석사학위논문

컴퓨터과학 교육용
계산 원리 학습도구의 기능요소 고찰

제주대학교 교육대학원

컴퓨터교육전공

김 형 철

2011년 8월

컴퓨터과학 교육용
계산 원리 학습도구의 기능요소 고찰

지도교수 김 철 민

김 형 철

이 논문을 교육학 석사학위 논문으로 제출함

2011년 6월

김형철의 교육학 석사학위 논문을 인준함

심사위원장 김 한 일 ㊞

위 원 김 성 백 ㊞

위 원 김 철 민 ㊞

제주대학교 교육대학원

2011년 7월

<국문초록>

컴퓨터과학 교육용 계산 원리 학습도구의 기능요소 고찰

김 형 철

제주대학교 교육대학원 컴퓨터교육전공

지도교수 김 철 민

지식정보 사회에서 우리는 인간 생활과 자연 현상 속에서 다양한 형태의 수많은 문제들에 당면하며 살아가고 있다. 그러기에 주어진 문제들을 어떠한 관점에서 바라보고 접근하여 얼마나 효율적인 방법을 찾아내는 지가 개인과 국가의 경쟁력과 수준을 결정하는 주요 요소 중 하나임에 틀림없다. 이와 관련해 우리가 해결해야 할 문제들이 점점 복잡해지고 있을 뿐만 아니라 그 규모도 커지고 있다는 사실을 간과해서는 안 될 것이다. 왜냐하면 우리 두뇌는 복잡한 문제나 규모가 큰 문제를 해결하는데 취약하기 때문이다. 따라서 개인과 국가 모두 실세계의 크고 복잡한 문제를 효과적으로 다룰 수 있도록 관련 지식 및 능력 증진 방안을 탐색・시행하는데 힘써야 한다.

이와 관련해 본 논문은 컴퓨터과학이 가지고 있는 훌륭한 도구에 주목하길 요청하고 있다. 문제에 접근할 때 컴퓨터과학이 적용하는 사고의 패러다임의 핵심 축인 '추상개념'과 '자동화'가 그들이다. 추상개념과 자동화는 각각 문제의 복잡성과 규모를 효과적으로 공략할 수 있는 좋은 무기이다. 컴퓨터과학은 문제와 관련된 상황 혹은 현상을 '정보처리 과정' 혹은 '계산'이라는 관점에서 관찰해 추상

※ 본 논문은 2011년 8월 제주대학교 교육대학원 위원회에 제출된 교육학 석사학위 논문임.

모델을 설정하고, 해당 모델을 구성하는 요소(추상개념)들을 컴퓨터로 자동화함으로써 문제를 해결한다. 이때 추상개념은 모델링할 대상이 복잡할 경우 그것을 다수의 계층 혹은 영역으로 분할해 보다 작은 대상들 간의 상호작용으로 모델링할 수 있게 해준다. 이와 같은 작업과정은 복잡한 대상이 충분히 작고 다루기 쉬운 추상개념들 간의 상호작용으로 모델링될 때까지 반복된다. 최종적으로 설정된 추상개념들은 컴퓨터가 수행할 수 있는 알고리즘들로 정형화되어 그것이 생성하는 계산에 의해 문제가 해결되도록 자동화된다.

컴퓨터과학은 궁극적으로 자동화된 문제해결 시스템을 목표하기 때문에 추상개념 설정 과정 자체를 정확하고 치밀하게 유지하는데, 이때 추상개념과 자동화를 한 흐름으로 연결시켜 주는 것은 바로 '계산'이다. 그래서 표현(상태)의 나열로 정의되는 계산은 컴퓨터과학의 핵심 원리이다. 어떤 문제에 대해서든지 '계산'이라는 일관된 관점에서 관련 현상을 관찰·해석하고 정형화해 추상개념을 설정하고, 적정 컴퓨터를 선정해 이루어지는 자동화 역시 각 추상개념과 관련된 '계산' 생성 알고리즘을 구현하는 데에 초점을 맞춘다.

문제는 계산 원리가 컴퓨터과학의 본질적 요소로서 문제해결에 있어 엄청나게 중요한 역할을 하는데, 초중등 교육뿐만 아니라 대학 전공 교육에서조차도 계산 원리 학습이 제대로 이루어지고 있지 않다는 것이며, 계산 원리 학습을 효과적으로 지원해 주는 도구도 거의 없다는 것이다. 본 논문은 먼저 계산 원리 학습과 관련된 다양한 측면을 개괄적으로 정리·제시함으로써 계산 원리 학습의 중요성과 필요성을 강조하고 있다. 또한 계산 원리 학습도구가 갖추어야 할 기능 요소들을 분석적으로 고찰·도출해 향후 관련 도구 개발 시 검토하고 참고할 기준 지침으로 제시하고 있다.

목 차

표 목 차

그 림 목 차

Ⅰ. 서론

1. 연구 배경 및 목적

인간은 일상 속이나 각자의 전문 분야에서 다양한 수준의 수많은 문제들을 직면하며 살아간다. 그러기에 해결해야 할 문제들을 어떻게 발견해내고, 당면한 문제를 어떤 관점에서 탐색 · 접근해 얼마나 효율적인 방법으로 해결하는 지는 개인의 수준과 경쟁력을 결정하는 매우 중요한 요소가 된다.

실세계의 문제들 가운데 그리 어렵지 않게 해답을 찾을 수 있는 문제들도 많지만, 차선의 해결책을 찾는 것조차 어려운 복잡한 문제들도 적지 않다. 때때로 우리는 전자의 문제가 주어졌는데 그 해답을 찾지 못해 많은 비용을 치르기도 하고, 후자의 문제에 부딪혔는데 그것을 붙들고 시간과 자원을 허비하기도 한다. 유념해야 할 것은 누구라도 문제 해결과 관련된 기초 지식이나 역량이 부족하면 언제라도 이런 상황에 놓일 수 있다는 것이다. 이점에서 우리는 문제해결과 관련된 개인의 역량 제고에 지속적인 관심을 가져야 한다. 더불어 요소 간 소통의 시공간적 제약이 없어지고 요소 간 융합이 급격히 촉진됨에 따라 우리가 해결해야 할 문제들이 더 복잡해져 가고 그 규모도 더 커져갈 것이라는 사실도 간과해서는 안 된다. 우리 두뇌가 복잡한 문제나 규모가 큰 문제를 해결하는데 취약하기 때문에 다가올 상황들에 대해 어떤 대책이 필요함을 인식하고 그것을 찾아내야 한다는 것이다. 우리는 우리 자신이 반복 · 지속적인 작업에 쉬 지치고, 기억 용량이나 처리 속도가 그다지 크거나 빠르지 않을 뿐만 아니라, 실수 없이 작업하기가 쉽지 않음을 잘 알고 있다. 따라서 우리는 실세계의 문제들, 특히 대규모의 복잡한 문제들을 효과적으로 다루는데 필요한 개개인의 역량을 어떻게 향상시킬 것인지, 그 방법을 면밀히 탐색해야 한다.

본 논문은 이와 관련해 문제해결을 위한 컴퓨팅 패러다임을 탐색했고 컴퓨팅의 기본 원리로서 '계산 원리'가 지닌 교육적 가치에 주목했다. 컴퓨터과학은 자연적 혹은 인공적 '정보처리 과정(*information process*)'에 대한 학문으로, '무엇이

계산될 수 있는 것인가?', '그것이 어떻게 계산될 수 있는가?'라고 질문하며 모든 문제를 '계산(*computation*)'이라는 관점에서 접근·해결하는 학문이다.

컴퓨터과학은 문제와 관련된 현상 속에 어떤 표현들이 어떻게 상호작용하며 바뀌어 가는지 관찰한다. 그리고 그 속에서 표현들의 나열(*sequence of representations*), 즉 정보처리 과정이나 계산을 발견한다. 이 과정에서 복잡한 요소가 있으면 불필요한 부분은 제거하고 보다 작은 요소들로 분할해 서로 상호작용하는 별개의 '추상개념(*abstraction*)'으로 설정함으로써 추상개념들 상호 간에 서로의 세밀한 부분을 감출 수 있게 한다. 이는 원래 요소가 가지고 있던 복잡성을 보다 작은 추상개념 단위로 분할해 다룰 수 있게 하기 위함이다. 물론 그렇게 설정된 어떤 추상개념이 여전히 통괄적으로 다루기 복잡하다면 또 다시 해당 추상개념의 복잡성을 보다 작은 단위의 추상개념들로 분할할 수 있다. 이와 같은 복잡성 분할 작업은 최종적으로 설정된 추상개념들이 충분히 단순해질 때까지 반복된다. 이렇게 설정된 최종적인 추상개념들은 이제 '컴퓨터'에 의해 수행될 수 있도록 '자동화(*automation*)'된다. 즉 문제의 해답으로서 요구되는 계산을 생성해 내도록 알고리즘으로 정형화한다.

컴퓨터과학의 문제해결 핵심 도구인 추상개념과 자동화는 각각 문제의 복잡성과 규모를 효과적으로 공략할 수 있는 좋은 무기이다. 추상개념은 분할하고 감추는 방식으로 문제의 복잡성과 씨름할 수 있게 해 주고, 자동화는 컴퓨터를 이용해 계산 생성 속도를 높이는 방식으로 문제의 규모를 다룰 수 있게 해 주기 때문이다. 그렇다면 추상개념과 자동화를 연결시켜 하나의 흐름으로 정밀하게 엮어주는 것은 무엇인가? 그것은 바로 '계산'이다. 어떤 문제에 대해서든지 '계산'이라는 일관된 관점에서 관련 현상을 관찰·해석하고 정형화해 추상개념들 설정하고, 적정 컴퓨터를 선정해 이루어지는 자동화 역시 각 추상개념과 관련된 '계산' 생성 알고리즘을 구현하는 데에 초점을 맞추기에 그렇다. 계산은 컴퓨터과학의 가장 본질적인 요소이다. 계산은 모든 현상을 일관되게 관찰하고 포괄적으로 종합할 수 있게 해 주는 핵심 요소이다. 우리 일상의 전 영역에서 일어나는 급격한 변화와 발전의 저변에서 컴퓨터과학이 어떻게 작용하고 있는지 이해하는 열쇠이다. 앞으로 부딪히게 될 대규모의 복잡한 문제들을 효과적으로 다루기 위해 요구되는 추상개념과 자동화를 한 틀에 담을 수 있게 해 주는 바탕이다.

문제는 계산 원리가 컴퓨터과학의 본질적 요소로서 문제해결에 있어 엄청나게 중요한 역할을 하는데, 초중등 교육뿐만 아니라 대학 전공 교육에서조차도 계산 원리 학습이 제대로 이루어지고 있지 않다는 것이며, 계산 원리 학습을 효과적으로 지원해 주는 도구도 거의 없다는 것이다. 본 논문은 먼저 계산 원리 학습과 관련된 다양한 측면을 개괄적으로 정리·제시함으로써 계산 원리 학습의 중요성과 필요성을 논하고 계산 원리 학습의 내용 요소와 방향을 설정·제시하고 있다. 그리고 그와 같은 활동을 효과적으로 지원하기 위해 계산 원리 학습도구가 갖추어야 할 기능 요소들을 분석적으로 고찰·도출해, 향후 관련 도구 개발 시 검토하고 참고할 기준 지침으로 제안하고 있다.

2. 연구 내용 및 방법

본 연구의 필요성 및 목적에 따라 수행된 구체적인 연구 내용 및 방법은 다음과 같다.

첫째, 컴퓨터과학의 발전 역사와 컴퓨팅이 기반하고 있는 기본 원리들을 고찰함으로써 컴퓨터과학의 정의와 본질이 무엇인지 명확히 정리하였다.

둘째, 컴퓨터과학의 본질과 관련해 컴퓨터과학 교육의 방향과 방법을 고찰하였다.

셋째, 컴퓨팅 원리로서의 계산 원리와 계산이 기반하고 있는 계산모델들, 컴퓨터과학의 사고 패러다임인 계산적 사고, 수학/공학/과학과 차별화된 컴퓨팅 패러다임 등 계산과 관련된 이론과 선행연구들을 정리함으로써 '계산(*computation*)'의 의미를 보다 정확히 설정하였다.

넷째, 계산 원리 학습이 제대로 이루어지지 않을 경우 얻을 수 없는 유익들을 중심으로 계산 원리 학습의 중요성에 대해 논하였다.

다섯째, 계산모델 시뮬레이터, 프로그래밍 언어, 로봇 제어 프로그램 개발 도구 등 계산 관련 도구들의 특성을 분석하고 이들을 계산 원리 학습에 활용할 경우

어떤 문제점이 있는지 정리하였다.

여섯째, 다양한 영역의 계산 원리 학습 콘텐츠 사례를 구체적으로 정리·제시하였다.

일곱째, 컴퓨팅 패러다임을 틀로 사용해 콘텐츠의 어떤 요소들을 어떤 관점에서 학습해야 하는지 정리·제시하였다.

여덟째, 컴퓨팅 패러다임에 따른 계산 원리 학습과 연계시켜 계산 원리 학습도구의 기능적 요건을 4 가지로 분류·제시하였다.

마지막으로 계산 원리 학습도구의 기능요소 4가지를 비교 관점으로 설정해 기존의 관련 도구들을 분석·정리하였다.

Ⅱ. 이론적 배경

1. 컴퓨터과학 교육의 본질

1) 컴퓨터과학의 본질

(1) 컴퓨터과학의 정의

컴퓨터과학(computer science)은 태동된 이후 지금까지 지속적으로 또 급속히 성장해 왔으며, 독자적 학문으로 그 고유 영역을 확보해 가고 있다. 이러한 발전 과정에서 컴퓨터과학의 정의 역시 그에 부합되게 바뀌어 왔는데, 이는 컴퓨터과학의 본질을 보다 정확히 인식해 가는 과정에서 발생한 자연스런 귀결로 이해할 수 있다. 컴퓨터과학의 본질 상 잉태되어 있던 혁명들, 즉 1940년대에 시작된 '도구의 혁명', 1980년대에 시작된 '방법의 혁명', 2000년대에 시작된 '근본적 처리의 혁명' 등이 컴퓨터과학의 본질에 대한 보다 깊은 인식을 갖게 했던 것이다. 이와 관련해 Denning은 자신의 논문 "*Computing is a Natural Science*"에서 다음과 같이 기술하고 있다[1].

> 최초의 전자 디지털 컴퓨터 시대였던 1940년대에 계산은 방정식 해결, 코드 풀이, 데이터 분석, 사업과정 관리, 시뮬레이션 수행, 모델 해결 등의 도구로 간주되었다. 계산은 곧 다루기 힘든 분석을 다룰 수 있게 만드는 강력한 도구로 스스로를 확고히 했으며, 원자 에너지, 고급 항공/선박 설계, 의약품 설계, 건물 구조 분석, 일기 예보 등 많은 기술을 새로운 단계로 끌어 올렸다.
>
> 1980년대까지 계산은 많은 분야에서 완전히 필수적인 요소가 되었고, 기존 지식을 활용하는 도구로부터 새로운 지식을 발견해내는 수단으로 발전하였다. 노벨물리학상 수상자 Ken Wilson은 계산(*computation*)이 이론(*theory*)과 실험(*experiment*)이라는 과학의 전통과 함께 하는 과학의 세 번째 다리가 되었다고 말했다. 그는 다른 이들과 함께 계산을 주된 방법으로 사용해 새로운 발견을 탐색하는 것을 지칭하기 위해 '계산과학(*computational science*)'이라는 용어를 만들

어냈다. 이 아이디어는 1989년 미국 국회가 고성능 계산을 통한 기술적 발전을 고무하고자 '*High Performance Computing and Communication Initiative*'라는 법안을 통과시킬 만큼 강력한 것이었다.

2000년까지 계산은 더욱 발전했고, 많은 분야의 과학자들은 자기 분야의 심층 구조 속에서 정보처리를 발견했다고 말하고 있다. 노벨상 수상자이자 칼텍 회장인 David Baltimore는 "오늘날 생물학은 정보과학이다. 생명 조직으로서의 시스템의 출력은 디지털 매체로 인코딩되고 일련의 판독헤드에 의해 판독된다. 생물학은 더 이상 작은 실험실만의 독자 영역이 아니다. 많은 분야로부터의 기여가 있다."라고 언급했다. Baltimore는 "자연은 오래전에 조직에 대한 정보를 DNA 안에 어떻게 인코딩하는지와, 고유의 계산 방법을 통해 DNA로부터 새로운 조직을 어떻게 생성해내는 지를 배웠다."고 말했다. 오늘날 생물학자와 컴퓨터과학자는 자연의 정보처리를 이해하고 궁극적인 영향을 미치고자 할 때 긴밀하게 협력한다. 생물학만이 이와 같이 이야기할 수 있는 분야는 아니다. 물리학자들은 양자파가 물리적 효과를 생성할 수 있는 정보를 전송한다고 말한다. 그들은 양자계산과 양자암호 작성/해독 방법에 중대한 진전을 이루었다. 노벨상 수상자 Richard Feynman은 양자 전기역학(*QED*: *quantum electrodynamics*)이 양자 입자 상호작용을 조합하는 자연의 계산 방법임을 보임으로써 유명해졌다. 1980년대에 NASA-Ames의 계산과학자들은 Schroedinger Equation으로부터 그 분자구조를 계산해 목성 탐사(*Jupiter Probe*)를 위한 성공적인 메탄저항성 열 보호 물질을 발견했다. 자연은 수학으로 기록되어져 있다는 Galileo의 주장에 도전하며, Stephen Wolfram은 2002년 그의 저서 '*A New Kind of Science*'에서 자연은 계산 언어로 기록되어 있다고 선언했다. 경제학자들은 그들 고유의 정보 흐름에 맞추어 경제 시스템을 분석한다. 경영과학자들은 작업흐름(*workflows*), 책무(*commitments*), 사회망(*social networks*)을 모든 조직에 있어서의 근본적인 정보 처리라고 주장한다. 예술가들과 인문학자들은 분석으로부터 새로운 작품의 창조에 이르는 모든 것에 계산을 사용한다. 웹 연구자들은 웹 전체를 실험실로 사용해 새로운 사회적 행동과 컴퓨팅 방법을 발견하였다. iPod, eBay, Wikipedia, Google, Playstation, Xbox, Wii 등 수많은 인공물들은 스타일과 문화의 산물들이 되었다. 심지어 정치인들도 정교한 사회적 데이터 분석, 계산적 게리멘더링(자신에게 유리하게 선거구를 정하는 것), 블로깅 등을 활용하고 있다. Jeanette Wing은 계산 관련 개념들이 많은 분야에서 일상의 사고활동 속에 깊이 스며들

어 있다고 결론짓고 있다. 계산은 어디에나 있다.

컴퓨터과학은 1940년대에 자동 컴퓨팅(automatic computing)에 대한 학문에서 1950년대에 정보처리(information processing)에 대한 학문으로, 그리고 1960년대에 컴퓨터 관련 현상(phenomena surrounding computers)에 대한 학문으로 발전했고, 1970년대에 자동화 가능한 대상(what can be automated)에 대한 학문, 1980년대에 계산(computation)에 대한 학문이라는 인식 단계를 거쳐, 2000년대에는 자연적/인공적 정보처리 과정(information processes, both natural and artificial)에 대한 학문으로 발전되어 왔다.

(2) 컴퓨터과학의 기본 원리

컴퓨터과학은 '무엇이 계산될 수 있는 것인가?'와 '그것이 어떻게 계산될 수 있는가?'라는 질문에 답하는 '계산(*computation*)'에 대한 학문이며, 정보를 기술하고 변환하는 알고리즘적 과정(관련 이론, 분석, 설계, 효율성, 구현, 응용 등)에 대한 체계적 학문이다[2][3]. 그래서 본질적으로 컴퓨터과학은 실세계 혹은 상상 세계의 현상 속에 존재하는 정보처리 과정들에 관심을 가진다. 적정 컴퓨터를 선정하고 발견된 정보처리 과정들을 해당 컴퓨터가 수행할 수 있는 알고리즘으로 정형화함으로써 문제의 해답을 얻는다.

컴퓨터과학은 수학과 과학, 공학에 기초하고 있기에 그들 각 분야와 공유하는 특성을 가지고 있지만, 그들의 합집합만으로 설명할 수 없는 독특하고 강력한 학문적 차별성을 가지고 있다. 이와 같은 학문적 차별성을 강조하는 한편, 컴퓨터과학의 학문적 정체성을 체계적으로 정립하기 위해 Denning(2007)은 컴퓨팅 분야의 핵심이 되는 기본 원리들을 다음과 같은 7개의 범주로 구분하여 정리·제시하였다[4].

① Computation

데이터 표현(*representation*)과 절차적 표현, 정보처리 과정과 계산, 계산의 해결시간, 열린 계산과 닫힌 계산, 복잡도, 표현의 압축, 무한을 내포한 유한 표현, 유한 표현의 오류 등 컴퓨팅 분야의 핵심 원리인 계산과 관련된 원리들을 포함하고 있다. 여기에는 "계산적 처리를 통해 할 수 있는 것과

없는 것은 무엇인가?", "고유의 계산 복잡도나 만연한 계산 복잡도에 어떻게 대응할 것인가? 등에 대한 답이 포함되어 있다.

② Communication

메시지, 데이터 통신, 채널, 인코더와 디코더, 채널의 용량, 오류검출 및 정정 코드, 메시지 압축과 암호화/복호화 등 신뢰성이 있는 데이터 전송과 관련된 원리들을 포함하고 있다.

③ Coordination

상호작용 에이전트, 프로토콜(*protocol*), 유한게임과 무한게임, 작업흐름(*work flow*), 협력시스템, 흐름종속/공유종속/조화종속(*flow/sharing/fit dependency*), 병행성(*concurrency*), 중재(*arbitration*), 동기화(*synchronization*), 순차화(*serialization*), 경쟁상황, 확정성(*determinacy*) 등과 관련된 원리들을 포함하고 있으며, 자율적 개체들이 공동의 결과를 산출하기 위해 어떻게 작업할 수 있는지에 대해 답한다.

④ Recollection

저장소, 계산과 저장시스템, 휘발성 저장장치와 영구성 저장장치, 계층적 저장시스템, 캐시(*cache*)와 적중률/부재율, 실질적 접근시간(*effective access time*), 교체정책, 시간적/공간적 지역성(*temporal/spatial locality*), 작업집합(*working set*), 쓰래싱(*thrashing*), 저장객체의 이름/핸들/주소/위치 간 동적 바인딩 등과 관련된 원리를 포함하고 있어, 계산이 정보를 어떻게 저장하고 회수하는지와, 저장 시스템에서 데이터 배치가 그들의 성능에 어떤 영향을 미치는지에 대해 설명한다.

⑤ Automation

어려운 계산 작업(*hard computational tasks*), 휴리스틱 시스템, 물리적 자동화(*physical automaton*), 인공지능, 모델링(논리시스템, 퍼지시스템, 전문가 시스템, 신경망, 희소분산메모리), 탐색(상태공간, 휴리스틱 탐색, 유전자 알고리즘), 연역, 귀납, 집단지성(*collective intelligence*) 등과 관련된 원리들을 포함하고 있으며, 이는 인간의 작업을 수행하기 위한 효율적인 계산적 방법을 찾는 것과 관련된다. 작업의 성격은 조립라인을 작동시키고 차를 운전하고 항공기 표면을 제어하는 일과 같이 물리적일 수 있고, 계산, 체스

게임, 일정 짜기 등과 같은 정신적일 수도 있다.

⑥ Evaluation

평가 도구(모델링, 시뮬레이션, 실험, 데이터에 대한 통계 분석), 성능 계산과 성능 분석 등 다양한 계산 작업량 하에서 컴퓨팅 시스템이 어떻게 작용하는지와, 정시에 결과를 내놓아야 할 때 얼마만큼의 용량이 필요한지에 관한 원리들을 포함한다.

⑦ Design

설계의 요건(정확성, 속도, 결함 감내성, 적정성), 오류(*error*)와 결함(*fault*), 오류제한(내부검사와 외부검사)과 오류복구, 소프트웨어 설계 원리(계층적 집성, 수준, 가상기계, 객체), 모듈성(추상개념, 정보은익, 분할) 등 의존할 만하고(*dependable*), 신뢰할 만하며(*reliable*), 유용하고(*usable*), 안전하며(*safe*), 안정적인(*secure*) 소프트웨어와 컴퓨팅 시스템을 어떻게 설계하는지에 관한 원리들을 포함한다.

컴퓨터과학이 접목된 모든 영역에는 실질적으로 Denning이 제시한 컴퓨팅의 기본 원리들이 다양한 방식으로 스며들어 있다. 문제는 이 같은 원리가 사회 제반 영역에 적용되었을 때 나타나는 외적인 특성들로 인해 대부분의 사람들이 컴퓨터과학을 단지 기술로, 혹은 응용과학으로 인식하고 있다는 것이다. 컴퓨터과학의 본질에 대한 왜곡된 인식이 여러 가지 부정적 영향을 끼치지만, 그로 인해 빚어지는 가장 근본적인 문제는 컴퓨터과학의 학문적/교육적/사회적/경제적 가치를 올바로 바라보고 평가하지 못 한다는 것이다. 컴퓨터교육의 중요성과 방향성을 발견하고 탐색하는데 있어 이점을 간과하지 말아야 할 것이다.

2) 컴퓨터과학 교육의 방향

지식정보사회의 구성원들을 대상으로 한 컴퓨터과학 교육의 방향 설정은 매우 중요하다. 이는 지식정보사회를 이해 · 선도하는 데 필요한 핵심 요소들이 컴퓨터과학의 본질 속에 들어 있기 때문이다. 그러므로 컴퓨터과학 교육의 방향을 설정할 때 고려해야 할 사항은, 사회구성원들이 컴퓨터과학의 본질적 요소들을 제대로 알고 그 요소들을 개인의 일상이나 전문 분야에 적용할 수 있도록 그와 관련

된 기초 역량을 키워주어야 한다는 것이다. 이를 위해 컴퓨터과학 교육이 지향해야 할 첫째 목표는 컴퓨터과학의 본질을 오해하게 만드는 요소들에 대해 올바른 인식을 갖게 하는 것이고, 둘째 목표는 컴퓨터과학의 본질적 요소인 계산 원리를 다양한 측면에서 생각하고 접근해 볼 수 있도록 기회를 제공하는 것이다. 첫째 목표와 관련해 올바로 인식시켜야 할 핵심 대상은 '컴퓨터'이며, 둘째 목표와 관련해 고려해야 할 사항은 컴퓨터과학이 가진 제반 도구들을 활용하되 계산 원리의 여러 측면을 생각하고 다루어 볼 수 있게 도와주어야 한다는 것이다.

첫째 목표와 관련해, 컴퓨터과학의 본질을 인식하는데 있어 현실적인 큰 장애물 중 하나가 '컴퓨터'라는 사실은 아이러니다. 그것은 '컴퓨터'에 대한 잘못된 인식이 컴퓨터과학에 대한 잘못된 이미지를 갖게 하기 때문이다. 컴퓨터와 관련해 알아야 할 사항 중 하나는 컴퓨터가 컴퓨터과학의 본질적 연구 대상이 아니라 자연물이나 인공물과 관련된 정보처리 과정을 발견·분석하고 그것을 구현·실험·입증하는 과정에서 도움을 주는 유용한 도구라는 것이다. 컴퓨터 자체가 연구의 대상이 되기도 하지만, 그것은 컴퓨터를 컴퓨터과학의 보다 효과적인 도구로 만들기 위함이다. 컴퓨터과학을 보다 좋은 성능의 컴퓨터를 연구하고, 관련 기술을 적용해 새로운 시스템을 개발하는 영역으로 인식할 경우, 컴퓨터과학은 그저 공학이요 기술이요 응용과학일 뿐이다. 컴퓨터과학을 컴퓨터를 잘 활용할 수 있도록 몇 가지 운영체제나 소프트웨어의 사용법을 익히기만 하면 굳이 더 배우지 않아도 되는 그런 학문으로 인식한다면, 컴퓨터를 활용하는 수준과 방식에 있어서의 근본적 한계를 벗어날 수 없게 된다. 어떤 성격의 일을 하든지 해당 영역에서 컴퓨터를 제2의 두뇌처럼 자유롭게 활용하려면, 컴퓨터에 대한 많은 지식을 익혀야 하는 것이 아니라 자연적 · 인공적 환경 속에서 일어나는 현상을 '정보처리 과정'이라는 관점에서 관찰해 '계산'으로 정형화하는데 필요한 지식과 사고 능력을 갖추어야 한다. 컴퓨터과학의 본질과 기본 원리를 알지 못한 채 그와 같은 지식과 능력을 제대로 갖추어 가는 것이 쉽지 않은 일임을 분명히 인식해야 한다.

첫째, 목표와 관련해 또 한 가지 알아야 할 사항은, '컴퓨터'는 넓게 볼 때 계산의 주체가 되는 개체, 즉 고유의 계산 능력을 가진 '컴퓨팅 에이전트(computing agent)'를 의미한다는 것이다. 처리와 저장, 통신 능력을 갖춘 물리적

장치로서의 컴퓨터가 컴퓨팅 에이전트인 것은 틀림없다. 하지만 인공물이나 자연물, 혹은 인간까지도 컴퓨터로 포괄될 수 있음을 알아야 한다. 이는 인간 역시 정보처리, 즉 계산을 행하기 때문이다. 인간과 기계의 처리 능력을 조합·활용할 필요가 있다면, 인간과 기계의 조합을 컴퓨터로 생각할 수도 있다. 이미지의 분석과 해석 측면에서는 인간이 기계보다 훨씬 뛰어나지만, 특정 부류의 명령어를 인간보다 훨씬 빨리 수행한다거나 인간이 다룰 수 있는 것보다 훨씬 큰 규모의 데이터를 처리할 수 있다는 점에서는 기계가 인간보다 더욱 뛰어나기에, 인간과 기계의 조합은 아주 멋진 조화를 이룬다. 기계나 인간 각각이, 혹은 기계와 인간의 조합이 컴퓨터가 될 수 있을 뿐만 아니라, 재귀적으로 그와 같은 컴퓨터들의 조합(예: 네트워크) 역시 컴퓨터가 될 수 있다는 인식의 유연성은 컴퓨터과학적 사고의 폭과 깊이를 더하는데 있어 매우 중요한 요소이다.

둘째, 목표와 관련해 한 가지 생각해 볼 사항은 컴퓨터 활용교육이다. 이는 최근까지 초중등 컴퓨터교육의 주된 형태이기도 했다. 컴퓨터 활용교육은 응용 소프트웨어의 특정 버전을 사용해 현장에서 요구되는 작업을 보다 효과적으로 수행하는 방법을 가르치는 형태로 이루어져 왔다. 문제는 이와 같은 성격의 교육이 해당 소프트웨어를 사용하는데 당장의 도움은 되겠지만, '계산' 관점에서 해당 소프트웨어의 구조적/기능적 요소들이 그 소프트웨어의 동작과 어떻게 연계되는지에 대해 알고 있지 못할 경우 업그레이드된 버전의 소프트웨어나 유사 기능의 다른 소프트웨어를 활용하려면 해당 소프트웨어의 사용법을 거의 새로 배워야 하는 악순환을 피하기 어렵다는 것이다. 둘째 목표는 어떤 대상(하드웨어, 소프트웨어, 컴퓨터과학 이론 등)에 대해서 가르치든지 계산의 관점에서 그 대상을 이해하고 다룰 수 있게 해야 한다는 것이다. 이영준은 [5]에서 컴퓨터과학 교육에 있어 정보의 중요성을 강조하며, 정보의 개념 정의를 위한 핵심 요소로 표현(representations), 처리(processes), 장치(machines), 관계(relationships), 구성(construction)을 들었다. 이영준은 이들 다섯 가지 핵심 요소를 들어 컴퓨터과학이 실제 생활의 복잡한 객체(object)와 행위들(behaviors)을 표현하고 통제 및 관찰 가능한 행위들의 실제를 자동화하거나 시뮬레이션 할 수 있는 강력한 분야임을 주장하였다. 정보처리 과정 혹은 계산의 대상으로서 정보(표현)를 다룰 때 이들 다섯 가지 핵심 요소를 고려해 보는 것도 의미 있는 일이다.

컴퓨터과학 교육의 방향을 단지 응용 프로그램 활용법을 숙달시키는 것으로, 혹은 컴퓨터과학 분야에 축적된 여러 가지 개념과 정교한 이론을 이해시키는 것으로 설정해서는 안 된다. 그렇다고 모든 이에게 프로그래밍을 가르쳐 뛰어난 프로그래머가 되게 하는 것으로 설정해서도 안 된다. 컴퓨터과학 분야 전반에 작용해 놀라운 발전을 이끌어 내고 있는 본질적 요소, 컴퓨터과학과 다른 분야가 부딪히는 경계에서 작용해 엄청난 변화를 일으키고 있는 근본 요소에 주목하도록 컴퓨터과학을 교육해야 한다. 도대체 무엇이 어떻게 작용해서 그런 변화를 이끌어내고 있는지 살펴볼 수 있고, 어떤 분야라도 우리가 원하는 영역에 그 요소를 적절히 작용시켜 그곳에서도 놀라운 변화를 만들어낼 수 있는 능력을 키워 주어야 한다. 그래서 컴퓨터과학의 주요 원리와 개념은 물론, 타 학문 혹은 교과 영역의 내용, 일상의 상황 등 다양한 사례들을 끌어와 계산 관점에서 다루어 주어야 한다.

2. 계산 : 컴퓨터과학의 원리

계산 원리는 컴퓨터과학의 원리 중 컴퓨터과학의 본질과 직접적으로 연관된 핵심 원리이다. Denning(2007)이 제시한 7가지 범주의 컴퓨팅의 원리 중 계산 원리 부문을 간략히 정리하면 다음과 같다[4].

- **표현**(*representations*)**은 정보**(*information*)**를 가지고 있다.**
  - 표현은 정보를 전하는 심볼들의 패턴이며 모든 표현은 '매개체'라 불리는 물리 현상 속에서 구현된다.
  - 관찰자들은 패턴을 읽어냄으로써 의미를 파악하고 영향을 끼친다.
  - 표현의 의미(해석)는 관찰자와 표현 간의 상호작용 수단에 종속된다.
  - 표현은 유한하며, 두 표현이 동일 정보를 가지면 서로 동일하다.
- **계산**(*computation*)**은 표현들의 순차**(*a sequence of representations*)**이다.**
  - 계산은 알고리즘에 의해 야기된 데이터 표현 상태들의 순차이다. 아날

로그 기계에서, 계산은 연산자 네트워크에 의해 생산된 연속 함수이다. 계산은 처리라 불리기도 한다.

- 상태 그 차체가 값들의 표현이므로, 계산은 표현들의 순차(연속적인 순차 포함)라고 단순하게 말해도 부족함이 없다. 연이은 표현들은 연산자에 구현된 논리적 규칙에 의해 제어된다.
- 미래에 새로운 계산 형태는 양자 컴퓨팅, 생물정보학, 시맨틱 네트워크, 경제학 등에서 나올 것으로 보인다. 공통 요소는 모든 형태가 표현의 순차라는 것이다.

• **표현은 그리 많게는 아닐지라도 압축될 수 있다.**

- 긴 표현 **L**과 짧은 표현 **S**가 동일한 정보를 가진다면, **L**로부터 **S**를 구성하는 알고리즘을 압축(*compression*)이라 부른다. 되돌릴 수 있는 압축인 경우 **S**로부터 **L**을 복구해내는 알고리즘을 압축해제(*decompression*)라 한다. 압축해제를 통해 **S**로부터 **L**을 완전히 복구해낼 수 있을 때, 해당 압축을 무손실(*lossless*) 압축이라 한다.
- 무손실 압축의 한 가지 유익은 **L** 내의 인코딩된 심볼들은 보다 짧은 코드로 대체할 수 있다는 것이다. 예를 들어 **L** 내의 8-비트 아스키(ASCII) 코드들 대신 평균 길이 6 비트인 가변 길이 코드를 사용해 보다 짧은 표현 **S**를 구성할 수 있다. 원래 8-비트 코드를 단순 복원함으로써 **S**는 **L**로 압축해제 될 수 있다.

• **계산은 열려 있거나 닫혀 있다.**

- 시작 상태에서 출발해 유한 시간 내에 종료 상태에 이르는 것을 그 목적으로 할 경우, 해당 계산이 닫혀 있다고 한다.
- 닫힌 계산(*closed computation*)은 시작 상태를 종료 상태로 대응시키는 수학적 함수와 결부된다.
- 무한히 계속되는 것을 그 목적으로 할 경우, 해당 계산이 열려 있다(*open computation*)고 한다. 어떤 상태에서 계산은 주변 환경과 정보를 교환한다. 외부에서 주어진 요구(태스크라 불림)는 계산의 상태를 변경하며, 태스크에 대한 응답은 주변 환경으로의 출력이다.

• **계산은 문제해결에 대한 특유의 속도를 가진다.**

- 닫힌 계산의 해결 시간은 시작 상태에서 출발한 후부터 종료 상태에 이르기까지 소요된 시간이다.
- 열린 계산(*open computation*)의 해결 시간은 태스크(task)가 주어진 후부터 응답이 이루어지기까지 소요된 시간을 의미한다.

• **복잡도(*complexity*)는 계산을 완료하는데 필요한 시간이나 공간을 측정한다.**

- 모든 컴퓨팅에 걸쳐 거론되는 주제로서 복잡도는 알고리즘의 계산을 끝마치기 위해 요구되는 시간 혹은 공간을 의미한다. 잠재적으로 알고리즘은 그 입력에 따라 발생 가능한 계산들의 무한한 공간을 가지고 있기 때문에, 복잡도 평가는 어려운 일이다. 일반적으로 보다 큰 입력 데이터 집합은 더 긴 수행시간과 더 많은 메모리 공간을 요구한다.
- 휴리스틱(*heuristics*)은 **NP** 문제에 대한 근사해를 찾기 위해 경험에 기반해 주먹구구식 방법을 적용하는 다항시간 알고리즘이다. 최적치에 근접한 답을 찾아준다고 보장하지는 못하지만, 관례상 휴리스틱은 종종 충분히 좋은 답을 제시한다.
- 계산 **A**를 인코딩해 계산 **B**의 사례로 만든 후 해결 가능하면 **A**는 **B**로 환원될 수 있다(*reducible*)고 말한다. 이 경우 인코딩 과정은 다항시간보다 나쁘지 않아야 한다.

• **실제 처리과정의 유한 표현(*finite representations*)은 항상 오류를 포함한다.**

- 연속적인 실제 변수나 처리과정에 대한 표현은 유한한 개수의 표현된 점들로 무한히 많은 실제 점들을 다룰 수 없기 때문에 결코 정확할 수 없다. 표현 오류는 실수와 그것을 표현한 수 사이의 차이이다. 신중히 계획함으로써 출력의 표현 오류를 제한하도록 알고리즘을 조직화할 수 있다.
- 제대로 조직화되지 않은 긴 계산을 수행하게 되면, 표현 오류가 누적되어 최종 결과로서 엄청난 오류가 내재된 부동소수점 수가 산출될 수 있다.

3. 계산모델(computation model)

계산은 그것이 기반하고 있는 계산모델(computation model)의 관점에서 정의되는 처리 과정이다. 계산모델은 컴퓨팅 시스템에 대한 수학적 추상체로, 계산에서 사용될 수 있는 연산들의 집합과 각 연산의 비용을 정의한다. 따라서 어떤 계산에 대해 정확히 이해하려면 그것이 기반하고 있는 계산 모델에 대해 올바로 알아야 한다. 이것이 계산모델과 연계한 계산 원리 학습이 필요한 이유다.

특정 운영체제나 프로그래밍 언어를 알면 해당 시스템 상에서 발생하는 여러 가지 현상이나 해당 언어로 구현된 프로그램의 수행 행태를 보다 잘 이해 할 수 있듯이, 특정 계산모델을 알면 그 모델에 기반한 계산에 대해 보다 잘 이해할 수 있다. 또한 다양한 운영체제나 프로그래밍 언어를 익히고 있을 경우 특정 작업의 수행 혹은 구현에 적합한 시스템과 언어를 보다 잘 선택할 수 있게 되듯이, 다양한 유형의 계산모델을 알고 있을 경우 적정 계산모델을 선택하거나 변형함으로써 원하는 계산을 생성해 주는 알고리즘을 보다 효과적으로 찾을 수 있게 된다.

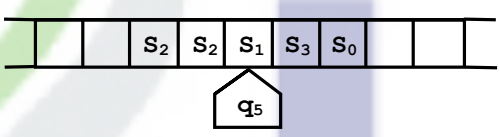
많은 계산모델들이 있다. 튜링기계(Turing machine), 재귀함수(recursive function), 람다계산(lambda calculus), 유한상태 기계(finite state machine), 유한상태 오토마타(finite state automata) 등 대표적인 계산모델들도 다양하다. 본 절에서는 튜링기계, 유한상태기계, 유한상태 오토마타만 정리 · 소개한다.

1) 튜링기계

튜링기계(Turing machine)는 컴퓨터과학 분야에서 가장 중요한 순차 계산 모델이다. 튜링기계는 1936년 튜링(Turing)이 최초로 제안했던 계산모델로, 계산장치(알고리즘적 시스템)의 능력(알고리즘적 문제해결 능력)과 한계를 연구하기 위한 모델로 사용되고 있다. 이는 “직관적으로 계산 가능한 어떤 것도 튜링기계로 계산할 수 있다.”는 처치-튜링 가설(Church-Turing Thesis)에 근거한 것으로, 튜링기계로 어떤 문제의 해답을 찾을 수 없다고 증명하면 그 문제가 풀 수 없는 문제임을 입증할 수 있다.

튜링기계는 연필과 지우개를 가지고 기다란 종이테이프(칸 단위로 나뉘어 있으

며, 각 칸에는 유한한 알파벳 중 하나의 기호가 쓰여 있음) 위에 기본적인 연산(현재 칸에 기록되어 있는 기호를 읽고 지우고, 새로운 기호를 기록한 후 왼쪽 혹은 오른쪽 칸을 현재 칸으로 설정하는 작업)을 반복하는 사람을 모델링한 것으로, 튜링 자신은 a(utomatic)-machine이라 불렀다.



<그림 1> 튜링기계의 개념도

<그림 1>은 튜링기계 표현의 한 방법을 보여 준다. 튜링기계의 구성요소들을 이와 관련시켜 설명하면 다음과 같다.

- 테이프(tape)

  튜링기계의 테이프는 띠 모양을 가진 양방향 무한 길이의 판독/기록 가능 매체로, 길이 방향에 따라 배치된 셀(cell)들로 구성된다. 셀은 하나의 기호(symbol)를 저장(기록)해 둘 수 있는 테이프 상의 저장 단위로 헤드를 통해 읽거나 쓸 수 있는데, 이들 기호(테이프 기호라 함) 모두의 집합을 튜링기계의 알파벳(machine's alphabet : 튜링기계가 인식하는 기호들의 유한집합)이라 한다. 튜링기계의 알파벳은 특수 기호인 '*blank*'를 포함하고 있는데, 어떤 기호가 기록되기 전 혹은 기록된 기호가 지워진 후의 셀은 *blank* 기호를 가진 것으로 간주된다. <그림 1>에서 가로 방향의 띠가 테이프이고, 세로 선으로 구분된 칸들 각각이 셀이다. 셀에 표시된 s_0, s_1, s_2, s_3 등이 해당 셀에 기록된 기호들이고 아무 표시가 없는 셀에는 *blank* 기호가 표시된 것이다.

- 헤드(head)

  테이프 상에 기호를 읽고 쓰는 장치로, 한 번에 최대 한 셀만큼 왼쪽 혹은 오른쪽으로 헤드를 이동시킬 수 있는 장치이다(헤드는 멈추어 있고 테이프가 움직이는 것으로 설정한 튜링기계 모델이 사용되기도 함). 현재 헤드가 위치해 있어 읽고 쓸 수 있는 셀을 '현재 셀(*current cell*)'이라 하

고, 그곳에 기록되어 있는 기호를 '현재 기호(*current symbol*)'라 한다. 그림에서 테이프 아래에 그려진 5각형이 헤드이다. 헤드의 뾰족한 부분이 가리키고 있는 헤드 바로 위의 셀(기호 s_1이 기록되어 있는 셀)이 현재 셀이다. 현 상태를 기준으로 헤드를 오른쪽으로 한 셀 이동시키면 s_3이 표시된 셀이 현재 셀이 되고, 왼쪽으로 한 셀 이동시키면 s_2가 표시된 셀이 현재 셀이 된다.

- 테이블(table)

  튜링기계가 수행할 계산에 대한 유한 개의 명령(instruction) 혹은 규칙(rule)들이 나열된 목록으로, 튜링테이블(*Turing table*), 상태테이블(*state table*), 액션테이블(*action table*), 전이함수(*transition function*) 등으로 불린다. 아래 표가 튜링테이블의 구체적 사례 하나를 보여 준다.

현재상태	입력기호	출력기호	헤드이동	다음상태
S	0	1	L	T
S	1	0	L	T
S	*	* 혹은 None	L	T
T	0	1	R	H
T	1	0	R	H
T	*	* 혹은 None	R	H

각 행마다 5가지 요소로 구성된 하나씩의 명령이 명시되므로, 이 테이블에는 총 6 개의 명령이 포함되어 있다. 명령의 나열 순서는 우리가 이해하기 좋게 하기 위한 것일 뿐, 튜링 기계의 작동에 영향을 주지 않는다. 첫 번째 명령과 여섯 번째 명령을 통해 이 테이블에 규정된 규칙에 따라 작동하는 튜링기계에 대해 우리가 알 수 있는 것은 다음과 같다.

- 첫 번째 명령의 의미와 역할 : 튜링기계의 '현재 상태(*current state*)'가 **S**이고 현재 셀로부터 입력된 기호, 즉 현재 기호가 '**0**'이면, 현재 셀에 '**1**'을 출력 · 기록하고 헤드를 왼쪽(**L**)으로 한 칸 이동시킨 후 현재 상태를 **T**로 전환시킨다.
- 여섯 번째 명령의 의미와 역할 : 현재 상태가 **T**이고 현재 기호가 '*****'이면, 현재 셀에 '*****'를 출력 · 기록하고(기록 작업을 생략해도 이

미 기호 '*'이 저장되어 있으므로 결과는 동일함) 헤드를 오른쪽(R)으로 한 칸 이동시킨 후 현재 상태를 H로 전환시킨다.

여기서 현재 상태란 현재 기호를 읽은 뒤 튜링 테이블의 어떤 명령을 수행할지 결정할 때 사용되는 튜링기계의 내부 상태로 '상태 레지스터(*state register*)'에 저장되며, 튜링 상태(*Turing state*) 혹은 *m-configuration*(machine's configuration)이라고도 한다. 특정 시점에 튜링 기계의 전체 상태(complete configuration)는 해당 시점의 튜링 상태와 현재 셀, 그리고 테이프에 기록된 모든 기호들을 총괄해 규정할 수 있다. 첫 번째 명령과 여섯 번째 명령을 각각 아래와 같이 표기해 보면 각 명령의 의미가 보다 명확해진다.

```
S 0 → T 1 L
T * → H * R
```

이 표기법을 사용할 경우 다음 명령은

$$q_i\, a_j \rightarrow q_{i1}\, a_{j1}\, d_k$$

튜링기계의 현재 상태가 q_i이고 테이프의 현재 셀에서 읽어 들인 기호가 a_j일 때, 튜링기계가 다음과 같은 작업을 순서대로 수행하도록 규정한다.

① 현재 셀에 기호 a_{j1}을 기록(출력)한다. a_{j1}이 *blank* 기호인 경우 지우는 작업이 되고, a_j와 같은 기호일 경우 출력 작업을 하지 않는 것과 같은 결과를 갖게 된다(테이블 표현 상 지우는 작업은 **Erase**, 출력을 생략하는 경우에는 **None**이라 표기하기도 함).

② d_k 값에 따라 헤드를 이동시킨다. d_k는 3가지 값을 가질 수 있는데, **L**이면 왼쪽으로, **R**이면 오른쪽으로 한 셀 이동시키라는 의미이고, **N**이면 헤드를 이동시키지 말라는 뜻이다.

③ 현재 상태를 q_i에서 q_{i1}로 전환한다.

위의 작업이 모두 끝나면 튜링기계의 현재 상태와 현재 셀이 새롭게

설정되고 한 주기의 작업이 마무리된다. 튜링 기계는 작동이 시작된 후부터 정지할 때까지 위와 같은 작업 주기를 반복한다. 즉 현재 셀에 기록된 기호를 읽어 현재 상태와 읽은 기호가 일치하는 테이블의 명령을 찾은 후 위의 ①, ②, ③의 작업을 수행한다. 이후 현재 상태와 현재 셀이 새롭게 설정된 상태에서 다음 주기를 시작한다. 튜링 테이블은 튜링기계가 수행할 계산을 생성해내는 알고리즘(혹은 프로그램)으로 간주할 수 있다.

- 상태 레지스터(*state register*)

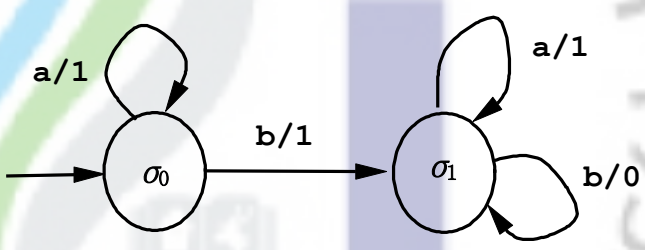
  튜링기계의 현재 상태가 저장되는 곳이 상태 레지스터이다. 상태 레지스터에 저장될 수 있는 상태의 수는 유한하며, 특별한 상태로서 '시작 상태(*start state*)'와 '정지 상태(*halt state*)'가 있다. 튜링기계의 작동이 시작될 때 상태 레지스터는 시작 상태로 초기화되고 상태 레지스터 값이 정지 상태가 되면 튜링기계의 작동이 멈추게 된다. 앞에서 예시했던 튜링 테이블과 관련해 튜링기계의 시작 상태는 **S**이고 정지 상태는 **H**이다.

위와 같은 특성에 따라 작동하는 모든 기계는 튜링기계이다. 따라서 무수히 많은 서로 다른 튜링기계들이 존재할 수 있다. 그렇다면 튜링기계들을 서로 구별시켜 주는 핵심 요소는 무엇인가? 그것은 바로 튜링테이블이다. 이는 다른 요소들 모두가 튜링기계 구성 및 작동 방식의 공통적 측면인 반면, 알고리즘으로서 각 튜링기계의 계산 작업을 차별적으로 규정지어 주는 것이 튜링테이블이기 때문이다.

2) 유한상태기계

유한상태기계(finite-state machine)는 유한의 메모리를 가진 기계에 대한 추상적 모델이다. 특정 시점에 유한상태기계의 상태는 그 당시 메모리에 저장되어 있는 값들의 총체로 정의될 수 있으며, 메모리 용량이 유한해 서로 다른 상태의 개수가 유한하기 때문에 유한상태기계라 한다. 동작 중인 유한상태기계는 언제나 메모리에 저장된 값들의 총체로서 특정 상태를 갖게 되고, 외부에서 어떤 기호가 입력되면 특정 기호를 출력하게 된다. 이 과정에서 유한상태기계의 상태가 바뀔

수 있다. 유한상태기계가 출력하는 기호나 전이하게 되는 상태는 입력 기호가 주어질 때의 기계 상태와 주어진 입력 기호에 의해 결정된다. <그림 2>는 어떤 유한상태기계 M_1의 입출력 행태와 상태 전이과정을 보여준다.



<그림 2> 유한상태기계 M_1의 상태전이도

M_1은 두 가지 상태(원으로 표시됨) σ_0과 σ_1을 가지 수 있는데, 기계의 초기 상태(레이블이 붙여지지 않은 화살표가 들어오게 표시된 상태)는 σ_0이다. M_1의 상태가 σ_0일 때 기호 **a**가 입력되면 기호 **1**을 출력한 후 상태 σ_0에 머물고(상태 σ_0에서 출발해 다시 상태 σ_0으로 들어가는, 레이블 '**a/1**'이 붙여진 화살표가 이를 나타냄), M_1의 상태가 σ_0일 때 기호 **b**가 입력되면 기호 **1**을 출력한 후 그 상태가 σ_1로 바뀐다(상태 σ_0에서 출발해 상태 σ_1로 들어가는, 레이블 '**b/1**'이 붙여진 화살표가 이를 나타냄). M_1의 상태가 σ_1일 경우 입력 기호가 **a**인지 **b**인지에 따라 출력되는 기호는 각각 **1**과 **0**으로 달라지지만 어떤 기호가 입력되든 M_1의 상태는 σ_1에 머물게 된다.

M을 유한상태기계라 할 때 $M = (I, O, S, f, g, \sigma)$와 같이 여섯 가지 요소로 정의될 수 있는데, 그 구성 요소들은 다음과 같다.

- 입력기호의 유한집합 I
- 출력기호의 유한집합 O
- 상태의 유한집합 S
- $S \times I$에서 S로의 상태전이함수 f

- $S \times I$에서 O로의 출력함수 g
- 초기상태 $\sigma \in S$

<그림 2>에서 예시한 유한상태기계 M_1을 $M_1 = (I, O, S, f, g, \sigma_0)$와 같이 나타낼 때, I, O, S, f, g는 각각 다음과 같이 정의된다.

- $I = \{a, b\}$
- $O = \{0, 1\}$
- $S = \{\sigma_0, \sigma_1\}$
- $f : S \times I \rightarrow S$, $g : S \times I \rightarrow O$

	f		g	
S \ I	a	b	a	b
σ_0	σ_0	σ_1	0	1
σ_1	σ_1	σ_1	1	0

$f(\sigma_0, a) = \sigma_0$ $g(\sigma_0, a) = 0$
$f(\sigma_0, b) = \sigma_1$ $g(\sigma_0, b) = 1$
$f(\sigma_1, a) = \sigma_1$ $g(\sigma_1, a) = 1$
$f(\sigma_1, b) = \sigma_0$ $g(\sigma_1, b) = 0$

3) 유한상태 오토마타

유한상태 오토마타(finite state automata)는 출력 기호가 0과 1 중 하나이고, 현재 상태가 직전 출력을 결정짓는 유한상태기계를 말한다. 즉 어느 한 상태로 전이할 때 출력되는 기호 모두가 0으로 통일되어 있거나 1로 통일되어 있다는 의미이다. 이점에서 유한상태 오토마타의 상태는 두 가지로 분류할 수 있는데, 하나는 그 상태로 진입할 때 0이 출력되는 상태들이고, 다른 하나는 진입시 1이 출력되는 상태들이다. 이들 중 1을 출력하고 진입되는 상태들을 수용 상태(accepting state)라 한다.

유한상태 오토마타는 기호들의 나열로서 주어진 문자열이 어떤 문법에 맞게 표현된 언어인지 아닌지 판별하는데 사용된다. 해당 문법에 맞게 유한상태 오토마타를 정의하고, 주어진 문자열을 구성하는 기호들이 순서대로 하나씩 유한상태 오토마타의 입력으로 주어진다고 가정해 해당 문자열의 마지막 기호까지 입력시키면 유한상태 오토마타는 수용 상태에서 멈추거나 비 수용 상태에서 멈추게 된다. 수용 상태로 끝나게 만든 문자열은 해당 문법에 맞는 언어로, 비 수용 상태

로 끝나게 만든 문자열은 해당 문법에 따른 언어가 아닌 것으로 판단할 수 있다. 유한상태 오토마타 M은 다섯 가지 요소를 명시해 $M = (I, S, f, A, O)$와 같이 정의할 수 있는데, 유한상태기계의 구성요소 중 O는 항상 {0, 1}이기 때문에 명시될 필요가 없어 제외 된 것이고, 출력 함수 g는 수용 상태의 집합 A로 대체 · 명시된 것이다. 집합 A에 속한 상태들로 전이되는 경우의 출력은 모두 1이고 집합 A에 속하지 않은 상태들로 전이되는 경우의 출력은 모두 0이므로, 실질적으로 출력 함수 g 대신 집합 A를 명시해도 모든 각각의 상태 전이 시 출력되는 기호가 정의되기 때문이다.

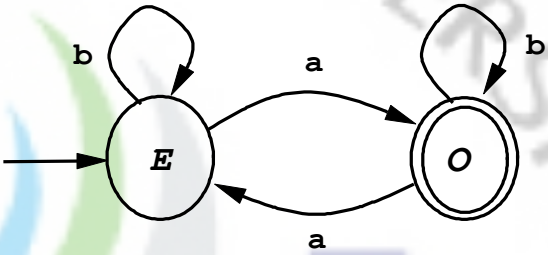
<그림 3>은 a와 b로 구성된 문자열 중 홀수 개의 a를 포함하는 문자열만 수용하는 유한상태 오토마타 M_2의 상태전이도이다. 이는 <그림 2>의 상태전이도 표현 방법과 두 가지 점에서 차이가 있는데, 하나는 두 겹의 원이 존재한다는 것이고, 또 다른 하나는 출력기호가 표시되어 있지 않다는 것이다. 일반적으로 유한상태 오토마타의 상태전이도에는 수용 상태를 비 수용 상태와 구별해 인식할 수 있도록 전자는 두 겹의 원으로 후자는 한 겹의 원으로 표시한다. 유한상태 오토마타의 상태전이도에 출력 기호가 표시되지 않은 이유는 수용 상태(두 겹의 원으로 표시된 상태)로 진입하는 화살표에 표시될 출력 기호는 무조건 1이고, 나머지 화살표에 표시될 출력 기호는 무조건 0이기 때문이다. 유한상태 오토마타 M_2를 $M_2 = (I, S, f, A, E)$와 같이 나타낼 때, I, S, f, A는 각각 다음과 같이 정의된다.

- $I = \{a, b\}$
- $S = \{E\}$
- $f : S \times I \rightarrow S$

	f	
S \ I	a	b
E	O	E
O	E	O

$f(E, a) = O$
$f(E, b) = E$
$f(O, a) = E$
$f(O, b) = O$

- $A = \{O\}$



<그림 3> 유한상태 오토마타 M_2의
상태전이도

4. 계산적 사고

1) 계산적 사고의 정의

계산적 사고는 컴퓨팅의 파워와 제약을 바탕으로 문제를 해결하고, 시스템을 구축하고, 인간행동을 이해하는 접근방법으로, 분석적 사고의 한 종류이다. 계산적 사고는 문제해결 시 취하는 일반적 방법들을 수학적 사고와 공유한다. 또한, 실세계의 제약 하에서 작동하는 대형의 복잡한 시스템을 설계하고 평가할 때 취하는 일반적 방법들을 공학적 사고와 공유한다. 계산적 사고는 계산 가능성, 지능, 정신, 인간 행동을 이해할 때 취하는 일반적 방법들을 과학적 사고와 공유한다.

계산적 사고는, 좁은 의미로는 계산 시스템(*computational system*)을 활용해 효과적으로 작업하기 위해 습득해야 할 사고방식이나 태도이며, 넓은 의미로는 세상을 이해하는 양식(단순한 방법을 초월한 양식, 광범위한 인간노력에 두루 접목 가능한 양식)이다. 이 개념에 함축된 것들과, 그 실질적/이론적 응용을 탐구하는 것이 컴퓨터과학의 핵심과제이다. 지난 수십 년 간 물리적인 계산 시스템은 바뀌어 왔지만, 사고의 기본 습성(계산적 사고)은 바뀌지 않았음에 유의할 필요가 있다.

문제해결을 위한 기본 패러다임으로서, 계산적 사고는 '추상개념(*abstraction*)'과 '자동화(*automation*)'를 주된 도구로 활용한다. 추상개념은 계산적 사고의 정수이다. 컴퓨팅에서 관념의 추상화는 시공간의 물리적 차원을 넘어선다. 컴퓨터과학

에서의 추상개념은 그것이 상징적이기에 극히 일반적이며, 수치적 추상개념은 특별한 경우일 뿐이다. 컴퓨터과학의 추상개념은 두 가지 면에서 수학이나 물리학에서 말하는 추상개념보다 풍부하고 복잡한 경향이 있다. 첫째는, 실수나 집합과 같이 물리적 세계에 대한 수학적 추상개념이 명료하고 쉽게 정의 가능한 대수적 속성을 향유하지만 컴퓨터과학의 추상개념은 반드시 그렇지 않다는 것이다. 예를 들어 컴퓨팅에서 사용되는 일반적인 추상데이터타입(*abstract data type*)으로 스택(*stack*)이 있다. 두 스택에 대한 '더하기(*to add*)'를 정의한다면 분명 두 정수에 대한 더하기보다 복잡할 것이다. 또 다른 예로 알고리즘은 입력을 취해서 원하는 어떤 출력을 생성하는 단계적 절차에 대한 추상개념이다. 혹 효율적 병행처리를 위해 생각할 수도 있는, 두 알고리즘에 대한 '더하기(*to interleave*)' 역시 훨씬 더 복잡한 상황을 전제로 정의되어야 한다. 프로그래밍 언어는 일련의 문자열들에 대한 추상개념으로, 각 문자열은 해석될 때 어떤 계산을 유발하게 된다. 두 개의 프로그래밍 언어를 '더하기(*to combine*)'하는 것을 어떤 의미의 작업으로 정의할 수 있는가? 스택에 대한 더하기, 알고리즘들을 대상으로 한 더하기, 프로그래밍 언어에 대한 더하기 등 추상개념에 대한 '조합자(*combinator*)'들은 그 자체가 정의하는데 신중히 생각해야 할 추상개념이며, 온전한 연구 의제이기도 하다. 둘째는 컴퓨팅에서의 추상개념은 궁극적으로 물리적 세계의 제약 안에서 동작하도록 구현되어야 하기 때문에, 경계 사례나 오류 사례 등에 대해 고민해야 한다는 것이다. 디스크가 가득차거나 서버가 응답하지 않으면 어떤 일이 발생하게 되는가? 컴파일시간에 처리되었어야 할 오류가 프로그램 수행 중에 발생하게 되면 어떻게 되는가? 로봇이 사람에게 부딪히지 않고 복도를 따라 이동하게 하려면 어떻게 해야 하는가? 이와 같은 성격의 고민들은 분명 수학이나 물리학이 그들의 추상개념을 다룰 때 하는 고민이 아니다.

계산적 사고의 관점에서 볼 때, 컴퓨팅은 '추상개념에 대한 자동화'로 이해될 수 있다. 추상개념이 컴퓨팅의 '정신적' 도구라면, 자동화는 '금속적' 도구이며, 추상개념의 힘은 자동화의 힘에 의해 증폭된다. 사실 추상개념과 자동화의 조합은 컴퓨팅의 엄청난 능력과 영향력의 기저를 이룬다. 추상개념과 자동화는 계산적 사고를 통해 어려운 문제, 즉 복잡하거나 규모가 큰 문제에 효과적으로 접근할 수 있게 해 주는 놀라운 '무기'이다. 왜냐하면 추상개념을 활용해 문제의

복잡도를, 자동화를 통해 문제의 규모를 적절히 공략할 수 있기 때문이다.

비근한 예로, 인터넷을 하나의 추상개념으로 상정해 볼 수 있다. 물론 그 규모가 너무 커서 속속들이 알지는 못하지만 분명 금속적 도구(정보통신기기)에 의해 자동화된 하나의 커다란 시스템이다. 그 속에 수많은 정보와 서비스가 녹아들어 있고 엄청난 사이버 공간이 구축되어 있으며, 시공간적 제약을 초월해 실세계의 컴퓨팅 기기들과 사람들을 소통시킨다. 인터넷은 분명 다양한 기능의 수많은 '컴퓨터'들에 의해 자동화되어 있고 생명을 가진 것처럼 활성화되어 있지만, 대부분의 사람들은 그 속에서 구체적으로 어떤 요소들이 어떻게 연계되어 어떻게 상호작용하고 있는지 알지 못한다. 이는 인터넷을 하나의 커다란 추상개념으로 간주할 수 있다는 점에서 당연한 것이다.

추상개념은 그 크기나 구조, 역할 등에 있어 정말 다양하다. 한 추상개념 속에서 그보다 작은 추상개념을 찾을 수 있고, 또 그 안에서 더 작은 추상개념을 발견해낼 수 있다. 어떤 시스템을 올바로 이해하고 제대로 활용하려면 해당 시스템을 구성하는 다양한 수준의 추상개념들과, 그들 간의 상호작용을 알아야 한다. 인터넷과 관련해 어느 수준의 추상개념까지 제대로 이해하고 있는지는 인터넷 활용의 생산성과 직결된다. 인터넷에 대해 왜곡된 지식을 가지고 있을 경우 인터넷을 잘못 사용함으로써 엄청난 피해를 입거나 끼칠 수 있지만, 적정 수준까지 인터넷과 관련된 추상개념들을 알고 있을 경우 엄청난 유익을 얻거나 끼칠 수 있기 때문이다.

인터넷과 같이 이미 존재하는 대상을 분석하고 이해하는데 있어서도 그렇지만, 어떤 문제를 해결해야 하는 상황에서도 추상개념은 매우 의미 있는 역할을 한다. 우선 그 문제를 해결하기 위해 필요한 대상들을 크기가 큰 몇 개의 추상개념으로 설정하고, 그들 간의 상호작용 양식을 정의한다. 그런 후 다시 각 추상개념이 수행해야 할 역할을 적절히 분할해 각각의 역할을 담당할 보다 작은 추상개념을 설정한다. 추상개념의 '크기'를 줄여가는 작업은 자동화시키기에 적합한 크기로 추상개념이 작아질 때까지 반복된다. 추상화는 불필요한 부분을 배제케 해 줌으로써 복잡한 것을 단순화시켜 준다. 또한 자동화 작업이 시작될 때까지 복잡한 구체적 작업을 지연시켜 준다. 추상개념 간에 정의된 상호작용 양식만 엄격히 지킨다면 각 추상개념의 자동화 작업을 서로 다른 사람에게 맡길 수도 있다. 실제

로 웹브라우저를 자동화한 사람과 웹서버를 자동화한 사람이 서로 다르다. 웹브라우저를 자동화한 사람에게 웹서버는 추상적인 대상일 뿐 웹서버가 어떻게 자동화되었는지 모르더라도 문제될 게 없다. 결국 추상개념은 해결해야 할 문제의 복잡성을 효과적으로 해소시켜 나갈 수 있게 해준다. 반면, 자동화는 각 추상개념이 수행해야 할 역할을 금속적 도구(컴퓨팅기기)가 수행할 수 있게 코드화해 관련 작업의 처리시간을 극적으로 줄여 줌으로써, 이전에 상상할 수 없었던 대규모의 작업도 적정 시간 내에 처리할 수 있게 해 준다.

2) 계산적 사고의 특성

계산적 사고는 컴퓨터과학에 기초하고 있다. 따라서 컴퓨터과학이 기초하고 있는 수학, 공학, 과학과의 관련성을 내포하고 있으며, 이들 분야의 사고 패러다임과 나름의 관계를 가지고 있다. 계산적 사고는 수학적 · 공학적 · 과학적 사고 각각과 공유하는 특성을 가진 반면, 그들의 독특하고 강력한 조합으로서 그들 각각이나 그들의 합집합만으로 설명하기 어려운 차별화된 특성을 가진다. 계산적 사고의 특성을 수학적 · 공학적 · 과학적 사고와의 관련성을 전제로 정리하면 다음과 같다.

- 수학적 사고와의 관계
  - 컴퓨터과학은 본질적으로 수학적 사고에서 비롯되었고 수학 위에 그 형식적 기초를 구축하고 있다. 그런 이유로 계산적 사고는 문제해결 시 취하는 일반적 방법들을 수학적 사고와 공유한다. 실제로, Boole, Shannon, Turing, Von Neumann과 같은 수학자들의 노력이 컴퓨터과학 발전에 큰 도움이 되었다.
  - 수학적 사고는 수학 체계를 수학적 공리(axioms)와 정리(theorems)에 의해 구축된 건물로 간주하며, 모든 정리는 엄격히 증명(일단의 공리들로 시작해 논리적 빈틈없이 유도)되어야 한다고 요구한다. 따라서 한번 증명된 수학적 정리는 절대적으로 옳다고 보장된다. 이는 물리, 천문학 등 과학 분야에서 실험에 의해 입증된 법칙이라도 실험 오류로 인해 완전히 옳다고 보장받지 못할 뿐만 아니라 차후의 보다 정확한 실험에 의해 폐기될 수도 있다는 점과 다르다.

- 컴퓨터과학은 자동화 과정에서 계산도구(컴퓨팅기기)의 제약을 받게 된다. 그래서 컴퓨터과학자들은 수학적으로가 아니라 계산적으로 생각하게 된다. 컴퓨터 발전의 각 단계가 수학 체계 속에서 정리를 증명하는 것과 동일한 방식으로 전개될 수밖에 없었다면, 아마도 컴퓨터과학은 침체되었을 것이다. 계산 도구가 기계식 컴퓨터에서 전자기식 컴퓨터로, 또 현대의 전기식 컴퓨터로 급격히 발전할 수 있었던 이유는 수학적 사고를 넘어선 계산적 사고가 있어서이다.

• 공학적 사고와의 관계

- 컴퓨터과학은 실세계와 상호작용하는 시스템을 구축한다는 점에서 공학과 본질적·태생적 연계성을 가진다. 그런 이유로 계산적 사고는 실세계의 제약 하에서 작동하는 대형의 복잡한 시스템을 설계하고 평가할 때 취하는 일반적 방법들을 공학적 사고와 공유한다.
- 가상세계 구축의 자유로움은 컴퓨터과학자들이 물리 세계를 넘어선 어떤 시스템도 설계·제작할 수 있게 해 준다. 이것이 공학적 사고를 넘어선 계산적 사고의 한 단면이다.

• 과학적 사고와의 관계

- 계산적 사고는 계산 가능성(computability), 지능, 정신, 인간 행동을 이해할 때 취하는 일반적 방법들을 과학적 사고와 공유한다. 실험 알고리즘, 실험 컴퓨터과학, 계산과학 분야의 컴퓨터과학자들은 정보처리에 대해 연구할 때 가설을 세운 뒤 실험을 통해 그들을 검증하며, 그렇게 검증된 가설은, 다른 분야의 과학적 가설들이 그렇듯이 세상의 현상을 설명 · 예측하는 모델로 사용된다.
- 계산도구나 계산적 사고의 발전은 기존의 과학적 사고에 중대한 영향을 끼치고 있다. 예로, 계산 생물학은 생물학자들이 생각하는 방식을 바꾸고 있고, 나노컴퓨팅과 양자컴퓨팅은 각각 화학자들과 물리학자들의 사고방식을 바꾸고 있다.

Wing이 제시한 계산적 사고의 특성은 다음 6가지이다[6].

• **계산적 사고는 프로그래밍**(*programming*)**이 아니라 개념화**(*conceptualizing*)

이다. 컴퓨터과학은 컴퓨터 프로그래밍이 아니다. 컴퓨터과학자처럼 생각하는 것은 컴퓨터 프로그램을 작성할 수 있는 것 이상을 의미한다. 계산적 사고는 다양한 추상화 수준에서 생각하도록 요구한다.

- **계산적 사고는 암기된 소양**(*rote skill*)**이 아니라 기본 기량**(*fundamental skill*)**이다.** 여기서 기본 기량은 모든 사람이 현대사회에서 제대로 활동하기 위해 반드시 습득해야 할 기량을 뜻하고, 암기된 소양은 기계적인 동작을 의미한다. 아이러니하게도 컴퓨터과학자들이 인공지능 분야의 원대한 도전과제인 "컴퓨터를 인간처럼 생각하게 만들기"를 달성하게 되면 인간의 사고는 '기계적인 동작'이 될 것이다.
- **계산적 사고는 컴퓨터가 아니라, 인간이 생각하는 방법이다.** 계산적 사고는 인간이 문제를 해결하는 방법이며, 인간을 컴퓨터처럼 생각하게 만들려고 시도하지 않는다. 컴퓨터는 아둔하고 따분하지만 인간은 영리하고 창의적이다. 컴퓨터를 흥미롭게 만드는 것은 다름 아닌 우리 인간이다. 컴퓨팅 기기를 '장착'해 자신의 기량을 확장한다면, 인간은 컴퓨팅 시대 이전에 엄두도 내지 못했던 문제를 해결할 수 있게 되고, 상상 가능한 기능의 시스템이라면 무엇이라도 구축할 수 있게 된다.
- **계산적 사고는 수학적 사고와 공학적 사고를 보완·조합한다.** 모든 과학이 수학 위에 그 형식적 기초를 구축하고 있다고 가정하면, 컴퓨터과학은 태생적으로 수학적 사고에 의지하고 있는 것이다. 컴퓨터과학이 실세계와 상호작용하는 시스템을 구축한다면, 공학적 사고와 연계될 수밖에 없다. 기반이 되는 컴퓨팅 기기의 제약조건들로 인해 컴퓨터과학자들은 수학적 사고가 아니라 계산적 사고를 하게 된다. 가상세계 구축의 자유로움은 물리 세계를 초월한 어떤 시스템도 설계 가능하게 만든다.
- **계산적 사고는 인공물**(artifacts)**이 아니라 아이디어**(ideas)**다.** 계산적 사고는 어느 곳에나 물리적으로 존재하고 언제나 우리 삶과 맞닿아 있는 소프트웨어나 하드웨어가 아니다. 계산적 사고는 문제를 해결할 때나 우리 일상을 관리할 때, 혹은 다른 사람들과 소통하고 상호작용할 때 사용하는 '계산'에 기반한 개념들이다.

• **계산적 사고는 모든 사람, 모든 영역을 위한 것이다**. 그것이 명시적 철학(explicit philosophy)으로 보이지 않게 될 만큼, 계산적 사고는 인간 노력의 필수 불가결한 요소가 될 것이다.

컴퓨터과학자들은 계산적 사고를 통해 제반 문제를 접근·해결한다. 개인의 일상뿐만 아니라 경제, 산업, 국방, 교육, 학문 등 지식정보 사회의 제반 영역이 계산적 사고에 의해 변화되고 있고, 또 변화되어 갈 것이다. 계산적 사고가 컴퓨터과학자들만의 전유물이 아님은 물론이다. 이미 많은 학자들이 자신의 학문 영역에 계산적 사고를 적용해 새로운 성취를 이루고 있고, 기초적인 수준이긴 하지만 일반인들도 누구나 일상생활에 계산적 사고를 적용하고 있다.

5. 컴퓨팅 패러다임

컴퓨팅은 '어떤 방법으로 컴퓨터가 이 문제를 해결하게 만들 것인가?'라는 질문에 답하는 것과 관련된다. 이 질문에 답하는 데 내포된 것은 적합한 추상개념을 설정하고 해당 작업을 담당할 적정 컴퓨터를 선택하는 일이다. 컴퓨터과학자들은 계산적 사고를 통해, 즉 현상 속에서 발견되는 계산들 중에 어떤 부분을 추상화시켜 자동화함으로써 상황을 개선할 수 있는지 찾아내는 방식으로 문제를 발견한다. 문제가 정의되면 관련 개체 내부나 개체 간에서 발생하는 계산을 어떻게 효율적으로 자동화할 수 있는지 고민하고 적정 컴퓨터를 설정하여 해당 계산을 실행하도록 구현함으로써 주어진 문제를 해결한다. 컴퓨터과학자들이 문제 해결 과정에서 적용하는 단계적 사고의 틀을 컴퓨팅 패러다임이라 할 때, 컴퓨팅 패러다임은 계산적 사고를 효과적으로 적용·전개하기 위한 틀로 간주할 수 있다. Denning과 Freeman은 컴퓨팅 패러다임이 수학과 과학, 공학이 지닌 패러다임들(<표 1>)과의 공명을 넘어 독특한 차별성을 가지고 있음을 역설하며, 컴퓨팅 패러다임을 다음과 같이 정리 및 제시했다[7].

• **초기화**(*initiation*) : 관찰 혹은 구축 대상 시스템이 유한(종료 가능)한 정보

처리 과정으로 표현될 수 있는지, 혹은 무한(지속 대화식)의 정보처리 과정으로 표현될 수 있는지 결정한다.

- **개념화**(*conceptualization*) : 대상 시스템의 행동을 생성하는 계산 모델(예: 알고리즘, 컴퓨팅 에이전트 등)을 설계 혹은 발견하다.
- **실현**(*realization*) : 설계된 처리과정을 명령 수행 능력이 있는 매체 내에 구현한다. 발견된 처리과정에 대한 시뮬레이션 혹은 모델을 설계한다. 정보처리과정의 행동을 관찰한다.
- **평가**(*evaluation*) : 해당 구현에 대해 논리적 정확성, 가설과의 일관성, 성능 제약, 원 목표 충족 여부 등을 검증한다. 필요할 경우 구현을 개선하다.
- **실행**(*action*) : 선출된 결과를 실세계에 반영한다. 지속적 평가를 위해 모니터링 한다.

<표 1> 컴퓨팅에 내재된 서브 패러다임

	수 학	과 학	공 학
초기화	연구 대상을 특징 지음(정의)	현상의 재현이나 패턴을 관찰(가설)	목표시스템의 행동과 응답에 대한 문장을 기술(요구조건)
개념화	객체 간의 관계에 대한 가설을 제시(정리)	관찰을 설명하고 예측을 가능케 하는 모델을 구성(모델)	시스템 기능과 상호 작용에 대한 형식적 문장 기술하기(사양)
실현	어떤 관계가 참인지 연역(증명)	실험을 수행하고 데이터를 수집(검증)	프로토타입을 설계 및 구현(설계)
평가	결과를 해석	결과를 해석	프로토타입을 시험
실행	결과에 따라 조치를 취함(적용)	결과에 따라 조치를 취함(예측)	결과에 따라 조치를 취함(구축)

컴퓨팅 패러다임의 특징은 관심의 대상이 '계산 기계'로부터 자연 현상까지 포괄한 '정보처리 과정'으로 옮겨 갔다는 것이다. 경험적 알고리즘, 분산 데이터, 융합 데이터, 디지털 법의학, 분산 네트워크, 소셜 네트워크, 자율로봇시스템 등 수학적 분석이 어려운 다양한 영역에 과학적 접근방법이 접목되고 있어, 발견

(discovery)이 구축이나 설계만큼 중시되고 있다. 나아가 컴퓨팅 패러다임 안에서 발견과 설계가 긴밀하게 연결되고 있는데, 웹과 같이 설계된 대규모 시스템의 행동이 관찰에 의해 발견되고 있고, 발견된 정보처리과정을 모방하는 시뮬레이션이 설계되고 있으며, 다양한 분야에서 과학적 발견을 돕는 탐색 도구들이 개발되고 있다. 컴퓨팅은 또한 많은 분야의 자연적 정보처리과정을 인식하고 있으며, 컴퓨팅 개념을 활용해 자연적 처리과정에 대한 새로운 발견과 이해를 가능케 하고 있다.

6. 계산 원리 학습의 중요성

계산 원리는 컴퓨터과학의 핵심 원리이다. 계산 원리는 인공물 혹은 자연물들 속에서 발생하고 있는 모든 현상들을 '정보처리 과정'이라는 통일된 관점에서 접근할 수 있게 해 주기 때문에 그렇다. 정보처리 과정이라는 관점에서 해당 현상을 해석·모델링하고 적정컴퓨터를 선정해 시뮬레이션 할 수 있는 것도, 현상 속에 존재하는 요소들과 컴퓨팅 시스템이 특정의 방식으로 상호작용할 수 있도록 구상·설계·자동화할 수 있는 것도 컴퓨터가 발생시키는 현상과 주변 세계에서 일어나는 현상 모두를 '정보처리 과정' 혹은 '계산'이라는 공통의 틀로 해석해 엮어 놓을 수 있기 때문에 가능한 것이다.

계산적 사고나 컴퓨팅 패러다임의 기저에 계산 원리가 자리 잡고 있다. 그래서 계산 원리를 모르거나 올바로 이해하지 못하면 계산적 사고 능력을 키우거나 컴퓨팅 패러다임을 적용하는데 있어 근본적 제약을 받게 된다. 앞서 설명한 바와 같이 컴퓨팅 패러다임에 근간한 계산적 사고 능력은 어떤 분야에서 일을 하든지 누구에게나 요구될 기초 역량이다. 그 역량이 계산 원리에 기반하고 있다는 점이 계산 원리 학습의 중요성을 설명해 준다. 계산 원리에 대해 학습하지 못할 경우 얻지 못하게 되는 기회와 유익을 개괄적으로 정리하면 다음과 같다.

- 다양한 영역에서 발생하는 현상을 '계산'의 관점에서 접근함으로써 얻을 수 있는 발견과 성취의 기회를 제공받지 못한다.

- 계산 개념을 매개로 다양한 분야의 지식이나 기술, 업적 등을 통합 혹은 융합할 수 있는 기회를 얻지 못한다.
- 단순화, 새로운 발견, 혁신 등을 가능케 한 기술들 간의 공통 원리를 밝힐 수 없다.
- 분야 간 공동 관심사를 효과적으로 논의할 수 있는 공통의 언어를 제공받을 수 없다.
- 계산을 가르치고 배우기 위한 새로운 접근방법에 대해 영감을 줄 수 없다.
- 컴퓨팅 관련 지식과 기술을 제반 영역에 효과적으로 적용하려 할 때 요구되는 유연성과 일관성에 근본적인 제약을 받게 된다.
- 컴퓨터과학에 대해 직간접적으로 관심을 가진 젊은이들에게 영감을 줄 수 없다.

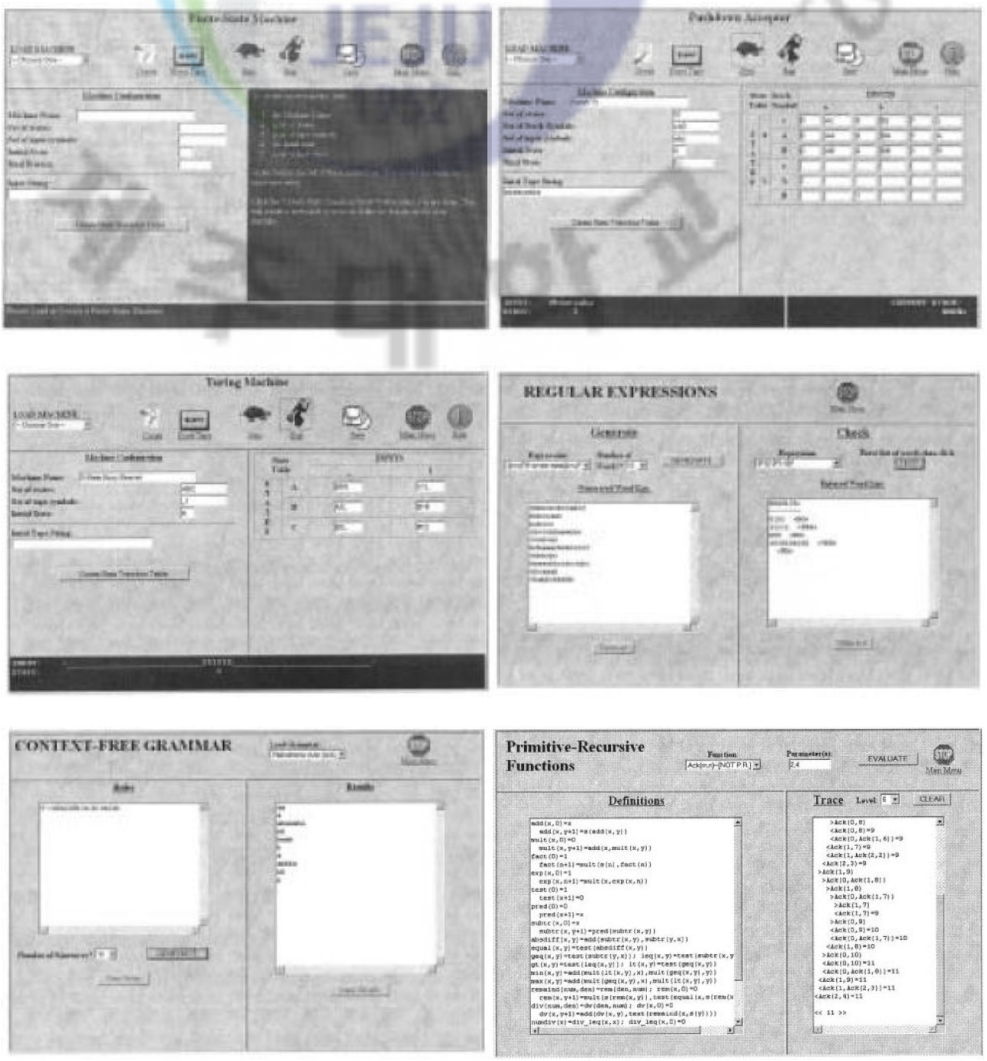
이와 같은 기회와 유익을 얼마나 풍성히 얻고 향유하는 지가 계산 원리 학습의 충실성에 달려 있음을 고려해야 한다. 이와 관련해 안타까운 문제 중 하나는 컴퓨터과학과 관련된 전공을 이수하는 학생들조차 계산 원리에 대한 학습 기회를 제대로 부여받지 못하고 있는 것이다. 대부분의 전공자들이 계산에 대한 개념을 제대로 정립하지 못한 채 컴퓨터 사용법과 프로그래밍을 배우며 컴퓨터과학의 전문 지식을 쌓고 있는 것이 우리의 현실이다. 이점에서 계산 원리 학습도구의 필요성이 보다 절실하다 하겠다.

7. 계산 원리 학습 관련 도구

1) 계산모델 시뮬레이터

계산모델 시뮬레이터(computation model simulator)는 특정 계산모델의 특성과 동작을 시뮬레이션 해주는 학습 도구이다. 학습자는 이 도구를 이용해 해당 계산모델에 기반한 계산 작업을 구체적으로 수행시켜 보면서 계산모델의 특성 및 동작 원리를 학습할 수 있다. 한 사례로서 [8]에서 제안하고 있는 <그림 4>의

계산이론 학습 도구는 각각 한 가지의 계산모델을 학습할 수 있게 지원하는 6개의 모듈이 통합된 웹기반 상호작용 도구이다. 이 도구는 유한상태기계(*finite-state machine*), 푸시다운기계(*pushdown machine*), 튜링기계(*Turing machine*), 정규수식(*regular expressions*), 문맥자유 문법(*context-free grammars*), 재귀함수(*recursive functions*) 등에 대한 학습 요소를 선택적으로 활용할 수 있게 지원하는데, <그림 4>에서 그 인터페이스를 볼 수 있다[8]. 학습자는 도구가 미리 정의해 제공하는 계산모델 사례(특정 계산을 수행하는 튜링기계 등)들을 이용해 학습할 수도 있지만, 사용자 스스로 새로운 계산모델 사례를 정의하면서 관련된 계산이론을 학습할 수도 있다.



<그림 4> 계산 원리 학습도구의 사례[8]

<그림 5>는 유한상태기계 및 튜링기계 시뮬레이터 기능과 더불어, 관련된 학습/평가 자료와 대화식의 협동 학습 기능을 제공하는 계산이론에 대한 통합 학습 환경이다[9]. 이 도구 역시 이미 정의된 계산모델 사례들을 이용한 학습 형태와, 새로운 계산모델 사례들을 정의하는 학습 형태를 모두 지원한다.

<그림 5> 계산 원리 학습도구의 사례[9]

<그림 4>과 <그림 5>의 도구들은 대표적인 계산모델들과 관련 이론을 효과적으로 학습할 수 있게 지원하는 도구이다. 각 계산모델의 동작을 대화식으로 조작해 보면서 해당 모델의 구체적 기능과 개념을 익힐 수 있게 지원하지만, 다수의 계산들을 조합해 복합적인 계산을 정의할 수 없을 뿐만 아니라, 계산과 실세계 간의 상호작용을 정의하거나 관찰할 수 없다.

2) 프로그래밍 언어

프로그래밍 언어는 다양한 계산을 생성을 할 수 있는 알고리즘을 실제 혹은 가상 기계 상에서 수행될 수 있게 구현하는 도구이다. 일반적으로 기본적인 데이터 타입과 데이터 표현, 기본 연산을 제공하며, 이들을 조합해 보다 복잡한 데이터 구조와 알고리즘을 표현할 수 있는 규칙(문법)을 지원한다. 문제는 프로그래밍 언어가 계산 원리 학습이 아니라 실제로 동작하는 프로그램을 작성하기 위한 목적으로 만들어졌다는 것이다. 그래서 계산 원리 학습을 위해 기본적인 프로그램을 작성해 보려 해도 계산 원리와 무관한, 프로그래밍 언어 개념이나 문법들을 다수 익혀야 하는 부담을 준다. 이들 요소는 실제로 프로그래밍 언어 학습이나 프로그래밍 학습 자체를 어렵게 만드는 주된 요소로 작용하기도 한다. 그와 같은 요소들을 최소화해 곧 바로 알고리즘을 표현하는데 집중할 수 있도록 지원하는 교육용 프로그래밍 언어들이 다수 있지만, 이들 역시 궁극적으로 프로그래밍 언어 학습이나 프로그래밍 학습을 목적으로 하고 있어 효과적인 계산 원리 학습에 필요한 제반 요소들을 제공하고 있지 않다는 한계를 가지고 있다.

(1) C

C는 1972년 PDP-11 시스템에 구현된 범용 프로그래밍 언어로 초기에는 UNIX 운영체제를 위한 시스템 언어로 사용되었다. 오늘날 C는 대부분의 컴퓨터와 운영체제에서 사용될 수 있는 저급의 프로그래밍까지 지원하는 범용의 고급 프로그래밍 언어이다. C는 대부분의 대학이 기초 프로그래밍 교육 과정에서 선택하고 있는 언어로, 다음과 같은 특징을 가지고 있다[10].

- C는 강력하다. Pascal 보다 적은 개수의 예약어를 갖고 있다. C는 정확한

제어 구조와 자료형을 포함하고 있으며, 이를 논리적으로 의미 있게 사용할 경우 많은 제한 없이 사용할 수 있다. 또한 기능의 단순성 때문에 배우기 쉽다.

- C는 워크스테이션, 서버, 메인프레임을 위해 가장 많이 사용하고 있는 대화식 운영체제인 UNIX의 토착 언어이다. 또한 C는 PC를 위한 표준 개발 언어이다. MS-DOS와 OS/2의 대부분의 코드는 C로 작성되어 있다. 많은 윈도우즈 패키지, 데이터베이스 프로그램, 그래픽스 라이브러리, 그리고 다른 큰 응용 프로그램 패키지들이 C로 작성되어 있다.
- C는 이식성이 좋다. C는 모든 컴퓨터에서 동일하게 작동하는 표준 라이브러리 함수를 프로그래머에게 제공한다. 또한 C에 내장된 전처리기를 사용하면 시스템 독립적으로 코드를 작성할 수 있다.
- C는 간결하다. 강력한 연산자 집합을 가지고 있으며, 그 중 어떤 연산자는 프로그래머가 컴퓨터를 비트 수준까지 접근할 수 있게 한다. 증가 연산자 ++는 많은 컴퓨터에서 기계 언어와 직접적으로 연관되므로 효율적이다. 간접지정과 주소연산을 수식 내에 함께 사용할 수 있어서, 다른 언어로는 여러 문장으로 표현해야 하는 것을 한 문장이나 수식으로 표현할 수 있다. 이것은 프로그래머에게 있어서 효과적이라 볼 수 있다.
- C는 모듈성이 있다. C는 값에 의한 호출(call by value)로 인자를 전달하는 한 가지 방식의 외부함수만을 제공한다. 함수들의 중첩은 허용하지 않는다. 제한적인 형태의 프라이버시는 파일 내에서 기억장소 클래스 static을 사용함으로써 제공된다. 이러한 기능들은 운영체제가 제공하는 툴들과 함께 함께 사용자 정의 라이브러리 함수와 모듈화 프로그래밍을 쉽게 지원한다.
- C는 대부분의 컴퓨터에서 효율적이다. C언어의 어떤 구문은 컴퓨터 종속적이기 때문에, C는 컴퓨터 구조에 적합하도록 구현될 수 있다. 또한 컴퓨터는 그 구조에 종속적인 코드를 바로 수행할 수 있기 때문에 컴파일된 C코드는 매우 효율적일 수 있다.
- C는 C++와 JAVA의 기반이라 볼 수 있다. C 프로그래머가 일반적으로 많이 사용하는 구문과 방법이 C++와 JAVA 프로그래머에 의해 보편적으

로 사용됨을 의미한다. C를 배우는 것은 C++와 JAVA를 배우기 이전 첫 걸음이라 할 수 있다.

위와 같은 특성들로 인해 C 언어를 이용할 경우 물리적 기계의 특성을 보다 구체적으로 관찰하며 계산 원리를 학습할 수 있다. 하지만 문제는 그와 같은 수준의 C 프로그램을 작성하려면 상당 기간의 학습이나 작업이 필요하다는 것이다.

(2) JAVA

JAVA는 1996년에 개발된 객체지향 프로그래밍 언어이다. 자바의 중요한 특징 중 하나는 자바가상기계(**JVM**: *Java Virtual Machine*) 상에서 수행되기 때문에 운영체제 독립적이라는 것이다. 그래서 JAVA로 작성된 프로그램은 운영체제의 종류에 상관없이 실행이 가능하다. 이러한 이점으로 인해 다양한 기종의 컴퓨터와 운영체제가 공존하는 인터넷 환경에 적합한 언어이다. 또한, 풍부한 클래스 라이브러리를 제공하여 프로그래머는 이를 활용해 강력한 기능의 프로그램을 작성할 수 있다. Java의 특징을 정리하면 다음과 같다[11].

- JAVA는 운영체제에 독립적이다. 다수의 프로그래밍 언어는 특정 운영체제에서 개발된 프로그램을 다른 운영체제에 적용하기 위해 복잡한 작업이 필요했으나, JAVA는 자바가상기계(**JVM**)를 통해서 운영체제 독립적으로 프로그래밍을 할 수 있다. 이는 자바로 작성된 프로그램이 특정 운영체제나 하드웨어가 아닌 **JVM**과만 통신하고 **JVM**이 JAVA 응용프로그램으로부터 전달받은 명령을 해당 운영체제가 이해할 수 있도록 변환하여 전달하기 때문이다.
- JAVA는 객체지향언어이다. 객체지향개념의 특징인 상속, 캡슐화, 다형성이 잘 적용된 순수한 객체지향언어이다.
- JAVA는 자동으로 메모리를 관리한다. JAVA로 작성된 프로그램이 실행되면, 가비지컬렉터(garbage collector)가 자동적으로 메모리를 관리해주기 때문에 프로그래머는 메모리를 따로 관리 하지 않아도 된다. 가비지컬렉터가 없다면 프로그래머가 직접 사용하지 않는 메모리를 관리해야 한다.

- JAVA는 네트워크와 분산처리를 지원한다. 인터넷과 대규모 분산환경을 고려하여 설계된 자바는 풍부하고 다양한 네트워크 프로그래밍 라이브러리를 통해 비교적 짧은 시간에 네트워크 관련 프로그램을 효율적으로 개발할 수 있도록 지원한다.
- JAVA는 멀티쓰레드(multi-thread)를 지원한다. 프로그래밍에서 멀티쓰레드의 지원은 사용되는 운영체제에 따라 구현 방법도 상하며, 처리 방식도 다르다. 그러나 자바에서의 멀티쓰레드 프로그래밍은 시스템과 관계없이 구현이 가능하며, 관련된 라이브러리 제공으로 구현이 쉽다. 멀티쓰레드에 대한 스케쥴링(scheduling)은 자바 인터프리터(interpreter)가 담당한다.

위와 같은 특성들로 인해 Java 언어를 이용할 경우 객체지향 패러다임에 의해 생성되는 계산과, 가상기계가 계산 모델로 작용하는 계산을 접할 수 있고, 병행성을 가진 계산을 관찰하며 계산 원리를 학습할 수 있다. 하지만 Java 언어를 이용한 계산 학습 역시 상당 기간의 학습이나 작업이 전제되어야 한다는 문제를 안고 있다.

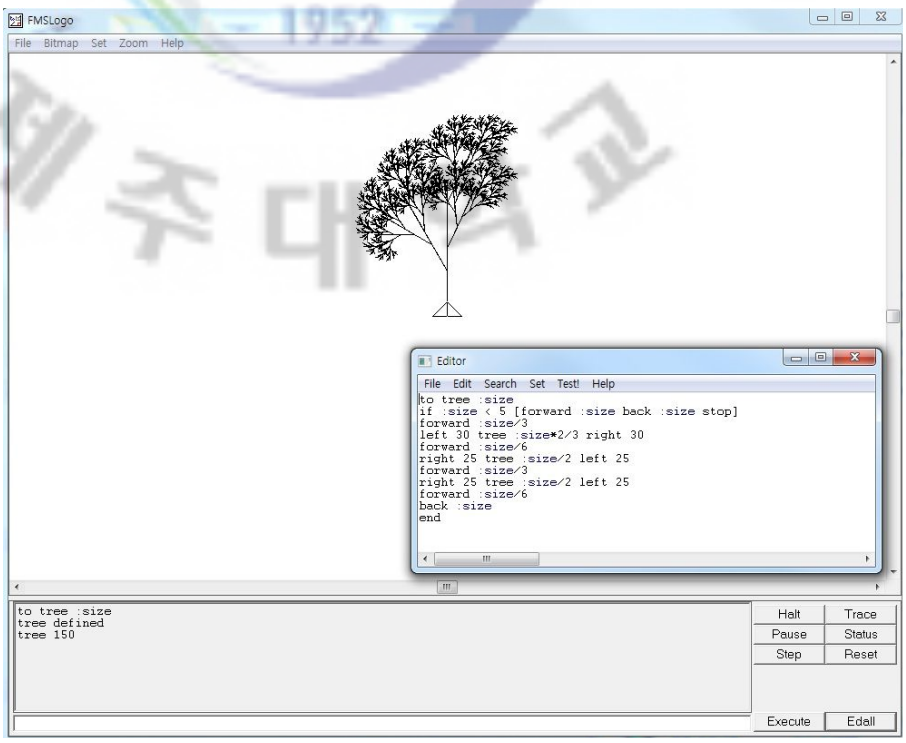
(3) 로고(LOGO)

로고는 1960년대 후반 MIT 인공지능 연구실에서 개발된 교육용 프로그래밍 언어이다. 로고는 학습자들의 자연스런 욕구를 채워줄 수 있는 교육 환경을 제공하기 위해 구성주의적 관점에서 학습자의 사고 체계를 구성할 수 있는 학습 환경을 제공한다. 로고 언어를 활용할 경우 전통적인 프로그래밍 언어와 달리 그래픽 요소인 터틀(*turtle*)을 이용해 프로그램 수행과정을 시각적으로 관찰하면서 프로그래밍 능력을 함양할 수 있게 된다. <그림 6>의 상단에 나무가 그려져 있는데 이는 나무의 뿌리 쪽에 위치한 팬을 든 터틀(삼각형으로 표시된 요소)의 이동을 적절히 제어해 그린 것이다. 로고의 특징은 다음과 같다.

- 그래픽 요소인 터틀의 속성과 움직임을 제어하며 프로그래밍을 익힐 수 해 준다. 터틀은 전후 방향으로 특정 거리를 이동하도록 지시할 수 있을 뿐만 아니라 현재 위치에서 특정 각도를 회전하도록 지시할 수도 있다. 또한 터틀은 펜을 가지고 있어, 펜을 캔버스에 대고 있도록 설정해 두면

터틀의 이동 궤적에 따라 그림이 그려지고, 펜을 떼고 있으면 터틀이 이동하더라도 그 궤적이 그림으로 그려지지 않는다.

- 인터프리터 언어로서 지시어 단위로 수행 결과를 확인할 수 있게 해 준다. 이는 연산 하나하나가 적용될 때마다 그로 인한 영향을 확인할 수 있다는 것이다.
- 로고는 지시어를 적정 순서로 나열하고 선택 혹은 반복 수행할 수 있게 지원해 문제해결 절차를 단계적, 구조적으로 표현할 수 있게 지원한다.
- 이미 정의된 프로시저들을 조합해 상위의 프로시저를 정의·활용할 수 있게 지원함으로써 문제해결에 통합적·체계적으로 접근할 수 있게 도와준다[12].



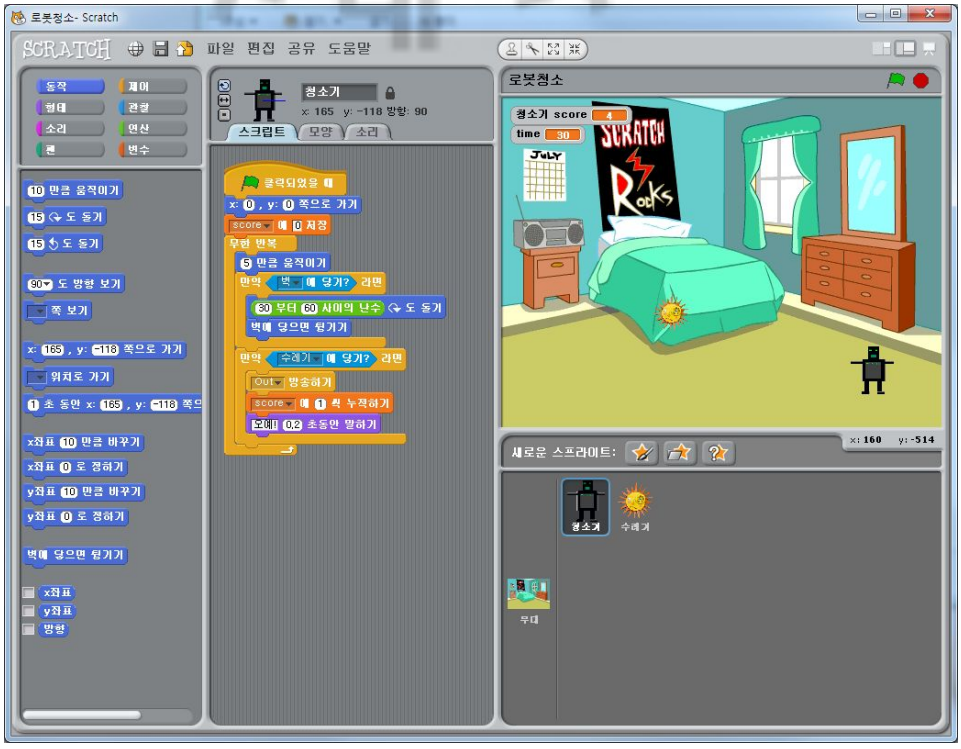
<그림 6> LOGO 수행화면의 예

위와 같은 특성들로 인해 로고 언어를 이용할 경우 대화식 계산을 경험할 수 있고, 각 연산의 결과로 발생하는 표현(상태)의 변화가 외부 세계에 어떤 영향을 줄 수 있는지 관찰하며 계산 원리를 학습할 수 있다. 하지만 로고 언어가 수행할

수 있는 계산(다룰 수 있는 표현)의 유형이 한정되어 있고, 실세계와의 상호작용 양식이 제한되어 있어 계산의 다양한 측면을 체계적으로 학습할 수 있도록 지원하지는 못한다.

(4) 스크래치(Scratch)

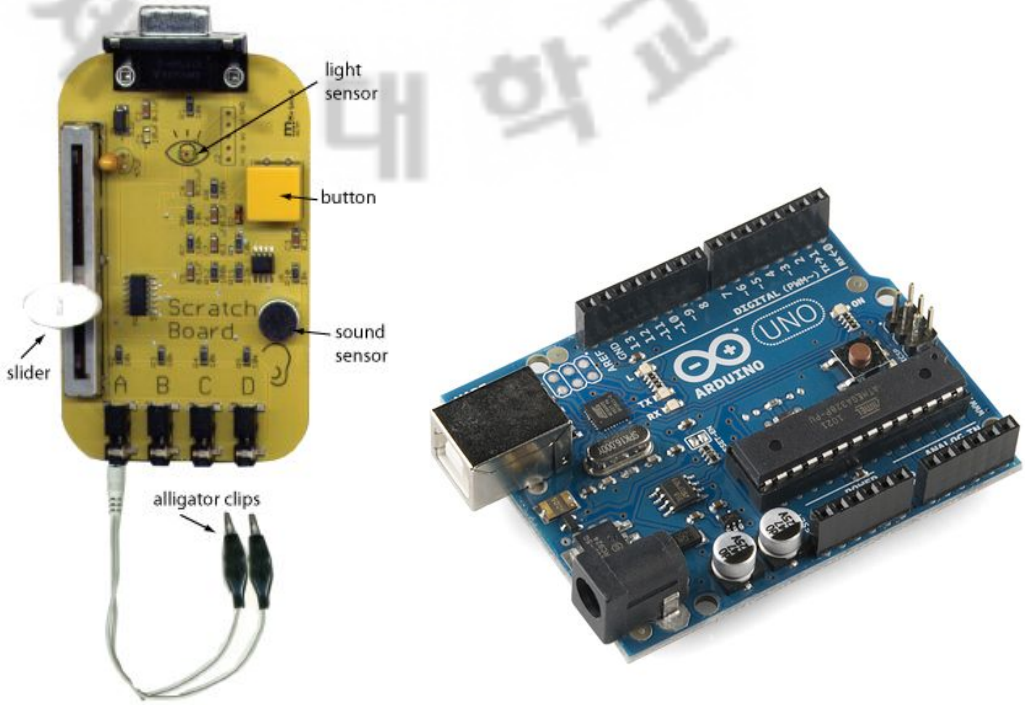
스크래치는 MIT 미디어 연구소와 UCLA의 연구진들이 미국과학재단, Intel, Google, MS 등의 지원 하에 공동으로 개발한 교육용 프로그래밍 언어로, 객체와 이벤트를 이용하여 게임이나 스토리, 애니메이션 등을 제작할 수 있는 그래픽 기반 프로그래밍 툴이다[13]. 로고가 터틀의 움직임을 제어할 수 있게 하듯이, 스크래치는 '스프라이트(*sprite*)'라 불리는 그래픽 요소를 제어해 애니메이션을 만들 수 있게 지원한다. 블록들을 연결해 조립 작품을 만들 듯이, 스크래치 프로그래밍은 단위 기능을 가진 여러 가지 범주(동작, 제어, 형태, 관찰, 소리, 연산, 펜, 변수 등)의 블록들을 선택·조합해 수행될 작업의 절차, 즉 스크립트를 작성하는 작업이다. <그림 7>에서 작성된 스크립트의 예와 그 수행 화면을 볼 수 있다.



<그림 7> Scratch 수행화면의 예

스크래치의 특징을 정리하면 다음과 같다[14].

- 블록 쌓기 놀이 형식의 활동을 통해 학습하는 프로그래밍(*building-block programming*) 언어로, 프로그래밍 툴이 제공하는 8개의 블록 카테고리에서 원하는 블록을 선택하여 조립하는 것으로 프로그램을 작성할 수 있다.
- 다양한 미디어를 활용하여 재미있게 학습하는 프로그래밍이 가능하다.
- 뛰어난 공유성(*share-ability*)을 제공하여 프로그래밍 과정에서 다른 사용자와 제작 내용 및 산출물을 공유할 수 있으며, PC, 모바일 장치, 태블릿(tablet) 등 여러 형태의 기기에서 공유가 가능하다.
- <그림 8>과 같은 센서보드 등을 통해 물리적 장치(모터, 전구, 센서, 동작 컨트롤러)와의 상호작용을 지원해, 학습자의 호기심 충족과 학습동기 부여가 가능하다.



<그림 8> Scratch board의 사례

위와 같은 특성들로 인해 스크래치를 이용할 경우 다양한 미디어 효과나 장치

구동과 연계된 계산 과정을 학습할 수 있고, 시각화된 프로그램 구성 요소들을 조합하는 방식으로 기본 알고리즘을 작성할 수 있다. 하지만 스크래치가 지원하는 이벤트의 종류가 한정되어 다양한 유형의 상호작용을 설정할 수 없고, 복합적인 데이터 표현이나 절차적 표현을 정의·활용할 수 없으며, 계산 과정에서 나타나는 표현들이 미디어 효과나 장치 구동과 어떤 방식으로 연계되는 지에 대한 관찰이 어렵다는 문제를 안고 있다.

3) 로봇 제어 프로그램 개발 도구

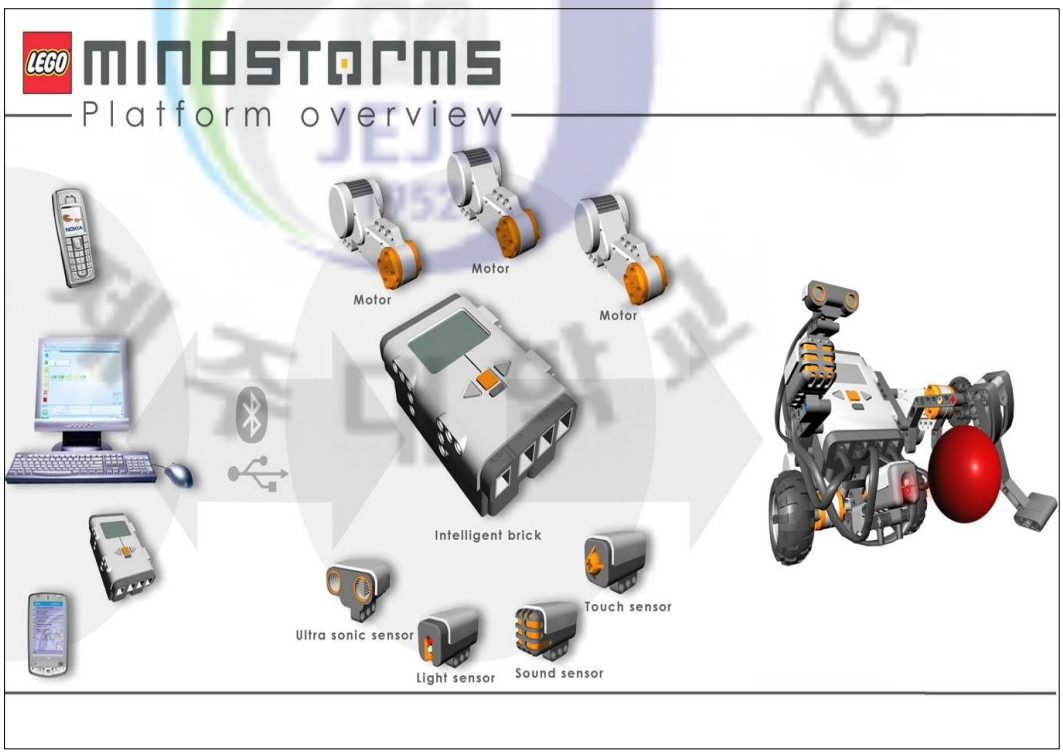
로봇 프로그래밍은 프로그래밍 교육의 효과를 높이기 위한 동기부여의 한 방법으로, 혹은 물리 시스템 제어와 관련된 프로그래밍 교육의 한 영역으로서, 혹은 IT 융합교육의 주 요소 중 하나로서 크게 주목 받고 영역이다. 로봇 프로그래밍 활동을 통해 계산이 실세계와 어떻게 상호작용할 수 있는지를 실질적으로 실험하며 계산 원리를 학습할 수 있지만 프로그래밍 과정을 거쳐야 계산과 실세계 간의 상호작용을 관찰할 수 있어 로봇 프로그래밍 학습 역시 프로그래밍 언어 학습에 내재된 교육적 취약점을 그대로 가지고 있다.

일반적으로 로봇 키트는 로봇 제작용 부품들과 로봇 제어 프로그램 개발 도구로 구성되는데, 여기서는 레고 마인트스톰 NXT(LEGO® Mindstorms® NXT)용 제어 프로그램 개발 도구에 대해 살펴본다. 레고 마인드스톰 NXT(이하 NXT)는 LEGO사와 MIT가 1998년부터 연구하여 개발한 사용자의 명령에 따라 움직일 수 있도록 프로그램이 가능한 블록 조립형의 교육용 로봇 시스템이다. NXT의 첫 번째 버전은 RIS(Robotics Invention System)라는 이름으로 출시되었으며, 핵심 부품인 컨트롤러는 RCX라고 명명되었다. 이후 LEGO사는 이를 더욱 발전시켜 2006년 지능형 브릭(Intelligent Brick), 인터렉티브 모터, 센서, 블루투스 통신 등의 기능이 추가된 로봇 시스템인 NXT를 개발하였다[15]. <그림 9>에서 보듯이 NXT의 핵심 요소는 로봇 제어용 마이크로컴퓨터인 NXT 지능형 브릭으로, 다양한 입력 센서들과 다수의 출력 모터를 연결해 NXT가 생성하는 계산과 로봇 간의 상호작용이 가능케 지원해 준다. NXT의 특성은 다음과 같다.

- NXT는 조립이 쉽고 전문적 전공지식이 없이도 프로그래밍이 가능하다는

특성으로 프로그래밍, 알고리즘 교육 외에 기초 로봇 공학, 임베디드 시스템, 인공지능 등의 교과목에서 제한 요소를 반영한 설계 교육에 활용될 수 있다[16].

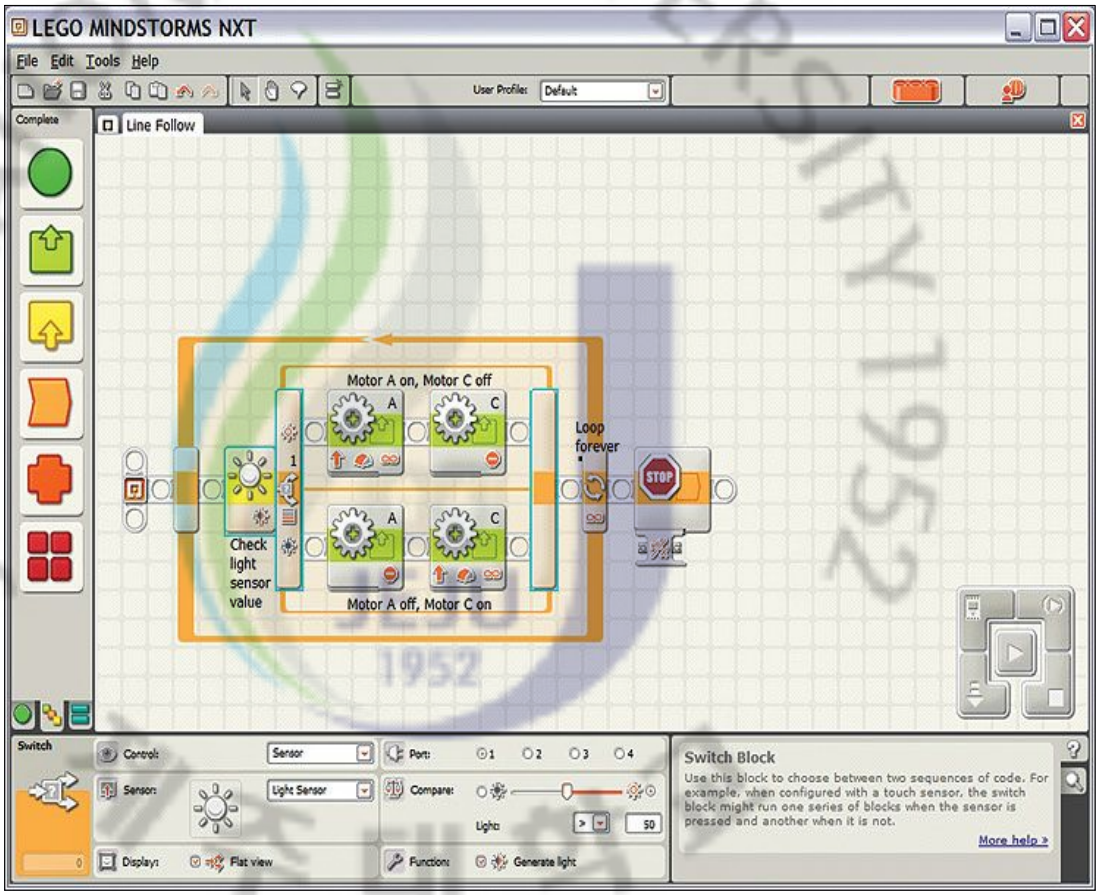
- 로봇 제어 프로그래밍 언어로 C와 유사한 NXC가 있으며, 자바 프로그래밍을 지원하는 leJOS NXJ 환경도 있다[17]. leJOS는 NXT 펌웨어이고 NXJ는 자바 기반의 프로그래밍 도구이다. 이외에도 NXT-G나 LOBOLAB과 같은 그래픽 프로그래밍 도구들도 제공된다.



<그림 9> Mindstorm NXT 플랫폼 개관

(1) NXT-G

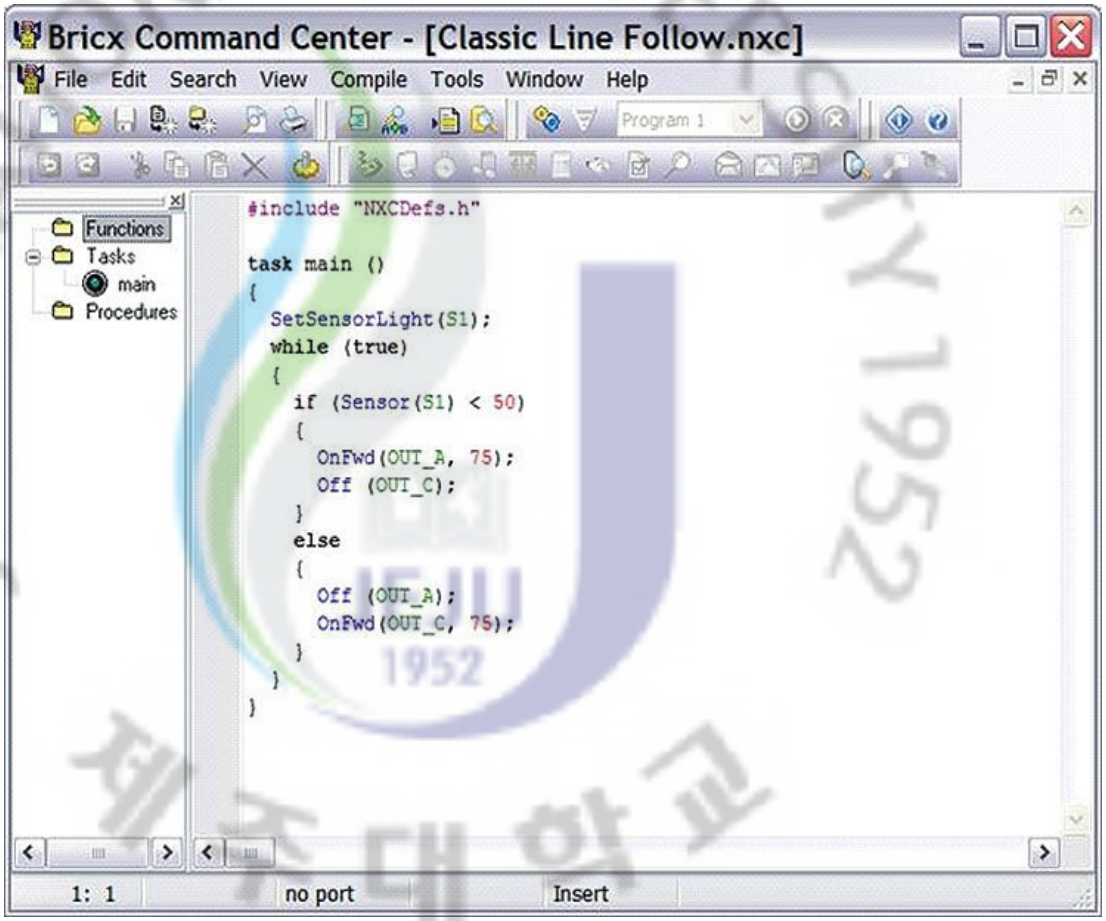
NXT-G(Lego Mindstorms Education NXT Software)는 LabView 언어 기반의 그래픽 프로그래밍 툴이다. 별도의 프로그래밍 언어를 습득하지 않아도 <그림 10>에서와 같이 플로우차트와 유사한 형태의 프로그램을 개발할 수 있다.



<그림 10> NXT-G 수행화면의 예

(2) NXT-C

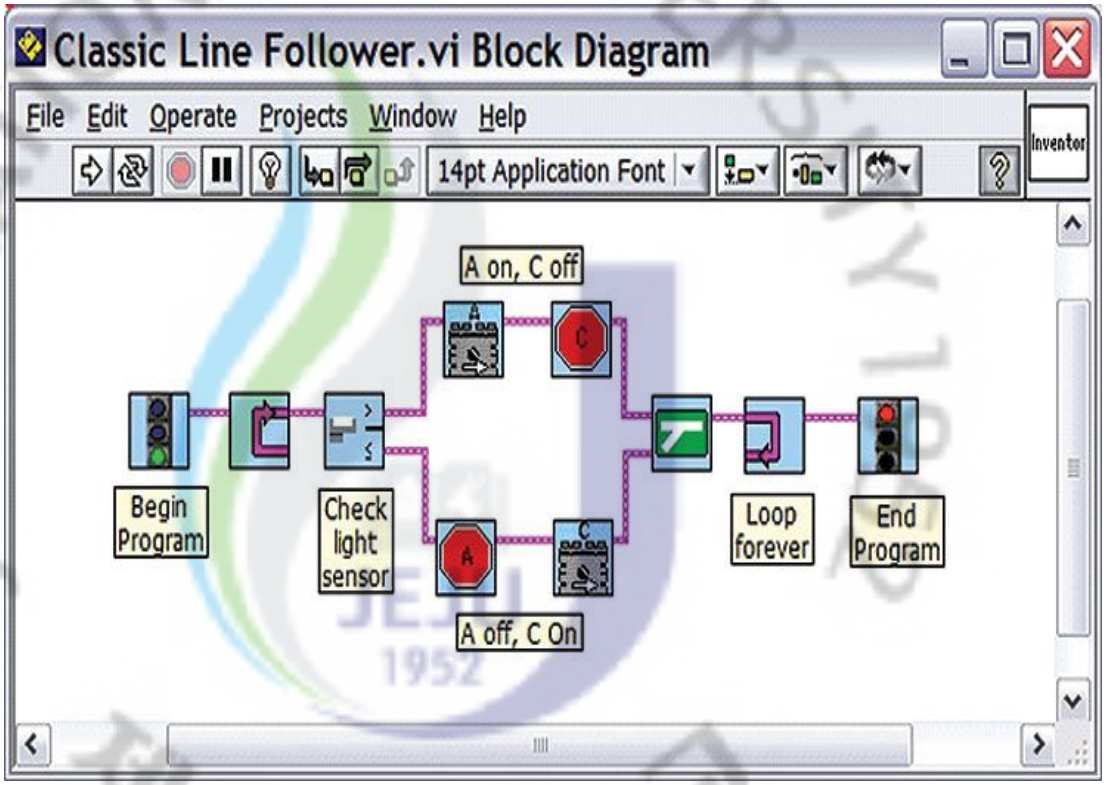
<그림 11>에서 보듯이 NXT-C는 C언어와 유사한 NXC(Not eXactly C)로 NXT 제어 프로그램을 개발할 수 있게 지원하는 툴이다. NXC는 C 언어처럼 **int** 타입의 변수 사용은 지원하지만 **float** 타입의 변수 사용은 지원하지 않는다. NXT-C가 NXT-G와 동일한 펌웨어를 사용하기 때문에 NXT-C와 NXT-G 중 어느 하나를 이용해 개발한 프로그램이라면 NXT 브릭에 전송해 로봇을 작동시킬 수 있다. NXT-C는 실시간 디버깅도 지원한다.



<그림 11> NXT-C 수행화면의 예

(3) LOBOLAB

ROBOLAB은 LabView 언어를 이용해 LEGO RCX 제어 프로그램을 개발할 수 있게 지원하는 툴로, 현재는 그 기능이 확장되어 NXT 제어 프로그램도 개발할 수 있다. ROBOLAB은 NXT-G와 같이 그래픽 환경을 가진 프로그래밍 툴이지만 NXT-G 보다 더 정교한 프로그램 개발이 가능하다. ROBOLAB은 **`int`** 타입과 **`float`** 타입의 계산을 지원한다.



<그림 12> LOBOLAB 수행화면의 예

Ⅲ. 계산 원리 학습도구의 기능요소

1. 계산 원리 학습의 내용 및 방법

1) 관련 용어 정의

컴퓨터과학은 본질적으로 엄밀한 정확성을 요구하는 분야이다. 하지만 아이러니하게도 소프트웨어 공학, 구조적 프로그래밍, 클라우드 컴퓨팅 등 컴퓨터과학 분야의 주요 용어 중 명확한 정의 없이 사용되고 있는 것들이 많다. 물론 그 이유로 상정할 수 있는 것들은 많다. 컴퓨터과학의 역사가 짧은 반면, 그 성장 속도가 너무 빠르기 때문에 어떤 용어나 개념이 다듬어지고 정립될 시간이 없이 도입·사용되기 때문이라 해석할 수 있다. 혹은 컴퓨터과학이 타 분야와 융합되면서 형성되는 외연이 너무 빨리 확장되어 수많은 대상과 개념들에 대한 용어가 체계화되지 못한 채 혼재하기 때문이라 이해할 수도 있다. 어떻든 이제 우리 스스로가 명시적으로 “우리는 지금 우리가 사용하는 주요 용어에 대해 명확한 정의를 탐색하고 있다”라고 선언하지 않으면, 다른 이들 혹은 우리 자신이 “우리는 부정확한 것에 만족하는 사람들이다”라고 생각하도록 허용하게 될 것이다. 우리는 ‘우리가 하고 있는 것이 무엇인지’ 기술할 때 사용하는 기본 용어를 보다 정확히 정의할 필요가 있다. 여기서는 컴퓨터과학을 “자연적/인공적 정보처리 (계산)에 대한 학문”이라 간주하고, 이와 관련된 주요 용어들은 보다 정확히 정의하려 한다. 이는 계산 원리 학습 도구와 관련해 이후에 설명할 내용의 의미를 부다 명확히 하기 위함이다.

계산(computation)은 계산모델(computation model)의 관점에서 설명되어야 한다. 왜냐하면 계산적 형태를 보이는 시스템들의 본질이 다양하고, 어떤 사람이 연구 중인 계산 시스템이나 문제의 유형에 따라 그 사람이 계산에 대해 갖고 있는 인식이 다를 수 있기 때문이다. 계산은 그것이 기초하고 있는 계산모델의 관점에서 정의된 처리과정이다. 여기서 계산모델은 컴퓨팅 시스템에 대한 수학적

추상체이다. 1930년대에 제안된 4가지 계산모델(재귀함수, rewriting rules, lambda-calculus, Truing machine)을 비롯해 probabilistic machines, non-deterministic machines, parallel program schemata, Petri Nets, neural networks, DNA string systems 등 다양하다. 이들 중 계산에 대한 대표적인 참조 모델은 단연 튜링기계인데, 이는 튜링기계가 디지털 전자컴퓨터와 가장 많이 닮았기 때문이다. 다중테이프 튜링기계, lambda-calculus, random access machines, production systems, cellular automata는 물론, 모든 범용의 프로그래밍 언어(general-purpose programming language)는 모두 튜링완전 계산모델들인데, 튜링기계와 동등한 계산모델들이 이렇게 다양한 이유는 튜링기계 프로그램의 형태로 알고리즘을 구현하고 싶어 하는 이들이 거의 없기 때문이다. 계산모델은 항상 해당 계산을 수행하는 에이전트(agent)를 명시하고 있다. 따라서 계산은 어떤 에이전트(튜링기계, 전자컴퓨터, continuous device, 인간의 두뇌 등)에 의해 생성되는 처리과정으로 간주될 수 있다.

- 정보처리(information process)는 표현(혹은 상태)의 순차(a sequence of representations)이다. 계산은 표현 간 전이들이 또 다른 표현(알고리즘)에 의해 제어되는 정보처리(계산은 알고리즘에 의해 야기된 데이터 표현 상태들의 순차)이며, 계산 수행의 주체는 에이전트이다. 컴퓨터는 정보처리 중 일부를 구현하는 수단이지만, 모든 정보처리가 컴퓨터로 구현되지는 않는다. 어디에나 표현이 존재하기에, 계산 역시 어디에나 존재한다.
- 계산은 그것을 수행하는 에이전트의 본질에 의해서가 아니라 심볼(symbol)들로 구성된 표현(representation)의 순차에 의해 정의된다. 표현은 정보를 전하는 심볼들의 패턴(a pattern of symbols)이다. 표현 순차 상의 어떤 한 표현에서 그 다음 표현으로의 전이는 어떤 에이전트에 의해 수행되는 연산(operation), 즉 연산자(operator)의 적용에 의해 야기된다.
- 모든 표현은 매개체(carrier)라 불리는 물리적 현상 속에 담겨진다. 종이 위의 잉크나 디스크 표면상의 자기 패턴, 혹은 전자기 파형이나 컴팩트디스크 상의 요철, DNA 내부 아미노산의 3차원 접힌 배열 등이 그 예이다. 표현은 '사물의 전략적 배열(strategic arrangements of stuff)'이다.
- 계산을 발생시킬 수 있는 수많은 매체 중 하나인 컴퓨터(computer)는 절차

적 표현(procedural representation)의 제어 하에 데이터 표현(data representation)을 변환하는 장치로 에이전트의 범주에 포함된다. 입출력 장치들 역시 물리적 효과와 연계된 표현을 다루는 에이전트로 간주될 수 있다.

- 계산은 이산 처리(discrete process)로서 심볼들로 정의되는 상태들의 순차이다. 한 상태에서 다른 상태로의 전이는 하나 혹은 다수의 처리에 의해 빚어진 결과이다. 여기서 하나의 처리는 한 컴퓨터상에서 수행 중인 알고리즘이나 컴퓨터상의 응용과 상호작용 중인 인간, 혹은 인터넷 원격 사이트의 또 다른 컴퓨터, 심볼 표현이 가능한 상태를 가진 물리/생물 시스템일 수도 있다.

2) 계산 원리 학습 콘텐츠 사례

계산은 정보가 있는 모든 곳에서 탐색될 수 있다. 정보를 나타내는 표현들의 순차가 계산이기 때문이다. 우리 개개인이 감당할 수 없을 만큼 많은 정보가 지금 이 순간에도 생성·처리되고 있고 언제 어디서나 원하는 정보에 접근할 수 있으며, 자연 현상 속에서 일어나고 있는 심오한 정보처리들이 여러 학문 영역에서 지속적으로 발견되고 있는 시대적 상황을 고려할 때, 계산 원리 학습 콘텐츠는 언제 어디서나 찾아질 수 있는 것이다. 따라서 계산 원리 학습은 우리 주변의 사물과 현상, 혹은 관심 영역 속에서 정보처리를 발견·표현하고, 또 발견된 표현의 순차를 생성할 수 있는 알고리즘이나 시스템을 구상해보는 활동 속에서 자연스럽게 이루어질 수 있다. 계산 원리 학습에 사용할 수 있는 몇 가지 콘텐츠 사례를 든다면 다음과 같다.

- 일상의 사례
  - **[표현 찾기]** 주변에서 여러 가지 표현을 찾아 그것이 어떤 매개체 상에 어떤 정보를 어떻게 나타내고 있는지 말하라.
    ☞ 종이 위의 잉크, 디스크 표면상의 자기 패턴, 전자기 파형, 컴팩트 디스크 상의 요철, 나무의 나이테 등의 매개체 상에 나타난 심볼들의 패턴
  - **[내일 날짜 알아내기]** 내일이 몇 월 며칠인 지에 대한 질문에 답을 찾

는 과정을 표현하라.

- **[양치질 하는 법]** 양치질 과정을 5 단계와 15 단계로 구분해 각각 기술하라.
- **[역기 정리]** 체육관 역기걸이에 역기들이 임의의 순서로 걸려 있다. 이들을 역기 무게의 오름 순으로 정리하려 할 때 들어 옮길 역기의 최소 무게 합을 구하시오. 단, 작업의 안전을 위해 한 번에 하나의 역기를 들어 옮기되, 역기걸이에 놓인 역기를 바닥에 내려놓거나, 한 역기걸이에 놓인 역기를 비어있는 다른 역기걸이로 옮겨 놓거나, 바닥에 놓인 역기를 비어있는 역기걸이로 옮겨놓는 세 가지의 이동 방식만 허용된다.

  ☞ 역기걸이에 걸려 있는 역기(무게)들의 순서가 아래와 같을 때,

  (20 40 50 10 90 30 80 70 60)

  다음 표현을 이용해 보다 효과적으로 원하는 해답을 구할 수 있음을 보여줄 경우, 복합적인 표현구조 설정의 필요성과 문제 해결에 적합한 표현 구조 탐색의 중요성을 학습할 수 있다.

  ((10 4) (20 1) (30 6) (40 2) (50 3) (60 9) (70 8) (80 7))

- **[촌수 계산]** 사람들 간 '부모-자식' 관계를 이용해, 지정된 두 사람 간의 촌수를 구하시오. 단, 각 사람은 **1**부터 **N**까지의 서로 다른 정수 번호로 표시된다.

  ☞ 12명의 사람들 간의 '부모-자식' 관계(**x**가 **y**의 부모일 때 **(x y)** 형태의 쌍)가 아래와 같이 주어졌을 때 두 사람 **4**와 **6** 간의 촌수를 구해야 할 때,

  ((1 4) (2 1) (5 7) (6 12) (2 3) (1 5) (8 6) (5 8))

  다음 표현이 원하는 해답을 구하는데 어떻게 사용될 수 있는지와 이 표현을 어떻게 얻을 수 있는지 등을 이야기할 경우 특정 구조의 표현들을 다루는 알고리즘을 어떻게 정의하는지 학습할 수 있고, 다양한 표현 구조를 학습해 두었을 경우의 유익을 알 수 있다.

  (2 0 2 1 1 8 5 5 0 0 0 6)

• 수학 영역의 사례

- [곱셈 개념]

① 각 곱셈을 덧셈으로 표기하고 답을 구한 후, 답을 구할 때 요구되는 덧셈의 반복(iteration) 횟수를 구하라.

② 피연산자 순서가 역순인 두 곱셈의 답을 구할 때 요구되는 덧셈의 반복 횟수를 구한 후 어떤 것이 효율적인지 답하라.

- [제곱근 구하기] EDA(estimate-divide-average) 알고리즘, 선형탐색(linear search), 이진탐색(binary search)을 적용해 특정 값 n의 제곱근을 구하라.

① n = 60, 초기 추정값 g = 2에 대해 EDA 알고리즘을 적용한 계산 과정을 다루면서 알고리즘이 다수의 계산을 생성할 수 있다는 개념을 학습할 수 있다(아래 계산에서 ⇒ 는 연산자로서 함수 λg.(n/g+g)/2에 대한 추상개념을, 각 숫자는 '표현의 순차'를 구성하는 요소로서의 표현을 나타냄).

2 ⇒ 16 ⇒ 9.875 ⇒ 7.975 ⇒ 7.749 ⇒ 7.746

② 추정값을 선형적으로 증가시켜 가며 해답에 근접해 가는 선형탐색(linear search), 특정 경계값의 쌍(상태를 표현)으로 시작해 해답에 충분히 가까워질 때까지 그 간격을 줄여가는 이진탐색(binary search)을 적용해 특정 값의 제곱근을 구하는 과정들을 다루면서 연산자에 대한 추상개념, 계산시간 면에서의 효율성(efficiency), 계산의 정확성(correctness), 무한 반복, 조건에 따른 선택 개념 등을 학습할 수 있다.

☞ 선형탐색: λg.g+0.1

☞ 이진탐색: λa,b.if((a+b)/2)2>n,

(a,(a+b)/2);else((a+b+/2,b)

• 다른 학문의 사례

- [영어: 랩과 시] 랩과 시가 어떤 점에서 유사한지 말하고 스타일이나 특성에 따라 랩퍼(rapper)와 시인들을 대응시켜라.

- [역사: 인구이동의 역사적 원인] 역사적 사건들과 통계 데이터를 조사

해 시대에 따른 인구 이동 원인을 분석하라.

- [사회: 개인의 관계와 사회] 전교생들의 친구 관계를 조사해 2개 이상의 반에 걸친 5명 이상의 친구 그룹(서로가 서로를 친구라고 인식하고 있는 학생 그룹)을 찾아라.

3) 계산 원리 학습의 방향

계산은 자연이 행하는 어떤 것이며 컴퓨터는 계산을 연구하는 도구이다. 계산은 표현을 다루고 조작해 목표하는 표현을 얻는 과정이기에, 컴퓨팅의 중심은 컴퓨터가 아니라 표현이다. 따라서 계산 원리 학습은 표현을 중심으로 이루어져야 한다. 여기서 중요한 문제는 표현을 중심으로 한 계산 원리 학습의 큰 방향을 설정하는 일이다. 즉, 어떤 요소들을 어떤 관점에서 학습해야 할 지 정하는 일이다. 본 논문은 앞서 소개한 컴퓨팅 패러다임[7]을 근간으로 계산 원리 학습의 방향을 설정하고 있다. 컴퓨팅 패러다임은 컴퓨터과학이 세계를 어떻게 바라보고 문제의 해답에 어떻게 접근하는지를 나타내는 것이기에, 컴퓨터과학과 관련된 모든 학습은 궁극적으로 컴퓨팅 패러다임과 연계될 필요가 있다는 판단에서다. 특히 계산 원리는 컴퓨터과학의 핵심 원리이기에 그에 대한 학습은 처음부터 컴퓨팅 패러다임에 맞추어 설계될 필요가 있으며, 그렇게 되어야 계산 원리 학습자들이 컴퓨터과학의 본질에 보다 정확히 효과적으로 접근할 수 있게 된다. 표현을 중심으로 이루어져야 할 계산 원리 학습의 요소들과 기본 관점을 컴퓨팅 패러다임의 틀에 맞추어 정리하면 다음과 같다.

- 초기화**(Initiation)** : 관찰 혹은 구축 대상 시스템이 유한(종료 가능)한 정보처리 과정으로 표현될 수 있는지, 혹은 무한(지속 대화식)의 정보처리 과정으로 표현될 수 있는지 결정한다.
  - 다양한 표현(선형/비선형 표현, 이산적/연속적 표현, 비트 패턴, 방법의 표현)과 표현의 양상(패턴 구성의 규칙과 매개체), 표현의 순차로서 정보처리(계산), 표현 간 전이의 주체로서 에이전트(인공물과 자연물), 인터럽트 처리와 열린 계산
  - 사물, 상황, 현상 속에 발현된 계산을 발견·관찰하고 적합한 표현들을 나열해 그 과정을 나타낼 수 있어야 한다.

- 개념화**(Conceptualization)** : 대상 시스템의 행동을 생성하는 계산모델(예: 알고리즘, 컴퓨팅 에이전트 등)을 설계 혹은 발견한다.
  - 다양한 계산모델(에이전트), 원시 표현(primitive representation), 원시 연산자(primitive operator), 제어구조(호출, 순차, 선택, 반복), 복합 표현(다수의 표현을 조합한 표현)과 복합 연산(다수의 연산을 조합·추상화한 상위 연산자), 문제해결과 알고리즘
  - 문제를 정형화하고 해결하는데 적합한 계산모델을 선택·설정·적용할 수 있어야 하며, 원시 표현 및 연산을 조합해 보다 높은 차원의 계산을 표현할 수 있어야 한다.
- 실현**(Realization)** : 설계된 처리과정을 명령 수행 능력이 있는 매체 내에 구현한다. 발견된 처리과정에 대한 시뮬레이션 혹은 모델을 설계한다. 정보처리과정의 행동을 관찰한다.
  - 명령 수행 능력이 있는 다양한 매체, 물리적 효과를 가진 표현, 외부세계와 상호작용하는 에이전트
  - 계산과 계산, 혹은 계산과 외부세계 간 상호작용을 이해·활용할 수 있어야 한다.
- 평가**(Evaluation)** : 해당 구현에 대해 논리적 정확성, 가설과의 일관성, 성능 제약, 원 목표 충족 여부 등을 검증한다. 필요할 경우 구현을 개선한다.
  - 표현의 크기, 문제해결 시간, 복잡도
  - 표현을 다루는 과정에서 시간적/공간적 자원이 어떻게 사용되는지 이해할 수 있어야 한다. 여러 수준에서 계산의 과정을 고찰하며 개선 가능한 요소들을 찾아낼 수 있어야 한다.
- 실행**(Action)** : 산출된 결과를 실세계에 반영한다. 지속적 평가를 위해 모니터링 한다.
  - 표현을 매개로 한 명령 수행 매체 간 상호작용
  - 계산과 계산, 혹은 계산과 외부세계 간 상호작용을 이해·활용할 수 있어야 한다.

표현, 계산, 계산모델, 알고리즘 등 위에서 언급한 계산 원리 학습 요소 대부분

에 대한 기초 학습은 컴퓨터의 도움 없이도 가능하다. 하지만 표현이나 계산과 관련된 주요 기능을 제공하는 도구가 있다면 계산 원리 학습을 보다 효과적으로 수행할 수 있을 것이다. 본 연구는 위에서 제시한 계산 원리 학습 방향을 고려해 계산 원리 학습도구가 갖추어야 할 기능적 요건들을 도출하였다.

2. 계산 원리 학습도구의 기능적 요건

1) 다양한 원시 표현과 원시 연산자 제공

학습자가 사물, 상황, 현상 속에 발현된 다양한 계산(표현의 순차)들을 적합한 계산모델(에이전트, 알고리즘)을 선택해 생성할 수 있도록 지원해야 한다. 이를 위해 계산 원리 학습도구가 제공해야 할 기능 요소는 다음과 같다.

- 튜링기계, 유한상태기계 등 다양한 '계산모델 에이전트' 제공
  - 적합한 계산모델을 이용해 필요한 원시 연산자를 정의할 수 있게 지원
  - 필요에 따라 새로운 '계산모델 에이전트'를 학습도구에 추가할 수 있게 지원
- 입력 장치로 입력되는 표현들을 적정 형태로 전환해 주는 다양한 '장치 에이전트' 제공
  - 입력을 지원하는 '장치 에이전트'들은 학습자가 다양한 표현(패턴 구성의 규칙과 매개체)을 다룰 수 있도록 각 표현을 특정 계산모델에 적합한 표현으로 전환
  - 필요에 따라 새로운 입력지원 '장치 에이전트'를 학습도구에 추가할 수 있게 지원

2) 복합 표현/연산자 정의 기능 지원

학습자가 복합적인 표현이나 계산을 이용해 보다 큰 문제에 효과적으로 접근할 수 있도록 지원해야 한다. 이를 위해 계산 원리 학습도구가 제공해야 할 기능 요소는 다음과 같다.

• 다른 표현들을 나열·조합해 복합적인 표현을 정의할 수 있게 지원
• 다른 연산자들이 적용될 순서를 명시해 복합적인 연산을 정의할 수 있게 지원
  - 연산자들의 적용 순서를 제어할 수 있게 기본 제어구조(호출, 순차, 선택, 반복) 제공
  - 서로 다른 에이전트가 수행하는 연산자들의 조합을 허용(다수의 에이전트들이 연계·협력해 하나의 계산을 생성)함으로써 다수의 계산모델이 적용된 계산을 효과적으로 다룰 수 있게 지원
  - 복합 연산을 '사용자 정의 에이전트'로 정의·등록해 활용할 수 있게 지원

3) 외부환경과의 다양한 형태의 상호작용 지원

학습자가 계산 과정에서 개입하거나 면밀히 관찰할 수 있도록, 혹은 계산이 외부 세계와 상호작용하는 방식을 구체적으로 이해·설정할 수 있도록 지원해야 한다. 이를 위해 계산 원리 학습도구가 제공해야 할 기능 요소는 다음과 같다.

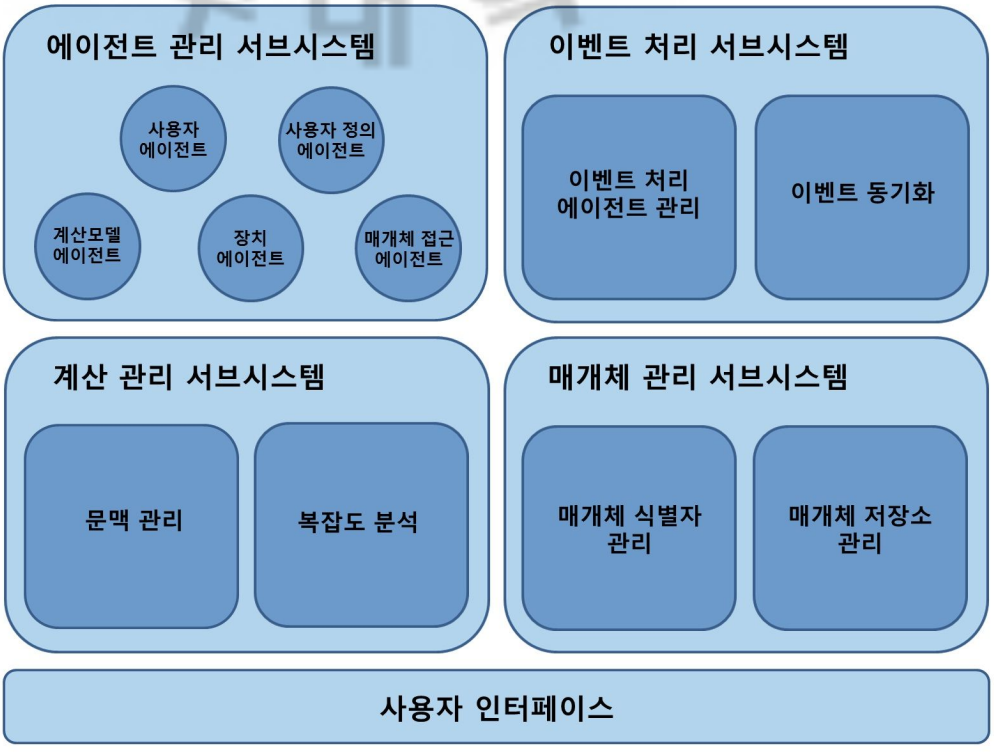
• '사용자 에이전트'를 제공해 사용자 스스로 에이전트가 되어 계산을 수행·관찰할 수 있게 지원
• 특정 표현에 '매개체 접근 에이전트'를 연계시킬 수 있게 지원
  - 표현을 수정하거나 다양한 방식으로 관찰할 수 있게 지원
  - 필요에 따라 새로운 '매개체 접근 에이전트'를 학습도구에 추가할 수 있게 지원
• 다양한 외부 장치(프린터, 스피커, 보드, 로봇 등)에 대응하는 '장치 에이전트' 제공
  - 특정 표현을 해당 장치에 대한 물리적 효과로 전환
  - 계산 과정이나 결과가 물리적 장치의 작동에 어떤 영향을 주는지 관찰할 수 있게 지원
• 대화식 계산 지원
  - 이벤트 처리 에이전트 관리 지원
  - 이벤트 처리를 위한 문맥 관리 기능 지원

- 이벤트에 대한 동기화 기능 지원

4) 계산 생성 과정에 대한 통제 및 분석 기능 지원

학습자가 에이전트(사람, 알고리즘, 계산모델 등)의 계산 생성 과정을 적절히 통제하며 관찰·분석할 수 있도록 지원해야 한다. 이를 위해 계산 원리 학습도구가 제공해야 할 기능 요소는 다음과 같다.

- 에이전트의 연산자 적용에 대한 통제 및 분석 기능 제공
  - 적정 단위로 연산자를 적용하며 계산 과정을 분석할 수 있게 지원
  - 적용된 연산자 수의 합을 산출해 제공함으로써 시간 복잡도 분석 지원
- 매개체 재사용 및 분석 기능 제공
  - 매개체에 대한 참조(reference) 설정 기능을 제공함으로써 특정 매개체에 담긴 표현을 적정 상황, 적정 시점에 재사용할 수 있도록 지원
  - 표현이 담긴 매개체들의 크기 합을 산출해 제공함으로써 공간 복잡도 분석 지원



<그림 13> 계산 원리 학습도구의 시스템 구성

3. 관련 도구 분석

계산 원리 학습도구가 갖추어야 할 4가지 기능 요소들에 대해 기존의 계산 원리 학습 관련 도구들을 비교·분석한 내용은 다음에 제시된 <표 3>, <표 4>, <표 5>, <표 6>과 같다.

<표 3> 계산모델 시뮬레이터 분석

	계산모델 시뮬레이터 [8] [9]
원시 표현/연산자	• 적정 계산모델을 선택해 논리적 단위의 특정 연산을 수행하는 컴퓨팅에이전트를 정의할 수 있음 • 논리적 단위의 연산들을 보다 큰 계산에 활용할 수 있도록 원시연산 형태로 설정할 수 없음 • 모든 표현을 패턴구성의 규칙과 매개체의 요소로 인식하도록 유도하기 어려움
복합 표현/연산자	• 컴퓨팅 에이전트로서 원시연산들을 나열·조합해 상위 연산을 정의·활용할 수 없음 • 미리 정해진 형식의 표현(패턴구성 규칙과 매개체)만 지원함
외부환경과의 상호작용	• 외부장치와 적정 형태의 표현을 주고받을 수 있게 지원함으로써 계산과 외부세계 간의 상호작용을 적절히 설정·관찰·조작할 수 없음 • 컴퓨팅 에이전트로서 사용자의 역할을 설정할 수 없음 • 표현 매개체에 대해 초기값을 설정하거나 미리 정해진 형태의 관찰만 허용됨 • 이벤트 동기화 및 처리를 위한 계산은 설정할 수 없음
통제 및 분석	• 수행 속도 제어를 위한 이벤트 처리만 지원됨 • 매개체 참조 기능은 없으며, 특정 매개체만 한정된 방식으로 관찰 가능함 • 복잡도 관점에서 계산 과정을 관찰·분석할 수 있게 지원하지 않음

<표 4> 프로그래밍 언어 분석

	프로그래밍 언어	
	C	Java
원시 표현/연산자	• char, short, int, long, float, double 등 기본적인 데이터 표현을 위한 타입과, 데이터의 위치(주소)를 표현하기	• boolean, char, short, int, long, float, double 등 기본적인 데이터 표현을 위한 타입을 제공해 다양한 원시

	위한 포인터 타입을 제공해 다양한 원시 표현을 지원함 • 사칙연산자(+, -, *, /)를 비롯해 산술/관계/논리/치환/비트 연산자, 주소 연산자, 간접참조 연산자 등 다양한 원시 연산자를 제공함 • 주기억장치를 매개체로 다양한 타입(패턴구성 규칙)의 표현을 다룰 수 있게 지원함 • 라이브러리 수준에서 제공되는 파일(*file*) 개념을 이용해, 외부장치를 매개체로 바이트 순차 형식의 데이터 표현을 다룰 수 있음	표현을 지원함 • 사칙연산자(+, -, *, /)를 비롯해 산술/관계/논리/치환 연산자 등 다양한 원시 연산자를 제공함 • 자바가상기계(*JVM*)의 동적메모리(*dynamic memory*)를 매개체로 다양한 타입(패턴구성 규칙)의 표현을 다룰 수 있게 지원함 • 라이브러리 수준에서 제공되는 파일(*file*) 관련 클래스들을 이용해, 외부장치를 매개체로 바이트 순차 형식의 데이터 표현을 다룰 수 있음
복합 표현/연산자	• 구조체와 배열, 멤버지정 연산자, 배열첨자 연산자 등을 이용해 복합적인 데이터를 표현하고 다룰 수 있음 • 식(*expression*)을 이용해 일련의 원시 연산자를 특정 순서로 적용되도록 표현할 수 있음 • 식문, 선택문, 반복문, 분기문, 블록문, 레이블문 등 다양한 문장들을 나열·조합해 식들의 수행절차를 표현할 수 있음 • 다수 연산이 합성된 복합 연산자로서 함수(*function*) 개념을 제공해, 함수 정의(*function definition*) 및 함수 호출(*function call*)을 통해 복합 연산자를 정의·활용할 수 있음	• 클래스(*class*) 정의 및 인스턴스(*instance*) 생성, 배열 클래스, 참조(*reference*) 타입, 객체멤버참조 연산자 등을 이용해 데이터와 연산의 복합체를 표현하고 다룰 수 있음 • 식(*expression*)을 이용해 일련의 원시 연산자를 특정 순서로 적용되도록 표현할 수 있음 • 식문, 선택문, 반복문, 분기문, 블록문, 레이블문 등 다양한 문장들을 나열·조합해 식들의 수행절차를 표현할 수 있음 • 다수 연산이 합성된 복합 연산자로서 메소드(*method*) 개념을 제공해, 특정 클래스의 객체에 대한 메소드정의(*method definition*) 및 메소드호출(*method call*)을 통해 복합 연산자를 정의·활용할 수 있음
외부환경과의 상호작용	• 파일에 대한 읽기, 쓰기 등을 통해 외부 환경과의 상호작용을 설정·조작할 수 있음 • 주기억장치나 파일 상의 표현을 관찰하려면 별도의 도구를 이용하거나 추가적인 코딩 작업이 필요함 • 컴퓨팅 에이전트로서의 사용자 역할을 가정하고 있지 않음 • 이벤트 처리 및 동기화와 관련된 시스템 호출 라이브러리 등을 이용해 이벤트 처리 및 동기	• 파일에 대한 읽기, 쓰기 등을 통해 외부 환경과의 상호작용을 설정·조작할 수 있음 • *JVM*의 동적메모리나 파일 상의 표현을 관찰하려면 별도의 도구를 이용하거나 추가적인 코딩 작업이 필요함 • 컴퓨팅 에이전트로서의 사용자 역할을 가정하고 있지 않음 • GUI 클래스와 이벤트 처리, 다중스레딩(*multithreading*) 등을 이용해 이벤트 처리 및 동기화

	화를 위한 계산은 설정할 수 있지만, 이해하고 구현하기 어려움	를 위한 계산은 설정할 수 있지만, 상당히 높은 수준의 설계와 코딩이 필요함 • 객체지향 패러다임을 지원해 외부 세계의 다양한 사물들이 상호작용하는 상황을 보다 효과적으로 모델링 · 구현할 수 있지만 상당히 높은 수준의 설계와 코딩이 필요함
통제 및 분석	• 포인터 등을 이용해 주기억장치 상의 매개체에 대한 관찰 · 조작 가능함 • 파일 관련 라이브러리를 이용해 외부장치 상의 매개체들에 대한 관찰 · 조작 가능함 • 계산의 수행 속도를 제어하려면 디버거와 같은 별도의 도구를 이용하거나 추가적인 코딩 작업이 필요함 • 복잡도 관점에서 계산 과정을 관찰 · 분석하려면 별도의 도구 혹은 추가 코딩이 필요함	• 반영(*reflection*) 개념을 지원하는 클래스 객체들을 이용해 *JVM*의 동적메모리 상에 설정되는 매개체들에 대한 관찰 · 조작 가능함 • 파일 관련 라이브러리 클래스들을 이용해 외부장치 상의 매개체들에 대한 관찰 · 조작 가능함 • 계산의 수행 속도를 제어하려면 디버거와 같은 별도의 도구를 이용하거나 추가적인 코딩 작업이 필요함 • 복잡도 관점에서 계산 과정을 관찰 · 분석하려면 별도의 도구 혹은 추가 코딩이 필요함

〈표 5〉 교육용 프로그래밍 언어 분석

	교 육 용 프 로 그 래 밍 언 어	
	로고(Logo)	**스크래치(Scratch)**
원시 표현/연산자	• 원시 데이터 표현으로서 0개 이상의 문자들의 나열을 다룰 수 있는 **word** 타입만 지원함 • 사칙연산자(**+**, **-**, *****, **/**)를 비롯해 산술/관계/논리/치환/비트 연산자 등 다양한 원시 연산자(프로시저)를 제공함 • 로고 시스템 내의 저장 공간을 매개체로 문자들의 나열로서 '단어(*word*)' 표현만을 다룰 수 있게 지원함 • 파일(*file*)을 대상으로 한 원시 프로시저를 이용해, 외부장치를 매개체로 문자들의 순차 형식의 데이터 표현을 다룰 수 있음	• 숫자와 문자열 표현을 다룰 수 있는 변수를 지원함 • 사칙 연산과 논리 연산, 미디어 대상 연산 등의 원시 연산자를 제공함 • 변수, 스프라이트, 외부 장치 등의 매개체를 이용해 여러 유형의 표현을 다룰 수 있도록 지원함 • 객체 스프라이트(*sprite*)의 모습과 소리, 특정 이벤트에 연동된 스크립트(*script* : 이벤트에 대한 반응으로 수행될 멀티미디어 효과의 발생 절차를 명시한 프로그램)를 스크래치 블록(*block*)들

	• 터틀(*turtle*) 그래픽 화면 상의 '터틀'의 속성과 이동을 제어할 수 있는 원시 프로시저를 이용해 기본적인 그래픽 작업이 가능함	을 쌓아 정의한 후, 해당 이벤트를 발생시켜 스테이지(*stage*) 영역 등을 통해 애니메이션, 게임 등이 수행되도록 지원함
복합 표현/연산자	• 단어들의 선형적 나열 형식의 리스트(*list*)/배열 타입과, 그들에 대한 원시 프로시저를 이용해 일부 복합적인 데이터를 표현하고 다룰 수 있지만, 비선형 패턴을 표현하기 어려움 • 식(*expression*) 혹은 프로시저 호출의 중첩을 이용해 일련의 원시 연산자를 특정 순서로 적용되도록 표현할 수 있음 • 제어명령들을 이용해 지시어들의 수행절차를 표현할 수 있음 • 다수 연산이 합성된 복합 연산자로서 프로시저(*procedure*) 개념을 제공해, 프로시저 정의 및 호출을 통해 복합 연산자를 정의·활용할 수 있음 • 절차적 표현인 프로시저를 데이터 표현(지시어들의 텍스트 표현)을 이용해 동적으로 정의할 수 있고, 프로시저를 구성하는 지시어들의 나열을 텍스트 형태의 데이터 표현으로 다룰 수 있음	• 복합적인 데이터 표현으로서 일련의 숫자나 문자열들을 다룰 수 있는 리스트 변수와, 리스트에 대한 연산들을 지원하지만, 비선형 패턴을 표현하기 어려움 • 연산을 중첩시켜 일련의 연산자들이 특정 순서로 적용되도록 표현할 수 있음 • 팔레트 상에 있는 순차/반복/선택 관련 제어 블록들을 스크립트 영역에 갖다 놓고 다른 하위 블록들을 적정 순서로 쌓아 블록들의 수행절차(스크립트)를 표현할 수 있음 • 스크립트에 대한 파라미터 전달이나 리턴값 없음 • 새롭게 정의한 스크립트를 재사용 가능한 블록으로 설정할 수 없음 • 재귀적 호출 기능을 지원하지 않아 수행 구조의 제약이 있음 • 스프라이트를 객체로서 정의할 수 있게 지원하지만 그들의 클래스나 상속 개념은 지원하지 않음
외부환경과의 상호작용	• 파일에 대한 읽기, 쓰기 등을 통해 외부 환경과의 상호작용을 설정·조작할 수 있음 • 로고 시스템 내의 저장 공간에 설정된 표현들에 접근할 수 있는 원시 프로시저를 제공하므로, 인터프리터 방식의 작업 도중 수시로 원하는 표현을 관찰할 수 있음 • 일시정지 및 재개 기능을 제공하는 원시 프로시저를 이용해 컴퓨팅 에이전트로서의 사용자 역할이 가능함 • 이벤트 처리 및 동기화 기능을 지원하지 않음	• 파일에 대한 입출력 기능을 지원하지 않음 • 키를 누르거나 스프라이트를 클릭하는 등 사용자가 발생시킨 이벤트 혹은 프로그램의 다른 부분에서 발생된 이벤트의 처리가 가능함 • 병행처리를 지원하고, 메시지 전달을 통해 협력 작업이나 동기화가 가능함 • 키보드 입력, 마우스 이동, 주변의 소음 등에 대한 입력 처리를 지원하고, 그래픽 처리, 애니메이션, 음악, 사운드 등 다양한 멀티미디어 효과를 생성해 줌 • 센서보드를 통해 외부장치와의 상호작용을 지원함

통제 및 분석	• 변수 설정을 이용해 로고 시스템 내의 매개체에 대한 관찰·조작 가능함 • 파일 관련 라이브러리를 이용해 외부장치 상의 매개체 관찰·조작 가능함 • 인터프리터와의 대화식 작업을 통해 계산의 수행 속도를 제어할 수 있음 • 복잡도 관점에서 계산 과정을 관찰·분석하려면 추가 코딩이 필요함	• 변수 설정 등을 이용해 스크래치 시스템 내의 매개체에 대한 관찰·조작 가능함 • 이벤트 처리 기능을 이용해 계산의 수행 속도를 제어할 수 있음 • 복잡도 관점에서 계산 과정을 관찰·분석하려면 추가 코딩이 필요함

〈표 6〉 로봇 제어용 프로그램 개발 도구 분석

	로봇 제어용 프로그램 개발도구
원시 표현/연산자	• 일반적으로 그래픽 기반 프로그래밍 도구일 경우 원시 표현/연산자에 대한 다양성이 적음 • 데이터 표현과 절차적 표현이 로봇 내부의 적정 매개체에 담겨져 관련 계산이 생성됨 • 로봇에 장착된 장치(모터, 센서 등)들을 제어할 수 있도록 관련된 데이터 표현과 연산자를 제공함
복합 표현/연산자	• 일반적으로 그래픽 기반 프로그래밍 도구일 경우 복합적인 데이터 표현에 대한 지원이 크게 제한됨 • 일반적으로 그래픽 기반 프로그래밍 도구일 경우 원시 연산자들을 나열·조합해 복합적인 연산을 정의할 수 있지만, 이렇게 정의된 상위 연산자의 활용은 크게 제한됨
외부환경과의 상호작용	• 로봇에 장착된 다양한 장치를 제어함으로써 물리적 주변 환경과의 상호작용을 적절히 설정·관찰·조작할 수 있음 • 로봇과 상호작용하는 컴퓨팅 에이전트로서 사용자의 역할을 설정할 수 있음 • 센서 등을 매개로 이벤트 처리 및 동기화를 위한 계산을 설정·관찰·조작할 수 있음
통제 및 분석	• 수행 속도를 제어하려면 해당 기능을 수행하는 코드를 로봇 제어 프로그램에 포함시켜 코딩해야 함 • 로봇 내부의 매개체를 일부 관찰할 수 있음 • 로봇의 성능이나 용량의 제한, 내부 상태에 대한 관찰의 어려움 등으로 복잡도 관점에서 계산 과정을 관찰·분석하기 어려움

Ⅳ. 결론

컴퓨터과학 교육에 있어 계산 원리 학습은 매우 중요한 의미를 갖는다. 이는 계산(computation)이 컴퓨터과학의 중심 연구 대상이고 기본 원리이기 때문이다. 따라서 컴퓨터과학에 대해 올바로 이해하고 본질적 측면에서 접근하려면 계산에 대한 깊은 이해가 바탕이 되어야 한다. 더욱이 계산 원리에 기초한 사고 능력은 지식기반 사회 구성원들의 일상적/학문적 제반 활동의 기본 역량으로 요구되고 있어, 계산 원리 학습의 중요성은 더욱 커질 것이다. 문제는 우리가 지금까지 초중등 교육과 대학 교육에서 '계산 원리'에 대한 교수·학습을 제대로 하고 있지 않다는 것이다. 초중등 컴퓨터 교과목에서는 물론, 컴퓨터과학 관련 전공 교육에서조차 '계산 원리' 자체에 주안점을 둔 교육이 이루어지고 있지 않다는 것은 매우 심각한 것이다. 개론 교과목에서는 컴퓨터의 구성이나 컴퓨터과학의 세부 분야 개괄을 소개하는데 그치고 있고, 프로그래밍 교과목에서도 프로그래밍 언어 표현의 습득과 기본 프로그램 작성 훈련에 주안점을 두고 있을 뿐이다. 알고리즘 교육도 대부분 문제해결 방법이나 패러다임, 복잡도 분석 등에 초점을 맞추고 있고, 다른 교과들 역시 해당 영역의 이론이나 기술을 가르치는데 대부분의 시간을 할애하고 있다. 초중등 교육 역시 대학 교육과 동일 성격의 내용을 조금 쉽게 개괄적으로 제시하고 있는 게 컴퓨터과학 교육의 현실이다.

모든 변화의 심연에는 그와 관련된 영역의 본질이 자리하고 있고 기본 원리가 작용하고 있다. 아무리 급격한 표면의 변화도 심연의 거대한 흐름 위에서 그와 연계되어 일어나는 것이다. 그렇기에 우리 삶의 전 영역에서 일어나고 있는 급격한 변화와 발전의 저변에 컴퓨터과학이 작용하고 있고 컴퓨터과학의 밑바닥에 계산 원리가 있음을 생각할 때, 계산 원리를 인식하지 못하고 놓치게 되면 우리 삶의 모든 영역에서 일어나고 있는 변화를 선도하지도 적응하지도 못하게 될 수 있음을 정확히 인식하고 그 대책을 마련해야 한다. 이와 관련해 '계산 원리 학습'과 관련된 교육적·제도적 방안도 시급히 강구해야 하겠지만, '계산 원리 학습'을 보다 효과적으로 지원하는 교수학습 방법의 도출 또한 병행·연구되어

야할 매우 중요한 일이다.

본 논문에서는 먼저 컴퓨터과학 관점에서의 계산 원리 학습을 도와주는 학습 도구의 필요성을 논하고 계산 원리 학습의 내용 요소와 방향을 도출한 후, 그와 같은 활동을 효과적으로 지원하기 위해 계산 원리 학습도구가 갖추어야 할 기능 요소들을 고찰·제안하고 있다. 계산모델 시뮬레이터나 프로그래밍 언어, 로봇 등 계산 원리 교육에 활용 가능한 도구들이 있지만, 이들은 근본적으로 계산 원리에 초점을 맞추고 있지 않아 계산 원리 학습을 효과적으로 지원하지 못한다. 계산모델 시뮬레이터들은 주요 계산모델들에 대한 실질적 조작 경험을 제공함으로써 계산모델의 작동 방식과 관련 이론을 보다 구체적으로 습득할 수 있게 도와줄 뿐이다. 프로그래밍 언어나 교육용 프로그래밍 언어도 문제 해결 프로그램을 보다 잘 이해하고 작성할 수 있게 도와주지만, 프로그래밍에 내재된 계산 관련 요소들을 계산 원리 관점에서 다루어 보면서 학습할 수 있게 지원하지는 않는다. 로봇과 같이 계산과 실세계의 상호작용을 관찰·설정할 수 있는 도구들이 있지만, 로봇 교육용 도구들은 로봇제어 관련 활동을 보다 쉽게 수행할 수 있도록 도와주는 데에 주안점을 두고 있을 뿐이다.

본 논문에서는 컴퓨터과학이 세계를 어떻게 바라보고 문제의 해답에 어떻게 접근하는지를 나타내는 핵심 틀이 '컴퓨팅 패러다임' 임을 인식하고, 컴퓨팅 패러다임에 따른 활동에 계산 원리의 어떤 요소들이 어떻게 적용되는지를 분석해 계산 원리 학습도구가 갖추어야 할 기능적 요소들을 도출하였다. 그리고 계산 원리 학습도구의 개괄적 시스템 구성을 제시하고 있다. 향후 본 논문에서 제안한 계산 원리 학습도구를 보다 세부적으로 설계해 개발할 계획이다. 또한 개발한 계산 원리 학습도구를 실제 학습에 다각도로 적용·분석해 보면서 지속적으로 보완·발전시켜 나갈 것이다.

참고문헌

[1] Peter J. Denning, “Computing is a Natural Science”, Communications of the ACM, Vol. 50, No. 7, July 2007, pp. 13–18.

[2] Peter J. Denning, et al., “Computing as a Discipline”, Communications of the ACM, Vol. 32, No. 1, January 1989, pp. 9–23.

[3] Peter J. Denning, “Is Computer Science Science?”, Communications of the ACM, Vol. 48, No. 4, April 2005, pp. 27–31.

[4] Great Principles of Computing Web site; http://cs.gmu.edu/cne/pjd/GP/

[5] 이영준, 이응경, “정보교육의 본질과 전망, 컴퓨터교육학회 논문지”, Vol. 11, No. 3, 2008, pp. 1-11.

[6] Jeannette M. Wing, “Computational Thinking”, Communications of the ACM, Vol. 49, No. 3, March 2006, pp. 33–35.

[7] Peter J. Denning and Peter A. Freeman, “The Profession of IT Computing’s Paradigm”, Communications of the ACM, Vol. 52, No. 12, December 2009, pp. 28-30.

[8] David G. Hannay, “Interactive Tools for Computation Theory”, ACM SIGCSE Bulletin, Vol. 34, No. 4, December 2002, pp. 68-70.

[9] Mohamed Hamada, “An Integrated Virtual Environment for Active and Collaborative e-Learning in Theory of Computation“, IEEE Transactions on Learning Technologies, Vol. 1, No. 2, April-June 2008, pp. 117-130.

[10] Al Kelly, Ira Pohl, “A Book On C”, Boston: Addison-Wesley Professional.

[11] 남궁성, “Java의 정석”, 경기: 도우출판.

[12] 홍재운, 이수정, “메타인지 수준에 따른 EPL 프로그래밍 학습이 논리적 사고에 미치는 영향”, 정보과학회논문지, Vol. 36, No. 6, 2009, pp. 498-507.

[13] http://info.scratch.mit.edu/ko/About_Scratch/

[14] 조성환, 송정범, 김성식, 이경화, “CPS에 기반한 스크래치 EPL이 문제해결력

과 프로그래밍 태도에 미치는 효과", 한국정보교육학회논문지, Vol. 12, No. 1, 2008, pp. 77-87.

[15] David J. Perdue, "THE UNOFFICIAL LEGO® MINDSTORMS® NXT INVENTOR'S GUIDE", San Francisco: No Starch Press Inc.

[16] 최병윤, 이용재, 조영완, 신경욱, 손승일, 이문기, "NXT-G와 Java 언어를 이용한 레고 마인드스톰 NXT 프로그래밍", 서울: 그린.

[17] 유영대, 장선아, 양재군, 박지현, 배재학, "교육용 소형 로봇을 이용한 군집 로봇 시스템 구현", 한국컴퓨터종합학술대회 논문집, Vol. 35, No. 1, 2008, pp. 387-390.

<Abstract>

A Study on Required Elements of Computation Principle Learning Tools for Computer Science Education

Kim, Hyung-Chul

Graduate School of Education in Jeju National University

(Majoring in Computer Education)

Supervised by Professor Kim, Cheol Min

In the information age, we are faced with various kinds of problems in social and natural environment. How we solve these problems is an important determination factor of our competitiveness. So we should not overlook the fact that the scale and complexity of problems we have to solve increases day by day. This can be problematic, since human's brains are not developed to solve such complicated matters well. Therefore each of us has to focus on improving his/her knowledge and abilities in solving such problems with more efficiency.

This paper asks each of us to pay attention to the excellent tools computer science has on this matter. Computer scientists use *abstraction* to tackle the complexity of a problem and *automation* to tackle the scale of a problem. Those are powerful weapons for developing strategies to solve large complex

problems. Computing is the automation of our abstraction within the framework of the '*information process*' and '*computation*'. Computing chooses abstractions for modeling the relevant aspects of a problem to make it tractable. Computing uses abstraction and decomposition when attacking a large complex task. This decomposition process is repeated until the phenomena surrounding the problem is modelled with interactions among the proper abstractions each of which is easy enough to be represented as computational steps and algorithm.

Since computing pursues computational doing through implementing computational systems, all the abstractions and their relationships should be mechanized. Automation with mechanization is possible due to the precise and exacting notations and models based on *computation*. So, the computation, which means a sequence of representations, is the core principle of computer science.

The problem is that although the computation principle plays a key role in solving difficult problems, in reality, there is little school education on this matter. Further more, there aren't any tools just for teaching and learning the computation principle. This paper first surveys the computation principle and put emphasis on the importance and necessity of learning it. Also, this paper analyses and proposes the required functional elements of computation principle learning tools for increasing skills in solving large complex problems. The functional elements are expected to be a good reference when someone develops related tools for computation principle learning.