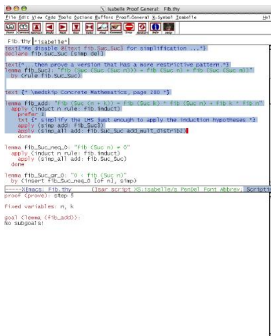


Name: Christian Urban

- I am using theorem provers:



```
Isabelle Proof General: Fib.thy
File Edit View Options Settings Database Buffer Proof General System Isabelle
[Icons]

Fib.thy [1:1] [1:1]
syntax declare syntax and syntax For simplification...
include Fib_Suc_Suc (Fib.thy)

text {* This proves a version that has a nice inductive pattern. *}
lemma Fib_Suc_0: "(0 <= Suc n) <=> Fib (Suc n) = Fib (Suc n) + Fib (Suc n - 1)"
by (induct n) (simp)

text {* See also Concrete Mathematics, page 240-9. *}
lemma Fib_induct: "(0 <= Suc n) <=> Fib (Suc n) = Fib (Suc n) + Fib (Suc n - 1)"
apply (induct n rule: fib_induct)
apply (simp)
text {* Identify the SMC rule enough to apply the induction hypothesis. *}
apply (simp add: fib_Suc)
apply (simp add: Fib_Suc_Suc)
done

lemma Fib_Suc_eq_0: "(Fib (Suc n) = 0)"
apply (induct n rule: fib_induct)
apply (simp add: Fib_Suc_Suc)
done

lemma Fib_Suc_eq_1: "(Fib (Suc n) = 1)"
by (simp add: Fib_Suc_eq_0)

end

syntax Fib.thy
(* Use script 00 IsabelleTools/proof from @libdir/IsabelleTools *)
proof (prove) - step 5
Fixed variables: n, k
goal (prove) (Fib_Suc_0):
no subgoals

```

- My goal is to reduce the number of bugs in programs.

Imagine You Are a Perfect Programmer



What can make your program still **not** behave as you intended?

Imagine You Are a Perfect Programmer

An orange circle containing the text "cosmic rays ;o)".

cosmic
rays ;o)

A large blue circle containing the text "perfect program".

perfect
program

What can make your program still **not** behave as you intended?

Why Bothering with Compilers?

- Ken Thompson hid a Trojan horse in a compiler **without** leaving any traces in the source code.



Ken Thompson
Turing Award, 1983

Why Bothering with Compilers?

- Ken Thompson hid a Trojan horse in a compiler **without** leaving any traces in the source code.



Ken Thompson
Turing Award, 1983

- Assume you ship binary and sources of a compiler.
 - 1) Make the compiler aware when it compiles itself.
 - 2) Add the Trojan horse.
 - 3) Compile.
 - 4) Delete Trojan horse from sources.
 - 5) Go on holiday for the rest of your life. ;o)

Why Bothering with PLs?

"I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language. My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years..." Tony Hoare recently in a talk



Tony Hoare
Turing Award, 1980
(Quicksort)

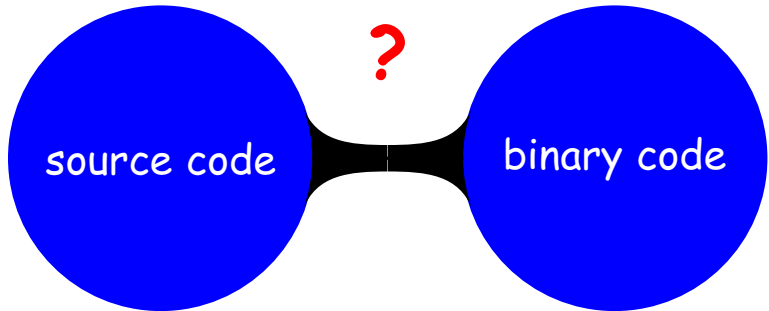
Why Bothering with PLs?

Q: Why bother doing proofs about programming languages? They are almost always boring if the definitions are right.

A: The definitions are almost always wrong.

"Anonymous" cited in B. Pierce's book on
Types and Programming Languages

What Do We Have to Do?



What We Have to Do?

- specify precisely which programs we can write (syntax)
- specify precisely what a program means (semantics)
- specify precisely how the compiler translates a program to machine code
- specify precisely what machine code is and how it is executed
- finally check (**prove**) that the result of the machine code run is what we expect

What We Have to Do?

- specify precisely which programs we can write (syntax)
- specify precisely what a program means (semantics)
- specify precisely how the compiler translates a program to machine code
- specify precisely what machine code is and how it is executed
- finally check (**prove**) that the result of the machine code run is what we expect
- **everything in 2h!**

Simplifying Assumptions

- our language will access the **infinitely** big memory
- every **memory location** contains an arbitrary big natural number
- therefore a memory snapshot (a **state**) is a function from locations to natural numbers

types

state = "loc \Rightarrow nat"

- for example: $s\ 42 = 666$

Simplifying Assumptions

- our language will access the **infinitely** big memory
- every **memory location** contains an arbitrary big natural number
- therefore a memory snapshot (a **state**) is a function from locations to natural numbers

types

state = "loc \Rightarrow nat"

- for example: $s\ 42 = 666$, $s'\ 42 = 0$

Our Language

- Each program is a sequence of commands:

datatype cmd =

```
SKIP  
| ASSIGN loc aexp      ("_ ::= _" 60)  
| SEQ  cmd cmd        ("_;_" [60, 60] 10)  
| COND bexp cmd cmd  ("IF _ THEN _ ELSE _" 60)  
| WHILE bexp cmd     ("WHILE _ DO _" 60)
```

where aexp and bexp are arithmetic and boolean expressions (in a moment).

for example

```
WHILE true DO (42 ::= 1; SKIP)
```

Arithmetic Expressions

- Arithmetic expressions:

datatype

aexp = N nat

| Op1 "nat \Rightarrow nat" aexp

| Op2 "nat \Rightarrow nat \Rightarrow nat" aexp aexp

Arithmetic Expressions

- Arithmetic expressions:

datatype

aexp = N nat

| Op1 "nat \Rightarrow nat" aexp

| Op2 "nat \Rightarrow nat \Rightarrow nat" aexp aexp

- For example:

N 2, Op1 Suc (N 3), Op2 Plus (N 5) (N 6)

Arithmetic Expressions

- Arithmetic expressions:

datatype

aexp = N nat

| Op1 "nat \Rightarrow nat" aexp

| Op2 "nat \Rightarrow nat \Rightarrow nat" aexp aexp

- For example:

N 2, Op1 Suc (N 3), Op2 Plus (N 5) (N 6)

- What is the meaning of an arithmetic expressions?

Meaning of an Arithmetic Expression

datatype

aexp = N nat

| Op1 "nat \Rightarrow nat" aexp

| Op2 "nat \Rightarrow nat \Rightarrow nat" aexp aexp

Meaning of an Arithmetic Expression

datatype

aexp = N nat

| Op1 "nat \Rightarrow nat" aexp

| Op2 "nat \Rightarrow nat \Rightarrow nat" aexp aexp

$$\overline{N\ n \longrightarrow a\ n}$$
$$\frac{e \longrightarrow a\ n}{Op1\ f\ e \longrightarrow a\ f\ n}$$
$$\frac{e_0 \longrightarrow a\ n_0 \quad e_1 \longrightarrow a\ n_1}{Op2\ f\ e_0\ e_1 \longrightarrow a\ f\ n_0\ n_1}$$

Memory Access

- Arithmetic expressions:

datatype

aexp = N nat

| X loc

| Op1 "nat \Rightarrow nat" aexp

| Op2 "nat \Rightarrow nat \Rightarrow nat" aexp aexp

$\frac{}{(N\ n,s) \longrightarrow a\ n}$

$\frac{}{(X\ i,s) \longrightarrow a\ s\ i}$

$\frac{}{(e,s) \longrightarrow a\ n}$

$\frac{}{(Op1\ f\ e,s) \longrightarrow a\ f\ n}$

$(e_0,s) \longrightarrow a\ n_0$

$(e_1,s) \longrightarrow a\ n_1$

$\frac{}{(Op2\ f\ e_0\ e_1,s) \longrightarrow a\ f\ n_0\ n_1}$

How Far We Got

- specify precisely which programs we can write (syntax)
- specify precisely what a program means (semantics)
- specify precisely how the compiler translates a program to machine code
- specify precisely what machine code is and how it is executed
- finally check (prove) that the result of the machine code run is what we expect