Search for Program Structure

Gabriel Scherer

Northeastern University, Boston

SNAPL 2017
May 8, 2017

Perl vs. Python

Perl vs. Python

TIMTOWTDI

vs.

TIOOWTDI

Perl vs. Python

"There Is More Than One Way To Do It"

vs.

"There Is Only One Way To Do It"

Warmup

Arithmetic expressions over one variable x: meaning in $\mathbb{N} \to \mathbb{N}$

$$a, b \quad ::= \quad n \in \mathbb{N} \mid x \mid a + b \mid a \times b$$

Warmup

Arithmetic expressions over one variable x: meaning in $\mathbb{N} \to \mathbb{N}$

$$a, b \quad ::= \quad n \in \mathbb{N} \mid x \mid a + b \mid a \times b$$

This representation has redundancies: $2 + 2$ and 4, same meaning.
TIMTOWTDI

Warmup

Arithmetic expressions over one variable x: meaning in $\mathbb{N} \to \mathbb{N}$

$$a, b \quad ::= \quad n \in \mathbb{N} \mid x \mid a + b \mid a \times b$$

This representation has redundancies: $2 + 2$ and 4, same meaning.
TIMTOWTDI

Polynomials:

$$\sum_{0 \leqslant k \leqslant d} c_k x^k$$

More **canonical** representation: $2 + 2$ and 4 both become $4x^0$.
TIOOWTDI

Warmup

Arithmetic expressions over one variable x: meaning in $\mathbb{N} \to \mathbb{N}$

$$a, b \quad ::= \quad n \in \mathbb{N} \mid x \mid a + b \mid a \times b$$

This representation has redundancies: $2 + 2$ and 4, same meaning.
TIMTOWTDI

Polynomials:

$$\sum_{0 \leqslant k \leqslant d} c_k x^k$$

More **canonical** representation: $2 + 2$ and 4 both become $4x^0$.
TIOOWTDI

Helps for application: does a asymptotically dominate b?

Warmup

Arithmetic expressions over one variable x: meaning in $\mathbb{N} \to \mathbb{N}$

$$a, b \quad ::= \quad n \in \mathbb{N} \mid x \mid a + b \mid a \times b$$

This representation has redundancies: $2 + 2$ and 4, same meaning.
TIMTOWTDI

Polynomials:

$$\sum_{0 \leqslant k \leqslant d} c_k x^k$$

More **canonical** representation: $2 + 2$ and 4 both become $4x^0$.
TIOOWTDI

Helps for application: does a asymptotically dominate b?
Less convenient to write: $P \times Q$.

Representation and structure

Representations are human-designed.

Good representations reveal the **structure** of formal objects.

Canonical representations (no redundancies at all)
precisely capture/expose this structure.

What about PL?

For programming languages, clear notion of **equivalence** given by contextual equivalence.

But **representations** are under-studied.

What is a canonical representation of the programs of your language?

Some applications:

- Equivalence algorithms.
- Program synthesis.

Logic

Logicians have studied **proof representations** for decades.

- Natural deduction

Logic

Logicians have studied **proof representations** for decades.

- Natural deduction
- Sequent calculus

Logic

Logicians have studied **proof representations** for decades.

- Natural deduction
- Sequent calculus
- Tableaux

Logic

Logicians have studied **proof representations** for decades.

- Natural deduction
- Sequent calculus
- Tableaux
- Matrices/connections

Logic

Logicians have studied **proof representations** for decades.

- Natural deduction
- Sequent calculus
- Tableaux
- Matrices/connections
- Expansion proofs

Logic

Logicians have studied **proof representations** for decades.

- Natural deduction
- Sequent calculus
- Tableaux
- Matrices/connections
- Expansion proofs
- Proof nets

Logic

Logicians have studied **proof representations** for decades.

- Natural deduction
- Sequent calculus
- Tableaux
- Matrices/connections
- Expansion proofs
- Proof nets
- Focusing

Logic

Logicians have studied **proof representations** for decades.

- Natural deduction
- Sequent calculus
- Tableaux
- Matrices/connections
- Expansion proofs
- Proof nets
- Focusing
- Multi-focusing

Logic

Logicians have studied **proof representations** for decades.

- Natural deduction
- Sequent calculus
- Tableaux
- Matrices/connections
- Expansion proofs
- Proof nets
- Focusing
- Multi-focusing

Logic

Logicians have studied **proof representations** for decades.

- Natural deduction
- Sequent calculus
- Tableaux
- Matrices/connections
- Expansion proofs
- Proof nets
- Focusing
- Multi-focusing

Eliminates redundancies, clarifies the structure of proof search, restricts the search space.

Contribution

A new Curry-Howard connection.

"The structure of **programs**
corresponds to
the structure of **proof search**."

Contribution

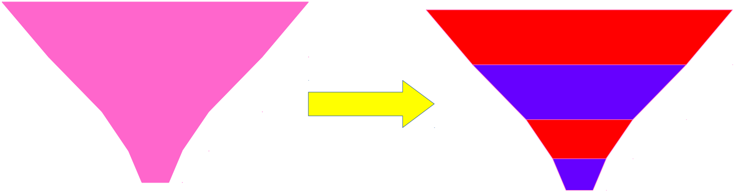
A new Curry-Howard connection.

"The structure of **programs**
corresponds to
the structure of **proof search**."

To find good program representations, go read logic papers.

Focusing

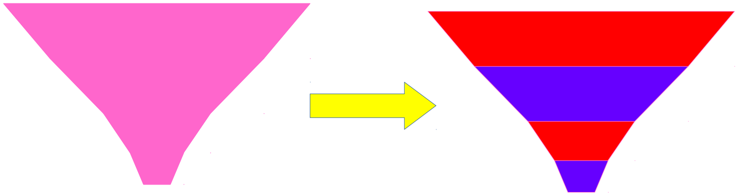
(Andreoli 1992)



$\Gamma \vdash A$ $\qquad\qquad$ $\Gamma \vdash_{\mathrm{foc}} A$

Focusing

(Andreoli 1992)



$\Gamma \vdash A$ $\qquad$ $\Gamma \vdash_{\mathrm{foc}} A$

Gives canonical representations for **impure** λ-calculi.
(Zeilberger 2009)

Nice sequent syntax in Munch-Maccagnoni (2013).

Saturation

(Scherer and Rémy 2015)
Combines **backward** and **forward** proof-search.

Gives canonical representation of the **pure** simply-typed λ-calculus.

Application: equivalence of programs with sums and the empty type (Scherer 2017).

Program synthesis

Types with a unique inhabitant (Scherer and Rémy 2015):
correct-by-construction synthesis.

Type-directed synthesis builds on focusing. Can it use saturation?
(Osera and Zdancewic 2015; Frankle, Osera, Walker, and Zdancewic 2016;
Polikarpova, Kuraj, and Solar-Lezama 2016)

Jean-Marc Andreoli (1992). "Logic Programming with Focusing Proofs in Linear Logic". **Journal of Logic and Computation** 2.3.

Noam Zeilberger (2009). "The Logical Basis of Evaluation Order and Pattern-Matching". PhD thesis.

Guillaume Munch-Maccagnoni (2013). "Syntax and Models of a non-Associative Composition of Programs and Proofs". PhD thesis.

Peter-Michael Osera and Steve Zdancewic (2015). "Type-and-Example-Directed Program Synthesis". **PLDI**.

Gabriel Scherer and Didier Rémy (2015). "Which simple types have a unique inhabitant?" **ICFP**.

Jonathan Frankle, Peter-Michael Osera, David Walker, and Steve Zdancewic (2016). "Example-directed synthesis: a type-theoretic interpretation". **POPL**.

Nadia Polikarpova, Ivan Kuraj, and Armando Solar-Lezama (2016). "Program synthesis from polymorphic refinement types". **PLDI**.

Gabriel Scherer (2017). "Deciding equivalence with sums and the empty type". **POPL**.