Semantic Equivalence

Semantics and applications to verification

Xavier Rival

École Normale Supérieure

Reasoning on program equivalence

Properties considered in the previous lectures:

- Sets of **states**: absence of runtime errors...
- Sets of **traces**: termination
- Sets of **sets or traces**: security, dependences

In all these cases, **only one program is considered**.

Today: reasoning about several programs

Kinds of questions we will consider:

- Does P and Q have the same behaviors ?
- Does P have more behaviors than Q ?

A short introduction to these properties and **the verification of a micro-compiler** with Coq

Program transformations

Informal definition: program transformation

- A **program transformation** is a **partial function**, mapping a program P into another program $\mathcal{T}(P)$
- It should **preserve some semantic properties** of programs

- **Compilation:**
  the target code behaviors should match those of the source code
- **Optimization:** the target code may differ strongly from the source, yet "produce the same observation"
- **Slicing:** the target code should perform the same actions over the "slicing criterion"

A first (and naive) definition of correctness

Correctness by semantic equivalence

Correctness of transformation $\mathcal{T}$ writes down as an equivalence of semantics:

$$[\![\mathcal{T}(P)]\!] = [\![P]\!]$$

Why is it naïve ?

- P and $\mathcal{T}(P)$ may not be expressed in the same language, and thus have "comparable" semantics
- e.g., if we consider compilation, $\mathcal{T}(P)$ is much lower level (machine language, with registers, etc) than P

Limitations (1)

Example: compilation of a simple imperative language

- Variables
- Syntax:

$$\begin{array}{rcl} \mathtt{e} & ::= & v \mid \mathtt{e} + \mathtt{e} \mid \ldots \\ \mathtt{i} & ::= & x := \mathtt{e}; \\ & \mid & \mathbf{if}(\mathtt{e} \leq 0)\ \mathtt{b}\ \mathbf{else}\ \mathtt{b} \\ & \mid & \mathbf{while}(\mathtt{e} \leq 0)\ \mathtt{b} \\ \mathtt{b} & ::= & \{\mathtt{i}; \ldots; \mathtt{i};\} \end{array}$$

- Variables + registers
- Program counter
- Instructions:

$$\begin{array}{rcl} i & ::= & \mathbf{add}\ \mathbf{r}_d, \mathbf{r}_{s0}, \mathbf{r}_{s1} \mid \mathbf{li}\ \mathbf{r}_d, \mathbf{r}_{s0} v \\ & \mid & \mathbf{b}\ dst \mid \mathbf{blt}\ \mathbf{r}_{s0}, \mathbf{r}_{s1}, dst \\ & \mid & \mathbf{ld}\ [s], \mathbf{r}_d \mid \mathbf{st}\ [d], \mathbf{r}_s \end{array}$$

Translation of a simple code fragment

$$\left. \begin{array}{rl} l_0 : & \mathtt{x} = 7; \\ l_1 : & \mathtt{y} = 8 + \mathtt{x}; \\ l_2 : & \ldots \end{array} \right\} \stackrel{\mathcal{T}}{\longmapsto} \left\{ \begin{array}{rlrl} 0 : & \mathbf{li}\ \mathbf{r}_0, 7 & 3 : & \mathbf{ld}\ [x], \mathbf{r}_1 \\ 1 : & \mathbf{st}\ [x], \mathbf{r}_0 & 4 : & \mathbf{add}\ \mathbf{r}_1, \mathbf{r}_0, \mathbf{r}_1 \\ 2 : & \mathbf{li}\ \mathbf{r}_0, 8 & 5 : & \mathbf{st}\ [y], \mathbf{r}_1 \end{array} \right.$$

Limitations (1)

Translation of a simple code fragment:

Translation of a simple code fragment

$$\left.\begin{array}{ll} \ell_0: & \mathrm{x} = 7; \\ \ell_1: & \mathrm{y} = 8 + \mathrm{x}; \\ \ell_2: & \ldots \end{array}\right\} \stackrel{\mathcal{T}}{\longmapsto} \left\{\begin{array}{llll} 0: & \mathbf{li}\ \mathbf{r}_0, 7 & 3: & \mathbf{ld}\ [x], \mathbf{r}_1 \\ 1: & \mathbf{st}\ [x], \mathbf{r}_0 & 4: & \mathbf{add}\ \mathbf{r}_1, \mathbf{r}_0, \mathbf{r}_1 \\ 2: & \mathbf{li}\ \mathbf{r}_0, 8 & 5: & \mathbf{st}\ [y], \mathbf{r}_1 \end{array}\right.$$

If we attempt at comparing traces point by point:

- **intermediate assembly points** $1, 3, 4$ have **no counterpart in the source**
- **registers** $\mathbf{r}_0, \mathbf{r}_1$ have **no counterpart in the source**

A semantic equality is **too tight**.

A second definition of correctness

Fix: apply an **observation function** to traces

Correctness up to observation

Correctness up to observation $\mathcal{O}$ of transformation $\mathcal{T}$ writes down as an equivalence of semantics, after applying $\mathcal{O}$ to the semantics:

$$\mathcal{O}[\![\mathcal{T}(P)]\!] = [\![P]\!]$$

Example:

- $\mathcal{O}$ ignores $1, 3, 4$ and registers
- $\mathcal{O}$ maps 0 to l_0; 2 to l_1 and 5 to l_2

Limitations (2)

Floating point computations:

- source semantics: allows **any IEEE-754 compliant rounding mode**
- target machine semantics: may choose **a specific rounding mode** (e.g., before running the program)
- all target program behavior are admissible in the source
- but **not all source behavior occur in the target program**

Execution order:

- **unspecified** in the C semantics
- **chosen** by the compiler (different compilers may make different choices)

A semantic equality is **too strong**

A third definition of correctness

Fix: weaken the previous statement to an inclusion

Correctness as an inclusion

Correctness up to observation $\mathcal{O}$ of transformation $\mathcal{T}$ writes down as an inclusion of semantics, after applying $\mathcal{O}$ to the semantics:

$$\mathcal{O}[\![\mathcal{T}(P)]\!] \subseteq [\![P]\!]$$

In both examples, only an inclusion holds

Summary

- **Correctness** relies on **comparing executions**
- This comparison is usually **not tight**:
  - **up-to observation** (abstraction)
    intricate aspects of the execution of initial and transformed programs typically do not match
  - **inclusion** (one-way only)
    transformed programs often **refine** the initial one