DOI: 10.3745/KIPSTB.2009.16B.6.489

계층적 작업 망 계획을 위한 변환-기반의 접근법

김 현 식[†] · 신 병 철[††] · 김 인 철[†††]

요 약

계층적 작업 망(HTN) 계획방식은 문제영역의 고유한 제어지식을 효과적으로 이용할 수 있는 대표적인 계획방식으로서, 오랫동안 복잡도가 높은 실세계 응용분야들에 널리 이용되어 왔다. 하지만 아직은 HTN 계획방식에 대한 이론적 정형화와 표준화가 미흡하며, HTN 계획기들 사이에도 원리와 성능 면에서 약간의 차이를 보이고 있다. 본 논문에서는 HTN 계획 영역 명세를 표준 PDDL 명세로 변환하는 효과적인 방법을 제시한다. 이 방법의 주된 장점은 표준 PDDL 명세를 이해할 수 있는 많은 영역-독립적인 고전적 계획기들에서도 HTN 명세에 포함된 영역 고유의 제어지식을 효과적으로 이용할 수 있다는 점이다. 본 논문에서는 블록쌓기, 로봇 심부름, 하노이 탑 등 3 가지 다른 문제영역에 HTN 명세변환 방법을 적용해보고, 전향-추론 휴리스틱 상태-공간 계획기인 FF를 이용한 실험을 통해 변환-기반의 HTN 계획방식의 효율성을 분석해 본다.

키워드 : 자동계획, 제어지식, 계층적 작업 망

A Translation-based Approach to Hierarchical Task Network Planning

Hyun-Sik Kim[†] · Byung-Cheol Shin[††] · In-Cheol Kim[†††]

ABSTRACT

Hierarchical Task Network(HTN) planning, a typical planning method for effectively taking advantage of domain-specific control knowledge, has been widely used in complex real applications for a long time. However, it still lacks theoretical formalization and standardization, and so there are some differences among existing HTN planners in terms of principle and performance. In this paper, we present an effective way to translate a HTN planning domain specification into the corresponding standard PDDL specification. Its main advantage is to allow even many domain-independent classical planners to utilize domain-specific control knowledge contained in the HTN specifications. In this paper, we try our translation-based approach to three different domains such as Blocks World, Office Delivery, Hanoi Tower, and then conduct some experiments with a forward-chaining heuristic state-space planner, FF, to analyze the efficiency of our approach.

Keywords : Automated Planning, Control Knowledge, Hierarchical Task Network

1. 서 론

일반적으로 HSP, FF, LPG와 같은 많은 영역-독립적인 고전적 계획기들(domain-independent classical planners)은 계획문제의 풀이를 위해 동작(operator)들의 명세는 이용하지만, 문제영역에 고유한 별도의 제어지식(control knowledge)을 이용하지는 않는다. 특히 최근 들어 기본 동작들의 명세로부터 영역-독립적인 휴리스틱(heuristic)을 자동으로 추출해내는 방법들이 활발히 연구됨에 따라, 영역-독립적인 계획기들의 성능이 이전에 비해 많이 향상되었다[1]. 하지만 복잡도가 높은 계획문제들에 대해서는 여전히 영역-독립적인 계획기들의 성능이 충분치 않은 실정이다.

계층적 작업 망(Hierarchical Task Network, HTN) 계획방식은 문제영역에 고유한 제어지식을 효과적으로 이용할 수 있는 대표적인 계획방식이다[2]. HTN 계획방식은 기본 동작들에 대한 명세뿐만 아니라 계획문제를 푸는데 적용할 수 있는 제어지식으로서 메서드(method)들을 정의하고 이것들을 이용한다. 각 메서드는 달성해야 하는 작업 목표들 간의 계층적 네트워크 구조를 표현한 것이다. 하나의 메서드는 하나의 작업 목표를 달성하기 위해 수행되어야 할 부속 작업들(subtasks)을 포함하고 있으며, 각 부속 작업은 또 다른 메서드나 기본 동작으로 달성된다. HTN 계획기들은 메서드들을 이용하여 최상위 작업을 하위의 부속 작업들로 거

※ 본 연구는 경기도의 경기도지역협력연구센터사업의 일환으로 수행하였음.
† 준 회 원 : 경기대학교 전자계산학과 박사과정
†† 준 회 원 : 경기대학교 컴퓨터과학과 석사과정
††† 종신회원 : 경기대학교 컴퓨터과학과 교수
논문접수 : 2009년 10월 28일
수 정 일 : 1차 2009년 11월 3일
심사완료 : 2009년 11월 3일

듭 분해(decompose)함으로써 최하위 기본 동작들로 이루어진 하나의 계획을 효과적으로 구해낼 수 있다. HTN 계획방식이 갖는 높은 효율성 때문에 인공지능 자동계획분야의 초기부터 O-Plan[3], SIPE[4], SHOP2[5] 등 많은 HTN 계획기들이 개발되어 왔고, 오늘날도 복잡도가 높은 다양한 실세계 응용분야에 널리 이용되고 있다. 하지만 현재까지도 HTN 계획방식에 대한 이론적 정형화와 표준화가 충분히 이루어지지 않고 있다. 따라서 현존하는 HTN 계획기들 사이에도 원리와 성능 면에서 약간의 차이를 보이고 있다.

이러한 문제점을 극복해보고자 그동안 HTN 계획방식과 영역-독립적인 계획방식을 결합하는 다양한 시도들이 이루어져 왔다[6]. 최근 들어서는 HTN 계획문제 명세를 전-처리과정(preprocessing)을 통해 표준 PDDL 명세로 변환하고, 변환된 이 PDDL 명세를 영역-독립적인 고전적 계획기를 이용하여 푸는 변환-기반의 HTN 계획방식들이 제안되었다[7, 8]. 이러한 변환-기반의 HTN 계획방식들은 HTN 전용 계획기가 없어도 표준 PDDL 명세를 이해할 수 있는 많은 영역-독립적인 고전적 계획기들에서 HTN 명세에 포함된 제어지식을 효과적으로 이용할 수 있다는 장점이 있다. 본 논문에서는 앞선 선행 연구들에서 제안된 방법들의 문제점들을 보완한 새로운 HTN 명세변환 방법을 제시한다. 논문에서 제시하는 방법은 특히 하나의 HTN 메서드를 부속 작업의 개수에 비례하는 수의 PDDL 동작들로 변환하는 Alford[8]의 선행 연구를 개선한 방법이다. 본 논문의 변환 방법에서는 불필요한 메서드 적용 레벨에 관한 조건들을 삭제하였다. 또한, 각 메서드의 적용 가능성을 체크하기 위한 별도의 동작을 생성하지 않고 대신 메서드에 포함된 첫 번째 부속 작업에 대응되는 동작에 그 기능을 흡수 통합함으로써 변환 결과를 최적화하였다.

본 논문에서는 블록쌓기, 로봇 심부름, 하노이 탑 등 3 가지 대표적인 문제영역을 예로 HTN 명세변환 방법을 적용해보고, 전향 추론 휴리스틱 상태-공간 계획기(forward chaining heuristic state-space planner)인 FF[9]를 이용한 실험을 통해 변환-기반의 HTN 계획방식의 효율성을 분석해본다.

2. 계층적 작업 망 계획문제

[정의] 하나의 고전적 계획문제(classical planning problem) P는 다음과 같이 정의한다.

$P = (s_0, g, O)$,

이때 s_0는 초기 상태(initial state),
g는 목표 조건들(goal conditions),
O는 동작들(operators)의 집합을 나타낸다.

각 동작 $o \in O$는 다음과 같은 요소들로 구성된다. o = (name(o), precond(o), effects(o)), 여기서 name(o)는 동작의 이름을, precond(o)는 동작의 전-조건들(preconditions)을, effects(o)는 동작의 효과들(effects)을 각각 나타낸다. 단위 동작(action) α는 하나의 동작(operator)을 구체화한 한 인스턴스(ground instance)이다. 상태 s가 전-조건 precond(α)를 만족하면, 단위 동작 α는 상태 s에 실행 가능하며 다음과 같은 결과상태 s′를 만든다. $s' = \Upsilon(s, \alpha) = (s - effects^-(\alpha)) \cup effects^+(\alpha)$, 이때 $effects^-(\alpha)$와 $effects^+(\alpha)$는 단위 동작 α의 부정 효과와 긍정 효과들을 각각 나타낸다. 계획문제 P에 대한 하나의 계획(plan)은 단위 동작들의 시퀀스인 $\pi = <\alpha_1, \alpha_2, ..., \alpha_n>$이다. 초기 상태 s_0에서 시작하여 계획 π에 포함된 단위 동작들이 차례대로 실행 가능하며 목표 조건 g를 만족하는 하나의 최종 상태 s_n에 도달 가능하면, 계획 π는 문제 P에 대한 하나의 해(solution)가 된다.

[정의] 하나의 계층적 작업 망(HTN) 계획문제 P는 다음과 같이 정의한다.

$P = (s_0, T_0, O, M)$,

이때 s_0는 초기 상태를,
T_0는 초기 작업 리스트(initial task list)를,
O는 동작(operator)들의 집합을,
M는 메서드(method)들의 집합을 각각 나타낸다.

하나의 작업(task) τ는 $t(x_1, x_2, ..., x_k)$ 형태로 표현되며, 이때 t는 작업 이름을 나타내고 각 인자 x_i는 변수나 상수를 나타낸다. 특히 작업 이름 t가 동작 이름 중 하나와 일치하는 경우에는 작업 τ를 단위 작업(primitive task)이라고 한다. 단위 작업은 하나의 단위 동작으로 실현 가능한 반면, 단위 작업이 아닌 경우, 즉 비-단위 작업(non-primitive task)의 경우에는 이것을 여러 개의 부속 작업(subtask)들로 분해할 필요가 있다. 메서드(method)는 하나의 작업을 어떻게 부속 작업들로 분해할 수 있는지를 나타내는 문제영역의 고유한 제어지식이다. 하나의 메서드 m은 다음과 같은 요소들로 구성된다.

m = (name(m), task(m), precond(m), subtasks(m)) 이때 name(m)은 메서드 이름과 인자들을, task(m)은 메서드가 분해할 수 있는 작업을, precond(m)은 전-조건들을, subtasks(m) = $<t_1, t_2, ..., t_n>$은 부속 작업들의 시퀀스를 각각 나타낸다. HTN 계획방식에서 계획 π를 생성하는데 사용된 일련의 단위 동작들과 메서드 인스턴스들을 계획 π에 대한 유도과정(derivation)이라 부른다. 일반적인 HTN 계획과정은 다음과 같다. 초기 작업 리스트 T_0와 부분 작업 리스트 T′이 각각 $T_0 = <t_1, ..., t_k>$, $T' = <t_2, ..., t_k>$와 같다고 가정하자. 만약 t_1이 하나의 단위 작업인 경우, $name(\alpha)=t_1$를 만족하는 실행 가능한 한 단위 동작 α가 존재하면, 다음 상태는 $s' = \Upsilon(s, \alpha)$이 된다. 계획문제 $P' = (s_1, T', O, M)$이 유도과정 δ인 하나의 해 계획(solution plan) π를 가진다면, 단위 동작 α와 계획 π를 접합한 새로운 계획 $\alpha \cdot \pi$ 는 계획문제 P의 한 해(solution)가 될 수 있다. 만약 t_1이 단위작업이 아닌 경우, task(m) = t_1이면서 s_0가 precond(m)을 만족하는 하나의 메서드 m이 존재하면, 계획문제 $P' = (s_0, subtasks(m) \cdot T', O, M)$이 유도과정 δ인 하나의 해 계획 π를 가질 때, π는 유도과정이 $m \cdot \delta$인 계획문제 P의 한 해가 될 수 있다.

3. 변환 방법

이 절에서는 하나의 HTN 계획문제 P = (s_0, T_0, O, M)를 동등한 하나의 고전적 계획문제 *trans*(P) = (s'_0, g, O′)로 변환하는 방법을 제시한다. 본 논문에서는 변환의 편의성과 최적화를 위해, 집합 O에 포함된 동작들을 적어도 하나의 메서드에 등장하는 동작들 O_m과 그렇지 않은 동작들 O_p로 구분한다. 그리고 메서드에 등장하지 않는 동작들 O_p은 제외하고, 적어도 하나의 메서드에 등장하는 동작들 O_m만을 변환의 대상으로 삼는다. 또한 집합 O_p에 속한 각 동작 o는 고전적인 방식대로 o = (name(o), precond(o), effects(o))로 표현하지만, 집합 O_m에 속한 각 동작 o는 o = (name(o), task(o), precond(o), effects(o))로 표현한다고 가정한다. 즉, 동작 o가 달성하는 작업(목표)인 task(o)를 추가적으로 기술한다고 가정한다. 따라서 본 논문에서 제안하는 방법은 하나의 HTN 계획문제 P = (s_0, T_0, $O_p \cup O_m$, M)를 동등한 고전적 계획문제 *trans*(P) = (s'_0, g, $O_p \cup O'$)로 변환하는 것으로 볼 수 있으며, 특히 동작들의 집합 O′은 변환 이전의 동작들인 O_m과 메서드들인 M으로부터 얻어진다.

변환에는 각 메서드 $m(x_1, ..., x_k)$와 작업 $t(y_1, ..., y_j)$에 대응하는 새로운 단위 논리식 $start_m(x_1, ..., x_k)$와 $finished_m(x_1, ..., x_k)$, $start_t(y_1, ..., y_j)$와 $finished_t(y_1, ..., y_j)$를 각각 이용한다.

먼저 동작(operator)에 대한 변환 방법을 소개한다. 변환의 대상이 되는 각 동작 o는 다음과 같은 구성요소로 정의된 것으로 가정한다.

o = (name(o), task(o), precond(o), effects(o))
name(o) = $o(x_1, ..., x_n)$
task(o) = $t(y_1, ..., y_j)$
precond(o) = $\{p_1, ..., p_j\}$
effects(o) = $\{e_1, ..., e_k\}$

<표 1>은 동작 o에 대한 변환 방법을 요약하고 있다. *trans*(o)에 의해 기존 동작 o는 새로운 동작 o′로 변환된다. 새로운 동작 o′의 이름 name(o′)은 기존 동작의 이름 $o(x_1, ..., x_n)$과 같고, 동작 o′의 전-조건들 precond(o′)과 효과들 effects(o′)은 기존 동작 o의 그것과 같다. 다만, 동작 o가 달성할 수 있는 작업 task(o) = $t(y_1, ..., y_j)$으로부터 생성된 새로운 단위 조건식 $start_t(y_1, ..., y_j)$와 $finished_t(y_1, ..., y_j)$가 새로운 동작 o′의 전-조건들과 효과들에 추가 삽입된다.

다음은 메서드(method)에 대한 변환 방법을 소개한다. 변환의 대상이 되는 각 메서드 m은 다음과 같은 구성요소로 정의된 것으로 가정한다.

m = (name(m), task(m), precond(m), subtasks(m))

name(m) = $m(x_1,....,x_n)$
task(m) = $t(y_1, ..., y_{jt})$
precond(m) = $\{p_1, ..., p_{jm}\}$
subtasks(m) = $\{t_1, ..., t_k\}$

<표 2>는 부속 작업이 없는 메서드, 즉 subtasks(m) = { } 인 메서드의 경우, 메서드 변환 방법 *trans*(m)을 요약 기술하고 있다. 이 방법에 따르면, 메서드 m은 새로운 하나의 동작 m′으로 변환된다. 이때 동작 m′의 이름은 메서드 m의 이름 $m(x_1,.....x_n)$과 같고, 동작 m′의 전-조건들 precond(m′)은 메서드 m의 전-조건들 precond(m) = $\{p_1, ..., p_{jm}\}$과 같다. 다만, 메서드 m이 달성할 수 있는 작업 task(m) = $t(y_1, ..., y_{jt})$으로부터 생성된 새로운 단위 조건식 $start_t(y_1, ..., y_{jt})$와 $finished_t(y_1, ..., y_{jt})$가 새로운 동작 o′의 전-조건들과 효과들에 적절히 추가 삽입된다. 이러한 변환을 통해 얻는 새로운 동작 m′은 작업 $t(y_1, ..., y_{jt})$와 연관된 조건식인 $start_t(y_1, ..., y_{jt})$가 만족될 때만 실행 가능하고, 실행 효과로 $\neg start_t(y_1, ..., y_{jt})$와 $finished_t(y_1, ..., y_{jt})$를 추가 생성한다.

<표 3>은 k개의 부속 작업을 포함하고 있는 메서드, 즉 subtasks(m) = $\{t_1, ..., t_k\}$인 메서드의 경우, 메서드 변환 방법 *trans*(m)을 요약 기술하고 있다. 이 방법에 따르면, k 개의 부속 작업을 포함하고 있는 하나의 메서드 m은 부속 작업마다 하나씩 총 k 개의 동작들 $\{m'_1, ..., m'_k\}$로 변환된다. 변환 방법은 첫 번째 부속 작업인 $t_1(y_{11}, ..., y_{1jt})$의 경우와 나머지 부속 작업들 $t_i(y_{i1}, ..., y_{ijt})$ i=2,...,k의 경우를 나누어 소개한다. 메서드 m의 첫 번째 부속 작업인 $t_1(y_{11}, ..., y_{1jt})$로부터 생성된 새로운 동작 m'_1의 전-조건들 precond(m'_1)에는 메서드 m의 원래 전-조건들인 $\{p_1,.....p_{jm}\}$, 이 메서드가 달성하는 작업 $t(y_1, ..., y_{jt})$와 연관된 새로운 조건식들인 $\{start_t(y_1, ..., y_{jt})\}$ 등이 포함된다. 동작 m'_1의 효과들 effects(m'_1)에는 작업 $t(y_1, ..., y_{jt})$와 연관된 조건식들인 $\{\neg start_t(y_1, ..., y_{jt}), finished_t(y_1, ..., y_{jt})\}$, 부속 작업 t_1을 위한 조건식들인 $\{start_{t1}(y_{11}, ..., y_{1jt}), \neg finished_{t1}(y_{11}, ..., y_{1jt})\}$, 부속 작업 t_2로부터 생성되는 동작 m'_2를 위한 조건식들인 $\{start_{m'2}(x_1, ..., x_n), \neg finished_{m'2}(x_1, ..., x_n)\}$ 등이 함께 포함된다. 즉, 동작 m'_1은 $start_t(y_1, ..., y_{jt})$가 만족되어 작업 $t(y_1, ..., y_{jt})$의 달성이 요구되면, 메서드 m의 적용 가능성을 체크한다. 그리고 실행 효과로 첫 번째 부속 작업 t_1과 두 번째 부속 동작 m'_2를 실행 가능하게 해준다. 나머지 부속 작업 t_i에 대응되는 새로운 동작 m'_i의 전조건들 precond(m'_i)에는 앞선 부속

〈표 1〉 동작 변환

trans(o) = o′ = (name(o′),precond(o′),effects(o′))
name(o′) = $o(x_1,....,x_n)$
precond(o′) = $\{start_t(y_1, ..., y_j), p_1,.....p_j\}$
effects(o′) = $\{\neg start_t(y_1, ..., y_j), finished_t(y_1, ..., y_j), e_1,.....,e_k\}$

〈표 2〉 부속 작업이 없는 메서드 변환

trans(m) = m′= (name(m′),precond(m′),effects(m′))
name(m′) = $m(x_1,.....x_n)$
precond(m′) =$\{start_t(y_1, ..., y_{jt}), p_1,.....p_{jm}\}$
effects(m′) =$\{\neg start_t(y_1, ..., y_{jt}),finished_t(y_1, ..., y_{jt})\}$

〈표 3〉 k개의 부속 작업을 포함한 메서드 변환

trans(m) = {m'_1, ..., m'_k}

m'_1= (name(m'_1),precond(m'_1),effects(m'_1))
name(m'_1) = $m_{t1}(x_1,.....x_n)$
precond(m'_1) ={$start_t(y_1, ..., y_{jt})$, $p_1,.....p_{jm}$}
effects(m'_1) ={$\neg start_t(y_1, ..., y_{jt})$, $start_{t1}(y_{11}, ..., y_{1jt})$,$\neg finished_{t1}(y_{11}, ..., y_{1jt})$, $start_{m'2}(x_1, ..., x_n)$,$\neg finished_{m'2}(x_1, ..., x_n)$}

m'_i= (name(m'_i),precond(m'_i),effects(m'_i)), i=2, ...,k
name(m'_i) = $m'_{ti}(x_1,.....,x_n)$
precond(m'_i) ={$start_{m'i}(x_1, ..., x_n)$, $finished_{t(i-1)}(y_{(i-1)1}, ..., y_{(i-1)jt})$}
effects(m'_i) = {$\neg start_{m'i}(x_1, ..., x_n)$, $finished_{m'i}(x_1, ..., x_n)$, $start_{ti}(y_{i1}, ..., y_{ijt})$, $\neg finished_{ti}(y_{i1}, ..., y_{ijt})$, $start_{m'(i+1)}(x_1, ..., x_n)$, $\neg finished_{m'(i+1)}(x_1, ..., x_n)$}

작업 $t_{(i-1)}$의 종료를 의미하는 조건식들인 {$finished_{t(i-1)}(y_{(i-1)1}$, ..., $y_{(i-1)jt}$)}, 자신의 시작 조건식들인 {$start_{m'i}(x_1, ..., x_n)$} 등이 포함된다. 동작 m'_i의 효과들 effects(m'_i)에는 자신의 종료를 의미하는 조건식들인 {$\neg start_{m'i}(x_1, ..., x_n)$, $finished_{m'i}(x_1$, ..., $x_n)$}, 부속 작업 t_i의 시작을 위한 조건식들인 {$start_{ti}(y_{i1}$, ..., y_{ijt}), $\neg finished_{ti}(y_{i1}$, ..., y_{ijt})}, 다음 부속 동작인 $m'_{(i+1)}$의 시작을 위한 조건식들인 {$start_{m'(i+1)}(x_1, ..., x_n)$, $\neg finished_{m'(i+1)}(x_1, ..., x_n)$} 등이 포함된다. 즉, 동작 m'_i는 앞선 부속 작업 $t_{(i-1)}$의 종료를 확인하고, 실행 효과로 i 번째 부속 작업 t_i과 다음 (i+1)번째 부속 동작 $m'_{(i+1)}$를 실행 가능하게 해준다.

다음은 계획문제를 구성하는 초기 상태(initial state) s_0와 초기 작업 리스트(initial task list) T_0에 대한 변환 방법을 소개한다. HTN 계획문제 P = (s_0, T_0, O, M)의 초기 상태 s_0와 초기 작업 리스트 T_0로부터, 고전적 계획문제 *trans*(P) = (s'_0, g, O')의 새로운 초기 상태 s'_0를 다음과 같이 생성한다.

$$s'_0 = s_0 \cup \{start_{t0}(y_{01}, ..., y_{0jt}), \neg finished_{t0}(y_1, ..., y_{0jt})\}$$

또한, 초기 작업 리스트 T_0 가 모두 달성되었을 때의 상태를 고려하여 T_0로부터 고전적 계획문제 *trans*(P)의 목표 조건 g를 생성한다.

4. 변환 예

이 절에서는 블록쌓기(Block World) 문제영역의 예를 통해 앞서 소개한 HTN 변환 방법에 따른 동작(operator)과 메서드(method), 그리고 초기 상태(initial state) 등의 변환이 이루어지는 과정을 설명한다. <표 4>는 블록쌓기 영역의 한 동작인 pickup에 대한 HTN 명세를 보여주고 있다. 이

〈표 4〉 변환 전 동작 pickup의 명세

```
(:action pickup
    :task (acquire)
    :parameters (?b - BLOCK)
    :precondition (and (clear ?b) (on-table ?b)
                       (not (done ?b)))
    :effect (and (not (clear ?b))
                 (not (on-table ?b))(holding ?b)))
```

명세에서 acquire는 로봇이 블록 하나를 손에 확보하기 위한 작업을 의미하며, release는 반대로 로봇이 손에 쥐고 있던 블록을 내려놓는 작업을 의미하는 것으로 가정한다. 그리고 acquire 작업은 pickup 혹은 unstack 동작의 실행을 통해 달성될 수 있고, 반면에 release 작업은 putdown 혹은 stack 동작의 실행을 통해 달성될 수 있다고 가정한다. <표 5>는 변환 결과로 얻어진 새로운 pickup 동작의 명세를 보여주고 있다.

변환된 pickup 동작의 명세에는 작업 부분 :task (acquire)이 삭제되고, 대신 acquire 작업의 시작을 의미하는 조건식인 (start_acquire) 이 전-조건들에 포함된다. 또한 동작의 효과들에도 기존의 효과들 외에 acquire 작업의 종료를 나타내는 조건식들인 (not (start_acquire)) 와 (finished_acquire)가 추가된다.

<표 6>은 블록쌓기 영역의 한 메서드(method)인 achieve_goals 의 HTN 명세를 나타낸다. 이 메서드는 동일한 이름의 작업인 achieve_goals를 달성하기 위해 특별한 전-조건(precondition)없이 적용될 수 있다고 가정한다. <표 7>은 변환 작업에 의해 생성된, 이 메서드를 대신하는 3개의 동작들 achieve_goals_w1, achieve_goals_w2, achieve_goals_w3에 대한 PDDL 명세를 나타내고 있다.

앞에서 정의한 HTN 영역 명세를 이용하는 임의의 블록쌓기 계획문제를 PDDL 계획문제로 변환하기 위해서는 초

〈표 5〉 동작 pickup의 변환 결과

```
(:action pickup
  :parameters (?b - BLOCK)
  :precondition
        (and (start_acquire)
        (clear ?b) (on-table ?b) (not (done ?b)))
  :effect (and
        (not (start_acquire)) (finished_acquire)
        (not (clear ?b)) (not (on-table ?b))
        (holding ?b)))
```

〈표 6〉 변환 전 메서드 achieve__goals의 명세

```
(:method achieve_goals
    :task (achieve_goals)
    :precondition ()
    :subtasks ((acquire)(release)
              (achieve_goals)))
```

〈표 7〉 메서드 achieve-goals의 변환 결과

```
(:action achieve_goals_w1
 :parameters ()
 :precondition (start_achieve_goals)
 :effect (and
         (not (start_achieve_goals))
         (finished_achieve_goals)
         (start_acquire)
         (not (finished_acquire))
         (start_achieve_goals_w2)
         (not (finished_achieve_goals_w2))))

(:action achieve_goals_w2
 :parameters ()
 :precondition (and (start_achieve_goals_w2)
             (finished_acquire))
 :effect (and
         (not (start_achieve_goals_w2))
         (finished_achieve_goals_w2)
         (start_release)
         (not (finished_release))
         (start_achieve_goals_w3)
         (not (finished_achieve_goals_w3))))

(:action achieve_goals_w3
 :parameters ()
 :precondition (and (start_achieve_goals_w3)
                    (finished_release))
 :effect (and
         (not (start_achieve_goals_w3))
         (finished_achieve_goals_w3)
         (start_achieve_goals)
         (not (finished_achieve_goals)))))
```

기 상태(initial state)를 나타내는 조건식들에 최상위 작업의 시작을 나타내는 (start_achieve_goals)를 추가해야 한다.

5. 관련 연구

Biplav Srivastava의 연구[7]에서는 HTN 계획방식의 장점을 이용하기 위해, 메서드(method)를 함께 기술할 수 있도록 표준 PDDL 명세언어의 제한적인 확장을 제안하였다. 또한 현재 개발되어있는 PDDL 기반의 많은 고전적 계획기들에서 이러한 메서드를 계획수립에 활용할 수 있도록 전-처리과정을 통해 모두 PDDL 형태의 융합동작(merged action)들로 변환하는 방법을 함께 제안하였다. 융합동작은 하나의 메서드로부터 시작하여 부속 작업들에 대한 계층적 분해과정을 통해 얻을 수 있는 단위 동작들(primitive actions)의 시퀀스 중 하나를 나타낸다. 따라서 하나의 융합동작은 마치 단위 동작들의 시퀀스를 하나로 묶어 정의한 하나의 매크로 동작(macro action)과 같은 실행효과를 가진다. 만약 각 메서드가 평균 b 개의 분기(branch)들을 포함하고, 각 분기는 평균길이 k인 부속 작업 리스트로 구성되며, 각 메서드로부터 평균 d의 깊이까지 계층적 분해과정이 진행된다고 가정한다면, 이 변환방법에 따라 하나의 메서드는 약 b^d 개의 융합동작들로 변환되며 이때 요구되는 계산시간은 $O(k^d b^d)$에 비례한다. 따라서 이 방법은 변환과정과 계획과정 모두 많은 계산시간이 요구되는 문제점이 있다. 또 목표까지 추정 도달거리(estimated goal distance)를 휴리스틱으로 사용하는 고전적 계획기들에서는 계획과정동안 언제나 융합동작들이 일반 동작들보다 우선적으로 선택되어 해 계획의 길이가 필요이상으로 길어지는 단점이 있다.

Ron Alford의 연구[8]에서는 길이가 k인 부속 작업 리스트를 포함하는 하나의 메서드를 단지 O(k)에 비례하는 계산시간을 사용하여 k+1 개의 PDDL 동작들로 변환하는 방법을 제안하였다. 그 중 첫 번째 동작은 해당 메서드의 적용가능성을 체크하고, 첫 번째 부속 작업을 시작할 수 있도록 해주는 역할을 수행한다. 나머지 k 개의 동작들은 메서드에 포함된 k 개의 부속 작업들 각각에 대응되는 동작들로서, 부속 작업들을 순차적으로 수행할 수 있도록 제어해주는 역할을 담당한다. Ron Alford의 방법에서는 메서드에 포함된 제어지식을 PDDL 동작들로 변환해내기 위해, 새로 생성되는 각 동작의 전-조건들과 효과들에는 그 동작에 대응되는 작업이나 메서드의 실행 순서를 제어하기 위한 조건들과 적용 레벨을 추가 삽입하였다. Alford의 방법과는 달리, 본 논문의 변환방법에서는 불필요한 메서드 적용 레벨에 관한 조건들을 삭제하였다. 또한, 각 메서드의 적용 가능성을 체크하기 위한 별도의 동작을 생성하지 않고 대신 메서드에 포함된 첫 번째 부속 작업에 대응되는 동작에 그 기능을 흡수 통합하였다. 따라서 본 논문의 방법은 길이가 k인 부속 작업 리스트를 포함한 각 메서드를 모두 계층적 작업 분해과정의 깊이와는 무관한 k 개의 PDDL 동작들로 변환함으로써 변환과정과 계획과정의 효율성을 한 단계 개선시켰다.

한편, Alexandre Albore의 연구[11]에서는 조건부 계획문제(contingent planning problem)를 PDDL 형태의 고전적 계획문제로 변환하는 방법을 제안하였다. 이 연구는 HTN 계획문제 대신 조건부 계획문제를 변환한다는 점에서 본 연구와는 다른 연구목적과 방법을 제시하고 있다.

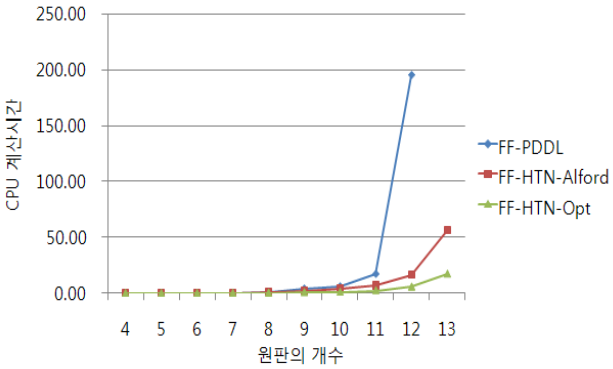
6. 실험 및 분석

앞에서 제안한 변환-기반의 HTN 계획방식의 효율성을 분석하기 위해, 블록쌓기(Block World), 로봇 심부름(Office Delivery), 하노이 탑 등 대표적인 3개의 계획문제 영역에 대해 비교 실험을 전개하였다. 실험에서는 각 문제영역별로 처음부터 PDDL로 기술한 고전적 계획문제들, Alford의 HTN 변환방법을 적용한 계획문제들, 그리고 본 논문에서 제안한 HTN 변환방법을 적용한 계획문제들을 각각 전향 추론 상태 공간 계획기인 FF로 풀어 계획수립에 소요된 총 계산시간을 비교하였다. HTN 명세에 포함된 제어지식이 변환작업을 통해 계획수립과정에 어느 정도 실질적인 효과를

미칠 수 있는지를 분석하기 위해, 실험에서는 각 영역별로 문제의 복잡도를 충분한 수준까지 점진적으로 증가시키면서 계산시간을 비교하여 보았다.

로봇 심부름(Office Delivery) 계획문제 영역은 로봇이 사무실들을 옮겨 다니며 짐을 나르는 일을 계획하는 것으로, HTN 영역 명세에는 open, move, pickup, putdown 등 4 개의 동작들과 achieve_goals 등 1 개의 메서드들을 포함하고 있다. 실험을 위한 계획문제들은 방의 개수는 총 10개로 정하는 대신, 옮겨야 할 짐의 개수를 최소 10 개에서 최대 60 개로 점진적으로 증가시켜 가면서 총 30 개를 임의로 생성하였다. (그림 1)은 로봇 심부름 계획문제들에 대한 실험 결과를 나타낸다. 계획수립에 소요된 평균 CPU 시간은 초(second) 단위로 측정하였다. 그림에서 FF-PDDL은 순수 고전적 계획문제를, FF-HTN-Alford는 Alford 방식의 변환-기반 HTN 계획문제를, FF-HTN-Opt는 본 논문의 방식에 따른 HTN 계획문제를 각각 FF로 푼 결과를 나타낸다. 그림을 통해 이 영역의 문제들에 대해 FF-HTN-Opt가 가장 빠른 성능을 보여주었음을 확인할 수 있다. 실험에서 방의 개수가 증가함에 따라 FF-PDDL의 계산시간은 FF-HTN-Alford나 FF-HTN-Opt보다 훨씬 빠르게 증가하였고, 짐의 개수가 50 개 이상인 계획문제들에 대해서는 모두 5분(= 300 초)의 제한시간 이내에 해 계획을 구하는데 실패하였다. 짐의 개수가 40 개인 계획문제들(평균 해 계획의 길이 = 약 186 개의 단위 동작들)의 경우 FF-HTN-Opt는 FF-PDDL에 비해 평균 약 24.5 배, FF-HTN-Alford에 비해 평균 약 10.5 배 정도의 빠른 성능을 보였다.

하노이 탑(Hanoi Tower) 문제영역에 대한 HTN 명세는 원판(disk) 한 장을 옮기는 move 동작 1 개와 이것을 이용하는 shiftTower, rotateTower 등 7 개의 메서드들로 구성된다. 실험에 사용한 계획문제들은 원판의 개수를 최소 4 개부터 최대 13 개까지 점진적으로 증가시켜 가면서 총 50 개를 임의로 생성하였다. (그림 2)는 하노이 탑 문제들에 대한 실험결과를 나타낸다. 하노이 탑 계획문제들에서도 문제의 복잡도가 증가함에 따라 FF-PDDL의 계산시간이 FF-HTN-Alford나 FF-HTN-Opt에 비해 더 빠르게 증가함을 알 수 있고, 특히 원판의 개수가 13 개인 문제들에서는

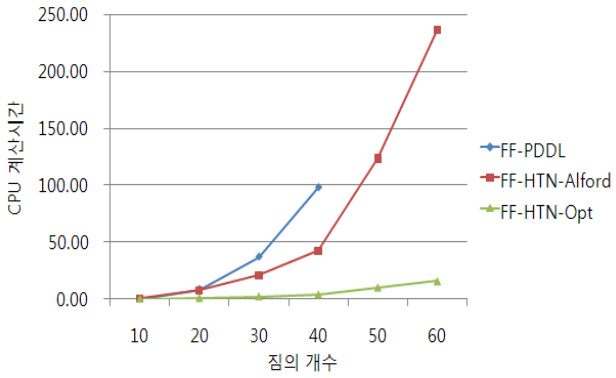


(그림 2) 하노이 탑 문제들에 대한 실험 결과

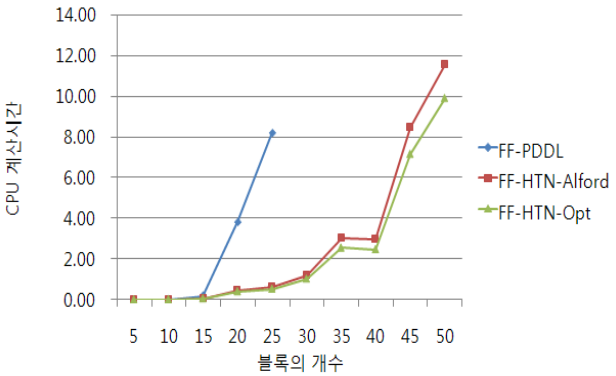
거의 대부분 제한시간 이내에 해 계획을 구하지 못했다. 원판의 개수가 11 개인 문제들(평균 해 계획의 길이 = 약 1800 개의 단위 동작들)을 기준으로 보면 FF-HTN-Opt가 FF-PDDL에 비해 평균 약 6.9 배의, FF-HTN-Alford에 비해 평균 약 2.7 배의 빠른 성능을 보였다.

블록쌓기(Block World) 문제영역에 대한 HTN 명세는 pickup, putdown, stack, unstack 등 4 개의 동작들과 이들을 이용하는 achieve_goals 1 개의 메서드로 구성된다. 이 영역에서는 블록의 개수를 5, 10, 15, ..., 50 등으로 증가시키면서 임의로 생성한 총 50 개의 계획문제들을 풀어 보았다. (그림 3)은 블록쌓기 문제들에 대한 실험결과를 보여주고 있다.

블록쌓기 영역에서 FF-PDDL은 블록의 개수가 25 개까지는 주어진 계획문제들의 대부분을 제한시간이내에 풀 수 있으나, 블록의 개수가 30 개 이상인 어려운 문제들에 대해서는 거의 대부분 제한시간 이내에 해 계획을 구하는데 실패하였다. 반면에 변환된 HTN 제어지식을 이용하는 FF-HTN-Alford와 FF-HTN-Opt는 블록의 개수가 50 개인 문제들까지 거의 모든 계획문제들을 풀 수 있었다. 블록의 개수가 20 개인 문제들(평균 해 계획의 길이 = 약 41 개의 단위 동작들)을 기준으로 비교했을 때, FF-HTN-Opt는 FF-PDDL에 비해 평균 약 9.6 배, FF-HTN-Alford에 비해 평균 약 1.2 배 정도의 빠른 성능을 보였다. 따라서 실험에 사용된 3 가지 문제영역 모두에서, 문제영역 고유의 제어지



(그림 1) 로봇 심부름 문제들에 대한 실험 결과



(그림 3) 블록쌓기 문제들에 대한 실험 결과

식을 이용할 수 있는 FF-HTN-Opt가 그렇지 못한 FF-PDDL에 비해 복잡도가 높은 계획문제들을 더 많이 풀 수 있으며, 더 빠른 성능을 보인다는 것을 확인할 수 있었다. 또한 기존의 FF-HTN-Alford에 비해서도 FF-HTN-Opt가 더 빠른 성능을 보임으로써 FF-HTN-Opt가 채용하고 있는 최적화 기법의 효과를 확인할 수 있었다.

7. 결 론

본 논문에서는 HTN 계획문제 명세에 포함된 문제영역 고유의 제어지식을 많은 영역-독립적인 고전적 계획기들에서 활용할 수 있도록 전-처리과정을 통해 HTN 계획문제 명세를 표준 PDDL 명세로 변환하는 방법을 제시하였다. 이러한 변환방법은 HTN 전용 계획기가 없어도 표준 PDDL 명세를 이해할 수 있는 많은 영역-독립적인 고전적 계획기들을 통해 HTN 명세에 포함된 제어지식을 효과적으로 이용함으로써 영역-독립적인 고전적 계획기의 성능을 높일 수 있다. 본 논문에서는 블록쌓기, 로봇 심부름, 하노이 탑 등 3개의 문제영역과 전향-추론 상태 공간 계획기인 FF를 이용한 실험을 통해 제안된 변환-기반의 HTN 계획방식의 효율성을 입증하였다. 하지만 이 방법은 다른 문제영역들과 다른 영역-독립적인 고전적 계획기들에도 동일하게 적용 가능하며, 계획기의 성능 개선이 가능하다고 판단한다. 이 방법의 한 가지 제한점은 부속 작업들이 전체 순서(total order)를 만족하는 HTN 명세에만 적용 가능하다는 점이다. 향후에는 부분 순서(partial order) HTN 명세에도 적용 가능한 변환 방법으로 확장하는 연구가 필요하다고 판단한다.

많은 실세계 응용분야들에서는 고전적 계획기들의 가정들과는 달리 상태와 동작에 많은 불확실성이 포함되어 있고, 시간적 제약과 사용자 선호도가 반영된 계획을 실시간적으로 생성하여야 하는 경우가 많다. 따라서 최근 들어 조건부 계획문제(contingent planning problem), 시간/자원 계획문제(temporal/resource planning problem), 제약만족 계획문제(satisficing planning problem) 등 다양한 형태의 계획문제들에 대한 연구가 활발히 진행되고 있다. 하지만 이들 계획문제들은 모두 영역-독립적인 접근법으로는 복잡도가 매우 높아, 현재의 기술수준으로는 계획과정의 효율성이 낮은 상태이다. 본 논문에서 제안한 HTN 변환 방법을 확장하여 이들 문제에도 적용한다면 성능 개선을 기대할 수 있을 것으로 판단한다.

참 고 문 헌

[1] 김인철, 신행철, "컴포넌트 서비스 기반의 휴리스틱 탐색 계획기," 정보처리학회논문지B, 제15B권, 제2호, pp.159-170, 2008.

[2] M. Ghallab, D. Nau, and P. Traverso, '*Automated Planning: Theory and Practice*,' Morgan Kaufmann, 2004.

[3] K. Currie, and A. Tate, "O-Plan: The Open Planning Architecture," *Artificial Intelligence*, Vol.52, No.1, pp.49-86, 1991.

[4] D. Wilkins, *Practical Planning: Extending the Classical AI Planning Paradigm*, Morgan Kaufmann, 1988.

[5] D. Nau, T.C. Au, O. Ilghami, U. Kuter, J.W. Murdock, D. Wu, and F. Yaman, "SHOP2: An HTN Planning System," JAIR, Vol.20, pp.379-404, 2003.

[6] A. Gerevini, U. Kuter, D. Nau, A. Saetti, and N. Waisbrot, "Combining Domain-Independent Planning and HTN Planning: The Duet Planner," Proceedings of ECAI-08, pp.573-577, 2008.

[7] B. Srivastava, "A Limited Extension of PDDL for Planning with Non-Primitive Actions," Proceedings of ICAPS-03 Workshop on Planning Competition, 2003

[8] R. Alford, U. Kuter, and D. Nau, "Translating HTNs to PDDL: A Small Amount of Domain Knowledge Can Go a Long Way," Proceedings of IJCAI-09, pp.1629-1634, 2009.

[9] J. Hoffmann and B. Nebel, "The FF Planning System: Fast Plan Generation through Heuristic Search," JAIR, Vol.14, pp.253-302, 2001.

[10] J. Baier, C. Fritz, and S. McIlraith, "Exploiting Procedural Domain Control Knowledge in State-of-the-Art Planners," Proceedings of ICAPS-07, 2007.

[11] A. Albore, H. Palacios, and H. Geffner, "A Translation-based Approach to Contingent Planning," Proceedings of IJCAI-09, pp.1623-1628, 2009.

김 현 식

e-mail : advance7@kyonggi.ac.kr
2001년 경기대학교 전자계산학과(학사)
2004년 경기대학교 전자계산학과(이학석사)
2005년~현 재 경기대학교 전자계산학과 박사과정
관심분야 : 자동계획, 시맨틱 웹 서비스, 지능로봇, 에이전트

신 병 철

e-mail : sbc1984@kyonggi.ac.kr
2009년 경기대학교 전자계산학전공(학사)
2009년~현 재 경기대학교 컴퓨터과학과 석사과정
관심분야 : 자동계획, 지능로봇, 기계학습, 에이전트

김 인 철

e-mail : kic@kyonggi.ac.kr

1985년 서울대학교 수학과(학사)

1987년 서울대학교 전산과학과(이학석사)

1995년 서울대학교 전산과학과(이학박사)

1996년~현 재 경기대학교 컴퓨터과학과 교수

관심분야 : 자동계획, 기계학습, 지능로봇, 에이전트