ATMS를 이용한 모순처리 방식

서 정학, 박 영택
숭실대학교 정보과학대학 컴퓨터학부

조 동래, 박 영우, 주 재우
국방과학연구소 5-1-4

Contradiction Handling Using Assumption-based TMS

Jung-Hack Seo, Young-Tack Park{{jhseo, park}@multi.soongsil.ac.kr}
School of Computing Soongsil University

Dong-Lae Cho, Young-Woo Park, Jae-Woo Joo
{{dlcho, ywpark, jwjoo}@sunam.kreonet.re.kr}
Agency for Defense Development 5-1-4

요 약

ATMS(Assumption-based Truth Maintenance System)는 추론기관의 추론 과정을 기억하고 각 추론 상태의 진위를 관리해주는 기능을 수행한다. ATMS는 JTMS나 LTMS와는 다르게 각 노드의 레이블과 Nogood를 관리함으로써, 추론기관의 추론에 모순(Contradiction)이 발생하였을 때 이를 효과적으로 처리해준다. 기존의 ATMS는 모순에 영향을 주는 가정(Assumption)을 제거(Retract)함으로써 모순에 영향을 주는 원인을 제거하는 방식을 취하고 있다. 그러나, 본 논문에서는 이와 같은 방식으로 문제가 해결되지 못하는 새로운 종류의 모순을 설명하고 이를 처리하기 위해서는 ATMS가 추론기관과 연동하여 모순을 처리하는 방식에 대해서 서술하고자한다

1. 서 론

ATMS는 추론기관이 추론한 명제(Proposition)를 기억하고 이를 관리함으로써, 추론기관의 추론 과정에 도움을 준다[3,4]. 즉, 추론기관은 서술 논리(Predicate Logic)로 표현된 규칙이나 사례를 주어진 상황에 적용하여, 추론하는 과정에서는 서술 논리로 표현된 규칙이나 사례의 변수를 상수로 Unify하여 명제 형태로 바꾸어 ATMS에 전달하고, ATMS는 이와 같은 명제를 Dependency 구조로 표현하여 관리하게 된다. ATMS는 이외 같은 구조를 활용하여 추론기관에 각 사실에 대한 믿음값(Belief Value)을 기억하였다가 제공함으로써, 추론기관이 같은 추론을 다시 수행하지 않아도 결과에 대한 믿음값을 알 수 있도록 해주므로 추론 속도를 개선하는 효과가 있다[1,2].

또한, ATMS는 추론기관이 선언한 모순 규칙에 해당하는 추론이 발생하는 상우에 이를 탐지하고, 모순을 해결하는 기능을 가지고 있다. 일반적으로, 'a, b →⊥'라고 모순 규칙이 정의되었다고 가정할 때, a, b가 동시에 발생한 경우에 모순이 발생하게 된다. ATMS에서는 이러한 경우에 a 또는 b 중에서 어느 하나 이상을 제거(Retract)함으로써 모순을 해결하게 된다. 이와 같은 방식은 대부분의 경우에 문제를 해결하는 것으로 알려지고 있다[2]

본 논문에서는 이러한 경우 외의 새로운 종류의 모순을 발견하고, 이를 처리하기 위한 방법론을 제안하고자 한다. 즉, 기존의 방식은 모순에 영향을 주는 가정(Assumption)을 알아내어 그 가정을 제거함으로써, 발생한 모순을 제거하고 있다. 그러나, 추론 기관이 하나의 사실에 근거하여 여러 가지 사실을 추론한 경우에는 모순의 영향을 주는 가정만을 제거하는 것으로 추론 사실에 대한 믿음값을 관리하지 못하는 경우가 생긴다. 이와 같은 문제를 해결하기 위한 방법은 ATMS 만으로는 해결할 수 없고, ATMS와 추론 기관이 공동으로 문제를 해결하는 것이 가장 적절한 경우로 판단되고 본 논문에서는 이와 같은 방법론에 대해서 서술하고자 한다

2. 문 제(Problem)

일반적으로 ATMS가 추론 과정을 기억하는 경우는 그림1 과 같은 구조를 가지게 된다. 즉, 사실 '미국인'은 앞단의 4개의 노드에 의해서 추론되었으므로 레이블이 '{{A,B,C}}'가 배정되고, 같은 방식으로 '한국인'은 '{{A,B,D}}'가 배정된다. 이러한 경우에 '미국인' 과 '한국인'은 동시에 발생할 수 없는 경우이므로, 이 두 노드에 의해서 생성되는 레이블 '{{A,B,C,D}}'는 Nogood으로 정의된다[5]. Nogood 레이블은 추론기관이 추론할 수 없는 상황이므로, 이러한 경우에

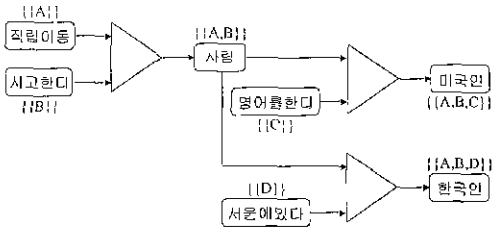

그림 1 일반적인 Nogood 처리

'{A,B,C,D}'에 있는 어느 한 가정을 제거하면 모순이 해결된다 예를 들면, 가정 'B'를 제거하기로 결정하면, 그림1에서 가정 'B'에 영향을 받는 노드 '사람'이 OUT으로 되고, 노드 '사람'는 노드 '미국인'과 '한국인'에 모두 영향을 주므로 '한국인'과 '미국인' 두 노드가 모두 OUT으로 되어 모순을 해결한다. 만약에, '영어를 한다'라는 가정이 잘못된 것으로 판정된다면, 가정 '{C}'가 제거되므로 '사람'은 IN이라 하더라도 '미국인'은 OUT이 되고 '한국인'은 가정 'C' 제거에 영향을 받지 않으므로 그대로 IN이 된다. 따라서, 모순되는 두 사실 중의 하나가 제거됨으로써 모순이 해결되게 된다

위와 같은 방식으로 모순을 해결하는 과정은 ATMS가 전적으로 수행하게 된다 즉, 추론 기관에 의해서 위의 두 추론이 실행된 경우에 ATMS는 모순을 탐지하고, 모순에 영향을 주는 가정을 추론기관이나 사용자에게 제시하여, 제거할 가정을 알아내고, 그 가정과 가정이 영향을 준 노드들의 레이블을 관리하여 ATMS 자료 구조에서 OUT으로 만들어 주는 모든 과정을 ATMS가 수행하게 된다[2] 이 과정에서 어느 가정을 제거할 것인가를 결정하는 과정만을 추론기관이나 사용자가 수행하도록 하고 있다

이와 같은 방식은 대부분 성공적이지만, 다음과 같은 경우는 문제가 위의 방식으로는 처리가 안되게 된다

그림2는 앞에서 보여준 구조와 같은 구조를 가지는 ATMS Dependency 구조이다 즉, 'move1' 추론은 'value(5)'를 이용하여 'value(7)'을 추론하고 이 값을 이용하여 'sense' 노드는 'f'를 추론하며, 'move2'는 'not f'를 추론한 경우이다 따라서, 앞의 예제와 마찬가지로 'f'와 'not f'에 의해서 Nogood 레이

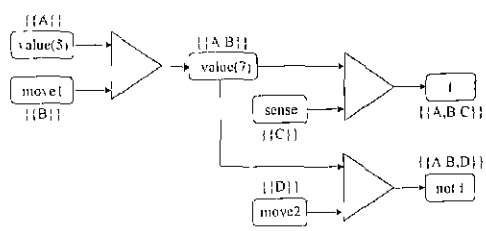

그림 2 추론기관의 도움이 필요한 Nogood 처리

블 '{{A,B,C,D}}'이 생기게 된다 이 경우에 가정 'B'를 제거하기로 결정하면 앞의 예제와 마찬가지로 'value(7), f, not f'가 모두 OUT이 된다 이 결과는 의미적으로는 정확하다. 즉, 'move1'이라는 추론을 하지 않았으면 'value(7)'이 추론되지 않았으므로 'f'와 'not f'가 동시에 발생하지 않기 때문이다.

그러나, 이 경우에 추론 기관은 상태값인 'value(5)'에 연산자 'move1'과 'move2'를 순차적으로 적용하였고, 'move2'는 'move1'의 결과를 이용하여 추론을 하였다 그러므로, 가정 'move1'을 제거한 경우라 하더라도, 'move2'는 실행이 되는 것을 가정하므로 이 경우에 'move2'는 'move1'이 실행되지 않았을 때의 상태값인 'value(5)'를 가지고 연산한 값이 필요하게 된다

즉, 그림3이 보여주는 바와 같이, 'not f'를 OUT으로 하는 기능과 아울러, 'move1'이 OUT되었으므로 'value(5)'를 이용하여 'move2'가 새로운 노드 'g'를 추론할 수 있도록 하여야 한다 이와 같은 이유는, 'not f'는 'move1'과 'move2'가 순차적으로 적용되었을 때의 결과치이고, 'g'는 'move1'이 실행되지 않았다고 가정하고 이전 value를 이용하여 'move2'가 실행된 경우의 추론 결과가 되기 때문이다.

이와 같은 Nogood 처리에는 이전 값을 발견하고 이를 활용하여 새로운 추론 값을 유도하는 과정이 필요하다 그러나, ATMS는 단순히 추론 기관의 추론 과정만을 기억하고 관리함으로 추론의 기능이 없는 모듈이다 따라서, ATMS의 일반성을 유지하면서 이와 같은 Nogood 처리를 위해서는 ATMS와 추론기관이 공동으로 모순을 처리하는 방법이 적절할 것으로 판단된다.

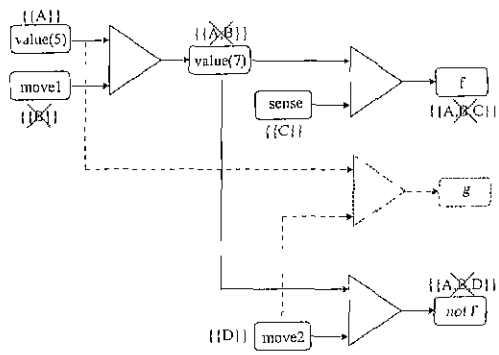

그림 3 Nogood 처리 결과

3. ATMS의 Dependency 구조

ATMS는 추론기관에 의해서 추론된 명제(Proposition)를 전달받아서 상호 의존 관계를 그래프로 표현하게 된다 앞의 그림들에서 볼 수 있듯이 추론기관이 수행한 각 사실은 ATMS의 노드로 매핑이 된다. 또한, 각 ATMS 노드는 Environment 와 Label에 의해서 표현된다. Environment는 ATMS 노드가 IN이 되는 Context를 의미하게 된다 또한, ATMS 노드는 하나 이상의 Context에서 IN이 될 수 있으므로, 이와 같은 Context들의 집합을 Label이라고 한다. 즉, Label은 Environment 들의 집합이다. 또한, Environment의 가정은 AND의 의미를 가지며, Label내의 Environment는 OR의 의미를 가진다

예를 들면, 그림2에서 노드 'f'의 Label은 '{{A,B,C}}'이다 따라서, 노드 'f'는 Environment '{A,B,C}'에서만 IN이 된다. 각 Label은 Consistent, Complete, Minimal한 특성을 가지도

록 ATMS가 관리를 한다. 만약 임의의 노드 'g'의 Label이 '{{A,B}, {C,D}}'로 정의되었다면, 노드 'g'는 두 개의 Context에서 IN이 될 수 있다 따라서, Context '{C,D}'가 성립하지 않는다 하더라도 노드 'g'는 '{{A,B}}'에 의해서 IN이 될 수 있다

이와 같이 ATMS에서 각 노드에 Label를 정의하는 이유 중의 하나는 추론기관이 추론하는 과정을 명릭적으로 표현하고, 모순이 발생하였을 때에 효과적으로 처리하기 위함이다. 그림 2에서 노드 'f'의 Label은 '{{A,B,C}}'이고 노드 'not f'의 Label은 '{{A,B,D}}'이고, 'f'와 'not f'는 동시에 발생할 수 없는 Nogood 상태이므로 Environment '{A,B,C,D}'는 Nogood이 된다 즉, 가정 A,B,C,D가 모두 IN이 되는 Context에서는 'f'와 'not f'가 모두 IN이 되므로 모순이 생기게 된다.

따라서, '{A,B,C,D}' Nogood이 성립 안되도록 하나 이상의 가정을 제거하여야 모순이 해결된다. 즉, 가정 'C'를 제거하면 노드 'f'가 OUT이 되므로 시스템에서는 'not f' 만 IN이 되어 모순이 해결된다. 이와 같은 방식으로 Nogood이 발생하면 관계되는 가정 중의 어느 하나를 제거하는 방식으로 이를 해결하게 된다.

4. ATMS와 추론기관을 이용한 모순 처리 방식

앞에서 언급한 바와 같이 대부분의 경우는 모순과 관련이 있는 가정 중에 하나 이상을 제거함으로써 모순을 처리하게 된다 그러나, 그림 2에 있는 바와 같이 추론 기관이 여러 연산자를 수행하고 각 연산자는 공통의 상태 변수를 이용하면서 그 결과로 상태 변수를 변화한다고 가정을 하자. 그러면, 앞에서 수행한 연산자가 상태를 변화시키게 되고, 뒤에 수행하는 연산자는 변화된 상태를 이용하여 새로운 연산을 수행하게 된다 즉, 그림 2에서 move2는 move1의 결과인 value(7)를 이용하여 연산을 수행하였다.

이 때, 앞단에서 수행한 연산자가 모순의 원인이 되고 그 연산자를 제거하도록 사용자가 지시하는 경우가 있다. 예를 들면, 그림 2의 Nogood 상황에서 '{A,B,C,D}' 중의 'B'가 잘못된 것이라고 사용자가 지정하는 경우이다. 즉, 사용자는 처음에 'move1'과 'move2'를 순차적으로 수행할 수 있다고 생각을 하였으나, 'move1'과 'move2'를 순차적으로 치리한 경우에 모순이 발생하였다는 것을 ATMS가 지적하였을 때, move1이 잘못되었다는 것을 발견하고 이를 제거하기로 결정할 수 있다[1]

이와 같이, 이전의 상태를 이용하여 다시 연산을 수행하는 경우에 있어서, ATMS 구조를 구축하는 데는 크게 두 가지 방식을 생각할 수 있다. 첫 번째는 ATMS에 의해서 자체적으로 실행하는 방식이다 그러나, 이 방식은 ATMS가 추론 기능을 가지어한다는 제약성이 있다. 일반적으로 ATMS는 추론기관이 추론한 명제를 효과적으로 저장하고 관리하는 기능만을 가지고 있고, 이를 추론하는 기능은 가지지 않기 때문이다 따라서, 본 연구에서는 두 번째 방식으로 추론기관이 ATMS의 Nogood 처리 과정을 모니터하여 그림 2와 같은 경우가 발생하면, Nogood의 원인으로 제거되는 가정에 대한 OUT과 이의 전달은 ATMS가 수행하도록 하고, 이에 영향을 받은 추론 과정은 다시 수행하도록 하는 방식을 취하고 있다. 그림 4가 보여주는 바와 같이 ATMS 모니터가 ATMS의 모순 제거 과정을 보면서, 부수적인 연산이 필요한 경우는 이를 추론 기관에 일러주게 된다 따라서, 이와 같은 시스템을 구성하는 경우에 ATMS 모니터는 추론기관에 Dependent한 특성을 가지게 된다.

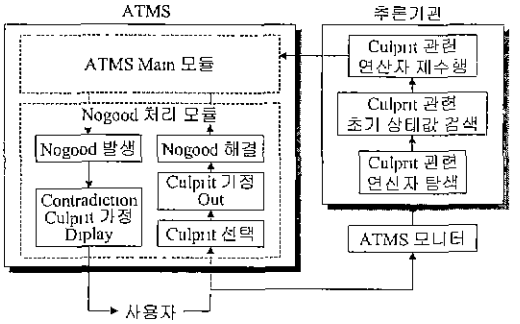

그림 4 ATMS와 추론기관을 이용한 모순 처리

5. 결론

추론기관의 추론 과정을 기억하는 ATMS는 추론기관의 추론에 모순이 발생하였을 경우 이를 효과적으로 처리해준다. 이러한 과정은 추론기관이 선언한 모순 규칙이 만족하는 경우에 ATMS가 이를 탐지하게 되고, 모순에 영향을 주는 가정을 제거함으로 처리하게 된다 하지만, 단순히 모순에 영향을 주는 가정을 제거함으로써 문제가 해결되지 않는 경우가 있다. 이에 본 논문에서는 이러한 경우를 제시하고, 해결방안을 제시하였다.

이러한 경우에는 모순이 발생한 가정을 제거하고, 제거된 가정과 연관된 연산자들은 이전 상태를 가지고 다시 계산되어져야 한다. 이렇게 하기 위해서 추론기관이 ATMS의 Nogood처리과정을 모니터하여 제거된 가정과 연관된 연산자들을 재수행하여, 그 결과를 ATMS가 다시 기억하도록 한다 이러한 방법은 중복된 연산으로 인한 더 많은 추론 과정을 ATMS가 기억해야하지만, ATMS의 독립성을 살리며 문제를 해결할 수 있다.

참고문헌

[1] Jon Dovle, A truth maintenance system, Artificial Intelligence 12, pp231-272, 1979

[2] Johan de Kleer, An Assumption-based TMS, Artificial Intelligence 28, pp127-162, 1986

[3] Johan de Kleer, Problem solving with the ATMS, Artificial Intelligence 28, pp163-196, 1986

[4] Kenneth D Forbus, Johan de Kleer, Building Problem Solvers, The MIT Press, 1993

[5] Steele, G L., The definition and implementation of a computer programming language based on constraints, Artificial Intelligence Laboratory, MIT. Cambridge, 1979

1 이 상황에서 move2를 제거하면 일반적인 Nogood처리 방식으로 해결된다