

효율적인 검증 속성 위반 탐지를 위한 PBE 기반 함수요약 및 정제 기법

김요엘, 최윤자

연구 배경

추상화 기법: 대상 프로그램을 추상화하여, 상태 공간을 줄여 검증 효율을 향상시키는 기법

- 그러나, **coarse한 (매우 작고 부정확한) 추상 모델에서부터** 시작

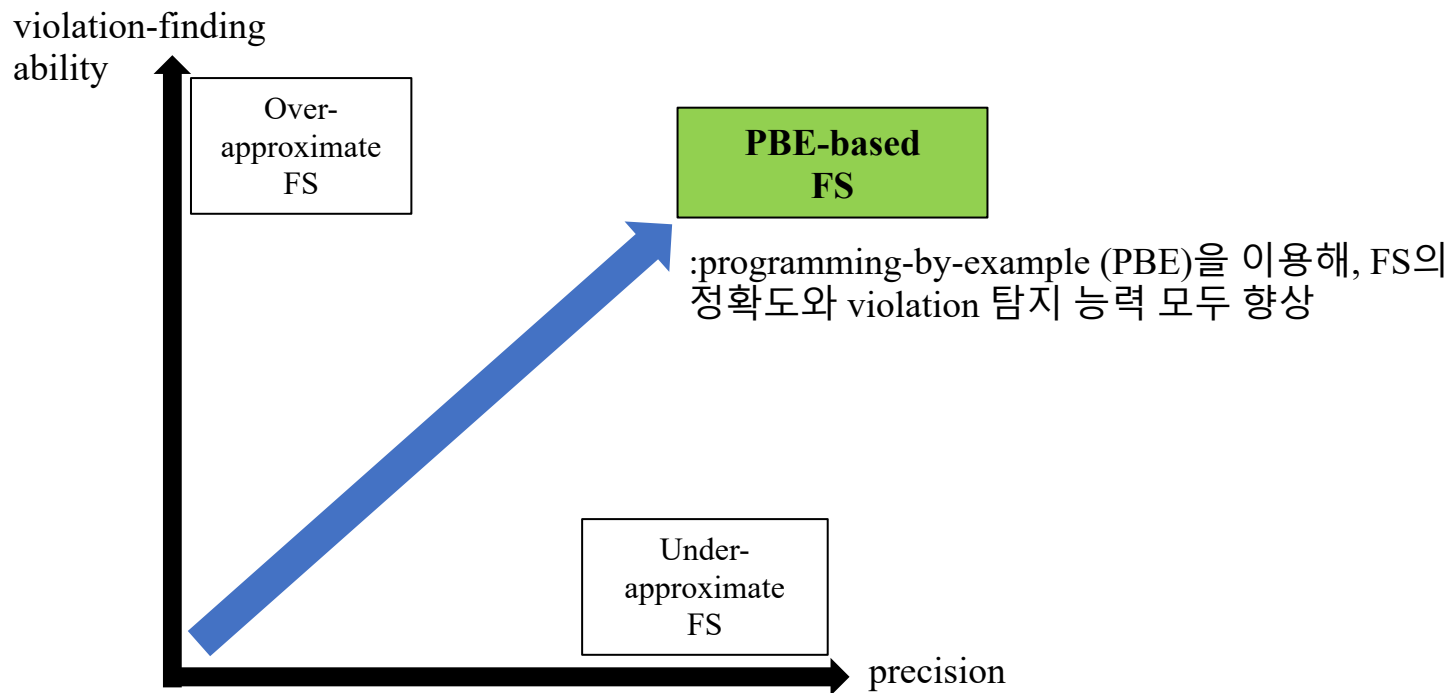
→ 실제로는 재현이 불가능한 반례를 많이 만들게 됨 (즉, 수 많은 정제 과정이 필요)

→ 프로그램 전반에 걸쳐서 필요 이상으로 추상화되고 있음

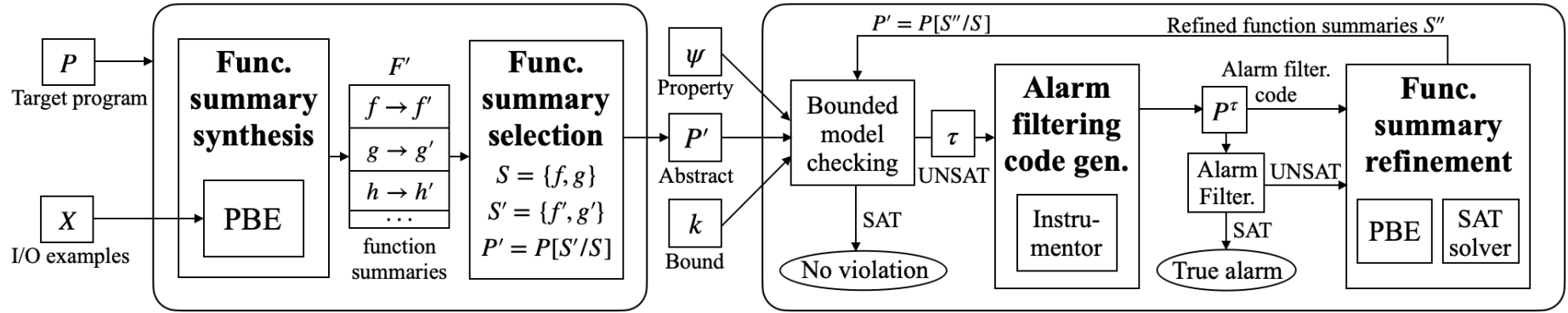
함수요약 (Function Summary, FS)의 적용: 전체 프로그램이 아닌, 함수 단위로 요약 (추상화)하면 충분히 효율적이면서도, 추상화되지 않은 나머지 코드들이 정확도에도 기여할 수 있음

연구 동기

- Over-approximate FS: 함수의 모든 행동을 포함하나, 불필요한 행동들도 포함
- Under-approximate FS: 함수의 정확한 행동만을 포함하나, 수집된 몇몇 입력만 처리 가능



PBEAR



주요 논의:

1. **FS synthesis:** FS을 어떻게 합성할 것인가?
2. **FS selection:** 프로그램에서 어떤 함수를 FS로 대체할 것인가?
3. **FS refinement:** 정제된 FS가 충분히 잘 정제 되었는가?

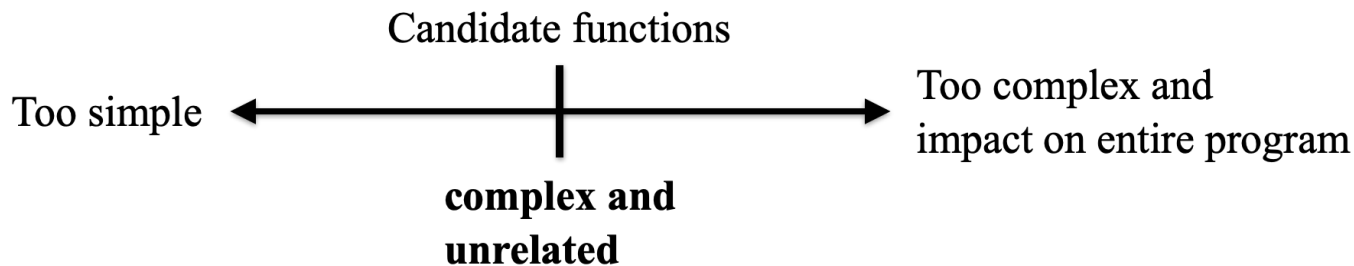
함수요약 합성

FS의 주요 measurement:

출력 범위 $\approx$ violation 탐지 능력; 복잡도 $\approx$ 검증 효율성; 정확도 $\approx$ true/false alarm 비율

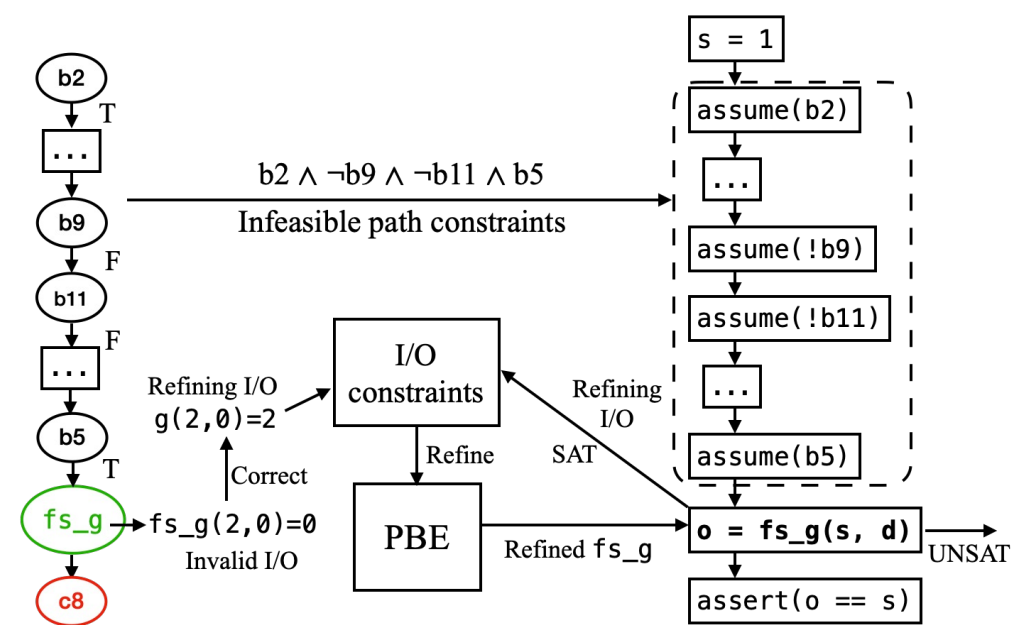
- 위 3개 조건을 동시에 만족하기 위해, *best-effort* 패러다임 채택:
 - 원본 함수와 똑같은 함수를 합성하고 싶은 것이 아님 (*all-or-nothing*)
 - 최대한 출력 범위가 넓은, 원본 함수 대비 간단한, 어느정도 정확한 FS을 합성
- CEGIS (counterexample-guided inductive synthesis):
 - Iterative하게 합성하면서 이전 FS의 평가 결과를 다음 FS을 합성하는데 피드백 정보로 사용

함수요약 선택 기준



- 기준 1: 검증 효율에 도움이 되지 않을 것 같은 함수는 FS로 대체 X
 - 예 1: 너무 단순한 함수
 - 예 2: 너무 복잡하고 프로그램에 전반적인 영향을 미치는 함수
- 기준 2: FS가 어디서 호출되는지를 고려해야 함
 - 예: 전체 실행에서 한 번만 호출되고 끝나는 함수 vs. loop 안에서 여러번 호출되는 함수

함수요약 정제 (주요 발전 내용)



(a) Infeasible path (b) Refinement (c) SAT solving

1. 반례 트레이스에서 FS의 입출력값 수집 → 수집된 입출력값을 교정 → PBE를 통한 FS 정제
2. 정제된 FS가 도달 불가능한 경로 제약 조건하에서 속성 위반을 찾을 수 있는지 확인 (SAT solving)
3. 속성 위반을 찾았다면 (SAT), 과정 1로 이동 (여전히 정제된 FS의 잘못된 행동이 존재하므로)
4. 그렇지 않다면 (UNSAT), 다른 path의 탐색이 보장되는 정제된 FS 반환

실험: 3개 embedded SW, 17개의 검증 속성

P	ψ	PBEAR						CBMC			CBMC -refine			
		k	Filer.(s)	Refine.(s)	#R	Total (s)	Result	k	Total.(s)	Result	k	Total.(s)	#R	Result
TS1	p9	50	8.6	0.0	-	634.8	violated	50	3207.0	violated	50	1009.1	288	violated
	p6	50	3.5	0.0	-	1146.3	violated	50	3248.2	violated	50	899.0	289	violated
	p2	60	22.6	0.0	-	2275.4	violated	60	12062.1	violated	60	2772.6	726	violated
	p10	60	14.6	0.0	-	2292.3	violated	60	8823.0	violated	60	2538.9	598	violated
	p4	60	19.2	0.0	-	2809.1	violated	60	12241.3	violated	60	2635.0	591	violated
	p7	70	0.0	0.0	-	3824.0	violated	70	29075.4	violated	70	5217.5	780	violated
	p1	130	0.0	0.0	-	17469.4	OOM	110	12481.1	OOM	120	24302.7	-	OOM
	p3	110	219.3	0.0	-	33212.3	violated	110	13921.4	OOM	110	30278.0	1537	violated
	p8	120	337.2	0.0	-	47533.4	violated	110	14812.3	OOM	120	37168.3	-	OOM
	p11	110	288.6	0.0	-	49534.8	violated	110	21064.2	OOM	110	39789.9	1556	violated
TS2	p1	610	10.1	8.6	1	9749.7	violated	360	30120.0	OOM	360	31166.8	-	OOM
	p2	610	20.6	38.4	3	10047.3	violated	360	29255.3	OOM	360	30765.4	-	OOM
	p3	610	16.4	21.7	2	115357.0	violated	360	105237.3	OOM	340	181168.5	-	OOM
TS3	p1	20	0.6	0.0	-	165.3	violated	20	3657.1	violated	20	3678.4	-	violated
	p2	30	0.9	0.0	-	825.1	violated	30	41070.0	violated	30	42029.7	-	violated
	p3	70	0.0	0.0	-	20823.9	OOM	50	15007.5	OOM	50	15192.9	-	OOM
	p4	60	0.0	0.0	-	186958.9	OOM	40	47635.8	OOM	40	47009.3	-	OOM

- **Performance:** CBMC에 비해 평균 약 8.1배, CBMC –refine에 비해 1.4배 속도 향상 (그러나, TS1는 PBEAR가 더 느림)
- **# of refinements:** total 6번 (PEBAR) vs. 6,365번 (CBMC –refine)
- **Scalability:** out of memory (OOM)때문에 찾지 못했던 4개 (CBMC –refine)의 속성 위반을 더 찾음.