Esterel에서 근사-제어흐름그래프의 효율적인 생성
(Efficient Construction of Over-approximated CFG on Esterel)

김 철 주 † (Chul-Joo Kim)　윤 정 한 † (Jeong-Han Yun)
서 선 애 †† (Sunae Seo)　최 광 무 ††† (Kwang-Moo Choe)
한 태 숙 ††† (Taisook Han)

요 약 프로그램에 대한 자료흐름분석(data flow analysis)를 수행하기 위해서는 입력된 프로그램에 대응하는 제어흐름그래프(control flow graph)가 필요하다. 본 논문에서는 동기(synchronous)식 절차(imperative)형 언어 중 하나인 Esterel로 작성된 프로그램에 대해서 단순하면서 입력 프로그램의 구조와 흡사한 형태로 표현되는 근사-제어흐름그래프(over-approximated CFG) 생성방법을 제안한다. 제안된 방법을 이용하면 병렬 제어흐름을 표현하는 부분에서 실행 불가능한 경우까지 포함할 수 있다. 그렇지만, 생성방법이 직관적이고, 실제 수행경로를 모두 포함하기 때문에 다른 분석을 수행하는데 매우 적합하다.

키워드 : Esterel, 제어흐름그래프, 동기식언어

Abstract A control flow graph(CFG) is an essential data structure for program analyses based on graph theory or control-/data- flow analyses. Esterel is an imperative synchronous language and its synchronous parallelism makes it difficult to construct a CFG of an Esterel program. In this work, we present a method to construct over-approximated CFGs for Esterel. Our method is very intuitive and generated CFGs include not only exposed paths but also invisible ones. Though the CFGs may contain some inexecutable paths due to complex combinations of parallelism and exception handling, they are very useful for other program analyses.

Key words : Esterel, Control Flow Graph, Synchronous Language

· 본 연구는 교육과학기술부/한국과학재단 우수연구센터 육성사업(R11-2008-007-02004-0)의 지원과 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업(NIPA-2009-C1090-0902-0020) 및 2008년 정부(교육과학기술부)의 재원으로 한국학술진흥재단(KRF-2008-313-D00968)의 지원으로 수행되었음
· 이 논문은 2009 한국컴퓨터종합학술대회에서 'Esterel에서 근사-제어흐름그래프의 효율적인 생성'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : KAIST 전산학과
chuljoo.kim@gmail.com
jeonghan.yun@gmail.com
†† 비 회 원 : 삼성종합기술원 연구원
sunae.seo@samsung.com
††† 종신회원 : KAIST 전산학과 교수
choe@kaist.ac.kr
han@cs.kaist.ac.kr

논문접수 : 2009년 8월 13일
심사완료 : 2009년 10월 1일

Copyright©2009 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 제15권 제11호(2009.11)

1. 서 론

Esterel[1-4]은 반응형(reactive) 실시간(real time) 시스템을 설계하기 위한 동기식(synchronous) 언어[5-7]이다. 특히, Esterel의 절차적(imperative)인 특징으로 인해 프로그램 내에 제어흐름(control flow)이 존재하고, 이를 이용해 자료흐름분석(data flow analysis)과 같은 정적분석(static analysis)을 수행할 수 있다.

자료흐름분석을 수행하려면 주어진 프로그램에 대한 제어흐름그래프(control flow graph: CFG)가 필요하다. Esterel에서 단순하고 효율적인 CFG 생성이 어려운 근본적인 이유는 병렬수행(parallel execution)때문이다. 한 프로그램 위치에서 발생한 제어흐름이 병렬로 수행중인 다른 프로그램 위치에 영향을 줄 수 있어서 프로그램 전체의 제어흐름을 추적하는 것이 쉽지 않다.

하지만, 주목해야 할 점은 Esterel이 동기식 언어이기 때문에 외부 클럭(clock)에 맞춰서 동기화가 이루어지고 제어흐름의 변화도 클럭 단위로 나타난다는 점이다. 따라서, 기존의 일반적인 CFG 생성방법에 동기화 시점의 제어흐름을 추가해 주는 것으로 근사-제어흐름그래프(over-approximated CFG)를 생성할 수 있다.

본 논문은 2장에서 Esterel에 대한 간략한 소개를 하고 3장에서 관련된 기존 연구에 관해 논의한 후에 4장에서 CFG 생성방법을 제안한다. 5장에서는 제안된 방법의 구현과 실험결과를 보여주고 결론 및 향후 방향에 대한 논의는 6장에서 이루어진다.

2. Esterel 프로그래밍 언어

본 논문에서는 Pure Esterel을 대상으로 한다. Pure Esterel은 "`nothing`", "`pause`", "`emit` *s*", "`exit` *t*" 등

4개의 단위문장과 signal test, loop, sequence, parallel, suspension, exception handling 등의 7개의 블록문장으로 이루어져 있다. 구체적인 문법과 간단한 의미는 2.1절과 그림 1에 소개되어 있다. 여기서 *p*와 *q*는 문장을 나타내고, *s*는 signal, *t*는 exception을 의미한다.

2.1 Pure Esterel 문장

Esterel은 외부 클럭에 맞춰서 동기화가 이루어진다. 이때 "pause"를 이용해 동기화 시점을 결정하고 단위시간(instant)의 흐름을 제어할 수 있다. "pause"는 단위시간을 변경하고, 모든 signal의 상태를 초기화 한다.

Esterel 프로그램은 signal의 상태에 따라 동작(reaction)한다. signal의 상태는 단위시간마다 present 또는 absent가 되는데, "emit *s*"에 의해 present가 되면 단위시간이 끝날 때까지 그 상태가 유지된다.

다른 문장들의 직관적인 의미는 다음과 같다. "nothing"은 아무 일도 하지 않고, "*p*;*q*"는 *p*를 수행한 후 즉시(instantly) *q*를 수행한다. "present *s* then *p* else *q*"는 *s*의 상태에 따라 *p* 또는 *q*를 수행한다. "loop *p* end"는 *p*를 무한히(infinitely) 반복하고, "*p*||*q*"는 *p*와 *q*를 병렬로 수행하고, 두 문장이 모두 종료되어야 병렬수행이 끝난다. "signal *s* in *p* end"는 *p*를 수행하는 동안에만 signal *s*를 유효(scope)하게 한다. "suspend *p* when *s*"는 *s*가 present하는 단위시간 동안 *p*의 수행을 멈춘다. "trap *t* in *p* end"는 새로운 예외(exception) *t*를 선언하고, 이는 *p*를 수행하는 동안 유효하다. 만일, *p*에서 "exit *t*"에 의해 예외가 발생하면, 그 즉시 *p*의 수행을 멈추고 "trap"을 종료한다.

2.2 Esterel에서 제어흐름 결정 시점

Esterel에서 선점(preemption)[8]이 일어나는 경우를 제외한 모든 제어흐름은 단위시간에 맞춰서 순차적으로 결정된다. 하지만, "suspend"나 "exit"과 같이 이를 방해하는 선점 문장에 의해 제어흐름은 바뀔 수 있다.

강한 선점(strong preemption)

선점조건이 발생하는 즉시 남아있는 수행을 멈추고 선점에 해당하는 일을 수행하는 것이다. "suspend *p* when *s*"에서 *s*가 present인 경우 그 즉시 *p*의 수행을 멈추고, *s*의 상태가 absent가 될 때까지 기다려야 한다.

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

그림 1 Pure Esterel 문장

약한 선점(weak preemption)

선점조건이 발생하더라도, 현재 단위시간까지는 수행을 마친 후에 선점된 일을 수행하는 경우에 해당된다. "*p*||*q*"에서 *p*가 "exit *t*"에 의해 종료 된 경우, 동기화가 이루어지기 전까지 *q*는 종료 사실을 인지하지 못한다. 현재 단위시간에 하도록 정의된 일을 모두 수행한 후에, 비로소 "exit *t*"에 의해 종료되었음을 인식하고 뒤따르는 일의 유무와 관계없이 종료 된다.

3. 관련 연구

CEC[9,10]에서는 Esterel을 컴파일하는 과정에서 GRC[11]를 이용한 CFG를 생성한다. 생성된 CFG는 매 단위시간에 시작 노드부터 끝 노드까지 전체가 한번 수행한 후에 계산된 상태는 저장공간에 기록되고, 이는 다음 단위시간의 제어흐름을 결정하는데 이용된다. 원본 프로그램으로부터 문법적 변환을 거쳐 얻을 수 있긴 하지만 원본프로그램의 절차적인 특성과 구조가 모두 제거된 형태이기 때문에 프로그램분석에서 필요로 하는 제어흐름을 보여주진 못한다.

Esterel과 유사한 Quartz에 관한 연구[12]에서 CEC와 유사한 접근을 하고 있으나 그래프 대신 프리디킷(predicate)과 논리식으로 제어흐름을 표현한다. 이는 명제증명기(theorem prover)를 통한 분석에 사용된다.

[13,14]에서는 Esterel 프로그램 슬라이서(slicer)를 구현하면서 CFG를 사용하였다. 이는 가장 기본적인 형태로서 병렬 수행중인 문장들 간에 발생하는 제어흐름의 간섭은 그래프에 직접 나타나지 않는다. 이를 해결하기 위해 그래프를 해석하는 단계에서 "pause" 처리기(pause handler)와 결정자(decision maker)를 도입하여 해결하고 있다. 이는 PDG(program dependency graph)와 함께 사용하여 슬라이서를 구현하는 데에는 도움을 주지만 일반적인 프로그램분석을 수행하기에 적절하지 않다.

4. 제어흐름그래프 생성

CFG 생성과정을 단순화하기 위해, 프로그램 내의 모든 식별자(identifier)는 유일하며 같은 이름을 여러 번 사용하지 않는다고 가정한다. 식별자가 사용될 때 범위규칙(scoping rule)에 의해 중첩되는 식별자는 유효하지 않게 되므로, 간단한 조작을 통해 이를 제거할 수 있다.

그리고, Esterel에서 유효하지 않는 예외 *t*에 대한 "exit *t*"는 "nothing"과 동일한 의미를 갖는다. 이 역시 전처리 과정을 통해 교체가 가능하므로 이러한 경우는 고려하지 않는다.

4.1 정의

CFG는 그림 2에 정의된 바와 같이 *s*, *f*, *N*, *E*, *W*로 구성된 튜플로 표현된다.

$$\textit{Control Flow Graph} ::= \langle s,\ f,\ N,\ E,\ W \rangle$$

$$\left(\begin{array}{llll} s: & \text{index of start } node & f: & \text{index of finish } node \\ N: & \text{a set of } node & E: & \text{a set of } edge \\ W: & \text{a set of } (\mathtt{t},\ i,\ [\curvearrowright|\rightsquigarrow]) & & \end{array}\right)$$

$$\begin{array}{rcll} node & ::= & \overset{i}{type} & \\ type & ::= & \mathtt{nothing} \mid \mathtt{pause} \mid \mathtt{emit\ s} \mid \mathtt{exit\ t} & \\ & | & \mathtt{B} & \text{(entry node)} \\ & | & \mathtt{E} & \text{(exit node)} \\ edge & ::= & i \xrightarrow{l} j & \text{(normal edge)} \\ & | & i \twoheadrightarrow j & \text{(parallel edge)} \\ & | & i \curvearrowright j & \text{(exit edge)} \\ & | & i \rightsquigarrow j & \text{(may exit edge)} \\ & | & i \nrightarrow j & \text{(unreachable edge)} \\ i,\ j & ::= & \text{index of } node & l\ ::=\ \epsilon \mid \mathtt{s} \mid \neg\mathtt{s} \\ \mathtt{s} & ::= & \text{signal identifier} & \mathtt{t}\ ::=\ \text{trap identifier} \end{array}$$

그림 2 제어흐름그래프

s와 f는 각각 시작 노드와 끝 노드의 인덱스를 나타내고, N과 E는 각각 노드(*node*)와 엣지(*edge*)의 집합이다. 노드는 식별자 역할을 하게 될 정수 인덱스(i)와 타입(*type*)으로 구성된다. 노드의 타입에는 단위문장을 의미하는 4개의 타입과 블록의 시작(B)과 끝(E)을 표현하는 타입이 포함된다. 엣지는 5종류가 있는데, 출발노드(i)와 대상노드(j), 조건(l)로 이루어진 *normal* 엣지 및 출발노드와 대상노드로만 이루어진 *parallel, exit, may-exit, unreachable* 엣지가 있고 각각, 병렬수행, 예외발생, 예외발생가능, 수행불가능 한 경로를 의미한다. 조건은 signal의 존재여부를 표시하거나 필요 없는 경우에는 사용하지 않는다. W는 "exit"에서 발생한 예외(t)와 발생한 노드의 인덱스(i), 엣지의 집합인데, W의 활용에 대해서는 4.2절과 4.3절에서 설명한다.

4.2 생성규칙

주어진 program에 대응되는 CFG를 생성하는 규칙은 아래와 같은 형식으로 표현된다.

$$program \triangleright \langle s, f, N, E, W \rangle$$

그림 3에서 각 문장에 대한 생성규칙을 나타낸다.

- "nothing", "pause", "emit s"는 각각을 의미하는 노드 하나로 이루어진 CFG가 된다.
- "exit t"는 뒤따르는 문장이 더 이상 수행되지 않으므로 exit t 노드(s)와 더미노드(f)를 이용해서 두 노드 사이를 *unreachable* 엣지로 연결한다. "trap t"의 바깥으로 제어흐름을 이어줘야 하므로, 식별자 t와 노드 인덱스, exit 엣지로 구성된 튜플을 W에 추가한다.
- "p;q"는 하위문장 p와 q에 대응되는 CFG를 완성하고 p의 끝(f_p)과 q의 시작(s_q) 사이를 *normal* 엣지로 연결해 준다. p의 시작(s_p)이 결과의 시작이 되고 q의 끝이 결과의 끝(f_q)이 된다.
- "p||q"는 4.3절에서 자세히 다룬다.

$$(nothing)\quad \mathtt{nothing} \triangleright \langle i,\ i,\ \{\overset{i}{\mathtt{nothing}}\},\ \emptyset,\ \emptyset \rangle$$

$$(pause)\quad \mathtt{pause} \triangleright \langle i,\ i,\ \{\overset{i}{\mathtt{pause}}\},\ \emptyset,\ \emptyset \rangle$$

$$(emit)\quad \mathtt{emit\ s} \triangleright \langle i,\ i,\ \{\overset{i}{\mathtt{emit}}\ \mathtt{s}\},\ \emptyset,\ \emptyset \rangle$$

$$(exit)\quad \mathtt{exit\ t} \triangleright \langle s,\ f,\ \{\overset{s}{\mathtt{exit}}\ \mathtt{t},\ \overset{f}{\mathtt{E}}\},\ \{s \nrightarrow f\},\ \{(\mathtt{t}, s, \curvearrowright)\} \rangle$$

$$(sequence)\quad \frac{p \triangleright \langle s_p,\ f_p,\ N_p,\ E_p,\ W_p \rangle \quad q \triangleright \langle s_q,\ f_q,\ N_q,\ E_q,\ W_q \rangle}{p;q \triangleright \langle s_p,\ f_q,\ N_p \cup N_q,\ E_p \cup E_q \cup \{f_p \rightarrow s_q\},\ W_p \cup W_q \rangle}$$

$$(parallel)\quad \frac{p \triangleright \langle s_p,\ f_p,\ N_p,\ E_p,\ W_p \rangle \quad q \triangleright \langle s_q,\ f_q,\ N_q,\ E_q,\ W_q \rangle}{\begin{array}{l} p \,||\, q \triangleright \langle s_p,\ f_q,\ N_p \cup N_q \cup \{\overset{s}{\mathtt{B}},\ \overset{f}{\mathtt{E}}\},\ E_p \cup E_q \cup \{s \twoheadrightarrow s_p,\ s \twoheadrightarrow s_q,\ f_p \rightarrow f,\ f_q \rightarrow f\} \\ \qquad ,\ add_may(W_p, E_q, f_q) \cup add_may(W_q, E_p, f_p) \rangle \end{array}}$$

$$\begin{array}{l} \text{where, } add_may(W, E', f') \doteq W \cup \bigcup_{(\mathtt{t},i,-)\in W} \{(\mathtt{t}, j, \rightsquigarrow) \mid (j \rightarrow f' \in E') \\ \qquad \vee\ (j \rightarrow k \in E' \wedge \overset{k}{\mathtt{pause}} \in N') \\ \qquad \vee\ (j \rightarrow k \in E' \wedge \underset{k}{\mathtt{exit}}\ \mathtt{t}' \wedge \mathtt{t} \text{ is outer than } \mathtt{t}')\} \end{array}$$

$$(loop)\quad \frac{p \triangleright \langle s_p,\ f_p,\ N_p,\ E_p,\ W_p \rangle}{\mathtt{loop}\ p\ \mathtt{end} \triangleright \langle s,\ f,\ N_p \cup \{\overset{s}{\mathtt{B}},\ \overset{f}{\mathtt{E}}\},\ E_p \cup \{s \rightarrow s_p,\ f_p \rightarrow s_p,\ f_p \nrightarrow f\},\ W_p \rangle}$$

$$(present)\quad \frac{p \triangleright \langle s_p,\ f_p,\ N_p,\ E_p,\ W_p \rangle \quad q \triangleright \langle s_q,\ f_q,\ N_q,\ E_q,\ W_q \rangle}{\begin{array}{l} \mathtt{present\ s\ then}\ p\ \mathtt{else}\ q \triangleright \langle s,\ f,\ N_p \cup N_q \cup \{\overset{s}{\mathtt{B}},\ \overset{f}{\mathtt{E}}\} \\ \qquad ,\ E_p \cup E_q \cup \{s \xrightarrow{\mathtt{s}} s_p,\ s \xrightarrow{\neg\mathtt{s}} s_q,\ f_p \rightarrow f,\ f_q \rightarrow f\},\ W_p \cup W_q \rangle \end{array}}$$

$$(signal)\quad \frac{p \triangleright \langle s_p,\ f_p,\ N_p,\ E_p,\ W_p \rangle}{\mathtt{signal\ s\ in}\ p\ \mathtt{end} \triangleright \langle s,\ f,\ N_p \cup \{\overset{s}{\mathtt{B}},\ \overset{f}{\mathtt{E}}\},\ E_p \cup \{s \rightarrow s_p,\ f_p \rightarrow f\},\ W_p \rangle}$$

$$(suspend)\quad \frac{p \triangleright \langle s_p,\ f_p,\ N_p,\ E_p,\ W_p \rangle}{\mathtt{suspend}\ p\ \mathtt{when\ s} \triangleright \langle s,\ f,\ N_p \cup \{\overset{s}{\mathtt{B}},\ \overset{f}{\mathtt{E}}\},\ adjust_edge(N_p, E_p) \cup \{s \rightarrow s_p,\ f_p \rightarrow f\},\ W_p \rangle}$$

$$\text{where, } adjust_edge(N, E) \doteq \forall \overset{i}{\mathtt{pause}} \in N.\ \forall (i \rightarrow j) \in E.\ (E / \{i \rightarrow j\}) \cup \{i \xrightarrow{\neg\mathtt{s}} j,\ i \xrightarrow{\mathtt{s}} i\}$$

$$(trap)\quad \frac{p \triangleright \langle s_p,\ f_p,\ N_p,\ E_p,\ W_p \rangle}{\begin{array}{l} \mathtt{trap\ t\ in}\ p\ \mathtt{end} \triangleright \langle s,\ f,\ N_p \cup \{\overset{s}{\mathtt{B}},\ \overset{f}{\mathtt{E}}\},\ add_exit(\mathtt{t}, E_p, W_p, f) \cup \{s \rightarrow s_p,\ f_p \rightarrow f\} \\ \qquad ,\ W_p / \{(\mathtt{t}', -, -) \in W_p \mid \mathtt{t}' \doteq \mathtt{t}\} \rangle \end{array}}$$

$$\text{where, } add_exit(\mathtt{t}, E, W, f) \doteq Ep \cup \{i \curvearrowright f \mid (\mathtt{t}, i, \curvearrowright) \in W\} \cup \{i \rightsquigarrow f \mid (\mathtt{t}, i, \rightsquigarrow) \in W\}$$

그림 3 CFG 생성

- "loop p end"는 무한반복을 표현하기 위해, p의 끝(f_p)과 시작(s_p)을 *normal* 엣지로 연결한다. 결과의 시작(s)과 p의 시작(s_p)을 연결하고, p의 끝(f_p)과 결과의 끝(f) 사이는 *unreachable* 엣지가 된다.
- "present s then p else q"는 p와 q에 대응하는 CFG에 대해 결과의 시작(s)과 p의 시작(s_p)은 조건 s가 포함된 *normal* 엣지가 연결되고, q의 시작(s_q) 대해서는 조건 $\neg s$가 포함된다. 그리고, 각각의 끝과 결과의 끝을 연결하다.
- "signal s in p end"는 식별자의 중복을 없애는 처리를 이미 마쳤으므로 별다른 조작 없이 p의 시작(s_p)과 끝(f_p)을 결과(s, f)에 연결한다.
- "suspend p when s"는 먼저 p에 상응하는 CFG를 생성하고, 결과의 시작(s)과 끝(f)을 p의 시작(s_p)과 끝(f_p)에 연결한다. 그리고, p에 포함된 모든 pause에 대해 signal s가 present인 동안 수행이 지연되어야 하는 "suspend"의 의미를 유지할 수 있도록 s의 present를 뜻하는 재귀엣지($i \rightarrow i$)를 추가한다. 동시에 pause를 출발로 하는 엣지($i - j$)에 대해서는 s의 absent를 조건으로 덧붙여 준다.
- "trap t in p end"도 다른 블록문처럼 p의 시작(s_p)과 끝(f_p)을 결과(s, f)에 연결한다. 그리고, p에서 발생한 예외 W 중 식별자가 t인 경우에 대한 처리를 해주어야 한다. 해당 예외에 대해서는 예외가 발생한

노드(i)와 결과의 끝(f)을 *exit* 또는 *may-exit* 엣지를 이용해서 연결해 준다.

4.3 병렬수행에서 CFG 생성규칙

2.2절에 언급되어 있듯이, Esterel에서 선점이 발생한 경우 각 문장의 의미에 맞게 처리해야 한다. 그림 4의 예를 보면, "exit *T*"와 "exit *U*"는 같은 단위시간에 수행된다. "trap *U*"가 "trap *T*"를 포함하고 있으므로, "exit *U*"가 우선순위가 높고, 전체 제어는 "trap *U*" 바깥으로 흘러간다.

```
1:  trap U in
2:     trap T in
3:        emit A; pause; exit T
4:       ||
5:        emit B; pause; exit U
6:     end trap
7:  end trap
```

그림 4 중첩된 trap

실제로 "$p||q$"에 대해, p에서 "exit"이 수행된 경우, q는 현재 단위시간까지 수행을 마치고 종료하거나, 마지막 단위시간인 경우 q의 끝까지 수행하고 종료한다. 만약 q에서 다른 "exit"이 같은 단위시간에 수행된 경우 두 "exit" 중에서 더 바깥쪽 "trap"에 해당하는 "exit"을 처리한다.

본 논문에서, 이를 표현하기 위하여 *may-exit* 엣지를 추가하였다. *may-exit* 엣지란, 병렬수행에서 한쪽 프로그램에서 "exit"에 의해 수행이 종료된 경우 다른 한쪽 프로그램의 제어가 종료될 가능성이 있는 문장에 대한 엣지를 의미한다.

그림 3의 "$p||q$" 규칙을 보면, p와 q에 대응하는 CFG를 계산하고, 각각의 시작(s_p, s_q)과 끝(f_p, f_q)을 결과의 시작(s)과 끝(f)에 *parallel* 엣지로 연결해 준다. 그리고, p에 포함된 exit의 집합인 W_p에 대해, q의 엣지의 집합인 E_q를 참조하여 다음의 각 노드에 *may-exit* 엣지를 추가한다.

- q의 마지막 노드의 선행노드(j)
- q에 포함된 pause의 선행노드(j)
- q에 포함된 exit t 중 우선순위가 낮은 노드(j)

W_q와 E_p에 대해서도 동일한 절차를 진행한다.

4.4 생성된 CFG의 안전성(soundness)

본 논문에서 제안한 방법으로 CFG를 생성할 때, 앞 절에서 언급된 병렬수행 중 "exit"을 포함한 경우 *may-exit* 엣지로 인해, 실제로는 불필요한 엣지가 추가될 수 있다. 하지만, Esterel의 언어 정의와 의미[1-4]에 기반하여 모든 수행 가능한 경로를 빠트림 없이 포함하고 있으므로 안전하다. 그리고, 이러한 문장이 포함된 경우를 제외하면, 실제 실행 중에 나타날 수 있는 경로와 동일한 CFG를 얻을 수 있다.

5. 구 현

본 논문에서 제안된 CFG 생성방법은 OCaml로 구현되었으며 여러 모듈로 작성된 프로그램은 CEC[9]를 이용해서 하나의 모듈로 합쳤다(dismantle). 파싱이 이루어지는 전단부는 Esterel 추가문장들을 Pure Esterel로 녹여(de-sugaring) 받아들이도록 구현하였다.

그림 5는 그림 4의 프로그램에 대한 CFG이다. 실선, 굵은 실선, 점선, 가는 점선, 가는 실선은 각각 normal, parallel, exit, may-exit, unreachable 엣지를 의미한다.

표 1은 Esterel 벤치마크 프로그램 Estbench[15]와 Ramesh's case study[16]에 대한 실험결과이다. *may-exit* 엣지가 하나도 생성되지 않은 mca200과 dlx의 경우 mca200은 병렬수행을 포함한 "trap"을 사용하지 않았으며, dlx는 "trap" 자체를 사용하지 않았다. tcint, ww, fbus는 각각 여러 형태의 "trap"을 사용하고 있지만, 병렬수행을 포함하면서 "exit"이 수행되는 형태는 생각만큼 많지 않았다. 하지만 atds100과 mejia는 전체 엣지의 약 13~14%가 *may-exit*이라고 인식이 되었는데, 이는 그림 6의 mejia의 예와 같이 "trap", 병렬수행, "exit"들이 다양한 조합으로 구성되어 있기 때문이다. "await"는 원하는 signal이 present로 바뀔 때까지 기다리는 추가문장인데, 17번째 줄의 "exit"이 2, 6, 10번 줄 "await"의 선행 노드와 각 병렬수행문의 끝에 총 9개의 *may-exit* 엣지를 삽입하게 된다.

대부분 실제로 수행될 가능성이 있는 엣지이나 일부 전혀 실행이 불가능한 것도 있다. 이는 데이터에 의존적인(data dependent) 제어흐름에서 나타나는데, 이를 제거하는 방법에 관해서는 향후 연구에서 다룰 예정이다.

4장의 생성 규칙으로 CFG를 만드는 데에는 프로그램 크기에 비례하는 정도의 시간과 메모리가 소모된다.

6. 결 론

본 논문에서는 Esterel의 특징들을 다 표현하면서 손쉽게 사용 가능한 근사-제어흐름그래프의 생성방법에 대해 제안하였다. 생성된 CFG는, 몇몇 경우에 대해서는 근사값(over-approximation)으로 나타나지만, 기본적으로 실제 실행을 모두 포함하고 있기 때문에 안전하다.

그리고, 실험적으로 확인해본 바에 의하면 예상만큼 큰 근사(approximation)가 적용되지는 않음을 알 수 있었다. 또한, CFG 생성자체에 부하가 걸리지 않고, 원본 프로그램의 문장구조를 정확하게 유지하기 때문에, 이를 응용하는 분석에서 손쉽게 활용이 가능하다.

추후에 전체 Esterel 문장을 모두 포함하는 시스템으로 확장할 계획이며, CFG에 기반한 다양한 분석시스템에 적용할 예정이다.

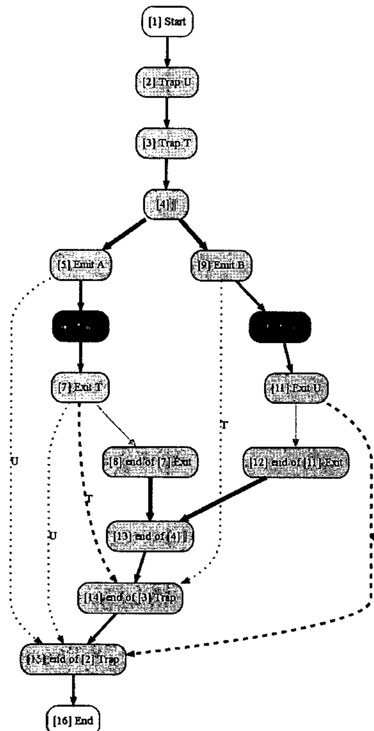

그림 5 중첩된 trap의 CFG

표 1 실험결과

Programs	Size	Lines	Nodes	Edges	May edges
atds100	22,038	622	987	1,716	223
mca200	227,599	5,354	1,037	1,440	-
mejia	9,782	361	510	782	94
tcint	9,364	353	504	779	33
ww	11,952	360	591	861	14
dlx	7,862	334	426	618	-
fbus	6,287	285	673	943	21
Total	294,857	7,673	4,728	7,139	385

```
 1: trap DETEC_ERR in
 2:     await ERR do
 3:        emit ERR_RECEPTION
 4:     end await
 5:   ||
 6:     await DD do
 7:        emit ERR_RECEPTION
 8:     end await
 9:   ||
10:     await
11:        case DF do
12:           emit ERR_RECEPTION
13:        case ATTEND_DF
14:     end await
15:   ||
16:     await ERR_RECEPTION do
17:        exit DETEC_ERR
18:     end await
19: end trap
```

그림 6 mejia 예제

참 고 문 헌

[1] G. Berry, "The Esterel Primer," Included in the Esterel distribution, Available on http://www.inria.fr/meije/personnel/Gerard.Berry.html, 1998.

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

[3] G. Berry, "The Foundations of Esterel. Proof, Language and Interaction: Essays in Honour of Robin Milner," pp.425-454, 2000.

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

[5] N. Halbwachs, "Synchronous Programming of Reactive Systems," Kluwer Academic Publishers, Dordrecht, 1993.

[6] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone, "The Synchronous Languages 12 Years Later," *Embedded Systems, Proceedings of the IEEE*, 91(1):64-83, 2003.

[7] S. Benvenistre and G. Berry, "The synchronous approach to reactive real-time systems," *Another Look of Real Time Programming, Proceedings of the IEEE*, 79(9):1270-1282, 1991.

[8] G. Berry, "Preemption in concurrent systems," In *Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science*, pp.72-93, Springer-Verlag, London, UK, 1993.

[9] "CEC: The columbia esterel compiler v0.4," http://www1.cs.columbia.edu/~sedeards/cec/

[10] S. A. Edwards and Jia Zeng, "Code Generation in the Columbia Esterel Compiler," *EURASIP Journal On Embedded Systems*, Volume 2007, Article ID 52651, pp.31, 2007.

[11] D. Potop-Butucaru, "Optimizations for faster execution of Esterel programs," in *Proceedings of the 1st International Conference on Formal Methods and Models for Codesign (MEMOCODE'03)*, pp.227-236, Mont St. Michel, France, June 2003.

[12] K. Schneider, "Embedding Imperative Synchronous Languages in Interactive Theorem Provers," ACSD, pp.143, Second International Conference on Application of Concurrency to System Design (ACSD'01), 2001.

[13] S. Ramesh, A. Kulkarni, V. Kamat, "Slicing tools for synchronous reactive programs," *Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, July 2004.

[14] A. R. Kulkarni, S. Ramesh, "Static Slicing of Reactive Programs," scam, pages 98, Third IEEE International Workshop on Source Code Analysis and Manipulation, 2003.

[15] "Estbench esterel benchmark suite," http://www1.cs.columbia.edu/~sedwards/software/estbench-1.0.tar.gz.

[16] S. Ramesh, Ramesh's homepage. http://www.cse.iitb.ac.in/~ramesh.