X-Logic Models

Roger Bishop Jones

Abstract

Formal models of various aspects of X-Logic in Z

Created 2005/04/09

Last Change Date: 2010/05/26 19:20:05

http://www.rbjones.com/rbjpub/pp/doc/t016.pdf

Id: t016.doc,v 1.6 2010/05/26 19:20:05 rbj Exp

Contents

1 Introduction

This document is a partial conversion to use ProofPower of a document which was originally done with Isabelle. Now being massaged into something different.

2 Authorities

2.1 Introduction

From a computer science interactive theorem proving perspective authorities are an elaboration of a certain approach to tracking the use of oracles in interactive proofs. In that context the idea is that a tool which is good at checking proofs but not so very hot at constructing them, might invoke some other piece of software to decide upon the truth of some conjecture (typically by use of some special decision procedure) and would then be willing to admit that conjecture into a proof without having seen a proof of it.

Such invocation of an external "oracle" introduces a risk of unsoundness, and so it may be thought desirable to mark each theorem whose proof depends on an oracle so that this possibly lower standard of proof is understood. Where an interactive proof tool provides access to multiple such oracles. each oracle will have its own tag, and each theorem may be tagged with all the oracles on which its proof depends.

This tagging provides some indication of the relative confidence which one might attach to a theorem. Even if no comparative information is available about the reliability of an oracle, there is a natural ordering on the sets of tags which are attached to theorems. The more oracles used, the greater the risk, if we take 'more' here in the sense of set inclusion. This just amounts to saying that each extra oracle invoked introduces some extra risk.

One can imagine more refined methods of tagging, there is more information to be had. It may make a difference how many times each oracle is used, or on what conjectures. It is doubtful that such detail will prove sufficiently beneficial to justify the extra complexity involved.

What we consider here is a slight complication which we hope will be beneficial in extending the scope of this very simple way of marking confidence or risk. This is sufficient to justify a small theory of the resulting system.

Two extra features of the system are introduced:

1. As well as allowing that more than one oracle might be essentially involved in a proof (which we think of as a conjunction of authority levels), we admit that a proposition might have more than one proof and admit a corresponding disjunction of authority levels. This has the effect of creating a free distributive lattice of authority levels generated by any given set of authorities.

2. To simplify marking we allow authorities to be introduced as a kind of abbreviation of a complex authority level. This is done by allowing inequalities to be introduced into the lattice of authority levels. Only inequalities in which some authority expression is held to be at least as safe as some particular authority are admitted.

SML

```
open_theory "rbjmisc";
force_new_theory "X-Logic-Auth";
new_parent "tc";
set_merge_pcs["hol", "'savedthm_cs_∃_proof"];
```

Think of authority levels as represented by sets of sets of authorities, in which the whole is a disjunction or join and the parts are conjunctions or meets. However, I will actually use properties of properties.

SML

```
declare_type_abbrev("AL", ["'a"], ⌜:('a → BOOL) → BOOL⌝);
```

Over these we define the join and meet operations. There is an issue arising from a lack of uniqueness of the representatives relative to the intended meaning. There several ways of addressing this.

A further complication arises from the desire to have *endorsements* which supplement the partial ordering between the levels, and theredore make the partial ordering determined not just be the set of authorities but also by the set of endorsements.

HOL Constant

```
Join: 'a AL → 'a AL → 'a AL
```

$$\forall x\ y\bullet\ Join\ x\ y = \lambda p\bullet\ x\ p \vee y\ p$$

HOL Constant

```
Meet: 'a AL → 'a AL → 'a AL
```

$$\forall x\ y\bullet\ Meet\ x\ y = \lambda p\bullet\ \exists r\ s\bullet\ x\ r \wedge y\ s \wedge p = \lambda x\bullet\ r\ x \vee s\ x$$

SML

```
declare_alias("∪", ⌜Join⌝);
declare_alias("∩", ⌜Meet⌝);
```

$$\mathbf{Join\_id\_thm} = \vdash \forall\ x\ y\bullet\ x \cup x = x$$
$$\mathbf{Join\_com\_thm} = \vdash \forall\ x\ y\bullet\ x \cup y = y \cup x$$
$$\mathbf{Meet\_com\_thm} = \vdash \forall\ x\ y\bullet\ x \cap y = y \cap x$$
$$\mathbf{Join\_assoc\_thm1} = \vdash \forall\ x\ y\ z\bullet\ x \cup (y \cup z) = (y \cup x) \cup z$$
$$\mathbf{Join\_assoc\_thm2} = \vdash \forall\ x\ y\ z\bullet\ (x \cup y) \cup z = x \cup (y \cup z)$$
$$\mathbf{Meet\_assoc\_thm1} = \vdash \forall\ x\ y\ z\bullet\ x \cap (y \cap z) = (y \cap x) \cap z$$
$$\mathbf{Meet\_assoc\_thm2} = \vdash \forall\ x\ y\ z\bullet\ (x \cap y) \cap z = x \cap (y \cap z)$$
$$\mathbf{al\_distrib\_thm1} = \vdash \forall\ x\ y\ z\bullet\ x \cap (y \cup z) = (x \cap y) \cup (x \cap z)$$
$$\mathbf{al\_distrib\_thm2} = \vdash \forall\ x\ y\ z\bullet\ (x \cup y) \cap z = (x \cap z) \cup (y \cap z)$$

2.2 Endorsements

Authorities thus constituted do not have a satisfactory algebraic structure. Though they look a bit like lattices, the meet operation is not idempotent.

To make this work better we would have to restrict the authorities to a smaller set of representations and make the operators normalise their results, or else use equivalence classes of the present representative. In any case I want to impose on the structure further constraints, and this can be done in such a way as to improve the algebraic structure at the same time.

I therefore define equivalence classes generated by a set of "endorsements", each of which asserts that some singleton atomic assurance level is less than or equal to some other assurance level, beefing up the equivalence to ensure identification of each level with its self meet. This is done by defining a partial ordering on the levels.

HOL Constant

$$\boldsymbol{PathLt}: ('a \to BOOL) \to ('a \to BOOL) \to BOOL$$

$$\forall p\ q\bullet\ PathLt\ p\ q = \forall a\bullet\ q\ a \Rightarrow p\ a$$

HOL Constant

$$\boldsymbol{AuthLt}: 'a\ AL \to 'a\ AL \to BOOL$$

$$\forall a\ b\bullet\ AuthLt\ a\ b = \forall p\bullet\ a\ p \Rightarrow \exists q\bullet\ PathLt\ p\ q \wedge b\ q$$

HOL Constant

$$\boldsymbol{LevelLt}: (('a \times 'a\ AL) \to BOOL) \to ('a\ AL \to 'a\ AL \to BOOL)$$

$$\forall cl\bullet\ LevelLt\ cl = tc\ (\lambda l\ m\bullet\ l = m \vee AuthLt\ l\ m \vee \exists a\bullet\ cl\ (a, m) \wedge l = (\lambda p\bullet\ p = (\lambda q\bullet\ q = a)))$$

$$\boldsymbol{trans\_LevelLt\_thm} =$$
$$\vdash \forall\ es\bullet\ trans\ (LevelLt\ es)$$

$$\boldsymbol{trans\_LevelLt\_thm2} =$$
$$\vdash \forall\ es\ s\ t\ u\bullet\ LevelLt\ es\ s\ t \wedge LevelLt\ es\ t\ u \Rightarrow LevelLt\ es\ s\ u$$

Equivalence classes under this pre-order should yield a lattice of levels.

HOL Constant

$$\boldsymbol{LevelEquiv}: (('a \times 'a\ AL) \to BOOL) \to ('a\ AL \to 'a\ AL \to BOOL)$$

$$\forall cl\bullet\ LevelEquiv\ cl = \lambda a\ b\bullet\ LevelLt\ cl\ a\ b \wedge LevelLt\ cl\ b\ a$$

HOL Constant

$$\boldsymbol{Levels}: (('a \times 'a\ AL) \to BOOL) \to ('a\ AL)\ \mathbb{P}\ \mathbb{P}$$

$$\forall cl\bullet\ Levels\ cl = QuotientSet\ Universe\ (LevelEquiv\ cl)$$

In order to "lift" join and meet to the equivalence classes it is desirable to prove that as defined they "respect" level equivalence.

3 Model 1 - Languages Truth and Inference

3.1 Introduction

3.1.1 Purposes

The main ideas explored in this model are:

1. Logic at the document level - the propositions of this logic are complete documents.
2. Multilingual logic - the documents can be in any language which has a propositional semantics.
3. Specified computation as inference - any computation over documents by a program which meets a specification will be an inference in the logic.
4. Language as specification - the specification of a program for the purposes here is simply the identification of the input and output languages, the specification then means that whenever the program is supplied true documents in the input languages then if it outputs a document it will be a true document of the output language.

3.1.2 The Theory X-Logic-1

This version written in HOL as a child of rbjmisc.

SML

```
open_theory "rbjmisc";
force_new_theory "X-Logic-1";
set_merge_pcs["hol", "'savedthm_cs_∃_proof"];
```

3.1.3 Caveats

The caveats on metamodel 1 are not necessarily applicable here. The model is rewritten in Z to see whether Z might be better for present purposes. The caveats on metamodel 1 are a bit vaguely stated and so I'm not sure what I had in mind then. Its clear that a major defect is that inferences may have only one premise, and so I am fixing that problem, and in the process maybe I will remember whether there were other problems.

3.2 Types

An overview of the model with specifications of the various types of entity involved in it.

3.2.1 Domain of Discourse

The subject matter of the model is programs which perform transformations over documents, which are written in a variety of languages.

We take the set of documents to be unspecified, and hence use a given set. A language is modelled as a set of documents.

This should be understood as a purely *semantic* definition, in which the essential feature of a document is that it expresses an assertion, and the only matter of interest for our present purposes is whether that assertion is true.

Thus the set of documents which constitutes a language is the set of documents which when interpreted in that language are *true*.

A program is any partial function over the documents. A specification is a pair, the first element is a list of languages which are the languages of the inputs to the program and the second element is the language of the single output document.

3.2.2 Basic Types

The following definitions should be understood as introducing the semantic objects which correspond to the parts of a very simple language. Of these, the last three are kinds of "proposition" corresponding two three kinds of sentence which are available in our language. Informally they are the propositions respectively that a document in some language is true, that a program satisfies some specification, and that one document has been computed from some others by some program.

SML

$$
\begin{array}{l}
|\ new\_type(\texttt{"}DOC\texttt{"},0); \\
|\ declare\_type\_abbrev(\texttt{"}LANG\texttt{"},\ [],\ \ulcorner{:}DOC\ \mathbb{P}\urcorner); \\
|\ declare\_type\_abbrev(\texttt{"}PROG\texttt{"},\ [],\ \ulcorner{:}DOC\ LIST \rightarrow DOC\ LIST\urcorner); \\
|\ declare\_type\_abbrev(\texttt{"}SPEC\texttt{"},\ [],\ \ulcorner{:}LANG\ LIST \times LANG\ LIST\urcorner); \\
|\ declare\_type\_abbrev(\texttt{"}DOCPROP\texttt{"},\ [],\ \ulcorner{:}LANG \times DOC\urcorner); \\
|\ declare\_type\_abbrev(\texttt{"}PROGPROP\texttt{"},\ [],\ \ulcorner{:}SPEC \times PROG\urcorner); \\
|\ declare\_type\_abbrev(\texttt{"}INFERPROP\texttt{"},\ [],\ \ulcorner{:}PROG \times DOC\ LIST \times DOC\ LIST\urcorner);
\end{array}
$$

HOL Constant

$$
\begin{array}{l}
|\ (s\_inl\colon SPEC \rightarrow LANG\ LIST)\ (s\_outl\colon SPEC \rightarrow LANG\ LIST) \\
|\ (dp\_lan\colon DOCPROP \rightarrow LANG)\ (dp\_doc\colon DOCPROP \rightarrow DOC) \\
|\ (pp\_spec\colon PROGPROP \rightarrow SPEC)\ (pp\_prog\colon PROGPROP \rightarrow PROG) \\
|\ (ip\_prog\colon INFERPROP \rightarrow PROG)\ (ip\_inl\colon INFERPROP \rightarrow DOC\ LIST) \\
|\quad (ip\_outl\colon INFERPROP \rightarrow DOC\ LIST) \\
|\ \hline \\
|\quad s\_inl = Fst \wedge s\_outl = Snd \\
|\ \wedge\ dp\_lan = Fst \wedge dp\_doc = Snd \\
|\ \wedge\ pp\_spec = Fst \wedge pp\_prog = Snd \\
|\ \wedge\ ip\_prog = Fst \wedge ip\_inl = Fst\ o\ Snd \wedge ip\_outl = Snd\ o\ Snd
\end{array}
$$

3.2.3 Constructors and Projections

Since we are using schema types where appropriate the binding display and component selection notations will serve instead of defining constructor and projection functions.

3.3 Propositions

Three kinds of proposition are defined, concerning truth of documents, correctness of programs and derivation of documents.

In this section we provide the meaning for the three kinds of proposition introduced above. In each case this is a property which should be understood as defining when the relevant kind of sentence is true.

3.3.1 Truth

First, a document is true in some language if it is a member of that language. (it suffices for our present purposes to model a language by its set of true sentences)

HOL Constant

$$TrueDocP: DOCPROP \rightarrow BOOL$$

$$\forall dp{:}DOCPROP \bullet\ TrueDocP\ dp \Leftrightarrow dp\_doc\ dp \in dp\_lan\ dp$$

3.3.2 Inference

Next, a list of documents (the conclusions) is inferred by a program from a list of documents (the premises) if the function which is the value of the program maps the premises to the conclusions.

HOL Constant

$$DocInferP: INFERPROP \rightarrow BOOL$$

$$\forall ip{:}INFERPROP \bullet\ DocInferP\ ip \Leftrightarrow (ip\_prog\ ip)\ (ip\_inl\ ip) = ip\_outl\ ip$$

3.3.3 Soundness

A program is sound with respect to some specification if any list of documents computed by that program from a sequence of rue documents in its input languages will be a true document of the output language.

HOL Constant

$$SoundProgP: PROGPROP \rightarrow BOOL$$

$$\begin{array}{l}\forall pp: PROGPROP \bullet\ SoundProgP\ pp \Leftrightarrow (\forall idl: DOC\ LIST \bullet \\ \quad \forall_L\ (Map\ TrueDocP\ (Combine\ (s\_inl(pp\_spec\ pp))\ idl)) \\ \Rightarrow \forall_L\ (Map\ TrueDocP\ (Combine\ (s\_outl(pp\_spec\ pp))\ (pp\_prog\ pp\ idl))))\end{array}$$

3.3.4 Soundness of Identity Function

This is the most trivial soundness result I could think up, that the identity function is sound wrt any specification in which the output languages are the same as the input languages.

```
SML
|set_goal([],⌜∀ll: LANG LIST• SoundProgP ((ll, ll), (λx•x))⌝);
|a (rewrite_tac [
|    get_spec ⌜SoundProgP⌝,
|    get_spec ⌜s_inl⌝]);
|val id_spec_thm =
|    save_pop_thm "id_spec_thm";
```

3.4 Meta Reasoning

Elementary reasoning about inferences and their composition.

3.4.1 Introduction

In the following we investigate the kind of reasoning which could be undertaken in a language suitable for talking about the model we have introduced. Rather than invent a language, we use the syntax already available to us, and build up some elementary tools for reasoning about the model. If we then devised a suitable special concrete syntax and gave this its semantics in terms of the model by a semantic embedding into ProofPower GST, the proof tools we devise here would then serve as a tools for reasoning about this new language. The step to concrete syntax will be omitted, however, since we expect further elaborations to the model before this would be worthwhile.

3.4.2 What kind of logic?

We have a language in which programs, modelled as functions, are applied to documents to yield new documents, and hence an expression language involving function application. We have a couple of predicates which are applied to these programs and documents. So it looks like we are heading for some kind of predicate calculus.

On the other hand, if we consider languages as types of document and specifications as types of program, then both our predicates become typing assertions in an applicative calculus. Since typing inference in pure combinatory logic is the same as a fragment of propositional logic we may hope that a metalanguage based on the simple model we have in hand will logically much simpler than the predicate calculus, and we may hope for fully automatic proofs.

3.4.3 Rule Application

The central principle is that a sound program when applied to true premises yields true results. "Soundness" is of course, relative to a specification, and the specification tells you relative to which semantics the premises and conclusions are expected to be "true".

```
SML
|set_goal([],⌜∀(in_docs:DOC LIST) (out_docs:DOC LIST) (in_lans: LANG LIST)
|    prog out_docs out_lans•
| (∀_L (Map TrueDocP (Combine in_lans in_docs)))
|∧ SoundProgP ((in_lans, out_lans), prog)
|∧ DocInferP (prog, (in_docs, out_docs))
|⇒ ∀_L (Map TrueDocP (Combine out_lans out_docs))⌝);
|a (rewrite_tac[
|    get_spec ⌜TrueDocP⌝,
|    get_spec ⌜SoundProgP⌝,
|    get_spec ⌜DocInferP⌝,
|    get_spec ⌜Combine⌝,
|    get_spec ⌜Uncurry⌝,
|    get_spec ⌜Map⌝,
|    get_spec ⌜s_outl⌝,
|    get_spec ⌜∀_L⌝,
|    get_spec ⌜Uncurry⌝,
|    get_spec ⌜Fold⌝]
|    THEN REPEAT strip_tac);
|a (ALL_ASM_FC_T (MAP_EVERY ante_tac) []
|    THEN asm_rewrite_tac[]);
|val mm1_mp_thm =
|    save_pop_thm "mm1_mp_thm";
```

Because programs are permitted only one input and one output document, the limitations of this metamodel are very severe. Now we have modus ponens we can easily compose inferences to give results over chains of computations, but that's about all we can do.

3.4.4 Composition of Inferences

The following proof demonstrates that sound inferences compose.

```
SML
|set_goal([],⌜∀lans1 lans2 lans3 prog1 prog2•
|  SoundProgP ((lans1, lans2), prog1)
|∧ SoundProgP ((lans2, lans3), prog2)
|⇒ SoundProgP ((lans1, lans3), prog2 o prog1)⌝ );
|a (rewrite_tac (map get_spec [⌜SoundProgP⌝, ⌜s_inl⌝])
|    THEN REPEAT strip_tac THEN REPEAT (all_asm_ufc_tac[]));
|val mm1_comp_thm = save_pop_thm "mm1_mpc_thm";
```

4 Model 2 - Oracles and Assurance

4.1 Overview

Model 2 revolved around the truth of documents. If the semantics of an abstract language is fixed then the truth of documents is thereby determined. The semantics of specifications is defined in terms of the relationship between truth of the inputs to the program and truth of outputs. Thus we may expect that various programs will help us to determine the truth of documents.

Their help will be incomplete. When a document is output by some program its truth depends on the truth of any inputs to the program, and even if there are none, depends also on the fact that the program meets its specification. There is here an infinite regress which prevents our ever knowing the truth of a document with absolute certainty.

In the model which follows this is made explicit. The truth of documents is to be certified relative to certain assumptions. Since in practical applications of the proposed assurance calculus the number of assumptions might otherwise prove large, the assumptions we consider are effectively that some authority is infallible.

This permits authorities to be established which endorse a package of "assumptions" which provides a practical basis for establishing truth in some domain of interest. A simple example might be an authority for set theoretic truth which endorses the deductive aparatus of first order logic, the axioms known as "ZFC" and one or more proof tools which are judged capable of checking whether a document contains contains only claims provable in first order logic from the axioms of ZFC.

This you may view as a fairly exacting kind of scepticism. No document will ever receive unconditional or absolute assent. Docuemnts will sometime received qualified assent, which means that we affirm that they are true if some collection of authories has been infallible in their judgements on "less difficult" documents.

In this context an oracle is a program alleged to satisfy some specification, the specification prescribing in effect some subject matter in which the oracle is competent.

It is expected that when an authority endorses something that this endorsement is recorded as a digital signature, however, this is not covered by the model.

4.2 Abstract Syntax

In this model we introduce a language in which certain kinds of statement can be expressed and asserted relative to some set of authorities. The language is simple and non-recursive, and so its abstract syntax does not require an recursive definition, though the semantics will be recursively defined.

4.2.1 Theory X-Logic-2

Theory X-Logic-2 is a theory in ProofPower HOL, derived from a previous model written in Isabelle HOL.

This model begins with an abstract syntax for the language. 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 documents languages and programs are all understood to inhabit the World Wide Web and each

identified by a URI, which is a string. So we begin with a type abbreviation indicating that URIs are to be represented by strings.

SML
```
open_theory "X-Logic-1";
force_new_theory "X-Logic-2";
set_merge_pcs["hol", "'savedthm_cs_∃_proof"];
```

SML
```
declare_type_abbrev("URI", [], ⌜:STRING⌝);
declare_type_abbrev("DOCU", [], ⌜:URI⌝);
declare_type_abbrev("LANU", [], ⌜:URI⌝);
declare_type_abbrev("PROGU", [], ⌜:URI⌝);
declare_type_abbrev("AUTH", [], ⌜:URI⌝);
```

4.2.2 sentences

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 metalanguage therefore contains sentences which express the claim 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

SML
```
declare_type_abbrev("SENT", [], ⌜:(DOCU LIST × LANU LIST)
    + (PROGU × LANU LIST × LANU LIST)
    + (PROGU × DOCU LIST × DOCU LIST)⌝);
```

The significance of endorsements will become clearer shortly.

4.2.3 judgements

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.

SML
```
declare_type_abbrev("STAMP", [], ⌜:ℕ⌝);
declare_type_abbrev("JUDG", [],
    ⌜:(STAMP × AUTH × AUTH SET × SENT) + (AUTH SET)⌝);
```

A special authority called $\perp$ (bottom) may be used without signature, and is therefore untrustworthy.

HOL Constant

$\perp : AUTH$

T

```
consts
    jstamp :: judgement => stamp
    jauth  :: judgement => authority
    jauths :: judgement => authority set
    jsent  :: judgement => sentence

primrec
    "jstamp (Assert st as se) = st"
primrec
    "jauth (Endorse a as) = a"
primrec
    "jauths (Assert st as se) = as"
    "jauths (Endorse a as) = as"
primrec
    "jsent (Assert st as se) = se"
```

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.

4.3 Semantics

The semantics of sentences and judgements is defined as truth valuations relative to appropriate interpretations.

4.3.1 sentence interpretations

Isabelle

```
types
    document_map = document => string
    language_map = language => string set
```

```
    program_map = program => (string list => string list)

datatype Sinterp = SI document_map language_map program_map

consts
    docmap :: Sinterp => document_map
    langmap :: Sinterp => language_map
    progmap :: Sinterp => program_map

primrec "docmap (SI d l p) = d"
primrec "langmap (SI d l p) = l"
primrec "progmap (SI d l p) = p"
```

4.3.2 true sentences

Isabelle

```
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)"

constdefs
  trueprogspec :: [Sinterp, program, language list, language list] => bool
   "trueprogspec i p ill oll ==
    (! idl . truedoclist i idl ill
            --> truedoclist i (progmap i p idl) oll)"

  truecompute :: [Sinterp, program, document list, document list] => bool
   "truecompute i p idl odl == (odl = progmap i p idl)"

  trueendorse :: [Sinterp, authority set] => bool
   "trueendorse i al == True"

consts
  truesen :: [Sinterp, sentence] => bool

primrec
  "truesen i (TrueDocs dl ll) = truedoclist i dl ll"
  "truesen i (ProgSpec p ill oll) = trueprogspec i p ill oll"
  "truesen i (Compute p idl odl) = truecompute i p idl odl"
```

4.3.3 judgement interpretations

A judgement is true if infallibility of its authorities implies the truth of its sentence. To formalise this we need to talk about infallibility, and to talk about infallibility we need to have an interpretation which tells us which judgements have been affirmed by which authorities.

We therefore devise an extended interpretation for judgements in which a judgement map is available mapping each authority to the judgements it has affirmed.

Isabelle

```
types
    judgement_map = authority => judgement set

datatype Jinterp = JI Sinterp judgement_map

consts
    judgemap :: Jinterp => judgement_map
    sinterp :: Jinterp => Sinterp

primrec "judgemap (JI s j) = j"
primrec "sinterp (JI s j) = s"
```

4.3.4 infallibility

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 slighly ¡i¿strengthening¡/i¿ the meaning of judgements, so that their truth depends only on the truth of ¡i¿previous¡/i¿ 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. It will be clear from the proof rules which we show later that this mechanism does not have to be implemented with timestamps.

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.

Isabelle

```
types
    inftest = [nat, 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))}"

  hirec :: "[nat, (inftest * truthtest)] => inftest"
  "hirec n1 tsts n2 auth ji == case n1 of
    0       => True |
    (Suc m) => !a. (auth,a):(authrel (judgemap ji))
               --> (!j. j:(judgemap ji a) & (jstamp j) <= m
                    --> (snd tsts j ji))"

  jtrec :: "[nat, (inftest * truthtest)] => truthtest"
  "jtrec n tsts j ji == case n of
    0       => snd tsts j ji |
    (Suc m) => (!auth. auth:(jauths j) --> (fst tsts) (n-1) auth ji)
               --> snd tsts j ji"

consts
  hijt :: "nat => (inftest * truthtest)"

primrec
  "hijt 0       = ((λx y z. True), (λx y. True))"
  "hijt (Suc n) = (hirec n (hijt n), jtrec n (hijt n))"

constdefs
  hitherto_infallible  :: [nat, authority, Jinterp] => bool
  "hitherto_infallible n == fst (hijt n) n"
```

4.3.5 true judgements

Isabelle

```
consts
  truej  :: [Jinterp, judgement] => bool

primrec
  "truej ji (Assert stamp auths sent)
   = snd (hijt stamp) (Assert stamp auths sent) ji"
  "truej ji (Endorse auth auths) = True"
```

4.4 Proof Rules

4.4.1 inference

Isabelle

```
consts
  thms :: judgement set => judgement set
  "|-"  :: [judgement set, judgement] => bool   (infixl 50)

translations
  "H |- p" == "p : thms(H)"

inductive "thms(H)"
  intrs
  H "p:H ==> H |- p"
  E "[| H |- Assert n1 (ll Un levels1) sent;
        H |- Endorse l ll;
        n1 < n3;
        n2 < n3 |]
     ==> H |- Assert n3 ({l} Un levels2) sent"
  TI "[| H |- Assert n1 levels1 (TrueDocs [d] [l]);
        H |- Assert n2 levels2 (TrueDocs dl ll);
        n1 < n3;
        n2 < n3 |]
     ==> H |- Assert n3 (levels1 Un levels2) (TrueDocs (d#dl) (l#ll))"
  TEH "[| H |- Assert n1 levels (TrueDocs (d#dl) (l#ll));
        n1 < n2 |]
     ==> H |- Assert n2 levels (TrueDocs [d] [l])"
  TET "[| H |- Assert n1 levels (TrueDocs (d#dl) (l#ll));
        n3 < n2 |]
     ==> H |- Assert n2 levels (TrueDocs dl ll)"
  MP "[| H |- Assert n1 levels (TrueDocs idl ill);
        H |- Assert n2 levels (ProgSpec p ill oll);
        H |- Assert n3 levels (Compute p idl odl);
        n1 < n4;
        n2 < n4;
        n3 < n4 |]
     ==> H |- Assert n4 levels (TrueDocs odl oll)"
end
```

5 Model 3 - Digital Signatures

6 The Theory X-Logic-Auth

6.1 Parents

tc *rbjmisc*

6.2 Constants

Join	$'a\ AL \to 'a\ AL \to 'a\ AL$
Meet	$'a\ AL \to 'a\ AL \to 'a\ AL$
PathLt	$('a \to BOOL) \to 'a\ AL$
AuthLt	$'a\ AL \to ('a \to BOOL)\ AL$
LevelLt	$('a \times 'a\ AL \to BOOL) \to 'a\ AL \to ('a \to BOOL)\ AL$
LevelEquiv	$('a \times 'a\ AL \to BOOL) \to 'a\ AL \to ('a \to BOOL)\ AL$
Levels	$('a \times 'a\ AL \to BOOL) \to 'a\ AL\ \mathbb{P}\ \mathbb{P}$

6.3 Aliases

$\cup$	$Join : 'a\ AL \to 'a\ AL \to 'a\ AL$
$\cap$	$Meet : 'a\ AL \to 'a\ AL \to 'a\ AL$

6.4 Type Abbreviations

$'a$ ***AL***	$'a\ AL$

6.5 Definitions

Join

$$\vdash \forall\ x\ y\bullet\ x \cup y = (\lambda\ p\bullet\ x\ p \vee y\ p)$$

Meet

$$\vdash \forall\ x\ y \bullet\ x \cap y = (\lambda\ p\bullet \exists\ r\ s\bullet\ x\ r \wedge y\ s \wedge p = (\lambda\ x\bullet\ r\ x \vee s\ x))$$

PathLt

$$\vdash \forall\ p\ q\bullet\ PathLt\ p\ q \Leftrightarrow (\forall\ a\bullet\ q\ a \Rightarrow p\ a)$$

AuthLt

$$\vdash \forall\ a\ b \bullet\ AuthLt\ a\ b \Leftrightarrow (\forall\ p\bullet\ a\ p \Rightarrow (\exists\ q\bullet\ PathLt\ p\ q \wedge b\ q))$$

LevelLt

$$\begin{array}{l}\vdash \forall\ cl \\ \quad \bullet\ LevelLt\ cl \\ \qquad = tc \\ \qquad\quad (\lambda\ l\ m \\ \qquad\qquad \bullet\ l = m \\ \qquad\qquad\quad \vee\ AuthLt\ l\ m \\ \qquad\qquad\quad \vee (\exists\ a \\ \qquad\qquad\quad \bullet\ cl\ (a, m) \\ \qquad\qquad\qquad \wedge\ l = (\lambda\ p\bullet\ p = (\lambda\ q\bullet\ q = a))))\end{array}$$

LevelEquiv

$$\vdash \forall\ cl \bullet\ LevelEquiv\ cl = (\lambda\ a\ b\bullet\ LevelLt\ cl\ a\ b \wedge LevelLt\ cl\ b\ a)$$

Levels

$$\vdash \forall\ cl\bullet\ Levels\ cl = Universe\ /\ LevelEquiv\ cl$$

6.6 Theorems

Join_id_thm $\vdash \forall\ x\ y \bullet\ x \cup x = x$

Join_com_thm $\vdash \forall\ x\ y \bullet\ x \cup y = y \cup x$

Meet_com_thm $\vdash \forall\ x\ y \bullet\ x \cap y = y \cap x$

Join_assoc_thm1

$\vdash \forall\ x\ y\ z \bullet\ x \cup y \cup z = (x \cup y) \cup z$

Join_assoc_thm2

$\vdash \forall\ x\ y\ z \bullet\ (x \cup y) \cup z = x \cup y \cup z$

Meet_assoc_thm1

$\vdash \forall\ x\ y\ z \bullet\ x \cap y \cap z = (x \cap y) \cap z$

Meet_assoc_thm2

$\vdash \forall\ x\ y\ z \bullet\ (x \cap y) \cap z = x \cap y \cap z$

al_distrib_thm1

$\vdash \forall\ x\ y\ z \bullet\ x \cap (y \cup z) = x \cap y \cup x \cap z$

al_distrib_thm2

$\vdash \forall\ x\ y\ z \bullet\ (x \cup y) \cap z = x \cap z \cup y \cap z$

trans_LevelLt_thm

$\vdash \forall\ es \bullet\ trans\ (LevelLt\ es)$

trans_LevelLt_thm2

$\vdash \forall\ es\ s\ t\ u$

$\bullet\ LevelLt\ es\ s\ t \wedge LevelLt\ es\ t\ u \Rightarrow LevelLt\ es\ s\ u$

7 The Theory X-Logic-1

7.1 Parents

rbjmisc

7.2 Children

X−Logic−2

7.3 Constants

ip_outl	*INFERPROP → DOC LIST*
ip_inl	*INFERPROP → DOC LIST*
ip_prog	*INFERPROP → PROG*
pp_prog	*PROGPROP → PROG*
pp_spec	*PROGPROP → SPEC*
dp_doc	*DOCPROP → DOC*
dp_lan	*DOCPROP → LANG*
s_outl	*SPEC → LANG LIST*
s_inl	*SPEC → LANG LIST*
TrueDocP	*DOCPROP → BOOL*
DocInferP	*INFERPROP → BOOL*
SoundProgP	*PROGPROP → BOOL*

7.4 Types

DOC

7.5 Type Abbreviations

LANG	*LANG*
PROG	*PROG*
SPEC	*SPEC*
DOCPROP	*DOCPROP*
PROGPROP	*PROGPROP*
INFERPROP	*INFERPROP*

7.6 Definitions

s_inl
s_outl
dp_lan
dp_doc
pp_spec
pp_prog
ip_prog
ip_inl
ip_outl $\vdash s\_inl = Fst$
$\wedge\ s\_outl = Snd$
$\wedge\ dp\_lan = Fst$
$\wedge\ dp\_doc = Snd$
$\wedge\ pp\_spec = Fst$
$\wedge\ pp\_prog = Snd$
$\wedge\ ip\_prog = Fst$
$\wedge\ ip\_inl = Fst\ o\ Snd$
$\wedge\ ip\_outl = Snd\ o\ Snd$

TrueDocP $\vdash \forall\ dp \bullet\ TrueDocP\ dp \Leftrightarrow dp\_doc\ dp \in dp\_lan\ dp$

DocInferP $\vdash \forall\ ip$
$\bullet\ DocInferP\ ip \Leftrightarrow ip\_prog\ ip\ (ip\_inl\ ip) = ip\_outl\ ip$

SoundProgP $\vdash \forall\ pp$
$\bullet\ SoundProgP\ pp$
$\Leftrightarrow (\forall\ idl$
$\bullet\ \forall_L$
$(Map$
$TrueDocP$
$(Combine\ (s\_inl\ (pp\_spec\ pp))\ idl))$
$\Rightarrow \forall_L$
$(Map$
$TrueDocP$
$(Combine$
$(s\_outl\ (pp\_spec\ pp))$
$(pp\_prog\ pp\ idl))))$

7.7 Theorems

id_spec_thm $\vdash \forall\ ll \bullet\ SoundProgP\ ((ll,\ ll),\ (\lambda\ x \bullet\ x))$

mm1_mp_thm $\vdash \forall\ in\_docs\ out\_docs\ in\_lans\ prog\ out\_docs\ out\_lans$
$\bullet\ \forall_L\ (Map\ TrueDocP\ (Combine\ in\_lans\ in\_docs))$
$\wedge\ SoundProgP\ ((in\_lans,\ out\_lans),\ prog)$
$\wedge\ DocInferP\ (prog,\ in\_docs,\ out\_docs)$
$\Rightarrow \forall_L\ (Map\ TrueDocP\ (Combine\ out\_lans\ out\_docs))$

mm1_mpc_thm $\vdash \forall\ lans1\ lans2\ lans3\ prog1\ prog2$
$\bullet\ SoundProgP\ ((lans1,\ lans2),\ prog1)$
$\wedge\ SoundProgP\ ((lans2,\ lans3),\ prog2)$
$\Rightarrow SoundProgP\ ((lans1,\ lans3),\ prog2\ o\ prog1)$

8 The Theory X-Logic-2

8.1 Parents

$X-Logic-1$

8.2 Constants

$\perp$	$AUTH$

8.3 Type Abbreviations

URI	$AUTH$
DOCU	$AUTH$
LANU	$AUTH$
PROGU	$AUTH$
AUTH	$AUTH$
SENT	$SENT$
STAMP	$STAMP$
JUDG	$JUDG$

8.4 Definitions

$\perp$	$\vdash T$

9 INDEX