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.

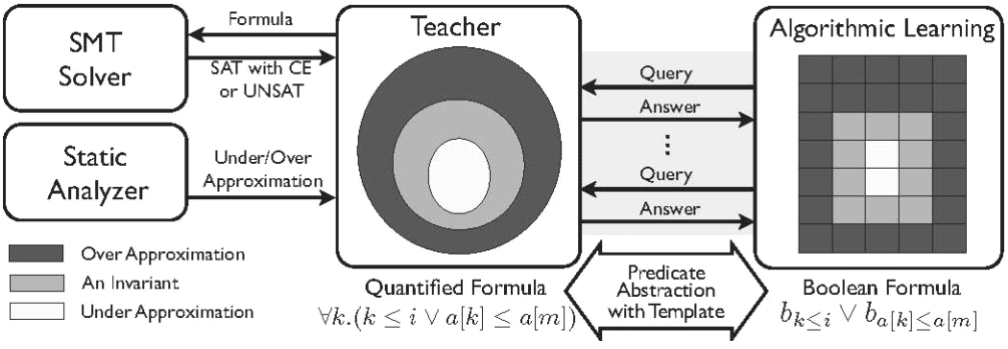
우리는 적합한 조건식 η를 찾아서 $t[\eta]$가 주어진 반복문에 불변성이 되도록 하려 한다. CDNF 알고리즘은 이 η를 찾으면 된다.

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).

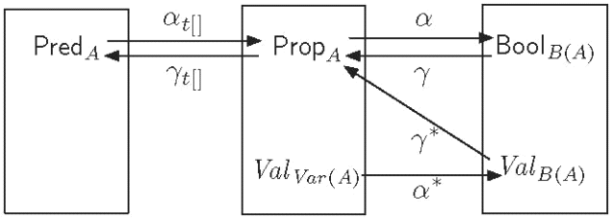
4.1 거푸집을 이용한 조건식 요약

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

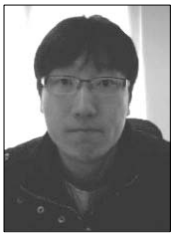
[7] Bertot, Y., Castéeran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. Springer Verlag (2004)

[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