ATM 교환기 S/W 검증을 위한 테스팅 도구 설계 및 구현

정 창 신[†] · 황 선 명[††] · 이 경 환[†††] · 김 행 곤[††††]

요 약

ATM 교환기 소프트웨어 품질 특성은 신뢰성이 높고 기능성, 확장성 및 유지 보수성이 좋아야만 한다. 소프트웨어의 개발 과정이나 설계 후에 이러한 품질 특성이 만족되었는지의 여부를 평가하고 검증하는 테스팅 도구들은 많이 개발되어 왔으나 ATM 관련 소프트웨어의 품질 평가 도구는 기능성에서 뒤떨어지고 품질면에서도 부적합하다.

본 논문에서는 CHILL 언어로 작성된 ATM 교환기 소프트웨어를 대상으로 이를 평가하기 위한 테스팅 분석 자료, 디버깅 및 유지 보수 정보를 제공하는 정적 분석 도구를 설계하여 구현한다. 이 도구는 입력 프로그램에 대한 높은 가시성을 제공하며 병렬 프로세싱에 관한 정보를 통계표와 그래프로 나타내는 특징을 갖는다.

Implementation of Testing Tool Verification of ATM Switching Software

Chang-Sin Chung[†] · Sun-Myung Hwang[††] · Kyung-whan Lee[†††] · Haeng-Kon Kim[††††]

ABSTRACT

ATM switching software should be required high reliability, functionality, extendability and maintainability. After development of the software, it is verified and tested by analyzer whether the software is accomplished the characteristics of it or not. There are so many CASE tools in other area, but the CHILL testing tools that can verify ATM softwares have not various functions are not many.

In this paper we develop the testing tool which can evaluate and test CHILL programmed ATM software. This Tool supports testing reports, debugging informations and maintenance informations including parallel process about CHILL original programs.

1. 서 론

교환기용 소프트웨어는 일반응용 소프트웨어와는 달리 실시간 처리가 가능하고 고도의 안전성 및 신뢰성을 갖추어야 한다. 이 때문에 사회 전반적으로 컴퓨터와 통신 서비스 등에 대한 이론 연구와 응용 연구가 심도 있게 진행되어 교환기 분야의 다양한 서비스 요구에 따른 소프트웨어의 분산화, 고신뢰도화, 유지보수화가 절실히 요구되어왔다. 이에 대응할 수 있는 새로운 프로그래밍 언어로 CCITT에서 권고한 CHILL (CCITT High Level Programming Language) 언어는 소프트웨어 개발 언어로 기존의 프로그래밍 언어인 Module-2. PASCAL, C, FORTRAN 등의 특징을 대부분 포함하는 언어이다. 또한 CHILL은 프로그래밍할 때 발생하는 오류를 줄이고, 모듈 프로그래밍, 구조적 프로그래밍, 병렬처리, 분산컴파일, strong type checking, 데이터 추상화 등의 기능을 갖고있다[1][2].

† 정 회 원 : 한국전자통신연구원
†† 종신회원 : 대전대학교 컴퓨터공학과
††† 정 회 원 : 중앙대학교 컴퓨터공학과
†††† 정 회 원 : 대구효성가톨릭대학교 전자정보공학부
논문접수 : 1996년 12월 19일, 심사완료 : 1997년 7월 10일

기존에 개발된 테스팅 도구들을 살펴보면 프로그램 복잡도 측정이 가능한 McCabe Tools과 정적분석과 동적분석 기능을 함께 가진 Verilog사의 LOGISCOPE, 그래프 정보를 포함한 경로테스팅 정보를 제공하는 SR사의 STW 도구 등 개발 과정별로 여러 테스팅 도구들이 있다[11]. 그러나 이들 도구의 대부분이 고가인 동시에 기능면에서도 단일 기능을 제공함으로 다양한 분석 정보가 요구되는 환경에서는 부족함이 있다. 아울러 본 논문의 대상분야인 교환기 소프트웨어는 대부분 CHILL 언어로 작성되어 있는데 CHILL 프로그램에 대한 CASE 도구는 더욱 부족한 실정이다.

본 논문은 제어흐름 분석을 기반으로 하여 CHILL 언어로 작성된 ATM 교환기 소프트웨어에 대한 검증을 목적으로 정적분석자료를 작성하는 시험기(testing tool)를 개발하고자 한다. 일반적으로 모든 시험기는 시험대상 소프트웨어(프로그램)의 구현언어에 의존하여 구성된다. 그러므로 CHILL 언어로 구현되어있는 ATM 교환기 소프트웨어에는 CHILL 언어문법과 의미에 맞도록 시험기를 구현하여야 한다.

CHILL의 특징을 살펴보면 MODULE은 CHILL 프로그램의 구성단위일 뿐 아니라 프로그램상에 나타나는 이름들에 대한 가시성을 제공해주며 PROCEDURE와 BEGIN-END 블록은 구조적 프로그래밍 개념을 제공한다. 또한 프로그램상에서 동시에 수행되는 프로그램 동적인 부분인 프로세스의 병렬수행 기능을 제공한다[2].

CHILL 언어의 특징을 살펴보면 크게 다음과 같다.

(1) 가시성 제어(Visibility control)

CHILL 프로그램 구성단위는 Modulion(MODULE, REGION)과 블록(PROCEDURE, PROCESS, BEGIN-END)이 있으며 이들은 선언된 데이터 변수들에 대한 사용가능범위를 가시화 한다. 블록은 이름의 가시성과 존재성을 모두 결정하지만 Modulion은 가시성만을 결정한다.

(2) 모드제어(Mode control)

CHILL에 있어서 모든 장소는 하나의 모드를 가지며, 이 장소의 모드는 그 장소에서 저장 될 수 있는 값(value)들의 집합과 그러한 값들에게 허용되는 operation들을 결정한다.

(3) 병렬처리(Parallel Processing)

병렬처리란 임의의 프로그램들이 다중 프로세서 시스템에서 병렬로 처리되도록하거나, 단일 프로세서 시스템상에서 비동기적으로 동시에 처리되도록 하는 것을 의미한다.

1960년대에 각 장치별로 프로그래밍이 가능한 독립적인 장치들이 컴퓨터 시스템에 접속되기 시작하였으며, 이러한 시스템의 각 장치에 대한 제어와 장치간의 자료 전달 및 처리를 위하여 동시에 수행되는 프로세서들의 집합체로 운영체계가 구성됨으로써 시작된 병렬 처리에 대한 연구는 운영체계 분야에서 오랜 동안 중요한 개념으로 인식되어 왔다.

이와 더불어, 프로세서들의 상호 동작을 위하여 필요한 프로세스간 통신 및 동기화와 관련되어 운영체계에 추가된 병렬처리 개념들이 프로그래밍 언어에 반영되어 사용자 직접 병행 및 병렬 프로그램을 작성하기 위한 도구로 발전하게 되었다. 또한, 병렬 처리의 기본 단위로 프로세스 개념이 주로 사용되어 왔으나, 근래에는 쓰레드 개념을 병렬 프로그래밍에 도입하여 프로그램의 실행을 효율적으로 개선하려는 많은 연구가 수행되었으며, 최근에는 객체 지향 프로그래밍 언어에서 객체 클래스의 병렬처리에 대한 연구가 활발히 진행되고 있다.

본 논문은 이와같은 CHILL 입력프로그램의 특징을 분석하여 구문 분석을 통하여 가시화 정보, 메트릭스 정보, 각종 통계정보, 그래프 정보와 병렬 프로세싱 정보 및 인터페이스 정보등 다양하고 효과적인 분석 평가 정보를 제공하는 도구를 개발한다.

2. CHILL 정적분석기(CSA) 설계

2.1 정적분석과 동적분석

정적분석은 소프트웨어 시스템의 실행을 동반하지 않고 그 정당성을 조사하는 것으로 프로그램 뿐만 아니라 요구, 설계, 코딩과정에서 생성된 문서를 체크하는데도 이용되지만 여기에서는 프로그램에 대한 정적 분석만을 다루기로 한다.

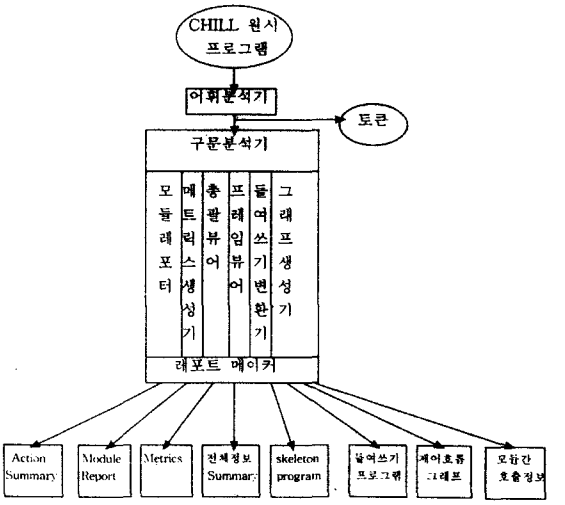
에러발생 가능한 프로그램 구조에서 정적 에러 분석에 대한 잘 알려진 한 예로 자료흐름의 변칙을 찾는 것이다. 즉, 프로그램에 있는 문장이 실행될 때 기억장소로부터 어떤 변수의 값을 가져오는 변수의 참

조(reference)와 변수에 값을 할당하는 정의(define), 그리고 어떤 변수가 특정지역에서 정의되지 않는 비정의(undefine)들간의 관계에서 발생할 수 있는 참조의 변칙(reference anomaly)을 발견하는 일이다. 또한 프로그램의 제어흐름 그래프를 작성하거나 변수의 단순한 정의와 참조에 대한 정보를 찾아 이들의 활동범위(scope)를 제공할 수 있다.

원시코드의 실행 상태에 관한 자료나 특정한 목표의 달성도를 측정하기 위하여 동적분석에서는 프로그램내에 목표에 맞는 적당한 계측도구(probe)를 삽입하는 방법이 있는데 이를 계측도구(instrumentation)이라고 한다. 이는 분석을 위한 특별한 코드를 프로그램내에 삽입하므로 프로그램 실행중에 순간적인 변화를 체크할 수 있다[3][6]. 인스트루멘테이션은 계측에 필요한 필수적인 과정인데 이때 주의해야 할 점은 테스트 프로그램의 문장들간 실행순서와 결과에 변화를 주어서는 안된다는 점이다. 즉, 프로브가 삽입되더라도 프로그램의 실행순서 및 결과에 영향을 주지 않도록 한다는 것이다[8].

2.2 정적 분석기

현재 실용중인 ATM 교환기 소프트웨어를 대상으로 UNIX 시스템 환경에서 이들을 테스팅하고 디버깅 정보를 얻으며, 나아가 성능 및 품질 향상을 도모하기 위하여 개발하고자 하는 CHILL 정적 분석기의 구조는 다음 (그림 1)과 같다.



(그림 1) 정적 분석기의 구조
(Fig. 1) Components of CASE

2.3 정적 분석기 콤포넌트

2.3.1 어휘분석기(Lexical Analyzer)

CHILL 프로그램에서 나타날 수 있는 모든 예약어, 키워드 등을 토큰단위로 출력하는 분석기로서 테스트 대상 CHILL 프로그램을 입력으로 받아서 제일 먼저 처리해야 할 시험기(분석기)의 첫단계이다.

2.3.2 구문분석기(Syntetic Analyzer)

입력된 CHILL 프로그램 문장별 문법구조에 따라 사용자가 원하는 관심테스트 항목에 따라서 다음과 같은 기능을 수행가능토록 한다.

(1) 모듈레포터(Module Reporter)

테스트 대상 프로그램은 모듈단위로 테스트 할 수도 있고 모듈내의 파일단위 또는 프로시듀어 단위로 분석자료를 볼 수 있도록 한다.

(2) 메트릭스 생성기(Metrics Generator)

프로그램 품질 측정을 위한 정보를 제공하는 메트릭스로서 Mccabe의 복잡도인 V(G)와 Halstead의 노력평가 메트릭스를 제공한다.

(3) 그래프 생성기(Control Graphic Drawer)

제어흐름 그래프는 프로시듀어 별로 노드와 아크를 연결시켜 그리게 되는데 이때 새로이 병렬수행 프로세스가 발생하게 되면 새로운 프로세스의 제어흐름에 관련된 정보도 함께 제공된다.

그래프의 가시성을 높이기 위하여 각 노드마다 노드의 레이블을 붙이고 그래프의 높이(height)와 넓이(width)를 고려하여 노드를 배치하였으며[4], 아크(arc)연결시 문장구조와 주변노드의 영역을 분석하여 아크와 아크사이에 knot가 발생하지 않도록 고려하였다.

(4) 단위 프로시듀어 마다의 구체적 정보를 제공하는데 특히 해당 프로시듀어에서 함수들의 종류, 호출횟수, 호출당하는 프로시듀어와 그 횟수, 병렬처리되는 프로세스 이름과 병렬처리 횟수 등의 정보를 제공

한다.

(5) 레포트 포매터(Report Formatter)

분석된 레포트 자료를 가시성을 높이고, 분석자료의 이해를 높이기 위한 출력 형태를 만드는 기능을 갖는다.

3. CHILL 정적 분석기의 세부기능 및 설계 방법

3.1 총괄 정보

3.1.1 파일정보

- 총괄 정보를 통계적 수치로 제공
- CHILL 원시 프로그램의 파일 개수, 전체 라인수, 문자수, 토큰수
- CHILL 프로그램의 문장분석정보
  - 일반 문장 개수와 전체 문장에대한 비율
  - Comment 문장 개수와 전체 문장에 대한 비율
  - DB Access 문장 개수와 전체 문장에 대한 비율
  - Preprocessor 문장 개수와 전체 문장에 대한 비율
  - Empty/Blank 문장 개수와 전체 문장에 대한 비율

3.1.2 프로그램 내부 정보

- 전체 문장에 대한 통계정보를 제공
- 프로그램 문장 구조별 출현 횟수
- 프로그램에 나타난 제어 구조문의 종류

Total files	Total lines	Total chars	Total tokens
4	1693	57628	7767

Statement Analysis

	Codes	Comments	DB Access	Preprocessor	Empty/Blank
Num of	989	339	25	8	332
Lines	58.42%	20.02%	1.48%	0.47%	19.61%
Num of	31749	14775	1337	168	9599
Chars	55.09%	25.64%	2.32%	0.29%	16.66%

Action Summary

Assignment	391	CALL	130	GOTO	0
CONTINUE	0	IF	74	ELSEIF	11
RETURN	5	CASE	22	CASE alternatives	99
DO	9	DELAY Case	0	DELAY alternations	0
START	21	RECEIVE Case	2	RECEIVE alternatives	22
STOP	21	EXIT	3	Builtin Action call	0
DELAY	0	SEND	20	DB Access	25

(그림 2) 총괄 정보의 예
(Fig. 2) Outputs of overviewer

3.1.3 프레임 뷰어

원시 프로그램의 각 프로시듀어, 프로세스 별 전체 구조파악이 수월하도록 키워드 중심의 핵심 문장으로 표현하고 해당 중심문장의 원시 프로그램에서의 위치(전체 라인 번호)정보와 구조화 정도(depth) 및 핵심문장 각각의 범위를 블록 형태로 표시한다.

3.1.4 메트릭스 생성기

원시 프로그램의 복잡도를 프로시듀어 별(단위별)로 구분하여 McCabe 복잡도(VCG)와 Halstead의 분석 정보를 제공한다.

(1) McCabe의 MCC

- V(G) 값은 프로그램의 복잡도 뿐만아니라 테스팅 기본 경로의 수로도 이용되어 경로테스팅시에 테스트 경로를 선정하는데도 사용된다.
- V(G) = e − n + 2(e : edge수, n : node의 수)
- V(G) = 경로수 − 문장수 + 2
- V(G) = 제어흐름 그래프의 폐구간 + 1
- V(G) = IF count + 1

(2) Halstead의 Hss

- Halstead의 메트릭스는 연산자(operator)와 피연산자(operand)의 정보로 프로그램의 길이, 노력, 신뢰도 측정에 이용된다.

  n1 = unique operators
  N1 = total operators
  n2 = unique operands
  N2 = total operands

- Vocabulary-n = n1 + n2
- Length N = N1 + N2
- Estimated length N* = (n1 × log2(n2) + (n2 × log2(n2))
- purity ratio = N* ÷ N
- Volum V = N × log2(n)
- Effort E = V ÷ L

3.1.5 들여쓰기

원시코드 전체에 대한 들여쓰기를 통하여 구조화 코딩을 유도하였으며 각 문장들의 구조화 정도를 들여쓰기 정도로 측정한다.

3.2 그래프 생성기

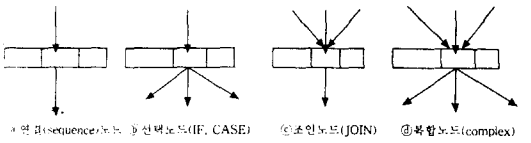
코드 분석기중 가장 가시성이 높은 정보로써 원시 코드를 흐름 그래프로 표현한다. 이때 사용되는 그래프의 노드와 아크에 관한 내용은 다음과 같다.

3.2.1 노드의 작성

그래프를 구성하는 요소는 노드와 아크로서 먼저 노드의 설계는 다음과 같은 방법에 의하여 작성한다.

원시코드가 입력된 상태에서 skeleton code 정보가 생성되며 각 skeleton 코드로부터 노드 정보를 얻게 되는데, 이때의 핵심 문장에 대한 노드 표현은 한 노드가 이전의 모든 순차구조의 문장들을 포함하고 있는 것으로 간주한다. 노드의 구성은 연결 리스트 구조를 갖는데 첫 번째 영역은 현재 노드의 정보를 갖고 있으며 두번째 영역은 자식노드를 가리키고 다음 연결노드를 가리키며, 마지막은 형재 노드들을 가리키게 되는데 이 경우 loop 구조일때는 마지막 영역은 loop 시작지점으로 연결 된다.

이상의 노드 표현을 위하여 원시 프로그램의 구조를 크게 4가지 유형으로 구분하였는데 그 유형은 다음과 같다.



(그림 3) 노드 표현을 위한 연결 리스트 타입
(Fig. 3) Linked list types for node

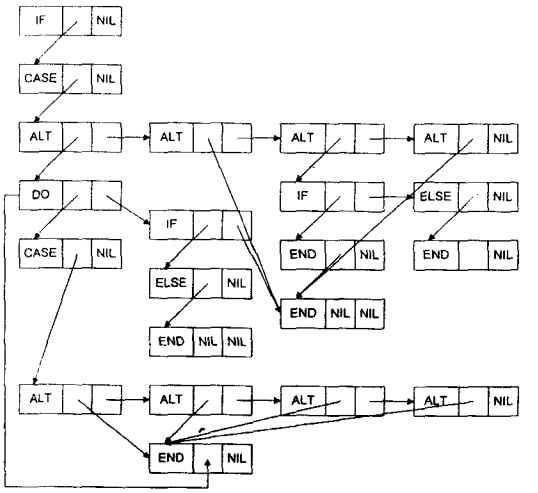
이와 같은 노드의 유형을 연결리스트 표현으로부터 노드의 간접적인 위치와 노드간 관계를 파악하고 노드의 실제 표현시 노드에 레이블을 붙이는데 이때 노드의 이름은 다음과 같다.

- start : 시작노드
- if : IF의 ELSE 관련 노드
- then : IF의 THEN 관련노드
- switch : CASE 구조에 관련된 노드
- case : CASE 구조의 각 경우에 대한 노드
- join : 여러구조의 합에 대한 노드
- fork : 새로운 프로세스의 생성을 표시하는 노드
- end : 끝 노드

다음은 입력 코드를 변환한 skeleton code에 대하여 노드 구성과 그의 연결 리스트 작성에 대한 예를 보인다.

```
$$test:PROCEDURE
:IF
:CASE
:CASE-ALTERNATIVE
:DO
:CASE
:CASE-ALTERNATIVE
:CASE-ALTERNATIVE
:CASE-ALTERNATIVE
:CASE-ALTERNATIVE
:END-CASE
:END-DO
:CASE-ALTERNATIVE
:CASE-ALTERNATIVE
:IF
:END-IF
:CASE-ALTERNATIVE
:END-CASE
:STOP
:ELSEIF
:ELSE
:END-IF
:END PROCEDURE
```

(그림 4) 노드 생성을 위한 skeleton code
(Fig. 4) Skeleton codes for node generation



(그림 5) 노드 생성을 위한 연결 리스트
(Fig. 5) Linked lists for node generation

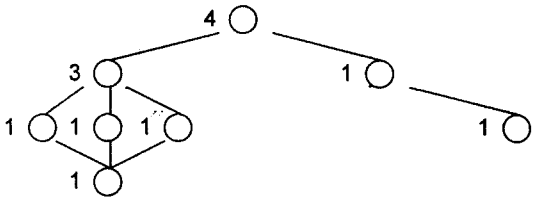
3.2.2 노드의 높이(height)와 넓이(width)

노드의 높이와 넓이는 프로시듀어별로 그래프를

작성할 때 한 페이지 내에서 균형 잡힌 그래프를 작성하기 위하여 고려된다. 본 연구에서는 넓이(width)는 40개 까지의 형제노드가 올 수 있도록 하였으며 높이는 start 노드에서부터 end 노드까지 25개의 순차노드가 올 수 있도록 하였다. 그 이상의 높이나 넓이의 노드를 나타내면 더 이상 그래프 그리는 것을 멈춘다. 출력 양식안에 노드의 위치를 결정하기 위하여 노드의 넓이와 높이를 결정하는 방법은 다음과 같다.

첫째, start node는 양식의 상단 중앙에 위치하도록 좌표(x,y)결정

둘째, 노드의 종류 분석 : decision, join, sequence, complex 등



(그림 6) Complex 노드의 넓이 결정
(Fig. 6) Determination for complex node

셋째, 넓이의 결정
- decision, sequence 노드인 경우 : Σ(자식노드 수)
- join node : 1
- complex node : 결과값을 delay한 후 최후 결정

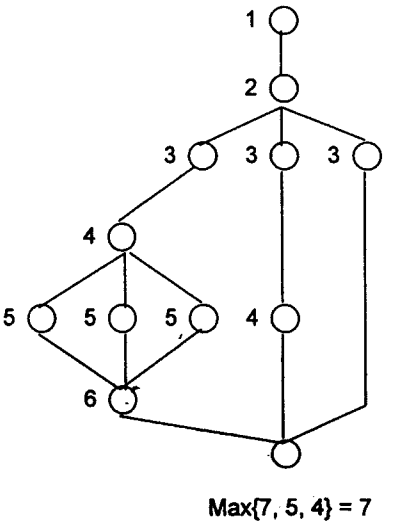
넷째, 높이의 결정
- 한 자식 노드 레벨로 내려올 때마다 1씩 증가
- join 노드의 높이는 다른 노드 결정될 때마다 delay한 후 최종 결정
- 만일 한 노드의 높이값이 서로 다른 값일 경우 (그림 7)은 최대값을 선택한다.

3.2.3 arc의 표현

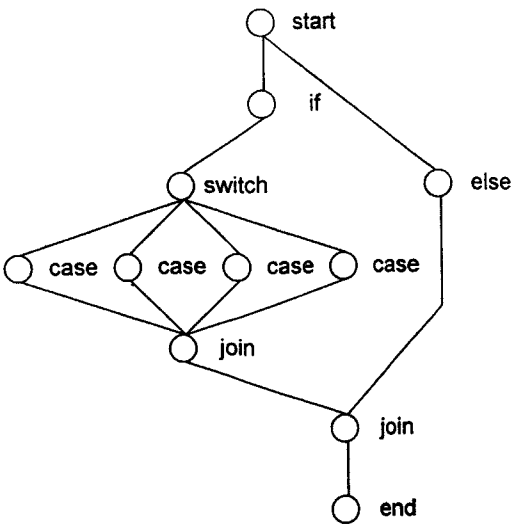
아크는 한 노드에서 그의 자식노드로의 연결은 모두 직선으로 하였으며 그 이상의 관계(grand-child 이상)에서는 곡선을 이용하였다. 또한 주변 다른 노드들을 고려하여 기존의 아크와 중복되어 겹쳐지는 현상을 막기위하여(knot 발생방지) 기존의 노드의 범위를 계산한 후 arc를 연결하도록 하였다. 이때 연결선(arc)은 실제적으로 유향(directed arc)으로 표시되어야 하지만 편의상 방향성은 표시하지 않았다.

3.2.4 프로세스 생성 노드

한 프로시듀어나 프로세스의 어떤 노드에서 다른 프로세스가 생성될 때 이때의 노드를 일반 노드와 구



(그림 7) 그래프의 높이 결정
(Fig. 7) Determination of height



(그림 8) 흐름 그래프에서의 노드와 아크 표현
(Fig. 8) Labeled control flow group

별하기 위하여 노드의 크기를 크게 표현하였다.

이 같은 노드와 아크의 표현으로부터 knot나 스크래칭(scraching)이 발생되지 않는 흐름 그래프를 그릴 수 있으며 (그림 8)은 임의의 프로시듀어에 대한 흐름 그래프의 예이다.

3.3 호출관련 정보 생성기

모듈, 프로시듀어, 프로세스간의 모든 호출 관련 정보를 제공함으로 이들간의 인터페이스가 올바른가를 판단할 수 있다.

3.3.1 파라메터 정보

해당되는 모듈이나, 프로시듀어 형식 파라메터(formal parameter)들을 출력하고 이들 각각의 타입은 어떤것인지를 알 수 있다.

3.3.2 callee/caller 정보

해당 모듈이나 프로시듀어 또는 프로세스가 호출하는 타 모듈이나 프로시듀어의 종료와 호출횟수가 출력된다. 이때 테스트 대상 프로그램내에 속한 모든 함수들도 함께 출력된다.

또한 현재의 모듈 또는 프로시듀어를 어느 모듈(프로시듀어)이 호출하고 있는지 상위 모듈(프로시듀어)에 대한 호출정보를 제공한다.

3.3.3 병렬프로세스 생성정보

현재 테스팅 모듈 또는 프로시듀어(프로세스)의 실행중에 생성되는 자식 프로세서들의 종류와 횟수 등에 관한 정보를 제공한다. 또한 현재의 프로세스가 어떤 모듈이나 프로시듀어(프로세스)에 의하여 생성되는지 이에 관련된 정보를 알 수 있으므로 현 프로세스의 상위 프로세스 정보와 하위 프로세스 정보로부터 현재 병렬 수행되고 있는 프로세스의 개수와 종류들을 파악할 수 있다.

Callees					
printf	2	pd_DisSpcInfo	1	pd_DisSpcConf	1
pd_DisSpcCnt	1	pd_DisPtmpInfo	1	pd_DisPtmpConf	1
pd_DisImconf	1	pd_DisImkConf	1	pd_DisImSts	1
pd_DisImkSts	1	pd_DisVpcSts	1		
Creaters					
UNIF_main	10				
No_Start_actions					

(그림 9) 프로세스에서의 호출관련 정보
(Fig. 9) Interface informations of a process

4. 결 론

모든 교환기 소프트웨어는 CHILL 언어에 의해 대부분 구현되어 왔으나 이들의 검증, 분석, 시험을 위한 CASE 도구들은 아직까지 찾아보기 힘든 상황이다. 정확성과 신뢰성 및 실시간 처리등 여러 가지 품질특성을 만족하기 위한 소프트웨어를 생산하기 위하여는 먼저 그 소프트웨어의 테스팅 과정과 검증과정이 철저하여야 한다.

본 논문은 CHILL 언어로 구현된 소프트웨어에 대한 정적분석기를 설계함으로 교환기 소프트웨를 대상으로 품질향상을 도모하고자 한다. 보다 다양하고 정확한 시험기가 바람직 하지만 본 논문에서는 정적분석기만의 기능을 중점적으로 설계하도록 하였다. 개발된 정적분석기의 기능은 매우 다양하여 각종 프로그램 분석 통계자료를 포함하여 복잡도 메트릭스 값을 산출하고, 스크래칭이 발생되지 않도록 가시화를 높인 그래프로 생성하여, 원시코드를 구조화시키고 각 문장별 구조화 정보와 skeleton 코드 및 영역 표시를 포함하여 모듈(프로시듀어)간 인터페이스 정보를 제공한다. 특히 병렬처리가 가능한 언어 형태의 분석 정보와 그래프 정보는 실용성과 효율성이 매우 높이 평가되고 있다.

참 고 문 헌

[1] 한국전자통신연구소, "CHILL Reference Manual", 1990. 10.

[2] 한국전자통신연구소, "CHILL 프로그래밍 가이드", 1990. 12.

[3] B. Beizer, Software Techniqes, Second Edition, VanNostrand Reinhold, New York, 1990.

[4] C. Berge, Graphs and Hypergraphs, North-Holland, New york, 1973.

[5] A. Bertolino and M. Marre, Hew many paths are needed for branch testing?, Vol. No. The journal System and Software, 1996.

[6] A. Bertolino and M. Marre, Automatic generation of path covers based on the control flow analysis of computer programs, IEEE Trans. On Software Engineering, 20(12):885-889, December

1994.

[7] A. Bertolino, R. Mirandola, and E. Peciola, A case study in branch testing automation, In Proc. of the 3rd Int. Conf. on Achieving Quality in Software(AQuIS96).

[8] P. G. Frankl and Weiss, An experimental comparison of the branch testing and data flow testing, IEEE Trans. On Software Engineering, 19 (8):774-787, August 1993.

[9] P. G. Frankl and E. J. Weyuker, An applicable family of data flow testing criteria, IEEE Trans on Software Engineering, 14(10):1483-1498, 1988.

[10] R. Gupta and M. L. Soffa, Employing static information in the generation of test cases, Software Testing, Verification and Reliablity, 3(1): 29-48, 1993.

[11] B. Beizer, Software System Testing and Quality Assurance, Van Nostrand Reinhold, New York, 1994.

[12] B. Beizer, Software Testing Techniques, Second Edition, Van Nostrand Reinhold, New York, 1990.

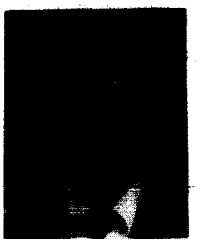
[13] E. Gansner, S. North, DAG-A program that draws directed graphs, software-practice and Experience, Vol. 18, No. 11, Nov. 1988.

[14] J. Ferrante, K. Oflenstein and J. Warren, The program dependence graph and its use in optimization, ACM Trans. Progr ...ning Languages and Systems, Vol. 9, No. 3, July. 1987.

[15] 노미나, 최병주, CASE: 소프트웨어 테스트 도구, 소프트웨어공학회지 제 9권 2호 p. 49-59, 1996년 6월호.

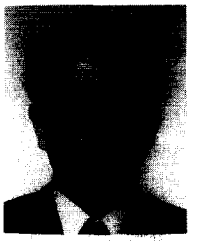
정 창 신

1983년 홍익대학교 전자계산학과 졸업(학사)
1987년 홍익대학교 대학원 전자계산학과(이학석사)
1984년~현재 한국전자통신연구소 선임연구원
관심분야: 소프트웨어 공학, 소프트웨어 품질보증 및 평가, 객체지향 개발 방법론.



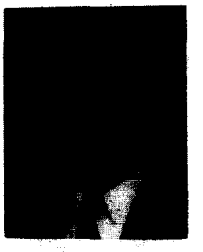
황 선 명

1982년 중앙대학교 전자계산학과 졸업(학사)
1984년 중앙대학교 대학원 전자계산학과 졸업(이학석사)
1987년 중앙대학교 대학원 전자계산학과 졸업(이학박사)
1988년 독일 Bonn대학 Informatik III post doctor
현재 대전대학교 컴퓨터공학과 부교수
관심분야: 품질보증, 소프트웨어 테스팅, 소프트웨어 재공학, 유지보수 방법 및 도구



이 경 환

1980년 중앙대학교 대학원 응용수학 전공(이학박사)
1982년~1983년 미국 Aurban대학 객원교수
1986년 서독 Bonn대학 객원교수
1971년~현재 중앙대학교 컴퓨터공학과 교수
관심분야: 소프트웨어 공학, 객체지향 모델링, 소프트웨어 재사용



김 행 곤

1985년 중앙대학교 전자계산학과 졸업(학사)
1987년 중앙대학교 대학원 전자계산학과 졸업(이학석사)
1991년 중앙대학교 대학원 전자계산학과 졸업(공학박사)
1978년~1979년 미 항공우주국 객원 연구원
1987년~1990년 한국전기통신공사 전임연구원
1988년~1989년 AT&T 객원 연구원
1990년~현재 대구효성가톨릭대학교 컴퓨터공학과 부교수
관심분야: 객체지향시스템 설계, 사용자 인터페이스, 소프트웨어 재공학, 유지보수 자동화툴, CASE