□ 기술해설 □

실시간 시스템을 위한 정형적 방법의 도구들

한국과학기술원 김수면* · 이귀영** · 차혜경*** · 이흥규***

● 목 차 ●

1. 서 론

최근 컴퓨터가 사회에서 중요한 곳에 이용되어 감에 따라 소프트웨어의 안정성에 대한 우려가 높아지고 있다. 특히 인간의 생명과 직결된 원자력 발전소와 같은 시스템에 사용되는 경우와 같이 오류가 발생될 경우 엄청난 손해를 초래하는 시스템의 경우에 있어서는 더욱 더 시스템의 안정성이 강조되고 있다. 이와같은 시스템을 개발하기 위해서는 개발단계의 초기에서부터 시스템을 정확히 명세하고, 시스템의 특성을 미리 검증하여 보는 것이 필수적이다.

이와 같이 엄격한 디자인과 검증을 요구하는 시스템들은 대부분 실시간 분야에서 많이 사용된다. 이러한 실시간 시스템을 정확히 명세하기 위해서 근래에는 정형적인 방법을 많이 응용하는데, 이는 자연어로 명세한 경우보다 모호함이 줄어들며 의사소통을 하는데 있어서 개발자들 사이에서 뿐만 아니라 개발자와 요구자 사이에서도 명확성을 제공하기 때문이다. 이와 더불어 정형적인 명세를 사용한 경우 원하는 시스템의 특성에 대한 검증이 가능하기 때문에 널리 이용되기 시작하고 있다.

대부분의 실시간 시스템의 경우 그 요구사항이나 시스템의 명세는 상당히 방대하고 복잡하기 때문에 이에 대한 정형적 명세와 검증을 실제 사람이 직접 한다는 것은 매우 복잡하기 때문에 이를 도와줄 자동화된 도구를 필요로 한다. 이러한 정형적 방법의 도구들은 명세 과정에서 편리성을 제공하고 사람의 실수를 막아줄 뿐만 아니라, 직접 그 정형적 명세를 가지고 필요한 성질을 검증할 수 있다. 이러한 이유로 실제 정형적 방법의 적용에서는 정형적 방법 도구의 사용이 필수적이며, 이에 많은 도구들이 현재 개발되거나 사용되고 있다.

본 고에서는 실시간 시스템의 정형적 명세 및 검증에 사용될수 있는 도구들에 대하여 알아보고자 한다.

먼저 2장에서는 간단하게나마 정형적 방법에 대해서 알아보고, 제 3장에서는 실시간 시스템에 사용할 수 있는 몇 가지 정형적 방법 도구들에 대하여 상세히 알아본다. 4장에서는 3장

*비 회 원
**학생회원
***종신회원

에서 알아본 도구 이외의 도구들에 대하여 간략히 알아보며, 마지막으로 5장에서는 결론으로 끝을 맺는다.

2. 정형적 방법

실시간 시스템은 일반적으로 concurrent 시스템에 시간 조건(timing constraint)이 부여된 것으로 인식되고 있다. 본 절에서는 실시간 시스템 개발을 효율적으로 하기 위해서 필요한 정형적 방법에 대하여 간략히 설명한다.

2.1 정형적 방법이란?

정형적 방법이란 일반적으로는 어떤 시스템들을 명세하고 검증하는데 있어서 수학에 기반을 둔 방법이라 할 수 있으나, 좀더 소프트웨어 쪽에 국한을 시키자면 수학적 체계에 바탕을 둔 소프트웨어 생산방법이라고 말할 수 있으며, 사용된 알고리즘과 시스템의 요구사항에 대한 수학적 명세와 검증 방법이라고 말할 수 있다.

2.2 잘못된 개념

다음은 일반적으로 정형적 방법에 대하여 잘못 알고 있는 몇 가지 개념에 대하여 살펴본다 [1].

- 정형적 방법은 고도로 훈련된 수학자들을 필요로 한다
  물론 엔지니어들이 정형적 방법을 사용하기 위해서 교육을 받을 필요는 있지만, 정형적 방법을 적용한 예에서 보면 이러한 교육이 어렵지 않고, 일반적인 수학 교육을 받은 엔지니어들이 쉽게 적응할 수 있는 방법이라는 것을 알 수 있다.
- 정형적 방법은 실제 큰 시스템에서는 사용될 수 없을 것이다
  외국에서는 많은 시스템 개발에서 실제로 정형적 방법이 사용되어지고 있다. 다음은 산업용 프로젝트에서 정형적 방법이 사용된 몇 가지 실제 예이다
  1. IBM CISC 트랜잭션 프로세싱 시스템은 50만 라인에 해당하는 코드를 가진 큰 시스템으로 유지보수성을 높이기 위해 Z 명세언어를 사용하여 정형적으로 시스템을 기술하였다
  2. Danish Datamatik 센터는 여러해 동안 정형적 방법을 사용하여 상업용 컴파일러 개발을 하고 있다
- 정형적 방법은 소프트웨어가 완벽하다는 것을 보장할 것이다
  정형적 방법은 어떤 시스템에 대하여 완전하다고 보장할 수는 없는데, 이것은 실제 실생활의 시스템과 증명하고자 하는 모델이 완전히 동일하지는 않다는 것과 정형적 방법을 사용할 때 명세를 하는 기술자의 실수로 오류가 발생할 수 있다는 것이다. 그러나 이러한 한계에도 불구하고 정형적 방법을 사용해야 하는 이유로는 다음 두 가지를 들 수 있는데, 첫 번째로는 프로그램의 정당성을 확인할 수 있는 다른 방법이 존재하지 않는다는 점이다. 두 번째 이유로는 정형적 방법을 사용하면 모든 종류의 에러를 발견하기 쉽다는 점이다.
- 정형적 방법은 단지 safety-critical한 시스템에만 유용하다
  정형적 방법은 안정성 보장뿐만 아니라 고장으로 인한 비용이 많이 들게 되는 시스템에서 요구의 적합성, 유지보수성, 구현의 용이성 등을 높이기 위해서 사용될 수 있다. 즉 정형적 방법으로 기술된 시스템은 요구명세에서 구현까지의 과정에서 각 개발자간의 개념상의 모호함을 없애줌으로써 개발자간의 명확한 의견교환을 통해 개발을 쉽게 할 수 있고, 시스템을 유지 보수하거나 변경할 때도 모호성을 없애줌으로써 유지보수나 변경을 좀더 쉽게 할 수 있다는 장점을 가지고 있다

2.3 정형적 방법의 분류

현재까지 연구된 정형적 방법들은 여러가지 부류로 나뉠 수 있으나, 그중 일반적으로는 다음과 같이 분류를 할 수 있다.

1) Real-time temporal logic

Temporal logic은 first-order predicate

logic과 마찬가지로 철학에서부터 기원을 찾을 수 있으며, 시간에 따라 변하는 상황을 추론하기 위하여 사용된다. 그 의미론은 상태로 나타내어지는 어떤 상황의 정적인 면과 상태와 상태사이의 관계를 나타내는 동적인 면 사이의 구분을 뚜렷이 하고 있다. 다음과 같은 4가지로 세분하여 분류할 수 있다

- TTM/RTTL(Timed Transition Model/ Real-Time Temporal Logic)
- MTL(Metric Temporal Logic)
- Branch time temporal logic
- Interval based temporal logic and others

2) Process algebra

언어의 구문이 대수(algebra)론에 의해 정의 되어 있고, 사용된 대수론의 공리를 나타내는 몇개의 대수식이 모여서, 이 언어로 명세된 시스템을 증명하기 위한 기본 이론을 이루고 있다. concurrent 시스템을 분석하는데 중요한 역할을 하며, 구조적인 명세 방법을 제공할 수 있다.

- CSP(Communication Sequential Process)
- CCS(Calculus of Communicating Systems)
- Timed CSP
- Communicating Shared Resources
- Temporal CCS

3) Petri nets

Petri nets은 어떤 체계(시스템 + 환경)를 모델링하기 위한 방법으로서, 그 이론에 의해 어떤 체계를 수학적 표현인 petri nets으로 모델링 할 수 있다. 그러므로 petri nets을 분석함으로써 모델링되는 체계의 구조와 동적 행동에 대해 중요한 정보를 얻을 수 있다. 다음과 같은 2가지 큰 부류로 나눌 수 있다

- Petri nets
- Timed petri nets

4) Assertional calculi

Temporal logic을 기반으로 하며, 그 위에 가정적 추론(assertional reasoning)을 함께 함으로써 concurrent 시스템의 명세를 쉽게 할 수 있는 장점을 가지고 있다

- Real-time Hoare logic

5) Programming languages

정형적으로 정의된 의미를 가지는 고수준 프로그래밍 언어를 말한다. 이러한 언어들은 프로그램의 검증을 할 경우 정형적인 목표를 제공할 수 있다. 대표적인 실시간 프로그래밍 언어로는 다음과 같은 것이 있다

- Ruth
- Lustre
- RT-Aslan

3. 정형적 방법의 도구들

앞 장에서 정형적 방법과 오해하기 쉬운 점 그리고 마지막으로 정형적 방법에 대한 간단한 분류를 하였다. 그러면 이러한 방법을 실제 응용하기 위하여 필요한 도구에 대하여 알아본다.

다음에 설명된 도구들은 현재 연구되고 있으며 실험적으로 실제 문제에 사용되고 있는 것으로서, 실시간 시스템을 위한 시간 성질을 표현할 수 있는 것들이다.

3.1 CWB(The Edinburgh Concurrency Workbench)

CWB는 concurrent 시스템을 다루고 분석하기에 적합한 자동화된 도구로서 여러가지 다른 프로세스 의미론을 사용하여, 명세한 두 시스템간의 동질성과 다른점 등을 확인할 수 있는 equivalence, preorder등과 모델 체크를 지원하고 있다. CWB가 지원하는 기능은 다음과 같다[2].

- TCCS(Temporal CCS)나 SCCS(Synchronous CCS)로 정형 명세를 할 수 있고, 이에 대하여 상태 공간의 분석이나 여러 equivalence와 preorder를 확인할 수 있다
- 확인하고자 하는 어떤 성질을 modal logic으로 표현하여, 주어진 정형 명세가

이를 만족시키는지 확인하여 볼 수 있다
- 만약에 명세된 두 프로세스가 서로 다르다면, 그 차이점을 자동화된 과정을 통하여 논리식으로 나타내어준다
- 사용자와의 상호 교류에 의하여 한 agent (명세하려는 시스템내에서 독자성을 가지는 부분)의 상태 공간을 원하는 방식으로 추적해볼 수 있다
- 어떤 전체적인 명세가 주어지면 세부적으로 빠진 부분에 대하여 반자동으로 명세를 만들어 낼 수 있다

3.1.1 사용 언어

CWB가 사용하는 명세 언어는 모두 Milner의 CCS[3]에 바탕을 두고, 여기에 시간 특성을 첨가한 것으로 TCCS와 SCCS가 있다[4]. 이들에 대하여 살펴보면 다음과 같다.

TCCS : Timed-CCS는 비동기적 지연, 시간 지연, 그리고 강한 선택자와 약한 선택자 등을 기본 개념으로 포함하고 있으며 시간적 모형은 성기고, 전역 클럭이 있다고 가정하였다. 또 시간과 관계된 action(예를 들면 delay)은 시간 개념이 필요없는 action(예를 들면 커뮤니케이션)과 분리되어 따로 정의되었다. 시간개념이 없는 action들은 순간적인 것으로 가정하였다.

SCCS : 프로세스들은 기본적인 action들의 집합인 Act에서 모집된 action들의 비결정적 연속으로 정의가 가능하며, 동기의 개념은 프로세스들에 대한 병행 합체 연산자의 의미에 의하여 보강되었다. 병행 합체를 하고있는 두 프로세스들은 각 주기당 하나씩의 action을 취하게 되어있다. 그리고 전역 클럭이 있어, 주기마다 각 프로세스에게 다음 action을 수행하도록 신호를 보내주고 있다. 운영적 의미론은 명칭붙은 전이 체계에 의해 정의된다.

3.1.2 지원 도구

CWB는 CWB와 daVinci라는 도구로 구성되어 있다.

CWB : 사용자와 상호작용을 하는 시스템으로, 사용자는 어떤 명세에 대하여 인식자를 지정할 수 있으며 프로세스의 유도에 대한 분석과 프로세스들이 서로 equivalence를 이루는지 확인해 볼 수 있다. GUI(Graphical User Interface)는 제공되지 않으며, emacs를 사용하여 daVinci와 연결시킬 수 있으며 몇 가지 간단한 사용상의 편의를 제공받을 수 있다

daVinci : CWB를 위하여 만들어진 도구는 아니지만 CWB와 연계하여 사용할 수 있다. CWB에서 명세된 agent에 대하여 CWB는 텍스트로된 상태 변이를 출력할 수 있는데, daVinci는 그것을 입력으로 받아 변이 상태를 그래프로 나타낼 수 있다

3.1.3 예 제

Buff3라는 agent와 Spec이라는 agent를 명세한 후 두개의 agent사이에 equivalence가 성립하는지를 확인해 보는 과정을 나타낸 것이다. agent Buff3는 크기 3의 버퍼를 나타내며,

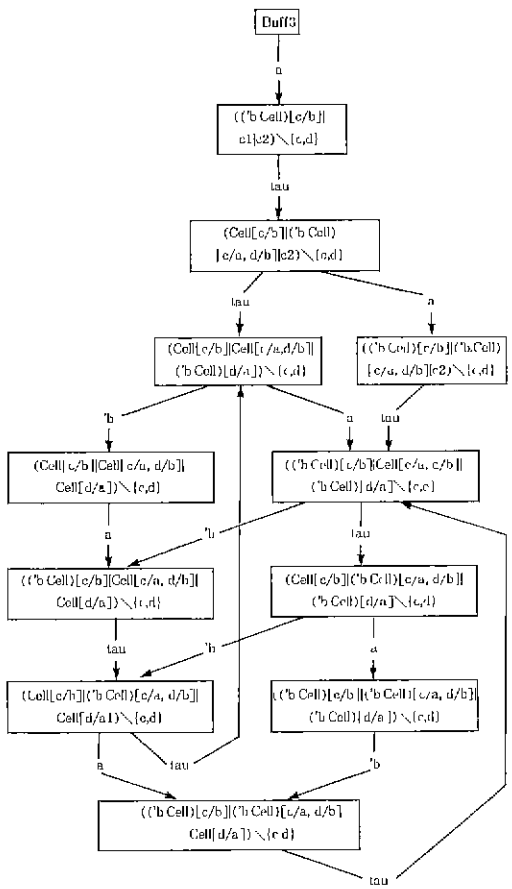

그림 1 daVinci를 이용하여 나타낸 Agent Buff3의 상태 전이도

3개의 Cell을 연결하여 구현하였다. Buff3의 명세는 크기 3의 버퍼의 구조가 실제 어떻게 이루어지는지를 나타낸 것이며, Spec은 크기 3의 버퍼가 어떻게 동작해야 하는지 그 외형적인 것에 대한 명세이다. 다음의 결과에서 Spec은 Buff3와 외부에서 관찰하기에는 똑같은 움직임을 보여준다는 observational equivalence를 이루고 있는 것을 알수 있다. 결국 Buff3와 같이 시스템을 구성하면 Spec과 같은 움직임을 할 것이라는 것을 확신할 수 있는 것이다. 그림 1에서는 agent Buff3가 전이 가능한 상태를 찾아 전이 상태도를 daVinci를 사용하여 나타낸 것이다.

```
Edinburgh Concurrency Workbench, version 7.0alpha3,
Tue Sep 20 20 : 42 : 57 BST 1994
  (1) Command : agent Cell = a.'b.Cell;
  (2) Command : agent C0 = Cell[c/b];
  (3) Command : agent C1 = Cell[c/a,d/b];
  (4) Command : agent C2 = Cell[d/a];
  (5) Command : agent  Buff3 = (C0|C1|C2) \
      {c,d};
  (6) Command : agent Spec = a.Spec';
  (7) Command : agent  Spec' = 'b.Spec + a.
      Spec";
  (8) Command : agent  Spec" = 'b.Spec' + a.'b.
      Spec";
  (9) Command : eq (Buff3, Spec);
  (9a) true
  (10) Command : quit;
```

3.2 StateTime

StateTime은 시스템의 정형 명세와 검증에 visual한 방법과 temporal logic을 사용하는 도구들의 묶음으로서, 실시간 safety-critical 시스템을 디자인하는데 사용되도록 만들어졌다[5]. 또한 이러한 도구들과 같이 사용될 수 있도록 디자인 방법론이 제공되어진다.

3.2.1 사용 언어

StateTime이 사용하는 언어는 Build가 시스템을 디자인하는 과정에서 사용하는 TTM과, VERIFY나 DEVELOP가 검증하기 위한 성질을 명세하는데 사용하는 RTTL이 있다.

TTM : TTM은 실제 시스템을 표현하는 효과적인 방법을 제공하는데, 이산 실시간 프로그래밍 언어, 페트리 네트, STATECHART등을 표현할 수 있으며 지연, 시간 초과 그리고 다양한 스케쥴링 제약 등 대부분의 실시간 특성을 표현할 수 있다. 다중 프로세싱, 멀티 프로세싱 그리고 다른 채널을 통한 메시지 전달뿐 아니라 공유 변수를 이용한 통신도 모델링할 수 있다. TTM은 세개의 tuple $M=(\nu, \Theta, \mathcal{T})$로 정의 되는데, 여기서 ν는 행동과 데이터 변수의 집합이며, Θ는 변수의 초기 조건을 나타내는 predicate이고 $\mathcal{T}$는 모든 전이의 집합이다. 각각의 전이는 enabling 조건, 전이 함수 그리고 시간 범위를 갖는다.

RTTL : RTTL은 검증되어야 할 특성을 명시하는데 사용되어지고 있으며, 다음과 같이

시스템 = 제어를 받는 부분 + 제어 부분

시스템이 주어졌을 때, 시스템이 어떤 성질을 만족하는가를 검사하고자 할 때에 그 성질을 표현하기 위해 RTTL을 사용한다. RTTL은 수식이 고정된 시간 변수에 대해 임의의 first-order quantification을 사용하여 클럭 시간 변수 t를 수식 연산자와 명시적으로 함께 사용하기 때문에 explicit clock logic으로 분류되며, 이 때문에 표현성은 매우 좋으나 비결정적인 특성을 갖는다.

3.2.2 지원 도구

StateTime이 지원하는 도구는 BUILD, VERIFY 그리고 DEVELOP의 3가지 도구이다.

BUILD : Build는 사용자가 시스템의 visual 명세(TTMcharts)를 조직적인 방법으로 만들고 수행시키면서, 만족할 만한 결과가 나올 때까지 명세를 개량하는데 직접 사용하는 도구이다(그림 2 참조). TTMcharts는 최종적인 명세가 완성되기 전에도 어느 부분이든지 수행이 가능하다.

VERIFY : VERIFY는 RTTL로 기술된 성질을 TTMchats에 대하여 확인하는 모델 확인 도구이다. 우선 유한 상태 TTM의 상태 도달 가능성 그래프를 계산한 후, 그 TTM이 주어진 요구를 만족시키는가를 확인할 수 있다.

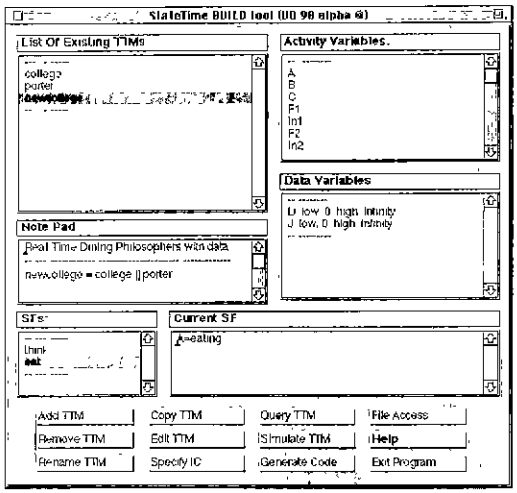

그림 2 BUILD 도구에서 현재의 모든 TTMcharts를 나타낸 모습

DEVELOP : DEVELOP는 모델이 무한 상태일 때 완전히 자동화된 검증방법이 사용될 수 없기 때문에, 반자동화된 검증방법을 찾는데 사용하는 도구이다. 만약 그러한 방법이 찾아진다면 그 모델은 주어진 요구 조건을 만족시키는 것이며, 그 찾아진 방법으로 전체 명세로부터 일부 명세를 규칙적으로 찾아낼 수 있다.

3.2.3 예 제

예제는 dining philosophers라는 고전적인 문제에 실시간의 제한된 응답시간이라는 조건을 첨가한 실시간 dining philosophers이다. 여기서 BUILD 도구는 시스템을 명세하기 위하여 문제를 TTMcharts로 나타내는데 사용되었다. 그림 2는 현재 이미 명세된 TTMcharts들을 나타내고 있으며, 이중 college는 제어를 받는 부분이다. 이 제어는 porter라는 곳에서 맡는다고 가정하자. 그러면 이제 VERIFY를 사용하여 문제를 검증함으로써 porter가 갖어야 하는 조건을 찾아내고, 이 조건을 만족시키는 porter의 명세를 만들 수 있다. 그림 3은 philosopher라는 activity를 편집하는 화면을 나타낸 것이다. 그리고 이렇게 명세된 모델이 실제 원하는 모델과 틀림이 없는가는 시뮬레이션을 이용하여 확인할 수 있으며, 그 결과를 가지고

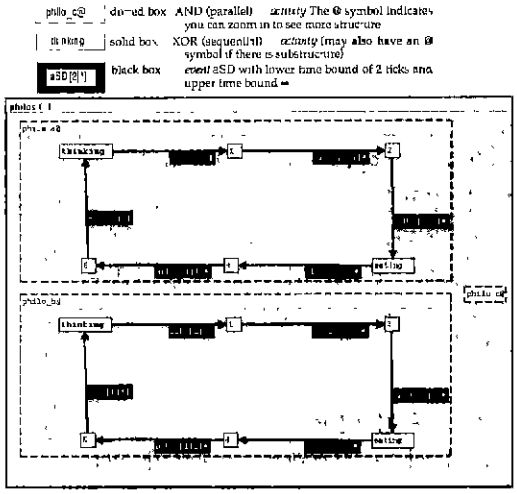

그림 3 BUILD 도구를 이용하여 philosopher를 편집하는 모습

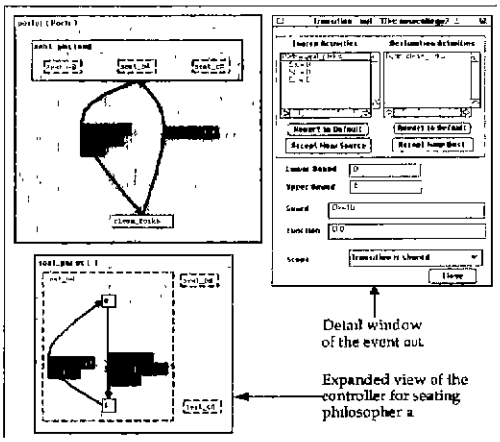

그림 4 BUILD 도구로 porter를 디자인하는 모습

제어부분을 명세하는데 사용할 수 있다. 그림 4는 porter를 디자인 하는 과정을 나타낸 것이다.

3.3 HyTech(The Cornell HYbrid TECHnology Tool)

HyTech는 선형의 혼합형 시스템을 위한 심볼릭 모델 체커로서, 기본 모델로 사용한 시스템 모델은 유한 오토마타를 확장한 혼합 오토마타이다. 요구 명세 언어로는 ICTL(Integrator Computation Tree Logic)을 사용하는데, ICTL은 시간 제약을 위해 클럭과 stop-watch를 가지는 branching-time logic을 사용한다.

ICTL을 이용하여 안정성, liveness, 실시간성 그리고 지속성 등의 성질을 명세할 수 있다. 만약에 시스템을 명세하는 혼합 오토마타와 요구 사항을 나타내는 ICTL 식이 주어지면, HyTech는 요구 조건을 만족시키는 시스템 상태의 집합을 나타내는 상태 서술식을 계산해 낼 수 있다. 즉 ICTL의 성질을 이용해서는 모델 체크를 할 수 있으며, 혼합 오토마타의 성질을 이용해서는 도달 가능성 분석을 할 수 있다[6].

3.3.1 사용 언어

HyTech는 시스템을 기술하기 위한 언어로 혼합 오토마타를 이용하며, 주어진 시스템에서 어떤 성질을 검증하기 위해서는 ICTL을 사용한다.

Hybrid Automaton : 혼합 오토마타는 실제 값을 가지는 유한 벡터 χ와 명칭붙은 다중 그래프 (V,E)로 이루어진다. χ로 벡터 χ의 첫 번째 유도 벡터를 나타내며, edge E는 분산된 시스템의 행동들을 나타내며 벡터 χ에 대한 비결정적 guarded 지정으로 나타내어진다. Vertices V는 환경들의 연속된 행동을 나타내며 χ와 $\dot{\chi}$에 대한 제약으로 이름지어진다. 오토마타의 상태는 단발적인 시스템의 행동이나 시간이 경과함에 따라 연속된 행동으로 모두에 의하여 변할 수 있다

ICTL : A라는 주어진 선형 혼합 오토마타 대한 ICTL은 A의 데이타와 제어 변수 그리고 integrator라는 2가지 변수를 포함한다. Integrator(또는 stop watch)는 정지되고 또 다시 시작될 수 있는 클럭을 말하며, ICTL 식은 부울 연산자를 사용한 상태 기수식으로부터 만들어 진다[7].

3.3.2 지원 도구

HyTech : HyTech의 구조는 그림 5와 같으며, 주어진 명세에 대하여 도달 가능성 분석, 추상 해석 그리고 모델 체크를 수행할 수 있다.

3.3.3 예 제

"leaking gas burner"란 문제인데, 가스 버너는 두 가지 상태로 존재할 수 있다. 가스가 새고 있거나 그렇지 않은 두 가지 상태인데, 가스가 새는 것은 1초 이내에 발견되고 버너는 정지 되어야 한다. 그리고 만약 새는 것이 멈춘다면, 버너는 적어도 다시는 세지 않는다는 보장을 받을 수 있는 시스템이라고 가정한다. 시스템의 초기 상태는 가스가 새는 상태이다. 가스 버너의 선형 혼합 오토마타는 그림 6과 같이 모델링 되어 진다. 클럭 x는 현재 상태로 들어온 뒤 소비된 시간을 나타낸다. 하지만 시스템을 분석하기 위해서는 가상의 변수가 2개 더 필요한데, 이 중 stopwatch t는 가스가 새는 누적된 시간을 나타내며 y는 전체 시스템에서 소요되는 시간을 나타낸다. 이 시스템을 위한 입력 화일은 다음과 같으며, HyTech는 결과로서 "Non-leaking duration requirement satisfied"를 출력한다.

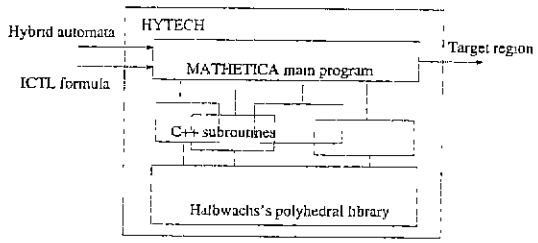

그림 5 HyTech의 구조

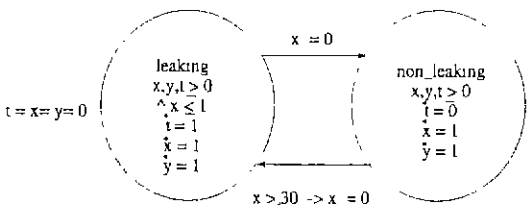

그림 6 leaking gas burner의 오토마타

```
-- leaking gas burner
vars :
    x,                  -- time spent in current location
    y : clock;          -- total elapsed time
    t : stopwatch;      -- leakage time
automaton gas_burner
synclabs : ;
initially leaking & t = 0 & x = 0 & y=0;
```

```
loc leaking : while x>=0 & y>=0 & t>=0 & x <=1
wait {dt=1}
  when True do {x' = 0} goto not_leaking;
loc not_leaking : while x>=0 & y>=0 & t>=0 wait
{dt=0}
  when x>=30 do {x' = 0} goto leaking;
end
var init_reg, final_reg, b_reachable, f_reachable : region;
init_reg := loc[gas_burner] = leaking & x=0 &t=0
& y=0;
final_reg := y>=60 & t >= 1/20 y;
-- forward reachability does not terminate
-- f_reachable := reach forward from init_reg
endreach;
b_reachable := reach backward from final_reg
endreach;
if empty(b_reachable & init_reg)
  then prints "Non-leaking duration requirement satis-
fied"
  else prints "Non-leaking duration requirement not satis-
fied"
endif;
```

4. 그 외의 도구들

앞 절에서는 시간 성질을 표현할 수 있는 실시간 시스템에 적합한 도구들에 대하여 알아보았다. 물론 실시간 시스템의 특성상 시간 성질을 표현할 수 있는 능력이 필요하기는 하나, 시간 성질을 사용하지 않고 시스템을 기술함으로써도 충분히 정형적 방법의 이득을 취할 수 있다. 이에 본 절에서는 시간 성질을 표현할 수는 없지만 다른 분야뿐만 아니라 실시간 시스템에서도 충분히 사용할 수 있는 도구들에 대하여 알아 본다.

4.1 도구에 대한 조사 항목

여기에서 소개하는 것이 도구에 대하여 필요한 모든 조사가 될 수는 없겠지만 몇 가지 꼭 필요한 항목에 대하여 소개를 한다. 이외에도

표 1 도구의 사용 언어, 증명 지원 여부 그리고 시간 성질 제공 여부

도구 이름	주관 기관	사용 언어	증명	GUI	시간 성질	문서
Action Semantics	BRICS	Action Semantic notation	–	Yes	No	A
B-Method	B-Core(UK) Limited	Abstract Machine Notation	Yes	–	–	B–
Centaur	INRIA	–	–	Yes	No	–
DisCo	Tampere Univ. of Technology	DisCo	Yes	Yes	–	A
FDR	Formal Systems Ltd.	CSP	–	–	No	–
Isabelle	C U Computer Lab T U Munich	1st, higher order logic	Yes	Yes	No	A
JACdK	IEI-CNR	CCS, Meiji, BLotos FC2, logic ACTL	Some	Yes	–	–
LOTOS Toolbox	Info. tech. Arch. B.V.	LOTOS	–	Yes	No	A
Larch	MIT Lab. for Computer Sci.	Multisorted first order logic	Yes	No	No	A
Mural	Rutherford Appleton Laboratory	VDM-SL	Yes	Yes	No	–
MWB	Dept. Comp. Syst. Upsala University	Pi-calculus	Yes	No	No	B
Nqthm-1992	Computational Logic, Inc.	variant of Pure Lisp	Yes	–	No	C
ProofPower	ICL Secure Systems	HOL, Z	Yes	–	No	A+
PVS	SRI Computer Science Laboratory	typed high-order logic	Yes	Yes	No	A
SPIN	AT&T Bell Labs	PROMELA	No	Yes	No	A–
StateTime	Dept. of CS. York Univ.	timed-transition model	Yes	Yes	Yes	C+

표 2 도구의 실제 사용된 예

도구이름	실예
DisCo	industrial application within the ESSI/FOCUS project
FDR	Used to develop and verify communications hardware at Inmos
Isabelle	used to reason about functional programs written in Miranda.
JACK	A software control system of an hydro power plant, Formal Verification of Safety Properties of a Railway Signalling Control System
LOTOS Toolbox	Specification of OSI protocol
Nqthm-1992	used to check proofs of over 16,000 theorems from many areas of number theory, proof theory, and computer science.
ProofPower	to formally verify their SWORD multi-level secure relational database management system
PVS	microprocessor for aircraft flight-control, diagnosis and scheduling algorithms for fault-tolerant architectures, requirement specification for the Jet-Select function of the Space Shuttle flight-control system

표 3 도구를 사용하는데 필요한 H/W와 S/W 환경

도구이름	사용 환경
Action Semantics	Sun4 running SunOS 4.1.x(5.x), 14Mb Disk Space
B-Method	IBM/RS 6000 running AIX, Sun Sparc running SunOS 4.1.X(5.X) 16Mb RAM, 20Mb Disk Space, C compiler
Centaur	Sun Sparc running SunOA 4.x(5.3), DECstation running Ultrix V4.x, Silicon Graphics running IRIX 4.x, 32Mb RAM, 67Mb disk space, X11R5 or a compatible version(like OpenWindow 3.3), Le-Lisp(provided) ECLiPSe prolog (not provided)
DisCo	SunSparc running SunOS 4.1 with Open Window3.0, 16Mb RAM, 12Mb disk
FDR	SunSparc running SunOS or Solaris 2.x, IBM RS/6000 workstation running AIX 3.2.2, 16Mb Ram, 60Mb Virtual Memory
Isabelle	Requires an SML compiler
JACK	SunSparc running SunOS 4.1.3(Solaris 2.4)
LOTOS Toolbox	Sun 3 and Sun 4 running SunOS, Hp running HP Unix, 16Mb RAM, 35Mb disk
Larch	SunSparc running SunOS, DECstation under Ultrix, DEC AXP under OSF/1
Mural	Full license for Smalltalk-80 release 4.1 is required, Workstation with 12Mb RAM
MWB	SunSparc running SunOS 4.1.3(Solaris 2.4)
Nqthm-1992	Common Lisp on a platform with about 8Mb RAM
PrrofPower	SunSparc running SUNOS 4.1.3(Solaris 2.3), Full licencees must have a licence also for the relevant version of the AHL implementation of Standard ML, known as PolyML
PVS	SunSparc, 20Mb RAM, 60Mb disk, Unixs and GNU Emacs
SPIN	Unix Operating System, Tcl/Tk is needed for XSPIN
StateTime	Quintus Prolog V3.0 or later on any architecture needed, smalltalk environment Parcplace Visualworks V1.0 needed

명세 언어의 문법을 체크할 수 있는지, 타입을 체크할 수 있는지, 정적 의미론을 가지는지, 에니메이션 기능이 있는지, refinement가 가능한지등의 물음을 할 수 있다.

표 1은 도구들에 대하여 기본적인 사항에 대한 조사를 나타내고 있으며, 표 2는 실제 응용분야에서 사용된 예에 대하여 나타내고 있다. 마지막으로 표 3은 주어진 도구를 사용하기 위하여 필요한 S/W와 H/W의 환경을 보여주고 있다.

4.2 실재 조사 항목

주관 기관 : 도구를 만드는 것을 주도하고 있는 기관이다.

사용 언어 : 사용하는 정형 명세 언어이며, 실제 이 언어는 독립적인 명세 언어가 아니라 기존의 명세 언어를 도구에 맞게 변형한 것이므로 이름이 생소하고 다양하다. 하지만 실제 범주는 2장에서 조사된 범주에 속해 있으며, 약간의 변형을 거친 것이다.

증명 지원 : 증명을 지원하는 가의 여부로서, 도구를 사용하는 목적이 어떤 증명을 필요로 하는 것이라면, 정형적 증명이 지원되는 도구를 선택하여야만 한다.

GUI 제공 : 그래픽 유저 인터페이스의 지원 여부이다.

Timing Property : 시간 성질을 표현할 수 있는지의 여부로서, 실시간 시스템을 명세하면서 시간 성질을 꼭 표현하려면 이 특성이 지원되어야 한다.

사용된 예 : 실제 사용된 사례이며, 도구의 신뢰도를 확인할 수 있다.

H/W, S/W 환경 : 도구를 수행시키는데 필요한 H/W , S/W 환경

관련 문서 : 관련된 도큐먼트들이 얼마가 있나 하는 것이다. 여기서는 등급을 단순히 A,B, C로 나누어 표기했다.

5. 결 론

실시간 시스템의 중요성이 증가하고, 안정성이 특별히 요구되는 분야뿐 아니라 다른 여러 곳에서도 많이 사용되면서 실시간 시스템 소프트웨어 개발시에 안정성을 보장할 수 있는 개발방법이 매우 중요하게 되었다. 특히 실시간 시스템에서는 요구되는 시스템의 안정성에 따라 차이는 있지만 개발이 시작될 때부터 끝날 때까지 완벽한 검증을 할 수 있는 방법이 필요하다.

이에 따라, 유일하게 이러한 요구를 만족시킬 수 있는 정형적 방법이 필요하게 되었고, 이의 사용을 뒷바침하기 위해 정형적 방법 도구들이 필요하게 되었다. 본 고에서는 실시간 시스템의 개발과정에서 요구되고 있는 정형적 방법을 실제 문제에 사용하기 위하여 필요한 정형적 방법의 도구에 대하여 간략한 정형적 방법의 설명과 더불어 알아 보았다.

알아본 내용에 따르면, 여러가지 도구들이 각자 어떠한 응용분야를 가지고 있으며 현재도 계속 연구중이며 발전되고 있다는 점이다. 그리고 실제 적용된 사례는 많지는 않으나 적용된 경우에는 충분한 효과를 올렸다는 것이다. 그리고 아직 많은 수의 도구들이 시스템 개발과정중 한 부분만을 지원하고 있으나 점차로 명세부터 시작하여 검증뿐만 아니라 프로그램 synthesis까지의 개발과정 전 과정에 응용될 수 있는 종합적인 도구로 발전을 모색하고 있다는 것이다.

요약하자면, 정형적 방법이 사용하기에 까다로운 점이 있지만, 현재의 정형 방법 도구들은 이러한 문제를 많이 도와줄 수 있으며, 더욱 더 사용하기에 편리하도록 바뀌고 있다. 즉, 그래픽 유저 인터페이스들을 통한 사용자의 편의성 증대, 그리고 명세, 검증뿐만 아니라 프로그램 synthesis 등의 개발과정 전 과정에 응용할 수 있는 종합적인 구성을 갖추는 노력을 하고 있으며, 앞으로 이 추세는 계속될 것이다.

참고문헌

[1] A. Hall, "Seven myths of formal methods," *IEEE Software*, pp. 11-19, Sept. 1990.

[2] F. Moller, "The edinburgh concurrency workbench, *tech. rep.*, *University of* Edinburgh, 1994.

[3] R. R. Milner, *Communication and Concurrency*. Prentice Hall, 1989.

[4] D. Scholefield, "The formal development of real-time systems : A review," tech. rep., University of York, Feb. 1992.

[5] J. S. Ostroff, "Statetime - a visual toolset for thr design and verification of real-time systems," Tech. Rep. CS-ETR-94-07, Department Of Computer Science, York University, 1994.

[6] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "A user guide to HYTECH," in *Lecture Notes in Computer Science 1019*, pp. 41-71, 1996.

[7] R. Alur, T. A. Henzinger, and P.-H. Ho, "Automatic symbolic verification of embeded systems," in *Proceedings of the 14th Annual IEEE Real-time Systems Symposium*, pp. 2-11, 1993.

[8] C. Sturling, "An introduction to modal and temporal logics for ccs," in *Lect. Notes in Computer Science 491*, pp. 2-20, 1991.

[9] J. S. Ostroff, "Formal methods for the specification and design of real-time safety critical systems," *The Journal of Systems and Software*, pp. 33-60, Apr. 1992.

[10] R. R. Milner, "A synchronous calculus for communicating systems," Tech. Rep. CSR-xx-82, University of Edinburgh, 1982.

[11] C. Hoare, "Communicating sequential processes," *Communication of the ACM*, vol. 21, pp. 666-677, Aug. 1978.

[12] J. S. Ostroff, "Statetime - a visual toolset for the design and verification of real-time systems," Tech. Rep. CS-ETR-94-07, University of York, Sept. 1994.

[13] J. V. Guttag, e. James J. Horning, *Larch : Languages and Tools for Formal Specification*. Springer-Verlag, 1993.

김 수 면

1995 연세대학교 이과대학 수학과 학사
1995~현재 한국과학기술원 석사
관심분야 : 실시간 운영체제, 점형적 방법론, 멀티미디어 컴퓨팅

이 귀 영

1992 부산대학교 자연과학대학 전자계산학과 학사
1994 한국과학기술원 석사
1995~현재 한국과학기술원 박사
관심분야 : 실시간 운영체제, 결함허용 시스템, 멀티미디어 컴퓨팅

차 혜 경

1979 서울대학교 자연과학대학 수학과 학사
1983 미국 위스콘신대학 전산학과 석사
1984~86 미국 럿거스대학 전산학과 박사과정 수료
1986~89 한국과학기술대학 전자전산학부 전임강사
1989~92 한국과학기술원 전산학과 전임강사
1993~현재 한국과학기술원 교양학부 기초과학과정 전임강사
관심분야 : 프로그래밍 언어, 병렬 알고리즘

이 흥 규

1978 서울대학교 공과대학 전자공학과 학사
1981 한국과학기술원 전산학과 석사
1984 한국과학기술원 전산학과 박사
1985~86 Univ. of Michigan, U.S.A. Research Scientist
1986~현재 한국과학기술원 전산학과 부교수
관심분야 : 고장감내 시스템, 엄격한 실시간 시스템