ST0244 Lenguajes de Programacion
Introducción

Andrés Sicard Ramírez

Universidad EAFIT

Semestre 2024-2

Pacto pedagógico

Como miembros de la Universidad EAFIT, nos comprometemos a actuar de manera íntegra siguiendo los más altos estándares éticos y morales.

- Respeto
- Tolerancia
- Honradez
- Compromiso

Pacto pedagógico

Página web del curso

`http://www1.eafit.edu.co/asr/cursos/st0244-lenguajes-de-programacion`

Pacto pedagógico

Página web del curso

`http://www1.eafit.edu.co/asr/cursos/st0244-lenguajes-de-programacion`

Programa de la materia

El programa de la materia está en EAFIT Interactiva.

Pacto pedagógico

Página web del curso

`http://www1.eafit.edu.co/asr/cursos/st0244-lenguajes-de-programacion`

Programa de la materia

El programa de la materia está en EAFIT Interactiva.

Conducto regular, fechas y porcentajes de las evaluaciones

La información está en la página web del curso.

Pacto pedagógico

Página web del curso

`http://www1.eafit.edu.co/asr/cursos/st0244-lenguajes-de-programacion`

Programa de la materia

El programa de la materia está en EAFIT Interactiva.

Conducto regular, fechas y porcentajes de las evaluaciones

La información está en la página web del curso.

Responsabilidad compartida

- Profesor
- Estudiantes

Pacto pedagógico

Orientaciones para el curso

- Se recomienda seis horas de trabajo por semana (dos horas por cada hora de clase).
- Las clases son presenciales.
- La evaluación a la docencia es obligatoria.
- Se recomienda revisar periódicamente los canales de comunicación institucionales (EAFIT Interactiva, correo institucional, Microsoft Teams).
- Se pueden sacar las notas de clase escritas a mano durante los exámenes parciales.
- Las prácticas no se pueden realizar de manera individual y se deben realizar máximo entre dos estudiantes.

Preliminares

Convenciones

- La numeración (capítulos, teoremas, figuras, páginas, etc) en estas diapositivas corresponde a la numeración del texto guía [Louden y Lambert 2011].
- Los ejemplos que incluyen código fuente están en el repositorio del curso.

Para usuarios de Windows

En el curso se trabajará en algunas ocasiones desde la línea de comandos. Si su sistema operativo es Windows, se sugiere instalar una versión reciente de Ubuntu vía WSL (*Windows Subsystem for Linux*).

Preliminares

Repositorio del curso

El enlace al repositorio del curso se encuentra en la página web del curso. A continuación se presentan algunos comandos de `git` para interactuar con el repositorio:

- Clonar el repositorio:

```
$ git clone https://github.com/asr/st0244-pl.git
$ cd st0244-pl
```

- Actualizar la copia local del repositorio con los últimos cambios hechos en el repositorio remoto:

```
$ cd st0244-pl
$ git fetch
$ git merge origin/main
```

(continua en la próxima diapositiva)

Preliminares

Repositorio del curso

- Revertir los cambios hechos en la copia local del repositorio:

```
$ cd st0244-pl
$ git reset --hard
```

Esquema de la presentación

Tema

Programa del curso

- Introducción
- Programación funcional
- Programación lógica
- Programación orientada a objetos
- Sintaxis
- Semántica

Tema

Algunas preguntas iniciales

Pregunta

¿Qué es un lenguaje de programación?

Algunas preguntas iniciales

Pregunta

¿Qué es un lenguaje de programación?

> *«A notation for the precise description of computer programs or algorithms. Programming languages are artificial languages, in which the syntax and semantics are strictly defined. Thus while they serve their purpose they do not permit the freedom of expression that is characteristic of a natural language.» (Definición en la séptima edición del Dictionary of Computer Science de Oxford, 2016)*

Algunas preguntas iniciales

Pregunta

¿Un lenguaje de programación universal?

Algunas preguntas iniciales

Pregunta

¿Un lenguaje de programación universal?

Una respuesta en el artículo *The Next 7000 Programming Languages* por [Chatley, Donaldson y Mycroft 2019, pág. 279]:

> *«We might hope for a single universal language which is suitable for all niches, as has been a recurring hope since Landin's time. However, the evolutionary model does not predict this. It says nothing about the existence of such a language, and past attempts to create universal languages do not add encouragement.»*

Algunas preguntas iniciales

Pregunta

¿Cuál lenguaje de programación debería usar?

Algunas preguntas iniciales

Pregunta

¿Cuál es el lenguaje de programación más popular?

Algunas preguntas iniciales

Pregunta

¿Cuál es el lenguaje de programación más popular?

- Índice TIOBE: `https://www.tiobe.com/tiobe-index/`
- GitHub: `https://github.blog/2023-11-08-the-state-of-open-source-and-ai/`

Tema

Evolución de los lenguajes de programación

La arquitectura de von Neumann

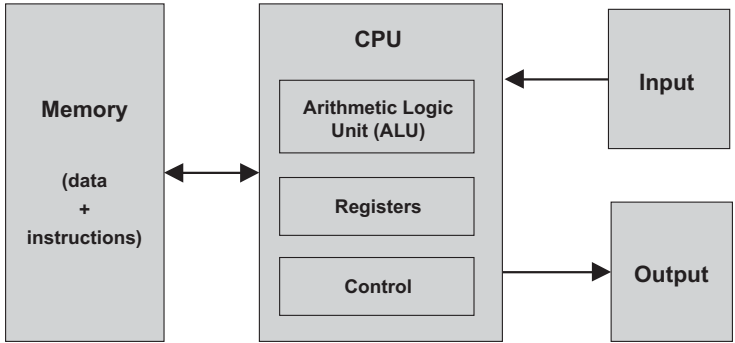


Figura 5.1 en [Nisan y Shimon 2005]

Evolución de los lenguajes de programación

Línea de tiempo

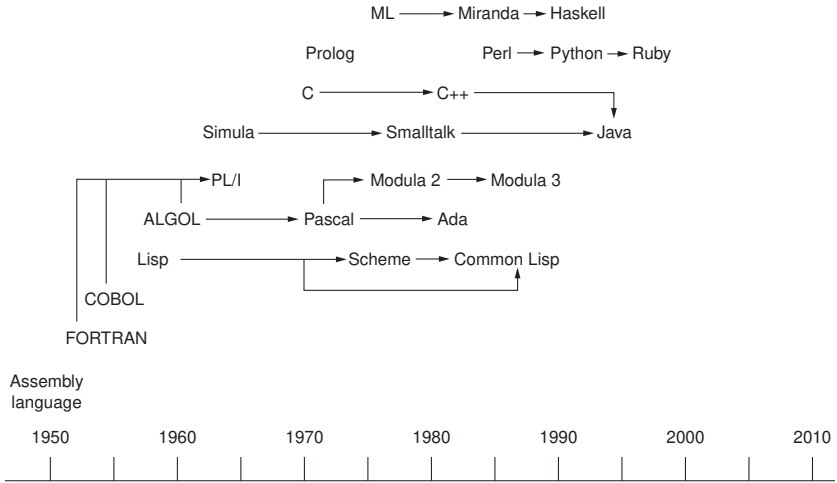


Figura 1.1

Evolución de los lenguajes de programación

Definición

El **lenguaje de máquina** (*machine language*) es el lenguaje binario que es leído, interpretado y ejecutado por la CPU.

Observación

Los lenguaje de máquina son dependientes del *hardware*.

Evolución de los lenguajes de programación

Ejemplo

Programa en lenguaje de máquina que adiciona los números 5 y 6.

```
0010001000000100
0010010000000100
0001011001000010
0011011000000011
1111000000100101
0000000000000101
0000000000000110
0000000000000000
```

Figura 1.2

Evolución de los lenguajes de programación

Definición

Un **lenguaje ensamblador** (*assembly language*) es una representación simbólica (leíble por los humanos) del lenguaje de máquina.

Observación

Los lenguajes ensambladores son dependientes del *hardware*.

Evolución de los lenguajes de programación

Ejemplo

Programa en lenguaje ensamblador que adiciona los números en las variables `first` y `second` y almacena el resultado en la variable `sum`.

```
.ORIG x3000       ; Address (in hexadecimal) of the first instruction
LD  R1, FIRST     ; Copy the number in memory location FIRST to register R1
LD  R2, SECOND    ; Copy the number in memory location SECOND to register R2
ADD R3, R2, R1    ; Add the numbers in R1 and R2 and place the sum in
                  ; register R3
ST  R3, SUM       ; Copy the number in R3 to memory location SUM
HALT              ; Halt the program
FIRST  .FILL #5   ; Location FIRST contains decimal 5
SECOND .FILL #6   ; Location SECOND contains decimal 6
SUM    .BLKW #1   ; Location SUM (contains 0 by default)
.END              ; End of program
```

Figura 1.3

Evolución de los lenguajes de programación

Ejemplo

Programa en lenguaje C que adiciona los números en las variables `first` y `second` y almacena el resultado en la variable `sum`.

```
int first = 5;
int second = 6;
sum = first + second;
```

Tema

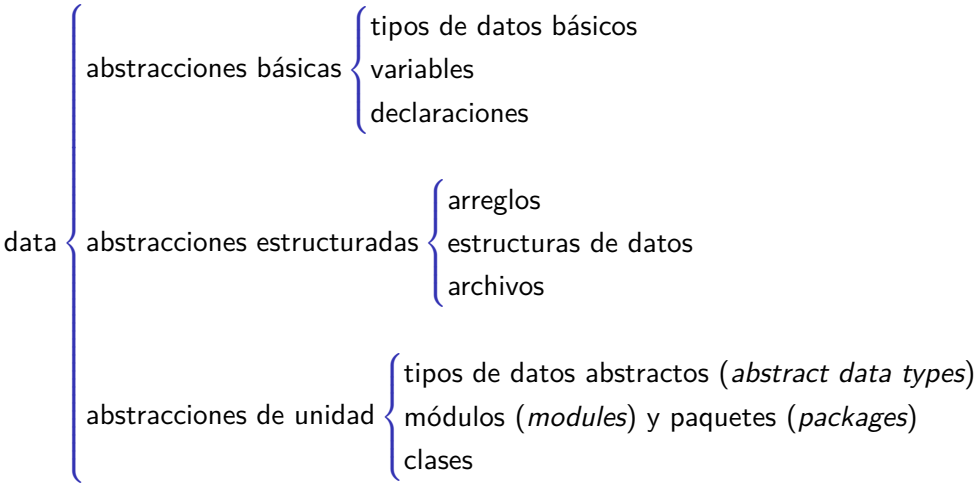
Abstracciones en lenguajes de programación

Descripción

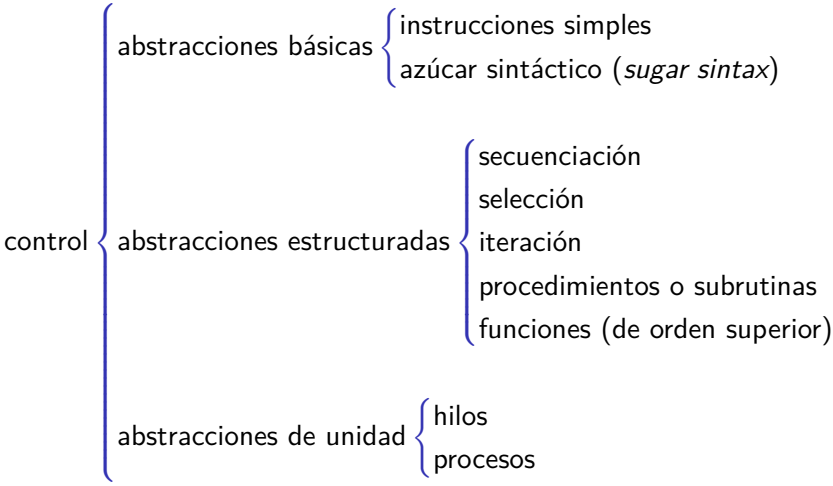
Las abstracciones en los lenguajes de programación las podemos clasificar en:

- abstracciones para la data (información) y
- abstracciones para el control.

Abstracciones en lenguajes de programación



Abstracciones en lenguajes de programación



Tema

Paradigmas de programación

Descripción

Los paradigmas de programación son:

> *«Ways of thinking about programming.» (pág. v)*
>
> *«High-level approaches for viewing computation.» (Turbark y Gifford 2008, pág. 16)*
>
> *«A way to classify programming languages based on their features.» (Wikipedia, 2024-01-23)*

Paradigmas de programación

Motivación

Evitar un sesgo cognitivo al usar una sola forma de programar:

«Si sólo tienes un martillo, todo parece un clavo».

Paradigmas de programación

Clasificación

- Declarativos (qué)
  - Funcionales (Lisp, ML, Haskell, ...)
  - Lógicos, basados en restricciones (Prolog, CLP, ...)
- Imperativos (cómo)
  - von Neumann (Fortran, C, Ada, ...)
  - Orientados a objetos (Smalltalk, C++, Java, ...)
  - *Scripting* (Perl, Python, PHP, ...)

Tema

Diseño de lenguajes de programación

Descripción

La definición de un lenguaje de programación se puede clasificar en dos partes:

- Sintaxis
- Semántica

Diseño de lenguajes de programación

Sintaxis

«The rules defining the legal sequences of symbolic elements in a language. The syntax rules define the form of the various constructs in the language, but say nothing about the meaning of these constructs. Examples of constructs are: expressions, procedures, and programs (in the case of programming languages) and terms, well-formed formulas, and sentences (in the case of logical languages).» (Definición en la séptima edición del Dictionary of Computer Science de Oxford, 2016)

Diseño de lenguajes de programación

Semántica

«That part of the definition of a language concerned with specifying the meaning or effect of a text that is constructed according to the syntax rules of the language.»
(Definición en la séptima edición del Dictionary of Computer Science de Oxford, 2016)

Tema

Translación de programas

Descripción

Los programas en los lenguajes de programación deben ser trasladados a código de máquina. La translación puede ser realizada por un(a):

- compilador,
- interpretador o
- máquina virtual.

Translación de programas

Definición

Una **plataforma** (*platform*) es una combinación especifica de *hardware* y sistema operacional.

Translación de programas: Compiladores

Definición

Un **compilador** es un programa que traduce un programa fuente a lenguaje de máquina.

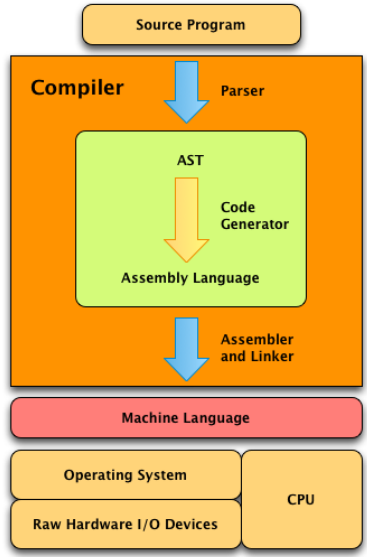
Características

- *Abstract syntax tree (AST)*: Representación interna del programa fuente.
- Si el programa fuente cambia éste debe ser recompilado.

Observación

Los compiladores son dependientes de la plataforma.

(Figura 1.12 en [Lee 2017])



Translación de programas: Compiladores

Ejemplo

C, C++, COBOL, Fortran, Haskell, Pascal, Rust y Scheme son lenguajes compilados.

Translación de programas: Interpretadores

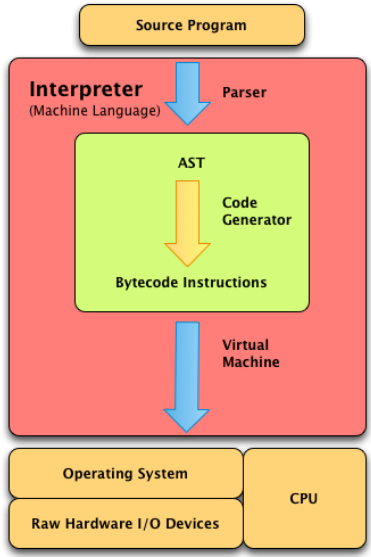
Definición

Un **interpretador** es un programa que ejecuta programas fuente.

Características

- El usuario ejecuta su programa ejecutando el interpretador.
- Ventaja: Portabilidad (el interpretador aísla el programa fuente de la plataforma).
- Desventaja: Velocidad de ejecución.

(Figura 1.13 en [Lee 2017])



Translación de programas: Interpretadores

Observación

Los interpretadores son dependientes de la plataforma.

Ejemplo

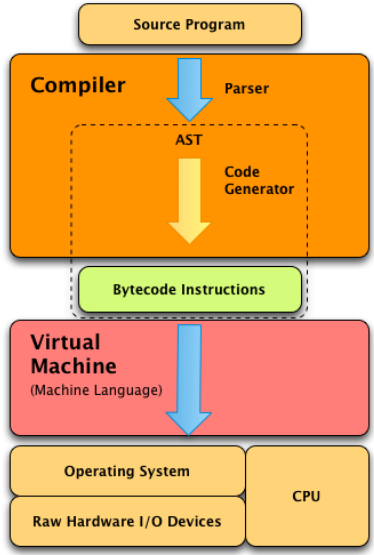
Bash, Haskell, Lisp, Prolog, Python, Ruby, y Scheme y Standard ML son lenguajes interpretados.

Translación de lenguajes: Máquinas virtuales

Definición

Una **máquina virtual** es un programa que interpreta un código intermedio, usualmente llamado **bytecode**, generado por un compilador.

(Figura 1.14 en [Lee 2017])

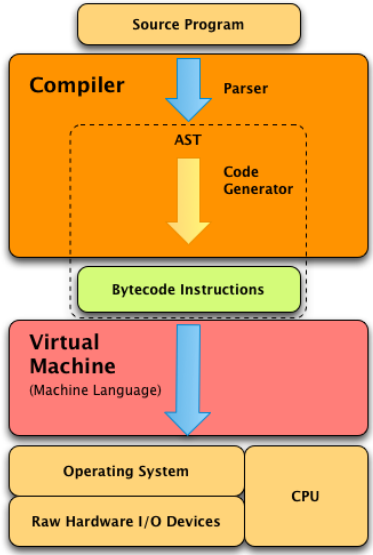


Translación de programas: Máquinas virtuales

Características

- El programa fuente es compilado a *bytecode*.
- El código *bytecode* es interpretado.
- La interpretación de *bytecode* es más rápida que la interpretación de código fuente.
- Los programas trasladados vía máquinas virtuales son más portables que los programas compilados.

(Figura 1.14 en [Lee 2017])



Translación de programas: Máquinas virtuales

Observación

Las máquinas virtuales son dependientes de la plataforma.

Observación

Las instrucciones *bytecode* son independientes de la plataforma.

Ejemplo

C#, Java, Python, Standard ML y Visual Basic.Net son lenguajes trasladados por máquinas virtuales.

Translación de programas

Pregunta

La translación de un programa fuente en lenguaje de programación depende del paradigma al cual pertenece?

Translación de programas

Pregunta

La translación de un programa fuente en lenguaje de programación depende del paradigma al cual pertenece?

¡No!

Tema

Tipos y chequeo de tipos

Tipos en los lenguajes de programación

> *«A type is an approximation of a dynamic behaviour that can be derived from the form of an expression.» (Kiselyov y Shan 2008, pág. 8)*

Tipos y chequeo de tipos

Sistemas de tipos en los lenguajes de programación

> *«A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.» (Pierce 2002, pág. 1)*

Tipos y chequeo de tipos

Chequeo de tipos: estático o dinámico*

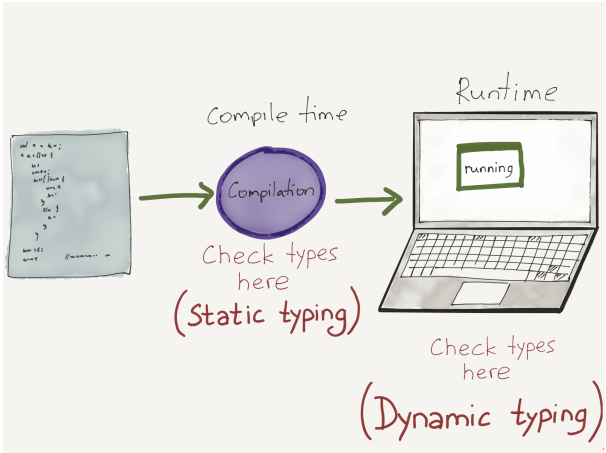


*Figura tomada de en.hexlet.io/courses/intro_to_programming/lessons/types/theory_unit.

Tipos y chequeo de tipos

Ejemplo

- Lenguajes tipeados dinámicamente: JavaScript, PHP, Python y Scheme.
- Lenguajes tipeados estáticamente: C, C++, C#, Haskell, Java, Rust y Standard ML.

Tipos y chequeo de tipos

Discusión

¿Qué es mejor, un lenguaje tipeado estáticamente o dinámicamente?

Tipos y chequeo de tipos

The static programmer says:	**The dynamic programmer says:**
"Static typing catches bugs with the compiler and keeps you out of trouble."	"Static typing only catches some bugs, and you can't trust the compiler to do your testing."
"Static languages are easier to read because they're more explicit about what the code does."	"Dynamic languages are easier to read because you write less code."
"At least I know that the code compiles."	"Just because the code compiles doesn't mean it runs."
"I trust the static typing to make sure my team writes good code."	"The compiler doesn't stop you from writing bad code."
"Debugging an unknown object is impossible."	"Debugging overly complex object hierarchies is unbearable."
"Compiler bugs happen at midmorning in my office; runtime bugs happen at midnight for my customers."	"There's no replacement for testing, and unit tests find more issues than the compiler ever could."

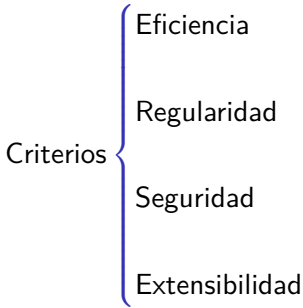
Tomado de: `www.smashingmagazine.com/2013/04/introduction-to-programming-type-systems/`.

Tema

Criterios para el diseño de lenguajes de programación

Lectura para la próxima clase: *Chapter 2. Language Design Criteria*.

Criterios para el diseño de lenguajes de programación



Criterios para el diseño de lenguajes de programación

Eficiencia

- Eficiencia del código ejecutable producido
- Eficiencia del programador
- Fiabilidad (*reability*)

  «Software engineers estimate that 90 % of their time is spent on debugging and maintenance, and only 10 % on the original coding of a program. Thus, maintainability may ultimately be the most important index of programming language efficiency.» (pág. 30)

Criterios para el diseño de lenguajes de programación

Regularidad

La regularidad se refiere a la apropiada integración de las características del lenguaje.

- Generalidad: Evitar casos especiales en los constructores del lenguaje
- Ortogonalidad: Los constructores del lenguaje se pueden combinar sin generar comportamientos diferentes
- Uniformidad: Consistencia y apariencia de los constructores del lenguaje

Criterios para el diseño de lenguajes de programación

Seguridad

Seguridad y fiabilidad son criterios relacionados.

Ejemplos

- Restricción y eliminación de apuntadores
- Declaración de variables
- Tipos y chequeo de tipos
- Seguridad semántica
  - Índice del arreglo fuera de los límites (*array index out of bounds*)
  - Recolector de basura (*garbage collector*)

Criterios para el diseño de lenguajes de programación

Extensibilidad

Un lenguaje extensible permite adicionar nuevas características.

Ejemplos

- El programador puede definir nuevos tipos de datos y operaciones
- Nuevas características del lenguaje en nuevas versiones (compatibles con versiones anteriores)
- El programador puede adicionar nueva sintaxis y semántica

Tema

Referencias


Chatley, Robert, Donaldson, Alastair y Mycroft, Alan (2019). The Next 7000 Programming Languages. En: Computing and Software Science. State of the Art and Perspectives. Ed. por Steffen, Bernhard y Woeginger, Gerhard. Vol. 10000. Lecture Notes in Computer Science. Springer, págs. 250-282. DOI: 10.1007/978-3-319-91908-9_15 (vid. págs. 17, 18).


Kiselyov, Oleg y Shan, Chung-chieh (2008). Interpreting Types as Abstract Values. Formosan Summer School on Logic, Language and Computacion (FLOLAC 2008) (vid. pág. 55).


Lee, Kent D. [2014] (2017). Foundations of Programming Languages. 2.ª ed. Undergraduate Topics in Computer Science. Springer (vid. págs. 45, 47, 49, 50).


Louden, Kenneth C. y Lambert, Kenneth A. [1993] (2011). Programming Languages. Principles and Practice. 3.ª ed. Cengage Learning (vid. pág. 8).


Nisan, Noam y Shimon, Schocken (2005). The Elements of Computing Systems. Building a Modern Computer from First Principles. MIT Press (vid. pág. 23).


Pierce, Benjamin C. (2002). Types and Programming Languages. MIT Press (vid. pág. 56).


Turbark, Franklyn y Gifford, David (2008). Design Concepts in Programming Languages. MIT Press (vid. pág. 35).