Steps Towards a Computational Metaphysics *

Branden Fitelson
University of California–Berkeley

Edward N. Zalta
Stanford University

June 16, 2006

Abstract. In this paper, the authors describe their initial investigations in computational metaphysics. Our method is to implement axiomatic metaphysics in an automated reasoning system. In this paper, we describe what we have discovered when the theory of abstract objects is implemented in PROVER9 (a first-order automated reasoning system which is the successor to OTTER). After reviewing the second-order, axiomatic theory of abstract objects, we show (1) how to represent a fragment of that theory in PROVER9's first-order syntax, and (2) how PROVER9 then finds proofs of interesting theorems of metaphysics, such as that every possible world is maximal. We conclude the paper by discussing some issues for further research.

Keywords: axiomatic metaphysics, computational metaphysics, automated reasoning

1. Introduction

The first steps towards a computational metaphysics begin with an axiomatic metaphysics. The idea of axiomatic metaphysics is not a new one. Philosophers since Plato and Aristotle have always looked to Euclid's axiomatization of geometry as an ideal way of organizing a science. More recently, Spinoza (Spinoza, 1677) attempted to produce an axiomatic philosophical theory. Leibniz went a step further by including a computational component, as can be seen from the following:[1]

 The authors would like to thank Chris Menzel and Paul Oppenheimer for extremely helpful discussions about our representation of object theory in PROVER9 syntax. A web page has been built in support of the present paper; see <http://mally.stanford.edu/cm/> and its mirror at <http://fitelson.org/cm/>.

[1] The first passage is from G.vii, 21:

> Car si nous l'avions telle que je la conçois, nous pourrions raisonner en metaphysique et en morale à pue pres comme en Geometrie et en Analyse

See (Leibniz, 1890, G.vii, 21). The second citation is from G.vii, 200:

> Quo facto, quando orientur controversiae, non magis disputatione opus erit inter duos philosophos, quam inter duos Computistas. Sufficiet enim calamos in manus sumere sedereque ad abacos, et sibi mutuo ... dicere: calculemus.

See (Leibniz, 1890, G.vii, 200).

> If we had it [a *characteristica universalis*], we should be able to reason in metaphysics and morals in much the same way as in geometry and analysis. (Russell, 1900, 169)

> If controversies were to arise, there would be no more need of disputation between two philosophers than between two accountants. For it would suffice to take their pencils in their hands, to sit down to their slates, and to say to each other ... : Let us calculate. (Russell, 1900, 170)

Notice that the Latin word Russell translates as 'accountants' is 'Computistas', and the Latin word translated as 'slates' is 'abacos'. Clearly, this Latin usage, together with the final sentence 'Calculemus', reveals that a computational metaphor is governing Leibniz's thought.

In this paper, we show how one can implement an axiomatic metaphysics within an automated reasoning environment, thereby validating ideas of the ancients, Spinoza, and Leibniz. In Section 2, we review our preferred axiomatic metaphysical theory. In Section 3, we describe a partial implementation of that theory of using the first-order automated reasoning program PROVER9 (the successor to OTTER), developed at the Argonne National Laboratory.[2] In Section 4, we apply our implementation to a well-known philosophical domain: situation and world theory (Zalta, 1993). We conclude, in Section 5, by discussing (1) how the first-order automated model-finding program MACE, which comes with PROVER9 and OTTER, helped us to discover an error of reasoning in a previous publication on object theory, concerning the Platonic Theory of Forms (Pelletier and Zalta, 2000), and (2) some problems for future research involving applications to Leibniz's theory of concepts and Frege's theory of the natural numbers.

2. Brief Review of Axiomatic Metaphysics

In this section, we review the principles of object theory, as described in detail in (Zalta, 1983) and elsewhere. Object theory is naturally formulated in a second-order modal language that has been modified only so as to admit a second kind of atomic formula. In addition to the usual 'exemplification' formulas such as '$F^n x_1 \ldots x_n$', object theory allows 'encoding' formulas such as 'xF^1', where 'F^1' is a one-place predicate. These new atomic formulas are to be read: x encodes F^1. (In what follows, we drop the superscript indicating the arity of the relation in both kinds of formulas, since this can be inferred.) These encoding formulas represent a new form of predication. The idea for such formulas derives from the work of Ernst Mally (Mally, 1912), who

[2] See <http://www-unix.mcs.anl.gov/~mccune/prover9/>.

suggested that abstract objects are determined by (i.e., encode), but need not exemplify, the properties by which we conceive of them. On Mally's view, any group of properties (including inconsistent sets of properties) determine an abstract object, and this is captured in object theory by a comprehension schema that asserts, for any condition ϕ on properties, that there is an abstract object x which encodes all and only the properties satisfying (in Tarski's sense) ϕ. Object theory asserts nothing about the properties that ordinary objects exemplify, though it does assert that they don't encode properties. This is something only abstract objects do.

Object theory has been applied in various ways, and some example descriptions of interesting abstract objects are provided below. However, before we state these examples, we provide a formal development of the theory. Readers who want more of an introduction to object theory, including more motivation and history, should consult one of the other texts on object theory listed in the Bibliography.

The language of object theory is as follows:

Object variables and constants: $x, y, z, \ldots$; $a, b, c, \ldots$

Relation variables and constants: $F^n, G^n, H^n, \ldots$;
$P^n, Q^n, R^n, \ldots$ (when $n \geq 0$); $p, q, r, \ldots$ (when $n=0$)

Distinguished 1-place relation: $E!$ (read: *concrete*)

Atomic formulas:
$F^n x_1 \ldots x_n$ ('$x_1, \ldots, x_n$ exemplify F^n')
xF^1 ('x encodes F^1')

Complex Formulas: $\neg\phi$, $\phi \rightarrow \psi$, $\forall\alpha\phi$ (α any variable), $\Box\phi$

Complex Terms:
Descriptions: $\imath x\phi$
λ-predicates: $[\lambda x_1 \ldots x_n\ \phi]$ (ϕ no encoding subformulas)

The only difference between this language and the language of the second-order modal predicate calculus (with descriptions and λ-expressions) is the presence of a second atomic formula, 'xF', which expresses the fact that object x encodes property (one-place relation) F.

The most important definitions of object theory can be summarized as follows:

&, $\vee$, $\equiv$, $\exists$, and $\Diamond$ are all defined in the usual way

$O!\ =_{df}\ [\lambda x\ \Diamond E!x]$ ('ordinary')

$A!\ =_{df}\ [\lambda x\ \neg\Diamond E!x]$ ('abstract')

$x =_E y\ =_{df}\ O!x\ \&\ O!y\ \&\ \Box\forall F(Fx \equiv Fy)$

$x = y\ =_{df}\ x =_E y\ \vee\ (A!x\ \&\ A!y\ \&\ \Box\forall F(xF \equiv yF))$

$F^1 = G^1\ =_{df}\ \Box\forall x(xF^1 \equiv xG^1)$

$F^n = G^n\ =_{df}$ (where $n > 1$)
$\forall x_1 \ldots\ \forall x_{n-1}([\lambda y\ F^n y x_1 \ldots x_{n-1}] = [\lambda y\ G^n y x_1 \ldots x_{n-1}]\ \&$
$[\lambda y\ F^n x_1 y x_2 \ldots x_{n-1}] = [\lambda y\ G^n x_1 y x_2 \ldots x_{n-1}]\ \& \ldots \&$
$[\lambda y\ F^n x_1 \ldots x_{n-1} y] = [\lambda y\ G^n x_1 \ldots x_{n-1} y])$

$p = q\ =_{df}\ [\lambda y\ p] = [\lambda y\ q]$

The definition of relation identity, $F^n = G^n$, can be read more simply as follows: if the properties that can be constructed from F^n and G^n are identical, when considering all the pairwise ways of plugging $n-1$ objects into both F^n and G^n in the same order, then $F^n = G^n$. In the last definition, 'p' and 'q' are used as 0-place relation variables, which range over propositions.

The logic underlying the theory of objects can now be summarized. With the exception of the Logic of Descriptions, the modal closures of all of the following are axioms:

Simplest second-order quantified S5 modal logic (Linsky and Zalta, 1994), including 1st and 2nd order Barcan formulas (i.e., fixed domains)

Logic of Encoding: $\Diamond xF \rightarrow \Box xF$

Logic of Identity: $\alpha = \beta \rightarrow [\phi(\alpha, \alpha) \equiv \phi(\alpha, \beta)]$
(β substitutable for α)

Logic of λ-Predicates: (β, η, and α conversion)
$[\lambda x_1 \ldots x_n\ \phi]y_1 \ldots y_n \equiv \phi^{y_1,\ldots,y_n}_{x_1,\ldots,x_n}$ (ϕ free of descriptions)
$[\lambda x_1 \ldots x_n\ F^n x_1 \ldots x_n] = F^n$
$[\lambda x_1 \ldots x_n\ \phi] = [\lambda x'_1 \ldots x'_n\ \phi']$ (ϕ, ϕ' alphabetic variants)

Logic of Descriptions:
$\psi^{\imath x\phi}_z \equiv \exists x(\phi\ \&\ \forall y(\phi^y_x \rightarrow y = x)\ \&\ \psi^x_z)$ (ψ atomic/identity)

This is essentially classical S5 modal logic for the second-order modal predicate calculus, modified only to include (a) the logic of encoding, (b) the logic of λ-expressions (complex expressions for denoting complex relations), and (c) the logic of definite descriptions which are rigid

and which may fail to denote. The Logic for Descriptions schema accomodates rigid definite descriptions by requiring that only non-modal instances (and their not modal closures) are axioms. It allows for descriptions which may fail to denote being applicable only to descriptions which occur in atomic and identity formulas.

Finally, the proper axioms of object theory can be stated as follows:

$$O!x \rightarrow \Box\neg\exists F\, xF$$

$$\exists x(A!x \,\&\, \forall F(xF \equiv \phi)), \text{ where } \phi \text{ has no free } x\text{s}$$

The second axiom is the comprehension schema for abstract objects. Notice that the following is an immediate consequence of the comprehension schema, given the logic of descriptions and the definition of identity for objects:

$$\imath x(A!x \,\&\, \forall F(xF \equiv \phi))G \;\equiv\; \phi^G_F$$

This says: the abstract object which encodes just the properties satisfying ϕ encodes property G if and only if G satisfies ϕ. These proper theorems all involve canonical descriptions of abstract objects. In what follows, these canonical descriptions play an important role in the applications.

Though we shall assume some familiarity with object theory, here are some examples for those who might be encountering it for the first time. All of the following identify abstract objects in terms of canonical descriptions:

The Complete Concept of $y =$
$\imath x(A!x \,\&\, \forall F(xF \equiv Fy))$

The Actual World $=$
$\imath x(A!x \,\&\, \forall F(xF \equiv \exists p(p \,\&\, F{=}[\lambda y\, p])))$

The Truth Value of $p =$
$\imath x(A!x \,\&\, \forall F(xF \equiv \exists q(q{\equiv}p \,\&\, F{=}[\lambda y\, q])))$

The Extension of the Concept $G =$
$\imath x(A!x \,\&\, \forall F(xF \equiv \forall y(Fy \equiv Gy)))$

The Form of $G =$
$\imath x(A!x \,\&\, \forall F(xF \equiv \Box\forall y(Gy \rightarrow Fy)))$

As an example of how the theory can be used to prove consequences which are proper theorems of metaphysics, consider the following series of definitions and theorems:

$x \vDash p$ (‘p *is true in* x’) $=_{df}$ $x[\lambda y\, p]$

$World(x) =_{df} \Diamond\forall p(x \vDash p \;\equiv\; p)$

$Maximal(x) =_{df} \forall p(x \vDash p \;\vee\; x \vDash \neg p)$

$Consistent(x) =_{df} \neg\exists p[x \vDash (p \,\&\, \neg p)]$

$Actual(x) =_{df} \forall p(x \vDash p \rightarrow p)$

Theorem: $\forall x(\, World(x) \rightarrow Maximal(x))$

Theorem: $\forall x(\, World(x) \rightarrow Consistent(x))$

Theorem: $\exists! x(\, World(x) \,\&\, Actual(x))$

Theorem: $\Box p \;\equiv\; \forall w(w \vDash p)$

In what follows, we show how one can use PROVER9 to implement this axiomatic metaphysics, and to find proofs of the above theorems.

3. Implementing Object Theory in PROVER9

This theory of abstract objects is couched in a second-order language with predicates, modal operators, λ-expressions, and definite descriptions. This poses several challenges for representation and implementation in PROVER9, which we describe below. However, the challenges are not as imposing as they might seem. Models of object theory developed independently by Dana Scott (Zalta, 1983, 160–164), and by Peter Aczel (Zalta, 1999, 11–13), show that the theory is is essentially first-order despite being couched in second-order language. The only aspect of object theory that can't be fully captured in a first-order framework is the use of comprehension schemata in formulating the axioms. But we shall sidestep this issue by invoking particular instances of the comprehension schemata whenever we need them.

PROVER9 is an automated reasoning system which supports full first-order functional and predicate calculus with equality (but not second-order logic or λ-expressions). Our challenge is to represent the axioms and definitions of object theory in this environment. We must first translate claims from the language of object theory into PROVER9's language. Here, we use four key techniques:

1. Second-order language is translated into a first-order language with distinguished sortal predicates. Instead of quantifying over properties, propositions, etc., we quantify over a single domain, and introduce PROVER9 predicates to sort the domain into properties, objects, etc. Here we are simulating multi-sorted first-order logic.

2. The modal language of S5 is translated into quantified statements over 'points', using well known semantic representation of modal claims in first-order terms.

3. λ-expressions are translated into complex terms involving functions.

4. Definite descriptions are represented using predicates that guarantee the existence and uniqueness of any object satisfying them.

The first three of these techniques for representing higher-order and quantified modal formulas in a multi-sorted first-order environment are discussed in chapters 6 and 7 of (Manzano, 1996). We will give examples of all four techniques below.

The following table illustrates how the basic notation of object theory can be translated into PROVER9 syntax. Throughout the paper, we will write PROVER9 syntax in `typewriter` typeface.

Predicates	A, B, C (`A`, `B`, `C`)
Constants	a, b, c (`a`, `b`, `c`)
Variables	x, y, z (`x`, `y`, `z`)
Functions	f, g, h (`f`, `g`, `h`)
Quantifiers	$\forall$, $\exists$ (`all`, `exists`)
Connectives	$\&$, $\rightarrow$, $\vee$, $\neg$, $=$ (`&`, `->`, `|`, `-`, `=`)

Here, we see PROVER9's *formula* syntax, which uses quantifiers and all standard (infix) logical connectives. PROVER9's more basic syntax is *clausal*, which means quantifier-free, and using only disjunction, negation and identity. After one supplies PROVER9 with well-formed formulas (in PROVER9's formula syntax) as premises and conclusion, it will 'clausify' these formulas by eliminating quantifiers (by Skolemizing), and converting formulas to 'conjunctive normal form'. The resulting statements are in 'clausal normal form.' Here are some examples:

Formula	Clause (PROVER9 — Q-free, and CNF)
$(\forall x)(Px \rightarrow Qx)$	`-P(x) | Q(x).`
$(\exists x)(Px \,\&\, Qx)$	`P(a). Q(a).` (two clauses, new "`a`")
$(\forall x)(\exists y)(Rxy \vee x \neq y)$	`R(x,f(x)) | -(x = f(x)).` (new "`f`")
$(\forall x)(\forall y)(\exists z)(Rxyz \,\&\, Rzyx)$	`R(x,y,f(x,y)). R(f(x,y),x,y).` (new "`f`")

For readers unfamiliar with such clausification techniques, we recommend chapters 1 and 10 of (Kalman, 2001). For details on PROVER9's clause notation and syntax (which differs little from its predecessor OTTER (McCune, 2003b)), see (McCune, 2006).

Next, we will briefly explain how PROVER9 implements (mechanical) rules of inference. See (Portoraro, 2005) and (Wos et al., 1992) for encyclopedic general introductions to automated reasoning and its associated rules, techniques, and strategies. For our present purposes, it will suffice to discuss just one of these rules, namely, hyperresolution. Before we explain this rule, we first need to introduce the notion of a *most general unifier* of two expressions. If θ_1 and θ_2 are terms, there may or may not exist a set of substitutions σ such that $\sigma(\theta_1) = \sigma(\theta_2)$. If such a σ exists, it is called a *unifier* of θ_2 and θ_2, which are said to be *unifiable*. In such a case, $\sigma(\theta_1)$ $(= \sigma(\theta_2))$ is called a *common instance* of θ_1 and θ_2. For example, the terms $f(x, b)$ and $f(a, y)$ are unifiable, with unifier $\{a/x, b/y\}$, yielding the common instance $f(a, b)$. But the terms $f(x)$ and $g(y)$ are not unifiable, since any instance of $f(x)$ must begin with "f" and any instance of $g(x)$ must begin with "g". There are often *many* unifiers of two terms (up to alphabetic variants). A substitution σ is called a *most general unifier* of two terms θ_2 and θ_2 if it yields a *most general common instance* of θ_2 and θ_2. For instance, $\{a/x, a/y\}$ is a unifier of $f(x, y)$ and $f(a, x)$, yielding the common instance $f(a, a)$. But, this is not a most general unifier, because the unifier $\{a/x, z/y\}$ yields the more general common instance $f(a, z)$, which subsumes $f(a, a)$. The unification theorem (Robinson, 1963) guarantees the existence of a *unique* most general unifier for any two terms of first-order logic, and an algorithm for computing it (called the unification algorithm). This crucial algorithm makes it computationally feasible to implement the standard substitution rules for classical logic. Robinson's unification algorithm (or some variant of it) undergirds all modern automated reasoning systems.

Hyperresolution was also invented by Robinson (Robinson, 1965). Kalman (Kalman, 2001, chapter 2) gives many detailed examples of hyperresolution inferences in OTTER. We will provide just a brief introduction to hyperresolution here. Basically, hyperresolution is a generalization/mechanization of the *modus ponens* (or disjunctive syllogism) rule of classical logic, based on most general unification rather than arbitrary substitution. In general, hyperresolution infers from a mixed clause (a clause containing both positive and negative atomic subformulae) or negative clause (a clause containing only negative atomic subformulae) as *nucleus*, and positive clauses (with only positive literals) as *satellites*, one or more positive clauses as *hyperresolvents*. The inference is made by using the satellites to cancel, or "clash" against,

or "resolve", the negative literals in the nucleus. For example, from the nucleus `-P | M` and the satellite `P`, hyperresolution enables us to infer the hyperresolvent `M` by clashing the satellite against the negative literal in the nucleus; from the nucleus `-P(x) | M(x)` and the satellite `P(s)`, hyperresolution enables us to infer the hyperresolvent `M(s)` by clashing the satellite against the negative literal `-P(s)` in the "instance" `-P(s) | M(s)`. Here are some examples of valid hyperresolution inferences:

```
-P | M.      -P(x) | M(x).      -L(x,f(b)) | L(x,f(a)).
P.           P(x).              L(y,f(y)).
-------      ----------         ----------------------
∴ M.         ∴ M(x).            ∴ L(b,f(a)).
```

So far, this sounds just like *modus ponens* (or disjunctive syllogism). But, unlike *modus ponens*, hyperresolution must make use of *most general* common instances, rather than the arbitrary common instances that are allowed in standard classical logical inferences. Thus,

```
-P(x) | M(x).
P(x).
-------------
∴ M(a).
```

is *not* a valid hyperresolution inference! In this case, `M(a)` is strictly less general than `M(x)`, which is the correct hyperresolution conclusion, based on the most general instance of the nucleus `-P(x) | M(x)` that can be clashed with the satellite `P(x)`. While this may seem like a limitation, it turns out that resolution techniques are complete for first-order logic without equality (Robinson, 1965).[3] In the case above, hypreresolution yields `M(x)`, which subsumes the desired `M(a)`.

PROVER9 establishes the validity of first-order arguments *via reductio ad absurdum*: PROVER9 reasons from the conjunction of the premises and the denial of the conclusion of a valid argument to a contradiction.

[3] We have chosen to discuss here only theorems of object theory that do not involve equality reasoning (see the next section for concrete examples). Of course, many theorems of object theory do explicitly involve equality reasoning. For such problems, we have used (in addition to hyperresolution) paramodulation (Robinson and Wos, 1969) and demodulation (Wos et al., 1967), which are mechanical rules for first-order equality reasoning that have been efficiently implemented in OTTER and PROVER9. See (Wos et al., 1992) and (Kalman, 2001) for extensive discussions of automated first-order equality reasoning. And, see our computational metaphysics website (at <`http://mally.stanford.edu/cm/`> or <`http://fitelson.org/cm/`>) for several examples of object theoretic reasoning involving equality. We have omitted such problems from the present discussion for reasons of simplicity and economy of presentation. It is worth noting, however, that (in principle) the use of equality *rules* like paramodulation (in addition to hyperresolution) can be eliminated by the addition of explicit (relational) equality *axioms* (Kowalski, 1970). As such, this does not (in principle) constitute a significant loss of generality in our discussion.

Here is a simplified description of PROVER9's main loop, which is similar to OTTER's (McCune, 2006; McCune, 2003b; Kalman, 2001):

1. Begin with a *list of premises* and a *list of conclusions*, the latter having a single member (the conclusion) at the outset.
2. Add the denial of the conclusion to the list of premises.
3. Using inference rules, *e.g.*, hyperresolution (and/or other forms of resolution), paramodulation (and/or other equality rules – see *fn.* 3), infer all clauses you can.
4. Process the clauses (check for subsumption, apply restriction strategies, *etc.*), discard unusable ones, and add the remaining ones to the list of conclusions.
5. Pick a member of the list of conclusions (using a heuristic — default is "pick shortest" or "best first" — others can be used), and add it to the list of premises.
6. Repeat steps 3 – 5 until you reach a contradiction (i.e., until you derive $\mathcal{P}$ and `-`$\mathcal{P}$, for some atomic PROVER9 sentence $\mathcal{P}$). $\bowtie$

Consider, as an example, the following simple argument:

$\forall x(Greek(x) \rightarrow Person(x))$.
$\forall x(Person(x) \rightarrow Mortal(x))$.
$Greek(socrates)$.

$Mortal(socrates)$

Here's a simple PROVER9 proof of the above:

```
1 Greek(socrates). [clausify]
2 -Mortal(socrates). [clausify]
3 -Greek(x) | Person(x). [clausify]
4 -Person(x) | Mortal(x). [clausify]
5 Person(socrates). [resolve (3 a 1 a)]
6 -Person(socrates). [resolve (4 b 2 a)]
7 [F]. [resolve (6 a 5 a)]
```

To implement axiomatic metaphysics in this framework, second-order object theory must be represented in PROVER9's first-order language with at least two *sortal predicates*: `Property` and `Object`. In addition, exemplification formulas of the form 'Fx' and encoding formulas of the form 'xF' (which are the two forms of predication in object theory) can be represented in PROVER9 as follows:

```
all x all F (Ex1(F,x) -> Property(F) & Object(x)).

all x all F (Enc(x,F) -> Property(F) & Object(x)).
```

It is important to remember here, and in what follows, that the variables `F` and `x` are both untyped. PROVER9 treats the above formulas as if they were the following:

```
all x all y (Ex1(y,x) -> Property(y) & Object(x)).

all x all y (Enc(x,y) -> Property(y) & Object(x)).
```

But it is convenient to use the variable `F` instead of `y` in what follows, since it helps one to remember the sortal categories of the arguments to relations and functions. Note that 2-place predication requires a new relation: `Ex2(R,x,y)`, etc.

Quantified modal (S5) claims can be translated into PROVER9 Kripke-style (Manzano, 1996, chapter 7), with the use of a third sortal predicate: `Point` (*not* `World`![4]).

```
all F all x all w (Ex1(F,x,w) ->
            Property(F) & Object(x) & Point(w)).
```

Here is an example of a simple theorem in object theory and how it gets translated into PROVER9:

> Necessarily, every object exemplifies some property.
> $\Box(\forall x)(\exists Q)Qx$.

We can translate this into the following PROVER9 formula:

```
all p all x ((Point(p) & Object(x)) ->
          (exists Q (Property(Q) & Ex1(Q,x,p))).
```

PROVER9 will process such a formula appearing in an input file and clausify it as:

```
-Point(p) | -Object(x) | Ex1(f(p,x),x,p).
```

The next obstacle to overcome involves propositions and complex properties. Propositions can't be defined as 0-place properties (PROVER9 has no such properties), so a fourth sortal predicate is required: `Proposition`. For instance, we'll need to represent in PROVER9 the fact that if p is a proposition, then so is its negation $\neg p$. We can do so using the following formula:

[4] We emphasize the use of the predicate `Point` and not the predicate `World` since the latter is a defined concept in world theory. The former is merely a semantic device for translating modal claims into PROVER9 syntax.


```
all p (Proposition(p) -> Proposition(~p)).
```

Notice that we use `~` to form a term which is a negation of the proposition `p`, rather than the symbol '–', which is PROVER9's syntax for formula negation.

Complex properties (i.e., λ-expressions) can be represented in PROVER9 using functions. E.g., we represent the property *being such that p* ('$[\lambda yp]$') using a functor `VAC`:

```
all p (Proposition(p) <-> Property(VAC(p))).

all x all p all w ((Object(x) & Proposition(p) & Point(w)) ->
            (Ex1(VAC(p),x,w) <-> True(p,w))).
```

Finally, definite descriptions are represented as follows. In object theory, one finds, for example, the following definition of The Form of F ('Φ_F'), where '$\Rightarrow$' is necessary implication:

$$\imath x(A!x \,\&\, \forall G(xG \equiv F \Rightarrow G))$$

We represent this in PROVER9 syntax by using the following two predicates. To represent Φ_F, for example, we must first define z is *a* Form of F, and then define z is *the* Form of F, as follows.[5]

```
all z all F ((Object(z) & Property(F)) ->
  (IsAFormOf(z,F) <->
     (Ex1(A,z,W) &
        (all F (Property(G) -> (Enc(z,G) <-> Implies(F,G)))))))).

all z all F ((Object(z) & Property(F)) ->
  (IsTheFormOf(z,F) <->
     (IsAFormOf(z,F) &
        (all y ((Object(y) & IsAFormOf(y,F)) -> y=z))))).
```

Now, if one wants to assert "x is the Form of F" in PROVER9, one writes `IsTheFormOf(x,F)`.

With our four sortal predicates and the other techniques in place, PROVER9 requires the following explicit sorting conditions, which assert that the sorts are disjoint:

```
all x (Property(x) -> -Object(x)).
all x (Property(x) -> -Proposition(x)).
```

[5] The expression '`Ex1(A,z,W)`' is used to assert that `z` exemplifies the property of being abstract ('`A`') at the distinguished semantic point `W`. Recall that the exemplification extension of a property such as `A` can vary from point to point. The definition therefore tells us what it is for an object `z` to be a Form of `F` at the distinguished point `W`, not what it is for an object `z` to be a Form of `F` at an arbitrary point `w`.

```
all x (Property(x) -> -Point(x)).
all x (Proposition(x) -> -Object(x)).
all x (Proposition(x) -> -Point(x)).
all x (Point(x) -> -Object(x)).
```

With these techniques we've used PROVER9 to find proofs (without guiding it using known lemmas) of all the theorems reported in (Zalta, 1993) on world theory and situation theory. The proof of one of these theorems is described in Section 4. We've also used PROVER9 to find proofs (again, without guiding its search) of the theorems reported in (Pelletier and Zalta, 2000) on the Third Man Argument, with one exception. The exception is discussed in Section 5, where we show how MACE, the automated model-finding program that comes with PROVER9, helped us to discover a countermodel to one of the propositions alleged to be a theorem of object theory in (Pelletier and Zalta, 2000).

4. Example PROVER9 Proof of a Theorem of Object Theory

We now present the PROVER9 proof of the claim that every world is maximal from axioms, theorems and definitions of object theory. We used the following axioms and definitions; in each case, we present the original version in object theory followed by both its translation and clausification in PROVER9:

1. Negations of propositions are propositions. (Metalogical Theorem)

```
all p (Proposition(p) -> Proposition(~p)).
```

This clausifies to:

```
-Proposition(x) | Proposition(~x).
```

2. 'Truth at a point' is coherent. (Metalogical Theorem)

```
all w all p ((Point(w) & Proposition(p)) ->
  (True(~p,w) <-> -True(p,w))).
```

This clausifies to:

```
-Point(x) | -Proposition(y) | True(~y,x)  | True(y,x).
-Point(x) | -Proposition(y) | -True(~y,x) | -True(y,x).
```

3. $Maximal(x) =_{df} \forall p(x \models p \vee x \models \neg p)$[6] (Definition)

[6] A few clarificatory notes about this definition are in order. First, as noted above at the end of Section 2, where 'truth in' is defined, 'p is true in x' is written


```
all x (Object(x) -> (Maximal(x) <->
  (Situation(x) & (all p (Proposition(p) ->
    TrueIn(p,x) | TrueIn(~p,x)))))).
```

This clausifies to:

```
-Object(x)| -Maximal(x) | Situation(x).
-Object(x)| -Maximal(x) | -Proposition(z)| TrueIn(z,x)| TrueIn(z,x).
-Object(x)| Maximal(x) | -Situation(x) | Proposition(f1(x)).
-Object(x)| Maximal(x) | -Situation(x) | -TrueIn(f1(x),x).
-Object(x)| Maximal(x) | -Situation(x) | -TrueIn(~f1(x),x).
```

4. $World(x) =_{df} \Diamond\forall p(x \models p \equiv p)$ (Definition)

```
all x (Object(x) -> (World(x) <->
  (Situation(x) & (exists y (Point(y) &
    (all p (Proposition(p) ->
      (TrueIn(p,x) <-> True(p,y)))))))))).
```

This clausifies to:

```
-Object(x) | -World(x) | Situation(x).
-Object(x) | -World(x) | Point(f2(x)).
-Object(x) | -World(x) | -Proposition(u) | TrueIn(u,x) | -True(u,f2(x)).
-Object(x) | -World(x) | -Proposition(u) | TrueIn(u,x) | -True(u,f2(x)).
-Object(x) | World(x) | -Situation(x) | -Point(y) | Proposition(f3(x,y)).
-Object(x) | World(x) | -Situation(x) | -Point(y) | TrueIn(f3(x,y),x) | True(f3(x,y),y).
-Object(x) | World(x) | -Situation(x) | -Point(y) | -TrueIn(f3(x,y),x) | -True(f3(x,y),y).
```

5. Worlds are objects. (MetaTheorem)

```
all x (World(x) -> Object(x)).
```

This clausifies to:

```
-World(x) | Object(x)
```

Now the claim to be proved is that all worlds are maximal:

Theorem: $\forall x(World(x) \rightarrow Maximal(x))$

```
all x (World(x) -> Maximal(x)).
```

'$x \models p$' in object theory, and in PROVER9, this gets written as '`TrueIn(p,x)`'. Second, maximality is defined on situations, which in object theory are objects x such that $\forall F(xF \rightarrow \exists p(F = [\lambda y\ p]))$, i.e., objects x such that every property x encodes is a property of the form *being such that* p, for some proposition p. Since the definition of a situation will play no role in the proof, we omit the representation of this definition in PROVER9. Third, truth *in a situation* is not to be confused with truth *at a point* in the semantic structure. So we must distinguish `TrueIn(p,x)` where `x` is a situation from `True(p,y)` where `y` is a point (i.e., an index in our Kripke translation for modal content). The latter was used in the representation of Proposition 2 above.

This clausifies to:

```
-World(x) | Maximal(x).
```

After PROVER9 clausifies the premises and the negation of the conclusion, it then implements its main loop. When it prints out a proof, it only lists the clauses that were actually used in the proof. So, in the first 11 steps of the following proof, the reader will find that not all of the clauses connected with the above premises are used in the proof. For example, not all of the clauses in the clausification of premises 3 and 4 above are used in the following proof.

```
1 -Proposition(x) | Proposition(~x). [clausify].
2 -Point(x) | -Proposition(y) | True(~y,x) | True(y,x). [clausify].
3 -Object(x) | Maximal(x) | -Situation(x) | Proposition(f1(x)). [clausify].
4 -Object(x) | Maximal(x) | -Situation(x) | -TrueIn(f1(x),x). [clausify].
5 -Object(x) | Maximal(x) | -Situation(x) | -TrueIn(~f1(x),x). [clausify].
6 -Object(x) | -World(x) | Situation(x). [clausify].
7 -Object(x) | -World(x) | Point(f2(x)). [clausify].
8 -Object(x) | -World(x) | -Proposition(y) | TrueIn(y,x) | -True(y,f2(x)). [clausify].
9 -World(x) | Object(x). [clausify].
10 World(c1). [clausify].
11 -Maximal(c1). [clausify].
12 Object(c1). [hyper(9,a,10,a)].
13 Point(f2(c1)). [hyper(7,a,12,a,b,10,a)].
14 Situation(c1). [hyper(6,a,12,a,b,10,a)].
15 Proposition(f1(c1)). [hyper(3,a,12,a,c,14,a),unit_del(a,11)].
16 True(~f1(c1),f2(c1)) | True(f1(c1),f2(c1)). [hyper(2,a,13,a,b,15,a)].
17 Proposition(~f1(c1)). [hyper(1,a,15,a)].
18 TrueIn(~f1(c1),c1) | True(f1(c1),f2(c1)). [hyper(8,a,12,a,b,10,a,c,17,a,e,16,a)].
19 TrueIn(f1(c1),c1) | TrueIn(~f1(c1),c1). [hyper(8,a,12,a,b,10,a,c,15,a,e,18,b)].
20 TrueIn(f1(c1),c1). [hyper(5,a,12,a,c,14,a,d,19,b),unit_del(a,11)].
21 $F. [hyper(4,a,12,a,c,14,a,d,20,a),unit_del(a,11)].
```

We have chosen a relatively simple proof here, for ease of exposition. Readers interested in seeing how PROVER9 proves the other theorems in (Zalta, 1993) and the *bona fide* theorems in (Pelletier and Zalta, 2000) are encouraged to consult <http://mally.stanford.edu/cm/> or <http://fitelson.org/cm/>. Some of those proofs are very complex.

5. Observations

Why did we choose a first-order system like PROVER9, instead of a higher-order system? When we started this project, the only higher-order systems with which we were familiar were systems designed mainly for verification, e.g., Boyer and Moore's system NQTHM (Boyer and Moore, 1979). But we wanted the reasoning engine to *discover* proofs of non-trivial depth rather than merely verifying them. Moreover, we wanted to be able to constructively establish the consistency of the

premises used in the proofs of the theorems. So we decided to use the first-order systems PROVER9 and MACE, which are flexible, powerful, and robust first-order systems for proving theorems and finding models, respectively, and which are freely available and compile easily on a wide variety of platforms. More recently, we have become aware of the existence of various higer-order theorem provers that can find non-trivial proofs (rather than just verifying known proofs).[7] But one advantage of our current approach is that we have both proofs of our theorems *and* models of the premises involved in the theorems. This gives us an automated check of the consistency of our assumptions. Our current focus is on whether any limitations of our first-order approach arise when we attempt to automate the reasoning concerning Leibniz's theory of concepts (Zalta, 2000). This involves far more complexity in terms of nested definite descriptions and λ-expressions. We discuss this issue further in Section 5.2.

5.1 MACE

PROVER9 and OTTER come with an especially powerful tool, MACE, which allows one to find finite models of arbitrary finite sets of first-order formulas (McCune, 2003a). Once PROVER9 finds a proof of a theorem of computational metaphysics, we then use MACE to find a model of the premises alone, to ensure that the premises used in PROVER9's proof are consistent. And whenever we encounter a proposition that is supposed to be a theorem of axiomatic metaphysics but for which PROVER9 can't find a proof, we then use MACE to see whether there is a countermodel, i.e., a model of both the premises and the negation of the conclusion. Indeed, MACE led us to discover that one of the propositions alleged to be a theorem in (Pelletier and Zalta, 2000) is not in fact such.

Consider 'Theorem 4' in (Pelletier and Zalta, 2000). It is stated in terms of the following definitions:

$$F \Rightarrow G \ =_{df} \ \Box\forall x(Fx \rightarrow Gx).$$

[7] Despite the fact that second-order logic is undecidable, it turns out that true automated theorem proving (and not mere verification) for second-order logic is possible (in principle). That is, the methods of resolution and unification can be extended to second-order logic (Pietrzykowski, 1973). Interestingly, however, these methods cannot be extended to third-order logic (Huet, 1973). See (Kohlhase, 1998) for a recent survey of theorem proving techniques for higher-order logical systems. Be that as it may, it is not clear to us whether existing higher-order theorem-proving systems would be more effective for the present applications. Moreover, as far as we know, there are no (general purpose) higher-order model finding programs. Thus, moving to a higher-order framework would mean forfeiting our ability to automatically find models for the premises of the theorems we prove.

$$\textit{The Form of } F \text{ (`}\Phi_F\text{')} =_{df} \imath x(A!x \,\&\, \forall G(xG \equiv F \Rightarrow G))$$

$$\textit{Participates}_{\text{PH}}(x, y) =_{df} \exists F(y = \Phi_F \,\&\, xF)$$

Now the proposition alleged to be Theorem 4 was:

$$xF \equiv \textit{Participates}_{\text{PH}}(x, \Phi_F)$$

After translating these definitions and claims into PROVER9 syntax, we found that PROVER9 didn't seem to be able to find a proof. We therefore used MACE to check for countermodels to both directions of the biconditional.

Indeed, the right-to-left direction of the biconditional has a countermodel. We therefore take the opportunity here to correct the error in the earlier paper, by briefly describing the countermodel. In what follows, we shall identify object b and property P such that:

$$\textit{Participates}_{\text{PH}}(b, \Phi_P) \,\&\, \neg bP$$

To form the countermodel, choose P to be the necessarily empty property being-Q-and-not-Q (for some arbitrary property Q) and consider a second, distinct necessarily empty property T, say being-round-and-square. That is, let $P = [\lambda z\ Qz \,\&\, \neg Qz]$ and let $T = [\lambda z\ Rz \,\&\, Sz]$. Note that in object theory, one may consistently assert that $P \neq T$ even though $\Box\forall x(Px \equiv Tx)$. The reason is that identity for properties (`$F = G$') is defined as $\Box\forall x(xF \equiv xG)$. So properties may be distinct even though necessarily equivalent.

Now consider Φ_P and Φ_T. Even though $P \neq T$, it is provable in object theory that $\Phi_P = \Phi_T$. To see this, note the following theorems of quantified modal logic, namely, that that necessarily empty properties are necessarily equivalent and that necessarily equivalent properties entail the same properties, i.e.,[8]

[8] Here is a simple, semantic-based sketch of the former, which relies on the fact that the modal logic of object theory is the simplest possible: fixed domains, S5 with Barcan formulas, and no accessibility relation. Assume F and G are necessarily empty. It follows immediately that they are necessarily equivalent, i.e., that $\Box\forall x(Fx \equiv Gx)$. If neither F nor G are exemplified by any objects at any possible world, then F and G are exemplified by all and only the same objects at every possible world.

Here is now a semantic-based proof of the latter. Assume F and G are necessarily equivalent. Now, for the left-to-right direction, assume that for an arbitrary property H, $F \Rightarrow H$, i.e., that $\Box\forall x(Fx \rightarrow Hx)$. If we can show $G \Rightarrow H$, we are done (without loss of generality, since the proof of the right-to-left direction goes exactly the same way). To show $\Box\forall x(Gx \rightarrow Hx)$, we first prove the embedded, non-modal universal claim holds at an arbitrary world, say w. So, for an arbitrary object, say c, suppose Gc at w. Then since G and F are necessarily equivalent, they are equivalent at w,

$$[\Box\forall y\neg Fy \,\&\, \Box\forall y\neg Gy] \rightarrow \Box\forall x(Fx \equiv Gx)$$

$$\Box\forall x(Fx \equiv Gx) \rightarrow \forall H(F \Rightarrow H \equiv G \Rightarrow H).$$

So, we can infer from these two theorems that necessarily empty properties P and T entail the same properties. But, by definition, Φ_P encodes all and only the properties entailed by P, and Φ_T encodes all and only the properties entailed by T. Since P and T entail the same properties, Φ_P and Φ_T encode the same properties. Thus, $\Phi_P = \Phi_T$, by the definition of identity for abstract objects.

Now to complete the countermodel, let b be the abstract object that encodes exactly one property, namely, T. We can establish (i) $\textit{Participates}_{\text{PH}}(b,\Phi_P)$, and (ii) $\neg bP$. To establish (i), we have to show:

$$\exists F(\Phi_P = \Phi_F \,\&\, bF)$$

But this follows by Existential Generalization after conjoining the facts that $\Phi_P = \Phi_T$ and bT. To show (ii), note that by definition, b encodes only a single property, namely, T. Since $P \neq T$, it follows that $\neg bP$.

So, we've identified objects, properties, and Forms that constitute a counterexample to the 'theorem', since we have established:

$$\textit{Participates}_{\text{PH}}(b, \Phi_P) \,\&\, \neg bP$$

Thus, Theorem 4 in (Pelletier and Zalta, 2000) should have been weakened to:

Theorem 4: $xF \rightarrow \textit{Participates}_{\text{PH}}(x, \Phi_F)$

Our research in computational metaphysics shows, however, that the other nine theorems in (Pelletier and Zalta, 2000) are correct.[9]

5.2 Future Research Questions

Despite the above successes and reasons for using PROVER9, our current research suggests that new techniques might be required when the complexity of the theorems increases. We are currently working on the question, how should we automate the proofs of the theorems reported in (Zalta, 2000), concerning Leibniz's theory of concepts? And after that, we want to find proofs of theorems reported in (Zalta, 1999), concerning Frege's theory of the natural numbers in the *Grundgesetze*.

so it follows that Fc at w. But F necessarily implies H, and so materially implies H at w. Thus, Hc. So, we've proved, with respect to an arbitrary object c and world w, that $Gc \rightarrow Hc$ at w. Thus, by universal generalization on c and then w, we've established $\Box\forall x(Gx \rightarrow Hx)$, i.e., $G \Rightarrow H$.

[9] See <`http://mally.stanford.edu/cm/`> or <`http://fitelson.org/cm/`>.

Our research suggests that the representation of object theory's second-order language used so far may face certain difficulties when more complex applications are considered. We conclude by describing one such issue which has come up during the course of our research and which suggests that more sophistication may be needed.

In object theory, the Leibnizian notion *the concept of* G ('c_G') is defined in exactly the same way as the Platonic Form of G (Φ_G) is defined, namely, as (Zalta, 2000):

$$c_G = \imath x(A!x \,\&\, \forall F(xF \equiv G \Rightarrow F))$$

By analogy with our earlier discussion of defining The Form of F, we would represent The Concept G in PROVER9 in terms of the following two predicates:

```
all z all G ((Object(z) & Property(G)) ->
  (IsAConceptOf(z,G) <->
     (Ex1(A,z,W) &
        (all F (Property(F) -> (Enc(z,F) <-> Implies(G,F)))))))).

all z all G ((Object(z) & Property(G)) ->
  (IsTheConceptOf(z,G) <->
     (IsAConceptOf(z,G) &
        (all y ((Object(y) & IsAConceptOf(y,G)) -> y=z))))).
```

In the usual way, the second definition tells us that an object z is *the* concept of G whenever z is a concept of G and any other concept of G is identical to z.

Now unlike the work we did with Platonic Forms, the theorems for Leibnizian concepts involve other abstract objects defined in terms of complex definite descriptions. For example, such notions as $x \oplus y$ (the sum of the concepts x and y) and $c_{F \vee G}$ (the disjunctive concept of F and G) are introduced.[10] These notions would also be defined in PROVER9's language in two steps, yielding the terms `IsTheSumOf(z,x,y)` and `IsTheDisjConceptOf(z,G,H)`.

Now Theorem 4 of the Leibnizian theory of concepts is:

$$\forall G, H(c_G \oplus c_H \;=\; c_{G \vee H})$$

i.e., in PROVER9's language:

[10] In (Zalta, 2000), the first is explicitly defined as:

$x \oplus y =_{df} \imath z(A!z \,\&\, \forall F(zF \equiv xF \vee yF))$

Though the second is not explicitly defined, the definiens in the following definition shows up in the statement of Theorem 4 of that paper:

$c_{G \vee H} \;=_{df}\; \imath x(A!x \,\&\, \forall F(xF \equiv G \Rightarrow F \vee H \Rightarrow F))$

In the above definitions, we've used '$A!x$' (x is abstract) since it is more familiar to the readers of this paper; in (Zalta, 2000), we simply identified *Concept*(x) as the same notion and used that instead.


```
all G all H all x all y all z all u
((Property(G) & Property(H) & Object(x) & Object(y) & Object(z) &
Object(u) & IsTheConceptOf(x,G) & IsTheConceptOf(y,H) &
IsTheSumOf(z,x,y) & IsTheDisjConceptOf(u,G,H)) -> z=u).
```

As one can imagine, there is a great deal of quantifer depth and clausal complexity involved in representing all the requisite definite descriptions and proving Theorem 4 in PROVER9. The automated reasoning engine populates the search space with a large number of clauses and this space grows much larger as PROVER9 executes its main loop.

We haven't yet been able to prove Theorem 4 in PROVER9. At this point, we are unsure whether our failure here is due only to issues of computational complexity or to a more fundamental shortcoming of our first-order representation of claims with this depth of embedding.

If this problem proves to be intractable for first-order methods and systems, we may move to a higher-order automated reasoning system in our future work (e.g., Leibniz's theory of concepts and Frege's theory of numbers). We hope to report on any results using such higher-order reasoning systems, or further results in our current first-order framework, should they become available.

References

Boyer, R. and J. Moore: 1979, *A computational logic*. New York: Academic Press [Harcourt Brace Jovanovich]. ACM Monograph Series.

Huet, G.: 1973, 'The undecidability of unification in third order logic'. *Information and Control* **22**(3), 257–267.

Kalman, J.: 2001, *Automated Reasoning with Otter*. Princeton, N.J.: Rinton Press.

Kohlhase, M.: 1998, 'Higher-order automated theorem proving'. In: *Automated deduction—a basis for applications, Vol. I*, Vol. 8 of *Applied Logic Series*. Dordrecht: Kluwer Academic Publishers, pp. 431–462.

Kowalski, R.: 1970, 'The case for using equality axioms in automatic demonstration'. In: *Symposium on Automatic Demonstration (Versailles, 1968)*, Lecture Notes in Mathematics, Vol. 125. Berlin: Springer, pp. 112–127.

Leibniz, G.: 1890, 'untitled'. In: C. Gerhardt (ed.): *Die philosophischen Schriften von Gottfried Wilhelm Leibniz*, Vol. vii. Berlin: Olms.

Linsky, B. and E. Zalta: 1994, 'In Defense of the Simplest Quantified Modal Logic'. *Philosophical Perspectives* **8**, 189–211.

Mally, E.: 1912, *Gegenstandstheoretische Grundlagen der Logik und Logistik*. Leipzig: Barth.

Manzano, M.: 1996, *Extensions of first order logic*, Vol. 19 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge: Cambridge University Press.

McCune, W.: 2003a, 'Mace4 Reference Manual and Guide'. Technical Memorandum 264, Argonne National Laboratory, Argonne, IL. URL = <`http://www-unix.mcs.anl.gov/AR/mace4/July-2005/doc/mace4.pdf`>.

McCune, W.: 2003b, 'Otter 3.3 Reference Manual'. Technical Memorandum 263, Argonne National Laboratory, Argonne, IL. URL = <http://www.mcs.anl.gov/AR/otter/otter33.pdf>.

McCune, W.: 2006, 'Prover9 Manual'. Technical report, Argonne National Laboratory. URL = <http://www-unix.mcs.anl.gov/~mccune/prover9/manual/>.

Pelletier, F. and E. Zalta: 2000, 'How to Say Goodbye to the Third Man'. *Nous* **34**(2), 165–202.

Pietrzykowski, T.: 1973, 'A complete mechanization of second-order logic'. *Journal of the Assocatlon for Computing Machinery* **20**(2), 333–365.

Portoraro, F.: Winter 2005, 'Automated Reasoning'. In: E. N. Zalta (ed.): *The Stanford Encyclopedia of Philosophy*.

Robinson, G. and L. Wos: 1969, 'Paramodulation and theorem-proving in first-order theories with equality'. In: *Machine Intelligence, 4*. American Elsevier, New York, pp. 135–150.

Robinson, J. A.: 1963, 'Theorem-proving on the computer'. *Journal of the Association of Computing Machinery* **10**, 163–174.

Robinson, J. A.: 1965, 'Automatic deduction with hyper-resolution'. *International Journal of Computer Mathematics* **1**, 227–234.

Russell, B.: 1900, *A Critical Exposition of the Philosophy of Leibniz*. Cambridge: Cambridge University Press.

Spinoza, B.: 1677, *Ethics*. Edited and translated by G.H.R Parkinson, Oxford: Oxford University Press, 2000.

Wos, L., R. Overbeek, E. Lusk, and J. Boyle: 1992, *Automated Reasoning: Introduction and Applications,* 2nd edition. New York: McGraw-Hill.

Wos, L., G. Robinson, D. Carson, and L. Shalla: 1967, 'The concept of demodulation in theorem proving'. *Journal of the ACM* **14**(4), 698–709.

Zalta, E.: 1983, *Abstract Objects: An Introduction to Axiomatic Metaphysics*. Dordrecht: D. Reidel Publishing Company.

Zalta, E.: 1993, 'Twenty-five basic theorems in situation and world theory'. *Journal of Philosophical Logic* **22**(4), 385–428.

Zalta, E.: 1999, 'Natural numbers and natural cardinals as abstract objects: a partial reconstruction of Frege's *Grundgesetze* in object theory'. *Journal of Philosophical Logic* **28**(6), 619–660.

Zalta, E.: 2000, 'A (Leibnizian) Theory of Concepts'. *Philosophiegeschichte und logische Analyse/Logical Analysis and History of Philosophy* **3**, 137–183.