Compiladores 24-2
Back-end y generación de código

Lourdes del Carmen González Huesca
luglzhuesca@ciencias.unam.mx

Facultad de Ciencias, UNAM

6 Mayo 2024

Back-end

- Un compilador debe implementar exactamente las abstracciones incluidas en el código fuente respetando la definición y facilidades del lenguaje de programación.
- En el back-end, además de lo anterior se debe cooperar con el sistema operativo para obtener un código objeto eficiente:
  - crear y manejar un ambiente de ejecución
  - almacenar y manejar las posiciones de memoria
  - acceder a variables
  - ligar procedimientos
  - manejo y paso de parámetros
  - comunicación
- El back-end toma una representación intermedia (eg. árbol de sintaxis decorado o código de tres direcciones) y la tabla de símbolos para obtener refinamientos de la representación o representaciones y optimizar el código.
- La generación de código es la última parte del compilador.

Back-end

subfases para la producción de código

- Selección de Instrucciones
  seleccionar operaciones de máquina adecuadas, eg. operaciones aritméticas, `load` y `store word`
- Register Transfer Language
  convenciones de llamadas usando pseudo-registros, eliminar cálculos repetitivos
- Explicit Register Trasfer Language
  hacer explícitas las llamadas en los registros
- Location Transfer Language
  asignación de registros físicos del procesador mediante el análisis de vida de las variables
- Código Linearizado de Bajo Nivel

Back-end

selección de instrucciones

- Reemplazar las operaciones aritméticas del lenguaje de programación por aquellas de la máquina o procesador.
    - seleccionar las operaciones para optimizar el código:
      suma de registros y constantes, corrimiento de bits para multiplicar o dividir, comparaciones básicas, etc.

Back-end

selección de instrucciones

- Reemplazar las operaciones aritméticas del lenguaje de programación por aquellas de la máquina o procesador.
    - seleccionar las operaciones para optimizar el código:
      suma de registros y constantes, corrimiento de bits para multiplicar o dividir, comparaciones básicas, etc.
- Reemplazar los accesos a las estructuras por operaciones explícitas de acceso a la memoria para simplificar el código
    - respetar la evaluación del lenguaje de programación (perezosa, estricta, por valor, por referencia, etc.)
    - optimizar mediante evaluaciones parciales respetando la semántica

Back-end

selección de instrucciones

- Reemplazar las operaciones aritméticas del lenguaje de programación por aquellas de la máquina o procesador.
  - seleccionar las operaciones para optimizar el código:
    suma de registros y constantes, corrimiento de bits para multiplicar o dividir, comparaciones básicas, etc.
- Reemplazar los accesos a las estructuras por operaciones explícitas de acceso a la memoria para simplificar el código
  - respetar la evaluación del lenguaje de programación (perezosa, estricta, por valor, por referencia, etc.)
  - optimizar mediante evaluaciones parciales respetando la semántica

Estas traducciones o reemplazos se realizan palabra por palabra y utilizan las operaciones explícitas de la memoria (`load` y `store`).
Una dirección de memoria está dada por una expresión y un corrimiento.

Back-end

Selección de instrucciones

Consideremos que se traduce código de C en operaciones de procesador x86-64.

```
int fact(int x) {

  if (x <= 1) return 1;
  return x * fact(x-1);
}
```

```
fact(x) {
  locals:
  if (setle x $1) return 1;
  return imul x fact(addi $-1 x);
}
```

Explicación de la arquitectura x86:

https://cs.lmu.edu/~ray/notes/x86overview/

Back-end

Register Transfer Language

Transformación en un lenguaje de registros (Register Transfer Language)

- organizar el código en bloques conjunto de instrucciones secuenciales determinados por estructuras de control y saltos
- obtener una gráfica de control de flujo de la representación del código intermedio
- introducir pseudo-registros para representar los cálculos intermedios los cuales son limitados y están en relación con los registros de la máquina
- la gráfica puede estar representada por un diccionario que relaciona cada instrucción con una etiqueta para indicar las etiquetas/instrucciones siguientes

Back-end

Register Transfer Language: instrucciones

Instrucción	Acción
`mov n r -> L`	cargar la constante *n* en *r* y continuar en *L*
`load n(r1) r2 -> L`	cargar
`store r1 n(r2) -> L`	almacenar
`unop op r -> L`	operación unaria
`binop op r1 r2 -> L`	operación binaria
`ubranch br r -> L1, L2`	decisión unaria
`bbranch br r1 r2 -> L1, L2`	decisión binaria
`call r <- f(r1,...,r2) -> L`	llamar la función *f* en *r* y continuar en *L*
`goto -> L`	saltar a la etiqueta *L*

Observaciones:

- las instrucciones están relacionadas a etiquetas para linearizar el código
- la traducción de cada instrucción está etiquetada y también está ligada a una etiqueta para transferir el control al siguiente bloque
- los nombres de los registros son para identificar pseudoregistros

Back-end

Register Transfer Language: traducción

- Una expresión se traduce usando un registro destino (para almacenar el resultado) y una etiqueta para ligar el bloque siguiente.
- La función traductora RTL devuelve la etiqueta que identifica el bloque traducido

$$RLT(expr, dest, sig)$$

- La traducción se lee de abajo hacia arriba y se construye de forma backward para conocer las etiquetas destino y los registros necesarios.

Back-end

Register Transfer Language: traducción

- Una expresión se traduce usando un registro destino (para almacenar el resultado) y una etiqueta para ligar el bloque siguiente.
- La función traductora RTL devuelve la etiqueta que identifica el bloque traducido

$$RLT(expr, dest, sig)$$

- La traducción se lee de abajo hacia arriba y se construye de forma backward para conocer las etiquetas destino y los registros necesarios.

La escritura o lectura de una constante se traduce usando `mov`.

	acciones para obtener traducción
$RTL(n, r_d, L_d)$	agregar una etiqueta nueva L_n hacer L_n :`mov n rd -> Ld` regresar L_n

Back-end

Register Transfer Language: traducción

	acciones para obtener traducción
$RTL(e_1 + e_2, r_d, L_d)$	agregar una etiqueta nueva L_3
	hacer L_3: `add r2 rd -> Ld` con r_2 nuevo
	asignar a L_2 $RTL(e_2, r_2, L_3)$
	asignar a L_1 $RTL(e_1, r_d, L_2)$
	regresar L_1
$RTL(e_1 \&\& e_2, L_t, L_f)$	$RTL(e_1, RTL(e_2, L_t, L_f), L_f)$
$RTL(e_1 \|\| e_2, L_t, L_f)$	$RTL(e_1, L_t, RTL(e_2, L_t, L_f))$
$RTL(e_1 <= e_2, L_t, L_f)$	`L3: bbranch jle r2 r1 -> Lt,Lf`
	asignar a L_2 $RTL(e_2, r_2, L_3)$
	asignar a L_1 $RTL(e_1, r_1, L_2)$
	regresar L_1
$RTL(e, L_t, L_f)$	`L2: bbranch jz r -> Lf,Lt`
	asignar a L_1 $RTL(e, r, L_2)$

Back-end

Register Transfer Language: ejemplo if

Traducir el código:

```
if ( p!=0 && p -> val ==1)
    B1
else
    B2
```

Back-end

Register Transfer Language: ejemplo while

Traducir el código:

```
while (e) {
      s
  }
```

Back-end

Register Transfer Language: ejemplo while

Traducir el código:

```
while (e) {
      s
  }
```

	acciones para obtener traducción
$RTL(while(e)s, L_d)$	asignar a L_e $RTL(e, RTL(s, L), L_d)$ agregar `L : goto Le` regresar L_e

Las instrucciones o estructuras de control se traducen y sólo devuelven la etiqueta del siguiente bloque, no usan un registro para almacenar el resultado.

Back-end

Register Transfer Language: ejemplo funciones

1. cargar los pseudoregistros nuevos para los argumentos, el resultado y las variables locales de la funcón $RTL(\mathit{fun\ args\ body},\ L_f,\ L_t)$
2. construir una gráfica vacía
3. crear una etiqueta nueva para la salida de la función
4. traducir el cuerpo de la función usando *RTL* donde el resultado de la traducción es la etiqueta relacionada con la función

Back-end

Register Transfer Language: ejemplo funciones

1. cargar los pseudoregistros nuevos para los argumentos, el resultado y las variables locales de la funcón $RTL(\mathit{fun\ args\ body},\ L_f,\ L_t)$
2. construir una gráfica vacía
3. crear una etiqueta nueva para la salida de la función
4. traducir el cuerpo de la función usando *RTL* donde el resultado de la traducción es la etiqueta relacionada con la función

```
int fact(int x) {
    if (x<=1) return 1;
    return x * fact(x-1);
}
```

Back-end

Register Transfer Language: ejemplo funciones

```
int fact(int x) {
    if (x<=1) return 1;
    return x * fact(x-1);
}
```

Back-end

Explicit Register Transfer Language

- Traducción para hacer explícitas las convenciones de llamadas para el manejo y activación de los registros.
- Tiene las mismas instrucciones que RTL salvo las llamadas a funciones:

```
call t <- f(r1,r2,...,rn)    -> L
```

donde las llamadas a las funciones sólo conservan el número de argumentos dado que los propios argumentos serán almacenados en registros específicos y disponibles para la función.

Back-end

Explicit Register Transfer Language

- Traducción para hacer explícitas las convenciones de llamadas para el manejo y activación de los registros.
- Tiene las mismas instrucciones que RTL salvo las llamadas a funciones:

```
call t <- f(r1,r2,...,rn)    -> L
```

  donde las llamadas a las funciones sólo conservan el número de argumentos dado que los propios argumentos serán almacenados en registros específicos y disponibles para la función.
- Se incluyen las funciones para registros y memoria:

`alloc_frame -> L`	almacenar un frame en la pila
`delete_frame -> L`	liberar un frame en la pila
`get_param n r -> L`	acceder a un argumento en la pila
`push_param r-> L`	agregar el valor del registro en la pila
`return`	

Back-end

Explicit Register Transfer Language

De la gráfica de control de flujo derivada de la traducción anterior se agregan las instrucciones nuevas:

- al inicio de cada función se agrega un frame, se guardan los registros y los argumentos en pseudoregistros
- al fin de cada función se copia del pseudoregistro del resultado en el registro de la pila para el resultado; restaurar los registros, liberar el frame y finalmente ejecutar `return`
- por cada llamada de funciones: copiar en los pseudoregistros los argumentos, el resulado al final de la llamda y liberar memoria

Back-end

Explicit Register Transfer Language

De la gráfica de control de flujo derivada de la traducción anterior se agregan las instrucciones nuevas:

- al inicio de cada función se agrega un frame, se guardan los registros y los argumentos en pseudoregistros
- al fin de cada función se copia del pseudoregistro del resultado en el registro de la pila para el resultado; restaurar los registros, liberar el frame y finalmente ejecutar `return`
- por cada llamada de funciones: copiar en los pseudoregistros los argumentos, el resulado al final de la llamda y liberar memoria

RTL

```
#2 fact(#1)
  entry : L10
  exit  : L1
  locals:




  L10: mov #1 #6     --> L9
```

ERTL

```
fact(1)
  entry : L17

  locals: #7, #8
  L17: alloc_frame   --> L16
  L16: mov %rbx #7   --> L15
  L15: mov %r12 #8   --> L14
  L14: mov %rdi #1   --> L10
  L10: mov #1 #6     --> L9
```

Back-end

Explicit Register Transfer Language

```
fact(1)
  entry : L17
  locals: #7,#8
  L17: alloc_frame  --> L16
  L16: mov %rbx #7  --> L15
  L15: mov %r12 #8  --> L14
  L14: mov %rdi #1  --> L10
  L10: mov #1 #6    --> L9
  L9 : jle $1 #6 --> L8, L7
  L8 : mov $1 #2    --> L1
  L1 : goto         --> L22
  L22: mov #2 %rax  --> L21
  L21: mov #7 %rbx  --> L20
```

```
  L20: mov #8 %r12  --> L19
  L19: delete_frame --> L18
  L18: return
  L7 : mov #1 #5    --> L6
  L6 : add $-1 #5   --> L5
  L5 : goto         --> L13
  L13: mov #5 %rdi  --> L12
  L12: call fact(1) --> L11
  L11: mov %rax #3  --> L4
  L4 : mov #1 #4    --> L3
  L3 : mov #3 #2    --> L2
  L2 : imul #4 #2   --> L1
```

Back-end

Location Transfer Language

- Última traducción antes de obtener código linearizado.
- Se eliminan los pseudoregistros para usar los registros físicos y los espacios en la pila para preparar la ejecución.

Back-end

Location Transfer Language

- Última traducción antes de obtener código linearizado.
- Se eliminan los pseudoregistros para usar los registros físicos y los espacios en la pila para preparar la ejecución.
- Es decir: asignar los registros (register allocation)
  - analizar la vida útil de los valores
  - construcción de la gráfica de asignación eficiente de registros
  - coloración de la gráfica para la asignación en la pila

Referencias

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman.
Compilers, Principles, Techniques and Tools.
Pearson Education Inc., Second edition, 2007.

[2] Jean-Christophe Filliâtre.
Curso Compilation (inf564) école Polytechnique, Palaiseau, Francia.
http://www.enseignement.polytechnique.fr/informatique/INF564/, 2018.
Material en francés.

[3] François Pottier.
Presentaciones del curso Compilation (inf564) École Polytechnique, Palaiseau, Francia.
http://gallium.inria.fr/~fpottier/X/INF564/, 2016.
Material en francés.