

# Project proposal for prettyprinting Twelf signatures

Troels Henriksen (athas@sigkill.dk)

20th October 2009

## Motivation

Twelf<sup>1</sup> is a language and a program for expressing and property-checking deductive systems. The language is reminiscent of traditional logic programming languages, suitable for editing in a plain text editor and easy to parse for a computer (see Figure 1.1 on page 3, ignore the `%ppr` annotations for now). When preparing papers whose proofs have been checked by Twelf, we do not wish to include the Twelf-code as-is, but rather translate it to traditional human-readable logic notation (such as inference rules) first. Performing this translation by hand is tedious and error-prone, and it is therefore desirable to write a program that can do it for us. See Figure 1.2 on page 3 for an example of how the output of such a program could look.

## Goal

The ambition of this project is to construct a program that can *prettyprint* Twelf signatures in some printing language (for example  $\text{\LaTeX}$  or MathML). The details on how to perform the prettyprinting are provided by *annotating* the Twelf program. For example, in figures 1.1 and 1.2, the `eval` relation is prettyprinted as an arrow separating the two terms. Note that it is outside the scope of the project to interact with Twelf's actual query and correctness engine.

## Methods

- I will analyse how the program will be used in practice. No formal analysis method is mandated (or expected), but design choices and restrictions must be based on objective analysis of potential use cases.
- A high-level design of how to control the prettyprinting of Twelf must be constructed. This is relatively straightforward for many signatures, but those involving higher-order abstract syntax (HOAS) require nontrivial design considerations. Control will likely be provided by annotations or similar rules (the form of which will have to be developed), but there may be alternatives to storing the annotations in the Twelf source code.
- The design will be implemented as a program that is practically useful (modulo the design limitations mentioned below). This involves (at least) parsing Twelf, possibly type reconstruction, transformations based on prettyprinting rules, and finally generating output.

---

<sup>1</sup><http://twelf.plparty.org/>

## Limit of scope

Twelf is a somewhat complex language, and it is not clear how some signatures, notably those involving sophisticated use of HOAS, should be printed in a human-readable form. My program must be able to prettyprint all signatures that do not make use of HOAS, as well as some to-be-determined “reasonable” subset of those that do.

Additionally, my program will not pay special special care to rules whose prettyprinted representation is too large to fit on a page (or some other restricted medium).

## Learning goals

- Analysis of how to solve a somewhat complex problem (in this case, prettyprinting a formal language). This implies achieving a good understanding of the formal language.
- Analysing the use cases of the program to decide how the user will control its behaviour. In particular, how the user specifies how a given judgement should be printed.
- The writing of a nontrivial, usable program based on the conclusions of the above-mentioned analysis.

```

%%ppr-grammar exp
exp : type.
z : exp.
s : exp -> exp.
%%ppr (case E1 E2 E3) (\texttt{case $E1$ of $E2$ | $E3$})
case : exp -> exp -> (exp -> exp) -> exp.
%%ppr (lam E) (\lambda x.E [x])
lam : (exp -> exp) -> exp.
%%ppr (app $1 $2) ($1\ $2)
app : exp -> exp -> exp.
%%ppr-rule eval
%%ppr (eval $1 $2) ($1 \rightarrow $2)
eval : exp -> exp -> type.
ev_z : eval z z.
ev_s : eval E V -> eval (s E) (s V).
ev_case_z : eval E1 z -> eval E2 V -> eval (case E1 E2 E3) V.
ev_lam : eval (lam [x]E x) (lam [x]E x).
ev_app : eval (app E1 E2) V
  <- eval E1 (lam E1')
  <- eval E2 V'
  <- eval (E1' V') V.

```

Figure 1.1: Sample Twelf signatures

$$\begin{array}{c}
\text{exp} ::= z \mid s \text{ exp}_2 \mid \text{case exp}_1 \text{ of exp}_2 \mid \text{exp}_3 \mid (\lambda x. \text{exp}[x]) \mid \text{exp}_1 \text{ exp}_2 \\
\\
\frac{}{z \rightarrow z} \quad \text{(EV-Z)} \\
\\
\frac{E \rightarrow V}{(s E) \rightarrow (s V)} \quad \text{(EV-S)} \\
\\
\frac{E_1 \rightarrow z \quad E_2 \rightarrow V}{\text{case } E_1 \text{ of } E_2 \mid E_3 \rightarrow V} \quad \text{(EV-CASE-Z)} \\
\\
\frac{}{\lambda x. E[x] \rightarrow \lambda x. E[x]} \quad \text{(EV-LAM)} \\
\\
\frac{E_1 \rightarrow \lambda x. E'_1[x] \quad E_2 \rightarrow V' \quad E'_1[V'] \rightarrow V}{E_1 E_2 \rightarrow V} \quad \text{(EV-APP)}
\end{array}$$

Figure 1.2: Example of how Figure 1.1 could be prettyprinted