# The Language of Logic

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

## Overview

- Motivation and preliminaries.
- Propositional logic.
- Predicate logic.
- Example.

# Motivation and Preliminaries

## Motivation

- Precise language.
  - Formulating statements.
  - Designating objects.

  Resolve ambiguities.

- Intellectual framework.
  - Sound reasoning.
  - Sound arguing.

  Guide thinking.

Let's focus on the language aspect first.

# Formality vs Informality

- Symbolic Arithmetic:

  "The product of $x$ and of the sum of $y$ and $z$ if the sum of the products of $x$ and $y$ and of $x$ and $z$":

$$x * (y + z) = x * y + x * z.$$

- Symbolic Logic:

  "Between every natural number and its double (the bounds included) there is at least one prime number":

$$\forall x : x \in \mathbb{N} \Rightarrow \exists y : x \leq y \wedge y \leq 2x \wedge y \text{ is prime.}$$

Formal notation is a tool to write clear and concise statements.

# Preliminaries

- Truth values (Boolean values):

  true, false.

- Formula:

  Syntactic phrase whose semantics is a truth value

  - Syntax: external representation.
  - Semantics: underlying meaning.

$$\boxed{\text{Formula} \xrightarrow{\text{Meaning}} \text{Truth Value}}$$

## Operational Interpretation

```
public interface Formula
{
  boolean eval() throws EvalException;
}

Formula formula = ...;
boolean meaning = formula.eval();
```

Evaluation of a formula returns a truth value (normally).

# Equivalence of Formulas

$$A \text{ iff } B$$

- Formulas $A$ and $B$ have the same meaning in any context.
- Operationally, then the following always holds:

$$A\texttt{.eval() == }B\texttt{.eval()}$$

# Propositional Logic

# Propositional Logic

- Logic of formula composition:
  - Basic formulas are "black boxes": only truth value known.
  - Combination of simpler formulas to more complex ones.
- Logical Connective (Junktor)
  - Syntactic operator that combines formulas to a new formula.
  - F, T, $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$.

Rules for construction of formulas and computation of their meanings.

## Syntax of Propositional Logic

Let $A$ and $B$ be formulas. Then the following are formulas:

- Logical constants: F, T.

  “false”, “true”.

- Negation: $(\neg A)$.

  “not $A$”.

Connectives of arity 0 and 1.

# Syntax of Propositional Logic (Continued)

- Conjunction: $(A \land B)$.

  “$A$ and $B$”.

- Disjunction: $(A \lor B)$.

  “$A$ and $B$”.

- Implication: $(A \Rightarrow B)$.

  “$A$ implies $B$”.

- Equivalence: $(A \Leftrightarrow B)$.

  “$A$ is equivalent to $B$”.

Connectives of arity 2.

# Hierarchical Construction of Formulas

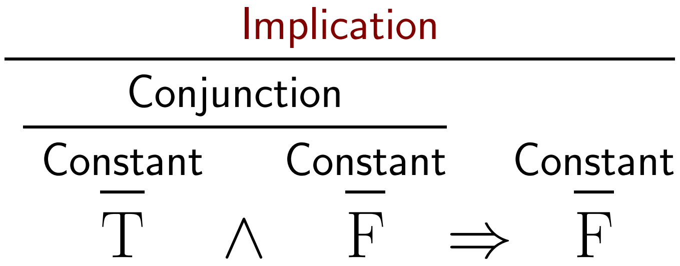
$$((\mathrm{T} \land \mathrm{F}) \Rightarrow \neg(\mathrm{F} \lor \neg \mathrm{F}))$$

$$\overbrace{((\overbrace{\overbrace{\mathrm{T}}^{\text{Constant}} \land \overbrace{\mathrm{F}}^{\text{Constant}}}^{\text{Conjunction}}) \Rightarrow \overbrace{\neg(\overbrace{\overbrace{\mathrm{F}}^{\text{Constant}} \lor \overbrace{\neg \overbrace{\mathrm{F}}^{\text{Constant}}}^{\text{Negation}}}^{\text{Disjunction}})}^{\text{Negation}})}^{\text{Implication}}$$

If syntactic structure clear, parentheses may be omitted.

# Ambiguous Syntax

$$\mathrm{T} \wedge \mathrm{F} \Rightarrow \mathrm{F}$$

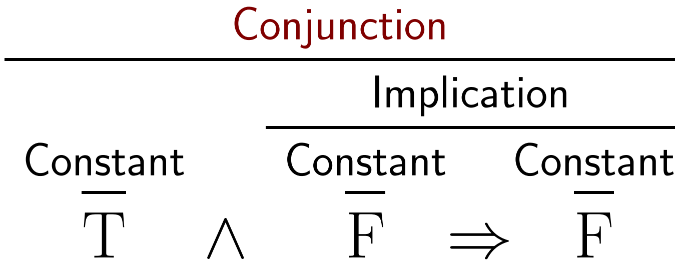


Implication

Conjunction

Constant Constant Constant

$\mathrm{T} \quad \wedge \quad \mathrm{F} \quad \Rightarrow \quad \mathrm{F}$



Conjunction

Implication

Constant Constant Constant

$\mathrm{T} \quad \wedge \quad \mathrm{F} \quad \Rightarrow \quad \mathrm{F}$

# Logical Constants

$$\mathrm{F}, \mathrm{T}.$$

- Other syntactic forms:
  - 0, 1.
  - wrong, right.
  - incorrect, correct.
  - false, true.
- Semantics:
  - Meaning of F is false.
  - Meaning of T is true.

# Operational Interpretation

```
public final class False implements Formula
{
  public boolean eval() throws EvalException
  {
    return false;
  }
}

public final class True implements Formula
{
  public boolean eval() throws EvalException
  {
    return true;
  }
}
```

# Negations

$$(\neg A).$$

- Other syntactic forms:
  - $\overline{A}$, $\sim A$, $-A$, $!A$;
  - non-$A$, "not $A$", "never $A$", "in no case $A$";
  - `not(`$A$`).`
- Semantics: $\neg A$ is true, if and only if $A$ is false.

| $A$ | $\neg A$ |
|---|---|
| false | true |
| true | false |

# Operational Interpretation

```
public final class Not implements Formula
{
  private Formula formula;

  ...

  public boolean eval() throws EvalException
  {
    if (formula.eval())
      return false;
    else
      return true;
  }
}
```

# Inversion of Negation

Proposition: For every formula $A$, we have

$$\neg\neg A \text{ iff } A.$$

Proof: Let $A$ be an arbitrary formula. We have to show that the meaning of $\neg\neg A$ is the same as the meaning of $A$, i.e., that they have the same truth values.

| $A$ | $\neg A$ | $\neg\neg A$ |
|---|---|---|
| false | true | false |
| true | false | true |

Since the last column coincides with the first column, we are done.

# Conjunctions

$$(A \wedge B)$$

- Other syntactic forms:
  - $A, B$; $A * B$; $A \mathbin{\&} B$; $A$ `&&` $B$;
  - "$A$ and $B$", "$A$ as well as $B$" ("sowohl $A$ als auch $B$");
  - `and(`$A$`, `$B$`).`
- Semantics: $A \wedge B$ is true, if and only if both $A$ and $B$ are true.

| $A$ | $B$ | $A \wedge B$ |
|---|---|---|
| false | false | false |
| false | true | false |
| true | false | false |
| true | true | true |

# Operational Interpretation

```
public final class And implements Formula
{
  private Formula formula0; private Formula formula1;

  public boolean eval() throws EvalException
  {
    if (formula0.eval())
      if (formula1.eval())
        return true;
      else
        return false;
    else
      return false;
  }
}
```

## Conjunctive Laws

Proposition:

1. Conjunction is commutative: for all formulas $A$ and $B$,
$$A \wedge B \text{ iff } B \wedge A.$$
2. Conjunction is associative: for all formulas $A$, $B$, and $C$,
$$A \wedge (B \wedge C) \text{ iff } (A \wedge B) \wedge C.$$

# Proof

Proof: We prove associativity. Take arbitrary formulas $A$, $B$, and $C$. We show that $A \wedge (B \wedge C)$ and $(A \wedge B) \wedge C$ have same truth value.

| $A$ | $B$ | $C$ | $B \wedge C$ | $A \wedge (B \wedge C)$ | $A \wedge B$ | $(A \wedge B) \wedge C$ |
|---|---|---|---|---|---|---|
| false | false | false | false | false | false | false |
| false | false | true | false | false | false | false |
| false | true | false | false | false | false | false |
| false | true | true | true | false | false | false |
| true | false | false | false | false | false | false |
| true | false | true | false | false | false | false |
| true | true | false | false | false | true | false |
| true | true | true | true | true | true | true |

# Notation

Because of associativity, setting of parentheses in nestings of conjunctive formulas do not matter.

- $A \wedge B \wedge C$
  - $A \wedge (B \wedge C)$
  - $(A \wedge B) \wedge C$
- $A_0 \wedge A_1 \wedge \ldots \wedge A_{n-1}$
- `and(`$A_0$`, `$A_1$`, ..., `$A_{n-1}$`)`

# Disjunctions

$$(A \vee B)$$

- Other syntactic forms:
  - $A + B$, $A \mid B$, $A$ `||` $B$;
  - "$A$ or $B$";
  - `or(`$A$`,` $B$`)`.
- Semantics: $A \vee B$ is false, if and only if both $A$ and $B$ are false.

| $A$ | $B$ | $A \vee B$ |
|---|---|---|
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | true |

# Operational Interpretation

```
public final class Or implements Formula
{
  private Formula formula0; private Formula formula1;

  public boolean eval() throws EvalException
  {
    if (formula0.eval())
      return true;
    else
      if (formula1.eval())
        return true;
      else
        return false;
  }
}
```

## Disjunctive Laws

Proposition:

1. Disjunction is commutative: for all formulas $A$ and $B$

$$A \lor B \text{ iff } B \lor A.$$

2. Disjunction is associative: for all formulas $A$, $B$, and $C$, we have

$$A \lor (B \lor C) \text{ iff } (A \lor B) \lor C.$$

Proof: by truth tables.

## De Morgan's Laws

Proposition: For all formulas $A$ and $B$, the following holds:

$$\begin{aligned}\neg(A \land B) &\text{ iff } \neg A \lor \neg B,\\ \neg(A \lor B) &\text{ iff } \neg A \land \neg B.\end{aligned}$$

Proof: by truth table.

Consequence: For all formulas $A$ and $B$, we have:

$$A \lor B \text{ iff } \neg(\neg A \land \neg B).$$

Proof: by inversion of negation.

Disjunction can thus be defined by conjunction and negation.

## Notation

Because of associativity, setting of parentheses in nestings of disjunctive formulas do not matter.

- $A \vee B \vee C$
  - $A \vee (B \vee C)$
  - $(A \vee B) \vee C$
- $A_0 \vee A_1 \vee \ldots \vee A_{n-1}$
- $\texttt{or(}A_0\texttt{, } A_1\texttt{, } \ldots\texttt{, } A_{n-1}\texttt{)}$

## Exclusive Disjunction

$$(A \text{ xor } B)$$

- Read: either $A$ or $B$.
- Semantics: $A \text{ xor } B$ is true, iff exactly one of $A$ or $B$ is true.

| $A$ | $B$ | $A \text{ xor } B$ |
|---|---|---|
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | false |

- Exclusive disjunction can thus be defined by other connectives.

$$(A \text{ xor } B) \text{ iff } (A \wedge \neg B) \vee (\neg A \wedge B).$$

# Implication

$$(A \Rightarrow B)$$

- Other syntactic forms:
  - "$A$ implies $B$"; "$B$ follows from $A$"
  - "if $A$, then $B$"; "$B$, (only) if $A$"
  - "$A$ is sufficient for $B$"; "$B$ is necessary for $A$"
  - `implies(`$A$`, `$B$`).`
- Semantics: $A \Rightarrow B$ is false if and only if $A$ is true and $B$ is false.

| $A$ | $B$ | $A \Rightarrow B$ |
|---|---|---|
| false | false | true |
| false | true | true |
| true | false | false |
| true | true | true |

# Operational Interpretation

```
public final class Implies implements Formula
{
  private Formula formula0; private Formula formula1;

  public boolean eval() throws EvalException
  {
    if (formula0.eval())
      if (formula1.eval())
        return true;
      else
        return false;
    else
      return true;
  }
}
```

## Implicative Laws

Proposition: For all formulas $A$ and $B$, we have

$$\begin{aligned} A \Rightarrow B &\text{ iff } \neg A \vee B \\ A \Rightarrow B &\text{ iff } \neg B \Rightarrow \neg A \\ \neg(A \Rightarrow B) &\text{ iff } A \wedge \neg B. \end{aligned}$$

Implication can thus be defined by negation and disjunction.

Proposition: Implication is not associative.
Proof: by construction of a counterexample:

- Meaning of $\mathrm{F} \Rightarrow (\mathrm{F} \Rightarrow \mathrm{F})$ is true.
- Meaning of $(\mathrm{F} \Rightarrow \mathrm{F}) \Rightarrow \mathrm{F}$ is false.

# Equivalences

$$(A \Leftrightarrow B)$$

- Other syntactic forms:
  - $A = B$, $A \sim B$;
  - "$A$ and $B$ are equivalent", "$A$ if and only if $B$", "$A$ iff $B$";
  - "$A$ is necessary and sufficient for $B$";
  - `equiv(`$A$`, `$B$`).`
- Semantics: $A \Leftrightarrow B$ is true iff $A$ and $B$ have the same truth value.

| $A$ | $B$ | $A \Leftrightarrow B$ |
|---|---|---|
| false | false | true |
| false | true | false |
| true | false | false |
| true | true | true |

# Operational Interpretation

```
public final class Equiv implements Formula
{
  private Formula formula0; private Formula formula1;
  ...

  public boolean eval() throws EvalException
  {
    return (formula0.eval() == formula1.eval());
  }
}
```

## Equivalence Laws

Proposition: For all formulas $A$ and $B$, the following holds:

$$A \Leftrightarrow B \text{ iff } B \Leftrightarrow A$$
$$A \Leftrightarrow B \text{ iff } (A \land B) \lor (\neg A \land \neg B)$$
$$A \Leftrightarrow B \text{ iff } (A \Rightarrow B) \land (B \Rightarrow A)$$

Equivalence can thus be defined by implication and conjunction.

Proposition: Equivalence is not associative.
Proof: by construction of a counterexample.

## Propositional Logic

- Semantics:

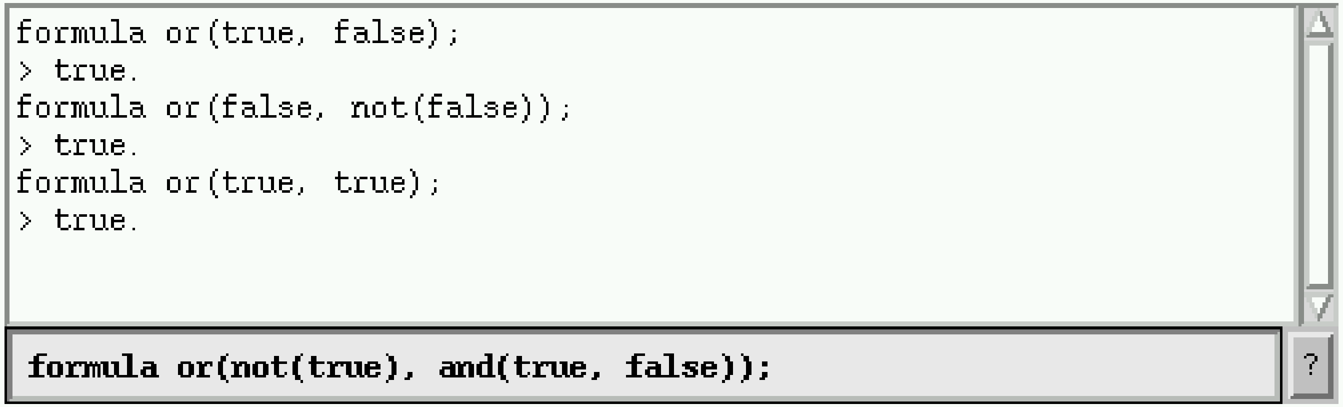
| $A$ | $B$ | F | T | $\neg A$ | $A \wedge B$ | $A \vee B$ | $A \Rightarrow B$ | $A \Leftrightarrow B$ |
|---|---|---|---|---|---|---|---|---|
| false | false | false | true | true | false | false | true | true |
| false | true | false | true | true | false | true | true | false |
| true | false | false | true | false | false | true | false | false |
| true | true | false | true | false | true | true | true | true |

- Relationships:

$$\begin{aligned} A \vee B &\text{ iff } \neg(\neg A \wedge \neg B) \\ A \Rightarrow B &\text{ iff } \neg A \vee B \\ A \Leftrightarrow B &\text{ iff } (A \Rightarrow B) \wedge (B \Rightarrow A) \end{aligned}$$

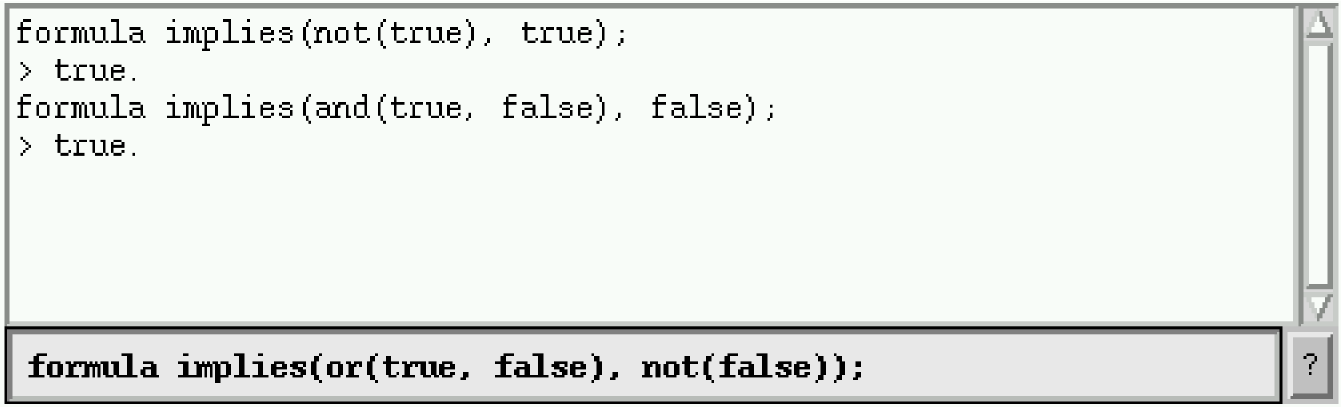
All connectives can be ultimately reduced to negation and conjunction.

# Logic Evaluator



```
formula or(true, false);
> true.
formula or(false, not(false));
> true.
formula or(true, true);
> true.
```

```
formula or(not(true), and(true, false));
```



```
formula implies(not(true), true);
> true.
formula implies(and(true, false), false);
> true.
```

```
formula implies(or(true, false), not(false));
```

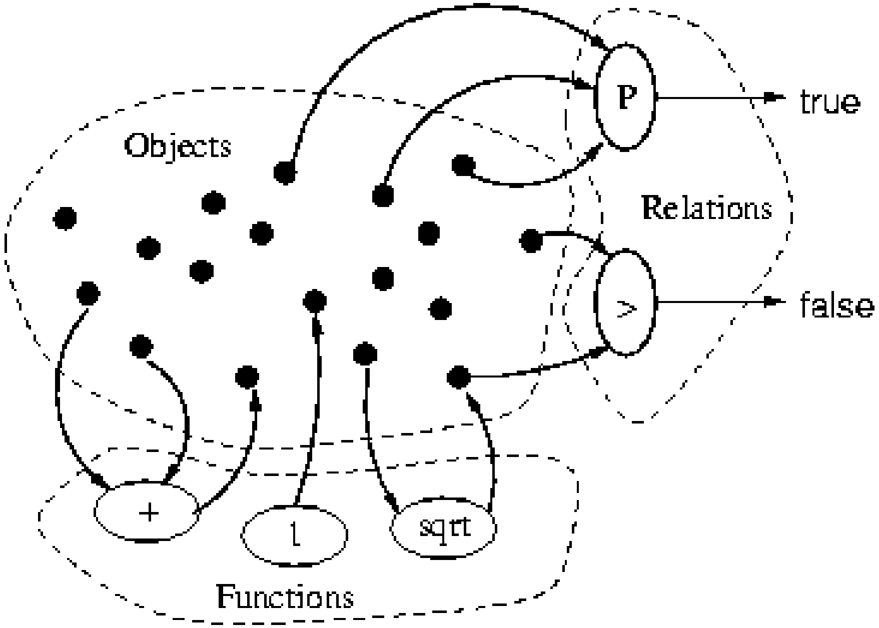
# Predicate Logic

# Predicate Logic

Superset of propositional logic that describes values and properties of a domain consisting of:

1. A collection of values (objects, entities).
2. A collection of functions (mappings):
   - Takes a certain number of values (the arguments);
   - Returns a value (the result);
   - Number of arguments is the arity of the function;
   - Zero-arity functions are called (object) constants.
3. A collection of predicates (relations, properties, attributes):
   - Takes a certain number of values (the arguments);
   - Returns a truth value;
   - Number of arguments is the arity of the predicate.

# Domain



An abstract image of the real world.

## Example

- The set of natural numbers
  - constants “zero” and “one”, binary functions “addition”, “multiplication”;
  - binary predicate “is less than’;
- The people in Austria
  - functions “mother of” and “father of”;
  - unary predicates “is male” and “is female”; ternary predicate “are parents of”;
- The set of Java values of types `int` and `int[]`
  - function “indexed array access”;
  - predicate “is sorted”.

# Predicate Logic

Extension of propositional logic by

- Terms
  - Syntactic phrases denoting objects.
  - Variables.
  - Function applications.
- Atomic formulas
  - Formulas using terms to express facts about particular objects of a domain.
- Quantified formulas
  - Formulas expressing facts about all objects of a domain.

# Terms

## Definition: Term

A syntactic phrase whose semantics is a value (of the domain being considered).

$$\boxed{\text{Term} \xrightarrow{\text{Meaning}} \text{Domain Value}}$$

# Operational Interpretation

```
public interface Term
{
  Value eval() throws EvalException;
}

Term term = ...;
Value meaning = term.eval();
```

Evaluation of a term returns a domain value (normally).

# Variables and Assignments

Definition: Variable.

- A name that may represent any value of the domain.
- First-order: must not represent predicates or functions.

Example: $x$, $y$, ...

Definition: (Variable) assignment.

- A mapping of all variables to domain values.

Example: $[x \mapsto 1, y \mapsto 2, \ldots]$

# Operational Interpretation

```
public final class Variable implements Term
{
  private String variable;

  public Value eval() throws EvalException
  {
    Value value = Context.get(variable);
    if (value == null) throw
      new EvalException("no variable " + variable + " in context");
    return value;
  }
}
```

# Function Constants

Definition: Function constants.

- A name that may represent some function.
- The arity of the constant determines the arity of the functions it may stand for.

Examples:

- $0$, $1$, $3.14$, “Wolfgang Schreiner”, “Austria”, $\pi$ (object constants);
- $|\ |$, sin(), $\sqrt{\ }$, “mother of” (unary constants);
- $+$, $\circ$, $[\ ]$ (binary constants);

# Syntax of Terms

Proposition:

1. Every variable is a term.
2. If $fc$ is a function constant of arity $n$ and $t_0, \ldots, t_{n-1}$ are terms,

   $$fc(t_0, \ldots, t_{n-1})$$

   is a term, called an elementary term or function application.

Example:

- $0$, “Wolfgang Schreiner”, $\pi$ (constant terms);
- $\sqrt{2}$, sin($x$), sum(2, $s$), “the mother of Thomas” (prefix terms);
- $|1|$, $a \circ b$, $a[i]$, $1 + 2$ (infix terms);

## Natural Language Terms

"the roof of the house of her father"

$$\mathrm{roof}(\mathrm{house}(\mathrm{father}(\mathrm{she})))$$

- Variables:
  - she
- Unary function constants:
  - father
  - house
  - roof

# Semantics of Terms

Definition: The meaning of a term $t$ under an assignment $a$ is:

- If $t$ is a variable, then its meaning is the value to which the variable is mapped by the assignment.
- If $t$ is an elementary term $fc(t_0, \ldots, t_{n-1})$ then its meaning is the result of the application of the function $f$ denoted by the function constant $fc$ to the values of the terms $t_i$ for the given assignment.
  1. Determine values of the terms $t_i$ under $a$.
  2. Determine function $f$ denoted by constant $fc$.
  3. Apply $f$ to values and get domain value.

# Examples

Term $x + (y + 0)$.

- Domain “natural numbers”.
  - Object constant $0$ interpreted as “zero”.
  - Binary function constant $+$ interpreted as addition.
  - Assignment $[x \mapsto$“one”, $y \mapsto$“two”$]$. gives natural number “three” as meaning.
  - Assignment $[x \mapsto$“one”, $y \mapsto$“zero”$]$ gives natural number “one” as meaning.
- Domain “character strings”.
  - Object constant $0$ interpreted as “empty string”.
  - Binary function constant $+$ interpreted as string concatenation.
  - Assignment $[x \mapsto ``hi,\ '', y \mapsto ``babe'']$ gives string “hi, babe” as meaning.

Semantics of a term depends on domain and variable assignment.

# Operational Interpretation

```
public final class Application implements Term
{
  private String name; private Term[] arguments;

  public Value eval() throws EvalException
  {
    Function function = Model.getFunction(name, arguments.length);
    if (function == null) throw new EvalException("unknown function");
    Value[] values = new Value[arguments.length];
    for (int i=0; i< values.length; i++)
      values[i] = arguments[i].eval();
    return function.apply(values);
  }
}
```

## Predicate Constants

Definition: Predicate Constant.

- A name that may represent some predicate.
- The arity of the constant determines the arity of the predicates it may stand for.

Example:

- "is positive" (unary predicate constant);
- $\leq$, $|$ (binary predicate constants);
- "is father of" (binary predicate constant);
- "is a child of ... and of ..." (ternary predicate constant).

# Syntax of Atomic Formulas

## Definition: Atomic Formula

$$pc(t_0, \ldots, t_{n-1})$$

where $pc$ is a predicate constant of arity $n$ and $t_0, \ldots, t_{n-1}$ are terms.

## Example:

- “1 is positive”;
- $2 \leq \sqrt{x+3}$;
- $2|5$;
- “Thomas is the father of Susanne”;
- “Susanne is a child of Thomas and of Birgit”.

# Natural Language Atomic Formulas

“Bill Clinton is a better president than his predecessor”.

$$\mathrm{isBetterPresident}(\mathrm{BillClinton}, \mathrm{Predecessor}(\mathrm{BillClinton}))$$

- Function constants:
  - Object constant “Bill Clinton”.
  - Unary constant “Predecessor”.
- Predicate constants:
  - Binary constant “isBetterPresident”.

## Semantics of Atomic Formulas

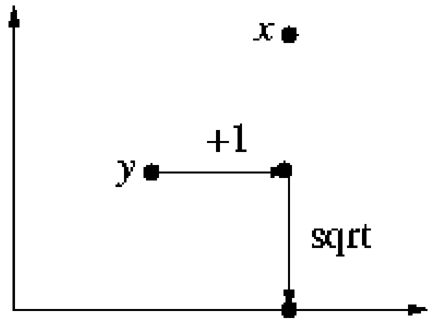
Definition: The meaning of an atomic formula $pc(t_0, \ldots, t_{n-1})$ under an assignment $a$

- The truth value of the predicate denoted by the predicate constant $pc$ for the values of the terms $t_i$ under the given assignment.
  1. Determine values of the terms $t_i$ under $a$.
  2. Determine predicate $p$ denoted by constant $pc$.
  3. Apply $p$ to values and get truth value.

# Example

Formula $x \leq \sqrt{y+1}$.

- Domain of real numbers.
  - Function and object constants interpreted as usual.
  - Assignment $[x \mapsto 3, y \mapsto 8]$ gives meaning true;
  - Assignment $[x \mapsto 4, y \mapsto 3]$ gives false.
- Domain of points in a plane.
  - $\leq$interpreted as "is not farther from the zero point than";



# Operational Interpretation

```
public final class Atomic implements Formula
{
  private String name; private Term[] arguments;

  public boolean eval() throws EvalException
  {
    Predicate predicate = Model.getPredicate(name,arguments.length);
    if (predicate == null) throw new EvalException("unknown predicate");
    Value[] values = new Value[arguments.length];
    for (int i=0; i<values.length; i++)
      values[i] = arguments[i].eval();
    return predicate.apply(values);
  }
}
```

# Equality

Predicate constant with the following properties, for all $x$, $y$, and $z$:

- Reflexivity: $x = x$;
- Symmetry: $x = y \Rightarrow y = x$;
- Transitivity: $(x = y \wedge y = z) \Rightarrow x = z$.
- Equality Axiom (every function constant $f$ or predicate constant $p$)

$$\begin{aligned} x = y \ &\Rightarrow\ f(\ldots,x,\ldots) = f(\ldots,y,\ldots); \\ x = y \ &\Rightarrow\ p(\ldots,x,\ldots) \Leftrightarrow p(\ldots,y,\ldots). \end{aligned}$$

First Order Predicate Logic with Equality.

## Quantified Formulas

Definition: Quantifier

- A syntactic operator that combines a variable and a phrase to form a new phrase.
- The meaning of the new phrase does not depend on the value of the variable in the given assignment.
- We call this variable bound by the quantifier.
- If all variables in a phrase are bound, the phrase is called closed.

Example: Formula $x < 1$.

- $x$ is free (not bound) in this formula.

# Universal Quantification

$$(\forall x : A)$$

- Variable $x$ bound by universal quantifier $\forall$.
- Other syntactic forms:
  - $\forall_x A$; $\bigwedge\limits_x A$;
  - "for all $x$ we have $A$"; "every $x$ has $A$"; "$A$, for all $x$";
  - `forall(`$x$`:`$A$`)`.
- Semantics: $(\forall x : A)$ is true, iff $A$ is true for every value of $x$.

  $A$ is true in every extension of the given assignment where $x$ is mapped to a domain value.

# Operational Interpretation

```
public final class ForAll implements Formula
{
  private String variable; private Term domain; private Formula formula;

  public boolean eval() throws EvalException
  {
    Iterator iterator = Model.iterator(domain);
    while (iterator.hasNext()) {
      Context.begin(variable, iterator.next());
      boolean result = formula.eval();
      Context.end();
      if (!result) return false; }
    return true;
  }
}
```

# Existential Quantification

$$(\exists x : A)$$

- Variable $x$ bound by existential quantifier $\exists$.
- Other syntactic forms:
  - $\exists_x\ A$; $\bigvee_x A$;
  - "there exists $x$ with $A$"; "there is some $x$ with $A$"; "some $x$ has $A$"; "$A$, for some $x$";
  - `exists(`$x$`:`$A$`)`.
- Semantics: $(\exists x : A)$ is true iff $A$ is true for some value of $x$.

  $A$ is true in some extension of the given assignment where $x$ is mapped to a domain value.

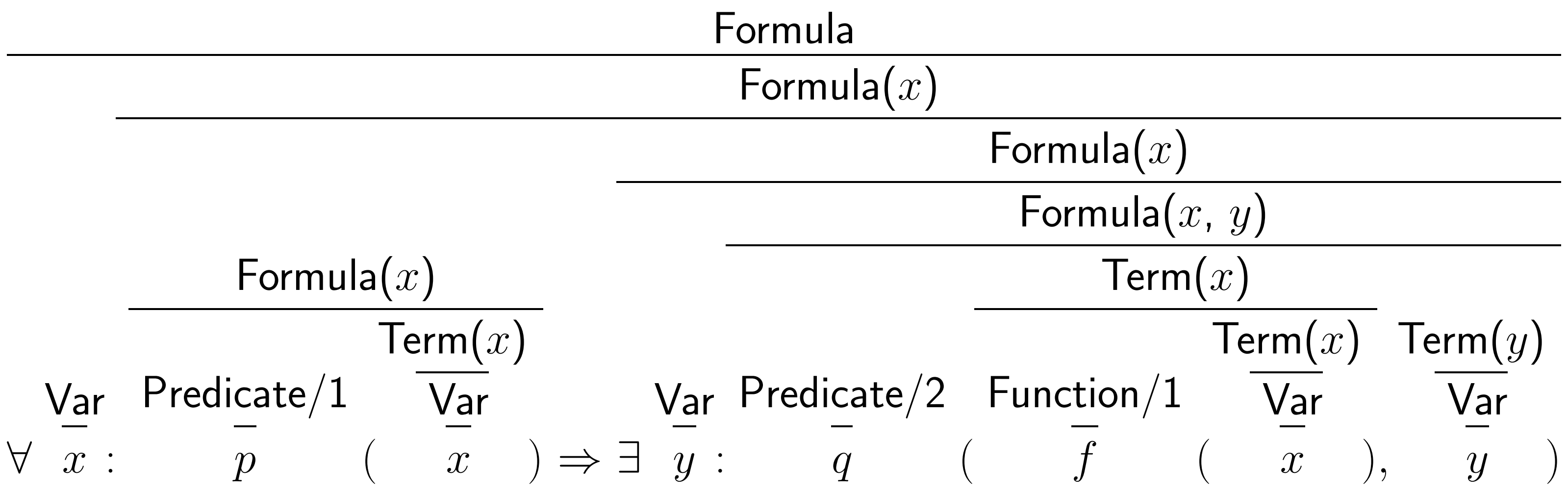
# Operational Interpretation

```
public final class Exists implements Formula
{
  private String variable; private Term domain; private Formula formula;

  public boolean eval() throws EvalException
  {
    Iterator iterator = Model.iterator(domain);
    while (iterator.hasNext()) {
      Context.begin(variable, iterator.next());
      boolean result = formula.eval();
      Context.end();
      if (result) return true; }
    return false;
  }
}
```

# Syntax Analysis

$$\forall x : p(x) \Rightarrow \exists y : q(f(x), y)$$

$$\forall x : (p(x) \Rightarrow \exists y : q(f(x), y)).$$



Formula

Formula($x$)

Formula($x$)

Formula($x$, $y$)

Formula($x$) Term($x$)

Term($x$) Term($x$) Term($y$)

Var Predicate/1 Var Var Predicate/2 Function/1 Var Var

$\forall$ $x$ : $p$ ( $x$ ) $\Rightarrow$ $\exists$ $y$ : $q$ ( $f$ ( $x$ ), $y$ )

# Examples

- $(\forall x : y \leq x)$
  - $x$ is bound, $y$ is free; the formula is therefore not closed.
  - true in assignment $[y \mapsto 0]$ over the domain of natural numbers with the usual interpretation of '$\leq$', because $0 \leq x$ is true for every natural number $x$:

$$0 \leq 0, 0 \leq 1, 0 \leq 2, \ldots$$

- $(\exists x : x|15)$
  - $x$ is bound; the formula is closed.
  - true for every assignment over the natural numbers with '$|$' interpreted as 'divides', because $x|15$ is true for some natural number $x$ (e.g. for 3):

$$3|15.$$

## Bound Variables

The meaning of

$$\forall x : \exists y : x * y \leq z$$

is the same as the meaning of

$$\forall y : \exists w : y * w \leq z$$

but it is not the same as the meaning of

$$\forall x : \exists y : x * y \leq w.$$

Renaming a bound variable does not change the meaning of a phrase.

## Closed Formulas

Do not just write

$$x + y = y + x$$

if you wish to say, e.g.

$$x + y = y + x, \text{ for every } x \text{ and } y,$$

i.e.,

$$\forall x, y : x + y = y + x.$$

Bind variables by quantifiers to avoid misinterpretations.

# Abbreviations

- $\forall x : \forall y : A$

$$\forall x, y : A$$

- $\exists x : \exists y : A$

$$\exists x, y : A$$

Multiple quantifiers of the same type can be merged.

## Quantifier Patterns

- $\forall x : A_x \Rightarrow B$

$$\forall A_x : B$$

- $\exists x : A_x \wedge B$

$$\exists A_x : B.$$

Example:

- $\forall x : \exists 0 \leq y \leq x : |x - 2 * y| \leq 1$
- $\forall x : (\exists y : (0 \leq y) \wedge (y \leq x) \wedge (|x - 2 * y| \leq 1))$

Bound variable has to be deduced from the context.

# Examples

- “All Ferraris are red”
  - “It holds for all objects that, if the object is a Ferrari, then the object is red”.
  - $\forall x : \text{isFerrari}(x) \Rightarrow \text{isRed}(x)$.
- “Some Ferraris are black”
  - “There exists an object such that the object is a Ferrari and the object is black”
  - $\exists x : \text{isFerrari}(x) \wedge \text{isBlack}(x)$.
- “Everybody has exactly one father”.
  - “For every person $x$, there is exactly one person $y$ such that $y$ is the father of $x$ and every person $z$ who is father of $x$ is identical to $y$”.
  - $(\forall x : (\exists y : \text{isfather}(y, x) \wedge (\forall z : \text{isfather}(z, x) \Rightarrow z = y)))$.
  - $\forall x : \exists y : \text{isfather}(y, x) \wedge \forall z : \text{isfather}(z, x) \Rightarrow z = y$.

## De Morgan's Laws

For every variable $x$ and formula $A$, we have

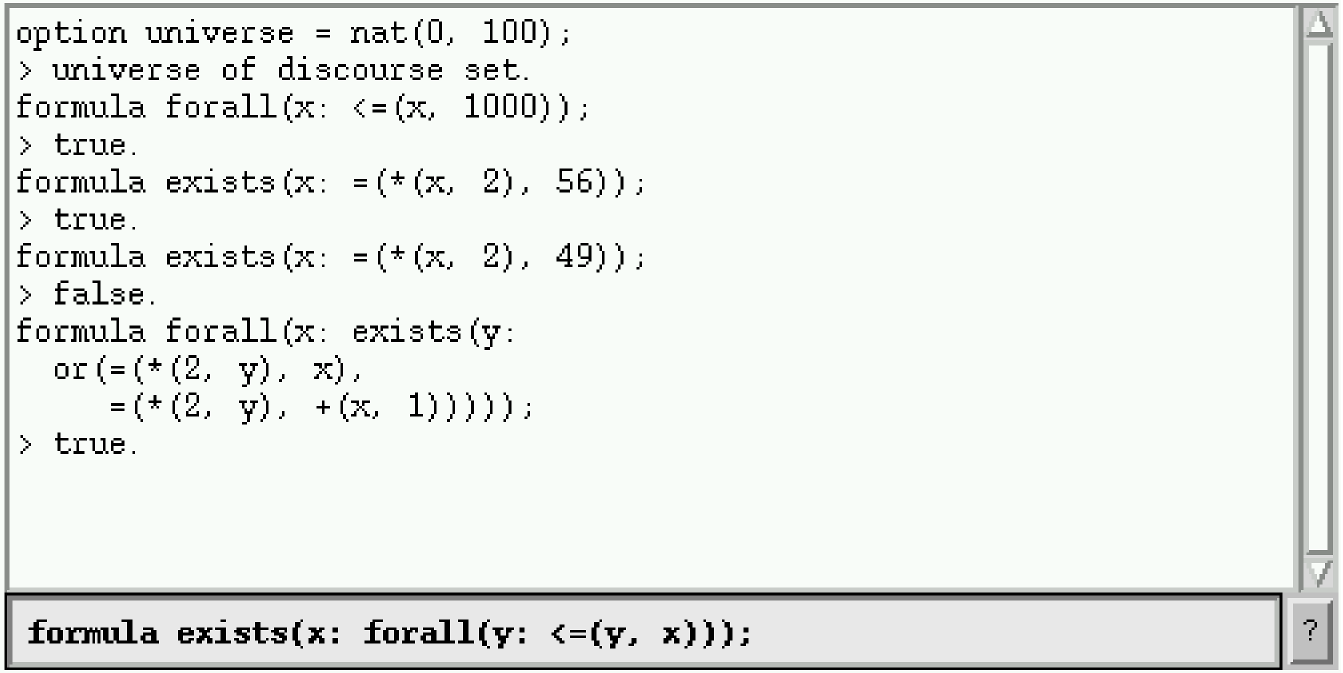
- $$\neg\forall x : A \text{ iff } \exists x : \neg A$$

  "Not every $x$ satisfies $A$" equals "there exists some $x$ with $\neg A$".
- $$\neg\exists x : A \text{ iff } \forall x : \neg A$$

  "There exists some $x$ with $A$" equals "not every $x$ satisfies $\neg A$".

Universal and existential quantification are dual concepts.

# Logic Evaluator



```
option universe = nat(0, 100);
> universe of discourse set.
formula forall(x: <=(x, 1000));
> true.
formula exists(x: =(*(x, 2), 56));
> true.
formula exists(x: =(*(x, 2), 49));
> false.
formula forall(x: exists(y:
  or(=(*(2, y), x),
     =(*(2, y), +(x, 1)))));
> true.
```

```
formula exists(x: forall(y: <=(y, x)));
```

## Local Definitions

$$(\textbf{let } x = T : P) \ (P \textbf{ where } x = T)$$

- Term $T$, any phrase (term or formula) $P$.
- Variable $x$ bound by "block quantifier" **let**/**where**.
- Alternative forms:
  - Let $x$ be $T$. Then we have $A$.
  - Let $x$ be $T$ in $P$; $A$, where $x = T$.
  - `let(`$x$ `=` $T$`:` $A$`)`
- Semantics: identical to the semantics of $P$ with $x$ replaced by $T$.

  Semantics of $P$ in an extension of the given assignment with $x$ mapped to value of $T$.

# Operational Interpretation

```
public final class LetTerm implements Term
{
  private String variable; private Term term; private Term body;

  public Value eval() throws EvalException
  {
    Context.begin(variable, term.eval());
    Value result = body.eval();
    Context.end();
    return result;
  }
}
```

# Example

Take the domain of natural numbers and assignment $[x \mapsto 1]$.

- The proposition

  $$\textbf{let } y = 0 : x \leq y$$

  is false because $1 \leq 0$ does not hold.
- The function $f$ defined as

  $$f(x) := s * x + s \textbf{ where } s = x + 1$$

  has the same meaning as if it were defined as

  $$f(x) := (x + 1) * x + (x + 1).$$

# Example

- The formula

  $$x|y \land \exists z : y|z \textbf{ where } y = 2x$$

  is equivalent to $x|2x \land \exists z : 2x|z$.
- The formula

  $$x|y \land \exists y : y|x \textbf{ where } y = 2x$$

  is equivalent to $x|2x \land \exists y : y|x$ but not to $x|2x \land \exists y : 2x|x$.

## Abbreviation

Instead of writing

$$\textbf{let } x = T_0 : \textbf{let } y = T_1 : A$$

we usually write

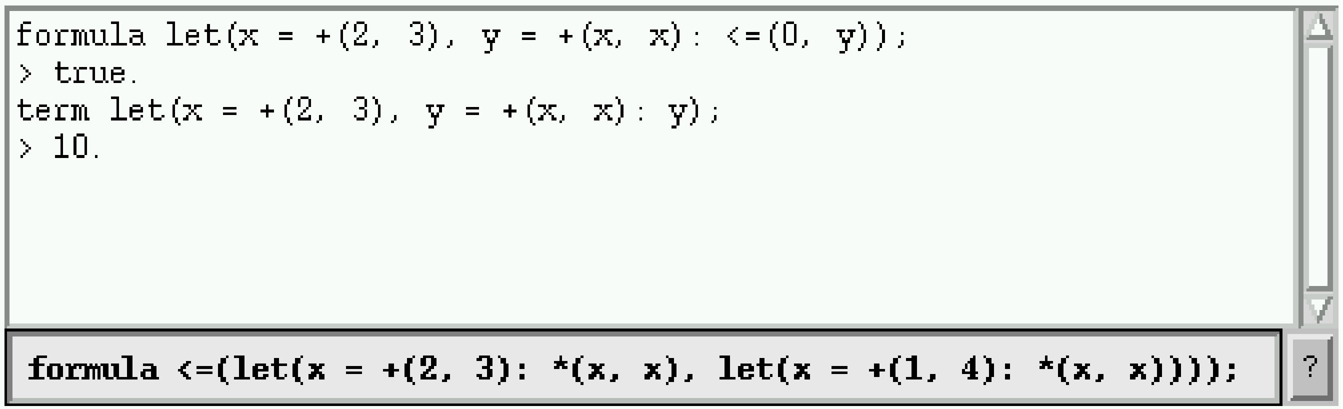
$$\textbf{let } x = T_0, y = T_1 : A$$

or correspondingly

$$A \textbf{ where } x = T_0, y = T_1.$$

Multiple bindings can be merged.

# Logic Evaluator



```
formula let(x = +(2, 3), y = +(x, x): <=(0, y));
> true.
term let(x = +(2, 3), y = +(x, x): y);
> 10.
```

```
formula <=(let(x = +(2, 3): *(x, x), let(x = +(1, 4): *(x, x)));
```

# Example

# Example

## Problem Statement:

Write a program that takes a number and returns the next prime number.

## Program Specification

- Input: $n$ such that $isNumber(n)$;
- Output: $p$ such that $isNextPrime(n, p)$.

# Program Specification

- Input condition: $isNumber$

  Unary predicate that describes well-formed input $n$.

- Output condition: $isNextPrime$

  Binary predicate that describes how the output $p$ is related to the input $n$.

Specification of an explicit construction problem.

## Input and Output Conditions

Definition: $n$ is a number.

$$isNumber(n) :\Leftrightarrow n \in \mathbb{N}$$

Definition: $p$ is the next prime number after $n$.

$$\begin{aligned}
&isNextPrime(n, p) :\Leftrightarrow \\
&\quad isNumber(p) \land \\
&\quad isPrime(p) \land \\
&\quad isNextP(n, p).
\end{aligned}$$

# Output Condition

Definition: $p$ is a prime number.

$$\begin{aligned}&isPrime(p) :\Leftrightarrow\\&\quad 1 < p \land\\&\quad \neg(\exists 1 < n < p : n | p).\end{aligned}$$

A bit more elegant:

$$\begin{aligned}&isPrime(p) :\Leftrightarrow\\&\quad 1 < p \land\\&\quad \forall 1 < n < p : n \not| \, p.\end{aligned}$$

# Output Condition

Definition: $p$ is the next such number after $n$.

Option 1: if $n$ is prime, then $isNextP(n,n)$.

$$\begin{aligned}&isNextP(n,p) :\Leftrightarrow\\ &\quad n \leq p \land\\ &\quad \neg(\exists n \leq q < p : isPrime(q)).\end{aligned}$$

Option 2: even if $n$ is prime, $\neg isNextP(n,n)$.

$$\begin{aligned}&isNextP(n,p) :\Leftrightarrow\\ &\quad n < p \land\\ &\quad \neg(\exists n < q < p : isPrime(q)).\end{aligned}$$

# Complete Specification

- **Input:** $n \in \mathbb{N}$;
- **Output:** $p \in \mathbb{N}$ such that $isNextPrime(n, p)$.

$$\begin{aligned}
&isNextPrime(n, p) :\Leftrightarrow \\
&\quad n \leq p \wedge isPrime(p) \wedge \\
&\quad \neg(\exists n \leq q < p : isPrime(q))
\end{aligned}$$

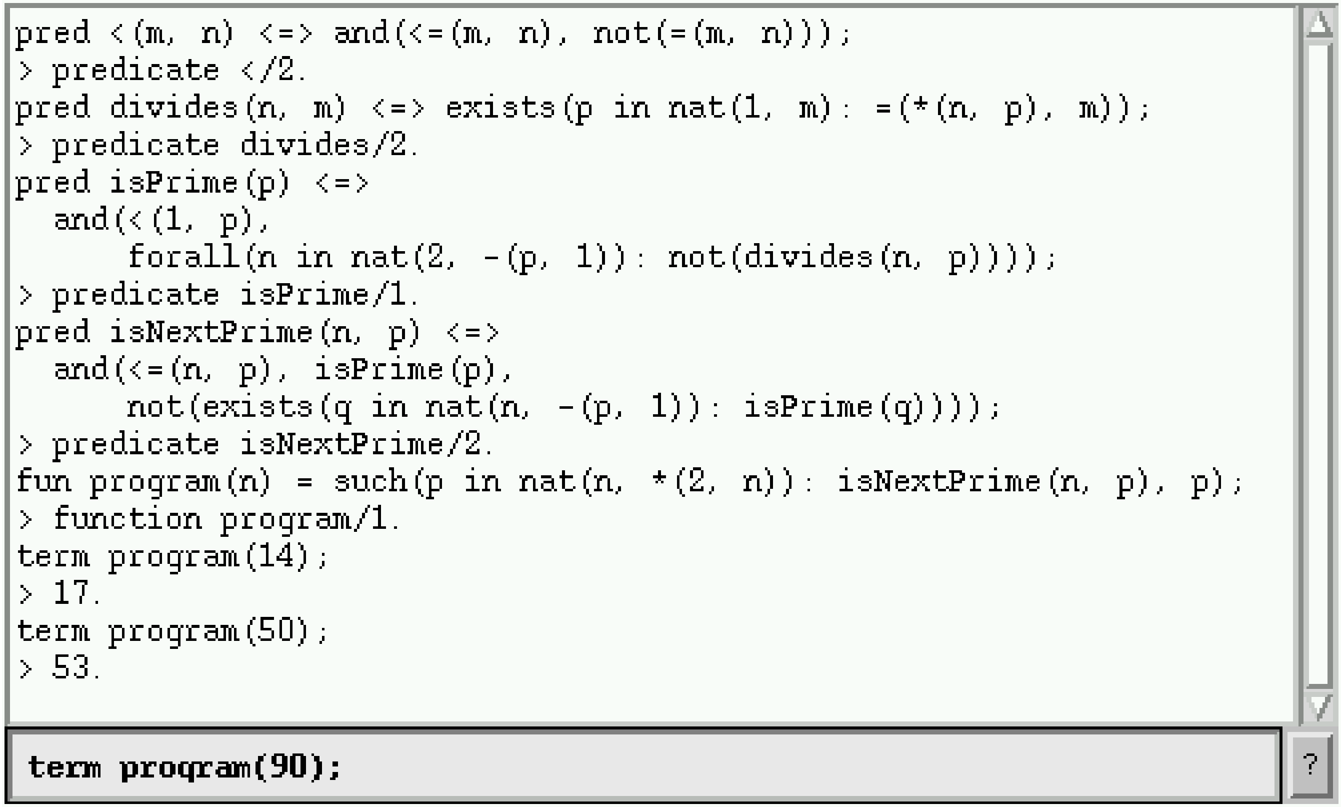
$$\begin{aligned}
&isPrime(p) :\Leftrightarrow \\
&\quad 1 < p \wedge \forall 1 < n < p : n \not| \; p
\end{aligned}$$

$$n|m :\Leftrightarrow \exists p : n * p = m.$$

# Logic Evaluator

```
pred <(m, n) <=> and(<=(m, n), not(=(m, n)));

pred divides(n, m) <=> exists(p in nat(1, m): =(*(n, p), m));

pred isPrime(p) <=>
  and(<(1, p),
      forall(n in nat(2, -(p, 1)): not(divides(n, p))));

pred isNextPrime(n, p) <=>
  and(<=(n, p), isPrime(p),
      not(exists(q in nat(n, -(p, 1)): isPrime(q))));

fun program(n) =
  such(p in nat(n, *(2, n)): isNextPrime(n, p), p);
```

# Logic Evaluator



```
pred <(m, n) <=> and(<=(m, n), not(=(m, n)));
> predicate </2.
pred divides(n, m) <=> exists(p in nat(1, m): =(*(n, p), m));
> predicate divides/2.
pred isPrime(p) <=>
  and(<(1, p),
      forall(n in nat(2, -(p, 1)): not(divides(n, p))));
> predicate isPrime/1.
pred isNextPrime(n, p) <=>
  and(<=(n, p), isPrime(p),
      not(exists(q in nat(n, -(p, 1)): isPrime(q))));
> predicate isNextPrime/2.
fun program(n) = such(p in nat(n, *(2, n)): isNextPrime(n, p), p);
> function program/1.
term program(14);
> 17.
term program(50);
> 53.
```

```
term program(90);
```

## Summary

- Logical connectives.
  - Truth tables.
- Domains, constants, interpretations.
- Terms
  - Variables and assignments.
  - Function applications.
- Atomic formulas.
- Quantified formulas.
  - Free and bound variables; closed formulas.
  - Universal and existential quantification.
  - Local definitions.