(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2012년07월02일
(11) 등록번호 10-1130351
(24) 등록일자 2012년03월19일

(51) 국제특허분류(Int. Cl.)
G06F 17/20 (2006.01) *G10L 15/00* (2006.01)
(21) 출원번호 10-2005-0068265
(22) 출원일자 2005년07월27일
심사청구일자 2010년07월19일
(65) 공개번호 10-2006-0048800
(43) 공개일자 2006년05월18일
(30) 우선권주장
10/941,439 2004년09월15일 미국(US)
(56) 선행기술조사문헌
US20020069059 A1
US5642519 A
US20030050772 A1

(73) 특허권자
마이크로소프트 코포레이션
미국 워싱턴주 (우편번호 : 98052) 레드몬드 원 마이크로소프트 웨이
(72) 발명자
아세로, 알레잔드로
미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내
콜린스, 레오나르드 앨런
미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내
(뒷면에 계속)
(74) 대리인
제일특허법인

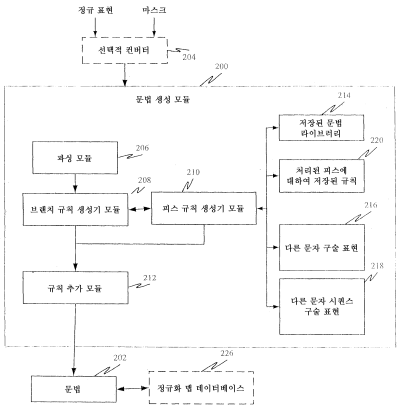
전체 청구항 수 : 총 21 항 심사관 : 신유철

(54) 발명의 명칭 **영숫자 개념용 음성 인식 문법 생성**

(57) 요 약

음성 인식기에 의해 사용하기 적합한 문법을 생성하는 방법 및 시스템이 영숫자 표현의 표시를 수신하는 단계를 포함한다. 예를 들면, 이 표시는 정규 표현 또는 마스크 형태를 취할 수 있다. 이 문법은 표시에 기초하여 생성된다.

대 표 도 - 도2



(72) 발명자

세시스, 마크 엘.

미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내

왕, 예-이

미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내

주, 윤-쳉

미국 98052 워싱턴주 레드몬드 원 마이크로소프트 웨이마이크로소프트 코포레이션 내

특허청구의 범위

청구항 1

음성 인식기에 의해 사용하기 위한 문법을 생성하는 컴퓨터 구현 방법에 있어서,

영숫자 표현(alphanumeric expression)의 표시를 수신하는 단계; 및

상기 표시에 기초하여 문법을 생성하는 단계

를 포함하며,

상기 문법을 생성하는 단계는

상기 표시를 상기 표시의 서브그룹들을 표시하는 브랜치들로 파싱(parsing)하는 단계;

상기 브랜치들의 각각에 대한 문법 규칙 생성 단계;

상기 브랜치들의 각각의 보다 작은 부분을 표시하는 피스들(pieces)을 식별하는 단계; 및

상기 피스들의 각각에 대한 규칙 생성 단계

를 포함하는, 컴퓨터 구현 방법.

청구항 2

제1항에 있어서,

상기 피스들의 각각에 대한 문법 규칙 생성 단계는, 라이브러리에 저장된 규칙을 갖는 피스를 식별하는 단계와, 상기 라이브러리에 저장된 상기 규칙에 기초하여 상기 문법에 대한 규칙을 생성하는 단계를 포함하는, 컴퓨터 구현 방법.

청구항 3

제1항에 있어서,

상기 피스들의 각각에 대한 문법 규칙 생성 단계는, 다른(alternative) 문자 구술 표현에 대한 규칙을 생성하는 단계를 포함하는, 컴퓨터 구현 방법.

청구항 4

제3항에 있어서,

상기 문법을 생성하는 단계는 다른 문자 구술 표현의 인식을 위해 음성 인식기로부터 정규화된 출력을 제공하는 메커니즘을 사용하는 단계를 포함하는, 컴퓨터 구현 방법.

청구항 5

제4항에 있어서,

상기 문법을 생성하는 단계는 다른 문자 구술 표현의 인식을 위해 음성 인식기로부터 정규화된 출력을 제공하도록 상기 문법에 정규화 정보를 제공하는 단계를 포함하는, 컴퓨터 구현 방법.

청구항 6

제5항에 있어서,

상기 문법을 생성하는 단계는 상기 문법에 관련된 데이터베이스를 생성하는 단계를 포함하고, 상기 데이터베이스는 정규화 정보를 갖는, 컴퓨터 구현 방법.

청구항 7

제1항에 있어서,

상기 피스들의 각각에 대한 문법 규칙 생성 단계는, 다른 문자 시퀀스 구술 표현에 대한 규칙을 생성하는 단

계를 포함하는, 컴퓨터 구현 방법.

청구항 8

제1항에 있어서,

상기 피스들의 각각에 대한 문법 규칙 생성 단계는, 상기 문법 규칙이 이전에 기초한 표시에서 제2 피스와 동일한 제1 피스를 식별하는 단계와, 상기 제2 피스에 기초한 규칙을 사용하여 상기 제1 피스에 기초한 규칙을 생성하는 단계를 포함하는, 컴퓨터 구현 방법.

청구항 9

제1항에 있어서,

상기 문법을 생성하는 단계는 프리픽스 최적화된 문법 규칙(prefix optimized grammer rules)을 생성하는 단계를 포함하는, 컴퓨터 구현 방법.

청구항 10

제1항에 있어서,

상기 표시는 W3C에 의해 정의된 정규 표현 또는 마스크의 형태인, 컴퓨터 구현 방법.

청구항 11

음성 인식기에 의해 사용하기 위한 문법을 생성하도록, 컴퓨터 상에 동작가능한 명령어를 갖는 컴퓨터 판독가능 기록매체에 있어서,

상기 명령어는

컴퓨터로 하여금 제1항의 컴퓨터 구현 방법의 모든 단계를 실행하도록 하는, 컴퓨터 판독가능 기록매체.

청구항 12

제11항에 있어서,

상기 표시는 W3C에 의해 정의된 정규 표현 또는 마스크인, 컴퓨터 판독가능 기록매체.

청구항 13

제11항에 있어서,

상기 피스 각각에 대한 규칙 생성 단계는, 다른 문자 구술 표현에 대한 규칙을 생성하는 단계; 다른 문자 시퀀스 구술 표현에 대한 규칙을 생성하는 단계; 및 라이브러리에 저장된 규칙을 갖는 피스를 식별하고 상기 라이브러리에 저장된 규칙에 따라 상기 문법에 대한 규칙을 생성하는 단계 중 적어도 하나를 포함하는, 컴퓨터 판독가능 기록매체.

청구항 14

제13항에 있어서,

상기 문법을 생성하는 단계는, 다른 문자 구술 표현과 다른 문자 시퀀스 구술 표현 중 적어도 하나의 인식을 위해 음성 인식기로부터 정규화된 출력을 제공하는 메커니즘을 사용하는 단계를 포함하는, 컴퓨터 판독가능 기록매체.

청구항 15

제14항에 있어서,

상기 문법을 생성하는 단계는, 다른 문자 구술 표현과 다른 문자 시퀀스 구술 표현 중 적어도 하나의 인식을 위해 음성 인식기로부터 정규화된 출력을 제공하도록 상기 문법에 정규화 정보를 제공하는 단계를 포함하는, 컴퓨터 판독가능 기록매체.

청구항 16

제15항에 있어서,

상기 문법을 생성하는 단계는, 상기 문법에 관련된 데이터베이스를 생성하는 단계를 포함하고, 상기 데이터베이스는 다른 문자 구술 표현과 다른 문자 시퀀스 구술 표현 중 적어도 하나에 대한 정규화 정보를 갖는, 컴퓨터 판독가능 기록매체.

청구항 17

음성 인식기에 의해 사용하기 위한 문법을 생성하는 컴퓨터 시스템에 있어서,

표시의 서브그룹들을 표시하는, 상기 표시의 브랜치들을 식별하고 영숫자 표현의 상기 표시를 수신하도록 구성된 파싱 모듈;

상기 브랜치들에 기초하여 상기 문법에 대한 규칙을 생성하도록 구성된 브랜치 규칙 생성기 모듈; 및

상기 브랜치들 각각의 보다 작은 부분들을 표시하는 피스들을 식별하고 상기 피스들 각각에 기초하여 상기 문법에 대한 규칙을 생성하도록 구성된 피스 규칙 생성기 모듈

을 포함하는, 컴퓨터 시스템.

청구항 18

제17항에 있어서,

피스들에 기초하여 문법 규칙을 저장하는 라이브러리를 더 포함하고,

상기 피스 규칙 생성기 모듈은 상기 라이브러리에 저장된 규칙을 갖는 피스에 대응하는 표시의 피스를 식별하도록 구성되고,

상기 피스 규칙 생성기 모듈은 상기 라이브러리에서 상기 규칙에 기초하여 문법에 대한 규칙을 생성하도록 구성된, 컴퓨터 시스템.

청구항 19

제18항에 있어서,

상기 피스 규칙 생성기 모듈은 처리된 표시의 제1 피스에 대하여 상기 라이브러리에 규칙을 저장하도록 구성되고,

상기 피스 규칙 생성기 모듈은, 상기 표시의 제1 피스와 동일한 표시의 제2 피스를 식별하고, 상기 표시의 제1 피스에 대하여 상기 라이브러리 내의 규칙에 기초하여 상기 표시의 제2 피스에 대한 규칙을 생성하도록 구성된, 컴퓨터 시스템.

청구항 20

제17항에 있어서,

상기 피스 규칙 생성기 모듈은 다른 문자 구술 표현과 다른 문자 시퀀스 구술 표현 중 적어도 하나에 대한 규칙을 생성하도록 구성된, 컴퓨터 시스템.

청구항 21

제20항에 있어서,

상기 피스 규칙 생성기 모듈은 다른 문자 구술 표현과 다른 문자 시퀀스 구술 표현 중 적어도 하나에 대하여 상기 문법에 정규화 정보를 제공하도록 구성된, 컴퓨터 시스템.

청구항 22

삭제

청구항 23

삭제

청구항 24

삭제

청구항 25

삭제

청구항 26

삭제

청구항 27

삭제

청구항 28

삭제

청구항 29

삭제

청구항 30

삭제

청구항 31

삭제

명 세 서

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

[0019] 본 발명은 음성 인식에 관한 것이다. 보다 구체적으로는, 본 발명은 영숫자 개념용 음성 인식 문법의 자동 생성에 관한 것이다.

[0020] 음성 인식 시스템은 회사 및 기관에 의해 점차 사용되어 비용을 감소하고, 고객 서비스를 향상시키며, 및/또는 작업을 전부 또는 일부 자동화한다. 이러한 시스템은 단독형 데스크탑 머신, 네트워크 장치 및 모바일 핸드헬드 컴퓨팅 장치에 결친 광범위한 컴퓨팅 장치 상에서 사용되고 있다. 음성 인식은 애플리케이션 개발자에 대한 자연 사용자 인터페이스를 제공한다. 예를 들면, 핸드헬드 모바일 장치와 같은 컴퓨터 장치에 있어서, 완전한 영숫자 키보드는 컴퓨팅 장치의 크기를 상당히 증가시키지 않으면 비실용적이다. 따라서, 음성 인식은 소형 장치에 대한 편리한 입력 방법을 제공하며, 또한 사용자가 간단한 전화기 등을 통해 컴퓨터에 원격 액세스할 수 있게 한다.

[0021] 필수적인 것은 아니지만 음성 인식이 보다 광범위하게 허용됨에 따라, 유연하고, 정확하며, 음성 인에이블되는 애플리케이션을 신속하고 효율적으로 생성할 필요가 있다. 구술 언어 이해 모델에 관한 연구는 이러한 시스템이 시스템과 사용자 간의 혼합된 주도 대화(mixed-initiative dialog)를 가능하게 하기 때문에 유연성을 달성하였다. 이러한 시스템과 연구는 항공 여행 정보 시스템의 영역에서 "비용이 400달러 이하인 화요일자 시애틀에서 보스톤까지"와 같은 예를 들면 "항공편 보기"인 다수의 어구 의미 단위(슬롯)를 포함하는 명령을 모델링할 때 정확성을 달성하지만, 날짜, 시간, 신용 카드 번호, 비행기 번호 등과 같은 로우 레벨 개념에 대한 어구 모델의 취득은 거의 연구하지 않았다. 그 대신, 이들은 해결을 위해 문법 라이브러리와 데이터베이스 엔트리(예를 들면, 애플리케이션 데이터베이스로부터의 도시명)를 이용하였다.

[0022] 그럼에도 불구하고, 지금까지 전개된 다수의 구술 언어 시스템은 시스템 주도, 지향 대화 시스템이다. 이러

한 시스템에서, 대부분의 문법 개발 노력은 로우 레벨 개념에 제공된다. 문법 라이브러리와 데이터베이스 엔트리가 경쟁력있는 솔루션이기는 하지만, 이들이 문제를 완전히 해결하지는 않는다. 예를 들면, 문법 라이브러리 개발자는 모든 가능성있는 도메인 특정 개념을 예측하여 이들에 대한 문법을 미리 구성할 수 없다. 또한, 데이터베이스 엔트리의 올바른 철자 형태는 종종 음성 인식 문법으로서 충분하지 않다. 예를 들면, 적절한 음성 인식 문법은 영숫자 문자열에 대한 다양한 다른 구술 표현을 모델링하는데 필요하다. 애플리케이션이 부품 번호(parts number)를 인식할 필요가 있고 "ABB123"이 부품 번호 중 하나라고 가정하자. 음성 인에이블 시스템은 "A B B one two three" 또는 "A double B one twenty three"라고 말해진 경우에도 이러한 부품 번호를 인식할 수 있어야 한다.

[0023] 따라서, 부품 번호와 자동차 면허 번호와 같은 영숫자 개념에 대한 문법 구현은 가장 도전적인 작업 중 하나라는 것이 공지되어 있다. 이러한 시도 중 하나는 단일 상태 유한 상태 모델에 기초한 단순 문법을 사용하는 것이었다. 이러한 모델은 각 문자(A 내지 Z)와 각 숫자(0 내지 9)에 대한 루프를 갖는다. 그러나, 이 모델은 통상 문법이 타겟 서브 언어의 특정성을 캡처하지 않는다는 등의 이유로 인해 잘 동작하지 않는다. 따라서, 이 모델의 난점은 예상보다 훨씬 어렵다. 예를 들면, 부품 번호가 문자 "B"로 항상 시작한 것으로 알려진 경우, 문법은 "E"를 "D", "E", "G" 및 "P"와 혼동하는 인식 에러는 결코 발생하지 않는다는 제한을 명시적으로 모델링하여야 한다.

[0024] 또한, 단순 문법은 많은 유형의 문자열에 대하여 언어적 표현의 다양성을 모델링하지 않는다. 상기 예에서, "ABB123"의 "ABB" 부분과 "123" 부분 모두 상이하지만 매우 일반적인 방식으로 제공될 수 있으며, 이들 중 다수는 단순 문법에 의해 모델링되지 않는다.

[0025] 더욱이, "-", "*" 등과 같은 특별 문자는 종종 부품 번호와 같은 영숫자 시퀀스에서 나타난다. 이는 일반 영숫자 문법은 이러한 경우에 커스텀화될 것을 요구한다.

[0026] 상기 문제의 관점에서, 개발자는 종종 특정 영숫자 개념에 대한 그들 자신의 문법을 기입하도록 강요받는다. 이 프로세스는 번거롭고 실수하기 쉽다. 문법 라이브러리와는 달리, 덜 숙련된 개발자에 의해 작성된 문법은 종종 최적화되지 않아서, 디코더에 의해 사용되는 경우 열악한 성능을 갖는다.

[0027] 따라서, 상기 문제점의 하나, 일부 또는 전부를 해결하는 영숫자 문법을 생성하는 시스템 및 방법이 요구된다.

발명이 이루고자 하는 기술적 과제

[0028] 음성 인식기에 의해 사용하기 적합한 문법을 생성하는 방법 및 시스템은 영숫자 표현의 표시를 수신하는 단계를 포함한다. 예를 들면, 이 표시는 정규 표현 또는 마스크의 형태를 취할 수 있다. 이 문법은 표시에 기초하여 생성된다. 이러한 방식으로, 개발자는 예를 들면 영숫자 표현에 대한 정규 표현을 제공할 수 있으며, 시스템은 문법을 자동 구성한다.

[0029] 다른 실시예에서, 문법은 다른 문자 구술 표현(예를 들면, "0"에 대하여 "제로" 또는 "영"이라 말할 수 있음), 및/또는 다른 문자 시퀀스 구술 표현(예를 들면, "AA",에 대하여 "A A" 또는 "double A"라고 말할 수 있음)에 적합할 수 있다. 이러한 경우, 문법은 정규화 정보를 제공하도록 변형되어 정규 출력이 문법을 사용하여 음성 인식기에 의해 출력될 수 있다. 또다른 실시예에서, 이 문법은 프리픽스 최적화로 구성된다.

[0030] 문법의 품질은 궁극적으로 표시가 제공되는 방식에 의해 결정되지만, 여기서 설명한 방법 및 시스템은 개발자에게 특히 특정 애플리케이션에 고유할 수 있는 영숫자 표현에 대한 문법 개발의 힘든 작업을 덜어 준다. 이러한 방식으로, 영숫자 표현에 대한 문법은 문법 개발자의 고유 경험없이 보다 빨리 개발될 수 있다.

발명의 구성 및 작용

[0031] 본 발명은 영숫자 개념 또는 표현에 대한 문법을 생성하는 시스템, 모듈 및 방법에 관한 것이다. 그러나, 본 발명을 보다 상세히 설명하기 전에, 본 발명이 사용될 수 있는 하나의 예시적인 실시예를 우선 설명한다.

[0032] 예시적인 운영 환경

[0033] 도 1은 본 발명이 구현될 수 있는 적절한 컴퓨팅 시스템 환경(100)의 일 예를 나타낸다. 컴퓨팅 시스템 환경(100)은 적절한 컴퓨팅 환경의 단지 일 예이며 본 발명의 사용 또는 기능의 범위에 대한 임의의 한정을 암시하려는 것은 아니다. 또한, 컴퓨팅 환경(100)은 예시적인 운영 환경(100)에 나타낸 컴포넌트 중 임의의 하나 또는 조합에 관한 임의의 의존성 또는 요건을 갖는 것으로서 해석되어서는 안 된다.

[0034] 본 발명은 수많은 다른 범용 또는 특정 목적 컴퓨팅 시스템 환경 또는 구성으로 동작한다. 본 발명에 사용하기에 적합할 수 있는 공지된 컴퓨팅 시스템, 환경 및/또는 구성의 예는 범용 컴퓨터, 서버 컴퓨터, 핸드 헬드 또는 랩탑 장치, 마이크로프로세서 시스템, 마이크로프로세서 기반 시스템, 셋탑 박스, 프로그래머블 소비자 전자제품, 네트워크 PC, 미니컴퓨터, 메인프레임 컴퓨터, 상기 시스템 또는 장치 중 임의의 것을 포함하는 분산 컴퓨팅 환경 등을 포함하지만 이에 한정되지 않는다.

[0035] 본 발명은 컴퓨터에 의해 실행되는, 프로그램 모듈과 같은, 컴퓨터 실행가능 명령의 일반적인 경우에 대하여 설명할 수 있다. 통상, 프로그램 모듈은 특정 작업을 수행하거나 특정 추상형 데이터를 구현하는 루틴, 프로그램, 객체, 컴포넌트, 데이터 구조 등을 포함한다. 당업자는 여기서의 설명 및/또는 도면을 후술하는 컴퓨터 판독가능 매체의 임의의 형태로서 구체화될 수 있는 컴퓨터 실행가능 명령으로서 구현할 수 있다.

[0036] 본 발명은 또한 작업이 통신 네트워크를 통해 연결된 원격 처리 장치에 의해 수행되는 분산 컴퓨팅 환경에서 실시될 수 있다. 분산 컴퓨팅 환경에서, 프로그램 모듈은 메모리 저장 장치를 포함하는 로컬 및 원격 컴퓨터 저장 매체에 배치될 수 있다.

[0037] 도 1을 참조하면, 본 발명을 구현하는 예시적인 시스템은 컴퓨터(110) 형태의 범용 컴퓨팅 장치를 포함한다. 컴퓨터(110)의 컴포넌트는 처리부(120), 시스템 메모리(130), 시스템 메모리를 포함하는 다양한 시스템 컴포넌트를 처리부(!20)에 결합하는 시스템 버스(!21)를 포함할 수 있다. 시스템 버스(121)는 메모리 버스 또는 메모리 컨트롤러, 주변 버스 및 다양한 버스 아키텍처 중 임의의 것을 사용하는 로컬 버스를 포함하는 여러 유형의 버스 구조 중 임의의 것일 수 있다. 예를 들면, - 한정이 아님 -, 이러한 아키텍처는 산업 표준 아키텍처(ISA) 버스, 마이크로 채널 아키텍처(MCA) 버스, 개선된 ISA(EISA) 버스, 비디오 전자 표준 협회(VESA) 로컬 버스, 및 메자닌 버스(Mezzanine bus)로도 불리는 주변 컴포넌트 상호접속(PCI) 버스를 포함한다.

[0038] 컴퓨터(110)는 통상 다양한 컴퓨터 판독가능 매체를 포함한다. 컴퓨터 판독가능 매체는 컴퓨터(110)에 의해 액세스될 수 있는 임의의 이용가능 매체일 수 있으며, 휘발성 및 비휘발성, 분리형 및 비분리형 매체를 모두 포함한다. 예를 들면, - 한정이 아님 -, 컴퓨터 판독가능 매체는 컴퓨터 저장 매체와 통신 매체를 포함할 수 있다. 컴퓨터 저장 매체는 컴퓨터 판독가능 명령, 데이터 구조, 프로그램 모듈 또는 기타 데이터와 같은 정보의 저장을 위한 임의의 방법 또는 기술로 구현되는 휘발성 및 비휘발성, 분리형 및 비분리형을 모두 포함한다. 컴퓨터 저장 매체는 RAM, ROM, EEPROM, 플래시 메모리 또는 다른 메모리 기술, CD-ROM, 디지털 다기능 디스크(DVD) 또는 다른 광 디스크 스토리지, 자기 카세트, 자기 테이프, 자기 디스크 스토리지, 또는 다른 자기 스토리지 장치, 또는 원하는 정보를 저장하는데 사용될 수 있고 컴퓨터(100)에 의해 액세스될 수 있는 임의의 다른 매체를 포함하지만 이에 한정되지 않는다. 통신 매체는 통상 컴퓨터 판독가능 명령, 데이터 구조, 프로그램 모듈 또는 기타 데이터를 반송파 WAV 또는 다른 전송 메커니즘과 같은 변조된 데이터 신호로 구체화하며, 임의의 정보 전달 매체를 포함한다. "변조된 데이터 신호(modulated data signal)"라는 용어는 하나 이상의 특성 신호의 정보를 인코딩하는 방식으로 설정 또는 변경되도록 하는 신호를 의미한다. 예를 들면, - 한정이 아님 -, 통신 매체는 유선 네트워크 또는 다이렉트 와이어드 접속과 같은 유선 매체와 어쿠스틱, RF, 적외선 및 다른 무선 매체와 같은 무선 매체를 포함한다. 상기의 것의 임의의 조합이 컴퓨터 판독가능 매체의 범위 내에 또한 포함되어야 한다.

[0039] 시스템 메모리(130)는 판독 전용 메모리(ROM; 131)와 랜덤 액세스 메모리(RAM; 132)와 같은 휘발성 및/또는 비휘발성 메모리의 형태의 컴퓨터 저장 매체를 포함한다. 기본 입출력 시스템(133; BIOS)은 시동 시와 같이 컴퓨터(110) 내의 요소들 간의 정보를 전송할 수 있게 하는 기본 루틴을 포함하며, 통상 ROM(131)에 저장된다. RAM(132)은 통상 처리부에 즉시 액세스가능하고 및/또는 현재 이에 동작하는 데이터 및/또는 프로그램을 포함한다. 예를 들면, - 한정이 아님 -, 도 1은 운영 체계(134), 애플리케이션 프로그램(135), 다른 프로그램 모듈(136), 및 프로그램 데이터(137)를 나타낸다.

[0040] 컴퓨터(110)는 또한 다른 분리형/비분리형 휘발성/비휘발성 컴퓨터 저장 매체를 포함할 수 있다. 단지 예를 들면, 도 1은 비분리형, 비휘발성 자기 매체로부터 판독하거나 이에 기입하는 하드 디스크 드라이브(141), 분리형, 비휘발성 자기 디스크(152)로부터 판독하거나 이에 기입하는 자기 디스크 드라이브(151), 및 CD 또는 다른 광 매체와 같은 분리형, 비휘발성 광 디스크(156)로부터 판독하거나 이에 기입하는 광 디스크 드라이브(155)를 나타낸다. 예시적인 운영 환경에서 사용될 수 있는 다른 분리형/비분리형, 휘발성/비휘발성 컴퓨터 저장 매체는 자기 테이프 카세트, 플래시 메모리 카드, 디지털 다기능 디스크, 디지털 비디오 테이프, 고체 상태 RAM, 고체 상태 ROM 등을 포함하지만 이에 한정되지 않는다. 하드 디스크 드라이브(141)는 인터페이스(140)와 같은 비분리형 메모리 인터페이스를 통해 시스템 버스(121)에 통상 접속되고, 자기 디스크 드라이브(151)와 광 디스크 드라이브(155)는 통상 인터페이스(150)와 같은 분리형 메모리 인터페이스에 의해 시스템

버스(121)에 접속된다.

[0041] 도 1에 나타내고 상술한 드라이브 및 이들의 관련 컴퓨터 저장 매체는 컴퓨터 판독가능 명령, 데이터 구조, 프로그램 모듈 및 컴퓨터(110)에 대한 기타 데이터를 제공한다. 도 1에서, 예를 들면, 하드 디스크 드라이브(141)는 운영 체계(144), 애플리케이션 프로그램(145), 다른 프로그램 모듈(147), 및 프로그램 데이터(147)를 저장하는 것으로서 나타낸다. 이들 컴포넌트는 운영 체계(134), 애플리케이션 프로그램(135), 다른 프로그램 모듈(136), 및 프로그램 데이터(137)와 동일하거나 상이할 수 있다. 운영 체계(144), 애플리케이션 프로그램(145), 다른 프로그램 모듈(146), 및 프로그램 데이터(147)는 최소한 이들이 상이한 복사본임을 나타내기 위해 여기서 상이한 번호가 주어진다.

[0042] 사용자는 키보드(162), 마이크로폰(163), 및 마우스 볼, 트랙볼 또는 터치 패드와 같은 포인팅 장치(161)와 같은 입력 장치를 통해 컴퓨터(110)에 명령과 정보를 입력할 수 있다. 다른 입력 장치(미도시)는 조이스틱, 게임 패드, 위성 접시, 스캐너 등을 포함할 수 있다. 이들 및 다른 입력 장치는 종종 시스템 버스에 결합되지만 다른 인터페이스와 병렬 포트, 게임 포트 또는 범용 직렬 버스(USB)와 같은 버스 구조에 접속될 수 있는 사용자 입력 인터페이스(160)를 통해 처리부(120)에 접속된다. 모니터(191) 또는 다른 유형의 디스플레이 장치가 또한 비디오 인터페이스(190)와 같은 인터페이스를 통해 시스템 버스(121)에 접속된다. 모니터에 더하여, 컴퓨터는 또한 출력 주변 인터페이스(190)를 통해 접속될 수 있는 스피커(197)와 프린터(196)와 같은 다른 주변 출력 장치를 포함할 수 있다.

[0043] 컴퓨터(110)는 원격 컴퓨터(180)와 같은 하나 이상의 원격 컴퓨터(180)로의 논리적 접속을 사용하여 네트워크 환경에서 동작할 수 있다. 원격 컴퓨터(180)는 핸드헬드 장치, 서버, 라우터, 네트워크 PC, 피어 장치 또는 다른 공통 네트워크 노드일 수 있으며, 통상 컴퓨터(110)에 대하여 상술한 모든 또는 다수의 요소를 포함한다. 도 1에 도시한 논리적 접속은 근거리 네트워크(LAN; 171)와 광역 네트워크(WAN; 173)를 포함하지만, 다른 네트워크를 포함할 수 있다. 이러한 네트워킹 환경은 사무실, 범사내망, 인트라넷과 인터넷에 흔한 것이다.

[0044] LAN 네트워킹 환경에서 사용되는 경우, 컴퓨터(110)는 네트워크 인터페이스 또는 어댑터(170)를 통해 LAN(171)에 접속된다. WAN 네트워킹 인터페이스에 사용되는 경우, 컴퓨터(110)는 통상 인터넷과 같은 WAN(173)을 통해 통신을 설정하는 모뎀(172) 또는 다른 수단을 포함한다. 모뎀(172)은 내장형 또는 외장형일 수 있으며 사용자 입력 인터페이스(160) 또는 다른 적절한 메커니즘을 통해 시스템 버스(121)에 접속될 수 있다. 네트워크 환경에서, 컴퓨터(110) 또는 그 일부에 대하여 나타낸 프로그램 모듈은 원격 메모리 저장 장치에 저장될 수 있다. 예를 들면, - 한정이 아님 -, 도 1은 원격 컴퓨터(180)에 상주하는 것으로서 원격 애플리케이션 프로그램(185)을 나타낸다. 도시한 네트워크 접속은 예시적이며 컴퓨터 간의 통신 링크를 설정하는 다른 수단이 사용될 수 있음이 이해될 것이다.

[0045] 본 발명은 도 1에 대하여 설명한 바와 같은 컴퓨터 시스템 상에서 실행될 수 있음이 이해되어야 한다. 그러나, 본 발명은 서버, 메시지 핸들링 전용 컴퓨터 상에서, 또는 본 발명의 상이한 부위가 분산 컴퓨팅 시스템의 상이한 부분에 실행될 수 있는 분산 시스템 상에서 실행될 수도 있다.

[0046] 문법 생성 시스템

[0047] 상술한 바와 같이, 본 발명의 일 양태는 문법 작성 경험이 거의 없는 개발자가 부품 번호, 자동차 면허 등과 같은 영숫자 개념 또는 표현에 대한 고성능 음성 문법을 구성할 수 있게 하는 시스템 및 방법을 포함한다. 이들 표현 유형은 많은 애플리케이션 유형에 존재하지만, 이 표현들은 통상 애플리케이션에 고유하고, 이에 따라 많은 애플리케이션 유형에 걸쳐 사용되는 라이브러리 문법에 미리 구성될 가능성이 적다.

[0048] 도 2는 예를 들면 상술한 운영 환경의 형태 중 임의의 것에서 동작가능한 문법 생성 모듈(200)을 나타낸다. 통상, 문법 생성 모듈은 인식될 영숫자 개념 또는 표현의 유형을 나타내는 표시를 수신한다. 이 표시는 통상 인식될 영숫자 표현이 포함할 수 있는 문자 및 서로에 대한 위치를 규정한다. 달리 말하면, 이 표시는 영숫자 표현이 인식될 한계를 규정한다. 이 표시는, 아래 일 예로서 사용될 수 있으며, www.w3.org/TR/xmlschema-2/에서 입수가능한 "XML Schema Part 2: Datatypes W3C Recommendation 02 May 2001"에서 W3C에 의해 한정된 바와 같은 "정규 표현"을 포함하지만 이에 한정되지 않는 정보를 나타내도록 여러 형태를 취할 수 있다. 다른 형태에서, 이 표시는 "마스크"의 형태를 취하여 영숫자 표현이 인식될 패턴을 사용자가 구체적으로 한정할 수 있게 한다.

[0049] 문법 생성 모듈(200)은 이 표시를 수신하고 처리하여 원하는 영숫자 표현을 인식하기에 적합한 문법(202)을

생성한다. 문법 생성 모듈(200)은 하나 이상의 표시 형태를 처리하기에 적합할 수 있다. 그러나, 인식될 공지된 형태의 영숫자 표현이 취할 수 있는 관점에서, 하나 이상의 표시 형태를 허용하는 선택적 컨버터(204)를 사용하고 문법 생성 모듈(200)에 의해 수신하도록 설계된 단일 형태로 모든 표시 형태를 전환시킬 수 있다.

[0050] 예를 들면, 문법 생성 모듈(200)은 W3C에 의해 한정된 바와 같은 정규 표현을 처리하는 것으로서 나타낼 수 있다. 이 표준의 간단한 리뷰가 도움이 될 수 있다. W3C 표준은 정규 표현에 대하여 다음 형태 정의를 갖는다:

[0051] regExp ::= branch('|' branch)*

[0052] branch ::= pieces*

[0053] piece ::= atom quantifier?

[0054] atom ::= char|charClass|('('regExp')')

[0055] 이 정의에 따르면, 정규 표현은 대안이 "|"으로 구분되는 하나 또는 다수의 대안들(브랜치)로 이루어진다. 각 브랜치는 피스(piece)의 시퀀스로 이루어진다. 각 피스는 선택적으로 양자화되는 원소(atom)이다. 이 수량사는 원소의 반복을 규정한다. 이는 숫자(예를 들면, {3}), 수치 범위(예를 들면, {0-3}), 또는 예약 문자(예를 들면, 1번 이상에는 '+', 0번 이상에는 '*')일 수 있다. 원소는 문자, 문자 클래스(예를 들면, 모든 대문자에 대한 [A-Z], 또는 10진수 [0-9]에 대한 \d), 또는 반복적으로 괄호친 정규 표현일 수 있다.

[0056] 반복 정규 표현 원소를 갖는 정규 표현은 반복 원소 없이 하나에 전환될 수 있음이 주목되어야 한다. 예를 들면, "(\d{3} | [A-Z]){2}C"는 "\d{3}[A-Z]C | \d{6}C | [A-Z]\d{3}C | [A-Z]{2}}C"와 동일하게 정의한다. 후술하는 알고리즘은 반복 정규 표현 원소없이 정규 표현을 처리하며, 이에 따라 반복 정규 표현이 표시에 존재하면, 적절한 전환이 필요할 수 있다.

[0057] 또한, 그러나 전환 알고리즘에서는 사용되지 않지만 본 설명의 목적으로, "브랜치"와 "피스"는 또한 마스크 형태로 표시 처리에 적용될 수 있다. 예를 들면, 부품 번호에 대한 마스크는 "&##-###-&&&"의 형태이며, 여기서 "&"는 {A-Z} 집합 중 임의의 문자를 나타내고 "#"는 아라비아 숫자 {0-9} 중 임의의 것을 나타내므로, "피스"는 "&##", "###" 및 "&&&"를 포함하는 반면, "원소"는 임의의 "&", "#" 또는 "-"를 포함한다.

[0058] 전환 알고리즘

[0059] 다음 의사 코드는 표시(여기서는, 정규 표현, "regexp")를 처리하여 regexp에 의해 한정된 원하는 영숫자 표현의 인식에 적합한 문법 "gram"을 획득하는 일 실시예이다. 의사 코드에 포함된 모듈과 도 2의 대응 표현은 한정적인 것으로 간주되어서는 안 된다. 의사 코드와 도 2의 블록도는 모두 처리 개념을 일부 설명하기 위해 제공되며, 그 형태는 한정적인 것으로 간주되어서는 안 된다. 당업자가 이해하는 바와 같이, 최종 결과에 영향을 주지 않으면서 상이한 단계 또는 순서 변화를 사용하여 처리가 수행될 수 있다. 또한, 도 2의 모듈에 의해 수행된 처리는 다른 모듈로 분리되거나 및/또는 도시한 모듈 또는 본 발명의 양태에 벗어나지 않는 또다른 모듈에 결합될 수 있다.

표 1

```
Line
Num.
1. create_regexp_grammar(regexp, gram) {
2.   foreach branch in (regexp.branches()) {
3.     string symbol =
4.       create_branch_grammar(branch, gram);
5.     rule_token token(symbol,1,1);
6.     array RHS = (token);
```

[0060]

```
7.     gram.add_rule(gram.root(), RHS);
8.   }
9. }
10.   create_branch_grammar(branch, gram) {
11.     array RHS = ();
12.     foreach piece in (branch.pieces()) {
13.       string symbol =
14.         create_piece_grammar(piece, gram);
15.       rule_token token(symbol,1,1);
16.       RHS.add(token);
17.     }
18.     string LHS=new_symbol();
19.     gram.add_rule(LHS, RHS);
20.     return LHS;
21.   }
22.   create_piece_grammar(piece, gram) {
23.     atom unit = piece.atom();
24.     pair (min, max) = piece.quantity();
25.     set charset = unit.CharSet();
26.     if (charset == {0, …, 9}) {
27.       string LHS=new_symbol();
28.       for (int i=min; i<=max; i++) {
29.         string ref = ruleref(lib, digit_i);
30.         array RHS=(rule_token(ref, 1, 1));
31.         gram.add_rule(LHS, RHS);
32.       }
33.       return LHS;
34.     }
35.     else {
36.       string charset_nt =
```

[0061]

```
37.         create_charset_grammar(charset,gram);
38.       return
39.         create_repeats(charset_nt, min, max);
40.     }
41.   }
42.   create_repeats(symbol, min, max) {
43.     if (hash[symbol, min, max] != null)
44.       return hash(symbol, min, max);
45.     string LHS = new_symbol();
46.     hash[symbol, min, max] = LHS;
47.     array RHS=();
48.     if (min == 0) {
49.       rule_token token(symbol, 0, 0);
50.       RHS.add(token);
51.       gram.add_rule(LHS, RHS);
52.     }
53.     if (max <= 0) return LHS
54.     rule_token token(symbol, 1, 1);
55.     RHS = (token);
56.     if (min <= 1)
57.       gram.add_rule(LHS, RHS);
58.     if (max >= 2) {
59.       string rest1=
60.         create_repeats(symbol, min-1, max-1);
61.       rule_token rest1_token(rest1, 1, 1);
62.       RHS.add(rest1_token);
63.       gram.add_rule(LHS, RHS);
64.       RHS = (rule_token("double", 1, 1));
65.       RHS.add(token);
66.       string rest2=
```

[0062]

```
67.         create_repeats(symbol, min-2, max-2);
68.       rule_token rest2_token(rest2, 1, 1);
69.       RHS.add(rest2_token);
70.       gram.add_rule(LHS, RHS);
71.     }
72.     if (max >= 3) {
73.       RHS = (rule_token("triple", 1, 1));
74.       RHS.add(token);
75.       string rest3=
76.         create_repeats(symbol, min-3, max-3);
77.       rule_token rest3_token(rest3, 1, 1);
78.       RHS.add(rest2_token);
79.       gram.add_rule(LHS, RHS);
80.     }
81.     return LHS;
82.   }
83.   create_charset_grammar(charset,gram) {
84.     string LHS=new_symbol();
85.     array RHS=();
86.     foreach ch in (charset) {
87.       switch (ch) {
88.       case '0': RHS=(rule_token("zero",1,1));
89.                 gram.add_rule(LHS, RHS);
90.                 RHS=(rule_token("oh",1,1));
91.                 gram.add_rule(LHS, RHS);
92.                 break;
93.       case '1': ……
94.     }
95.     return LHS;
96.   }
```

[0063]

[0064] 상술한 의사 코드는 정규 표현 파싱 모듈(206)이 이들 컴포넌트를 액세스하는 메소드를 갖는다는 점이 주목되어야 한다. 예를 들면, 메소드 branches()는 정규 표현으로 브랜치 리스트를 리턴하지만, 메소드 pieces()는 정규 표현의 브랜치로 피스 리스트를 리턴한다. 또한, 이는 rule_token의 어레이로서 규칙의 우측을 나타낸다. 각각의 rule_token은 재기입 규칙에서 최대에서 최대 횟수까지 심벌이 반복함을 규정하는 튜플(symbol, min, max)이다.

[0065] 표시를 처리하는 메소드(300)을 나타내는 도 3을 또한 참조하면, 처리는 문법 생성 모듈(200)에 의한 표시를 수신하는 단계 302에서 개시한다. 이 표시는 그 후 파싱 모듈(206)에 의해 파싱되어 단계 304에서 표시의 서브그룹(즉, 브랜치)를 식별한다. 통상, 브랜치는 "|"와 같은 문자를 분리하여 결정된다.

[0066] 식별된 브랜치를 사용하여, 일 실시예에서, 각 브랜치는 문법(202)에 대한 규칙을 생성하도록 처리되고, 특히 각 브랜치의 각 피스가 처리된다. 이 단계는 도 3의 단계 306으로 나타내며, 브랜치 처리는 브랜치 규칙 생성기 모듈(208)에 의해 제공되고 피스 처리는 도 2의 피스 규칙 생성기 모듈(210)에 의해 제공된다. 보다 넓은 용어로 언급하면, 브랜치 규칙 생성기 모듈(208)은 표시의 식별된 브랜치에 대한 다른 규칙을 생성하지만, 피스 규칙 생성기(210)는 처음 언급한 브랜치 각각의 보다 작은 부분(피스)에 대한 규칙을 생성한다. 통상, 정규 표현 파싱 모듈은 구분자 "-", "/", 빈칸 등으로 구분한 피스를 식별한다.

[0067] 도 2에서, 처리는 브랜치 규칙 생성기 모듈(208)로부터 발생하고, 피스 규칙 생성기 모듈(210)은 규칙에 문법(202)을 추가하는 규칙 추가 모듈(212)에 제공된다.

[0068] 상기 전환 알고리즘을 참조하면, 제1(주) 평션 create_regexp_grammar(1 내지 9 줄)은 제2 평션 create_branch_grammar(10 내지 21 줄)를 호출하여 입력 정규 표현의 각 브랜치에 대한 규칙을 생성하고, 루트 심벌을 재기입하는 규칙을 브랜치에 대한 "심벌"(알고리즘에 의해 생성)에 추가한다. 제2 평션 create_branch_ grammar는 제3 평션 create_piece_grammar(22 내지 41 줄)를 호출하여 브랜치에서의 각 피스에 대한 규칙을 생성하고, 브랜치 심벌에 재기입하는 규칙을 피스 시퀀스에 추가한다(즉, 피스를 함께 접합한

다). 표시가 "-", 또는 피스 구분자로서 사용된 다른 문자를 포함한 경우, 이는 또한 피스로 간주되고, 특히, 피스가 함께 접합된 경우에는 선택적 피스로 간주된다. 따라서, "AXD-134"를 포함하는 부품 번호에 있어서, 사용자는 "A X D dash one two three" 또는 "A X D one two three"로 말할 수 있지만(즉, dash 없이 말함), 문법은 이 언급들을 등가로서 간주할 수 있다. 대시, 슬래시 등과 같은 문자는 문법 규칙에서 선택적인 것으로서 구체화됨을 인식하여야 한다.

[0069] 제3 평션 create_piece_grammar는 피스에 대한 규칙을 생성한다. 규칙은 평션 gram.add.rule(LHS, RHS)에 의해 추가되고, 여기서 LHS와 RHS는 규칙의 좌측과 우측을 의미한다.

[0070] 피스 처리는 영숫자 표현의 많은 형태에 존재할 수 있는 일부 유사성을 선택적으로 취할 수 있다. 예를 들면, 하나 이상의 아라비아 숫자의 시퀀스의 구술 변형의 인식은 공지되어 있다. 따라서, "AXD-134"를 포함하는 부품 번호에 대하여, 사용자는 "A X D one three four", "A X D one thirty four", "A X D one hundred and thirty four" 등을 말할 수 있다. "\d3"와 같이 정규 표현에서 피스로 표시한 수치열 또는 아라비아 숫자 집합 "134"는 그 후 피스 규칙 생성 모듈(210)에 의해 식별될 수 있으며, 여기서 피스 규칙 생성 모듈(210)은 문법 규칙의 저장된 라이브러리(214)를 평가하여 아라비아 숫자 집합의 유형을 인식하는데 사용되는 문법 규칙을 획득한다. 저장된 라이브러리(214)는 또한 예를 들면 "\d{1-3}"와 같은 범위로 한정된 선택적 길이의 아라비아 숫자 집합을 인식하는 문법 규칙을 포함할 수 있다. 저장된 라이브러리(214)는 아라비아 숫자 집합에 대한 문법에 한정되지 않지만, 가장 흔한 것일 수 있다. 전환 알고리즘에서, 26 내지 34 줄은 라이브러리로부터 규칙을 획득함으로써 아라비아 숫자 집합을 식별 및 처리한다.

[0071] 피스 처리가 라이브러리에 저장된 문법 규칙을 갖는 피스를 식별하지 않거나, 처리된 피스가 라이브러리(214)에 저장된 문법 규칙을 갖는 규칙이 아닌 경우에는, 이 피스의 특징은 그 후 피스에 대한 문법 규칙의 많은 집합을 제공하기 위해서 식별된다. 전환 알고리즘에서, 이러한 처리는 36 내지 41 줄에 제어된다.

[0072] 예를 들면 피스 정규 표현 "[A-C] {1-3}"을 사용하면, 피스 규칙 생성 모듈(210)은 피스 내에서 개별 문자를 식별하여 대응하는 규칙을 생성할 수 있다. 그러나, 또한, 피스 규칙 생성 모듈은 각 문자에 대하여 적절한 다른 구술 표현을 식별하여 각각의 다른 문자 구술 표현에 대한 문법(202) 내에 대응하는 규칙을 포함시킬 수 있다. 전환 알고리즘에서, 36 및 37 줄에서 호출되고 83 내지 96 줄에서 규정되는 평션 create_charset_grammar은 문자 집합의 모든 요소를 커버하는 문법 규칙을 생성하고, 여기서 각 문자에 대한 모든 문자 표현은 "case" 문장으로 한정되며, 예를 들면, "case '0'"이 제공된다.

[0073] 88 내지 92 줄에 나타낸 바와 같이, "0"의 발음에 대하여 "제로" 뿐만 아니라 "영"으로서 발음하는 규칙이 문법에 추가된다. 도시하지 않은 다른 예는 문자 "A"를 "a" 또는 "알파"로 발음하는 것에 대한 문법 규칙을 제공한다. 도 2에서, 다른 문자 구술 표현에 관련된 데이터는 216에서 나타내며, 라이브러리에 저장되거나 피스 규칙 생성 모듈(210)에 인코딩될 수 있다.

[0074] 상술한 바와 같은 다른 문자 구술 표현에 대한 규칙을 식별 및 생성하는 것에 더하여, 피스 규칙 생성 모듈(210)은 또한 다른 문자 시퀀스 구술 표현을 식별할 수 있다. 상기 예 "[A-C]{1-3}"를 사용하여, "A B", "A", "A B C"와 같은 표현 뿐만 아니라, 사용자는 "AA"에 대하여 "double A", 또는 "BBB"에 대하여 "triple B"를 제공할 수 있다. 전환 알고리즘에서, 42 내지 82 줄에 한정되고 39줄에서 필요시 반복적으로 초기 호출되는 평션 create_repeats는, 피스 표현에 따라, 0번 발생에 대한 프리픽스 문법 규칙을 생성한다(48 내지 52 줄); 1번 발생 (54 내지 57 줄); 1번 초과 발생 (58 내지 80 줄) --- 64 내지 70 줄에서 "double zero"와 같은 표현을 어떻게 모델링하는지에 주의하자; 및 2번 초과 발생 (72 내지 80 줄). 물론, "quadruple"와 같은 다른 문자 시퀀스 구술 표현의 기타 변형 또는 문자열에 대한 임의의 다른 구술 표현이 포함될 수 있다. 도 2에서, 다른 문자 시퀀스 구술 표현에 관련된 데이터는 218에서 나타내며, 라이브러리에 저장되거나 피스 규칙 생성 모듈(210)에 인코딩될 수 있다.

[0075] 일부 영숫자 표현에서, 동일한 몇몇 피스가 존재한다. 이 피스는 아라비아 숫자 집합과 같이 저장된 문법 라이브러리(214)에 발견된 바대로 문법 규칙 집합을 저장한 것에 대응한다. 그러나, 이 피스가 라이브러리(214) 내의 문법 규칙에 대응하는 경우에도, 피스는 표시 내에서 동일할 수 있다. 예를 들면, "[A-Z]{1-2}-\d{2}-[A-Z]{1-2}"의 표시에서, 피스 "[A-Z]{1-2}는 2번 존재한다. 따라서, 다른 실시예에서, 피스 규칙 생성기 모듈(210)은 동일한 피스를 식별하고, 피스에 대한 규칙의 생성을 다시 반복할 필요가 없기 위해서는 피스의 이전 처리에서 생성된 문법 규칙을 사용할 수 있다. 전환 알고리즘에서, 다른 메커니즘이 사용될 수도 있지만, 이 표시가 처리됨에 따라, 해시 테이블(43 내지 46 줄)은 각 피스 부분을 추적하는 메커니즘으로서 사용된다. 표시의 추가 처리 동안, 동일한 피스가 발견되어 이전에 생성된 규칙이 복사될 수 있다. 도 2

에서, 이 양태는 처리된 피스 부분에 대한 규칙을 저장하는 라이브러리(220)로 표시한다.

[0076] 상술한 바와 같이, 문법(202)의 인식 규칙의 생성은 "0"에 대하여 "제로" 또는 "영", "AA"에 대하여 "double A" 그리고 "23"에 대하여 "twenty three"와 같은 다른 구술 표현에 대한 문법 규칙의 생성을 포함한다. 다른 실시예에서, 음성 인식기로부터의 출력은 일정하여 음성 인식기로부터 출력을 수신하는 애플리케이션은 "AA"와 "double A"가 등가임을 분별할 필요가 없도록 정규화가 제공된다.

[0077] 제1 실시예에서, 정규화는 문법(202)에 대한 규칙의 인코딩 또는 기입 동안 제공될 수 있다. 특히, 피스 규칙 생성 모듈(210)은 다른 문자 구술 표현과 다른 문자 시퀀스 구술 표현을 식별하도록 인코딩되는 것이 바람직하기 때문에, 피스를 처리하여 규칙을 생성하는 경우, 문법(202)에서의 정규화 정보를 적절하게 제공할 수 있다. 예를 들면, 문법(202)이 XML 의미 해석 태그를 사용하는 W3C 음성 인식 문법 규칙(SRGS) 포맷으로 기입되는 경우, 문법 내 태그는 정규화를 제공한다. 예를 들면, W3C SRGS 포맷으로 기입된 "AA"의 인식에 대한 규칙은 다음 형태를 취할 수 있다:

[0078]
```
<rule id="S2">
```

[0079]
```
  <one-of>
```

[0080]
```
    <item>A A
```

[0081]
```
      <tag>$=AA</tag>
```

[0082]
```
    </item>
```

[0083]
```
    <item>double A
```

[0084]
```
      <tag>$=AA</tag>
```

[0085]
```
    </item>
```

[0086]
```
  <one-of>
```

[0087]
```
</rule>
```

[0088] 여기서, " <tag>$=AA</tag>"는 정규화된 출력을 나타낸다. 전환 알고리즘에서는 구체적으로 나타내지 않았지만, 이 알고리즘은 의미 해석 태그를 이들이 생성될 때 규칙 토큰에 부착할 수 있기 때문에, 문법에 기초하여 획득한 인식 출력이 적절하게 정규화된다. 정규화는 문자 시퀀스에 대하여 상술하였지만, 단일 문자뿐만 아니라 아라비아 숫자 시퀀스에 대한 정규화도 유사한 방식으로 제공될 수 있다.

[0089] 도 2에 나타낸 실시예에서, 아라비아 숫자 집합 또는 시퀀스에 대한 문법 규칙이 구현되어 라이브러리(214)에 저장된다. 따라서, 아라비아 숫자 집합 또는 다른 피스에 대한 정규화도 문법 규칙과 함께 저장될 수 있다.

[0090] 다른 실시예에서, 정규화 규칙은 문법(202)과 분리되어 저장될 수 있다. 예를 들면, 정규화는 문법(202)에 관련된 정규화 맵 데이터베이스(226)에 정규화 매핑(예를 들면, "A A" "AA"; "double A" "AA")를 저장하여 제공될 수 있다. 적절한 정규화 매핑은 피스 처리 동안 피스 규칙 생성기 모듈(210)에 의해 제공되며, 또는 적절한 매핑은 아라비아 숫자 집합 등과 같은 대응하는 저장된 문법 규칙에 대하여 라이브러리(214)로부터 획득될 수 있다. 이러한 형태의 정규화에서, 문법은 "double A"와 같이 사용자의 실제 언급을 나타내지만, 음성 인식기가 그 결과를 리턴하기 전에, 이는 정규화된 형태가 그 결과에 존재하는지를 확인하여 존재 시에는 그 결과를 정규화된 결과와 대체한다. 음성 인식기의 구현 또는 동작은 이들 각 기술마다 변할 수 있지만, 개발자는 영숫자 표현의 표시만을 제공하면 되고, 시스템은 개발자에 의해 제공된 표시에 의해 규정된 포맷으로 적절하게 정규화될 수 있는 다른 구술 표현을 담당한다.

[0091] 일 실시예에서, 문법(202)은 프리픽스 최적화로 생성된다. 이는 인식 동안 다른 가설을 최소화함으로써 음성 인식기와 효율적으로 동작하는 문법을 제공한다. 프리픽스 최적화가 없는 경우, 문법은 도 4a에도 나타낸 별도 규칙을 포함할 수 있다:

[0092] S->aB

[0093] S->aC

[0094] 그러나, 상술한 바와 같이, 음성 인식기가 "a"를 인식하는 경우, 두 개의 가설 "aB"과 "aC"를 고려하여야 한다.

[0095] 그와 달리, 프리픽스 최적화된 문법에서, 이 규칙은 도 4b에 나타내는 아래의 형태일 수 있다:

[0096] S->aD

[0097] D->B

[0098] D->C

[0099] 따라서, "a"의 인식 시에, 음성 인식기는 하나의 가설 "aD" 만을 고려할 필요가 있다.

[0100] 상기 전환 알고리즘에서, 문법을 형성하는 규칙은 탑 노드가 좌측 "LHS"를 포함하고 다른 노드는 우측 "RHS"를 형성하는 어레이로서 저장되는 프리픽스 트리(예를 들면, 도 4b의 그림 표시로 나타낸 형태)로서 저장된다. 이러한 방식으로, 도 4a에 나타낸 것과 같은 규칙이 문법에 추가되는 경우, 프리픽스 최적화는 이미 도 4b의 프리픽스 트리로 구현되어 있다. 전환 알고리즘에서, gram.add.rule() 펑션은 각 규칙에 문법을 추가 또는 부가하며, 이는 규칙 추가 모듈(212)로 도 2에 나타낸다. 전환 알고리즘에서와 같이, 규칙이 프리픽스로서 초기에 저장되는 경우, XML을 사용하는 W3C SRGS 포맷과 같은 임의의 문법 형태로의 적절한 전환은 또한 규칙 추가 모듈(212)에 의해 구현될 수 있다.

[0101] 요컨대, 본 발명의 양태는 높은 품질의 음성 인식 문법이 정규 표현 또는 마스크와 같은 적절한 표시로부터 영숫자 개념에 대하여 자동적으로 구성될 수 있게 한다. 자동 문법 생성은 개발자가 애플리케이션 특정 영숫자 개념에 대하여 효율적이고 정확하게 동작하는 문법을 생성하는 힘든 작업을 덜어준다. 추가 특징은 생성된 문법이 프리픽스 구조를 사용하여, 예를 들면, 적절한 의미 해석 태그를 할당함으로써 최적화될 수 있게 한다. 이러한 방식으로, 여기서 설명하는 방법 및 시스템은 음성 인식 문법 작성 경험이 거의 없는 개발자에게 문법 개발을 상당히 가속시킨다.

[0102] 본 발명은 특정 실시예를 참조하여 설명하였지만, 당업자는 본 발명의 취지 및 범위를 벗어나지 않으면서 형태 및 세부사항에 대한 변형이 행해질 수 있음을 이해할 것이다.

발명의 효과

[0103] 상술한 바와 같이, 본 발명에 따르면, 영숫자 표현의 표시를 수신하는 단계를 포함함으로써, 개발자가 예를 들면 영숫자 표현에 대한 정규 표현을 제공할 수고 있고 시스템이 문법을 자동 구성할 수 있다.

도면의 간단한 설명

[0001] 도 1은 본 발명이 실시될 수 있는 통상의 컴퓨팅 환경을 나타내는 블록도.

[0002] 도 2는 영숫자 개념 또는 표현의 표시에 따라 문법을 생성하는 시스템을 개략적으로 나타내는 블록도.

[0003] 도 3은 문법을 생성하는 흐름도.

[0004] 도 4a는 프리픽스(prefix) 최적화되지 않은 문법의 일부를 나타내는 그림 표시.

[0005] 도 4b는 프리픽스 최적화된 문법의 일부를 나타내는 그림 표시.

[0006] <도면의 주요 부분에 대한 부호의 설명>

[0007] 200 : 문법 생성 모듈

[0008] 202 : 문법

[0009] 204 : 선택적 컨버터

[0010] 206 : 파싱 모듈

[0011] 208 : 브랜치 규칙 생성기 모듈

[0012] 210 : 피스 규칙 생성기 모듈

[0013] 212 : 규칙 추가 모듈

[0014] 214 : 저장된 문법 라이브러리

[0015] 216 : 다른 문자 구술 표현

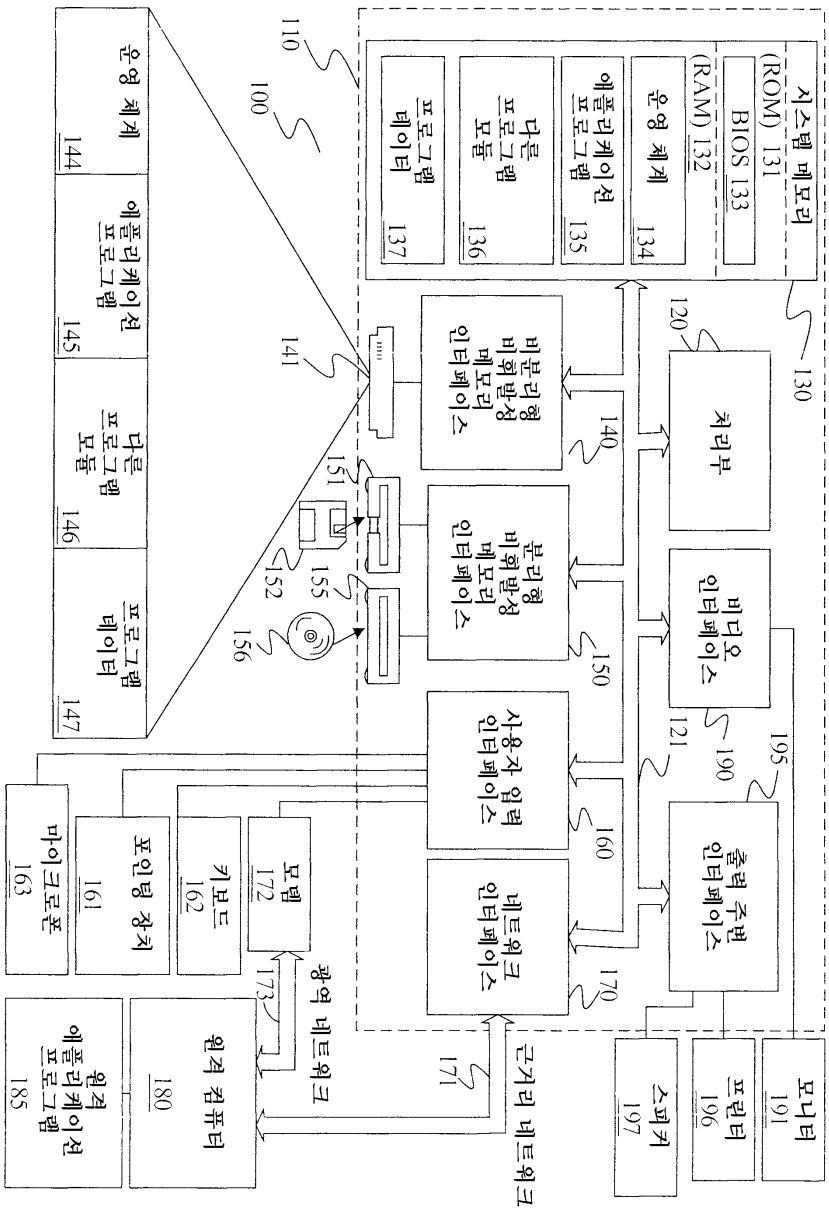
[0016] 218 : 다른 문자 시퀀스 구술 표현

[0017] 220 : 처리된 피스에 대하여 저장된 규칙

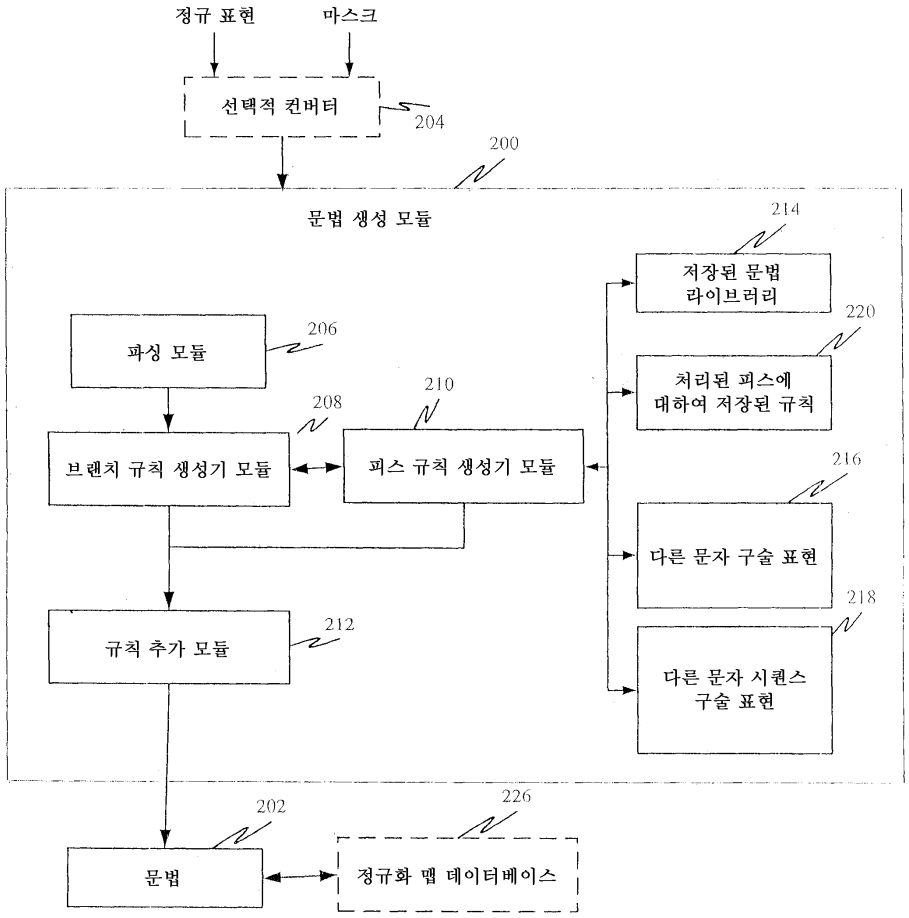
[0018] 226 : 정규화 맵 데이터베이스

도면

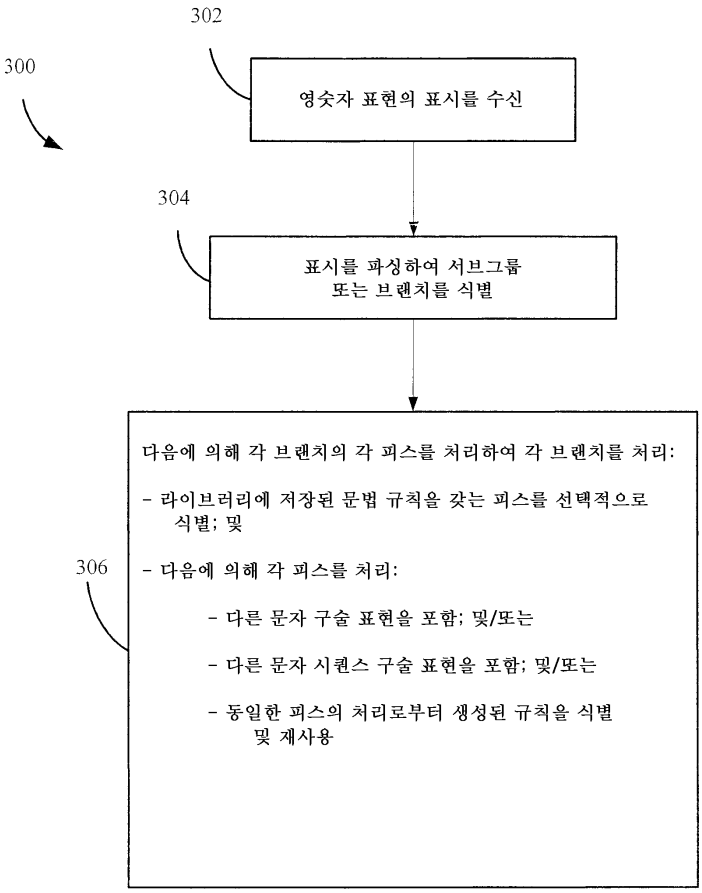
도면1



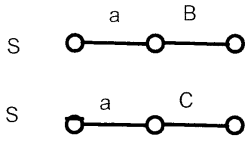
도면2


정규 표현
마스크
선택적 컨버터
204
200
문법 생성 모듈
214
저장된 문법 라이브러리
220
처리된 피스에 대하여 저장된 규칙
216
다른 문자 구술 표현
218
다른 문자 시퀀스 구술 표현
206
파싱 모듈
208
210
브랜치 규칙 생성기 모듈
피스 규칙 생성기 모듈
212
규칙 추가 모듈
202
문법
226
정규화 맵 데이터베이스

도면3



도면4a



도면4b