Syntax
Components of Syntax

Words

\( a, b, g, p \)

Terms

\( g(a,a) \)

Sentences

\( \forall x. (p(x) \Rightarrow p(x,g(x,x))) \)
Words

Words are strings of letters, digits, and occurrences of the underscore character.

*Variables* begin with characters from the end of the alphabet (from \textit{u} through \textit{z}).

\begin{center}
\textit{u, v, w, x, y, z}
\end{center}

*Constants* begin with digits or letters from the beginning of the alphabet (from \textit{a} through \textit{t}).

\begin{center}
\textit{a, b, c, 123, comp225, barack_obama}
\end{center}
Constants

Object constants represent objects.

joe, stanford, usa, 2345

Function constants represent functions.

father, mother, age, plus, times

Relation constants represent relations.

knows, loves
Arity

The *arity* of a function constant or a relation constant is the number of arguments it takes.

*Unary* function or relation constant - 1 argument

*Binary* function or relation constant - 2 arguments

*Ternary* function or relation constant - 3 arguments

*n-ary* function or relation constant - *n* arguments
Signatures

A signature consist of a set of object constants, a set of function constants, and a set of relation constants together with a specification of arity for the function constants and relation constants.

Object Constants: $a, b$
Unary Function Constant: $f$
Binary Function Constant: $g$
Unary Relation Constant: $p$
Binary Relation Constant: $q$
Terms

A *term* is either a variable, an object constant, or a functional term (defined shortly).

Terms represent objects.

Terms are analogous to noun phrases in natural language.
Functional Terms

A functional term is an expression formed from an \( n \)-ary function constant and \( n \) terms enclosed in parentheses and separated by commas.

\[
\begin{align*}
& f(a) \\
& f(x) \\
& g(a, y)
\end{align*}
\]

Functional terms are terms and so can be nested.

\[
g(f(a), g(y,a))
\]
Sentences

Three types of sentences in Herbrand Logic:

Relational sentences - analogous to the simple sentences in natural language

Logical sentences - analogous to the logical sentences in natural language

Quantified sentences - sentences that express the significance of variables
Relational Sentences

A *relational sentence* is an expression formed from an $n$-ary relation constant and $n$ terms enclosed in parentheses and separated by commas.

$$q(a, f(a))$$

Relational sentences are *not* terms and *cannot* be nested in terms or relational sentences.

No! $q(a, q(a, y))$ No!
Logical Sentences

Logical sentences in Herbrand Logic are analogous to those in Propositional Logic.

\(-q(a,b))
(p(a) \land p(b))
(p(a) \lor p(b))
(q(x,y) \Rightarrow q(y,x))
(q(x,y) \iff q(y,x))
Quantified Sentences

Universal sentences assert facts about all objects.

\[(\forall x. (p(x) \Rightarrow q(x, f(x))))\]

Existential sentence assert the existence of objects with given properties.

\[(\exists x. (p(x) \land q(x, f(x))))\]

Quantified sentences can be nested within other sentences.

\[(\forall x. p(x)) \lor (\exists x. q(x, f(x)))\]
\[(\forall x. (\exists y. q(f(x), y)))\]
Parentheses

Parentheses can be removed when precedence allows us to reconstruct sentences correctly.

Precedence relations same as in Propositional Logic with quantifiers being of *higher* precedence than logical operators.

\[
\forall x. p(x) \Rightarrow q(x,x) \Rightarrow (\forall x. p(x)) \Rightarrow q(x,x) \\
\exists x. p(x) \land q(x,x) \Rightarrow (\exists x. p(x)) \land q(x,x)
\]
Semantics
Herbrand Base

The *Herbrand base* for a Herbrand language is the set of all ground relational sentences that can be formed from the vocabulary of the language.
Herbrand Base Without Functions

Object Constants: $a, b$
Unary Relation Constant: $p$
Binary Relation Constant: $q$

Herbrand Base:

$$\{ p(a), p(b), q(a,a), q(a,b), q(b,a), q(b,b) \}$$
Herbrand Base With Functions

Object Constants: $a$
Unary Function Constant: $f$
Unary Relation Constant: $p$

Herbrand Base:

\[
\{ p(a), p(f(a)), p(f(f(a))), \ldots \}\]
A truth assignment is an association between ground atomic sentences and the truth values true or false. As with Propositional Logic, we use 1 as a synonym for true and 0 as a synonym for false.

\[
\begin{align*}
  p(a)^i &= 1 \\
  p(b)^i &= 0 \\
  q(a,a)^i &= 1 \\
  q(a,b)^i &= 0 \\
  q(b,a)^i &= 1 \\
  q(b,b)^i &= 0
\end{align*}
\]
Everything Else

All other notions are defined the same as in Relational Logic.

The main difference is that now we have truth assignments that are infinitely large and there are infinitely many of them. This means that it is no longer possible in general to determine properties and relationships like logical entailment in finite time.
Example - Peano Arithmetic
Peano Arithmetic

In *Peano Arithmetic*, we are concerned with all of the natural numbers, not just a finite subset, and functions do not wrap around as in Modular Arithmetic.

\[
\begin{align*}
0+0 &= 0 \\
1+0 &= 1 \\
2+0 &= 2 \\
3+0 &= 3 \\
0+1 &= 1 \\
1+1 &= 2 \\
2+1 &= 3 \\
3+1 &= 5 \\
0+2 &= 2 \\
1+2 &= 3 \\
2+2 &= 4 \\
3+2 &= 6 \\
0+3 &= 3 \\
1+3 &= 4 \\
2+3 &= 5 \\
3+3 &= 7 \\
\ldots & \quad \ldots & \quad \ldots & \quad \ldots & \quad \ldots
\end{align*}
\]
Possible Representations

Object Constants: 0, 1, 2, …

Ground Terms: 0, 1, 2, …
Possible Representations

Object Constants: 0, 1, 2, …

Ground Terms: 0, 1, 2, …

Object Constant: 0
Unary Function Constant: s

Ground Terms: 0, s(0), s(s(0)), …
Signature

Object Constant: 0

Unary Function Constant: $s$

Binary Relation Constant:
\[same\] - the first and second arguments are identical

Ternary Relation Constant:
\[plus\] - the third argument is the sum of the first two
\[times\] - third argument is the product of the first two
Axiomatization

Enumerating ground relational data impossible

\[
\begin{align*}
\text{same}(0,0) &\quad \text{plus}(0,0,0) &\quad \text{times}(0,0,0) \\
\neg \text{same}(0,s(0)) &\quad \neg \text{plus}(0,0,s(0)) &\quad \neg \text{times}(0,0,s(0)) \\
\neg \text{same}(0,s(s(0))) &\quad \neg \text{plus}(0,0,s(s(0))) &\quad \neg \text{times}(0,0,s(s(0))) \\
\ldots &\quad \ldots &\quad \ldots \\
\end{align*}
\]

Solution - write logical and quantified sentences
Same

Definition:

\[ \forall x. \text{same}(x,x) \]

\[ \forall x. (\neg \text{same}(0,s(x)) \land \neg \text{same}(s(x),0)) \]

\[ \forall x. \forall y. (\neg \text{same}(x,y) \Rightarrow \neg \text{same}(s(x), s(y))) \]
Same

Definition:

\[ \forall x. \text{same}(x,x) \]
\[ \forall x. (\neg \text{same}(0,s(x)) \land \neg \text{same}(s(x),0)) \]
\[ \forall x. \forall y. (\neg \text{same}(x,y) \implies \neg \text{same}(s(x), s(y))) \]

Examples:

\text{same}(0,0)
\text{same}(s(0), s(0))
\text{same}(s(s(0)), s(s(0)))
\ldots
Same

Definition:

\[ \forall x. \text{same}(x,x) \]

\[ \forall x. (\neg \text{same}(0,s(x)) \land \neg \text{same}(s(x),0)) \]

\[ \forall x. \forall y. (\neg \text{same}(x,y) \Rightarrow \neg \text{same}(s(x), s(y))) \]

Examples:

\[ \text{same}(0,0) \]

\[ \neg \text{same}(0, s(0)) \]

\[ \text{same}(s(0), s(0)) \]

\[ \neg \text{same}(0, s(s(0))) \]

\[ \text{same}(s(s(0)), s(s(0))) \]

\[ \neg \text{same}(0, s(s(s(0)))) \]

\[ \text{same}(s(s(s(0))), s(s(s(0)))) \]

\[ \cdots \]
Same

Definition:

\[ \forall x. \text{same}(x,x) \]

\[ \forall x. (\neg \text{same}(0,s(x)) \land \neg \text{same}(s(x),0)) \]

\[ \forall x. \forall y. (\neg \text{same}(x,y) \Rightarrow \neg \text{same}(s(x), s(y))) \]

Examples:

\[ \neg \text{same}(0,s(0)) \]

\[ \text{same}(0,0) \]

\[ \neg \text{same}(0,s(s(0))) \]

\[ \text{same}(s(0),s(0)) \]

\[ \neg \text{same}(s(0),s(s(0))) \]

\[ \text{same}(s(s(0)),s(s(0))) \]

\[ \neg \text{same}(s(s(0)),s(s(s(0)))) \]

\[ \text{…} \]
Addition

Identity:

\( \forall y. \text{plus}(0,y,y) \)

Successor:

\( \forall x. \forall y. \forall z. (\text{plus}(x,y,z) \Rightarrow \text{plus}(s(x),y,s(z))) \)

Functionality:

\( \forall x. \forall y. \forall z. \forall w. (\text{plus}(x,y,z) \land \text{plus}(x,y,w) \Rightarrow \text{same}(z,w)) \)
Multiplication

Identity:

\( \forall y.\text{times}(0,y,0) \)

Successor:

\( \forall x.\forall y.\forall z. (\text{times}(x,y,z) \land \text{plus}(y,z,w) \Rightarrow \text{times}(s(x),y,w)) \)

Functionality:

\( \forall x.\forall y.\forall z.\forall w. (\text{times}(x,y,z) \land \text{times}(x,y,w) \Rightarrow \text{same}(z,w)) \)
Diophantine Equations

A *polynomial equation* is an algebraic expression composed using only addition and multiplication and exponentiation with fixed exponents.

\[ 3x^2 + 2y = 2z^3 \]

A *natural Diophantine equation* is a polynomial in which the values of variables are restricted to natural numbers.

\[ x = 2 \]
\[ y = 2 \]
\[ z = 2 \]
Diophantine Equations in Peano

Diophantine equation:

\[ 3x^2 = 1 \]

Diophantine equation in Peano Arithmetic:

\[ \forall x. \forall y. \forall z. \]
\[ (times(x,x,y) \land times(s(s(s(0))),y,z) \implies same(z,s(0))) \]

Diophantine equation with “syntactic sugar”: 

\[ \forall x. \forall y. \forall z. (x*x=y \land 3*y=z \implies z=1) \]
\[ \forall x. \forall y. \forall z. (3*x*x=1) \]
Example - Linked Lists
Linked Lists

Flat Lists:

\[ [a, b, c, d] \]

Nested Lists:

\[ [a, [a, b], b, [c, d], d] \]

Linked List:
Representation

Example

```
  a ——> b ——> c ——> d
```

Representation as Term

\[
\text{cons}(a,\text{cons}(b,\text{cons}(c,\text{cons}(d,\text{nil}))))
\]
Signature

Object Constants: \( a, b, c, d, \text{nil} \)

Binary Function Constant: \textit{cons} 

Binary Relation Constants: \textit{member, among} 

Ternary Relation Constant: \textit{append}
Membership

Example:

\[ member(b, \text{cons}(a,\text{cons}(b,\text{cons}(c,\text{nil})))) \]

Definition:

\[ \forall x. \forall y. member(x, \text{cons}(x,y)) \]
\[ \forall x. \forall y. \forall z. (member(x,z) \Rightarrow member(x,\text{cons}(y,z))) \]
Concatenation

Example:
\[
\text{append}(\text{cons}(a,\text{cons}(b,nil)), \text{cons}(c,\text{cons}(d,nil)), \text{cons}(a,\text{cons}(b,\text{cons}(c,\text{cons}(d,nil)))))
\]

Definition:
\[
\forall x. \forall y. \text{append}(\text{nil},y,y) \\
\forall x. \forall y. \forall z. \forall w. (\text{append}(x,y,w) \\
\quad \Rightarrow \text{append}(\text{cons}(x,y),z,\text{cons}(x,w)))
\]
Containment

Example:

\[
\text{among}(c, \text{cons}(a, \\
\text{cons}(\text{cons}(b, \text{cons}(c, \text{nil})), \\
\text{cons}(d, \text{nil}))))
\]

Definition:

\[
\forall x. \text{among}(x, x) \\
\forall x. \forall y. \forall z. (\text{among}(x, y) \lor \text{among}(x, z) \rightarrow \\
\text{among}(x, \text{cons}(y, z)))
\]
Example - Pseudo English
Pseudo English

Good Sentences:
  Mary likes Pat.
  Mary likes Pat and Quincy.
  Pat and Quincy like Mary.

Bad Sentences:
  Mary Pat likes.
  Likes and Mary Pat Quincy.
Backus Naur Form

<sentence> ::= <np> <vp>
<np> ::= <noun>
<np> ::= <noun> "and" <noun>
<vp> ::= <verb> <np>
<noun> ::= "mary" | "pat" | "puincy"
<verb> ::= "like" | "likes"
Logical Grammar

\[ np(x) \land vp(y) \land append(x, y, z) \Rightarrow sentence(z) \]

\[ noun(x) \Rightarrow np(x) \]
\[ noun(x) \land noun(y) \land append(x, and, z) \land append(z, y, w) \Rightarrow np(w) \]

\[ verb(x) \land np(y) \land append(x, y, z) \Rightarrow vp(z) \]

\[ noun(mary) \]
\[ noun(pat) \]
\[ noun(quincy) \]

\[ verb(like) \]
\[ verb(likes) \]
Logical Grammar

\[ np(x) \land vp(y) \land append(x,y,z) \Rightarrow sentence(z) \]

\[ noun(x) \Rightarrow np(x) \]
\[ noun(x) \land noun(y) \land append(x,\text{and},z) \land append(z,y,w) \Rightarrow np(w) \]

\[ verb(x) \land np(y) \land append(x,y,z) \Rightarrow vp(z) \]

\[ noun(mary) \]
\[ noun(pat) \]
\[ noun(quincy) \]

\[ verb(like) \]
\[ verb(likes) \]
Logical Grammar

\[ np(x) \land vp(y) \land append(x,y,z) \Rightarrow sentence(z) \]

\[ noun(x) \Rightarrow np(x) \]
\[ noun(x) \land noun(y) \land append(x,and,z) \land append(z,y,w) \Rightarrow np(w) \]

\[ verb(x) \land np(y) \land append(x,y,z) \Rightarrow vp(z) \]

\[ noun(mary) \]
\[ noun(pat) \]
\[ noun(quincy) \]

\[ verb(like) \]
\[ verb(likes) \]
Logical Grammar

\[ np(x) \land vp(y) \land append(x,y,z) \Rightarrow sentence(z) \]

\[ noun(x) \Rightarrow np(x) \]
\[ noun(x) \land noun(y) \land append(x,and,z) \land append(z,y,w) \Rightarrow np(w) \]

\[ verb(x) \land np(y) \land append(x,y,z) \Rightarrow vp(z) \]

\[ noun(mary) \]
\[ noun(pat) \]
\[ noun(quincy) \]
\[ verb(like) \]
\[ verb(likes) \]
Logical Grammar

\[ np(x) \land vp(y) \land append(x,y,z) \Rightarrow sentence(z) \]

\[ noun(x) \Rightarrow np(x) \]
\[ noun(x) \land noun(y) \land append(x,\textit{and},z) \land append(z,y,w) \Rightarrow np(w) \]

\[ verb(x) \land np(y) \land append(x,y,z) \Rightarrow vp(z) \]

\[ noun(\textit{mary}) \]
\[ noun(\textit{pat}) \]
\[ noun(\textit{quincy}) \]

\[ verb(\textit{like}) \]
\[ verb(\textit{likes}) \]
Examples

Sentences:
√ Mary likes Pat.
√ Mary likes Pat and Quincy.
√ Pat and Quincy like Mary.

Not Sentences:
× Mary Pat likes.
× Likes and Mary Pat Quincy.
Glitch

Sentences:
   Mary likes Pat.
   Mary likes Pat and Quincy.
   Pat and Quincy like Mary.

Allowed but not sentences in natural English:
   Mary likes Pat.
   Pat and Quincy likes Mary.

How can we enforce subject-verb number agreement?
Augmented Logical Grammar

\[ np(x,w) \land vp(y,w) \land append(x,y,z) \Rightarrow sentence(z) \]

\[ noun(x) \Rightarrow np(x,0) \]
\[ noun(x) \land noun(y) \land append(x,and,z) \land append(z,y,w) \Rightarrow np(w,1) \]

\[ verb(x,w) \land np(y,v) \land append(x,y,z) \Rightarrow vp(z,w) \]

\[ noun(mary) \]
\[ noun(pat) \]
\[ noun(quincy) \]

\[ verb(like,1) \]
\[ verb(likes,0) \]
Augmented Logical Grammar

\[ np(x,w) \land vp(y,w) \land append(x,y,z) \Rightarrow sentence(z) \]

\[ noun(x) \Rightarrow np(x,0) \]
\[ noun(x) \land noun(y) \land append(x,\textit{and},z) \land append(z,y,w) \Rightarrow np(w,1) \]

\[ verb(x,w) \land np(y) \land append(x,y,z) \Rightarrow vp(z,w) \]

noun(mary)
noun(pat)
noun(quincy)

verb(like,1)
verb(likes,0)
Augmented Logical Grammar

\[ np(x,w) \land vp(y,w) \land append(x,y,z) \Rightarrow sentence(z) \]

\[ noun(x) \Rightarrow np(x,0) \]
\[ noun(x) \land noun(y) \land append(x,\text{and},z) \land append(z,y,w) \Rightarrow np(w,1) \]

\[ verb(x,w) \land np(y) \land append(x,y,z) \Rightarrow vp(z,w) \]

\[ noun(mary) \]
\[ noun(pat) \]
\[ noun(quincy) \]

\[ verb(like,1) \]
\[ verb(likes,0) \]
Augmented Logical Grammar

\[ np(x,w) \land vp(y,w) \land append(x,y,z) \Rightarrow sentence(z) \]

\[ noun(x) \Rightarrow np(x,0) \]
\[ noun(x) \land noun(y) \land append(x,and,z) \land append(z,y,w) \Rightarrow np(w,1) \]

\[ verb(x,w) \land np(y) \land append(x,y,z) \Rightarrow vp(z,w) \]

* noun(mary) *
* noun(pat) *
* noun(quincy) *

* verb(like,1) *
* verb(likes,0) *
Augmented Logical Grammar

\[ np(x,w) \land vp(y,w) \land append(x,y,z) \Rightarrow sentence(z) \]

\[ noun(x) \Rightarrow np(x,0) \]
\[ noun(x) \land noun(y) \land append(x,\text{and},z) \land append(z,y,w) \Rightarrow np(w,1) \]

\[ verb(x,w) \land np(y) \land append(x,y,z) \Rightarrow vp(z,w) \]

\[ noun(\text{mary}) \]
\[ noun(\text{pat}) \]
\[ noun(\text{quincy}) \]

\[ verb(\text{like},1) \]
\[ verb(\text{likes},0) \]
Example - Metalevel Logic
Talking Head
Metalevel Logic

Basic idea: represent expressions in Propositional Logic as terms in Herbrand Logic, write Herbrand sentences to define basic concepts of Propositional Logic, prove metatheorems.

NB: We can extend to Herbrand Logic as well. The formalization is messier, and some nasty problems need to be handled (notably paradoxes).
Syntactic Metavocabulary

Object Constants (propositions)

\[ p, q, r \]
Syntactic Metavocabulary

Object Constants (propositions)

\( p, q, r \)

Function constants

\( \text{not}(p) \quad \text{if}(p,q) \)
\( \text{and}(p,q) \quad \text{iff}(p,q) \)
\( \text{or}(p,q) \)
Syntactic Metavocabulary

Object Constants (propositions)

\[ p, q, r \]

Function constants

\[ \text{not}(x), \text{if}(x,y) \]
\[ \text{and}(x,y), \text{iff}(x,y) \]
\[ \text{or}(x,y) \]

Relation Constants

\[ \text{proposition}(x), \text{implication}(x) \]
\[ \text{negation}(x), \text{biconditional}(x) \]
\[ \text{conjunction}(x), \text{sentence}(x) \]
\[ \text{disjunction}(x) \]
Syntactic Metadefinitions

proposition(p)
proposition(q)
proposition(r)

negation(not(x)) ⇔ sentence(x)
conjunction(and(x,y)) ⇔ sentence(x) ∧ sentence(y)
disjunction(or(x,y)) ⇔ sentence(x) ∧ sentence(y)
implication(if(x,y)) ⇔ sentence(x) ∧ sentence(y)
equivalence(iff(x,y)) ⇔ sentence(x) ∧ sentence(y)

sentence(x) ⇔
  proposition(x) ∨ negation(x) ∨ conjunction(x) ∨
disjunction(x) ∨ implication(x) ∨ biconditional(x)
Rules of Inference

And Introduction:

\[ \forall x. (\text{sentence}(x) \land \text{sentence}(y) \Rightarrow ai(x,y,\text{and}(x,y))) \]

And Elimination:

\[ \forall x. (\text{sentence}(x) \land \text{sentence}(y) \Rightarrow ae(\text{and}(x,y),x)) \]
\[ \forall x. (\text{sentence}(x) \land \text{sentence}(y) \Rightarrow ae(\text{and}(x,y),y)) \]
Metatheorems

Validity of Axiom Scemata:

\[ \text{valid}(\text{or}(x, \text{not}(x))) \iff \text{sentence}(x) \]

Soundness:

\[ \text{proves}(x,y) \iff \text{entails}(x,y) \]

Deduction Theorem:

\[ \text{proves}(\text{and}(x,y),z) \iff \text{proves}(x,\text{implies}(y,z)) \]
Herbrand Logic in Herbrand Logic

In similar fashion, we can axiomatize Herbrand Logic in Herbrand Logic. But with some care to avoid paradoxes (sentences that are both true and false) and self-contradictory terms.

Paradox:

This sentence is false.

Self-contradictory terms:
This set of all sets that do not contain themselves
Undecidability
Good News

HL is highly expressive.

For example, possible to axiomatize arithmetic as a finite set of axioms.

This is not possible with Relational Logic and in other logics (e.g. First-Order Logic).
Bad News

The question of unsatisfiability and the question of logical entailment for HL are not even semidecidable.
Diophantine Equations

Diophantine equation:

\[ 3x^2 = 1 \]

Diophantine equation in Peano Arithmetic:

\[ \exists x, \exists y. (times(x,x,y) \land times(s(s(s(0))),y,s(0))) \]

Unsolvability Question:

Axioms of Arithmetic \( \cup \)
\[ \{ \exists x, \exists y. (times(x,x,y) \land times(s(s(s(0))),y,s(0))) \} \]
HL Not Even Semidecidable

It is known that the question of unsolvability of Diophantine equations is \textit{not} semidecidable.

Since we can represent the question of unsolvability of Diophantine equations in Herbrand Logic, if we could determine the unsatisfiability of sentences in Herbrand Logic, we could answer questions of unsolvability of Diophantine equations.