Markov's Principle for Propositional Type Theory*

Alexei Kopylov and Aleksey Nogin

Department of Computer Science
Cornell University
Ithaca, NY 14853
{kopylov,nogin}@cs.cornell.edu

Abstract. In this paper we show how to extend a constructive type theory with a principle that captures the spirit of Markov's principle from constructive recursive mathematics. Markov's principle is especially useful for proving termination of specific computations. Allowing a limited form of classical reasoning we get more powerful resulting system which remains constructive and valid in the standard constructive semantics of a type theory. We also show that this principle can be formulated and used in a propositional fragment of a type theory.

1 Introduction

1.1 Overview

The main goal of this paper is to support limited classical reasoning in a generally intuitionistic framework. We use a `squash` operator for this purpose. This operator can creates a proposition stating that a certain type is non-empty without providing an inhabitant, i.e. `squash` "forgets" proofs. It was first introduced in [6] and also used in [15] and MetaPRL system [9,10]. Using `squash` it is possible to define the notion of squash–stability, which is similar to self–realizability.

The `squash` operator can be considered as a modality. The propositional logic equipped with this modality can express a principle that allows turning classical proofs of squash–stable propositions into constructive ones. This principle is valid in a standard type theory semantics if we consider it in the classical metatheory. Therefore this principle does not destroy the constructive nature of type theory in the sense that we can always extract a witness term from a derivation.

It turns out that this principle implies Markov's principle providing us a propositional analog of Markov's principle. It is rather surprising that such analog exists because normally one needs quantifiers in order to formulate Markov's principle.

We also show an equivalent way of defining the same principle using a membership type instead of the `squash` operator.

* This work was partially supported by AFRL grant F49620-00-1-0209

L. Fribourg (Ed.): CSL 2001, LNCS 2142, pp. 570–584, 2001.

1.2 Markov's Constructivism

Constructive mathematics is interesting in Computer Science because of program correctness issues. There are several approaches to constructivism (see [4,5,19] for an overview). We are especially interested in the constructive recursive mathematics (CRM) approach developed by Markov [12,13] and in constructive type theories (especially those that are based on Martin-Löf type theory [14]) since we believe them to be highly relevant to Computer Science. In this paper we demonstrate how to apply the ideas of CRM to a constructive type theory thus creating a more powerful type theory that combines the strengths of both approaches to constructive mathematics.

According to Markov's CRM approach, all objects are algorithms, where algorithms are understood as finite strings in a finite alphabet. All logical connectives are understood in a constructive way. That is, a statement is true if and only if there exists an algorithm that produces a *witness* of this statement. For example, the witness for $\forall x.A(x) \vee \neg A(x)$ is an algorithm that for a given x tells us either that $A(x)$ is true (and provides a witness for $A(x)$) or that $\neg A(x)$ is true (and provides a witness for $\neg A(x)$). That means that $\forall x.A(x) \vee \neg A(x)$ is true only for decidable predicates A. Since not all predicates are decidable, Markov's school has to reject the rule of excluded middle.

Note that a witness of a proposition does not "prove" that proposition. For example, $\forall x.A(x) \vee \neg A(x)$ is *true* when there is a decision algorithm for A, but it does not mean that there is a *proof* [1] that this algorithm works properly (i.e. always terminates and gives the correct answer). In this respect the constructive recursive mathematics differs from the Brouwer–Heyting–Kolmogorov's intuitionism. We will return to the topic of differences between "proof witnesses" and "algorithm witnesses" in Section 1.4.

The question arisen in the CRM is *which means is one allowed to use in order to establish that a particular algorithm is indeed a witness for the given proposition?* This is not an obvious question since the termination problem is undecidable. Even if algorithm terminates for every input, we can not test it explicitly, because there are infinitely many possible inputs. But to establish that an algorithm is applicable to an object a, the algorithm does not have to be executed explicitly from the beginning to the end. According to Markov [13] we can prove this by contradiction. That is, we are allowed use some classical reasoning to prove that a particular algorithm has some particular properties.

1.3 Markov's Principle

The Markov school uses the intuitionistic predicate arithmetic with an additional principle (known as Markov's principle):

$$\forall x.(A(x) \vee \neg A(x)) \to \neg\neg\exists x.A(x) \to \exists x.A(x) \tag{1.1}$$

[1] In this paper we use terms *proof* and *derivation* interchangeably.

where variables range over natural numbers. Note that this principle does not hold for Brouwer–Heyting–Kolmogorov's intuitionism, thus Markov's principle distinguishes these two schools of constructivism.

Here is the justification of this principle in the CRM framework. Assume $\forall x.(A(x) \vee \neg A(x))$. Then there exists an effective procedure that for every x decides whether $A(x)$ or $\neg A(x)$. To establish (1.1), we need to write a program that produces the witness of $\exists x.A(x)$. We can achieve that by writing a program that will try every natural number x and check $A(x)$ until it finds such an x that $A(x)$ is true. We know that it is impossible that such x would not exist (because of $\neg\neg\exists x.A(x)$). Therefore it is impossible that this algorithm does not terminate. Hence according to recursive constructivism it eventually stops.

Markov's principle is an important technical tool for proving termination of computations. Adding Markov's principle to a traditional constructive type theory would considerably extend the power of the latter in a pivotal class of verification problems.

1.4 Type Theory

We assume the type theory under consideration adheres to the *propositions–as–types principle*. This principle means that a proposition is identified with the type of all its witnesses. A proposition is considered true if the corresponding type is inhabited and is considered false otherwise. This makes terms *an element of a type* and *a witness of a proposition* synonyms. The elements of a type are actually λ–terms, i.e. programs that evaluates to a "canonical" element of this type.

We also assume that the type theory is extensional. That is, to prove that a term f is a function from A to B, it should be sufficient to show that for any $a \in A$ the application fa eventually evaluates to an element of the type B. This allows us to deal with recursive functions that we can prove will always terminate. We will use the `fix` operator to define recursive functions, where $\texttt{fix}(f.p[f])$ is defined as $(\lambda x.p[xx])(\lambda x.p[xx])$, i.e. `fix` is the operator with the following property $\texttt{fix}(f.p[f]) \mapsto p[\texttt{fix}(f.p[f])]$. Although the general typing rule for `fix`

$$\frac{f : A \to A \vdash p[f] \in A \to A}{\vdash \texttt{fix} f.p[f] \in A \to A}$$

is unsound, but for some particular p we can prove that $\texttt{fix}(f.p[x])$ is a well-typed function.

Note that in an extensional type theory *a witness of a proposition* T is not the same as *a derivation of a proposition* T. In general, a witness (i.e. an element) of the type T may potentially range from full encoding of some derivation of T to a trivial constant. For example, if V is an empty type, then *every* function has type $V \to W$, e.g. a function $\lambda x.foo$ is an element of $(A \wedge \neg A) \to \bot$ (although $\lambda x.foo$ does not encode any derivations of proposition $(A \wedge \neg A) \to \bot$). Note that the question of whether a particular term is a witness of a particular proposition is in general undecidable.

We assume that the type theory has a membership type – "$t \in T$" which stands for the proposition "t is an element of type T". The only witness of a membership proposition is a special constant $\bullet$[2].

We assume that $t \in A$ implies A. The inverse should also be true:

Property 1.1. If we can prove $\Gamma \vdash A$ then there is a term t such that $\Gamma \vdash t \in A$.

Remark 1.2. The reason we want judgments of the form $\Gamma \vdash T$ and not just $\Gamma \vdash t \in T$ is that we are interested in type theories that can be used as a foundation for theorem provers. In a theorem prover situation we want user to be able to state and prove a judgment of the for $\Gamma \vdash T$ and have the system "extract" t from the resulting derivation instead of being required to figure out and provide t upfront.

In this paper we present several formal derivations. These derivations were machine–checked in the MetaPRL system [10]. The rules used in those derivations are summarized in Appendix A. The results of Sections 2, 3 and 4 are valid for any type theory containing this set of rules and satisfying Property 1.1. The results of the sections Sections 5 and 6 also require an intentional semantics. NuPRL is an example of a type theory that satisfies all these constraints.

However, most of our ideas can be easily applied to an even wider class of type theories. For example, typing rules are not really essential. Additionally, we do not need a membership type to express Markov's principle, we can use the `squash` operator instead (Section 4). And as Section 7 shows, our form of Markov's principle can be used even in a purely propositional fragment of type theory without arithmetic and quantifiers.

2 Constructive Recursive Mathematics in a Type Theory with a Membership Type

Suppose one has proved that A implies $t \in T$ and $\neg A$ also implies $t \in T$. Then *classically* we can conclude that T is inhabited. Moreover the philosophy of recursive constructivism allows us to conclude that T is true *constructively*, because we can explicitly provide a constructive object t as an element of T. In other words, since the witness of T (which is just t) does not depend on the proof of $t \in T$, then T has a uniform witness regardless whether A is true or not (although the *proof* that t is a witness of T may depend on A).

This argument establishes that the following type theory rule is valid according to recursive constructivism:

$$\frac{\Gamma; x : A \vdash t \in T \quad \Gamma; y : \neg A \vdash t \in T \quad \Gamma \vdash A\,\mathrm{Type}}{\Gamma \vdash t \in T} \tag{2.1}$$

This rule formalizes exactly the philosophy of the recursive constructivism.

[2] MetaPRL system [9,10] uses the unit element () or "it" as a $\bullet$, NuPRL uses `Ax` and [18] uses *Triv*.

Remark 2.1. Note that in NuPRL–like type theories $t \in T$ is well-formed only when t is in fact an element of T. Therefore the rule stating that $\neg\neg(t \in T)$ implies $t \in T$ would be useless. On the other hand, in NuPRL type theory (2.1) is equivalent to

$$\frac{\Gamma \vdash (s \in T) \quad \Gamma \vdash (t \in T) \quad \Gamma \vdash \neg\neg(s = t \in T)}{\Gamma \vdash s = t \in T} \tag{2.2}$$

but the proof of (2.2) $\Rightarrow$ (2.1) is very NuPRL–specific.

3 Squashed Types and Squash–Stability

Below we will assume that our type theory contains a "`squash`" operator. For each type A we define a type $\lceil A \rceil$ ("squashed A") which is empty *if and only if* A is empty and contains a single element $\bullet$ when A is inhabited. Informally one can think of $\lceil A \rceil$ as a proposition that says that A is a *non-empty type*.

We define the `squash` operator as a primitive type constructor with the following rules (cf. [15])[3]:

$$\frac{\Gamma \vdash A}{\Gamma \vdash \lceil A \rceil}(I_{sq}) \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \bullet \in \lceil A \rceil}(M_{sq})$$

$$\frac{\Gamma;\, x : A;\, \Delta \vdash \lceil B \rceil}{\Gamma;\, v : \lceil A \rceil;\, \Delta \vdash \lceil B \rceil}(E_{sq})\ (v, x \text{ are not free in } \Delta, B)$$

$$\frac{\Gamma \vdash \lceil t \in T \rceil}{\Gamma \vdash t \in T}(MembershipSqstable)$$

$$\frac{\Gamma \vdash A\,\mathrm{Type}}{\Gamma \vdash \lceil A \rceil\,\mathrm{Type}}(W_{sq})$$

Note that (E_{sq}) and $(MembershipSqstable)$ can be replaced by one rule

$$\frac{\Gamma;\, x : A;\, \Delta[\bullet] \vdash t \in T}{\Gamma;\, v : \lceil A \rceil;\, \Delta[v] \vdash t \in T}(E^2_{sq})\ (v, x \text{ are not free in } \Delta, t, T)$$

The (I_{sq}) rule allows us to prove that $A \vdash \lceil A \rceil$. Note that $\lceil A \rceil$ does not imply A, because $\lceil A \rceil$ does not provide a witness for A. We can only derive that $\lceil A \rceil \vdash \neg\neg A$.

Squash operator allows us to formulate an important notion of squash stability. Although $\lceil A \rceil$ does not provide a witness for A in general, in some cases

[3] In a type theory that has a set type (also sometimes called "subset type") constructor, the squashed type $\lceil A \rceil$ can also be defined as $\{x : \mathsf{Unit} \mid A\}$ where Unit is a singleton type which contains only $\bullet$ and x is a variable that does not occur in A.

we know what witness would be in the case when A is non-empty. For example we know that if $t \in T$ is true, then $\bullet$ is the witness for the type $t \in T$. We will refer to such types as *squash–stable* types. For such types we can conclude that $\lceil A \rceil \vdash A$.

Definition 3.1. *A type T is squash–stable (in context Γ) when $\Gamma;\, x : T \vdash t \in T$ is provable for some t that does not have free occurrences of x.*

Using `squash` operator we can formulate the squash stability as a proposition in a type theory.

Lemma 3.2. *T is squash–stable in a context Γ if one can derive*

$$\Gamma;\, v : \lceil T \rceil \vdash T$$

Proof. Suppose T is squash–stable. Then we have the following derivation of $\Gamma;\, v : \lceil T \rceil \vdash T$:

$$\dfrac{\dfrac{\Gamma;\, x : T \vdash t \in T}{\Gamma;\, v : \lceil T \rceil \vdash t \in T}(E^2_{sq})}{\Gamma;\, v : \lceil T \rceil \vdash T}$$

Now assume that $\Gamma;\, v : \lceil T \rceil \vdash T$. Then for some term t we have $\Gamma;\, v : \lceil T \rceil \vdash t \in T$. Term t may depend on v, i.e. v may be a free variable of t. Let $t = t[v]$. Now let x be an arbitrary variable not occurring freely in t. Now we can derive the following:

$$\dfrac{\dfrac{\dfrac{}{\Gamma;\, x : T \vdash T}}{\Gamma;\, x : T \vdash \bullet \in \lceil T \rceil}(M_{sq}) \qquad \Gamma;\, v : \lceil T \rceil \vdash t[v] \in T}{\Gamma;\, x : T \vdash t[\bullet] \in T}(Let)$$

Therefore T is squash–stable.

We have already seen that $t \in T$ is a squash–stable type. Squash type itself is also squash–stable because we know that $\bullet \in \lceil A \rceil$ whenever $\lceil A \rceil$ is true. Other examples of squash–stable types include empty type ($\bot$), negations of arbitrary types ($\neg A$). Note that conjunction of two squash–stable types is also squash–stable. But a disjunction of two squash–stable types is not necessarily squash–stable since there is no way to figure out which of the disjuncts is true when we only know that at least one of them must be true.

4 Classical Reasoning on Squashed Types

Squash operator gives us an alternative way of formulating constructive recursive mathematics in a type theory. Let us consider a problem similar to the one we have considered in Section 2.

Suppose we have constructively proved that $A \to B$ and $(\neg A) \to B$. It means that there is an algorithm that produces an element of B when A is true, and another algorithm that produces an element of B when A is false. Classically we know that B is true, because in each case we can produce an element of B. But in the case when A is undecidable, B is not necessary known to be constructively true, since we do not have a uniform algorithm for producing an element of B. In intuitionistic mathematics we can only prove $\neg\neg B$ in this case.

Now suppose B is squash–stable. Then there exists an element b, such that $b \in B$ whenever B is non-empty. We know that B is not an empty type regardless of whether A is true. The constant algorithm that returns b does not depend on the truth of A. Therefore in constructive recursive mathematics we can conclude that B is *constructively* true, because we have an element b such that $b \in B$.

This reasoning establishes the following rule:

$$\frac{\Gamma;\, x : A \vdash B \quad \Gamma;\, y : \neg A \vdash B \quad \Gamma;\, v : \lceil B \rceil \vdash B \quad \Gamma \vdash A\,\mathrm{Type}}{\Gamma \vdash B} \tag{4.1}$$

This rule allows us to turn classical proofs of *squash–stable* statements into constructive ones. It is clear that rule (2.1) from Section 2 is a particular instance of (4.1). We will show that these two rules are in fact equivalent. We can also write a simpler version of the same rule:

$$\frac{\Gamma \vdash \neg\neg A \quad \Gamma \vdash A\,\mathrm{Type}}{\Gamma \vdash \lceil A \rceil} \tag{4.2}$$

This rules states that $\lceil A \rceil \Leftrightarrow \neg\neg A$ or informally, A is *a non-empty type* if and only if it is *not an empty type*.

Another way to formulate the same principle is to allow classical reasoning inside `squash` operator:

$$\frac{\Gamma \vdash A\,\mathrm{Type}}{\Gamma \vdash \lceil A \vee \neg A \rceil} \tag{4.3}$$

The following two theorems state that all the above rules are equivalent and that they imply Markov's principle. This shows that we can formulate Markov's principle in a very simple language – we only need propositional language with the modal operator "`squash`".

Theorem 4.1. *The rules* (2.1), (4.1), (4.2) *and* (4.3) *are equivalent.*

Proof. (4.1) $\Rightarrow$ (2.1). Take $B = (t \in T)$. We know that $t \in T$ is squash stable, therefore we can apply rule (4.1) to derive (2.1).

(2.1) ⇒ (4.2).

$$\dfrac{\dfrac{}{\Gamma;\, x : A \vdash A}}{\dfrac{\Gamma;\, x : A \vdash \bullet \in [A]}{\dfrac{\Gamma \vdash \bullet \in [A]}{\Gamma \vdash [A]}(E_{\in})}} \quad \dfrac{\dfrac{\Gamma \vdash \neg\neg A \quad \dfrac{}{y : \neg A;\, z : \neg\neg A \vdash \bot}}{\Gamma;\, y : \neg A \vdash \bot}(Cut)}{\Gamma;\, y : \neg A \vdash \bullet \in [A]}(E_{\bot}, Cut) \qquad \Gamma \vdash A\,\mathrm{Type}}(2.1)$$

(4.2) ⇒ (4.3).

It is easy to establish that $\neg\neg(A \vee \neg A)$ is an intuitionistic tautology. Therefore we have the following derivation:

$$\dfrac{\dfrac{\Gamma \vdash A\,\mathrm{Type}}{\Gamma \vdash \neg\neg(A \vee \neg A)} \quad \Gamma \vdash A\,\mathrm{Type}}{\Gamma \vdash [A \vee \neg A]}(4.2)$$

(4.3) ⇒ (4.1).

$$\dfrac{\dfrac{\Gamma \vdash A\,\mathrm{Type}}{\Gamma \vdash [A \vee \neg A]}(4.3) \qquad \dfrac{\dfrac{\dfrac{\Gamma;\, x : A \vdash B \quad \Gamma;\, y : \neg A \vdash B}{\Gamma;\, z : A \vee \neg A \vdash B}(E_{\vee})}{\Gamma;\, z : A \vee \neg A \vdash [B]}(I_{sq})}{\Gamma;\, v : [A \vee \neg A] \vdash [B]}(E_{sq}) \qquad \Gamma;\, [B] \vdash B}{\Gamma \vdash B}(Cut)$$

Theorem 4.2. *Using one of these rules one can prove Markov's principle in a type theory:*

$$\forall x : \mathbb{N}.(A(x) \vee \neg A(x)) \to \neg\neg\exists x : \mathbb{N}.A(x) \to \exists x : \mathbb{N}.A(x)$$

Proof. We need to show that the following sequent is derivable:

$$d : \forall x : \mathbb{N}.(A(x) \vee \neg A(x));\, v : \neg\neg\exists x : \mathbb{N}.A(x) \vdash \exists x : \mathbb{N}.A(x)$$

The proof is just a formalization of Markov's reasoning [13]. We are given the element d of the type $\forall x : \mathbb{N}.(A(x) \vee \neg A(x))$. That means that d is an algorithm that given a natural number x decides whether $A(x)$ holds. Now we construct a function f_d that would find an x such that $A(x)$. Let f_d be the following function

$$\mathtt{fix}\Big(f.\lambda x.decide\big(d(x); a.\,\langle x, a\rangle\,; b.f(x+1)\big)\Big)$$

that is a function such that

$$f_d(x) = \begin{cases} \langle x, a\rangle\,, & \text{if } A(x) \text{ is true and } a \in A(x) \\ f_d(x+1), & \text{if } A(x) \text{ is false} \end{cases}$$

If we are given natural n such that $A(n)$ is true, then we have a bound for computation of $f(n-k)$. One can prove that $\forall k \leq n. f_d(n-k) \in \exists x : \mathbb{N}. A(x)$ by induction on k. Therefore $f_d(0) \in \exists x : \mathbb{N}. A(x)$. Then we have the following derivation:

$$\dfrac{\dfrac{d : \forall x : \mathbb{N}.(A(x) \vee \neg A(x)); n : \mathbb{N}; u : A(n) \vdash f_d(0) \in \exists x : \mathbb{N}. A(x)}{d : \forall x : \mathbb{N}.(A(x) \vee \neg A(x)); \exists x : \mathbb{N}. A(x) \vdash f_d(0) \in \exists x : \mathbb{N}. A(x)}(E_\exists)}{d : \forall x : \mathbb{N}.(A(x) \vee \neg A(x)); \left[\exists x : \mathbb{N}. A(x)\right] \vdash f_d(0) \in \exists x : \mathbb{N}. A(x)}(E^2_{sq})$$

Now we are left to show that $\neg\neg\exists x : \mathbb{N}. A(x)$ implies $\left[\exists x : \mathbb{N}. A(x)\right]$. This is true because of the rule (4.2).

5 Semantical Consistency of Markov's Principle

Theorem 5.1. *The rule* (4.3) *(as well as its equivalents* – (2.1), (4.1) *and* (4.2)*) is valid in S. Allen's semantics [1,2] if we consider it in a classical meta-theory.*

Proof. We need to show that $\Gamma \vdash \left[A \vee \neg A\right]$ is true when A is a type. It is clear that $\left[A \vee \neg A\right]$ is a well–formed type. To prove that it is a true proposition we have to find a term in this type. Let us prove that $\bullet$ is the witness of $\left[A \vee \neg A\right]$. Since we are in a classical meta-theory, for every instantiation of variables introduced by Γ, A is either empty or not. If A is non-empty, then $A \vee \neg A$ is non-empty and so $\bullet \in \left[A \vee \neg A\right]$. If A is an empty type, then $\neg A$ is non-empty type and so, $\bullet$ is again in $\left[A \vee \neg A\right]$. Therefore $\bullet \in \left[A \vee \neg A\right]$ always holds.

Note that we can not prove that $\Gamma \vdash A \vee \neg A$ is valid even using a classical meta-theory, because there is no uniform witness for for $A \vee \neg A$.

Corollary 5.2. *The rule* (4.3) *(and its equivalents) is consistent with the NuPRL type theory containing the theory of partial functions [7].*

Note however that the rule of excluded middle $\Gamma \vdash A \vee \neg A$ is known to be inconsistent with the theory of [7]. In particular, in that theory we can prove that there exists an undecidable proposition. That is, for some P the following is provable:

$$\neg(\forall n : \mathbb{N}. P(n) \vee \neg P(n)) \tag{5.1}$$

Therefore even using rule (4.3) we can not prove that

$$\left[\forall n : \mathbb{N}. P(n) \vee \neg P(n)\right]$$

(which would contradict (5.1)). But we can prove a weaker statement

$$\forall n : \mathbb{N}. \left[P(n) \vee \neg P(n)\right]$$

that does not contradict (5.1).

6 Squash Operator as Modality

The `squash` operator can be considered as an intuitionistic modality. It turns out that it behaves like the lax modality (denoted by $\bigcirc$) in the Propositional Lax Logic (PLL) [8]. This logic was developed independently for several different purposes (see [8] for an overview).

PLL is the extension of intuitionistic logic with the following rules (in Gentzen style):

$$\frac{\Gamma \vdash A}{\Gamma \vdash \lceil A \rceil} \qquad \frac{\Gamma;\, A \vdash \lceil B \rceil}{\Gamma;\, \lceil A \rceil \vdash \lceil B \rceil}$$

PLL$^+$ is PLL+($\neg\lceil\bot\rceil$), i.e. PLL$^+$ has an additional rule:

$$\frac{\Gamma;\, A \vdash \bot}{\Gamma;\, \lceil A \rceil \vdash \bot}$$

PLL* is PLL$^+$+($\lceil A \rceil \leftrightarrow \neg\neg A$). We can write this axiom as the rule in Gentzen style:

$$\frac{\Gamma;\, \neg A \vdash \bot}{\Gamma \vdash \lceil A \rceil}$$

PLL$^+$ and PLL* are decidable and have natural Kripke models [8]. They meet cut elimination property. PLL$^+$ has the subformula property. PLL* also has the subformula property if we define $\neg A$ to be a subformula of $\lceil A \rceil$.

Theorem 6.1. *Let A be a propositional formula with the `squash` modality. Let Γ be a set of hypothesis of the form $x : (p\,\mathrm{Type})$ for all propositional variables p in A. Then*

(i) PLL$^+$ $\vdash A$ iff $\Gamma \vdash A$ is derivable in the type theory without 4.3

(ii) PLL* $\vdash A$ iff $\Gamma \vdash A$ is derivable in the type theory with 4.3

Proof. From left to right this theorem can be proved by induction on derivation in PLL$^+$ (PLL*). The right to left direction needs a semantical reasoning. We will only outline the proof for PLL*.

Let A' be the formula A where all subformulas of the form $\lceil B \rceil$ are replaced by $\neg\neg B$. If $\Gamma \vdash A$ is derivable in the type theory with 4.3 then this sequent is valid in the standard semantics in classical meta-theory (Theorem 5.1). Since $\lceil B \rceil \leftrightarrow \neg\neg B$ is true in this semantics then $\Gamma \vdash A'$ is also true. A' is a modal-free formula. Therefore A' is a valid intuitionistic formula. Hence A' is derivable in the intuitionistic propositional logic. Since we have $\lceil B \rceil \leftrightarrow \neg\neg B$ in PLL*, we can derive A in PLL*.

Remark 6.2. It is possible to consider the lax modality in PLL$^+$ as the diamond modality in the natural intuitionistic analog of S4 (in the style of [20]) with an additional rule $\Box A \leftrightarrow A$. Note that since in intuitionistic logics $\Box$ and $\Diamond$ are not interdefinable, $\Box A \leftrightarrow A$ does not imply $\Diamond A \leftrightarrow A$.

Example 6.3. We can prove some basic properties of `squash` in PLL$^+$:
$\lceil A\rceil \to \neg\neg A$
$\lceil A\rceil \leftrightarrow \lceil\lceil A\rceil\rceil$
$\lceil A \wedge B\rceil \leftrightarrow (\lceil A\rceil \wedge \lceil B\rceil)$
$\lceil A \to B\rceil \to (\lceil A\rceil \to \lceil B\rceil)$, but $(\lceil A\rceil \to \lceil B\rceil) \to \lceil A \to B\rceil$ is true only in PLL*
$\lceil \neg A\rceil \leftrightarrow \neg\lceil A\rceil$
$(\lceil A\rceil \vee \lceil B\rceil) \to \lceil A \vee B\rceil$, however $\lceil A \vee B\rceil \not\to \lceil A\rceil \vee \lceil B\rceil$ even in PLL*

We can express the notion of squash–stability in this logic as $sqst(A) = \lceil A\rceil \to A$.

Example 6.4. The following properties of squash-stability are derivable in PLL$^+$:
$sqst(\bot)$
$sqst(\neg A)$
$sqst(\lceil A\rceil)$
$sqst(A) \wedge sqst(B) \to sqst(A \wedge B)$, but $sqst(A) \vee sqst(B) \not\to sqst(A \vee B)$
$sqst(B) \to sqst(A \to B)$
In PLL* we can also prove
$sqst(A) \to (\neg\neg A \to A)$

7 Example: What Logic Do We Use in the Court?

It is clear that we do not use pure classical logic in the court. Let us consider the following cases.

Case 7.1. One night a jewelry shop was robbed. The same night a barber shop was robbed in another town. It was clear that these two robberies were committed by different people. There were two suspects, X and Y for both cases. It was determined that no one else could have committed these crimes. Is this information enough to sentence someone?

Let us formulate this problem in logic. Let

- J stand for "the jewelry shop was robbed",
- J_t stand for "t is guilty in the robbing of the jewelry shop",
- B stand for "the barber shop was robbed",
- B_t stand for "t is guilty in the robbing of the barber shop",
- G_t stand for "t is guilty".
  where t is X or Y.

We know the following:

1. $J \wedge B$ (the jewelry and barber shops were robbed)
2. $(\neg J_X \wedge \neg J_Y) \to \neg J$ (no one but X or Y robbed the jeweler)
3. $(\neg B_X \wedge \neg B_Y) \to \neg B$ (no one but X or Y robbed the barber)
4. $\neg(J_t \wedge B_t)$, where $t = X, Y$ (no one could rob both shops)
5. $J_t \to G_t$, where $t = X, Y$ (Criminal Law)
6. $B_t \to G_t$, where $t = X, Y$ (Criminal Law)

Using this assumptions one can *classically* prove G_X and G_Y. Here is the proof that X is guilty in the prosecutor's words.

"X is guilty! Only X and Y could rob the jeweler. If X robbed the jewelry shop, then he is guilty. Suppose for a moment that X is innocent in this robbery. Then Y robbed the jewelry shop. Therefore he could not have robbed the barber. Hence the barber was robbed by X, because no one else did this. Again X is guilty."

Should the jury accept this *classical* reasoning? Even if they were convinced by the prosecutor, they would be unable to bring in a verdict of "guilty" on X since such verdict must specify a particular crime of which X is found guilty. This is especially clear if the punishment for robbing jewelry shops is different from the punishment for robbing barber shops. Judge would not be able to "extract a constructive element" (i.e. determine the sentence) from the prosecutor's proof.

In terms of Section 3, G_t is not a squash–stable proposition.

Case 7.2. The jewelry shop was robbed again. Now there were three suspects, two twin brothers X and Y, and their friend Z. At the trial, it was determined that only X, Y and Z were able to commit the robbery. It was also determined that the shop was robbed by at least two robbers. One of the twins X or Y was seen in another town at the night of the crime, but unfortunately, since they are twins, there was no way to determine exactly who it was. Can jury find someone guilty?

It is easy to see that we can prove that Z is guilty using classical reasoning. But as we have learned from the previous case classical proofs are not sufficient in court. However this case differs from the previous one! We can prove (classically, of course) not only that Z is guilty, but also that Z is guilty of a particular crime. Therefore the judge has enough information to pass sentence on Z.

In the terms of Section 3 we can said that while the proposition "Z is guilty" is not squash–stable, the proposition "Z robbed a particular jewelry store on a certain date" is squash–stable (since we know how to pass a sentence when this proposition is true). This assumption together with the rule (4.1) allows us to bring Z to justice.

Here is a formal *constructive* proof (again J_t stands for "t robbed the jewelry shop" and G_t stands for "t is guilty"):

$$\dfrac{\dfrac{\dfrac{\dfrac{X \text{ or } Y \text{ has an alibi}}{J_X \vdash \neg J_Y} \quad \dfrac{\text{two of suspects are robbers}}{\neg J_Y \vdash J_X \wedge J_Z}}{J_X \vdash J_Z}(Cut) \quad \dfrac{\dfrac{\text{two of suspects are robbers}}{\neg J_X \vdash J_Y \wedge J_Z}}{\neg J_X \vdash J_Z} \quad \dfrac{J_Z \text{ is squash–stable}}{\lfloor J_Z \rfloor \vdash J_Z}}{\vdash J_Z}(4.1)}{\vdash G_Z}(\text{Criminal Law})$$

8 Related Work

Our notion of *squash–stability* is very similar to the squash–stability defined in [9, Section 14.2] and to the notion of *computational redundancy* [3, Section 3.4].

The `squash` operator we use is similar to the notion of proof irrelevance [11,17]. Each object in a proof irrelevance type is considered to be equal to any other object of this type. In [17] proof irrelevance was expressed in terms of a certain modality $\triangle$. If A is a type then $\triangle A$ is a type containing all elements of A considered equal. Using NuPRL notation we can write $\triangle A = A//True$, where $A//P$ is a quotient of a type A over relation P. We can prove the following chain:

$$A \to \triangle A \to \left[A\right] \to \neg\neg A$$

The main difference between $\left[A\right]$ and $\triangle A$ is that there is no uniform element for $\triangle A$. Therefore $\triangle A$ is not squash–stable and $\left[A\right]$ does not imply $\triangle A$. However it seems that modal logic of $\triangle$ modality is the same as logic of `squash` (i.e. PLL^{+}).

As far as we know Markov's principle in type theory was considered only by Erik Palmgren in [16]. He proved that a fragment of *intentional* Martin-Löf type theory is closed under Markov's rule:

$$\frac{\Gamma \vdash \neg\neg\exists x : A.P[x]}{\Gamma \vdash \exists x : A.P[x]}$$

where $P[x]$ is an equality type (i.e. $P[x]$ is $t[x] = s[x] \in T$. It is easy to see that this formulation of Markov's rule is not valid for type theories with undecidable equality and, in particular, in extensional type theories.

A The "Minimal" Set of Rules

The judgments of the type theory are the sequents of the following form

$$x_1 : A_1; x_2 : A_2[x_1]; \ldots; x_n : A_n[x_1, \ldots, x_{n-1}] \vdash C[x_1, \ldots, x_n]$$

This sequent is true if we have a uniform witness $t[x_1, \ldots, x_n]$ such that for every $x_1, \ldots, x_n$ if $x_i \in A_i[x_1, \ldots, x_{i-1}]$ then $t[x_1, \ldots, x_n]$ is a member of $C[x_1, \ldots, x_n]$.

The inference rules are presented below[4]. For every type constructor we have a well-formedness rule (W), an introduction rule (I), an elimination rule (E) and a membership introduction rule (M).

Syntax rules:

$$\frac{}{\Gamma;x{:}A;\Delta[x]\vdash A}(ax) \qquad \frac{\Gamma;\Delta\vdash A \quad \Gamma;x{:}A;\Delta\vdash C}{\Gamma;\Delta\vdash C}(Cut) \qquad \frac{\Gamma\vdash a\in A \quad \Gamma;x{:}A\vdash C[x]}{\Gamma\vdash C[a]}(Let)$$

Membership:

$$\frac{\Gamma\vdash t\in A}{\Gamma\vdash (t\in A)\ \mathrm{Type}}(W_{\in}) \qquad \frac{\Gamma\vdash t\in A}{\Gamma\vdash \bullet\in(t\in A)}(M_{\in}) \qquad \frac{\Gamma\vdash A\ \mathrm{Type}}{\Gamma\vdash \bullet\in(A\ \mathrm{Type})}(M_{\mathrm{Type}})$$

$$\frac{}{\Gamma;x{:}A;\Delta[x]\vdash x\in A}(I_{\in}) \qquad \frac{\Gamma\vdash t\in A}{\Gamma\vdash A}(E_{\in})$$

[4] Some of this rules are redundant. For example most of introduction rules are derivable from their membership introduction counterparts. The (Let) rule is derivable from the (Cut) rule using function type.

Disjunction:

$$\frac{\Gamma\vdash A\,\mathrm{Type}\quad \Gamma\vdash B\,\mathrm{Type}}{\Gamma\vdash A\vee B\,\mathrm{Type}}(W_\vee)\qquad \frac{\Gamma\vdash a\in A}{\Gamma\vdash \mathtt{inl}\,a\in A\vee B}(M^1_\vee)\qquad \frac{\Gamma\vdash b\in B}{\Gamma\vdash \mathtt{inr}\,b\in A\vee B}(M^2_\vee)$$

$$\frac{\Gamma\vdash A}{\Gamma\vdash A\vee B}(I^1_\vee)\qquad \frac{\Gamma\vdash B}{\Gamma\vdash A_1\vee A_2}(I^2_\vee)\qquad \frac{\Gamma;x{:}A;\Delta[\mathtt{inl}\,x]\vdash C[\mathtt{inl}\,x]\quad \Gamma;y{:}B;\Delta[\mathtt{inr}\,y]\vdash C[\mathtt{inr}\,y]}{\Gamma;z{:}A;\Delta[z]\vee B\vdash C[z]}(E_\vee)$$

Universal quantifier:

$$\frac{\Gamma;x{:}A\vdash B[x]\,\mathrm{Type}}{\Gamma\vdash\forall x{:}A.B[x]\,\mathrm{Type}}(W_\forall)\qquad \frac{\Gamma;x{:}A\vdash fx\in B[x]}{\Gamma\vdash f\in\forall x{:}A.B[x]}(M_\forall)$$

$$\frac{\Gamma;x{:}A\vdash B[x]}{\Gamma\vdash\forall x{:}A.B[x]}(I_\forall)\qquad \frac{\Gamma;f{:}\forall x{:}A.B[x];\Delta[f]\vdash a\in A}{\Gamma;f{:}\forall x{:}A.B[x];\Delta[f]\vdash fa\in B[a]}(E_\forall)$$

Existential quantifier:

$$\frac{\Gamma;x{:}A\vdash B[x]\,\mathrm{Type}}{\Gamma\vdash\exists x{:}A.B[x]\,\mathrm{Type}}(W_\exists)\qquad \frac{\Gamma;x{:}A\vdash a\in A\quad \Gamma;x{:}A\vdash b\in B[a]}{\Gamma\vdash\langle a,b\rangle\in\exists x{:}A.B[x]}(M_\exists)$$

$$\frac{\Gamma;x{:}A\vdash a\in A\quad \Gamma;x{:}A\vdash B[a]}{\Gamma\vdash\exists x{:}A.B[x]}(I_\exists)\qquad \frac{\Gamma;x{:}A;y{:}B[x];\Delta[\langle x,y\rangle]\vdash C[\langle x,y\rangle]}{\Gamma;z{:}\exists x{:}A.B[x];\Delta[z]\vdash C[z]}(E_\exists)$$

False:

$$\frac{}{\Gamma\vdash\bot\,\mathrm{Type}}(W_\bot)\qquad \frac{}{\Gamma;x{:}\bot;\Delta[x]\vdash C}(E_\bot)$$

Computation:

$\frac{\Gamma\vdash b\in T}{\Gamma\vdash a\in T}$ where $a\mapsto b$ Usual reduction rules: $\lambda x.a[x]\ b\longrightarrow a[b]$, *etc*

Arithmetic:

Induction, *etc*

We assume the following definitions:

$$A\to B=\forall x:A.B\qquad A\wedge B=\exists x:A.B,\text{ where } x\text{ is not free in } B$$
$$\neg A=A\to\bot\qquad \mathtt{fix}(f.p[f])=(\lambda x.p[xx])(\lambda x.p[xx])$$

We can establish the property 1.1 in this fragment by a straightforward induction on the derivation.

Acknowledgments

The authors would like to thank Bob Constable for his general guidance in this research and for extremely productive discussions of various constructive theories. We would also like to thank Sergei Artemov for his very useful comments on the early drafts of this paper. We would also like to thank the anonymous reviewers for their valuable comments and pointers.

References

1. Stuart F. Allen. *A Non-Type-Theoretic Semantics for Type-Theoretic Language.* PhD thesis, Cornell University, 1987.
2. Stuart F. Allen. A Non-type-theoretic Definition of Martin-Löf's Types. In *Proceedings of the Second Symposium on Logic in Computer Science*, pages 215–224. IEEE, June 1987.
3. Roland C. Backhouse, Paul Chisholm, Grant Malcolm, and Erik Saaman. Do-it-yourself type theory. *Formal Aspects of Computing*, 1:19–84, 1989.
4. M. J. Beeson. *Foundations of Constructive Mathematics.* Springer-Verlag, 1985.

5. Douglas Bridges and Fred Richman. *Varieties of Constructive Mathematics*. Cambridge University Press, Cambridge, 1988.
6. Robert L. Constable, Stuart F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, Douglas J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the NuPRL Development System*. Prentice-Hall, NJ, 1986.
7. Robert L. Constable and Karl Crary. Computational complexity and induction for partial computable functions in type theory. In *Preprint*, 1998.
8. Matt Fairtlough and Michael Mendler. Propositional lax logic. *Information and Computation*, 137(1):1–33, 1997.
9. Jason J. Hickey. *The MetaPRL Logical Programming Environment*. PhD thesis, Cornell University, January 2001.
10. Jason J. Hickey, Aleksey Nogin, et al. MetaPRL home page. `http://metaprl.org/`.
11. Martin Hofmann. *Extensional concepts in intensional Type theory*. PhD thesis, University of Edinburgh, Laboratory for Foundations of Computer Science, 1995.
12. A.A. Markov. On the continuity of constructive function. *Uspekhi Matematicheskikh Nauk*, 9/3(61):226–230, 1954. In Russian.
13. A.A. Markov. On constructive mathematics. *Trudy Matematicheskogo Instituta imeni V.A. Steklova*, 67:8–14, 1962. In Russian. English Translation: A.M.S. Translations, series 2, vol.98, pp. 1-9. MR 27#3528.
14. Per Martin-Löf. Constructive mathematics and computer programming. In *Proceedings of the Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, Amsterdam, 1982. North Holland.
15. Aleksey Nogin. Quotient types – a modular approach. Technical report, Cornell University, 2001. To appear.
16. Erik Palmgren. The Friedman translation for Martin-Löf's type theory. *Mathematical Logic Quarterly*, 41:314–326, 1995.
17. Frank Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In *Proceedings of the 16th Annual Symposium on Logic in Computer Science*, Boston, Massachusetts, June 2001. LICS. To appear.
18. Simon Thompson. *Type Theory and Functional Programming*. Addison-Wesley, 1991.
19. A.S. Troelstra and D. van Dalen. *Constructivism in Mathematics, An Introduction, Vol. I,II*. North-Holland, Amsterdam, 1988.
20. Duminda Wijesekera. Constructive modal logics I. *Annals of Pure and Applied Logic*, 50:271–301, 1990.