Relative Definability of Higher-order Functions in Simply Typed Lambda Calculus

Evan Marzion

Challenge

Write a program $P : (\mathbf{B} \to \mathbf{B}) \to \mathbf{B} \to \mathbf{B}$ with the following behavior[1] using only $\mathbf{0}, \mathbf{1}$, and $\mathbf{If} : \mathbf{B} \to \mathbf{B} \to \mathbf{B} \to \mathbf{B}$:

f	Pf
$\lambda x.\mathbf{0}$	$\lambda x.x$
$\lambda x.x$	$\lambda x.\mathbf{0}$
$\neg$	$\lambda x.\mathbf{1}$
$\lambda x.\mathbf{1}$	$\lambda x.x$

Just using $\mathbf{If}, \mathbf{0}$? $\mathbf{If}, \mathbf{1}$?

[1] extensional behavior, so for instance $\lambda x.\mathbf{0}$ and $\lambda x.\mathbf{If}\ x\ \mathbf{0}\ \mathbf{0}$ are identified.

Challenge

Write a program $P : (\mathbf{B} \to \mathbf{B}) \to \mathbf{B} \to \mathbf{B}$ with the following behavior[1] using only $\mathbf{0}, \mathbf{1}$, and $\mathbf{If} : \mathbf{B} \to \mathbf{B} \to \mathbf{B} \to \mathbf{B}$:

f	Pf
$\lambda x.\mathbf{0}$	$\lambda x.x$
$\lambda x.x$	$\lambda x.\mathbf{0}$
$\neg$	$\lambda x.\mathbf{1}$
$\lambda x.\mathbf{1}$	$\lambda x.x$

Just using $\mathbf{If}, \mathbf{0}$? $\mathbf{If}, \mathbf{1}$?

[1] extensional behavior, so for instance $\lambda x.\mathbf{0}$ and $\lambda x.\mathbf{If}\ x\ \mathbf{0}\ \mathbf{0}$ are identified.

We want to study relative definability of boolean constants/functions/functionals in the context of STLC: Given $M_1, \ldots, M_n$, can I express N? Why or why not?

Previous work:

1. Post's work on clones of boolean connectives ($\mathbf{B}^k \to \mathbf{B}$)

2. Pure lambda-definability in the finite type hierarchy (Plotkin, others)

We want to study relative definability of boolean constants/functions/functionals in the context of STLC: Given $M_1, \ldots, M_n$, can I express N? Why or why not?

Previous work:

1. Post's work on clones of boolean connectives ($\mathbf{B}^k \to \mathbf{B}$)

2. Pure lambda-definability in the finite type hierarchy (Plotkin, others)

We want to study relative definability of boolean constants/functions/functionals in the context of STLC: Given $M_1, \ldots, M_n$, can I express N? Why or why not?

Previous work:

1. Post's work on clones of boolean connectives ($\mathbf{B}^k \to \mathbf{B}$)

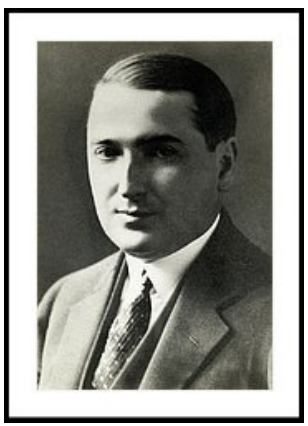
2. Pure lambda-definability in the finite type hierarchy (Plotkin, others)

Clones

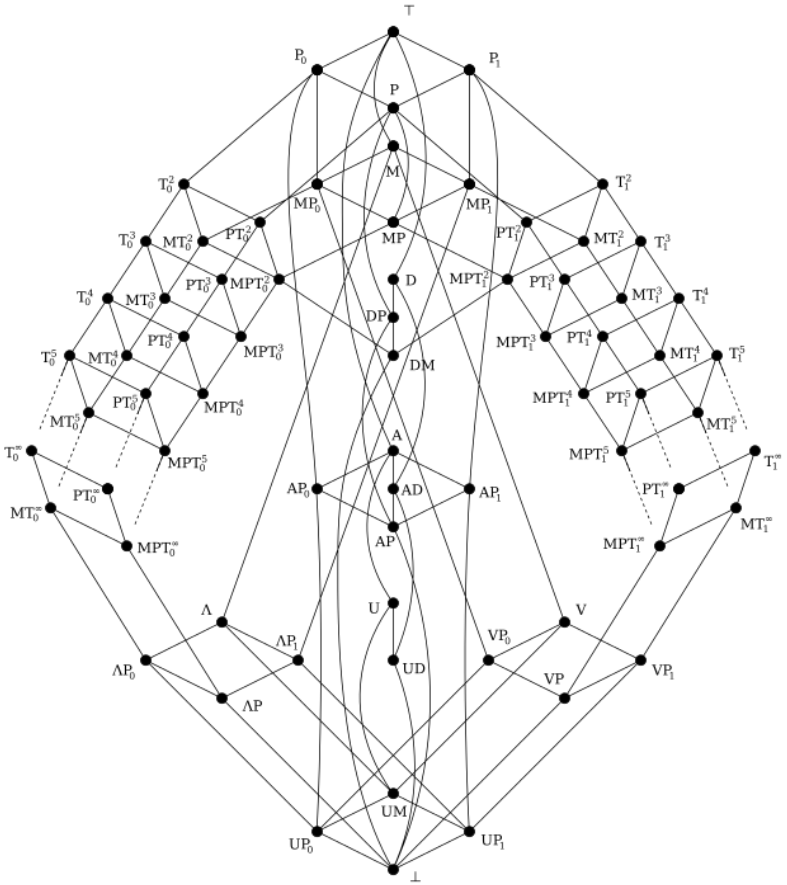
Fix a set X.

A **clone** over X is a set $\mathcal{C}$ of operations $X^k \to X$ containing projections and closed under generalized composition:

$$f, g_1, \ldots, g_k \in \mathcal{C} \Rightarrow \mathbf{x} \mapsto f(g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})) \in \mathcal{C}$$



Emil Post (1941) gives a complete characterization of clones over **B** along with bases.



Five maximal classes:

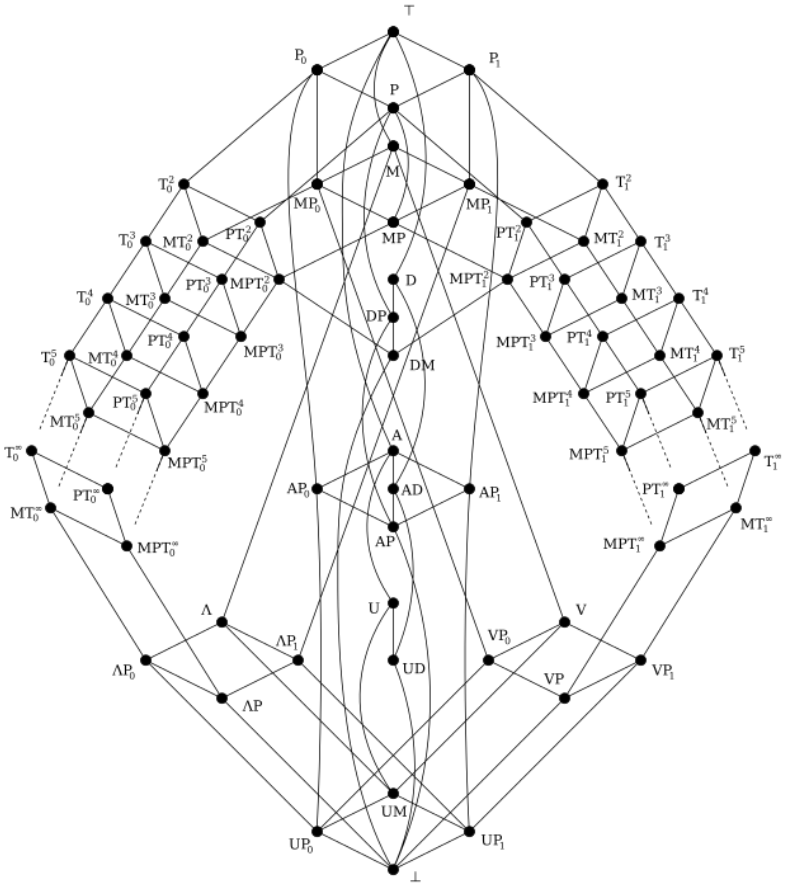
(i) **False-preserving (P_0)**
$f(\mathbf{0},\ldots,\mathbf{0}) = \mathbf{0}$

(ii) **True-preserving (P_1)**
$f(\mathbf{1},\ldots,\mathbf{1}) = \mathbf{1}$

(iii) **Monotone (M)**
$\mathbf{x} \leq \mathbf{y} \Rightarrow f(\mathbf{x}) \leq f(\mathbf{y})$

(iv) **Self-dual (D)**
$f(\neg\mathbf{x}) = \neg f(\mathbf{x})$

(v) **Affine (A)**
$f(\mathbf{x}) = \mathbf{v} \cdot \mathbf{x} \oplus b$



Five maximal classes:

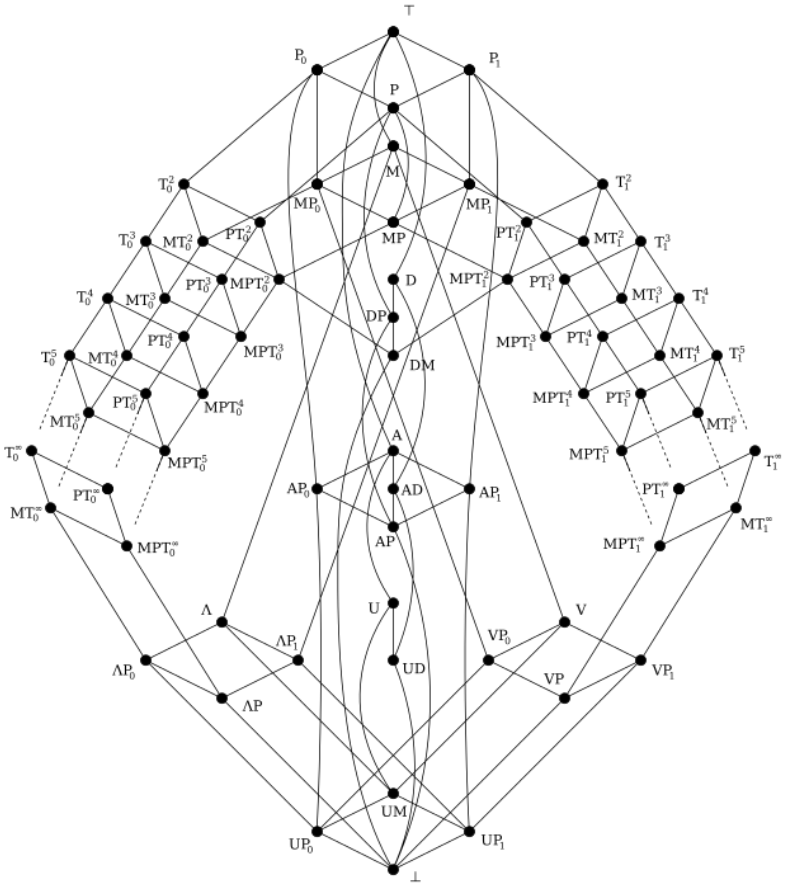
(i) **False-preserving (P_0)**
$f(\mathbf{0},\ldots,\mathbf{0}) = \mathbf{0}$

(ii) **True-preserving (P_1)**
$f(\mathbf{1},\ldots,\mathbf{1}) = \mathbf{1}$

(iii) **Monotone (M)**
$\mathbf{x} \leq \mathbf{y} \Rightarrow f(\mathbf{x}) \leq f(\mathbf{y})$

(iv) **Self-dual (D)**
$f(\neg\mathbf{x}) = \neg f(\mathbf{x})$

(v) **Affine (A)**
$f(\mathbf{x}) = \mathbf{v} \cdot \mathbf{x} \oplus b$



Five maximal classes:

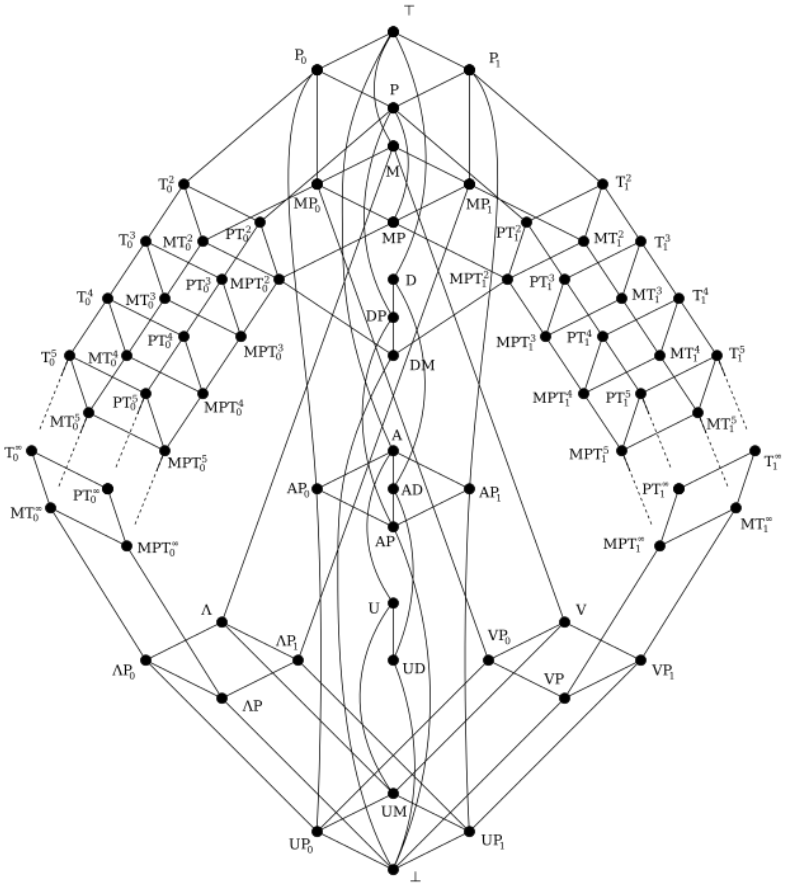
(i) **False-preserving (P_0)**
$f(\mathbf{0},\ldots,\mathbf{0}) = \mathbf{0}$

(ii) **True-preserving (P_1)**
$f(\mathbf{1},\ldots,\mathbf{1}) = \mathbf{1}$

(iii) **Monotone (M)**
$\mathbf{x} \leq \mathbf{y} \Rightarrow f(\mathbf{x}) \leq f(\mathbf{y})$

(iv) **Self-dual (D)**
$f(\neg\mathbf{x}) = \neg f(\mathbf{x})$

(v) **Affine (A)**
$f(\mathbf{x}) = \mathbf{v} \cdot \mathbf{x} \oplus b$



Five maximal classes:

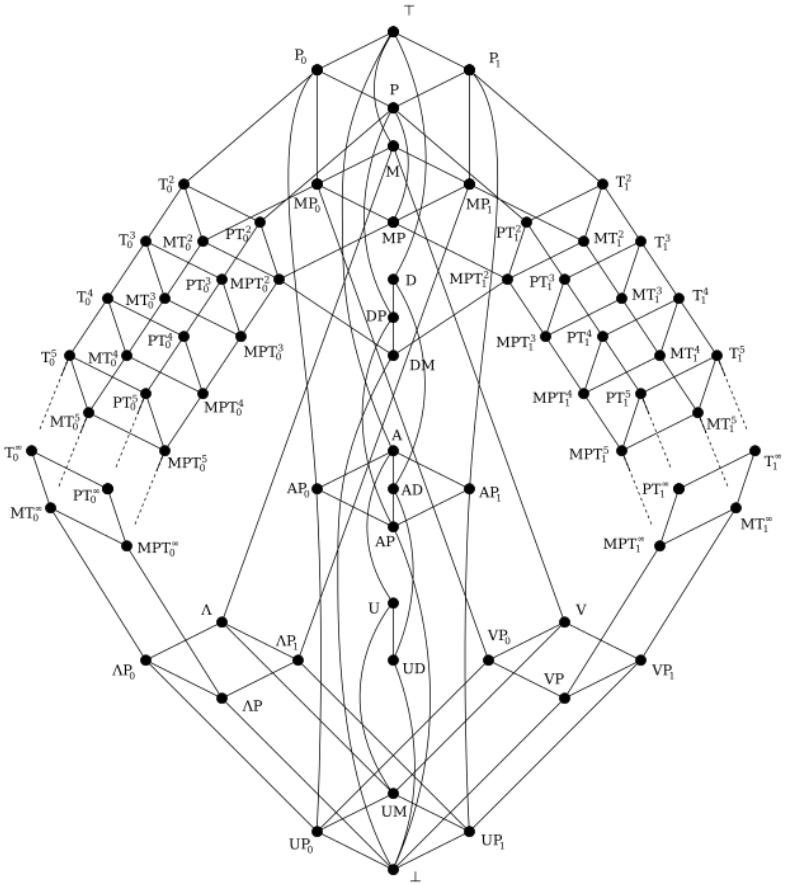
(i) **False-preserving (P_0)**
$f(\mathbf{0},\ldots,\mathbf{0}) = \mathbf{0}$

(ii) **True-preserving (P_1)**
$f(\mathbf{1},\ldots,\mathbf{1}) = \mathbf{1}$

(iii) **Monotone (M)**
$\mathbf{x} \leq \mathbf{y} \Rightarrow f(\mathbf{x}) \leq f(\mathbf{y})$

(iv) **Self-dual (D)**
$f(\neg\mathbf{x}) = \neg f(\mathbf{x})$

(v) **Affine (A)**
$f(\mathbf{x}) = \mathbf{v} \cdot \mathbf{x} \oplus b$



Five maximal classes:

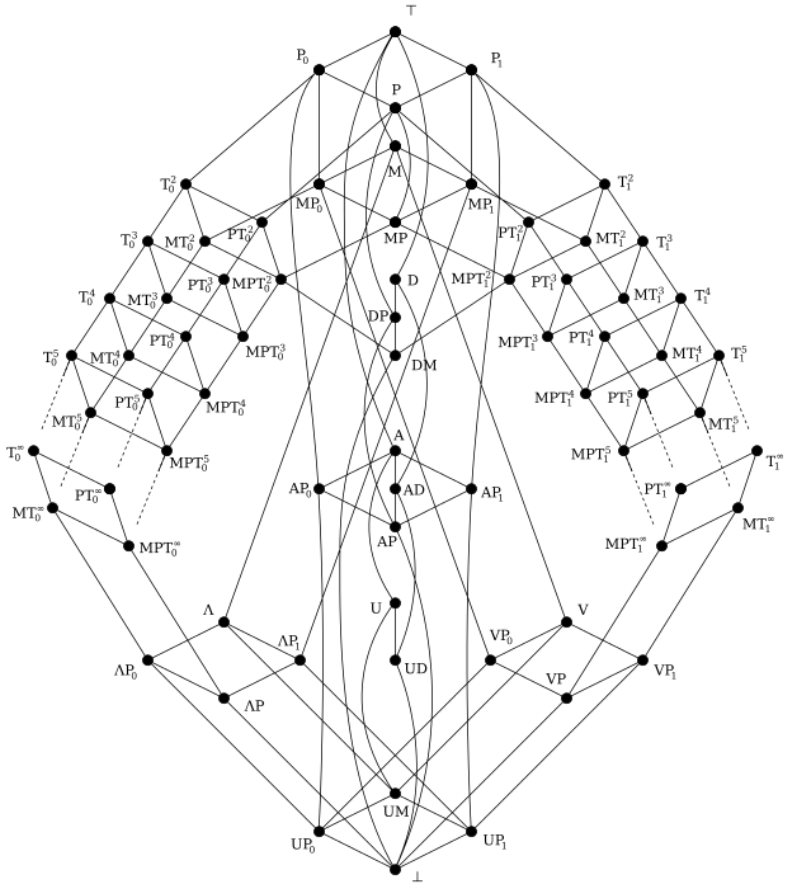
(i) **False-preserving (P_0)**
$f(\mathbf{0},\ldots,\mathbf{0}) = \mathbf{0}$

(ii) **True-preserving (P_1)**
$f(\mathbf{1},\ldots,\mathbf{1}) = \mathbf{1}$

(iii) **Monotone (M)**
$\mathbf{x} \leq \mathbf{y} \Rightarrow f(\mathbf{x}) \leq f(\mathbf{y})$

(iv) **Self-dual (D)**
$f(\neg\mathbf{x}) = \neg f(\mathbf{x})$

(v) **Affine (A)**
$f(\mathbf{x}) = \mathbf{v} \cdot \mathbf{x} \oplus b$



Five maximal classes:

(i) **False-preserving (P_0)**
$f(\mathbf{0},\ldots,\mathbf{0}) = \mathbf{0}$

(ii) **True-preserving (P_1)**
$f(\mathbf{1},\ldots,\mathbf{1}) = \mathbf{1}$

(iii) **Monotone (M)**
$\mathbf{x} \leq \mathbf{y} \Rightarrow f(\mathbf{x}) \leq f(\mathbf{y})$

(iv) **Self-dual (D)**
$f(\neg\mathbf{x}) = \neg f(\mathbf{x})$

(v) **Affine (A)**
$f(\mathbf{x}) = \mathbf{v} \cdot \mathbf{x} \oplus b$

Clones in a higher order setting

Fix a set X. Types: $\mathcal{T} := 0 \mid \sigma \to \tau$

$$X_0 := X$$
$$X_{\sigma\to\tau} := X_\tau^{X_\sigma}$$

A **combinatory clone** over X is a set of operations in X_σ containing all **S** and **K** combinators and closed under application.

By combinatory completeness, we just use lambda notation.

ex. $\vee$ is in the combinatory clone generated by $\wedge, \neg$ since $\vee = \lambda xy.\neg((\neg x) \wedge (\neg y))$.

Clones in a higher order setting

Fix a set X. Types: $\mathcal{T} := 0 \mid \sigma \to \tau$

$$X_0 := X$$
$$X_{\sigma\to\tau} := X_\tau^{X_\sigma}$$

A **combinatory clone** over X is a set of operations in X_σ containing all **S** and **K** combinators and closed under application.

By combinatory completeness, we just use lambda notation.

ex. $\vee$ is in the combinatory clone generated by $\wedge, \neg$ since $\vee = \lambda xy.\neg((\neg x) \wedge (\neg y))$.

Clones and combinatory clones

First-order types: 0, $0 \to 0$, $0 \to 0 \to 0$, etc.
All functions $X^k \to X$ can be thought as having a first-order type (currying).
Clone definability is preserved in combinatory clone setting. Thus, the restriction of a comb. clone to first-order types yields a clone.

Theorem

Let g be comb. clone-definable from $f_1, \ldots, f_n$, all first-order. Then g is already clone-definable from $f_1, \ldots, f_n$.

Proof.

Consider long normal forms. □

Corollary

For every clone $\mathcal{C}$, there is always a comb. clone $\mathcal{G}$ whose restriction is $\mathcal{C}$.

Clones and combinatory clones

First-order types: 0, $0 \to 0$, $0 \to 0 \to 0$, etc.
All functions $X^k \to X$ can be thought as having a first-order type (currying).
Clone definability is preserved in combinatory clone setting. Thus, the restriction of a comb. clone to first-order types yields a clone.

Theorem

Let g be comb. clone-definable from $f_1, \ldots, f_n$, all first-order. Then g is already clone-definable from $f_1, \ldots, f_n$.

Proof.

Consider long normal forms. □

Corollary

For every clone $\mathcal{C}$, there is always a comb. clone $\mathcal{G}$ whose restriction is $\mathcal{C}$.

Clones and combinatory clones

First-order types: 0, $0 \to 0$, $0 \to 0 \to 0$, etc.
All functions $X^k \to X$ can be thought as having a first-order type (currying).
Clone definability is preserved in combinatory clone setting. Thus, the restriction of a comb. clone to first-order types yields a clone.

Theorem

Let g be comb. clone-definable from $f_1, \ldots, f_n$, all first-order. Then g is already clone-definable from $f_1, \ldots, f_n$.

Proof.

Consider long normal forms. □

Corollary

For every clone $\mathcal{C}$, there is always a comb. clone $\mathcal{G}$ whose restriction is $\mathcal{C}$.

Clones and combinatory clones

First-order types: 0, $0 \to 0$, $0 \to 0 \to 0$, etc.
All functions $X^k \to X$ can be thought as having a first-order type (currying).
Clone definability is preserved in combinatory clone setting. Thus, the restriction of a comb. clone to first-order types yields a clone.

Theorem

Let g be comb. clone-definable from $f_1, \ldots, f_n$, all first-order. Then g is already clone-definable from $f_1, \ldots, f_n$.

Proof.

Consider long normal forms. □

Corollary

For every clone $\mathcal{C}$, there is always a comb. clone $\mathcal{G}$ whose restriction is $\mathcal{C}$.

Counterexample

Let $F : (\mathbf{B} \to \mathbf{B}) \to \mathbf{B} \to \mathbf{B}$ denote the following functional:

f	Ff
$\lambda x.\mathbf{0}$	$\lambda x.\mathbf{0}$
$\lambda x.x$	$\lambda x.x$
$\neg$	$\lambda x.\mathbf{0}$
$\lambda x.\mathbf{1}$	$\lambda x.\mathbf{1}$

Lemma

F is not lambda definable.

Proof.

Lnf's of $(0 \to 0) \to 0 \to 0$ are $\lambda fx.f^k x$, none of which represent F.

Claim

The only first-order functions definable from F are projections.

Corollary

There are two distinct comb. clones with the same clone restriction.

Counterexample

Let $F : (\mathbf{B} \to \mathbf{B}) \to \mathbf{B} \to \mathbf{B}$ denote the following functional:

f	Ff
$\lambda x.\mathbf{0}$	$\lambda x.\mathbf{0}$
$\lambda x.x$	$\lambda x.x$
$\neg$	$\lambda x.\mathbf{0}$
$\lambda x.\mathbf{1}$	$\lambda x.\mathbf{1}$

Lemma

F is not lambda definable.

Proof.

Lnf's of $(0 \to 0) \to 0 \to 0$ are $\lambda fx.f^k x$, none of which represent F.

Claim

The only first-order functions definable from F are projections.

Corollary

There are two distinct comb. clones with the same clone restriction.

Counterexample

Let $F : (\mathbf{B} \to \mathbf{B}) \to \mathbf{B} \to \mathbf{B}$ denote the following functional:

f	Ff
$\lambda x.\mathbf{0}$	$\lambda x.\mathbf{0}$
$\lambda x.x$	$\lambda x.x$
$\neg$	$\lambda x.\mathbf{0}$
$\lambda x.\mathbf{1}$	$\lambda x.\mathbf{1}$

Lemma

F is not lambda definable.

Proof.

Lnf's of $(0 \to 0) \to 0 \to 0$ are $\lambda fx.f^k x$, none of which represent F.

Claim

The only first-order functions definable from F are projections.

Corollary

There are two distinct comb. clones with the same clone restriction.

Counterexample

Let $F : (\mathbf{B} \to \mathbf{B}) \to \mathbf{B} \to \mathbf{B}$ denote the following functional:

f	Ff
$\lambda x.\mathbf{0}$	$\lambda x.\mathbf{0}$
$\lambda x.x$	$\lambda x.x$
$\neg$	$\lambda x.\mathbf{0}$
$\lambda x.\mathbf{1}$	$\lambda x.\mathbf{1}$

Lemma

F is not lambda definable.

Proof.

Lnf's of $(0 \to 0) \to 0 \to 0$ are $\lambda fx.f^k x$, none of which represent F.

Claim

The only first-order functions definable from F are projections.

Corollary

There are two distinct comb. clones with the same clone restriction.

Counterexample

Let $F : (\mathbf{B} \to \mathbf{B}) \to \mathbf{B} \to \mathbf{B}$ denote the following functional:

f	Ff
$\lambda x.\mathbf{0}$	$\lambda x.\mathbf{0}$
$\lambda x.x$	$\lambda x.x$
$\neg$	$\lambda x.\mathbf{0}$
$\lambda x.\mathbf{1}$	$\lambda x.\mathbf{1}$

Lemma

F is not lambda definable.

Proof.

Lnf's of $(0 \to 0) \to 0 \to 0$ are $\lambda fx.f^k x$, none of which represent F.

Claim

The only first-order functions definable from F are projections.

Corollary

There are two distinct comb. clones with the same clone restriction.

A brief nitpick

Post excludes constants (0-ary functions) from his classification. Why? Because constants can never be defined from non-constants.

ex. Does the formula $P \wedge \neg P$ *really* define $\mathbf{0}$? In Post's framework this is actually a unary function ($P \mapsto P \wedge \neg P$) which represents a constant function ($\lambda x.\mathbf{0}$).

Issue: $X^k \to X$ is tautology (via C-H) iff $k > 0$.

We can create boring comb. clones from a deductively closed theory in minimal int. logic.

Every type is logically equivalent to either 0 or $0 \to 0$ (in single atom case) so there are only two in our case. ($\mathcal{G}(\mathbf{B})$ and $\mathcal{G}^{taut}(\mathbf{B})$)

Theorem

$\mathcal{G}(\mathbf{B})$ is generated by the first-order elements ($\{\mathbf{If}, \mathbf{0}, \mathbf{1}\}$).

x	fx
s_1	t_1
s_2	t_2
$\vdots$	$\vdots$
s_{N-1}	t_{N-1}
s_N	t_N

Proof.

Induction on type. Key ideas:

(i) $\mathbf{If}_\sigma : \mathbf{B} \to \mathbf{B}_\sigma \to \mathbf{B}_\sigma \to \mathbf{B}_\sigma$ is definable from $\mathbf{If}_0 = \mathbf{If}$

(ii) $\mathbf{Eq}_{\sigma\to\tau} \in \mathbf{B}_{(\sigma\to\tau)\to(\sigma\to\tau)\to 0}$ is definable from the elements of $\mathbf{B}_\sigma$, $\mathbf{Eq}_\tau$, and $\wedge$.

(iii) An arbitrary $f \in \mathbf{B}_{\sigma\to\tau}$ can be defined from elements of $\mathbf{B}_\sigma$, $\mathbf{B}_\tau$, $\mathbf{Eq}_\sigma$, and $\mathbf{If}_\tau$ ("if x equals s_1 then t_1 else if x equals s_2 then t_2 else...").

Theorem

$\mathcal{G}(\mathbf{B})$ is generated by the first-order elements ($\{\mathbf{If}, \mathbf{0}, \mathbf{1}\}$).

x	fx
s_1	t_1
s_2	t_2
$\vdots$	$\vdots$
s_{N-1}	t_{N-1}
s_N	t_N

Proof.

Induction on type. Key ideas:

(i) $\mathbf{If}_\sigma : \mathbf{B} \to \mathbf{B}_\sigma \to \mathbf{B}_\sigma \to \mathbf{B}_\sigma$ is definable from $\mathbf{If}_0 = \mathbf{If}$

(ii) $\mathbf{Eq}_{\sigma\to\tau} \in \mathbf{B}_{(\sigma\to\tau)\to(\sigma\to\tau)\to 0}$ is definable from the elements of $\mathbf{B}_\sigma$, $\mathbf{Eq}_\tau$, and $\wedge$.

(iii) An arbitrary $f \in \mathbf{B}_{\sigma\to\tau}$ can be defined from elements of $\mathbf{B}_\sigma$, $\mathbf{B}_\tau$, $\mathbf{Eq}_\sigma$, and $\mathbf{If}_\tau$ ("if x equals s_1 then t_1 else if x equals s_2 then t_2 else...").

Theorem

$\mathcal{G}(\mathbf{B})$ is generated by the first-order elements ($\{\mathbf{If}, \mathbf{0}, \mathbf{1}\}$).

x	fx
s_1	t_1
s_2	t_2
$\vdots$	$\vdots$
s_{N-1}	t_{N-1}
s_N	t_N

Proof.

Induction on type. Key ideas:

(i) $\mathbf{If}_\sigma : \mathbf{B} \to \mathbf{B}_\sigma \to \mathbf{B}_\sigma \to \mathbf{B}_\sigma$ is definable from $\mathbf{If}_0 = \mathbf{If}$

(ii) $\mathbf{Eq}_{\sigma\to\tau} \in \mathbf{B}_{(\sigma\to\tau)\to(\sigma\to\tau)\to 0}$ is definable from the elements of $\mathbf{B}_\sigma$, $\mathbf{Eq}_\tau$, and $\wedge$.

(iii) An arbitrary $f \in \mathbf{B}_{\sigma\to\tau}$ can be defined from elements of $\mathbf{B}_\sigma$, $\mathbf{B}_\tau$, $\mathbf{Eq}_\sigma$, and $\mathbf{If}_\tau$ ("if x equals s_1 then t_1 else if x equals s_2 then t_2 else...").

Theorem

$\mathcal{G}(\mathbf{B})$ is generated by the first-order elements ($\{\mathbf{If}, \mathbf{0}, \mathbf{1}\}$).

x	fx
s_1	t_1
s_2	t_2
$\vdots$	$\vdots$
s_{N-1}	t_{N-1}
s_N	t_N

Proof.

Induction on type. Key ideas:

(i) $\mathbf{If}_\sigma : \mathbf{B} \to \mathbf{B}_\sigma \to \mathbf{B}_\sigma \to \mathbf{B}_\sigma$ is definable from $\mathbf{If}_0 = \mathbf{If}$

(ii) $\mathbf{Eq}_{\sigma\to\tau} \in \mathbf{B}_{(\sigma\to\tau)\to(\sigma\to\tau)\to 0}$ is definable from the elements of $\mathbf{B}_\sigma$, $\mathbf{Eq}_\tau$, and $\wedge$.

(iii) An arbitrary $f \in \mathbf{B}_{\sigma\to\tau}$ can be defined from elements of $\mathbf{B}_\sigma$, $\mathbf{B}_\tau$, $\mathbf{Eq}_\sigma$, and $\mathbf{If}_\tau$ ("if x equals s_1 then t_1 else if x equals s_2 then t_2 else...").

Theorem

$\mathcal{G}(\mathbf{B})$ is generated by the first-order elements ($\{\mathbf{If}, \mathbf{0}, \mathbf{1}\}$).

x	fx
s_1	t_1
s_2	t_2
$\vdots$	$\vdots$
s_{N-1}	t_{N-1}
s_N	t_N

Proof.

Induction on type. Key ideas:

(i) $\mathbf{If}_\sigma : \mathbf{B} \to \mathbf{B}_\sigma \to \mathbf{B}_\sigma \to \mathbf{B}_\sigma$ is definable from $\mathbf{If}_0 = \mathbf{If}$

(ii) $\mathbf{Eq}_{\sigma\to\tau} \in \mathbf{B}_{(\sigma\to\tau)\to(\sigma\to\tau)\to 0}$ is definable from the elements of $\mathbf{B}_\sigma$, $\mathbf{Eq}_\tau$, and $\wedge$.

(iii) An arbitrary $f \in \mathbf{B}_{\sigma\to\tau}$ can be defined from elements of $\mathbf{B}_\sigma$, $\mathbf{B}_\tau$, $\mathbf{Eq}_\sigma$, and $\mathbf{If}_\tau$ ("if x equals s_1 then t_1 else if x equals s_2 then t_2 else...").

Generalizing true-preserving

Is there a maximal comb. clone which corresponds to true-preserving functions? Yes!

$$\mathbf{TP}_0 := \{\mathbf{1}\}$$
$$\mathbf{TP}_{\sigma\to\tau} := \{f : \mathbf{B}_\sigma \to \mathbf{B}_\tau \mid \forall x \in \mathbf{TP}_\sigma . fx \in \mathbf{TP}_\tau\}$$

$\mathbf{FP}_\sigma$ is similar defined in false-preserving manner.

Claim

TP *is a maximal comb. clone and furthermore contains all true-preserving (in the old sense) first-order functions.*

Theorem

TP *is generated by its first-order elements* ($\{\mathbf{If}, \mathbf{1}\}$).

Proof.

Idea: Mimic the proof for $\mathcal{G}(\mathbf{B})$. Two issues:

First, $\mathbf{Eq}_\sigma$ is generally not in $\mathbf{TP}$: consider s, s' distinct in $\mathbf{TP}_\sigma$, then $\mathbf{Eq}_\sigma ss' = \mathbf{0}$.

Solution: Augment our domain of "truth values" from $\mathbf{B}$ to $\mathbf{B} \to \mathbf{B}$, treating $\lambda x.\mathbf{1}$ as true and $\lambda x.\mathbf{0}, \lambda x.x$ as false (ignore $\neg$). We can then define an equality operator which is correct w.r.t. these truth values. We now need to modify our logical branching operations, but it all works out.

Second, the induction hypothesis won't give us all elements as before. How do we test equality on some u_i non-$\mathbf{TP}$ if we don't already have an expression for it?

Proof.

Idea: Mimic the proof for $\mathcal{G}(\mathbf{B})$. Two issues:

First, $\mathbf{Eq}_\sigma$ is generally not in $\mathbf{TP}$: consider s, s' distinct in $\mathbf{TP}_\sigma$, then $\mathbf{Eq}_\sigma s s' = \mathbf{0}$.

Solution: Augment our domain of "truth values" from $\mathbf{B}$ to $\mathbf{B} \to \mathbf{B}$, treating $\lambda x.\mathbf{1}$ as true and $\lambda x.\mathbf{0}, \lambda x.x$ as false (ignore $\neg$). We can then define an equality operator which is correct w.r.t. these truth values. We now need to modify our logical branching operations, but it all works out.

Second, the induction hypothesis won't give us all elements as before. How do we test equality on some u_i non-$\mathbf{TP}$ if we don't already have an expression for it?

Proof (con't).

Solution: We *almost* have an expression for it. By virtue of previous theorem, $\{\mathbf{If}, \mathbf{0}, \mathbf{1}\}$ is a basis for $\mathcal{G}(\mathbf{B})$.

For any $s \in \mathbf{B}_\sigma$ there is a $T_s \in \mathbf{TP}_{0\to\sigma}$ such that $T_s\mathbf{0} = s$ (abstract away all instances of $\mathbf{0}$ in the expression for s.

Can we guarantee that $\mathbf{0}$ can be recovered from a non-$\mathbf{TP}$ element? Indicator function for $\mathbf{TP}_\sigma$: $\mathbf{IsTP}_\sigma : \mathbf{B}_\sigma \to \mathbf{B}$.

$$\mathbf{IsTP}_0 = \lambda x.x$$

$$\mathbf{IsTP}_{\sigma\to\tau} = \lambda f. \bigwedge_{i=1}^{N} \mathbf{IsTP}_\tau(fs_i) \quad (\mathbf{TP}_\sigma = \{s_1, \ldots, s_N\})$$

Proof (con't).

Solution: We *almost* have an expression for it. By virtue of previous theorem, $\{\mathbf{If}, \mathbf{0}, \mathbf{1}\}$ is a basis for $\mathcal{G}(\mathbf{B})$.

For any $s \in \mathbf{B}_\sigma$ there is a $T_s \in \mathbf{TP}_{0\to\sigma}$ such that $T_s\mathbf{0} = s$ (abstract away all instances of $\mathbf{0}$ in the expression for s.

Can we guarantee that $\mathbf{0}$ can be recovered from a non-$\mathbf{TP}$ element? Indicator function for $\mathbf{TP}_\sigma$: $\mathbf{IsTP}_\sigma : \mathbf{B}_\sigma \to \mathbf{B}$.

$$\mathbf{IsTP}_0 = \lambda x.x$$

$$\mathbf{IsTP}_{\sigma\to\tau} = \lambda f. \bigwedge_{i=1}^{N} \mathbf{IsTP}_\tau(fs_i) \quad (\mathbf{TP}_\sigma = \{s_1, \ldots, s_N\})$$

Proof (con't).

Solution: We *almost* have an expression for it. By virtue of previous theorem, $\{\mathbf{If}, \mathbf{0}, \mathbf{1}\}$ is a basis for $\mathcal{G}(\mathbf{B})$.

For any $s \in \mathbf{B}_\sigma$ there is a $T_s \in \mathbf{TP}_{0\to\sigma}$ such that $T_s\mathbf{0} = s$ (abstract away all instances of $\mathbf{0}$ in the expression for s.

Can we guarantee that $\mathbf{0}$ can be recovered from a non-$\mathbf{TP}$ element? Indicator function for $\mathbf{TP}_\sigma$: $\mathbf{IsTP}_\sigma : \mathbf{B}_\sigma \to \mathbf{B}$.

$$\mathbf{IsTP}_0 = \lambda x.x$$

$$\mathbf{IsTP}_{\sigma\to\tau} = \lambda f. \bigwedge_{i=1}^{N} \mathbf{IsTP}_\tau(fs_i) \quad (\mathbf{TP}_\sigma = \{s_1, \ldots, s_N\})$$

Proof (con't).

Solution: We *almost* have an expression for it. By virtue of previous theorem, $\{\mathbf{If}, \mathbf{0}, \mathbf{1}\}$ is a basis for $\mathcal{G}(\mathbf{B})$.

For any $s \in \mathbf{B}_\sigma$ there is a $T_s \in \mathbf{TP}_{0\to\sigma}$ such that $T_s\mathbf{0} = s$ (abstract away all instances of $\mathbf{0}$ in the expression for s.

Can we guarantee that $\mathbf{0}$ can be recovered from a non-$\mathbf{TP}$ element? Indicator function for $\mathbf{TP}_\sigma$: $\mathbf{IsTP}_\sigma : \mathbf{B}_\sigma \to \mathbf{B}$.

$$\mathbf{IsTP}_0 = \lambda x.x$$

$$\mathbf{IsTP}_{\sigma\to\tau} = \lambda f. \bigwedge_{i=1}^{N} \mathbf{IsTP}_\tau(fs_i) \quad (\mathbf{TP}_\sigma = \{s_1, \ldots, s_N\})$$

Challenge revisited

From previous theorem, $\{\mathbf{If}, \mathbf{1}\}$ is a basis for **TP** and analogously $\{\mathbf{If}, \mathbf{0}\}$ is a basis for **FP**.

Recall the following functional from beginning of talk:

f	Pf
$\lambda x.\mathbf{0}$	$\lambda x.x$
$\lambda x.x$	$\lambda x.\mathbf{0}$
$\neg$	$\lambda x.\mathbf{1}$
$\lambda x.\mathbf{1}$	$\lambda x.x$

Since P is false-preserving, you **can** define it using just $\mathbf{If}, \mathbf{0}$.

Since P is **not** true-preserving, you **cannot** define it from $\mathbf{If}, \mathbf{1}$ alone.

Challenge revisited

From previous theorem, $\{\mathbf{If}, \mathbf{1}\}$ is a basis for **TP** and analogously $\{\mathbf{If}, \mathbf{0}\}$ is a basis for **FP**.

Recall the following functional from beginning of talk:

f	Pf
$\lambda x.\mathbf{0}$	$\lambda x.x$
$\lambda x.x$	$\lambda x.\mathbf{0}$
$\neg$	$\lambda x.\mathbf{1}$
$\lambda x.\mathbf{1}$	$\lambda x.x$

Since P is false-preserving, you **can** define it using just $\mathbf{If}, \mathbf{0}$.

Since P is **not** true-preserving, you **cannot** define it from $\mathbf{If}, \mathbf{1}$ alone.

Challenge revisited

From previous theorem, $\{\mathbf{If}, \mathbf{1}\}$ is a basis for **TP** and analogously $\{\mathbf{If}, \mathbf{0}\}$ is a basis for **FP**.

Recall the following functional from beginning of talk:

f	Pf
$\lambda x.\mathbf{0}$	$\lambda x.x$
$\lambda x.x$	$\lambda x.\mathbf{0}$
$\neg$	$\lambda x.\mathbf{1}$
$\lambda x.\mathbf{1}$	$\lambda x.x$

Since P is false-preserving, you **can** define it using just $\mathbf{If}, \mathbf{0}$.

Since P is **not** true-preserving, you **cannot** define it from $\mathbf{If}, \mathbf{1}$ alone.

Challenge revisited

From previous theorem, $\{\mathbf{If}, \mathbf{1}\}$ is a basis for **TP** and analogously $\{\mathbf{If}, \mathbf{0}\}$ is a basis for **FP**.

Recall the following functional from beginning of talk:

f	Pf
$\lambda x.\mathbf{0}$	$\lambda x.x$
$\lambda x.x$	$\lambda x.\mathbf{0}$
$\neg$	$\lambda x.\mathbf{1}$
$\lambda x.\mathbf{1}$	$\lambda x.x$

Since P is false-preserving, you **can** define it using just $\mathbf{If}, \mathbf{0}$.

Since P is **not** true-preserving, you **cannot** define it from $\mathbf{If}, \mathbf{1}$ alone.

An easy corollary for false- and true-preserving functions

Comb. clones are closed under intersections. Thus, let $\mathbf{FTP} := \mathbf{FP} \cap \mathbf{TP}$.

Lemma

Let $s \in \mathbf{B}_\sigma$ be false- and true-preserving. Then σ is a tautology.

Proof.

Suppose not. Then there is a lambda def. $S : \sigma \to 0$, meaning that $Ss \in \mathbf{B}$ is both false- and true-preserving. □

Theorem

FTP *is generated by its firstorder elements ($\{\mathbf{If}\}$).*

An easy corollary for false- and true-preserving functions

Comb. clones are closed under intersections. Thus, let $\mathbf{FTP} := \mathbf{FP} \cap \mathbf{TP}$.

Lemma

Let $s \in \mathbf{B}_\sigma$ be false- and true-preserving. Then σ is a tautology.

Proof.

Suppose not. Then there is a lambda def. $S : \sigma \to 0$, meaning that $Ss \in \mathbf{B}$ is both false- and true-preserving. □

Theorem

$\mathbf{FTP}$ *is generated by its firstorder elements* ($\{\mathbf{If}\}$).

An easy corollary for false- and true-preserving functions

Comb. clones are closed under intersections. Thus, let $\mathbf{FTP} := \mathbf{FP} \cap \mathbf{TP}$.

Lemma

Let $s \in \mathbf{B}_\sigma$ be false- and true-preserving. Then σ is a tautology.

Proof.

Suppose not. Then there is a lambda def. $S : \sigma \to 0$, meaning that $Ss \in \mathbf{B}$ is both false- and true-preserving. □

Theorem

FTP *is generated by its firstorder elements* ($\{\mathbf{If}\}$).

An easy corollary for false- and true-preserving functions

Comb. clones are closed under intersections. Thus, let $\mathbf{FTP} := \mathbf{FP} \cap \mathbf{TP}$.

Lemma

Let $s \in \mathbf{B}_\sigma$ be false- and true-preserving. Then σ is a tautology.

Proof.

Suppose not. Then there is a lambda def. $S : \sigma \to 0$, meaning that $Ss \in \mathbf{B}$ is both false- and true-preserving. □

Theorem

FTP *is generated by its firstorder elements ($\{\mathbf{If}\}$).*

Proof.

Let $f \in \mathbf{FTP}_{\sigma_1 \to \ldots \to \sigma_N \to 0}$. By previous lemma there is some i and lambda def. $S : \sigma_i \to 0$.

Let $f = M\mathbf{If1}$ and $f = N\mathbf{If0}$ for lambda def. M, N. Claim:

$$f = \lambda x_1 \ldots x_N.\mathbf{If}(Sx_i)(M\mathbf{If}(Sx_i)\bar{x})(N\mathbf{If}(Sx_i)\bar{x}).$$

For arbitrary $s_1, \ldots, s_N$ we have that

$$\begin{aligned}(\lambda \bar{x}.\mathbf{If}(Sx_i)(M\mathbf{If}(Sx_i)\bar{x})(N\mathbf{If}(Sx_i)\bar{x}))\bar{s} &= \mathbf{If}(Ss_i)(M\mathbf{If}(Ss_i)\bar{s})(N\mathbf{If}(Ss_i)\bar{s}) \\ &= \begin{cases} M\mathbf{If1}\bar{s} & Ss_i = \mathbf{1} \\ N\mathbf{If0}\bar{s} & Ss_i = \mathbf{0} \end{cases} \\ &= \begin{cases} f\bar{s} & Ss_i = \mathbf{1} \\ f\bar{s} & Ss_i = \mathbf{0} \end{cases} \\ &= f\bar{s}.\end{aligned}$$

□

Generalizing self-duality

Intuition: dual of an element is that element with the roles of $\mathbf{0}, \mathbf{1}$ reversed.

$$d_0 x := \neg x$$
$$d_{\sigma\to\tau} f := \lambda x. d_\tau (f (d_\sigma x))$$

ex. Let $f : \mathbf{B} \to \mathbf{B} \to \mathbf{B}$.

$$\begin{aligned} d_{0\to 0\to 0} f &= \lambda x. d_{0\to 0}(f(d_0 x)) \\ &= \lambda xy. d_0(f(d_0 x)(d_0 y)) \\ &= \lambda xy. \neg(f(\neg x)(\neg y)) \end{aligned}$$

Let $\mathbf{SD}_\sigma := \{x \in \mathbf{B}_\sigma \mid d_\sigma x = x\}$.

Claim

SD *is a maximal comb. clone w.r.t.* $\mathcal{G}^{taut}(\mathbf{B})$ *and furthermore contains all self-dual (in the old sense) first-order functions.*

Claim

SD *is generated by its first-order elements* ($\{\neg, \mathbf{Maj}\}$).

Proof uses same ideas from **TP** proof (reinterpret logical branching operations, use expressions that "almost" define terms) but trickier.

Let $\mathbf{DP} := \mathbf{FTP} \cap \mathbf{SD}$.

Claim

DP *is also generated by its first-order elements* ($\{\mathbf{Maj}, \oplus_3\}$).

Claim

SD *is generated by its first-order elements* ($\{\neg, \mathbf{Maj}\}$).

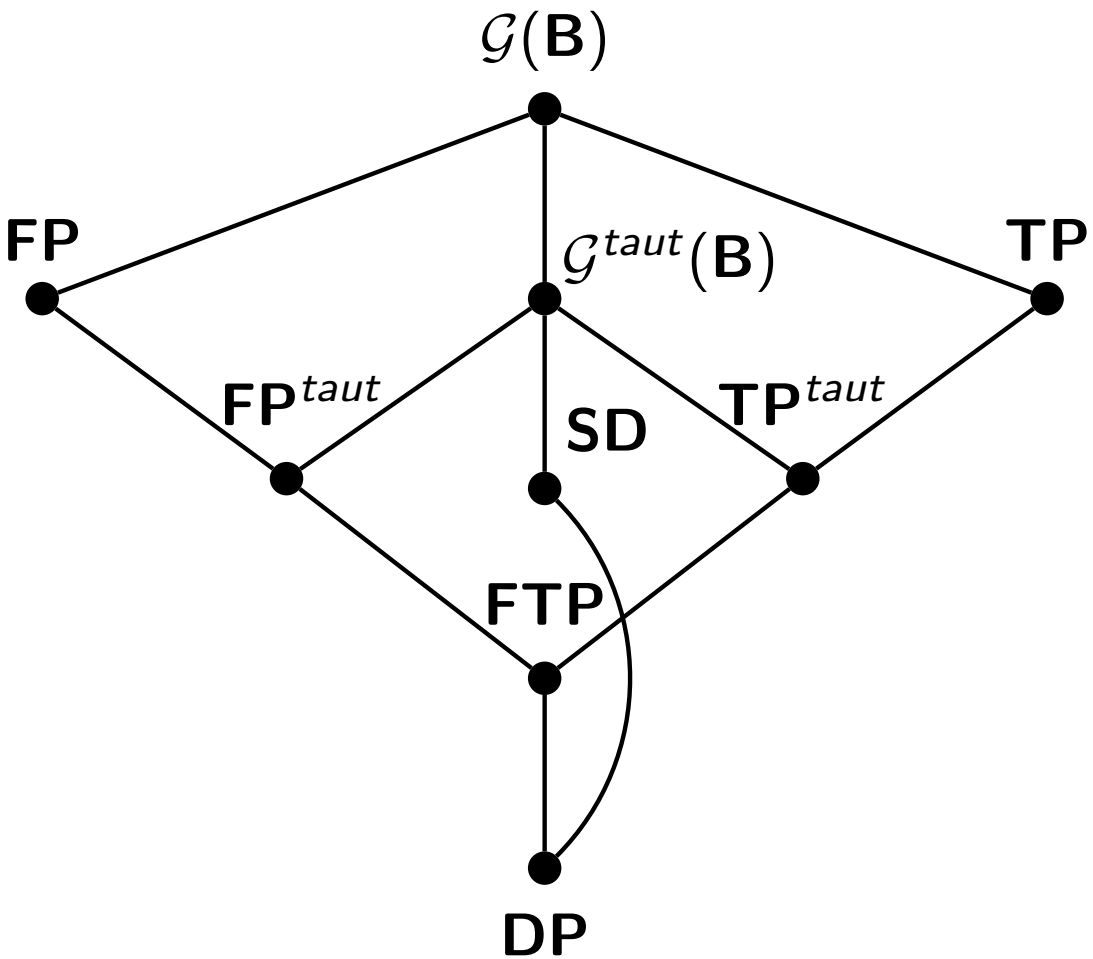
Proof uses same ideas from **TP** proof (reinterpret logical branching operations, use expressions that "almost" define terms) but trickier.

Let $\mathbf{DP} := \mathbf{FTP} \cap \mathbf{SD}$.

Claim

DP *is also generated by its first-order elements* ($\{\mathbf{Maj}, \oplus_3\}$).

Summary



$(\mathbf{FP}^{taut} := \mathbf{FP} \cap \mathcal{G}^{taut}(\mathbf{B})$ and $\mathbf{TP}^{taut} := \mathbf{TP} \cap \mathcal{G}^{taut}(\mathbf{B}))$

The not so nice stuff

The proper generalization of the monotone functions is tricky:

$$\begin{aligned}
\mathbf{Mon}_0 &:= \{\mathbf{0}, \mathbf{1}\} \\
\leq_0 &:= \leq \\
\mathbf{Mon}_{\sigma\to\tau} &:= \{f \mid \forall s \in \mathbf{Mon}_\sigma . fs \in \mathbf{Mon}_\tau \ \& \\
&\qquad \forall s, s' \in \mathbf{Mon}_\sigma, s \leq_\sigma s \ \Rightarrow fs \leq_\tau fs'\}. \\
f \leq_{\sigma\to\tau} g &:= \forall s \in \mathbf{Mon}_\sigma . fs \leq_\tau gs.
\end{aligned}$$

Claim

Mon *is not generated by its first-order elements (*$\mathbf{isMon}_{0\to 0}$ *is a counterexample).*

Little is known about how to generalize the affine functions.

The not so nice stuff

The proper generalization of the monotone functions is tricky:

$$\begin{aligned}
\mathbf{Mon}_0 &:= \{\mathbf{0}, \mathbf{1}\} \\
\leq_0 &:= \leq \\
\mathbf{Mon}_{\sigma\to\tau} &:= \{f \mid \forall s \in \mathbf{Mon}_\sigma . fs \in \mathbf{Mon}_\tau \ \& \\
&\qquad \forall s, s' \in \mathbf{Mon}_\sigma, s \leq_\sigma s \ \Rightarrow fs \leq_\tau fs'\}. \\
f \leq_{\sigma\to\tau} g &:= \forall s \in \mathbf{Mon}_\sigma . fs \leq_\tau gs.
\end{aligned}$$

Claim

Mon *is not generated by its first-order elements (*$\mathbf{isMon}_{0\to 0}$ *is a counterexample).*

Little is known about how to generalize the affine functions.

The not so nice stuff

The proper generalization of the monotone functions is tricky:

$$\begin{aligned}
\mathbf{Mon}_0 &:= \{\mathbf{0}, \mathbf{1}\} \\
\leq_0 &:= \leq \\
\mathbf{Mon}_{\sigma\to\tau} &:= \{f \mid \forall s \in \mathbf{Mon}_\sigma . fs \in \mathbf{Mon}_\tau \ \& \\
&\qquad \forall s, s' \in \mathbf{Mon}_\sigma, s \leq_\sigma s \ \Rightarrow fs \leq_\tau fs'\}. \\
f \leq_{\sigma\to\tau} g &:= \forall s \in \mathbf{Mon}_\sigma . fs \leq_\tau gs.
\end{aligned}$$

Claim

Mon *is not generated by its first-order elements (*$\mathbf{isMon}_{0\to 0}$ *is a counterexample).*

Little is known about how to generalize the affine functions.

The not so nice stuff

The proper generalization of the monotone functions is tricky:

$$\begin{aligned}
\mathbf{Mon}_0 &:= \{\mathbf{0}, \mathbf{1}\} \\
\leq_0 &:= \leq \\
\mathbf{Mon}_{\sigma\to\tau} &:= \{f \mid \forall s \in \mathbf{Mon}_\sigma . fs \in \mathbf{Mon}_\tau \ \& \\
&\qquad \forall s, s' \in \mathbf{Mon}_\sigma, s \leq_\sigma s \ \Rightarrow fs \leq_\tau fs'\}. \\
f \leq_{\sigma\to\tau} g &:= \forall s \in \mathbf{Mon}_\sigma . fs \leq_\tau gs.
\end{aligned}$$

Claim

Mon *is not generated by its first-order elements (*$\mathbf{isMon}_{0\to 0}$ *is a counterexample).*

Little is known about how to generalize the affine functions.

Thank you for your attention.