An Integrated Formal Methods Tool-Chain and its Application to Verifying a File System Model

From Miguel A. Ferreira and José N. Oliveira

Thematic Seminar - Paper Recitation

July 21, 2012

Skeleton

- Brief Introduction
- Tool-chain Scheme
- Point-free Specification and Allow Mapping
- Execution and proof chain (VDM++ and HOL)
- Conclusion and discussion

Brief Introduction

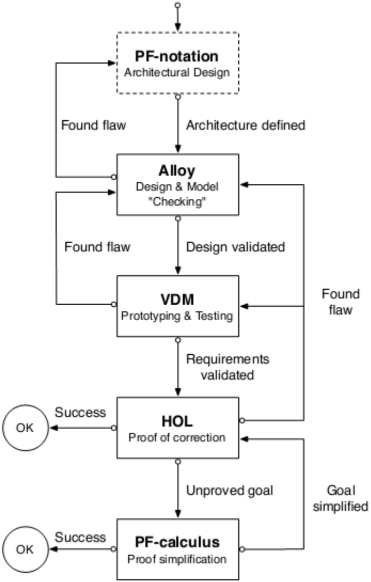
The paper proposes a tool-chain and also a short case study for validation.
The tool-chain has the following requirements:

1. promote incremental development and verification of specifications;
2. be agile enough to encourage users to verify even the smallest unit of their specifications;
3. be capable of producing immediate feedback to unveil problems;
4. be capable of performing fully automated consistency proofs;
5. be amenable to automatic code generation.

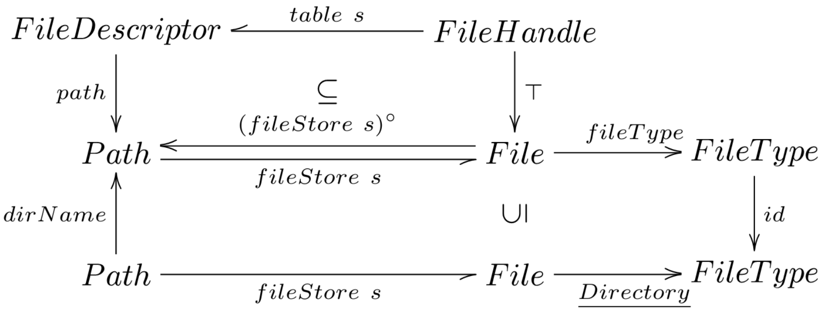
In addiction:

- The case study of the formal model for an abstract file system (following the Intel architecture).

Tool-chain Scheme



Point-free Model - An abstract file system.



- Referential integrity: non existing files cannot be handled by applications.
  $\forall p \in Path, fh \in FileHandle : (\exists fd \in FileDescriptor : p\,(path)\,fd \wedge fd\,(table\,s)\,fh) \implies (\exists f \in File : p\,(fileStore\,s) f)$
- Paths closure: parent directories always exist and are indeed directories.
  $(fileStore\,s).Directory.id^{o}.fileType^{o} \subseteq dirName.(fileStore\,s)$

Alloy Model

```
sig System {
  fileStore: Path -> lone File,
  table: FileHandle -> lone FileDescriptor }
abstract sig Path {dirName: one Path}
sig File {fileType : one FileType}
sig FileDescriptor {path: one Path}
sig FileHandle {}
```

And the Alloy model can be refined extending the path symbols and structure...

```
one sig Root extends Path
sig FileNames extends Path {}
pred ps[] {
  Reflexive[id[Root].dirName, Root]
  Acyclic[id[FileNames].dirName, FileNames] }
```

Lastly, after the model is validated can be translated to ...

VDM++

What is VDM++?

- A language and a set of tools that allows proof obligations generation.

Here we need to refine the path data structure according to VDM++...

- However, model translation to VDM++ involves additional effort and increases the steepness of the learning curve.
- The outcome is a sizeable VDM++ model; the abstraction level is lowered, in order for the specification to become executable.

VDM++ to HOL

For each PO arising from the specification, the Overture proof system can yield three different results:

- the PO evaluates to true (discharged) — no inconsistency found;
- the PO evaluates to false — a design inconsistency exists;
- the PO evaluates to an unproven goal — no conclusion from proof.

In the case that PO cannot be evaluated or is an unproved goal, we need to apply PF-Calculus techniques manually. This is clearly an hard step.

Conclusion and discussion

Pros

1. The paper is well structured and presents an ambitious refinement scheme, using point free specifications.
2. Given a PF-specification, we can convert PF-specifications to Alloy with a mechanized scheme, however, those refinements are very far from the first model. The conversion to VDM++ allows the generation of proof obligations to be proved in HOL. If cannot be proved it can be split in a point free fashion.
3. Making this steps in an automated way is a possible choice, and implies error freedom on translation but not on refinement.

Cons

1. The paper assumes that the reader is very comfortable with Alloy and VDM++.