2009 프로그래밍언어연구회 운영위원

위 원 장	변석우 (경성대)
부위원장	이광근 (서울대)
총 무	안준선 (한국항공대)
부 총 무	이욱세 (한양대)
재 무	이은영 (동덕여대)
학 술	신승철 (한국기술교육대) 조은선 (충남대) 조장우 (동아대)
사 업	김익순 (ETRI) 창병모 (숙명여대) 한환수 (성균관대)
기 획	이재진 (서울대)
해외학술	최진영 (고려대)
감 사	도경구 (한양대) 표창우 (홍익대)
편집위원장	백윤흥 (서울대)
편집위원	박성우 (포항공대) 방기석 (한림대) 우균 (부산대) 한경숙 (한국산업기술대) 이수현 (창원대)
고 문	김영택 (서울대) 김기태 (중앙대) 이기호 (이화여대) 우치수 (서울대) 원유헌 (홍익대) 최광무 (KAIST) 유재우 (숭실대) 오세만 (동국대)

프로그래밍언어논문지

한국정보과학회
프로그래밍언어연구회

제 23 권 제 1호 통권 제42호 2009년 12월 ISSN 1975-5961

차 례

■ **편집사** ····· 편집위원회

■ **연구논문**

• 알고리즘적인 학습을 통한 불변성 유추 ····· 정영범 · 공순호 · 이광근 1

• Dynamic-Partitioned SIMD 기법을 사용한 Reconfigurable Processor와 이를 위한 컴파일러 설계 ····· 권용인 · 윤종희 · 이종원 · 김용주 · 백윤흥 19

• 스타일을 설정할 수 있는 웹 기반 문서 편집 환경 ····· 황준형 · 윤정한 · 최석우 · 한태숙 29

• 소스코드 보안취약성 자동진단도구 개발 사례 ····· 권현준 · 김유경 · 김현하 · 도경구 · 신승철 · 안준선 · 이욱세 · 이은영 · 한환수 37

■ **학회소식**

• 연구회 회칙 ····· 46

• 논문지 투고 및 심사규정 ····· 47

한국정보과학회 프로그래밍언어연구회

편•집•사

프로그래밍언어 연구회 회원 여러분께,

안녕하십니까?

프로그래밍언어논문지의 2009년 제 23권 제 1호를 발간하게 된 것을 기쁘게 생각합니다. 이번 논문지는 3편의 연구 논문과 1편의 과제 소개 논문을 싣고 있습니다. 논문을 소개하면, 알고리즘적인 학습을 통한 불변성 유추, Dynamic-Partitioned SIMD 기법을 사용한 Reconfigurable Processor와 이를 위한 컴파일러 설계, 스타일을 설정할 수 있는 웹 기반 문서 편집 환경의 연구 논문과, 소스코드 보안취약성 자동진단도구 개발과제의 소개가 게재되었습니다.

본 논문지에 연구 결과들을 기고해 주신 저자 여러분과 논문지 발간을 위해 수고해주신 프로그래밍언어연구회 운영위원 여러분께 감사를 드리며, 모든 회원들의 건승과 프로그래밍언어연구회의 무궁한 발전을 기원합니다.

2009년 12월 25일
프로그래밍언어연구회
편집위원회

논문 23-1-R01

알고리즘적인 학습을 통한 불변성 유추[1)]

Inferring Quantifid Invariants via Algorithmic Learning, Decision Procedures, and Predicate Abstraction

정영범 · 공순호 · 이광근

서울대학교 컴퓨터공학부 프로그래밍 연구실

{dreameye; soon; kwang}@ropas.snu.ac.kr

요 약

알고리즘적인 학습(algorithmic learning), 참거짓 자동 판별기(decision procedures), 조건식 요약(predicate abstraction), 그리고 주어진 거푸집(templates)을 이용하여 프로그램의 반복문에 대하여 정량자(quantifir) 를 이용하여 표현되는 불변 성질(invari을 생성해내는 기술을 제시한다. 우리의 기술은 주어진 거푸집을 이용하여 임의의 1차 논리식로 표현되는 프로그램 반복문이 가지는 불변 성질을 찾아내며 불변 성질들에 존재하는 유연성을 간단한 무작위 방법(randomized mechanism) 을 통하여 활용하였다. 제안된 기술은 리눅스 소스 코드와 이전 연구의 벤치마크들로부터 정량자를 포함하는 프로그램 불변성들을 찾아낼 수 있었다.

1. 도 입

최근에 알고리즘적인 학습(algorithmic learning) 을 통해 프로그램의 불변성(program invari을 찾아내는 연구[28]가 발표되었다. 알고리즘적인 학습, 참거짓 자동 판별기(decision procedure), 조건식 요약(predicate abstraction) 을 결합하여 리눅스 장치 드라이버(Linux device driver) 프로그램에 있는 반복문(loop) 들에서 프로그램 불변성을 자동으로 찾아낼 수 있었다. 하지만, 그 연구[28]는 명백한 한계를 가지고 있다. 찾아 낼 수 있는 프로그램 불변성은 정량자를 쓸 수 없는 조건식(quantifir-free formula) 뿐이었다. 배열(array) 이나 그래프 같은 데이타 구조의 성질을 표현하기 위해서는 많은 경우에 정량자가 꼭 필요하다.

이 논문은 정량자가 필요한 프로그램의 불변성을 알고리즘적인 학습으로 구하는 방법에 관한 것이다. 알고리즘적인 학습, 참거짓 자동 판별기, 조건식 요약에 거푸집을 추가하여 정량자를 포함하는 프로그램 불변성을 찾아낸다. 리눅스 라이브러리, 커널, 그리고 장치 드라이버 프로그램에서 정량자를 가진 프로그램 불변성을 찾을 수 있었다.

우리의 방법은 주어진 거푸집에 대해 매우 일반적이다. 정량자를 가진 거푸집의 빈 부분에 들

1) 본 연구는 교육과학기술부/한국과학재단 우수연구센터 육성사업의 지원(과제번호 : R11-2008-007-01002-0)과 교육인적자원부 두뇌한국21사업의 지원으로 수행하였다. 이 논문의 영문버전 [14]은 CAV 2010에 제출되었습니다.

어갈 조건식을 학습 알고리즘을 이용해서 유추한다. 빈 부분에 들어가는 조건식은 어떤 모양이 되어도 상관없다. 이는 거푸집을 이용한 기존의 접근 방식[35]과는 다른 우리만 의장점이다. 기존 방식은 논리합(disjunction)을 포함하지 않는 식만을 빈 부분에 채울 수 있었다. 이는 거푸집을 정의할때 사용자가 프로그램 불변성에 대해 보다 자세한 정보를 제 공해야 하는 부담이 된다. 우리가 실험에서 (5장) 사용한 거푸집은 모두 "$\forall k.$[]"나 "$\forall k.\exists$i.[]"와 같이 간단한 거푸집이고, 이와 같은 간단한 거푸집으로도 실제 프로그램에서 프로그램 불변성을 찾을 수 있었다.

우리의 알고리즘은 다음과 같다. 이전 조건(precondition)과 이후 조건(postcondition)이 주어진 반복문에 대해서 학습 알고리즘은 질문을 통해 이진 논리식(Boolean formula)를 찾고, 이 논리식은 산술 조건식으로 조건식 요약을 통해 변환이 된다. 또, 산술 조건식은 정량자를 갖는 거푸집의 빈 공간을 채움으로써 완전한 정량자를 갖는 식이 완성이 된다. 학습 알고리즘은이진 논리식을 참거짓 자동 판별기는 산술 조건식을 다루기 때문에 산술식 요약과 구체화를 통해 참거짓 자동 판별기가 주어진 질문을 풀 수 있도록 연결했다. 반복문의 불변성이 어떤 것인지 모르는 상태로 질문에 답해야 하기 때문에 때로는 답을 하지 못하기도 한다. 이 경우에는 마구잡이로(random) 참, 거짓의 답을 한다. 물론 프로그램 분석을 통해 보다 정확한 정보를 얻어 답을 해줄 수도 있지만, 구하고자 하는 반복문에 대한 불변성질은 상당히 많이 존재하기 때문에 임의의 답을 통해 다른 반복문에 대한 불변성질 을 찾아가도록 안내할 수 있다. 너무 많은 답을 틀리는 최악의 경우에는 모든 알고리즘을 새로 시작하기도 한다. 이와는 반대로 이전의 반복문에 대한 불변성질 찾기기술은 이러한 유연성을 가지지 못한다. 이전 방법들은 정해진 답을 향해서 나아가 하나의 답을 얻거나 실패하기도 한다.

예 1. 우선 정량자가 없는 반복문에 대한 불변성질을 찾는 예제를 보여주겠다. 이전 논 문[28]에 있는 예제를 살펴보자.

$\{i=0\}$ while $i<10$ do b := nondet if b then $i:= i+1$ end $\{i=10\wedge b\}$

반복문이 끝난 후에는 변수 b는 참이어야 한다는 사실을 알 수 있다. 주어진 이전조건과 이후조건 그리고 반복문 조건으로 부터 $i=0$과 $(i=10\wedge b)\vee i<10$를 작은 근사(under-approximation) 큰 근사(over-approximation)로 사용한다. 참거짓 자동 판별기는 이 근사들을 가지고 학습 알고리즘의 질문에 답을 해준다. 몇번의 질문과 답이 오간 후 학습 알고리즘은 $i\neq 0\wedge i<10\wedge\neg b$이 불변성에 포함이 되어야 하느냐고 물어보게 된다. 이 질문은 그런데 작은 근사와 큰 근사사이에 속하므로 답을 해줄 수가 없다. 이때, 우리는 임의로 답을 준다. 만약 답이 틀릴 경우 학습 알고리즘은 주어진 정보로 부터 얻을 수 있는 최선의 답으로서 $i=0\vee i=10\wedge b)$를 물어보게 된다. 이는 우리가 찾는 답이 아니므로 반례(counter-example)를 줘야 하지만 주어진 근사로는 답을 할 수가 없으므로 모든 프로세스를 처음부터 다시 시작한다. 반면 만약 임의로 준 답이 맞았다면 두 번의 질문을 더 한후$(i=10\wedge b)\vee i<10$를 반복문의 불변성으로 찾게 된다.

예2. 정량자가 있는 반복문의 불변성을 찾는 예를 살펴보자.

while $i<n$ do if a[m] < $a[i]$ then $m = i$ fi; $i = i+1$ end

이 간단한 반복문은 $m=0\wedge i=0$를 이전조건으로 $\forall k.0 \le k < n \Rightarrow a[k] \le a[m]$을 이후 조건으로 갖는다. 이 반복문는 배열을 $a[0]$부터 $a[n-1]$까지 검사하여 최대값의 인덱스를 찾아내는 일을

한다. 이 반복문에 대해 $\forall k.[]$을 거푸집으로 주고 아래와 같은 단위 명제(atomic proposition) 들을 제공한다. 이 단위 명제들은 프로그램 불변성을 구성하는 단일 식으로 사용된다.

$\{i$<n, m $=0, i=0, k<n, a[m]<a[i], a[k] \leq a[m], k=i, k<i\}$.

단위 명제들은 $k=i$와 $k<i$를 제외하고는 모두 프로그램으로부터 도출될 수 있다. 정량자 변수 k를 다른 프로그램 변수 i나 n과 관계를 주어서 단위 명제들을 만드는 것은 직관적이다. 이와 같은 입력이 주어 졌을 때 우리의 알고리즘은 $\forall k.[]$와 같은 형식을 가지는 다음의 두가지 성질을 만족하는 반복문의 불변성 ι를 찾아낸다. (1) $(m=0 \wedge i=0) \Rightarrow \iota$ (2) $(\iota \wedge \neg(i<n)) \Rightarrow \forall k.0 \leq k<n \Rightarrow a[k] \leq a[m]$. 이는 곧 다음의 두가지 성질을 만족하는 산술 조건식 θ를 찾는 것과 동일하다. (1) $(m=0 \wedge i=0) \Rightarrow \forall k.\theta$ (2) $\forall k.\theta \Rightarrow (i<n \vee \forall k.0 \leq k<n \Rightarrow a[k] \leq a[m])$.

알고리즘적인 학습은 때때로 동전 던지기(임의의 답을 주는 것)를 하면서 식 $\forall k.(k \neq i) \vee (a[k] \leq a[m])$을 위의 조건을 만족하는 반복문에 대한 불변성질으로 찾아내게 된다. 하지만, 이것만이 우리 알고리즘이 찾아내는 유일한 답은 아니다. 실제로, 또 다른 실행을 통해 $\forall k.(i=0 \wedge k \neq n) \vee (a[k] \leq a[m]) \vee (k \neq i)$를 또 하나의 반복문에 대한 불변성질으로 찾아내기도 한다.

이 논문의 기여

- 우리는 알고리즘적인 학습, 참거짓 자동 판별기, 조건식 요약에 거푸집을 결합하여 정량자를 갖는 반복문에 대한 불변성질을 자동으로 찾는 방법을 제안했다.
- 제안하는 기술은 주어진 거푸집으로부터 정량자를 갖는 임의의 반복문 불변성을 찾아 낼 수 있다. 거푸집은 유연하다. 빈 부분이 하나이기만 하면 어떠한 거푸집도 사용될 수 있다. 예로, 빈부분에 들어갈 식은 논리합을 포함할 수 있다.
- 우리는 이 기술이 실제 프로그램에 대해서 적용이 가능하다는 사실을 증명했다. 리눅스 라이브러리, 커널, 장치 드라이버 프로그램과 다른방식의 논문[35]에 있는 벤치마크 프로그램에 대해 반복문 불변성을 찾아내었다.
- 이 기술은 반복문에 대한 불변성질을 찾는 간단하지만 효과적인 프레임 워크(framework) 로 볼 수 있다. 이 기술은 기존의 방법들과 독립적으로 다른 프로그램 분석이 알고리즘적 인 학습에 보다 자세한 정보를 주는 역할을 할 수 있다. 이 기술은 블랙박스로 사용하는 학습 알고리즘이나 참거짓 자동 판별기의 발전함에 따라 자동으로 발전할 수 있다.

논문의 구성은 다음과 같다. 프로그램에서 사용하는 용어들을 정리하고(2장), 우리의 프레임 워크의 전반적인 내용을 보여주고(3장), 어떻게 알고리즘적인 학습의 질문에 답하는 자세한 방법을 설명하겠다(4장). 이후에 실험결과를 보고하고(5장), 관련 연구(6장)와, 결론으로 논문을 마무리 하겠다(7장).

2. 용어 정리

우리가 대상으로 하는 프로그램의 핵심 문법 구조는 다음과 같다.

Stmt $\triangleq$ nop | Stmt Stmt | x := Exp | b := Prop
| a[Exp] := Exp | a [Exp] := nondet
| x := nondet | b:= nondet
| if Prop then Stmt else Stmt
| {Pred} while Prop do Stmt {Pred}

Exp $\triangleq$ n | x | a[Exp] | Exp+Exp | Exp−Exp

Prop $\triangleq$ F | b | ¬Prop | Prop∧Prop | Exp<Exp
| Exp=Exp

$$\text{Pred} \triangleq \text{Prop} \mid \forall x.\text{Pred} \mid \exists x.\text{Pred} \mid \text{Pred} \wedge \text{Pred} \mid \neg \text{Pred}$$

대상 언어는 이진값(Boolean)과 정수값의 두 가지 기본 타입을 갖는다. 식 Exp는 자연수의 Prop는 이진 타입의 식이다. 명령문 nondet을 통해 임의의 값이 변수에 저장될 수 있다. 반복문 $\{\delta\}$ while k do S $\{\epsilon\}$ k를 반복문 조건으로, δ와 ϵ을 이전조건과 이후조건으로 부르겠다. 이전조건과 이후 조건은 정량자가 있는 식인 Pred에 속한다. 아래와 같은 산술 조건식 b, $\pi_0 < \pi_1$, $\pi_0 = \pi_1$을 단위 명제(atomic proposition) 라고 한다. 만약 A를 단위 명제들의 집합이라고 하면, $Prop_A$과 $Pred_A$는 집합 A로 만들 수 있는 산술 조건식의 집합과 정량자를 갖는 식의 집합을 각각 나타낸다.

템플릿 $t[] \in \tau$은 1차논리식(fist-order logic formula)의 집합인 Prop_A의 한 원소로 표현이 되며 빈 공간을 하나 가지고 있다. 이 공간은 산술 조건식의 집합 $Prop_A$의 한 원소로 채우게 된다.

$$\tau \triangleq [] \mid \neg\tau \mid \text{Prop}_A \wedge \tau \mid \text{Prop}_A \vee \tau \mid \forall I.\tau \mid \exists I.\tau.$$

산술 조건식 $\theta \in \text{Prop}_A$에 대해서 $t[\theta]$은 템플릿 $t[]$의 빈 공간을 θ로 채운 식이 된다.

반복문 $\{\delta\}$ while κ do S $\{\epsilon\}$와 템플릿 $t[] \in \tau$가 주어졌다고 가정하자. 그리고, 주어진 명령문 S에 대한 이전조건 Pre(ρ, S)은 1차논리식으로서 이 조건을 가지고 S를 실행하면 ρ를 만족하게 한다. 반복문의 불변성을 템플릿 $t[]$를 가지고 찾는 문제는 다음을 만족하는 1차논리식 $t[\theta]$를 찾는 문제이다. (1) $\delta \Rightarrow t[\theta]$; (2) $\neg\kappa \wedge t[\theta] \Rightarrow \epsilon$; 그리고, (3) $\kappa \wedge t[\theta] \Rightarrow \text{Pre}(t[\theta], S)$.

값할당 ν는 자연수를 자연수 변수에 참거짓 값을 이진 변수(Boolean variable)에 할당하는 것이다. 단위 명제의 집합 A에 대해서 $Var(A)$는 A에 들어있는 변수들의 집합이고, $Val_{Var(A)}$는 그 변수들에 대한 값할당의 집합이다. 만약 ρ가 주어진 값할당 ν에 의해 참값 을 가지게 되면 ν가 1차 논리식 ρ의 모델(model) 이라고 하며 $\nu \models \rho$라고 쓴다. 이진 변수의 집합 B에 대해 BoolB는 그 이진 변수들로 이루어진 이진 식(Boolean formula)의 집합을 의미한다. 이진 값할당(Boolean valuation) μ는 참거짓 값을이진 변수들에 할당하는 것을 뜻한다. 이진 값할당의 집합은 Val_B과같이 쓴다. 이진 식 β가 이진 값할당 μ에 의해 참값을 가지면 이진 모델(Boolean model)이라고 하며 $\mu \models \beta$라고 쓴다.

주어진 1차논리식 ρ에 대해 참거짓 자동 판별기(satisfibility modulo theories(SMT) solver)는 만약 모델이 존재하면 그 모델 ν를 돌려주거나 ($SMT(\rho) \rightarrow \nu$라고 쓴다.), 존재 하지 않으면 $UNSAT$을 돌려준다($SMT(\rho) \rightarrow UNSAT$라고 쓴다.). [15, 30].

2.1 CDNF 학습 알고리즘

CDNF(Conjunctive Disjunctive Normal Form) 알고리즘은 정확한 학습 알고리즘이다. 이 알고리즘은 오라클과 소통하면서 이진 식 $\lambda \in Bool_B$를 정확히 찾아낸다. 오라클은 다음의 두가지 종류의 질문에 답을 해야 한다.

- 소속 질문(Membership query $MEM(\mu)$ where $\mu \in Val_B$). 만약 μ가 찾고자 하는 이진 식 λ의 모델이면 YES라고 답을, 아니면 NO라고 대답을 해야 한다.
- 동치 질문(Equivalence query $EQ(\beta)$ where $\beta \in Bool_B$). 만약 주어진 이진식 β가 찾고자 하는 이진 식 λ와 동치라면 YES라고 대답을 해야 하고, 아니면 반례(counterexample)를 하나 돌려줘야 한다. 반례는 값할당 $\mu \in Val_B$B인데 β와 λ과 다른 값을 갖도록 해야 한다.

이진 식 $\lambda \in Bool_B$에 대해 $|\lambda|$CNF와 $|\lambda|$DNF를 λ와 동치인 제일 작은 이진 식의 CNF와 DNF의 크기라고 하면, CDNF 알고리즘은 임의의 $\lambda \in Bool_B$를 $|\lambda|$CNF, $|\lambda|$DNF, $|B|$의 다항 횟수(polynomial number)의 질문만으로 찾아낸다[9].

3. 프레임웍(Framework) 개괄

우리는 알고리즘적인 학습 [9], 참거짓 자동 판별기 [15], 조건식 요약 [18]과 거푸집을 결합한 구조를 사용한다. [그림 1]은이 기술들이 어떻게 관계를 맺고 사용되고 있는지를 보여준다. 왼쪽 편은 오라클(teacher), SMT solver와 정적 분석기(static analyzer)가 1차논리식을 다루는 구체적 도메인(concrete domain)을 나타낸다. 오른 편은 알고리즘적 학습이이진 식을 다루는 요약 도메인(abstract domain)을 나타낸다.

주어진 반복문과 거푸집 $t[]$에 대해 우리는 CDNF 알고리즘을 통해 거푸집에 맞는 불변성을 찾고자 한다. 거푸집은 빈공간을 하나 가지고 있는 1차논리식이라는 사실을 상기하자.

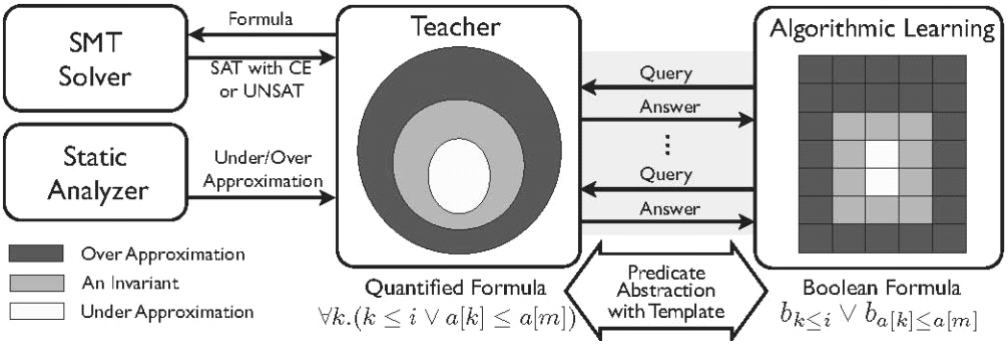
우리는 적합한 조건식 η를 찾아서 $t[\eta]$가 주어진 반복문에 불변성이 되도록 하려 한다. CDNF 알고리즘은 이 η를 찾으면 된다.

이 목표를 위해 우리는 두가지 문제를 해결해야 한다. 첫째로, CDNF 알고리즘은 조건식 이 아닌 이진 식을 학습하는 알고리즘이다. 따라서, 이 알고리즘은 주어진 거푸집을 채울 조건식을 찾을 수 없다. 둘째로, CDNF 알고리즘은 두가지 유형의 질문에 답을 해줄 오라클 을 필요로 한다. 자동으로 불변성을 찾아내기 위해 우리는 이 오라클 역할을 해줄 기계적인 프로시져를 고안해야만 한다.

첫번째 문제를 해결하기 위해 우리는 조건식 요약을 통해 이진 식과 산술식을 연결한다. 조건식 요약을 통해 각각의 단위 명제를 하나의 이진 변수들로 대응시킨다. 조건식 η를 유추하는 대신 CDNF 알고리즘이 그에 대응되는 이진 식 λ를 유추할 수 있도록 한다.

두번째 문제를 해결하기 위해 우리는 이진 식 λ에 대한 질문들에 답을 해줄 알고리즘을 고 안해야 한다. 두가지 유형의 질문이 있다. 소속 질문은 주어진 이진 값할당이 반복문에 대한 불변성질의 모델인지를 물어본다. 동치 질문은 주어진 이진 식이 반복문에 대한 불변성질과 같은지를 물어보고 아니면 반례를 돌려줘야 한다. 요약 도메인에서의 질문을 구체화 하는 것은 어렵지 않지만, 질문의 답을 하기 위해서는 반복문에 대한 불변성질에 대한 정보가 필요하다.

반복문의 불변성을 모르고 있지만 그것의 근사치는 반복문의 이전조건, 이후 조건 혹은 프로그램 분석을 통해 얻을 수 있다. 질문에 답하는 알고리즘은 이 근사치를 이용하여 답을 한다. 만



[그림 1] 전체적인 구조

약 이 근사치를 통해서도 답을 할 수 없는 경우는 간단히 랜덤하게 CDNF 알고리즘에게 답을 한다. 소속 질문에 대해서는 그것의 구체화된 할당값이 반복문의 작은 근사치(under-approximation)에 들어가는지, 혹은 반복문의 큰 근사치(upper-approximation)에 바깥에 해당하는지를 살펴본다. 만약 작은 근사치에 속한다면 이 할당값은 실제 반복문에 대한 불변성질의 모델일 수밖에 없다. 만약 큰 근사치의 바깥에 있다면 이 할당값은 반복문에 대한 불변성질의 모델이 될 수가 없다. 둘 다 아니라면 랜덤하게 답을 해준다. 동치 질문에 대해서는 만약 질문의 구체화된 식이 작은 근사치보다 약하지 않거나, 큰 근사치보다 강하지 않으면 반례를 SMT solver를 이용하여 만들어 낸다. 아니라면 랜덤하게 반례를 만들어 준다. 실제로 충분히 많은 반복문에 대한 불변성질이 존재한다면, 우리의 이 간단한 랜덤 알고리즘이 CDNF 알고리즘을 많은 반복문 불변성 중 하나를 찾을 수 있도록 안내할 것이다.

4. 정량자를 포함하는 불변성 학습

우리의 목표는 알고리즘적인 학습과 거푸집을 이용해 정량자를 갖는 반복문의 불변성을 찾는 것이다. 이 목표를 위해 우리는 (1) 세 개의 도메인들이 어떤 관계를 갖고 있는지 확인해야 한다(4.1장); (2) 거푸집에 맞는 1차식들과 조건식과의 관계를 설명하는 보조 정리들을 만들어야 한다(4.2장); (3) CDNF 알고리즘이 만들어내는 질문에 답을 해주는 오라클을 고안해야 한다(4.3장).

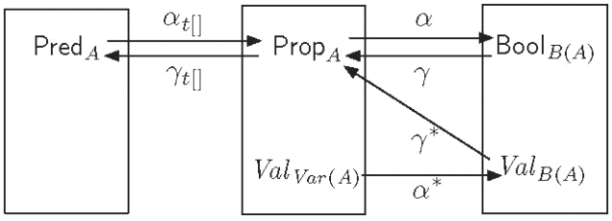
4.1 거푸집을 이용한 조건식 요약

단위 명제의 집합 A에 대해서 $B(A) \triangleq \{b_p : p \in A\}$를 대응하는 이진 변수들의 집합이라고 하자. 그림 2는 우리가 사용하는 도메인들을 보여주고 있다. 왼쪽의 상자는 A로부터 생성 할 수 있는 1차논리식들의 집합 $Pred_A$을 나타낸다. 여기서, 우리는 주어진 거푸집 $t[] \in \tau$에 해당하는 1차논리식에 관심을 가지고 있다. 따라서 $S_t[] = \{t[\theta] : \theta \in Prop_A\} \subseteq Pred_A$이 반복문에 대한 불변성질 찾기 문제의 해답 영역(solution space)을 형성한다. 가운데 상자는 A로부터 생성되는 조건식 $Prop_A$을 나타낸다. 해답 영역 $S_t[]$은 주어진 템프릿 $t[]$에 의해 형성되기 때문에, $Prop_A$은 사실 $S_t[]$의 본질을 구성한다. 오른쪽 상자는 B(A)로부터 생성되는 이진 식들의 집합 $Bool_{B(A)}$을 나타낸다. CDNF 알고리즘이 여기에 있는 이진 식을 유추하게 된다.

쌍 (γ, α)은 두 도메인 $Bool_{B(A)}$과 $Prop_A$ 사이에 관계를 정립한다. 이진 식 $\beta \in Bool_{B(A)}$이 논리곱으로 묶이고 각각의 변수들이 한번씩 밖에 등장하지 않는 경우에 규범 단항식(canonical monomial)이라고 부른다. 다음을 정의한다.

$\gamma : Bool_{B(A)} \rightarrow Prop_A$ $\alpha : Prop_A \rightarrow Bool_{B(A)}$

$\gamma(\beta) = \beta[\overline{b_p} \mapsto \overline{p}]$ $\alpha(\theta) = \vee \in Bool_{B(A)} : \beta$ is a canonical monomial and $\theta \wedge \gamma(\beta)$ is satisfible}.



[그림 2] 도메인 $Pred_A$, $Prop_A$, $Bool_{B(A)}$

구체화 함수 $\gamma(\beta) \in Prop_A$는 $B(A)$에 속하는 이진 변수들을 그에 해당하는 A에 속하는 단위 명제로 변환한다. 반면 $\alpha(\theta) \in Bool_{B(A)}$는 조건식 $\theta \in Prop_A$의 요약에 해당한다. 단 위 명제에서 성립하던 어떠한 관계도 요약을 하고 나면 사라지게 된다. 예를 들어, 조건식

$x = 0 \wedge x > 0$는 만족될 수 없지만, 이것의 요약인 $\alpha(x = 0 \wedge x > 0) = b_x = 0 \wedge b_x > 0$은 값할당 $b_x = 0 = b_x > 0 = T$를 통해 만족될 수 있다.

쌍($\gamma_t[]$, $\alpha_t[]$)은 도메인 $Prop_A$과 $Pred_A$의 관계를 정립한다.

$$\gamma_t[] : Prop_A \rightarrow Pred_A \qquad \alpha_t[] : Pred_A \rightarrow Prop_A$$
$$\gamma_t[](\theta) = t[\theta] \qquad \alpha_t[](\gamma_t[](\theta)) = \theta$$

주어진 거푸집 $t[]$에 대해 $\gamma_t[](\theta)$는 거푸집의 빈공간을 조건식 θ로 채워 넣는다. 반면, $\alpha_t[]$는 빈공간에 해당하는 조건식 θ를 돌려준다. 예를 들어, 거푸집 $t[] = \forall k.(k \neq i) \vee []$에 대해서 $\alpha_t[](\forall k.(k \neq i) \vee (a[k] \leq a[m])) = a[k] \leq a[m]$이고, $\gamma_t[](a[k] = a[m]) = \forall k.(k \neq i) \vee (a[k] = a[m])$이다.

소속 질문에 답하기 위해 우리는 이진 값할당 $\mu \in Bool_{B(A)}$와 조건식 $\gamma^*\mu$) 혹은 1차논리식 $\gamma^*_\tau(\mu, t[])$ 간의 관계를 정립해야 한다. 값할당 $\nu \in Var(A)$는 이진 값할당 $\alpha^*(\nu) \in Val_{B(A)}$를 도출한다. 이것은 동치 질문에 답을 할 때 반례를 찾는 데에 유용하다.

$$\gamma(\mu) = \bigwedge_{p \in A} \{p : \mu(b_p) = \mathrm{T}\} \wedge \bigwedge_{p \in A} \{\neg p : \mu(b_p) = \mathrm{F}\}$$
$$\gamma^*(\mu, t[]) = \gamma_t[](\gamma^*\mu))$$
$$(\alpha^*(\nu))(b_p) = \begin{cases} T & \text{if } \nu \models p \\ F & \text{otherwise} \end{cases}$$

편의상 $\alpha_\tau(\rho, t[]) = \alpha(\alpha_t[](\rho))$와 $\gamma_\tau(\beta, t[]) = \gamma_t[](\gamma(\beta))$를 사용하겠다. 다음을 증명하는 것은 너무도 쉽다:

$$\alpha(\gamma(\beta)) = \beta \qquad \alpha_t[](\gamma_\tau(\beta, t[])) = \gamma(\beta)$$
$$\alpha_t[](\gamma^*_\tau(\mu, t[])) = \gamma^*(\mu)$$

다음 보조정리가 앞으로의 내용을 위해 중요하다.

보조정리 4.1 ([28]) *Let A be a set of atomic propositions, $\theta \in Prop_A$, $\beta \in Bool_{B(A)}$, and ν a valuation for $Var(A)$. Then*

(1) $\nu \models \theta$ if and only if $\alpha^*(\nu) \models \alpha(\theta)$ and
(2) $\nu \models \gamma(\beta)$ if and only if $\alpha^*(\nu) \models \beta$.

보조정리 4.2 ([28]) *Let A be a set of atomic propositions, $\theta \in Prop_A$, and μ a Boolean valuation for $B(A)$. Then $\gamma^*(\mu) \Rightarrow \theta$ if and only if $\mu \models \alpha(\theta)$.*

4.2 거푸집의 성질들

$t[] \in \tau$를 거푸집 그리고 $\theta \in Prop_A$을 조건식이라고 하자. 만약 $t[\theta]$가 불변식이 되기에 너무 강한 조건이라면, 우리는 θ를 변경하여 $t[\theta]$를 완화시키고자 한다. 하지만 $t[\theta]$의 의미는 빈공간의 문맥에 의존한다. 두 조건식 θ_1, $\theta_2 \in Prop_A$에 대하여 θ_1이 θ_2보다 약하다고 해서, 그것을 거푸집에 적용한 $t[\theta_1]$가 $t[\theta_2]$에 비하여 항상 약한 것은 아니다. 그러므로 우리는 θ_1, θ_2와 그에 상응하는 $t[\theta_1]$, $t[\theta_2]$ 사이의 연결 관계를 정립할 필요가 있다(정리 4.4). 이 연결은 질문에 답하는 알고리즘의 설계에 필수적이다.

우리는 거푸집의 빈공간이 갖는 극성에 따라서 거푸집들을 다음과 같이 분류할 수 있다.

정의 Let $t[] \in \tau$ be a template.

(1) $t[]$ is positive if $\forall\theta_1, \theta_2 \in Prop_A$:
$(\theta_1 \Rightarrow \theta_2) \Rightarrow (t[\theta_1] \Rightarrow t[\theta_2])$;
(2) $t[]$ is negative if $\forall\theta_1, \theta_2 \in Prop_A$:
$(\theta_1 \Rightarrow \theta_2) \Rightarrow (t[\theta_2] \Rightarrow t[\theta_1])$.

위 정의 4.2의 의미 조건들은 확인하기 어려운 것이다. 그러므로 우리는 다음과 같은 문법적인 분류를 도입한다.

$$\tau+ \triangleq [] \mid \forall I.\tau+ \mid \exists I.\tau+ \mid \neg\tau- \mid \tau+ \vee Prop_A \mid \tau+ \wedge Prop_A$$
$$\tau- \triangleq \forall I.\tau- \mid \exists I.\tau- \mid \neg\tau+ \mid \tau- \vee Prop_A \mid \tau- \wedge Prop_A$$

문법적인 클래스들인 $\tau+$와 $\tau-$는 본래의 의미적인 정의들을 안전하게 기술한다.

보조정리 4.3 *Let $t[] \in \tau$ be a template. If $t[] \in \tau+$(or $t[] \in \tau-$, then $t[]$ is a positive template (or negative template respectively).*

보조정리 4.3는 $\theta_1 \Rightarrow \theta_2$가 만족되었을 때에 $t[\theta_1]$와 $t[\theta_2]$ 사이의 관계를 추론하는데 유용 하다. 또 다른 방향을 위해서, 다음의 정의를 생각해보자.

정의 Let $\theta \in Prop_A$ be a propositional formula over A. A well-formed template $t[]$ with respect to θ is defied as follows.

- [] is well-formed with respect to θ;
- $\forall I.t'[]$, $\exists I.t'[]$, and $\neg t'[]$ is well-formed with respect to θ if $t'[]$ is well-formed with respect to θ
- $t'[] \vee \theta'$ is well-formed with respect to θ if t' $[\theta] \not\Rightarrow \theta'$ and $t'[]$ is well-formed with respect to θ
- $t'[] \wedge \theta'$ is well-formed with respect to θ if $t[\theta]$ and $t'[]$ is well-formed with respect to θ.

잘 정의된 거푸집이 기반하고 있는 직관은 Pred 도메인의 관계인 $t[\theta_1] \Rightarrow t[\theta_2]$은 반드시 거푸집의 빈 공간의 입력이 되는 조건식 θ_1, θ_2에 의존해야 한다는 것이다. 만약 θ_1과 θ_2에 무관하게 $t[\theta_1] \Rightarrow t[\theta_2]$ 이 성립한다면 Prop 도메인의 어떠한 관계도 보장될 수 없다. 그러므로 잘 정의된 거푸집$t[]$를 이용하면 다음의 정리를 보일 수 있다.

정리 4.4 *Let A be a set of atomic propositions, $\theta_1 \in Prop_A$ and $t[] \in \tau$ a template.*

(1) *If $t[]$ is positive and well-formed with respect to θ_1, then $\forall\theta_2 \in Prop_A.(t[\theta_1] \Rightarrow t[\theta_2]) \Rightarrow (\theta_1 \Rightarrow \theta_2)$;*
(2) *If $t[]$ is negative and well-formed with respect to θ_2, then $\forall\theta_1 \in Prop_A.(t[\theta_1] \Rightarrow t[\theta_2]) \Rightarrow (\theta_2 \Rightarrow \theta_1)$.*

증명 *The proof uses structural induction on $t[]$.*[2)]

4.3 질문들에 답하기

집합 A를 단위 명제의 집합이라고 하자. 이전 조건과 이후 조건이 덧붙여진 반복문과 불변 식의 거푸집이 주어졌다고 하자. 우리의 목표는 CDNF 알고리즘을 적용하여서 $\gamma_\tau(\lambda, t[])$가 주어진 반복문에 대한 불변식이 되는 이진식 $\lambda \in Bool_{B(A)}$을 찾는 것이다. CDNF 알고리즘은 두 가지 종류의 질문들을 만들어내는 것을 기억하자. 소속 질문 $MEM(\mu)$는 μ가 λ에 대한 모델인지를 물어본다. 동치 질문 $EQ(\beta)$은 β가 λ와 동치인지를 묻는다. 우리의 설정에서는, 목표 이

2) 전체 증명은 [14]에 수록 되어 있다.

진 함수 λ는 알려져 있지 않다. λ를 알지 못하는 상태에서 질문들에 답하는 알고리즘을 설계하기 위해서 우리는 불변식에 대한 근사값들을 이용한다.

$\iota \in Pred_A$가 반복문 $\{\delta\}$ while κ do S $\{\epsilon\}$ 대한 불변식이라 하자. 우리는 두 조건 $\delta \Rightarrow \underline{\iota}$ 과 $\underline{\iota} \Rightarrow \iota$를 만족할 때에 $\underline{\iota} \in Pred_A$를 불변식 ι에 대한 작은 근사(under-approximation)라고 한다. 마찬가지로, 두 조건 $\iota \Rightarrow \overline{\iota}$ and $\overline{\iota} \Rightarrow \epsilon \vee \kappa$을 만족하는 $\overline{\iota} \in Pred_A$을 불변식 ι에 대한 큰 근사(over-approximation)이라고 한다. 가장 강력한 근사값 (δ)과 가장 약한 근사값 ($\epsilon \vee \kappa$)은 모든 불변식에 대하여 당연한 작은 근사값과 큰 근사값이 될 수 있다.

이어지는 논의에서, 우리는 어떤 $\eta \in Prop_A$가 존재하고 $\iota = t[\eta]$가 주어진 거푸집 $t[]$에 대한 1차 논리 불변식임을 가정한다.

4.3.1 *소속 질문.* 소속 질문 $MEM(\mu)$에서 우리의 소속 질문에 대답하는 알고리즘은 $\mu \models \lambda$인지를 답해야한다. A에 속하는단위 명제들 사이의 관계는 요약 도메인인 $Bool_{B(A)}$에서 잃어버리게 되는 것을 기억하자. 한 값할당이 모순되지 않는 조건식과 상응하지 않을 수도 있다. 만약 값할당 $\mu \in Val_{B(A)}$이 모순되는 조건식(즉, $\gamma^*(\mu)$을 만족시키는 모델이 존재하지 않는 경우)과 상응되는 경우에, 우리는 소속 질문에 대하여 "아니요"라고 답한다. 다른 경우에 우리는 $\gamma^*_\tau(\mu, t[])$를 불변식 근사값들과 비교한다.

알고리즘 1은 우리의 소속 질문에 답하는 알고리즘을 제공한다. 주어진 거푸집 $t[]$과 조건 식 $\theta \in Prop_A$에 대해서 is WellFormed($t[]$, θ) 프로시저는 거푸집 $t[]$가 θ에 대해서 적합한지(well-formed)를 검사한다. 이것은 거푸집의 구조를 이용한 간단한 귀납법을 이용하여서 구현되어 있고, 따라서 이곳에서는 생략하기로 한다. 소속 질문에 답하는 알고리즘에서, 우리는 먼저 μ가 해를 갖는 식과 대응되는 지를 검사한다. 만약 그렇다면 우리는 거푸집의 빈 공간에 의해서

(* $\underline{\iota}$: an under-approximation; ι: an over-approximation *)
(* $t[]$: the given template *)
Input : a valuation μ for $B(A)$
Output : *YES* or *NO*
if *SMT* $(\gamma^*\mu)) \rightarrow$ *UNSAT* then return *NO;*
ρ := $\gamma^*_t(\mu, t[])$;
if $t[] \in \tau_+$ then
    if *SMT* $(\rho \wedge \neg\iota) \rightarrow \nu$ then return *NO;*
    if *is WellFormed*($t[]$, $\gamma^*(\mu)$) and *SMT*($\rho \wedge \neg\underline{\iota}$) $\rightarrow$ *UNSAT* then return *YES;*
if $t[] \in \tau_-$ then
    if *SMT*($\underline{\iota} \wedge \neg\rho$) $\rightarrow \nu$ then return *NO*;
    if *is WellFormed*($t[]$, $\gamma^*(\mu)$) and *SMT*($\overline{\iota} \wedge \neg\rho$) $\rightarrow$ *UNSAT* then return *YES*;
return *YES or NO randomly*

Algorithm 1: 소속 질문에 답하는 알고리즘

(* $\underline{\iota}$: an under-approximation; $\bar{\iota}$: an over-approximation *)
(* $t[]$: the given template *)
Input : $\beta \in Bool_{B(A)}$
Output : *YES*, or a counterexample
$\rho := \gamma_\tau(\beta,\ t[])$;
if ρ *is an invariant weaker than* $\underline{\iota}$ *and stronger than* $\bar{\iota}$ then return YES;
if $SMT(\underline{\iota} \wedge \neg\rho) \rightarrow \nu$ then return $\alpha^*(\nu)$;
if $SMT(\rho \wedge \neg\iota) \rightarrow \nu$ then return $\alpha^*(\nu)$;
let $SMT(\rho \wedge \neg\iota) \rightarrow \nu_0$ and $SMT(\bar{\iota} \wedge \neg\rho) \rightarrow \nu_1$;
return $\alpha^*(\nu_0)$ *or* $\alpha^*(\nu_1)$ *randomly*

Algorithm 2: 동치 질문에 답하는 알고리즘

결정되는 극성에 따라서 두가지 경우를 고려한다. 거푸집 $t[]$가 문법적으로 양극인 경우를 가정하자. 만약 $\neg(\gamma_\tau^*(\mu,\ t[]) \Rightarrow \bar{\iota})$이라면, 알고리즘은 "아니요"를 반환하게 된다. 만약 거푸집 $t[]$가 $\gamma^*(\mu)$에 대해서 적합한 거푸집이고, $\gamma_\tau^*(\mu,\ t[]) \Rightarrow \underline{\iota}$가 만족된다면, 알고리즘은 "예"를 반환한다.

If $t[]$ is well-formed with respect to $\gamma^*(\mu)$, and $\gamma_\tau^*(\mu,\ t[]) \Rightarrow \underline{\iota}$, then the algorithm returns *YES*.

거푸집 $t[]$이 문법적으로 음극이라면, 알고리즘 1은 소속 질문에 대하여 대칭적으로 양극인 경우와 동일하게 대답한다. 알고리즘 1은 소속 질문이 위의 방법으로 답해질 수 없는 경우에는 랜덤한 답변(random answer)을 하게 됨을 주목하자.

4.3.2 *동치 질문.* 동치 질문 $EQ(\beta)$에 대해서 우리는 $\gamma_\tau(\beta,\ t[])$이 반복문의 불변식인지를 검사한다. 만약 이것이 불변식이라면, 우리는 목표는 달성되었다. 만약 이것이 불변식이 아니라면, 동치 질문에 답하는 알고리즘은 $\gamma_\tau(\beta,\ t[])$와 불변식의 근사값들을 비교하여 λ와 β를 구분해줄 수 있는 반례를 찾게 된다.

알고리즘 2은 동치 질문에 대답하는 알고리즘을 제공한다. 이 알고리즘은 우선 $\gamma_\tau(\beta,\ t[])$ 이 반복문에 대한 불변식인지를 SMT solver를 이용하여서 세가지 식 $\underline{\iota} \Rightarrow \rho$, $\rho \Rightarrow \bar{\iota}$, 그리고 $\kappa \wedge \rho \Rightarrow Pre(\rho,\ S)$이 성립하는지를 검사함으로써 알아낸다. 만약 그렇지 않다면 이 알고리즘은 $\underline{\iota} \Rightarrow \gamma_\tau(\beta,\ t[])$을 거짓으로 만드는 값할당을 계산해낸다. 이 값할당은 β와 λ를 다르게 평가하게 하는 반례를 동치 질문에 대한 답의 일부로 제공한다.

만약 이러한 반례들을 찾는데 실패한다면, 이 알고리즘은 $\gamma_\tau(\beta,\ t[])$을 불변식의 근사값들로부터 구별짓는 값할당을 랜덤한 답변으로 반환한다. 동치 질문과 소속 질문들에 대한 답변들이 상호 독립적으로 만들어지기 때문에, 두 종류의 답변들 사이에 모순이 발생할 수 있다. 이러한 경우에 우리의 불변식 생성 알고리즘은 단순히 다시 시작한다.

4.4 메인 반복문

우리의 불변식 생성 알고리즘은 이제 복잡하지 않다. 이전조건과 이후 조건이 덧붙여진 반복문 $\{\delta\}$ while κ do S $\{\epsilon\}$ 거푸집 $t[] \in \tau$에 대해서 우리는 작은 근사값 δ와 큰 근사값 $\kappa \vee \epsilon$을 이용하여 CDNF 알고리즘으로부터 생성되는 질문들에 답하게 된다. 만약 CDNF 알고리즘이 이진식 $\lambda \in Bool_{B(A)}$을 추론한다면, 1차 논리식

Input : $\{\delta\}$ while κ do S $\{\epsilon\}$ an annotated loop; $t[]$: a template
Output : an invariant in the form of $t[]$
$\underline{\iota} := \delta$;
$\overline{\iota} := \kappa \vee \epsilon$;
repeat
  try $\lambda :=$ call CDNF with Algorithm 1 and 2 when abort $\rightarrow$ continue
until λ *is defied* ;
return $\gamma_\tau(\lambda, t[])$;

Algorithm 3: 메인 반복문

$\gamma_\tau(\lambda, t[])$은 주어진 반복문에 대하여 거푸집 $t[]$의 형태의 불변식이게 된다(알고리즘 3).

CDNF 알고리즘은 소속 질문에 답하는 알고리즘(알고리즘 1)과 동치 질문에 답하는 알고리즘(Algorithm 2)을 이용하여서 질문들에 답한다. 질문이 확실하게 해결되지 못할 때에, 우리의 질문에 답하는 알고리즘은 랜덤한 답변을 할 수도 있다. 동치 질문에 답하는 알고리즘에서 SMT solver를 이용하여서 찾아낸 1차식이 실제 불변식인지를 검증하기 때문에 랜덤한 답변들은 부정확한 결과들을 만들어내지 않게 된다. 반면에, 랜덤한 답변들 은 우리의 알고리즘으로 하여금 다양한 불변식들을 탐색하도록 도와준다. 만약 수많은 1 차논리 불변식들이 주어진 거푸집의 모양으로 존재한다면, 몇개의 랜덤한 답변들은 우리의 알고리즘으로 하여금 그것들 중의 하나를 찾는 것을 방해하지 못한다. 실제로 우리의 실험 결과는 우리의 간단한 랜덤 알고리즘이 서로 다른 1차논리 불변식들을 서로 다른 실행에서 찾아냄을 보여주고 있다. 그러나 우리의 알고리즘은 너무도 많은 틀린 랜덤한 답변들로 인한 충돌 때문에 실패할 수도 있다. 그러한 경우에는 전체 알고리즘이 다시 시작된다.

case	Template	AP	MEM	EQ	MEM_R	EQ_R	$RESTART$	Time (sec)
max	$\forall k.[]$	7	622	215	613	46	50	1.02
selection sort	$\forall k_1.\exists k_2.[]$	6	7594	4762	4577	243	1845	5.21
rm pkey	$\forall k.[]$	8	2946	1268	2036	253	167	3.40
tracepoint1	$\exists k.[]$	4	62060	65410	62060	7205	18327	5.06
tracepoint2	$\forall k_1.\exists k_2.[]$	7	323611	128334	323611	5207	31256	317.78

[표 I]성능 지표들.

AP : 단위 명제의 개수, MEM : 소속 질문의 횟수, EQ : 동치 질문의 횟수,
MEM_R : 랜덤하게 답변된 소속 질문의 개수, EQ_R : 랜덤하게 답변된 동치 질문의 개수,
$RESTART$: CDNF 알고리즘이 호출된 횟수.

$\{i = 0 \wedge \neg \text{ret}\ \}$
1 while i<n$\wedge\neg ret$ do
2 if $tbl[i] = addr$ then
3 $tbl[i]$: = 0; ret:=true
4 else
5 i:=i+1
6 end
$\{(\neg ret \Rightarrow \forall k.\ k<n \Rightarrow tbl[k] \neq addr) \wedge (ret \Rightarrow tbl[i] = 0)\}$

[그림 3] 리눅스 라이브러리에서 추출한 devres

5. 실험 결과

우리는 이 논문에서 제시한 알고리즘들을 OCaml 언어를 이용하여 구현하였다[3] 우리는 구현에서 Yices를 SMT solver를 질문들에 답하기 위하여 이용하였다(알고리즘 1, 2).

[표 1]은 우리 실험의 성능을 보여준다. 우리는 [35]으로 부터 2개의 예제를 동일한 이전/ 이후 조건과 함께 가지고 와서 실험하였다. 또한 4개의 for문을 리눅스 2.6.28에서 가지고 와서 실험하였다. 우리는 각각의 반복문를 우리의 언어로 변환하고 이전/이후 조건들을 수작업을 덧붙였다. devres는 라이브러리에서, tracepoint1와 tracepoint2는 커널에서, 그리고 rm_pkey은 Infi iBand 디바이스 드라이버에서 찾을 수 있다. 실험 결과는 500번의 수행을 평균 낸 결과이다. 2.66GH Intel Core2 Quad CPU와 8GB 메모리를 가지고 리눅스 2.6.28을 OS로 이용하는 컴퓨터를 이용하여 실험하였다.

5.1 devres

[그림 3]은 리눅스 라이브러리에서 추출한 이전/이후 조건이 덧붙여진 반복문를 보여준다.[4] 이후 조건에서, 우리는 ret이면 $tbl[i] = 0$임을 의미하고, 그렇지 않다면 $tbl[]$의 모든 원소들이 addr와 같지 않음을 명시하고 있다. 단위 명제의 집합인 $\{tbl[k]$= addr, $i<n$, $i = n$, $k<i$, $tbl[i] = 0$, ret$\}$과 단순한 거푸집인 $\forall k.[]$을 이용하여서 우리의 알고리즘은 다음과 같은 정량자를 포함한 불변식을 찾아낸다:

$$\forall k.(k<i \Rightarrow tbl[k] \neq addr) \wedge (ret \Rightarrow tbl[i]=0).$$

우리의 알고리즘은 주어진 거푸집의 빈 공간을 채우기 위하여 기초 산술식들로부터 조립된 임의의 조건식을 추론할 수 있음을 관찰할 수 있다. $\forall k.[]$과 같은 단순한 거푸집만으로도 우리의 방법에 충분한 힌트가 될 수 있다.

5.2 selection sort [35]

[그림 4]의 선택 정렬 (selection sort) 알고리즘을 살펴보자. '$a[]$를 알고리즘이 수행되기 이전의 배열 $a[]$의 내용을 나타낸다고 하자. 이후 조건은 알고리즘이 수행된 이후의 배열 $a[]$이 수행 이전의 배열들의 순서만을 뒤섞은 것임을 명시하고 있다. 이 예제에서, 우리는 우리의 불변식 생성 알고리즘을 바깥쪽 반복문에 대한 불변식을 찾기 위해서 사용하였다. 이를 위하여 우리는 안쪽 반복문에 대한 명세를 사용하였다.

3) 다음 위치에서 찾아볼 수 있음: http://ropas.snu.ac.kr/cav10/qinv-learn-released.tar.gz
4) 이 소스 코드는 리눅스 2.6.28의 lib/devres.c 파일의 devres 함수에서 찾을 수 있다.

$\{i = 0\}$
1 while $i<n-1$ do
2 *min* := i;
3 j: = i+1;
4 while $j<n$ do
5 if $a[j] < a[min]$ then *min* := j
6 j: = j+1;
7 if $i \neq min$ then
8 *tmp* : = $a[i]$; $a[i]$: = $a[min]$; $a[min]$: = *tmp*;
9 i: = i+1;
$\{i \geq (n-1) \wedge \forall k_1.k_1 < n \Rightarrow \exists k_2.k_2 < n \wedge a[k_1] = 'a[k_2]\}$

[그림 4] selection sort [35]

우리는 다음과 같은 단위 명제의 집합을 사용한다: $\{k_1 \geq 0$, $k_1 < i$, $k_1 = i$, $k_2 < n$, $k_2 = n$, a $[k_1] = 'a[k_2]$, $i<n-1$, i=min$\}$. 거푸집 $\forall k_1.\exists k_2.[]$을 이용하여서 우리는 다음과 같은 불변식을 찾아낸다:

$$\forall k_1.(\exists k_2.[(k_2<n \wedge a[k_1]='a[k_2]) \vee k_1 \geq i]).$$

거푸집은 우리로 하여금 단순히 전체 정량자(universal quantifir)를 포함하는 불변식만을 추론하도록 하는 것이 아니라, 정량자들이 교차하는 1차논리 불변식에 대한 추론을 가능하게 한다. 기초 조건식들로 부터 구성되는 임의의 조건식을 추론할 수 있다는 사실이이 예제에서 사용하는 거푸집의 모양을 단순화하는데 큰 도움을 준다.

5.3 tracepoint1

[그림 5]은 리눅스 커널에서 추출한 반복문를 보여준다.[5] 반복문 몸체로부터 우리는 이후 조건을 추론할 수 있다. 만약 세 번째 줄의 if 문의 조건이 적어도 한번이라도 참이었고, p가 0이 아니라면, 이것은 분기 조건의 두번째 선언지 (disjunct)인 $A[i] = p$가 반드시 참이어야함을 의미한다. 이것은 $A[k] = p$를 만족하는 어떤 k값이 반드시 존재해야함을 의미한다. 이 이후 조건으로부터, 우리는 불변식이 k에 대한 존재 정량자를 포함해야한다는 것을 알 수 있다. 그래서 우리는 거푸집 $\exists k.[]$을 이용한다. 이 거푸집과 기초 조건식의 집합 $\{p = 0$, $n<0$, $k<i$, $A[k]=p\}$을 이용하여 우리의 알고리즘은 다음의 불변식을 찾아낸다:

$$\exists k.(p = 0) \vee (k<n \wedge A[k] = p) \vee (n<=0).$$

$\{n = 0 \wedge i = 0\}$
1 while $A[i] \neq 0$ do
2 if $(p = 0 \vee A[i] = p)$ then
3 n : = n+1;
4 i: = i+1;
$\{n>0 \wedge p \neq 0 \Rightarrow \exists k.k < i \wedge A[k] = p\}$

[그림 5] 리눅스 커널에서 추출한 tracepoint 1

5) 이 소스 코드는 리눅스 2.6.28의 kernel/tracepoint.c 파일의 tracepoint entry remove probe 함수에서 찾을 수 있다.

$\{i = 0 \wedge j = 0\}$
1 while $old[i] \neq 0$ do
2 if $(probe \wedge old[i] \neq probe)$ then
3 $new[j] := old[i]$;
4 $j := j+1$;
5 $i := i+1$;
$\{\forall k_2.k_2 < j \Rightarrow \exists k_1.k_1 < i \wedge old[k_1] = new[k_2] \wedge new[k_2] \neq probe\}$

[그림 6] 리눅스 커널에서 추출한 tracepoint 2

5.4 tracepoint 2

[그림 6]은 리눅스 커널에서 찾아낸 또 다른 반복문이다.[6] 반복문 몸체로부터 우리는 다음의 이후 조건을 찾아낼 수 있다. 변수 j의 값은 오직 프로그램이 세 번째 줄의 if이 참일 때에만 증가되게 된다. 따라서 반복문의 첫부분에서 new의 모든 원소들인 new [0] ... $new[j-1]$은 각각에 해당하는 *old* 변수의 값인 $old[0]$ ... $old[i-1]$과 같게 된다(세번째 줄). 이 이후 조건으로부터, 우리는 불변식이 $\forall k_1.\exists k_2.[]$의 형태임을 알 수 있다. 이 거푸집과 단위 명제의 집합 $\{old[i] = 0$, $old[i] = probe$, $k<i$, $k<j$, $old[k] = p$, $new[k] = p$, $old[k] = new[k]\}$을 이용하여서 우리의 알고리즘은 다음의 불변식을 찾아낸다:

$$\forall k_2.\exists k_1(k_2<j) \Rightarrow (k_1<i) \wedge (old[k_1]=\text{new}[k_2]) \wedge (old[k_1] \neq probe).$$

5.5 rm_pkey

[그림 7]은 리눅스 InfiiBand 디바이스 드라이버에서 추출해낸 while 문이다.[7]

이후 조건 P의 논리곱 인자들(conjuncts)은 각각 다음을 나타낸다. (1) 만약 반복문이 break 없이 종료된다면, *pkeys*의 모든 원소들은 *key*와 같지 않다(두번째 줄); (2) 만약 반복문이 *break*로 종료되었지만 *ret*가 거짓이라면 $pkeys[i]$는 *key*와 같다(두번째 줄). 하지만 $pkeyrefs[i]$는 0이 아니다(네번째 줄); (3) 만약 반복문 종료 후에 *ret* 가 참이라면 $pkeyrefs[i]$와 $pkeys[i]$가 0과 같다(네번째, 다섯 번째 줄).

이 이후 조건으로부터, 우리는 불변식이 k에 대한 전체 정량자(universal quantifir)를 포 함해야한다는 것을 알 수 있다. 단순한 거푸집인 $\forall k.[]$과 단위 명제의 집합인 {ret, break, $i < n$, $k < i$, $pkeys[i] = 0$, $pkeys[i] = key$, $pkeyrefs[i] = 0$, $pkeyrefs[k] = key$}을 이용하여서 우리의 알고리즘은 다음의 정량자를 포함한 불변식을 찾아낸다:

$$(\forall k.(k<i) \Rightarrow pkeys[k] \neq key) \wedge (\neg ret \wedge \text{break} \Rightarrow pkeys[i] = key \wedge pkeyrefs[i] = 0) \wedge (ret \Rightarrow pkeyrefs[i] = 0 \wedge pkeys[i] = 0)$$

학습기반의 우리의 방법의 일반성은 이 예제를 통해서도 다시 관찰된다. 우리의 알고리즘 은 정량화된 불변식을 유저로부터의 작은 도움을 통해서 추론할 수 있다. 우리의 해결책은 다른 거푸집 기반 방법들에 비하여 실제의 문제를 해결하는 데에 보다 적합하다.

6) 이 소스 코드는 리눅스 2.6.28의 kernel/tracepoint.c 파일의 tracepoint entry remove probe 함수에서 찾을 수 있다.
7) 이 소스 코드는 리눅스 2.6.28의 drivers/infiniband/hw/ ipath/ipath mad.c 파일의 rm pkey 함수에서 찾을 수 있다.

$\{i = 0 \wedge key \neq 0 \wedge \neg ret \wedge \neg break\}$
1 while($i < n \wedge \neg break$) do
2 if($pkeys[i] = key$) then
3 $pkeyrefs[i] := pkeyrefs[i] - 1$;
4 if($pkeyrefs[i] = 0$) then
5 $pkeys[i] := 0$; $ret := true$
6 $break := true$
7 else $i := i + 1$;
8 done
$\{ (\neg ret \wedge \neg break \Rightarrow (\forall k.(k < n) \Rightarrow pkeys[k] \neq key))$
$\wedge (\neg ret \wedge break \Rightarrow pkeys[i] = key \wedge pkeyrefs[i] \neq 0$
$\wedge (ret \Rightarrow pkeyrefs[i] = 0 \wedge pkeys[i] = 0) \}$

[그림 7] 리눅스의 InfiiBand 디바이스 드라이버에서 추출한 rm_pkey

6. 관련 연구

거푸집에 기반한 기존의 연구들 [35, 21] 과는 다르게 우리의 거푸집은 빈 공간을 채우는 조건식에 대한 제약 조건이 없다는 점에서 보다 일반적이다. [35]에 소개된 기술은 주어진 집합으로부터 논리곱으로 조합되는 조건식만을 빈 공간에 허용하는 거푸집만을 다루고 있고, 따라서 논리합은 반드시 거푸집에 의해서 명시적으로 기술되어야만 한다. Gulwani 등은 [21] 정량자가 없는 E, F_j and e_j에 대해서 불변식의 형태를 $E \wedge \wedge_{j=1}^{n} \forall U_j(F_j \Rightarrow e_j)$로 제한하였다.

기존의 기술들은 우리의 프레임웍을 보다 유용하게 만들어준다. 우선 우리 기술의 완결성은 강력한 참거짓 자동 판별기들 [15, 17, 36] 과 정리 증명기들 [33, 7, 34]에 의해서 향상될 수 있다. 또한 우리의 접근 방식은 기존의 불변식 생성 기술들을이용하여서 보다 정확한 근사값들을 제공해줌으로써 속도 향상을 꾀할 수 있다. Gupta 등은 [24] InvGen 이라는 도구를 개발하였다. 이 도구는 프로그램의 불변식을 만족하는 모든 도달 가능한 상태들을 모은다. 또한 효율적인 불변식 생성을 위하여 이러한 불변식들의 모음을 계산한다. 이러한 결과들은 우리의 접근 방식에서 작은 근사값과 큰 근사값으로 각각 이용될 수 있다.

정량자를 포함하지 않는 불변식의 생성에 관해서는 Gulwani 등이 [22] 제약조건식 분석에 기반한 접근 방식을 제안하였다. InvGen [24] 이 선형 계산식으로 이루어진 불변식에 대한 효율적인 생성 방법을 제시하였다면, 선형 계산 이론과 비해석 함수 이론을 결합한 이론에 대한 불변식은 [8]에서 합성되었다. 정량자를 포함하는 반복문 불변식의 생성에 관한 연구에서는 Flanagan 등이 [16] 전체 정량자(universal quantifir)에 대하여 안전하게 정량자를 제거하는 기법(skolemization)을 이용되었다. [33]에서는 도출 원리(paramodulation)에 기반하여 새로운 사실이 도출되지 않을 때까지 유도하는 증명기(saturation prover)를 전체 정량자에 대해서 완결성을 지니는 중간값을 채워나가는 증명기(interpolating prover)로 확장하였다. 우리가 제안한 기술과 다르게, 다른 접근 방법들은 [16, 33] 오직 전체 정량자(universal quantifir)를 포함하는 불변식만을 생성할 수 있다. 배열의 내용에 대한 속성을 분석하는 것에 관해서는 Halbwachs 등 [25]이 순차적인 방문을 하거나, 매 반복마다 배열의인덱스 값을 증가시키거나 혹은 감소/증가시키거나, 배열을 간단한 식을

인덱스로 접근하는 프로그램들에 대하여 연구하였다. 다차원 배열을 탐색하는 반복문에 대한 속성을 생성하는 방법은 [26]에 소개되어 있다. 범위 조건식을 추론하는 것에 대하여 Jhala와 Mcmillan[27]은 수행 불가능한 반례가 되는 경로들을 생성하는 프 레임웍을 기술하였다. 이 방법의 결점으로써, 증명기는 짧은 경로를 반박하지만 그렇다고 보다 긴 경로들로 일반화시킬 수 없는 증명들을 발견할 수도 있다. 이러한 문제로 인하여 이 방법 [27]은 삽입 정렬이 배열을 올바르게 정렬한다는 것을 증명하지 못한다.

7. 결 론

알고리즘적인 학습, 참거짓 자동판별기, 조건식 요약, 그리고 거푸집을 결합하여 우리는 정량자를 포함하는 불변식을 생성해내는 기술을 제시하였다. 새로운 기술은 주어진 거푸 집 모양의 불변식을 질문에 답하는 알고리즘들을 통하여 찾아낸다. 알고리즘적인 학습은 불변식 생성에 필요한 다양한 기술들을 통합할 수 있는 기반을 마련해준다. 정량자를 포함하는 불변식에 대한 단순한 거푸집만으로도 기존의 기술들과 함께 알고리즘적인 학습이 만들어내는 질문들을 대답하는 새로운 알고리즘들을 설계할 수 있었다.

우리의 기술은 많은 불변식들에 대한 알고리즘적인 학습의유연함을 실제 세계에서 사용되는 코드들에 대한 정량자를 포함하는 불변식들을 통해서 보여준다. 우리는 이러한 유연성을 랜덤한 방식으로 질문에 답하는 알고리즘을 통하여 이용하였다. 질문이 결정적인 방법으로 답해질 수 없을 때 랜덤한 답변이 사용된다. 학습 알고리즘이 어떤 특정한 불변 식을 목표로 하고 있지 않기 때문에, 이것은 유연하게 현재까지의 질문들에 대한 답변들과 모순되지 않는 해결책을 찾아낸다. 우리의 실험은 알고리즘적인 학습이자명하지 않은 정량자를 포함하는 불변식을 간단한 무작위적인 알고리즘을 통하여서 찾을 수 있음을 보여주었다. 실험에서 거푸집들은 단지 어떤 변수들이 정량화되었는지를 나타내는 데에만 필요하였다.

참 고 문 헌

[1] Alur, R., Cerný, P., Madhusudan, P., Nam, W.: Synthesis of interface specifiations for java classes. In: POPL, ACM (2005) 98–109

[2] Alur, R., Madhusudan, P., Nam, W.: Symbolic compositional verifiation by learning assumptions. In: CAV. Volume 3576 of LNCS., Springer (2005) 548–562

[3] Alur, R., Černỳ, P., Madhusudan, P., Nam, W.: Synthesis of interface specifiations for java classes. In: POPL, New York, NY, USA, ACM (2005) 98–109

[4] Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2) (1987) 87–106

[5] Balaban, I., Pnueli, A., Zuck, L.: Shape analysis by predicate abstraction. In: VMCAI. Volume 3385 of LNCS., Springer (2005)

[6] Ball, T., Cook, B., Das, S., Rajamani, S.K.: Refiing approximations in software predabstraction. In: TACAS. Volume 2988 of LNCS., Springer (2004) 388–403

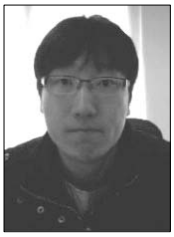
[7] Bertot, Y., Castéeran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. Springer Verlag (2004)

[8] Beyer, D., Henzinger, T.A., Majumdar, R., Rybalchenko, A.: Invariant synthesis for combined theories. In: VMCAI. (2007) 378–394

[9] Bshouty, N.H.: Exact learning boolean functions via the monotone theory. Informa-

tion and Computation 123 (1995) 146–153
[10] Chen, Y.F., Farzan, A., Clarke, E.M., Tsay, Y.K., Wang, B.Y.: Learning minimal separating DFA's for compositional verifiation. In: TACAS. Volume 5505 of LNCS., Springer (2009) 31–45
[11] Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abre fiement. In: CAV. Volume 1855 of LNCS., Springer (2000) 154–169
[12] Cobleigh, J.M., Giannakopoulou, D., Păsăreanu, C.S.: Learning assumptions for comverifiation. In: TACAS. Volume 2619 of LNCS., Springer (2003) 331–346
[13] Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: POPL, ACM (1978) 84–96
[14] David, C., Jung, Y., Kong, S., Bow-Yaw, W., Yi, K.: Inferring quantifid invariants via algorithmic learning, decision procedure, and predicate abstraction. Technical MemoROSAEC-2010-007, Research On Software Analysis for Error-Free Computing (2010)
[15] Dutertre, B., Moura, L.D.: The Yices SMT solver. Technical report, SRI International (2006)
[16] Flanagan, C., Qadeer, S.: Predicate abstraction for software verifiation. In: POPL, ACM (2002) 191–202
[17] Ge, Y., Moura, L.: Complete instantiation for quantifid formulas in satisfibiliby modulo theories. In: CAV '09: Proceedings of the 21st International Conference on Computer Aided Verifiation, Berlin, Heidelberg, Springer-Verlag (2009) 306–320
[18] Graf, S., Saïdi, H.: Construction of abstract state graphs with pvs. In: CAV. Volume 1254 of LNCS., Springer (1997) 72–83
[19] Gulwani, S., Jain, S., Koskinen, E.: Control-flw refiement and progress invariants for bound analysis. In: PLDI, ACM (2009) 375–385
[20] Gulwani, S., Jojic, N.: Program verifiation as probabilistic inference. In: POPL, ACM (2007) 277–289
[21] Gulwani, S., McCloskey, B., Tiwari, A.: Lifting abstract interpreters to quantifid logical domains. In: POPL, ACM (2008) 235–246
[22] Gulwani, S., Srivastava, S., Venkatesan, R.: Constraint-based invariant inference over predicate abstraction. In: VMCAI. Volume 5403 of LNCS., Springer (2009) 120–135
[23] Gupta, A., McMillan, K.L., Fu, Z.: Automated assumption generation for compositional verifiation. In: CAV. Volume 4590 of LNCS., Springer (2007) 420–432
[24] Gupta, A., Rybalchenko, A.: Invgen: An effiient invariant generator. In: CAV. Volume 5643 of LNCS., Springer (2009) 634–640
[25] Halbwachs, N., Péron, M.: Discovering properties about arrays in simple programs. In: PLDI. (2008) 339–348
[26] Henzinger, T.A., Hottelier, T., Kovács, L., Voronkov, A.: Invariant and type inference for matrices. In: VMCAI. (2010) 163–179
[27] Jhala, R., Mcmillan, K.L.: Array abstractions from proofs. In: CAV, volume 4590 of LNCS, Springer (2007)
[28] Jung, Y., Kong, S., Wang, B.Y., Yi, K.: Deriving invariants in propositional logic by algorithmic learning, decision procedure, and predicate abstraction. In: VMCAI. LNCS, Springer (2010)
[29] Kovács, L., Voronkov, A.: Finding loop invariants for programs over arrays using a theorem prover. In: FASE. LNCS, Springer (2009) 470–485
[30] Kroening, D., Strichman, O.: Decision Procedures - an algorithmic point of view. EATCS. Springer (2008)

[31] Lahiri, S.K., Bryant, R.E., Bryant, A.E.: Constructing quantifid invariants via predIabstraction. In: VMCAI. Volume 2937 of LNCS., Springer (2004) 267–281
[32] Lahiri, S.K., Bryant, R.E., Bryant, A.E., Cook, B.: A symbolic approach to predicate abstraction. In: CAV. Volume 2715 of LNCS., Springer (2003) 141–153
[33] McMillan, K.L.: Quantifid invariant generation using an interpolating saturation prover. In: TACAS, Springer (2008) 413–427
[34] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic. Volume 2283 of LNCS. Springer (2002)
[35] Srivastava, S., Gulwani, S.: Program verifi ation using templates over predicate abIn: PLDI, ACM (2009) 223–234
[36] Srivastava, S., Gulwani, S., Foster, J.S.: Vs3: Smt solvers for program verifiation. In: CAV '09: Proceedings of Computer Aided Verifiation 2009. (2009)



정 영 범

- 1998-2003 서울대학교 학사
- 2004-현재 서울대학교 석박사통합과정

<관심분야> 프로그램 분석 및 검증

공 순 호

- 2000-2007 서울대학교 학사
- 2007-2009 서울대학교 석사
- 2009-현재 소프트웨어 무결점 연구센터 연구원

<관심분야> 프로그램 분석 및 검증

이 광 근

- 1983-1987 서울대학교 학사
- 1988-1990 Univ. of Illinois at Urbana-Champaign 석사
- 1990-1993 Univ. of Illinois at Urbana-Champaign 박사
- 1993-1995 Bell Labs., Murray Hill 연구원
- 1995-2003 한국과학기술원 교수
- 2003-현재 서울대학교교수 <관심분야> 프로그램 분석 및 검증, 프로그래밍 언어

논문 23-1-R02

Dynamic-Partitioned SIMD 기법을 사용한 Reconfigurable Processor와 이를 위한 컴파일러 설계

Design of Reconfigurable Processor using Dynamic-Partitioned SIMD and Compiler

권용인 · 윤종희 · 이종원 · 김용주 · 백윤흥

서울대학교 전기 · 컴퓨터공학부 전기공학전공

yikwon@optimizer.snu.ac.kr, jhyoun@optimizer.snu.ac.kr, jwlee@optimizer.snu.ac.kr
yjkim@optimizer.snu.ac.kr, ypaek@snu.ac.kr

요 약

최근 다양한 응용프로그램을 위한 임베디드 프로세서가 많이 개발 되고 있다. 또한 반도체 기술의 발달과 새로운 응용프로그램들에서의 요구로 인해서 하나의 칩에서 수행되어야 하는 작업들이 더욱 많아지고 있다. 전통적으로 DSP와 ASIP은 코드의 성능을 높이기 위하여 손으로 assembly 작성을 많이 해왔다. 그러나 응용프로그램들의 복잡성과 time-to-market 제약조건 때문에 임베디드 시스템 개발자들은 High Level Language와 컴파일러를 이용하여 코드를 생성함으로써 직접 손으로 assembly 코드를 만드는 과정에서 오는 부담을 줄이기를 원하고 있다.

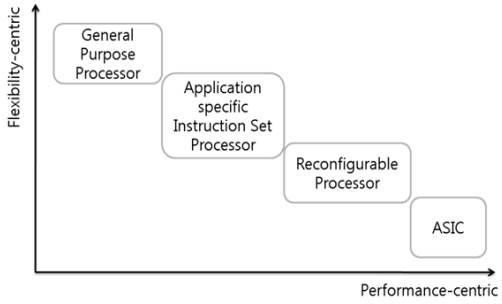
한 편 Multimedia Processing을 위한 Reconfigurable Processor ERP는 DP-SIMD(Dynamic-Partitioned Single Insturction Multiple Data)를 사용한다. DP-SIMD기법은 Reconfigurable Processor의 Processing Unit들이 동시에 네 가지의 Instruction 중 하나를 선택하여 수행할 수 있게 함으로써 SIMD(Single Instruction Multiple Data)에 비해 Operation Efficiency를 높임으로서 H.264의 성능을 향상시켰다.

하지만 DP-SIMD를 활용하기 위한 assembly 코딩 작업은 많은 노력과 시간을 필요로 하기 때문에 컴파일러의 필요성이 증대되었다. 이에 대해 본 논문은 ERP를 위한 컴파일러를 개발 설계하고, H.264의 de-blocking filter(DF), intra prediction(IP)를 실험하였다. 실험결과 기존 SIMD Compiler에 비해 DF의 경우 1.48배, IP의 경우 1.83배 성능이 향상되었음을 알 수 있었다.

1. 서 론

오늘날, H.264는 비디오 압축 기술로 널리 사용되고 있다. 점점 더 높은 성능의 비디오 압축 기술이 필요해지고 있고, 따라서 H.264를 위한 다양한 하드웨어가 개발되고 있다[1-8]. 그 중 하나로 Application-specific integrated circuit(ASIC)을 사용하여 H.264의 일부분이나 전체가 개발되고 있다. ASIC을 사용하여 개발된 H.264 encoder와 decoder는 매우 높은 성능을 보여주고 있으나 수정이 필요할 때 flexibility에 제한이 있다는 단점이 있다. 반면, Application specific in-

struction set processor(ASIP)과 Configurable processor는 변화에 대해 높은 flexibility를 가지고 있으나 성능이 ASIC에 비해 떨어진다. General Purpose Processor 역시 다양한 시스템에 적용될 수 있도록 flexibility가 높지만 ASIC과 같은 성능을 내기 위해서는 하드웨어 비용이 높아지고, 발열량과 전력소모량이 커지며 그에 따라 크기가 커지기 때문에 휴대성 또한 떨어지게 된다. 반면 Reconfigurable Processor는 General Purpose Processor와 ASIC이 가진 장점인 높은 성능과 높은 flexibility 모두를 지니고 있다. [그림 1]은 Performance와 Flexibility 사이의 Tradeoff를 나타내고 있다.

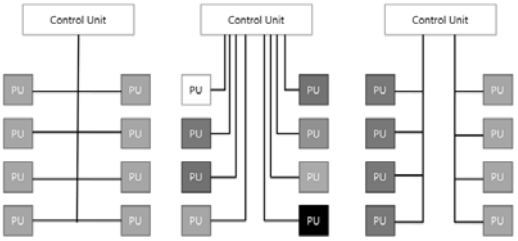


[그림 1] tradeoff between flexibility and performance

또한 이렇게 다양한 하드웨어가 개발되는 만큼 어플리케이션 개발속도도 중요하다. 특히 컴파일러를 이용하여 어플리케이션을 개발할 경우 개발기간을 크게 단축시킬 수 있으리라 본다. 이 논문은 DP-SIMD 프로세서의 특징과 Soargen을 사용하여 컴파일러를 생성하는 과정을 보여주고, 실험을 통해 그 성능측정을 하였다.

2. Dynamic-Partitioned SIMD 아키텍쳐

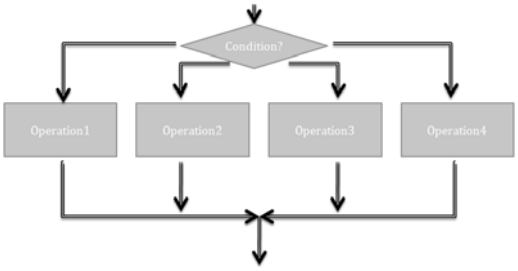
Reconfigurable Processor는 보통 성능향상을 위해 Array Architecture를 가지고 있고, 효율을 높이기 위해서 Array를 현명하게 컨트롤 해야할 필요성이 있다. Reconfigurable Array를 컨트롤하는 방법을 크게 두가지로 나눌 수 있다. 첫 번째는 Single instruction multiple data(SIMD)이고 두 번째는 Multiple instruction multiple data(MIMD) 이다. MIMD 컨트롤 방식은 각 Processing Unit에 다른 Processing을 수행하게 하여 Processing 성능을 향상시킬 수 있다. 하지만 이 방식은 Command가 길어진다는 단점을 갖고 있으며, 이로 인해 프로그램 메모리에 부담이 커지게 된다. 반면에 SIMD 컨트롤 방식은 각 Processing Unit이 동일한 Processing을 수행하게 하는 방식이다. 따라서 MIMD 컨트롤 방식이 가진 단점을 해소할 수는 있지만 성능은 MIMD에 비해 낮아지게 된다. SIMD의 단점을 개선하기 위해 Partitioned SIMD(P-SIMD) 컨트롤 방식이 제안되었고, P-SIMD는 Processing Unit들을 여러개의 SIMD 그룹으로 나누어 그룹마다 서로다른 Instruction을 수행하도록 한다[9]. 하지만 FFT와 Integer Transform 등과 같은 여러 그래픽 알고리즘에서 뛰어난 성능을 보이지만 H.264의 De-blocking Filter와 같이 각각의 Processing Unit들이 Dynamic하게 그룹이 형성된다면 P-SIMD의 기능만으로 성능 향상을 기대하기 힘들다. 그래서 새롭게 제안된 Dynamic-Partitioned SIMD (DP-SIMD) 아키텍쳐는 기존 SIMD가 가진 문제점들을 해결하고 시스템 성능을 높일 수 있다[10].



[그림 2] SIMD vs MIMD vs P-SIMD

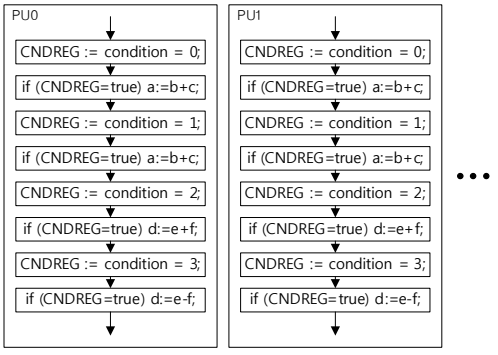
2.1. Dynamic-Partitioned SIMD의 Control Method

H.264는 Variable Length Decoding(VLD), Inverse Transform and Quantization(ITQ), Motion Compensation (MC, Intra Prediction(IP), De-blocking filter(DF)로 이루어져있다. 이 중 VLD는 병렬수행이 불가능하고 ITQ는 SIMD Control Method으로 충분히 성능을 낼 수 있다. 하지만 IP, MC, DF는 SIMD Control Method보다 MIMD Control Method를 사용할 경우 성능이 훨씬 더 좋아진다. 왜냐하면 SIMD Control Method를 사용하면 Conditional Operation이 많아져 Efficiency를 떨어뜨리기 때문이다.



[그림 3] Conditional Operation

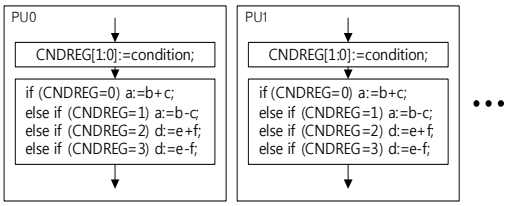
[그림 3]은 Conditional Operation의 예이다. Condition에 따라 4가지 다른 Operation을 수행해야 하는 경우를 나타내고 있다.



[그림 4] Conditional Operation for SIMD

모든 Processing Unit들은 같은 Operation을 수행해야하기 때문에 SIMD Control Method으로 Conditional Operation을 수행하기 위해서는 위 [그림 4]와 같은 코드가 수행되어야한다. 우선 flag register인 CNDREG가 설정되고 그 다음에 오는 Operation은 CNDREG에 따라 수행될지 말지를 결정한다. 그리고 다음 Conditional Operation들도 이와 같은 작업이 수행된다. 따라서 Conditional Operation이 존재 할때 SIMD Control Method은 시스템의 성능을 떨어뜨리게 된다. 반면에 MIMD Control Method은 Conditional Opeartion이 있을 때 병렬로 Operation을 수행함으로 성능을 향상시킬 수 있다.

한편, DP-SIMD에서는 Processing Unit들이 자신이 가지고 있는 데이터에 따라 Instruction을 선택할 수 있다. 따라서 병렬 프로세서들이 매번 다이나믹하게 Instruction을 선택 수행하게 되는 것이다.

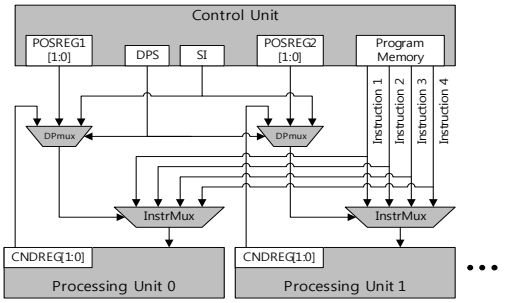


[그림 5] Conditional Operation for DP-SIMD

[그림 5]는 DP-SIMD Control Method에 의해 각 Processing Unit들이 수행하는 Operation을 보여주고 있다. 2bit의 CNDREG를 Condition에 따라 설정한 후, CNDREG에 따라 Processing Unit이 Instruction을 선택하여 수행하게 되어, [그림 3]의 SIMD Control Method에 비해 성능이 4배 빨라졌음을 알 수 있다. 이와 같이 한 번에 네 개의 인스트럭션 중 하나를 선택하여 수행해야할 경우, DP-SIMD Control Method를 사용하면 SIMD Control Method에 비해 최대 4배의 성능이 향상된다.

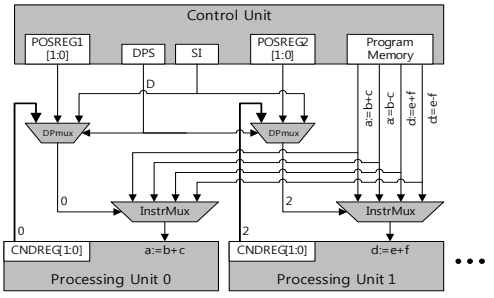
2.2. Dynamic-Partitioned SIMD 아키텍쳐

Processing Unit들에 DP-SIMD Control Method을 적용함으로써 낮은 Operation Efficiency 문제는 해결되었다. DP-SIMD Instruction은 여러개의 Instruction을 하나로 묶는다. Very Long Instruction word(VLIW)와 비슷하지만 VLIW와는 다르게 Function Block에 따라 순서가 정해져있지 않고 Processing Unit이 Condition에 따라 Instruction을 선택하여 수행하게 된다.



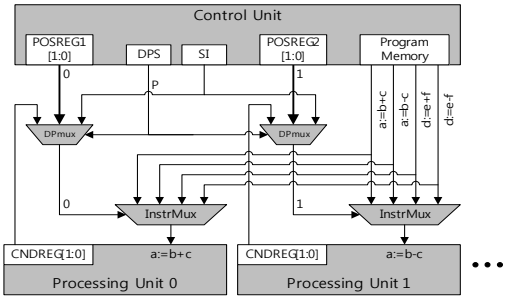
[그림 6] 프로세서의 구조

[그림 6]은 DP-SIMD Control Method을 사용하는 프로세서의 구조를 나타내고있다. 각 Processing Unit들은 병렬로 연결되어있고 Control Unit들은 Processing Unit들을 컨트롤한다. Control Unit은 Program Memory로부터 Instruction들을 읽어들여 Processing Unit으로 전달하고 Processing Unit은 Instruction들 중 하나를 선택하여 수행하게 된다.



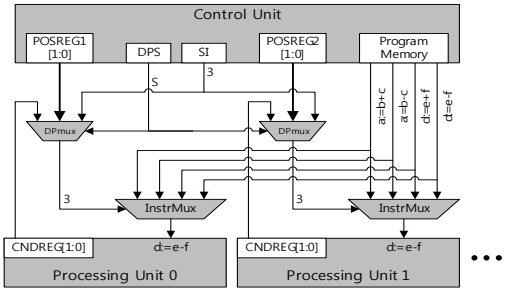
[그림 7] Data based partitioned SIMD control

DP-SIMD Control Method으로 Processing Unit을 컨트롤 하는 방법은 세가지가 있다. 첫번째로 Processing Unit에 의해 계산된 데이터를 바탕으로 Processing Unit을 컨트롤 하는 방법이다. 이를 'data based partitioned SIMD control'이라고 부른다. [그림 7]은 data based partitioned SIMD control를 나타내고 있다. 이와 같은 경우 conditional operation을 수행하기 위해 PU는 CNDREG Register에 값을 넣고 Control Unit은 DPS 시그널에 D를 인가하여 CNDREG의 값에 따라 Instruction 넷 중 하나를 선택하여 수행하게 한다.



[그림 8] Position based partitioned SIMD control

두 번째는 Control Unit에 의해 어느 Processing Unit이 어느 Instruction을 수행할 지가 미리 정해져있는 경우이다. 이를 'position based partitioned SIMD control'이라 부른다. [그림 8]은 예를 보여주고 있다. Control Unit에 의해 POSREG Register들이 설정되고 DPS에 P Singal을 인가시키면 Processing Unit들은 Control Unit에 의해 선택된 Instruction을 수행하게 된다.



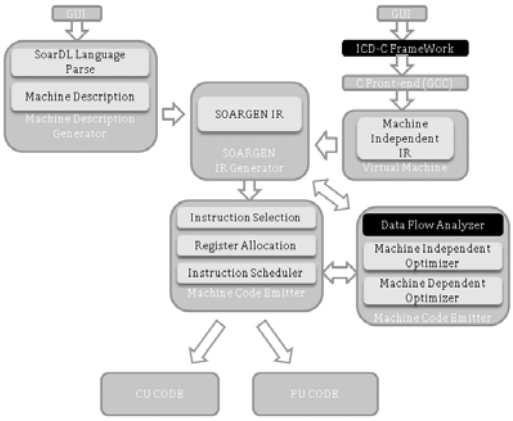
[그림 9] SIMD control

마지막으로 세 번째 경우는 모든 Processing Unit들이 같은 Instruction을 수행하여 마치 SIMD Control Method과 같다. 이를 'SIMD contrl'이라 부르고 [그림 9]는 그 예이다. Control Unit은 DPS Signal에 S를 인가하고 SI signal로 넷 중 어느 Instruction을 수행할 지 결정한다. PU는 Control Unit이 정해주는 Instruction을 수행하게 된다.

3. Soargen Retargetable Compiler

SoarGen은 소프트웨어 최적화 연구실에서 개발한 Retargetable compiler platform이다. Code generation에 필요한 각 알고리즘이 component로 구성되어 이 component를 서로 연결하여 code generation 작업이 수행되도록 고안되었다. 각 Component중에서 target machine의 정보가 필요한 부분은 SoarDL parser로부터 자동으로 생성된다.

[그림 10]은 ERP를 위한 SoarGen compiler의 전체 구조를 나타낸 그림이다.



[그림 10] Soargen Compiler for Dynamic-partitioned SIMD

SoarGen compiler는 크게 다음의 component들로 구성되어 있다.

- front end
- virtual machine & virtual assembly
- soargen intermediate representation(IR)
- analyzer
- optimizer
- local code generator
- global register allocator
- machine description information

여기에 DP-SIMD 기능을 위해 추가한 component는 다음과 같다.

- ICD-C FrameWork
- analyzer
  - Function Call to Instruction
  - CU / PU Seperator
- local code generator
  - DP-SIMD Mode
  - IF Mode

3.1. ICD-C FrameWork

C Language로는 변수 선언 시 Control Unit (CU)의 변수인지 Processing Unit(PU)의 변수인지를 표현하지 못한다. 하지만 C Language를 확장한다면[11] 이 문제를 해결 할 수 있다. 본 컴파일러에서는 #pragma를 이용하여 변수에 CU, PU구분을 해 주었다. ICD-C FramWork 에서는 pre-processing을 하여 #pragma를 C Language가 readable한 Code로 만들어준다.

3.2. front end

front end는 일반적으로 compiler에서 source code를 분석하고 문법적인 오류를 checking한 뒤에 이를 compiler내부에서 사용되는 intermediate representation을 변환해주는 역할을 하는 module이다. 우리는 독자적인 front end를 제작하지

않고 gcc compiler를 우리의 front end로 사용하였다.

3.3. Soargen intermediate representation(IR)

Soargen IR은 SoarGen compiler 내부에서 code를 생성하기 위해 필요한 intermediate re-epresentation이다. 특히 tree code generation을 하기 위해서 virtual assembly를 graph의 형태로 표현한다.

3.4. analyzer

analyzer는 virtual assembly code에 대해서 data flow analysis를 한다. 여기에서는 reaching definition analysis, DU/UD chain, web, live range analysis 등의 data flow analysis를 하고 ERP 를 위해 Function Call To Instruction과 CU/PU Seperator를 수행한다.

Instruction	Description
MAX a b > c	c := (a > b) ? a : b
MIN a b > c	c := (a < b) ? a : b
CLIP a b > c	c := max(0, min(a, b))
SUBABS a b > c	c := abs(a - b)
SUBABS4 a b c d > e	e := abs(a - b) + abs(c - d)
SRAC a b > c	c := a >> b + carry
CMP a cnd b > c	c := (a cnd b) ? 1 : 0

[그림 11] special instruction

[그림 11]의 special instruction들은 Vedio Processing을 위해 필수적인 instruction이지만 C language로 표현이 불가능하다. 따라서 C Language로는 Function Call 형태로 기술하고 analyzer단계에서 이를 Instruction으로 바꿔준다. 알고리즘은 다음과 같다.

1. *Function Call의 Out Parameter를 Control Flow Graph에서 모두 찾아낸다.*
2. *Function Call의 Return Register가 사용되는 instruction들을 모두 찾아낸다.*
3. *Function Call의 Memory Argument를 GPR로 바꾼다.*
4. *Function Call을 Instruction 으로 바꾼 후 Destination Register를 할당한다.*
5. *Return Register를 4에서 할당한 Register로 바꾼다.*

Function Call To Instruction을 수행하고 난 후 analyzer에서는 web을 만든다. web을 만들기 위해서 우선 reaching definition analysis를 하여 definition과 use 사이의 관계를 파악한 뒤에 UD chain을 만들고 미리 생성된 정보를 이용하여 DU/UD chain 을 만든다. DU/UD chain 의 maximal이 web이 된다. 이렇게 해서 생성된 web으로 묶인 symbolic variable은 renaming을 거쳐서 unique한 이름을 갖도록 바꾸어 준다. unique한 이름을 가지는 이유는 나중에 global register allocation을 쉽게 하도록 하기 위함이다.

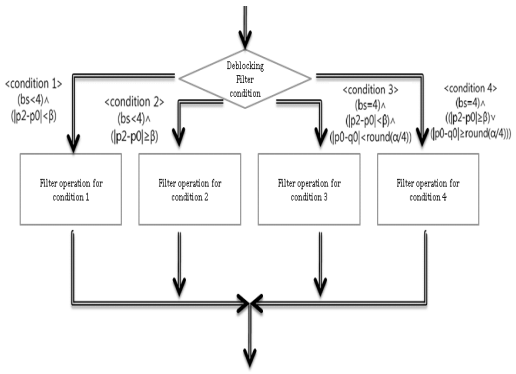
그런 후 pre-processing때 전달되었던 CU/PU 정보를 통해 모든 Register에 대해 PU class와 CU class로 나눠준다. Data flow graph을 이용하여 모든 instruction을 탐색하면서 만약 PU Register일 경우 Operator를 PU용 Operator로 바꿔준다. 예를 들어 ss_plus의 source, destination register가 PU register였을 경우 ss_plus를 pu_ss_plus로 바꿔 저장하게 된다.

4. 실험

De-blocking Filter(DF)와 Intra Prediction(IP)은 H.264의 성능에 큰 부분을 차지하고있다. DP-SIMD Control Method을 사용하여 DF와 IP를 구현하였을 때 기존 SIMD Control Method에 비해 얼마나 빨라졌는지 실험 결과를 통해 알 수 있다.

4.1. De-blocking Filter

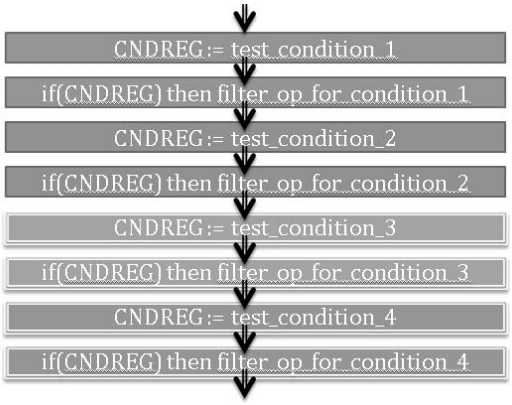
De-blocking Filter란 영상을 여러개의 블록으로 나눠서 인코딩 하는 경우 부작용으로 블록화 현상이 생기게 되는데 이런 블록화 현상이 있는 영상을 보정해주는 것을 말한다. Deblocking Filter의 알고리즘은 다음 [그림 12]와 같다.



[그림 12] De-blocking Filter Algorithm

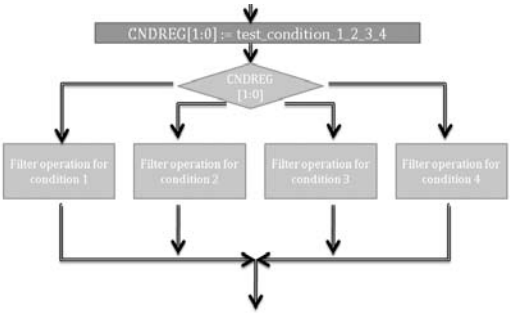
Boundary Strength(bs) 값과 p2, p0값 등에 따라서 네가지 경우로 나뉘어 져 각기 다른 Operation을 수행한다. 먼저 bs값이 4보다 작을 경우 condition 1과 condition 2로 분류된다. 그리고 p2와 p0의 차 값에 따라 다시 분류된다. 두번째로 bs값에 4보다 클 경우 condition 3과 condition 4로 분류된다. 그리고 p2와 p0의 차 값에 따라 다시 condition 3과 condition 4가 분류 된다.

[그림 13]은 DF를 SIMD Control Method으로 작동시켰을 때의 Operation method를 보여준다. 먼저 CNDREG을 condition 1에 의해 셋팅한다. 그리고 conditional operation으로 CNDREG가 1일 경우 Filter Operation 1을 수행한다. 그런 다음 CNDREG를 condition 2에 의해 셋팅하고 Filter Operation 2을 수행한다. 이런 방법으로 Filter Operation 4까지 모두 수행을 하고나면 DF는 종료된다.



[그림 13] De-blocking Filter for SIMD

하지만 DP-SIMD Control Method을 사용하면 [그림 14]과 같이 Operation을 수행할 수 있다. 먼저 모든 condition에 대해 2bit의 CNDREG에 셋팅하고 병렬로 네가지 operation 중 하나를 수행하게 한다.



[그림 14] De-blocking Filter for DP-SIMD

De-blocking filter를 SIMD를 지원 하는 컴파일러와 DP-SIMD를 지원 하는 컴파일러를 각각 사용하여 구현하였다. 결과는 아래 표와 같다.

	Old Compiler	DP-SIMD Compiler	Efficiency(Old compiler / DP-SIMD Compiler)
cycle	222	150	1.48

DP-SIMD Control Method으로 코드를 생성하는 Compiler의 결과가 SIMD Control Method

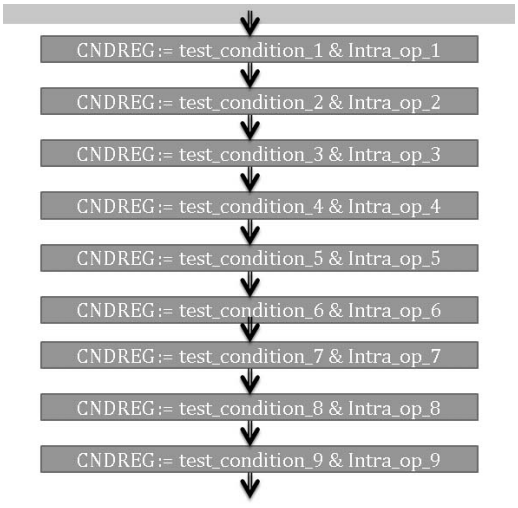
으로 코드를 생성하는 Compiler에 비해 성능이 1.48배 빨라졌음을 볼 수 있다. DF를 Hand-optimize한 코드의 결과는 아래 표와 같다.

	SIMD	DP-SIMD	Efficiency (SIMD / DP-SIMD)
cycle	78	50	1.56

Hand-Optimized 코드 역시 DP-SIMD 코드가 SIMD 코드에 비해 성능이 뛰어났으며 Compiler에 비해 hand-optimized Code가 성능이 3배 더 좋음을 알 수 있다.

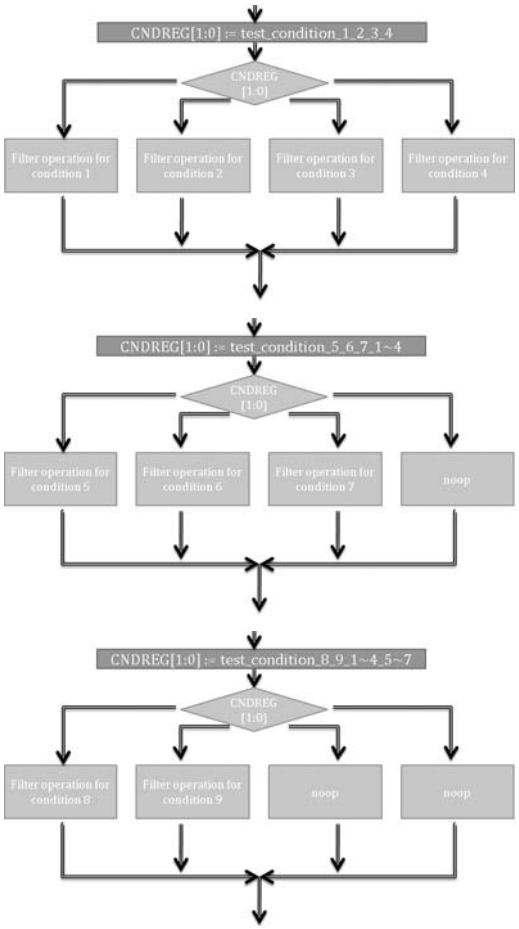
4.2. Intra Prediction

영상은 자기 주변의 화소값들이 대체로 비슷한 값을 가진다. H.264의 최소 block인 4x4나 16x16 block들도 이웃한 block 들과 값이 비슷하다. 이러한 block 간의 값의 유사성을 이용해서 block 간의 값의 차를 인코딩한다. Intra prediction은 9가지 모드가 있다. 9가지 모드에 따라 9가지 다른 Operation을 수행해야한다. SIMD 컨트롤 방법으로 구현하면 [그림 15]와 같다.



[그림 15] Intra Prediction for SIMD

9개의 Operation을 위해 9번 CNDREG를 셋팅하고 각각의 Operation을 수행하게 된다. 반면에 DP-SIMD 컨트롤 방법으로 작동을 시키려면 먼저 9가지 모드 중 네가지 모드에 대해 DP-SIMD로 Operation을 수행한 후, 추가로 세가지 모드에 대한 Operation과 이미 Operation이 수행된 모드들에 대해서는 nop으로 채운 Operation을 수행시킨다. 그다음 단계에서 남은 두가지 모드에 대한 Operation과 이미 수행된 모드들에 대한 nop Operation을 수행하면 9가지 모드에 대한 DP-SIMD Operation을 모두 수행할 수 있게 된다. [그림16]은 DP-SIMD로 IP를 구현한 모습이다.



[그림 16] Intra Prediction for DP-SIMD

intra prediction을 SIMD를 지원 하는 컴파일러와 DP-SIMD를 지원하는 컴파일러를 각각 상용하여 구현해 보았다. 결과는 아래 표와 같다.

	Old Compiler	DP-SIMD Compiler	Efficiency(Old compiler / DP-SIMD Compiler)
cycle	384	210	1.83

DP-SIMD 컨트롤 방법으로 코드를 생성하는 Compiler의 결과가 SIMD 컨트롤 방법으로 코드를 생성하는 Compiler에 비해 성능이 1.83배 좋아졌음을 볼 수 있다. IP를 Hand-optimize한 코드의 결과는 아래 표와 같다.

	SIMD	DP-SIMD	Efficiency (SIMD / DP-SIMD)
cycle	199	105	1.9

5. 결론

본 논문에선 Retargetable Compiler인 SoarGen을 사용하여 DP-SIMD 컨트롤 방법을 지원하는 ERP 프로세서용 컴파일러를 제작하였고, 이 컴파일러 사용해서 생성된 H.264의 De-blocking Filter와 Intra Prediction을 코드를 사용해 성능을 검증하였다. 실험 결과는 사람이 직접 작성한 코드보다 다소 성능이 떨어지는 것으로 나왔다. 물론 Hand-optimized code보다 성능이 뛰어난 코드를 컴파일러가 생성하기는 거의 불가능하지만 Soargen Compiler의 성능이 떨어지는 이유는 다음과 같다. 우선 Array의 load / store 시 hand-optimized Code의 경우에는 주소를 바로 입력해서 사용하지만 컴파일러의 경우 주소를 4씩 계속 더해서 사용하기 때문에 Code가 길어진다. 두 번째는 ERP의 경우 R0는 항상 0이 assign되어있기 때문에 immediate value로 0을 사용할 시 R0를 사용하면 되나 Soargen Compiler는 이를 위한 최적화 기능이 없어 항상 MOVI를 이용하여 0을 assign해야한다. 특히 Processing unit은 MOVI가 없기 때문에 Control Unit에서 MOVI를 한 다음 Processing Unit으로 전달을 해 주어야 하기 때문에 더욱 코드가 길어지게 된다. 세 번째로 Soargen Compiler에 현재 구현되어있는 최적화 기법은 DAG을 Tree로 바꿔주는 Dismantle 밖에 없기 때문에 여러 redundant 한 코드들이 남아있다. 이는 추후에 최적화 기법을 적용함으로써 개선될 수 있으리라 본다. 이런 단점에도 불구하고 컴파일러를 사용하면 어플리케이션 개발 시간이 크게 단축될 수 있다. DF의 경우 최적화된 Hand-optimized code를 만들기 위해 compiler로 code를 생성할 때 보다 10배 정도의 시간이 소모된다. 컴파일러를 이용하여 어플리케이션을 작성한 뒤 이를 이용하여 테스트를 하거나 hand-optimizing을 하는 것 또한 좋은 방법이다. 따라서 Soargen compiler를 사용함으로써 생기는 performance loss를 감안하더라도 이의 사용은 충분히 가치가 있는 일이다.

참 고 문 헌

[1] M. Horowitz, A. Joch, F. Kossentini, etal., “H.264/AVC Baseline Profile Decoder Complexity Analysis,” IEEE Trans. Circuits Syst. for Video Technol., vol.13, July 2003, pp.704-716

[2] S. López, F. Tobajas, G. M. Callicó, etal., “A Novel High Performance Architecture for H.264/AVC Deblocking Filtering,” ETRI Journal, vol.29, no.3, Jun. 2007, pp.396-398

[3] M. Oh, W. Lee, Y. Jung, et al., “Design of High-Speed CAVLD Decoder Architecture for H.264/AVC,” *ETRI Journal,* vol. 30, no.1, Feb. 2008, pp.167-169

[4] S. M. Park, M. Lee, S. Kim, et al., “VLSI Implementation of H.264 Video Decoder for Mobile Multimedia Application,” ETRI

Journal, vol.28, no.4, Aug. 2006, pp.525-528
[5] D. Yeo and H. Shin, "High Throughput Parallel Decoding Method for H.264/AVC CAVLC," *ETRI Journal,* vol. 31, no. 5, Oct. 2009, pp.510-517
[6] K.-S. Choi and S.-J. Ko, "Adaptive Scanning Based on a Morphological Representation of Coefficients for H.264/ AVC," *ETRI Journal,* vol.31, no.5, Oct. 2009, pp.607-609
[7] Y. Kun, Z. Chun, and W. Zhihua, "Application Specific Processor Design For H.264 Baseline Profile Bit-Stream Decoding," *Proceedings of The 8th International Conferenceon Signal Processing,* 2006, pp.16-20
[8] J. H. Han, M. Y. Lee, Y. Bae, et al., "Application Specific Processor Designfor H.264 Decoder with a Configurable Embedded Processor," ETRI Journal, vol. 27, no.5, Oct.2005, pp.491-496
[9] H. Singh, M.-H. Lee, G. Lu, etal., "Morpho Sys: An Integrated Reconfigurable System for Data-Paralleland Computation-Intensive Application," *IEEE Transactionson Computers,* vol.49, no.5, May 2000, pp.465-481.
[10] Chun-Gi Lyuh, Jung-Hee Suk, Ik-Jae Chun, and Tae Moon Roh, "A Novel Reconfigurable Processor Using Dynamically Partitioned SIMD for Multimedia Applications" ETRI Journal, vol.31, no.6, Dec. 2009, pp.709-716.
[11] Shorin Kyo, Shin'ichiro Okazaki, Tamio Arai, "An Integrated Memory Array Processor Architecture for Embedded Image Recognition Systems", ACM SIGARCH Computer Architecture News, vol 33, issue 2, 2005, pp.134-145
[12] JVT H.264/AVC Joint Model Reference S oftware version 11.0, http://iphome.hhi.de/suehring/tml/download/old_jm/jm11.0.zip, Aug. 2007

권 용 인
2002~2007년 한국과학기술원 전기 및 전자공학과(학사)원 전산학과(학사)
2008년~2009년 서울대학교 전기컴퓨터공학부(석사)
관심분야: 프로그래밍 언어, 컴파일러

윤 종 희
1972년~1976년 서울대학교 전
2003년 경북대학교 전기공학부(석사)
2003년~현재 서울대학교 전기컴퓨터공학부 석박사통합과정
관심분야 : 임베디드소프트웨어, 컴파일러

이 종 원
2007년 서울대학교 전기공학부(학사)
2007년~현재 서울대학교 전기컴퓨터공학부 석박통합과정
관심분야 : 임베디드소프트웨어, 컴파일러

김 용 주
2006년 서울대학교 전기공학부(학사)
2006년~현재 서울대학교 전기컴퓨터공학부 석박통합과정
관심분야 : 임베디드소프트웨어, 저전력 설계

백 윤 흥
1988년 서울대학교 컴퓨터공학과 (학사)
1990년 서울대학교 컴퓨터공학과 (석사)
1997년 UIUC 전산과학(박사)
1997년~1999년 NJIT 조교수
1999년~2003년 KAIST 전자전산학과 부교수
2003년~2007년 서울대학교 전기컴퓨터공학부 부교수
2007년~현재 서울대학교 전기컴퓨터공학부 교수
관심분야 : 임베디드 소프트웨어, 임베디드 시스템 개발 도구, 컴파일러, MPSoCs

논문 23-1-R03

스타일을 설정할 수 있는 웹 기반 문서 편집 환경

Web Based Document Editing Environment with User-Defined Styles

황준형* · 윤정한* · 최석우 · 한태숙***

카이스트 전산학과* · KT 네트워크 연구소**

envia@envia.pe.kr, jeonghan.yun@gmail.com, choi.seokwoo@kt.com, han@cs.kaist.ac.kr

요 약

문서의 특정 구조를 표시할 때 쓰이는 표현 방법을 스타일이라고 한다. 웹 문서에서 스타일을 사용하면 문서에서 쓰이는 표현 방법들을 한 곳에 모아서 문서의 내용과 표현 방법을 분리하는 것이 가능하다. 이를 통해 문서에서 필요한 정보를 추출하거나 유지 보수하는 것을 편리하게 할 수 있다. 웹에 있는 다양하고 시간에 따라 변화하는 정보를 고정된 스타일만으로 나타내는 것에는 한계가 있으며, 사용자의 필요에 따라 스타일을 바꾸어줄 필요가 있다. 기존에는 원하는 형태의 스타일을 사용하기 위해서는 웹 사이트에 있는 스타일 관련 파일을 직접 편집해 주어야 했다. 따라서 파일을 수정할 권한이 없거나 웹 프로그래밍에 대한 지식이 부족한 경우 자신이 원하는 스타일을 지정해서 사용하는 것이 쉽지 않았다. 따라서 제공되는 스타일이 사용자가 원하는 것과 다를 경우 스타일을 사용했을 때의 장점을 얻는데 어려움이 있었다. 본 논문에서는 기본으로 주어지는 스타일들을 바탕으로 사용자들이 자신의 필요에 맞는 스타일을 설정하는 것을 지원하는 웹 기반 편집 환경을 제안한다. 제안한 편집 환경을 이용하면 웹 프로그래밍에 대한 특별한 지식이 없어도 스타일을 쉽게 작성하고 적용할 수 있다. 이를 이용하면 사용자가 스타일을 유연하게 변경할 수 있으므로 스타일을 사용했을 때의 장점을 충분히 얻을 수 있다. 스타일 정보가 웹에 저장되기 때문에 스타일 정보를 여러 곳에서 사용하거나 공유하는 것도 가능하다. 우리는 제안한 편집 환경을 위키 사이트에 적용해 보았다. 편집 환경을 구현한 방법과 스타일의 활용 사례도 함께 소개한다.

1. 서 론

인터넷이 보급되면서 점점 더 많은 정보들이 웹을 통해 제공되고 있으며, 웹에서 정보를 제공하는 사람들의 수도 늘어나고 있다. 따라서 웹 사이트를 유지 보수하는 일이 점점 더 중요해지고 있으며, 체계적인 방법으로 작성한 웹 문서의 가치가 주목을 받고 있다.

스타일은 문서의 특정 구조를 표시할 때 쓰이는 표현 방법을 말한다. 스타일을 사용하면 웹 사이트의 유지 보수를 편리하게 할 수 있다. 스타일을 이용해서 웹 문서에서 쓰이는 표현 방법들을 한 곳에 모을 수 있는데, 표현 방법들을 한 곳에 모으면 문서의 내용과 그 표현 방법을 분리

하는 효과를 얻을 수 있다. 문서의 내용과 표현 방법을 분리하면 한 쪽은 그대로 둔 채 다른 한 쪽만 독립적으로 바꾸는 것이 가능하며, 한 쪽의 정보만을 따로 이용하는 것도 가능하다. 따라서 사이트의 구조를 바꾸거나 필요한 정보를 추출하는 일이 간단해진다.

스타일을 이용해서 최대의 효과를 얻기 위해서는 실제 문서의 내용에 맞는 스타일을 제공해야 한다. 제공되는 스타일이 실제 문서의 내용과 맞지 않으면 스타일을 사용할 수 없거나 사용하더라도 혼란을 초래하게 된다. 웹 사이트의 내용은 다양하고 시간에 따라 변화하기 때문에 고정된 스타일만으로 나타내는 데에는 한계가 있다. 사이트에 맞는 스타일을 장기적으로 제공하기 위해서는 문서의 내용에 맞추어 스타일을 변경하는 것이 가능해야 한다.

기존에는 원하는 형태의 스타일을 사용하기 위해서 웹 사이트에 있는 스타일 관련 파일을 직접 편집해 주어야 했다. 파일을 수정할 권한이 있는 웹 사이트 운영자는 사이트에서 사용할 스타일을 쉽게 설정하여 사용할 수 있지만, 권한이 없고 웹 프로그래밍에 대한 지식도 부족한 사이트 사용자의 경우 자신이 원하는 스타일을 설정하여 사용하는 데 어려움을 겪게 된다. 사이트에서 제공하는 스타일이 사용자가 원하는 것과 다를 경우 이를 수정하는 것이 힘들기 때문에 스타일을 사용해서 얻을 수 있는 장점을 경험할 수 없게 된다.

본 논문에서는 사용자들이 자신의 필요에 맞는 스타일을 직접 설정하는 것을 지원하는 웹 기반 편집 환경을 제안한다. 제안한 편집 환경을 이용하면 웹 프로그래밍에 대해 특별한 지식이 없는 사용자도 스타일을 쉽게 설정하고 적용할 수 있으며, 이를 통해 스타일을 사용했을 때 얻을 수 있는 이점을 온전히 누릴 수 있다. 스타일 정보가 웹에 저장되기 때문에 스타일 정보를 여러 곳에서 사용하거나 공유하는 것도 가능하다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구들을 소개하고 그들의 문제점과 한계를 지적한다. 3장에서는 사용자가 스타일을 설정할 수 있는 편집 환경과 그 구현 방법을 제안한다. 4장에서는 편집 환경을 위키 사이트에 적용한 사례를 소개하며, 5장에서는 스타일의 활용 사례를 소개한다. 6장에서는 결론을 맺고 앞으로의 연구 방향에 대해 생각해 본다.

2. 기존 방법과 문제점

웹 문서의 내용과 표현 방법을 스타일을 이용해서 분리하려고 할 경우 각각에 해당하는 부분을 독립적으로 작성하게 된다. 보통 내용에 해당하는 부분은 HTML이나 XML로 작성하고 표현 방법에 해당하는 부분은 CSS로 작성한다. 내용에 해당하는 부분을 XML로 작성하고 표현 방법에 해당하는 부분을 XSL로 작성한 뒤 이들을 HTML로 변환하는 경우도 있다. 본 논문에서는 첫 번째 방법을 사용한 경우에 대해서만 다루지만, 본 논문의 내용은 두 번째 방법을 사용한 경우에도 유사하게 적용할 수 있다.

스타일을 이용해서 문서의 내용과 표현 방법을 분리하는 방법은 기존 문서 편집기에서 널리 쓰이고 있다. LaTeX 편집기인 LyX[1]에서는 이러한 개념을 WYSIWYM(What You See Is What You Mean)이라는 표현으로 나타내고 있다. 하지만 기존 웹 편집기들의 대부분은 문서 편집기와 같이 스타일을 설정하는 것을 지원하지 않는다.

웹 사이트에서 쓰이는 스타일을 바꾸는 가장 간단한 방법은 웹 사이트에 있는 CSS 파일을 직접 수정하는 방법이다. 하지만 CSS 파일을 직접 수정하기 위해서는 웹 사이트에 대한 높은 수준의 권한이 필요하거나 상당한 웹 프로그래밍 지식이 필요한 경우가 많아 일반 사용자가 이용하기에는 어려움이 있다.

일부 웹 편집기는 스타일을 편집할 수 있는 기

능을 제공한다. Amaya[2]는 사용자가 HTML 파일의 내용과 함께 CSS의 내용을 바꾸는 것을 허용함으로써 스타일 편집을 지원한다[3]. 하지만 Amaya는 웹 기반이 아니므로 Amaya를 이용해서 웹 문서를 수정하기 위해서는 Amaya를 먼저 설치해야 하는 번거로움이 있다. 한편 Amaya는 수정한 파일을 PUT 방식(method)으로 업로드하기 때문에 Amaya를 이용해서 웹 문서의 내용을 편집하려면 웹 서버에서 PUT 방식을 지원해야 한다. 현재 대부분의 웹 서버들은 보안상의 이유로 PUT 방식을 지원하지 않고 있으므로 이 방법을 이용해서 일반적인 웹 문서의 내용을 편집하는 데에는 한계가 있다.

스타일을 지원하는 웹 기반 편집기도 여럿 존재한다. 그림 1은 WYMeditor[4]의 화면이다. 사용자는 화면 오른쪽에 있는 목록에서 스타일을 선택할 수 있다. 화면 왼쪽의 편집 창에 있는 문서 내용은 해당하는 스타일을 나타내는 상자 안에 표시된다. 이와 같은 편집기는 이미 정해진 스타일을 사용하는 것은 지원하지만, 스타일을 새로 설정하거나 변경하는 것은 지원하지 않는다. 따라서 사이트에서 지원하는 스타일이 사용자가 원하는 것과 다를 경우 스타일을 충분히 활용하는 데 어려움이 있다.

3. 스타일을 설정할 수 있는 편집 환경

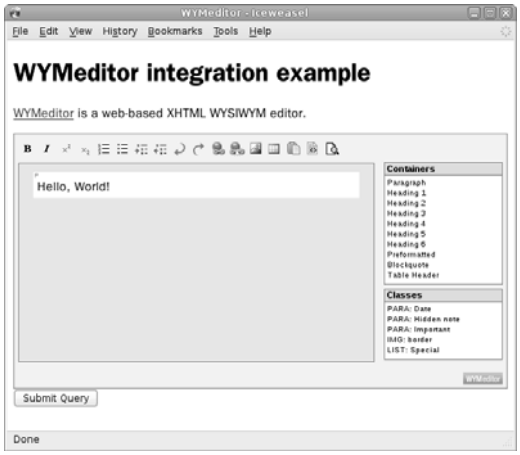
3.1. 구현 내용

우리는 웹 환경에서 사용자가 자신만의 스타일을 설정하여 사용할 수 있는 편집 환경을 개발하였다. 이를 위해 먼저 스타일을 관리하는 기능들을 개발하였다. 그 다음 기존의 편집기를 수정하여 스타일을 적용할 수 있도록 하였는데, 이것에 대해서는 4장에서 자세히 다루기로 한다.

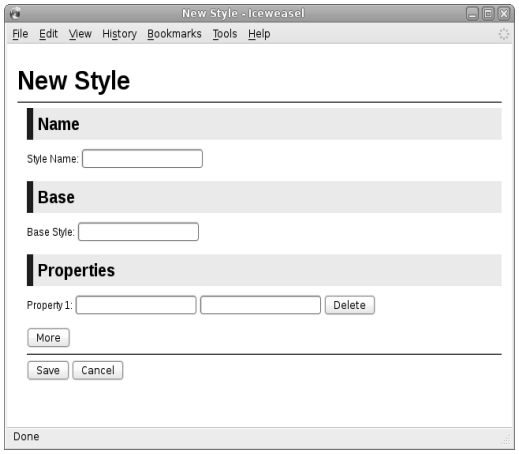
[그림 2]에서 [그림 5]까지는 스타일을 관리하는 기능에 해당하는 화면을 나타내고 있다. 구체적으로 다음 4가지 기능을 구현하였다.

- 스타일을 추가하는 기능 [그림 2]
- 스타일을 변경하는 기능 [그림 3]
- 스타일을 내보내는 기능 [그림 4]
- 스타일을 불러오는 기능 [그림 5]

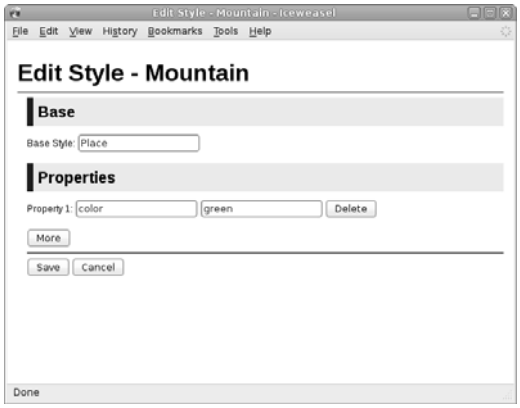
스타일을 추가하고 변경하는 화면에서는 스타일의 이름과 바탕 스타일, 스타일 속성을 지정할 수 있다. 스타일을 내보내는 화면에서는 내보낼 스타일을 선택할 수 있고, 스타일을 가져오는 화면에서는 스타일을 가져올 곳을 선택할 수 있다.



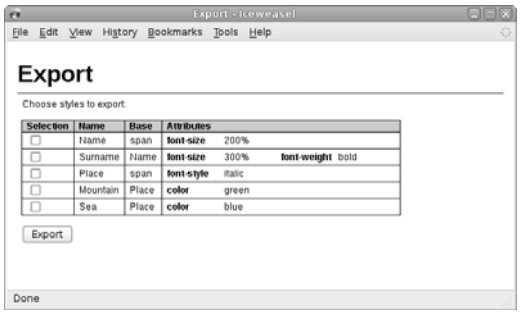
[그림 1] WYMeditor의 화면



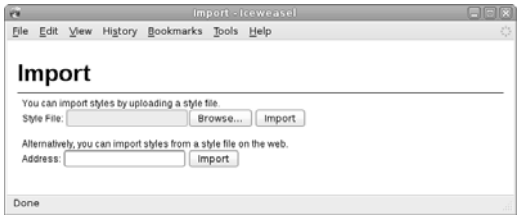
[그림 2] 스타일 추가 화면



[그림 3] 스타일 변경 화면



[그림 4] 스타일 내보내기 화면



[그림 5] 스타일 가져오기 화면

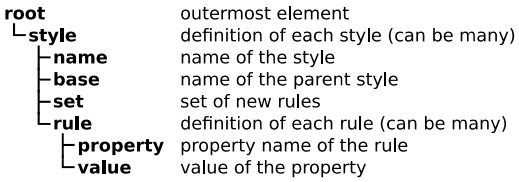
3.2. 스타일 구조

스타일을 설정할 때에는 스타일 이름과 바탕 스타일, 스타일 속성을 지정한다. 스타일 이름은 스타일을 구분하기 위해 사용하며, HTML과 CSS 파일을 만들 때 클래스(class) 이름으로도 쓰인다. 바탕 스타일은 기존 스타일 이름을 사용할 수도 있고, HTML 태그 이름을 사용할 수도 있다. 바탕 스타일에 HTML 태그 이름을 사용할 경우 실제 HTML 문서를 만들 때 해당 태그를 사용하게 된다. 바탕 스타일을 지정하지 않을 경우 기본 스타일을 사용하는데, 기본 스타일은 블록 요소의 경우 div이고 인라인 요소의 경우 span이다. 스타일 속성은 CSS 규칙에서 쓰이는 속성에 대응하며, 속성 이름과 값을 지정할 수 있다. 아무 속성도 지정하지 않을 경우 부모 스타일에 해당하는 표현 방법만을 사용하게 된다.

스타일을 적용할 때에는 모든 부모 스타일에 해당하는 표현 방법과 스타일 속성을 이용하여 지정한 표현 방법을 함께 적용한다. 부모 스타일과 자식 스타일의 표현 방법이 충돌할 경우에는 자식 스타일의 표현 방법을 적용한다.

3.3. 스타일 저장

스타일을 저장하는 것은 XML 형식을 사용하였다. 텍스트 파일 형식이기 때문에 직접 편집하기도 편하고, 많은 언어들이 XML을 지원하기 때문이다. PukiWiki를 작성할 때 사용한 PHP의 경우 PHP 5부터 XML을 기본으로 지원한다.



[그림 6] 스타일 파일의 구조

```
<?xml version="1.0"?>
<root>
 <style>
  <name>Name</name>
  <base>span</base>
  <set>
   <rule>
    <property>font-weight</property>
    <value>bold</value>
   </rule>
  </set>
 </style>
</root>
```

[그림 7] 스타일 파일의 예

스타일 파일의 구조는 [그림 6]과 같다. 가장 바깥에 root 요소(element)가 있고, 그 안에 style 요소들이 있다. style 요소 안에는 name 요소 하나, base 요소 하나, set 요소 하나가 있다. set 요소 안에는 rule 요소가 여럿 있는데, 각각에는 property 요소 하나, value 요소 하나가 있다.

스타일 파일에서 root 요소는 가장 바깥에 있는 요소이며, style은 각각의 스타일에 해당하고, name은 스타일의 이름, base는 스타일의 부모 스타일, set은 새로 추가되는 규칙들에 해당한다. rule 요소는 각각의 규칙들에 해당하고, property는 규칙이 적용되는 속성의 이름에, value는 속성의 값에 해당한다.

[그림 7]은 표현 방법으로 굵은 글꼴을 사용하도록 하는 Name 스타일이 있는 스타일 파일의 내용을 나타낸 것이다.

4. 위키에서 스타일 사용하기

우리는 스타일을 편집하는 기능을 위키 소프트웨어에 적용하였다. 위키는 사용자가 위키 문법에 맞추어 입력한 내용을 HTML로 변환하여 표시한다. 위키는 구조가 단순하여 새로운 기능을 추가하기가 편리하다. 위키 중에서는 소스 코드가 비교적 간단한 PukiWiki[5]를 선택하였다. 이를 통해 제안한 편집 환경을 기존 웹 사이트에 적용하는 것이 가능하다는 것을 보였다.

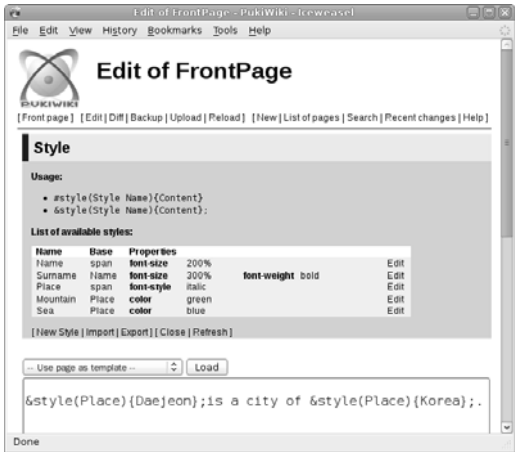
4.1. 스타일 관리

스타일을 추가하고, 변경하고, 내보내고, 불러오는 부분을 편집 창에 출력하기 위해 다음과 같은 방법들을 사용할 수 있다.

1. 전체 페이지를 새로 불러오는 방법
2. 팝업 윈도우를 사용하는 방법
3. AJAX를 사용하는 방법
4. IFRAME(내부 프레임)을 사용하는 방법

전체 페이지를 새로 불러오는 방법은 웹 표준에 가장 잘 맞는 방법이지만 반응 속도가 느리다는 단점이 있다. 팝업 윈도우를 사용하는 방법의 경우 팝업 윈도우에서 스타일을 관리한 다음 그 결과를 편집 창에 반영하는 방법인데 역시 느리고 사용하기 불편하다는 단점이 있다. AJAX를 이용할 경우 동적으로 자료를 받아와서 페이지를 고치게 되는데 반응성이 좋아서 널리 쓰이는 방법이지만 웹 표준이 아니라는 단점이 있다. IFRAME을 사용할 경우 편집기 화면의 일부를 IFRAME으로 지정해서 필요할 때마다 해당하는 페이지의 내용을 표시한다. IFRAME은 표준에 포함되어 있는 방법이고 반응 속도도 나쁘지는 않지만 한 페이지의 내용이 여러 파일로 나뉘게 되면서 사용성이 떨어진다는 단점이 있다.

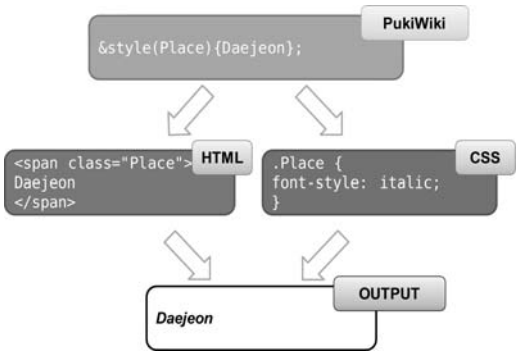
우리는 비교적 구현이 간단한 IFRAME을 사용하여 스타일을 관리하는 화면을 편집 창에 표시하였으며, 다른 방법을 사용하더라도 기능에 큰 차이는 없다. 다른 웹 사이트에서도 비슷한 방법으로 스타일 관리 화면을 추가하는 것이 가능하다. [그림 8]은 스타일 기능을 추가한 PukiWiki의 화면을 나타내고 있다.



[그림 8] 스타일 관리 화면이 추가된 편집창

4.2. 스타일 적용

PukiWiki에서 문서의 내용에 스타일을 적용할 수 있도록 하기 위해 style 플러그인을 만들어 PukiWiki에 포함시켰다. style 플러그인을 적용한 내용은 HTML을 생성할 때 스타일을 적용할 수 있는 형태로 출력되며, CSS를 생성할 때 해당하는 표현 방법이 포함된다. 이 과정을 그림으로 나타내면 [그림 9]와 같다.



[그림 9] 위키에서 HTML과 CSS의 생성 과정

HTML을 생성하는 부분은 PukiWiki의 플러그인 구조를 이용하여 구현하였다. 스타일을 적용할 내용을 스타일에 해당하는 HTML 태그로 둘러싼 다음 해당하는 클래스 속성을 추가하였다. 이 때 HTML 태그는 바탕 스타일을 이용하여 정하고, 클래스 속성은 스타일 이름을 이용하여 정한다. Daejeon라는 단어에 span 태그를 바탕으로 하는 Place 스타일을 적용할 경우 Daejeon과 같이 출력하게 된다.

CSS를 생성하는 부분은 PukiWiki의 플러그인 구조로는 변경할 수 없어서 CSS를 출력하는 코드를 직접 수정하였다. 수정한 코드는 스타일에 해당하는 규칙들을 CSS 파일에 추가한다. 이 때 규칙들은 스타일의 속성과 바탕 스타일의 속성들을 이용해서 정한다. 규칙들을 적용할 대상을 찾을 때 쓰이는 선택자(selector)는 클래스 속성 이름을 이용하여 만든다. 예를 들어 위에서 언급한 Place 스타일에 해당하는 부분을 기울여서 표현해야 할 경우 .Place{font-style: italic;}을 추가하게 된다.

이와 같이 HTML과 CSS를 출력하는 방법으로 스타일이 의도한 것과 같이 적용되는 것을 확인할 수 있었다. PukiWiki에는 프린터에 맞도록 출력하는 기능이 있는데, 이 기능을 사용할 때에도 스타일은 의도한 바와 같이 적용되었다.

다른 웹 사이트에도 비슷한 방법으로 스타일을 적용하는 것이 가능하다. 웹 사이트에서 문서를 저장하는 방법을 스타일도 저장할 수 있도록 확장한 다음, HTML을 출력하는 부분과 CSS를 출력하는 부분을 각각 수정해 주면 된다.

5. 스타일의 활용 사례

스타일은 문서의 내용에 대한 표현 방법을 지정하기 위해 쓰이지만 웹의 특성을 활용하면 다른 용도로도 쓸 수 있다.

스타일은 일종의 메타데이터로 활용할 수 있다. 스타일의 이름을 이용해서 스타일을 적용한 내용이 어떠한 종류의 내용인지를 알아내는 것이 가능하다. 예를 들어 스타일의 이름이 Place라면 스타일을 적용한 표현이 특정 장소에 해당할 것이라는 추론을 할 수 있다. 사용자나 작성 시간에 따라 다른 스타일을 적용할 경우 문서의 해당 부분을 누가 언제 작성했는지를 쉽게 알 수 있도록 표시하는 것이 가능하다.

마이크로포맷(microformat)[6]은 웹 문서의 정보에 대한 메타정보들을 현재 존재하는 웹 표준들을 이용해서 제공하기 위한 형식이다. 마이크로포맷은 메타정보를 나타내기 위해 HTML 태그의 속성을 이용하고 있으며, 가장 많이 쓰이는 속성은 div 태그와 span 태그의 클래스 속성이다. 제안한 편집 환경에서도 스타일을 나타내기 위

해 클래스 속성을 사용하므로 스타일을 이용하면 마이크로포맷을 자연스럽게 나타낼 수 있다.

한 사람이 글을 쓰면서 사용하는 스타일은 일정한 경우가 많으므로 하나의 스타일을 여러 사이트에서 사용한다면 편리할 것이며, 개인의 개성을 나타내는 데에도 도움이 될 것이다. 그리고 여러 사람이 하나의 스타일을 함께 사용한다면 통일된 효과를 주는 것이 가능할 것이며, 스타일을 일관성 있게 변경하는 일도 쉬워질 것이다.

6. 결론과 향후 연구 과제

우리는 사용자가 스타일을 설정할 수 있는 웹 기반 편집 환경을 구현하였으며, 이를 위키 사이트에 적용하였다. 사용자는 웹 프로그래밍에 대한 특별한 지식이나 웹 사이트에 대한 특별한 권한 없이도 필요에 따라 스타일을 설정할 수 있으며, 스타일을 이용하여 웹 문서의 내용과 표현 방법을 분리하여 수정할 수 있다. 웹 사이트에서 사용하는 스타일을 유연하게 변경할 수 있으므로 정보의 활용과 유지 보수가 더욱 편리해진다.

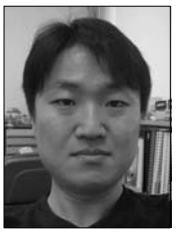
본 연구에서는 기능의 구현에 중점을 두었기 때문에 전반적인 성능은 고려하지 않았다. 현재 제안한 편집 환경을 사용하기 위해서는 HTML 태그와 CSS 속성에 대한 기본적인 지식이 필요하다. 제안한 편집 환경이 널리 쓰이기 위해서는 이러한 지식 없이 이용할 수 있도록 사용자 인터페이스를 개선할 필요가 있다. 이 때 어떠한 편집 환경이 사용하기 편리한지 정량적으로 평가할 수 있는 방법을 고안하여 적용한다면 도움이 될 것이다. 스타일을 자동으로 생성하거나 스타일의 일부 속성을 자동으로 생성하는 방법을 개발하여 적용하면 사용하기가 더욱 편리해질 것으로 기대한다.

참 고 문 헌

[1] LyX Team. LyX. http://www.lyx.org/.
[2] W3C and WAM Project. Amaya. http://www.w3.org/Amaya/.
[3] Vincent Quint and Irne Vatton. Editing with Styles. In *Proceedings of the ACM Symposium on Document Engineering*, pages 121-140, 2007.
[4] WYMteam. WYMeditor. http://www.wymeditor.org/.
[5] PukiWiki Developers Team. PukiWiki. http://pukiwiki.sourceforge.jp/.
[6] Microformats Community. Microformats. http://microformats.org/.

황 준 형

2002년~2009년 한국과학기술원 전산학과(학사)
2009년~현재 카이스트 전산학과 석사과정
관심분야 : 프로그래밍 언어, 컴파일러.



윤 정 한

1997년~2001년 한국과학기술원 전산학과(학사)
2001년~2003년 한국과학기술원 전자전산학과 전산학전공 (석사)
2003년~현재 카이스트 전산학과 박사과정
관심분야 : 프로그래밍 언어, 컴파일러

최 석 우

1994년~1998년 한국과학기술원 전산학과(학사)
1998년~2000년 한국과학기술원 전자전산학과 전산학전공(석사)
2001년~2003년 카이스트 전산학과(박사)
2009년~현재 KT 네트워크 연구소 대리
관심분야: 네트워크 보안, 프로그램 분석

한 태 숙

1972년~1976년 서울대학교 전자공학과(학사)
1976년~1978년 한국과학기술원 전산학과(석사)
1990년~1995년 University of North Carolina at Chapel Hill(박사)
1996년~현재 카이스트 전산학과 부교수
관심분야: 프로그래밍 언어론, 함수형 언어

23-1-R04

소스코드 보안취약성 자동진단도구 개발 사례

Development of a Source Code Security Vulnerability Analyzer

권현준 · 김유경* · 김현하* · 도경구* · 신승철 · 안준선*** · 이욱세* · 이은영**** · 한환수*****

지티원 · 한양대학교* · 한국기술교육대학교** · 한국항공대학교*** ·
동덕여자대학교**** · 성균관대학교*****

hjkweon@gtone.co.kr · {yukoung, hhkim, doh, oukseh}@hanyang.ac.kr · scshin@kut.ac.kr,
jsahn@kau.ac.kr · elee@dongduk.ac.kr · hhan@skku.edu

요 약

본 논문은 2009년도 행전안전부의 주관 및 지원 하에 한국인터넷진흥원을 중심으로 (주)지티원, (주)파수닷컴, 한국정보보호학회 소프트웨어보안연구회에서 개발한 소스코드 보안취약성 자동진단 도구 개발 사례를 소개한다. 본 과제와 관련하여 규칙 기반 취약성 분석엔진이 개발되었고, 보안 취약성 데이터베이스가 구축되었다. 개발된 소프트웨어는 전자정보시스템 취약성 분석에 적용됨으로써 국가 정보시스템의 보안 안전성 강화에 활용될 것이다.

1. 서 론

컴퓨팅 시스템의 보안문제의 해결에 있어서 소프트웨어 소스코드 수준의 보안성 강화를 통한 근원적인 접근법에 관심이 증가하고 있다. 최근 들어 응용프로그램의 취약성을 사용한 공격의 빈도가 사이버 위협의 주요 원인이 되고 있는 것으로 보고되고 있으며, 거의 모든 종류의 정보서비스가 불특정 다수가 직접 접근할 수 있는 인터넷을 통해 제공되는 상황에서 소프트웨어 소스코드 수준의 보안성 강화가 중요한 관심사로 등장하고 있다. 또한 2009년 7월 7일 국내에서 발생한 대규모 DDoS(분산서비스거부) 공격과 같은 소프트웨어 시스템에 대한 공격 피해사례가 전 세계적으로도 증가하는 추세이며, 이러한 사례의 근원적인 원인도 대부분 소프트웨어 소스코드의 보안 취약성에서 찾을 수 있는 것으로 밝혀지고 있다.

코드리뷰를 통하여 소스코드의 버그 또는 보안 취약성을 찾아내는 작업이 이루어지고 있지만, 개인의 숙련도에 의해서 품질이 좌우되고 경제적이지도 못하므로 자동으로 찾아주는 도구가 필요하다. 또한 다양한 보안 취약성 각각에 대하여 검출 방법과 공격 방법에 대한 방지책을 일일이 마련하여야 하므로 취약성에 대한 다양한 정보를 구축하는 작업 또한 필수적이다.

이와 관련하여 미국에서 fortify와 같은 보안 취약성 도구가 개발되어 산업현장에서 이미 활용되고 있으며[1] 국내적으로는 메모리 오류를 자동으로 검출해주는 정적분석시스템인 Sparrow가 파수닷컴(http://www.fasoo.com)에 의해서 상용화되었고[2], 보안 취약성을 야기하는 소스

코드 내의 구문 패턴과 흐름 패턴을 규칙 명세에 기반하여 검출하는 CodePrism이 지티원((http://www.gtone.co.kr)[3]에 의하여 개발된 바 있다.

이러한 필요성에 부응하여 정부에서도 행정안전부를 중심으로 전자정부 소프트웨어 개발 시에 보안 취약성에 대한 검사를 의무화하는 정책을 추진하고 있다. 이러한정책의 시행을 위해서는 현재까지 알려진 보안 취약성과 검사 방법을 데이터베이스화하고, 이를 바탕으로 소프트웨어를 자동 검사하는 도구를 개발해야 한다. 뿐만 아니라 보안 취약성의 지속적인 갱신 관리체계 구축, 검사의 의무화를 위한 법률 제정, 소프트웨어 개발 시에 취약성이 발생하지 않도록 개발하는 방법(안전한 코딩 표준; secure coding standard)의 보급과 교육 등도 병행되어야 한다. 본 논문은 2009년도 행전안전부의 주관 및 지원 하에 한국인터넷진흥원을 중심으로 (주)지티원, (주)파수닷컴, 한국정보보호학회 소프트웨어보안연구회에서 개발한 소스코드 보안취약성 자동진단 도구의 개발 사례를 소개하고자 한다. 본 과제와 관련하여 규칙 기반 취약성 분석엔진이 개발되었고, 분석엔진과 프로그램 개발자, 검수자 등이 사용할 수 있는 보안 취약성 데이터베이스가 구축되었다.

본 논문의 전체적인 구조는 다음과 같다. 2장에서 분석도구의 전체적인 구조와 분석엔진에 대하여 기술하고 3장에서는 취약성 데이터베이스의 구축에 대하여 기술한다. 4장에서는 개발된 시스템의 활용을 위한 진단 체계에 대하여 설명하며, 5장에서 결론으로 맺는다.

2. 소스코드 취약성 분석 엔진

2.1 취약성 진단도구의 전체 구조

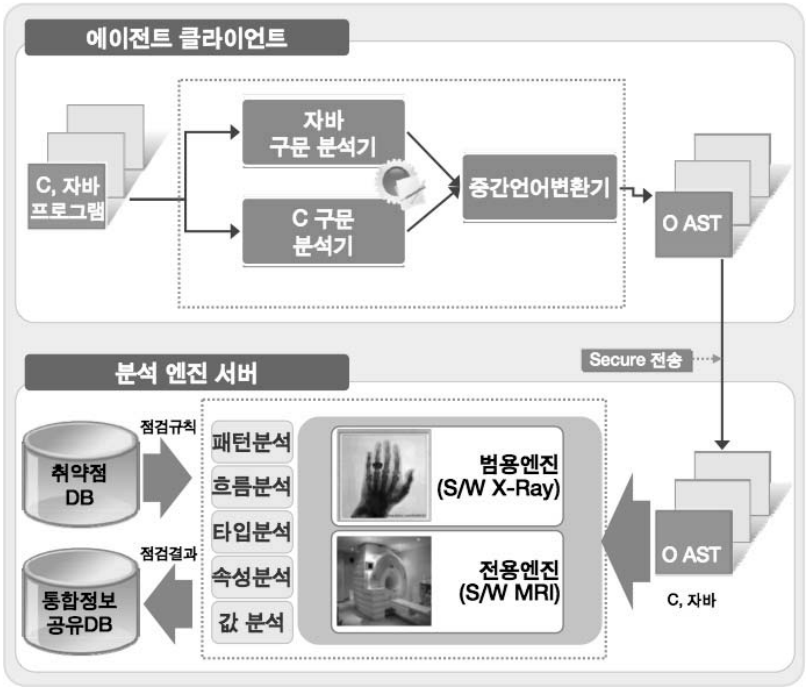
소스코드 취약성 자동진단도구는 [그림 1]과 같이 클라이언트에서 수행되는 1차 분석과 분석엔진서버에서 수행되는 2차 분석으로 나누어진다. 1차 분석에서는 C와 Java 소스를 파싱한 후, 분석에 필요한 구문 및 의미 정보를 망라해서 가지고 있는 중간코드로 변환한다. 소스코드 유출의 우려를 불식시키기 위해 클라이언트 쪽에서 중간코드로 변환하고 암호화하여 분석엔진서버로 전송한다. 중간코드는 유출이 된다 하더라도 형식 및 저장방식이 비공개이므로 중간코드에서 소스코드의 복원은 원천적으로 불가능하다. 2차 분석은 각 취약성의 특성 및 난이도에 따라서 두 가지 방법으로 나누어서 분석한다. 범용엔진은 단순한 구문 패턴이나 흐름 패턴으로 표현할 수 있는 다양한 취약성을 자체개발한 규칙명세언어로 기술하여 목록을 구축하고 이 목록에 수록된 패턴과 일치하는 취약한 부분을 일괄적으로 찾아낸다. 전용엔진은 단순한 패턴으로 표현할 수 없는 취약성에 대해서, 취약성 별로 고안된 분석기가 별도로 구현된 진단 엔진이다. 특정 취약성에 대해서 탐지의 정밀도를 높일 필요가 있는 경우에 전용엔진으로 진단하기도 한다.

개발된 소스코드 취약성 분석엔진은 현재 총 259개의 취약성을 진단할 수 있는데, 이 중에서 Java 취약성은 163개, C 취약성 96개이다. Java의 경우 119개의 취약성이 범용엔진으로, 44개 취약성이 전용엔진으로 진단되고, C의 경우 64개 취약성이 범용엔진으로, 32개 취약성이 전용엔진으로 진단된다.

2.2 범용 분석엔진

범용엔진은 일정한 구문 또는 흐름 패턴으로 취약성을 찾을 수 있는 분석엔진이다. 구문 또는 흐름 패턴은 규칙명세언어 RDL(Rule Description Language)로 기술할 수 있으며, 이 언어로 기술된 취약성 패턴취약성 패턴은 취약성 DB로 유지되고 수시로 갱신할 수 있다.

범용엔진은 진단 대상이 되는 소스코드의 중간코드와 RDL로 기술된 검사규칙목록을 입력으



[그림 1] 소스코드 취약성 분석 엔진

로 받아서, 취약성을 탐지한다. 중간코드는 생성 과정에서 모아둔 타입정보와 같은 프로그램의 기본적인 요소들을 가지고 있으며, 이러한 정보들은 분석에서 사용된다. 상수분석이나 널값여부 분석 등 간단한 수준의 값분석은 성능을 위해서 미리 수행되며, 단일패턴 분석기는 입력받은 규칙명세목록을 단일패턴단위로 쪼개서 각 단일패턴들의 위치와 부가정보들을 모은다. 흐름분석을 위해서 중간코드는 흐름그래프로 변환되고, 흐름패턴 분석기는 단일패턴들의 지점들과 흐름그래프로 흐름패턴으로 명시된 단일패턴들 사이의 관계를 분석하여 취약성 진단 결과를 생성한다.

2.3 전용 분석엔진

전용엔진은 패턴을 RDL로 기술할 수 없거나 정밀분석이 요구되는 경우 취약성별로 개별적으로 구축되는 분석엔진이다. 일반적으로 분석엔진의 정밀도를 높이기 위해서는 복잡한 분석을 감수해야 하고 따라서 분석시간이 많이 걸린다. 따라서 정밀도와 분석시간 사이에 적절한 조정이 필요하다. C의 경우 전용분석으로 처리된 32개의 취약성 중에서 18개 취약성은 값 분석을 자세히 수행하는 Sparrow 엔진을 기초로 제작되어서 정밀도가 이미 상당히 높다. C의 나머지 14개 취약성과 Java의 44개 취약성은 아직 정밀도 향상의 여지가 많이 남아있으며, 공학적이며 경제적인 측면에서 균형유지를 위한 적절한 의사결정이 필요하다.

3. 보안 취약성 데이터베이스

3.1 전체 구조

취약성 데이터베이스는 기본 취약성 관련 정보인 취약성 테이블, 언어별 관련 취약성 정보와 검출 규칙을 정리한 취약성 규칙 테이블 및 취약성과 관련한 공격 패턴을 정리한 공격패턴 테이

블로 구성된다. 본 취약성 데이터베이스는 CWE, CVE, CAPEC 등의 국외 관련 연구 내용[5,6,7,8,9,10]과 행안부 발주 주요 소프트웨어의 특성 및 업계 관련 제품의 성능 등을 참고하여 구축되었으며 xml 형식으로 정리되었다.

취약성 테이블은 취약성 식별자, 설명, 중요도, 관련 언어, 분류, 관련 공격 패턴, 참고문헌 등의 정보로 이루어져 있으며 KCWE(Korea Common Weakness Enu- meration) 일련번호를 부여하였다. 취약성 규칙 테이블은 현재 C 및 Java와 관련된 취약성 각각에 대하여 취약성 규칙 식별자, 취약성 예제 및 수정 예제와 관련설명, 취약성 검출 규칙 및 규칙 설명 등으로 이루어져 해당 언어의 프로그램 개발 시 참고할 수 있다. 또한 취약성 검출 규칙은 RDL로 기술되어, 프로그램 취약성 자동 분석 엔진의 입력으로 사용된다. 공격 패턴 테이블은 공격 패턴 식별자, 공격 패턴 제목, 관련 공격 도구, 공격 패턴에 대한 취약성 테스트 방법 및 취약성 회피 방법 등의 정보를 담고 있다. 취약성 테이블의 각각의 취약성에 대해서는 해당하는 언어별 취약성 규칙이 1:n으로 연결되며, 취약성과 공격 패턴에 대해서는 n:n의 관계로 연관된다.

3.2 보안 취약성 분류

취약성 데이터베이스를 구축함에 있어서 본 연구에서는 취약성들을 원인 및 특성에 따라 분류하여 구축 작업을 수행하였다. 본 절에서는 이러한 분류에 따라 분석 대상이 되는 보안 취약성에 대하여 간단히 설명한다.

3.2.1 검증되지 않은 값의 사용(Data Handling)

본 취약성 분류는 검증되지 않은 부적절한 값이 보안에 민감한 작업에 사용됨으로써 발생하는 취약성을 말한다. 부적절한 값이란 외부의 불특정 입력이나, 범위를 벗어나는 주소값 및 첨자(index) 값, 각각의 자료형(types)의 범위를 벗어나는 값 등이 되며, 민감한 작업이란 메모리 접근, 명령어 수행, 경로값을 사용한 파일(file) 접근, 데이터베이스 접근, 동적 웹페이지의 생성, 형식 문자열(format string)을 사용하는 입출력문 등을 들 수 있다.

본 취약성 분류에 속하는 대표적인 취약성으로는 SQL 삽입 공격(SQL Injection Attack, KCWE-89) 취약성, 사이트 교차접속 스크립트 공격 취약성(Cross Site Scripting, KCWE-79), 메모리 버퍼 범위 조건 위반(KCWE-119) 등이 있다.

3.2.2 API 사용 오류 (API Abuse)

API 사용 오류란 개발환경에서 제공하는 시스템 라이브러리 중 자체에 보안취약성이 있는 것을 사용하거나, 라이브러리에 대한 이해 부족으로 잘못 사용하여 보안취약성을 발생시키는 것을 말한다.

대표적인 API 사용 오류로는 위험하다고 알려진 함수 사용 (KCWE-242), 함수 결과 검사 부재(Unchecked Return Value, KCWE-252), super.finalize()를 호출하지 않는 finalize() 메소드(KCWE-568) 등이 있다.

3.2.3 보안 관련 오류(Security Features)

보안성과 관련된 오류란 패스워드 관리를 담당하는 프로그램이나 암호화 등을 이용하는 과정에서 발생한 프로그램 상의 오류가 시스템 전체를 취약하게 만드는 것을 말한다. 여기에는 사용자 암호를 정하지 않고 사용하는 경우나, 프로그램 내부에서 문자열 상수로 패스워드를 저장하고 그대로 사용하는 경우, 혹은 이미 취약성이 알려진 암호화 기법을 사용하는 경우 등이 포함된다. 보안성과 관련된 오류들은 오류가 발생하

면 사용자 아이디나 패스워드 등이 노출되면서 시스템 전체의 기밀성이나 무결성이 침해되는 경우가 발생할 수 있다. 이 때문에 패스워드를 상수의 형태로 노출하거나 잘못된 암호화 알고리즘을 사용하는 오류는 발생빈도는 낮더라도 발생하는 경우, 침해의 결과는 치명적일 수 있다.

대표적인 보안성 관련 오류로는 코드에 고정된 패스워드(Hard-coded Password, KCWE-259), 취약한 암호화 알고리즘의 사용(KCWE-327), 주석문 안에 포함된 패스워드(KCWE-9302), 적절하지 않은 난수 값의 사용(KCWE-330), 보안 속성이 결여된 HTTPS 세션으로 보내지는 민감한 내용의 쿠키 (KCWE-614), 익명의 LDAP 바인딩(KCWE-9311) 등이 있다.

3.2.4 시간과 상태 관련 오류(Time and State)

시간과 상태에 대한 오류란 프로그램의 동작 과정에서 시간적 개념을 포함한 개념(프로세스, 혹은 쓰레드 등)이나 시스템 상태에 대한 정보(자원 잠금이나 세션 정보)에 관련된 오류를 말한다.

대표적인 시간과 상태에 관련된 오류로는 외부에서 제한 없이 접근 가능한 잠금(KCWE-412), 중복 검사된 잠금(Double-Checked Locking, KCWE-609), 세션 고착(Session Fixation, KCWE-384) 등이 있다.

3.2.5 에러 처리(Error Handling)

프로그램 수행 중에 발생할 수 있는 에러를 부적절하게 처리할 경우 발생할 수 있는 소프트웨어 취약성을 의미한다. 이러한 소프트웨어의 취약성을 공격자가 인지할 경우, 다양한 조작된 입력 값을 주어 프로그램이 에러를 내도록 하여 다른 공격에서 필요한 프로그램의 내부 작동 구조나 내부 정보를 습득할 수 있으며 제대로 에러 상황을 처리하지 않아 프로그램을 크래쉬 시킬 수도 있다.

여기에 속하는 취약성으로는 액션 없는 오류 조건 탐지(KCWE-390), 미검사된 에러 조건(KCWE-391), NullPointer- Exception을 통한 Null 포인터 사용 포착(KCWE-395), 포괄적 예외를 사용한 catch 선언(KCWE-396), 포괄적 예외를 사용한 throws 선언(KCWE-397), Finally 블록 내에서의 리턴(KCWE-584) 등이 있다.

3.2.6 코드 품질이 낮음을 표시하는 요소 (Indicator of Poor Code Quality)

작성된 프로그램에는 직접적으로 결점이나 보안과 관련된 취약성을 나타내지는 않으나 코드가 충분한 주의를 기울여 개발되고 관리되었는지를 표시하는 요소들이 있다. 즉, 프로그램 코드가 너무 복잡하여 관리하기 힘들거나 다른 시스템에 이식하기 힘들도록 되어있다든지 충분한 주의를 기울여 작성되지 않았음을 나타내는 요소가 있다면 이 프로그램에는 안전성을 위협할 취약성들이 코드 안에 숨겨져 있을 가능성이 높다. 이와 같이 코드의 품질을 나타내는 주요 취약성들로는 타입 불일치(KCWE-195, KCWE-9613, KCWE-9614, KCWE-9617), 메모리 누수(KCWE-401), 자원의 부적절한 반환(KCWE-404), 메모리 중복 반환(Double Free, KCWE- 411), Null 포인터 참조(KCWE-476), 더 이상 지원되지 않는 함수의 사용(Use of Obsolete Functions, KCWE-477), 사용되지 않는 코드(Dead Code, Unused Field/Method, KCWE-561, KCWE-9609, KCWE-9610), 중복된 Null 검사 (KCWE-9606), 쓰레드 조기 종료(Premature Thread Termination, KWCE-9620), 포맷 스트링 에러(Format String Error, KCWE-9607, KCWE-9622) 등이 있다.

3.2.7 불충분한 캡슐화(Insufficient Encapsulation)

캡슐화는 데이터와 연산에 대한 강력한 울타리를 치는 것인데, 이것이 실패하면 웹 브라우저 같은 경우에 이동 코드(mobile code)가 다른 이동 코드에 의해서 의도와 다르게 사용될 수 있으며, 서버의 경우에는 검증된 데이터와 검증되지 않은 데이터를 구분하지 못하게 되거나, 허용되지 않는 사용자들간의 데이터 누출이 가능해진다.

본 취약성 분류는 주로 Java 언어와 관련되며, 클래스 이름으로 클래스를 비교하기(KCWE-486), 세션 간의 데이터 누출(Data Leak between Sessions, KCWE-488), 디버깅용 코드 남겨두기(KCWE-489), 공개 메소드가 비밀 배열타입 필드를 반환하기(Private Array-Typed Field Returned From a Public Method, KCWE-495), 신뢰 경계 위배(Trust Boundary Violation, KCWE-501), super.clone을 사용하지 않는 clone 메소드 (KCWE-580) 등이 있다.

3.3 보안 취약성의 중요도 평가

취약성의 위험성을 나타내는 중요도를 계산하여 영향력을 평가하고, 문제 해결에 대한 우선순위 결정을 지원하기 위해 취약성 데이터베이스 내의 보안 취약성들에 대한 중요도 평가체계를 마련하였다. 이러한 작업은 기존의 CVE, CWE, CAPEC 등의 취약성 평가와의 관련성 및 전자정부 시스템 관련 각종 보안성 측정 방법과 연계를 고려하여 설계되었다.

정보보호란 데이터 또는 시스템에 대한 고의적이거나 실수에 의한 불법적인 공개(노출), 변조, 파괴 및 지체로부터의 보호를 의미한다. 즉, 데이터 및 시스템의 기밀성, 무결성, 가용성을 확보하는 것이며, 이들은 보안의 핵심 원칙들로서, 어떤 한 요소가 손상을 받게 되는 경우, 그 조직의 지속적인 존재 여부까지 위협할 수 있다. 따라서 보안 취약성 중요도는 기밀성, 무결성, 가용성을 기준으로 평가가 이루어진다.

기밀성, 무결성, 가용성 평가 메트릭은 3점 척도로서, Complete=1, Partial=0.7, None=0 값으로 정의된다. 각 기준에 대한 가중치 산정은 보안 영향력의 비중을 표현하기 위해 3가지 기준이 모두 균등한 경우 0.333으로, 각 기준에 비중을 둔 경우는 비중을 둔 요소의 가중치는 0.5, 나머지 두 요소의 가중치는 0.25로 할당하였다.

보안 취약성 항목의 중요도는 보안에 영향을 주게 되는 위험성(Severity)을 나타내며, 다음과 같이 S에 의해 계산된다. 계산된 S값의 범위는 0과 5 사이의 값으로 각 취약성 항목은 값에 따라 VeryHigh, High, Medium, Low, VeryLow의 중요도를 갖게 된다.

$$S = 5*(C*w_C) + (I*w_I) + (A*w_A)$$

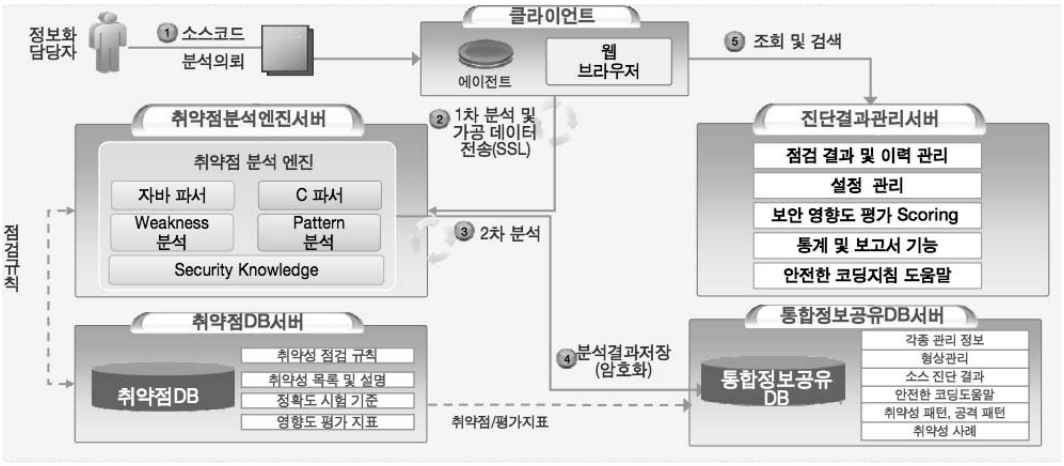
이렇게 계산된 중요도와 공격가능성 메트릭을 이용하여, 각 취약성들이 갖는 보안 영향력 지표 SI(Security Index)를 정의하였다. 취약성 항목별 보안 영향도는 소스코드 취약성의 보안 위험 관리에 대한 지표로 사용될 수 있다. 소스코드 취약성 항목별 보안 영향도 값이 큰 순서로 관리해야 하며, 평가 대상 프로그램에서 검출된 취약성들 가운데 영향도가 높은 취약성들이 존재한다면, 전체 프로그램의 보안 위험수준에 위협요소가 될 가능성이 높아지게 될 것이다.

4. 진단 체계

개발된 진단도구를 사용하여 실제 정부 발주 소프트웨어에 취약성 검사를 적용하는 체계는 [그림 2]와 같다. 전체적인 진단체계의 수행 과정을 설명하면 다음과 같다.

1) 소스코드 분석의뢰

- 정부의 정보화담당자는 클라이언트 에이전



[그림 2] 소스코드 취약성 자동진단 체계

트를 통해서 개발한 소스코드의 분석을 의뢰한다.

2) 1차 분석 및 가공 데이터 전송(SSL)
- 정보화담당자의 클라이언트 에이전트는 분석의뢰 대상 소스코드를 중간코드로 변환한다.
- 변환된 중간코드와 1차분석 도중에 가공된 데이터를 암호화하여 취약성분석엔진서버로 보낸다.

3) 2차 분석
- 취약성분석엔진에 1차분석의 결과가 전달되면 취약성DB로부터 분석에 요청된 최신의 점검규칙들을 가져온다.
- 취약성 분석엔진에서 변환된 분석의뢰 대상 중간코드와 점검규칙을 입력으로 받아서 취약성 분석을 수행한다.

4) 분석결과저장 (암호화)
- 분석된 중간코드의 진단결과를 암호화하여 통합정보공유DB서버로 전송한다.
- 통합정보공유DB서버는 전달된 진단결과를 저장 및 관리한다.

5) 조회 및 검색
- 클라이언트는 진단결과관리서버를 통해 분석이 완료된 코드들의 진단 결과, 보안영향도 평가지표에 따른 평가 성적표, 취약성이 없는 안전한 코딩 제안 등을 조회할 수 있다.

5. 결 론

2009년 행정안전부가 주관하고 한국인터넷진흥원이 수행한 “정보시스템 보안강화체계 구축” 사업은 정보화 프로세스에서 보안을 고려한 관리체계 구축을 통해 운영하기 전에 취약성 사전 제거로 보안 사고를 예방하고 이에 선도적으로 대응할 목적으로 시행한 사업이다. 이 사업의 결과로 개발된 소스코드 취약성 자동진단도구는 전자정부시스템에서 보안결함 발생을 최소화하는데 기여하게 될 것이며, 궁극적으로 국가 소프트웨어 보안수준 향상을 위한 안전한 소프트웨어 개발체계를 정립하는데 초석이 될 것이다.

앞으로 해야 할 과제로 다양한 언어에 대한 확장 작업과 추후로 드러나는 취약성에도 탄력적으로 대응하기 위한 체계 구축을 수행할 예정이며, 진단도구의 정밀도 향상을 위한 분석 기술 개발을 수행할 예정이다.

참 고 문 헌

[1] Fortify, http://www.fortify.com
[2] Sparrow, http://www.spa-arrow.com
[3] 코드프리즘, http://www.gtone.co.kr
[4] Hyunha Kim, Tae-Hyoung Choi, Seung-Cheol Jung, Oukseh Lee, Kyung-Goo Doh, Soo-Yong Lee, "Rule-based Source-code Analysis for Detection of Security Vulnerability", WISA2009:The 10th International Workshop on Information Security Applications, Busan, South Korea, August 25~27, 2009
[5] CWSS - Common Weakness Scoring System, http://cwe.mitre.org/cwss/
[6] Common Vulnerabilities and Exposures, http://cve.mitre.org
[7] Common Vulnerability Scoring System, http://www.first.org/cvss/
[8] CAPEC : Comon Attack Pattern Enumeration and Classification, http://capec.mitre.org/
[9] 2009 CWE/SANS Top 25 Most Dangerous Programming Errors, http://cwe.mitre.org/top25/
[10] Common Weakness Enumeration, http://cwe.mitre.org

권 현 준

1986 서울대학교 자연과학대학 수학과 학사
1990~1998 펜타컴퓨터 Senior 소프트웨어 엔지니어
1998~2008 아이티플러스 CTO
2008~현재 지티원 정보통신연구소 개발1본부장
관심분야: 소프트웨어 공학, 소프트웨어 구조분석, 애플리케이션 보안

김 유 경

1991 숙명여자대학교 수학과(학사)
1994 숙명여자대학교 전산학과 (석사)
2001 숙명여자대학교 컴퓨터과학과 (박사)
2001~2005 숙명여자대학교 정보과학부 컴퓨터과학전공 초빙교수
2005~2006 University of California Davis, 박사후연구원
2006~현재 한양대학교 공학대학 컴퓨터공학전공 연구교수
관심분야: 소프트웨어 품질평가, 웹서비스 신뢰성 평가, SOA 모델링

김 현 하

2003 한양대학교 전자컴퓨터공학부 (학사)
2005 한양대학교 컴퓨터공학과 (석사)
2006~현재 한양대학교 컴퓨터 공학과 (박사과정수료)
관심분야 : 프로그램분석, 소프트웨어보안

도 경 구

1980 한양대학교 산업공학과(학사)
1987 Iowa State University, Computer Science(석사)
1992 Kansas State University, Computer Science(박사)
1993~1995 University of Aizu 교수
2005~2006 University of California Davis 방문교수
1995~현재 한양대학교 ERICA캠퍼스 교수
관심분야: 프로그래밍언어, 프로그램분석, 소프트웨어 보안, 소프트웨어공학

신 승 철

1987 인하대학교 전산학과 (학사)
1989 인하대학교 전산학과 (석사)
1996 인하대학교 전산학과 (박사)
1999~2000 Kansas State University 박사후연구원
1996~2006 동양대학교 컴퓨터공학부 교수
2006~현재 한국기술교육대학교 컴퓨터공학부 교수
관심분야: 프로그램 분석 및 검증, 소프트웨어 보안, 수리논리

안 준 선

1992 서울대학교 계산통계학과(학사)
1994 KAIST 전산학과 (석사)
2000 KAIST 전산학과 (박사)
2000~2001 KAIST 프로그램분석 시스템연구단 연구원
2001~현재 한국항공대학교 항공전자 및 정보통신공학부 교수
관심분야: 프로그램 분석, 소프트웨어 보안, 유비쿼터스 컴퓨팅, 프로그램 병렬화

이 욱 세

1995 KAIST 전산학과 (학사)
1997 KAIST 전산학과 (석사)
2003 KAIST 전산학과 (박사)
2003~2004 서울대학교 컴퓨터 공학과 박사후연구원
2004~현재 한양대학교 컴퓨터공학과 교수
관심분야: 프로그램 분석, 포인터 분석, 프로그램 검증, 타입 시스템

이 은 영

1996년 고려대학교 전산과학과 (학사)
1998년 고려대학교 전산과학과 (석사)
2004년 Princeton University, U.S.A. (박사)
2005년~현재 동덕여자대학교 컴퓨터학과 조교수
관심분야: 프로그래밍언어, 소프트웨어 보안, 컴파일러

한 환 수

1993 서울대학교 컴퓨터공학과 (학사)
1995 서울대학교 컴퓨터공학과 (석사)
2001 University of Maryland, Computer Science (박사)
2001~2002 Intel, Senior Engineer
2003~2008 KAIST 전산학과 교수
2008~현재 성균관대학교 정보통신공학부 교수
관심분야: 컴파일러, 컴퓨터구조, 멀티코어 컴퓨팅

프로그래밍언어연구회 회칙

제 1 조 (명칭) 본 회는 한국정보과학회 프로그래밍 언어 연구회라 칭한다.

제 2 조 (목적) 본 회는 회원 상호간의 학술 활동과 친목을 목적으로 한다.

제 3 조 (회원의 자격) 본 회의 회원은 한국정보과학회 회원으로서 프로그래밍 언어 분야에 관심이 있는 자로 한다.

제 4 조 (회원의 의무) 회원은 소정의 회비를 납부하고 본 회의 사업에 적극 참여할 의무를 갖는다.

제 5 조 (기구) 본 회는 총회와 운영위원회를 둔다.

제 6 조 (총회) 총회는 본 회의 최고 의결기관이다. 총회는 정기총회와 임시총회로 구분하며, 정기총회는 매년 1회 소집하는 것을 원칙으로 하고 임시총회는 운영위원장(이하 "위원장"이라 한다)이 소집할 수 있다. 총회의 의장은 위원장이 겸한다.

제 7 조 (고문) 본 회의 고문을 추대할 수 있다. 고문의 자격, 대상, 인원수, 혜택 등은 운영위원회에서 결정하고 고문의 추대는 총회의 의결을 거친다. 고문은 본 회 운영에 관하여 위원장의 자문에 응하여야 한다.

제 8 조 (운영위원회) 본 회의 회무를 수행하기 위하여 다음과 같이 운영위원회를 둔다.

1. 위원장: 1인
2. 부위원장: 1인
3. 편집위원장: 1인
4. 운영위원: 편집위원(4인 이내)을 포함하여 15인 이내
5. 감사: 2인 이내

제 9 조 (운영위원의 직무) 위원장은 본 회를 대표하며, 회무를 통괄한다. 부위원장은 위원장을 보좌하여 위원장 유고시 그 직무를 대행한다. 편집위원장은 편집위원회를 구성하여 본 회의 발행지 편집에 관한 직무를 관할한다. 운영위원은 총무, 재무, 학술사업, 편집 등의 회무를 담당한다. 감사는 본 회의 업무 및 회계를 감사한다.

제10조 (운영위원의 선출) 위원장은 정기총회 출석회원의 과반수의 득표로 선출한다. 부위원장, 편집위원장 및 운영위원은 위원장이 임명하되 편집위원은 편집위원장의 추천을 받아 위원장이 임명한다. 감사는 정기총회에서 전임 및 전전임 위원장으로 선임한다.

제11조 (임기) 위원장의 임기는 1년으로 한다. 단 1회에 한하여 연임할 수 있다.

제12조 (재정) 본 회의 재정은 회원의 회비, 찬조금, 기타 수입금으로 충당한다.

제13조 (예산과 결산) 위원장은 매년 정기총회에서 예산과 결산의 승인을 얻어야 한다.

제14조 (기타) 기타 사항은 한국정보과학회 회칙에 따른다.

부 칙

1. (효력발생) 본 회의 규정은 2005년 11월 19일부터 효력을 발생한다.

프로그래밍언어논문지 투고 및 심사 규정

제 1 조 연구회지에 게재할 원고의 종류는 프로그래밍 언어에 관련한 논문(정규논문, 초청논문, 학술대회논문의 수정 및 증보판 등), 학술강좌(기술해설, 기술소개, 기술보고, 튜토리얼 등), 특별기고(서평, 학술대회 참관기, 연구기관 방문기 등) 및 기타 편집위원회가 인정하는 것으로 한다.

제 2 조 투고자는 원칙적으로 본회 회원에게 한한다.
단 회원과의 공동기고자 및 초청기고자는 예외로 한다.

제 3 조 국내외 타학술지에 게재되었던 원고는 원칙적으로 투고할 수 없다. 단 편집위원회가 인정하는 경우는 예외로 한다.

제 4 조 투고된 원고는 다음과 같은 조건을 구비하여야 한다. 이 조건을 갖추었는지의 여부는 편집위원회에서 결정한다.
가. 프로그래밍 언어 및 시스템 분야에 관련되는 내용일 것
나. 학술, 교육 및 산업발전에 기여하는 내용일 것

제 5 조 원고 접수는 전자 우편을 이용하여 수시로 하며 접수일은 본 연구회지 편집위원들 중 일인 이상에게 도착한 날로 한다.

제 6 조 원고는 원칙적으로 한글로 쓰되 가능한 한 널리 쓰이는 한글 워드프로세서를 시용하여 A4 용지에 한 줄 건너서(double sapcing) 편집하고, 그림 및 표를 포함하여 가급적 30면 이내로 한다.

제 7 조 연구 내용에 직접 관련이 있는 문헌에 대해서는 이들 문헌에 관련이 있는 본문 중에 참고문헌 번호를 쓰고 그 문헌을 참고문헌란에 반드시 기술한다.
참고문헌은 학술지의 경우는 저자, 표제, 학술지명, 권, 호, 면수, 발행년의 순서로, 단행본은 저자, 서명, 면수, 발행소, 발행년의 순서로 기술한다.

제 8 조 본문의 경우 한글과 영문으로 작성된 제목, 저자성명, 초록을 포함해야 한다.

제 9 조 논문의 경우 편집위원회가 2인의 심사위원을 선정하여 심사를 거쳐 게재 여부를 결정하며 필요하면 수정을 요구할 수 있다.

제10조 본 규정은 2000년 9월 1일부터 효력을 발생한다.

프로그래밍언어연구회 소식 및 기사투고 안내

프로그래밍언어연구회에서는 회원 여러분의 유익한 소식 및 기사들을 기다리고 있습니다. 프로그래밍 언어 이론 및 응용 연구 결과, 논문, 특별 기고(본인의 견해, 전망, 분야별 개관(survey), 특정 분야의 소개, 제품 소개), 국내 기사(회원 동정, 회의 결과, 제품 개발 안내), 해외 기사, 연구 기관 소개, 책 리뷰, 특별 모임 안내, 세미나 안내 등에 대한 기사를 보내주시기 바랍니다. 프로그래밍언어 논문지는 회원들간의 소식을 전달하는 매체로서 연구회지와 연구회 web 페이지를 병행할 계획을 하고 있습니다. 원고를 web을 통하여 접수하고, 회원들에게 필요한 web을 이용하여 수시로 제공하는 체제를 만들려고 합니다. 이러한 것이 가능하기 전까지는 기존의 방법대로 원고를 접수하겠습니다. 아래 편집위원 중 한 분에게 원고를 보내주시기 바랍니다.

- 서울대학교 백윤흥: ypaek@ee.snu.ac.kr (Tel. 02-880-1742)

프로그래밍언어연구회 연회비 안내

연회비는 프로그래밍언어논문지 발행 및 운영에 필요한 경비입니다. 본 회의 회원께서는 다음과 같이 납부하여 주시기 바랍니다.

- 연회비 : 일반회원 10,000원/년
  학생회원 5,000원/년
- 납부처 : 조흥은행 313-03-002919 (예금주 : 사단법인 한국정보과학회)
- 문 의 : 동덕여자대학교 이은영: elee@dongduk.ac.kr (Tel. 02-940-4588)

프로그래밍언어연구회 가입 안내

입회 원서를 작성하신 후 입회비(일반 : 10,000원, 학생 : 5,000원)를 다음의 방법으로 지불하고 입회원서 및 회비 납부 사본을 같이 본 연구회 총무에게 보내주시기 바랍니다. 특정 단체의 회원 가입에 대해서도 총무에게 문의하시길 바랍니다.

한국항공대학교 안준선 : jsahn@kau.ac.kr (Tel. 02-300-0144)

입 회 원 서		
성 명	한 글	
	영 문	
연 락 주 소		
근 무 처	기 관	
	주 소	
정보과학회 회원입니까?		예( ) 아니오( )
관 심 분 야		

본인은 프로그래밍언어연구회의 취지에 회원으로 가입하고자 합니다.

20 년 월 일

성명 : 인

변경사항 통지서		
성 명	한 글	
	영 문	
변경사항	(주소, 근무처, 전화번호)	

발행인 : 변석우
편집인 : 백윤흥, 안준선, 박성우, 방기석, 우균, 한경숙, 이수현
인 쇄 : 2009년 12월 25일
발 행 : 2009년 2009년 12월 25일

발행소 : 대종파이오 (Tel. 02-2631-0146)
발행처 : 한국정보과학회 프로그래밍언어연구회