Programación Lógica y Funcional

Programación Funcional

Dr. Alejandro Guerra-Hernández

Instituto de Investigaciones en Inteligencia Artificial
Universidad Veracruzana
Campus Sur, Calle Paseo Lote II, Sección Segunda No 112,
Nuevo Xalapa, Xalapa, Ver., México 91097

mailto:aguerra@uv.mx
https://www.uv.mx/personal/aguerra/plf

Maestría en Inteligencia Artificial 2024

Justificación

- Modelo preferido de computabilidad máquina de Turing $\neq$ algoritmos en lenguajes de alto nivel.
- Modelo AL [5], un lenguaje tipificado para expresar de manera precisa problemas computacionales.
- Completo, en el sentido de que cualquier computación intuitiva, pueda ser especificada por medios finitos.
- Puente con el cálculo-λ, una teoría de las funciones computables

Universidad Veracruzana

Valores, Expresiones y Transformaciones

- AL es un lenguaje orientado a expresiones.
- Cómputo: Algoritmos = Expresiones $\rightarrow$ Valores.
- Las computaciones se definen mediante un conjunto de reglas de transformación, aplicadas sistemáticamente hasta que ya no es posible ninguna transformación.
- La última transformación regresa el valor computado.
- Los valores tienen una interpretación más general: no son necesariamente atómicos.

Universidad Veracruzana

Notación

- Denotemos a las expresiones e ó e_i para $i \in \{0, \ldots, n\}$.
- Una computación es una secuencia:

$$e_0 \rightarrow e_1 \rightarrow \cdots \rightarrow e_i \rightarrow e_{i+1} \rightarrow \cdots \rightarrow e_n$$

- donde e_0 es la expresión inicial y e_n es la expresión terminal, ó el valor de e_0.
- La transformación de e_i a e_{i+1} se da por la aplicación de una única regla de transformación.
- Si e_n es el valor de e_0, se dice que ambas expresiones significan lo mismo, que tienen la misma semántica.

Universidad Veracruzana

Hablado de semántica

- Las reglas de transformación preservan el significado, pues obviamente remplazan iguales por iguales.
- Ejemplo: La evaluación de una expresión aritmética preserva significado:

$$((5+3) \times (8/4)) \to (8 \times (8/4)) \to (8 \times 2) \to 16$$

- Las expresiones en la cadena difieren sintácticamente, pero son equivalentes semánticamente.
- La equivalencia semántica, por otro lado, no nos dice nada sobre lo que significa una expresión.

Universidad Veracruzana

Sintaxis y Semántica

- En un sentido estricto, hablaremos de figuras sintácticas y transformaciones puramente sintácticas de expresiones, en otras expresiones que consideramos semánticamente equivalentes;
- Sin que podamos hablar del significado o semántica de las expresiones en si mismas.

Universidad Veracruzana

Trabajo relacionado

- Aunque el lenguaje que estamos presentando no es un lenguaje estrictamente funcional, si tiene como implementación más cercana a Lisp [6] y a su intérprete `EVAL-APPLY`.
- Ofrecer una aproximación formal a la computabilidad cercana a Lisp y otros lenguajes funcionales.

Universidad Veracruzana

Expresiones atómicas y compuestas

- Las expresiones atómicas en AL incluyen valores constantes:
  - Números
  - Cadenas de caracteres
  - Valores booleanos (`true` y `false`).
- Además de:
  - Variables
  - Operadores aritméticos, lógicos y relacionales.
- Estas expresiones se llaman átomos (o términos de base), porque no están compuestos por otras expresiones del lenguaje.
- Ahora podemos definir expresiones compuestas por expresiones.

Universidad Veracruzana

Aplicaciones

- $(e_0\ e_1 \ldots e_n)$ denota la aplicación de una expresión e_0 a las expresiones $e_1 \ldots e_n$.
- La expresión e_0 está en la posición del operador y las otras expresiones en la posición de los operandos.
- Las aplicaciones son las expresiones más importantes del lenguaje, ya que son a ellas que las reglas de transformación se aplican.
- Observen el uso de la notación prefija con fines de uniformidad.

Universidad Veracruzana

Condicionales

- `if` e_0 `then` e_1 `else` e_2 es una forma sintáctica especial que denota la aplicación del predicado e_0 a las expresiones consecuente e_1 y alternativa e_2.
- Su objetivo es elegir entre e_1 o e_2 para su posterior evaluación, dependiendo del valor del predicado e_0 (`true` o `false`).

Universidad Veracruzana

Abstracción funcional

- `lambda` $v_1 \dots v_n$ `in` e_0 denota una abstracción de las variables $v_1 \dots v_n$ de la expresión e_0; o una función sin nombre de n parámetros formales $v_1 \dots v_n$ cuyo cuerpo e_0 especifica la computación de los valores funcionales.
- Se dice que `lambda` liga las variables $v_1 \dots v_n$ en el cuerpo e_0 que determina su alcance en lo que se conoce como el abstractor lambda.
- Las abstracciones son operadores legítimos.
- Las aplicaciones con abstracciones en la posición del operador regresan el cuerpo de la abstracción con las variables substituidas por las expresiones en la posición de los operandos.

Universidad Veracruzana

Listas

- $\langle e_1 \dots e_n \rangle$ denota una lista n-aria ó una secuencia de expresiones en un orden particular, en el que tiene sentido hablar del primer, último e i-ésimo elemento.
- La lista vacía se denota por $\langle \rangle$.

Universidad Veracruzana

Variables locales recursivas

- `letrec` $f_1 = e_1 \dots\ f_k = e_k$ `in` e_0 define un conjunto de ecuaciones que igualan las variables $f_1, \dots, f_k$ con las expresiones $e_1, \dots, e_k$ recursivamente y las ligan en las expresiones $e_1, \dots, e_k$ y también en e_0.
- Las variables pueden verse como nombres de funciones, para hacer referencia a la parte derecha de la ecuación más adelante en la expresión `letrec`.
- El valor de la expresión es el valor de la expresión meta e_0 en la cual la ocurrencia de los identificadores $f_1, \dots, f_k$ es remplazada recursivamente por el lado derecho de sus definiciones.

Universidad Veracruzana

Variables locales no recursivas

- `let` $v_1 = e_1 \dots v_n = e_n$ `in` e_0 es una versión no recursiva de `letrec`.
- Ninguna otra expresión es válida en el lenguaje.

Universidad Veracruzana

Formas sintácticas

- De manera concisa las formas sintácticas pueden escribirse como:

$$
\begin{aligned}
e \quad =_s \quad & \mathit{const} \mid \mathit{var} \mid \mathit{oper} \mid \\
& (e_0\ e_1 \dots e_n) \mid \\
& \texttt{if } e_0 \texttt{ then } e_1 \texttt{ else } e_2 \mid \\
& \texttt{let } v_1 = e_1 \dots v_n = e_n \texttt{ in } e_0 \mid \\
& \texttt{lambda } v_1 \dots v_n \texttt{ in } e_0 \mid \\
& \langle\rangle \mid \langle e_1 \dots e_n \rangle \mid \\
& \texttt{letrec } f_1 = e_1 \dots f_n = e_n \texttt{ in } e_0.
\end{aligned}
$$

donde el signo $=_s$ denota equivalencia sintáctica.

Universidad Veracruzana

Ejemplos de programas válidos en AL

- Factorial:

$$\text{fac} = \text{lambda } n \text{ in if } (> \ n\ 1) \text{ then } (* \ n\ (fac(-\ n\ 1))) \text{ else } 1)$$

- Ejemplo de segundo orden:

$$\text{doble} = \text{lambda } f \ \ u \ \ \text{in} \ \ (\text{f (f u)})$$
$$\text{cuadrado} = \text{lambda } x \ \ \text{in} \ \ (\text{* x x})$$

- Intuitivamente (fac 5) $\Rightarrow$ 120 y ($doble\ cuadrado$ 2) $\Rightarrow$ 16, pero...
- ¿Cómo sucede ese cómputo?

Universidad Veracruzana

Evaluador

- Definiremos un evaluador abstracto llamado `EVAL`.
- Se le debe considerar una meta-función que mapea cada forma sintáctica en otra que representa su valor.
- En este sentido, `EVAL` puede considerarse la función semántica del lenguaje.
- Esta meta-función se define mediante la aplicación recursiva a:
  - Todas las sub-expresiones de las formas sintácticas, o
  - A algunas de ellas seleccionadas.

Universidad Veracruzana

Estrategia de evaluación

- La selección de las sub-expresiones estará determinada por una estrategia de evaluación que debería computar valores resultantes con un casi mínimo costo computacional.
- Ejemplo. Los operandos de una aplicación generalmente deberán evaluarse antes que el operador se aplique a ellos, i.e., llamada por valor.
- Esta estrategia es implementada por la mayoría de los lenguajes imperativos de programación.

Universidad Veracruzana

Evaluación de expresiones

- Las expresiones atómicas evalúan a si mismas:

$$\texttt{EVAL}[\![\ e\ ]\!] \ = \ e$$

t.q. $e =_s const \mid var \mid oper$.

- Para aplicaciones con expresiones no especificadas en las posiciones de operador y operandos:

$$\texttt{EVAL}[\![\ (e_0\ e_1 \dots e_n)\ ]\!] \ = \ \texttt{EVAL}[\![\ (\texttt{EVAL}[\![\ e_0\ ]\!]\ \texttt{EVAL}[\![\ e_1\ ]\!] \dots \texttt{EVAL}[\![\ e_n\ ]\!])\ ]\!]$$

Universidad Veracruzana

Evaluación de los condicionales

- Para los condicionales tenemos:

$$\texttt{EVAL}[\![\ \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2\ ]\!] =$$
$$\begin{cases} \texttt{EVAL}[\![\ e_1\ ]\!] & \text{Si } \texttt{EVAL}[\![\ e_0\ ]\!] = \textit{true} \\ \texttt{EVAL}[\![\ e_2\ ]\!] & \text{Si } \texttt{EVAL}[\![\ e_0\ ]\!] = \textit{false} \\ \texttt{if}\ \texttt{EVAL}[\![\ e_0\ ]\!]\ \texttt{then}\ e_1\ \texttt{else}\ e_2 & \text{En cualquier otro caso} \end{cases}$$

- Observen que si la forma e_0 no es un predicado, la expresión queda casi idéntica, excepto que la expresión e_0 es evaluada.

Universidad Veracruzana

Evaluación de variables locales

- Para las expresiones `let` tenemos:

$$\texttt{EVAL}[\![\ \texttt{let}\ v_1 = e_1 \dots v_n = e_n\ \texttt{in}\ e_0\ ]\!] =$$
$$\texttt{EVAL}[\![\ e_0[\ v_1 \leftarrow \texttt{EVAL}[\![\ e_1\ ]\!] \dots v_n \leftarrow \texttt{EVAL}[\![\ e_n\ ]\!]\ ]\ ]\!]$$

- La forma evalúa al valor de e_0 en donde todas las ocurrencias de las variables ligadas por `let` han sido substituidas por sus valores respectivos.
- Cada substitución se denota como $v_i \leftarrow\ \texttt{EVAL}[\![\ e_i\ ]\!]$ para $i \in \{1, \dots, n\}$.

Universidad Veracruzana

Evaluación de abstracciones funcionales

- Para las abstracciones anónimas que ocurren en una posición distinta a la del operador, tenemos que:

$$\texttt{EVAL}[\![\ \texttt{lambda}\ v_1 \ldots v_n\ \texttt{in}\ e_0\ ]\!] = \texttt{lambda}\ v_1 \ldots v_n\ \texttt{in}\ \texttt{EVAL}[\![\ e_0\ ]\!]$$

- Las abstracciones necesitan tener su cuerpo evaluado para determinar sus valores.

Universidad Veracruzana

Evaluación de listas

- Para las listas:

$$\begin{aligned} \texttt{EVAL}[\![\ \langle\rangle\ ]\!] &= \langle\rangle \quad \text{y} \\ \texttt{EVAL}[\![\ \langle e_1 \ldots e_n\rangle\ ]\!] &= \langle \texttt{EVAL}[\![\ e_1\ ]\!] \ldots \texttt{EVAL}[\![\ e_n\ ]\!]\rangle \end{aligned}$$

- se computan recursivamente los valores de sus sub-expresiones, preservando su estructura.

Universidad Veracruzana

Evaluación de variables locales recursivas

- Para las expresiones `letrec`:

$$\texttt{EVAL}[\![\ \texttt{letrec}\ \ldots f_i = e_i \ldots\ \texttt{in}\ e_0\ ]\!] =$$
$$\texttt{EVAL}[\![\ e_0[\ \ldots f_i \leftarrow \texttt{EVAL}[\![\ e_i[\ \ldots f_i \leftarrow \texttt{letrec}\ \ldots\ \texttt{in}\ f_i \ldots]\ \ ]\!] \ldots]\ ]\!]$$

- Similar al caso de las variables locales, salvo que las expresiones e_i pueden:
  - Ser recursivas, hacer referencia a f_i.
  - Hacer referencia a f_j con $j \neq i$.

Universidad Veracruzana

Evaluación de variables locales recusivas (2)

- Estos valores a su vez deben computarse substituyendo nuevamente las ocurrencias de los identificadores f_i en las expresiones, por copias de las expresiones completas en `letrec`, las cuales sin embargo, tienen su expresión meta remplazada por los mismos f_i.
- Las formas sintácticas especializada `letrec` $\ldots f_i = e_i \ldots$ `in` f_i de hecho representa las expresiones e_i en su forma no evaluada. Tan pronto como los f_i desaparecen de la expresión (caso base), la expresión `letrec` misma desaparece pues ya no es necesaria para que la computación continúe.

Universidad Veracruzana

Completando la semántica

- Hemos recorrido todas las formas sintáctica, pero la definición de `EVAL` está lejos de ser completa.
- Hasta el momento sólo hemos cubierto los casos generales de las aplicaciones que tienen expresiones no específicas en las posiciones de operador y operandos.
- Lo que falta son los casos especiales donde las expresiones en la posición del operador son (o evalúan a) abstracciones y operaciones primitivas.

Universidad Veracruzana

Aplicaciones completas, caso base

- Estas aplicaciones definen realmente las acciones de transformación de expresiones, por ejemplo, los casos estándar de aplicaciones completas se transforman con `EVAL` como sigue:

$$\texttt{EVAL}[\![\; (\texttt{lambda}\ v_1 \ldots v_n\ \texttt{in}\ e_0\ e_1 \ldots e_n) \;]\!] =$$
$$\texttt{EVAL}[\![\; e_0\ [v_1 \leftarrow \texttt{EVAL}[\![\; e_1 \;]\!] \ldots v_n \leftarrow \texttt{EVAL}[\![\; e_n \;]\!]] \;]\!]$$

Universidad Veracruzana

Más argumentos que parámetros formales

- Si el número de argumentos m es mayor que la aridad de la abstracción n, su valor es una nueva aplicación cuyo operador es una expresión resultado de la aplicación de la abstracción de aridad n a n argumentos y cuyos operandos son los restantes $m - n$ argumentos evaluados:

$$
\begin{aligned}
&\texttt{EVAL}[\![\ (\texttt{lambda}\ v_1 \dots v_n\ \texttt{in}\ e_0\ e_1 \dots e_m)\ ]\!] \mid m > n = \\
&\quad \texttt{EVAL}[\![\ (\ \texttt{EVAL}[\![\ e_0[\ v_1 \leftarrow \texttt{EVAL}[\![\ e_1\ ]\!] \dots v_n \leftarrow \texttt{EVAL}[\![\ e_n\ ]\!]\ ]\ ]\!] \\
&\qquad \texttt{EVAL}[\![\ e_{n+1}\ ]\!] \dots \texttt{EVAL}[\![\ e_m\ ]\!]\ )\ ]\!]
\end{aligned}
$$

Universidad Veracruzana

Aplicaciones parciales

- Si la aridad n de la abstracción excede el número de argumentos m, en cuyo caso tenemos una aplicación parcial, el valor es definido como sigue:

$$\begin{aligned}&\mathtt{EVAL}[\![\ (\mathtt{lambda}\ v_1 \ldots v_n\ \mathtt{in}\ e_0\ e_1 \ldots e_m)\ ]\!] \mid m < n = \\ &\qquad \mathtt{lambda}\ v_{m+1} \ldots v_n\ \mathtt{in}\ \mathtt{EVAL}[\![\ e_0[\ v_1 \leftarrow \mathtt{EVAL}[\![\ e_1\ ]\!] \ldots v_m \leftarrow \mathtt{EVAL}[\![\ e_m\ ]\!]\ ]\ ]\!]\end{aligned}$$

- Hay que ser cuidadosos en este caso con los conflictos entre nombres, *i.e.*, variables acotadas que ocurren libremente en los argumentos.

Universidad Veracruzana

Conflictos entre nombres y Cerraduras

- El renombrado de variables no se considera una solución apropiada dada su complejidad inherente. Podemos adoptar una estrategia conservadora definiendo las evaluaciones parciales como:

$$\texttt{EVAL}[\![\ (\texttt{lambda}\ v_1 \ldots v_n\ \texttt{in}\ e_0\ e_1 \ldots e_m)\ ]\!] \mid m < n =$$
$$[\ \texttt{EVAL}[\![\ e_m\ ]\!] \ldots \texttt{EVAL}[\![\ e_1\ ]\!]\ \texttt{lambda}\ v_1 \ldots v_n\ \texttt{in}\ e_0\ ]$$

- Una cerradura, que tiene los argumentos evaluados y la abstracción empaquetados entre corchetes, representa el valor de una nueva abstracción de aridad $n - m$ sin realmente computarlo.
- La cerradura se evaluará cuando se convierta en una aplicación completa.

Universidad Veracruzana

Operaciones aritméticas

- Para las operaciones aritméticas (binarias) que denotaremos con *op_arit*, y usando *num*, num_1 y num_2 para denotar números, tenemos que:

$$\texttt{EVAL}[\![\; (\; op\_arit \; e_{\mathbf{1}} \; e_{\mathbf{2}} \;) \;]\!] =$$

$$\begin{cases} num & \text{Si } num_{\mathbf{1}} = \texttt{EVAL}[\![\; e_{\mathbf{1}} \;]\!] \wedge num_{\mathbf{2}} = \texttt{EVAL}[\![\; e_{\mathbf{2}} \;]\!] \\ (\; op\_arit \; \texttt{EVAL}[\![\; e_{\mathbf{1}} \;]\!] \; \texttt{EVAL}[\![\; e_{\mathbf{2}} \;]\!] \;) & \text{En cualquier otro caso} \end{cases}$$

Universidad Veracruzana

Predicados

- Para las operaciones relacionales, denotadas por *op_rel*, tenemos que además, str_1 y str_2 denotan cadenas de caracteres, mientras que *bool* denota un valor booleano:

$$
\texttt{EVAL}[\![\ (\ op\_rel\ e_{\mathbf{1}}\ \ e_{\mathbf{2}}\ )\ ]\!] =
$$

$$
\begin{cases}
bool & \text{Si } str_{\mathbf{1}} = \texttt{EVAL}[\![\ e_{\mathbf{1}}\ ]\!] \wedge str_{\mathbf{2}} = \texttt{EVAL}[\![\ e_{\mathbf{2}}\ ]\!] \\
bool & \text{Si } num_{\mathbf{1}} = \texttt{EVAL}[\![\ e_{\mathbf{1}}\ ]\!] \wedge num_{\mathbf{2}} = \texttt{EVAL}[\![\ e_{\mathbf{2}}\ ]\!] \\
(\ op\_rel\ \texttt{EVAL}[\![\ e_{\mathbf{1}}\ ]\!]\ \texttt{EVAL}[\![\ e_{\mathbf{2}}\ ]\!]\ ) & \text{En cualquier otro caso}
\end{cases}
$$

Universidad Veracruzana

Operaciones sobre listas

$$\texttt{EVAL}[\![\; (\; \texttt{empty}\; e \;) \;]\!] = \begin{cases} \texttt{true} & \text{Si } \texttt{EVAL}[\![\; e \;]\!] = \langle\rangle \\ \texttt{false} & \text{Si } \texttt{EVAL}[\![\; e \;]\!] = \langle e_1 \dots e_n \rangle \\ (\texttt{empty}\; \texttt{EVAL}[\![\; e \;]\!]\;) & \text{En cualquier otro caso} \end{cases}$$

$$\texttt{EVAL}[\![\; (\; \texttt{first}\; e \;) \;]\!] = \begin{cases} e_1 & \text{Si } \texttt{EVAL}[\![\; e \;]\!] = \langle e_1 \dots e_n \rangle \\ (\; \texttt{first}\; \texttt{EVAL}[\![\; e \;]\!]\;) & \text{En cualquier otro caso} \end{cases}$$

$$\texttt{EVAL}[\![\; (\; \texttt{rest}\; e \;) \;]\!] = \begin{cases} \langle e_2 \dots e_n \rangle & \text{Si } \texttt{EVAL}[\![\; e \;]\!] = \langle e_1 \; e_2 \dots e_n \rangle \\ (\; \texttt{rest}\; \texttt{EVAL}[\![\; e \;]\!]\;) & \text{En cualquier otro caso} \end{cases}$$

$$\texttt{EVAL}[\![\; (\; \texttt{append}\; e_1 \; e_2 \;) \;]\!] =$$

$$\begin{cases} \langle e_{11} \dots e_{1n} \dots e_{21} \dots e_{2m} \rangle & \text{Si } \texttt{EVAL}[\![\; e_1 \;]\!] = \langle e_{11} \dots e_{1n} \rangle \wedge \\ & \texttt{EVAL}[\![\; e_2 \;]\!] = \langle e_{21} \dots e_{2m} \rangle \\ (\; \texttt{append}\; \texttt{EVAL}[\![\; e_1 \;]\!]\; \texttt{EVAL}[\![\; e_2 \;]\!]\;) & \text{En cualquier otro caso} \end{cases}$$

Universidad Veracruzana

Observaciones

- La definición de `EVAL` incluye una verificación dinámica de tipos. Las aplicaciones de esas funciones son evaluadas solo si sus argumentos son tipo compatibles, en cualquier otro caso quedan casi intactas (sus argumentos son substituidos por sus valores).
- La meta función `EVAL` define la semántica operacional del lenguaje AL. Nos dice:
  - que significa una expresión en el lenguaje (su valor);
  - y también cómo una persona o una máquina pueden computar ese valor.

Universidad Veracruzana

Paralelismo y concurrencia

- Cuando `EVAL` aparece en una aplicación, se propaga al frente de sus sub-expresiones, pero el `EVAL` al frente de la aplicación no desaparece.
- Esto significa que las sub-expresiones deben ser evaluadas antes que la aplicación completa pueda ser evaluada.
- Sin embargo, puesto que las sub-expresiones son sintácticamente independientes, pueden ser evaluadas en cualquier orden, incluso simultáneamente; y sus valores siempre serán los mismos!

Universidad Veracruzana

Ejemplo de evaluación: factorial

$$\texttt{EVAL}[\![\ (\mathit{fac}\ 5)\ ]\!]$$
$$\downarrow$$
$$\texttt{EVAL}[\![\ \texttt{EVAL}[\![\ \mathit{fac}\ ]\!]\ \texttt{EVAL}[\![\ 5\ ]\!]\ ]\!]$$
$$\downarrow$$
$$\texttt{EVAL}[\![\ \text{lambda } n \text{ in if } (>\ \ n\ 1) \text{ then } (*\ n\ (\mathit{fac}(-\ \ n\ 1))) \text{ else } 1)\ 5\ ]\!]$$
$$\downarrow$$
$$\text{if } \texttt{EVAL}[\![\ (>\ \ 5\ 1)\ ]\!] \text{ then } \texttt{EVAL}[\![\ (*\ 5\ (\mathit{fac}(-\ 5\ 1)))\ ]\!] \text{ else } \texttt{EVAL}[\![\ 1\ ]\!]$$
$$\downarrow$$
$$\text{if } \mathit{true} \text{ then } \texttt{EVAL}[\![\ (*\ 5\ (\mathit{fac}(-\ 5\ 1)))\ ]\!] \text{ else } \texttt{EVAL}[\![\ 1\ ]\!]$$
$$\downarrow$$
$$\texttt{EVAL}[\![\ (*\ 5\ (\mathit{fac}(-\ 5\ 1))))\ ]\!]$$
$$\downarrow$$
$$\texttt{EVAL}[\![\ (*\ 5\ \texttt{EVAL}[\![\ (\mathit{fac}(4))\ ]\!]\ ]\!]$$
$$\vdots$$
$$\texttt{EVAL}[\![\ (*\ 5\ (*\ 4\ (*\ 3\ (*\ 2\ 1))))\ ]\!]$$
$$\vdots$$
$$120$$

Universidad Veracruzana

Referente histórico

- Leibniz tenía como ideal [1]:
  1. Crear un lenguaje universal en el que se pudieran enunciar todos los problemas posibles.
  2. Encontrar un método de decisión para resolver los problemas así enunciados.
- (1) se logra adoptado alguna forma de teoría de conjuntos formulada en lógica de primer orden. (2) Church [3, 4] y Turing [7] demostraron independientemente la inexistencia de tal método.
- Para ello, Church [2] inventó el sistema formal llamado Cálculo-λ.
- Turing demostró que sus máquinas y el Cálculo-λ eran modelos fuertemente equivalentes, en el sentido que definen la misma clase de funciones computables.

Universidad Veracruzana

Ideas principales I

- Es el modelo más cercano a los algoritmos y su evaluación tal y como se definen en AL –la reducción por evaluación de las expresiones del lenguaje.
- Paradigma de todos los lenguajes de programación funcional y general siguiendo el teorema de Church-Turing.
- Una teoría de las funciones computables y las propiedades fundamentales de los operadores, sus aplicaciones a operandos y la construcción sistemática de operadores más complejos –algoritmos.

Universidad Veracruzana

Ideas principales II

- Su núcleo tiene que ver con poco más que la definición de variables, su alcance, y la substitución ordenada de variables por expresiones.
- Se trata de un lenguaje cerrado en el sentido de que su semántica puede definirse en base a equivalencia entre expresiones (o términos) del cálculo mismo.

Universidad Veracruzana

Abstracción funcional

- Una función f de n variables $v_1, \ldots, v_n$ se denota como:

$$f = \lambda v_1 \ldots v_n . e_0$$

- El símbolo f denota el nombre de la función y puede ser referenciado en cualquier parte.
- La expresión del lado derecho de la ecuación define una abstracción de las variables $v_1, \ldots, v_n$ en el cuerpo de la función e_0.
- Ejemplo. $suma = \lambda xy.(+\ x\ y)$

Universidad Veracruzana

Aplicación de funciones

- Una aplicación de la función f sobre m expresiones argumento $e_1, \dots, e_m$ tiene la forma:

$$(f\ e_1 \dots\ e_m) = (\lambda v_1 \dots v_n.e_0\ e_1 \dots e_m)$$

  donde $m = n$ no es necesariamente el caso.

- Ejemplos:
  - $(suma\ 3\ 4) = 7$
  - $(suma\ 1) = \lambda y.(+\ 1\ y)$
- El caso especial de la aplicación de una abstracción a las variables abstraídas, nos da como resultado el cuerpo de la abstracción:

$$(\lambda v_1 \dots v_n.e_0\ v_1 \dots v_n) = e_0$$

Universidad Veracruzana

Variables libres, acotadas y substituciones

- Se dice que una abstracción $\lambda v_1.e_0$ acota la variable v_1 en e_0, ssi v_1 ocurre en e_0. En ese caso se dice que la variable v_1 está acotada;
- Si la variable v_1 es exactamente la variable de la abstracción λ, se dice que esta ligada;
- En cualquier otro caso se dice que la variable se dice libre.
- Ejemplo. $\lambda x.(+\ x\ y)$ tiene a x como variable acotada/ligada y a y como variable libre.
- Una substitución $e_0[v_1/t_1]$ solo afecta a las ocurrencias libres de v_1 en e_0.
- Ejemplo. $(x\ \lambda x.x)[x/1] = (1\ \lambda x.x)$

Universidad Veracruzana

Funciones Curryficadas

- Para mantener el aparato formal simple y conciso, el Cálculo-λ considera, sin perdida de generalidad, únicamente abstracciones de una variable.
- Las funciones de más de un argumento se obtienen mediante una iteración de aplicaciones descubierta por Schöenfinkel y Curry:

$$f = \lambda v_1 \dots v_n.e_0 \equiv \lambda v_1.\lambda v_2 \dots .\lambda v_n.e_0$$

- Ejemplo:

$$suma = \lambda x.\lambda y.(+ \ x \ y)$$

Universidad Veracruzana

Sintaxis

- Asumiendo un conjunto infinito de variables *Var* y uno finito, infinito o vacío de constantes *Const*.
- Una expresión (fbf) del Cálculo-λ se define como sigue:

$$e =_s v \mid c \mid (e_0\ e_1) \mid \lambda v.e_0$$

- Es decir, una fbf (o término-λ) es una variable ($v \in Var$), o una constante ($c \in Const$), o una aplicación, o una abstracción.
- Solo estas formas son fbfs.
- Las variables y las constantes se conocen como átomos del sistema.
- En su forma pura, el Cálculo-λ no define constantes. En otro caso, se dice aplicado.

Universidad Veracruzana

Observaciones

- La expresión $(e_0\ e_1)$ denota la aplicación de e_0 a e_1.
- Se dice que e_0 está en la posición del operador; y que e_1 está en la del operando.
- Por ello, es común que e_0 y e_1 se identifiquen como la función y el argumento de la aplicación, lo cual no es totalmente correcto –la sintaxis del Cálculo-λ permite que cualquier expresión esté en cualquiera de las dos posiciones.
- Ejemplo. (*suma* 1 3) es una expresión válida, pero también lo es (3 *suma*).

Universidad Veracruzana

Aplicaciones de abstracciones

- Estamos interesados particularmente en aplicaciones de la forma

$$(\lambda v.e_0\ e_1)$$

que tienen una abstracción en la posición del operador y una expresión válida en la posición del operando.
- Sin operadores primitivos, las abstracciones son las únicas funciones en juego.

Universidad Veracruzana

β-reducción

- La belleza del Cálculo-λ puro reside en que sólo necesitamos preocuparnos por una sola regla de transformación:

$$\lambda v.e_0\ e_1 \rightarrow_\beta\ e_0[v \leftarrow e_1]$$

- La regla se conoce como β-reducción y se dice que reduce el β-redex $\lambda v.e_0\ e_1$ a su reductum $e_0[v \leftarrow e_1]$.
- Observen la substitución $[v \leftarrow e_1]$ inherente a la reducción.

Universidad Veracruzana

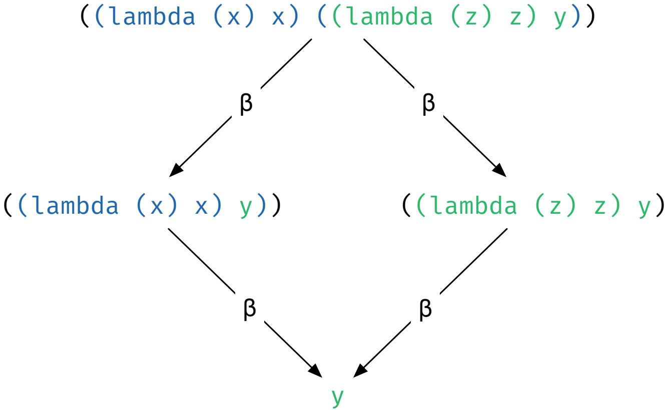
Conflictos entre nombres de variables

- Desafortunadamente esta regla no es tan simple como parece a primera vista.
- Existen problemas con respecto a las variables ligadas en las abstracciones y la existencia de variables libres con los mismos nombres, en cuyo caso, las substituciones no pueden llevarse a cabo sin una acción correctiva.
- Ejemplo:

$$(\lambda u.\lambda v.u\ w) \rightarrow \lambda v.w \quad \text{y} \quad (\lambda u.\lambda v.u\ v) \rightarrow \lambda v.v$$

Universidad Veracruzana

No determinismo



Universidad Veracruzana

Variables libres

- El conjunto de variables libres de una expresión e se define por casos sintáticos:

$$FV(e) = \begin{cases} \{v\} & \text{Si } e =_s v \in Var \\ FV(e_0) \cup FV(e_1) & \text{Si } e =_s (e_0 \ e_1) \\ FV(e_0) \setminus \{v\} & \text{Si } e =_s \lambda v.e_0 \end{cases}$$

Universidad Veracruzana

Variables acotadas

- Es posible ofrecer una definición complementaria del conjunto de variables acotadas en la expresión e:

$$BV(e) = \begin{cases} \emptyset & \text{Si } e =_s v \in V \\ BV(e_0) \cup BV(e_1) & \text{Si } e =_s (e_0\ e_1) \\ BV(e_0) \cup \{v\} & \text{Si } e =_s \lambda v.e_0 \end{cases}$$

- Una variable v está libre en una expresión e, si y sólo si $v \in FV(e)$; y que una variable v está acotada en una expresión e, si y sólo si $v \in BV(e)$.

Universidad Veracruzana

Alcance del abstractor

- En una abstracción $\lambda v.e$, llamamos al cuerpo e el alcance del abstractor λv.
- Esto significa que todas las ocurrencias libres de v en e están acotadas por λv.
- Ejemplo. En la expresión:

$$\lambda x.x$$

x no es libre, pero en:

$$\lambda x.y$$

y si lo es.

Universidad Veracruzana

Abstracciones cerradas, abiertas y combinadores

- Una abstracción cuyo conjunto de variables libres está vacío, se dice cerrada, o que es un combinador;
- En otro caso se dice que la abstracción es abierta.
- Los combinadores son de gran relevancia para definir lenguajes basados en Cálculo-λ.
- Ejemplo. Los combinadores estándar:

$$\begin{aligned} \mathsf{I} &= \lambda x.x \\ \mathsf{K} &= \lambda xy.x \\ \mathsf{K}_* &= \lambda xy.y \\ \mathsf{S} &= \lambda xyz.xz(yz) \end{aligned}$$

Universidad Veracruzana

Substituciones

- Como se lleva a cabo la substitución en la β-reducción

$$(\lambda v.e_b\ e_a) \rightarrow_\beta e_b[v \leftarrow e_a]$$

$$e_b[v \leftarrow e_a] = \begin{cases} e_a & \text{Si } e_b =_s v \in Var \\ u & \text{Si } e_b =_s u \in Var \wedge v \neq_s u \\ (e_0[v \leftarrow e_a]\ e_1[v \leftarrow e_a]) & \text{Si } e_b =_s (e_0\ e_1) \\ \lambda v.e_0 & \text{Si } e_b =_s \lambda v.e_0 \\ \lambda u.e_0[v \leftarrow e_a] & \text{Si } e_b =_s \lambda u.e_0 \quad \wedge \\ & u \notin FV(e_a) \vee v \notin FV(e_b) \\ \lambda w.e_0[u \leftarrow w][v \leftarrow e_a] & \text{Si } e_b =_s \lambda u.e_0 \quad \wedge \\ & u \in FV(e_a) \wedge v \in FV(e_b) \quad \wedge \\ & w \in Var \wedge w \notin FV(e_a) \cup FV(e_b) \end{cases}$$

Universidad Veracruzana

α-conversión

- La transformación que renombra la variable acotada:

$$\lambda u.e_0 \rightarrow_\alpha \lambda w.e_0[u \leftarrow w]$$

se conoce como α-conversión.
- Su realización se basa en la aplicación de una función de α-conversión a la abstracción cuya variable acotada necesitamos cambiar:

$$(\lambda v.\lambda w.(v\ w)\ \lambda u.e_0)$$

- Esta transformación procede en dos pasos: primero mapea a $\lambda w.(\lambda u.e_0\ w)$ y después a $\lambda w.e_0[u \leftarrow w]$.

Universidad Veracruzana

Estructura de ligado

- La elección de los nombres de las variables ligadas no es importante.
- Su única función es relacionar a los abstractores con posiciones sintácticas en el cuerpo de la abstracción.
- A esta relación se le conoce como estructura de ligado.
- Ejemplo:

$$\lambda u.\lambda v.\lambda w.(((w\ v)\ u)(x\ u)) \equiv \lambda x_1.\lambda x_2.\lambda x_3.(((x_3\ x_2)\ x_1)(x\ x_1))$$

Universidad Veracruzana

Estructura de substitución

- La posición de un abstractor en una secuencia de abstractores también identifica, en el caso de aplicaciones anidadas, el nivel de anidamiento en que el argumento será tomado.
- A esto se le llama estructura de substitución.

estructura de substitucion

$$(\;(\;(\;\lambda u.\lambda v.\lambda w.(\;(\;(\;w\;v\;)\;u\;)\;(\;x\;u\;)\;)\;x\;)\;\lambda z.z\;)\;\lambda x.\lambda y.x\;)$$

estructura de ligado

Universidad Veracruzana

Secuencias de reducción

- Dadas dos λ-expresiones e y e', se dice que e es β-reducible a e', denotado por $e \longmapsto_\beta e'$, si y sólo si e puede ser transformado en e' por una secuencia finita (posiblemente vacía) de β-reducciones y α-conversiones.
- Esto produce una secuencia $e_0, \ldots, e_n$ de λ-expresiones con $e_0 =_s e$ y $e_n =_s e'$ tal que para todos los índices $i \in \{0, \ldots, n-1\}$ tenemos que $e_i \rightarrow_\beta e_{i+1}$ ó $e_i \rightarrow_\alpha e_{i+1}$.

Universidad Veracruzana

Equivalencia semántica

- Dos λ-expresiones son semánticamente equivalentes, denotado por $e \equiv e'$, si y sólo si e puede ser transformada en e' por una secuencia finita (posiblemente vacía) de β-reducciones, β-reducciones inversas y α-conversiones.
- Esto es, para todos los índices $i \in \{0, \ldots, n-1\}$, debemos tener $e_i \rightarrow_\beta e_{i+1}$ ó $e_{i+1} \rightarrow_\beta e_i$ ó $e_i \rightarrow_\alpha e_{i+1}$.

Universidad Veracruzana

Forma Normal

- El objetivo de reducir una expresión-λ e es transformarla en alguna expresión e^{NF} que no contiene más redices, ó a la que no se le puedan aplicar más reglas de β-reducción.
- Esta expresión se conoce como la forma normal ó el valor de la expresión e.
- Si para llegar de e a e^{NF} necesitamos una secuencia finita de β-reducciones, entonces e^{NF} es también la forma normal de las expresiones-λ intermedias.

Universidad Veracruzana

Indeterminismo

- Una expresión-λ que contiene diversos redices, permite una elección entre diferentes secuencias alternativas de β-reducciones.
- Podemos alcanzar una forma normal, en:
  - todas las posibles secuencias, p. ej., tras muchas β-reducciones finitas; si tenemos suerte.
  - ninguna de las secuencias posibles, porque no existe una forma normal, i.e., ninguna de las secuencias termina en un número finito de pasos;
  - algunas de las secuencias posibles, pero no en todas, p. ej, algunas secuencias terminan de manera finita, pero otras no.
- El último caso es muy interesante porque demanda una estrategia

Universidad Veracruzana

Ejemplo del caso 1

- Consideremos la expresión: $(\lambda u.(\lambda w.(\lambda w.u\ u)\ u)\ w)$ donde:
  - procediendo de afuera hacía adentro:

    $$(\lambda u.(\lambda w.(\lambda w.u\ u)\ u)\ w) \rightarrow_\beta (\lambda w.(\lambda(w.//w\ /w)w) \rightarrow_\beta (\lambda w./w\ w) \rightarrow_\beta w$$

  - procediendo de adentro hacía afuera:

    $$(\lambda u.(\lambda w.(\lambda w.u\ u)\ u)\ w) \rightarrow_\beta (\lambda u.(\lambda w.u\ u)\ w) \rightarrow_\beta (\lambda u.u\ w) \rightarrow_\beta w$$

  - y aún si comenzásemos por el radice de en enmedio, también llegaríamos a la forma normal w.

Universidad Veracruzana

Ejemplo del caso 2

- Ahora, una expresión simple que no tiene forma normal es la auto-aplicación:

$$(\lambda u.(u\ u)\ \lambda u.(u\ u)) \rightarrow_\beta (\lambda u.(u\ u)\ \lambda u.(u\ u)) \rightarrow_\beta \ldots$$

que se reproduce a si misma.

Universidad Veracruzana

Ejemplo del caso 3

- Observen la siguiente aplicación:

$$((\lambda u.\lambda v.u \ \lambda w.w) \quad (\lambda u.(u \ u) \ \lambda u.(u \ u)))$$

- La abstracción $\lambda u.\lambda v.u$ es una función selector que reproduce su primer argumento, es decir $\lambda w.w$, pero elimina el segundo, que en este caso particular es la forma auto replicante del ejemplo anterior.
- De forma que una reducción de adentro hacía afuera lleva a $\lambda w.w$, pero una de afuera hacía adentro no termina.

Universidad Veracruzana

Reducción en orden normal

- Una estrategia que garantiza la terminación, mediante una función de transformación τ_N:

$$\tau_N(e) = \begin{cases} v & \text{Si } e =_s v \in Var \\ \lambda v.\tau_N(e_b) & \text{Si } e =_s \lambda v.e_b \\ \tau'_N(e) & \text{Si } e =_s (\lambda v.e_b\ e_2) \\ \tau'_N(\tau_N(e_1)e_2) & \text{Si } e =_s (e_1\ e_2) \text{ y } e_1 \neq_s \lambda v.e_b \end{cases}$$

- donde:

$$\tau'_N(e) = \begin{cases} \tau_N(e_b[v \leftarrow e_2]) & \text{Si } e =_s (\lambda v.e_b\ e_2) \\ (e_1\ \tau_N(e_2)) & \text{Si } e =_s (e_1\ e_2) \text{ y } e_1 \neq_s \lambda v.e_b \end{cases}$$

Universidad Veracruzana

Evaluador abstracto

- La función τ_N define un evaluador abstracto para expresiones del Cálculo-λ puro, similar al evaluador `EVAL` de AL.
- A diferencia de `EVAL`, τ_N solo se propaga recursivamente a través de los operadores, pero no toca los operandos hasta que el operador es procesado y no es una abstracción (el último caso en τ'_N).
- Si es una abstracción, entonces el operando es substituido por ocurrencias libres de la variable ligada (el primer caso de τ'_N).

Universidad Veracruzana

Llamadas por nombre y por valor

- La reducción en orden normal también se conoce como de afuera a adentro y de izquierda a derecha.
- Esta estrategia se conoce también como llamada por nombre y es usada por lenguajes de programación como Algol y Simula.
- Problemas: Reducciones redundantes si el operando es substituído en más de un lugar. Solución: Evaluación perezosa.
- Es posible definir una estrategias como la usada por `EVAL`, conocida como primero operandos o llamada por valor.

Teorema de Church-Rosser

- Sean e_0, e_1, e_2, e_3 λ-expresiones. Entonces $e_0 \longmapsto_\beta e_1$ y $e_0 \longmapsto_\beta e_2$ implican que existe una expresión e_3 tal que $e_1 \longmapsto_\beta e_3$ y $e_2 \longmapsto_\beta e_3$.
- La prueba está más allá del contenido de este curso, pero observen que si e_3 es una forma normal, entonces esta forma es única.
- Dicho de otra manera, la forma normal de una expresión-λ es invariante con respecto a las diversas secuencias de reducción que se pueden seguir para alcanzarla.

Universidad Veracruzana

Transparencia referencial

- Si obviamos el problema del conflicto entre nombres, la reducción-β es bastante sencilla.
- Lleva a cabo substituciones libres de contexto entre iguales; es decir, remplaza una aplicación por otra, sin causar ningún efecto colateral en la expresión donde se encuentra (su contexto).
- A esta propiedad se le llama transparencia referencial.
- Además, la sintaxis del Cálculo-λ asegura que los redices-β no se traslapen: No comparten componentes.

Universidad Veracruzana

Punto fijo

- El Cálculo-λ no sabe como definir ecuaciones.
- Podríamos intentar lograr la recursividad vía la auto-aplicación, de la que conocemos el caso especial $(\lambda u.(u\ u)\ \lambda u.(u\ u))$.
- Mejor aún, deberíamos buscar un Operador de recursividad universal, digamos s, tal que: $(s\ f) = (f(s\ f))$.
- Es fácil ver que $(s\ f)$ es un punto fijo de f: es una expresión que f mapea a si misma. Por lo que necesitamos un combinador de punto fijo:

$$Y =_s (p\ p)$$

donde $p =_s \lambda u.\lambda v.(v((u\ u)\ v))$.

Universidad Veracruzana

Comportamiento del combinador-Y

- Probemos un ejemplo para observar el comportamiento de Y:

$$
\begin{gathered}
(Y\ f) =_s ((\lambda u.\lambda v.(v((u\ u)\ v))\ p)\ f) \rightarrow_\beta \\
(\lambda v.(v\ ((p\ p)\ v))\ f) \rightarrow_\beta \\
(f\ ((p\ p)\ f)) =_s (f\ \ Y(f))
\end{gathered}
$$

Universidad Veracruzana

Ejemplo I

- Consideren la ecuación para una función recursiva:

$$f = \lambda u.\lambda v.(((f\ u)\ v)((f\ v)\ u))$$

- Podemos escribirla como una aplicación equivalente:

$$f = (\lambda z.\underbrace{\lambda u.\lambda v.(((z\ u)\ v)((z\ v)\ u))}_{e_b}\ f)$$

- Y usando e_b como una abreviatura, como:

$$f = (\lambda z.e_b\ f)$$

Universidad Veracruzana

Ejemplo II

- Para obtener la forma de $(Y\ f) = (f(Y\ f))$, tenemos:

$$f = (Y\ \lambda z.e_b)$$

de forma que:

$$(Y\ \lambda z.e_b) = (\lambda z.e_b\ (Y\ \lambda z.e_b))$$

Universidad Veracruzana

Factorial revisitado

- Definiciones auxiliares:

$$\begin{aligned} true &\equiv \lambda x.\lambda y.x \\ false &\equiv \lambda x.\lambda y.y \\ cond &\equiv \lambda p.\lambda c.\lambda a.((p\ c)\ a) \end{aligned}$$

- Recursiva:

$$fac \equiv \lambda n.cond\ (=\ n\ 0)\ 1\ (*(fac(-\ n\ 1)))$$

- Con el combinador-Y:

$$fac \equiv Y(\lambda fac.\lambda n.cond\ (=\ n\ 0)\ 1\ (*n(fac(-\ n\ 1)))$$

donde:

$$(=\ n\ n) \mapsto true \text{ y } (=\ n\ m) \mapsto false$$

Universidad Veracruzana

Referencias I

[1] H Barendregt y E Barendsen. *Introduction to Lambda Calculus*. 2000.

[2] A Church. "A Set of Postulates for the Foundation of Logic". En: *Annals of Mathematics* 33.2 (1932), págs. 346-366.

[3] A Church. "A Note on the Entscheidungsproblem". En: *Journal of Symbolic Logic* 1 (1936), págs. 40-41.

[4] A Church. "An unsolvable problem of elementary number theory". En: *American journal of mathematics* 58.2 (1936), págs. 345-363.

[5] W Kluge. *Abstract Computing Machines: A Lambda Calculus Perspective*. Berlin, Germany New York: Springer-Verlag, 2005.

[6] J McCarthy. *LISP 1.5 Programmer's Manual*. The MIT Press, 1962. ISBN: 0262130114.

[7] AM Turing. "On the Computable Numbers, with Applications to the Entscheidungsproblem". En: *Proceedints of the London Mathematical Society*. Vol. 42. series 2. 1936, págs. 230-265.

Universidad Veracruzana