LD 프로그램의 모델 체킹을 위한 자동변환
(Automatic Translations for Model Checking of LD Programs)

권 민 혁 † 신 승 철 ††
(Minhyuk Kwon) (Seungcheol Shin)

요 약 발전소와 임베디드 시스템, 지능형 빌딩과 같은 현대의 대부분의 시스템은 PLC 라는 특수목적 컴퓨터를 이용하여 자동제어된다. PLC 프로그래밍 언어 중에서 가장 많이 사용하는 것은 LD 프로그램이다. LD 프로그램의 검증은 시뮬레이션과 테스팅등을 통해 이루어지는데 이러한 방법은 프로그램 검증에 한계가 있다. 본 논문에는 LD 프로그램의 검증시 모델 체킹을 사용할 수 있도록 주어진 LD 프로그램을 모델 체커의 입력으로 자동으로 변환하는 방법을 기술한다. LD 프로그램과 SMV 모델의 의미구조를 정의하고 이를 바탕으로 의미가 보존되는 변환 함수를 정형적으로 나타낸다.

키워드 : PLC, 모델 체킹, LD 프로그램 모델링, IEC 61131-3

Abstract PLCs are special purpose microcontrollers used in most automatic control systems such as plants, embedded systems, and intelligent buildings. LD is one of the most popular languages among PLC languages. For now LD programs are mainly verified by simulation and testing which has a lot of limitation. This paper describes how to translate a given LD program into an input of a model checker so that LD program is verified by model checking. We define formal semantics of LD programs and SMV models and specify a formal definition of the translation function which preserves semantics between LD programs and SMV models.

Key words : PLC, Model checking, LD program modeling, IEC 61131-3

1. 서 론

산업현장의 특수목적 컴퓨터인 PLC(Programmable Logic Controller)를 위한 프로그래밍 언어들은 IEC 61131-3에서 정의하며 그 중 LD(Ladder Diagram)는 가장 일반적으로 사용하는 언어이다[1]. LD는 RLL (Relay Ladder Logic)과 겉모양은 유사하지만 병렬 제어가 아닌 순차 제어 방식의 그래픽 프로그래밍 언어이다.

LD를 이용하여 프로그래밍하는 것을 세부적으로 따져보면 실제로는 AND, OR, 그리고 NOT 연산을 이용하여 매우 복잡한 논리 표현식을 구성하는 것으로 생각할 수 있다. 언어의 추상화나 타입 시스템과 같은 현대의 프로그래밍 언어 기술이 탑재되어 있지 않기 때문에 프로그램이 매우 복잡해지고 오류 가능성이 높아진다. 또한, C나 JAVA와 같이 일반적인 컴퓨터 프로그래밍 언어에 비해 PLC 언어를 대상으로 하는 테스팅 방법에 대한 연구와 검증 방법에 대한 연구는 많지 않고, 있다 하더라도 사례 연구에 그치는 경우가 많아 PLC 프로그램을 위한 자동 검증 도구의 개발이 필요하다.

휘모리 CDK는 이러한 요구를 충족시키기 위해 개발된 PLC 통합개발환경이다[7]. 여기에는 LD-to-SMV 변환기와 NuSMV모델체커를 연동하는 LD 검증기등이 포함되어 있다.

본 논문에서는 이들 중 LD-to-SMV의 변환방법인 LD 프로그램을 모델체커에서 검증가능한 모델로 자동변환하는 방법을 설명한다. 이를 위해 LD 언어와 모델

· 본 연구는 교육과학기술부/한국연구재단 우수연구센터 육성사업의 지원으로 수행되었음(과제번호 2009-0081599)

† 정 회 원 : 슈어소프트테크 S/W 시험 자동화 연구소 연구원
minhyuk@suresofttech.com
†† 종신회원 : 한국기술교육대학교 인터넷미디어학부 교수
scshin@kut.ac.kr
(Corresponding author임)

논문접수 : 2009년 7월 29일
심사완료 : 2009년 11월 2일

정보과학회논문지: 컴퓨터의 실제 및 레터 제16권 제2호(2010.2)

체커의 입력인 상태전이기계(모델)의 구문구조(syntax)와 실행의미구조(operational semantics)를 정형기법을 이용하여 정의하고 LD 프로그램을 상태전이기계로 변환하는 귀납함수를 정의한다. 또한, 변환의 정확성을 위해 변환된 상태전이기계가 원래의 프로그램의 의미대로 수행됨을 보인다.

이를 위해 본 논문은 다음과 같이 구성한다. 2절에서는 LD 프로그램에 대한 모델 체킹 연구들을 소개한다. 3절과 4절에서는 변환의 소스언어인 LD와 타깃언어인 모델묘사언어(model description language)의 구문구조와 의미구조를 서술하다. 5절은 변환함수를 정의하며, 마지막 6절은 결론이다.

2. 관련 연구

90년대 이후 모델 체킹을 이용한 LD 프로그램의 검증에 대하여 다양한 연구가 이루어졌지만 사례연구가 대부분이며, 대표적으로 Probst의 연구들이 있다[2,3].

조금 체계적인 연구는 Rausch의 연구로, 언어의 모든 생성자와 함수들에 대한 모듈을 미리 만들어 놓고 하나의 메인모듈에서 조건에 따라 호출하는 방법을 제시하였다[4].

가장 실용적인 연구는 Rossi의 연구로 LD의 타이머나 펄스입출력과 같은 PLC의 실용적인 기능들을 포함한 검증을 했다[5]. 그러나 이들 연구에서의 LD 프로그램의 모델링은 모두 수동으로 이루어지며 변환된 LD 프로그램의 의미가 보존되었는지 대한 증명이나 언급은 하지 않고 있다. 본 논문에서는 이 연구들로부터 유용한 것을 취하고 나아가 자동으로 모델링하는 방법을 기술하고, 변환의 정확성 증명 방법을 설명한다.

3. LD

그래픽 프로그래밍 언어인 LD의 구문구조와 의미구조는 IEC 61131-3에서 비정형적으로 정의하고 있다. 정확한 변환과 증명을 위해 이것을 정형적으로 정의할 필요가 있다.

이 절에서 우리는 LD의 구문구조와 의미구조를 표준문서에서 정의한 것을 기반으로 정형적으로 정의한다.

3.1 LD의 추상구문구조

그림 1은 BNF로 표현된 LD 언어를 구성하는 핵심적인 구문구조이다. 여기에서 *id*는 문자열의 조합으로 일반 프로그래밍 언어에서의 변수이름을 만드는 규칙에 의해 조합되며 *memory_id*는 IEC 61131-3에서 언급한 직접주소 지정자를 의미한다.

구문구조의 **jumpto** 문은 LD 프로그램의 흐름을 제어하기 위해 사용한다. **jumpto**는 PLC의 실행 모델의 제한으로 인해 스캔 내에 이루어져야 하기 때문에 **jumpto**에 실인자(actual parameter)로는 **jumpto**가 위치하는 곳으로 부터 다음에 나오는 단(rung)만을 지정할 수 있다고 가정한다.

rungs	::=	*label* : *front* # *rear rungs*
		| *label* : *front* # *rear*
front	::=	*front* & *front* | *front* ‖ *front* | *input*
rear	::=	*rear* ‖ *rear* | *front* & *rear* | *output*
input	::=	contact(*var*) | contactn(*var*)
output	::=	coil(*var*) | coiln(*var*) | jumpto(*label*)
		| jumpton(*label*)
var	::=	*id* | *memory_id*

그림 1 LD의 추상구문

3.2 LD의 실행적 의미구조

LD의 구체적인 실행적 의미구조를 정의함에 앞서 몇 가지 필요한 정의들을 소개한다.

정의 1. [LD기계] *LD기계는 (p, a, inst, E)로 구성된다. 단, p는 LD 프로그램, a는 누산기(accumulator), inst은 단이 실행된 후의 최종 명령어이다. 그리고 E는, LD기계의 메모리로 "참"인 변수들만을 포함한다.*

정의 1은 LD기계를 표현한다. 실제 PLC는 입력채널과 출력채널 내부메모리를 모두 가지고 있지만 *E*는 마치 내부메모리처럼 다룬다. 이것은 원래 PLC의 기계에서 입력채널과 출력채널의 값은 PLC운영체제에 의해 프로그램 실행 전에 내부메모리로 복사되며, PLC 프로그램에서 입출력 정보는 외부모듈이 아닌 이 내부메모리의 정보를 사용하기 때문이다. 또한, *E*에서 "참"인 변수들만 모으는 것은 나중의 증명과정을 위함이다.

LD 프로그램이 실행됨은 다음과 같이 LD기계의 상태(*p, a, inst, E*)가 전이됨을 의미한다.

$$(p,\ a,\ inst,\ E\ ) \rightarrow (\ a',\ inst',\ E'\ )$$

전이관계(→)는 그림 2, 3과 같이 정의했다. 특별히, *inst*은 **jumpto**문의 수행을 위해 고려했다. *inst*은 하나의 단의 *rear*에서 **jumpto**문이 나타나는 경우에만 할당(←)이 이루어지며, **jumpto**가 나오고 현재까지의 누산기의 상태가 tt일 때는 즉각 분기를 한다. 분기할 위치를 프로그램카운터(pc)없이 알아내기 위해 다음과 같이 sub라는 함수를 도입하였다.

$$\text{sub} : rungs \times label \rightarrow rungs$$

이 함수는 *rungs*와 분기하려는 *label*을 입력으로 받아 *rungs* 중 *label*이 위치한 곳부터 프로그램 끝까지의 부분 프로그램을 내어주는 함수이다. 또한, 그림 3에서 *front*와 *rear*의 전이관계 정의를 위해 *front*전이관계(⇢)와 *rear*전이관계(⇢)를 도입하였다. 그림 3에서 real-and, coil 그리고 jumpto 룰(rule)과 각각 듀얼(dual)인 rear-or과 coiln, jumpton는 지면관계상 생략하였다.

$$\frac{(label : front \# rear, a, inst, E) \rightarrow (a', inst', E') \quad (rungs', a', inst, E') \rightarrow (a'', inst'', E'')}{(label : front \# rear\ rungs, a, inst, E) \rightarrow (a'', inst'', E'')}\ [rungs\text{-}jump]$$

where $\begin{cases} ((inst' = \text{jumpto}(label')) \wedge (a'=\text{tt}) \vee ((inst' = \text{jumpton}(label')) \wedge (a'=\text{ff}) \\ rungs' \leftarrow \text{sub}(rungs, label') \end{cases}$

$$\frac{(label : front \# rear, a, inst, E) \rightarrow (a', inst', E') \quad (rungs', a', inst, E') \rightarrow (a'', inst'', E'')}{(label : front \# rear\ rungs, a, inst, E) \rightarrow (a'', inst'', E'')}\ [rungs\text{-}none]$$

where $\neg((inst' = \text{jumpto}(label')) \wedge (a'=\text{tt}) \vee \neg((inst' = \text{jumpton}(label')) \wedge (a'=\text{ff})$

$$\frac{(front, a, inst, E) \rightarrow (a', inst, E) \quad (rear, a', inst, E) \rightarrow (a'', inst', E')}{(label : front \# rear, a, inst, E) \rightarrow (a'', inst', E')}\ [rung]$$

그림 2 Natural Semantics for LD 1/2

$$\frac{(front_1, a, inst, E) \rightarrow (a', inst, E) \quad (front_2, a, inst, E) \rightarrow (a'', inst, E)}{(front_1 \,\&\, front_2, a, inst, E) \rightarrow (a''', inst, E)}\ [front\text{-}and]$$

where $a''' = a' \wedge a''$

$$\frac{}{(\text{contact}(var), a, inst, E) \rightarrow (a', inst, E)}\ [\text{contact}]$$

where $\begin{cases} a' \leftarrow \text{tt} & \text{if } var \in E \\ a' \leftarrow \text{ff} & \text{otherwise} \end{cases}$

$$\frac{}{(\text{contactn}(var), a, inst, E) \rightarrow (a', inst, E)}\ [contact\text{-}not]$$

where $\begin{cases} a' \leftarrow \text{ff} & \text{if } var \in E \\ a' \leftarrow \text{tt} & \text{otherwise} \end{cases}$

$$\frac{(front, a, inst, E) \rightarrow (a', inst, E) \quad (rear, a', inst, E) \rightarrow (a'', inst', E')}{(\text{front \& rear}, a, inst, E) \rightarrow (a''', inst', E')}\ [rear\text{-}and]$$

where $a''' = a' \wedge a''$

$$\frac{(\text{rear}_1, a, inst, E) \rightarrow (a', inst', E') \quad (rear_2, a, inst', E') \rightarrow (a'', inst'', E'')}{(\text{rear}_1 \parallel \text{rear}_2, a, inst, E) \rightarrow (a', inst', E'')}\ [rear\text{-}or\text{-}jump]$$

where $((inst' = \text{jumpto}(label')) \wedge (a'=\text{tt}) \vee ((inst' = \text{jumpton}(label')) \wedge (a'=\text{ff})$

$$\frac{(\text{rear}_1, a, inst, E) \rightarrow (a', inst', E') \quad (rear_2, a, inst', E') \rightarrow (a'', inst'', E'')}{(\text{rear}_1 \parallel \text{rear}_2, a, inst, E) \rightarrow (a'', inst'', E'')}\ [rear\text{-}or\text{-}none]$$

where $\neg((inst' = \text{jumpto}(label')) \wedge (a'=\text{tt}) \vee \neg((inst' = \text{jumpton}(label')) \wedge (a'=\text{ff})$

$$\frac{}{(\text{coil}(var), a, inst, E) \rightarrow (a, inst, E')}\ [coil]$$

where $\begin{cases} E' \leftarrow E \cup \{var\} & \text{if } a = \text{tt} \\ E' \leftarrow E \setminus \{var\} & \text{otherwise} \end{cases}$

$$\frac{}{(\text{jumpto}(label), a, inst, E) \rightarrow (a, inst', E)}\ [jumpto]$$

where $inst' \leftarrow \text{jumpto}(label)$

그림 3 Natural Semantics for LD 2/2

4. NuSMV 모델체커

LD 프로그램을 검증하기 위해서는 LD 프로그램을 상태전이모델로 변환해야 한다. 이 절에서는 우리가 변환하려는 타깃언어인 모델묘사언어의 구문구조와 의미구조를 정의한다.

4.1 모델묘사언어의 추상구문구조

NuSMV의 모델묘사언어는 상태들의 전이관계를 묘사할 수 있도록 해준다. NuSMV의 모델묘사언어의 전체 구문구조는 [6]에서 확인할 수 있으며, 우리는 LD 프로그램의 변환에서 사용되지 않는 범위의 문법을 제외한 간략화된 모델묘사언어를 그림 4와 같이 다시 정의하였다.

nusmv	::=	MODULE *id* VAR *vars* ASSIGN *assigns*
vars	::=	*id* : *type vars* | *id* : *type*
assigns	::=	init(*id*) := *basic_exp assigns* | next(*id*) := *exp assigns* | init(*id*) := *basic_exp* | next(*id*) := *exp*
basic_exp	::=	*aexp* | *bexp*
aexp	::=	*n* | *id* | *aexp* mod *aexp* | *aexp* + *aexp* | *aexp* - *aexp*
bexp	::=	0 | 1 | *b* | *id* | !*bexp* | *bexp* & *bexp* | *bexp* || *bexp* | *bexp* = *bexp* | *aexp* = *aexp*
exp	::=	basic_exp | case cases esac
cases	::=	*bexp* : *basic_exp cases* | *bexp* : *basic_exp*
type	::=	*boolean* | *n..n*

그림 4 간략화된 모델묘사언어의 추상구문

4.2 모델묘사언어의 실행의미

정의 2. [상태전이기계] *상태전이기계 $M=(S,\Rightarrow,L)$은 상태집합 S와 전이관계 $\Rightarrow$ (S에 대한 이항관계) 즉, 모든 $s\in S$는 어떤 $s'\in S$으로의 전이를 가지는 관계와, 레이블링 함수 $L : S \rightarrow P(V)$ 로 구성된다.*

모델묘사언어의 의미는 정의 2의 상태전이기계를 묘사하는 것으로 해석할 수 있다. 이 상태전이기계 $M=(S,\Rightarrow,L)$는 모델묘사언어로 정의된 모델로부터 구할 수 있다.

모델에 나타나는 모든 변수집합을 V라고 할 때, 상태전이기계의 상태집합 S는 P(V)으로 정의한다. 또한 전이관계 $\Rightarrow$는 모델의 모든 next 구문에 대해 그림 5와 같이 정의된 함수를 통하여 구한다. 이 함수의 결과는 전이관계 순서쌍(tuple)의 집합으로, 각 순서쌍은 전이가 이루어지기 전 상태와 전이가 이루어진 후의 상태를 나타낸다. 이 함수에서 사용하는 C는, $S \times S$로 상태집합 S의 카테시안 곱(cartesian product)으로 가능한 모든 전이관계이다. 또한 $s\vDash exp$는 어떤 exp가 상태 s에서 만족하는지, 즉 상태 s가 exp를 참으로 만드는지를 확인하는 기호이다.

따라서 모델묘사언어의 의미는 상태집합 S와 전이관계 $\Rightarrow$를 정의하는 것으로 해석할 수 있다.

5. LD 프로그램의 변환

5.1 변환 함수 정의

변환함수정의에 앞서 LD 프로그램내의 모든 입출력 변수의 집합을 Var^*라고 정의하고 단의 개수를 n이라 가정한다. 모델묘사언어는 그림 6을 기본형태(form)로 하기 때문에 *[vars]*부분은 이미 알고 있는 Var^*집합으로 부터 그림 7의 템플릿(template)을 이용하여 변환전에 구성할 수 있다. 이 템플릿에서 eoc(end of cycle)는 LD 프로그램의 한 스캔이 끝났는지를 식별하기 위한 변수로 마지막 단의 상태전이가 일어날 때 true로 전이한다.

```
MODULE main
VAR
[vars]
ASSIGN
[assigns]
```

그림 6 모델묘사언어의 기본형식

```
[∀variable∈Var*.variable:boolean;]
eoc:boolean;
lc:0..n + 1
```

그림 7 variable 템플릿

또한 lc(location)는 현재 수행중인 단의 위치를 가리키는 변수로 모델체커 내에서 상태전이가 LD기계에서의 단의 실행 순서와 동일하게 순차적으로 이루어지도록 제어하는 역할을 한다.

*[assigns]*은 각 출력에 대한 전이관계를 그림 8에 정의된 변환함수 T를 적용하여 구할 수 있다.

$$
\begin{aligned}
T[\![\,(label{:}front\#rear\ rungs,\ n)\,]\!] &= T_R[\![\,(\mathrm{EXP}(front),rear,n)\,]\!]\\
&\quad\ T[\![\,(rungs,\ n{+}1)\,]\!]\\
T[\![\,(label{:}front\#rear,n)\,]\!] &= T_R[\![\,(\mathrm{EXP}(front),rear,n)\,]\!]\\
T_R[\![\,(exp,rear_1 \parallel rear_2,n)\,]\!] &= T_R[\![\,(exp,rear_1,n)\,]\!]\\
&\quad\ T_R[\![\,(exp,rear_2,n)\,]\!]\\
T_R[\![\,(exp,front\&rear,n)\,]\!] &= T_R[\![\,(exp\&\mathrm{EXP}(front),rear,n)\,]\!]\\
T_R[\![\,(exp,\mathrm{coil}(var),n)\,]\!] &= \mathrm{init}(var) := 0;\\
&\quad\ \mathrm{next}(var) := \mathrm{case}\\
&\quad\ lc = n\ \&\ exp : \mathrm{TRUE};\\
&\quad\ lc = n\ \&\ !(exp) : \mathrm{FALSE};\\
&\quad\ 1 : var;\\
&\quad\ \mathrm{esac};\\
T_R[\![\,(exp,\mathrm{coiln}(var),n)\,]\!] &= \mathrm{init}(var) := 1;\\
&\quad\ \mathrm{next}(var) := \mathrm{case}\\
&\quad\ lc = n\ \&\ exp : \mathrm{FALSE};\\
&\quad\ lc = n\ \&\ !(exp) : \mathrm{TRUE};\\
&\quad\ 1 : var;\\
&\quad\ \mathrm{esac};\\
T_R[\![\,(exp,\mathrm{jumpto}(label),n)\,]\!] &= \epsilon\\
T_R[\![\,(exp,\mathrm{jumpton}(label),n)\,]\!] &= \epsilon
\end{aligned}
$$

그림 8 변환함수 T 정의

$$
\begin{aligned}
N^*[\![\,\mathrm{next}(id) := exp\ assigns\,]\!] &= N[\![\,\mathrm{next}(id) := exp\,]\!] \cap N^*[\![\,assigns\,]\!]\\
N^*[\![\,\epsilon\,]\!] &= C\\
N[\![\,\mathrm{next}(id) := bexp\,]\!] &= \{\ (s,t)\ |\ (s,t) \in C \wedge id \sqsubset t\ \}\\
&\text{where } \sqsubset = \begin{cases} \in & \text{if } s\vDash exp\\ \notin & \text{otherwise}\end{cases}\\
N[\![\,\mathrm{next}(id) := \mathrm{case}\ cases\ \mathrm{esac}\,]\!] &= N_C^*[\![\,(cases,\ id,\ \varnothing)\,]\!]\\
N_C^*[\![\,(bexp_1{:}bexp_2\ cases,\ id,\ \sigma)\,]\!] &= N_C[\![\,(bexp_1{:}bexp_2,\ id,\ \sigma)\,]\!]\ \cup\ N_C^*[\![\,(cases,\ id,\ \sigma\cup bexp_1)\,]\!]\\
N_C^*[\![\,(\epsilon,\ id,\ \sigma)\,]\!] &= \varnothing\\
N_C[\![\,(bexp_1{:}bexp_2,\ id,\ \sigma)\,]\!] &= \{\ (s,t)\ |\ (s,t) \in C \wedge s\vDash bexp_1 \wedge \forall_{exp\in\sigma}.s\nvDash exp \wedge id \sqsubset t\ \}\\
&\text{where } \sqsubset = \begin{cases} \in & \text{if } s\vDash exp\\ \notin & \text{otherwise}\end{cases}
\end{aligned}
$$

그림 5 전이관계(⇒)의 정의

변환함수 T는 각 출력변수에 대한 변환작업을 수행한다. 변환과정 내에 사용된 EXP함수는 LD의 &나 ‖ 와 같은 표현식을 NuSMV의 BOOL표현식으로 변환해주는 함수로, 예를 들어, contact(a) & contact(b) ‖ contact(c)를 a&b‖c 와 같이 변환해 준다.

변환함수 T는 변환하려는 프로그램 p와 자연수 0으로 구성된 순서쌍 $(p,0)$을 입력으로 변환을 수행한다. 0을 초기 값으로 입력하는 이유는 단의 실행순서를 변환과정 내에서 파악하기 위함이다.

우리는 이 변환 과정 내에서 입력에 대한 변환을 다루지 않는데, 이것은 LD 프로그램 자체가 가지고 있는 초기상태의 비결정성을 표현하기 위함이다. lc값은 jumpto문이 나타나지 않는 단에 대해서는 순차적으로 증가시키고 jumpto문이 나타나면 jumpto문에 나타나는 *label*의 단 번호를 알아내어 그림 9의 형태로 lc 값을 변경시킴으로써 제어한다.

```
init(lc) := 0;
next(lc) := case
[ jump 조건들 ]
1:(lc+1) mod n
esac;
```

그림 9 lc값의 제어

예를 들어, 간단한 프로그램 p가 "l1:**contact**(a)||**contact**(b) # **coil**(c)"라면, 처음에 적용되는 변환함수는 T가 된다. 이 함수는 EXP 함수를 통해 front에 해당하는 **contact**(a) || **contact**(b)를 a||b 로 바꾸어 T_R 에 적용시킨다. 이때 rear는 **coil**(c) 이므로 최종적으로, 아래와 같은 모델의 코드 조각이 얻어진다.

```
init(c) := 0;
next(c) := case
 lc = 0 & (a|b) : TRUE;
 lc = 0 & !(a|b) : FALSE;
 1 : c;
esac;
```

이와 같이 모든 단들에 대해 변환함수를 적용하면 최종적인 모든 그림 6의 모든 *[assigns]* 에 대한 모델 코드가 얻어지며, 그림 6이 모두 채워지면 이 텍스트(text)를 NuSMV의 입력으로 사용할 수 있다. 이외의 다른 예제를 이용한 전체 변환 예제는 [8]에서 확인할 수 있다.

5.2 변환의 정확성

앞에서 정의한 변환 함수 T의 변환이 정확함을 보장하기 위해서는 변환전후의 의미가 보존되었는지 증명하여야 한다. 아래는 증명에 필요한 몇 가지 용어들에 대해서 정의한다.

정의 3. [스캔전이] *상태전이기계 $M=(S,\Rightarrow,L)$에서의 임의의 상태 s_0로부터 스캔전이는 다음과 같이 정의한다. 단, n은 프로그램 p에 나타나는 총 단의 개수이다.*

$$s_0 \Downarrow s_n \text{ iff } s_0 \Rightarrow s_1 \Rightarrow \cdots \Rightarrow sn$$

정의 3은 상태전이기계에서의 전이가 실제 LD기계에서의 스캔전이가 되기 위해서는 전이가 n번 이루어져야 함을 의미한다. 상태전이기계의 전이는 각 단에 따라 이루어지기 때문에 증명을 위해서는 LD 기계와 같은 전이가 필요하다.

정의 4. [동등관계] *상태전이기계 $M=(S,\Rightarrow,L)$에서의 상태 s_0와 LD기계의 상태 E가 동등함($\sim$)은 다음과 같이 정의한다. 단, n은 단의 개수이다.*

$$E \sim s_0 \text{ iff } E \sqsubset (s_0 \setminus \{eoc,\ lc_{0..n}\}) \wedge$$
$$E \sqsupset (s_0 \setminus \{eoc,\ lc_{0..n}\})$$

상태전이기계는 LD 기계와는 다르게 eoc와 lc라는 제어변수를 가지고 있다. 두 상태가 동등함은 각 상태에서 이 변수를 제외한 원소들이 모두 같을 때(=) 성립한다.

정리 1. [정확성 정리] *LD 프로그램 p의 변환된 상태전이기계를 $M=(S,\Rightarrow,L)$ 이라고 할 때, LD기계의 초기상태 E와 같은 임의의 $s_0 \in S$에서의 상태전이 $s_0 \Downarrow s_n$의 s_n은 p를 LD기계에서 전이한 후의 상태 E'과 동등하다.*

즉, $E \sim s_0$ 이고, LD기계의 전이와 상태전이기계의 전이가 각각 $((p,\ a,\ inst,\ E) \rightarrow (a',\ inst',\ E'))$ 와 $s_0 \Downarrow s_n$ 일 때,

$$E' \sim s_n$$

이다.

Proof. 증명은 생략.

정리 1은 LD 프로그램 p가 상태전이기계에서 스캔전이 하는 것과 LD 기계에서 실행되는 것의 의미가 동등함을 보장한다. 따라서 변환된 상태전이기계 모델에 대한 모델 체킹은 LD 프로그램을 실행시킬 때 나타나는 모든 경우를 검사하는 것을 가능하게 한다.

6. 결 론

본 논문에서는 LD 프로그램을 기존의 모델체커의 입력으로 변환하는 과정을 설명하고 변환된 상태전이기계와 원래의 LD 프로그램의 의미가 일치함을 증명하는 방법을 보였다.

모델 체킹을 이용하면 모든 가능한 상태에 대한 LD 프로그램의 기능검증을 완전 자동으로 수행 할 수 있다. 이것은 신뢰성 높은 프로그램의 작성에 도움을 주고 LD 프로그램의 버그를 찾아내는 데에도 매우 유용할 것이다. 우리는 이 연구를 통해 구현된 변환기를 이클립스 플러그인 방식의 제어프로그램 개발환경인 휘모리 CDK에 연동하여 추가적인 연구를 수행하고 있다[7].

참 고 문 헌

[1] IEC, "International Standard IEC 61131-3 Programmable controllers-Part3: Programming languages 2nd Edition," International Electrotechincal Commission, 2003.

[2] Probst, S. T., "Chemical Process Safety and Operability Analysis Using Symbolic Model Checking," *PhD Thesis, Department of Chemical Engineering*, Carnege Mellon University, 1996.

[3] Probst, S. T., Powers, G. J., Long, D. E., and Moon, I., "Verification of a Logically Controlled Solid Transport System Using Symbolic Model-Checking," Department of Computer and Chemical Engineering, carnegie Mellon University, 1996.

[4] M. Rausch and B. H. Krogh, "Formal Verification of PLC Programs," In *Proc. American Control Conference*, 1998.

[5] O. Rossi, Ph. Schnoebelen, "Formal Modeling of Timed Function Blocks for the Automatic Verification of Ladder Diagram Programs," *Proc. 4th Int. conf. Automation of Mixed Processes:Hybrid Dynamic systems (ADPM)*, Dortmund, Germany, Sept. 2000.

[6] Roberto C., et al., "NuSMV 2.4 User Manual," ITC-irst, Carnege Mellon University, 2005.

[7] Seungcheol Shin, Minhyuk Kwon, Sanghoon Rho, "Whimori CDK: a Control Program Development Kit," The International Conference of COMPUTING in Engineering, Science and Informatics (ICC2009), 2009.

[8] Minhyuk Kwon, Seungcheol Shin, "A Experiment report for a LD-to-SMV translator," http://pllab.kut.ac.kr/tr/2009/ld2smvexp.pdf, Technical Report, PLLAB, Korea University of Technology and Education, 2009.

권 민 혁

2007년~2009년 한국기술교육대 석사과정. 2009년~현재 슈어소프트테크 S/W 시험 자동화 연구소 연구원. 관심분야는 프로그래밍 언어, 프로그램 분석, 수리 논리

신 승 철

1992년~1996년 인하대 전자계산학과 박사. 1999년~2000년 캔사스 주립대 연구원. 1996년~2006년 동양대 컴퓨터학부 부교수. 2006년~현재 한국기술교육대 인터넷미디어학부 조교수. 관심분야는 프로그래밍 언어, 프로그램 분석 및 검증, 소프트웨어 보안, 수리 논리