Representations of Boolean Functions in Constructive Type Theory

Gilles Nies

Saarland University

April 20, 2012

Motivation and Goal

- Goal:
  Define a *informative canonical* representation of boolean functions in Coq $\Longrightarrow$ *prime trees*.
- Why:
  Coq's type theory is *intentional*.
- How:
  By showing that boolean functions and prime trees are isomorphic.

$$\textit{Prime Trees} \;\cong\; \textit{Boolean Functions}$$

The notion of isomorphism in Coq

- Trickier than expected!

The notion of isomorphism in Coq

- Trickier than expected!
- In mathematics: An invertible function $f : A \to B$

$$f\ isomorphism_{A,B} \ := \ \exists g : B \to A, \begin{cases} \forall b : B, & (f \circ g)(b) = b \\ \forall a : A, & (g \circ f)(a) = a \end{cases}$$

The notion of isomorphism in Coq

- Trickier than expected!
- In mathematics: An invertible function $f : A \to B$

$$f\ isomorphism_{A,B} \ := \ \exists g : B \to A, \left\{ \begin{array}{ll} \forall b : B, & (f \circ g)(b) = b \\ \forall a : A, & (g \circ f)(a) = a \end{array} \right.$$

- Generalization to relations needed but problematic:

$$f\ isomorphism_{A,B}^{\equiv_A,\equiv_B} := \exists g : B \to A, \left\{ \begin{array}{ll} \forall b : B, & (f \circ g)(b) \equiv_B b \\ \forall a : A, & (g \circ f)(a) \equiv_A a \end{array} \right.$$

The notion of isomorphism in Coq

- Trickier than expected!
- In mathematics: An invertible function $f : A \to B$

$$f \ isomorphism_{A,B} \ := \ \exists g : B \to A, \begin{cases} \forall b : B, & (f \circ g)(b) = b \\ \forall a : A, & (g \circ f)(a) = a \end{cases}$$

- Generalization to relations needed but problematic:

$$f \ isomorphism_{A,B}^{\equiv_A, \equiv_B} := \exists g : B \to A, \begin{cases} \forall b : B, & (f \circ g)(b) \equiv_B b \\ \forall a : A, & (g \circ f)(a) \equiv_A a \end{cases}$$

- With $\equiv_{all} \ := \ \mathbb{N} \times \mathbb{N}$ we now have

$$id \ isomorphism_{\mathbb{N},\mathbb{N}}^{\equiv_{all}, =}$$

The notion of isomorphism in Coq

- Trickier than expected!
- In mathematics: An invertible function $f : A \to B$

$$f\ isomorphism_{A,B} \ := \ \exists g : B \to A, \begin{cases} \forall b : B, & (f \circ g)(b) = b \\ \forall a : A, & (g \circ f)(a) = a \end{cases}$$

- Generalization to relations needed but problematic:

$$f\ isomorphism_{A,B}^{\equiv_A, \equiv_B} := \exists g : B \to A, \begin{cases} \forall b : B, & (f \circ g)(b) \equiv_B b \\ \forall a : A, & (g \circ f)(a) \equiv_A a \end{cases}$$

- With $\equiv_{all} \ := \ \mathbb{N} \times \mathbb{N}$ we now have

$$id\ isomorphism_{\mathbb{N},\mathbb{N}}^{\equiv_{all}, =}$$

- Additional constraint: Mappings must preserve equality.

$$\forall a_1, a_2 \in A : \ a_1 \equiv_A a_2 \Rightarrow g(a_1) \equiv_B g(a_2)$$

$$\forall b_1, b_2 \in B : \ b_1 \equiv_B b_2 \Rightarrow f(b_1) \equiv_A f(b_2)$$

The notion of isomorphism in Coq

- Use *setoids* and *morphisms* for elegant definition:

$$\begin{aligned} Setoid \quad &:= \quad \{(T : Type, \ \equiv_T : T \times T) \mid \ \equiv_T \ \ ER\} \\ (A, \equiv_A) \twoheadrightarrow (B, \equiv_B) \quad &:= \quad \{f : A \to B \mid f \text{ preserves } \equiv_B\} \end{aligned}$$

The notion of isomorphism in Coq

- Use *setoids* and *morphisms* for elegant definition:

$$\begin{aligned} \mathit{Setoid} &:= \{(T : \mathit{Type},\ \equiv_T : T \times T) \mid\ \equiv_T\ \mathit{ER}\} \\ (A, \equiv_A) \twoheadrightarrow (B, \equiv_B) &:= \{f : A \to B \mid f \text{ preserves } \equiv_B\} \end{aligned}$$

- $f : (A, \equiv_A) \twoheadrightarrow (B, \equiv_B)$ and $g : (B, \equiv_B) \twoheadrightarrow (A, \equiv_A)$ form a *setoid-isomorphism* iff

$$\begin{aligned} \forall b : B, &\quad (f \circ g)(b) \equiv_B b \\ \forall a : A, &\quad (g \circ f)(a) \equiv_A a \end{aligned}$$

Boolean functions in theory

- Finite set of variables: V
- Assignments (σ): $V \rightarrow bool$
- Boolean functions (ϕ, ψ): $(V \rightarrow bool) \rightarrow bool$

Decision trees in theory

- Based on *conditionals*:

$$(x, s, t) \;:=\; (x \wedge s) \vee (\neg x \wedge t)$$

where x is a variable and s, t are formulas.

Decision trees in theory

- Based on *conditionals*:

$$(x, s, t) \;:=\; (x \wedge s) \vee (\neg x \wedge t)$$

  where x is a variable and s, t are formulas.
- Defined inductively:
  - $\top$ and $\perp$ are decision trees.
  - (x, s, t) is a decision tree iff x variable and s, t decision trees.

Decision trees in theory

- Based on *conditionals*:

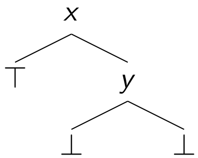
$$(x, s, t) \;:=\; (x \wedge s) \vee (\neg x \wedge t)$$

  where x is a variable and s, t are formulas.

- Defined inductively:
  - $\top$ and $\bot$ are decision trees.
  - (x, s, t) is a decision tree iff x variable and s, t decision trees.

- Tree interpretation:

$$(x, \top, (y, \bot, \bot)) \implies$$



Prime trees in theory

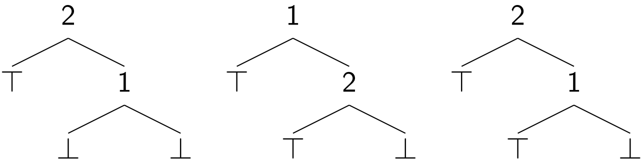
- Prime trees are *reduced and ordered* decision trees.
  Let t be a decision tree.

Prime trees in theory

- Prime trees are *reduced and ordered* decision trees.
  Let t be a decision tree.
  - t is reduced if none of its subtrees is of the form (x, t', t').
  - t is ordered if the variables become smaller as one descends t.

Prime trees in theory

- Prime trees are *reduced and ordered* decision trees.
  Let t be a decision tree.
  - t is reduced if none of its subtrees is of the form (x, t', t').
  - t is ordered if the variables become smaller as one descends t.
- Examples: Let $V := \{1, 2\}$



ordered not reduced	not ordered reduced	ordered reduced $\Rightarrow$ prime tree

Roadmap

- Whatever definitions used,
  - $\mathcal{BF} :=$ Boolean functions
  - $\mathcal{DT} :=$ Decision trees
  - $\mathcal{PT} :=$ Prime trees

  we will need:
  - Decidable equality: $\forall t_1 t_2 : \mathcal{DT},\ \{t_1 = t_2\} + \{t_1 \neq t_2\}$
  - Denotational completeness: $\forall \phi : \mathcal{BF},\ \{t : \mathcal{PT} \mid [\![t]\!] \equiv \phi\}$
  - Core result : $\forall t_1 t_2 : \mathcal{PT},\ t_1 \neq t_2 \to [\![t_1]\!] \not\equiv [\![t_2]\!]$
- Morphisms:
  - Denotational Completeness (ex. V4) : $(\mathcal{BF}, \equiv) \twoheadrightarrow (\mathcal{PT}, =)$
  - Denotational Semantics $[\![\cdot]\!]$: $(\mathcal{PT}, =) \twoheadrightarrow (\mathcal{BF}, \equiv)$
- Isomorphism:

$$(\mathcal{BF}, \equiv) \cong (\mathcal{PT}, =)$$

- Cascaded boolean functions:

$$bool^n \rightarrow bool$$

Version 1: The dependently typed approach

Boolean functions in Coq

- Cascaded boolean functions:

$$bool^n \rightarrow bool$$

- `Fixpoint` $nfun\ A\ n\ B$:=

  `match` n `with`

  $\mid 0 \Rightarrow B$

  $\mid S\ n \Rightarrow A \rightarrow (nfun\ A\ n\ B)$

  `end.`

Version 1: The dependently typed approach
Boolean functions in Coq

- Cascaded boolean functions:

$$bool^n \to bool$$

- `Fixpoint` *nfun A n B* $:=$

```
match n with
  | 0 => B
  | S n => A -> (nfun A n B)
end.
```

- Equivalence:

$$\phi \equiv \psi \;:=\; \forall \vec{x},\ \phi\ \vec{x} = \psi\ \vec{x}$$

Version 1: The dependently typed approach

Decision trees in Coq

- `Inductive` DT : $nat \rightarrow$ `Type` :=
  $\mid DT_0 : DT\ 0$
  $\mid DT_1 : DT\ 0$
  $\mid DT_I : \forall \{n\}, DT\ n \rightarrow DT\ n \rightarrow DT\ (S\ n)$
  $\mid DT_L : \forall \{n\}, DT\ n \rightarrow DT\ (S\ n).$
- Dependency indicates number of variables the tree depends on.

- Denotational Semantics: Via recursion on the decision tree.

$$
\begin{array}{rcl}
[\![\bot]\!] & = & \mathit{false} \\
[\![\top]\!] & = & \mathit{true} \\
[\![(\_, t_1, t_2)]\!] & = & \lambda b : \mathit{bool}. \begin{cases} [\![t_1]\!], & b = \mathit{true} \\ [\![t_2]\!], & b = \mathit{false} \end{cases} \\
[\![t^1]\!] & = & \lambda \_ : \mathit{bool}.\ [\![t]\!]
\end{array}
$$

Version 1: The dependently typed approach

Denotational semantics and prime trees

- Denotational Semantics: Via recursion on the decision tree.

$$
\begin{array}{rcl}
[\![\bot]\!] & = & \mathit{false} \\
[\![\top]\!] & = & \mathit{true} \\
[\![(\_, t_1, t_2)]\!] & = & \lambda b : \mathit{bool}. \begin{cases} [\![t_1]\!], & b = \mathit{true} \\ [\![t_2]\!], & b = \mathit{false} \end{cases} \\
[\![t^1]\!] & = & \lambda \_ : \mathit{bool}.\ [\![t]\!]
\end{array}
$$

- Members of $DT\ \ n$ are already ordered by design.
- Thus prime trees are defined by

$$PT\ \ n \ := \ \{t : DT\ \ n \mid \mathit{reduced}\ \ t\}$$

- We need to write an *inversion* function ourselves:

  Definition $DT\_Inv\ \{n : nat\}\ (t : DT\ n)$:
   match n as z return $DT\ z \rightarrow$ Type with
    $|\ O \Rightarrow$ fun $t \Rightarrow \{t = \bot\} + \{t = \top\}$
    $|S\ n' \Rightarrow$ fun $t \Rightarrow$

  $$\{p : (DT\ n') \times (DT\ n') \mid t = (\_, fst\ p, snd\ p)\} + \{dt : DT\ n' \mid t = dt^1\}$$

   end t.

Version 1: The dependently typed approach
Decidable equality

- We need to write an *inversion* function ourselves:

  $\texttt{Definition}\ DT\_Inv\ \{n : nat\}\ (t : DT\ n) :$
  $\quad \texttt{match}\ n\ \texttt{as}\ z\ \texttt{return}\ DT\ z \to \texttt{Type with}$
  $\qquad \mid O \Rightarrow \texttt{fun}\ t \Rightarrow \{t = \bot\} + \{t = \top\}$
  $\qquad \mid S\ n' \Rightarrow \texttt{fun}\ t \Rightarrow$
  $\qquad\qquad \{p : (DT\ n') \times (DT\ n') \mid t = (\_, fst\ p, snd\ p)\}$
  $\qquad\qquad\qquad + \{dt : DT\ n' \mid t = dt^1\}$
  $\quad \texttt{end}\ t.$

- Decidable equality

  $$\forall t_1 t_2 : DT\ n,\ \{t_1 = t_2\} + \{t_1 \neq t_2\}$$

  by recursion on n.

- Get rid of annoying inversion function!

- Get rid of annoying inversion function!
- Solution: Recursive definition instead of inductive definition
- Fixpoint DT_{rec} $(n : nat)$: Type :=

  match n with

  $\mid 0 \Rightarrow bool$

  $\mid S\ n \Rightarrow (DT_{rec}\ n \times DT_{rec}\ n) + DT_{rec}\ n$

  end.

- Get rid of annoying inversion function!
- Solution: Recursive definition instead of inductive definition
- Fixpoint DT_{rec} $(n : nat)$: Type :=
  match n with
  $\mid 0 \Rightarrow bool$
  $\mid S\ n \Rightarrow (DT_{rec}\ n \times DT_{rec}\ n) + DT_{rec}\ n$
  end.
- No complicated inversion function needed.

Version 1 and 2: Rest of roadmap

- Denotational Completeness

$$\forall \phi : bool^n \to bool,\ \{t : PT\ n \mid [\![t]\!] \equiv \phi\}$$

by recursion on number of arguments n.

Version 1 and 2: Rest of roadmap

- Denotational Completeness

$$\forall \phi : bool^n \to bool,\ \{t : PT\ \ n \mid [\![t]\!] \equiv \phi\}$$

by recursion on number of arguments n.

- Core result:

$$\forall t_1 t_2 : PT\ \ n,\ \ t_1 \neq t_2 \to [\![t_1]\!] \not\equiv [\![t_2]\!]$$

via induction on level n.

Version 1 and 2: Rest of roadmap

- Denotational Completeness

$$\forall \phi : bool^n \to bool,\ \{t : PT\ \ n \mid [\![t]\!] \equiv \phi\}$$

  by recursion on number of arguments n.
- Core result:

$$\forall t_1 t_2 : PT\ \ n,\ \ t_1 \neq t_2 \to [\![t_1]\!] \not\equiv [\![t_2]\!]$$

  via induction on level n.
- Both straightforward.

Versions 1 and 2: Summary

- Copies!
  - Boolean functions: *true*, $\lambda\_.true$, $\lambda\_\_.true$, ...
  - Prime Trees: $\top$, $\top^1$, $\top^2$, ...

Versions 1 and 2: Summary

- Copies!
  - Boolean functions: true, $\lambda\_.\mathit{true}$, $\lambda\_\_.\mathit{true}$, ...
  - Prime Trees: $\top$, $\top^1$, $\top^2$, ...
- Almost no automation for inductive DTs:
  - Tactics `inversion` and `injection` fail to deliver.
  - Inversion function

Versions 1 and 2: Summary

- Copies!
  - Boolean functions: true, $\lambda\_.\mathit{true}$, $\lambda\_\_.\mathit{true}$, ...
  - Prime Trees: $\top$, $\top^1$, $\top^2$, ...
- Almost no automation for inductive DTs:
  - Tactics `inversion` and `injection` fail to deliver.
  - Inversion function
- Remedy: Recursive Decision trees.
  - `injection` works perfectly $\Rightarrow$ shorter proofs.
  - Coq's `destruct` and reduction instead of inversion function.

Versions 1 and 2: Summary

- Copies!
  - Boolean functions: true, $\lambda\_.\mathit{true}$, $\lambda\_\_.\mathit{true}$, ...
  - Prime Trees: $\top$, $\top^1$, $\top^2$, ...
- Almost no automation for inductive DTs:
  - Tactics `inversion` and `injection` fail to deliver.
  - Inversion function
- Remedy: Recursive Decision trees.
  - `injection` works perfectly $\Rightarrow$ shorter proofs.
  - Coq's `destruct` and reduction instead of inversion function.
- Actually pretty convenient to work with.

Simply typed decision trees

- Goal: Get rid of dependency and copies.

Simply typed decision trees

- Goal: Get rid of dependency and copies.
- `Inductive` SDT : `Type` :=
  $\mid DT_0 : SDT$
  $\mid DT_1 : SDT$
  $\mid DT_I : nat \to SDT \to SDT \to SDT.$
- Variable on which to branch explicitly given to branching constructor.
- No dependency, only one $\top$-tree

Simply typed decision trees

- Goal: Get rid of dependency and copies.
- `Inductive` SDT : `Type` :=
  $| \ DT_0 : SDT$
  $| \ DT_1 : SDT$
  $| \ DT_l : nat \to SDT \to SDT \to SDT.$
- Variable on which to branch explicitly given to branching constructor.
- No dependency, only one $\top$-tree
- Decidable equality comes for free: `decide equality`.

- Semantics:

$$
\begin{aligned}
[\![\bot]\!] &= \lambda\_.\ \mathit{false} \\
[\![\top]\!] &= \lambda\_.\ \mathit{true} \\
[\![(n, t_1, t_2)]\!] &= \lambda\sigma : (\mathit{nat} \to \mathit{bool}). \begin{cases} [\![t_1]\!]\sigma, & \sigma\ n = \mathit{true} \\ [\![t_2]\!]\sigma, & \sigma\ n = \mathit{false} \end{cases}
\end{aligned}
$$

Simply typed decision trees
Sematics and prime trees

- Semantics:

$$
\begin{array}{rcl}
[\![\bot]\!] & = & \lambda \_.\ \mathit{false} \\
[\![\top]\!] & = & \lambda \_.\ \mathit{true} \\
[\![(n, t_1, t_2)]\!] & = & \lambda \sigma : (\mathit{nat} \to \mathit{bool}). \begin{cases} [\![t_1]\!]\sigma, & \sigma\ n = \mathit{true} \\ [\![t_2]\!]\sigma, & \sigma\ n = \mathit{false} \end{cases}
\end{array}
$$

- Unfortunately not ordered by design:

$$
SPT \ := \ \{t : SDT \mid \mathit{reduced}\ t \wedge \mathit{ordered}\ t\}
$$

Alternative boolean functions

- *SPT* not isomorphic to cascaded boolean functions while preserving meaning: $\mathit{true}, \lambda\_.\mathit{true}$, ... all map to $\top$.

Alternative boolean functions

- SPT not isomorphic to cascaded boolean functions while preserving meaning: $true, \lambda\_.true$, ... all map to $\top$.
- Alternative definition:

$$BF \;:=\; (nat \to bool) \to bool$$

- Equivalence:

$$\phi \equiv \psi \;:=\; \forall \sigma : (nat \to bool),\; \phi\ \sigma = \psi\ \sigma$$

Alternative boolean functions

- *SPT* not isomorphic to cascaded boolean functions while preserving meaning: $\mathit{true}, \lambda\_.\mathit{true}$, ... all map to $\top$.
- Alternative definition:

$$BF \;:=\; (\mathit{nat} \to \mathit{bool}) \to \mathit{bool}$$

- Equivalence:

$$\phi \equiv \psi \;:=\; \forall \sigma : (\mathit{nat} \to \mathit{bool}),\ \phi\ \sigma = \psi\ \sigma$$

- Only one constant true function : $\lambda\_ : (\mathit{nat} \to \mathit{bool}).\ \mathit{true}$

Alternative boolean functions

- *SPT* not isomorphic to cascaded boolean functions while preserving meaning: $\mathit{true}, \lambda\_.\mathit{true}$, ... all map to $\top$.
- Alternative definition:

$$\mathit{BF} \;:=\; (\mathit{nat} \to \mathit{bool}) \to \mathit{bool}$$

- Equivalence:

$$\phi \equiv \psi \;:=\; \forall \sigma : (\mathit{nat} \to \mathit{bool}),\ \phi\ \sigma = \psi\ \sigma$$

- Only one constant true function : $\lambda\_ : (\mathit{nat} \to \mathit{bool}).\ \mathit{true}$
- Infinitely many variables $\Rightarrow$ infinite decision trees!
- Restriction to only the *continuous* boolean functions

Continuous boolean functions

- $cts_n\ \phi\ :=$ "it suffices to consider the first n variables to evaluate ϕ".
  - n is a *modulus of continuity*.
- $cts\ \phi\ := \exists n : nat,\ cts_n\ \phi$

Continuous boolean functions

- $cts_n\ \phi \ := $ "it suffices to consider the first n variables to evaluate ϕ".
  - n is a *modulus of continuity*.
- $cts\ \phi \ := \exists n : nat,\ cts_n\ \phi$
- Initial idea:

$$BF_{cts} \ := \ \{\phi : BF \mid cts\ \phi\}$$

Continuous boolean functions

- $cts_n\ \phi\ :=$ "it suffices to consider the first n variables to evaluate ϕ".
  - n is a *modulus of continuity*.
- $cts\ \phi\ := \exists n : nat,\ cts_n\ \phi$
- Initial idea:

$$BF_{cts}\ :=\ \{\phi : BF \mid cts\ \phi\}$$

- Denotational completeness

$$\forall \phi : BF_{cts},\ \{t : SPT \mid [\![t]\!] \equiv \phi\}$$

impossible to obtain $\Longrightarrow$ *Elim restriction*.

Continuous boolean functions

- $cts_n\ \phi\ :=$ "it suffices to consider the first n variables to evaluate ϕ".
  - n is a *modulus of continuity*.
- $cts\ \phi\ := \exists n : nat,\ cts_n\ \phi$
- Initial idea:

$$BF_{cts}\ :=\ \{\phi : BF \mid cts\ \phi\}$$

- Denotational completeness

$$\forall \phi : BF_{cts},\ \{t : SPT \mid [\![t]\!] \equiv \phi\}$$

  impossible to obtain $\Longrightarrow$ *Elim restriction*.
- 3 possibilities to circumvent the elim restriction.

Version 3: Using the Axiom of Continuity
Denotational Completeness

- Drastic solution:
  - $ctsT\ \phi\ := \ \{n : nat \mid cts_n\ \phi\}$
  - `Axiom` $CTS : \forall\ \phi :\ BF, ctsT\ \phi.$

- Drastic solution:
  - $ctsT\ \phi \ := \ \{n : nat \mid cts_n\ \phi\}$
  - $\texttt{Axiom}\ CTS : \forall\ \phi :\ BF,\ ctsT\ \phi.$
- Plausible

$$\varphi\ \sigma := \begin{cases} true : & \forall n,\ \sigma\ n = true \\ false : & otherwise \end{cases} \qquad : (nat \to bool) \to \color{red}{Prop}$$

- Drastic solution:
    - $ctsT\ \phi\ := \ \{n : nat \mid cts_n\ \phi\}$
    - `Axiom` $CTS : \forall\ \phi :\ BF,\ ctsT\ \phi.$
- Plausible

$$\varphi\ \sigma := \begin{cases} true : & \forall n,\ \sigma\ n = true \\ false : & otherwise \end{cases} \quad : (nat \to bool) \to Prop$$

- Denotational completeness by recursion on the modulus of continuity given by CTS.

- Drastic solution:
  - $ctsT\ \phi\ :=\ \{n : nat \mid cts_n\ \phi\}$
  - $\texttt{Axiom}\ CTS : \forall\ \phi :\ BF,\ ctsT\ \phi.$
- Plausible

$$\varphi\ \sigma := \begin{cases} true : & \forall n,\ \sigma\ n = true \\ false : & otherwise \end{cases} \quad : (nat \rightarrow bool) \rightarrow \color{red}{Prop}$$

- Denotational completeness by recursion on the modulus of continuity given by CTS.
- CTS inconsistent with $CDP := \forall P : Prop,\ \{P\} + \{\neg P\}$

$$CTS \rightarrow CDP \rightarrow False$$

Version 3: Using the Axiom of Continuity
Core Result

- Direct proof of

$$\forall t_1 t_2 : SPT,\ t_1 \neq t_2 \rightarrow [\![t_1]\!] \neq [\![t_2]\!]$$

via `induction` is **HUGE** : 9 cases!

- Direct proof of

$$\forall t_1 t_2 : SPT,\ t_1 \neq t_2 \rightarrow [\![t_1]\!] \not\equiv [\![t_2]\!]$$

  via `induction` is **HUGE** : 9 cases!

- Use *size induction* on pairs of decision trees.
  - Divide proof into *case analysis*
    - $|t_1| + |t_2| = 0 \rightarrow (t_1 = \top \vee t_1 = \bot) \wedge (t_2 = \top \vee t_2 = \bot)$.
    - $|t_1| + |t_2| = m + 1 \rightarrow \begin{cases} t_1 = (n, t, t') & \wedge\ n \notin t_2 \\ n \notin t_1 & \wedge\ t_2 = (n, t, t') \\ t_1 = (n, t_1', t_1'') & \wedge\ t_2 = (n, t_2', t_2'') \end{cases}$

  and *main proof*:
    - $|t_1| + |t_2| = 0$ case is trivial.
    - $|t_1| + |t_2| = m + 1$ cases (3) of moderate difficulty.

Version 4: Using the Axiom of Description

- CTS too drastic.

Version 4: Using the Axiom of Description

- *CTS* too drastic.
- Restriction to continuous boolean functions

$$BF_{cts} \;:=\; \{\phi : BF \mid cts\ \phi\}$$

- Equivalence on BF_{cts} := Equivalence of underlying functions.

Version 4: Using the Axiom of Description

- CTS too drastic.
- Restriction to continuous boolean functions

$$BF_{cts} \;:=\; \{\phi : BF \mid cts\ \phi\}$$

- Equivalence on BF_{cts} := Equivalence of underlying functions.
- Prove denotational completeness as proposition

$$\forall \phi : BF_{cts},\ \exists t : SPT,\ [\![t]\!] \equiv \phi$$

Version 4: Using the Axiom of Description

- CTS too drastic.
- Restriction to continuous boolean functions

$$BF_{cts} \;:=\; \{\phi : BF \mid cts\ \phi\}$$

- Equivalence on BF_{cts} := Equivalence of underlying functions.
- Prove denotational completeness as proposition

$$\forall \phi : BF_{cts},\ \exists t : SPT,\ [\![t]\!] \equiv \phi$$

- Prove core result like before

$$\forall t_1 t_2 : SPT,\ t_1 \neq t_2 \to [\![t_1]\!] \not\equiv [\![t_2]\!]$$

- From denotational completeness and core result derive that there is a unique SPT for every BF_{cts}:

$$\forall \phi : BF_{cts},\ \exists! t : SPT,\ [\![t]\!] \equiv \phi$$

Version 4: Using the Axiom of Description

A morphism from $(BF_{cts}, \equiv)$ to (SPT,=)

- From denotational completeness and core result derive that there is a unique *SPT* for every BF_{cts}:

$$\forall \phi : BF_{cts},\ \exists! t : SPT,\ [\![t]\!] \equiv \phi$$

- Turn this proof into a mapping using *Axiom of Description*:

$$\forall (T : Type)(P : T \to Prop),\ (\exists! t : T, P\ t) \to \{t : T \mid P\ t\}$$

- We want to use $[\![\cdot]\!]$

Version 4: Using the Axiom of Description

A morphism from (SPT,=) to (BF_{cts}, $\equiv$)

- We want to use $[\![\cdot]\!]$
- Prove that decision trees describe continuous functions

$$\forall t : SDT,\ \ cts\ [\![t]\!]$$

by writing a function that determines a modulus of continuity

$$\forall t : SDT,\ \ ctsT\ [\![t]\!]$$

- We want to use $[\![\cdot]\!]$
- Prove that decision trees describe continuous functions

$$\forall t : SDT,\ \ cts\ [\![t]\!]$$

by writing a function that determines a modulus of continuity

$$\forall t : SDT,\ \ ctsT\ [\![t]\!]$$

- Modulus of continuity is largest variable in the tree.

- Pair boolean functions with their modulus of continuity.

$$BF_{ctsT} \;:=\; \{\phi : BF \;\&\; cts_T\ \phi\}$$

- Equivalence:

$$(\phi, n) \equiv (\psi, m) \;:=\; \phi \equiv \psi$$

Version 5: The No-Axiom Version

Boolean functions as dependent pairs

- Pair boolean functions with their modulus of continuity.

$$BF_{ctsT} \;:=\; \{\phi : BF \;\&\; cts_T\;\phi\}$$

- Equivalence:

$$(\phi, n) \equiv (\psi, m) \;:=\; \phi \equiv \psi$$

- Denotational Completeness

$$\forall(\phi, n) : BF_{ctsT},\; \{t : SPT \mid [\![t]\!] \equiv \phi\}$$

  by recursion on modulus of continuity n.
- Core result as before.

Version 3,4 and 5: Summary

- No copies!
- More work:
  - Ordering
  - $n \in t$
  - Lemmas relating orderedness and variable occurrences.
- Axioms (Versions 3 and 4)!

Things to improve

- Goal: Canonical representation for boolean functions

Things to improve

- Goal: Canonical representation for boolean functions
- A representative should have the same meaning as the function it describes

Things to improve

- Goal: Canonical representation for boolean functions
- A representative should have the same meaning as the function it describes
- Setoid-isomorphism *not* meaning preserving!

Things to improve

- Definition of *representative* of boolean functions:

$$\left\{ \begin{array}{lcl} T & : & \mathit{Type} \\ =_T & : & T \to T \to \mathit{Prop} \\ [\![\cdot]\!] & : & T \to \mathcal{BF} \\ ER & : & =_T \text{ is equivalence relation} \\ P & : & \forall t_1\, t_2 : T,\; t_1 =_T t_2 \to [\![t_1]\!] \equiv_{\mathcal{BF}} [\![t_2]\!] \end{array} \right\}$$

Things to improve

- Definition of *representative* of boolean functions:

$$\left\{ \begin{array}{lcl} T & : & \mathit{Type} \\ =_T & : & T \to T \to \mathit{Prop} \\ [\![\cdot]\!] & : & T \to \mathcal{BF} \\ ER & : & =_T \ \text{ is equivalence relation} \\ P & : & \forall t_1\, t_2 : T,\ t_1 =_T t_2 \to [\![t_1]\!] \equiv_{\mathcal{BF}} [\![t_2]\!] \end{array} \right\}$$

- *Meaning preserving* morphisms:

$$\left\{ \begin{array}{lcl} \varrho & : & T \to T' \\ EP & : & \forall t_1\, t_2 : T,\ t_1 =_T t_2 \to \varrho\ t_1 =_{T'} \varrho\ t_2 \\ MP & : & \forall t : T,\ [\![t]\!]_T \equiv_{\mathcal{BF}} [\![\varrho\ t]\!]_{T'} \end{array} \right\}$$

Things to improve

- Definition of *representative* of boolean functions:

$$\left\{\begin{array}{lcl} T & : & Type \\ =_T & : & T \to T \to Prop \\ [\![\cdot]\!] & : & T \to \mathcal{BF} \\ ER & : & =_T \ \text{ is equivalence relation} \\ P & : & \forall t_1\, t_2 : T,\ t_1 =_T t_2 \to [\![t_1]\!] \equiv_{\mathcal{BF}} [\![t_2]\!] \end{array}\right\}$$

- *Meaning preserving* morphisms:

$$\left\{\begin{array}{lcl} \varrho & : & T \to T' \\ EP & : & \forall t_1\, t_2 : T,\ t_1 =_T t_2 \to \varrho\ t_1 =_{T'} \varrho\ t_2 \\ MP & : & \forall t : T,\ [\![t]\!]_T \equiv_{\mathcal{BF}} [\![\varrho\ t]\!]_{T'} \end{array}\right\}$$

- Used setoids are representatives, morphisms are meaning preserving

References

[1] Robin Adams. Formalized metatheory with terms represented by an indexed family of types. In Jean-Christophe Filliâtre, Christine Paulin-Mohring, and Benjamin Werner, editors, *Types for Proofs and Programs*, volume 3839 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin / Heidelberg, 2006.

[2] The Coq Proof Assistant. Standard library. `http://coq.inria.fr/stdlib/`.

[3] Gilles Barthe, Venanzio Capretta, and Olivier Pons. Setoids in type theory, 2000.

[4] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer Verlag, 2004. `http://www.labri.fr/publications/l3a/2004/BC04`.

[5] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[6] Adam Chlipala. Certified programming with dependent types. `http://adam.chlipala.net/cpdt/`.

[7] Gert Smolka and Chad E. Brown. Introduction to computational logic lecture notes, 2009.