Formal Verification of Domain Specific Languages

Youngjae Moon

Contents

1. What is a Domain Specific Language (DSL)?
2. Why DSLs and new programming languages?
3. Why Haskell for Formal Verification
4. Formal Verification for DSLs
5. About Exo

What is a Domain Specific Language (DSL)?

- A DSL is a computer programming langauge designed for a particular domain.
- Examples:
  - Structured Query Language (SQL)
  - HyperText Markup Language (HTML)
  - Cascading Style Sheets (CSS)
  - MATLAB
  - LaTeX
  - Verilog
  - Regular expressions
  - Yacc/Bison

Why DSLs and new programming languages?

- Based on Moore's law, computing power doubles every two years as the density of transistors in a chip doubles every two years.
- This allows business ideas that did not have commercial values in the past to be commercialised.
- Developing an existing programming language to support for new domains created make it heavier and slower.
- It becomes easier to instead develop a new lighter and faster programming language designed specifically for new domain.
- DSLs reduces the workload of a programmer to optimize its code for a particular domain
    - The built-in compiler for the DSL can do it instead.

Why Haskell for Formal Verification

- Haskell is a statically typed and strongly typed functional programming language.
  - Strongly typed
    - Prevents some types of errors.
    - Examples:
      - Null pointer exceptions
      - Type mismatches.
  - Immutability of data structures
    - Makes it easier to figure out the behavior of the code by eliminating the side effects which can interfere during the execution.
  - Functional
    - Makes it easier to predict about the behavior of the code (prevents bugs) .
    - Makes it easier to prove the correctness of the code.

Why Haskell for Formal Verification

- Haskell has a lot of libraries and frameworks for formal verification.
- Examples:
  - QuickCheck
  - Liquid Haskell

Formal Verification for DSLs

- Two research questions:
  - How are formal methods used for model checking, theorem proving, and type checking in DSLs?
  - How can formal methods ensure DSLs to fulfill their intended semantics and design goals?

Model Checking

- A formal verification technique that automatically checks whether a system satisfies a given specification.
- Useful for verifying properties such as safety, liveness, and temporal logic properties in DSLs.
- In Haskell, model checking can be performed using tools such as the QuickCheck library.

Theorem Proving

- Another formal verification technique that uses mathematical proofs to verify that a system meets its intended specifications.
- Can be used to verify properties such as functional correctness, type safety, and security in DSLs.
- In Haskell, theorem proving can be performed using tools such as the Coq proof assistant.

Type Checking

- A technique used to ensure that a program does not have type errors before it is executed.
- In Haskell, the type system is quite strong, and it can be used to ensure that programs written in DSLs are type-safe.
- Type checking can be performed using the built-in type checker in Haskell.

How to ensure DSLs to fulfill their intended semantics and design goals

- By providing rigorous verification techniques.
- These techniques can catch errors that might not be detected by traditional testing methods.
- They can help ensure that the DSL behaves correctly under all possible inputs and scenarios.

Which DSL I have chosen to test it out

- Chosen Exo
    - A DSL that helps low-level performance engineers transform very simple programs that specify what they want to compute into very complex programs that do the same thing as the specification, only much, much faster.
    - Home page: https://exo-lang.dev/
    - GitHub repository: https://github.com/exo-lang/exo

Any questions?

Thank you!