EECS498-008
Formal Verification of Systems Software

Material and slides created by

Jon Howell and Manos Kapritsos

Dafny Modules

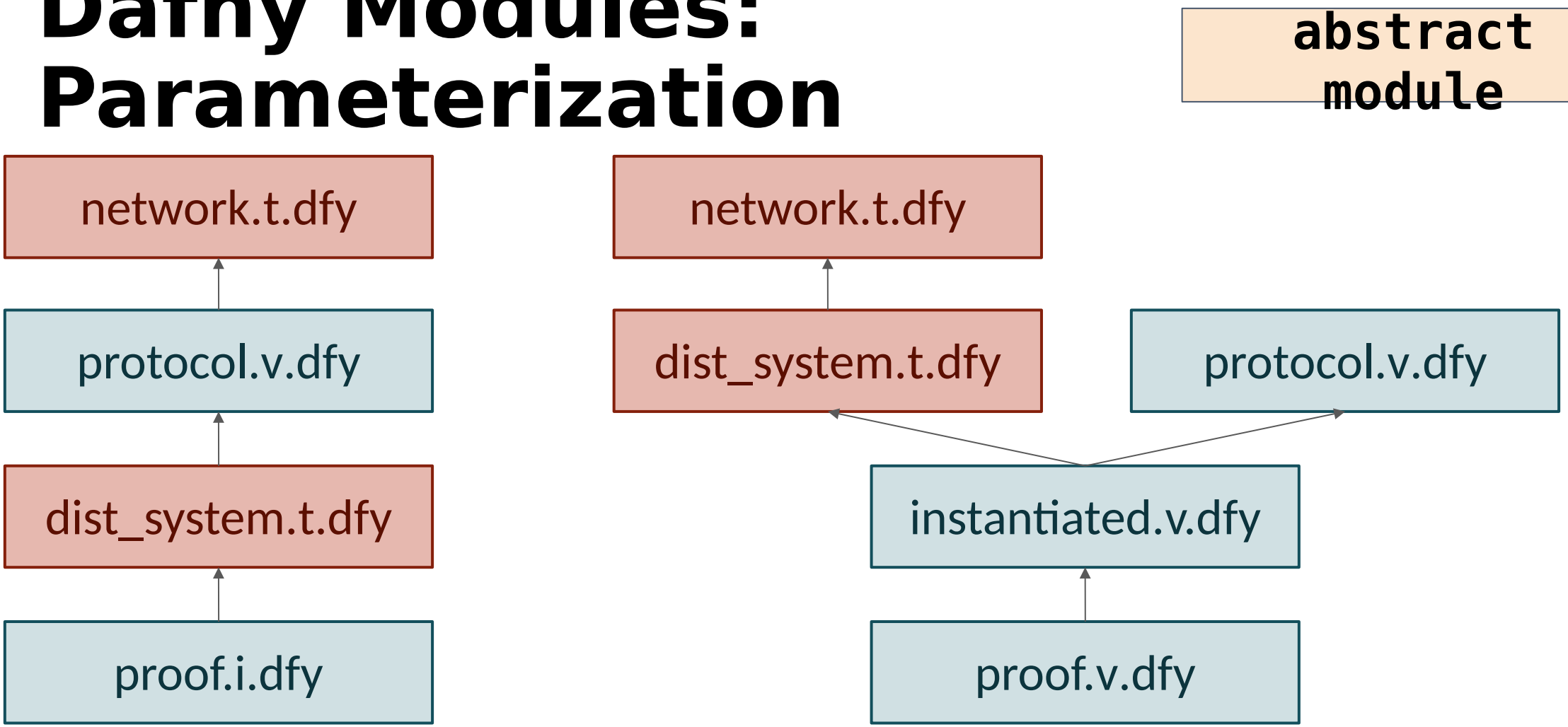
- Namespace scoping
- Parameterization
- Automation control

Dafny Modules: namespace scoping

import

```
module Network { … }
module Host { … }
module DistributedSystem {
  import Host;
  datatype Variables = Variables(host:Host.Variables,
network:Net.Variables)
  predicate Next(v:Variables, v’:Variables) {
    Host.Next(v.host, v’.host)
  }
}
```

Dafny Modules: Parameterization



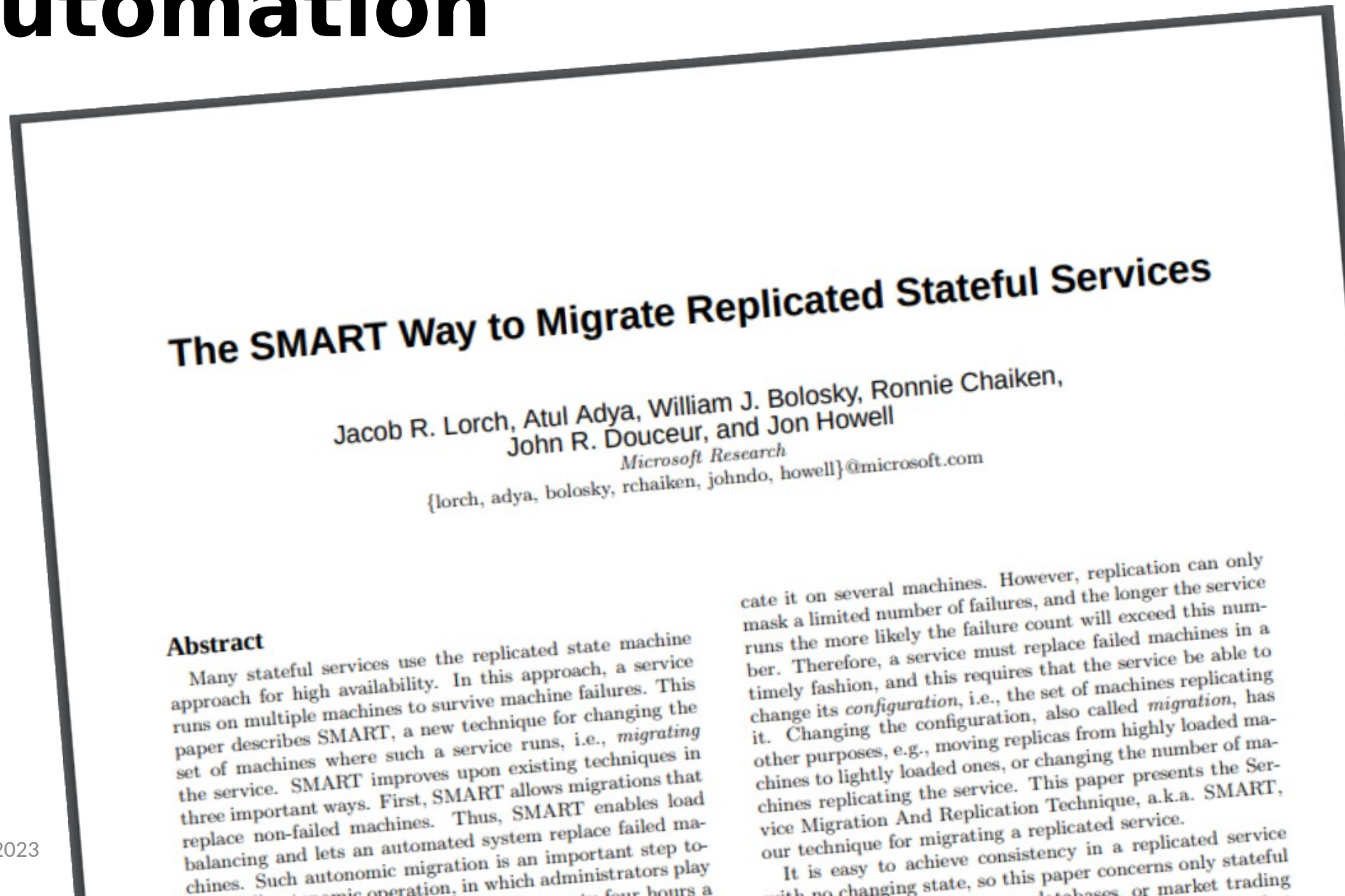
Dafny Modules: Automation Control

`export`

Control visibility into function & predicate bodies.

Transparent on the inside, opaque from the outside.

Automation



The SMART Way to Migrate Replicated Stateful Services

Jacob R. Lorch, Atul Adya, William J. Bolosky, Ronnie Chaiken,
John R. Douceur, and Jon Howell
Microsoft Research
{lorch, adya, bolosky, rchaiken, johndo, howell}@microsoft.com

Abstract

Many stateful services use the replicated state machine approach for high availability. In this approach, a service runs on multiple machines to survive machine failures. This paper describes SMART, a new technique for changing the set of machines where such a service runs, i.e., *migrating* the service. SMART improves upon existing techniques in three important ways. First, SMART allows migrations that replace non-failed machines. Thus, SMART enables load balancing and lets an automated system replace failed machines. Such autonomic migration is an important step to-

cate it on several machines. However, replication can only mask a limited number of failures, and the longer the service runs the more likely the failure count will exceed this number. Therefore, a service must replace failed machines in a timely fashion, and this requires that the service be able to change its *configuration*, i.e., the set of machines replicating it. Changing the configuration, also called *migration*, has other purposes, e.g., moving replicas from highly loaded machines to lightly loaded ones, or changing the number of machines replicating the service. This paper presents the Service Migration And Replication Technique, a.k.a. SMART, our technique for migrating a replicated service.

It is easy to achieve consistency in a replicated service

Automation

- The power of automation is to wipe out tedious tasks
- When it stops short, you have to add an "observe"
- When it goes too far, the verifier times out

Automation

```
function double(x:int) : int { 2*x }

  forall x :: double(x) == 2*x

assert double(4) == 8
```

```
double(4) != 8
```

```
double(4) == 2*4
```

```
8 != 2*4
```

```
false
```

Automation

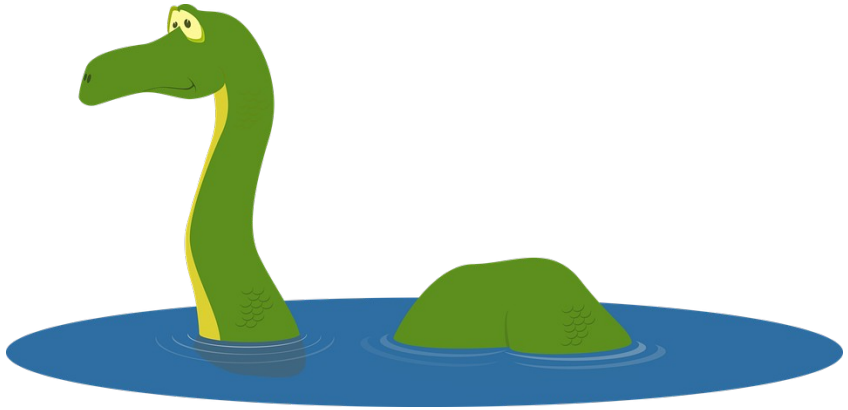
```
function dbl(x:nat) {
  if x==0 then 0 else dbl(x-1)+2
}

∀ x :: dbl(x) ==
   if x==0 then 0 else dbl(x-1)+2

assert dbl(4) == 8
```

```
dbl(4) != 8
if 4==0 then 0 else dbl(4-1)+2 != 8
if 4==0 then 0 else if 4-1==0 then 0 else dbl(4-1-1)+2+2 != 8
=0 then 0 else if 4-1==0 then 0 else if 4-1-1==0 then 0 else dbl(4-1-1-1)+2+2+2 != 8
```

The Timeout Monster rears its head only when you're trying to prove false things…



…which is basically all the time.

image: pixabay

Now imagine you have a tall tree of definitions

- data structures
- functions to manipulate them
- set and map comprehensions
- predicates to define transitions

…all of which are creating implicit forall quantifications!

The landscape of heuristic automation

- we want to open most definitions a few times, but
- we never want an infinitely-reachable space.

- we can afford a pretty big space, but
- we eventually have to curtail even the finite space

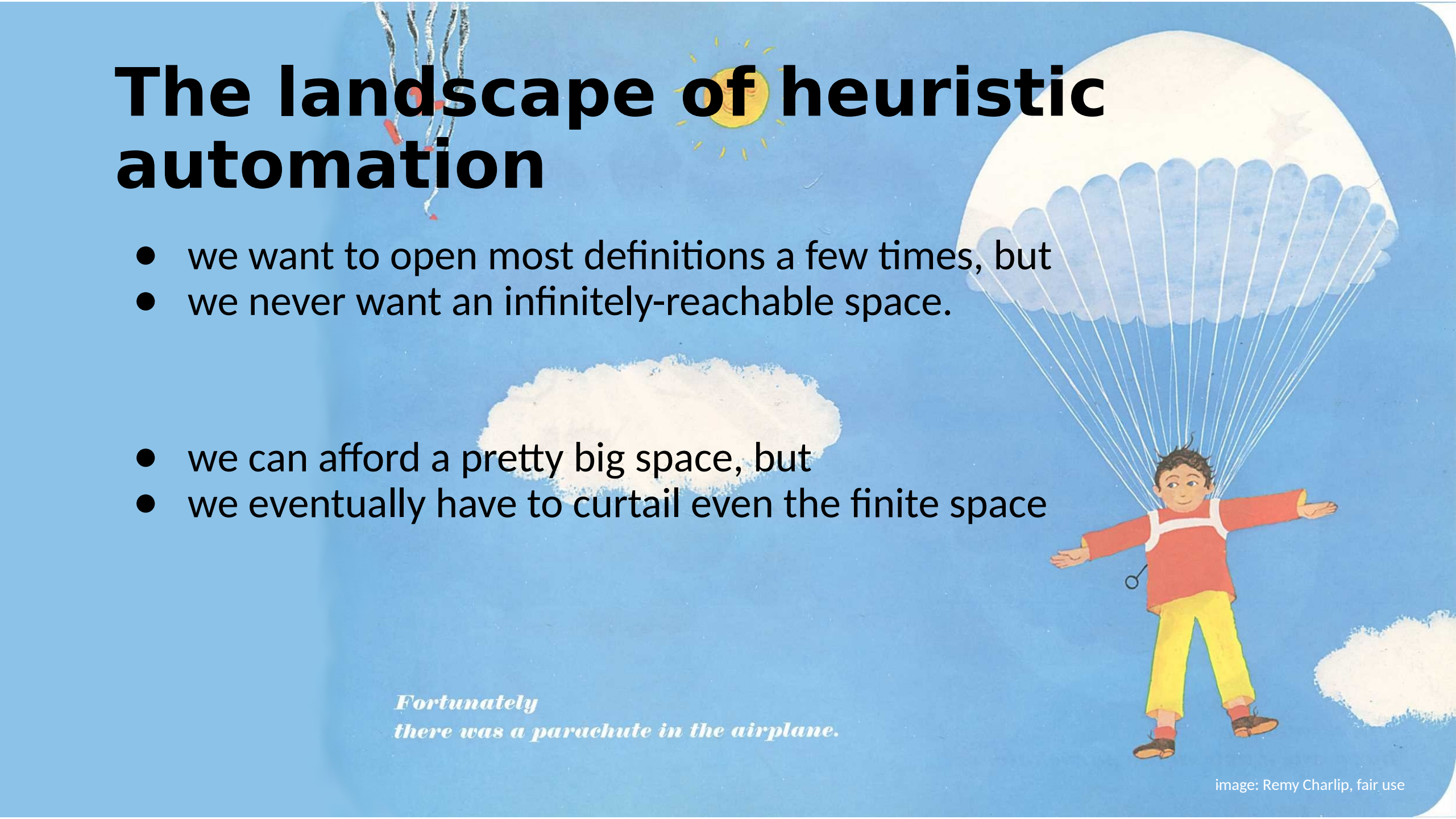


image: Remy Charlip, fair use

- We add automation to cover more ground
- We remove automation when it gets us in trouble

The result is a convoluted manifold… and we always want to be right on its edge!

What do we do when we slip off the edge?

Four-Step Timeout Cure

- Detection
- Diagnosis
- Correction
- Proof Repair

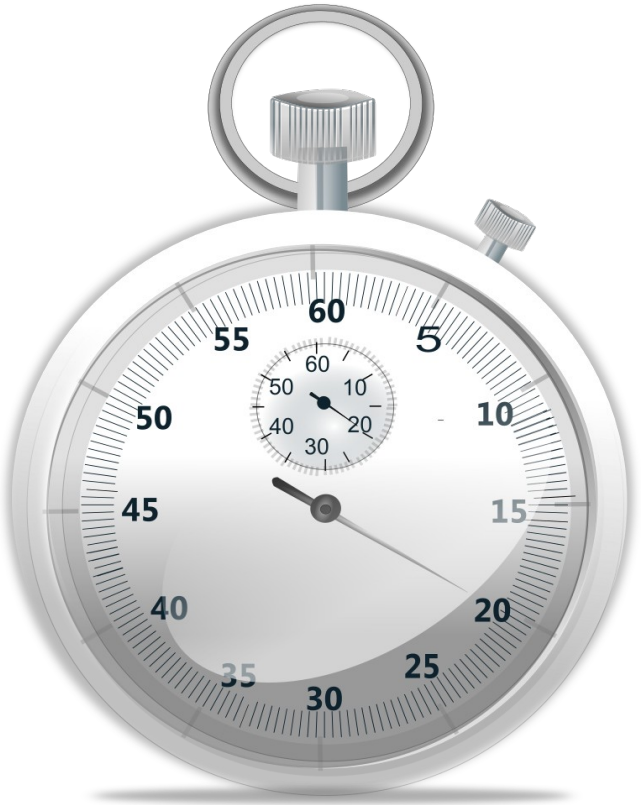
Detection

```
time dafny

dafny /timeLimit:20

dafny /trace

time dafny /proc:"*Pattern"
```



You need this wildcard because you're actually matching a mangled Boogie name.

image: freesvg.org

Diagnosis

```
dafny /timeLimit:10 /trace
chapter10/demos/exercise01_solution_with_timeouts
.dfy

dafny-profile.py 10 "*SendShardKeepsMapsFull"
chapter10/demos/exercise01_solution_with_timeouts
.dfy
```

```
Dafny program verifier finished with 0 verified, 0
errors, 1 time out
       max
     67000 exercise01solutionwithtimeoutsdfy.275:12
      3000 funType:MapType0Select
      2000 cast:U_2_bool
      1000 DafnyPreludebpl.141:18
      1000 exercise01solutionwithtimeoutsdfy.270:12
      1000 funType:$Unbox
```

Don't chase phantoms.

```
273   predicate MapsAreDisjoint(maps:MapGathering)
274   {
275     forall src1, src2 :: src1 in maps && src2 in maps && src1 != src2
276       ==> maps[src1].Keys !! maps[src2].Keys
277   }
```

Syntactic Triggers

print-triggers chapter10/demos/exercise01_with_timeouts.dfy > triggers

```
exercise01_with_timeouts.dfy(275,4): Info: Selected
triggers:

   {maps[src2], maps[src1]}, {maps[src2], src1 in maps},
{maps[src1], src2 in maps}, {src2 in maps, src1 in maps}
```

```
273   predicate MapsAreDisjoint(maps:MapGathering)
274   {
275     forall src1, src2 :: src1 in maps && src2 in maps && src1 != src2
276       ==> maps[src1].Keys !! maps[src2].Keys
277   }
```

Correction

Hiding Technique

```
∀ x :: layer(x) == layer(x+1) - 1
→ ∀ x :: layer(x) ==
LayerRelation(x)

function {:opaque} Foo()
```

hide an ∃ or map/set comprehension

controlled `module export`

Exposing Technique

```
assert LayerRelation(y);


reveal_Foo()
```

export the forall part
(+rarely mention or `reveal_`)

good export design

Proof Repair

Run a full verification to see which proofs failed

New failures are usually easy to fix: apply the exposing technique!

What about funType?

"nearby" instantiations may point at a map comprehension

use {:opaque} to see if you can replace the timeout with a fast failure (don't bother repairing the proof until you're certain you understand the cause)

Four-Step Timeout Cure

- Detection: fix timeouts right away!
- Diagnosis: quantifier-instantiation profiler
- Correction: hide definitions
- Proof Repair: fix right after timeout correction