계산이란 무엇인가에 대한 아이디어들
Semantic Ideas in Computing *

이 광근
KAIST

1999년 11월 20일

이글은 1996년에 *Computing Tomorrow*라는 제목으로 출판된 책의 한 챕터를 축약해서 번역한 내용이다. 원 저자는 Robin Milner이고 프로그래밍 언어 분야의 연구로 Turing Award를 수상하였는데, 그의 주장에 무조건 정신이 팔릴 필요는 없겠지만, 나름대로 이 분야를 정리하는 데 좋은 참고가 될 것이라고 생각했다. 이 졸역은 38 페이지의 내용 중에서 본인이 이해할 수 있는 부분만을 취사 선택한 후 간략하게 정리한 것이라는 걸 밝혀둔다. 따라서 모든 왜곡과 몰이해와 겉핥기는 본 번역자의 책임이다.

"계산" 혹은 "프로그램"이라는 것을 설명해주는 독특한 원리와 개념들은 과연 있는 것일까? 그래서 우리가 전산학이라는 것을 별개의 학문분과라고 내세울 수 있는 게 있는 것일까? 아니면 그것은 물과 같이 다른 분야의 틀속에서 완전히 이해된 대상이고 우리가 연구할 것은 단순히 그 것을 좀더 잘 써먹는 방법을 고안하는 것 뿐일까?

결론은 명확하다. 프로그램에 대한 풍부한 연구들이 한계를 예측하기 힘들만큼 활발히 진행중에 있고, 그 성과들은 분명히 독자적인 학문분과를 탄생시킬 것이다. 이 과정에서 우리는 어느 학문분과에서나 경험하는 그대로, 경이로운 결과를 맛보기도 하고 예측불허의 방향으로 발전하는 것을 겪게도 될 것이다. 그런데 이 분야는 아직 미숙하고 지금의 알량한 이해의 수준을 가지고는 미래의 연구들이 어떤 결과를 드러낼 지 짐작할 수 없다. 그런고로, 우리는 미래를 예측하기 보다는 지난 20년 동안 "프로그램"이라는 것에 대해 고안된 독특한 개념들을 소개하는 것으로 대신해야 할 것 같다.

프로그램의 실행 이라는 것을 이해하는 데 기본이 되고 있는 개념들은 크게 두 줄기로 부터 왔다. 하나는 수리논리의 이론적인 내용들이다. 지난 100년 동안 집중적으로 발달한 이 분야는 우리 분야의 든든한 기초로 자리하고 있다. 프로그램의 "그럴 리"를 설명해 준다. 다른 하나는, 20세기 중반에 디지탈 컴퓨터의 출현으로 현실화 된 실제 현상이 되겠다. 이 현상들을 수리논리의 표면적인 응용 정도로 생각해서는 곤란하다. 실제 프로그램 시스템의 복잡한 현상들은, 우리분야의 색깔을 결정하는 보석같은 아이디어를 제공하고 있고, 동시에 왜 엄밀한 과학적인 기반이 필요한지를 드러내고 있기 때문이다.

*Chapter 13 in *Computing Tomorrow: Future Research Directions in Computer Science*, edited by Iwan Wand and Robin Milner, Cambridge University Press, 1996.

이 두 줄기가 서로를 휘감으면서 탄생시킨 개념들은 엄밀한 분석을 통해 유기적인 짜임새를 만들어 왔는데, 이 과정에서 두드러지기 시작한 몇개의 독특한 개념들을 살펴보도록 하자. 수학이나 다른 공학 분과에서 찾아볼 수 없는 그야말로 독특한.

- 같은 프로그램이란 무엇인가 **(semantic equivalence)**:

  프로그램은 생각대로 실행되야 한다. 그 생각을 엄밀히 기획한 프로그램대로, 실행 프로그램이 구현되었는지를 확인할 수 있으려면, 이 두 프로그램이 과연 같은지를 검증할 수 있어야 한다. 예를들어, 통신 프로토콜에서는, 기획한 프로토콜대로 정확하게 프로토콜 프로그램이 구현되야 한다. 프로토콜의 구현이 그 기획과 일치하는 지를 검증할 수 있는가? 컴파일러라는 소프트웨어는 주어진 입력 프로그램과 같은 일을 하는 프로그램을 만들어내야 한다. 기획한 대로 되었는가?

  이와같은 검증들이 가능해지려면 두 프로그램이 같다는 것이 무엇인지 정의되어 있어야 한다. 이때, 기획하는 언어는 구현하는 언어와 같을 수도 있고 다를 수 도 있는데, 두 프로그램이 같다는 것에 대해서 아직은 모든 전문가들이 동의하는 정의는 나오고 있지 않다. 대신에, 대상 현실과 목적에 따라 다양한 정의와 검증방법들이 고안되고 있다.

- 순서대로 계산한다는 것은 무엇인가 **(sequentiality)**:

  정보가 흘러다니는 순서를 한 줄로 꿸 수는 있는 것일까? 놀랍게도, 그러한 과정을 함수로 표현하고자 할 때, 그러한 함수들만으로 모여진 정확한 수리적 모델을 찾기가 매우 어렵다는 것이 지금까지의 결론이다.

  이러한 당황스런 결과를 겪으면서, 우리는 정보의 흐름이라는 현상에 대해 보다 근본적인 이해에 도다르고 있다고 생각한다. 정보 흐름의 순서가 무엇인지를 이해하면 병렬 계산이 무엇인지를 이해하게 될 것이고, 순차적인 프로그래밍 언어에서 부족한 것이 정확하게 무엇인지가 찾아지게 될 것이다.

- 프로그램의 안과 밖은 무엇인가 **(interaction)**:

  프로그램의 실행은 더이상 외부와 단절된 단일 컴퓨터 내부에서 일어나는 것이 아니다. 넓은 의미의 컴퓨팅 시스템(예를 들어, 통신 시스템)에서 프로그램은 외부와 끊임없이 접촉하고 변화하며 그 계산 과정을 바꾸어 간다.

  이러한 상황을 정확히 정의하고 이해할 수 있을까? 예를들어, 결정될 수 없음, 가로채기, 선점하기, 충돌, 연관 없음등의 개념을 정의하기 위한 여러가지 독특한 모델들(프로그래밍 언어에 성공적으로 적용된 고차 함수 모델, 파이프라인 모델 (stream model), 상호 작용의 동시성을 중시하는 반응 모델 (reactive model), 계산과정이 구성되고 상호작용하면서 변화해 가는 룰(algebra)에 대한 방법등)이 고안되어 왔지만, 실제의 복잡한 전체 시스템의 상호작용 현상을 만족스럽게 요약해주는 방법으로 발전하지는 못했다. 하지만, 위와같은 정의와 실험이 반복되면서 서서히 그 경지에 접근할 것으로 보인다.

- 옮겨다니는 계산이란 무엇인가 **(mobility)**:

데이타는 이미 컴퓨터의 메모리 주소를 옮겨다녀왔고, 병렬 프로그램에서는 동시 다발 프로세스가 컴퓨터 자원을 최대한 이용하기 위해서 CPU를 옮겨다녀왔다. 요즈음은 통화중인 휴대용 전화기가 통화권역을 옮겨다니고 있다.

옮겨다니는 것, 그래서, 연결되기도 하고 끊어지기도 하는 것이 계산의 전부가 아닐까? 여기에 촛점을 맞추는 계산모델이 다양하게 만들어지고 있다. 파이 계산법(π calculus)은 연결이 만들어지고 끊어지는 것을 핵심으로하는 기초적인 형식이고, 울타리 계산법(ambient calculus)은 옮겨다니면서 계산 테두리 영역이 변화하는 것을 핵심으로한다. 모두가 기존의 프로그래밍을 표현할 수 있는 일반적인 언어라는 것이 밝혀졌고, 계속된 연구 성과를 바탕으로 새로운 계산 파라다임에 기초한 프로그래밍 언어로 발전할 것으로 보인다. 마치, 1940년대의 람다 계산법(λ calculus)이 지금까지의 프로그래밍 언어를 이해하고 디자인하는 데 뼈대가 되었듯이.

위의 개념들은 소프트웨어라는 분야를 독립적인 분야로 설 수 있도록 하는 아이디어들로 자리잡고 있는데, 그중에는 충분히 엄밀하게 발달한 덕택에 직관적이면서 정확하게 설명할 수 있는 것도 있었지만, 그렇지 못한 까닭에 구름잡는 논조로 밖에는 설명할 수 없는 것도 있었다. 어느 분야가 제대로 성장했느냐는 그 개념의 틀이 충분히 성숙해서, 정확하면서도 일상적인 언어로 그 내용을 설명할 수 있느냐에 달렸다고 본다. 이 기준으로 살펴 본다면 우리 분야는 아직도 많은 실험과 실패의 과정을 겪어보지 않은, 아직 첫 미소가 터지지 않은 아기의 모습이라고 해도 심한 과장은 아니라고 본다.