M

Predicate logic

Syntax and semantics

Patrick D. Elliott

May 30, 2023

Recursion

Factorial

$$
\begin{aligned}
4! &= 4 * 3 * 2 * 1 \\
&= 12 * 2 * 1 \\
&= 24 * 1 \\
&= 24
\end{aligned}
$$

- How do we define a function $n!$ in haskell?

Evaluation without a base case

```
brokenFact1 :: Integer -> Integer
brokenFact1 n = n * brokenFact1 (n - 1)
```

Eval without a base cont.

```
brokenFact1 4 =
  4 * brokenFact1 3
  4 * (3 * brokenFact1 2)
  4 * (3 * (2 * brokenFact1 1))
  4 * (3 * (2 * (1 * brokenFact1 0)))
  4 * (3 * (2 * (1 * (0 * (brokenFact1 -1)))))
  -- this never reaches normal form
```

Warning: if you run this code yourself, you won't get an error, but you *will* have to manually terminate the running haskell process.

Adding a base

The base case provides a point at which evaluation halts.

```
factorial :: Integer -> Integer
factorial 0 = 1
factorial n = n * factorial (n - 1)
```

Eval with base

```
factorial 4 =
  4 * factorial 3
  4 * (3 * factorial 2)
  4 * (3 * (2 * factorial 1))
  4 * (3 * (2 * (1 * factorial 0))) -- end of recursion
  4 * (3 * (2 * (1 * 1)))
  4 * (3 * (2 * (1 * 1)))
  24
```

Defining the base

- Question: why does the function return `1` for the base case?
  - What would happen if we used `0` as the return value for the base?

First-order logic

First-order logic goes significantly beyond the expressive power of propositional logic.

It allows us, e.g., to model syllogistic reasoning like the following:

- Patrick admires every philosopher.
- Stalnaker is a philosopher.
- $\Rightarrow$ *Patrick admires Stalnaker.*

A grammar of first-order logic consists of:

- A stock of variables Var.
- A stock of *predicate symbols*, each assigned an arity n.

An *atomic formula* of first order logic consists of a predicate symbol of arity n followed by a sequence of n variables.

If P is a predicate symbol of arity n, and $x_1 \dots x_n \in Var$, then $P(x_1, \dots, x_n)$ is a wff of predicate logic.

First-order logic includes all of the same logical operators as propositional logic:

- If ϕ, ψ are sentences of first-order logic, then $(\phi \wedge \psi)$ is a sentence of first-order logic.
- If ϕ, ψ are sentences of first-order logic, then $(\phi \vee \psi)$ is a sentence of first-order logic.
- If ϕ, ψ are sentences of first-order logic, then $(\phi \rightarrow \psi)$ is a sentence of first-order logic.
- If ϕ is a sentence of first-order logic, then so is $\neg\phi$

Much of first-order logic's interesting properties stems from its addition of *quantifiers*:

- If ϕ is a sentence of first-order logic, and $x \in Var$, then $\exists x(\phi)$ is a sentence of first-order logic.
- If ϕ is a sentence of first-order logic, and $x \in Var$, then $\forall x(\phi)$ is a sentence of first-order logic.

The existential/universal quantifier is also standardly defined as the dual of the other, i.e.:

$$\exists x(\phi) := \neg\forall\neg(\phi)$$

Implementation in haskell

```haskell
type Name = String
newtype Var = Var Name deriving (Eq,Show)
```

Atomic formulas

```
data Formula = Atomic String [Var]
  | Neg Formula
  | Formula `Impl` Formula
  | Formula `Conj` Formula
  | Formula `Disj` Formula
  | Forall Var Formula
  | Exists Var Formula
  deriving Eq
```

- A semantics for predicate logic is stated relative to a *model* and an *assignment*.
    - A model consists of a *domain of individuals* D, and an interpretation function I mapping predicate symbols to boolean-valued functions.
    - I maps a predicate symbol of arity 0 to a boolean value.
    - I maps a predicate symbol of arity 1 to a function $f : D \rightarrow \{\mathbf{True}, \mathbf{False}\}$.
    - I maps a predicate symbol of arity 2 to a function $f : D \times D \rightarrow \{\mathbf{True}, \mathbf{False}\}$
    - ...and so on.
- You're probably more familiar with a presentation where I maps predicate symbols to *sets* rather than functions, but this is equivalent.

An *assignment function* g is a total function from the set of variables Var the domain of individuals D.

$$g_1 := \begin{bmatrix} x \rightarrow & \textbf{Bart} \\ y \rightarrow & \textbf{Milhouse} \\ z \rightarrow & \textbf{Bart} \\ \ldots & \end{bmatrix}$$

We can now define $[\![.]\!]^{M,g}$ for atomic sentences, where $[\![.]\!]^{M,g}$ is a total function from wffs of predicate logic to boolean values.

- $[\![P(x_1, \ldots, x_n)]\!]^{M,g} = I(P)(g(x_1), \ldots, g(x_n)))$

The semantics of the logical connectives is the same as in propositional logic.

- $[\![\phi \wedge \psi]\!]^{M,g} = \textbf{True}$ iff $[\![\phi]\!]^{M,g} = \textbf{True}$ and $[\![\psi]\!]^{M,g} = \textbf{True}$
- $[\![\phi \vee \psi]\!]^{M,g} = \textbf{True}$ iff $[\![\phi]\!]^{M,g} = \textbf{True}$ or $[\![\psi]\!]^{M,g} = \textbf{True}$
- $[\![\phi \rightarrow \psi]\!]^{M,g} = \textbf{True}$ iff $[\![\phi]\!]^{M,g} = \textbf{False}$ or $[\![\psi]\!]^{M,g} = \textbf{True}$
- $[\![\neg\phi]\!]^{M,g} = \textbf{True}$ iff $[\![\phi]\!]^{M,g} = \textbf{False}$

- The semantics for quantifiers is a little more involved.:
    - $[\![\exists x \phi]\!]^{M,g} =$
      True iff there is some assignment g' s.t. $g[x]g'$ and $[\![\phi]\!]^{M,g'} = \textbf{True}$
    - $[\![\forall x \phi]\!]^{M,g} =$
      True iff there is no assignment g' s.t. $g[x]g'$ and $[\![\phi]\!]^{M,g'} = \textbf{False}$

$\mathcal{F}in$

References