지식표현과 추론 – 1
규칙과 프레임

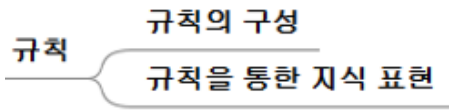
이건명

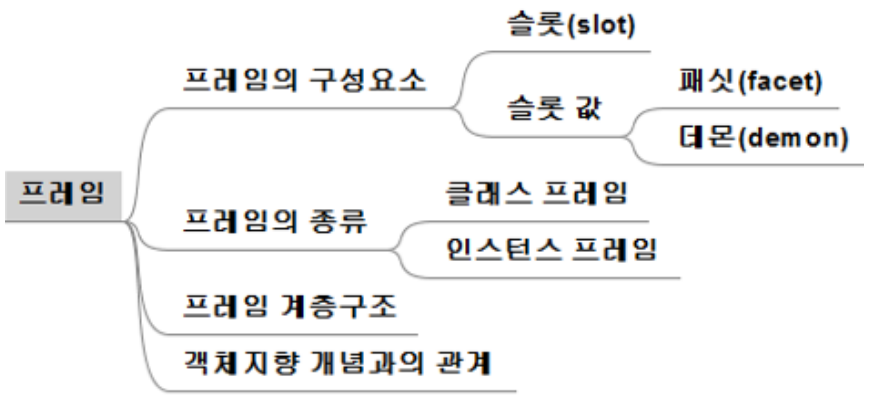
충북대학교 산업인공지능학과

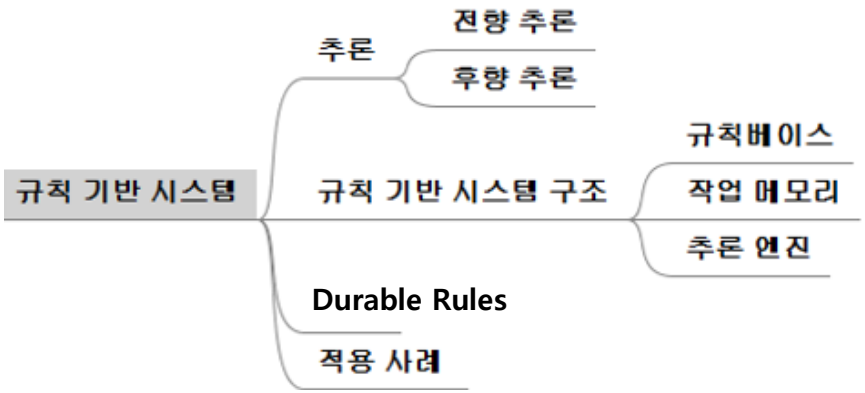
인공지능 : 튜링 테스트에서 딥러닝까지

학습 내용

- **규칙**과 **프레임**의 지식표현에 대해서 알아본다.
- **규칙 기반 시스템**의 구성과 동작 방법에 대해서 알아본다.







1. 지식 표현

❖ **지식 표현** (knowledge representgation)

- **인간**의 **지식**을 이용한 **인공지능 구현**을 위해 필요
- **프로그램**을 통한 구현을 위해 **정형적 표현**과 **처리(추론) 방법** 필요

❖ **데이터 피라미드**

- **데이터** (data)
  - 특정 분야에서 **관측된 아직 가공되는 않은 것**
  - 사실인 것처럼 관측되지만 **오류**나 **잡음**을 포함 가능
- **정보** (information)
  - 데이터를 **가공**하여 어떤 **목적**이나 **의미**를 갖도록 한 것
- **지식** (knowledge)
  - 정보를 **취합**하고 **분석**하여 얻은 대상에 대해 사람이 **이해**한 것
- **지혜** (wisdom)
  - 경험과 학습을 통해서 얻은 지식보다 높은 수준의 **통찰**

그림 3.1 데이터 피라미드

지식 표현

❖ **지식**(知識, knowledge)
- **경험**이나 **교육**을 통해 얻어진 **전문적인 이해**(understanding)와 체계화된 **문제 해결 능력**
- 어떤 주제나 분야에 대한 **이론적** 또는 **실제적인 이해**, 또는 현재 알려진 **사실**과 **정보**의 모음

- **암묵지**(暗黙知, tacit knowledge)
  - 형식을 갖추어 표현하기 어려운, 학습과 경험을 통해 쌓은 지식
- **형식지**(形式知, explicit knowledge)
  - 비교적 쉽게 형식을 갖추어 표현될 수 있는 지식

- **절차적 지식**(procedural knowledge)
  - 문제해결의 절차 기술
- **선언적 지식**(declarative knowledge)
  - 어떤 대상의 성질, 특성이나 관계 서술

❖ **인공지능에서의 지식 표현 및 처리**
- 프로그램이 쉽게 처리할 수 있도록 **정형화된 형태**로 표현
- 규칙, 프레임, 논리, 의미망, 스크립트, 수치적 함수 등

2. 규칙

❖ 규칙 (rule)

- **'~이면, ~이다'** 또는 **'~하면, ~하다'**와 같은 **조건부의 지식**을 표현하는 **IF-THEN 형태**의 문장
- 직관적
- 이해하기 쉬움

❖ 규칙 획득 및 표현

- **예:** 신호등이 녹색일 때는 건널목을 안전하게 건널 수 있고, 빨간색일 때는 길을 건너지 말아야 한다
- **대상**, **속성**, **행동** 또는 **판단**의 정보 추출
  - 대상 : 신호등
  - 속성 : 녹색, 빨간색
  - 행동/판단 : 건넌다, 멈춘다.
- **표현**
  - IF 신호등이 녹색이다 THEN 행동은 건넌다
  - IF 신호등이 빨간색이다 THEN 행동은 멈춘다

규칙

❖ 규칙 (rule)

- IF 신호등이 녹색이다 THEN 행동은 건넌다
- IF 신호등이 빨간색이다 THEN 행동은 멈춘다

- IF trafficLight = green THEN action = cross
- IF trafficLight= red THEN action = stop

- IF 부분
  - 주어진 정보나 사실에 대응될 조건
  - 조건부(conditional part, antecedent)

- THEN 부분
  - 조건부가 만족될 때의 판단이나 행동
  - 결론부(conclusion, consequent)

규칙

❖ 규칙의 구성

- 조건부
  - 둘 이상의 조건을 AND 또는 OR로 결합하여 구성 가능
    - IF <조건1> AND <조건2> AND <조건3> THEN <결론>
    - IF <조건1> OR <조건2> OR <조건3> THEN <결론>

- 결론부
  - 여러 개의 판단 또는 행동 포함 가능
    - IF <조건>
      THEN <결론1>
      AND <결론2>
      AND <결론3>

규칙

❖ **규칙을 통한 지식 표현**

- **인과관계**
  - **원인**을 조건부에 **결과**는 결론부에 표현
    - IF 연료통이 빈다
      THEN 차가 멈춘다

- **추천**
  - **상황**을 조건부에 기술하고 이에 따른 **추천 내용**을 결론부에 표현
    - IF 여름철이다 AND 날이 흐리다
      THEN 우산을 가지고 가라

- **지시**
  - **상황**을 조건부에 기술하고 이에 따른 **지시 내용**을 결론부에 표현
    - IF 차가 멈추었다 AND 연료통이 비었다
      THEN 주유를 한다

규칙

❖ **규칙을 통한 지식 표현 – cont.**

- **전략** (strategy)
  - **일련의 규칙**들로 표현
  - 이전 단계의 판정 결과에 따라 다음 단계에 고려할 규칙이 결정
    - IF 차가 멈추었다
      THEN 연료통을 확인한다 AND 단계1을 끝낸다
    - IF 단계1이 끝났다 AND 연료통은 충분히 찼다
      THEN 배터리를 확인한다 AND 단계2를 끝낸다

- **휴리스틱** (heuristic)
  - **경험적인 지식**을 표현하는 것
  - 전문가적 견해는 최적을 항상 보장하는 것이 아니고 **일반적으로 바람직한 것**을 표현
    - IF 시료가 액체이다 AND 시료의 PH가 6미만이다 AND 냄새가 시큼하다
      THEN 시료는 아세트산이다

3. 프레임

❖ 프레임(frame)

- 민스키(M. Minsky, 1927~2016)가 제안한 지식표현 방법
- 특정 **객체** 또는 **개념**에 대한 **전형적인 지식**을 **슬롯(slot)의 집합**으로 표현하는 것

- 예. **컴퓨터**를 표현한 프레임

frame-name	Computer	
frame-type	Class	
CPU	default	Intel
	data-type	string
	require	Intel, AMD, ARM, SPARC
OS	default	Windows
	data-type	string
memory	data-type	integer
warranty	default	3years
HDD	default	1TB
price	data-type	integer
stock	default	in-stock

```
(frame
   (frame-name Computer)
   (frame-type class)
   (CPU (default Intel)
        (data-type string)
        (require (Intel AMD ARM SPARC)))
   (OS (default Windows)
       (data-type string)
   (memory (data-type integer))
   (warranty (default 3years)
   (HDD (default 1TB))
   (price (data-type integer))
   (stock (default in-stock)))
```

프레임

❖ 프레임의 구성요소

- **슬롯(slot)**
  - 객체의 **속성**(attribute)을 기술하는 것
  - **슬롯 이름**(slot name)과 **슬롯 값**(slot value)으로 구성
    - 슬롯 이름: 속성 이름
    - 슬롯 값: 속성의 값
  - **슬롯 값(slot value)**
    - 복수 개의 **패싯**(facet)과 **데몬**(demon)으로 구성

frame-name	Computer	
frame-type	Class	
CPU	default	Intel
	data-type	string
	require	Intel, AMD, ARM, SPARC
OS	default	Windows
	data-type	string
memory	data-type	integer
warranty	default	3years
HDD	default	1TB
price	data-type	integer
stock	default	in-stock

프레임

❖ 프레임의 구성요소 – Cont.

- **패싯 (facet)**
  - '측면' 또는 '양상'을 의미
  - 속성에 대한 부가적인 정보를 지정하기 위해 사용
  - **패싯 이름**과 **패싯 값**의 쌍으로 구성

  - **패싯 이름**
    - value : **속성값** (수, 문자열, 다른 프레임의 포인터 등)
    - data-type : 속성값의 **자료형**
    - default : 디폴트값(속성값이 주어지지 않을 때 사용되는 **초기값**)
    - require : 슬롯에 **들어갈 수 있는 값**이 만족해야 할 **제약조건**

CPU	default	Intel
	data-type	string
	require	Intel AMD ARM SPARC
memory	data-type	integer

프레임

❖ 프레임의 구성요소 – Cont.

- **데몬(demon)**
  - 지정된 조건을 만족할 때 실행할 절차적 지식(procedure) 기술
  - 슬롯 값으로 데몬 실행조건과 데몬 이름의 쌍

  - 데몬의 실행조건의 예
    - if_needed : 슬롯 값을 알아야 할 때(즉, 사용하려고 할 때)
    - if_added : 슬롯 값이 추가될 때
    - if_removed : 슬롯 값이 제거될 때
    - if_modified : 슬롯 값이 수정될 때

HDD	value	512GB
price	if-needed	look-up-the-list
stock	if-needed	ask-for-vendor

데몬 실행조건 데몬 이름

프레임

❖ 프레임의 종류
- 클래스(class) 프레임
  - 부류(class)에 대한 정보 표현
- 인스턴스(instance) 프레임
  - 특정 객체에 대한 정보 표현

❖ 프레임 계층구조(hierachy)
- 상위 프레임
  - 클래스를 나타내는 프레임
- 하위 프레임
  - 하위 클래스 프레임 또는 상위 클래스 프레임의 객체
  - 상위 프레임을 상속(inheritance) 받음

프레임

❖ 프레임 표현의 예 : 컴퓨터

- **클래스 프레임 Computer**

```
(frame
   (frame-name Computer)
   (frame-type class)
   (CPU (default Intel)
          (data-type string)
          (require (Intel AMD ARM SPARC)))
   (OS (default Windows)
        (data-type string)
   (memory (data-type integer))
   (warranty (default 3years)
   (HDD (default 1TB))
   (price (data-type integer))
   (stock (default in-stock)))
```

frame-name	Computer	
frame-type	Class	
CPU	default	Intel
	data-type	string
	require	Intel AMD ARM SPARC
OS	default	Windows
	data-type	string
memory	data-type	integer
warranty	default	3years
HDD	default	1TB
price	data-type	integer
stock	default	in-stock

프레임

❖ 프레임 표현의 예 : 컴퓨터

- 인스턴스 프레임 Ultra-Slim-Notebook

frame-name	Computer	
frame-type	Class	
CPU	default	Intel
	data-type	string
	require	Intel AMD ARM SPARC
OS	default	Windows
	data-type	string
memory	data-type	integer
warranty	default	3years
HDD	default	1TB
price	data-type	integer
stock	default	in-stock

```
(frame
   (frame-name Ultra-Slim-Notebook)
   (frame-type instance (class Computer))
   (CPU (value ARM))
   (OS (value Android))
   (memory (value 4G))
   (HDD (value 512GB))
   (price (if-needed look-up-the-list)
   (stock (if-needed ask-for-vendor)))
```

(warranty 3years) ?

frame-name	Ultra-Slim-Notebook	
frame-type	instance (class Computer)	
CPU	value	ARM
OS	value	Android
memory	value	4G
warranty	value	3years
HDD	value	512GB
price	if-needed	look-up-the-list
stock	if-needed	ask-for-vendor

데몬

프레임

❖ 프레임과 규칙을 결합한 지식 표현

- 프레임은 특정 개념이나 대상에 대한 속성들 표현
  - 관련된 속성들을 하나의 덩어리로 관리

- 규칙을 사용하여 조건적인 지식 표현
  - 데몬에 규칙 사용
  - 또는 규칙의 조건부나 결론부에서 프레임 사용

- 대부분의 규칙기반 시스템에서 객체(object) 개념 사용
  - 객체의 표현에 프레임 사용 가능

4. 규칙 기반 시스템

❖ **규칙 기반 시스템**(rule-based system)

- 지식을 규칙의 형태로 표현
- 주어진 문제 상황에 적용될 수 있는 규칙들을 사용하여 문제에 대한 해를 찾도록 하는 **지식 기반 시스템**(knowledge-based system)
- **전문가 시스템**(expert system)을 구현하는 전형적인 형태
  - 특정 문제영역에 대해서 전문가 수준의 해를 찾아주는 시스템

4.1 추론

❖ **추론**

- 구축된 **지식**과 주어진 **데이터**나 **정보**를 이용하여 **새로운 사실**을 **생성**하는 것

- **전향 추론**(forward chaining, **순방향 추론**)
  - 규칙의 **조건부**와 **만족**하는 **사실**이 있을 때 규칙의 **결론부**를 **실행**하거나 **처리**

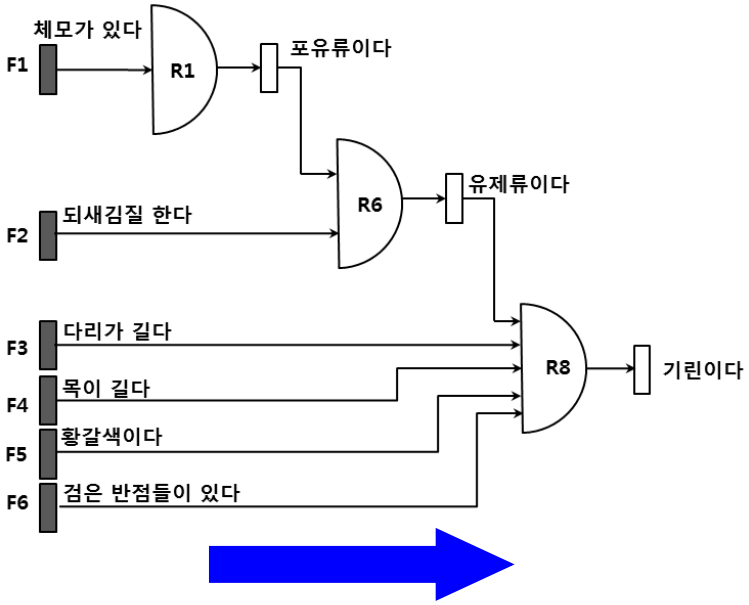
- **후향 추론**(backward chaining, **역방향 추론**)
  - 어떤 **사실**을 **검증**하거나 **확인**하고 싶은 경우에 관심 대상 사실을 **결론부**에 가지고 있는 규칙을 찾아서 **조건부의 조건들**이 모두 만족하는지 **확인**

전향 추론의 예

R1: IF ?x는 체모가 있다 THEN ?x는 **포유류**이다.
R2: IF ?x는 수유를 한다 THEN ?x는 **포유류**이다.
R3: IF ?x는 깃털이 있다 THEN ?x는 **조류**이다.
R4: IF ?x는 난다 AND ?x는 알을 낳는다 THEN ?x는 **조류**이다.
R5: IF ?x는 포유류이다 AND ?x는 고기를 먹는다 THEN ?x는 **육식동물**이다.
R6: IF ?x는 포유류이다 AND ?x는 되새김질한다 THEN ?x는 **유제류**이다.
R7: IF ?x는 육식동물이다 AND ?x는 황갈색이다 AND ?x는 검은 반점들이 있다 THEN ?x는 **치타**이다.
R8: IF ?x는 유제류이다 AND ?x는 다리가 길다 AND ?x는 목이 길다 AND ?x는 검은 반점들이 있다
THEN ?x는 **기린**이다.
R9: IF ?x는 포유류이다 AND ?x는 눈이 앞을 향해있다 AND ?x는 발톱이 있다 AND ?x는 이빨이 뾰족하다
THEN ?x는 **육식동물**이다.

F1: 래더는 체모가 있다.
F2: 래더는 되새김질을 한다.
F3: 래더는 다리가 길다.
F4: 래더는 목이 길다.
F5: 래더는 황갈색이다.
F6: 래더는 검은 반점들이 있다

래더는 뭘까?

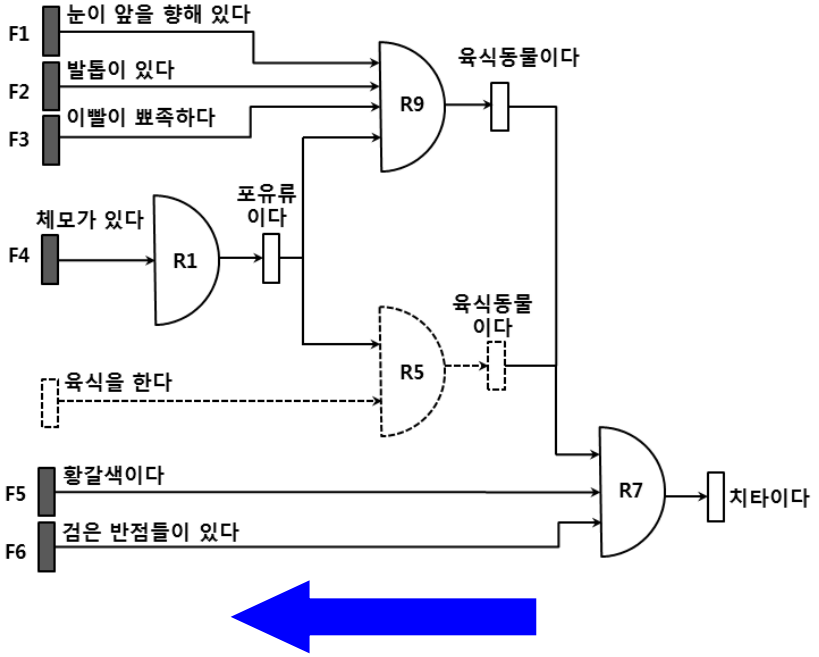


후향 추론의 예

R1: IF ?x는 체모가 있다 THEN ?x는 포유류이다.
R2: IF ?x는 수유를 한다 THEN ?x는 포유류이다.
R3: IF ?x는 깃털이 있다 THEN ?x는 조류이다.
R4: IF ?x는 난다 AND ?x는 알을 낳는다 THEN ?x는 조류이다.
R5: IF ?x는 포유류이다 AND ?x는 고기를 먹는다 THEN ?x는 **육식동물**이다.
R6: IF ?x는 포유류이다 AND ?x는 되새김질한다 THEN ?x는 유제류이다.
R7: IF ?x는 **육식동물**이다 AND ?x는 **황갈색**이다 AND ?x는 **검은 반점**들이 있다 THEN ?x는 **치타**이다.
R8: IF ?x는 유제류이다 AND ?x는 다리가 길다 AND ?x는 목이 길다 AND ?x는 검은 반점들이 있다
THEN ?x는 기린이다.
R9: IF ?x는 포유류이다 AND ?x는 눈이 앞을 향해있다 AND ?x는 발톱이 있다 AND ?x는 이빨이 뾰족하다
THEN ?x는 **육식동물**이다.

F1: 스프린터는 눈이 앞을 향해 있다.
F2: 스프린터는 발톱이 있다.
F3: 스프린터는 이빨이 뾰족하다.
F4: 스프린터는 체모가 있다.
F5: 스프린터는 황갈색이다.
F6: 스프린터는 검은 반점들이 있다.

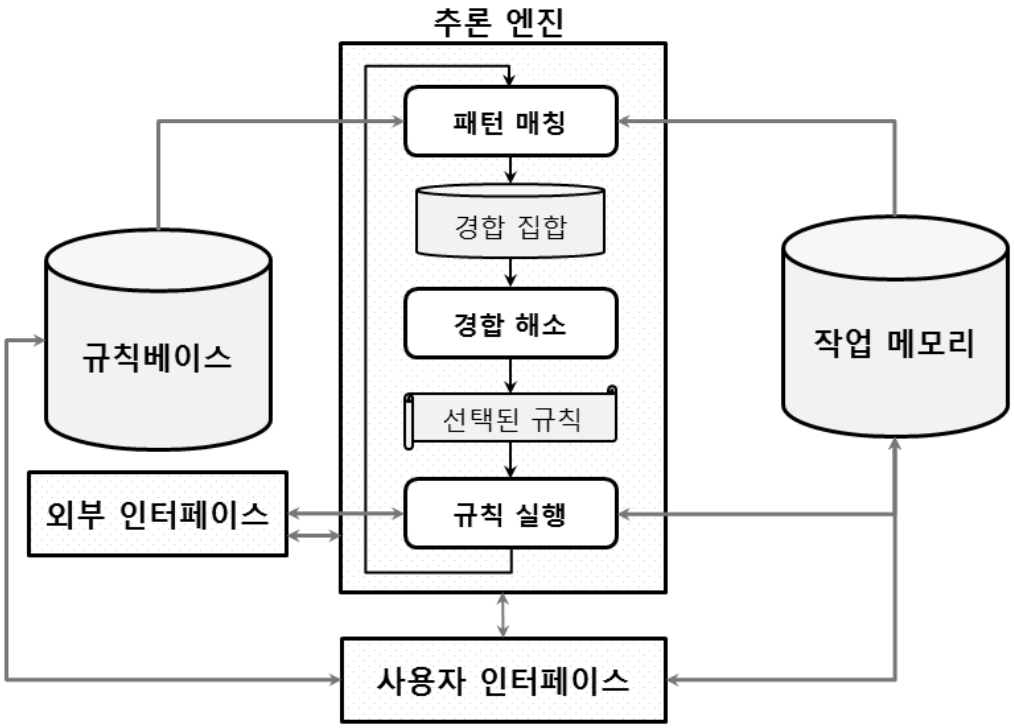
스프린터는 치타인가?



4.2 규칙 기반 시스템 구조

❖ 규칙 기반 시스템 구조

- **지식**
  - 규칙과 사실로 기술
  - **규칙**(rule) : 문제 해결을 위한 지식
  - **사실**(fact) : 문제 영역에 대해 알려진 데이터나 정보



규칙 기반 시스템 구조

❖ 규칙 기반 시스템 구조

- **규칙베이스**(rule base)
  - 전체 규칙의 집합을 관리하는 부분
  - **생성 메모리**(production memory)라고도 함

- **작업 메모리**(working memory)
  - 사용자로부터 받은 문제에 대한 **정보**를 관리
  - 추론과정의 **중간결과**를 저장하고, 유도된 **최종해** 저장
  - 작업 메모리에 저장되는 모든 것을 **사실**(fact)이라 함

규칙 기반 시스템 구조

❖ **추론 엔진(inference engine)**

- **실행할 수 있는 규칙**을 찾아서, 해당 규칙을 **실행**하는 역할
- **패턴 매칭 – 경합 해소 – 규칙 실행**의 과정 반복

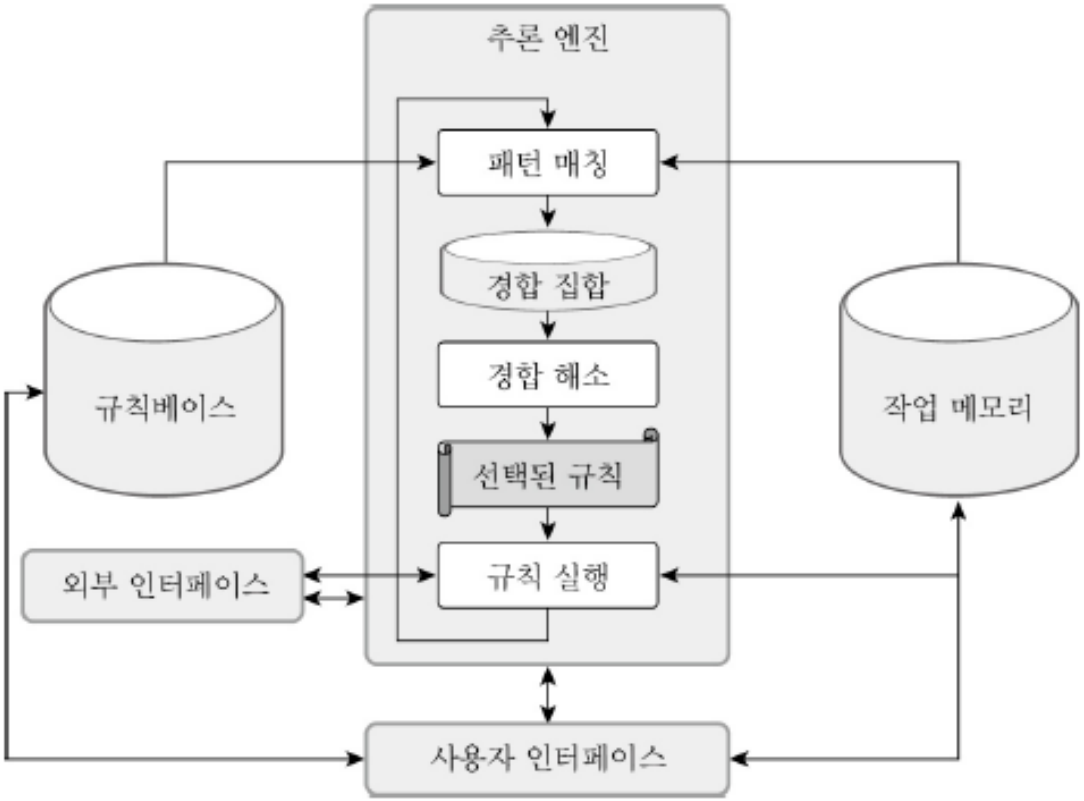
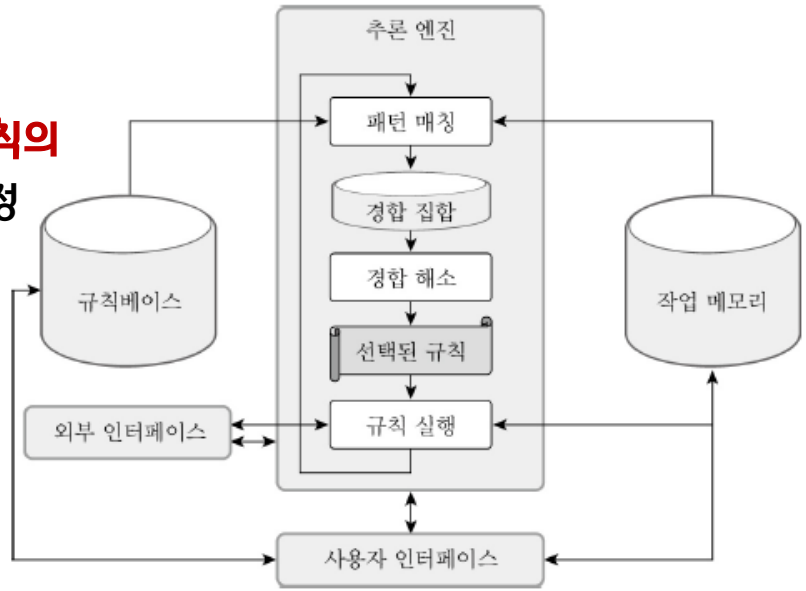


그림 3.26 규칙 기반 시스템의 구조

규칙 기반 시스템 구조

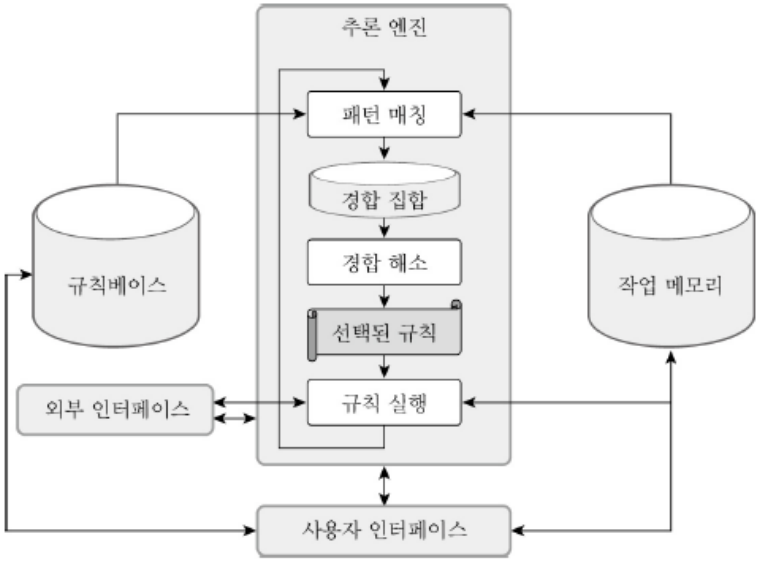
❖ 추론 엔진(inference engine)

- **패턴 매칭**(pattern matching)
  - 작업 메모리의 **사실**과 규칙베이스에 있는 **규칙의 조건부**를 대조하여 **일치**하는 규칙을 찾는 과정
- **경합 집합**(conflict set)
  - **실행 가능한 규칙**들의 집합
- **경합 해소**(conflict resolution)
  - 경합 집합에서 **하나**의 규칙을 **선택**
- **사용자 인터페이스**(user interface)
  - 규칙베이스 및 작업 메모리 관리, 추론 엔진 조작
- **외부 인터페이스**(external interface)
  - 외부 데이터나 함수의 기능 사용 지원



규칙 기반 시스템 구조

❖ 추론 엔진(inference engine)



- 경합 해소 전략
  - 규칙 우선순위(rule priority)
    - 미리 각 규칙에 우선순위 부여
    - 경합 집합에서 우선순위가 가장 높은 규칙 선택
  - 최신 우선(recency, depth)
    - 가장 최근에 입력된 데이터와 매칭된 규칙 선택
  - 최초 우선(first match, breath)
    - 경합 집합에서 가장 먼저 매칭된 규칙 선택
  - 상세 우선(specificity)
    - 가장 상세한 조건부를 갖는 규칙 선택
    - 규칙의 조건부가 가장 복잡하게 기술된 것 선택

규칙 기반 시스템

❖ 경합 해소 전략 – cont.

- **규칙 우선순위**(rule priority)
  - 규칙 1: 뇌막염 처방전 1 (우선순위 100)

    IF 감염이 뇌막염이다
    AND 환자가 어린이다
    THEN 처방전은 Number_1이다
    AND 추천 약은 암피실린(ampicillin)이다
    AND 추천 약은 겐타마이신(gentamicin)이다
    AND 뇌막염 처방전 1을 보여준다

  - 규칙 2: 뇌막염 처방전 2(우선순위 90)

    IF 감염이 뇌막염이다
    AND 환자가 어른이다
    THEN 처방전은 Number_2이다
    AND 추천 약은 페니실린(penicillin)이다
    AND 뇌막염 처방전 2를 보여준다

규칙 기반 시스템

❖ **경합 해소 전략 – cont.**

- **상세 우선**(specificity)
  - **가장 구체적인 조건의 규칙 선택**

  - **규칙 1**
    IF 가을이다
    AND 하늘이 흐리다
    AND 일기예보에서는 비가 온다고 한다
    THEN 조언은‘집에 머무르시오’

  - **규칙 2**
    IF 가을이다
    THEN 조언은‘우산을 가져가시오’

규칙 기반 시스템

❖ **경합 해소 전략 – cont.**

- **최신 우선**(recency, depth)

  - 가장 최근에 입력된 데이터(data most recently entered) 사용 규칙 선택
  - 각 **사실**에 **시간 태그** 부여

    – **규칙 1**

    IF 일기예보에서는 비가 온다고 한다 [03/25 08:16 PM]

    THEN 조언은'우산을 가져가시오'

    – **규칙 2**

    IF 비가 온다 [03/26 10:18 AM]

    THEN 조언은'집에 머무르시오'

규칙 기반 시스템

❖ 지식 표현

- 개발 도구에 따라 고유한 형식 사용
- 사실(fact)
  - 객체(object)나 프레임(frame)처럼 여러 개의 속성 포함 가능
  - 예. '이름이 멍키인 원숭이가 나이가 세 살이고 거실에 있다'
    (monkey (name 멍키) (age 3) (room 거실))

- 규칙
  - Jess의 규칙 표현 예

```
(defrule ruleBirthday
   ?c <- (monkey (name cheetah) (age ?old) (room ?where) (birthdate ?day))
         (calendar (date ?day))
      =>
         (bind ?newAge (+ ?old 1))
         (retract ?c)
         (assert (monkey cheetah ?newAge ?where)))
```

규칙 기반 시스템

❖ 규칙 기반 시스템 개발 도구

- 규칙 기반 시스템의 **기본 컴포넌트들**을 미리 **제공**하여
  규칙 기반 시스템을 쉽게 구현할 수 있게 하는 소프트웨어
- 문제 영역의 **지식**을 잘 **획득**하여 정해진 형태로 **표현**만 하면
  규칙 기반 시스템을 비교적 쉽게 구현 가능
- **Jess**, **Durable_Rules**, CLIPS, EXSYS, JEOPS 등

프로그래밍 실습: 규칙기반 시스템

❖ **Durable Rules 패키지**

- Redis DB 상에서 RETE 구조를 C언어로 구현한 규칙기반 시스템
- Python, Node.js, Ruby 등 지원

```
!pip install durable_rules
```

```
from durable.lang import *

with ruleset('testRS'):  # 규칙집합
    # antecedent(조건부). @when_all, @when_any를 사용하여 표기
    @when_all(m.subject == 'World')  # m: rule이 적용되는 데이터
    def say_hello(c):
        # consequent (결론부)
        print ('Hello {0}'.format(c.m.subject))

post('testRS', { 'subject': 'World' }) # 규칙 집합에 데이터 'subject': 'World' 전달
```

```
Hello World
{'$s': 1, 'id': 'sid-0', 'sid': '0'}
```

#https://github.com/jruizgit/rules/blob/master/docs/py/reference.md

```python
from durable.lang import *

with ruleset('animal'):
    @when_all(c.first << (m.predicate == 'eats') & (m.object == 'flies'),  # << 해당 조건을 만족하는 대상 지시하는 이름
              (m.predicate == 'lives') & (m.object == 'water') & (m.subject == c.first.subject))
    def frog(c):
        c.assert_fact({ 'subject': c.first.subject, 'predicate': 'is', 'object': 'frog' })
        # 사실(fact)의 추가

    @when_all(c.first << (m.predicate == 'eats') & (m.object == 'flies'),
              (m.predicate == 'lives') & (m.object == 'land') & (m.subject == c.first.subject))
    def chameleon(c):
        c.assert_fact({ 'subject': c.first.subject, 'predicate': 'is', 'object': 'chameleon' })

    @when_all((m.predicate == 'eats') & (m.object == 'worms'))
    def bird(c):
        c.assert_fact({ 'subject': c.m.subject, 'predicate': 'is', 'object': 'bird' })

    @when_all((m.predicate == 'is') & (m.object == 'frog'))
    def green(c):
        c.assert_fact({ 'subject': c.m.subject, 'predicate': 'is', 'object': 'green' })

    @when_all((m.predicate == 'is') & (m.object == 'chameleon'))
    def grey(c):
        c.assert_fact({ 'subject': c.m.subject, 'predicate': 'is', 'object': 'grey' })

    @when_all((m.predicate == 'is') & (m.object == 'bird'))
    def black(c):
        c.assert_fact({ 'subject': c.m.subject, 'predicate': 'is', 'object': 'black' })

    @when_all(+m.subject)  # m.subject가 한번 이상
    def output(c):
        print('Fact: {0} {1} {2}'.format(c.m.subject, c.m.predicate, c.m.object))

assert_fact('animal', { 'subject': 'Kermit', 'predicate': 'eats', 'object': 'flies' })
```

```
35
36 assert_fact('animal', { 'subject': 'Kermit', 'predicate': 'eats', 'object': 'flies' })
37 assert_fact('animal', { 'subject': 'Kermit', 'predicate': 'lives', 'object': 'water' })
38 assert_fact('animal', { 'subject': 'Greedy', 'predicate': 'eats', 'object': 'flies' })
39 assert_fact('animal', { 'subject': 'Greedy', 'predicate': 'lives', 'object': 'land' })
40 assert_fact('animal', { 'subject': 'Tweety', 'predicate': 'eats', 'object': 'worms' })
```

```
Fact: Kermit eats flies
Fact: Kermit is green
Fact: Kermit is frog
Fact: Kermit lives water
Fact: Greedy eats flies
Fact: Greedy is grey
Fact: Greedy is chameleon
Fact: Greedy lives land
Fact: Tweety is black
Fact: Tweety is bird
Fact: Tweety eats worms
{'$s': 1, 'id': 'sid-0', 'sid': '0'}
```

```
with ruleset('risk'):
    @when_all(c.first << m.t == 'purchase',
              c.second << m.location != c.first.location)
    def fraud(c):
        print('이상거래 탐지 -> {0}, {1}'.format(c.first.location, c.second.location))

post('risk', {'t': 'purchase', 'location': 'US'})
post('risk', {'t': 'purchase', 'location': 'CA'})
```

```
이상거래 탐지 -> CA, US
{'$s': 1, 'id': 'sid-0', 'sid': '0'}
```

```
with ruleset('bookstore'):
    @when_all(+m.status) # status를 갖는 것에 대해서 실행되는 규칙
    def event(c):
        print('bookstore-> Reference {0} status {1}'.format(c.m.reference, c.m.status))

    @when_all(+m.name)
    def fact(c):
        print('bookstore-> Added "{0}"'.format(c.m.name))

    @when_all(none(+m.name)) # name이 없는 것(삭제되는 것)에 호출
    def empty(c):
        print('bookstore-> No books')

# 새로운 fact 추가하는 경우
assert_fact('bookstore', {
    'name': 'The new book',
    'seller': 'bookstore',
    'reference': '75323',
    'price': 500
})

# 기존의 fact를 다시 추가하는 경우 MessageObservedError 발생
try:
    assert_fact('bookstore', {
        'reference': '75323',
        'name': 'The new book',
        'price': 500,
        'seller': 'bookstore'
    })
except BaseException as e:
    print('Error: {0}'.format(e.message))

```

```
post('bookstore', {
    'reference': '75323',
    'status': 'Active'
})

retract_fact('bookstore', {
    'reference': '75323',
    'name': 'The new book',
    'price': 500,
    'seller': 'bookstore'
})
```

```
bookstore-> Added "The new book"
Error: Message has already been observed: {"reference": "75323", "name": "The new book", "price": 500, "seller": "bookstore"}
bookstore-> Reference 75323 status Active
bookstore-> No books
{'$s': 1, 'id': 'sid-0', 'sid': '0'}
```

Quiz

❖ **규칙을 사용한 지식표현의 특징으로 옳지 않은 것은?**

① 규칙은 조건부와 결론부로 구성된다.

② 인과관계를 표현하는 규칙에서 결과는 조건부와 원인은 결론부에 둔다.

③ 규칙은 일반적으로 사람이 만들기 때문에 휴리스틱적인 요소가 많다.

④ 하나의 규칙이 실행된 결과에 의해서 다른 규칙이 실행될 수 있다.

❖ **지식과 관련한 다음 설명에서 가장 옳지 않는 것은**

① 데이터를 가공하여 어떤 목적이나 의미를 갖도록 한 것이 정보이다.

② 문제 해결 절차를 기술하는 지식을 절차적 지식이라 한다.

③ 어떤 대상의 성질, 특성이나 관계를 서술하는 지식을 선언적 지식이라 한다.

④ 학습과 경험을 통해 몸으로 배우는 말로 표현하기 어려운 지식을 형식지라고 한다.

❖ **규칙기반 시스템에서 규칙의 실행 순서가 중요한 이유는?**

① 규칙은 항상 일정한 순서대로 실행되어야 한다.

② 규칙의 실행 순서에 따라 결과가 달라질 수 있다.

③ 모든 규칙은 동시에 실행되어야 한다.

④ 규칙의 실행 순서는 시스템의 성능에 영향을 주지 않는다.

Quiz

❖ **규칙기반 시스템에서 "IF-THEN" 구문은 무엇을 의미하는가?**

① 만약 조건이 참이면 특정 행동을 실행한다.
② 특정 행동을 실행한 후 조건을 확인한다.
③ 조건과 행동 사이의 상관관계를 나타낸다.
④ 조건을 무시하고 항상 행동을 실행한다.

❖ **프레임에 대한 설명으로 올바르지 않은 것은?**

① 슬롯과 값의 쌍으로 구성된다.
② 상속 메커니즘이 포함될 수 있다.
③ 각 프레임은 고유한 이름을 가질 수 있다.
④ 프레임은 정적인 정보만을 포함한다.

❖ **추론 엔진의 역할에 대한 설명으로 올바르지 않은 것은?**

① 주어진 데이터에 대해 적절한 규칙을 적용한다.
② 결과를 사용자에게 제시한다.
③ 새로운 규칙을 생성하고 지식 베이스에 추가한다.
④ 규칙의 충돌 시, 적절한 규칙을 선택한다.

Quiz

❖ **프레임 지식표현은 주로 어떤 형태의 정보를 표현하는데 사용되는가?**

① 개별 사물의 특성
② 함수의 방정식
③ 로직 게이트
④ 배열의 원소

❖ **프레임의 슬롯은 무엇을 나타내는가?**

① 특정 정보의 값을 담는 공간
② 프레임 간의 관계
③ 프레임의 고유한 식별자
④ 프레임의 하위 클래스

❖ **프레임 지식표현에서 상속은 무슨 의미인가?**

① 데이터의 저장
② 하위 프레임이 상위 프레임의 특성을 받아오는 것
③ 프레임의 삭제
④ 프레임 간의 데이터 교환

Quiz

❖ **프레임에 대한 다음 설명에서 가장 옳지 않는 것은?**

① 클래스와 객체는 소프트웨어 개발에 있어서 모듈화, 재사용성 및 유지보수의 용이성을 고려한 프로그래밍 개념이다.
② 프레임에도 정보은닉 등 정보 접근에 대한 제한 메커니즘이 있다.
③ 객체 지향 프로그래밍에는 데몬 개념이 일반적이지 않다.
④ 패싯은 프레임에서 하나의 속성에 여러가지 부가적인 정보를 표현할 수 있도록 한다.

❖ **프레임 시스템에서 데몬 프로시저는 어떤 역할을 할 수 있는가?**

① 데이터 저장
② 프레임 삭제
③ 슬롯 값의 조건적 변경을 위한 동작을 정의
④ 슬롯 간의 관계 설정

❖ **규칙 기반 시스템에 대한 다음 설명에서 가장 옳지 않는 것은?**

① 전향 추론은 규칙의 조건부와 만족하는 사실이 있을 때 규칙의 결론부를 실행하거나 처리하도록 하는 것이다.
② 규칙을 표현할 때 변수를 포함할 수 있다.
③ Rete 알고리즘은 후향 추론을 하는 규칙 기반 시스템에서 매칭되는 규칙들을 신속하게 결정하기 위한 패턴 매칭 알고리즘이다.
④ 규칙 기반 시스템 중에는 전향 추론 또는 후향 추론을 지원하는 것도 있다.

Quiz

❖ **전향추론과 후향추론의 차이점 중 틀린 것은?**

① 전향추론은 알려진 사실에서 시작하여 결론을 도출하는 방식이다.
② 후향추론은 결론에서 시작하여 필요한 조건을 찾는 방식이다.
③ 전향추론은 결과를 예측하는 데 주로 사용되며, 후향추론은 주로 해석에 사용된다.
④ 후향추론은 주어진 사실을 기반으로 가능한 모든 결론을 도출하는 방식이다.

❖ **전향추론의 과정 중 올바르지 않은 것은?**

① 규칙의 선행 조건과 현재 상태를 비교한다.
② 일치하는 규칙이 있으면 해당 규칙의 결론부를 적용하여 새로운 사실을 도출한다.
③ 도출된 사실을 기반으로 다시 규칙을 적용한다.
④ 모든 규칙을 한 번씩 적용한 후, 프로세스를 종료한다.

❖ **후향추론의 특징 중 올바른 것은?**

① 항상 빠른 연산 속도를 보장한다.
② 필요한 규칙만을 적용하여 추론 과정을 최적화할 수 있다.
③ 추론의 시작점은 주어진 사실들이다.
④ 전향추론보다 더 많은 메모리를 요구한다.

Quiz

❖ 규칙기반 시스템의 규칙이 충돌하는 경우, 어떤 전략을 사용하여 처리할 수 있는가?

① 경합해소 전략 적용
② 모든 규칙 동시 적용
③ 규칙 무시
④ 사용자에게 선택 요청

❖ 규칙기반 시스템에서 추론엔진이 하는 주된 역할은 무엇인가?

① 규칙의 실행
② 데이터의 저장
③ 사용자 인터페이스 제공
④ 데이터 암호화