(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2017년05월02일
(11) 등록번호 10-1731629
(24) 등록일자 2017년04월24일

(51) 국제특허분류(Int. Cl.)
G06F 9/45 (2006.01) *G06F 11/30* (2006.01)
G06F 17/30 (2006.01) *G06F 9/44* (2006.01)
(52) CPC특허분류
G06F 8/40 (2013.01)
G06F 11/302 (2013.01)
(21) 출원번호 10-2015-0186580
(22) 출원일자 2015년12월24일
심사청구일자 2015년12월24일
(56) 선행기술조사문헌
KR1020110020520 A
US20070261120 A1
US20090132995 A1
US20100131929 A1

(73) 특허권자
고려대학교 산학협력단
서울특별시 성북구 안암로 145, 고려대학교 (안암동5가)
(72) 발명자
최진영
서울특별시 노원구 동일로248길 16, 506동 1003호(상계동, 수락파크빌아파트)
신지훈
인천광역시 남동구 장승남로 70-30 101동 1603호 (만수동,삼익아파트)
(74) 대리인
김등용, 김홍석

전체 청구항 수 : 총 6 항 심사관 : 서광훈

(54) 발명의 명칭 **CIRCUS로부터 GO 코드를 자동으로 생성하는 방법 및 장치**

(57) 요 약

입력 모듈, 실행 코드 변환 모듈, 및 모니터링 코드 변환 모듈을 포함하는 Go 코드 자동 생성 장치에서의 Circus 명세로부터 Go 코드를 자동으로 생성하는 방법이 개시된다. Go 코드 자동 생성 방법은 Circus 정형 명세 언어로 작성된 Circus 명세를 입력받는 Circus 명세 입력 단계, 및 상기 Circus 명세를 변환 함수를 이용하여 Go 코드로 자동 변환하는 Go 코드 생성 단계를 포함하되, 상기 Go 코드 생성 단계는 상기 Circus 명세에서 상태 스키마, 오퍼레이션 스키마, 또는 시스템 행위(system behavior)에 대한 명세 부분을 Go 코드의 실제 동작하는 부분인 제1 코드로 변환하는 제1 코드 변환 과정, 및 상기 Circus 명세에서 불변속성(invariants)에 대한 부분을 상기 제1 코드의 동작 중에 검증 속성 위배 여부를 모니터링하기 위한 부분인 제2 코드로 변환하는 제2 코드 변환 과정을 포함한다.

대 표 도 - 도1



(52) CPC특허분류
G06F 17/30371 (2013.01)
G06F 8/30 (2013.01)
G06F 8/443 (2013.01)
G06F 8/73 (2013.01)

이 발명을 지원한 국가연구개발사업
과제고유번호 1711026372
부처명 미래창조과학부
연구관리전문기관 정보통신산업진흥원
연구사업명 정보통신기술인력양성
연구과제명 고품질 융합 소프트웨어 개발 지원 도구 연구
기 여 율 1/1
주관기관 고려대학교 산학협력단
연구기간 2015.01.01 ~ 2015.12.31

명 세 서

청구범위

청구항 1

Circus 정형 명세 언어로 작성된 Circus 명세를 입력받는 입력 모듈,

상기 Circus 명세에서 상태 스키마, 오퍼레이션 스키마, 또는 시스템 행위(system behavior)에 대한 명세 부분을 Go 코드로 변환하는 제1 변환 모듈, 및

상기 Circus 명세에서 불변속성(invariants)에 대한 부분을 Go 코드로 제2 변환 모듈을 포함하되,

상기 제1 변환 모듈은 Go 코드의 실제 동작하는 부분인 제1 코드를 생성하고,

상기 제2 변환 모듈은 상기 제1 코드의 동작 중에 검증 속성 위배 여부를 모니터링하기 위한 제2 코드를 생성하는,

Go 코드 자동 생성 장치.

청구항 2

제1항에 있어서,

상기 제1 코드를 실행할 때, 상기 제2 코드를 동시에 실행하여 검증 속성 위배 여부를 모니터링하는 검증 모듈을 더 포함하는,

Go 코드 자동 생성 장치.

청구항 3

Go 코드 자동 생성 장치가 Circus 명세로부터 Go 코드를 자동으로 생성하는 방법에 있어서,

Go 코드 자동 생성 장치의 입력 모듈이 Circus 정형 명세 언어로 작성된 Circus 명세를 입력받는 Circus 명세 입력 단계; 및

Go 코드 자동 생성 장치의 변환 모듈이 상기 Circus 명세를 변환 함수를 이용하여 Go 코드로 자동 변환하는 Go 코드 생성 단계;를 포함하되,

상기 Go 코드 생성 단계는,

상기 Circus 명세에서 상태 스키마, 오퍼레이션 스키마, 또는 시스템 행위(system behavior)에 대한 명세 부분을 Go 코드의 실제 동작하는 부분인 제1 코드로 변환하는 제1 코드 변환 과정; 및

상기 Circus 명세에서 불변속성(invariants)에 대한 부분을 상기 제1 코드의 동작 중에 검증 속성 위배 여부를 모니터링하기 위한 부분인 제2 코드로 변환하는 제2 코드 변환 과정;을 포함하는,

Circus로부터 Go 코드를 자동으로 생성하는 방법.

청구항 4

제3항에 있어서,

Go 코드 자동 생성 장치의 검증 모듈이 상기 제1 코드를 실행할 때, 상기 제2 코드를 동시에 실행하여 검증 속성 위배 여부를 모니터링하는 모니터링 단계를 더 포함하는,

Circus로부터 Go 코드를 자동으로 생성하는 방법.

청구항 5

제4항에 있어서,

Go 코드 자동 생성 장치의 검증 모듈이 상기 Circus 명세를 정형 검증(Formal Verification)하는 단계를 더 포함하는,

Circus로부터 Go 코드를 자동으로 생성하는 방법.

청구항 6

제3항에 있어서,

상기 Go 코드 생성 단계는,

상기 Circus 명세의 시스템 프로세스는 상기 Go 코드의 메인 함수 내에서 호출되는 각각의 함수로 변환하고,

상기 Circus 명세의 채널은 상기 Go 코드의 채널로 변환하고,

상기 Circus 명세의 상태 스키마의 상태는 Go 코드의 채널로 전역 변수로 변환하고,

상기 Circus 명세의 상태 스키마의 불변속성은 Go 코드의 메인 함수 내의 고루틴(goroutine) 함수로 변환하고,

상기 Circus 명세의 오퍼레이션 스키마는 Go 코드의 함수로 변환하고,

상기 Circus 명세의 오퍼레이션 스키마 내에 선언된 상태는 Go 코드의 함수 내의 지역 변수로 변환하고,

상기 Circus 명세의 오퍼레이션 스키마 내에 선언된 입력 및 출력은 Go 코드의 함수로 입력 및 출력으로 변환하고,

상기 Circus 명세의 액션은 Go 코드의 함수로 변환하는 것을 특징으로 하는,

Circus로부터 Go 코드를 자동으로 생성하는 방법.

발명의 설명

기 술 분 야

[0001] 본 발명의 개념에 따른 실시 예는 정형명세언어인 Circus로부터 프로그램 언어인 Go 코드를 자동으로 생성하는 방법 및 장치에 관한 것으로, 특히 Circus의 상태, 오퍼레이션 및 행위 명세는 Go 코드의 실제 동작하는 부분으로 변환하고, 검증 속성인 불변속성(Invariants)은 동작하는 코드를 모니터링하기 위한 코드로 변환하는 Go코드 자동 생성 방법 및 장치에 관한 것이다.

배 경 기 술

[0002] 본 특허는 정형명세 언어인 Circus로 작성된 명세를 바탕으로 대응하는 Go 코드로 자동 변환하는 기술을 제시하고 있다. 즉, Circus 명세를 입력으로 받아 실행 가능한 Go 코드를 자동으로 생성한다.

[0003] 정형 기법은 수학이나 논리 기반의 명세와 설계의 기술을 사용하여 컴퓨터 시스템이나 소프트웨어를 개발하는 기법으로 정형 명세(Formal Specification) 및 정형 검증(Formal Verification)을 포함한다. 정형 명세는 수리논리 등을 이용하여 시스템의 동작 환경, 시스템 요구사항, 시스템의 설계 등을 기술하는 것이고, 정형 검증은 수학적, 논리적 증명방법을 통해 시스템의 설계와 시스템 요구사항의 만족 여부를 증명하는 것이다.

[0004] 정형 검증 등을 통하여 정형 명세가 제대로 동작함을 증명하였다 하더라도, 그 정형 명세를 소프트웨어 코드(소스 코드)로 구현하는 과정이 정확하지 않거나 코드가 명세와 다르게 작성되는 경우, 개발에서 의도하였던 결과물을 생성하지 못하는 문제점이 발생한다.

[0005] 특히, 정형 명세를 바탕으로 개발자가 수동으로 개발할 경우, 개발자의 실수(Human Faults)가 발생할 수

있으며, 따라서 명세대로 동작하지 않는 코드가 작성될 수 있다. 즉, 개발자가 정형 명세를 제대로 이해하지 못하거나 실수가 코드에 삽입되어 올바르지 못한 코드가 생성될 수 있는 문제점이 있다.

[0006] 따라서, 정형 명세로부터 정확한 코드를 자동으로 생성할 수 있는 코드 변환 방법이 요구된다.

[0007] 『A. Freitas and A. Cavalcanti, "Automatic Translation from Circus to Java", In FM 2006, pages 1 15-130, Springer, 2006 Circus』에서 프로그램 언어로 자동 번역하는 기술에 대해 공지하고 있으나, 프로그램 언어가 Go코드가 아닌 Java를 이용하며, 번역 시 실제 동작 부분과 모니터링 부분으로 각각 변환하지 않는다.

선행기술문헌

특허문헌

[0008] (특허문헌 0001) US 2002-0062475 A1

(특허문헌 0002) US 2009-0089234 A1

(특허문헌 0003) KR 2011-0020520 A

발명의 내용

해결하려는 과제

[0009] 본 발명이 이루고자 하는 기술적인 과제는 정형명세언어인 Circus로부터 프로그램 언어인 Go 코드를 자동으로 생성하는 방법 및 장치를 제공하는 것이다.

[0010] 또한, Circus의 불변속성(Invariants)을 모니터링 코드로 변환하고, 이를 이용하여 변환된 코드의 동작을 모니터링함으로써 신뢰성 높은 Go 코드 자동 생성 방법 및 장치를 제공하는 것이다.

과제의 해결 수단

[0011] 본 발명의 실시 예에 따른 Circus 명세로부터 Go 코드를 자동으로 생성하는 방법은 Circus 정형 명세 언어로 작성된 Circus 명세를 입력받는 Circus 명세 입력 단계, 및 상기 Circus 명세를 변환 함수를 이용하여 Go 코드로 자동 변환하는 Go 코드 생성 단계를 포함하되, 상기 Go 코드 생성 단계는 상기 Circus 명세에서 상태 스키마, 오퍼레이션 스키마, 또는 시스템 행위(system behavior)에 대한 명세 부분을 Go 코드의 실제 동작하는 부분인 제1 코드로 변환하는 제1 코드 변환 과정, 및 상기 Circus 명세에서 불변속성(invariants)에 대한 부분을 상기 제1 코드의 동작 중에 검증 속성 위배 여부를 모니터링하기 위한 부분인 제2 코드로 변환하는 제2 코드 변환 과정을 포함한다.

발명의 효과

[0012] 본 발명의 실시 예에 따른 Go 코드 자동 생성 장치는 Circus로 작성된 정형 명세를 입력받아 이에 대응하는 Go 코드를 자동으로 생성함으로써 개발자의 실수가 삽입될 가능성을 제거하고 소스 코드의 신뢰도 및 정확도를 높이는 효과가 있다.

[0013] 또한, 본 발명의 실시 예에 따른 Go 코드 자동 생성 장치는 다양한 측면에서 명세한 Circus 기반 정형 명세를 스켈레톤 단위의 코드 또는 함수 단위의 코드가 아닌 전체적인 실행 가능한 코드로 생성할 수 있다.

[0014] 또한, 본 발명의 실시 예에 따른 Go 코드 자동 생성 장치는 Circus의 불변속성(Invariants)를 이용하여 모니터링 코드를 생성함으로써, Go 코드 동작 중에 검증 속성을 위배하는지 여부를 모니터링할 수 있다.

도면의 간단한 설명

[0015] 본 발명의 상세한 설명에서 인용되는 도면을 보다 충분히 이해하기 위하여 각 도면의 상세한 설명이 제공된다.

도 1은 본 발명의 일 실시예에 Go 코드 자동 생성 장치의 기능 블럭도이다.

도 2는 Circus로 명세된 정형 명세로부터 Go 코드 변환하는 변환 개념을 도시한 도면이다.

도 3은 본 발명의 일 실시예에 따른 Circus로부터 Go 코드를 자동으로 생성하는 방법을 도시한 흐름도이다.

도 4는 본 발명의 일 실시예에 따른 변환 함수의 변환 규칙을 도시한 도면이다.

도 5는 본 발명의 일 실시예에 따른 변환 함수 동작의 일례를 도시한 도면이고, 도 6은 본 발명의 일 실시예에 따른 변환 함수의 일례를 표시한 도면이다.

도 7a는 본 발명의 일 실시예에 따른 코드 자동 생성 장치의 입력값인 Circus 명세의 일례를 도시한 도면이고, 도 7b는 도 7a의 Circus 명세를 자동 변환하여 출력한 Go 코드를 도시한 도면이다.

발명을 실시하기 위한 구체적인 내용

[0016] 본 명세서에 개시되어 있는 본 발명의 개념에 따른 실시 예들에 대해서 특정한 구조적 또는 기능적 설명은 단지 본 발명의 개념에 따른 실시 예들을 설명하기 위한 목적으로 예시된 것으로서, 본 발명의 개념에 따른 실시 예들은 다양한 형태들로 실시될 수 있으며 본 명세서에 설명된 실시 예들에 한정되지 않는다.

[0017] 본 발명의 개념에 따른 실시 예들은 다양한 변경들을 가할 수 있고 여러 가지 형태들을 가질 수 있으므로 실시 예들을 도면에 예시하고 본 명세서에서 상세하게 설명하고자 한다. 그러나, 이는 본 발명의 개념에 따른 실시 예들을 특정한 개시 형태들에 대해 한정하려는 것이 아니며, 본 발명의 사상 및 기술 범위에 포함되는 모든 변경, 균등물, 또는 대체물을 포함한다.

[0018] 제1 또는 제2 등의 용어는 다양한 구성 요소들을 설명하는데 사용될 수 있지만, 상기 구성 요소들은 상기 용어들에 의해 한정되어서는 안 된다. 상기 용어들은 하나의 구성 요소를 다른 구성 요소로부터 구별하는 목적으로만, 예컨대 본 발명의 개념에 따른 권리 범위로부터 벗어나지 않은 채, 제1 구성 요소는 제2 구성 요소로 명명될 수 있고 유사하게 제2 구성 요소는 제1 구성 요소로도 명명될 수 있다.

[0019] 본 명세서에서 사용한 용어는 단지 특정한 실시 예를 설명하기 위해 사용된 것으로서, 본 발명을 한정하려는 의도가 아니다. 단수의 표현은 문맥상 명백하게 다르게 뜻하지 않는 한, 복수의 표현을 포함한다. 본 명세서에서, "포함하다" 또는 "가지다" 등의 용어는 본 명세서에 기재된 특징, 숫자, 단계, 동작, 구성 요소, 부분품 또는 이들을 조합한 것이 존재함을 지정하려는 것이지, 하나 또는 그 이상의 다른 특징들이나 숫자, 단계, 동작, 구성 요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

[0020] 다르게 정의되지 않는 한, 기술적이거나 과학적인 용어를 포함해서 여기서 사용되는 모든 용어들은 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자에 의해 일반적으로 이해되는 것과 동일한 의미를 가진다. 일반적으로 사용되는 사전에 정의되어 있는 것과 같은 용어들은 관련 기술의 문맥상 가지는 의미와 일치하는 의미를 갖는 것으로 해석되어야 하며, 본 명세서에서 명백하게 정의하지 않는 한, 이상적이거나 과도하게 형식적인 의미로 해석되지 않는다.

[0021] 이하, 본 명세서에 첨부된 도면들을 참조하여 본 발명의 실시 예들을 상세히 설명한다.

[0022] 우선, 본 발명에서 이용하는 정형 명세 기법 및 코드에 대하여 설명한다.

[0023] 본 발명에서 코드 변환을 위하여 입력하는 정형 명세 언어는 Circus이다. Circus는 2000년 York Univ.에서 개발한 언어로써, 상태 기반 정형 명세 언어인 Z와 프로세스 알제브라(Process Algebra)인 CSP를 합친 언어이다. 시스템의 상태 및 각각의 오퍼레이션은 Z로 정의하며 시스템 전체 또는 부분 행위는 CSP를 이용하여 정의한다. Circus는 시스템 기능 명세뿐만 아니라 행위 명세를 함께 나타냄으로써 다양한 측면 명세 가능하다는 특징을 가지고 있다.

[0024] 또한, 본 발명에서 코드 변환에 결과물에 이용하는 프로그램 언어는 Go 코드(Golang이라고도 함)이다. Go 코드는 2009년 12월 Google에서 개발되었으며, C언어 기반으며 동시성(Concurrancy) 부분을 Golang언어 설계 시 CSP 언어에 기반하였다. Golang은 가비지 컬렉션 및 타입 안정성(type safety)을 제공한다. 특히, 고루틴, 채널 및 select 구문을 제공함으로써 동시성(Concurrancy)을 제공하는 특징이 있다.

[0025] 이하, 도 1 내지 도 2를 참조하여, 본 발명의 일 실시예에 따라 Circus 명세를 Go 코드로 변환하는 Go 코드 자동 변환 장치에 대해 상술한다.

[0026] 도 1은 본 발명의 일 실시예에 따른 Go 코드 자동 생성 장치(100)의 기능 블럭도이다. 도 1을 참조하면, Go 코드 자동 생성 장치(100)는 입력 모듈(110), 변환 모듈(120), 모니터링 모듈(130), 메모리(180), 및 제어 모듈

(190)를 포함한다.

[0027] 본 명세서에서 사용되는 모듈이라 함은 본 발명의 기술적 사상을 수행하기 위한 하드웨어 및 상기 하드웨어를 구동하기 위한 소프트웨어의 기능적, 구조적 결합을 의미할 수 있다. 예컨대, 상기 모듈은 소정의 코드와 상기 소정의 코드가 수행되기 위한 하드웨어 리소스의 논리적인 단위를 의미할 수 있으며, 반드시 물리적으로 연결된 코드를 의미하거나 한 종류의 하드웨어를 의미하는 것은 아니다.

[0028] Go 코드 자동 생성 장치(100)의 입력 모듈(110)은 Circus 정형 명세 언어로 명세한 Circus 명세를 입력으로 받는다.

[0029] 변환 모듈(120)은 Circus 정형 명세 언어를 Go 코드로 변환한다. 구체적으로, Circus 명세의 전체 또는 일부 명세 및 명세 내의 심볼을 입력받아 대응하는 코드를 도출하거나 해당하는 함수에 입력값을 전달하여 최종 Go 코드를 토출한다.

[0030] 변환 모듈(120)은 크게 Go 코드에서 실제 동작하는 부분의 코드를 생성하는 실행 코드 변환 모듈(121, 이하 제1 변환 모듈이라고 함) 및 동작하는 코드를 모니터링 하기 위한 코드를 생성하는 모니터링 코드 변환 모듈(123, 이하 제2 변환 모듈이라고 함)로 나뉜다.

[0031] 도 2는 Circus로 명세된 정형 명세로부터 Go 코드 변환하는 변환 개념을 도시한 도면이다. 도 2를 참조하면, 제1 변환 모듈(121)은 Circus 정형 명세 언어 중 상태 스키마(State Schema), 오퍼레이션 스키마(Operation Schema), 및 시스템 행위(System Behavior)와 관련된 명세를 Go 코드의 실제 동작하는 부분(correct program)으로 변환한다.

[0032] 제2 변환 모듈(123)는 Circus 정형 명세 언어 중 검증 속성인 불변속성(Invariants)을 실행 코드 변환 모듈(121)에 의하여 생성된 코드가 동작할 때, 즉 프로그램을 실행(execution)할 때, 동시에 동작하며 검증 속성을 위배하는지 여부를 모니터링(Runtime Mornitoring)할 수 있는 모니터링 코드(Mornitoring Code)로 변환한다.

[0033] 검증 모듈(130)은 상기 제2 변환 모듈(123)에서 생성한 모니터링 코드를 이용하여 상기 제1 변환 모듈(121)에서 생성한 프로그램 코드가 동작하는 동안, 검증 속성 위배 여부를 모니터링한다.

[0034] 메모리(180)는 프로그램 메모리와 데이터 메모리를 포함할 수 있다. 상기 프로그램 메모리에는 Go 코드 자동 생성 장치(100)의 동작을 제어하기 위한 프로그램들이 저장될 수 있다. 상기 데이터 메모리에는 상기 프로그램들을 수행하는 과정 중에 발생하는 데이터들이 저장될 수 있다.

[0035] 제어 모듈(190)은 Go 코드 자동 생성 장치(100)의 전반적인 동작을 제어한다. 즉, 입력 모듈(110), 변환 모듈(120), 검증 모듈(130) 및 메모리(180)의 동작을 제어할 수 있다.

[0036] 이하, 도 3 내지 도 7b를 참조하여, 본 발명의 일 실시예에 따른 Go 코드 자동 생성 장치를 이용한 Circus 명세로부터 Go 코드를 자동으로 생성하는 방법에 대하여 자세히 살펴보도록 한다.

[0037] 도 3은 자동으로 Circus 명세를 Go 코드로 변환하는 방법을 설명하기 위한 흐름도이다.

[0038] 먼저, Go 코드 자동 변환 장치는 Circus 명세를 입력받는다(S100). 다음, 도면에는 도시하지 않았으나, 입력된 Circus 명세를 정형 검증하는 단계를 더 포함할 수 있다.

[0039] 다음, 입력된 Circus 명세를 Go 코드로 변환한다(S200). Go 코드 변환 단계는 실행 코드 변환 단계(또는 '제1 코드 변환 단계'라고 함, S210) 및 모니터링 코드 변환 단계(또는 '제2 코드 변환 단계'라고 함, S230)를 포함한다.

[0040] 제1 코드 변환 단계(S210)에서 Circus의 상태 및 오퍼레이션과 CSP의 행위 명세를 Go 코드의 실제 동작하는 코드 부분인 제1 코드 변환하고, 제2 코드 변환 단계(S230)에서 Circus 명세 중 검증 속성인 불변속성(Invariants)을 모니터링 코드인 제2 코드로 변환한다. 제2 코드는 이후 모니터링 단계(S300)에서 검증 속성을 위배 여부를 모니터링하는데 이용한다.

[0041] 구체적으로 본 발명의 일 실시예에 따른 코드 변환 전략은 표 1과 같다.

표 1

[0042]

Translation Strategy		
	Circus	golang
1	시스템 프로세스	각각의 함수(메인 함수 내에서 콜)

2	채널	채널
3	상태 스키마의 상태	전역 변수(global variables)
4	상태 스키마의 불변속성	각각의 함수(메인 함수 내의 고루틴으로 생성)
5	오퍼레이션 스키마	함수
6	오퍼레이션 스키마 내에 선언된 상태	함수 내의 지역 변수(local variables)
7	오퍼레이션 스키마 내에 선언된 입력, 출력	함수의 입력, 출력
8	액션	함수

[0043] 본 발명의 일 실시예에 따른 코드 변환 단계(S200)에서 코드 변환은 함수들을 통해 이루어진다. 도 4는 본 발명의 일 실시예에 따른 변환 함수의 변환 규칙을 도시한 도면이다. 도 4를 참고하면, 함수들은 Circus의 전체 또는 일부 명세 및 명세 내의 심볼을 입력받아 대응하는 코드를 도출하거나 또는 해당하는 함수에 입력값을 전달한다. 상기 함수들은 반복적으로 수행함으로써 최종 Go 코드를 도출한다. 변환 함수(Translation Function)은 $|[\ ]|_x$ 로 표시한다.

[0044] 이하, 도 5 내지 도 6을 참고하여, Circus로 작성된 명세의 각각의 부분들을 Go 코드로 변환하는 과정에 대하여 살펴보도록 한다. 도 5는 본 발명의 일 실시예에 따른 변환 함수 동작의 일례를 도시한 도면이다.

[0045] 도 5의 (a)를 참조하면, 예를 들어, 전체 명세는 Program으로써 이는 AC 또는 B일 수 있다. 또한, A는 a로 B는 b로 C는 c로 문법적인(Syntactic) 변환이 이루어질 수 있다. 변환 함수를 살펴보면 전체 명세인 Program은 Program 함수에 적용된다. 이때, Program이 AC일 경우 AC 함수에 적용하며, B일 경우 B 함수에 적용한다. AC일 경우 각각 A 함수와 C 함수에 적용하며 사이에 프로그램 코드인 “&”를 추가한다. A, B, C는 다시 a, b, c 함수에 입력값 a, b, c로 적용하며 a, b, c는 각 해당 함수에 적용되어 프로그램 코드인 “Program code a”, “Program code b”, “Program code c”를 도출한다.

[0046] 도 5의 (b)를 참조하여 변환 함수를 이용하여 변환하는 과정을 살펴보면, AC라는 전체 명세를 변환할 경우, Program함수 정의에 따라 명세 AC를 AC 함수에 적용한다. AC 함수는 다시 A를 A함수에 C를 C함수에 적용하고 프로그램 코드 &를 중간에 삽입한다. A와 C는 각각 A, C 함수에 적용되어 입력값 a, c를 a, c함수에 적용하며 “Program code a”와 “Program code c”를 도출한다. 이에 최종적으로 “Program code a & Program code c”라는 코드가 도출된다.

[0047] 도 6은 본 발명의 일 실시예에 따른 변환 함수의 일례를 표시한 도면이다. 도 6에 도시된 변환 함수는 ProcDecl 함수로써 프로세스 정의에 대한 변환을 정의하고 있다. ProcDecl 함수에 “process N = ProcDef” 입력이 오면 해당하는 Go 코드('func main{' 부터 '}'까지)를 생성하고 메인 함수 내에 ProcDef 심볼을 procDef 함수에 다시 적용한다.

[0048] 이하, 도 7a 및 도 7b를 참고하여, 본 발명의 일 실시예에 따른 Go 코드 변환 장치가 Circus로 작성된 명세를 입력받아 자동으로 변환하여 출력한 Go 코드의 예를 자세히 살펴보도록 한다.

[0049] 도 7a는 입력값인 Circus로 작성된 정형 명세의 예시적인 도면이고, 도 7b는 도 7a의 Circus 명세를 입력값으로 하여 미리 정의된 변환 함수들에 의하여 변환된 결과값인 Go 코드를 도시한 도면이다.

[0050] 상태 스키마의 상태 v는 전역변수 v로 선언되며 오퍼레이션 AddOp는 함수 AddOp()로 변환된다. 또한, 프로세스의 전체 액션 (u X ? out!v -> AddOp ; X)는 함수 Accumulator1Main()함수로 정의되며 내용은 함수 내의 코드로 변환된다. 함수 v_Type은 상태 변수 v에 대한 타입 속성이며 함수 콜 앞에 go 키워드가 붙어 코드가 동작할 때 동시에 실행하여 속성 위배 여부를 검증하는 모니터링 코드로 변환된다. 만약 상태 스키마에 불변속성이 정의되어 있다면 이 역시 마찬가지로 동시에 실행하며 모니터링 코드로 변환된다.

[0051] 다음, Go 코드를 실행하고 동시에 모니터링 코드를 이용하여 속성 위배 여부를 모니터링한다(S300). 일반적으로, 소프트웨어 개발 시 우선 명세 단계에서 1차적으로 Circus 명세가 만족해야 할 속성들을 위배하는지 검증한다. 이후 코딩 단계에서 자동으로 변환된 코드의 동작 중에 만족해야 할 속성들을 위배하는지 여부를 모니터링 코드를 이용하여 모니터링함으로써, 속성 위배 여부를 2차적으로 검증할 수 있다.

[0052] 본 발명의 일 실시예에 따른 Go 코드 자동 생성 장치 및 Go 코드 자동 생성 방법을 이용하여, 보다 정확한 Go 코드를 자동으로 생성할 수 있으며, 또한, 모니터링 코드를 추가적으로 생성하여 속성 위배 여부에 대한 모니터링을 수행함으로써 보다 정확하고 신뢰도 높은 소프트웨어 개발이 가능하다.

[0053] 상술한 스케줄러가 계층적으로 존재하는 실시간 시스템을 정형 검증하는 방법은 컴퓨터에서 실행될 수 있는 프로그램으로 작성가능하고, 컴퓨터로 읽을 수 있는 기록매체를 이용하여 상기 프로그램을 동작시키는 범용 디지털 컴퓨터에서 구현될 수 있다. 상기 프로그램은 상기 기록매체에 저장되며, 상기 기록매체는 마그네틱 저장매체(예를 들면, 롬, 플로피 디스크, 하드디스크 등), 광학적 판독 매체(예를 들면, 시디롬, 디브이디 등)와 같은 저장매체를 포함한다. 또한, 상기 기록매체는 네트워크로 연결된 컴퓨터 시스템에 분산되어 분산방식으로 컴퓨터가 읽을 수 있는 명령어 세트가 저장되고 실행될 수 있다.

[0054] 본 발명에서 개시된 블록도들은 본 발명의 원리들을 구현하기 위한 회로를 개념적으로 표현한 형태라고 당업자에게 해석될 수 있을 것이다. 유사하게, 임의의 흐름 차트, 흐름도, 상태 전이도, 의사코드 등은 컴퓨터 판독가능 매체에서 실질적으로 표현되어, 컴퓨터 또는 프로세서가 명시적으로 도시되든지 아니든지 간에 이러한 컴퓨터 또는 프로세서에 의해 실행될 수 있는 다양한 프로세스를 나타낸다는 것이 당업자에게 인식될 것이다.

[0055] 본 발명은 도면에 도시된 실시 예를 참고로 설명되었으나 이는 예시적인 것에 불과하며, 본 기술 분야의 통상의 지식을 가진 자라면 이로부터 다양한 변형 및 균등한 타 실시 예가 가능하다는 점을 이해할 것이다. 따라서, 본 발명의 진정한 기술적 보호 범위는 첨부된 등록청구범위의 기술적 사상에 의해 정해져야 할 것이다.

부호의 설명

[0056] 100 : Go 코드 자동 생성 장치

110 : 입력 모듈 120 : 변환 모듈

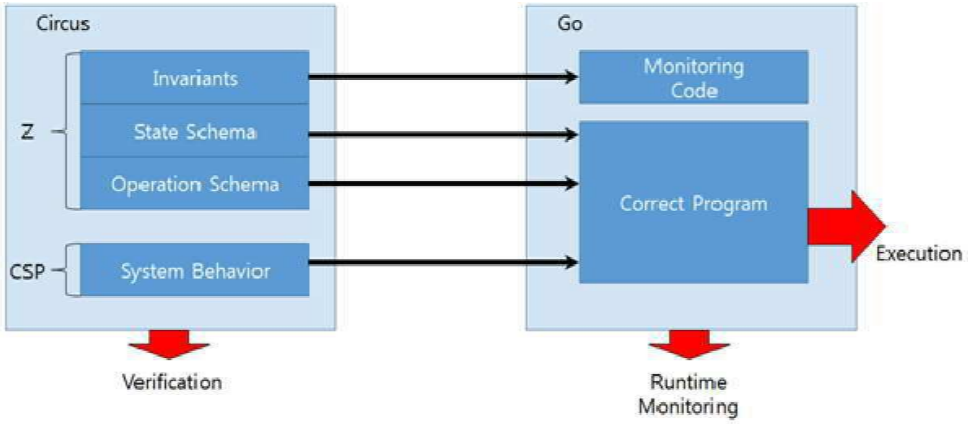
130 : 모니터링 모듈 180 : 메모리

도면

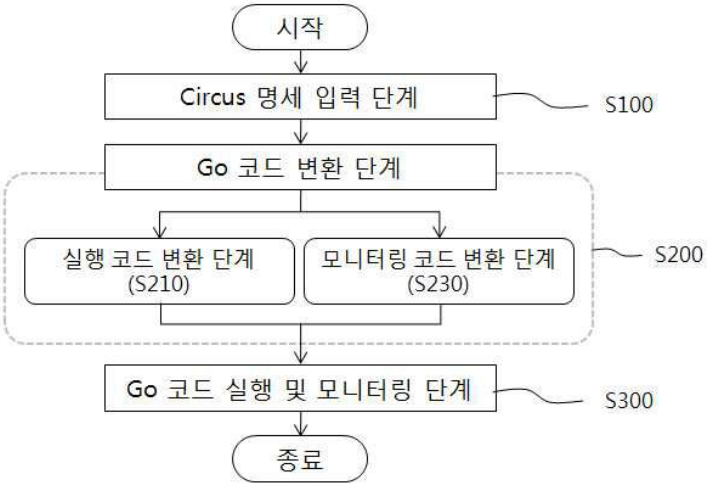
도면1



도면2

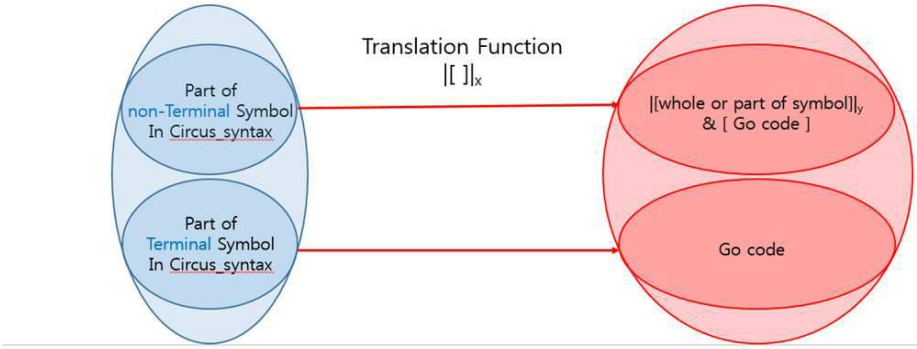


도면3

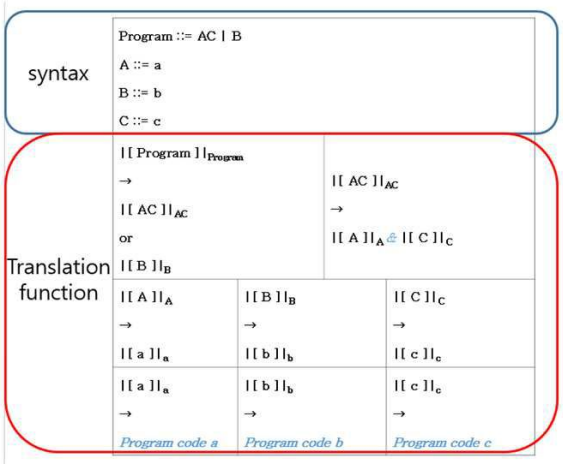


도면4

|[_]| : symbol in circus → Go_code



도면5



Program = AC
|[Program]|Program
→
|[AC]|AC
→
|[A]|A & |[C]|C
→
|[a]|a & |[C]|C
→
Program code a & |[C]|C
→
Program code a & |[c]|c
→
Program code a & Program code c

도면6

```
|[ process N ≝ ProcDef ]|ProcDecl
    → func main {
            go NMain()
      }
      func NMain(){
            |[ ProcDef ]|procDef
      }
```

도면7a

$$
\begin{aligned}
&\textbf{channel } out : \mathbb{N} \\
&\textbf{process } Accumulator1 \mathrel{\widehat=} \\
&\textbf{begin} \\
&\quad \textbf{state} \\
&\quad State \mathrel{\widehat=} [v : \mathbb{N}] \\
&\quad AddOp \mathrel{\widehat=} [\Delta State \mid v' = v + 1] \\
&\quad \bullet (\mu X \bullet out!v \rightarrow AddOp \,;\, X) \\
&\textbf{end}
\end{aligned}
$$

도면7b

```
package main

import (
    "fmt"
    "runtime"
//  "math/rand"
    "time"
)

var out = make(chan int, 10)

var v int

func v_Type(){

    if( v < 0 ) {
        fmt.Print( "V error")
    }
}

func AddOp(){
    v = v + 1
}

func Accumulator1Main(){

    for{
        out <- v
        AddOp()
    }
}

func main(){

    runtime.GOMAXPROCS(runtime.NumCPU())

    go Accumulator1Main()
    go v_Type()


    time.Sleep(time.Second * 10)
}
```