Verification of Program Properties with Dependent Types

Gujgiczer Anna

Software Verification and Validation - Homework Presentation

Fall semester, 2020/21

Theory

Type Theory

Type theory is the academic study of type systems.

Intuitionistic type theory[1]

Intuitionistic type theory or constructive type theory is at the same time a programming language and logic.

Per Martin-Löf

- wanted to provide an alternative foundation of mathematics
- designed the type theory on the principles of mathematical constructivism

[1] P. Martin-Löf, G. Sambin, *Intuitionistic type theory*, (Bibliopolis Naples, 1984), vol. 9.

Propositions as types

Classical A proposition is something which is either true or false

$Prop = Bool$

- If P is true, then $P \vee Q$ is true.

Constructive A proposition is something for which there can be an evidence or not

$Prop = Set$

- If x is a proof for P, then $inl(x)$ is a proof for $P \vee Q$.
- An f function is evidence for $P \to Q$ if for every $x \in P$, $f(x) \in Q$. The type of the f function is $P \to Q$

Fun fact: $P \vee \neg P$ is not a tautology in the constructive way.

Dependent Type Theory

In dependent type theory we can parameterise types over values as well as other types.

Example

$A : Set$	A is a type
$\mathbb{N} : Set$	$\mathbb{N}$ is a type
$Vect : Set \rightarrow \mathbb{N} \rightarrow Set$	

$Vect(A, n)$ is an n length list with values of type A.

Dependent Type Theory

Curry-Howard Equivalence[2]

For every proposition we can associate a type.

Type	Martin-Löf's dependent type theory
Proposition	First-order logic

[2] W. A. Howard, *To HB Curry: essays on combinatory logic, lambda calculus and formalism* **44**, 479–490 (1980).

Dependent Type Theory

Rules for constructing types and proofs

Example

Formation rule for $\vee$: If $A : Set$ and $B : Set$, then $A \vee B : Set$

Introduction rule for $\vee$: If $x : A$, then $inl(x) : A \vee B$,
if $y : B$, then $inr(y) : A \vee B$.

Formation rule for $\rightarrow$: If $A : Set$ and $B : Set$, then $A \rightarrow B : Set$

Introduction rule for $\rightarrow$: If from a context where $x : A$, we can
derive $y : B$, then in an empty context $\lambda x.y : A \rightarrow B$

It is like a computer program.

Interactive proof assistants

Proving a proposition using rules, interactively.
Artefact: formal proof.
Some examples: Agda, Isabelle, Ivor, Coq

The formal proof is the program. Usage in V&V:

- Proving a specification with a theorem prover
- Giving a precise type of the program

Verification and Validation

Avoiding runtime errors

With dependent type theory we can formulate stronger syntax rules.

Example (Adding vectors)

Usually	$[a, b] + [c, d, e]$ causes runtime error
Dependent Types	$[a, b]$ and $[c, d, e]$ have different types $Vec(A, 2)$ and $Vec(A, 3)$, therefore '$+$' is not well typed.

Hidden implementation details

Type theory is designed in a way, that we can not talk about the details of how things are actually implemented

> *In homotopy type theory we even have the Univalence Principle: $(=)\cong(\cong)$. If two things behaves the same, they are the same.*

Practice

Practice

Constructing Correct Circuits: Verification of Functional Aspects of Hardware Specifications with Dependent Types[3]

- defined two number representations
- defined two additions
  - binary number
  - natural number
- proved properties for natural number additions
- proved the two additions are equivalent
- deducing binary addition is correct

[3] E. Brady *et al.*, *Trends in Functional Programming* **8**, 159–176 (2007).

Practice

Formal Certification of a Compiler Back-end[4]

Used a proof assistant to program a compiler from Cminor (a C-like imperative language) to PowerPC assembly code.

[4] X. Leroy, presented at the Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 42–54.

Summary

- Type theory serves as alternative to set theory as a foundation of mathematics
- It is like a programming language (a program is a constructive proof)
- Significant theoretical interest
- Limited practical examples
- Usefulness in computer science
  - type checking instead of runtime error
  - proving specifications
  - hiding implementation details

Questions

References

- W. A. Howard, *To HB Curry: essays on combinatory logic, lambda calculus and formalism* **44**, 479–490 (1980).
- P. Martin-Löf, G. Sambin, *Intuitionistic type theory*, (Bibliopolis Naples, 1984), vol. 9.
- X. Leroy, presented at the Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 42–54.
- E. Brady, J. McKinna, K. Hammond, *Trends in Functional Programming* **8**, 159–176 (2007).