제어 및 데이터 신호에 의한 Esterel에서의 새로운 회로 중복사용 문제

(New Schizophrenia Patterns on Esterel caused by Control/Data Signals)

윤 정 한 † 김 철 주 † 김 성 건 † 최 광 무 †† 한 태 숙 ††
(Jeong-Han Yun) (Chul-Joo Kim) (Seonggun Kim) (Kwang-Moo Choe) (Taisook Han)

요 약 Esterel은 명령형 동기언어로서, 많은 경우에 메모리, 캐쉬 컨트롤러, 버스 인터페이스 등을 개발하는데 사용하고 있다. Esterel 프로그램은 특정 상황에서 한 문장이 한 단위시간 안에 2번 이상 수행될 수 있다. 이러한 문장을 하드웨어로 컴파일 할 경우, 하나의 회로(circuit)가 한 클럭 안에 2번 수행되어 정상적으로 동작하지 않을 수 있다. 이러한 문제를 회로 중복사용(schizophrenia) 문제라고 부른다. 기존 연구에서는 지역신호선언문과 병렬문만이 회로 중복사용 문제를 유발할 수 있다고 보았다. 하지만, 예외선언문에 의해 생성되는 제어 신호와 출력문이 만들어내는 데이터 신호도 회로 중복사용 문제를 유발할 수 있다. 특히, 출력문의 경우에는 기존 회로 중복사용 문제에 대한 해결책들의 출발점인 단순한 루프 펼치기(loop unrolling)로는 해결되지 않았다. 본 논문에서는 예외선언문과 출력문이 만들어 내는 2가지 새로운 회로 중복사용 문제들을 열거하고 회로 중복사용 문제를 재정의 하였다.

키워드 : Esterel, 동기식 언어, 회로 중복사용, 회로

Abstract Esterel is an imperative synchronous language that is used to develop memories, cache controllers, bus interfaces, and so on. An Esterel statement is called schizophrenic if it is executed more than once in an instant. A schizophrenic statement may cause problems when it is translated to hardware circuits; a circuit performs more than one reaction in a clock. Previous works claim that only local signal declarations and parallel statements may cause schizophrenic problems. However, control signals produced by a trap statement or data signals used by emit statements can cause schizophrenia. They are new schizophrenic patterns. Especially, schizophrenic problems caused by emit statements cannot be solved by a loop unrolling technique that is the key idea of previous curing techniques for schizophrenic problems. In this paper, we introduce and define the two schizophrenic problems.

Key words : Esterel, synchronous language, schizophrenia, circuit

· 본 연구는 지식경제부 및 정보통신산업진흥원이 대학 IT연구센터 지원사업(NIPA-2010-C1090-1031-0004)과 교육과학기술부/한국과학재단 우수연구센터 육성사업(R11-2008-007-02004-0)의 지원으로 수행된 연구임

† 학생회원 : KAIST 전산학과
jeonghan.yun@gmail.com
chuljoo.kim@gmail.com
seonggun.kim@arcs.kaist.ac.kr
†† 종신회원 : KAIST 전산학과 교수
choe@kaist.ac.kr
han@cs.kaist.ac.kr

논문접수 : 2009년 11월 9일
심사완료 : 2010년 2월 9일

정보과학회논문지: 소프트웨어 및 응용 제37권 제4호(2010.4)

1. 서 론

동기식(synchronous) 언어[1,2]는 실시간(real-time) 반응형(reactive) 시스템 개발에 널리 사용되고 있다. 동기식 언어는 실제 시간을 *논리적 단위시간*(instant)으로 나누어, 프로그래밍 언어에서 하드웨어 타이밍 관련 문제들을 쉽게 관리할 수 있게 한다. 생활 이 곳 저 곳에서 내장형 시스템을 사용하면서, 시스템에 대한 요구는 계속 복잡해지고 안전성은 더욱 중요해지고 있다. 동기식 언어는 복잡한 내장형 시스템의 효율적인 개발과 결과물에 대한 안전성 검증에 도움을 준다.

Esterel[3-5]은 절차형(imperative) 동기식 언어이다. Esterel이 가지고 있는 절차형 언어 특징과 선점(preemption) 구조[3]는 프로토콜(protocol)이나 제어 시스템을 디자인하는데 매우 효과적이어서, 주요 반도체 회사에서

컨트롤 로직과 시스템 버스를 디자인 하는 데에 사용하고 있다.[1)]

Esterel로 작성한 프로그램은 Esterel의 정형화된 의미 구조(*formal semantics*)[3,6]를 기반으로 C++, C, SystemC, Verilog, VHDL로 컴파일이 가능하다. 즉, 하드웨어 뿐 아니라 소프트웨어로의 변환이 가능하여, 실제 시스템 구현 및 시뮬레이션을 동시에 할 수 있다.

Esterel을 하드웨어로 컴파일 할 경우 주의해야 하는 대표적인 문제로 *회로 중복사용*(schizophrenia)[3,7]이 있다. Esterel의 절차적 특성 때문에 한 문장이 한 단위 시간 안에 2번 이상 수행될 때 일어나는 문제를 말한다. 회로 중복사용은 Esterel 의미 구조상으로는 문제가 되지 않지만, 게이트 수준의 회로(circuit)로 컴파일 되었을 때에는 전체 시스템이 비정상적으로 동작하는 문제가 발생하기도 한다.

이전 연구[3,7]에서는 루프(loop)에 포함된 지역신호선언문이나 병렬문이 회로 중복사용 문제를 유발할 수 있다고 보았다. 그러므로 회로 중복사용 문제를 검사할 경우 지역신호선언문이나 병렬문이 하나의 단위시간에 두 번 수행되는지 만을 확인하고 있다[7]. 그림 1은 그 대표적인 예제 코드이다. 그림 1(a)는 *지역신호 중복사용(schizophrenic signal declaration)* 의 예제이다. 이 프로그램은 의미 구조상으로는 아무런 문제가 없다. 하지만 이 프로그램을 하드웨어로 컴파일 하였을 경우, 회로에서 지역신호(local signal)의 유효영역(scoping rule)을 표현하는데 문제가 발생한다. 그림 2는 그림 1(a) 프로그램을 회로로 컴파일한 모습이다[3].[2)] 루프가 끝났을 때, 이미 출력된 신호(signal) S는 자신의 유효영역을 벗어났지만 여전히 전기신호로서는 그 값이 유효하다. 그러므로 루프가 다시 시작되었을 때 신호 S의 상태를 테스트할 경우 회로에서는 S가 출력되어 있다고 잘못 판단하게 된다.

그림 1(b)는 병렬문 중복사용(schizophrenic parallel statement)의 예제이다. 이 프로그램은 loop문이 시작할 때 신호 I가 켜져 있고, 그 다음 단위시간에 I가 꺼져

```
loop                          loop
  local signal S in             [ present I then
    present S then                pause
      emit O                    end;
    end;                        V:=V+1
    pause;                    ||
    emit S                      pause
  end                         ]
end                           end
        (a)                           (b)
```

그림 1 회로 중복사용 문제 예제

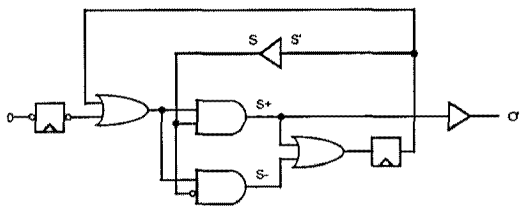


그림 2 그림 1(a)로부터 생성된 회로[3]

있으면 V:=V+1 문장이 하나의 단위시간에 2번 수행된다. 이 경우 V의 값을 한 단위 시간에 두 번 업데이트 하면서 V의 값을 결정할 수 없는 문제가 발생한다.[3)]

Esterel 컴파일러에서는 이러한 문제를 해결해야 한다. 기존 연구에서 여러 가지 해결책들[3,7,8]이 제시되어 있는데, 이 방법들의 기본 아이디어는 루프 펼치기(loop unrolling)이다. 루프 내의 코드를 한 번 복사하여 하나의 문장이 한 단위시간에 두 번 이상 수행되지 않도록 하는 것이다.

본 논문에서는 기존의 회로 중복사용 문제에 대한 연구가 문제점으로 지적하지 못한 새로운 패턴의 회로 중복사용 현상을 제시하고자 한다.

Esterel 프로그램을 회로로 컴파일 할 때 내부적으로 많은 제어 신호가 생성되는데, 이들도 각각 지역 신호의 특징을 가진다. 우리는 *지역신호 중복사용* 문제가 지역신호 선언문에서 사용하는 지역 신호 뿐 아니라 이 제어 신호에서도 발생한다는 것을 발견하였다. 우리는 이것을 *예외 중복사용(schizophrenic trap statement)* 이라 이름 지었다.

또한, 그림 1(b)에서 소개한 문제점은 루프 펼치기 방법으로도 해결되지 않는다. 그러므로 이와 같은 하나의 단위시간 내의 변수 중복 수정 문제는 *병렬문 중복사용*과 분리하여 고려해야 한다. 신호의 중복출력이라는 점에서 우리는 이 문제를 *출력문 중복사용(schizophrenic emit statement)* 이라고 명명하였다.

본 논문은 다음과 같이 구성하였다. 2장에서 Esterel의 의미 구조와 하드웨어 컴파일 방법에 대해 간략히 소개한다. 이를 바탕으로 3장과 4장에서는 각각 예외 중복사용과 출력문 중복사용에 대해 알아본다. 5장에서 새로이 제시한 회로 중복사용 문제들에 대한 탐지 알고리즘을 소개하고, 마지막으로 6장에서 결론 및 향후 연구 방향을 기술한다.

2. Esterel 프로그래밍 언어

2.1 문법 및 의미

본 논문에서는 Pure Esterel의 문장을 대상으로 한다.

1) http://www.synfora.com/

2) Esterel 프로그램을 하드웨어로 컴파일하는 방법은 Berry[3]의 규칙을 기반으로 하였다.

3) 이 외에도 병렬문 중복사용은 동기 장치 중복사용(schizophrenic parallel synchronizer)[3] 문제도 일으키지만 본 논문에서는 생략하였다.

Pure Esterel은 "nothing", "pause", "emit S", "exit T" 등 4개의 단위문장과 신호 테스트, 루프, 연속문, 병렬문, 일시정지문, 지역신호선언문, 예외선언문의 7개 블럭문장으로 이루어져있다. 구체적인 문법과 간단한 의미는 그림 3에 소개되어 있다[3,4]. 여기서 p와 q는 문장을 나타내고, S는 신호, T는 선언된 예외를 의미한다.

nothing	아무 것도 하지 않음
pause	시간 소모
emit S	출력
p; q	연속
present S then p else q	신호 테스트
loop p end	루프 (무한히 반복)
p || q	병렬
suspend p when S	S가 출력되면 일시 중지
signal S in p end	지역신호 선언
trap T in p end	예외 선언
exit T	예외 발생

그림 3 Esterel 문법 및 의미

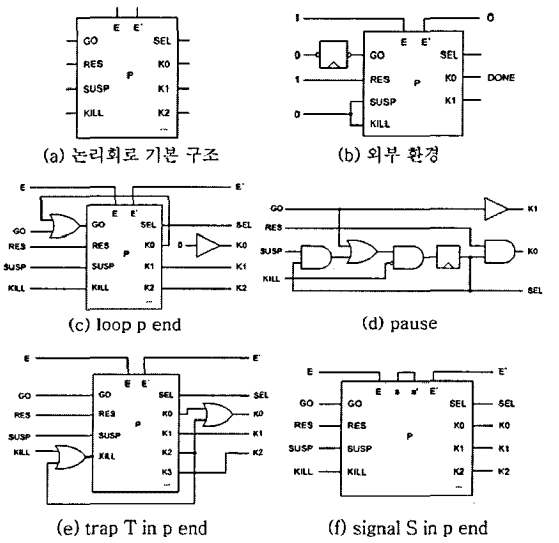


그림 4 Esterel의 회로 변환 규칙[3]

Esterel 프로그램은 외부 클럭에 맞춰 전체 프로그램의 동기화가 이루어진다. 이때 "pause"를 이용해 동기화 시점과 단위시간의 흐름을 결정할 수 있다.

Esterel 프로그램은 신호의 상태에 따라 동작(reaction)한다. 신호의 상태는 단위시간마다 켜지거나(present) 또는 꺼지게(absent) 되는데, "emit S"에 의해 켜지게 되면 단위시간이 끝날 때까지 그 상태가 유지된다.

2.2 회로(circuit)로의 변환

우리는 Berry의 회로 변환 규칙[3]을 기준으로 하였다. 이 규칙은 Esterel 최신 버전[5]에서도 하드웨어 컴파일 규칙의 핵심 알고리즘으로 사용하고 있다[9].

그림 4는 변환 규칙의 일부를 보여준다. (a)는 하나의 Esterel 문장이 회로로 변환될 때의 기본 구조를 보여준다. 각 핀(pin)의 의미는 다음과 같다[3].

- GO: 문장을 시작하라는 입력 제어 신호
- RES: 현 문장의 내부에서부터 다시 수행하라는 입력 제어 신호
- KILL: 예외선언 블럭을 벗어나는 경우 해당 문장을 다음 단위시간에는 수행하지 못하게 하는 입력 제어 신호
- SEL: 다음 단위시간에 해당 문장이 아직 수행할 것이 남아 있음을 나타내는 출력 제어 신호
- K0, K1, ...: K0은 문장 수행이 끝났음을, K1은 pause가 수행되어 현재 단위시간까지의 수행은 끝났으나 다음에 할 일이 남아 있음을 뜻한다. 그리고 2 이상의 Kn은 (n-1)번째 가까운 예외선언 블럭을 빠져나가는 출력 제어 신호
- E, E': 입출력 데이터 신호

프로그램 p를 회로로 변환 시 외부와의 연결은 그림 4(b)와 같이 이루어진다. 첫 번째 클럭에 GO를 켜 준 다음 계속 RES을 켜 주어 매 클럭마다 프로그램을 수행시킨다.

그림 4(c)부터 (f)까지는 루프, pause, 예외선언문, 지역신호선언문이 어떻게 회로로 변화하는지 그림으로 나타내고 있다. 그 외 문장에 대한 변환 규칙들은 여기서 생략하도록 하겠다.

그림 4(c) 루프는 루프가 끝나면 (K0=1) 다시 시작한다.

그림 4(d) pause는 시작하면 (GO=1) K1이 켜지면서, 다음 클럭에 다시 수행해야 함을 레지스터에 저장해 둔다. 다음 클럭에 RES이 켜지면 레지스터가 1일 때만 K0를 출력하여 수행이 끝났음을 알린다.

그림 4(e) 예외선언 블럭은 자신 블럭을 빠져나간다면 (K2=1) 블럭 전체가 끝나면서 (K0=1) KILL을 켜서 다음 단위시간에 수행할 것이 남아있어도 하지 않도록 한다.

그림 4(f) 지역신호는 선언된 블럭 안에서 자신이 출력하는 것을 자신이 사용할 수 있다는 것을 표현한다. 그리고 지역신호는 입출력에 해당하지 않으므로 E, E'에 연결하지 않는다.

3. 예외 중복사용

회로에서는 신호의 유효영역을 표현하기가 어렵다. 그래서 지역신호선언문이 한 단위시간 안에 2번 이상 수행될 경우 회로 중복사용 문제가 발생할 수 있다[3,7].

하나의 문장을 회로로 변환하면, 각 회로마다 컨트롤 신호를 생성하여 사용한다. 이 컨트롤 신호는 문장 단위로 사용되는 지역변수라 할 수 있다.

지금까지 컨트롤 신호 유효영역에 의한 회로 중복사용 문제를 언급한 연구는 없었다. 우리는 여기서 trap 문을 회로로 변환할 때 발생할 수 있는 컨트롤 신호의 유효범위 문제를 소개한다.

3.1 예제

예외 발생을 통해 예외선언 블럭문을 빠져 나오는 경우 예외선언 블럭 안의 레지스터들이 다음 단위시간에는 1로 되지 못하도록 KILL 신호를 켠다. 이 때 켜진 KILL 신호는 지역변수와 같아서 현재 예외선언 블럭을 빠져나가면 효과가 없어져야 한다.

의미 구조상 예외선언 블럭에서 빠져 나온 후 다시 들어갔을 때는 서로 다른 예외선언 블럭의 인스턴스(instance)라 할 수 있다. 그러므로 이전 블럭에서의 KILL 신호는 다시 블럭에 들어갔을 때는 효과가 없어야 한다. 하지만, 생성된 예외선언 블럭 전자회로에서 한 클록 안에 켜진 신호가 꺼지지 않는다. 그러므로 이전에 켜진 KILL 신호에 의해 다시 들어간 예외선언 블럭도 빠져나가야 하는 것으로 잘못 인식한다.

그림 5는 한 단위시간 안에 2번 수행되는 예외선언 블럭에 대한 간단한 예제를 보여 주고 있다. 그리고 그림 6[4]은 그림 5의 예제 프로그램들을 Berry의 하드웨어 변환 규칙에 의해 생성된 회로를 보여 준다.

의미 구조상 그림 5의 두 프로그램은 아무런 문제가 없이 동작한다. 두 프로그램 모두 매 단위시간마다 O 신호를 출력하는 프로그램이다. 하지만, 그림 5(a)의 경우, exit T에 의해 켜진 KILL신호는 trap 블럭에 다시 들어갔을 때까지 켜진 상태로 유지되어 오동작을 일으킨다. 그림 6(a)에서 보듯이, 예외선언 블럭을 빠져 나올 때 켠 KILL 신호가 계속 레지스터에 영향을 미쳐, 다음 단위시간에 레지스터 값이 0이 된다. 그림 4(d)의 pause문을 보면 알 수 있듯이 레지스터는 프로그램의 제어흐름이 어디까지 왔는가를 기억해야 한다. 레지스터가 제어흐름을 기억하지 못하면 그 이후 전자회로 전체가 종료된 것과 같이 되어 버린다.

그림 5(b)는 조금 다르다. 한 단위시간 안에 예외선언 블럭에 다시 들어가는 것은 같다. 하지만 KILL 신호를 켜지 않으므로, 다시 시작하는 예외선언 블럭은 그림 6(b)에서처럼 이전 KILL 신호에 의해 오동작하지 않는다. 이는 지역신호 중복사용과 비슷한 특징이다[10].

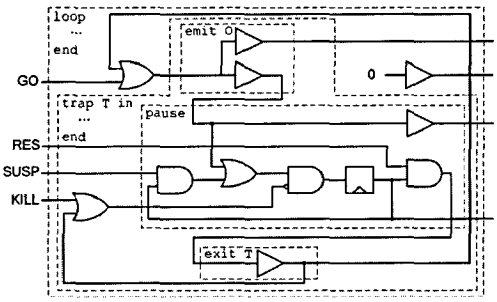
3.2 문제 정의

예외선언 블럭에 한 단위시간 안에 2번 이상 들어갈 경우 예외선언 블럭도 여러 개의 인스턴스가 존재하게 된다. 하나의 예외선언 블럭 인스턴스에서 출력된 KILL 신호가 다른 예외선언 블럭 인스턴스에 영향을 끼칠 경우 전체 회로는 오동작을 일으킨다. 모든 인스턴스에서 KILL 신호가 출력되지 않았다면 서로에게 영향을 끼칠 신호가 존재하지 않게 되어, 회로는 정상 동작한다.

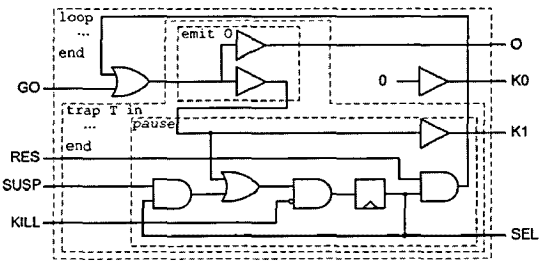
KILL 신호는 예외선언 블럭을 빠져나갈 때 출력한다. 그러므로 우리는 예외 중복사용을 다음과 같이 정의한다.

```
loop                    loop
  trap T in               trap T in
    emit O;                 emit O;
    pause;                  pause;
    exit T                  end
  end                     end
end
     (a)                     (b)
```

그림 5 두 번 수행되는 예외선언문



(a) 그림 6(a)의 회로



(b) 그림 6(b)의 회로

그림 6 그림 5로부터 생성된 회로

정의 1. 하나의 예외선언문이 한 단위시간 안에 2번 이상 수행되고, 그 중 한 번이라도 해당 예외선언문을 빠져나가는 경우(exit)가 있다면, 그 예외선언문은 *예외 중복사용*이라고 한다.

4. 출력문 중복사용

한 단위시간 안에 하나의 신호가 2번 이상 출력될 경우, 쓰기-쓰기 충돌이 발생할 수 있다. 이전에는 이와 같은 문제가 *병렬문 중복사용*[7,10]의 일부로 간주되었다. 하지만, 우리는 이 문제가 이와 별도로 관리해야 하는 이유를 두 가지 발견하였다.

4) 일반적인 컴파일 형태를 보이기 위해 외부 환경과의 연결을 위한 부분을 생략하였다.

본 논문에서 데이터 관련 문장을 고려하지 않아 두번 수행되어 문제를 일으키는 데이터 출력 문장을 출력문(emit)으로 제한하였다.

4.1 예제

그림 7(a)는 그림 1(b)에서 살펴 본 병렬문 중복사용 예제[7]에서 V:=V+1 문장을 emit O로 변경한 것이다. 그림 7(b)는 (a)와 같은 반응을 보이는 프로그램이다. 첫 번째 단위시간에 신호 I가 켜져 있고 두 번째 단위시간에 신호 I가 꺼져 있으면, emit O는 두 번째 단위시간에 두 번 수행되게 된다.

과거의 연구에서는 데이터 출력 문장이 두 번 이상 수행되는 경우를 병렬문과 연관해 생각했으나[7,10], 이 예제는 병렬문 없이도 그러한 문제가 일어날 수 있음을 보여준다.

더 중요한 것은, 데이터 출력 문장이 두 번 이상 수행되는 문제는 기존 회로 중복사용 해결책으로 해결되지 않는다는 점이다. 회로 중복사용의 가장 큰 특징은 한 문장이 회로로 변경되었을 때, 한 클록 안에 하나의 회로가 두번 이상 사용된다는 것이다. 그러므로 기본적으로 루프 내용 전체를 복사하게 되면 이와 같은 문제는 해결된다.

```
loop
[ present I then
    pause
    end;
    emit O
||
  pause
]
end
```

(a)

```
loop
 signal S in
  emit S;
  present I then
    pause
  end;
  emit O;
  present S then
    pause
  end
 end
end
```

(b)

그림 7 두 번 수행되는 emit 문장

```
loop
[ present I then
    pause
  end;
  emit O
||
  pause
];
[ present I then
    pause
  end;
  emit O
||
  pause
]
end
```

(a)

```
loop
 signal S1 in
  emit S1;
  present I then
    pause
  end;
  emit O;
  present S1 then
    pause
  end
 end;
 signal S2 in
  emit S2;
  present I then
    pause
  end;
  emit O;
  present S2 then
    pause
  end
 end
end
```

(b)

그림 8 루프 펼치기된 코드

하지만, 그림 8과 같이 루프 내용을 복사하더라도 첫 번째 단위시간에 신호 I가 켜져 있고, 다음에 I가 꺼져 있으면 emit O는 두 번 수행된다. 지역신호선언문과 병렬문은 복사를 하면 각각이 자신의 회로를 생성하여 한 클록 안에 하나의 회로를 다시 사용하는 문제가 없어진다. 하지만, emit O와 같이 데이터와 연관된 내용은 그 자체가 복사되더라도 여전히 쓰기-쓰기 충돌과 같은 문제는 남아있다. 이 문제를 해결하기 위해서는 프로그래머가 하나의 신호가 중복 출력될 경우 그 값을 결정할 수 있는 기능(combine function)을 소스 코드에 추가해야 한다[5].

4.2 문제 정의

데이터 출력 문장을 출력문만 고려하였으므로 우리는 출력문을 기준으로 문제를 재정의하였다.

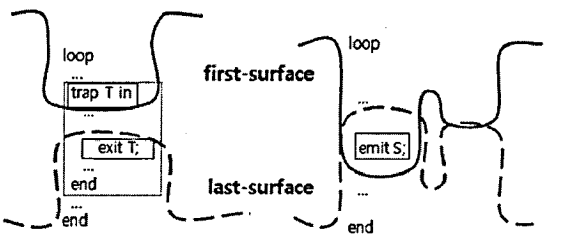
정의 2. 하나의 출력문이 한 단위시간 안에 두 번 이상 수행되면 *출력문 중복사용*이라고 한다.

5. 탐지 알고리즘

우리는 이전 연구에서 Esterel에 대해 CFG를 생성하고[11], CFG 상에서 루프의 시작 영역(first-surface)와 끝 영역(last-surface)에 대한 정의 및 계산 알고리즘을 제시하였다[10]. 시작 영역은 루프가 시작하는 단위시간에 CFG 상에서 수행될 수 있는 노드들의 집합을 말하며, 끝 영역은 루프 바디가 끝날 때 수행될 수 있는 노드들의 집합을 뜻한다. 루프가 끝나고 다시 시작할 경우 시작 영역의 노드와 끝 영역의 노드들은 동시에 수행될 가능성을 가지게 되어 회로 중복사용을 일으킬 수 있는 노드들을 모두 포함하게 된다.

그러므로 그림 9와 같이 시작 영역, 끝 영역을 이용하면 지금까지 제시한 두 가지 회로 중복사용 문제 탐지 알고리즘을 구현할 수 있다.

예외선언 블럭의 시작 노드가 시작 영역에 들어오고, 해당 exit문이 끝 영역에 들어오면 예외 발생으로 빠져나온 후 바로 예외선언 블럭에 들어갈 수 있으므로 예외 중복사용이다. 또한 하나의 출력문이 시작 영역과 끝



(a) Schizophrenic trap statement (b) Schizophrenic emit statement

그림 9 탐지 알고리즘

영역에 동시에 포함되면, 한 단위시간에 2번 이상 수행될 수 있음을 뜻한다.

6. 결론 및 향후 연구 과제

우리는 Esterel 프로그램을 하드웨어로 컴파일 할 때 발생할 수 있는 회로 중복사용 문제에서, 기존 연구들이 놓치고 있던 문제점 2가지를 소개하였다.

본 논문을 바탕으로 향후 연구 목표는 다음과 같다.

먼저, 본 논문에서 5장에서 소개한 탐지하는 알고리즘을 구현하려 한다. 우리의 이전 연구[10,11]를 활용하여 알고리즘을 구현하고, 이를 실제 프로그램에 적용해 보고자 한다.

더 나아가 명확히 정의되어 있지 않은 회로 중복사용 문제들을 엄밀하게 정의하여, 각각의 문제점을 찾아내는 분석기를 제작하고자 한다. 회로 중복사용 문제의 해결책은 회로 복사로부터 출발한다. 회로 중복사용 분석기는 Esterel 프로그램을 하드웨어로 컴파일 할 때 회로 중복사용 문제를 일으킬 수 있는 프로그램과 그렇지 않은 프로그램을 구분해 주므로, 필요 없는 회로의 증가를 줄여 줄 것이다.

참 고 문 헌

[1] N. Halbwachs, *Synchronous Programming of Reactive Systems*, Kluwer Academic Publishers, 1993.

[2] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE Embedded Systems*, vol.91(1), pp.64-83, 2003.

[3] G. Berry, *The Constructive Semantics of Pure Esterel*. Draft book available at http://www.inria.fr/meije/esterel/esterel-eng.html, 1999.

[4] D. Potop-Butucaru, S. A. Edwards, and G. Berry. *Compiling Esterel*, Springer, 2007.

[5] Esterel Technologies, *The Esterel v7 Reference Manual Version v7.30. initial IEEE standardization proposal*, Esterel Technologies, 679 av. Dr. J. Lefebvre 06270 VilleneuveLoubet, France, November 2005.

[6] O. Tardieu, "A deterministic logical semantics for pure Esterel," *ACM Transactions on Programming Languages and Systems*, vol.29(2), pp.1-24, 2007.

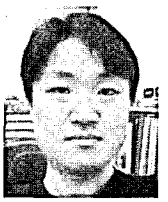
[7] O. Tardieu and R. de Simone, "Loops in Esterel," *Transactions on Embedded Computing Systems*, vol.4, no.4, pp.708-750, 2005.

[8] K. Schneider, J. Brandt, and T. Schuele, "A verified compiler for synchronous programs with local declarations," *Electronic Notes in Theoretical Computer Science*, vol.153, no.4, pp.71-97, 2006.

[9] G. Berry, "Circuit design and verification with Esterel v7 and Esterel Studio," *IEEE International High-Level Design, Validation, and Test Workshop*, pp.133-136, 2007.

[10] J. Yun, C. Kim, S. Seo, T. Han, and K. Choe, "Refining schizophrenia via graph reachability in Esterel," *7th ACM-IEEE International Conference on Formal Methods and Models for Codesign*, 2009.

[11] C. Kim, J. Yun, S. Seo, K. Choe, and T. Han, "Efficient construction of over-approximated CFG on Esterel," *Journal of KIISE: Computing Practices and Letters*, vol.15(11), pp.876-880, 2009. (in Korean)



윤 정 한

2001년 KAIST 전산학전공 학사. 2003년 KAIST 전산학전공 석사. 2003년~현재 KAIST 전산학과 박사과정. 관심분야는 임베디드 시스템, 프로그램 분석

김 칠 주

2001년 동국대학교 컴퓨터공학전공 학사 2003년 KAIST 전산학전공 석사. 2003년~2010년 KAIST 전산학과 박사. 현재 삼성전자 연구원. 관심분야는 프로그래밍 언어, 프로그램 정적 분석, 컴파일러 최적화

김 성 건

2003년 KAIST 전기 및 전자공학과 학사. 2005년~현재 KAIST 전산학과 석박사 통합과정. 관심분야는 병렬처리, 멀티코어 아키텍처, 최적화 컴파일러

최 광 무

1976년 서울대학교 전자공학과 학사. 1978년 KAIST 전산학과 석사. 1984년 KAIST 전산학과 박사. 1984년~현재 KAIST 전산학과 교수. 관심분야는 정형언어 이론, 로직프로그램의 병렬화, 컴파일러

한 태 숙

1976년 서울대학교 전자공학과 학사. 1978년 KAIST 전산학과 석사. 1990년 Univ. of North Carolina at Chapel Hill 박사 1991년~현재 KAIST 전산학전공 교수 관심분야는 프로그래밍 언어, 함수형 언어, 임베디드 시스템