정규표현식을 이용한 통신 프로토콜의 최소 시험 경로 생성

Test Case Generation of Communication Protocol with Regular Expressions

김 한 경*
Han-Kyung Kim

요 약

프로토콜을 시험하기 위하여 페트리 네트나 동적인 FSM을 이용하여 시험열을 생성하는 방법이 제시되고 있지만, 이 방법은 프로토콜 오류를 허용하거나 루핑 경로가 포함되어 있는 경우에는 에러가 발생하거나 상태가 폭발하여 시험열 생성이 불가능하다. 또 프로토콜을 구현하고 시험하기 위한 시험 범위를 결정하는 것은 시간적, 기술적 및 경제적으로 어려운 문제이다. 이를 위하여 정규 표현식을 이용하여 정적으로 간단하게 프로토콜 기능을 커버하는 시험열 생성 방안을 제시하였다. 제안한 방법에 의하여 Q.2971 프로토콜의 최소 시험열을 생성한 결과 38가지의 시험열을 구하였으며, 동적인 방법을 사용할 때 루프 상태의 형성에 기인하는 반복 시험 횟수 문제는 표현식을 단순화하는 과정에서 최소화 시킬 수 있었다. 이 과정에서, 시험열의 생성은 정규 표현식을 사용하는 것이 간단하고 쉽다는 것을 확인하였다. 또한 구해진 정규 표현식에서 임의의 시험열의 포함 관계를 검출하기 위한 방법도 검토하였다.

Abstract

Though it is proposed to use Petri net or dynamic FSM methods for the generation of test sequences on some specific protocols, those methods are unavailable on the cases where the protocol allows faults processing or includes paths in looping which cause errors or endless looping by the explosion of states. The determination of test coverage on the protocol software that has been designed and implemented is difficult by the reason of development periods, technical solutions to support and also economical limitations. It is suggested to generate timely protocol software test sequences on the basis of regular expressions covering the functions of protocol. With this regular expression method, the 38 test sequences of Q.2971 protocol has been generated and also minimized the endless looping problem when dynamic test suites are used by simplifying the test path expressions that denotes loops. According to the works, the suggested method is confirmed as simple and easy compare to the other dynamic test sequence generation techniques. Moreover, the method to search an optional test path whether it is included or not in the regular path expression is reviewed.

1. 서 론

프로토콜을 설계하는 것은 매우 복잡하고 또 그것을 구현하는 것은 다양하게 이루어지기 때문에 구현된 프로토콜을 원래의 규격과 일치하는지 검증하는 기술의 중요성이 강조되어 왔다. 이를 위해 여러 가지 프로토콜의 적합성 여부를 판단하는 시험열(test sequence) 생성 방법이 제시되었다[1]. 시험열을 만들기 위한 방법론으로는 T, U, D, W-method 등이 있으며, 이들은 주로 프로토콜의 메시지 흐름에 대한 시험을 목적으로 하고 있다. 즉, 프로토콜의 적합성 여부를 시험하는 것은 기본적으로 IUT(Implementation Under Test)에 입력을 가한 다음 그것에 대한 출력을 관찰함으로써 프로토콜 장치의 상태 변화를 유추하는 것이다[2,3,4]. 이러한 방법론의 기본적인 생각은 시험 대상물에게 하나의 입력을 부여한 다음 그 대상물로부터의 출력 내용을 기대한 내용과 비교해보는 방법이다.

프로토콜은 문법과 형식 및 프로시져로 구성되는데 문법과 형식은 위에서 제시된 방법론에 의해 동적으로 적법성 여부 시험이 가능하지만 프로시져는 초기 시스템 시험을 위해 제어 차원에

* 정회원 : 창원대학교 컴퓨터공학과 부교수
hkim@sarim.changwon.ac.kr

서의 시험이 불가피하다. 화이트박스 시험(whitebox test)의 경우, 구현된 프로토콜 소프트웨어를 시험해야 하는 개발자의 입장에서는, 프로토콜의 운행 가능한 범위(coverage)에 대하여 소프트웨어적인 시험을 수행해야 하므로, 소프트웨어 시험을 위한 시험 경로(test path)가 먼저 분류되어야 한다[5,6]. 이와 같은 것은 메가 또는 테라 소프트웨어라는 거대한 통신용 프로그램에 대한 시험을 수행해야 하는 경우에는 문제 영역의 분리와 함께 test suite 개발 노력을 절감하기 위해서 연구가 진행되고 있다[7].

최근의 프로토콜에 대한 시험 영역에는 선행검증 기법을 통해 프로토콜의 표준성 오류로 인한 기능의 장애 또는 저하가 발생하는 문제 해결을 시도하고 있다[8,9]. 그러나 메시지의 반복적인 입력을 허용함으로써 시험 경로 중에 루프가 발생하는 문제를 해결하는 문제는 여전히 남아있다. 루프가 발생하는 경로에 대하여 페트리 네트나 동적 FSM(finite State Machine) 모델을 만들어 도달성 검사를 하게 되면 시험 경로 생성이 무한 루프에 빠지고, 상태가 폭발하기 때문에 자동화된 도구의 적용은 대체로 불가능하다[10]. 이 문제를 해결하기 위하여 정규적인 표현을 이용하여 실용적으로 시험열을 구하는 알고리즘을 제시하고, 네트워크 측에서 수행되는 Q.2971 프로토콜[11,12]의 시험 경로를 생성하였다. 이는 페트리 네트나 다른 동적인 시험열 생성 규칙을 이용하는 것보다 간편하고 정적이며, 시험이 필수적으로 이루어져야 하는 최소한의 시험 경로를 생성하였음을 의미한다.

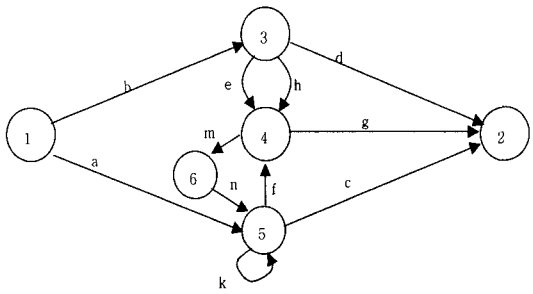
2. 시험경로의 정규적 표현

프로토콜 상태 천이도는 링크와 노드로 구성되는데, 두 노드 사이에 관계성이 존재하면 링크로 표현한다. 두 노드 사이의 관계성을 나타내는 링크는 메시지로 추상화 되며 화살표로 표현한다. 모든 링크를 조합하면 상태천이도 상에서 메시지를 주고 받는 일련의 경로(path)를 나타낼 수 있다. 동일 링크 상에 여러 메시지의 흐름이 존재할 수 있기 때문에 도달 가능한 링크열의 범위(coverage) 뿐만 아니라 해당 링크의 메시지까지 명세화 할 수 있어야 한다. 링크에 주어질 수 있는 가장 단순한 링크의 이름을 사용하면 대수학적 표현에 의해 상태 천이도의 초기 상태에서 종료 상태까지의 유한하거나 또는 무한한 모든 일련의 경로를 표시할 수 있다. 이 기본적인 알고리즘은 비정상적인 메시지의 검출과 타이밍 분석 그리고 다양한 디버깅, 프로토콜 시험에도 사용할 수 있다.

상태도에서의 모든 링크에는 메시지의 이름에 의해 결정되는 링크의 이름이 주어진다. 링크의 이름은 소문자에 의해 표기하며 하나의 링크에 여러 메시지가 존재하는 경우 링크 이름에 아래 첨자로 구분하여 사용한다. 상태의 천이는 이 링크의 이름으로 자연스럽게 표현이 가능하다. 상태의 증가는 경로의 복잡성을 증대 시키지만 상태 노드를 제거하면서 경로를 생성하면 시험을 수행해야 할 경로를 추출할 수 있다[2].

2.1 시험 경로의 합과 곱

만일 그림 1에서 임의의 두 노드 1과 2 사이에서 상태의 천이가 상태 1, 5, 4, 2를 경유하여 진행된다면 a, f, g로 이름이 부여된 링크를 경유하여 순차적으로 발생하게 되고, 이것의 중간 천이 상태를 제거하여 경로 링크를 afg로 나타낸다. 이러한 경로의 표현 방법을 경로의 곱(path product)이라고 한다. 또한 동일한 두 노드 사이에 또 다른 상태 천이 링크가 존재하는데 상태 1, 3, 2를 경유하는 경로를 bd라고 표현 한다면 두 노드 사이에 afg와 bd 경로가 존재하게 된다. 임의의 두 노드 사이에 구성되는 일련의 링크 경로(path)에 대한 집합을 X 또는 Y와 같은 대문자로 나타내는데, 경로의 집합은 구성되는 경로들의 합으로 즉, ac + afg + bd + . . . 와 같이 표기될 수 있다. 여기에서 "+" 기호는 "or"의 의미를 가지며, 두 노드 사이에 구성되는 세부 경로들을 나타낸다. 즉, 두 노드 사이에 ac, afg, bd 그리고 기타



(그림 1) Path의 예

다른 경로가 존재할 수 있음을 뜻한다. 이러한 표현을 경로의 합(path sum)이라고 하며 두 노드 사이의 병렬성을 표시한다. 그림 1에서 상태 3과 상태 4 사이에 존재하는 링크 e와 h는 병렬 링크이며 e + h로 표시된다. 마찬가지로 그림 1에서 상태 1과 2 사이의 모든 경로는 ac + afg + bd + beg + bhg로 표시될 수 있다. 이와 같이 경로의 곱과 합을 이용하여 표현한 것을 경로 표현식(path expression)이라고 한다.

두개의 연속되는 경로 세그먼트는 하나의 경로 곱으로 표현이 가능하다. 예를 들어서 만일 A = abc, X = xyz으로 정의된다면, 다음과 같이 경로를 표현할 수 있다.

AX = abcxyz

XA = xyzabc

aX = axyz

Xa = xyza

XaX = xyzaxyz

만일 B와 Y가 일련의 경로 또는 경로식으로 나타난다면, B와 Y의 곱 역시 B와 Y의 원소들이 상호 연결 가능한 경로의 합으로 표현 가능하다. 예를 들어서, B = b, Y = e + h 라고 하면, BY = be + bh와 같이 나타낼 수 있다. 경로의 곱은 교환적이 아니다. 이것은 BY가 항상 YB와 동일하지 않다는 것을 의미한다. 그러나 연산식에 의해 도출된 결과는 연산식과 동일하다.

2.2 상태의 반복 표현

하나의 링크나 경로 세그먼트가 반복되면 지수 형태로 표현할 수 있다. 지수 값은 상태의 반복되는 횟수를 표현한다.

$k^1 = k$, $k^2 = kk$, $k^3 = kkk$,

$k^n = kkk \ldots$ (n 번 반복)

마찬가지로 X = fmn으로 구성되어 있다면, 위에서의 방법대로 다음과 같이 나타낼 수 있다.

$X^1 = fmn$

$X^2 = fmnfmn = (fmn)^2$

$X^3 = fmnfmnfmn$

$= (fmn)^2fmn = fmn(fmn)^2 = (fmn)^3$

지수가 0인 경로는 그 결과 값이 숫자 1로서 표시되는데 이것은 어떠한 링크도 갖지 않음을 의미한다. 1은 곱셈의 단위 원소로서 산술식에서의 숫자 1과 동일한 속성을 가진다.

$a^0 = 1$

$X^0 = 1$

2개의 상태 노드 사이에서 동일한 상태가 반복적으로 천이 되는 경우에는 병렬 경로가 천이되는 횟수 만큼 발생하는 것으로 해석할 수 있다. 이것은 경로가 링크 b 하나로 구성되었을 경우에는 반복 횟수로 나타나는 모든 경우를 더한 것이 된다. 즉, $b^0 + b^1 + b^2 + b^3 + b^4 + b^5 + \ldots$으로 나타난다. 여기에서 반복되는 단위가 경로 표현식이면 X^*, 단일 링크라고 하면 b^*로 표시한다. 만일 적어도 한번 이상 반복하게 되는 경우에는 X+ 또는 a+로 표시한다. 식에서 왼쪽 표현식의 의미는 오른쪽과 동일하다.

$ab^*c = ac + abc + abbc + abbbc + \ldots$

$a(bc)^*bd$

$= abd + abcbd + abcbcbd + a(bc)^3bd + \ldots$

또한 $aa^* = a^*a = a+$이고, $XX^* = X^*X = X+$이다.

3. Q.2971의 최소 시험경로의 추출

3.1 시험경로 추출 알고리즘

1. 연속되는 링크들을 곱의 형태로 나타낸다. 병렬로 수행되는 링크는 합의 형태로 경로 표현식을 나타낸다.
2. 운행 경로를 나타내는 표현식에 대수적인 규칙을 적용하여 정리한다.
3. 자기 자신의 상태로 천이 되는 링크를 X* 형식으로 바꾸어 루프를 제거한다.
4. 상태 노드를 가능하면 제거하고, 그 상태 노드에 의해 연결된 입출력 링크를 결합하여 새로운 링크를 나타낸다.
5. 연속되는 링크가 남아 있으면 이들을 경로의 곱으로 나타낸다.
6. 병렬 링크가 존재하면 이들을 경로의 합으로 나타낸다.
7. 이 단계에서 경로 표현식이 자신의 노드로 천이하는 경우가 있으면 단계 3에서처럼 루프를 제거한다.
8. 구해진 경로 표현식이 그래프에서 초기 상태에서 종료 상태에 이르기 까지 단일 링크로 나타나도록 단계 4로 돌아가서 반복 수행한다.

(그림 2) 최소 시험 경로 표현식을 구하는 알고리즘

Q.2971 프로토콜 상태 천이도에서 초기의 Null 상태에서 종료 상태인 N11 및 N12 상태에 이르는 모든 경로를 모두 시험할 수 있는 최소 시험 경로를 경로 표현식으로 나타내는 절차의 알고리즘을 그림 2와 같이 정의하였다. 이 알고리즘은 노드를 하나씩 제거해 나감으로써 전체 상태도에서 최소의 운행으로 각 상태를 한번씩 방문하는 절차이다.

4번째 단계에서 노드를 하나씩 제거하는 방법은 어느 임의의 상태 노드에 입력 링크 a, b가 있고 출력 링크로는 c, d, e가 있을 경우 (a + b)(c + d + e) 형태의 경로 표현식을 ac + ad + ae + bc + bd + be 형태로 변환한다. 이 단계의 성공적인 수행은 결국 하나의 시작 노드와 하나의 종료 노드를 얻을 수 있다

7번째 단계에서 루프를 제거하는 방법은 두 가지가 있다. 첫번째 방법은 먼저 자체 루프를 Z^* 형태로 변환하여 제거한 다음 여기에 모든 출력 링크들을 곱한다. 두번째 방법은 해당 노드를 2개로 나누어 A와 A'라는 두개의 동등한 노드를 생성한다. 그리고 두 노드 사이의 링크에 경로 표현식을 Z^*로 설정한다. 그런 다음 단계 4와 5를 이용하여 A'노드를 제거한다.

Q.2971 프로토콜의 시험 경로를 생성하기 위하여 먼저 프로토콜을 발신 프로세스와 착신 프로세스로 나누어 분석한다. 하나의 호에 대하여 발신과 착신 프로세스가 동시에 기동되지는 않기 때문에 시험 경로도 마찬가지로 나누어 생성하는 것이 바람직하다.

3.2 교환기 착신 호처리 프로세스

교환기에서의 착신 호처리 프로세스는 사용자로부터의 호 설정 요구 메시지를 접수하여 기능이 시작되므로 착신 호처리 프로세스에 대한 시험 경로는 Null 상태에서 사용자로부터 SETUP 메시지를 입력 받는 것에서 시작한다. 상태의 천이와 입력 메시지에 대한 내용은 표 1에 나타나 있지만 표의 내용에서 호 해제의 경우를 생략하고 간략하게 상태 천이도를 도시하면 다음 그림3과 같다. 상태 천이도에서 링크의 이름이 여러 개로 나열된 경우 즉, N1 상태에서 b_6, b_7, b_8, b_9와 같이 표현된 것은 각각의 링크가 병렬로 처리되는 병렬 링크이다.

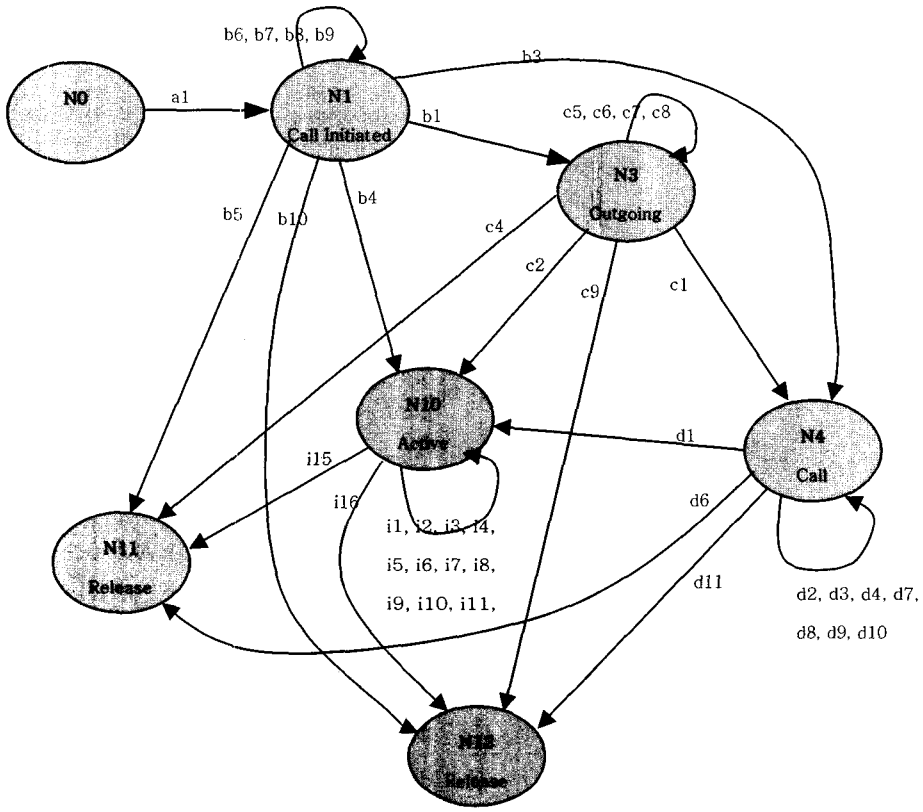
3.2.1 사용자가 호를 해제하는 경우

상태 N0에서 시작되어 상태 N11에서 호처리가 종료되는 경우에 대한 경로 식은 앞 절에서의 경로의 곱과 합을 구하여 대수식으로 나타내는 알고리즘을 적용하면 다음 식 1과 같이 구해진다.

$$a_1 (b_6^* + b_7^* + b_8^* + b_9^*)^* (b_1 ((c_5^* + c_6^* + c_7^* + c_8^*)^* (c_1 (d_2^* + d_3^* + d_4^* + d_7^* + d_8^* + d_9^* + d_{10}^*)^* (d_1((i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^*)^* i_{15}) + d_6) + c_2 (i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* +$$

(표 1) Q.2971호 상태 및 메시지(예)

상태 정보		입력 정보		출력 정보		비 고
상태 번호	상태이름	입력 정보	링크 부호	출력 정보	다음 상태	
NO	Null	SETUP	a1	SETUP(#), Setup ind.(#)	N1	Message to PC
		Setup req	a2	SETUP(#), Setup ind.(#)	N6	Primitive to PC
N1	Call Initiated	Proceeding req.(#)	b1	CALL PROCEEDING(#)	N3	
		RELEASE COMPLETE(#)	b2	RELEASE CMPLTE(#), Release conf.(cause, #)	X	Message to all PC, Process stop
		Alerting req.(#)	b3	Alerting req.(#), ALERTING(#)	N4	Primitive to PC
		Setup resp.(#)	b4	Setup resp.(#), CONNECT(#)	N10	Primitive to PC
		RELEASE(#)	b5	RELEASE(#), Release ind.(cause, #)	N11	Message to all PC
		DROP PARTY(#)	b6	DROP PARTY(#)	N1	Message to PC
		DROP PARTY ACK(#)	b7	DROP PARTY ACK(#)	N1	Message to PC
		Drop party req.(#)	b8	Drop party req.(#)	N1	Primitive to PC
		Drop party ack req.(#)	b9	Drop party ack req.(#)	N1	Primitive to PC
		Release req.(cause, #)	b10	Release req.(cause, #), RELEASE(cause, #)	N12	Primitive to all PC
N3	Outgoing Call Proceeding	Alerting req.(#)	c1	Alerting req.(#), ALERTING(#)	N4	Primitive to PC
		Setup resp.(#)	c2	Setup resp.(#), CONNECT(#)	N10	Primitive to PC
		RELEASE COMPLETE(#)	c3	RELEASE CMPLTE(#), Release conf.(cause, #)	X	Message to all PC, Process stop
		RELEASE(#)	c4	RELEASE(#), Release ind.(cause, #)	N11	Message to all PC
		DROP PARTY(#)	c5	DROP PARTY(#)	N3	Message to PC
		DROP PARTY ACK(#)	c6	DROP PARTY ACK(#)	N3	Message to PC
		Drop party req.(#)	c7	Drop party req.(#)	N3	Primitive to PC
		Drop party ack req.(#)	c8	Drop party ack req.(#)	N3	Primitive to PC
		Release req.(cause, #)	c9	Release req.(cause, #), RELEASE(cause, #)	N12	Primitive to all PC



(그림 3) 착신호 상태 천이도

$i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^{*})^* i_{15} + c_4) + b_3 (d_2^* + d_3^* + d_4^* + d_7^* + d_8^* + d_9^* + d_{10}^{*})^* (d_1 ((i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^{*})^* i_{15}) + d_6) + b_4 (i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^{*})^* i_{15} + b_5)$ (식 1)

식 1에서 자신의 노드로 천이되는 선택적인 링크를 식 2, 3으로 정의한다.

$B = (b_6^* + b_7^* + b_8^* + b_9^{*})^*$, $C = (c_5^* + c_6^* + c_{7P} + c_8^{*})^*$ (식 2)

$D = (d_2^* + d_3^* + d_4^* + d_7^* + d_8^* + d_9^* + d_{10}^{*})^*$, $I = (i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^{*})^*$ (식 3)

식 2와 식 3을 이용하여 식 1을 간략화하면 식 4가 구해진다.

$a_1 B (b_1 C (c_1 D (d_1 I i_{15} + d_6) + c_2 I i_{15} + c_4) + b_3 D (d_1 I i_{15} + d_6) + b_4 I i_{15} + b_5)$ (식 4)

식 4를 전개하면 식 5가 구해진다.

$(a_1 B b_1 C c_1 D d_1 I i_{15} + a_1 B b_1 C c_1 D d_6 + a_1 B b_1 C c_2 I i_{15} + a_1 B b_1 C c_4 + a_1 B b_3 D d_1 I i_{15} + a_1 B b_3 D d_6 + a_1 B b_4 I i_{15} + a_1 B b_5)$ (식 5)

식 5에서 시험해야 할 경로에 대한 경우의 수가 8가지가 됨을 알 수 있다. 이 8가지 경로에 대한 소프트웨어 제어 흐름 시험에 대한 경우의 수

를 산출하면 된다.

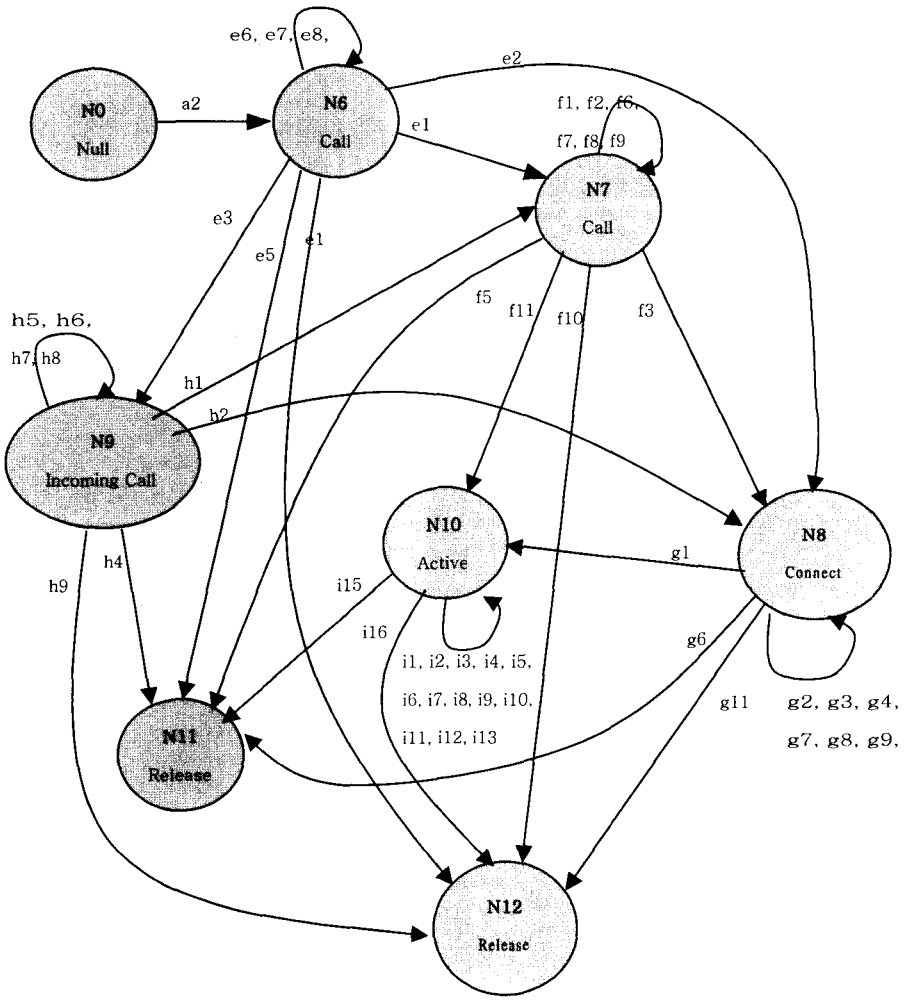
3.2.2 교환기가 호를 해제하는 경우

상태 N0에서 시작되어 상태 N12에서 호처리가 종료되는 경우에 대한 경로는 식 6으로 주어진다.

$$a_1 (b_6^* + b_7^* + b_8^* + b_9^*)^* (b_1 ((c_5^* + c_6^* + c_7^* + c_8^*)^* (c_1 (d_2^* + d_3^* + d_4^* + d_7^* + d_8^* + d_9^* + d_{10}^*)^* (d_1 ((i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^*)^* i_{16}) + d_{11}) + c_2 (i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^*)^* i_{16} + c_9) + b_3 (d_2^* + d_3^* + d_4^* + d_7^* + d_8^* + d_9^* + d_{10}^*)^* (d_1 ((i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^*)^* i_{16}) + d_{11}) + b_4 (i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^*)^* i_{16} + b_{10})$$ (식 6)

3.2.1절에서와 같이 루핑 경로를 정의한 식 2와 3을 식 6에 대치하면 식 7이 유도된다.

$$a_1 B (b_1 C (c_1 D (d_1 I i_{16} + d_{11}) + c_2 I i_{16} + c_9)$$



(그림 4) 발신호 상태 천이도

$+ b_3\ D\ (d_1\ I\ i_{16} + d_{11}) + b_4\ I\ i_{16} + b_{10})$ (식 7)

식 7을 전개하여 식 8을 구한다.

$(a_1\ B\ b_1\ C\ c_1\ D\ d_1\ I\ i_{16} + a_1\ B\ b_1\ C\ c_1\ D\ d_{11} + a_1\ B\ b_1\ C\ c_2\ I\ i_{16} + a_1\ B\ b_1\ C\ c_9 + a_1\ B\ b_3\ D\ d_1\ I\ i_{16} + a_1\ B\ b_3\ D\ d_{11} + a_1\ B\ b_4\ I\ i_{16} + a_1\ B\ b_{10})$ (식 8)

식 8에서 병렬적으로 존재가 가능한 경로 즉, 시험해야 할 경로에 대한 경우의 수가 또한 8가지가 됨을 알 수 있다.

3.3 교환기 발신호 처리 프로세스

교환기의 발신호 처리 프로세스는 응용 계층으로부터 호 설정 요구 프리미티브를 접수하여 기능이 시작되므로 발신 호처리 프로세스에 대한 시험 경로는 Null 상태에서 응용 계층의 Setup_req 프리미티브를 입력 받는 것에서 시작한다. 표 1에 나타난 상태의 천이도를 도시하면 다음 그림 4와 같다.

3.3.1 교환기가 호를 해제하는 경우

상태 0(N0)에서 시작되어 상태 11(N11)에서 호처리가 종료되는 경우에 대한 경로는 식 9로 정규화된다.

$a_2\ (e_6^* + e_7^* + e_8^* + e_9^*)^*\ (e_1\ (f_1^* + f_2^* + f6^* + f_7^* + f_8^* + f_9^*)^*\ (f_3\ ((g_2^* + g_3^* + g_4^* + g_7^* + g_8^* + g_9^* + g_{10}^*)^*\ (g_1\ (i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^*)^*\ i_{15} + g_6)) + f_{11}\ ((i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^*)^*\ i_{15}) + f_5) + e_2\ (g_2^* + g_3^* + g_4^* + g_7^* + g_8^* + g_9^* + g_{10}^*)^*\ (g_1\ (i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^*)^*\ i_{15} + g_6) + e_3\ (h_5^* + h_6^* + h_7^* + h_8^*)^*\ (h_1 + h_2\ (g_2^* + g_3^* + g_4^* + g_7^* + g_8^* + g_9^* + g_{10}^*)^*\ (g_1\ (i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^*)^*\ i_{15} + g_6) + h_4) + e_5)$ (식 9)

식 9에서 자기 자신에게로 반복되는 루핑 경로를 식 10, 11, 12로 정리한다.

$E = (e_6^* + e_7^* + e_8^* + e_9^*)^*,$

$F = (f_1^* + f_2^* + f_6^* + f_7^* + f_8^* + f_9^*)^*$ (식 10)

$G = (g_2^* + g_3^* + g_4^* + g_7^* + g_8^* + g_9^* + g_{10}^*)^*$ (식 11)

$H = (h_5^* + h_6^* + h_7^* + h_8^*)^*,\ I = (i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^*)^*$ (식 12)

식 10, 11, 12를 원래의 경로 표현식 식 9에 대치하여 간략화된 식 13을 구한다.

$a_2\ E\ e_1\ F\ f_3\ G\ g_1\ I\ i_{15} + a_2\ E\ e_1\ F\ f_3\ G\ g_6 + a_2\ E\ e_1\ F\ f_{11}\ I\ i_{15} + a_2\ E\ e_1\ F\ f_5 + a_2\ E\ e_2\ G\ g_1\ I\ i_{15} + a_2\ E\ e_2\ G\ g_6 + a_2\ E\ e_3\ H\ h_1 + a_2\ E\ e_3\ H\ h_2\ G\ g_1\ I\ i_{15} + a_2\ E\ e_3\ H\ h_2\ G\ g_6 + a_2\ E\ e_3\ H\ h_4 + e_5$ (식 13)

식 13에서 병렬적으로 발생 가능한 시험 경로가 11가지가 됨을 알 수 있다.

3.3.2 사용자가 호를 해제하는 경우

상태 0(N0)에서 시작되어 상태 12(N12)에서 호처리가 종료되는 경우에 대한 경로는 식 14와 같이 구해진다.

$a_2\ (e_6^* + e_7^* + e_8^* + e_9^*)^*\ (e_1\ (f_1^* + f_2^* + f_6^* + f_7^* + f_8^* + f_9^*)^*\ (f_3\ ((g_2^* + g_3^* + g_4^* + g_7^* + g_8^* + g_9^* + g_{10}^*)^*\ (g_1\ (i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^*)^*\ i_{16} + g_{11})) + f_{11}\ ((i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^*)^*\ i_{16}) +$

$f_{10}) + e_2 (g_2^* + g_3^* + g_4^* + g_7^* + g_8^* + g_9^* + g_{10}^*)^* (g_1 (i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^*)^* i_{16} + g_{11}) + e_3 (h_5^* + h_6^* + h_7^* + h_8^*)^* (h_1 + h_2 (g_2^* + g_3^* + g_4^* + g_7^* + g_8^* + g_9^* + g_{10}^*)^* (g_1 (i_1^* + i_2^* + i_3^* + i_4^* + i_5^* + i_6^* + i_7^* + i_8^* + i_9^* + i_{10}^* + i_{11}^* + i_{12}^* + i_{13}^*)^* i_{16} + g_{11}) + h_9) + e_{10}$ (식 14)

식 14에서 자기 자신에게로 반복되는 루핑 경로를 3.3.1절에서의 식 10, 11, 12와 같이 정의하고 원래의 경로 표현식 식 14에 대치하면 식 15와 같이 구해진다.

$a_2\ E\ e_1\ F\ f_3\ G\ g_1\ I\ i_{16} + a_2\ E\ e_1\ F\ f_3\ G\ g_{11} + a_2\ E\ e_1\ F\ f_{11}\ I\ i_{16} + a_2\ E\ e_1\ F\ f_{10} + a_2\ E\ e_2\ G\ g_1\ I\ i_{16} + a_2\ E\ e_2\ G\ g_{11} + a_2\ E\ e_3\ H\ h_1 + a_2\ E\ e_3\ H\ h_2\ G\ g_1\ I\ i_{16} + a_2\ E\ e_3\ H\ h_2\ G\ g_{11} + a_2\ E\ e_3\ H\ h_9 + e_{10}$ (식 15)

이 산술식 식 15에서 시험해야 할 경로에 대한 경우의 수 또한 11가지가 됨을 알 수 있다.

4. 루프 경로의 정규적 처리

3.2.1, 3.2.2, 3.3.1, 3.3.2절에서 구한 프로토콜의 시험 경로 표현식 식 5, 8, 13, 15에는 프로토콜에서 명세화한 프로시져에 따라 메시지 사이에 순서와 흐름이 있다. 특별히 식 3의 경우에, 반복 시험 경로의 집합인 D에서 d4와 d7은, ADD PARTY 메시지가 입력이 되어야 DROP PARTY 메시지의 입력이 가능하며, ADD PARTY 메시지가 없이 DROP PARTY 메시지가 입력 되는 경우에는 오류로 검출이 되어야 함을 명시하고 있다. 이러한 경로식의 순서에 관한 문제는 주로 반복되는 시험 경로에서 발생되는데 반복 시험 경로 표현식을 단순화 시켜서 시맨틱스 상의 오류를 지적할 수 있다. 이러한 반복 시험 경로에 대하여 페트리 네트나 동적 FSM(finite State Machine) 모델을 만들어 도달성 검사를 하게 되면 무한 루프에 빠지고 상태가 폭발하기 때문에 적용이 불가능하다. 따라서 간단하게 정규적인 표현을 이용하면 실용적으로 해결할 수 있다.

앞에서처럼 구해진 시험경로에 대한 정규 표현식은 단순화 시키기 위한 연산을 수행하는 순서에 따라 표현식이 서로 다른 형태를 가질 수도 있다. 궁극적으로 경로 표현식은 abc...의 형태를 갖도록 유도하여야 하며, 이를 위한 여러 등가식이 제시되었다[5]. 정적인 방법에 의해 시험 경로에 대한 정규 표현식이 구해지면, 노드를 제거하는 순서를 다르게 한 다음, 연속되는 링크와 병렬 링크, 반복 수행되는 링크들의 처리를 적용함에 따라 단순화가 이루어진다. 처리 순서의 다양함은 경로 표현식의 다양함으로 나타나는데, 이것은 동일한 경로 표현식이 여러 형태로 존재한다는 것을 의미한다. 이들 등가식은 반복 횟수가 무한할 경우에만 성립이 되므로 유한한 지수 값이나 "+" 기호를 적용할 때 주의가 필요하다.

식 2의 B에 대하여는, $B = (b_6^* + b_7^* + b_8^* + b_9^*)^* = (b_6^* b_7^* b_8^* b_9^*)^* = (b_6\ b_7\ b_8\ b_9)^* = 1 + b_6\ b_7\ b_8\ b_9 + (b_6\ b_7\ b_8\ b_9)^2 + ...$의 등가식으로 전개를 할 수 있는데 $b_6\ b_7\ b_8\ b_9$는 각각 서로 독립적으로 발생하는데, DROP PARTY(#)가 발생한 다음에는 DROP PARTY ACK(#), Drop party req(#) 프리미티브가 발생한 다음에는 Drop party ack req(#)가 발생하는지 순서 시맨틱스만 시험하면 된다. 따라서 시험 횟수의 문제는 Party 가입자 호를 허용하는 횟수만큼 ($b_6\ b_7$), ($b_8\ b_9$) 링크열에 대한 반복 시험을 수행하면 된다. 3.3.1절의 교환기가 호를 해제하는 경우에서 간략화된 정규 표현식의 첫번째 항을 예로 들어 정리하면 다음과 같다.

$a_2\ E\ e_1\ F\ f_3\ G\ g_1\ I\ i_{15}$

/* $(e_6^* + e_7^* + e_8^* + e_9^*)^*$의 분석 */

$= a_2\ (1 + e_6\ e_7\ e_8\ e_9 + (e_6\ e_7\ e_8\ e_9)^2 + ...\ )\ e_1$

$(1 + f_1\ f_2\ f_6\ f_7\ f_8\ f_9 + (f_1\ f_2\ f_6\ f_7\ f_8\ f_9)^2 + ...)\ f_3\ (1 + g_2\ g_3\ g_4\ g_7\ g_8\ g_9\ g_{10} + (g_2\ g_3\ g_4\ g_7\ g_8\ g_9\ g_{10})^2 + ...)\ g_1\ (1 + i_1\ i_2\ i_3\ i_4\ i_5\ i_6\ i_7\ i_8\ i_9\ i_{10}\ i_{11}\ i_{12}\ i_{13} + (i_1\ i_2\ i_3\ i_4\ i_5\ i_6\ i_7\ i_8\ i_9\ i_{10}\ i_{11}\ i_{12}\ i_{13})^2 + ...)\ i_{15}$

$= a_2\ (1 + e_6\ e_7\ e_8\ e_9 + (e_6\ e_7\ e_8\ e_9)^2 + ...)\ e_1\ f_3\ g_1\ i_{15} + ...$

$= a_2\ e_1\ f_3\ g_1\ i_{15} + a_2\ (e_6\ e_7 + e_8\ e_9)\ e_1\ f_3\ g_1\ i_{15} + ...$

$= a_2\ e_1\ f_3\ g_1\ i_{15} + a_2\ e_6\ e_7\ e_1\ f_3\ g_1\ i_{15} + a_2\ e_8\ e_9\ e_1\ f_3\ g_1\ i_{15} + ...$

상태가 천이 되는 모든 가능한 순서, 즉 검사하고자 하는 시험 경로의 순서를 정규 표현으로 나타내고 이 표현식이 프로토콜에서 수용하고 있는지를 시험할 수도 있다[6]. 이러한 문제는 시험 경로의 정규적인 표현에서 반복 시험 경로를 반복되지 않는 경로 표현식으로 전개할 반복 횟수를 검사할 시험경로의 링크 수 만큼만 취하면 된다. 검출하고자 하는 시험경로가 2개의 링크로 구성되어 있다면 2개의 링크가 정규적 표현에 포함되는지 검사하는 문제이므로 반복 횟수가 2차인 표현식을 유도하면 된다. X* 형식의 모든 표현을 $1 + X^2$ 형태로 대체하면, 이 대체된 결과로 구해지는 시험 경로에 의해 주어진 두개의 문자열이 존재하는지 아닌지를 결정할 수 있음을 보여준다. 이러한 변환은 두 번 반복 되는 것을 분별하는 방법이다. 두 개의 문자열로 표현되는 프로토콜 메시지 상의 오류는 시험 경로가 두 번 반복되는 것을 찾는 것과 같다. 위의 방법은 시험경로가 세 링크 이상이 되면 복잡해져 수동적으로 처리하기는 어렵다.

임의의 메시지 흐름에 대한 적합성 및 강건성을 시험하기 위한 주어진 경로로 시작하거나 종료하는 시험열을 찾는 알고리즘도 있지만[13], 이들은 표현식 내부에 숨어 있는 문자열을 찾아내기에는 적합하지 않은데 비해 본 논문에서와 같이 정규적인 표현기법을 이용하면 시작 순서열이나 종료 순서열을 육안으로 검사할 수 있기 때문에, 임의의 시험 경로를 찾는 것을 대수적으로 처리할 수도 있지만 수학적으로 부가 처리할 필요성을 느끼지 않는다. 정규 표현식을 사용하는 주된 이유는 편리한 표기법과 시험 경로 또는 일련의 시험 경로들을 추적하는 방법 때문이다. 표현식에 대하여 대수적으로 처리하거나 유추하는 것이 상태 천이도에서 잘못된 경로를 추적하는 것보다 쉽다는 것을 알 수 있다.

5. 결론 및 향후 연구과제

페트리 네트나 동적인 FSM을 이용하여 프로토콜의 시험열을 구하는 문제는 오류를 허용하거나 반복 루프가 발생하는 경우에는 자동적인 생성이 현재로는 제시되지 않고 있다. 이 문제를 정규 표현식에 의해 프로토콜의 최소한의 시험 경로를 구하는 알고리즘을 제시하고, 이를 Q.2971 프로토콜에 적용하여 최소 시험 경로를 정적으로 구할 수 있음을 보였다. 결과로 38가지 경우의 시험 경로가 있음을 확인하였다. 이는 페트리 네트나 다른 동적인 시험열을 생성하는 자동화 도구를 이용하였을 때 발생하는 무한 루프 현상과 수행 모델을 수정하기 위한 노력이 요구되지 않아 간편함을 확인하였다.

시험 경로에 대한 종류가 얻어지면, 각각의 시험 경로에 대한 소스 프로그램을 제어흐름도(control flow diagram)로 나타내고 이를 프로토콜 상태 천이도를 이용한 것과 마찬가지로 내부적인 제어 시험 경로를 도출하여 화이트 박스 시험이 가능하다. 내부 시험 경로는 프로토콜 시험 경로의 반복 횟수를 프레디키트(predicate)의 값으로 할당되어야 하는데 아쉽게도 정규 표현식으로 구해진 시험 경로에 대한 반복 시험 횟수를 구하는 문제는 Q.2971에서와 같이 정형화할 방법은 없으며, 각 시스템 별로 작성되는 규격서의 용량 조건을 참조하여 결정할 수 밖에 없다. 경로별로 프레디키트을 부여하는 방안과 이에 따라 흐름을 실행시키기 위

한 연구가 계속 이루어져야 할 것으로 사료된다.

참 고 문 헌

[1] Gerard J. Holzmann, Design and Validation of Computer Protocols, Prentice Hall, 1991.

[2] Bechet Sarikaya, "Principles of Protocol Engineering and Conformance Testing," Ellis Horwood, 1993.

[3] T. Chow, "Testing Software Design Modelled by Finite-State Machines," IEEE Trans. on Software Engineering, Vol. SE-4, Mar. 1978, pp.178-187.

[4] B. Sarikaya, G. V. Bochmann, "Some Experience with Test Sequence Generation for Protocols," PSTV II, North-Holland, 1982, pp. 555-567.

[5] Boris Beizer, Software Testing Techniques, 2nd edition, Van Nostrand Reinhold, 1990.

[6] Brzozowski, J. A., McCluskey, E. J., "Signal flow graph techniques for sequential circuit state diagrams," IEEE Transactions on Electronic Computers 12, 1963, pp.67-76.

[7] 이현정, 최영일, 이병선, "SDL로부터 프로토콜 시험열 자동 생성 기법", 한국통신학회논문지, 2000, 7 V.25, N.7B, pp.1253-1259

[8] J M. Atlee, J Gannon, "State-Based Model Checking of Event-Driven System requirements," IEEE Transaction on SW Eng., Vol. 19, No 1, pp. 24~40, Jan., 1993.

[9] 박진호, 양대헌, 송주석, 임상용, "완전표준성을 만족하는 선행 검증 시험열 생성방법에 관한 연구", 한국통신학회논문지, 1998, 09 V.23, N.9A, pp.2383-2390.

[10] Dong-Won Jang, Sang-kook Jeong, Myung-hee Kim, Han-kyoung Kim, "Automatic Test Case Generation of Wireless Interface Protocols for Next Generation Mobile Telecommunications," Proceeding of International Conference on Communication Technology ICCT'98, Oct.,22-24, 1998, pp.S22-07-1~5

[11] ITU-T Rec. Q.2971 Broadband Integrated Services Digital Network(B-ISDN) Digital Subscriber Signalling System No.2(DSS2) User-Network Interface Layer 3 Specification for Point-to-Multipoint Call/Connection Control, 1997.

[12] ITU-T Rec. Q.2931 Broadband Integrated Services Digital Network(B-ISDN) Digital Subscriber Signalling System No.2(DSS2) User-Network Interface Layer 3 Specification for Basic Call/Connection Control, 1997.

[13] Huang, J.C., "Detection of Data Flow Anomaly through Program instrumentation," IEEE Transactions on Software Engineering 5, 1979, pp.226-236.

◐ 저 자 소 개 ◑

김 한 경

1973년 서울대학교 원자력공학과(공학사)
1987년 충북대학교 대학원 전산통계학과(이학석사)
1992년 충북대학교 대학원 전자계산학과(이학박사)
1997~현재 창원대학교 컴퓨터공학과 교수
관심분야 : 정보통신, ATM, MPLS, 소프트웨어공학
E-mail : hkim@sarim.changwon.ac.kr