Logik für Informatiker
Logic for computer scientists

Induction

Till Mossakowski

WiSe 2009/10

Induction is like a chain of dominoes. You need

- the dominoes must be close enough together $\Rightarrow$ one falling dominoe knocks down the next (*inductive step*)
- you need to knock down the first dominoe (*inductive basis*)

Note: in the inductive step, branching is possible.

Induction is like a chain of dominoes. You need

- the dominoes must be close enough together $\Rightarrow$ one falling dominoe knocks down the next (*inductive step*)
- you need to knock down the first dominoe (*inductive basis*)

Note: in the inductive step, branching is possible.

Inductive definition: Natural numbers

1. 0 is a natural number.
2. If n is natural number, then $suc(n)$ is a natural number.
3. There is no natural number whose successor is 0.
4. Two different natural numbers have different successors.
5. Nothing is a natural number unless generated by repeated applications of (1) and (2).

Recursive definition of functions

$$\forall y(0 + y = y)$$
$$\forall x \forall y(suc(x) + y = suc(x + y))$$

$$\forall y(0 * y = 0)$$
$$\forall x \forall y(suc(x) * y = (x * y) + y)$$

Formalization of Peano's axioms

1. a constant 0
2. a unary function symbol *suc*
3. $\forall n \ \neg suc(n) = 0$
4. $\forall m \forall n \ suc(m) = suc(n) \rightarrow m = n$
5. $(\Phi(x/0) \wedge \forall n(\Phi(x/n) \rightarrow \Phi(x/suc(n)))) \rightarrow \forall n \ \Phi(x/n)$
   if Φ is a formula with a free variable x, and
   $\Phi(x/t)$ denotes the replacement of x with t within Φ

Inductive proofs

Take $\Phi(x) := \forall y \forall z (x + (y + z) = (x + y) + z)$. Then

$$(\Phi(x/0) \wedge \forall n(\Phi(x/n) \rightarrow \Phi(x/suc(n)))) \rightarrow \forall n\ \Phi(x/n)$$

is just

$$\begin{array}{l}(\forall y \forall z(0 + (y + z) = (0 + y) + z) \\ \quad \wedge \forall n \forall y \forall z\ (n + (y + z) = (n + y) + z \\ \qquad \rightarrow suc(n) + (y + z) = (suc(n) + y) + z)) \\ \quad \rightarrow \forall n \forall y \forall z\ (n + (y + z) = (n + y) + z)\end{array}$$

With this, we can prove $\forall n \forall y \forall z\ (n + (y + z) = (n + y) + z)$

1. The empty list [] is a list.
2. If l is a list and n is natural number, then $cons(n, l)$ is a list.
3. Nothing is a list unless generated by repeated applications of (1) and (2).

Note: This needs *many-sorted* first-order logic.
We have two sorts of objects: natural numbers and lists.

Recursive definition of functions over lists

$$
\begin{array}{l}
length([\,]) \;=\; 0 \\
\forall n : Nat\ \forall l : List\ (length(cons(n, l)) \;=\; suc(length(l)))
\end{array}
$$

$$
\begin{array}{l}
\forall l : List\ ([\,] \;++\; l \;=\; l) \\
\forall n : Nat\ \forall l_1 : List\ \forall l_2 : List \\
\qquad\qquad (cons(n, l_1) \;++\; l_2 \;=\; cons(n, l_1 \;++\; l_2))
\end{array}
$$

Inductive proofs over lists

$$\forall l_1 : List\ \forall l_2 : List\ \forall l_3 : List$$
$$(\ l_1\ ++\ (l_2\ ++\ l_3)\ =\ (l_1\ ++\ l_2)\ ++\ l_3\ )$$

$$\forall l_1 : List\ \forall l_2 : List$$
$$(\ length(l_1\ ++\ l_2)\ =\ length(l_1)\ +\ length(l_2)\ )$$

A recursive program computing $0 + 1 + 2 + \ldots + n$

```
public natural sumToRec(natural n) {
  if(n == 0) return 0;
  else return n + sumToRec(n - 1);
}
```

An imperative program

```
public natural sumUpTo(natural n) {
  natural sum = 0;
  natural count = 0;
  while(count < n) {
    count += 1;
    sum += count;
  }
  return sum;
}
```

Invariant: $sum = 0 + 1 + 2 + \ldots + count$

An imperative program

```
public natural sumUpTo(natural n) {
  natural sum = 0;
  natural count = 0;
  while(count < n) {
    count += 1;
    sum += count;
  }
  return sum;
}
```

Invariant: $sum = 0 + 1 + 2 + \ldots + count$

A second imperative implementation

```
public natural sumDownFrom(natural n) {
  natural sum = 0;
  natural count = n;
  while(count > 0) {
   sum += count;
   count -= 1;
  }
  return sum;
}
```

Invariant: $sum = (count + 1) + \ldots + n$

A second imperative implementation

```
public natural sumDownFrom(natural n) {
  natural sum = 0;
  natural count = n;
  while(count > 0) {
   sum += count;
   count -= 1;
  }
  return sum;
}
```

Invariant: $sum = (count + 1) + \ldots + n$

The Java Modeling Language (JML)

http://www.cs.ucf.edu/~leavens/JML

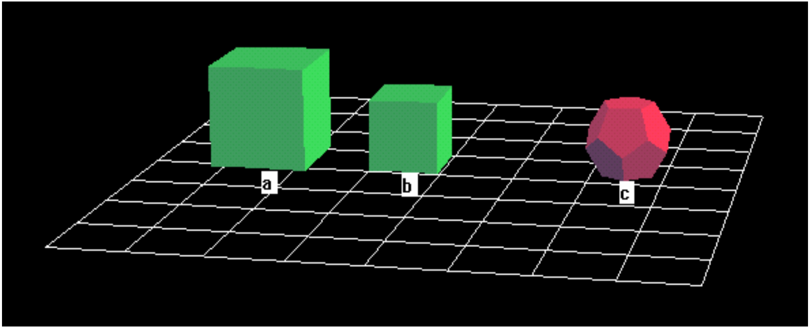
Exercises

- Language, proof and logic, 16.27 - 16.31
- write a small Java program annotated with JML, such that universal and existential quantifiers are used

First-order structures: motivation

- How to make the notion of *logical consequence* more formal?
- For propositional logic: truth tables $\Rightarrow$ tautological consequence
- For first-order logic, we need also to interpret quantifiers and identity
- The notion of *first-order structure* models Tarski's world situations and real-world situations using *set theory*

Example

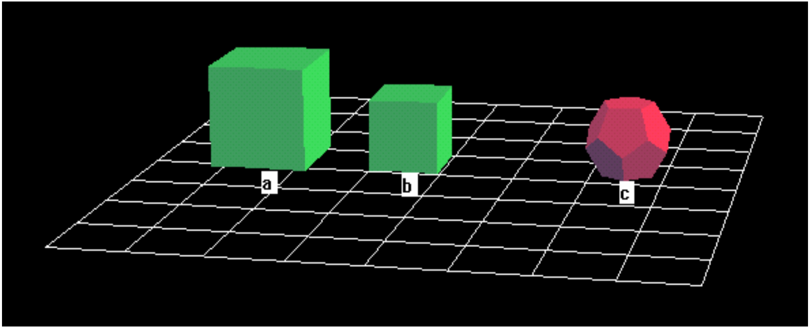


First-order structures: definition

A first-order structure $\mathfrak{M}$ consists of:

- a nonempty set $D^{\mathfrak{M}}$, the *domain of discourse*;
- for each n-ary predicate P of the language,
  a set $\mathfrak{M}(P)$ of n-tuples $\langle x_1, \ldots, x_n \rangle$ of elements of $D^{\mathfrak{M}}$,
  called the *extension* of P.
  The extension of the identity symbol $=$ must be
  $\{\langle x, x \rangle \mid x \in D^{\mathfrak{M}}\}$;
- for any name (individual constant) c of the language, an element $\mathfrak{M}(c)$ of $D^{\mathfrak{M}}$.

Example



Assume the language consists of the predicates *Cube*, *Dodec* and *Larger* and the names *a*, *b* and *c*.

- $D^{\mathfrak{M}} = \{Cube1, Cube2, Dodec1\}$;
- $\mathfrak{M}(Cube) = \{Cube1, Cube2\}$;
- $\mathfrak{M}(Dodec) = \{Dodec1\}$;
- $\mathfrak{M}(Larger) = \{(Cube1, Cube2), (Cube1, Dodec1)\}$;
- $\mathfrak{M}(=) = \{(Cube1, Cube1), (Cube2, Cube2), (Dodec1, Dodec1)\}$;
- $\mathfrak{M}(a) = Cube1; \quad \mathfrak{M}(b) = Cube2; \quad \mathfrak{M}(c) = Dodec1.$

An interpretation not conformant with Tarski's World

- $D^{\mathfrak{M}} = \{Cube1, Cube2, Dodec1\}$;
- $\mathfrak{M}(Cube) = \{Dodec1, Cube2\}$;
- $\mathfrak{M}(Dodec) = \emptyset$;
- $\mathfrak{M}(Larger) = \{(Cube1, Cube1), (Dodec1, Cube2)\}$;
- $\mathfrak{M}(=) = \{(Cube1, Cube1), (Cube2, Cube2), (Dodec1, Dodec1)\}$;
- $\mathfrak{M}(a) = Cube1$; $\mathfrak{M}(b) = Cube2$; $\mathfrak{M}(c) = Dodec1$.

Variable assignments

A *variable assignment* in $\mathfrak{M}$ is a (possibly partial) function g defined on a set of variables and taking values in $D^{\mathfrak{M}}$.
Given a well-formed formula P, we say that the variable assignment g is *appropriate* for P if all the free variables of P are in the domain of g, that is, if g assigns objects to each free variable of P.

Examples

$D^{\mathfrak{M}} = \{Cube1, Cube2, Dodec1\}$
g_1 assignes $Cube1$, $Cube2$, $Dodec1$ to the variables x, y, z, respectively.
g_1 is appropriate for $Between(x, y, z) \wedge \exists u(Large(u))$, but not for $Larger(x, v)$.

g_2 is the empty assignment.
g_2 is only appropriate for well-formed formulas without free variables (that is, for sentences).

Variants of variable assignments

If g is a variable assignment, $g[v/b]$ is the variable assignment

- whose domain is that of g plus the variable v, and
- which assigns the same values as g, except that
- it assigns b to the variable v.

$[t]_g^{\mathfrak{M}}$ is

- $\mathfrak{M}(t)$ if t is an individual constant, and
- $g(t)$ if t is a variable.

Satisfaction (A. Tarski)

1. $\mathfrak{M} \models R(t_1, \ldots, t_n)[g]$ iff $\langle [t_1]_g^{\mathfrak{M}}, \ldots, [t_n]_g^{\mathfrak{M}} \rangle \in \mathfrak{M}(R)$;
2. $\mathfrak{M} \models \neg P[g]$ iff it is not the case that $\mathfrak{M} \models P[g]$;
3. $\mathfrak{M} \models P \land Q[g]$ iff both $\mathfrak{M} \models P[g]$ and $\mathfrak{M} \models Q[g]$;
4. $\mathfrak{M} \models P \lor Q[g]$ iff $\mathfrak{M} \models P[g]$ or $\mathfrak{M} \models Q[g]$ or both;
5. $\mathfrak{M} \models P \to Q[g]$ iff not $\mathfrak{M} \models P[g]$ or $\mathfrak{M} \models Q[g]$ or both;
6. $\mathfrak{M} \models P \leftrightarrow Q[g]$ iff ($\mathfrak{M} \models P[g]$ iff $\mathfrak{M} \models Q[g]$);
7. $\mathfrak{M} \models \forall x\ P[g]$ iff for every $d \in D^{\mathfrak{M}}$, $\mathfrak{M} \models P[g[x/d]]$;
8. $\mathfrak{M} \models \exists x\ P[g]$ iff for some $d \in D^{\mathfrak{M}}$, $\mathfrak{M} \models P[g[x/d]]$.