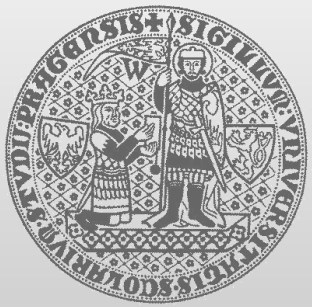
Code Contracts, Viper

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

Pavel Parízek



CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

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

Code Contracts

- Part of the .NET framework
  - Support for many programming languages
- https://www.microsoft.com/en-us/research/project/code-contracts/
- Open source (since 2015)
  - https://github.com/Microsoft/CodeContracts
- Main features
  - Declarative language
  - Static verification
  - Runtime checking
  - Single-threaded apps

Department of Distributed and Dependable Systems D3S

Example

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

- Simple assertions

```
Contract.Assert(cond)
```

Task 1

- Add contracts into `Rational.cs`
  - Use http://riseforfun.com/CodeContracts

```
class Rational {
  protected int numerator;
  protected int denominator;

  public Rational(int numerator, int denominator) {
    this.numerator = numerator;
    this.denominator = denominator;
  }

  public int toInt() {
    return numerator / denominator;
  }

  static void Main(string[] args) {
    Rational r = new Rational(10, 5);
    int i = r.toInt();
  }
}
```

Department of Distributed and Dependable Systems

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`

Department of Distributed and Dependable Systems D3S

Runtime checking

- 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)

Dependable Systems

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

Task 2

- Take the String`List.cs` example
  - It is a collection which stores strings in an array
  - Write contracts so that static checker does not report any error (violation)
  - Focus especially on the following properties:
    - `Contract.Assert(sl.Count() == 3)` in `Main`
    - Accesses to array elements are inside the bounds
    - No null dereferences occur during program execution
  - What you can also experiment with
    - Inconsistency between contracts and implementation

Support in Visual Studio

- Available through plugin
- Configuration options
  - Project -> Properties -> Code Contracts “tab”

Task 3

- Try to use Code Contracts on your programs
  - http://riseforfun.com/CodeContracts
  - Visual Studio plugin (your computer)

Task 4

- 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