The Use of HOL in the Development of Secure Systems

Roger Bishop Jones

ICL Defence Systems

ABSTRACT

This document consists of the overheads for a presentation to BCS FACS on May 25 1989.

The ICL Defence Systems
FORMAL METHODS UNIT

HOL
LANGUAGE, LOGIC
and TOOL

SPECIAL CHARACTERISTICS
of SECURITY APPLICATIONS

METHODOLOGICAL
IDIOSYNCRASIES

THE ICL DEFENCE SYSTEMS
FORMAL METHODS UNIT

STARTED in 1985 by Roger Stokes

SLOGAN:

working on real problems
with real tools

CURRENT TEAM:

Dr. Rob Arthan
(tools)

Dr. Kevin Blackburn
(proof technology)

Dr. Barry Homer
(security modelling)

Dr. Clive Jervis
(hardware verification)

Roger Jones
(leader, foundations)

Geoff Scullard
(HOL/VISULA link)

The TYPED LAMBDA-CALCULUS

TYPES are:

- type variables
- type constants
- closed under function space constructor

TERMS are:

- variables
- constants
- applications
- abstractions

```
TYPE ::=   Tvar <<string>>                    |
           Tcon <<string>>                    |
           Fun <<TYPE × TYPE>>

TERM ::=   Ivar <<string × TYPE>>             |
           Icon <<string × TYPE>>             |
           App <<TERM × TERM>>                |
           Abs <<string × TYPE × TERM>>
```

HOL
LANGUAGE and LOGIC and TOOL

the HOL **LANGUAGE** is:

- The typed Lambda-Calculus

with:

- 2 primitive types ("bool" and "ind")
- 3 primitive constants ("==>" "=" and "@")

The HOL **LOGIC** has:

- Equality rules (α, β and η reduction) from TYPED LAMBDA CALCULUS
- 2 extra inference rules (MP, DISCH)
- 4 extra axioms (2 propositional, choice and infinity)

HOL is a FOUNDATION SYSTEM,
within which most of mathematics
can be developed using only
CONSERVATIVE EXTENSIONS

The HOL **TOOL** is:

a proof development tool, developed from Cambridge LCF, by Mike Gordon and his hardware verification group at the University of Cambridge in 1985.

THE LCF PARADIGM

implement proof checker using
a TYPED FUNCTIONAL programming LANGUAGE
as META-LANGUAGE (e.g. SML)

abstract data type of THEOREMS
GUARANTEED SOUND by the type checker
(assuming the logic is well defined)

META-LANGUAGE is AVAILABLE TO the USER
for programming proofs and other customisation,
WITHOUT risk of COMPROMISING the CHECKER.

BENEFITS of the LCF PARADIGM

- HIGH ASSURANCE of SOUNDNESS

- EASY to CUSTOMISE and EXTEND

- COMPLETE USER CONTROL
  of PROOF STRATEGY

- RERUNNABLE PROOF SCRIPTS

- LEG WORK convertible to HEAD WORK
  by PROGRAMMING in META-LANGUAGE

SPECIAL CHARACTERISTICS
of
SECURITY APPLICATIONS

VERY HIGH ASSURANCE SOUGHT

IMPORTANCE of FORMAL METHODS
RECOGNISED in EVALUATION GUIDELINES

VERY NARROW AREA of CONCERN

FORMAL TREATMENT CONFINED
to SECURITY CHARACTERISTICS

REASONING ABOUT INFORMATION FLOW

METHODOLOGICAL ISSUES

CONSISTENCY

Machine checked formal proofs are worthless unless the logical system in which they are derived is known to be sound.

LOOSENESS

To obtain highest possible levels of assurance in respect of critical requirements the statement of these requirements must not be cluttered either by:

(i)
details of non-critical requirements

(ii)
implementation detail

SATISFACTION and REFINEMENT

To obtain formal correctness proofs a formally precise account of the 'implementation relation' is needed.

SOLUTIONS

SPECIFICATION LANGUAGES
should be
MATHEMATICAL FOUNDATION SYSTEMS

all APPLICATIONS should require
ONLY CONSERVATIVE EXTENSIONS

SPECIFICATIONS should be
PROPERTIES or SETS

SATISFACTION is MEMBERSHIP
(or possession of property)

"I satisfies S"
should be rendered formally as
"$I \in S$" or "$S\ I$"

REFINEMENT
is
INCLUSION or ENTAILMENT

"S^1 refines S^2"
should be rendered formally as
"$S^1 \subseteq S^2$" or "$\forall I \bullet S^1\ I \Rightarrow S^2\ I$"

CONSISTENCY

ABSOLUTE ASSURANCE
of
CONSISTENCY of LOGICAL SYSTEM
NOT POSSIBLE

but we SHOULD HAVE

MACHINE CHECKABLE RELATIVE CONSISTENCY
for
USER EXTENSIONS
(i.e. all applications)

LANGUAGE DESIGNERS and TOOL BUILDER'S
should provide a
SAFE DEVELOPMENT ENVIRONMENT

within which

USER ERRORS CANNOT COMPROMISE
the CONSISTENCY OF the PROOF ENVIRONMENT

in other words
SPECIFICATION LANGUAGES
should be
MATHEMATICAL FOUNDATION SYSTEMS

MATHEMATICAL FOUNDATION SYSTEMS

- A mathematical foundation system is a FORMAL LOGICAL SYSTEM within which most of MATHEMATICS CAN BE DEVELOPED using only CONSERVATIVE EXTENSIONS

- CONSERVATIVE EXTENSIONS, are typically well formed definitions
  e.g. "name = term"
  or loosely "name $\in$ non_empty_set"

- FOUNDATION SYSTEM come with
  WELL POPULATED UNIVERSES
  (witnesses for consistency proofs)

The following are FOUNDATION SYSTEMS:

- Higher Order Logic
- First Order Set Theory (e.g. ZFC)
- Constructive Type Theories
  (e.g. ITT, NUPRL)
- Z (if **N** is primitive)

The following are NOT foundation systems:

- First Order Logic
- Anything weaker than first order logic!
- Probably not VDM-SL

LOOSENESS

LOOSE SPECIFICATIONS
are
CRUCIAL TO SECURITY APPLICATIONS

PRE/POST - CONDITIONS
NOT SUFFICIENTLY EXPRESSIVE

Z-SCHEMAS
NOT SUFFICIENTLY EXPRESSIVE

working in HOL or in set theory
(even in Z)
we can use PROPERTIES or SETS
as SPECIFICATIONS

and
SATISFACTION is MEMBERSHIP

Give a state consisting of one highly classified and one lowly classified object:

```
┌─ STATE[p] ──────────────────────
│ high, low :bool
└──────────────────────
```

can we specify loosely an operation on the state which does not result in any information transfer from ‘high’ to ‘low’?

```
┌─ ΔSTATE[p] ──────────────────────
│ STATE, STATE’
├──────────────
│ ?
└──────────────────────
```

It is easy enough to give a specific operation satisfying this requirement, but to capture the requirement loosely we have to use a loose specification outside of the schema, e.g.:

$$f: STATE \rightarrow STATE$$

$$\forall s^1\ s^2 : STATE \bullet (s^1.low = s^2.low) \Rightarrow (f\ s^1).low = (f\ s^2).low$$

We could then write our schema:

$\Delta STATE[p]$

$STATE, STATE'$

$$\theta STATE' = f\ \theta STATE$$

but since all the work has been done in the specification of ‘f’ the use of the schema appears superfluous.

Note that in the axiomatic definition of f, the requirement is expressed as a property of f, but this property has not itself been given a name.

It is therefore not possible to express in the object language the claim that some other explicitly defined function has this property.

For example the following function has the required property:

$$
\begin{array}{|l}
\quad g: STATE \rightarrow STATE \\
\hline
\quad \forall s : STATE \bullet f\ s = s
\end{array}
$$

but we cannot state this in Z without restating the original property (though it can be said in the meta-language).

To enable such correctness propositions to be expressed we must give a name to the property itself as follows:

$$secure : \mathbb{P}\,(STATE \rightarrow STATE)$$

$$f \in secure \Leftrightarrow$$
$$\forall s^1\ s^2{:}STATE \bullet (s^1.low = s^2.low)$$
$$\Rightarrow (f\ s^1).low = (f\ s^2).low$$

The conjecture that ‘g’ satisfies this specification can now be expressed:

$$\vdash?\ g \in secure$$

If we define a further requirement:

$$\begin{array}{|l}
\quad \mathrm{safe} : \mathbb{P}\ (\mathrm{STATE} \rightarrow \mathrm{STATE}) \\
\hline
\quad f \in \mathrm{safe} \Leftrightarrow \\
\quad \forall s^1\ s^2 : \mathrm{STATE} \bullet (s^1.\mathrm{high} = s^2.\mathrm{high}) \\
\qquad\qquad\qquad \Rightarrow (f\ s^1).\mathrm{high} = (f\ s^2).\mathrm{high}
\end{array}$$

Then the combination of these two requirements:

$$\begin{array}{|l}
\quad \mathrm{no_flow} : \mathbb{P}\ (\mathrm{STATE} \rightarrow \mathrm{STATE}) \\
\hline
\quad \mathrm{no_flow} = \mathrm{secure} \cap \mathrm{safe}
\end{array}$$

may be regarded as a REFINEMENT of the original specification "secure".

That it is a refinement can be expressed in the object language as the conjecture:

$$\vdash ?\ \mathrm{no\_flow} \subseteq \mathrm{secure}$$

Note that here refinement is defined as a relationship between specifications which is distinct from the relationship between a specification and an implementation.

1. SPECIFYING OPERATIONS AS FUNCTIONS

Type of *Object*

$\quad$ **AUTO**

Type of *Specification*

$\quad$ $\mathbb{P}$ **AUTO**

Type of *Operation*

$$\mathbf{IN \times STATE \rightarrow STATE \times OUT}$$
$$\subseteq \mathbb{P}(\mathrm{IN \times STATE \times STATE \times OUT})$$

Type of *Specification of* Operation

$$\mathbb{P}\ \mathbf{(IN \times STATE \rightarrow STATE \times OUT)}$$

Type of *Non-Deterministic* Operation

$$\mathbf{IN \times STATE \rightarrow \mathbb{P}\ (STATE \times OUT)}$$

Type of *Specification of Non-Deterministic* Operation

$$\mathbb{P}\ \mathbf{(IN \times STATE \rightarrow \mathbb{P}\ (STATE \times OUT))}$$

Type of *Partial* Operation

$$\mathbf{IN \times STATE \nrightarrow STATE \times OUT}$$

Type of *Specification of Partial* Operation

$$\mathbb{P}\ \mathbf{(IN \times STATE \nrightarrow STATE \times OUT)}$$

Type of *Partial Non-Deterministic* Operation

$$\mathbf{IN \times STATE \nrightarrow \mathbb{P}\ (STATE \times OUT)}$$

Type of *Specification of Partial Non-Deterministic* Operation

$$\mathbb{P}\ \mathbf{(IN \times STATE \nrightarrow \mathbb{P}\ (STATE \times OUT))}$$

THE IMPLEMENTATION RELATION

Using SETS or PROPERTIES as SPECIFICATIONS we can express in HOL (or Z) the CORRECTNESS PROPOSITION which asserts that:

some ‘IMPLEMENTATION’
SATISFIES
its ‘SPECIFICATION’

NO CLUTTER OF PROOF OBLIGATIONS

NO DEPENDENCE
on
VERIFICATION CONDITION GENERATORS

THE SLOGANS

SPECIFICATION LANGUAGES
should be
FOUNDATION SYSTEMS

SATISFACTION
is
MEMBERSHIP