Automated Generation of C# and .NET Code Contracts from VDM-SL Models

Steffen P. Diswal, **Peter W. V. Tran-Jørgensen** and Peter Gorm Larsen

AARHUS UNIVERSITY
DEPARTMENT OF ENGINEERING

14th Overture workshop, FM 2016
Limassol, Cyprus – November 7

Agenda

Agenda

Code generating a VDM specification

- Leverage model during implementation
  - Contracts describe desired system properties
  - Does the implementation satisfy the specification?
- A VDM-SL-to-Java/JML translation already exists
  - JML is a Java-based technology
  - JML tools are falling behind
- .NET Code Contracts
  - A DbC technology for .NET (several languages)
  - Library-based (unlike JML)
  - Robust, open-source technology

Code generating a VDM specification

- Leverage model during implementation
  - Contracts describe desired system properties
  - Does the implementation satisfy the specification?
- A VDM-SL-to-Java/JML translation already exists
  - JML is a Java-based technology
  - JML tools are falling behind
- .NET Code Contracts
  - A DbC technology for .NET (several languages)
  - Library-based (unlike JML)
  - Robust, open-source technology

Code generating a VDM specification

- Leverage model during implementation
  - Contracts describe desired system properties
  - Does the implementation satisfy the specification?
- A VDM-SL-to-Java/JML translation already exists
  - JML is a Java-based technology
  - JML tools are falling behind
- .NET Code Contracts
  - A DbC technology for .NET (several languages)
  - Library-based (unlike JML)
  - Robust, open-source technology

Agenda

Introduction

The translation

Performance results

Conclusion and future plans

VDM-SL-to-C# translation

- Tries to make the generated code look *natural*
  - Uses .NET Code Contracts
- Inspired by Overture's Java/JML translation
  - Addresses issues with the JML translation
- No support for traces yet
- Translation formulated as *rules*
- Visit the project on Github[1]

[1] Github: `https://github.com/SPDiswal/VdmSl-to-Cs`

VDM-SL-to-C# translation

- Tries to make the generated code look *natural*
  - Uses .NET Code Contracts
- Inspired by Overture's Java/JML translation
  - Addresses issues with the JML translation
- No support for traces yet
- Translation formulated as *rules*
- Visit the project on Github[1]

[1] Github: `https://github.com/SPDiswal/VdmSl-to-Cs`

VDM-SL-to-C# translation

- Tries to make the generated code look *natural*
  - Uses .NET Code Contracts
- Inspired by Overture's Java/JML translation
  - Addresses issues with the JML translation
- No support for traces yet
- Translation formulated as *rules*
- Visit the project on Github[1]

[1] Github: `https://github.com/SPDiswal/VdmSl-to-Cs`

VDM-SL-to-C# translation

- Tries to make the generated code look *natural*
  - Uses .NET Code Contracts
- Inspired by Overture's Java/JML translation
  - Addresses issues with the JML translation
- No support for traces yet
- Translation formulated as *rules*
- Visit the project on Github[1]

[1] Github: `https://github.com/SPDiswal/VdmSl-to-Cs`

VDM-SL-to-C# translation

- Tries to make the generated code look *natural*
  - Uses .NET Code Contracts
- Inspired by Overture's Java/JML translation
  - Addresses issues with the JML translation
- No support for traces yet
- Translation formulated as *rules*
- Visit the project on Github[1]

[1] Github: `https://github.com/SPDiswal/VdmSl-to-Cs`

Example: pre- and postconditions

```
operations
AddCard: Card ==> ()
AddCard(c) == validCards := validCards union {c}
pre c not in set validCards
post c in set validCards;
```

```
public static void AddCard(Card c) {
    Contract.Requires(c != null);
    Contract.Requires(PreAddCard(c, State));
    Contract.Ensures(
        PostAddCard(c, Contract.OldValue(State.
            Copy()), State));
    State.ValidCards.Add(c);
}
```

Pre- and postcondition functions

```
[Pure]
public static bool PreAddCard(Card c, St st) {
    Contract.Requires(c != null);
    Contract.Requires(st != null);
    return !st.ValidCards.Contains(c);
}

[Pure]
public static bool PostAddCard(Card c, St oldSt,
     St st) {
    Contract.Requires(c != null);
    Contract.Requires(oldSt != null);
    Contract.Requires(st != null);
    return st.ValidCards.Contains(c);
}
```

Example: type aliases

```
types
Pin = nat
inv p == p <= 9999;
```

- Type used to represent a pin code
- $p \in \{0, 1, \ldots, 9999\}$

Type aliases

```
public sealed class Pin : ICopyable<Pin>, IEquatable<Pin> {
    public int Value { get; }

    public Pin(int @value) { Value = @value; }

    [ContractInvariantMethod]
    private void ObjectInvariant() {
        Contract.Invariant(Value >= 0);
        Contract.Invariant(InvPin(Value));
    }

    [Pure]
    public static bool InvPin(int p) {
        Contract.Requires(p >= 0);
        return p <= 9999;
    }
    // Equals, GetHashCode etc. have been omitted.
}
```

Rule-based translation (Example)

Translating invariants

Let i be an invariant for type T, let e_i be the logical predicate of i, and let T_{inv} : T -> **bool** be the self-contained function for i in VDM-SL. Then T becomes an appropriate type T' in C# and T_{inv} becomes a member of T' as the pure method T'_{inv}. The special `ObjectInvariant` helper method of T' calls `Contract.Invariant(`T'_{inv}`(this))`. T'_{inv} evaluates and returns e_i.

Agenda

Experiments

- Exhaustive testing of FAD code obfuscation algorithm
- Performance analysis
  - **Experiment I:** No contracts checked
  - **Experiment II:** Contracts specified, but not checked
  - **Experiment III:** Contracts specified and checked

Results

Size	.NET I [ms]	.NET II [ms]	.NET III [ms]	Java I [ms]	Java II [ms]	Java III [ms]
1	1	1	1	1	2	2
2	1	1	1	2	20	22
3	1	1	1	4	245	254
4	15	15	23	22	3,103	3,212
5	190	189	295	212	37,626	38,401
6	2,273	2,279	3,610	2,498	440,716	443,523

- .NET III completes in $\approx$ 3.6 seconds
- Java III completes in $\approx$ 7.4 *minutes*
- Huge difference between Java I and II

Analysing the results

- .NET Code Contracts vs. JML
  - Slightly different set of constructs
  - Semantics of constructs sometimes different
- .NET contracts add 60% overhead
- Java II/III indicate poor OpenJML performance

Analysing the results

- .NET Code Contracts vs. JML
  - Slightly different set of constructs
  - Semantics of constructs sometimes different
- .NET contracts add 60% overhead
- Java II/III indicate poor OpenJML performance

Analysing the results

- .NET Code Contracts vs. JML
    - Slightly different set of constructs
    - Semantics of constructs sometimes different
- .NET contracts add 60% overhead
- Java II/III indicate poor OpenJML performance

Agenda

Conclusion and future plans

- VDM-SL-to-C# translation
  - Uses .NET Code Contracts
  - Fully automated
  - Command-line support
- Promising performance results
- Future plans
  - Integrate with the Overture IDE (GUI)
  - Pattern matching (native support in C# 7.0)
  - Add regression tests
  - Add support for traces

Conclusion and future plans

- VDM-SL-to-C# translation
  - Uses .NET Code Contracts
  - Fully automated
  - Command-line support
- Promising performance results
- Future plans
  - Integrate with the Overture IDE (GUI)
  - Pattern matching (native support in C# 7.0)
  - Add regression tests
  - Add support for traces

Conclusion and future plans

- VDM-SL-to-C# translation
    - Uses .NET Code Contracts
    - Fully automated
    - Command-line support
- Promising performance results
- Future plans
    - Integrate with the Overture IDE (GUI)
    - Pattern matching (native support in C# 7.0)
    - Add regression tests
    - Add support for traces