A Formalization of Polytime Functions

Sylvain Heraud[1] David Nowak[2]

[1] INRIA Sophia Antipolis-Méditerranée, France

[2] ITRI, AIST, Japan

ITP 2011

Why formalizing polytime functions is important ?

- In cryptography:
  - An adversary with unlimited computational power could break most cryptographic schemes.
  - But it would not be realistic.
  - In security proofs, adversaries' running time is assumed to be polynomial in the security parameter (typically the size of the inputs).

Why formalizing polytime functions is important ?

- In cryptography:
  - An adversary with unlimited computational power could break most cryptographic schemes.
  - But it would not be realistic.
  - In security proofs, adversaries' running time is assumed to be polynomial in the security parameter (typically the size of the inputs).
- For NP-Completeness reductions:

  To check that a reduction between two NP-complete problems is polytime [C. Schürmann and J. Shah, 2003]

How formalizing polytime functions ?

- To compute running time, one could formalize a precise execution model, e.g., Turing machines.

How formalizing polytime functions ?

- To compute running time, one could formalize a precise execution model, e.g., Turing machines.
- It is simpler to use implicit computational complexity that relates programming languages with complexity classes.

Outline

Outline

Cobham's Class - 1964

- Cobham's class characterizes polytime functions.
- This is exactly the class of functions computable in polynomial time on a deterministic Turing machine.
- But this is not a fully syntactic characterization:
  A bound has to be proved for recursion.

Class $\mathcal{C}$ (Cobham)

$$
\begin{array}{rcll}
\mathcal{C} & ::= & O & \text{constant zero} \\
 & | & \Pi_i & \text{projection} \quad (i < n) \\
 & | & S_b & \text{successor} \\
 & | & \# & \text{smash } 2^{|x|.|y|} \\
 & | & \text{Comp } h\ \overline{g} & \text{composition} \\
 & | & \text{Rec } g\ h_0\ h_1\ j & \text{recursion}
\end{array}
$$

- Comp $h\ \overline{g}(\overline{x}) = h(\overline{g}(\overline{x}))$
- Rec $g\ h_0\ h_1\ j$ is equal to the function f such that:

$$
\begin{array}{lcll}
f(\epsilon, \overline{x}) & = & g(\overline{x}) & \\
f(yi, \overline{x}) & = & h_i(y, f(y, \overline{x}), \overline{x}) & \\
|f(y, \overline{x})| & \leq & |j(y, \overline{x})| & (\mathit{RecBounded})
\end{array}
$$

$\Rightarrow$ The bound j has to be defined and proved by hand!

Properties of $\mathcal{C}$

For each well-formed expression in $\mathcal{C}$, these is a polynomial that bounds the result.

$$|f(\overline{x})| \leq (\mathsf{Pol}_{\mathcal{C}}(f))(\overline{|x|})$$

Properties of $\mathcal{C}$

For each well-formed expression in $\mathcal{C}$, these is a polynomial that bounds the result.

$$|f(\overline{x})| \leq (\mathsf{Pol}_{\mathcal{C}}(f))(\overline{|x|})$$

defined as

$$\begin{array}{lcl}
\mathsf{Pol}_{\mathcal{C}}(O) & = & 0 \\
\mathsf{Pol}_{\mathcal{C}}(\Pi_i^n) & = & x_i \\
\mathsf{Pol}_{\mathcal{C}}(S_b) & = & x_0 + 1 \\
\mathsf{Pol}_{\mathcal{C}}(\#) & = & x_0.x_1 + 1 \\
\mathsf{Pol}_{\mathcal{C}}(\mathsf{Comp}^n\ h\ \overline{g}) & = & (\mathsf{Pol}_{\mathcal{C}}(h))(\overline{\mathsf{Pol}_{\mathcal{C}}(g)}) \\
\mathsf{Pol}_{\mathcal{C}}(\mathsf{Rec}\ g\ h_0\ h_1\ j) & = & \mathsf{Pol}_{\mathcal{C}}(j)
\end{array}$$

Outline

Bellantoni-Cook's Class - 1991

- This class is equivalent to the Cobham's one:
  It also characterizes polytime functions.
- But there is no bound to prove here.
- Instead, there are two kinds of variables: "normal" and "safe".

$$f(\underbrace{x_1, \ldots, x_n}_{\text{normal}}; \underbrace{a_1, \ldots, a_s}_{\text{safe}})$$

- It is not allowed to recur on safe variables.

Class $\mathcal{B}$

$$f(\underbrace{x_1, \ldots, x_n}_{\text{normal}}; \underbrace{a_1, \ldots, a_s}_{\text{safe}})$$

$\mathcal{B}$	$::=$	0	constant zero
	$\mid$	$\pi_i^{n,s}$	projection $(i < n + s)$
	$\mid$	s_b	successor
	$\mid$	pred	predecessor
	$\mid$	cond	conditional
	$\mid$	$\text{comp}^{n,s}\ h\ \overline{g_N}\ \overline{g_S}$	composition
	$\mid$	$\text{rec}\ g\ h_0\ h_1$	recursion

Class $\mathcal{B}$

$$f(\underbrace{x_1, \ldots, x_n}_{\text{normal}}; \underbrace{a_1, \ldots, a_s}_{\text{safe}})$$

$\mathcal{B}$	$::=$	0	constant zero	$(0,0)$
	$\mid$	$\pi_i^{n,s}$	projection	(n,s)
	$\mid$	s_b	successor	$(0,1)$
	$\mid$	pred	predecessor	$(0,1)$
	$\mid$	cond	conditional	$(0,4)$
	$\mid$	$\text{comp}^{n,s}\ h\ \overline{g_N}\ \overline{g_S}$	composition	(n,s)
	$\mid$	$\text{rec}\ g\ h_0\ h_1$	recursion	(n_h, s_g)

Here we infer two arities: the number of normal and safe arguments

Class $\mathcal{B}$ - Safe recurtion

It is not allowed to recur on safe variables.

Recursion : rec g h_0 h_1

$$\begin{array}{lcl} f(\epsilon, \vec{x}; \vec{a}) & := & g(\vec{x}; \vec{a}) \\ f(yi, \vec{x}; \vec{a}) & := & h_i(y, \vec{x}; f(y, \vec{x}; \vec{a}), \vec{a}) \end{array}$$

Note that:

- The recursive argument y is on normal position.
- The result $f(y, \vec{x}; \vec{a})$ of the recursive call is in safe position.

Class $\mathcal{B}$ - Safe recurtion

It is not allowed to recur on safe variables.

Recursion : rec $g\ h_0\ h_1$

$$\begin{array}{lcl} f(\epsilon, \vec{x}; \vec{a}) & := & g(\vec{x}; \vec{a}) \\ f(yi, \vec{x}; \vec{a}) & := & h_i(y, \vec{x}; f(y, \vec{x}; \vec{a}), \vec{a}) \end{array}$$

Note that:

- The recursive argument y is on normal position.
- The result $f(y, \vec{x}; \vec{a})$ of the recursive call is in safe position.

Composition : $\mathrm{comp}^{n,s}\ h\ \overline{g_N}\ \overline{g_S}$

$$f(\vec{x}; \vec{a}) := h(\vec{g}_n(\vec{x};); \vec{g}_s(\vec{x}; \vec{a}))$$

Example

$plus\ x\ y := \text{match } x \text{ with}$
$\mid 0 => y$
$\mid S\ x' => S(plus\ x'\ y)$

Example

$$
\begin{array}{l}
\mathit{plus}\ x\ y := \text{match } x \text{ with} \\
\mid 0 => y \\
\mid S\ x' => S(\mathit{plus}\ x'\ y)
\end{array}
$$

$$
\begin{array}{l}
\mathit{plus} := \text{rec} \\
\quad (\pi_0^{0,1}) \\
\quad (\text{comp}^{1,2}\ S_1\ \langle\rangle\ \langle \pi_1^{1,2} \rangle) \\
\quad (\text{comp}^{1,2}\ S_1\ \langle\rangle\ \langle \pi_1^{1,2} \rangle)
\end{array}
$$

Example

$plus\ x\ y := \text{match } x \text{ with}$
$\mid 0 => y$
$\mid S\ x' => S(plus\ x'\ y)$

$plus := \text{rec}$
$\quad (\pi_0^{0,1})$
$\quad (\text{comp}^{1,2}\ S_1\ \langle\rangle\ \langle\pi_1^{1,2}\rangle)$
$\quad (\text{comp}^{1,2}\ S_1\ \langle\rangle\ \langle\pi_1^{1,2}\rangle)$

$\mathcal{A}(plus) = (1, 1)$

Example

$plus\ x\ y := \text{match } x \text{ with}$
$\mid 0 => y$
$\mid S\ x' => S(plus\ x'\ y)$

$mult\ x\ y := \text{match } x \text{ with}$
$\mid 0 => y$
$\mid S\ x' => plus\ y\ (mult\ x'\ y)$

$plus := \text{rec}$
$\quad (\pi_0^{0,1})$
$\quad (\text{comp}^{1,2}\ S_1\ \langle\rangle\ \langle\pi_1^{1,2}\rangle)$
$\quad (\text{comp}^{1,2}\ S_1\ \langle\rangle\ \langle\pi_1^{1,2}\rangle)$

$\mathcal{A}(plus) = (1, 1)$

Example

$plus\ x\ y := \text{match } x \text{ with}$
$\mid 0 => y$
$\mid S\ x' => S(plus\ x'\ y)$

$plus := \text{rec}$
$\quad (\pi_0^{0,1})$
$\quad (\text{comp}^{1,2}\ S_1\ \langle\rangle\ \langle\pi_1^{1,2}\rangle)$
$\quad (\text{comp}^{1,2}\ S_1\ \langle\rangle\ \langle\pi_1^{1,2}\rangle)$

$\mathcal{A}(plus) = (1, 1)$

$mult\ x\ y := \text{match } x \text{ with}$
$\mid 0 => y$
$\mid S\ x' => plus\ y\ (mult\ x'\ y)$

$mult := \text{rec}$
$\quad (\text{comp}^{1,0}\ O\ \langle\rangle\ \langle\rangle)$
$\quad (\text{comp}^{1,2}\ plus\ \langle\pi_1^{2,0}\rangle\ \langle\pi_2^{2,1}\rangle)$
$\quad (\text{comp}^{1,2}\ plus\ \langle\pi_1^{2,0}\rangle\ \langle\pi_2^{2,1}\rangle)$

Example

$plus\ x\ y := \text{match } x \text{ with}$
$\mid 0 => y$
$\mid S\ x' => S(plus\ x'\ y)$

$plus := \text{rec}$
$\quad (\pi_0^{0,1})$
$\quad (\text{comp}^{1,2}\ S_1\ \langle\rangle\ \langle\pi_1^{1,2}\rangle)$
$\quad (\text{comp}^{1,2}\ S_1\ \langle\rangle\ \langle\pi_1^{1,2}\rangle)$

$\mathcal{A}(plus) = (1, 1)$

$mult\ x\ y := \text{match } x \text{ with}$
$\mid 0 => y$
$\mid S\ x' => plus\ y\ (mult\ x'\ y)$

$mult := \text{rec}$
$\quad (\text{comp}^{1,0}\ O\ \langle\rangle\ \langle\rangle)$
$\quad (\text{comp}^{1,2}\ plus\ \langle\pi_1^{2,0}\rangle\ \langle\pi_2^{2,1}\rangle)$
$\quad (\text{comp}^{1,2}\ plus\ \langle\pi_1^{2,0}\rangle\ \langle\pi_2^{2,1}\rangle)$

$\mathcal{A}(mult) = (2, 0)$

Properties of $\mathcal{B}$

Theorem (Polymax Bounding)

For all f in $\mathcal{B}$ with well-defined arities $\mathcal{A}(f)$, there exists a length-bounding monotone polynomial $\mathrm{Pol}_{\mathcal{B}}(f)$ such that, for all $\overline{x}$ and $\overline{y}$

$$|f(\overline{x};\overline{y})| \leq (\mathrm{Pol}_{\mathcal{B}}(f))(\overline{|x|}) + \max_i |y_i|$$

Outline

Bellantoni's thesis: $\mathcal{C} \leftrightarrow \mathcal{B}$

The two classes are equivalents if:

- we can translate every $\mathcal{C}$ into $\mathcal{B}$ $\quad (\mathcal{C} \subset \mathcal{B})$
- we can translate every $\mathcal{B}$ into $\mathcal{C}$ $\quad (\mathcal{B} \subset \mathcal{C})$

Bellantoni and Cook paper proof Bellantoni and Cook shows for each function in $\mathcal{B}$ (resp. $\mathcal{C}$) the existence of a function with the same semantics in $\mathcal{C}$ (resp. $\mathcal{B}$).

Coq proof We write two certified compilers $\mathcal{B} \to \mathcal{C}$ and $\mathcal{C} \to \mathcal{B}$

Compilation: $\mathcal{B} \to \mathcal{C}$

Difficulty : find the bound j

$$\text{rec } g\ h_0\ h_1 \Rightarrow \text{Rec } g'\ h'_0\ h'_1\ j$$

Compilation: $\mathcal{B} \to \mathcal{C}$

Difficulty : find the bound j

$$\text{rec } g\ h_0\ h_1 \Rightarrow \text{Rec } g'\ h'_0\ h'_1\ j$$

We need to encode polynomials into Cobham expressions

$$\forall P, \left|\text{Poly} \to \mathcal{C}(\overline{x})(P)\right| = P\left(\overline{|x|}\right)$$

Compilation: $\mathcal{B} \to \mathcal{C}$

Difficulty : find the bound j

$$\text{rec } g\ h_0\ h_1 \Rightarrow \text{Rec } g'\ h'_0\ h'_1\ j$$

We need to encode polynomials into Cobham expressions

$$\forall P, \left|\text{Poly}\to\mathcal{C}(\overline{x})(P)\right| = P\left(\overline{|x|}\right)$$

j is given by composition of $\text{Poly}\to\mathcal{C}$ and $\text{Pol}_{\mathcal{B}}$

$$j = \text{Poly}\to\mathcal{C}\left(\text{Pol}_{\mathcal{B}}(\text{rec } g\ h_0\ h_1)\ + x_n\ +\ \cdots\ +\ x_{n+s-1}\right)$$

Compilation: $\mathcal{C} \to \mathcal{B}$

Difficulty : recursion

$$\text{Rec } g' \ h'_0 \ h'_1 \ j \Rightarrow \text{rec } g \ h_0 \ h_1$$

Compilation: $\mathcal{C} \to \mathcal{B}$

Difficulty : recursion

$$\text{Rec } g' \ h'_0 \ h'_1 \ j \Rightarrow \text{rec } g \ h_0 \ h_1$$

Simulate the recursion with an artificial argument w

$$\forall f \text{ in } \mathcal{C}, f(\vec{x}) = \mathcal{C} \to \mathcal{B}(w; \vec{x})$$

if w is big enough

$$Pol_{\mathcal{C}\to\mathcal{B}}(f)(\overline{|x|}) \leq |w|$$

Summary of our implementation in Coq

- we have deep embedded the Cobham and Bellantoni-Cook's classes and formally proved their relations.

```
$ coqwc -e *.v
spec proof comments
2600 6321 1189 total
```

Summary of our implementation in Coq

- we have deep embedded the Cobham and Bellantoni-Cook's classes and formally proved their relations.

```
$ coqwc -e *.v
spec proof comments
2600 6321 1189 total
```

- Compared to the paper proof by Bellantoni and Cook, we have been careful in making our proof fully constructive.

  We obtain more precise bounding polynomials and efficient translations between the two characterizations.

Summary of our implementation in Coq

- we have deep embedded the Cobham and Bellantoni-Cook's classes and formally proved their relations.

```
$ coqwc -e *.v
spec proof comments
2600 6321 1189 total
```

- Compared to the paper proof by Bellantoni and Cook, we have been careful in making our proof fully constructive.

  We obtain more precise bounding polynomials and efficient translations between the two characterizations.

- Another difference is that we consider functions on bitstrings instead of functions on positive integers.

  This latter change is motivated by the application of our formalization in the context of formal security proofs in cryptography: we want to distinguish, for example, bitstrings 0 and 00.

Outline

CertiCrypt

- CertiCrypt is a Coq library for formalizing security proofs in cryptography by using a probabilistic imperative programming language.
- A command c is PPT if:
  - It terminates;
  -
    - if polynomials (p, q) bound the size and time of the input
    - then there exists $(F(p), q + G(q))$ that bounds size and time of the output.
- The language of expressions admits user-defined expressions:
  - The user must prove a bound on size.
  - The bound on time is axiomatized.

Our extension to CertiCrypt

- We add Bellantoni-Cook's expression to the language.
- The size bound is given by:

$$F(p) = \mathrm{Pol}_{\mathcal{B}}(e)(p)$$

It follows immediately from properties of Bellantoni-Cook's class.

Our extension to CertiCrypt

- We add Bellantoni-Cook's expression to the language.
- The size bound is given by:

$$F(p) = \mathrm{Pol}_{\mathcal{B}}(e)(p)$$

  It follows immediately from properties of Bellantoni-Cook's class.
- The time bound is given:

$$G(p) = \mathrm{Pol}_{\mathrm{time}}(e)(p)$$

  - A Bellantoni-Cook's expression is guaranteed to be executable in polynomial time.
  - But the precise polynomial depends on the particular implementation.
  - We consider the obvious implementation on a stack machine to derive Pol_{time}.

Conclusion

Contributions

- Formalization of Cobham and Bellantoni-Cook's Classes

  Adaptation and verification in Coq of the Chapter 3 of Bellantoni's PhD thesis

- Certified compilers between $\mathcal{C} \to \mathcal{B}$ and $\mathcal{B} \to \mathcal{C}$

  We had to fix a lot of details that were left to the reader in the paper proof.

- Usability in proof of security in cryptography

  By extending CertiCrypt

Conclusion

Contributions

- Formalization of Cobham and Bellantoni-Cook's Classes

  Adaptation and verification in Coq of the Chapter 3 of Bellantoni's PhD thesis

- Certified compilers between $\mathcal{C} \to \mathcal{B}$ and $\mathcal{B} \to \mathcal{C}$

  We had to fix a lot of details that were left to the reader in the paper proof.

- Usability in proof of security in cryptography

  By extending CertiCrypt

Future Work

- Library of polytime functions
- Formalizing other characterizations of polytime functions
- Dealing with probabilities inside the class

Library:

http://staff.aist.go.jp/david.nowak/polytime/

Questions ?

Well-formedness of $\mathcal{C}$

$$\begin{aligned} f(\epsilon, \vec{x}) &:= g(\vec{x}) \\ f(yi, \vec{x}) &:= h_i(y, f(y, \vec{x}), \vec{x}) \end{aligned}$$

h_i takes two more arguments than g.

Well-formedness of $\mathcal{C}$

$$\begin{aligned} f(\epsilon, \vec{x}) &:= g(\vec{x}) \\ f(yi, \vec{x}) &:= h_i(y, f(y, \vec{x}), \vec{x}) \end{aligned}$$

h_i takes two more arguments than g.

Infer Arity of $\mathcal{C}$ Expression

$$\mathcal{A} : \mathcal{C} \to (\mathsf{nat} + \mathsf{error_i})$$

Well-formedness of $\mathcal{C}$

$$\begin{array}{lcl} f(\epsilon, \vec{x}) & := & g(\vec{x}) \\ f(yi, \vec{x}) & := & h_i(y, f(y, \vec{x}), \vec{x}) \end{array}$$

h_i takes two more arguments than g.

Infer Arity of $\mathcal{C}$ Expression

$$\mathcal{A} : \mathcal{C} \to (\mathsf{nat} + \mathsf{error_i})$$

We must add information to compute the arity.

$$\begin{array}{ll} \Pi_i{}^n & \text{(projection with } i < n\text{)} \\ \mathsf{Comp}^n\ h\ \vec{l} & \text{(composition)} \end{array}$$