Chapter 5: Predicate Calculus: Formulas, Models, Tableaux

November 3, 2008
Outline

1 5.1 Relations and Predicates
2 5.2 Predicate Formulas
3 5.3 Interpretations
4 5.4 Equivalence and Substitution
5 5.5 Semantic Tableaux
6 5.7 Finite and Infinite Models
7 5.8 Undecidability of the Predicate Logic
5.1 Relations and Predicates

- $R$: an $n$-ary relation on a set $D$

$$R \subseteq D^n = D \times D \times \ldots \times D \quad \text{n times}$$

$D$: domain of the relation $R$.

Observation: A unary relation $R$ is simply a subset of $D$

$$R \subseteq D$$
Examples

(a) Binary relation $<$ on $\mathbb{N}$:

$x < y$ if $x$ is a positive integer less than $y$

$\leq \{(0,1), (0,2), \ldots, (1,2), (1,3), \ldots, (2,3), \ldots\}$

(b) Unary relation $\text{Prime}(x)$ on $\mathbb{N}$:

$\text{Prime} = \{2, 3, 5, 7, 11, \ldots\}$
(c) Given the graph $G$:

\[ r(x, y) \iff \text{vertex } x \text{ is connected by a path to vertex } y \]

\[ r = \{(a, a), (b, b), (c, c), (d, d), (e, e), (a, b), (b, a), (a, c), (c, a), (b, c), (c, b), (d, e), (e, d)\} \]
• We can think of an $n$-ary function

$$(x_1, x_2, \ldots, x_n) \mapsto f(x_1, x_2, \ldots, x_n)$$

as an $(n + 1)$-ary relation $R_f$ containing the $(n + 1)$-tuples

$$(x_1, x_2, \ldots, x_n, f(x_1, x_2, \ldots, x_n))$$

$R_f$ is called the **graph** of the function $f$.

• Also, we can think of an $n$-ary relation $R \subseteq D^n$ as a function

$$f : D^n \rightarrow \{T, F\}$$

$$R(d_1, d_2, \ldots, d_n) = T \iff (d_1, d_2, \ldots, d_n) \in R$$
5.2 Predicate Formulas

Predicate (relation) symbols \( \mathcal{P} = \{p, q, r, \ldots\} \)
Constant symbols \( \mathcal{A} = \{a, b, c, \ldots\} \)
Variables \( \mathcal{V} = \{x, y, z, \ldots\} \)
BNF Grammar for Predicate Formulas

\[
\begin{align*}
\text{argument} &::= x, \quad \text{for any } x \in \mathcal{V} \\
\text{argument} &::= a, \quad \text{for any } a \in \mathcal{A} \\
\text{argumentList} &::= \text{argument} \\
\text{argumentList} &::= \text{argument}, \text{argumentList} \\
\text{atomicFormula} &::= p \mid p(\text{argumentList}), \quad \text{for any } p \in \mathcal{P}
\end{align*}
\]
formula ::= atomicFormula
formula ::= ¬formula
formula ::= formula ∧ formula
formula ::= formula ∨ formula
formula ::= formula → formula
formula ::= formula ↔ formula
formula ::= ∀x formula, for all x ∈ V
formula ::= ∃x formula, for all x ∈ V
Examples

1. \( p(x, a) \) (atomic formula)
2. \( p(x, a) \rightarrow q(x) \)
3. \( \exists x \ p(x, a) \rightarrow \forall y \ q(y) \)
4. \( \forall x \ (p(x, a) \rightarrow q(x, y)) \rightarrow (\forall x \ p(x, a) \rightarrow \forall x \ q(x, y)) \)
Bound and Free Variables

Definition
Suppose $A$ is a predicate formula. An occurrence of a variable $x$ in $A$ is a free variable of $A$ if it is not within the scope of any quantifier $\forall x$ or $\exists x$. 


Examples

(a) \( \exists y \ p(x, y) \)
    \( x \)-free, \( y \)-not free
(b) \( p(x, y) \)
    \( x, y \)-free
(c) \( \forall x \exists y p(x, y) \)
    neither \( x \) nor \( y \) are free
(d) \( \forall x p(x) \lor q(x) \)
    the first occurrence of \( x \) is not free while the second occurrence is
• A variable which is not free is said to be **bound**.

• If we write

\[ A(x_1, x_2, \ldots, x_n), \]

we mean that the free variables of the formula \( A \) are among \( x_1, x_2, \ldots, x_n \).
5.3 Interpretations

- $U$: a set of formulas
- $\{p_1, p_2, \ldots, p_m\}$: all predicate symbols appearing in $U$
- $\{a_1, a_2, \ldots, a_k\}$: all constant symbols appearing in $U$
Definition
An interpretation $I$ of $U$ is a triple

$$I = (D, \{R_1, R_2, \ldots, R_m\}, \{d_1, d_2, \ldots, d_k\})$$

where
- $D$ is a non-empty set (domain of $I$)
- $R_i$ are $n_i$-ary relations on $D$.
- $d_i$ are some fixed elements of $D$.

\[
\begin{align*}
p_i & \mapsto R_i & i = 1, 2, \ldots, m \\
a_j & \mapsto d_j & j = 1, 2, \ldots, k
\end{align*}
\]
Example
Consider the formula

\[ \forall x \ p(a, x) \]

Some of its possible interpretations are:

(1) \( I_1 = (\mathbb{N}, \{\leq\}, \{0\}) \)

“For every natural number \( x \), \( 0 \leq x \).”

(2) \( I_2 = (\mathbb{N}, \{|\}, \{1\}) \)

“For every natural number \( x \), \( 1 | x \).”
(3) \( I_3 = (\{0, 1\}^*, \{ \text{substring relation} \}, \{\varepsilon\}) \)

“For every string \( x \) over alphabet \( \{0, 1\} \), empty string is a substring of \( x \).”

(4) \( I_4 = (G, E, \{a\}) \)

“For every vertex \( x \) of \( G \), \((a, x)\) is an edge in \( G \).”
Definition
Suppose $I$ is an interpretation for a predicate formula $A$. An assignment

$$\sigma_I : \mathcal{V} \rightarrow D$$

is a function which assigns a value in the domain $D$ to any variable appearing in the formula $A$. 
Truth Value of a Predicate Formula

Suppose:
- $A$ - formula.
- $I$ - an interpretation for $A$.
- $\sigma_I$ - an assignment

We define $v_{\sigma_I}(A)$, the truth value of $A$ under $\sigma_I$, inductively:
(a) If $A = p(c_1, c_2, \ldots, c_n)$ is an atomic formula, where each $c_i$ is either a variable $x_j$ or a constant symbol $a_j$, then

$$v_{\sigma_I}(A) = T \text{ iff } (\sigma_I(c_1), \sigma_I(c_2), \ldots, \sigma_I(c_n)) \in R$$

(b) $v_{\sigma_I}(\neg A) = \neg v_{\sigma_I}(A)$.

(c) $v_{\sigma_I}(A_1 \land A_2) = v_{\sigma_I}(A_1) \land v_{\sigma_I}(A_2)$.

(d) $v_{\sigma_I}(A_1 \lor A_2) = v_{\sigma_I}(A_1) \lor v_{\sigma_I}(A_2)$.

[Similarly for $\rightarrow$, $\leftrightarrow$.]
(e) $v_{\sigma_1}(\forall x \ A) = T$ iff $v_{\sigma_1}(A) = T$ for all $x \in D$

(f) $v_{\sigma_1}(\exists x \ A) = T$ iff $v_{\sigma_1}(A) = T$ for some $x \in D$

**Theorem**

*If A is a closed formula, then $v_{\sigma_1}(A)$ does not depend on $\sigma_1$. In that case, we write*

$v_I(A)$
Theorem
Let $A' = A(x_1, x_2, \ldots, x_n)$ be a non-closed formula and let $I$ be an interpretation. Then:

(a) $v_{\sigma_I}(A') = T$ for assignment $\sigma_I$ iff

$$v_I(\exists x_1 \exists x_2 \ldots \exists x_n \ A') = T$$

(b) $v_{\sigma_I}(A') = T$ for all assignments $\sigma_I$ iff

$$v_I(\forall x_1 \forall x_2 \ldots \forall x_n \ A') = T$$

Definition
A closed formula $A$ is true in $I$, or $I$ is a model for $A$, if $v_I(A) = T$.

$$I \models A$$
Definition
A closed formula $A$ is **satisfiable** if, for **some** interpretation $I$,

$$I \models A$$

$A$ is **valid** if, for **all** interpretations $I$,

$$I \models A$$

We can also define unsatisfiable and falsifiable formulas in the usual way.
Examples

(a) $\forall x \, p(a, x) \rightarrow p(a, a)$  \hspace{1cm} valid
(b) $\forall x \forall y \, (p(x, y) \rightarrow p(y, x))$  \hspace{1cm} not valid, satisfiable
(c) $\forall x \exists y \, p(x, y)$  \hspace{1cm} not valid, satisfiable
(d) $\exists x \exists y \, (p(x) \land \neg p(y))$  \hspace{1cm} not valid, satisfiable
(e) $\forall x \,(p(x) \land q(x)) \leftrightarrow (\forall x \, p(x) \land \forall x \, q(x))$  \hspace{1cm} valid
(f) $\exists x \, (\neg p(x) \land p(x))$  \hspace{1cm} unsatisfiable
5.4 Equivalence and Substitution

- Suppose $A_1, A_2$ are two closed formulas. If, for all interpretations $I$

  $$v_I(A_1) = v_I(A_2)$$

  we say that $A_1$ and $A_2$ are **equivalent**, and we write

  $$A_1 \equiv A_2$$

- Suppose $U$ is a set of closed formulas, and $A$ a closed formula

  $$U \models A$$

  means that, in all interpretations $I$ in which all formulas from $U$ are true, we also have

  $$v_I(A) = T.$$
Examples

(a) $\forall x \ A(x) \equiv \neg \exists x \ \neg A(x)$

(b) $\exists x \ A(x) \equiv \neg \forall x \ \neg A(x)$

(c) $\forall x \forall y \ A(x, y) \equiv \forall y \forall x \ A(x, y)$

(d) $\exists x \exists y \ A(x, y) \equiv \exists y \exists x \ A(x, y)$

(e) $\exists x \forall y A(x, y) \not\equiv \forall y \exists x A(x, y)$

To see that these two formulas are not equivalent, consider

$I = (\mathbb{Z}, \{\leq\})$.

Clearly,

$I \not\models \exists x \forall y \ x \leq y, \quad I \models \forall y \exists x \ x \leq y$
Theorem
(a) $A \equiv B$ if and only if $\models A \leftrightarrow B$.
(b) Suppose $U = \{A_1, A_2, \ldots, A_n\}$

$U \models A$ if and only if $\models A_1 \land A_2 \land \ldots A_n \rightarrow A$. 
Examples

The following are valid formulas

(a) \( \exists x (A(x) \lor B(x)) \leftrightarrow \exists x A(x) \lor \exists x B(x) \)

(b) \( \forall x (A(x) \land B(x)) \leftrightarrow \forall x A(x) \land \forall x B(x) \)

(c) \( \exists x (A(x) \land B) \leftrightarrow \exists x A(x) \land B, \text{ if } x \text{ is not free in } B. \)

(d) \( \forall x (A(x) \lor B) \leftrightarrow \forall x A(x) \lor B, \text{ if } x \text{ is not free in } B. \)

(e) \( \exists x (A(x) \rightarrow B(x)) \leftrightarrow (\forall x A(x) \rightarrow \exists x B(x)) \)

(f) \( \forall x (A(x) \rightarrow B(x)) \leftrightarrow (\exists x A(x) \rightarrow \forall x B(x)) \)

[For more pairs of equivalent formulas, see Fig. 5.2 in Section 5.4]
Proof.

(e)

\[ \exists x(A(x) \rightarrow B(x)) \equiv \exists x(\neg A(x) \lor B(x)) \]
\[ \equiv \exists x \neg A(x) \lor \exists x B(x) \]
\[ \equiv \neg \forall x A(x) \lor \exists x B(x) \]
\[ \equiv \forall x A(x) \rightarrow \exists x B(x) \]
Example
Prove that
\[ \exists x \forall y \ A(x, y) \rightarrow \forall y \exists x \ A(x, y) \]
is a valid formula, yet its converse is not valid.

Solution:
Let \( I \) be an interpretation. Suppose
\[ I \models \exists x \forall y \ A(x, y). \]
Then, for some \( a \in D \)
\[ I \models \forall y \ A(a, y) \]
So,
\[ I \models \forall y (\exists x \ A(x, y)) \]
which proves that, for every \( I \),
\[ I \models \exists x \forall y \ A(x, y) \rightarrow \forall y \exists x \ A(x, y) \]
\( I = (\mathbb{Z}, \{\leq\}) \) shows that the implication cannot be reversed if we want the formula to be valid. \( \Box \)
5.5 Semantic Tableaux

Example
We will try to show that

$$\forall x(p(x) \rightarrow q(x)) \rightarrow (\forall x p(x) \rightarrow \forall x q(x))$$

is a valid formula.

We consider its negation

$$\neg[\forall x(p(x) \rightarrow q(x)) \rightarrow (\forall x p(x) \rightarrow \forall x q(x))]$$

and try to show that it is unsatisfiable.
$\neg [\forall x (p(x) \rightarrow q(x)) \rightarrow (\forall x p(x) \rightarrow \forall x q(x))]$

$\forall x (p(x) \rightarrow q(x)), \neg (\forall x p(x) \rightarrow \forall x q(x))$

$\forall x (p(x) \rightarrow q(x)), \forall x p(x), \neg \forall x q(x)$

$\forall x (p(x) \rightarrow q(x)), \forall x p(x), \neg q(a)$

$\forall x (p(x) \rightarrow q(x)), p(a), \neg q(a)$

$p(a) \rightarrow q(a), p(a), \neg q(a)$

$\neg p(a), p(a), \neg q(a), q(a), p(a), \neg q(a)$

$\times$ $\times$
Example

Now, we consider the formula

\[ \forall x (p(x) \lor q(x)) \rightarrow (\forall x p(x) \lor \forall x q(x)) \]

which is not valid, but is satisfiable.
\[
\neg [\forall x(p(x) \lor q(x)) \rightarrow (\forall x p(x) \lor \forall x q(x))] \\
\forall x(p(x) \lor q(x)), \neg(\forall x p(x) \lor \forall x q(x)) \\
\forall x(p(x) \lor q(x)), \exists x \neg p(x), \exists x \neg q(x) \\
\forall x(p(x) \lor q(x)), \neg p(a), \exists x \neg q(x) \\
p(a) \lor q(a), \neg p(a), \exists x \neg q(x) \\
p(a), \neg p(a), \exists x \neg q(x) q(a), \neg p(a), \exists x \neg q(x) \\
q(a), \neg p(a), \neg q(a) \\
\neg q(a)
\]
**Question:** What went wrong?

- We used the same constant $a$ twice to eliminate two distinct existential quantifiers.
- We were forced to use the same constant since, once we eliminated the universal quantifier in

$$\forall x(p(x) \lor q(x))$$

we replaced it with $a$ and were forced to work with that constant exclusively from that point on.

**Solution:** We will not delete universal quantifiers from nodes of the tableau; instead, we introduce some instance of that variable but keep writing the universal quantifier. E.g.

$$\begin{array}{c}
\forall x \ p(x) \\
\ \ \ |
\forall x \ p(x), \ p(a)
\end{array}$$
Using these guidelines, if we construct a correct tableau for the formula from the previous example (exercise!), we notice that one branch ends with the open leaf

\[ p(a), \neg q(a), \neg p(b), q(b) \]

In fact, this leaf gives us a model for this satisfiable formula; the domain is

\[ D = \{ a, b \} \]

and the unary relations are subsets

\[ p = \{ a \}, \quad q = \{ b \} \]

[This is what we will define as an Herbrand model for this formula in Chapter 7.]
Example

Consider the formulas

\[ A_1 = \forall x \exists y \ p(x, y) \]
\[ A_2 = \forall x \neg p(x, x) \]
\[ A_3 = \forall x \forall y \forall z (p(x, y) \land p(y, z) \rightarrow p(x, z)) \]

Check whether

\[ A = A_1 \land A_2 \land A_3 \]

is a satisfiable formula and, if so, find one model for \( A \).
Solution: We will first construct a semantic tableau for the formula:

\[ \forall x \exists y \, p(x, y), A_2, A_3 \]

\[ \forall x \exists y \, p(x, y), \exists y(a_1, y), A_2, A_3 \]

\[ \forall x \exists y \, p(x, y), p(a_1, a_2), A_2, A_3 \]

\[ \forall x \exists y \, p(x, y), \exists y \, p(a_2, y), p(a_1, a_2), A_2, A_3 \]

\[ \forall x \exists y \, p(x, y), p(a_2, a_3), p(a_1, a_2), A_2, A_3 \]

\[ \vdots \]
We see that the tableau does not terminate; namely, every time we drop the universal or an existential quantifier, we can introduce a new constant symbol $a_i$, to get an infinite sequence of constants:

$$a_1, a_2, \ldots, a_n, \ldots$$

The formula does have an obvious infinite model:

$$I = (\mathbb{N}, \{<\})$$

Furthermore, one can prove, using the formulas $A_2$ and $A_3$ (see the proof of Theorem 5.24 in the textbook) that every model of

$$A = A_1 \land A_2 \land A_3$$

must be infinite. So, the tableau construction effectively produces a “generic” infinite model for $A$. □
• One stark difference in comparison with semantic tableaux for propositional logic is (as seen in the previous example) that a tableau of a predicate formula may not terminate.
• The reason for this anomaly is that, in propositional logic, nodes of a tableau simplify in terms of the formula complexity. In predicate logic, this is not the case, since we can never eliminate universal quantifiers.
Algorithm for Semantic Tableaux

- Two new types of rules:

<table>
<thead>
<tr>
<th>( \gamma )</th>
<th>( \gamma(a) )</th>
<th>( \delta )</th>
<th>( \delta(a) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall x \ A(x) )</td>
<td>( A(a) )</td>
<td>( \exists x \ A(x) )</td>
<td>( A(a) )</td>
</tr>
<tr>
<td>( \neg \exists x \ A(x) )</td>
<td>( \neg A(a) )</td>
<td>( \neg \forall x \ A(x) )</td>
<td>( \neg A(a) )</td>
</tr>
</tbody>
</table>

- **Literal**: closed atomic formula \( p(a_1, a_2, \ldots, a_n) \) or the negation of such a formula.
Input: $A$ - a predicate formula

Output: Semantic tableau $\mathcal{T}$ for $A$; all branches are either infinite, or finite with leaves marked $\times$ (closed) or $\circ$ (open).

(1) Initially, $\mathcal{T}$ is a single node, labeled $\{A\}$.

(2) We build the tableau inductively by choosing an unmarked leaf $l$, labeled $U(l)$, and applying one of the following rules:
• If $U(I)$ is a set of literals and $\gamma$-formulas containing a pair of complementary literals
  \[ \{p(a_1, a, \ldots, a_n), \neg p(a_1, a_2, \ldots, a_n)\} \], mark it as closed ($\times$)
• If $U(I)$ is not a set of literals, choose a formula $A$ in $U(I)$ which is not a literal:
  \begin{itemize}
  \item $\alpha$- and $\beta$-rules are applied just as in propositional logic.
  \item If $A$ is a $\gamma$-formula, add a new node $l'$, a child of $I$, and label it
    \[ U(l') = U(I) \cup \{\gamma(a)\} \]
    where $a$ is a constant appearing in $U(I)$. If $U(I)$ consists of literals and $\gamma$-formulas only, mark it $\times$ or $\circ$, depending on whether there is a set of complementary literals.
  \item If $A$ is a $\delta$-formula, create a new node $l'$ as a child of $I$ and label it
    \[ U(l') = (U(I) - \{A\}) \cup \{\delta(a)\} \]
    where $a$ is some constant that does not appear in $U(I)$.
Definition
A branch in $\mathcal{T}$ is **closed** if it terminates in a leaf marked $\times$. Otherwise, it is **open**.

Theorem
*(Soundness)* Suppose A is a predicate formula and $\mathcal{T}$ its semantic tableau. If $\mathcal{T}$ closes, then A is unsatisfiable.

Theorem
*(Completeness)* Suppose A is a valid formula. Then, the systematic semantic tableau for A terminates and is closed.
• **Systematic tableau:** a tableau in which every node is labeled

\[ W(I) = (U(I), C(I)) \]

where \( U(I) \) is a list of formulas and \( C(I) \) is the list of all constant symbols appearing in \( U(I) \).

• In a systematic tableau, if using a \( \gamma \)-rule, we do the following: suppose \( \{\gamma_1, \ldots, \gamma_m\} \) are all \( \gamma \)-formulas in \( U(I) \) and

\[ C(I) = \{ a_1, \ldots, a_k \} \]

The new node \( I' \) will be labeled

\[ (U(I) \cup \{\gamma_i(a_j)\}, C(I)) \]

In other words, we create all possible instances of formulas \( \gamma_i \) where the variable is replaced by all possible constants \( a_j \).
5.7 Finite and Infinite Models

**Theorem**
*(Löwenheim)* If a formula is satisfiable, then it is satisfiable in a countable model.

**Theorem**
*(Löwenheim - Skolem)* If a countable set of predicate formulas is satisfiable, then it is satisfiable in a countable model.

**Theorem**
*(Compactness Theorem)* Let \( U \) be a countable set of formulas. If all finite subsets of \( U \) are satisfiable, then so is \( U \).
5.8 Undecidability of the Predicate Logic

- Turing machines can be viewed as devices which compute functions on natural numbers; i.e. given a Turing machine $T$, we can associate to it a function

$$f_T : \mathbb{N} \rightarrow \mathbb{N}$$

so that $f_T(n) = m$ if $T$ halts with the tape consisting of $m$ 1’s when started on the tape with the input of $n$ consecutive 1’s. If $T$ never halts on the input of $n$ consecutive 1’s, then $f_T(n)$ is undefined.

**Theorem**

*(Church)* It is undecidable whether a Turing machine, started on a blank tape, will halt.

- In other words, it is undecidable, given a Turing machine $T$, whether $f_T(0)$ is defined.
Two-Register Machines

Definition
Two-register machine (or, a Minsky machine) $M$ consists of a pair of registers $(x, y)$ which can store natural numbers, and a program $P = \{L_0, L_1, \ldots, L_n\}$, which is a sequential list of instructions. $L_n$ is always the command “halt”, and for $0 \leq i < n$, $L_i$ has one of the two forms

1. $r := r + 1$, for $r \in \{x, y\}$
2. If $r = 0$ then go to $L_j$ else $r := r - 1$, for $r \in \{x, y\}$, $0 \leq j \leq n$. 
• **Execution** of $M$: sequence of states

$$s_k = (L_i, x, y)$$

where $L_i$ is the current instruction during the execution, and $x, y$ are current contents of the two registers.

• **Initial state:**

$$s_0 = (L_0, m, 0), \text{ for some } m$$

• If

$$s_k = (L_n, x, y), \text{ for some } k$$

then $M$ halts and

$$y = f(m)$$

is computed by $M$. 
Theorem
For every Turing machine $T$ that computes $f : \mathbb{N} \to \mathbb{N}$, a two-register machine $M$ can be constructed which computes the same function.

Corollary
It is undecidable whether, given a two-register machine $M$, whether $f_M(0)$ exists or not.
Theorem
(Church) Validity in predicate calculus is undecidable.

Sketch of the Proof.
To each two-register machine $M$, we associate a predicate formula $S_M$ such that

$M$ halts started at $(L_0, 0, 0) \iff \models S_M$

We use the language:
- Binary relations: $p_i(x, y) \ (i = 0, 1, \ldots, n)$
- Unary function: $s(x)$
- Constant symbol: $a$

Intended interpretation:
- $p_i(x, y)$: $M$ is at the state $(L_i, x, y)$
- $s(x)$: successor function $s(x) = x + 1$
- $a$: $a = 0$
<table>
<thead>
<tr>
<th>$L_i$</th>
<th>$S_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x := x + 1$</td>
<td>$\forall x \forall y (p_i(x, y) \rightarrow p_{i+1}(s(x), y))$</td>
</tr>
<tr>
<td>$y := y + 1$</td>
<td>$\forall x \forall y (p_i(x, y) \rightarrow p_{i+1}(x, s(y)))$</td>
</tr>
<tr>
<td>if $x = 0$ then goto $L_j$</td>
<td>$\forall y (p_i(a, y) \rightarrow p_j(a, y))$</td>
</tr>
<tr>
<td>else $x := x - 1$</td>
<td>$\forall x \forall y (p_i(s(x), y) \rightarrow p_{i+1}(x, y))$</td>
</tr>
<tr>
<td>if $y = 0$ then goto $L_j$</td>
<td>$\forall x (p_i(x, a) \rightarrow p_j(x, a))$</td>
</tr>
<tr>
<td>else $y := y - 1$</td>
<td>$\forall x \forall y (p_i(x, s(y)) \rightarrow p_{i+1}(x, y))$</td>
</tr>
</tbody>
</table>
Finally, define

\[ S_M = (S_0 \land S_1 \land \ldots \land S_n \land p_0(a, a)) \rightarrow \exists z_1 \exists z_2 \ p_n(z_1, z_2) \]

\( S_M \) says the following: if a machine with the program

\[ P = \{L_0, L_1, \ldots, L_n\} \]

is started at the initial state \((L_0, 0, 0)\), then the computation will halt with the values at the registers being \((z_1, z_2)\), for some natural numbers \(z_1, z_2\).

Since the Halting Problem for two-register machines is undecidable, it is impossible to verify algorithmically whether

\[ \models S_M \]

or not. \(\square\)
Church’s Theorem is also true for some restricted classes of predicate logic:

1. Formulas containing only a finite number of binary predicate symbols, one unary function symbol, and one constant symbol.
2. Formulas written as Prolog programs.
3. Formulas with no function symbols.

[Skip ’Solvable Cases of the Decision Problem’ in Section 5.8]