From QED to X-Logic

following the lead of Leibniz

Roger Bishop Jones
rbj@rbjones.com

2007/06/12

Abstract

Was the QED project for the formalisation of mathematics too ambitious? Leibniz, whose conception of a universal language and calculus of reasoning preceded QED by 300 years, would have thought it faint-hearted, the two most difficult problems in realising his grander vision having been resolved by the revolution in modern logic and the invention of the electronic digital computer.

"X-Logic" is a contemporary interpretation of Leibniz's project. Like the original, it is a multi-disciplinary enterprise, straddling the boundaries between philosophy, logic, computing, and science.

In this talk after introducing Leibniz and his project, I will enumerate some philosophical principles endemic to the Cambridge Automated Reasoning Group (implicit in the methods and/or explicit in some of the publications of the ARG). Some consequences of these philosophical principles for an architecture for globally distributed, linguistically pluralistic, automated deduction will then be considered, making use of a formal model in Isabelle-HOL.

At this level inference takes place between documents, which are understood as complex propositions, and may be in any language which has a well-defined abstract semantics. The notion of proof is generalised to encompass all correct computation relative to program specifications expressed in terms of the languages of documents. Assurance of the truth of propositions thus proven is managed by the qualification of judgements with a list of those authorities upon whose infallibility certainty in the content of the judgement rests.

A central concern is the articulation of a notion of soundness appropriate to this context. A manner of resolving the foundational problem of semantic regress, and that of a final resort on matters of consistency. will be touched upon.

Last Change Date: 2007/06/17 13:16:27
http://www.rbjones.com/rbjpub/pp/doc/tp002a.pdf
Id: tp002.doc,v 1.4 2007/06/17 13:16:27 rbj01 Exp

Outline

Contents

1 Introduction

1.1 Leibniz and His Project

Leibniz (1646 -1716)

- .. an intellectual innovator of the first rank, his work included:
  - A philosophical system which was "unusually coherent and complete". (Russell)
    Mostly in scattered fragments not discovered for centuries and hence not very influential.
  - The integral and differential calculus.
    This was probably the most influential of Leibniz's innovations. Newton first discovered the calculus, but Leibniz discovered it independently and published it first (1684), and his notation proved much more convenient so that the subsequent development of the mathematics took place primarily in continental Europe using his notation. Failure to adopt Leibniz's notation marginalised English mathematics for some time.
  - A calculating machine.
    Calculators capable of addition and subtraction had already been developed before Leibniz. Leibniz designed a calculator to do multiplication and division, and made innovations which later became standard, for example, the stepped reckoner (or "Leibniz wheel"), which had cogs of varying lengths. He presented a model of one of his calculators to the Royal Society on a visit to London in 1673.
  - Many grand and unrealistic schemes, including one for the automation of reason.
- He was "an encyclopedic savant whose philosophy was nourished by the study of all the sciences and in turn inspired all of his scientific discoveries" (Couturat)
- "Mathematicians have as much need to be philosophers as philosophers have to be mathematicians." (Leibniz)
- Inspired:
  - Frege's *Begriffsschrift*,
  - Russell's work on *Principia* and other aspects of his philosophy
  - (indirectly) Carnap's philosophical programme, including the formalisation of physics.

Leibniz's Grand Projects

- Universal Mathematics

> Since happiness consists in peace of mind, and since durable peace of mind depends on the confidence we have in the future, and since that confidence is based on the science we should have of the nature of God and the soul, it follows that science is necessary for true happiness.
>
> But science depends on demonstration, and the discovery of demonstrations by a certain Method is not known to everybody. For while every man is able to judge a demonstration (it would not deserve this name if all those who consider it attentively were not convinced and persuaded by it), nevertheless not every man is able to discover demonstrations on his own initiative, nor to present them distinctly once they are discovered, if he lacks leisure or method.
>
> The true Method taken in all of its scope is to my mind a thing hitherto quite unknown, and has not been practised except in mathematics. It is even very imperfect in regard to mathematics itself, as I have had the good fortune to reveal by means of surprising proofs to some of those considered to be among the best mathematicians of the century. And I expect to offer some samples of it, which perhaps will not be considered unworthy of posterity.

[From: The Method of Mathematics - preface to the General Science, G.W. Leibniz]

- The General Science
- Universal Encyclopedia

  Leibniz sought the collaborative development of an encyclopedia in which would be presented in non-symbolic form all that was so far known. This was to provide a basis for the lingua characteristica in which the knowledge could be formally expressed. This enterprise was not completed, but beneficial side effects were the foundation of new academies and of the journal Acta Eruditorum.
- Universal Language

  he promoted the development of a language universal in the sense of being spoken by all. There have been many such projects of which the best known today is Esperanto.
- Academies and Journals

  Leibniz was involved in many schemes for setting up academies. His main efforts were for academies in Berlin, Dresden, Vienna and St Petersburg, but others included Mainz, Hanover, Hamburg and Poland.

  He was involved in the founding pf the journal Acta Eruditorum in 1682 and much later, in 1700, he started his own journal, the Monatliche Auszug.
- Universal Characteristic and Calculus Ratiocinator
  - all knowledge formally expressible using the universal characteristic
  - all disputes resolvable by computation using the calculus ratiocinator

  The *universal characteristic* or *lingua characteristica* was to be a language in which predicates were numerically encoded in such a way as to render the truth of subject predicate proposition (and Leibniz considered all propositions to have this form) could be obtained by arithmetical computation.

  The *calculus ratiocinator* was to be a calculus which permitted the truth of any sentence in the lingua characteristica to be computed (possibly by a machine). Leibniz believed that every predicate was either simple or complex and that complex predicates could be analysed into simple constituents. He proposed to assign prime numbers to each simple predicate and then represent a complex predicate by the product of the primes representing its simple constituents (complex predicates are therefore invariably conjunctions of finite numbers of simple predicates). He also believed that every proposition had subject predicate form, and that in a true proposition the predicate was contained in the subject, i.e. the set of simple predicates from which the predicate was composed was a subset of the set from which the subject was composed. This can be sorted out by numerical computation, you just check whether the predicate divides the subject without remainder.

  His main difficulty in this was in discovering what the simple predicates are. Leibniz thought the complete analysis beyond mere mortals, but believed that a sufficient analysis (into predicates which are relatively primitive) for the purposes of the calculus ratiocinator would be realisable. From this it appears that Leibniz expected his language and calculus to be not completely universal.
- Should we take this (characteristic and calculus) seriously?
  1. as an alternative to "the singularity"

     This is an idea of Vernor Vinge, one among many along the lines that global AI will emerge outside our control and take over.

     > "Within thirty years, we will have the technological means to create superhuman intelligence. Shortly after, the human era will be ended."
  2. to establish the domain in which machines can be authoritative and make sure that they are reliable where they can be
  3. as promoting systematisation and simplification

     In academic research innovation is prized over mere reorganisation. This has many detrimental effects.
  4. ...

1.2 Some Philosophical Context

"ARG philosophy"

Some pragmatic philosophical positions which appear to be endemic to the Cambridge Automated Reasoning Group.

1. Pragmatic Foundationalism

    - There are good practical reasons for doing formal deduction in the context of a "foundation system" (Gordon, Paulson ...)

      (i.e. a system in which one can work by *conservative extension*).

      The foundational culture in ARG/HVG begins with the adoption by Mike Gordon of HOL for hardware verification. There are probably many expositions of this kind of idea in the publications of the ARG, e.g. Paulson on datatypes.
    - Getting deductive inference right is not enough, need to ensure that the context is coherent.

2. Deduction alone tells us nothing about anything physical (Fetzer!), ...

    See Avra Cohn on the verification of the Viper processor [**?**]. She anticipates the criticism that formal proofs about the behaviour of hardware are impossible and explains that the verification of the microprocessor involves reasoning about abstract models of a processor which capture the design of the processor. This may establish that the design is correct, but not that any physical realisation of the design will perform correctly.

    Fetzer's "Program Verification: The Very Idea" was recently published [**?**]. Fetzer, in his paper makes the mistake of supposing that people in program verification are not aware of the analytic/synthetic distinction, and think they can logically prove synthetic truths. He was probably incorrect in this, Many, possibly most, were and are aware of the distinction, but do not (as Fetzer did) regard a program verification as making any claim about the behaviour of computers. We all know that computer hardware can fail, and that even a correct program will not elaborate correctly on a broken computer.

    In this matter, "the ARG" exhibits an awareness of a crucial and pretty definite distinction between different kinds of statement or proposition which is significant for their methods, and has a story about how deductive reasoning relating to synthetic or contingent claims can be conduction entirely through analytic or necessary propositions (not expressed in these terms).

    but we can reason about physical reality using abstract models (Cohn).

    The methodological impact of the analytic/synthetic distinction is small. It does not prevent deductive reasoning about contingent matters. It just requires that in order to do this you have to formulate an abstract model of the system and then reason about that model. There is then a non-deductive step required to get from any deductively established conclusion about the model to a conclusion about the system modelled.

    This means that, even when reasoning about physical systems, insofar as the reasoning is deductive it can be conducted in a foundation system using only conservative extension. This is pragmatically useful because it solves the problem of establishing the coherence of the context in which reasoning takes place.

3. To get your proof system sound its a good idea to have a semantics.

    Very early versions of HOL had insufficient checks on polymorphic definitions to ensure that they are conservative. After discovering this it was decided that a mathematical semantics would be a good idea.

    A natural place to give the semantics of a foundation system is in set theory (e.g. Pitts and Arthan),

    The semantics of HOL in set theory written by Andy Pitts is now (long past) part of the HOL documentation. A formal semantics was done by Rob Arthan using ProofPower-HOL.

    Hence:

    - For establishing deductive soundness, abstract semantics suffices...
    - even if we want to reason about the real world.

4. A correct computation is as good as a proof (LCF paradigm?).

   I argue that this is implicit in the LCF paradigm though it sounds mode liberal than the LCF paradigm. As far as the nature of proof is concerned the LCF paradigm (it seems to me) embodies two principles. The first is that if we have a program which is known to compute only theorems, then an evaluation of that program is as good as a proof for most purposes. The second, which is more clearly associated with the LCF paradigm is that if theoremhood is encapsulated in an abstract data type in a strongly typed functional programming language, and is implemented with constructors which correctly implement derived rules of the object logic, then any program which computes theorems will be sound. Of course, this latter principle doesn't legitimate an LCF proof tool without the first principle, which could also be used in the justification of systems (such as are modelled below) in which soundness is established by other means. (soundness, i.e. semantic entailment, is a more liberal constraint than derivability, at least in some logics, but it is sufficient).

5. Interactive tools get you further (than plain automation), ... and we have ways of making them safe (LCF paradigm)

1.3 Interpreting Leibniz's Project Today

Why Should YOU Care (about Leibniz's project)?

Meaning here, by "YOU", specifically the members of the Cambridge ARG. (other groups may have quite different reasons to care).

Think:

- Deductive Design Automation (instead of EDA, etc.)
- Computer Aided Deductive Design (instead of CADE)

These are economic motivators.

Our contemporary architecture for the Leibniz Project (let's call it X-Logic) seeks to reduce to a bare minimum the cost of "Deductive Rigour".

- This is done by:
  1. Assimilating proof and computation.
  2. Separating rigour and assurance.
  3. Placing no lower bound on assurance.
- "Deductive Rigour" requires (of computation as proof):
  1. Every program has a formal specification (with which it complies!).
  2. Programs are only relied upon to meet their specification.
- and admits:
  - any correct program as an inference rule, with
  - assurance of correctness of conclusion no worse than assurance of correctness of program

We liberalise the notion of proof to encompass the computations of any correct program. If we code up a procedure which for some nice class of design problems takes a specification of a particular problem and computes a design of a system meeting that specification, then that program if correct will implement a sound inference rule, and can be used directly in Deductive Design Automation.

For a class of problems too hard to crack automatically, we could implement a CADD tool in which the problem is solved with human help, but in a manner which is safe (as in LCF) from delivering a non-conforment design as a result of human error. Human assistance might only be used in optimising the output, or might be required in establishing correctness (by something like proof).

How would we assure ourselves that such tools are correct? The obvious answer would be to verify them. This is hard work, and for parity of assurance with LCF proof tools may not be necessary. We should instead discard the "specify/implement/verify" minset in favour of a "correct by construction" mindset. In this latter, the program would be obtained by "compilation" of the specification, and would be known correct because the program doing the construction/compilation is sound. Again, we could allow human assistance (or a machine assisted and checked human construction).

Universal Formal Language

Is a Universal Formal Language possible or desirable?

- In Principle:
  - Concerned only with domains in which deductive reasoning is possible.

    This would probably rule out most poetry and much else, but it still leaves a great deal
  - Any domain can be covered using abstract models
  - Hence, need language universal only for abstract modelling.
  - Set theory is therefore a contender.

    ... but "semantic regress" is a problem.

    Could a set theory be universal for abstract modelling?

    Doubt arises from Tarski's result on undefinability (of arithmetic truth in itself), and his view that one can give the semantics of a language only in a language which is stronger in some appropriate sense. This leads us to the idea that the best we can get is an infinite hierarchy of (possibly similar) languages of progressively increasing strength. This, even if it were the best we could do, would no be so bad. You have a single syntax for set theory, but its semantics and proof rules may need to be strengthened from time to time.

    However, Tarski's result has not been rigorously proven in a wide enough context, and we offer below a conception of the semantics of set theory which probably renders it universal. Either a positive or a negative result here depends upon careful definition of the notion of universality involved, and there may be no single definition which is obviously the right one to use.
- In Practice:
  - Pluralism desirable

    It seems now to be highly desirable to seek a pluralistic architecture in which any language satisfying some minimal requirement (e.g. having a well defined syntax and semantics) will be supported.
  - The development of new material (i.e. adding definitions) may constitute the definition of a new language.

    It may be desirable to assimilate logical context into language so that the development of a theory constitutes the definition of a new language. This is consistent with the notion of language used in mathematical logic.
  - Single extensible language for Mathematics.

    Nevertheless, in the development of mathematics it seems natural to have a single language which is extensible by defining new constants and supplying appropriate syntactic information on how applications of the constants should be presented.

    There are interesting philosophical problems here. It is commonly doubted that there could be a universal language, even for abstract semantics. However, the concepts involved have not been sufficiently precisely defined for the question to have a definite answer (what is a language, what is a semantics, what does it mean to say that the semantics of one language is definable in another). This is not sophistical hair splitting, its a real issue. These issues connect with the practical question of how a logical system revolving around semantics resolves the problem of semantic regress. For one answer, see below under semantics.
  - "Semantic regress" isn't a problem.
- Coherent interworking in a pluralistic context is realisable through common semantics foundations.

Proof and Computation

Calculus Ratiocinator

- We have universal calculating machines.

- We know there is no decision procedure, but ...

  ... we do have near complete formal deductive systems.

- Computational feasibility limits the range of soluble problems,

  ... but formalisation is worthwhile and realisable,

  ... with automation sufficient to facilitate formal derivation of theory.

Feasible Goal

- An architecture supporting distributed co-operative large scale formalisation,
- Existing proof tools easily incorporated into the architecture.
- ... enabling continued development of automated deductive problem solving.

2 X-Logic Architecture

2.1 What is Architecture?

What is Architecture?

An architecture should supply:

- The most important high-level requirements
- A high level model of the system sufficient to expose...
- A rationale showing how or why the system will meet the requirements,

2.2 Requirements

Requirements

- Soundness
  - An architecture for automated deductive reasoning, must of course be *deductive*.
  - So what does that mean?

    A demonstration is *deductive* if it is accomplished exclusively by the use of *sound* rules of inference.
  - Therefore:
    - the system must be sound,
    - languages should have a well-defined *abstract* semantics.
- Functionality, Performance, Assurance
  - Flexible and known assurance.
  - Cost of rigour minimised.
  - Admit existing methods and tools.
  - Good for formal mathematics and formalisable science and engineering.
  - Economic driver / long term goal: The Automation of Design.

    The idea is that the user is able to chose where he wants to be in the functionality/performance/assuracnce trade-offs. If you don't need absolute certainty you can set the standard low and get results faster and cheaper. The important thing is that you can get high assurance when you need it and you know what you are getting.

2.3 A Formal Model

A Formal Architectural Model

This model addresses just some of the most important requirements.

These are key to the aim of "reducing the cost of rigour".

The model:

- is document oriented
- is multilingual
- admits inference by correct computation
- involves an assurance calculus

It is presented as the definition of an elementary top-level language in which proofs at lower levels can be sewn together while tracking the assurance level of the results.

2.3.1 Abstract Syntax

The abstract syntax of our metalanguage is defined using a datatype in isabelle HOL.

The language is about documents (which are understood as propositions) expressed in various object languages, and programs (whose computations are interpreted as inferences) which read documents and create new documents. These things have to be uniquely identified in a global context, and that's what URI's (Universal Resource Identifiers) are for, so that's what we call them. So we begin with a type abbreviation indicating that URI's are to be represented by strings, and that various other things are given by URI's.

A Model

$types\ URI = string$
$\quad document = URI$
$\quad language = URI$
$\quad program = URI$
$\quad authority = URI$

Here:

- URI stands for Universal Resource Identifier but may also include a digest of the resource referred to.
- A document may be any static representation of information which has (or can be given) a *propositional* semantics, e.g. an XML document, a snapshot of a view of a database.
- Think of a language as encompassing a context, and a document as potentially creating a new context (like a theory).
- Any program for which a specification is "known" counts as a sound inference rule.
- We do not seek absolute assurance, sets of authorities serve as assurance levels.

Sentences and Judgements

The subject matter of the metalanguage is the truth of documents. The metalanguage permits the establishment (proof) of truth to be compounded from inferences performed by a variety of programs in various languages. From premises about the inferences performed by these various programs (which may be thought of as demonstrating lemmas) it is to be possible in the metalanguage to infer an overall conclusion.

The language contains sentences which express the claims that:

1. certain documents are true documents of particular languages
2. a certain program satisfies a specification formulated as soundness with respect to given lists of input and output languages
3. a specified list of output documents was computed by a program from a list of input documents

```
datatype sentence =
        TrueDocs "document list" "language list"
|       ProgSpec program "language list" "language list"
|       Compute  program "document list" "document list"
```

In general sentences are not proven absolutely, but on the assurance of various authorities (sometimes called oracles). The combination of a sentence with a set of authorities which have contributed to our grounds for asserting the sentence is called a "judgement". For reasons connected with well-definedness of the semantics of judgements a judgement also contains a number. This may be thought of as a time-stamp, but is more loosely specified.

Judgements either assert sentences or endorse authorities:

```
types stamp = nat

datatype judgement =
    Assert stamp "authority set" sentence |
    Endorse authority "authority set"
```

2.3.2 Semantics

The set of authorities can be empty, but when asserted a judgement must be signed by an authority. The meaning of a judgement is that *if* all the authorities cited in the list have been hitherto infallible then the sentence is true.

However, the judgement is known only with that degree of confidence which we attach to the authority which asserts it (and has signed it), so even an unconditional judgement (one with an empty set of cited authorities) is still no better assured than its signing authority.

An authority has been "hitherto infallible" if all the judgements which it has signed with numbers less than that of the judgement in hand are true. In fallibility and truth are therefore mutually defined, the numbers attached to judgements relativise infallibility so as to make the mutual recursion well-founded.

Sentence Interpretations

To interpret a sentence we need to know:

1. the content of the documents
2. the semantics of the languages
3. the functions computed by the programs

```
types
        document_map = "document ⇒ string"
        language_map = "language ⇒ string set"
        program_map  = "program ⇒ (string list ⇒ string list)"
```

```
record Sinterp =
  docmap :: document_map
  langmap:: language_map
  progmap:: program_map
```

True Sentences

```
consts
    truedoclist :: "[Sinterp, document list, language list] ⇒ bool"
primrec
   "truedoclist i (h_d#t_d) l_l = (case l_l of
      []        ⇒ False |
      (h_l#t_l) ⇒ (docmap i h_d) ∈ (langmap i h_l) & truedoclist i t_d t_l)"
   "truedoclist i [] l_l = (case l_l of [] ⇒ True | (h_l#t_l) ⇒ False)"
```

```
consts
   truesen :: "Sinterp ⇒ sentence ⇒ bool"

primrec
   "truesen i (TrueDocs dl ll)      = truedoclist i dl ll"
   "truesen i (ProgSpec p ill oll) = (∀ idl . truedoclist i idl ill
                                    --> truedoclist i (progmap i p idl) oll)"
   "truesen i (Compute p idl odl)  = (odl = progmap i p idl)"
```

Judgement Interpretations

- A judgement is true if infallibility of its authorities implies the truth of its sentence.
- To determine truth we therefore need to know what judgements have been asserted by each authority.

```
types
        judgement_map = "authority ⇒ judgement set"
```

```
record Jinterp =
   sinterp::Sinterp
   judgemap::judgement_map
```

Infallibility and Truth - I

Informally an authority is infallible if it only asserts true judgements. However, the definition of truth of a judgement will depend upon the infallibility of authorities, and this naive view does not lead to a well defined concept.

This is fixed by slightly *strengthening* the meaning of judgements, so that their truth depends only on the truth of *previous* judgements, and it is for this reason that judgements have been given a "stamp". This leads us to the property of being "hitherto infallible" at some stamp value. This is the property that all judgements affirmed by the authority with smaller stamp values are true.

One further complication is necessary, arising from endorsement. The infallibility of an authority is conditional on the infallibility of the authorities it has endorsed in a way which cannot be allowed for by attaching a truth value to the judgement in which the endorsement takes place. This is because the truth value of the endorsement can only depend on that of previous judgements, but the infallibility of an authority at some time depends on judgements made by authorities he has endorsed between the time at which the endorsement took place and the later time at which an infallibility judgement may be taking place.

Endorsements are therefore held to create a timeless partial ordering on authorities, and we require for the infallibility of an authority at some moment that neither he nor any greater authority has made a previous error. Greater in this case means directly or indirectly endorsed by the authority in question.

```
types
      inftest = "authority ⇒ Jinterp ⇒ bool"
      truthtest = "judgement ⇒ Jinterp ⇒ bool"


constdefs
  authrel :: "judgement_map ⇒ (authority ∗ authority)set"
  "authrel jm == rtrancl {p. ∃as. snd p ∈ as
                          & Endorse (fst p) as ∈ jm (fst p)}"

  basett :: "truthtest"
  "basett j ji == case j of (Endorse a as) ⇒ True |
                            (Assert n as s) ⇒ truesen (sinterp ji) s"


  itrec :: "nat ⇒ (inftest ∗ truthtest) ⇒ inftest"
  "itrec n tsts auth ji == ∀a. (auth, a) ∈ authrel (judgemap ji)
      --> (∀j. j ∈ judgemap ji a & jstamp j < n --> snd tsts j ji)"

  ttrec :: "(inftest ∗ truthtest) ⇒ truthtest"
  "ttrec tsts j ji == (∀auth. auth ∈ (jauths j) --> (fst tsts) auth ji)
          --> basett j ji"
```

Infallibility and Truth - II

```
consts
  ittt :: "nat ⇒ (inftest ∗ truthtest)"

primrec
  "ittt 0       = ((λx y z. True), basett)"
  "ittt (Suc n) = (itrec (Suc n) (ittt n), ttrec (ittt n))"


constdefs
  hitherto_infallible :: "nat ⇒ authority ⇒ Jinterp ⇒ bool"
  "hitherto_infallible n == fst (ittt n)"

  true_judgement :: "judgement ⇒ Jinterp ⇒ bool"
  "true_judgement j == snd (ittt (jstamp j)) j"
```

Critical Requirements Formalised

Soundness

Inference preserves truth.

```
constdefs
  sound ::"(judgement set ⇒ judgement set) ⇒ bool"
  "sound f == ∀ji js. (∀j. j:js --> true_judgement j ji)
          --> (∀j. j:(f js) --> true_judgement j ji)"
```

Assurance

Lowly assured premises neither yield not influence highly assured conclusions.

```
constdefs
  filter ::"authority  set ⇒ judgement set ⇒ judgement set"
  "filter as js == {j . jauths j ⊆ Image (authr js) as}"

  assured ::"(judgement set ⇒ judgement set) ⇒ bool"
  "assured f == ∀js1 js2 as. filter as js1 = filter as js2 -->
                        filter as (f js1) = filter as (f js2)"
```

Outstanding Problems

This model addresses only a very few of the critical requirememts, and more modelling is needed in the following areas:

- The problem of consistency.
- Semantics and semantic definitions.
- Context
- Integrity

3 Foundations

Foundations

- Semantics and consistency are crucial.
- Abstract semantics suffices for deductive soundness.
- If we require each language to have a formal abstract semantics, then there will be a problem of semantic regress.
- Hence we need foundations for abstract semantics, where the buck stops.
- Well-founded sets provide the most plausible universal foundation.
- This can also serve as the last resort on questions of consistency.
- Does not provide the best language in which to do mathematics.

A Semantic Stack

Illative Lambda Calculus / Type Theory
(more flexible, better structuring)

Non-well-founded ontology
(incorporating the full well-founded ontology)

Well-founded ontology

Well-Founded Set Theoretic Truth

Here is a concise and definite semantics for well-founded Set theory:

- a well-founded set is a definite collection of well-founded sets
- an interpretation of set theory is a transitive well-founded set
- a sentence is false if the collection of interpretations in which it is true is definite
- a sentence is true if the collection of interpretations in which it is false is definite

This semantics:

- is maximally rich (large cardinal axioms are true)
- is definite (CH has a truth value, particular facts about cardinal arithmetic have truth values)
- makes ZFC neither true nor false
- is not limited to first order languages (will work for infinitary set theory)
- is self defining (conjecture)

Consistency:

- consistent "monotone" sentences are true

  A sentence S is consistent if it has a model M, it is monotone if for any interpretations M, N with M a subset of N then if M satisfies S so does N.

- this generalises the idea that consistent large cardinal principles are true

  It includes all large cardinal axioms suitably formulated, so that the do not assert closure properties of the universe, only the existence of large cardinals. Think of these as placing a lower bound on the height of the universe. The generalisation is that this principle also gives width as well as height. So, for example, it fixes the truth of CH, though we don't know what value it gets fixed at.

The End

Presentation slides may be downloaded from:

http://www.rbjones.com/rbjpub/pp/doc/tp002b.pdf

Presentation slides and notes may be downloaded from:

http://www.rbjones.com/rbjpub/pp/doc/tp002a.pdf