Proof Checking Using Partial Functions

Hans de Nivelle

if($current_date < 1 october 2017)

then

Wrocław University, Wrocław

else

Nazarbayev University, Astana

Introduction

My (long term) goal is to obtain a proof checker that I like, and hopefully some more people.

Automated checking of mathematical proofs is important.

I have recently been reviewing a lot for respectable conferences and journals.

Even famous authors are often not able to express their thoughts properly in semi formal, natural language. It is a problem.

A formal mathematical language with a type checker would already be a big step forward.

I also would like to check some of the results of my PhD thesis. Things are not going quick.

Wish List

- Easy to introduce types. Easy type quantification and subtyping. (Lesson from modern programming languages)
- Communication and clarity is as important as correctness. (Another lesson from programming.)

  Proofs, definitions, theorems must be readable. Structure must be transparant.
- Contexts must be small and have only explicit parameters. Global variables are bad. (Yet another lesson from programming.)

Analogy with Programming?

We know how to make Turing-complete programming languages since 50-ies.

Progress has been in type systems, organization of large programs, and compiler technology.

Learn something from Object-Oriented programming?
Unfortunately, nobody can define what it is.

Programming languages are not systems. They have standard committees, one language can have different compilers. Verification is still completely system-oriented.

Verification is not as rewarding as programming.

Partial Classical Logic (PCL)

We are surrounded by partial functions:

$$\forall x{:}\,\mathrm{Real}\ (\sqrt{x})^2 = x,$$

$$\forall L{:}\,\mathrm{List}\ \ \mathrm{cons}(\mathrm{first}(L), \mathrm{rest}(L)) = L,$$

$$\forall m, n{:}\,\mathrm{Nat}\ 0 \leq (m \bmod n) \wedge (m \bmod n) < |n|.$$

Can be handled by relativization:

$$\forall x{:}\,\mathrm{Real}\ x \geq 0 \to (\sqrt{x})^2 = x,$$

$$\forall L{:}\,\mathrm{List}\quad L \neq \mathrm{nil} \to \mathrm{cons}(\mathrm{first}(L), \mathrm{rest}(L)) = L,$$

$$\forall m, n{:}\,\mathrm{Nat}\ n \neq 0 \to 0 \leq (m \bmod n) \wedge (m \bmod n) < |n|.$$

One can also relativize the types:

$$\forall x\ \mathrm{Real}(x) \wedge x \geq 0 \rightarrow (\sqrt{x})^2 = x,$$

$$\forall x\ \mathrm{List}(x) \wedge L \neq \mathrm{nil} \rightarrow \mathrm{cons}(\mathrm{first}(L), \mathrm{rest}(L)) = L,$$

$$\forall m, n\ \mathrm{Nat}(m) \wedge \mathrm{Nat}(n) \wedge n \neq 0 \rightarrow 0 \leq (m \bmod n) \wedge (m \bmod n) < |n|.$$

Is it really the same?

Yes, but only because the formulas are accidentily correct.

When formulas are not correct, their relativizations have unwanted meanings that are either too weak or too strong.

Built-in (to the logic) typing ensures that a formula becomes unusable when the typing rules are not respected. With relativization, this strictness is lost.

If You are certain that all formulas that You write are correct, then you don't need type checking.

PCL

Introduce new truth value **e** for meaningless formulas. Split $\rightarrow$ into $\rightarrow$ and [ ]. Split $\wedge$ into $\langle\ \rangle$ and $\wedge$.

$$\forall x\ [\ \mathrm{Real}(x)\ ]\ [\ x > 0\ ](\sqrt{x})^2 = x,$$

$$\forall x\ [\ \mathrm{List}(L) \wedge L \neq \mathrm{nil}\ ]\ \mathrm{cons}(\mathrm{first}(L), \mathrm{rest}(L)) = L,$$

$$\forall m, n[\ \mathrm{Nat}(m) \wedge \mathrm{Nat}(n)\ ][\ n \neq 0\ ]\ 0 \leq (m \bmod n) \wedge (m \bmod n) < |n|,$$

$$\forall m[\ \mathrm{Nat}(m)\ ]\ m \neq 0 \rightarrow \exists n\langle\ \mathrm{Nat}(n)\ \rangle\ \langle\ m \geq n\ \rangle\ m - n = 1.$$

PCL-operators (Strict)

The standard operators $\neg$, $\rightarrow, \wedge, \vee, \leftrightarrow$, $\forall, \exists$ have the usual semantics, ...

but always detect **e**. If one of the arguments is **e**, then result is **e**.

For the quantifiers, if there is a domain element d, for which $I_d^x(P) = \mathbf{e}$, then $I(\ Qx\ P) = \mathbf{e}$. Otherwise, interpretation as usual.

PCL-operators (Lazy)

Lazy implication $[A]B$:

If $I(A) = \mathbf{e}$, then $I([A]B) = \mathbf{e}$.

If $I(A) = \mathbf{f}$, then $I([A]B) = \mathbf{t}$.

If $I(A) = \mathbf{t}$, then $I([A]B) = I(B)$.

Lazy conjunction $\langle A\rangle B$:

If $I(A) = \mathbf{e}$, then $I([A]B) = \mathbf{e}$.

If $I(A) = \mathbf{f}$, then $I([A]B) = \mathbf{f}$.

If $I(A) = \mathbf{t}$, then $I([A]B) = I(B)$.

PCL-operators (Prop)

The # operator:

If $I(A) = \mathbf{f}, \mathbf{t}$, then $I(\# A) = \mathbf{t}$.

If $I(A) = \mathbf{e}$, then $I(\# A) = \mathbf{f}$.

$[\ ]$ is associated to $\forall$:

$$\forall x\ [N(x)]\ x \geq x.$$

$\langle\ \rangle$ is associated to $\exists$:

$$\exists x\ \langle N(x) \rangle\ x \geq x.$$

Contexts

PCL reasons with contexts:

$\forall x \ \# \ \mathrm{Nat}(x)$,

$\mathrm{Nat}(0)$,

$\forall x \ \mathrm{Nat}(x) \rightarrow \mathrm{Nat}(\ \mathrm{succ}(x)\ )$,

$\forall xy \ \mathrm{Nat}(x) \wedge \mathrm{Nat}(y) \rightarrow \# \ x \leq y$,

$\forall xy \ [\ \mathrm{Nat}(x) \wedge \mathrm{Nat}(y) \wedge \mathrm{Nat}(z)\ ] \ x \leq y \wedge y \leq z \rightarrow x \leq z$,

$\forall xy \ [\ \mathrm{Nat}(x) \wedge \mathrm{Nat}(y)\ ] \ y \leq x \rightarrow \mathrm{Nat}(x - y)$,

$\forall xy \ [\ \mathrm{Nat}(x) \wedge \mathrm{Nat}(y)\ ] \ x \leq y \rightarrow \exists z \ \langle\ \mathrm{Nat}(z)\ \rangle\langle\ x \geq z\ \rangle \ x - z = y$.

Definition: A context is a sequence of formulas $\Gamma_1, \ldots, \Gamma_m$, in which some Γ_j are possibly marked with a $^\theta$.

The formulas that are marked with θ are theorems, the others are assumptions.

Example

$$\#A, \#B, A, B, (A \wedge B)^\theta, \ldots$$

is a context.

A context is strongly valid if in every interpretation $I = (D, \mathbf{f}, \mathbf{t}, \mathbf{e}, [\,])$, for which there is an i, s.t. $I(\Gamma_i) \neq \mathbf{t}$, the first such i satisfies the following condition:

- Γ_i is not marked as a theorem, and $I(\Gamma_i) = \mathbf{f}$.

Examples

Not strongly valid:

$$A, A^\theta$$

Strongly valid:

$$\#A, A, A^\theta$$

Not strongly valid:

$$\#A, A, (A \vee B)^\theta$$

Strongly valid:

$$\#A, \#B, A, (A \vee B)^\theta$$

$$\#A, \#B, \ \neg B, \ \neg A \vee B, (\neg A)^\theta$$

PCL has strictness for types and preconditions: Nothing can be done with a formula that does not respect the types and the preconditions.

Undefined should not be confused with underspecified: Undefined objects are objects that one should not think, speak, or write about.

PCL still has truth-value based semantics, so that it is not too exotic. When a formula passes type checking, it becomes two-valued.

The notion of strong validity is essential. Otherwise, assuming something would implicitly make it well-typed.

There is no restriction on the form of types or preconditions.

Type checking can be very complicated, but in most cases it is not.

Auf zur Higher-Order!

Proof checking needs some higher-order logic or set theory. If one chooses set theory, one still needs some higher-order logic to define the comprehensions schemas.

Ergo: There is no way around higher-order logic.

Goal of PCL was to avoid types that are built-in into the logic.

In higher-order logic, this causes inconsistency, because one can define a fixed point of $\neg$.

Some weak form of typing will be unavoidable, just enough to avoid inconsistency.

Metatypes

Metatypes are recursively defined as follows:

- O and T are metatypes.
- If $U_1, \ldots, U_m$ with $m \geq 2$ is a sequence of metatypes, then $U_1 \times \cdots \times U_m$ is a metatype.
- If U and V are metatypes, then $(U)V$ is a metatype (denoting functions from U to V.)

O is the metatype of objects, T is the metatype of truth-values $\{\mathbf{f}, \mathbf{e}, \mathbf{t}\}$.

Currying is restricted. It simplifies the logic, but makes its use harder.

Explicit Types are Unpleasant

The main design goal of PCL is to be able to express type conditions explicitly:

$$\forall x{:}O\# \; N(x).$$

$$N(0).$$

$$\forall x{:}O \quad N(x) \to N(\mathrm{succ}(x)).$$

$$\forall x, y{:}O \quad N(x) \wedge N(y) \to N(x + y).$$

$$\forall x, y{:}O \; [\; N(x) \wedge N(y) \;] \; (x \geq y) \to N(x - y).$$

This goal was reached, but in unpleasant way.

Das Typenwalhall:

Von Morgen bis Abend, in Müh' und Angst,

nicht wonnig ward sie gewonnen!

Especially for higher-order types:

Example: A recursion operator **rec** on N.

$$\forall P{:}(O)T \;\; [\, \forall x{:}O \; \#P(x) \,]$$
$$\forall a{:}O \; P(a) \rightarrow$$
$$\forall f{:}(O,O)O \;\; (\forall n,p{:}O \;\; N(n) \rightarrow P(p) \rightarrow P(\; f(n,p)\;)\;) \rightarrow$$
$$\forall n{:}O \; N(n) \rightarrow P(\; \mathrm{rec}(a,f,n)\;).$$

Readable Type Definitions (1)

Introduce an alternative application operator :

$(t_1, \ldots, t_n){:}u$ is (sort of) alternative notation for $u(t_1, \ldots, t_n)$.

It follows that if $t_1, \ldots, t_n$ have metatypes $X_1, \ldots, X_n$ and term u has metatype $(X_1, \ldots, X_n)T$ then $(t_1, \ldots, t_n){:}u$ has metatype T.

Furthermore, assume a reduction

$$(t_1, \ldots, t_n){:}u \implies u(t_1, \ldots, t_n).$$

Readable Type Definition (2)

Define the following operators:

PROP: (T)T, FORM: (T)T, OBJ: (O)T.

Operator $\times$ with metatype rule: If u_1 has metatype $(X_1, \ldots, X_m)T$, and t_2 has metatype $(Y_1, \ldots, Y_n)T$, then $u_1 \times u_2$ has metatype $(X_1, \ldots, X_m, Y_1, \ldots, Y_n)T$.

Operator $\Rightarrow$ with metatype rule: If term u_1 has metatype $(X_1, \ldots, X_n)T$ and term u_2 has metatype $(Y)T$ then $u_1 \Rightarrow u_2$ has metatype $((X_1, \ldots, X_n)Y)T$.

Operator Π with metatype rule: If term u_1 has metatype $(X_1, \ldots, X_n)T$ and extending the context by $v_1{:}X_1, \ldots, v_n{:}X_n$ results in u_2 having metatype $(Y)T$, then
$\Pi\ (v_1, \ldots, v_n){:}u_1\ u_2$ also has metatype $(Y)T$.

Type Reductions (1)

Define the following reductions:

$$\begin{aligned} A{:}\,\mathrm{PROP} &\Longrightarrow \#A \\ A{:}\,\mathrm{FORM} &\Longrightarrow \top \\ A{:}\,\mathrm{OBJ} &\Longrightarrow \top \end{aligned}$$

Operator $\times$

If u_1 has metatype $(X_1, \ldots, X_i)T$ then

$$(t_1, \ldots, t_n){:}\,(u_1 \times u_2)$$

$$\Longrightarrow$$

$$(t_1, \ldots, t_i){:}\,u_1 \wedge (t_{i+1}, \ldots, t_n){:}\,u_2.$$

Type Reductions (2)

Operator $\Rightarrow$

If u_1 has metatype $(X_1, \ldots, X_n)T$, then

$$f{:}\,(u_1 \Rightarrow u_2)$$

$$\Longrightarrow$$

$$\forall v_1{:}\,X_1 \cdots v_n{:}\,X_n\ (v_1, \ldots, v_n){:}\,u_1 \rightarrow f(v_1, \ldots, v_n){:}\,u_2.$$

Operator Π

If u_1 has metatype $(X_1, \ldots, X_n)T$, then

$$t{:}\,\Pi\ (t_1, \ldots, t_n){:}\,u_1\ u_2$$

$$\Longrightarrow$$

$$\forall v_1{:}\,X_1 \cdots v_n{:}\,X_n\ [\ (v_1, \ldots, v_n){:}\,u_1\ ]\ t{:}\,u_2.$$

Type Reductions (3)

Now it is possible to write:

$$N{:}\,\mathrm{OBJ} \Rightarrow \mathrm{PROP},$$

$$0{:}\,N,$$

$$S{:}\,\mathrm{N} \Rightarrow \mathrm{N}.$$

We have

$$N{:}\,\mathrm{OBJ} \Rightarrow \mathrm{PROP}$$

$$\Longrightarrow$$

$$\forall x{:}\,O\ \top \rightarrow N(x){:}\,\mathrm{PROP}$$

$$\Longrightarrow$$

$$\forall x{:}\,O\ \top \rightarrow \#\ N(x)$$

Type Reductions (4)

Declare

$$\text{rec: } \Pi\text{P:O(T) [P:(OBJ} \Rightarrow \text{PROP)] P} \times (\text{N} \times \text{P} \Rightarrow \text{P}) \Rightarrow \text{N} \Rightarrow \text{P}.$$

It follows that rec must have metatype $(O, (O, O)O, O)O$.

The declaration can be reduced into:

$$\forall P\text{:}(O)T \text{ [} P\text{:(OBJ} \Rightarrow \text{PROP)] rec:P} \times (\text{N} \times \text{P} \Rightarrow \text{P}) \Rightarrow \text{N} \Rightarrow \text{P}.$$

After that,

$$P\text{:(OBJ} \Rightarrow \text{PROP)} \quad \Longrightarrow \forall x\text{:}O \ \#P(x),$$

and

$$\text{rec:P} \times (\text{N} \times \text{P} \Rightarrow \text{P}) \Rightarrow \text{N} \Rightarrow \text{P} \Longrightarrow$$

$$\forall a\text{:}O, \quad f\text{:}(O, O)O, \quad n\text{:}O,$$

$$a\text{:}P \wedge f\text{:}(N \times P \Rightarrow P) \wedge n\text{:}N \rightarrow \text{rec}(a, f, n)\text{:}P.$$

Some More

Some more type constructors and their reductions:

$$
\begin{array}{lll}
(t_1, \ldots, t_n)\colon\ u_1 \cap u_2 & \Longrightarrow & (t_1, \ldots, t_n)\colon u_1 \wedge (t_1, \ldots, t_n)\colon u_2 \\
(t_1, \ldots, t_n)\colon\ u_1 ; u_2 & \Longrightarrow & \langle\ (t_1, \ldots, t_n)\colon u_1\ \rangle (t_1, \ldots, t_n)\colon u_2 \\
(t_1, \ldots, t_n)\colon\ u_1 \times u_2 & \Longrightarrow & (t_1, \ldots, t_i)\colon u_1 \wedge (t_{i+1}, \ldots, t_n)\colon u_2
\end{array}
$$

Now we can declare:

$\geq\colon N \times N \Rightarrow \mathrm{PROP}$
$-\colon (\ N \times N; (\geq)\ ) \Rightarrow N$

$\mathrm{div}\colon \mathrm{N} \times (\ \mathrm{N};\ \lambda \mathrm{x}\colon \mathrm{O}\ \mathrm{x} \neq 0\ ) \Rightarrow \mathrm{N}.$

Advantages of Type Reductions

- Types can be expressed in readable fashion most of the time.
- Metatypes can be completely hidden from the user. They can be deduced from the type expressions.
- Flexibility is still there, when needed.

User Definable Metatypes

The user should be able to define new metatypes. A user defined metatype is a **struct** definition with a characteristic predicate, defined by a context:

struct preorder($D{:}(O)T$, $R{:}(O,O)T$)

$\forall x{:}O\ \#D(x)$

$\forall x_1, x_2{:}O\ D(x_1) \wedge D(x_2) \rightarrow \#R(x_1, x_2)$

$\forall x{:}O\ [\ D(x)\ ]\ R(x,x)$

$\forall x, y, z{:}D\ [\ D(x) \wedge D(y) \wedge D(z)\ ]\ R(x,y) \wedge R(y,z) \rightarrow R(x,z)$

Metatypes must be minimal, in the sense that they cannot be weakened in a meaningful fashion.

Definitions

It is possible to attach definitions to a **struct**, which act like additional fields:

One can define in **preorder**:

$$\equiv := \lambda x, y{:}O\ R(x,y) \wedge R(y,x)$$

$$\succ := \lambda x, y{:}O\ R(x,y) \wedge \neg R(y,x)$$

Properties

Properties are user defined predicates over user defined metatypes. For example:

interface equivalence:
$\langle$**preorder**$\rangle$ $\forall x, y{:}O$ [$D(x) \wedge D(y)$] $R(x, y) \rightarrow R(y, x)$

interface partorder:
$\langle$**preorder**$\rangle$ $\forall x, y{:}O$ [$D(x) \wedge D(y)$] $x \equiv y \leftrightarrow x = y$

interface totalorder:
$\langle$**partorder**$\rangle$ $\forall x, y{:}O$ [$D(x) \wedge D(y)$] $R(x, y) \vee R(y, x)$

Logically, properties are just strict conjunctions.

One can also view metatypes as substantives, and properties as adjectives.

Some Open Questions

Proper treatment of quaternions and complex numbers. They have fields, which suggest that they are **struct** (separate metatype). At the same time, they are like numbers (part of O).

Some natural constructions (e.g. subset construction on finite automata, viewing functions are groups) require set theory axioms in order to stay in O.

Is it all worth it? Depends on the set of potential users.

An earlier version failed (was userunfriendly).