Programming Languages and Compiler Design

Intermediate-Code Generation

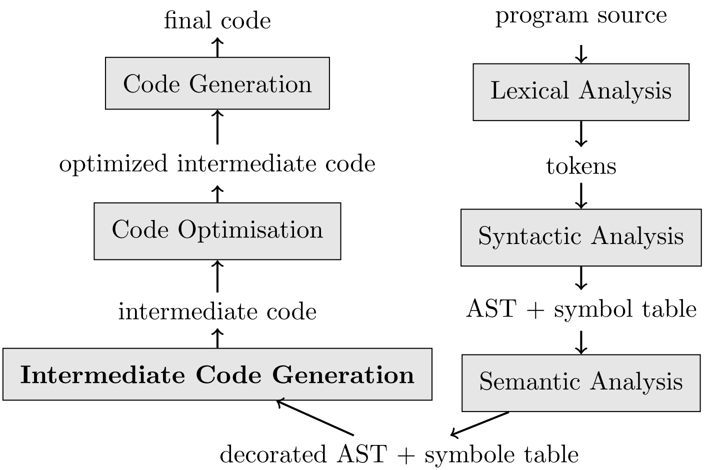
Ylìès Falcone, Jean-Claude Fernandez

Master of Sciences in Informatics at Grenoble (MoSIG)
Univ. Grenoble Alpes
(Université Joseph Fourier, Grenoble INP)

Academic Year 2015 - 2016

Intermediate Code Generation

Where are we in the compiler steps?



Intermediate Code Generation

We are interested in the **While** programming language with an abstract grammar that describes abstract syntax trees.

We consider the following problem:

Intermediate-Code Generation Problem

"Given an abstract syntax tree, how to generate 3*-address code?"*

About 3-address code

- General-purpose intermediate representation of programs.
- Many analysis can be performed on it.
- Easy to translate to assembly code.

Abstract Syntax

Syntactic Categories

Metavariables	Categories	Comments
a	AExp	arithmetical expressions
b	BExp	Boolean expressions
S	Inst	statements

Statements

$$\begin{array}{lcl} S & \rightarrow & \mathit{name} := a \mid \texttt{skip} \mid S ; S \mid \texttt{If}\ b\ \texttt{then}\ S\ \texttt{else}\ S \\ & & \mid \texttt{while}\ b\ \texttt{do}\ S\ \texttt{od} \\ \mathit{name} & \rightarrow & x \mid \mathit{tab}[i] \mid \mathit{tab}[i,j] \end{array}$$

Arithmetical expressions

$$a \rightarrow n \mid x \mid a + a \mid \mathit{tab}[i] \mid \mathit{tab}[i,j]$$

Boolean expressions

$$b \rightarrow \texttt{True} \mid \texttt{False} \mid \neg b \mid b \wedge b \mid a = a \mid a < a$$

3-address code

We first define the *syntax* of 3-address code.

We use the following functions:

- Let `Name` be the space of names that can either:
  - appear in the program, or
  - are created by function `newTemp` $: \to$ `Name`.
- Let $\mathbb{N}$ be the set of natural numbers.
  There is a partial function

$$\texttt{val} : \textbf{Aexp} \to (\texttt{Name} \cup \mathbb{N}).$$

- Let $\mathcal{L}$abel be the set of labels. They are created by function `newLabel` $: \to \mathcal{L}$abel

Meta-variables: $\mathsf{x} \in \texttt{Name}$ and $\mathsf{y}, \mathsf{z} \in (\texttt{Name} \cup \mathbb{N})$, $\mathsf{l} \in \mathcal{L}$abel.

Syntactic Categories and Grammar

Syntactic categories

Metavariables	**Categories**	**Comments**
C	Code	3-address code
op	Op$=\{+,-,*\}$	
oprel	Oprel$=\{<,>,=,\leq,\geq\}$	

Grammar

$$
\begin{array}{lcl}
\mathsf{C} & \rightarrow & \mathsf{x} := \mathsf{y}\ \mathsf{op}\ \mathsf{z} \mid \mathsf{x} := \mathsf{y} \\
 & & \mid \texttt{if}\ \mathsf{y}\ \mathsf{oprel}\ \mathsf{x}\ \texttt{goto}\ \mathsf{l} \mid \texttt{goto}\ \mathsf{l} \\
 & & \mid \mathsf{x} := \mathsf{y}[\mathsf{z}] \\
 & & \mid \mathsf{y}[\mathsf{z}] := \mathsf{x}
\end{array}
$$

Principles of Code Generation

Our objective is to define 3-address code:

Code Generation Functions

$$\begin{array}{lcl}
\texttt{GCodeAExp} & : & \mathsf{AExp} \rightarrow \mathsf{Code}^{*} \times (\texttt{Name} \cup \mathbb{N}) \\
\texttt{GCodeBExp} & : & \mathsf{BExp} \times \mathcal{L}\mathsf{abel} \times \mathcal{L}\mathsf{abel} \rightarrow \mathsf{Code}^{*} \\
\texttt{GCodeStm} & : & \mathsf{Inst} \rightarrow \mathsf{Code}^{*}
\end{array}$$

where:

- Code^{*} is the set of 3-address code sequences,
- the sequence delimiter is $\|$.

Code Generation for Arithmetical Expressions

We consider:

- 1-dimensional arrays with N elements ranging from 0 to $N-1$, and
- 2-dimensional arrays with $N \times M$ elements, where
  - N is the number of lines,
  - M is the number of columns.

The size of an element is T.

Access to an element

Consider a 2-dimensional array:

- if the array is sorted by columns:

$$Tab[i,j] \quad \text{is at} \quad N * T * j + i * T$$

- if the array is sorted by lines:

$$Tab[i,j] \quad \text{is at} \quad M * T * i + j * T$$

Code Generation for Arithmetical Expressions

GCodeAExp(x)	=		(ε,x)
GCodeAExp(n)	=		(ε,n)
GCodeAExp(tab[i])	=	Let	
			t1=newTemp, t2=newTemp
		in	$(t_1 := T{*}i \parallel$
			$t_2 := tab[t_1],t_2)$
GCodeAExp(tab[i,j])	=	Let	
			t1=newTemp(), t2=newTemp()
			t3=newTemp(), t4=newTemp()
			t5=newTemp()
		in	$(t_1 := T{*}i \parallel$
			$t_2 := N\times T \parallel$
			$t_3 := t_2\times j \parallel$
			$t_4 := t_1+t_3 \parallel$
			$t_5 := tab[t_4],t_5)$
GCodeAExp($a_1 + a_2$)	=	Let	(C_1,t_1)=GCodeAExp(a_1),
			(C_2,t_2)=GCodeAExp(a_2),
			t=newTemp
		in	$(C_1 \parallel C_2 \parallel t := t_1+t_2,t)$

Code Generation for Boolean Expressions

GCodeBExp($a_1 < a_2$, ltrue, lfalse)	$=$	Let	(C_1,t_1)$=$GCodeAExp(a_1),
			(C_2,t_2)$=$GCodeAExp(a_2),
		in	$C_1 \Vert C_2 \Vert$
			if $t_1 < t_2$
			goto ltrue $\Vert$
			goto lfalse
GCodeBExp($b_1 \wedge b_2$, ltrue, lfalse)	$=$	Let	l$=$newLabel()
		in	GCodeBExp(b_1, l, lfalse)$\Vert$
			l:$\Vert$
			GCodeBExp(b_2, ltrue, lfalse)
GCodeBExp($\neg$ b, ltrue, lfalse)	$=$		GCodeBExp(b, lfalse, ltrue)

Code Generation for Statements

Assignment and sequential composition

To each node of the abstract syntax tree, we associate code.

GCodeStm (x := a)	=	Let	(C,t)=GCodeAExp(a)
		in	C$\parallel$ x := t
GCodeStm(tab[i] := a)	=	Let	t1=newTemp,
			(C,t)=GCodeAExp(a)
		in	(t_1 := T*i $\parallel$
			C $\parallel$ tab[t_1] :=t)
GCodeStm (S_1 ; S_2)	=	Let	C_1 = GCodeStm(S_1),
			C_2 = GCodeStm(S_2)
		in	C_1 $\parallel$ C_2

Code Generation for Statements

Iterative statement

```
GCodeStm (while b do S od)  =  Let  lbegin=newLabel(),
                                     ltrue=newLabel(),
                                     lfalse=newLabel()
                                in   lbegin:||
                                     GCodeBExp(b, ltrue, lfalse)||
                                     ltrue:||
                                     GCodeStm(S)||
                                     goto lbegin||
                                     lfalse:
```

Code Generation for Statements

Conditional statement

```
GCodeStm (if b then $S_1$ else $S_2$)  =  Let  lnext=newLabel(),
                                               ltrue=newLabel(),
                                               lfalse=newLabel()
                                          in   GCodeBExp(b,ltrue,lfalse)||
                                               ltrue:
                                               GCodeStm($S_1$)||
                                               goto lnext ||
                                               lfalse:||
                                               GCodeStm($S_2$)||
                                               lnext:
```

Summary - Intermediate-Code Generation

Intermediate-Code Generation

- From `While` to 3-address code.
- 3-address code = general-purpose representation of code:
  - easy to generate,
  - suitable for optimization,
  - easy to generate to assembly code.
- Ready for optimization!