Exploring a Type Theoretic Library for Python Environment

Ryan Brill : UC Berkeley
Tianyu Liu : NUS
Arshay Sheth : NUS
Yutong Zou : NUS

August 7, 2019

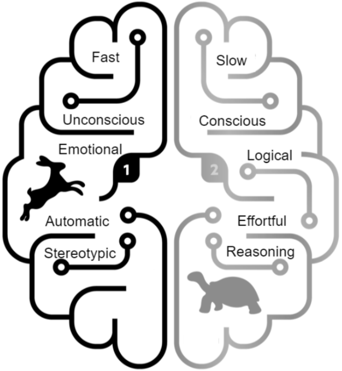
Overview

Artificial Intelligence Frontier

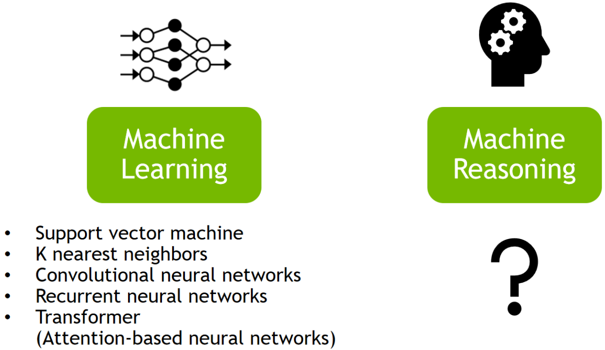
- Image Recognition
- General Game Playing: AlphaZero
- Autonomous Cars
- ......

Statistical Intuitive vs. Symbolic Reasoning Systems

System 1	**System 2**
drive a car on highways	drive a car in cities
come up with a good chess move (if you're a chess master)	point your attention towards the clowns at the circus
understands simple sentences	understands law clauses
correlation	causation
hard to explain	easy to explain



Machine Learning vs Machine Reasoning



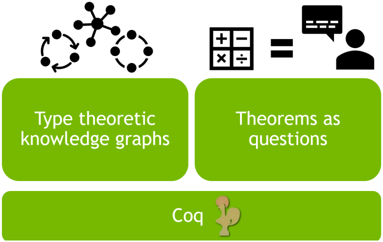
[1]

[1] Lai. Zhangsheng. "Exploring a Type theoretic library for python Environment". In: Presented by Nvidia, 2019.

Type Theoretic Machine Reasoning

TYPE THEORETIC APPROACH

Using type theory for knowledge graphs and QA

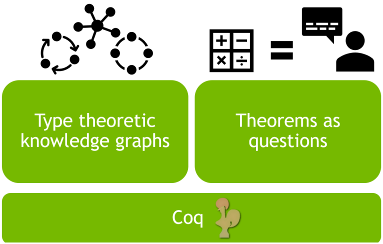


- Type theoretic machine reasoning is a reasoning system based on type theory through which a machine can generate conclusions or inferences from known knowledge

Type Theoretic Machine Reasoning

TYPE THEORETIC APPROACH

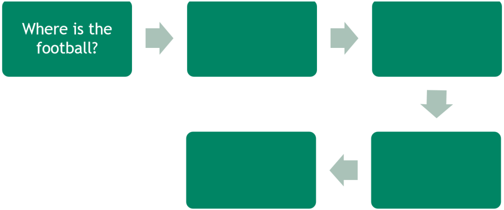
Using type theory for knowledge graphs and QA



- Type theoretic machine reasoning is a reasoning system based on type theory through which a machine can generate conclusions or inferences from known knowledge
- The key idea behind this is to regard a question posed to the computer as a theorem to be proven and a proof of the theorem as an answer to the question

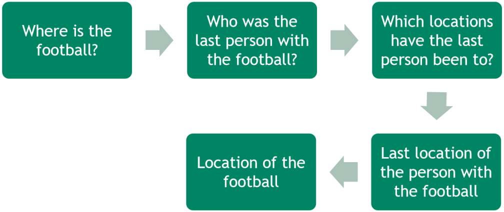
Naive Example of type theoretic machine reasoning

1. Mary moved to the bathroom.
2. Sandra journeyed to the bedroom.
3. Mary got the football there.
4. John went to the kitchen.
5. Mary went back to the kitchen.
6. Mary went back to the garden.
7. Where is the football? garden 3 6



Naive Example of type theoretic machine reasoning

1. Mary moved to the bathroom.
2. Sandra journeyed to the bedroom.
3. Mary got the football there.
4. John went to the kitchen.
5. Mary went back to the kitchen.
6. Mary went back to the garden.
7. Where is the football? garden 3 6



History of type theory

- In 1901, Russell discovered his famous paradox. Let $R = \{x | x \notin x\}$. Then $R \in R \implies R \notin R$ and $R \notin R \implies R \in R$!

History of type theory

- In 1901, Russell discovered his famous paradox. Let $R = \{x | x \notin x\}$. Then $R \in R \implies R \notin R$ and $R \notin R \implies R \in R$!
- It was realized that mathematics must be put on a strong logical and axiomatic foundation

History of type theory

- In 1901, Russell discovered his famous paradox. Let $R = \{x | x \notin x\}$. Then $R \in R \implies R \notin R$ and $R \notin R \implies R \in R$!
- It was realized that mathematics must be put on a strong logical and axiomatic foundation
- Two theories were put forward as competing candidates - axiomatic set theory & type theory

History of type theory

- In 1901, Russell discovered his famous paradox. Let $R = \{x | x \notin x\}$. Then $R \in R \implies R \notin R$ and $R \notin R \implies R \in R$!
- It was realized that mathematics must be put on a strong logical and axiomatic foundation
- Two theories were put forward as competing candidates - axiomatic set theory & type theory
- Axiomatic set theory gained wider acceptance at that time, but in recent years type theory and its enhanced version, homotopy type theory, have gained traction amongst mathematicians and computer scientists

Introducing Type Theory

- The expressions of type theory are *terms*, and all typable terms have a *type*

Introducing Type Theory

- The expressions of type theory are *terms*, and all typable terms have a *type*
- The expression "x has type T" is written x:T

Introducing Type Theory

- The expressions of type theory are *terms*, and all typable terms have a *type*
- The expression “x has type T” is written x:T
- For example, $0 : \mathbb{N}$ means 0 is a natural number

Introducing Type Theory

- The expressions of type theory are *terms*, and all typable terms have a *type*
- The expression "x has type T" is written x:T
- For example, $0 : \mathbb{N}$ means 0 is a natural number
- There are types for functions, types for proofs, and types for the types themselves

Introducing Type Theory

- The expressions of type theory are *terms*, and all typable terms have a *type*
- The expression "x has type T" is written x:T
- For example, $0 : \mathbb{N}$ means 0 is a natural number
- There are types for functions, types for proofs, and types for the types themselves
- Any object handled in the formalism must belong to a type. For instance, universal quantification is relative to a type and takes the form "for all x of type T, P "

Set Theory vs Type Theory

Set theory	**Type theory**
A set x either belongs or does not belong to a set y	A term x either has type y or does not have type y
Built on top of first order and second order logic	Propositional and predicate logic can be encoded in type theory
Elements can belong to multiple sets	Terms can only belong to one type

Set Theory vs Type Theory

Set theory	**Type theory**
A set x either belongs or does not belong to a set y	A term x either has type y or does not have type y
Built on top of first order and second order logic	Propositional and predicate logic can be encoded in type theory
Elements can belong to multiple sets	Terms can only belong to one type

Main advantage of Type Theory: It is easier for machines to check type-theoretic proofs

Curry-Howard Isomorphism

- Central theme: two readings of typing judgments:

$$\alpha : A \to A$$

- α is a term (program, expression) of the function type $A \to A$
- α is a proof (derivation) of the proposition $A \implies A$
- Note that implications are represented as functions types

Theorem Proving : Coq

- Coq is a proof assistant based on type theory

Theorem Proving : Coq

- Coq is a proof assistant based on type theory
- *Interactive* theorem prover : Incremental construction of proof
- Write commands known as *tactics* which break the theorem down into simpler subproblems

Theorem Proving : Coq

- Coq is a proof assistant based on type theory
- *Interactive* theorem prover : Incremental construction of proof
- Write commands known as *tactics* which break the theorem down into simpler subproblems
- Curry-Howard correspondence: Propositions as types, proofs as terms

Simple example of theorem proving in Coq

```
Variables A B : Prop.

Theorem thm: A -> ((A -> B) -> B).
```

```
1 subgoal

______________________________________
A -> (A -> B) -> B
```

Simple example of theorem proving in Coq

```
Variables A B : Prop.

Theorem thm: A -> ((A -> B) -> B).
Proof.
intro.
```

```
1 subgoal
H : A
______________________________________
(A -> B) -> B
```

Simple example of theorem proving in Coq

```
Variables A B : Prop.

Theorem thm: A -> ((A -> B) -> B).
Proof.
intro.
intro.
```

```
1 subgoal
H : A
H0 : A -> B
______________________________
B
```

Simple example of theorem proving in Coq

```
Variables A B : Prop.

Theorem thm: A -> ((A -> B) -> B).
Proof.
intro.
intro.
apply H0 in H as H1.
```

```
1 subgoal
H : A
H1 : B
H0 : A -> B
______________________
B
```

Simple example of theorem proving in Coq

```
Variables A B : Prop.

Theorem thm: A -> ((A -> B) -> B).
Proof.
intro.
intro.
apply H0 in H as H1.
exact H1.
```

```
No more subgoals.
```

Main components of Coq

- The critical kernel: a **type-checker**. A theorem is accepted only if its statement is the type of its proof term.

[2] Pierre Castéran. "The Coq proof assistant: principles, examples and main applications". In: Presented at NII, Shonan Meetings 100th Commemorative Symposium, Tokyo, June 22, 2018. Tokyo, 2018.

Main components of Coq

- The critical kernel: a **type-checker**. A theorem is accepted only if its statement is the type of its proof term.
- A programmable **tactic engine**, which helps the user to build proofs of statements semi-automatically.

[2] Castéran, "The Coq proof assistant: principles, examples and main applications".

Main components of Coq

- The critical kernel: a **type-checker**. A theorem is accepted only if its statement is the type of its proof term.
- A programmable **tactic engine**, which helps the user to build proofs of statements semi-automatically.
- A standard **library** containing definitions and theorems about mathematical structures and data types.[2]

[2] Castéran, “The Coq proof assistant: principles, examples and main applications”.

The roles played by inductive & record types

- During its implementation of machine reasoning in Coq, Nvidia found that **inductive types** and **record types** played important roles

The roles played by inductive & record types

- During its implementation of machine reasoning in Coq, Nvidia found that **inductive types** and **record types** played important roles
- Inductive types were useful to encode basic information (persons, locations, objects)

The roles played by inductive & record types

- During its implementation of machine reasoning in Coq, Nvidia found that **inductive types** and **record types** played important roles
- Inductive types were useful to encode basic information (persons, locations, objects)
- Record types were useful to collect all the subquestions needed to answer the main question

The roles played by inductive & record types

- During its implementation of machine reasoning in Coq, Nvidia found that **inductive types** and **record types** played important roles
- Inductive types were useful to encode basic information (persons, locations, objects)
- Record types were useful to collect all the subquestions needed to answer the main question
- Thus, our task was to implement inductive and record types into Python

Inductive Types

- An arbitrary expression $a : A$ simply tells us that term a is of type A

Inductive Types

- An arbitrary expression $a : A$ simply tells us that term a is of type A
- This expression gives us no information on the nature, number or properties of such terms

Inductive Types

- An arbitrary expression $a : A$ simply tells us that term a is of type A
- This expression gives us no information on the nature, number or properties of such terms
- This motivates the notion of an **inductive type**: an inductive type is a type equipped with rules (formally called **constructors**) that explain how the terms of a type are built

Inductive Types

- An arbitrary expression $a : A$ simply tells us that term a is of type A
- This expression gives us no information on the nature, number or properties of such terms
- This motivates the notion of an **inductive type**: an inductive type is a type equipped with rules (formally called **constructors**) that explain how the terms of a type are built

Quintessential example of inductive types is $\mathbb{N}$; we have two constructors $0 : \mathbb{N}, \quad S : \mathbb{N} \to \mathbb{N}$.

Record types

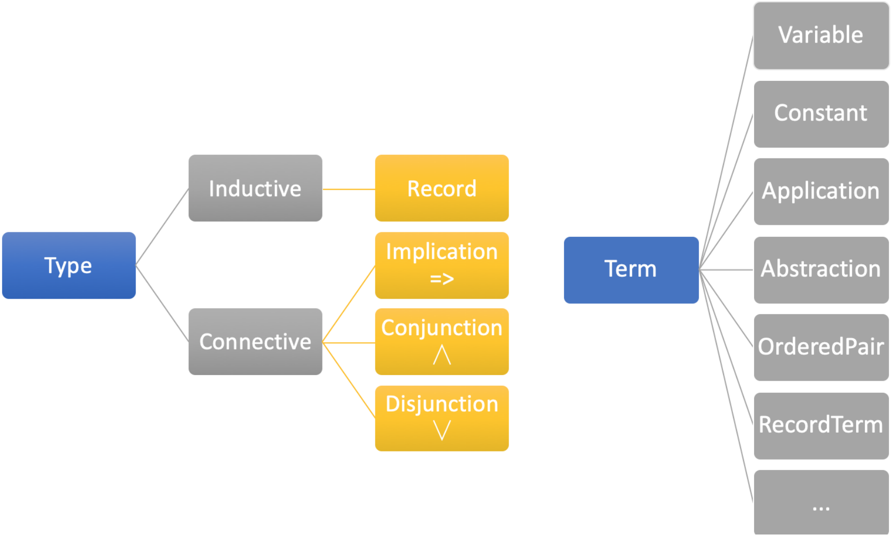
- Record type is an inductive type with only one constructor that is comprised of fields of different types

Record types

- Record type is an inductive type with only one constructor that is comprised of fields of different types

Quintessential example of record types is $\mathbb{Q}$, which has these fields: numerator, denominator, sign, bottom $\neq 0$, and irreducibility

Our Logical Framework



Implementing inductive types in Coq & Python

$\mathbb{N}$ in Coq	$\mathbb{N}$ in Python

`Inductive nat :=`
`  O : nat`
`| S : nat -> nat.`
	`nat = Inductive("nat")`
	`O = Const("O")`
	`S = Const("S")`
	`InductiveTypeIntro(O, nat)`
	`InductiveTypeIntro(S, Implication(nat, nat))`

Type checking ℕ in Python

```
one = Application(S, O)
two = Application(S, Application(S, O))
print(type_check(two))
>>> nat
```

Implementing record types in Coq & Python

$\mathbb{Q}$ in Coq	$\mathbb{Q}$ in Python

`Record rat := mkRat {`
`    sign: bool;`
`    top: nat;`
`    bottom: nat;`
`    cond: bottom <> 0;`
`    irred: forall x,y,z: nat,`
`        x*y=top /\ x*z=bottom`
`        -> x=1`
`}`
	`rat = Record("rat", {`
	`    top: nat,`
	`    bottom: nat,`
	`    sign: bool`
	`})`

Type Checking $\mathbb{Q}$ in Python

```
half = RecordTerm("rat", one, two, True)
print(type_check(half))
>>> rat
```

Holpy

- Holpy: a proof assistant in Python
- "Higher order logic in Python"

Example: theorem proving in Holpy

Prove $A \to ((A \to B) \to B)$

0. $A \to B \vdash A \to B$ by assume $A \to B$
1. $A \vdash A$ by assume A
2. $A, A \to B \vdash B$ by implication elimination on 0,1
3. $A \vdash (A \to B) \to B$ by implication introduction on 2
4. $\vdash A \to (A \to B) \to B$ by implication introduction on 3

```
th0 = Thm.assume(Term.mk_implies(A, B))
th1 = Thm.assume(A)
th2 = Thm.implies_elim(th0, th1)
th3 = Thm.implies_intr(Term.mk_implies(A, B), th2)
th4 = Thm.implies_intr(A, th3)
print(printer.print_thm(thy, th4, unicode=True))
```

```
⊢ A ⟶ (A ⟶ B) ⟶ B
```

Some features of Holpy

- Holpy is implemented in an object oriented framework

Some features of Holpy

- Holpy is implemented in an object oriented framework
- Theorem and proof objects in Holpy may be extracted as JSON files, which allows for easy integration into other systems

Some features of Holpy

- Holpy is implemented in an object oriented framework
- Theorem and proof objects in Holpy may be extracted as JSON files, which allows for easy integration into other systems
- Holpy makes extensive use of macros (abbreviations for potentially large proof terms) makes it scalable

Some features of Holpy

- Holpy is implemented in an object oriented framework
- Theorem and proof objects in Holpy may be extracted as JSON files, which allows for easy integration into other systems
- Holpy makes extensive use of macros (abbreviations for potentially large proof terms) makes it scalable
- Holpy has an API for proof automation

Some features of Coq

- Strong theorem prover: tactics, libraries

Some features of Coq

- Strong theorem prover: tactics, libraries
- Proof automation: formalization of the four colour theorem

Some features of Coq

- Strong theorem prover: tactics, libraries
- Proof automation: formalization of the four colour theorem
- Widely used in academia

Some features of Coq

- Strong theorem prover: tactics, libraries
- Proof automation: formalization of the four colour theorem
- Widely used in academia
- Versatile
  - can prove a wide variety of statements
  - theorems about most mathematical structures: natural numbers (induction proofs), groups, graphs (4 color theorem)
  - "theorems" on English sentences: what room is the football in?

Some features of Coq

- Strong theorem prover: tactics, libraries
- Proof automation: formalization of the four colour theorem
- Widely used in academia
- Versatile
  - can prove a wide variety of statements
  - theorems about most mathematical structures: natural numbers (induction proofs), groups, graphs (4 color theorem)
  - "theorems" on English sentences: what room is the football in?
- Software verification: used to reason on the correctness of software

Some features that Holpy lacks (as of now)

- Minimal implementation of inductive types and record types

Some features that Holpy lacks (as of now)

- Minimal implementation of inductive types and record types
- Interactive theorem proving

Some features that Holpy lacks (as of now)

- Minimal implementation of inductive types and record types
- Interactive theorem proving
- Predicate logic is underdeveloped (minimal proof support for Forall,Exists)

Some features that Holpy lacks (as of now)

- Minimal implementation of inductive types and record types
- Interactive theorem proving
- Predicate logic is underdeveloped (minimal proof support for Forall,Exists)
- No developer guide

Key differences between Coq & Holpy

Coq	Holpy
Uses backward chaining logic i.e. a top-down approach	Uses forward chaining logic i.e. a bottom-up approach
Based on a functional programming language	Based on an OOP language

Recap

- Type theoretic machine reasoning is a paradigm that helps the computer to make simple logical deductions by regarding questions as theorems and answers as proofs

Recap

- Type theoretic machine reasoning is a paradigm that helps the computer to make simple logical deductions by regarding questions as theorems and answers as proofs
- Coq is a proof assistant that Nvidia used to implement type theoretic machine reasoning; in this process they found that inductive and record types were especially important

Recap

- Type theoretic machine reasoning is a paradigm that helps the computer to make simple logical deductions by regarding questions as theorems and answers as proofs
- Coq is a proof assistant that Nvidia used to implement type theoretic machine reasoning; in this process they found that inductive and record types were especially important
- We built a small prototype library in Python which had some functionalities of inductive and record types

Recap

- Type theoretic machine reasoning is a paradigm that helps the computer to make simple logical deductions by regarding questions as theorems and answers as proofs
- Coq is a proof assistant that Nvidia used to implement type theoretic machine reasoning; in this process they found that inductive and record types were especially important
- We built a small prototype library in Python which had some functionalities of inductive and record types
- Holpy is an ongoing project which has a similar goal as that of our library but is implemented on a much larger scale

Acknowledgements

Special Thanks To...

- Zhangsheng Lai
- Ziyuan Gao
- Nvidia
- IMS at NUS
- IPAM at UCLA
- NSF grant DMS-1440415