Fachhochschule Nordwestschweiz
Hochschule für Technik

# IML – Enhancements for Robust Programming

## Implementing Design-by-Contract for IML

# Inhalt

- Was? / Warum?
- Spracherweiterung / Beispiel
- Vergleich
- Checking
- Code Generation
- Demo
- Tooling
- Ausblick

# Was / Warum

- Ünterstüzung von Design by Contract
  - Mittels pre-postconditions

- Robuste Software
- Wachsender Support in anderen Sprachen/Umgebungen
  - .NET: Code contracts
  - Ada 2012
  - Eiffel, usw.

# Spracherweiterung - Beispiel

```
proc incrementPositive(inout var a: int32)
requires [ a > 0 : ParameterShouldBePositive]
ensures  [ a = old(a + 1), a > 0 : ResultShouldBePositive]
{
        a := a + 1
}
```

# Spracherweiterung – Lexikalische Syntax

```
[           (Token: LBRACKET)
]           (Token: RBRACKET)
requires    (Token: REQUIRES)
ensures     (Token: ENSURES)
```

# Spracherweiterung – Grammatikalische Syntax (1/3)

```
requires      ::= REQUIRES conditionList
ensures       ::= ENSURES conditionList

conditionList    ::= LBRACKET [condition {COMMA condition}] RBRACKET
condition        ::= expr [COLON IDENT]
```

# Spracherweiterung – Grammatikalische Syntax (2/3)

```
procDecl  ::= PROC IDENT paramList
              [GLOBAL globImpList]
              [LOCAL cpsDecl]
              [requires]
              [ensures]
              blockCmd
```

# Spracherweiterung – Grammatikalische Syntax (3/3)

```
funDecl   ::= FUN IDENT paramList
              RETURNS storeDecl
              [GLOBAL globImpList]
              [LOCAL cpsDecl]
              [requires]
              [ensures]
              blockCmd
```

# Vergleich

- Eiffel sehr ähnlich wie unser Ansatz
- Abwärtskompatibel mit Standard IML
- In Sprache integriert (Java, C#)

```
procedure Update_Person (P : in out Person)
   with Post => P.Sex = P.Sex'Old
                and P.Birth_Date = P.Birth_Date'Old;

function Inc(X: Integer) return Integer
   with Pre  => X /= Integer'Last,
        Post => Inc'Result = X'Old+1;
```

# Checking - Erweiterungen

- Context
  - `old` nur in Postconditions
  - Keine Deklaration von `old` Funktion
  - Kein rekursiver Aufruf in Condition
- Type
  - Conditions müssen boolean zurückliefern

# Checking - Erweiterungen

- Scope - Preconditions
  - Alle initialisierten Variablen(Paramenter, Imports, in/inout)
- Scope - Postconditions
  - Alle die in Precondition verfügbar sind
  - Return-Wert und out Parameter verfügbar
  - Zugriff auf `old` funktion
- Scope – In Parameter der old-Funktion
  - Wie in Precondition
- keine lokalen Variablen

# Code Generierung – Mapping IML -> JVMKlasse

```
program outtest
global
    proc loadTwice(ref i: int32, out ref a: int32, out ref b: int32)
    requires[ i > 0 : exampleCondition ]
    ensures[ a = old(i), b = old(i) ]
    {
        a init := i;
        b init := i
    };
    a: int32;
    b: int32;
    var v: int32
{
    v init := 12;
    call loadTwice(v, a init, b init);
    ! a;
    ! b
}
```

```
public final class outtest {
    public void loadTwice(int, int[], int[]);
    public outtest();
    public void outtest();
    public static void main(java.lang.String[]);
}
```

# Demo - Implementierung

- Kompletter IML-Sprachumfang
  - Funktionen / Prozeduren
  - Flowmodes / Mechmodes
  - Conditionals / Loops
  - Globals

- Komplette Erweiterung
- (fast) alle Context-Checks

# Demo !

# Tooling

- Scala
  - Parser Kombinatoren
- ASM Bytecode Library
  - Schreiben der class-Files (z.B. checksumming)

```
def factor : Parser[Expr] = positioned( literal                ^^ {case l => l}
                          | (ident ~ Init)                     ^^ {case i ~ init => StoreExpr(i, true)}
                          | (ident ~ exprList)                 ^^ {case i ~ list => FunCallExpr(i, list)}
                          | ident                              ^^ {case i => StoreExpr(i, false) }
                          | (monadicOpr ~ factor)              ^^ {case opr ~ fac => MonadicExpr(opr, fac)}
                          | (LParen ~> expr <~ RParen)         ^^ {case rest => rest} )
```

# Ausblick

- Statische Verifikation zur Compilezeit
- Labels mit String
- Invarianten

Fachhochschule Nordwestschweiz
Hochschule für Technik

# IML – Enhancements for Robust Programming

## Implementing Design-by-Contract for IML