Markov's Principle for Propositional Type Theory

Alexei Kopylov Aleksey Nogin

Cornell

September 13, 2001

Computational Type Theories and Classical Reasoning

- Computational type theories (such as Martin-Löf type theory) are widely used in many different areas of logic and computer science
- In this talk we focus on the *practical* benefit of being able to extract computational content ("evidence" or "witness") from a constructive proof in such type theory.

Question: If we believe in classical reasoning, but still want to extract constructive content from proofs, how can we extend the type theory?

There are many possible answers to this question.

Type Theory with a Markov–like Rule

Our answer: We can add a rule to type theory that is:

- very simple
- analogous to Markov's principle, but more general
- quantifier–free (only needs propositional connectives and a modal operator)
- a resulting modal fragment of the type theory is a known modal logic
- allows classical reasoning in subderivations that do not directly contribute to computational content of the proof without a need for some *external* accounting (such as in Howe, 1996)

Propositions-as-types

- We assume that each proposition is identified with the type of its "*witnesses*".
- For example, a witness for $\exists x : A. B(x)$ would be a pair $\langle a, b \rangle$ where $a \in A$ and b is a witness in $B(a)$.
- We can always reconstruct a witness of P from a derivation of P.
- The opposite is not always true: in general, a witness of P may range from a full encoding of a derivation of P to a trivial constant.
- The witnesses are the *carrier of a computational content*.

Squash OPERATOR

- Informally, squash operator *hides* a witness of the type "squash-ing" all contents down to a single point:
  - $[A]$ states that A is non-empty type without providing a witness for A.
  - $[A]$ is a singleton type with an element $\bullet$ when A is non-empty and is empty otherwise.
  - Squash operator was first introduced in Constable *at al*, 1986

Squash STABILITY

- Type A is *squash–stable* iff we can provide a witness term w such that $w \in T$ when T is non-empty.
- Examples
  - Squashed type: $[A]$ (witness $\bullet$)
  - Negation: $A \to \bot$ (witness $\lambda x.x$)
  - Membership and equality: $a \in A$, $a = b \in A$ (witness $\bullet$)
  - Well-formedness: A Type (witness $\bullet$)
  - Program P terminates on input i (witness: just run P!)
- The squash operator allows us to formulate squash stability *internally*:

$$A \text{ is } \textit{squash–stable} \text{ iff } [A] \to A$$

- This notion of *squash–stability* is very similar to the notion of *computational redundancy* (Backhouse, Chisholm, Malcolm and Saaman, 1989)

PROVING Squash-stable STATEMENTS CLASSICALLY

- We explore the use of classical reasoning to prove squash–stable statements:

$$\frac{\Gamma;\, x : A \vdash B \quad \Gamma;\, y : \neg A \vdash B \quad \Gamma;\, v : [B] \vdash B}{\Gamma \vdash B} \qquad (*)$$

  (assuming A is a well–formed proposition)
- All of the following are equivalent to $(*)$:
  1. $\neg\neg A \leftrightarrow [A]$
  2. $[A \vee \neg A]$
  3. $\neg\neg(a = b \in A) \to a = b \in A$
- Each of the above rules is valid in standard PER semantics of type theory under *classical meta-theory* (*cf.* Allen, 1987).

MARKOV'S PRINCIPLE

```
If it is not a case that a program does not terminate,
then it does terminate.
```

A. Markov, 1954

Formalization:

$$\forall x.(A(x) \vee \neg A(x)) \quad \& \quad \neg\neg\exists x.A(x) \quad \Rightarrow \quad \exists x.A(x)$$

Justification:

Program P: try every natural number x and check whether $A(x)$ is true.
Stop when find x s.t. $A(x)$ and return that x.

Markov's principle in Type Theory

Markov's principle is derivable in type theory with our $[A \vee \neg A]$ principle!

It is easy to prove that if it is the case that $\exists x : \mathbb{N}.A(x)$, then P will find such an x.

Therefore

$$\frac{\dfrac{d : \forall x : \mathbb{N}.(A(x) \vee \neg A(x));\ \exists x : \mathbb{N}.A(x)\ \vdash\ P \in \exists x : \mathbb{N}.A(x)}{d : \forall x : \mathbb{N}.(A(x) \vee \neg A(x));\ [\exists x : \mathbb{N}.A(x)]\ \vdash\ \exists x : \mathbb{N}.A(x)}}{d : \forall x : \mathbb{N}.(A(x) \vee \neg A(x));\ \neg\neg\exists x : \mathbb{N}.A(x)\ \vdash\ \exists x : \mathbb{N}.A(x)} (1)$$

Implementation

This version of type theory have been implemented in the MetaPRL system (http://metaprl.org/) *in addition* to more traditional NuPRL – like type theory.

All rule equivalences that we have mentioned so far were *proven* in MetaPRL .

Note: NuPRL group continues to use pure constructive logic for most purposes in both NuPRL and MetaPRL systems.

Propositional Type Theory and Modal Logic

- **Question:** What logic corresponds to propositional fragment of type theory with squash operator.
- **Answer:** PLL$^+$ — a variant of the propositional Lax logic (PLL) that was already presented on Tuesday with $[\cdot]$ as a "diamond" modality.

$$\text{PLL}^+ = \text{PLL} + \neg[\bot]$$

- To allow restricted classical reasoning we add a new axiom

$$\text{PLL}^* = \text{PLL}^+ + [A \vee \neg A]$$

- The names PLL, PLL$^+$ and PLL* were introduced in Fairtlough & Mendler, 1997

PLL Examples

We can prove the following facts in PLL$^+$:

- False is squash–stable: $[\bot] \to \bot$
- Negation is always squash–stable: $[\neg A] \to \neg A$
- Conjunction of two squash–stable propositions is always squash–stable.
- If A is decidable then A is squash–stable:

$$(A \vee \neg A) \to ([A] \to A)$$

The following is derivable in PLL*:

- Squash commutes with implication:

$$([A] \to [B]) \leftrightarrow [A \to B]$$

Example: THE THEORY OF PARTIAL FUNCTIONS ("BAR" TYPES)

- The NuPRL theory of "bar" types (Smith and Constable, 1987) is incompatible with $P \vee \neg P$.
- We can derive $\neg\big(\forall P : \texttt{Prop}.\,(P \vee \neg P)\big)$ in NuPRL theory with "bar" types.
- However our $\mathbf{[}P \vee \neg P\mathbf{]}$ is consistent with NuPRL theory with "bar" types.
- Note, our $\mathbf{[}P \vee \neg P\mathbf{]}$ implies $\forall P : \texttt{Prop}.\mathbf{[}P \vee \neg P\mathbf{]}$, but not $\mathbf{[}\forall P : \texttt{Prop}.\,(P \vee \neg P)\mathbf{]}$

Example: "Fuzzy" FINITE SETS

- Consider a type $\{i : [0..n] \mid P(i)\}$.
- We know such type is finite, but in general we do not even know how much elements it has.
- Can we prove $\forall f : \{i : [0..n] \mid P(i)\} \to \mathbb{N}$. "$f$ is limited"?
- If "f is limited" is squash–stable, our $\mathbf{[}A \vee \neg A\mathbf{]}$ principle allows us to consider P to be decidable.