(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2010년01월28일
(11) 등록번호 10-0938489
(24) 등록일자 2010년01월15일

(51) Int. Cl.
G06F 9/45 (2006.01) *G06F 9/44* (2006.01)
(21) 출원번호 10-2007-0107829
(22) 출원일자 2007년10월25일
심사청구일자 2007년10월25일
(65) 공개번호 10-2009-0041996
(43) 공개일자 2009년04월29일
(56) 선행기술조사문헌
IEEE Tran.on Computer-aided disign of intergrated ciruits and systems,v.16, n.12, pp.1477-1487, 1997.12. Yau-Tsun S. Li, "Performance Analysis of Embedded Software Using Implicit Path Enumeration"
KR100575582 B1
US20060112377 A1

(73) 특허권자
김태효
서울 서초구 서초동 1506-62 송단62-502
방호정
서울 강남구 압구정동 433 현대아파트 126-801
(72) 발명자
김태효
서울 서초구 서초동 1506-62 송단62-502
방호정
서울 강남구 압구정동 433 현대아파트 126-801
차성덕
대전광역시 유성구 전민동 464-1 엑스포아파트 307동 608호
(74) 대리인
특허법인정직과특허

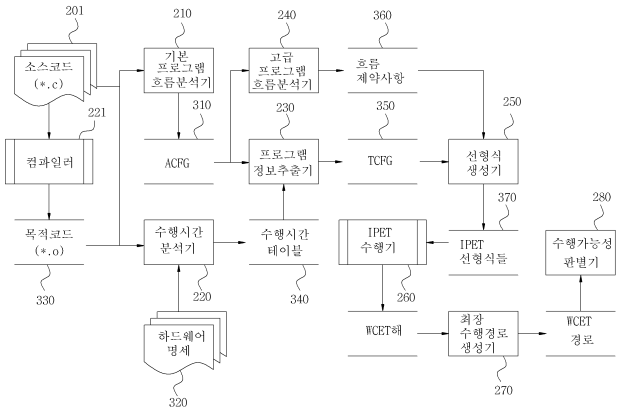
전체 청구항 수 : 총 6 항 심사관 : 황승희

(54) ＩＰＥＴ기법을 이용하여 실시간 시스템의 최장수행시간을분석하는 자동화된 도구 및 방법

(57) 요 약

본 발명은 IPET 기법을 이용하여 실시간 시스템의 최장수행시간을 분석하는 도구 및 방법으로서, 프로그램 언어로 작성된 소스코드를 정적으로 분석하여 프로그램 흐름정보를 추출하고 이를 토대로 수행 가능한 경로 및 최장수행시간을 자동으로 예측하는 최장수행시간 자동분석도구 방법 및 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체에 관한 것이다.

대 표 도 - 도1



특허청구의 범위

청구항 1

IPET 기법을 이용하여 실시간 시스템의 최장수행시간을 분석하는 도구로서, 프로그램 언어로 작성된 소스코드(201)를 정적으로 분석하여 프로그램 흐름정보를 추출하고 이를 토대로 수행 가능한 경로 및 최장수행시간을 자동으로 예측하는 최장수행시간 자동분석도구 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체에 있어서,

프로그램 언어로 작성된 소스코드(201)를 입력받아서 기본블록(311)들과 상기 기본블록(311)들 사이의 제어전이(transition or edge)(312)들을 파악하여, 상기 기본블록(311)과 상기 제어전이(312)을 플로우그래프 형태로 구성한 ACFG(Annotated Control Flow Graph)(310)를 생성하는 기본프로그램 흐름분석기(210)와;

하드웨어의 어셈블리 명령어별 수행시간정보를 명세한 하드웨어명세(320)를 사전에 작성하고, 상기 소스코드(201)를 입력받아 컴파일러(221)를 통해 상기 소스코드(201)와의 추적성을 유지하는 목적코드(330)(어셈블리 명령어로 작성된 코드)를 생성하고, 상기 ACFG(310)의 기본블록(311)들에 해당하는 목적코드(330)들을 찾아 상기 하드웨어명세(320)를 참조하여 상기 기본블록(311)당 수행시간을 계산하고, 그 결과값을 저장하는 수행시간테이블(340)을 생성하는 수행시간분석기(220)와;

상기 수행시간테이블(340)과 상기 ACFG(310)를 참조하여 기본블록당 수행시간과 기본블록사이의 의존성에 따른 수행시간 제약사항을 상기 ACFG(310)에 추가한 TCFG(Timed CFG)(350)를 생성하는 프로그램 정보추출기(230);

상기 ACFG(310)을 참조하여 각 기본블록에서의 변수들이 가질 수 있는 범위를 파악하여 수행불가능 경로들을 추출하고, 상기 수행불가능 경로들을 BDD(binary decision diagram)로 만들어 논리합을 함으로써 요약된 수행불가능 경로들을 추출하고, 상기 요약된 수행불가능 경로들로 구성된 흐름제약사항(360)을 생성하는 고급프로그램 흐름분석기(240);

상기 TCFG(Timed CFG)(350)로부터 기본블록의 수행횟수에 대한 선형식(flow facts)(이하 IPET 선형식)(370)들을 추출하고, 상기 흐름제약사항(360)에 저장된 수행불가능 경로들을 IPET 선형식(370)들로 자동으로 변환하는 선형식 생성기(250);

상기 선형식 생성기(250)에서 생성한 모든 IPET 선형식(370)을 제약조건으로 하고, 상기 TCFG(350)의 모든 기본블록에 대하여 각 기본블록의 수행횟수와 수행시간의 곱의 총합을 목적식으로 하여, 상기 제약조건을 만족하는 상기 목적식의 최대값이 되는 기본블록 수행횟수의 조합을 ILP(Integer linear Programming) 기법을 통해 구하는 IPET 수행기(260);

상기 IPET 수행기(260)로부터 구한 각 기본블록의 수행횟수를 이용하여 최장수행경로를 추출하는 최장수행경로 생성기(270);

를 포함하는 것을 특징으로 하는 최장수행시간 자동분석도구 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체.

청구항 2

제 1 항에 있어서, 상기 고급프로그램 흐름분석기(240)는,

상기 변수의 범위를 이용하여 추출된 수행불가능 경로들에 대하여, 상기 ACFG(310)에서 참거짓에 의해 분기하는 모든 기본블록을 단위명제(atomic proposition)들로 정하고,

상기 모든 수행불가능 경로들 각각에 대하여, 상기 단위명제로 정한 기본블록을 지나면 해당 단위명제가 참이고 지나지 않으면 거짓으로 하는 논리술어(predicate)로 만듦으로서, 각 수행불가능 경로를 BDD로 만드는 것;

을 특징으로 하는 최장수행시간 자동분석도구 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체.

청구항 3

제 1 항에 있어서, 상기 선형식 생성기(240)는,

상기 TCFG(350)에서 한 기본블록(311)의 수행횟수는 들어오는 제어전이(312)의 수행횟수의 합과 같고, 나가는 제어전이(312)의 수행횟수의 합과 같다는 것으로부터 구해지는 기본 IPET 선형식들을 생성하고;

상기 TCFG(350)에서 한 기본블록(311)이 자체적으로 수행횟수를 한정되는 경우 상기 수행횟수를 한정하는 한정 IPET 선형식들을 생성하고;

상기 수행불가능 경로들 각각에 대하여, 수행불가능 경로가 기본블록 b1에서 bn까지 차례로 수행되었을 경우 기본블록 c가 수행될 수 없는 경로이라고 하면,

기본블록 c에서 기본블록 bn이 도달 불가능하고, (b1,b2)(b2,b3) ... (bn-1,bn)들이 지배(dominance)관계가 성립할 경우,

$$\{ X_{bn} \le V(b_n)\times(1-X_c) \} \vee \{ X_c \le V(c)\times(1-X_{bn}) \}$$

(V(z)는 기본블록 z의 수행횟수의 한계, Xz 는 z의 수행횟수)

과 같은 형태로 변환하는 기능적 IPET 선형식들을 생성하는 것;

을 특징으로 하는 최장수행시간 자동분석도구 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체.

청구항 4

제 1 항에 있어서, 상기 최장수행시간 자동분석도구는,

상기 IPET 수행기(260)로부터 구한 각 기본블록의 수행횟수를 검증하는 체킹코드를 소스코드(201)에 추가삽입하여 모델체킹(model checking)기법으로 수행가능여부를 검사하고;

수행이 불가능하면 상기 체킹코드의 체킹사항을 IPET 선형식(370)으로 변환하여 상기 IPET 선형식들에 새롭게 추가하고;

상기 IPET 수행기(260)를 통해 다시 최장수행경로를 구하도록 하는;

수행가능성 판별기(280)를 더 포함하는 것을 특징으로 하는 최장수행시간 자동분석도구 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체.

청구항 5

제 4 항에 있어서, 상기 수행가능성 판별기(280)는,

일부 기본블록들의 수행횟수를 알면 나머지 모든 기본블록의 수행횟수를 알 수 있는 상기 일부 기본블록들의 집합인 핵심집합(dominant set)을 구하고;

상기의 핵심집합의 각 기본블록의 수행횟수에 대하여, 수행횟수를 카운트하는 변수를 삽입하고 상기 변수가 수행횟수에 도달하는지 확인하는 체킹코드를 삽입하여, 상기 모델체킹기법에 의해 수행불가능하다는 판단이 되면 상기 기본블록의 수행횟수가 상기 수행횟수보다 적다는 IPET 선형식을 새롭게 추가하고,

상기의 핵심집합의 기본블록들의 모든 짝들의 각각에 대하여, 하나의 기본블록의 수행횟수가 p 이고 또 다른 하나의 기본블록 수행횟수가 q 이면, 상기 짝의 기본블록들의 합이 p + q 와 같은지를 확인하는 체킹코드를 삽입하여, 상기 모델체킹기법에 의해 수행불가능하다는 판단이 되면 상기 기본블록들의 짝의 합이 p + q 보다 작다는 IPET 선형식을 추가하고;

기본블록들의 수행횟수에 관한 수행가능성 검사와 새로운 IPET 선형식의 추출하는 과정과 다시 최장수행경로를 구하는 과정을 더 이상 새로운 IPET 선형식을 추출되지 않거나 더 이상 계산이 실질적으로 불가능할 때까지 반복적으로 수행하는 것;

을 특징으로 하는 하는 최장수행시간 자동분석도구 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체.

청구항 6

IPET 기법을 이용하여 실시간 시스템의 최장수행시간을 분석하는 방법으로서, 프로그램언어로 작성된 소스코드(201)를 정적으로 분석하여 프로그램 흐름정보를 추출하고 이를 토대로 수행 가능한 경로 및 최장수행시간을 자동으로 예측하는 최장수행시간 자동분석방법에 있어서,

프로그램언어로 작성된 소스코드(201)를 입력받아서 기본블록(311)들과 상기 기본블록(311)들 사이의 제어전이(312)들을 파악하여, 상기 기본블록(311)과 상기 제어전이(312)을 플로우그래프 형태로 구성한 ACFG(Annotated

Control Flow Graph)(310)를 생성하는 기본프로그램 흐름분석단계(S510)와;

하드웨어의 어셈블리 명령어별 수행시간정보를 명세한 하드웨어명세(320)를 사전에 작성하고, 상기 소스코드(201)를 입력받아 컴파일러(221)를 통해 상기 소스코드(201)와의 추적성을 유지하는 목적코드(330)(어셈블리 명령어로 작성된 코드)를 생성하고, 상기 ACFG(310)의 기본블록(311)들에 해당하는 목적코드(330)들을 찾아 상기 하드웨어명세(320)를 참조하여 상기 기본블록(311)당 수행시간을 계산하고, 그 결과값을 저장하는 수행시간테이블(340)을 생성하는 수행시간분석단계(S520);

상기 수행시간테이블(340)과 상기 ACFG(310)를 참조하여 기본블록당 수행시간과 기본블록사이의 의존성에 따른 수행시간 제약사항을 상기 ACFG(310)에 추가한 TCFG(Timed CFG)(350)를 생성하는 프로그램 정보추출단계(S530);

상기 ACFG(310)을 참조하여 각 기본블록에서의 변수들이 가질 수 있는 범위를 파악하여 수행불가능 경로들을 추출하고, 상기 수행불가능 경로들을 BDD(binary decision diagram)로 만들어 논리합을 함으로써 요약된 수행불가능 경로들을 추출하고, 상기 요약된 수행불가능 경로들로 구성된 흐름제약사항(360)을 생성하는 고급프로그램 흐름분석단계(S540);

상기 TCFG(Timed CFG)(350)로부터 기본블록의 수행횟수에 대한 선형식(flow facts)(이하 IPET 선형식)(370)들을 추출하고, 상기 흐름제약사항(360)에 저장된 수행불가능 경로들을 IPET 선형식(370)들로 자동으로 변환하는 선형식 생성단계(S550);

선형식 생성기(250)에서 생성한 모든 IPET 선형식(370)을 제약조건으로 하고, 상기 TCFG(350)의 모든 기본블록에 대하여 각 기본블록의 수행횟수와 수행시간의 곱의 총합을 목적식으로 하여, 상기 제약조건을 만족하는 상기 목적식의 최대값이 되는 기본블록 수행횟수의 조합을 ILP(Integer linear Programming) 기법을 통해 구하는 IPET 수행단계(S560);

IPET 수행기(260)로부터 구한 각 기본블록의 수행횟수를 이용하여 최장수행경로를 추출하는 최장수행경로 생성단계(S570);

상기 최장수행경로의 각 기본블록의 수행횟수를 검증하는 체킹코드를 소스코드(201)에 추가삽입하여 모델체킹(model checking)기법으로 수행가능여부를 검사하고, 수행이 불가능하면 상기 체킹코드의 체킹사항을 IPET 선형식(370)으로 변환하여 상기 IPET 선형식들에 새롭게 추가하여, 상기 IPET 수행단계(S560)를 다시 반복하는 수행가능성 판별단계(S580);

를 포함하는 것을 특징으로 하는 최장수행시간 자동분석방법.

명 세 서

발명의 상세한 설명

기 술 분 야

<1> 본 발명은 프로그램언어로 작성된 소스코드를 정적으로 분석하여 프로그램 흐름정보를 추출하고 이를 토대로 상기 소스코드의 수행 가능한 경로 및 최장수행시간을 자동으로 예측하는 최장수행시간 자동분석도구 방법 및 프로그램을 기록한 컴퓨터로 읽을 수 있는 기록매체에 관한 것이다.

<2> 최장수행시간(Worst-Case Execution Time: WCET)은 실시간 시스템 분석의 기본 정보로 작업 스케줄 분석 등 다양한 분석에 사용된다. 기존의 최장수행시간 분석은 주로 측정에 기반한 수작업으로 이루어진 바, 시간과 비용이 많이 소요될 뿐 아니라 결과의 안전성을 보장하지 못하는 단점이 있었다. 이와 같은 단점을 보완하기 위하여 최근 정적 분석기법을 이용한 최장수행시간 분석 기법에 대한 연구들이 활발하게 진행되고 있다. 정적 최장수행시간 분석 기법들은 결과의 안전성을 보장할 뿐 아니라, 자동화가 가능하여 분석 시간과 노력을 크게 절감할 수 있는 장점이 있다. 그러나 종래의 정적 최장수행시간 분석기법들은 수행 불가능한 경로를 정적인 방법으로 모두 파악하는 것이 불가능하기 때문에 일반적으로 예측치를 과다 계상되는 경향이 있다.

<3> 본 발명은 상기와 같은 정적 최장수행시간 분석기법의 단점을 보완하여 실제 최장수행시간에 근접하는 예측치를 계산하는 기법을 개발하고, 더 나아가 단순한 기법개발만이 아니라, 이를 통해 소스코드만 입력하면 자동으로 수행가능경로와 최장수행시간을 추출할 수 있는 전체 시스템 및 방법을 제시하고 있다.

배 경 기 술

<4> 최근 내장형 시스템의 활용이 폭발적으로 증가하고 있다. 이는 시스템 기능이 복잡해지고 변화에 대한 높은 적응력이 요구됨에 따라, 하드웨어 중심의 기존 시스템들이 점차 소프트웨어에 기반한 내장형 시스템으로 옮겨가고 있기 때문이다. 내장형 시스템은 통상 실시간 시스템의 특성을 지닌다. 실시간 시스템이란 어떤 기능을 수행함에 있어 수행 결과의 정확성뿐 아니라, 시간적 요구사항의 만족 여부도 중요한 시스템이다. 따라서 내장형 시스템에서는 기능적 요구사항에 대한 검증뿐 아니라, 비기능적 요구사항 특히 시간제약에 대한 검증이 필수적이다.

<5> 시간제약의 오류는 디버깅이나 테스팅을 통하여 발견하기 매우 어려운 특성을 가진다. 왜냐하면 오류 상황을 발견하더라도 이를 재연하기가 힘들고 그 원인을 파악하기 힘들기 때문이다. 따라서 시간제약 오류는 개발 후기 또는 개발 완료 후에 발견되는 경우가 많으며, 이를 수정하는데 많은 비용이 소요됨은 물론 제품의 신뢰도를 하락시키는 주요 원인이 된다.

<6> 또한 시간적 요구사항의 분석은 내장형 시스템의 성능을 높이고 한정된 자원을 최대한 활용하기 위해서도 필수적이다. 내장형 시스템의 경우 매우 제한된 환경에서 동작하기 때문에 구동 소프트웨어가 최적화되어야 한다. 이 최적화 과정을 보다 효율적으로 엄밀히 하기 위해서는 소프트웨어 수행시간에 대한 분석이 이루어져야 한다.

<7> 최장수행시간이란 특정 작업 혹은 프로그램이 선점이 없는 상태에서 그 수행에 소요되는 시간의 최댓값으로 시간제약 관련 분석의 기본 정보이다. 따라서 최장수행시간을 예측함에 있어서 다음과 같은 특성이 지켜져야 한다. i) 예측 값은 실제 최장수행시간보다 작아서는 안되고(안전성), ii) 가능한 한 실제 값에 가까워야한다(정확성). 최장 수행시간 분석의 목표는 안전한 최장수행시간 예측치를 가능한 정확하게 구하는 것이다.

<8> 최장수행시간 분석 기법은 크게 측정 기반의 분석과 정적 분석으로 나눌 수 있다. 측정 기반의 분석은 실제 입력 데이터를 구하고 이를 실제로 수행하여 측정된 소요 시간의 최댓값을 구하는 반면, 정적 분석은 프로그램 소스 코드 자체를 정적으로 분석함으로써 최장수행시간을 예측한다. 측정 기반의 분석이 시간과 비용이 많이 소요되고 결과의 안전성을 보장할 수 없는 단점이 있는 반면, 정적 분석은 항상 안전한 결과를 산출하나 정확도가 떨어지는 단점이 있다.

<9> 측정기반의 분석기법에서는 개발자가 자신의 지식과 경험에 근거해 비교적 시간이 많이 소요될 것으로 예상되는 후보 경로들을 선택하고(경로 추출), 각각의 경로에 대한 입력 값을 준비한다(테스트 데이터 생성). 이러한 입력들을 해당 프로그램에 시뮬레이션 혹은 대상 하드웨어에서 직접 수행하여 그 수행시간을 측정한다(실행 및 측정). 측정된 수행시간들 중 최댓값을 예측치로 산정한다(최장수행시간 추출).

<10> 하지만 프로그램의 제어구조가 복잡한 경우에는 후보경로를 선정하는 것과 그 경로에 대한 입력 값을 추출하는 것은 매우 많은 노력과 시간이 소요되고 그 과정에서 오류 가망성이 존재하게 된다. 일례로 항공우주연구원에서 개발된 다목적 실용위성 아리랑 2호의 명령어 처리모듈의 경우, 측정기반의 기법으로 그 최장수행시간을 분석한 바 있으며, 이를 완료하는데 약 6명이 1개월의 기간(6 man-month)의 인력비용이 소요되었다.

<11> 또한 많은 경우 모든 경로를 고려할 수 없기 때문에 측정기반의 기법으로 도출한 최장수행시간의 예측치의 경우 그 안전성을 보장할 수 없다. 안전성을 보완하기 위하여 분석결과에 일정비율의 시간을 덧붙여 최장수행시간 예측 값으로 사용하기도 하지만, 안전성 문제를 완전히 해결하지는 못한다.

<12> 정적기반의 분석기법들은 프로그램을 실행하는 대신 소스 혹은 목적코드(objective code)를 정적으로 분석함으로써 최장수행시간을 예측한다. 이러한 기법들은 보통 프로그램 경로를 분석하여 실현가능한 경로들을 파악하고(프로그램 경로 분석), 각 기본블록(basic block)의 수행시간을 캐시, 파이프라인 등 하드웨어 특성을 고려하여 계산한 후(하위수준 분석), 이러한 정보를 이용해 최장수행시간 예측치를 계산하게 된다(계산). 대표적인 정적 최장수행시간 계산방법에는 트리기반기법(tree-based technique), 경로기반기법(path-based technique), IPET(Implicit Path Enumeration Technique) 기법 등이 있다.

<13> 상기 트리기반기법과 관련하여, [C. Y. Park and A. C. Shaw. Experiments with a program timing tool based on a source-level timing schema. In 11th IEEE Real-Time Systems Symposium (RTSS''90), 1990.]와 [Gustav Pospischil, Peter Puschner, Alexander Vrchoticky, and Ralph Zainlinger. Developing real-time tasks with predictable timing. IEEE Software, 9(5), 1992.]에 보다 자세히 제시하고 있다. 상기 경로기반기법(path-based technique)과 관련하여, [C. Healy, R. Arnold, F. Muller, D. Whalley, and M. Harmon. Bounding pipeline and instruction cache performance. IEEE Transactions on Computers, 48(1), 1999.]에서 보다 자세히 기재되어 있다. IPET 기법과 관련하여, [Yau-Tsun Steven Li and Sharad Malik. Performance analysis of embedded software using implicit path enumeration. In Proceedings of the 32nd ACM/IEEE

Design Automation Conference, 1995.]와, [P. Puschner and A. Schedl. Computing maximum task execution times with linear programming techniques. Technical report, Technische University at Wien, Institut fur Technische Informatik, 1995.]에 보다 구체적이고 자세하게 기재되어 있다.

<14> 정적 최장수행시간 분석의 장점은 (1) 자동화가 가능하여 시간과 비용을 절감할 수 있을 뿐 아니라 수작업에서 오는 오류를 배제할 수 있으며, (2) 개발자 및 분석자가 반복 작업에서 탈피하여 실제 개발에만 집중할 수 있게 하며, (3) 하드웨어 완성 전에도 검증이 가능하므로 하드웨어와 소프트웨어를 동시에 개발할 수 있는 점이다. 또한 (4) 분석 시 프로그램을 실제 수행할 필요가 없으므로 실행에 소요되는 시간을 절감할 수 있다.

<15> 그러나 정적분석기법은 수행 불가능한 경로를 정적인 방법으로 모두 파악하는 것이 불가능하기 때문에 일반적으로 예측치를 과다 계상하는 경향이 있다. 정적분석에 있어 결과의 정확도는 추출되는 프로그램 흐름 정보의 양과 직접적인 관계가 있으므로 가능한 많은 흐름정보를 추출해야 하지만, 그럴수록 더 많은 계산비용이 소모되게 된다. 따라서 정적분석의 경우 이러한 정확성과 비용사이의 트레이드오프(trade-off) 관계를 어떻게 조화시키느냐가 관건이 된다.

발명의 내용

해결 하고자하는 과제

<16> 상기와 같이 정적 최장수행시간 분석기법의 단점을 해결하기 위하여, 본 발명은 모든 수행불가능 경로를 자동으로 추출하고, 특히 계산이 가능하도록 요약된 수행불가능 경로만 재차 추출하여, 수행불가능 경로를 제외한 최장수행시간 분석기법을 제공함을 목적으로 한다.

<17> 또한, 단순히 최장수행시간 및 경로를 한 번에 계산하는 것에 머물지 않고, 결과값을 다시 검증하는 반복적 단계를 거치게 함으로써 실제 최장수행시간에 근접하도록 하는 기술을 제공함을 목적으로 한다.

<18> 또한, 정적 최장수행시간 분석기법을 적용하여 소스코드만을 입력하면 자동으로 최종 최장수행시간과 경로를 구할 수 있도록 전체과정을 자동화하는 것을 목적으로 한다.

<19> 특히, 본 발명에서 사용하고자 하는 IPET 분석기법이 자동화되지 못하는 가장 큰 문제점인 불가능 수행경로로부터 IPET 선형식(Flow facts)들을 자동으로 추출하지 못하는 문제점을 해결함으로써, IPET 분석기법을 이용한 자동화된 정적 최장수행시간 분석장치 및 방법을 제시하고자 한다.

<20> 또한, 본발명의 장치 및 방법이 특정한 프로그램 언어와 하드웨어에 의존적이지 않고 보다 확장성을 손쉽게 할 수 있는 기술을 제시하고자 한다.

과제 해결수단

<21> 상기 목적을 달성하기 위한 본 발명은 IPET 기법을 이용하여 실시간 시스템의 최장수행시간을 분석하는 도구 및 방법으로서, 프로그램 언어로 작성된 소스코드를 정적으로 분석하여 프로그램 흐름정보를 추출하고 이를 토대로 수행 가능한 경로 및 최장수행시간을 자동으로 예측하는 최장수행시간 자동분석 도구 및 방법에 관한 것이다.

<22> 보다 상세하게는, 본 발명은 프로그램 언어로 작성된 소스코드를 입력받아서 기본블록들과 상기 기본블록들 사이의 제어흐름들을 파악하여, 상기 기본블록과 상기 제어흐름을 플로우그래프 형태로 구성한 ACFG(Annotated Control Flow Graph)를 생성하는 기본프로그램 흐름분석기와;

<23> 하드웨어의 어셈블리 명령어별 수행시간정보를 명세한 하드웨어명세를 사전에 작성하고, 상기 소스코드를 입력받아 컴파일러를 통해 상기 소스코드와의 추적성을 유지하는 목적코드(어셈블리 명령어로 작성된 코드)를 생성하고, 상기 ACFG의 기본블록들에 해당하는 목적코드들을 찾아 상기 하드웨어명세를 참조하여 상기 기본블록당 수행시간을 계산하고, 그 결과값을 저장하는 수행시간테이블을 생성하는 수행시간분석기와;

<24> 상기 수행시간테이블과 상기 ACFG를 참조하여 기본블록당 수행시간과 기본블록사이의 의존성에 따른 수행시간 제약사항을 상기 ACFG에 추가한 TCFG(Timed CFG)를 생성하는 프로그램 정보추출기;

<25> 상기 ACFG을 참조하여 수치도메인(numerical domain) 상에서의 요약해석(abstract interpretation)을 이용한 추출방법을 통해 수행불가능 경로들을 추출하고, 상기 수행불가능 경로를 BDD(binary decision diagram)를 이용한 방법을 통해 요약된 수행불가능 경로들을 추출하고, 상기 요약된 수행불가능 경로들로 구성된 흐름제약사항을 생성하는 고급프로그램 흐름분석기;

<26> 상기 TCFG(Timed CFG)로부터 기본블록의 수행횟수에 대한 선형식(flow facts)(이하 IPET 선형식)들을 추출하고, 상기 흐름제약사항에 저장된 수행불가능 경로들을 IPET 선형식들로 자동으로 변환하는 선형식 생성기;

<27> 상기 선형식 생성기에서 생성한 모든 IPET 선형식을 제약조건으로 하고, 상기 TCFG의 모든 기본블록에 대하여 각 기본블록의 수행횟수와 수행시간의 곱의 총합을 목적식으로 하여, 상기 제약조건을 만족하는 상기 목적식의 최대값이 되는 기본블록 수행횟수의 조합을 ILP(Integer linear Programming) 기법을 통해 구하는 IPET 수행기;

<28> 상기 IPET 수행기로부터 구한 각 기본블록의 수행횟수를 이용하여 최장수행경로를 추출하는 최장수행경로 생성기;

<29> 를 포함하는 것을 특징으로 하는 최장수행시간 자동분석도구에 관한 것이다.

효 과

<30> 본 발명은 모든 수행불가능 경로를 자동으로 추출하고, 계산이 가능하도록 요약된 수행불가능 경로만 재차 추출하여, 수행불가능 경로를 제외한 최장수행시간을 계산할 수 있도록 하고 있다. 더 나아가, 단순히 최장수행시간 및 경로를 한 번에 계산하는 것에 머물지 않고, 결과값을 다시 검증하는 반복적 단계를 거치게 함으로써 실제 최장수행시간에 근접하도록 하고 있다.

<31> 도 9는 본 발명이 다목적 실용위성 명령어 처리모듈에 대해 최장수행분석을 한 결과를 도시한 것이다. 해당 위성의 명령어 처리모듈의 경우 C 프로그램으로 구현되어 있으며, 모두 17개의 주요 함수로 구성되어 있고, 총 크기는 약 4,200줄 정도이다. 이 모듈은 항공우주연구원에서 위성을 개발할 당시에 측정기반의 최장수행시간 분석을 수행한 바 있어 그때의 결과값을 비교값으로 사용하였다. 도 9에 도시된 표는 측정기반의 분석결과값과 본 발명으로 예측한 결과값을 비교한 것이다. 해당 CPU 사이클은 인텔386 임베디드 프로세스 상에서의 수행사이클을 의미한다. 전체적으로 본 발명으로 예측한 값이 최장수행시간을 122% 과다 계상하고 있는 것을 볼 수 있다. 이와 같은 결과는 위성 소프트웨어의 경우 안정성을 위하여 측정치에 추가적인 비율만큼의 시간을 더하여 최장수행시간의 예측치로 사용하는 것을 고려할 때 큰 자원의 낭비없이 사용할 수 있는 예측치라 할 수 있다. 실제 측정기반 예측을 위하여 약 6명이 1개월의 기간(6 man-month)의 인력비용이 소요된 것에 비하여 본 발명은 인텔 코어 2 듀오 2.0 GHz CPU 환경에서 5분 이내에 분석을 완료할 수 있었다.

<32> 또한, 정적 최장수행시간 분석기법을 적용하여 소스코드만을 입력하면 자동으로 최종 최장수행시간과 경로를 구할 수 있도록 전체과정을 자동화하고 있다.

<33> 특히, 본 발명에서 사용하고자 하는 IPET 분석기법이 자동화되지 못하는 가장 큰 문제점인 불가능 수행경로로부터 IPET 선형식(Flow facts)들을 자동으로 추출하지 못하는 문제점을 해결하였고, 그로부터 IPET 분석기법을 이용한 자동화된 정적 최장수행시간 분석장치 및 방법을 제시하고 있다.

<34> 또한, 본발명의 장치 및 방법이 특정한 프로그램 언어와 하드웨어에 의존적이지 않고 보다 확장성을 손쉽게 할 수 있는 장치 및 방법을 제시하고 있다.

발명의 실시를 위한 구체적인 내용

<35> 본 발명은 IPET 기법을 이용하여 실시간 시스템의 최장수행시간을 분석하는 도구 및 방법으로서, 프로그램 언어로 작성된 소스코드를 정적으로 분석하여 프로그램 흐름정보를 추출하고 이를 토대로 수행 가능한 경로 및 최장수행시간을 자동으로 예측하는 최장수행시간 자동분석 도구 및 방법에 관한 것이다.

<36> 이하, IPET 기법을 이용하여 실시간 시스템의 최장수행시간을 자동으로 예측하는 최장수행시간 자동분석 도구 및 방법을 참조도면을 통하여 상세히 설명하고자 한다.

<37> 도 1은 본 발명의 최장수행시간 자동분석도구의 구성에 대한 바람직한 일실시예이다.

<38> 도 1에 보는 바와 같이, 최장수행시간 자동분석도구는 기본프로그램 흐름분석기(210)와, 수행시간분석기(220), 프로그램 정보추출기(230), 고급프로그램 흐름분석기(240), 선형식 생성기(250), IPET 수행기(260), 최장수행경로 생성기(270), 수행가능성 판별기(280)를 포함한다.

<39> 상기 기본프로그램 흐름분석기(210)는 프로그램 언어로 작성된 소스코드(201)를 입력받아서 기본블록(311)들과 상기 기본블록(311)들 사이의 제어전이(transition or edge)(312)들을 파악하여, 상기 기본블록(311)과 상기 제

어전이(312)을 플로우그래프 형태로 구성한 ACFG(Annotated Control Flow Graph)(310)를 생성한다.

<40> 상기 기본프로그램 흐름분석기(210)는 프로그램의 소스 코드를 그 입력으로 받는다. 일반적인 내장형 시스템의 소스 코드는 다양한 언어로 작성될 수 있다. 하지만 현재 대다수의 내장형 시스템은 C 언어를 사용하여 구현되는 실정이기 때문에 본 발명은 프로그램을 작성하기 위한 언어로서 C 언어를 대상으로 바람직한 실시예를 구성하였다. 그러나 C 언어이외의 프로그램을 작성하기 위한 언어에도 확장하여 적용이 가능하다. 도 2는 본 발명의 수행과정을 예시하기 위하여 본 발명에 입력되는 C언어로 작성된 소스코드를 예시한 것이다.

<41> 상기 기본프로그램 흐름분석기(210)의 결과물로 생성되는 ACFG(310)에 대하여, 도 3을 참조하여 보다 상세히 설명하고자 한다. 도 3은 도 2에 예시된 소스코드의 기본블록과 제어전이을 분석하여 생성되는 ACFG(310)를 예시한 것이다. 일반적으로 프로그램의 흐름을 제어 및 데이터 흐름 분석의 결과물로 생성하는 것이 CFG(Control flow graph)로 표현된다. 본 발명의 ACFG(310)는 소스코드(201)로부터 프로그램 분석의 일반적인 방법에 의해 CFG를 만들고 이 CFG 상의 노드와 에지(edge)에 추가적인 정보를 첨부한 것을 명명한 것이다.

<42> 상기 ACFG(310)는 기본블록(311)과 제어전이(312)로 구성된다. 기본블록(311)은 CFG 상의 노드에 해당하는 것으로서, 프로그램에서 무조건 같이 수행되는 명령구문의 집합이고, 제어전이(312)은 CFG 상의 에지(edge)에 해당하는 것으로서, 프로그램의 흐름을 나타내는 방향을 나타낸다. 도 3을 참조하면, 동그라미나 네모 등이 기본블록(311)에 해당하고 화살표가 제어전이(312)에 해당한다.

<43> 기본블록(311)은 프로그램에서 무조건 같이 수행되는 명령구문의 집합이라는 점에서 일반적인 CFG 상의 에지와는 약간의 차이가 있다. 즉, 도 3에서 기본블록 BB2나 기본블록 BB8은 각각 2개의 명령구문으로 구성되어 있다. 즉, 2개의 명령구문은 반드시 같이 수행되는 명령구문이기 때문이다. 기본블록(311)에 추가되는 정보는 기본블록의 ID, 소스코드(201)상의 기본블록이 시작라인과 끝라인에 대한 정보, 종류 등이 있다. 기본블록(311)의 종류는 실제 프로그램이 수행되는 것이 아니라 프로그램의 흐름을 표현하기 위한 가상노드와 실제 프로그램이 수행되는 기본노드로 구분된다. 가상노드는 엔트리(entry), 엑시트(exit), 헤더(Header) 등이 있으며 실제 프로그램이 수행되는 것이 아니므로 수행시간은 0이다. 도 3에서는 BB1, BB5, BB6, BB9, BB13이 가상노드에 해당된다. 기본노드는 분기문(Branch), 실제 명령어가 수행되는 연산블록, 함수호출 등이 있으며, 실제 프로그램이 수행되어 수행시간은 0 보다 크게 된다. 도 3에서는 BB2, BB3, BB4, BB7, BB8, BB10, BB11, BB12이 기본노드에 해당된다. 특히, 분기문(Branch)은 판단에 의해 참과 거짓에 따라 2개의 제어전이(312)이 나온다. 도 3에서는 BB3과 BB7, BB10이 이에 해당한다.

<44> 제어전이(312)은 출발 노드에서 도착 노드로 프로그램이 진행되는 흐름을 나타내는 것으로, 출발 노드의 조건판단의 결과, 즉, 참거짓에 따라 도착노드가 2개로 나뉘는 조건부 제어전이(312)과 출발노드에서 무조건 도착 노드로 진행하는 무조건부 제어전이(312)으로 구분된다. 특히, 조건부 제어전이(312)은 출발노드인 기본블록(311)의 종류가 분기문(branch)인 경우에 만들어진다.

<45> 상기 ACFG(310)는 그래프를 나타내는 표준 포맷들 중 하나인 GraphML ("The GraphML File Format." http://graphml.graphdrawing.org/.) 로 생성한다.

<46> 상기 수행시간분석기(220)는 하드웨어의 어셈블리 명령어별 수행시간정보를 명세한 하드웨어명세(320)를 사전에 작성하고, 상기 소스코드(201)를 입력받아 컴파일러(221)를 통해 상기 소스코드(201)와의 추적성을 유지하는 목적코드(330)(어셈블리 명령어로 작성된 코드)를 생성하고, 상기 ACFG(310)의 기본블록(311)들에 해당하는 목적코드(330)들을 찾아 상기 하드웨어명세(320)를 참조하여 상기 기본블록(311)당 수행시간을 계산하고, 그 결과값을 저장하는 수행시간테이블(340)을 생성한다.

<47> 상기 하드웨어명세(320)는 타깃 하드웨어의 수행시간정보를 명세한 것이다. 도 4에 보는 바와 같이, 하드웨어명세(320)는 하드웨어의 실행되는 어셈블리어(또는 기계어)의 각 명령어에 대한 수행시간정보를 명세한 것이다. 즉, 도 4의 하드웨어명세(320)의 첫 번째 줄은 명령어의 번호이고, MNEMONIC 은 어셈블리의 명령어 이름이며, DESCRIPTION 은 명령어의 간단한 설명이고, OPERAND는 인자의 형식, BITPATTERN은 이진코드 비트패턴으로서 어셈블리 명령어(또는 기계어)의 이진코드를 나타내고 소스코드(201)를 컴파일한 목적코드(330)에서 상기 BITPATTERN과 일치되면 해당 명령어를 파악한다.

<48> 추적성을 유지하는 목적코드(330)는 소스코드(201)를 디버깅 옵션과 함께 컴파일하여 만들어진다. 즉, 생성된 목적코드(330)의 기계어들에 상응하는 소스코드(201) 상의 줄번호 정보를 가지게 되고, 이를 통해 소스코드(201)의 추적성을 유지한다. 도 5는 입력된 도 2에 예시된 소스코드를 소스코드와 추적성을 유지하도록 컴파일한 목적코드를 예시하고 있다. 도 5에서 2번째 줄의 < c:₩target₩...₩exam1.c:4 >는 기계어들에 상응하는 소스

코드상의 줄번호를 나타낸다. 즉, <파일이름>:<줄번호> 형식이다. 즉, 3번째에서 5번째 줄의 기계어들은 소스코드의 4번째 줄을 컴파일한 기계어임을 알 수 있다. 3번째에서 5번째 줄의 < 0: >, < 1: >, < 3: > 은 기계어 코드의 주소가 된다. < 55 >, < 89 e5 >, < 83 ec 08 > 은 이진코드로 된 기계어코드이며, < push %epb, mov %esp, %esp, ... > 는 어셈블리 코드에 해당된다.

<49> 따라서 상기 ACFG(310)의 기본블록(311)들에는 해당 소스코드(201)의 시작라인과 끝라인을 알 수 있고, 상기 목적코드(330)가 상기와 같이 소스코드 추적성이 유지되므로, 목적코드에서 상기 기본블록에 해당하는 기계어코드를 찾을 수 있다. 기계어코드를 찾으면 하드웨어명세(320)를 참조하여 각 기계어코드의 수행시간을 알아 낼 수 있다.

<50> 예를 들어 도 5의 예제 중 기계어 코드(16진수)를 이진수로 나타내면 다음과 같다.

<51>

0: 55 push %ebp	0: 0101 0101 push %ebp

<52> 이는 도 5에서 PUSH 명령어에 해당하는 비트 패턴 (0 1 0 1 0 r r r)과 일치 되므로 PUSH 명령어 임을 파악할 수 있고, 그 수행시간은 도 4의 수행시간 정보를 참조하여 리얼모드에서 2 사이클이 소요되는 것으로 계산한다.

<53> 상기와 같은 방법으로, 상기 ACFG(310)의 모든 기본블록(311)들에 대하여, 기본블록(311)당 수행시간을 계산하고, 그 결과값을 저장하는 수행시간테이블(340)을 생성한다. 상기 수행시간테이블(340)은 기본블록의 이름과 수행시간으로 구성될 수 있다.

<54> 본 발명의 하드웨어명세(320)는 하나의 타깃 하드웨어만을 대상으로 하지 않는다. 즉, 다양한 타킷 하드웨어에 대하여 하드웨어명세(320)를 각각 작성하고 소스코드(201)가 어느 타킷 하드웨어에서 수행될 것인가를 선택할 수 있도록 할 수 있다. 즉, 대상 하드웨어에 대한 정보를 사용자가 쉽게 입력하기 위하여 본 발명은 현존하는 하드웨어에 대한 이미 작성된 명세들인 하드웨어 명세 컬렉션을 제공한다. 따라서 사용자는 제공되는 하드웨어의 명세들 중 원하는 하드웨어를 선택만 함으로써 수행시간을 측정할 수 있다. 그리고 하나의 소프트웨어의 수행시간이 다양한 하드웨어 상에서 어떻게 소요되는 지를 하드웨어 명세 변경만을 통하여서 예측이 가능하다. 이와 같은 기능은 새로운 시스템에 필요한 성능을 발휘할 수 있는 하드웨어를 선정하는데 컬렉션의 개발을 더 용이하게 생성하기 위하여, 본 발명은 내부적으로 Microsoft Excel의 비주얼 베이직 스크립트를 활용한다. 이 엑셀시트에는 해당 기계어를 파싱하기 위한 정보와 해당 기계어가 수행되는 데 걸리는 시간 등을 채우기 위한 테이블들이 존재하고, 이 정보는 비주얼 베이직 스크립트를 통하여 상기 하드웨어명세(320)에 입력된다. 따라서 새로운 대상하드웨어를 지원하기 위해 매번 수행시간분석기(220)를 전체적으로 재작성할 필요가 없기 때문에 새로운 하드웨어 명세 작성에 큰 이점을 가진다.

<55> 상기 기본프로그램 흐름분석기(210)와 수행시간분석기(220)는 프로그램 소스코드(201)를 입력 받아 컴파일러 등을 이용하여 우선적으로 입력 구문(syntax)을 정의하고 이를 파싱(parsing)하는 과정을 거친다. 이는 프로그램의 구조 및 의미를 파악하는 첫 단계로써 모든 정적 분석기 및 컴파일러에 필수적인 과정이다. 이 구문 분석 과정이 끝나면 프로그램은 정의된 내부 자료구조를 이용하여 표현되고, 내부적으로는 이 내부 자료구조를 이용하게 된다.

<56> 하지만 C언어로 작정된 프로그램을 구문 분석하는 것에 많은 어려움이 존재한다. C언어의 경우 이미 ANSI-C나 POSIX-C 등과 같은 표준 구문이 존재한다. 이와 같은 표준화 작업에도 불구하고 대다수의 시스템 개발에서는 특정 컴파일러에서 확장한 비표준 구문들을 사용하여 구현이 이루어지고 있는 실정이다. 일례로써, GNU gcc, Microsoft C, 와 Borland C 등의 경우 서로 호환되지 않는 구문이 경우가 다수 존재한다. 그리고 동일한 컴파일러 계열을 사용한다고 할지라도 버전에 따라서 서로 호환되지 않는 경우가 발생한다. 일례로 GNU gcc 2.xx, 3.xx, 4.xx 들은 모두 호환되지 않는 구문을 다수 포함한다.

<57> 따라서 C언어를 지원하기 위해서는 사용하는 컴파일러 종류와 버전에 따라서 다른 구문 분석 방법을 사용하여야 한다. 이와 같은 C 언어의 다양한 변종들을 입력 받기 위하여 컴파일러 지원 모듈을 플러그인(plug-in) 형식으로 제공할 수 있다. 컴파일러 지원 모듈은 입력 받은 C 프로그램의 기본적인 구문 분석을 수행하고, 해당 컴파일러를 이용하여 목적코드(object code)를 생성하는 역할을 한다.

<58> 본 발명은 상기 기본프로그램 흐름분석기(210)와 수행시간분석기(220)에 의해 이용되는 컴파일러 지원을 모듈화함으로써 새로운 C언어의 변종을 지원하기 위하여 소스코드 분석기를 전체적으로 재작성할 필요가 없으며, 이미 지원 중인 분석기를 수정할 필요가 없다.

<59> 상기 프로그램 정보추출기(230)는 상기 수행시간테이블(340)과 상기 ACFG(310)를 참조하여 기본블록당 수행시간과 기본블록사이의 의존성에 따른 수행시간 제약사항을 상기 ACFG(310)에 추가한 TCFG(Timed Control Flow Graph)(350)를 생성한다.

<60> 상기 기본블록당 수행시간은 상기 수행시간테이블(340)에 저장된 각 기본블록에 해당하는 수행시간을 말한다.

<61> 상기 기본블록사이의 의존성에 따른 수행시간 제약사항은, 분기문(branch)의 경우 참(true)으로 진행할 때와 거짓(false)로 진행할 때 수행시간이 달라지는 경우와, 변수를 읽어올 때 메모리에서 직접 가져오는 시간과 캐시에 저장된 값을 가져오는 시간간의 차이가 나기 때문에 고려되는 사항이다. 예를 들어, 변수 a를 사용하는 기본블록이 A와 B가 있고 C는 사용하지 않는다고 하면, A -> B 로 왔을 때, 기본블록 A를 수행하면서 변수 a를 메모리에서 읽어 캐시(cache_에 넣어 두기 때문에 기본블록 B를 수행할 때 변수 a를 메모리에서 읽을 필요가 없게 된다. 그러나 C -> B로 왔을 때는 변수 a를 처음 사용하므로 메모리에서 읽어야 한다. 따라서 A -> B 로 왔을 때 B의 수행시간이 C -> B 로 왔을 때 B의 수행시간보다 짧아진다. 이와 같은 것을 고려하기 위한 제약 사항들을 말한다.

<62> TCFG(350)는 프로그램 분석의 결과물들을 모두 포함하고 있기 때문에, 이후 모든 과정에서는 TCFG만을 참조하여 작업들을 수행한다.

<63> 상기 고급프로그램 흐름분석기(240)는 상기 ACFG(310)을 참조하여 각 기본블록에서의 변수들이 가질 수 있는 범위를 파악하여 수행불가능 경로들을 추출하고, 상기 수행불가능 경로들을 BDD(binary decision diagram)로 만들어 논리합을 함으로써 요약된 수행불가능 경로들을 추출하고, 상기 요약된 수행불가능 경로들로 구성된 흐름제약사항(360)을 생성한다.

<64> 더 자세하게는, 상기 고급프로그램 흐름분석기(240)는 상기 변수의 범위를 이용하여 추출된 수행불가능 경로들에 대하여, 상기 ACFG(310)에서 참거짓에 의해 분기하는 모든 기본블록을 단위명제(atomic proposition)들로 정하고, 상기 모든 수행불가능 경로들 각각에 대하여, 상기 단위명제로 정한 기본블록을 지나면 해당 단위명제가 참이고 지나지 않으면 거짓으로 하는 논리술어(predicate)로 만듦으로서, 각 수행불가능 경로를 BDD로 만드는 것을 특징으로 한다.

<65> 각 기본블록에서의 변수들이 가질 수 있는 범위를 파악하기 위해서는 수치도메인상(numerical domain)에서의 요약해석(abstract interpretation) 방법을 이용한다. 상기 방법은 프로그램의 각각의 위치에서 변수들이 가질 수 있는 값의 범위를 파악하는 방법으로서, 변수의 구간범위(interval range)를 그 결과물로 생성한다. 상기 수치도메인상에서의 요약해석에 대한 기술은 논문 [Jan Gustafsson and Andreas Ermedahl, "Automatic derivation of path and loop annotations in object-oriented real-time programs," Journal of Parallel and Distributed Computing Practices, vol. 1 no. 2, June 1998]에 자세히 기술되고 있다.

<66> 상기의 방법으로 구한 변수들의 구간범위로부터 실행 불가능한 경로를 추출할 수 있다. 예를 들면, 도 3의 ACFG(310)에서 b의 값이 항상 양수이기 때문에 기본블록 BB11로는 도달할 수 없다. 따라서 BB1→BB2→BB3→BB4→BB5→BB6→BB7→BB9→BB10→BB11과 같이 BB11로 들어오는 모든 경로들은 수행 불가능한 경로들이 된다.

<67> 하지만, 위와 같이 모든 수행 불가능한 경로를 나열하는 것은 비용적으로나 성능적으로나 가능한 것이 아니기 때문에 수행 불가능한 경로들을 요약할 수 있는 방법이 필요하다. 이를 위하여 우선적으로 모든 분기문(branch)의 기본블록에 고유의 단위명제(atomic proposition)를 부여하고 수행 불가능한 경로들을 상기 논리명제들의 논리술어(predicate)의 형태로 경로들을 모두 표현하여 BDD(Binary Decision Diagram)를 만든다.

<68> 예를 들어 BB3, BB5, BB10의 분기하는 기본블록이므로 단위명제로서 각각 P, Q, R 이라 정할 수 있다. 한 수행경로는 이들이 참 혹은 거짓 방향으로 진행한지로 표현할 수 있다. 즉, 수행 불가능한 경로 BB1→ BB2→ BB3→ BB4→ BB5→ BB6→ BB7→ BB9→ BB10→ BB11는 Q만이 거짓방향으로 진행한 것이므로 P∧￢Q∧R로 표현할 수 있고, BB4를 지나지 않는 수행 불가능한 경로인 BB1→ BB2→ BB3→ BB5→ BB6→ BB7→ BB9→ BB10→ BB11의 경우는 ￢P∧￢Q∧R 과 같이 표현할 수 있다. 이와 같은 경로 정보들을 모두 BDD로 표현하여 BDD의 논리합(disjunction)을 수행한다. 즉, (P ∧ ￢Q ∧ R) ∨ (￢P ∧ ￢Q ∧ R) ∨ … 을 수행하면 BDD의 요약(reduction) 알고리즘으로 인하여 자동으로 요약된 논리술어(predicate)를 얻을 수 있고 이를 통하여 요약된 수행 불가능 경로들의 추출할 수 있다. 이 예제에서는 BB1 -> BB11과 같이 함수에 진입 후 BB11을 통과하는 모든 경로들이 수행 불가능한 경로임을 나타내는 요약 정보를 추출할 수 있다. (상기 BDD에 대한 기술은 논문 [R. E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," ACM Computing Surveys, Vol. 24, No. 3, September 1992.]에 자세히 기재되고 있다.)

<69> 상기 고급프로그램 흐름분석기(240)는 상기한 방법으로 추출된 요약된 수행 불가능 경로들로 흐름제약사항(360)을 구성하여 생성한다.

<70> 상기 선형식 생성기(250)는 상기 TCFG(Timed CFG)(350)로부터 기본블록의 수행횟수에 대한 선형식(flow facts)(이하 IPET 선형식)(370)들을 추출하고, 상기 흐름제약사항(360)에 저장된 수행불가능 경로들을 IPET 선형식(370)들로 자동으로 변환한다.

<71> 상기 선형식 생성기(250)에서 생성되는 IPET 선형식(370)은 3가지 종류가 있다. 즉, 첫 번째는, 상기 TCFG(310)에서 한 기본블록(311)의 수행횟수는 들어오는 제어전이(312)의 수행횟수의 합과 같고, 나가는 제어전이(312)의 수행횟수의 합과 같다는 것으로부터 구해지는 기본 IPET 선형식이다. 두 번째는, 상기 TCFG(310)에서 한 기본블록(311)이 자체적으로 수행횟수를 한정되는 경우 상기 수행횟수를 한정하는 한정 IPET 선형식이 있다. 세 번째는, 상기 수행불가능 경로들 각각에 대하여, 수행불가능 경로로부터 변환하는 기능적 IPET 선형식이 있다.

<72> IPET(Implicit Path Enumeration Technique)는 정적 최장수행계산 방법들 중 하나로, 프로그램의 제어흐름(control flow)을 기본블록(basic block)의 수행 횟수에 대한 선형식들(flow facts)(이하 IPET 선형식)로 나타내고 이를 ILP(integer linear programming)를 사용하여 최대치를 계산하는 방법이다.

<73> 도 3의 ACFG(310)(TCFG는 ACFG에서 기본블록에 수행시간이 부가된 것으로 그래프 자체는 변함이 없으므로 이하 ACFG로 설명한다.)에서 기본블록 BBi의 수행횟수를 Xi라 하고 BBi로 부터 BBj로의 제어전이(transition or edge)의 수행 횟수를 fi_j라 하면, 한 기본 블록의 수행 회수는 들어오는 제어전이의 수행횟수의 합과 같고, 나가는 제어전이의 수행횟수의 합과 같다. 예를 들어, BB6의 수행횟수 X6은 BB5와 BB8로부터 들어오는 제어전이의 수행횟수의 합 f5_6 + f8_6과 BB7f로 나가는 제어전이의 수행횟수의 합 f6_7과 같다. 이를 IPET 선형식으로 표현하면 다음과 같이 표현할 수 있다.

<74> X6 = f5_6 + f8_6, X6 = f6_7

<75> 위와 같은 방법으로 모든 가능한 제어 흐름을 IPET 선형식으로 나타내면 도 6a와 같은 기본 IPET 선형식이 생성된다.

<76> 하지만 기본적인 IPET 선형식은 제어흐름의 대한 정보만을 포함하기 때문에 수행횟수를 한정하기 위한 추가적인 IPET 선형식들인 한정 IPET 선형식 (finiteness flow facts)들이 필요하다. 한정 IPET 선형식에는 함수의 ENTRY 노드의 수행횟수나 LOOP 헤더의 수행 횟수 등이 속한다. 예제는 한정 IPET 선형식으로 함수의 ENTRY 노드가 한번만 수행되는 것과 LOOP가 최대 4회 수행될 수 있다는 것을 6b와 같이 추가한다.

<77> 마지막으로, 상기 흐름제약사항(360)에 저장된 요약된 수행불가능한 경로로부터 기능적 IPET 선형식들을 만드는 기술을 설명하고자 한다. 지금까지 수행 불가능 경로를 IPET 선형식(370)으로 변환하는 일반적인 방법이 존재하지 않았다. 본 기술에서는 수행 불가능한 경로의 요약 정보를 IPET 선형식(370)으로 자동 변환하기 위한 알고리즘을 사용한다. 변환 알고리즘은 특징적으로 논리적 분리(disjunction)가 포함되게 되면 IPET의 성능이 심각하게 저하되기 때문에 되도록 최소한의 논리적 분리(disjunction)만을 포함하도록 설계되었다.

<78> 상기 수행불가능 경로들 각각에 대하여, 수행불가능 경로는 기본블록 b1에서 bn까지 차례로 수행되었을 경우 기본블록 c가 수행될 수 없는 경로이라고 하면, 다음과 같은 조건하에서 자동변환될 수 있다.

<79> 기본블록 c에서 bn이 도달 불가능하고, (b1,b2)(b2,b3) ... (bn-1,bn)들이 지배(dominance)관계가 성립할 경우,

<80> $\{ X_{bn} \le V(b_n) \times (1-X_c) \} \vee \{ X_c \le V(c) \times (1-X_{bn}) \}$

<81> (V(z)는 기본블록 z의 수행횟수의 한계, Xz 는 z의 수행횟수)

<82> 과 같은 형태로 변환할 수 있다.

<83> 예를 들어 BB1 -> BB11과 같은 수행 불가능한 경로는 BB11에서 BB1에 도달하는 경로가 존재하지 않기 때문에 위의 방법에 의하여

<84> { X1 <= 1 * (1 - X11) } ∨ { X11 <= 1 * (1 - X1) }

<85> ≡{ X1 <= 1 - X11 } ∨ { X11 <= 1 - X1 }

<86> ≡ X1 + X11 <= 1

<87> 과 같은 추가적인 기능적 IPET 선형식으로 변환될 수 있다.

<88> 상기 IPET 수행기(260)는 상기 선형식 생성기(250)에서 생성한 모든 IPET 선형식(370)을 제약조건으로 하고, 상기 TCFG(350)의 모든 기본블록에 대하여 각 기본블록의 수행횟수와 수행시간의 곱의 총합을 목적식으로 하여, 상기 제약조건을 만족하는 상기 목적식의 최대값이 되는 기본블록 수행횟수의 조합을 ILP(Integer linear Programming) 기법을 통해 구한다.

<89> 상기 ACFG(310)의 제어흐름이 기본 IPET 선형식과 한정 IPET 선형식 형태의 제약사항들로 표현되고 수행 불가능한 경로들도 기능적 IPET 선형식으로 변환되면, ACFG(310)상에서 가능한 프로그램의 수행경로는 이 제약사항들을 항상 만족하게 된다. 이 경로를 수행하는데 소요되는 시간은 각각의 기본블록의 수행횟수와 수행 시간의 곱의 총합으로 다음과 같은 목적식으로 표현할 수 있다. (Ti는 기본 블록 BBi의 수행 시간)

<90> $$ET = \sum_{i \in \text{모든 기본 블록}} X_i \times T_i$$

<91> 따라서 최장 수행 시간을 계산하는 문제는 모든 IPET 선형식들을 만족하는 수행 횟수의 조합들 중 목적식 ET의 값이 최대값이 되는 조합을 구하는 문제가 된다.

<92> $$WCET = MAX(ET) = MAX \sum_{i \in \text{모든 기본 블록}} X_i \times T_i$$

<93> 이와 같은 문제는 범용 ILP 솔루션(solver)들을 이용하여 구할 수 있다. 예제를 ILP 솔루션(solver)을 이용하여 계산한 최대값의 결과는 도 7과 같다. 이때 각각의 기본블록의 수행 횟수 조합을 WCET해, 최대값을 WCET값이라 정의한다.

<94> 상기 최장수행경로 생성기(270)는 상기 IPET 수행기(260)로부터 구한 각 기본블록의 수행횟수를 이용하여 최장수행경로를 추출한다.

<95> 분석자가 최장수행시간이 소요되는 경로를 파악할 수 있어야 프로그램 성능 최적화나 debugging에 그 정보를 이용할 수 있다. 때문에 최장수행경로 생성기(270)는 각각의 기본블록(311)이 수행된 횟수정보를 사용하여 최장수행경로를 추출한다. 최장수행경로는 입력 받은 프로그램상에서는 나타난 하나의 수행경로로 분석자가 시간이 많이 소요되는 병목지점을 파악하는 데 효율적인 정보가 된다.

<96> 상기 수행가능성 판별기(280)는 상기 IPET 수행기(260)로부터 구한 각 기본블록의 수행횟수를 검증하는 체킹코드를 소스코드(201)에 추가삽입하여 모델체킹(model checking)기법으로 수행가능여부를 검사하고, 수행이 불가능하면 상기 체킹코드의 체킹사항을 IPET 선형식(370)으로 변환하여 상기 IPET 선형식들에 새롭게 추가하고, 상기 IPET 수행기(260)를 통해 다시 최장수행경로를 구하도록 한다.

<97> 수행가능성 판별은 IPET 계산 결과로 나온 WCET 예측 값이 실제로 수행가능한지를 모델체킹(model checking) 기법을 이용하여 검사한다. 보통 IPET 결과는 여러 개의 프로그램 수행 경로에 해당되며 때에 따라서 그 수가 매우 많을 수 있으므로, 이들 경로를 하나하나 추출하고 그 수행 가능성을 검사하는 것은 심히 비효율적이다. 본 발명은 IPET 계산 결과에 포함된 각 기본블록의 수행횟수를 모델체킹 기법을 사용하여 직접 검사하여 그 수행가능성을 판별하므로 효율적이다.

<98> 이하 상기 수행가능성 판별기(280)의 구체적인 판별기능을 설명하고자 한다. 상기 수행가능성 판별기(280)에서 판별하는 기능은 각 기본블록들을 하나씩 검사하는 개별체킹(Individual checking)방식과 기본블록들의 짝의 관계를 검사하는 페어체킹(Pair checking)방식이 있다.

<99> 개별체킹방식은, 일부 기본블록들의 수행횟수를 알면 나머지 모든 기본블록의 수행횟수를 알 수 있는 상기 일부 기본블록들의 집합인 핵심집합(dominant set)을 구하고, 상기의 핵심집합(dominant set)의 각 기본블록의 수행횟수에 대하여, 수행횟수를 카운트하는 변수를 삽입하고 상기 변수가 수행횟수에 도달하는지 확인하는 체킹코드를 삽입하여, 상기 모델체킹기법에 의해 수행불가능하다는 판단이 되면 상기 기본블록의 수행횟수가 상기 수행횟수보다 적다는 IPET 선형식을 새롭게 추가하는 방식이다.

<100> 즉, IPET결과의 수행 가능성을 판별하기 위해 각 블록 하나씩 검사하는 방법이다. 이를 위해 해당 기본 블록이 실행될 때 마다 1씩 증가되는 새로운 변수를 추가한 후 그 변수의 값이 IPET결과로 나온 실행횟수가 될 수 있는지를 검사한다. 모든 기본 블록의 IPET 결과대로 수행될 수 있다면 IPET 결과가 수행 가능하다고 판단한다.

<101> 그런데 기본 블록들의 수행횟수는 프로그램 흐름에 의해 제약되므로 그 일부들의 수행횟수만 알아도 나머지 기본 블록의 수행횟수를 계산할 수 있다. 예를 들어 위 예제에서 BB4, BB8, BB11의 수행 횟수가 정해지면 나머지 블록들의 수행 횟수도 정해지므로 이들만 검사하면 IPET 결과의 수행 가능성을 알아낼 수 있다. 이러한 블록들을 핵심집합(dominant set)이라 하고, 이에 포함된 것들만 검사함으로써 모델체킹(model checking)으로 검사해야 하는 기본블록의 수를 줄여 검사의 효율성을 높일 수 있다.

<102> 도 7의 결과에서, 핵심집합(dominant set)인 {BB4, BB8, BB11}의 수행횟수는 각각 1, 3, 1이다. 이들의 수행횟수를 검증하기 프로그램에 수행을 카운터하는 변수를 삽입 후 자동 프로그램 검증 도구인 모델체커(model checker)를 사용하여 수행 횟수가 가능한지를 검증한다. 즉, 체킹코드는 각각 다음과 같다.

<103> assert(BB4 = 1),

<104> assert(BB8 = 3),

<105> assert(BB11 = 1)

<106> 즉, 핵심집합의 기본블록들에 대한 카운터 변수가 구해진 수행횟수에 도달하는지를 검증하는 것이다. 이 검사를 수행하면 (BB4 = 1), (BB8 = 3)은 가능하나 (BB11 = 1)은 불가능한 것으로 판별된다.

<107> 이에 불가능한 수행 횟수를 WCET계산에서 제외하기 위하여 (X11 < 1)이라는 새로운 IPET 선형식(flow facts)을 추가하여 다시 IPET 계산을 다시 수행하면 더욱 정확한 결과를 얻을 수 있다.

<108> 페어체킹(Pair checking)방식은, 상기의 핵심집합의 기본블록들의 모든 짝들의 각각에 대하여, 하나의 기본블록의 수행횟수가 p 이고 또 다른 하나의 기본블록 수행횟수가 q 이면, 상기 짝의 기본블록들의 합이 p + q 와 같은지를 확인하는 체킹코드를 삽입하여, 상기 모델체킹기법에 의해 수행불가능하다는 판단이 되면 상기 기본블록들의 짝의 합이 p + q 보다 작다는 IPET 선형식을 추가하는 방식이다.

<109> 개별체킹(Individual checking)방식은 각 블록의 최대 수행횟수를 검사하는 것으로 LOOP의 최대반복횟수 등을 검사하는 데는 유용하지만, 기본 블록들의 수행 횟수가 서로 영향을 미치는 블록간의 의존성을 검사할 수 없다. 만약 기본블록 BB4와 기본블록 BB11이 상호 배타적이어서 동시에 수행될 수는 없다고 할지라도 각각은 수행될 수 있기에 개별체킹방식은 이러한 의존성을 검사할 수 없다.

<110> 페어체킹(Pair checking)은 이러한 문제를 해결하기 위해 상기 핵심집합(dominant set)에 포함된 블록의 모든 짝(pair)을 검사한다. 도 3의 ACFG(310)의 예제에서는 {(BB4, BB8), (BB4, BB11), (BB8, BB11)}, 세 번의 검사가 이루어진다.

<111> 그런데 (BB4, BB11)를 검사하기 위해 체킹코드를 (BB4==1 && BB11==1)로 사용하는 경우 IPET 선형식(flow facts)으로의 변환에 문제가 생기므로 체킹코드를 (BB4 + BB11 == 2)로 사용한다. 만약 모델체킹 결과 체킹코드인 (BB4 + BB11 == 2)가 불가능하다고 판별되면 (BB4 + BB11 < 2) 를 새로운 IPET 선형식(flow facts)으로 사용하여 IPET를 다시 수행한다.

<112> 상기 수행가능성 판별기(280)는 기본블록들의 수행횟수에 관한 상기 수행가능성 검사와 새로운 IPET 선형식의 추출하는 과정과 다시 최장수행경로를 구하는 과정을 더 이상 새로운 IPET 선형식을 추출되지 않거나 더 이상 계산이 실질적으로 불가능할 때까지 반복적으로 수행한다.

<113> 수행 가능성 검사는 개별체킹(individual checking)방식을 먼저 하고 페어체킹(pair checking)방식을 후에 수행한다. 이러한 계산, 검사, 제약조건, 계산의 반복적으로 수행하는 과정에서 WCET 예측 값은 점점 더 정확해진다.

<114> 도 8은 본 발명의 최장수행시간 자동분석 방법에 대한 바람직한 일실시예를 도시한 것이다.

<115> 발명의 최장수행시간 자동분석 방법은 기본프로그램 흐름분석단계(S510)와, 수행시간분석단계(S520), 프로그램 정보추출단계(S530), 고급프로그램 흐름분석단계(S540), 선형식 생성단계(S550), IPET 수행단계(S560), 최장수행경로 생성단계(S570), 수행가능성 판별단계(S580)를 포함하는 것을 특징으로 한다.

<116> 상기 기본프로그램 흐름분석단계(S510)는 프로그램언어로 작성된 소스코드(201)를 입력받아서 기본블록(311)들과 상기 기본블록(311)들 사이의 제어전이(312)들을 파악하여, 상기 기본블록(311)과 상기 제어전이(312)을 플로우그래프 형태로 구성한 ACFG(Annotated Control Flow Graph)(310)를 생성하는 단계이다.

<117> 상기 수행시간분석단계(S520)는 하드웨어의 어셈블리 명령어별 수행시간정보를 명세한 하드웨어명세(320)를 사

전에 작성하고, 상기 소스코드(201)를 입력받아 컴파일러(221)를 통해 상기 소스코드(201)와의 추적성을 유지하는 목적코드(330)(어셈블리 명령어로 작성된 코드)를 생성하고, 상기 ACFG(310)의 기본블록(311)들에 해당하는 목적코드(330)들을 찾아 상기 하드웨어명세(320)를 참조하여 상기 기본블록(311)당 수행시간을 계산하고, 그 결과값을 저장하는 수행시간테이블(340)을 생성하는 단계이다.

<118> 상기 프로그램 정보추출단계(S530)는 상기 수행시간테이블(340)과 상기 ACFG(310)를 참조하여 기본블록당 수행시간과 기본블록사이의 의존성에 따른 수행시간 제약사항을 상기 ACFG(310)에 추가한 TCFG(Timed CFG)(350)를 생성하는 단계이다.

<119> 상기 고급프로그램 흐름분석단계(S540)는 상기 ACFG(310)을 참조하여 각 기본블록에서의 변수들이 가질 수 있는 범위를 파악하여 수행불가능 경로들을 추출하고, 상기 수행불가능 경로들을 BDD(binary decision diagram)로 만들어 논리합을 함으로써 요약된 수행불가능 경로들을 추출하고, 상기 요약된 수행불가능 경로들로 구성된 흐름제약사항(360)을 생성하는 단계이다.

<120> 선형식 생성단계(S550)는 상기 TCFG(Timed CFG)(350)로부터 기본블록의 수행횟수에 대한 선형식(flow facts)(이하 IPET 선형식)(370)들을 추출하고, 상기 흐름제약사항(360)에 저장된 수행불가능 경로들을 IPET 선형식(370)들로 자동으로 변환하는 단계이다.

<121> 상기 IPET 수행단계(S560)는 상기 선형식 생성기(250)에서 생성한 모든 IPET 선형식(370)을 제약조건으로 하고, 상기 TCFG(350)의 모든 기본블록에 대하여 각 기본블록의 수행횟수와 수행시간의 곱의 총합을 목적식으로 하여, 상기 제약조건을 만족하는 상기 목적식의 최대값이 되는 기본블록 수행횟수의 조합을 ILP(Integer linear Programming) 기법을 통해 구하는 단계이다.

<122> 상기 최장수행경로 생성단계(S570)는 상기 IPET 수행기(260)로부터 구한 각 기본블록의 수행횟수를 이용하여 최장수행경로를 추출하는 단계이다.

<123> 상기 수행가능성 판별단계(S580)는 상기 최장수행경로의 각 기본블록의 수행횟수를 검증하는 체킹코드를 소스코드(201)에 추가삽입하여 모델체킹(model checking)기법으로 수행가능여부를 검사하고, 수행이 불가능하면 상기 체킹코드의 체킹사항을 IPET 선형식(370)으로 변환하여 상기 IPET 선형식들에 새롭게 추가하여, 상기 IPET 수행단계(S560)를 다시 반복하는 단계이다.

<124> 상기 최장수행시간 자동분석방법에 대한 각 단계에 대한 구체적인 설명은 앞서 설명된 최장수행시간 자동분석장치에서 설명된 사항들을 참고하는 것으로 한다.

도면의 간단한 설명

<125> 도 1은 본 발명의 최장수행시간 자동분석도구의 구성에 대한 바람직한 일실시예,

<126> 도 2는 본 발명의 수행과정을 예시하기 위하여 본 발명에 입력되는 C언어로 작성된 소스코드의 예시도,

<127> 도 3은 본 발명에서 도 2에 예시된 소스코드의 기본블록과 제어흐름을 분석하여 생성되는 ACFG의 예시도,

<128> 도 4는 본 발명의 하드웨어명세의 바람직한 일실시예,

<129> 도 5는 본 발명에서 입력된 도 2에 예시된 소스코드를 소스코드와 추적성을 유지하도록 컴파일한 목적코드의 예시도,

<130> 도 6은 본 발명에서 입력된 도 2에 예시된 소스코드로부터 추출한 IPET 선형식들의 예시도,

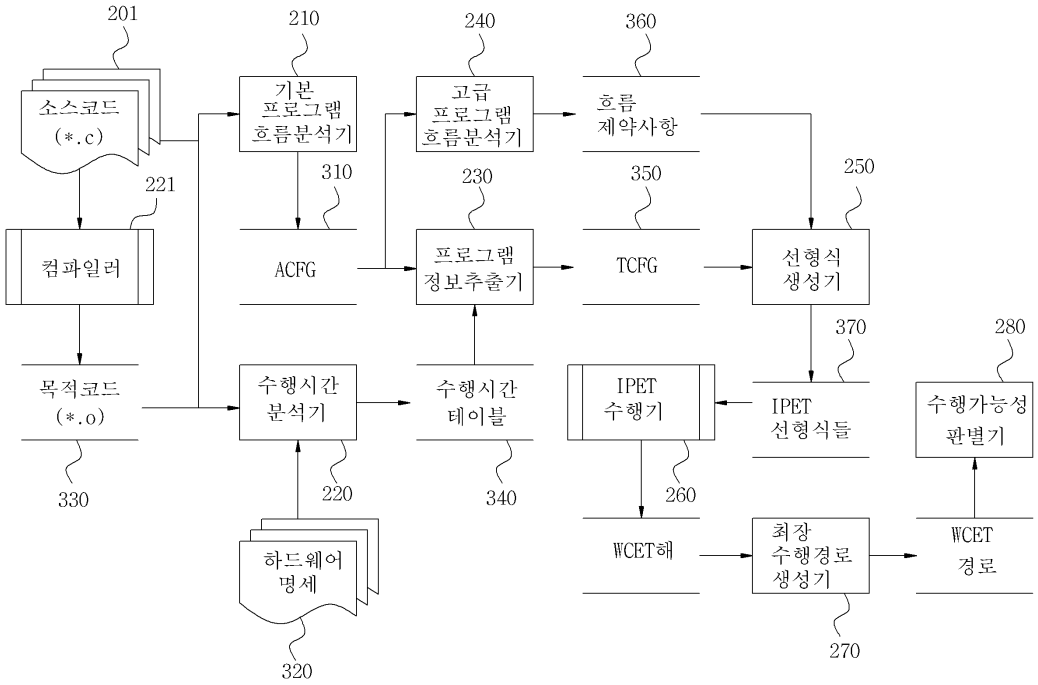
<131> 도 7은 본 발명에서 도 2에 예시된 소스코드로부터 IPET 기법을 이용하여 계산된 각 기본블록의 수행횟수와 최장수행시간의 예시도,

<132> 도 8은 본 발명의 최장수행시간 자동분석 방법에 대한 바람직한 일실시예,

<133> 도 9는 본 발명이 다목적 실용위성 명령어 처리모듈에 대해 최장수행분석을 한 결과를 도시한 것.

도면

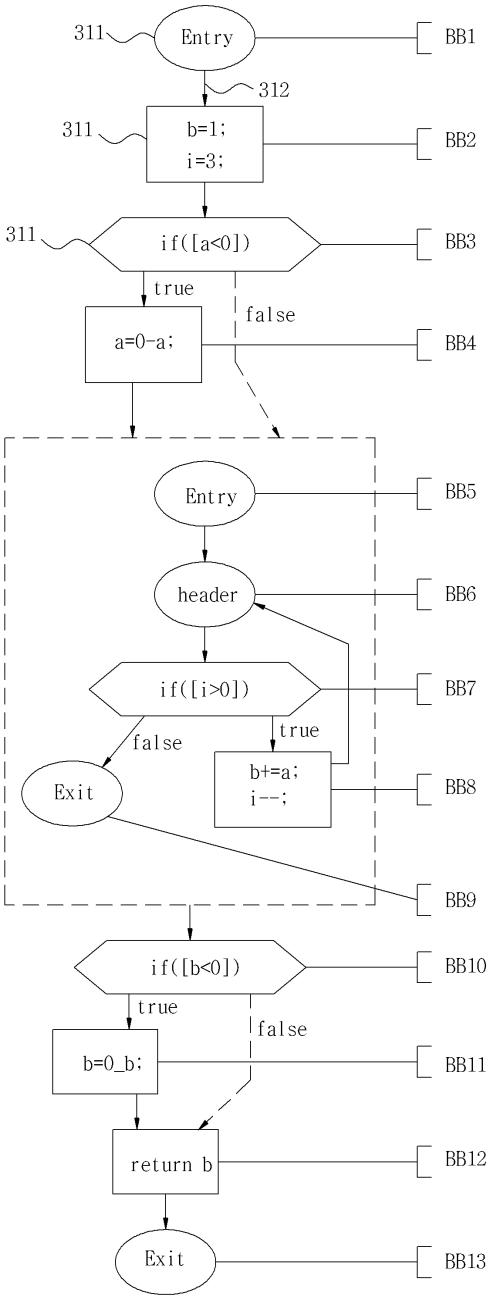
도면1


201
210
240
360
소스코드
(*.c)
기본
프로그램
흐름분석기
고급
프로그램
흐름분석기
흐름
제약사항
221
310
230
350
250
컴파일러
ACFG
프로그램
정보추출기
TCFG
선형식
생성기
370
280
목적코드
(*.o)
수행시간
분석기
수행시간
테이블
IPET
수행기
IPET
선형식들
수행가능성
판별기
330
220
340
260
하드웨어
명세
WCET해
최장
수행경로
생성기
WCET
경로
320
270

도면2

```
 1   int
 2   foo (int a)
 3   {
 4
 5     int b = 1;
 6     int i = 3;
 7
 8   if (a < 0)  {
 9       a = 0  -  a;
10   }
11
12   while (i > 0)  {
13       b = b + a;
14       i--;
15   }
16
17   if (b < 0)  {
18       b = 0 - b;
19     }
20
21   return b;
22
23   }
```

도면3


311
Entry
BB1
312
311
b=1;
i=3;
BB2
311
if([a<0])
BB3
true
false
a=0-a;
BB4
Entry
BB5
header
BB6
if([i>0])
BB7
false
true
Exit
b+=a;
i--;
BB8
BB9
if([b<0])
BB10
true
false
b=0_b;
BB11
return b
BB12
Exit
BB13

도면4

```
INSTRUCTION#241
    MNEMONIC=PUSH
    DESCRIPTION=Push (register (short form) )
    OPERAND=REG
    BITPATTERN=0 1 0 1 0 r r r
    TIME=Real=CYCLE:Static:2:b
    TIME=Virtual=CYCLE:Static:4:h
    CONDITION=
```

도면5

```
foo():
C:₩target₩NewProject₩Modules₩NewModule₩exam1.c:3
   0: 55                    push   %ebp
   1: 89 e5                 mov    %esp,%ebp
   3: 83 ec 08              sub    $0x8,%esp
C:₩target₩NewProject₩Modules₩NewModule₩exam1.c:5
   6: c7 45 fc 01 00 00 00  movl   $0x1,0xfffffffc(%ebp)
C:₩target₩NewProject₩Modules₩NewModule₩exam1.c:6
   d: c7 45 f8 03 00 00 00  movl   $0x3,0xfffffff8(%ebp)
C:₩target₩NewProject₩Modules₩NewModule₩exam1.c:8
  14: 83 7d 08 00           cmpl   $0x0,0x8(%ebp)
  18: 79 03                 jns    1d <_foo+0x1d>
C:₩target₩NewProject₩Modules₩NewModule₩exam1.c:9
  1a: f7 5d 08              negl   0x8(%ebp)
C:₩target₩NewProject₩Modules₩NewModule₩exam1.c:12
  1d: 83 7d f8 00           cmpl   $0x0,0xfffffff8(%ebp)
  21: 7e 0f                 jle    32 <_foo+0x32>
C:₩target₩NewProject₩Modules₩NewModule₩exam1.c:13
  23: 8b 45 08              mov    0x8(%ebp),%eax
  26: 8d 55 fc              lea    0xfffffffc(%ebp),%edx
  29: 01 02                 add    %eax,(%edx)
C:₩target₩NewProject₩Modules₩NewModule₩exam1.c:14
  2b: 8d 45 f8              lea    0xfffffff8(%ebp),%eax
  2e: ff 08                 decl   (%eax)
  30: eb eb                 jmp    1d <_foo+0x1d>
C:₩target₩NewProject₩Modules₩NewModule₩exam1.c:17
  32: 83 7d fc 00           cmpl   $0x0,0xfffffffc(%ebp)
  36: 79 05                 jns    3d <_foo+0x3d>
C:₩target₩NewProject₩Modules₩NewModule₩exam1.c:18
  38: 8d 45 fc              lea    0xfffffffc(%ebp),%eax
  3b: f7 18                 negl   (%eax)
C:₩target₩NewProject₩Modules₩NewModule₩exam1.c:21
  3d: 8b 45 fc              mov    0xfffffffc(%ebp),%eax
C:₩target₩NewProject₩Modules₩NewModule₩exam1.c:23
  40: c9                    leave
  41: c3                    ret
```

도면6a

기본블록	들어오는 제어전이에 관한 IPET 선형식들	나가는 제어전이에관한 IPET 선형식들
BB1		X1 = f1 _ 2
BB2	X2 = f1 _ 2	X2 = f2 _ 3
BB3	X3 = f2 _ 3	X3 = f3 _ 4 + f3 _ 5
BB4	X4 = f3 _ 4	X4 = f4 _ 5
BB5	X5 = f3 _ 5 + f4 _ 5	X5 = f5 _ 6
BB6	X6 = f5 _ 6 + f8 _ 6	X6 = f6 _ 7
BB7	X7 = f6 _ 7	X7 = f7 _ 8 + f7 _ 9
BB8	X8 = f7 _ 8	X8 = f8 _ 6
BB9	X9 = f7 _ 9	X9 = f9 _ 10
BB10	X10 = f9 _ 10	X10 = f10 _ 11 + f10 _ 12
BB11	X11 = f10 _ 11	X11 = f11 _ 12
BB12	X12 = f10_12 + f11_12	X12 = f12_13
BB13	X13 = f12 _ 13	

도면6b

수행횟수에 대한 한정	IPET선형식
함수 entry 노드에 대한 한정	X1 = 1
Loop의 반복 횟수에 대한 한정	X6 ≤ 4

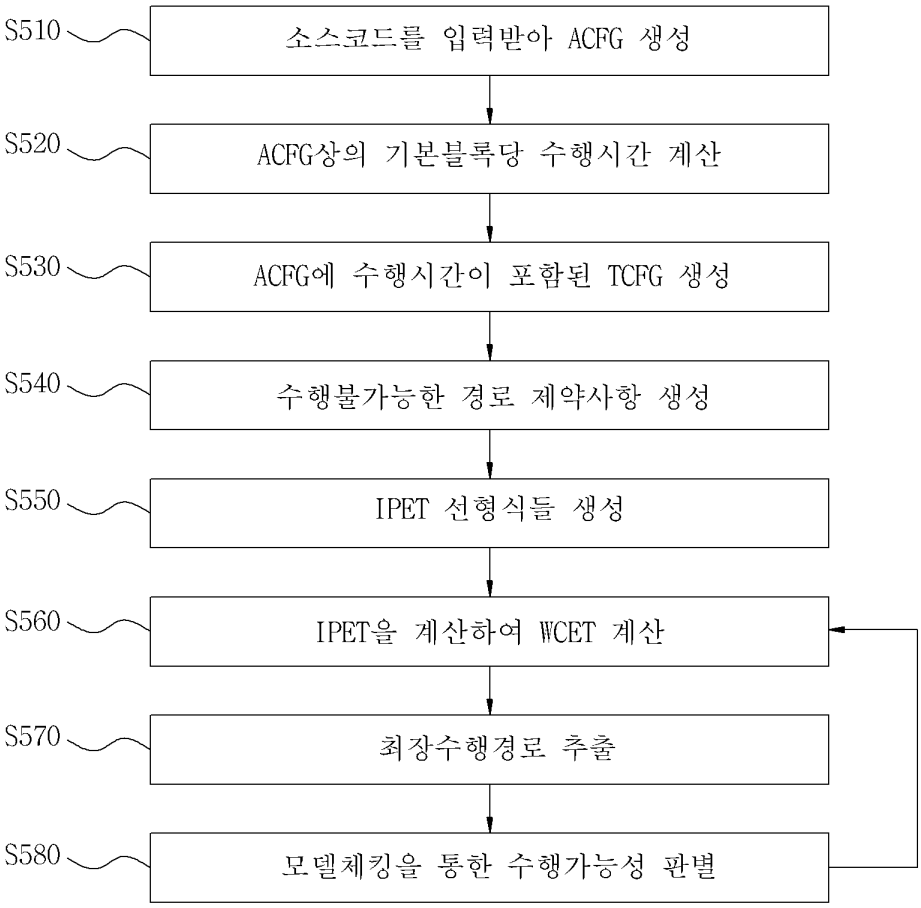
도면6c

수행불가능 경로	IPET 선형식
BB1 -> BB11	X1 + X11 ≤ 1

도면7

	수행횟수 (Xi)	수행시간 (Ti)	소요시간 (Xi * Ti)
BB1	1	0	0
BB2	1	4	4
BB3	1	13	13
BB4	1	6	6
BB5	1	0	0
BB6	4	0	0
BB7	4	13	52
BB8	3	29	87
BB9	1	0	0
BB10	1	13	13
BB11	1	8	8
BB12	1	61	61
BB13	1	0	0
최장수행시간	-	-	224

도면8


S510
소스코드를 입력받아 ACFG 생성
S520
ACFG상의 기본블록당 수행시간 계산
S530
ACFG에 수행시간이 포함된 TCFG 생성
S540
수행불가능한 경로 제약사항 생성
S550
IPET 선형식들 생성
S560
IPET을 계산하여 WCET 계산
S570
최장수행경로 추출
S580
모델체킹을 통한 수행가능성 판별

도면9

파일	라인수	측정기반분석 (CPU cycle)	본발명 (CPU cycle)	B/A
MAIN.C	178	315,480	357,019	113.2%
C001.C	205	12,988	16,824	129.5%
C002.C	227	12,903	14,392	111.5%
C003.C	516	313,339	325,717	104.0%
C004.C	141	9,551	14,061	147.2%
C005.C	169	212	222	104.7%
C006.C	163	9,835	143,66	146.1%
C007.C	295	1,231	1,284	104.3%
C008.C	646	312,123	308,820	98.9%
C009.C	250	12,704	14,061	110.7%
C010.C	172	12,957	14,359	110.8%
C011.C	94	9,385	13,922	148.3%
C012.C	243	9,952	14,679	147.5%
C013.C	196	10,347	16,440	158.9%
C014.C	73	169	171	101.2%
C015.C	89	11,486	13,921	121.2%
C016.C	718	315,340	354,011	112.3%
	4,197	1,054,522	1,137,250	122.3%

【심사관 직권보정사항】

【직권보정 1】

【보정항목】 청구범위

【보정세부항목】 제3항

【변경전】

TGFG(310)

【변경후】

TGFG(350)