

ScottCheck: An Adventure in Symbolic Execution

Gergő Érdi

<http://unsafePerform.IO/>

Berlin Functional Programming Group, 13th October 2020.

1. Structure and Interpretation of Adventure Games

Text adventure games

```
[cactus@galaxy: ~]$
> no
YOU ARE STANDING AT THE END OF A ROAD BEFORE A SMALL BRICK BUILDING.
AROUND YOU IS A FOREST.  A SMALL STREAM FLOWS OUT OF THE BUILDING AND
DOWN A GULLY.

> enter building
YOU ARE INSIDE A BUILDING, A HELL HOUSE FOR A LARGE SPRING.

THERE ARE SOME KEYS ON THE GROUND HERE.

THERE IS A SHINY BRASS LAMP NEARBY.

THERE IS FOOD HERE.

THERE IS A BOTTLE OF WATER HERE.

> take keys
OK

> take bottle
OK

>
```

```
[cactus@galaxy: ~/prog/haskell/abv/1/1-sat/import/scottsk/games/adams]$
I'm in a sunny meadow

Obvious exits: South, East, West.

I can also see: Large sleeping dragon -
Sign here says "In many cases mud is good. In others..."

A voice BOOOOMS out:

Welcome to Adventure number: 1 "ADVENTURELAND".
In this Adventure you're to find "TREASURES" & store them away.

To see how well you're doing say: "SCORE"
Remember you can always say "HELP"

Tell me what to do ? go east

Tell me what to do ? score
I've stored 0 treasures. On a scale of 0 to 100, that rates 0 .

Tell me what to do ?
```


```
VICE (C64)
File Edit Snapshot Settings Help

1000
C64
VICE
000
000

Egy óriási mezőn vagy. Nyugat felé egy hatalmas épület körvonalai tűnnek fel.
>ny
Napfényes mezőn állsz. Nyugatra egy hatalmas kastélyt, délre egy kutat láatsz.
>_

604% cpu, 27 fps (warp)  CRT controls  Tape: 000  8:18.0  Mixer controls  Joysticks:+
```

```
Időregész - Rátka István (eredeti): Értelmező Információs Reprodukció
Erő: 25  Eredmény: 0  D K Ny



Időregész (Rátka István, 1987)
Információs reprodukció, Érdi Gergő, 2015.

Egy óriási mezőn vagy. Nyugat felé egy hatalmas épület körvonalai tűnnek fel.

> ny
Napfényes mezőn állsz. Nyugatra egy hatalmas kastélyt, délre egy kutat láatsz.

> lista
Nincs nálad semmi.

>|
```

Text adventure games

- Interactive fiction
- Text adventure games
 - Subset of interactive fiction: well-defined **win state**
 - (Debian 2.0 installer in 1998?)
 - Unwinnable games are faulty
- Under the hood
 - Game data + interpreter
 - *Infocom, ADS, Inform, ...*
 - *Adventure International*

Adventure International

- Scott Adams (unrelated to the Dilbert guy)
- Late '70s home computers
- Goal: move all items marked as *treasure* to the single location marked as the *treasury*
- SCORE command shows number of treasure collected. Win by issuing SCORE with max score.
- Game data: text file containing magic numbers and string literals

Example data file: Header

```
0
6      -- Number of items
10     -- Number of script lines
18     -- Dictionary size
5      -- Number of rooms
-1     -- Player carry capacity
3      -- Starting room
1      -- Max score
3      -- Word length
-1     -- Lamp capacity
7      -- Number of messages
1      -- Treasury room number
```

Example data file: Dictionary

```
"AUT" "ANY"  
"GO"  "NOR"  
"SCO" "SOU"  
"INV" "EAS"  
"LOO" "WES"  
"OPE" "UP"  
" "   "DOW"  
" "   "DOO"  
" "   "KEY"  
" "   "CRO"  
"GET" "COI"  
" "   ""  
" "   ""  
" "   ""  
" "   ""  
" "   ""  
" "   ""  
" "   ""  
" "   ""  
" "   ""  
"DRO" ""
```

Example data file: Rooms & messages

```
0 0 0 0 0 0 ""
0 2 0 0 0 0 "gorgeously decorated throne room"
1 0 3 0 0 0 "square chamber"
5 0 0 2 0 0 "gloomy dungeon"
3 0 0 0 0 0 "dungeon cell"
0 3 0 0 0 0 "damp, dismal crypt"
```

```
""
```

```
"I smell something rotting to the north."
```

```
"It's locked."
```

```
"OK"
```

```
"Vampire cowers away from the cross!"
```

```
"Vampire looks hungrily at me."
```

```
"Vampire bites me! I'm dead!"
```

```
"I'm not going anywhere near that vampire!"
```


Example data file: Items

```
"Sign says: leave treasure here, then say SCORE" 1  
"Wooden cross/CRO/" 2  
"Locked door" 3  
"Brass key/KEY/" 5  
"Open door leads south" 0  
"*Gold coin*/COI/" 4  
"Vampire" 5
```

Example data file: Script

```
300 0 0 0 0 0 9750 0
450 0 0 0 0 0 9900 0
600 0 0 0 0 0 9600 0
25 64 0 0 0 0 150 0
757 42 72 0 0 0 300 0
757 42 40 80 0 0 10803 9600
157 82 80 0 0 0 8164 0
100 122 21 0 0 0 600 0
100 122 26 0 0 0 750 0
25 122 26 0 0 0 963 0
1508 122 26 0 0 0 1050 0
```

Example data file: Script

757 42 72 0 0 0 300 0

- User input: OPE D00
- Condition: Item 2 "Locked door" in current room
- Condition: Item 3 "Brass key" not available
- Action: Print message 2 "It's locked"

The Grand Plan

- Martin Lester's talk at WPTE 2020:
 - *ScottFree*: Off-the-shelf game interpreter written in C
 - Manually “partially evaluated” for a given game, infinite loops replaced with bounded ones, passed to *CBMC*
 - *CBMC* output passed to an SMT solver
- My approach: bubblegum and duct tape
 - Idiomatic Haskell interpreter
 - Changed just so, to use symbolic values
 - In one week

2. Monad Transformers for Concrete Interpreters

Interactive interpreter

- Parse game data into algebraic datatype `GameData`

- Game state

```
data St = St
  { currentRoom    :: Int16
  , itemLocations  :: Array Int16 Int16
  }
```

- Stack of monad transformers:

```
type Engine =
  ReaderT GameData (WriterT [String] (State St))
```

```
step :: (Int16, Int16) -> Engine (Maybe Bool)
```

(Return value: win/fail state)

- Thin layer of IO: get input, parse with dictionary, print output

3. Reducability Among Fictional Problems

Satisfiability

- *SAT*: Boolean satisfiability problem
 - Given a Boolean formula with variables, is it satisfiable?
 - Trivially in NP: we can verify a given var assignment by eval
 - But also NP-hard!
 - 3-SAT (CNF SAT with ≤ 3 literals per clause): the OG NP-complete problem
- Mature solvers chock full of heuristics
- *SMT*: Satisfiability modulo theories (e.g. integer arithmetic)
- Represent full game logic as a boolean formula with user input as the variables

4. Symbolic Execution and Puzzle Testing

Symbolic execution

- What is the value of

`if x + (if b then 0 else 2) > x + 1 then 3 else 4`

Symbolic execution

- What is the value of

```
if x + (if b then 0 else 2) > x + 1 then 3 else 4
```

- If b has a concrete value, it's obvious.
- If b isn't known, we can represent the result symbolically:

```
if $x + (if $b then 0 else 2) > $x + 1 then 3 else 4
```

Symbolic execution

- What is the value of

```
if x + (if b then 0 else 2) > x + 1 then 3 else 4
```

- If b has a concrete value, it's obvious.
- If b isn't known, we can represent the result symbolically:

```
if $x + (if $b then 0 else 2) > $x + 1 then 3 else 4
```

- We can still make *some* progress:

```
if (if $b then $x + 0 else $x + 2) > $x + 1  
  then 3 else 4
```

```
if (if $b then $x + 0 > $x + 1 else $x + 2 > $x + 1)  
  then 3 else 4
```

Symbolic execution

- What is the value of

```
if x + (if b then 0 else 2) > x + 1 then 3 else 4
```

- If b has a concrete value, it's obvious.
- If b isn't known, we can represent the result symbolically:

```
if $x + (if $b then 0 else 2) > $x + 1 then 3 else 4
```

- We can still make *some* progress:

```
if (if $b then $x + 0 else $x + 2) > $x + 1  
  then 3 else 4
```

```
if (if $b then $x + 0 > $x + 1 else $x + 2 > $x + 1)  
  then 3 else 4
```

```
if (if $b then False else True) then 3 else 4
```

```
if $b then 4 else 3
```

5. Notions of Adventuring and Monads

SBV-based interpreter

- Change representation of St:

```
data St = St
  { currentRoom    :: SInt16
  , itemLocations  :: Array Int16 SInt16
  } deriving (Generic, Mergeable)
```

- Structure of items is static!
- Follow type errors
 - Num operations Just Work™, Boolean operations need SBV-specific versions
 - Mergeable instances for MTL monad transformers
- `if_then_else` using rebindable syntax!

```
ifThenElse :: (Mergeable a) => SBool -> a -> a -> a
ifThenElse = ite
```

SBV-based interpreter: Combinators

- Mergeable-based versions of monadic combinators

```
sWhen :: (Monad m, Mergeable (m ())) =>  
      SBool -> m () -> m ()
```

```
sWhen b act = if b then act else return ()
```

- Pattern matching on script code symbolically

```
sCase
```

```
  :: (Mergeable a)
```

```
  => SInt16
```

```
  -> [(Int16, a)] -> a -> a
```

```
sCase x cases def = go cases
```

```
  where
```

```
    go [] = def
```

```
    go ((k,v):kvs) =
```

```
      if x .== literal k then v else go kvs
```


Turning the crank

```
step :: (SInt16, SInt16) -> Engine (SMaybe Bool)
```

- *SBV's* Query monad allows incremental SMT queries
- We keep running `step` with new inputs, until it can be satisfied
- Satisfying input recovered from SMT solver's state

```
loopState
```

```
  :: (SymVal i)  
  => Query (SBV i)  
  -> s  
  -> (SBV i -> State s SBool)  
  -> Query [i]
```

Recovering interactive mode

- The only source of “really symbolic” values is the input to step
- Idea: feed interactive input as single possible value, this ensures output has a single possible value as well

Recovering interactive mode

- The only source of “really symbolic” values is the input to step
- Idea: feed interactive input as single possible value, this ensures output has a single possible value as well
- Same interpreter can be used for checking and interactive execution!

6. Demo: *Here's one I made earlier*

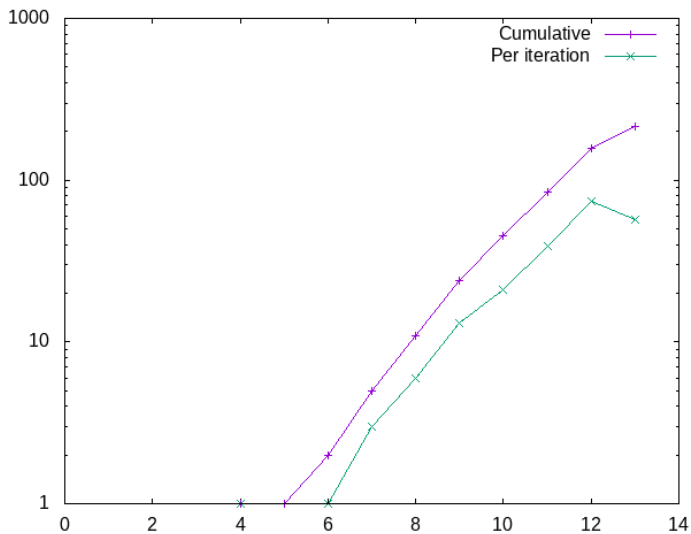
Demo: *ScottKit*, tutorial #4

```
Throne Room      Crypt
[sign]           [vampire, key]
|               |
|               |
Chamber-----Dungeon (starting room)
[cross]          [door]
                 =
                 |
                 Cell
                 [*coin*]
```

Demo: *ScottKit*, tutorial #4

```
00:00 Searching at depth: 1
...
02:38 Searching at depth: 14
03:35 Solution found:
03:35   1. GO WEST
03:35   2. GET CROSS
03:35   3. GO EAST
03:35   4. GO NORTH
03:35   5. GET KEY
03:35   6. GO SOUTH
03:35   7. OPEN DOOR
03:35   8. GO DOOR
03:35   9. GET COIN
03:35  10. GO NORTH
03:35  11. GO WEST
03:35  12. GO NORTH
03:35  13. DROP COIN
03:35  14. SCORE
```

Demo: *ScottKit*, tutorial #4



(check-sat)

<http://unsafePerform.IO/scottcheck/>

Special thanks to Levent Erkök for helping out with SBV.

Questions?