Compiler Practical 2013

Methods with Parameters

Berthold Hoffmann (B. Gersdorf, T. Röfer)
hof@informatik.uni-bremen.de
Cartesium 2.48

Structure

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

1. Method Call and Stack
2. Extending Methods with Parameters
3. Bonus Task: Evaluation of Constant Sub-expressions

Methods with Parameters (1)

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

- Syntax analysis
  - Extend the grammar
  - *Extend MethodDeclaration* and *VarOrCall*
- Context analysis
  - For formal and actual parameters
  - Pairwise comparison of formal and actual parameter types
  - Assigning relative addresses in the stack frame

Methods with Parameters (2)

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

- Synthesis
  - Evaluate expressions of actual parameters one by one
  - Results stay on the stack
  - Call method
  - Remove parameters from stack at end of method execution

Parameters, Syntax Extension

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

memberdecl ::=

vardecl ';'

| METHOD *identifier* **[** '(' ***vardecl*** **{** ';' ***vardecl*** **}** ')' **]**

IS *methodbody*

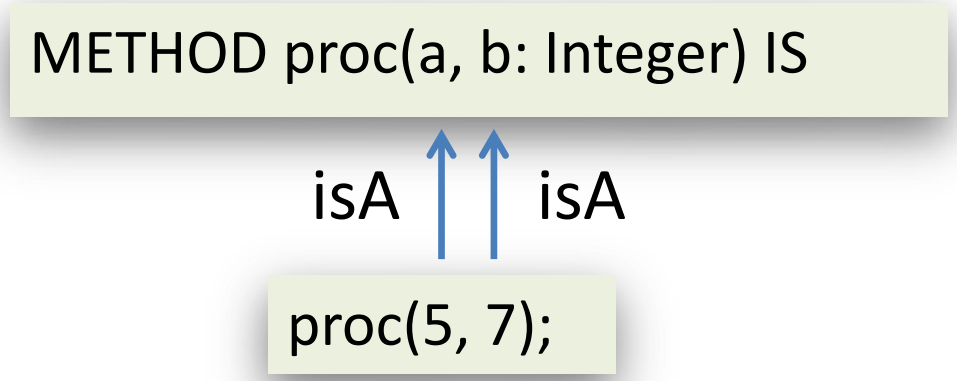
varorcall ::=

identifier **[** '(' ***disjunction*** **{** ',' ***disjunction*** **}** ')' **]**

Typ Checking

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

- Parameter passing corresponds to an assignment
  - Actual to formal parameter



- Actual parameters are references
  - *Boxing* or *Dereferencing*, if needed
- Numbers of actual and formal parameters must match
- Types of actual parameters must be compatible to those of formal parameters (*isA*)

Stack Frames without Parameters

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

```
CLASS Main
   METHOD  proc  IS
      c, d: Integer;
   BEGIN END METHOD
   METHOD main IS BEGIN
      proc; | Aufruf von 'proc'
   END METHOD
END CLASS
```

main

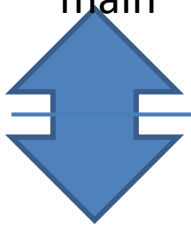
proc

Address	Method frame, call of 'proc'
R3-2	SELF
R3-1	Return address
R3	Predecessor frame (main)
R3+1	c
R3+2	d
…	…
R2	Last intermediate value

Stack Frames with Parameters

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

```
CLASS Main
   METHOD proc(a,b: Integer) IS
      c, d: Integer;
   BEGIN END METHOD
   METHOD main IS BEGIN
      proc(8, 15); | Aufruf von 'proc'
   END METHOD
END CLASS
```



Address	Method frame, call of 'proc'
R3-4	SELF
R3-3	a
R3-2	b
R3-1	Return address
R3	Predecessor frame (main)
R3+1	c
R3+2	d
…	…
R2	Last intermediate value

10%

Bonus: Constant Sub-Expressions

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

- Evaluating constant sub-expressions at compile time (runtime reduction)
  - After context analysis, before code generation
- Expressions
  - Attempt to evaluate the operands of an operator
  - If all operands of an operator are literals, evaluate the operator and replace the expression by the result (a literal)
  - Apply transformations of expressions otherwise
  - This concerns *UnaryExpression* and *BinaryExpression*

5%

Transformations: Statements

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

- This works with some statements as well:
  - IF TRUE THEN *s* ELSE *t* → *s*
  - IF FALSE THEN *s* ELSE *t* → *t*
  - WHILE TRUE DO *s* → FOREVER *s*
  - WHILE FALSE DO *s* → (nothing)

Transformations: +, -

- $- -x \rightarrow x$
- $0 + x \rightarrow x$
- $x + 0 \rightarrow x$
- $x + -y \rightarrow x - y$
- $c \pm x \pm .. \pm y \pm d \pm .. \rightarrow (c \pm d) \pm x \pm .. \pm y \pm ..$
- $0 - x \rightarrow -x$
- $x - 0 \rightarrow x$
- $x - -y \rightarrow x + y$

Transformations: Multiplikation

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

- $0 * x \rightarrow 0$
- $x * 0 \rightarrow 0$
- $1 * x \rightarrow x$
- $x * 1 \rightarrow x$
- $c * x * .. * y * d * .. \rightarrow (c * d) * x * .. * y * ..$
- $c * -x \rightarrow (-c) * x$
- $-x * c \rightarrow (-c) * x$
- $-x * y \rightarrow -(x * y)$
- $x * -y \rightarrow -(x * y)$
- $-x * -y \rightarrow x * y$

Transformationen: Division

- $0 / x$ → 0, error if $x = 0$
- $x / 1$ → x
- $c / x / .. / y / d / ..$ → $(c / d) / x / .. / y / ..$
- $x / c / .. / y / d / ..$ → $x / (c * d) / .. / y / ..$
- $-x / c$ → $x / (-c)$
- $-x / y$ → $-(x / y)$
- $x / -y$ → $-(x / y)$
- $-x / -y$ → x / y

Transformations: AND, OR, NOT

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH
Universität Bremen

- NOT NOT *x* → *x*
- FALSE AND *x* → FALSE (error in *x* is ignored!)
- *x* AND FALSE → FALSE (error in *x* is ignored!)
- TRUE AND *x* → *x*
- *x* AND TRUE → *x*
- FALSE OR *x* → *x*
- *x* OR FALSE → *x*
- TRUE OR *x* → TRUE (error in *x* is ignored!)
- *x* OR TRUE → TRUE (error in *x* is ignored!)