Theorem Proving in Logic with Partial Functions

Wrocław, 6 december 2013

Hans de Nivelle

University of Wrocław, Poland

Partial Functions

We are surrounded by partial functions:

$$\forall x{:}\,\mathrm{Real}\ (\sqrt{x})^2 = x,$$

$$\forall L{:}\,\mathrm{List}\ \ \mathrm{cons}(\mathrm{first}(L), \mathrm{rest}(L)) = L,$$

$$\forall m, n{:}\,\mathrm{Nat}\ 0 \leq (m \bmod n) \wedge (m \bmod n) < |n|.$$

Can be handled by relativization:

$$\forall x{:}\,\mathrm{Real}\;\; x \geq 0 \rightarrow (\sqrt{x})^2 = x,$$

$$\forall L{:}\,\mathrm{List}\;\; L \neq \mathrm{nil} \rightarrow \mathrm{cons}(\mathrm{first}(L), \mathrm{rest}(L)) = L,$$

$$\forall m, n{:}\,\mathrm{Nat}\;\; n \neq 0 \rightarrow 0 \leq (m \bmod n) \wedge (m \bmod n) < |n|.$$

One can also relativize the types:

$$\forall x \ \mathrm{Real}(x) \wedge x \geq 0 \rightarrow (\sqrt{x})^2 = x,$$

$$\forall x \ \mathrm{List}(x) \wedge L \neq \mathrm{nil} \rightarrow \mathrm{cons}(\mathrm{first}(L), \mathrm{rest}(L)) = L,$$

$$\forall m, n \ \mathrm{Nat}(m) \wedge \mathrm{Nat}(n) \wedge n \neq 0 \rightarrow 0 \leq (m \bmod n) \wedge (m \bmod n) < |n|.$$

Is it really the same?

Yes, but only because the formulas are correct.

When formulas are not correct, their relativizations have meanings that are either too weak or too strong.

Built-in (to the logic) typing ensures that a formula becomes unusable when the typing rules are not respected. With relativization, strictness is lost.

If you are certain that all formulas that you use are correct, then you don't need type checking,

or more precisely:

When a formula has been typechecked, its types can be replaced by relativizations without changing the meaning.

Putting Preconditions in the Types?

For partial functions, one wants the same strictness as simple types. Include preconditions in the types:

$$\forall x{:}\,\mathrm{Real} \geq 0 \;\; (\sqrt{x})^2 = x,$$

$$\forall L{:}\,\mathrm{List} \neq \mathrm{nil} \;\; \mathrm{cons}(\mathrm{first}(L), \mathrm{rest}(L)) = L,$$

$$\forall m{:}\,\mathrm{Nat}\;\; n{:}\,\mathrm{Nat} \neq 0 \;\; 0 \leq (m \bmod n) \wedge (m \bmod n) < |n|.$$

There is no general form of preconditions:

$$\forall m{:}\,\mathrm{Nat}\;\; m \neq 0 \to \exists n{:}\,\mathrm{Nat} \leq m \;\; m - n = 1.$$

PCL

Introduce new truth value **e** for meaningless formulas. Split $\to$ into $\to$ and $[\ ]$. Split $\wedge$ into $\langle\ \rangle$ and $\wedge$.

$$\forall x\ [\ \mathrm{Real}(x) \wedge x > 0\ ](\sqrt{x})^2 = x,$$

$$\forall x\ [\ \mathrm{List}(L) \wedge L \neq \mathrm{nil}\ ]\ \mathrm{cons}(\mathrm{first}(L), \mathrm{rest}(L)) = L,$$

$$\forall m, n[\ \mathrm{Nat}(m) \wedge \mathrm{Nat}(n)\ ][\ n \neq 0\ ]\ 0 \leq (m \bmod n) \wedge (m \bmod n) < |n|,$$

$$\forall m[\ \mathrm{Nat}(m)\ ]\ m \neq 0 \to \exists n\langle\ \mathrm{Nat}(n)\ \rangle\ \langle\ m \geq n\ \rangle\ m - n = 1.$$

PCL-operators

$\neg$: Argument must be well-typed. For the rest, semantics is as usual.

$\rightarrow, \wedge, \vee, \leftrightarrow$: Both arguments must be well-typed. For the rest, semantics is as usual.

$[A]B$: First formula A must be well-typed. If A is true, then second formula must be well-typed. If $[A]B$ is well-typed, it means the same as $A \rightarrow B$.

$\langle A \rangle B$: First formula A must be well-typed. If A is true, then second formula must be well-typed. If $\langle A \rangle B$ is well-typed, it means the same as $A \wedge B$.

$\forall x \; P(x)$: Every $I_d^x(P(x))$ must be well-typed. For the rest, semantics is as usual.

$\exists x \; P(x)$: Every $I_d^x(P(x))$ must be well-typed. For the rest, semantics is as usual.

Operators $\wedge, \vee, \rightarrow, \leftrightarrow, \neg, \forall, \exists$ are strict.

Operators [ ] and $\langle\ \rangle$ are lazy.

Equality $=$ is total, i.e. always well-typed. (One can define weaker forms of equality.)

The Prop( ) operator is always well-typed. It is true if its argument is well-typed.

Interpretations

Definition: An interpretation $I = (D, \mathbf{f}, \mathbf{t}, \mathbf{e}, [\ ])$ is defined by:

- A domain D.
- Two distinct truth values $\mathbf{f}$ and $\mathbf{t}$.
- An error value $\mathbf{e}$.
- A function $[\ ]$ that interprets the function symbols: If f is a function symbol with arity n, then $[f]$ is a total function from D^n to D.
- A function $[\ ]$ that interprets the predicate symbols: If p is a predicate symbol with arity n, then $[p]$ is a total function from D^n to $\{\mathbf{f}, \mathbf{e}, \mathbf{t}\}$.

Semantics of Binary Operators

$\langle\ \rangle$:

f	**f**	**f**
e	**e**	**e**
f	**e**	**t**

$\wedge$:

f	**e**	**f**
e	**e**	**e**
f	**e**	**t**

$[\ ]$:

t	**t**	**t**
e	**e**	**e**
f	**e**	**t**

$\rightarrow$:

t	**e**	**t**
e	**e**	**e**
f	**e**	**t**

$\vee$:

f	**e**	**t**
e	**e**	**e**
t	**e**	**t**

Unary Operators

	t			**t**
$\neg$:	**e**		Prop :	**f**
	f			**t**

Quantifiers

Quantifiers must be associated to the modified $\vee$ and $\wedge$:

For $\forall x\ P(x)$ and $\exists x\ P(x)$, construct $S = \{I_d^x(\ P(x)\ ) \mid d \in D\}$. Then select most preferred value from S, using the preferences:

$$\forall : \mathbf{e} > \mathbf{f} > \mathbf{t}, \qquad \exists : \mathbf{e} > \mathbf{t} > \mathbf{f}.$$

Contexts

PCL reasons with contexts:

$\forall x$ Prop(Nat(x)),

Nat(0),

$\forall x$ Nat(x) $\rightarrow$ Nat(succ(x)),

$\forall xy$ Nat(x) $\wedge$ Nat(y) $\rightarrow$ Prop($x \leq y$),

$\forall xy$ [Nat(x), Nat(y), Nat(z)] $x \leq y \wedge y \leq z \rightarrow x \leq z$,

$\forall xy$ [Nat(x), Nat(y)] $y \leq x \rightarrow$ Nat($x - y$),

$\forall xy$ [Nat(x), Nat(y)] $x \leq y \rightarrow \exists z \ \langle$ Nat(z), $x \geq z \ \rangle \ x - z = y$.

Definition: We call an object of form $\|\Gamma_1, \ldots, \Gamma_m\|$, in which all Γ_j are formulas, and in which some Γ_j are possibly marked with a ${}^{\theta}$, a context.

The formulas that are marked with θ are theorems, the others are assumptions.

Example

$$\|\mathrm{Prop}(A), \mathrm{Prop}(B), A, B, (A \wedge B)^{\theta}, \ldots\|$$

is a context.

A context is strongly valid if in every interpretation $I = (D, \mathbf{f}, \mathbf{t}, \mathbf{e}, [\ ])$, for which there is an i, s.t. $I(\Gamma_i) \neq \mathbf{t}$, the first such i satisfies the following condition:

- Γ_i is not marked as a theorem, and $I(\Gamma_i) = \mathbf{f}$.

Examples

Not strongly valid:

$$\|A, A^\theta\|$$

Strongly valid:

$$\|\text{Prop}(A), A, A^\theta\|$$

Not strongly valid:

$$\|\text{Prop}(A), A, (A \vee B)^\theta\|$$

Strongly valid:

$$\|\text{Prop}(A), \text{Prop}(B), A, (A \vee B)^\theta\|$$

$$\|\text{Prop}(A),\ \text{Prop}(B),\ \neg B,\ \neg A \vee B, (\neg A)^\theta\|$$

PCL has strictness for types and preconditions: Nothing can be done with a formula that does not respect the types of the preconditions.

It has truth-value based semantics.

There is no restriction on the form of types or preconditions.

Theorem Proving in PCL

Intuition: Check type correctness of the formulas. After that, replace by relativizations, and use a standard approach for classical logic.

$\Rightarrow$ Almost possible, but one needs Kleene logic.

Kleene logic can be used for type checking and for proving.

Kleene Logic

The semantics of $\neg$ and Prop is defined by the following truth tables:

$\neg$:

t
e
f

Prop :

t
f
t

The semantics of the operators $\oplus$, $\otimes$ is defined by the following truth tables:

$\oplus$:

f	**e**	**t**
e	**e**	**t**
t	**t**	**t**

$\otimes$:

f	**f**	**f**
f	**e**	**e**
f	**e**	**t**

$\rightarrow$ and $\leftrightarrow$ can be defined in the usual way.

Kleene Logic (2)

The semantics of the quantifiers is defined by the following preferences:

$$\Pi : \mathbf{f} > \mathbf{e} > \mathbf{t}, \qquad \Sigma : \mathbf{t} > \mathbf{e} > \mathbf{f}.$$

In order to evaluate a quantified formula $Qx\ F(x)$ in interpretation I, form the set $S = \{ I_d^x(\ F(x)\ ) \}$. (The set of possible truth values that $F(x)$ can have in I, by picking a value for x.)

After that, select the most preferred value for the quantifier in the list above from S.

Π is connected to $\otimes$, while Σ is connected to $\oplus$.

Expressivity of Kleene Logic

Kleene logic may seem different from classical logic, but the differences are minimal:

One can ask questions of the following form:

Is P satisfiable? (Is there a 3-valued interpretation I, in which $I(P) = \mathbf{t}$?)

Do $P_1, \ldots, P_n$ imply Q? (Is, in every 3-valued interpretation where $I(P_1) = \cdots = I(P_n) = \mathbf{t}$, also $I(Q) = \mathbf{t}$?)

Let $T(P)$ denote $I(P) = \mathbf{t}$, let $F(P)$ denote $I(P) = \mathbf{f}$.

$T()$ and $F()$ can be viewed as logical operators, whose result is in $\{\mathbf{f}, \mathbf{t}\}$.

Equivalences involving $T()$ and $F()$:

$T(P \otimes Q)$	$T(P) \wedge T(Q)$
$F(P \otimes Q)$	$F(P) \vee F(Q)$
$T(P \oplus Q)$	$T(P) \vee T(Q)$
$F(P \oplus Q)$	$F(P) \wedge F(Q)$
$T(\neg P)$	$F(P)$
$F(\neg P)$	$T(P)$
$T(\ \Pi x\ P(x)\ )$	$\forall x\ T(P(x))$
$F(\ \Pi x\ P(x)\ )$	$\exists x\ F(P(x))$
$T(\ \Sigma x\ P(x)\ )$	$\exists x\ T(P(x))$
$F(\ \Sigma x\ P(x)\ )$	$\forall x\ F(P(x))$
$\mathrm{Prop}(A)$	$T(A) \vee F(A)$

We see that Kleene logic is only slightly more expressive than classical logic.

The only only point where one can find any difference at all is in the atoms.

For atoms $p(t_1, \ldots, t_n)$, define

$$\begin{aligned} p_{\mathbf{f}}(t_1, \ldots, t_n) &:= F(\ p(t_1, \ldots, t_n)\ ), \\ p_{\mathbf{t}}(t_1, \ldots, t_n) &:= T(\ p(t_1, \ldots, t_n)\ ), \\ p_{\mathbf{e}}(t_1, \ldots, t_n) &:= \neg\mathrm{Prop}(\ p(t_1, \ldots, t_n)\ ), \end{aligned}$$

and Kleene logic is (almost) gone.

$\Rightarrow$ theorem proving in Kleene logic is easy!

Use superposition, tableaux, or geometric logic.

Sequents, Strong Representation

Definition: A sequent is a set of formulas $\{F_1, \ldots, F_n\}$. For an interpretation I, we define $I(\{F_1, \ldots, F_n\}) = \bigotimes_{1 \le i \le n} I(F_i)$.

Let $S_1, \ldots, S_n$ be a set of sequents. We define $I(S_1, \ldots, S_n) = \bigoplus_{1 \le i \le n} I(S_i)$.

We say that a set of sequents $S_1, \ldots, S_n$ represents a property P if $P \Leftrightarrow I(S_1, \ldots, S_n) \neq \mathbf{t}$.

We say that a set of sequents $S_1, \ldots, S_n$ strongly represents a property P if

$$\begin{cases} P \Rightarrow I(S_1, \ldots, S_n) = \mathbf{f}, \\ \neg P \Rightarrow I(S_1, \ldots, S_n) = \mathbf{t}. \end{cases}$$

It is important to see the redundancy in strong representation!

From PCL to Kleene Logic

Definition: Let $\|\Gamma_1, \ldots, \Gamma_n\|$ be a context. Recursively define $E(\ \|\Gamma_1, \ldots, \Gamma_n\|\ )$, the expansion of $\|\Gamma_1, \ldots, \Gamma_n\|$, as follows:

- $E(\ \|\ \|\ ) = \emptyset$.
- $E(\ \|\Gamma_1, \ldots, \Gamma_n, \Gamma_{n+1}\|\ ) =$

$$E(\ \|\Gamma_1, \ldots, \Gamma_n\|\ ) \cup \{\Gamma_1, \ldots, \Gamma_n, \neg\text{Prop}(\Gamma_{n+1})\}.$$

- $E(\ \|\Gamma_1, \ldots, \Gamma_n, \Gamma_n^{\theta}\|\ ) =$

$$E(\ \|\Gamma_1, \ldots, \Gamma_n\|\ ) \cup$$

$$\{\ \{\Gamma_1, \ldots, \Gamma_n, \neg\Gamma_{n+1}\},\ \{\Gamma_1, \ldots, \Gamma_n, \neg\text{Prop}(\Gamma_{n+1})\}\ \}.$$

Theorem: For a context $\Gamma = \|\Gamma_1, \ldots, \Gamma_n\|$, the expansion $E(\ \Gamma\ )$ strongly represents the property 'Γ is strongly valid.'

Relation $\preceq$

Definition: Write $A \preceq B$ if in every interpretation I,

$$\begin{cases} I(A) = \mathbf{f} & \Rightarrow & I(B) = \mathbf{f}, \\ I(A) = \mathbf{t} & \Rightarrow & I(B) = \mathbf{t}. \end{cases}$$

Define $A \equiv B$ if $A \preceq B$ and $B \preceq A$.

Lemma $\preceq$ is a reflexive and transitive relation.

Theorem Let $S_1, \ldots, S_n,\ S \cup \{A\}$ be a set of sequents. Let A and B be formulas for which $A \preceq B$.

Let P be a property.

If $S_1, \ldots, S_n,\ S \cup \{A\}$ strongly represents P, then $S_1, \ldots, S_n,\ S \cup \{B\}$ also strongly represents P.

Some More Reasoning Rules

$S_1, \ldots, S_n,\ S \cup \{A \otimes B\}$ strongly represents P iff
$S_1, \ldots, S_n,\ S \cup \{A, B\}$ strongly represents P.

If $S_1, \ldots, S_n,\ S \cup \{A \oplus B\}$ strongly represents P, then
$S_1, \ldots, S_n,\ S \cup \{A\},\ S \cup \{B\}$ also strongly represents P.

If $S_1, \ldots, S_n,\ S \cup \{\Sigma x\ A(x)\}$ strongly represents P, then for every variable c that does not occur in $S_1, \ldots, S_n, A$, or S,

$S_1, \ldots, S_n,\ S \cup \{A(c)\}$ also strongly represents P.

Result is a sequent calculus can be used. (Basing a calculus on ordinary representation is much harder.)

Theorem

- Except for Prop, all Kleene and PCL-operators, including the quantifiers, are $\preceq$-monotone.
- For formulas F and G,

$$\begin{array}{lcllcl} F \to G & \preceq & \neg F \oplus G & \langle F \rangle G & \preceq & F \otimes G, \\ [F]G & \preceq & \neg F \oplus G & F \wedge G & \preceq & F \otimes G, \\ F \vee G & \preceq & F \oplus G & F \leftrightarrow G & \preceq & (\neg F \oplus G) \otimes \\ & & & & & (F \oplus \neg G). \end{array}$$

- $\forall x\ Q(x) \prec \Pi x\ Q(x)$ and $\exists x\ Q(x) \prec \Sigma x\ Q(x)$.

Radicalization

A radicalization operator ! is an operator for which for every formula A, we have $A! \in \{\mathbf{f}, \mathbf{t}\}$, and $A \preceq A!$.

Theorem

For every radicalization operator !,

$$A \otimes B \preceq A! \wedge B!,$$

$$A \oplus B \preceq A! \vee B!,$$

$$\Pi x \; Q(x) \preceq \forall x \; (\; Q(x)! \;),$$

$$\Sigma x \; Q(x) \preceq \exists x \; (\; Q(x)! \;).$$

(This is the same result that we had before, but now generalized to strong representation. It confirms the weakness of Kleene logic.)

Back to Relativization

The radicalizations of the Kleenings of the sequents $S_1, S_1, \ldots, S_n$ in $E(\|\Gamma\|)$ cover the original intuition:

When the formula has been typechecked, it can be replaced by its relativization.

Sequent S_n is nearly always classical.

The only way of making it non-classical is by explicitly reasoning about Prop, which is usually done only in assumptions.

Geometric Formulas

A geometric literal is an object of one of the following three forms:

1. A variable atom $p_\lambda(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are variables, and $\lambda \in \{\mathbf{f}, \mathbf{e}, \mathbf{t}\}$. Repeated variables are allowed.

2. An equality atom $x_1 \approx x_2$.

3. An existentially quantified atom $\exists y\ p_\lambda(x_1, \ldots, x_n, y)$, where $x_1, \ldots, x_n$ and y are variables, and $\lambda \in \{\mathbf{e}, \mathbf{t}\}$. There must be at least one occurrence of y in the atom $p_\lambda(x_1, \ldots, x_n, y)$. Repeated occurrences of variables (including y) are allowed, and y does not have to be on the last position.

A geometric formula is a formula of form $\forall \overline{x}\ A_1 \vee \cdots \vee A_p$, where each A_i is a geometric literal with all its free variables among $\overline{x}$.

Interpretations

Definition: We assume an infinite set of elements (constants) $\mathcal{E}$. A ground atom is

1. an object of form $p_\lambda(e_1, \ldots, e_n)$, where $n \geq 0, \quad e_1, \ldots, e_n \in \mathcal{E}$ and $\lambda \in \{\mathbf{f}, \mathbf{e}, \mathbf{t}\}$.
2. an object of form $e_1 \approx e_2$, with $e_1, e_2 \in \mathcal{E}$.

Definition An interpretation is a pair (E, M) in which $E \subseteq \mathcal{E}$ is a set of elements, and M is a set of ground atoms over E, s.t. M contains no ground atoms of form $p_{\mathbf{f}}(e_1, \ldots, e_n)$, no ground atoms of form $e_1 \approx e_2$, and no conflicting pairs of ground atoms $p_{\mathbf{e}}(e_1, \ldots, e_n),\ p_{\mathbf{t}}(e_1, \ldots, e_n)$.

Conflict, False

Let A be a geometric literal, let (E, M) be an interpretation. Let Θ be a ground substitution:

$A\Theta$ is in conflict with (E, M) if

- A has form $x \approx y$, and $x\Theta \neq y\Theta$.
- A has form $p_\lambda(x_1, \ldots, x_n)$, and there is an atom of form $p_\mu(x_1\Theta, \ldots, x_n\Theta) \in E$, for which $\lambda \neq \mu$.

$A\Theta$ is true in (E, M) if

- A has form $x \approx y$, and $x\Theta = y\Theta$.
- A has form $p_\lambda(x_1, \ldots, x_n)$ and $A\Theta \in M$.
- A has form $\exists y\ p_\lambda(x_1, \ldots, x_n, y)$, and there is a $e \in E$, s.t. $A\Theta\{y := e\} \in M$.

For a geometric formula $F = \forall\overline{x}\ A_1 \vee \cdots \vee A_p$,

$F\Theta$ is false in (E, M) if all A_i are false in (E, M).

Lemma If $A\Theta$ conflicts (E, M), then $A\Theta$ is false in (E, M).

If $A\Theta$ conflicts (E, M), and $E \subseteq E',\ M \subseteq M'$, then $A\Theta$ also conflicts (E', M').

Search Algorithm

Find a geometric formula F and a substitution Θ, s.t. $F\Theta$ is false in (E, M). If no such F and Θ exist, then (E, M) is a model.

Write $F = \forall\overline{x}\ A_1 \vee \cdots \vee A_p$.

If all $A_i\Theta$ are in conflict with (E, M), then give up.

Otherwise, let $B_1, \ldots, B_q \subseteq A_1, \ldots, A_p$ be the literals that are not in conflict with (E, M). (but they are still false)

Guess a B_j. If B_j has form $p_\lambda(x_1, \ldots, x_n)$, then add $B_j\Theta$ to M, and (recursively) continue search.

Otherwise, B_j must have form $\exists y\ p_\lambda(x_1, \ldots, x_n, y)$.

- Either guess a value $e \in E$, add $p_\lambda(x_1, \ldots, x_n, y)\Theta\{y := e\}$ to M, and (recursively) continue search,
- Or create an $\hat{e} \notin E$, add $p_\lambda(x_1, \ldots, x_n, y)\Theta\{y := \hat{e}\}$ to M, add $\hat{e}$ to E, and continue search.

Learning, Effectiveness of the Calculus

The search algorithm can be enhanced with learning in the same way as with classical logic. The learning rules are rather complicated. They are similar to resolution rules.

The calculus on classical logic was fairly effictive, and I hope that the calculus on Kleene logic will also be effective. Users of ATP ask for type systems.

The calculus can be adopted to certain applications. (e.g. type checking.)

Example of Geometric Formulas

$\mathrm{Prop}(A)$	$A_{\mathbf{f}} \vee A_{\mathbf{t}}$
$A \to \mathrm{Prop}(B)$	$A_{\mathbf{f}} \vee B_{\mathbf{f}} \vee B_{\mathbf{t}}$
$A \to \mathrm{Prop}(C)$	$A_{\mathbf{f}} \vee C_{\mathbf{f}} \vee C_{\mathbf{t}}$
$[\mathrm{Prop}(A)]\ A$	$A_{\mathbf{e}} \vee A_{\mathbf{t}}$
$[\mathrm{Prop}(A)]\ \neg\mathrm{Prop}(B)$	$A_{\mathbf{e}} \vee B_{\mathbf{e}}$

Another Refutation

We want to prove $a \approx b \to s(a) \approx s(b)$. This is done by refuting $a \approx b,\ s(a) \not\approx s(b)$.

We obtain the following geometric formulas:

$$
\begin{array}{ll}
(1) & \exists y\ A_{\mathbf{t}}(y) \\
(2) & \exists y\ B_{\mathbf{t}}(y) \\
(3) & \forall x\ \exists y\ S_{\mathbf{t}}(x, y) \\
(4) & \forall \alpha\beta\ A_{\mathbf{f}}(\alpha) \vee B_{\mathbf{f}}(\beta) \vee \alpha \approx \beta \\
(5) & \forall \alpha\beta\gamma\ A_{\mathbf{f}}(\alpha) \vee B_{\mathbf{f}}(\beta) \vee S_{\mathbf{f}}(\alpha, \gamma) \vee S_{\mathbf{f}}(\beta, \gamma)
\end{array}
$$

Transformation to Geometric Formulas

Tranformations is mostly straightforward, but one needs subformula replacement.

In classical logic, subformula replacement is defined as follows:

Let $F[A]$ be a formula, with subformula A. Assume that $\overline{x}$ are the free variables of F. Replace $F[A(\overline{x})]$ by

$$F[p(\overline{x})], \quad \forall \overline{x}\ p(\overline{x}) \leftrightarrow A(\overline{x}).$$

Subformula Replacement with $\preceq$

We say that A occurs positively in F, if A is not in the scope of a $\neg$ or $\leftrightarrow$, not inside a left argument of a $\rightarrow$, and not in the scope of a Prop.

If A occurs positively in F, then

$$F[A(\overline{x})] \preceq \Sigma p \left(\begin{array}{l} F[p(\overline{x})] \otimes \\ \Pi \overline{x} \ \mathrm{Prop}(P(\overline{x})) \otimes \\ \Pi \overline{x} \ \neg p(\overline{x}) \oplus A(\overline{x}) \end{array} \right)$$

If $I(\ F[A(\overline{x})]\ ) = \mathbf{t}$, then take $p := \lambda \overline{x} \ \langle \ \mathrm{Prop}(A(\overline{x})\ \rangle A(\overline{x})$.

If $I(\ F[A(\overline{x})]\ ) = \mathbf{f}$, then assume that the second formula is not false for some predicate p. We have
$I'(A(\overline{x})) = \mathbf{f} \Rightarrow I'(p(\overline{x})) \neq \mathbf{t} \Rightarrow I'(p(\overline{x})) = \mathbf{f}$.

By monotonicity, we obtain $F[A(\overline{x})] = \mathbf{f} \Rightarrow F[p(\overline{x})] = \mathbf{f}$.

Subformula Replacement

Positive subformula replacement is sufficient for transformation to geometric formulas.

1. Transform to Kleene logic and NNF, using $\preceq$.
2. Introduce predicates for functions of form $F := \lambda\overline{x}y\ f(\overline{x}) \approx y$. Introduce axioms $\forall\overline{x}\exists y\ F(\overline{x}, y)$. Use definition of F to remove function symbols.
3. Remove negative equality by substitution.
4. Use positive subformula replacement to obtain geometric format.

Conclusions, Future Work

- I believe that the higher-order variant of PCL (PHOLI, which I totally didn't speak about) is 'the right logic' for applications.
- I have shown how to do theorem proving in PCL. The main difference with classical logic is in the clause transformation. Once we have geometric format, there is not much difference with classical logic.
- Resolution can be developed in a similar way.
- The theorem proving methods clarify the connections between PCL, Kleene logic and classical logic. It confirms that PCL is close to simple type theory.
- All of this needs to be implemented.