제22회 한국정보처리학회 추계학술발표대회 논문집 제11권 제2호 (2004. 11)

CTOC에서 3-주소 코드를 위한 정적 타입 추론

김기태*, 유원희
*인하대학교 컴퓨터 정보공학과
e-mail:kimkitae@inha.ac.kr

Inference of Static Types for 3-Address Codes in CTOC

Ki-Tae Kim*, Weon-Hee Yoo
*Dept of Computer Science & Information, Inha University

요 약

자바 바이트 코드 명령어는 타입에 관한 정보를 포함하고 있다. 그러나 기본적으로 스택 기반으로 동작이 수행되기 때문에 지역 변수를 위한 명시적인 타입을 가지지 않는다. 하지만 프로그램의 최적화나 역컴파일을 위해서는 지역 변수의 타입을 아는 것이 중요하다. 본 논문은 스택을 사용하지 않는 3-주소 코드에서 지역 변수를 위한 정적 타입 추론을 구현한다. 이를 위해 본 논문에서는 SSA와 방향성 그래프를 적용한다.

1. 서론

최근 여러 분야에서 자바 바이트 코드는 중간 표현으로 사용된다[4]. ML, Scheme, Eiffel, C, C# 등 많은 언어들이 중간 표현으로 자바 바이트 코드를 사용한다. 다른 어떤 언어로 작성되어도 자바 바이트 코드 형태로 변환하면 자바 가상 기계가 설치된 어떤 플랫폼에서도 수행이 가능하기 때문이다. 자바 바이트 코드가 좋은 특징을 많이 가지지만 프로그램 분석이나 최적화 또는 프로그램을 이해하기 위해 사용하기에는 적절한 표현이 아니다. 또한 스택 코드이기 때문에 수행 속도가 느리다는 단점이 발생한다.

실행 속도를 빠르게 하기 위해서는 코드의 최적화가 요구되는데 스택 코드인 바이트 코드를 사용해서 최적화와 분석을 하는 것 보다는 컴파일러에서 사용하는 전통적인 3-주소 형태의 코드로 변환하는 것이 더욱 바람직하다[1,2,6]. 프로그램 이해를 위해서도 저수준의 바이트코드를 사용하는 것보다는 3-주소 형태의 코드를 사용하는 것이 바람직하다[3].

CTOC에서 최적화를 위해 바이트 코드를 3-주소 형태로 변환할 때 사용되는 모든 변수는 올바른 정적 타입이 주어져야 한다[6]. 이를 위해서는 기존의 타입 없는 3-주소 표현을 정적 단일 배정(Static Single Assignment)을 이용하여 모든 변수를 나눈 후 새로운 이름을 배정하고 올바른 정적 타입을 위해 타입 추론이 수행되어야 한다. 일반적으로 자바에서는 실행 시간에 타입 분석(run-time type analyzes)을 수행한다[5,6].

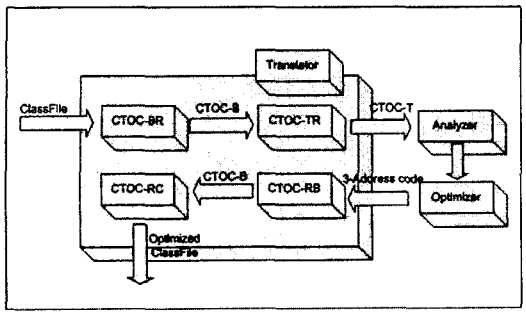
본 논문에서는 CTOCT라고 불리는 바이트 코드의 3-주소 표현에서 각 변수를 위해 정적 타입 추론의 문제를 해결한다. CTOCT는 자바의 최적화와 분석을 위한 프레임워크인 CTOC 컴파일러 프레임워크의 한 부분이다.

2. CTOC에서 CTOCB와 CTOCT

2.1 CTOC

[그림 1]은 CTOC의 전체 구성이다[9]. CTOC는 바이트 코드 형태인 .class 파일을 입력으로 받아 3-주소 코드로 변환한 후 최적화를 수행하고 다시 바이트 코드로 변환하여 자바 가상 기계에서 최적화된 코드를 수행하도록 구성되어 있다.

본 연구는 한국과학재단 목적기초연구(R05-2004-000-11694-0) 지원으로 수행되었음.



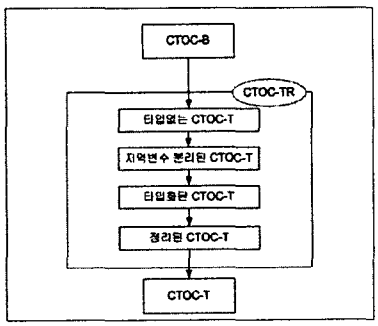
[그림 1] CTOC 구성도

2.2 CTOCB

CTOCB는 바이트 코드가 CTOC-BR을 통과한 후 생성된 코드이다. 이 코드는 기존의 자바 바이트 코드와 아주 유사하지만 3-주소 변환을 위해 명령어를 단순화시킨 형태의 코드이다.

2.3 CTOCT

CTOCT는 CTOCB가 CTOC-TR을 통과한 후 생성되는 코드이다. 기존의 스택 기반 코드인 바이트 코드가 3주소 형태로 변환되기 때문에 CTOC-TR에서 기존의 스택에서 사용되던 변수에 대해 명시적인 타입을 제공해야 한다.



[그림 2] CTOC-TR의 동작

[그림 2]는 CTOCT가 생성되는 과정에 관한 것이다.

3. CTOCT에서 타입 추론의 문제점

3.1 타입을 가지지 않는 CTOCT

CTOCT는 자바의 최적화와 분석을 위한 프레임워크인 CTOC 컴파일러 프레임워크에 의해서 생성되는 중간 코드이다. 이러한 동작은 CTOC-TR에 의해서 이루어지는데, CTOC-TR은 기존의 스택 기반 코드를 3주소 형태로 바꿔주는 변환기이다. 이 변환기를 통해서 처음 생성되는 코드는 타입이 결정되지 않은 CTOCT 코드이다. 타입이 없는 CTOCT의 예제는 [그림 3]과 같다.

```
public int runningExample (){
        untyped this, sum, z, $s0 , $s1, $s2 ;
        $s0 = 0.0;
        ...
        $s0= newB;
        ...
        return $s0;
}
```

[그림 3] untyped CTOCT

[그림 3]은 타입을 가지지 않는 CTOCT 코드의 모습이다. 바이트 코드로부터 지역 변수 배열에 있던 정보를 가져와 지역 변수 이름을 생성하고 타입에 관련된 부분은 untyped라고 임시의 공통된 타입을 선언한다. [그림 3]을 살펴보면 코드를 통해서 몇 가지 정보를 얻을 수 있다. 메소드의 시그니처를 보면 반환되는 값의 타입을 고정적으로 결정할 수 있다. 그리고 new 명령어를 통해서 타입에 관한 정보를 얻을 수 있다. 또한 this는 현재 클래의 이름을 통해서 타입을 결정할 수 있다. 하지만 지역 변수인 sum과 z는 명확한 타입을 가지지 않는다. 또한 코드 중에 untyped $S0에는 정확히 하나의 타입만을 제공해야 하기 때문에, 정적인 타입 추론이 수행되면 다음에 있는 형태의 타입이 할당되게 된다. 이렇게 정적인 타입이 할당되면 typed CTOCT라 부르고 타입을 가진 3-주소 형태의 코드이기 때문에 최적화와 분석에 사용 가능하게 된다.

3.2 타입 추론을 할 때 발생할 수 있는 문제점들

앞에서 살펴본 untyped CTOCT는 단순히 자바로부터 생성된 바이트 코드가 아닌 여러 다른 언어로부터 생성된 중간 코드이기 때문에 예상치 못한 문제가 발생할 수 있으며 정적 타입을 배정해야 하기 때문에 발생할 수 있는 여러 가지 문제들을 해야 한다. 일반적으로 바이트 코드 검증의 특정 속성, 인터페이스에 의한 다중 상속, 그리고 배열에 대한 타입 문제가 발생할 수 있다.

4. CTOCT에서 정적 타입 추론

최적화와 분석을 위한해서 타입이 없는 CTOCT를 타입을 가지는 CTOCT로 변환하여야 한다. 이를 위해서는 모든 지역 변수에 정적 타입을 주어야 한

다. 본 논문에서는 방향성 그래프(directed-graph)를 이용하여 타입을 추론한다. 이러한 시스템은 그래프에 연결된 요소를 찾고, 병합하고, 전이 제한을 제거하고, 단일 제한으로 병합하는 일을 수행한다.

4.1 타입을 위한 방향성 그래프(Directed Graph)

방향성 그래프 시스템은 메소드 몸체에 있는 CTOCT 명령어에 의해 지역 변수에 정적 타입을 지정하기위해 사용되는 그래프 표현이다. 이 시스템의 입력은 지역 변수에 타입이 지정되지 않은 타입을 가지지 않는 CTOCT이다.

타입 지정 그래프는 방향성 그래프로 다음과 같은 구성요소로 이루어진다.

①타입 노드(Type node) : 명확한 타입을 나타낸다.
②타입 변수 노드(Type variable node) : 타입 변수를 나타낸다.
③방향 간선(directed edge) : 두 노드간의 제한을 나타낸다.

가장 일반적인 기본적인 타입 지정은 배정 호환성에 관한 것이라고 할 수 있다. 방향 간선이 b로부터 a로 간다는 것은 프로그램 내에서는 a ← b라고하고 이것의 의미는 b는 a에 배정 가능하다는 의미이다. 이때의 b는 a와 동일한 타입이거나 a는 b의 상위 클래스일 때 가능하다. 이런 배정은 단순 배정문, 지역 변수에 이진 표현 배정, 그리고 메소드 호출이 일어난 경우에 주로 발생한다.

단순 배정일 경우 배정은 두개의 지역변수 사이에 일어난다. 예를 들면 [a = b]와 같은 경우이다. 만약 변수 b가 변수 a에 배정된다면 배정 호환성의 제한은 T(a) ← T(b)라고 할 수 있다. 여기서 T(a)와 T(b)는 각각 a와 b의 아직 알려지지 않은 타입을 나타낸다.

지역 변수 a에 이진 표현을 배정하는 경우엔 [a = b +3] 이라고 나타내고, T(a) ← T(b), T(a) ← int, T(b) ← int 와 같은 제한을 생성할 수 있다.

타입을 추론하는데 있어 대부분 복잡한 경우는 메소드를 호출하는 경우 생긴다. 메소드 호출의 예는 a = invokevirtual b.[boolean java.lang.Object.equals (java.lang.Object)](c) 와 같이 표현된다. 메소드 호출 시에는 많은 타입 정보들이 이용한다. 여기서 생성되는 제한은 다음과 같다.

반환 타입으로 T(a) ← int가 된다. 여기서 원래 반환 타입은 boolean이지만 타입의 단순화를 위해 boolean 타입은 단순 정수 타입으로 처리한다. equals 메소드의 클래스 선언으로부터 java.lang.Object ← T(b)이 되고, 메소드 시그니처의 매개변수 타입으로부터 java.lang.Object ← T(c)을 가지는 제한을 생성하게 된다.

타입 노드를 가지는 타입 변수노드의 병합은 지역 변수를 위한 타입 추론과 동등하게 된다. 일반적으로 둘 이상의 연관된 타입을 가지는 경우는 존재하지 않아야 한다. 만약 이런 일이 발생하면 SSA를 통해 문제를 해결해야 한다.

4.2 연결된 컴포넌트(Connected Components)

제한 그래프에서 처음으로 수행하는 동작은 연결된 컴포넌트들을 찾는 것이다. 연결된 컴포넌트를 찾게 되면 연결된 컴포넌트의 모든 모드들은 병합된다. 이것의 의미는 연결된 컴포넌트에 있는 모든 노드는 병합된 타입 노드에 의해 타입이 결정 된다는 것을 말한다.

연견된 컴포넌트는 세 가지 형태로 분류할 수 있다. 첫 번째의 경우는 타입 노드를 가지 않는 연결 컴포넌트인 경우이다. 이 경우 노드들은 단순히 병합되고, 모든 노드의 제한은 다음에 연결되는 타입 노드에 의해 결정되어 진다. 두 번째 경우는, 연결된 컴포넌트들이 하나의 타입 노드를 가지는 경우이다. 이 경우 모든 타입 변수 노드들은 타입 노드에 의해 병합되어 지고 모든 제한은 검증되어 지게 된다. 마지막 경우는 연결된 컴포넌트들이 두개 또는 그 이상의 타입 노드와 연결된 경우이다. 이런 경우엔 타입을 결정할 수 없게 된다. 이러한 경우 역시 SSA를 사용하여 문제를 해결한다.

4.3 전이 제한(Transitive Constraints)

연결된 컴포넌트들은 제한 그래프에서 제거되고, 비순환 방향 그래프(DAG)가 남게 된다. 이제 수행할 동작은 의사 제한을 비순환 방향 그래프부터 제거하는 것이다. 즉 그래프에서 전이가 발생하는 제한을 제거하는 것이다.

노드 y로부터 x로의 전이 제한은 다음과 같이 표현된다. 우선 x와 y 사이에 x ← y 제한이 존재한다면 x ≠ p 에 대해 p ← y 가 존재하고 x ← p 가 방향 그래프에 존재하게 된다.

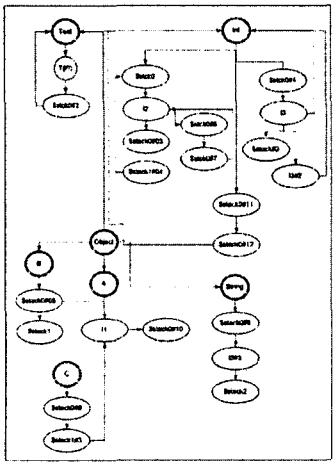
4.4 정적 단일 배정(Static Single Assignment)

분기문 후에 결합되는 변수의 정적 타입 배정을 위해서 정적 단일 타입 형식(SSA Form)을 사용한

다. SSA를 사용하기 위해서는 DF(dominance frontier)를 구해야 하는데 DF를 구하는 방법은 DF_{local}과 DF_{up}를 결합해서 DF를 계산한 후 Ø-함수(Ø-functions)을 추가한 후 단일 배정된 변수들에게 새로운 이름을 배정하여 구할수 있다.

4.5 공통 조상(Common Ancestor)

SSA를 통해서도 타입이 결정되지 않는 경우가 발생한다. 인터페이스에 의해서 다중 상속이 된 경우인데, 이 경우에는 상속관계에서 각 변수들의 공통 조상을 찾아서 결정해주면 된다. 자바에서 가장 최상위 조상은 java.lang.Object이다. 따라서 Object를 배정하면 대부분의 경우는 타입 충돌없이 해결할 수 있다. 하지만 가능하다면 가장 가까운 조상을 찾는 것이 가정 적절한 방법이다.



[그림 4] 타입 추론 그래프

[그림 4]는 타입 추론이 적용된 방향성 그래프이다. 타입을 추론하는 함수 T()에 해당 변수를 적용하면 [그림 5]와 같은 타입을 가지는 CTOCT를 생성할 수 있게 된다.

5. 결론

[그림 5]는 CTOC에 있는 CTOC-TR을 통해 생성한 3주소 코드인 타입을 가지는 CTOCT코드이다. CTOC에서 CTOCB는 기본적으로 스택 기반으로 동작이 수행되기 때문에 지역 변수를 위한 명시적인 타입을 가지지 않는다. 하지만 프로그램의 최적화나 역컴파일을 위해서는 지역 변수의 타입을 아는 것이 중요하다. 본 논문은 스택을 사용하지 않는 3-주소 코드에서 지역 변수를 위한 정적 타입 추론을 구현

```
public int runningExample(){
  Test l0, $stack0#2;
  int $stack0, l2, l3, $stack0#3, $stack0#4, $stack0#6,
      $stack1#2, l3#2, $stack0#7, $stack0#11, $stack1#4,
      $stack0#12;
  B $stack1, $stack0#5;
  A l1, $stack0#10;
  java.lang.String $stack2, $stack0#8, l3#3;
  C $stack0#9, $stack1#3;

  $stack0#0 = 0;
  ...
  $stack0#5 = new B;
  $stack1 = $stack0#5;
  ...
  return $stack0#12;
}
```

[그림 5] typed CTOCT

하였다. 이를 위해 본 논문에서는 SSA와 제한 시스템을 적용하였다. 향후 과제로는 CTOCT의 코드를 사용하여 3주소에서 최적화에 관한 연구를 수행하는 것이다.

참고문헌

[1] Alfred V. Aho, Ravi Sethi and Jeffrey D.Ullman, Compilers Principles, Techniques, and Tools, 1993

[2] Raja vallee-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. Soot - a Java bytecode optimization framework. In Proceedings of CASCON'99, 1999

[3] Etienne M. Gagnon and Laurie J. Hendren. Intra-procedural inference of static types for java bytecode. Technical Report Sable 1998-5, Mc Gill University, Montreal, Canada, October 1998

[4] John Meyer, Troy Downing, "Java Virtual Machine", O' RELLAY, 1997

[5] Andrew W. appel, Modern Compiler Implementation in Java. CAMBRIDGE UNIVERSITY PRESS, 1998, pp 437-477

[6] 김영국, 김기태, 조선문, 유원희,"바이트코드 최적화 프레임워크의 설계," 제21회 춘계학술발표대회 제11권 제1호, pp. 297-300, 한국정보처리학회 2004. 05.