# Teaching *Semantics* with a Proof Assistant
# or
# No more “LSD trip proofs”

Tobias Nipkow

Fakultät für Informatik
TU München

## The problem: students and proofs

*NP-completeness reductions done in the wrong direction*

*Arguments that start out by assuming what has to be proved*

*Proofs that look more like LSD trips than coherent chains of logic*

Scott Aaronson (MIT)

## Majority of informatics students

Proofs

## Disclaimer

Not (entirely) the students' fault

- *Writing* precise proofs is not demanded outside theory/formal methods courses.
- Even there, it is often incorrectly assumed, or not demanded for fear of the cost incurred.

## A glimmer of hope

*Proof Assistants*

## What is a proof assistant (PA)

An interactive tool
for constructing
mathematical definitions and proofs

The ideal:

- the user gives the proof outline
- the system fills in the routine steps

## The proof assistant universe

Agda

HOL

ACL2

Coq

Isabelle

Mizar

PVS

HOL Light

## The hope

Proof Assistant = Video Game

Proof War
QED or DIE !

Tobias Nipkow. *Winskel is (almost) Right:*
*Towards a Mechanized Semantics Textbook*.
In: Proceedings FSTTCS 1996.

Formalized the first 100 pages of Winskel's Semantics textbook in Isabelle.

Used in my teaching since then.

Without forcing the students to write formal proofs.

But writing proofs requires *PRACTICE*.

## An experiment (WS 2010/11)

Practice via use of PA in *Semantics* course

PA gives *immediate feedback* and is

- untiring
- unerring
- pedantic
- impartial

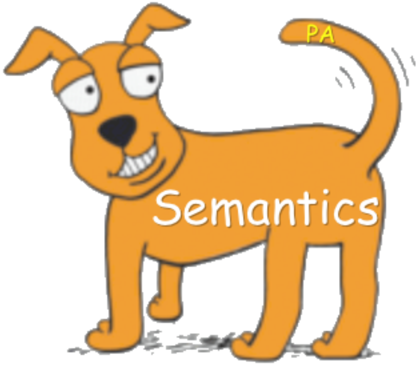
## Programming Language Semantics course at TUM

- $\approx 15$ MSc students
- theory section of curriculum, 8 ECTS
- 2 $\times$ 90 minutes lectures / week
- 90 minutes exercise class / week
- 14 weeks

## Homework — the key!

- In the past: homework did not count.
- Predictable result: some do it, some don't.
- Now: homework 40% of final grade.
- Homework based entirely on proof assistant.

We want the Semantics dog to wag its PA tail:



Not the other way around!

**Semantics** with a proof assistant

Not

**Theorem Proving** with semantics examples

Teach structured (readable) proofs,
not proof scripts

```
proof(induct n)
  assume formula
  have formula by ...
  ⋮
  have formula by ...
  show formula by ...
qed
```

~~apply(...)~~

~~⋮~~

~~apply(...)~~

~~done~~

nontransferable skill

## However . . .

- Proof scripts are easier to learn and to hack.
  At least for small proofs.
- Also useful for “proof debugging”

$\Longrightarrow$ We start with simple proof scripts
and upgrade to structured proofs soon after.

## Proof versus logic

Do not teach logic,

teach how to write proofs.

???

- Single step natural deduction proofs belong in logic courses.
- Application-oriented courses should reason modulo logic.
- If you believe that $A$ and $B$ imply $C$, write

  `from` $A$ `and` $B$ `have` $C$ `by auto`

  and let the machine perform the proof.
- If it cannot, refine proof.

Not a new idea: Mizar

## In a nutshell

Do not let logic dominate your thinking.

Not a new idea: Mathematics

Needs good automation to work well.

Issue: when automation fails,
proof scripts simplify debugging

## Operational semantics

of a simple imperative language.

- Focus on *one* language
- Present spectrum of concepts and applications

Student comment:

> *I thought theoreticians do not like imperative languages and prefer the $\lambda$-calculus?*

Initially:

Mainly live demos
of Isabelle specifications and proofs

Once the students are familiar with Isabelle
(after 1/3 of the course):

More slides and blackboard

I believe in blackboard and slides
for presenting concepts and proofs.

## The benefits of structured Isabelle proofs

- Close to standard proofs
- Ease the move from Isabelle to blackboard
- Provide language for blackboard proofs

## Isabelle: Functional programming

- Natural numbers and lists
- Recursive datatypes and functions
- Proof by induction

Typical proof:

```
apply(induct ...)
apply auto
done
```

Challenges:

- syntax, syntax, syntax
- finding auxiliary lemmas
- getting definitions right

User experience:

frustration but fascination

## Expressions

A first (motivating!) glimpse of semantics:

- Arithmetic and boolean expressions
- State
- Evaluation functions
- Expression optimization
- Stack machine
- Compilation to stack machine

Proofs still `induct-auto`

# Isabelle: Logic and proofs

- Logic: hardly more than syntax of formulas
- Proofs
  - Automation
  - Structured proofs

Introductory example:

lemma Cantor: $\neg surj(f :: \alpha \to (\alpha)set)$
proof
  assume $surj(f)$
  hence $\exists a.\ f(a) = \{x \mid x \notin f(x)\}$
    by(auto simp: surj-def)
  thus $False$ by blast
qed

Not typical for later proofs

## Automation

The students' best friend: SLEDGEHAMMER

Employs external automatic provers to find proofs.

Student comment:

*Isabelle's automation makes me lazy.*

I approve of this!

Isabelle does not work magic.
It merely automates the obvious. Mostly.

## Isabelle: Inductively defined predicates

- The idea: simple enough
- Rule induction: a new and nontrivial concept

Main problem: when to induct on what

After 4 weeks ($\approx 1/4$ semester),
the logical foundations are in place.

Now Semantics takes over.

## IMP

A simple imperative language:

$$\begin{array}{lcl} com & ::= & \texttt{SKIP} \\ & | & nat\ \texttt{:=}\ aexp \\ & | & com\ \texttt{;}\ com \\ & | & \texttt{IF}\ bexp\ \texttt{THEN}\ com\ \texttt{ELSE}\ com \\ & | & \texttt{WHILE}\ bexp\ \texttt{DO}\ com \end{array}$$

The rest of the semester focuses on IMP.

- Big and small-step semantics
- Stack machine and compiler
- Type system
- Static analyses: definite assignment, liveness
- Information-flow security type systems
- Hoare logic
- Verification condition generation
- Extensions of IMP

Semantic correctness of each concept is proved
Almost everything is executable

## Sample semantics

```
(SKIP,s) ⇒ s |

(x := a,s) ⇒ s(x := aval a s) |

(c1,s1) ⇒ s ⟹ (c2,s2) ⇒ s3
⟹ (c1;c2, s1) ⇒ s3 |

bval b s ⟹ (c1,s) ⇒ t
⟹ (IF b THEN c1 ELSE c2, s) ⇒ t |

¬ bval b s ⟹ (c2,s) ⇒ t
⟹ (IF b THEN c1 ELSE c2, s) ⇒ t |

¬ bval b s ⟹ (WHILE b DO c,s) ⇒ s |

bval b s1 ⟹ (c,s1) ⇒ s2
⟹ (WHILE b DO c, s2) ⇒ s3
⟹ (WHILE b DO c, s1) ⇒ s3
```

# Sample proof

```
lemma hoare-sound:  $\vdash \{P\}c\{Q\} \implies \models \{P\}c\{Q\}$
proof(induct rule: hoare.induct)
  case (While P b c)
  { fix s t
    have (WHILE b DO c,s) $\Rightarrow$ t
       $\implies$ P s $\implies$ P t $\wedge$ $\neg$ bval b t
    proof(induct rule: big-step-induct)
      case WhileFalse thus ?case by blast
    next
      case WhileTrue thus ?case using While(2)
        unfolding hoare-valid-def by blast
    qed
  }
  thus ?case unfolding hoare-valid-def by blast
qed (auto simp: hoare-valid-def)
```

## Sample homework (2 weeks)

Define a *dependency analysis* between variables.
We say that $x$ depends on $y$ after command $c$
if the value of $y$ before the execution of $c$
may influence the value of $x$ after the execution.

Prove its soundness w.r.t. to the big-step semantics.

## Claim

It is challenging.

It is motivating.

It is exciting.

*It works!*

## Evidence

- Practically *everybody* hands in homework — unheard of in the past
- Homework grades: 88% of points (on avg)
- Only one attempt at cheating detected
- Student evaluation of contents of course: improved from 2.1 to 1.6 (on avg) [scale: 1–5, avg $= 2.3$]
- Avg grade in final (oral!) exam: 1.6

## Anonymous student feedback

*Learning to use a theorem prover is exciting.*

*Thanks for offering this great course!*

*In general really good but very demanding course.*

*Homework is too time consuming, with all the syntax problems etc. [≈ 8 hours/week]*

*I will certainly recommend this course to other students.*

*It is really difficult to find something bad about this course :-)*

## Why the proof part works

- Small repertoire of proof principles: induction, simplification, case distinction, logic
- Standard proof pattern: induction, in each case combining assumptions to reach conclusion, maybe with a case distinction
- Proof automation
- Background theories: only natural numbers and lists
- Very focused material: IMP
- *We are excited about the new course*

Mission accomplished

## Help yourself!

- 500 $\LaTeX$-beamer slides
- Isabelle theories
- Exercises and homework

`www.in.tum.de/~nipkow/semantics`

## Related courses

- Benjamin Pierce, Software Foundations, U of Pennsylvania, Coq
- Christian Urban, Semantics, TUM, Isabelle
- Rex Page, Software Engineering, U of Oklahoma, ACL2
- Matthias Felleisen, Logic, Northeastern U, ACL2
- ...?

## Other areas ripe for the PA treatment (?)

- Any area concerned mostly with syntactic structures:
  $\rightsquigarrow$ Programming Languages and Logic
- Avoid subjects where proof steps are trivial for the student but tedious on the machine — demotivating.

We need more experiments!