Context: Reasoning on Imperative Programs

Imperative programs
- Specify computational behavior
- Used in all fields of technology
- Often complex & constructed by hand

Automated analysis
- Semantic analysis (behavior of programs)
- Useful for many practical purposes

Previous approaches include:
- Hoare logic [1]; calculus for proving semantic properties of programs
- Relating semantic properties to valid FOL formulae [3]
- Denotational semantics, cf. [4]; interpreting programs as functions

Setting
- Simple imperative language (yet Turing-complete)
- Variable assignment, if-then statements, while statements
- Termination and equivalence of programs

Example: programs (1) and (2) are equivalent.

1. \[ z := x; (\text{if } x < 5 \text{ then } (x := y; y := 8) \text{ else } \text{skip}) \]  
2. \[ z := x; x := y; y := 8; \text{if } (z \geq 5) \text{ then } y := x; x := z \text{ else } \text{skip} \]

Problem Statement
- How can we provide automated reasoning on programs, resulting in practical algorithms for decidable fragments?
- Can we identify additional decidable fragments?

Approach: Translation to Description Logic

Translation of programs
- Syntactic translation into description logic \( \mathcal{ALC}(D) \)
- Prototypical \( \mathcal{ALC} \) extended with concrete domains, cf. [2]

Simulating behavior
- Behavior of programs encoded into models of translation
- Reducing reasoning to general logic reasoning

Example program \( p \):

\[ p = (x := 0; (\text{while } x < 2 \text{ do } x := x + 1); \text{skip}) \]

Its encoding \( T^p \) includes:

- \( \exists \text{value}_x, N \) \( \text{next}(\exists \text{value}_x) = 0 \cap C_p' \)
- \( \exists \text{value}_x, C_x < 2 \) \( (\neg D_x < 2 \cup C_x := x + 1; p') \cap (D_x < 2 \cup C\text{skip}) \)
- \( C_x := x + 1; p' \)

Main Result 1: Semantic Correspondence

Correspondence
- Operational semantic of programs corresponds to model-theoretic semantics of translation

Reducing reasoning
- Termination and equivalence of programs reduced to \( \mathcal{ALC}(D) \) (un)satisfiability
- Use existing \( \mathcal{ALC}(D) \) (un)satisfiability algorithms for several fragments

Theorem (soundness). For every model \( I \) of the translation \( T^p \) of \( p \), if \( I \) contains an object \( a \) corresponding to a state \( s \) in the concept \( C_p' \) for program \( p \), then the model \( I \) contains the derivation of \( p \) on \( s \) starting at object \( a \).

Theorem (completeness). For every derivation \( d \) of \( p \), there is a model \( I \) of the translation \( T^p \) of \( p \) that contains the derivation \( d \).

Main Result 2: Decidable Fragments

Undecidability
- Reasoning on programs is undecidable in general e.g., via reduction from Hilbert’s Tenth Problem
- Decidable fragments: finite numerical domains, no while-loops

Additional decidable fragment
- Based on finite partition of state space into appropriate equivalence classes
- Algorithm for equivalence of programs in this fragment based on encoding into \( \mathcal{ALC}(D) \) and its model-theoretic semantics

Example program \( q \) in the decidable fragment, computing \[ f(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{otherwise} \end{cases} \]

\[ q = x_0 := 0; \text{while } (x < 5) \text{ do } (x_0 := x_0 + x_1; x_0 := x_0 + 1); \]

\[ x_0 := 0; \text{while } (x < 4) \text{ do } (x_0 := x_0 + x_2; x := x + 1); \]

\[ (x_1 \leq 0) \text{ then } x_0 := x_0 + 3 \text{ else skip} \]

Practical Application

Implementation of automated reasoning
- In cooperation with Siemens AG
- Industrial-sized imperative rule base
- Algorithms based on translation approach

References

[6] King’s College London. Computational Logic. Faculty of Informatics Supervision: Ao.Univ.-Prof. Dr. Bernhard Gramlich

Contact: dehaan.ronald@gmail.com