스마트 코드 생성을 위한 프로그램 분석 시스템 연구
The LET Project: program analysis for smalL/safE/smarT code

이 광근
email: kwang@cs.kaist.ac.kr phone: 042.869.3536

전산학과
한국과학기술원(KAIST)

1 문제제기

지구상의 모든 컴퓨터가 초고속 넷트웍으로 연결되면 지금까지와는 판이한 컴퓨팅 환경이 만들어진다. 지금까지 실험실에서 소규모로 구성된 분산 병렬 컴퓨터가, 지구위에서 거대한 스케일과 부정형적인 토폴로지로 조성되는 것이다. 이러한 컴퓨팅 환경에서는 프로그램 실행중에 필요한 코드부품이나 데이타들이 한 컴퓨터에 모여있지 않고 전 세계의 컴퓨터에 흩어져 있게 된다. 프로그램 부품들은 그 부품에 특화된 기술을 갖춘 팀에의해 최상의 품질로 넷트웍을 통해 지체없이 제공될 수 있기 때문이다. 모든 프로그램은 지구위를 광속으로 돌아다니면서 필요한 곳에서 필요한 자원(코드 부품, 데이타, CPU, 메모리등)을 적시에 동원하여 작업을 끝마치는 것이 일상적이 되는 것이다.

이러한 컴퓨팅 환경에 최적으로 적응하는 코드를 만들기 위해서는 지금까지 생각지 못한 많은 문제들이 해결되야한다. 물론, 현재의 컴퓨팅환경(desk-top computing) 에서도 실행코드의 부품과 데이타를 적시에 동원해서(디스크로부터 불러들여서) 프로그램을 실행하는 것은 가능해왔던 일이다. 그러나 미래의 전 지구적 컴퓨팅 환경에서 문제가 되는 것은, 그러한 일들이 전세계의 모든 컴퓨터를 묶은 초고속 넷트웍을 타고 엄청난 양과 빈도로 발생하게 된다는 것이다. 전세계 모든 코드가 광속으로 지구위를 움직이면서 사용자에게 최적의 성능을 제공하려고 경쟁하는 환경에서 그러한 일들이 일어나는 것이다. 이러한 환경에서 최적으로 수행되는 코드는 다음의 문제를 해결하고 있어야 한다.

- 코드의 안전성 보장: 넷트웍을 통해 외부에서 제공될 불특정 다수의 코드가 현재 진행중인 프로그램의 안전을 해치지 않는다는 것을 미리 검증할 수 있어야 한다. 코드를 제공하는 측에서 안전성을 검증해줘야 할 뿐 아니라, 받아든 측이 그 검증내용을 다시 확인할 수 있도록 해야 하는데, 그 안전성의 조건에는 진행중인 프로그램을 파행적으로 중단시키지 말아야한다는 기본적인 조건뿐 아니라, 자원(CPU, 메모리)을 어느 이상은 사용하지 말아야 하는 등의 광범위한 제약조건이 모두 포함된다.

- 코드의 최소화: 코드의 크기가 최대한 작아서 넷트웍상에서의 물류비용이 최소로 유지되야 한다. 같은 일을 하는 다른 경쟁 코드보다 빨리 사용자에게 전달되야 하기 때문이다.

- 코드의 자기 최적화: 이렇게 해서 사용자에게 빨리 접근한 코드는 사용자의 사용 패턴에 자신의 성능을 최적화 시킬 수 있는 시간이 더 있을 것이고, 이렇게 실행 코드 스스로가 사용패턴을 인식해서 자신을 최적화할 수 있다면, 작아서 빨리 움직인 장점은 더욱 배가될 것이다.

2 세계적 연구동향

글로벌 프로그래밍 시스템에 대한 연구는 현재의 인터넷 프로그래밍 시스템에 대한 연구들[DFW96, Rou96]이나 차세대 넷트웍 아키텍춰[TW96, SFG$^+$96]에 대한 연구로 나누어질 수 있다. 인터넷 프로그래밍 환경연구는 Java를 중심으로 현재의 인터넷에서 효과적으로 사용될 소프트웨어 시스템에 관한 연구이고, 차세대 넷트웍 아키텍춰 연구는 현재의 인터넷의 한계를 넘어서서, 본 제안서에서 상정하는, 전 지구적 넷트웍 컴퓨팅 환경을 구축하고자 하는 방향의 연구이다.

대표적인 연구 그룹은 아래와 같다.

- Proof Carrying Code [Nec97], Carnegie Mellon University: 코드를 제공하는 측에서 코드의 안전성 증명을 코드와 함께 보내고, 사용하는 측에서 그 증명이 따라온 코드의 증명인지를 확인한 후 받아들이는 시나리오다. 아직은 소규모 코드부품에 한해 일차논리식(first-order logic)으로 표현되는 성질의 증명을 다루면서 그 가능성을 연구하고 있다.

- SwitchWare [SFG$^+$96], University of Pennsylvania: 넷트웍 스위칭 노드에 프로그램을 실어서 스위칭 노드의 컴퓨팅 능력을 향상시키려는 연구로서, 이러한 시스템을 분석하는 수학적인 모델을 고안하려는 연구와 함께 진행중에있다.

- Active Network [TW96], MIT: 스위칭 노드에 전달되는 팩킷(packet)이 단순한 목적지와 데이타만으로 구성된 것이 아니고 프로그램도 가지고 있어서(capsule), 캡슐이 각 스위칭 노드를 통과할때 따라온 코드가 수행되도록 하는 넷트웍 아키텍춰의 구현을 목표로하고 있다.

- Safe Internet Programming [DFW96], Princeton University: 현재 인터넷 프로그래밍 언어로 부상중인 Java를 좀더 안전하게 수행시킬 수 있도록 하기 위한 시스템 기술을 연구하고 있다.

- Project Cristal [Pro, Rou96], INRIA: 불란서에서는 ML[MTHM97]이라는 차세대 언어를 기반으로 독자적인 인터넷 프로그래밍 시스템을 연구하고 있다.

- Mobile Agent/Softbot/Intellignet Sensor [Lan94], MIT,Stanford,CMU: 넷트웍을 움직이며 지정된 작업을 능동적으로 수행해 가는 코드에 대한 연구로 인공지능 분야에서 개발된 기술을 바탕으로 한다.

본 연구계획과 가장 가까운 연구를 진행하고 있고 있는 그룹은 CMU 그룹인데, 증명(theorem proving) 시스템과 타입 이론을 이용해서 코드의 안전성 검증 문제를 다루고 있다. 이 그룹은 아직 프로그램 분석(static analysis) 분야에서 축적한 이론들을 이용하고 있지 못하고 있다.

Java나 Agent관련 연구그룹들은 글로벌 컴퓨팅 모델의 엄밀한 분석에 기초하고 있지 않을 뿐더러, 미래의 넷트웍은 지금의 인터넷과는 많은 차이(각 스위칭 노드가 프로그램 가능한 일반 프로세서의 수준으로 향상되는 등)가 있을 것이기 때문에, 이러한 연구들이 차세대 넷트웍 프로그래밍 환경에 쉽게 적용되는 기반기술을 만들어 낼지는 의문이다.

국내의 프로그래밍 시스템 연구들은 Java 관련 연구들이 활발하게 진행되고 있는데, 이 연구들도 프로그래밍 언어 이론이나 엄밀한 컴퓨팅 모델에 기초하고 있지 않기 때문에, 미래의 효과적인 글로벌 컴퓨팅을 가능하게 하는 핵심 기반기술로 부각되기는 힘들것으로 보인다.

그리고, 본 연구가 진행중에 항상 그 발전과정을 염두에 두어야 하는 분야로, 프로그래밍 언어 모델에 대한 연구가 있다. 미래의 글로벌 컴퓨팅 환경의 계산 모델을 제안하고 그 모델에 기초한 다양한 프로그래밍 언어 모델들을 고안하는 연구로써, Milner의 Pi-calculus[MPW89a, MPW89b], Fournet의 Join-calculus[FG96, FGL$^+$96, FLMR96], Cardelli의 Ambient-calculus[CG98], Abadi의 Spi-calculus[AG97a, AG97b]등이 있다. 미래에는, 이러한 기초적인 모델들에 기반한

새로운 글로벌 프로그래밍 언어가 만들어지거나(Obliq[Car94, BN96], PLAN[HKM+98] 등), 기존의 언어에 적절한 확장이[TLP+93] 이루어 질 것으로 예상된다.

우리는 이러한 글로벌 프로그래밍 모델의 핵심 구조들에 대한 이론 연구를 항상 고려하게 될 것이다. 첫째 이유는, 본 연구가 이러한 모델에 기초한 미래의 언어들의 컴파일러 시스템에 쉽게 적용되는 핵심기술을 만들어내야 하기 때문이고, 둘째 이유는, 그러한 엄밀한 모델에 기초한 연구는 우리가 대상으로 하는 프로그래밍 언어의 엄밀한 의미구조를 제공하는 기초가 되어 우리 연구를 엄격히 검증하고 그 한계를 규명하는 데에 필수이기 때문이다.

2.1 본 연구의 차별성

우리는 프로그래밍 언어 이론에 기초한 엄밀한 프로그램 분석(static analysis) 이론을 이용해서 제안한 문제들의 해결책을 연구할 것이다. 물론, 지금까지의 연구들이 수학적으로 잘 정리된 이론에 기초하지 않고도 성공적인 연구결과를 가져올 수 있었고, 소프트웨어 이론들도 컴퓨팅 현실을 충분히 분석할 수 있는 모델을 제공해 주지 못한 면이 있다. 그러나 전 지구적 넷트웍 컴퓨팅 환경은 그 규모와 복잡성이 지금까지와는 비교될 수 없기 때문에 수학적으로 요약된 모델이 꼭 필요하고, 최근의 프로그래밍 언어 이론에 대한 연구성과들은 이러한 환경에서 실행되는 프로그램을 분석하는데 유용한 이론이 될 수 있을 만큼 비약적으로 성장하고 있다. 이러한 이론에 기초하지 않고서는 고안된 해결방안의 한계를 파악할 수 도 없고, 흔히 있을 수 있는 연구결과의 오류를 엄격히 검증할 수도 없다. 소홀한 기반위에서 고안된 연구결과는 어느순간 무너져 버리는 다리와 같이 위태로울 수 밖에 없고, 영향력있는 연구결과로 발전하기에는 역부족이 될 것이다.

전세계적으로, 프로그래밍 시스템을 연구하는 그룹은 대상으로 하는 프로그래밍 인구의 관성(programming sociology)을 따라가기 때문에(Java 주변의 경우와 같이) 프로그래밍 언어의 신 이론을 받아들이는 것이 느리고, 프로그래밍 언어 이론에 몰두하는 그룹은 프로그래밍 시스템 기술에의 응용에 무관심하다. 이 틈새를 본 연구가 공략할 것이다. 본인이 이 두 분야의 대표적인 연구그룹(Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign 과 Software Principles Research Department, Bell Laboratories) 에서 연구한 잇점을 모두 동원할 것이며, 그 중심에는 본인이 지금까지 성공적으로 수행해온, 의미구조에 입각한 프로그램 분석 (semantics-based static analysis) 연구가 자리할 것이다.

3 연구가설

본 과제에서 제안한 문제들의 해결에는, 프로그래밍 언어의 엄밀한 의미구조에 기초한 프로그램 분석(semantics-based static analysis) 기술이 핵심이 된다는 것이 우리의 입장이다. 프로그램 분석(static analysis)을 통해서 프로그램이 실행중에 가질 수 있는 성질을 미리 예측할 수 있는데, 이 예측을 통해서

- 코드가 주어진 안전성의 조건을 실행중에 항상 만족하는지를 검증하고
- 같은 실행 결과를 가지는 가장 밀도있는 코드로 변형하며
- 실행중에만 가능한 코드의 최적화 과정을 미리 함수로 요약해서 코드에 딸려 보내는 것이

가능하게 될 수 있기 때문이다.

본인은, 프로그래밍 언어의 정형적인 의미구조(mathematical semantics)로 부터 프로그램의 성질을 예측하는 연구를 수행에 왔는데, 이 연구 성과들을 계기로 해서 위의 세 가지 문제들이 해결될 수 있을 것으로 믿게 되었다:

- 코드의 안전성: 프로그램이 실행중에 파행적으로 중단되지 않는다는 기본적인 안전성 (type safety), 프로그램이 실행중에 자원을 얼마 이하만 사용해야 한다는 실행비용의 안전성 (resource safety), 실행중에 항상 약속한 원칙(예를들어, 메모리에 표현되는 리스트구조는 항상 특정 메모리 영역에서만 구현된다는 등)을 준수해야 한다는 안전성 (invariant safety) 등을 검증하는 것이 모두 프로그램의 분석을 통해서 가능해진다.

  이때, 고안된 분석기가 임의의 프로그램의 실행중에 발생하는 모든 경우를 예측한다는 것이 증명되야 하고, 이 증명은 대상 프로그래밍 언어의 의미구조 식으로 부터만이 엄밀하게 진행될 수 있다.

  한편, 프로그램을 받아든 측에서 안전성 검증이 제대로 된 것인지를 확인할 수 있어야 하는데, 이 방법도 프로그램 분석이론을 이용해서 고안될 수 있다. 모든 프로그램 분석의 계산 과정은 반복법(fixpoint iteration)을 사용해서 구현 될 수 있는데, 이 반복 과정을 거쳐서 최종적으로 변하지 않는(repeat until no change) 분석 값을 찾는 것이 분석의 과정이 된다. 이때, 반복적으로 계산되는 분석 값들은 항상 단조증가하게 되는데, 이러한 분석 계산의 단조증가(monotonicity) 성질을 이용할 수 있다. 코드를 제공하는 측에서는 코드의 각 식들에 대해 분석된 내용을 코드에 딸려 보내고, 코드를 받아든 측에서는 이 내용에서 부터 분석의 반복 계산을 수행해 본다. 이때 계산의 첫 반복 후에 그 따라온 내용이 변하지 않는 것이었다면, 그 내용들은 올바른 분석 내용이라는 것을 확신하게 된다.

- 코드의 최소화: 주어진 프로그램을 같은 일을 하면서 더욱 밀도 있는 프로그램으로 변형해 주기 위해서는, 그 프로그램이 실행중에 무슨일을 벌이는 지를 분석해야 한다. 고안된 분석기는 프로그램이 실행중에 일으킬 수 있는 모든 메모리 반응을 빠짐없이 예측해서 동일한 메모리 반응을 이끌어내는 고 밀도 코드를 생성하게 된다.

  프로그램 분석을 통해서 코드에서 같은 값을 계산하는 중복된 식이나 실행중에 사용되지 않을 값을 계산하는 식들을 찾아낼 수 있다. 또, 코드의 물류비용을 줄이는 한 방법으로, 코드의 크기와 그 코드가 만들어내는 값의 크기를 비교해서 작은 것을 실어 보내는 아이디어가 있을 수 있는데, 이때 코드가 만들어내는 값의 크기를 프로그램 분석을 통해서 예측할 수 있다.

- 코드의 자기 최적화: 코드 스스로 입력패턴을 인식해서 자신을 최적화 시키기 위해서도 프로그램 분석이 유용하게 쓰일 수 있다. 프로그램을 최적화하기 위해서는 프로그램의 실행 내용을 미리 분석하고 있어야 하는데, 분석하는 도중에 부수적으로 파악될 수 있는 것이, 실행중에 어떤 정보를 알게되면 어떠한 성질을 가지게 되고 따라서 어떻게 최적화 될 수 있는지이다. 이러한 최적화 가능성을 함수꼴로 요약해서 코드에 딸려 보내는 것이 가능해 진다. 이렇게 따라간 최적화 함수는 실행중에 발생하는 입력이 도화선이 되어서, 같이 온 코드를 최적화 시킬 것이고 이렇게 최적화된 코드는 다음에 오는 비슷한 입력에 대해서 최적의 실행을 할 것이다.

4 연구내용및 추진체계

연구 추진은 두 가지 축으로 구성된다: 한 방향은 위의 세가지 연구 문제를 하나씩 해결해 가는 것이고, 다른 한 방향은 고안된 해결책들을 실제 프로그래밍 시스템을 구축하면서 실현해 보는 것이다.

이때 구축하는 프로그래밍 시스템은 ML[MTHM97]이라는 언어의 새로운 – 위의 문제들에 대한 답을 갖춘 코드를 생성해내는 – 컴파일러가 될 것이다. ML은 위에 제안한 문제들을 푸는데 필요한 언어 이론을 가장 충실히 갖춘 언어중의 하나이면서, 차세대 시스템 프로그래밍 언어로 부상하고 있기 때문이다. 본인은 지난 3년동안 ML의 엄밀한 의미구조에 기초해서

ML 프로그램의 안전성을 검증하는 기술(exception analysis)과 타입 유추 알고리즘에서 세계적으로 주목받는 연구성과를 이루어오고 있으며, 우리는 이 연장선상에서, 위에서 논의한 해결방안들을 ML의 컴파일러로 구현해서 그 실용성을 확인해 갈 것이다.

이렇게 해서 성공적으로 판별된 해결방안과, ML이 드러냈던 제한점들을 가지고 ML의 대안으로 글로벌 컴퓨팅 환경에 적합한 프로그래밍 언어 시스템을 구축하는 것도 가능할 것으로 예상된다. 이 단계까지 전세계 연구팀과의 교류:협력:경쟁의 비율이 4:3:3 정도가 될 것이다.

5 연구테마의 가치

- **과학기술적 가치**

  지금은 우리가 1950년대에 놓쳐버렸던 정보 기술의 주도권 대열에 들어설 수 있는 두번째 기회가 되고 있다. 지금은 곧잘 1950년대 - 디지탈 컴퓨터가 막 만들어지면서 그 기계를 사용할 프로그램의 개념이 막 무르익던 시기 - 에 비견된다. 당시에 컴퓨터 메모리에 저장된 프로그램을 실행하는 컴퓨팅 환경이 만들어 지면서, 프로그램이란 것을 효과적으로 기술하는 언어를 디자인하고 구현하는 문제가 대두되었다. 이 문제에 대한 오늘날과 같은 해답을 가능하게 했던 핵심 연구들(형식언어 (formal language or automata) 이론에 기초한 프로그램 문법의 검증기술(parsing theory), 타입 이론(type theory)에 기초한 프로그램의 실행 값 검증기술(strong typing, type inference), 컴파일러등의 시스템 구현 기술, 람다 계산법(Lambda Calculus)에 기초한 프로그래밍 언어 이론등)을 이룩한 그룹들이 오늘날 정보 기술의 주도권을 잡고 있게 되었다.

  오늘날 새롭게 출현하고 있는 컴퓨팅 환경은 1950년대와 똑 같은 문제를 부각시킨다: 새로운 환경에서 효과적으로 수행될 프로그램을 만드는 기술이 필요한 것이다. 본 연구에서는, 이러한 기술의 핵심이라고 여겨지는 세가지 문제 - 코드의 안전성 검증, 코드의 최소화, 코드의 자기 최적화 - 를 엄밀한 프로그램 분석 이론과 구체적인 시스템 구현을 통해서 해결하려고 하는 것이다. 새롭게 태동하는 컴퓨팅 환경에 최적인 프로그래밍 이론과 실제를 본 연구가 천착해 간다면, 다음 시대의 정보 기술의 주도권 대열에 당당하게 설 수 있을 것이다.

- **사회경제적 가치**

  전 세계를 하나의 시장으로 묶는 글로벌라이제이션의 물결은, 정보기술분야에서는 전지구적 넷트웍 컴퓨터의 모습으로 다가 오고있고, 이 구조는 마치 계곡에 흘러드는 물처럼 모든 것을 균일화하는 습성을 가지고 있다. 커뮤니케이션의 발달은 세계적으로 동일한 문제를 해결하려는 프로그램을 대량으로 요구할 것이고, 최고의 품질을 갖춘 소수정예의 코드나 데이타만이 그러한 다수의 프로그램들속으로 적시에 끼어들어갈 것이기 때문이다.

  따라서, 이 넷트웍위에 가장 빈번히 사용되는 프로그램이나 데이타를 올려놓을 수 있으면 우리는 번영하게 되지만, 자칫하면 우리앞에 모든 문제를 해결한 모습으로 외부로부터 강요되기 쉽다.

  이와같은 미래에는 전 지구위의 컴퓨터를 광속으로 이동하면서 주어진 일을 수행해가는 수없이 많고 다양한 코드 부품들이 산업의 쌀 구실을 할 것이고, 본 연구의 성공적인 결실은 최고 품질의 쌀을 생산하는 데 필수적인 프로그래밍 도구 기술을 세계적으로 선도해 갈 것이다.

참고 자료

[AG97a] Martin Abadi and Andrew D. Gordon. A Calculus of Cryptographic Protocols: the Spi Calculus. In *the Proceedings of the Fourth ACM Conference on Computer and Communications Security*, 1997.

[AG97b] Martin Abadi and Andrew D. Gordon. Reasoning about Cryptographic Protocols in the Spi Calculus, 1997.

[BN96] Marc Brown and Marc Najork. Distributed Active Objects. Technical Report 141a, DEC Systems Research Center, 1996.

[Car94] Luca Cardelli. A Language with Distributed Scope, 1994.

[CG98] Luca Cardelli and Andrew D. Gordon. A Calculus of Mobile Ambients, 1998.

[DFW96] D. Dean, E. Felten, and D. Wallach. Java security: From hotjava to netscape and beyond. In *Proceedings of 1996 IEEE Symposium on Security and Privacy*, May 1996.

[FG96] Cédric Fournet and Georges Gonthier. The Reflexive CAHM and the Join-Calculus. In *Proceedings of the Annual ACM Symposium on Principles of Programming Languages*, 1996.

[FGL+96] Cédric Fournet, Georges Gonthier, Jean-Jacque Lévy, Luc Maranget, and Didier Rémy. A Calculus of Mobile Agents, 1996.

[FLMR96] Cédric Fournet, Cosimo Laneve, Luc Maranget, and Didier Rémy. Implicit Type à la ML for the Join-Calculus, 1996.

[HKM+98] Michael Hicks, Pankaj Kakkar, Jonathan Moore, Carl Gunter, and Scott Nettles. PLAN: A Programming Langauge for Active Networks, 1998.

[Lan94] C. Langton. Modeling adaptive autonomous agents. *Artificial Life Journal*, 1(1 and 2), 1994.

[MPW89a] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part i. Technical Report ECS-LFCS-89-85, University of Edinburgh, 1989.

[MPW89b] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part ii. Technical Report ECS-LFCS-89-86, University of Edinburgh, 1989.

[MTHM97] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.

[Nec97] George C. Necula. Proof-Carrying Code. In *ACM Symposium on Principles of Programming Languages*, pages 106–119, 1997.

[Pro] INRIA Project Cristal. (`http://pauillac.inria.fr/cristal/`).

[Rou96] F. Rouaix. A web navigator with applets in CAML. In *Proceedings of the 5th International World Wide Web Conference, in Computer Networks and Telecommunications Networking*, volume 28, pages 7–11, 1365–1371, 1996.

[SFG+96] J. Smith, D. Farber, C. Gunter, S. Nettles, D. Feldmeier, and W. Sincoskie. SwitchWare: Accelerating network evolution(white paper), 1996.

[TLP+93] Bent Thomsen, Lone Leth, Sanjiva Prasad, Tsung-Min Kuo, Andre Kramer, Fritz Knabe, and Alessandro Giacalone. Facile Antigua release programming guide. Technical Report ECRC-93-20, ECRC, 1993.

[TW96] D. Tennenhouse and D. Wetherall. Towards an active network architecture. *Computer Communication Review*, 26(2), 1996.