5 Formalizaciones - Un Panorama III

José de Jesús Lavalle Martínez

Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias de la Computación
Computabilidad CCOS 257

Contenido

Motivación

- Aquí hay otro lenguaje de programación.

Motivación

- Por un lado, es extremadamente simple, incluso más simple que el lenguaje de los programas loop-while.

Motivación

- Por otro lado, el lenguaje es “desestructurado”; incorpora (de hecho) comandos go-to.

Motivación

- Esta formalización fue presentada por Shepherdson y Sturgis en un artículo de 1963.

Máquinas de Registros I

- Una *máquina de registros* debe considerarse como un dispositivo de cómputo con un número finito de "registros", numerados $0, 1, 2, \ldots, K$.

Máquinas de Registros I

- Cada registro es capaz de almacenar un número natural de cualquier magnitud; no hay límite para el tamaño de este número.

Máquinas de Registros I

- El funcionamiento de la máquina está determinado por un programa.

Máquinas de Registros I

- Un programa es una secuencia finita de instrucciones, extraídas de la siguiente lista:

Máquinas de Registros II

- "Incrementa r", $I\ r$ (donde $0 \leqslant r \leqslant K$): El efecto de esta instrucción es aumentar el contenido del registro r en 1. Luego, la máquina pasa a la siguiente instrucción del programa (si corresponde).

Máquinas de Registros II

- "Decrementa r", $D\ r$ (donde $0 \leqslant r \leqslant K$): El efecto de esta instrucción depende del contenido del registro r. Si ese número es distinto de cero, se decrementa en 1 y la máquina no pasa a la siguiente instrucción, sino a la siguiente de la siguiente. Pero si el número en el registro r es cero, la máquina simplemente pasa a la siguiente instrucción. En resumen, la máquina intenta disminuir el registro r y, si tiene éxito, se salta una instrucción.

Máquinas de Registros II

- "Jump q", $J\ q$ (donde q es un número entero: positivo, negativo o cero): todos los registros se dejan sin cambios. La máquina toma como siguiente instrucción la $q-$ésima instrucción que sigue a ésta en el programa (si $q \geqslant 0$), o la $|q|-$ésima instrucción que precede a ésta (si $q < 0$). La máquina se detiene si no existe tal instrucción en el programa. La instrucción $J\ 0$ da como resultado un bucle, en el que la máquina ejecuta esta instrucción una y otra vez.

Máquinas de Registros III

- Y eso es todo.

Máquinas de Registros III

- El lenguaje tiene sólo estos tres tipos de instrucciones.

Máquinas de Registros III

- Estrictamente hablando, en estas instrucciones, r y q son numerales, no números.

Máquinas de Registros III

- Es decir, una instrucción debe ser una secuencia de símbolos.

Máquinas de Registros III

- Si usamos números en base 10, entonces el alfabeto es $\{I, D, J, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -\}$.

Máquinas de Registros III

- Una instrucción es una palabra correctamente formada sobre este alfabeto.

Ejemplo 1

CLEAR 7: un programa para borrar el registro 7.

$D\ 7$	Trata de decrementar el registro 7.
$J\ 2$	Para.
$J\ -2$	Regresa y repite.

Ejemplo 2

MOVE from r to s: un programa para mover un número del registro r al registro s (donde $r \neq s$).

CLEAR s	Use el programa del primer ejemplo.
$D\ r$	Trata de decrementar el registro r.
$J\ 3$	Para cuando sea cero.
$I\ s$	incrementa el registro s.
$J - 3$	Regresa y repite.

Ejemplo 3

ADD 1 to 2 and 3: un programa para sumar el contenido del registro 1 a los registros 2 y 3.

$$D\ 1$$
$$J\ 4$$
$$I\ 2$$
$$I\ 3$$
$$J - 4$$

Ejemplo 4 I

COPY from r to s: un programa para copiar un número del registro r al registro s (dejando el registro r sin cambios). Combinamos los ejemplos anteriores.

CLEAR s	Use el programa del primer ejemplo.
MOVE from r to s	Use el programa del segundo ejemplo.
ADD t to r and s	Use el programa del tercer ejemplo.

Ejemplo 4 II

- Este programa tiene 15 instrucciones.

Ejemplo 4 II

- Utiliza un tercer registro, el registro t.

Ejemplo 4 II

- Al final, se restaura el contenido del registro r.

Ejemplo 4 II

- Pero durante la ejecución, se debe borrar el registro r; ésta es la única manera de determinar su contenido.

Ejemplo 4 II

- Aquí se supone que r, s y t son distintos.

Ejemplo 5

- (Adición) Digamos que x y y están en los registros 1 y 2. Queremos $x + y$ en el registro 0, y queremos dejar x y y todavía en los registros 1 y 2 al final.

	Contenido de los registros			
CLEAR 0	0	x	y	
MOVE from 1 to 3	0	0	y	x
ADD 3 to 1 and 0	x	x	y	0
MOVE from 2 to 3	x	x	0	y
ADD 3 to 2 and 0	$x + y$	x	y	0

Ejemplo 5

- (Adición) Digamos que x y y están en los registros 1 y 2. Queremos $x + y$ en el registro 0, y queremos dejar x y y todavía en los registros 1 y 2 al final.

	Contenido de los registros			
CLEAR 0	0	x	y	
MOVE from 1 to 3	0	0	y	x
ADD 3 to 1 and 0	x	x	y	0
MOVE from 2 to 3	x	x	0	y
ADD 3 to 2 and 0	$x + y$	x	y	0

- Este programa tiene 27 instrucciones tal como está escrito, pero tres de ellas son innecesarias.

Ejemplo 5

- (Adición) Digamos que x y y están en los registros 1 y 2. Queremos $x + y$ en el registro 0, y queremos dejar x y y todavía en los registros 1 y 2 al final.

	Contenido de los registros			
CLEAR 0	0	x	y	
MOVE from 1 to 3	0	0	y	x
ADD 3 to 1 and 0	x	x	y	0
MOVE from 2 to 3	x	x	0	y
ADD 3 to 2 and 0	$x + y$	x	y	0

- Este programa tiene 27 instrucciones tal como está escrito, pero tres de ellas son innecesarias.
- En la cuarta línea, comenzamos borrando el registro 3, que ya está limpio.

- Ahora supongamos que f es una función parcial de aridad n sobre $\mathbb{N}$.

- Posiblemente, habrá un programa $\mathcal{P}$ tal que si arrancamos una máquina de registros (que tiene todos los registros a los que se refiere $\mathcal{P}$) con $x_1, \ldots, x_n$ en los registros $1, \ldots, n$ y 0 en los otros registros, y aplicamos el programa $\mathcal{P}$, entonces se cumplen las siguientes condiciones:

- - Si $f(x_1, \ldots, x_n)$ está definida, entonces el cálculo finalmente termina con $f(x_1, \ldots, x_n)$ en el registro 0. Además, el cálculo termina al llegar a la $(p+1)$-ésima instrucción, donde p es la longitud de $\mathcal{P}$.
  - Si $f(x_1, \ldots, x_n)$ no está definida, el cálculo nunca termina.

- Si existe tal programa $\mathcal{P}$, decimos que $\mathcal{P}$ calcula f.

Máquinas de Registros IV

- ¿Qué funciones son computables por programas de máquinas de registros?

Máquinas de Registros IV

- El lenguaje es tan simple (parece un lenguaje de juguete) que la primera impresión podría ser que sólo funciones muy simples son computables.

Máquinas de Registros IV

- Esta impresión es engañosa.

Teorema 1

Sea f una función parcial. Entonces, existe un programa de máquinas de registros que calcula f si y sólo si f es una función parcial recursiva general.

Máquinas de Registros IV

- Así, al utilizar máquinas de registros, llegamos exactamente a la clase de funciones parciales recursivas generales, una clase que definimos originalmente en términos de recursión primitiva y búsqueda.

Definibilidad en lenguajes formales I

- Esbozaremos brevemente otras formas en las que podría formalizarse el concepto de calculabilidad efectiva. Los detalles quedarán a la imaginación.

Definibilidad en lenguajes formales I

- En 1936, en su artículo en el que presentaba lo que ahora se conoce como la tesis de Church, Alonzo Church utilizó un sistema formal, el *cálculo*$-\lambda$.

Definibilidad en lenguajes formales I

- Church había desarrollado este sistema como parte de su estudio de los fundamentos de la lógica.

Definibilidad en lenguajes formales I

- En particular, para cada número natural, n, existe una fórmula $\bar{n}$ del sistema que denota n, es decir, un numeral para n.

Definibilidad en lenguajes formales I

- Más importante aún, se podrían utilizar fórmulas para representar la construcción de funciones.

Definibilidad en lenguajes formales I

- Definió una función F de aridad dos como *definible*$-\lambda$ si existía una fórmula F del cálculo lambda tal que siempre que $F(m,n) = r$, entonces la fórmula $\{\boldsymbol{F}\}(\bar{m},\bar{n})$ fuera convertible, siguiendo las reglas del sistema, a la fórmula $\bar{r}$.

Definibilidad en lenguajes formales I

- Esta definición se puede extender a funciones de aridad k.

Definibilidad en lenguajes formales II

- Stephen Kleene, que fue alumno de Church, demostró que una función era definible$-\lambda$ si y sólo si era recursiva general.

Definibilidad en lenguajes formales II

- Church y su alumno J. B. Rosser también participaron en el desarrollo de este resultado.

Definibilidad en lenguajes formales II

- Church escribió en su artículo:

Definibilidad en lenguajes formales II

- "El hecho. . . de que dos definiciones tan diferentes y (en opinión del autor) igualmente naturales de calculabilidad efectiva resulten ser equivalentes aumenta la fuerza de las razones. . . por creer que constituyen una caracterización tan general de esta noción como es consistente con la comprensión intuitiva habitual de la misma".

Definibilidad en lenguajes formales III

- Anteriormente, en 1934, Kurt Godel, en conferencias en Princeton, formuló un concepto que ahora se conoce como computabilidad de Godel-Herbrand.

Definibilidad en lenguajes formales III

- Sin embargo, en ese momento no propuso el concepto como una formalización del concepto de calculabilidad efectiva.

Definibilidad en lenguajes formales III

- El concepto implicaba un cálculo formal de ecuaciones entre términos construidos a partir de variables y símbolos de funciones.

Definibilidad en lenguajes formales III

- El cálculo permitía pasar de una ecuación $A = B$ a otra ecuación obtenida sustituyendo una parte C de A o B por otro término D del que se había derivado la ecuación $C = D$.

Definibilidad en lenguajes formales III

- Si un conjunto $\mathcal{E}$ de ecuaciones permitía derivar, en un sentido adecuado, exactamente los valores correctos para una función f sobre $\mathbb{N}$, entonces se decía que $\mathcal{E}$ era un conjunto de ecuaciones recursivas para f.

Definibilidad en lenguajes formales III

- Una vez más, resultó que existía un conjunto de ecuaciones recursivas para f si y sólo si f era una función recursiva general.

Definibilidad en lenguajes formales IV

- Un enfoque bastante diferente para caracterizar las funciones efectivamente calculables implica la definibilidad mediante expresiones en lógica simbólica.

Definibilidad en lenguajes formales IV

- Un lenguaje formal para la aritmética de números naturales podría tener variables y un número para cada número natural, y símbolos para la relación de igualdad y, al menos, para las operaciones de suma y multiplicación.

Definibilidad en lenguajes formales IV

- Además, el lenguaje debería poder manejar los conectivos lógicos básicos como “y”, “o” y “no”.

Definibilidad en lenguajes formales IV

- Finalmente, debe incluir las expresiones "cuantificadoras" $\forall v$ y $\exists v$ que significan "para todos los números naturales v" y "para algún número natural v", respectivamente.

Definibilidad en lenguajes formales V

- Por ejemplo,

$$\exists s(u_1 + s = u_2)$$

podría ser una expresión en el lenguaje formal, afirmando una propiedad de u_1 y u_2.

Definibilidad en lenguajes formales V

- Por ejemplo,

$$\exists s(u_1 + s = u_2)$$

  podría ser una expresión en el lenguaje formal, afirmando una propiedad de u_1 y u_2.
- La expresión es verdadera (en $\mathbb{N}$ con sus operaciones habituales) cuando a u_1 se le asigna 4 y a u_2 se le asigna 9 (tome $s = 5$).

Definibilidad en lenguajes formales V

- Por ejemplo,

$$\exists s(u_1 + s = u_2)$$

  podría ser una expresión en el lenguaje formal, afirmando una propiedad de u_1 y u_2.
- La expresión es verdadera (en $\mathbb{N}$ con sus operaciones habituales) cuando a u_1 se le asigna 4 y a u_2 se le asigna 9 (tome $s = 5$).
- Pero es falso cuando a u_1 se le asigna 9 y a u_2 se le asigna 4.

Definibilidad en lenguajes formales V

- Por ejemplo,

$$\exists s(u_1 + s = u_2)$$

  podría ser una expresión en el lenguaje formal, afirmando una propiedad de u_1 y u_2.
- La expresión es verdadera (en $\mathbb{N}$ con sus operaciones habituales) cuando a u_1 se le asigna 4 y a u_2 se le asigna 9 (tome $s = 5$).
- Pero es falso cuando a u_1 se le asigna 9 y a u_2 se le asigna 4.
- De manera más general, podemos decir que la expresión define (en $\mathbb{N}$ con sus operaciones habituales) la relación binaria "$\leqslant$" sobre $\mathbb{N}$.

Definibilidad en lenguajes formales VI

- Para otro ejemplo,

$$v \neq 0 \wedge \forall x \forall y [\exists s(v + s = y) \vee \exists t(v + t = y) \vee v \neq x \cdot y]$$

podría ser una expresión en el lenguaje formal, afirmando una propiedad de v.

Definibilidad en lenguajes formales VI

- Para otro ejemplo,

$$v \neq 0 \wedge \forall x \forall y [\exists s(v + s = y) \vee \exists t(v + t = y) \vee v \neq x \cdot y]$$

  podría ser una expresión en el lenguaje formal, afirmando una propiedad de v.
- La expresión es falsa (sobre $\mathbb{N}$ con su operación habitual) cuando a v se le asigna el número 6 (pruebe con $x = 2$ y $y = 3$).

Definibilidad en lenguajes formales VI

- Para otro ejemplo,

$$v \neq 0 \wedge \forall x \forall y [\exists s(v + s = y) \vee \exists t(v + t = y) \vee v \neq x \cdot y]$$

  podría ser una expresión en el lenguaje formal, afirmando una propiedad de v.
- La expresión es falsa (sobre $\mathbb{N}$ con su operación habitual) cuando a v se le asigna el número 6 (pruebe con $x = 2$ y $y = 3$).
- Pero la expresión es verdadera cuando a v se le asigna 7.

Definibilidad en lenguajes formales VI

- Para otro ejemplo,

$$v \neq 0 \wedge \forall x \forall y [\exists s(v + s = y) \vee \exists t(v + t = y) \vee v \neq x \cdot y]$$

  podría ser una expresión en el lenguaje formal, afirmando una propiedad de v.
- La expresión es falsa (sobre $\mathbb{N}$ con su operación habitual) cuando a v se le asigna el número 6 (pruebe con $x = 2$ y $y = 3$).
- Pero la expresión es verdadera cuando a v se le asigna 7.
- De manera más general, la expresión es verdadera cuando a v se le asigna un número primo, y sólo entonces.

Definibilidad en lenguajes formales VI

- Para otro ejemplo,

$$v \neq 0 \wedge \forall x \forall y [\exists s(v + s = y) \vee \exists t(v + t = y) \vee v \neq x \cdot y]$$

  podría ser una expresión en el lenguaje formal, afirmando una propiedad de v.
- La expresión es falsa (sobre $\mathbb{N}$ con su operación habitual) cuando a v se le asigna el número 6 (pruebe con $x = 2$ y $y = 3$).
- Pero la expresión es verdadera cuando a v se le asigna 7.
- De manera más general, la expresión es verdadera cuando a v se le asigna un número primo, y sólo entonces.
- Podemos decir que esta expresión define el conjunto de los números primos (sobre $\mathbb{N}$ con sus operaciones habituales).

Definibilidad en lenguajes formales VI

- Decimos que una función parcial f de aridad k sobre $\mathbb{N}$ es *definible*$-\Sigma_1$ si la gráfica de f (es decir, la relación de aridad $(k+1)$ $\{<\vec{x}, y> | f(\vec{x}) = y\}$) puede ser definida sobre $\mathbb{N}$ con las operaciones de suma, multiplicación y exponenciación, mediante una expresión de la siguiente forma:

$$\exists v_1 \exists v_2 \cdots \exists v_n (\text{expresión sin cuantificadores})$$

Definibilidad en lenguajes formales VI

- Decimos que una función parcial f de aridad k sobre $\mathbb{N}$ es *definible*$-\Sigma_1$ si la gráfica de f (es decir, la relación de aridad $(k+1)$ $\{< \vec{x}, y > | f(\vec{x}) = y\}$) puede ser definida sobre $\mathbb{N}$ con las operaciones de suma, multiplicación y exponenciación, mediante una expresión de la siguiente forma:

$$\exists v_1 \exists v_2 \cdots \exists v_n (\text{expresión sin cuantificadores})$$

- Entonces la clase de funciones parciales *definibles*$-\Sigma_1$ coincide exactamente con la clase de funciones parciales dada por las otras formalizaciones de calculabilidad descritas aquí.

Definibilidad en lenguajes formales VI

- Decimos que una función parcial f de aridad k sobre $\mathbb{N}$ es *definible*$-\Sigma_1$ si la gráfica de f (es decir, la relación de aridad $(k+1)$ $\{< \vec{x}, y > | f(\vec{x}) = y\}$) puede ser definida sobre $\mathbb{N}$ con las operaciones de suma, multiplicación y exponenciación, mediante una expresión de la siguiente forma:

$$\exists v_1 \exists v_2 \cdots \exists v_n (\text{expresión sin cuantificadores})$$

- Entonces la clase de funciones parciales *definibles*$-\Sigma_1$ coincide exactamente con la clase de funciones parciales dada por las otras formalizaciones de calculabilidad descritas aquí.
- Además, Yuri Matiyasevich demostró en 1970 que aquí no era necesaria la operación de exponenciación.

Definibilidad en lenguajes formales VII

- Finalmente, digamos que una función parcial f de aridad k sobre $\mathbb{N}$ es *representable* si existe alguna teoría finitamente axiomatizable $\boldsymbol{T}$ en un lenguaje que tenga un numeral adecuado $\bar{n}$ para cada número natural n, y existe una fórmula φ de ese lenguaje tal que (para cualquier número natural) $f(x_1, \ldots, x_k) = y$ si y sólo si $\varphi(\bar{x_1}, \ldots, \bar{x_k}, \bar{y})$ es una oración deducible en la teoría $\boldsymbol{T}$.

Definibilidad en lenguajes formales VII

- Finalmente, digamos que una función parcial f de aridad k sobre $\mathbb{N}$ es *representable* si existe alguna teoría finitamente axiomatizable $\boldsymbol{T}$ en un lenguaje que tenga un numeral adecuado $\bar{n}$ para cada número natural n, y existe una fórmula φ de ese lenguaje tal que (para cualquier número natural) $f(x_1, \ldots, x_k) = y$ si y sólo si $\varphi(\bar{x_1}, \ldots, \bar{x_k}, \bar{y})$ es una oración deducible en la teoría $\boldsymbol{T}$.
- Entonces, una vez más, la clase de funciones parciales representables coincide exactamente con la clase de funciones parciales dada por las otras formalizaciones de calculabilidad descritas aquí.

La tesis de Church revisada I

- En resumen, para una función parcial f de aridad k, las siguientes condiciones son equivalentes:

La tesis de Church revisada I

- En resumen, para una función parcial f de aridad k, las siguientes condiciones son equivalentes:
  - La función f es una función parcial Turing-computable.

La tesis de Church revisada I

- En resumen, para una función parcial f de aridad k, las siguientes condiciones son equivalentes:
  - La función f es una función parcial Turing-computable.
  - La función f es una función parcial recursiva general.

La tesis de Church revisada I

- En resumen, para una función parcial f de aridad k, las siguientes condiciones son equivalentes:
  - La función f es una función parcial Turing-computable.
  - La función f es una función parcial recursiva general.
  - La función parcial f es computable-while.

La tesis de Church revisada I

- En resumen, para una función parcial f de aridad k, las siguientes condiciones son equivalentes:
  - La función f es una función parcial Turing-computable.
  - La función f es una función parcial recursiva general.
  - La función parcial f es computable-while.
  - La función parcial f se calcula mediante algún programa de máquinas de registros.

La tesis de Church revisada I

- En resumen, para una función parcial f de aridad k, las siguientes condiciones son equivalentes:
  - La función f es una función parcial Turing-computable.
  - La función f es una función parcial recursiva general.
  - La función parcial f es computable-while.
  - La función parcial f se calcula mediante algún programa de máquinas de registros.
  - La función parcial f es definible$-\lambda$.

La tesis de Church revisada I

- En resumen, para una función parcial f de aridad k, las siguientes condiciones son equivalentes:
  - La función f es una función parcial Turing-computable.
  - La función f es una función parcial recursiva general.
  - La función parcial f es computable-while.
  - La función parcial f se calcula mediante algún programa de máquinas de registros.
  - La función parcial f es definible$-\lambda$.
  - La función parcial f es definible$-\Sigma_1$ (sobre los números naturales con suma, multiplicación y exponenciación).

La tesis de Church revisada I

- En resumen, para una función parcial f de aridad k, las siguientes condiciones son equivalentes:
  - La función f es una función parcial Turing-computable.
  - La función f es una función parcial recursiva general.
  - La función parcial f es computable-while.
  - La función parcial f se calcula mediante algún programa de máquinas de registros.
  - La función parcial f es definible$-\lambda$.
  - La función parcial f es definible$-\Sigma_1$ (sobre los números naturales con suma, multiplicación y exponenciación).
  - La función parcial f es representable (en alguna teoría finitamente axiomatizable).

La tesis de Church revisada II

- ¡La equivalencia de estas condiciones es seguramente un hecho notable!

La tesis de Church revisada II

- ¡La equivalencia de estas condiciones es seguramente un hecho notable!
- Además, es evidencia de que las condiciones caracterizan alguna propiedad natural y significativa.

La tesis de Church revisada II

- ¡La equivalencia de estas condiciones es seguramente un hecho notable!
- Además, es evidencia de que las condiciones caracterizan alguna propiedad natural y significativa.
- La tesis de Church es la afirmación de que las condiciones, de hecho, capturan el concepto informal de una función efectivamente calculable.

La tesis de Church revisada II

- ¡La equivalencia de estas condiciones es seguramente un hecho notable!
- Además, es evidencia de que las condiciones caracterizan alguna propiedad natural y significativa.
- La tesis de Church es la afirmación de que las condiciones, de hecho, capturan el concepto informal de una función efectivamente calculable.

Definición 1

Una función parcial f de aridad k sobre los números naturales se dice que es una función parcial computable si se cumplen las condiciones anteriores.

La tesis de Church revisada II

- ¡La equivalencia de estas condiciones es seguramente un hecho notable!
- Además, es evidencia de que las condiciones caracterizan alguna propiedad natural y significativa.
- La tesis de Church es la afirmación de que las condiciones, de hecho, capturan el concepto informal de una función efectivamente calculable.

Definición 1

Una función parcial f de aridad k sobre los números naturales se dice que es una función parcial computable si se cumplen las condiciones anteriores.

- Entonces la tesis de Church es la afirmación de que esta definición es la que queremos.

La tesis de Church revisada III

- La situación es algo análoga a la del cálculo.

La tesis de Church revisada III

- Una función intuitivamente continua (definida en un intervalo) es aquella cuya gráfica se puede dibujar sin levantar el lápiz del papel.

La tesis de Church revisada III

- Pero para demostrar teoremas se necesita alguna contraparte formalizada de este concepto. Y así se da la definición habitual de continuidad $\epsilon - \delta$.

La tesis de Church revisada III

- Entonces es justo preguntar si el concepto preciso de continuidad $\epsilon - \delta$ es una formalización precisa de la continuidad intuitiva.

La tesis de Church revisada III

- En todo caso, la clase de funciones continuas $\epsilon - \delta$ es demasiado amplia. Incluye funciones no diferenciables en ninguna parte, cuyas gráficas no se pueden dibujar sin levantar el lápiz; no hay forma de impartir un vector de velocidad al lápiz.

La tesis de Church revisada III

- Pero sea exacta o no, se ha descubierto que la clase de funciones continuas $\epsilon - \delta$ es una clase natural e importante en el análisis matemático.

La tesis de Church revisada IV

1. La misma situación ocurre con la computabilidad.

La tesis de Church revisada IV

2. Es justo preguntarse si el concepto preciso de función parcial computable es una formalización precisa del concepto informal de función efectivamente calculable.

La tesis de Church revisada IV

3. Una vez más, la clase definida con precisión parece ser, en todo caso, demasiado amplia, porque incluye funciones que requieren, para grandes entradas, cantidades absurdas de tiempo de cálculo.

La tesis de Church revisada IV

4. La computabilidad corresponde a la calculabilidad efectiva en un mundo idealizado, donde se ignoran la duración del cálculo y la cantidad de espacio de memoria.

La tesis de Church revisada IV

5. Pero en cualquier caso, se ha descubierto que la clase de funciones parciales computables es una clase natural e importante.

Ejercicios I

1. De un programa-loop para calcular la siguiente función:

$$f(x,y,z) = \begin{cases} y & \text{si } x = 0 \\ z & \text{si } x \neq 0 \end{cases} \tag{1}$$

2. Sea $x \dot{-} y = max(x - y, 0)$, el resultado de restar y de x, pero con un "piso" de 0. Proporcione un programa loop que calcule la función de aridad dos $x \dot{-} y$.
3. Proporcione un programa loop que cuando se inicia con todas las variables asignadas a 0, se detiene con X_0 asignado algún número mayor que 1000.
4. Proporcione un programa de máquinas de registros que calcule la función de resta, $x \dot{-} y = max(x - y, 0)$, como en el ejercicio 2.

Ejercicios II

5. Proporcione un programa de máquinas de registros que calcule la función parcial de resta:

$$f(x,y) = \begin{cases} x - y & \text{si } x \geqslant y \\ \uparrow & \text{si } x < y \end{cases} \tag{2}$$

6. Proporcione un programa de máquinas de registros que calcule la función de multiplicación, $x \cdot y$.
7. Proporcione un programa de máquinas de registros que calcule la función $\max(x,y)$.
8. Proporcione un programa de máquinas de registros que calcule la función de paridad:

$$f(x) = \begin{cases} 1 & \text{si } x \text{ es impar} \\ 0 & \text{si } x \text{ es par} \end{cases} \tag{3}$$