Formal Reasoning in Logic and Programming Languages

Prof. Dr.-Ing. Sebastian Schlesinger

Berlin School for Economics and Law

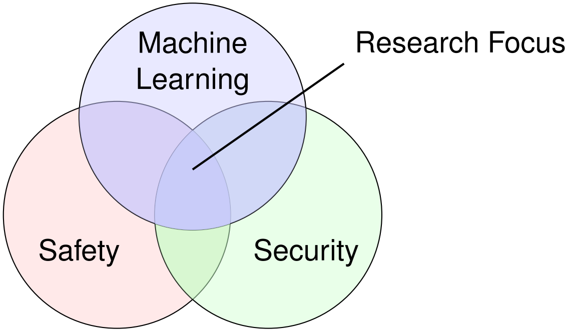
June 15, 2024

Objectives

- Obtain an initial understanding of formal concepts
- Survey of classical and recent approaches to formal verification
- Also establish the bridge to related work and future research directions I am aiming at

My Research Focus

My background: **formal verification** (particularly model-driven engineering of embedded or cyber-physical systems) and **security**. Recently, also **machine learning**.
So, in essence, I am interested in **safety** and **security** of **AI-enabled systems** or the application of **Machine Learning** to classical approaches for the verification of safety and security of systems.



Outline

Language of first-order logic

A language $\mathscr{L}$ of first-order logic consists of the following components:

- Variable symbols: $x_1, x_2, \ldots$
- For each $n \in \mathbb{N}$, a set of n-ary function symbols: $f_0, f_1, \ldots$ The 0-ary function symbols are called constant symbols.
- For each $n \in \mathbb{N}$, a set of n-ary predicate symbols: $p_0, p_1, \ldots$ The 0-ary predicate symbols are the constants $\top$ (for **true**) and $\bot$ (for **false**).
- special symbols: $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication), $\leftrightarrow$ (equivalence), $\forall$ (universal quantification), $\exists$ (existential quantification), and parentheses.

Terms

The set of terms of $\mathscr{L}$ is defined inductively as follows:

- Each variable is a term.
- If $t_1, \ldots, t_n$ are terms and f is an n-ary function symbol, then if $f(t_1, \ldots, t_n)$ is a term.

Variables in terms

We define a function var : Terms $\rightarrow$ Variables that maps each term to the set of variables occurring in it. The function is defined as follows:

- $var(x) = \{x\}$ for each variable x.
- $var(f(t_1, \ldots, t_n)) = var(t_1) \cup \ldots \cup var(t_n)$.

Formulas

The set of formulas of $\mathscr{L}$ is defined inductively as follows:

- If $t_1, \ldots, t_n$ are terms and p is an n-ary predicate symbol, then if $p(t_1, \ldots, t_n)$ is a formula.
- If φ is a formula, then if $\neg\varphi$ is a formula.
- If φ_1 and φ_2 are formulas, then if $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, and $\varphi_1 \leftrightarrow \varphi_2$ are formulas.
- If φ is a formula and x is a variable, then if $\forall x.\varphi$ and $\exists x.\varphi$ are formulas.

An example of a formula is $\forall x.\exists y.p(x, y) \rightarrow \neg q(y)$.

Interpretations

An interpretation $\mathcal{M}$ of $\mathscr{L}$ consists of the following components:

- A non-empty set D called the domain of $\mathcal{M}$.
- For each n-ary function symbol f of $\mathscr{L}$, a function $f^{\mathcal{M}} : D^n \to D$.
- For each n-ary predicate symbol p of $\mathscr{L}$, a relation $p^{\mathcal{M}} \subseteq D^n$.

Interpretations of Terms

Let $\mathcal{M}$ be an interpretation for our first-order language. An assignment σ of values to variables, i.e., $\sigma : Variables \to D$.
The value of a term t under σ is denoted by $t^{\mathcal{M}}[\sigma]$ and defined as follows:

- If $t = x$ for a variable x, then $t^{\mathcal{M}}[\sigma] = \sigma(x)$.
- If $t = f(t_1, \ldots, t_n)$, then $t^{\mathcal{M}}[\sigma] = f^{\mathcal{M}}(t_1^{\mathcal{M}}[\sigma], \ldots, t_n^{\mathcal{M}}[\sigma])$.

Validity of Formulas under Interpretations

We say an assignment σ satisfies a formula φ under an interpretation $\mathcal{M}$, denoted by $\mathcal{M}, \sigma \models \varphi$, iff the following conditions hold:

- $\varphi = p(t_1, \ldots, t_n)$, then if $(t_1^{\mathcal{M}}[\sigma], \ldots, t_n^{\mathcal{M}}[\sigma]) \in p^{\mathcal{M}}$.
- $\varphi = \neg\psi$, then if $\mathcal{M}, \sigma \not\models \psi$.
- $\varphi = \psi_1 \vee \psi_2$, then if $\mathcal{M}, \sigma \models \psi_1$ or $\mathcal{M}, \sigma \models \psi_2$.
- $\varphi = \psi_1 \wedge \psi_2$, then if $\mathcal{M}, \sigma \models \psi_1$ and $\mathcal{M}, \sigma \models \psi_2$.
- $\varphi = \psi_1 \to \psi_2$, then if $\mathcal{M}, \sigma \models \psi_1$ implies $\mathcal{M}, \sigma \models \psi_2$.
- $\varphi = \psi_1 \leftrightarrow \psi_2$, then if $\mathcal{M}, \sigma \models \psi_1$ if and only if $\mathcal{M}, \sigma \models \psi_2$.
- $\varphi = \forall x.\psi$, then if $\mathcal{M}, \sigma[x \mapsto d] \models \psi$ for all $d \in D$.
- $\varphi = \exists x.\psi$, then if $\mathcal{M}, \sigma[x \mapsto d] \models \psi$ for some $d \in D$.

A formula φ is satisfiable if there exists an interpretation $\mathcal{M}$ and an assignment σ such that $\mathcal{M}, \sigma \models \varphi$.

Models

An interpretation $\mathcal{M}$ is a model of a formula φ, denoted by $\mathcal{M} \models \varphi$, if for all assignments σ, $\mathcal{M}, \sigma \models \varphi$.

A formula is satisfiable if it has a model, i.e., if there exists an interpretation $\mathcal{M}$ such that $\mathcal{M} \models \varphi$.

Validity

A formula φ is valid if for all interpretations $\mathcal{M}$ and all assignments σ, $\mathcal{M}, \sigma \models \varphi$.
We write $\models \varphi$ to denote that φ is valid.

Free Variables in Fomulas

The set of free variables of a formula φ, denoted by $FV(\varphi)$, is defined inductively as follows:

- $FV(p(t_1, \dots, t_n)) = var(t_1) \cup \dots \cup var(t_n)$.
- $FV(\neg\psi) = FV(\psi)$.
- $FV(\psi_1 \wedge \psi_2) = FV(\psi_1) \cup FV(\psi_2)$.
- $FV(\psi_1 \vee \psi_2) = FV(\psi_1) \cup FV(\psi_2)$.
- $FV(\psi_1 \rightarrow \psi_2) = FV(\psi_1) \cup FV(\psi_2)$.
- $FV(\forall x.\psi) = FV(\psi) \setminus \{x\}$.
- $FV(\exists x.\psi) = FV(\psi) \setminus \{x\}$.

Term Substitution

Let φ be a formula, x a variable, and t a term. The formula $\varphi[t/x]$ is obtained by replacing all occurrences of x in φ by t. The substitution is defined inductively as follows:

- $(p(t_1, \dots, t_n))[t/x] = p(t_1[t/x], \dots, t_n[t/x])$.
- $(\neg\psi)[t/x] = \neg\psi[t/x]$.
- $(\psi_1 \land \psi_2)[t/x] = \psi_1[t.x] \land \psi_2[t/x]$.
- $(\psi_1 \lor \psi_2)[t/x] = \psi_1[t/x] \lor \psi_2[t/x]$.
- $(\psi_1 \to \psi_2)[t/x] = \psi_1[t/x] \to \psi_2[t/x]$.
- $(\forall y.\psi)[t/x] = \forall y.\psi[t/x]$ if $x \in FV(t)$.
- $(\exists y.\psi)[t/x] = \exists y.\psi[t/x]$ if $x \in FV(t)$.
- $(\forall x.\psi)[t/x] = \forall x.\psi$.
- $(\exists x.\psi)[t/x] = \exists x.\psi$.

So, $\varphi[t/x]$ represents the formular obtained by substituting every **free** occurrence of the variable x in φ by the term t.

Calculus

A calculus is a mechanism to prove formulas by applying rules.
A rule of a calculus has the form $\frac{\varphi_1,\dots,\varphi_n}{\psi}$, where $\varphi_1, \dots, \varphi_n$ are premises and ψ is the conclusion. The rule states that if $\varphi_1, \dots, \varphi_n$ are derivable, then ψ is derivable.
We denote that a formula can be proved by a calculus by $\vdash \varphi$.

Sequent Calculus

In sequent calculus, we have sequences $\Gamma \vdash \Delta$, where Γ and Δ are sets of formulas.

The interpretation is that if all formulas in Γ are true, then at least one formula in Δ is true.

Sequent Calculus Rules

$$\frac{\text{-}}{\Gamma, \varphi \Rightarrow \varphi, \Delta}\ \text{Taut}$$

$$\frac{\Gamma \Rightarrow \Delta, \varphi \qquad \varphi, \Pi \Rightarrow \Lambda}{\Gamma, \Pi \Rightarrow \Delta, \Lambda}\ \text{Cut}$$

$$\frac{\text{-}}{\Gamma, \bot \Rightarrow \Delta}\ \bot \Rightarrow$$

$$\frac{\text{-}}{\Gamma \Rightarrow \Delta, \top}\ \Rightarrow \top$$

$$\frac{\Gamma \Rightarrow \Delta}{\varphi, \Gamma \Rightarrow \Delta}\ \text{Weakening left}$$

$$\frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \varphi}\ \text{Weakening right}$$

$$\frac{\varphi, \varphi, \Gamma \Rightarrow \Delta}{\varphi, \Gamma \Rightarrow \Delta}\ \text{Contraction left}$$

$$\frac{\Gamma \Rightarrow \Delta, \varphi, \varphi}{\Gamma \Rightarrow \Delta, \varphi}\ \text{Contraction right}$$

$$\frac{\Gamma, \varphi, \psi, \Pi \Rightarrow \Delta}{\Gamma, \psi, \varphi, \Pi \Rightarrow \Delta}\ \text{Exchange left}$$

$$\frac{\Gamma \Rightarrow \Delta, \varphi, \psi, \Lambda}{\Gamma \Rightarrow \Delta, \psi, \varphi, \Lambda}\ \text{Exchange right}$$

Sequent Calculus Rules

$$\frac{-}{\Gamma, \bot \Rightarrow \Delta} \bot \Rightarrow \qquad \frac{-}{\Gamma \Rightarrow \Delta, \top} \Rightarrow \top$$

$$\frac{\Gamma \Rightarrow \Delta, \varphi}{\Gamma, \neg\varphi \Rightarrow \Delta} \neg \Rightarrow \qquad \frac{\Gamma, \varphi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg\varphi} \Rightarrow \neg$$

$$\frac{\Gamma, \varphi \Rightarrow \Delta \quad \Gamma, \psi \Rightarrow \Delta}{\Gamma, \varphi \vee \psi \Rightarrow \Delta} \vee \Rightarrow \qquad \frac{\Gamma \Rightarrow \Delta, \varphi, \psi}{\Gamma \Rightarrow \Delta, \varphi \vee \psi} \Rightarrow \vee$$

$$\frac{\Gamma, \varphi, \psi \Rightarrow \Delta}{\Gamma, \varphi \wedge \psi \Rightarrow \Delta} \wedge \Rightarrow \qquad \frac{\Gamma \Rightarrow \Delta, \varphi \quad \Gamma \Rightarrow \Delta, \psi}{\Gamma \Rightarrow \Delta, \varphi \wedge \psi} \Rightarrow \wedge$$

$$\frac{\Gamma \Rightarrow \Delta, \varphi \quad \psi, \Pi \Rightarrow \Lambda}{\varphi \rightarrow \psi, \Gamma, \Pi \Rightarrow \Delta, \Lambda} \rightarrow\Rightarrow \qquad \frac{\Gamma, \varphi \Rightarrow \Delta, \psi}{\Gamma \Rightarrow \Delta, \varphi \rightarrow \psi} \Rightarrow\rightarrow$$

Sequent Calculus Rules

$$\frac{\Gamma, \varphi[t/x] \Rightarrow \Delta}{\Gamma, \forall x.\varphi(x) \Rightarrow \Delta} \forall\Rightarrow \qquad \frac{\Gamma \Rightarrow \Delta, \varphi[y/x]}{\Gamma \Rightarrow \Delta, \forall x.\varphi(x)} \Rightarrow\forall$$

$$\frac{\Gamma, \varphi[y/x] \Rightarrow \Delta}{\Gamma, \exists x.\varphi(x) \Rightarrow \Delta} \exists\Rightarrow \qquad \frac{\Gamma \Rightarrow \Delta, \exists x.\varphi(x), \varphi[t/x]}{\Gamma \Rightarrow \Delta, \exists x.\varphi(x)} \Rightarrow\exists$$

In the quantifier rules, t is a term, and y is a 'fresh' variable, i.e., a variable that does not occur in Γ, Δ, or φ.
Alternatively, the rules can also be stated in the form

$$\frac{\Gamma \Rightarrow \Delta, \varphi}{\Gamma \Rightarrow \Delta, \forall x.\varphi(x)} \Rightarrow\forall$$

Here, it must be guaranteed that x is not free in any formula in Γ or Δ.
The existential formula can be handled similarly.

Example Deduction: $\forall x.(P(x) \land Q \Rightarrow \forall x.P(x)$

$$\cfrac{\cfrac{\cfrac{\cfrac{}{P(x), Q \Rightarrow P(x)}\,\text{Taut}}{P(x) \land Q \Rightarrow P(x)}\,\land\Rightarrow}{\forall x.(P(x \land Q) \Rightarrow P(x))}\,\forall\Rightarrow}{\forall x.(P(x) \land Q) \Rightarrow \forall x.P(x)}\,\Rightarrow\forall$$

Here, $\forall \Rightarrow$ uses $[x/x]$ as replacement, i.e., just the same free variable is taken.

Example Deduction: $\forall x.(A \to B) \Rightarrow A \to \forall x.B$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \Rightarrow A, B}\,\text{Taut} \qquad \dfrac{}{A, B \Rightarrow B}\,\text{Taut}}{A, A \to B \Rightarrow B}\,{\to}{\Rightarrow}}{A, \forall x.(A \to B) \Rightarrow B}\,\forall \Rightarrow}{A, \forall x.(A \to B) \Rightarrow \forall x.B}\,\Rightarrow \forall}{\forall x.(A \to B) \Rightarrow A \to \forall x.B}\,{\Rightarrow}{\to}$$

Here, the application of $\Rightarrow \forall$ requires that x is not free in A.

Example of a Failing Deduction: $\exists x.P(x) \land \exists x.Q(x) \Rightarrow \exists x.(P(x) \land Q(x))$

$$\dfrac{\dfrac{\dfrac{\dfrac{P(x), Q(y) \Rightarrow P(x) \land Q(x)}{P(x), Q(y) \Rightarrow \exists x.(P(x) \land Q(x))}\ {\Rightarrow \exists}}{P(x), \exists x.Q(x) \Rightarrow \exists x.(P(x) \land Q(x))}\ {\exists \Rightarrow}}{\exists x.P(x), \exists x.Q(x) \Rightarrow \exists x.(P(x) \land Q(x))}\ {\exists \Rightarrow}}{\exists x.P(x) \land \exists x.Q(x) \Rightarrow \exists x.(P(x) \land Q(x))}\ {\land \Rightarrow}$$

Here, the deduction fails because the variable x is not fresh in the application of $\exists \Rightarrow$ and therefore the new variable y is introduced. However, then the deduction cannot be completed.

Soundness and Completeness of Sequent Calculus

- A calculus is sound if all provable formulas are valid, denoted by $\vdash \varphi \Rightarrow \models \varphi$.
- A calculus is complete if all valid formulas are provable, denoted by $\models \varphi \Rightarrow \vdash \varphi$.

The sequent calculus is sound and complete for first-order logic, i.e.,

- $\vdash \Gamma \Rightarrow \Delta$, then $\models \Gamma \Rightarrow \Delta$.
- $\models \Gamma \Rightarrow \Delta$, then $\models \Gamma \Rightarrow \Delta$.

Proof of soundness by induction on the structure of the formulas. Proof of completeness by constructing a Herbrand model.

Goedel's Incompleteness Theorem

- Goedel's first incompleteness theorem states that in any consistent formal system that is powerful enough to express arithmetic, there are true statements that cannot be proven.
- Goedel's second incompleteness theorem states that in any consistent formal system that is powerful enough to express arithmetic, the system cannot prove its own consistency.

Rice’s Theorem

- Rice’s theorem states that for any non-trivial property of partial functions, i.e., a property that is not true for all partial functions or not true for none, there is no algorithm that can decide whether a given program has that property.
- A property is non-trivial if there are two partial functions that are computable and one has the property and the other does not.

Halting Problem

The halting problem is the problem of determining, given a program and an input, whether the program will eventually halt when run with that input.
The halting problem is undecidable, i.e., there is no algorithm that can decide whether a given program halts on a given input.

Some Conclusions

- There are properties of programs that cannot be decided by an algorithm.
- There are properties of programs that cannot be verified by a formal system.
- It is impossible to generally prove behavioural equivalence of programs.

Semantics of programs

There are three main types of semantics for programs:

- Operational semantics: Describes the execution of programs.
- Denotational semantics: Describes the meaning of programs (as a mathematical mapping of states).
- Axiomatic semantics: Describes properties of programs.

The while language

The while language is a simple imperative programming language with the following constructs:

- Arithmetic expressions: $E ::= n \mid x \mid E + E \mid E - E \mid E * E \mid E/E$ (i.e., terms), where n is a number and x is a variable.
- Boolean expressions: $B ::=$ true $\mid$ false $\mid E = E \mid E < E \mid E \leq E \mid$ not $B \mid B$ and $B \mid B$ or B.
- Statements:
  $S ::=$ skip $\mid x := E \mid S_1 ; S_2 \mid$ if B then S_1 else $S_2 \mid$ while B do S.

Semantics domains for while

- Values: $V = \mathbb{Z} \cup \{\text{true}, \text{false}\}$.
- Interpretation for constants: $V \to \mathbb{Z}$
- States: Σ : **Var** $\to V$, where **Var** is the set of variables. We denote an update of a state by $\sigma' = \sigma[x \mapsto v]$, which means that $\sigma'(x) = v$ and $\sigma'(y) = \sigma(y)$ for $y \neq x$.
- Expression interpretation: $E \to \Sigma \to \mathbb{Z}$. An application of an expression under a state is traditionally written as $val[\![E]\!]\sigma$.

The interpretation of an expression under a state is defined by induction on the structure of the expression.

- $val[\![n]\!]\sigma = n$
- $val[\![x]\!]\sigma = \sigma(x)$
- $val[\![E_1 + E_2]\!]\sigma = val[\![E_1]\!]\sigma + val[\![E_2]\!]\sigma$
- $val[\![\text{true}]\!]\sigma = \text{true}$
- $val[\![E_1 = E_2]\!]\sigma =$ true if $val[\![E_1]\!]\sigma = val[\![E_2]\!]\sigma$ and false otherwise.
- etc.

Operational Semantics of while

It describes how the execution of while programs is done operationally.
A transition system is a triple $(\Gamma, T, \rightarrow)$ where

- Γ is a set of configurations. A configuration is a pair (c, σ), where c is a command and σ is a state.
- T is a set of terminal configurations.
- $\rightarrow \subseteq \Gamma \times \Gamma$ is a transition relation ($\rightarrow^*$ is the reflexive transitive closure of $\rightarrow$, $\rightarrow^+$ is the transitive closure of $\rightarrow$).

There are two types of operational semantics:

- Small-step semantics: Describes the execution of a program step by step.
- Big-step semantics: Describes the execution of a program in one step.

We'll focus on small-step semantics.

Operational Semantics of while

$$\frac{}{x := E \to \sigma[x \mapsto val[\![E]\!]\sigma]}\ \text{Assign}$$

$$\frac{}{(\text{skip}, c) \to \sigma}\ \text{Skip}$$

$$\frac{(c_1, \sigma) \to \sigma'}{(c_1; c_2, \sigma) \to (c_2, \sigma')}\ \text{Seq1} \qquad \frac{(c_1, \sigma) \to (c_1', \sigma')}{(c_1; c_2, \sigma) \to (c_1'; c_2, \sigma')}\ \text{Seq2}$$

$$\frac{}{(\text{if } b \text{ then } c_1 \text{ else } c_2, \sigma) \to (c_1, \sigma)}\ \text{IF for } val[\![B]\!]\sigma = \text{true}$$

$$\frac{}{(\text{if } b \text{ then } c_1 \text{ else } c_2, \sigma) \to (c_2, \sigma)}\ \text{IF for } val[\![B]\!]\sigma = \text{false}$$

$$\frac{}{(\text{while } B \text{ do } c, \sigma) \to (\text{if } B \text{ then } c; \text{while } B \text{ do } c \text{ else skip}, \sigma)}\ \text{WHILE}$$

Denotational semantics of while

While the operational semantics describes the execution of programs in a step-by-step manner via relations, the denotational semantics describes the meaning of programs as a mathematical mapping of states.
We define $[\![\cdot]\!] : \Sigma \to \Sigma$ as the denotational semantics of the while language. The denotational semantics of the while language is defined by induction on the structure of the program.

Denotational semantics of while

$$[\![\text{skip}]\!](\sigma) = \sigma$$

$$[\![x := E]\!](\sigma) = \sigma[x \mapsto val[\![E]\!]\sigma]$$

$$[\![c_1; c_2]\!] = [\![c_2]\!] \circ [\![c_1]\!]$$

$$[\![\text{if } b \text{ then } c_1 \text{ else } c_2]\!] = [\![c_1]\!] \text{ if } val[\![b]\!]\sigma = \text{true, otherwise} [\![c_2]\!]$$

$$[\![\text{while } B \text{ do } c]\!] = [\![\text{skip}]\!] \text{ if } val[\![b]\!]\sigma = \text{false, otherwise } [\![\text{while } B \text{ do } c]\!] \circ [\![c]\!]$$

However, the last definition is not well-defined, as it is not guaranteed that the while loop will terminate. Therefore, we need to define a fixpoint semantics for the while language.

CPOs

A complete partial order (CPO) is a partially ordered set $(M, \leq)$ where

- M has a least element $\bot$.
- Each chain $x_1 \leq x_2 \leq x_3 \leq \ldots$ has a least upper bound $\bigsqcup x_i$ in M.

Least Fixed Point

If for a function $M \to M$ a fixed point, i.e., a point x such that $f(x) = x$, exists, then we denote the least fixed point as μ_f, i.e., $x = \mu_f$ if x is a fixed point and $x \leq y$ for all fixed points y.

Knaster-Tarski Theorem

For a cpo $(M, \leq)$ and a function $f : M \to M$, the least fixed point μ_f exists and the following holds:

$$\mu_f = \bigsqcup_{i \geq 0} f^{(i)}(\bot)$$

Denotational Semantics for while statement revised

We now can define the semantics for while B do c as fixed point of a function $F : (\Sigma \to \Sigma) \to (\Sigma \to \Sigma)$ with

$$F(f)(\sigma) = f([\![c]\!](\sigma)) \text{ if } val[\![b]\!]\sigma = \text{true, otherwise } \sigma$$

Note that in our definitions we did not consider the termination of the while loop.