Formal Methods — Panel Discussion

D. N. Hoover
Odyssey Research Associates, Inc.
301 Dates Dr.
Ithaca NY 14850-1326
Internet: hoove@oracorp.com

May 24, 1995

FM Challenges—Object Orientation

Object Orientation is a complex collection of concepts, some of which are already part of FM (come from FM), some of which have yet to be incorporated in FM.

Static Aspects of Object Orientation

OO	FM
class	abstract data type, logical theory
subclass	theory extension
object	state machine

Dynamic Aspects of OO (Function Dispatching)

E.g. constructed type definitions can be given in pieces.

"Normal" code

```
BOOL_EXP ::= Vble V | Not BOOL_EXP | And BOOL_EXP BOOL_EXP


value assn (Vble v)  = assn v
         | (Not e)  = not (value assn e)
         | (And e1 e2) = (value assn e1) and (value assn e2)
```

Object-oriented code

```
Class Type BOOL_EXP with Function value

BOOL_EXP Clause
{
  Vble V
  value assn (Vble v) = assn v
}

BOOL_EXP Clause
{
  Not BOOL_EXP
  value assn (Not e) = not (value assn e)
}
```

etc.

FM Support for Object Orientation

- I don't know of any FM Spec or Proof tool that supports the "decentralized" OO definition.
- Perhaps new ideas in logic are needed to properly support reasoning about dynamic subtyping.
- There are some related problems regarding formal development of concurrent programs.
- What is the right meaning of "compositional?"

FM and Program Methodology

Structural/Functional Specification/Design

- Less "Black box" nature than FM.
- Not entirely formal.
- Nevertheless a good start for an FM application (e.g. Statemate).
- It also includes heuristics for trying to figure out what the spec should be.

OO Specification/Design

- In practise (e.g. Booch), seems to say a lot about how modules are related, not much about data representations, or what the modules actually do.
- Is this a problem or an opportunity?

Assorted Theses

1. FM is very powerful and should be used for more things than telling people their specs/code are wrong.
   - Generate code, tests, docs.
2. For example, HOL or PVS as an ML code generator.
3. FM tools should be usually automatic and should give helpful information about errors as possible.
4. Proving in general is hard, but in most applications it is not hard, unless induction is involved.
5. General purpose spec languages and theorem provers are not for everybody.

   But they are *very* useful for developing specialized tools.
6. In many settings, very simple concepts and tools are helpful.
7. The idea that you can establish something by proving it instead of testing all possible cases is inconceivable to most programmers/engineers.
8. Most programmers have no idea of what has to be in a useful spec. Learning this is probably the most valuable product of FM experience.
9. God bless the FAA.
   - Better "informal" specs to start from.

- Requirements for critical software reasonably well-defined.
- There are a number of reasonable places for FM to contribute.
- Ok to start by automating existing processes, later try to replace them.

10. Tool qualification is the main obstacle to use of FM in critical systems.

11. If you use FM, you can program fancier and still be safe.