Teaching Deductive Verification to Teenagers

Jean-Christophe Filliâtre
CNRS

IFIP WG 1.9/2.15
Leuven, Belgium
May 11–12, 2017

Université Paris Sud participates to a programme called
Les Apprentis Chercheurs (the research apprentices)

where teenagers meet researchers to get an initiation to science

started in 2004; involves several universities, grandes écoles, and research institutes; more than 1,000 apprentices so far

apprenticeship

the apprentices

- are volunteers
- meet researchers 3 hours a month, over one year
- observe, but also practice
- work in pair (one from middle school, one from high school)
- have to give a 7-minute presentation at the very end

my colleague Andrei Paskevich and I supervised four apprentices

- one from French 4$^{\text{ième}}$ grade (age 13, $\sim$ US 7/8th grade)
- one from French 2$^{\text{nde}}$ grade (age 15, $\sim$ US 9/10th grade)
- two from French 1$^{\text{ière}}$ grade (age 16, $\sim$ US 10/11th grade)

a challenge

our apprentices had a very light exposure to programming so far

- one with MIT's Scratch
- one with programming on a calculator only
- two with Python

plan

1. basic notions of programming first
   - with Python

2. then an introduction to deductive verification
   - with Python (and Why3 under the hood)

basic notions of programming

a pragmatic choice

we chose Python

- far from being a good programming language
- not that bad as a first language

in a browser, using `https://repl.it/`

subset of Python

we use only

- the while language
- integers and arrays
- `input`, `random`, and `print`

no functions, no libraries

first program: guess my number

a number is chosen randomly in 0..100 and guessed by the user

built interactively with the apprentices

introduces input/output, conditionals, and loops
(but also the idea of binary search)

note: we won't try to prove anything about this program

second program: Russian multiplication

```
r = 0
while q > 0:
    if q % 2 == 1:
        r = r + p
    p = p + p
    q = q // 2
```

- we explain it at the blackboard, on an example
  - the invariant shows up
- we test it, exhaustively for $\mathtt{p}, \mathtt{q} \in \{0..N\}$
  - but N cannot be too large

implement the 21 Nim game (*jeu des allumettes*), where the user plays against the machine

the program

- must check that the user is playing by the rules
- displays the outcome ("you win", "you lose")

- first, implement an opponent playing randomly
- then an opponent playing perfectly

other exercises

we took other exercises from Project Euler
`https://projecteuler.net/`

- the first problems are really easy
- fits nicely in our fragment (the answer is a number)
- entertaining

Project Euler problem 1

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Find the sum of all the multiples of 3 or 5 below 1000.

of course, they first implement a laborious, brute force solution

then we go to the blackboard and we figure out

$$\begin{array}{rr} & 3 \times (1 + 2 + \cdots + \lfloor \frac{999}{3} \rfloor) \\ + & 5 \times (1 + 2 + \cdots + \lfloor \frac{999}{5} \rfloor) \\ - & 15 \times (1 + 2 + \cdots + \lfloor \frac{999}{15} \rfloor) \end{array}$$

as well as

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

(without induction)

deductive verification

the main idea is sketched

(with simpler words), but that's not really important

at the end, we'll have a big button with a yes/no outcome

main objectives

- keep going with Python (no new language to learn)
- keep working within a browser (nothing to install)
- as few logical concepts as possible
  - avoid connectives and quantifiers in the first place

we reuse the Russian multiplication to make a first demo

the concept of loop invariant

p	q	r	
34	13	0	$34 \times 13 + 0$
68	6	34	$= 68 \times 6 + 34$
136	3	34	$= 136 \times 3 + 34$
272	1	170	$= 272 \times 1 + 34$
544	0	442	$= 544 \times 0 + 442$

```python
r = 0
while q > 0:
    #@ invariant 0 <= q
    #@ invariant r + p * q == a * b
    print(p, q, r)
    if q % 2 == 1:
        r = r + p
    p = p + p
    q = q // 2
print(p, q, r)
print("a * b =", r)
#@ assert r == a * b
```

prove the identity

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

with a program (their first lemma function!)

```
n = int(input("enter␣n:␣"))
#@ assume n >= 0

s = 0
k = 0
while k <= n:
    #@ invariant k <= n+1
    #@ invariant s == (k-1) * k // 2
    s = s + k
    k = k + 1

print(s)
#@ assert s == n * (n+1) // 2
```

verify the following program

```
n = int(input("enter␣n:␣"))
#@ assume n >= 0

r = 0
s = 1
while s <= n:
    r = r + 1
    s = s + 2 * r + 1

print(r)
#@ assert r*r <= n < (r+1)*(r+1)
```

a more complex exercise: binary search

we first explain the problem and let them devise a solution

then they have to

1. implement it
2. test it on small, manually-written arrays
3. generate random, sorted arrays to make larger tests
4. prove safety
5. prove soundness
6. prove completeness
7. prove termination

note: no arithmetic overflow issue here,
as Python uses arbitrary-precision integers

we start by verifying that

```
a = [0] * n
a[0] = randint(0, 100)
for i in range(1, n):
    a[i] = a[i-1] + randint(0, 10)
```

ends up with a sorted array

we have to introduce quantifiers and implication, so that we can write annotations such as

```
#@ assert forall i, j. 0 <=i<=j<len(a) -> a[i]<=a[j]
```

(we briefly mention why this is better than

```
#@ assert forall i. 0 <=i<len(a)-1 -> a[i]<=a[i+1]
```

but we try to avoid a technical discussion)

other exercises

we prepared two other exercises:

- insertion sort (invariants are more involved)

- Nim game opponent wins whenever possible
  (requires axiomatization of win/lose predicates)

but they were not used at the end (lack of time)

under the hood

translating Python to WhyML

Why3's programming language $\sim$ a small subset of OCaml

we translate Python (and the annotations) to this language

some caveats

- Python is untyped
- Python variables are mutable (including loop indices)
- Python has constructs such as `break` or `return`

```
# first time we assign id
id = e
...

# and later
id = e
```

```
(* we introduce id *)
let id = ref e in
...

id := e;
```

within annotations, we dereference all variables

e.g. the loop invariant

```
#@ invariant r + p * q == a * b
```

gets translated to

```
invariant { let a = !a in let b = !b in
            let p = !p in let q = !q in
            let r = !r in r + p * q = a * b }
```

we account for arguments being passed by value, yet received in mutable variables

```
def f(x1, ..., xn):
  body
```

```
let f x1 ... xn =
  let x1 = ref x1 in
  ...
  let xn = ref xn in
  ...
```

```
for id in e:
  #@ invariant inv
  body
```

```
let l = e in
for i = 0 to len(l) - 1 do
  invariant { let id = l[i] in inv }
  let id = ref l[i] in
  body
done
```

```
for id in range(e1, e2):
  #@ invariant inv
  body
```

```
for id = e1 to e2 - 1 do
  invariant { inv }
  let id = ref id in
  body
done
```

break and return

break and return are translated using exceptions

```
while test:
  body
```

```
try
  while ... do
   ...
  done
with Break →
  ()
end
```

`break` and `return` are translated using exceptions

```
def f(x1, ..., xn):
  body
```

```
let f x1 ... xn =
  try
    ...
  with Return v →
    v
  end
```

we let Why3 inferring types
(arbitrary-precision integers, arrays, etc.)

our translator fails on a program that is ill-typed, e.g.

```
def f(x):
  if x == 0:
    return 1
```

(so we turn some run-time errors into compile-time errors)

Python's lists are actually resizable arrays

we make a simplification, using mutable arrays only

it would be easy to model Python's lists instead,
at the cost of extra annotations regarding lengths being unchanged

support library

a small Why3 library provides definitions for things such as

- `int(input(s))`, `randint(l, u)`
- `len(a)`, `range(l, u)`
- `//` and `%`
  caveat: this is neither Euclidean division, nor computer division (but defined in Python's manual)

Why3 in your browser — `why3.lri.fr/try`

we are using

- `js_of_ocaml` to compile both Why3 and Alt-Ergo to JavaScript
- Ace (Ajax.org Cloud9 Editor)
- Font Awesome
- a few lines of CSS and HTML (600 loc)

even possible to build an offline version

much simpler than running a server

going further?

to support a larger fragment of Python,
it is likely that we should do first a Python-specific static typing,
then translate to Why3

missing features

- tuples, parallel assignments, etc.
- objects
- dynamic scope?

questions ?