한없이 다양한 분야에 걸린 관심, 하나의 공통된 맥락

데니스 샤샤, 캐시 레이저어, 박영숙 옮김

컴퓨터 프로그래밍은 시나 음악의 창작과 같은 예술의 형식이다.
- 도날드 크누스

이 글은 세종연구원에서 출간된 '컴퓨터를 만든 15인의 과학자(Out of their minds)' 중 Donald Knuth 편만을 LaTeX을 이용해서 다시 편집한 글입니다.

도널드 크누스http://www-cs-faculty.stanford.edu/~knuth/ 가 정말로 단지 한 사람의 인물일까? 그가 쓴 150편의 논문들에는 이 분야에서 가장 중요한 세가지 알고리즘이 포함되어 있다. 그의 최대 걸작(현재 네 권째 집필 중)인 '컴퓨터 프로그래밍의 예술 The Art of Computer Programming'에는 이 분야의 대부분을 포괄하는 초기의 연구 및 조사자료들이 실려 있다. 크누스는 그의 일생에서 30년이라는 짧은 시간동안 인쇄에 사용되는 강력한 소프트웨어 시스템들을 만들어 내고, 고대 바빌로니아의 알고리즘이나 성경의 시편 등 다양한 주제들을 다룬 글을 발표하였으며, 게다가 직접 펜으로 소설까지 쓰는 시간들을 가져 왔다. 또한, '여가 시간'에는 손수 설계한 파이프 오르간을 연주한다.

전 생애를 통해 크누스는 대중들의 갈채와 더불어 많은 상을 받아 왔다. 그 중에는 1974년에는 컴퓨터 과학 분야 최고의 상으로 꼽히는 튜링상을 수상하였으며, 1979년에는 지미 카터 대통령으로부터 국가 과학 훈장을 받기도 하였다. 그러나 크누스는 그처럼 영예로운 포상들에 대해 그저 무관심한 태도를 보인다. 그의 튜링상 수상 기념으로 제작된 접시에는 지금 과일들이 담겨져 있다.

1. 알프레드 E. 뉴먼에서 폰 노이만 까지

크누스는 1938년 밀워키에서 태어났다. 크누스 가문에서는 처음으로 대학을 나온 그의 아버지는 초등학교 교사로 처음 교단에 들어서 나중에 루터란 고등학교에서 부기를 가르쳤다. 그는 또 주일엔 교회에서 오르간을 연주하였다. 도널드는 아버지로부터 음악과 교육에 대한 이해를 물려받았으며, 특히 말투를 그대로 닮았다.

> *난 대체로 교사들이 가장 잘하는 분야에 관심을 나타냈습니다. 우리 학교에는 문장의 다이어그램 만들기를 훈련하는 훌륭한 과정이 있었지요. 우리 친구들 일당은 수업을 마친 뒤 시에서 문장의 부분들을 맞추는 놀이를 하곤 했습니다.*

학교 신문의 편집인 일을 하면서 크누스는 크로스워드 퍼즐을 만들었다. 그는 단어들 가운데 어떤 패턴을 찾아 내는 일을 즐겼던 것으로 기억한다. 크누스는 일찍부터 수상 경력을 찾기 시작했다. 그가 8학년이 되었을 때 한 사탕 제조업자가 후원을 하여, '지글러의 왕 막대사탕(Ziegler's Giant bar)'이라는 구절에 들어 있는 글자들을 가지고 누가 가장 많은 단어를 만들어 내는지를 겨루는 대회가 열렸다. 크누스는 한 번 시도해 보기로 마음먹었다.

> *난 어포스트로피를 사용하지 않고도 약 4500개에 달하는 단어를 찾아냈지요. 어포스트로피까지 사용했다면 훨씬 더 많은 단어를 찾아낼 수도 있었을 겁니다. 심사 위원들의 마스터 리스트에 수록된 단어의 수도 겨우 2500개 정도밖에 되지 않았어요.*

그는 1등상으로 텔레비전 수상기(당시로선 값비싼 품목이었던)한 대와 전교생이 다 먹을 수 있을 만큼의 지글러 막대사탕을 받았다. 고등학교 시절엔 웨스팅하우스 과학 재능 경진대회(Westinghouse Science Talent Search)에서 '무게와 측정의 potrzebie 시스템'이라는 탁월한 안을 제출하여 가작에 당선되었다. 크누스는 훗날 그의 생애를 단적으로 설명해 주게 되는 그 특유의 주의력으로 정밀하게 자신의 기본적인 장치들을 정의하였다. 그것은 'potrzebie는 MAD Magazine 26호의 두께, MAD는 48가지, whatmeworry전력의 기본 단위'라는 내용이었다. 1957년 6월엔 바로 *Mad Magazine*자체에서 그 소품을 25달러에 사들여, 도널드의 다작 인생에서 정식으로 발행된 첫 작품이 나오게 되었다. 하지만 고등학교 시절 그가 가장 많은 시간을 쏟아 부은 건 글쓰기나 과학이 아니라 바로 음악이었다.

> *난 대학에 가면 음악을 전공할 생각이었습니다. 원래는 색소폰을 불었지만, 튜바 주자가 사고를 당하면서 내가 튜바 연주를 맡게 되었지요. 단 드래그넷(Dragnet), 하우디 두디 타임(Howdy Doody Time), 브라일크림(Brylcreem)같은 TV 프로들서 나온 온갖 종류의 테마들을 결합시켜 밴드를 위한 소품으로 편곡을 했습니다. 당시 나는 저작권법에 대해 전혀 모르고 있었지요.*

음악가가 되겠다던 그의 계획이 달라진 것은 케이스 연구소(Case Institute)[1]로 부터 물리학 장학생 제의를 받으면서였다.

> *그 시스템은 과학에 소질이 있는 사람이면 누구나 물리학으로 관심을 돌리도록 만들었습니다. 당시에는 제2차 세계대전의 전후 상황하에서 그 분야가 크게 활성화되고 있었지요.*

크누스는 고등학교 시절 수학이 따분한 과목이라는 것을 깨달았다. 하지만 케이스에서 신입생들의 계산법을 담당했던 폴 겐터(Paul Guenther)는 그에게 전공을 물리학에서 수학으로 바꾸도록 설득하였다. 그 과정에서 겐터는 크누스의 믿을 만한 지도자가 되었다.

> *난 그 때까지는 한번도 수학자를 만나 본 적이 없었습니다. 그분은 유머감각이 뛰어 났지만 누가 어떤 이야기를 해도 영향을 받지 않는 사람이었어요.*

1956년 크누스는 난생 처음 우연히 컴퓨터를 만나게 되었다. 그것은 포트란 이전의 기계인 IBM 650이었다. 그는 밤새도록 매뉴얼을 읽어 혼자 힘으로 기본적인 프로그래밍을 익혔다.

> *우리가 IBM에서 얻은 매뉴얼들은 프로그램의 실례를 보여주었습니다. 난 내가 그것보다 훨씬 더 잘 할 수 있다는 것을 알았습니다. 그래서 내게 어느 정도 재능이 있다고 생각했지요. 그 당시 난 거의 누구라도 그 프로그램들을 개선할 수 있을 것이란 사실을 깨닫지 못했던 것입니다. 사실상 기존의 책들은 말도 안 되는 수준이었거든요. 내가 컴퓨터를 배우기 시작한 때는 정확히 배커스가 포트란의 개발로 가장 분주했던 바로 그 시점이었습니다.*

크누스의 첫 번째 프로그램은 숫자들을 소인수분해하는 것이었다. 또 하나의 프로그램은 컴퓨터에서 3목놀이(Tic-tac-toe) 게임법을 가르치는 것이었다. 하지만 그것은 단지 빈둥거리며 시간을 보낸 것에 지나지 않았다. 1958년 그는 케이스의 야구팀을 위한 프로그램을 하나 제작하였다. 그 프로그램은 실책, 도루, 득점 등의 기준들을 토대로 각 선수들의 등급을 매기는 것이었다. 코치는 그 프로그램을 인상 깊게 받아들여 나중에 그것이 팀의 리그 챔피온십을 따내는데 도움이 되었다고 주장하였다. '뉴스위크(Newsweek)'지는 그 프로그램에 대한 기사를 실었고, IBM은 650옆에서 포즈를 취하고 있는 크누스의 사진을 광고에 사용하였다.

크누스는 자신이 기계가 실행하도록 만들 수 잇는 것에 완전히 마음을 빼앗겼다. 나중에 심지어 그의 음악적 관심에까지도 컴퓨터가 잘 맞는다고 생각하게 되었다.

> *수학은 패턴의 과학입니다. 그리고 음악역시 패턴들이지요. 컴퓨터 과학은 추상적인 작업과 패턴을 만드는 것에 많은 관련을 가지고 있습니다. 나는 컴퓨터 과학에서 다른 분야들과 비교하여 가장 차이가 나는 부분은 무엇보다도 지속적으로 차원이 급상승 하다는 점이라고 생각합니다. 즉, 무언가를 미시적으로 보는 데에서 거시적으로 보는 방향으로 도약하는 것이지요.*

[1]훗날의 케이스 웨스턴 리저브(Case Western Reserve)

수많은 직업이 인지된 필요를 그 토대로 하고 있으며, 사람들은 결정적인 문제들을 해결할 방법을 찾아냅니다. 예를 들면, 의료 분야의 직업들이 그렇지요. 반면에, 그 밖의 컴퓨터 과학과 같은 직업들은 어린 시절에 개발되는 의식 구조의 형태에 따라 선택됩니다.

만일, 전체 인구 가운데 어떤 2퍼센트의 그룹에 속하는 사람이라면 컴퓨터에 반향을 일으키기 쉬운 정신적 소양을 가지고 있는 것이며, 선천적으로 컴퓨터 과학에 끌리도록 타고 난 것이라고 할 수 있습니다. 그것이 바로 우리를 다른 사람들과 구별하는 사고 방식이지요. 결국 나는 내가 컴퓨터 과학자라는 것을 알게 되었습니다.

크누스는 1960년 최우등으로 케이스를 졸업하였다. 또한, 전례 없는 교수단 투표에 의해 수학 석사 학위도 동시에 받을 수 있었다. 그는 이어서 캘리포니아 공과대학에 들어가 3년 후 수학 박사 학위를 마쳤다. 그는 조합 기하학 분야에서 '유한 세미필드와 사영면 (Finite Semifields and Projective Planes)'이라는 논문을 썼다.

1963년 졸업을 한 뒤 크누스는 수학 조교수로 칼텍 교수단에 합류하였지만 컴퓨터에 대한 관심을 계속 밀고 나갔다. 1960년 부터 크누스는 버로즈사(Burroughs Corporation)의 자문 역할을 맡고 있었다. 버로즈[2]는 당시 컴퓨터 산업을 선도하는 기업으로서 에드스거 다익스트라처럼 기라성 같은 권위자들과 교류를 맺고 있었다.

크누스는 버로즈사에서 하드웨어와 소프트웨어 설계에 모두 관여하면서, 특히 새로 개발된 알골 60 프로그래밍 언어를 지원하였다. 이 작업을 하면서 그는 다익스트라와 개인적으로 접촉하는 기회를 가질 수 있었다. 그것은 컴파일에 대한 두 사람의 공통된 관심덕택이었다. 다익스트라는 조네벨트(J. A. Zonneveld)와 함께 1960년 8월 알골 60 컴파일러의 첫 실행을 완성한 바 있다.

우리는 직접 만나기도 하고 편지를 주고 받기도 하였습니다. 그의 위대한 강점은 타협하지 않는 미학입니다. 나로 말하자면, 항상 흐릿하게 우유부단한 태도를 보여왔지요. 그가 나에게 내가 하는 것이 마음에 든다고 말하면, 그건 그가 정말 좋아하는 것입니다. 만일 싫다고 말하면, 그가 정말로 마음에 들어하지 않는 것이라고 이해하면 됩니다. 바로 그런 점이 그를 아주 귀중한 통신자로 만듭니다.

당시에는 수학과 컴퓨터 과학 사이에 상당한 거리가 있었습니다. 프로그램을 작성할 때면 그것이 유효할 것이라는 생각이 들때까지 계속 만지작 거려야 했습니다. 하나의 프로그램이 유효할 것인가를 증명하는 데 수학을 사용한다는 발상, 그것은 당시로선 아주 급진적인 인식이었기 때문에 어느 누구도 그것이 가능하다고 믿지 않았지요. 다익스트라는 컴퓨터 프로그램에 관련된 사실들을 증명하는 데 있어서 위대한 선구자들 가운데 하나로 꼽을 수 있습니다.

2. 컴퓨터 프로그래밍의 예술

1962년 1월 크누스가 아직 대학원생이었을 때, 교재 발행인인 어드슨 웨슬리(Addison-Wesley)가 그에게 컴파일러에 관한 책을 쓰도록 권하였다. 당시 그것은 거의 제대로 이해조차 되지 않은 연구주제였다. 크누스는 그 해 여름 그 프로젝트를 시작하여, 1963년 가을에는 칼텍의 제자들을 대상으로 1차 초안을 시험해 보았다.

1966년 까지는 3000페이지 분량의 초고를 마무리하고 그걸 타이핑하기 시작하였습니다. 내가 직접 쓴 글씨체의 크기와 인쇄된 페이지의 글자들을 비교해 본 결과, 손으로 쓸 때는 3000페이지였던 것이 700페이지로 줄어들 것이라는 계산이 나왔습니다. 하지만 발행자 측에는 그 결과를 부정하며, 그 비율이 1대 1이었다고 했지요. 정신 없이 회의를 진행한 뒤 우리는 총 7권의 시리즈를 기획하기로 결정했습니다.

1966년엔 어느 한 개인 단독으로도 컴퓨터 과학분야를 아주 훌륭하게 아우를 수가 있었습니다. 하지만 그 분야는 계속해서 점점 커져왔지요. 난 거기에 뒤떨어지지 않고 따라가기 위해 최선을 다했습니다. 지금 나는 네째 권(조합 알고리즘에 관한 내용). 그러니까 4A,4B,4C로 구성된 그 하권만으로도

[2] 지금은 유니시스(Unisys)에 합병됨

약 2000페이지 분량에 달할 것이라는 걸 알고 있습니다. 그리고 2003년 까지는 그것을 마무리 할 수 있을 것으로 생각하고 있습니다.

갓 학위를 받은 박사가 그처럼 포괄적인 교재를 쓴다는 것도 충분히 놀라운 일이지만, 그에 대한 평가는 훨씬 더 놀랍다. '컴퓨터 프로그래밍의 예술'의 첫 세권은 70년대 전반에 걸쳐 줄곧 교재로 채택되었고 지금까지도 참고 문헌으로 자주 사용되고 있다. 그들의 지속적인 인기는 크누스가 자신의 주제를 다루면서 보여 준 꼼꼼하고 치밀한 태도에서 비롯된 결과이다. 확률이론이 요구될 때면 그 책에서는 모든 세부 사항들을 철저히 다룬다. 어떤 알고리즘에 대한 설명을 하고 난 뒤면 크누스는 그것을 실행하는 프로그램을 제시하여 실례를 보여준다. 그것은 절대로 잘못 이해되는 사례가 생기지 않도록 하기 위한 것이다. 전체 집필과정에서 크누스는 엄격함과 기지를 섞어가며 모든 아이디어의 바탕에 깔려있는 미를 보여주고자 애를 쓴다. 뉴욕 대학의 컴파일러 설계자인 에드 쇤베르그(Ed Schonberg)가 표현한 바와 같이, "다익스트라는 우리에게 잘못된 것으로부터 옳은 것을 식별해 내도록 가르쳐 주었고, 크누스는 굉장한 것에서 그저 그렇고 그런 것을 가려 내는 방법을 가르쳐 주었다."

3. 컴파일러

교재를 집필하면서 크누스는 컴파일러에 대한 연구를 시작하였다. '컴파일러(Compiler)'란 하나의 언어[3]를 다른 언어(타겟 코드)로 번역하는 프로그램이다. 원시 언어는 고급언어, 즉 포트란이나 코볼, C, 혹은 워드 프로세싱이나 그래픽, 스프레드시트언어 등까지 포함하는 현대의 모든 컴퓨터 언어가 될 수 있다. 목적언어란 특정한 컴퓨터가 이해하는 0과 1을 연속적으로 배열한 순서이다. 60년대 초반에는 이 번역을 실행하는 방법이 전혀 명확하지 않았다. IBM에서 활동한 배커스의 포트란 그룹은 4년에 걸친 지난한 노력을 통해 최초의 대형 컴파일러를 작성하였다. 컴파일러라는 주제는 아주 어렵게 간주되었기 때문에, 컴파일러 작성에는 흔히 대학원 과정에서 최고 과정을 지정하는 번호가 부여되곤 했다.

내가 컴파일러에 빠져 들었던 이유는 컴퓨터를 가지고 할 수 있는 가장 놀라운 일이 바로 컴퓨터가 자체의 프로그램을 작성하도록 만드는 것이라 생각했기 때문이었습니다. 컴퓨팅이 컴퓨팅에 적용 될때, 그 때가 바로 컴퓨터 과학이 궁극적인 완벽함에 이르게 되는 시기입니다.

1950년대의 프로그래머들은 대수 표기를 할때 카드에 구멍을 뚫어 그것들을 기계에 주입하였을 것입니다. 그러면 등이 깜빡거리면서 펀치, 펀치, 펀치, 즉 기계가 컴퓨터 명령어들을 뚫어 나가는 것이지요! 그건 굉장했습니다. 난 그 일이 실제로 이루어지고 있으며, 내가 그것이 작동되는 방법을 알아야 한다는 사실을 믿을 수가 없었습니다. 그리고 그것을 이해했을 때에는 훨씬 더 나은 방법들이 실현 가능하다는 것을 알게 되었지요.

사실상 현대의 소프트웨어 도구들은 컴파일러 작성을 훨씬 더 쉽게 만들어 주었다. 그래서 오늘날엔 대부분의 학교에서 같은 과정을 학부생에게 제공하고 있다. 크누스는 이러한 도구들을 개발하는 데 큰 역할을 했다.

컴파일러에 대한 나의 연구에서 가장 잘 알려져 있는 것은 LR(k)파싱이라는 것입니다. 그 발상은 내가 제 10장의 1차 초고를 마친 직후 문득 떠올랐지요. (난 내가 한 권의 책을 쓰고 있다고 생각했다는 것을 기억하십시오) 난 그 때 막 이전에 알려져 있는 것에 대한 조사를 마친 상태였기 때문에, 기본 발상은 아주 자연스럽게 나왔습니다.

크누스가 발견한 것은 파싱작업을 위한 일반적인 방법, 즉 번역을 위한 구조를 찾아내는 것이었다. 파서(parser)의 작업은 한 문자열(어떤 순서로 배열된 단어들)을 취해 그 문자열이 어떠한 문법 규칙에 적용되는지를 판별하여 그것이 번역될 수 있도록 하는 것이다. 효율성을 기하기 위해 실제의 파서는 이 같은 번역을 하나의 패스(pass)에서 수행한다. 다시 말해, 파서는 자신들이 이미 결정해 놓은 사항을 결코 뒤집지 않는 것이다.

[3] 소스코드

하지만 때때로 결정을 내리기 어려운 경우들이 있다. 예를 들어, 두개의 영어 문장 'Flying planes is fun'과 'Flying planes can crash'를 생각해 보자. 첫번째 문장의 경우 'Flying planes'의 적절한 해석은 '날고 있는 비행기들의 행동'이 될 것이며, 두 번째 문장에서는 그것을 '대기 중에 떠 있는 비행기들'이라고 해석하는 것이 적합할 것이다. 이러한 해석들은 'Flying'과 'planes'에 지정된 문법의 범주로부터 나오는 것이다.

이와 같은 문장들을 파싱하는 문제를 해결하기 위해 크누스는 '룩어헤드 (lookahead)'라고 하는 기존의 기술을 사용하였다. 예견을 사용하면 일단 'Flying planes'라고 하는 표현이 될때 파서는 그 문자열에서 보다 앞쪽을 보고 어떠한 문법적 해석을 적용해야 하는 지를 결정할 것이다. 크누스의 알고리즘은 이전의 방법들에 비해 훨씬 많은 언어들을 처리할 수 있다.

4. 속성 문법(Attribute grammar)

크누스는 프로그래밍 언어 문장의 파싱에 대한 연구에 이어, 프로그램들의 의미를 찾는 일반적 방법에 대한 연구를 계속하였다. 그는 배커스와 나우어가 구문에 대해 발견하였던 것만큼 세련된 형식을 찾고 있었다.

> *배커스의 연구는 내게 아주 흥미로운 것이었습니다. 배커스는 컴퓨터 언어들이 이처럼 멋진 구조를 갖고 있으며, 세련된 형식 구문론이 실현가능하다는 것을 인식하고 있었지요. 내가 그것에 완전히 매료당했던 이유는 그것이 수학처럼 보이기도 하는 컴퓨터에 관련된 유일한 것이라는 사실 때문이었습니다.*
>
> *난 배커스-나우어 형식의 구문에 적합한 의미론을 정의할 수 있는 훌륭한 방법을 찾게 되길 바랬습니다. 속성 문법의 개발은 당연한 것이었지요. 바로 그 문법들이 의미의 직관적 개념들에 꼭 들어 맞았기 때문입니다.*

크누스가 이연구를 시작하기 전까지는 어떤 프로그래밍 언어의 의미에 대해 가장 경제적인 설명이라는 것이 결국 수천 행에 달하는 그 언어의 번역기 아니었던가! 그것은 그 구문론에 대한 배커스-나우어의 설명이 보여 준 세련된 경제성과는 현격한 대조를 이루는 것이엇다. 크누스는 식의 규칙들(그가 속성 규칙이라고 불렀던)을 문법의 규칙들과 연결시킬 방법을 생각해 내었다.

예를 들면, $x = y/z$ 와 같은 대수식에서 속성문법은 y/z의 파스 트리 노드에서 구문 명령에 호출을 첨가시키며, $x + y/z$의 파스 트리 노드에서는 가산에 호출을 첨가시킨다(그림 1 참조). 따라서, 컴퓨터 프로그램의 의미 (이 예에서는 x를 y/z의 몫에 더하는 것)는 그것을 구성 부분들로 형성, 즉 '합성(synthesized)'된다. 프로그래밍 언어 이론가 네드 아이언즈(Ned Irons)는 1960년 단순한 언어들을 위해 이 발상을 제안하였지만, 크누스가 아는 바로는 합성은 충분치 못하였다. 복잡한 언어로 된 프로그램의 경우 한 프로그램 내에서 이름들이 의미를 정확히 해석하기 위해서는 반드시 알아야 하는 문맥의 정보들이 있다. 1967년의 한 대화를 통해 피트 웨그너(Peter Wegner)는 정보를 파스트리 아래에서 위쪽으로 합성하는 것 뿐만아니라 아래쪽 방향으로도 전달할 것을 제안하였다. 크누스는 처음엔 이것을 터무니 없는 발상이라고 생각하였으나 이후 그것을 사용할 방법을 찾아 내었고, 그 결과 '이어받는'속성들이 탄생되었다.

오늘날 컴파일러의 기술은 대부분 이러한 통찰에서 비롯된 것이다.

5. 정밀한 분석

크누스는 컴파일에 대한 작업을 하면서 부터 알고리즘, 즉 컴퓨터가 빈번하게 요구되는 작업들을 완수하기 위한 효율적인 방법들에 대한 연구를 시작하였다. 가장 기본적인 알고리즘 가운데에는 리스트상의 숫자나 이름의 소팅을 위한 알고리즘들이 포함되어 있다. 그와 같은 소팅은 숫자나 이름들을 순서대로 정렬하기 위한 것일 수도 있고, 혹은 어떤 이름에 연관된 번지들을 찾고자 리스트를 탐색하기 위한 것일 수도 있다. 크누스는 'The art of computer programming'시리즈 가운데 한권을 바로 이 작업에

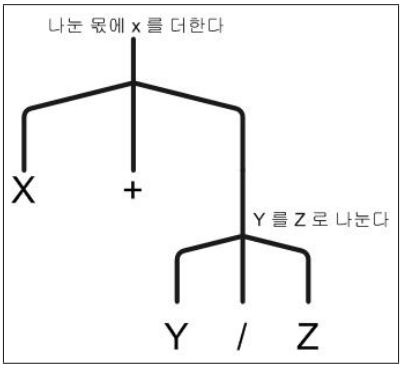

Figure 1: $x+y/z$의 파스트리 합성속성들(글자로 표시된)은 파스트리의 아래에서 위쪽으로 식의 의미를 전달한다.

대한 내용에 할애하였다. 하지만 이 영역에서 그가 기여한 바는 주로 알고리즘의 '정확한 분석'에 관한 것이다.

아마도 다른 컴퓨터 과학자들은 알고리즘이 소요하는 시간이 그것의 입력 제곱에 비례한다고 알고 있었을 시기에 크누스는 그 시간이 정확히 입력 제곱의 3.65개가 걸린다는 것을 증명하려고 했다.

> *내가 알고리즘의 분석을 수행하는 방법은 추측컨데 이 분야에 속한 대다수의 연구자들과 다릅니다. 내 연구의 가장 독특한 부부은 어떤 것이 다른 것보다 10내지 15퍼센트 더 낫다고 말할 수 있다는 것입니다. 그것은 기호와 기질의 문제이지요. 난 내가 받은 훈련의 영향으로 작은 그림을 보는데 우위가 있는 것이고, 다른 사람은 큰 그림을 보는 데 우위가 있는 것입니다.*

그러나 법칙 제공자로서 크누스는 자신의 컴파일러에 대한 연구를 훨씬 뛰어 넘는 아주 중대한 탐구를 해 왔다. 예를 들어, 그는 칼텍 학부과정에서 가르친 제자 피터 벤딕스(Peter Bendix)와의 공동연구를 통해 수학적 공리들의 결과들을 조사하는 알고리즘을 만들어 내었다.

> *수학자들은 군(group)이라는 추상적 구조에 대한 공리들을 배웁니다. 그리고는 그 공리들의 수많은 결과들을 증명하지요. 어느 날 나는 컴퓨터가 그러한 증명들을 체계적으로 발견하도록 만들어 질 수 있다는 사실을 우연히 발견하였습니다. 군에 대한 세 가지 공리를 가지고 출발하여 일곱가지 결과를 도출해 낸다면, 그 다음엔 이 열가지가 '완전'하다는 것을 보여줄 수 있습니다. 누구든지 처음의 세 가지에서 이끌어 낼 수 있는 모든 항등식이 아주 단순한 절차를 통해 그 열가지에서 도출될 수 있는 것이지요.*

알고리즘의 기본적인 발상은 등식의 형태로 표현된 공리들을 가지고 시작하여 그것들을 '환원(reductions)'으로 간주하는 것이다. 예를 들면, 공리는 $a \times (b \times c) = (a \times b) \times c$ 그리고 $a \times 1 = a$가 될 수 있다. $a \times (b \times c) \rightarrow (a \times b) \times c$ 와 $a \times 1 \rightarrow a$ 와 같이 환원을 생성한다. 대표적인 결론은 $a \times (1 \times b) = a \times (b \times 1)$이 될 것이다. 그 바탕에 깔려 있는 발상은 두개의 식 x 와 y 모두가 어떤 세번째 '정규(canonical)'식 z 로 환원된다는 것을 보임으로써 이 두식이 동일하다는 것을 보여주는 것이다. 알고리즘은 x에 더 이상 환원이 적용되지 않을 때까지 임의의 순서로 환원을 적용하여 z를 찾아낸다. y에 대해서도 마찬가지이다. 위의 예에서는 $a \times (1 \times b) \rightarrow (a \times 1) \times b \rightarrow a \times b$이며, 마찬가지로 $a \times (b \times 1) \rightarrow a \times b$ 이다. 크누스-밴딕스 알고리즘은 초기의 집합이 불완전할 경우 새로운 환원을 생성할 것이다. 예를 들면, 추가적인 공리 $a \times a' = 1$은 자동적으로 $(a \times b)' \rightarrow b' \times a'$와 같은 환원을 이끌어 낼 것이다.

물리학자에서 정치학자에 이르기 까지 과학자들은 항상 공리 시스템을 고안해 낸다. 크누스-밴딕스 알고리즘은 그같은 공리들에 함축된 의미를 조사하고 그것을 사용하는 증명들을 검증하는데, 실제로 과학자들이 직접 만들어 낸 방법보다 더 정확한 경우가 많다.

1968년 크누스는 스탠퍼드 대학으로 옮겨갔다. 그 학교는 이미 세계 3대 컴퓨터 과학과로 (M.I.T., 카네기 멜론대학과 함께) 손꼽히고 있었다. 그는 대학원생 본 프래트(Vaughn Pratt)과 함께 어떤 문자

열을 찾기 위해 텍스트들을 검색하는 단순하고도 지극히 효율적인 방법을 발견하였다. 거의 같은 시기에 제임스 모리스도 유사한 방법을 발견해 지금은 그것을 크누스-모리스-프래트 알고리즘이라고 한다.

긴 텍스트에서 'init'이라는 문자열을 찾는 문제를 생각해 보자. 명백한 방법은 텍스트의 문자들을 연속적으로 검사하여 i를 발견하게 되면 다음의 문자를 검사하고, 그 문자가 n일 경우 또 그다음 문자를 검사하는 식으로 계속해 나가는 것이다. 여기서 아주 골치아픈 부분은 한동안 부합되는 걸 찾아가다가 불일치를 발견하게 되는 경우 어떻게 해야 하는가 하는 문제이다. 예를 들어서 'isininity ...'라는 텍스트가 주어진다면 그 텍스트의 첫 문자에서 일치하는 것을 찾는 것으로 시작하겠지만, 곧이어 두번째 문자에서는 불일치를 검출하게 될 것이다. 세번째 문자에서 다시 시작하면 네번째와 다섯번째 문자에서 계속 일치 사항을 찾게 되지만, 또 다시 여섯 번째에서 불일치를 검출하게 된다. 이 시점에서 그대로 일곱번째 문자로 진행하여 'init'의 첫 i를 찾는 다면 그것은 오류가 되고 만다. 그럴 경우 'ity ...'를 찾게 되어 결과적으로 그 문자열의 어디에도 'init'가 없다는 잘못된 결론을 내리게 될 것이기 때문이다. 그리고는 세번째 문자에서 시작해 부합하는 것을 찾는데 실패했기 때문에, 아마도 네번째 문자에서부터 다시 비교를 시작하는 방법으로 이 문제를 해결할 것이다. 그럴 경우 문제가 되는 것은 그 텍스트의 네번째, 다섯번째, 여섯 번째의 문제들은 두 번씩 검사하게 된다는 점이다. 이것은 비효율적이다. 특히, 패턴의 길이가 길 경우엔 보다 많은 문자드을 거듭하여 검사하게 되므로 효율성이 더 떨어지게 된다.

크누스, 모리스, 프래트 이 세사람은 정확하면서도 효율적인 알고리즘을 찾아 내려고 하였다. 그들은 로버트 보이어(Robert Boyer)와 스트로터 무어(G. Strother Moore)가 발전시킨 이 아이디어 들과 스트브 쿡(Steve Cook)이 개척한 오토마타(Automata)또는 자동 기계이론의 진보로 부터 영감을 얻었다.

> *우리의 논문에는 실제로 '오토마타 이론은 유용할 수 있다.'라는 제목이 붙여질 생각이었습니다. 왜냐하면, 그 방법은 일반적으로 프로그래머가 생각하는 방법이 아니기 때문입니다. 우리는 오토마타 이론에 대한 이해를 통해 우리가 나아갈 방법에 관련된 중요한 실마리를 얻게 되었습니다.* [4]

그 알고리즘은 그것이 단순화된 유한 상태 오토마타 혹은 유한 상태 기계를 에뮬레트 하게 만드는 표를 작성한다. (라빈의 장 109쪽 참조).

이들은 상태와 전이를 가진 이론상의 기계들이다. 일부 상태는 시작상태(기계가 처리를 시작하는 상태)라 부르며, 또 다른 일부 상태는 인정상태(accepting state: 기계가 어떤 패턴을 발견하였음을 선언하는 상태)라 부른다. 이같이 단순화된 형태에서 각각의 상태는 문자열에서 부합되는 문자의 수에 상응한다.

따라서, 4글자 패턴 'init'의 경우, 상태 0에서 시작하고 상태 4에서 선언을 하게 된다. 전이란 어떤 하나의 상태에서 다음 번 입력 문자에 따라 다른 하나의 상태로 옮겨 가는 것을 나타내는 개념이다. 'isininity'라는 문자열에서 패턴 'init'을 검색할 때 기계가 수행하는 작업은 다음과 같다.

문자	결과 상태
i	1
s	0
i	1
n	2
i	3
n	2
i	3
t	4,부합
t	0

첫번째 문자가 'i'이기 때문에 기계는 상태 1로 나아가지만, 두번째 문자가 's'이기 때문에 상태 0 으로 되돌아간다. 대부분의 알고리즘이 이런 식일 것이다. 크누스-모리스-프래트 알고리즘이 간파한 핵심

[4] 최종적으로 채택된 제목은 보다 직접적이었다 : '문자열에서 패턴의 부합을 빠르게 찾는 방법(Fast Pattern Matching in Strings)'

적인 부분은 맨 앞의 'ini'에 이어 't'를 발견하는데 실패할 경우 이전의 문자들을 재고하기 위해 다시 후진할 필요가 없다는 것이다. 결국 이 알고리즘은 앞 문자들의 재고를 피하는 것이며, 이러한 특징은 이론상만이 아니라 실제적으로도 아주 큰 장점이다.

6. 폰트

크누스는 일생동안 인쇄와 그래픽의 구조에 흥미를 가져왔다. 1940년대의 소낸 시절엔 위스콘신 여름 캠프에서 식물에 대한 안내서를 쓰면서 당시 인쇄에 보편적으로 사용되고 있었던 청복사지(blue ditto paper)에 첨필을 사용해 꽃의 삽화를 그렸다. 그리고 그는 대학 시절 자신의 수학 교재에 사용되었던 활자체(Modern)을 보고 감탄했던 것을 기억한다. 하지만 그는 활자의 디자인이나 세팅 부분은 전문가들의 몫으로 남겨두는데 만족하였다.

> *난 한번도 내가 인쇄를 관리하게 될 것이라고 생각한 적은 없었습니다. 인쇄는 핫 리드(hot lead)같은 것을 사용하여 인쇄 기술자에 의해 수행되는 것이었으니까요. 그러다가 1977년 0과 1, 즉 리드없이 비트(bit)로 이루어진 문자들을 인쇄하는 새로운 인쇄기계에 관해 알게 되었습니다. 갑자기 인쇄가 컴퓨터 과학의 문제로 떠올랐지요. 난 내가 교재를 집필하는데 사용하는 새로운 기술로 컴퓨터 도구들을 개발한다는 도전을 물리칠 수가 없었습니다.*

크누스는 결국 9년에 걸친 긴 시간동안 다른 프로젝트들은 뒤로 미루어 둔 채 디지털 인쇄 방식에 사용될 두가지 컴퓨터 언어를 설계하고 실행하는데 전념하였다. 그 첫번째 것은 텍(TEX)이라고 불리는 것으로, 문자와 그 밖의 기호들을 인쇄면 위에 배치한다. 다른 하나인 메타폰트(METAFONT)라고 하는 것은 글자 자체의 형태를 정의한다. 이들 프로그램은 현재 자유소프트웨어(Free software)로 전세계 어디에서나 사용이 가능하다. 그리고 실제로 이 책의 발행자를 포함해 백만이 넘는 수의 사용자들을 확보하고 있다.

크누스는 그 특유의 면밀함으로 인쇄술 분야를 파고 들었다. 예를 들어 그는 '문자 S(The Letter S)'라는 논문을 통해 그 문자의 수학적 형태를 분석하고 자신이 가장 만족스런 방정식을 찾기 위해 며칠에 걸쳐 노력을 들였던 과정을 설명한다.

컴파일러, 알고리즘 분석, 폰트에 이르기 까지 실로 광범위한 그의 업적들 사이에서 어떤 공통된 줄거리가 보이지 않는가? 크누스는 그렇다고 주장한다.

> *오직 필요만이 발명의 어머니 혹은 아버지라고 이야기 하는 것은 사실이 아닙니다. 또 하나의 측면은 바로 그 문제에 적합한 배경을 가지고 있어야 한다는 것입니다. 난 그저 눈에 보이는 모든 문제에 대한 연구를 하면서 돌아다니고 있는 것이 아닙니다. 난 내가 해결하는 문제들에 대해 이렇게 말합니다. "아, 그래. 난 바로 그걸 해결하도록 만들어 줄 수 있는 고유한 배경을 지니고 있는 거야. 그건 나의 운명이자 내게 주어진 책임이지."*

그의 고유한 배경에는 언어와 문법에 대한 사랑, 수학에 대한 명석하고도 광범위한 지식, 시각적인 강한 미적 감각, 이해에 대한 의지, 프로그래밍에 대한 애정 등이 포함되어 있다. 크누스에게서 마지막 두 가지는 아주 밀접하게 연관된 것이었다.

> *일반적으로 무엇이든지 배우려고 할때, 만일 그것을 컴퓨터에게 설명하고자 시도하는 것을 상상해 볼 수 잇다면, 그 주제에 대해 자신이 모르고 있었던 것을 배우게 될 것입니다. 그건 적절한 질문을 제기하는 데 도움을 주지요. 그건 바로 자신이 무엇을 알고 있는가에 대한 궁극적인 테스트입니다.*
>
> *예를 들어, 음악 이론은 무엇이 듣기 좋으며 무엇이 그렇지 않은지에 대한 주관적인 답보다는 객관적인 답을 얻어 내려는 목적에서 발전되었습니다. 우리는 모차르트가 하모니 때문에 듣기 좋다는과 같은 사실들을 알고 있습니다. 하지만 정말 좋은 음악을 만들어 낼 기계를 얻고자 애쓰면서 컴퓨터 프로그램의 작성에 대해 생각한다면 그러한 규칙들이 얼마나 불완전한지 모릅니다.*

이제 스탠퍼드의 젊은 명예교수가 된 크누스는 '컴퓨터 프로그래밍의 예술' 가운데 순열조합을 집필하는 것으로 대부분의 시간을 보내고 있다. 하지만 그에겐 음악을 더 쓰라고 손짓하는 파이프 오르간 처

럼 다른 유혹들도 여전히 남아있다. 그의 문학에 대한 관심은 아마도 그로 하여금 또 다른 소설을 쓰도록 강요할 것이다.[5] 또, 세계적 수준의 회의나 자문 프로젝트들은 그의 주의를 다른 곳으로 돌려 놓는다. 재능과 열정, 그리고 전설에 남을 만한 연구 질을 제하고 나면, 그의 광대한 업적을 이룬 비결은 아무것도 없다.

> *나는 한번에 한가지 일만 합니다. 이것은 바로 컴퓨터 과학자들이 일괄 처리(batch processing)이라고 부르는 것이지요. 다른 대안은 스왑-인 (swap-in)과 스왑-아웃(swap-out)을 실행하는 것인데, 난 스왑-인이나 스왑-아웃은 하지 않습니다.*

어떻게 전설을 낳았는가?

많은 사람들의 입을 통해 크누스는 역사상 가장 위대한 컴퓨터 프로그래머라는 전설이 전해지고 있다. 앨런 케이가 들려주는 다음과 같은 일화를 생각해 보자.

> *내가 스탠퍼드에서 AI 프로젝트를 맡고 있던 시절(1960년대 후반) 매년 추수 감사절 때마다 가졌던 행사 중에 하나는 그 샌프란시스코 만안 지방(Bay area)에서 연구 프로젝트에 종사하고 있는 사람들로 프로그래밍 컨테스트를 여는 것이었습니다. 상품은 칠면조였던 것으로 기억됩니다.*
>
> *문제는 주로 매커시가 출제하였지요. 한 해는 크누스가 그 대회에 참가해서 가장 빠른 시간에 프로그래밍을 실행시킨 사람에게 주는 상과 알고리즘을 가장 빨리 실행하는 사람에게 주는 상을 모두 휩쓸었습니다. 그는 윌버(Wilber)시스템이라는 원격 일괄 시스템을 이용해 최악의 시스템에서 그것을 해 내었지요. 그는 기본적으로 참가자들을 바보로 만들어 버렸습니다.*
>
> *그들은 그에게 이렇게 물었지요. "저, 혹시 어떻게 이렇게 이 일을 할 수 있습니까?" 그의 대답은 이런 것이었습니다. "내가 프로그래밍을 배울때 당신들이 만일 하루에 5분씩만 그 기계와 시간을 가졌으면 좋았을 겁니다. 프로그램이 작동하도록 만들고 싶으면 그저 그것을 제대로 작성하기만 하면 되었지요. 그래서 사람들이 그저 돌에 조각을 하는 것처럼 프로그래밍을 배웠습니다. 그냥 그것을 향해 조금씩 다가가기만 하면 됩니다. 그것이 바로 내가 프로그래밍을 익힌 방법입니다."*

[5] 그가 1974년에 내놓은 '초현실적인 숫자들 Surreal Numbers'는 수학적 체계를 발전시킨 두 명의 대학 중퇴자들의 이야기 이다.