Compiler Practical 2013

Methods Returning Values (Functions)

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

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Universität Bremen

Structure

1. Extending Methods with Return Values
2. RETURN Statement
3. Task: Methods as Functions
4. Bonus Tasks: Error Handling

Methods as Functions

- Lexical Analysis
  - *RETURN* keyword
- Syntax Analysis
  - Extend the grammar
  - Add new class *ReturnStatement*
  - Add attribute *ReturnValue* in *MethodDeclaration*

```
METHOD factorial(n : Integer)
  : Integer IS
BEGIN
  IF n = 0 THEN
    RETURN 1;
  ELSE
    RETURN n * factorial(n - 1);
  END IF
END METHOD
```

Functions: Context, Synthesis

- Context Analysis
  - Handle return type in declaration
  - Add return type to *VarOrCall*
  - Does a function always *RETURN* a value before exiting?

- Synthesis for *RETURN [Expression]*
  - *Push return value, if present*
  - Jump to code for method exit

Functions: Syntax Extension

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Universität Bremen

memberdecl ::= *vardecl* ';'

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

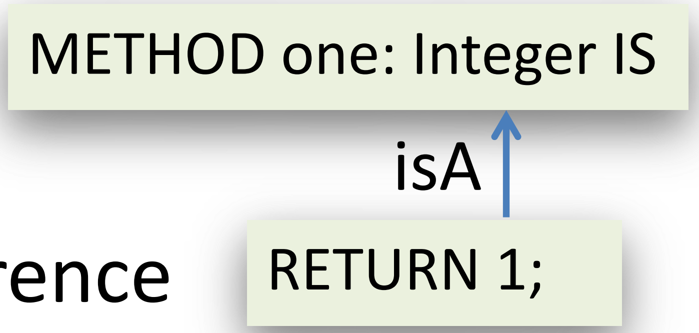
[':' *identifier*] IS *methodbody*

statement ::= ...

| RETURN [*disjunction*] ';'

Functions: Type Checking

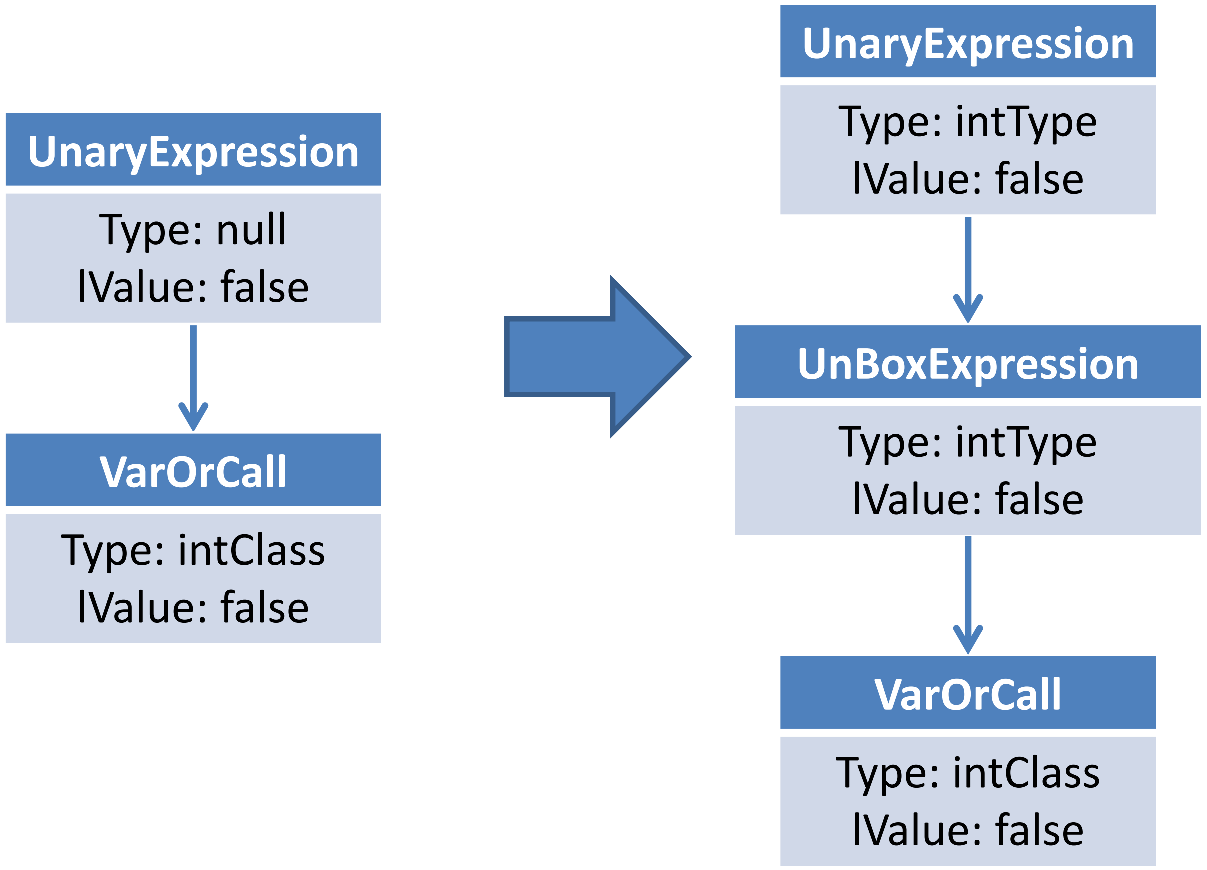
- Method call corresponds a variable access, but yields a return value



```
METHOD one: Integer IS
RETURN 1;
```

- Return value must be a reference
  - *Boxing* or *Dereferencing*, if needed
- Type of return value must be compatible with the return type of the method (*isA*)
- *Declarations.currentMethod* could be useful (HINT)…

Functions: Unboxing



Functions: RETURN statement

Will a RETURN be reached?

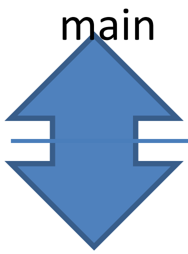
- RETURN statement reaches a RETURN
- An IF statement reaches a RETURN if its THEN branch and its ELSE branch do reach a RETURN
- A sequence S_1; … S_i; S_{i+1}; … S_n reaches RETURN if S_i reaches return (making S_{i+1} to S_n *dead code*)
- All other statements do not reach a RETURN

- Methods with a return value
  - It is an error if the body does not reach a RETURN

Task: Methods as Functions

	Address	Methode frame, call of 'proc'
R3'	R2	Return value

```
METHOD proc(a,b: Integer)
    : Integer IS
    c, d: Integer;
BEGIN
    RETURN a*b;
END METHOD
METHOD main IS BEGIN
    … 10*proc(8, 15) …
END METHOD
```



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

10%

Bonus: Several Error Messages (1)

Where may they occur?

- Lexical and Syntax Analysis
  - *LexicalAnalysis.nextSymbol()*
  - *SyntaxAnalysis.expectSymbol(...)*
  - *SyntaxAnalysis.expect*[*Resolvable*]*Ident()*
  - *SyntaxAnalysis.literal()*
- Context Analysis (optionally)
  - *Declarations.resolve*[*Type*|*VarOrMethod*]*(...)*
  - *ClassDeclaration.check(...)*
    (suppression does not make sense)

Bonus: Several Error Messages (2)

- Lexical Analysis
  - No error message, return *Symbol.Id.UNKNOWN* for an unknown sequence of characters
- Syntax Analysis
  - Report errors
  - Handle errors
    - Insert one symbol that is expected (do as if it has been read)
    - Skip to a symbol that may follow

Bonus: Several Error Messages (3)

- None of the start symbols of *literal* is found
- Valid successor symbols in LOOP are
  - . (*memberaccess*)
  - *, /, MOD (*term*)
  - +, - (*expression*)
  -) (*literal*)
  - =, #, <, <=, >, >= (*relation*)
  - ;, THEN, DO (*statement*)

```
statement     ::= READ memberaccess ';'
                |     WRITE relation';'
                |     IF relation
                      THEN statements
                      END IF
                |     WHILE relation
                      DO statements
                      END WHILE
                |     memberaccess [ ':=' relation ] ';'
relation      ::= expression
                  [ ( '=' | '#' | '<' | '>' | '<=' | '>=' ) expression ]
expression    ::= term { ( '+' | '-' ) term }
term          ::= factor { ( '*' | '/' | MOD ) factor }
factor        ::= '-' factor | memberaccess
memberaccess  ::= literal { '.' varorcall }
literal       ::= '(' relation ') '
                |     number | …
```

Bonus: Several Error Messages (4)

- **Hint**
  - The enmumeration *Symbol.Id* can be ordered, and the *ordinal*()s of symbols can be compared
  - Order symbols in *Symbol.Id* by the depth at which they appear in the grammar
- Context Analysis
  - Missing declaration: declare identifier with *universal error type* that passes every subsequent type check

5%