V8 JS 엔진의 JIT 컴파일러를 위한 번역검산

권승완, 권재성, 강우석, 이준영, 허기홍

2024 Winter ERC Workshop / 01.30

Programming Systems Laboratory
KAIST
aws

V8 JS 엔진

연구 동기

V8: 구글에서 만든 자바스크립트(JS) 엔진

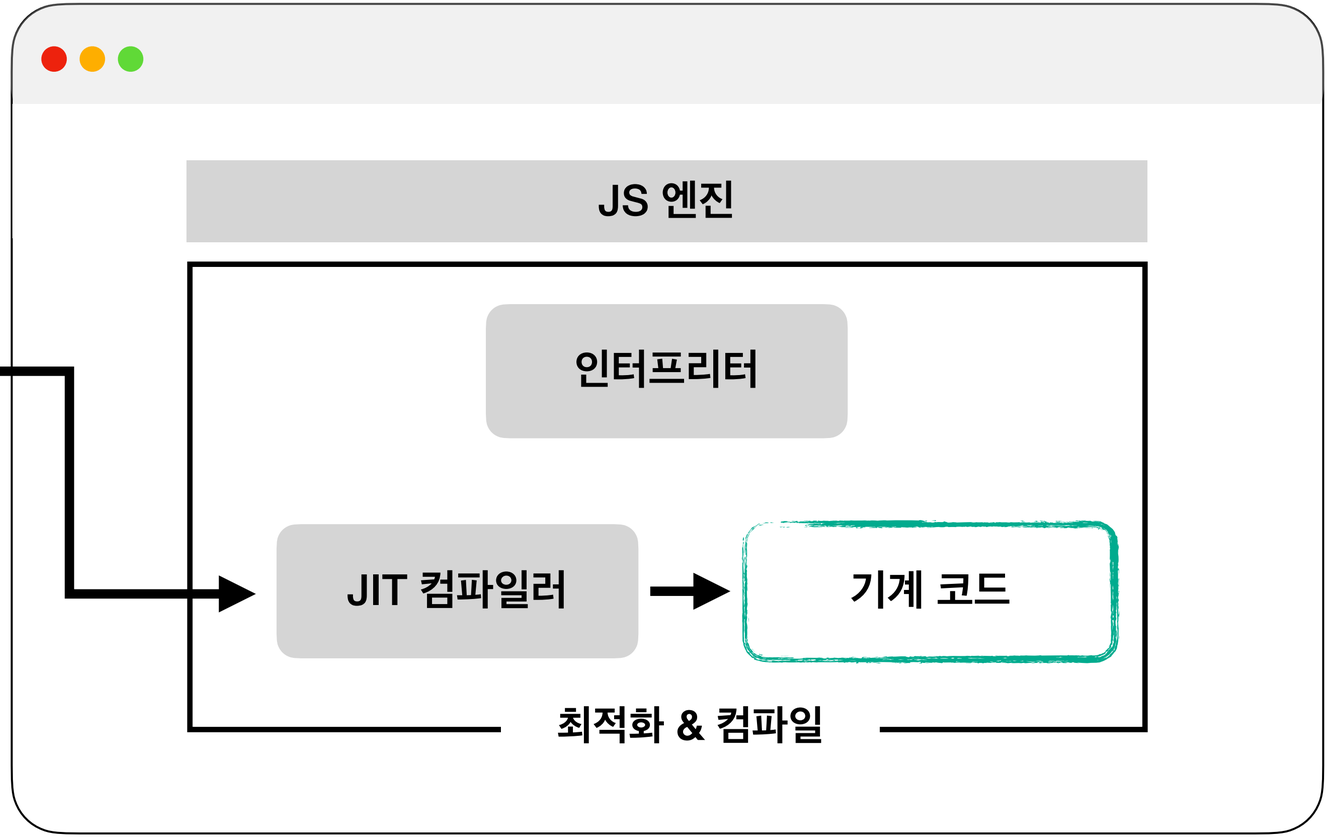
Chrome, Microsoft Edge, Node JS, Deno JS, Electron 등 다양한 곳에서 사용

TurboFan: V8의 JIT 컴파일러

연구 동기

```
function add(x, y) {
    return x + y;
}

for (i=0; i<1000000; i++){
    add(1,0);
}
```

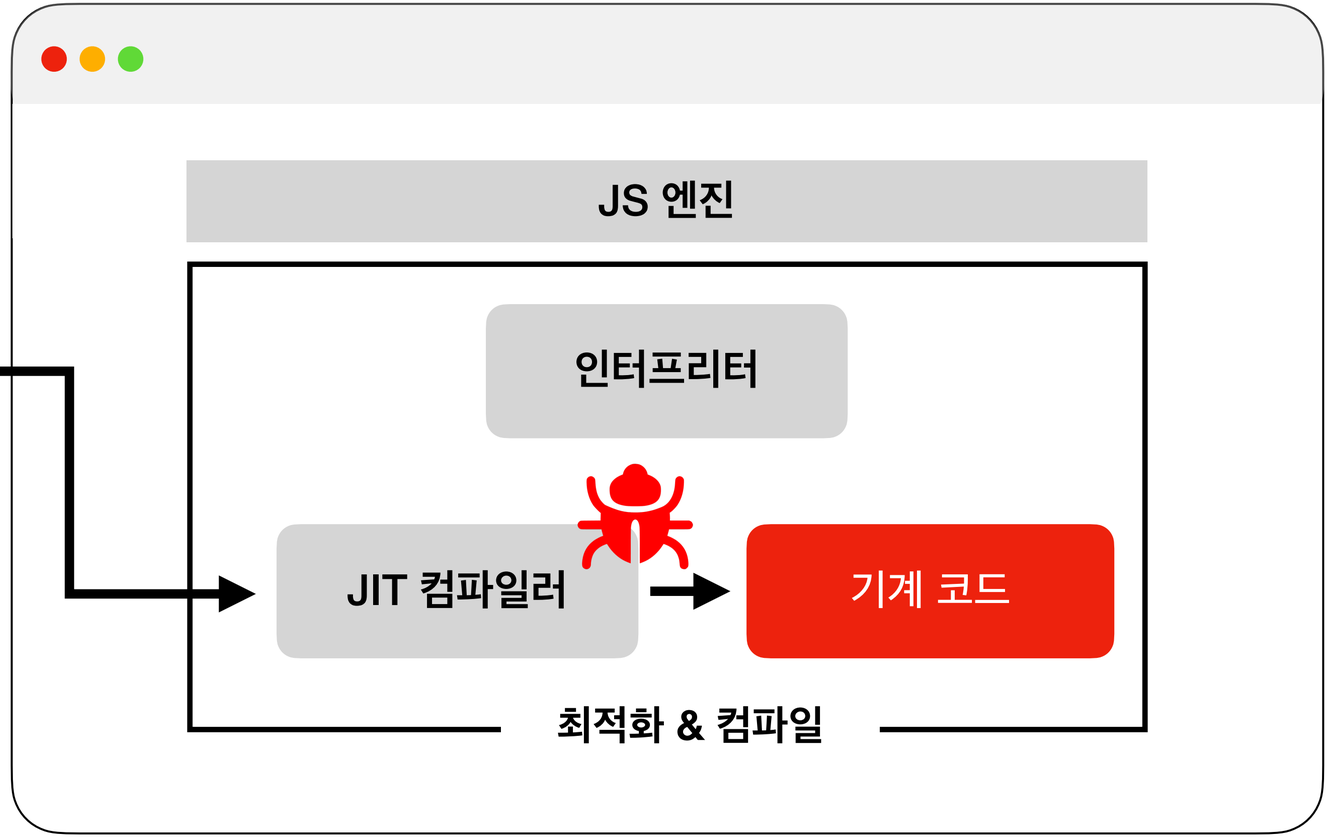


TurboFan: V8의 JIT 컴파일러

연구 동기

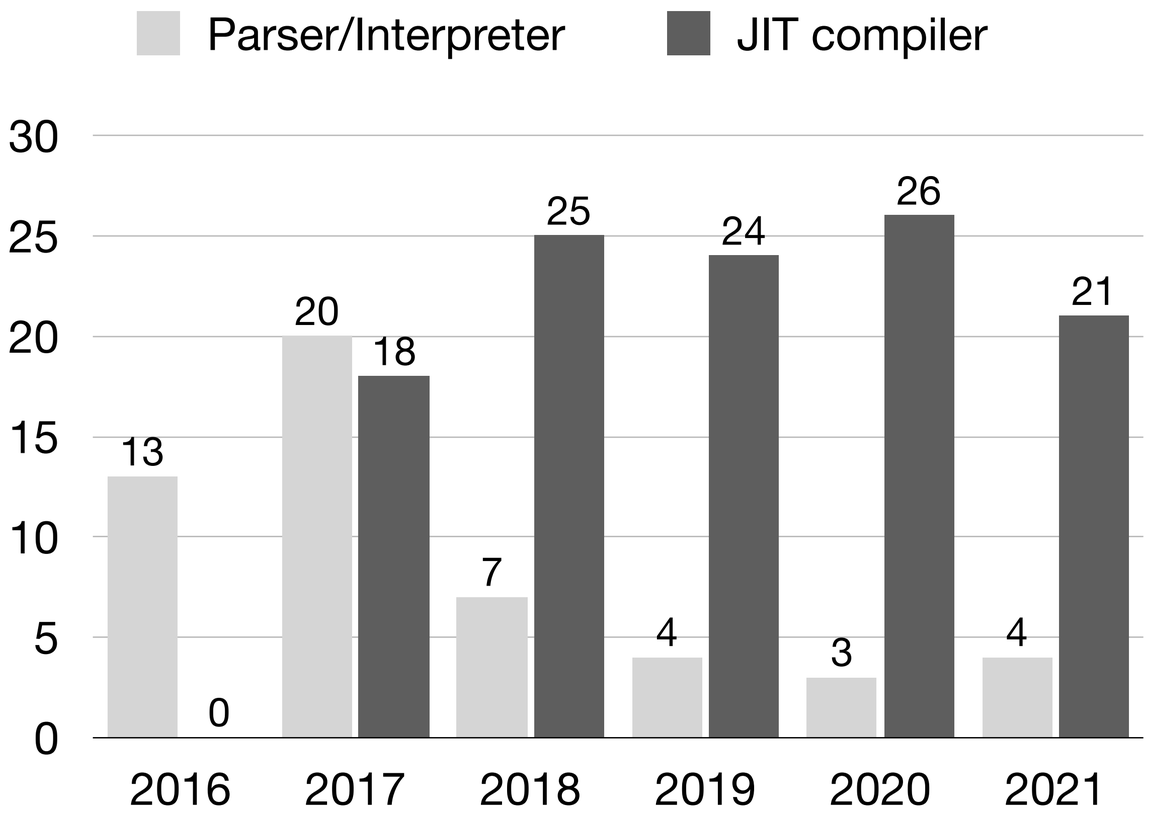
```
function add(x, y) {
    return x + y;
}

for (i=0; i<1000000; i++){
    add(1,0);
}
```



JIT 컴파일러의 버그

연구 동기



JS엔진 파서/인터프리터와 JIT 컴파일러에서 구글 프로젝트제로가 발견한 버그

J Wang et al. 2023. Oracle-Enhanced Fuzzing for JavaScript Engine JIT Compiler. In USENIX security

컴파일러 검증의 어려움

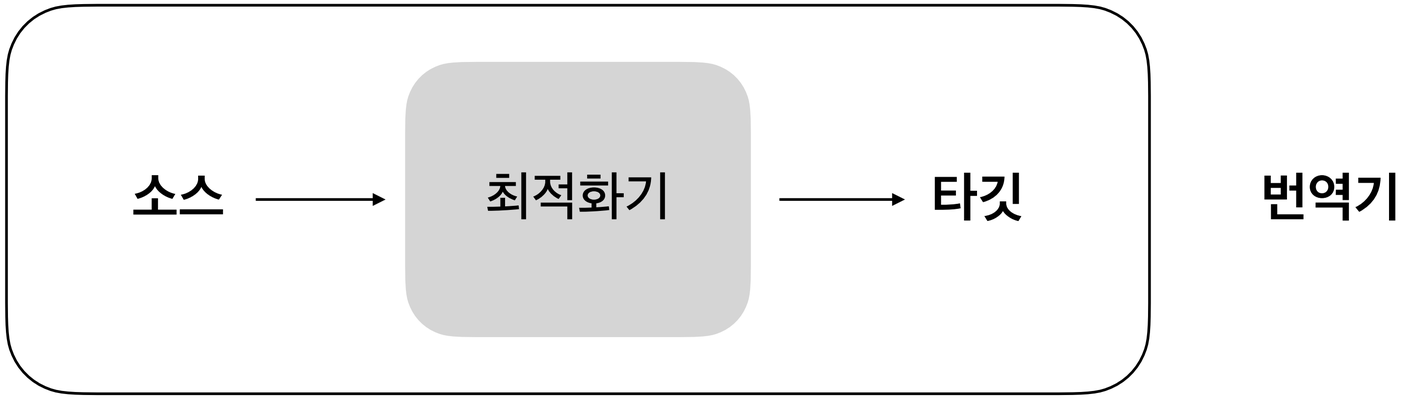
V8의 복잡성

- **2.3M** 이상의 코드 라인 수
- **16**개의 언어로 구현
- **900**명 이상이 구현에 기여
- 매 일 **6**번 이상의 코드 변화
- **TurboFan**: V8 자바스크립트 엔진의 핵심 JIT 컴파일러

번역 검산

번역이 프로그램의 의미를 보존하는가

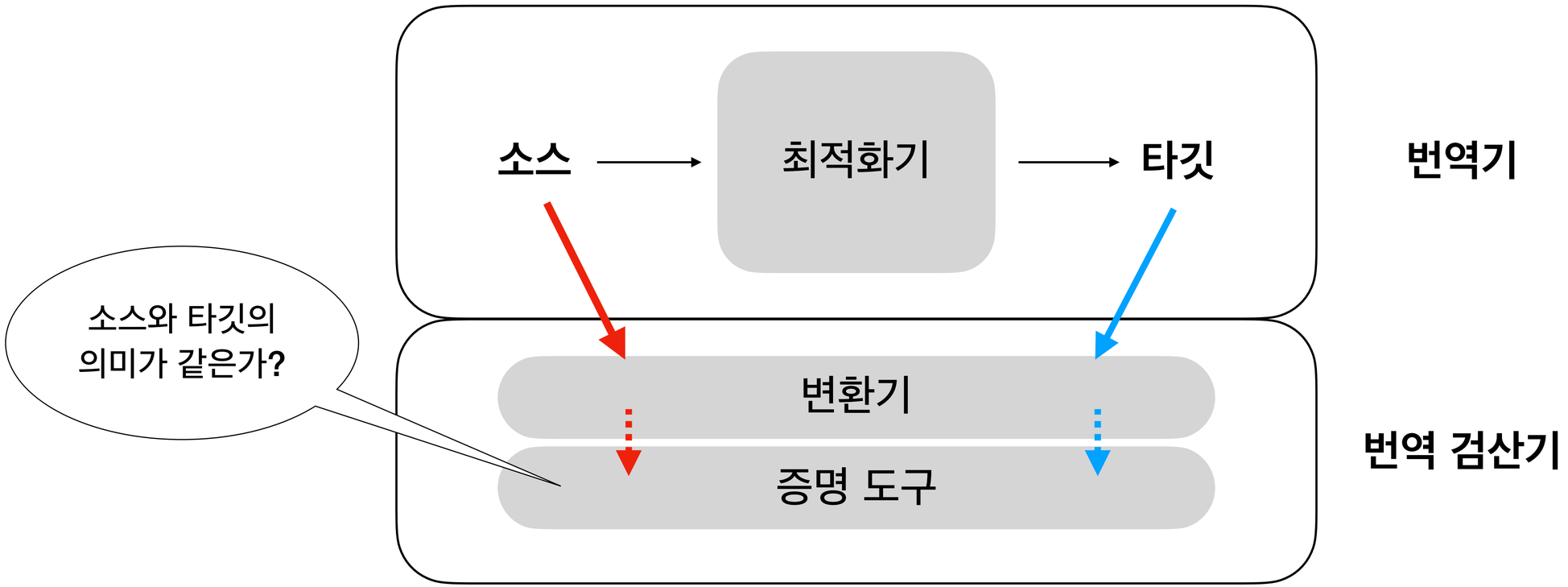
- 번역기의 내부 구현과 무관
- 목표 언어의 의미에만 집중



번역 검산

번역이 프로그램의 의미를 보존하는가

- 번역기의 내부 구현과 무관
- 목표 언어의 의미에만 집중



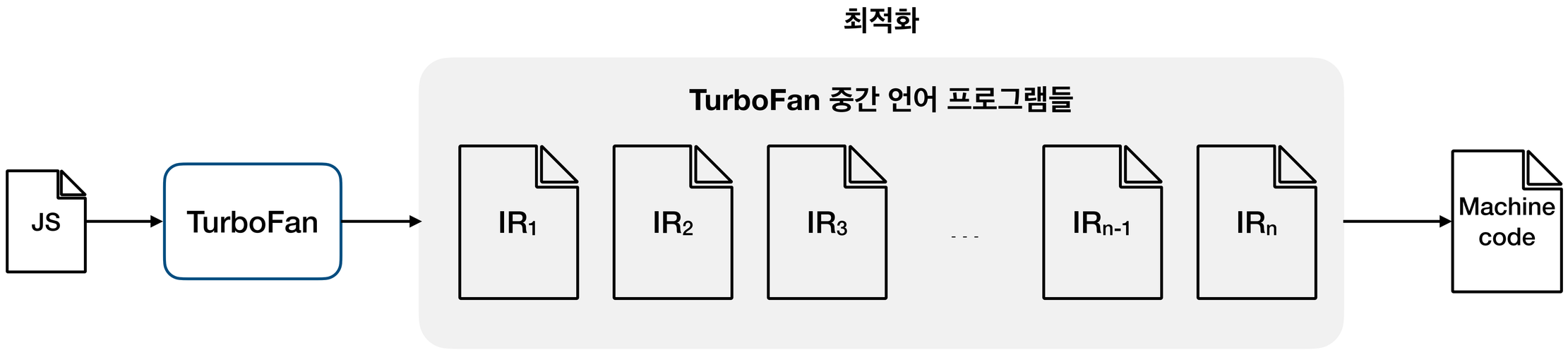
TurboTV

TurboFan 번역 검산 시스템

TurboFan 최적화

최적화 과정에서 프로그램의 의미는 유지되어야 한다.

- **목표:** TurboFan이 최적화 과정에서 프로그램의 의미가 **보존**되는지 검사



JIT 컴파일러 번역 검산의 특수성

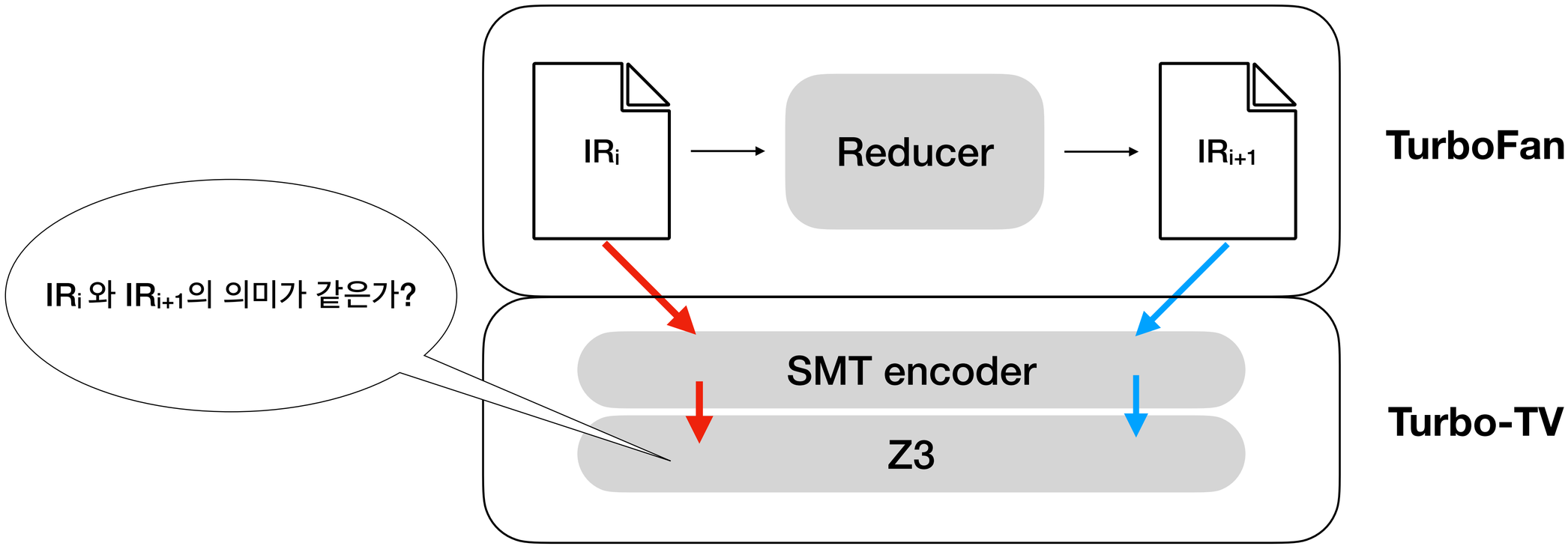
동작 과정

- 다른 AOT 컴파일러(예: gcc, llvm)와 달리, JIT 컴파일러는 **런타임** 중 컴파일 실행
- AOT 컴파일러의 번역 검산은 **컴파일 실행 단계**에서 사용 가능
- JIT 컴파일러는 컴파일 시간에 더욱 **민감**, 현장에서 사용되는 단계에서 번역 검산이 어려움

➡ JIT 컴파일러의 **개발, 유지 보수 단계**에서 올바른 최적화를 위한 번역 검산

TurboFan의 최적화가 의미를 보존하는가

TurboTV



Reducer: TurboFan의 최적화기

TurboTV

TurboFan의 올바른 최적화를 위한 번역 검산 시스템

- TurboFan의 중간 언어의 의미를 **엄밀**하게 정의
  - 최근 보고된 TurboFan 최적화 버그 **9**개를 탐지 성공, 오탐 개수 **0**개

TurboTV

TurboFan의 올바른 최적화를 위한 번역 검산 시스템

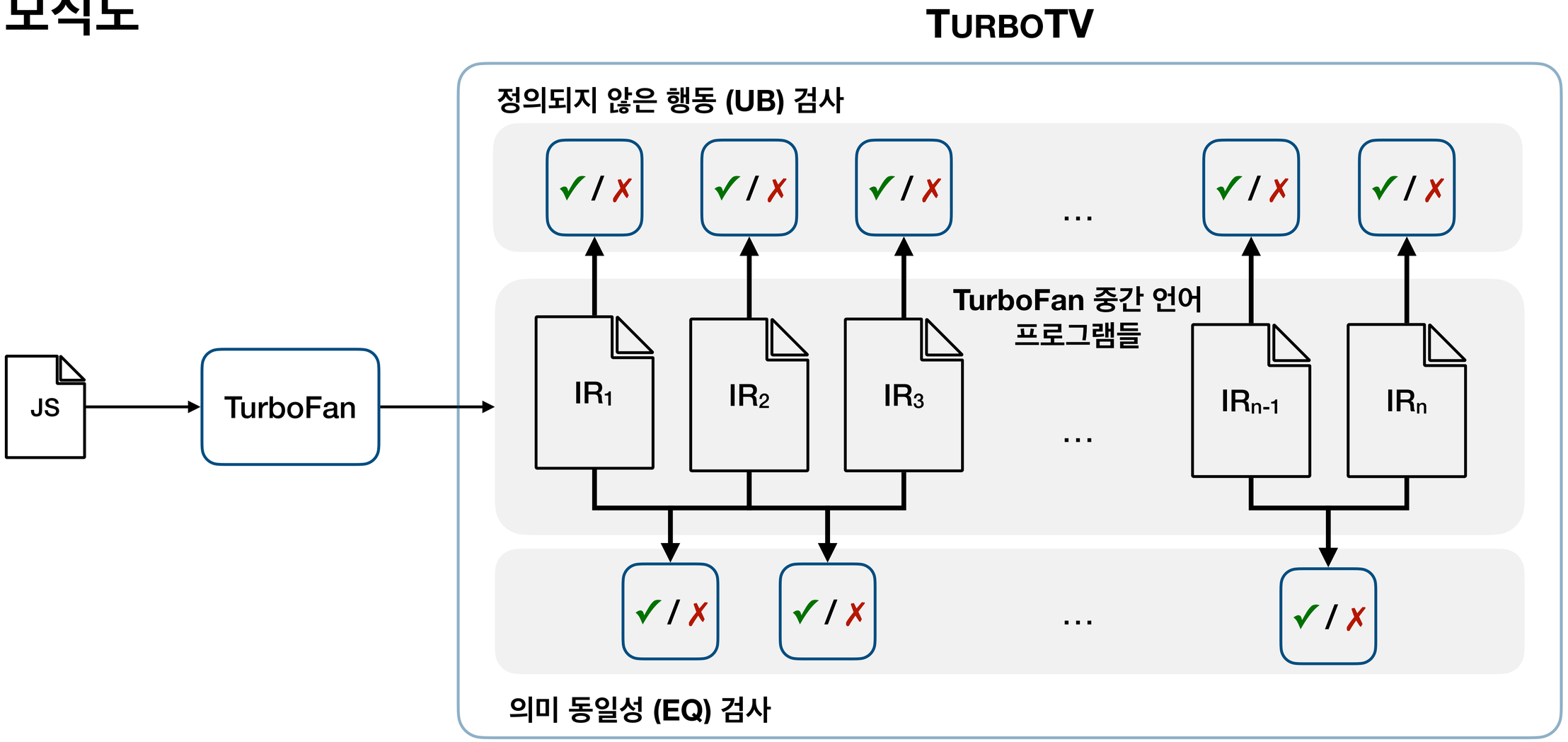
- TurboFan의 중간 언어의 의미를 **엄밀**하게 정의
  - 최근 보고된 TurboFan 최적화 버그 **9**개를 탐지 성공, 오탐 개수 **0**개
- 단계적 번역 검산을 통한 검산 성능 **최적화**
  - 196K개 JS파일 최적화 대상, 번역 검산의 **90%**가 **1**초 내에 완료, **1%**의 오탐 발생
  - **퍼징**과 결합하여 사용될 수 있을 정도로 성능이 우수

TurboTV

TurboFan의 올바른 최적화를 위한 번역 검산 시스템

- TurboFan의 중간 언어의 의미를 **엄밀**하게 정의
  - 최근 보고된 TurboFan 최적화 버그 **9**개를 탐지 성공, 오탐 개수 **0**개
- 단계적 번역 검산을 통한 검산 성능 **최적화**
  - 196K개 JS파일 최적화 대상, 번역 검산의 **90%**가 **1**초 내에 완료, **1%**의 오탐 발생
  - **퍼징**과 결합하여 사용될 수 있을 정도로 성능이 우수
- TurboTV의 활용: TurboFan + LLVM 대상의 **교차 언어 번역 검산**
  - LLVM의 Wasm 백엔드에서 1개의 버그 발견 및 패치

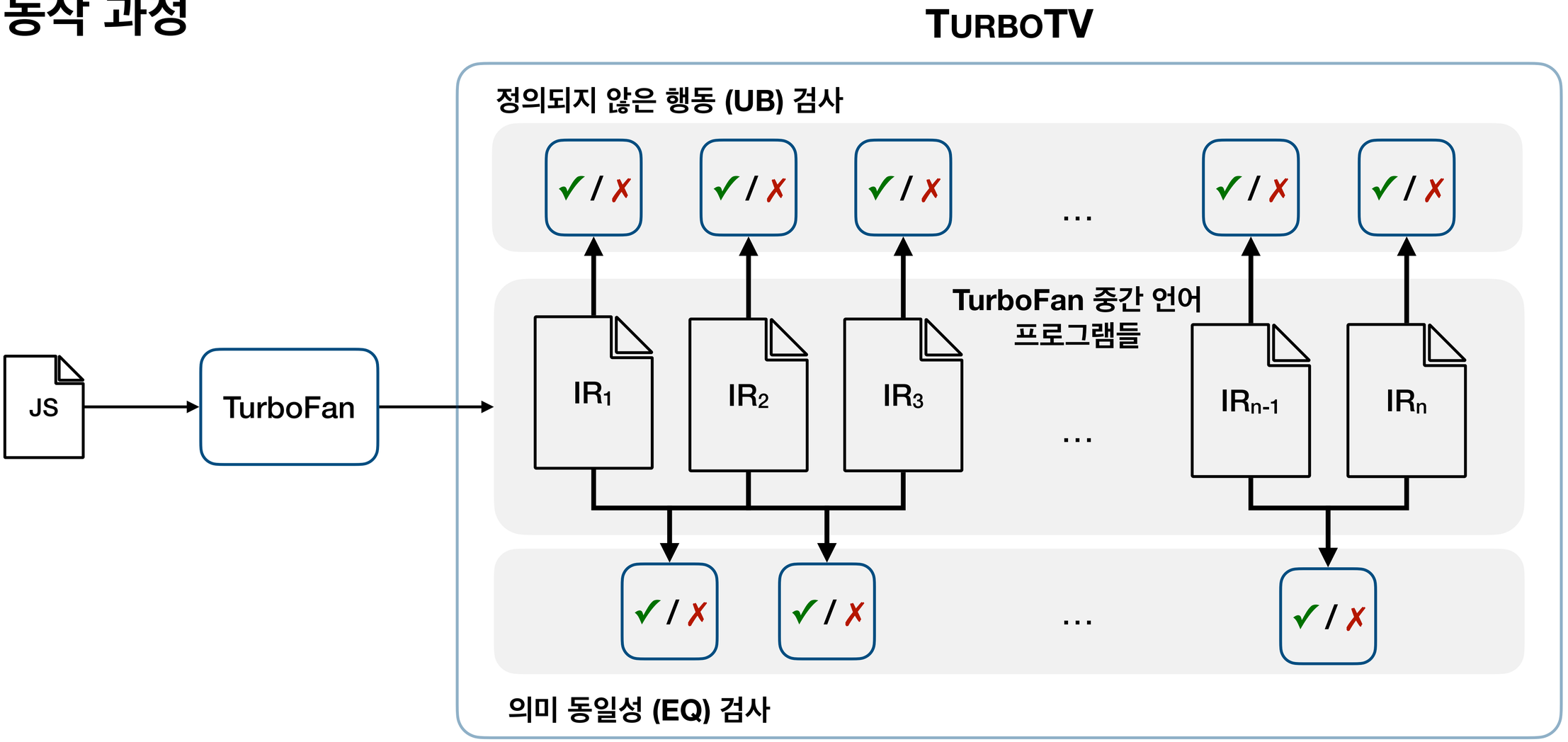
TurboTV
모식도


TurboTV
정의되지 않은 행동 (UB) 검사
✓ / ✗
✓ / ✗
✓ / ✗
...
✓ / ✗
✓ / ✗
TurboFan 중간 언어
프로그램들
JS
TurboFan
IR1
IR2
IR3
...
IRn-1
IRn
✓ / ✗
✓ / ✗
...
✓ / ✗
의미 동일성 (EQ) 검사

TurboTV 동작

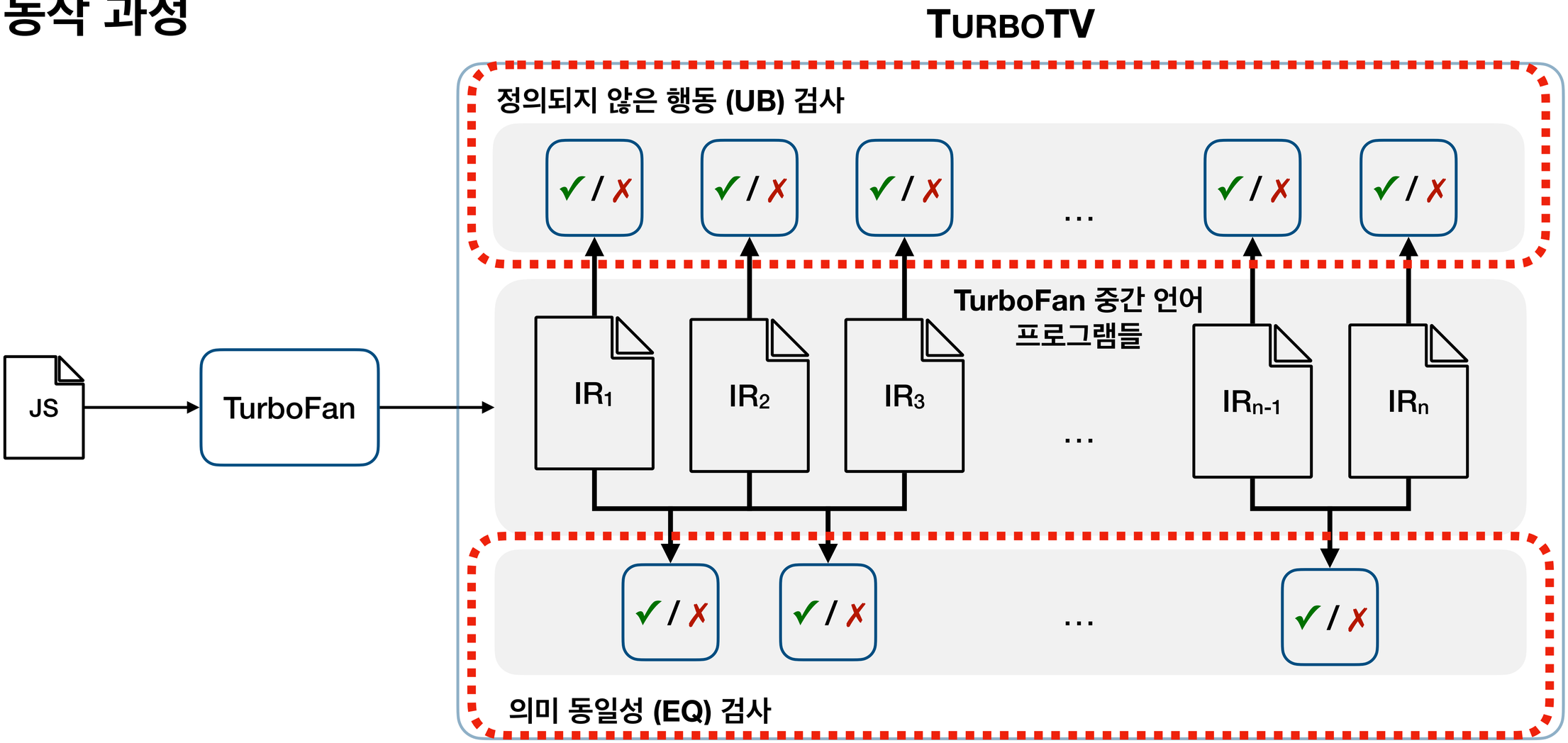
TurboTV

동작 과정



TurboTV

동작 과정

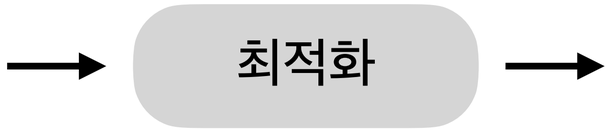


전통적 번역 검산

정의되지 않은 행동

- **정의되지 않은 행동**(Undefined Behavior, UB)은 **복잡성**을 증가시키는 주요 원인
  - 두 프로그램의 의미간의 **부분집합(포함)** 관계를 계산해야 한다.

```
// out-of-bound.c
int foo(int i) {
  int arr[1] = {0,};
  return arr[i];
}
```




```
// optimized.c
int foo(int i) {
  return 0;
}
```

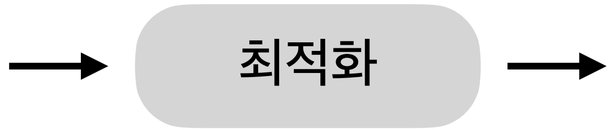
A. Pnueli et al. 1998. Translation Validation. In TACAS

전통적 번역 검산

정의되지 않은 행동

- **정의되지 않은 행동**(Undefined Behavior, UB)은 **복잡성**을 증가시키는 주요 원인
  - 두 프로그램의 의미간의 **부분집합(포함)** 관계를 계산해야 한다.

```
// out-of-bound.c
int foo(int i) {
  int arr[1] = {0,};
  return arr[i];
}
```




```
// optimized.c
int foo(int i) {
  return 0;
}
```

i가 **0**일 때: $0 = 0$

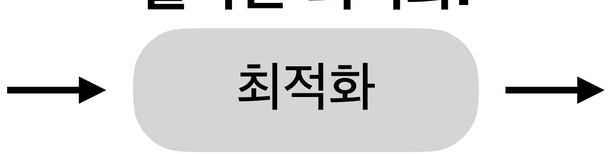
A. Pnueli et al. 1998. Translation Validation. In TACAS

전통적 번역 검산

정의되지 않은 행동

- **정의되지 않은 행동**(Undefined Behavior, UB)은 **복잡성**을 증가시키는 주요 원인
  - 두 프로그램의 의미간의 **부분집합(포함)** 관계를 계산해야 한다.

```
// out-of-bound.c
int foo(int i) {
  int arr[1] = {0,};
  return arr[i];
}
```




```
// optimized.c
int foo(int i) {
  return 0;
}
```

i가 **0**일 때: $0 = 0$

i가 **0**이 아닐 때: $UB \supseteq \{0\}$

A. Pnueli et al. 1998. Translation Validation. In TACAS

TurboTV의 단계적 검산

정의되지 않은 행동 검사와 의미 동일성 검사를 분리하여 검산을 최적화

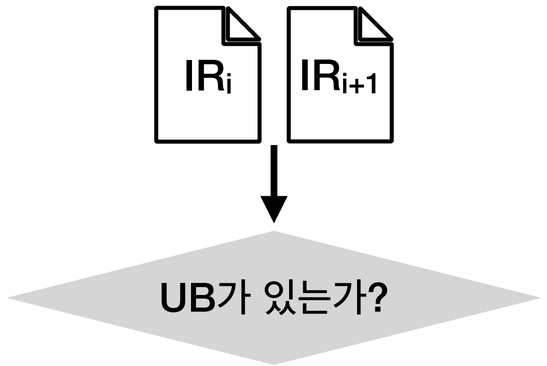
자바스크립트 표준에는 **UB**가 존재하지 않기에 **효율적인 번역 검산**이 가능

https://ecma-international.org/publications-and-standards/standards/ecma-262/

TurboTV의 단계적 검산 - UB 검사

정의되지 않은 행동 검사와 의미 동일성 검사를 분리하여 검산을 최적화

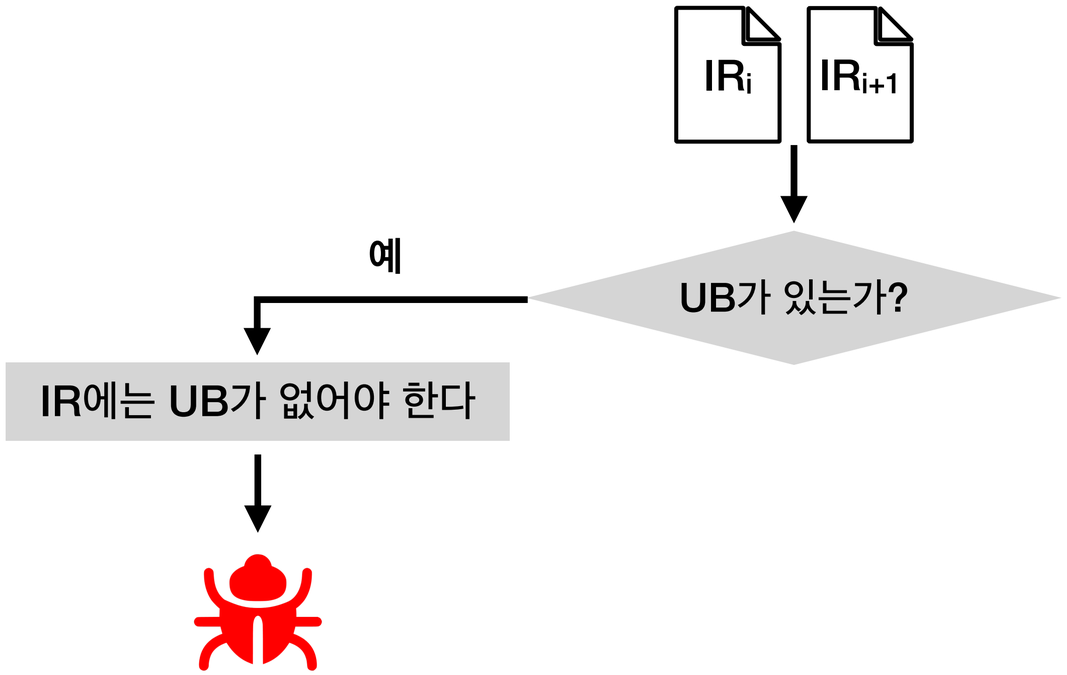
1. 자바스크립트에 UB가 없으므로, IR에도 **UB**가 없어야 한다.



TurboTV의 단계적 검산 - UB 검사

정의되지 않은 행동 검사와 의미 동일성 검사를 분리하여 검산을 최적화

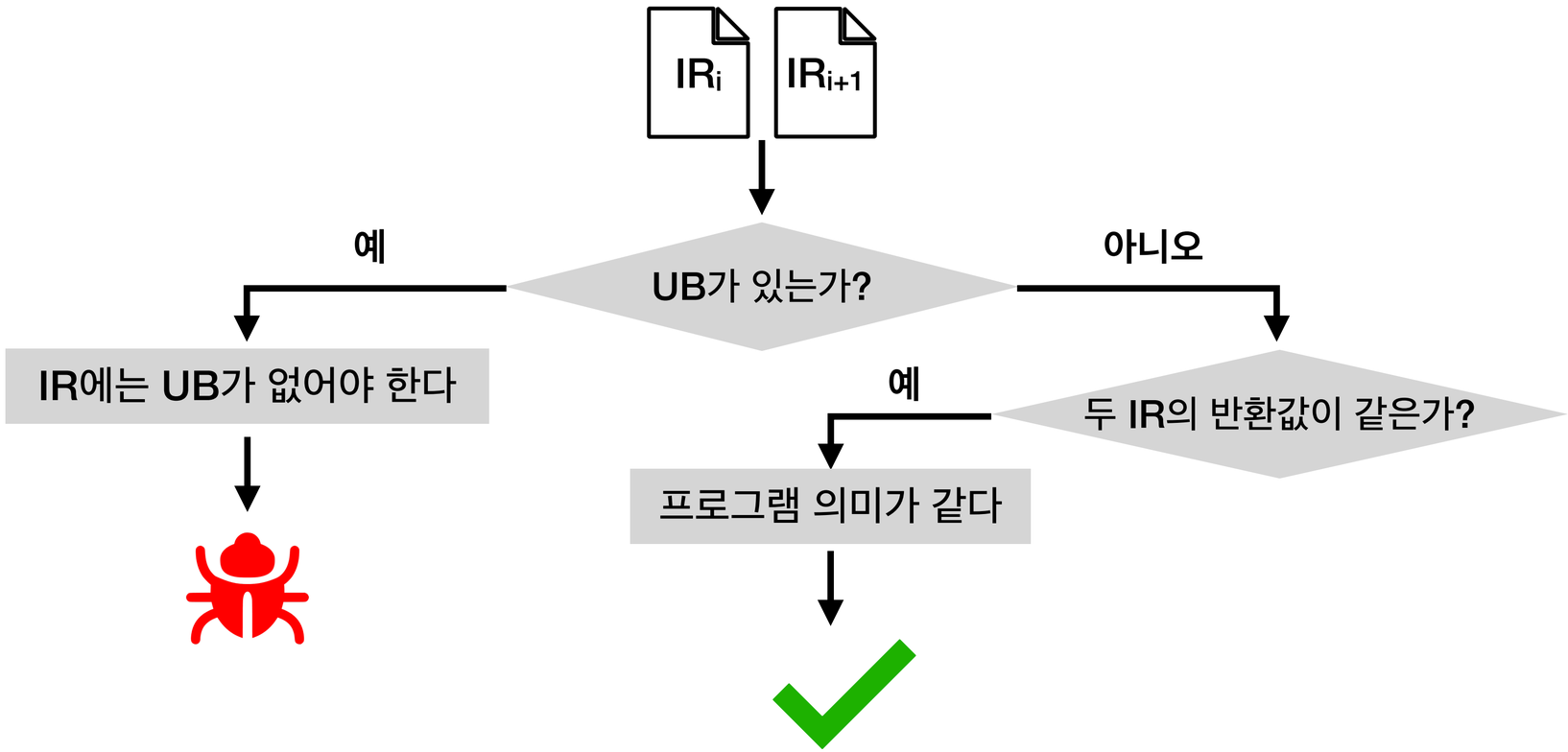
1. 자바스크립트에 UB가 없으므로, IR에도 **UB**가 없어야 한다.



TurboTV의 단계적 검산 - EQ 검사

정의되지 않은 행동 검사와 의미 동일성 검사를 분리하여 검산을 최적화

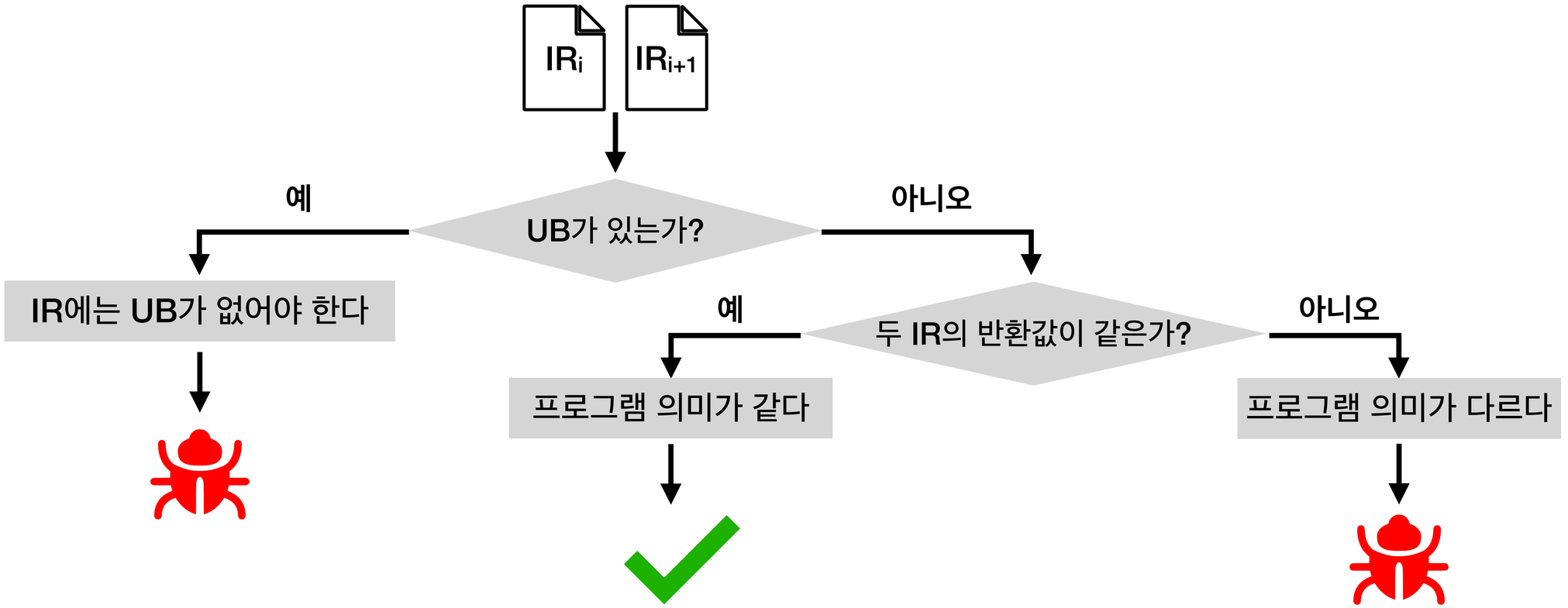
2. 최적화 후에도 프로그램의 의미는 보존되어야 한다.



TurboTV의 단계적 검산 - EQ 검사

정의되지 않은 행동 검사와 의미 동일성 검사를 분리하여 검산을 최적화

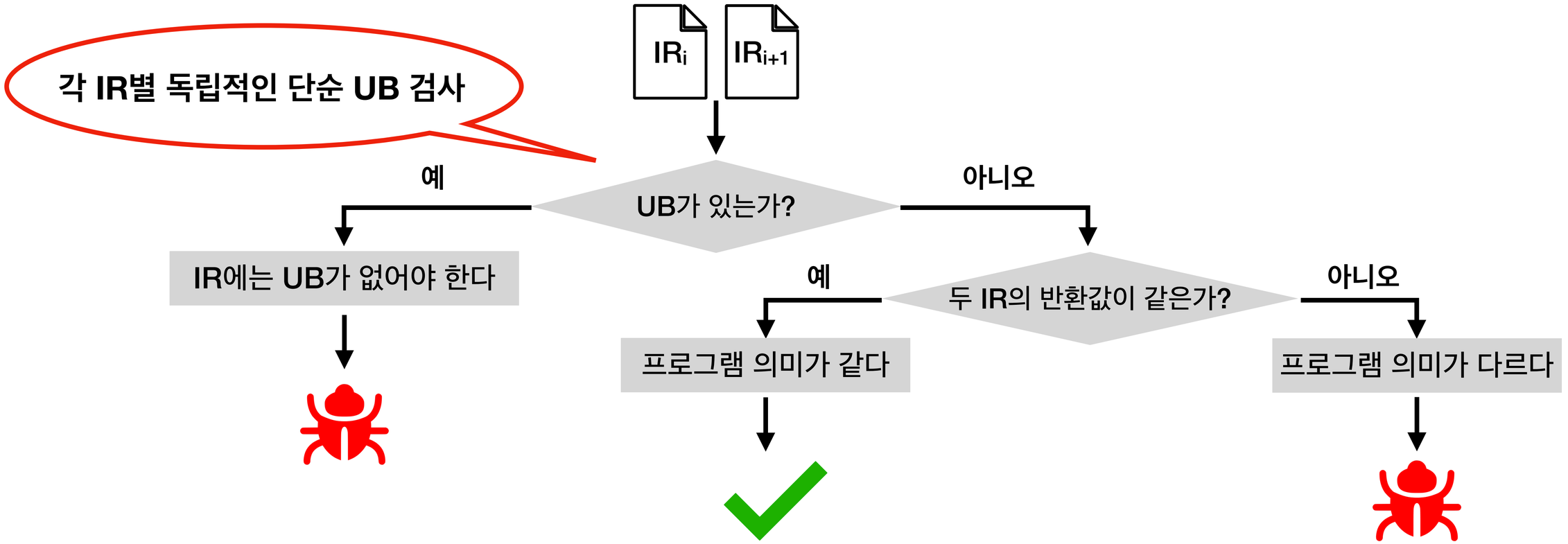
2. 최적화 후에도 프로그램의 의미는 보존되어야 한다.



TurboTV의 단계적 검산 - EQ 검사

정의되지 않은 행동 검사와 의미 동일성 검사를 분리하여 검산을 최적화

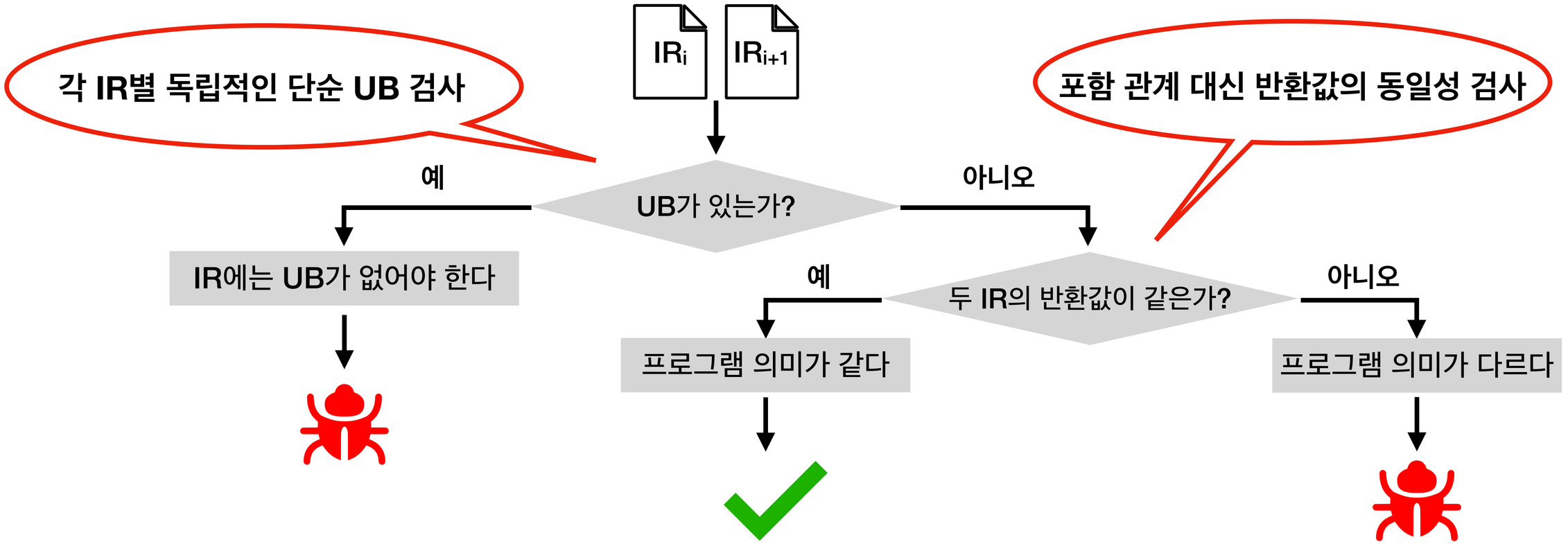
2. 최적화 후에도 프로그램의 의미는 보존되어야 한다.



TurboTV의 단계적 검산 - EQ 검사

정의되지 않은 행동 검사와 의미 동일성 검사를 분리하여 검산을 최적화

2. 최적화 후에도 프로그램의 의미는 보존되어야 한다.



TurboTV 평가

실험 설정
평가

- TurboTV는 정확한가?
  - 최근 보고된 TurboFan 최적화 버그 **9**개 대상으로 번역 검산 수행

실험 설정
평가

- TurboTV는 정확한가?
  - 최근 보고된 TurboFan 최적화 버그 **9**개 대상으로 번역 검산 수행
- TurboTV의 성능은 우수한가?
  - 576개의 TurboFan 유닛 테스트 대상으로 번역 검산 성능 측정
  - 다양한 최적화를 유발하는 196K개 JS파일 대상으로 번역 검산 성능 측정
  - 퍼징의 오류 **판독기**로 활용하였을 때의 부하 측정 (퍼징: 7일)

실험 설정
평가

- TurboTV는 정확한가?
  - 최근 보고된 TurboFan 최적화 버그 **9**개 대상으로 번역 검산 수행
- TurboTV의 성능은 우수한가?
  - 576개의 TurboFan 유닛 테스트 대상으로 번역 검산 성능 측정
  - 다양한 최적화를 유발하는 196K개 JS파일 대상으로 번역 검산 성능 측정
  - 퍼징의 오류 **판독기**로 활용하였을 때의 부하 측정 (퍼징: 7일)
- TurboTV를 또 어떻게 활용할 수 있는가?
  - **교차 언어 번역 검산** 아이디어 제시, LLVM의 유닛 테스트 대상으로 번역 검산 수행

TurboTV 정확성

Bug ID	TurboTV			탐지 결과
	IR	오탐	시간 초과	
1126249	20	0	0	✓
1195650	13	0	8	✓
1404607	33	0	0	✓
1198705	32	0	3	✓
1199345	13	0	0	✓
1200490	30	0	0	✓
1234764	19	0	5	✓
1234770	12	0	5	✓
1323114	11	0	0	✓

IR: TurboFan 최적화 과정에서 발생한 모든 중간 언어 프로그램의 개수
시간 제한: 5분

TurboTV 정확성

Bug ID	TurboTV			탐지 결과
	IR	오탐	시간 초과	
1126249	20	0	0	✓
1195650	13	0	8	✓
1404607	33	0	0	✓
1198705	32	0	3	✓
1199345	13	0	0	✓
1200490	30	0	0	✓
1234764	19	0	5	✓
1234770	12	0	5	✓
1323114	11	0	0	✓

IR: TurboFan 최적화 과정에서 발생한 모든 중간 언어 프로그램의 개수
시간 제한: 5분

TurboTV 정확성

Bug ID	TurboTV			탐지 결과	
	IR	오탐	시간 초과		
1126249	20	0	0	✓	UB Checker
1195650	13	0	8	✓	
1404607	33	0	0	✓	
1198705	32	0	3	✓	EQ Checker
1199345	13	0	0	✓	
1200490	30	0	0	✓	
1234764	19	0	5	✓	
1234770	12	0	5	✓	
1323114	11	0	0	✓	

IR: TurboFan 최적화 과정에서 발생한 모든 중간 언어 프로그램의 개수
시간 제한: 5분

TurboTV 성능

벤치마크	JS	TurboTV				평균 검산 시간
		IR_{All}	IR_{TV}	오탐	시간 초과	
UnitJS	580	16,011	4,387	41	328	2.61s
Corpus	196K	160,324	13,870	0	298	0.71s

IR_{All}: TurboFan 최적화 과정에서 발생한 모든 중간 언어 프로그램의 개수
IR_{TV}: TurboTV가 지원하는 중간 언어 프로그램 개수
시간제한: 5분

TurboTV 성능

벤치마크	JS	TurboTV				평균 검산 시간
		IR_{All}	IR_{TV}	오탐	시간 초과	
UnitJS	580	16,011	4,387	41	328	2.61s
Corpus	196K	160,324	13,870	0	298	0.71s

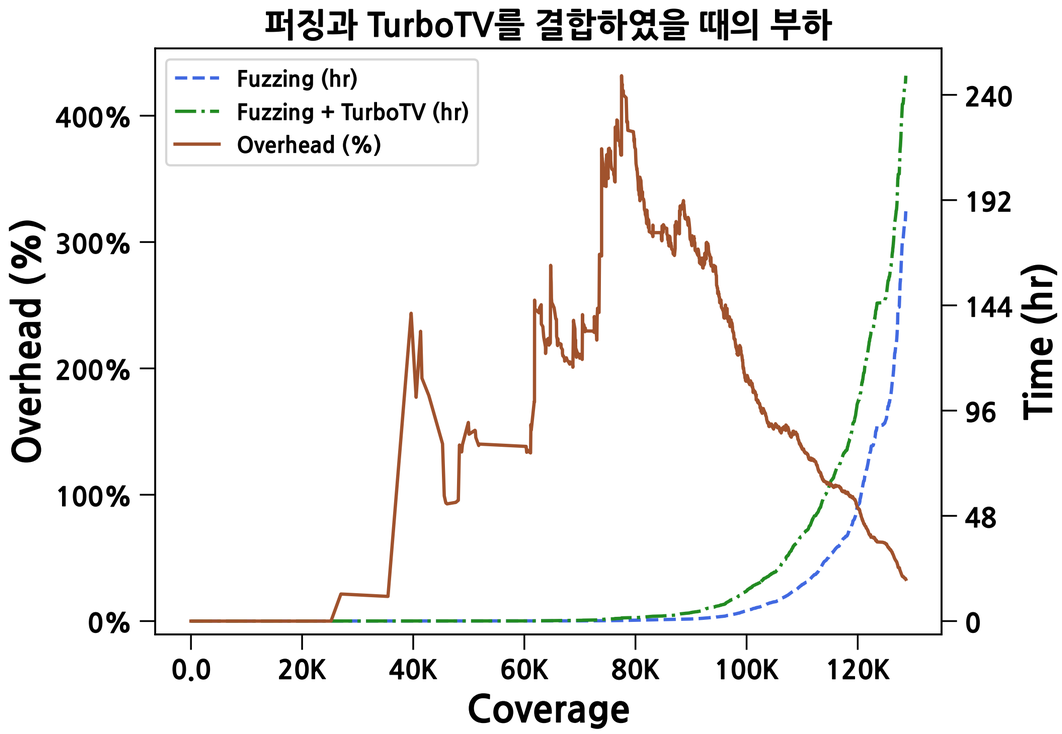
IR_{All}: TurboFan 최적화 과정에서 발생한 모든 중간 언어 프로그램의 개수
IR_{TV}: TurboTV가 지원하는 중간 언어 프로그램 개수
시간제한: 5분

TurboTV 성능

TurboTV의 활용: 퍼징 오류 판독기

총 7일 간 퍼징, 퍼징의 오류 판독기로 TurboTV를 사용하였을 때 **부하(Overhead)** 측정

Overhead: 같은 커버리지를 달성할 때, 퍼징과 TurboTV의 결합이 얼마나 시간이 소요되는가

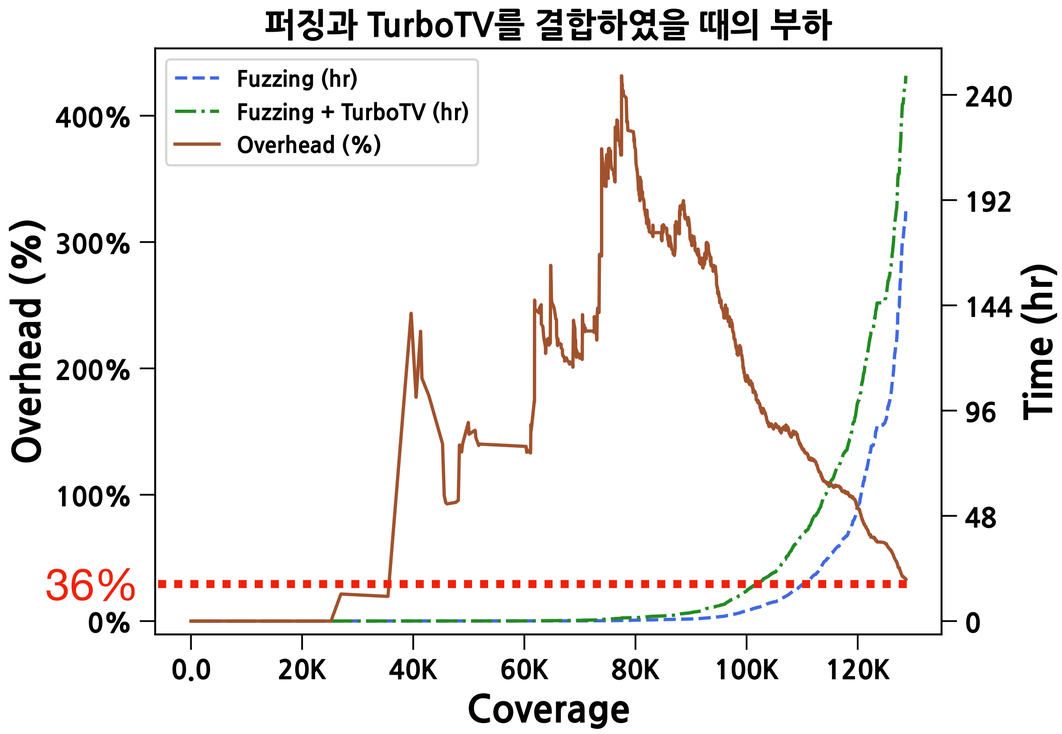


TurboTV 성능

TurboTV의 활용: 퍼징 오류 판독기

총 7일 간 퍼징, 퍼징의 오류 판독기로 TurboTV를 사용하였을 때 **부하(Overhead)** 측정

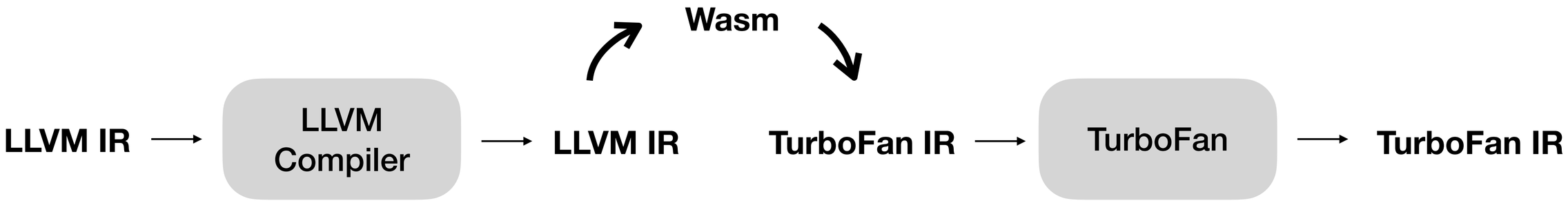
Overhead: 같은 커버리지를 달성할 때, 퍼징과 TurboTV의 결합이 얼마나 시간이 소요되는가



교차 언어 번역 검산

TurboTV의 활용

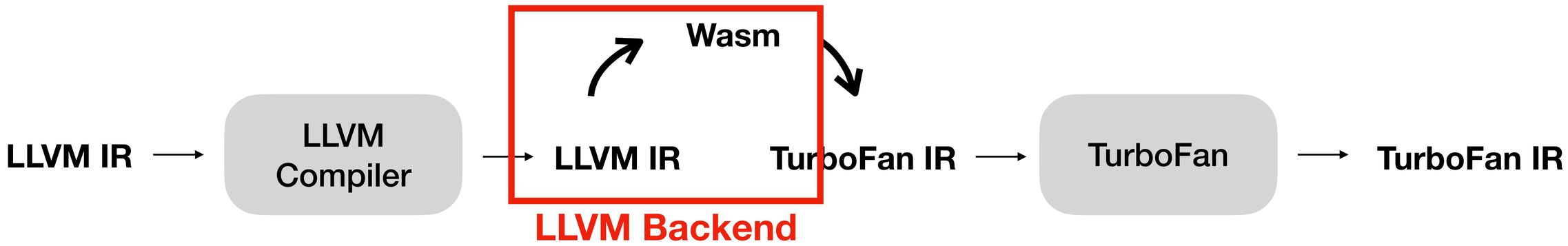
두 번역 검산기를 활용하여 두 컴파일러에 걸친 과정을 번역 검산



교차 언어 번역 검산

TurboTV의 활용

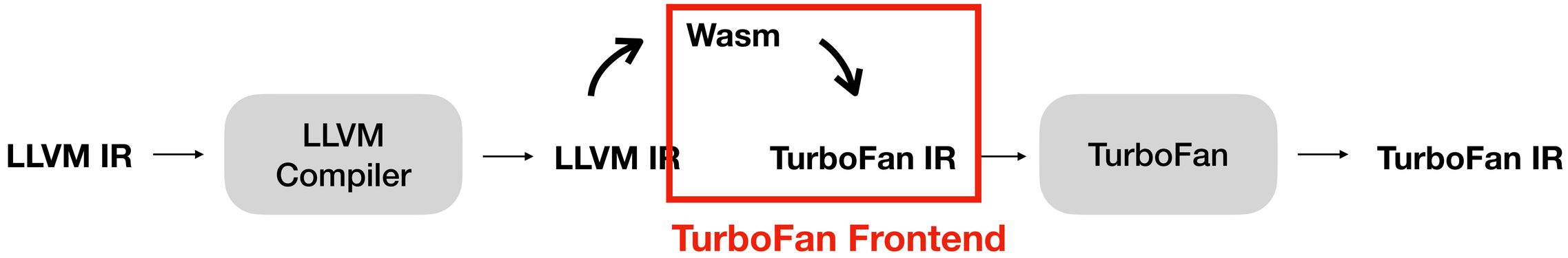
두 번역 검산기를 활용하여 두 컴파일러에 걸친 과정을 번역 검산



교차 언어 번역 검산

TurboTV의 활용

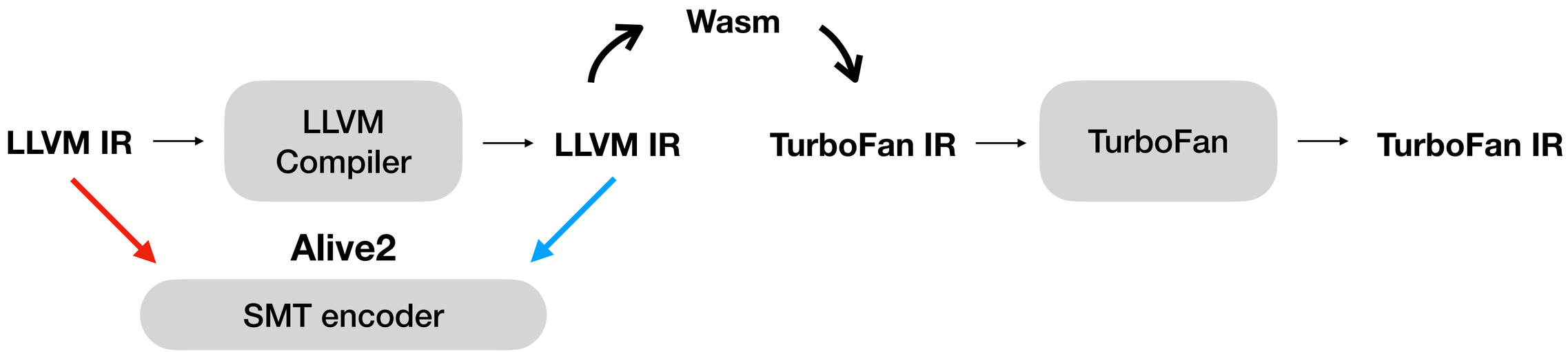
두 번역 검산기를 활용하여 두 컴파일러에 걸친 과정을 번역 검산



교차 언어 번역 검산

TurboTV의 활용

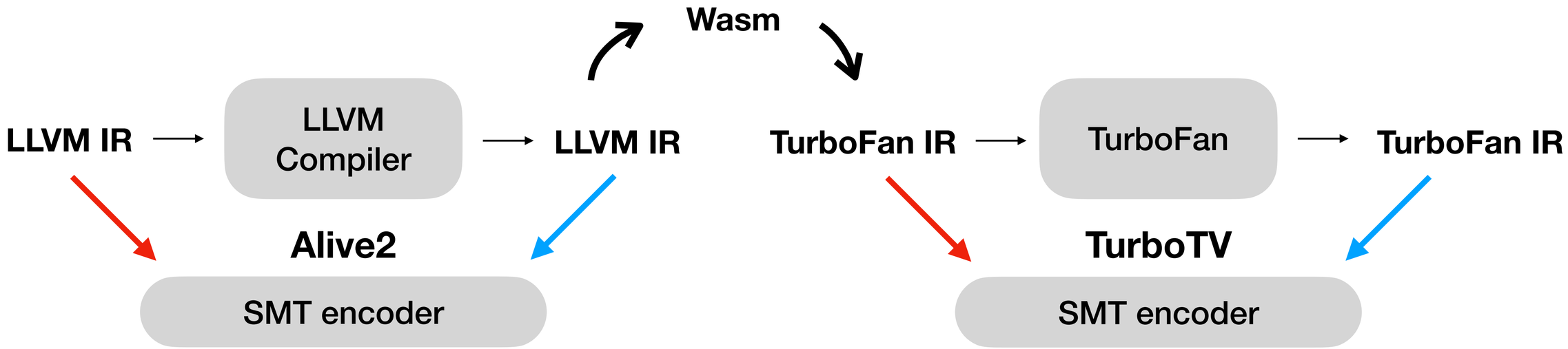
두 번역 검산기를 활용하여 두 컴파일러에 걸친 과정을 번역 검산



교차 언어 번역 검산

TurboTV의 활용

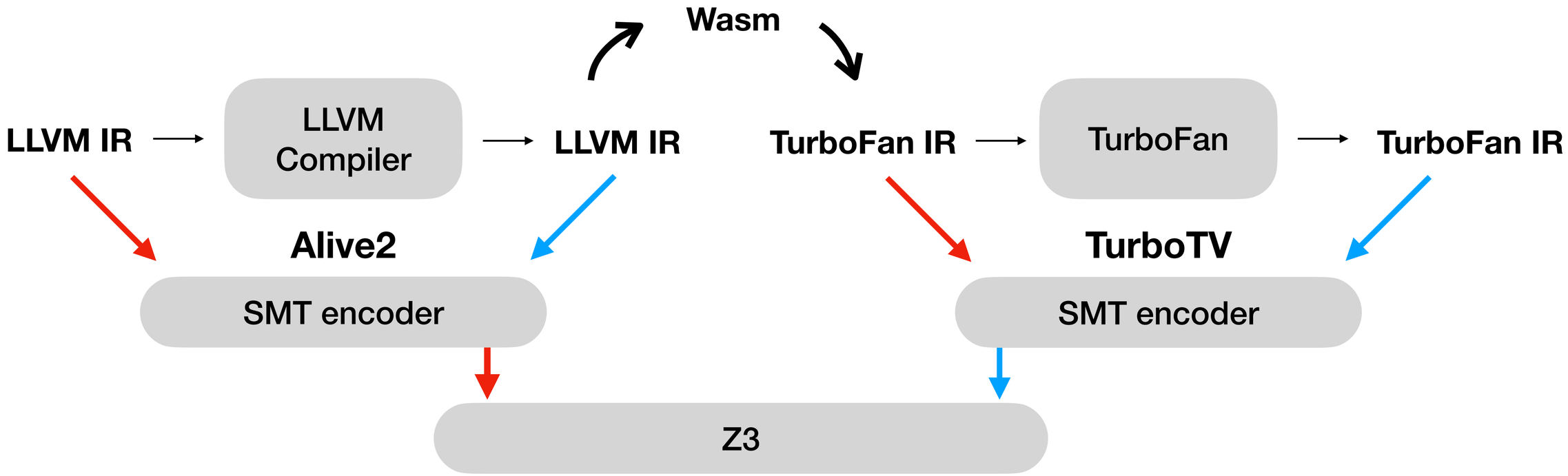
두 번역 검산기를 활용하여 두 컴파일러에 걸친 과정을 번역 검산



교차 언어 번역 검산

TurboTV의 활용

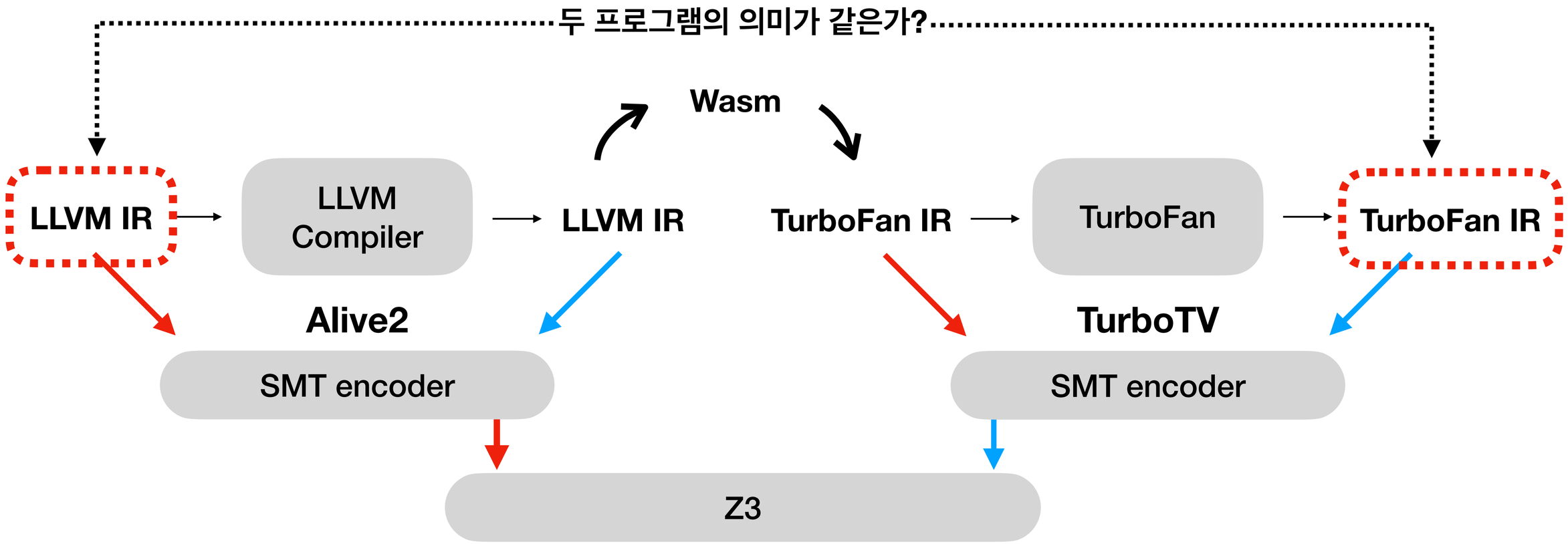
두 번역 검산기를 활용하여 두 컴파일러에 걸친 과정을 번역 검산



교차 언어 번역 검산

TurboTV의 활용

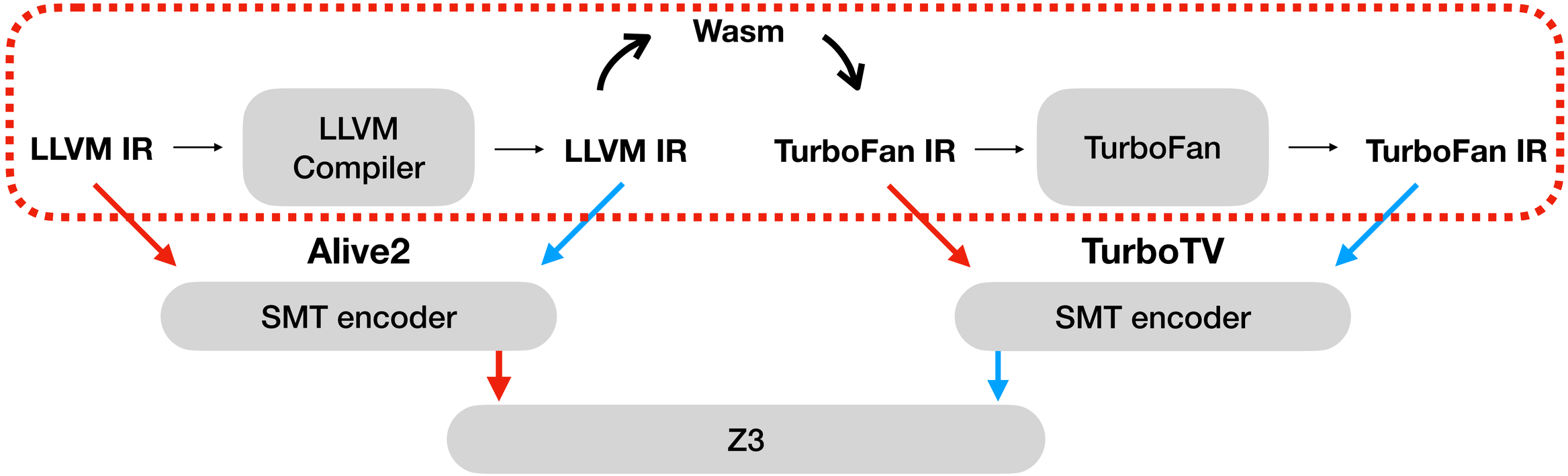
두 번역 검산기를 활용하여 두 컴파일러에 걸친 과정을 번역 검산



교차 언어 번역 검산

TurboTV의 활용

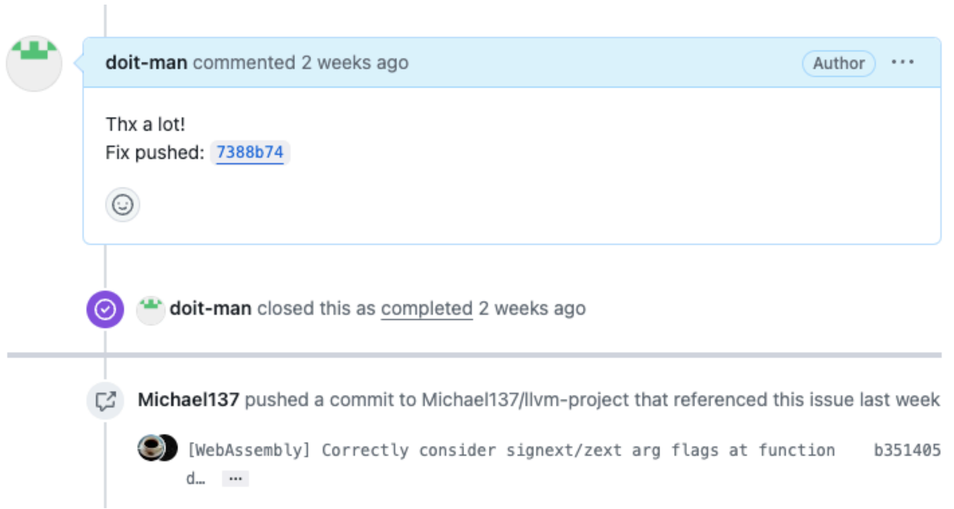
두 번역 검산기를 활용하여 두 컴파일러에 걸친 과정을 번역 검산



교차 언어 검산

발견된 버그

LLVM의 Wasm 백엔드에서 버그 발견, 제보 및 패치



https://github.com/llvm/llvm-project/issues/63388

마무리

홈페이지 QR 코드

- **TurboTV:** TurboFan의 **올바른 최적화**를 위한 **번역 검산** 시스템
- TurboFan의 복잡한 내부 구현과 **무관**하게 특정 최적화가 올바른지 검사

마무리

홈페이지 QR 코드

- **TurboTV:** TurboFan의 **올바른 최적화**를 위한 **번역 검산** 시스템
- TurboFan의 복잡한 내부 구현과 **무관**하게 특정 최적화가 올바른지 검사
- 단계적 번역 검산을 통한 **성능** 최적화
  - 퍼징의 오류 판독기로 활용할 수 있을 정도로 우수한 성능

마무리

홈페이지 QR 코드

- **TurboTV:** TurboFan의 **올바른 최적화**를 위한 **번역 검산** 시스템
- TurboFan의 복잡한 내부 구현과 **무관**하게 특정 최적화가 올바른지 검사
- 단계적 번역 검산을 통한 **성능** 최적화
  - 퍼징의 오류 판독기로 활용할 수 있을 정도로 우수한 성능
- **교차 언어 번역 검산:** 다른 번역 검산기를 활용, LLVM에서 TurboFan으로의 과정을 번역 검산
  - LLVM Wasm 백엔드에서 **잘못된 컴파일 버그** 발견

마무리

홈페이지 QR 코드

- **TurboTV:** TurboFan의 **올바른 최적화**를 위한 **번역 검산** 시스템
- TurboFan의 복잡한 내부 구현과 **무관**하게 특정 최적화가 올바른지 검사
- 단계적 번역 검산을 통한 **성능** 최적화
  - 퍼징의 오류 판독기로 활용할 수 있을 정도로 우수한 성능
- **교차 언어 번역 검산:** 다른 번역 검산기를 활용, LLVM에서 TurboFan으로의 과정을 번역 검산
  - LLVM Wasm 백엔드에서 **잘못된 컴파일 버그** 발견
- **홈페이지**: https://prosys.kaist.ac.kr/turbo-tv/

부록

TurboTV가 검사하는 UB

선정 기준

- **TurboFan 중간 언어에는 표준이 없기에 다음과 같은 기준으로 UB 선정**
  - TurboFan의 과거 **보안 버그**에서 일어난 대표적 케이스 수집
  - 해당 UB들이 **LLVM**에서도 UB로 분류되는 지 교차 검사
  - 우리가 구현한 UB Checker가 실제로 TurboFan의 잘못된 행동을 탐지하는지 실험

TurboTV가 검사하는 UB

- **Out-of-bound (정의 되지 않은) 메모리** 접근
- **Unrechable** 연산자에 도달
  - **Unrechable:** TurboFan의 분석 결과로 프로그램이 도달하면 안되는 지점에 삽입
- **잘못된 타입**을 가진 연산자가 존재

TurboTV의 검산 대상

현재 검산 대상과 발전 가능성

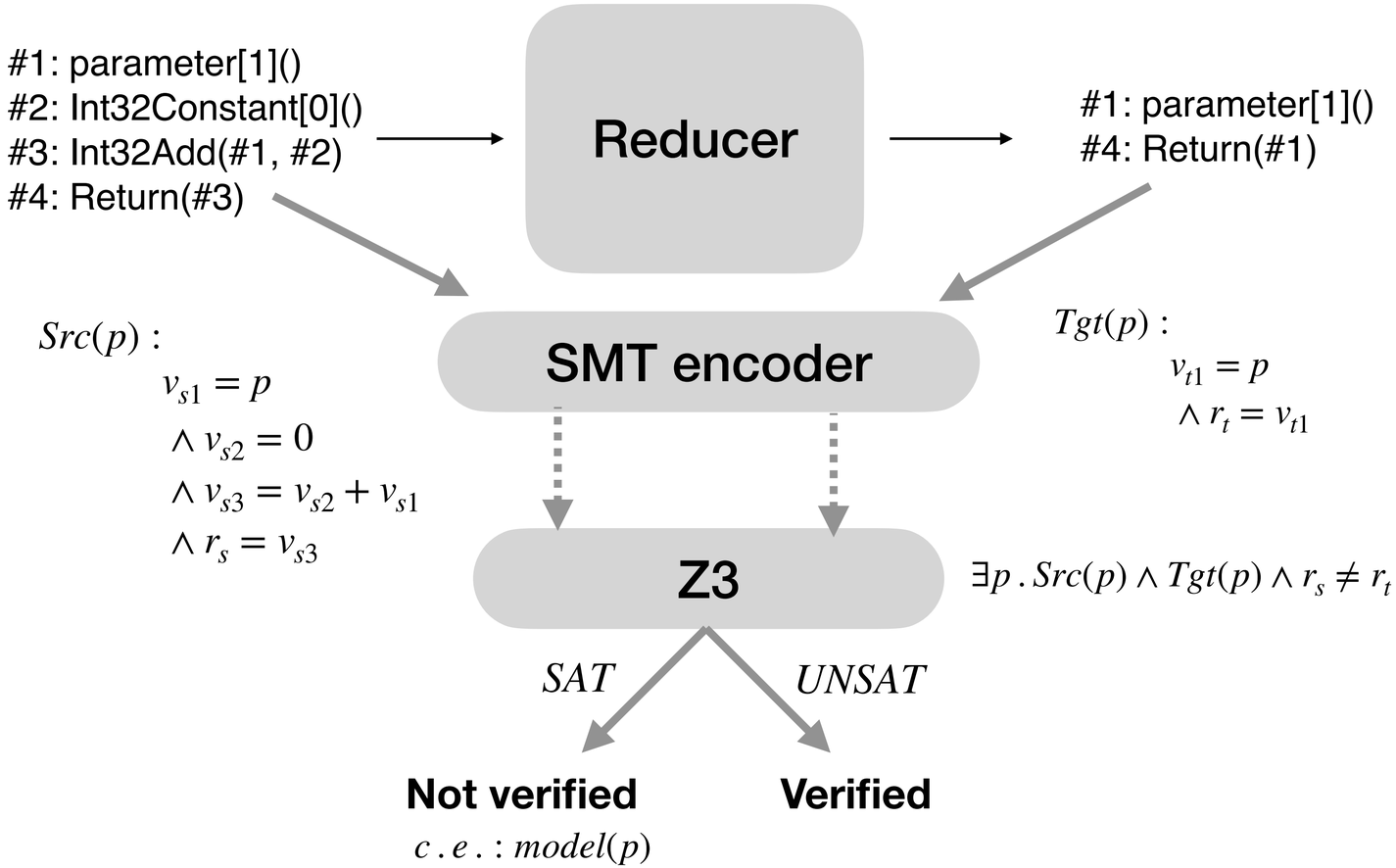
- **현재:** 순환문이 존재하지 않는 단일 함수 대상의 번역 검산

- **발전 가능성**
  - 함수 간의 관계 고려: 각 함수의 의미를 인코딩, 적절하게 함수를 호출 했는지 검사
  - 순환문이 존재하는 함수: Loop invariant synthesis, loop unrolling

번역 검산

예시

```
function foo(x) {
    return x+0
}
```

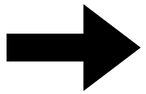


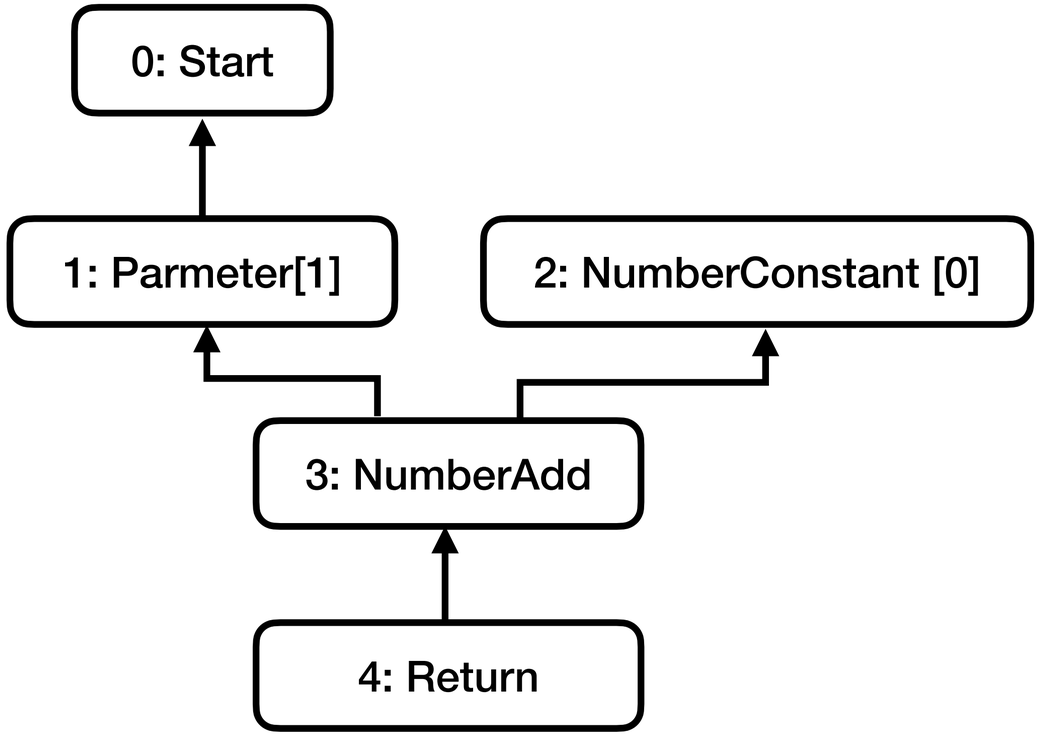
TurboFan 중간 언어

함수는 노드와 간선의 유향 그래프

$G_S = \langle Node, \rightarrow_S \rangle$

```
function foo(x) {
    return x+0
}
```

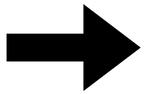


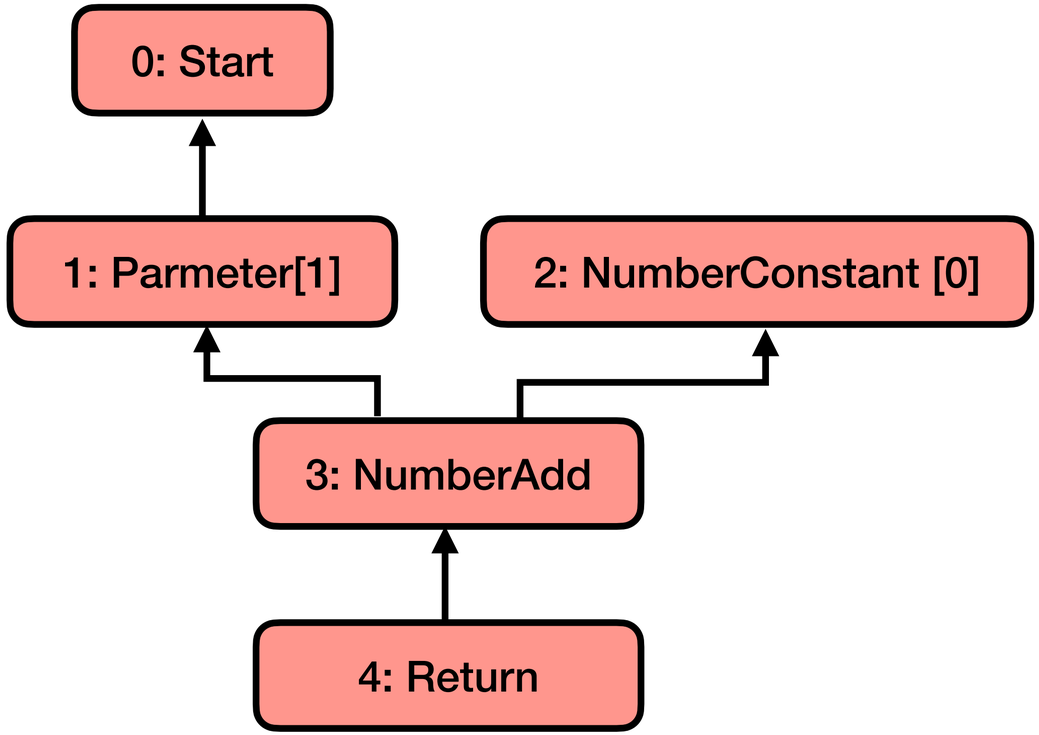


TurboFan 중간 언어

노드: 연산자, 값

```
function foo(x) {
    return x+0
}
```

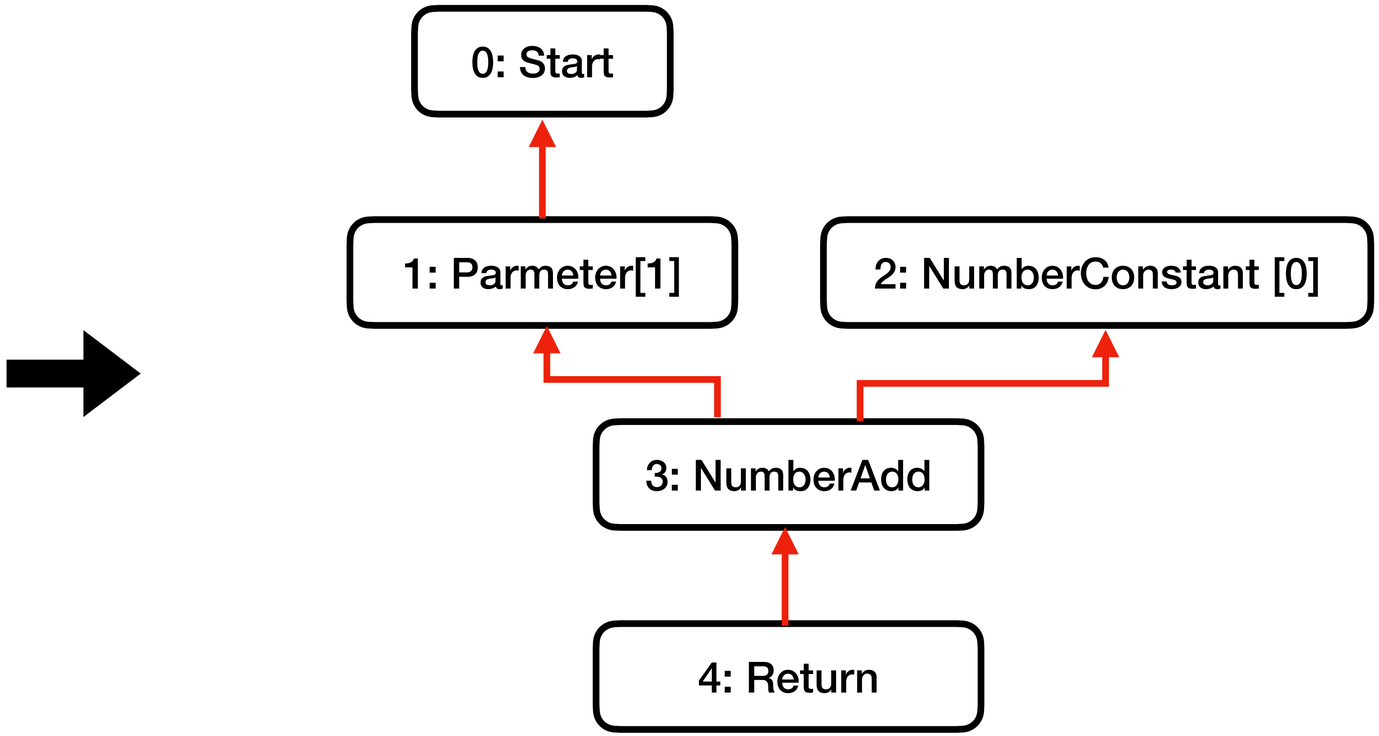




TurboFan 중간 언어

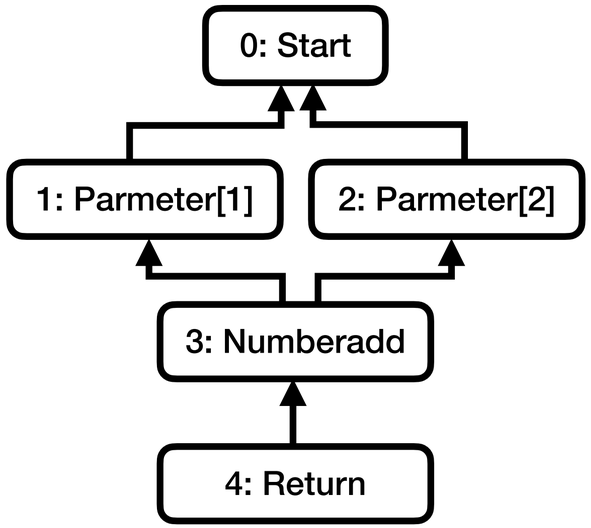
간선: 정보의 흐름

```
function foo(x) {
    return x+0
}
```



TurboTV - 예시

TurboFan의 최적화

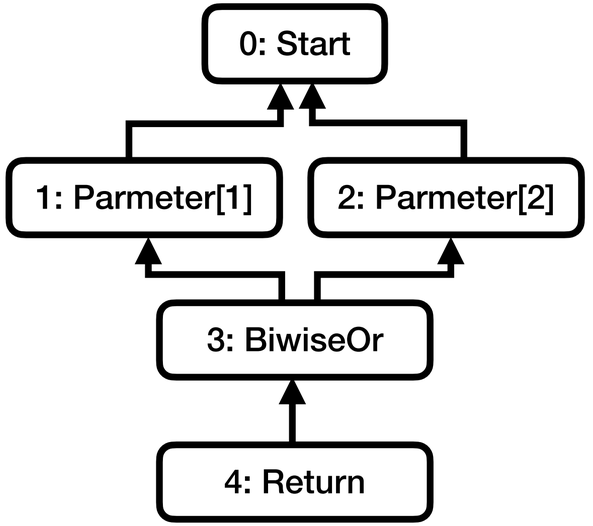


소스 TurboFan 중간 언어

Reducer

$$X + Y \rightarrow X \mid Y$$

잘못된 최적화



타깃 TurboFan 중간 언어

Reducer: TurboFan의 최적화기

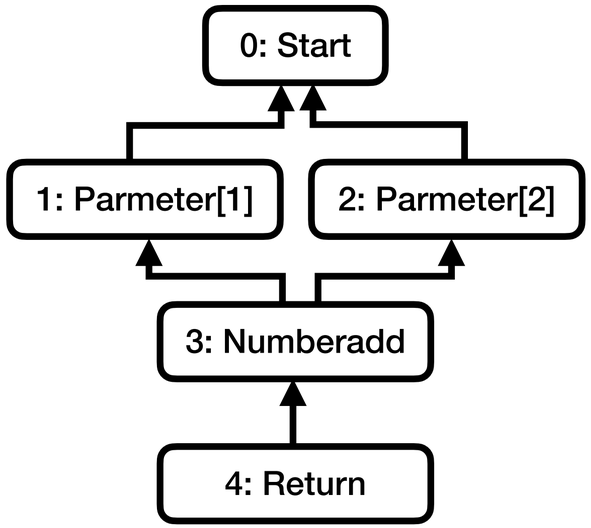
TurboTV - 예시

프로그램 인코딩

p_i : 프로그램 인자

v_{si} : 소스 프로그램 중간 변수

r_s : 소스 프로그램 반환 값



소스 TurboFan 중간 언어

SMT Encoder

$$Src(p_1, p_2):$$
$$v_{s1} = p_1$$
$$\wedge\, v_{s2} = p_2$$
$$\wedge\, v_{s3} = v_{s2} + v_{s1}$$
$$\wedge\, r_s = v_{s3}$$

인코딩된 소스 프로그램

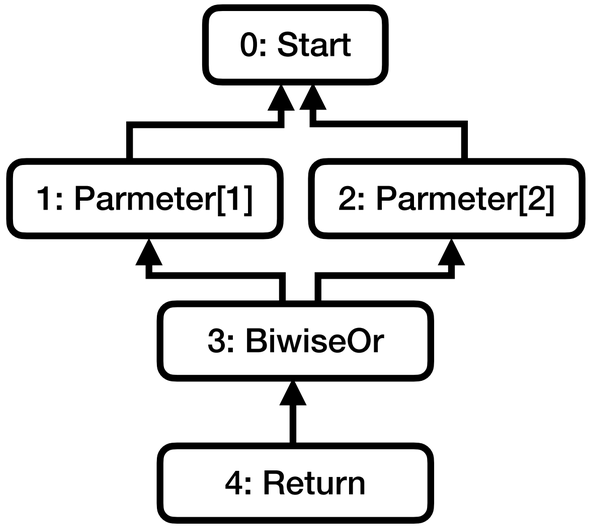
TurboTV - 예시

프로그램 인코딩

p_i : 프로그램 인자

v_{ti} : 타깃 프로그램 중간 변수

r_t : 타깃 프로그램 반환 값



타깃 TurboFan 중간 언어

SMT Encoder

$$
\begin{aligned}
&Tgt(p_1, p_2): \\
&\quad v_{t1} = p_1 \\
&\quad \wedge\, v_{t2} = p_2 \\
&\quad \wedge\, v_{t3} = v_{t1} \mid v_{t2} \\
&\quad \wedge\, r_t = v_{t3}
\end{aligned}
$$

인코딩된 타깃 프로그램

TurboTV - 예시
프로그램 동일성 검사

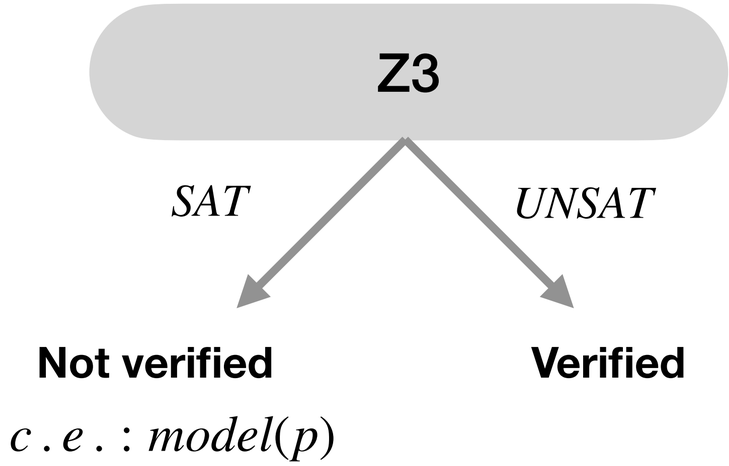
$$
\begin{aligned}
&Src(p_1, p_2): \\
&\quad v_{s1} = p_1 \\
&\quad \wedge\, v_{s2} = p_2 \\
&\quad \wedge\, v_{s3} = v_{s2} + v_{s1} \\
&\quad \wedge\, r_s = v_{s3}
\end{aligned}
$$

인코딩된 소스 프로그램

$$
\begin{aligned}
&Tgt(p_1, p_2): \\
&\quad v_{t1} = p_1 \\
&\quad \wedge\, v_{t2} = p_2 \\
&\quad \wedge\, v_{t3} = v_{t1} \mid v_{t2} \\
&\quad \wedge\, r_t = v_{t3}
\end{aligned}
$$

인코딩된 타깃 프로그램

$$\exists p_1 \exists p_2 \,.\, Src(p_1, p_2) \wedge Tgt(p_1, p_2) \wedge r_s \neq r_t$$



TurboTV - 예시

프로그램 동일성 검사

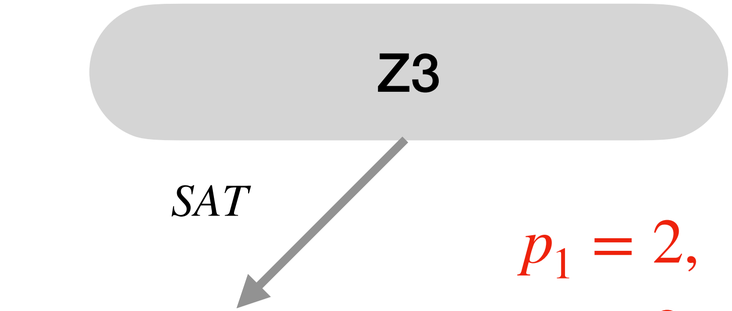
$$Src(p_1, p_2):$$
$$v_{s1} = p_1$$
$$\wedge\, v_{s2} = p_2$$
$$\wedge\, v_{s3} = v_{s2} + v_{s1}$$
$$\wedge\, r_s = v_{s3}$$

인코딩된 소스 프로그램

$$Tgt(p_1, p_2):$$
$$v_{t1} = p_1$$
$$\wedge\, v_{t2} = p_2$$
$$\wedge\, v_{t3} = v_{t1} \mid v_{t2}$$
$$\wedge\, r_t = v_{t3}$$

인코딩된 타깃 프로그램

$$\exists p_1 \exists p_2 \,.\, Src(p_1, p_2) \wedge Tgt(p_1, p_2) \wedge r_s \neq r_t$$



Not verified

$c\,.\,e\,.\,: model(p)$

$p_1 = 2,$
$p_2 = 2,$
$r_s = 4,$
$r_t = 2,$
$\ldots$

생성 기반 번역 검산

번역 검산기를 퍼징 판독기로 활용

- **퍼징**
  - 소프트웨어 입력값을 무작위 **생성**, 소프트웨어에 대입하는 테스팅 기법
  - 퍼징 판독기: 생성된 입력값을 대입 했을 때 발생하는 소프트웨어 에러나 충돌을 모니터링
- **퍼징 판독기로 번역 검산기 활용**
  - **컴파일러 퍼징 과정**에서 생성된 입력이 올바르게 번역 되는지 검산
  - 단순 에러, 충돌 보다 **더 넓은 범위**의 버그 탐지 가능

생성 기반 번역 검산

번역 검산기를 퍼징 판독기로 활용