효과 타입 시스템을 이용한 기계어 코드의 검증
(Verification of Machine Codes using an Effect Type System)

정 재 윤 † 류 석 영 † 이 광 근 ††
(Jaeyoun Chung) (Sukyoung Ryu)(Kwangkeun Yi)

요 약 네트웍을 이용한 분산 컴퓨팅 환경이나 소프트웨어의 안전성이 중요한 시스템 등에서, 외부로부터 전달되는 코드의 안전성을 검증하는 일이 점점 더 중요한 문제가 되고 있다. 전자메일에 첨부되어 온 코드나 웹브라우저를 통해 외부로부터 전달된 코드를 수행하는 일이 일상생활에서 자주 발생하고 있다.

본 논문에서는 기계어 코드의 성질을 검증하는 방법을 제안한다. 코드제공자가 기계어를 만듦과 동시에 그 기계어 코드의 성질을 만들어내어 전달하면, 코드사용자는 전달받은 코드와 그 성질이 부합하는지를 검사하는 것이다. 소스언어의 안전성은 잘 정의된 컴파일러 시스템이 검증해 주지만, 중간언어나 기계어 코드의 성질을 검증하는 잘 정의된 방법은 아직 개발되어있지 않다.

중간언어로 etySECK을 설계하고 이 언어로 작성된 프로그램의 성질을 검증하는 방법을 제안한다. 그리고 타입과 효과분석방식을 사용하여 설계된 시스템의 안전성을 증명한다.

Abstract Verification of the safety of untrusted codes becomes an important issue in the mobile computing environment and the safety-critical software systems. Recently, it is very common to run the codes attached to the electronic mails or downloaded from the web browsers.

We propose the verification method of the machine code property. The code producer delivers the machine code and its property, then the code consumer checks whether the delivered code satisfies the delivered property. The safety of source codes is verified by the well-defined compiler systems but the verification mechanism for machine codes is not well defined yet.

We design an intermediate language etySECK and propose the verification method of the property of etySECK programs. And then we prove the soundness of our system which is the type system with effect extension.

1. 서 론

이 장에서는 외부로부터 전송된 코드를 수행할 때 발생할 수 있는 문제에 대해서 살펴보고, 이를 해결하기 위해 본 논문에서 제안하는 방법을 간략히 소개한다.

1.1 문제점

출처를 알 수 없는 코드를 수행할 경우에는 이 코드를 검증하는 일이 매우 중요해진다. 전자 우편을 통해 전해진 코드라든지 웹브라우저에서 수행되는 자바 코드[1] 등과 같이, 다른 호스트로부터 전달된 코드를 검증 없이 수행하는 것은 위험한 일이다. 이러한 코드를 수행해서 컴퓨터가 바이러스에 감염된다든지, 시스템에 불필요한 부하를 걸게 되는 등의 일이 발생할 수 있기 때문이다.

소스 언어는 코드를 전송하는 데 있어서 적당한 방법이 아니다. 첫째로, 코드 소비자가 매번 목적 코드로 번역해야 하는 부담이 있기 때문이다. 코드 소비자는 외부로부터 전달받은 코드를 짧은 시간에 수행할 수 있어야 하는데, 매번 다시 번역하는 것은 상당한 부담이 된다. 둘째로, 외부로부터 전달되는 코드는 한 가지 언어만으로 짜여지지 않는 경우가 많기 때문이다. 두 가지 이상의 언어로 짜여진 프로그램을 번역하여 중간 언어 형태로 전송한다면, 코드 소비자는 한가지 언어에 대한 코드 검증만을 다루면 되므로 부담이 줄어든다. 셋째로, 소프트웨어 회사측에서는 소스 코드를 그대로 전달하는 일을 꺼리고 있다. 컴파일 된 HTML문서나 코드를 알아보지 못하도록 재배치한 코드들이 쉽게 볼 수 있는 예

† 비 회 원 : 한국과학기술원 전자전산학과
puppy@ropas.kaist.ac.kr
jay@mail.koscom.co.kr
†† 종신회원 : 한국과학기술원 전자전산학과 교수
kwang@cs.kaist.ac.kr
논문접수 : 2000년 1월 17일
심사완료 : 2000년 6월 8일

이다.

따라서, 기계어나 중간 언어 코드에 대한 성질을 검증하는 방법이 필요하다. 이 논문에서는 코드의 성질이 기계어 코드와 함께 전달되는 상황을 다루기로 한다. 코드와 함께 전달되는 정보들은 코드의 성질이 변하지 않음을 보장해 주거나, 코드를 목적어 코드로 변환할 때 최적화 과정을 도와주는 정보들이다.

저수준 언어의 검증에 관한 연구가 몇 가지 진행되고 있다. 어떤 연구들은 코드 수행 시의 안전성만을 보장해 주고[2], 어떤 연구들은 잘 정의된 방법으로 기술되어 있지 않거나[1] 사용하기에 어렵다[3]. 기존의 연구들에 대해서는 다음 장에서 자세히 다루기로 한다.

1.2 우리의 접근 방법

본 논문에서는 etySECK이라는 기계를 설계하고, etySECK 프로그램의 성질을 검증하기 위하여 타입 시스템을 효과 분석 방법으로 확장한 효과 타입 시스템(effect type system)을 개발하였다. etySECK은 타입과 효과가 코드에 표시되어 있는 스택 기반 기계이다. 임의의 타입 구조를 갖춘(*higher-order & typed*) 언어를 쉽게 번역할 수 있게 하기 위하여 함수값과 재귀호출을 지원할 수 있도록 설계하였다. 효과 타입 시스템을 이용하여 코드의 안전성뿐만 아니라 코드가 갖는 다른 성질들을 표현할 수 있기 때문에, 목적어 코드로 번역할 때 필요한 최적화 정보를 전달하기가 편리하게 되어 있다.

본 연구는 외부로부터 전달받은 코드의 안전성과 다양한 정적 분석 결과를 효과적으로 검증할 수 있다. 본 논문에서는 정적 분석의 한 예로서 코드가 수행되면서 발생하는 메모리 반응을 표현하고 검증하는 방법을 제안하였다. 효과 타입 시스템을 사용하여 비슷한 방법으로 다른 정적 분석 결과를 표현하고 검증할 수 있도록 확장 가능하다.

1.3 논문의 구성

2장에서는 관련 연구들에 대해 알아보고, 3장에서는 본 논문에서 제안하는 방법을 상세하게 기술한 후 제시된 방법이 올바르다는 것을 증명한다. 4장에서는 구현을 통한 구체적인 예를 보이고, 다른 연구들과 비교 분석한다. 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

외부 코드의 성질을 기계적으로 검증하는 방법에 관한 연구가 몇 가지 진행되고 있다. Leroy의 기존 연구들에 대한 조사 자료[4]와 Kozen의 논문[5]은 코드의 안전성을 보장하기 위한 다양한 방법들을 설명하고 있다. 또한, 정적 분석 방법 중에서 코드의 안전성 검증을 위해 사용할 수 있는 몇 가지 방법들이 있다. 2.1절에서는 기존 연구들의 결과와 그 제약점에 대해서 알아보고, 2.2절에서는 본 논문에서 사용하는 타입 분석 방법과 효과 분석 방법에 대해서 간단히 살펴본다.

2.1 코드 검증의 방법들

이 절에서는 코드 검증에 관한 이미 알려진 몇 가지 방법들에 대해서 알아본다. 본 논문에서 제안한 방법과의 차이에 대해서는 4장에서 다루기로 한다.

2.1.1 Java

자바[1]는 선(SUN) 마이크로시스템즈사가 개발한 객체 지향, 플랫폼 독립적인 언어인데, 소스 언어를 컴파일 하여 여러 기계에서 수행할 수 있는 바이트 코드를 생성해 낸다. 코드 소비자인 자바 가상 기계(Java Virtual Machine)[6, 7]는 전송되어 온 바이트 코드를 수행해도 잘못된 경우가 발생하지 않는지를 코드 수행 전에 검증하게 된다.

코드의 안전성을 보장하기 위해서 자바 코드에는 많은 제약이 가해진다. 이들 제약은 정적인 제약(static constraints)과 구조적인 제약(structural constraints)으로 나누어 볼 수 있다. 정적인 제약([1] 4.8.1절) 은 코드 부분에 대한 제약으로, 연산 코드(opcode)의 길이와 값, 각 명령어(instruction)가 가질 수 있는 피연산자(*operand*)의 종류 등이다. 구조적인 제약([1] 4.8.2절)은 명령어들 사이의 관계에 대한 제약으로, 변수에 값이 할당되기 전에 꺼내려고 하지 않는다거나 특정 명령어 다음에 오는 명령어의 종류 등이다.

초기 가상 기계의 구현에는 많은 문제점이 발견되었다. Freund와 Mitchell[8] 논문의 관련 연구 부분에서는 자바 바이트 코드의 안전성을 증명하기 위한 여러 가지 프로젝트들과 문제점들을 소개한다. 자바 전체 언어에 대한 검증 규칙이 옳은지에 대한 증명은 알려진 것이 없다.

2.1.2 증명 전달 코드 - PCC (*Proof Carrying Code*)

증명 전달 코드(PCC: Proof Carrying Code[3])는 코드 검증의 시초가 되는 연구이며 가장 일반적인 방법이다. 코드 제공자는 코드의 중간 중간에 그 코드의 성질에 대한 증명을 함께 전달한다. 코드 소비자는 코드와 함께 따라온 증명이 제대로 된 증명인지를 검증함으로써, 그 코드의 성질을 검증할 수 있다.

증명 전달 코드는 타입이 있는 람다 계산법(typed lambda calculus)에 기초해서 증명을 표현하고 검증한다. 코드 제공자는 코드뿐 아니라 코드 성질의 증명을 함께 만들어야 하는 어려움이 있다. 반면에 코드 소비자

는 증명이 제대로 된 것인지 검증만 하면 되기 때문에, 코드 수행 중에 검증해야 하는 부담이 줄어들게 된다.

2.1.3 타입이 붙은 기계어 - TAL (Typed Assembly Language)

타입이 붙은 기계어(TAL: Typed Assembly Language[2])는 기계어에 타입이 붙은 형태이다. 초기에는 RISC 기계어에 대해서 설계가 되었으나 최근에 와서는 다른 기계어[9, 10]로까지 확장되었다. 이 연구는 정적 타입(static type)을 이용하기 때문에, 동적 타입(dynamic type)에 비해 효과적이고 수행 중에 타입 검사를 하지 않아도 된다는 이점이 있다.

이 연구의 핵심적인 생각은 타입 정보를 기계어에까지 보존함으로써 정적 타입 시스템을 사용하는 언어의 장점 - 수행 중에 잘못된 값을 만들어 내는 코드를 미리 걸러 낼 수 있다 - 를 이용하는 것이다. 타입이 붙은 기계어를 이용하여, 코드가 수행 중에 잘못된 값을 만들어 낼 지를 검증할 수 있다. 타입이 붙은 중간 언어를 기계어로 번역하는 것은 [2]에 나와 있다. 이 방법은 의존적인 타입에 대한 연구(Dependently Typed Assembly Language[11])로 발전되었다.

2.2 타입 시스템과 효과 분석 시스템

본 논문에서 제안하는 방법은 타입 시스템[12]과 효과 분석 방법[13]을 이용한다. 타입은 프로그램이 수행하면서 만들 수 있는 가능한 값들의 집합을 의미하고, 효과는 프로그램이 그 타입의 값을 만드는 데 발생시킬 수 있는 반응들의 집합을 의미한다. 이들은 모두 프로그램이 실제로 수행했을 때 발생시킬 수 있는 값이나 반응들을 충분히 포함하는 정적인 성질이다.

2.2.1 타입 시스템

타입 시스템은 프로그램이 발생시킬 수 있는 오류를 줄이기 위한 방법이다. 프로그램은 수행 중에 프로그래머가 예측하지 못한 잘못으로 인해서 오류를 발생시킬 수 있다. 이런 문제를 줄이기 위해 언어를 설계하는 사람들이 고안해 낸 방법이 타입 시스템이다.

Miller[12]는 let을 이용하여 하나의 타입으로 여러 타입을 표시할 수 있는 복합형 타입 시스템(Let-polymorphic type system)을 제안하였다. 이 타입 시스템은 실행 중에 문제를 발생시킬 수 있는 프로그램을 컴파일 시에 미리 걸러 낼 수 있도록 정교하고, 동시에 대부분의 올바른 프로그램을 문제없이 받아들일 수 있도록 유연하게 설계되어 있다. 이 시스템은 "타입 검증이 성공적으로 이루어진 프로그램은 수행 중에 잘못된 값을 만들어 내지 않는다"라는 좋은 성질을 제공한다.

복합형 타입 시스템에 대한 몇 가지 타입 유추 알고리즘이 개발되었다. Miller는 알고리즘 W[14]를 제안하였고, Tofte[15]와 Leroy[16]는 메모리 값을 변화시키는 언어로 확장하였다. 컴파일러 설계자들 사이에 사용되던 타입 유추 알고리즘으로 M이 있었고, 이것은 후에 안전성과 완전성이 증명되었다[17].

2.2.2 효과 분석 시스템

효과 분석 방법은 프로그램이 수행 중에 발생시킬 수 있는 반응들을 표현하기 위해서 고안되었다. 효과 분석 방식은 주로 타입 시스템과 연결되어서 표현되는데, 함수가 발생시킬 수 있는 반응들이 함수의 타입에 붙어 있게 된다.

효과 분석 방식은 다양한 프로그램 반응들을 표현할 수 있다. Jouvelot과 Gifford는 프로그램 실행 시에 프로그램의 흐름(control)이 어떻게 변하는지를 분석하는 데 효과 분석 방법을 사용하였고[18], Leroy와 Pessaux는 프로그램이 발생시킬 수 있는 예외 상황을 분석하는 데 사용하였다[19]. 효과 분석 방법은 주로 메모리 반응을 추적하기 위해 사용되어 왔는데[13, 20, 21], 본 논문에서도 메모리 반응을 분석하는 데 이용하였다. 기존의 연구는 소스 언어를 대상으로 한 데 반하여, 본 연구에서는 기계어 코드의 메모리 반응을 분석하는 데에 효과 분석 시스템을 사용하였다.

효과 분석 시스템을 이용한 메모리 반응 분석 결과는 컴파일러가 기계어 코드를 목적어 코드로 변환할 때 최적화 정보로 사용될 수 있다. Tofte와 Talpin은 모든 데이타를 스택 형식으로 할당하고 반환할 수 있도록 하기 위해 이 방법을 사용하였고[22, 23], Talpin과 Jouvelot은 벡터 명령어를 병렬화하는 데 사용하여 FX 컴파일러[24]에 구현하였다.

3. 우리의 방법

본 논문에서 사용하는 etySECK은 스택 기반 기계로서 타입과 효과가 기계어 코드에 표시되어 있고, 언어의 정형화된 의미가 변환 규칙으로 정의되어 있다. 함수를 주고받을 수 있는 함수형 언어를 이 언어로 쉽게 번역하기 위해서, 함수값과 재귀호출을 지원할 수 있도록 설계하였다.

3.1절에서는 본 연구가 가정하는 코드 수행 환경에 대해서 설명하고 3.2절에서는 etySECK의 문법 구조와 의미 구조에 대해서 설명한다. 3.3절에서는 검증 방식으로 쓰이는 타입 규칙을 설명하고 3.4절에서 이 규칙이 올바름을 증명한다.

3.1 코드 수행 방법

본 연구는 그림 1과 같은 수행 환경을 가정한다. 외

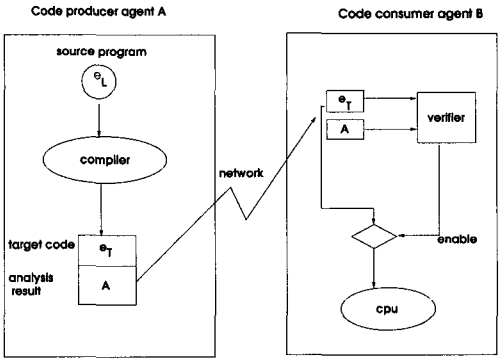

그림 1 코드 수행 방법

부의 코드 제공자로부터 코드가 전달되고, 이 코드의 성질은 타입과 효과로 코드에 표시되어 있다. 표시된 성질들은 코드를 수행해도 안전함을 보장하는 동시에 코드가 수행될 때 일으킬 수 있는 메모리 반응을 기술한다.

코드 제공자는 코드를 생성하면서 그 코드의 성질도 얻게 된다. 이 성질을 타입과 효과의 형태로 코드 사이에 넣게 되고, 생성된 코드는 코드 소비자에게 전달된다. 코드 소비자는 전달받은 코드에 표시된 타입 정보를 검증함으로써, 코드에 대한 안전성과 목적 코드로의 변환 시 필요한 최적화 정보를 얻게 된다.

3.2 etySECK

etySECK은 Landin이 제시한 SECD 기계[25, 26, 27, 28]의 변형된 형태로서, 타입과 효과 분석 정보가 전달될 수 있도록 설계되어 있다. Plotkin[28]이 보인 것처럼 etySECK은 함수의 인수로 값을 전달하는 (call-by-value) 의미 구조와 함수의 인수로 이름을 전달하는(call-by-name) 의미 구조를 모두 처리할 수 있는데, 본 논문에서는 함수의 인수로 값을 전달하는 의미 구조를 사용하기로 한다.

문법 구조와 의미 구조는 각각 그림 2와 그림 3에 있다. 수행 상태는 스택, 메모리, 환경, 코드, 다음 수행할 코드, 그리고 현재 수행 상태까지 일어났던 메모리 반응 정보로 구성되어 있다. 앞으로 이 논문에서는 .을 리스트를 생성하는 연산자로 사용하며, 길이가 1인 리스트는 그 값만을 표시하기로 한다.

3.2.1 문법 구조

스택(stack)과 환경은 값들의 나열이다. 대부분의 명령은 스택 상위의 값 몇 개를 사용해서 새로운 값을 스택에 만들어 내게 된다. 코드를 수행할 때 필요한 값들이 환경에 포함되어 있다. 다음에 할 일 부분은 환경과

기계의 수행 상태: (S, H, E, C, K, φ)

$S \in Stack$	=	*Value* list	스택
$H \in Heap$	=	$Loc \rightarrow Value$	
$E \in Env$	=	*Value* list	환경
$K \in Cont$	=	$(Env \times Cmds)$ list	다음에 할 일
$Value \ni v$	::=	p	기본값
		$()$	유닛
		l_ρ	메모리 주소
		$\mathtt{fn}(\tau, C, E)$	함수
		$\mathtt{rfn}(\tau, C, E)$	재귀함수
		$\Lambda(\vec{\alpha}, \vec{\rho}).v$	복합형 타입의 함수
$Prim.Value \ni p$	::=	$n \mid \mathtt{true} \mid \mathtt{false}$	정수, 진리값
$Type \ni \tau$	::=	$i \mid b$	정수 타입, 진리 타입
		unit	유닛 타입
		$ref_\rho \tau$	주소 타입
		$\tau \xrightarrow{\varphi} \tau$	함수 타입
		α	타입 변수
		$\forall(\vec{\alpha}, \vec{\rho}).\tau$	복합형 타입
$Region \ni \rho$	::=	r_i	영역 상수
		ρ_i	영역 변수
$Cmd \ni C$	::=	*nil*	
		$\mathtt{frame}.C \mid \mathtt{deframe}.C$	
		$\mathtt{fetch}(n).C \mid \mathtt{quote}(p).C$	
		$\mathtt{add}.C \mid \mathtt{cond}(C_1, C_2).C$	
		$\mathtt{fn}(\forall\vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi} \tau_2).C$	
		$\mathtt{rfn}(\forall\vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi} \tau_2).C$	
		$\mathtt{app}.C$	
		$\mathtt{tapp}(\vec{\tau}, \vec{\rho}).C$	
		$\mathtt{ref}(\rho).C \mid \mathtt{get}.C \mid \mathtt{set}.C$	
$Effect \ni \varphi$	::=	ϵ	효과 없음
		$\{init(\rho)\}$	메모리 할당
		$\{read(\rho)\}$	메모리 읽기
		$\{write(\rho)\}$	메모리 쓰기
		$\varphi \cup \varphi'$	

그림 2 etySECK의 기계 수행 상태

코드의 쌍들의 나열인데, 각각의 쌍은 app 명령으로 인해서 잠시 수행이 멈춰진 기계의 수행 상태이다.

효과(effect)는 현재의 수행 상태까지 수행하면서 일으킨 메모리 반응을 담고 있다. 일반적으로 프로그램이 수행하면서 사용하는 메모리의 양을 컴파일 시간에 결정하는 것은 불가능하므로 사용되는 메모리를 요약(abstract)할 필요가 있는데, *영역*(region)은 이러한 용도로 사용된다. 영역은 실제로 수행할 때 할당되어서 사용되는 메모리의 한 부분을 의미한다. 의미상 같은 영역으로 표시되는 메모리는 같은 블록에 할당된다고 생각할 수 있으며, 그 블록의 특정 부분을 읽거나 쓰는 일은 그 블록 안에서 일어나는 일이므로, 그 블록에 대한 영역에서 일어나는 반응으로 요약할 수 있다. 이들 메모리 반응을 각각 *init*, *read*, *write*으로 표시하겠다.

이 언어가 다루는 값에는 정수, 유닛 등의 기본적인 값과 메모리 주소값, 함수값, 그리고 임의의 타입을 받아서 특정 타입의 값을 만들어 내는 매개화된 값을 포함한다. 기본적인 값과 이들에 대한 명령어들은 설명을 간단히 하기 위해서 간소화시켰으나 쉽게 확장할 수 있

다. 함수값은 코드와 그 코드가 수행 중에 참조하는 환경의 쌍이다. 여기서 함수값에는 그 함수값이 받아들이는 값과 반환값의 타입이 표시되어 있음에 주목할 필요가 있다. 함수값에는 fn, rfn의 두 가지 형태가 있는데, 이들은 재귀적인 값과 그렇지 않은 값을 구분하기 위해서 사용되었다. 매개화된 값은 타입을 받아서 여러 타입의 값을 나타내는 값을 표현한다. 매개화된 값이 될 수 있는 값은 함수값뿐이다.

언어가 다루는 타입은 기본적인 타입, 유닛 타입, 주소 타입, 함수 타입, 그리고 타입 변수와 복합형 타입 등으로 구성된다. 이들은 각각 언어가 다루는 값에 대응되는 타입이다. 특히, 함수 타입은 함수가 수행 중에 일으킬 수 있는 메모리 반응들을 타입에 표시해 두었다. 복합형 타입은 매개화된 값에 대한 타입이다. 값, 스택, 환경, 기계 수행 상태 등에 대해서 자유 타입 변수(FTV: Free Type Variables)가 정의되어 있는데, 이것은 각각의 타입에서 제한되지 않은 타입 변수들의 집합을 의미한다. 치환은 타입 변수를 새로운 타입이나 영역으로 대체하는, 정의역이 유한한 함수를 의미한다. 이 논문에서는 자유 타입 변수와 치환의 엄밀한 정의는 생략하기로 한다.

프로그램은 자유 타입 변수가 없는 코드로 제한하기로 한다. 즉,

정의 1 (프로그램) *프로그램은 자유 타입 변수가 없는 코드를 말한다.* ($\mathrm{FTV}(C) = \phi$)

3.2.2 의미 구조

언어의 의미 구조는 그림 3과 같이 변환 관계식($\Rightarrow$)으로 기술되어 있다. 각각의 명령은 스택 상위의 값 몇 개를 소비하여, 새로운 값을 스택에 만들어 낸다.

현재 코드가 아무 것도 없다면, 다음에 수행할 일들 중에서 가장 처음의 쌍으로 환경과 코드를 대체한다. frame 명령은 스택의 가장 위에 있는 값을 환경으로 옮겨서 남은 코드에서 참조할 수 있도록 한다. deframe은 이에 대응하는 명령어로 환경의 상위에 있는 값을 제거한다. 환경에 있는 값을 사용하기 위해서는 fetch 명령어를 사용하여 그 값을 스택의 가장 위로 복사하게 된다. 새로운 상수는 quote 명령어를 사용해서 만들 수 있다. add와 같은 기본적인 산술 명령어들은 필요로 하는 인자를 스택에서 제거하고 결과값을 다시 스택에 쌓는다. cond 명령어는 스택의 최상위에 있는 값을 보고 계속 수행의 흐름을 조정한다.

함수값은 fn과 rfn 명령어로 만들어진다. 이 함수값은 매개화된 값이므로 직접 사용할 수 없고, 반드시 tapp

$$
\begin{array}{l}
(S,H,E,nil,(E',C).K,\varphi) \Rightarrow (S,H,E',C,K,\varphi)\\
(v.S,H,E,\mathtt{frame}.C,K,\varphi) \Rightarrow (S,H,v.E,C,K,\varphi)\\
(S,H,v.E,\mathtt{deframe}.C,K,\varphi) \Rightarrow (S,H,E,C,K,\varphi)\\
(S,H,v_0.\cdots.v_n.E,\mathtt{fetch}(n).C,K,\varphi) \Rightarrow (v_n.S,H,v_0.\cdots.v_n.E,C,K,\varphi)\\
(S,H,E,\mathtt{quote}(p).C,K,\varphi) \Rightarrow (p.S,H,E,C,K,\varphi)\\
(i_2.i_1.S,H,E,\mathtt{add}.C,K,\varphi) \Rightarrow (i_1+i_2.S,H,E,C,K,\varphi)\\
(\mathtt{true}.S,H,E,\mathtt{cond}(C_1,C_2).C,K,\varphi) \Rightarrow (S,H,E,C_1.C,K,\varphi)\\
(\mathtt{false}.S,H,E,\mathtt{cond}(C_1,C_2).C,K,\varphi) \Rightarrow (S,H,E,C_2.C,K,\varphi)\\
(S,H,E,\mathtt{fn}(\forall\vec{\alpha},\vec{\rho}.C':\tau_1\xrightarrow{\varphi_f}\tau_2).C,K,\varphi) \Rightarrow\\
\quad(\Lambda(\vec{\alpha},\vec{\rho}).\mathtt{fn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C',E).S,H,E,C,K,\varphi)\\
(S,H,E,\mathtt{rfn}(\forall\vec{\alpha},\vec{\rho}.C':\tau_1\xrightarrow{\varphi_f}\tau_2).C,K,\varphi) \Rightarrow\\
\quad(\Lambda(\vec{\alpha},\vec{\rho}).\mathtt{rfn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C',E).S,H,E,C,K,\varphi)\\
(v.\mathtt{fn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C_f,E_f).S,H,E,\mathtt{app}.C,K,\varphi) \Rightarrow (S,H,v.E_f,C_f,(E,C).K,\varphi)\\
(v.\mathtt{rfn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C_f,E_f).S,H,E,\mathtt{app}.C,K,\varphi) \Rightarrow\\
\quad(S,H,v.\mathtt{rfn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C_f,E_f).E_f,C_f,(E,C).K,\varphi)\\
(\Lambda(\vec{\alpha},\vec{\rho}).\mathtt{fn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C_f,E_f).S,H,E,\mathtt{tapp}(\vec{\tau},\vec{r}).C,K,\varphi) \Rightarrow\\
\quad([\tau_i/\alpha_i,r_i/\rho_i]\mathtt{fn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C_f,E_f).S,H,E,C,K,\varphi)\\
(\Lambda(\vec{\alpha},\vec{\rho}).\mathtt{rfn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C_f,E_f).S,H,E,\mathtt{tapp}(\vec{\tau},\vec{r}).C,K,\varphi) \Rightarrow\\
\quad([\tau_i/\alpha_i,r_i/\rho_i]\mathtt{rfn}(\tau_1\xrightarrow{\varphi_f}\tau_2,C_f,E_f).S,H,E,C,K,\varphi)\\
(v.S,H,E,\mathtt{ref}(\rho).C,K,\varphi) \Rightarrow\\
\quad(l_\rho.S,H[l_\rho\rightarrow v],E,C,K,\varphi\cup\{init(\rho)\}) \quad l_\rho\notin \mathrm{Dom(H)}\\
(l_\rho.S,H,E,\mathtt{get}.C,K,\varphi) \Rightarrow\\
\quad(H(l_\rho).S,H,E,C,K,\varphi\cup\{read(\rho)\}) \quad l_\rho\in \mathrm{Dom(H)}\\
(v.l_\rho.S,H,E,\mathtt{set}.C,K,\varphi) \Rightarrow\\
\quad(().S,H[l_\rho\rightarrow v],E,C,K,\varphi\cup\{write(\rho)\}) \quad l_\rho\in \mathrm{Dom(H)}
\end{array}
$$

그림 3 etySECK의 기계변환 관계식

를 사용해서 필요한 값으로 변환시킨 후에야 사용이 가능하다. fn과 rfn은 tapp 명령에 의해 사용되는 방법이 다르다. app는 fn이나 rfn이 수행할 환경을 만들고 그 환경에서 함수를 수행한다. 이 때 rfn의 경우는 이 함수값 자체를 참조할 수 있도록 환경에 쌓아 두어서 재귀호출이 가능하도록 한다. tapp 명령어는 앞에서 기술한 것과 같이 매개화된 값을 특정한 타입의 값으로 구체화시킨다.

메모리에 관한 명령어에는 ref, get, set 등이 있다. 먼저 ref는 새로운 메모리 영역을 할당하고, 이에 대한 주소값을 스택의 최상위에 놓게 된다. get은 스택 최상위에 있는 메모리 주소값에 저장된 값을 읽어 낸다. set 명령어는 메모리 주소와 값을 받아서 메모리 주소에 해당하는 메모리 영역에 저장된 내용을 새로운 값으로 대체한다. 이 때 각각의 메모리 관련 명령어는 효과를 발생시키게 된다.

전체 프로그램의 수행은 프로그램(코드)를 받아서 기계의 여섯 가지 요소 중 코드 부분만을 가진 상태에서 변환 규칙을 적용하여 진행하게 된다. 기계의 코드 부분과 앞으로 할 일 부분이 비어 있게 되거나 더 적용할 변환 규칙이 없으면 수행이 끝나게 된다. 전자의 경우를 정상 종료라고 하고, 후자의 경우를 오류 상태라고 한

다. 프로그램의 정상 종료와 오류 상태는 다음과 같이 정의할 수 있다:

정의 2 (정상 종료) 프로그램 C에 대해서

$$(nil, \phi, nil, C, nil, \phi) \stackrel{*}{\Rightarrow} (v.s, H, e, nil, nil, \varphi)$$

이면, 이 프로그램은 정상 종료 후 값 v를 만든다고 한다.

정의 3 (오류 상태) *etySECK 기계의 수행 상태* (S, H, E, C, K, φ) $(C \neq nil,\ K \neq nil)$ 는

$$(S, H, E, C, K, \varphi) \Rightarrow (S', H', E', C', K', \varphi')$$

인 다음 수행 상태 $(S', H', E', C', K', \varphi')$이 존재하지 않으면 오류 상태라 한다.

다음절에서는 타입 검증을 거친 프로그램은 오류 상태에 도달하지 않음을 보인다.

3.3 타입 시스템

프로그램의 오류는 기계 수행 상태가 정상 종료가 아닌 수행 상태에서 다음 상태로의 변환이 없는 경우에 발생한다. 잘 정의된 타입 시스템은 일단 타입 시스템의 검증을 통과한 프로그램은 수행 중에 오류 상태에 도달하지 않는다는 것이 알려져 있다.

기계 수행 상태의 타입은 다음과 같은 판단 규칙에 의해서 결정된다:

$$(S, H, E, C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o).$$

이 식이 의미하는 것은 "*주어진 기계 수행 상태가 수행을 끝마쳤다면, 그 결과의 스택, 환경, 그리고 효과의 타입이 각각 s_o, e_o, φ_o가 된다.*"이다. 각각의 타입 규칙은 그림 4에 나와 있다.

주어진 기계 수행 상태의 타입을 결정하기 위해서는 요약된 형태의 수행이 필요하다 - 다시 말해서, 타입에 기초한 수행이 필요하다. 이를 위해서는 스택, 환경, 다음에 할 일에 대한 타입을 결정해야 하는데, 이는 다시 값이 어떤 타입을 갖는지를 결정해야 정의할 수 있다.

스택, 환경, 다음에 할 일에 대한 타입은 단순히 각 값의 타입의 나열이다. 판단 규칙 $H \models v : \tau$는 주어진 메모리 H 상에서 값 v가 τ라는 타입을 가짐을 의미한다. 메모리 주소값의 타입을 결정하는 규칙 l_ρ는 그 주소가 가리키는 메모리 영역에 저장된 값의 타입에 의해서 정의되므로, 값에 대한 타입을 결정하기 위해서는 메모리 H가 필요하다.

정수, 참, 거짓 등과 같은 기본적인 값은 각각의 타입을 갖는다. 확장성을 위해서 이 논문에서는 기본적인 값

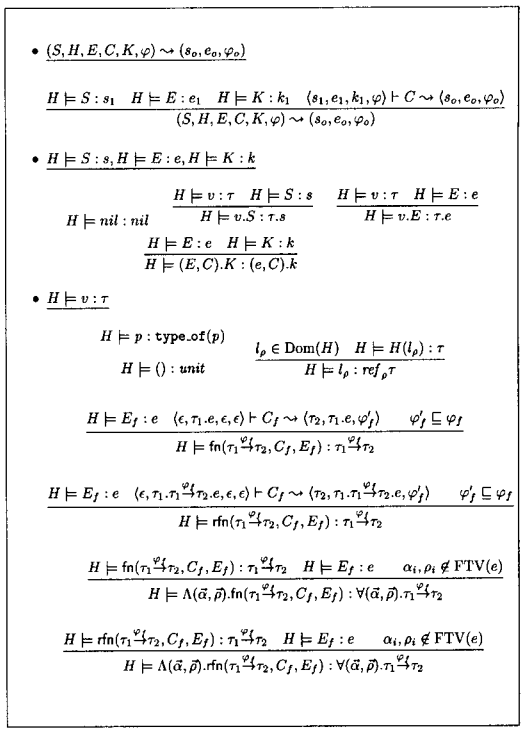

그림 4 etySECK의 타입 규칙(1)

에 대해서 그 타입을 결정해 주는 함수(type_of)가 있다고 가정한다. 메모리 주소값 l_ρ는 이 주소에 해당하는 메모리에 저장된 값의 타입이 τ일 때 $ref_\rho \tau$ 타입을 갖는다. 함수값에 대한 타입은 그 함수값에 표시되어 있는 타입이 맞는지를 검증함으로써 결정된다. 인자의 타입을 주고 함수의 몸체를 요약적인 방법으로 수행했을 때 생기는 타입이 함수의 결과 타입과 일치하는지를 확인하면 된다. 이 요약 수행은 함수의 몸체 부분에서 필요한 외부 변수는 환경 부분에 있으므로, 제대로 된 함수라면 요약 수행이 성공하게 된다. 주목할 점은 함수를 실제로 수행함으로써 결정되는 효과는 타입에 표시된 것보다 적을 수 있다는 것이다.

코드 부분을 타입에 기초하여 요약 수행하는 것은 다음 형태의 판단 규칙에 의해서 정의된다:

$$\langle s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$$
$$\langle s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle.$$

이 판단 규칙은 "*주어진 타입 s, e, k, φ에 대하여, 코드 부분이 수행을 계속해서 종료 상태에 도달했다면, 그 결과로 생기는 스택, 환경, 그리고 효과의 타입은 각각 s_o, e_o, φ_o이다.*"를 의미한다. 그림 5와 6은 이를 설

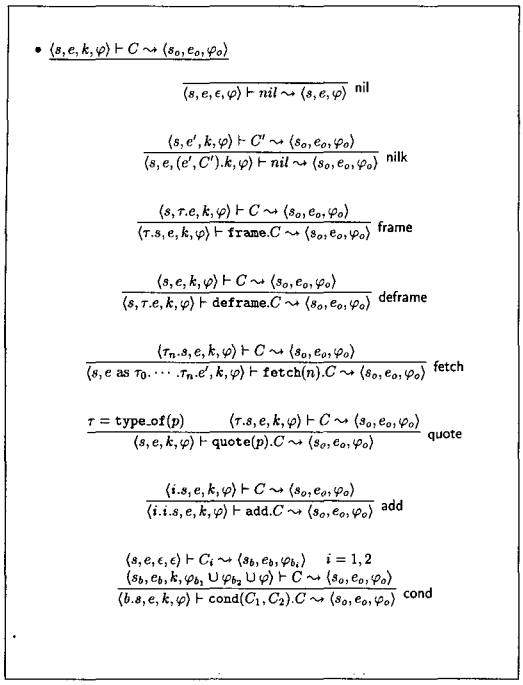

그림 5 etySECK의 타입 규칙(2)

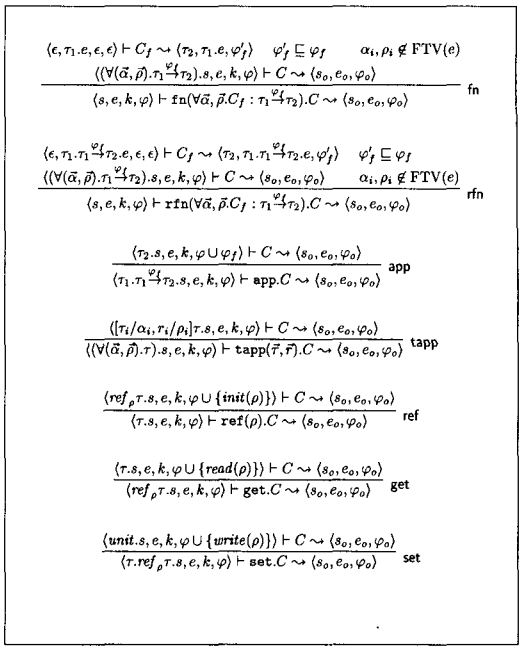

그림 6 etySECK의 타입 규칙(3)

명한다.

각각의 규칙은 실제 변환 관계식에 대한 요약이다. nil 규칙은 더 이상 수행할 코드나 다음에 할 일이 비어 있는 경우에 대한 타입 결정 규칙이다. 이 상태는 기계의 최종 수행 상태이다. nilk 규칙은 더 이상 수행할 코드가 없더라도 다음에 할 일이 비어 있지 않으면, 여기에서 환경과 코드를 얻어 수행을 계속하도록 하고 있다. 같은 방식으로 대부분의 타입 규칙이 정의되어 있다. 주의할 것은 fn, rfn 규칙들은 코드에 명시된 타입이 올바른지를 검사한 다음에 요약적인 방법으로 수행을 계속하게 된다는 것이다. 또한, 결과로 스택에 쌓이게 되는 값은 매개변수화된 타입이다. 재귀적 함수와 그렇지 않은 함수는 app의 타입 검사 시에 차이가 나지 않는다. tapp는 스택 가장 위에 있는 매개화된 타입을 특정한 타입의 값으로 구체화시키는 일을 한다. 메모리에 관한 연산들 - ref, get, set - 에 대해서도 실제 수행과 비슷하게 정의가 된다. 각각이 발생시키는 효과는 타입의 효과 부분에 기록된다.

타입 규칙에 대하여 몇 가지 살펴볼 것이 있다. 첫 번째, 타입 규칙에 새로운 명령어들을 추가하여 확장하는 것이 쉽게 이루어진다. 각각의 규칙은 프로그램의 실제 수행에 대한 요약 수행이기 때문에 확장하기가 어렵지 않게 된다. 두 번째, 효과 부분에 대한 특별한 제약(이 경우 "집합"의 조건만 갖추면 충분하다.)이 없기 때문에, 메모리 반응 이외의 다른 효과 분석을 포함하도록 쉽게 확장할 수 있다. 세 번째, 각각의 타입 검증 규칙은 모호한 문제가 발생하지 않기 때문에 구현이 용이하다. 이것은 위의 타입 규칙이 코드에 붙어 있는 타입의 검증만을 위해서 설계되어 있기 때문이다. 따라서, 필요한 대부분의 일은 코드를 만들어 내는 컴파일러에게 주어지게 된다.

3.4 안전성

지금까지 정의한 타입 규칙의 핵심적인 성질 - 안전성 -을 Wright와 Felleisen[29]의 방법을 사용하여 증명하기로 한다. 안전성 정리를 설명하기 전에, 타입 규칙에 관한 몇 가지 성질들을 알아볼 필요가 있다.

첫째 성질은 타입 판단 규칙$\langle s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$에 나타나는 자유 변수들을 치환하여 주어진 규칙을 구체화할 수 있다는 것이다. 보조 정리 1이 이 성질을 설명한다.

보조정리 1 (치환)

$\langle s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$라면, 임의의 치환 S에 대해서 $\langle \mathcal{S}s, \mathcal{S}e, \mathcal{S}k, \mathcal{S}\varphi\rangle \vdash \mathcal{S}C \rightsquigarrow \langle \mathcal{S}s_o, \mathcal{S}e_o, \mathcal{S}\varphi_o\rangle$가 성립한다.

증명. 타입 규칙 $\langle s,e,k,\varphi\rangle \vdash C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$를 유추하는 각 경우에 대해서 증명해야 한다. 증명의 형태는 각 경우에 대해서 비슷하므로, 이 논문에서는 간단하지 않은 두 가지 경우만을 다루기로 한다.

- $C = \mathtt{fn}(\forall\vec{\alpha},\vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C'$ 인 경우

$\langle s,e,k,\varphi\rangle \vdash \mathtt{fn}(\forall\vec{\alpha},\vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C' \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$ 라 하면 fn에 의해서,

$$\langle \epsilon,\tau_1.e,\epsilon,\epsilon\rangle \vdash C_f \rightsquigarrow \langle \tau_2,\tau_1.e,\varphi'_f\rangle$$
$$\varphi'_f \sqsubseteq \varphi_f \quad \alpha_i,\rho_i \notin \mathrm{FTV}(e) \tag{1}$$

$$\langle(\forall(\vec{\alpha},\vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2).s,e,k,\varphi\rangle \vdash C' \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle \tag{2}$$

이다. 새로운 변수 β_i, η_i에 대해서$\mathcal{S}' = \mathcal{S}[\beta_i/\alpha_i,\eta_i/\rho_i]$ 라 하면, $\alpha_i,\rho_i \notin \mathrm{FTV}(e)$이므로 $\mathcal{S}'e = \mathcal{S}e$ 이다. 그러면 귀납 추론의 가정에 의해서 식 (1)은 다음을 의미한다

$$\langle \epsilon,(\mathcal{S}'\tau_1).\mathcal{S}e,\epsilon,\epsilon\rangle \vdash \mathcal{S}'C_f \rightsquigarrow \langle \mathcal{S}'\tau_2,(\mathcal{S}'\tau_1).\mathcal{S}e,\mathcal{S}'\varphi'_f\rangle$$
$$\mathcal{S}'\varphi'_f \sqsubseteq \mathcal{S}'\varphi_f \quad \beta_i,\eta_i \notin \mathrm{FTV}(\mathcal{S}e). \tag{3}$$

또한 귀납 추론의 가정에 의해서, 식 (2)에 $\mathcal{S}$를 적용하면 다음과 같은 결과를 얻는다

$\langle \mathcal{S}(\forall(\vec{\alpha},\vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2).\mathcal{S}s,\mathcal{S}e,\mathcal{S}k,\mathcal{S}\varphi\rangle \vdash \mathcal{S}C' \rightsquigarrow \langle \mathcal{S}s_o,\mathcal{S}e_o,\mathcal{S}\varphi_o\rangle$.

$\mathcal{S}(\forall(\vec{\alpha},\vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2) \equiv \forall(\vec{\beta},\vec{\eta}).\mathcal{S}'\tau_1 \xrightarrow{\mathcal{S}'\varphi_f} \mathcal{S}'\tau_2$ 이므로,

$$\langle \forall(\vec{\beta},\vec{\eta}).\mathcal{S}'\tau_1 \xrightarrow{\mathcal{S}'\varphi_f} \mathcal{S}'\tau_2.\mathcal{S}s,\mathcal{S}e,\mathcal{S}k,\mathcal{S}\varphi\rangle \vdash \mathcal{S}C' \rightsquigarrow \langle \mathcal{S}s_o,\mathcal{S}e_o,\mathcal{S}\varphi_o\rangle \tag{4}$$

이다. 이제 규칙 fn을 식 (3)과 (4)에 적용하면

$\langle \mathcal{S}s,\mathcal{S}e,\mathcal{S}k,\mathcal{S}\varphi\rangle \vdash \mathtt{fn}(\forall\vec{\beta},\vec{\eta}.\mathcal{S}'C_f : \mathcal{S}'\tau_1 \xrightarrow{\mathcal{S}'\varphi_f} \mathcal{S}'\tau_2).\mathcal{S}C' \rightsquigarrow \langle \mathcal{S}s_o,\mathcal{S}e_o,\mathcal{S}\varphi_o\rangle$,

이고, 이것은 곧

$\langle \mathcal{S}s,\mathcal{S}e,\mathcal{S}k,\mathcal{S}\varphi\rangle \vdash \mathcal{S}(\mathtt{fn}(\forall\vec{\alpha},\vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C') \rightsquigarrow \langle \mathcal{S}s_o,\mathcal{S}e_o,\mathcal{S}\varphi_o\rangle$

를 의미한다.

- $C = \mathtt{tapp}(\vec{\tau},\vec{\tau}).C'$인 경우

$\langle(\forall(\vec{\alpha},\vec{\rho}).\tau).s,e,k,\varphi\rangle \vdash \mathtt{tapp}(\vec{\tau},\vec{\tau}).C' \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$

라 하면, 새로운 변수 β_i, η_i에 대해서 다음이 성립한다

$\langle(\forall(\vec{\beta},\vec{\eta}).[\beta_i/\alpha_i,\eta_i/\rho_i]\tau).s,e,k,\varphi\rangle \vdash \mathtt{tapp}(\vec{\tau},\vec{\tau}).C' \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$.

그러면 규칙 tapp 에 의해서 다음이 성립하고

$$\langle[\tau_i/\beta_i,r_i/\eta_i]([\beta_i/\alpha_i,\eta_i/\rho_i]\tau).s,e,k,\varphi\rangle \vdash C' \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle, \tag{5}$$

귀납 추론의 가정에 의해서 식 (5)에 $\mathcal{S}$를 적용하면 다음을 얻게 된다

$$\langle \mathcal{S}([\tau_i/\beta_i,r_i/\eta_i]([\beta_i/\alpha_i,\eta_i/\rho_i]\tau).s),\mathcal{S}e,\mathcal{S}k,\mathcal{S}\varphi\rangle \vdash \mathcal{S}C' \rightsquigarrow \langle \mathcal{S}s_o,\mathcal{S}e_o,\mathcal{S}\varphi_o\rangle$$

β_i, η_i는 새 변수들이기 때문에 다음이 성립하고

$$\mathcal{S}([\tau_i/\beta_i,r_i/\eta_i]([\beta_i/\alpha_i,\eta_i/\rho_i]\tau).s)$$
$$\equiv$$
$$[(\mathcal{S}\tau_i)/\beta_i,(\mathcal{S}r_i)/\eta_i](S[\beta_i/\alpha_i,\eta_i/\rho_i]\tau).\mathcal{S}s,$$

따라서 식 (6)이 성립한다

$$\langle[(\mathcal{S}\tau_i)/\beta_i,(\mathcal{S}r_i)/\eta_i](S[\beta_i/\alpha_i,\eta_i/\rho_i]\tau).\mathcal{S}s,\mathcal{S}e,\mathcal{S}k,\mathcal{S}\varphi\rangle \vdash \mathcal{S}C' \rightsquigarrow \langle \mathcal{S}s_o,\mathcal{S}e_o,\mathcal{S}\varphi_o\rangle \tag{6}$$

식 (6)에 타입 규칙 tapp를 적용하면

$$\langle(\forall(\vec{\beta},\vec{\eta}).\mathcal{S}[\beta_i/\alpha_i,\eta_i/\rho_i]\tau).\mathcal{S}s,\mathcal{S}e,\mathcal{S}k,\mathcal{S}\varphi\rangle \vdash \mathtt{tapp}(\mathcal{S}\vec{\tau},\mathcal{S}\vec{\tau}).\mathcal{S}C' \rightsquigarrow \langle \mathcal{S}s_o,\mathcal{S}e_o,\mathcal{S}\varphi_o\rangle$$

이 되고, 이것은

$\langle(\mathcal{S}(\forall(\vec{\alpha},\vec{\rho}).\tau).s),\mathcal{S}e,\mathcal{S}k,\mathcal{S}\varphi\rangle \vdash \mathcal{S}(\mathtt{tapp}(\vec{\tau},\vec{\tau}).\mathcal{S}C') \rightsquigarrow \langle \mathcal{S}s_o,\mathcal{S}e_o,\mathcal{S}\varphi_o\rangle$

를 의미한다. □

보조 정리 2는 한정된 타입 변수 (혹은 영역 변수)를 임의의 타입 (혹은 영역)으로 구체화할 수 있다는 것을 나타낸다. 보조 정리 2의 증명은 부록에 나와 있고, 보조 정리 1을 사용하여 쉽게 증명할 수 있다.

보조정리 2 (구체화)

$H \models \Lambda(\vec{\alpha},\vec{\rho}).\mathtt{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2,C_f,E_f) : \forall(\vec{\alpha},\vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2$ 라면,

$H \models [\tau_i/\alpha_i,r_i/\rho_i]\mathtt{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2,C_f,E_f) : [\tau_i/\alpha_i,r_i/\rho_i](\tau_1 \xrightarrow{\varphi_f} \tau_2)$ 이다.

보조 정리 3은 주어진 기계 상태의 타입에서 코드의 타입이 결정된다면, 스택의 하위 타입이나 효과를 확장하여도 코드의 타입을 결정하는 데 영향을 미치지 않는다는 것을 의미한다. 보조 정리 4는 주어진 기계 상태의 타입에서 효과를 줄여서 타입 분석을 하면, 결과로 나오는 효과도 줄어든다는 것을 의미한다. 보조 정리 3과 4의 증명은 간단하므로 생략하기로 한다.

보조정리 3 (스택 타입과 효과의 확장성)

$\langle s,e,k,\varphi\rangle \vdash C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$라면, 임의의 s', φ'에 대해서 $\langle s.s',e,k,\varphi\cup\varphi'\rangle \vdash C \rightsquigarrow \langle s_o.s',e_o,\varphi_o\cup\varphi'\rangle$ 이다.

보조정리 4 (효과의 단조 감소)

$\langle s,e,k,\varphi\rangle \vdash C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$이고 $\varphi' \sqsubseteq \varphi$ 라면,

$\langle s,e,k,\varphi'\rangle \vdash C \rightsquigarrow \langle s_o,e_o,\varphi'_o\rangle$이고 $\varphi'_o \sqsubseteq \varphi_o$이다.

보조 정리 5는 다음에 할 일의 타입을 유추하는 방법을 설명하는 것으로서, 안전성 정리를 증명하는 데 결정적인 역할을 한다.

보조정리 5 (다음에 할 일에 대한 타입 유추)

$\langle s_1,e_1,\epsilon,\varphi_1\rangle \vdash C_1 \rightsquigarrow \langle s_2,e',\varphi_2\rangle$ 이고

$\langle s_2.s_3,e_2,k_2,\varphi_2\cup\varphi'\rangle \vdash C_2 \rightsquigarrow \langle s,e,\varphi\rangle$ 라면

$\langle s_1.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1\rangle \vdash C_1 \rightsquigarrow \langle s, e, \varphi\rangle$이다.

증명. C_1의 길이에 대해 귀납적으로 증명해야 한다. 몇 가지 경우만을 살펴보고 나머지는 부록에서 증명하기로 한다.

- $C_1 = \texttt{frame}.C'$인 경우

귀납 추론의 가정에 의해, $s_1 = \tau.s_1'$ 인 s_1'이 존재하고 식 (7)과 (8)이 성립한다

$$\langle \tau.s_1', e_1, \epsilon, \varphi_1\rangle \vdash \texttt{frame}.C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \tag{7}$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi'\rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi\rangle. \tag{8}$$

식 (7)에 규칙 frame을 적용하면

$$\langle s_1', \tau.e_1, \epsilon, \varphi_1\rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \tag{9}$$

를 얻는다. 식 (9), (8)에 귀납 추론의 가정을 적용하면

$\langle s_1'.s_3, \tau.e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1\rangle \vdash C' \rightsquigarrow \langle s, e, \varphi\rangle$

를 얻게 되고, 여기에 규칙 frame를 적용하면

$\langle \tau.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1\rangle \vdash \texttt{frame}.C' \rightsquigarrow \langle s, e, \varphi\rangle$

를 얻게 된다.

- $C_1 = \texttt{cond}(C_t, C_f).C'$인 경우

귀납적 추론의 가정에 의해, $s_1 = b.s_1'$ 인 s_1'이 존재하고 식 (10)과 (11)이 성립한다

$$\langle b.s_1', e_1, \epsilon, \varphi_1\rangle \vdash \texttt{cond}(C_t, C_f).C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \tag{10}$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi'\rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi\rangle. \tag{11}$$

식 (10)과 규칙 cond에 의해서

$$\langle s_1', e_1, \epsilon, \epsilon\rangle \vdash C_t \rightsquigarrow \langle s_b, e_b, \varphi_{b_1}\rangle$$
$$\langle s_1', e_1, \epsilon, \epsilon\rangle \vdash C_f \rightsquigarrow \langle s_b, e_b, \varphi_{b_2}\rangle \tag{12}$$

$$\langle s_b, e_b, \epsilon, \varphi_{b_1} \cup \varphi_{b_2} \cup \varphi_1\rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \tag{13}$$

이다. 보조 정리 3을 식 (12)에 적용하면

$$\langle s_1'.s_3, e_1, \epsilon, \epsilon\rangle \vdash C_t \rightsquigarrow \langle s_b.s_3, e_b, \varphi_{b_1}\rangle$$
$$\langle s_1'.s_3, e_1, \epsilon, \epsilon\rangle \vdash C_f \rightsquigarrow \langle s_b.s_3, e_b, \varphi_{b_2}\rangle \tag{14}$$

를 얻게 된다. 식 (13), (11)에 귀납 추론의 가정을 적용하면

$$\langle s_b.s_3, e_b, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \cup \varphi_{b_1} \cup \varphi_{b_2}\rangle \vdash C' \rightsquigarrow \langle s, e, \varphi\rangle \tag{15}$$

를 얻는다. 식 (14), (15)에 규칙 cond를 다시 한번 적용하면

$\langle b.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1\rangle \vdash \texttt{cond}(C_t, C_f).C' \rightsquigarrow \langle s, e, \varphi\rangle$

를 얻는다. □

이제 타입 규칙의 안전성을 증명하기로 한다. 이 정리의 의미는 어떤 기계 수행 상태가 타입 규칙에 의해서 타입이 결정되었다면, 한 단계의 수행을 하고 난 후에도 같은 타입을 얻게 된다는 것이다. 이 경우 스택과 환경은 동일하지만 효과는 줄어들 수 있는데, 이것은 타입에 표시된 효과가 실제로 일어나지 않을 수 있기 때문이다.

정리 1 (안전성) $(S, H, E, C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o)$이고

$(S, H, E, C, K, \varphi) \Rightarrow \mathcal{M}$이라면, $\mathcal{M} \rightsquigarrow (s_o, e_o, \varphi_o')$이고 $\varphi_o' \sqsubseteq \varphi_o$이다.

증명. 변환 관계식($\Rightarrow$)의 각 경우에 대해 증명해야 한다. 대부분의 증명이 동일한 형태이기 때문에, 몇 가지 경우만을 다루고 나머지는 부록에서 증명하기로 한다.

- $(v.S, H, E, \texttt{frame}.C, K, \varphi) \Rightarrow (S, H, v.E, C, K, \varphi)$인 경우

가정에 의해서

이므로

$(v.S, H, E, \texttt{frame}.C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o)$

인 τ, s, e, k가 존재하고

$\langle \tau.s, e, k, \varphi\rangle \vdash \texttt{frame}.C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$

이다. 그러면 규칙 frame에 의해서

$\langle s, \tau.e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$

이고 $H \models v.E : \tau.e$ 이므로

$(S, H, v.E, C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o) \qquad \varphi_o \sqsubseteq \varphi_o$

이다.

- $(\texttt{true}.S, H, E, \texttt{cond}(C_1, C_2).C, K, \varphi) \Rightarrow (S, H, E, C_1.C, K, \varphi)$인 경우

같은 방법으로 $H \models S : s$, $H \models E : e$, $H \models K : k$인 s, e, k 가 존재하고

$\langle b.s, e, k, \varphi\rangle \vdash \texttt{cond}(C_1, C_2).C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$

이다. 그러면 규칙 cond에 의해서

$$\langle s, e, \epsilon, \epsilon\rangle \vdash C_1 \rightsquigarrow \langle s_b, e_b, \varphi_{b_1}\rangle$$
$$\langle s, e, \epsilon, \epsilon\rangle \vdash C_2 \rightsquigarrow \langle s_b, e_b, \varphi_{b_2}\rangle \tag{16}$$

$$\langle s_b, e_b, k, \varphi_{b_1} \cup \varphi_{b_2} \cup \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle \tag{17}$$

이다. 보조 정리 4를 식 (17)에 적용하면

$$\langle s_b, e_b, k, \varphi_{b_1} \cup \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o'\rangle \qquad \varphi_o' \sqsubseteq \varphi_o \tag{18}$$

인 φ_o'이 존재한다. 또한 보조 정리 3을 식 (16)에 적용하면

$$\langle s, e, \epsilon, \varphi\rangle \vdash C_1 \rightsquigarrow \langle s_b, e_b, \varphi_{b_1} \cup \varphi\rangle \tag{19}$$

이다. 그러면 식 (19)와 (18)에 의해서 다음을 쉽게 보일 수 있다.

$H \models v : \tau$, $H \models S : s$, $H \models E : e$, $H \models K : k$

- $(v.\texttt{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, \texttt{app}.C, K, \varphi) \Rightarrow$

$(S, H, v.E_f, C_f, (E, C).K, \varphi)$ 인 경우

같은 방법으로 $H \models v : \tau_1$, $H \models \mathrm{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \tau_1 \xrightarrow{\varphi_f} \tau_2$, $H \models S : s$, $H \models E : e$, $H \models K : k$ 인 τ_1, s, e, k가 존재하고

$$\langle \tau_1.\tau_1 \xrightarrow{\varphi_f} \tau_2.s, e, k, \varphi \rangle \vdash \mathrm{app}.C \rightsquigarrow \langle s_o, e_o, \varphi_o \rangle \quad (20)$$

이다. $H \models \mathrm{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \tau_1 \xrightarrow{\varphi_f} \tau_2$이므로 $H \models E_f : e_f$이고

$$\langle \epsilon, \tau_1.e_f, \epsilon, \epsilon \rangle \vdash C_f \rightsquigarrow \langle \tau_2, \tau_1.e_f, \varphi'_f \rangle \quad \varphi'_f \sqsubseteq \varphi_f \quad (21)$$

이다. 식 (20)에 규칙 app를 적용하면

$\langle \tau_2.s, e, k, \varphi \cup \varphi_f \rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o \rangle$

이고, 이것과 보조 정리 4에 의해서 다음을 성립하는 φ'_o이 존재한다.

$$\langle \tau_2.s, e, k, \varphi \cup \varphi'_f \rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi'_o \rangle \quad \varphi'_o \sqsubseteq \varphi_o. \quad (22)$$

식 (21)과 (22)에 보조 정리 5를 적용하면

$\langle s, \tau_1.e_f, (e, C).k, \varphi \rangle \vdash C_f \rightsquigarrow \langle s_o, e_o, \varphi'_o \rangle$

이고, 이는 $(S, H, v.E_f, C_f, (E, C).K, \varphi) \rightsquigarrow (s_o, e_o, \varphi'_o)$와 $\varphi'_o \sqsubseteq \varphi_o$를 의미한다. □

이제 "*타입이 결정되는 기계 수행 상태는 수행 중에 잘못된 값을 만들어 내지 않는다*"는 성질을 증명하기로 한다. 정리 2가 이 성질을 설명한다. 한 가지 주목할 것은 오류 상태에 대해서는 타입이 결정되지 않는다는 것이다. 주어진 기계 상태에 대해 적용 가능한 타입 유추 과정은 정리 1의 증명에서 모두 사용되었으므로, 오류 상태로 갈 수 있는 기계 수행 상태에 대해서는 타입이 유추될 수 없다.

```
λinitial.
   let counter = new initial in
       λinc. (counter := !counter + inc;
              !counter)

==> etySECK으로 번역

fn (∀ρ.
      fetch (0); ref ρ; frame;
      fn (fetch (1); fetch (1); get; fetch (0);
          add; set; frame; deframe; fetch (1);
          get
        : int --read(ρ),write(ρ)--> int); deframe
     :int --init(ρ)--> (int --init(ρ),write(ρ)--> int))
;;

==> 타입 검증의 결과

: (∀ρ.int --init(ρ)--> (int --read(ρ),write(ρ)--> int), nil, nil)

==> 실제 수행 결과

(Λρ.fn (fetch (0); ref ρ; frame;
        fn (fetch (1); fetch (1); get; fetch (0);
            add; set; frame; deframe; fetch (1);
            get
          : int --read(ρ),write(ρ)--> int); deframe
       :int --init(ρ)--> (int --init(ρ),write(ρ)--> int)),
nil, nil)
```

그림 7 예제 프로그램

정리 2 (타입을 갖는 프로그램의 성질) 주어진 프로그램 C에 대하여 $(nil, \phi, nil, C, nil, \phi) \rightsquigarrow (s_o, e_o, \varphi_o)$이고 $(nil, \phi, nil, C, nil, \phi) \stackrel{*}{\Rightarrow} \mathcal{M}$이라면, $\mathcal{M} \rightsquigarrow (s_o, e_o, \varphi'_o)$이고 $\varphi'_o \sqsubseteq \varphi_o$이다.

증명. 변환 관계식($\Rightarrow$)가 쓰인 횟수에 대해 귀납적으로 증명해야 한다. 정리 1을 사용하여 쉽게 증명할 수 있다.

4. 구현 및 토론

타입을 유추하는 규칙이 확정적(deterministic)이기 때문에 타입 검증을 쉽게 구현할 수 있게 된다. 이 장에서는 두 개의 예를 간단히 살펴보고, 2장에서 제시한 다른 연구들과의 차이점을 언급하기로 한다.

4.1 타입 검증의 구현

그림 4와 5의 타입 규칙이 확정적이기 때문에, 타입을 검증하는 알고리즘은 타입 규칙 그대로 구현할 수 있게 된다.

```
fn (∀ρ.
      fetch (0); ref ρ; frame;
      fn (fetch (1); fetch (1); get; fetch (0);
          add; set; frame; deframe; fetch (1);
          get
        : int --read(ρ),write(ρ)--> int); deframe
     :int --init(ρ)--> (int --init(ρ)--> int))
;;

==> 타입 검증의 결과

Type Error: Function return type mismatch
(nil,
 nil,
 nil,
 fn (∀ρ.
     fetch (0); ref ρ; frame;
     fn (fetch (1); fetch (1); get; fetch (0);
         add; set; frame; deframe; fetch (1);
         get
       : int --read(ρ),write(ρ)--> int); deframe
    :int --init(ρ)--> (int --init(ρ)--> int)),
 nil)
DECLARED : int --write(ρ)--> int,
ACTUALLY : int --read(ρ),write(ρ)--> int,
```

그림 8 고쳐진 예제 프로그램

Talpin의 논문[21]에서 쓰인 예제를 대상으로 한 타입 검증 결과가 그림 7에 있다. 주어진 코드의 타입과 효과가 올바르게 표시되어 있는지를 검증할 수 있고, 실제로 수행이 끝난 후의 기계 상태 타입과 타입 검증 때 얻은 타입이 같다는 것을 알 수 있다.

프로그램의 타입과 효과가 올바르지 않게 표시되어 있도록 수정한 예제가 그림 8에 있다. 바깥 함수의 결과 타입(이 타입 자체가 함수 타입)이 타입 검증 프로그램

이 기대하는 타입이 아니기 때문에, 타입 검증이 실패하게 된다. 이것은 효과 분석이 잘못 표시되어 있어서 실제로 있어야 할 효과 분석 자료가 없기 때문에 발생하는 것이다.

작은 ML(core part)에서 etySECK으로의 변형은 쉽게 이루어질 수 있는데 자세한 내용은 생략하기로 한다. 핵심 방법은 let에 의해 생기는 복합형 타입의 값이 사용될 때마다, tapp 명령어로 구체화시키는 것이다. 이 때 tapp가 사용하는 인자는 복합형 타입과 그 값이 실제로 쓰이는 위치에서의 타입을 동일화(unification)하여 구할 수 있다.

4.2 토론

증명 전달 코드(PCC)는 코드의 증명을 만들어 내기가 어렵다는 문제점이 있다. 이 문제점을 해결하기 위해서 우리는 잘 정의된 타입 시스템을 이용하였다. 우리의 방법은 프로그램을 순차적으로 읽어 가면서 검증만 하면 되기 때문에 프로그램의 길이에 비례하는 시간이 걸리게 되는데, 이것은 코드 소비자에게 큰 부담이 되지 않는다.

자바의 가상 기계 상에서의 검증은 코드가 수행 중에 잘못되지 않는가에 관한 검증뿐이며 특별히 코드의 안전성에 관한 정보가 전달되어 오는 것은 아니다. 또한 검증 규칙의 안전성에 관한 증명도 아직까지는 알려진 것이 없다. 반면에 우리의 방법은 안전성 증명에 필요한 타입 정보가 프로그램에 표시되서 전달되어 오고, 잘 정의된 타입 이론에 기반해서 안전성을 증명하는 것이다.

타입이 붙은 기계어(TAL)의 연구는 본 논문의 접근 방법과 비슷하지만, 그들의 연구에서 사용하는 언어는 코드가 다음 할 일을 넘겨받는 형태(CPS : Continuation Passing Style)로 구성된 프로그램을 가정하기 때문에 결과적으로 사용할 수 있는 최적화 기법에 제약이 따르게 된다. 또한 단순히 코드의 안전성만을 보장해 줄뿐이다. 우리의 접근 방법은 코드의 안전성 검증 외에도 컴파일러가 코드의 수행을 빠르게 할 수 있도록 여러 가지 프로그램의 성질을 전달할 수 있다. 본 논문에서는 그 예로서 메모리 반응에 관한 정보를 전달했는데, 다른 정보로의 확장이 어렵지 않게 이루어질 수 있다.

5. 결 론

신뢰할 수 없는 외부로부터 전달받은 코드를 검증하는 일이 점점 더 중요한 문제가 되고 있다. 검증 시스템은 코드의 안전성을 보장해 줌과 동시에 그 코드의 최적화를 가능하게 하는 성질도 검증할 수 있어야 한다.

본 논문에서는 타입과 효과가 코드에 표시된 etySECK을 설계하고, 복합형 타입 시스템을 효과 분석 방식으로 확장한 효과 타입 시스템을 이용하여 이 언어의 안전성을 검증할 수 있도록 하였다. 본 논문에서 제시한 방법은 코드의 안전성을 잘 정의된 타입 이론에 기반하여 검증할 수 있다는 이점이 있다. 또한 메모리 반응이라는 프로그램 성질을 검증할 수 있도록 설계하였고, 프로그램의 다른 성질을 검증하는 데에도 사용될 수 있다. 이 방법의 문제점은 프로그램의 성질이 코드에 표시되어서 전달될 때 코드의 크기가 커질 수 있다는 것이다.

참 고 문 헌

[1] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[2] G. Morrisett, D. Walker, K. Cray, and N. Glew. From system F to typed assembly language (extended version). Technical report, Cornell University, Nov. 1997.

[3] G. Necula. Proof-carrying code. In *Twenty-Fourth ACM Symposium on Principles of Programming Languages*, 1997.

[4] Xavier Leroy. An overview of Types in Compilation. In *Lecture Notes in Computer Science*, volume 1473, pages 1-8. Springer-Verlag, mar 1998.

[5] Dexter Kozen. Language-based security. Technical report, Cornell University, 1999.

[6] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, 1996.

[7] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification, Second Edition*. Addison-Wesley, 1999.

[8] Stephen N. Freund and John C. Mitchell. A Type System for Object Initialization In the Java Bytecode Language. In *ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications*, pages 310-327, Oct. 1998.

[9] Greg Morrisett, Karl Crary, Neal Glew, Dan Grossman, Richard Samuels, Frederick Smith, David Walker, Stephanie Weirich, and Steve Zdancewic. TALx86: A realistic typed assembly language. In *1999 ACM SIGPLAN Workshop on Compiler Support for System Software*, pages 25-35, 1999.

[10] Greg Morrisett, Karl Crary, Neal Glew, and David Walker. Stack-based typed assembly language. In *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[11] Hongwei Xi and Robert Harper. A Dependently Typed Assembly Language. Unpublished, 5 1999.
[12] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348-375, 1978.
[13] John M. Lucassen and David K. Gifford. Polymorphic effect systems. In *Proceedings of The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 47-57, 1988.
[14] Luis Damas and Robin Milner. Principal type-scheme for functional programs. In *Proceedings of The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1982.
[15] Mads Tofte. *Operational Semantics and Polymorphic Type Inference.* (tech report ecs-lfcs-88-54), University of Edinburgh, May 1988.
[16] Xavier Leroy. *Polymorphic Typing of an Algorithmic Language.* PhD thesis, University Paris VII, October 1992.
[17] Oukseh Lee and Kwangkeun Yi. Proofs about a Folklore Let-polymorphic Type Inference Algorithm. *ACM Transactions on Programming Languages and Systems*, 20(4):707-723, 1998.
[18] Pierre Jouvelot and David K. Gifford. Reasoning about continuations with control effects. In *Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation*, 1989.
[19] François Pessaux and Xavier Leroy. Type-based analysis of uncaught exceptions. In *ACM Symposium on Principles of Programming Languages*, pages 276-290, January 1999.
[20] Pierre Jouvelot and David K. Gifford. Algebraic reconstruction of types and effects. In *Proceedings of The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 303-310, 1991.
[21] Jean-Piere Talpin and Pierre Jouvelot. Polymorphic type, region and effect inference. *Journal of Functional Programming*, 2(3): 245-271, July 1992.
[22] Mads Tofte and Jean-Pierre Talpin. Implementation of the typed call-by-value λ - calculus using a stack of regions. In *Proceedings of The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 188-201, January 1994.
[23] Mads Tofte and Jean-Pierre Talpin. Region-based memory management. *Information and Computation*, 132(2):109-176, 1997.
[24] Jean-Piere Talpin and Pierre Jouvelot. Compiling FX on the CM-2. In *Lecture Notes in Computer Science*, volume 724, pages 87-98. Springer-Verlag, proceedings of the 3rd workshop on semantics analysis edition, September 1993.
[25] P. J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308-320, January 1964.
[26] P. J. Landin. A correspondence between ALGOL 60 and Church's lambda-notation: Part I. *Communications of the ACM*, 8(2):89-101, February 1965.
[27] P. J. Landin. A correspondence between ALGOL 60 and Church's lambda-notation: Part II. *Communications of the ACM*, 8(3):158-165, March 1965.
[28] Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125-159, 1975.
[29] Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. Technical Report TR91-160, 1992.

A 부록

[*보조 정리 2의 증명*]

$H \models \Lambda(\vec{\alpha},\vec{\rho}).\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \forall(\vec{\alpha},\vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2$로부터 식 (23)과

$$H \models E_f : e \tag{23}$$

$H \models \mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \tau_1 \xrightarrow{\varphi_f} \tau_2 \quad \alpha_i, \rho_i \notin \mathrm{FTV}(e)$를 얻게 되는데, 이것은 다시

$$\langle \epsilon, \tau_1.e, \epsilon, \epsilon\rangle \vdash C_f \rightsquigarrow \langle \tau_2, \tau_1.e, \varphi'_f\rangle \qquad \varphi'_f \sqsubseteq \varphi_f \tag{24}$$

를 의미한다. $\mathcal{S} = [\tau_i/\alpha_i, r_i/\rho_i]$라 하자. $\alpha_i, \rho_i \notin \mathrm{FTV}(e)$이므로 $\mathcal{S}e = e$이고

$$H \models \mathcal{S}E_f : e$$

이다. 식 (24)와 보조 정리 1에 의해서

$$\langle \epsilon, \mathcal{S}(\tau_1.e), \epsilon, \epsilon\rangle \vdash \mathcal{S}C_f \rightsquigarrow \langle \mathcal{S}\tau_2, \mathcal{S}(\tau_1.e), \mathcal{S}\varphi'_f\rangle \qquad \mathcal{S}\varphi'_f \sqsubseteq \mathcal{S}\varphi_f \tag{25}$$

이다. 즉, $H \models \mathsf{fn}(\mathcal{S}(\tau_1 \xrightarrow{\varphi_f} \tau_2), \mathcal{S}C_f, \mathcal{S}E_f) : \mathcal{S}(\tau_1 \xrightarrow{\varphi_f} \tau_2)$이므로

$$H \models \mathcal{S}\ \mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \mathcal{S}(\tau_1 \xrightarrow{\varphi_f} \tau_2)$$

이다. □

[*보조 정리 5의 증명*]

- $C_1 = nil$인 경우

가정에 의해서 $s_2 = s_1, e' = e_1, \varphi_2 = \varphi_1$이고

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi \rangle$$

이다. 그러면 규칙 nilk에 의해서

$$\langle s_1.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash nil \rightsquigarrow \langle s, e, \varphi \rangle$$

이다.

- $C_1 = \texttt{deframe}.C'$인 경우

가정에 의해서 $e_1 = \tau.e_1'$이고

$$\langle s_1, \tau.e_1', \epsilon, \varphi_1 \rangle \vdash \texttt{deframe}.C' \rightsquigarrow \langle s_2, e', \varphi_2 \rangle \quad (27)$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi \rangle \quad (28)$$

이다. 식 (27)과 규칙 deframe에 의해서

$$\langle s_1, e_1', \epsilon, \varphi_1 \rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2 \rangle \quad (29)$$

이다. 식 (28)과 (29), 귀납적 추론의 가정에 의해서

$$\langle s_1.s_3, e_1', (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash C' \rightsquigarrow \langle s, e, \varphi \rangle$$

이고, 이것과 규칙 deframe에 의해서

$$\langle s_1.s_3, \tau.e_1', (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash \texttt{deframe}.C' \rightsquigarrow \langle s, e, \varphi \rangle$$

이다.

- $C_1 = \texttt{fetch}(n).C'$인 경우

가정에 의해서 $e_1 = \tau_0.\cdots.\tau_n.e_1'$이고

$$\langle s_1, \tau_0.\cdots.\tau_n.e_1', \epsilon, \varphi_1 \rangle \vdash \texttt{fetch}(n).C' \rightsquigarrow \langle s_2, e', \varphi_2 \rangle \quad (30)$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi \rangle \quad (31)$$

이다. 식 (30)과 규칙 fetch에 의해서

$$\langle \tau_n.s_1, \tau_0.\cdots.\tau_n.e_1', \epsilon, \varphi_1 \rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2 \rangle \quad (32)$$

이다. 식 (32)와 (31), 귀납 추론의 가정에 의해서

$$\langle \tau_n.s_1.s_3, \tau_0.\cdots.\tau_n.e_1', (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash C' \rightsquigarrow \langle s, e, \varphi \rangle$$

이고, 이것과 규칙 fetch에 의해서

$$\langle s_1.s_3, \tau_0.\cdots.\tau_n.e_1', (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash \texttt{fetch}(n).C' \rightsquigarrow \langle s, e, \varphi \rangle$$

이다.

- $C_1 = \texttt{quote}(p).C'$인 경우

$\tau = \texttt{type_of}(p)$라 하면 가정에 의해서

$$\langle s_1, e_1, \epsilon, \varphi_1 \rangle \vdash \texttt{quote}(p).C' \rightsquigarrow \langle s_2, e', \varphi_2 \rangle \quad (33)$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi \rangle \quad (34)$$

이다. 식 (33)과 규칙 quote에 의해서

$$\langle \tau.s_1, e_1, \epsilon, \varphi_1 \rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2 \rangle \quad (35)$$

이다. 식 (35)와 (34), 귀납 추론의 가정에 의해서

$$\langle \tau.s_1.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash C' \rightsquigarrow \langle s, e, \varphi \rangle$$

이고, 이것과 규칙 quote에 의해서

$$\langle s_1.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash \texttt{quote}(p).C' \rightsquigarrow \langle s, e, \varphi \rangle$$

이다.

- $C_1 = \texttt{add}.C'$인 경우

가정에 의해서 $s_1 = i.i.s_1'$이고,

$$\langle i.i.s_1', e_1, \epsilon, \varphi_1 \rangle \vdash \texttt{add}.C' \rightsquigarrow \langle s_2, e', \varphi_2 \rangle \quad (36)$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi \rangle \quad (37)$$

이다. 식 (36)과 규칙 add에 의해서

$$\langle i.s_1', e_1, \epsilon, \varphi_1 \rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2 \rangle \quad (38)$$

이다. 식 (38)과 (37), 귀납 추론의 가정에 의해서

$$\langle i.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash C' \rightsquigarrow \langle s, e, \varphi \rangle$$

이고, 이것과 규칙 add에 의해서

$$\langle i.i.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash \texttt{add}.C' \rightsquigarrow \langle s, e, \varphi \rangle$$

이다.

- $C_1 = \texttt{fn}(\forall \vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C'$인 경우

가정에 의해서

$$\langle s_1, e_1, \epsilon, \varphi_1 \rangle \vdash \texttt{fn}(\forall \vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C' \rightsquigarrow \langle s_2, e', \varphi_2 \rangle \quad (39)$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi' \rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi \rangle \quad (40)$$

이다. 식 (39)와 규칙 fn에 의해서

$$\langle \epsilon, \tau_1.e_1, \epsilon, \epsilon \rangle \vdash C_f \rightsquigarrow \langle \tau_2, \tau_1.e_1, \varphi_f' \rangle$$
$$\varphi_f' \sqsubseteq \varphi_f \quad \alpha_i, \rho_i \notin \mathrm{FTV}(e_1) \quad (41)$$

$$\langle (\forall(\vec{\alpha}, \vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2).s_1, e_1, \epsilon, \varphi_1 \rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2 \rangle \quad (42)$$

이다. 식 (42)와 (40), 귀납 추론의 가정에 의해서

$$\langle (\forall(\vec{\alpha}, \vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2).s_1.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash C' \rightsquigarrow \langle s, e, \varphi \rangle$$

이고, 이것과 식 (41), 규칙 fn에 의해서

$$\langle s_1.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \rangle \vdash \texttt{fn}(\forall \vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C' \rightsquigarrow \langle s, e, \varphi \rangle$$

이다.

• $C_1 = \mathtt{rfn}(\forall\vec{\alpha},\vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C'$인 경우

이 경우의 증명은 $C_1 = \mathtt{fn}(\forall\vec{\alpha},\vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C'$인 경우와 동일하다.

• $C_1 = \mathtt{app}.C'$인 경우

가정에 의해서 $s_1 = \tau_1.\tau_1 \xrightarrow{\varphi_f} \tau_2.s_1'$이고

$$\langle \tau_1.\tau_1 \xrightarrow{\varphi_f} \tau_2.s_1', e_1, \epsilon, \varphi_1\rangle \vdash \mathtt{app}.C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \quad (43)$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi'\rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi\rangle \quad (44)$$

이다. 식 (43)과 규칙 app에 의해서

$$\langle \tau_2.s_1', e_1, \epsilon, \varphi_1 \cup \varphi_f\rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \quad (45)$$

이다. 식 (44)와 (45), 귀납 추론의 가정에 의해서

$$\langle \tau_2.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \cup \varphi_f\rangle \vdash C' \rightsquigarrow \langle s, e, \varphi\rangle$$

이고, 이것과 규칙 app에 의해서

$$\langle \tau_1.\tau_1 \xrightarrow{\varphi_f} \tau_2.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1\rangle \vdash \mathtt{app}.C' \rightsquigarrow \langle s, e, \varphi\rangle$$

이다.

• $C_1 = \mathtt{tapp}(\vec{\tau},\vec{r}).C'$인 경우

가정에 의해서 $s_1 = (\forall(\vec{\alpha},\vec{\rho}).\tau).s_1'$이고

$$\langle (\forall(\vec{\alpha},\vec{\rho}).\tau).s_1', e_1, \epsilon, \varphi_1\rangle \vdash \mathtt{tapp}(\vec{\tau},\vec{r}).C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \quad (46)$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi'\rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi\rangle \quad (47)$$

이다. $\mathcal{S} = [\tau_i/\alpha_i, r_i/\rho_i]$라 하면 규칙 tapp에 의해서

$$\langle \mathcal{S}\tau.s_1', e_1, \epsilon, \varphi_1\rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \quad (48)$$

이다. 식 (47)과 (48), 귀납 추론의 가정에 의해서

$$\langle \mathcal{S}\tau.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1\rangle \vdash C' \rightsquigarrow \langle s, e, \varphi\rangle$$

이고, 이것과 규칙 tapp에 의해서

$$\langle (\forall(\vec{\alpha},\vec{\rho}).\tau).s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1\rangle \vdash \mathtt{tapp}(\vec{\tau},\vec{r}).C' \rightsquigarrow \langle s, e, \varphi\rangle$$

이다.

• $C_1 = \mathtt{ref}(\rho).C'$인 경우

가정에 의해서 $s_1 = \tau.s_1'$이고

$$\langle \tau.s_1', e_1, \epsilon, \varphi_1\rangle \vdash \mathtt{ref}(\rho).C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \quad (49)$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi'\rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi\rangle \quad (50)$$

이다. 식 (49)와 규칙 ref에 의해서

$$\langle ref_\rho\tau.s_1', e_1, \epsilon, \varphi_1 \cup \{init(\rho)\}\rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \quad (51)$$

이다. 식 (50)과 (51), 귀납 추론의 가정에 의해서

$$\langle ref_\rho\tau.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \cup \{init(\rho)\}\rangle \vdash C' \rightsquigarrow \langle s, e, \varphi\rangle$$

이고, 이것과 규칙 ref에 의해서

$$\langle \tau.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1\rangle \vdash \mathtt{ref}(\rho).C' \rightsquigarrow \langle s, e, \varphi\rangle$$

이다.

• $C_1 = \mathtt{get}.C'$인 경우

가정에 의해서 $s_1 = ref_\rho\tau.s_1'$이고

$$\langle ref_\rho\tau.s_1', e_1, \epsilon, \varphi_1\rangle \vdash \mathtt{get}.C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \quad (52)$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi'\rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi\rangle \quad (53)$$

이다. 식 (52)와 규칙 get에 의해서

$$\langle \tau.s_1', e_1, \epsilon, \varphi_1 \cup \{read(\rho)\}\rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \quad (54)$$

이다. 식 (53)과 (54), 귀납 추론의 가정에 의해서

$$\langle \tau.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \cup \{read(\rho)\}\rangle \vdash C' \rightsquigarrow \langle s, e, \varphi\rangle$$

이고, 이것과 규칙 get에 의해서

$$\langle ref_\rho\tau.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1\rangle \vdash \mathtt{get}.C' \rightsquigarrow \langle s, e, \varphi\rangle$$

이다.

• $C_1 = \mathtt{set}.C'$인 경우

가정에 의해서 $s_1 = \tau.ref_\rho\tau.s_1'$이고

$$\langle \tau.ref_\rho\tau.s_1', e_1, \epsilon, \varphi_1\rangle \vdash \mathtt{set}.C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \quad (55)$$

$$\langle s_2.s_3, e_2, k_2, \varphi_2 \cup \varphi'\rangle \vdash C_2 \rightsquigarrow \langle s, e, \varphi\rangle$$

이다. 식 (55)와 규칙 set에 의해서 (56)

$$\langle unit.s_1', e_1, \epsilon, \varphi_1 \cup \{write(\rho)\}\rangle \vdash C' \rightsquigarrow \langle s_2, e', \varphi_2\rangle \quad (57)$$

이다. 식 (56)과 (57), 귀납 추론의 가정에 의해서

$$\langle unit.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1 \cup \{write(\rho)\}\rangle \vdash C' \rightsquigarrow \langle s, e, \varphi\rangle$$

이고, 이것과 규칙 set에 의해서

$$\langle \tau.ref_\rho\tau.s_1'.s_3, e_1, (e_2, C_2).k_2, \varphi' \cup \varphi_1\rangle \vdash \mathtt{set}.C' \rightsquigarrow \langle s, e, \varphi\rangle$$

이다. □

[정리 1의 증명]

• $(S, H, E, nil, (E', C).K, \varphi) \Rightarrow (S, H, E', C, K, \varphi)$인 경우

가정에 의해서 $H \models S : s$, $H \models E : e$, $H \models K : k$, $H \models E' : e'$, $H \models (E', C).K : (e', C).k$인 s, e, k가 존

재하고

$$\langle s, e, (e', C).k, \varphi\rangle \vdash nil \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$$

이다. 규칙 nilk에 의해서 $\langle s, e', k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$ 이므로 $(S, H, E', C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o)$이다.

- $(S, H, v.E, \texttt{deframe}.C, K, \varphi) \Rightarrow (S, H, E, C, K, \varphi)$인 경우

$H \models S : s, H \models E : e, H \models v : \tau, H \models K : k$인 s, e, τ, k가 존재하고

$$\langle s, \tau.e, k, \varphi\rangle \vdash \texttt{deframe}.C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$$

이다. 규칙 deframe에 의해서

$$\langle s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$$

이므로 $(S, H, E, C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o)$이다.

- $(S, H, v_0.\cdots.v_n.E, \texttt{fetch}(n).C, K, \varphi) \Rightarrow$

$(v_n.S, H, v_0.\cdots.v_n.E, C, K, \varphi)$ 인 경우

$H \models S : s, H \models E : e$, $H \models v_i : \tau_i$ $(i = 0, \cdots, n)$, $H \models K : k$ 인 s, e, τ_i, k가 존재하고

$$\langle s, \tau_0.\cdots.\tau_n.e, k, \varphi\rangle \vdash \texttt{fetch}(n).C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$$

이다. 규칙 fetch에 의해서

$$\langle \tau_n.s, \tau_0.\cdots.\tau_n.e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle,$$

이므로 $(v_n.S, H, E, C, K, \varphi) \rightsquigarrow (s_o, v_o, \varphi_o)$이다.

- $(S, H, E, \texttt{quote}(p).C, K, \varphi) \Rightarrow (p.S, H, E, C, K, \varphi)$인 경우

$H \models S : s, H \models E : e, H \models K : k$ 인 s, e, k가 존재하고

$$\langle s, e, k, \varphi\rangle \vdash \texttt{quote}(p).C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$$

이다. $\tau = \texttt{type_of}(p)$라 하면 규칙 quote에 의해서

$$\langle \tau.s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$$

이므로, $(p.S, H, E, C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o)$이다.

- $(i_2.i_1.S, H, E, \texttt{add}.C, K, \varphi) \Rightarrow (i_1 + i_2.S, H, E, C, K, \varphi)$인 경우

$H \models S : s, H \models i_j : i$ $(j = 1, 2), H \models E : e$, $H \models K : k$인 s, e, k가 존재하고

$$\langle i.i.s, e, k, \varphi\rangle \vdash \texttt{add}.C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$$

이다. 규칙 add에 의해서

$$\langle i.s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$$

이고 $H \models i_i + i_2 : i$이므로

$$(i_i + i_2.S, H, E, C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o)$$

이다.

- $(\texttt{false}.S, H, E, \texttt{cond}(C_1, C_2).C, K, \varphi) \Rightarrow (S, H, E, C_1.C, K, \varphi)$인 경우

$(\texttt{true}.S, H, E, \texttt{cond}(C_1, C_2).C, K, \varphi) \Rightarrow (S, H, E, C_1.C, K, \varphi)$인 경우의 증명과 동일하다.

- $(S, H, E, \texttt{fn}(\forall\vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C, K, \varphi) \Rightarrow$

$(\Lambda(\vec{\alpha}, \vec{\rho}).\texttt{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E).S, H, E, C, K, \varphi)$인 경우

$H \models S : s, H \models E : e, H \models K : k$ 인 s, e, k가 존재하고

$$\langle s, e, k, \varphi\rangle \vdash \texttt{fn}(\forall\vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle$$

이다. 규칙 fn에 의해서

$$\langle \epsilon, \tau_1.e, \epsilon, \epsilon\rangle \vdash C_f \rightsquigarrow \langle \tau_2, \tau_1.e, \varphi'_f\rangle$$
$$\varphi'_f \sqsubseteq \varphi_f \quad \alpha_i, \rho_i \notin \mathrm{FTV}(e) \tag{58}$$

$$\langle(\forall(\vec{\alpha}, \vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2).s, e, k, \varphi\rangle \vdash C \rightsquigarrow \langle s_o, e_o, \varphi_o\rangle \tag{59}$$

이다. 식 (58)과 $H \models E : e$ 에 의해서

$$H \models \texttt{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E) : \tau_1 \xrightarrow{\varphi_f} \tau_2 \tag{60}$$

이고, 식 (60)과 (58), $H \models E : e$ 에 의해서

$$H \models \Lambda(\vec{\alpha}, \vec{\rho}).\texttt{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E) : \forall(\vec{\alpha}, \vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2 \tag{61}$$

이다. 마지막으로 식 (59)와 (61)에 의해서

$$(\Lambda(\vec{\alpha}, \vec{\rho}).\texttt{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E).S, H, E, C, K, \varphi) \rightsquigarrow (s_o, e_o, \varphi_o)$$

이다.

- $(S, H, E, \texttt{rfn}(\forall\vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C, K, \varphi) \Rightarrow$

$(\Lambda(\vec{\alpha}, \vec{\rho}).\texttt{rfn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E).S, H, E, C, K, \varphi)$인 경우

코드 부분이 $\texttt{fn}(\forall\vec{\alpha}, \vec{\rho}.C_f : \tau_1 \xrightarrow{\varphi_f} \tau_2).C$인 경우의 증명과 동일하다.

- $(v.\texttt{rfn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, \texttt{app}.C, K, \varphi) \Rightarrow$

$(S, H, v.\texttt{rfn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).E_f, C_f, (E, C).K, \varphi)$인 경우

$$(v.\texttt{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, \texttt{app}.C, K, \varphi) \Rightarrow (S, H, v.E_f, C_f, (E, C).K, \varphi)$$

인 경우의 증명과 동일하다.

- $(\Lambda(\vec{\alpha}, \vec{\rho}).\texttt{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, \texttt{tapp}(\vec{\tau}, \vec{r}).C, K, \varphi) \Rightarrow$

$([\tau_i/\alpha_i, r_i/\rho_i]\texttt{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S, H, E, C, K, \varphi)$인 경우

$$H \models \Lambda(\vec{\alpha}, \vec{\rho}).\texttt{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \forall(\vec{\alpha}, \vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2, \tag{62}$$

$H \models S : s, H \models E : e, H \models K : k$ 인 s, e, k가 존

재하고

$\langle \forall(\vec{\alpha},\vec{\rho}).\tau_1 \xrightarrow{\varphi_f} \tau_2.s,e,k,\varphi\rangle \vdash \mathtt{tapp}(\vec{\tau},\vec{r}).C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$

이다. 규칙 tapp에 의해서

$\langle [\tau_i/\alpha_i, r_i/\rho_i](\tau_1 \xrightarrow{\varphi_f} \tau_2).s,e,k,\varphi\rangle \vdash C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$ (63)

이고, $\mathcal{S} = [\tau_i/\alpha_i, r_i/\rho_i]$ 라 하면 식 (62)와 보조 정리 2에 의해서

$H \models \mathcal{S}\ \mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f) : \mathcal{S}(\tau_1 \xrightarrow{\varphi_f} \tau_2)$ (64)

이다. 그러므로 식 (63)과 (64)에 의해서

$(\mathcal{S}\ \mathsf{fn}((\tau_1 \xrightarrow{\varphi_f} \tau_2), C_f, E_f).S,H,E,C,K,\varphi) \rightsquigarrow (s_o,e_o,\varphi_o)$

이다.

- $(\Lambda(\vec{\alpha},\vec{\rho}).\mathsf{rfn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S,H,E,\mathtt{tapp}(\vec{\tau},\vec{r}).C,K,\varphi) \Rightarrow$
  $([\tau_i/\alpha_i, r_i/\rho_i]\mathsf{rfn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S,H,E,C,K,\varphi)$인 경우

$(\Lambda(\vec{\alpha},\vec{\rho}).\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S,H,E,\mathtt{tapp}(\vec{\tau},\vec{r}).C,K,\varphi) \Rightarrow$
$([\tau_i/\alpha_i, r_i/\rho_i]\mathsf{fn}(\tau_1 \xrightarrow{\varphi_f} \tau_2, C_f, E_f).S,H,E,C,K,\varphi)$

인 경우의 증명과 동일하다.

- $(v.S,H,E,\mathtt{ref}(\rho).C,K,\varphi) \Rightarrow$
  $(l_\rho.S,H[l_\rho \rightarrow v],E,C,K,\varphi \cup \{init(\rho)\})$ $l_\rho \notin \mathrm{Dom(H)}$
  인 경우

$H \models v : \tau$, $H \models S : s$, $H \models E : e$, $H \models K : k$ 인 s, e, k가 존재해서

$\langle \tau.s,e,k,\varphi\rangle \vdash \mathtt{ref}(\rho).C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$

이고, 규칙 ref에 의해서

$\langle ref_\rho \tau.s,e,k,\varphi \cup \{init(\rho)\}\rangle \vdash C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$ (65)

이다. $H' = H[l_\rho \mapsto v]$ 라 하면 $H'(l_\rho) = v$ 이고, $H' \models v : \tau$이므로 $H \models v : \tau$이다. 따라서,

$H' \models l_\rho^{\bullet} : ref_\rho \tau,\ H' \models S : s,\ H' \models E : e,\ H' \models K : k$ (66)

이고, 식 (65)와 (66)에 의해서

$(l_\rho.S, H[l_\rho \rightarrow v], E, C, K, \varphi \cup \{init(\rho)\}) \rightsquigarrow (s_o,e_o,\varphi_o)$

이다.

- $(l_\rho.S,H,E,\mathtt{get}.C,K,\varphi) \Rightarrow (H(l_\rho).S,H,E,C,K,\varphi \cup \{read(\rho)\})$
  $l_\rho \in \mathrm{Dom(H)}$인 경우

$H \models l_\rho : ref_\rho \tau$, $H \models S : s$, $H \models E : e$, $H \models K : k$ 인 s, e, k가 존재해서

$\langle ref_\rho \tau.s,e,k,\varphi\rangle \vdash \mathtt{get}.C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$

이고, 규칙 get에 의해서

$\langle \tau.s,e,k,\varphi \cup \{read(\rho)\}\rangle \vdash C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$ (67)

이다. $H \models l_\rho : ref_\rho \tau$이고 $l_\rho \in \mathrm{Dom}(H)$이므로

$H \models H(l_\rho) : \tau$ (68)

이고, 식 (67)과 (68)에 의해서
$(H(l_\rho).S,H,E,C,K,\rho \cup \{read(\rho)\}) \rightsquigarrow (s_o,e_o,\varphi_o)$

이다.

- $(v.l_\rho.S,H,E,\mathtt{set}.C,K,\varphi) \Rightarrow$
  $((),S,H[l_\rho \rightarrow v],E,C,K,\varphi \cup \{write(\rho)\})$
  $l_\rho \in \mathrm{Dom(H)}$인 경우

$H \models v : \tau, H \models l_\rho : ref_\rho \tau, H \models S : s, H \models E : e\ H \models K : k$ 인 τ, s, e, k가 존재해서

$\langle \tau.ref_\rho \tau.s,e,k,\varphi\rangle \vdash \mathtt{set}.C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$

이고, 규칙 set에 의해서

$\langle unit.s,e,k,\varphi \cup \{write(\rho)\}\rangle \vdash C \rightsquigarrow \langle s_o,e_o,\varphi_o\rangle$ (69)

이다. $H' = H[l_\rho \mapsto v]$라 하면 $H' \models l_\rho : ref_\rho \tau$이므로

$H' \models () : unit,\ H' \models S : s,\ H' \models E : e,\ H' \models K : k$ (70)

이고, 식 (69)와 (70)에 의해서

$((),S,H[l_\rho \rightarrow v],E,C,K,\varphi \cup \{write(\rho)\}) \rightsquigarrow (s_o,e_o,\varphi_o)$

이다. □

정 재 윤
1998년 한국과학기술원 전산학과 학사(B.S.). 2000년 한국과학기술원 전산학과 석사(M.S.). 2000년 ~ 현재 한국 증권전산원 연구원.

류 석 영
정보과학회논문지 : 소프트웨어 및 응용 제 27 권 제 4 호 참조

이 광 근
정보과학회논문지 : 소프트웨어 및 응용 제 27 권 제 3 호 참조