(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2011년11월25일
(11) 등록번호 10-1086082
(24) 등록일자 2011년11월16일

(51) Int. Cl.
G06F 9/45 (2006.01)
(21) 출원번호 10-2005-7025045
(22) 출원일자(국제출원일자) 2004년05월21일
심사청구일자 2009년04월21일
(85) 번역문제출일자 2005년12월27일
(65) 공개번호 10-2006-0026896
(43) 공개일자 2006년03월24일
(86) 국제출원번호 PCT/US2004/015964
(87) 국제공개번호 WO 2005/006119
국제공개일자 2005년01월20일
(30) 우선권주장
10/607,601 2003년06월27일 미국(US)
(56) 선행기술조사문헌
US06356955 B1*
TALx86: 'A Realistic Typed Assembly Language', Greg Morrisett외8명, in the Proceedings of Second ACM SIGPLAN Workshop on Compiler Support for System Software (WCSSS99), pp.25~35*
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
마이크로소프트 코포레이션
미국 워싱턴주 (우편번호 : 98052) 레드몬드 원 마이크로소프트 웨이
(72) 발명자
플레스코, 마크, 로날드
미국 98033 워싱턴주 커클랜드 넘버에이201 노스 이스트 68번스트리트 10304
타르디티, 데이비드, 리드, 쥬니어.
미국 98033 워싱턴주 커클랜드 넘버103 노스이스트 60번 스트리트10110
(74) 대리인
제일특허법인

전체 청구항 수 : 총 11 항 심사관 : 지정훈

(54) 컴파일러에서 프로그래밍 언어를 타입-검사하는 방법 및 컴퓨터 판독 가능 기록 매체

(57) 요 약

여러 형태의 중간 언어에서 일관성을 검사하기 위한 타입의 표현, 타입-검사기 및 컴파일러가 제공된다. 컴파일러에서 프로그래밍 언어를 타입-검사하는 것은 타입-검사기에 대한 입력으로서 하나 이상의 규칙 집합을 이용함으로써 달성되며, 여러 기준 중 임의의 하나, 또는 둘 이상의 조합에 기초하여 규칙 집합 중 하나 이상을 선택한다. 특히 소스 언어, 아키텍처, 및 언어에 존재하는 타입 지정의 레벨이 타입 검사되는 것은 컴파일 단계이다. 그 후 선택된 하나 이상의 규칙 집합을 사용하여 언어가 타입-검사된다. 규칙 집합은 강한 타입-검사에 대응하는 하나의 규칙 집합, 약한 타입-검사에 대응하는 하나의 규칙 집합, 및 표현 타입-검사에 대응하는 하나의 규칙 집합을 포함할 수 있다. 대안적으로, 컴파일러는 이전에 언급된 기준 중 임의의 하나, 또는 둘 이상의 조합에 기초하여 규칙들의 더 큰 집합으로부터 실행 시에 규칙들 중 하나 이상의 집합을 구성하는 타입-검사기를 제공받을 수 있다.

특허청구의 범위

청구항 1

하나 이상의 규칙 집합에 따라 컴파일러에서 프로그래밍 언어를 타입-검사하는 방법으로서,

컴파일의 현재 단계에 기초하여 상기 규칙 집합 중 하나 이상의 규칙 집합을 선택하는 단계, 및

상기 선택된 하나 이상의 규칙 집합에 기초하여 상기 프로그래밍 언어를 타입-검사하는 단계를 포함하며,

상기 하나 이상의 규칙 집합은 강한 타입-검사(strong type-checking)에 대응하는 하나의 규칙 집합, 약한 타입-검사(weak type-checking)에 대응하는 하나의 규칙 집합, 및 표현 타입-검사(representation type-checking)에 대응하는 하나의 규칙 집합을 포함하는, 컴파일러에서 프로그래밍 언어를 타입-검사하는 방법.

청구항 2

제1항에 있어서,

상기 프로그래밍 언어를 타입-검사하는 단계는 상기 프로그래밍 언어의 다수의 중간 표현 각각을 타입-검사하는 단계를 포함하는, 컴파일러에서 프로그래밍 언어를 타입-검사하는 방법.

청구항 3

제2항에 있어서,

상기 선택된 하나 이상의 규칙 집합은 각 표현에 대해 상이한, 컴파일러에서 프로그래밍 언어를 타입-검사하는 방법.

청구항 4

제1항에 있어서,

상기 프로그래밍 언어는 상기 프로그래밍 언어의 구성요소가 다수의 타입 중 하나일 수 있음을 나타내는 타입을 포함하는, 컴파일러에서 프로그래밍 언어를 타입-검사하는 방법.

청구항 5

제4항에 있어서,

상기 하나 이상의 규칙 집합은 상기 프로그래밍 언어의 구성요소가 다수의 타입 중 하나일 수 있음을 나타내는 타입을 타입-검사하기 위한 규칙을 포함하는, 컴파일러에서 프로그래밍 언어를 타입-검사하는 방법.

청구항 6

제1항에 있어서,

상기 하나 이상의 규칙 집합은 다수의 규칙을 계층적 포맷으로 포함하는, 컴파일러에서 프로그래밍 언어를 타입-검사하는 방법.

청구항 7

제1항에 있어서,

상기 표현 타입-검사는 상기 프로그래밍 언어의 구성요소에 대해 드롭(drop)된 타입 정보를 허용하는, 컴파일러에서 프로그래밍 언어를 타입-검사하는 방법.

청구항 8

제1항에 있어서,

상기 약한 타입-검사는 타입 캐스팅(type casting)을 허용하는, 컴파일러에서 프로그래밍 언어를 타입-검사하는 방법.

청구항 9

소스 언어로 작성된 소스 코드를 컴파일하기 위한 컴파일러로 컴퓨터를 기능시키기 위한 프로그램을 기록한 컴퓨터 판독 가능 기록 매체로서,

상기 컴파일러는

다수의 타입 규칙, 및

규칙 집합을 구성하기 위해 상기 소스 언어에 기초하여 상기 다수의 타입 규칙으로부터 타입 규칙의 부분집합을 선택하는 타입-검사기를 포함하며,

강한 타입-검사에 대응하는 하나의 규칙 집합, 약한 타입-검사에 대응하는 하나의 규칙 집합, 및 표현 타입-검사에 대응하는 하나의 규칙 집합을 포함하는 3개의 규칙 집합이 구성되는, 컴퓨터 판독 가능 기록 매체.

청구항 10

제9항에 있어서,

상기 표현 타입-검사는 상기 소스 코드의 구성요소에 대해 드롭된 타입 정보를 허용하는, 컴퓨터 판독 가능 기록 매체.

청구항 11

제9항에 있어서,

상기 약한 타입-검사는 타입 캐스팅을 허용하는, 컴퓨터 판독 가능 기록 매체.

청구항 12

삭제

청구항 13

삭제

청구항 14

삭제

청구항 15

삭제

청구항 16

삭제

청구항 17

삭제

청구항 18

삭제

청구항 19

삭제

청구항 20

삭제

청구항 21

삭제

청구항 22

삭제

청구항 23

삭제

청구항 24

삭제

청구항 25

삭제

청구항 26

삭제

청구항 27

삭제

청구항 28

삭제

청구항 29

삭제

청구항 30

삭제

청구항 31

삭제

청구항 32

삭제

청구항 33

삭제

청구항 34

삭제

명 세 서

기 술 분 야

[0001] 본 발명은 타입 시스템(type system)에 관한 것으로, 특히, 새롭고 업데이트된 프로그래밍 언어로 확장가능한 타입 시스템에 관한 것이다.

배 경 기 술

[0002] 타입 시스템이란 런 타임 에러에 대한 탐지와 방지를 돕는 프로그래밍 언어에서 사용되는 시스템이다. 프로그래밍 언어가 변수, 함수 등의 객체에 대해 선언된 타입 집합을 포함하면, 이 프로그래밍 언어는 "타입 지정된 것이고", 이들 타입들은 프로그래밍 언어로 작성된 프로그램의 컴파일 동안 규칙 집합에 대해 검사된다. 만일 타입 지정된 언어로 작성된 소스 코드가 타입 규칙들 중 하나를 위반한 경우에는, 컴파일러 에러가 확정된다.

[0003] 컴파일러에서 사용하기 위한 타입 지정된 중간 언어는 지난 수년에 걸쳐 연구 단체에서 상당히 연구되어 왔다. 이들 언어는 컴파일러의 신뢰성 및 견고성을 향상시켰을 뿐 아니라, 불필요 정보 수집기(garbage collector)가 필요로 하는 정보를 추적하여 검사하는 체계적인 방식을 제공한다. 이 개념은 중간 표현을 갖는 것인데, 이 중간 표현은 타입들이 첨부되어 있으며 소스 프로그램에 대한 타입 검사와 유사한 방식으로 타입 검사가 행해질 수 있다. 그러나, 타입 지정된 중간 언어는 컴파일 프로세스 동안 명시적으로 만들어진 아이템들을 표현하는 타입들을 필요로 하기 때문에 구현하기가 매우 곤란하다.

[0004] 타입 지정된 중간 언어는 이것이 다수의 상이한 고레벨의 프로그래밍 언어를 표현해야할 경우에는 구현하기가 훨씬 더 곤란하다. 언어가 다르면 서로 다른 프리미티브 연산 및 타입을 가질 뿐 아니라, 고레벨의 프로그래밍 언어는 상이한 레벨의 타입 지정을 갖는다. 예를 들어, 어셈블리어와 같은 일부 언어는 일반적으로 타입이 지정되지 않는다. 환언하자면, 이들 언어는 타입 시스템을 갖지 않는다. 타입이 지정되는 언어들 중, 일부는 강하게 타입이 지정되는 반면, 다른 것은 보다 느슨하게 타입이 지정된다. 예컨대, C++는 일반적으로 느슨하게 타입 지정된 언어인 것으로 여겨지는 한편, ML 또는 파스칼은 강하게 타입 지정되는 언어인 것으로 여겨진다. 또한, 느슨하게 타입 지정된 일부 언어는 프로그램 내의 대부분의 코드 섹션들이 강하게 타입 지정될 수 있는 반면, 다른 코드 섹션들은 느슨하게 타입 지정될 수 있는 언어의 보다 작은 부분 집합을 갖는다. 예를 들어, C# 및 .NET에서 사용되는 마이크로소프트 중간 언어(MSIL)가 이것을 허용한다. 따라서, 이들 고레벨의 언어들 중 임의의 것을 표현하는 데 사용되는 타입 지정된 중간 언어는 서로 다른 타입 강도를 표현할 수 있어야 한다. 마찬가지로, 이런 타입 지정된 중간 언어의 타입 시스템은 타입 검사가 이루어진 코드의 특성에 따라 여러 규칙들을 실행할 수 있어야 한다.

[0005] 타입 지정된 중간 언어의 레벨이 컴파일 프로세스 전체에 걸쳐 낮아지면 다른 문제가 발생된다. 언어의 레벨이 낮아진다는 것은 언어의 형태를, 예를 들어, 프로그래머가 작성할 수 있는 고레벨의 형태로부터, 예를 들어, 중간 언어와 같은 저레벨의 형태로 변경하는 프로세스를 의미하는 것이다. 이어서, 언어는 중간 언어에서 기계-의존성 원시 코드와 같은 컴퓨터가 실행하는 것에 근접한 레벨로 더욱 낮아진다. 컴파일 프로세스 동안 다른 레벨로 낮아진 중간 언어에 대해 타입-검사를 행하기 위해, 각 표현마다 다른 규칙 집합을 사용해야 한다.

[0006] 타입 지정된 중간 언어를 작성하려는 시도는 종종 상술된 문제점의 해결에 미치지 못한다. 예를 들어, Cedilla Systems' Special J 컴파일러는 타입 지정된 중간 언어를 사용한다. 그러나, 이 컴파일러는 자바 소스 언어에 고유하므로, 예컨대, 비-타입-안전 코드를 가질 수 있는 다수의 언어를 처리할 필요가 없었다. 또한, 이 컴파일러는 단지 타입-검사를 위한 하나의 규칙 집합만을 사용하므로, 여러 레벨의 컴파일에서는 사용될 수 없다. 연구자들에게 있어서, 타입 지정된 중간 언어는 종종 여러 단계의 컴파일 동안 소스 언어에 고도로 고유하며 엔지니어(및 타입 설계)에게 곤란하게 되는 경향이 있다.

발명의 상세한 설명

[0007] 여러 형태의 중간 언어에서 일관성을 검사하기 위한 타입 표현, 타입-검사기, 방법, 및 컴파일러가 제공된다. 상세히 기술하자면, 타입 지정된 중간 언어는 타입 지정된 언어 및 타입 미지정된 언어, 느슨하게 또는 강하게 타입 지정된 언어, 및 불필요 정보 수집을 갖거나 갖지 않는 언어를 포함하여 다수의 (이종) 소스 언어로 작성된 프로그램을 표현하는 데 사용하기에 적합하다. 또한, 타입 검사기 이키텍처는 타입 및 프리미티브 연산이 서로 다른 새로운 언어를 처리하도록 확장가능하다. 타입 표현, 타입-검사기, 방법, 및 컴파일러는 여러 양상들을 포함한다. 이들 여러 양상들은 각자 독립적으로 사용될 수 있거나, 여러 양상들은 여러 결합 및 부분-결합에 사용될 수 있다.

[0008] 본 발명의 한 양상에 있어서, 컴파일러에서 프로그래밍 언어의 타입을 검사하는 방법이 제공된다. 타입-검사기로의 입력으로서 하나 이상의 규칙이 택해지며, 이 타입-검사기는 다수의 기준 중 임의 하나, 또는 둘 이상의 결합에 기초하여 하나 이상의 규칙 집합을 선택한다. 이들 가운데는 컴파일 단계, 소스 언어, 아키텍처, 및 타입이 검사된 언어에 제공된 타입 지정 레벨이 있다. 이 언어는 선택된 하나 이상의 규칙 집합을 사용하여 타입

검사된다.

[0009] 본 발명의 다른 양상에 있어서, 다수의 기준 중 임의 하나, 또는 둘 이상의 결합에 기초하여 하나 이상의 규칙 집합을 구성하는 타입 검사기를 갖는 컴파일러가 제공된다. 이들 규칙 집합은 강한 타입 검사에 대응하는 하나의 규칙 집합, 약한 타입 검사에 대응하는 하나의 규칙 집합, 및 표현 타입 검사에 대응하는 하나의 규칙 집합을 포함할 수 있다. 약한 규칙 집합은 타입 캐스트(type cast)를 허용하는 등의 타입 지정 시에 보다 많은 유연성을 허용할 수 있는 한편, 표현 규칙 집합은 중간 프로그램 표현의 부분들에 드롭된(drop) 타입 정보를 허용할 수 있다.

[0010] 본 발명의 다른 양상에 있어서, 프로그램의 중간 표현의 일관성을 검사하기 위한 복수의 규칙들 구성하는 프로그래밍 인터페이스가 제공된다. 중간 표현의 일관성을 검사하는 것에는 소정의 기준에 따라 하나의 중간 표현에 제1 규칙 집합을 적용시키고 또 다른 중간 표현에 제2 규칙 집합을 적용시키는 타입-검사기에 복수의 규칙을 제공하는 것이 포함될 수 있다.

[0011] 본 발명의 상기 및 기타 양상들은 첨부된 도면을 참조하여 이하에서 기술한 상세한 설명으로부터 명백해질 것이다.

실 시 예

[0021] 여러 형태의 중간 언어에서 일관성을 검사하기 위한 타입 표현, 타입-검사기, 및 컴파일러가 제공된다. 타입-검사기 및 컴파일러는 프로그램 컴포넌트에 대한 소스 언어 및/또는 컴파일 단계에 따라 서로 다른 타입 및 타입-검사 규칙을 사용할 수 있다. 예컨대, 고레벨의 최적화기를 각종 언어로 작성된 프로그램에 적용시키는 것이 바람직할 수 있다. 이들 언어는 서로 다른 프리미티브 타입 및 프리미티브 연산들을 가질 수 있다. 어느 한 언어는 예를 들어, 복합 산술을 위한 타입 및 연산을 포함할 수 있는 한편, 또 다른 언어는 컴퓨터 그래픽에 고유한 타입 및 연산을 포함할 수 있다. 중간 표현이 서로 다른 타입 시스템에 의해 파라미터화되는 것이 가능함으로써, 최적화기는 서로 다른 프리미티브 타입 및 연산을 갖는 언어에 사용될 수 있다. 다른 예는 어떤 컴포넌트들은 강하게 타입 지정된 언어의 부분 집합으로 작성되고, 다른 컴포넌트는 타입이 안전하지 않은 완전(full) 언어로 작성되는 프로그램을 포함할 수 있다. 제1 컴포넌트 집합에 대해 보다 많은 에러 검사를 행하는 바람직하다. 이는 서로 다른 컴포넌트마다 서로 다른 타입 검사 규칙을 사용함으로써 달성될 수 있다. 또 다른 예로서는 컴파일 동안 타입 정보를 드롭하는 것이다. 타입-검사기 및 컴파일러는 초기 단계 동안은 정확한 정보를 유지시키면서 나중 단계에서는 타입 정보를 드롭시킬 수 있다. 이는 서로 다른 컴파일 단계 동안 서로 다른 타입 검사 규칙들과 결합하여 알려지지 않은 타입을 사용함으로써 달성될 수 있다.

[0022] 도 1은 다수의 서로 다른 소스 언어를 표현하도록 여러 레벨이 낮아진 타입 지정된 중간 언어를 이용하는 시스템에 대한 포괄적인 컴파일 프로세스를 도시한다. 소스 코드(100 내지 106)는 타입 지정될 수 있거나 지정되지 않을 수 있고 타입 강도 레벨이 다른 4개의 상이한 소스 언어로 작성된다. 예를 들어, C#으로 작성된 소스 코드(100)는 예컨대 C++로 작성된 소스 코드(106)보다 훨씬 강하게 타입 지정될 것이다. 소스 코드는 우선 처리되어 판독기(108)에 의해 시스템 내로 입력된다. 그런 후 소스 언어는 타입 지정된 중간 언어의 고레벨의 중간 표현(HIR)으로 번역된다. 그 후, HIR은 블럭(110)에서 선택적으로 분석 및 최적화될 수 있다. 다음에, HIR은 타입 지정된 중간 언어의 중간 레벨의 중간 표현(MIR)으로 번역된다. 이 표현은 HIR보다는 낮은 레벨이지만 여전히 기계 독립형이다. 이 때, MIR은 블럭(112)에서 도시된 바와 같이 선택적으로 분석 및 최적화될 수 있다. 그리고 나서, MIR은 블럭(114)에서 코드 생성에 의해 타입 지정된 중간 언어의 기계-의존성 저레벨 표현으로 번역된다. 그 후, LIR은 블럭(116)에서 선택적으로 분석 및 최적화될 수 있으며, 블럭(118)에서 에미터에 공급될 수 있다. 이 에미터는 시스템 내로 판독된 초기 소스 코드를 표현하는 다수의 포맷(120 내지 126) 중 하나로 코드를 출력할 것이다. 이 프로세스 거쳐, 프로세스를 완료하는 데 필요한 데이터는 임의 형태의 영속성 메모리(128)에 저장된다.

[0023] 이와 같이, 컴파일 프로세스는 중간 언어 명령어를 한 레벨의 표현에서 다른 레벨의 표현으로 변환하는 것으로 이루어진다. 예를 들어, 도 2는 소스 코드문을 HIR로 변환하는 것뿐 아니라, HIR을 기계-의존성 LIR로 변환하는 것을 도시한다. 소스 코드문(200)은 다수의 고레벨 프로그래밍 언어로 작성될 수 있다. 이들 언어는 프로그래머가 쉽게 이해하는 방식으로 코드를 작성하고 판독할 수 있도록 설계된다. 따라서, 프로그래머는 추가를 위해 '+'와 같은 문자를 사용할 수 있고, 소스 코드문(200)에 도시된 바와 같이 둘 이상의 연산자를 추가하는 등의 보다 강력한 형태를 사용할 수 있다.

[0024] 명령문(202 내지 206)은 동일한 기능을 표현하지만, 컴퓨터가 이해할 수 있으나 여전히 아키텍처 독립형인 것에

근사한 포맷으로 표현하는 소스 코드문(200)의 HIR 표현이다. 명령문(202)은 제1 및 제2 변수를 추가하는 'ADD' 명령을 사용하고, 그 결과를 제1 일시적 변수 t1에 할당한다. 다음에, 명령문(204)은 제3 변수에 t1을 추가하기 위한 또 다른 'ADD' 명령을 사용하여 그 결과를 제2 일시적 변수 t2에 할당한다. 이어서, 명령문(206)은 t2의 값을 'ASSIGN' i 명령어를 이용하여 최종 변수 z에 할당한다.

[0025] 명령문(208 내지 212)은 명령문(202 내지 206)의 중간 언어의 LIR이다. 명령문(208)은 x86 아키텍처에 고유한 추가 명령어를 이용하여 특정 레지스터에 저장된 두 변수의 값들을 추가하고 그 결과를 일시적 변수 t1에 할당된 레지스터에 저장한다. 명령문(210)은 x86 아키텍처에 고유한 추가 명령어를 사용하여 특정 레지스터에 저장된 t1의 값 및 제3 변수를 추가하여 t2에 할당된 특정 레지스터(EAX)에 그 결과를 저장한다. 다음에, 명령문(212)은 x86 아키텍처에 고유한 이동 명령어를 이용하여 EAX에 저장된 값을 출력 변수 z로 이동시킨다.

[0026] 타입 검사를 실행하기 위해, 타입 지정된 중간 언어는 명시적 또는 암시적으로 표현된 타입 표현을 포함한다. 명시적 타입 표현식은 표현에서 직접 선언된다. 예를 들어, 명령문:

[0027] int a;

[0028] 은 변수 'a'를 타입 int로서 명확히 정의한다. 타입 표현은 소정의 코드 명령문에 대한 디폴트 타입을 정의함으로써 암시적으로 표현될 수 있다. 예컨대, 함수에 대한 디폴트 반환 타입은 int이므로, 명령문:

[0029] f_start ();

[0030] 은 인수를 택하지 않고 타입 int의 값을 반환하는 함수 f_start를 선언할 것이다.

[0031] 다수의 프로그래밍 언어를 여러 레벨의 표현으로 사용하기에 적합한 타입 지정된 중간 언어에 대한 타입 표현의 일 실시예가 부록 A에 나타나 있다. 이는 단지 여러 가능한 실시예들 중 일례에 불과한 것임에 유의해야 한다.

[0032] 부록 A를 참조해 보면, 여러 언어의 타입 시스템이 타입 지정된 중간 언어로 표현될 수 있도록 다수의 타입 표현이 타입 클래스 계층에 정의되어 있다. 추상 베이스 클래스는 모든 타입에 대해 'Phx::Type'로서 정의된다. 베이스 클래스는 예컨대, 실제, 심볼 또는 미지의 (또는 변수) 타입 등의 여러 타입에 대한 'sizekind'의 사이즈 정보를 포함할 수 있다. 베이스 클래스는 또한 타입 분류를 지정하기 위해 'typekind'를 포함할 수 있다. 추가로, 타입 지정된 중간 언어로부터 초기 소스 코드로의 역 매핑을 제공하기 위해 외부적으로 정의된 타입을 래핑(wrap)하는 추상 타입으로서 외부 타입이 제공될 수 있다.

[0033] 베이스 클래스 아래, 'Phx::PtrType'으로 정의된 클래스는 포인터 타입을 나타낼 수 있다. 다양한 종류의 포인터들도 정의될 수 있다. 그 다양한 종류의 포인터들에는, 예를 들어, 관리되는 폐영역 회수된 포인터(폐영역 회수된 객체 내의 위치로의 포인터), 관리되는 폐영역 회수되지 않은 포인터(폐영역 회수되지 않은 객체 내의 위치로의 포인터), 관리되지 않는 포인터(예를 들어, C++로 작성된 코드 내에서 발견되는 것과 같음), 참조 포인터(폐영역 회수된 객체의 베이스로의 포인터), 및 널이 있다.

[0034] 계층 내의 동일 레벨에서, 'Phx::ContainerType'으로 정의된 클래스는 내부 멤버들을 포함하는 타입들과 같은 컨테이너 타입을 나타낼 수 있다. 내부 멤버들은 필드, 메소드 및 기타 타입들을 가질 수 있다. 'Phx::FuncType'으로 정의된 클래스는 임의의 필수적인 호출 규정, 인수들의 리스트, 및 리턴 타입들의 리스트를 포함하는 함수 타입을 나타낼 수 있다. 또한, 'Phx::UnmgdArrayType'으로 정의된 클래스는 관리되지 않은 어레이 타입을 나타낼 수 있다. 계층 내의 'Phx::ContainerType' 하에는, 4개의 클래스가 더 정의될 수 있다. 'Phx::ClassType'으로 정의된 클래스는 클래스 타입을 나타낼 수 있으며, 'Phx::StructType'으로 정의된 클래스는 구조 타입을 나타낼 수 있고, 'Phx::InterfaceType'으로 정의된 클래스는 인터페이스 타입을 나타낼 수 있으며, 'Phx::EnumType'으로 정의된 클래스는 열거된 타입을 나타낼 수 있다. 계층 내의 'Phx::ClassType' 하에는, 'Phx::MgdArrayType'으로 정의된 추가적인 클래스가 관리된 어레이 타입을 나타낼 수 있다.

[0035] 부록 A에 나타낸 표현들에서, 'primtype'이란 클래스는 구조 타입의 특별한 인스턴스로 정의된다. 'primtype'은 int, float, unknown, void, condition code, unsigned int, xint 등의 다양한 타입을 포함할 수 있다. 이러한 표현은 타입 지정된 중간 언어(typed intermediate language)의 HIR 또는 LIR 모두에서 사용될 수 있다.

[0036] 부가적으로, 타겟 특유 프리미티브 타입(target specific primitive type)들이 타입 표현에 포함될 수 있다. 몇몇 언어들은 타입 시스템이 그들을 알고 만들어졌으면 효율적으로 처리될 수 있는 복잡한 산술 타입을 갖는다. 'MMX' 명령어의 경우를 살펴보기로 한다. 이러한 명령어는 멀티미디어 및 통신 데이터 타입 상의 단일 명령어/다수의 데이터 연산을 지원하기 위해 몇몇 버전의 x86 프로세서에 구축된 잉여 명령어들의 집합 중 하나이다. 타입 시스템은 타입 표현이 최소한으로 변형된 이러한 명령어들을 인식 및 이용하도록 맞춤화될 수

있다.

[0037] 부록 A에 나타낸 타입들의 타입 표현의 실시예는 또한 어떤 타입도 나타낼 수 있으며 선택적으로 그것에 연관된 크기를 갖는 "unknown" 타입을 포함한다. 이 크기는 값의 기계적 표현에 대한 크기이다. unknown 타입은, 컴파일러가 타입 정보를 특정 타입에서부터 unknown 타입으로 변경함으로써 제어된 방식으로 타입 정보를 드롭(drop)하는 것을 허용한다. 이는, 타입을 모르더라도, 컴파일러가 조작되는 값의 크기에 의존하는 코드를 생성하는 것을 허용한다. 다른 타입들도 unknown 타입을 이용할 수 있으므로, unknown 타입은 부분적 타입 정보의 표현을 허용한다(모두가 아닌 몇몇 정보만이 알려짐).

[0038] 예를 들어, int 타입으로의 포인터를 가정한다. 하향화(lowering)의 몇몇 단계에서, 타입 정보 int를 드롭하는 것이 바람직할 수 있다. unkown 타입은 컴파일러가 int 타입을 unkown 타입으로 대체시키는 것을 허용한다. 그 후에, 타입 검사기는 해당 포인터가 올바른 타입을 가리키고 있는지를 검사할 필요가 없다. 이것은 본질적으로 가리켜진 값이 실행시간에 프로그램 기능에 불리하게 영향을 주지 않는 방식으로 전달될 수 있게 한다.

[0039] unkown 타입을 이용하는 또 다른 예는 함수에 대한 타입을 정의하기 위한 것이다. 인수가 이미 int로의 타입 포인터를 갖는 경우, unkown으로의 타입 포인터의 인수를 갖는 함수가 호출되면, 컴파일러는 올바른 타입이 전달되었다고 믿어야 한다. 포인터를 역참조(dereferencing)한 결과는 int라고 알려지거나 알려지지 않을 수 있지만, int로 이용될 것이다. 보다 복잡한 예는 가상 함수 호출의 고레벨 중간 표현에서부터 저레벨 중간 표현으로의 변환 동안의 중간 임시 변수의 도입이다. 가상 테이블(vtable)은 객체-지향 언어로 가상 호출을 구현하는 데 광범위하게 이용된다. 저레벨 중간 표현으로 가상 함수 호출을 만드는 첫번째 단계는 메모리의 객체의 첫번째 필드를 페치(fetch)하는 것이다. 첫번째 필드는 vtable로의 포인터를 포함한다. 그 후 페치의 결과가 임시 변수에 할당된다. 임시 변수의 타입(vtable이 다수의 필드를 가질 수 있을 때, vtable로의 포인터를 나타내는 타입)을 구성하는 것은 표현하기 복잡하고 부담스러울 수 있다. 대신, 컴파일러는 단순하게 중간 임시 변수를 "unknown으로의 포인터"로 할당할 수 있다. 따라서, unknown 타입의 사용은, 상세한 타입 정보를 유지하는 것이 불필요하거나 컴파일러 구현기에 상당한 부담을 나타낼 수 있는 경우, 컴파일의 나중 단계들을 단순화시킨다.

[0040] 도 3은 컴파일의 다양한 단계에서 타입 지정된 중간 언어를 타입 검사(type-checking)하고, 이에 따라 하향화의 다양한 레벨에서 타입 지정된 중간 언어를 타입 검사하기 위한 컴파일러 시스템의 일 실시예를 예시한다. 소스 코드(300)는 다양한 소스 언어들 중 임의의 하나를 나타낸다. 소스 코드(300)는 타입 지정된 중간 언어의 HIR(302)로 변환된다. 이것을 행할 때, 소스 언어의 타입 표현은 타입 지정된 중간 언어에 내부적인 타입 표현으로 변환된다.

[0041] 도 1 및 도 2에 관련하여 설명된 바와 같은 HIR은 컴파일 프로세스 곳곳에서 하향화된다. 예시를 위해, 고(HIR, 302), 중(MIR, 304), 및 저(LIR, 306) 레벨 표현이 도시된다. 그러나, 실시예가 이에 제한되는 것은 아니다. 임의의 수의 컴파일 단계들이 타입 검사될 수 있다.

[0042] 표현의 각 레벨에서의 중간 언어는 타입 검사기(308)에 의해 타입 검사될 수 있다. 타입 검사기(308)는 하나 이상의 규칙 집합(310)을 컴파일 프로세스의 각 단계에 적용하고, 이에 따라 중간 언어의 각 표현에도 적용하기 위한 알고리즘 또는 절차를 구현한다. 규칙 집합(310)은 소스 언어, 컴파일 단계, 타입 지정 강도 등과 같은 언어의 다양한 속성에 대해 설계된 규칙들의 집합이다.

[0043] 예를 들어, 소스 코드(300)가 C++ 프로그래밍 언어로 작성된 코드를 포함한다고 가정한다. C++ 소스 코드(300)는 먼저 타입 지정된 중간 언어의 HIR(302)로 변환된다. 원한다면, 이 시점에서 타입 검사기(308)는 임의의 수의 속성들을 결정하기 위해 HIR(302)과 상호작용할 수 있다. 이러한 속성들은 컴파일의 단계(HIR), 소스 코드 표현 타입(C++), 언어의 타입이 지정되었는지 아닌지(예), 타입이 느슨하게 또는 강하게 지정되었는지(느슨하게) 등을 포함할 수 있다. 속성에 기초하여, 타입 검사기는 적절한 규칙 집합을 선택할 수 있다. 일단 규칙 집합이 선택되면, 타입 검사기는 그 규칙 집합에 따라 HIR을 타입 검사한다. 일단 HIR이 MIR 또는 LIR로 하향화되면, 속성은 다시 액세스될 것이고, 동일하거나 상이한 규칙 집합들이 적합할 수 있다.

[0044] 일 실시예에서, 세 가지 타입 검사 규칙 집합이 타입 검사기에 제공될 수 있다. 하나의 집합은 C# 또는 MSIL을 타입 검사하기에 바람직한 것과 같은 "강한" 타입 검사에 대응할 수 있다. 또 다른 집합은 "강한" 타입 검사보다 느슨한 타입 검사인 "약한" 타입 검사에 대응할 수 있다. 예를 들어, 약한 타입 검사 규칙 집합은 타입 캐스트(type cast)를 허용할 수 있다. 타입 캐스트는, 한가지 타입의 변수가 단일 사용을 위해 또 다른 것처럼 동작하도록 하는 경우이다. 예를 들어, int 타입의 변수는 char(character)처럼 동작하도록 할 수 있다. 다음

코드는 문자 'P'를 출력하기 위해 타입 캐스트를 이용한다.

[0045] int a;

[0046] a = 80;

[0047] cout<<(char)a;

[0048] 따라서, 'a'가 int 타입으로 정의되고 값 80이 할당되지만, cout 명령문은 타입 캐스트로 인해 변수 'a'를 char 타입으로 취급하여, 이에 따라 80이 아닌 'P'(ASCII 값 80)를 디스플레이할 것이다.

[0049] 마지막으로, 집합은 "표현" 검사에 대응할 수 있다. "표현" 검사는 unknown형을 사용하는 것과 같이 중간 프로그램 표현의 일부에서 드롭된 타입 정보를 허용할 수 있으며, 이러한 타입 정보가 드롭될 수 있는 경우 또는 unknown 타입이 또 다른 타입을 대신할 수 있는 경우를 나타내는 규칙을 포함할 수 있다. 예를 들어, Void 타입의 값을 리턴하는 함수의 결과는 unknown 타입의 변수에 할당되지 않도록 방지될 수 있다.

[0050] 부가적으로, 두 개 이상의 규칙 집합이 컴파일의 단일 단계에서 이용될 수 있다. 예를 들어, 소스 코드(300)는 단일 언어를 포함하지만, 강하게 타입 지정된 섹션과 느슨하게 타입 지정된 몇몇의 섹션을 포함한다고 가정한다. 타입 검사기는 특정 강하게 타입 지정된 섹션 내의 HIR에 대해 하나의 규칙 집합, 및 느슨하게 타입 지정된 코드 섹션에 대해 또 다른 규칙 집합을 이용할 수 있다.

[0051] 도 4는 도 3에 설명된 것과 유사한, 컴파일러 시스템 내에서의 사용을 위한 타입 검사기의 블럭도이다. 타입 검사기(400)는 입력으로 상이한 소스 언어 및/또는 상이한 컴파일 단계들에 대응하는 임의의 수의 규칙 집합을 수락할 수 있다. 도 4에서, 네 가지의 규칙 집합(402-408)이 타입 검사기(400)에 제공된다. 규칙 집합(402)은 강하게 타입 지정된 언어들의 HIR에 대한 규칙 집합을 나타내고, 규칙 집합(404)은 약하게 타입 지정된 언어들의 HIR에 대한 규칙 집합을 나타내며, 규칙 집합(406)은 타입 지정되지 않은 언어들의 HIR에 대한 규칙 집합을 나타내고, 규칙 집합(408)은 LIR에 대한 규칙 집합을 나타낸다. 프로그램 모듈(410)은 HIR 내의 강하게 타입 지정된 언어를 나타내고, 프로그램 모듈(412)은 LIR로 하향화된 후의 프로그램 모듈(410)을 나타낸다.

[0052] 타입 검사기(400)는 타입 검사되는 프로그램 모듈의 속성들에 기초하여 적절한 규칙 집합을 선택하고, 통합된 절차 또는 알고리즘을 이용하여 선택된 규칙 집합을 프로그램 모듈에 적용한다. 예를 들어, 타입 검사기(400)는 프로그램 모듈(410)(HIR의 강하게 타입 지정된 언어를 나타냄)을 타입 검사하기 위한 규칙 집합(402)(강하게 타입 지정된 언어들의 HIR에 대한 규칙 집합을 나타냄)을 선택할 수 있다. 이어서, 그 후 타입 검사기(400)는 프로그램 모듈(412)(LIR의 강하게 타입 지정된 언어를 나타냄)을 타입 검사하기 위한 규칙 집합(408)(LIR에 대한 규칙 집합을 나타냄)을 선택할 수 있다.

[0053] 도 5는 타입 검사기에 의해 적용될 규칙 집합을 선택하기 위한 절차의 하나의 가능한 실시예의 흐름도이다. 블럭(500)에서, 타입 검사기는 소스 코드의 타입 지정된 중간 표현의 섹션 내에서 판독하고, 타입 검사하기 위한 규칙 집합을 선택해야 한다. 결정 블럭(502)은 타입 지정된 중간 언어가 HIR, MIR 또는 LIR인지를 판정한다.

[0054] HIR 또는 MIR이면, 원시 소스 코드가 느슨하게 타입 지정되었는지 또는 강하게 타입 지정되었는지를 판정하기 위한 결정 블럭(504)이 프로세싱된다. 느슨하게 타입 지정된 경우, 약한 타입 검사에 대응하는 규칙 집합을 선택하기 위한 블럭(506)이 프로세싱된다. 강하게 타입 지정된 경우, 강한 타입 검사에 대응하는 규칙 집합을 선택하기 위한 블럭(508)이 프로세싱된다.

[0055] LIR인 경우, 표현 타입 검사에 대응하는 규칙 집합을 선택하기 위한 결정 블럭(510)이 프로세싱된다. 도 5는 단지 하나의 실시예라는 것을 명심해야 한다. 상이한 속성들에 대응 및 기초하여 임의의 수의 규칙 집합들이 선택될 수 있다.

[0056] 설명된 타입 검사 시스템의 규칙 집합은 전체적으로 새로운 언어로 확장될 수 있고, 또한 기존 언어의 새로운 특성으로 용이하게 확장될 수 있다. 예를 들어, 새로운 언어가 도입되어야 하는 경우, 단순히 새로운 언어를 위한 새로운 규칙 집합이 작성된다. 규칙 집합이 타입 검사기 또는 컴파일 시스템 자체와 개별적이고 규칙 집합을 개별적인 엔티티로 수용하도록 설계되기 때문에, 기존 타입 검사 시스템 또는 컴파일러를 재분배 또는 갱신할 필요없이, 새로운 언어에 대한 새로운 규칙 집합이 분배될 수 있다. 마찬가지로, 예를 들어, C++에 XML 지원을 추가하는 것처럼, 새로운 특징이 기존 언어에 추가되면, 다양한 컴파일 단계에서 C++에 대응하는 규칙 집합이 새로운 특징을 처리하도록 용이하게 동적으로 재구성될 수 있다. 또다시, 새로운 핵심 시스템이 갱신 또는 분배될 필요가 없다.

[0057] 규칙 집합은 또한 타입에 대한 제약을 허용할 수 있다. 예를 들어, 클래스가 또 다른 것으로부터 상속될 때 특정한 타입에 대한 하위 타입을 지정하는 것이 허용되는지가 규칙 내에 설명된 제약일 수 있다. 또 다른 제약은, 데이터가 데이터를 포함하는 가상 테이블로 변환될 수 있다는 것을 나타내는 데 바람직할 수 있는 것과 같이 정리된(boxed) 제약일 수 있다. 다른 것들에는 크기 제약, 또는 프리미티브의 동일한 타입에 대한 필요성을 나타내는 프리미티브 타입 제약이 포함될 수 있다. 규칙 집합의 임의의 다른 부분과 같이, 새로운 제약들이 원하는 대로 추가될 수 있다.

[0058] 타입 검사기에 의해 사용된 규칙 집합은 규칙 집합을 작성하기 위한 애플리케이션에의 프로그래밍 인터페이스를 통해 구성될 수 있다. 어플리케이션은, 규칙 집합이 타입 지정된 중간 언어의 개별적인 명령어들에 할당된 규칙을 갖는 타입 프리미티브의 계층에 표현되도록 규칙을 구성할 수 있다. 계층은 특정 프로그램 모듈 또는 컴파일 유닛에 관련된 다양한 타입의 구성요소들을 명시적으로 표현할 타입 그래프의 형태로 제공될 수 있다. 심볼 및 연산과 같은 IR 구성요소는 타입 시스템의 구성요소에 연관될 것이다. 타입 그래프 노드는 컴포넌트, 네스팅된 타입, 함수 서명, 인터페이스 타입, 계층의 구성요소, 및 소스 이름, 및 모듈/어셈블리 외부 타입 구성요소에의 참조 등의 기타 정보와 같은 구성된 타입, 그들의 관계, 및 프리미티브를 설명할 것이다.

[0059] 단순한 타입 규칙의 예는 다음과 같다:

[0060] ADD

[0061] N = add n, n

[0062] 본 예에서, I는 부호화된 정수 타입이고, U는 부호화되지 않은 정수 타입이고, X는 둘 중 한 타입의 정수이고, F는 실수이고, N은 상기의 것들 중 임의의 것이라고 가정한다. 도 6은 이러한 타입들 간의 계층적 관계를 나타낸다. 타입 N은 계층구조의 상위에 존재한다. 타입 F 및 X는 타입 N으로부터 갈라져 내려와 계층의 다음 레벨을 형성한다. 마지막으로, 타입 U 및 I는 타입 X로부터 갈라져 내려와 계층의 최하위 레벨을 형성한다. 따라서, 'ADD' 중간 언어 명령어에 대해서, 이 규칙에 따라 오직 타입 N 또는 계층 내의 그보다 하위의 타입만이 add 명령어에 의해 프로세싱될 수 있으며, 연산은 계층 상에서 결과보다 높이 있어서는 안 된다. 예를 들어, 두 개의 정수를 더해서 하나의 정수를 만들 수 있으며(I=ADD i, i), 또는 정수와 실수를 더해서 하나의 실수를 만들 수 있다(F=ADD i, f). 그러나, 실수 와 정수를 더해서 하나의 정수를 만들 수는 없다(I=ADD i, f).

[0063] 타입 프리미티브를 계층으로서 표현하는 것은 규칙 집합이 쉽게 변경되도록 한다. 예전에는, 타입 규칙이 종종 소스 코드를 사용하여 프로그램적으로 표현되었다. 예를 들어, 타입-검사기는 타입-검사기 규칙을 구현하는 많은 수의 스위치 명령문을 포함할 수 있다. 따라서, 규칙을 변경하는 것은 타입-검사기에 대한 소스 코드를 수정할 것을 필요로 하였다. 그러나, 계층적 규칙 집합은 훨씬 쉬운 확장성을 제공한다. ADD 명령어에 대한 이전 규칙을 고려한다. 개발자가 타입(예를 들어, 복합 타입에 대한 C)을 추가하기를 원한다면, 단순히 도 7에 도시된 것과 같은 계층에서 N 타입 아래에 추가될 수 있으며, ADD 명령어에 대한 규칙은 원하는 대로 동작하도록 변경될 필요가 없다.

[0064] 타입 검사 시스템에서 명령어를 타입 규칙에 대해 검사하기 위한 한 방법이 도 8에 도시된다. 먼저, 명령어를 구문적으로 검사하기 위한 블럭(800)이 프로세싱된다. 따라서, 참조번호(806)의 명령어를 고려하는 경우, 타입-검사기는 올바른 수의 소스 및 대상 표현이 ADD 명령어에 대한 타입 규칙에 따라 존재함(예를 들어, 이 경우에는, 2개의 소스 표현 및 하나의 대상 표현이 존재함)을 보장할 것이다. 각 표현(및 부표현)은 중간 표현으로 자신에 대한 명시적 타입을 가질 수 있다. 그러면 블럭(802)에서, 타입-검사기는 e1, e2 및 foo(e3)에 대한 명시적 타입이 ADD 명령어에 대한 타입 규칙에 따른다는 것을 실제로 검증할 것이다. 블럭(804)에서, 타입-검사기는 필요하다면 명령어들을 계속 타입-검사하기 위하여 서브-레벨을 횡단할 것이다. 예를 들어, 타입-검사기는 표현 e1, e2 및 foo(e3)이 그들의 명시적 타입과 일치하는지를 검사할 수 있다. 예를 들어, 타입-검사기는 foo가 함수 타입을 갖는지를 검사할 수 있다. 함수 타입의 결과 타입이 foo(e3)의 명시적 타입과 동일한지를 검사할 수 있다. 단일 인수 타입(single argument type)이 존재하는지, 및 타입 e3가 그 타입과 매칭하는지를 더 검사할 수 있다. 이것은 e3에의 호출의 타입이 타입 규칙과 일치함을 보장한다.

[0065] 도 9는 타입-검사 시스템의 실시예에 대한 운영 환경으로서 동작하는 컴퓨터 시스템의 예를 도시한다. 컴퓨터 시스템은 프로세싱 유닛(921), 시스템 메모리(922), 및 시스템 메모리를 포함하는 다양한 시스템 컴포넌트를 프로세싱 유닛(921)에 상호연결하는 시스템 버스(923)를 포함하는 개인용 컴퓨터(920)를 포함한다. 시스템 버스는 메모리 버스 또는 메모리 제어기, 주변 버스, 및 PCI, VESA, 마이크로채널(MCA), ISA 및 EISA 등과 같은 버스 아키텍처를 사용하는 로컬 버스를 포함하는 여러 타입의 버스 구조 중 임의의 것을 포함할 수 있다. 시스템

메모리는 ROM(924) 및 RAM(925)을 포함한다. 시작할 때 등에 개인용 컴퓨터(920) 내의 구성요소들 사이에 정보 전달을 돕는 기본 루틴을 포함하는 기본 입력/출력 시스템(926)(BIOS)은 ROM(924)에 저장된다. 개인용 컴퓨터(920)는 또한 하드 디스크 드라이브(927), 예를 들어 분리형 디스크(929)에 대한 판독 또는 기입을 위한 자기 디스크 드라이브(928), 및 예를 들어 CD-ROM 디스크(931)에 대한 판독 또는 기타 광 매체에 대한 판독 또는 기입을 위한 광 디스크 드라이브(930)를 포함한다. 하드 디스크 드라이브(927), 자기 디스크 드라이브(928) 및 광 디스크 드라이브(930)는 각각 하드 디스크 드라이브 인터페이스(932), 자기 디스크 드라이브 인터페이스(933), 및 광 드라이브 인터페이스(934)에 의해 시스템 버스(923)에 연결된다. 드라이브 및 관련 컴퓨터-판독가능 매체는 개인용 컴퓨터(920)에 대한 데이터, 데이터 구조, 컴퓨터-실행가능 명령어(동적 링크 라이브러리와 같은 프로그램 코드, 및 실행 파일) 등 의 비휘발성 저장을 제공한다. 상기에서 컴퓨터-판독가능 매체에 대한 설명은 하드 디스크, 분리형 자기 디스크 및 CD를 참조하지만, 자기 카세트, 플래시 메모리 카드, DVD, 베르누이 카트리지 등과 같이 컴퓨터에 의해 판독가능한 다른 타입의 매체도 포함할 수 있다.

[0066] 운영 시스템(935), 하나 이상의 어플리케이션 프로그램(936), 기타 프로그램 모듈(937), 및 프로그램 데이터(938)를 포함하는 여러 프로그램 모듈이 드라이브 및 RAM(925)에 저장될 수 있다. 사용자는 키보드(940), 및 마우스(942)와 같은 포인팅 장치를 통해 개인용 컴퓨터(920)에 명령어 및 정보를 입력할 수 있다. 기타 입력 장치(도시되지 않음)는 마이크, 조이스틱, 게임 패드, 위성 디쉬, 스캐너 등을 포함할 수 있다. 이들 및 기타 입력 장치는 종종 시스템 버스에 연결되는 직렬 포트 인터페이스(949)를 통해 프로세싱 유닛(921)에 연결되지만, 병렬 포트, 게임 포트 또는 USB와 같은 기타 인터페이스에 의해서 연결될 수도 있다. 모니터(947) 또는 기타 타입의 디스플레이 장치 또한 디스플레이 제어기 또는 비디오 어댑터(948)와 같은 인터페이스를 통해 시스템 버스(923)에 연결된다. 모니터 이외에, 개인용 컴퓨터는 전형적으로 스피커 및 프린터와 같은 기타 주변 출력 장치(도시되지 않음)를 포함한다.

[0067] 개인용 컴퓨터(920)는 원격 컴퓨터(949)와 같은 하나 이상의 원격 컴퓨터에의 논리적 연결을 사용하는 네트워크 환경에서 동작할 수 있다. 원격 컴퓨터(949)는 서버, 라우터, 피어 장치 또는 기타 일반적인 네트워크 노드일 수 있으며, 도 9에는 메모리 저장 장치(950)만이 도시되었지만, 전형적으로 개인용 컴퓨터(920)에 관련하여 기술된 구성요소 중 다수 또는 모두를 포함한다. 도 9에 도시된 논리적 연결은 LAN(951) 및 WAN(952)을 포함한다. 그러한 네트워크 환경은 사무실, 기업-규모 컴퓨터 네트워크, 인트라넷 및 인터넷에서 일반적인 것이다.

[0068] LAN 네트워크 환경에서 사용되는 경우, 개인용 컴퓨터(920)는 네트워크 인터페이스 또는 어댑터(953)를 통해 로컬 네트워크(951)에 연결된다. WAN 네트워크 환경에서 사용되는 경우, 개인용 컴퓨터(920)는 전형적으로 인터넷과 같은 WAN(952)를 통한 통신을 설립하기 위한 모뎀(954) 또는 기타 수단을 포함한다. 내장 또는 외장일 수 있는 모뎀(954)은 직렬 포트 인터페이스(946)를 통해 시스템 버스(923)에 연결된다. 네트워크 환경에서, 개인용 컴퓨터(920)에 관련하여 도시된 프로그램 모듈, 또는 그의 일부는 원격 메모리 저장 장치에 저장될 수 있다. 도시된 네트워크 연결은 단지 예일 뿐이며, 컴퓨터들 사이에 통신 링크를 설립하는 기타 수단이 사용될 수 있다.

[0069] 예시된 실시예들의 원칙을 도시 및 기술함에 의해, 그러한 원칙으로부터 벗어나지 않고서 실시예들이 배치 및 세부사항에 있어서 수정될 수 있음이 본 분야에서 숙련된 기술을 가진 자들에게 명백해질 것이다.

[0070] 예를 들어, 본원의 일 실시예는 타입-검사기 또는 컴파일러에 제공될 수 있는 하나 이상의 규칙 집합을 기술하며, 컴파일러 또는 타입-검사기가 타입-검사되는 언어 및/또는 컴파일의 페이즈에 기초하여 언어를 타입-검사하기 위해 규칙 집합 중 하나 이상을 선택한다. 그러나, 대안적으로, 컴파일러 또는 타입-검사기가 타입-검사되는 언어 및/또는 컴파일의 페이즈에 기초하여 정적으로든지 런타임 시에 동적으로든지 규칙들의 단일 집합으로부터 규칙들의 하나 이상의 부집합을 구성하도록, 규칙들의 단일 집합이 타입-검사기 또는 컴파일러에 제공될 수 있다.

[0071] 여러 가능한 실시예의 관점에서, 예시된 실시예들은 단지 예를 포함하는 것이며, 본 발명의 범위에 대한 제한으로 여겨져서는 안 된다는 것을 인식할 것이다. 오히려, 본 발명은 후술되는 특허청구범위에 의해 정의된다. 그러므로 이러한 특허청구범위의 범위 내에 속하는 그러한 모든 실시예들을 본 발명으로서 특허청구한다.

[0072] 부록 A

```
//------------------------------------------------------------------------------//
// Description:
//
//  IR types
//
//
//   Type Class Hierarchy          Description & Primary Properties Introduced
//   --------------------          -------------------------------------------
//   Phx::Type                    - Abstract base class for types
//     Phx::PtrType               - Pointer types
//     Phx::ContainerType         - Container types (types that have members)
//         Phx::ClassType         - Class types
//           Phx::MgdArrayType - Managed array types
//         Phx::StructType        - Struct types
//         Phx::InterfaceType     - Interface types
//         Phx::EnumType          - Enumerated types
//     Phx::FuncType              - Function types
//                                Properties: ArgTypes, ReturnType
//
//     Phx::UnmgdArrayType        - Unmanaged arrays
//                                Properties: Dim, Referent
//
//------------------------------------------------------------------------------

//------------------------------------------------------------------------------
//
// Description:
//
//  Base class for IR types
//
//------------------------------------------------------------------------------

__abstract __public __gc
class Type : public Phx::Object
{

public:

  // Functions for comparing types.

  virtual Boolean IsAssignable(Phx::Type * srcType);
  virtual Boolean IsEqual(Phx::Type * type);

public:

  // Public Properties

  DEFINE_GET_PROPERTY(Phx::TypeKind,      TypeKind,    typeKind);
  DEFINE_GET_PROPERTY(Phx::SizeKind,      SizeKind,    sizeKind);
  DEFINE_GET_PROPERTY(Phx::BitSize,       BitSize,     bitSize);
```

[0073]

```
    DEFINE_GET_PROPERTY(Symbols::ConstSym *, ConstSym,     constSym);
    DEFINE_GET_PROPERTY(Phx::ExternalType *, ExternalType, externalType);
    DEFINE_GET_PROPERTY(Phx::PrimTypeKindNum, PrimTypeKind, primTypeKind);
    GET_PROPERTY(Phx::TypeSystem *, TypeSystem);

protected:

    // Protected Fields

    Phx::TypeKind        typeKind;       // type classification
    Phx::SizeKind        sizeKind;       // size classification
    Phx::BitSize         bitSize;        // size in bits when constant
    Symbols::ConstSym *  constSym;       // size in bits when symbolic
    Phx::PrimTypeKindNum primTypeKind;
    Phx::ExternalType *  externalType;   // optionally null
};

//------------------------------------------------------------------------------
//
// Description:
//
//   Container Type - Abstract class for types that have members.
//
//------------------------------------------------------------------------------

__abstract __public __gc
class ContainerType : public QuantifiedType, public IScope
{

    DEFINE_PROPERTY(Symbols::FieldSym *, FieldSymList, fieldSymList);

private:

    // Private Fields

    Symbols::FieldSym * fieldSymList;

};

//------------------------------------------------------------------------------
//
// Description:
//
//    Class Container Type
//
//------------------------------------------------------------------------------

__public __gc
class ClassType : public ContainerType
{

public:
```

[0074]

```
    // Public Static Constructors

    static Phx::ClassType * New
    (
        Phx::TypeSystem *       typeSystem,
        Phx::BitSize            bitSize,
        Phx::ExternalType *     externalType
    );

public:

    // Public Properties

    DEFINE_GET_PROPERTY(Phx::Type *, UnboxedType,     unboxedType);
    DEFINE_PROPERTY(    ClassType *, ExtendsClassType, extendsClassType);

    DEFINE_GET_PROPERTY(Phx::Collections::InterfaceTypeList *,
        ExplicitlyImplements, explicitlyImplements);

protected:

    // Protected Properties

    DEFINE_SET_PROPERTY(Phx::Collections::InterfaceTypeList *,
        ExplicitlyImplements, explicitlyImplements);

private:

    // Private Fields

    Phx::Type *                         unboxedType;
    Phx::ClassType *                    extendsClassType;
    Phx::Collections::InterfaceTypeList * explicitlyImplements;
};

//------------------------------------------------------------------------------
//
// Class: StructType
//
// Description:
//
//    Type of structs.
//
//
//------------------------------------------------------------------------------

__public __gc
class StructType : public ContainerType
{

public:
```

[0075]

```
    // Public Static Constructors

    static Phx::StructType * New
    (
        Phx::TypeSystem *       typeSystem,
        Phx::BitSize            bitSize,
        Phx::ExternalType *     externalType
    );

public:

    // Public Properties

    DEFINE_GET_PROPERTY(Phx::ClassType *, BoxedType, boxedType);

private:

    // Private Fields

    Phx::ClassType * boxedType;
};

//------------------------------------------------------------------------------
//
// Class: PrimType
//
// Description:
//
//    Primitive types
//
//------------------------------------------------------------------------------

__public __gc
class PrimType : public StructType
{

public:

    // Public Static Constructors

    static Phx::PrimType *
    New
    (
        Phx::TypeSystem *       typeSystem,
        Phx::PrimTypeKindNum    primTypeKind,
        Phx::BitSize            bitSize,
        Phx::ExternalType *     externalType
    );

public:

    // Public Methods
```

[0076]

```
    static Phx::PrimType *
    GetScratch
    (
      Phx::PrimType * type,
      PrimTypeKindNum kind,
      Phx::BitSize bitSize
    );

    Phx::PrimType * GetResized
    (
      Phx::BitSize bitSize
    );

public:

    // Public Properties

    DEFINE_GET_PROPERTY(Phx::TypeSystem *,  TypeSystem, typeSystem);

private:

    // Private Fields

    Phx::TypeSystem * typeSystem;
};

//-------------------------------------------------------------------------------
//
// Class: InterfaceType
//
// Description:
//
//    Interface types
//
//-------------------------------------------------------------------------------

__public __gc
class InterfaceType : public ContainerType
{

public:

    // Public Static Constructors

    static Phx::InterfaceType * New
    (
      Phx::TypeSystem *        typeSystem,
      Phx::ExternalType *      externalType
    );

public:

    // Public Properties
```

[0077]

```
  DEFINE_PROPERTY(    Phx::ClassType *, ExtendsClassType, extendsClassType);
  DEFINE_GET_PROPERTY(Phx::Collections::InterfaceTypeList *,
    ExplicitlyImplements, explicitlyImplements);

protected:

  // Protected Properties

  DEFINE_SET_PROPERTY(Phx::Collections::InterfaceTypeList *,
    ExplicitlyImplements, explicitlyImplements);

private:

  // Private Fields

  Phx::ClassType *                    extendsClassType;
  Phx::Collections::InterfaceTypeList * explicitlyImplements;
};

//------------------------------------------------------------------------------
//
// Class: EnumType
//
// Description:
//
//  Enumeration types
//
//------------------------------------------------------------------------------

__public __gc
class EnumType : public ContainerType
{

public:

  // Public Static Constructors

  static Phx::EnumType * New
  (
    Phx::TypeSystem *       typeSystem,
    Phx::ExternalType *     externalType,
    Phx::Type *             underLyingType
  );

public:

  // Public Properties

  DEFINE_GET_PROPERTY(Phx::ClassType *, BoxedType, boxedType);
  DEFINE_GET_PROPERTY(Phx::Type *, UnderlyingType, underlyingType);

private:
```

[0078]

```
    // Private Fields

    Phx::Type * underlyingType;
    Phx::ClassType * boxedType;
};

//-------------------------------------------------------------------------
//
// Class: MgdArrayType
//
// Description:
//
//    Managed array types.
//
//
//-------------------------------------------------------------------------

__public __gc
class MgdArrayType : public ClassType
{

public:

    // Public Static Constructors

    static Phx::MgdArrayType * New
    (
        Phx::TypeSystem *         typeSystem,
        Phx::ExternalType *       externalType,
        Phx::Type *               elementType
    );

public:

    // Public Properties

    DEFINE_GET_PROPERTY(Phx::Type *,      ElementType,  elementType);

private:

    // Private Fields

    Phx::Type *  elementType;
};

//-------------------------------------------------------------------------
//
// Class: UnmgdArrayType
//
// Description:
//
//    Unmanaged array types.
```

[0079]

```
//
//
//------------------------------------------------------------------------------

__public __gc
class UnmgdArrayType : public Type
{

public:

  // Public Static Constructors

  static Phx::UnmgdArrayType * New
  (
    Phx::TypeSystem *       typeSystem,
    Phx::BitSize            bitSize,
    Phx::ExternalType *     externalType,
    Phx::Type *             referentType
  );

public:

  // Public Properties

  DEFINE_GET_PROPERTY(int,        Dim,          dim);
  DEFINE_GET_PROPERTY(Phx::Type *, Referent,     referent);

private:

  // Private Fields

  int         dim;
  Phx::Type *  referent;
};


//------------------------------------------------------------------------------
//
// Description:
//
//    Pointer types
//
//
//------------------------------------------------------------------------------

__public __gc
class PtrType : public Type
{

public:

  // Public Static Constructors
```

[0080]

```
  static Phx::PtrType * New
  (
    Phx::TypeSystem *        typeSystem,
    Phx::PtrTypeKind         ptrTypeKind,
    Phx::BitSize             bitSize,
    Phx::Type *              referent,
    Phx::ExternalType *      externalType
  );

  //  Constructor without type pointed to.

  static Phx::PtrType * New
  (
    Phx::TypeSystem *        typeSystem,
    Phx::PtrTypeKind         ptrTypeKind,
    Phx::BitSize             bitSize,
    Phx::ExternalType *      externalType
  );

public:

  // Public Properties

  DEFINE_GET_PROPERTY(Phx::PtrTypeKind,   PtrTypeKind,   ptrTypeKind);
  DEFINE_GET_PROPERTY(Phx::Type *,        Referent,      referent);

private:

  // Private Fields

  Phx::PtrTypeKind    ptrTypeKind;
  Phx::Type *         referent;
};

//------------------------------------------------------------------------------
//
// Enum: CallingConvention
//
// Description:
//
//   A preliminary enum to represent calling convention types.
//
//------------------------------------------------------------------------------

BEGIN_ENUM(CallingConventionKind)
{
  _Illegal = 0,
  CLRCall,
  CDecl,
  StdCall,
  ThisCall,
  FastCall
}
```

[0081]

```
END_ENUM(CallingConventionKind);

//-----------------------------------------------------------------------------
//
// Class: FuncType
//
// Description:
//
//   Function types.
//
//
//-----------------------------------------------------------------------------

__public __gc
class FuncType : public QuantifiedType
{

public:

   // Public Static Constructors

public:

   // Public Methods

   Int32          CountArgs();
   Int32          CountRets();
   Int32          CountArgsForInstr();
   Int32          CountRetsForInstr();
   Int32          CountUserDefinedArgs();
   Int32          CountUserDefinedRets();
   Phx::FuncArg * GetNthArgFuncArg(Int32 index);
   Phx::FuncArg * GetNthRetFuncArg(Int32 index);
   Phx::FuncArg * GetNthArgFuncArgForInstr(Int32 index);
   Phx::FuncArg * GetNthRetFuncArgForInstr(Int32 index);
   Phx::FuncArg * GetNthUserDefinedArgFuncArg(Int32 index);
   Phx::FuncArg * GetNthUserDefinedRetFuncArg(Int32 index);
   Phx::Type *    GetNthArgType(Int32 index);
   Phx::Type *    GetNthRetType(Int32 index);
   Phx::Type *    GetNthArgTypeForInstr(Int32 index);
   Phx::Type *    GetNthRetTypeForInstr(Int32 index);
   Phx::Type *    GetNthUserDefinedArgType(Int32 index);
   Phx::Type *    GetNthUserDefinedRetType(Int32 index);

public:

   // Public Properties

   DEFINE_GET_PROPERTY(Phx::CallingConventionKind,                CallingConvention,
callingConvention);
   GET_PROPERTY(Phx::Type *,    RetType);

   // True if this function type has an ellipsis funcarg.
```

[0082]

```
    GET_PROPERTY(Boolean,       IsVarArgs);
    GET_PROPERTY(Boolean,       IsInstanceMethod);
    GET_PROPERTY(Boolean,       IsClrCall);
    GET_PROPERTY(Boolean,       IsCDecl);
    GET_PROPERTY(Boolean,       IsStdCall);
    GET_PROPERTY(Boolean,       IsThisCall);
    GET_PROPERTY(Boolean,       IsFastCall);

    // Not True if this function has a return value.

    GET_PROPERTY(Boolean,       ReturnsVoid);

protected:

    // Protected Fields

    Phx::CallingConventionKind  callingConvention;
    Phx::FuncArg *              argFuncArgs;
    Phx::FuncArg *              retFuncArgs;
};

//-------------------------------------------------------------------------
//
// Description:
//
//    The global type system used during compilation.
//
//
//-------------------------------------------------------------------------

__public __gc
class TypeSystem : public Phx::Object
{

public:

    // Public Static Constructors

    static TypeSystem *
    New
    (
        Phx::BitSize      regIntBitSize,
        Phx::BitSize      nativeIntBitSize,
        Phx::BitSize      nativePtrBitSize,
        Phx::BitSize      nativeFloatBitSize,
    );

public:

    // Public Methods

    void Add(Type * type);
```

[0083]

```
public:

  // lists of created types

  DEFINE_GET_PROPERTY(Phx::Type *, AllTypes,      allTypes);

private:

  // List of all types in the system.

  Phx::Type * allTypes;
};

//-------------------------------------------------------------------------
//
// Description:
//
//   Various enums to describe types.
//
//-------------------------------------------------------------------------

// Classes and value types for IR types

// The different kinds of types.

BEGIN_ENUM(TypeKind)
{
  _Illegal = 0,
  Class,
  Struct,
  Interface,
  Enum,
  MgdArray,
  UnmgdArray,
  Ptr,
  Func,
  Variable,
  Quantifier,
  Application,
  TypedRef,
}
END_ENUM(TypeKind);

// The different kinds of type sizes.

BEGIN_ENUM(SizeKind)
{
  _Illegal = 0,
  Constant,
  Symbolic,
  Unknown
}
```

[0084]

```
  END_ENUM(SizeKind);

  // The different types of pointers

  BEGIN_ENUM(PtrTypeKind)
  {
    _Illegal = 0,
    ObjPtr,          // __gc pointer to object whole.
    MgdPtr,           // __gc pointer to object interior.
    UnmgdPtr,          // __nogc pointer.
    NullPtr,         // pointer to nothing.
    _NumPtrTypeKinds
  }
  END_ENUM(PtrTypeKind);
  }
```

[0085]

도면의 간단한 설명

[0012] 도 1은 포괄적인 컴파일 프로세스의 흐름도.

[0013] 도 2는 소스 코드문이 고레벨의 표현으로, 다시 기계-의존성 저레벨의 표현으로 변환되는 것을 나타내는 테이블 리스팅을 도시.

[0014] 도 3은 여러 단계의 컴파일에서 타입 지정된 중간 언어의 타입을 검사하기 위한 컴파일러 시스템의 일 실시예를 예시하는 데이터 흐름도.

[0015] 도 4는 컴파일러 시스템에서 사용하기 위한 타입-검사기의 블럭도.

[0016] 도 5는 타입-검사기에 의해 적용될 규칙 집합을 선택하기 위한 하나의 가능한 절차에 대한 흐름도.

[0017] 도 6은 타입들 간에서의 계층 관계를 나타내는 직접 그래프 다이어그램.

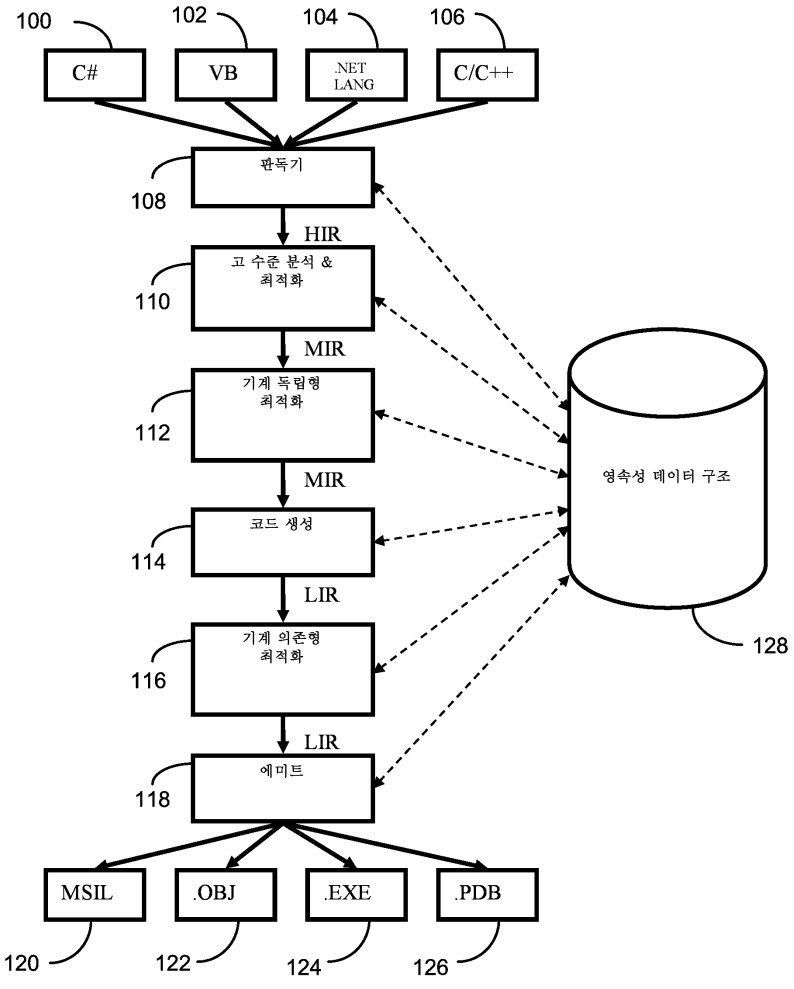
[0018] 도 7은 타입들 간에서의 계층 관계에 타입을 추가하는 것을 나타내는 직접 그래프 다이어그램.

[0019] 도 8은 타입-검사 시스템에서 타입 규칙에 대한 명령어를 검사하기 위한 방법에 대한 흐름도.

[0020] 도 9는 타입-검사 시스템의 실시예에 대해 운영 환경으로서 동작하는 컴퓨터 시스템의 일례의 블럭도.

도면

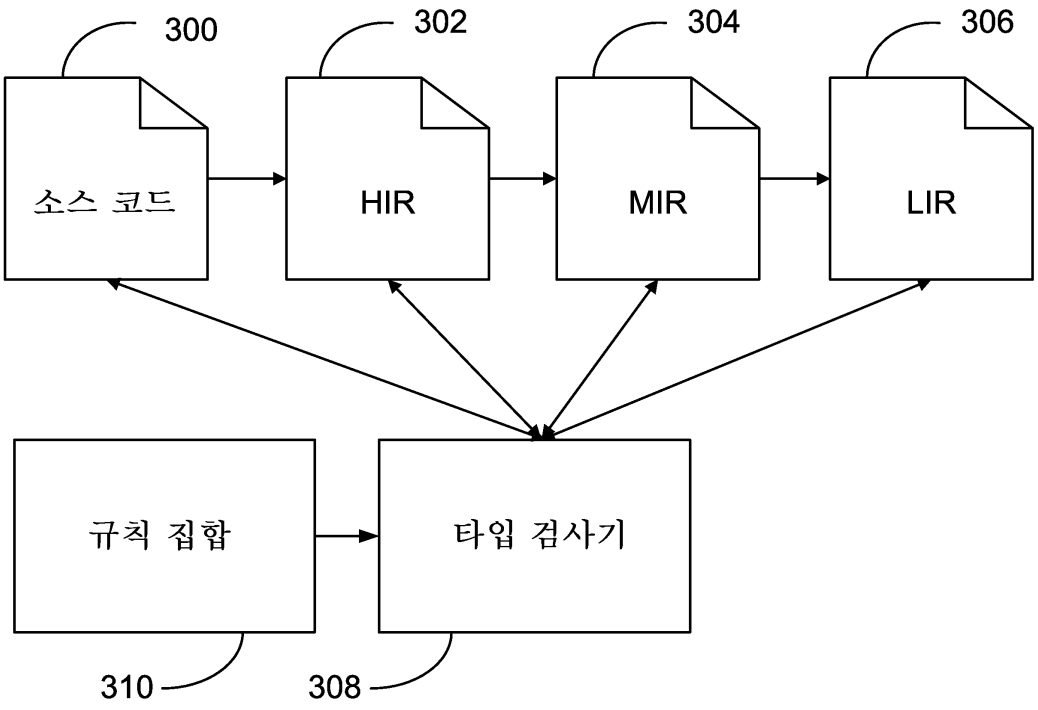
도면1



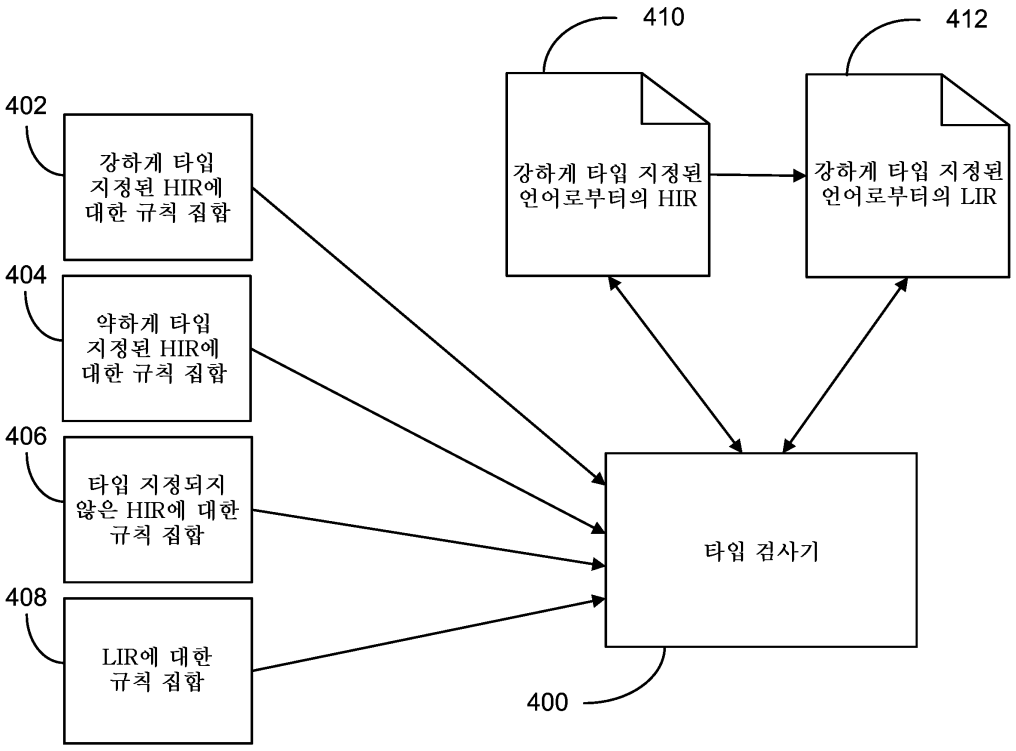
도면2

```
Source:
                    z = a + b + c;   - 200
HIR:
                  t1 = ADD    a, b   - 202
                  t2 = ADD    t1, c  - 204
                   z = ASSIGN t2     - 206
LIR:
       t1(EAX), cc = x86add a(EAX), b(EDX)  - 208
       t2(EAX), cc = x86add t1(EAX), c(EBX) - 210
                   z = x86mov t2(eax)       - 212
```

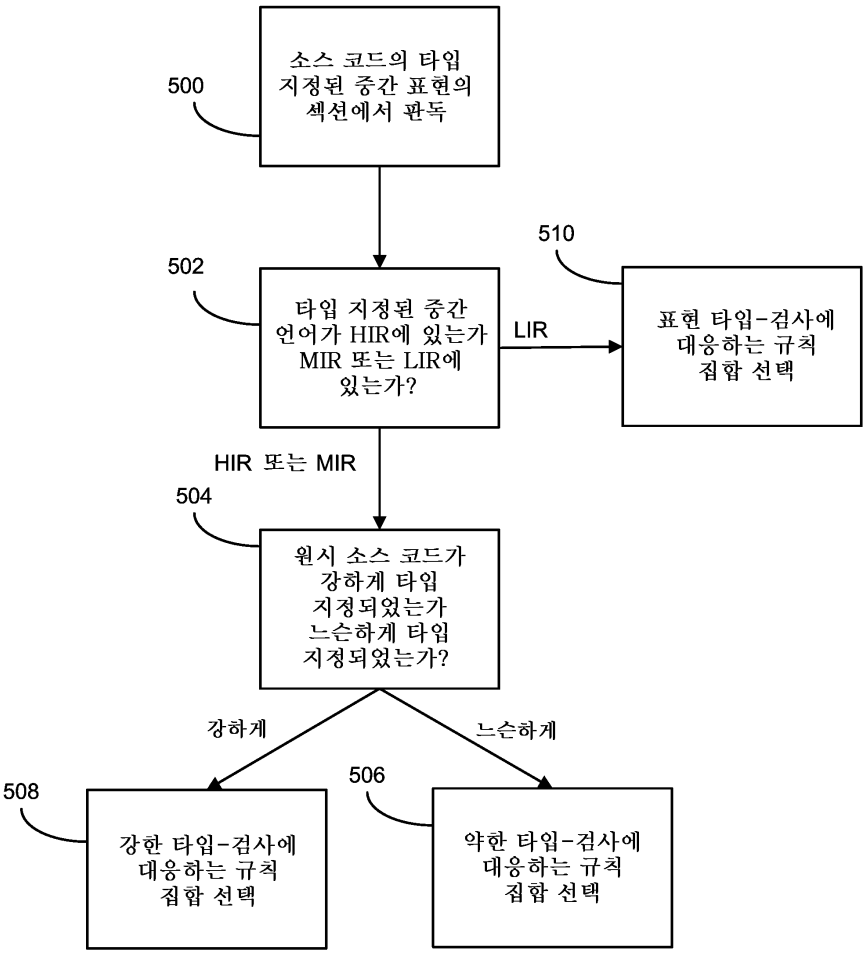
도면3



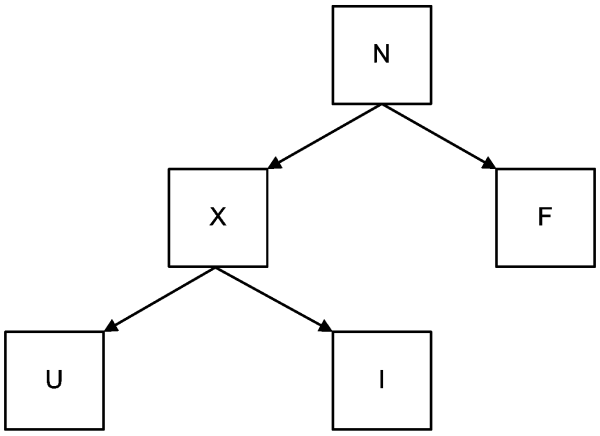
도면4


410
412
402
강하게 타입 지정된 HIR에 대한 규칙 집합
강하게 타입 지정된 언어로부터의 HIR
강하게 타입 지정된 언어로부터의 LIR
404
약하게 타입 지정된 HIR에 대한 규칙 집합
406
타입 지정되지 않은 HIR에 대한 규칙 집합
408
LIR에 대한 규칙 집합
타입 검사기
400

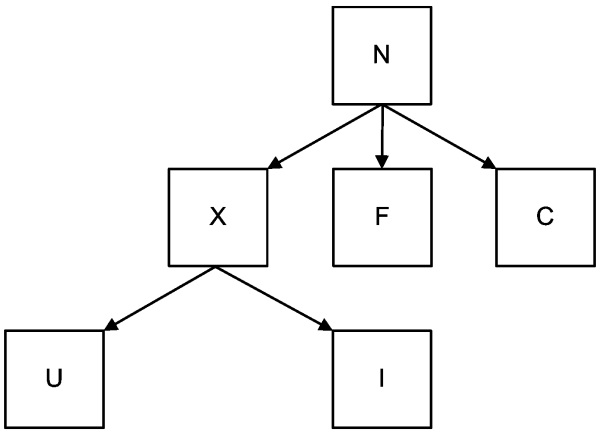
도면5


소스 코드의 타입 지정된 중간 표현의 섹션에서 판독
500
타입 지정된 중간 언어가 HIR에 있는가 MIR 또는 LIR에 있는가?
502
LIR
510
표현 타입-검사에 대응하는 규칙 집합 선택
HIR 또는 MIR
504
원시 소스 코드가 강하게 타입 지정되었는가 느슨하게 타입 지정되었는가?
강하게
느슨하게
508
강한 타입-검사에 대응하는 규칙 집합 선택
506
약한 타입-검사에 대응하는 규칙 집합 선택

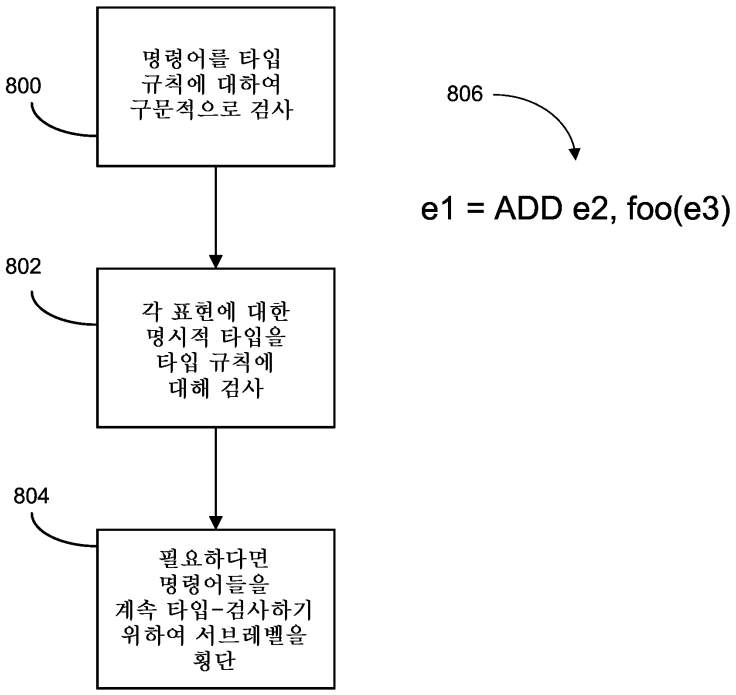
도면6


N
X
F
U
I

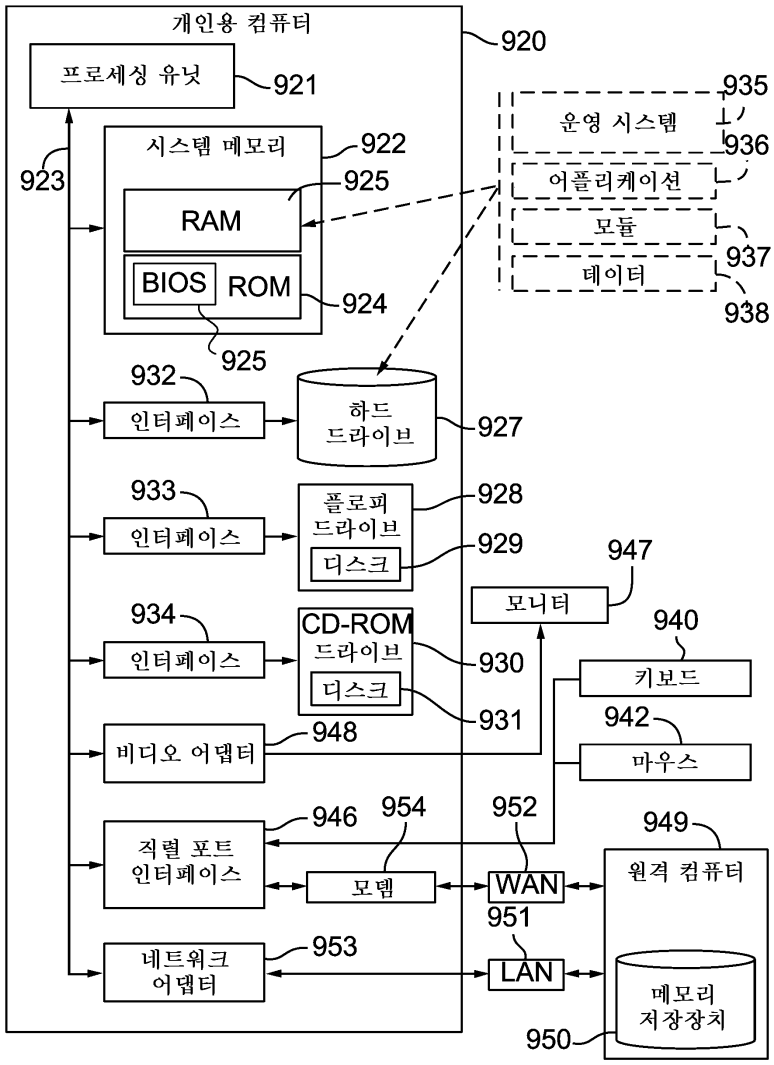
도면7


N
X
F
C
U
I

도면8


800
명령어를 타입 규칙에 대하여 구문적으로 검사
802
각 표현에 대한 명시적 타입을 타입 규칙에 대해 검사
804
필요하다면 명령어들을 계속 타입-검사하기 위하여 서브레벨을 횡단
806
e1 = ADD e2, foo(e3)

도면9


개인용 컴퓨터
920
프로세싱 유닛
921
시스템 메모리
922
923
925
RAM
BIOS
ROM
924
925
운영 시스템
935
936
어플리케이션
모듈
937
데이터
938
932
인터페이스
하드 드라이브
927
933
인터페이스
플로피 드라이브
928
디스크
929
947
모니터
934
인터페이스
CD-ROM 드라이브
930
디스크
931
940
키보드
942
마우스
948
비디오 어댑터
946
954
952
949
직렬 포트 인터페이스
모뎀
WAN
951
원격 컴퓨터
953
네트워크 어댑터
LAN
메모리 저장장치
950