Esterel 문법구조 바탕의 오토마타 생성

(Syntax-driven Automata Generation for Esterel)

이 철 우 † 김 철 주 ††
(Chul-Woo Lee) (Chul-Joo Kim)
윤 정 한 † 한 태 숙 †††
(Jeong-Han Yun) (Taisook Han)
최 광 무 †††
(Kwang-Moo Choe)

요 약 Esterel 언어는 동기(synchronous)식 절차(imperative)형 언어로 유한 상태 기계(finite state machine)를 기반으로 한 정형적 의미구조를 가지고 있어 오토마타를 이용한 프로그램 분석에 매우 용이하다. 본 논문에서는 프로그램의 수행 과정이 필요 없는 문법 구조 바탕의 오토마타 생성 규칙을 제안한다. 우리의 생성 규칙은 문법 구조를 직관적으로 표현하고 모든 가능한 경로를 나타내기 때문에 다양한 분석을 적용하기에 적합하다.

키워드 : Esterel, 오토마타

Abstract Esterel is an imperative synchronous language and its formal semantic based on finite state machine makes it easy to perform program analyses using automata. In this paper, we propose a syntax-driven automata generation rule. Because our rule intuitively expresses syntactic structure, it is very useful for other program analyses.

Key words : Esterel, Automata

· 2010년도 정부(교육과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(No. 2010-0000258)
· 본 연구는 교육과학기술부/한국연구재단 우수연구센터 육성사업(2010-0001712)과 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업(NIPA-2010-C1090-1031-0004)의 지원으로 수행되었음
· 이 논문은 2010 한국컴퓨터종합학술대회에서 'Esterel 문법구조 바탕의 오토마타 생성'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : KAIST 전산학과
posticaruss@gmail.com
dolgam@gmail.com
†† 정 회 원 : 삼성전자 DMC 연구소 책임연구원
chuljoo.kim@gmail.com
††† 종신회원 : KAIST 전산학과 교수
han@cs.kaist.ac.kr
choe@kaist.ac.kr

논문접수 : 2010년 8월 10일
심사완료 : 2010년 10월 5일

Copyright©2010 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.
정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제11호(2010.11)

1. 서 론

Esterel[1,2]은 반응(reactive)형 실시간(real-time) 시스템을 설계하기 위한 절차형 동기식 언어[이다. Esterel은 유한 상태 기계를 기반으로 한 정형적(formal) 의미구조를 가지고 있어 정적 검증에 매우 용이하다. 유한 상태 기계와 동일한 의미를 가지고 있기 때문에 Esterel 프로그램을 오토마타로 변환하면 기존의 오토마타 이론이나 모델 체킹 기법들을 이용한 분석이 가능하다[3].

Esterel 컴파일러[4,5]는 프로그램의 시뮬레이션을 통해 오토마타를 결과물로 얻을 수 있는 옵션을 지원한다. 시뮬레이션 과정에서 여러 가지 최적화 기법들을 이용하고 매 단위시간마다 도달 가능한 모든 상태만을 고려하기 때문에 비교적 빠른 시간 안에 오토마타를 생성한다. 하지만 프로그램의 오토마타 변환은 상태 폭발 문제에 영향을 받기 때문에 큰 프로그램을 대상으로 하기는 현실적으로 어렵다[1].

기존 연구[6]에서 Esterel 프로그램의 동작 의미(operational semantics)를 기술하기 위하여 오토마타를 LTS[7] 형태로 정의하였다. 즉, LTS로 기술된 오토마타는 프로그램의 의미에 따라 수행되는 진행 과정을 정형적으로 표현한다. 또한 오토마타 생성 규칙은 잉여의 정보를 제거하여 본래의 구조와는 다르지만 동일한 동작 의미를 지니는 단순한 구조로 나타내도록 정의하고 있다. 하지만 이 생성 규칙을 바탕으로 오토마타를 생성하기 위해서는 외부 환경에 대한 고려가 필요하다. 프로그램의 동작 의미가 외부 환경에 따라 달라지기 때문이다. 결과적으로 프로그램 전체 구조를 나타내는 오토마타를 직접적으로 표현하기 위해서는 가능한 모든 외부 환경에 대한 프로그램의 수행 동작을 살펴봐야 한다. 결국 이 오토마타 생성 방법은 시뮬레이션과 유사한 추가적인 프로그램 수행 과정을 필요로 한다.

위와 같은 프로그램 수행 과정을 거치지 않고 오토마타를 생성하기 위해서 본 논문에서는 새로운 오토마타 생성 규칙을 제안한다. 우리의 생성 규칙은 프로그램의 수행 과정 없이 문법 구조를 바탕으로 오토마타를 생성한다. 또한 우리의 오토마타 생성 규칙을 바탕으로 생성 과정 중에 오토마타 축약 과정을 적용할 수 있다.

본 논문은 2장에서 Esterel 언어를 소개한다. 3장에서는 오토마타 생성 규칙을 제안하고 생성 규칙을 통해

표 1 Esterel 문장

문장	의미
emit S	signal emission
pause	consuming a clock tick
nothing	no operation
exit t	exception raise
p; q	sequence of statements
present S then p else q end	signal test
p || q	parallel execution
trap t in p end	exception declaration
suspend p when S	suspending program
loop p end	nonterminating loop

생성된 Esterel 프로그램에 대한 오토마타 예를 보여준다. 결론 및 향후 연구 방향은 4장에서 논의한다.

2. Esterel

본 논문에서는 "signal S in p end"을 제외한 kernel Esterel[2]을 다룬다. 문법 소개는 표 1에 소개되어 있다. 문법과 의미를 설명하기에 앞서 S는 시그널, p와 q는 문장을 나타낼 때 사용하였다. t는 예외 발생 식별자로 사용한다.

Esterel은 동시성 언어로서 매 단위시간을 표현하기 위해 "pause"를 사용한다. "pause"와 "pause" 사이는 하나의 단위시간이다. "pause"에 도달 했을 때만 한 단위시간을 소모한다. 한 단위시간 내에서는 "emit S"가 수행되어야 시그널 S가 켜진다. 시그널의 상태는 단위시간 안에서만 유지된다. "nothing"은 아무 일도 하지 않는 문장이다.

"p; q"는 문장 사이의 순서를 나타낸다. p를 수행한 뒤에 순차적으로 q를 수행 한다. "present S then p else q end"는 시그널 S의 상태에 따라 p 또는 q를 실행한다. "p || q"는 p와 q를 병렬 수행한다. 두 문장 모두 종료되어야 "p || q"가 종료된다. "loop p end"는 p를 무한히 반복 수행한다. "suspend p when S"는 시그널 S가 켜진 상태인 단위시간 동안 p의 수행을 멈춘다. "trap t in p end"는 새로운 예외(exception) t를 선언한다. p를 수행하는 동안 "exit t"에 의한 예외가 발생하면 즉시 p 수행을 멈추고 trap문을 종료한다.

3. 오토마타 생성

Esterel 프로그램을 위한 오토마타는 그림 1과 같이 정의한다. 오토마타는 Q,Σ,δ,s,f로 이루어진 튜플(tuple)로 정의된다.

Q는 노드(node)들의 집합이다. 노드는 일반(normal) 노드와 트랩(trap) 노드로 나누어진다. 트랩 노드는

$$Automata ::= \langle Q,\Sigma,\delta,s,f\rangle$$

$$\begin{pmatrix} Q: \text{a set of nodes} & \Sigma: \text{a set of signal states} \\ \delta: \text{a set of edges} & s: \text{a start node} \\ f: \text{a final node} & \end{pmatrix}$$

$Q = Q_N \cup Q_{tr}$, where $Q_N \subseteq D$, $Q_{tr} \subseteq T \times D$

Q_N = a set of normal nodes — Q_{tr} = a set of trap nodes

D = a set of indexes — T = a set of trap identifiers

signal state = S or ¬S — S = a signal identifier

edge = $i \xrightarrow{I/O} j$ — i, j = a node

O = a set of output signal states

I = a set of input signal states

그림 1 오토마타 정의

"exit t"에서 발생한 예외(t)를 나타내기 위해 트랩 식별자(t)와 인덱스의 튜플로 나타낸다. s,f는 각각 프로그램 시작 노드와 종료 노드를 나타낸다.

Σ는 프로그램에서 사용된 시그널들의 상태 집합이다. 시그널 S의 켜진 상태는 S, 꺼진 상태는 ¬S로 나타낸다. δ는 두 노드 사이의 간선(edge)을 의미하고, 단위시간 동안 입출력되는 시그널들의 상태가 표현된다.

3.1 오토마타 생성 규칙

주어진 프로그램의 오토마타 생성 규칙은 다음과 같은 형식으로 표현된다.

$$program \rhd \langle Q,\Sigma,\delta,s,f\rangle$$

문장의 오토마타를 생성할 때 기본적으로 고려해야 할 점 두 가지가 있다.

첫 번째, Esterel 프로그램은 적어도 하나의 단위시간을 소모한다. 프로그램 전체에서 "pause"를 한 번도 사용하지 않았다면 이 프로그램은 한 단위시간만을 소모하는 프로그램이 된다.

두 번째, 문장들의 결합을 통한 블록문장을 생성할 때의 동일한 단위시간에 일어나는 시그널 입출력을 일치시켜 주어야 한다. 그림 2의 예제를 살펴보자. "pause; emit O1;" 과 "emit O2; pause;" 두 문장의 오토마타는 (1)과 (2)에 나타나 있다. 위의 두 문장을 순차적으로 수행할 때 시그널 O1과 O2는 동일한 단위시간에 켜진 상태로 존재한다. 이 문장에 대한 오토마타를 표현하기 위해서는 (1)의 마지막 단위시간과 (2)의 첫 번째 단위시간을 일치시켜주어야 한다. 이를 고려한 합성된 결과 오토마타는 (3)과 같은 형태로 표현되어 시그널 O1과 O2가 동일한 단일시간에 나타나 있음을 볼 수 있다.

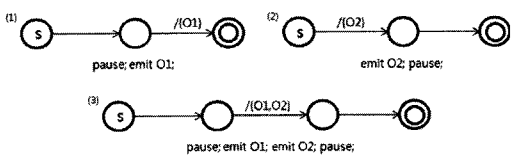

그림 2 오토마타 예제

위와 같은 점들을 고려하여 우리는 Esterel의 문법구조에서부터 직접 오토마타를 생성하는 규칙을 제안한다. 그림 3은 각 문장에 대한 생성 규칙을 보여주고 있다. 오토마타 생성규칙의 형식은 [8]에서 정의한 형식을 참고하였다.

"emit S"는 간선에 시그널 S의 켜진 상태를 추가한다. "pause"는 시작 노드와 종료 노드 사이에 중간 노드(k)를 두어 추가된 단위시간을 표현한다. "nothing"은 "emit S"와 동일하지만 시그널 상태에 영향을 주지 않는다. "exit t"를 뒤따르는 문장은 더 이상 실행되지 않는다. 이를 나타내기 위해 트랩 노드를 추가해 "trap t in p end"의 바깥 간선과 연결될 수 있게 한다.

$$(nothing) \qquad nothing \triangleright \langle\{i,j\},\emptyset,\{i \xrightarrow{\{\}/\{\}} j\},i,j\rangle$$

$$(pause) \qquad pause \triangleright \langle\{i,j,k\},\emptyset,\{i \xrightarrow{\{\}/\{\}} k, k \xrightarrow{\{\}/\{\}} j\},i,j\}\rangle$$

$$(emit) \qquad emit\ \mathsf{S} \triangleright \langle\{i,j\},\{\mathsf{S}\},\{i \xrightarrow{\{\}/\{\mathsf{S}\}} j\},i,j\rangle$$

$$(exit) \qquad exit\ \mathsf{t} \triangleright \langle\{i,j,(\mathsf{t},k)\},\emptyset,\{i \xrightarrow{\{\}/\{\}} (\mathsf{t},k)\},i,j\rangle$$

$$(sequence) \qquad \frac{p \triangleright \langle Q_p,\Sigma_p,\delta_p,s_p,f_p\rangle \quad q \triangleright \langle Q_q,\Sigma_q,\delta_q,s_q,f_q\rangle}{p\ ;\ q \triangleright \langle(Q_p \cup Q_q)\backslash\{f_p,s_q\},\Sigma_p \cup \Sigma_q,\delta,s_p,f_q\rangle}$$

$$where \quad \delta = ((\delta_p \cup \delta_q)\backslash D_r) \cup (\langle Q_p,\Sigma_p,\delta_p,s_p,f_p\rangle \oplus \langle Q_q,\Sigma_q,\delta_q,s_q,f_q\rangle),$$

$$D_r = \{i \xrightarrow{I/O} f_p \in \delta_p\} \cup \{s_q \xrightarrow{I/O} j \in \delta_q\},$$

$$\langle Q_p,\Sigma_p,\delta_p,s_p,f_p\rangle \oplus \langle Q_q,\Sigma_q,\delta_q,s_q,f_q\rangle = \{i \xrightarrow{\{I_1 \cup I_2\}/\{O_1 \cup O_2\}} j \mid i \xrightarrow{I_1/O_1} f_p \in \delta_p \wedge s_q \xrightarrow{I_2/O_2} j \in \delta_q\}$$

$$(present) \qquad \frac{p \triangleright \langle Q_p,\Sigma_p,\delta_p,s_p,f_p\rangle \quad q \triangleright \langle Q_q,\Sigma_q,\delta_q,s_q,f_q\rangle}{present\ \mathsf{S}\ then\ p\ else\ q\ end \triangleright \langle Q,\Sigma_p \cup \Sigma_q \cup \{\mathsf{S},\neg\mathsf{S}\},\delta,s,f\rangle}$$

$$where \quad Q = (Q_p \cup Q_q \cup \{s,f\})\backslash\{s_p,s_q,f_p,f_q\},$$

$$\delta = (\delta_1/D_{r_2}) \cup D_{a_2},\ \delta_1 = ((\delta_p \cup \delta_q)\backslash D_{r_1}) \cup D_{a_1},$$

$$D_{r_1} = \{s_p \xrightarrow{I/O} j \in \delta_p\} \cup \{s_q \xrightarrow{I/O} j \in \delta_q\},$$

$$D_{r_2} = \{i \xrightarrow{I/O} f_p \in \delta_1\} \cup \{i \xrightarrow{I/O} f_q \in \delta_1\},$$

$$D_{a_1} = \{s \xrightarrow{I \cup \{\mathsf{S}\}/O} j \mid s_p \xrightarrow{I/O} j \in \delta_p\} \cup \{s \xrightarrow{I \cup \{\neg\mathsf{S}\}/O} j \mid s_q \xrightarrow{I/O} j \in \delta_q\}$$

$$D_{a_2} = \{i \xrightarrow{I/O} f \mid i \xrightarrow{I/O} f_p \in \delta_1\} \cup \{i \xrightarrow{I/O} f \mid i \xrightarrow{I/O} f_q \in \delta_1\}$$

$$(trap) \qquad \frac{p \triangleright \langle Q_p,\Sigma_p,\delta_p,s_p,f_p\rangle}{trap\ \mathsf{t}\ in\ p\ end \triangleright \langle Q_p\backslash\{(\mathsf{t},k)\},\Sigma_p,\delta,s_p,f_p\rangle}$$

$$where\ \delta = (\delta_p \setminus \{i \xrightarrow{I/O} (\mathsf{t},k) \in \delta_p\}) \cup \{i \xrightarrow{I/O} f_p \mid i \xrightarrow{I/O} (\mathsf{t},k) \in \delta_p\}$$

$$(suspend) \qquad \frac{p \triangleright \langle Q_p,\Sigma_p,\delta_p,s_p,f_p\rangle}{suspend\ p\ when\ \mathsf{S} \triangleright \langle Q_p,\Sigma_p \cup \{\mathsf{S},\neg\mathsf{S}\},(\delta_p\backslash D_r) \cup D_a,s_p,f_p\rangle}$$

$$where \quad D_r = \{i \xrightarrow{I/O} j \in \delta_p \mid i \neq s_p\},$$

$$D_a = \{i \xrightarrow{\{\mathsf{S}\}/O} i \mid i \in Q_p \setminus \{s_p,f_p\}\} \cup \{i \xrightarrow{I \cup \{\neg\mathsf{S}\}/O} j \mid i \xrightarrow{I/O} j \in \delta_p \wedge i \neq s_p\}$$

$$(parallel) \qquad \frac{p \triangleright \langle Q_p,\Sigma_p,\delta_p,s_p,f_p\rangle \quad q \triangleright \langle Q_q,\Sigma_q,\delta_q,s_q,f_q\rangle}{p \parallel q \triangleright GenerateParallel(\langle Q_p,\Sigma_p,\delta_p,s_p,f_p\rangle,\langle Q_q,\Sigma_q,\delta_q,s_q,f_q\rangle)}$$

$$(loop) \qquad \frac{p \triangleright \langle Q_p,\Sigma_p,\delta_p,s_p,f_p\rangle}{loop\ p\ end \triangleright \langle(Q_p \cup \{f\})\backslash\{f_p\},\Sigma_p,(\delta_p\backslash D_r) \cup (\langle Q_p,\Sigma_p,\delta_p,s_p,f_p\rangle \oplus \langle Q_p,\Sigma_p,\delta_p,s_p,f_p\rangle),s_p,f\rangle}$$

$$where \quad D_r = \{i \xrightarrow{I/O} f_p \in \delta_p\}$$

그림 3 오토마타 생성 규칙

"p; q"에서 두 문장 p와 q 사이를 연결해 주기 위해서 연산자 '⊕'를 정의한다. 이 연산자는 p의 종료 노드(f_p)와 q의 시작 노드(s_q)에 연결되어 있는 노드들 사이에 간선을 생성한다.

"present S then p else q end"는 시그널 S의 상태에 따른 분기를 나타낸다. p, q의 시작노드 (s_p), (s_q)는 새로운 시작 노드(s)로 합쳐진다. 새로운 시작 노드에서 p로 연결되는 첫 단위시간을 나타내는 간선에는 시그널 S의 켜진 상태를 추가한다. q로 연결되는 첫 단위 시간을 나타내는 간선에는 시그널 S의 꺼진 상태를 추가한다.

"p || q"에 대한 자세한 설명은 3.2절에서 다룬다.

"loop p end"는 p를 무한히 반복 수행한다. 이는 p를 순차적으로 다시 실행한다는 의미로 생각할 수 있다. 때문에 p의 마지막 단위 시간과 시작 단위 시간은 동일한 단위 시간이 된다. 위에서 정의된 연산자 '⊕'를 이용하여 이 두 단위시간을 하나의 단위시간으로 표현한다.

"suspend p when S"는 시작 노드(s_p)와 종료 노드(f_p)를 제외한 내부 노드들에 시그널 S의 켜진 상태를 나타내는 재귀간선들을 추가한다. p의 내부 간선들에는 시그널 S의 꺼진 상태를 추가한다.

"trap t in p end"는 p 내부에서 "exit t"문에서 생성된 트랩 노드에 연결된 간선들을 trap문의 종료 노드(f_p)에 연결하고 트랩 노드를 삭제한다.

3.2 병렬 수행 오토마타 생성

그림 4의 알고리즘은 병렬 수행하게 되는 두 오토마타 A_p와 A_q를 입력으로 받고, 병렬 수행을 표현하는 하나의 오토마타 A_f를 출력한다. A_p와 A_q의 종료 시점으로 인한 수행 차이를 고려하여 우리 알고리즘은 크게 두 부분으로 나누어 병렬 수행 오토마타 생성 과정을 수행한다. 알고리즘의 전반부는 A_p와 A_q에서 병렬 수행될 수 있는 부분을 대상으로 오토마타 생성 과정을 수행 한다. 수행이 남은 오토마타의 노드를 F에 저장한다. 알고리즘의 후반부는 병렬 수행 도중 한 오토마타의 종료 노드 도달로 인하여 수행이 남은 다른 오토마타만을 대상으로 오토마타 생성 과정을 수행한다. F에 저장된 노드들로부터 시작하여 A_f의 새로운 노드와 간선 생성을 이어서 진행하게 된다.

새로운 간선은 병렬 수행되는 두 간선을 합성하여 생성된다. 이때 간선내의 시그널들의 상태들도 합쳐지게 된다. 새로운 노드는 그림 5의 노드 생성 함수들로 생성한다. 일반 노드 생성 함수에서 고려해야 할 점은 종료 노드 도달 여부이다. 예를 들어 (i,j)에서 가 종료 노드인 경우 i가 속한 오토마타를 대상으로 i에서부터 A_f의 생성을 이어가야 한다. 때문에 이러한 경우는 새로운 노드를 생성하지 않고 i를 반환해 준다.

Algorithm pseudocode for generating Parallel Automata

```
input : Automata ⟨Q_p, Σ_p, δ_p, s_p, f_p⟩
        Automata ⟨Q_q, Σ_q, δ_q, s_q, f_q⟩
output : Q, Σ_p ∪ Σ_q, δ, s, f
W : Worklist
 1:  push (s_p, s_q) into W
 2:  while W is not empty
 3:    pop (i, j) in W
 4:    if i = f_p ∨ j = f_q
 5:      if (i, j) ≠ (f_p, f_q)
 6:        push (i, j) into F
 7:      end if
 8:    end if
 9:    k = getID1(i, j)
10:    push k into Q
11:    N_p = successor(i)
12:    N_q = successor(j)
13:    for (n, m) ∈ N_p × N_q
14:      if isTrapNode(n) ∧ isTrapNode(m)
15:        r = getTrapID1(n, m)
16:        put r into Q
17:      else if isTrapNode(n)
18:        r = getTrapID2(n)
19:        put r into Q
20:      else if isTrapNode(m)
21:        r = getTrapID2(m)
22:        put r into Q
23:      else
24:        if getID1(n, m) ∉ Q
25:          r = getID1(n, m)
26:          put (n, m) into W
27:        end if
28:      end if
29:      put
30:      {k --I1∪I2/O1∪O2--> r | i --I1/O1--> n ∈ δ_p ∧ j --I2/O2--> m ∈ δ_q}
31:      into δ
32:    end for
33:  end while
34:  if ¬isEmpty(F)
35:    for (i, j) ∈ F
36:      n = (i ≠ f_p ? i : j)
37:      push n into W
38:      while W is not empty
39:        pop k in W
40:        push getID2(k) into Q
41:        N = successor(k)
42:        for n ∈ N
43:          if isTrapNode(n)
44:            r = getTrapID2(n)
45:            put n into Q
46:          else
47:            if n ∉ Q
48:              r = getID2(n)
49:              put n into W
50:            end if
51:          end if
52:          put {getID2(k) --I/O--> r | k --I/O--> n ∈ δ_p ∪ δ_q} into δ
53:        end for
54:      end while
55:    end for
56:  end if
57:  if f ∉ Q
58:    put f into Q
59:  end if
```

그림 4 병렬 수행 오토마타 생성 알고리즘 의사 코드

Functions for getting a node	

getID1(i, j) :
if $i = s_p \wedge j = s_q$
  return s
else if $i = f_p \wedge j = f_q$
  return f
else if $i = f_p$
  return j
else if $j = f_q$
  return i
else
  return a new normal node
end if
	getID2(i) :
	if $i = f_p \vee i = f_q$
	  return f
	else
	  return a new normal node
	end if

getTrapID1((t_1, k_1), (t_2, k_2)) :
if t_1 is outer than t_2
  return (t_1, k_1)
else
  return (t_2, k_2)
end if
	getTrapID2(i) :
	  return a new trap node

그림 5 노드 생성 함수

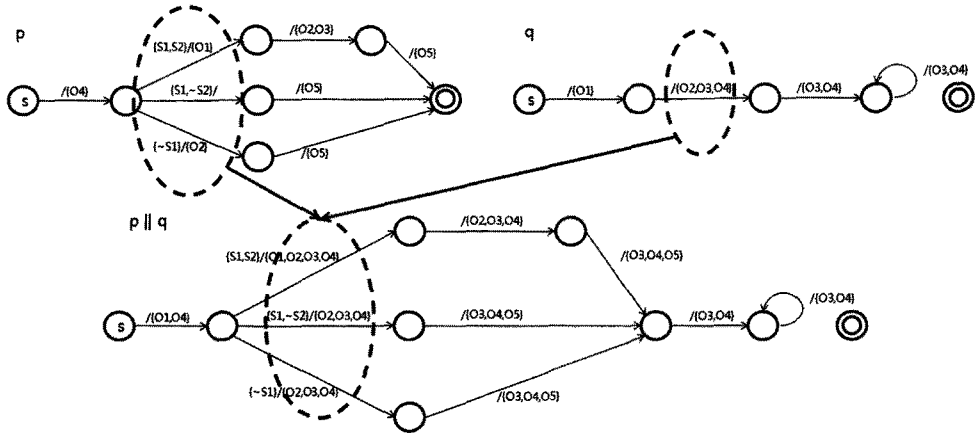

그림 6 그림 7의 오토마타 예제

3.3 오토마타 생성 예제

그림 7의 Esterel 프로그램은 p와 q를 병렬 수행하는 프로그램이다. 그림 6에서는 본 논문에서 제시한 오토마타 생성 알고리즘을 이용하여 그림 7의 p, q, 그리고 "p || q"에 대한 오토마타를 보여준다. "p || q"의 오토마타에서 시그널 S1과 S1에 대한 조건 분기는 두 번째 단위 시간에 표현되어 있다. 병렬 수행 시 q는 종료되지 않고 무한히 반복되기 때문에 "p || q"의 오토마타는 종료 노드에 도달하지 못하고 무한히 반복 수행된다. 전체 프로그램의 두 번째 단위시간을 살펴보자. p에서 시그널 S1과 S2가 켜짐 상태일 때 시그널 O1이 출력된다. q에서는 시그널 O2, O3, O4이 출력된다. 때문에 p와 q를 병렬 수행 시 시그널 S1과 S2가 켜짐 상태일 때 시그널 O1, O2, O3, O4는 켜짐 상태로 존재한다. "p || q"의 오토마타를 살펴보면 두 번째 단위 시간들 중에서 가장 위의 간선이 이를 정확하게 표현하고 있음을 알 수 있다.

```
p
emit O4;
pause; %(1)
present S1 then
  present S2 then
    emit O1; pause; %(2)
    emit O2;
    emit O3; pause; %(3)
  else
    pause; %(4)
  end
else
  emit O2; pause; %(5)
end
emit O5; pause; %(6)

||

q
emit O1; pause %(7)
emit O2;
loop
  emit O3; emit O4; pause; %(8)
end
```

그림 7 Esterel 프로그램 예제

4. 결론 및 향후 연구

본 논문에서는 Esterel 문법 구조 바탕의 오토마타 생성 규칙을 제안하였다. 제안한 규칙들은 Esterel 프로그램의 수행 과정 없이 직접적으로 프로그램의 전체 구조를 표현할 수 있는 오토마타를 생성한다. 생성된 오토마타는 반응 시스템 상에서 일어나는 각 단위시간에서의 입력 시그널과 출력 시그널의 상태들을 나타낸다.

현 오토마타 생성 규칙에서는 의미구조상 수행 불가능한 경우를 제외하지 않았다. 추후 이에 대한 분석을 통해서 생성된 오토마타를 최적화 할 수 있는 방법에 대한 연구를 진행할 계획이다. 이를 바탕으로 생성 규칙 바탕의 요약 분석이나 생성된 오토마타를 이용하여 Esterel 에서 나타날 수 있는 특이한 성질들을 분석할 수 있는 분석 도구를 개발할 예정이다.

참 고 문 헌

[1] D. Potop-Butucaru, S. A. Edwards, and G. Berry, "Compiling ESTEREL," Springer, 2007.

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

[3] A. Bouali, "XEVE, an ESTEREL Verification Environment," In Proceedings of International Conference on Computer Aided Verification, vol.1427, pp.500-504, 1998.

[4] "Esterel Compiler version 5.21," http://www.inria.fr/meije/esterel/esterel-eng.html.

[5] "CEC: The Columbia Esterel Compiler," http://www.cs.columbia.edu/~sedwards/cec/.

[6] S. Tini, "An axiomatic semantics for Esterel," Theoretical Computer Science, 2001.

[7] GD Plotkin, "A Structural Approach to Operational Semantics," Lectures Notes, Aarhus University, Aarhus, Denmark, 1981.

[8] 김철주, 윤정한, 서선애, 최광무, 한태숙, "Esterel에서 근사-제어흐름그래프의 효율적인 생성," *정보과학회논문지 : 컴퓨팅의 실제 및 레터*, 제15권 제11호, pp.801-880, 2009.