Compiler for Oberon-II
CS335

Sumit Bhagwani Karanveer Singh Sailesh R
Ish Dhand Ramesh

April 16, 2011

Outline

Outline

SIT Language Specifications

Source Language	Oberon-2
Target Language	MIPS Assembly level Instructions
Implementation Language	Python

Outline

Basic Types

Basic Type	Bytes Allocated	Remarks
Char	4	Is used to implement characters in ASCII
Int	4	Numeric. Integer type. Is exact.
Real	4	Numeric. Stored in Floating Point Representation. Is Inexact.
Boolean	4	Used to Store True/False Value

Complex Types

Complex Type	Remarks
Array	Homogeneous and structured Fixed length Single Dimensional data types
Strings	Treated in the .data section of the MIPS code
Pointers	Implemented pointers to Arrays (to return first element of Array)

Expressions

For the above data types, we have implemented expressions in the following form: The general form of the expression is:

$$T_0 \oplus T_1 \oplus T_2 \oplus \ldots T_n$$

where T_i's are Terms and $\oplus$'s are Add-type operations like plus or minus. A Term consists of consecutive factors like:

$$f_0 \otimes f_1 \otimes f_2 \otimes \ldots f_n$$

where f_i's are Factors and $\otimes$'s is the multiplcation operators

Short Circuit Expressions using Back-Patching

We have used the method of Short circuit evaluation of boolean expressions. With it, we can translate boolean expressions without:

1. generating code for boolean operators
2. evaluating the entire expression

Using back-patching, we could implement boolean expressions and flow of control statements in just one pass

Constants

The following Constants have been defined:

1. Integer Constant
2. Real Constant/ Floating Point Constant
3. Character Constant
4. String Constant
5. Boolean Constant

Comments

Comments in Oberon-II are of the following form:
(* Comments come here. *)

Logical Operators

We have implemented boolean operators in the case of simple Boolean Expressions:

symbol	result
OR	logical disjunction
&	logical conjunction
~	negation

Arithmetic operators

The following numerical operators have been implemented:

symbol	result
$+$	sum
$-$	difference
$*$	product
$/$	quotient
DIV	integer quotient
MOD	modulus

Keywords

DIV	MOD	OR	OF
THEN	DO	END	ELSE
ELSIF	IF	WHILE	ARRAY
RECORD	CONST	VAR	PROCEDURE
BEGIN	MODULE	INTEGER	DEFINITION
IMPORT	CHAR	BOOLEAN	IN
RETURN	LOOP	TO	INC
UNTIL	ABS	MAX	MIN
SIZE	DEC		

Outline

Typechecking

Strong Typing shall be implemented for the compiler. The Type checking shall be done Statically.
Oberon has the following rules for type-checking and coercion:

- INT op INT gives INT
- REAL op REAL gives REAL
- INT op REAL gives REAL
- REAL op INT gives REAL

Built-ins

The following Built-in function procedures have been implemented(Type Related)

Procedure	Description
ABS(x)	Absolute value (accepts and returns any numeric type)
MAX(T)	If T is a type, returns the maximum value for that type
MIN(T)	As MAX but minimum
SIZE(T)	Size of type in bytes (integral type)

Built-ins

The following Built-in function procedures have been implemented(Integer Related)

Procedure	Description
DEC(i)	Decrement integer
DEC(i, n)	Subtract n from i
INC(i)	Increment
INC(i, n)	Add n to i

Built-ins

The following Built-in function procedures have been implemented(Char Related)

Procedure	Description
CAP(x)	Capitalize Char
CHR(i)	Returns Character for the given ASCII value
ORD(x)	Returns ASCII value of a Char

The IF statement

The general form of the IF statement in oberon is:

IF B_1 THEN S_1
ELSEIF B_2 THEN S_2
...
IF B_n THEN S_n
ELSE S

Note that we have not implemented the case statement.

The While statement

The following loop statements have been implemented:

Like other languages, Oberon defines the statements for the while loop as:

```
WhileStatement =    WHILE expression DO
                          StatementSequence
                    END
```

The FOR statement

Like other languages, Oberon defines the statements for the for loop as:

ForStatement = FOR ID ASSIGN expression TO expression (optional)BY e
StatementSequence
END

Procedures

The Procedure is a statement with a name. The syntax is given by:
PROCEDURE $<$name $>$
BEGIN
...
END $<$name $>$

I/O using Modules

Sequential Input and Output is done by Modules In and Out. These Modules are the equivalent of Streams in Java.

In module out, we have implemented the following:

```
DEFINTION Out;
PROCEDURE Int(i:INTEGER);
PROCEDURE Char(i: CHAR);
PROCEDURE Real(x: REAL)
PROCEDURE String(VAR b: STRING)
END Out
```

In module in, we have implemented the following:

```
PROCEDURE ReadInt(VAR i: INTEGER)
PROCEDURE ReadReal(VAR r: REAL)
```

Optimization

The following optimizations have been implemented:

1. Constant folding
2. Strength Reduction

Outline

Features not implemented

- More than one dimensional arrays.
- Pointers to RECORDs.
- Arrays of RECORDs.
- Inheritance and polymorphism.
- Support for graphics IO.

Outline

Learning Experience

We made the following mistakes along the way and later, learned from them:

- Initially, to store information about a particular instruction of the three-address code, we used strings intead of Quads.
- We tried to break down the array into relavant instructions in the IR stage.
- We tried to implement boolean expressions without Back-patching.

References

We have used the OO2C open source *Optimizing Oberon-2 Compiler* as a cross reference for our compiler. For reference, the following source is used:

HOW TO PROGRAM A COMPUTER

Using the oo2c Oberon-2 Compiler

by Donald Daniel

2001

Revised Apr 2011

Updated for version 2 of oo2c

www.waltzballs.org

CONTENTS

Thanks