Formal Methods and Specification (SS 2021) Lecture 3: Logical Theories and Modeling of Data Structures

Stefan Ratschan

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

Evropský sociální fond Praha & EU: Investujeme do vaší budoucnosti

Reduction to Logic

Specification:

- Input: $x \in \mathbb{Z}$
- Output: $\hat{x} \in \mathbb{Z}$, $\hat{x} < x$, ...

Program:

```
if $f(x) \geq 0$ then
    return $x - f(x) - 1$
else
    return $x + f(x) - 1$
```

Program is correct iff *verification condition*

$$\forall x \in \mathbb{Z} \,.\; \begin{array}{l} f(x) \geq 0 \Rightarrow x - f(x) - 1 < x \wedge \\ \neg f(x) \geq 0 \Rightarrow x + f(x) - 1 < x \end{array}$$

holds.

Specifications with (Non-Numerical) Data Structures

Example:

		example:
Input:	pair p s.t. $\mathrm{snd}(p) = 1$	$\mathrm{pair}(7, 1)$
Output:	pair p' s.t. $\mathrm{snd}(p') = 2\mathrm{fst}(p)$	$\mathrm{pair}(888, 14)$

Well-formed? At least one output for each input?

Formally:

$$\forall p\ .\ \mathrm{snd}(p) = 1 \Rightarrow \exists p'\ .\ \mathrm{snd}(p') = 2\mathrm{fst}(p)$$

Correctness of Implementation

Input: pair p s.t. $\mathrm{snd}(p) = 1$
Output: pair p' s.t. $\mathrm{snd}(p') = 2\mathrm{fst}(p)$

Example implementation:

return $\mathrm{pair}(\mathrm{snd}(p), 2\mathrm{fst}(p))$

Does algorithm fulfill specification?

Corresponding formula:

$$\forall p\, \forall p' \,.\, [\mathrm{snd}(p) = 1 \wedge p' = \mathrm{pair}(\mathrm{snd}(p), 2\mathrm{fst}(p))] \Rightarrow \mathrm{snd}(p') = 2\mathrm{fst}(p)$$

For comparison:

$$\forall p \,.\, \mathrm{snd}(p) = 1 \Rightarrow \exists p' \,.\, \mathrm{snd}(p') = 2\mathrm{fst}(p)$$

We will check it later . . .

Problem: $\forall p\, \forall p'$

Correctness of Implementation

We assume an I/O-specification:

- $I(x)$: x is an allowed input
- $O(x, x')$: x' is an allowed output for input x

(x, x' may stand for several variables)

algorithm for inputs x:

return $f(x)$

where f is a term like x^2, $a[2i]$, $\mathrm{rest}(x)$

Correctness:

$$\forall x \forall x' \,.\, [I(x) \wedge x' = f(x)] \Rightarrow O(x, x')$$

Correctness Proof

We want to prove

$$\forall p \forall p' . [\mathrm{snd}(p) = 1 \wedge p' = \mathrm{pair}(\mathrm{snd}(p), 2\mathrm{fst}(p))] \Rightarrow \mathrm{snd}(p') = 2\mathrm{fst}(p)$$

Let p, p' be arbitrary, but fixed pairs. We assume $\mathrm{snd}(p) = 1$ and $p' = \mathrm{pair}(\mathrm{snd}(p), 2\mathrm{fst}(p))$, and we prove $\mathrm{snd}(p') = 2\mathrm{fst}(p)$. Due to the assumption $p' = \mathrm{pair}(\mathrm{snd}(p), 2\mathrm{fst}(p))$, it suffices to prove $\mathrm{snd}(\mathrm{pair}(\mathrm{snd}(p), 2\mathrm{fst}(p))) = 2\mathrm{fst}(p)$.

And now? Is this not obvious? obvious to whom?

Which knowledge about pairs is obvious??

$$\forall x, y . \mathrm{fst}(\mathrm{pair}(x, y)) = x$$

$$\forall x, y . \mathrm{snd}(\mathrm{pair}(x, y)) = y$$

$$\forall p . \mathrm{pair}(\mathrm{fst}(p), \mathrm{snd}(p)) = p$$

We can add such axioms beforehand as known facts.

Correctness Proofs Based on Axioms

We want to prove that

$$\forall p \forall p' \,.\, [\mathrm{snd}(p) = 1 \land p' = \mathrm{pair}(\mathrm{snd}(p), 2\mathrm{fst}(p))] \Rightarrow \mathrm{snd}(p') = 2\mathrm{fst}(p)$$

holds in the *theory of pairs* (also short: we want to prove ... in the theory of pairs)

- intuitively: formula follows from the axioms
- formally (see BI-MLO): theory: set of axioms,
  the formula is a logical consequence of the theory.

Proof continued:

... it suffices to prove $\mathrm{snd}(\mathrm{pair}(\mathrm{snd}(p), 2\mathrm{fst}(p)))) = 2\mathrm{fst}(p)$.

From the axiom

$$\forall x, y \,.\, \mathrm{snd}(\mathrm{pair}(x, y)) = y,$$

after choosing $x \leftarrow \mathrm{snd}(p)$, $y \leftarrow 2\mathrm{fst}(p)$ we know that

$$\mathrm{snd}(\mathrm{pair}(\mathrm{snd}(p), 2\mathrm{fst}(p))) = 2\mathrm{fst}(p),$$

which is what we wanted to prove.

Pairs: Summary

Type: $\mathcal{P}$, elements: Type $\mathcal{T}$, we will also write $\mathcal{P}[\mathcal{T}]$

Function symbols

- pair: $\mathcal{T} \times \mathcal{T} \to \mathcal{P}[\mathcal{T}]$
- fst: $\mathcal{P}[\mathcal{T}] \to \mathcal{T}$
- snd: $\mathcal{P}[\mathcal{T}] \to \mathcal{T}$

Axioms ($x, y \in \mathcal{T}$, $p \in \mathcal{P}[\mathcal{T}]$):

- $\forall x, y \,.\, \mathrm{fst}(\mathrm{pair}(x, y)) = x$
- $\forall x, y \,.\, \mathrm{snd}(\mathrm{pair}(x, y)) = y$
- $\forall p \,.\, \mathrm{pair}(\mathrm{fst}(p), \mathrm{snd}(p)) = p$

types + function symbols + predicate symbols: *signature*

Modeling of Data Structures

First we agree on a description of the behavior of the data structure using certain axioms.

Such a description is called *axiomatization*.

In mathematics: group, ring, field

Then we can prove theorems in the corresponding theory, i.e. we assume the axioms from the beginning as known facts.

Further examples of proofs: section 2.3. in the preliminary lecture notes.

How can we axiomatize further data structures?

Lists: Signature

Type: $\mathcal{L}$, elements: $\mathcal{T}$, we will also write $\mathcal{L}[\mathcal{T}]$

Function symbols:

- cons : $\mathcal{T} \times \mathcal{L}[\mathcal{T}] \rightarrow \mathcal{L}[\mathcal{T}]$
- first : $\mathcal{L}[\mathcal{T}] \rightarrow \mathcal{T}$
- rest : $\mathcal{L}[\mathcal{T}] \rightarrow \mathcal{L}[\mathcal{T}]$
- empty $:\rightarrow \mathcal{L}[\mathcal{T}]$

Usual abbreviation:

$$< x_1, \dots, x_n > \qquad \text{for} \qquad \text{cons}(x_1, \text{cons}(x_2, \dots, \text{cons}(x_n, \text{empty}()))$$

or something similar

Lists: Axiomatization

For $l \in \mathcal{L}[\mathcal{T}]$, $x \in \mathcal{T}$:

- $\forall l, x \,.\, \mathrm{first}(\mathrm{cons}(x, l)) = x$
- $\forall l, x \,.\, \mathrm{rest}(\mathrm{cons}(x, l)) = l$
- $\forall l \,.\, \neg[l = \mathrm{empty}()] \Rightarrow \mathrm{cons}(\mathrm{first}(l), \mathrm{rest}(l)) = l$
- $\forall l, x \,.\, \neg[\mathrm{cons}(x, l) = \mathrm{empty}()]$
- $\forall l_1, l_2 \,.\, l_1 = l_2 \Rightarrow [l_1 = \mathrm{empty}() \Leftrightarrow l_2 = \mathrm{empty}()]$

Behavior of first(empty()), rest(empty()) is not specified

We could introduce a constant error with first(empty()) = error and so on

Lists: Summary

Type: $\mathcal{L}$, elements: $\mathcal{T}$, we will also write $\mathcal{L}[\mathcal{T}]$

Function symbols:

- $\text{cons} : \mathcal{T} \times \mathcal{L}[\mathcal{T}] \to \mathcal{L}[\mathcal{T}]$
- $\text{first} : \mathcal{L}[\mathcal{T}] \to \mathcal{T}$
- $\text{rest} : \mathcal{L}[\mathcal{T}] \to \mathcal{L}[\mathcal{T}]$
- $\text{empty} :\to \mathcal{L}[\mathcal{T}]$

Axioms ($l \in \mathcal{L}[\mathcal{T}]$, $x \in \mathcal{T}$):

- $\forall l, x \,.\, \text{first}(\text{cons}(x, l)) = x$
- $\forall l, x \,.\, \text{rest}(\text{cons}(x, l)) = l$
- $\forall l, x \,.\, \neg[l = \text{empty}()] \Rightarrow \text{cons}(\text{first}(l), \text{rest}(l)) = l$
- $\forall l, x \,.\, \neg[\text{cons}(x, l) = \text{empty}()]$
- $\forall l_1, l_2 \,.\, l_1 = l_2 \Rightarrow [l_1 = \text{empty}() \Leftrightarrow l_2 = \text{empty}()]$

Example Proof

For proving

$$< 1 > \neq < 2 >$$

we first observe that this is a way of writing

$$\neg\ \mathrm{cons}(1, \mathrm{empty}()) = \mathrm{cons}(2, \mathrm{empty}()).$$

Using the prove rule for negation, we assume $< 1 > = < 2 >$ and try to find a contradiction. From this, using a lemma we can prove

$$\mathrm{first}(< 1 >) = \mathrm{first}(< 2 >).$$

From the axiom

$$\forall l, x\, .\, \mathrm{first}(\mathrm{cons}(x, l)) = x,$$

using the choices $l \leftarrow <>$, $x \leftarrow 1$ and $l \leftarrow <>$, $x \leftarrow 2$ respectively, we get

$$\mathrm{first}(< 1 >) = 1 \text{ and } \mathrm{first}(< 2 >) = 2.$$

This means that $1 = 2$, a contradiction.

Natural Numbers: Signature

Type: $\mathcal{N}$

Function Symbols:

- $0 : \to \mathcal{N}$
- $1 : \to \mathcal{N}$
- $+ : \mathcal{N} \times \mathcal{N} \to \mathcal{N}$
- $\cdot : \mathcal{N} \times \mathcal{N} \to \mathcal{N}$

3? abbreviation for $1 + 1 + 1$

every natural number *n*, abbreviation for

$$\underbrace{1 + \cdots + 1}_{n \text{ times}}$$

Natural Numbers: Axiomatization

Axioms (Peano arithmetic) [Peano, 1889]

- $\forall x \,.\, \neg[x+1=0]$
- $\forall x, y \,.\, x+1=y+1 \Rightarrow x=y$
- $\forall x \,.\, x+0=x$
- $\forall x, y \,.\, x+(y+1)=(x+y)+1$
- $\forall x \,.\, x0=0$
- $\forall x, y \,.\, x(y+1)=xy+x$

For each formula F with precisely one free variables x, a further axiom

$$\Big[F[x \leftarrow 0] \wedge \big[\forall x \,.\, F \Rightarrow F[x \leftarrow x+1]\big]\Big] \Rightarrow \forall x \,.\, F$$

In other words: induction

Infinite number of axioms!

Induction Axiom: Example

For each formula F with precisely one free variables x, a further axiom

$$\Big[F[x \leftarrow 0] \wedge \big[\forall x\,.\,F \Rightarrow F[x \leftarrow x+1]\big]\Big] \Rightarrow \forall x\,.\,F$$

Example: For the formula $0x = 0$,

$$\Big[00 = 0 \wedge \big[\forall x\,.\,0x = 0 \Rightarrow 0(x+1) = 0]\big]\Big] \Rightarrow \forall x\,.\,0x = 0$$

Usual usage: Prove the left-hand side using a lemma which implies the right-hand side.

Unfortunately, there is no finite axiomatization

Unsuccessful Proofs . . .

From previous lecture:

For showing that a formula does not hold we may

- find a counter-example, or
- try to prove the negation of the formula.

Second strategy may fail:
Example: neither $\models \forall x \, . \, P(x)$ nor $\models \neg\forall x \, . \, P(x)$.

Completeness $\sim$:
For every sentence ϕ, either ϕ or $\neg\phi$ follows from the axioms

. . . hence the second strategy can always be successful.

Attention:
(in)completeness of a theory $\neq$ (in)completeness of logical calculi!

Example of a complete theory:
Presburger arithmetic (Peano arithmetic without multiplication)

Incompleteness

Gödel's first incompleteness theorem:
Every theory that is strong enough to be able to express natural number arithmetic is incomplete [Gödel, 1931].

We still have good luck: In practice this is no problem.

Arrays

If an array has a known fixed size n:
  it is often enough to use n variables, otherwise:

Signature:

Type: $\mathcal{A}$, elements: $\mathcal{T}$, we will also write $\mathcal{A}[\mathcal{T}]$

Function symbols:

- $\cdot[\cdot] : \mathcal{A}[\mathcal{T}] \times \mathcal{N} \to \mathcal{T}$
- write : $\mathcal{A}[\mathcal{T}] \times \mathcal{N} \times \mathcal{T} \to \mathcal{A}[\mathcal{T}]$

Abbreviation: $\begin{array}{|c|c|c|c|c|}\hline 7 & 5 & \quad & 6 & 8 \\ \hline \end{array}$ for

$$\mathrm{write}(\mathrm{write}(\mathrm{write}(\mathrm{write}(a, 0, 7), 1, 5), 3, 6), 4, 8)$$

or, equivalently

$$\mathrm{write}(\mathrm{write}(\mathrm{write}(\mathrm{write}(a, 1, 5), 3, 6), 0, 7), 4, 8)$$

etc., where a is a new array (no new() operation needed)

Arrays: Axiomatization

Axioms $(a, b \in \mathcal{A}[\mathcal{T}], v \in \mathcal{T}, i, j \in \mathcal{N})$

- $\forall a, v, i, j \,.\, i = j \Rightarrow \mathrm{write}(a, i, v)[j] = v$
- $\forall a, v, i, j \,.\, \neg[i = j] \Rightarrow \mathrm{write}(a, i, v)[j] = a[j]$
- $\forall a, b \,.\, [\forall i \,.\, a[i] = b[i]] \Leftrightarrow a = b$

Attention: many programming languages: different array equality!

Prove of equality of two arrays: use third axiom,
replace the right-hand side equality by the left-hand side.

Here: no bounds on array length (can be checked separately)

Multi-dimensional arrays: 2 possibilities:

- arrays of arrays
- arrays with index set $\mathcal{N} \times \mathcal{N}$ etc.

Example Proof

$$\forall a \in \mathcal{A}, s \in \mathcal{N}, t \in \mathcal{N}, x, y \,.\, s \neq t \Rightarrow$$
$$\text{write}(\text{write}(a, s, x), t, y) = \text{write}(\text{write}(a, t, y), s, x)$$

Let $\forall a \in \mathcal{A}, s \in \mathcal{N}, t \in \mathcal{N}, x, y$ be arbitrary, but fixed with $s \neq t$. We prove

$$\text{write}(\text{write}(a, s, x), t, y) = \text{write}(\text{write}(a, t, y), s, x).$$

This means that we have to prove the equality of two arrays, and so we use the array axiom

$$\forall a, b \,.\, [\forall i \,.\, a[i] = b[i]] \Leftrightarrow a = b$$

Substituting write(write(a, s, x), t, y) for a and write(write(a, t, y), s, x) for b, we get

$$\forall i \,.\, \text{write}(\text{write}(a, s, x), t, y)[i] = \text{write}(\text{write}(a, t, y), s, x)[i] \Leftrightarrow$$
$$\text{write}(\text{write}(a, s, x), t, y) = \text{write}(\text{write}(a, t, y), s, x)$$

So, instead of proving the equality above, we prove

$$\forall i \,.\, \text{write}(\text{write}(a, s, x), t, y)[i] = \text{write}(\text{write}(a, s, x), t, y)[i].$$

Example Continued

$$\forall i \,.\, \mathrm{write}(\mathrm{write}(a,s,x),t,y)[i] = \mathrm{write}(\mathrm{write}(a,t,y),s,x)[i]$$

$$\mathrm{write}(\mathrm{write}(a,s,x),t,y)[i] = \mathrm{write}(\mathrm{write}(a,t,y),s,x)[i]$$

Now we have three cases:

- $i \neq s,\ i \neq t$:
  $\mathrm{write}(\mathrm{write}(a,s,x),t,y)[i] = \mathrm{write}(a,s,x)[i] = a[i]$
  $\mathrm{write}(\mathrm{write}(a,t,y),s,x)[i] = \mathrm{write}(a,t,y)[i] = a[i]$
- $i = s,\ i \neq t$:
  $\mathrm{write}(\mathrm{write}(a,s,x),t,y)[i] = \mathrm{write}(a,s,x)[i] = x$
  $\mathrm{write}(\mathrm{write}(a,t,y),s,x)[i] = x$
- $i \neq s,\ i = t$:
  $\mathrm{write}(\mathrm{write}(a,s,x),t,y)[i] = y$
  $\mathrm{write}(\mathrm{write}(a,t,y),s,x)[i] = \mathrm{write}(a,t,y) = y$

In all cases both sides are the same which finishes the proof

Further Theories

Extension of theory of natural numbers:
$\leq$?

$x \leq y :\Leftrightarrow \exists k \,.\, x + k = y$

Integers: $x - y$

Many further theories, e.g., the theory of real numbers.

Set Theory

Axiomatization: Zermelo-Fraenkel set theory (ZFC) (C: axiom of choice)

Too complicated for this course

Some rules for everyday usage

We form sets as follows: $\{x \mid A\}$, where A is a logical formula

For sets S and T,

- $S \cap T := \{x \mid x \in S \land x \in T\}$
- $S \cup T := \{x \mid x \in S \lor x \in T\}$
- $S \setminus T := \{x \mid x \in S \land x \notin T\}$
- The empty set is a set $\emptyset$ s.t. $\neg\exists x \,.\, x \in \emptyset$

Equivalences: For sets S and T,

- $S \subseteq T$ is equivalent to $\forall x \,.\, x \in S \Rightarrow x \in T$
- $S = T$ is equivalent to $\forall x \,.\, x \in S \Leftrightarrow x \in T$
- $a \in \{x \mid A\}$ is equivalent to $A[x \leftarrow a]$

Sets

Cartesian product: $S \times T \doteq \{(x, y) \mid x \in S, y \in T\}$

Power set: $\mathcal{P}(T) \doteq \{S \mid S \subseteq T\}$, sometimes also 2^S

Moreover:

- For a set S, $\{x \mid x \in S\} = S$

Very often the following notation is used:

- $\{a\}$ for $\{x \mid x = a\}$
- $\{a_1, \ldots, a_n\}$ for $\{x \mid x = a_1 \vee \cdots \vee x = a_n\}$
- $\forall x \in S \,.\, A$ for $\forall x \,.\, x \in S \Rightarrow A$
- $\exists x \in S \,.\, A$ for $\exists x \,.\, x \in S \wedge A$
- $x \notin A$ for $\neg[x \in A]$
- $\bigcap$, $\bigcup$, $\ldots$

Set Theory as The Foundation of Mathematics

In set theory everything seen so far can be formalized!

Example: Pairs [Kuratowski, 1921]

- $\forall x, y \,.\, \mathrm{pair}(x, y) := \{\{x\}, \{x, y\}\}$
- $\forall z, p \,.\, z = \mathrm{fst}(p) :\Leftrightarrow \forall \alpha \in p \,.\, z \in \alpha$
- $\forall z, p \,.\, z = \mathrm{snd}(p) :\Leftrightarrow \exists^1 \alpha \in p \;.\, z \in \alpha$

Further examples:

- Functions, relations: sets of pairs
- Natural numbers: 0: $\emptyset$, $x + 1$: $\{x\}$
- Arrays: functions from the natural numbers
- Lists: either the empty set, or a pair consisting of an element and the rest of the list

Consequences

We can formalize the natural numbers in set theory

Consequence: again Gödel!

Every theory that is strong enough to be able to express natural number arithmetic is incomplete [Gödel, 1931].

Again we have good luck:
 In practice, predicate logic + set theory is enough for
  building all of mathematics (i.e., everything that can be formalized).

at least until now . . . .

So why use different theories at all?

Decision Procedures

Some logical theories are *decidable*, that is, there is an algorithm with

- Input: formula ϕ with symbols from $\mathcal{T}$
- Output: $\top$, if ϕ holds in the theory $\mathcal{T}$, $\bot$ otherwise.

Most of our theories are decidable (demo)

In such cases we can automatically
prove the correctness of algorithms
that consist of just one single return statement.

But: Peano arithmetic, set theory are undecidable
[Church, 1936, Turing, 1937]

Presburger arithmetic: decidable

In practice, software often

- checks satisfiability instead of validity,
- also returns counter-examples.

Discussion

Why do we need

- integers
  (computer integers are only a finite subset),
- real numbers
  (in computer we have floating point numbers), and
- dynamical data structures
  (computers have only finite memory)?

Each individual computer only has a finite number of bits,
i.e., an algorithm has only finite behavior,
we only would have to test all of it!

In hardware verification this is sometimes done (see SAT),
but in the case of software we usually do not have a chance with this.

Axiomatization and Parametric Data Types

For example: We have a list where we want to store
integers, rational numbers, arrays etc.

Parametrization can be
dynamic (i.e., sub-typing) or static (i.e., templates).

It suffices to axiomatize the required behavior of allowed types.

Example: The elements of the list will be a type, that is a group.

We can use the axioms of group theory:

- Associativity: $\forall x, y, z \, . \, x + (y + z) = (x + y) + z$
- 0 is neutral element: $\forall x \, . \, 0 + x = x \wedge x + 0 = x$
- Existence of inverse: $\forall x \exists v \, . \, x + v = 0 \wedge v + x = 0$

Combination Procedures

Programs usually do not use
only integers, or only lists, arrays etc.

. . . but combinations of various data structures.

Combination procedures:
Algorithms that can in certain cases
decide sentences from a combination of various theories

Course Forecast

and now . . . apply all of this to software . . .

Literature I

Alonzo Church. An unsolvable problem of elementary number theory. *Am. J. Math.*, 58:345–363, 1936.

Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik*, 38:173–198, 1931.

Casimir Kuratowski. Sur la notion de l'ordre dans la théorie des ensembles. *Fundamenta mathematicae*, 2(1):161–171, 1921.

Giuseppe Peano. *Arithmetices principia: nova methodo*. Turino, 1889.

A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.