The SoftLab Isabelle Chronicles: Locally Nameless Experiences in Proving Type Safety

Νίκος Παπασπύρου

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Περί τίνος πρόκειται;

- Type safety proofs
  Αν ένα πρόγραμμα δεν έχει static type errors, τότε η εκτέλεσή του δεν οδηγεί σε σφάλμα
- Συνήθης πορεία
  - syntax
  - operational semantics
  - type system
  - safety = preservation + progress
- Mechanized!
  Με τη χρήση κάποιου proof assistant

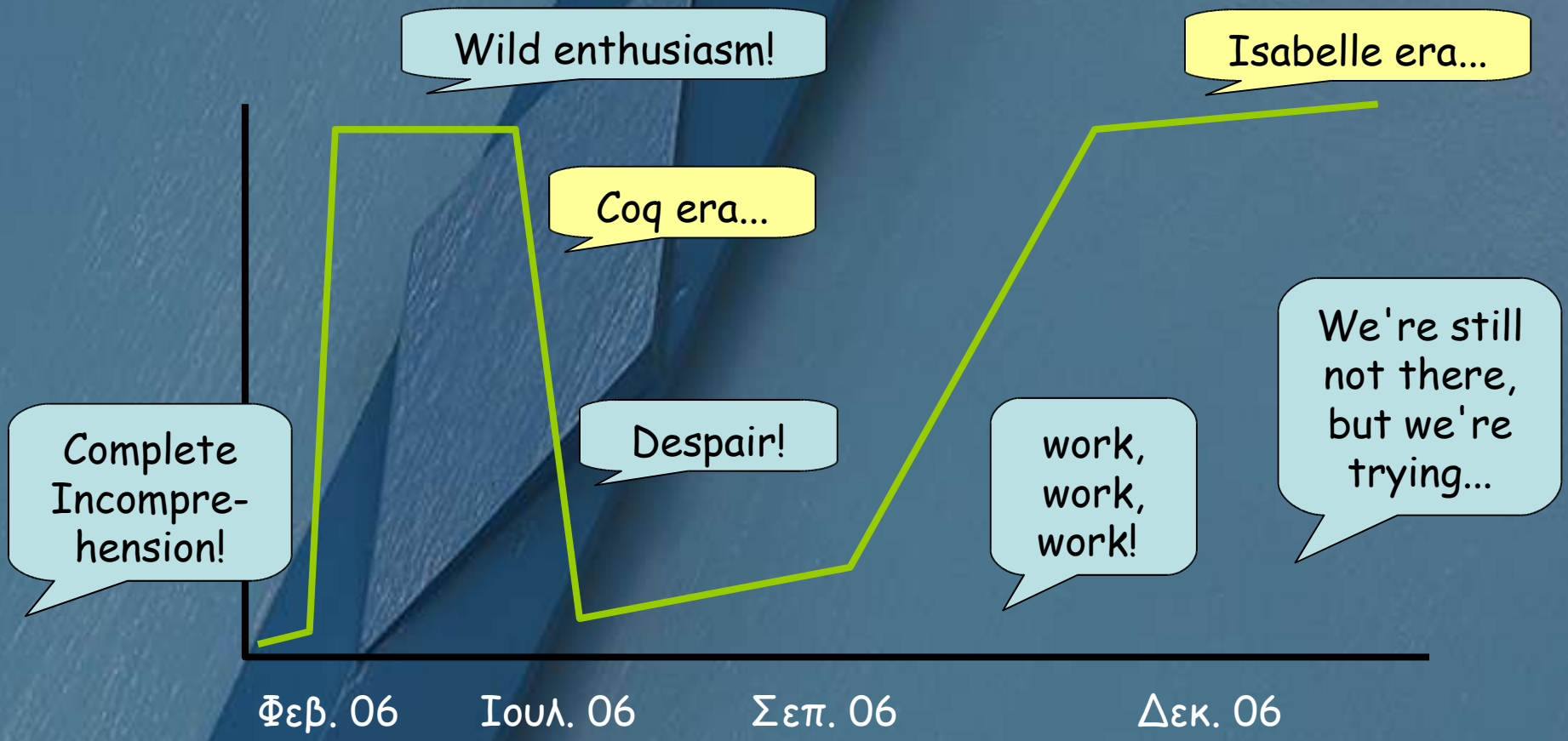
Γιατί όλα αυτά;

- (γιατί type safety;)
- γιατί όχι στο χαρτί;
  - εύκολο να γίνει λάθος
  - εύκολο να “διορθωθεί” ένα λάθος
  - αν λοιπόν είμαστε διατεθειμένοι να κάνουμε εξονυχιστική απόδειξη στο χαρτί, γιατί να μη χρησιμοποιήσουμε ένα proof assistant;

Από πού ξεκινήσαμε;

- regions + references στην Nflint
- ⇒ linear types
- ⇒ μετατροπή linear σε unrestricted: let!
- ⇒ linear lambda calculus + let!
- ⇒ type safety?
- ⇒ Coq (oops)
- ⇒ Isabelle

Πού βρισκόμαστε τώρα;



Πάμε πάλι, πού ακριβώς είναι το πρόβλημα;

- Το μεγαλύτερο πρόβλημα είναι η δέσμευση μεταβλητών (variable binding)
- Επίσης, τί “τύπο δεδομένων” χρησιμοποιώ για τα περιβάλλοντα τύπων; Γ, x:τ
- Πώς τα κάνουμε όλα αυτά στο χαρτί;
  - συνήθως το αγνοούμε

Δέσμευση μεταβλητών

- Named variables
- DeBruijn indices
- Locally nameless
- Nominal
- Higher-order abstract syntax

Named variables

- Ανάγκη μετονομασίας
- π.χ. στην αντικατάσταση

$$
\begin{array}{lcll}
x[x := N] & \equiv & N & \\
y[x := N] & \equiv & y & \text{αν } y \neq x \\
(P\,Q)[x := N] & \equiv & P[x := N]\,Q[x := N] & \\
(\lambda x.\,P)[x := N] & \equiv & \lambda x.\,P & \\
(\lambda y.\,P)[x := N] & \equiv & \lambda y.\,P[x := N] & \text{αν } y \neq x,\ \text{και } (y \notin FV(N)\ \text{ή}\ x \notin FV(P)) \\
(\lambda y.\,P)[x := N] & \equiv & \lambda z.\,P[y := z][x := N] & \text{αν } y \neq x,\ \text{και } y \in FV(N)\ \text{και } x \in FV(P), \\
 & & & \quad \text{όπου } z \notin FV(P) \cup FV(N)
\end{array}
$$

- και αλλού: typing rule για abstraction

DeBruijn indices

- Ιδέα: $\lambda f{:}\tau\to\tau.\ \lambda x{:}\tau.\ f\ (f\ x)$
  γίνεται: $\lambda[\tau\to\tau].\ \lambda[\tau].\ 1\ (1\ 0)$
- Αρκετά πλεονεκτήματα, αλλά...
- Τί γίνεται με τις “global” variables;
- Αντικατάσταση ακόμα πιο πολύπλοκη: χρειάζεται shifting

DeBruijn indices

$$
\begin{aligned}
\uparrow_c^d(\mathsf{k}) &= \begin{cases} \mathsf{k} & \text{if } k < c \\ \mathsf{k} + d & \text{if } k \geq c \end{cases} \\
\uparrow_c^d(\lambda.\mathsf{t}_1) &= \lambda.\ \uparrow_{c+1}^d(\mathsf{t}_1) \\
\uparrow_c^d(\mathsf{t}_1\ \mathsf{t}_2) &= \uparrow_c^d(\mathsf{t}_1)\ \uparrow_c^d(\mathsf{t}_2)
\end{aligned}
$$

Shifting

$$
\begin{aligned}
[\mathsf{j} \mapsto \mathsf{s}]\mathsf{k} &= \begin{cases} \mathsf{s} & \text{if } \mathsf{k} = \mathsf{j} \\ \mathsf{k} & \text{otherwise} \end{cases} \\
[\mathsf{j} \mapsto \mathsf{s}](\lambda.\mathsf{t}_1) &= \lambda.\ [\mathsf{j}+1 \mapsto \uparrow^1(\mathsf{s})]\mathsf{t}_1 \\
[\mathsf{j} \mapsto \mathsf{s}](\mathsf{t}_1\ \mathsf{t}_2) &= ([\mathsf{j} \mapsto \mathsf{s}]\mathsf{t}_1\ [\mathsf{j} \mapsto \mathsf{s}]\mathsf{t}_2)
\end{aligned}
$$

Substitution

Locally nameless

- Ιδέα:
  - ονόματα για “global” variables
  - **και** DeBruijn indices για bound variables
- Πλεονεκτήματα:
  - αντικατάσταση μόνο σε bound variables
  - χωρίς shifting
  - χωρίς μετονομασία των named variables
  - αλλά αντικατάσταση bound variables με named variables — “freshening”

Hands on now!

- Πρόταση dvitin (+1)
  - Emacs
  - Proofgeneral
  - x-symbol
  - Isabelle/HOL σε isar mode (structured proofs)
- Case study
  - simply typed lambda calculus
  - with heap-stored values
  - with references

Conclusion

- τα συμπεράσματα δικά σας...

proof?

by auto