

Dijkstra¹⁾의 mini language

(개요) Dijkstra가 정의한 작은 operational 프로그래밍 언어

1. **skip** 문장

“**skip**”은 아무런 일도 하지 않는 문장(continue)이다.

2. **abort** 문장

“**abort**”은 프로그램이 비정상적으로 동작을 멈추는 문장이다.

3. Concurrent assignment 문장

$$“x_1, x_2, \dots, x_n := E_1, E_2, \dots, E_n”$$

변수 x_1 에 식 E_1 의 값이 assign되고, 변수 x_2 에 식 E_2 의 값이 assign되고, ..., 변수 x_n 에 식 E_n 의 값이 모두 동시(concurrent)에 assign되는 문장이다.

여기서 모두 “동시에 assign된다”는 문장에 주의하여야 한다.

4. 문장의 sequencing

$$“S_1; S_2”$$

문장 S_1 이 수행된 후 문장 S_2 가 수행된다.

문장의 sequencing도 또 다른 문장이므로

$$S_1; S_2; \dots; S_n \text{와 같이 여러 문장을 sequencing 할 수 있고,}$$

이를 Statement list(SL)이라고도 부른다.

(예) Sequential assignment “ $x_1 := E_1; x_2 := E_2$ ”와 “ $x_2 := E_2; x_1 := E_1$ ”는 다르고, concurrent assignment “ $x_1, x_2 := E_1, E_2$ ”와도 다르다.

즉, $x=0 \wedge y=1$ 인 상황에서

“ $x := y; y := x$ ”를 하고나면 $x=1 \wedge y=1$ 이 되고,

“ $y := x; x := y$ ”를 하고나면 $x=0 \wedge y=0$ 이 되고,

“ $x, y := y, x$ ”를 하고나면 $x=1 \wedge y=0$ 이 된다.

5. Alternative(**if**) 문장
$$“\text{if } B_1 \rightarrow SL_1 \mid B_2 \rightarrow SL_2 \mid \dots \mid B_n \rightarrow SL_n \text{ if}”$$

“**if fi**”는 “**abort**”와 같다.

기존의 **if then**이나 **if then else**와는 좀 다르다.

$B_i \rightarrow SL_i$ 를 guarded command라고 부르고,

불 식 B_i 를 guard, statement list SL_i (guarded) command라고 부른다.

Guard B_i 가 **true** 일 경우 statement list SL_i 이 차례로 수행된다.

True인 guard가 B_i, B_j 로 둘 이상일 경우,

1) Edsger W. Dijkstra(1930-2002)

SL_i 나 SL_j 중 **아무나(non-determinism)** **한번 만** 수행이 된다.

True인 guard가 하나도 없을 경우 이는 프로그래머가 예상하지 않은 경우이므로, 프로그램은 동작은 비정상적으로 동작을 멈추어야 한다. 즉 “**abort**”와 같다²⁾.

Nondeterminism은 Dijkstra의 mini언어의 특징인데, 다음 예를 보자.

```

if  $x \geq y \rightarrow m := x$ 
  |  $y \geq x \rightarrow m := y$ 
fi

```

이 문장에서 $x=y$ 인 경우가 문제인데, 이 경우 두 개의 가드 $x \geq y$ 와 $y \geq x$ 가 모두 **true** 인데, 위에 있는 가드 $x \geq y$ 를 선택하여 $m := x$ 를 수행하여도 아래에 있는 가드 $y \geq x$ 를 선택하여 $m := y$ 를 수행하여도 되도록 프로그램을 작성하여야 한다는 것이 Dijkstra의 주장이다.

이 주장을 이해하기 위하여 이를 “**if then else**”로 바꾸어보자.

```

if  $x \geq y \rightarrow m := x$            if  $x > y \rightarrow m := x$ 
else (*  $y > x \rightarrow$  *)  $m := y$    else (*  $y \geq x \rightarrow$  *)  $m := y$ 
fi                               fi

```

이를 Dijkstra의 미니언어로 바꾸어보자

```

if  $x \geq y \rightarrow m := x$            if  $x > y \rightarrow m := x$ 
  |  $y > x \rightarrow m := y$        |  $y \geq x \rightarrow m := y$ 
fi                               fi

```

Dijkstra의 미니언어의 “**if ... fi**”구조가 기존의 “**if then**”이나 “**if then else**”보다는 더 일반적이라는 예이다. 위의 “**if then else**”의 어떤 구현도 두 수 x, y 의 최댓값의 정확한 수학적 의미를 구현하고 있지 않다고 Dijkstra는 보고 있다.

6. Repetitive(**do**) 문장

“**do** $B_1 \rightarrow SL_1 \mid B_2 \rightarrow SL_2 \mid \dots \mid B_n \rightarrow SL_n$ **od**”

“**do od**”는 “**skip**”과 같다.

반복구조(loop) “**do ... od**”는 **true**인 가드 B_i 에 대응하는 statement list SL_i 이 수행되고 이 작업은 **true**인 가드가 하나도 없을 때까지 계속 반복(loop)된다. 물론 **true**인 가드가 두 개 이상이면 **true**인 가드중 임의의 SL 이 수행되는 것은 위의 “**if ... fi**”구조와 같고, loop 시작부터 **true**인 가드가 하나도 없으면, 기존의 “**while**”구조와 마찬가지로, “**do ... od**”의 body는 하나도 수행되지 않고 loop를 끝낸다.³⁾

2) “**if fi**”를 “**abort**”와 같은 뜻으로 정의하였다.

3) “**do od**”를 “**skip**”과 같은 뜻으로 정의하였다.

기존의 “while”구조와 “do … od”구조의 가장 큰 차이는 가드가 하나가 아니고 여러 개가 있을 수 있다는 점이다. 아래 “do … od”문장을 살펴보자.

```

u, v, x, y, z := U, V, X, Y, Z;
do u > v → u, v := v, u
  | v > x → v, x := x, v
  | x > y → x, y := y, x
  | y > z → y, z := z, y
od

```

True인 guard가 , B_j 로 둘 이상일 경우,

SL_i 나 SL_j 중 **아무나**(non-determinism) **한번 만** 수행이 된다.

True인 guard가 하나도 없을 경우 이는 프로그래머가 예상하지 않은 경우이므로, 프로그램은 동작은 비정상적으로 동작을 멈추어야 한다.

즉 “**abort**”와 같다. “**if fi**”가 “**abort**”와 같은 이유이기도하다.

(문장) **skip** | **do od**

abort | **if fi**

$x_1, x_2, \dots, x_n := E_1, E_2, \dots, E_n$

$S_1; S_2$

if $B_1 \rightarrow SL_1$ | $B_2 \rightarrow SL_2$ | \dots | $B_n \rightarrow SL_n$ **if**

do $B_1 \rightarrow SL_1$ | $B_2 \rightarrow SL_2$ | \dots | $B_n \rightarrow SL_n$ **od**