PLC를 적용한 실시간 시스템의 가상 프로토타이핑

천성욱, 강순주, 서대화
경북대학교 전자전기공학부 대구광역시 북구 산격동 1370번지, 702-701
chun@palgong.kyungpook.ac.kr

Virtual Prototyping of Programmable Logic Controller based Real-time Systems

Seong Wook Chun, Soon Ju Kang, and Dae Wha Seo
School of Electronic and Electrical Eng., Kyungpook Nat'l Univ.
1370 Sankyuk-dong, Buk-ku, Taegu, 702-701

Abstract To develop an effective virtual prototyping methodology for the PLC(Programmable Logic Controller) based real-time systems, a conversion algorithm from RLL(Relay Ladder Logic) to statechart is presented in this paper. The RLL is the main programming language to represent the operation of the PLC, and the statechart is the most widely used tool in the field of virtual prototyping in order to represent the behaviour of real-time systems. A virtual prototyping for an example case is implemened to evaluate the benefit of the proposed algorithm..

Ⅰ. 서 론

1.1 개요

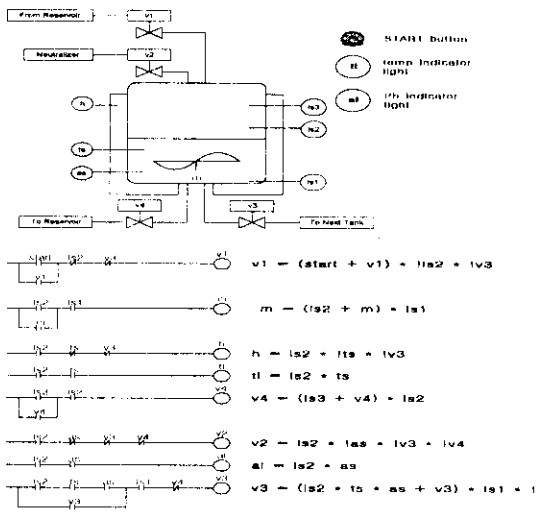
현재 PLC는 각종 생산 혹은 제어 공정 자동화에 널리 이용되고 있다. PLC의 프로그램 언어로 RLL[4],[7]이 주로 이용되고 있지만 디버깅이 쉽지 않고, 제어 수행 순서를 명확하게 나타내지 못하므로 올바르게 동작하는 것을 확인하기 위해서는 PLC의 메모리에 직접 탑재하여 작동해 보아야한다. 이렇게 테스트하는데 불편함이 있고, 고가이면서 고도의 안전성, 무결성이 요구되는 실시간 제어 시스템의 경우 RLL의 잘못에 의해 초기 공정 셋업 시에 큰 손상을 일으킬 수 있다. 그래서 동작 상태를 시각적으로 나타내고, 가상적으로 테스트를 할 수 있도록 하기 위해 PLC 적용 시스템 자체를 VP(Virtual Prototype)으로 구현할 필요성이 있다. VP를 이용한다면 가상적으로 테스트해 볼 수 있고, 발견된 오류는 바로 수정이 가능하므로 PLC를 직접 테스트하는 방법보다 매우 안정적이고, 경제적이다. 그러나 대부분의 PLC 적용 시스템은 복잡한 실시간 특성을 가지므로 VP의 구현이 쉽지 않다. 특히 PLC 프로그램 언어로 사용되는 RLL은 그 표현의 제약성 때문에 VP 구현에 필요한 PLC 적용 시스템의 수행 순서를 효과적으로 표현할 수 없다. 본 논문에서는 이러한 문제를 해결하기 위해 RLL을 VP 제작 도구[1]에서 널리 이용되는 statecharts로 변환하는 알고리즘을 제안하고, 이 알고리즘에 의해 변환된 statecharts를 이용하여 PLC 적용 시스템의 VP 구현이 가능함을 검증하고자 한다.

1.2 응용 예

본 논문에서는 중화 공정 시스템을 예로 RLL과 statecharts에 대해서 설명하고, 변환 알고리즘에 대해서도 설명하고자 한다. 이 시스템의 동작 순서는 다음과 같다.

(1) 모든 밸브들은 닫힌 상태, mixer인 'm'과 heater인 'h'는 OFF, 탱크 안은 비어있는 상태로 둔다.
(2) 'start' 버튼이 눌러지면 'ls2'가 ON될 때까지 'v1'을 열어 용액을 레벨 'ls2'까지 채운다.
(3) 다음 세 공정은 동시에 일어난다. : ①용액이 'ls2'를 넘으면 'm'을 가동시키고, 'ls1'보다 아래에 있을 경우 멈춘다. ②용액의 온도가 설정 치보다 낮을 때 'h'를 가동시킨다. ③용액의 Ph가 설정 치와 맞지 않은 경우 'v2'를 열어 중화제를 투입한다. ①탱크가 차면 'ls3이 ON된다. 이때 'v2'를 닫고, 'v4'를 연다. 용액의 양이 'ls2'까지 줄어들면 'v4'를 닫고, '③'과정을 다시 수행한다.
(4) 온도, Ph 모두 설정 치와 일치하면, 'h'를 끄고, 'v2'를 닫고, 'v3'을 열어 용액을 다음 탱크로 보낸다. 탱크가 비어 있게 되면 'ls1'이 ON되고, 이때 'v3'을 닫고, 공정 (1)을 다시 수행한다.

수행 순서에서 (3)의 '①'는 '③'의 내부에서 수행되는 공정이다. 이 시스템의 구성과 RLL은 [그림 1]과 같다.

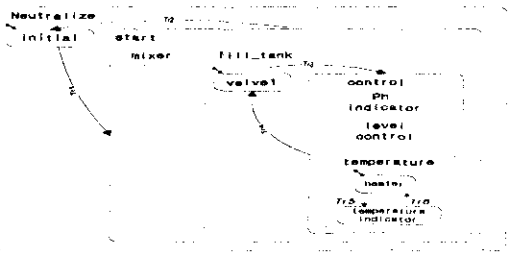


[그림 1] 중화 공정 시스템의 구성과 RLL

RLL에 나타나는 수평선을 렁(rung)이라 하는데 이는 불 대수식(boolean expression)을 그래픽 적으로 나타낸 것으로 입력, 출력 소자들은 접점과 코일로만 표현된다[6]. [그림 1]에서 보면 'AND'는 '*', 'OR'은 '+', 'NOT'은 '!'으로 나타내어짐을 알 수 있다. RLL은 상태 변수(state variable)들의 집합인 'Q'와 제어 변수(control variable)들의 집합인 'U'로 정해진다. 이 시스템에서 Q = {v1, m, h, tl, v4, v2, al, v3} 이고, U =

{start, ls1, ls2, ls3, ts, as}이다. 상태 변수 'v1'은 저장소에서 탱크로 용액을 보내는 밸브, 'm'은 mixer, 'h'는 heater, 'tl'은 온도를 나타내는 램프, 'v4'는 용액을 탱크에서 다시 저장소로 보내는 밸브, 'v2'는 중화제를 조절하는 밸브, 'al'은 Ph를 나타내는 램프, 'v3'은 용액을 다음 탱크로 보내는 밸브이다. 제어 변수 'start'는 중화 공정 시스템을 시작시키는 버튼, 'ls1', 'ls2', 'ls3'은 탱크 내의 용액의 레벨을 측정하는 센서들이고, 'ts', 'as'는 각각 용액의 온도, Ph를 측정하는 센서이다. 렁의 출력은 제어 변수와 이전에 스캐닝 된 상태 변수에 의존한다. [그림 1]에서 보는 것처럼 RLL에서는 제어 수행 순서와 렁의 출력에 나타나는 상태 변수가 어떤 상태 변수와 제어 변수에 의존하는가에 관한 정보를 명확하게 나타내지 못함을 알 수 있다.

[그림 2]는 중화 공정 시스템을 statecharts로 직접 나타낸 것이다.



[그림 2] 중화 공정 시스템의 statecharts

statecharts는 STD(State Transition Diagram)에 'hierarchy', 'concurrency', 'broadcast communication' 개념을 추가시킨 것이다. STD로는 복잡한 실시간 시스템을 설계하는데 한계가 있어 위의 세 개념을 적용하여 실시간 시스템의 특성들을 보다 효과적으로 표현할 수 있도록 하였다[2],[3],[5]. statecharts에서는 천이(transition)를 "event[condition]/action"의 형태로 나타낸다. 'event'로 천이를 트리거 시키고, event에서 'condition'이 참일 때에만 트리거가 일어난다. 'action'은 트리거가 일어나면서 수행하는 것이다.

statecharts는 시스템을 여러 상태(state)들의 조합으로 나타낸다. 상태는 배타 상태(exclusive state)와 병행 상태(concurrent state)로 나뉘어진다. 배타 상태는 부모 상태(parent state)가 활성(active) 되었을 때 자식 상태(children state)들 중 하나만 활성이 된다. 병행 상태는 부모 상태가 활성이 되었을 때 모든 자식 상태가 동시에 활성이 된다. 시스템이 서로 의존하지 않고 동시에 동작되는 부분을 병행 상태로 나타낸다. [그림 2]의 statecharts에서 보면 'valve 1'의 open 공정이 heater, mixer, 중화제 첨가와 같은 공정들보다 상위에 위치하므로 먼저 수행됨을 알 수 있고, heater, mixer, 중화제 첨가 공정들은 병행 상태에 위치하여 동시에 수행됨을 명확하게 알 수 있다. 그러나 이러한 공정의 수행 순서와 동시성을 [그림 1]의 RLL에서는 효과적으로 나타내지 못함을 알 수 있다.

Ⅱ. 본 론

현재 RLL의 대안으로 다른 언어들에 관한 연구[6]가 진행 중이며 그 중 논리적인 표현, 제어 수행 순서, 로직간의 의존성을 비교적 명백히 나타낼 수 있는 SFC(Sequential Function Charts)로 변환하는 알고리즘은 이미 구현되었다[4]. SFC가 RLL의 단점을 보완하였지만 PLC 제어 시스템을 시각적으로 나타내는데 제약이 있으므로 이 보다는 RLL을 statecharts로 변환한 후 이를 이용하여 VP을 만들어 테스트하고, 오류들을 수정할 수 있도록 한다면 시스템을 적은 비용으로 보다 빨리 구현할 수 있을 것이다. 그러므로 RLL을 SFC로 변환하는 알고리즘[4]을 참고하여 RLL을 statecharts로 변환하는 알고리즘을 제안한다.

2.1 RLL→statecharts 변환 알고리즘에 필요한 요소

1) SG(Simultaneity Graph) : SG는 "동시에 어떤 상태 변수들이 ON 되는가?"를 나타내기 위해 동시에 ON이 되는 노드들을 직선으로 연결한 그래프이다.

2) DG(Dependency Graph) : DG는 "RLL의 한 스캔 동안 어떤 렁의 출력이 이전에 ON 되었던 렁들에 의존하는가?"를 나타내는 그래프이다. 의존 관계를 알아야만 렁의 스캔 순서를 정할 수 있다.

3) CSG(Condensed Simultaneity Graph) : CSG는 DG에서 의존하는 노드들은 하나의 노드로 나타내고, SG의 노드 연결 상태를 그대로 나타낸 그래프이다.

RLL→statecharts 변환 알고리즘에서는 CSG를 분해할 필요성이 있으며 그 방법으로는 CCD(The Connected Component Decomposition)과 FCD(The Fully Connectivity Decomposition) 두 가지가 있다.

CCD : 어떤 그래프 G의 연결된 부분들을 분해하는 것으로 CCD(G)={G1,G2,G3,.....}이며 Gi 내의 어떤 노드도 Gj 내의 어떤 노드와 연결되어 있지 않다[i≠j]. CCD 이후 생성된 서브 그래프들은 statecharts에서 동일 레벨의 배타 상태에 놓이므로 수행 순서를 결정해 주어야 한다. 수행 순서에 관한 정보를 나타내기 위해 'sequence graph'를 이용하며 여기에서 St[i]를 활성 시키는 천이를 'A St[i]'로 비 활성화(deactivate) 시키는 것은 'D St[i]'로 나타낸다. "A St[i] := St[i]와 관련된 렁의 조건(condition)들을 .OR. 취한 것"으로 나타내고, "D St[i] := St[i]와 연관된 상태 변수를 각각 .NOT.을 취한 후 .AND. 취한 것"으로 나타낸다.

FCD : 어떤 연결된 그래프 G를 분해하는 것으로써 FCD(G)={G1,G2,G3,.....}이며 이때 Gi 내의 모든 노드들은 Gj의 모든 노드들과 서로 연결되어져 있다[i≠j]. 생성된 서브 그래프의 노드를 연결하는 상호 연결 에지(edge)들은 제거하고 서브 그래프는 동일 레벨의 병행 상태에 둔다.

2.2 RLL→statecharts 변환 알고리즘

[단계 1] 초기 statecharts 생성 : 최상위 상태에 제어 시스템의 이름을 사용하고, 자식으로 배타 상태인 St[initial state]와 St[second state]를 둔다.

[단계 2] CSG의 생성 : RLL로부터 SG와 DG를 구하고 이들을 이용하여 CSG를 생성한다.

[단계 3] 배타 상태의 생성 : CSG에서 CCD가 가능하면 서브 그래프들을 동일 레벨의 배타 상태에 두고, sequence graph를 이용하여 수행 순서를 결정한다. 수행 순서가 결정된 후에 아래의 규칙에 따라 천이를 결정한다. ①천이가 sequence graph의 첫 상태(first state)의 앞에서 일어난다면 이는"모든 A St[i]들을 OR 취한 것"이다. ②천이가 단말 상태(terminal state)인 St[i]보다 나중에 일어나거나 St[i]는 St[j]와 연결한 유일한 경로에서 일어나는 것이면 이는 "D St[i]"가 된다. ③천이가 St[i]와 St[j] 사이에 있고, 이것이 St[i]에서의 유일한 경로가 아니면 천이는 "D St[j] .AND. (.OR. of A St[k])"이다. St[k]는 이 천이에 의해 도달 가능한 상태이다. 천이가 결정된 후 더미 상태(dummy state)를 추가한다.

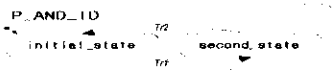
[단계 4] 병행 상태의 생성 : CSG에 FCD가 가능하다면 서브 그래프를 동일 레벨의 병행 상태에 둠으로써 동시에 활성이 되도록 한다.

[단계 5] 반복 : CSG가 더 분해되지 않을 때까지 [단계 3],[단계 4]를 반복하여 최종 statecharts를 얻는다.

2.3 RLL→statecharts 변환 알고리즘의 적용

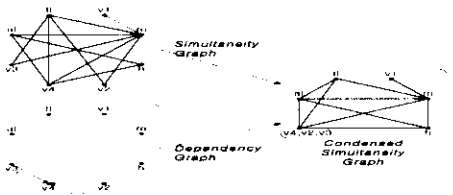
(1) 초기 statecharts의 생성 : St[initial_state]는 시스템이 시작될 때 처음 활성이 되는 디폴트 상태이고, St[second_state]는 모든 상태 변수와 관련이 있는 상태로써 임시로 이렇게 정하였다. 이 상태에 다음 두 천이를 추가하여 statecharts를 구성한다.

Tr1 := A St[second_state] = start+ls2
Tr2 := D St[second_state]
= !v1*!m*!h*!tl*!v4*!v2*!al*!v3



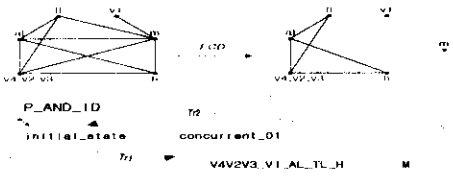
[그림 3] 초기 statecharts

(2) CSG의 생성 : [그림 1]의 RLL에서 SG와 DG를 생성하고, 이를 이용하여 CSG를 생성한다. DG의 의존하는 세 변수들 v4, v2, v3을 하나로 통합하여 새 노드 'v4,v2,v3'으로 바꾼다. 노드들 간의 연결은 SG의 것을 그대로 유지한다. 'v4,v2,v3'에서 알 수 있듯이 세 상태 변수들은 동시에 스캔되어 출력에 나타내어진다.



[그림 4] SG, DG and CSG

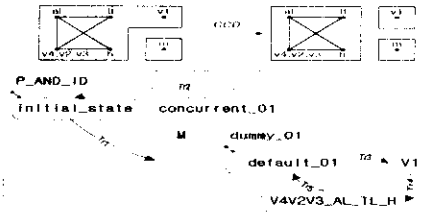
(3) 노드 'm'에 FCD를 적용 : CSG의 노드 'm'은 다른 모든 노드에 연결되어 있으므로 FCD가 가능하다. 생성된 서브 그래프 {m}, {(v4, v2, v3), v1, h, tl, al}은 동일 레벨의 병행 상태에 놓이므로 초기 statecharts에서의 St[second_state]를 St[concurrent_01]로 바꾸고, 이 아래 두 서브 그래프를 자식 상태로 둔다. statecharts는[그림 5]와 같다. 여기에서 mixer의 동작은 전적으로 탱크 내의 레벨(ls1, ls2, ls3)에만 의존하지 다른 어떤 상태 변수들과는 무관하게 동작한다는 것을 알 수 있다.



[그림 5] 'm'의 FCD 이후 생성된 statecharts

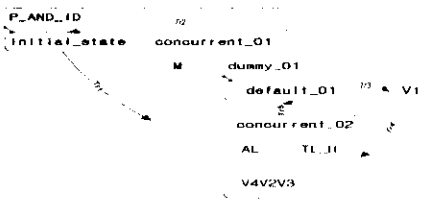
(4) 노드 'v1'에 CCD 적용 : 서브 그래프 {(v4, v2, v3), v1, al, tl, h}에서 노드 'v1'은 다른 어떤 노드와도 연결되어있지 않으므로 CCD가 가능하다. 생성된 두 서브 그래프 {v1}, {(v4, v2, v3), al, tl, h}는 동일 레벨의 배타 상태에 놓인다. St[V4V2V3_V1_AL_TL_H]를 St[dummy_01]로 바꾸고 자식 상태에 생성된 두 서브 그래프를 둔다. St[dummy_01]의 디폴트 상태로 St[default_01]을 두었다. 자식 상태의 수행 순서는 용액을 중화시키는 공정보다 'v1' 밸브를 열어 용액을 채우는 공정이 먼저 수행되어야하므로 St[V1] →St[V4V2V3_V1_AL_TL_H]가 성립된다. 수행 순서가 결정되었으므로 다음 세 천이를 추가하여 statecharts를 생성한다.

Tr3 := A St[V1] + A St[V4V2V3_V1_AL_TL_H]
= start+ls2
Tr4 := D St[V1] = !v1
Tr5 := D St[V4V2V3_V1_AL_TL_H]
= !v4*!v2*!v3*!v1*!al*!tl*!h



[그림 6] 'v1'의 CCD 이후 생성된 statecharts

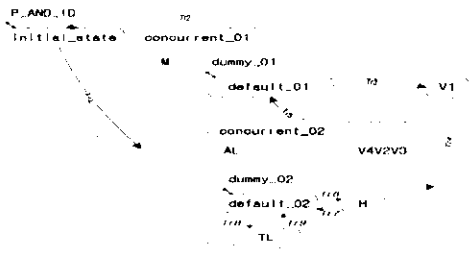
(5) 노드 'al', 'v4,v2,v3'에 FCD 적용 : 노드 'al'과 '(v4, v2, v3)'에서 FCD가 가능하므로 생성된 서브 그래프 {al}, {(v4,v2,v3)}, {tl, h}를 동일 레벨의 병행 상태에 둔다. 그리고 St[V4V2V3_AL_TL_H]를 St[concurrent_02]로 바꾸고 이 아래에 서브 그래프를 자식 상태로 두어 용액의 가열, Ph 조절, 레벨 조절을 동시에 수행할 수 있도록 한다.



[그림 7] 'al', 'v4,v2,v3'의 FCD 이후 생성된 statecharts

(6) 노드 'tl', 'h'에 CCD 적용 : CCD이후 생성된 두 서브 그래프 {tl}, {h}를 동일 레벨의 배타 상태에 둔다. St[TL_H]를 St[dummy_02]로 바꾸고 자식 상태로 St[H], St[TL]과 디폴트 상태인 St[default_02]를 둔다. 시스템이 시작될 때 heater가 동작하든, 온도 indicator lamp가 켜지든 두 공정의 순서가 시스템의 전체 동작에는 아무 영향을 끼치지 않으므로 St[H], St[TL]의 수행 순서를 결정하는 것은 무의미하다. 다음의 네 천이를 추가하여 최종 statecharts를 생성한다.

Tr6 := A St[H] = ls2*!ts*!v3
Tr7 := D St[H] = !h
Tr8 := A St[TL] = ls2*ts
Tr9 := D St[TL] = !tl



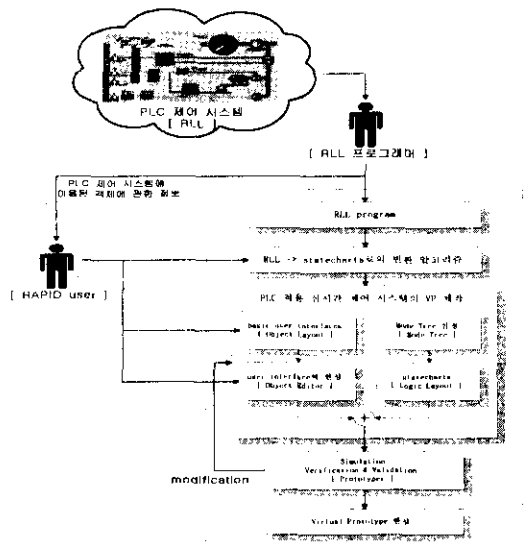
[그림 8] 생성된 최종 statecharts

최종 statecharts [그림 8]을 직접 작성한 statecharts [그림 2]와 비교해 보면 일단 각 상태의 이름이 다름을 알 수 있다. 그리고 [그림 2]에서는 병행 상태의 아래

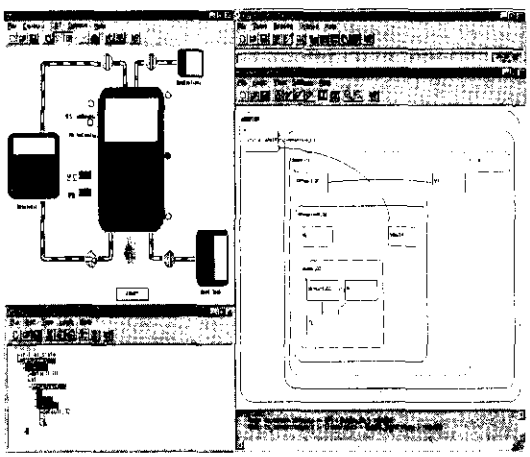
에 있는 자식 상태에 별도의 디폴트 상태를 정해주지 않았지만 알고리즘에 의해 변환된 [그림 8]의 최종 statecharts에서는 디폴트 상태가 존재한다. 알고리즘의 전개 과정에 의해 별도의 디폴트 상태가 필요하다. 이렇게 디폴트 상태가 추가되었기 때문에 3개의 transition이 더 추가되어 총 9개의 transition이 statecharts에 필요하게 된다. 디폴트 상태는 중화 공정의 수행과는 무관하게 단지 부모 상태가 활성이 될 때 자동적으로 활성이 되도록 미리 정해 놓은 것에 불과하다. 디폴트 상태와 이와 관련된 3개의 transition이 더 추가되었지만 이들이 중화 공정의 수행 순서에 영향을 미치는 부분은 아니므로 직접 작성한 statecharts와 알고리즘에 의해 변환된 statecharts는 의미론적으로 일치한다.

2.4 구현 및 검증

제안된 알고리즘에 의해 생성된 최종 statecharts를 VP 제작 도구인 RAPID[3]에 적용하여 중화 공정 시스템의 VP을 구현하였다. 구현 과정은 [그림 9], 구현 결과는 [그림 10]에 나타내었다.



[그림 9] PLC 제어 시스템의 VP 구현 과정



[그림 10] 중화 공정 시스템의 VP 결과 화면

①RLL 프로그래밍 : [그림 1]의 RLL을 그대로 이용
②RLL→statecharts 변환 알고리즘 적용
③사용자 인터페이스 완성 : 시스템에 사용된 객체들을 편집하여 사용자 인터페이스를 나타낸다.
④시뮬레이션 : 생성된 statecharts에 객체와 관련된 속성(property), 기능(function)들을 천이 부분에 추가하여 로직을 결정하고, 구현된 VP을 시뮬레이션 해보고, 오류가 발견되면 이들을 수정하고, 오류가 없을 때까지 이 작업을 반복한다.
⑤VP의 완성 : 시뮬레이션과 수정의 과정을 반복하여 중화 공정 시스템의 최종 VP를 완성하였다.

구현된 VP을 시뮬레이션 해보면 수행이 중화 공정 순서와 동일하게 동작함을 확인 할 수 있다. 임으로 설정 온도는 50 ℃, 설정 Ph는 7로 정하였다. 천이 부분에 사용된 객체들의 속성과 기능을 추가한 것 외에는 제안한 알고리즘에 의해 생성된 statecharts를 그대로 이용하였으므로 알고리즘에 대한 검증이 되었다고 볼 수 있다.

Ⅲ. 결 론

본 논문에서 PLC 프로그램에 널리 사용되는 RLL을 실시간 시스템의 모델링에 이용되는 statecharts로 변환하는 알고리즘을 제안하였다. 그리고 생성된 statecharts의 천이 부분에 객체의 속성, 기능을 추가하여 중화 공정 시스템의 VP을 구현하였다. 그 결과 RLL에서 효과적으로 나타낼 수 없었던 제어 수행 순서와 로직 사이의 의존성을 statecharts에서는 명확하게 나타낼 수 있었을 뿐만 아니라 statecharts를 이용하여 VP을 구현하여 이를 가상적으로 시뮬레이션 할 수 있음을 확인할 수 있었다. 따라서 제안한 알고리즘을 적용하면 매우 복잡한 실시간 특성을 갖는 PLC 적용 시스템에 대한 VP의 구현도 가능함을 입증할 수 있었다.

참고 문헌

[1] Emultek Ltd., Rapid User Manual, Emultek Ltd., Press, 1996.

[2] A. Bar-Tur, D. Drusinsky and D. Harel, "Using Statecharts for Describing the Communication between Complex Systems", Department of Applied Mathmatics, The Weizmann Institute Science, Rehovot, Israel, 1986.

[3] Moshe S. Cohen, "Use of Statecharts in a HW/SW Co-Design Environment", Design SuperCon '95, Feb. 1995.

[4] A. Falcione and B. H. Krough, "Design Recovery for Relay Ladder Logic", IEEE Control Systems Magazine, Vol. 13, Iss. 2, 1993, pp. 90-98

[5] D. Harel, "Statecharts: A Visual Formalism for Complex Systems", Science of Computer Programming, Vol. 8, 1987, pp. 231-274

[6] M.A. Jafari, T.O. Boucher and G.A. Meredith, "A transformation from a boolean equation control specification to Petri Net," IE Working Pap. 90-106. Dept. of IE. Rutgers Univ.

[7] C. D. Simpson, "Programmable Logic Controllers", Prentice Hall, 1994.