(Still) Program Extraction from Large Proof Developments

Luís Cruz-Filipe[1,2]

Bas Spitters[1]

[1] University of Nijmegen, Netherlands

[2] Center for Logic and Computation, IST (Lisbon), Portugal

Why?

- Large constructive library
- Coq has extraction mechanism
- It doesn't work. . .

Disclaimer

For reasons beyond the authors' control, none of the programs which will be discussed were executable. Therefore, all statements of type

$$\text{program A is} \left\{ \begin{array}{c} \text{more} \\ \text{as} \\ \text{less} \end{array} \right\} \text{efficient} \left\{ \begin{array}{c} \text{than} \\ \text{as} \\ \text{than} \end{array} \right\} \text{program B}$$

should be taken with the proverbial grain of salt.

Contents

Extraction

- BHK-interpretation: connectives
- Kleene's realizability: a more formal approach
- Curry–Howard isomorphism: proofs $\longleftrightarrow$ programs
- In practice: algorithm vs. properties; types as "markers"

FTA-logic

- No elimination of $\mathbf{Prop}$ terms over $\mathbf{Set}$ $\leadsto$ no function definition by cases
- All logic in $\mathbf{Set}$
- Extracted program too big

A solution?

Identify *computationally meaningful* propositions; put everything else in $\mathbf{Prop}$.

$\leadsto$ most proof terms can be put back in $\mathbf{Prop}$

$\leadsto$ significant amount of "dead code" is eliminated

(for more details see paper in Procs. TPHOLS 2003)

$$
\begin{aligned}
\neg &: s \to \mathbf{Prop} \\
\to &: s_1 \to s_2 \to s_2 \\
\vee &: s_1 \to s_2 \to \mathbf{Set} \\
\wedge &: s_1 \to s_2 \to \begin{cases} \mathbf{Prop} & s_1 = s_2 = \mathbf{Prop} \\ \mathbf{Set} & \text{otherwise} \end{cases} \\
\forall &: \Pi(A : s_1).(A \to s_2) \to s_2 \\
\exists &: \Pi(A : \mathbf{Set}).(A \to s) \to \mathbf{Set}
\end{aligned}
$$

Results

- FTA: extracts, compiles, runs... but does not terminate

- Rational numbers: everything is (almost) instantaneous

- Somewhere in between: e, π and $\sqrt{2}$

Computing e

$$e \stackrel{\text{def}}{=} \sum_{n=0}^{+\infty} \frac{1}{k!}$$

$\leadsto$ each term is a rational (constant sequence)

$\leadsto$ but much is going on. . .

Immediate Problems...

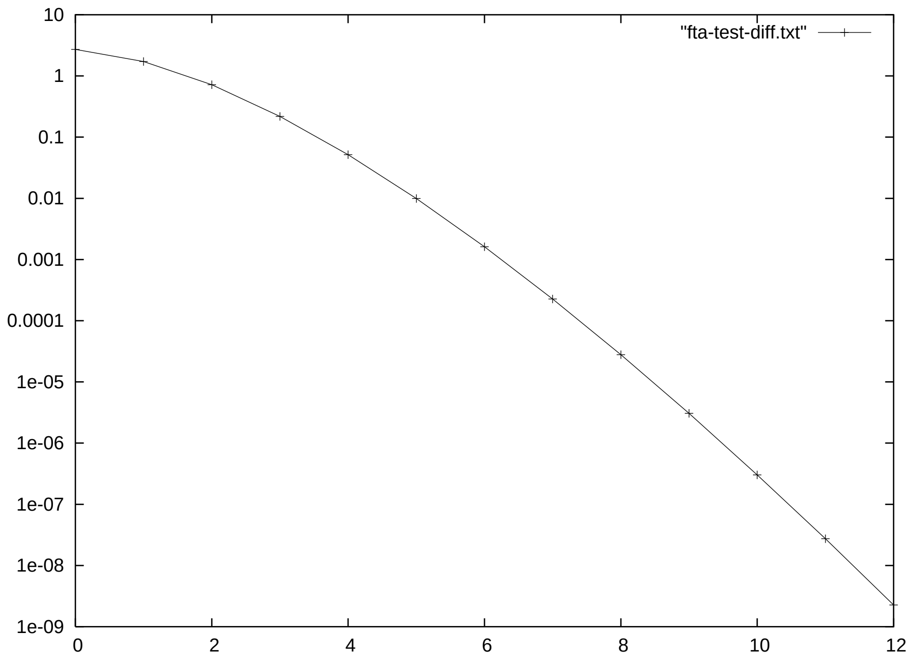
- Unary natural numbers

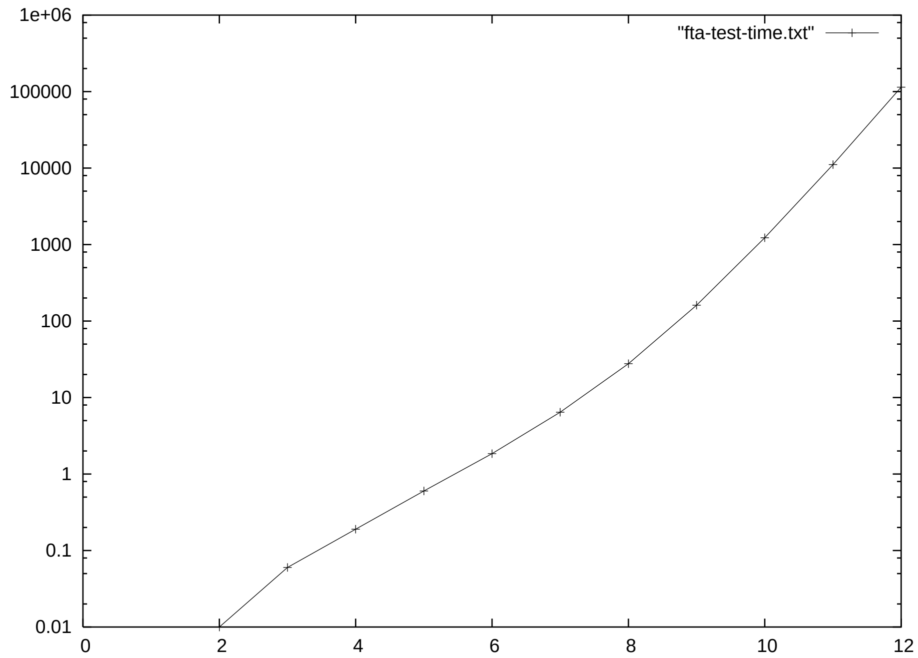
- A direct proof of $k! \neq 0$ requires computing $k!$ in unary notation

. . . & Solutions

- Directly inject $\mathbb{Z}^+$ into $\mathbb{R}$
- Prove $k! \neq 0$ by induction on k

Some statistics. . .





Still better (Thanks, Pierre!)

Optimize performance by working directly in the model:

- More efficient definition of factorial

- Simpler proofs and smaller proof terms

$\leadsto$ 100^{th} approximation in 77 seconds (with 157 correct digits)

The next step: $\sqrt{2}$

Different constructive formulations of the IVT. . .

- for total functions;
- for partial functions;
- for monotone functions;
- for locally non-constant functions;
- for polynomials.

. . . and different extracted programs:

- new $\sqrt{2}$ now yields first approximation after just 6 seconds (instead of 52 hours);

- complexity is still exponential;

- key lemma (for increasing version of IVT):

$$a < b \Rightarrow f(a) < f(b), \text{ where } f \text{ is the function being iterated}$$

Conclusions

- The more abstract the formalization, the less efficient the extracted program

- Obtaining a *working* program is far from straightforward

- Small, carefully thought, modifications in the formalization can make huge differences in the extracted program

- Future improvements in Coq may also make huge differences. . .

Future Work

- “Computable” $\sqrt{2}$

- Improving the extraction mechanism: pruning, modules (?)

- Eventually: the FTA