혼합 도달성 분석을 이용한 상태 불변식의 단순화

(Simplification of State Invariant with Mixed Reachability Analysis)

권 기 현 †

(Gihwon Kwon)

요 약 상태 불변식은 도달 가능한 모든 상태에서 만족되는 속성이다. 불변식은 복잡한 소프트웨어 시스템의 분석과 이해에 사용될 뿐만 아니라 안전성, 궁극성, 일관성등과 같은 시스템 검증에도 사용된다. 이와 같은 이유로 인해서, 유한 상태 기계 모델로부터 상태 불변식을 추출하는 연구가 활발히 진행되고 있다. 상태 불변식을 생성하는 기존 연구에서는 도달 가능한 상태가 모두 고려됐다. 따라서 생성된 상태 불변식은 길고 복잡해서, 사용자가 이해하기 어려웠다. 본 논문에서는 '어떻게 상태 불변식을 단순화 할 것인가?' 란 질문에 대한 답을 보인다. 상태 불변식의 복잡성은 고려되어진 상태의 크기에 강하게 좌우된다. 고려된 상태들이 작으면 작을수록, 상태 불변식의 길이는 짧다. 단순한 상태 불변식을 생성하기 위해서는, 전체 상태 공간보다는 관심 있는 특정 부분(즉 범위)에 집중해야 한다. 관심 있는 범위를 표현하기 위하여 본 논문에서는 CTL 논리를 사용한다. CTL로 범위가 표현되면, 혼합 도달성 분석을 이용하여 범위 내에 속하는 상태들을 찾는다. 명백히, 이 방법으로 계산된 상태 집합은 도달 가능한 모든 상태의 부분 집합이다. 따라서, 더 약하지만 더 이해력 있는 상태 불변식을 얻는다.

키워드 : 상태 불변식, 시제 논리 식, 고정점 계산, 도달성 분석, 속성 패턴

Abstract State invariant is a property that holds in every reachable state. It can be used not only in understanding and analyzing complex software systems, but it can also be used for system verifications such as checking safety, liveness, and consistency. For these reasons, there are many vital researches for deriving state invariant from finite state machine models. In previous works every reachable state is to be considered to generate state invariant. Thus it is likely to be too complex for the user to understand. This paper seeks to answer the question 'how to simplify state invariant?' Since the complexity of state invariant is strongly dependent upon the size of states to be considered, so the smaller the set of states to be considered is, the shorter the length of state invariant is. For doing so, we let the user focus on some interested scopes rather than a whole state space in a model. Computation Tree Logic(CTL) is used to specify scopes in which he/she is interested. Given a scope in CTL, mixed reachability analysis is used to find out a set of states inside it. Obviously, a set of states calculated in this way is a subset of every reachable state. Therefore, we give a weaker, but comprehensible, state invariant.

Key words : State invariant, Temporal logic formula, Fixed-point computations, Reachability analysis, Property specifications

1. 서 론

상태 불변식(state invariant)은 도달 가능한 모든 상태에서 만족되는 속성이다. 불변식은 복잡한 소프트웨어 시스템의 분석과 이해에 사용될 뿐만 아니라 안전성(safety), 궁극성(liveness), 일관성(consistency)등과 같은 시스템 검증에도 사용된다. 대부분의 속성이 상태 불변식과 밀접한 관련이 있기 때문에 모델 검사[1], 정리

• 본 연구는 한국전자통신연구원 위탁과제(3010-2002-0090) 연구비 지원으로 수행되었음.

† 종신회원 : 경기대학교 정보과학부 교수
khkwon@kyonggi.ac.kr

논문접수 : 2002년 8월 2일
심사완료 : 2002년 11월 27일

증명[2] 등과 같이 정형 검증에서 중요한 역할을 수행한다. 이러한 이유로, 프로그램 코드[3,4]와 유한 상태 모델[5,6]로부터 상태 불변식을 추출하는 연구가 활발히 진행되고 있다. 특히, 유한 상태 모델로부터 유도된 상태 불변식은 명세 또는 모델의 이해에 유용하다[7,8].

유한 상태 모델로부터 상태 불변식을 생성하는 기존 연구에서는 도달 가능한 상태가 모두 고려되었다. 비록 이렇게 생성된 불변식이 유용한 정보를 많이 포함하고 있다 할지라도, 불변식의 길이가 길고 복잡해서 사용자가 이해하기 어려웠다. 사실, 모델을 이해하거나 디버그 할 때 모델의 전체 상태 공간 보다는 특정 부분만을 조사하는 것이 보통이다. 불변식 또한 전체 상태 공간보다는 관심 있는 특정 부분과 관련되어 있다. 따라서, 도달 가능한 상태 공간을 모두 고려하기 보다는 관심 있는 특정 부분만을 고려하여 불변식을 생성한다면, 보다 단순한 상태 불변식을 얻을 수 있다.

요약하자면, 상태 불변식의 복잡도는 고려된 상태들의 크기에 강하게 좌우된다. 고려된 상태들의 집합이 작을수록, 상태 불변식의 길이는 더욱 짧다. 단순한 상태 불변식을 생성하기 위해서는, 전체 상태 공간보다는 관심 있는 특정 부분(즉 범위)에 집중해야 한다. 여기에서는, 관심 있는 범위를 CTL(Computation Tree Logic, [9]) 논리로 표현한다. CTL로 범위가 표현되면 전 방향 및 역 방향 도달성 분석을 혼용한 혼합 도달성 분석을 이용하여 범위에 속하는 상태 집합을 찾아내며, 이것은 도달 가능한 전체 상태의 부분 집합이다. 따라서, 더 약하지만 이해력 있는 상태 불변식을 얻는다.

혼합 도달성 분석과 관련된 연구로서, Park [6]은 RSML(Requirement State Machine Language) 명세로부터 불변식을 추출하기 위하여 혼합 도달성 분석을 사용하였다. 먼저 역 방향 탐색 후, 전 방향으로 나아가면서 불변식을 생성한다. Chan [8]은 시제 논리 질의에 대한 답을 찾기 위해서, 혼합 도달성 분석을 사용하고 있다. 그러나 이들 연구에서는 상태 공간 전체를 다루고 있다. 본 논문에서는 혼합 도달성 분석 뿐만 아니라 Dwyer [10]가 제시한 범위를 이용하여 상태 공간 전체를 다루기 보다는 관심 있는 범위에서 항상 만족되는 상태 불변식을 추출하고자 하였다.

논문의 구성은 다음과 같다: 2장에서는 유한 상태 모델과 상태 불변식에 대한 기본적인 개념을 제시한다. 그리고 3장에서는 범위를 소개하고 이를 CTL로 명세 하는 방법을 보인다. 4장에서는 혼합 도달성 분석을 이용하여 주어진 범위 내에 있는 상태들을 찾는다. 5장에서는 결론 및 향후 연구를 제시한다.

2. 배경지식

2.1 유한 상태 모델

모델 M = <S, S0, R, X, L>의 정의는 다음과 같다.

- S는 상태들의 유한 집합이다.
- $S_0 \subseteq S$는 초기 상태들의 집합이다.
- $R \subseteq S \times S$는 상태들의 전이를 나타내는 전체 관계(total relation)로서, $\forall s \in S \bullet s' \in S \bullet (s,s') \in R$이다. 즉, 모든 상태 $s \in S$에 있어 $(s,s') \in R$이면 하나의 상속자(successor) $s' \in S$이 존재한다.
- X 는 단순 명제의 유한 집합이다.
- $L : S \rightarrow 2^X$는 참이 되는 단순 명제들을 각 상태에 배정한다.

그림 1은 모델의 예이다. 이 경우 $S_0 = \{s_0\}$, $R = \{(s_0,s_1), (s_0,s_4), (s_1,s_2), (s_1,s_5), (s_2,s_5), (s_5,s_2), (s_5,s_6), (s_3,s_2), (s_3,s_3), (s_4,s_4), (s_5,s_5), (s_6,s_6)\}$이다.

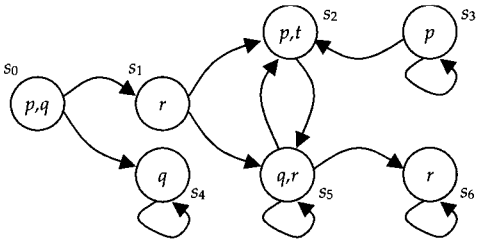


그림 1 모델의 예

$(s,s') \in R$는 s에서 s'로 한 번의 전이를 의미한다. 즉, s'은 s의 상(image)이다. 상태들의 부분 집합 P가 주어지면, P의 상들의 집합은 다음과 같다:

$image(P) = \{s' \mid \exists\, s \bullet s \in P \wedge (s,s') \in R\}$

예를 들어, $image(\{s_0\}) = \{s_1,s_4\}$. 즉, s_0로부터 한번에 s_1 과 s_4 로 전이할 수 있다. 하나의 경로(path)는 전이 가능한 상태들의 무한 시퀀스이다. 초기 상태로부터 출발하여 어떤 경로 상에 위치하면, 그 상태는 도달 가능(reachable) 하다고 한다. 예제에서, s_6는 도달 가능하다. 왜냐하면 s_6가 나타나는 경로 s_0,s_1,s_2,s_5,s_5,s_6가 있기 때문이다. 반면 s_3는 도달 가능하지 못하다. 왜냐하면 이 상태에 이르게 하는 경로가 없기 때문이다. 초기 상태로부터 고정점(fixed point)에 도달될 때까지 함수 *image*를 연속적으로 적용하면, 아래와 같이 도달 가능한 상태를 계산할 수 있다:

$P_0 := S_0$

repeat

$P_{i+1} := P_i \cup image(P_i)$
until $(P_{i+1} = P_i)$

$P_{i+1} = P_i$일 때 멈춘다. 따라서 P_i를 고정점 이라 부르며, P_i는 도달 가능한 상태 집합이다. 편의를 위해, 도달 가능한 상태들의 집합을 Q라고 하자. 그림 1의 경우, $Q = \{s_0,s_1,s_2,s_4,s_5,s_6\}$이다. 단지 도달 불가능한 s_3 만이 Q에 포함되지 않는다.

2.2 불변식의 단순 생성 방법

명제 논리의 구문은 다음과 같다:

$\phi ::= x \mid \neg\phi \mid \phi_1 \wedge \phi_2$

여기서, $x \in X$는 단순 명제를 나타내며, ¬은 부정 연사자, ∧는 논리곱 연산자이다. 상태 s가 명제식 ϕ를 만족한다면, $s \models \phi$ 라고 쓴다. 다시 말해, 상태 s에서 ϕ는 참이 됨을 의미한다.

정의 1 : 명제식과 상태와의 만족성 관계는 다음과 같다:

$s \models x$	iff	$x \in L(s)$
$s \models \neg\phi$	iff	$s \not\models \phi$
$s \models \phi_1 \wedge \phi_2$	iff	$s \models \phi_1$ and $s \models \phi_2$ ■

정리 2 : 상태 s에서 참인 단순 명제의 집합을 Y 라고 하자. 즉, $Y = L(s)$. 주어진 $Y \subseteq X$에 대해, 다음 식은 항상 만족한다.

$$s \models \bigwedge_{x \in Y} x \wedge \bigwedge_{x \in X - Y} \neg x$$

여기서 ∧는 일반화된 논리곱 연산자이다.

증명 : 귀납법을 이용하여 증명한다. 먼저 기본 경우의 증명은 다음과 같다. 기본 식은 x 혹은 $\neg x$이다. 식이 x 이라면, $x \in L(s)$ 이기 때문에 $s \models x$이다. 그렇지 않다면, $s \models x$의 경우가 아니기 때문에 $s \models x$이다. 귀납적인 경우의 증명은 다음과 같다. 길이가 n인 식 ϕ가 만족된다고 가정하자. 즉 $s \models \phi$ 라고 가정하자. ϕ의 길이가 $n+1$인 경우는 $s \models \phi \wedge x$ 또는 $s \models \phi \wedge \neg x$이다. 가정에 의해 $s \models \phi$이기 때문에 나머지 할 일은 $s \models x$ 아니면 $s \models \neg x$을 보이는 것이다. 하지만 기본 경우에서 이들을 이미 증명했다. ■

간단히 말해서, 정리 2는 하나의 상태에서 항상 만족되는 속성을 나타낸다. 예를 들어, s_2의 경우 $Y = L(s_2) = \{p, t\}$이기 때문에 $s_2 \models p \wedge \neg q \wedge \neg r \wedge t$ 이다.

정의 3 : 상태 불변식은 도달 가능한 모든 상태에서 만족되는 속성으로서 다음과 같다:

$$I = \bigvee_{s \in Q}(\bigwedge_{x \in Y} x \wedge \bigwedge_{x \in X-Y} \neg x)$$ ■

위의 정의에서, Q는 도달 가능한 상태들의 집합이며, ∨는 일반화된 이접 연산자이다. 따라서 그림 1의 경우, 상태 불변식 I는 다음과 같다:

$$\begin{aligned} I = & (p \wedge q \wedge \neg r \neg t) \\ & \vee (\neg p \wedge \neg q \wedge r \neg t) \\ & \vee (p \wedge \neg q \wedge \neg r \wedge t) \\ & \vee (\neg p \wedge q \wedge \neg r \wedge \neg t) \\ & \vee (\neg p \wedge q \wedge r \wedge \neg t) \\ & \vee (\neg p \wedge \neg q \wedge r \neg t) \end{aligned}$$

$|Q| = n$, $|X| = m$라고 했을 때, 이와 같이 단순한 방법에 의해 생성된 불변식의 길이는 $n*m$ 이다. 비록 이렇게 생성된 상태 불변식이 유용한 정보를 많이 포함하고 있다 할지라도, 불변식의 길이가 길어서 사용자가 이해하기 어렵고 복잡하다.

3. CTL로 범위 표현

3.1 범위

상태 불변식의 길이를 길게 하는 원인은 무엇인가? 길이는 Q의 크기에 밀접하게 좌우된다. Q의 크기가 클수록, 상태 불변식의 길이는 더욱 길다. 상태 불변식의 단순화를 위해, Q의 크기를 줄여야 한다. 일반적으로, 사용자는 전체 상태 공간보다는 모델의 특정 범위(scope)에 관심이 많다.

본 논문에서는 Dwyer [10]가 제시한 속성 패턴의 범위를 이용하여 사용자의 관심 범위를 표현한다. Dwyer의 속성 패턴 중에서 불변식에 해당하는 것은 '항상 참(universality)'이며, 이것의 범위를 다음과 같이 분류했다:

'φ는 전체(Globally)에서 참이다'

'φ는 이전(Before)에서 참이다',

'φ는 이후(After)에서 참이다',

'φ는 와 사이(Between and)에서 참이다',

범위가 전체인 경우 모든 상태에서 φ가 참이어야 하며, α이전의 경우 α 발생 시점 이전까지 φ가 항상 참이어야 한다. α 이후의 경우 α 발생 이후부터 φ가 항상 참이어야 하며, α와 β사이인 경우 α 발생 이후부터 β 발생 시점 이전까지 φ가 항상 참이어야 한다. 범위에서 시작 상태는 포함되지만, 종료 상태는 포함되지 않는다 (다시 말해서 left-closed & right-open 이다). 즉 α 이전 범위의 경우 α는 포함되지 않으며, α 이후의 경우 α는 포함되며, α와 β사이의 경우 α는 포함되나 β는 포함되지 않는다. 아래 그림 2는 상태 시퀀스 상에서의 각각의 범위가 차지하는 부분을 보인다.

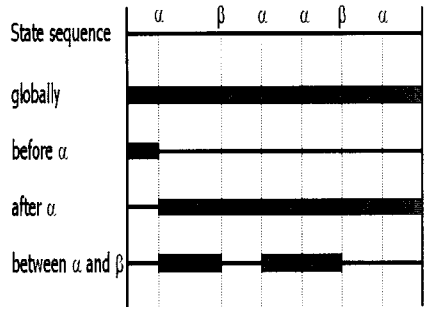


그림 2 상태 시퀀스 상에서의 범위 ([10]에서 인용)

3.2 CTL

범위를 명세하기 위해 분기 시제인 CTL 논리를 사용한다. CTL식의 구문은 다음과 같다:

ϕ ::= true | x | $\neg\phi$ | $\phi_1 \vee \phi_2$ | AGϕ | A(ϕ_1 Wϕ_2 |

여기서, A는 모든 경로(all path)를 나타내는 연산자이며, G는 전역(global) 연산자, 그리고 W는 약한(weak) until 연산자이다. 직관적으로, AG ϕ는 모든 경로에서 ϕ가 항상 참임을 의미하고, A(ϕ_1 W ϕ_2)는 모든 경로에 대해 ϕ_2 가 참이 될 때까지 ϕ_1이 참이거나 영원히 ϕ_1이 참으로 유지됨을 의미한다. $s_0 \models \phi$는 CTL식 ϕ가 상태 s_0에서 참임을 나타낸다. CTL식의 의미는 다음과 같다:

$s_0 \models$ true		(true는 항진이다.)
$s_0 \models x$	iff	$x \in L(s_0)$
$s_0 \models \neg\phi$	iff	$s_0 \not\models \phi$
$s_0 \models \phi_1 \vee \phi_2$	iff	$s_0 \models \phi_1$ or $s_0 \models \phi_2$
$s_0 \models$ AG	iff	모든 경로 $s_0, s_1, s_2, \cdots$ 와 모든 $i \geq 0$ 에 대해서 $s_i \models \phi$
$s_0 \models$ A(ϕ_1 W ϕ_2)	iff	모든 경로 $s_0, s_1, s_2, \cdots$와 $0 \leq j < i$,에 대해서, 모든 j에 대해서 $s_j \models \phi_1$ and $s_i \models \phi_2$ 이든가 또는 모든 $i \geq 0$에 대해서 $s_i \models \phi_1$ 이다.

표 1 CTL로 범위 표현

범 위	CTL로 범위 표현
ϕ는 전체에서 참이다.	AG true
ϕ는 α이전에서 참이다.	A(true $\vee$ AG($\neg\alpha$))W α
ϕ는 α이후에서 참이다.	AG($\alpha \rightarrow$AG(true))
ϕ는 α와 β 사이에서 참이다.	AG($\alpha \wedge \neg\beta$) $\rightarrow$A((true $\vee$ AG($\neg\beta$)) W β

표 1은 범위에 대한 CTL 표현을 나타내며, 범위는 그림 3과 같이 상태 시퀀스 상에서 해석된다.

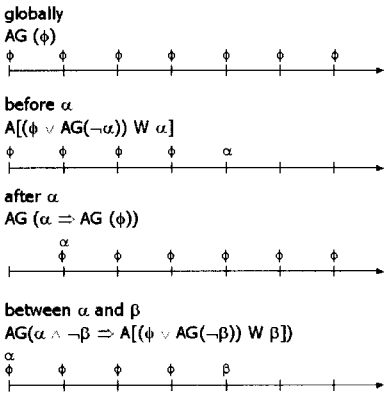


그림 3 상태 시퀀스 상에서 범위 해석

예를 들어 'AG true'는 전체 범위(즉 모든 상태 집합)를 나타내며, A((true $\vee$ AG($\neg\alpha$)) W α)는 α이전 범위(즉 α 발생 시점 이전까지의 상태 집합)를 나타낸다.

4. 상태 불변식의 생성

4.1 역 방향 도달성 분석

함수 $pre_\forall(P)$는 상태 집합 P의 선임자(predecessor) 집합을 찾는다:

$pre_\forall(P) = \{s \in S \mid \forall s \bullet \text{if } (s,s) \in R \text{ then } s \in P\}$

즉, 역 방향 도달성 분석(backward reachability analysis)을 통해서 P 로만 전이할 수 있는 상태들의 집합을 구한다. 함수 $pre(P)$를 이용하여, CTL식 를 만족하는 상태 집합을 다음과 같이 정의한다 (여기서 $||\phi||$는 ϕ를 만족하는 상태들의 집합을 나타낸다):

$||\text{true}|| = S$

$||\text{false}|| = \phi$

$|| x || = \{s \mid x \in L(s)\}$

$||\neg\phi|| = S \setminus ||\phi||$

$||\phi_1 \wedge \phi_2|| = ||\phi_1|| \cap ||\phi_2||$

$||\text{AG } \phi|| = \nu Z.(||\phi|| \cap pre_\forall(Z))$

$||\text{A}(\phi_1 \text{ W } \phi_2)|| = \nu Z.(||\phi_1 \vee \phi_2|| \cap (||\phi_2|| \cup pre_\forall(Z)))$

여기서 ν는 최대 고정점 연산자이며, CTL의 의미는 역 방향 도달성 분석으로 정의된다 [9]. 역 방향 도달성 분석만을 이용하여 상태 불변식을 생성하는 경우를 살펴보자. 예를 들어, 전체 범위의 상태 불변식을 생성하

기 위해서는 아래와 같이 전체 범위에 해당하는 상태 집합을 구한다:

$||AG\ true|| = \nu Z.(||true|| \cap pre_\forall(Z)) = S$

왜냐하면,

$Y_0 = S$

$Y_1 = \nu Z.(||true|| \cap pre_\forall(S))$

$= S \cap S$

$= S$

$Y_1 = Y_0 = S$

최대 고정점 연산자의 정의에 따라서 고정점 계산은 전체 상태 집합 S로부터 시작한다. Y_0,Y_1,Y_2는 고정점 계산 과정을 나타내는 반복 시퀀스이다. Y_i는 고정점 계산을 i 번 반복한 후의 결과를 나타낸다. 위의 경우, $Y_1 = Y_0$ 으로 고정점에 도달했고 결과는 S이다. 즉, 상태들의 전체 집합이다. 우리는 S가 아니라 도달 가능한 상태 집합인 $\{s_0,s_1,s_2,s_4,s_5,s_6\}$를 얻기 원했으나, 계산 결과에 도달 불가능한 상태 s_3도 포함되어 있다. 이와 같은 상황은 아래와 같이 'r 이전의 상태 불변식'을 생성할 때 더욱 심하다:

$||A((true \vee AG(\neg r))\ W\ r)||$

$= \nu Z.(||(true \vee AG(\neg r)) \vee r|| \cap (||r|| \cup pre_\forall(S)))$

$= S$

왜냐하면,

$Y_0 = S$

$Y_1 = \nu Z.(||(true \vee AG(\neg r)) \vee r|| \cap (||r|| \cup pre_\forall(S)))$

$= S \cap (\{s_1,s_5,s_6\} \cup S)$

$= S$

$Y_1 = Y_0 = S$

범위를 만족하는 상태 집합이 전체 상태 공간이기 때문에 'r 이전의 상태 불변식'은 다음과 같다:

$$\begin{aligned} J &= (p \wedge q \wedge \neg r \wedge \neg t) \\ &\vee (\neg p \wedge \neg q \wedge r \wedge \neg t) \\ &\vee (p \wedge \neg q \wedge \neg r \wedge t) \\ &\vee (p \wedge \neg q \wedge \neg r \wedge \neg t) \\ &\vee (\neg p \wedge q \wedge \neg r \wedge \neg t) \\ &\vee (\neg p \wedge q \wedge r \wedge \neg t) \\ &\vee (\neg p \wedge \neg q \wedge r \wedge \neg t) \end{aligned}$$

이는 2장에서의 상태 불변식 I 보다 오히려 더 길다. 즉 $I \Rightarrow J$ 이다. 그림 1을 살펴보면 $\{s_0,s_4\}$만이 'r 이전 범위'에 속한다. 그것을 제외한, 나머지 상태는 불필요하며 오히려 상태 불변식의 길이를 길게 하는 원인이다. 따라서 단순화된 상태 불변식을 생성하기 위해서는 역방향 도달성과 전 방향 도달성 분석을 혼합한 분석 기법이 필요하다.

4.2 혼합 도달성 분석

전 방향 도달성 분석(forward reachability analysis)을 수행하는 함수 $post_\exists(P)$는

$post_\exists(P) = \{s \in S \mid \exists s' \cdot s' \in P \wedge (s',s) \in R\}$

상태 집합 P를 받아서, P에서 전이할 수 있는 다음 상태들의 집합을 반환한다. 즉, P의 상속자(successors) 집합을 구한다. $R_\phi(P)$는 P로부터 ϕ를 만족하는 상태들만을 통해 도달 가능한 상태들의 집합을 구하는 함수이다 ([8]에서 R_ϕ는 집합으로 제시되었으나, 여기에서는 이를 함수로 변형하여 사용한다).

$R_\phi(P) = \mu Z.((\{s_0\} \cup post_\exists(Z)) \cap ||\neg t||)$

여기서 μ는 최소 고정점 연산자이다. $R_\phi(P)$는 상태들의 집합을 구하기 위해서 전 방향 및 역 방향을 도달성 분석을 혼용한다. 즉 ϕ를 만족하는 상태들의 집합 $||\phi||$는 역 방향으로 계산하고, 도달 가능한 상속자 집합인 $post_\exists\ (Z)$는 전 방향으로 계산하다. 예를 들어, 상태 s_0 에서 $\neg t$ 를 만족하는 상태들만을 통해 도달 가능한 상태 집합은 다음과 같다:

$R_{\neg t}(\{s_0\}) = \mu Z.((\{s_0\} \cup post_\exists(Z)) \cap ||\neg t||)$

$= \{s_0,s_1,s_4,s_5,s_6\}$

왜냐하면.

$Y_0 = \Phi$

$Y_1 = (\{s_0\} \cup post_\exists(\Phi)) \cap \{s_0,s_1,s_3,s_4,s_5,s_6\} = \{s_0\}$

$Y_2 = (\{s_0\} \cup post_\exists(\{s_0\})) \cap \{s_0,s_1,s_3,s_4,s_5,s_6\}$

$= \{s_0,s_1,s_4\}$

$Y_3 = (\{s_0\} \cup post_\exists(\{s_0,s_1,s_4\})) \cap \{s_0,s_1,s_3,s_4,s_5,s6\}$

$= \{s_0,s_1,s_4,s_5\}$

$Y_4 = (\{s_0\} \cup post_\exists(\{s_0,s_1,s_4,s_5\})) \cap \{s_0,s_1,s_3,s_4,s_5,s_6\}$

$= \{s_0,s_1,s_4,s_5,s_6\}$

$Y_5 = (\{s_0\} \cup post_\exists(\{s_0,s_1,s_4,s_5,s_6\})) \cap \{s_0,s_1,s_3,s_4,s_5\ ,s_6\}$

$= \{s_0,s_1,s_4,s_5,s_6\}\}$

$Y_5 = Y_4 = \{s_0,s_1,s_4,s_5,s_6\}$

혼합 도달성 분석을 수행하는 $R_\phi(P)$를 이용하여 범위 내에 속하는 상태들을 계산하는 알고리즘 *Find*를 아래와 같이 정의한다: *Find*는 CTL식 ϕ와 상태 집합 P를 받아서, P로부터 ϕ가 참인 도달 가능한 상태 집합을 반환한다.

Find(AG(true), P)
  $= R_{true}(P)$
Find(A(true $\vee$ AG($\neg\phi$)) W ϕ), *P*)
  $= R_{\neg\phi}(P))$
Find(AG($\phi \Rightarrow$ AG true), *P*)
  $= R_{true}(||\phi|| \cap post_\exists(R_{\neg\phi}(P)))$
Find(AG($\phi_1 \wedge \neg\phi_2 \Rightarrow$ A ((true $\vee$ AG(¬22)) W ϕ2)), *P*)
  $= R_{\neg\phi}(||\phi_1 \wedge \neg\phi||) \cap R_{true}(P))$

그림 4 *Find* 알고리즘

범위가 전체인 경우 그 결과는, P로부터 도달 가능한 모든 상태 집합이다. 범위가 'ϕ 이전'인 경우 그 결과는, P로부터 $\neg\phi$를 만족할 때까지 도달 가능한 상태 집합이다. 범위가 'ϕ 이후'인 경우 그 결과는, 처음 ϕ를 만족하는 상태부터 시작해서 도달 가능한 상태 집합이다. 범위가 'ϕ_1 와 ϕ_2 사이'의 경우 ϕ_1 를 만족하면서 $\neg\phi_2$ 를 만족하지 않는 상태부터 시작해서 $\neg\phi_2$를 만족하는 상태까지 찾아나간다. 4가지 범위에 *Find* 알고리즘을 적용한 결과를 보이면 다음과 같다.

(1) '**전체**' 범위

$Find(\mathrm{AG}(\mathrm{true}), \{s_0\})$

$= R_{\mathrm{true}}(\{s_0\})$

$= \{s_0,s_1,s_2,s_4,s_5,s_6\}$

왜냐하면,

$R_{\mathrm{true}}(\{s_0\}) = \mu Z.((\{s_0\} \cup post_{\exists}(Z)) \cap ||\mathrm{true}||)$

$Y_0 = \Phi$

$Y_1 = (\{s_0\} \cup post_{\exists}(\Phi)) \cap S$

$= \{s_0\}$

$Y_2 = (\{s_0\} \cup post_{\exists}(\{s_0\})) \cap S$

$= \{s_0,s_1,s_4\}$

$Y_3 = (\{s_0\} \cup post_{\exists}(\{s_0,s_1,s_4\})) \cap S$

$= \{s_0,s_1,s_2,s_4,s_5\}$

$Y_4 = (\{s_0\} \cup post_{\exists}(\{s_0,s_1,s_2,s_4,s_5\})) \cap S$

$= \{s_0,s_1,s_2,s_4,s_5,s_6\}$

$Y_5 = (\{s_0\}\ post_{\mathrm{E}}(\{s_0,s_1,s_2,s_4,s_5,s_6\})) \cap S$

$= \{s_0,s_1,s_2,s_4,s_5,s_6\}$

$Y_5 = Y_4 = \{s_0,s_1,s_2,s_4,s_5,s_6\}$

(2) 'r **이전**' 범위

$Find(\mathrm{A}((\mathrm{true} \vee \mathrm{AG}(\neg r))\ \mathrm{W}\ r), \{s_0\})$

$= R_{\neg r}(\{s_0\})$

$= \{s_0,s_4\}$

왜냐하면,

$Rr(\{s_0\}) = \mu Z.((\{s_0\} \cup post_{\exists}(Z)) \cap ||\neg r||)$

$Y_0 = \Phi$

$Y_1 = (\{s_0\} \cup post_{\exists}(\Phi)) \cap \{s_0,s_2,s_3,s_4\}$

$= \{s_0\}$

$Y_2 = (\{s_0\} \cup post_{\exists}(\{s_0\})) \cap \{s_0,s_2,s_3,s_4\}$

$= \{s_0,s_4\}$

$Y_3 = (\{s_0\} \cup post_{\exists}(\{s_0,s_4\})) \cap \{s_0,s_2,s_3,s_4\}$

$= \{s_0,s_4\}$

$Y_3 = Y_2 = \{s_0,s_4\}$

(3) 'r**이후**' 범위

$Find(\mathrm{AG}(r \Rightarrow \mathrm{AG\ true}), \{s_0\})$

$= R_{\mathrm{true}}(||\phi|| \cap post_{\exists}(R_{\neg r}(\{s_0\})))$

$= \{s_1,s_2,s_5,s_6\}$

왜냐하면,

$R_{\mathrm{true}}(||r|| \cap post_{\exists}(R_{\neg\phi}(\{s_0\})))$

$= R_{\mathrm{true}}(\{s_1,s_5,s_6\} \cap post_{\exists}(\{s_0,s_4\}))$

$= R_{\mathrm{true}}(\{s_1,s_5,s_6\} \cap \{s_1,s_4\})$

$= R_{\mathrm{true}}(\{s_1\})$

$= \mu Z.((\{s_1\} \cup post_{\exists}(Z)) \cap ||\mathrm{true}||)$

$Y_0 = \Phi$

$Y_1 = (\{s_1\} \cup post_{\exists}(\Phi)) \cap S$

$= \{s_1\}$

$Y_2 = (\{s_1\} \cup post_{\exists}(\{s_1\})) \cap S$

$= \{s_1,s_2,s_5\}$

$Y_3 = (\{s_1\} \cup post_{\exists}(\{s_1,s_2,s_5\})) \cap S$

$= \{s_1,s_2,s_5,s_6\}$

$Y_4 = (\{s_1\} \cup post_{\exists}(\{s_1,s_2,s_5,s_6\})) \cap S$

$= \{s_1,s_2,s_5,s_6\}$

$Y_4 = Y_3 = \{s_1,s_2,s_5,s_6\}$

(4) 'q**와** t **사이**' 범위

$Find(\mathrm{AG}(q \wedge \neg t \Rightarrow \mathrm{A}((\mathrm{true} \vee \mathrm{AG}(\neg t))\ \mathrm{W}\ t)), \{s_0\})$

$= Find(\mathrm{AG}(q \wedge \neg t \Rightarrow \mathrm{A}(\mathrm{true}\ \mathrm{W}\ t)), \{s_0\})$

$= R_{\neg r}(\ ||\ q \wedge \neg t\ ||\ \cap R_{\mathrm{true}}(\{s_0\}))$

$= R_{\neg r}(\{s_0,s_4,s_5\} \cap S)$

$= R_{\neg r}(\{s_0,s_4,s_5\})$

$= \mu Z.((\{s_0,s_4,s_5\} \cup post_{\exists}(Z)) \cap ||\neg t||)$

$Y_0 = \Phi$

$Y_1 = (\{s_0,s_4,s_5\} \cup post_{\exists}(\Phi)) \cap \{s_0,s_1,s_3,s_4,s_5,s_6\}$

$= \{s_0,s_4,s_5\}$

$Y_2 = (\{s_0,s_4,s_5\} \cup post_{\exists}(\{s_0,s_4,s_5\})) \cap \{s_0,s_1,s_3,s_4,s_5,s_6\}$

$= \{s_0,s_1,s_4,s_5,s_6\}$

$Y_3 = (\{s_0,s_4,s_5\} \cup post_{\exists}(\{s_0,s_1,s_4,s_5,s_6\})) \cap \{s_0,s_1,s_3,s_4,s_5,s_6\}$

$= \{s_0,s_1,s_4,s_5,s_6\}$

$Y_3 = Y_2 = \{s_0,s_1,s_4,s_5,s_6\}$

혼합 도달성 분석은 역 방향 도달성 분석과 비교해서, 주어진 범위를 만족하는 최소 상태 집합을 제공한다. 예를 들어 'r 이전' 범위를 혼합 도달성 분석으로 계산하면 $\{s_0,s_4\}$ 이지만, 역 방향 분석만으로 계산하면 $\{s_0,s_1,s_2,s_3,s_4,s_5,s_6\}$ 이다. 혼합 도달성 분석에 의해 생성된 불변식 K는

$K = (p \wedge q \wedge \neg r \wedge \neg t)$

$\vee(\neg p \wedge q \wedge \neg r \wedge \neg t)$

4.1절의 역 방향 분석에 의해서 생성된 I 보다 간단하다. K=I 이기 때문에, K는 I 보다는 약하지만 이해력 있는 상태 불변식이다.

4.3 활용

이러한 연구 결과는 모델 검사에 활용될 수 있다. 예를 들어, α 발생 이전에 AG p 가 만족되어야 함에도 불구하고 모델 검사기로부터 불만족 판정을 받은 경우를 고려해 보자. 이러한 경우, 혼합 도달성 분석으로 이용하여 α 이전 범위 Q 를 구한다. 그런 후 아래와 같은 식을 생성한다.

$\vee_{s \in Q} (\wedge_{x \in Y} x)$

그러면 왜 AG p 가 만족되지 않은지를 알 수 있다. 예를 들어 그림 1에서 'r 이전'의 경우 $Q = \{s_0, s_4\}$ 이다. 따라서 생성된 식은 $(p \wedge q) \vee q$ 이다. 따라서 'r 이전'의 경우 AG p는 만족할 수 없고, 대신 AG q 또는 AG $(p \vee q)$ 가 만족됨을 알 수 있다. 이러한 정보는 모델 검사기의 반례(counter example)를 이해하는데 도움을 준다. 또한 본 연구는 속성 명세를 도울 수 있다. 예를 들어 전체 범위에 해당하는 불변식을 생성 함으로서, 안전성 명세 작성을 도울 수 있다. 현재 속성 명세를 도와주기 위해서 명세 패턴 [10], GIL(Graphical Interval Logic) [11] 등이 사용되고 있다.

5. 결론 및 향후 연구

최근, 정형 검증 기술의 활성화에 발맞추어 상태 불변식의 생성에 관한 연구가 많이 수행되고 있다 [3,4,5,6,7,8]. 기존 연구에서 생성된 상태 불변식은 유용한 정보를 매우 많이 포함하고 있지만, 불변식의 길이가 길어서 이해하기 어려웠다. 왜냐하면 도달 가능한 상태 공간을 모두 고려했기 때문이다. 하지만, 대부분의 상태 불변식은 전체 상태 공간 보다는 특정 범위에 관련되어 있다. 이러한 사실에 착안해서, 본 연구에서는 Dwyer [10]가 제시한 범위 패턴을 이용하여 범위를 CTL로 표현한 후, 혼합 도달성 분석으로 범위를 만족하는 상태들의 집합을 구했다. 그 결과 약하지만 이해력 높은 상태 불변식을 얻게 되었다.

향후 연구 내용은 두 가지가 있다. 첫째, OBDD (Ordered Binary Decision Diagram) [12]를 이용하여 상태 불변식을 자동적으로 생성하는 것이다. 또 다른 하나는, 산업 현장에서 가장 광범위하게 이용되는 상태도(statecharts) [13] 로부터 불변식을 생성하는 것이다.

참 고 문 헌

[1] E.M. Clarke, E.A. Emerson, A.P. Sistla, "Automatic verification of finite state concurrent systems using temporal logic specifications," ACM Transactions on Programming Languages and Systems, Vol.8, No.2, pp. 244-263, 1986.

[2] Z. Manna, et.al., "STeP: The Stanford temporal prover," Technical Report STAN-CS-TR-94-1518, Computer Science Department, Stanford University, 1994.

[3] S. Bensalem, Y. Lakhnech, "Automatic generation of invariants," Formal Methods in System Design, Vol.15, No.1, pp.75-92, 1999.

[4] S. Bensalem, Y. Lakhnech, H. Saidi, "Powerful techniques for the automatic generation of invariants," Proceedings of CAV' 96, LNCS 1102, pp. 323-335, 1996.

[5] R. Jeffords, C. Heitmeyer, "Automatic generation of state invariants from requirements specifications," In Proceedings of ACM SIGSOFT Symposium on Foundations of Software Engineering, 1998.

[6] Y.W. Park, et.al., "Static analysis to identify invariants in RSML specifications," In Proceedings of Formal Techniques in Real-Time and Fault-Tolerant '98, LNCS 1486, Springer, 1998.

[7] C. Heitmeyer, et.al., "Using abstraction and model checking to detect safety violations in requirements specifications," IEEE Transactions on Software Engineering, Vol.24, No.11, 1998.

[8] W. Chan, "Temporal Logic Queries," In Proceedings of CAV 2000, LNCS 1855, Springer, 2000.

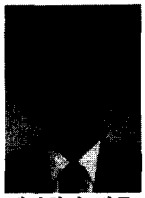
[9] E.M. Clarke, O. Grumberg, D. Peled, Model Checking, MIT Press, 1999.

[10] M.B. Dwyer, G.S. Avrunin, J.C. Corbett, "Property specification patterns for finite-state verification," In Proceedings of the Workshop on Formal Methods in Software Practice, 1998.

[11] L. K. Dillon, et.al., "A graphical interval logic for specifying concurrent systems", ACM Transactions on Software Engineering and Methodology, Vol.3, No.2, pp.131-165, 1994.

[12] R.E. Bryant, Graph-Based Algorithms for Boolean Function Manipulation, IEEE Transactions on Computer, Vol. 35, No.8, pp.677-691, 1986.

[13] D.Harel, A. Naamad, "The STATMATE semantics of statecharts, ACM" Transactions on Software Engineering and Methodology, Vol.5, No.4, pp.293-333, 1996.



권 기 현

1985년 경기대학교 전자계산학과 졸업. 1987년 중앙대학교 전자계산학과 졸업(이학석사). 1991년 중앙대학교 전자계산학과 졸업(공학박사). 1998년~1999년 독일 드레스덴 대학 전자계산학과 방문교수. 1999년~ 2000년 미국 카네기 멜론 대학 전자계산학과 방문교수. 1991년~2002년 경기대학교 정보과학부 교수