Verifying Intel Flash File System Core Specification

M.A. Ferreira, S.S. Silva, J.N. Oliveira

DIUM/CCTC, University of Minho

Overture/VDM++ WS, May 26, 2008

Outline

Outline

VTTSE'05

Hoare and Misra proposed [HM05]

- Grand Challenge for research in computing science
- Verified Software Repository (`http://vsr.sourceforge.net`)

Goals

- Apply formal methods to real problems
- Automation of verification processes
- Focus on tool **interoperability**

Case Studies

Mondex

- Electronic purse protocol
- Great community response
- Practical results in model based verification

POSIX File Store

- on going effort to verify a POSIX compliant file system
- wide spread impact on many kinds of devices
- increased complexity

VFS (POSIX file store)

Mini-challenge

- proposed by Joshi and Holzmann (NASA - JPL) [JH07]
- specific for FLASH hardware
- intended for (critical) Mars Rover system

Outline

Past & Present

BSc course 2006/07

Preliminary work:

- VFS model: POSIX file store (VDM++) [S+07]
- ONFI model: flash device (VDM++) [DF07]

MSc course 2007/08 [DIU]

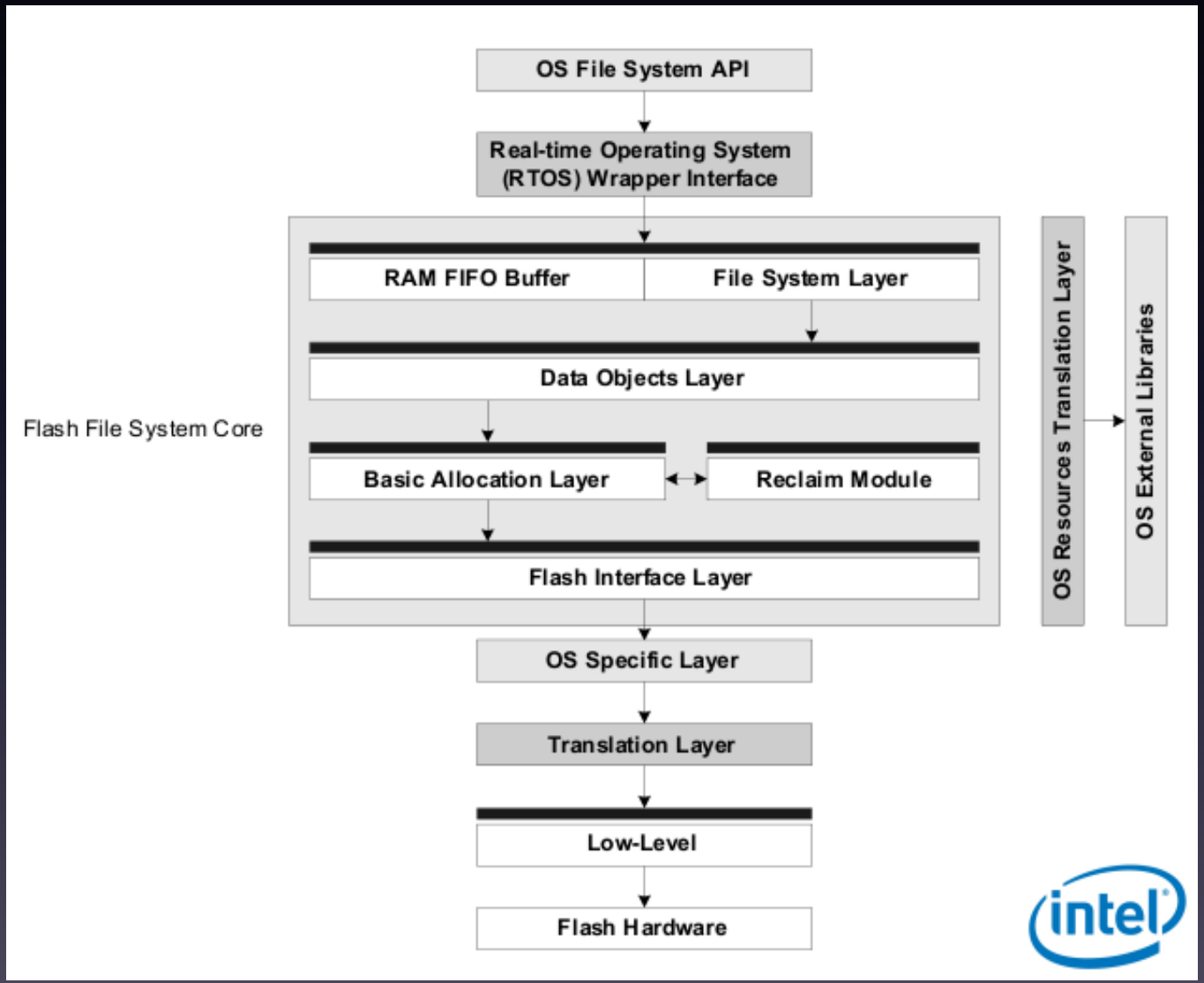
Currently working on modeling IFFSC (Intel Flash File System Core) in

- Alloy
- HOL
- VDM++

Why three different models?

Outline

Intel Flash File System — Architecture



Why the IFFSC [Cor04]

Advantages

- POSIX aware
- designed for FLASH memory
- layered architecture
- VFS and ONFI fit in IFFSC

Disadvantages

- document is currently deactivated
- some inconsistencies (eg. data type mismatch)

Outline

"All-in-one" Verification Strategy

We are using several tools of different kinds at the same time. Why?

Consider a typical proof obligation:

Satisfiability

$$\forall\, a \cdot a \in A \wedge \text{pre-}Op(a) : \exists\, b \cdot b \in B \wedge \text{post-}Op(b, a) \quad (1)$$

that is (in case of deterministic operations):

$$\forall\, a \cdot a \in A \wedge \text{pre-}Op(a) : Op(a) \in B \quad (2)$$

($a \in A$ and $b \in B$ check for the invariants associated to A and B, respectively)

"All-in-one" Verification Strategy

We are using several tools of different kinds at the same time.
Why?
Consider a typical proof obligation:

Satisfiability

$$\forall\, a \;\cdot\; a \in A \wedge \text{pre-}Op(a) : \exists\, b \;\cdot\; b \in B \wedge \text{post-}Op(b, a) \qquad (1)$$

that is (in case of deterministic operations):

$$\forall\, a \;\cdot\; a \in A \wedge \text{pre-}Op(a) : Op(a) \in B \qquad (2)$$

($a \in A$ and $b \in B$ check for the invariants associated to A and B, respectively)

"All-in-one" Verification Strategy

Different scenarios:

1. *Op* satisfies (2) but is semantically wrong — its does not behave according to the requirements
   - need for manual tests
   - strategy is to run the model as a prototype

   Thus the **VDMTools**

2. *Op* survives all tests (including dynamic type checking) and yet it does not satisfy (2)
   - a *model checker* able to generate counter-examples to (2) is useful
   - suggestions on how to improve *Op* are welcome

   Thus **Alloy**

"All-in-one" Verification Strategy

Different scenarios:

1. *Op* satisfies (2) but is semantically wrong — its does not behave according to the requirements
   - need for manual tests
   - strategy is to run the model as a prototype

   Thus the **VDMTools**

2. *Op* survives all tests (including dynamic type checking) and yet it does not satisfy (2)
   - a *model checker* able to generate counter-examples to (2) is useful
   - suggestions on how to improve *Op* are welcome

   Thus **Alloy**

"All-in-one" Verification Strategy

3. Model checker doesn't find any counter examples
   - a theorem prover is welcome to mechanically discharge (2)

   Thus **HOL**

4. PO (2) is too complex for the available theorem prover
   - decompose too complex PO into smaller sub-goals
   - the ultimate fallback: **manual** proof

   Thus the **PF-transform**

"All-in-one" Verification Strategy

3. Model checker doesn't find any counter examples
   - a theorem prover is welcome to mechanically discharge (2)

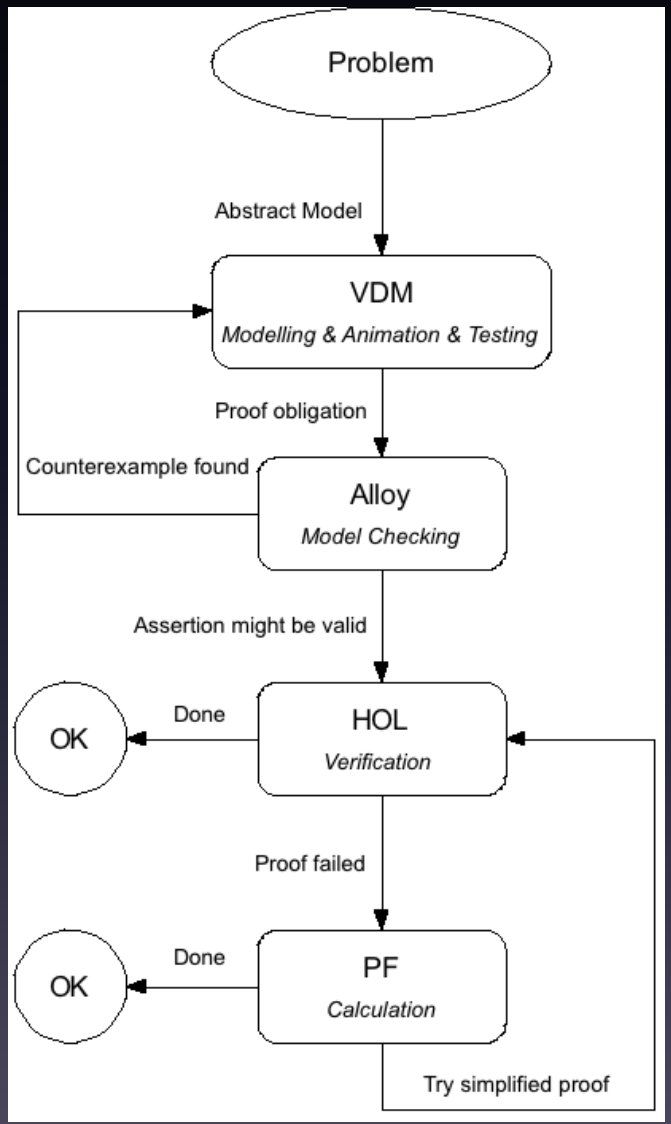
   Thus **HOL**

4. PO (2) is too complex for the available theorem prover
   - decompose too complex PO into smaller sub-goals
   - the ultimate hope is a pen-and-paper **manual** proof

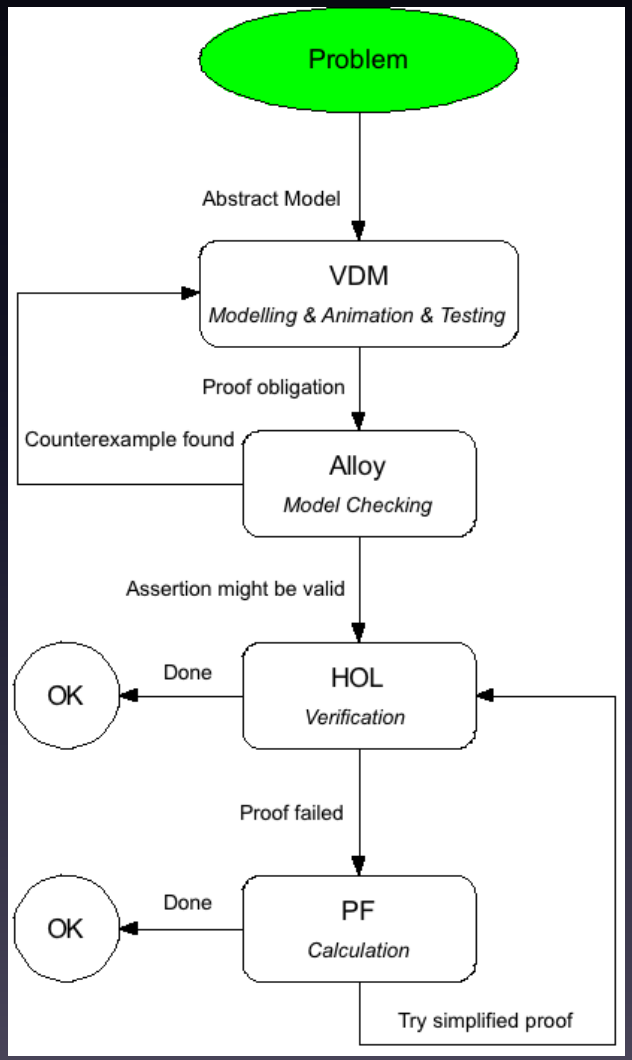
   Thus the **PF-transform**

Outline

Development & Verification Process

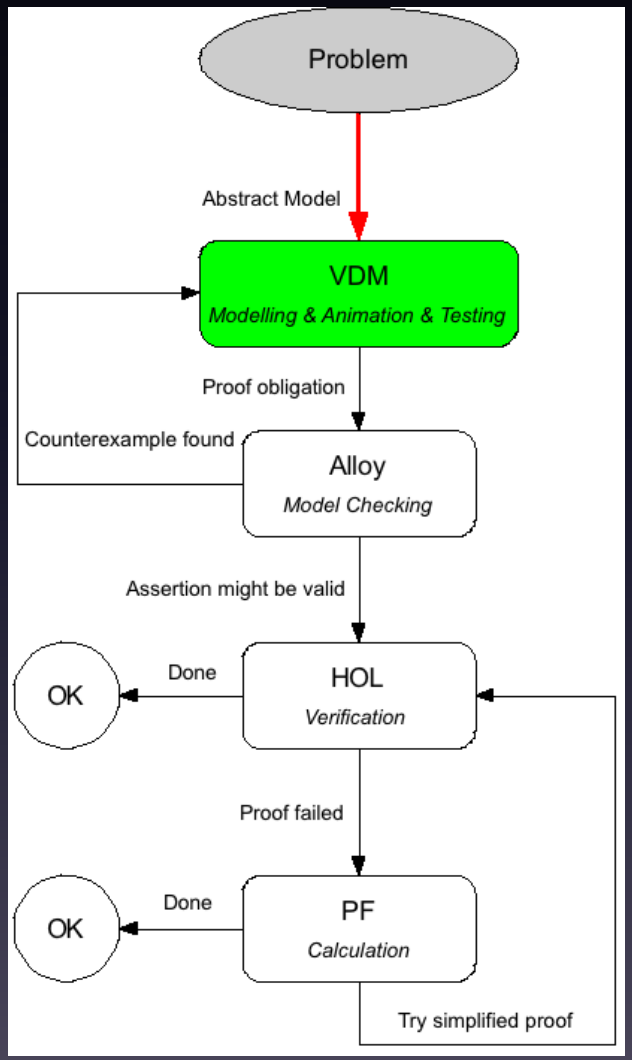


Problem



Understand the Problem

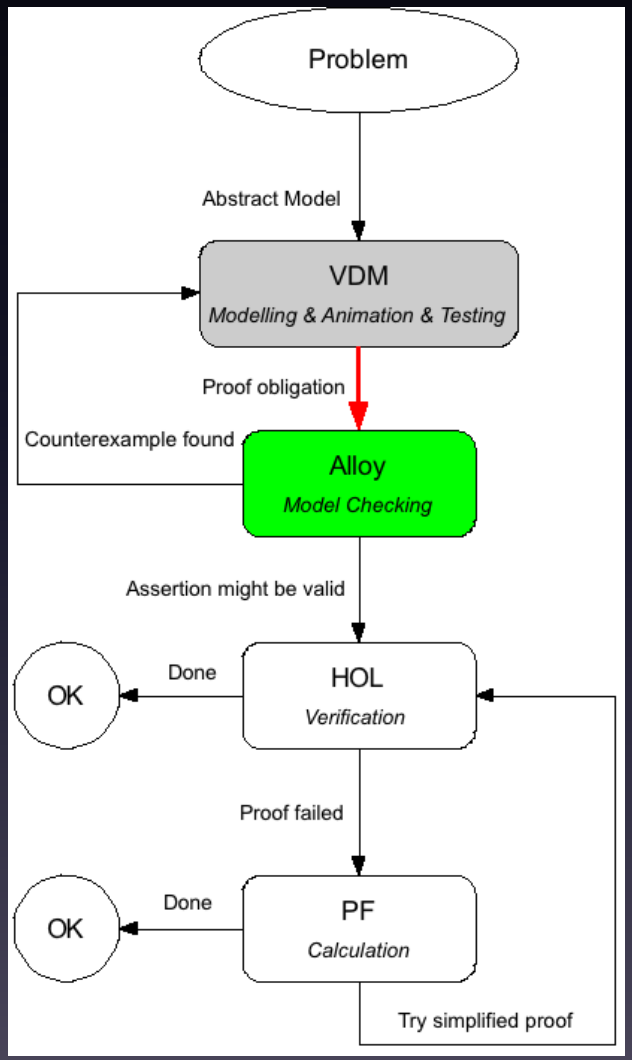
Model



Write VDM++ Model

- animate prototype
- run test suites
- run **Integrity Checker** to generate POs

Model Check



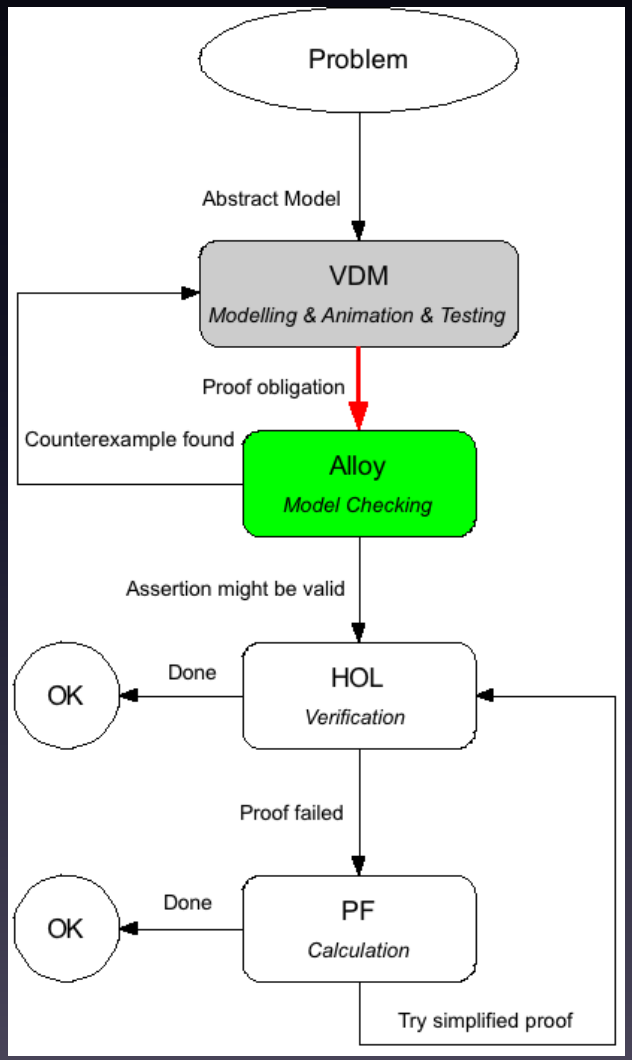
Write Alloy Model

- capture data types and invariants
- capture functions, pre and post conditions
- **abstract** sequences and numbers

Model Check in Alloy

- assert proof obligations
- try and find counter-examples

Model Check



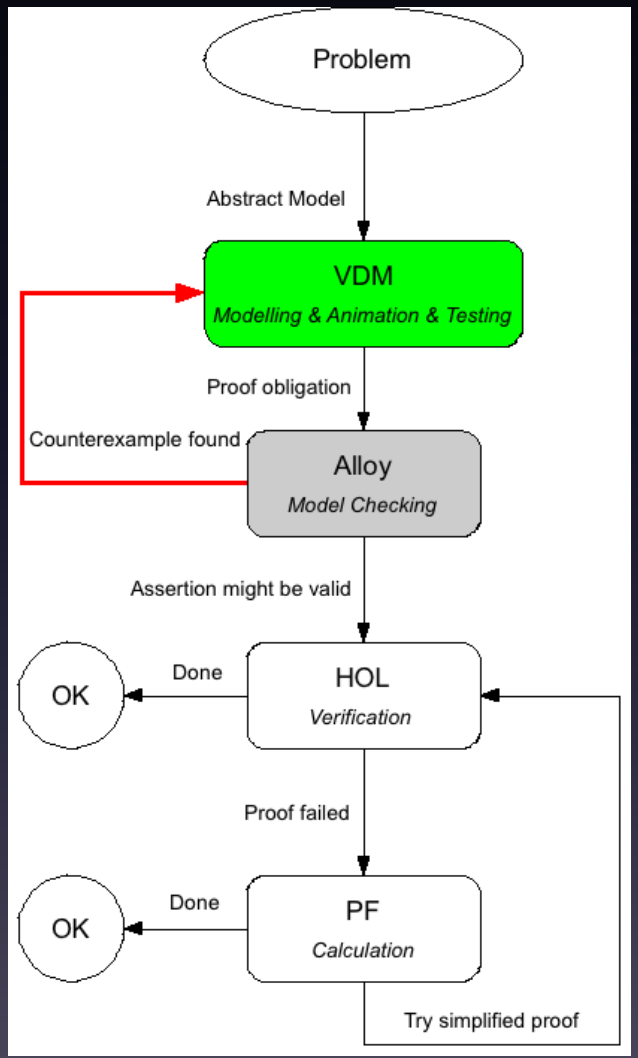
Write Alloy Model

- capture data types and invariants
- capture functions, pre and post conditions
- **abstract** sequences and numbers

Model Check in Alloy

- assert proof obligations
- try and find counter-examples

Review VDM++ Model



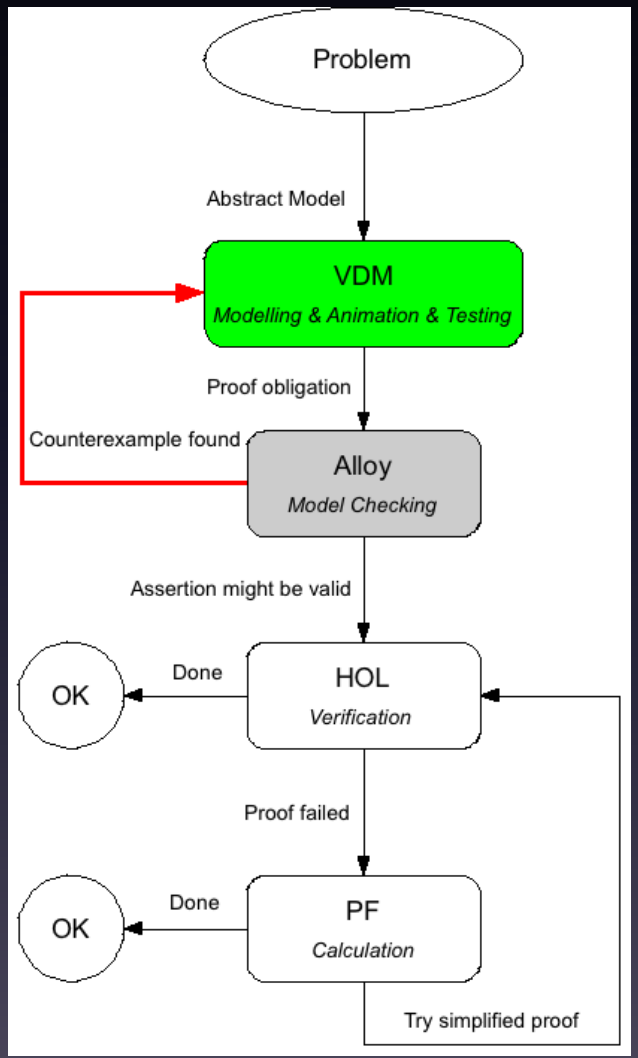
Counter-Example Found

- the assertion is invalid
- counter-example show **why** and **how**

Back to VDM++ Model

- error in model?
- too weak a pre-condition?
- too strong an invariant?

Review VDM++ Model



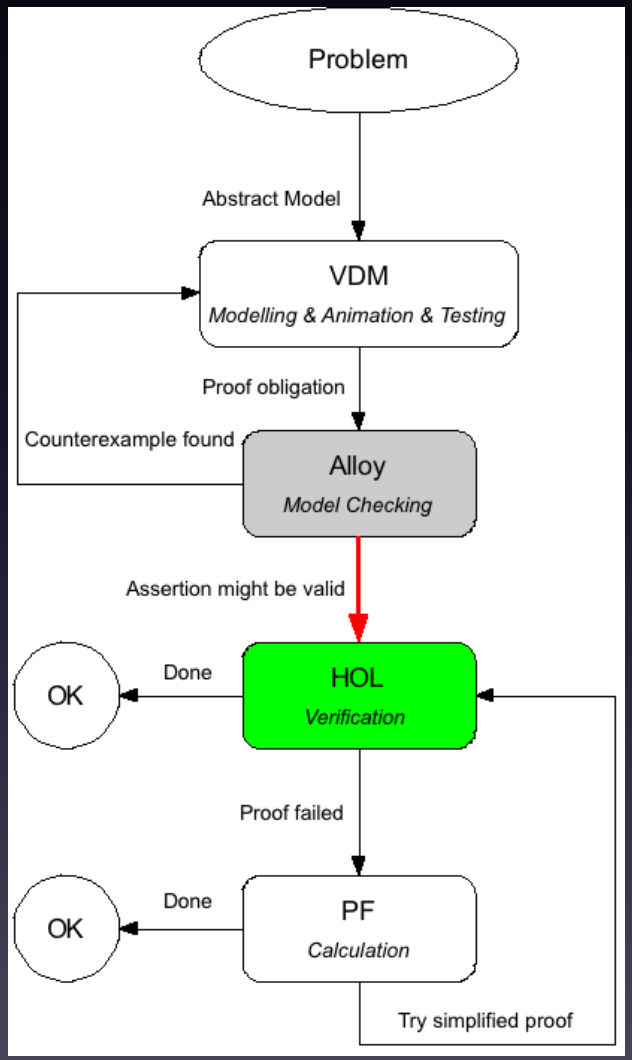
Counter-Example Found

- the assertion is invalid
- counter-example show **why** and **how**

Back to VDM++ Model

- error in model?
- too weak a pre-condition?
- too strong an invariant?

Attempt Automatic Proof



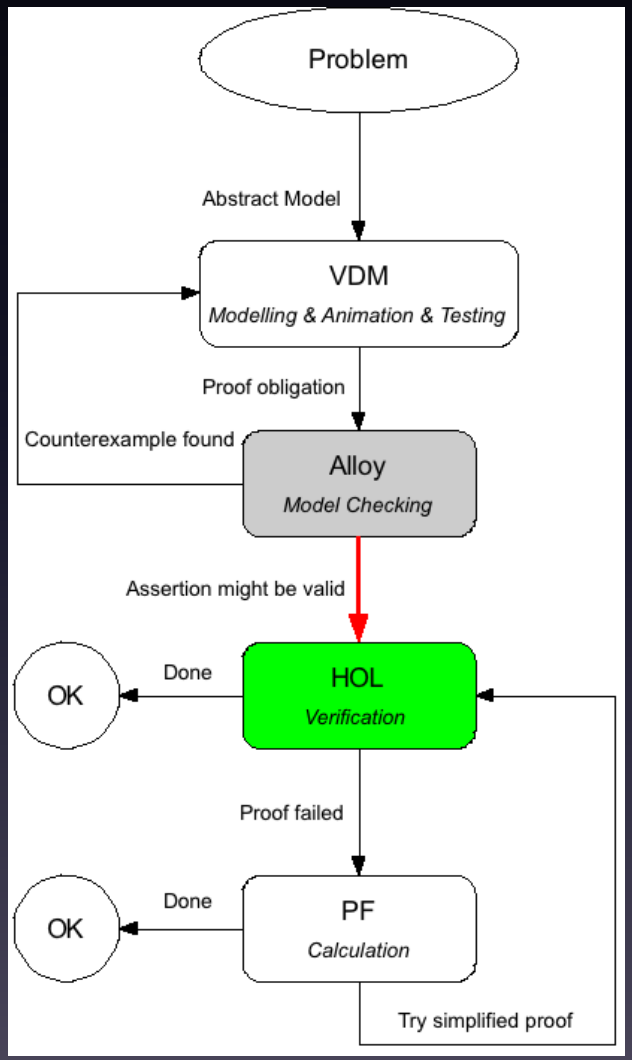
No Counter-Example Found

- assertion of PO **may be** valid
- gained increased confidence, but no certainty

Translate VDM++ to HOL

- VdmHolTranslator
  - translate model
  - generate proof commands for POs
- ask HOL to discharge proof

Attempt Automatic Proof



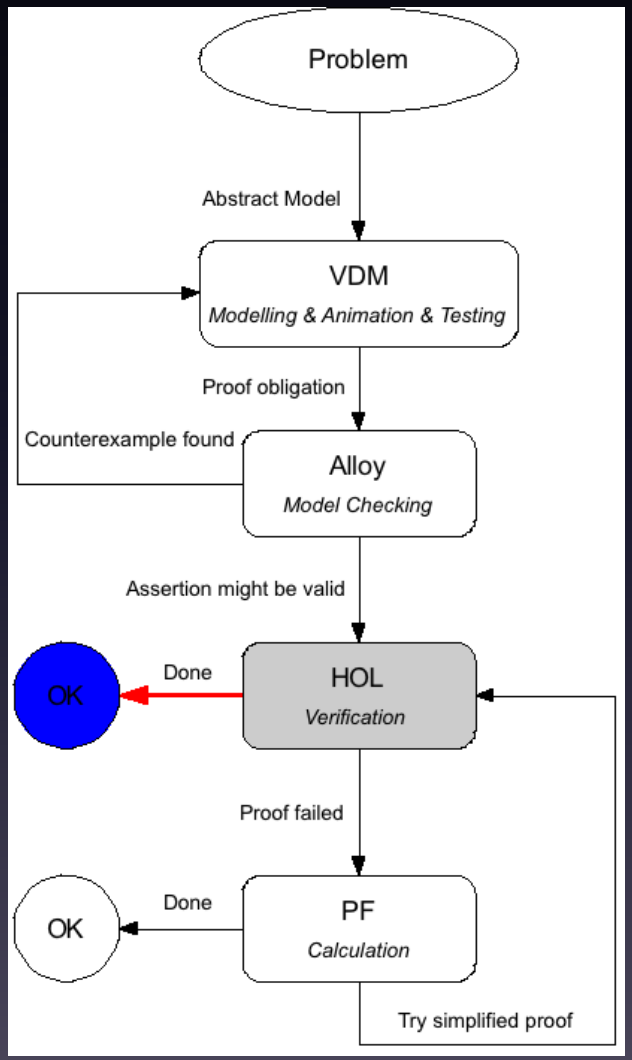
No Counter-Example Found

- assertion of PO **may be** valid
- gained increased confidence, but no certainty

Translate VDM++ to HOL

- VdmHolTranslator
  - translate model
  - generate proof commands for POs
- ask HOL to discharge proof

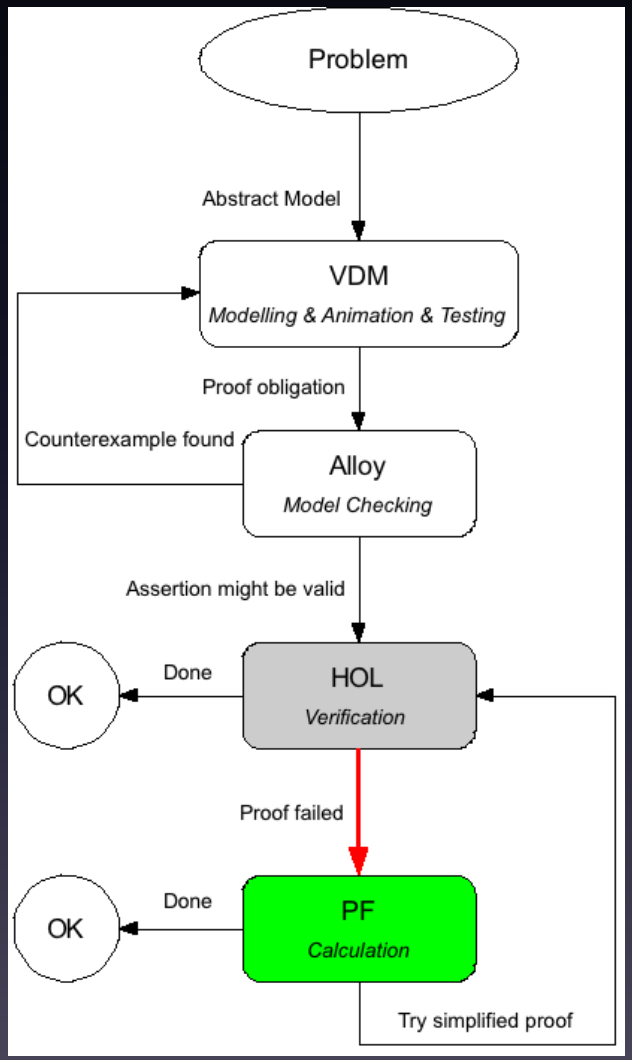
Proof Successful



HOL Completes the Proof

- PO discharged!

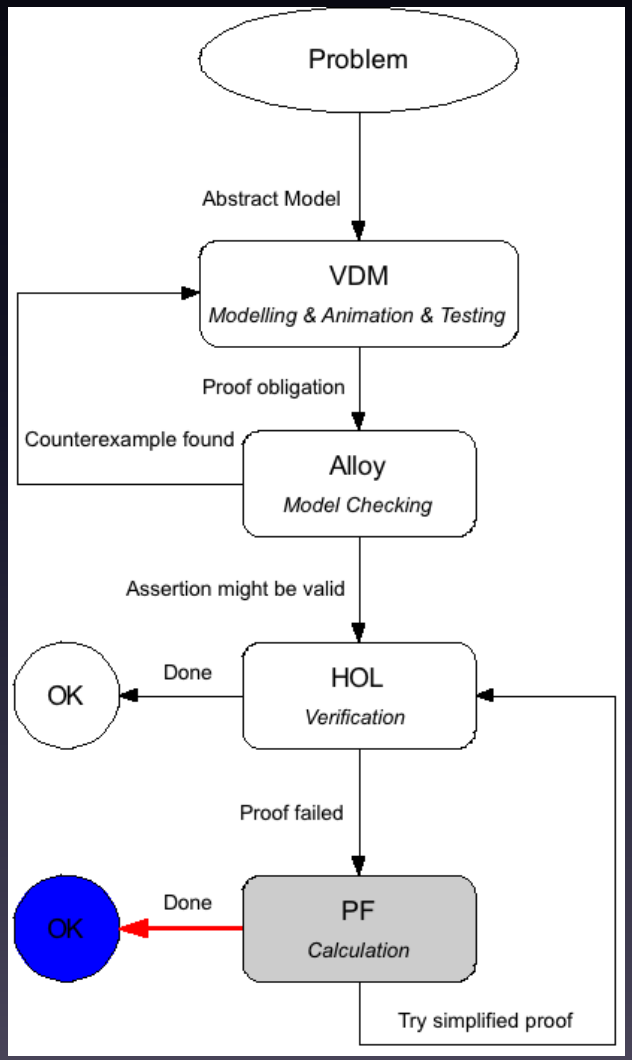
Proof Unsuccessful



HOL Proof Fails

- is PO invalid?
- is PO too complex?

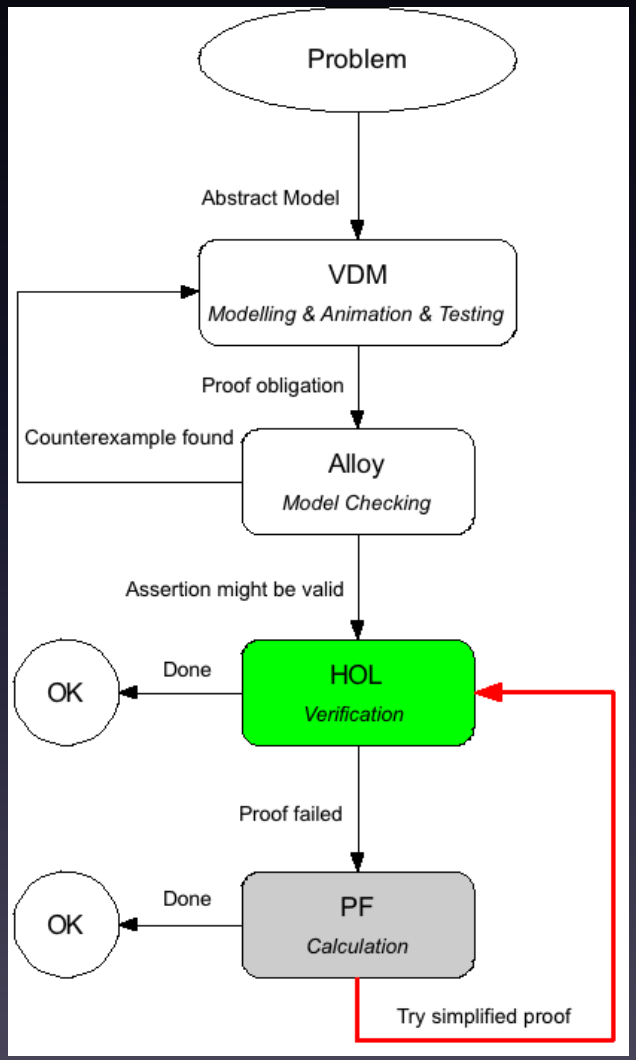
Proof Successful



PF Proof Successful

- PO discharged!

PO Decomposition



Try Simplified Proof

- decompose complex PO with PF-transform
- re-feed HOL with sub-proofs

Outline

Why VDM-HOL first

Integrity Checker

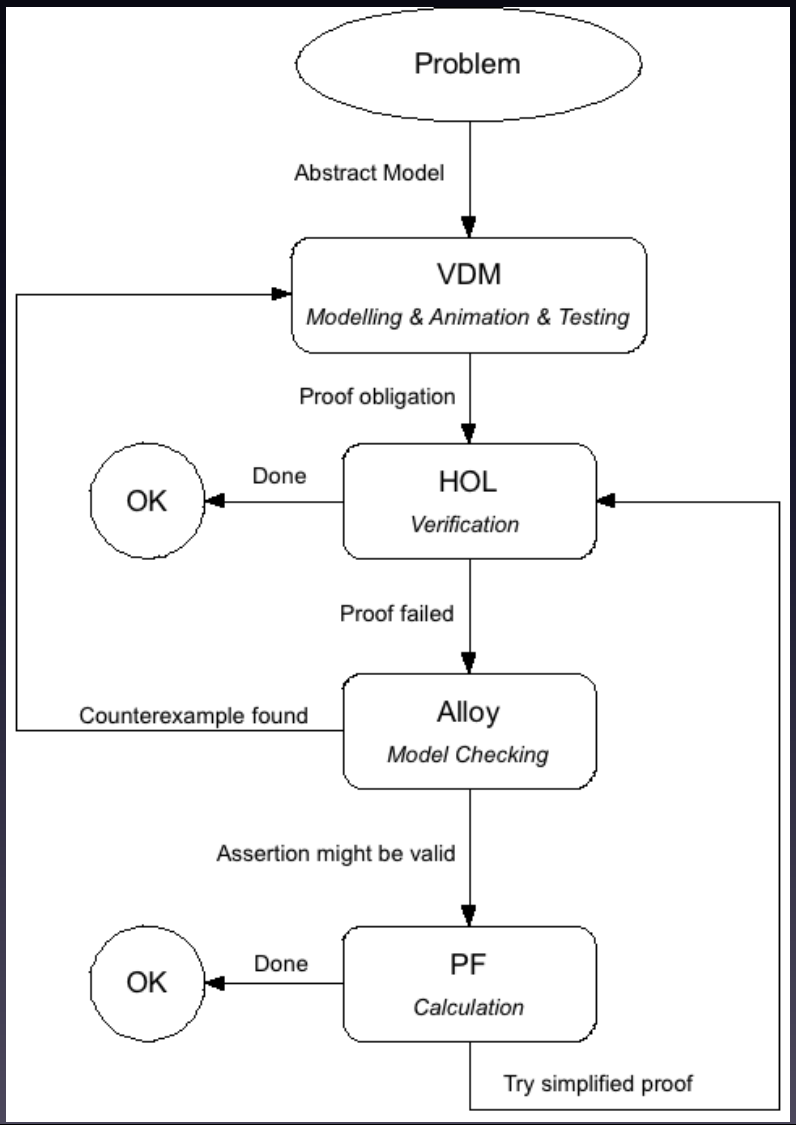
VDMTools generates **POs** (but doesn't discharge them)

Automatic Proof Support

Generates HOL from a VDM++ model + POs (developed by Sander Vermolen):

- supports a subset of the VDM++ syntax
- specialized proof tactics that can discharge proofs in HOL
- still under development

VDM-HOL-Alloy strategy



- first go for proofs in HOL
- then model check (if needed)
- **NB:** as earlier on, Alloy models have to be written by hand

Outline

File System Layer — FileStore

VDM++

```
FileStore = map Path to File
inv fileStore ==
  forall path in set dom fileStore &
    let parent = dirName(path) in
    isElemFileStore(parent, fileStore) and
    isDirectory(fileStore(parent));
```

File System Layer — FileStore

Translated to HOL

```
Hol_datatype `System
  = <| table     :((num, OpenFileDescriptor) fmap);
       fileStore:((Path, File) fmap) |>`;
```

```
Define `inv_FileStore (inv_FileStore_subj:((Path, File) fmap
    )) =
  (let fileStore = inv_FileStore_subj in
   (!uni_1_var_1.
     ((((uni_1_var_1 IN (FDOM fileStore) )  /\
      (?  path.(path  = uni_1_var_1 )) )  /\ T ) ==>
  (let path = uni_1_var_1 in
   (((dirName path) IN (FDOM fileStore) ) /\
```

File System Layer — FileStore

Alloy

```
sig FileStore {
  map: Path -> File
}
```

```
pred FileStoreInvariantVDM[fs: FileStore]{
  RelCalc/Simple[fs.map, File]           and
  RelCalc/Injective[fs.map, Path]        and
  PathInvariantVDM[RelCalc/dom[fs.map]] and
  FileInvariantVDM[RelCalc/rng[fs.map]] and
  FileStoreInvariant[fs]
}
```

File System Layer — FileStore

Alloy

```
pred FileStoreInvariant[fs: FileStore]{
  all path: RelCalc/dom[fs.map] {
    isElemFileStore[path.dirName,fs] and
    isDirectory[fs.map[path.dirName]]
  }
}
```

Outline

File System Layer — FS_Delete_FileDir_FileStore

VDM++

```
private
FS_DeleteFileDir_FileStore: FileStore * set of Path ->
    FileStore
FS_DeleteFileDir_FileStore(fileStore, paths) ==
  paths <-: fileStore
pre forall path in set dom fileStore &
    dirName(path) in set paths => path in set paths;
```

Outline

FileStore Invariant Preservation PO

- Is the FileStore Invariant still valid after excuting FS_DeleteFileDir_FileStore?

Generated PO

```
forall fileStore : FileStore, paths : set of Path &
  (forall path in set dom (fileStore) &
     dirName(path) in set paths => path in set paths)
  => inv_FileStore(paths <-: fileStore)
```

FileStore Invariant Preservation PO

- Let's Model Check this PO

Alloy equivalent

```
all fs,fs': FileStore, paths: set Path {
  FileStoreInvariantVDM[fs] and
  PathInvariantVDM[paths]    => (
    (all path : RelCalc/dom[fs.map] |
       path.dirName in paths => path in paths) and
     fs'.map = fs.map - (paths -> paths.(fs.map))
      => FileStoreInvariantVDM[fs']
  )
}
```

`FileStore` Invariant Preservation PO

As a matter of fact . . .

- HOL fails to discharge this PO
- Alloy doesn't find any counter-examples
- PF-transformed pen-and-paper proof required
- PF-transform removes variables and quantifiers from predicates — everything becomes a relation

FileStore Invariant

PF-transform blends particularly well with Alloy:

- Alloy is a relational language :-)
- Recall invariant PW definition:

Alloy PW FileStore invariant

```
pred FileStoreInvariant[fs: FileStore]{
  all path: RelCalc/dom[fs.map] {
    isElemFileStore[path.dirName,fs] and
    isDirectory[fs.map[path.dirName]]
  }
}
```

FileStore invariant

- PF version of the invariant is much shorter — in Alloy reads as follows:

Alloy PF FileStore invariant

```
pred FileStoreInvariant[fs: FileStore]{
  (fs.map).(File->Directory)
  in (dirName).(fs.map).attributes.fileType
}
```

FS_DeleteFileDir_FileStore Pre-Condition

Using the relational calculus, one easily calculates WP for FileStore invariant to be maintained – details in [Oli08]:

Alloy PF weakest pre-condition

```
pred pre_FS_DeleteFileDir_FileStorePF[fs:FileStore,paths:set
     Path] {
  (((Path - paths)->Path) & iden).(fs.map)
  in dirName.((Path - paths)->File)
}
```

FS_DeleteFileDir_FileStore Pre-Condition

Corresponding WP in PW Alloy

```
pred pre_FS_DeleteFileDir_FileStorePW[fs:FileStore,paths:set
     Path] {
  all path: RelCalc/dom[fs.map] {
    path.dirName in paths => path in paths
  }
}
```

FS_DeleteFileDir_FileStore Pre-Condition

Checking *PF* <=> *PW*

```
assert pw_equiv_pf {
  all fs: FileStore, paths: set Path {
    pre_FS_DeleteFileDir_FileStorePW[fs,paths] <=>
        pre_FS_DeleteFileDir_FileStorePF[fs,paths]
  }
}
check pw_equiv_pf for 20
```

FS_DeleteFileDir_FileStore Pre-Condition

Back to VDM

```
FS_DeleteFileDir_FileStore: FileStore * set of Path ->
    FileStore
FS_DeleteFileDir_FileStore(fileStore, paths) ==
  paths <-: fileStore
pre forall path in set dom fileStore &
    dirName(path) in set paths => path in set paths;
```

Outline

"All-in-one" Verification Strategy

- working with three different technologies is harder but worthwhile
- learning a lot on verification tool interoperability
- different tools show different aspects of the problem
- VDM-Alloy-HOL **complement** each other

VDM-HOL

The only automatic step of the Verification Process:

Automatic Proof Support

- still "semi" automatic
- on-going work towards increasing automation

Also researching on

- how to use HOL for *Weakest Pre-Condition* calculation

VDM-Alloy translation

Automatic bidirectional conversion would bring great benefit:

- need to keep models consistent
- need for uniform rules across translations
- Alloy more abstract (declarative) than VDM

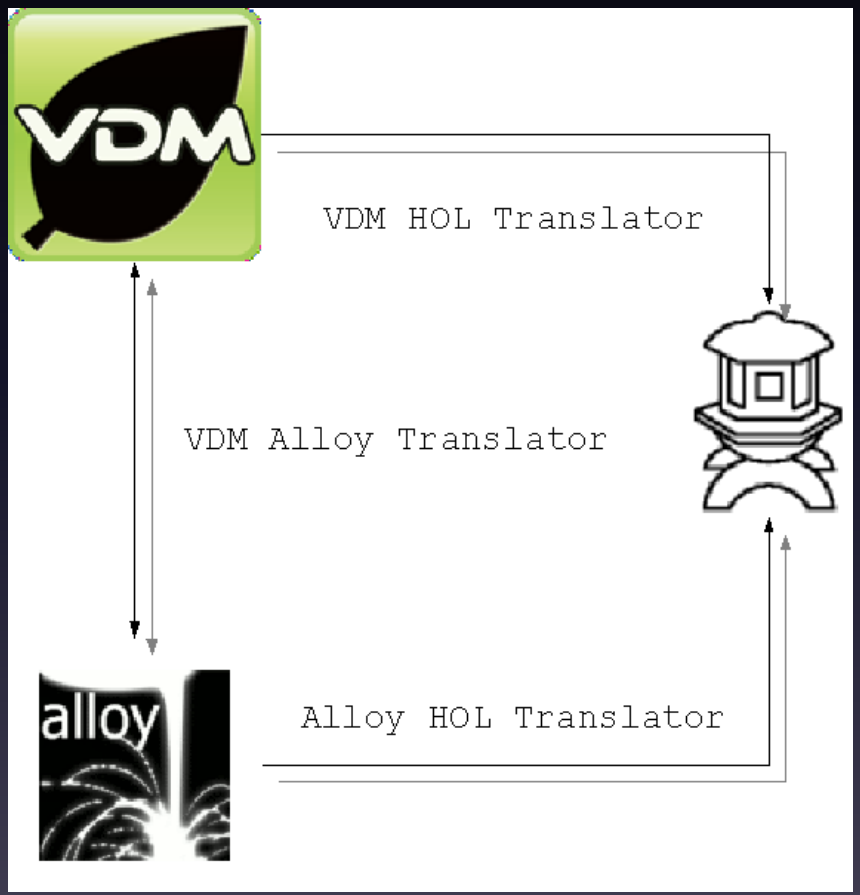
Other lessons learned from interoperability

Verifying complex models in more than one tool calls for code **slicing**:

- one PO at a time ("**single PO, multiple tool**")
- need to isolate the smallest model which accommodates given PO in each tool / notation (slice)

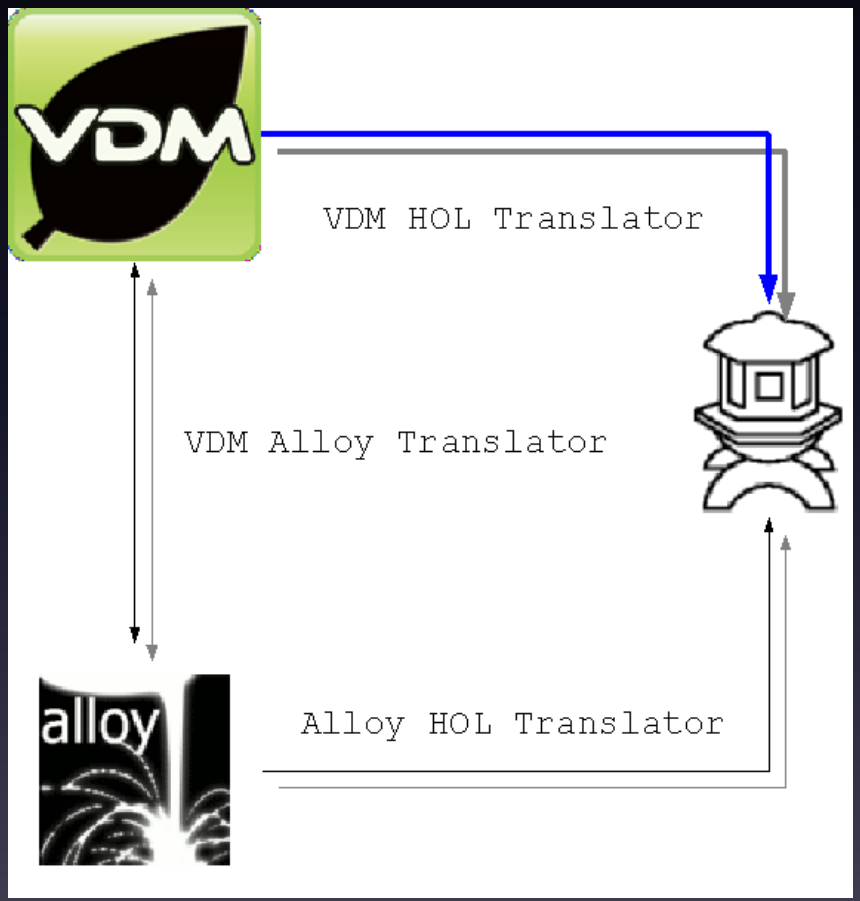
Outline

Translations



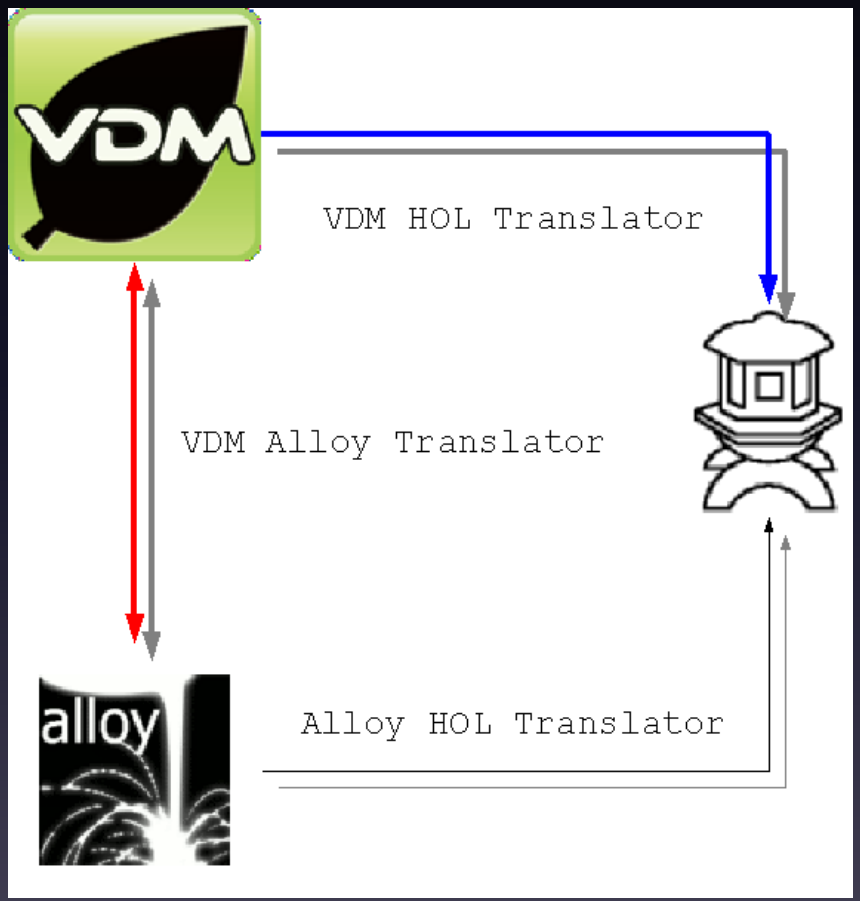
Very useful for verification

Translations



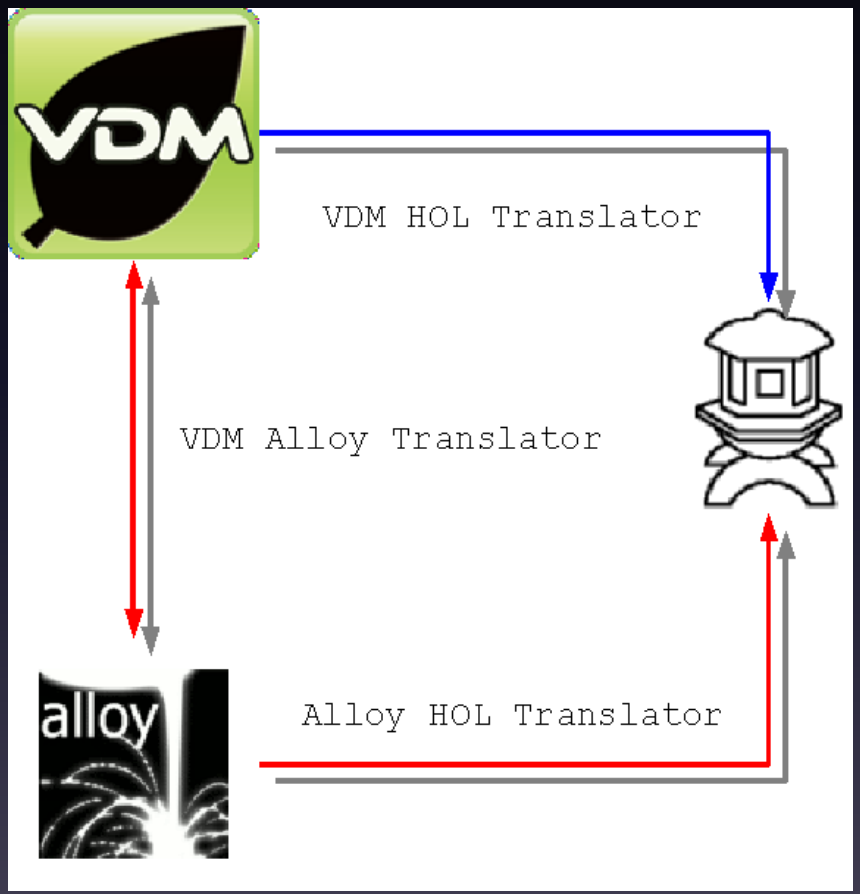
Already automated by the VdmHolTranslator

Translations



- would allow for direct model checking in Alloy
- will remove the need to synchronize separate VDM++ and Alloy models

Translations



- Alloy-HOL inter-operation interesting on its own
- increase the level of confidence in all models

Thank You

Thank you for your attention

Work has just started

- everyone interested in the approach is welcome on board!
- `http://twiki.di.uminho.pt/twiki/bin/view/Research/VFS/`

Questions

VERIFYING INTEL'S FLASH FILE SYSTEM CORE

Miguel Ferreira and Samuel Silva
University of Minho
{pg10961,pg11034}@alunos.uminho.pt

Deep Space lost contact with Spirit on 21 Jan 2004, just 17 days after landing.

Initially thought to be due to thunderstorm over Australia.

Spirit transmited an empty message and missed another communication session.

After two days controllers were surprised to receive a relay of data from Spirit.

Spirit didn't perform any scientific activities for 10 days.

This was the most serius anomaly in four-year mission.

Fault caused by Spirit's FLASH memory subsystem

Why formal methods?
Software bugs cost millions of dolars.

What we can do?
Build abstract models (VDM).
Gain confidence on models (Alloy).
Proof correctness (HOL & PF-Transform).

Acknowledgments:
Thanks to José N. Oliveira for its valuable guidance and contribution on Point-Free Transformation.
Thanks to Sander Vermolen for VDM to HOL translator support.
Thanks to Peter Gorm Larsen for VDMTools support.


Intel Corporation.
Intel Flash File System Core Reference Guide, October 2004.
Doc. Ref. 304436-001.


B.S. Dias and M.A. Ferreira.
Nand flash interface specification.
Technical report, University of Minho, July 2007.


DIUM/CCTC.
Verifiable file system project.
Website:
`http://twiki.di.uminho.pt/twiki/bin/view/Research/VFS/WebHome`.


T. Hoare and J. Misra.
Verified software: theories, tools, experiments-vision of a grand challenge project.

Proceedings of IFIP working conference on Verified Software: theories, tools, experiments, 2005.


Rajeev Joshi and Gerard Holzmann.
A mini challenge: build a verifiable filesystem.
Formal Aspects of Computing, 19(2):269–272(4), June 2007.


J.N. Oliveira.
Theory and applications of the PF-transform, Feb. 2008.
Tutorial at LerNET'08, Piriápolis, Uruguay (slides available from the author's website). Post workshop full text intended for LNCS publication is under way.


Samuel Silva et al.
Verified file-system v1.0.
Technical report, University of Minho, September 2007.