Closable and uniquely closable skeletons of untyped lambda terms, formally

Catherine Dubois[1], Nicolas Magaud[2] and Alain Giorgetti[3]

[1] Samovar, ENSIIE, Evry, France
[2] ICube, UMR 7357 CNRS - Université de Strasbourg, France
[3] Institut FEMTO-ST, UMR 6174 CNRS - Université de Franche-Comté, Besançon, France

CLA 2023 (École Polytechnique) - January 12-13, 2023

Outline

Outline

Previously in CLA

Formal methods about λ-terms and maps: formalizations, formal proofs, random generators

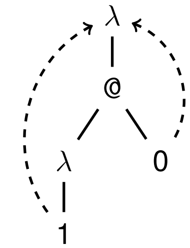
- Lambda terms and maps, formally,
  at CLA'15 (A. Giorgetti, C. Dubois and N. Zeilberger)
  - Bijection between labelled CLT and labelled ATM_1
  - Formalized in Coq and Prolog
  - No formal proof there, but validation by enumeration
- Lambda terms and maps, formally (II),
  at CLA'18 (A. Giorgetti and C. Dubois)
  - Introduction of blooms in the middle
  - semantics of blooms and maps, visually
  - Prolog specification of blooms
  - First application of blooms: Random and bounded-exhaustive testing

Pure λ terms in de Bruijn form

$$T ::= \mathbb{N} \mid \lambda\, T \mid T\ T$$

Implemented in Coq as an inductive definition:
unary-binary trees, aka labeled Motzkin trees

```
Inductive lmt : Set :=
| var : nat → lmt
| lam : lmt → lmt
| app : lmt → lmt → lmt.
```



for
$\lambda x.\,(\lambda y.\,x)\ x$

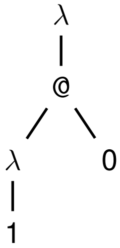
```
Definition ex1 := lam (app (lam (var 1)) (var 0)).
```

Closed λ terms

A λ-term in de Bruijn form is *closed* if the label at each leaf (*de Bruijn index*) is (strictly) smaller than the number of λs above it (*de Bruijn level*).

Closure property is not a *catamorphism* for `lmt`

i.e., "to be closed" cannot be defined recursively on the structure of labeled Motzkin trees.
$(\lambda\, t)$ can be closed for terms t that are not closed themselves.



$\lambda x.\,(\lambda y.\,x)\ x$

A possible solution: ignore labels,
use "skeletons" and a "closable" property instead [BT17].

[BT17] Olivier Bodini and Paul Tarau. On uniquely closable and uniquely typable skeletons of lambda terms. In *Logic-Based Program Synthesis and Transformation, LOPSTR 2017*, pages 252–268.

Outline

Our contribution: formalizing [BT17]

- Contents of [BT17]
  - Definitions and propositions
  - Implementation in Prolog (and a bit of Haskell)
  - Enumerators
- Our contribution: formalizing it in Coq
  - Coq definitions inspired by Prolog ones
  - Propositions proved in Coq
  - Random Generators
  - Testing before Proving (through Quickchick)

Motzkin trees (without labels)

A Motzkin tree is a rooted ordered tree built from binary nodes, unary nodes and leaf nodes. [BT17]

```
Inductive motzkin : Set :=
| v : motzkin
| l : motzkin → motzkin
| a : motzkin → motzkin → motzkin.
```

Closable Motzkin trees

A Motzkin tree is a skeleton of a closed lambda term if and only if it exists at least one λ binder on each path from the leaf to the root. [BT17, Prop. 2]

$\rightsquigarrow$ proposition proved later -
red part taken as a characterization of closable Motzkin trees

```
Fixpoint is_closable (mt: motzkin) :=
 match mt with
 | v ⇒ False
 | l m ⇒ True
 | a m1 m2 ⇒ is_closable m1 ∧ is_closable m2
 end.
```

```
      a
     / \
   l    l
   |    |
   l    a
   |   / \
   v  v    v
```

a closable Motzkin tree

```
      a
     / \
   l    a
   |   / \
   l  v    v
   |
   v
```

a non closable Motzkin tree

From a representation to another

```
Record rec_closable : Type := Build_rec_closable {
  closable_struct :> motzkin;
  closable_prop : is_closable closable_struct
}.
```

$$\texttt{rec\_closable2closable} \Big\downarrow \quad \Big\uparrow \texttt{closable2rec\_closable}$$

(two roundtrip lemmas proved in Coq)

```
Inductive closable :=
| La : motzkin → closable
| Ap : closable → closable → closable.
```

From a representation to another, an example

```
    a
   / \
  l   l
  |   |
  l   a
  |  / \
  v v   v
```

+ a proof that it's a closable Motzkin tree

`rec_closable2closable` $\downarrow$ $\uparrow$ `closable2rec_closable`

```
   Ap
  /  \
La    La
|     |
l     a
|    / \
v  v    v
```

Random generators

We use QuickChick (Coq port of Haskell QuickCheck) to test a Coq conjecture before proving it

- Executable properties and random generators needed!
- With the help of QuickChick combinators, quite easy to write (verified) random generators (sometimes derived)

We provide random generators for

- Motzkin trees (derived from `motzkin` def.)
- closable Motzkin trees (obtained by filtering, not efficient)
- closable Motzkin trees of type `rec_closable`
- objects of type `closable` (derived from `closable` def.)

```
(** ** Tests for [closable2rec_closableK] *)
QuickCheck (sized (fun n⇒
  forAll (gen_closable n) (fun c⇒
  (rec_closable2closable (closable2rec_closable c)) =? c)))
(*  +++ Passed 10000 tests  *)
```

Uniquely Closable Motzkin trees

A skeleton is uniquely closable if and only if exactly one lambda binder is available above each of its leaf nodes. [BT17, Prop. 4]

```
Definition is_ucs: motzkin → Prop := ....
```

```
Record rec_ucs : Type := Build_rec_ucs {
  ucs_struct : motzkin;
  ucs_prop : is_ucs ucs_struct
}.
```

$$\texttt{rec\_ucs2ucs} \downarrow \quad \uparrow \texttt{ucs2rec\_ucs}$$

(two roundtrip lemmas proved in Coq)

```
Inductive ca :=
| V : ca
| B : ca → ca → ca.
```

```
Inductive ucs :=
| L : ca → ucs
| A : ucs → ucs → ucs.
```

And random generators for `rec_ucs`, `ca` and `ucs`

Characterization of closable Motzkin trees

A Motzkin tree is the skeleton of a closed λ-term
if and only if
it exists at least one λ-binder on each path from the leaf to the root [BT17, Proposition 2]

How to formalize closed λ-terms?

"to be closed" cannot be defined recursively on the structure of labeled Motzkin trees: $(\lambda\, t)$ can be closed for terms t that are not closed themselves
$\leadsto$ extension to open terms (m-open terms)

[BT17] O. Bodini and P. Tarau. On uniquely closable and uniquely typable skeletons of lambda terms. In *LOPSTR 2017*, pages 252–268.

m-open λ-terms

The λ-term t is *m-open* if the term $(\lambda \ldots \lambda\, t)$ with m abstractions before t is closed [BBD19]

Aka. "λ-terms containing at most m distinct free variables" [Les13, GL13]

[Les13] Pierre Lescanne. On counting untyped lambda terms. *Theoretical Computer Science*, 474:80–97, February 2013.

[GL13] Katarzyna Grygiel and Pierre Lescanne. Counting and generating lambda terms. *Journal of Functional Programming*, 23(5):594–628, September 2013.

[BBD19] Maciej Bendkowski, Olivier Bodini, and Sergey Dovgal. Statistical Properties of Lambda Terms. *The Electronic Journal of Combinatorics*, pages P4.1, October 2019.

m-open λ-terms, formally

```
Fixpoint is_open (m: nat) (t: lmt) : Prop :=
 match t with
  | var i ⇒ i < m
  | lam t1 ⇒ is_open (S m) t1
  | app t1 t2 ⇒ is_open m t1 ∧ is_open m t2
 end.

Record rec_open (m:nat) : Set := Build_rec_open {
 open_struct :> lmt;
 open_prop : is_open m open_struct
}.
```

`rec_open2open` ↓ ↑ `open2rec_open`

(two roundtrip lemmas proved in Coq)

```
Inductive open : nat → Set :=
| open_var : ∀ (m i:nat), i < m → open m
| open_lam : ∀ (m:nat), open (S m) → open m
| open_app : ∀ (m:nat), open m → open m → open m.
```

m-open terms and skeletons

Label erasure

```
Fixpoint skeleton (t: lmt) : motzkin :=
 match t with
 | var _ ⇒ v
 | lam t1 ⇒ l (skeleton t1)
 | app t1 t2 ⇒ a (skeleton t1) (skeleton t2)
 end.
```

Characterization of closable Motzkin trees, in Coq

A Motzkin tree is the skeleton of a closed λ-term
if and only if
it exists at least one λ-binder on each path from the leaf to the root [BT17, Proposition 2]

```
Definition is_closed t := is_open 0 t.

Proposition proposition2 : ∀ mt : motzkin,
  (∃ t : lmt, skeleton t = mt ∧ is_closed t)
  ↔ is_closable mt.
```

Propositions 2 and 4 of [BT17], formalized in Coq

[BT17, Prop. 2]: *A Motzkin tree is the skeleton of a closed λ-term if and only if it exists at least one λ-binder on each path from the leaf to the root.*

```
Definition is_closed t := is_open 0 t.

Proposition proposition2 : ∀ mt : motzkin,
  (∃ t : lmt, skeleton t = mt ∧ is_closed t)
  ↔ is_closable mt.
```

[BT17, Prop. 4]: *A skeleton is uniquely closable if and only if exactly one lambda binder is available above each of its leaf nodes.*

```
Definition is_ucs m := is_ucs_aux m false.

Proposition proposition4: ∀ mt : motzkin,
  (∃! t, skeleton t = mt ∧ is_closed t) ↔ is_ucs mt.
```

Outline

Generalization

rec_P

```
Record rec_closable : Type := Build_rec_closable {
  closable_struct :> motzkin;
  closable_prop : is_closable closable_struct
}.
```

T

is_P

rec_closable2closable ↓ ↑ closable2rec_closable

(two roundtrip lemmas proved in Coq)

P

T2P : ∀ x:T, is_P x → P
P2T :P → T

```
Inductive closable :=
| La : motzkin → closable
| Ap : closable → closable → closable.
```

is_P_lemma : ∀ v, is_P (P2T v)
P2T_is_P : ∀ (t : T) (H : is_P t), P2T (T2P t H) = t

Generalization

rec_P

```
Record rec_closable : Type := Build_rec_closable {
  closable_struct :> motzkin;
  closable_prop : is_closable closable_struct
}.
```

T

is_P

rec_closable2closable ↓ ↑ closable2rec_closable

(two roundtrip lemmas proved in Coq)

P

```
Inductive closable :=
| La : motzkin → closable
| Ap : closable → closable → closable.
```

T2P : ∀ x:T, is_P x → P
P2T :P → T

is_P_lemma : ∀ v, is_P (P2T v)
P2T_is_P : ∀ (t : T) (H : is_P t), P2T (T2P t H) = t

Generalization

rec_P

```
Record rec_closable : Type := Build_rec_closable {
  closable_struct :> motzkin;
  closable_prop : is_closable closable_struct
}.
```

T

is_P

rec_closable2closable ↓ ↑ closable2rec_closable

(two roundtrip lemmas proved in Coq)

P

```
Inductive closable :=
| La : motzkin → closable
| Ap : closable → closable → closable.
```

T2P : ∀ x:T, is_P x → P
P2T :P → T

is_P_lemma : ∀ v, is_P (P2T v)
P2T_is_P : ∀ (t : T) (H : is_P t), P2T (T2P t H) = t

Generalization

rec_P

```
Record rec_closable : Type := Build_rec_closable {
  closable_struct :> motzkin;        T
  closable_prop : is_closable closable_struct
}.                is_P
```

rec_closable2closable ↓ ↑ closable2rec_closable

(two roundtrip lemmas proved in Coq)

P

```
Inductive closable :=
| La : motzkin → closable
| Ap : closable → closable → closable.
```

T2P : ∀ x:T, is_P x → P
P2T :P → T

is_P_lemma : ∀ v, is_P (P2T v)
P2T_is_P : ∀ (t : T) (H : is_P t), P2T (T2P t H) = t

An abstract representation of two datatypes

- Generic interface

```
Module Type family.
  Parameter T : Set.
  Parameter is_P : T → Prop.
  Parameter P : Set.
  Parameter T2P : ∀ (x:T), is_P x → P.
  Parameter P2T : P → T.
  Parameter is_P_lemma : ∀ v, is_P (P2T v).
  Parameter P2T_is_P :
    ∀ (t : T) (H : is_P t), P2T (T2P t H) = t.
  Parameter proof_irr :
    ∀ x (p1 p2:is_P x), p1 = p2.
End family.
```

Two instances: closable and uniquely closable Motzkin trees

Abstraction	Closable Skeletons	Uniquely Closable Skeletons
`T`	`motzkin`	`motzkin`
`is_P`	`is_closable`	`is_ucs`
`P`	`closable`	`ucs`
`T2P`	`motzkin2closable`	`motzkin2ucs`
`P2T`	`closable2motzkin`	`ucs2motzkin`
`is_P_lemma`	automatically proved using Ltac	
`P2T_is_P`	automatically proved using Ltac	
`proof_irr`	`PI_is_closable`	`PI_is_ucs`
`rec_P`	automatically derived in the functor	
`rec_P2P`	automatically derived in the functor	
`P2rec_P`	automatically derived in the functor	
`P2rec_PK`	automatically derived in the functor	
`rec_P2PK`	automatically proved using Ltac	

Random generators

3 interfaces for random generators, parametrized by a `family` module and 3 functors

- one pair (interface,functor) to derive `gen_P_filter` from `gen_T` and an executable version of `is_P`
- one pair to derive `gen_P_rec` from `gen_P` using the transformation `P_rec2P`
- one pair to derive `gen_P` from `gen_P_rec` using the transformation `T_rec2P`

```
Module Type family_gen3 (Import f : family).
  Parameter gen_P : nat→ G P.
End family_gen3.

Module genfamily3(Import f : family)(Import g : family_gen3 f)
 (Import facts : equiv_sig f).
  Definition gen_rec_P n : G rec_P :=
  do! p← gen_P n;
  returnGen (P2rec_P p).
End genfamily3.
```

Outline

Conclusions and Perspectives

- Achievements
  - Closable and uniquely closable Motzkin trees, formally!
  - Partial automation of proofs of roundtrip lemmas
  - Some verified random generators
- Perspectives
  - Typable Motzkin trees [BT17]
  - Linear λ-terms by Tarau et al. [TP20]
  - Catalan (Rémy bij.), Motzkin and Schröder trees [Les23]
  - Revisit [DG18] (permutations and rooted maps)

[BT17] Olivier Bodini and Paul Tarau. On uniquely closable and uniquely typable skeletons of lambda terms. In *Logic-Based Program Synthesis and Transformation, LOPSTR 2017*, pages 252–268.

[TP20] Paul Tarau and Valeria de Paiva. Deriving Theorems in Implicational Linear Logic, Declaratively. In Proceedings ICLP 2020, UNICAL, Rende (CS), Italy.

[DG18] Catherine Dubois and Alain Giorgetti. Tests and proofs for custom data generators. Formal Aspects Comput. 30(6): 659-684 (2018)

[Les23] Pierre Lescanne. Holonomic equations and efficient random generation of binary trees. CLA 2023.

Thanks! Questions?

http://arxiv.org/abs/2212.10453