

ITP 2018

## HOL Light QE

Jacques Carette, William M. Farmer, and Patrick Laskowski

Department of Computing and Software  
McMaster University

11 July 2018



# Outline

- Syntax-based mathematical algorithms.
- Local and global reflection.
- $\text{CTT}_{\text{qe}}$ , Church's type theory with quotation and evaluation.
- HOL Light QE, an implementation of  $\text{CTT}_{\text{qe}}$ .

# A Warm-Up Example

- **Problem:** Are the following two functions equal?

$$\lambda x : \mathbb{R} . (x + 2) * (2 * x + 1) + 3 * x$$

$$\lambda x : \mathbb{R} . 2 * (x + 2)^2 - 6$$

# A Warm-Up Example

- **Problem:** Are the following two functions equal?

$$\lambda x : \mathbb{R} . (x + 2) * (2 * x + 1) + 3 * x$$

$$\lambda x : \mathbb{R} . 2 * (x + 2)^2 - 6$$

Notice the polynomials.

# A Warm-Up Example

- **Problem:** Are the following two functions equal?

$$\lambda x : \mathbb{R} . (x + 2) * (2 * x + 1) + 3 * x$$

$$\lambda x : \mathbb{R} . 2 * (x + 2)^2 - 6$$

Notice the polynomials.

- **Solution:** Normalize the polynomial

$$((x + 2) * (2 * x + 1) + 3 * x) - (2 * (x + 2)^2 - 6)$$

and see if it is the 0 polynomial.

# A Warm-Up Example

- **Problem:** Are the following two functions equal?

$$\lambda x : \mathbb{R} . (x + 2) * (2 * x + 1) + 3 * x$$

$$\lambda x : \mathbb{R} . 2 * (x + 2)^2 - 6$$

Notice the polynomials.

- **Solution:** Normalize the polynomial

$$((x + 2) * (2 * x + 1) + 3 * x) - (2 * (x + 2)^2 - 6)$$

and see if it is the 0 polynomial.

- “normalize” is an algorithm that:
  1. Manipulates mathematical expressions having the form of polynomials.
  2. Preserves the mathematical meaning of the expressions.

# Syntax-Based Mathematical Algorithms

- A **syntax-based mathematical algorithm (SBMA)** is an algorithm that manipulates the syntax of mathematical expressions in a mathematically meaningful way.

# Syntax-Based Mathematical Algorithms

- A **syntax-based mathematical algorithm (SBMA)** is an algorithm that manipulates the syntax of mathematical expressions in a mathematically meaningful way.
  - ▶ SBMAs are commonplace in mathematics.
  - ▶ “normalize” is an example.



# Syntax-Based Mathematical Algorithms

- A **syntax-based mathematical algorithm (SBMA)** is an algorithm that manipulates the syntax of mathematical expressions in a mathematically meaningful way.
  - ▶ SBMAs are commonplace in mathematics.
  - ▶ “normalize” is an example.
- A SBMA  $A$  has two fundamental properties:
  1. The **computational behavior** of  $A$  is the relationship between the input and output expressions of  $A$ .
  2. The **mathematical meaning** of  $A$  is the relationship between what the input and output expressions of  $A$  mean mathematically.

# Syntax-Based Mathematical Algorithms

- A **syntax-based mathematical algorithm (SBMA)** is an algorithm that manipulates the syntax of mathematical expressions in a mathematically meaningful way.
  - ▶ SBMAs are commonplace in mathematics.
  - ▶ “normalize” is an example.
- A SBMA  $A$  has two fundamental properties:
  1. The **computational behavior** of  $A$  is the relationship between the input and output expressions of  $A$ .
  2. The **mathematical meaning** of  $A$  is the relationship between what the input and output expressions of  $A$  mean mathematically.
- A **meaning formula** for  $A$  is a statement that expresses the mathematical meaning of  $A$ .

## Example: Meaning Formula for “normalize”

- $\text{normalize} : \text{Poly} \rightarrow \text{Poly}$

# Example: Meaning Formula for “normalize”

- $\text{normalize} : \text{Poly} \rightarrow \text{Poly}$

$\forall p, q : \text{Poly} .$

$(\forall x : \mathbb{R} . p = \text{normalize}(p)) \wedge$

$(\forall x : \mathbb{R} . p = q) \equiv \text{normalize}(p) = \text{normalize}(q)$

## Example: Meaning Formula for “normalize”

- $\text{normalize} : \text{Poly} \rightarrow \text{Poly}$

$\forall p, q : \text{Poly} .$

$(\forall x : \mathbb{R} . p = \text{normalize}(p)) \wedge$

$(\forall x : \mathbb{R} . p = q) \equiv \text{normalize}(p) = \text{normalize}(q)$

## Example: Meaning Formula for “normalize”

- $\text{normalize} : \text{Poly} \rightarrow \text{Poly}$

$\forall p, q : \text{Poly} .$

$$(\forall x : \mathbb{R} . p = \text{normalize}(p)) \wedge$$

$$(\forall x : \mathbb{R} . p = q) \equiv \text{normalize}(p) = \text{normalize}(q)$$

## Example: Meaning Formula for “normalize”

- $\text{normalize} : \text{Poly} \rightarrow \text{Poly}$

$\forall p, q : \text{Poly} .$

$$(\forall x : \mathbb{R} . \llbracket p \rrbracket = \llbracket \text{standardize}(p) \rrbracket) \wedge$$

$$(\forall x : \mathbb{R} . \llbracket p \rrbracket = \llbracket q \rrbracket) \equiv \text{standardize}(p) = \text{standardize}(q)$$

$\llbracket \cdot \rrbracket$  is the **evaluation operator**.

## Example: Meaning Formula for “normalize”

- $\text{normalize} : \text{Poly} \rightarrow \text{Poly}$

$\forall p, q : \text{Poly} .$

$$(\forall x : \mathbb{R} . \llbracket p \rrbracket = \llbracket \text{standardize}(p) \rrbracket) \wedge$$

$$(\forall x : \mathbb{R} . \llbracket p \rrbracket = \llbracket q \rrbracket) \equiv \text{standardize}(p) = \text{standardize}(q)$$

$\llbracket \cdot \rrbracket$  is the **evaluation operator**.

- The required result is obtained by instantiating  $p$  and  $q$  with

$$\ulcorner (x + 2) * (2 * x + 1) + 3 * x \urcorner$$

and

$$\ulcorner 2 * (x + 2)^2 - 6 \urcorner.$$

$\ulcorner \cdot \urcorner$  is the **quotation operator**.



# Quotation and Evaluation

- **Quotation** is used to gain access to the **syntax** of an expression.
  - ▶ The value of  $\ulcorner e \urcorner$  is a **syntactic value** that represents the syntactic structure of the expression  $e$ .
  - ▶ Note:  $\ulcorner 2 + 3 \urcorner \neq \ulcorner 5 \urcorner$ .

# Quotation and Evaluation

- **Quotation** is used to gain access to the **syntax** of an expression.
  - ▶ The value of  $\ulcorner e \urcorner$  is a **syntactic value** that represents the syntactic structure of the expression  $e$ .
  - ▶ Note:  $\ulcorner 2 + 3 \urcorner \neq \ulcorner 5 \urcorner$ .
- **Evaluation** is used to obtain the **semantics** of the expression represented by a syntactic value.
  - ▶ If the value of  $e'$  is a syntactic value  $e$ , then the value of  $\llbracket e' \rrbracket$  is the value of  $e$ .

# Quotation and Evaluation

- **Quotation** is used to gain access to the **syntax** of an expression.
  - ▶ The value of  $\ulcorner e \urcorner$  is a **syntactic value** that represents the syntactic structure of the expression  $e$ .
  - ▶ Note:  $\ulcorner 2 + 3 \urcorner \neq \ulcorner 5 \urcorner$ .
- **Evaluation** is used to obtain the **semantics** of the expression represented by a syntactic value.
  - ▶ If the value of  $e'$  is a syntactic value  $e$ , then the value of  $\llbracket e' \rrbracket$  is the value of  $e$ .
- The two operators are related by the **law of disquotation**:
$$\llbracket \ulcorner e \urcorner \rrbracket = e.$$

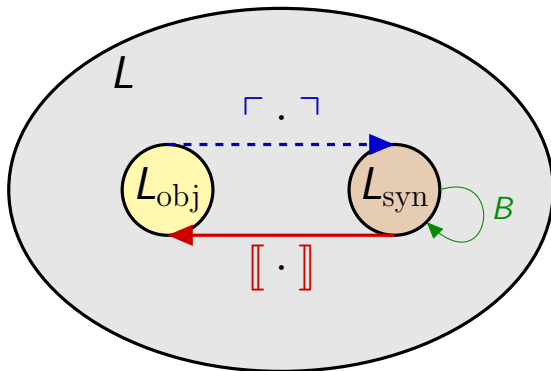
# Formalizing SBMAs

- To employ an SBMA  $A$  in a **proof assistant** (or other **formal environment**) we need to formalize  $A$  in the **logic** of the system.
- To formalize an SBMA  $A$  in a logic  $\text{Log}$  we need to be able to:
  1. Define in  $\text{Log}$  a function  $B$  on syntactic values representing  $A$ .
  2. State and prove in  $\text{Log}$  the meaning formula for  $B$  from the definition of  $B$ .
  3. Apply  $B$  to mathematical expressions in  $\text{Log}$  by instantiating the meaning formula for  $B$  and then simplifying.

# Standard Approach: Local Reflection

- Let  $A$  be an SBMA on expressions in a language  $L_{\text{obj}}$  of some logic  $\text{Log}$ .
- We build a **metareasoning infrastructure** in  $\text{Log}$  consisting of:
  1. An **inductive type**  $L_{\text{syn}}$  of syntactic values representing the expressions in  $L_{\text{obj}}$ .
  2. A **quotation operator**  $\ulcorner \cdot \urcorner$  mapping expressions in  $L_{\text{obj}}$  to syntactic values of  $L_{\text{syn}}$ .
  3. An **evaluation operator**  $\llbracket \cdot \rrbracket$  mapping syntactic values of  $L_{\text{syn}}$  to values of  $L_{\text{obj}}$ .
- We define a function  $B$  in  $\text{Log}$  from syntactic values representing inputs of  $A$  to syntactic values representing outputs of  $A$ .
- The infrastructure is **local** in the sense that  $L_{\text{obj}}$  is not the whole language  $L$  of  $\text{Log}$ .

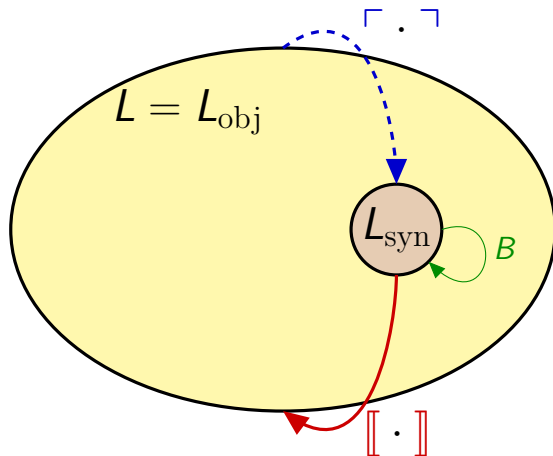
# Local Reflection



# An Alternate Approach: Global Reflection

- Local reflection does not scale up well:
  - ▶ Each collection of SBMAs requires a separate infrastructure.
  - ▶ Extending an SBMA to a new domain requires a new infrastructure.
- Global reflection employs a single infrastructure for all SBMAs:
  1. An **inductive type** representing the entire set of expressions.
  2. A **global quotation operator**  $\ulcorner \cdot \urcorner$ .
  3. A **global evaluation operator**  $\llbracket \cdot \rrbracket$ .
- Global reflection requires a logic with global quotation and evaluation operators.
- **It is an open problem whether global reflection is viable!**
- To test the viability of global reflection, we want to incorporate global quotation and evaluation into a traditional logic.

# Global Reflection





# Design Problems

Several challenging **design problems** face the logic engineer who seeks to incorporate global quotation and evaluation into a traditional logic:

1. **Evaluation Problem.** In a sufficiently strong theory, it is possible to express the liar paradox as an expression LIAR that equals  $\ulcorner \neg \llbracket \text{LIAR} \rrbracket \urcorner$  so that

$$\llbracket \text{LIAR} \rrbracket = \llbracket \ulcorner \neg \llbracket \text{LIAR} \rrbracket \urcorner \rrbracket = \neg \llbracket \text{LIAR} \rrbracket$$

by the law of disquotation.

2. **Variable Problem.** If  $c = \ulcorner x \urcorner$ , then  $x$  is free in  $\llbracket c \rrbracket$  since

$$\llbracket c \rrbracket = \llbracket \ulcorner x \urcorner \rrbracket = x$$

by the law of disquotation.

3. **Double Substitution Problem.**
4. **Constant Interpretation Problem.**

# CTT<sub>qe</sub>, a Version of Church's Type Theory

- Based on Andrews'  $\mathcal{Q}_0$ , CTT<sub>qe</sub> is a version of Church's type theory with a built-in global metareasoning infrastructure:
  1. An inductive type  $\epsilon$  of syntactic values that represent all the expressions of the logic.
  2. A partial global quotation operator  $\ulcorner \cdot \urcorner$ .
  3. A typed global evaluation operator  $\llbracket \cdot \rrbracket_\alpha$ .

# CTT<sub>qe</sub>, a Version of Church's Type Theory

- Based on Andrews'  $Q_0$ , CTT<sub>qe</sub> is a version of Church's type theory with a built-in global metareasoning infrastructure:
  1. An inductive type  $\epsilon$  of syntactic values that represent all the expressions of the logic.
  2. A partial global quotation operator  $\ulcorner \cdot \urcorner$ .
  3. A typed global evaluation operator  $\llbracket \cdot \rrbracket_\alpha$ .
- CTT<sub>qe</sub> is the third major attempt (after Chiron and  $Q_0^{uqe}$ ) to develop a traditional logic with global quotation and evaluation.

# CTT<sub>qe</sub>, a Version of Church's Type Theory

- Based on Andrews'  $Q_0$ , CTT<sub>qe</sub> is a version of Church's type theory with a built-in global metareasoning infrastructure:
  1. An inductive type  $\epsilon$  of syntactic values that represent all the expressions of the logic.
  2. A partial global quotation operator  $\ulcorner \cdot \urcorner$ .
  3. A typed global evaluation operator  $\llbracket \cdot \rrbracket_\alpha$ .
- CTT<sub>qe</sub> is the third major attempt (after Chiron and  $Q_0^{\text{uqe}}$ ) to develop a traditional logic with global quotation and evaluation.
- The proof system for CTT<sub>qe</sub> provides the ability to express, prove, and instantiate schemas and meaning formulas.

# CTT<sub>qe</sub>, a Version of Church's Type Theory

- Based on Andrews'  $Q_0$ , CTT<sub>qe</sub> is a version of Church's type theory with a built-in global metareasoning infrastructure:
  1. An inductive type  $\epsilon$  of syntactic values that represent all the expressions of the logic.
  2. A partial global quotation operator  $\ulcorner \cdot \urcorner$ .
  3. A typed global evaluation operator  $\llbracket \cdot \rrbracket_\alpha$ .
- CTT<sub>qe</sub> is the third major attempt (after Chiron and  $Q_0^{uqe}$ ) to develop a traditional logic with global quotation and evaluation.
- The proof system for CTT<sub>qe</sub> provides the ability to express, prove, and instantiate schemas and meaning formulas.
- We believe CTT<sub>qe</sub> is the first readily implementable version of simple type theory with global quotation and evaluation.

# CTT<sub>qe</sub>: Syntax

- A **expression of type  $\alpha$**  is inductively defined by the following formation rules:
  1. **Variable**: A variable  $\mathbf{x}_\alpha$  is an expression of type  $\alpha$ .
  2. **Constant**: A constant  $\mathbf{c}_\alpha$  is an expression of type  $\alpha$ .
  3. **Function application**:  $(\mathbf{F}_{\alpha \rightarrow \beta} \mathbf{A}_\alpha)$  is an expression of type  $\beta$ .
  4. **Function abstraction**:  $(\lambda \mathbf{x}_\alpha . \mathbf{B}_\beta)$  is an expression of type  $\alpha \rightarrow \beta$ .
  5. **Quotation**:  $\lceil \mathbf{A}_\alpha \rceil$  is an expression of type  $\epsilon$  if  $\mathbf{A}_\alpha$  is **eval-free**.
  6. **Evaluation**:  $\llbracket \mathbf{A}_\epsilon \rrbracket_{\mathbf{B}_\beta}$  is an expression of type  $\beta$ .
- **The restriction on quotation resolves the Evaluation Problem!**

## CTT<sub>qe</sub>: Substitution

- Due to the Variable Problem, CTT<sub>qe</sub> requires a semantics-dependent form of substitution.

## CTT<sub>qe</sub>: Substitution

- Due to the Variable Problem, CTT<sub>qe</sub> requires a semantics-dependent form of substitution.
- We could introduce a logical constant `sub` in CTT<sub>qe</sub> such that

$$\text{sub } \ulcorner \mathbf{A}_\alpha \urcorner \ulcorner \mathbf{x}_\alpha \urcorner \ulcorner \mathbf{B}_\beta \urcorner = \ulcorner \mathbf{C}_\beta \urcorner$$

holds iff “ $\mathbf{C}_\beta$  is the result of substituting  $\mathbf{A}_\alpha$  for each free occurrence of  $\mathbf{x}_\alpha$  in  $\mathbf{B}_\beta$  without any variable captures”

— but this works only for eval-free expressions.



## CTT<sub>qe</sub>: Substitution

- Due to the Variable Problem, CTT<sub>qe</sub> requires a semantics-dependent form of substitution.
- We could introduce a logical constant `sub` in CTT<sub>qe</sub> such that

$$\text{sub } \ulcorner \mathbf{A}_\alpha \urcorner \ulcorner \mathbf{x}_\alpha \urcorner \ulcorner \mathbf{B}_\beta \urcorner = \ulcorner \mathbf{C}_\beta \urcorner$$

holds iff “ $\mathbf{C}_\beta$  is the result of substituting  $\mathbf{A}_\alpha$  for each free occurrence of  $\mathbf{x}_\alpha$  in  $\mathbf{B}_\beta$  without any variable captures”

— but this works only for eval-free expressions.

- We implement substitution using Andrews' beta-reduction rules:
  - ▶ In the function abstraction rule we replace the syntactic notion of “a variable is free in an expression” with the more restrictive semantic notion of “a variable is effective in an expression”.
  - ▶ We add rules for quotations and evaluations.

# CTT<sub>qe</sub>: Substitution

- Due to the Variable Problem, CTT<sub>qe</sub> requires a semantics-dependent form of substitution.
- We could introduce a logical constant sub in CTT<sub>qe</sub> such that

$$\text{sub } \ulcorner \mathbf{A}_\alpha \urcorner \ulcorner \mathbf{x}_\alpha \urcorner \ulcorner \mathbf{B}_\beta \urcorner = \ulcorner \mathbf{C}_\beta \urcorner$$

holds iff “ $\mathbf{C}_\beta$  is the result of substituting  $\mathbf{A}_\alpha$  for each free occurrence of  $\mathbf{x}_\alpha$  in  $\mathbf{B}_\beta$  without any variable captures”

— but this works only for eval-free expressions.

- We implement substitution using Andrews’ beta-reduction rules:
  - ▶ In the function abstraction rule we replace the syntactic notion of “a variable is free in an expression” with the more restrictive semantic notion of “a variable is effective in an expression”.
  - ▶ We add rules for quotations and evaluations.
- This approach resolves the Variable Problem!

# CTT<sub>qe</sub>: New Beta-Reduction Rules

- Function Abstraction Rule

$$(\neg \text{IS-EFFECTIVE-IN}(\mathbf{y}_\beta, \mathbf{A}_\alpha) \vee \neg \text{IS-EFFECTIVE-IN}(\mathbf{x}_\alpha, \mathbf{B}_\gamma)) \supset \\ (\lambda \mathbf{x}_\alpha . \lambda \mathbf{y}_\beta . \mathbf{B}_\gamma) \mathbf{A}_\alpha = \lambda \mathbf{y}_\beta . ((\lambda \mathbf{x}_\alpha . \mathbf{B}_\gamma) \mathbf{A}_\alpha)$$

where  $\mathbf{x}_\alpha$  and  $\mathbf{y}_\beta$  are distinct.

- Quotation Rule

$$(\lambda \mathbf{x}_\alpha . \ulcorner \mathbf{B}_\beta \urcorner) \mathbf{A}_\alpha = \ulcorner \mathbf{B}_\beta \urcorner.$$

- Evaluation Rule

$$(\text{is-expr}_{\epsilon \rightarrow o}^\beta ((\lambda \mathbf{x}_\alpha . \mathbf{B}_\epsilon) \mathbf{A}_\alpha) \wedge \\ \neg (\text{is-free-in}_{\epsilon \rightarrow \epsilon \rightarrow o} \ulcorner \mathbf{x}_\alpha \urcorner ((\lambda \mathbf{x}_\alpha . \mathbf{B}_\epsilon) \mathbf{A}_\alpha))) \supset \\ (\lambda \mathbf{x}_\alpha . \llbracket \mathbf{B}_\epsilon \rrbracket_\beta) \mathbf{A}_\alpha = \llbracket (\lambda \mathbf{x}_\alpha . \mathbf{B}_\epsilon) \mathbf{A}_\alpha \rrbracket_\beta.$$

# HOL Light QE

- HOL Light QE is an implementation of  $\text{CTT}_{\text{qe}}$  obtained by modifying the HOL Light proof assistant.
  - ▶ HOL Light QE is, roughly speaking, HOL Light plus global quasiquotation and evaluation operators.
  - ▶ Developed by Patrick Laskowski under the supervision of Jacques Carette and William Farmer on a summer undergraduate research project.
  - ▶ It is available at  
<https://github.com/JacquesCarette/hol-light-qe>.
- Why we are using HOL Light:
  1. It is open source software.
  2. Its underlying logic is a version of Church's type theory.
  3. It provides support for defining inductive types.

# Implementation Stages

1. HOL term structure is extended.
  - ▶ Constructors are added to the OCaml type: **Quote**, **Hole**, and **Eval** for building **quotations**, **antiquotations**, and **evaluations**.
  - ▶ Inductive types of syntactic values are added for HOL Light QE **types** and **eval-free terms**.
2. HOL Light proof system is modified.
  - ▶ Rules for reasoning about quotations and evaluations.
  - ▶ INST rule is modified to work like substitution in  $\text{CTT}_{\text{qe}}$ .
3. Machinery for quotations and evaluations is created.
  - ▶ New functions, tactics, and theorems.
4. Examples are developed in the new system.
  - ▶ Law of Excluded Middle as a schema.
  - ▶ Induction schemas for Presburger and Peano arithmetic.

# Future Work

- Continue the development of HOL Light QE.
- Formalize simple SBMAs such as:
  - ▶ isEven on unary natural number numerals.
  - ▶ Tautology checker on boolean expressions.
  - ▶ Addition and multiplication on binary natural number numerals.
- Formalize symbolic differentiation algorithms.
- Formalize a graph of biform theories encoding natural number arithmetic.

# Conclusion

1. SBMAs can be formalized using reflection.
2. Unlike local reflection, global reflection requires only a single metareasoning infrastructure.
3.  $\text{CTT}_{\text{qe}}$  is a version of Church's type theory with a built-in global metareasoning infrastructure.
4. HOL Light QE demonstrates that  $\text{CTT}_{\text{qe}}$  is implementable.
5. Further development of HOL Light QE is needed to demonstrate the advantages global reflection has over local reflection.

# References

- [And02] P. B. Andrews, *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof, Second Edition*, Kluwer, 2002.
- [CarFar17] J. Carette and W. Farmer, “Formalizing Mathematical Knowledge as a Biform Theory Graph: A Case Study”, in: *Intelligent Computer Mathematics*, LNCS 10383:9–24, 2017.
- [Far13] W. M. Farmer, “The Formalization of Syntax-Based Mathematical Algorithms using Quotation and Evaluation”, in: *Intelligent Computer Mathematics*, LNCS 7961:35–50, 2013.
- [Far13a] W. M. Farmer, “Chiron: A Set Theory with Types, Undefinedness, Quotation, and Evaluation”, in: CoRR, abs/1305.6206 (154 pp.), 2013.
- [Far14] W. M. Farmer, “Simple Type Theory with Undefinedness, Quotation, and Evaluation”, in: CoRR, abs/1406.6706 (87 pp.), 2014.
- [Far17] W. M. Farmer, “Theory Morphisms in Church’s Type Theory with Quotation and Evaluation”, *Intelligent Computer Mathematics*, LNCS 10383:147–162, 2017.
- [Far18] W. M. Farmer, “Incorporating Quotation and Evaluation into Church’s Type Theory”, *Information and Computation*, 260:9–50, 2018.