ENE414 프로그래밍언어론
강의노트 [1]

프로그래밍언어 설계의 역사

한양대학교 ERICA캠퍼스 컴퓨터공학과
도경구

2012년 1학기
(version 0.1)

인류의 역사를 공부하면 현재와 미래의 우리 삶을 더 잘 이해할 수 있듯이, 프로그래밍언어 설계의 역사를 살펴보면 프로그래밍 언어에 대해서 더 잘 이해할 수 있을 것이다.

1950년대 - 폰 노이만 머신, 어셈블리 언어, Fortran

- 진공관 컴퓨터의 출현: 튜링머신Turing machine을 진공관을 이용하여 현실화함. 진공관끼리 선을 연결하는 작업자체가 프로그래밍
- 존 폰 노이만John von Neumann이 괴델Goedel의 이론에서 영감을 얻어 프로그램을 수로 표현하여 메모리에 저장하여 실행시킴 – 기계어의 출현. 어셈블리 언어는 기계어에 이름을 붙인 것에 불과함. 어셈블러는 어셈블리 프로그램을 기계어로 번역함
- IBM의 존 백커스John Backus가 과학자들이 풀 문제를 수식으로 쓰면 이를 컴퓨터가 계산할 수 있도록하는 "자동 프로그래밍 언어"를 개발함 - Fortran(FORmula TRANslator). 프로그램을 읽고 어셈블리 언어로 번역하는 파서와 컴파일러를 개발함. 대성공!!
- 비즈니스에 사용할 수 있는 "자동 프로그래밍 언어"로 COBOL(COmmon Business-Oriented Language)이 이어서 개발됨.

1960년대 - 문법, 블록구조, 시스템 언어, 파싱기술, 컴파일 기술, 실행 의미론

- 50년대 후반, 유럽의 컴퓨터 학자들이 위원회를 조직하여 모여서 스택stack이라는 새로운 개념의 자료구조를 이용하여 새로운 언어 Algol60를 설계함. 혁신 포인트는
  - 문법으로 구문을 정의함. 노암 촘스키Noam Chomski가가찾아낸 아이디어였으나, 존 백커스가 개별적으로 실용화하였음
  - 블록 구조를 사용하여 변수를 국지적으로 사용하게 함으로써 실행 중 자동 메모리 관리 가능하게 함
  - 자신을 호출하는 재귀 프로시저 개념
  - 코펜하겐의 피터 나우어Peter Naur는 프로그래밍언어의 의미가 정의된 문서를 작성함. 최초로 의미를 귀납적으로 설명함.

  Algol은 혁신적인 언어였으며 버로우Burroughs회사에서 스택머신을 만들어 상용화하였으나 IBM의 마케팅에 밀려 빛을 보지 못함.
- 하드웨어를 동작하기 위한 운영체제 등의 시스템을 만들기 위해서 유용하게 사용할 어셈블리 같은 언어들이 지속적으로 개발됨. 캠브리지와 옥스포드 대학의 크리스토퍼 스트라치Christopher Strachey의 CPLCommon Programming Language,에서 출발하여, BCPL, B, C가 차례로 개발됨.

- 파서와 번역기 연구가 활발하게 진행됨. 문법으로 구문을 정의하고, 이를 기반으로 파서를 제작하고 검증함. 파서가 만들어 내는 파서트리를 입력으로 받아서 번역기는 의미를 계산하여 출력을 내줌. 따라서 인터프리터라고 불렀음. 출력이 컴퓨터에서 실행할 수 있는 기계어코드 또는 어셈블리코드인 경우 컴파일러라고 했음.

- 문법을 분류하는 연구가 활발하게 이루어졌으며 결과로 LL(k), LR(k)과 같은 특정 성질을 가진 문법들로 분류하였으며 이 특징을 이용한 하향식, 상향식 파싱 알고리즘이 개발됨. 도날드 크누스Donald Knuth가 주도하였음. 그 이후, 문법을 입력으로 받아서 파서를 자동으로 생성하주는 퍼서생성기 출현. 프로그램이 어떻게 실행되는지도 표현할 수 있어서 언어의 프로토타입의 제작이 가능했음. 실용성은 아주 미약

- 언어의 의미를 귀납적으로 정의하여 그 언어의 프로그램이 직접 실행되는 가상머신 개발 시작. 대표적으로 비엔나의 IBM 연구소에서 디네스 비요너와 크리프 존스가 주도하여 개발한 Vienna Definition Language. 프로그래밍언어의 실행의미를 표현해주는 최초의 가상머신이었음

- MIT에서 존 맥카시John McCarthy는 자연어처리를 용이하게 할 수 있는 언어개발에 착수. 리스트 자료구조를 만들고 이와 알론조 처치Alonzo Church의 람다계산법lambda calculus을 응용하여 LISP (LISt Processing language)를 설계하고 구현함. 이 언어를 전용으로 실행하는 LISP 머신이 개발되었으나 상업적으로 성공하지는 못함. 이후 인공지능 언어로 널리 사용됨

- 프로그램을 수학과 논리학적인 엔티티로 표현된 지식을 변환기로 인식하기 시작하면서, 지식을 "주장(assertion)"으로 표현하며, 프로그램의 실행은 이 지식을 변환하는 과정으로 이해하게됨. 로버트 플로이드Robert Floyd와 토니 호어Tony Hoare가 주도한 공리적 의미론axiomatic semantics으로 발전하였으며, 이후 명세과 검증을 엄밀하게 할 수 있게되는 기초를 닦음. 하지만 언어를 어떻게 설계하고 구현하는지에 대해서 알아내기기 불가능.

1970년대 - 가상기계, SLR/LALR문법, 표시적의미론, 함수중심언어

- 프랭크 드레머Frank DeRemer가 SLR(k)언어를 찾아내고 후속타 LALR(k)언어가 발견됨으로써 파서 제작 문제는 완전히 풀렸다. 그 결과 지금은 Yacc, ANTLR과 같은 파서생성도구를 사용하여 문법만 가지고 파서를 자동으로 생성할 수 있음

- Algol60 설계 위원회 출신들이 연구를 계속해서 괄목할 만한 결과들을 내놓았다. 크리스틴 나이가드Kristin Nygaard는 Simula67을 설계했고, 니클라우스 워스Nicklaus Wirth는 Pascal과 Modula를 설계했다. Simula는 최초의 객체지향언어이다. Pascal는 체계적인 데이터타입을 최초로 제공한 언어였고, 가상머신의 개념을 최초로 도입하여 컴파일러의 표준화를 시도하였다. 가상머신코드를 해당 CPU의 기계어로 번역하는 어셈블러만 만들어서 포팅하면 컴파일러 하나로 여러 CPU에 사용이 가능하였다. 이 개념은 결국 Java에 적용되어

성공에 크게 이바지 한 셈이 되었다.

- C 부류 언어의 시조인 클리스토퍼 스트라치Christopher Strachey는 타겟머신이나 가상머신과는 상관없이 언어의 의미를 기술할 수 있어야 한다고 생각했다. 프로그램은 입력을 받아서 출력을 내주는 함수를 뜻하므로 그렇게 표현할 수 있으리라 스트라치는 믿었다. 마침내 수학적 함수로 표현하기 위해서 알론조 처치Alonzo Church 교수의 람다 표기법lambda calculus을 채택하고, 프로그램 구절을 수학적 의미로 귀납적으로 정의할 수 있었다. 이 기술을 표시적 의미론denotational semantics이라고 한다. 오토마타 이론과 직관수학의 선구자인 다나 스캇Dana Scott은 스트라치와 함께 스트라치의 함수가 계산하는 계산값의 래티스lattice 구조를 정의했다.

  표시적 의미론은 언어분석 분야의 엄청난 연구를 유발시켰다. 수학에서 유래된 증명기술이 프로그램과 언어에 대한 정확성에 대해서 증명하는데 사용가능했기 때문이다. 람다 표기법 자체를 프로그래밍언어로 사용하는 연구도 촉발시켰다. 그 결과로 함수중심 언어인 ML, CAML, OCAML, Haskell과 같은 언어들이 탄생했다. 이 부류의 언어의 특징은 루프와 값이 변경되는 변수가 없다는 것이다. 계산은 모두 (재귀) 함수호출과 파라미터 전달로 이루어진다.

- 스트라치와 같은 시기에 도날드 크누스Donald Knuth는 파스트리의 마디에 그 부분에 해당하는 의미(속성attributes이라고 함)를 부착하여 프로그램의 의미를 표현할 수 있다는 사실을 발견하였다. 이렇게 속성이 부착된 문법을 속성문법attribute grammar이라고 하며, 프로그램 전체의 의미는 해당 파스트리의 하위트리 의미를 조합하여 결정하도록 하였다. 이 포맷은 수학 함수나 특정 타겟코드와도 연관이 없지만, 스트라치의 귀납적 정의를 이용하여 소스코드를 타겟코드 또는 의미로의 번역을 자연스럽게 표현할 수 있다.

1980년대 - 컴포넌트 언어, 객체지향 모델, 논리=알고리즘, 의미계산법

- 분산시스템의 필요성과 그래픽 장비의 출현으로 소프트웨어가 대형화해가기 시작했다. 이에 맞추어 소프트웨어 개발이 부품을 조립하여 조립하는 형태가 되면서 이를 지원하는 Modula, Ada, Euler가 설계되었다. 부품이 결합하면서 이름 충돌이나 부품의 규격이 맞지 않아 재사용이 힘들게 될 수 있다. 이런 부류의 문제를 해결하기 위해서 이름붙이기, 수입import, 가시거리, 영역scope과 관련된 연구가 활발해지기 시작했다.

- "소형프로그램 부품"끼리 서로 계산 기능을 공유하고 상호협조하면서 계산을 하도록 하는 연구도 진행되고 있었다. 행위자actor라고 하는 의미 모델이 구상되었고, Smalltalk은 이러한 개념에 영감을 받아서 개발된 언어이다. Smalltalk에서는 이 행위자를 객체objects라고 하였으며, 이리하여 객체지향 프로그래밍object-oriented programming이 시작되었다.

  또한 메시지 전달과 이벤트처리를 통하여 GUIgraphic user interface 조립을 자연스럽게 할 수 있어야 한다는 요청사항도 Smalltalk 언어를 설계하는데 영향을 끼쳤다.

- 논리식으로 작성된 명세에 맞게 소프트웨어가 작성되었는지를 보증하기 위해서 프로그램의 개발에 (직관적) 서술논리를 적용시키는 연구가 활발히 이루어졌다. 직관 논리intuitionistic logic의 관점에서 보면, 명세는 데이터타입이고, 프로그램의 실행이 명세를 만족하게 되면 그 명세가 바로 정확히 프로그램의 타입이 된다. 놀라운 점은 직관논리를 사용하여 명세에 대한 증명을 만들면, 그 증명이 바로 프로그램 코드가 된다는 사실이다! 이 발견을 실제 소프트웨어에 실현시키려면 갈길이 멀지만, 실현되는 경우 얻게 되는 이득은 실제로 상당히 클 것이다. 특히 안전제일 시스템이나 보안 시스템에 이를 적용시키려는 노력은 지금까지도 계속되고 있다. 개발된 관련 논리 시스템은 Intuitionistic Type Theory, nuPRL, LCF 등이 있고, 관련 프로그램 증명 도구로는 NuPRL, LCF, HOL, Isabelle, Coq 등이 있다.

- 논리학이 컴퓨팅에 적용되 사례는 또 있다. 말 그대로 "명세를 실행하자" 이다. 이 경우 명세가 주어지면 그 명세를 증명하는 "증명"을 찾는 작업을 수행하게 된다. 이 결과 탄생한 대표적인 언어가 Prolog이다. Prolog 인터프리터는 혼 절 논리식Horn-clause logic에 대에 대하여 Alan-Robinson 발견한 해법을 사용하는 정리증명기 이다. Prolog는 전통적인 문제뿐만 아니라 인공지능과 데이터베이스 분야의 문제를 해결하는데 많이 사용되고 있다.

1990년대 - 도메인특화 언어, 소프트웨어 구조, 추론기반 의미론

- 특정 응용분야 전용 언어가 개발되는 빈도가 점점 많아 지기 시작했다. GUI 구축용 언어, 표계산spreadsheet 언어, 웹문서 작성 언어, 네트워킹 프로그램, 수학계산용 언어 등 특정 분야에 특화된언어를 도메인특화 언어domain-specific language라고 하며, 지금도 계속 발전하고 있다.

  도메인특화언어로 개발되었으나 실패한 언어중의 하나가 Java이다. Java는 원래 Smalltalk에서 쓰는 것과 같은 객체를 처리하는 프로그램을 내장된 칩에 장착하기 위해서 개발되었다. 이후에 웹언어로도 사용할 수 있도록 확장했는데, 애플릿applets이라는 객체를 웹으로 보내서 웹브라우저에서 실행되게 하였다. Java는 가상머신에서 돌아가도록 만들었기 때문에 포팅을 하고 웹브라우저에 심기가 쉽어서 Smalltalk의 실용적인 대안으로 각광을 받기 시작했다. 객체지향 프로그래밍을 처음 배우는 등용문 역할도 잘 해냈다. 당시 유일한 대안은 기존의 C언어에다 객체지향을 복잡하게 장착한 C++뿐이었다.

- 분산 계산 또는 동시 계산을 수행하는 언어는 스캇과 스트라치의 표시적 의미론으로 분석하기 힘들었다. 어딘버러 대학의 고든 플롯킨Gordon Plotkin과 프랑스 INRIA의 질레 칸Gilles Kahn은 계산이 실행하는 방식을 자연추론 형태의 증명 규칙으로 표현하기를 제안하였다. 이를 실행과정 의미론operational semantics이라고 하는데 큰걸음big-step 표현방식과 작은걸음small-step 표현방식으로 나뉜다. 가상머신이 있다고 가정을 해야하고, 프로그램의 실행성질을 증명하기 그리 간단하지 않다. 하지만 분산, 비결정성, 동시성을 표현하기 좋은 장점은 가지고 있다.

  이 외에도 프로그램의 의미를 표현하기 위한 메타언어meta-language인 전통적인 람다계

산법 대신 밀너Milner의 CCS, 아바디와 카델리Abadi-Cardeli의 객체 계산법object calculus, 프로세스 대수Process Algebra 등이 제안되었다.

- 소프트웨어 구조를 표현하기 위한 그림언어(가장 유명한 건 UML)가 유명해졌다. 의미가 불분명한 점이 있어 다소 혼란이 있지만 의미를 고정하기 위한 노력들이 이루어지고 있다.

2000년대 - 정적분석, 프로그램검증, 보안, 안전제일 애플리케이션

- C#과 같은 대규모 언어와 Ajax, XML과 같은 소규모 언어가 계속적으로 나타나고 있다.
- 최근에는 의미론을 응용한 분석기법을 사용하여 프로그래머가 정확한 코드를 작성하게 도와주는 도구를 개발하는 쪽으로 많은 연구가 이루어지고 있다. JMLJava Modelling Language, Spec#, Coq과 같은 도구들은 서술 논리학predicate logic 또는 요약해석abstract interpretation 이론을 기반으로 개발되었다. 요약해석이란 프로그램을 실제 입력값이 아닌 요약값으로 돌리는 기술을 말한다.

에필로그

- 새로운 언어는 필요에 의해서 개발된다. 예를 들면, Fortran은 물리학자들이 세운 식을 프로그램하게 도와주기 위해서 개발되었다. Simula는 시뮬레이션을 하기 위해서 개발되었고, Pascal은 이동성을 목표로 개발되었고, C 언어는 시스템 프로그래밍 용으로 개발되었고, Java는 내장 시스템에 쓰려고 개발되었다. 도메인 특화 언어로 보면 사례가 훨씬 더 많이 있다. Javascript, Matlab, HTML, CSS, SQL, Ajax... 특정 응용프로그램 분야에서 작업을 할때, 그 분야에 특화된 언어가 있으면 편할 것이다. 별로 맞는 걸 찾지 못하면 아마도 하나 만들고 싶어질지도 모른다.
- 언어는 한 사람이 독자적으로 설계하거나, 아니면 비전을 공유하고 있는 소규모 팀이 설계한다. 설계자는 해당 응용 분야와 실행 (하드웨어) 프랫폼을 잘 이해하고 있어야 한다. 보통 설계자는 언어를 구현해본 사전 경험이 있다.
- 실행 프랫폼 부터 선정해야 한다. 실행 머신이 될 수도, 가상 머신이 될 수도 있을 것이다. 결국 프로그래밍 언어는 기계를 돌리기 위한 것이니 어쩌면 당연한 일이다.
- 설계하는 그 언어를 사용하든지 (부트스트래밍bootstrapping이라고 함) 아니면 다른 언어를 사용하여 인터프리터 형식으로 프로토타입을 만들어본다. 프로토타입으로 좀 테스트를 해본 후 만족하면 본격적으로 효율적인 구현에 들어간다.
- 정형 의미론 기술은 일반적으로 설계 개발 단계에서는 덜 사용되고, 사후 문서화 또는 분석용으로 주로 사용된다.

이 강의의 막바지에 소규모 도메인특화 언어를 설계하고 구현해보도록 할 것이다. 실제로 이러한 소규모 언어는 소프트웨어 개발하면서 빈번히 새로 만들어지고 사용된다.