추상 시간 기계를 이용한 실시간 시스템의 도달성에 대한 검증 방법

(A Method to Verify the Reachability of Real-Time Systems using Abstract Timed Machines)

박 지 연 † 이 문 근 ††

(Ji-yeon Park) (Moon-kun Lee)

요 약 본 논문은 ATM(Abstract Timed Machine)으로 명세된 실시간 시스템을 검증하기 위한 방법을 기술한다. ATM은 임무 위급 시스템인 실시간 시스템을 명세, 분석, 검증하기 위한 정형기법이다. ATM은 모드와 전이, 포트로 구성되어 있으며 모드는 머신의 압축된 상태를 표현한다. 전이는 하나의 모드에서 다른 모드로의 전환을 나타내며 조건과 이벤트로 구성되어 있다. 포트는 ATM간의 상호작용을 위한 진입을 표현한다. 다른 정형기법과 비교하여 ATM은 소프트웨어의 순환공학 과정에서 사용하기 위해 설계되었다. 역공학 측면에서 볼 때 ATM은 계산 논리뿐만 아니라 실시간 시스템의 실제 소스코드에 있는 설계나 환경정보를 표현할 수 있다. 이러한 목적을 위해 ATM의 모드는 계산모드, 추상화 모드, 주제 모드로 구분된다. 계산 모드는 코드 상에서의 논리와 계산을 나타내며 추상화 모드는 모드와 전이의 블록을 하나의 ATM으로써 표현한다. 대개의 경우, 이것은 코드 상에서의 블록을 ATM내 하나의 모드로 나타낼 때 사용한다. 주제 모드는 예외나 주기적 동작 등과 같은 다수의 ATM의 주제를 표현한다. 실시간 시스템을 검증하기 위해 시스템의 소스 코드는 역명세 과정을 통하여 ATM으로 표현된다. 검증은 ATM에 대한 도달성 그래프를 생성하는 것에 의해 수행된다. 도달성 그래프는 상태와 시간을 추상화되고 압축된 형태로 표현할 수 있으며 그 결과 시간 속성을 지닌 상태 공간을 감소시킬 수 있다. 또한 시스템의 교착상태를 쉽게 발견할 수 있다. 본 논문은 ATM과 실행 모델, 도달성 그래프, 검증을 위한 속성 등을 기술하며 이들을 다른 정형 방법들과 예제를 통하여 비교한다.

Abstract This paper presents a method to verify the reachability property of real-time systems in abstract timed machine (ATM). ATM is a new formal method to specify, analyze and verify real-time systems. ATM consists of modes, transitions, and ports. A mode represents a compressed state of a machine. A transition represents a change from a mode to another mode. A port represents medium for interactions among ATMs. Comparing with other methods of real time systems, ATM is designed for software round-trip engineering, which is one of the distinctive features of ATM comparing with other method. In the reverse engineering perspective, the design and environmental information, as well as the computational logic, in the existing source codes of real-time systems can be represented in ATM. For this purpose, mode is classified into computational, abstract, and subject. The computational mode is to represent the logic and computation in the codes. The abstract mode is to represent a sub-ATM as mode by abstraction. Subject mode is to represent a number of subject issues of ATM, i.e., exceptions, periodic behaviors, etc. To verify real-time systems, the source codes of the system should be represented in ATMs through reverse specification. The verification is performed by generating a reachability graph from ATMs. The main advantage of the reachability graph is that the abstraction and compression of states and time, results in the considerable amount of reduction in timed state space. In addition, deadlock can be found effectively. The paper presents the notions of ATM, execution model, reachability graph, properties to be verified, etc. Further these notions are compared with other methods with examples.

· 본 연구는 한국과학재단 특정기초연구(1999-2-303-003-3) 지원으로 수행되었음

† 비 회 원 : 전북대학교 전산통계학과
jypark@cs.chonbuk.ac.kr
†† 종신회원 : 전북대학교 전자정보공학부 교수
mklee@cs.chonbuk.ac.kr

논문접수 : 2000년 4월 7일
심사완료 : 2001년 1월 3일

1. 서 론

실시간 시스템은 매우 규모가 방대하며, 병렬성, 통신, 동기/비동기, 입출력과 같은 다양한 속성을 지니고 있어 복잡도가 일반 소프트웨어에 비해 높다. 이러한 속성 이외에도 분산성, 형상성 등과 같은 환경적 성질 또한 실시간 소프트웨어에 중요한 요소로 작용한다. 실시간 시스템이 가진 속성과 환경, 복잡도로 인해 실시간 시스템을 명세할 경우 일반 시스템에 비해 많은 상태가 발생한다. 정형 기법들[1, 2, 3, 4, 5]이 실시간 시스템을 명세하기 위해 개발 되었으나 실시간 시스템의 크기와 복잡성을 고려해 볼 때 역공학 과정에서 이러한 정형 기법들을 통하여 시스템의 동적행위를 이해하는 것은 매우 어렵다. 이는 기존의 정형방법들이 순공학 과정에서 시스템을 설계하거나 명세하기 위하여 고안되었으며 특정 물리적 환경에서 실행되는 특정 실시간 소프트웨어의 동적 행위를 완전하게 표현할 수 없기 때문이다. 이와 같이 순공학 과정의 제한된 영역에서만 적용 가능한 정형기법을 사용하여 역공학 측면에서 수백만 줄의 코드를 가진 소프트웨어를 표현하기에는 부적합하다. 따라서, 소프트웨어의 물리적 실행 환경에서 소프트웨어의 여러 동적 행위를 표현하며 매우 방대한 크기의 소프트웨어를 표현하고 관리할 수 있는 속성을 지닌 새로운 정형기법을 필요로 하게 되었다.

ATM은 순환공학과정에서 실시간 시스템을 명세하기 위해 개발된 정형기법이다. 순공학과정에서는 요구사항을 분석하여 ATM을 통한 시스템의 명세가 이루어지며, 역공학과정에서는 기존의 소스 코드로부터 역으로 시스템의 ATM 명세를 생성하게된다. 본 논문에서는 순공학과 역공학 과정에서 생성되는 실시간 시스템의 명세인 ATM을 검증하는 방법을 기술한다. 그림 1은 순환공학과정에서 ATM과 이의 검증 관계를 나타내는 그림으로 점선으로 된 부분이 본 논문의 연구 영역을 나타낸다.

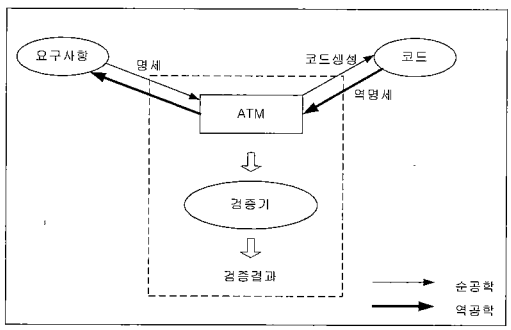


그림 1 순환공학에서 ATM과 검증 관계

시스템의 검증은 도달성 그래프를 생성함으로써 이루어진다. 생성되는 도달성 그래프는 시스템의 모든 동작을 고려하여 시스템 각각의 상태를 노드로 한다. 도달성 그래프가 가진 노드의 수는 유한하며 노드가 시스템의 전이과정에서 발생하는 상태를 표현하기 때문에 도달성 그래프의 노드와 에지를 분석함으로써 시스템의 모든 동작은 초기 상태로부터 추적가능하다.

다양한 속성과 방대한 규모를 특징으로 하는 실시간 시스템의 발생가능한 상태 공간은 일반 시스템에 비해 매우 크다. 그러므로 도달성 그래프 생성 시 발생하는 상태를 효율적으로 감소시키는 것이 검증의 주요 문제 중의 하나이다.

정형기법에 대한 도달성 그래프를 통한 검증은 다른 정형기법에서도 유사하게 고안되었으나 ATM을 이용한 명세에 대한 도달성 그래프는 검증에 대한 새로운 생성 조건을 제공함으로써 다른 정형기법에서 고안된 방법에 비해 많은 상태를 감소시킬 수 있다는 장점이 있다. 이는 ATM의 모드가 압축된 상태를 표현하기 때문에 하나의 모드에 포함된 여러 상태를 한꺼번에 고려하게 됨으로써 가능하다. 또한 그래프 생성 과정에서 시간의 이산적 증가에 대한 상태 생성에 조건을 둠으로써 이루어진다.

또한 검증 과정에서 ATM의 기술상의 특징으로 인해 전체 시스템이 최종 상태로 도달 가능한지 쉽게 파악할 수 있으며 이에 따라 최종 상태로 도달할 수 없는 교착상태(deadlock) 또는 무한실행(livelock) 등의 상태를 발견할 수 있다. ATM에 대한 도달성 그래프를 통하여 기존의 검증 방법보다 적은 상태 공간을 고려하며 시스템의 검증을 수행할 수 있다.

본 논문의 구성은 2절에서 정형기법의 검증과 관련된 다른 연구와의 비교, 3절에서는 실시간 시스템을 명세하기 위해 새롭게 고안된 ATM에 대해 간략히 기술하며 이에 대한 실제 PBC 예제를 보인다. 4절에서는 시스템을 검증하기 위한 도달성 그래프를 정의하고 이를 생성하는 알고리즘을 기술하며 도달성 그래프에 대한 다른 연구에서의 결과와 비교, 분석한다. 5절은 ATM과 이에 따른 검증에 대한 현재의 연구 진행 상황을 설명하고 6절에서는 결론 및 향후 연구에 대하여 기술한다.

2. 관련연구

시스템의 다양한 요구사항과 많은 제약을 가진 실시

간 시스템을 명세하는 정형기법들이 개발되었으며 이들 정형기법들이 명세한 시스템을 검증하기 위한 여러 방법론이 고안되었다. 이의 대표적인 것으로 CSM (*Communicating State Machines*)과 이에 대한 상태 최소화(*State Minimization*) 방법[3], CRSM(*Communicating Real-Time State Machines*)[6]과 이에 대한 도달성 그래프(*Reachability Graph*) 방법[5]이 있다.

상태 최소화 방법은 시스템의 전이 가능한 모든 상태를 명확하게 도달가능한 상태로 나누어가면서 전이되는 시스템의 상태를 중복되지 않게 표현하는 방법이다. 그러나 CSM이 시간 속성을 표현하지 못함으로 인해 상태 최소화 방법은 시간 개념을 가진 실시간 시스템을 검증하기에 어려움이 있다. 상태 최소화 방법은 시간 속성에 대한 검증 방법을 정의하지 않았으므로 시간 제약에 따른 검증을 수행하기 어렵다. 또한 상태 최소화 방법은 실제 전이와 관련 없는 여러 변수 값을 시스템의 검증을 위한 상태에 포함함으로써 실제 검증 과정에서 고려하지 않아도 되는 부가적인 상태를 유발한다. 이러한 방법은 시스템이 가진 모든 변수를 표현해야 하므로 매우 많은 상태의 증가를 피할 수 없으며 이에 따른 검증 과정의 오버헤드도 피할 수 없게 된다.

CRSM에 대한 도달성 그래프를 통한 검증방법은 시간 속성을 지닌 실시간 시스템을 검증하기 위한 방법을 제시하였다. 이 방법은 시간이 흐름에 따라 시스템의 전이에 영향을 주지 않는 시간 값을 특정 기호로 정의하여 고려하지 않아도 되는 시간 값에 대한 시스템의 상태를 처리하는 방법을 제안함으로써 무한한 도달성 그래프를 유한한 도달성 그래프로 표현하는 방법을 제시하여 시스템의 검증을 가능하도록 하였다. CRSM의 도달성 그래프는 CSM에 대한 상태 최소화 방법과 같이 모든 변수에 대해 시스템의 상태를 증가시킴으로 검증 과정에서 검증이 불필요한 상태를 생성시켜 검증의 상태 공간을 증가시킨다. 또한 CRSM에 대한 도달성 그래프에서의 상태는 시간의 이산적 변화를 민감하게 고려하여 상태를 증가시킴으로써 전이 가능한 시간 구간 안에서 조차도 이산적 시간 변화에 따른 많은 상태를 유발한다. 즉, 같은 전이 가능 상태에 있으며 다른 시간 값을 가진 상태를 분리하여 표현함으로써 시간의 변화에 따른 많은 상태의 증가를 초래한다.

Petri Net[7]에 대한 도달성 그래프는 Petri Net이 추상화를 지원하지 않기 때문에 추상화를 고려하지 않은 명세를 바탕으로 생성되게 된다. 따라서, 도달성 그래프 생성 과정에서 추상화되어 생성될 수 있는 모든 상태를 고려해야 하므로 상대적으로 많은 상태를 발생시킨다.

CSM에 대한 상태 최소화 방법과 CRSM에 대한 도달성 그래프 방법이 가진 이러한 단점을 극복하면서 실시간 시스템에 보다 적합한 검증 방법이 요구되며 이러한 요구사항을 만족시키기 위해 새로운 생성조건을 가진 도달성 그래프를 통한 검증방법을 고안하였다.

3. ATM

ATM은 계층성, 분산성, 실시간성, 우선권, 다수에 의한 동기화, 예외처리, 실행성 등 다양한 속성을 가진 실시간 시스템을 순환공학 과정에서 명세하기 위해 개발된 정형 기법이다.

ATM은 모드(mode)의 집합, 가드(guard)된 전이의 집합, 포트(port), 실행 시작점과 실행의 종료점으로 구성된 머신 단위로 표현된다.

ATM은 1) 해석(Interpretation), 2) 재구성(Restructuring), 3) 명세언어로의 표현의 세 단계를 거쳐 원시 소프트웨어를 명세한다. 해석 단계에서는 소스 코드로부터의 프로그램 블록 단위에 따른 머신과 머신 내의 모드를 포함하는 타입 ATM이 생성되고 프로그램 단위가 가진 구조적 정보가 각 머신 별로 생성된다. 재구성 단계에서는 시스템의 동적 행위에 따라 타입 ATM의 재구성이 이루어진다. 명세언어로의 표현 단계에서는 타입 ATM이 실제 실행 관계에서 어떻게 생성되며 전체적으로 어떤 구조정보를 갖는지 명세한다.

3.1절에서는 각 구성 요소의 정의를 하고 3.2절에서는 ATM이 가진 특징을 기술한다. 3.3절에서는 ATM의 정의된 속성을 설명하고, 3.4절에서는 ATM을 이용한 명세의 예를 보인다.

3.1 ATM 구성 요소 정의

3.1.1 머신

머신은 활성화되었을 때 내부의 흐름을 스스로 제어할 수 있다. 일반적으로 태스크, 프로시져와 같은 독립적 프로그램 블록 단위로 명세된다. 각 머신은 다음과 같은 레이블을 갖는다. 레이블 구성 요소 중 괄호 부분은 생략가능하다.

```
(<caller_name>>)(<container_name::>)
 <machine_name>(<parameter_list>)
(<:machine_type>)(<# priority_number>)
```

*<caller_name>*은 프로시져나 함수를 나타내는 머신일 경우 해당 머신을 호출하는 호출 머신의 이름, *<container_name>*은 추상화 모드에 의해 표현된 머신일 경우 해당 추상화 모드를 포함하는 머신의 이름,

<*machine_name*>은 해당 머신의 이름, <*parameter_list*>는 generic과 같은 경우의 머신을 인스턴스 하거나 호출 할 때의 파라미터 리스트, <*machine_type*>은 해당 머신의 실제 타입을 말한다. <*#priority_number*>는 머신이 가진 우선순위를 나타낸다.

3.1.2 모드

ATM의 모드는 계산 모드(*computational mode*), 추상화 모드(*abstract mode*), 주제 모드 (*subject mode*)의 세 종류가 있다. 계산 모드는 머신내의 제어와 흐름을 기준으로 전이와 관련있는 이벤트와 조건을 제외한 변수의 집합, 실행문의 집합을 내부적으로 직접 포함하고 있다. 추상화 모드는 소프트웨어의 계층관계를 표현하기 위한 방법으로 구조상 하위 계층에 속하는 기능을 상위 계층에서 추상화시키기 위한 모드이다. 머신 내에서 추상화 모드로 제어가 이동하는 즉시 해당하는 하위 단계의 머신이 동작을 시작하며 종료하는 즉시 상위 단계의 추상화 모드로 제어가 복귀하여 계속 진행된다. 주제 모드는 소프트웨어의 특정 기능을 수행하고 있는 상태들을 포함하는 모드이다. 주제 모드의 예로는 예외 처리나 주기적 동작과 같은 다수의 ATM의 주제를 표현한다. 모드는 이름과 아이디, 시간 제약 등을 다른 모드와 구별되는 속성으로 갖는다.

3.1.3 전이

전이는 모드에서 모드로의 제어의 이동을 나타내는 것으로 조건이나 이벤트로 구성된다. ATM에서의 전이의 종류는 일반 전이와 추상화 전이로 나뉘어진다. 일반 전이는 이벤트, 조건, 시간 제약을 레이블로 가지며 일반적인 제어의 흐름을 나타낸다. 레이블이 없는 전이는 머신의 즉각적인 전이를 나타낸다. 추상화 전이는 추상화 모드와 이를 상세화한 머신 사이의 전이를 나타낸다. 추상화 전이는 레이블이 없으며 상위 단계의 머신에서 하위 단계의 머신으로 향하는 점선으로 된 화살표로 표현된다. 전이의 레이블은 다음과 같은 형태로 구성된다.

```
(<machine_name.>)<port_name>
<event_flow_type>(parameter_list>
([<time_bound>])
```

<*machine_name*>은 머신의 이름을 의미하며 자신의 포트를 가리킬 경우에는 사용이 선택적이다. <*port_name*>은 포트의 이름을 말하며 <*event_flow_type*>은 이벤트의 방향으로 전송과 수신의 경우 CSP의 표기와 유사하게 출력은 '!', 입력은 '?'이며 CSP[2]에서 확장하여 입출력은 RPC(*Remote Procedure Call*)[1]과 같이 '!?'로 표기한다. <*parameter_list*>는 전달되는 데이타의 리스트로 Ada 프로그램과 유사하게 데이타 x에 대하여 *in* 형태는 x, *out* 형태는 $\bar{x}$, *in-out* 형태는 $\bar{\bar{x}}$로 표기한다. 따라서 상대 머신으로부터 데이타를 입력 받을 경우는 *in* 상태로, 상대 머신으로부터 데이타를 송신할 경우에는 *out* 상태로, 데이타를 입력과 출력을 위해서는 *in-out* 상태로 표현한다. <*time_bound*>는 이벤트가 실행 될 때 최소와 최대 시간 경계 값이나 실행의 지속시간으로 실시간 요구사항을 표현한다.

3.1.4 포트

ATM의 포트는 ATM이 다른 정형 기법과 차이가 있는 것 중의 하나로 머신 사이의 시그널 또는 데이타를 송, 수신하는 곳이며 수신하는 머신에 명시적으로 표시한다. 포트를 통하여 머신 간에 발생하는 통신의 종류를 명시적으로 명세한다. 포트의 종류로는 활성화 포트(Activation Port), 엔트리 포트(Entry Port), 대체 포트(Substitution port)가 있다:

1) 활성화 포트 : 머신을 활성화 또는 비활성화 시키는 시그널을 위한 포트.,

2) 엔트리 포트 : 통신을 위해 머신 간에 데이타를 전송하는 포트,

3) 대체 포트 : 한 머신의 대응하는 다른 머신에 대체됨을 의미하는 대체 이벤트 호출을 위한 포트로 프로시져나 함수를 나타내는 머신에 시그널이나 파라미터 리스트를 전송하는 포트.

이와 같이 시그널에 따른 포트를 정의함으로서 발생하는 이벤트의 유형을 명확하게 분석할수 있게 하여 소프트웨어의 행위적 측면을 보다 쉽게 이해할 수 있다.

지금까지 ATM의 구성 요소들에 대하여 기술하였다. 그림 2는 이들 구성요소에 대한 표기형태를 보여준다.

비 정형적으로 기술한 ATM을 보다 정형적으로 기술하면 다음과 같다.

Definition 3.1 <*ATM*> ATM 머신은 M = <M_{CAS}, S, F, T, P>의 5-튜플로 정의된다.

여기에서

- M_{CAS}는 모드의 집합을 말한다. 모드는 계산 모드 C와 추상화 모드 A, 주제 모드 S로 구성된다. CAS=<N, St, R>로 각 모드는 이름 N과 문장 St과 시간제약 R을 가질 수 있다. 시간에 대한 제약 R은 다시 준비 시간, 주기, 실행 시간, 데드라인의 속성을 갖게 된다.
- S는 머신의 시작점이며,
- F는 머신의 종료점을 말한다.
- P는 머신과 머신 간의 메시지나 데이타의 송수신이 일어나는 포트로 활성화 포트, 엔트리 포트, 대체 포

트로 구성된다.

· $T=M_{CAS}\times Co\times M_{CAS}$는 모드와 모드간의 혹은 모드와 머신 간의 전이의 집합이다. 전이는 레이블 $L = < Co, E, R>$을 갖는다. 레이블에서 Co는 조건(condition), E는 이벤트(event), R은 전이의 제약 시간(real time)을 나타낸다.

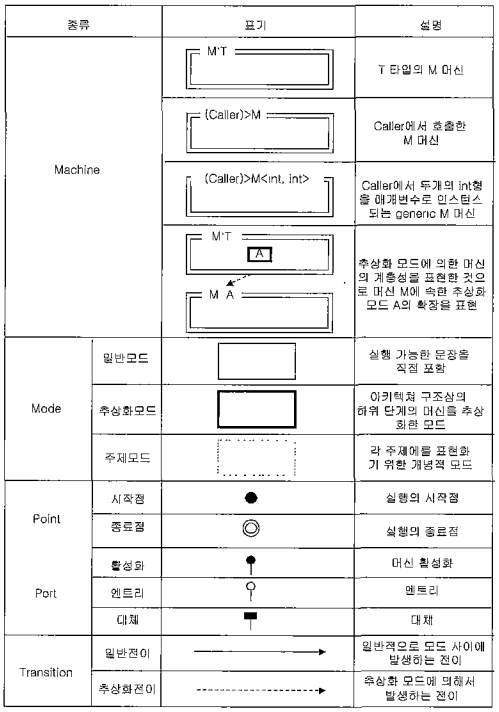


그림 2 ATM의 각 구성 요소 표기

3.2 ATM 특징

ATM은 추상화 모드를 이용하여 소프트웨어의 아키텍쳐를 계층적으로 표현한다[8]. 아키텍쳐의 각 단계의 머신은 추상화 모드를 이용하여 하위 단계 머신의 동작을 포함할 수 있다. 이러한 아키텍쳐 구조는 시스템의 동적 행위를 계층적으로 표현하여 실행의 특정 단계에서의 상태와 행위를 예상을 용이하게 하고 각 머신의 호출관계 및 인스턴스 관계를 명시적으로 보여줌으로써 시스템 전체의 실행을 추적할 수 있다.

ATM의 모드는 전이에 해당하는 이벤트나 컨디션이 발생하기 전까지의 상태들을 포함하므로 기존의 상태도에서 예상되는 상태폭발(state explosion)을 줄일 수 있다[8]. 모드가 하나 이상의 상태를 포함한다는 것은 시스템 검증 시 검증해야 할 상태가 감소한다는 것을 의미한다. 예를 들어 시스템의 진행과 시간의 흐름을 기준으로 한 상태 공간(states space) 상에서 시스템을 검증하는 경우, 특정 모드로 진입 했을 때 시스템이 진행 할 시스템의 방향을 상태의 집합으로 예상함으로써 검증의 많은 오버헤드를 줄일 수 있을 것이다.

또한, ATM은 클래스 또는 타입과 인스턴스의 객체지향 파라다임을 사용한다. 명세 언어로의 표현시 해석 단계에서 생성된 타입 ATM을 인스턴스화 시켜 사용한다. 이러한 인스턴스 개념은 같은 타입에 대해 여러 개의 머신이 병렬적으로 실행 될 경우 타입에 이름을 주어 인스턴스 시키는 과정만을 거치면 되므로 명세의 시뮬레이션 등에서 중복된 작업을 감소시킨다.

3.3 ATM 속성

ATM은 실시간 시스템이 가진 다양한 속성을 명세하기 위해 각 속성에 따른 표기와 방법을 정의한다. 이러한 속성으로는 병렬성, 구조성, 시간, 통신, 자원경쟁-우선순위, 예외처리 등이 있다.

ATM에서 모든 머신은 병렬적으로 수행된다. 따라서 소프트웨어의 태스크 또는 함수, 프로시져 단위를 나타내는 ATM이 병렬적으로 활성화 되거나 대체될 경우 이들 ATM은 병렬 수행된다.

소프트웨어가 계층 관계의 구조에 있거나 평면 구조에 있거나 역공학 과정에서 정형 기법은 이들 구조적 정보를 효과적이고 분명하게 표현할 수 있어야 한다. ATM은 원시 소프트웨어가 가진 구조적 정보를 해석 단계에서 프로그램 단위별 내부 구조를 파악한 후 재구성 단계와 명세 언어로의 표현 단계를 거쳐 전체 소프트웨어의 실행상의 계층 관계 정보를 제공한다.

실시간 시스템은 시스템의 특성 상 시간에 대한 제약이나 요구사항을 가진다. 시스템의 시간은 두 종류로 나눌 수 있다. 실행이 발생해야 하는 제한된 시간을 나타내는 것(interval time)과, 일정한 실행이 지속되는 시간을 나타내는 것(duration time)이다.

```
M.Send_Receive!?(x)[0,5]
Wait[4]
```

그림 3 시간 제약의 예

그림 3의 첫 번째 예는 전이에서의 시간 제약을 가진 메시지전달을 보여준다. 이 전이에서 메시지는 5단위 시간 이내에 이루어져야 한다. 이는 제한된 시간 내에 실행에 대한 예이다. 그림 3의 두 번째 예는 일정한 실행이 지속되는 시간 제약의 예로 4단위 시간 동안 wait라

는 실행을 지속해야 한다. ATM은 전역 타이머와 지역 타이머를 가지고 있어 시간에 대한 제약을 타이머의 시간에 따라 수행한다.

ATM의 통신은 CRSM으로부터 확장된 방법을 사용한다. 송신의 기호는 '!'로, 수신의 기호는 '?'로 표현한다. 여기에 확장된 개념이 RPC(Remote Procedure Call)와 같이 동기적 통신을 표현하기 위해 '!?'를 사용한다는 것이다. '!?'의 의미는 어떠한 시그널이나 이벤트를 상대 머신에 보내고, 이에 따른 대답이 수신될 때까지 기다림을 의미한다. 이렇게 함으로써 실시간 시스템에서 중요한 속성인 동기적 통신을 표현가능 하다. 또한 비동기적인 통신을 나타낼 경우, 단지 메시지만을 보내고, 다음 모드로의 전이를 수행하게 함으로써 비동기적 통신을 표현할 수 있다.

```
procedure PBC;

procedure PBC is
  protected Buffer is
    entry Read (C : out Character);
    entry Write(C : in  Character);
  private
    Pool     : String(1..100);
    Count    : Natual := 0;
    In_Index  : Positive := 1;
    Out_Index : Positive := 1;
  end Buffer;

  protected body Buffer is
    entry Write(C ; in Character)
      when Count < Pool'Length is
    begin
      Pool(In_Index) := C;
      In_Index:=( In_Index mod
                    Pool'Length) + 1;
      Count := Count + 1;
    end Write;

    entry Read(C : out Character)
      when Count > 0 is
    begin
      C := Pool(Out_Index);
      Out_Index:=(Out_Index mod
                    Pool'Length) + 1;
      Count := Count - 1;
    end Read;
  end Buffer;

  task type Producer_Type;
  task body Producer_Type is
    Msg : character;
  Begin
    Loop
      get(Msg);
      select
        Buffer.Write(Msg);
        Exit when Msg = ASCII.EOT;
      or
        delay 5.00;
      end select;
    end while;
  end Producer_Type;

  task type Consumer_Type;
  task body Consumer_Type is
    Msg : Character;
  Begin
    Loop
      Select
        Buffer.Read(Msg);
        Put(Msg);
        Exit when Msg = ASCII.EOT;

      or
        delay 5.00;
      end select;
    end while;
     end Consumer_Type;

type Producer_Ptr is access Producer_Type;
type Consumer_Ptr is access Consumer_Type;

  Producer : Producer_Ptr;
  Consumer : Consumer_Ptr;

Begin
  Select
    delay 10.00;
  then abort
    Producer := new Producer_Type;
  end select;

  select
    delay 10.00;
  then abort
    Consumer := new Consumer_Type;
  end select;
end PBC;
```

그림 4 Producer-Buffer-Consumer Ada 코드

통신을 위한 시그널과 이벤트, 데이타는 각 머신이 가진 포트를 통하여 이루어진다. 한 머신이 다른 머신의 포트를 통해 시그널과 이벤트를 보내고, 상대 머신이 해당 포트를 통해 시그널과 이벤트를 수신할 때에 통신이 이루어진다. 통신을 위한 시그널이나 이벤트는 전이상에 레이블로 표현되며 이 레이블에는 통신의 방향성과 데이타의 방향성, 그리고 통신이 이루어지는 포트가 명시적으로 표현된다. 그림2의 첫번째 예가 전이의 레이블을 보여준다. M이름을 가진 머신의 Send_Receive 포트에 동기적 시그널을 보내며 이에 대한 결과로 x에 값을 전달 받는다.

이 밖에도 ATM은 자원경쟁, 우선순위, 예외처리를 정의하고 있으며 이에 대한 자세한 기술은 [8]에 기술되어있다.

3.4 ATM을 이용한 명세 예제

지금까지 기술한 ATM의 정의를 이용한 실제 예제를 살펴본다. 그림 4는 Ada 코드로 Producer-Buffer-Consumer(PBC)의 실행을 나타내는 코드이다. PBC 예제는 PBC 프로시져로 이루어져 있으며 여기에는 protected 타입의 Buffer와 Producer_Type, Consumer_Type을 나타내는 2개의 태스크가 있다. 이들 태스크는 10 단위 시간을 두고 각각 Producer와 Consumer로 활성화 되어 작업이 이루어진다. 만약 10 단위 시간 내에 활성화되지 않는다면 Producer와 Consumer의 생성은 실패한다. Buffer는 두 개의 엔트리 Write와 Read를 가지며 이 엔트리를 통하여 데이타가 전달된다. Producer와 Consumer는 활성화된 후에 실제 실행에서 각각 Buffer의 Write와 Read 엔트리에 5 단위 시간 내에 데이타를 쓰고, 읽는 작업을 수행한다. 만약 이 작업이 시간 내에 이루어지지 않는다면 다시 쓰고, 읽기 위한 작업이 반복된다. 이 코드에서 Buffer는 제한된 용량을 가진 것으로 Producer는 실제로 Buffer에 데이타를 기록할 때 제한된 용량까지만 기록가능하고 Consumer는 Buffer에 기록된 데이타가 존재할 경우에만 데이타를 읽어 들일 수 있다. 따라서, Producer는 데이타를 Buffer에 기록하려 할 경우, Buffer의 저장 공간이 남지 않았을 때에는 저장 가능하게 될 때까지 기다린다. 마찬가지로, Consumer 또한 만약 Buffer에 아무런 데이타도 존재하지 않을 경우, 데이타가 기록될 때까지 기다리게 된다. 이 PBC 예제는 상호 배제의 성질을 가진 Buffer를 통해 Producer와 Consumer가 단방향으로 간접 통신하는 것을 보여주는 코드이다.

PBC 코드는 먼저 해석 단계와 재구성 단계에서 1차적으로 타입 ATM을 생성한다. 이는 앞 절에서 기술한 것과 같이 각 타입과 함수, 프로시져 단위 또는 그 안에서의 추상화 관계를 가진 타입 ATM을 생성하게 된다. 그림5, 6, 7은 PBC의 타입 ATM을 나타낸다.

원시 코드로부터 생성된 타입 ATM은 원시 코드의 타입 구문, 프로시져, 함수 구문의 내부적 제어의 흐름을 나타내며, 내부적 계층 관계를 표현한다. 그림 5의

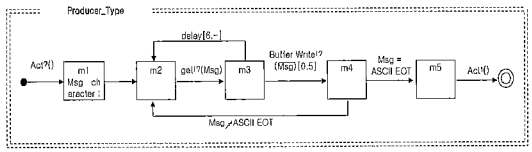


그림 5 태스크 Producer_Type에 대한 타입 ATM

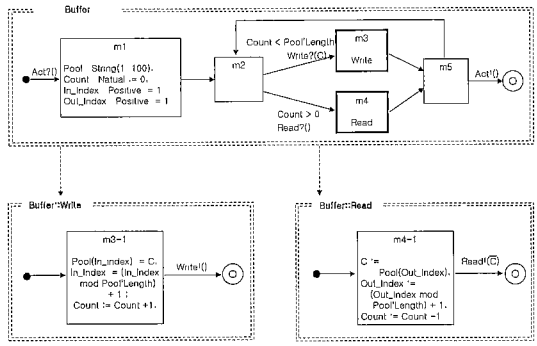


그림 6 Protected Buffer에 대한 타입 ATM

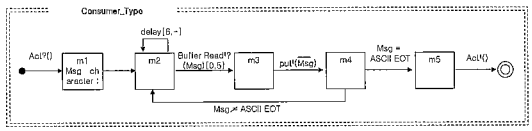


그림 7 태스크 Consumer_Type에 대한 타입 ATM

Buffer의 타입 ATM을 보면 코드상에서 발생한 제어의 흐름과 내부적 추상화 관계도 표현한다.

해석 단계와 재구성 단계에서 생성된 타입 ATM은 실제 명세 생성 단계에서 프로시져의 호출관계나 타입의 선언 관계에 따라 복제되어 인스턴스된다. 인스턴스된 ATM은 ATM의 명명 규칙에 따라 해당하는 머신의 이름을 가진다. 그림 8, 9, 10, 11은 PBC, Buffer, Producer, Consumer의 인스턴스 ATM을 보여준다.

인스턴스된 ATM은 복제 생성 되었기 때문에 내부적 제어 전이 관계와 구조적 관계가 타입 ATM과 같다. 그러나 인스턴스 ATM은 타입 ATM과 달리 해당하는

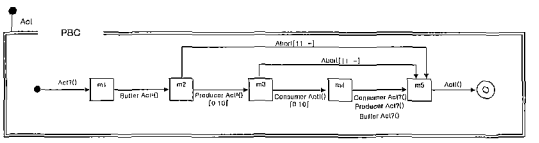


그림 8 PBC ATM

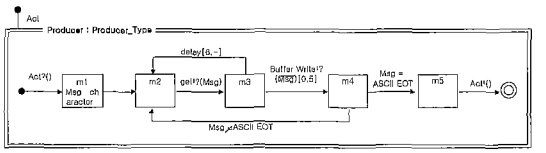


그림 9 Producer ATM

포트를 가지고 있다. 이들은 포트를 통하여 활성화 되거나 통신을 한다. 포트의 존재 유무가 타입 ATM과 인스턴스 ATM의 차이이며 머신을 결정하는 사각형의 점선, 실선의 표기 또한 타입 ATM과 인스턴스 ATM을 구분지어 준다.

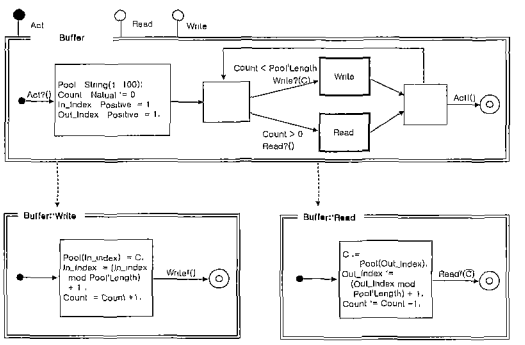


그림 10 Buffer ATM

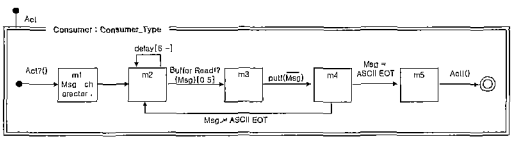


그림 11 Consumer ATM

4. 도달성 그래프

지금까지 ATM의 정의와 특징, 명세 예를 살펴보았다. 4절에서는 ATM을 이용하여 명세한 시스템의 검증을 수행하는 도달성 그래프를 정의하고 특징과 예제를 기술한다.

4.1 정의

도달성 그래프는 시스템의 실제 동작을 표현하는 실행 모델이다. ATM으로 명세된 시스템의 실행을 모델링 함으로써 실제 실행실 발생할 수 있는 결과와 각종 분석을 수행할 수 있다. 도달성 그래프는 병렬로 실행되거나 혹은 순차적으로 실행되는 시스템의 도달가능한 상태를 파악, 노드와 에지로써 표현하여 시스템을 검증, 분석하기위한 모델로써 사용한다.

Definition 4.1.1 <*Reachability Graph*> 주어진 일련의 ATM 머신의 집합 $M=\{M_1, M_2,..., M_n\}$, $n \geq 1$, 의 도달성 그래프 $G=<N, n_0, E>$의 3-튜플이다.

여기에서

· N은 도달성 그래프의 유한한 노드의 집합으로 $N=<M, V, T>$ 형태의 3-튜플로 정의된다. M은 관련된 ATM의 모드의 유한 집합이며 V는 전이와 관련된 변수 값의 집합, T는 전이와 관련된 시간 값의 집합이다.

· $n_0 \in N$는 각 ATM 머신의 초기 모드의 조합과 각 머신의 전이 관련 변수와 시간의 초기 값을 갖는 그래프의 시작 노드이다.

· $E = N \times C \times N$ 는 유한한 에지의 집합으로 C는 N에서 발생가능한 전이들의 조합이다. 에지는 C를 이루는 전이 중 발생 가능한 이벤트의 내용과 시간을 레이블로 갖는다.

노드는 각 ATM의 전이에 따른 시스템의 모드, 시간, 전이에 관련된 변수의 값을 원소로 하는 튜플 형태로 표현된다. 노드는 검증과 분석을 위해 도달성 그래프 생성에 관련된 머신을 구분하기 위해 구분자 '{', '}'와 모드의 진입에 따른 시간과 값의 변화를 나타내는 기호 '[', ']'를 사용한다.

에지는 ATM의 C의 조합 중 발생가능한 전이에 따라 생성되는 노드간의 이동을 의미한다. 이때 에지의 레이블은 C에 조합된 전이에 따라 결정되는데 조합된 전이가 중 전이 가능한 것을 레이블로 가진다.

그림 12은 노드의 형식을 보여준다. 예제 노드는 두 개의 ATM이 관련되어 있으며 m1과 n1은 각 ATM의 현재 모드를 나타낸다. x_f, x_t는 전이와 관련있는 변수로 xf는 모드로 진입할 때의 x 값을, x_t은 모드를 벗어날 때의 값을 나타낸다. 마찬가지로, t_f은 모드로 진입한 시간을 나타내고, t_t은 모드에 머무른 시간을 의미한다. 시간을 나타내는 값은 숫자 또는 이 된다. 은 전이가 발생하여 모드에 진입했을 경우 모드의 다음 전이가 시간과 관계 없는 경우에 표기되는 값이다. 만약 모드에 진입했을 경우 모드의 다음 전이가 시간 제약을 가졌다면 t_f은 0값으로 설정된다. 만약 전이가 t_t과 관련없이 진행되었고 이 아니라면 x_t은 모드에 머무른 만큼의 시간 값을

$<\{m1,[x_l, x_l],\ [t_l, t_l]\},\{n1,\ [t_l, t_l]\}>$

그림 12 도달성 그래프의 노드

가진다.

4.2 ATM 검증을 위한 도달성 그래프의 특징

실시간 시스템은 일반 시스템에 비해 상태 공간이 크다. 따라서, 시스템 검증 시에 상태 공간을 감소시키는 것이 중요한 문제이다. ATM의 도달성 그래프의 특징은 이러한 상태 공간을 감소하기 위해 생성조건 사항을 추가하였다는 것이다.

3절에서 기술한 바와같이 ATM의 모드 특성으로 인하여 다른 정형 기법에 비해 많은 상태 공간을 감소시켜 명세 과정에서 보다 적은 상태 공간을 표현하였다. 따라서 이를 표현하는 도달성 그래프도 다른 정형 기법을 위한 방법보다 적은 노드로 시스템을 검증한다. 이에 추가하여 실시간 시스템의 검증을 위한 상태 공간의 감소를 위해 본 논문의 도달성 그래프는 두 가지 생성조건을 가진다. 첫번째 생성조건은 도달성 그래프 노드의 한 구성 원소인 변수를 전이와 관련있는 변수로 제한하는 것이다. 2절에서도 언급하였듯이 CSM과 CRSM의 검증 방법은 모든 변수 값을 고려한다. 따라서 실제 전이와 관련없는 변수에 대한 노드가 생성되며 이에 따른 상태 공간의 증가는 막대하게 된다. ATM의 도달성 그래프에서의 노드를 이루는 변수를 전이와 관련된 것으로만 제한함으로써 상태 공간을 감소시킨다. 두 번째 조건은 시간에 관한 것이다. 한 노드가 생성되어 있고 다른 노드가 생성될 경우 시간을 제외한 다른 원소의 값이 같고 두 노드의 시간 값이 전이 가능한 시간 안에 존재하는 경우 두 노드를 같은 노드로 취급 함으로써 발생하는 노드의 수를 줄이는 것이다. 즉, 두 노드를 같은 상태 공간의 것으로 취급하는 것이다. 그림 13은 이러한 예를 보여준다. 발생 가능한 상태 공간은 변수 x, y 값의 변화와 시간 t의 변화 공간이다. 전이가 시간 $t1$, $t2$와 y값에 의해 결정되며 x값의 변화는 전이에 영향을 주지 않을 때 y값이 같고 노드의 시간 값이 $t1$과 $t2$ 안에 있다면 이의 상태는 빗금친 부분이며 사각형 내의 어떠한 값도 같은 상태로서 취급될 수 있다. 달리 말하면, 빗금친 부분은 같은 값으로 취급될 수 있는 상태 공간이다. 따라서 시스템의 상태 값이 사각형 내의 어느 점이라도 같은 상태 공간으로 보며 그 결과 검증을 위한 상태를 감소 시킬 수 있다.

시간에 관한 또 다른 상태 감소 요인은 노드의 시간 표기에서 발생한다. 시간과 관련없는 전이에 대한 노드

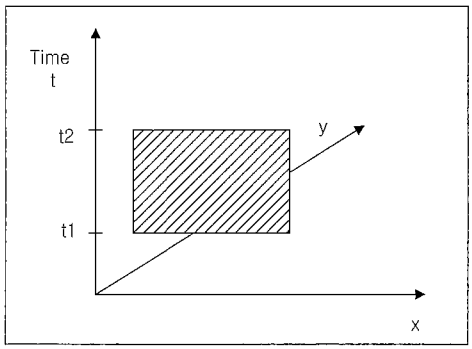


그림 13 상태 공간에 대한 예

생성시 '-'을 사용함으로써 시간과 관계없는 모드에서 시간의 이산적 변화에 의해 증가되는 상태 공간을 줄일 수 있다. 시간과 관련하여 또 다른 특징은 ATM의 명세 방법에 의해 발생한다. 일반적으로 시간 제약을 명세할 경우 전이상에 전이 가능한 제약만을 레이블로 한다. 그러나 ATM은 PBC 예제의 그림 7, 8의 delay[6,-]이 보여 주듯이 시간 제약을 만족하지 못했을 경우의 전이의 진행을 명시적으로 표현함으로써 다른 정형 기법의 경우에는 추론해야 만이 검증 가능하였던 시스템의 진행 상태를 명시적으로 그래프 내에 표현할 수 있으며 이에 따라 그래프 생성 알고리즘도 간략화 시킬 수 있게 되었다.

도달성 그래프에서 ATM의 명세 방법에 따른 추가적 특징은 ATM의 명세는 분명한 종료점이 명시된다는 것이다. 즉, 관련된 머신이 최종적으로 종료 상태를 가지지 못했을 경우 시스템은 교착상태 혹은 무한실행 상태에 있으며 이의 발견이 쉽게 이루어진다.

4.3 시멘틱(Semantic)

도달성 그래프를 통한 분석을 위한 시맨틱은 도달성 그래프의 실행에 기반한다.

Definition 4.2.1 <*Excution*> 도달성 그래프 $G=<N, n_0, E>$에 의해 표현되는 실행은 노드와 실행 이벤트, 시간에 의해 다음과 같은 시퀀스로 정의된다.

$$n_0 \xrightarrow{e_1,t_1} n_1 \xrightarrow{e_2,t_2} n_2,\ldots,n_i \xrightarrow{e_i,t_i} \ldots,\ n_i \in N,\ \xrightarrow{e_i,t_i} \in E,\ i \geq 1$$

시퀀스의 길이는 유한하거나 무한하다.

실행 시 에지의 레이블은 4.2절에서 언급하였듯이 다음 노드의 시간 제약 여부에 따라 시간 값의 레이블을 갖지 않는 경우도 있다. 실행에서 에지 $\xrightarrow{e_i,t_i} \in E$ 의 레이블 중 t_i는 생략가능하다. 도달성 그래프의 실행 시퀀스를 분석함으로써 실행에 대한 추적이 가능하며 시스템의 도달성과 교착상태, 무한실행 상태 등을 파악한

다. 도달성 그래프를 분석하기 위해 다음과 같은 여러 가지 표기법을 사용한다.

에지 $E_i \in E$에 대해서 *soruce*(E_i)는 전이를 발생시킨 노드, n_{i-1}를 말하며, *target*(E_i)는 전이에 의해 진행하는 다음 노드, n_i를 나타낸다. *event*(E_i)는 해당 에지가 레이블로 갖는 이벤트, e_i를, *time*(E_i)은 에지가 시간에 관한 레이블을 가진 경우 전이가 발생한 시간을 나타낸다. 또한 노드 n_i에 대해서 *true*(n_i)는 n_i에 조합된 각 머신의 노드에서 발생가능한 전이를 나타낸다. 특정 ATM 머신 $M_k \in M$에 대해 *node*(M_k, n_i)는 노드 n_i에 포함된 M_k의 모드를 나타낸다.

실행 모델인 도달성 그래프와 위의 표기법을 사용하여 시스템의 분석이 이루어진다.

Theorem 4.2.1 <*Reachability*> 다음 조건이 만족하면 $M_k \in M$의 모드 m은 도달가능하다.

$$m \in \bigcup_{i=0}^{FN} \{node(M_k, n_i)\}$$, FN은 유한한 노드의 수

proof 특정 모드가 실행시 도달 가능하다면 실행 모델의 노드에 존재한다. *node*(M_k, n_i)는 노드 n_i에 포함된 M_k의 모드를 가리키므로 실행 모델인 도달성 그래프 내의 노드 중 해당 머신의 노드를 선택한 집합중에 m이 포함되어 있다면 m은 도달가능한 모드이다. ■

Theorem 4.2.2 <*Absence of deadlock*> 교착상태는 다음과 같이 정의된다.

도달성 그래프에 관련된 ATM 머신의 유한 집합 $M=\{M_1, M_2, ..., M_n\}$, $n \geq 1$에 대해 각 리프노드의 집합을 NF라고 할 때 NF의 모든 원소 노드인 nf에 대해

$$\bigwedge_{i=0}^{n} node(M_i, nf) = F \quad \text{if} \begin{cases} true & \text{Final node} \\ false & \text{Deadlock} \end{cases}$$

F는 각 머신의 최종 모드이다.

proof 교착 상태는 노드에서 다른 노드로의 전이가 발생할 수 없는 경우이다. 따라서 교착 상태를 발생시킬 수 있는 노드의 후보는 도달성 그래프의 리프 노드들이다. ATM의 경우 교착 상태 분석은 명세 방법의 특징으로 인해 다른 정형 기법보다 수월하다. 모든 ATM 머신은 최종 상태를 포함하고 있으며 이를 통해 도달성 그래프의 리프 노드에 포함된 각 머신의 모드를 파악함으로써 교착 상태를 발견할 수 있다. 즉, 각 리프 노드에 조합된 모든 모드가 F를 나타낼 경우는 정상적인 종료를 나타내며 그 이외의 경우는 교착 상태이다. ■

기타 다른 속성 또한 도달성 그래프의 생성 제약과 ATM의 특징을 통하여 분석 가능하며 도달성 그래프의 실행 시퀀스 분석함으로써 시스템의 행위를 파악한다.

4.3 도달성 그래프 생성 알고리즘

ATM의 도달성 그래프 생성 알고리즘은 CRSM의 도달성 그래프 생성 알고리즘과 비교될 수 있다. CRSM에서의 방법이 시작 노드에서 발생하는 모든 후속 노드를 생성한 후 이 결과를 가지고 다시 유한한 도달성 그

```
input : initial value  of selected ATMs
output : rechability graph

rechability_graph()
begein
      no=initial value ;
      make_graph(n0);
end


make_graph(node N)
  begin
    //최소 전이 시간에 전이 가능한 전이가 존재하는한
    while(N's transition is Exist on lower time)
      begin
        T=choose_transiton(N);
                //탐색하지 않은 전이 중  하나 선택

        CN = compute_node(N,T);
                // 노드 N으로부터 전이 T에 의해
                // 생성되는 후속 노드 계산
        if(CN = one of Existing Nodes)
          begin
            add_edge(N, Existing Node);
                //계산된  노드가  이미 생성된 노드와
                //같은 경우 노드 N으로부터 존재하는
                //노드로 에지 추가
          end
        else
          begin
            add_edge(N, CN);  //노드 N으로부터 새로운
                              //노드 CN으로 에지 추가
            make_graph(CN);   //노드 CN에 대한 새로운
                              // 후속 그래프 생성
          end
      end
    return;
end
```

그림 14 도달성 그래프 생성 알고리즘

래프를 작성하는데 비해 ATM에 대한 도달성 그래프는 한 번의 실행으로 유한한 도달성 그래프를 생성한다. 이는 ATM이 시간 제약을 만족하는 경우와 만족하지 못하는 경우에 대한 전이를 제공하며 시간과 관계 없는 전이에 대해 ''를 사용함으로써 가능하다. 도달성 그래프는 검증할 ATM의 초기값을 입력받고 이를 시작 노드로 하여 생성된다. 후속 노드는 현재 노드의 전이 중 최소시간(lower bound)에 전이 가능한 모든 이벤트에 대하여 생성된다. 전이가 시간 제약을 가졌을 경우에는 최소시간에 전이가 이루어진다고 가정하고 시간 제약이 없는 경우는 사용자가 입력한 전이 시간을 이벤트 전이에 소요되는 최소시간으로 가정한다. 이와 같은 가정은 시스템이 전이 가능하게 된 순간에 전이가 이루어 짐을 의미한다. 만약 현재 노드의 다음 전이가 RPC 이벤트이고 시간 제약을 가졌다면 송신은 최소 시간에 수신은 최대 시간에 발생하며 이 두 전이 사이에 다른 전이는 발생하지 않는 것으로 한다. 그림 14는 도달성 그래프 생성 알고리즘에 대한 의사코드(pseudo-code)이다.

도달성 그래프 생성 알고리즘은 검증하고자 하는 ATM과 이들의 초기 값을 입력으로 하여 도달성 그래프를 생성한다. *make_graph*(*node N*)는 노드 *N*에 대하여 최소 시간에 대한 N의 모든 전이에 대하여 후속 노드를 생성하는 함수이다. *choose_transition*(*N*)은 노드 *N*의 전이 중 아직 탐색하지 않은 전이를 선택하는 함수이다. *add_edge*(*N*, *CN*)은 노드 *N*에서 노드 *CN*으로 에지를 생성시키는 함수이며 도달성 그래프는 초기 노드로부터 *make_graph*(node *N*)을 재귀적으로 호출하여 생성된다.

도달성 그래프는 메인 루틴인 *make_graph*(node *N*)를 재귀적으로 호출하면서 생성된다. 따라서 알고리즘의 복잡도는 노드의 수가 n일 때 $O(n^2)$이다.

4.4 도달성 그래프 예제

지금까지 기술한 ATM의 도달성 그래프를 참고문헌 [5]의 도달성 그래프와 예를 들어 비교하여 본다. 상태 기반의 정형 기법을 검증하기 위해 기술되는 도달성 그래프에 관련된 연구 중 대부분의 경우 도달성 그래프가 발생시키는 상태를 감소하기 위해 노력한다. 참고문헌 [5]는 이런 연구의 대표적인 방법으로 실시간 시스템의 상태 감소에 많은 영향을 미쳤다. [5]의 방법과 ATM 검증을 위한 도달성 그래프를 비교함으로써 본 논문에서 제시한 도달성 그래프의 효율성을 보인다.

그림 15는 [5]의 CRSM에서 도달성 그래프를 생성하기 위해 사용한 예제이다. 그림 16은 이를 ATM 방법을 사용하여 재 명세한 것으로 CRSM1이 ATM1, CRSM2가 ATM2, CRSM3가 ATM3을 각각 나타낸다.

ATM 명세의 경우 delay[4,-]과 같이 시간 제약이 만족하지 못했을 경우 시스템의 제어 이동을 명시적으로 표현하였다.

그림 17은 CRSM의 도달성 그래프이고 그림 18은 ATM의 도달성 그래프이다. 비교의 명확성을 위해 Act()시그널과 관련된 이벤트는 생략하였다.

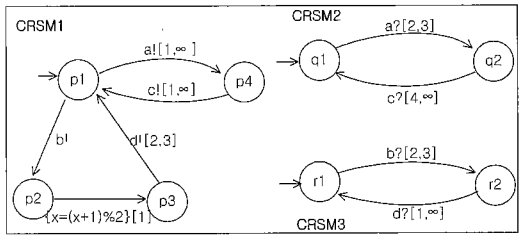


그림 15 [5]의 CRSM 예제

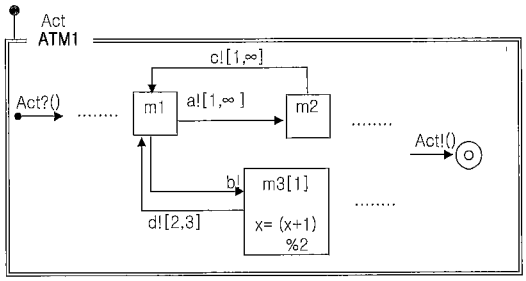


그림 16-a 그림 15의 CRSM1에 대응하는 ATM

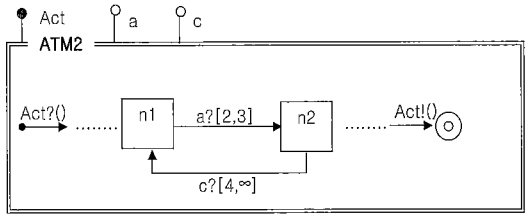


그림 16-b 그림 15의 CRSM2에 대응하는 ATM

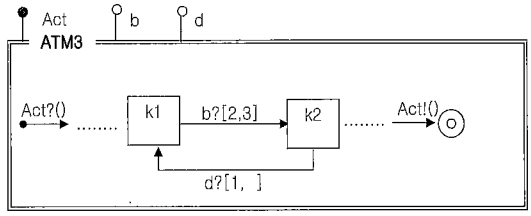


그림 16-c 그림 15의 CRSM3에 대응하는 ATM

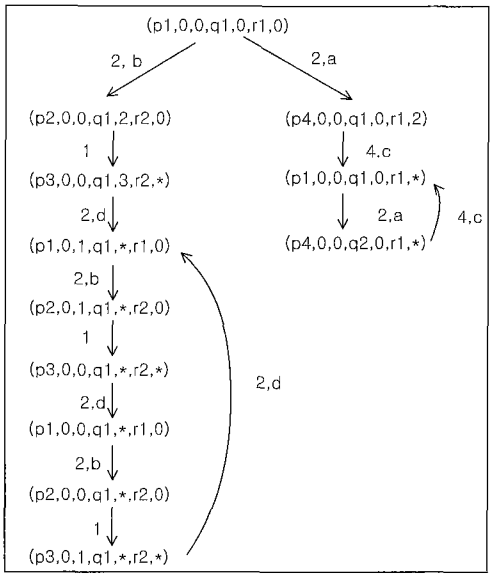


그림 17 그림 15에 대한 도달성 그래프

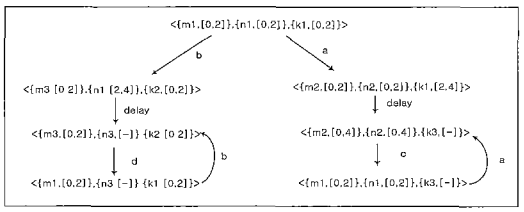


그림 18 그림 16에 대한 도달성 그래프

ATM의 경우 CRSM의 경우보다 적은 노드가 생성되었다. 즉, 시스템의 상태 공간이 작다. 이는 전이와 관계 없는 변수 x가 CRSM의 경우 노드에 포함된 것과는 달리 ATM의 도달성 그래프에서는 이를 고려하지 않음으로써 노드의 감소 효과를 가져올 수 있었다. 또한 그림 17의 도달성 그래프가 2번의 과정을 거쳐 생성된데 비해 ATM의 도달성 그래프는 1번의 실행에 의해 생성되었다. ATM이 시간 제약에 대하여 delay 이벤트의 추가로 시간의 흐름에 대한 명확한 전이를 제공하였기 때문이다. 따라서 전의의 발생 수는 증가하였지만 명확한 전이를 제공함으로써 도달성 그래프를 보다 효율적으로 생성할 수 있게 하였다.

CRSM에서의 도달성 그래프에서의 도달성 그래프 생성 결과와 비교해 볼 때 ATM을 위한 도달성 그래프가 상태 감소에 효율성을 보여주었다.

도달성 그래프의 상태 감소에 대한 또 다른 방법이 참고문헌 [8]에 기술되어 있다. 생성되는 도달성 그래프를 순방향과 역방향으로 분석하는데 있어서 히스토리 동일성(history equivalence)과 전이 동일성(transition bisimultaion)을 사용하여 효과적인 상태 감소 방법을 제시하고 있다. 1차적인 도달성 그래프를 생성한 후 히스토리 동일성으로 2차 상태 감소를 하고 그 결과를 다시 전이 동일성을 사용하여 최소 상태를 가지는 도달성 그래프를 생성하는 방법을 사용한다. 그러나 [8]이 기술하고 있듯이 이 방법의 단점은 시스템의 도달성 그래프의 시간 속성을 제거함으로써 이루어지는 방법으로 실시간 시스템의 시간 속성을 분석하기가 어렵다. 이는 실시간

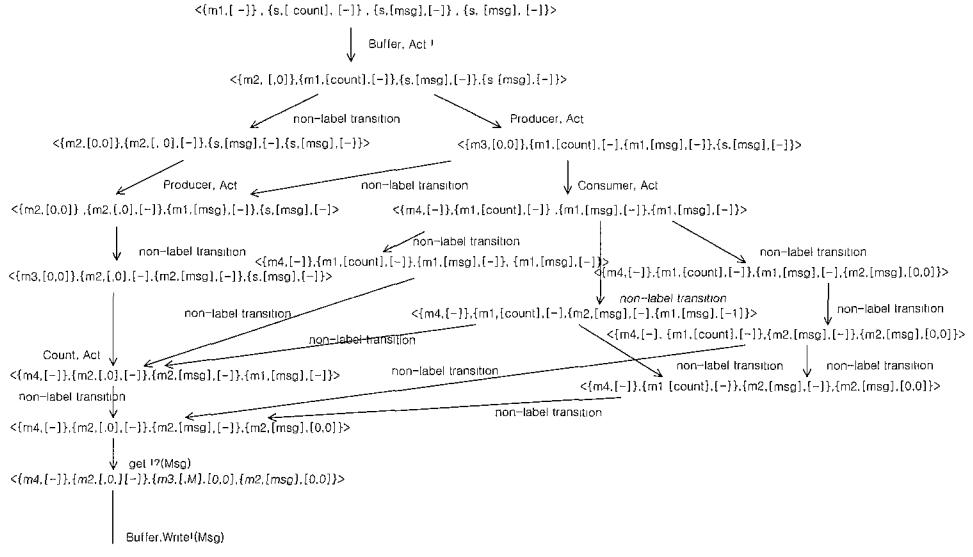


그림 19 PBC ATM에 대한 도달성 그래프

시스템의 시간 정보를 제공하지 못함으로써 동작에 대한 검증은 가능하나 시간에 대한 분석은 어렵게 된다.

ATM을 검증하기 위한 도달성 그래프는 실시간 시스템의 중요한 속성인 시간성을 유지하며 다른 여러 정형 기법들의 목적인 상태 감소를 제공하기 위한 방법을 제시하였다.

그림 18은 PBC예제 중 Producer와 Buffer에 대해서만 도달성 그래프를 작성한 것이다. 노드는 왼쪽에서부터 차례로 Producer, Buffer ATM을 나타낸다. 검증을 위한 시작 모드는 두 ATM 모두 m2이다. ATM의 종료 점은 F로 표기한다. Buffer의 길이는 2이고 'M' 이라는 메시지를 읽어 들이는 경우에 관한 도달성 그래프이다. 시간과 관련없는 전이의 시간은 '0'으로 하였다.

그림 19에서의 주요 사항은 RPC 이벤트의 처리이다. RPC의 경우 시그널을 보낸 뒤 답을 받을 때까지 기다려야 한다. 따라서 하나의 이벤트를 보내는 이벤트와 이 벤트 결과를 받는 이벤트로 나누어 노드를 생성한다. 그래프의 <{*F*},{*F*},{*F*},{*F*}> 노드는 모든 ATM이 종료 가능함을 보여주는 노드이다. 만약 시스템이 이러한 노드를 포함하지 않는 다면 교착상태 혹은 무한 실행 상태에 있다고 간주할 수 있다. 그림 19에서 종료 노드가 아닌 <{*m4*,[-]},{*m2*,[0,0],[-]},{*F*},{*F*}>은 교착상태에 있는 노드를 보여준다. 분석결과 명세 상에 존재하는 레이블이 없는 전이에 의한 에지와 노드의 생성결과 발생한 노드이다. 그래프를 분석함으로써 명세가 가진 결함 또는 실제 시스템이 가진 결함을 파악할 수 있다.

5. 현재 진행 상황

본 연구는 시스템의 순공학 환경 SEE(*Software Engineering Environment*)와 역공학 환경 SRE(*Software Reverse Engineering Environment*)을 통합하여 순환공학(*Round Trip Engineering*)을 지원하는 환경 개발 프로젝트인 SSIEM(*SEE/SRE Integrated Environment for Mission critical system*) 프로젝트의 일부이다. 이 프로젝트에서 ATM은 상태도에 기반을 둔 소프트웨어 순환 공학을 위한 정형 기법으로 개발되고 있다. 현재 ATM은 구조성, 병렬성, 시간, 예외처리 등의 실시간 시스템의 속성을 명세하기 위한 방법을 정의하였으며 명세된 시스템을 검증하기 위해 본 논문에서 기술하였듯이 도달성 그래프를 사용하는 검증 방법을 제안하였다. 현재 ATM을 위한 프로토타입의 명세 도구인 ASUVT(*ATM Specification, Understanding and Verification Tool*)을 개발하였으며 ASUVT는 정의한 실시간 시스템의 속성을 명세할 수 있다. 그림 20는 ASUVT를 사용한 PBC 예제의 일부이다. 연구는 실시간 시스템의 자원 사용과 통신 제약에 따른 전이를 명세하기 위해 확률과 분산성에 대한 명세 방법에 대해 진행하고 있다. 또한 ASUVT를 사용하여 명세한 시스템을 실제로 검증하기 위한 검증기를 개발 중이다. 검증기를 사용하여 도달성 그래프로 모델링되는 시스템의 실행을 다양한 추상화 수준에서 분석하며 병렬 실행되는 ATM의 상호 작용을 효과적으로 분석할 수 있을 것이다. ATM 정형 기법을 사용하여 시스템을 명세하고 검증기를 사용하여 명세의 정확성을 분석하게 된다. 현재 ATM으로부터 검증과 관련된 정보를 추출하고 추출된 정보를 사용하여 검증기를 개발하는 과정이다.

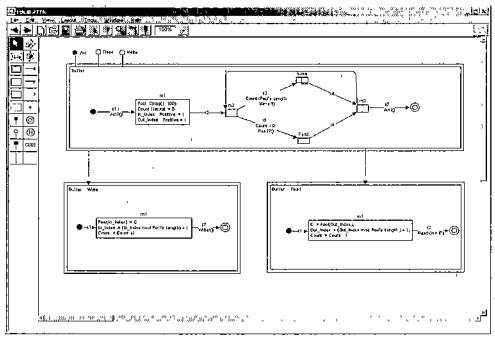


그림 20 ASUVT

6. 결론 및 향후 연구

본 논문에서는 실시간 시스템을 명세하는 정형기법은 ATM을 간략하게 기술하고 ATM으로 명세된 시스템을 도달성 그래프를 사용하여 검증하는 방법을 기술하였다. 다른 정형 기법이 제공하는 검증 방법의 단점을 보이고 이러한 단점을 줄일 수 있는 검증 방법을 기술하였다. 이는 ATM이 가지는 시간 명세 방법과 압축된 상태를 표현하는 모드의 사용과 그래프 생성 과정에서 상태 축소를 위한 추가적 생성 조건을 가함으로써 가능하였다. 이를 통해 시간에 대한 상태의 증가를 처리할 수 있었으며 알고리즘의 단순화를 유도할 수 있었다.

본 연구가 순환 공학을 위한 정형 기법을 대략 설계하고 프로토타입하는 단계에 있어 검증 방법도 추가로 정의되고 보완되어야할 사항이 존재한다. 예를 들어 레이블이 없는 전이와 레이블이 있는 전이가 동시에 전이 가능할 경우 레이블이 없는 경우 대해 어떠한 노드를 생성해야 하는 가 혹은 두 경우를 다 포함 가능한 노드는 없는가 등의 주제는 상태 공간을 줄일 수 있는 또

다른 중요한 문제가 된다. 전이에 따른 노드가 어떻게 생성되느냐에 따라 발생 가능한 노드의 수는 증가할 수도 있고 감소할 수도 있기 때문이다. 따라서 전이와 관련한 실시간 시스템의 상태 감소 방법이 계속 연구되어야 한다.

참 고 문 헌

[1] A. Shaw, "Communicating Real-Time State Machines," IEEE Transactions on Software Engineering, Vol. 18, No. 9, pp. 805-816, September 1992.

[2] D. Harel, "Statecharts: A Visual Formalism for Complex System," Science of Computer Programming. Vol. 8., pp. 231-274. 1987.

[3] I. Kang and I. Lee, "State Minimization for Concurrent System Analysis Based on State Space Exploration," Proceedings of Conference on Computer Assurance, Gaithersburg MD, June 1994.

[4] S. jahanian and A. Mok, "Modechart : A Specification Language for Real Time Systems," IBM Technical Report: RC 15140, November 1989.

[5] Sitaram C. V. Raju, "An Automatic Verification Technique for Communicating Real-Time State Machines,"

[7] C.A.R.Hoare, "Communicating Sequential Processes," Prentice-Hall. 1985.

[6] James L. Peterson, "Petri Net Theory and the Modeling of Systems," Prentice-Hall. 2000.

[8] 노경주, 박지연, 이문근, "추상 시간 기계를 이용한 순환 공학 정형기법", 한국정보과학회 소프트웨어공학회지, 제13권 제1호, 2000

[9] Inhye Kang, Insup Lee, Young-Si Kim, "An Efficient State Space Generation for the Analysis of Real-Time Systems," IEEE Transactions on Software Engineering, Vol. 26, No. 5, May 2000.

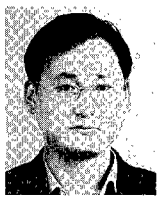
[10] Feldman and Koffman, "Ada95," Addison-Wesley, 1996.

[11] Moon Lee, "An Environment for Understanding of Real-time Systems," Ph.D. Thesis The University of Pennsylvania, 1995.

[12] W. Richard Stevens, "Unix Network Programming," Prentice Hall Software Series, 1990.

박 지 연

1999년 전북대학교 컴퓨터과학과 학사. 1999년 ~ 현재 전북대학교 전산통계학과 석사과정 중. 관심분야는 정형기법, 실시간 시스템, 검증, 소프트웨어 재·역공학, 운영체제 등



이 문 근

1989년 The Pennsylvania State University, Computer Science 학과 졸업 (이학사). 1992년 The University of Science 학과 졸업 (이공학석사). 1995년 The University of Pennsylvania, Computer and Information Science 학과 졸업 (이공학박사). 1992년 5월 ~ 1996년 1월 미국, Computer Command and Control Company, Computer Scientist로 근무. 1996년 4월 ~ 현재 전북대학교 컴퓨터과학과 조교수. 관심분야는 소프트웨어 재·역공학, 실시간 시스템, 운영체제, 형식언어, 병렬함수언어, 컴파일러 등