Outline

Introduction
- Correctness of Computer Systems
- Theorem Proving

Computer Systems and Correctness

Suppose we design a (complex) computer system, which may contain various components, for example, hardware, software etc.
We have high requirements to the correctness of the system (safety, reliability, security, consistent state, no deadlocks etc.)
How can one achive a 100% safety?
Computer systems are becoming increasingly unreliable.

Small Example: Software

Consider the following fragment of a C++ program:

```
int sumOfFirstNIntegers(int n)
  requires n >= 0
  ensures result = n * (n+1) / 2
{
  int sum = 0;
  for (i = n;i != 0;i = i-1) { sum = sum+i; }
  return sum;
}
```

We know that

$$1 + \ldots + n = \frac{n \cdot (n+1)}{2}$$

Is it true that for all integer n the program returns $\frac{n \cdot (n+1)}{2}$?

We can write a Spec#-specification.

How can we **prove** automatically that the program is correct w.r.t. this specification?

Small Example: Software

Consider the following fragment of a C++ program:

```
int sumOfFirstNIntegers(int n)
  requires n >= 0
  ensures result = n * (n+1) / 2
{
  int sum = 0;
  for (i = n;i != 0;i = i-1) { sum = sum+i; }
  return sum;
}
```

We know that

$$1 + \ldots + n = \frac{n \cdot (n+1)}{2}$$

Is it true that for all integer n the program returns $\frac{n \cdot (n+1)}{2}$?

We can write a Spec#-specification.

How can we **prove** automatically that the program is correct w.r.t. this specification?

Small Example: Software

Consider the following fragment of a C++ program:

```
int sumOfFirstNIntegers(int n)
  requires n >= 0
  ensures result = n * (n+1) / 2
{
  int sum = 0;
  for (i = n;i != 0;i = i-1) { sum = sum+i; }
  return sum;
}
```

We know that

$$1 + \ldots + n = \frac{n \cdot (n+1)}{2}$$

Is it true that for all integer n the program returns $\frac{n \cdot (n+1)}{2}$?
We can write a Spec#-specification.
How can we **prove** automatically that the program is correct w.r.t. this specification?

Small Example: Software

Consider the following fragment of a C++ program:

```
int sumOfFirstNIntegers(int n)
  requires n >= 0
  ensures result = n * (n+1) / 2
{
  int sum = 0;
  for (i = n;i != 0;i = i-1) { sum = sum+i; }
  return sum;
}
```

We know that

$$1 + \ldots + n = \frac{n \cdot (n+1)}{2}$$

Is it true that for all integer n the program returns $\frac{n \cdot (n+1)}{2}$?
We can write a Spec#-specification.
How can we **prove** automatically that the program is correct w.r.t. this specification?

Another example: circuit design

We used a circuit C_1 in a processor and would like to replace it by another circuit C_2. For example, we may believe that the use of C_2 results in a lower energy consumption.

We want to be sure that C_2 is correct, that is, it will behave according to some specification.

If we know that C_1 is correct, it is sufficient to **prove** that C_2 is functionally equivalent to C_1.

Another example: circuit design

We used a circuit C_1 in a processor and would like to replace it by another circuit C_2. For example, we may believe that the use of C_2 results in a lower energy consumption.

We want to be sure that C_2 is correct, that is, it will behave according to some specification.

If we know that C_1 is correct, it is sufficient to **prove** that C_2 is functionally equivalent to C_1.

Another example: circuit design

We used a circuit C_1 in a processor and would like to replace it by another circuit C_2. For example, we may believe that the use of C_2 results in a lower energy consumption.
We want to be sure that C_2 is correct, that is, it will behave according to some specification.
If we know that C_1 is correct, it is sufficient to **prove** that C_2 is functionally equivalent to C_1.

Automated Theorem Proving. Example

Group theory theorem: if a group satisfies the identity $x^2 = 1$, then it is commutative.

More formally: in a group "assuming that $x^2 = 1$ for all x prove that $x \cdot y = y \cdot x$ holds for all x, y."

What is implicit: axioms of the group theory.

$$\forall x(1 \cdot x = x)$$
$$\forall x(x^{-1} \cdot x = 1)$$
$$\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$$

Automated Theorem Proving. Example

Group theory theorem: if a group satisfies the identity $x^2 = 1$, then it is commutative.

More formally: in a group "assuming that $x^2 = 1$ for all x prove that $x \cdot y = y \cdot x$ holds for all x, y."

What is implicit: axioms of the group theory.

$$\forall x(1 \cdot x = x)$$
$$\forall x(x^{-1} \cdot x = 1)$$
$$\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$$

Automated Theorem Proving. Example

Group theory theorem: if a group satisfies the identity $x^2 = 1$, then it is commutative.

More formally: in a group "assuming that $x^2 = 1$ for all x prove that $x \cdot y = y \cdot x$ holds for all x, y."

What is implicit: axioms of the group theory.

$$\forall x(1 \cdot x = x)$$
$$\forall x(x^{-1} \cdot x = 1)$$
$$\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$$

Formulation in First-Order Logic

Axioms (of group theory): $\forall x(1 \cdot x = x)$
$\forall x(x^{-1} \cdot x = 1)$
$\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$

Assumptions: $\forall x(x \cdot x = 1)$

Conjecture: $\forall x \forall y(x \cdot y = y \cdot x)$

Formulation in First-Order Logic

Axioms (of group theory):	$\forall x(1 \cdot x = x)$ $\forall x(x^{-1} \cdot x = 1)$ $\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$
Assumptions:	$\forall x(x \cdot x = 1)$
Conjecture:	$\forall x \forall y(x \cdot y = y \cdot x)$

Formulation in First-Order Logic

Axioms (of group theory):	$\forall x(1 \cdot x = x)$ $\forall x(x^{-1} \cdot x = 1)$ $\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$
Assumptions:	$\forall x(x \cdot x = 1)$
Conjecture:	$\forall x \forall y(x \cdot y = y \cdot x)$

In the TPTP Syntax

TPTP library (Thousands of Problems for Theorem Provers), www.tptp.org.

```
%---- 1 * x = 1
fof(left_identity,axiom,
  mult(e,X) = X.
%---- i(x) * x = 1
fof(left_inverse,axiom,
  mult(inverse(X),X) = e).
%---- (x * y) * z = x * (y * z)
fof(associativity,axiom,
  mult(mult(X,Y),Z) = mult(X,mult(Y,Z))).
%---- x * x = 1
fof(group_of_order_2,hypothesis,
  mult(X,X) = e).
%---- prove x * y = y * x
fof(commutativity,conjecture,
  mult(X,Y) = mult(Y,X)).
```

Example: Proof by Vampire

.....

Theorem Provers

Theorem Prover: a system that can prove theorems automatically.

Two kinds of provers:

- automatic provers;
- interactive provers, or proof assistants.

Logics:

- in automatic provers mainly first-order logic (with built-in equality);
- in interactive provers higher-order logics or type theories.

This course will be mainly about fully automatic theorem provers for first-order logic.

Theorem Provers

Theorem Prover: a system that can prove theorems automatically.
Two kinds of provers:

- automatic provers;
- interactive provers, or proof assistants.

Logics:

- in automatic provers mainly first-order logic (with built-in equality);
- in interactive provers higher-order logics or type theories.

This course will be mainly about fully automatic theorem provers for first-order logic.

Theorem Provers

Theorem Prover: a system that can prove theorems automatically.
Two kinds of provers:

- automatic provers;
- interactive provers, or proof assistants.

Logics:

- in automatic provers mainly first-order logic (with built-in equality);
- in interactive provers higher-order logics or type theories.

This course will be mainly about fully automatic theorem provers for first-order logic.

Theorem Provers

Theorem Prover: a system that can prove theorems automatically.
Two kinds of provers:

- automatic provers;
- interactive provers, or proof assistants.

Logics:

- in automatic provers mainly first-order logic (with built-in equality);
- in interactive provers higher-order logics or type theories.

This course will be mainly about fully automatic theorem provers for first-order logic.

Main applications

- Software and hardware verification;
- Static analysis of programs;
- Query answering in first-order knowledge bases (ontologies), Semantic Web;
- Theorem proving in mathematics, especially in algebra;
- Verification of cryptographic protocols;
- Circuit design;
- Constraint satisfaction;
- Planning;
- Databases (semantics and query optimisation);
- Solving exercises for this course $\ddot{\smile}$

What We Expect of an Automatic Theorem Prover

Input:

- a set of axioms (first order formulas) or clauses;
- a conjecture (first-order formula or set of clauses).

Output:

- proof (hopefully).

What We Expect of an Automatic Theorem Prover

Input:

- a set of axioms (first order formulas) or clauses;
- a conjecture (first-order formula or set of clauses).

Output:

- proof (hopefully).