한국학교수학회논문집 제 1 권, 제 1 호
Journal of the Korean School Mathematics Society
Volume 1, Number 1, 19-28, 1998

수학과 컴퓨터

이 영 환[1)]

사회 각 분야에 널리 분포되어 있는 컴퓨터는 현대사회에서 중추적 역할을 담당하고 있으며 컴퓨터는 인공위성에서부터 작은 회사의 업무에 이르기까지 모든 사회기술의 결정적 요소가 되고 있다. 컴퓨터가 현대인들의 일상적인 생활 방식을 바꾸고 있으며 사고의 폭도 하루가 다르게 변화시키고 있다.

계속되는 컴퓨터 기술의 혁신적인 발전은 정보통신기술의 발달을 가져왔으며 우리 인간의 생활을 정보를 중심으로 하는 정보화 사회로 접어들게 하였다. 정보에 대한 새로운 가치판단으로 정보를 중요한 상품으로 간주하고 있으며 광범위한 인터넷의 활용으로 전세계가 동시에 정보를 공유하게 되고 시간과 공간의 차이를 무색하게 하며 더 나아가 컴퓨터로 세계가 모든 의사 소통을 할 수 있는 날이 오게 될 것이다. 또한 멀티미디어에 관련된 기술의 발달로 동영상을 비롯한 여러 가지 미디어를 통하여 컴퓨터의 활용 범위가 급속도로 확장되어지고 있다.

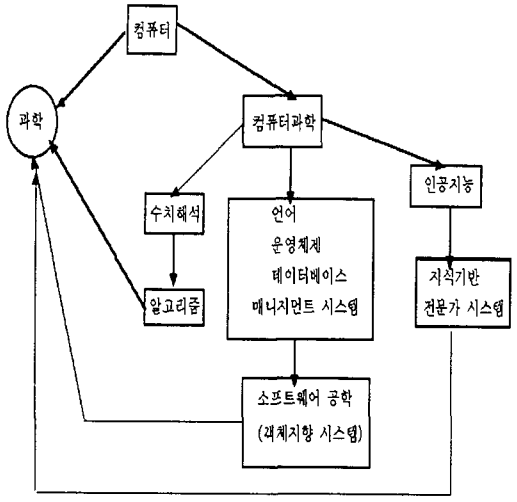
본 글에서는 이러한 컴퓨터는 어떤 과정을 통하여 과학에 응용되고 있으며 인간의 계산과 컴퓨터의 처리 과정은 어떤 단계를 거치는가를 알아보고, 초창기부터 많은 수학자들이 컴퓨터 발전에 큰 기여를 해왔는데 과연 "수학과 컴퓨터와의 연관 관계는 어떠한가?"와 "수학이 컴퓨터에 기여하고 있는 정도는 어떠한가?"를 조사해보고자 한다.

더 나아가 "컴퓨터의 계산능력은 어느 정도이며 그 한계점은 무엇인가?", "생각하는 컴퓨터는 가능한 것인가?", "가장 효과적인 소프트웨어 개발 방법은 무엇인가?" 등에 대하여 종합적으로 살펴보고 이에 대한 수학의 역할을 알아보고자 한다.

1. 컴퓨터가 과학에 응용되는 과정

컴퓨터는 이제 과학의 모든 분야에 일상적으로 쓰이고 있다. 컴퓨터의 종류는 데스크탑 개인용 컴퓨터에서 주 슈퍼컴퓨터에 이르기까지 다양하다. 컴퓨터가 하는 일은 문서작성이나 도안작성 기본적인 모델 분석이나 디자인을 비롯하여 날씨분석, 비선형 편미분 방정식에 의하여 해결할 수 있는 유체역학 문제와 같은 수학적 모델을 알고리즘을 통하여 해결하는 것 등 다양하다. 컴퓨터가 보통 과학에 응용되고 있는 현황은 인터넷이나 전자도서관을 통한 자료검색, 수치적인 코드와 알고리즘, 그래픽, 스프레드시트 및 데이터베이스 등에 이용하고 있다. 이에 따라 컴퓨터 과학의 발전은 전체 과학의 발전에 기여하게 된다. 이러한 일련의 사항을 딤과 레빗[5]이 제시한 컴퓨터의 응용과정에 대한 그림1을 참고로 하여 알아보자.

1) 대전대학교 수학과

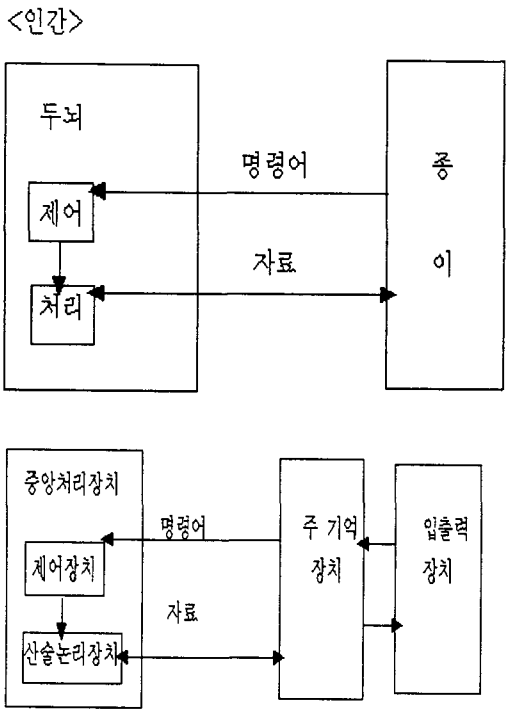


[그림 1]

위 그림에서와 같이 컴퓨터가 과학에 직접적으로 이용되기도 하고 컴퓨터과학이라는 학문을 통하여 연구되어지는 여러 분야가 있는데, 수치해석학 이론을 바탕으로 알고리즘을 통하여 과학에 응용되기도 하며 언어. 운영체제, 데이터 베이스, 매니지먼트 시스템 등의 종합적인 컴퓨터 공학의 연구를 토대로 한 소프트웨어 공학이 객체지향 시스템에 의하여 과학에 응용된다. 한편 인공지능 분야의 연구가 지식 기반 전문가 시스템에 의하여 과학에 응용되기도 한다.

2. 인간의 계산과 컴퓨터의 처리과정

우리가 연필과 종이를 가지고 계산하는 것과 컴퓨터가 처리하는 과정을 비교하여 보자. 우리는 종이에 기록되어 있는 데이터와 계산 알고리즘을 통하여 계산을 하고 계산이 처리되는 과정은 우리의 뇌를 통해서 수행하게 되는데 뇌는 정확한 순서로 계산이 수행되도록 하는 제어기능과 사칙연산을 비롯한 계산을 수행하는 처리기능을 가지고 있다. 반면에 컴퓨터는 종이에 대응하는 기억장치를 사용하고 뇌에 해당하는 중앙처리장치가 있어서 이것은 프로그램을 제어하는 제어장치와 명령어를 수행하는 산술논리장치로 구성되어 있다. 이것을 그림2로 살펴보면 다음과 같다.



[그림 2]

결론적으로 컴퓨터와 인간은 다음 요소를 가지고 있다.

1. 프로그램을 해석하고 수행하는 능력을 가진 처리기
2. 프로그램들과 데이터를 저장하기 위한 기억장치
3. 기억장치와 처리기, 그리고 외부 세계 사이에 정보를 전달하는 수단.

3. 수학과 컴퓨터의 연관성

로버트 스레이터[11]가 지은 " Portraits in silicon"(이규창 역 : 컴퓨터 영웅들)을 보면

빌 게이츠를 포함한 34명의 컴퓨터 영웅들의 이야기가 전기 형식으로 소개되고 있다. 이들 중 25%이상이 수학자이었다. 코볼 언어의 어머니로 불리는 호퍼, 포트란 언어를 개발한 존 베커스, 전자출판 시대를 열게 한 도널드 크누스를 비롯하여 시대를 앞서간 컴퓨터의 아버지로 불리는 찰스 배비지, 생각하는 기계를 연구한 알랜 튜링, 컴퓨터와 동의어로 불리는 폰 노이만 등의 수학자들의 공헌이 컴퓨터 발달에 큰 역할을 했다. 이것은 수학과 컴퓨터가 깊은 관련이 있음을 직관적으로 보여주는 결과일 것이다. 실제로 수학에서 대수학은 컴퓨터의 모델을 이끄는 구조적 형태를 제시하고 있으며 수학에서 수치해석학은 직접적으로 컴퓨터를 사용하여 알고리즘을 통하여 문제해결을 하고 있으며 프랙탈 기하학도 컴퓨터 응용 분야로 자리잡고 있다. 그 밖에도 컴퓨터와 연관하여 연구되는 수학분야가 다양하다. 특히 컴퓨터의 계산능력을 밝히는 데는 수학 이론이 직접적으로 연관된다.

컴퓨터에서 언어와 계산이론은 주로 20세기 전반에 걸쳐 공리계와 논리학 분야를 연구하던 수학자들에 의해 연구된 것이다. 수학은 공리론적 방법을 통하여 이론을 전개해 나가고 있다. 공리란 증명 없이 기정 사실로 받아들이는 기본 진술들의 모임이다. 기하학의 공리로부터 기하학 이론 전개라든가 집합론 공리로부터 실수공리까지 다양하게 각 분야마다 공리로부터 출발하여 엄밀하게 증명과정을 통하여 유도되어지는 추가 진술인 정리를 이끌어 내고 있다.

1900년 파리 수학자 대회에서 수학자 힐버트는 전체 수학이 하나의 공리계에 기반을 두고 전개 할 수 있는지 없는지 하는 문제를 던졌다. 이 당시 대부분의 수학자들은 하나의 공리계에 기반을 둘 수 있으리라고 믿었다. 이 후 30년 동안 수학 자체의 근본이 뒤흔들리며 많은 연구가 진행되었고 이것은 아직 태어나지도 않았던 컴퓨터 과학 분야까지 직접적으로 영향을 미치는 것이었다. 하나의 공리계에 기반을 둔 수학연구의 목표는 올바른 공리와 추론 규칙들의 집합을 발견하는 것이었다. 이것은 수학의 어떤 정리든 간에 공리로부터 일련의 추론 규칙들을 유한하게 적용시켜서 유도할 수 있다는 것을 의미하며 알고리즘을 통하여 어떠한 수학적 진술의 진위를 결정할 수 있는 계산 체계로 귀착되는 것이다. 그러나 1931년도에 크르트 괴텔이 불완전성 정리를 발표함으로써 하나의 공리계로 수학을 전개한다는 것이 불가능하다는 것이 판명되었다. 이 불완전성 정리의 핵심은 자연수와 덧셈 및 곱셈연산으로 이루어진 부분적인 수학을 기술하기에 충분한 어떤 적합한 공리계가 있다고 하더라도 그 공리계 자체만으로는 증명할 수도 반증할 수도 없는 수학적 진술이 존재한다는 것이다. 이것은 수학의 정리를 증명하기 위한 일반적 알고리즘이 없다는 것을 말한다. 그래서 알고리즘에 의한 방법으로 달성할 수 있는 것은 어디까지 일까하는 의문이 생기게 되었고 이 문제를 해결하기 위해서 수학자들은 이론적인 계산기계를 설계하고 이 기계의 능력과 한계를 연구하기 시작했다. 이러한 연구의 결과가 계산이론 분야의 골격을 이루고 있으며 계산이론에는 형식언어이론, 계산가능성 이론, 복잡도이론 등을 포함하고 있다. 그리고 이 계산기계는 오늘날 컴퓨터의 모델이 되기 때문에 컴퓨터 학자가 직면하는 계산가능성의 문제와 수학자가 직면한 증명가능성의 문제는 동등하며 다만 관점의 차이가 있을 뿐이다.

수학과 컴퓨터의 정의적 측면을 살펴보자. 컴퓨터란 인간의 생각을 대신해주는 기계로 정의 되어있다. 여기서 기계란 인간의 일을 대신해주는 장치이다. 수학은 생각을 추론형식에 따라 논리 있게 전개하게 한다. 이러한 관점에서도 밀접한 연관관계가 있다.

4. 컴퓨터의 계산능력과 한계점

컴퓨터로 문제를 해결하기 위해서는 그 문제를 푸는 계산과정, 즉 알고리즘의 개발이 필요하다. 따라서 "컴퓨터를 통해서 무엇을 할 수 있는가?"하는 문제는 "계산과정을 통해서 무엇을 할 수 있는가?"하는 문제와 동일하게 된다[6].

계산능력을 알아보기 위해서 함수적인 접근 방법을 알아보자.

계산이란 x가 주어진 입력데이터이고 $y=f(x)$가 원하는 출력데이터 일 때 어떤 함수 $f(x)$ 의 값을 구하는 과정으로 생각할 수 있다. x와 y 그리고 함수 f 는 매우 다양한 형태를 취할 수 있다. 예를 들어 x 와 y 는 숫자, 단어, 문장, 정보파일 등을 나타낼 수 있고 f 는 수치계산이나 어떤 이론의 증명 등을 나타낼 수 있다. $f(x)$를 계산하려면 컴퓨터의 명령어를 사용하여 순차함수 $f_1, f_2, \cdots, f_n$ 으로 f 를 표현하여야 한다. 컴퓨터 설계에 대한 연구에 있어서 제기되었던 의문점은 계산 기계가 합리적으로 수행할 수 없는 계산이 존재하는가 하는 것이다.

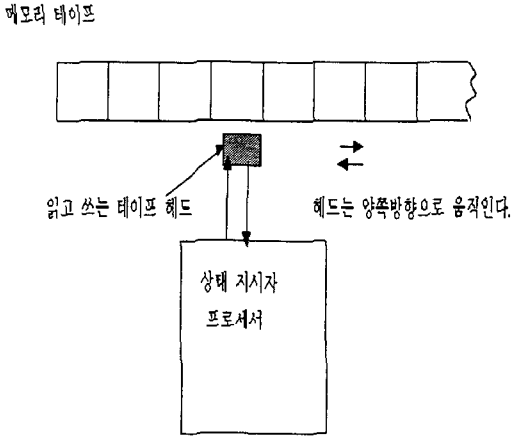
여기서 합리적이란 다음 두 가지를 말한다.

1. 컴퓨터는 문제들에 대한 모든 가능한 해답들을 저장하고 있지 않아야 한다.
2. 기계가 정보를 처리하는 속도는 유한하여야 한다.

이러한 조건을 만족하는 컴퓨터가 주어진 계산을 제한된 단계를 거쳐 답을 구할 수 있는 경우에만 주어진 계산을 수행할 능력을 갖추고 있다고 판단한다.

이러한 합리적인 컴퓨터의 추상적인 모형이 1936년 영국의 수학자 튜링에 의해 소개되었다. 튜링의 기계는 계산과정의 모든 능력을 파악하기 위해서 설계되었으며 계산이라고 생각할 수 있는 어떤 과정도 모형화 할 수 있는 시스템을 개발하기 위한 것이다. 아직까지 튜링기계보다 더 강력한 계산기계를 정의 할 수 없었고 어떤 계산 과정의 능력도 튜링기계의 부류 내에 포함된다는 가설은 일반적으로 오늘날의 컴퓨터 학자들에 의해 받아들여지고 있다[9,12].

모든 컴퓨터의 기본적인 요소들은 기억장치와 처리기이다. [그림 3]과 같이 튜링기계의 기억장치는 길다란 사각구역으로 나뉘어진 길이라 무한한 테이프이다. 각 구역은 공백이거나 기호들의 유한집합 중의 하나를 갖는다. 처리기는 유한한 수의 내부적인 배치로 된 디지털 기계이다. 이것은 테이프구역에 있는 내용을 바꾸고 그 테이프를 현재의 위치에서 좌 또는 우로 한 구역 올라가는 기능을 갖는다. 튜링의 가장 두드러진 연구 결과 중의 하나는 모든 형태의 계산을 수행할 수 있는 만능 튜링기계가 존재한다는 사실을 증명하였다는 것이다.



[그림 3]

"기계적인 수단에 의해 수행될 수 있는 모든 계산은 튜링기계에 의해서도 수행될 수 있다."는 튜링의 논제가 있다. 그러나 이것은 명확하게 증명될 수 없고 엄밀하게 이 논제를 증명하기 위해서는 먼저 기계적인 수단

의 의미를 명확하게 정의해야 한다. 튜링의 논제에 따라서 어떤 계산이 튜링기계에 의해 수행될 수 있으면 기계적이라고 말한다.

튜링의 논제를 기계적 계산의 정의로 여기는 몇 가지 근거는 다음과 같다,

1. 현재의 디지털 컴퓨터에 의해 수행이 가능한 계산은 튜링기계에서도 가능하다.

2. 튜링기계로는 해결이 불가능하나 컴퓨터 알고리즘으로는 해결할 수 있는 프로시저의 예를 아직까지 아무도 제시하지 못 했다.

3. 지금까지 기계적 계산을 위한 여러 가지 대체 모델이 제시되었지만 그 어떤 것도 튜링기계의 능력보다 우수한 것은 없었다.

이러한 근거로 볼 때 튜링기계의 논제는 명확히 증명될 수 없는 가설 상태에 머물러 있다. 그럼에도 불구하고 튜링기계는 컴퓨터와 관련된 분야에서 매우 핵심적인 역할을 담당하고 있다.

X를 자연수의 부분집합으로 생각하고

$$f(n)=\begin{cases}1 & n\in X\\ 0 & n\notin X\end{cases}$$

로 생각하자. 이렇게 정의되는 함수들은 자연수에서 자연수로 보내지는 함수로써 자연수의 부분집합 개수만큼 존재한다. 알고리즘에 의해서 계산할 수 있는 것들은 셀 수 있어야 하는 데 수학적으로 자연수의 부분집합들의 개수는 셀 수 없는 개수이다. 따라서 튜링기계로 계산할 수 없는 함수가 존재한다.

우리가 사용하는 프로그래밍 언어를 생각하여 보자. 이 언어로 어떤 프로그램을 작성할 때 사용할 수 있는 기호들의 숫자는 유한하다. 이러한 기호들에 대해 알파벳처럼 순서를 붙여보자. 그런 다음 먼저 하나의 기호로만 이루어진 프로그램들을 알파벳순서로 나열하고 다음에는 길이가 2인 프로그램들을 사전 식으로 나열하고 계속하여 길이가 3인 프로그램 순서로 나열하여 보자. 그러면 이 언어로 작성될 수 있는 어떠한 프로그램이라도 앞에서 열거한 프로그램들의 모임에 속하게 된다. 이것은 이 언어로 쓰여질 수 있는 모든 프로그램들의 집합이 셀 수 있는 가산집합임을 뜻한다. 즉 알고리즘에 의해서 계산할 수 있는 것들은 셀 수 있어야 하는데 계산하기를 원하는 함수들의 집합은 셀 수 없는 비가산 집합인 것이다. 각 프로그램에 하나의 함수를 계산할 때 프로그램들의 집합이 함수들의 집합보다 작을 수밖에 없다. 따라서 우리가 사용하는 언어로는 이러한 프로그램을 작성할 수 없다.

튜링기계로 계산이 가능한 함수는 기초함수와 기초함수들의 유한회수의 조합, 합성, 순환을 적용함으로서 구성되는 기초순환함수 그리고 기초순환함수에 최소화를 적용한 부분순환함수들이다.

해결 불가능 문제의 존재는 아주 주요한 의미를 가지고 있다. 경험이 없는 프로그래머가 저지르는 일반적인 오류는 무한루프를 포함하거나 또는 어떤 입력조건하에서는 중단하지 못하는 경우이다. 이러한 경우는 튜링기계가 계산할 수 없는 함수로 표현이 가능하므로 오류를 수정하는 디버깅도구가 설계될 수 없다는 사실을 의미한다. 실제 컴퓨터는 유한양의 기억장치를 가진다. 이것은 튜링기계에 의해 수행될 수 없는 어떤 계산은 근본적으로 유한상태 기계들에 의해서는 수행될 수 없다는 것을 의미한다.

속도와 용량문제를 컴퓨터가 다루기 어려운 문제도 생각해 보자.

엄청나게 큰 양의 기억공간을 요구한다던가 긴 시간을 요구하는 문제라면 컴퓨터가 합리적으로 해결하기란 어렵다. 최근 컴퓨터가 작동하는 속도는 빠르게 개선되고 있다. 그러나 기술개선으로 인한 컴퓨터의 속도 증가율은 줄어들기 때문에 속도의 혁혁한 증가는 알고리즘의 개선이나 경험적 지식의 도움으로 이루어져야 생각된다. 병렬처리 방법도 도입하여 컴퓨터 전체적인 연산속도를 증가시키는 방법이 진행되고 있으나 순차처리 해야되는 계산은 여기에 적용 할 수 없다.

5. 컴퓨터의 수학적 모델

컴퓨터로 문제를 해결하기 위해서 컴퓨터 언어를 사용해야 한다.

오늘날 대부분의 프로그래밍 언어는 일반 언어와 마찬가지로 문법에 의해 기술되는 구문을 가지고 있으며 컴파일 과정이 바탕을 두고 있는 것이 바로 이 문법적 구조이다. 우리는 가장 뛰어난 컴퓨터 기계와 이 기계가 문제를 풀기 위해 사용 될 수 있는 언어를 살펴보고자 한다. 이 것을 위해서는 가장 효과적인 이론기계와 이 기계의 언어 인식능력을 알아보아야 한다. 이제까지 아무도 앞에서 언급한 튜링기계 부류보다 더 강력한 부류를 정의 할 수 없었으며 어떤 계산과정의 능력이든지 튜링기계의 부류 내에 포착된다는 가설이 컴퓨터 학자들에 의해 받아들여지고 있다. 튜링기계는 오늘날 컴퓨터의 모델로 다음과 같이 수학에 있어서 대수적 구조로 분석되어진다.

튜링기계는 $(S, \Sigma, \Gamma, \delta, l, h)$ 형태의 6-튜플로 구성되어 있으며 여기서

1) S는 헤드에 의해서 읽고 사용되는 상태들의 유한집합이다.

2) Σ는 기계의 알파벳이라 부르는 비공백 기호들의 유한집합이다.

3) Γ는 기계의 테이프 기호라 부르고 Σ에 있는 기호를 포함한 기호들의 유한집합이다.

4) δ는 기계의 전시함수이다. 즉 $S\times\Sigma\rightarrow S$ 로서 $\delta(p,x)=q$ 라는 것을 기호 x를읽어들이 면서 상태 p로부터 상태 q 로 전이한다는 것이다.

5) l 은 초기상태라 부르는 S의 한 원소이다.

6) h 는 정지상태라 부르는 S의 한 원소이다.

모든 튜링-수락 가능 언어는 문법의 형태가 생성규칙의 구조에 제한을 두지 않는 구문-구조 언어이며 구문-구조언어는 튜링-수락가능언어라는 것이 밝혀졌다[9,12]. 따라서 문법적인 토대를 다진 언어들만이 튜링기계에 의해서 인식된다. 이것은 문법적인 토대를 갖지 않는 언어들은 어떤 알고리즘 적인 과정으로도 인식할 수 없는 것이다. 자연언어가 해석의 모호성을 가지듯이 엄밀한 문법적 구조를 가지고 있지 않기 때문에 이것을 알고리즘 적으로 처리할 수 없게 된다.

이러한 관점에서 컴퓨터와 이 컴퓨터가 인식하는 언어는 동일한 것이며 이들은 모두 수학적 모델을 가지고 있다. 따라서 컴퓨터 학자들이 수학을 컴퓨터의 철학이라고 생각하는 것은 당연하다.

6. 수학에 기반을 둔 생각하는 컴퓨터

기계가 사람처럼 생각하고 정서를 경험하며 의식을 가질 수 있는가 하는 문제는 새로운 것이 아니다. 인간의 사고를 기계화하는 아이디어는 기호논리학에 뿌리를 두고 있다. 기호논리학의 개념은 17세기 독일의 수학자 라이프니츠에서 찾아볼 수 있다. 또한 1854년 사고의 법칙을 출판하여 기호논리학의 탄생을 알린 수학자 부울은 기호논리학의 창시자로 평가된다.

컴퓨터 과학의 측면에서 볼 때 부울의 아이디어에서 가장 중요한 부분은 한 명제에 관하여 참이나 거짓의 두 가지 중에서 어느 하나만 인정하는 것이다, 즉 복잡한 논리식이 0과 1로 표시되는 부울의 아이디어는 컴퓨터 과학의 중추적인 개념이 되었다.

1930년대에 수학자 튜링은 수리논리학을 획기적으로 발전시켰고, 형식개념을 최초로 정립한 오토마타이론은 발표하였다. 그의 형식기계는 인간이 수효가 유한하고 완전하게 명시된 규칙에 의하여 수행될 수 있는 계산을 무엇이든지 적합한 알고리즘을 가진 기계에 의하여 수행할 수 있음을 보여주었다. 튜링기계는 기호조작에 있어서 사람이 할 수 있는 것은 무엇이든지 해낼 수 있기 때문에 오늘날 컴퓨터의 원형이 되었다.

1948년에 발표된 또 하나의 획기적인 논문은 수학자 폰 노이만의 자기증식 기계이다. 튜링기계의 개념을 이용하여 1945년에 오늘날 컴퓨터의 원형인 프로그램 내장식 컴퓨터를 최초로 설계한 폰 노이만은 튜링기계의 계산능력에 덧붙여서 그 자신을 증식하는 능력을 가진 기계에 대하여 연구하였다.

이와 같이 수학에 기반을 둔 인공지능이 독립된 학문으로서 발족을 본 것은 1956년이었다. 미국의 다트머스 대학이 인공지능의 출생지이다. 이 대학의 메카시 교수는 인간처럼 사고할 수 있는 컴퓨터 프로그램의 개발가능성을 검토하기 위하여 민스키, 사이먼, 뉴엘과 모임을 갖고 기호프로그래밍 시대를 열었다. 기호체계의 가설을 요약하면 다음과 같다.

1) 인간의 마음은 정보를 처리하는 체계이다.

2) 정보처리는 계산, 즉 기호를 조작하는 과정이다.

3) 컴퓨터의 프로그램은 기호를 조작하는 체계이다.

4) 인간의 마음은 컴퓨터의 프로그램으로 모형화될 수 있다.

이러한 기호체계 가설은 컴퓨터의 하드웨어는 인간의 두뇌로 보고, 소프트웨어는 인간의 마음에 해당되는 것으로 본다. 인공지능은 60년대 중반까지 일반문제 해결방법에 의하여 컴퓨터 프로그램으로 광범위한 종류의 문제해결을 모의실험 할 것으로 기대하고 연구가 추진되었으나 지각능력, 음성인식, 언어이체, 상식추론 등 어려운 문제에 봉착했다. 70년대 말에 내놓은 해결책은 프로그램의 문제해결능력은 프로그램에 사용된 추론전략에서 나오는 것이 아니라 프로그램이 보유하고 있는 지식의 양에 좌우된다는 것이다. 이러한 개념상의 방향전환에 힘입어 인공지능은 다시금 태어나는 계기를 맞게 되었다. 이후에 지식을 프로그램에 보다 효과적으로 표현하는 기법의 연구가 인공지능의 최대 과제가 되었다. 지식의 표시에 대한 연구의 가장 괄목할 만한 성과로 표출된 것은 전문가 시스템 개발이다. 전문가 시스템은 특정분야의 전문가가 소관 분야의 문제해결에 사용하고 있는 경험적 법칙을 모아놓은 지식베이스와 이것을 사용하여 실제로 문제를 해결하는 프로그램인 추론기관으로 구성된 소프트웨어이다. 전문가 시스템의 성공적인 개발과 응용에 따라 인공지능은 80년대 초반부터 활기를 되찾았으나 지각능력과 상식추론 능력은 여전히 해결의 실마리를 찾지 못하고 있다. 이러한 인공지능의 한계 때문에 그 대안으로 신경망이론이 각광을 받고 있다. 신경망은 연결주의, 인공지능은 계산주의라고 불린다. 신경망은 수많은 뉴론이 연결되어 정보가 병렬적으로 처리된다는 측면에서 연결주의라고 불리는 반면에 인공지능은 기호처리 방식에 의하여 정보가 직렬적으로 처리된다는 의미에서 계산주의라고 불리 운다. 감각정보의 처리측면에서는 연결주의가 앞설 것으로 보고 있다. 계산주의가 뇌의 구조와 전혀 다른 직렬식 구조를 사용하였기 때문에 실패했을 따름이며 연결주의는 뇌의 구조를 본 뜬 병렬식 구조이기 때문에 성공할 가능성이 높다는 주장도 제기되고 있다. 인공지능과 신경망 이론사이에 끝없는 공방전이 계속되고 있다[1].

지능적인 기계의 또한 측면은 생체컴퓨터이다. 생물체는 강력한 정보처리 체계를 갖고 있다. 신경계, 내분비계 및 면역체는 생체분자에 의하여 정보를 수용, 전달 및 처리되는 정보처리 시스템이다. 이와 같이 생체의 정보처리를 모형으로 하여 개발되고 있는 컴퓨터가 생체컴퓨터이다. 그러나 아직까지 이렇다 할 만한 실체가 나타나지 못하고 있다.

7. 프로그램 개발방법의 변천과 수학의 중요성

이제 거의 모두가 컴퓨터를 사용하여 일을 처리하고 서로 자료를 주고받으며 일상생활

에 없어서는 안될 필수품이 되었다. 이에 따라 다양한 소프트웨어도 요구되어지고 있으며 앞으로도 요구되어지는 소프트웨어 개발에 효과적으로 대처해야만 했고 불충분한 소프트웨어 제공에 따라 소프트웨어 위기라는 용어도 나왔다. 이러한 소프트웨어 개발에 대처하기 위하여 언어개발 및 효과적인 소프트웨어 개발방법이 주목받고 있다. 1960년대에는 고급언어 개발에 심혈을 기울였고 70년대에는 구조적 프로그래밍 개발에 80년대 이후에는 객체지향 개념의 도입과 이를 통한 소프트웨어의 재활용 문제가 주목을 받고 있다. 현재는 자바언어에 의한 애플릿을 만들어 효과적으로 인터넷에서 활용하도록 하고 분산 객체로 작용하는 컴포넌트 등을 이용한 COBA나 DCOM 프로그래밍에 박차를 가하고 있다. 이제 소프트웨어도 자동차의 부품처럼 부품화를 하자는 것이다.

그러면 어떻게 소프트웨어를 개발하는 것이 가장 효과적인가?

여러 연구자들은 소프트웨어의 위기를 타개하기 위해 많은 노력을 기울였는데 그들의 해결책은 소프트웨어도 하드웨어가 공장에서 규격화되어 생산되는 것과 같이 규격화하여 생산성과 정확성을 높이려는 접근을 시작하였다. 이것이 본격적인 소프트웨어공학의 태동이라고 할 수 있다. 소프트웨어공학을 연구하는 학자들의 주요 목표는 성공적인 생산품, 즉 빠른 시간 내에 소프트웨어를 생산하는 기법을 개발하는데 있다.

컴퓨터 하드웨어와 소프트웨어에 있어서 최근의 관심사는 예전에는 불가능하다고 생각했거나 현실성이 없어 보이는 큰 규모의 시스템을 만드는 것이다. 이러한 시스템을 만드는 것은 너무 크고 복잡한 프로젝트나 대부분 기존 시스템의 확장과 수정을 요구한다. 소프트웨어 디자인에 있어서 객체지향 접근은 매우 성공적이다. 기존의 소프트웨어 공학기술을 "프로시저지향"이라 부른다. 전통적으로 소프트웨어 시스템은 데이터를 다루는 프로시저의 집단과 몇 개의 정보를 표현하는 데이터의 집합체로 보고 있다. 그리고 이들 데이터와 프로시저를 독립적인 것으로 취급해 왔다.

H.H.Lee와 J.S.Arora[8]가 지적한 바와 같이 이러한 전통적인 프로시저지향 시스템은 다음과 같이 제한적이었다.

1) 프로시저들간의 결합이 쉽지 않다. 따라서 소프트웨어 디자이너들은 상세한 부분까지 마스터 해야한다. 다른 사람이 프로시저들간의 결합은 이해한다는 것은 불가능하다.

2) 소프트웨어 요구사항이 조금만 바뀌어도 소프트웨어 전체를 수정해야 한다.

3) 시스템이 바뀌면 전체 소프트웨어를 수정해야 하기 때문에 시간과 비용이 많이 든다.

4) 모듈의 재사용이 안되어 다른 소프트웨어에 사용할 수 없다.

객체지향 소프트웨어 개발 방법론은 클래스 계층구조를 이용한 매우 강력한 클래스 재사용을 지원한다. 하위 클래스 생성은 소프트웨어 재사용 측면에서 볼 때 두 가지 중요한 의미를 갖는다. 첫 번째로 하위 클래스 생성은 기존의 클래스에 대한 수정을 유연하게 지원한다. 두 번째는 하위 클래스 생성이 소프트웨어 재사용에 미치는 영향으로 기존의 클래스에 새로운 기능 추가가 용이하다.

다음으로 객체지향 소프트웨어 설계 방법론이 소프트웨어 재사용에 도움을 주는 점은 틀을 이용한 모듈의 재사용이다. 소프트웨어 베이스가 방대해 지면 모듈 검색이 용이하지 못하다는 점을 지적하였는데 객체지향 소프트웨어 개발 방법론에서는 이 문제점을 틀을 이용한 해결책을 제시한다. 객체지향 데이터베이스의 데이터 모델과 일치하므로 클래스 혹은 클래스 계층구조를 효과적으로 저장, 검색할 수 있어 소프트웨어 베이스를 객체지향 데이터베이스 관리 시스템으로 효과적으로 관리할 수 있다.

객체지향 분석과 설계는 다음과 같은 여러 가지 중요한 장점을 제공한다.

1) 잘 설계된 디자인 방법은 객체지향 프로

그래밍 언어의 장점을 최대한 발휘할 수 있게 한다. 더구나 잘 정의된 클래스 라이브러리가 있지 않을 때도 그 장점이 발휘된다. 또한 비록 설계가 객체지향 언어로 구현되지 않을 지라도 객체지향 시스템이 장점을 나타내게 한다.

2) 객체지향 설계는 프로세스 집약적인 기능 분해 접근 방법보다는 훨씬 적은 양의 코드와 좀더 재사용 성이 뛰어난 코드를 생산한다.

3) 객체지향 설계 방법은 변화에 좀더 탄력적인 시스템을 생산한다.

4) 객체지향 방법론의 모듈화는 시스템 개발팀에게도 좋은 이익을 제공한다. 데이터와 프로세스가 하나의 객체에 제한되기 때문에 여러 개발팀이 서로 독립적으로 설계의 여러 부분에 참가하여 시스템을 개발하기도 용이하다.

5) 객체지향 방법론은 문제공간을 이해하는 방법이 자연스럽기 때문에 시스템 분석가나 설계자뿐만 아니라 최종 사용자에게도 좀더 직관적으로 이해 될 수 있다.

객체지향 설계의 결과는 클래스의 계층구조이다. 각 클래스는 제어와 데이터 구조가 함께 포함된 모듈이다. 문제 공간은 객체와 그것에 관련된 방법들의 모임으로 좀더 자연스럽고 사실적으로 관찰될 수 있다. 객체는 객체지향 설계의 초기요소이다. 그 후로 점차 객체들간의 공통점이 인식되면서 같은 공통점을 갖는 객체들로 클래스를 만들게 되고 또한 이들 클래스로부터 좀더 추상적인 추상클래스를 형성하게 된다. 이와 같은 추상화 작업의 가장 상위 단계는 프레임워크이다.

간략하게 말하면 객체지향 설계는 다음의 네 가지 기본적인 단계를 거친다.

1) 객체의 인식과 정의

2) 클래스의 구성

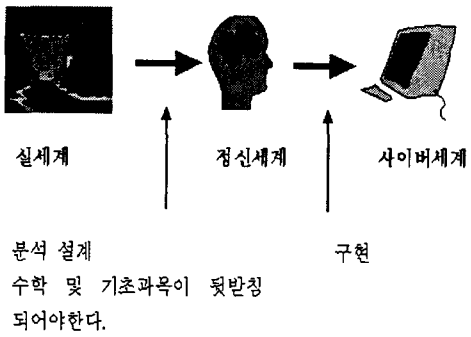
3) 클래스들간의 관계를 파악 및 클래스 계층구조 구성

4) 재사용 가능한 클래스 라이브러리와 응용 프레임워크를 제작한다.

객체지향 설계는 마치 기존의 시스템에 적용된 프로토타입 접근방법과 같이 순환적이다. 프로그래머는 클래스의 집합에서 설계를 시작해서 그것을 확장시키고 고쳐서 하나의 응용 프로토타입으로 꾸며 맞춘다. 주로 최종 사용자와의 교류를 통하여 프로토타입으로 꾸며 맞춘다. 주로 최종 사용자와의 교류를 통하여 프로토타입이 개선되길 원하면 설계자나 프로그래머는 다시 분석설계 작업을 반복한다. 이러한 재 작업을 통하여 클래스는 개선되어 재구성이 되며 만일 이렇게 개선된 클래스가 추후의 응용에 재 사용될 수 있을 만큼 일반적이라면 그것을 표준 클래스 라이브러리에 추가시킨다. 프레임워크 설계자는 응용 프로그램간의 유사성을 발견하여 미래의 비슷한 문제에 대한 프로그래밍 해결책을 유용한 프레임워크를 개발한다.

7절에서 지금까지 객체지향 방법론으로 소프트웨어 개발 방식이 바뀐 것을 알아보았다. 요즈음은 이에 관한 서적이 쏟아져 나오고 있다[6,4,10]. 올해 마이크로소프트에서는 비주얼 C^{++}6.0을 내놓으면서 분산 컴포넌트(COM/DCOM) 프로그래밍을 효과적으로 하도록 하였으며 래션널(Rational) 회사에서 개발했던 통합 모델링 언어인 UML을 기반으로 하는 객체지향 분석 및 설계 툴을 함께 제공하고 있다. 이러한 변화는 소프트웨어를 개발 할 때 객체지향 방법론에 의해서 이루어지고 이것은 프로그래밍 하는 것 보다 요구사항을 파악하여 분석 및 설계하는 단계가 더욱 중요하다는 것이다. 왜냐하면 이 단계에서 기계적으로 프로그래밍 단계로 전환되기 때문이다. 프로그래머들은 예술가가 아니며 서비스 업종에 종사하는 사람이다. 이제 이들은 공장의 직공으로 표현되고 있다. 소프트웨어를 개발할 때 분석 및 설계자는 신사복을 입고 있고 프로그래머는 작업복을 입고있는 그림이 이채롭다. 예전엔 소프트웨어 사장이 어떤 소프트웨어를 개발할 때 프로그래머에게 프로그래밍이 어떻게 되어 가는지

물으며 대우해 주었는데 이제는 객체지향 방법론을 다루는 분석설계자와 상의하며 이들을 대우해 준다고 한다[6]. 객체지향 방법론에서 가장 중요한 개념은 세상을 바라보는 방법(Paradigm)이다. 이 패러다임에 의해 실세계를 정신세계로 넣고 정신세계에서 컴퓨터로 구현되는 사이버(Cyber)세계로 전환된다. 실세계를 정신세계로 넣는 것은 현실에서 우리가 보거나 경험한 것을 추상화시키거나 일반화 시켜서 우리의 머리에 넣어 논리적으로 정립시키는 것이다. 이 과정에서 무엇보다도 수학과 기타 기초과목의 지식이 요구된다.



[그림 4]

참 고 문 헌

[1] 신동필역, 생각하는 컴퓨터, 홍릉과학출판사, 1982.

[2] 이인식, 사람과 컴퓨터, 까치, 1992.

[3] 하마노야스지(정보화 시리즈역), 컴퓨터 이후의 세계, 한국전자통신연구소, 1991.

[4] T. Budd, An introduction to object-oriented programming, Addison -Wesly,1991.

[5] C. L. Dym and R. E. Levitt, Toward the integration of knowledge for engineering modeling and computation, Engineering with computers, 1991 (209~224).

[6] H. E. Eroksson and M. Denker, UML Tookit, Jon Willy & Sons, 1998.

[7] J. P. Hayes, Computer architecture and organization, Mcgraw-Hill Inc, 1988.

[8] H. H. Lee and J. S. Arora, Object-oriented programing for engineering applications, Engineering with computers, 1991 (225~235).

[9] H. R. Lewis and C. H. Papadimitrion, Elements of the theory of computation, Prentice-Hall Inc, 1981.

[10] Rational Softmarw & Microsoft etc, UML proposal to the objective management group, www. rwtional. com/uml,1997

[11] R. Slater, Portraits in silicon, MIT press, 1987.

[12] T. A. Sudkamp, Language and machine, Addison-Wesley, 1991.

[13] W. W. Zourist and J. V. Leonard, Object-oriented simulation, IEZ press, 1996.