Contracts: Dafny, Viper

http://d3s.mff.cuni.cz

Pavel Parízek

Faculty of Mathematics and Physics
Charles University
Assertions

• Typically used as internal checks in the program source code

• Limitations
  ▪ Unclear semantics
    • Valid parameters (input)
    • Invariant of an algorithm
    • Correctness of the result
  ▪ Modular verification
  ▪ Inheritance
    • Consistency between parent and subclass
Dafny

- Programming language and system with built-in support for verification

  - [https://dafny.org/](https://dafny.org/)
  - [https://github.com/dafny-lang/dafny](https://github.com/dafny-lang/dafny)
  - [https://dafny-lang.github.io/dafny/](https://dafny-lang.github.io/dafny/)

- Features
  - Contracts (preconditions, postconditions, invariants)
  - Termination checking (see lecture 11)

- Usage: command-line interface, VS Code plugin
Dafny – usage & demo

• Running example from the lecture
• QuickSort
Task 1

- Write your own programs in Dafny
  - Define some contracts and verify them
Code Contracts

- Part of the .NET framework
  - Support for many programming languages


- Open source (since 2015)
  - [https://github.com/Microsoft/CodeContracts](https://github.com/Microsoft/CodeContracts)

- Main features
  - Declarative language
  - Static verification
  - Runtime checking
  - Single-threaded apps
using System.Diagnostics.Contracts;

class Test01 {
    public static int CountWhiteSpaces(string text) {
        Contract.Requires(text != null);
        Contract.Ensures(Contract.Result<int>() >= 0);
        Contract.Ensures(Contract.Result<int>() <= text.Length);

        int count = 0;
        char[] str = text.ToCharArray();

        for (int i = 0; i < str.Length; i++)
            if (char.IsWhiteSpace(str[i])) count++;

        return count;
    }
}
Basic syntax

- **Preconditions**
  - `Contract.Requires(cond);`
  - `Contract.Requires<exc>(cond);`

- **Postconditions**
  - `Contract.Ensures(cond);`
  - `Contract.EnsuresOnThrow<exc>(cond);`
  - `Contract.Result<T>()`
  - `Contract.ValueAtReturn<T>(out T t)`
  - `Contract.OldValue<T>(exp)`

- **Conditions must be side-effect free**
  - Allowed to call only methods with attribute `[Pure]`
Basic syntax

- Object invariants

  `[ContractInvariantMethod]`
  ```csharp
  private void ObjectInvariant()
  {
    Contract.Invariant(false);
  }
  ```

- Simple assertions

  ```csharp
  Contract.Assert(cond)
  ```
Quantifiers

- `Contract.ForAll<T>(IEnumerable<T> coll, Predicate<T> pred);`
- `Contract.ForAll(int fromInclusive, int toExclusive, Predicate<int> pred);`
- `public int Foo<T>(IEnumerable<T> xs) {
    Contract.Requires(Contract.ForAll(xs, x => x != null));
}
- `Contract.Exists`
- `System.Linq.Enumerable.All`
• Contracts translated into assertions

• Works like smarter testing

• Useful both for development and production

• Supports all features of Code Contracts
Static checking

- Based on abstract interpretation (lecture 9)

- Limitation: very hard to write contracts that can be proven correct by the static checker
  - False errors reported
  - Undecidable queries
  - Modular reasoning

- Hints: `Contract.Assume(cond)`
Modular reasoning

- Approach: verify just one method at a time

- Benefits: high scalability to large programs
- Limited precision (reporting spurious errors)

- Nested method calls
  1) Assert precondition of a given callee method
  2) Assume postcondition of the callee method
Advanced features

- **ContractAbbreviator**
  - Shared contracts

- **ContractArgumentValidator**
  - Legacy code (if-then-else checks)

- **Inheritance**
  - Contracts automatically reused from a parent class
  - Subclasses may add only new postconditions and object invariants
    - Goal: preserve consistency with respect to subtyping

- **Interfaces**
  - ContractClass(Type)
  - ContractClassFor(Type)
What problems you can encounter

• Inconsistencies among contracts
  ▪ Method boundaries: caller versus callee
  ▪ Consequence of modular verification

• Inconsistency between implementation and contract for a single method
  ▪ Hard to define sound and complete contracts
Support in Visual Studio

- Available through plugin

- Configuration options
  - Project -> Properties -> Code Contracts “tab”

- Does not work in recent versions (2017+)
Viper – additional examples

• Examples
  - http://viper.ethz.ch/examples/
  - Binary search
  - Graph copy
  - Encoding ADTs

• Tutorial
  - http://viper.ethz.ch/tutorial/
Task 2

- Try to use Viper
  - http://viper.ethz.ch/examples/blank-example.html
  - Write simple program (data structure, algorithm)
  - Define contracts with some access permissions
  - Run verification and fix bug reports from the tool