Las declaraciones *go to* consideradas perjudiciales

Edsger W. Dijkstra

Communications of the ACM, marzo 1968

Desde hace años estoy familiarizado con la observación de que la calidad de los programadores es una función decreciente en la densidad de declaraciones *go to* de los programas que crean. Descubrí más recientemente por qué el uso de los *go to* tiene efectos tan desastrososo y quedé convencido que las declaraciones *go to* deberían ser abolidas de todos los lenguajes de programación de "alto nivel" (es decir, de cualquier cosa excepto, quizá, código máquina). En ese momento no le di demasiada importancia a este descubrimiento; envío ahora mis apreciaciones para ser publicadas pues en discusiones muy recientes en las ha surgido esta cuestión se me ha urgido a hacerlo.

Mi primer comentario es que, a pesar que la actividad del programador termina cuando ha construido un programa correcto, el proceso que tiene lugar bajo el control del programa que ha escrito es el verdadero objeto de su actividad, pues es este proceso que debe conseguir el efecto deseado; es este proceso que en su proceso dinámico debe satisfacer las especificaciones deseadas. Ahora bien, una vez el programa ha sido creado, la 'creación' del proceso correspondiente es delegada a la máquina.

Mi segundo comentario es que nuestros poderes intelectuales están más bien dirigidos a dominar relaciones estáticas y que nuestros poderes de visualización procesos que se desarrollan en el tiempo son más bien pobres. Por este motivo (como sabios programadores conscientes de nuestras limitaciones) debemos hacer lo máximo posible para acortar el salto coneptual entre el programa estático y el proceso dinámico, para hacer la correspondencia entre programa (desarrollado en un espacio de texto) y el proceso (desarrollado en el tiempo) tan trivial como sea posible.

Consideremos ahora cómo podemos caracterizar el progreso de un proceso (puede considerar esta cuestión de una forma muy concreta: suponga que el proceso, considerado como una sucesión de acciones en el tiempo, se para después de una acción arbitraria, ¿qué datos debemos fijar de manera que podamos rehacer el proceso hasta exactamente el mismo punto?). Si el texto del programa es una pura concatenación de, digamos, declaraciones de asignación (para el propósito de esta discusión, consideradas como acciones simples) es suficiente apuntar en el texto del programa al punto entre las dos descripciones sucesivas de las acciones. (En ausencia de declaraciones *go to* me puedo permitir la ambigüedad del final de la frase anterior: si lo que describimos son las acciones, nos referimos a sucesivos en el espacio de texto; si lo que describimos es la sucesión en sí, entonces queremos decir sucesivas en el tiempo). Llamemos a dicho puntero a un lugar adecuado en el texto un "índice textual".

Si incluímos claúsulas condicionales (`if B then A`), alternativas(`if B then A1 else A2`), de elección, tal y como las introdujo C.A.R. Hoare (`case[i] of (A1, A2,···, An)`), o claúsulas condicionales como fueron introducidas por J. McCarthy (`B1` → `E1, B2` → `E2, ···, Bn` → `En`), se mantiene el hecho de que el progreso del proceso sigue estando caracterizado por un único índice textual.

En cuanto incluimos procedimientos en nuestro lenguaje debemos admitir que un único índice textual ya no es suficiente. En el caso de que un índice textual apunte al interior del cuerpo de un procedimiento, el progreso dinámico solo queda caracterizado cuando también indicamos a qué llamada al porcedimiento nos referimos. Con la inclusión de procedimientos podemos caracterizar el progreso de un proceso a través de una secuencia de índices textuales, siendo la longitud de esta secuencia igual a la profundidad dinámica de las llamadas a procedimientos.

Consideremos ahora claúsulas de repetición (como `while B repeat A` o `repeat A until B`). Desde el punto de vista lógico, estas claúsulas son superfluas, porque podemos expresar la repetición con la ayuda de procedimientos re-

cursivos. Por realismo no deseo excluirlas: por un lado, las claúsulas de repetición pueden ser implementadas fácilmente con nuestro equipo finito actual; por el otro lado, el patrón de razonamiento conocido como "inducción" permite a nuestro intelecto comprender los procesos generados por claúsulas de repetición. Con la inclusión de las claúsulas de repetición los índices textuales ya no son suficientes para describir el progreso dinámico del proceso. Pero, cada vez que este entra a una de estas claúsulas podemos asociar un "índice dinámico", que cuente inexorablemente el ordinal correspondiente a la repetición en curso. Como las claúsulas de repetición (del mismo modo que los procedimientos) pueden ser aplicados de forma anidada, vemos que el progreso de un proceso siempre puede ser caracterizado de forma única mediante una secuencia (combinada) de índices textuales y dinámicos.

La cuestión principal es que el valor de estos índices están fuera del control del programador; se generan (ya sea mediante la escritura de su programa o por la evolución dinámica del proceso) lo quiera o no. Nos proveen de unas coordenadas independientes con las que describir el progreso de un proceeso.

Por qué necesitamos estas coordenadas independientes? La razón es —y esto parece ser inherente a los procesos secuenciales— que podemos interpretar el valor de una variable solamente con respecto al progreso del proceso. Si deseamos contar el número, digámosle n, de personas en una habitación que inicialmente está vacía, lo podemos hacer incrementando n en uno cada vez que veamos a una persona entrando en la habitación. En el momento intermedio en el que hemos observado a una persona entrando en la habitación pero aún no hemos llevado a cabo el subsecuente incremento de n, ¡su valor equivale al número de gente en la habitación menos uno!

El uso desenfrenado de la declaración *go to* tiene la consecuencia inmediata que se convierte en enormemente difícil encontrar un conjunto significativo de coordenadas con el que describir el progreso del proceso. Normalmente la gente considera también el valor de algunas variables adecuadamente escogidas, pero esto no tiene sentido ¡porque es relativo al progreso del proceso que debe entenderse el significado de estas variables! Con la declaración *go to* uno puede, naturalmente, describir aún el progreso de forma única mediante un contador que cuente el número de acciones que han tenido lugar desde desde que se inició el programa (usando algún tipo de reloj normalizado). La dificultad es que tal coordenada, aunque única, es absolutamente inútil. En tal sistema de coordenadas es extremadamente complicado definir aquellos puntos del progreso donde, digamos, n equivale al número de personal menos uno.

La declaración *go to* tal y como es, es simplemente demasiado primitiva; es una invitación demasiado tentadora para convertir un programa en un lío. Uno puede ver y apreciar las claúsulas mencionadas como un freno para su uso. No afirmo que las clúsulas mencionadas sean exhausticas en el sentido que puedan satisfacer cualquier necesidad, pero cualesquiera claúsulas que se sugieran (por ejemplo, claúsulas abortivas) deben satisfacer el requisito que un sistema de coordenadas independientes al programador pueden ser mantenidas para describir el proceso de forma útil y manejable.

[*Eliminado una sección de agradecimientos y comentarios menores*]