QuickCheck for VDM

Nick Battle, Markus Ellyton

Proof Obligations

- *QuickCheck* is a new tool to help analyse proof obligations (POs)

- POs highlight where undefined results could occur or conditions must hold
- Short VDM-SL boolean expressions, which should always be true
- Produced by *VDMTools* and *Overture/VDMJ* for many years
- But *no proof support available* until recently

- *Isabelle* plugin can translate and discharge some POs
- Powerful, but sophisticated, requiring expertise and multiple tools
- Ideally, we want seamless integration of proof support in our VDM tools
- *QuickCheck* plugin attempts to perform a fast, lightweight check of POs

Proof Obligations

It would be useful if we could *quickly* divide obligations into three categories:

- Those that can be disproved ("failed" with a counterexample)
- Those that are very likely to be true ("probably provable")
- Those that are neither of the above ("maybe valid")

A *direct evaluation* of the PO expression may help?

But we have to be careful about LPF/McCarthy logic!

Proof Obligations: Check by Execution?

```
fbool: set of bool -> real
fbool(s) ==
     if s <> {}
     then 1 / card s  <-- potential divide by zero?
     else 0;
```

```
fbool: non-zero obligation in 'DEFAULT' (test.vdmsl) at line 16:16
(forall s:set of bool &
 ((s <> {}) =>
   (card s) <> 0))
```

```
> print (forall s:set of bool & ((s <> {}) => (card s) <> 0))
= true
Executed in 0.002 secs.
```

Proof Obligations: Check by Execution?

```
fbool: set of bool -> real
fbool(s) ==
     1 / card s;  <-- potential divide by zero?
```

```
fbool: non-zero obligation in 'DEFAULT' (test.vdmsl) at line 16:16
(forall s:set of bool &
  (card s) <> 0)
```

```
> p (forall s:set of bool & (card s) <> 0)
= false
Executed in 0.002 secs.
```

Proof Obligations: Check by Execution?

```
fnat: set of nat -> real
fnat(s) ==
     if s <> {}
     then 1 / card s  <-- potential divide by zero?
     else 0;
```

```
fnat: non-zero obligation in 'DEFAULT' (test.vdmsl) at line 10:16
(forall s:set of nat &
 ((s <> {}) =>
   (card s) <> 0))
```

```
> print (forall s:set of nat & ((s <> {}) => (card s) <> 0))
Error 4: Cannot get bind values for type nat in 'DEFAULT' (console) at line 1:2
MainThread>
```

Proof Obligations: Check by Execution?

- So the VDMJ interpreter can evaluate POs, but not that helpful by itself

- But we can tweak the interpreter (as a special case, in POs):
  - to allow *finite subsets* of infinite types to be checked in *forall/exists*
  - to remember counterexample/witness values

- PO generate/eval wrapped up in a command called "quickcheck" (*abbr.* "qc")
  - The objective is to find counterexamples or witnesses by evaluation
  - And some cases may be "probably provable" by simple checks

- But which subset of infinite type bind values do we choose?
  - Several different *strategies* are possible - so pluggable
  - Either return type bindings to try, or an indication of (dis)proof

QuickCheck Strategies

- A strategy is passed:
  - the PO (its AST)
  - a list of its type binds
  - an execution Context (eg. for evaluating type invariants)

- A strategy returns:
  - type bind value lists (that *might be* counterexamples or witnesses)
  - a "*hasAllValues*" flag if all of the bindings' values were generated
  - a (dis)proved flag and message, if it is able to conclude this

- *QuickCheck* applies all enabled strategies, then evaluates the PO, looking for counterexamples (unless a strategy has claimed the PO is provable).

QuickCheck Built-in Strategies

Six strategies are built-in:

- *The fixed strategy -* returns a fixed set of values for every VDM type
- *The random strategy -* similar to fixed, but using a pseudo-random number generator
- *The trivial strategy -* looks for "trivial" forms, like <expression> => <expression>
- *The finite strategy -* checks whether all bindings are of finite types (and not too big)
- *The search strategy -* looks for eg. "x <> 0" then returns "x = 0" (naively)
- *The direct strategy -* ignores the PO itself, but looks at what it is trying to verify

More strategies can be added by putting a jar on the classpath.

QuickCheck Example - "qc" (VDMJ)

```
> qc
PO #1, PROVABLE by direct (body is total) in 0.002s
PO #2, FAILED (unsatisfiable) in 0.001s
----
T: invariant satisfiability obligation in 'DEFAULT' (test.vdm) at line 3:9
exists t : set of bool & ((card t) = 3)

PO #3, PROVABLE by direct (body is total) in 0.0s
PO #4, PROVABLE by witness q = 11 in 0.001s
PO #5, PROVABLE by trivial s <> [] in 0.001s
PO #6, PROVABLE by direct (body is total) in 0.0s
PO #7, MAYBE in 0.001s
PO #8, MAYBE in 0.001s
PO #9, FAILED in 0.002s: Counterexample: r = 1.25
----
h: subtype obligation in 'DEFAULT' (test.vdm) at line 16:5
(forall r:real & pre_h(r) =>
 is_nat(r))

>
```

QuickCheck Example - "qr" (VDMJ)

```
> qr 9
=> print h(1.25)
Error 4065: Value 1.25 is not a nat in 'DEFAULT' (console) at line 1:1

> qr 2
=> print exists t : set of bool & ((card t) = 3)
= false
```

QuickCheck: Polymorphic Functions

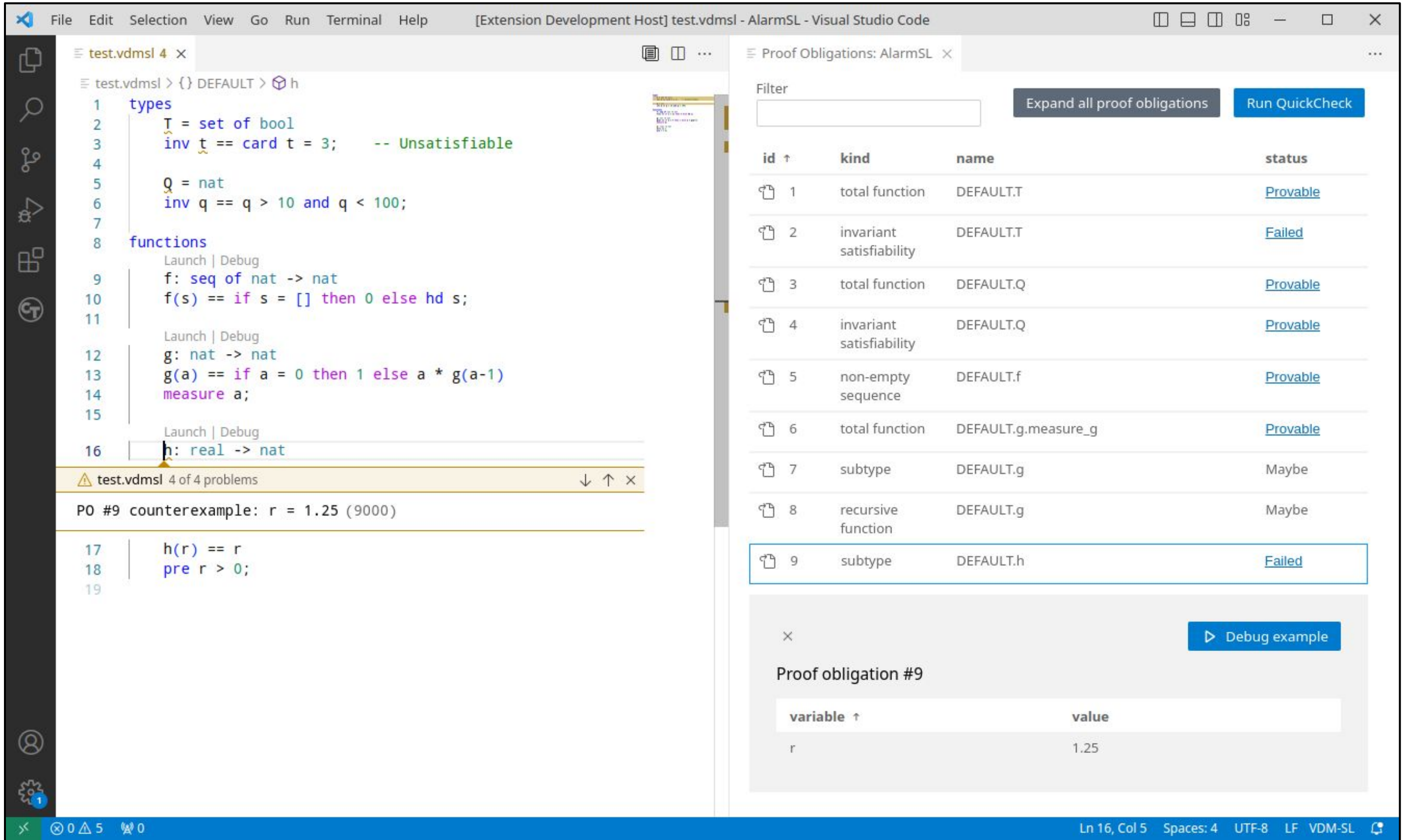
```
-- @QuickCheck @T = set of nat, set of bool;
f[@T]: seq of @T * nat -> @T
f(s, i) == s(i);

Proof Obligation 1: (Unproved)
f: sequence apply obligation in 'DEFAULT' (test.vdmsl) at line 4:16
(forall s:seq of (@T), i:nat &
  i in set inds s)

> qc 1
PO #1, FAILED in 0.003s: Counterexample:  i = 0, s = [], T = set of (nat)
----
f: sequence apply obligation in 'DEFAULT' (test.vdmsl) at line 4:16
(forall s:seq of (@T), i:nat &
  i in set inds s)

> qr 1
=> print f[set of (nat)]([], 0)
Error 4064: Value 0 is not a nat1 in 'DEFAULT' (test.vdmsl) at line 4:16
4:   f(s, i) == s(i);
```

QuickCheck Example - *VDM-VSCode*


File Edit Selection View Go Run Terminal Help
[Extension Development Host] test.vdmsl - AlarmSL - Visual Studio Code
test.vdmsl 4
test.vdmsl > {} DEFAULT > h
Launch | Debug
Proof Obligations: AlarmSL
Filter
Expand all proof obligations
Run QuickCheck
Debug example
0 ⚠ 5
0
Ln 16, Col 5
Spaces: 4
UTF-8
LF
VDM-SL

```
types
    T = set of bool
    inv t == card t = 3;    -- Unsatisfiable

    Q = nat
    inv q == q > 10 and q < 100;

functions
    f: seq of nat -> nat
    f(s) == if s = [] then 0 else hd s;

    g: nat -> nat
    g(a) == if a = 0 then 1 else a * g(a-1)
    measure a;

    h: real -> nat
```

test.vdmsl 4 of 4 problems

```
PO #9 counterexample: r = 1.25 (9000)
```

```
    h(r) == r
    pre r > 0;
```

id ↑	kind	name	status
1	total function	DEFAULT.T	Provable
2	invariant satisfiability	DEFAULT.T	Failed
3	total function	DEFAULT.Q	Provable
4	invariant satisfiability	DEFAULT.Q	Provable
5	non-empty sequence	DEFAULT.f	Provable
6	total function	DEFAULT.g.measure_g	Provable
7	subtype	DEFAULT.g	Maybe
8	recursive function	DEFAULT.g	Maybe
9	subtype	DEFAULT.h	Failed

Proof obligation #9

variable ↑	value
r	1.25

Performance

	VDM-SL	VDM++	VDM-RT	Totals	%age
Specs #	50	51	13	114	
PO #	4964	2830	435	8229	
PROVABLE	878	323	37	1238	15.04%
by trivial	*141*	*91*	*3*	*235*	*2.86%*
by finite	*227*	*135*	*16*	*378*	*4.59%*
by witness	*109*	*30*	*7*	*146*	*1.77%*
by direct	*401*	*67*	*11*	*479*	*5.82%*
MAYBE	2077	781	108	2966	36.04%
FAILED (counterexample)	942	128	5	1075	13.06%
UNCHECKED	1057	1598	285	2940	35.73%
TIMEOUT (5s)	10	0	0	10	0.12%
					100.00%

	VDM-SL	VDM++	VDM-RT	Average (ms)
PROVABLE	4.06	4.37	1.7	3.38
FAILED	10.41	3.11	11.2	8.24
MAYBE	45.5	49.28	13.34	36.04

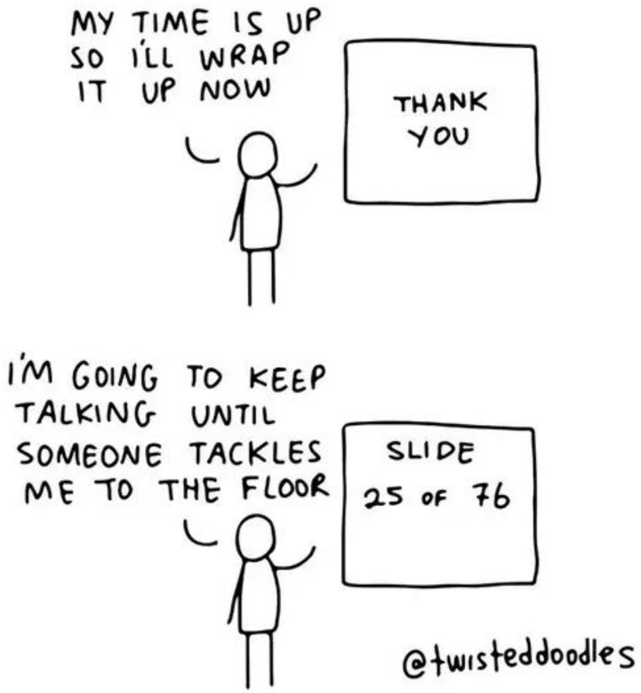
Future Directions

- *More strategies?*
  - Translate the PO to SMT-LIB (perhaps via Dafny)?
  - Strategies could return a *proved* status
  - Maybe use ML to identify counterexamples?

- *Improved analysis for UNCHECKED operation POs?*
  - Include relevant state in obligations
  - VDM++ and VDM-RT are a challenge

- *Better polymorphic type selection?*
  - Better checking of highly polymorphic specifications
  - Sensibly selecting type parameters to check

THE TWO TYPES OF SPEAKER AT A CONFERENCE


MY TIME IS UP SO I'LL WRAP IT UP NOW
THANK YOU
I'M GOING TO KEEP TALKING UNTIL SOMEONE TACKLES ME TO THE FLOOR
SLIDE 25 OF 76

@twisteddoodles