Automatically Discharging VDM Proof Obligations

Formal Methods 2008: VDM-Overture Workshop

Sander Daniël Vermolen

May 24, 2008

Department of Software Technology

TU Delft
Delft University of Technology

This work

Thesis at

Radboud University Nijmegen

- Jozef Hooman – Radboud University Nijmegen
- Peter Gorm Larsen – Engineering College of Århus
- Frits Vaandrager – Radboud University Nijmegen
- John Fitzgerald – Newcastle University

Outline

- Domain & Goals
- Approach
- Translation
- Proof
- Results & Concluding remarks

TU Delft

Arbitrary code

isPalindrome : seq of char -> bool
isPalindrome (pal) ==
    if *len* pal = 0 **then** *true*;
    else
        pal(1) = pal(*len* pal)
        &&
        isPalindrome(subSequence(pal, 2, len pal - 1))

madam

racecar

testset

TU Delft

Arbitrary code

```
isPalindrome : seq of char -> bool
isPalindrome (pal) ==
    if len pal = 0 then true;
    else
        pal(1) = pal(len pal)
        &&
        isPalindrome(subSequence(pal, 2, len pal - 1))
```

madam

racecar

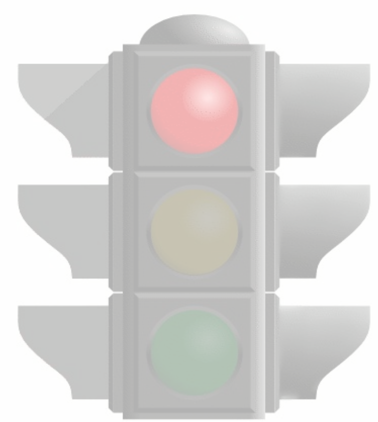
testset

“a”

TU Delft

Model Inconsistencies - VDM++

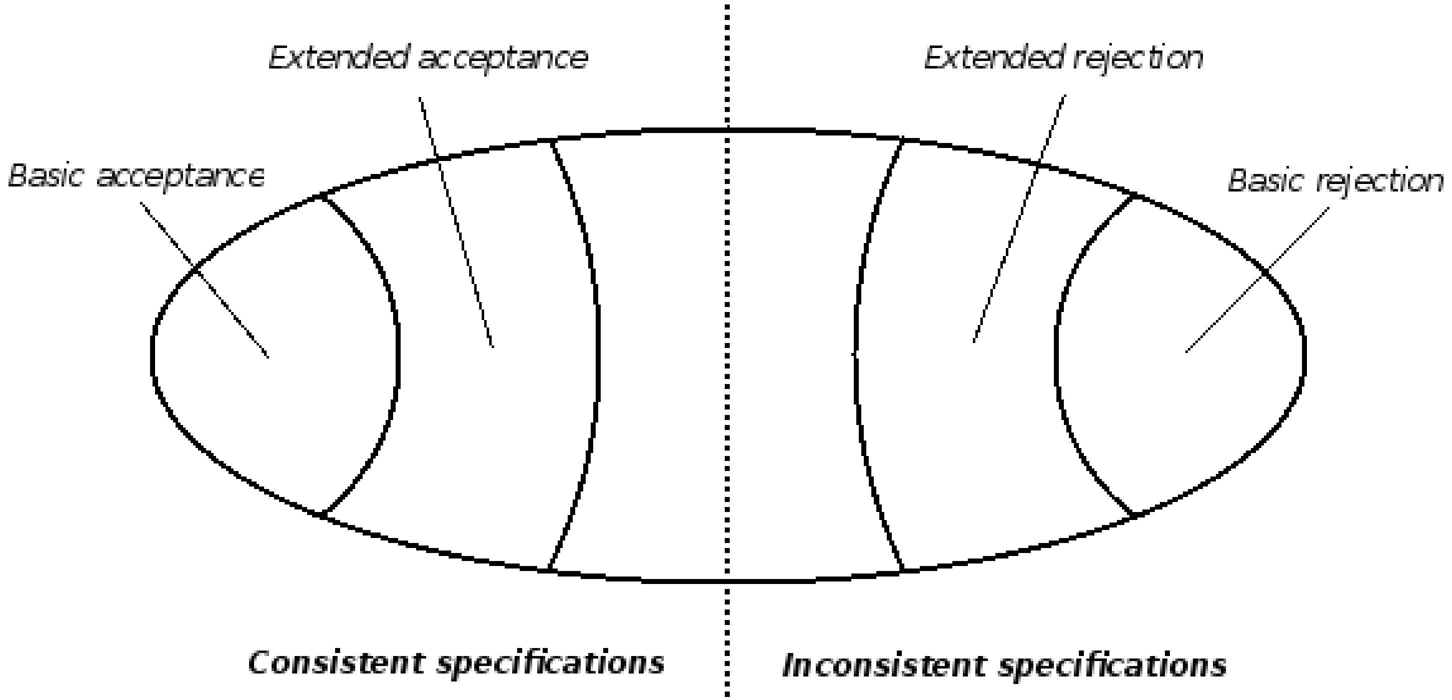
```
Color = <red> | <yellow> | <green>;
turnTo (..., <purple>);


turnTo(
      mk_TrafficLight(<red>),
      <red>
)


turnTo(x, <red>)
```



Model inconsistencies - Characteristics

- Trigger run-time errors
- Detectable using only the model

Model inconsistencies - Classes

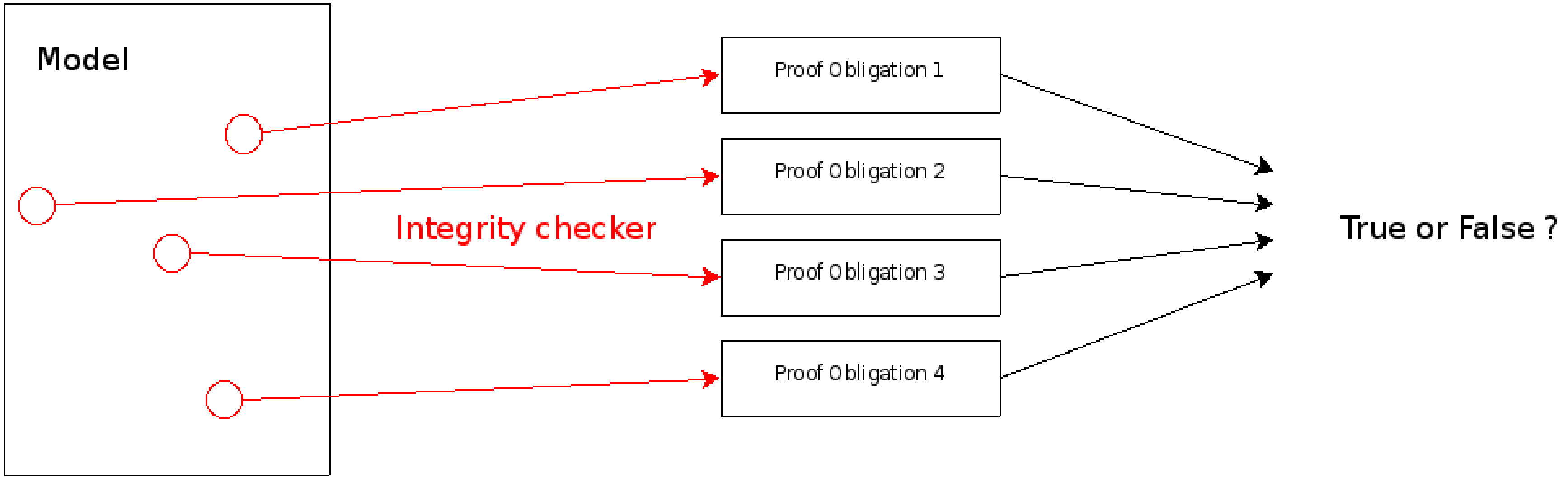


TUDelft

Model inconsistencies - Prevention

- Testing
  - Completeness
- Proof
  - Limited number of types of inconsistencies
  - Using proof obligations

Proof Obligations

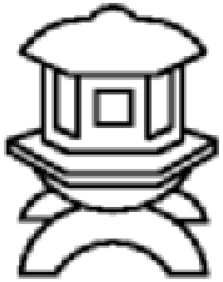


Theorem provers

- Theorems
- Proof search
  - Interactive
  - Automatic
- Tactics

Theorem provers - HOL

- Higher Order Logic (version 4)
  - Small axiom base
  - Adding theorems by proof only
  - Large number of libraries



TU Delft

Goal

Automating (as far as possible) the discharging of proof obligations generated by the integrity examiner

1. VDM++ to HOL translation
2. Automated proof of the Proof Obligations

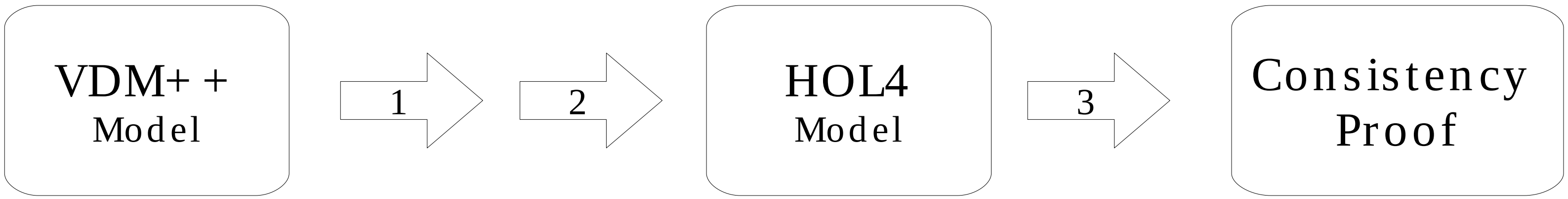
Outline

- Domain & Goals
- **Approach**
- Translation
- Proof
- Results & Concluding remarks

TU Delft

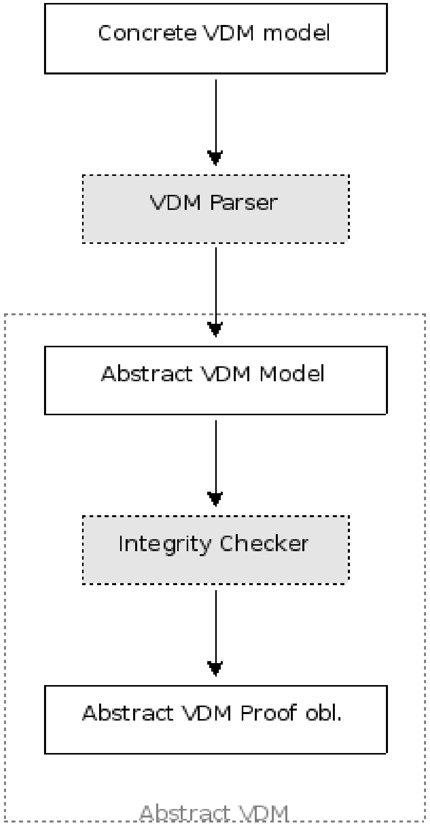
Architecture

1. Preparation
2. Translation
3. Proof attempts



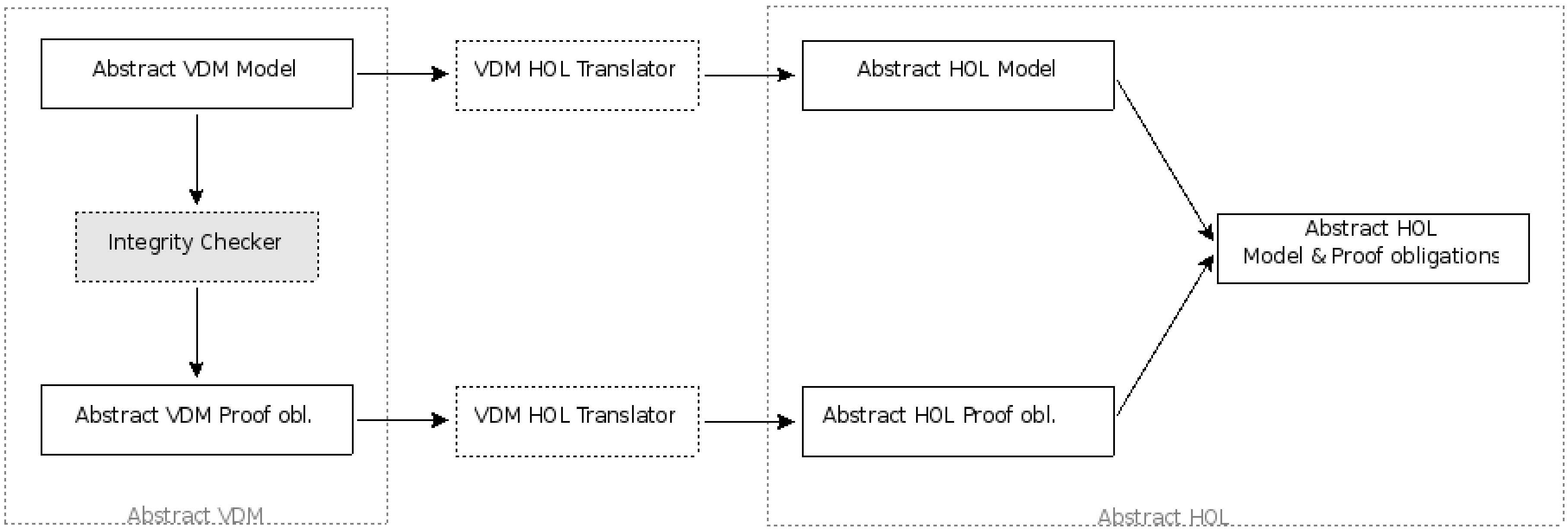
TU Delft

Architecture - Preparation

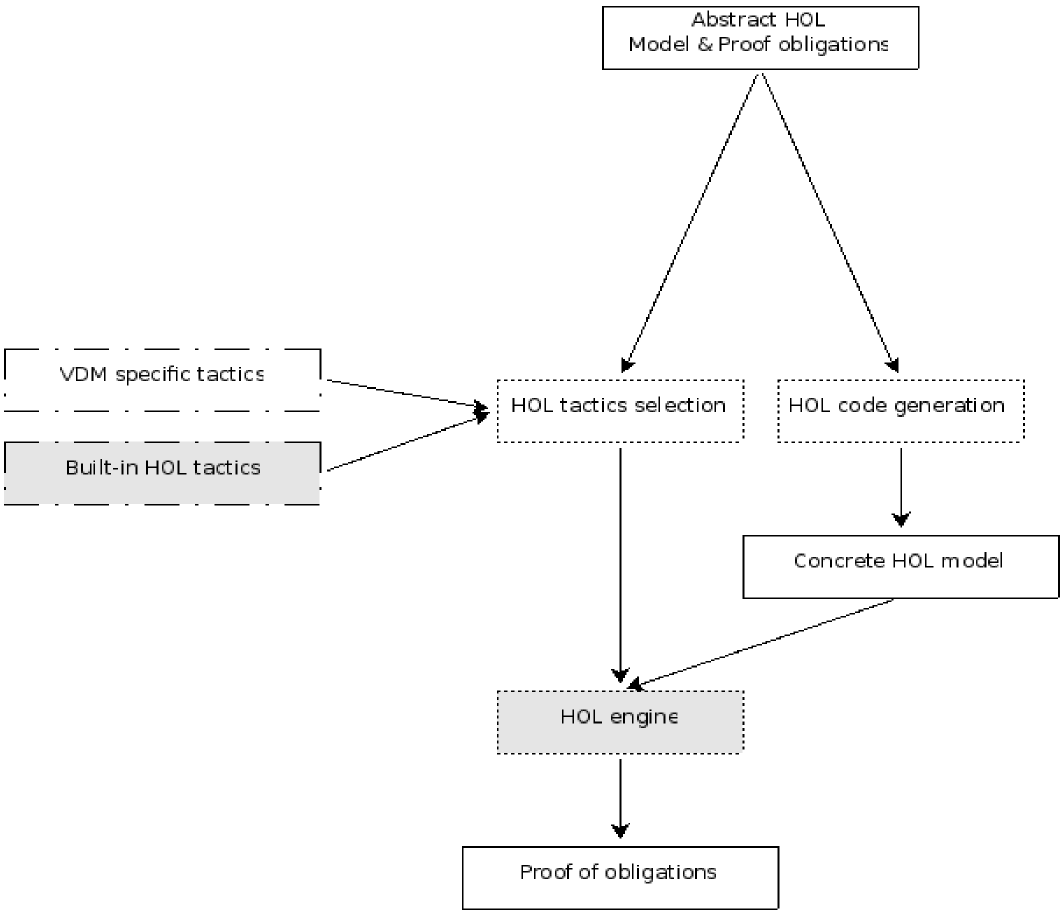


TUDelft

Architecture - Translation



Architecture - Proof



TU Delft

Outline

- Domain & Goals
- Approach
- **Translation**
- **Proof**
- Results & Concluding remarks

Translation - Types

$$\langle T_1 * T_2 * \cdots * T_n \rangle = \langle T_1 \rangle * \langle T_2 \rangle * \cdots * \langle T_n \rangle$$

$$\langle \text{map } T_d \text{ to } T_r \rangle = (\langle T_d \rangle | - > \langle T_r \rangle)$$

$$\langle T_1 * T_2 * \cdots * T_n\text{-}{>} T_{n+1} \rangle = \langle T_1 \rangle * \langle T_2 \rangle * \cdots * \langle T_n \rangle\text{-}{>}\langle T_{n+1} \rangle$$

Translation - Expressions

$$\langle|\text{Identifier}|\rangle = \text{Identifier}$$

$$\langle|\text{if } P \text{ then } E_1 \text{ else } E_2|\rangle = \text{if } \langle|P|\rangle \text{ then } \langle|E_1|\rangle \text{ else } \langle|E_2|\rangle$$

$$\langle|(E_f \;\; E_{P_1}, \ldots, E_{P_n})|\rangle = \langle|E_f|\rangle \; (\langle|E_{P_1}|\rangle) \ldots (\langle|E_{P_n}|\rangle)$$

Translation - Complications

- Type management
  - Type definition using existing type vs no type definition
  - Translating invariants
- Partiality
- Patterns

```
let
        mk_(a, b, 3, c) = mk_(1, 2, 3, 4)
in       ...
```

- Dependencies

Proof – Domain checking

- *Domain checking*
- Subtype checking
- Satisfiability of implicit definitions
- Termination

```
factorial (x) ==
      if    x = 0 then 1
      else x * factorial(x – 1)
pre x >= 0
```

$$\forall_{x:int}.(\text{pre\_factorial}(x) \wedge x \neq 0) \Rightarrow \text{pre\_factorial}(x-1)$$
$$\forall_{x:int}.(x \geq 0 \wedge x \neq 0) \Rightarrow (x-1) \geq 0$$

TU Delft

Proof – Domain checking

- TAUT_TAC
- MESON_TAC
- DECIDE_TAC
- VDM_ARITH_TAC
- REDUCE_TAC

```
(! val:num too:ind frm:ind wrld:AbWorld.(T ==> ((((((\x y . ~ (x =
    y)) frm too) /\ (frm IN (FDOM wrld.abPurses) ) ) /\ (too IN
    (FDOM wrld.abPurses) ) ) /\ ((GetBalance (FAPPLY
    wrld.abPurses frm)) >= val ) ) ==> (let RESULT = (let
    newFrm = (ReduceBalance (FAPPLY wrld.abPurses frm) val)
    and newTo = (IncreaseBalance (FAPPLY wrld.abPurses too)
    val) in (make_AbWorld wrld.authentic ((\x y . FUNION y x)
    wrld.abPurses ((FEMPTY |+ (too,newTo) ) |+ (frm,newFrm) ))))
    in (((((GetTotal (FAPPLY RESULT.abPurses frm)) + (GetTotal
    (FAPPLY RESULT.abPurses too)) ) = ((GetTotal (FAPPLY
    wrld.abPurses frm)) + (GetTotal (FAPPLY wrld.abPurses too)) )
    ) /\ (((GetBalance (FAPPLY RESULT.abPurses frm)) +
    (GetBalance (FAPPLY RESULT.abPurses too)) ) = ((GetBalance
    (FAPPLY wrld.abPurses frm)) + (GetBalance (FAPPLY
    wrld.abPurses too)) ) ) ) ==> (! name.(((name IN ((FDOM
    RESULT.abPurses) DIFF {frm;too} ) ) /\ T ) ==> (((GetBalance
    (FAPPLY wrld.abPurses name)) = (GetBalance (FAPPLY
    RESULT.abPurses name)) ) ==> (name IN (FDOM
    RESULT.abPurses) ) ) )) )) ) ))
```


TU Delft

Proof – Subtype checking

- Domain checking
- *Subtype checking*
- Satisfiability of implicit def.
- Termination

```
types:
  specialNat = nat
  inv x == x <> 2
functions:
  sum : specialNat * specialNat -> specialNat
  sum (x, y) ==
    x + y;
```

$$\forall_{x:specialNat, y:specialNat}.\mathrm{inv\_specialNat}(x+y)$$
$$\forall_{x:specialNat, y:specialNat}.(x+y) \neq 2$$

TU Delft

Proof – Subtype checking

Simplification support

- Stateful simplification
- Stateless simplification

Decision support

- Pre-defined theorems
- Custom theorems

Proof – Satisfiability

- Domain checking
- Subtype checking
- *Satisfiability of implicit def.*
- Termination

```
sqrt (x : real) r : real
pre x >= 0
post r * r = x;
```

$$\forall_{x:real}.\mathrm{pre\_sqrt}(x) \rightarrow \exists_{r:real}.\mathrm{post\_sqrt}(x, r)$$
$$\forall_{x:real}.x \geq 0 \rightarrow \exists_{r:real}.r \cdot r = x$$

TUDelft

Proof – Termination

- Domain checking
- Subtype checking
- Satisfiability of implicit definitions
- *Termination*

Results

- 15 case studies
- 4 significant case studies

Category	# valid obligations	# proved
Domain checking	37	37
Subtype checking	19	14
Satisfiability of implicit definitions	6	5

TUDelft

Conclusions

Translation

- Correctness of translation
- Correctness of implementation
- Object orientation

Proof

- Relative high rate of success
- Time efficient

Future & Current research

- Extension of translation
- Extension of tactic
- Process automation
- New concepts
  - Operational semantics
  - User guided proof

www.overturetool.org