A certifying extraction with time bounds from Coq to call-by-value λ -calculus

Yannick Forster <u>Fabian Kunze</u>



Germany

ITP 2019 September 12

Motivation

Formalising computability theory:

- Functions definable in constructive type theory are computable, but this is not provable *inside* the theory
- Explicit model of computation needed for negative results or complexity theory
- We use: call-by-value λ -calculus "L"
- Computability provable for every concrete function defined in Coq
- But computability proofs tedious and repetitive!

We provide a framework that $automates^1$ this *extraction*

¹for ML-like subset of Coq

Idea

Call-by-Value lambda calculus (Plotkin [1975], Forster and Smolka [2017]):

$$s,t$$
 ::= $x \mid \lambda s \mid st$

- For each concrete function $f: X_1 \rightarrow \ldots \rightarrow X_n$ over ML-like data types,
- find a λ -term t_f and a time complexity τ_f such that
- for all \vec{x} , t_f (enc \vec{x}) reduces to enc($f\vec{x}$)
- within $\tau_f \vec{x}$ many beta-reduction steps (Accattoli and Lago [2016]). Simple, syntax directed extraction process.

Example



Example with time



Example with recursive function



Example with higher order function



Yannick Forster, Fabian Kunze

Outline

- What *is* correctness?
- How to extract terms?
- How to prove those extractions correct?
- Case studies:
 - Self-interpreter for L
 - Enumerator for diophantine equations (\rightarrow Reduction to L)
 - Turing machine interpreter

What is Correctness?

Encoding values: Scott encoding

Inductive values encoded as functions (corresponding to their match):

true
$$\lambda tf. t$$

false $\lambda tf. f$
0 $\lambda sz.z$
1 $\lambda sz.s(\lambda sz.z)$
S $\lambda asz.sa$

 \rightarrow Encoding follows from definition of inductive data type

Generating encoding functions

Mechanically derivable² using Template Coq, part of MetaCoq (Sozeau et al. [2019]):



Stored in Typeclass, with properties (e.g. injectivity) proven by Ltac.

²for all ML-like (non-dependent, non-mutual) inductive types

When does a $(\lambda$ -)term t_f compute a (Coq-)function f? Example: $t_{orb} \sim orb$:

```
\forall xy : \mathbb{B}, t_{\texttt{orb}} (\texttt{enc} x) (\texttt{enc} y) \succ^* \texttt{enc}(\texttt{orb} x y)
```

Tool: Logical relation!

When does a $(\lambda$ -)term t_f compute a (Coq-)function f? Example: $t_{orb} \sim orb$:

 $\forall xy : \mathbb{B}, t_{\texttt{orb}} (\texttt{enc} x) (\texttt{enc} y) \succ^* \texttt{enc}(\texttt{orb} x y)$

Tool: Logical relation!

$$\cfrac{}{=} \operatorname{enc}_A a \sim a \quad (ext{for } a : A)$$

$$\frac{\forall (a:A)(t_a:\mathbf{T}). \ t_a \sim a \rightarrow \qquad t_f t_a \qquad \sim fa}{t_f \sim f} \quad (\text{for } f:A \rightarrow B)$$

When does a $(\lambda$ -)term t_f compute a (Coq-)function f? Example: $t_{orb} \sim orb$:

 $\forall xy : \mathbb{B}, t_{\texttt{orb}} (\texttt{enc} x) (\texttt{enc} y) \succ^* \texttt{enc}(\texttt{orb} x y)$

Tool: Logical relation!

$$\frac{t_f \text{ is closed value } \land}{\frac{\forall (a:A)(t_a:T). \ t_a \sim a \rightarrow \Sigma(v:T). \ t_f t_a \succ^* v \land v \sim fa}{t_f \sim f}} \quad (\text{for } f:A \rightarrow B)$$

When does a $(\lambda$ -)term t_f compute a (Coq-)function f? Example: $t_{orb} \sim orb$:

 $\forall xy : \mathbb{B}, t_{\texttt{orb}} (\texttt{enc}x) (\texttt{enc}y) \succ^* \texttt{enc}(\texttt{orb} x y)$

Tool: Logical relation!

$$rac{1}{\operatorname{enc}_A a \sim a}$$
 (for $a:A$)

$$\frac{t_f \text{ is closed value } \land}{\frac{\forall (a:A)(t_a:T). \ t_a \sim a \rightarrow \Sigma(v:T). \ t_f t_a \succ^* v \land v \sim fa}{t_f \sim f}} \quad (\text{for } f:A \rightarrow B)$$

Problem: Not strictly positive. Solution: Recursion on Type...?

```
Inductive \mathcal{T}: Type \rightarrow Type :=

\mathcal{T}_{\text{base } A} '{registered A} : \mathcal{T} A

| \mathcal{T}_{\text{arr } A \ B} (\tau_{-1} : \mathcal{T} \ A) (\tau_{-2} : \mathcal{T} \ B) : \mathcal{T} \ (A \rightarrow B).

Fixpoint computes {A} (\tau : \mathcal{T} \ A): A \rightarrow \mathbf{T} \rightarrow Type :=

match \tau with

\mathcal{T}_{\text{base } \Rightarrow} fun x xInt \Rightarrow (xInt = enc x)

| \mathcal{C}\mathcal{T}_{\text{arr } A \ B \ \tau_1 \ \tau_2 \Rightarrow

fun f t_f \Rightarrow

proc t_f * \forall (a : A) t_a,

computes \tau_1 a t_a

\rightarrow {v : term & (t_f \ t_a >* v) * computes \tau_2 (f a) v}

end%type.
```

Complexity functions

Describe number of steps dependent on input. Example:

```
orb true false
(fun x y : bool \Rightarrow if x then true else y) true false
(fun y : bool \Rightarrow if true then true else y) false
(if true then true else false)
true
```

 \Rightarrow Time described by function

$$au_{ t orb}:\mathbb{B} o\mathbb{N} imes(\mathbb{B} o\mathbb{N}):=\lambda x.(1,\lambda y.2)$$

Complexity functions (2)

• Recursive functions:

 $\tau_{+}:\mathbb{N}\to\mathbb{N}\times(\mathbb{N}\to\mathbb{N}):=\lambda m.(c_{1},\lambda n.m\cdot c_{2}+c_{3})$

- Higher-order functions: Type for time-complexity functions actually more involved, see paper for full truth.
- Correctness predicate easily extended with time-complexity functions:

$$t_f \sim^{\tau_f} f$$

Extracting Functions

Extracting functions

- Variables translate to variables according to environment
- Abstraction translates to lambda
- Fixpoints translate to fixed-point combinator ρ

```
Fixpoint extract (env : nat \rightarrow nat) (s : Ast.term) (fuel : nat) :

TemplateMonad T :=

match fuel with 0 \Rightarrow tmFail "out of fuel" | S fuel \Rightarrow

match s with

Ast.tRel n \Rightarrow ret (var (env n))

| Ast.tLambda _ s \Rightarrow

t \leftarrow extract (\uparrow env) s fuel ;;

ret (lam t)

| Ast.tFix [Ast.mkdef _ _ s _] \Rightarrow

t \leftarrow extract (\uparrow env) s fuel ;;

ret (\rho (lam t))
```

Extracting functions (2)

- Applications are application
- Extractions of constants get reused using Typeclasses

```
| Ast.tApp s R \Rightarrow
  params \leftarrow tmDependentArgs s;;
  if params =? 0 then
    t \leftarrow extract env s fuel::
    monad_fold_left (fun t1 s2 \Rightarrow t2 \leftarrow extract env s2 fuel ;; ret (app t1
    t2)) R t
  else
    let (P, L) := (firstn params R, skipn params R) in
    a \leftarrow tmUnquote s';;
    a' \leftarrow tmEval cbn (my_projT2 a);;
    i <- tmTryInfer nm (Some cbn) (extracted a') ;;
    let t := (@int ext i) in
    monad_fold_left (fun t1 s2 \Rightarrow t2 \leftarrow extract env s2 fuel ;; ret (app t1
    t2)) L t
```

Proving correctness

Each extracted term is certified using Ltac:

- Tactic reducing λ -terms keeping track of number of steps
- Show correctness by following same case distinctions/recursions as used in the extracted function









emacs Prelude/Research/complex_but_formal/repos/certifying-extraction-with-time-bounds-public/talkExamples/enExtractDetailsNoTime.v = =				
<pre>Require Import LTactics Datatypes.Lists ComputableTactics. Import Intern. Definition map (A B : Type) (f : A → B):= fix map (l : list A) : list B := match L with [] ⇒ [] a : t → f a :: map t end. Lemma term_map (X Y:Type) (Hx : registered X) (Hy:register ed Y): computable (@map X Y). Proof. extractAs s. computable_using_noProof s. cstep. all:cstep. all:cstep. Qed.</pre>	<pre>1 subgoal (ID 812) X : Type Y : Type Hx : registered X Hy : registered Y s := (lam (rho (lam (lam ((0 (int_ext [])) (lam (lam ((((-</pre>			

```
emacs Prelude - ~/Research/complex_but_formal/repos/certifying-extraction-with-time-bounds-public/talkExamples/exExtractDetailsNoTime.v
Require Import LTactics Datatypes.Lists ComputableTactics.
                                                                                 := rho rP : term
Import Intern.
                                                                                   : proc P
                                                                               IHP
Definition map (A B : Type) (f : A \rightarrow B):=
                                                                                       {v : term &
                                                                                         ((fix map (l : list X) : list Y := match l with
Lemma term_map (X Y:Type) (Hx : registered X) (Hy:register
ed Y):
                                                                            x1Int, x1Ints : T
Proof.
  cstep.
                                                                             subgoal 2 (ID 2266) is:
                                                                             computes (! list Y)
                                                                                (x \times_{\Omega} :: (fix map (l_{\Omega} : list X) :: list Y := match l_{\Omega} wit)
Qed.
                                                                                (enc (x x_{\Omega} :: (fix map (l_{\Omega} : list X) :: list Y := match l
```

Yannick Forster, Fabian Kunze





Solving recurrence relations interactively



Yannick Forster, Fabian Kunze

Solving recurrence relations interactively

emacs Prelude/Research/complex_but_formal/repos/certifying-extraction-with-time-bounds-public/talkEsamples/exExtractDetailsFindRec.v _ 0			
Require Import LTactics Datatypes.Lists ComputableTactics.	3 subgoals (ID 6913)		
<pre>Definition map (A B : Type) (f : A → B):= fix map (l : list A) : list B := match l with [] → [] a :: l → f a :: map l end.</pre>	<pre>Y : Type Y : Type Hx : registered X Hy : registered Y used_term := lam (rho (lam (lam ((0 (int_ext [])) (lam (* lam (((int_ext cons) (5 1)) (3 0))))))) : extracted (map (B:=Y)) x : X → Y</pre>		
Lemma term_map (X Y:Type) (Hx : registered X) (Hy:register:	xT : X \rightarrow unit \rightarrow N \ast unit		
ed Y): computableTime (@map X Y) (λ f fT → (cnst "c",	1 ≤ cnst "c"		
$\lambda \mid \ \) \Rightarrow (cnst ("f",l),tt))).$	subgoal 2 (ID 7179) is:		
extract.	subgoal 3 (ID 8200) is:		
solverec. 3:rename xT into τ _f , x ₂ into a, l into l.	fst (τ _f a tt) + cnst ("f", l) + 11 ≤ cnst ("f", a ∷ l)		
Qed.	נוקאים יובטלטי אלו הל גער (נוקל) – נוסק מאלה הו לקור כמקומיץ שלה לולוסילטילון ליטןפרולה עקאי העויד לוי		
U:0 exExtractOetailsFindRec.v All of 482 (17,53) (Coq Script(3-) +4 ₩ hs Outl Holes guru yas FlyC- company (No changes need to be saved)	H U:g8%- +response+ All of 0 (1,0) (Coq Response FlyC company Helm EditorConfig Projectile super-save Pre		

Yannick Forster, Fabian Kunze

Solving recurrence relations interactively

emacs Prelude/Research/complex_bot_formal/repos/certifying-extraction-with-time-bounds-public/talkExamples/exExtractDetailsFindRec2.v _ 0 =				
Require Import LTactics Datatypes.Lists ComputableTactics.	2 subgoals (ID 7136)			
Definition map (A B : Type) (f : A \rightarrow B):=	X : Type			
fix map (l : list A) : list B :=	Y : Type			
match l with	Hx : registered X			
[] \Rightarrow []	Hy : registered Y			
a :: l \Rightarrow f a :: map l	used_term := lam (rho (lam (lam ((0 (int_ext [])) (lam (*			
end.	ilam (((int_ext cons) (5 1)) (3 0)))))))			
Lemma term_map (X Y:Type) (Hx : registered X) (Hy:register	: extracted (map (B:=Y))			
ed Y):	x : X \rightarrow Y			
computableTime (@map X Y)	xT : X \rightarrow Unit \rightarrow N * Unit			
(λ f f \Rightarrow (1,	x ₁ , x ₀ : list X			
λ l $_ \Rightarrow$ (cnst ("f",l),tt))).	xIT ₀ : T			
Proof.	H : x ₀ = []			
extract.	7 \leq cnst ("f", [])			
Solverec. 2:rename xT into τ _f , x ₂ into a, l into l.	subgoal 2 (ID 8127) is:			
Qed.	fst (τ _f a tt) + cnst ("f", l) + 11 ≤ cnst ("f", a ∷ l)			
υγφ externationalizationeca, xN(qf 45 (17,50) (constructed) -4 ♥ ins duit index gave yet flyce company	Lene reals Altef al (D,d) (cog 6a)s + hjc cogeny hele faltedening frajetile separator for			

Yannick Forster, Fabian Kunze

Solving recurrence relations interactively



Yannick Forster, Fabian Kunze

Case Study

Library of datatypes and functions

Build upon shared extraction of:

- \mathbb{B} , options, pairs . . .
- $\bullet~\mathbb{N}:$ addition, multiplication, equality \ldots
- Lists: map, filter, ...
- pprox 360 lines

Universal L-term:

- Function eva : $\mathbb{N} \to \mathbf{T} \to \mathbf{T}_{\perp}$ with $s \succ^* t \Leftrightarrow \exists n, eva \ n \ s = t$
- Base for many results in computability theory
- 20 lines from specification in Coq to correct extraction in L



emacsPrelude /Research/complex_but_formal/reposicertifying-extraction-with-time-bounds-public/Functions/Universal.v _ 0 ×				
(** ** Encoding for L-terms *)				
Run TemplateProgram (tmGenEncode "term_enc" term). Hint Resolve term_enc_correct : Lrewrite.				
(* register the non-constant constructors *) Instance term_var : computableTime var (λ n _ \Rightarrow (1, tt)). Proof. extract constructor. solverec. Qed.				
Instance term_app : computableTime app ($\lambda s_1 = \Rightarrow$ (1, ($\lambda s_2 = \Rightarrow$ (1, tt)) \Rightarrow •)). Proof. extract constructor. solverec. Qed.				
Instance term_lam : computableTime lam (λ s _ ⇒ (1, tt)). Proof. extract constructor. solverec. Qed.				
<pre>(** ** Extracted equality on natural numbers **) Instance termI_nat_eqb: computableTime Nat.eqb (λ x xT ⇒ (5,(λ y yT ⇒ (* (* (min x y)+15 * 8,tt)))). Proof. extract. solverec. Qed.</pre>				
(** ** Extracted substitution **)				
Instance term_substT : computableTime subst ($\lambda s \rightarrow (5, (\lambda n _ \Rightarrow (1, (\lambda t _ \Rightarrow (15 * n * s) * z s + 43 * (size s) ^ 2 + 13, tt)))))).$				
Proof. extract. solverec. Qed.				
(** ** Extracted step-indexed L-interpreter *)				
Instance term_eva : computable eva. Proof. extract. Qed.	Big •palse All of 8 (1,0) (Cog Goals PlyC- company Welm EditorConfig Projectile super-sove Fre W			
	closed weaponer. Act of a (1,0) (cod keaponse Fiye company Heim Editorioning Projective Super-save Pre-			

emacs Prelude - ~/Research/complex_but_formal/repos/cert	ifying-extraction-with-time-bounds-public/Functions/Universal.v = * *
(** ** Encoding for L-terms *)	
Run TemplateProgram (tmGenEncode "term_enc" term). Hint Resolve term_enc_correct : Lrewrite.	
(* register the non-constant constructors *) Instance term_var : computableTime var (λ n $_{-}$ \Rightarrow (1, tt)). Proof. extract constructor. solverec. Qed.	
Instance term_app : computableTime app ($\lambda \le_1 _ \Rightarrow (1, (\lambda \le_2 _ \Rightarrow (1, tt)) \Rightarrow)$). Proof. extract constructor. solverec. Qed.	
Instance term_lam : computableTime lam (λ s _ ⇒ (1, tt)). Proof. extract constructor. solverec. Qed.	
(** ** Extracted equality on natural numbers **) Instance termT_nat_eqb: computableTime Nat.eqb (λ x xT ⇒ (5,(λ y yT ⇒ (* (min x y)+15 * 8,tt))). Proof. extract. solverec. Qed.	
(** ** Extracted substitution **)	
<pre>Instance term_substT : computableTime subst (\ s _ → (5, (\ n _ → (1, (\ t _ → (15 * n * si* ze s + 43 * (size s) ^ 2 + 13, tt))))). Proof. extract. solverec. Qed.</pre>	
(** ** Extracted step-indexed L-interpreter *)	
Instance term_eva : computable eva. Proof. extract. Qed.	rige reals: All of 0 (1,0) (Cog Gals Tyde company sele differently despective sport-one des ten_solutions
-:@ Universal.v Bot of 1.3k (36,30) Git:wip (Coq Script(0-) +3 ♥ hs Outl Holes guru yas FlyC- company Helm	E B U:055- ∗response∗ All of 23 (1,0) (Coq Response FlyC company Helm EditorConfig Projectile super-save Pre

emacs Prelude/Research/complex_but_formal/repos/certifying-extraction-with-time-bounds-public/Functions/Universal.v _ 0 ×				
(** ** Encoding for L-terms *) Run TemplateProgram (tmGenEncode "term_enc" term).				
<pre>Hint Resolve term_enc_correct : Lrewrite. (* register the non-constant constructors *)</pre>				
Instance term_var : computableTime var $(\lambda n _ \Rightarrow (1, tt))$. Proof . extract constructor. solverec. Qed .				
Instance term_app : computableTime app ($\lambda \leq_1 = \Rightarrow (1, (\lambda \leq_2 = \Rightarrow (1, tt)) + \cdot)$).				
Proof. extract constructor. solverec. Qea.				
Instance term_lam : computableTime lam ($\lambda \ s \ * \ (1, \ tt)$). Proof. extract constructor. solverec. Qed.				
(** ** Extracted equality on natural numbers **) Instance termT_nat_eqb: computableTime Nat.eqb (λ x xT \Rightarrow (5,(λ y yT \Rightarrow (*				
*(min x y)+15 + 8,tt)))). Proof . extract. solverec. Qed .				
(** ** Extracted substitution **)				
Instance term_substT : computableTime subst (λ s _ ⇒ (5, (λ n _ ⇒ (1, (λ t _ ⇒ (15 * n * si•				
•ze s + 43 * (size s) ^ 2 + 13, tt))))). P roof . extract. solverec. Qed .				
(** ** Extracted step-indexed L-interpreter *)				
Instance term_eva : computable eva.				
Proof. extract. Qed.	U:@ •goals• All of 0 (1,0) (Coq Goals FlyC- company Helm EditorConfig Projectile super-save	Pre WK		
	tern_eva is defined			
-:ℓ Universal.v Bot of 1.3k (41,20) Git:wip (Coq Script(0-) +3 ♥ hs Outl Holes guru yas FlyC- company Helm	E U:2000- •response• All of 20 (1,0) (Coq Response FlyC company Helm EditorConfig Projectile super-sav	e Pre V		

Diophantine equations

- Diophantine sets are recursively enumerable
- $\bullet \Rightarrow$ Many-one reduction from diophantine equations to L-halting problem
- \approx 200 lines

Turing machine interpreter

• Each step of a TM can be computed in constant time in L:

```
Global Instance term_loopM :

let c1 := (haltTime + n*121 + transTime + 76) in

let c2 := 13 + haltTime in

computableTime (loopM (M:=M))

(fun _ \Rightarrow (5,fun k \Rightarrow (c1 * k + c2,tt))).
```

- Many-one reduction from TM-halting to L-halting problem
- \approx 400 lines

Future Work

- Formalise complexity theory:
 - Extract efficient self interpreter
 - Time Hierarchy Theorem
 - NP-Completeness
- Include space analysis in framework
- Verify extraction process using MetaCoq
- Better treatment of dependent functions (realisability model for Coq)

Related Work

- Myreen and Owens: HOL4 to CakeML
- Hupel and Nipkow: Isabelle/HOL to CakeML
- Köpp: Minlog to λ -Calculus
- Œuf project: Verified compiler from Coq-Subset to Assembly
- CertiCoq project: Verified extraction from Coq to Clight.

Contribution

- A plugin extracting Coq functions of simple polymorphic types to cbv $\lambda\text{-calculus L}$
- Logical relations connecting Coq functions with correct extractions and time bounds
- Automated correctness and semi-automated time analysis for extracted terms
- \bullet Three case studies and library including $\mathbb N$ and lists:
 - Universal L-term
 - Reduction from Diophantine equations to L-halting problem
 - Polynomial-time simulation of Turing machines in L
- Contributed to library of undecidable problems in Coq: github.com/uds-psl/coq-library-undecidability

Thank you!

References

- Beniamino Accattoli and Ugo Dal Lago. (Leftmost-Outermost) Beta Reduction is Invariant, Indeed. Logical Methods in Computer Science, 12 (1), 2016. doi: 10.2168/LMCS-12(1:4)2016.
- Yannick Forster and Gert Smolka. Weak call-by-value lambda calculus as a model of computation in Coq. In *ITP 2017*, pages 189–206. Springer, 2017.
- Gordon D. Plotkin. Call-by-Name, Call-by-Value and the lambda-Calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975. doi: 10.1016/0304-3975(75)90017-1.
- Matthieu Sozeau, Abhishek Anand, Simon Boulier, Cyril Cohen, Yannick Forster, Fabian Kunze, Gregory Malecha, Nicolas Tabareau, and Théo Winterhalter. The MetaCoq Project. working paper or preprint, June 2019. URL https://hal.inria.fr/hal-02167423.

LoC

Framework:		
		loc
Correctness predicate		370
Extraction		380
Simplification Tactics		950
Verifying Tactics		420
total		2100
Case Studies:		
	spec	proof
Library	355	282
Universal Term	13	7
H10	79	124
Universal TM	243	151

Correctness with time bounds

$$rac{1}{\operatorname{enc}_{\mathcal{A}} a \sim^{ au} a}$$
 (for $a:A$)

$$\frac{t_{f} \text{ is a procedure } \wedge \qquad \forall at_{a}\tau_{a}. \ t_{a} \sim^{\tau_{a}} a \to \Sigma v : \mathbf{T}.}{t_{f}t_{a} \succ^{\leq n} v \wedge v \sim^{\tau} fa \text{ where } \tau_{f}a\tau_{a} = (n,\tau)}{t_{f} \sim^{\tau_{a}} f} \quad (\text{for } f : A \to B)$$