프로그래밍의 수련(修練)
(A Discipline of Programming)

엣져 위베 다익스트라
(Edsger Wybe Dijkstra)

세번째: 의미의 규정

해역(解譯)
김도형
성신여자대학교 컴퓨터정보학부
dkim@cs.sungshin.ac.kr
URL: http://cs.sungshin.ac.kr/∼dkim

3장. 의미(意味)의 규정(規定)

우리는, '초기 상태'에서 출발하면 대체로 그 선택된 초기 상태에 의해 결정되는 '최종 상태'에서 정지할 체계들에 주로 관심이 있다. 이것은, 한편으로는 입력 문자들의 열(列)을 소진해 가면서 출력 문자들의 열을 만들어 내는 유한 상태 자동장치(有限狀態自動裝置; finite state automaton)와는 다소 다른 관점이다.[1] 그것을[2] 우리의 관점으로 보자면, 입력(즉 매개 변수)의 값은 초기 상태의 선택에, 그리고 출력(즉 답)의 값은 최종 상태의 선택에 반영된다고 가정해야 한다. 이러한 관점 덕분에 우리들은 모든 종류의 주변의 복잡함을 피할 수 있다.

이 장의 첫 부분은 소위 '결정적(決定的) 기계'만을 거의 전적으로 다룰 것이고, 반면 두번째 부분(처음 이 책을 볼 때는 생략할 수도 있다)에서는 이른바 '비결정적(非決定的) 기계'를 다룬다. 이 두 가지의 차이는 결정적 기계의 경우 그 메커니즘이 활성화(活性化)된 뒤 일어나는 일들은 전적으로 그것의 초기 상태에 의해 결정된다는 것이다. (따라서) 동일한 초기 상태에서 두 번 활성화되면, 동일한 일들이 일어날 것이다. 즉 결정적 기계의 동작은 완전히 재현(再現)할 수 있는 것이다. 이것은 비결정적 기계와는 대조적인데, 비결정적 기계가 어떤 초기 상태에서 활성화되면 가능한 일들의 집합 중 하나가 일어난다. 초기 상태는 단지 그 집합을 결정할 따름이다.

이제 나는 그러한 체계의 설계는 목표-지향적(目標指向的; goal-directed) 작업이라고, 달리 표현하자면 우리는 그 체계를 가지고 무슨 일인가를 하고 싶어한다고 상정(想定)한다.[3] 예를 들어서 우리가 최대 공약수를 계산할 수 있는 기계를 만들려고 한다면, 그러한 기계의 최종 상태는 다음 조건

$$x = \mathrm{GCD}(X, Y) \tag{1}$$

[1] 이러한 형태의 유한 상태 자동 기계는 상당히 일반화된 형태이다. 독자가 알고 있는 것은 이것보다는 다소 제한이 가해진 형태일 가능성도 많다. 그러나 중요한 차이는 아니다.

[2] 유한 상태 자동장치를 말한다.

[3] 이 서술은 어떻게 보면 너무나 당연하게 들리기 때문에 전혀 새로운 정보를 주지 않는, 불필요한 문장이라는 생각이 들지도 모르겠다. 그러나 계속 읽어 보면 그렇게 무의미하기만 한 것이 아니라는 점을 알 수 있을 것이다. 물론 그렇다고 이 문장 단독으로 매우 심각한 뜻을 가지고 있는 것은 아니나, 이 책의 본격적인 얘기를 시작하기 위한 첫 선언(宣言)으로서의, 출발로서의 의미를 가지고 있다.

을 만족해야 한다고 요구할 수 있을 것이다. 우리가 지금까지 계속 관찰해 온 기계에서는 $x = y$일 때 게임이 끝나니까 $y = \mathrm{GCD}(X, Y)$도 또한 성립하겠으나, x의 최종 값을 '답'으로 받아들이기로 결정했다면 그 조건은 우리의 요구사항에 포함되지 **않는다**.[4]

우리는 조건 (1)을 (원하는) '사후 조건(事後條件; post-condition)'이라고 부른다. 왜 '사후'인가 하면, 그 체계가 작동이 끝난 **후의** 자신이 도달한 상태에 부과되어 있는 조건이기 때문이다. 그 사후 조건이 많은 가능한 상태들에서 충족될 수도 있음을 주의하라. 그럴 경우 우리는 그러한 상태 각각을 똑같이 만족스러운 것으로 분명히 간주할 것이고, 따라서 최종 상태가 초기 상태의 유일 함수(唯一函數; unique function)여야 한다고 요구할 이유는 없다. (독자들이 알게 되겠지만, 비결정적 메커니즘의 잠재적 유용성이 드러나는 것이 이 점이다.)

그러한 체계가 답을 만들어 내기를, 즉 예컨대 '주어진 값들 X와 Y에 대해 사후 조건 (1)을 만족하는 최종 상태에 도달하기를' 원해서 그 체계를 사용하려면, 상응하는 초기 상태들의 집합, 좀 더 정확하게 표현하자면, 그 체계를 활성화시키면 사후 조건을 만족하는 최종 상태에 도달한 뒤 적절하게 정지하는 일이 확실하게 일어나도록 하는 초기 상태들의 집합을 알아야만 할 것이다. 만약 우리가 계산의 수고가 없이 이러한 (초기) 상태들 중의 하나로 그 체계를 끌고 갈 수 있다면, 우리가 원하는 답을 만들어 내도록 그것을 사용하는 방법을 아는 셈이다! 유클리드 알고리즘을 위한 판지 게임에서의 예를 생각해 보기로 한다: 다음 조건

$$\mathrm{GCD}(x, y) = \mathrm{GCD}(X, Y) \textbf{ and } 0 < x \leq 500 \textbf{ and } 0 < y \leq 500 \qquad (2)$$

을 만족하는 어떤 초기 상태에 대해서도 사후 조건 (1)을 만족하는 최종 상태를 보장할 수 있다.[5] (상한선(500이라는)은 판지가 유한한 크기라는 점을 표현하기 위해 추가되었다. 만약 우리가 $\mathrm{GCD}(X, Y) = 713$인 (X, Y) 쌍에서 출발했다면, 조건 (2)를 만족하는 (x, y) 쌍은 없다. 즉, 그러한 X와 Y의 값에 대해서는 조건 (2)가 F로 귀착된다.[6] 그리고 이 사실은 그러한 X와 Y의 값들

[4] 이 문제에서는 포함시켜도 상관없으나, 다른 경우에는 그러한 꼭 필요하지 않은 조건의 추가가 메커니즘의 구성을(혹은 달리 표현하자면, 문제에 대한 알고리즘의 설계를) 복잡하게 하거나 오도할 가능성도 있다. 따라서 필요없는 것은 포함시키지 않으면 된다! 그러나 이 문제에서 x의 최종 값을 '답'으로 하기로 했다면, 최종 상태가 만족해야만 하는 조건의 서술에서 x는 꼭 포함되어 있어야 할 것이다.

[5] 앞에서 본 판지 기계를 사용하여 조건 (2)를 만족하는 임의의 점에서 그 메커니즘을 활성화시키면(곧, 출발시키면 또는 게임을 시작하면), 유한한 시간 내에 그 메커니즘은 멈추고(즉, 적절하게 정지하고) 그 멈춘 점에서는 사후 조건 (2)가 만족된다는 뜻이다. 이것은 생각해 보면 쉽게 알 수 있고, 이미 0장에서 그 메커니즘의 정합성을 언급한 바가 있다.

[6] $\mathrm{GCD}(X, Y) = 713$인 자연수 X와 Y의 쌍은 당연히 무수히 많다. 그러나 최대 공약수의

쌍에 대해서는 $\mathrm{GCD}(X, Y)$를 그 기계가 계산할 수 없음을 의미한다.)

많은 (X, Y) 조합에 대해서, 많은 상태들이 (2)를 만족한다. $0 < X \leq 500$이고 $0 < Y \leq 500$인 경우, ((2)를 만족시키는) **뻔한**[7] 선택은 $x = X$와 $y = Y$이다. 그것은 GCD-함수에 대한 어떤 계산도, 심지어 GCD-함수가 그 매개 변수들의 대칭 함수(對稱函數; symmetric function)라는[8] 사실조차 이용할 필요없이 할 수 있는 선택이다.

활성화된 결과, 주어진 사후 조건을 만족하는 최종 상태에 체계를 도달시킨 뒤 적절하게 정지하는 일이 확실하게 일어나도록 하는 **모든** 초기 상태들의 집합을 규정하는 조건을 '그 사후 조건에 대응하는 최약 사전 조건(最弱事前條件; weakest pre-condition)'이라고 부른다.[9] (우리가 그것을 '최약'이라고 부르는 이유는 조건이 약할수록 보다 많은 상태가 그 조건을 만족할 터이고, 우리는 여기서 원하는 최종 상태로 이끄는 것이 확실한 **모든** 가능한 시작 상태를 규정하는 것을 목적으로 하기 때문이다.)

만약 하나의 체계(기계, 메커니즘)를 'S'로 나타내고 원하는 사후 조건을 'R'로 나타낸다면, 해당하는 최약 사전 조건은

$$\mathrm{wp}(S, R)$$

로 나타낸다. 만약 초기 상태가 $\mathrm{wp}(S, R)$을 만족한다면, 그 체계는 종국(終局)에 가서는 R이 참임을 틀림없이 확립시킨다. $\mathrm{wp}(S, R)$은 최약 사전 조건이므로, 만약 초기 상태가 $\mathrm{wp}(S, R)$을 만족하지 않는다면 그러한 보장은[10] 할 수 없다. 즉 R을 만족하지 않는 최종 상태에서 끝나거나, 심지어는 최종 상태란 것에 도달조차 하지 못하는 일(독자들이 보게 되겠지만, 체계가 끝이 없는 작업에 말려 들거나 혹은 옴짝달싹하지 못하게 되거나 하여)이 일어날 수도 있다.

임의의 사후 조건 R에 대해 상응하는 최약 사전 조건 $\mathrm{wp}(S, R)$을 유도할 수 있다면, 우리는 그 메커니즘 S의 가능한 수행에 관해 충분히 잘 안다는 관점을 취한다. 왜냐하면, 그럴 경우 그 메커니즘이 우리를 위해 무얼 해 줄 수 있는가—전문 용어를 쓰자면 '그것의 의미(意味; semantics)'—를 파악한 셈이기 때문이다.

두 가지를 언급하는 것이 적절할 성 싶다. 첫째로 가능한 사후 조건들의 집합은 일반적으로 너무 거대(巨大)하기 때문에, 이 정보를 표(表) 형태(즉, 각

성질에 따라, $\mathrm{GCD}(x, y) = \mathrm{GCD}(X, Y)$가 성립할려면(조건 (2)의 첫 부분을 보라) 최소한 x, y, X, 그리고 Y는 모두 713보다는 크거나 같아야 할 것이다. 그러나 이렇게 되면 조건 (2)의 뒷 부분을 만족할 수가 없다.

[7] 원문에 있는 '*trivial*'의 번역이다.

[8] $\mathrm{GCD}(X, Y) = \mathrm{GCD}(Y, X)$인 것을 말한다.

[9] 이 문장은 이 장에서 상당히 중요한 정의를 담고 있는 부분이다. 다시 한 번 음미하면서 읽을 것을 권한다.

[10] 앞 문장에서 나온, 사후 조건 R을 확립시킨다는 보장을 말한다.

항목이 R과 그것에 대응하는 $\mathrm{wp}(S,R)$으로 되어 있는)로 나타낸다는 것은 도저히 다루기 어려우며, 따라서 쓸모가 없다. 그러므로 어떤 메커니즘의 의미 정의는 늘상 다른 방식, 즉 임의(任意)로 주어진 사후 조건 R에 대해 대응하는 최약 사전 조건 $\mathrm{wp}(S,R)$을 어떻게 유도할 것인가를 설명하는 규칙(規則)의 형태로 제시된다. 하나의 고정된 메커니즘 S에 대해서, 사후 조건을 나타내는 술어 R을 제공하면 해당하는 최약 사전 조건을 나타내는 $\mathrm{wp}(S,R)$을 내보내는, 그러한 규칙을 '술어 변환자(述語變換子)'라고 부른다. 우리가 어떤 메커니즘 S의 의미 정의(定義)를 요구할 때, 우리가 진짜 원하는 것은 그것의 술어 변환자이다.

두번째로—그리고 '고맙게도!'라고 덧붙이고 싶은데—우리는 종종 한 메커니즘의 완전한 의미에 대해서만 관심이 있는 것은 아니다. 이것은 그 메커니즘 S를 특정한 목적으로만 사용하는 것, 즉 그러한 점을 염두에 두고 설계된 메커니즘에서 매우 특별한 사후 조건 R의 참을 확립시키는 것이 우리 의도일 수도 있기 때문이다. 게다가 그 특정한 사후 조건 R에 대해서 조차, 우리는 종종 $\mathrm{wp}(S,R)$의 정확한 형태에 대해서는 관심이 없고, 다음 식

$$\text{임의의 모든 상태에 대해서, } P \Rightarrow \mathrm{wp}(S,R) \tag{3}$$

이 성립함을 증명할 수 있는 보다 강한 조건 P로 만족하기도 한다. ('$P \Rightarrow Q$'란 술어('P는 Q를 함축(含蓄)한다(imply)'라고 읽는다)는 상태 공간에서 P는 성립하나 Q는 성립하지 않는 점들에서만 거짓이고, 다른 모든 점들에서는 참이다. '$P \Rightarrow \mathrm{wp}(S,R)$'이 모든 상태에서 성립한다는 것은 곧 P가 참인 점에서는 $\mathrm{wp}(S,R)$도 그러해야 한다는 말이다. P는 충분[11] 사전 조건(充分事前條件; sufficient pre-condition)인 것이다. 집합으로 얘기하자면, P에 의해 규정되는 상태들의 집합이 $\mathrm{wp}(S,\ R)$에 의해 규정되는 상태들의 집합의 부분 집합임을 의미한다.) 만약 주어진 P, S, R에 대해서 (3)이라는 관계가 성립한다면, 그것은 $\mathrm{wp}(S,R)$이라는 술어를 명시적으로 구성—혹은 이 단어를 더 좋아한다면 '계산' 또는 '유도'—하지 않고도 종종 증명할 수가 있다. 이것은 좋은 일이다. 왜냐하면 뻔한 경우를 제외하고는 $\mathrm{wp}(S,R)$을 명시적으로 구성한다는 일은 적어도 우리의 종이 크기나[12] 참을성, 또는 (분석적) 창의력(혹은 이런 것들의 혼합) 같은 것들에 대한 도전이 될 것이기 때문이다.

$\mathrm{wp}(S,R)$은 '활성화시키면 사후 조건 R을 만족하는 최종 상태에 체계 S를 도달하게 한 후 적절하게 종료하는 일이 확실하게 일어나도록 할 초기 상태의 최약 사전 조건'이란 의미를 가지고 있기에, 술어 변환자는 사후 조건 R의 함수로 간주한다면 여러 가지 성질을 지니고 있다.

성질 1. 임의의 메커니즘 S에 대해서,

[11] '최약'이 아니고.

[12] 유도 과정이 너무나 길어지는 경우가 많다는 뜻이다.

$$\mathrm{wp}(S, F) = F \tag{4}$$

만약 이것이 참이 아니라고 가정하자. 그렇다면 $\mathrm{wp}(S, F)$를 만족하는 상태가 적어도 하나는 존재할 것이다. 그러한 상태를 메커니즘 S를 위한 초기 상태로 잡자. 그럴 경우 우리의 정의에 의해, 활성화시키면 그 체계 S를 F가 만족되는 최종 상태에 남겨 두고 적절하게 종료하는 일이 벌어질 것이다. 그러나 이것은 모순이다. 왜냐하면 정의에 의해 F를 만족하는 상태는 존재하지 않기 때문이다. 이렇게 해서 관계 (4)는 증명된다. 성질 1은 '기적 배제의 법칙(Law of the Excluded Miracle)'이라는 이름으로도 알려져 있다.[13]

성질 2. 임의의 메커니즘 S와

$$\text{임의의 모든 상태에 대해서, } Q \Rightarrow R \tag{5}$$

인 임의의 사후 조건 Q와 R에 대해서, 다음 관계도 역시 성립한다.

$$\text{임의의 모든 상태에 대해서, } \mathrm{wp}(S, Q) \Rightarrow \mathrm{wp}(S, R) \tag{6}$$

$\mathrm{wp}(S, Q)$를 만족하는 임의의 초기 상태에서 활성화되면, 정의에 의해서 Q의 참을 실제로 확립시킬 것이다. 또한 (5)에 의해 R의 참도 따라서 확립될 것이고, 그러한 ($\mathrm{wp}(S.Q)$를 만족하는) 초기 상태는 $\mathrm{wp}(S, R)$을 만족하게 된다. (6)에서 표현된 것이 그것이다. 성질 2는 단조성(單調性; property of monotonicity)이라고 한다.

성질 3. 임의의 메커니즘 S와 임의의 사후 조건 Q, R에 대해서, 다음 관계가 성립한다.

$$\text{모든 상태에 대해서, } (\mathrm{wp}(S, Q) \textbf{ and } \mathrm{wp}(S, R)) = \mathrm{wp}(S, Q \textbf{ and } R) \tag{7}$$

[13] 요컨대 F를 만족하는 상태로 어떤 체계가 진입(進入)하거나 머물러 있을 수는 없다. 그러한 조건을 만족하는 상태가 아예 없기 때문에, '기적은 없다'고 말하는 것이다. (뒤에서 예를 많이 보겠지만) 어떤 체계가 갖추어야 할 조건을 서술 혹은 유도하는 과정에서, 그 체계가 도달해야 될 상태가 복잡한 술어로 서술되었는데 그 술어나 조건이 F로 귀결될 수가 있다. 그럴 경우 이 성질이 얘기하고 있듯이, 그 체계의 그 사후 조건에 대한 최약 사전 조건은 F이다. 달리 얘기하면, 그러한 체계는 만들거나 존재할 수가 없는 것이다.

상태 공간의 모든 점에서 (7)의 좌변은 우변을 함축한다. 왜냐하면 $\mathrm{wp}(S, Q)$와 $\mathrm{wp}(S, R)$을 모두 만족하는 어떤 초기 상태이든지, Q와 R을 함께 만족하는 최종 상태를 확립할 것이라는 것을 우리는 알고 있기 때문이다. 게다가 정의에 의해서

$$\text{모든 상태에 대해서, } (Q \textbf{ and } R) \Rightarrow Q$$

이기 때문에, 성질 2를 이용하면

$$\text{모든 상태에 대해서, } \mathrm{wp}(S, Q \textbf{ and } R) \Rightarrow \mathrm{wp}(S, Q)$$

이고 마찬가지로

$$\text{모든 상태에 대해서, } \mathrm{wp}(S, Q \textbf{ and } R) \Rightarrow \mathrm{wp}(S, R)$$

임을 결론내릴 수 있다. 헌데 $A \Rightarrow B$이고 $A \Rightarrow C$로부터 명제 연산(命題演算; propositional calculus)에 따르면 $A \Rightarrow (B \textbf{ and } C)$라는 것을 알 수 있다. 따라서 (7)의 우변도 상태 공간의 모든 점에서 좌변을 함축한다. 양변이 서로를 모든 점에서 함축하므로, 그들은 동치(同値)여야만 하고 성질 3은 증명된다.

성질 4. 임의의 메커니즘 S와 임의의 사후 조건 Q, R에 대해서, 다음 관계가 성립한다.

$$\text{모든 상태에 대해서, } (\mathrm{wp}(S, Q) \textbf{ or } \mathrm{wp}(S, R)) \Rightarrow \mathrm{wp}(S, Q \textbf{ or } R) \qquad (8)$$

정의에 의해서

$$\text{모든 상태에 대해서, } Q \Rightarrow (Q \textbf{ or } R)$$

이기 때문에, 성질 2에 따르면

$$\text{모든 상태에 대해서, } \mathrm{wp}(S, Q) \Rightarrow \mathrm{wp}(S, Q \textbf{ or } R)$$

이고 마찬가지로

$$\text{모든 상태에 대해서, } \mathrm{wp}(S,R) \Rightarrow \mathrm{wp}(S, Q \textbf{ or } R)$$

임을 결론내릴 수 있다. 헌데 $A \Rightarrow C$이고 $B \Rightarrow C$라는 것으로부터, 명제 연산을 이용하면 $(A \textbf{ or } B) \Rightarrow C$가 되므로 (8)은 증명된다. 일반적으로 반대 방향으로의 함축은 성립하지 않는다: 임산부가 아들을 낳는다는 확실성(確實性)은 전혀 없고, 마찬가지로 딸을 낳을 확실성도 전혀 없으나, 아들 혹은 딸을 낳을 확실성은 절대적(絶對的)이다. 그러나 결정적 메커니즘에 대해서는 다음과 같은 보다 강한 성질이 있다.

성질 4′. 임의의 결정적 메커니즘 S와 임의의 사후 조건 Q, R에 대해서, 다음 관계가 성립한다.

$$\text{모든 상태에 대해서, } (\mathrm{wp}(S,Q) \textbf{ or } \mathrm{wp}(S,R)) = \mathrm{wp}(S, Q \textbf{ or } R)$$

우리는 우변이 좌변을 함축함을 보여야 한다.[14] $\mathrm{wp}(S,\ Q \textbf{ or } R)$을 만족하는 어떤 초기 상태를 생각해 보자. 이 초기 상태에 대해서는 Q나 R 혹은 둘 다를 동시에 만족하는 **유일한**[15] 최종 상태가 대응된다. 따라서 이들 경우 각각에 대해서 그 초기 상태는 $\mathrm{wp}(S,Q)$나 $\mathrm{wp}(S,R)$ 혹은 둘 다를 만족해야만 한다. 곧 $(\mathrm{wp}(S,Q) \textbf{ or } \mathrm{wp}(S,R))$을 만족해야만 하는 것이다. 이로써 성질 $4'$이 증명된다.

이 책에서—이 점은 책의 특색 중의 하나랄지도 모르겠는데—나는 비결정성을 일반적(一般的)인 것으로, 결정성을 예외로 취급할 것이다. 그래서 결정적 기계는 비결정적 기계의 특별한 경우로 간주되어, 보다 약한 성질 4가 아닌 성질 $4'$이[16] 성립하는 메커니즘으로 취급될 것이다. 이러한 결정은 나 자신의 생각에 일어난 크나큰 변화를 반영한다. 1958년으로 돌아가 보면, 나는 입출력 인터럽트(I/O interrupt) 기능을 가진 기계의 기본 소프트웨어를 개발한 최초 인력 중의 한 명이었는데, 그러한—어느 점으로 보아도 비결정적인—기

[14] 이미 좌변이 우변을 함축한다는 것은 앞의 (8)에서 보였기 때문에, 우변이 좌변을 함축한다는 것을 추가로 보이면 좌변과 우변은 동치라는 것을 증명하게 된다. 요약하면, 임의의 (결정적이든 비결정적이든) 메커니즘에 대해서 (8)의 좌변은 우변을 함축한다. 임의의 결정적 메커니즘의 경우는 (이제 증명을 하겠지만) 추가로 우변이 좌변을 함축하기도 한다. 임의의 비결정적 메커니즘의 경우는 이 추가의 보장을 할 수 없는 것이다.

[15] S가 결정적 메커니즘이기 때문에 '유일한' 최종 상태가 대응함을 강조하는 것이다.

[16] 즉 성질 $4'$은 성질 4보다 강한 혹은 까다로운 관계이다. 따라서 그러한 까다로운 조건이 적용될 수 있는 메커니즘들의 범위는 좁아질 것이다.

계의 재현 불가능한 동작을 경험하는 것은 너무나 괴로웠다.[17] 입출력 인터럽트라는 착상이 처음 제시되었을 때, 나는 그런 도무지 처치 곤란의 것을[18] 대상으로 신뢰성(信賴性)있는 소프트웨어를 작성해야만 한다는 생각에 소름이 끼쳐서, 그 기능을[19] 포함시켜야 하는지의 결정을 적어도 석 달은 미루었었다. 그리고 내가 굴복(屈服)하고 나서도(내가 그 저항(抵抗)을 포기했을 때 나는 칭찬을 받았었다![20]) 나는 몹시 마음이 편치 않았다. 그 시제품(試製品)이 어느 정도 동작함에 즈음하여 내가 그토록 두려워 하던 것이 확인되었다: 프로그램 내에 숨어 있는 오류(bug)는 기계의 재현 불가능한 문제점을 매우 강력하게 암시(暗示)하는 듯한 변덕스런 동작을 일으킬 수 있었다.[21] 그리고 두번째로는[22]—당시는 결정적 기계에 대해서는 '디버깅'의 효용을 여전히 믿고 있던 때인데—프로그램 검사는 그 (프로그램의) 신뢰성을 높이는 수단으로는 전혀 효과적이지 못하다는 점이 처음부터 명명백백했다는 것이다.[23]

그 이후 오랫동안 나는 비결정적 기계에서의 동작의 재현 불가능성을, 가능한 한 피해야만 하는 추가의 번거로움으로 간주해 왔다. 인터럽트란 하드웨어 공학자들이 불쌍한 소프트웨어 제작자에게 가한 저주(咀呪)에 다름아니었다! 이러한 나 자신의 두려움으로부터, '조화(調和)롭게 협력(協力)하는 순차(順次) 프로세스들(harmoniously cooperating sequential processes)'을[24] 위한 규칙이 만들어졌다. 그것의 성공에도 불구하고, 나는 여전히 염려스러웠

[17] 원문은 "· · · the irreproducibility of the behaviour of such a · · · machine was a traumatic experience."로 되어 있다.

[18] 입출력 인터럽트 기능을 가진 컴퓨터를 말한다. 그러한 컴퓨터의 동작은 근본적으로 비결정적이기 때문에, 당시 다익스트라의 생각으로는 도대체 다룰 수 없는 존재라고 생각한 것이다. 뒤에서 저자도 지적하는 바이지만, 우리가 스스로 작성한 프로그램의 디버깅(debugging)을 하는 때를 생각해 보자. 우선은 그러한 일이 가능한 근거는 그 프로그램의 동작이 항상 재현 가능하다는 점에 있다. 만약 그렇지 않고 수행할 때마다 동일한 프로그램이 상이한 동작을 보인다면, (일반적으로) 디버깅이라는 작업은 전혀 가능할 성 싶지 않다. 원문에는 'such an intractable beast'로 되어 있다.

[19] 입출력 인터럽트 기능을 말한다.

[20] 이 괄호 안의 문장의 원문은 "I had been flattered out of my resistance!"이다. 결국 입출력 인터럽트 기능을 포함시키게 되었다는 뜻이다.

[21] 앞서도 얘기했듯이 비결정적인 기계의 동작은 재현 불가능하기 때문에, 만약 그것을 수행하는 프로그램 내에 숨은 오류가 있다고 한다면 그 오류는 경우에 따라 드러나기도 하고 아무런 문제를 일으키지 않을 수도 있다. 그러한 변덕스런 수행은 바로 비결정성으로 인한 재현 불가능의 오류를 암시하는 것이다.

[22] 비결정성이 포함된 메커니즘의 재현 불가능성과 관련한 문제점이 첫번째이고.

[23] 이것은 다익스트라가 한 유명한 얘기들 중의 하나이다(조금 뒤에 또 나온다). 그는 "프로그램 검사는 오류가 있음을 증명하는 데는 유용하지만, **오류가 없음을 증명**하기에는 무용하다."는 요지의 말을 했었다. 사실 이 말은 원칙적으로 볼 때 절대적으로 옳은 주장이다. 그가 보기에는 결정적 기계에 대해서도 프로그램 검사는 신뢰성 향상의 수단으로 무력(無力)한 것으로 비치는데, 하물며 비결정적 기계를 위해서는 전혀 쓸모가 없다는 것이다.

[24] 독자들은 아마도 이것과 비슷한 이름을 들어본 적이 있을 것이다. 그것은 십중팔구 호어의 '통신하는 순차 프로세스들(communicating sequential processes)'이다. 아니면, 밀너(R. Milner)의 '통신하는 시스팀들의 연산(calculus of communicationg systems)'일 수도 있다. 모두가 다 컴퓨터의 수행을 형식적으로 모형화(模型化)하기 위해 제안된 틀들이다.

다. 왜냐하면 나의 해결책은—비록 맞다는 것이 증명되었지만—비결정성 중의 특별한 형태를 '길들인다는'(이것이 우리가 그것에 대해 느꼈던 식이다![25]) 문제에 대한 특수한(ad hoc) 해결책으로 보였기 때문이다.[26] 나의 두려움 배후(背後)에 있는 것은 일반적인 방법론(方法論)이 부재(不在)하다는 것이었다.

그 때 이래, 두 가지 사실이 상황을 변화시켰다. 한 가지는, 설사 전적으로 결정적인 기계의 경우라 하더라도 프로그램 검사는 거의 도움이 되지 않는다는 통찰(洞察)이다. 내가 이제는 여러번 얘기했고 또 여러 지면(紙面)에 썼다시피, 프로그램 검사는 숨어 있는 오류가 있음을 보이는 데는 매우 효과적(效果的)일 수 있으나, 없음을 증명하기 위해서는 어떻게 할 도리가 없을 정도로 부적절하다. 다른 한 가지는, 목적을 가진 하나의 메커니즘을 설계하는 일은 목표-지향적인 작업일 수 밖에 없다는 사실을 어떤 설계 규칙이든지 정당하게 취급해야 한다는 점이 그럭저럭하는 동안 드러났음을 발견한 것이다.[27] 우리의 특별한 경우에 이 얘기를 적용하면, 사후 조건이 우리가 설계시 고려해야 할 사항들의 시작점이 될 것이라는 뜻이다. 어떤 뜻에서 우리는 '거꾸로 작업을 할' 것이다. 그러기 위해서는 성질 4의 함축이 필수적인 부분이라는 것을 알게 될 것이다. 반면 성질 4′의 동치는 거의 쓸모가 없을 것이다.[28]

하나의 이루려는 목적을 가진 비결정적 메커니즘의 설계를 위해 필요한 수학적 장치가 일단 갖추어지면, 비결정적 기계는 더 이상 두렵지 않다. 오히려 그 반대다! 우리는 그것을 고마와 할 것이고, 심지어는 궁극적으로 완전히 결정적인 메커니즘을 설계하기 위한 가치있는 디딤돌로 인식하게 될 것이다.

---

[25]당시의 생각으로는 비결정성이란 마치 다루기 어려운 야수(野獸)처럼 '길들여야 할' 대상이었다는 뜻이다.

[26]비결정성을 포함하고 있는 컴퓨터 체계의 동작을 설명하기 위해 저자가 제안한 '조화롭게 협력하는 순차 프로세스들'이, 일반적인 해결책이 아니라 특수한 경우를 위한 부분 해법 혹은 미봉책(彌縫策)으로 느껴졌다는 뜻이다.

[27]번역이 별로 매끄럽지 못하다는 생각이 드는데, 요컨대 설계의 대상인 모든 메커니즘은 모름지기 어떤 목표를 가지고 있으므로, 따라서 설계 방법이라는 것은 그 점을 제대로 인식하고, 반영하고, 이용해야 한다는 사실이 많은 시행착오를 거치면서 이제는 분명해졌음을 알아차리게 되었다는 뜻이다. 참고로 원문은 다음과 같다: "The other one is the discovery that in the meantime it has emerged that any design discipline must do justice to the fact that the design of a mechanism that is to have a purpose must be a goal-directed activity."

[28]뒤에 실례(實例)를 보면 이 서술을 더 잘 이해하게 되겠지만, 약간의 직관적이고 일반적인 설명을 덧붙이면 다음과 같다: 동치는 단어 뜻 그대로 등호의 좌변과 우변이 같다는 것이다. 본질적인 차이는 없이 다만 겉모습이 바뀐 것일 뿐이라고 할 수 있다. 우리가 어떤 사후 조건으로부터 '거꾸로 작업을 하여' 사전 조건을 구하려고 한다고 가정하자. 만약 그 작업이 여의치 않다고 할 때, 성질 4′을 이용한 '동치의 연속'은 별 도움이 되지 못할 가능성이 농후하다. 오히려 성질 4의 함축이 긴요하게 쓰일 것이다. 비록 더 까다로운(stronger) 조건이라고 하더라도 말이다. 어떻게 '더 까다로운 조건'(대응되는 진리 집합의 크기가 더 작은 조건)이 '덜 까다로운 조건'(대응되는 진리 집합의 크기가 더 큰 조건)에 비해 구하기가 쉬울 수 있겠냐고 생각할 수도 있다. 그러나 그런 상황도 있을 수 있다. 비유를 하자면, 평면 상에서 부정형(不定形)의 도형보다 기하학적인 도형을 식으로 규정하기가 더 쉬운 것과 유사하다. 비록 전자(前者)가 후자(後者)보다 면적이 더 넓을 지라도 말이다.

(이 장의 나머지 부분은 처음 볼 때는 생략할 수도 있다.) 어떤 메커니즘 S의 술어 변환자 $\mathrm{wp}(S, R)$을 안다면 그 메커니즘의 가능한 동작에 대해 아는 셈이라는 우리들의 입장을 서술했다. 덧붙여 그 메커니즘이 결정적이기도 하다면 술어 변환자를 앎으로써 그것의 가능한 동작은 완전히 고정(固定)된다. 결정적 메커니즘 S와 어떤 사후 조건 R에 대해서, 각 초기 상태는 아래에 있는 것과 같은 상호 배제적인(mutually exclusive) 가능성에 따라 세 가지 배타적인(disjoint) 집합 중의 하나에 포함된다:

(a) S를 활성화시키면 R을 만족하는 최종 상태로 전이될 것이다.

(b) S를 활성화시키면 **non** R을[29] 만족하는 최종 상태로 전이될 것이다.

(c) S를 활성화시키면 어떤 최종 상태에도 도달하지 않을 것이다. 즉 그것의 동작은 적절하게 정지하지 못 할 것이다.

위의 첫번째 집합은 $\mathrm{wp}(S, R)$로 규정되고, 두번째 집합은 $\mathrm{wp}(S, \textbf{non}\ R)$로 규정되며, 그 합집합(合集合; union)은

$$(\mathrm{wp}(S, R)\ \textbf{or}\ \mathrm{wp}(S, \textbf{non}\ R)) = \mathrm{wp}(S, R\ \textbf{or non}\ R)^{30} = \mathrm{wp}(S, T)$$

로 규정된다. 따라서 세번째 집합은 **non** $\mathrm{wp}(S, T)$로 규정되게 된다.

비결정적 체계의 의미를 완벽하게 규정하기 위해서는 (위의 세 가지 경우보다) 좀 더 (많은 경우로 나누는 것이) 필요하다. 주어진 사후 조건 R에 대해서 위의 (a), (b), (c)로 나열한 일이 일어나는 것은 물론 가능하다. 그러나 비결정적 체계의 경우, 하나의 초기 상태가 정의에 의해 세 개의 상호 배제적인 범주 중의 유일한 하나의 일이 일어나게 할 필요가 없다. 이제 각 초기 상태에 대해 일어날 수 있는 일들은 그 세 범주 중의 두 개나 혹은 심지어 세 개 전부에 속할 수도 있게 된다.

그러한 점을 기술하기 위해서 '개방 사전 조건(開放事前條件; liberal precondition)'이라는 개념을 사용한다. 앞에서 우리는 사전 조건이란 것을 '옳은 결과', 즉 R을 만족하는 최종 상태에 도달하는 것을 보장하는 것으로 생각했다. 개방 사전 조건은 보다 약한 조건이다: 그것은 체계가 잘못된 결과를 내지는 않는다는 것, 즉 R을 만족하지 않는 최종 상태에는 이르지 않을 것이라는 것을 보장할 뿐, 정지하지 않을 가능성은 남겨 두고 있다. 개방 사

[29] 이 책에서 "**non**"은 논리학에서의 부정(否定; negation)을 나타낸다. 그 뜻은 뒤따르는 피연산자가 참이면 전체는 거짓, 거짓이면 전체는 참이 된다는 뜻이다.

[30] S는 결정적 메커니즘이기 때문에, 앞에 나온 성질 $4'$을 이용한 것이다.

전 조건에 대해서도 '최약 개방 사전 조건(最弱開放事前條件; weakest liberal pre-condition)'의 개념을 도입할 수 있고 그것을 $\mathrm{wlp}(S,R)$로 나타내기로 한다. 그러면 초기 상태 공간은 원칙적으로 어느 것도 공집합이 아닌 7개의 상호 배제적인 공간으로 세분(細分)된다. (세 개의 물건으로부터 적어도 하나는 포함하여 고르는 가짓수는 일곱이기 때문이다.)[31] 그것들은 세 개의 술어 $\mathrm{wlp}(S,R)$, $\mathrm{wlp}(S,\mathbf{non}\ R)$ 그리고 $\mathrm{wp}(S,T)$에 의해 쉽게 규정된다.

(a) $\mathrm{wp}(S,R) = (\mathrm{wlp}(S,R)\ \mathbf{and}\ \mathrm{wp}(S,T))$
활성화되면 R이 참임을 확립한다.

(b) $\mathrm{wp}(S,\mathbf{non}\,R) = (\mathrm{wlp}(S,\mathbf{non}\,R)\ \mathbf{and}\ \mathrm{wp}(S,T))$
활성화되면 **non** R이 참임을 확립한다.[32]

(c) $\mathrm{wlp}(S,F) = (\mathrm{wlp}(S,R)\ \mathbf{and}\ \mathrm{wlp}(S,\mathbf{non}\,R))$
활성화되면 적절하게 종료하는 동작으로 이끌지 못한다.

(ab) $\mathrm{wp}(S,T)\ \mathbf{and}\ \mathrm{wlp}(S,R)\ \mathbf{and\ non}\ \mathrm{wlp}(S,\mathbf{non}\,R)$
활성화되면 종료하는 동작을 일으킨다. 허나 초기 상태는 최종 상태가 R을 만족할지 아닐지를 결정하지 않는다.

(ac) $\mathrm{wlp}(S,R)\ \mathbf{and\ non}\ \mathrm{wp}(S,T)$
만일 활성화되어서 어떤 최종 상태에 도달한다면, 그 최종 상태는 R을 만족한다. 그러나 초기 상태는 그러한 동작이 종료할지 안할지를 결정하지 않는다.

(bc) $rmwlp(S,\mathbf{non}\,R)\ bfand\ non\ \mathrm{wp}(S,T)$
만일 활성화되어서 어떤 최종 상태에 도달한다면, 그 최종 상태는 R을 만족하지 않는다. 그러나 초기 상태는 그러한 동작이 종료할지 하지 않을지를 결정하지 않는다.

(abc) $\mathbf{non}\ (\mathrm{wlp}(S,R)\ \mathbf{or}\ \mathrm{wlp}(S,\mathbf{non}\,R)\ \mathbf{or}\ \mathrm{wp}(S,T)$
활성화시키면 종료하는 동작이 일어날지 아닐지, 종료하는 경우 R이 만족될지 아닐지를 초기 상태는 결정하지 않는다.

뒤의 네 가지 가능성은 비결정적 기계에 대해서만 존재한다.
$\mathrm{wlp}(S,R)$의 정의로부터

[31] 여기서 세 개의 물건에 해당하는 것은 아래에 나온다. 어떤 체계가 활성화된 뒤 일어날 수 있는 가능한 경우에 대응하는 초기 상태 공간의 분할 기준을 말한다.

[32] 바꿔 말하자면 R이 만족되지 않는 최종 상태에서 "적절하게 종료한다"는 뜻이다.

$$\mathrm{wlp}(S, T) = T$$

가 도출되고,

$$(\mathrm{wlp}(S, F)\ \mathbf{and}\ \mathrm{wp}(S, T)) = F$$

인 것도 또한 명백하다. 만약 그렇지 않다면 (동작의) 종료와 종료하지 않음을 동시에 보장하는 초기 상태가 존재할 것이기 때문이다.

그림 3.1에 그 내부가 각각 $\mathrm{wlp}(S, R)$, $\mathrm{wlp}(S, \mathbf{non}\ R)$, $\mathrm{wp}(S, T)$를 만족하는 상태들을 나타내는 세 개의 직사각형을 사용하여, 초기 상태 공간을 도시(圖示)한 것이 있다.

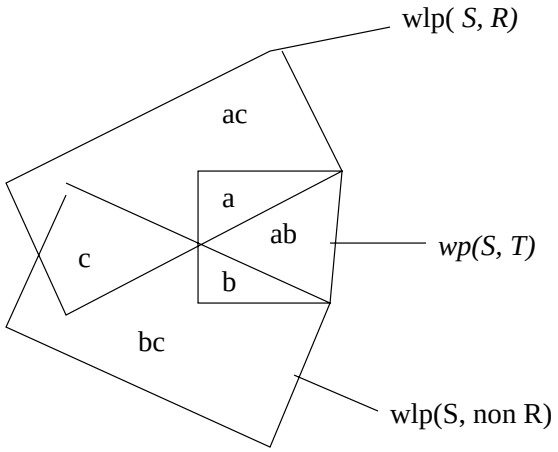

그림 3.1

앞에서 (비결정적 메커니즘에 관한) 분석을 서술한 것은 완결성을 갖추기 위함이며,[33] 또한 개방 사전 조건이라는 개념이 실제로 매우 쓸모있는 것이기 때문이기도 하다. 예를 들어서 어떤 사람이 하나의 프로그래밍 언어를 구현

[33] 결정적 메커니즘 뿐만 아니라 비결정적 메커니즘에 대해서도 빠뜨리지 않고 초기 상태 공간의 구분에 관한 분석을 했다는 뜻이다.

한다고 하자. 그 사람은 모든 옳은 프로그램이 그 구현에 의해 바르게 수행됨을 증명하려고 하지는 않을 것이다. 그는 그 구현에 의해서 옳은 프로그램이 경고도 없이 잘못 처리되지는 않을 것이라는[34] 보장만으로도 만족해야 할 것이다—물론 실제로 바르게 처리되는 프로그램들의 집합이, 그 구현을 충분히 흥미롭게 할 만큼 크다면 말이다.

그러나 당분간은 개방 사전 조건이라는 개념에는 신경을 쓰지 않고, 옳은 결과가 나옴을 보장하는 초기 상태의 규정에만 관심을 제한하겠다. 일단 이러한 도구가 개발되고 나면, 개방 사전 조건에 관해 우리가 흥미를 가질 범위의 토의를 할 수 있도록 어떻게 그 도구를 변형시킬 수 있는가를 고려할 것이다.

[34] 이것은 바로 개방 사전 조건을 어떤 컴퓨터(체계)에서 수행되는 프로그램에 적용시켜 달리 표현한 것에 불과하다. 앞에 나왔지만, 개방 사전 조건은 어떤 체계가 원하지 않는 잘못된 상태에 도달하여 정지하지는 적어도 않는다는 것이다. 그러나 사전 조건과는 달리, 항상 원하는 바른 상태에 도달하여 정지함을 보장하지는 않는다.