LOS PROGRAMAS-WHILE COMO MODELO DE COMPUTACIÓN IDEAL

- Potencia de los programas-while:
  - Definición de procesos
  - Definición de datos
- NUESTRA EXPERIENCIA: Todas las características de los lenguajes de programación convencionales que nos interesan son implementables mediante el lenguaje de los programas-while
  - Excepción: mecanismos de simulación de la recursión, que no hemos tratado por su complicación
- MÁS DE MEDIO SIGLO DE EXISTENCIA DE LA INFORMÁTICA: Nunca se ha podido diseñar un computador que implemente algoritmos intraducibles al lenguaje de los programas-while
- CASI UN SIGLO DE INFORMATICA TEÓRICA: Nunca se ha encontrado un modelo teórico de computación físicamente realizable que sea más potente que los programas-while

LA TESIS DE CHURCH-TURING

Todo sistema de computación es
A LO SUMO TAN POTENTE como
el de los programas-while

- Cualquier función (problema) entre cadenas de símbolos que admita un algoritmo (solución) de cualquier tipo es while-computable.

- Se puede decir lo mismo de cualquier función entre objetos de cualesquiera tipos de datos que se puedan implementar mediante cadenas de símbolos.

- La noción de while-computabilidad no es específica de los programas-while. Todos los lenguajes, ordenadores y entornos de programación conocidos computan exactamente las mismas funciones (podrían computar menos, pero tendrían que ser muy pobres).

- Hablaremos a secas de funciones COMPUTABLES y predicados DECIDIBLES

IMPLICACIONES DE LA TESIS DE CHURCH-TURING

- La Tesis de Church-Turing vale para modelos pasados, presentes y futuros.

- No puede ser demostrada, pero podría ser refutada

- Su corolario es muy importante:

> Si una función no es while-computable, TAMPOCO SE PUEDE NI SE PODRÁ ENCONTRAR UNA SOLUCIÓN O ALGORITMO para la misma en ningún sistema de computación

EL TIPO DE DATOS PROGRAMAS-WHILE

- Nos interesa trabajar con objetos en los que los problemas sean realmente difíciles

- Definiremos el tipo de datos W con sus operaciones básicas

- ¿Dónde usaremos los programas-while?
  - Como datos (análisis)
  - Como resultado (síntesis)

- ¿Qué nos interesará de los programas-while?
  - Su sintaxis (propiedades estáticas)
  - Su semántica (propiedades dinámicas)

FUNCIÓN DE INTERPRETACIÓN PARA PROGRAMAS-WHILE

$$\mathfrak{I}_W(w) = \begin{cases} \text{XI:= } \varepsilon; & \text{si } \mathfrak{I}_N(\text{decod}_{2,1}(w)) = 0 \wedge \\ & \wedge i = \mathfrak{I}_N(\text{decod}_{2,2}(w)) \\ \\ \text{XI:= cons}_s(\text{XJ}); & \text{si } 1 \le \mathfrak{I}_N(\text{decod}_{2,1}(w)) \le n \wedge \\ & \wedge i = \mathfrak{I}_N(\text{decod}_{3,2}(w)) \wedge \\ & \wedge s = \text{decod}_{3,1}(w) \wedge \\ & \wedge j = \mathfrak{I}_N(\text{decod}_{3,3}(w)) \\ \\ \text{XI:= cdr}(\text{XJ}); & \text{si } \mathfrak{I}_N(\text{decod}_{2,1}(w)) = n+1 \wedge \\ & \wedge \mathfrak{I}_N(\text{decod}_{3,2}(w)) = i \wedge \\ & \mathfrak{I}_N(\text{decod}_{3,3}(w)) = j \\ \\ \text{P Q} & \text{si } \mathfrak{I}_N(\text{decod}_{2,1}(w)) = n+2 \wedge \\ & \wedge \mathfrak{I}_W(\text{decod}_{3,2}(w)) = P \wedge \\ & \wedge \mathfrak{I}_W(\text{decod}_{3,3}(w)) = Q \\ \\ \text{if car}_s\text{?(XI) then P end if ;} & \text{si } n+3 \le \mathfrak{I}_N(\text{decod}_{2,1}(w)) \le 2*n+2 \wedge \\ & \wedge \mathfrak{I}_N(\text{decod}_{3,2}(w)) = i \wedge \\ & \wedge \mathfrak{I}_N(\text{decod}_{3,1}(w)) - n - 2 = \mathfrak{I}_N(s) \wedge \\ & \wedge \mathfrak{I}_W(\text{decod}_{3,3}(w)) = P \\ \\ \text{while nonem? (XI) loop P end loop ;} & \text{si } \mathfrak{I}_N(\text{decod}_{2,1}(w)) \ge 2*n+3 \wedge \\ & \wedge i = \mathfrak{I}_N(\text{decod}_{2,1}(w)) - 2*n - 3 \wedge \\ & \wedge \mathfrak{I}_W(\text{decod}_{2,2}(w)) = P \end{cases}$$

PW: FUNCIONES CONSTRUCTORAS

- haz_asig_vacia: $N \rightarrow W$

  X0 := cod_2 (0, X1);

- haz_asig_cons: $N \times \Sigma^* \times N \rightarrow W$

  if |X2| <> 1 then X0 := ⊥;

  else X0 := cod_3 (X2, X1, X3); end if;

- haz_asig_cdr: $N \times N \rightarrow W$

  X0 := cod_3 (N+1, X1, X2);

- haz_composición: $W \times W \rightarrow W$

  X0 := cod_3 (N+2, X1, X2);

- haz_condición: $N \times \Sigma^* \times W \rightarrow W$

  if |X2| <> 1 then X0 := ⊥;

  else X0 := cod_3(N+2+X2, X1, X3); end if;

- haz_iteración: $N \times W \rightarrow W$

  X0 := cod_2 (2*N+3+X1, X2);

PW: PREDICADOS DE INSPECCION

- asig_vacia?: $W \rightarrow B$

  X0 := decod_2_1 (X1) = 0;

- asig_cons?: $W \rightarrow B$

  X0 := (decod_2_1 (X1) >= 1) $\wedge$ (decod_2_1 (X1) <= N);

- asig_cdr?: $W \rightarrow B$

  X0 := decod_2_1 (X1) = N+1;

- composición?: $W \rightarrow B$

  X0 := decod_2_1 (X1) = N+2;

- condición?: $W \rightarrow B$

  X0 := decod_2_1 (X1) >= N+3 $\wedge$ decod_2_1 (X1) <= 2$*$N+2;

- iteración?: $W \rightarrow B$

  X0 := decod_2_1 (X1) >= 2$*$N+3;

PW: FUNCIONES DE ACCESO I

- ind_var: $W \longrightarrow N$

```
if  decod_2_1(X1) >= 2*N+3  then
      X0 :=  decod_2_1(X1) – 2*N – 3;
elsif  decod_2_1 (X1) = N+2  then
      X0 := ⊥⊥;
elsif  decod_2_1 (X1) = 0  then
      X0 := decod_2_2 (X1);
else    X0 := decod_3_2 (X1);
end if;
```

- ind_simb: $W \longrightarrow \Sigma^*$

```
if  decod_2_1 (X1) >= N+3  and
          decod_2_1 (X1) <= 2*N+2  then
      X0 := decod_2_1 (X1) – N – 2;
elsif  decod_2_1 (X1) >= 1  and
          decod_2_1 (X1) <= N  then
      X0 := decod_2_1 (X1);
else  X0 := ⊥⊥;
end if;
```

PW: FUNCIONES DE ACCESO II

- ind_var_2: W $\longrightarrow$ N

```
if decod_2_1 (X1) >= 1 and decod_2_1 (X1) <= N+1 then
    X0 := decod_3_3 (X1);
else X0 := ⊥; end if;
```

- inst_int: W $\longrightarrow$ W

```
if decod_2_1 (X1) >= N+3 and
        decod_2_1 (X1) <= 2*N+2 then
    X0 := decod_3_3 (X1);
elsif decod_2_1 (X1) = N+2 then X0 := decod_3_2 (X1);
elsif decod_2_1 (X1) >= 2*N+3 then
    X0 := decod_2_2 (X1);
else X0 := ⊥;
end if;
```

- inst_int_2: W $\longrightarrow$ W

```
if decod_2_1 (X1) = N+2 then
    X0 := decod_3_3 (X1);
else X0 := ⊥; end if;
```

ULT_VARIABLE

```
PILA := <];  PILA := empilar(X1, PILA);  X0 := 0;
while not P_vacía?(PILA) loop
      PROG := cima(PILA);   PILA  := desempilar(PILA);
      if asig_vacía?(PROG) then
            if ind_var (PROG)>X0 then  X0 := ind_var (PROG);
            end if;
      elsif asig_cons?(PROG) or asig_cdr?(PROG) then
            if ind_var (PROG)>X0 then  X0 := ind_var (PROG);
            end if;
            if ind_var_2(PROG)>X0 then  X0 := ind_var_2(PROG);
            end if;
      elsif condición?(PROG) or iteracion?(PROG) then
            if ind_var (PROG)>X0 then  X0 := ind_var (PROG);
            end if;
            PILA := empilar(inst_int(PROG), PILA);
      else PILA := empilar(inst_int_2(PROG), PILA);
            PILA := empilar(inst_int(PROG), PILA);
      end if;
end loop;
```

CONJUNTOS DE INDICES

- VAC = $\{ x \in W: \forall y \in \Sigma^* \varphi_x(y)\uparrow \} = \{ x \in W: W_x = \emptyset \} = \{ x \in W: \varphi_x \cong \perp\!\!\!\perp \}$
- FIN = $\{ x \in W^*: W_x$ es finito$\}$
- RFIN = $\{ x \in W: R_x$ es finito$\}$
- TOT = $\{ x \in W: \forall y \in \Sigma^* \varphi_x(y)\downarrow \} = \{ x \in W: W_x = \Sigma^* \}$
- CONS = $\{ x \in W: \varphi_x$ es constante $\} = \{ x \in$ TOT: $|R_x|=1 \}$
- INJ = $\{ x \in W: \varphi_x$ es inyectiva $\}$
- SUP = $\{ x \in W: \varphi_x$ es sobreyectiva $\} = \{ x \in W: R_x = \Sigma^* \}$
- PERM = $\{ x \in W: \varphi_x$ es biyectiva$\}$
- K = $\{ x \in W: \varphi_x (x)\downarrow \}$