $\mathcal{L}$ referring to each object in it. Unfortunately there may be more than one way; e.g., $1+2$ and $0+3$, for example, both refer to *three.* The problem is that, while $\mathfrak{A}_{\mathscr{A}}[\![1+2=0+3]\!]$ is *true*, $\mathfrak{A}_{\mathscr{A}}[\![\ulcorner 1+2 \urcorner = \ulcorner 0+3 \urcorner]\!]$ is *false* — though since the two are different strings of symbols, this is exactly what we would expect. What we need is some distinguished way of describing in $\mathcal{L}$ any particular object in $\mathfrak{A}$. We call this canonical form.

Definition 1 (Canonical Form) *A Canonical form for objects is is defined by a function c from objects in $\mathfrak{A}$ to terms in $\mathcal{L}$ such that, for any object $\mathfrak{a}$, c returns an object $\mathfrak{a}'$ which encodes the term with the meaning $\mathfrak{a}$.*

Thus we might have $c(\mathfrak{A}_{\mathscr{A}}[\![1+2]\!]) = \ulcorner 3 \urcorner$, or perhaps, if the language we are using does not have decimal notation (e.g., Peano, or primitive recursive arithmetic), then $c(\mathfrak{A}_{\mathscr{A}}[\![1+2]\!]) = \ulcorner s(s(s(0))) \urcorner$.

§ **2.4 Substitution** With encoding and the definition of canonical form in hand, the next thing to consider is substitution. Given a formula A_x with one free variable x, usually only some instances of $A_x[t]$ (where t is a ground term) are *true* while some are *false*, A_x itself being neither. Consider when A_x is the formula $x = 3$, here the particular instance where 3 is substituted for x is *true*, and all other instances are *false.* In order to consider whether A_x is *true* for a particular object $\mathfrak{a}$ in $\mathfrak{A}$ we have to build the instance where x has been replaced by a representation (canonical form) of $\mathfrak{a}$. So what is needed is a substitution facility, and this can now be defined. Given an $\mathscr{S}$ which supports encoding, and has a definition of canonical forms, $\mathscr{S}$ supports substitution if there is some formula sub_{wxyz} in $\mathcal{L}$ with only w, x, y, z free, such that, given a, b, c, d are ground terms, then $\mathfrak{A}[\![sub_{wxyz}[a,b,c,d]]\!]$ iff a encodes some formula A, b encodes some variable $\ulcorner x \urcorner$, c denotes the object with canonical form c', A^* is $A_x[c']$ and d encodes A^*. In future the functional abbreviation of this will be used, i.e., $sub(\cdot,\cdot,\cdot)$ where $\mathfrak{A}[\![sub_{wxyz}[a,b,c,sub(a,b,c)]]\!]$ Thus, for instance,

$$\mathfrak{A}_{\mathscr{A}}[\![sub[\ulcorner x = 1 \urcorner, \ulcorner x \urcorner, 1+1, \ulcorner 2 = 1 \urcorner]]\!]$$

a subset of itself (i.e., ground terms referring to distinct objects in $\mathfrak{A}$), a first restriction has been placed on possible self-referential systems: that they contain an infinite number of objects.

is *true*, and, importantly,

$$\mathfrak{A}_{\mathscr{A}}[\![sub[\ulcorner x = 1\urcorner, \ulcorner x\urcorner, 1+1, \ulcorner 1+1 = 1\urcorner]]\!]$$

is *false.*

I have not yet begun to discuss how $\mathscr{S}$ might interpret encoded strings (i.e., query itself), but what has been discussed above is already enough so that we can state some of the essential properties of self-referential systems. I mean by this that this provides sufficient tools to prove the basic result known as the diagonalisation lemma [1]:

Theorem 1 (Carnap) *Given a system $\mathscr{S}$ which has encoding and substitution, then for any formula A in $\mathfrak{L}$ with only one free variable x, there is a sentence B for which $\mathfrak{A}[\![A_x[\ulcorner B\urcorner] \leftrightarrow B]\!]$.*

§ **2.5 Reflection** Now it is possible to give a formal characterisation of reflection:

Definition 2 (Reflection) *A system $\mathscr{S}$ is reflective if it has encoding and substitution, and there is some formula P in $\mathfrak{L}$ such that for any sentence A in $\mathfrak{L}$, the formula $P_x[\ulcorner A\urcorner] \to A$ is* true. *In this case P is called a reflective predicate.*

There are obvious simple reflective predicates: $\bot$ is one example, since for any sentence A, clearly $\bot_x[\ulcorner A\urcorner] \to A$ is *true*[4]. This is not, however, very interesting. At the other end of the scale a formula P that defined exactly the true sentences about $\mathscr{S}$, i.e., $\mathfrak{A}[\![P[\ulcorner A\urcorner] \leftrightarrow A]\!]$, would be very interesting; however it is easy to show that such a formula does not exist: if it did, then by theorem 1, it would be possible to find a sentence B such that $\mathfrak{A}[\![\neg P[\ulcorner B\urcorner] \leftrightarrow B]\!]$, resulting in a contradiction[5]. However, there is a lot of room for mannoevre in the range between the extremes of $\bot$ and a perfect truth predicate. The question of what interesting reflective predicates we can find in this range is addressed next.

§ 3 Provability

§ **3.1 Theories** Consider a (true) sentence $\forall x A$ of $\mathscr{S}$; we do not in general show that this is true by finding its meaning directly; to do that we would have to check separately for every object $\mathfrak{a}$ of $\mathscr{S}$ that $\mathfrak{A}[\![A_x[\mathfrak{a}]]\!]$ is

[4]It is, of course, possible to substitute arbitrary terms into $\bot$, though any substitution will result simply in $\bot$.

[5]This is a form of Tarski's 'no truth definition' theorem [11].

true, and there may be an infinite number of possibilities that need checking. Instead, we *prove* that $\forall x A$ in some suitable theory. Informally, a theory of a system $\mathscr{S}$ is a finite[6] description of $\mathscr{S}$. This notion of a theory is now formalised:

Definition 3 (Theories) *A theory for a language $\mathfrak{L}$ is defined as either the closure of a set of formulae (called axioms) under a set of rules[7] R of a theory T'. A formula A is indicated to be in T by writing $\vDash_T A$. For a system $\mathscr{S}$, if $\vDash_T A$ implies $\mathfrak{A}[\![A]\!]$ then T is said to be true for $\mathscr{S}$.*

Definition 4 (Extensions) *A theory T_1 is an extension of a theory T if $T_1 \supset T$. In particular, if T is the theory defined as the closure under rules R of the theory T', and C is a set of axioms, then $T[C]$ is the extension of T defined as the closure of $T' \cup C$ under R.*

Usually some particular theory is given as the usual way to find out true statements of $\mathscr{S}$, and this is denoted $T_{\mathscr{S}}$. In the case of $\mathscr{A}$, for instance, the theory $T_{\mathscr{A}}$ is often taken to be Peano, or even primitive recursive, arithmetic.

There is an important difference between a theory and the system it represents: a system cannot be defined inside itself — that is clear from the impossibility of a perfect truth predicate. The same is certainly not true for theories: given a theory of a suitable system, it is very easy to build it inside that system: building a proof checking system for an arbitrary theory on a Lisp machine is an easy programming exercise.

§ 3.2 An interesting reflective predicate Given the observation in the paragraph above, for a system $\mathscr{S}$, an associated true theory $T_{\mathscr{S}}$ is an obvious candidate for an interesting reflective predicate. Consider the following definitions:

Definition 5 (Definable theories) *A theory T, with language $\mathfrak{L}$, is definable in $\mathscr{S}$ if there is some formula P in $\mathfrak{L}$ such that $\mathfrak{A}[\![[P_x[\ulcorner A \urcorner]]\!]$ if and only if $\vDash_T A$. P is said to define T.*

Definition 6 (Proof Predicate) *Given a theory T defined by P, where T is sufficiently expressive so that we can show $\vDash_T A$ if and only if $\vDash_T P_x[\ulcorner A \urcorner]$ then P is called a proof predicate for T. This is abbreviated to $Pr_T(\cdot)$.*

[6]Finite here includes schemas; for instance Peano arithmetic is a theory of arithmetic that cannot be defined with a finite number of axioms, but can be defined using a finite number of schemata.

[7]Rules are taken here to be relations between *finite* sets of formulae, and formulae.

Clearly, for a system $\mathscr{S}$ with associated definable true theory $T_{\mathscr{S}}$, $Pr_{T_{\mathscr{S}}}(\cdot)$ is also a reflective predicate (in which case it is called *sound*).

Above I suggested that a Lisp Machine is an example of a system for which it would be easy to implement a useful proof predicate. This is not really true — the only available semantics for a Lisp machine is the machine itself, hardware and software together, and there is no way to check against that whether any particular (remotely ambitious) theory of Lisp machines is true. If instead, we are willing to start with a more abstract system though, e.g. a programming system with a well defined semantics, then it it is remarkable how little we actually need — Gödel [3] provided a sound proof predicate for $\mathscr{A}$ taking $T_{\mathscr{A}}$ to be Peano arithmetic (in fact only a fragment of primitive recursive arithmetic is actually needed — something which any practical programming system one can imagine using would certainly contain).

A proof predicate for $T_{\mathscr{S}}$ immediately uncovers limitations with $T_{\mathscr{S}}$ as a characterisation of $\mathscr{S}$: by theorem 1 there is a sentence B such that $\mathfrak{A}[\![\neg Pr_{T_{\mathscr{S}}}(\ulcorner B \urcorner) \leftrightarrow B]\!]$, and $T_{\mathscr{S}}$ cannot distinguish whether B is true or false[8]. This might not be a problem, since there might not be any 'interesting' true formulae that $T_{\mathscr{S}}$ cannot prove. Unfortunately this is not so: one obvious useful class of true formulae, given a proof predicate $Pr(\cdot)$, is $Pr(\ulcorner A \urcorner) \rightarrow A$; but it is not the case that $\vdash_{T_{\mathscr{S}}} Pr_{T_{\mathscr{S}}}(\ulcorner A \urcorner) \rightarrow A$[9].

§ **3.3 Reflective extension** Since $T_{\mathscr{S}}$ is the tool used to identify true sentences of $\mathscr{S}$, it should be as complete a characterisation of $\mathscr{S}$ as possible. But once it is possible to build a proof predicate, any characterisation is necessarily incomplete by the results just mentioned, and in fact incomplete in a directly inconvenient way.

However, a corollary of the observation is that that there is a general method for building extensions to $T_{\mathscr{S}}$ that are more complete. For instance $T'_{\mathscr{S}} \equiv T_{\mathscr{S}}[Pr_{T_{\mathscr{S}}}(\ulcorner A \urcorner) \rightarrow A]$ is an extension of $T_{\mathscr{S}}$ that is true for $\mathscr{S}$, and $\vdash_{T'_{\mathscr{S}}} Pr_{T_{\mathscr{S}}}(\ulcorner A \urcorner) \rightarrow A$[10].

This is the application of what is known as a reflection principle, defined by Feferman [2] as follows:

Definition 7 (Reflection Principle) *'By a reflection principle we understand a description of a procedure for adding to any set of axioms C certain*

[8]This is Gödel's first incompleteness theorem [3].

[9]This is a generalisation of Gödel's second incompleteness theorem, which states that $\not\vdash_{T_{\mathscr{S}}} Pr_{T_{\mathscr{S}}}(\ulcorner \bot \urcorner) \rightarrow \bot$.

[10]It is important to distinguish this from $\vdash_{T'_{\mathscr{S}}} Pr_{T'_{\mathscr{S}}}(\ulcorner A \urcorner) \rightarrow A$, which is not true.

new axioms whose validity follow from the validity of axioms C and which formally express within the language of C evident consequences of the assumption that all the theorems of C are valid.'

There are two questions here: first, 'what are candidates for a reflective extension?'; and second, 'what are the effects of doing such a thing?'

The most obvious candidate is the one that has so far been mentioned, and which is known as the local reflection schema

$$(R_T^-) \qquad Pr_T(\ulcorner A \urcorner) \to A$$

However Feferman has proposed a more interesting possibility, subsuming R_T^-, called uniform reflection:

$$(R_T^+) \qquad \forall x(Pr_T(sub(\ulcorner A \urcorner, \ulcorner x \urcorner, x)) \to \forall x A.$$

It is interesting to consider the differences between, and implications of, the two: R_T^- does not obviously introduce new provable sentences. With R_T^+, on the other hand, the fact that each individual instance $A_x[n]$ is provable is no reason to believe that $\forall x A$ is provable, even if it is clearly true.

If we think in terms of the Curry Howard isomorphism between proofs and types, then a useful interpretation is to see that (at least in a constructive logic) a reflection principle amounts to an interpreter for the programming language corresponding to the logic. We can think of R_T^- as a function taking a proof (program) that there is something satisfying A, and normalising (executing) it, so as to recover that something. The meaning of R_T^+ is more complex: a functional that takes a proof (program) that there is something satisfying A for every instance of the free variable x, and returns a function that, given any term t will return something that satisfies $A_x[t]$[11].

§ 3.4 Effects of reflective extension The effect of extending a theory with uniform reflection can be quite dramatic. For instance, given, as an example, $\mathscr{A}$, then irrespective of what theory we use to characterise it, we can easily find a formula that is true, but not provable in the theory. However, if we start from even a very small theory, such as primitive recursive arithmetic (PRA), if we extend it repeatedly using reflection then for any formula true in $\mathscr{A}$ we will eventually produce a theory where it is provable [2].

[11]This interpretation suggests a formal analogue of self-referential programming languages such as, for instance, 3-Lisp [9], where the interpreter (i.e., the proof predicate) is available as part of the language itself.

This is an impressive result, though its importance should not be overstated, since in order to get to some of the more 'difficult to prove' true sentences we need to make use of 'transfinite' extensions, and the precise way that the theory is extended is no longer clear, even though the method appears, at first sight, systematic. However even finite extensions can produce large jumps in the power of a theory to describe its associated system. The result of single extensions using uniform reflection have been analysed. For instance $PRA[R^+_{PRA}] \equiv PA$, and $PA[R^+_{PA}]$ provides induction over the ordinal ϵ_0 [5].

At least as important as any gross increase in the strength of the proof system is simply that reflective extension allows us to make proper and intuitive use of reflective facilities[12].

§ 4 Implementation

If a proof predicate is available, it can provide interesting facilities, but the question of how to make it available still has to be addressed. Gödel's method in [3], for instance, is not practical: it is difficult to imagine actually doing, in a proof development system, what Gödel describes even once, for arithmetic, never mind for every particular case. Even if we allow a more practical programming language than pure arithmetic, it does not become much more practical. Some other method is needed.

As alternative to actually building a proof predicate, it might be possible to find a theory that captures the necessary behaviour of any such predicate. This is the idea that will be considered now.

§ 4.1 The derivability conditions A characterisation of the properties of the proof predicate was first attempted by Bernays, in his proof of the second incompleteness theorem [4]. However, his list of *derivability conditions* was designed for a particular proof, not as a proper characterisation of the proof predicate. This was done by Löb [6], who gave an alternative list of three derivability conditions. If $Pr(\cdot)$ is a proof predicate then:

$$(D_1) \qquad \vdash A \quad \text{implies} \quad \vdash Pr(\ulcorner A \urcorner)$$

$$(D_2) \qquad \vdash Pr(\ulcorner A \to B \urcorner) \to Pr(\ulcorner A \urcorner) \to Pr(\ulcorner B \urcorner)$$

$$(D_3) \qquad \vdash Pr(\ulcorner A \urcorner) \to Pr(\ulcorner Pr(\ulcorner A \urcorner) \urcorner).$$

[12] In fact it is possible to argue that arguments about how strong a proof system is are not, in practice, very important, since if a function cannot be shown to terminate using facilities already available in PRA, then it is either not guaranteed to terminate (which does not mean it is not useful), or unlikely to be very practical anyway.

These are obvious properties that a proof predicate should have. They do not, however, describe completely its behaviour; for this it is necessary to add Löb's theorem:

$$(LT) \qquad \vdash Pr(\ulcorner A \urcorner) \to A \quad \text{iff} \quad \vdash A.$$

This list of properties suggests that $Pr(\cdot)$ can be interpreted as $\Box$ in a modal propositional logic where the correspondents of the derivability conditions and Löb's theorem are used as the defining rules and axioms (call this theory PRL — for PRovability Logic). Then Solovay has shown [10] that if an interpretation of formulae in the language of arithmetic is defined to be a translation $*$ where

$$\begin{aligned} P* \ (P \text{ atomic}) &\equiv \text{some sentence of arithmetic} \\ \bot* &\equiv \bot \\ (A \circ B)* &\equiv (A*) \circ (B*) \\ (\Box A)* &\equiv Pr(\ulcorner A* \urcorner) \end{aligned}$$

(where $\circ \in \{\wedge, \vee, \to\}$), then PRL defines exactly the sentences provable in every formulation of primitive recursive arithmetic, $*$ where $Pr(\cdot)$ is a proof predicate.

Since the derivability conditions define the provable schemata for the proof predicate, would it not be possible to define a proof predicate using them? The problem is that they describe the behaviour of $Pr(\cdot)$ applied to ground terms and a slightly more general characterisation is needed if uniform reflection is to be implemented. Parametrised forms of the derivability conditions are needed, and these are defined as follows:

$$(D_1') \qquad \vdash A \quad \text{implies} \quad \vdash Pr(\ulcorner A_x[\check{t}\,] \urcorner)$$

$$(D_2') \qquad \begin{aligned} &\vdash Pr(\ulcorner (A \to B)_x[\check{t}\,] \urcorner) \to \\ &\qquad Pr(\ulcorner A_x[\check{t}\,] \urcorner) \to Pr(\ulcorner B_x[\check{t}\,] \urcorner) \end{aligned}$$

$$(D_3') \qquad \vdash Pr(\ulcorner A_x[\check{t}\,] \urcorner) \to Pr(\ulcorner Pr(\ulcorner A_x[\check{x}] \urcorner)_x[\check{t}\,] \urcorner)$$

where $\ulcorner A_x[\check{t}\,] \urcorner$ is an abbreviation for $sub(\ulcorner A \urcorner, \ulcorner x \urcorner, t)$, and in D_1' the variable x does not have to occur free in A (if it does not, then substitution for it is simply an identity operation). Löb's theorem could be parametrised in the same way, but it turns out that it can be derived from the others given the quotation and substitution facilities already defined, and the parametrised forms of the derivability conditions.

§ 4.2 Substitution and Encoding

Before defining a proof predicate using even the modified form of the derivability conditions, I have to deal with a further problem: the theory has to provided with some way of dealing with encoded formulae and terms, and (more tricky) I have to define the behaviour of substitution for non-ground terms. In § 2.4 I discussed the meaning of $sub(\cdot,\cdot,\cdot)$ in a system; however that discussion did not consider the proof theory of substitution — tools for reason about it. That is what I will do now.

Consider the behaviour of $sub(\ulcorner A \urcorner, \ulcorner x \urcorner, t)$ where t is a ground term denoting $\mathfrak{a}$. In a suitable true theory we can show this to be equal to $\ulcorner A^* \urcorner$, where a has canonical form t' and A^* is $A_x[t']$. To extend this discussion to non-ground terms, definition 1 has to be generalised.

Definition 8 (Non-Ground Canonical Form) *In a true theory, a non-ground term $t_{ab\ldots}$ is in canonical form if, given any ground canonical terms $t_1, t_2, \ldots$, the ground term $t_{ab\ldots}[t_1, t_2, \ldots]$ is the canonical form of the object it denotes.*

So, for the theory PRA, $s(s(x))$ would be in canonical form (at least for one of the common ways of representing integers). On the other hand, $s(x)+s(y)$ is not in canonical form; this is, though, equivalent to $s(s(x+y))$ — a canonical form into which has been substituted the term $x+y$.

Now I can axiomatize the behaviour of $sub(\cdot,\cdot,\cdot)$. If $t'_{ab\ldots}$ is in canonical form, then for any terms $t_1, t_2, \ldots$

$$\begin{aligned}&\vdash sub(\ulcorner A \urcorner, \ulcorner x \urcorner, t'_{ab\ldots}[t_1, t_2, \ldots]) \\ &\quad = \quad sub(\cdots(sub(sub(\ulcorner A^* \urcorner, \ulcorner a \urcorner, t_1), \ulcorner b \urcorner, t_2), \ldots)\ldots)\end{aligned}$$

where A^* is $A_x[t'_{ab\ldots}]$. Other axioms are equally easy to provide.

I can then provide encoding straightforwardly: I simply extend the definition of $\mathcal{L}$ with the square quotation marks that I have been using up to now as an abbreviation mechanism, then $\mathcal{L}^*$ is the smallest extension of $\mathcal{L}$ closed under the rule that if A (or t) is a formula (or term) of $\mathcal{L}^*$ then $\ulcorner A \urcorner$ (or $\ulcorner t \urcorner$) is a constant term of $\mathcal{L}^*$.

§ 4.3 Reflective extension

Now a proof predicate can be added to the theory. Notice that the derivability conditions on the proof predicate cannot be considered simply as axioms defining its behaviour: D'_1 is a rule, not an implication — if it is written as an implication it reads as a statement of completeness for the theory that is not even true, never mind provable. The other two however, can be used as axioms.

Given a theory T defined as the closure of a theory T' under rules R, it is possible to extend this to a theory with a proof predicate as follows: the language $\mathcal{L}$ of T is extended to $\mathcal{L}^*$ as described above and with the addition also of a new one place predicate $Pr_{T^*}(\cdot)$. Then if Ax_{sub} is the set of axioms for substitution and Ax_{Pr_T} is the axioms for D'_2 and D'_3, then there is still the question of how give $Pr_T(\cdot)$ the behaviour required by D'_1. This is done by adding a new rule to the system. So the extension T^* of T with quotation, substitution and a predicate $Pr_{T^*}(\cdot)$ defined by the derivability conditions, is $T' \cup Ax_{sub} \cup Ax_{Pr_T}$ closed under $R \cup \{\{A\} \mapsto Pr_{T^*}(\ulcorner A \urcorner)\}$.

T^* is (hopefully) a conservative extension of T, that is T^* proves nothing (that does not involve $Pr_{T^*}(\cdot)$) that T cannot already prove. Now we have to add a reflection schema to the theory as a new class of axioms. However we cannot do this simply by adding new axioms to T^*, to get the theory $T^*[R^+_{T^*}]$, since by D'_1 then $\vdash_{T^*[R^+_{T^*}]} Pr_{T^*}(\ulcorner R^+_{T^*} \urcorner)$ which is not true[13]. So at the same time as the reflection schema is added to the theory, D'_1 has to be taken out. Thus T^+ is defined as the closure of $T^* \cup R^+_{T^*}$ under R.

§ **4.4 The truth of T^+** An important issue is whether T^+ is true or not. We cannot just throw new axioms into a theory as we feel like it and expect that the result will be consistent, nevermind meaningful; especially when we are discussing an area as subtle as self-reference. Unless the theory I have produced is not only consistent, but also an accurate (if not complete) description of the system, then there is no point in using it.

However, I can show that the new theory is true, and therefore consistent, at least for systems $\mathscr{S}$ with theories $T_{\mathscr{S}} \supset PRA$, as follows: Assume T is a true theory for $\mathscr{S}$, and $T \supset PRA$. Then by [3] I can can define encoding, substitution and a proof predicate P in T; i.e., so that $\mathfrak{A}[\![P[\ulcorner A \urcorner]]\!]$ if and only if $\vdash_T A$ and $\vdash_T P[\ulcorner A \urcorner]$ if and only if $\vdash_T A$. Thus T^* is also true, since the new constants for the quoted terms in $\mathcal{L}^*$ can be interpreted as the Gödel numbers of the appropriate terms and formulae, the new substitution operator by the defined substitution function since the definition satisfies the axioms, and finally, by Löb's results, the rules and axioms for Pr_{T^*} are satisfied by P. Thus T^* is a true conservative extension of T. In the same way, for any A, $\vdash_{T^+} A$ implies $\vdash_{T[R^+_T]} A$, thus T^+ is also true.

And since any programming facility that we are likely to encounter in practice easily includes PRA, this proof is sufficent for most practical applications.

[13]It is another instance of the second incompleteness theorem.

§ 5 Non-Conservativity

Having suggested a way to extend a theory with a proof predicate, while avoiding the work of the usual approach, I have still to show that this produces the wanted non-conservative effects. But it is quite easy to show that the following result holds:

Theorem 2 *The theory PRA^{+} contains PA.*

In fact PRA^{+} and PA prove exactly the same theorems (the details, including the issues involved in a machine implementation, are in [7]).

§ 6 Conclusions

I have proposed a method for adding a proof predicate to a theory that is based on the necessary behaviour of such a thing, rather than being a 'Gödel-style' encoding of the proof theory, and I have shown that this captures many of the properties of a Gödel-style proof predicate, at least for simple theories.

The advantages of my approach seem to be severalfold: it provides a uniform method for developing the theory of a self-referential system by starting from a simple non-self-referential system, and the facilities that are provided are equally usable by the system itself, thus providing a safe logical foundation for systems to 'reason about themselves'. The strengthening of the theory that occurs is an substantial further benefit, offering the possibility of shorter proofs, and proofs of previously unprovable facts. This strengthening means that the initial theory can be very simple, since it can be strengthened to whatever level is necessary by iterating the reflective extension, so that we can avoid the work involved in having to build theories that try to capture as much as possible of the properties of a system all at once.

Finally, this approach can be used to provide an interesting formal analogue, if we consider proofs as programs, of the idea of a self referential programming language/system. And this can offer insights into the expected behaviour of such a system. This especially applies if we want to provide a machine with a self-referential facilities: in the same way that we have to provide a machine with formal tools for reasoning about, say, uncertain information if we want it to be able to treat such information in a sophisticated manner, we have to provide a machine with formal tools for self-referential reasoning if we want it to be safely/sensibly self-referential.

There remain obvious questions though: a theory like PRA can easily be extended to PA, but what is the effect of iterated extensions compared with using the full Gödel style approach.

One apparent criticism of the approach is that it does not allow the same sort of meta-level reasoning that is possible with a 'proper' proof predicate: a Gödel style proof predicate for a theory can be used for much more general meta-theoretic applications. To do this, however, a full proof predicate really is needed; an abstraction of it is no longer enough. I tend to think that a better approach here is to use a special generic meta-theory (or 'framework'), in which the language, axioms and rules of the object theory can be described formally, and which is designed specifically for the purpose of doing meta-theory. This approach is discussed in [8], and [7] where meta-theoretic and reflective reasoning are combined.

Another criticism is that the development here concentrates on systems that are close to the traditional mathematical idea of such things, i.e., stateless; this is fine as long as we are thinking of functional or applicative programming systems such as pure Lisp or pure Prolog. However it is an unfortunate but unavoidable fact that the real world involves, a lot of the time, a changing state; if we want to be able to supply such systems with introspective facilities, then we need some idea of a changing state too. Thus it would be very interesting to consider how the ideas here can be applied to systems based on temporal, rather than classical or constructive logics.

Addendum A second paper is planned as a followup to this work; it will look at some of the questions raised and try to answer them. The authors will be me and Alex Simpson, and it will appear as MPI–93–201.

References

[1] Rudolf Carnap. *Logische Syntax der Sprache.* Julius Springer, Vienna, 1934.

[2] Solomon Feferman. Transfinite recursive progressions of axiomatic theories. *Journal of Symbolic Logic*, 56:259–316, 1962.

[3] Kurt Gödel. Über formal unentscheidbare Satz der Principia Mathematica. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

[4] David Hilbert and Paul Bernays. *Grundlagen der Mathematik, Band 2.* Julius Springer, Berlin, 1939.

[5] Georg Kreisel and Azriel Lévy. Reflection principles and their use for establishing the complexity of axiomatic systems. *Zeitschrift für Mathematische Logik*, 14:97–142, 1968.

[6] Martin Löb. Solution of a problem of Leon Henkin. *Journal of Symbolic Logic*, 20:115–118, 1955.

[7] Seán Matthews. *Metatheoretic and Reflexive Reasoning in Mechanical Theorem Proving.* PhD thesis, University of Edinburgh, 1992.

[8] Seán Matthews, Alan Smaill, and David Basin. Experience with FS_0 as a framework theory. In Gerard Huet and Gordon Plotkin, editors, *Logical Frameworks II.* Cambridge University Press, 1992. (Provisional title; to appear).

[9] Brian Cantwell Smith. *Reflection and Semantics in a Procedural Language.* PhD thesis, Massachusetts Institute of Technology, 1982.

[10] Robert Solovay. Provability interpretations of modal logic. *Israel Journal of Mathematics*, 25:287–304, 1976.

[11] Alfred Tarski. Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1:261–405, 1936. English translation, *The concept of truth in formalised languages*, in [12].

[12] Alfred Tarski. *Logic, Semantics, Metamathematics.* Oxford University Press, 1956.