Compiladores 24-2
Recap front-end y middle-end

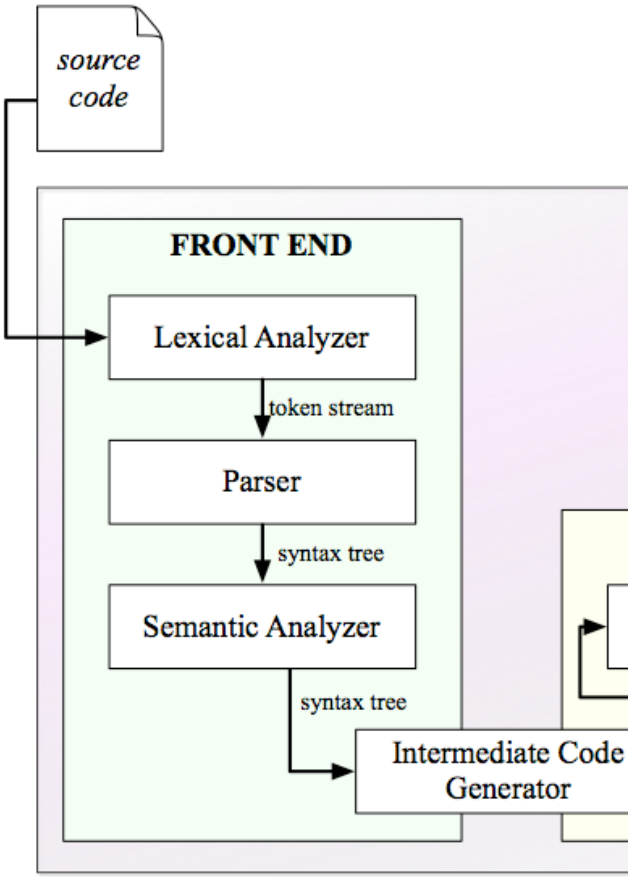
Lourdes del Carmen González Huesca
luglzhuesca@ciencias.unam.mx

Facultad de Ciencias, UNAM

17 abril 2024

Front-end

parte sintáctica



Fases de análisis del código fuente y se consideran de alto nivel:
usan herramientas y metodologías del área de Teoría de la Computación y Fundamentos de Lenguajes de Programación.

- ✓ Análisis Léxico: secuencia de tokens
- ✓ Análisis Sintáctico: parse tree
- ✓ Análisis Semántico: atributos

En este momento se tiene una representación intermedia de alto nivel.

Análisis Semántico

Type-checker

- Toda expresión del lenguaje tiene relacionado un tipo para asegurar que los valores, variables, funciones, etc. son usados correctamente, es decir proveen un contexto implícito para las operaciones al limitar las operaciones semánticamente válidas.

[1] https://en.wikipedia.org/wiki/Type_safety

Análisis Semántico

Type-checker

- Toda expresión del lenguaje tiene relacionado un tipo para asegurar que los valores, variables, funciones, etc. son usados correctamente, es decir proveen un contexto implícito para las operaciones al limitar las operaciones semánticamente válidas.
- Algunos lenguajes de programación incluyen en su definición a un sistema de tipos que son reglas sintácticas para definir la asignación de tipos a expresiones o programas.

[1] https://en.wikipedia.org/wiki/Type_safety

Análisis Semántico

Type-checker

- Toda expresión del lenguaje tiene relacionado un tipo para asegurar que los valores, variables, funciones, etc. son usados correctamente, es decir proveen un contexto implícito para las operaciones al limitar las operaciones semánticamente válidas.
- Algunos lenguajes de programación incluyen en su definición a un sistema de tipos que son reglas sintácticas para definir la asignación de tipos a expresiones o programas.
- Usando este sistema, en la implementación del compilador se puede incluir una verificación de tipos para asegurar que los programas son seguros: "*well-typed programs cannot go wrong*"– Robin Milner [1978] [1]

[1] https://en.wikipedia.org/wiki/Type_safety

Análisis Semántico

Type-checker

- Toda expresión del lenguaje tiene relacionado un tipo para asegurar que los valores, variables, funciones, etc. son usados correctamente, es decir proveen un contexto implícito para las operaciones al limitar las operaciones semánticamente válidas.
- Algunos lenguajes de programación incluyen en su definición a un sistema de tipos que son reglas sintácticas para definir la asignación de tipos a expresiones o programas.
- Usando este sistema, en la implementación del compilador se puede incluir una verificación de tipos para asegurar que los programas son seguros: "*well-typed programs cannot go wrong*"– Robin Milner [1978] [1]
- Idealmente toda expresión debe tener un tipo conocido y fijo en tiempo de compilación para así determinar el espacio de almacenamiento (direcciones, conversiones, versiones de operadores, etc.) que será usado en tiempo de ejecución.

[1] https://en.wikipedia.org/wiki/Type_safety

Sistemas de tipos

Es un método sintáctico para demostrar la ausencia de ciertos comportamientos al clasificar expresiones de acuerdo a (los tipos de) los valores que calculan:

cada expresión o programa tiene asociado un tipo

Sistemas de tipos

Es un método sintáctico para demostrar la ausencia de ciertos comportamientos al clasificar expresiones de acuerdo a (los tipos de) los valores que calculan:

cada expresión o programa tiene asociado un tipo

Esta metodología sintáctica genera una verificación semántica usando las reglas de semántica estática del lenguaje, es decir el **sistema de tipos**.

Sistemas de tipos

Es un método sintáctico para demostrar la ausencia de ciertos comportamientos al clasificar expresiones de acuerdo a (los tipos de) los valores que calculan:

cada expresión o programa tiene asociado un tipo

Esta metodología sintáctica genera una verificación semántica usando las reglas de semántica estática del lenguaje, es decir el **sistema de tipos**.

Existen varias clasificaciones de lenguajes de programación determinadas por los tipos que ofrecen y la forma en que realizan la verificación:

- Fuertemente Tipados
- Tipados estáticamente
- Tipados dinámicamente
- Tipado polimórfico
- Tipado Híbrido

Análisis Semántico

Type System

Conjunto de reglas que establecen la relación entre expresiones y tipos.

- ✓ Un programa es estáticamente correcto si satisface las reglas de tipado:
  - especificaciones del lenguaje mediante juicios de tipo

$$\Gamma \vdash e : \tau \qquad \text{el término } e \text{ tiene tipo } \tau \text{ bajo el contexto } \Gamma$$

Análisis Semántico

Type System

Conjunto de reglas que establecen la relación entre expresiones y tipos.

✓ Un programa es estáticamente correcto si satisface las reglas de tipado:

- especificaciones del lenguaje mediante juicios de tipo

$$\Gamma \vdash e : \tau \qquad \text{el término } e \text{ tiene tipo } \tau \text{ bajo el contexto } \Gamma$$

- los juicios de tipo se relacionan mediante reglas lógicas

$$\frac{\mathcal{J}_1 \qquad \mathcal{J}_2 \qquad \ldots \mathcal{J}_n}{\mathcal{J}}$$

★ Los lenguajes tienen tipos básicos o primitivos y pueden tener mecanismos para definir nuevos tipos.

Análisis Semántico

Type Inference

Dada una expresión del lenguaje (que no tenga anotaciones de tipo) reconstruir el tipo válido para dicha expresión o mostrar que no es posible asignarle un tipo.

- La inferencia se realiza en tiempo de compilación para encontrar los tipos de las expresiones en un programa.
- La verificación de tipos se puede hacer en tiempo de compilación o en tiempo de ejecución.

Análisis Semántico

Type Inference

Dada una expresión del lenguaje (que no tenga anotaciones de tipo) reconstruir el tipo válido para dicha expresión o mostrar que no es posible asignarle un tipo.

- La inferencia se realiza en tiempo de compilación para encontrar los tipos de las expresiones en un programa.
- La verificación de tipos se puede hacer en tiempo de compilación o en tiempo de ejecución.

Propiedades derivadas de un sistema de tipos:

- Progreso: todo programa se puede ejecutar y se obtiene un valor
- Preservación: siun programa está bien tipado y se puede ejecutar entonces las ejecuciones intermedias respetan el mismo tipo.

Análisis Semántico

type-checking vs type-inference

Type-checking

- ★ Proceso para asegurar que un programa obedece las reglas de compatibilidad de tipos.
  - examinar el cuerpo de una función y usar tipos declarados para verificar cualquier expresión
  - se realiza en tiempo de compilación usando la información estática del programa
  - asegura modularidad e identifica errores potenciales

Type-inference

- ★ Proceso para reconstruir un tipo de un término dado o mostrar que no tiene un tipo.
  - examinar el código sin información de tipos para obtener el tipo más general de una expresión
  - se puede realizar en compilación o en ejecución, puede ser más costoso que la verificación
  - reduce los casos expresivos, flexibilidad

Análisis Semántico

tipos

1. A partir del parse-tree obtenido en el análisis sintáctico se asignan/anotan tipos en las variables, siguiendo el alcance de las mismas.

Análisis Semántico

tipos

1. A partir del parse-tree obtenido en el análisis sintáctico se asignan/anotan tipos en las variables, siguiendo el alcance de las mismas.
2. Se agregan restricciones dependientes
    - del contexto que incluyen literales, operadores, funciones conocidas, etc.
    - del parse-tree

Análisis Semántico

tipos

1. A partir del parse-tree obtenido en el análisis sintáctico se asignan/anotan tipos en las variables, siguiendo el alcance de las mismas.
2. Se agregan restricciones dependientes
    - del contexto que incluyen literales, operadores, funciones conocidas, etc.
    - del parse-tree
3. Se resuelven restricciones mediante unificación y se determina el tipo de todas las declaraciones usando la definición del lenguaje y verificando compatibilidad de tipos.

Análisis Semántico

tipos

1. A partir del parse-tree obtenido en el análisis sintáctico se asignan/anotan tipos en las variables, siguiendo el alcance de las mismas.
2. Se agregan restricciones dependientes
    - del contexto que incluyen literales, operadores, funciones conocidas, etc.
    - del parse-tree
3. Se resuelven restricciones mediante unificación y se determina el tipo de todas las declaraciones usando la definición del lenguaje y verificando compatibilidad de tipos.

★ Interacción con la tabla de símbolos para obtener algún atributo de tipo, nivel de declaración de una variable o función/método, apuntadores, alias para nombres, número de argumentos, etc.

Análisis Semántico

expresiones de tipos

Las expresiones de tipo son representaciones textuales para los tipos de un lenguaje:

- tipos básicos
- nombres/declaraciones de tipos
- construcciones de tipo

Análisis Semántico

expresiones de tipos

Las expresiones de tipo son representaciones textuales para los tipos de un lenguaje:

- tipos básicos
- nombres/declaraciones de tipos
- construcciones de tipo

Los descriptores de tipo son estructuras que representan los tipos de las expresiones en tiempo de compilación:

```
char x char -> pointer(integer)
```

Análisis Semántico

expresiones de tipos

Las expresiones de tipo son representaciones textuales para los tipos de un lenguaje:

- tipos básicos
- nombres/declaraciones de tipos
- construcciones de tipo

Los descriptores de tipo son estructuras que representan los tipos de las expresiones en tiempo de compilación:

```
char x char -> pointer(integer)
```

```
type link = ^cell;
     cell = record
              info: integer;
              next: link;
            end;
```

Front-end y la Tabla de Símbolos

Las tablas de símbolos son estructuras de datos que se utilizan para almacenar información sobre construcciones del programa fuente, relaciona nombres léxicos y sus atributos, éstos son información relevante que es recolectada *incrementalmente* por las fases del front-end:

Front-end y la Tabla de Símbolos

Las tablas de símbolos son estructuras de datos que se utilizan para almacenar información sobre construcciones del programa fuente, relaciona nombres léxicos y sus atributos, éstos son información relevante que es recolectada *incrementalmente* por las fases del front-end:

- ✓ el lexer creará e inicializará la tabla ya sea con los lexemas o sólo con un apuntador al identificador;
- ✓ el parser podrá distiguir las diferentes declaraciones de un mismo identificador, también puede redefinir la tabla para algún símbolo si es necesario;
- ✓ el analizador semántico podrá incluir más información sobre cada identificador para completar la tabla.

Front-end y la Tabla de Símbolos

Las tablas de símbolos son estructuras de datos que se utilizan para almacenar información sobre construcciones del programa fuente, relaciona nombres léxicos y sus atributos, éstos son información relevante que es recolectada *incrementalmente* por las fases del front-end:

- ✓ el lexer creará e inicializará la tabla ya sea con los lexemas o sólo con un apuntador al identificador;
- ✓ el parser podrá distiguir las diferentes declaraciones de un mismo identificador, también puede redefinir la tabla para algún símbolo si es necesario;
- ✓ el analizador semántico podrá incluir más información sobre cada identificador para completar la tabla.

Será utilizada por las fases del back-end o las fases de síntesis para generar el código objeto.

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] Hanne Riis Nielson and Flemming Nielson.
Semantics with Applications: An Appetizer (Undergraduate Topics in Computer Science).
Springer-Verlag, Berlin, Heidelberg, 2007.

[4] Frank Pfenning.
Notas del curso (15-411) Compiler Design.
https://www.cs.cmu.edu/~fp/courses/15411-f14/, 2014.

[5] Michael Lee Scott.
Programming Language Pragmatics.
Morgan-Kauffman Publishers, Third edition, 2009.

[6] Yunlin Su and Song Y. Yan.
Principles of Compilers, A New Approach to Compilers Including the Algebraic Method.
Springer-Verlag, Berlin Heidelberg, 2011.

[7] Steve Zdancewic.
Notas del curso (CIS 341) - Compilers, Universidad de Pennsylvania, Estados Unidos.
https://www.cis.upenn.edu/~cis341/current/, 2018.