An Educational Proof Assistant for Language Theory

Jonathan Aldrich **Robert J. Simmons** **Key Shin**
Carnegie Mellon University Microsoft

The Challenge of Teaching Proofs

- Concepts like induction are inherently complex
- Easy to get details wrong, without realizing it
- Feedback in a typical course may take a week due to grading time

*Wouldn't it be nice if there were an **easy-to-use** tool that provides students with **immediate feedback** on mistakes in their proofs?*

Introducing SASyLF: a Second-order Abstract Syntax Logical Framework ("Sassy Elf")

SASyLF Design Goals

- ***Gentle learning curve.*** Professors don't have time to spend teaching tools!
- ***Familiar syntax.*** The surface syntax of SASyLF mirrors paper proofs.
- ***Explicit notation.*** Making clear what is going on aids the learning process.
- ***Minimal math context.*** SASyLF is based exclusively on rule induction—set theory, etc. is omitted.
- ***Support for variable binding.*** Encoding variables is difficult and distracting.
- ***Incremental proof development.*** Can write "unproved" for any statement and the assistant will assume it is true (but will yield a warning).
- ***Local checking and localized errors.*** Understanding what went wrong is critical to learning, so all checks are local and errors point to the specific low-level step that failed.

SASyLF Definitions

$$\frac{}{e \mapsto^* e}$$

```
--------- singlestep
e ->* e
```

$$\frac{e_1 \mapsto e_2 \qquad e_2 \mapsto^* e_3}{e_1 \mapsto^* e_3}$$

```
e1 -> e2
e2 ->* e3
--------- multistep
e1 ->* e3
```

SASyLF Proofs

Case $\dfrac{e_1 \mapsto e_2 \qquad e_2 \mapsto^* e_3}{e_1 \mapsto^* e_3}$ We have $e_2 : \tau$ by the progress lemma on the assumption that $e_1 : \tau$ and the first premise, and so by the induction hypothesis on the second premise e_3 is not stuck.

```
case rule
     d1 : e1 -> e2
     d2 : e2 ->* e3
     --------------- multistep
     d3 : e1 ->* e3
is
 d4 : e2 : t by theorem progress on dt, d1
 d5 : e4 notstuck
          by induction hypothesis on d2, d4
end case
```

```
e ::= x
  |   e e
  |   fn x : tau => e[x]
  |   "(" ")"

tau ::= unit
    |   tau -> tau

Gamma ::= *
      |   Gamma, x : tau
```

We define the syntax of the simply-typed λ-calculus using BNF.

The notation `e[x]` means that x is free in e. The `x` in "`fn x …`" is the binding occurrence. `x` is mentioned in the grammar for `e`, so the tool knows `x` is a variable representing an `e`.

Parentheses are special to SASyLF, so we quote them when they appear in the target language.

```
judgment value: e value

------------ val-unit
"(" ")" value

-------------------------- val-fn
fn x1 : tau => e1[x1] value
```

Declaring a judgment form.

Inference rules are defined with one premise per line above the bar, and the conclusion below it.

```
judgment step: e -> e

e1 -> e1'
-------------- c-app-l
e1 e2 -> e1' e2

e1 value
e2 -> e2'
-------------- c-app-r
(e1 e2) -> (e1 e2')

e2 value
------------------------------ r-app
(fn x : tau => e[x]) e2 -> e[e2]
```

In a rule, primes and numerical suffixes are used to distinguish different instances of the same syntactic class.

Often grammars are ambiguous, so parentheses can be used to disambiguate expressions. SASyLF uses a GLR parser to support ambiguous grammars, as long as individual expressions are not ambiguous.

`e[e2]` means substitute `e2` for `x` in `e` (binding and substitution are built in).

```
judgment has-type: Gamma |- e : tau
assumes Gamma

--------------------- t-unit
Gamma |- "(")" : unit

---------------------- t-var
Gamma, x:tau |- x : tau

Gamma, x1:tau |- e[x1] : tau'
----------------------------------- t-fn
Gamma |- fn x : tau => e[x] : tau -> tau'

Gamma |- e1 : tau' -> tau
Gamma |- e2 : tau'
----------------------- t-app
Gamma |- e1 e2 : tau
```

Hypothetical judgments are built in; here we state that `Gamma` is the context holding assumptions.

`t-var` shows how to use an assumption in `Gamma`, and also defines the meaning of that assumption.

Variable names aren't significant; we can use `x1` for `x` as long as we're consistent within the premise.

```
theorem preservation: forall dt: * |- e : tau
                      forall ds: e -> e'
                      exists     * |- e' : tau.

dt' : * |- e' : tau                  by induction on ds :
  case rule
    d1 : e1 -> e1'
    ------------------- c-app-l
    d2 : e1 e2 -> e1' e2
  is
    dt' : * |- e' : tau              by case analysis on dt :
      case rule
        d3 : * |- e1 : tau' -> tau
        d4 : * |- e2 : tau'
        ------------------------ t-app
        d5 : * |- (e1 e2) : tau
      is
        d6 : * |- e1' : tau' -> tau by induction hypothesis on d3, d1
        dt' : * |- e1' e2 : tau     by rule t-app on d6, d4
      end case
    end case analysis
  end case

  case rule... // case for rule c-app-r is similar

  case rule
    d1 : e2 value
    ------------------------------------- r-app
    d2 : (fn x : tau' => e1[x]) e2 -> e1[e2]
  is
    dt' : * |- e' : tau              by case analysis on dt :
      case rule
        d4 : * |- fn x : tau' => e1[x] : tau'' -> tau
        d5 : * |- e2 : tau''
        ------------------------------------------- t-app
        d6 : * |- (fn x : tau' => e1[x]) e2 : tau
      is
        dt' : * |- e' : tau          by case analysis on d4 :
          case rule
            d7: *, x:tau' |- e1[x] : tau
            ---------------------------------------- t-fn
            d8: * |- fn x : tau' => e1[x] : tau' -> tau
          is
            d9: * |- e1[e2] : tau    by substitution on d7, d5
          end case
        end case analysis
      end case
    end case analysis
  end case
end induction
end theorem
```

Like Twelf, SASyLF supports proving so-called ∀∃-statements of the form "for all judgments $J_1..J_n$ there exists a judgment J_{out}."

When we case-analyze, we give each possible rule with the conclusion instantiated to the thing we are case analyzing. We can introduce variables (e.g. `e1`, `e2`, and `tau'`) that can be used later.

Proofs are a sequence of statements of the form:
<name> : <judgment> <justification>
The name is used to refer to the fact later. The judgment is the fact, while the justification is an argument for while the fact holds. Here the fact holds due to induction.

We can use the induction hypotheses or apply a rule to get the results we want. In either case, we need to state the judgment names to which the rule applies. The last judgment in the derivation must prove the theorem (for this case).

Because SASyLF builds in hypothetical judgments, `x:tau'` in `Gamma` for judgment `d7` really means that the judgment holds exactly when `x` can be replaced with a judgment showing that some `e2` has type `tau'`. But we have judgment `d5` that tells us exactly that (the `t-fn` case analysis is enough to show that `tau''=tau'`), so we can get `d9` by substituting `d5` into `d7`!
As in Twelf, no substitution lemma is required, although we could prove one for pedagogical purposes if we want.

Teaching with SASyLF

We used SASyLF for an assignment on reasoning about the dynamic semantics of programs in a Spring 2008 graduate course at CMU. Our findings in a controlled experiment were generally promising:

- 11 of 13 students who used the SASyLF tool found that it helped them find errors in their proofs
- 12 of 13 said the tool increased their confidence that their proofs were correct.
- In contrast, 14 of 16 members of the control (no tool) group wished they had earlier feedback on their mistakes.
- Some students dropped out due to usability issues with the tool, many of which we have since addressed. Still, 7 of 11 surveyed students who completed the study with the tool would use the tool again, and another three would use it if the usability issues were addressed.

Comparing the tool to handwritten proofs, one student said, **"I actually did the entire assignment on paper first and then moved over to using the tool. I found the paper approach really easy. But once I started using the tool I started understanding the concepts better."**

www.sasylf.org

- Open source release of SASyLF
- Paper(s) describing the tool and our teaching experience
- Documentation and examples
- A solution to part 2A of the POPLmark challenge for mechanized metatheory

Related Work

Proof Assistants used for PL Metatheory

- Twelf
- Coq
- Isabelle/HOL
- Abella

Educational Tools for Logic

- Tarski's World
- Tutch

Educational Tools for PL Theory

- DrScheme