KIPS Tr. Software and Data Eng.
Vol.5, No.2 pp.59~68 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2016.5.2.59

A Light-Weight Rule Engine for Context-Aware Services

Seung-Kyu Yoo† · Sang-Young Cho††

ABSTRACT

Context-aware services recognize the context of situation environments of users and provide useful services according to the context for users. Usual rule-based systems can be used for context-aware services with the specified rules that express context information and operations. This paper proposes a light-weight rule engine that minimizes memory consumption for resource-constrained smart things. The rule engine manages rules at the minimum condition level, removes memories for intermediate rule matching results, and uses hash tables to store rules and context information efficiently. The implemented engine is verified using a rule set of a mouse training system and experiment results shows the engines consumes very little memory compared to the existing Rete algorithm with some sacrifice of execution time.

Keywords : Context-Aware Service, Rule-Based System, Rule Engine, Rete Algorithm

상황 인지 서비스를 위한 경량 규칙 엔진

유 승 규† · 조 상 영††

요 약

상황 인지 서비스는 서비스 대상의 주변 상황을 인지하여 상황에 맞는 유용한 서비스를 제공한다. 규칙 기반 시스템은 상황 정보를 IF 구문으로 표현하고 상황에 따른 동작을 THEN 구문으로 표현하는 규칙을 사용하여 상황 인지 서비스를 제공할 수 있다. 본 논문에서는 스마트 사물을 위하여 메모리 사용을 최적화한 경량 규칙 엔진을 제안한다. 제안된 엔진은 규칙을 기초 연산 단위로 관리하고 계산 값을 저장하는 메모리를 최소화하였으며 해시 표를 사용하여 규칙 및 상황 정보를 효율적으로 관리한다. 실제 쥐 훈련 시스템에서 사용하는 규칙 집합을 이용하여 제안된 엔진이 기존 Rete 알고리즘에 비하여 실행 속도는 다소 느리지만 매우 작은 메모리를 사용함을 확인하였다.

키워드 : 상황 인지 서비스, 규칙 기반 시스템, 규칙 엔진, Rete 알고리즘

1. 서 론

사물 인터넷에서의 스마트 사물은 관리하는 여러 센서들을 가지고 있으며, 특정 서비스를 위하여 여러 센서들의 데이터를 정제하여 유효한 데이터를 모으고, 이 데이터들을 이용하여 상황 정보를 추출하여 지식을 구축하거나 지식을 이용하여 외부와 지능적 상호작용 또는 통신을 할 수 있게 한다[1]. 이를 통하여 스마트 사물은 사물 인터넷 환경에서 발생하는 대규모의 센서 데이터를 상위 단계의 상황 정보로 요약하여 제공하며 상황 정보를 이용하여 상황 인지 서비스를 제공한다. 여기서 상황이란 사용자의 위치나 상태 등의 정보와 사용자가 위치한 환경에서 동작하거나 사용하는 장치들과 그 장치들의 상태 정보를 의미한다. 이 상황 정보는 사용자와 장치의 상태 변화에 따라 지속적으로 갱신된다. 예를 들어, 사용자가 이동하거나 특정한 행동을 할 때, 환경의 물리적 값의 변화에 따른 센서의 측정값이 변하거나 장치들의 출력이 변할 때에 상황 정보는 변경된다. 상황 인지 서비스가 가능한 애플리케이션은 사용자의 행동 또는 사용자가 위치한 환경에서 상황 데이터를 획득하고 이를 처리하여 상황을 인지하며 인지된 상황에 가장 알맞은 적절한 서비스를 제공한다[2].

상황 인지 서비스는 지식을 모델링하고 상황 정보로부터 상황을 인지하기 위한 추론 과정이 필요하기 때문에 지도 학습, 비지도 학습, 규칙, 애매 논리, 온톨로지, 확률적 추론 등과 같이 다양한 추론 기법이 사용되고 있다. 이 기법들 중에서 표현 및 적용의 간결함으로 인해 규칙 기반 추론 기법이 여러 응용 분야에서 가장 많이 사용되고 있다[3]. 규칙

※ 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업(No. 2015R1A5A7036384)과 2015년 한국외국어대학교 교내연구비 지원에 의하여 연구되었음.
† 정 회 원 : 한국외국어대학교 컴퓨터정보통신공학과 석사
†† 종신회원 : 한국외국어대학교 컴퓨터전자시스템공학부 교수
Manuscript Received : September 7, 2015
First Revision : October 22, 2015
Accepted : October 24, 2015
* Corresponding Author : Sang-Young Cho(sycho@hufs.ac.kr)

기반 시스템(rule-based system)은 IF-THEN 형태의 규칙을 사용하여 IF 조건이 만족할 때 THEN 문장의 동작 수행을 명시하며 이러한 규칙들의 집합을 추론 엔진(inference Engine)으로 지속적으로 평가하여 조건에 따른 동작을 수행하도록 한다[4]. 상황 인지 서비스의 경우에는 상황 변화에 따른 서비스 제공 장치의 동작을 규칙으로 작성하여 입력되는 상황 정보를 바탕으로 IF 조건을 만족하는 규칙들을 찾아 장치를 동작시키거나 서비스 정보를 제공한다.

Rete 알고리즘은 규칙 기반 시스템의 추론 엔진을 위한 가장 일반적인 알고리즘이다[5]. Rete 알고리즘은 다양한 응용 분야에서 사용 가능한 범용적인 알고리즘이지만 처리 속도를 빠르게 하기 위하여 메모리를 많이 사용하기 때문에 자원이 제한되어 있는 시스템에서 동작하기에 어려움이 있다. 이러한 단점을 극복하기 위한 알고리즘 연구와 상황 인지 서비스에 적용하기 위한 연구가 다양하게 진행되어 왔다[6-10].

본 논문은 계산 능력과 메모리 용량이 적은 시스템에서 상황 인지 서비스를 제공하기 위한 규칙 기반 시스템의 경량 규칙 엔진(LwRE: Light-weight Rule Engine)의 설계와 구현에 대하여 기술한다. LwRE는 기존의 규칙 엔진에 비하여 메모리의 사용량을 최대한 적게 사용할 수 있도록 설계하였으며 단순한 동작을 하는 장치들로 이루어진 환경에서 상황 인지 서비스를 제공할 수 있도록 ECA(Event-Condition-Action) 모델을 채용하였다. ECA 모델은 관심있는 장치의 사건(Event)이 발생했을 때 현재의 상태 조건(Condition)을 확인하고 모든 조건이 충족되었을 때 동작(Action)을 수행한다. 이러한 동작 모델은 시스템이 관심있는 상황에서만 동작할 수 있도록 하여 전체 시스템의 불필요한 수행을 줄이고 효율적으로 동작하게 한다[11-12].

본 논문에서 제안하는 LwRE의 평가는 쥐가 생활하는 환경에서 쥐의 행동에 따라 환경에 속해 있는 여러 장치들을 제어하는 쥐 훈련 시스템(MTS: Mouse Training system)을 이용하였으며 기존의 Rete 알고리즘을 이용한 규칙 엔진에 비하여 실행속도는 조금 손해를 보지만 훨씬 적은 메모리가 사용됨을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 Rete 알고리즘과 기존 관련 연구에 대하여 설명한다. 3장에서는 LwRE를 위한 규칙 표현 방식과 LwRE의 추론 방법을 포함한 설계에 대해 설명한다. 4장에서는 구현된 시스템의 검증과 실험 환경을 설명하며 실험 결과를 기술한다. 마지막으로 5장에서는 본 논문의 결론과 활용에 대해 기술한다.

2. 관련 연구

2.1 Rete 알고리즘

규칙 기반 시스템은 지식을 IF-THEN 형태의 규칙의 집합으로 표현한다. IF 구문(LHS: Left Hand Side)에는 규칙 기반 시스템이 수행할 동작의 조건(condition) 혹은 상황 정보를 서술하고 THEN 구문(RHS: Right Hand Side)에는 IF 구문이 조건을 만족할 경우에 수행해야 할 동작(action)을 서술한다. IF 구문은 만족해야 할 다수의 조건을 가질 수 있으며 THEN 구문의 동작도 시스템 내의 다른 변수를 변화시키거나 시스템에 필요한 특정 동작을 의미하며 복수로 기술될 수 있다. 규칙 기반 시스템은 시스템 상태를 보관하고 있으며 새로운 상태의 입력 또는 시스템의 상태의 변화에 따라 규칙들이 검사되며 조건의 부합되는 규칙을 찾아 해당하는 동작을 수행한다. 규칙 기반 시스템의 수행을 위해서 보통 규칙을 저장하는 규칙 메모리(rule memory), 시스템 상태를 저장하는 작업 메모리(working memory), 그리고 두 메모리를 기반으로 시스템 상태에 따른 적합한 규칙을 추론하는 추론 엔진으로 구성된다[4].

시스템 상태가 변했을 경우에 상태에 상응하는 규칙을 찾기 위해서, 다수의 조건을 갖는 모든 규칙들의 LHS를 작업 메모리의 모든 상태 값들을 가지고 검사하는 작업은 많은 수행 시간을 필요로 한다. Rete 알고리즘은 규칙에 대한 모든 정보와 이전 수행 결과를 규칙 네트워크라는 데이터 구조에 저장함으로써 규칙 매칭 작업을 변화된 부분에서만 수행하기 때문에 기존 알고리즘에 비해 매우 빠른 수행 속도를 보인다. Rete 알고리즘에서는 규칙 메모리는 규칙을 정적으로 보관하거나 초기에 규칙 네트워크를 구축하기 위하여 사용되며 실제 동작 시에는 시스템의 현재 상태 정보를 저장하는 작업 메모리와 규칙 네트워크만 사용하여 동작한다.

상황 정보를 나타내는 다양한 변수는 시스템 내에서 작업 메모리 요소(WME: Working Memory Element)로 다루어진다. WME는 시스템을 구성하는 객체의 사실(fact)들로 구성되며 사실은 객체의 속성(attribute)과 그 값(value)의 쌍으로 구성된다. Fig. 1은 작업 메모리 내의 두 개의 WME인 Mouse 타입인 micky 객체와 Food 타입인 cheese 객체를 보여준다. micky는 이전 위치를 나타내는 preLoc 속성과 값 P를 가지고 사실 하나를 표현하고 있으며 현재 위치를 나타내는 curLoc은 S 값을 가지고 있다. cheese는 S 위치에 있으며 Small 크기를 가지고 있다.

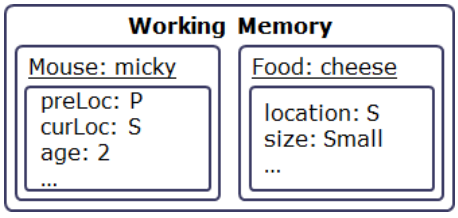

Fig. 1. Working memory element example

Rete 알고리즘의 규칙 네트워크는 비순환 그래프로 표현되며 알파 네트워크와 베타 네트워크로 구성된다[8]. Fig. 2는 쥐의 이전 위치가 P였고 현재의 위치에 음식이 놓여 있으면 파란 등을 켜는 규칙 예와 이에 상응하는 규칙 네트워크를 보여준다.

전체 네트워크는 루트 노드로 출발하며 단일 객체의 사실을 검사하는 알파 네트워크를 구성하고 두 객체 사이의 연

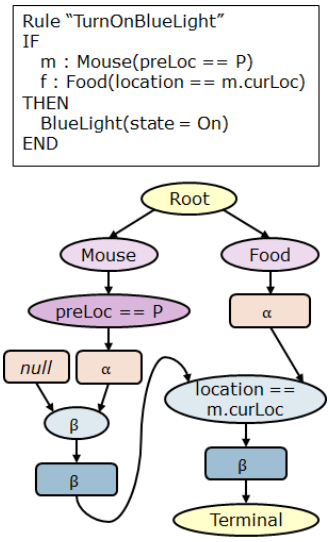

Fig. 2. A rule and the corresponding rule network

관 사실을 검증하는 베타 네트워크를 통하여 최종 규칙을 나타내는 터미널 노드로 끝난다. 알파 네트워크는 타입(type) 노드와 알파 노드로 구성되며 타입 노드는 자신과 일치하는 타입의 WME만을 검사할 수 있게 하여 불필요한 검사를 제한할 수 있다. 알파 노드는 한 가지 사실을 검사하는 알파 검사를 수행하며 규칙에 따라 여러 알파 노드가 직렬로 연결되어 여러 사실을 모두 만족하는지를 검사할 수 있다. 알파 검사를 통과한 WME는 알파 메모리에 저장된다. 규칙 예의 LHS에서 Mouse의 이전 위치가 P인지를 검사하는 부분과 같이 자체 속성에 대한 조건이 알파 노드가 되며 Food에 대해서는 자체 속성 비교가 없기 때문에 모든 Food 타입의 객체를 판단 없이 알파 메모리에 저장하게 된다.

베타 네트워크는 두 개의 입력을 받는 베타 노드와 베타 검사 결과를 저장하는 베타 메모리로 구성된다. 베타 노드는 두 객체 간의 연관 규칙을 처리하기 위하여 두 개의 입력을 갖는다. Fig. 2에서 Mouse는 다른 객체와의 연관 규칙이 없기 때문에 알파 검사를 통과한 WME는 null 입력을 갖는 베타 노드를 통해 베타 메모리에 저장된다. Food의 경우에는 자신의 위치와 Mouse의 위치의 일치를 검사하기 위하여 자신의 알파 메모리와 Mouse의 베타 메모리를 입력으로 하는 베타 노드가 있으며 베타 검사를 통과하면 WME가 베타 메모리에 저장이 되며 최종적으로 터미널 노드에는 규칙 검사를 통과한 WME의 집합과 수행되어야 할 규칙의 동작이 저장된다.

Fig. 1의 작업 메모리 내용이 Fig. 2의 규칙 네트워크에 적용되었다면 Mouse 타입의 알파 메모리와 베타 메모리에 micky 객체가 저장되고 Food 타입의 알파 메모리에 cheese가 그리고 베타 메모리에 micky와 cheese가 저장된다. 또한 최종적으로 micky와 cheese는 규칙을 만족하여 규칙의 동작과 함께 터미널 노드에 저장이 되고 시스템은 터미널 노드를 참고하여 필요로 하는 동작을 수행한다. Fig. 2는 하나의 규칙에 대하여 동작하는 규칙 네트워크를 보여주고 있지만 실제 응용에 따라서 규칙의 수가 백 또는 천 단위를 넘기도 하기 때문에 규칙 네트워크가 매우 복잡해지며 복잡한 조건을 갖는 규칙의 경우 중간 결과 값을 저장하는 메모리의 크기가 지수적으로 증가할 수도 있다. 따라서 Rete 알고리즘의 성능과 메모리 사용을 줄이기 위한 많은 연구가 진행되었다.

2.2 Rete 알고리즘의 개선 연구

객체 간의 연관이 많은 복잡한 규칙들의 경우 터미널 노드에 가까운 베타 메모리일수록 메모리 소모량이 많기 때문에 TREAT[6]는 전체 네트워크 유지하기 보다는 베타 메모리를 제거하고 베타 검사를 위한 재검사를 반복하도록 하였다. LEAPS[7]는 베타 메모리를 제거하고 스택 구조를 사용하여 규칙을 만족하는 한 개의 상황만 구하는 지연 연산(lazy evaluation)을 사용하여 계산 속도를 향상시켰지만 매칭 결과가 정확하지 않은 단점이 있다. Rete-OO[8]는 Rete 알고리즘의 객체 지향형 구현 버전으로 신경망, 베이지안 네트워크, 애매 논리 시스템을 모사할 수 있는 기능이 추가되었지만 규칙 네트워크가 복잡해지면서 많은 계산 부담을 갖게 되었다.

Rete-ADH(Alpha Network Dual Hash)는 스마트 사무실의 복합 상황 인지 서비스를 제어하기 위한 추론 엔진으로 Rete 알고리즘의 알파 네트워크에 이중 해싱 기법을 사용하였다[9]. 규칙 조건을 만족하는 사실의 변수를 갖는 해시 표와 다시 조건을 만족하는 모든 사실을 갖는 해시 표를 가지고 베타 검사 시에 가장 가능성이 높은 사실들을 선택할 수 있어 베타 메모리의 크기도 줄이며 매칭 탐색 공간을 줄여 수행 속도를 높였다. 그러나 조건을 만족하는 사실들이 알파 네트워크에 중복 저장되며 이중 해싱을 운용하기 위한 부담이 추가된다. MiRE[10]는 자원에 제한이 있는 휴대폰의 규칙 기반 상황 인지 서비스를 제공하기 위한 경량 규칙 엔진이다. Rete 알고리즘을 기반으로 메모리의 사용을 제한하기 위하여 사용되는 사실의 수를 제한하며 변하지 않을 사실과 갱신되는 사실을 구분하여 관리한다. 이를 통해 휴대폰 상황 인지 서비스에 적합하도록 메모리의 사용을 일정하도록 제어한다.

MidSen[11]은 센서 네트워크의 다양한 상황 인지 서비스를 지원하는 미들웨어로 개발되었으며 Rete 알고리즘을 사용하지 않지만 규칙 기반 아키텍처를 채용하고 있다. 명시적으로 사건을 감지하고 이에 따라 연결되어 있는 규칙의 조건을 검사하고 만족되면 정해진 동작을 수행하는 ECA 모델을 따르고 있다. MidSen은 상황을 기술하는 사실들의 정보를 기록하여 매 사건마다 규칙을 다시 평가하기 때문에 수행 시간이 많이 걸리는 단점이 있다. [12]는 입출력 장치들의 규칙 기반 제어를 위하여 ECA 엔진을 구현한 유비쿼터스 칩을 사용한다. 유비쿼터스 칩은 응용에 따른 ECA 규칙을 적재하고 입출력 장치들과 연결되거나 칩들 간의 연결을 통하여 사무실내의 상황 인지 서비스를 효과적으로 수행함을 보여준다. 각 칩별로 제한된 수의 연결만을 지원하기 때문에 센서의 수가 늘어날수록 하드웨어 비용이 커진다.

3. 경량 규칙 엔진: LwRE

LwRE는 사물 인터넷 환경에서 상황 인지 서비스를 제공하기 위한 저사양의 스마트 사물의 규칙 엔진을 목표로 설계되었다. 특히 메모리 사용량의 최소화에 중점을 두었으며 이로 인한 성능 감소를 줄이도록 효과적인 데이터 처리가 가능하도록 해시 표를 사용하였다. 메모리 사용량을 줄이기 위하여 기존의 다른 연구에서와 같이 베타 메모리를 제거하였고 알파 네트워크의 구성과 동작을 변경하여 규칙의 전체 조건 단위의 네트워크에서 조건을 더 세분화하여 네트워크를 구성함으로써 알파 노드의 중복 요소를 제거했다. 또한 LwRE가 관리하는 모든 데이터를 종류 별로 분류된 해시 테이블에 저장하고 키를 이용하여 데이터에 접근하도록 하여 하나 이상의 중복된 요소가 생성되는 것을 방지하였으며 이로 인해 데이터 갱신도 효율적으로 이루어지도록 하였다.

3.1 전체 LwRE의 구조

LwRE은 규칙을 저장하는 규칙 메모리, 사실을 저장하는 사실 메모리, 사실의 변동에 따라 사건의 발생을 검출하는 사건 검출기, 규칙과 사실을 이용하여 상황에 해당하는 규칙을 결정하는 규칙 해석기, 그리고 선택된 규칙에 의하여 동작을 수행하는 동작 해석기로 구성된다. Fig. 3은 LwRE의 내부 구조를 보여주고 있다.

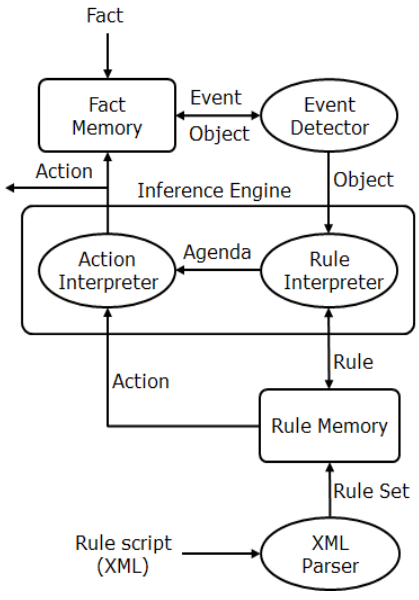

Fig. 3. The internal structure of LwRE

본 논문에서의 객체는 상황 판단을 위한 정보를 가지고 있거나 상황에 따른 동작을 수행하는 대상을 의미한다. 객체는 타입과 OID(object identifier)를 가지고 있고 이것이 객체를 고유하게 식별하는 키(key)로 사용된다. 객체에 대한 사실 정보는 객체 타입별로 모아 표 형태로 사실 메모리에 저장되며 객체 키를 이용하여 접근된다. 객체의 사실이 변경되면 사건 추출기에 의하여 해당 객체의 참조가 규칙 해석기에 전달되어 처리되도록 한다.

LwRE에서 규칙을 XML로 기술하기 위하여 규칙에 대한 XML 스키마를 정의하였다[13]. XML로 기술된 규칙은 XML 파서에 의하여 해석되어 시스템에서 사용될 규칙 집합이 규칙 메모리에 해시 표 형태로 저장된다. 본 논문에서는 LwRE 동작의 기술을 가독성 있게 하기 위하여 XML 표현 대신에 일반적인 규칙 형태로 표현한다. 규칙의 조건은 같은 타입의 객체들에 대한 사실 검사를 표현하며 내부에 속성 조건(attribute condition)과 조합 조건(join condition)으로 구성된다. 속성 조건은 객체의 속성 값에 대해 상수 값에 대한 비교를 기술하며 규칙 네트워크에서 속성 노드로 구현된다. 속성 노드는 Rete 알고리즘에서의 알파 노드와 같은 역할을 하며 속성 검사를 수행한다. 조합 조건은 객체의 속성 값을 다른 객체의 속성 값과 비교하는 연산을 기술한다. 동작은 대상과 명령으로 이루어진다. 대상은 규칙 또는 규칙에 속한 조건을 만족하는 객체들이 될 수 있으며 장치의 타입과 OID를 명시한 객체가 될 수 있다.

이와 같이 규칙은 5개의 요소로 구성되며 규칙 메모리의 5개의 해시 표로 저장된다. 규칙, 조건, 동작은 고유의 이름을 가지고 있어 각 규칙 요소를 접근하기 위한 키로 사용되며 속성 조건과 조합 조건은 규칙 초기화 시에 키가 고유하게 할당된다. 규칙 메모리 내에서 규칙의 5가지 요소는 규칙의 계층 구조에 따라 서로 참조하도록 키로 연결되어 있다. 규칙 표와 조건 표는 LwRE의 현재 상태에서 규칙 또는 조건을 만족하는 객체들의 키도 저장하며 이는 Rete 알고리즘의 베타 메모리와 알파 메모리의 역할을 한다. Fig. 4는 규칙 예와 규칙이 규칙 메모리에서 키를 이용해 연결되어 있는 형태를 도식화하여 보여 준다.

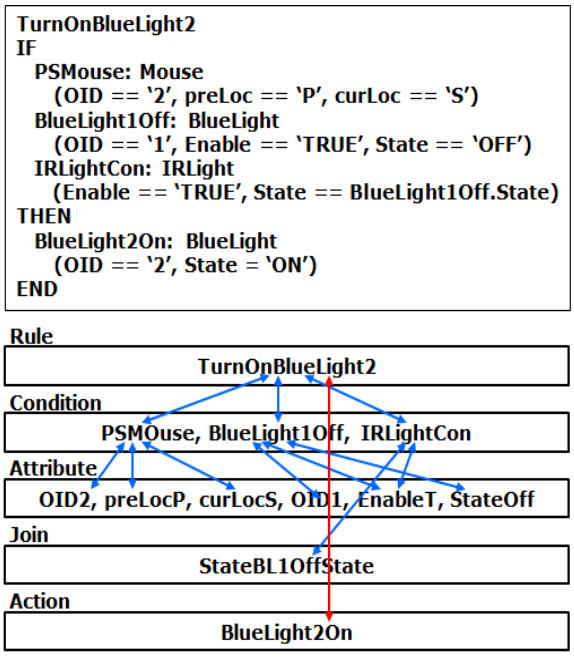

Fig. 4. A rule example and its structure in the rule memory

TurnOnBlueLight2 규칙은 PSMouse, BlueLight1Off, IRLightCon의 3개의 조건과 1개의 동작 BlueLight2On을 가지고 있다. Mouse 타입의 객체에 대해 동작하는 PSMouse 조건은 다시 3개의 속성 조건을 가지고 있으며, IRLight2On 조건은 1개의 속성 조건과 1개의 조합 조건을 가지고 있다. 이 규칙은 OID 2의 쥐가 'P'에서 'S'로 이동했고, OID 1의 파란등이 동작 가능하며 꺼져 있고, 적외선등이 동작 가능하며 OID 1의 파란등과 상태가 같은 것이 하나라도 있다면 OID 2의 파란등을 켜야 한다는 것을 표현하고 있다. 각 타입의 객체의 속성이 바뀌고 사실 메모리의 내용이 갱신되어 사건이 발생하면 추론 엔진에 의하여 규칙이 검사되고 동작하게 된다.

3.2 추론 엔진의 동작

LwRE의 규칙 네트워크 구조는 Fig. 5와 같다. 규칙 네트워크는 규칙 메모리와 독립적으로 구성되는 것이 아니라 규칙 메모리 내의 규칙 표현과 사실 메모리 내의 객체 표현의 키 연결을 통해서 구성된다.

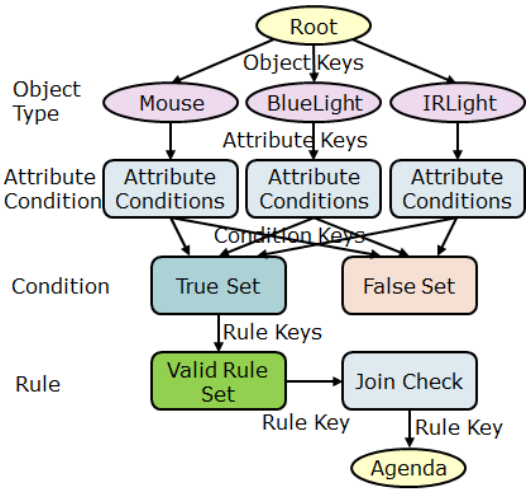

Fig. 5. A rule network structure of LwRE

루트 노드는 타입 노드와 연결되어 있으며 타입 노드는 자신이 대표하는 타입의 객체가 입력되었을 경우 검사해야 할 속성 조건들과 키를 매개로 연결되어 있다. 각 속성 조건들은 규칙 메모리 내에서 자신을 사용하는 조건과 연결되어 있기 때문에 조건의 만족은 조건이 속한 각 속성 조건들이 모두 만족하는 것으로 판단되며 만족할 경우에는 참 집합(true set)에 속하도록 하며 만족되지 않으면 거짓 집합(false set)에 속하도록 동작한다. 특정 규칙의 속성 조건이 모두 만족하는 규칙들은 유효 규칙 집합(VRS: Valid Rule Set)에 키로 저장되며 이 규칙들은 결합 조건이 있을 경우에는 다시 결합 검사를 수행하며 그렇지 않을 경우 최종적으로 만족하는 규칙으로 판단된다.

LwRE의 메모리 사용을 최소화하기 위하여 객체의 중복 저장을 제거하여 객체는 사실 메모리에서 하나로 관리되며 키를 이용하여 서로 참조할 수 있도록 하였다. 같은 타입의 속성 조건은 여러 조건에서 공유할 수 있다. 알파 메모리가 없는 대신에 참 집합의 조건에 조건을 만족하는 객체의 키를 저장한다.

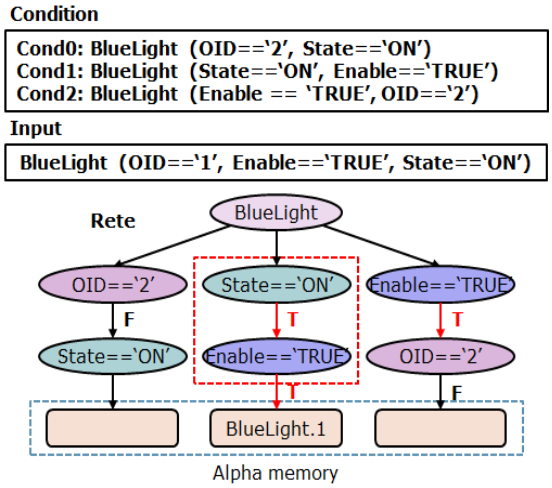

Fig. 6. An example of a condition set and its related alpha network of Rete algorithm

Rete 엔진과 LwRE의 규칙 네트워크 차이를 비교하기 위한 규칙 예와 각 규칙 네트워크의 구성 및 동작을 살펴본다. Fig. 6은 BlueLight 타입의 객체에 대해 세 개의 조건이 규칙으로 제공될 때의 Rete 알파 네트워크의 구성 및 동작을 보여준다. Cond0은 객체의 OID가 2이고, 상태가 "ON"이면 충족이고, Cond1은 상태가 "ON"이고 사용 가능하면 이면 충족이다. 마지막 Cond2는 사용 가능하며 OID가 2이면 충족이다. 그리고 입력된 객체는 OID가 1, 사용 가능하며, 상태가 "ON"이다. 예제로 주어진 입력 값에 의해 Cond0과 Cond2는 유효하지 않고, Cond1은 유효함을 알 수 있다.

알파 네트워크에서는 조건 별로 네트워크를 구성하기 때문에 서로 다른 세 조건은 각각 2개의 알파 노드로 구성되며 전체에서 속성 조건이 중복되어도 모두 6개의 알파 노드로 구성된다. 예제의 주어진 입력에 대하여 각 조건에서 알파 검사를 진행하며 중간에 만족하지 않는 속성 조건이 나타나면 계산을 멈추기 때문에 총 5번의 알파 검사를 수행하고 Cond1이 만족되기에 알파 메모리에 입력 객체를 저장한다.

Fig. 7은 Fig. 6의 조건과 입력에 대한 LwRE 규칙 네트워크와 검사 결과를 보여준다. 예제 규칙에서 속성 조건이 3개가 있기에 속성 노드는 3개가 만들어지고 입력에 대하여 3번의 속성 검사가 수행된다. 각 속성 조건은 조건과 키로 연결되어 있기에 각 속성 검사마다 참이면 거짓 집합에 있지 않은 부모 조건들을 참 집합에 넣고, 거짓이면 참 집합에 있는 부모 조건을 제거하고 거짓 집합에 넣는다. 참 집합의 조건은 자신이 관리하는 유효 객체 집합(VOS: Valid Object Set)에 입력된 객체의 키를 추가하며 거짓 집합의 조건은 VOS에 객체의 키가 존재한다면 이를 제거한다. 유효 객체 집합은 Rete의 알파 메모리와 같은 역할을 한다.

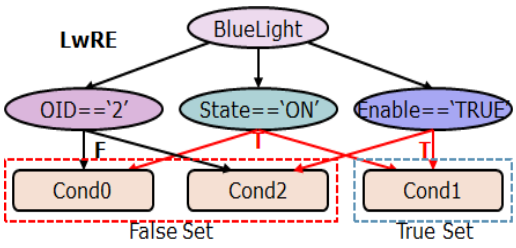

Fig. 7. A part of the rule network of LwRE for the case of Fig. 6

유효한 조건을 찾기 위한 검사 횟수를 비교해볼 때 LwRE는 모든 속성 검사를 수행하지만 Rete 알고리즘은 알파 검사 중간에 속성 조건을 만족하지 않는 알파 노드에서 멈추는 경우가 있기 때문에 규칙의 형태에 따라 검사 횟수가 달라진다. 모든 속성 조건 검사가 끝나면 참 집합에 저장된 조건들의 부모 규칙을 VRS에 추가한다. VRS에 추가된 규칙들에 대해 자신을 추가한 조건부터 조합 검사가 수행된다.

조합 검사는 Rete의 베타 테스트에 해당하며 속성 검사를 통과한 규칙들에 대하여 규칙의 조합 조건을 만족하는 객체의 유효성을 검사하여 규칙의 VOS에 유효한 객체를 반복해서 저장하는 과정이다. Fig. 8은 조합 검사를 수행하는 과정을 도시하고 있다.

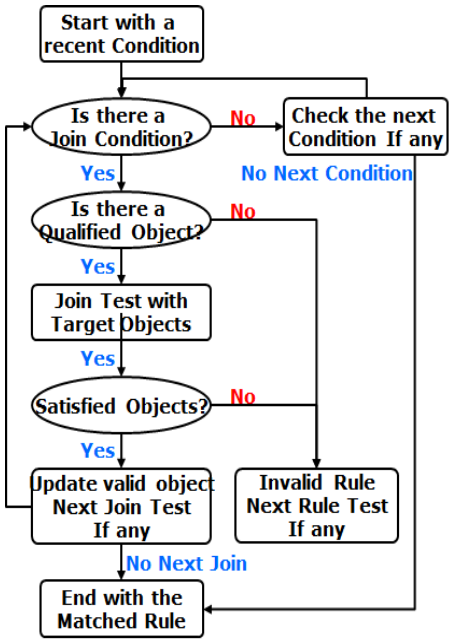

Fig. 8. The flow chart of join condition test

조합 검사는 VRS의 규칙들에 대하여 조합 조건을 만족하는 규칙이 발견될 때까지 진행한다. VRS에 있는 규칙들의 조건들에 대해 조합 조건이 있는 지를 하나씩 검사하고 조합 조건이 없다면 조건을 만족하였다고 판단하고 조합 조건이 있다면 조합 조건 검사를 수행하여 조건의 만족을 판단한다. 하나의 조건에 다수의 조합 조건이 있다면 모든 조합 조건이 만족되어야 조건이 만족되었다고 판단한다.

조합 조건이 있어서 검사를 수행할 경우에 먼저 부모 조건의 VOS을 규칙의 VOS에 추가한다. 그 다음 검사를 수행할 대상 조건에 유효한 객체가 있는지 확인한다. 조합 검사의 대상이 되는 조건은 같은 규칙 내에 속한 조건이므로, 대상 조건의 유효한 객체가 없다면 유효하지 않은 조건이기에 검사를 중단하고 현재 규칙이 유효하지 않다고 판단한다. 대상 조건을 충족하는 객체들이 있으면 규칙의 유효 객체 집합의 객체들과 조합 조건을 검사하고 조합 조건을 만족하는 객체의 쌍이 없다면 현재 규칙이 유효하지 않다고 판단한다. 조합 조건을 만족하는 모든 객체들은 양쪽 조건의 VOS에 추가하며 만족하지 않는 객체는 제거한다. 조합 조건이 더 있다면 이 과정을 반복하고 최종적으로 규칙의 조건들의 VOS 중에 하나라도 공집합이 있다면 현재의 규칙이 IF 구문을 만족하지 못한다고 판단한다.

LwRE의 조합 검사에서는 베타 메모리에 해당하는 영역을 삭제하고 매 회 연산을 반복해야 하므로 Rete의 베타 검사 보다 더 많은 시간을 소모할 수 있다. 그러나 객체를 저장하는 베타 메모리 영역이 없고 각 객체의 정보가 사실 메모리 한 곳에서만 저장되며 필요한 객체 정보는 동적으로 키를 통해서 관리되기 때문에 동적 메모리 사용량이 작다.

4. 검증 및 실험

4.1 LwRE의 구현

LwRE의 검증 및 성능 비교를 위하여 Rete 알고리즘과 LwRE를 윈도우즈 PC의 .NET Framework 4.0 프로그래밍 환경에서 C# 언어로 구현하였으며 기본 검사 규칙 집합을 이용하여 초기화 및 기본 엔진 동작을 검증하였고 최종적으로 간단한 동작을 하는 장치로 이루어진 MTS에 적용하여 전체 동작을 확인하였다. MTS는 기초과학연구소에서 쥐의 학습 및 행동을 연구하기 위한 실험 기구로 내부 방에 칸막이로 구역을 정하고 이 구역을 쥐가 돌아다니며 특정한 형태로 위치를 변경했을 때 포상을 주며 반대의 경우에는 벌칙을 주어 학습 행태를 강화하기 위한 실험 기구로 제작되었다. Fig. 9는 제작된 실험 기구를 보여주고 있다.

이 기구는 내부에 조명 장치, 파란색 LED, IR LED, 문, 음식공급기, 스피커 등을 가지고 있고 이것들은 제어 유닛에 연결되어 제어를 받는다. 쥐의 행동을 감시할 카메라가 있으며 100mS 주기로 쥐의 상태를 촬영한 영상을 제공한다. 제어 유닛과 카메라는 외부 PC와 연결이 되고 PC는 카메라 영상을 통해 쥐의 현재 위치 정보를 검출하여 규칙 엔진에 전달한다. 규칙 엔진은 사용자가 입력한 규칙 중 현재 상황과 일치하는 규칙을 찾아서 실험 기구 내부 장치들에 제어를 수행하는 동작을 제어 유닛에게 전달하고 제어 유닛은 실제 내부 장치들을 제어한다. 카메라 영상 처리를 위하여 OpenCVSharp 라이브러리를 사용하였고, 규칙 엔진이 동작하는 PC는 Intel Core i7-4770K 3.5 GHz CPU, 4 GiB DDR3 SDRAM, Toshiba 128 GB SSD 하드웨어 사양을 갖고 있다.

Fig. 9. Mouse training system

4.2 실험 환경 및 규칙

쥐가 다니는 영역은 Fig. 10과 같이 L, X, S, P, R, Y, A의 총 7개의 영역으로 구분된다. MTS는 다양한 포상과 벌칙을 사용할 수 있지만 본 논문의 LwRE의 동작 및 성능을 확인하기 위한 실험에서는 Fig. 10에서 치즈로 나타나 있는 포상을 위한 두 개의 먹이 제공 장치, 실험의 모드를 알리는 두 개의 파란 LED, 그리고 벌칙에 해당하는 스피커를 사용한다.

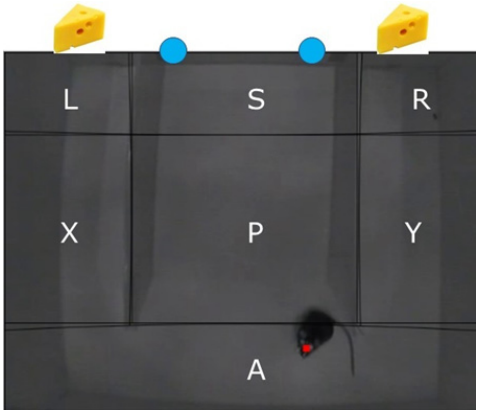

Fig. 10. The floor's regions of MTS and the equipped devices

Rete 엔진과 LwRE의 성능 및 자원 사용량을 평가하기 위하여 영상 데이터는 사전에 MTS에서 진행된 실험을 녹화한 영상을 이용하였다. 영상의 쥐의 행동은 다음과 같다. 쥐는 항상 A 구역에서 시작하며 A 구역에서 P 구역을 거쳐 S 구역으로 향하고 다시 S 구역에서 P, A, X 구역을 지나 L 구역으로 향한다. 이 영상 입력에 대해 쥐 실험자의 설정에 따라 쥐의 행동은 포상 또는 벌칙이 달라질 수 있으며 쥐의 행동 제어가 성공한 것일 수도 있으며 실패한 것일 수도 있다. 성능 비교에 사용된 규칙은 9개이다[13]. 사용된 규칙은 4개의 타입(Order, BlueLight, Speaker, Mouse)을 가지며 조건의 수는 19개이다. 중복되어 있는 조건은 9개이며 중복되어 있지 않은 조건은 10개이다. 속성 조건은 총 45개이며 중복된 속성 조건을 제외하면 16개이며 조합 조건은 사용하지 않았다. 이 경우 LwRE에서 규칙 메모리에 저장되는 조건은 중복되지 않는 10개가 저장되며 속성 조건 또한 중복되지 않는 16개가 저장된다.

주어진 영상 입력과 규칙 집합에 대하여 두 프로그램을 수행시키며 생성되는 노드의 수, 정적 규칙 엔진의 크기, 실행 시의 최대 동적 메모리 사용량, 사건 발생 시의 검사 횟수, 사건을 처리하기 위한 CPU 시간을 측정하였다.

4.3 실험 결과

1) 메모리 사용량

실험에 사용된 규칙에 대하여 LwRE는 16개의 속성 검사 노드를 생성했으며 Rete 알파 노드는 20 개가 생성되었다. 사용된 규칙들의 총 속성 조건은 45개이지만 중복된 속성 조건을 제외하면 총 16개로 LwRE는 중복된 요소를 정상적으로 제거하여 규칙 네트워크를 구성하였으며 Rete 알고리즘은 중복된 요소를 제거하지만 속성 조건 단위가 아닌 조건 단위의 중복 요소를 제거하고 조건을 공유할 수 있기 때문에 LwRE에 비하여 더 많은 알파 노드를 생성하게 된다. 즉, 속성 조건 단위의 중복성을 제거하는 LwRE 엔진이 Rete 알고리즘보다 메모리 사용에 있어 더 최적화되어 있다.

Rete 엔진과 LwRE는 규칙이 삽입될 때 각자의 방식으로 이전에 삽입된 규칙과 중복된 요소를 제거하며 규칙 네트워크를 구성한다. LwRE는 규칙을 이루는 모든 요소를 키 형식으로 공유하기에 규칙의 수가 증가할 때 기본적인 메모리 증가량이 Rete에 비해 매우 작으며 이로 인해 사용되는 전체 규칙의 수가 늘어날수록 Rete와는 많은 메모리 사용량 차이를 보여준다. Fig. 11은 9개의 규칙이 추가되면서 변화되는 엔진의 크기를 도시하고 있다.

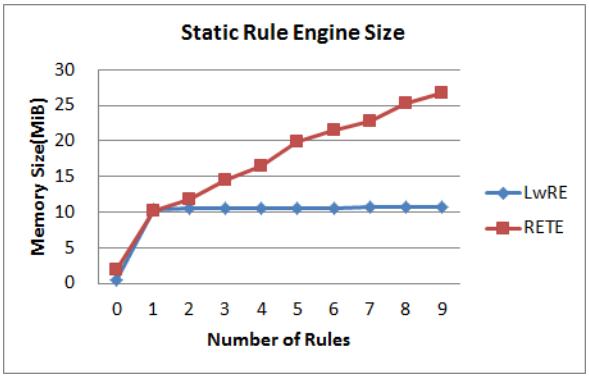

Fig. 11. Rule engine sizes of LwRE and Rete

두 엔진 모두 첫 규칙이 추가되면서 필요한 규칙 네트워크의 데이터 구조가 정리되고 비슷한 크기를 가진다. 그 후 LwRE는 규칙이 추가될 때마다 100 KiB 이내로 메모리 사용량이 증가했고 Rete 엔진은 작게는 1,226 KiB에서 크게는 3,308 KiB가 증가했다. 이는 Rete 규칙 엔진에서 규칙 네트워크 및 규칙 저장을 위한 메모리 사용이 많다는 것을 의미한다. LwRE는 규칙 메모리에 규칙의 요소들이 저장되고 각 요소들이 키를 통해 공유함으로써 메모리 사용량을 줄일 수 있었다. 복잡한 제어 시스템에서 규칙의 수가 클수록 LwRE과 Rete 엔진의 메모리 사용량 격차가 더욱 커질 것이다.

Rete 알고리즘은 수행 속도를 높이기 위하여 메모리를 많이 사용한다. LwRE는 Rete 알고리즘의 베타 메모리를 제거하고 대신에 키로 관리되는 VOS을 사용한다. 이러한 방법은 규칙 엔진 동작 중의 메모리 사용량을 줄일 수 있으며 동작 중의 메모리 범람으로 인한 시스템 정지를 피할 수 있다. Fig. 12는 LwRE와 Rete 엔진이 동작할 때의 동적 할당 메모리 사용량을 도시하고 있다. 각각 매 사건 처리를 하는 중 메모리를 최대한으로 사용하는 시점에서 측정한 값을 보여주고 있다.

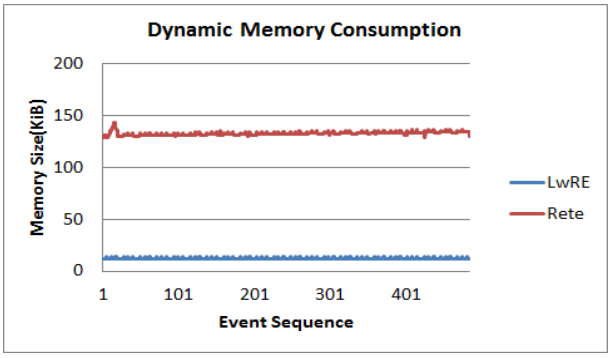

Fig. 12. Dynamic memory consumption of LwRE and Rete engine

LwRE는 계속해서 12 KiB 정도의 동적 메모리를 사용하고 있으며 상황에 따라 변동량이 많지 않다. Rete 알고리즘은 평균 133 KiB 정도의 메모리를 사용하고 있으며 상황에 따라 변동량이 LwRE에 비하여 크다. LwRE는 Rete 알고리즘보다 매우 작은 동적 메모리를 사용하고 있으며 규칙이 복잡해질수록 더 많은 차이가 있을 것으로 예측된다.

LwRE는 규칙을 조건보다 더 작은 단위인 속성 조건 단계에서 규칙 네트워크를 구성하고 관리함으로써 조건 단위로 네트워크를 구성하는 Rete 알파 네트워크보다 더 작은 크기의 규칙 네트워크를 구축할 수 있다. 또한 Rete 베타 메모리를 삭제하고 필요할 때 조합 조건 검사를 다시 계산하도록 하여 메모리 사용을 더욱 최적화 하였다.

2) 엔진의 수행 성능

Fig. 13은 사건이 발생했을 때 LwRE의 속성 검사와 Rete의 알파 검사의 수를 보여 준다. 하나의 시도에서 쥐의 움직임으로 20개의 사건이 생기고 각 사건에 따라 규칙 네트워크에 대한 노드 검사가 수행된다. LwRE는 서로 다른 상황이라도 사건을 일으킨 타입이 같다면 수행되는 속성 조건 검사의 수가 일정하며 Rete 알고리즘의 경우 같은 타입이라도 사건이 일어난 상황이 다를 경우 검사되는 알파 노드의 수가 다르다. 비록 규칙 네트워크에서 속성 조건에 대한 노드 수가 LwRE가 적지만 실제 실행 시에 검사 수는 LwRE가 많을 수 있다. Fig. 13에서 LwRE는 모든 일련의 사건에 대하여 쥐에 해당하는 모든 속성 조건 검사를 수행하여 총 11회의 검사가 수행되며 Rete 알고리즘은 상황에 따라 필요한 만큼의 검사를 수행하여 사건에 따라 6번 또는 7번의 검사를 수행한다.

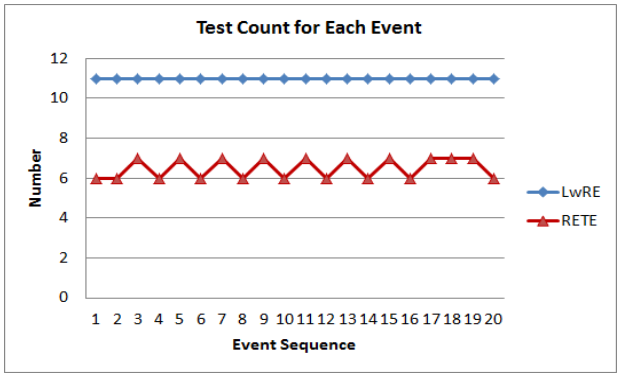

Fig. 13. The number of node tests for an event sequence

Fig. 14는 사건들을 처리하기 위해 소모되는 CPU 틱(tick)을 측정하여 비교한 그림이다. 전체 영상 구간은 세 등분하여 40분 동안 반복하여 측정한 CPU 수행 틱의 평균 값으로 데이터를 구하였다.

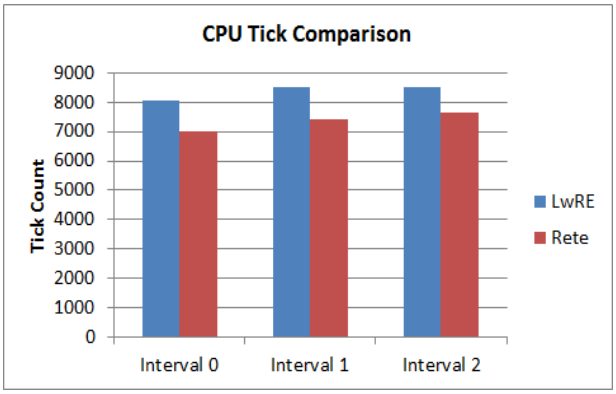

Fig. 14. CPU execution times of LwRE and Rete engine

LwRE는 Rete 엔진보다 발생한 사건을 처리하기 위해 전체적으로 13.8% 더 많은 CPU 자원을 사용했다. 이는 메모리 사용을 줄이기 위하여 알파 네트워크의 구조가 바뀌어 Rete 알고리즘의 알파 검사보다 더 많은 속성 검사를 할 수도 있으며 베타 메모리가 없기 때문에 매번 베타 검사를 수

행해야 하는 근본적인 문제로 인한 것이다. 사건이 발생했을 때 LwRE는 상황과 관계없이 사건을 야기한 객체의 타입 별로 속성 조건 검사 횟수가 고정되지만 Rete 알고리즘은 상황과 규칙 형태에 따라 한 번부터 타입별 총 속성 조건 수만큼 검사하게 된다. 따라서 속성 조건 검사의 경우 LwRE는 Rete 알고리즘에 비해 빠를 수도 느릴 수도 있다. 그러나 전체 규칙 네트워크의 규칙 검사 과정에서 해시 표를 이용한 효과적인 객체 참조 형태를 취하기 때문에 객체의 갱신이 중복되지 않고 한 번에 이루어질 수 있어 엔진 실행 속도에 있어서 사건 처리 횟수에 비하여 전체 엔진의 CPU 사용량은 감내할 수 있는 수준이다. 또한 실행 속도의 차이는 단순히 규칙의 수에 의해 증가하는 것이 아니라 규칙이 가진 속성 조건의 수와 관계가 있다. 그러므로 규칙의 수가 증가하더라도 속도 차이가 많이 늘어나는 것이 아니라 발생한 사건에 필요한 검사의 수에 의해 CPU 사용량이 정해진다. 만약 조합 조건을 포함한 규칙을 사용한다면 LwRE의 성능이 더 떨어질 수 있으나 대부분 규칙들의 특성상 조합 조건의 부분이 많지 않기 때문에 많은 성능 차이는 있지 않을 것이라 예상된다.

5. 결 론

상황 인지 서비스는 사용자가 위치한 주변 상황을 인지하여 사용자에게 상황에 맞는 유용한 서비스를 제공한다. 규칙 기반 시스템은 상황 인지 서비스를 제공하기 위하여 사용될 수 있다. 본 논문에서는 사물 인터넷 환경에서 능동적으로 동작하는 스마트 사물을 위한 계산 능력과 메모리 용량이 적은 시스템에서 상황 인지 서비스를 제공하기 위한 규칙 기반 시스템의 경량 규칙 엔진을 설계하고 구현하였다. 기존의 Rete 형태의 규칙 엔진은 다양한 응용 분야에서 사용할 수 있도록 제작되어 범용성과 처리 속도를 높였지만 메모리의 소모가 많기 때문에 자원이 제약된 환경에서 사용하기에는 어려움이 있다.

본 논문은 기존 연구에서 시도되었던 Rete 알고리즘의 엔진 크기를 줄이기 위한 베타 메모리 제거 방법을 포함하여 알파 네트워크의 구조를 변경하여 속성 조건 단위의 공유를 통한 메모리 사용을 최소화하는 접근 방법을 시도하였다. 메모리의 사용을 줄이면서 수행 시간이 늘어나는 것을 막기 위하여 모든 데이터를 해시 표를 이용하여 관리하도록 하여 데이터 갱신이 한곳에서만 이루어지도록 하였으며 데이터의 접근은 데이터 키를 이용하여 빠르게 접근하도록 하였다. 기존 Rete 알고리즘과 비교하여 실행 시간은 근소하게 늘어났지만 메모리 사용량이 현저히 줄어든 추론 엔진 LwRE를 구현하였다. LwRE는 쥐 훈련 시스템을 이용하여 동작 검증을 하였으며 쥐 이동 데이터를 기반으로 주어진 규칙들이 잘 적용되고 있음을 확인하였다. LwRE를 이용하면 실시간 동작을 요하지 않는 계산 능력과 메모리 용량이 적은 소형 임베디드 시스템과 장치들로 구성된 환경에서 상황 인지 서비스를 제공할 수 있으며 사물 인터넷 환경의 스마트 사물의 서비스 엔진으로 사용할 수 있을 것이다.

References

[1] Internet of Things [Internet], https://en.wikipedia.org/wiki/Internet_of_Things.

[2] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context aware computing for the internet of things: A survey,” *IEEE Communications Surveys & Tutorials*, Vol.16, No.1, pp.414-454, 2014.

[3] B. Y. Lim and A. K. Dey, “Toolkit to support intelligibility in context-aware applications,” in *Proc. of 12th ACM International Conference on Ubiquitous Computing*, New York, USA, pp.13-22, 2010.

[4] F. Hayes-Roth, “Rule-based systems,” *Communications of the ACM*, Vol.28, No.9, pp.921-932, 1985.

[5] C. L. Forgy, “Rete: A fast algorithm for the many pattern/many object pattern match problem,” *Artificial Intelligence*, Vol.19, No.1, pp.17-37, 1982.

[6] D. P. Miranker, “TREAT: A New and Efficient Match Algorithm for AI Production Systems,” Morgan Kaufmann Publishers, 1990.

[7] D. Batory, “The LEAPS algorithm,” Technical report, University of Texas at Austin, USA, 1994.

[8] D. Sottara, P. Mello, and M. Proctor, “A configurable Rete-OO engine for reasoning with different types of imperfect information,” *IEEE Transactions on Knowledge and Data Engineering*, Vol.22, No.11, pp.1535-1548, 2010.

[9] M. Kim, K. Lee, Y. Kim, T. Kim, Y. Lee, S. Cho, and C.-G. Lee, “Rete-ADH: An improvement to rete for composite context-aware service,” *Int. Journal of Distributed Sensor Networks*, pp.1-11, 2014.

[10] C. Choi, I. Park, S. J. Hyun, D. Lee, and D. H. Sim, “Mire: A minimal rule engine for context-aware mobile devices,” in *Proc. of the 3rd Int. Conf. on Digital Information Management*, London, UK, pp.172-177, 2008.

[11] P. Patel, S. Jardosh, S. Chaudhary, and P. Ranjan, “Context aware middleware architecture for wireless sensor network,” in *Proc. of IEEE International Conference of Services Computing*, pp.532-535, 2009.

[12] T. Terada, M. Tsukamoto, K. Hayakawa, T. Yoshihisa, Y. Kishino, A. Kashitani, and S. Nishio, “Ubiquitous chip: A rule-based i/o control device for ubiquitous computing,” in *Pervasive Computing, Lecture Notes in Computer Science*, Vol.3001, Springer Berlin Heidelberg, pp.238-253, 2004.

[13] Seung-Kyu Yoo, “LwRE: Light-weight rule engine for context-aware service,” Master’s thesis, Hankuk University of Foreign Studies, p.58, 2015.

유 승 규

e-mail : nova0120@naver.com

2013년 한국외국어대학교 컴퓨터공학과 (학사)

2015년 한국외국어대학교 컴퓨터정보통신공학과(석사)

2015년～현 재 (주)UDP 기술연구소 연구원

관심분야 : 임베디드 시스템, IoT 솔루션, 클라우드 서비스

조 상 영

e-mail : sycho@hufs.ac.kr

1988년 서울대학교 제어계측공학과(학사)

1990년 KAIST 전기전자공학과(석사)

1994년 KAIST 전기전자공학과(박사)

1994년～1995년 KAIST 정보전자연구소 위촉연구원

1995년～1996년 (미)UC Irvine ECE 방문연구원

1996년～1997년 삼성전자 선임연구원

1997년～현 재 한국외국어대학교 컴퓨터전자시스템공학부 교수

관심분야 : 임베디드 시스템, 가상 개발 환경, 병렬 시스템