Lenguajes de Programación 2023-1
Nota de clase 4: Semántica
Introducción

Javier Enríquez Mendoza

5 de septiembre de 2022

Hasta este punto se han formalizado los conceptos relacionados a la sintaxis de los lenguajes de programación, es momento de estudiar la semántica de éstos y como podemos dar una especificación formal de ella.

1 Semántica de lenguajes de programación

La semántica de un lenguaje de programación es la encargada de darle un significado a las expresiones que pertenecen a él. Un ejemplo de semántica para un lenguaje es el comportamiento que tiene en tiempo de ejecución, es decir, como se evalúan las expresiones del lenguaje, esta semántica es principalmente en la que nos concentraremos en esta nota de clase, sin embargo no es la única como veremos mas adelante.

Es común encontrar que la semántica de un lenguaje se especifica de manera informal utilizando manuales o documentación en lenguaje natural. Si bien este tipo de especificación es útil para las personas que quieren aprender o utilizar el lenguaje, también es importante formalizar la especificación de la semántica ya que resulta de utilidad en el diseño de lenguajes pues nos permite encontrar ambigüedades y detalles imprevistos en constructores del lenguaje.

El desarrollo y uso de herramientas matemáticas de formalización semántica pueden sugerir nuevos estilos programación o mejoras a los ya existentes. Quizá el ejemplo mas importante de esto sea la influencia que el Cálculo Lambda y la teoría de categorías han tenido en los lenguajes de programación funcional.

1.1 Niveles de Semántica

Normalmente en la literatura se divide la semántica en dos niveles. En donde cada nivel se encarga de abstraer el significado de una expresión utilizando criterios distintos. Estos dos niveles son:

- **Semántica Estática**: determina cuando un programa es correcto mediante criterios sintácticos.
- **Semántica Dinámica**: determina el valor o evaluación de un programa.

1.1.1 Semántica Estática

Este nivel de semántica incluye todas las propiedades de un programa que pueden verificarse en tiempo de compilación. La definición de estas propiedades, así como qué tanta información obtiene el compilador de ellas dependen del lenguaje de programación.

En general este nivel semántico se relaciona con propiedades sobre las reglas de tipado o alcance de un lenguaje. Sin embargo en algunos lenguajes, el tipo de una expresión o el alcance de una variable solo se puede determinar hasta tiempo de ejecución, por lo que ya no pertenecerían a la semántica estática.

Definición 1.1 (Semántica estática para capturar expresiones con variables libres en `EA`)**.** En el lenguaje de expresiones aritméticas con el que hemos estado trabajando, las expresiones de la forma:

```
let x = y in x + x end
```

son sintácticamente correctas pero semánticamente incorrectas, pues la variable `y` está libre en la expresión por lo que no podría evaluarse.

Ésta es una de las propiedades de las que se encarga la semántica estática, así que daremos un conjunto de reglas para definir una semántica estática encargada de evitar estos errores. Para esto se define un juicio

$$\Delta \sim e$$

En donde Δ es un conjunto de variables en donde se guardan las variables previamente definidas y e una expresión del lenguaje en sintaxis abstracta, y se lee como e no tiene variables libres bajo el conjunto Δ de variables definidas. Entonces se definen las reglas de semántica estática como:

$$\frac{x \in \Delta}{\Delta \sim x}\ \textsf{fvv} \qquad \frac{}{\Delta \sim \texttt{num}[n]}\ \textsf{fvn} \qquad \frac{\Delta, x \sim e}{\Delta \sim x.e}\ \textsf{fva}$$

$$\frac{\Delta \sim e_1 \quad \cdots \quad \Delta \sim e_n}{\Delta \sim O(e_1, \cdots, e_n)}\ \textsf{fvo}$$

Para garantizar que una expresión e no tiene variables libres se inicia con el conjunto vacío de variables definidas y se debe probar $\varnothing \sim e$ usando las reglas anteriores.

Ejemplo 1.1 (Ejemplo del uso de la semántica estática)**.** Veamos como funcionan las reglas de semántica estática para reconocer expresiones sin variables libres con la siguiente expresión:

$$\texttt{let}(y, x.\texttt{suma}(x, x))$$

la derivación queda como sigue:

$$\dfrac{\dfrac{\texttt{error}}{\{\} \sim y} \qquad \dfrac{\dfrac{\dfrac{x \in \{x\}}{\{x\} \sim x} \qquad \dfrac{x \in \{x\}}{\{x\} \sim x}}{\{x\} \sim \texttt{suma}(x, x)}\ \text{fvo}}{\{\} \sim x.\texttt{suma}(x, x)}\ \text{fva}}{\{\} \sim \texttt{let}(y, x.\texttt{suma}(x, x))}\ \text{fvo}$$

Como una de las ramas del árbol de derivación terminó en error, se puede concluir la expresión no es correcta pues la variable y aparece libre.

Definición 1.2 (Expresión cerrada)**.** Se dice que una expresión e del lenguaje es cerrada si no tiene apariciones de variables libres, es decir, e es cerrada si y sólo si $\varnothing \sim e$.

La semántica estática se utiliza también para definir restricciones de tipos sobre las expresiones del lenguaje, esto será estudiado en el curso con mas detalle en notas posteriores una vez que tengamos un lenguaje con un sistema de tipos mas rico. En está nota nos concentraremos principalmente en el otro nivel semántico, la semántica dinámica.

2 Semántica Dinámica

La semántica dinámica de un lenguaje de programación conecta la sintaxis del lenguaje a algún modelo comunicacional capaz de evaluarlo. Existen diferentes técnicas para definir la semántica dinámica: semántica axiomática, semántica denotativa y semántica operacional. A continuación explicamos brevemente cada una de ellas:

Semántica denotativa Con este estilo de semántica nos concentramos en el efecto que genera la ejecución del programa, modelándolo con objetos matemáticos como funciones con los siguientes criterios:

- El efecto de una secuencia de instrucciones se define como la composición de las funciones que modelan el efecto de cada instrucción individualmente.
- El efecto de una instrucción se define como una función que modela los cambios que ésta generó en la memoria.

En esta semántica el objeto de interés es el resultado de la ejecución de un programa y no el proceso de ejecución. La principal ventaja de este estilo es que abstrae el comportamiento del programa fuera de la ejecución de este, por lo que se vuelve sencillo razonar sobre programas como simples objetos matemáticos, sin embargo es importante establecer una base matemática sólida para la semántica denotativa lo cual no siempre es una tarea sencilla.

Semántica axiomática También conocida como Lógica de Hoare-Floyd, en este estilo se estudia la correctud parcial de los programas, la cual es dada por un conjunto de pre-condiciones y uno de post-condiciones. Si el estado inicial (es decir, el estado antes de la ejecución del programa) cumple las pre-condiciones y si al termino de la ejecución del programa el estado resultante cumple las post-condiciones decimos que el programa es parcialmente correcto. Por ejemplo:

$$\{\mathtt{x} = \mathtt{n} \land \mathtt{y} = \mathtt{m}\} \quad \mathtt{z} := \mathtt{x}; \mathtt{x} := \mathtt{y}; \mathtt{y} := \mathtt{z} \quad \{\mathtt{x} = \mathtt{m} \land \mathtt{y} = \mathtt{n}\}$$

en donde $\{\mathtt{x} = \mathtt{n} \land \mathtt{y} = \mathtt{m}\}$ es el conjunto de pre-condiciones y $\{\mathtt{x} = \mathtt{m} \land \mathtt{y} = \mathtt{n}\}$ es el conjunto de

post-condiciones. Se puede observar que el programa es parcialmente correcto bajo estas condiciones. Esta semántica provee un sistema lógico para probar ciertas propiedades de los programas individuales de manera sencilla. En la mayoría de los casos ha sido posible de automatizar este razonamiento, permitiendo que sea una computadora la que pruebe estas propiedades.

Semántica Operacional En este estilo de semántica el significado de un programa será el *cómo* se ejecuta en una máquina. Se estudia el proceso de ejecución de un programa y no el resultado que produce. Para esto se modela el computo producido en un sistema de transición o *máquina abstracta* en donde los estados son expresiones del lenguaje y las transiciones se definen con el proceso de ejecución de estas expresiones. Esta semántica es útil cuando se quiere estudiar propiedades de la ejecución en si del programa.

En este curso nos concentraremos únicamente en el estudio de la semántica operacional de un lenguaje. Si se quiere profundizar mas en el estudio de los otros estilos puede consultarse 4 y llevar como curso optativo Semántica y Verificación.

2.1 Semántica Operacional

Como vimos anteriormente la semántica operacional define la ejecución de un programa a partir de un sistema de transición que modela los cómputos realizados en el proceso de ejecución. Se tienen dos enfoques para definir la semántica operacional de un lenguaje:

- **Semántica Estructural**: se le conoce también con los nombres de semántica de paso pequeño o de transición. Describe paso a paso la ejecución mostrando los cómputos que genera cada paso individualmente.
- **Semántica Natural**: también conocida como semántica de paso grande. En este enfoque el objetivo es describir de forma general cómo se obtiene el resultado de una ejecución.

Para este curso se usará en general un enfoque de semántica de paso pequeño, sin embargo, a continuación se mostrarán las definiciones de ambos enfoques para el lenguaje EA . Para esto se define un sistema de transición a partir de las expresiones del lenguaje como sigue.

Definición 2.1 (Sistema de transición para semántica operacional de EA)**.** Se define la semántica operacional del lenguaje de expresiones aritméticas utilizando el sistema de transición siguiente:

Conjunto de estados $S = \{a \mid a \ \mathsf{asa}\}$, es decir, los estados del sistema son las expresiones bien formadas del lenguaje en sintaxis abstracta. Esta definición corresponde a la regla de inferencia:

$$\frac{a \ \mathsf{asa}}{a \ \texttt{estado}} \ \mathsf{state}$$

Estados Iniciales $I = \{a \mid a \ \mathsf{asa},\ \varnothing \sim a\}$, los estados iniciales son todas las expresiones cerradas del lenguaje, es decir, expresiones sin variables libres. Correspondiente a la regla:

$$\frac{a \ \mathsf{asa} \qquad \varnothing \sim a}{a \ \texttt{inicial}} \ \mathsf{init}$$

Estados Finales se definen como las expresiones que representan a los posibles resultados finales de un proceso de evaluación. Para poder modelarlos definimos una categoría de valores los cuales son un subconjunto de expresiones que ya se han terminado de evaluar y no pueden

reducirse más, con el juicio v `valor`. Para el caso de AE el único valor son los números, formalmente definido con la regla:

$$\frac{}{\texttt{num}[n]\ \texttt{valor}}\ \text{vnum}$$

Entonces se define el conjunto de estados finales $F = \{a \mid a\ \texttt{valor}\}$, correspondiente a la regla:

$$\frac{a\ \texttt{valor}}{a\ \texttt{final}}\ \text{fin}$$

Transiciones la definición de las transiciones se da de acuerdo al enfoque de semántica operacional que se utiliza por lo que se darán una para cada enfoque en secciones siguientes.

Formalmente para la definición de cada enfoque de semántica es necesario definir un sistema de transición distinto, en este caso la única diferencia entre los enfoques es la función de transición del sistema, por lo que definimos de forma general el conjunto de estados y a continuación se da la definición de cada función de transición según el enfoque utilizado.

2.1.1 Semántica estructural o de paso pequeño

En el caso de la semántica de paso pequeño el sistema de transición abstrae paso a paso la evaluación de la expresión mediante la función de transición.

Recordemos que los estados del sistema son expresiones del lenguaje, entonces definimos la relación $e_1 \to e_2$ como la transición del estado e_1 al estado e_2 si y sólo si en un paso de evaluación se puede reducir e_1 a e_2. En esta relación e_1 es llamado *redex* mientras que e_2 es el *reducto*.

Observación. No es posible definir transiciones en donde algún estado final sea *redex*. Es decir, no hay transiciones desde estados finales.

Definición 2.2 (Estado bloqueado)**.** Un estado s está bloqueado si no existe otro estado s' tal que $s \to s'$ y lo denotamos como $s \not\to$.

Con la definición de la relación $\to$ se define la semántica operacional de paso pequeño como sigue.

Definición 2.3 (Función de transición para semántica de paso pequeño)**.** Se da la definición de la función de transición para completar la definición de la semántica operacional de paso pequeño con el sistema de transición 2.1 mediante las siguientes reglas de inferencia:

Suma

$$\frac{}{\texttt{suma}(\texttt{num}[n], \texttt{num}[m]) \to \texttt{num}[n +_{\mathbb{N}} m]}\ \text{sumaf}$$

$$\frac{a_1 \to a_1'}{\texttt{suma}(a_1, a_2) \to \texttt{suma}(a_1', a_2)}\ \text{suma1} \qquad \frac{a_2 \to a_2'}{\texttt{suma}(\texttt{num}[n], a_2) \to \texttt{suma}(\texttt{num}[n], a_2')}\ \text{suma2}$$

Producto

$$\frac{}{\texttt{prod}(\texttt{num}[n], \texttt{num}[m]) \to \texttt{num}[n \times_{\mathbb{N}} m]}\ \textsf{prodf}$$

$$\frac{a_1 \to a_1'}{\texttt{prod}(a_1, a_2) \to \texttt{prod}(a_1', a_2)}\ \textsf{prod1} \qquad \frac{a_2 \to a_2'}{\texttt{prod}(\texttt{num}[n], a_2) \to \texttt{prod}(\texttt{num}[n], a_2')}\ \textsf{prod2}$$

Asignaciones locales

$$\frac{v\ \texttt{valor}}{\texttt{let}(v, x.a_2) \to a_2[x := v]}\ \textsf{letf} \qquad \frac{a_1 \to a_1'}{\texttt{let}(a_1, x.a_2) \to \texttt{let}(a_1', x.a_2)}\ \textsf{let1}$$

Hay que observar que en las transiciones están modelando paso a paso la reducción de las expresiones hasta encontrar un valor.

La forma en la que se definen la semántica operacional en la definición anterior 2.3 es la manera en la que estaremos definiendo la semántica para los lenguajes que estudiemos en el curso para poder estudiar las propiedades de la ejecución de los programas mediante la relación $\to$.

Ejemplo 2.1 (Uso de la semántica operacional de paso pequeño)**.** Para ejemplificar el funcionamiento de la semántica estructural se evalúa la expresión:

```
let x = 3 + (2 * 3) in (x + 2) * 4
```

$$\begin{aligned}
& \texttt{let}(\texttt{suma}(3, \texttt{prod}(2,3)), x.\texttt{prod}(\texttt{suma}(x,2),4)) \\
\to\ & \texttt{let}(\texttt{suma}(3,6), x.\texttt{prod}(\texttt{suma}(x,2),4)) \\
\to\ & \texttt{let}(9, x.\texttt{prod}(\texttt{suma}(x,2),4)) \\
\to\ & \texttt{prod}(\texttt{suma}(x,2),4)[x := 9] \\
=\ & \texttt{prod}(\texttt{suma}(9,2),4) \\
\to\ & \texttt{prod}(11,4) \\
\to\ & 44
\end{aligned}$$

Es importante observar que se usa una mezcla de sintaxis concreta y abstracta para simplificar la notación.

2.1.2 La relación de transición

Dada la relación de transición $\to$ con la que se definió la semántica estructural del lenguaje, se definen inductivamente las siguientes relaciones derivadas que serán de suma importancia para el estudio de la semántica del lenguaje:

Definición 2.4 (Cerradura transitiva y reflexiva)**.** La cerradura reflexiva y transitiva se denota como $\to^*$ y se define con la siguientes reglas:

$$\frac{}{s \to^* s} \qquad \frac{s_1 \to s_2 \quad s_2 \to^* s_3}{s_1 \to^* s_3}$$

Intuitivamente la relación $s_1 \to^* s_2$ modela que es posible llegar desde s_1 hasta s_2 en un número finito de pasos de la relación de transición $\to$, posiblemente 0.

Definición 2.5 (Cerradura transitiva)**.** La cerradura transitiva se denota como $\to^+$ y se define con la siguientes reglas:

$$\frac{s_1 \to s_2}{s_1 \to^+ s_2} \qquad \frac{s_1 \to s_2 \quad s_2 \to^+ s_3}{s_1 \to^+ s_3}$$

Intuitivamente la relación $s_1 \to^+ s_2$ modela que es posible llegar desde s_1 hasta s_2 en un número finito de pasos estrictamente mayor a cero de la relación de transición $\to$. Es decir, se llega de s_1 a s_2 en al menos un paso.

Definición 2.6 (Iteración en n pasos)**.** la iteración en n pasos se denota como $\to^n$ con $n \in \mathbb{N}$ y se define con la siguientes reglas:

$$\frac{}{s \to^0 s} \qquad \frac{s_1 \to s_2 \quad s_2 \to^n s_3}{s_1 \to^{n+1} s_3}$$

Intuitivamente la relación $s_1 \to^n s_2$ modela que es posible llegar desde s_1 hasta s_2 en exactamente n pasos de la relación de transición $\to$.

2.1.3 Semántica natural o de paso grande

Durante este curso estaremos estudiando la semántica de los lenguajes de programación a través de la semántica operacional de paso pequeño. Pero a modo de ejemplo se define también una semántica operacional de paso grande para el lenguaje EA , en esta semántica se definen de forma general los cómputos realizados por la ejecución de un programa utilizando una relación de transición similar a la definida con anterioridad.

La relación de transición usada es $e \Downarrow v$ en donde se relaciona una expresión e con un valor v, a diferencia de la relación $\to$ en donde se relacionaban dos expresiones. $\Downarrow$ se lee como *se evalúa a*.

Definición 2.7 (La relación $\Downarrow$ para EA)**.** Se define la transición sobre los estados definidos en 2.1 para la semántica operacional de paso grande de EA mediante las siguientes reglas de inferencia:

Números Para este enfoque si es necesario definir una regla de transición para los valores del lenguaje, pues es necesaria para el correcto funcionamiento del resto de las reglas.

$$\frac{}{\texttt{num}[n] \Downarrow \texttt{num}[n]}\ \textsf{bsnum}$$

Suma

$$\frac{e_1 \Downarrow \texttt{num}[n] \quad e_2 \Downarrow \texttt{num}[m]}{\texttt{suma}(e_1, e_2) \Downarrow \texttt{num}[n +_{\mathbb{N}} m]}\ \textsf{bssum}$$

Producto

$$\frac{e_1 \Downarrow \texttt{num}[n] \quad e_2 \Downarrow \texttt{num}[m]}{\texttt{prod}(e_1, e_2) \Downarrow \texttt{num}[n \times_{\mathbb{N}} m]}\ \textsf{bsprod}$$

Asignaciones locales

$$\frac{e_1 \Downarrow v_1 \quad e_2[x := v_1] \Downarrow v_2}{\texttt{let}(e_1, x.e_2) \Downarrow v_2}\ \textsf{bslet}$$

Ejemplo 2.2 (Ejemplo de uso de semántica de paso grande)**.** Para ejemplificar el funcionamiento de la semántica natural se evalúa la expresión:

```
let x = 3 + (2 * 3) in (x + 2) * 4
```

$$\dfrac{\dfrac{}{3 \Downarrow 3} \quad \dfrac{\dfrac{}{2 \Downarrow 2} \quad \dfrac{}{3 \Downarrow 3}}{\texttt{prod}(2,3) \Downarrow 6}}{\texttt{suma}(3, \texttt{prod}(2,3)) \Downarrow 9} \qquad \dfrac{\dfrac{\dfrac{}{9 \Downarrow 9} \quad \dfrac{}{2 \Downarrow 2}}{\texttt{suma}(9,2) \Downarrow 11} \quad \dfrac{}{4 \Downarrow 4}}{\texttt{prod}(\texttt{suma}(x,2),4)[x := 9] \Downarrow 44}$$

$$\dfrac{\texttt{suma}(3, \texttt{prod}(2,3)) \Downarrow 9 \qquad \texttt{prod}(\texttt{suma}(x,2),4)[x := 9] \Downarrow 44}{\texttt{let}(\texttt{suma}(3, \texttt{prod}(2,3)), x.\texttt{prod}(\texttt{suma}(x,2),4)) \Downarrow 44}$$

Como se ve en los ejemplos 2.1 y 2.2 el resultado de la evaluación usando ambos enfoques de la semántica operacional es el mismo, esto se puede generalizar en el siguiente teorema:

Teorema 2.8 (Equivalencia entre semántica estructural y natural)**.** Para cualquier expresión e del lenguaje EA se cumple:

$$e \to^* v \text{ si y sólo si } e \Downarrow v$$

Es decir, la semántica de paso pequeño es equivalente a la semántica de paso grande.

Demostración. La demostración de este teorema es por inducción sobre los constructores del lenguaje y queda fuera del alcance de este curso, pero es importante que el alumno se convenza de la equivalencia entre las semánticas. □

3 Propiedades de la semántica dinámica

La relación de transición $\to$ con la que se definió la semántica de paso pequeño cumple las siguientes propiedades:

Proposición 3.1 (Bloqueo de valores)**.** Si v `valor` entonces $v \not\to$, es decir v está bloqueado.

Demostración. Análisis de casos sobre el juicio v `valor`.
En este caso el único valor del lenguaje es $\texttt{num}[n]$ como se ve en 2.1 por lo que baste verificar que $\texttt{num}[n] \not\to$, esto se cumple pues en la definición de la función de transición 2.3 no se define ninguna regla sobre el constructor `num` por lo que no tiene transiciones definidas, de ahí entonces $\texttt{num}[n] \not\to$. □

Proposición 3.2 (Determinismo de la relación $\to$)**.** Si $e \to e_1$ y $e \to e_2$ entonces $e_1 = e_2$.

Demostración. Inducción sobre la relación $e \to e_1$

Caso Base: sumaf

Esta regla da la única transición definida para el estado $\texttt{suma}(\texttt{num}[n], \texttt{num}[m])$, por lo que es determinista.

Casos Base: prodf y letf son análogos al anterior.

Paso inductivo: suma1

Hipótesis de Inducción: $a_1 \to a_2$ es determinista.

Veamos que en la transición $\texttt{suma}(a_1, a_2) \to \texttt{suma}(a_1', a_2)$ la única reducción aplicada a la expresión se da por el cambio de a_1 hacia a_1'. Por lo que basta ver que la transición $a_1 \to a_2$ es determinista, que es exactamente la Hipótesis de Inducción.

Pasos inductivos: El razonamiento sobre el resto de las reglas es análogo al presentado en el caso anterior.

$\therefore$ la relación $\rightarrow$ es determinista. □

Corolario 3.3 (Determinismo de la relación $\rightarrow^*$)**.** Si $e \rightarrow^* e_1$ y $e \rightarrow^* e_2$, con $e_1 \not\rightarrow$ y $e_2 \not\rightarrow$ entonces $e_1 = e_2$.

Observación. Es importante aclarar que las propiedades anteriores se cumplen en el caso particular del de la semántica dinámica para el lenguaje EA , en los casos de semánticas para otros lenguajes no necesariamente se van a cumplir estas propiedades y es importante verificar cuales de se conservan y cuales no. Mas adelante en el curso trabajaremos con lenguajes en donde las propiedades de la semántica dinámica cambiarán.

3.1 Función de evaluación

Para evaluar las expresiones del lenguaje se define una función de evaluación a partir de la semántica dinámica, como sigue.

Definición 3.4 (Función de evaluación)**.** Se define la función `eval` en términos de la semántica dinámica del lenguaje como sigue:

$$\texttt{eval}(e) = e_f \text{ si y sólo si } e \rightarrow^* e_f \text{ y } e_f \not\rightarrow$$

Gracias al corolario 3.3 sabemos que la función `eval` es determinista, sin embargo no sabemos si se trata de total, es decir, no sabemos si para toda expresión e la expresión $\texttt{eval}(e)$ exista pues podría ser que el proceso de evaluación no termine. En los lenguajes de programación reales no se puede garantizar la terminación sin embargo en el caso de EA si.

Proposición 3.5 (Terminación)**.** Para cada expresión e del lenguaje existe una expresión e_f tal que $e \rightarrow^* e_f$ y $e_f \not\rightarrow$, es decir, para cada expresión e del lenguaje, la función $\texttt{eval}(e)$ termina.

Demostración. Esta demostración es compleja debido al constructor `let` y el desarrollo de la misma queda fuera del alcance del curso. □

Referencias

[1] Miranda Perea F., González Huesca L., Nota de Clase del curso de Lenguajes de Programación, Facultad de Ciencias UNAM, Semestre 2021-1.

[2] Keller G., O'Connor-Davis L., Class Notes from the course Concepts of programming language design, Department of Information and Computing Sciences, Utrecht University, The Netherlands, Fall 2020.

[3] Ramírez Pulido K., Soto Romero M., Enríquez Mendoza J., Nota de Clase del curso de Lenguajes de Programación, Facultad de Ciencias UNAM, Semestre 2021-2

[4] Nielson F., Nielson F., Semantics with Applications: An Appetizer, Springer Publishing, 2007

[5] Harper R., Practical Foundations for Programming Languages. Working draft, 2010.

[6] Mitchell J., Foundations for Programming Languages. MIT Press 1996.

[7] Krishnamurthi S., Programming Languages Application and Interpretation; Version 26.04.2007.