(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2009-0064374
(43) 공개일자 2009년06월18일

(51) Int. Cl.
G06F 7/76 (2006.01) *G06F 9/00* (2006.01)
G06F 17/00 (2006.01)
(21) 출원번호 10-2009-7004780
(22) 출원일자 2009년03월06일
심사청구일자 없음
번역문제출일자 2009년03월06일
(86) 국제출원번호 PCT/US2007/078540
국제출원일자 2007년09월14일
(87) 국제공개번호 WO 2008/034086
국제공개일자 2008년03월20일
(30) 우선권주장
11/532,374 2006년09월15일 미국(US)

(71) 출원인
마이크로소프트 코포레이션
미국 워싱턴주 (우편번호 : 98052) 레드몬드 원 마이크로소프트 웨이
(72) 발명자
루코, 스티븐 이.
미국 98052-6399 워싱톤주 레드몬드 원 마이크로소프트 웨이
랑워시, 데이비드 이.
미국 98052-6399 워싱톤주 레드몬드 원 마이크로소프트 웨이
델라-리버라, 지오바니 엠.
미국 98052-6399 워싱톤주 레드몬드 원 마이크로소프트 웨이
(74) 대리인
양영준, 백만기

전체 청구항 수 : 총 20 항

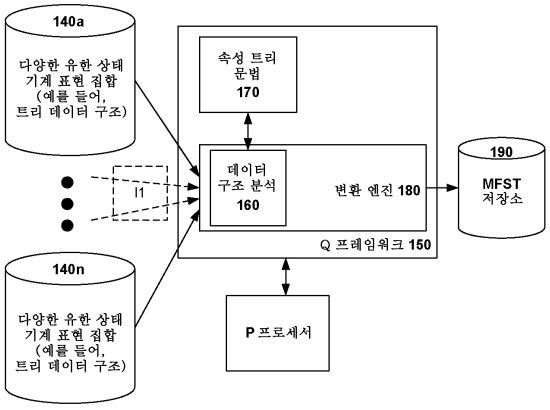
(54) 데이터 구조를 모듈형 유한 상태 변환기로 변환하는 방법, 컴퓨터 판독가능 매체, 변환 프레임워크, 및 컴파일러

(57) 요 약

FST에 대한 다양한 일련의 유형의 표현들에 걸쳐 그의 표현에서 액션 정보를 지원하는 FST에 대한 액션 의미의 보존을 달성하는 일반화된 방식으로 효율적인 트리 변환을 수행하는 Q 프레임워크(QFX라고 약칭함)가 제공된다.

다른 특징들 중에서도 특히, QFX는 또한 트리 변환을 수행하는 동안 순서있는 내포 정보 및 순서없는 내포 정보의 보존을 가능하게 해주고, 비결정론적 데이터 구조의 결정론적 데이터 구조로의 변환을 지원하며, 액션 의미를 갖는 기계들에 대한 교집합 연산을 가능하게 해준다.

대 표 도 - 도1c



특허청구의 범위

청구항 1

컴퓨팅 시스템에서 적어도 하나의 트리 구조를 규정하는 적어도 하나의 데이터 구조를 적어도 하나의 모듈형 유한 상태 변환기(MFST)로 변환하는 방법으로서,

순서있는 정보 및 순서없는 정보를 정의하는 의미를 포함하는 적어도 하나의 유한 상태 변환기(FST)를 표현하는 적어도 하나의 트리 구조(140a)를 규정하는 적어도 하나의 데이터 구조를 수신하는 단계(500), 및

상기 적어도 하나의 데이터 구조(140a)로 표현되는 임의의 유형의 유한 상태 기계(FSM)에 대해, 상기 적어도 하나의 데이터 구조의 상기 순서있는 정보 및 상기 순서없는 정보를 적어도 하나의 MFST에 보존하면서 상기 적어도 하나의 데이터 구조(140a)를 상기 적어도 하나의 MFST로 변환하는 단계(510)를 포함하며,

상기 변환하는 단계는 상기 적어도 하나의 데이터 구조에 대해 교집합, 합집합 및 여집합 연산 중 임의의 것을 수행하는 단계를 포함하는 것인, 컴퓨팅 시스템에서 적어도 하나의 트리 구조를 규정하는 적어도 하나의 데이터 구조를 적어도 하나의 모듈형 유한 상태 변환기로 변환하는 방법.

청구항 2

제1항에 있어서, 상기 변환하는 단계(510)는, 상기 적어도 하나의 데이터 구조(140a)에 의해 규정된 상기 적어도 하나의 트리 구조로 표현되는 FSM 모델의 유형에 상관없이, 상기 순서있는 정보 및 상기 순서없는 정보를 상기 적어도 하나의 MFST에 보존하는 변환 문법(translation grammar)(170)을 사용하여 상기 적어도 하나의 데이터 구조(140a)를 변환하는 단계를 포함하는 것인, 컴퓨팅 시스템에서 적어도 하나의 트리 구조를 규정하는 적어도 하나의 데이터 구조를 적어도 하나의 모듈형 유한 상태 변환기로 변환하는 방법.

청구항 3

제1항에 있어서, 상기 변환하는 단계(510)는 임의의 표현 유형에 대해 적어도 하나의 데이터 구조(140a)의 순서있는 의미 정보 및 순서없는 의미 정보를 보존하는 상기 적어도 하나의 데이터 구조(140a)로 표현된 상기 FST에 대해 적어도 하나의 제어 흐름 분석 알고리즘을 수행하는 단계를 포함하는 것인, 컴퓨팅 시스템에서 적어도 하나의 트리 구조를 규정하는 적어도 하나의 데이터 구조를 적어도 하나의 모듈형 유한 상태 변환기로 변환하는 방법.

청구항 4

제1항에 있어서, 상기 변환하는 단계(510)는, 상기 적어도 하나의 데이터 구조(140a)에 의해 규정되는 상기 적어도 하나의 트리 구조로 표현되는 FSM 모델의 유형에 상관없이, 상기 적어도 하나의 데이터 구조(140a)의 순서있는 내포 정보 및 순서없는 내포 정보 둘다를 보존하는 미리 정의된 변환 문법(170)을 사용하여 상기 적어도 하나의 데이터 구조(140a)를 변환하는 단계를 포함하는 것인, 컴퓨팅 시스템에서 적어도 하나의 트리 구조를 규정하는 적어도 하나의 데이터 구조를 적어도 하나의 모듈형 유한 상태 변환기로 변환하는 방법.

청구항 5

제1항에 있어서, 상기 수신하는 단계(500)는 목록 및 집합 패턴 정보를 정의하기 위한 순서있는 의미 및 순서없는 의미를 포함하는 상기 적어도 하나의 FST를 표현하는 적어도 하나의 유향 그래프 구조를 규정하는 적어도 하나의 데이터 구조(140a)를 수신하는 단계를 포함하는 것인, 컴퓨팅 시스템에서 적어도 하나의 트리 구조를 규정하는 적어도 하나의 데이터 구조를 적어도 하나의 모듈형 유한 상태 변환기로 변환하는 방법.

청구항 6

제4항에 있어서, 상기 수신하는 단계(500)는 순서있는 정보 및 순서없는 정보를 정의하기 위한 의미를 포함하는 상기 적어도 하나의 FST를 표현하는 적어도 하나의 XML(extensible markup language) 문서를 규정하는 적어도 하나의 데이터 구조(140a)를 수신하는 단계를 포함하는 것인, 컴퓨팅 시스템에서 적어도 하나의 트리 구조를 규정하는 적어도 하나의 데이터 구조를 적어도 하나의 모듈형 유한 상태 변환기로 변환하는 방법.

청구항 7

제1항에 있어서, 상기 변환하는 단계(510)는 적어도 하나의 데이터 구조(140a)에 대해 비어있는 내포 테스트(empty nest test)를 수행하는 단계를 포함하는 것인, 컴퓨팅 시스템에서 적어도 하나의 트리 구조를 규정하는 적어도 하나의 데이터 구조를 적어도 하나의 모듈형 유한 상태 변환기로 변환하는 방법.

청구항 8

제1항에 있어서, 상기 변환하는 단계(510)는 상기 순서있는 정보 및 상기 순서없는 정보를 보존하면서 적어도 2개의 데이터 구조에 대해 구조 호환성 테스트(structural compatibility test)를 수행하는 단계를 포함하는 것인, 컴퓨팅 시스템에서 적어도 하나의 트리 구조를 규정하는 적어도 하나의 데이터 구조를 적어도 하나의 모듈형 유한 상태 변환기로 변환하는 방법.

청구항 9

제1항에 있어서, 상기 수신하는 단계(500)는 적어도 하나의 비결정론적 트리 데이터 구조를 규정하는 적어도 하나의 데이터 구조(140a)를 수신하는 단계를 포함하고,

상기 변환하는 단계(510)는 상기 적어도 하나의 데이터 구조를 결정론적 MFST로 변환하는 단계를 포함하는 것인, 컴퓨팅 시스템에서 적어도 하나의 트리 구조를 규정하는 적어도 하나의 데이터 구조를 적어도 하나의 모듈형 유한 상태 변환기로 변환하는 방법.

청구항 10

제1항의 방법을 수행하는 컴퓨터 실행가능 명령어들을 포함하는 컴퓨터 판독가능 매체.

청구항 11

제1항의 방법을 수행하는 수단을 포함하는 컴퓨팅 장치.

청구항 12

컴퓨팅 시스템에서 유한 상태 변환기(FST)를 표현하는 유향 그래프 데이터 구조를 모듈형 유한 상태 변환기(MFST)로 변환하는 변환 프레임워크로서,

컴퓨팅 시스템에서 복수의 유향 그래프 데이터 구조(140a)의 순서있는 의미 정보 및 순서없는 의미 정보를 포함하는 FST를 표현하기 위한 다양한 유형의 복수의 유향 그래프 데이터 구조(140a)를 저장하는 수단, 및

미리 정해진 트리 문법(170)에 기초하여 상기 복수의 유향 그래프 데이터 구조(140a)를 분석하고 상기 복수의 유향 그래프 데이터 구조(140a)의 상기 순서있는 의미 정보 및 상기 순서있는 의미 정보를 보존하면서 상기 복수의 유향 그래프 데이터 구조(140a)를 적어도 하나의 MFST로 변환하는 변환기(150)를 포함하며,

상기 변환기(150)는 상기 복수의 유향 그래프 데이터 구조(140a)에 대해 교집합, 여집합 및 합집합 연산 중 임의의 것을 수행하는 것인, 컴퓨팅 시스템에서 유한 상태 변환기를 표현하는 유향 그래프 데이터 구조를 모듈형 유한 상태 변환기로 변환하는 변환 프레임워크.

청구항 13

제12항에 있어서, 상기 변환기(150)는 임의의 유향 그래프 표현 유형에 걸쳐 상기 복수의 유향 그래프 데이터 구조의 순서있는 의미 정보 및 순서없는 의미 정보를 보존하는 상기 복수의 유향 그래프 데이터 구조(140a)로 표현된 상기 FST에 대해 적어도 하나의 제어 흐름 분석 알고리즘을 수행하는 것인, 컴퓨팅 시스템에서 유한 상태 변환기를 표현하는 유향 그래프 데이터 구조를 모듈형 유한 상태 변환기로 변환하는 변환 프레임워크.

청구항 14

제12항에 있어서, 상기 변환기(150)는 상기 복수의 유향 그래프 데이터 구조(140a)를 변환할 때 특수 바인딩 처리를 수행하는 변환 엔진(180)을 포함하며,

상기 특수 바인딩 처리는 상기 복수의 유향 그래프 데이터 구조(140a)의 패턴들을 정합시키기 위한 패턴 정합 프로세스 동안에 상기 순서있는 의미 정보 및 상기 순서없는 의미 정보에 대한 바인딩을 수행하는 단계를 포함하는 것인, 컴퓨팅 시스템에서 유한 상태 변환기를 표현하는 유향 그래프 데이터 구조를 모듈형 유한 상태 변환

기로 변환하는 변환 프레임워크.

청구항 15

제12항에 있어서, 상기 변환기(150)는 상기 순서있는 의미 정보 및 상기 순서없는 의미 정보를 보존하면서 상기 복수의 유향 그래프 데이터 구조(140a)의 유향 그래프 데이터 구조로 정의된 바대로 적어도 하나의 변환기 정의를 인라인시키는 것인, 컴퓨팅 시스템에서 유한 상태 변환기를 표현하는 유향 그래프 데이터 구조를 모듈형 유한 상태 변환기로 변환하는 변환 프레임워크.

청구항 16

제12항에 있어서, 상기 변환기(150)는 상기 적어도 하나의 데이터 구조(140a)를, 상기 적어도 하나의 데이터 구조(140a)에 포함된 순서있는 내포 정보 및 순서없는 내포 정보 둘다를 포함하는 적어도 하나의 MFST에 매칭시키는 변환 엔진 컴포넌트(180)를 포함하는 것인, 컴퓨팅 시스템에서 유한 상태 변환기를 표현하는 유향 그래프 데이터 구조를 모듈형 유한 상태 변환기로 변환하는 변환 프레임워크.

청구항 17

제16항에 있어서, 상기 적어도 하나의 데이터 구조(140a)는 적어도 하나의 비결정론적 트리 구조를 규정하고,

상기 변환 엔진 컴포넌트(180)는 상기 적어도 하나의 데이터 구조(140a)를 결정론적 MFST로 변환하는 것인, 컴퓨팅 시스템에서 유한 상태 변환기를 표현하는 유향 그래프 데이터 구조를 모듈형 유한 상태 변환기로 변환하는 변환 프레임워크.

청구항 18

제12항에 있어서, 상기 미리 정의된 트리 문법(170)에 따라 유향 그래프 데이터 구조(140a)를 정의 및 변환하는 상기 복수의 유향 그래프 데이터 구조(140a)의 유향 그래프 데이터 구조를 수신하는 인터페이스(I1)를 더 포함하는, 컴퓨팅 시스템에서 유한 상태 변환기를 표현하는 유향 그래프 데이터 구조를 모듈형 유한 상태 변환기로 변환하는 변환 프레임워크.

청구항 19

컴퓨팅 시스템에서 유한 상태 변환기(FST)를 표현하는 유향 그래프 데이터 구조를 모듈형 유한 상태 변환기(MFST)로 변환하는 컴파일러로서,

컴퓨팅 시스템에서 프로세스에 대한 순서있는 목록 및 순서없는 집합을 정의하는 순서있는 정보 및 순서없는 정보를 포함하는 프로세스에 대한 FST를 표현하는 복수의 유향 그래프 데이터 구조(140a), 및

정의된 트리 문법(170)에 기초하여 상기 복수의 유향 그래프 데이터 구조(140a)를 분석하고 상기 복수의 유향 그래프 데이터 구조(140a)의 순서있는 정보 및 순서없는 정보를 보존하면서 상기 복수의 유향 그래프 데이터 구조(140a)를 적어도 하나의 MFST로 변환하는 변환 엔진(180)을 포함하며,

상기 변환 엔진(180)은 상기 적어도 하나의 MFST를 발생하기 위해 상기 복수의 유향 그래프 데이터 구조(140a)에 대해 교집합, 여집합 및 합집합 연산 중 임의의 것을 수행하는 것인, 컴퓨팅 시스템에서 유한 상태 변환기를 표현하는 유향 그래프 데이터 구조를 모듈형 유한 상태 변환기로 변환하는 컴파일러.

청구항 20

제19항에 있어서, 상기 복수의 유향 그래프 데이터 구조(140a)는 순서있는 정보 및 순서없는 정보를 표현하기 위한 다양한 FST 모델 유형들에 따라 표현되는 것인, 컴퓨팅 시스템에서 유한 상태 변환기를 표현하는 유향 그래프 데이터 구조를 모듈형 유한 상태 변환기로 변환하는 컴파일러.

명 세 서

기 술 분 야

<1> 본 발명은 일반적으로 유한 상태 변환기(finite state transducer) 등의 유한 상태 오토마타(finite state automata)에 관한 것으로서, 보다 구체적으로는 그의 변환에 관한 것이다.

배 경 기 술

<2> 일반적 배경으로서, 유한 상태 오토마타(finite state automata, FSA)라고도 하는 유한 상태 기계(finite state machine, FSM)는 상태, 천이 및 액션으로 이루어진 시스템의 거동 모델이다. FSM 또는 FSA는 기계 또는 오토마타를 구성하는 상태들, 천이들 및 액션들의 관계를 보존하는 상태도, 상태 천이도, 상태 테이블, 라벨 유향 그래프(labeled directed graph), 트리, 기타 등등을 사용하여 다양한 개념적 방식으로 표현될 수 있다. 또한, 서로 다른 종류의 FSM 또는 FSA, 이들 간의 등가성, 및 이들 중 어떤 것들 간의 여러가지 알려진 변환이 있다.

<3> 현재, 새로운 FST 또는 일련의 FST, 예를 들어, 새로운 유향 그래프 데이터 구조(들)를 형성하기 위해, 예를 들어, 어떤 유형의 유향 그래프 데이터 구조(들)에 대해, FST에 의해 모델링된 일련의 시스템을 변환하는 것의 어떤 측면들을 달성하는 몇몇 그래프 변환기 및 트리 변환기가 존재하지만, 이러한 시스템들은 여러가지 이유로 제한이 있다. 예를 들어, XSLT(Extensible Stylesheet Language Transformation)는 XML 문서의 변환에 사용되는 XML-기반 언어이다. 다른 XML 문서를 변환하는 데 사용될 때, 문서가 변경되지 않고, 오히려, 기존의 XML 문서의 내용에 기초하여 새로운 문서가 생성된다. 새로운 문서는 프로세서에 의해 표준 XML 구문으로 또는 다른 형식(HTML 또는 일반 텍스트 등)으로 직렬화(출력)될 수 있다. XSLT는 종종 서로 다른 XML 스키마 간에 데이터를 변환하는 데 또는 XML 데이터를 웹 페이지 또는 PDF 문서로 변환하는 데 사용된다.

<4> 본질적으로, XSLT는 다른 XML 문서(다른 형식으로 변환될 수도 있음)의 변환을 기술하는 XML 문서의 생성을 가능하게 해준다. 그렇지만, XSLT는 FST로 표현될 수 있는 액션 의미를 고려하지 않고 있다. 이 점에서, 액션은 "미리 정의된 상태에 있을 때, 어떤 정보(예를 들어, 이름)가 인식되는 경우, 어떤 액션을 수행함"일 수도 있다. 그렇지만, FST에 대해 이러한 임의의 액션들이 정의될 수 있는 한, XSLT는 이러한 임의의 액션들에 대한 호출을 그의 변환 기능의 일부로서 처리할 수 없다. 그에 부가하여, XSLT는 기계들에 대한 여집합, 교집합 및 합집합 연산 모두를 수행할 수 없다.

<5> XSLT 등의 또다른 기존의 변환도 역시 순서있는 계층적 정보 및 순서없는 계층적 정보[(예를 들어, 트리 구조로 표현되어 있는) 순서있는 내포(ordered nest) 및 순서없는 내포(unordered nest) 등] 둘다를 갖는 FSM을 매칭시켜 구성하지 못한다. 어떤 상황에서, 트리를 변환할 때 순서있는 정보만의 보존을 처리할 수 있는 몇몇 시스템들이 있고 변환을 할 때 순서없는 정보만의 보존을 처리할 수 있는 몇몇 시스템들이 있지만, XSLT 및 임의의 다른 공지된 시스템 어느 것도 유향 그래프 또는 트리 구조에 걸쳐 순서있는 내포 및 순서없는 내포 둘다를 보존할 수 있는 변환 기능을 포함하고 있지는 않다. 종래 기술의 유한 상태 변환기의 변환의 이들 및 다른 결점들이 이하에 더 상세히 기술되는 본 발명의 여러가지 예시적인 비제한적 실시예의 설명에서 명백하게 될 것이다.

발명의 상세한 설명

<6> 종래 기술의 상기한 단점들을 고려하여, 본 발명은 다양한 일련의 유형의 표현들에 걸쳐 그의 표현에서 액션 정보를 지원하는 FST에 대한 변환 연산들에 걸쳐 액션 의미를 보존하는 FST에 대한 효율적인 트리 변환(예를 들어, 교집합, 합집합, 여집합, 기타)을 제공하는 일반적인 프레임워크를 제공한다. FST에 대한 효율적인 트리 변환을 수행하는 프레임워크는 또한 트리 변환을 수행하는 동안 순서있는 내포 정보 및 순서없는 내포 정보를 보존할 수 있고 비결정론적 데이터 구조의 결정론적 데이터 구조로의 변환을 지원할 수 있다.

<7> 일 실시예에서, 본 발명은 컴퓨팅 시스템에서 트리 구조(들)를 규정하는 데이터 구조(들)를 모듈형 유한 상태 변환기(들)(modular finite state transducer, MFST)로 변환하는 방법을 제공한다. 이 방법은, 데이터 구조(들)에 의해 표현되는 임의의 유형의 유한 상태 기계(finite state machine, FSM) 모델에 대한, 순서있는 정보 및 순서없는 정보를 정의하는 의미를 포함하는 유한 상태 변환기(들)(FST)를 표현하는 트리 구조(들)를 규정하는 데이터 구조(들)를 수신하는 단계를 포함하고, 이 방법은 MFST에 상기 데이터 구조(들)의 상기 순서있는 정보 및 상기 순서없는 정보를 보존하면서 상기 데이터 구조(들)를 MFST로 변환한다. 이 변환하는 단계는 상기 데이터 구조(들)에 대해 교집합, 합집합 및 여집합 연산 중 임의의 것, 또는 교집합, 합집합 및 여집합 연산 중 임의의 것으로 환산될 수 있는 임의의 연산을 수행하는 단계를 포함한다. 본 발명의 다른 측면에서, 결과 FSM이 비어있지 않은 입력(non-empty input)을 받는지가 결과 FSM으로부터 결정될 수 있다.

<8> 이하의 상세한 설명 및 첨부 도면에 기술되어 있는 예시적이고 비제한적인 실시예들의 다양한 측면들에 대한 기본적인 또는 전반적인 이해를 가능하게 해주는 데 도움을 주기 위해 간략화된 요약이 본 명세서에 제공되어 있다. 그렇지만, 이 요약은 포괄적이거나 전수적인 개요로서 제공된 것이 아니다. 이 요약의 유일한 목적은 본

발명의 여러가지 예시적이고 비제한적인 실시예들에 관련된 몇몇 상위 레벨 개념들을 이하의 상세한 설명에 대한 서문으로서 간략화된 형태로 제공하기 위한 것이다.

실 시 예

<36> 본 발명의 모듈형 유한 상태 변환기 및 연관된 프로세스의 트리 문법을 변환하는 기법에 대해 첨부 도면을 참조하여 상세히 기술한다.

<37> 배경 기술에서 언급한 바와 같이, FST 및 트리 데이터 구조(예를 들어, 라벨 유향 그래프)를 변환하는 현재의 프레임워크는, 이 데이터 구조가 컴퓨팅 메모리에 표현되는 경향이 있기 때문에, 모든 유형의 FST에 걸쳐 변환을 할 때 어떤 유형의 정보를 보존하기에 충분할 정도로 다목적이지 못하다. 예를 들어, 현재의 프레임워크는 어떤 FST의 액션 의미, 또는 여러 유형의 FST에 걸쳐 FST에 의해 표현되는 순서있는 내포 정보 및 순서없는 내포 정보 둘다를 보존하지 못한다.

<38> 그에 따라, 본 발명은 그의 표현에서 액션 정보를 지원하는 FST에 대한 액션 의미의 보존을 달성하는 효율적인 트리 변환을 수행하는 일반적 프레임워크[본 명세서에서 Q 프레임워크(즉, QFX)라고 함]를 제공한다. 본 발명의 효율적인 트리 변환을 수행하는 Q 프레임워크는 또한 그의 효율적인 트리 변환을 수행하는 동안 순서있는 내포 정보 및 순서없는 내포 정보를 보존할 수 있다. 이하의 설명으로부터 잘 알 것인 바와 같이, QFX는 또한 정의된 트리 문법 및 대응하는 변환(translation)에 대한 변환 기능, 변환기 다이어그램, 변환 엔진(transformation engine) 및 변환 엔진에 의해 수행되는 연산, 변환 사례(translation case), 및 변수 이름 변경을 포함한 결정화(determinization)와 관련한 다수의 다른 새로운 측면들을 가능하게 해주며, 이에 대해서는 이하에서 더 상세히 설명한다. 본 발명의 일 실시예에서, 본 발명은 트리 표현 또는 일련의 트리 표현에 대한 적어도 교집합, 합집합, 여집합, 및 비어있음 테스트(emptiness test) 변환을 가능하게 해준다.

<39> 추가적으로 소개하면, 모듈형 유한 상태 변환기(MFST)는 변환을 구현하는 상태 기계이다. 상세하게는, MFST는 "액션"이라고 하는 프로그램 프래그먼트로 보강된 MFA이다. 통상의 액션으로는 환경에 저장된 다른 객체들로부터 객체를 생성하는 것, 환경 내의 변수들을 바인딩하는 것, 및 패턴-정합 동안에 서술자(predicate)를 실행하는 것이 있다.

<40> 이와 관련하여, 상위 레벨에서, 본 발명의 Q 프레임워크(QFX)는 프로그래머가 액션들을 사용하여 정규의 트리 문법을 정의하고 이들 문법을 MFST로 변환할 수 있게 해준다. QFX는 액션을 정의하기 위해 임의의 DSL(Domain Specific Language)을 사용하는 것을 지원한다. 그에 부가하여, QFX는 속성 트리 문법(attributed tree grammar)이라고 하는 일종의 트리 문법을 생성하는 변수-바인딩 액션에 대한 특별한 지원을 제공한다. 본 발명의 이하의 예시적이고 비제한적인 구현예(들)는 QFX가 속성 트리 문법을 어떻게 컴파일하고 실행할 수 있는지에 대해 설명한다.

<41> 이와 관련하여, 본 발명의 컴파일 방식은 여러가지 이유로 유익하다. 이 컴파일 방식은 최적화 및 다른 고성능 방식을 통한 고성능 변환에 대한 지원을 제공한다. 컴파일 기법은 또한 간단한 변환 기계의 생성이 용이하게 달성될 수 있도록 런타임 컴포넌트의 간단화를 가능하게 해준다. 이하에서 예시적이고 비제한적으로 상세히 기술되는 바와 같이, 본 명세서에 기술된 컴파일 기법은 명확한 의미 집합을 제공한다. 이와 관련하여, 본 발명의 컴파일 방식은 프로그래머가 트리 문법의 의미(합성 및 부수-효과를 포함함)에 관해 추론하는 것을 쉽도록 만들어준다.

<42> 일반적으로, 도 1a의 흐름도에 나타낸 바와 같이, 본 발명은 컴퓨팅 시스템에서 액션 의미 정보를 갖는 다양한 유형의 유한 상태 변환기(FST) 데이터 구조를 수신하고(단계 100), 이어서 프레임워크의 변환 엔진을 FST에 적용하는(단계 105) 일반적 프레임워크를 가능하게 해준다. 이어서, 단계(110)에서, 임의의 유형의 FST 표현에 대한 사전 정의된 트리 문법에 따라 FST의 분석 및 변환이 행해지며, 이에 대해서는 이하에서 더 상세히 설명한다. 그 결과, 그 중에서도 특히, 예를 들어, 2개 이상의 FST에 대해 교집합, 여집합 및 합집합 연산 중 임의의 것을 수행하는 동안 FST의 액션 정보를 보존하고, 순서있는 내포 정보 및 순서없는 내포 정보를 보존하며, 비결정론적 FST의 결정론적 결과로의 변환을 가능하게 해주는 MFST를 생성하기(단계 115) 위해 FST가 변환된다.

<43> 도 1b는 본 발명의 트리 문법에 의해 가능하게 되는 한 측면을 나타내는 상기 프로세스에 일반적으로 대응하는 블록도이다. 변환 프레임워크(125)는 액션 의미를 포함하는 하나 이상의 MFST(120)를 수신하고, 교집합, 합집합 및 여집합 변환 연산(및/또는 합성을 갖는 다른 부울 연산자) 중 임의의 것에 따라 이 MFST를 변환하며, 출력(130)에 액션 의미 정보가 보존되어 있다.

<44> 도 1c는 본 발명에 따라 FST를 변환하는 예시적인 프레임워크 및 변환 엔진을 나타낸 예시적이고 비제한적인 블록도를 나타낸 것이다. 도시된 바와 같이, 컴퓨팅 시스템 내의 다양한 저장 요소(140a - 140n)(본 발명에 따라 제공되는 Q 프레임워크(150)와 상호작용하는 프로세서(P)를 갖는 컴퓨팅 환경 내의 어디에서라도 발견됨)에서 다양한 종류의 유한 상태 기계 표현이 발견될 수 있다. 일반적으로, Q 프레임워크(150)는, MFST(190)를 변환 및 생성하는 변환 엔진(180)을 통해, 유한 상태 기계 표현(140a - 140n)의 부분집합(들)을 수신한다. 이 변환은 유한 상태 기계 표현(140a - 140n)의 부분집합(들)을 MFST(190)로 가장 잘 변환하는 방법을 결정하기 위해 속성 트리 문법(170)을 참조하여 유한 상태 기계 표현(140a - 140n)의 부분집합(들)에 대한 분석(160)을 수행한다. 미리 정의된 트리 문법(170)에 기초한 것으로서 이 미리 정의된 트리 문법(170)에서 사용하기 위한 유향 그래프 데이터 구조를 수신 또는 정의하기 위한 인터페이스(I1)도 컴퓨팅 환경 내에 제공되어 있을 수 있다.

<45> 따라서, 다양한 비제한적인 실시예에서, 본 발명은 컴퓨팅 시스템에서 트리 데이터 구조의 모듈형 유한 상태 변환기(MFST)로의 변환을 가능하게 해준다. FSM(예를 들어, FST)을 표현하는 트리 데이터 구조는 변환 프로세스에 걸쳐 보존되는 FSM에 관한 액션 정보를 정의하는 액션 의미를 포함한다. 일 실시예에서, 트리 데이터 구조(예를 들어, 유향 그래프 데이터 구조, XML 문서, 기타)로 표현되는 임의의 유형의 FSM 모델에 대해, 본 발명은 액션 정보를 보존하면서 트리 데이터 구조를 MFST로 변환한다. 트리 데이터 구조로 표현되는 FSM 모델의 유형에 상관없이 액션 정보를 보존하고, 트리 데이터 구조의 순서있는 내포 정보 및 순서없는 내포 정보 둘다를 보존하며, 비결정론적 데이터 구조를 결정론적 MFST로 변환하는 변환 엔진에 의해 확장가능한 사전 정의된 변환 문법이 구현된다. 유익하게도, 본 발명은 변환 프로세스의 일부로서 FSM의 유형의 여집합, 교집합 및 합집합을 처리할 수 있다.

<46> 추가의 실시예들에서, 본 발명은 유향 그래프의 모듈형 유한 상태 변환기(MFST)(컴퓨팅 시스템에서 임의의 유형의 유향 그래프 데이터 구조를 분석하고 유향 그래프 데이터 구조의 액션 의미를 보존하면서 이 분석에 기초하여 유향 그래프 데이터 구조를 MFST로 변환하는 기능을 포함함)로의 변환을 수행하는 컴포넌트 및 소프트웨어를 포함하는 Q 프레임워크(QFX)를 포함한다. QFX는 임의의 유형의 유향 그래프 표현에 걸쳐 유향 그래프 데이터 구조의 액션 의미 정보를 보존하는 방식으로 유향 그래프 데이터 구조로 표현된 유한 상태 기계(FSM)에 대해 제어 흐름 분석 알고리즘(들)을 수행한다.

<47> 그에 부가하여, QFX는 유향 그래프 데이터 구조를 변환하는 것과 관련하여 특수화된 바인딩 처리를 수행하는 변환 엔진을 포함한다. 이 특수화된 바인딩 처리는 패턴 정합 동안에 액션 의미 정보에 대한 바인딩을 수행하는 것을 포함한다. 게다가, 이 변환은 유향 그래프 데이터 구조들 중 한 유향 그래프 데이터 구조로 정의된 적어도 하나의 변환기 정의를 인라인(in-line)하는 기능을 포함한다. 또한, 이 변환은 유향 그래프 데이터 구조의 기존의 활성화 레코드(activation record) 내의 슬롯에의 변수 바인딩(variable binding)을 컴파일하는 것을 포함하고 유향 그래프 데이터 구조로 표현되는 유한 상태 기계(FSM)의 레지스터 분석(register analysis)도 포함한다. 본 발명의 상기 효과들 및 이점들을 달성하는 본 발명에 의해 정의되는 문법은 액션 의미 정보를 포함하는 유향 그래프 데이터 구조를 기술하는 C# 인터페이스 등의 소프트웨어 인터페이스를 통해 구현될 수 있다.

<48> 그에 부가하여, 여러가지 실시예들에서, 본 발명은 변수 바인딩을 사용하여 어휘 분석을 구현한다. 또한, QFX는 확장가능하며, 서로 다른 변환 제공자들과의 효율적인 상호 운용성을 제공한다. 게다가, 본 발명은 컴파일과 실행 성능 간의 트레이드오프에 대한 제어를 구현한다, 즉 구현들이 고비용의 컴파일 없이 패턴을 해석하고 반복 사용을 위해 특정의 패턴을 컴파일할 수 있다. 마지막으로, 런타임 컴포넌트는 가변-입도 연속점(variable-granularity continuation point)을 제공하여 동일한 메카니즘을 사용해 단일의 결과 또는 다수의 결과를 제공할 수 있게 해줌으로써 QFX 스케쥴링 규칙(scheduling discipline)에 참여할 수 있다.

<49> 이하의 표 1의 다음과 같은 간단화된 트리 문법 구문은 본 발명에 따른 컴파일 프로세스와 관련하여 이용될 수 있는 예시적이고 비제한적인 모델이다.

<50> <표 1> 간단화된 트리 문법 구문

TreeGrammar → Definition^{+}

Definition → RegexName '=' Choice

Choice → Rule ('|' Rule)* (OR)

Rule → Action? RuleTerm^{+} (ACTION-REGEX)

RuleTerm → NestedRegex Action? (REGEX-ACTION)

<51> NestedRegex → BaseSymbol (BASE)

| VariableBinding? Wildcard (WILD)

| Reference (REF)

| NestedRegex NestedRegex (SEQUENCE)

| VariableBinding? '[' NestedRegex ']' (BIND-NEST)

| VariableBinding? '(' NestedRegex ')' (BIND-GROUP)

| NestedRegex ('+' | '*' | '?') (REPEAT)

| NestedRegex 'and' NestedRegex (AND)

Reference → VariableBinding? RegexName (BIND-REF)

<52> VariableBinding → Variable ':'

<53> 상기의 문법에서, 적용될 수 있는 예시적인 연산자 우선순위는 REPEAT > BIND > SEQUENCE > AND > OR이다. 이하의 설명에서, x는 변수를 나타내는 데 사용되고, b는 기본 심볼(base symbol)(기본 유형의 유형 또는 값)을 나타내는 데 사용되며, T는 MFST를 나타내는 데 사용되고, α는 액션을 나타내는 데 사용되며, $T\,Ref$는 T에 대한 참조를 나타내는 데 사용된다.

<54> 정규 표현식의 NFS로의 변환의 기술 분야와 관련하여 또한 NFA의 결정화(determinization)와 관련하여, 이러한 변환 기법의 전반적인 개요가, 예를 들어, Aho 등의 "Compilers: Principles, Techniques and Tools," Addison- Wesley (1986)에서 또는 Hopcroft 등의 "Introduction to Automata Theory, Language and Computation," Addison- Wesley (2000)에서 찾아볼 수 있다. 이러한 최신 기술의 기법에 대한 개선으로서, QFX에 따르면, 본 발명의 기법을 통해 정의되는 정규 트리 문법(regular tree grammar)이 액션을 사용하여 MFST로 확장될 수 있다.

<55> 또한, Aho 등의 저서에서 논의된 변환은 상기의 문법으로부터의 다음과 같은 사례들(OR, AND, BASE, SEQUENCE, REPEAT 및 REF)을 포함시키기 위한 것이지만, 본 발명에 따른 (1) 액션, (2) 내포 및 (3) 변수 바인딩을 처리하기 위해, 이하의 표 2에 나타낸 바와 같이 표기된 다음과 같은 부가의 사례들에 대한 변환이 요망된다.

<56> <표 2> QFX에 의해 처리되는 강화된 변환 사례 집합

$\alpha\,T$ (ACTION-REGEX)

$T\,\alpha$ (REGEX-ACTION)

$x:[T]$ (BIND-NEST)

$x:[T]^{*|+|?}$ (BIND-NEST-REPEAT)

$x:(T)$ (BIND-GROUP)

$x:(T)^{*|+|?}$ (BIND-GROUP-REPEAT)

$x:T\,Ref$ (BIND-REF)

<57> $x:T\,\mathrm{Ref}^{*|+|?}$ (BIND-REF-REPEAT)

<58> ACTION-REGEX 및 REGEX-ACTION 사례들은 일반적인 액션을 사용하여 트리 문법의 컴파일을 처리한다. BIND-

REF, BIND-GROUP, BIND- NEST, BIND-REF-REPEAT, BIND-GROUP-REPEAT, 및 BIND-NEST-REPEAT 사례들은 변수 바인딩의 컴파일을 처리한다. BIND-NEST 사례는 또한 정규 표현식을 내포된 정규 표현식(nested regular expression)으로 확장시킨다.

<59> 표 1의 간단화된 트리 문법 구문에 나타내어져 있지는 않지만, 본 발명에 따른 비말단 정의(nonterminal definition)는 정규 파라미터(regular parameter) 및 유형 파라미터(type parameter) 둘다를 가질 수 있다. 일 실시예에서, 컴파일러는 정규 파라미터를 상속 속성(inherited attribute)으로서 처리하고, 유형 파라미터를 BIND-REF 및 BIND-REF-REPEAT 변환을 사용하여 처리한다.

<60> 본 발명에 따른 MFST로의 변환은 MFST를 정의하는 상태 기계도를 사용하여 예시적으로 기술될 수 있다. 상태 기계도에서, 천이는 다음과 같이 b/α로서 라벨이 붙여져 있으며, 여기서 b는 소비되는 입력 심볼이고 α는 수행되는 액션이며, $\in/\alpha$는 어떤 입력도 소비하지 않지만 어떤 액션을 수행하는 천이를 나타낸다. 이러한 부류의 천이를 액션 천이(action transition)라고 한다. 본 발명의 일 실시예에서는, 변환기를 실행함에 있어서, 어떤 상태로부터의 액션 천이에 그 상태로부터의 모든 입력-소비 천이보다 더 낮은 우선순위를 할당하는 최대 진행 정책(maximal progress policy)이 구현된다. 상태 기계도에서, 특수 기호 '[' 및 ']'는 각각 입력 컬렉션(input collection)의 처음과 끝을 나타낸다. 변수 바인딩은 x = expr로서 쓰여져 있고 변환기의 현재의 환경에 적용된다.

<61> 반복 연산자, 즉 '+','*', 및 '?'의 경우, 결과들의 컬렉션이 목록(list)을 사용하여 누적되지만, 이 메카니즘이, 예를 들어, 대응하는 조인 연산자(join operator)를 사용함으로써, 결과들의 임의의 컬렉션 유형으로의 누적을 처리하도록 일반화될 수 있다.

<62> 본 발명의 여러가지 실시예들에 따르면, BIND-REF 및 BIND-REF-REPEAT 사례는 2가지 방식(즉, 인라인(inline) 및 호출(call))으로 확장된다. 인라인 확장은 공간을 시간과 교환하며, 따라서, 본 발명의 비제한적인 한 구현예에서, MFST를 결정화할 때는 인라인 확장(inline expansion)이 선호된다. 일반적으로, 호출 확장(call expansion)은 3가지 경우에 사용된다. 첫번째 경우는 해석을 위해 컴파일할 때이다. 이 경우에, 컴파일러는 MFST를 결정화하지 않고 모든 참조를 호출로서 확장한다. 두번째 경우는 재귀(recursion)이다. 예를 들어, T Ref가 시작 심볼(start symbol)을 참조하는 경우, 컴파일러는 참조를 호출로서 확장한다. 세번째 경우는 연장(extension)이다. T Ref가 불명료한 MFST(opaque MFST)를 참조하는 경우, 컴파일러는 참조를 호출로서 확장한다.

<63> 본 발명의 컴파일러는 트리 문법을 변환 엔진(transformation engine, XE)이라고 하는 가상 기계에 대한 명령어로 변환한다. XE는 (1) 변환기의 정의 및 동작, (2) 다른 변환 엔진 인스턴스로의 제어의 이전, 및 (3) 환경의 관리 및 환경에의 액세스를 지원한다.

<64> 첫번째 카테고리와 관련하여, 상태 기계 정의를 존속시키고 해석하는 기법들은 공지되어 있다. 그에 따라, 이하에서는 두번째 및 세번째 카테고리, 즉 이전 및 관리를 달성하는 예시적이고 비제한적인 방법들에 대해 기술한다. 본 발명에 따르면, 이들 명령어는 한쌍의 인터페이스에서 메서드로서 정의된다. 컴파일러는 이들 인터페이스에 대한 호출을 직접 발생하지 않으며, 오히려 컴파일러는 XE 구현에 의해 해석되는 XE 명령어를 발생한다.

<65> 본 발명에 따라 제공되는 XEControlInstructions 인터페이스는 도 2a의 예시적인 의사-코드(200)에 나타내어져 있다. XEControlInstructions 인터페이스(200)는 현재의 XE 인스턴스를 암시적 피연산자(implicit operand), 즉 XEControlInstructions 인터페이스(200)의 'this 포인터'로서 갖는다.

<66> QFX의 예시적인 실시예에서 제공되는 Mark 메서드는 마크 스택(mark stack) 상의 입력 항(input term)에 현재 위치를 기록한다. Yield 메서드는 마크 M을 마크 스택으로부터 팝(pop)하고 M와 현재 위치 사이의 입력 항의 일부분을 반환한다. 일 실시예에서, QFX는 순회-제공자(traversal-provider) 인터페이스를 통해 우회적으로 이들 메서드를 구현한다. 이러한 방법은 유익하게도 항 표현 및 순회를 항 변환(term transformation)으로부터 분리시킨다. 순회 제공자는 따라서 QFX 변환 프레임워크를 임의의 특정의 데이터 표현에 적용함으로써 순회를 구현할 수 있다.

<67> 본 발명에 따라 제공된 Call 메서드는 새로운 환경(E)을 생성하고 continuation 및 callingEnvironment을 E에 저장한다(예를 들어, 도 2a의 XEControlInstructions 인터페이스를 참조할 것). 이어서, Call 메서드는 제어를 타겟으로 넘기는 continuation를 반환한다. Call 메서드는 또한 현재의 입력 위치를 저장하기 위해 Mark 메서

드를 호출한다. Return 메서드는 이 동작을 반대로 하며 Yield 메서드를 사용하여 일치된 항을 현재의 환경(E)에 저장하는 동작을 한다, 즉 E.term=Yield()이다. 이어서, Return 메서드는 E를 변수 callingEnvironment.result에 저장하고, 마지막으로 Return 메서드는 continuation을 반환한다. 호출될 때, continuation은 현재의 환경을 callingEnvironment로 설정하고 호출 상태(calling state)로부터 계속한다.

<68> NewEnvironment 메서드는 본 발명에 따라 새로운 환경을 생성한다. 그에 부가하여, 도 2b 및 도 2c의 예시적이고 비제한적인 의사-코드(210) 및 의사-코드(220)는 각각 PushEnvironment 및 PopEnvironment 메서드의 동작을 개략적으로 나타낸 것이다.

<69> PushEnvironment 메서드는 현재의 환경을 내부 환경 스택(internal environment stack) 상에 저장하고 새로운 환경(예를 들어, tempEnv라고 명명됨)을 생성한다. PushEnvironment는 이어서 tempEnv를 variableName에 바인딩하고, 현재의 환경을 tempEnv로 설정하며, 현재의 입력 위치를 기록하기 위해 Mark를 호출한다.

<70> PopEnvironment 메서드는 이전의 Mark 호출 이후에 일치되는 서브항(sub-term)을 변수 "term"에 기록한다. PopEnvironment는 이어서 환경 스택(environment stack)으로부터 현재의 환경을 복구한다. PopEnvironment는 팝된 환경을 저장하지 않는데, 그 이유는 PushEnvironment에 대한 대응하는 호출이 팝된 환경을 이미 바인딩하고 있기 때문이다.

<71> 그에 부가하여, Exec 메서드는 본 발명에 따라 액션을 실행한다. QFX의 예시적인 구현예에서, ActionReference는 별도로 존속되어 로딩된 메서드 포인터 테이블에 대응하는 테이블 인덱스이다.

<72> 본 발명에 따라 제공된 XEEnvironmentInstructions 인터페이스는 도 2d의 예시적인 의사코드(230)에 나타내어져 있다. XEEnvironmentInstructions 인터페이스(230)는 명시적 피연산자, 즉 XEEnvironmentInstructions 인터페이스(230)의 "this 포인터"로서 환경 인스턴스(environment instance)를 갖는다.

<73> ChildEnv 메서드는 this 포인터의 'this'를 부모로 갖는 새로운 내포된 환경을 생성한다. Bind 메서드는 변수를 값에 바인딩시키며, 이 바인딩이 성공적인 경우 true를 반환한다. 예시적이고 비제한적인 실시에에서, variableName이 현재의 바인딩을 갖는 경우, Lookup 메서드는 값을 그 바인딩으로 설정하고 true를 반환한다. 그렇지 않은 경우, false를 반환한다.

<74> 본 발명의 변환 엔진(XE)의 동작은 상태 천이에 대한 우선순위의 할당을 지원한다. 컴파일러는 각각의 상태에서 가능한 천이들을 순서화함으로써 패턴들 중의 선택항목들로부터 야기되는 모호성을 해결한다. 비제한적인 일 실시예에서, 보통의 입력에서의 모든 천이가 와일드카드 천이보다 우선하며, 이 와일드카드 천이는 액션 천이보다 우선한다. 또한, 이들 우선순위 그룹들 내에서, 프로그래머는 선택적으로 특정의 천이에 우선순위를 할당할 수 있다.

<75> XE 인스턴스는 어떤 상태로부터 계속하기 위해 모든 적용가능한 천이를 사용하는 모드를 지원하여 continuation을 우선순위 순서로 큐잉함으로써 다수의 결과의 발생을 해결할 수 있다.

<76> 도 3a는 본 발명에 따라 정의된 강화된 변환 사례 집합의 αT 사례의 예시적인 변환을 나타낸 것이다. 보다 구체적으로는, 도 3a는 ACTION-REGEX 사례의 변환을 나타낸 것이다. 이 변환은 NestedRegex SEQUENCE 사례에 대한 변환의 적용이다. REGEX-ACTION 사례가 나타내어져 있지 않은 이유는 SEQUENCE 변환의 간단한 적용이기 때문이다. 도면에서, 원(SC1, SC2 및 SC3)은 시작 상태를 나타내고, 원(IC1, IC2 및 IC3)은 중간 상태를 나타내며, 이중원(DC1, DC2 및 DC3)은 인정 상태(accepting state)를 나타낸다.

<77> 도 3b는 BIND-NEST 사례의 예시적인 변환, x : [T]를 나타낸 것이다. 도 3b에서, 메서드 PushEnvironment 및 PopEnvironment는 각각 PushEnv 및 PopEnv으로 축약되어 있다. 시작 상태(SC4), 중간 상태(IS3, IS4) 및 인정 상태(DC4)는 도 3b에 의해 정의된 기계의 서로 다른 상태를 나타낸다. a[를 읽을 때, 기계는 새로운 환경(E)을 생성하고 이를 x에 바인딩한다. 기계는 이어서 현재의 환경(C)을 푸시하고 E를 현재와 환경으로 한다. 기계는 또한 현재의 입력 위치를 표시한다. 이어서, 기계는 T를 실행하여 E 내의 변수들을 바인딩한다. T의 실행 동안에, 변수 바인딩이 E에 누적된다. 이어서, 기계는 [T]에 의해 일치된 항을 E.term에 저장하고 C를 현재의 환경으로서 복원한다.

<78> 도 3c는 BIND-NEST-REPEAT 사례의 예시적인 변환, $x : [T]^*|^+|^?$을 나타낸 것이다. 도 3c는 컴파일러가 반복을 사용하여 내포된 정규 표현식을 어떻게 변환하는지를 나타낸 것이다. 시작 상태(SC5), 중간 상태(IS5, IS6, IS7, IS8) 및 인정 상태(DC5)는 도 3c에 의해 정의된 기계의 서로 다른 상태를 나타낸다. 상태(IS5, IS6,

IS7, IS8)에 의해 정의된 변환기의 내부 부분이 도 3b의 기계와 동일하다는 것을 알 수 있다. 시작 상태(SC5) 및 인정 상태(DC5)를 포함하는 컨테이너 상태 기계(container state machine)가 이 기계를 둘러싸고 있다. 컨테이너 기계의 목표는 환경들을 변수(x)에 바인딩된 목록으로서 누적하는 것이다.

<79> 이와 관련하여, 먼저, 변환기는 비어있는 목록을 생성하고 이를 x에 바인딩한다. 이어서, 변환기는 [T]를 실행한 결과를 x에 첨부시킨다. 실행 및 첨부 단계는 Use this arc for '*' and '?'로 표시되어 있는 전방향 ∈ 천이(forward ∈ transition)에 의해 선택적으로 된다. 실행 및 첨부 단계는 Use this arc for '*' and '+'로 표시되어 있는 역방향 ∈ 천이(backward ∈ transition)의 순회에 의해 선택적으로 반복될 수 있다.

<80> 도 3d는 x : (T) 사례의 예시적인 변환을 나타낸 것이다. 시작 상태(SC6), 중간 상태(IS9, IS10) 및 인정 상태(DC6)는 도 3d에 의해 정의된 기계의 서로 다른 상태를 나타낸다. 도 3d는, 도 3d에서 T 내부로의 천이 및 T 외부로의 천이가 입력 심볼을 소비하지 않는다는 점을 제외하고는, 도 3b의 BIND-NEST 사례에 대한 변환기와 유사한 변환기를 나타낸 것이다.

<81> 도 3e는 x : T Ref 사례(호출을 사용한 확장임)의 예시적인 변환을 나타낸 것이다. 도 3e는 시작 상태(SC7), 중간 상태(IS11) 및 인정 상태(DC7)를 포함하는 기계를 정의하며, x : T Ref를 확장하기 위해 호출 전략을 사용하는 것을 나타낸 것이다. 첫번째 천이는 이하의 예시적이고 비제한적인 의사-코드에 대한 단축형인 호출 명령어 Call(T)를 실행한다.

<82> Call(T, CurrentContinuation, CurrentEnvironment)

<83> 호출 시점에서의 CurrentContinuation은 천이 화살표의 끝에 있는 중간 상태(IS11)이다. 호출된 변환기는 변수 결과에 그의 환경을 반환한다. 마지막으로, 호출하는 변환기는 결과를 x에 바인딩한다.

<84> 도 3f는 $x : T\ Ref^{*|+|?}$ 사례(호출을 사용하는 확장임)의 예시적인 변환을 나타낸 것이다. 도 3f는 시작 상태(SC8), 중간 상태(IS12, IS13) 및 인정 상태(DC8)를 포함하는 기계를 정의하며, 호출에 적용되는 반복 컨테이너(repetition container)를 나타내고 있다. 이 변환기는 도 3c의 변환기와 유사하며, T의 인라인 확장 대신에 T에 대한 호출을 내부 변환기(inner transducer)로서 갖는다.

<85> 도 3a 내지 도 3f에서 지금까지 기술되지 않은 변환 사례는 BIND-GROUP-REPEAT와 BIND-REF 사례의 인라인 버전이다. BIND-GROUP-REPEAT는 도 3f의 반복 컨테이너를 도 3d에 도시된 BIND-GROUP에 대한 변환기와 결합시킨다. 그에 부가하여, BIND-REF 및 BIND-REFREPEAT 사례의 인라인 버전은 각각 BIND-GROUP 및 BIND-GROUP-REPEAT 사례와 동일하다.

<86> 결정화(determinization)는 본 발명에 따라 액션을 가질 수 있는 천이를 고려하기 위해 부분집합 생성 기법이 어떻게 공식화될 수 있는지를 기술한다. 본 발명에 따르면, 액션과 천이 및 상태 둘다와의 연관이 가능하게 된다. 천이에 대해서만 액션을 갖는 결정론적 변환기가 생성될 수 있지만, 그러한 선택은 알고리즘을 더욱 복잡하게 만드는 반면, 상태 및 천이 둘다에 대한 액션을 효율적으로 저장하는 것이 간단하다.

<87> 어떤 상세들을 간단화하면, 도 2e는 QFX로 구현되는 예시적이고 비제한적인 ∈-클로저 알고리즘, 즉 State<T>.EClosure+을 나타낸 것이다. 이 메서드는 ∈ 천이에 대한 액션들을 누적하도록 보강되어 있다. 도 2e의 예시적인 의사-코드(240)에서 (A1) 및 (A2)로 표시된 라인은 ∈-클로저 생성 동안에 순회되는 입실론 천이에 대한 모든 액션들을 목록 'actions'에 저장한다. 전체적인 결정화 알고리즘은 이들 저장된 액션들을 ∈-클로저로부터 생성된 결정론적 유한 상태 변환기(deterministic finite state transducer, DFST) 상태로 전달한다.

<88> 도 2f는 QFX로 예시적으로 구현된 액션들을 수용하도록 수정된 부분집합 생성, 즉 NFA<T>.SubsetConstruction을 나타낸 것이다. 라벨 (B1) 내지 (B6)는 액션들을 수용하도록 수정되거나 추가된 도 2f의 예시적인 의사-코드(250)의 라인들을 표시한 것이다. 라인 (B1) 및 (B2)은 소스 상태 (src)가 심볼에 대한 천이를 갖는 상태들 모두를 발견하고, 이들 라인은 또한 각각의 천이의 액션들을 symActions에 누적시킨다. 라인 (B3) 및 (B4)는 ∈ 천이에 대한 유사한 기능을 수행한다. 마지막으로, 라인 (B5)은 symActions을 src 상태와 dst 상태 간의 DFST 천이에 추가하고, 라인 (B6)은 epsActions을 src 상태에 추가한다. 이들 액션은 이어서 src로의 천이의 완료시에 수행된다. 일 실시예에서, 이들 알고리즘은 액션들이 현재의 환경에서 변수의 바인딩 이외에 부수 효과를 갖지 않는 것으로 가정한다.

<89> 전반적으로, 유익하게도, 본 발명에 따른 이 결정화 방법은 이하의 것들을 보장해준다.

<90> 1. 한쌍의 액션 $\alpha 1$ 및 $\alpha 2$가 동일한 생성의 요소인 경우, 액션 $\alpha 1$ 및 $\alpha 2$은 어휘 순서로 호출된다.

<91> 2. 액션 $\alpha 1$이 생성 p1의 일부이고 액션 $\alpha 2$가 생성 p2의 일부인 경우(단, $p1 \neq p2$임), 변환기는 $\alpha 1$ 또는 $\alpha 2$ 중 어느 하나를 호출한다. 변환기는 p1 및 p2가 프리픽스 *pre*를 공유하고 *pre*가 p1 내의 $\alpha 1$을 포함하며 *pre*가 p2 내의 $\alpha 2$를 포함하는 경우 양 액션을 호출할 수 있다.

<92> 3. 문법에 대한 시작 정의 내에서, 각각의 생성의 최종 액션은 부수 효과를 가질수 있다. 일 실시예에서, 변환기는 모든 다른 액션들이 완료될 때까지 이 액션을 실행하지 않는다.

<93> 이 점에서, 속성 트리 문법은 프로그래머가 상기 보장들을 이용할 수 있게 해준다. 속성 문법을 사용하여, 프로그래머는 현재의 환경 내에서 모든 액션들을 변수 바인딩으로서 표현할 수 있다. 이러한 액션들은 2가지 형태, 즉 *x* : *T* 또는 *x* = *표현식* 중 하나를 취할 수 있으며, 여기서 *표현식*은 현재의 환경으로부터 값을 읽을 수 있거나 값을 계산하기 위해 코드를 호출할 수 있다. 표현식에 의해 실행되는 코드가 순서 의존성을 도입하지 않는 한, 트리 문법을 실행한 결과는 결정론적이다.

<94> 프로그래머는 부수 효과가 호출되어야 하는 것이 명확할 때까지 부수 효과를 지연시키기 위해 속성 문법을 사용할 수 있다. 예를 들어, 트리 인쇄 텍스트를 순회하기 위해, 프로그래머는 속성들을 사용해 텍스트를 모아서 문자열로 만들 수 있고 이어서 트리 문법의 시작 심볼과 연관된 최종 액션으로부터 그 문자열을 인쇄할 수 있다. 프로그래머는 또한 명확한 비말단 정의에 대한 인쇄 액션을 지연시킴으로써 "동작 중에" 서브트리를 인쇄하기로 할 수 있다.

<95> 본 발명의 예시적이고 비제한적인 실시예에 따른 변수 이름 변경과 관련하여, 변수-바인딩 액션의 독립적을 보장하기 위해, 컴파일러는 MFST에 바인딩되어 있는 각각의 변수가 고유하도록 변수들의 이름을 변경한다. 컴파일러가 또한 주어진 액션에 대한 코드를 발생하고 있는 경우, 컴파일러는 α 내의 변수들도 역시 이름을 변경한다. 코드 블록 B가 컴파일러에 불투명한 경우, 컴파일러는 최초의 변수 이름들을 그의 갱신된 이름을 통해 간접적으로 탐색하는 환경을 B에 전달하도록 한다.

<96> 선택적으로, 변수 이름 변경은 생존 범위(live range) 분석을 수행하고 몇개의 최초의 변수가 동일한 갱신된 이름을 공유하게 함으로써 최적화될 수 있다. 또한, 변환기 성능이 변수 참조를 컴파일러-결정 오프셋을 사용하여 어레이 액세스로서 구현함으로써 향상될 수 있다.

<97> 그에 따라, 한 측면에서, 본 발명은 다양한 일련의 FST 표현들에 걸쳐 그의 표현들에서 액션 정보를 지원하는 FST에 대한 액션 의미의 보존을 달성하는 효율적인 트리 변환을 수행하는 일반적인 프레임워크를 제공한다. 교집합, 합집합 및 여집합 변환 연산, 및 합성을 갖는 기타 부울 연산자 중 임의의 것이 액션 의미를 보존하면서 FST에 대해 수행될 수 있다.

<98> 그에 부가하여, 배경 기술에서 언급한 바와 같이, 제한된 상황 하에서, 트리 변환들에 걸쳐 순서있는 내포 정보를 보존할 수 있는 몇몇 시스템 및 트리 변환들에 걸쳐 순서없는 내포 정보를 보존할 수 있는 몇몇 시스템이 있지만, 트리 변환들(예를 들어, 교집합, 합집합, 여집합, 기타)에 걸쳐 순서있는 트리 정보 및 순서없는 트리 정보 둘다를 보존할 수 있는 시스템이 아직 없다. 그에 따라, 다양한 비제한적인 실시예들에서, 효율적인 트리 변환을 수행하는 프레임워크는 또한 FST에 대한 트리 변환을 수행하는 동안 순서있는 내포 정보 및 순서없는 내포 정보를 보존한다.

<99> 순서있는 정보와 순서없는 정보의 구별이 도 4a 및 도 4b에 개념적으로 나타내어져 있다. 도 4a는 목록 패턴(list pattern)이라고도 하는 순서있는 패턴을 나타낸 것으로서, 이 패턴은 패턴을 정합시키기 위해 red, 그 다음에 green, 그 다음에 blue, 그 다음에 다시 red의 시퀀스를 필요로 한다. 따라서, 패턴 정합 관점에서, 패턴의 요소들이 나타나는 순서가 고려된다. 도 4B는 집합 패턴(set pattern)이라고도 하는 순서없는 패턴을 나타낸 것이다. 도 4b의 집합 패턴은 도 4a의 목록 패턴과 동일한 요소를 보여주고 있지만, 이 때 어떤 순서도 없다. 도 4b의 집합 패턴은 따라서 순서가 어떻든지 간에 2개의 red, 하나의 green 및 하나의 blue의 집합을 나타낸다. 패턴 정합 시에 고려되는 것은 집합 내에 요소들이 있는지, 즉 요소들이 양 트리에 있는지이며, 요소들이 어떤 순서로 있는지가 아니다.

<100> 컴퓨팅 시스템에서 목록 패턴 정합 시나리오의 일례는 패스워드의 입력일 수 있으며, 이 경우 패스워드는 순차적인 일련의 숫자들이다. 패스워드에 대해 입력된 각각의 문자가 사용자에 대해 시스템에 저장된 정확한 패스

워드와 대조하기 위해 특정의 순서로 입력되어야만 하기 때문에, 패스워드 정합 시나리오는 순서있는 정보를 갖는 트리 구조에 기초하여 패턴을 정합시킨다.

<101> 컴퓨팅 시스템에서 집합 패턴 정합 시나리오의 일례는 파일 시스템에서 일련의 지정된 파일(예를 들어, Pic_Amy, Pic_Greg, Pic_Neyda)을 검색하는 것일 수 있다. 이들 사진이 있는 폴더를 검색할 때, 정합되는 것은 이들이 폴더에 있는지이며 폴더에 어느 순서로 있는지가 아니다. 환언하면, 사용자는 검색 시나리오에서 이들 파일 각각을 갖는 폴더를 찾고 싶을 뿐이며, 이 사진들이 시스템에 저장되어 있는 순서는 사용자에게 중요하지 않다. 집합 패턴 정합 시나리오의 다른 예는 데이터베이스에서 어떤 이름, 중간 이름 및 성을 찾아내는 것이며, 이 경우 결과는 데이터가 "성, 이름, 중간 이름", "이름, 중간 이름, 성" 또는 "이름, 성, 중간 이름"으로 저장되어 있는지에 의존하지 않는다. 세개의 이름 모두가 어떤 순서로든지 있기만 하면 데이터베이스 질의를 만족시킨다.

<102> 본 발명의 프레임워크에 따르면, 교집합, 합집합 및 여집합 변환 중 임의의 것을 포함하는 트리 데이터 구조로 표현되는 순서있는 정보 및 순서없는 정보 둘다를 보존하면서 MFST에 대해 변환이 수행될 수 있다. 따라서 트리의 자식들에 관한 집합 가정 및 목록 가정을 겸비한 패턴들에 대해 변환이 수행될 수 있다. 따라서, 트리 구조의 노드들에 내포되어 있는 순서있는 정보 또는 순서없는 정보가 변환들에 걸쳐 보존된다.

<103> 도 5a의 흐름도에 나타낸 바와 같이, 본 발명은 컴퓨팅에서 순서있는 정보 및/또는 순서없는 정보를 포함하는 다양한 유형의 유한 상태 변환기(FST) 데이터 구조를 수신(500)하고 이어서 본 발명의 프레임워크의 변환 엔진을 FST에 적용(505)하는 일반적인 프레임워크를 가능하게 해준다. 510에서, 임의의 유형의 FST 표현에 대한 상기한 트리 문법에 따라 FST의 분석 및 변환이 행해진다. 그 결과, FST가 변환되어, 예를 들어, 2개 이상의 FST에 대해 교집합, 여집합 및 합집합 연산 중 임의의 것을 수행하는 동안 FST의 액션 정보를 보존하고 순서있는 내포 정보 및 순서없는 내포 정보를 보존하며, 비결정론적 FST의 결정론적 결과로의 변환을 가능하게 해주는 MFST를 발생(515)한다.

<104> 도 5b는 상기 프로세스에 일반적으로 대응하는 블록도로서, 본 발명의 트리 문법에 의해 가능하게 되는 이 측면을 나타낸 것이다. 변환 프레임워크(530)는 순서있는 정보, 순서없는 정보, 또는 둘다를 각각 포함하는 하나 이상의 MFST(520, 522, 또는 524)를 수신하고, 교집합, 합집합 및 여집합 변환 연산(및/또는 합성을 갖는 기타 부울 연산자) 중 임의의 것에 따라 MFST를 변환하며, 순서있는 정보 및/또는 순서없는 정보가 출력(540)에 보존된다.

<105> 도 5c 내지 도 5i는 순서있는, 즉 라벨있는 정보의 존재 시에 상태 기계에 대한 변환의 예시적이고 비제한적인 측면을 나타낸 것이다. 도 5c의 블록도에 나타낸 바와 같이, 쟁점은 각자의 기계(M1, M2)에 대한 서로 다른 인정 상태(accept state)(550, 555)를 갖는 2개의 상태 기계(M1, M2)의 합집합, 즉 기계들의 변환 표현에서 이 정보를 어떻게 표현할지에 관한 것이다. 과거에는, 이것이 2개의 인정 상태(550, 555)를 이 둘을 표현하는 하나의 인정 상태로 결합함으로써 달성되었으며, 그 결과 원래의 기계들로부터의 정보가 손실되었다.

<106> 도 5d의 블록도에 따르면, 예시적이고 비제한적인 측면에서, 본 발명은 결과로 얻어진 변환 기계(560)에 홈 상태 기계의 라벨있는 인정 상태(home state machine labeled accept state)를 받아들인다. 여러가지 비제한적인 실시예에서, 본 발명은 변환 동안에 라벨있는 합집합 U_L의 처리를 도입하며, 이는 합집합 기계의 인정 상태에 인정 상태의 출처인 기계의 이름으로 라벨을 붙인 합집합의 변형이다.

<107> 도 5e의 예에 나타낸 바와 같이, $M_1 U_L M_2$을 수행할 때, 인정 상태 라벨의 3가지 가능한 경우, 즉 "M_1"(575), "M_1, M_2"(580) 및 "M_2"(585)가 있다. 이들은 라벨있는 합집합 기계에서의 인정 상태가 M_1, M_1 및 M_2 둘다, 그리고 M_2에 의해 각각 받아들여지는 입력을 표현한다는 것을 나타낸다. 도 5i에 더 상세히 나타낸 바와 같이, 예시적이고 비제한적인 구현예에서, 이들 라벨은 비트(최초의 기계마다 1 비트)로 볼 수 있다. 예를 들어, M_1은 하위 비트를 할당받을 수 있고, M_2는 상위 비트를 할당받을 수 있다. 그러면, 가능한 라벨은 "01"(즉 M_1이 입력을 받아들이고 M_2는 입력을 받아들이지 않음), "11"(즉, M_1 및 M_2 둘다 입력을 받아들임) 및 "10"(즉, M_2는 입력을 받아들이고 M_1은 입력을 받아들이지 않음)으로 표현될 수 있다. 다른 가능한 경우는 "00"이다. 합집합 기계의 비인정 상태(non- accept state) 모두가 이 라벨을 갖는 것으로 볼 수 있다(M_1 및 M_2 어느 것도 입력을 받아들이지 않기 때문임).

<108> 이와 관련하여, 도 5e의 블록도에 나타낸 바와 같이, 결합 M_1M_2를 표현하는 결과로 얻어진 인정 상태 노드(580)를 발생하는 것에 부가하여, 본 발명은 또한 인정 상태 M_1 및 인정 상태 M_2를 표현하는 인정 상태 노드(575, 585)를 발생한다. 도 5f에 도시된 바와 같이, 합집합 연산자 $M_1 \cup M_2$를 만날 때 이들 인정 상태(575, 580 및/또는 585) 중 임의의 것이 결과로 얻은 인정 상태의 일부일 수 있는 반면, 도 5g에 나타낸 바와 같이, 교집합 연산자 $M_1 \cap M_2$는 M_1M_2 노드(580)만을 포함한다. 예를 들어, M_1이 하위 비트를 할당받고 M_2가 상위 비트를 할당받는 경우, 비제한적인 일 실시예에서, "11"로 라벨이 붙여진 인정 상태를 찾아냄으로써 합집합 기계가 교집합 기계로 변환된다. 아무 것도 없는 경우, 교집합 기계는 비어 있다. "11"로 라벨이 붙여진 상태 집합 A가 주어진 경우, 합집합 기계로부터 A 내의 어떤 상태에도 도달할 수 없는 상태를 제거함으로써 교집합 기계가 발생된다.

<109> 도 5h의 블록도에 나타낸 바와 같이, 본 발명에 따른 변환 동안에 서브타입 지정(subtyping)도 캡처될 수 있다. 타입 지정과 관련하여 컴포넌트(570)로 결정화를 할 때 결정될 수 있는 3가지 결과가 있다. 위쪽에 나타낸 바와 같이, 어떤 인정 상태도 M_1으로 라벨이 붙여져 있지 않은 경우 $M_1 \subset M_2$이 한 결과이다. 중간에 나타낸 바와 같이, 인정 상태 M_1M_2(580)만이 발생되는 경우 $M_1 = M_2$가 또하나의 결과이다. 아래쪽에 나타낸 바와 같이, 어떤 인정 상태도 M_2로 라벨이 붙여져 있지 않은 경우, $M_2 \subset M_1$이 또하나의 결과이다. 상기 변환 프로세스가 주어진 경우, 서브타입 지정이 결정될 수 있다. 예를 들어, M_1이 하위 비트를 할당받고 M_2가 상위 비트를 할당받는 경우에, 모든 인정 상태가 11로 라벨이 붙여지면, M_1은 M_2와 같다. 10 및 11로 라벨이 붙여진 인정 상태가 있는 경우, M_2는 M_1을 포함한다(즉, M_1은 M_2의 서브타입임).

<110> 도 5i는 라벨있는 합집합 연산자에 따라 기계(590) 및 기계(592)를 변환하는 예시적이고 비제한적인 프로세스에 대한 흐름도를 나타낸 것으로서, 여기서 2개의 기계(590, 592) 각각은 2-비트 기계이다. 비트 라벨을 적용하는 상기한 예시적이고 비제한적인 구현예에 따르면, 기계(590)의 라벨 L_1 및 L_2에 2 비트가 할당되고, 기계(592)의 라벨 L_3 및 L_4에 2 비트가 할당된다. 본 발명의 라벨있는 합집합 연산자에 따라 결합될 때, 이것은 결과로 얻어진 4-비트 기계(595)에서 16개의 인정 상태를 생성한다. 예를 들어, L_2는 라벨 0010을 받게 되고, L_2L_4는 라벨 1010을 받으며, $L_1L_2L_3L_4$는 라벨 1111을 받고, 이들 조합 모두가 표현될 때까지 이하 마찬가지이다.

<111> 도 5j는 대응하는 순서없는 기계(들)을 구축 및 결합하기 위해 순서있는 기계(들)을 어떻게 변환하는지를 나타낸 것이다. 예를 들어, 패턴 집합 P는 적어도 하나의 p_1을 포함하는 집합, 0개 이상의 p_2를 포함하는 집합, 하나 이상의 p_3를 포함하는 집합을 나타내는 이하의 집합 $\{p_1, p_2^*, p_3^+\}$에 의해 표현되는 것과 같은 선택적인 존재 제약조건(occurrence constraint)을 포함할 수 있다. 본 발명의 예시적이고 비제한적인 구현예에 따르면, 집합 패턴 P는 2개의 구성 부분, (A) 지금 설명되는 예시적인 집합 P 내의 $p_1 \cup p_2 \cup p_3$ 등의 합집합 패턴(596), 및 (B) 각각이 어떤 입력을 받아들이는 라벨있는 실행 집합(598)으로 분할된다. 지금 설명되는 예에서, 이것의 결과 $\{L_1, L_2^*, L_3^+\}$이 얻어질 수 있으며, 여기서 L_1, L_2 및 L_3는 각각 p_1, p_2 및 p_3의 인정 상태를 나타낸다.

<112> 그에 따라, 도 5k에 나타낸 바와 같이, 본 발명의 예시적이고 비제한적인 구현예에 따라, 예를 들어, 순서없는 패턴 P_1 및 P_2가 주어진 경우, 순서없는 패턴에 대해 라벨있는 합집합 U_L을 수행하기 위해, 이하의 표현식 $\cup(P_1), \cup(P_2), \Gamma(P_1)$ 및 $\Gamma(P_2)$이 계산되며, 여기서 $\cup(x)$는 x에 대한 합집합 패턴이고, $\Gamma(x)$는 x에 대한 인정 실행(accepting run)의 집합이다. 이어서, 이들 표현식에 기초하여, 이하의 2개의 결과가 얻어지며,

$$(1)\ \cup(P_1 \cup_L P_2) = \cup(P_1) \cup_L \cup(P_2)$$

<113> $$(2)\ \Gamma(P_1 \cup_L P_2) = \Gamma(P_1) * \Gamma(P_2)$$

<114> 여기서, 도 5k 및 상기 식 (2)에서의 "*" 연산자 LUCP는 라벨있는 합집합 외적(labeled union cross product) 연산을 나타낸다.

<115> 그 다음에, 실행 라벨이 일관성이 있도록 $\cup(P_1 \cup_L P_2)$의 라벨이 변경된다. 예를 들어, 라벨 변경은 중복 라벨의 그룹 R 내의 모든 l을 고쳐쓰기함으로서 달성된다. R로부터, 임의의 멤버 z(예를 들어, 상기한 비트 라벨 구현에서 가장 낮은 번호의 라벨 또는 가장 낮은 순서의 비트)가 선택되고, R 내의 각각의 l에 대해 l이 z로 고쳐쓰기된다. 예를 들어, 도 5l에 나타낸 바와 같이, $\cup(P_1 \cup_L P_2)$은 "L_1,L_2", "L_1,L_3,L_4,L_2" 및 "L_2,L_4"를 포함하는 예시적인 인정 상태 라벨 ASL1을 갖는 경우, 간결한 인정 상태 라벨 ASL2, 즉 "L_1", "L_1,L_2" 및 "L_2"를 형성하기 위해 L_3는 L_1으로 대체될 수 있고, L_4는 L_2로 치환될 수 있다.

<116> 상기한 바와 같이, 도 5k 및 상기 식 (2)에서의 "*" 연산자 LUCP는 라벨있는 합집합 외적 연산을 나타낸다. 이전의 단락에 기술된 바와 같이 P_1 및 P_2의 인정 실행의 라벨을 변경하면, 라벨있는 합집합 외적 연산이 이하의 비제한적인 프로세스에 따라 정의될 수 있다. 먼저, 한 쌍의 실행(R_1, R_2)에 대해 연산자 U_L이 정의된다. 그렇게 하기 위해, 실행 R의 라벨로서 Label(R)이 정의된다. 이어서, R이 일련의 쌍 $(l, [x, y])$으로서 정의되며, 여기서 l은 라벨 변경된 $P_1 \cup_L P_2$으로부터의 어떤 라벨이고, x는 l의 최소 발생 횟수이며, y는 l의 최대 발생 횟수이고, $[x, y] \subseteq [0, \infty]$이다. 다음과 같이, R_1 * R_2(즉, R_1 및 R_2의 외적) 내의 각각의 요소쌍을 고려함으로써 $R_1 \cup_L R_2$가 계산된다. 각각의 쌍 $(l_1, [x_1, y_1]), (l_2, [x_2, y_2])$에 대해, $l_1 = l_2$이면, 구간 합집합(interval union) 연산을 사용하여 $(l_1, [x_1, y_1]), (l_2, [x_2, y_2])$는 $(l_1, [x_1, y_1]) \cup (l_2, [x_2, y_2])$로 된다. 그렇지 않은 경우, $(l_1, [x_1, y_1]), (l_2, [x_2, y_2])$은 공집합이다.

<117> 따라서, R_1 * R_2(즉, R_1과 R_2의 외적) 내의 멤버 z에 대해, U_L이 $(l_1, [x_1, y_1]) \cup (l_2, [x_2, y_2])$ 또는 공집합으로서 계산된다. 이어서, 결과 집합 S를 형성하기 위해 이들 결과에 대해 합집합을 구함으로써 $R_1 U_L R_2$가 계산된다. S가 R_1 및 R_2와 동일한 카디날리티(cardinality)를 갖는 경우, $R_1 U_L R_2$은 라벨(예를 들어, Label(R_1) 및 Label(R_2))을 갖는 S이다. 그렇지 않은 경우, $R_1 U_L R_2$은 $\{R_1, R_2\}$이며, 이 경우 R_1 및 R_2는 서로 소(disjoint)이고, 서로 다른 라벨을 사용하여 유지된다.

<118> 예를 들어,

$$R_1 = \{(L_1, [0, 1]), L_2, [1, \infty]), L_3, [1, 1])\}$$

<119> $$R_2 = \{(L_1, [1, 2]), L_2, [2, 2]), L_3, [1, 1])\}$$

<120> 인 경우,

<121> $R_1 U_L R_2 = \{(L_1, [0, 2]), L_2, [1, \infty]), L_3, [1, 1])\}$이고 라벨 Label($R_1$), Label($R_2$)을 갖는다.

<122> 다른 예로서,

$$R_1 = \{(L_1, [1, 1]), L_2, [2, \infty]), L_3, [1, 1])\}$$

<123> $$R_2 = \{(L_1, [0, \infty]), L_2, [1, 1]), L_3, [4, 4])\}$$

<124> 인 경우,

<125> $R_1 U_L R_2 = \{R_1, R_2\}$

<126> 인데, 그 이유는 R_1과 R_2가 서로 소이기 때문이다.

<127> 다른 예로서,

<128> $R_1 = \{L_1, L_2\}$(단, L_1이 $(L_1, [1, 1])$임)이고

<129> $R_2 = \{L_1, L_2, L_3^*\}$

<130> 인 경우,

<131> $R_1 U_L R_2 = \{L_1, L_2\}$(라벨 Label($R_1$), Label($R_2$)을 가짐)이고 $\{L_1, L_2, L_3^+\}$(라벨 Label(R_2)를 가짐)이다.

<132> 요약하면, 본 발명에 따른 실행 R_1 및 R_2에 대한 U_L의 정의인 $R_1 U_L R_2$가 상기한 절차에 따라 먼저 계산된다. 이어서, 이 결과가 R_1과 $(R_1 U_L R_2)$의 교집합 및 R_2와 $(R_1 U_L R_2)$의 교집합, 즉 $R_1 \cap (R_1 U_L R_2)$ 및 $R_2 \cap (R_1 U_L R_2)$을 구함으로써 정확해진다. $R_1 \cap (R_1 U_I R_2)$ 및 $R_2 \cap (R_1 U_L R_2)$은 합집합 연산자 U_L에 대해서와 유사한 방법을 사용하여 수행되지만, 구간 합집합 연산 대신에 구간 교집합(interval intersection)을 사용한다.

<133> 이것을 달성하기 위해, 본 발명의 예시적이고 비제한적인 실시예들에서, 합집합 실행은 여러 부분 $R_1, R_2,$및 $R_1 \cap R_2$으로 나누어진다. 이어서, 이들 부분 $R_1, R_2,$ 및 $R_1 \cap R_2$에는 각각 Label(R_1), Label(R_2), 및 Label(R_1), Label(R_2)로 라벨이 붙여진다.

<134> 그에 따라, P_1 및 P_2가 순서없는 패턴인 경우, $P_1 U_L P_2$의 결정화가 수행될 수 있다. 이와 관련하여, 본 발명은 $P_1 U_L P_2$가 합집합 패턴 부분 및 합집합 패턴의 인정 실행 집합을 사용하여 순서없는 패턴으로서 계산될 수 있게 해주며, 각각의 실행에는 P_1, P_2 또는 둘다로부터의 라벨을 사용하여 라벨이 붙여져 있다. 그러면, $P_1 U_L P_2$는 합집합 패턴 및 라벨있는 실행 집합을 역시 갖는 어떤 다른 순서없는 패턴(P_3)과의 합집합이 될 수 있고, 또다른 순서없는 패턴(P_4)에 대해서도 마찬가지이다.

<135> 예시적이고 비제한적인 사용 시나리오

<136> 보충 설명을 위해, 본 발명의 다양한 사용 시나리오는 컴퓨팅 시스템에서 본 발명의 Q 프레임워크를 사용한 패턴 정합이 적용될 수 있는 아주 다양한 응용들을 설명한다. 따라서, 변환되는 트리 내에 내포된 액션 의미 또는 순서있는 정보 및 순서없는 정보를 보존하면서 매칭시킬 수 있다는 것이 컴퓨팅 시스템에서 광범위한 일련의 패턴 정합 시스템을 가능하게 해준다. 그렇지만, 선택된 실제의 시나리오가 단지 예시적인 것에 불과하며, 따라서 본 발명이 적용되는 패턴 정합의 분야를 제한하는 것으로 해석되어서는 안된다. 실제로, 본 발명에 따라 MFST를 변환하는 기능은, 정의에 의하면, 아주 광범위한데, 그 이유는 액션 제약조건에 관한 정보, 또는 변환되는 트리 표현 내에 내포된 순서있는 정보 및 순서없는 정보의 손실에 대한 걱정을 할 필요가 없기 때문이다.

<137> 이와 관련하여, 상기한 바와 같이, Q 프레임워크는 적어도 교집합, 합집합 및 여집합 변환에 따라 MFST를 변환할 때 이러한 액션 또는 순서있는 정보 및 순서없는 정보의 보존을 가능하게 해준다. 이들 3가지 변환 연산을 사용하여, 당업자라면 잘 알고 있는 트리 데이터 구조에 대한 속성이 구조 호환성, 즉 서브 타입 지정으로서 수행될 수 있다. 종종, 컴퓨팅 애플리케이션은 주어진 트리가 다른 주어진 트리의 부분집합 또는 트리의 집합(또는 합집합, 여집합 또는 교집합에 의해 정의되는 트리들에 대한 어떤 변환 연산)인지를 알고자 한다.

<138> 본 발명이 적용될 수 있는 다른 중요한 부류의 시나리오는 정적 타입 검사를 수행하는 다수의 애플리케이션이며, 이 경우 컴파일러는 프로그램을 보고 절차적 호환성이 있는지 조회한다. 절차적 호환성 검사는 일련의 트리에 대한 패턴 정합을 수반하며, 컴퓨터 프로그램 내의 버그를 찾는 데 도움이 된다.

<139> 일반적으로, 사용자 또는 컴퓨팅 시스템이 트리 데이터의 집합 또는 트리 데이터의 부분집합에 대해 구현하고자 할 수 있는 테스트의 부류가 무제한이다. 그렇지만, 일반적으로 어떤 반복 테스트가 행해진다. 예를 들어, 종종 테스트는 제1 트리 또는 트리의 집합이 제2 트리 또는 트리의 집합과 상호작용하는지를 알고자 한다. 또는, 트리의 집합에 대한 합산이 있는지와 트리 순회 경로들 전부의 적용 범위가 있는지를 알고자 한다. 또는, 사용자는 어느 트리에 의해 기본 사례가 얻어지는지를 알고자 할 수 있다. 또한, 패턴 정합으로서 받아들여지는 트리가 있는지의 질문을 하는 비어있음 테스트(empty test)를 테스트하고자 할 수 있다. 상기한 바와 같이, 본 발명의 일 실시예에서, 본 발명은 이하의 4가지 변환 테스트, 즉 교집합, 합집합, 여집합 및 비어있음 테스트를 가능하게 해준다.

<140> 컴파일러에서의 본 발명의 유익한 사용에 대해 이상에서 설명하였다. 본 발명은 또한 스키마 유효성 검사 등의 다수의 다른 시나리오에서도 사용될 수 있다. 예를 들어, 구매 주문을 갖는 메시지가 들어오고, 질문은 구매 주문이 어떤 스키마에 대해 호환성이 있는지이다. 본 발명의 패턴 정합은 결과를 동일하게 보존하면서 액션 의미 또는 순서있는/순서없는 정보와 상관없이 스키마와 대조하여 메시지의 유효성을 검사하는 데 사용될 수 있다.

<141> 계약 검사는 본 발명의 또하나의 예시적인 사용이다. 예를 들어, 회사가 회사의 인적 자원 계산 시스템에 관한 정책을 가지고 있을 수 있으며, 이 정책에는 이 시스템이 물리적 시스템을 표현하는 일련의 컴퓨터 요구사항(예를 들어, X는 저장 용량을 나타내고, Y는 보안 수준을 나타내며, Z는 처리 능력을 나타냄, 기타 등등)을 준수해야만 하는 것으로 되어 있다. 이와 관련하여, 각각의 구성은 트리 표현으로 된 일련의 요구사항으로 표현될 수 있으며, 본 발명의 프레임워크의 트리 변환 기능에 따라, 시스템 분석가는 미리 정의된 변환에 따라 컴퓨터 구성을 변환할 수 있고 새로운 컴퓨터 구성이 HR 컴퓨팅 시스템에 요구되는 계약과 일치하는지를 알 수 있다.

<142> 본 발명은 또한 컴퓨팅 시스템에 의해 기록된 일련의 로그 데이터에서 보안 취약점 패턴을 찾아내는 것 또는 컴퓨팅 시스템에 대한 가능한 구성을 결정하는 것에 적용될 수 있다. 방화벽을 통과하는 메시지를 트리로 볼 수 있는 경우 본 발명이 방화벽에 적용될 수 있으며, 임의의 주어진 메시지가 방화벽을 통과해서는 안되는지 여부를 관찰하기 위해 패턴 정합이 이용될 수 있다. 본 명세서에서 언급되고 Q 프레임워크에 의해 지원되는 변환 부울 연산이 임의의 논리문을 형성하는 데 이용될 수 있기 때문에, 컴퓨팅 시스템에서의 임의의 규칙 시스템이 트리로 변환될 수 있고, 이들 규칙에 대한 준수 여부를 판정하기 위해 패턴 정합이 적용될 수 있다. 따라서, 본 발명의 응용이 무제한이라는 것이 명백하다.

<143> 유한 상태 오토마다 및 변환기

<144> 부가적으로, 유한 상태 기계(finite state machine, FSM) 또는 유한 상태 오토마톤(finite state automaton, FSA)는 상태, 천이 및 액션으로 이루어진 거동 모델이다. FSM의 상태는 과거에 관한 정보를 저장한다, 즉 상태는 시스템의 시작부터 현 시점까지의 입력 변화를 반영한다. 천이는 상태 변화를 말하며 천이를 가능하게 하기 위해 이행을 필요로 하는 조건으로 기술된다. 액션은 주어진 시점에 수행되어야 하는 활동에 대한 설명이다. 몇가지 액션 타입, 진입(Entry), 이탈(Exit), 입력(Input) 및 천이(Transition) 액션이 있다. 진입 액션은 상태에 들어갈 때 액션을 실행한다. 이탈 액션은 상태에서 빠져나올 때 액션을 실행한다. 입력 액션은 현재 상태 및 입력 조건에 따라 액션을 실행한다. 천이 액션은 어떤 천이를 수행할 때 액션을 실행한다.

<145> 유한 상태 변환기(FST)는 액션(들)을 사용하여 주어진 입력(들) 및/또는 상태(들)에 기초하여 출력(들)을 발생하는 FSM의 한 유형으로서, 제어 애플리케이션, 컴퓨터 프로그램의 작성, 기타 등등에 사용될 수 있다. 도 6a는 간단한 유한 상태 변환기(620)를 나타낸 것이다. 변환기(620)는, FSM(620)에 의해 번역 또는 변환되는 바와 같이, 주어진 입력(들)(610)에 기초하여 출력(들)(630)을 발생한다. 존재하는 2가지 유형의 변환기(FSM)로는 Moore 모델 및 Mealy 모델이 있으며, 이들에 대해서는 이하에서 더 상세히 설명한다. 혼합 모델도 종종 사용된다.

<146> 따라서, 많은 서로 다른 종류의 컴퓨터 시스템 및 프로세스가 FSM 및 FST로서 모델링될 수 있다는 것을 잘 알 것이다. 예를 들어, 임의의 XML(extensible markup language) 문서가 유향 그래프를 사용하여 FSM으로 표현될 수 있다. 관계형 데이터베이스 내의 관계 데이터도 역시 이와 같이 표현될 수 있으며, 예를 들어, 입력(예를 들어, 질의)이 기본적인 관계형 저장소를 표현하는 FST에 의해 출력(예를 들어, 질의 결과)으로 번역된다. 일

반적으로, 컴퓨터 프로세스를 모델링하는 데 사용될 때, FST는 통상적으로 엣지-라벨있는 유향 그래프(edge-labeled directed graph)로 표현되며, 각각의 정점은 n개의 상태 중 하나를 나타내고, 각각의 엣지는 엣지에 붙여져 있는 알파벳 심볼의 수신 시에 한 상태로부터 다른 상태로의 천이를 나타낸다.

<147> 전체 시스템에 대한 소프트웨어를 설계하기 위해 컴퓨터 시스템 내의 많은 복잡한 서브시스템 및 프로세스가 전체 시스템의 일부로서 통신 연결될 때, 설계자는 먼저 서브시스템 및 프로세스 각각을 FST로서, 예를 들어, 유향 그래프로서 또는 다른 등가의 표현으로서 표현할 수 있다. 이어서, 서로 다른 서브시스템을 연결시키는 단일의 시스템에 대한 복잡한 컴퓨터 프로그램을 생성하기 위해, FST(예를 들어, 유향 그래프)가 전체 시스템의 거동을 표현하는 새로운 유향 그래프(들)를 형성하기 위해 여러가지 연산에 따라 결합되거나 변환될 수 있다.

<148> 예를 들어, 사용자 Jane이 데이터베이스에 저장되어 인터넷 서버 상의 애플리케이션에 의해 서비스되는 친구 John의 방학 사진을 얻기 위해 클라이언트 컴퓨터로부터 인터넷 서버에 요청을 하는 것으로 가정한다. 잘 알 것인 바와 같이, 이러한 요청에 따라 행해지는 엔드-투-엔드 통신이, 요청 자체를 수행하는 것 이외에, Jane의 인증부터 시작하여 Jane이 John의 사진을 보도록 허가받게 해주는 것까지 수없이 많다. 간단한 예로서, 요청 자체를 작성 및 처리하는 것은 제1 FST로서 모델링될 수 있다. 서버에서, 제2 FST는 일련의 접근 제어 목록(ACL) 및 대응하는 정책에서 발견될 수 있는 것 등의 여러가지 규칙(일련의 XML 단편, 트리 또는 유향 그래프로 표현될 수 있음)을 참조하여 Jane이 John의 친구인지를 검사하는 거동을 모델링할 수 있다. 그에 부가하여, 제3 FST는 관계형 데이터베이스 자체를 모델링할 수 있다. 제1, 제2 및 제3 FST 세트의 변환, 결합, 매칭, 번역, 기타 등등에 의해, 시스템을 표현하고 특정의 입력 요청에 대해 예("허가됨") 또는 아니오("허가되지 않음") 대답을 반환하며 사진의 배달을 처리하는 새로운 유향 그래프가 형성될 수 있다.

<149> 일반적으로, FSM은 도 8a의 간단한 상태 천이도에서와 같이 상태도(또는 상태 천이도)를 사용하여 표현될 수 있다. 도 8a에서, 2개의 상태(S1, S2)가 각각 진입 액션(EntryA1 및 EntryA2)를 갖는 것으로 표현되어 있으며, 이는 상태(S1)에 들어갈 때 진입 액션(EntryA1)이 수행되고, 상태(S2)에 들어갈 때 진입 액션(EntryA2)이 수행된다는 것을 의미한다. 차례로, 천이 조건(TC1)이 일어날 때 상태(S1)에서 상태(S2)로의 천이(T1)가 행해지고, 천이 조건(TC2)이 일어날 때 상태(S2)에서 상태(S1)로의 천이(T2)가 행해진다. 상태(S1)는 "문이 열려 있음"일 수 있고, 상태(S2)는 "문이 닫혀 있음"일 수 있다. 따라서, 상태(S1)에서 상태(S2)로 가기 위해, 천이(T1)가 행해져야만 하고, 이는 천이 조건(TC1)("힘이 문을 닫힘 방향으로 움직이게 하고 있음"일 수 있음)이 일어날 때에만 행해진다. 상태(S2)에 들어갈 때 실행되는 진입 액션(EntryA2)은 따라서 "문을 닫음"일 수 있다. 문을 열 때, 즉 상태(S2)에서 상태(S1)로 갈 때, 유사한 천이 연쇄가 수행될 수 있다.

<150> 도면들 이외에, 다른 유형의 상태 천이 테이블도 FSA를 표현하는 데 사용될 수 있다. 이러한 상태 천이 테이블(STT1)의 통상적인 표현이 도 7a에 도시되어 있으며, 여기서 테이블(STT1)의 한 열(현재의 상태(B) 등)과 테이블(STT1)의 한 행(조건 Y 등)의 조합은 상태(B)에 있는 동안에 조건(Y)이 일어날 때 나타나는 그 다음 상태, 즉 상태(C)를 나타낸다. 그렇지만, 상태 천이 테이블(STT1) 등의 테이블을 사용하여, 완전한 액션 정보가 각주를 사용하는 것만으로 추가될 수 있다.

<151> 그렇지만, 상태 테이블을 사용하여 전체 액션 정보를 포함시키는 FSM 정의가 있다. 예를 들어, 가상 환경에서 정의된 FSM은 입력 제어 속성 및 출력 액션의 할당된 이름을 사용하여 제어 시스템의 거동을 기술하는 데 사용되는 소프트웨어 명세 방법과 관련된 가상 유한 상태 기계(virtual finite state machine, VFSM)라고 한다.

<152> 가상 환경은 VFSM이 동작하는 환경을 말하며, 3개의 이름 집합, 즉 입력 이름, 출력 이름 및 상태 이름에 의해 정의된다. 입력 이름은 모든 이용가능한 변수의 제어 속성에 의해 표현된다. 출력 이름은 변수에 대한 모든 이용가능한 액션에 의해 표현되고, 상태 이름은 FSM의 상태들 각각에 대해 정의된다. 입력 이름은 상태 천이 또는 입력 액션을 수행하기 위한 가상 조건을 작성하는 데 사용된다. 가상 조건은 정 논리 대수(positive logic algebra)를 사용하여 작성된다. 출력 이름은 액션(진입 액션, 이탈 액션, 입력 액션 또는 천이 액션)을 트리거하는 데 사용된다.

<153> 도 7b의 예시적인 상태 테이블(ST1)로 나타낸 바와 같이, 상태 테이블은 VFSM의 상태의 거동의 상세를 정의한다. 상태 천이 테이블(ST1)은 3개의 열을 포함하며, 첫번째 열에는 상태 이름(SN)이 사용되고, 두번째 열에는 정 논리 대수를 사용하여 입력 이름으로 작성된 가상 조건(CO)가 배치되며, 세번째 열에는 액션 AC를 트리거하는 데 사용되는 출력 이름이 나타난다.

<154> 여기에 제시된 반응형 시스템을 모델링하는 데 사용되는 것에 부가하여, FSA는 많은 다른 영역(언어학, 컴퓨터 과학, 철학, 생물학, 수학, 및 논리학을 포함함)에서 중요하다. 유한 상태 기계는 오토마타 이론 및 계산 이론

에서 연구되는 오토마타의 한 유형이다. 컴퓨터 과학에서, 유한 상태 기계는 애플리케이션 거동의 모델링, 하드웨어 디지털 시스템의 설계, 소프트웨어 엔지니어링, 컴파일러, 그리고 계산 및 언어의 연구에서 널리 사용되고 있다. FSA의 응용의 완전한 조사는 거의 불가능하며, 다른 시스템에의 FSA의 적용이 거의 무한하다고만 말해두자.

<155> 일반적으로, 변환기는 2개의 형식 언어(formal language) 간의 관계를 계산한다. FSM 및 FSA와 관련하여, 변환기는 액션을 사용하여 주어진 입력 및/또는 상태에 기초하여 출력을 발생하고, 제어 애플리케이션에 사용될 수 있다. 2가지 유형의 변환기 FSM(Moore 모델 및 Mealy 모델)이 일반적으로 구별된다. 실제로는, 혼합 모델이 종종 사용된다.

<156> 예시를 위해, 도 8b는 4개의 상태(S3, S4, S5 및 S6)[각각, 출력(03, 04, 05, 06)을 가짐]를 갖는 Moore 모델 예를 보여주는 변환기 FSM(800)을 나타낸 것이다. Moore 기계의 경우, FSM은 진입 액션만을 사용한다, 즉 출력이 상태에만 의존한다. Moore 모델의 이점은 거동의 간단화이다. 도 8b의 예는 엘리베이터 문의 Moore FSM(800)을 나타낸 것이다. 상태 기계는 상태 변화를 트리거하는 2개의 명령, "command open"(C1) 및 "command_close"(C2)를 인식한다. 예를 들어, "Opening" 상태(S6)에서의 진입 액션(EA1)은 문을 여는 모터를 기동시키고, "Closing" 상태(S4)에서의 진입 액션(EA2)는 문을 닫는 다른쪽 방향으로 모터를 기동시킨다. "Opened" 상태(S3) 및 "Closed" 상태(S5)는 이 예에서 어떤 액션도 수행하지 않으며, 오히려 이들 상태는 각각 "문이 열려 있음" 또는 "문이 닫혀 있음"을 외부 시스템(예를 들어, 다른 상태 기계)에 신호한다.

<157> 도 8c는 2개의 상태(S7 및 S8), 출력(07 및 08), 그리고 입력 액션(I1 및 I2)을 각각 갖는 Mealy 모델예를 보여주는 변환기 FSM(810)을 나타낸 것이다. Mealy 기계의 경우, FSM은 입력 액션만을 사용한다, 즉 출력이 입력 및 상태에 의존한다. Mealy FSM의 사용은 종종 상태의 수의 감소를 가져온다. 도 8c의 예는 도 8b의 Moore 예(800)에서와 동일한 거동을 구현하는 Mealy FSM(810)을 나타낸 것이다. 2가지 입력 액션, "command_close이 도착할 때 문을 닫기 위해 모터를 기동시킴"(I1) 및 "command_open이 도착할 때 문을 열기 위해 다른쪽 방향으로 모터를 기동시킴"(I2)이 있다.

<158> 유한 오토마타에 대한 또다른 구별은 결정론적 유한 오토마타(deterministic finite automata, DFA)와 비결정론적 유한 오토마타(non-deterministic finite automata, NDFA) 또는 일반화된 비결정론적 유한 오토마타(generalized non-deterministic finite automata, GNFA) 사이에서이다. 결정론적 오토마타에서는, 각각의 상태에 대해서, 각각의 가능한 입력에 대해 정확히 하나의 천이가 있다. 비결정론적 오토마타에서는, 주어진 가능한 입력에 대한 주어진 상태로부터의 천이가 없거나 2개 이상 있을 수 있다. 이 구별이 실제로는 의미가 있지만, 이론적으로는 의미가 없는데, 그 이유는 임의의 NDFA를 등가의 DFA로 변환할 수 있는 알고리즘이 존재하기 때문이다(그렇지만, 이 변환이 통상적으로 오토마톤의 복잡성을 상당히 증가시킨다).

<159> 단지 하나의 상태를 갖는 FSM은 조합 FSM(combinatorial FSM)이라고 하며 입력 액션만을 사용한다. 이 개념은 다수의 FSM이 함께 동작해야만 하는 경우와 설계 도구에 적합하게 하기 위해 순수 조합 부분을 FSM의 한 형태로서 생각하는 것이 편리한 경우에 유용하다.

<160> 일반적으로, 변환기는 2개의 형식 언어 간의 관계를 계산한다. 유한 상태 변환기(FST)에 의해 계산되는 관계의 부류는 합리적 관계(rational relation)의 부류라고 한다. FST는 통상적으로 자연어 처리 연구에 유용한다.

<161> FST는 2개의 테이프를 갖는 유한 상태 기계로서, 하나의 테이프를 갖는 보통의 유한 상태 오토마톤과 대비된다. 용어의 문제로서, 오토마톤은 그의 테이프의 내용을 입력으로 보는 경우 문자열을 인식한다고 말해진다. 환언하면, 오토마톤은 문자열을 집합 {0, 1}에 매핑하는 함수를 계산한다. 다른 대안으로서, 오토마톤이 문자열을 생성한다고 말해지며, 이는 그의 테이프가 출력 테이프로 간주된다는 것을 의미한다. 이를 고려하여, 오토마톤은 문자열의 집합인 형식 언어를 발생한다. 오토마타의 2가지 측면이 동등하며, 오토마톤이 계산하는 함수는 정확히 말하면 그가 인식한 문자열의 집합의 표시 함수(indicator function)이다. 유한 오토마타에 의해 발생된 언어의 부류는 정규 언어(regular language)의 부류라고 한다.

<162> 변환기의 2개의 테이프는 통상적으로 입력 테이프 및 출력 테이프로 간주된다. 이와 관련하여, 변환기는, 그의 입력 테이프 상의 문자열을 받아서 그의 출력 테이프 상에 다른 문자열을 발생함으로써, 그의 입력 테이프의 내용을 그의 출력 테이프로 변환(즉, 번역)한다고 말해진다. 변환기는 비결정론적으로 그렇게 할 수 있고, 각각의 입력 문자열에 대해 2개 이상의 출력을 생성할 수 있다. 변환기는 또한 주어진 입력 문자열에 대해 출력을 생성하지 않을 수도 있으며, 이 경우에 입력을 거부한다고 말해진다.

<163> 부가적으로, 형식상 유한 상태 변환기(T)는

<164> Q가 유한 집합, 즉 상태의 집합이고,

<165> Σ가 입력 알파벳이라고 하는 유한 집합이며,

<166> Γ가 출력 알파벳이라고 하는 유한 집합이고,

<167> I가 Q의 부분집합, 즉 초기 상태의 집합이고,

<168> F가 Q의 부분집합, 즉 최종 상태의 집합이며,

<169> $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times Q$ (단, ∈는 비어있는 문자열임)이 천이 관계(transition relation)가 되도록 하는 튜플$(Q, \Sigma, \Gamma, I, F, \delta)$이다.

<170> (Q, δ)는 *T*의 천이 그래프라고 하는 라벨있는 유향 그래프로 볼 수 있고, 정점들의 집합이 Q이며, $(q, a, b, r) \in \delta$는 정점(*q*)에서 정점(*r*)으로 가는 라벨있는 엣지가 있음을 의미한다. 이와 관련하여, a는 입력 라벨이고, *b*는 그 엣지의 출력 라벨이다.

<171> 확장된 천이 관계 δ^*를,

<172> $\delta \subseteq \delta^*$이고,

<173> $(q, \epsilon, \epsilon, q) \in \delta$ (모든 $q \in Q$에 대해)이며,

<174> $(q, x, y, r) \in \delta^*$이고 $(r, a, b, s) \in \delta$일 때, $(q, xa, yb, s) \in \delta^*$

<175> 인 최소 집합이라고 정의한다.

<176> 확장된 천이 관계는 기본적으로 엣지 라벨을 고려하기 위해 보강된 천이 그래프의 반사 이행적 폐쇄(reflexive transitive closure)이다. δ^*의 요소들은 경로라고 한다. 경로의 엣지 라벨(edge label)은 그의 성분 천이(constituent transition)의 엣지 라벨을 순서대로 연접시킴으로써 얻어진다.

<177> 변환기(*T*)의 *거동*은 다음과 같이 정의되는 합리적 관계[*T*]이다.

<178> $(i, x, y, f) \in \delta^*$인 $i \in I$ 및 $f \in F$이 존재하는 경우에만 $x[T]y$이다. 즉, 입력 라벨이 *x*이고 출력 라벨이 *y*인 초기 상태에서 최종 상태로의 경로가 존재하는 경우 T가 문자열 $x \in \Sigma^*$을 문자열 $y \in \Gamma^*$로 변환한다.

<179> 유한 오토마타에 대해 정의된 이하의 동작, 즉 합집합, 연접(concatenation), Kleene 폐쇄(Kleene closure), 합성(composition), 입력 테이프의 투영 및 출력 테이프의 투영이 유한 변환기에도 적용된다.

<180> 합집합 연산과 관련하여, 변환기(*T* 및 *S*)가 주어진 경우, $x[T]y$ 또는 $x[S]y$일 때에만 $x[T \cup S]y$인 변환기 $T \cup S$가 존재한다.

<181> 연접 연산과 관련하여, 변환기(*T* 및 *S*)가 주어진 경우, $w[T]y$이고 $x[S]z$일 때에만 $wx[T \cdot S]yz$인

변환기 $T \cdot S$가 존재한다.

<182> Kleene 폐쇄 연산과 관련하여, 변환기(*T*)가 주어진 경우, 이하의 속성을 갖는 변환기(*T*)가 존재한다.

<183> (1) $\varepsilon[T^*]\varepsilon$,

<184> (2) $w[T^*]y$이고 $x[T]z$인 경우 $wx[T^*]yz$이고,

<185> (1) 또는 (2)에 의해 요구되지 않는 한, $x[T^*]y$가 성립하지않는다.

<186> 유의할 점은 변환기의 교집합의 개념이 없다는 것이다. 그 대신에, 변환기에 특유하고 오토마타의 교집합의 구성과 유사한 구성을 갖는 합성의 연산이 있다. 합성은 다음과 같이 정의된다.

<187> 알파벳 Σ 및 Γ에 대한 변환기(*T*)와 알파벳 Γ 및 Δ에 대한 변환기(*S*)가 주어진 경우, $x[T]y$이고 $y[S]z$인 문자열 $y \in \Gamma^*$이 존재하기만 하면 $x[T \circ S]z$인 Σ 및 Δ에 대한 변환기 $T \circ S$가 존재한다.

<188> 또한, 오토마톤을 획득하기 위해 변환기의 어느 한 테이프를 투영할 수 있다. 2개의 투영 함수가 있으며, π_1은 입력 테이프를 보존하고, π_2는 출력 테이프를 보존한다. 제1 투영 π_1은 다음과 같이 정의된다.

<189> 변환기(*T*)가 주어진 경우, $x[T]y$인 문자열 y가 존재할 때에만 $\pi_1 T$가 *x*를 받아들이는 유한 오토마톤 $\pi_1 T$이 존재한다. 제2 투영 π_2도 이와 유사하게 정의된다.

<190> 그에 부가하여, 집합의 요소의 순서화(ordering), 시퀀싱(sequencing) 또는 배열의 직관적 개념을 공식화하는 부분 순서(partial order)를 표현하기 위해 유한 상태 기계가 사용될 수 있다. 부분 순서가 반드시 집합 내의 모든 객체의 상호 호환성을 보장해주는 전체 순서(total order)일 필요는 없다. 이와 관련하여, 전체 순서는 집합의 모든 항목 쌍들에 대해 정의된 일종의 부분 순서이다.

<191> 따라서, 부분 순서는 일부 항목쌍들에 대해서 정의되고, 반드시 모든 항목쌍들에 대해 정의될 필요는 없다. 예를 들어, 집합 {a, b} 및 {a, c, d}는 {a, b, c, d}의 부분집합이지만, 어느 것도 상대방의 부분집합이 아니다. 따라서, "~의 부분집합"은 집합에 대한 부분 순서이다. 다른 예로서, ≤(작거나 같음)은 정수에 대한 전체 순서인데, 그 이유는, 임의의 2개의 정수에 대해, 그 정수들 중 하나가 다른 정수보다 항상 작거나 같기 때문이다.

<192> 도 6b에 도시한 바와 같이, 순서있는 내포(ordered nest)는 "a"가 "b"를 "암시"한다는 정보를 표현하지만 순서화된 계층에 따라 그 반대는 성립하지 않는다(즉, "b"가 반드시 "a"를 "암시"하는 것은 아니다). 따라서, 도 6b의 트리는 순서있는 내포의 표현의 일례이다. 도 6c의 트리는, 이와 반대로, 순서없는 내포의 일례를 나타낸 것이다. 이와 관련하여, 도 6c의 트리에 의해 표현되는 실제 정보가 좌에서 우로 읽히는지 우에서 좌로 읽히는지에 상관없이, 동일한 실제 정보가 수집된다, 즉 "블록이 적색임" 또는 "적색이 블록임" 둘다가 논리적으로 블록이 적색(블록 = 적색)이라는 것을 나타낸다. 그에 따라, 이러한 정보는 순서없는 정보이며, 트리가 순회되는 순서가 순서없는 정보를 획득하는 데 중요하지 않다.

<193> 예시적인 네트워크화되고 분산된 환경

<194> 당업자라면 본 발명이 임의의 종류의 데이터 저장소에 연결되어 있는 임의의 컴퓨터 또는 다른 클라이언트나 서버 장치(컴퓨터 네트워크의 일부로서 또는 분산 컴퓨팅 환경에 설치될 수 있음)에서 구현될 수 있다는 것을 잘 알 것이다. 이와 관련하여, 본 발명은, 본 발명의 QFX의 실시예들과 관련하여 사용될 수 있는, 임의의 수의 메모리 또는 저장 유닛 및 임의의 수의 저장 유닛 또는 볼륨에 걸쳐 행해지는 임의의 수의 애플리케이션 및 프로세스를 갖는 임의의 컴퓨터 시스템 또는 환경에 관한 것이다. 본 발명은 네트워크 환경 또는 분산 컴퓨팅 환경에 설치되어 있는 서버 컴퓨터 및 클라이언트 컴퓨터(원격 또는 로컬 저장 장치를 가짐)를 갖는 환경에 적용될 수 있다. 본 발명은 또한 원격 또는 로컬 서비스 및 프로세스와 관련하여 정보를 발생, 수신 및 전송하기 위해 프로그래밍 언어 기능, 해석 및 실행 능력을 갖는 독립형 컴퓨터 장치에도 적용될 수 있다. 앞서 설명한 바와

같이, MFST는 다수의 기계 및 컴퓨팅 장치에 걸쳐 소프트웨어 프로세스들에 보편적으로 적용가능하며, 따라서 본 발명에 따라 문법을 MFST로 변환하는 기법들이 다양한 컴퓨팅 환경에서 아주 효과좋게 적용될 수 있다.

<195> 분산 컴퓨팅은 컴퓨팅 장치들 및 시스템들 간의 교환에 의해 컴퓨터 자원 및 서비스의 공유를 제공한다. 이들 자원 및 서비스는 정보의 교환, 파일 등의 객체에 대한 캐쉬 저장 장치 및 디스크 저장 장치를 포함한다. 분산 컴퓨팅은 네트워크 연결을 이용하여, 클라이언트들이 기업 전체에 이익이 되도록 그 클라이언트들의 총체적 능력을 이용할 수 있게 해준다. 이와 관련하여, 다양한 장치들이 본 발명의 QFX를 수반할 수 있는 애플리케이션, 객체 또는 자원을 가질 수 있다.

<196> 도 9는 예시적인 네트워크화된, 즉 분산 컴퓨팅 환경의 개략도이다. 분산 컴퓨팅 환경은 컴퓨팅 객체(910a, 910b, 기타) 및 컴퓨팅 객체 또는 장치(920a, 920b, 920c, 920d, 920e, 기타)를 포함한다. 이들 객체는 프로그램, 메서드, 데이터 저장소, 프로그램가능 논리, 기타 등등을 포함할 수 있다. 이 객체들은 동일한 또는 서로 다른 장치(PDA, 오디오/비디오 장치, MP3 플레이어, 퍼스널 컴퓨터, 기타 등등)의 일부분을 포함할 수 있다. 각각의 객체는 통신 네트워크(940)를 통해 다른 객체와 통신을 할 수 있다. 이 네트워크 자체는 도 9의 시스템에 서비스를 제공하는 다른 컴퓨팅 객체 및 컴퓨팅 장치를 포함할 수 있으며, 그 자체가 다수의 상호 접속된 네트워크를 나타낼 수 있다. 본 발명의 한 측면에 따르면, 각각의 객체(910a, 910b, 기타, 또는 920a, 920b, 920c, 920d, 920e, 기타)는, 본 발명에 따른 QFX의 여러가지 실시예들에서 사용하기에 적당한, API 또는 다른 객체, 소프트웨어, 펌웨어 및/또는 하드웨어를 이용할 수 있는 애플리케이션을 포함할 수 있다.

<197> 또한, 객체(920c 등)가 다른 컴퓨팅 장치(910a, 910b, 기타, 또는 920a, 920b, 920c, 920d, 920e, 기타)에서 호스팅될 수 있다는 것을 잘 알 것이다. 따라서, 도시된 물리적 환경이 컴퓨터 등의 접속된 장치들을 나타낼 수 있지만, 이러한 설명은 단지 예시적인 것이며, 이 물리적 환경이 다른 대안으로서 PDA, 텔레비전, MP3 플레이어, 기타 등등의 여러가지 디지털 장치들을 포함하는 것으로 도시되거나 설명될 수 있으며, 이들 장치 중 어느 것이라도 각종의 유선 및 무선 서비스, 인터페이스 등의 소프트웨어 객체, COM 객체, 기타 등등을 이용할 수 있다.

<198> 분산 컴퓨팅 환경을 지원하는 각종의 시스템, 컴포넌트, 및 네트워크 구성이 있다. 예를 들어, 컴퓨팅 시스템은 유선 또는 무선 시스템에 의해, 또는 로컬 네트워크나 광역 분산 네트워크에 의해 서로 접속될 수 있다. 현재, 많은 네트워크들이 인터넷에 연결되어 있으며, 이 인터넷은 광역 분산 컴퓨팅을 위한 인프라를 제공하고 많은 서로 다른 네트워크를 포함한다. 임의의 인프라가 본 발명의 QFX의 실시예들에 따라 행해지는 예시적인 통신을 위해 사용될 수 있다.

<199> 홈 네트워킹 환경에서, 각각이 고유의 프로토콜을 지원할 수 있는 적어도 4개의 서로 다른 네트워크 전송 매체[전력선, 데이터(무선 및 유선 둘다), 음성(예를 들어, 전화) 및 오락 매체 등]가 있다. 대부분의 홈 제어 장치(광 스위치 및 가전 기기 등)는 접속을 위해 전력선을 사용할 수 있다. 데이터 서비스는 광대역(예를 들어, DSL, 케이블 모뎀, 기타)으로서 가정에 들어갈 수 있으며, 가정 내에서 무선(예를 들어, HomeRF 또는 802.11B) 또는 유선(예를 들어, Home PNA, Cat 5, 이더넷, 심지어 전력선) 접속을 사용하여 액세스가능하다. 음성 트래픽은 유선(예를 들어, Cat 3) 또는 무선(예를 들어, 셀 전화)으로서 가정에 들어갈 수 있으며, 가정 내에서 Cat 3 배선을 사용하여 분산될 수 있다. 오락 매체 또는 기타 그래픽 데이터는 위성 또는 케이블을 통해 가정에 들어갈 수 있으며, 일반적으로 가정 내에서 동축 케이블을 사용하여 분산된다. IEEE 1394 및 DVI도 역시 미디어 장치의 클러스터들에 대한 디지털 상호접속이다. 이들 네트워크 환경 및 프로토콜 표준으로서 새로 등장할 수 있거나 이미 등장한 다른 네트워크 환경 모두가 인터넷 등의 원거리 통신망에 의해 외부 세계에 접속될 수 있는 인트라넷 등의 네트워크를 형성하기 위해 상호접속될 수 있다. 요약하면, 데이터의 저장 및 전송을 위해 각종의 서로 다른 소스가 존재하며, 그 결과, 본 발명의 컴퓨팅 장치들 중 어느 것이라도 임의의 기존의 방식으로 데이터를 공유하고 통신할 수 있으며, 본 명세서의 실시예들에 기술된 어떤 방식도 제한하기 위한 것이 아니다.

<200> 인터넷은 통상 컴퓨터 네트워킹 분야에서 잘 알려져 있는 TCP/IP(Transmission Control Protocol/Internet Protocol) 프로토콜 모음을 이용하는 네트워크 및 게이트웨이의 집합체를 말한다. 인터넷은 사용자들이 네트워크(들)를 통해 상호작용하고 정보를 공유할 수 있게 해주는 네트워킹 프로토콜을 실행하는 컴퓨터들에 의해 상호접속되어 있는 지리적으로 분산된 원격 컴퓨터 네트워크들로 된 시스템으로서 기술될 수 있다. 이러한 널리 보급된 정보 공유로 인해, 인터넷 등의 원격 네트워크는 지금까지 일반적으로 개방형 시스템으로 진화되어 왔으며, 개발자들은 이 개방형 시스템을 사용하여, 기본적으로 제한없이 특수한 동작 또는 서비스를 수행하는 소프트웨어 애플리케이션을 개발할 수 있다.

<201> 따라서, 네트워크 인프라는 클라이언트/서버, 피어-투-피어, 또는 하이브리드 아키텍처 등의 수많은 네트워크

토폴로지를 가능하게 해준다. "클라이언트"는 그와 관련되어 있지 않은 다른 부류 또는 그룹의 서비스를 사용하는 한 부류 또는 그룹의 멤버이다. 따라서, 컴퓨팅에서, 클라이언트는 다른 프로그램에 의해 제공되는 서비스를 요청하는 프로세스(즉, 대략적으로 말하면, 일련의 명령어 또는 작업)이다. 클라이언트 프로세스는 상대편 프로그램 또는 서비스 자체에 관한 동작 상세를 "알" 필요없이 요청된 서비스를 이용한다. 클라이언트/서버 아키텍처에서, 특히, 네트워크화된 시스템에서, 클라이언트는 보통 다른 컴퓨터(예를 들어, 서버)에 의해 제공되는 공유 네트워크 자원에 액세스하는 컴퓨터이다. 예로서, 도 9의 예시에서, 컴퓨터(920a, 920b, 920c, 920d, 920e, 기타)는 클라이언트로서 생각될 수 있고, 컴퓨터(910a, 910b, 기타)는 서버로서 생각될 수 있으며, 여기서 서버(910a, 910b, 기타)는 나중에 클라이언트 컴퓨터(920a, 920b, 920c, 920d, 920e, 기타)로 복제되는 데이터를 유지하고 있지만, 어느 컴퓨터라도 환경에 따라 클라이언트, 서버 또는 둘다로서 간주될 수 있다. 이들 컴퓨팅 장치 중 어느 것이라도 본 발명의 QFX의 다양한 실시예의 트리 문법 및 변환 기법을 포함할 수 있는 데이터를 처리하거나 서비스 또는 작업을 요청할 수 있다.

<202> 서버는 통상적으로 원격 또는 로컬 네트워크(인터넷 또는 무선 네트워크 인프라 등)를 통해 액세스가능한 원격 컴퓨터 시스템이다. 클라이언트 프로세스는 제1 컴퓨터 시스템에서 활성일 수 있고, 서버 프로세스는 제2 컴퓨터 시스템에서 활성일 수 있으며, 이들은 통신 매체를 통해 서로 통신함으로써 분산 기능을 제공하고 그에 따라 다수의 클라이언트가 서버의 정보-수집 능력을 이용할 수 있게 된다. 본 발명의 QFX에 따라 이용되는 임의의 소프트웨어 객체가 다수의 컴퓨팅 장치 또는 객체에 걸쳐 분산되어 있을 수 있다.

<203> 클라이언트(들) 및 서버(들)는 프로토콜 계층(들)에 의해 제공되는 기능을 사용하여 서로 통신을 한다. 예를 들어, HTTP(HyperText Transfer Protocol)는 월드 와이드 웹(WWW)(즉, "웹")과 관련하여 사용되는 통상의 프로토콜이다. 일반적으로, IP(Internet Protocol) 주소 등의 컴퓨터 네트워크 주소 또는 URL(Universal Resource Locator) 등의 다른 참조가 서버 또는 클라이언트 컴퓨터를 서로에게 식별시켜 주는 데 사용될 수 있다. 네트워크 주소를 URL 주소라고 할 수 있다. 통신 매체를 통해 통신이 제공될 수 있다, 예를 들어, 클라이언트(들) 및 서버(들)는 대용량 통신을 위해 TCP/IP 접속(들)을 통해 서로 연결되어 있을 수 있다.

<204> 따라서, 도 9는 본 발명이 이용될 수 있는 예시적인 네트워크화된 또는 분산 환경을 나타낸 것으로서, 서버(들)는 네트워크/버스를 통해 클라이언트 컴퓨터(들)와 통신을 한다. 보다 상세하게는, 본 발명에 따르면 다수의 서버(910a, 910b, 기타)는 통신 네트워크/버스(940)(LAN, WAN, 인트라넷, GSM 네트워크, 인터넷, 기타 등등일 수 있음)를 통해 다수의 클라이언트 또는 원격 컴퓨팅 장치(920a, 920b, 920c, 920d, 920e, 기타)[휴대용 컴퓨터, 핸드헬드 컴퓨터, 씬 클라이언트, 네트워크화된 가전 기기, 또는 기타 장치(VCR, TV, 오븐, 전구, 히터, 기타 등등)]와 상호접속되어 있다. 따라서, 본 발명이 본 발명에 의해 정의된 Q 프레임워크에 따라 속성 트리 문법을 컴파일하여 실행하고 MFST로 변환하는 것이 바람직한 임의의 컴퓨팅 장치에 적용될 수 있는 것으로 생각될 수 있다.

<205> 예를 들어, 통신 네트워크/버스(940)가 인터넷인 네트워크 환경에서, 서버(910a, 910b, 기타)는 다수의 공지된 프로토콜(HTTP 등) 중 임의의 것을 통해 클라이언트(920a, 920b, 920c, 920d, 920e, 기타)와 통신을 하는 웹 서버일 수 있다. 서버(910a, 910b, 기타)는 또한 클라이언트(920a, 920b, 920c, 920d, 920e, 기타)로서 역할할 수도 있으며, 이는 분산 컴퓨팅 환경의 특성일 수 있다.

<206> 상기한 바와 같이, 통신은, 적절한 경우, 유선 또는 무선이거나 이들의 조합일 수 있다. 클라이언트 장치(920a, 920b, 920c, 920d, 920e, 기타)는 통신 네트워크/버스(940)를 통해 통신을 하거나 통신을 하지 않을 수 있고, 그와 연관된 독립적인 통신을 가질 수 있다. 예를 들어, TV 또는 VCR의 경우에, 그의 제어에 대한 네트워크화된 측면이 있거나 있지 않을 수 있다. 각각의 클라이언트 컴퓨터(920a, 920b, 920c, 920d, 920e, 기타) 및 서버 컴퓨터(910a, 910b, 기타)는 다양한 애플리케이션 프로그램 모듈 또는 객체(135a, 135b, 135c, 기타)를 구비하고 있을 수 있고 다양한 유형의 저장 요소 또는 객체(이들을 통해 파일 또는 데이터 스트림이 저장될 수 있거나, 이들로 파일 또는 데이터 스트림의 일부가 다운로드, 전송 또는 이동될 수 있음)에의 접속 또는 액세스를 구비하고 있을 수 있다. 컴퓨터(910a, 910b, 920a, 920b, 920c, 920d, 920e, 기타) 중 임의의 하나 이상이 데이터베이스(930) 또는 다른 저장 요소[본 발명에 따라 처리 또는 저장되는 데이터를 저장하는 데이터베이스 또는 메모리(930) 등]의 유지 및 갱신을 책임지고 있을 수 있다. 따라서, 본 발명은 컴퓨터 네트워크/버스(940)에 액세스하여 그와 상호작용할 수 있는 클라이언트 컴퓨터(920a, 920b, 920c, 920d, 920e, 기타)와 클라이언트 컴퓨터(920a, 920b, 920c, 920d, 920e, 기타), 기타 유사한 장치 및 데이터베이스(930)와 상호작용할 수 있는 서버 컴퓨터(910a, 910b, 기타)를 갖는 컴퓨터 네트워크 환경에서 이용될 수 있다.

<207> 예시적인 컴퓨팅 장치

<208> 전술한 바와 같이, 본 발명은 본 발명에 따라 정의된 QFX의 기법을 적용하는 것이 바람직할 수 있는 임의의 장치에 적용된다. 따라서, 본 발명과 관련하여 사용하기 위해, 즉 장치가 상태 기계를 나타내는 소프트웨어 프로세스를 구현하거나 데이터를 수신, 처리 또는 저장할 수 있는 경우라면 어디에서라도 사용하기 위해 모든 종류의 핸드헬드, 휴대용, 및 기타 컴퓨팅 장치 및 컴퓨팅 객체가 생각되고 있다는 것을 잘 알 것이다. 따라서, 도 10에서 기술되는 이하의 범용 원격 컴퓨터는 일례에 불과하며, 본 발명은 네트워크/버스 상호운용성 및 상호작용을 갖는 임의의 클라이언트에서 구현될 수 있다. 따라서, 본 발명은 아주 적은 또는 최소한의 클라이언트 자원이 관여되어 있는 네트워크화된 호스트 서비스의 환경에서, 예를 들어, 클라이언트 장치가 네트워크/버스에의 인터페이스(가전 기기에 설치된 객체 등)로서만 역할하는 네트워크화된 환경에서 구현될 수 있다.

<209> 꼭 그럴 필요는 없지만, 본 발명은 장치 또는 객체에 대한 서비스의 개발자가 사용하기 위해 부분적으로 운영 체제를 통해 구현될 수 있고, 및/또는 본 발명의 컴포넌트(들)와 관련하여 동작하는 애플리케이션 소프트웨어 내에 포함될 수 있다. 소프트웨어는 일반적으로 하나 이상의 컴퓨터(클라이언트 워크스테이션, 서버 또는 기타 장치 등)에 의해 실행되는 프로그램 모듈 등의 컴퓨터-실행가능 명령어와 관련하여 기술될 수 있다. 당업자라면 본 발명이 다른 컴퓨터 시스템 구성 및 프로토콜에서 실시될 수 있다는 것을 잘 알 것이다.

<210> 도 10은 따라서 본 발명이 구현될 수 있는 적당한 컴퓨팅 시스템 환경(1000a)의 일례를 나타낸 것이지만, 앞서 설명한 바와 같이, 컴퓨팅 시스템 환경(1000a)은 미디어 장치에 적당한 컴퓨팅 환경의 일례에 불과하고 본 발명의 용도 또는 기능의 범위에 관한 어떤 제한을 암시하기 위한 것이 아니다. 컴퓨팅 환경(1000a)이 예시적인 운영 환경(1000a)에 예시된 컴포넌트들 중 임의의 것 또는 이들의 조합과 관련하여 어떤 종속성 또는 요건을 갖는 것으로 해석되어서도 안된다.

<211> 도 10을 참조하면, 본 발명을 구현하는 예시적인 원격 장치는 컴퓨터(1010a) 형태의 범용 컴퓨팅 장치를 포함한다. 컴퓨터(1010a)의 컴포넌트들은 처리 장치(1020a), 시스템 메모리(1030a), 및 시스템 메모리를 비롯한 다양한 시스템 컴포넌트들을 처리 장치(1020a)에 연결시키는 시스템 버스(1021a)를 포함할 수 있지만, 이에 한정되지 않는다. 시스템 버스(1021a)는 메모리 버스 또는 메모리 컨트롤러, 주변 장치 버스, 및 로컬 버스(각종 버스 아키텍처 중 임의의 것을 사용함)를 비롯한 몇가지 유형의 버스 구조 중 어느 것이라도 될 수 있다.

<212> 컴퓨터(1010a)는 통상적으로 각종의 컴퓨터 판독가능 매체를 포함한다. 컴퓨터 판독가능 매체는 컴퓨터(1010a)에 의해 액세스될 수 있는 이용가능한 매체라면 어느 것이라도 될 수 있다. 제한이 아닌 예로서, 컴퓨터 판독가능 매체는 컴퓨터 저장 매체 및 통신 매체를 포함할 수 있다. 컴퓨터 저장 매체는 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터 등의 정보를 저장하는 임의의 방법 또는 기술로 구현되는 휘발성 및 비휘발성, 이동식 및 비이동식 매체 둘다를 포함한다. 컴퓨터 저장 매체로는 RAM, ROM, EEPROM, 플래쉬 메모리 또는 기타 메모리 기술, CDROM, DVD(digital versatile disk) 또는 기타 광 디스크 저장 장치, 자기 카세트, 자기 테이프, 자기 디스크 저장 장치 또는 기타 자기 저장 장치, 또는 원하는 정보를 저장하는 데 사용될 수 있고 컴퓨터(1010a)에 의해 액세스될 수 있는 임의의 다른 매체가 있지만, 이에 한정되지 않는다. 통신 매체는 일반적으로 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터를 피변조 데이터 신호(반송파 또는 기타 전송 메카니즘 등)에 구현하며, 모든 정보 전달 매체를 포함한다.

<213> 시스템 메모리(1030a)는 ROM 및/또는 RAM 등의 휘발성 및/또는 비휘발성 메모리 형태의 컴퓨터 저장 매체를 포함할 수 있다. 시동 중과 같은 때에, 컴퓨터(1010a) 내의 구성요소들 간의 정보의 전송을 돕는 기본 루틴을 포함하는 기본 입/출력 시스템(BIOS)은 메모리(1030a)에 저장될 수 있다. 메모리(1030a)는 통상적으로 처리 장치(1020a)에 의해 즉각 액세스될 수 있고 및/또는 현재 처리되고 있는 데이터 및/또는 프로그램 모듈도 포함한다. 제한이 아닌 예로서, 메모리(1030a)는 또한 운영 체제, 애플리케이션 프로그램, 기타 프로그램 모듈, 및 프로그램 데이터를 포함할 수 있다.

<214> 컴퓨터(1010a)는 또한 다른 이동식/비이동식, 휘발성/비휘발성 컴퓨터 저장 매체를 포함할 수 있다. 예를 들어, 컴퓨터(1010a)는 비이동식 · 비휘발성 자기 매체로부터 판독을 하거나 그에 기록을 하는 하드 디스크 드라이브, 이동식 · 비휘발성 자기 디스크로부터 판독을 하거나 그에 기록을 하는 자기 디스크 드라이브, 및/또는 이동식 · 비휘발성 광 디스크(CD-ROM 또는 기타 광 매체 등)로부터 판독을 하거나 그에 기록을 하는 광 디스크 드라이브를 포함할 수 있다. 예시적인 운영 환경에서 사용될 수 있는 기타 이동식/비이동식, 휘발성/비휘발성 컴퓨터 저장 매체로는 자기 테이프 카세트, 플래쉬 메모리 카드, DVD, 디지털 비디오 테이프, 고상 RAM, 고상 ROM, 기타 등등이 있지만, 이에 한정되지 않는다. 하드 디스크 드라이브는 통상적으로 인터페이스 등의 비이동식 메모리 인터페이스를 통해 시스템 버스(1021a)에 접속되어 있고, 자기 디스크 드라이브 또는 광 디스크 드라이브는 통상적으로 인터페이스 등의 이동식 메모리 인터페이스에 의해 시스템 버스(1021a)에 접속되어 있다.

<215> 사용자는 키보드 및 포인팅 장치(통상 마우스, 트랙볼 또는 터치 패드라고 함) 등의 입력 장치를 통해 명령 및 정보를 컴퓨터(1010a)에 입력할 수 있다. 기타 입력 장치로는 마이크, 조이스틱, 게임 패드, 위성 안테나, 스캐너, 기타 등등이 있을 수 있다. 이들 및 기타 입력 장치는 종종 시스템 버스(1021a)에 연결되어 있는 사용자 입력(1040a) 및 관련 인터페이스(들)를 통해 처리 장치(1020a)에 접속되어 있지만, 병렬 포트, 게임 포트 또는 USB(universal serial bus) 등의 다른 인터페이스 및 버스 구조에 의해 접속될 수 있다. 그래픽 서브시스템도 역시 시스템 버스(1021a)에 접속되어 있을 수 있다. 모니터 또는 기타 유형의 디스플레이 장치도 출력 인터페이스(1050a) 등의 인터페이스를 통해 시스템 버스(1021a)에 접속되어 있으며, 이 인터페이스는 차례로 비디오 메모리와 통신할 수 있다. 모니터에 부가하여, 컴퓨터는 또한 출력 인터페이스(1050a)를 통해 접속되어 있을 수 있는 다른 주변 출력 장치(스피커 및 프린터 등)를 포함할 수 있다.

<216> 컴퓨터(1010a)는 원격 컴퓨터(1070a) 등의 하나 이상의 다른 원격 컴퓨터(장치(1010a)와 다른 미디어 기능을 가지고 있을 수 있음)로의 논리적 접속을 사용하여 네트워크화된, 즉 분산 환경에서 동작할 수 있다. 원격 컴퓨터(1070a)는 퍼스널 컴퓨터, 서버, 라우터, 네트워크 PC, 피어 장치 또는 기타 통상의 네트워크 노드, 또는 임의의 다른 원격 미디어 소비 또는 전송 장치일 수 있으며, 컴퓨터(1010a)와 관련하여 상기한 구성요소들 중 임의의 것 또는 그 전부를 포함할 수 있다. 도 10에 도시된 논리적 접속이 근거리 통신망(LAN) 또는 원거리 통신망(WAN) 등의 네트워크(1071a)를 포함하지만, 기타 네트워크/버스도 포함할 수 있다. 이러한 네트워킹 환경은 가정, 사무실, 전사적 컴퓨터 네트워크, 인트라넷 및 인터넷에서 흔히 볼 수 있는 것이다.

<217> LAN 네트워킹 환경에서 사용될 때, 컴퓨터(1010a)는 네트워크 인터페이스 또는 어댑터를 통해 LAN(1071a)에 접속된다. WAN 네트워킹 환경에서 사용될 때, 컴퓨터(1010a)는 통상적으로 인터넷 등의 WAN을 통해 통신을 설정하는 모뎀 또는 기타 수단 등의 통신 컴포넌트를 포함한다. 내장형 또는 외장형일 수 있는 모뎀 등의 통신 컴포넌트는 입력(1040a)의 사용자 입력 인터페이스 또는 기타 적절한 메카니즘을 통해 시스템 버스(1021a)에 접속될 수 있다. 네트워크화된 환경에서, 컴퓨터(1010a)와 관련하여 도시된 프로그램 모듈 또는 그의 일부분은 원격 메모리 저장 장치에 저장될 수 있다. 도시되고 설명된 네트워크 접속이 예시적인 것이며 컴퓨터들 간에 통신 링크를 설정하는 기타 수단이 사용될 수 있다는 것을 잘 알 것이다.

<218> 예시적인 분산 컴퓨팅 아키텍처

<219> 퍼스널 컴퓨팅과 인터넷의 융합을 고려하여 다양한 분산 컴퓨팅 프레임워크가 개발되었고 개발되고 있는 중이다. 개인 및 기업 사용자 모두는 애플리케이션 및 컴퓨팅 장치에 대한 매끄럽게 상호운용가능하고 웹을 지원하는 인터페이스를 구비하고 있으며, 컴퓨팅 활동이 점점 더 웹 브라우저 또는 네트워크-지향으로 되고 있다.

<220> 예를 들어, MICROSOFT® 의 관리 코드 플랫폼(managed code platform), 즉 .NET은 서버, 웹-기반 데이터 저장 등의 구성-블록 서비스(building-block service), 및 다운로드가능 장치 소프트웨어를 포함한다. 일반적으로 말하면, .NET 플랫폼은 (1) 전 범위의 컴퓨팅 장치들이 함께 동작하게 하고 이들 장치 모두에서 사용자 정보가 자동으로 갱신 및 동기화되게 하는 기능, (2) HTML보다 XML을 더 많이 사용함으로써 가능하게 되는 웹 페이지에 대한 향상된 대화 기능, 및 (3) 예를 들어, 이메일 또는 Office.NET 등의 소프트웨어와 같은 다양한 애플리케이션의 관리를 위한 중앙 시작점으로부터 제품 및 서비스에 대한 맞춤 액세스 및 이들의 사용자에의 전달을 특징으로 하는 온라인 서비스, (4) 정보에의 액세스의 효율성 및 용이성은 물론 사용자들 및 장치들 간의 정보의 동기화를 향상시키는 중앙집중식 데이터 저장, (5) 이메일, 팩스 및 전화 등의 다양한 통신 매체를 통합시키는 기능, (6) 개발자가 재사용가능 모듈을 생성할 수 있음으로써 생산성을 향상시키고 프로그램 오류의 수를 감소시키는 것, 및 (7) 많은 다른 교차-플랫폼 및 언어 통합 특징들을 제공한다.

<221> 본 명세서의 어떤 예시적인 실시예들이 컴퓨팅 장치 상에 존재하는 API(application programming interface) 등의 소프트웨어와 관련하여 기술되어 있지만, 본 발명에 따른 QFX의 실시예들이 관리 코드(.NET 코드 등)에 의해 또한 다른 분산 컴퓨팅 프레임워크에서 가능하게 되는 언어 및 서비스 전부에 포함되거나, 이들에서 지원되거나, 이들을 통해 액세스될 수 있도록, 본 발명의 하나 이상의 부분이 운영 체제, 또는 "미들 맨(middle man)" 객체, 제어 객체, 하드웨어, 펌웨어, 중간 언어 명령어 또는 개체, 기타 등등을 통해서도 구현될 수 있다.

<222> 본 발명의 QFX를 사용할 수 있게 해주는 본 발명을 구현하는 다수의 방식(예를 들어, 적절한 API, 툴킷, 드라이버 코드, 운영 체제, 컨트롤, 독립형 또는 다운로드가능 소프트웨어 객체, 기타 등등)이 있다. 본 발명은 API(또는 다른 소프트웨어 객체)의 관점으로부터는 물론 본 발명에 따라 QFX의 적어도 일부를 구현할 수 있는 데이터 구조, 소프트웨어 또는 하드웨어 객체의 관점으로부터도 본 발명의 사용을 생각하고 있다. 따라서, 본 명세서에 기술된 본 발명의 다양한 구현예는 전체가 하드웨어로 되어 있는, 일부는 하드웨어로 되어 있고 일부는 소

프트웨어로 되어 있는, 또한 소프트웨어로 되어 있는 측면들을 가질 수 있다.

<223> 단어 "예시적인"은 본 명세서에서 일례, 실례 또는 예시로서 역할하는 것을 의미하기 위해 사용된다. 모호함을 피하기 위해, 본 명세서에 개시된 발명 대상은 이러한 예들에 의해 제한되지 않는다. 그에 부가하여, 본 명세서에서 "예시적인" 것으로 기술된 임의의 측면 또는 설계가 반드시 다른 측면 또는 설계보다 선호되거나 유익한 것으로 해석될 필요는 없으며, 당업자라면 잘 알고 있는 등가의 예시적인 구조 및 기술을 배제하기 위한 것도 아니다. 게다가, 용어 "포함하는", "갖는", "내포하는" 및 기타 유사한 단어가 상세한 설명 또는 청구 범위에서 사용되는 한, 모호함을 피하기 위해, 이러한 용어가 임의의 부가적인 또는 다른 구성요소를 배제하지 않고 개방형 전이구(open transition word)처럼 용어 "포함하는"과 유사한 방식으로 포함적인 것으로 보아야 한다.

<224> 상기한 바와 같이, 본 발명의 예시적인 실시예들이 다양한 컴퓨팅 장치 및 네트워크 아키텍처와 관련하여 기술되어 있지만, 기본 개념들은 변환을 구현하는 MFST를 포함하는 것이 바람직한 임의의 컴퓨팅 장치 또는 시스템에 적용될 수 있다. 예를 들어, 본 발명의 QFX는 컴퓨팅 장치의 운영 체제에 적용되거나, 장치 상에 별도의 객체로서, 다른 객체의 일부로서, 재사용가능 컨트롤로서, 서버로부터 다운로드가능한 객체로서, 장치 또는 객체와 네트워크 간의 "미들 맨"으로서, 분산 객체로서, 하드웨어로서, 메모리에 제공될 수 있거나, 이들 중 임의의 것의 조합, 기타 등등일 수 있다. 예시적인 프로그래밍 언어, 이름 및 일례들이 본 명세서에서 다양한 선택 사항을 나타내는 것으로서 선택되어 있지만, 이들 언어, 이름 및 일례들은 제한하기 위한 것이 아니다. 당업자라면 본 발명의 다양한 실시예들에 의해 달성되는 동일한, 유사한 또는 등가의 기능을 달성하는 객체 코드 및 용어를 제공하는 많은 방식이 있다는 것을 잘 알 것이다.

<225> 상기한 바와 같이, 본 명세서에 기술된 다양한 기법들이 하드웨어 또는 소프트웨어와 관련하여 구현될 수 있거나, 적절한 경우, 이 둘의 조합으로 구현될 수 있다. 본 명세서에서 사용되는 바와 같이, 용어 "컴포넌트", "시스템", 기타 등등이 마찬가지로 컴퓨터-관련 개체(하드웨어, 하드웨어와 소프트웨어의 조합, 소프트웨어, 또는 실행 중인 소프트웨어)를 말하기 위한 것이다. 예를 들어, 컴포넌트는 프로세서 상에서 실행 중인 프로세스, 프로세서, 객체, 실행 파일, 실행 쓰레드, 프로그램, 및/또는 컴퓨터일 수 있지만, 이에 한정되지 않는다. 예시로서, 컴퓨터 상에서 실행 중인 애플리케이션 및 이 컴퓨터 둘다가 컴포넌트일 수 있다. 하나 이상의 컴포넌트가 프로세스 및/또는 실행 쓰레드 내에 존재할 수 있고, 컴포넌트가 하나의 컴퓨터 상에 로컬화되어 있고 및/또는 2개 이상의 컴퓨터 간에 분산되어 있을 수 있다.

<226> 따라서, 본 발명의 방법 및 장치 또는 이들의 어떤 측면 또는 그 일부분이 유형 매체(플로피 디스켓, CD-ROM, 하드 드라이브, 또는 임의의 다른 기계-판독가능 저장 매체 등)로 구현된 프로그램 코드(즉, 명령어)의 형태를 취할 수 있으며, 프로그램 코드가 기계(컴퓨터 등)에 로드되어 그에 의해 실행될 때, 그 기계는 본 발명을 실시하는 장치가 된다. 프로그램가능 컴퓨터 상에서의 프로그램 코드 실행의 경우에, 컴퓨팅 장치는 일반적으로 프로세서, 프로세서에 의해 판독가능한 저장 매체(휘발성 및 비휘발성 메모리 및/또는 저장 요소를 포함함), 적어도 하나의 입력 장치, 및 적어도 하나의 출력 장치를 포함한다. 예를 들어, 데이터 처리 API, 소프트웨어 객체, 기타 등등을 사용하여 본 발명의 QFX를 구현하거나 이용할 수 있는 하나 이상의 프로그램이 양호하게는 컴퓨터 시스템과 통신하기 위해 상위 레벨의 절차적 프로그래밍 언어 또는 객체 지향 프로그래밍 언어로 구현된다. 그렇지만, 프로그램(들)은, 원하는 경우, 어셈블리어 또는 기계어로 구현될 수 있다. 어쨌든, 이 언어는 컴파일된 또는 인터프리트된 언어일 수 있고 하드웨어 구현과 결합될 수 있다.

<227> 본 발명의 방법 및 장치는 또한 어떤 전송 매체를 통해(예를 들어, 전기 배선 또는 케이블을 통해, 광섬유를 통해, 또는 임의의 다른 형태의 전송을 통해) 전송되는 프로그램 코드의 형태로 구현되는 통신을 통해 실시될 수 있으며, 이 프로그램 코드가 수신되어 기계(EPROM, 게이트 어레이, PLD(programmable logic device), 클라이언트 컴퓨터, 기타 등등)에 로드되고 그에 의해 실행될 때, 이 기계는 본 발명을 실시하는 장치가 된다. 범용 프로세서 상에서 구현될 때, 프로그램 코드는 프로세서와 결합하여 본 발명의 기능을 호출하는 동작을 하는 독자적인 장치를 제공한다. 그에 부가하여, 본 발명과 관련하여 사용되는 임의의 저장 기술이 항상 하드웨어와 소프트웨어의 조합일 수 있다.

<228> 게다가, 개시된 발명 대상은 본 명세서에 상세히 기술된 측면들을 구현하도록 컴퓨터 또는 프로세서 기반 장치를 제어하는 소프트웨어, 펌웨어, 하드웨어, 또는 이들의 임의의 조합을 생성하기 위해 표준의 프로그래밍 및/또는 엔지니어링 기법을 사용하는 시스템, 방법, 장치, 또는 제조 물품으로서 구현될 수 있다. 용어 "제조 물품"(또는 다른 대안으로서, "컴퓨터 프로그램 제품")은 본 명세서에서 사용되는 경우 임의의 컴퓨터-판독가능 장치, 캐리어, 또는 매체로부터 액세스가능한 컴퓨터 프로그램을 포함하기 위한 것이다. 예를 들어, 컴퓨터 판독가능 매체로는 자기 저장 장치(예를 들어, 하드 디스크, 플로피 디스크, 자기 스트립...), 광 디스크(예를 들

어, CD, DVD), 스마트 카드, 및 플래쉬 메모리 장치(예를 들어, 카드, 스틱)가 있지만, 이에 한정되지 않는다. 그에 부가하여, 전자 메일을 전송 및 수신하는 데 또는 네트워크(인터넷 또는 LAN 등)에 액세스하는 데 사용되는 것과 같은 반송파가 컴퓨터-판독가능 전자 데이터를 전달하는 데 이용될 수 있다는 것이 공지되어 있다.

<229> 상기한 시스템들이 몇개의 컴포넌트들 간의 상호작용과 관련하여 기술되어 있다. 이러한 시스템 및 컴포넌트가 그 컴포넌트들 또는 지정된 서브컴포넌트들, 지정된 컴포넌트 또는 서브컴포넌트들 중 어떤 것, 및/또는 부가의 컴포넌트들을 이상의 것들의 다양한 치환 및 조합에 따라 포함할 수 있다는 것을 잘 알 것이다. 서브컴포넌트들은 또한 (계층적으로) 부모 컴포넌트 내에 포함되어 있기 보다는 다른 컴포넌트와 통신 연결되어 있는 컴포넌트로서 구현될 수 있다. 그에 부가하여, 유의할 점은 하나 이상의 컴포넌트가 통합 기능을 제공하는 단일의 컴포넌트로 결합될 수 있거나 몇개의 서로 다른 서브컴포넌트로 분할될 수 있고 이러한 서브컴포넌트에 통신 연결하여 통합된 기능을 제공하기 위해 임의의 하나 이상의 중간 계층(관리 계층 등)이 제공될 수 있다는 것이다. 본 명세서에 기술된 임의의 컴포넌트들은 또한 본 명세서에 구체적으로 기술되어 있지 않지만 당업자라면 일반적으로 잘 알고 있는 하나 이상의 다른 컴포넌트들과 상호작용할 수 있다.

<230> 상기한 예시적인 시스템과 관련하여, 개시된 발명 대상에 따라 구현될 수 있는 방법들이 도 5의 플로우차트를 참조하면 더 잘 이해된다. 설명의 간결함을 위해, 이 방법들이 일련의 블록으로 도시되고 설명되어 있지만, 본 명세서에 도시되고 기술된 것으로부터 어떤 블록들이 다른 순서로 및/또는 다른 블록들과 동시에 행해질 수 있거나 등가의 기능을 제공할 수 있기 때문에, 청구된 발명 대상이 블록들의 순서에 의해 제한되지 않는다는 것을 잘 알 것이다. 플로우차트를 통해 비순차적이거나 분기된 흐름이 예시되어 있는 경우, 동일하거나 유사한 결과를 달성하는 블록들의 여러가지 다른 분기, 흐름 경로 및 순서가 구현될 수 있다는 것을 잘 알 것이다. 게다가, 도시된 블록들 전부가 이후에 기술되는 방법들을 구현하는 데 꼭 필요한 것은 아닐 수도 있다.

<231> 게다가, 잘 알 것인 바와 같이, 이상의 개시된 시스템 및 이하의 방법들의 여러가지 부분들이 인공 지능 또는 지식 또는 규칙 기반 컴포넌트, 서브컴포넌트, 프로세스, 수단, 방법, 또는 메카니즘(예를 들어, 지원 벡터 기계, 신경망, 전문가 시스템, 베이지안 믿음 네트워크(Bayesian belief network), 퍼지 논리, 데이터 융합 엔진(data fusion engine), 분류기...)을 포함하거나 이들로 이루어져 있을 수 있다. 이러한 컴포넌트는, 그 중에서도 특히, QFX의 일부분을 더욱 적응적인 것은 물론 효율적이며 지능적으로 만들기 위해 어떤 메카니즘 또는 그에 의해 수행되는 프로세스를 자동화할 수 있다.

<232> 본 발명이 여러 도면들의 양호한 실시예들과 관련하여 기술되어 있지만, 다른 유사한 실시예들이 사용될 수 있거나 본 발명을 벗어나지 않고 본 발명의 동일한 기능을 수행하기 위해 여러 수정 및 추가가 기술된 실시예들에 행해질 수 있다는 것을 잘 알 것이다. 예를 들어, 본 발명의 예시적인 네트워크 환경이 네트워크화된 환경(피어-투-피어 네트워크화된 환경 등)과 관련하여 기술되어 있지만, 당업자라면 본 발명이 그에 제한되지 않으며 본 명세서에 기술된 방법들이 임의의 컴퓨팅 장치 또는 환경[게임 콘솔, 핸드헬드 컴퓨터, 휴대용 컴퓨터, 기타 등등(유선이든 무선이든 관계없음)]에 적용될 수 있고 통신 네트워크를 통해 접속되어 이 네트워크를 통해 상호작용하는 임의의 수의 이러한 컴퓨팅 장치에 적용될 수 있다는 것을 잘 알 것이다. 게다가, 특히 무선 네트워크 장치의 수가 계속 증가하고 있기 때문에, 핸드헬드 장치 운영 체제 및 다른 애플리케이션 관련 운영 체제를 비롯한 다양한 컴퓨터 플랫폼이 생각된다는 것이 강조되어야 한다.

산업상 이용 가능성

<233> 예시적인 실시예들이 특정의 프로그래밍 언어 구성자(programming language construct)와 관련하여 본 발명을 이용하는 것에 관한 것이지만, 본 발명은 그에 한정되지 않으며 오히려 어떤 언어로도 구현될 수 있고 액션을 포함하는 임의의 문법을 변환하는 것에 적용된다. 또한, 본 발명은 복수의 처리 칩 또는 장치에서 또는 이들에 걸쳐 구현될 수 있고, 저장이 복수의 장치에 걸쳐 유사하게 실행될 수 있다. 따라서, 본 발명은 임의의 하나의 실시예에 한정되지 않으며 첨부된 청구 범위에 따라 그 범위 및 폭이 해석되어야만 한다.

도면의 간단한 설명

<9> 도 1a는 액션 의미를 보존하기 위해 본 발명에 의해 제공되는 일반적인 변환 프레임워크를 사용하여 FST를 변환하는 예시적인 프로세스를 나타낸 예시적이고 비제한적인 흐름도.

<10> 도 1b는 액션 의미를 보존하기 위해 본 발명에 의해 제공되는 일반적인 변환 프레임워크를 사용하여 FST를 변환하는 예시적인 프로세스를 나타낸 예시적이고 비제한적인 블록도.

<11> 도 1c는 본 발명에 따라 FST를 변환하는 예시적인 프레임워크 및 변환 엔진을 나타낸 예시적이고 비제한적인 블

록도.

<12> 도 2a는 본 발명의 변환 엔진과 관련하여 제공되는 예시적인 인터페이스를 나타낸 도면.

<13> 도 2b 및 도 2c는 각각, 본 발명의 실시예들에 따라 제공되는, PushEnvironment 및 PopEnvironment 메서드의 예시적인 동작을 나타낸 도면.

<14> 도 2d는 본 발명의 변환 엔진과 관련하여 제공되는 다른 예시적인 인터페이스를 나타낸 도면.

<15> 도 2e는 본 발명에 의해 제공되는 변환 프레임워크에서 구현되는 예시적이고 비제한적인 ∈-클로저(closure) 알고리즘을 나타낸 도면.

<16> 도 2f는 본 발명에 의해 제공되는 변환 프레임워크에서 구현되는 액션들을 수용하는 부분집합 구성(subset construction) 기법의 수정예를 나타낸 도면.

<17> 도 3a는 본 발명에 따라 정의되는 강화된 변환 사례 집합의 ACTION-REGEX 사례 또는 $\alpha\, T$ 사례의 예시적인 변환을 나타낸 도면.

<18> 도 3b는 본 발명에 따라 정의되는 강화된 변환 사례 집합의 BIND-NEST 사례 또는 $x : [T]$사례의 예시적인 변환을 나타낸 도면.

<19> 도 3c는 본 발명에 따라 정의되는 강화된 변환 사례 집합의 BIND-NEST-REPEAT 사례 또는 $x : [T]^{*|+|?}$ 사례의 예시적인 변환을 나타낸 도면.

<20> 도 3d는 본 발명에 따라 정의되는 강화된 변환 사례 집합의 $x : (T)$ 사례의 예시적인 변환을 나타낸 도면.

<21> 도 3e는 본 발명에 따라 정의되는 강화된 변환 사례 집합의 $x : T\ Ref$ 사례(호출을 사용한 확장임)의 예시적인 변환을 나타낸 도면.

<22> 도 3f는 본 발명에 따라 정의되는 강화된 변환 사례 집합의 $x : T\ Ref^{*|+|?}$ 사례(호출을 사용한 확장임)의 예시적인 변환을 나타낸 도면.

<23> 도 4a 및 도 4b는 각각 순서있는 패턴(ordered pattern), 즉 목록 패턴(list pattern)과 순서없는 패턴(unordered pattern), 즉 집합 패턴(set pattern) - 이와 관련하여 본 발명의 QFX는 FST에 대한 변환에 걸쳐 순서있는 내포 정보 및 순서없는 내포 정보 둘다를 보존하는 동작을 함 - 을 개념적으로 나타낸 도면.

<24> 도 5a는 순서있는 정보 및/또는 순서없는 정보를 보존하기 위해 본 발명에 의해 제공되는 일반적인 변환 프레임워크를 사용하여 FST를 변환하는 예시적인 프로세스를 나타낸 예시적이고 비제한적인 흐름도.

<25> 도 5b는 순서있는 정보 및/또는 순서없는 정보를 보존하기 위해 본 발명에 의해 제공되는 일반적인 변환 프레임워크를 사용하여 FST를 변환하는 예시적인 프로세스를 나타낸 예시적이고 비제한적인 블록도.

<26> 도 5c 내지 도 5l은 본 발명에 따른 순서있는 또는 라벨있는 정보의 존재 시에 상태 기계들에 대한 변환의 예시적이고 비제한적인 측면들을 나타낸 도면.

<27> 도 6a는 본 발명의 QFX에 따른 예시적인 유한 상태 변환기(finite state transducer)를 나타낸 도면.

<28> 도 6b 및 도 6c는 각각 예시적인 순서있는 내포 및 순서없는 내포 - 이와 관련하여 본 발명의 QFX는 순서있는 내포 정보 및 순서없는 내포 정보 둘다를 보존하는 동작을 함 - 를 표현하는 트리 데이터 구조를 나타낸 도면.

<29> 도 7a는 본 발명에 따른 예시적인 상태 기계 데이터 구조인 상태 천이 테이블로서 FSM을 모델링하는 것의 일례를 나타낸 도면.

<30> 도 7b는 본 발명에 따른 예시적인 상태 기계 데이터 구조인 상태 테이블로서 FSM을 모델링하는 것의 일례를 나타낸 도면.

<31> 도 8a는 상태 천이도에서 FSM을 모델링하는 것의 일례를 나타낸 도면.

<32> 도 8b는 Moore 모델의 일례를 본 발명에 따른 예시적인 유형의 상태 기계 데이터 구조로서 표현하는 변환기를

나타낸 도면.

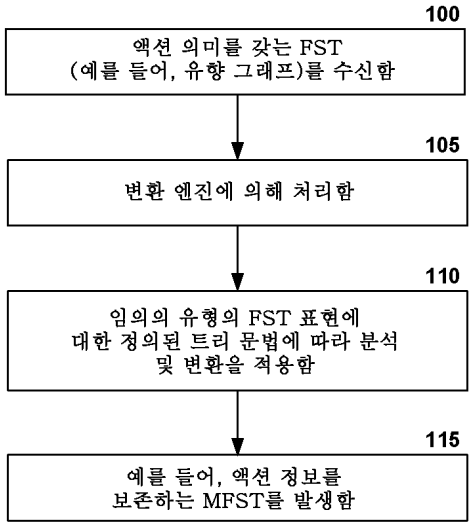
<33> 도 8c는 Mealy 모델의 일례를 본 발명에 따른 예시적인 유형의 상태 기계 데이터 구조로서 표현하는 변환기를 나타낸 도면.

<34> 도 9는 본 발명이 구현될 수 있는 예시적이고 비제한적인 네트워크화된 환경을 나타낸 블록도.

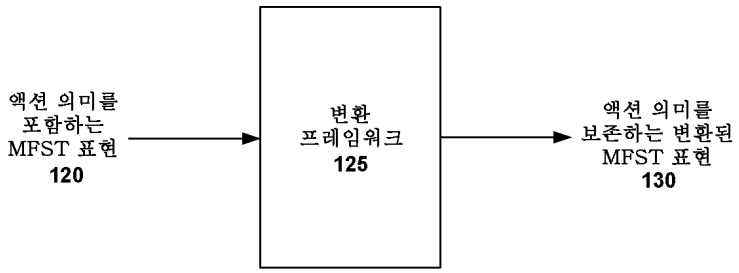
<35> 도 10은 본 발명이 구현될 수 있는 예시적이고 비제한적인 컴퓨팅 시스템 또는 운영 환경을 나타낸 블록도.

도면

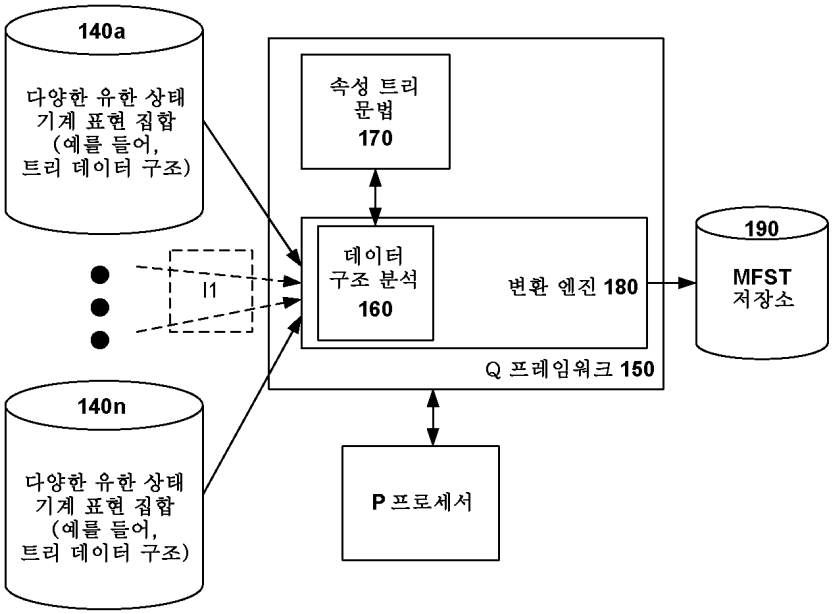
도면1a



도면1b



도면1c



도면2a

```
interface XEControlInstructions {
        IEnvironment CurrentEnvironment { get; set; }
        IContinuation CurrentContinuation { get; }
        IContinuation Call(XEInstance target, IContinuation continuation,
                IEnvironment callingEnvironment);
        IContinuation Return();
        void Mark();
        object Yield();
        IEnvironment NewEnvironment();
        void PushEnvironment(string variableName);
        void PopEnvironment();
        void Exec(ActionReference action);
}
```

200

도면2b

```
void PushEnvironment(string variableName) {
        IEnvironment tempEnv;
        environmentStack.Push(CurrentEnvironment);
        tempEnv=NewEnvironment();
        Bind(varName,tempEnv);
        CurrentEnvironment=tempEnv;
        Mark();
}
```

210

도면2c

```
220  void PopEnvironment() {
         Bind(''term'',Yield());
         environmentStack.Pop();
         CurrentEnvironment=environmentStack.Peek();
     }
```

도면2d

```
230  interface XEEnvironmentInstructions {
         IEnvironment ChildEnv();
         bool Bind(string variableName,object value);
         bool Lookup(string variableName,out object value);
     }
```

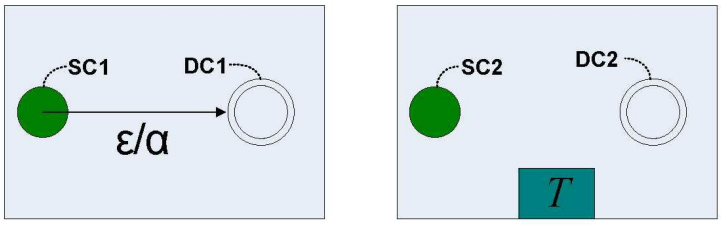
도면2e

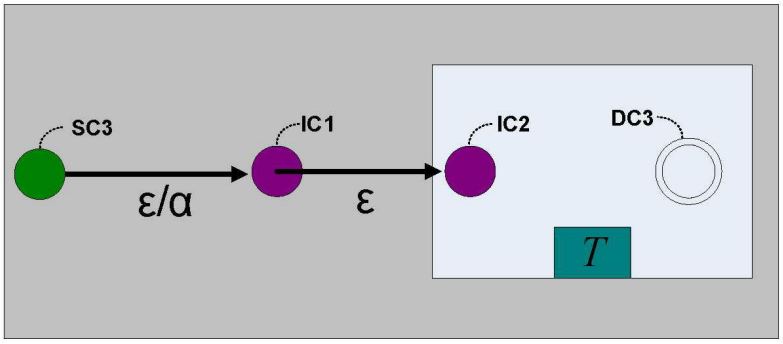
```
240  public static Set<State<T>> EClosure(Set<State<T>> nfaStates,
                                          List<Action> actions) {
         Set<State<T>> eClosure = new Set<State<T>>();
         Stack<State<T>> stack = new Stack<State<T>>();

         foreach (State<T> nfaState in nfaStates) {
             stack.Push(nfaState);
             eClosure.Insert(nfaState);
         }

         while (stack.Count > 0) {
             State<T> nfaSrc = stack.Pop();
             foreach (Transition<T> t in nfaSrc.EpsTransitions) {
(A1)             if (t.Action!=null)
(A2)                 actions.Add(t.Action);
                 foreach (State<T> nfaDst in t.States) {
                     if (!(eClosure.Contains(nfaDst))) {
                         eClosure.Insert(nfaDst);
                         stack.Push(nfaDst);
                     }
                 }
             }
         }
         return (eClosure);
     }
```

도면2f

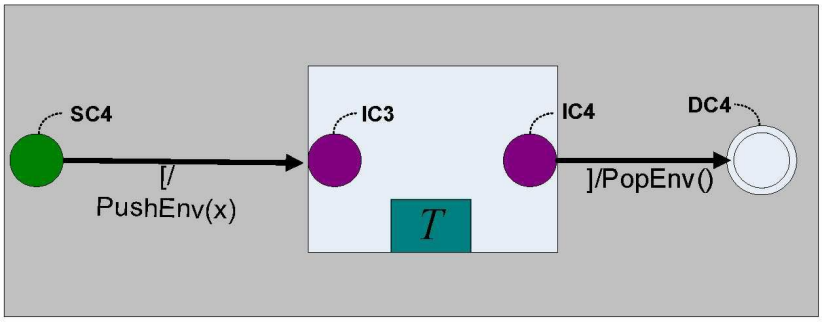
```
public NFA<T> SubsetConstruction(List<T> lexicon) {
        State<T> startDFAState =
                State<T>.SubsetState(State<T>.EClosure(startState));
        NFA<T> dfa = SubsetDFA(startDFAState);

        Queue<State<T>> q = new Queue<State<T>>();
        q.Enqueue(startDFAState);

        while (q.Count > 0) {
                State<T> src = q.Dequeue();
                foreach (T symbol in lexicon) {
250 (B1)                List<Actions> symActions=new List<Actions>();
    (B2)                Set<State<T>> neighbors=src.Move(symbol,symActions);
    (B3)                List<Actions> epsActions=new List<Actions>();
    (B4)                Set<State<T>> eclosure=
                                State<T>.EClosure(neighbors,epsActions);

                        State<T> dst = State<T>.SubsetState(eclosure);
    (B5)                src.AddTransition(symbol,dst,symActions);
    (B6)                src.AddActions(epsActions);
                        return (dfa);
                }
        }
}
```

도면3a

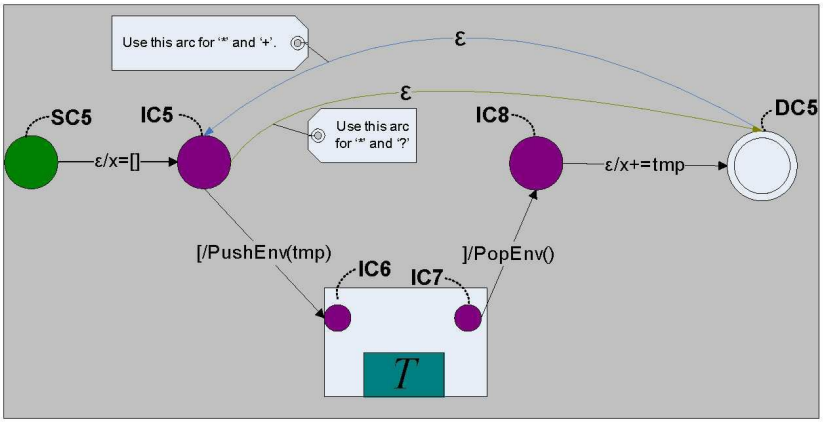




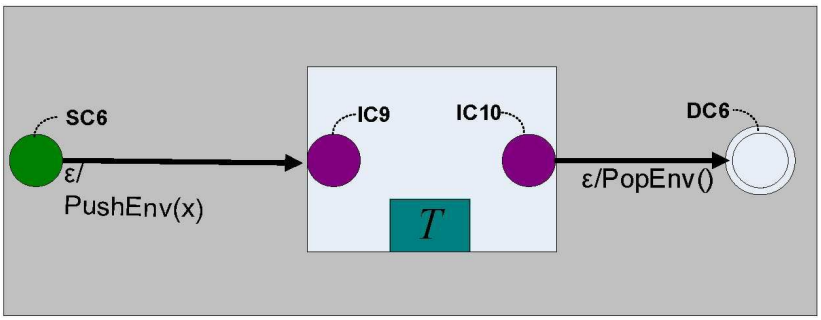
도면3b



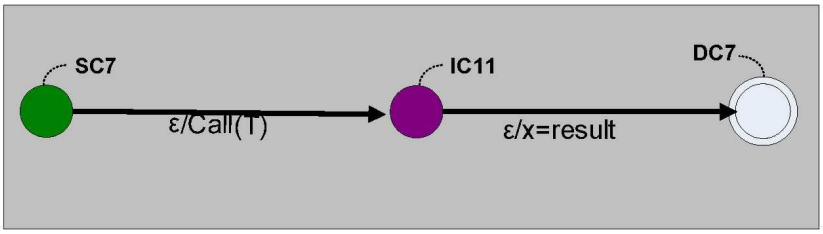
도면3c



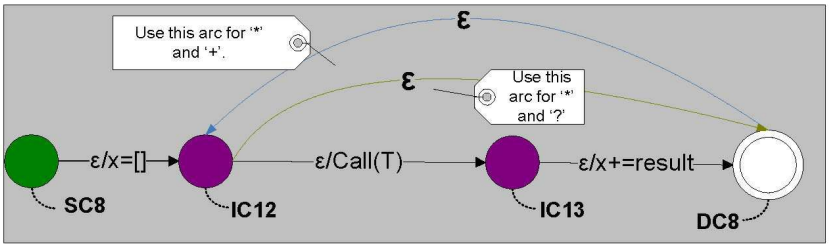
도면3d



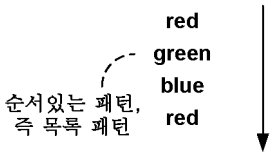
도면3e



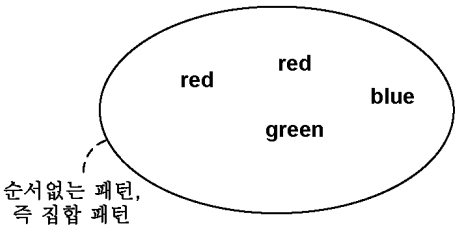
도면3f



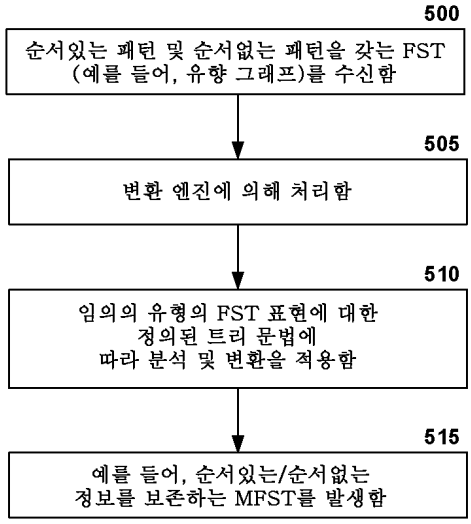
도면4a



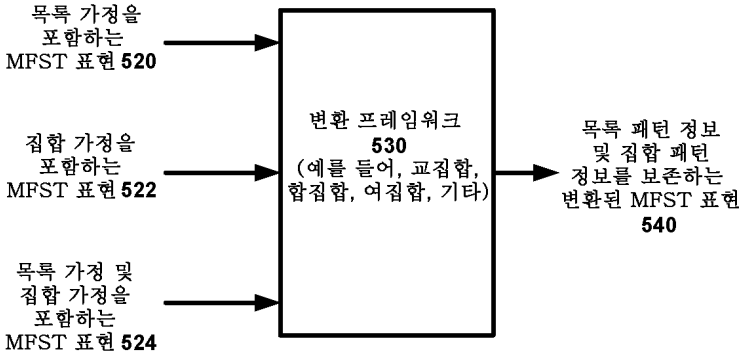
도면4b



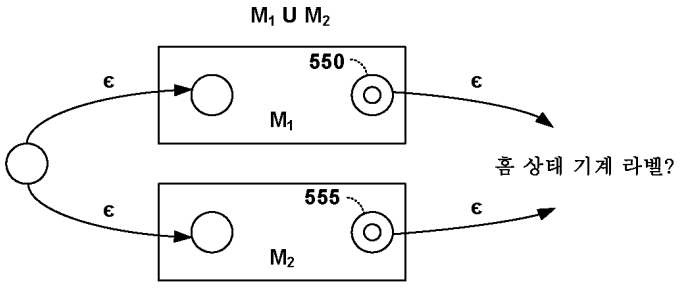
도면5a



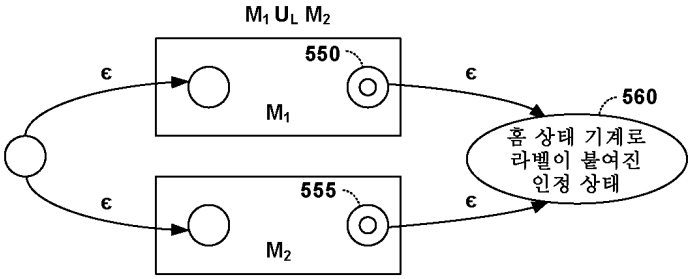
도면5b



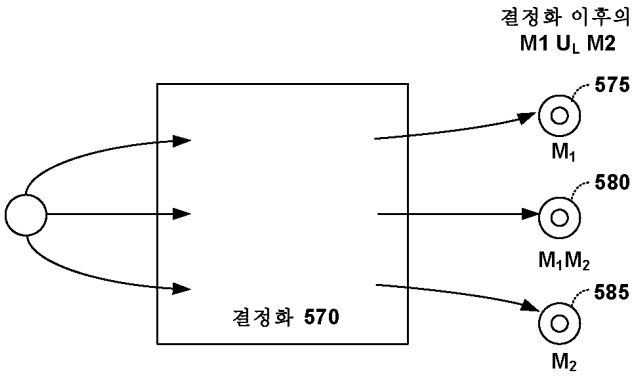
도면5c



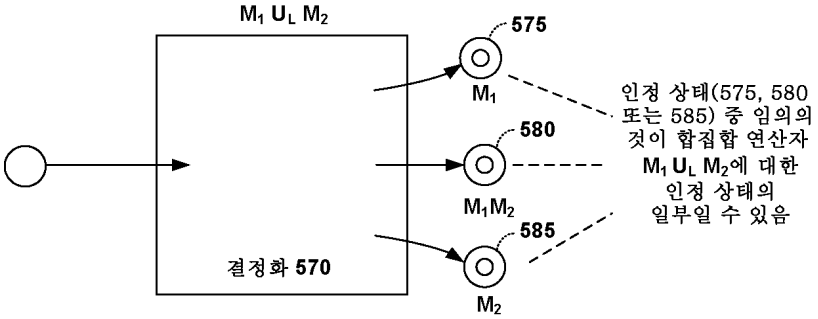
도면5d



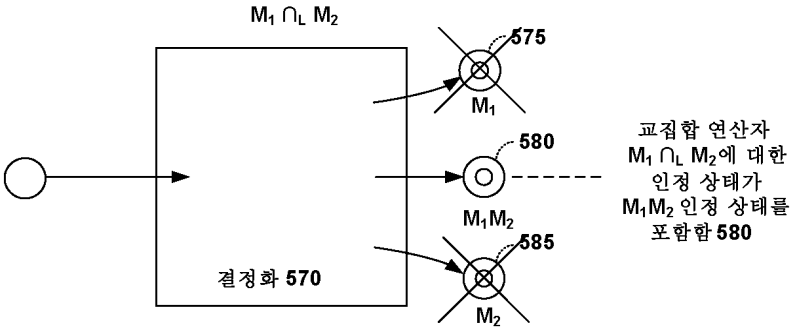
도면5e



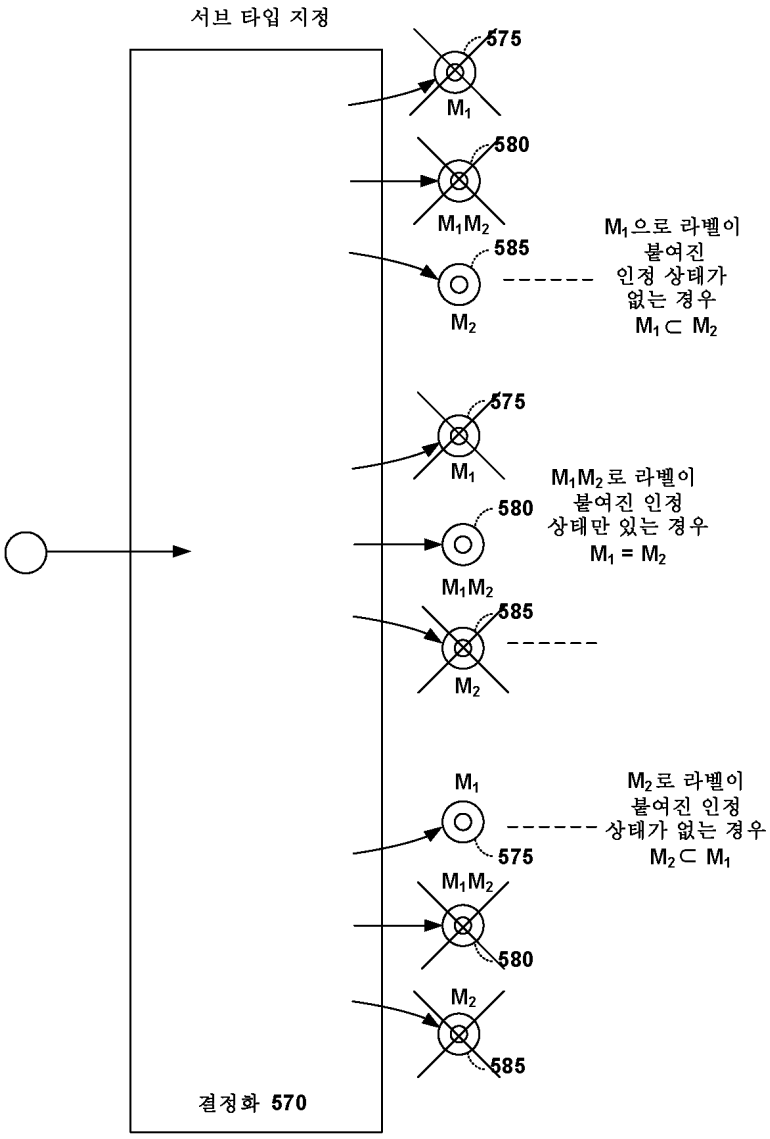
도면5f



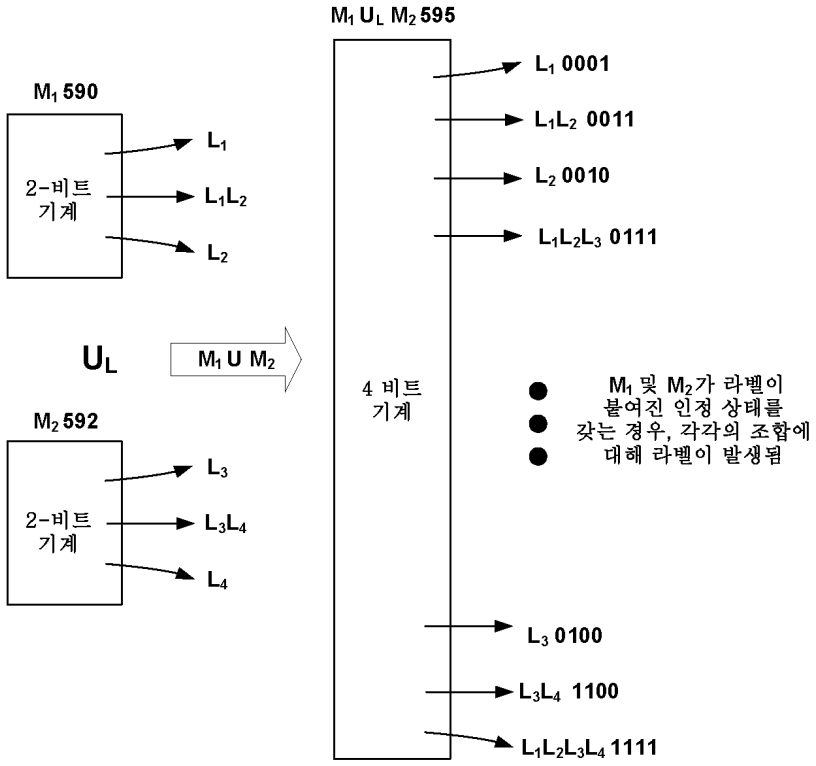
도면5g


$M_1 \cap_L M_2$
575
M_1
580
M_1M_2
585
M_2
결정화 570
교집합 연산자
$M_1 \cap_L M_2$에 대한
인정 상태가
M_1M_2 인정 상태를
포함함 580

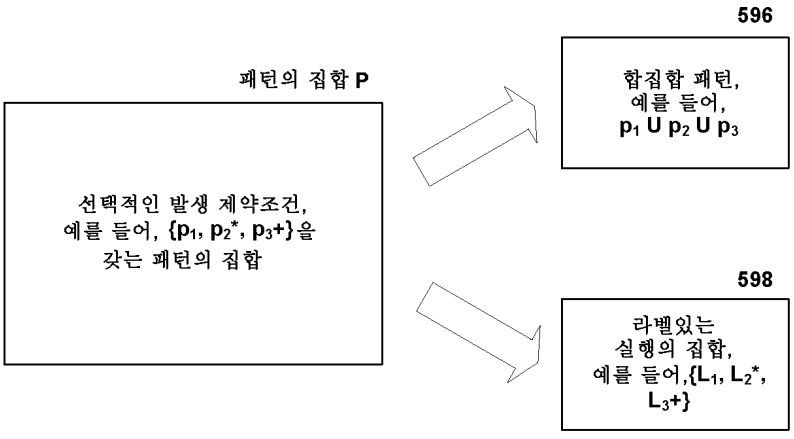
도면5h


서브 타입 지정
575
M_1
580
M_1M_2
585
M_2
결정화 570
M_1으로 라벨이
붙여진
인정 상태가
없는 경우
$M_1 \subset M_2$
M_1M_2로 라벨이
붙여진 인정
상태만 있는 경우
$M_1 = M_2$
M_2로 라벨이
붙여진 인정
상태가 없는 경우
$M_2 \subset M_1$

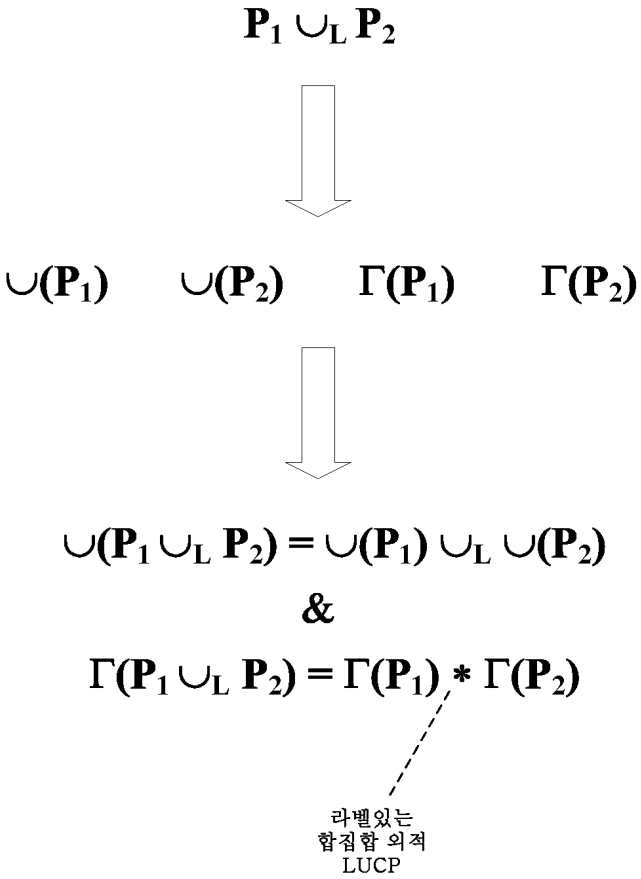
도면5i


M1 UL M2 595
M1 590
2-비트 기계
L1
L1L2
L2
UL
M1 U M2
M2 592
2-비트 기계
L3
L3L4
L4
4 비트 기계
L1 0001
L1L2 0011
L2 0010
L1L2L3 0111
M1 및 M2가 라벨이 붙여진 인정 상태를 갖는 경우, 각각의 조합에 대해 라벨이 발생됨
L3 0100
L3L4 1100
L1L2L3L4 1111

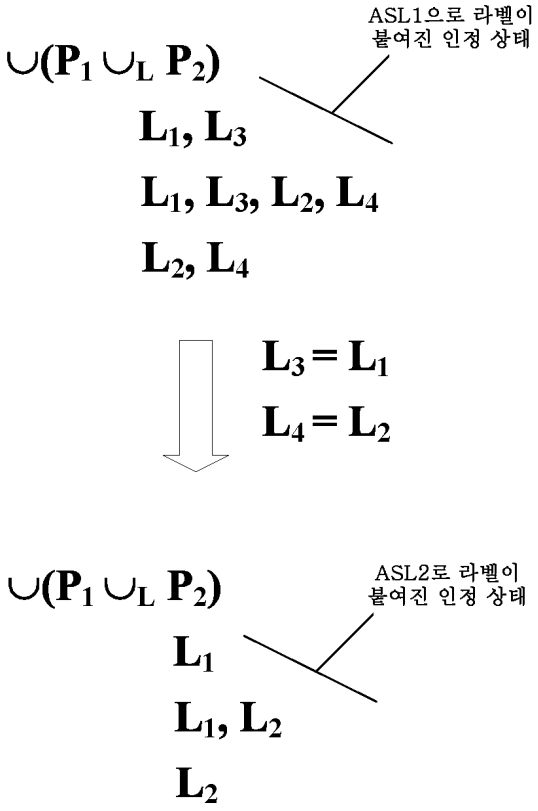
도면5j


596
패턴의 집합 P
선택적인 발생 제약조건, 예를 들어, {p1, p2*, p3+}을 갖는 패턴의 집합
합집합 패턴, 예를 들어, p1 U p2 U p3
598
라벨있는 실행의 집합, 예를 들어,{L1, L2*, L3+}

도면5k

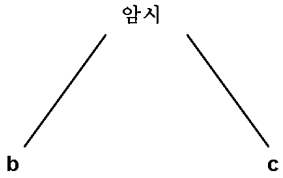

$P_1 \cup_L P_2$
$\cup(P_1)$ $\cup(P_2)$ $\Gamma(P_1)$ $\Gamma(P_2)$
$\cup(P_1 \cup_L P_2) = \cup(P_1) \cup_L \cup(P_2)$
&
$\Gamma(P_1 \cup_L P_2) = \Gamma(P_1) * \Gamma(P_2)$
라벨있는
합집합 외적
LUCP

도면51

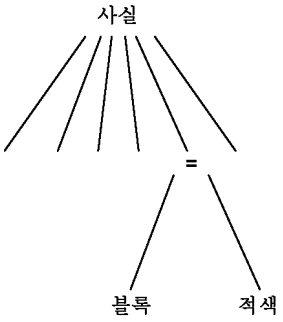


도면6a

도면6b



도면6c



도면7a

상태 천이 테이블 STT1

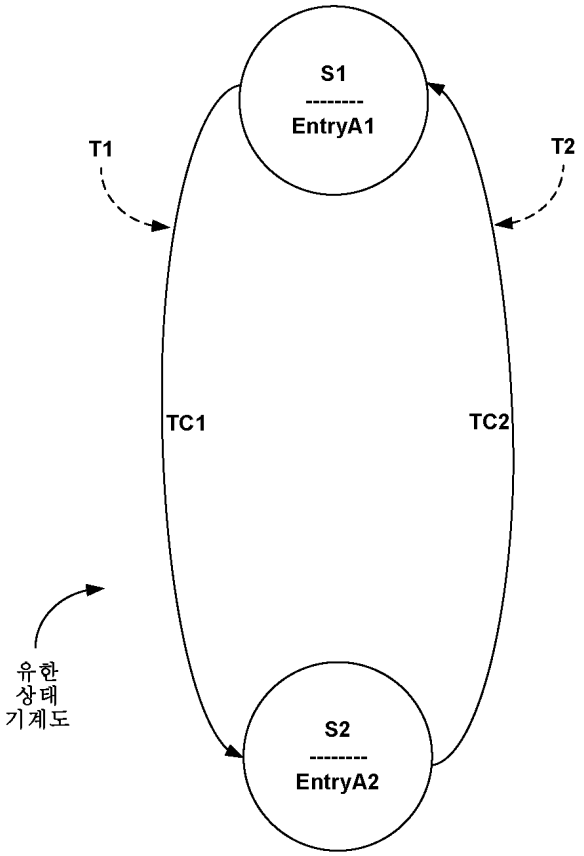
현재 상태/조건	상태 A	상태 B	상태 C
조건 X	...	...	...
조건 Y	...	상태 C	...
조건 Z	...	...	...

도면7b

상태 테이블 ST1

상태 이름 SN	조건 CO	액션 AC
현재 상태	진입 액션	출력 이름(들)
	이탈 액션	출력 이름(들)
	가상 조건	출력 이름(들)
	...	...
그 다음 상태 이름	가상 조건	출력 이름(들)
그 다음 상태 이름	가상 조건	출력 이름(들)
...	...	...

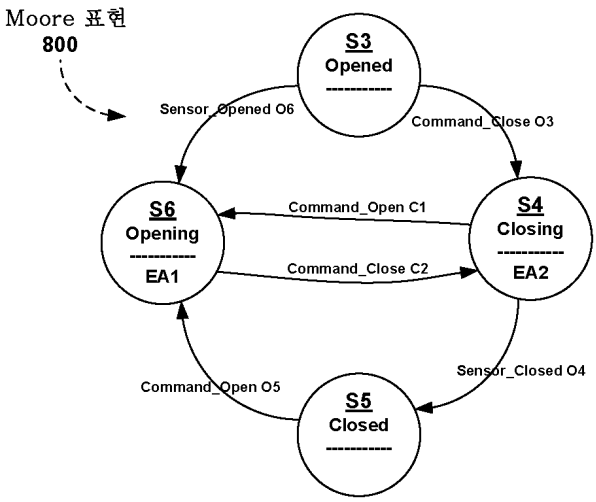
도면8a


S1

EntryA1
T1
T2
TC1
TC2
유한
상태
기계도
S2

EntryA2

도면8b

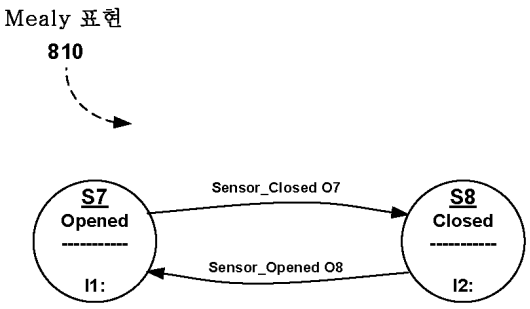

Moore 표현
800
S3
Opened

Sensor_Opened O6
Command_Close O3
S6
Opening

EA1
Command_Open C1
Command_Close C2
S4
Closing

EA2
Command_Open O5
Sensor_Closed O4
S5
Closed

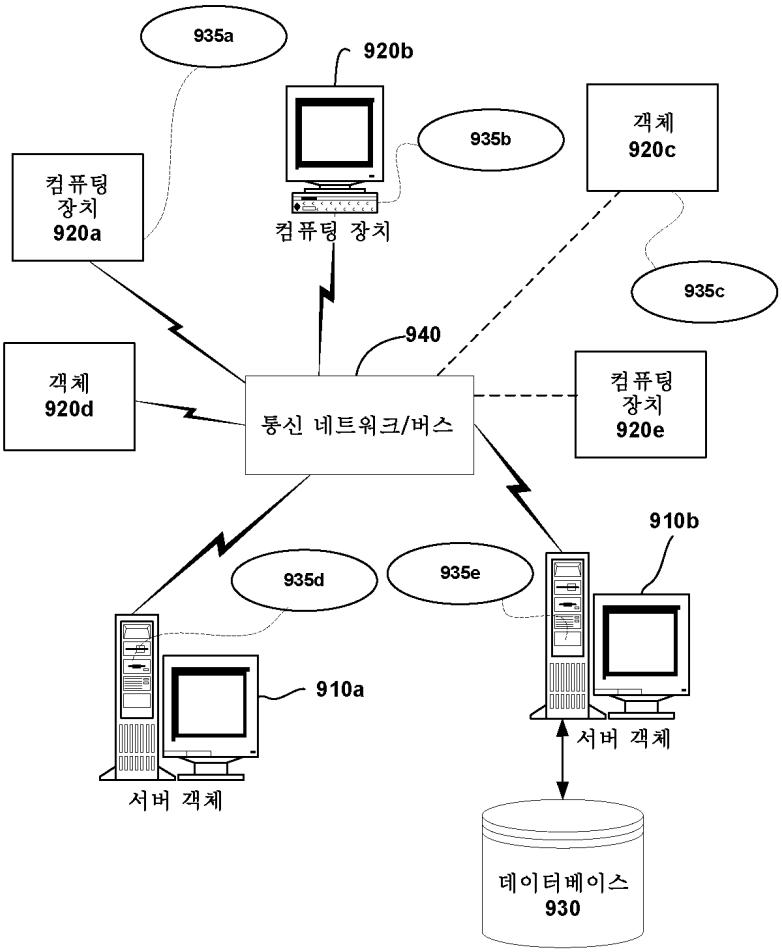
도면8c


Mealy 표현
810
S7
Opened

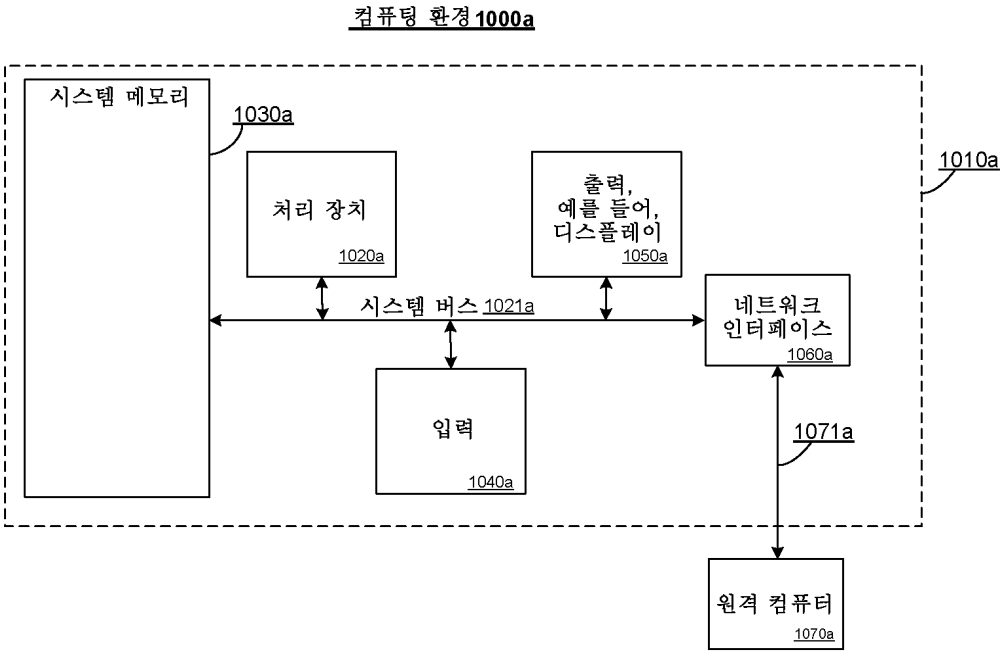
I1:
Sensor_Closed O7
Sensor_Opened O8
S8
Closed

I2:

도면9


935a
920b
935b
객체
920c
컴퓨팅
장치
920a
컴퓨팅 장치
935c
940
객체
920d
통신 네트워크/버스
컴퓨팅
장치
920e
910b
935d
935e
910a
서버 객체
서버 객체
데이터베이스
930

도면10


컴퓨팅 환경 1000a
시스템 메모리
1030a
처리 장치
1020a
출력,
예를 들어,
디스플레이
1050a
1010a
시스템 버스 1021a
네트워크
인터페이스
1060a
입력
1040a
1071a
원격 컴퓨터
1070a