# 지금의 정보기술을 보는 눈높이 II: 벅잡는 기술의 전개과정

이광근*
KAIST

[도올고신], 제 39신, pp.57-58, 2001년 9월

지난글에서는 소프트웨어가 생각한 대로(기획한 대로) 작동할 지를 자동으로 확인해 주는 기술이 왜 중요하고, 그 기술을 달성하기 위한 노력이 소프트웨어에 대한 연구의 한 핵심 축이었다는 것을 소개했었습니다. 이번 글에서는 그 기술의 발전이 지난 40년간 어떻게 진행되어 왔는지, 그 속내용을 소개하겠습니다. 현재는 미개하기 이를데 없는 기술들이지만, 미래 소프트웨어 기술의 완성을 향하여 한발한발 찬찬히 나아가고 있는 전세계 전산학자(computer scientists)들의 꾸준한 성과들 말입니다. 이 성과들은 오랜기간의 연구를 통해서 매우 천천히 이루어지고 있지만 그 매번의 성과들은 견고하고 엄밀한 모습으로 다음 단계의 발전을 튼튼히 떠받치고 있습니다.

소프트웨어 버그를 자동으로 찾는 기술은 세 박자의 호흡만에 한발 전진하는 형태라고 정리할 수 있습니다. 첫 번째 호흡에서는 우리가 확인할 수 있는 버그의 한계를 정확하게 정의하고, 두 번째 호흡에서는 정의한 버그의 존재를 찾는 방법을 고안하며, 세 번째 호흡에서는 그 방법을 컴퓨터가 자동으로 실행할 수 있도록 소프트웨어로 만들어 냅니다. 이때, 각 호흡마다 애매하고 허술한 구석이 없을 때에만 비로소 한 발짝의 전진이 있게 됩니다. 버그의 정의는 모호하지 않아야 하고, 고안된 방법은 프로그램이 그렇게 정의된 버그를 품고있다면 반드시 찾아낼 수 있어야 하고, 그 방법을 구현한 소프트웨어는 그 방법 그대로 충실히 구현되어야 합니다. "반드시"와 "충실히"라는 표현들을 쓴 것은 엄밀한 수학적인 증명과정을 거친다는 것을 뜻합니다.

이 세 박자 호흡을 반복하면서 확인할 수 있는 소프트웨어 벅의 정의는 매 스텝마다 점점 정교해 집니다. 첫 스텝에서는 비교적 쉬운 소프트웨

*현재 서울대학교 컴퓨터공학부 소속

어 벅들을 확인해주는 기술이 만들어지고, 두벗째 스텝에서는 그보다 정교한 벅들을 확인해주는 기술이 만들어지고, 세번째 스텝에서는 더욱 더 정교한 벅들을 확인해주는 기술이 만들어지고... 이렇게 하면서 궁극적으로는, 소프트웨어가 생각대로 작동할지를 자동으로 확인해주는 소프트웨어 기술을 달성해가는 것입니다.

의아해하시겠지만, 이러한 스텝을 밟으며 소프트웨어 기술이 전진한 횟수는 이제 겨우 두 발짝이랍니다. 1970년대에 달성된 첫 발짝은, 생긴게 잘못된 프로그램을 자동으로 찾아내는 기술입니다. 프로그램이 제대로 작동하려면 우선 생긴것이 제대로 되야 하지요. 이때 버그는 프로그램의 생김새가 틀린것이라고 정의된 경우고, 이러한 버그를 정확히 자동으로 찾아주는 기술이 달성되었습니다. 정확히 찾아준다는 뜻은, 안전하고(sound) 빠뜨림없다는(complete) 뜻입니다. 멀쩡한 프로그램을 기형이라고 진단하는 경우(불안전한 경우)도없고, 기형인 프로그램을 멀쩡하다고 하는 경우(빠뜨리는 경우)도 없다는 뜻이지요. 이 기술을 저는 1세대 벅잡는 기술이라고 합니다. 이 기술을 문법검증기술(parsing)이라고 하고, 완전히 완성되어 오늘날의 프로그래머가 프로그램을 짜면서 늘상 아무렇지도 않게 사용할 만큼 관심밖으로 사라져 버린 성숙한 기술입니다.

그러나 이 벅잡는 기술이 겨우 1세대인 까닭은, 생긴건 멀쩡한데 생각대로 돌아가지 않는 소프트웨어에는 속수무책인 기술이기 때문입니다. 겉모습의 벅 뿐이아닌, 속내용의 벅 까지 자동으로 찾아주는 기술이 필요한 거지요.

1990년대에 빛을 보기 시작한 두번째 발자욱은, 타입검증(type checking)이라는 기술입니다. 제 2세대 벅잡는 기술입니다. 이때 버그의 정의는 겉모양이 틀린 프로그램만이 아니고, 속 내용이 잘못될 수 있는 경우까지를 포함하게 됩니다. 여기서 잘못된 속 내용이란, 프로그램이 실행중에 잘못된 값이 잘못된 계산과정에 휩쓸리는 경우로 정의됩니다. 프로그램의 실행은 일종의 계산인데, 그 계산중에는 분별있는 일들이 일어납니다. 더하기에는 숫자만 들어와야 한다던지, 결혼하기라는 계산에는 남자와 여자가 한쌍으로 있어야 한다던지, 수력발전하기에는 우라늄대신에 물이 있어야 한다던지 말입니다. 이렇게 분별있게 값들이 소통되야 하는데, 그렇지 않은 경우를 타입에 맞지 않다고 합니다. 이러한 경우가 발생하면 프로그램의 진행은 생각대로 흐르지못하고 급기야는 값작스럽게 멈추고 맙니다. 휘발유가 공급되야할 전투기의 엔진에 물이 새어들어 비행중에 추락해 버리듯이 말입니다. 프로그램에서 이러한 무분별한 혼돈의 경우가 실행중에 발생할 지를 미리 검증하는 방법이 바로 타입검증입니다. 이 타입검증 기술은 지난 30년의 전산학의 연구성과중에서 가장 자랑스러운 결과중의 하나입니다. 이 성과는 프로그램을 작성하는 언어(프로그래밍 언어라고 합니다)를 연구하는 분야에서 달성되었는데, 프로그램의 안전한 타입검증은 그

프로그래밍 언어가 제대로 디자인된 경우에만 가능합니다. 그러한 프로그래밍 언어기술은 아직은 문법검증기술만큼 모든 프로그래머들이 늘상 사용하는 기술로 널리 퍼지지는 않았으나 그러한 분별있는 프로그래밍의 세계는 곧 펼쳐질 것으로 보입니다.

하지만, 이 2세대 벅잡는 기술로도 아직은 미흡한 실정입니다. 실행중에 모든 값이 분별있게 착착 흘러드는 프로그램이라고 해도, 생각대로 작동하지 않을 수 있기때문이지요. 타입에 맞는다는 것은 실은 초보적인 조건일 뿐입니다. 휘발유만이 비행기의 엔진에 흘러든다는 것이 검증되었다고 해도, 휘발유의 폭발력이 수준미달인 경우라면 비행기가 떨어질 수 있습니다. 라면을 끓이는 계산에 항상 라면과 물과 불이 들어선다고 해도, 물이 한방울 뿐이거나 들어선 불이 포항제철의 용광로정도라면 라면의 맛이 망쳐질 수 있습니다. 타입에 맞게 컴퓨터 프로그램이 실행되더라도 생각한 것과는 다르게 진행되는 경우, 이러한 버그를 자동으로 검증하는 기술이 필요하게되지요.

최근 가속이 붙기 시작한 제 3세대 벅잡는 기술은, 이렇게 확장된 버그를 검증하는 기술을 지향하고 있습니다. 생긴모습도 멀쩡하고, 실행중에 잘못된 값이 흘러들지도 않지만, 실행중에 가져야할 정교한 조건을 만족시킬 수 없는 프로그램, 이것을 집어내는 기술입니다. 이 기술은 지난 20년간의 자동증명기술(theorem proving)과 프로그램 분석기술(static analysis), 그리고 이미 지난 100년간 쌓아졌던 현대 논리학(mathematical logic 혹은 computational logic)의 성과위에서 자라고 있습니다. 이 기술의 얼개는 다음과 같습니다. 프로그램이 실행중에 만족해야 하는 정밀한 속사정이 엄밀한 논리식으로 정의됩니다. 주어진 프로그램이 실행중에 그 논리식을 거짓으로 만들 수 있는 가능성이 조금이라도 있는지를 검증합니다. 있으면, 그 프로그램은 벅이 있을 수 있는 것이고, 그 가능성이 전혀 없다고 판정되면 그 프로그램은 벅이 전혀 없는 것입니다. "가능성이 조금이라도 있는지"라는 표현을 쓴 것은, 벅이 없는데도 불구하고 벅이 있다고 결론내리는 경우가 생기기 때문입니다. 보수적인 것이지요. 하지만 안전한 것은 보장됩니다. 벅이 없다고 결론내려지면, 정말로 없다는 것은 보장됩니다. 안전하기는 하지만 완전하지는 못한 검증입니다. 하지만 우리는 완전하지 못하다는 것에 만족해야 합니다. 완전한 검증이 불가능하다는 것은 증명된 사실이기 때문이지요.

이 3세대 벅잡는 기술이 성숙되어지는 시기는 낙관적으로 잡아서 앞으로 20년정도 이후가 아닐까 생각됩니다. 저는 이 3세대 기술을 획득하는 그룹이 다음세대의 정보기술의 헤게모니를 잡을 것이라고 생각합니다. 그런데, 벅잡는 기술의 끝은 과연 있는걸까요? 3세대 벅잡는 기술도 완성되면 그 다음은 어떠한 정보기술의 문제가 우리를 시험에 들게 할런지 저는 궁금합니다.