

컴파일러 최적화를 자동으로 이식하는 방법

장봉준, 허기홍

컴파일러 최적화 (Compiler Optimization)

컴파일러 최적화 (Compiler Optimization)

- 컴파일러가 원본 프로그램을 더 빠른 프로그램으로 바꾸는 것

컴파일러 최적화 (Compiler Optimization)

- 컴파일러가 원본 프로그램을 더 빠른 프로그램으로 바꾸는 것



$$X \div 2 \longrightarrow X \gg 1$$



컴파일러 최적화 (Compiler Optimization)

- 컴파일러가 원본 프로그램을 더 빠른 프로그램으로 바꾸는 것
- 프로그램 성능 향상을 위한 필수적인 요소



$$X \div 2 \longrightarrow X \gg 1$$



컴파일러 최적화 (Compiler Optimization)

- 컴파일러가 원본 프로그램을 더 빠른 프로그램으로 바꾸는 것
- 프로그램 성능 향상을 위한 필수적인 요소
 - 널리 사용되는 모든 컴파일러에 구현되어 있음



$$X \div 2 \longrightarrow X \gg 1$$



컴파일러 최적화 (Compiler Optimization)

- 컴파일러가 원본 프로그램을 더 빠른 프로그램으로 바꾸는 것
- 프로그램 성능 향상을 위한 필수적인 요소
 - 널리 사용되는 모든 컴파일러에 구현되어 있음
- LLVM: 반복문 최적화 (LICM, 불변값 빼내기) 적용시 성능 14% 향상 [PLDI 2025]



$$X \div 2 \longrightarrow X \gg 1$$



컴파일러 최적화 (Compiler Optimization)

- 컴파일러가 원본 프로그램을 더 빠른 프로그램으로 바꾸는 것
- 프로그램 성능 향상을 위한 필수적인 요소
 - 널리 사용되는 모든 컴파일러에 구현되어 있음
 - LLVM: 반복문 최적화 (LICM, 불변값 빼내기) 적용시 성능 14% 향상 [PLDI 2025]
 - NVCC: 최적화 적용시 (-O0 vs -O3) 성능 약 4배 향상 [TACO 2024]



$$X \div 2 \longrightarrow X \gg 1$$



새로운 언어를 위한 새로운 컴파일러

새로운 언어를 위한 새로운 컴파일러

- 웹, 클라우드: WebAssembly (WASM)



새로운 언어를 위한 새로운 컴파일러

- 웹, 클라우드: WebAssembly (WASM)
- 인공지능: PyTorch Dynamo, CUDA Tile IR, Triton



컴파일러 최적화 구현의 까다로움

컴파일러 최적화 구현의 까다로움

- 최적화 기회 포착 및 규칙의 올바른 보장하기 위해 시간이 많이 소요

컴파일러 최적화 구현의 까다로움

- 최적화 기회 포착 및 규칙의 올바름 보장하기 위해 시간이 많이 소요
- 새로운 컴파일러에는 단순한 규칙도 빠져있는 경우 존재

$$\begin{aligned} (X - Y) + Y &\Rightarrow X \\ (X \oplus Y) \oplus Y &\Rightarrow X \end{aligned} \notin$$



컴파일러 최적화 구현의 까다로움

- 최적화 기회 포착 및 규칙의 올바름 보장하기 위해 시간이 많이 소요
- 새로운 컴파일러에는 단순한 규칙도 빠져있는 경우 존재
- WASM vs LLVM 컴파일러 성능 차이: 17.15% [ISSTA 2023]

$$\begin{aligned} (X - Y) + Y &\Rightarrow X \\ (X \oplus Y) \oplus Y &\Rightarrow X \end{aligned} \notin$$



관찰: 컴파일러 최적화 데자뷰

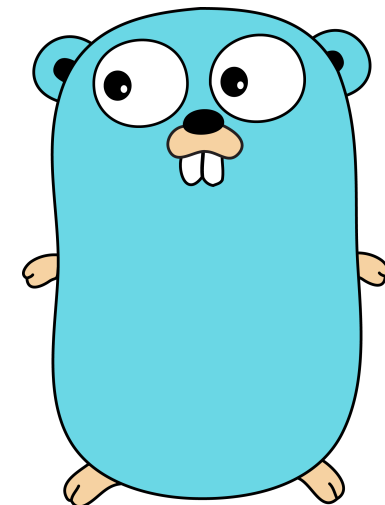
- 서로 다른 컴파일러지만 최적화 아이디어는 같음

$$\begin{aligned}(X - Y) + Y &\Rightarrow X \\ (X \oplus Y) \oplus Y &\Rightarrow X\end{aligned} \quad \notin$$



관찰: 컴파일러 최적화 데자뷰

- 서로 다른 컴파일러지만 최적화 아이디어는 같음
- 성숙한 컴파일러는 새로운 컴파일러에 비해 많은 최적화가 구현되어 있음



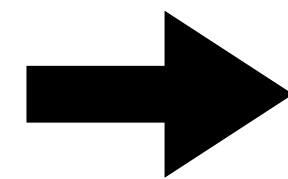
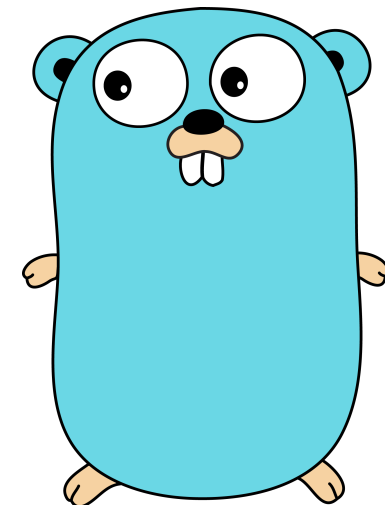
$\ni$

$$\begin{aligned} (X - Y) + Y &\Rightarrow X \\ (X \oplus Y) \oplus Y &\Rightarrow X \end{aligned} \notin$$

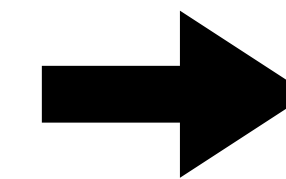


성숙한 컴파일러

해결책: 컴파일러 꼬마 최적화(Peephole Optimization) 자동 이식



$$\begin{aligned}(X - Y) + Y &\Rightarrow X \\ (X \oplus Y) \oplus Y &\Rightarrow X\end{aligned}$$



성숙한 컴파일러

최적화 자동 이식 시스템의 장점

나쁜 개발

더 느려지는 거 아님?

틀린 최적화 아님?

설명 중



- 최적화 손수 작성: 시간이 오래 걸림
- 역최적화 작성 가능
- 최적화가 틀릴 수 있음
- 논쟁에서 밀리면 머지안됨

좋은 개발

다른 컴파일러도 함

검증했음



비용모델 사용

- 이미 있는 최적화 사용: 금방 끝남
- 비용모델 사용: 성능 향상 보장
- 검증기 사용: 팩트임
- 논쟁에서 밀릴 확률 0에 수렴함

성과

성과



6,266

이식가능한 꼬마 최적화 규칙 생성

성과



6,266

이식가능한 꼬마 최적화 규칙 생성



6%

컴파일된 프로그램 성능 향상

성과



6,266

이식가능한 꼬마 최적화 규칙 생성

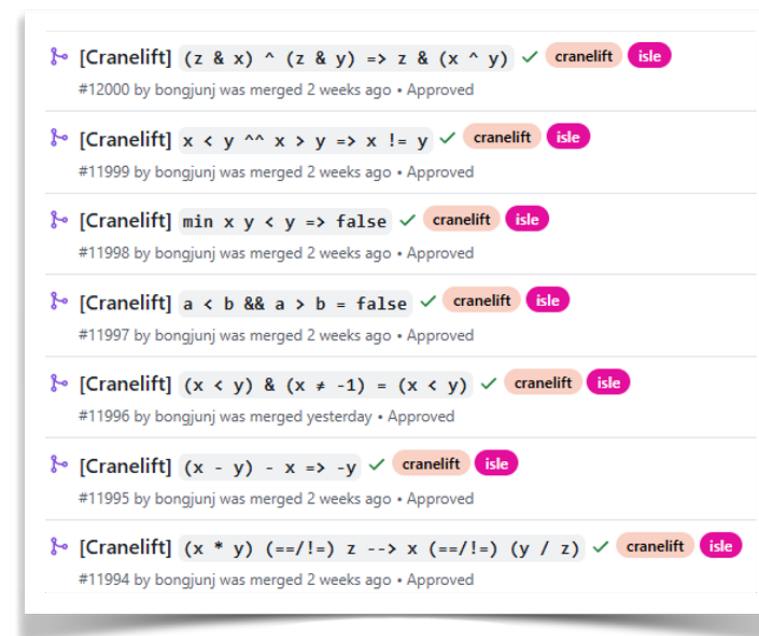
97

WASM 컴파일러에 이식



6%

컴파일된 프로그램 성능 향상



성과

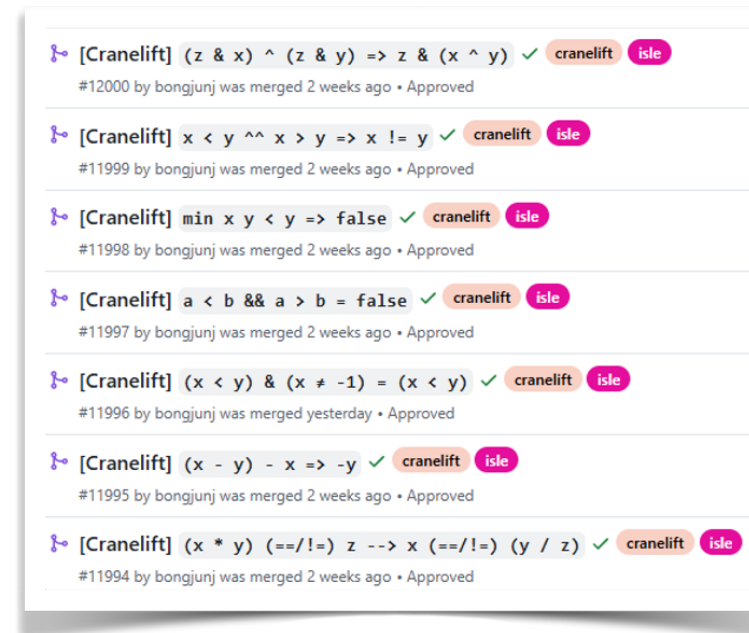


6,266

이식가능한 꼬마 최적화 규칙 생성

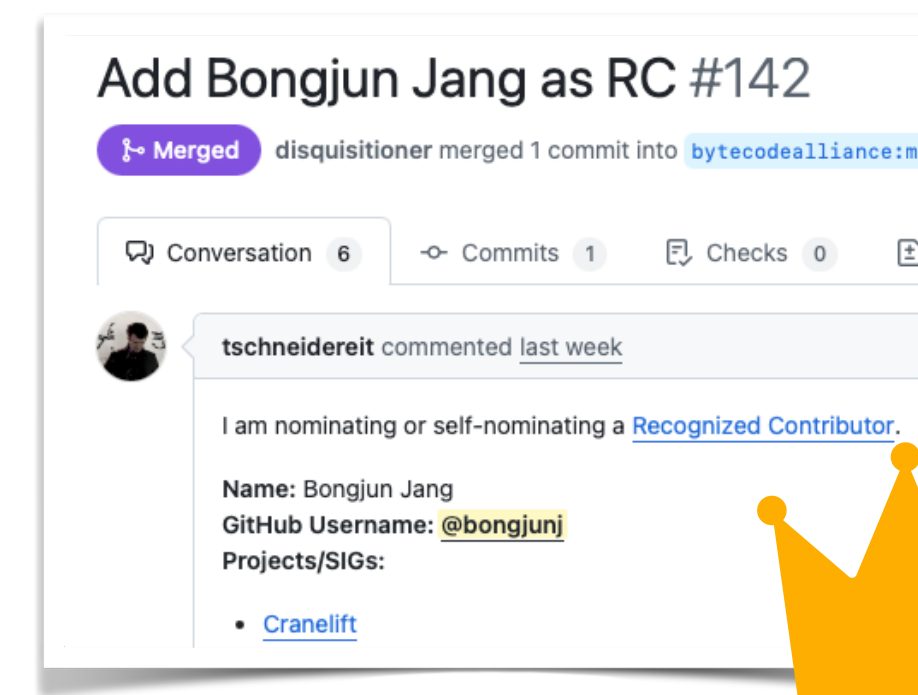
97

WASM 컴파일러에 이식



6%

컴파일된 프로그램 성능 향상



Bytecode Alliance 선정 알아주는 개발자

WASM 기술개발
비영리 단체

Recognized Contributor

시스템 개요: TransOpt

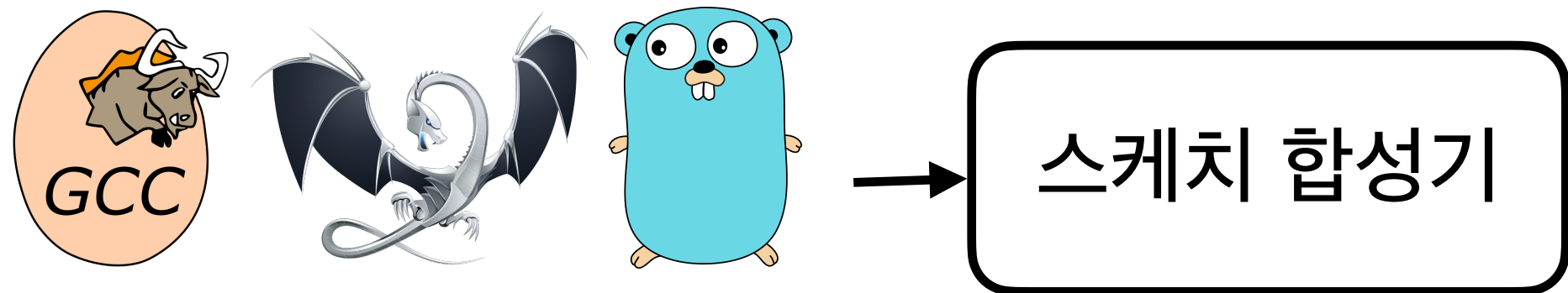


성숙한 컴파일러



새로운 컴파일러

시스템 개요: TransOpt

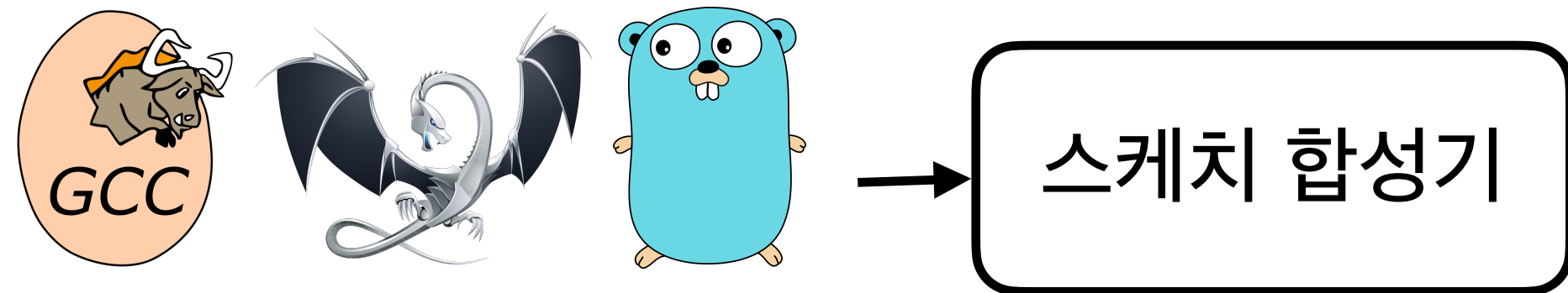


성숙한 컴파일러



새로운 컴파일러

시스템 개요: TransOpt



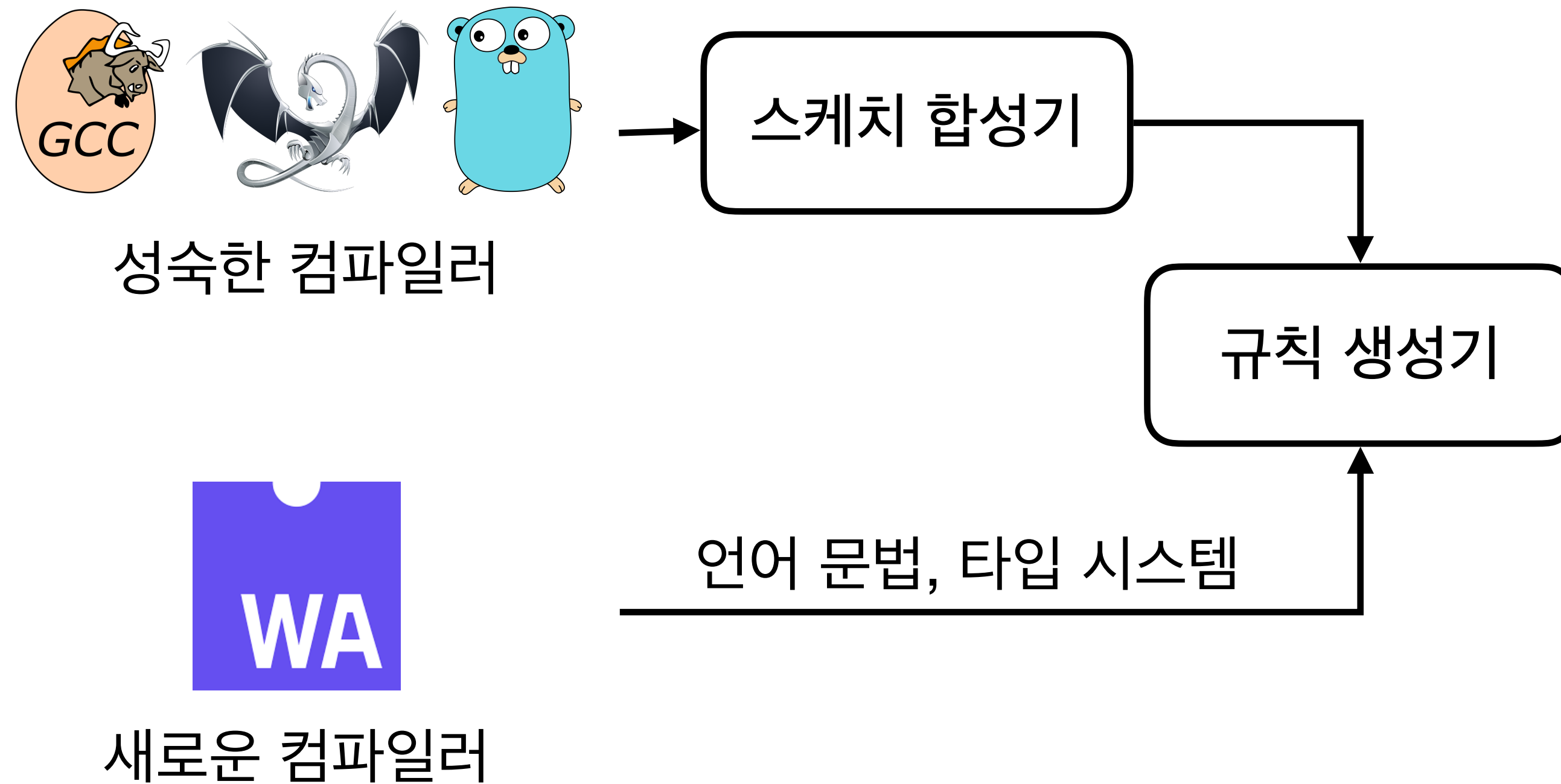
성숙한 컴파일러



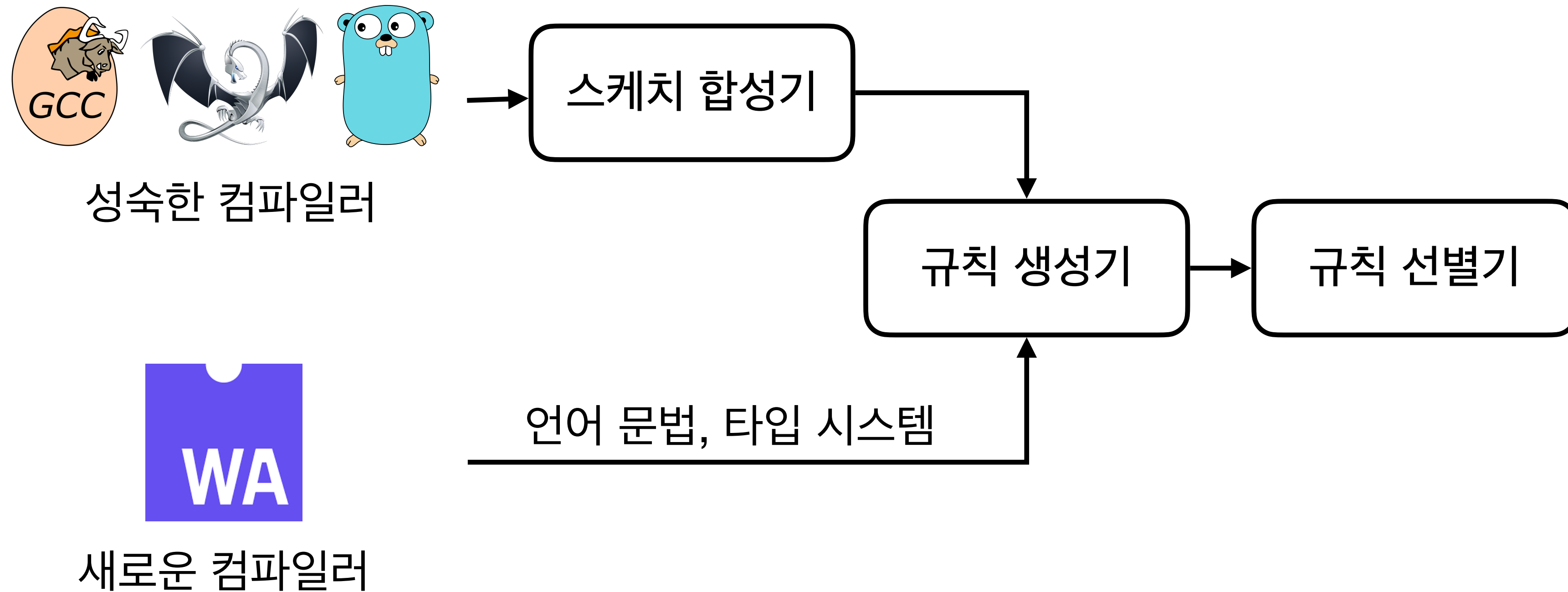
새로운 컴파일러

언어 문법, 타입 시스템

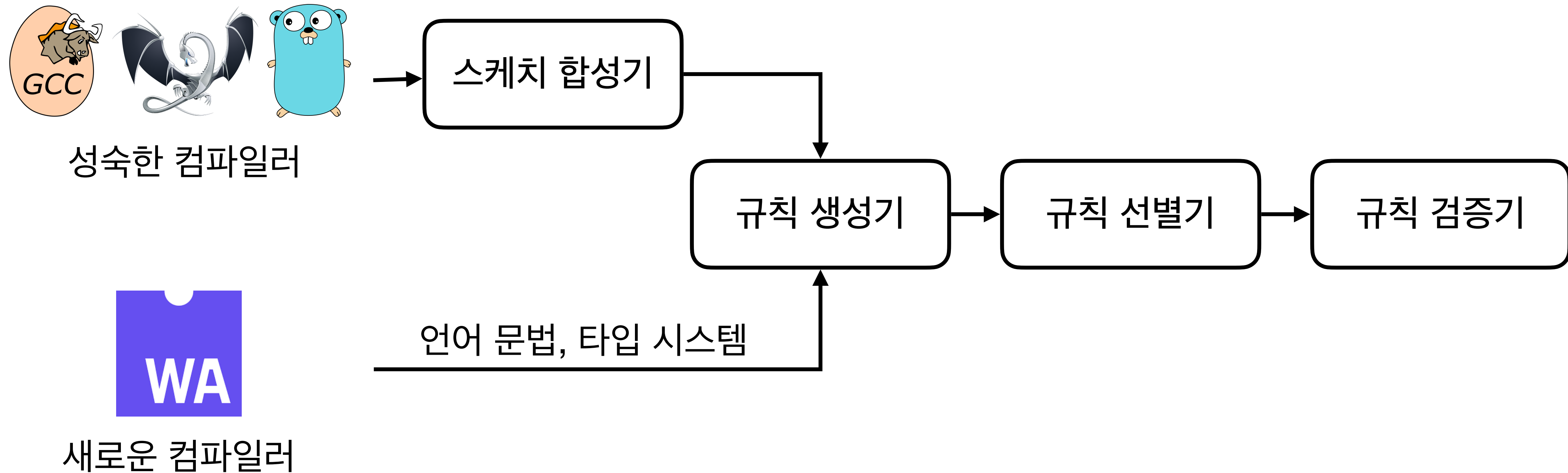
시스템 개요: TransOpt



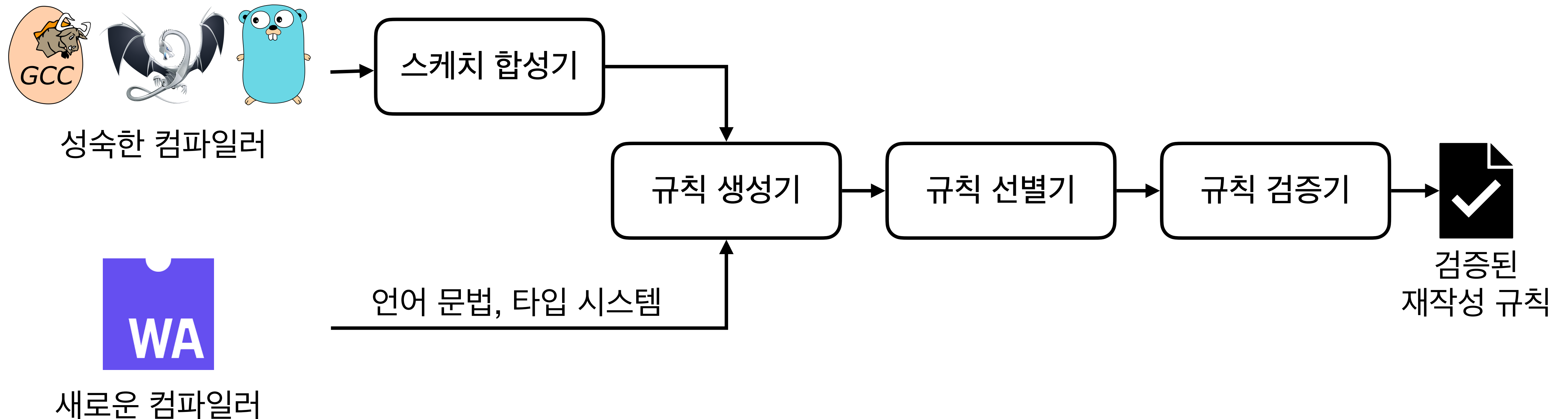
시스템 개요: TransOpt



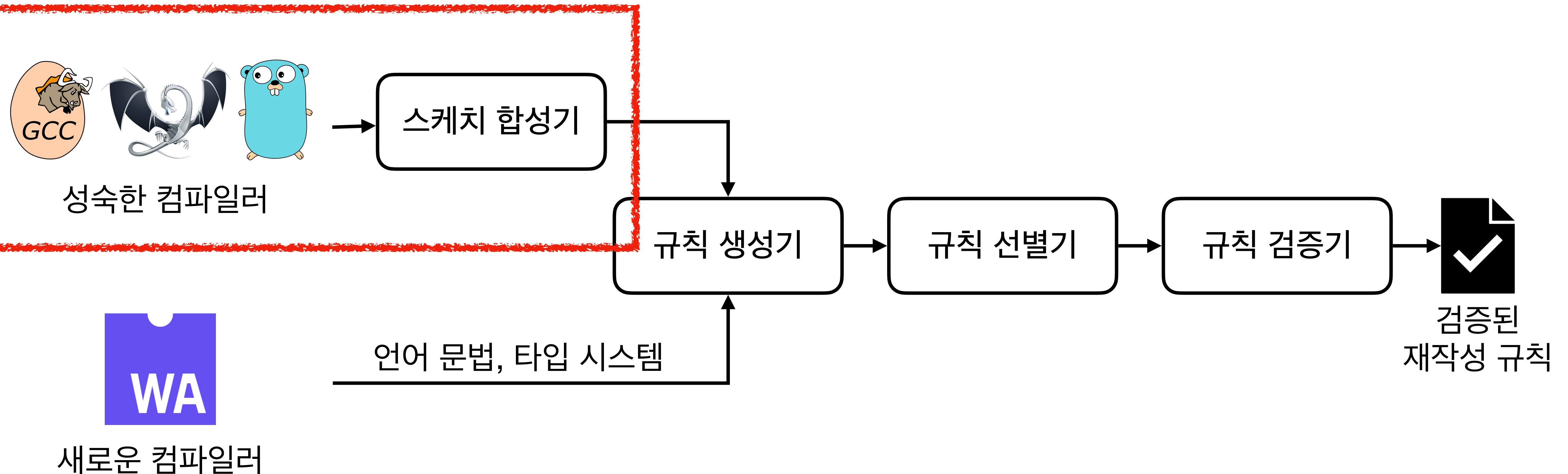
시스템 개요: TransOpt



시스템 개요: TransOpt



시스템 개요: TransOpt



성숙한 컴파일러에서 최적화 아이디어 추출

성숙한 컴파일러에서 최적화 아이디어 추출

- LLVM: C++로 최적화 규칙 구현



성숙한 컴파일러에서 최적화 아이디어 추출

- LLVM: C++로 최적화 규칙 구현
 - 최적화기 입출력 예제 참고



```
%y = mul i32 %x, 5  
%z = icmp eq i32 %y, 30
```



```
%z = icmp eq i32 %x, 6
```

$$X \times 5 = 30 \Rightarrow X = 6$$

성숙한 컴파일러에서 최적화 아이디어 추출

- LLVM: C++로 최적화 규칙 구현
 - 최적화기 입출력 예제 참고
- GCC/Go: 재작성 규칙 DSL을 이용해 구현



```
%y = mul i32 %x, 5  
%z = icmp eq i32 %y, 30
```



```
%z = icmp eq i32 %x, 6
```

$$X \times 5 = 30 \Rightarrow X = 6$$

성숙한 컴파일러에서 최적화 아이디어 추출

- LLVM: C++로 최적화 규칙 구현
 - 최적화기 입출력 예제 참고
- GCC/Go: 재작성 규칙 DSL을 이용해 구현
 - 재작성 규칙을 그대로 참고

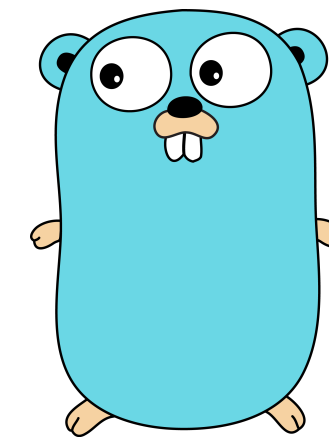


```
%y = mul i32 %x, 5  
%z = icmp eq i32 %y, 30
```



```
%z = icmp eq i32 %x, 6
```

$$X \times 5 = 30 \Rightarrow X = 6$$



```
(Mul32 (Const32 [0]) _) => (Const(32) [0])
```

$$X \times 0 \Rightarrow 0$$



```
(simplify (bit_and @0 integer_all_onesp) @0)
```

$$X \& 0b11..11 \Rightarrow X$$

언어모델을 이용해 스케치 합성



```
%y = mul i32 %x, 5  
%z = icmp eq i32 %y, 30
```



```
%z = icmp eq i32 %x, 6
```

$$X \times 5 = 30 \Rightarrow X = 6$$

언어모델을 이용해 스케치 합성

- 스케치 = 최적화의 핵심 아이디어



```
%y = mul i32 %x, 5  
%z = icmp eq i32 %y, 30
```



```
%z = icmp eq i32 %x, 6
```

$$X \times 5 = 30 \Rightarrow X = 6$$

언어모델을 이용해 스케치 합성

- 스케치 = 최적화의 핵심 아이디어
- 상수간 관계, 최적화 조건 추론



```
%y = mul i32 %x, 5  
%z = icmp eq i32 %y, 30
```



```
%z = icmp eq i32 %x, 6
```

$$X \times 5 = 30 \Rightarrow X = 6$$

언어모델을 이용해 스케치 합성

- 스케치 = 최적화의 핵심 아이디어
- 상수간 관계, 최적화 조건 추론

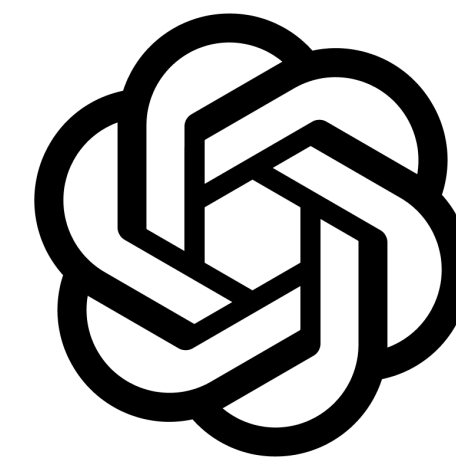


```
%y = mul i32 %x, 5  
%z = icmp eq i32 %y, 30
```



```
%z = icmp eq i32 %x, 6
```

$$X \times 5 = 30 \Rightarrow X = 6$$



언어모델을 이용해 스케치 합성

- 스케치 = 최적화의 핵심 아이디어
- 상수간 관계, 최적화 조건 추론



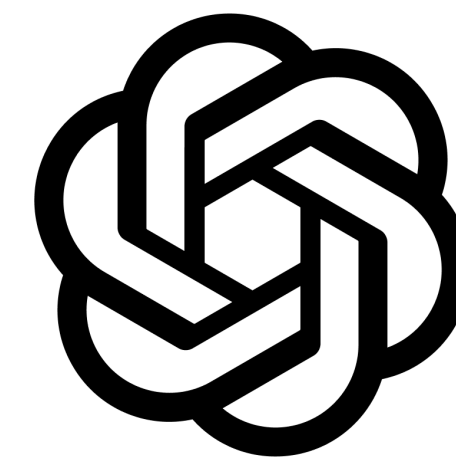
$$\frac{M \% N = 0 \quad N \neq 0}{X \times N = M \Rightarrow X = M / N}$$

```
%y = mul i32 %x, 5  
%z = icmp eq i32 %y, 30
```



```
%z = icmp eq i32 %x, 6
```

$$X \times 5 = 30 \Rightarrow X = 6$$



언어모델을 이용해 스케치 합성

- 스케치 = 최적화의 핵심 아이디어
- 상수간 관계, 최적화 조건 추론



$$\frac{M \% N = 0 \quad N \neq 0}{X \times N = M \Rightarrow X = M / N}$$

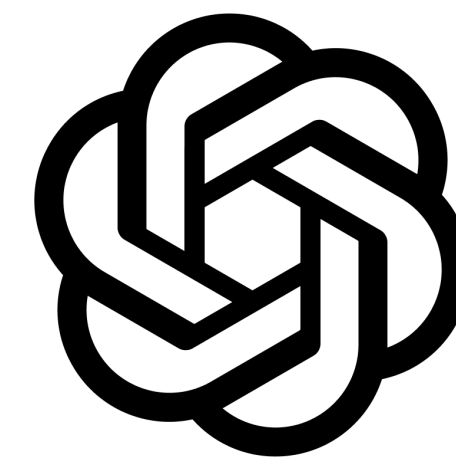
```
%y = mul i32 %x, 5  
%z = icmp eq i32 %y, 30
```



```
%z = icmp eq i32 %x, 6
```

$$X \times 5 = 30 \Rightarrow X = 6$$

$$6 = 30 / 5$$



언어모델을 이용해 스케치 합성

- 스케치 = 최적화의 핵심 아이디어
- 상수간 관계, 최적화 조건 추론

나눠떨어져야..

$$M \% N = 0 \quad N \neq 0$$

$$X \times N = M \Rightarrow X = M / N$$



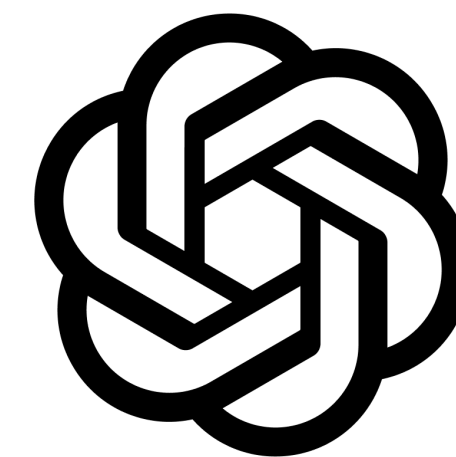
```
%y = mul i32 %x, 5  
%z = icmp eq i32 %y, 30
```



```
%z = icmp eq i32 %x, 6
```

$$X \times 5 = 30 \Rightarrow X = 6$$

$$6 = 30 / 5$$



언어모델을 이용해 스케치 합성

- 스케치 = 최적화의 핵심 아이디어
- 상수간 관계, 최적화 조건 추론

$$S ::= \frac{P^*}{P \Rightarrow P}$$

$$P ::= P \text{ binop } P \mid \text{unop } P \mid P \text{ cmp } P \mid n \mid X \mid Y \mid Z \mid \dots$$

$$\text{binop} ::= + \mid - \mid \times \mid \% \mid \div \mid \& \mid \mid \mid \oplus \mid \gg \mid \ll$$

...

나눠떨어져야..

$$M \% N = 0 \quad N \neq 0$$

$$X \times N = M \Rightarrow X = M / N$$



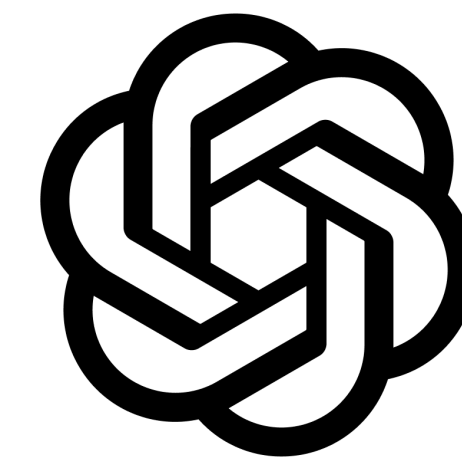
```
%y = mul i32 %x, 5
%z = icmp eq i32 %y, 30
```



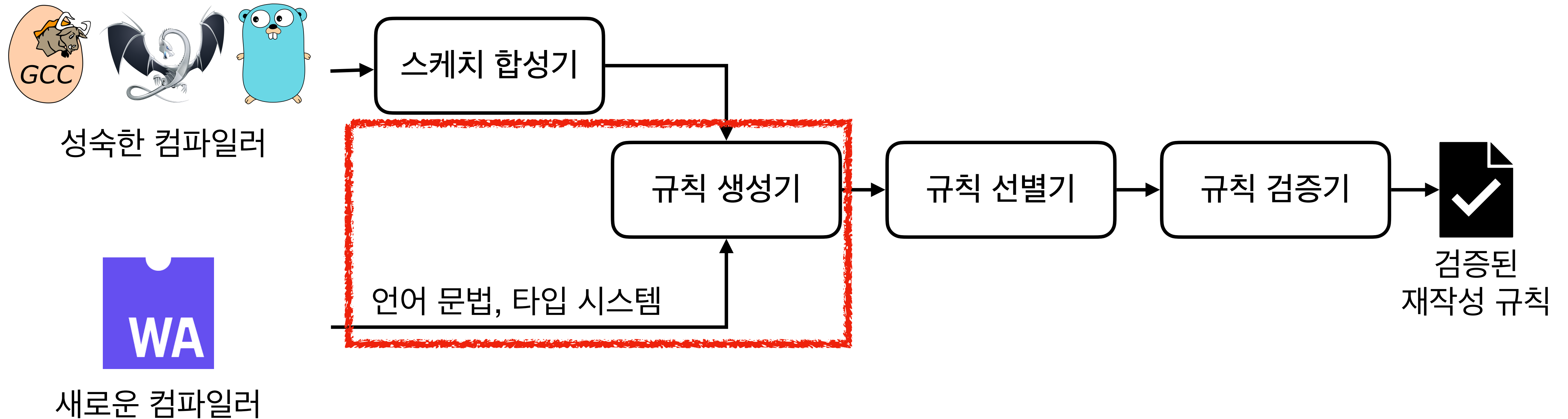
```
%z = icmp eq i32 %x, 6
```

$$X \times 5 = 30 \Rightarrow X = 6$$

$$6 = 30 / 5$$



규칙 생성 단계



재작성 규칙: 컴파일러 최적화 표현 수단

$$\frac{M \% N = 0 \quad N \neq 0}{X \times N = M \Rightarrow X = M / N}$$

스케치

재작성 규칙: 컴파일러 최적화 표현 수단

- 좌변에 있는 느린 프로그램을 우변에 있는 빠른 프로그램으로 재작성

$$\frac{M \% N = 0 \quad N \neq 0}{X \times N = M \Rightarrow X = M / N}$$

스케치

재작성 규칙: 컴파일러 최적화 표현 수단

- 좌변에 있는 느린 프로그램을 우변에 있는 빠른 프로그램으로 재작성

$$\begin{array}{c} \text{패턴} \Rightarrow \frac{M \% N = 0 \quad N \neq 0}{X \times N = M \Rightarrow X = M / N} \\ \text{스케치} \end{array}$$

재작성 규칙: 컴파일러 최적화 표현 수단

- 좌변에 있는 느린 프로그램을 우변에 있는 빠른 프로그램으로 재작성

전제조건

$$M \% N = 0 \quad N \neq 0$$

패턴

$$X \times N = M \Rightarrow X = M / N$$

스케치

재작성 규칙: 컴파일러 최적화 표현 수단

- 좌변에 있는 느린 프로그램을 우변에 있는 빠른 프로그램으로 재작성

전제조건

$$M \% N = 0 \quad N \neq 0$$

패턴

$$X \times N = M \Rightarrow X = M / N$$

재작성 식

스케치

재작성 규칙: 컴파일러 최적화 표현 수단

- 좌변에 있는 느린 프로그램을 우변에 있는 빠른 프로그램으로 재작성

전제조건

$$M \% N = 0 \quad N \neq 0$$

패턴

$$X \times N = M \Rightarrow X = M / N$$

재작성 식

스케치

$$m \text{ rem}_C n = 0 \quad n \neq 0$$

$$(x \text{ mul } n) \text{ eq } m \Rightarrow x \text{ eq } (m \text{ div}_C n)$$

WASM 컴파일러 재작성 규칙

스케치를 재작성 규칙으로 구체화

$$\frac{M \% N = 0 \quad N \neq 0}{X \times N = M \Rightarrow X = M / N}$$

스케치

스케치를 재작성 규칙으로 구체화

$(x \text{ mul}_R n) \text{ eq } m$

$$M \% N = 0 \quad N \neq 0$$

$$\frac{}{X \times N = M \Rightarrow X = M / N}$$

스케치

스케치를 재작성 규칙으로 구체화



WASM 명령어 문법



`(x mulR n) eq m`

$$M \% N = 0 \quad N \neq 0$$

$$X \times N = M \Rightarrow X = M / N$$

스케치

스케치를 재작성 규칙으로 구체화



WASM 명령어 문법



$(x \text{ mul}_R n) \text{ eq } m$

$m \text{ rem}_C n = 0 \quad n \neq 0$

$M \% N = 0 \quad N \neq 0$

$X \times N = M \Rightarrow X = M / N$

스케치

스케치를 재작성 규칙으로 구체화



WASM 명령어 문법



$(x \text{ mul}_R n) \text{ eq } m$

$m \text{ rem}_C n = 0 \quad n \neq 0$

$M \% N = 0 \quad N \neq 0$

$X \times N = M \Rightarrow X = M / N$

$x \text{ eq } (m \text{ div}_C n)$

스케치

스케치를 재작성 규칙으로 구체화



WASM 명령어 문법



$(x \text{ mul}_R n) \text{ eq } m$

$$M \% N = 0 \quad N \neq 0$$

$$\frac{}{X \times N = M \Rightarrow X = M / N}$$

스케치

$$m \text{ rem}_C n = 0 \quad n \neq 0$$

$$x \text{ eq } (m \text{ div}_C n)$$

$$\frac{m \text{ rem}_C n = 0 \quad n \neq 0}{(x \text{ mul } n) \text{ eq } m \Rightarrow x \text{ eq } (m \text{ div}_C n)}$$

구체화된 재작성 규칙

스케치를 재작성 규칙으로 구체화



WASM 명령어 문법



$$m \text{ rem}_C n = 0 \quad n \neq 0$$

$$M \% N = 0 \quad N \neq 0$$

$$(x \text{ mul}_R n) \text{ eq } m$$

$$\frac{}{X \times N = M \Rightarrow X = M / N}$$

$$x \text{ eq } (m \text{ div}_C n)$$

$$x \text{ eq } (m \text{ div}_R n)$$

스케치

$$\frac{m \text{ rem}_C n = 0 \quad n \neq 0}{(x \text{ mul } n) \text{ eq } m \Rightarrow x \text{ eq } (m \text{ div}_C n)}$$

$$(x \text{ mul } n) \text{ eq } m \Rightarrow x \text{ eq } (m \text{ div}_R n)$$

구체화된 재작성 규칙

스케치를 재작성 규칙으로 구체화



WASM 명령어 문법



$(x \text{ mul}_R n) \text{ eq } m$

$$M \% N = 0 \quad N \neq 0$$

$$\frac{M \% N = 0 \quad N \neq 0}{X \times N = M \Rightarrow X = M / N}$$

스케치

$$m \text{ rem}_C n = 0 \quad n \neq 0$$

컴파일중 미리 계산

$$x \text{ eq } (m \text{ div}_C n)$$

$$x \text{ eq } (m \text{ div}_R n)$$

실행시 계산

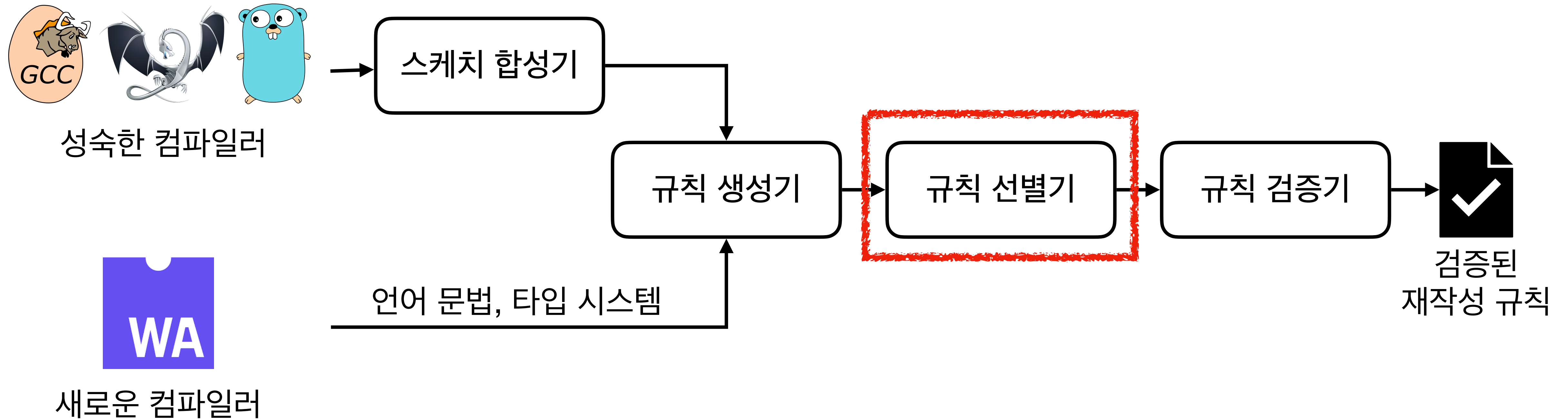
$$\frac{m \text{ rem}_C n = 0 \quad n \neq 0}{(x \text{ mul } n) \text{ eq } m \Rightarrow x \text{ eq } (m \text{ div}_C n)}$$

$$(x \text{ mul } n) \text{ eq } m \Rightarrow x \text{ eq } (m \text{ div}_C n)$$

$$(x \text{ mul } n) \text{ eq } m \Rightarrow x \text{ eq } (m \text{ div}_R n)$$

구체화된 재작성 규칙

규칙 선별 단계



실행 성능 향상시키는 규칙만 고르기

- 비용 모델 $Cost : \text{명령어} \rightarrow \text{비용}$

$$m \text{ rem}_C n = 0 \quad n \neq 0$$

$$M \% N = 0 \quad N \neq 0$$

컴파일중 미리 계산

$$(x \text{ mul}_R n) \text{ eq } m$$

$$X \times N = M \Rightarrow X = M / N$$

$$x \text{ eq } (m \text{ div}_C n)$$

스케치

$$x \text{ eq } (m \text{ div}_R n)$$

실행시 계산

$$m \text{ rem}_C n = 0 \quad n \neq 0$$

$$(x \text{ mul } n) \text{ eq } m \Rightarrow x \text{ eq } (m \text{ div}_C n)$$

$$(x \text{ mul } n) \text{ eq } m \Rightarrow x \text{ eq } (m \text{ div}_R n)$$

구체화된 재작성 규칙

실행 성능 향상시키는 규칙만 고르기

- 비용 모델 $Cost$: 명령어 $\rightarrow$ 비용

$$m \text{ rem}_C n = 0 \quad n \neq 0$$

$$M \% N = 0 \quad N \neq 0$$

$$(x \text{ mul}_R n) \text{ eq } m$$

$$X \times N = M \Rightarrow X = M / N$$

스케치

$$x \text{ eq } (m \text{ div}_C n)$$

$$x \text{ eq } (m \text{ div}_R n)$$

나누기 > 곱하기

실행시 계산

$$m \text{ rem}_C n = 0 \quad n \neq 0$$

$$(x \text{ mul } n) \text{ eq } m \Rightarrow x \text{ eq } (m \text{ div}_C n)$$

$$(x \text{ mul } n) \text{ eq } m \Rightarrow x \text{ eq } (m \text{ div}_R n)$$

구체화된 재작성 규칙

컴파일중 미리 계산

실행 성능 향상시키는 규칙만 고르기

- 비용 모델 $Cost$: 명령어 $\rightarrow$ 비용

$$m \text{ rem}_C n = 0 \quad n \neq 0$$

$$M \% N = 0 \quad N \neq 0$$

$$(x \text{ mul}_R n) \text{ eq } m$$

$$X \times N = M \Rightarrow X = M / N$$

스케치

$$x \text{ eq } (m \text{ div}_C n)$$

$$x \text{ eq } (m \text{ div}_R n)$$

나누기 > 곱하기

실행시 계산

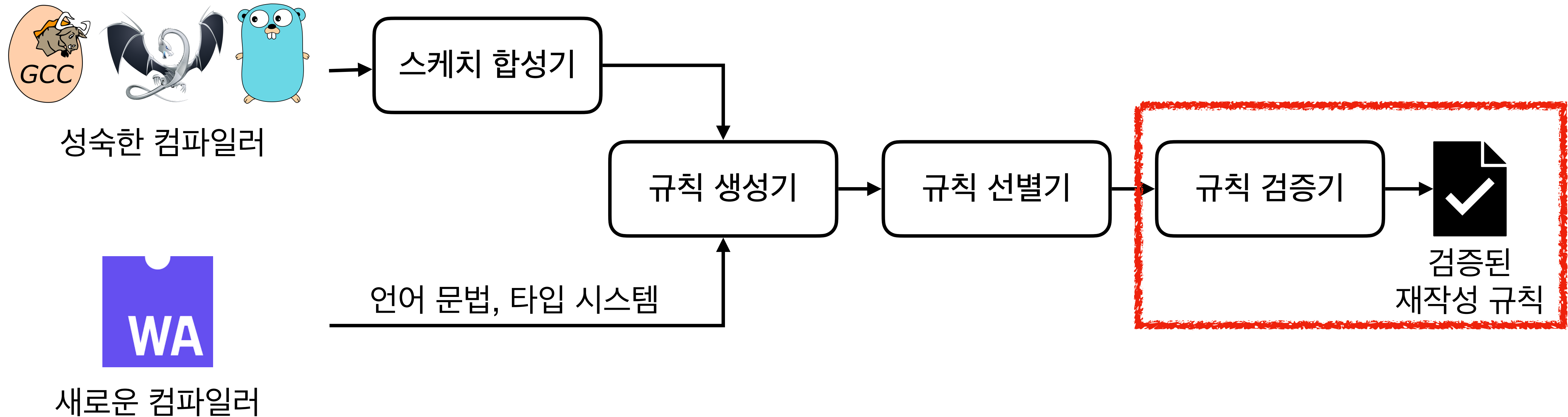
컴파일중 미리 계산

$$\frac{m \text{ rem}_C n = 0 \quad n \neq 0}{(x \text{ mul } n) \text{ eq } m \Rightarrow x \text{ eq } (m \text{ div}_C n)}$$

$$(x \text{ mul } n) \text{ eq } m \Rightarrow x \text{ eq } (m \text{ div}_R n)$$

구체화된 재작성 규칙

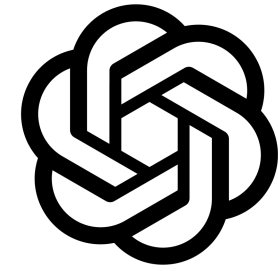
규칙 검증 단계



최적화 전제조건 합성

최적화 전제조건 합성

$$\frac{M \% N = 0 \quad N \neq 0}{X \times N = M \Rightarrow X = M / N}$$



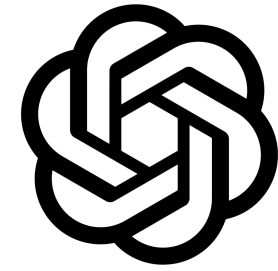
최적화 전제조건 합성

✗
$$\frac{M \% N = 0 \quad N \neq 0}{X \times N = M \Rightarrow X = M / N}$$
 

최적화 전제조건 합성



$$\frac{M \% N = 0 \quad N \neq 0}{X \times N = M \Rightarrow X = M / N}$$

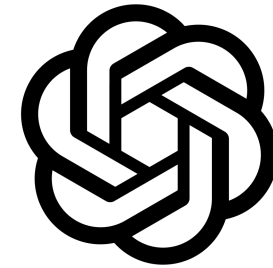


$$\frac{M \% N = 0 \quad N \% 2 = 1}{X \times N = M \Rightarrow X = M / N}$$

최적화 전제조건 합성



$$\frac{M \% N = 0 \quad N \neq 0}{X \times N = M \Rightarrow X = M / N}$$



올바른 추론



$$\frac{M \% N = 0 \quad N \% 2 = 1}{X \times N = M \Rightarrow X = M / N}$$

최적화 전제조건 합성

✗

$M \% N = 0$

$N \neq 0$

$X \times N = M \Rightarrow X = M / N$

올바른 추론

합성기가 보완

✓

$M \% N = 0$

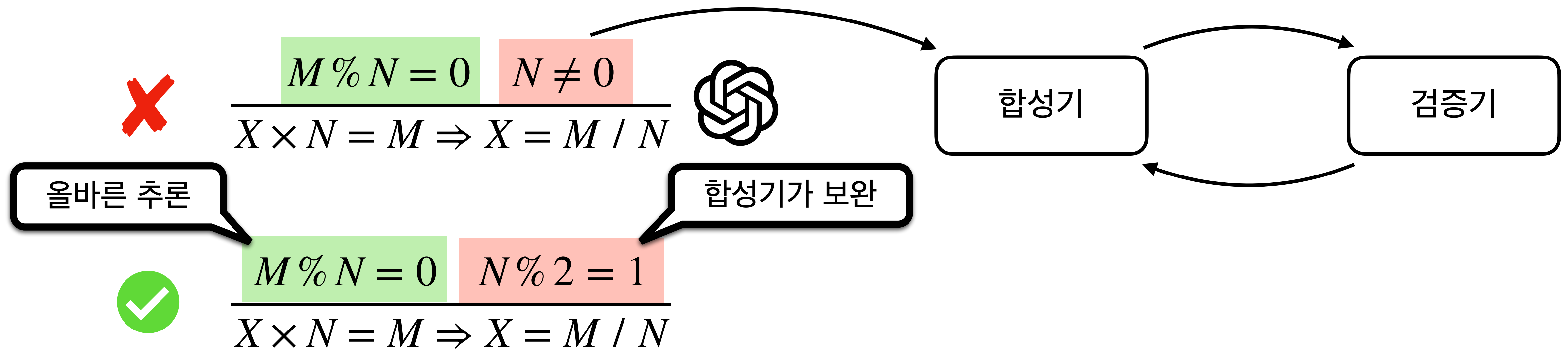
$N \% 2 = 1$

$X \times N = M \Rightarrow X = M / N$

18

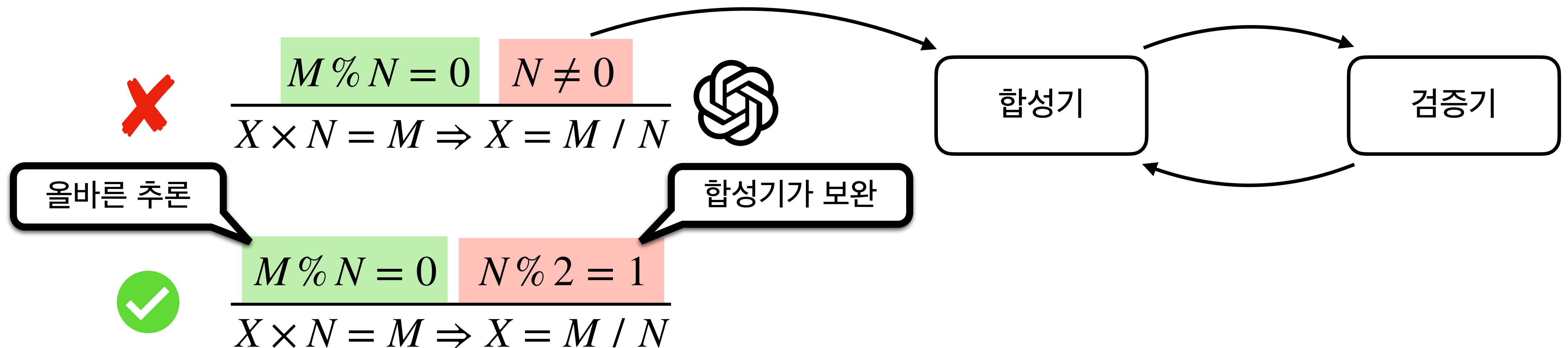
최적화 전제조건 합성

- 나열식(Enumerative) 합성기와 검증기를 이용한 CEGIS(Counter-example Guided Inductive Synthesis)



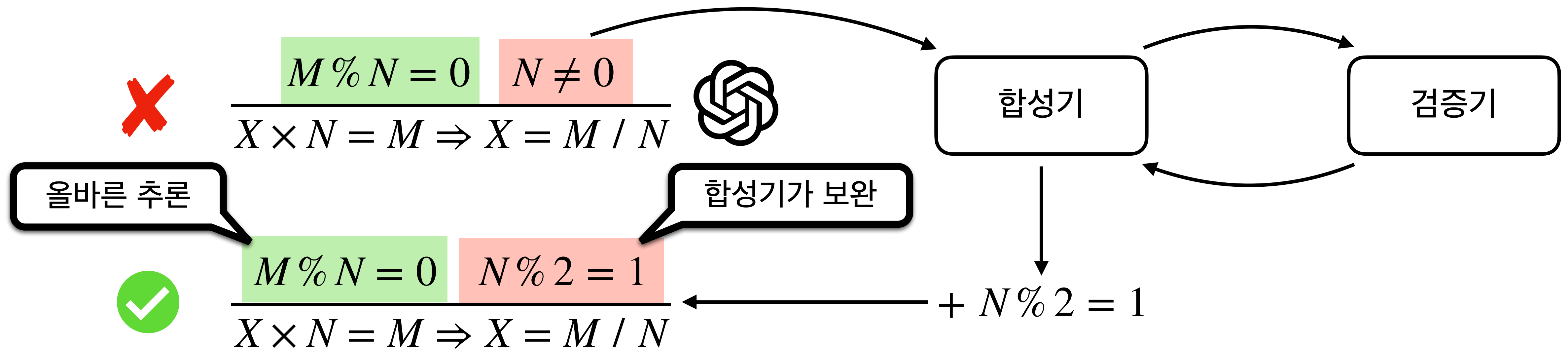
최적화 전제조건 합성

- 나열식(Enumerative) 합성기와 검증기를 이용한 CEGIS(Counter-example Guided Inductive Synthesis)
- 언어모델이 작성한 최적화 전제조건을 통해 합성 가속



최적화 전제조건 합성

- 나열식(Enumerative) 합성기와 검증기를 이용한 CEGIS(Counter-example Guided Inductive Synthesis)
- 언어모델이 작성한 최적화 전제조건을 통해 합성 가속



재작성 규칙 검증

$$\frac{M \% N = 0 \quad N \% 2 = 1}{X \times N = M \Rightarrow X = M / N}$$

재작성 규칙 검증

- 최종적으로 전제조건 하에서 모든 입력에 대해 출력이 동등한지 검사

$$\forall X, N, M \quad \frac{M \% N = 0 \quad N \% 2 = 1}{X \times N = M \Rightarrow X = M / N}$$

재작성 규칙 검증

- 최종적으로 전제조건 하에서 모든 입력에 대해 출력이 동등한지 검사

$$\forall X, N, M \quad \frac{M \% N = 0 \quad N \% 2 = 1}{X \times N = M \Rightarrow X = M / N}$$



재작성 규칙 이식을 위한 후처리

재작성 규칙 이식을 위한 후처리

- 재작성 규칙에서 사용된 변수명 정규화

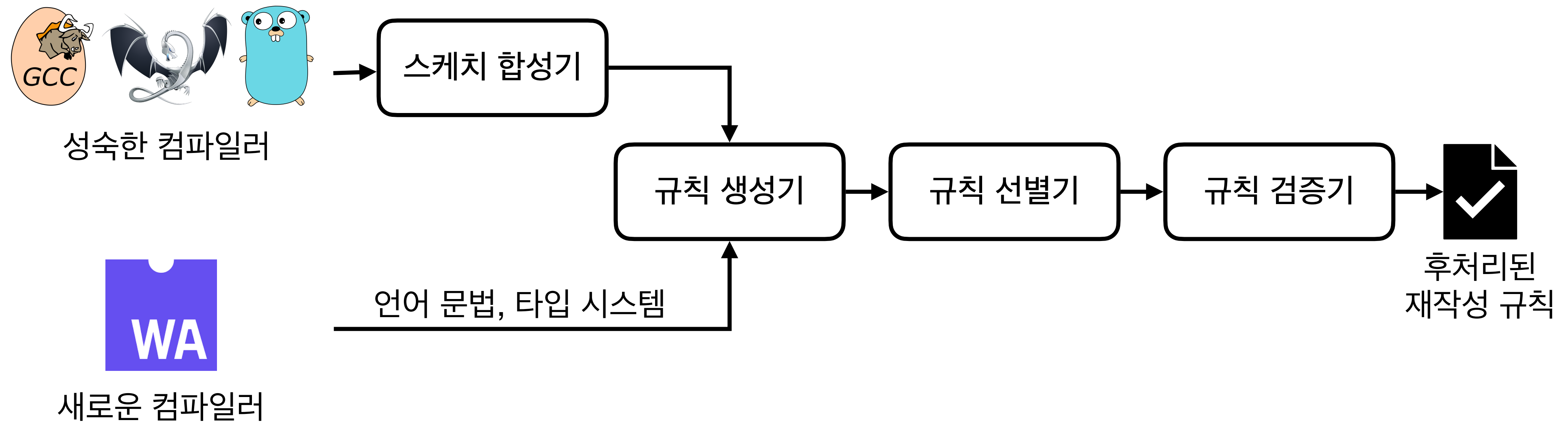
재작성 규칙 이식을 위한 후처리

- 재작성 규칙에서 사용된 변수명 정규화
- 교환법칙, 대칭성을 이용해 재작성 규칙 추가 생성
 - $(X - Y) + Y \Rightarrow X : Y + (X - Y) \Rightarrow X$
 - $\text{select } C \ X \ Y \Rightarrow \dots, \text{select } \neg C \ Y \ X \Rightarrow \dots$

재작성 규칙 이식을 위한 후처리

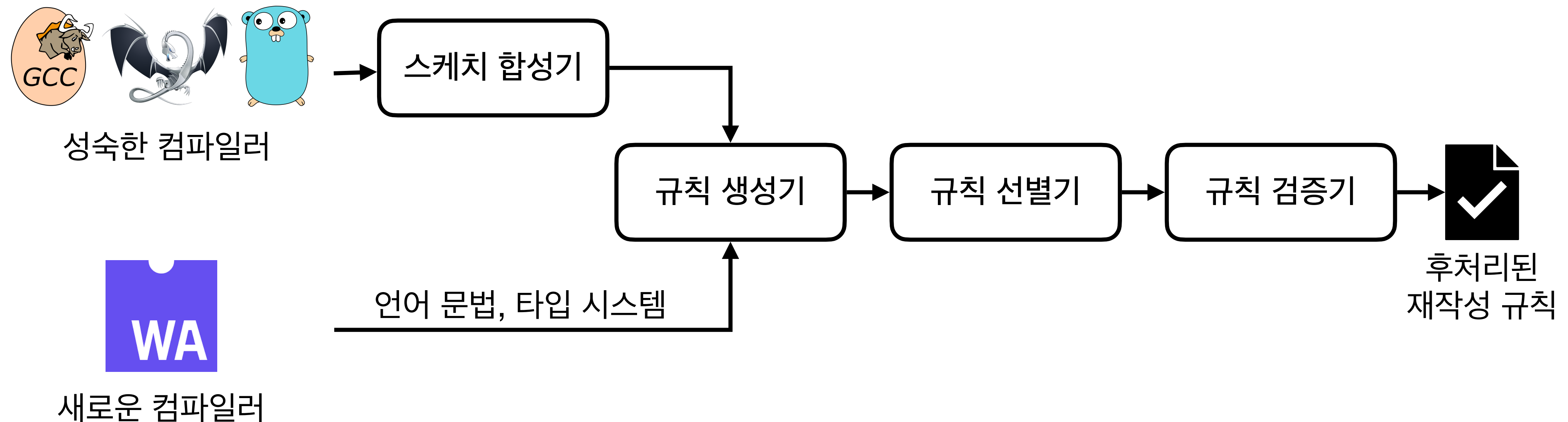
- 재작성 규칙에서 사용된 변수명 정규화
- 교환법칙, 대칭성을 이용해 재작성 규칙 추가 생성
 - $(X - Y) + Y \Rightarrow X : Y + (X - Y) \Rightarrow X$
 - $\text{select } C \ X \ Y \Rightarrow \dots, \text{select } \neg C \ Y \ X \Rightarrow \dots$
- 중복된 규칙 제거

평가: 규칙을 얼마나 효과적으로 이식할 수 있는가?



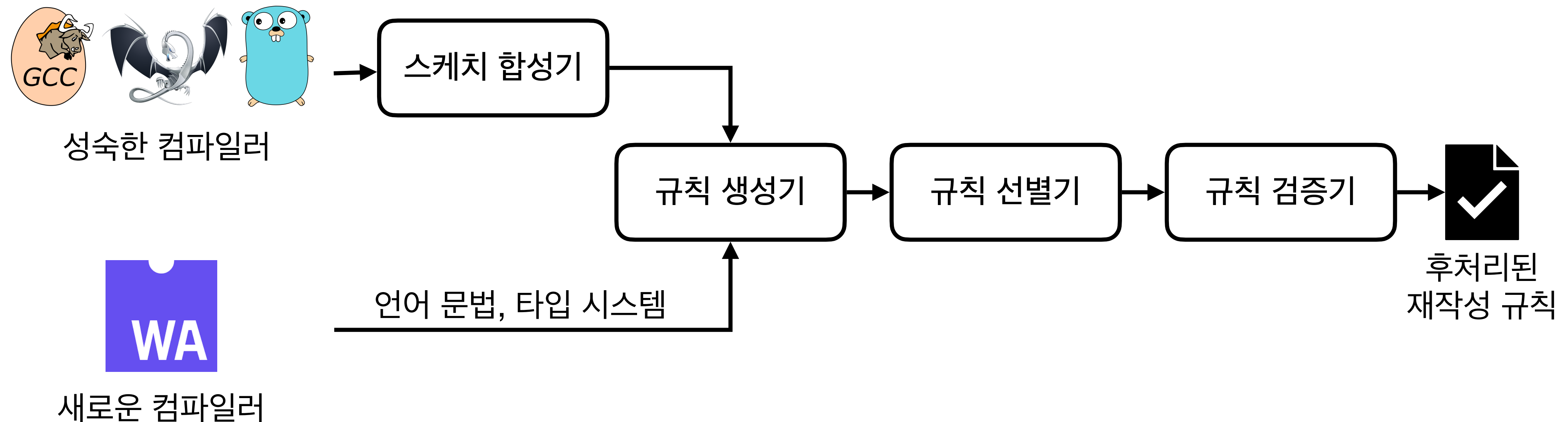
평가: 규칙을 얼마나 효과적으로 이식할 수 있는가?

- 성숙한 컴파일러: LLVM, GCC, Go



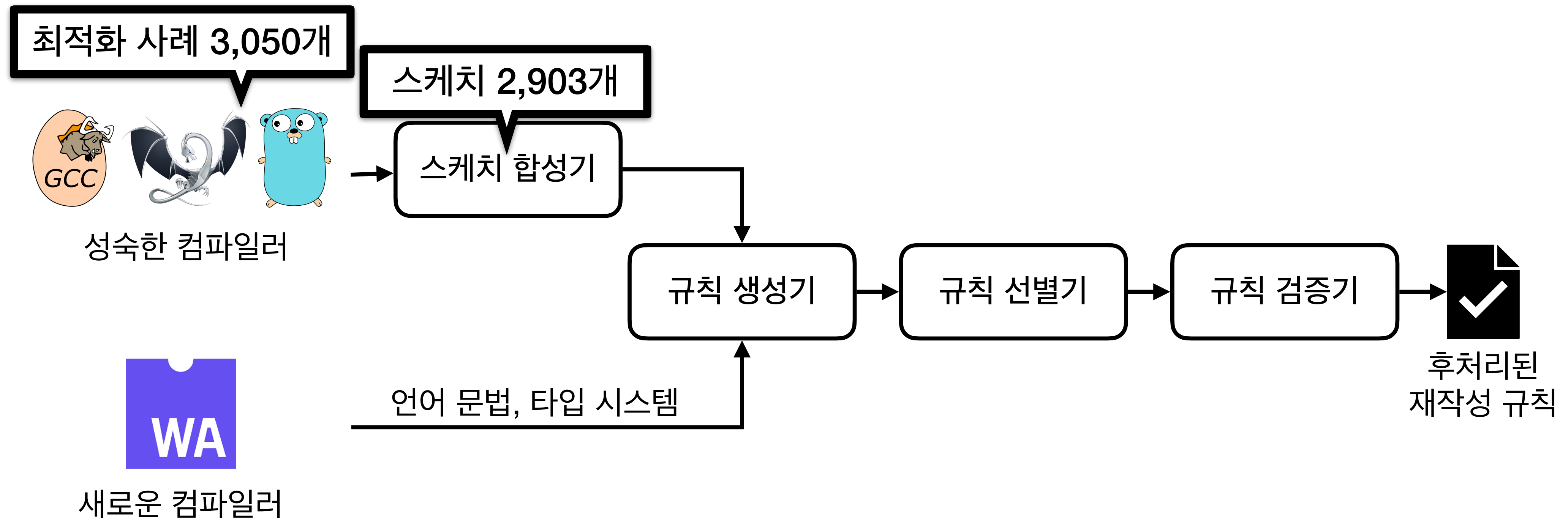
평가: 규칙을 얼마나 효과적으로 이식할 수 있는가?

- 성숙한 컴파일러: LLVM, GCC, Go
- 이식 대상 컴파일러: Cranelift



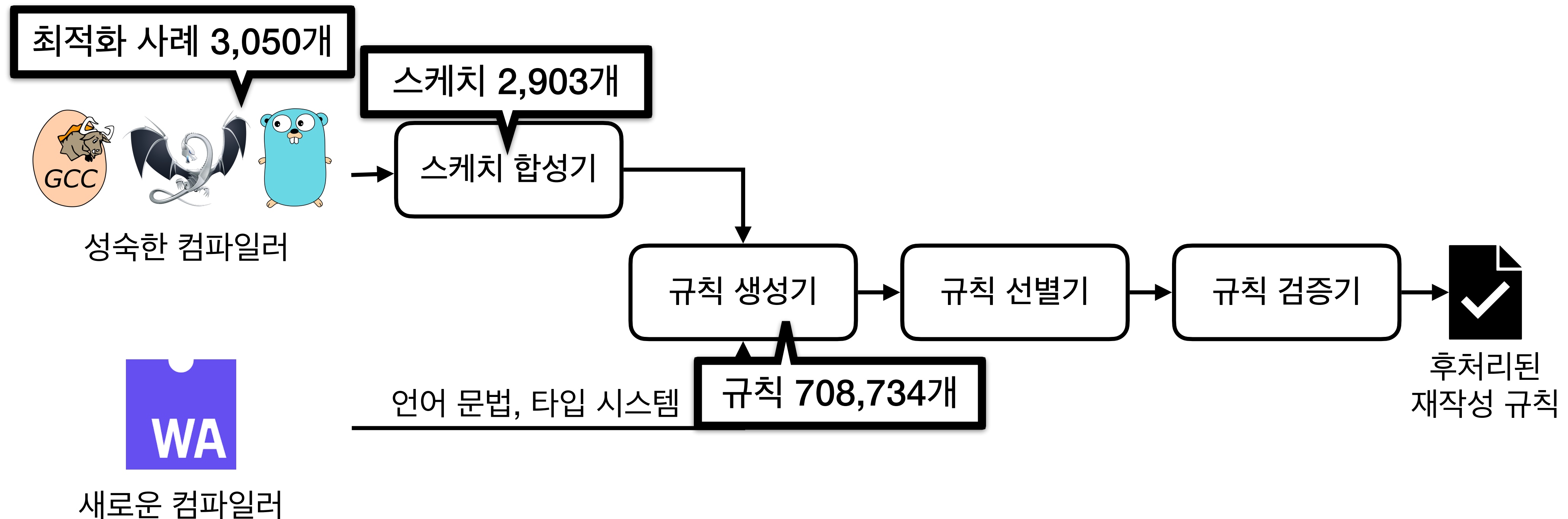
평가: 규칙을 얼마나 효과적으로 이식할 수 있는가?

- 성숙한 컴파일러: LLVM, GCC, Go
- 이식 대상 컴파일러: Cranelift



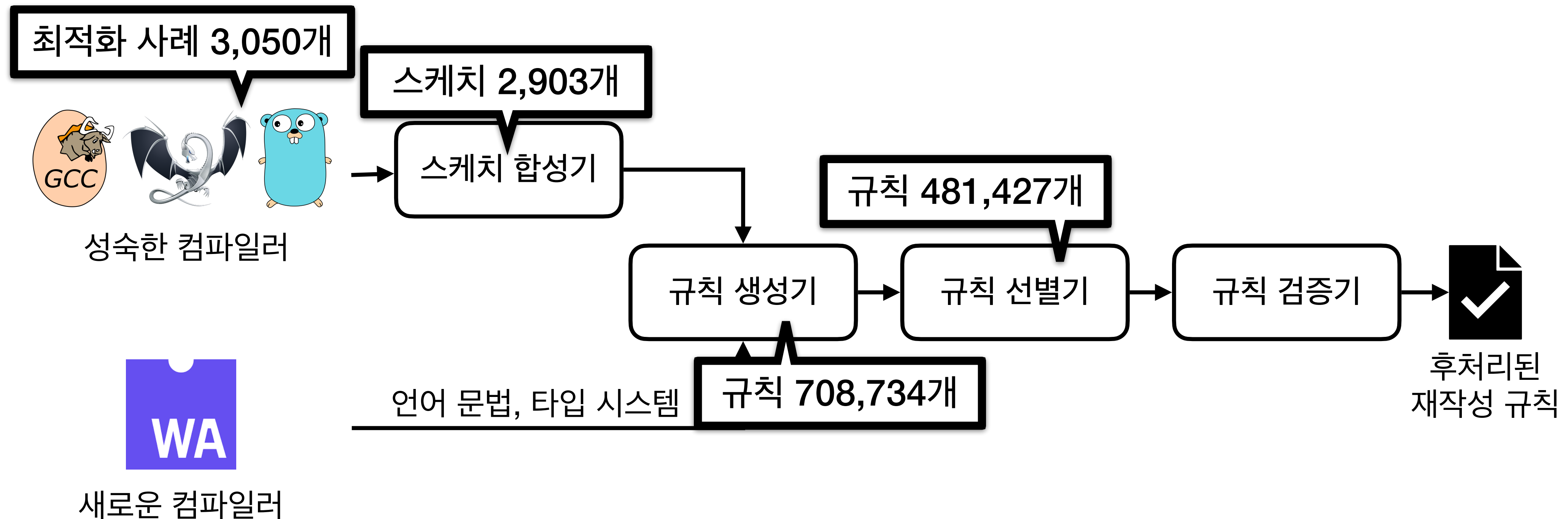
평가: 규칙을 얼마나 효과적으로 이식할 수 있는가?

- 성숙한 컴파일러: LLVM, GCC, Go
- 이식 대상 컴파일러: Cranelift



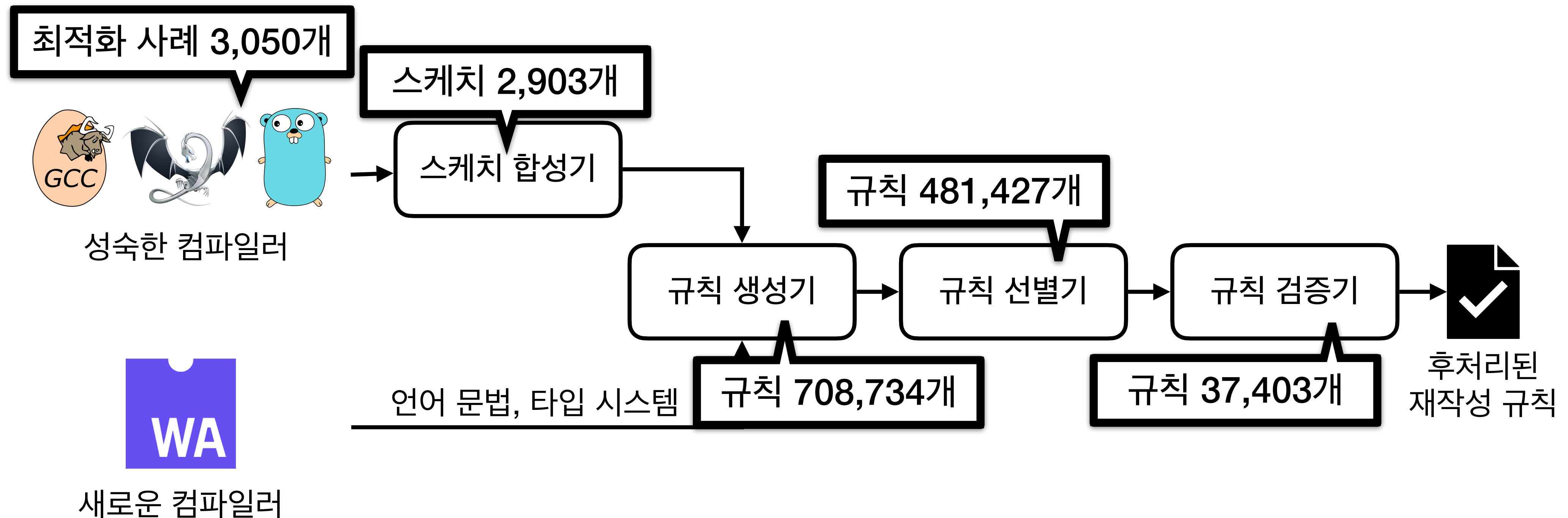
평가: 규칙을 얼마나 효과적으로 이식할 수 있는가?

- 성숙한 컴파일러: LLVM, GCC, Go
- 이식 대상 컴파일러: Cranelift



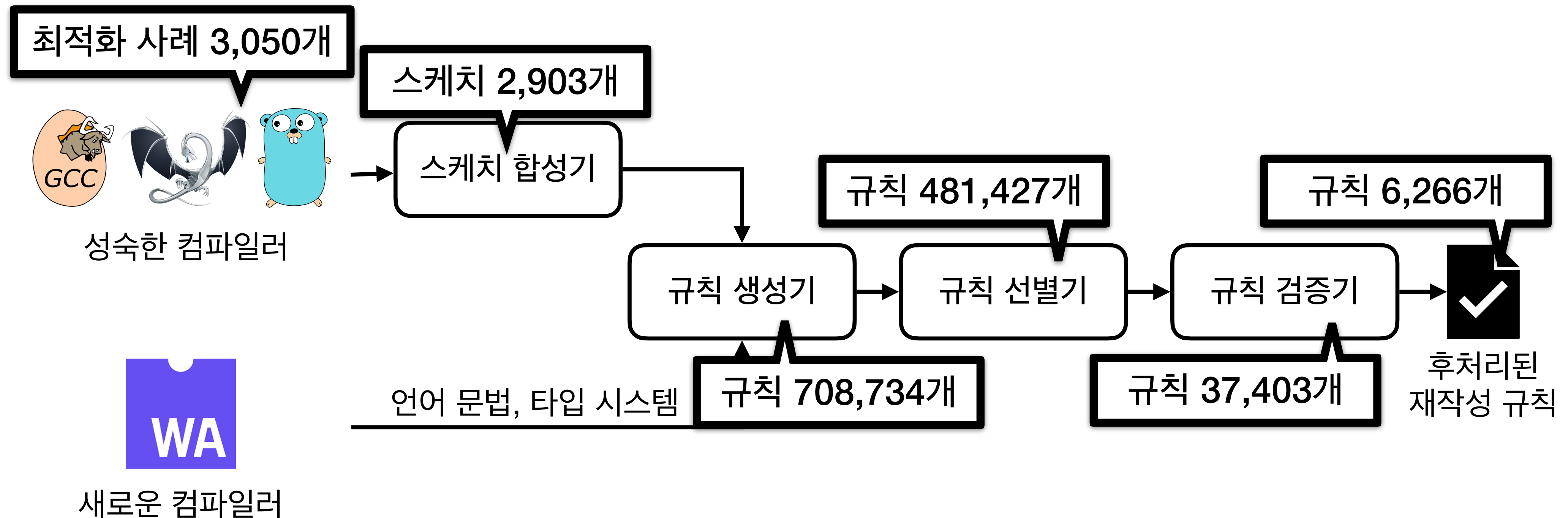
평가: 규칙을 얼마나 효과적으로 이식할 수 있는가?

- 성숙한 컴파일러: LLVM, GCC, Go
- 이식 대상 컴파일러: Cranelift



평가: 규칙을 얼마나 효과적으로 이식할 수 있는가?

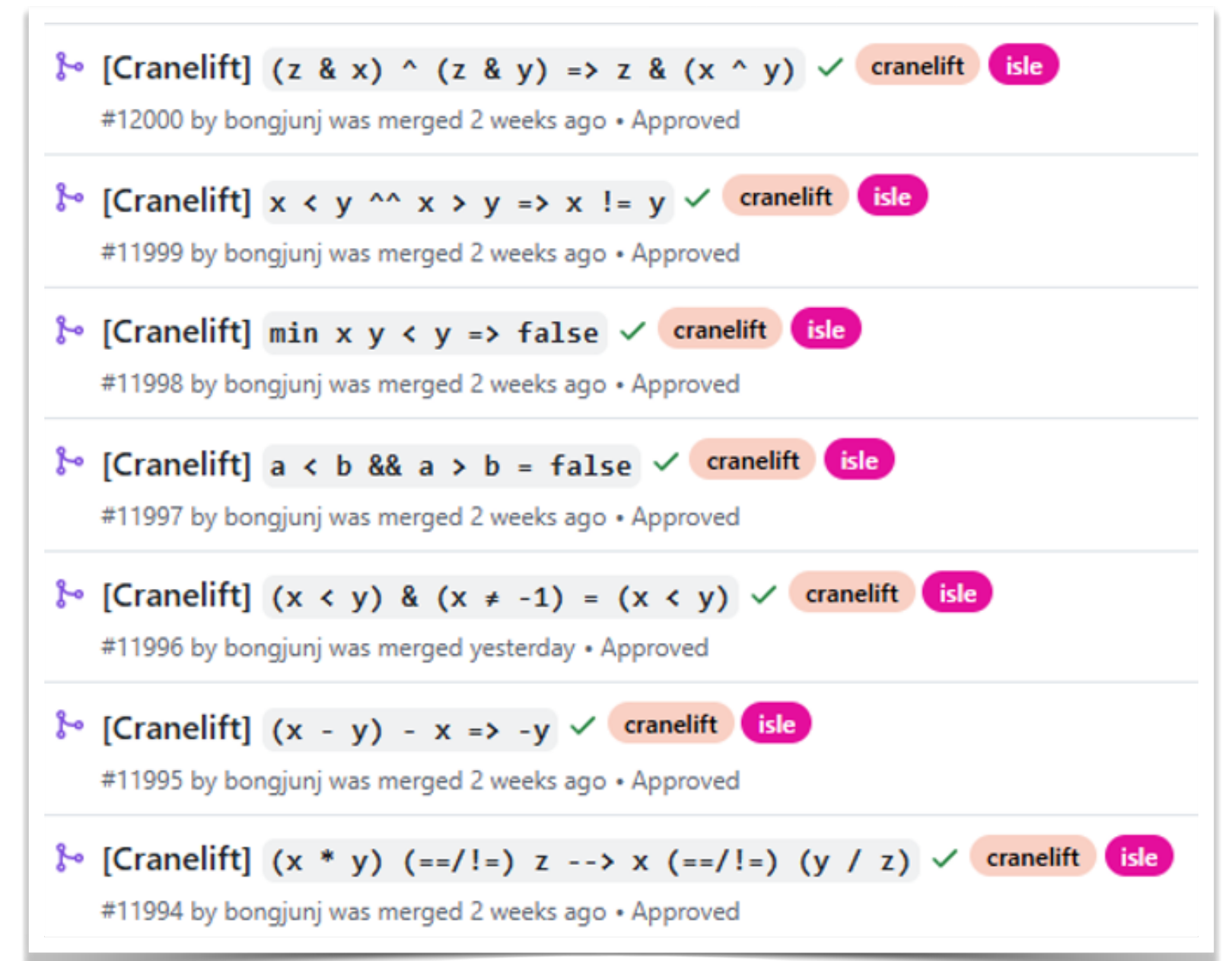
- 성숙한 컴파일러: LLVM, GCC, Go
- 이식 대상 컴파일러: Cranelift



오픈소스 기여: Cranelift에 최적화 규칙 이식

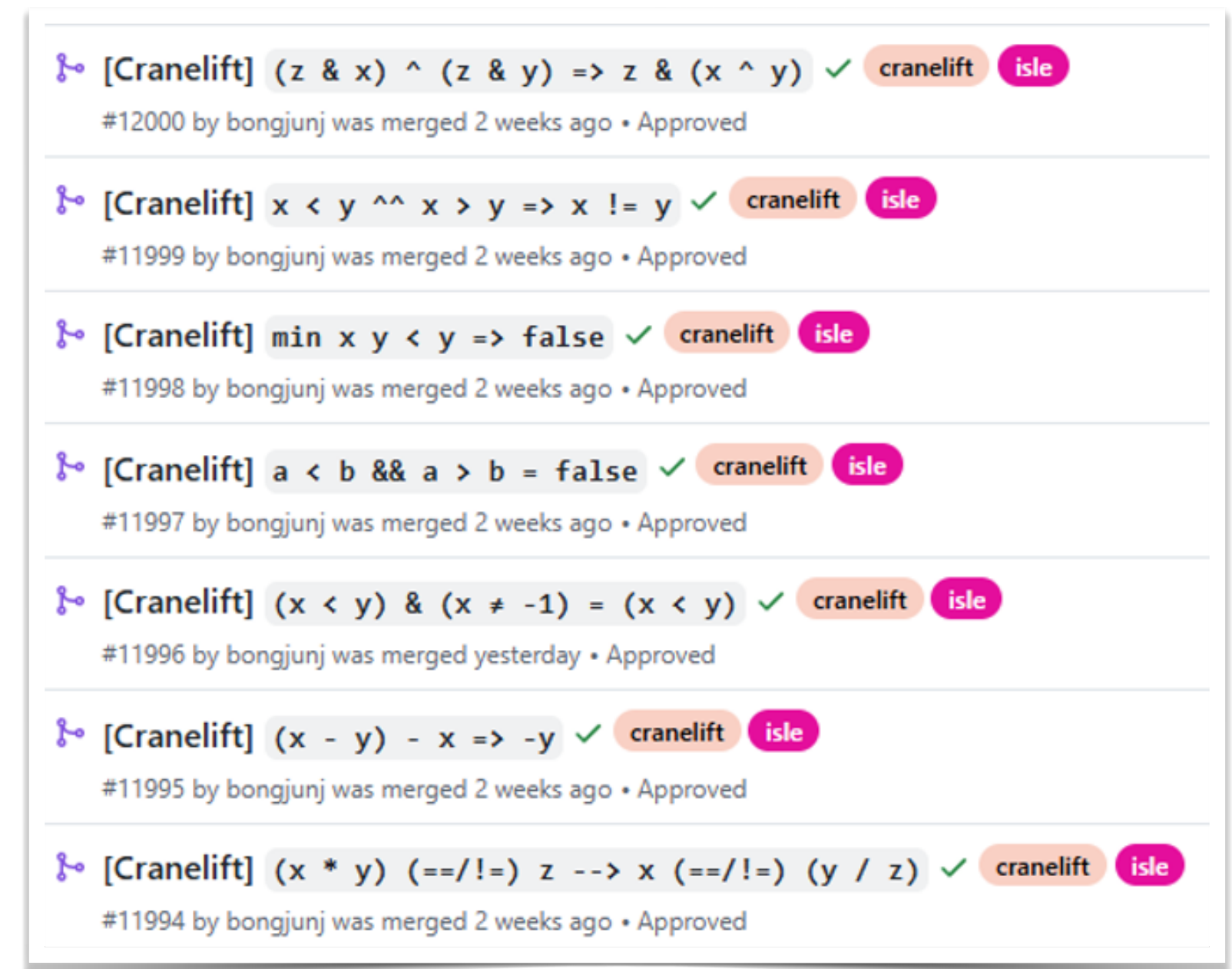
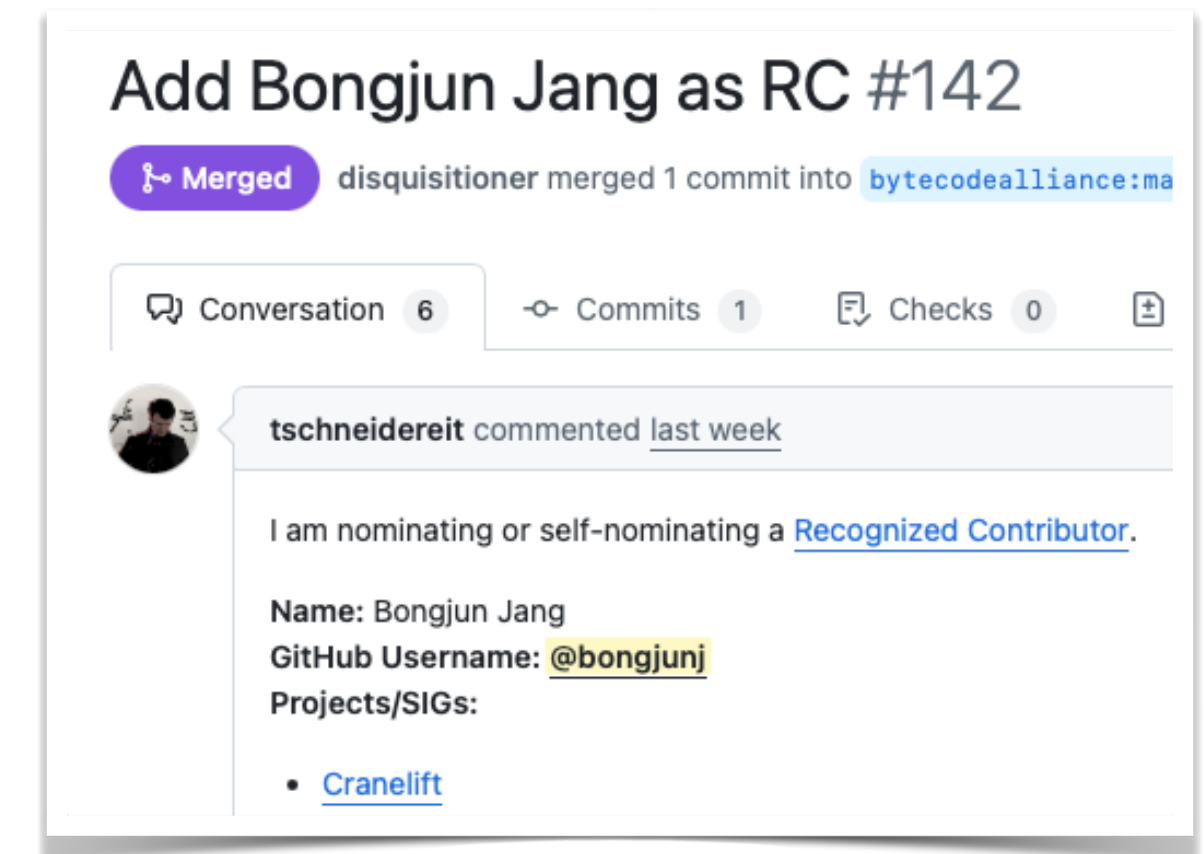
오픈소스 기여: Cranelift에 최적화 규칙 이식

- Cranelift에 없는 규칙을 골라 현재까지 최적화 규칙 97개 추가



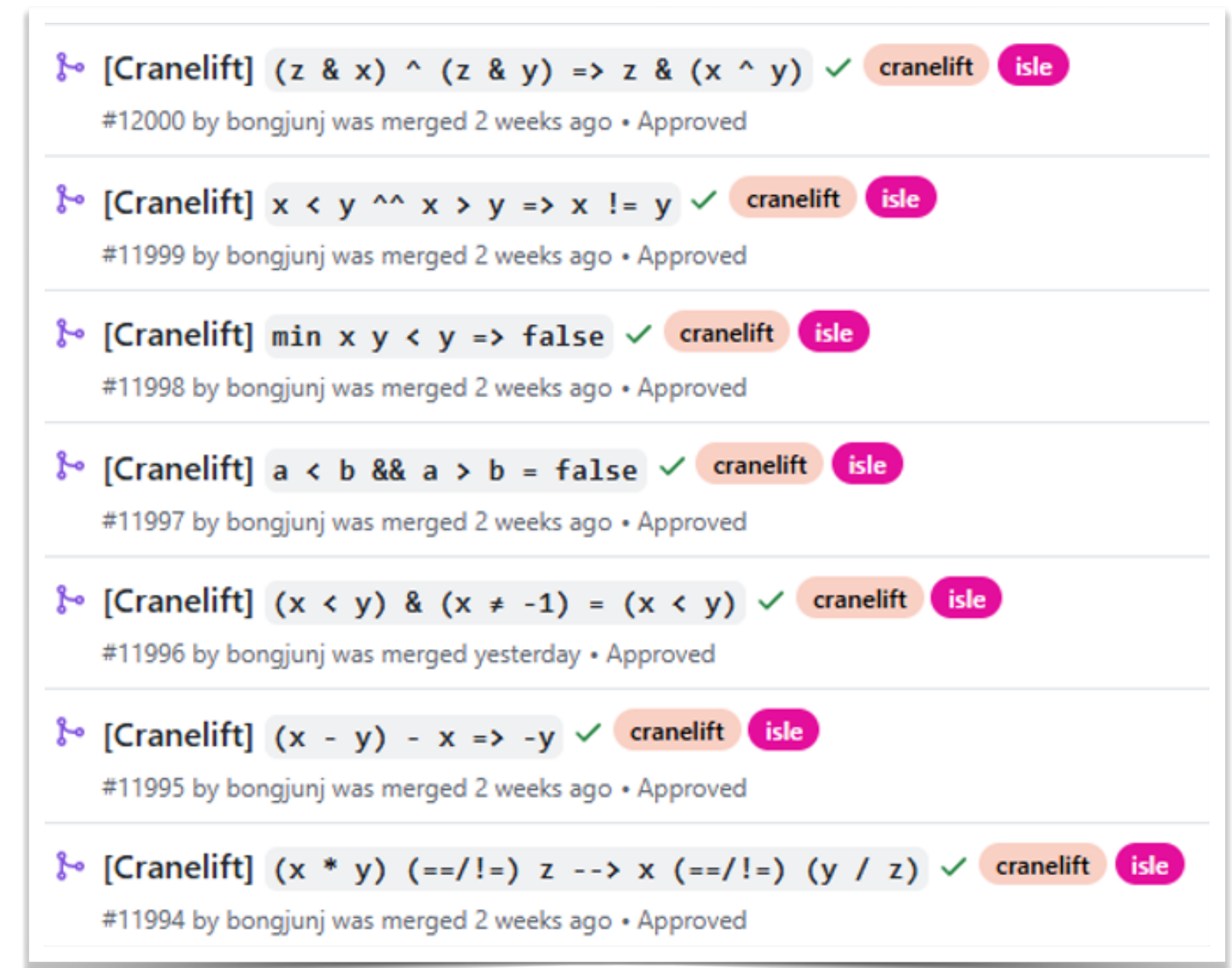
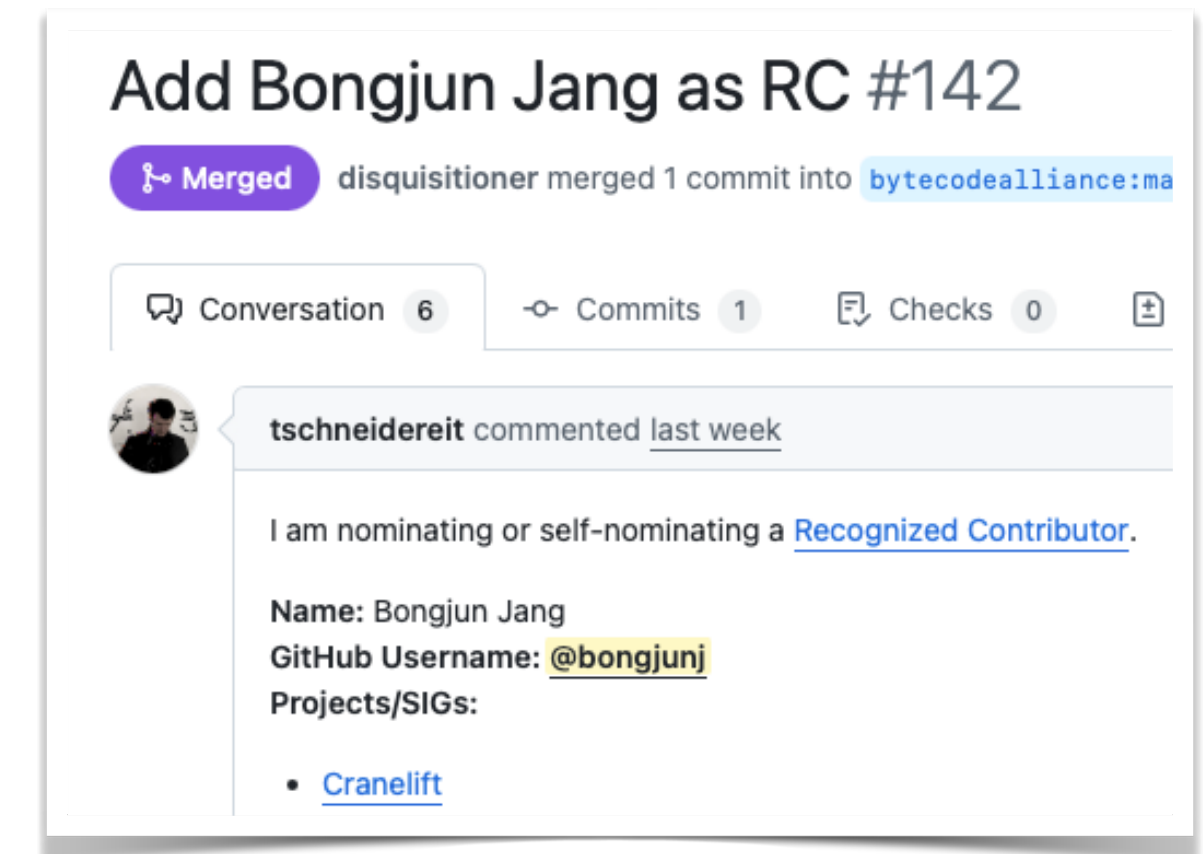
오픈소스 기여: Cranelift에 최적화 규칙 이식

- Cranelift에 없는 규칙을 골라 현재까지 최적화 규칙 97개 추가
- 알아주는 개발자(Recognized Contributor)로 선정



오픈소스 기여: Cranelift에 최적화 규칙 이식

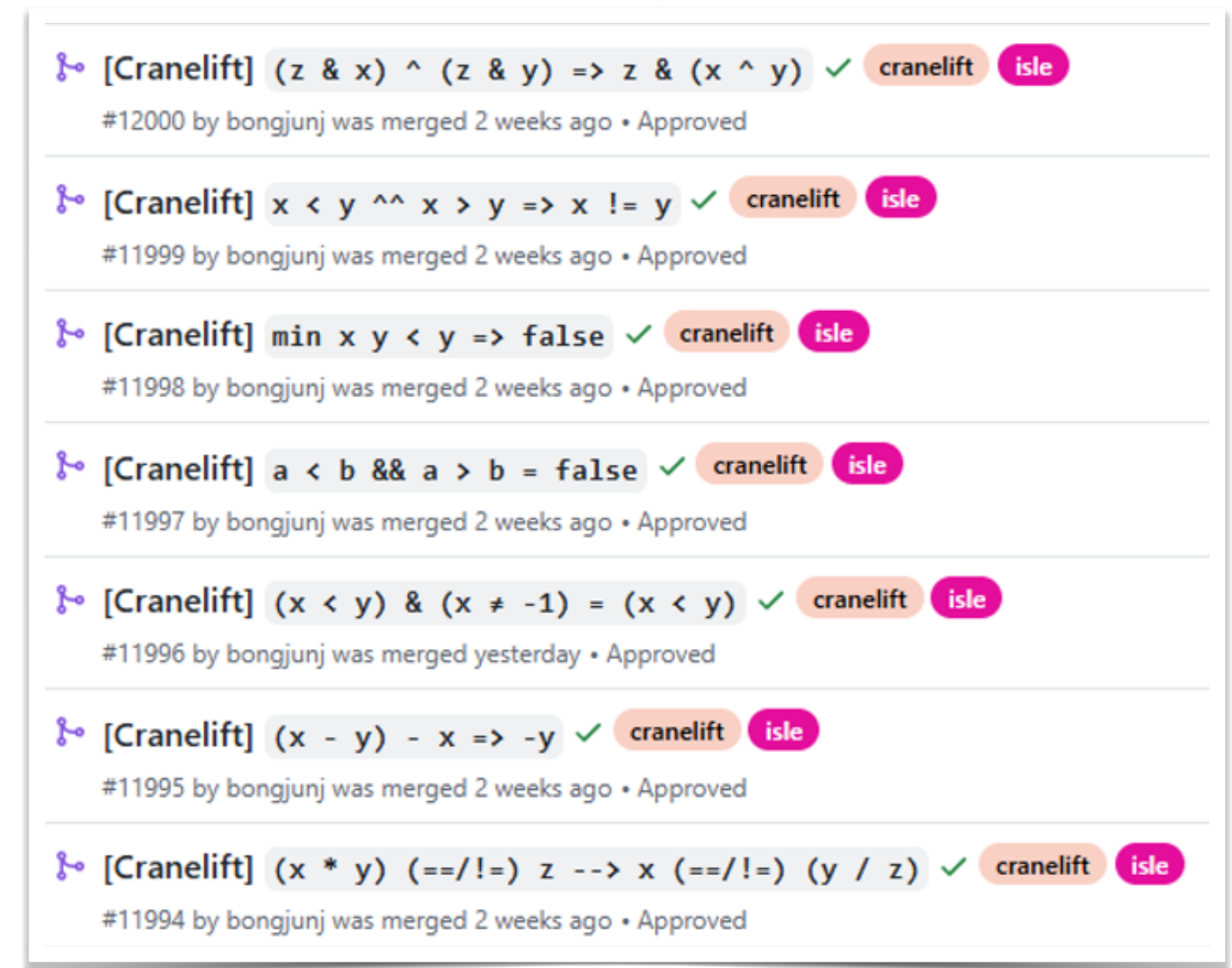
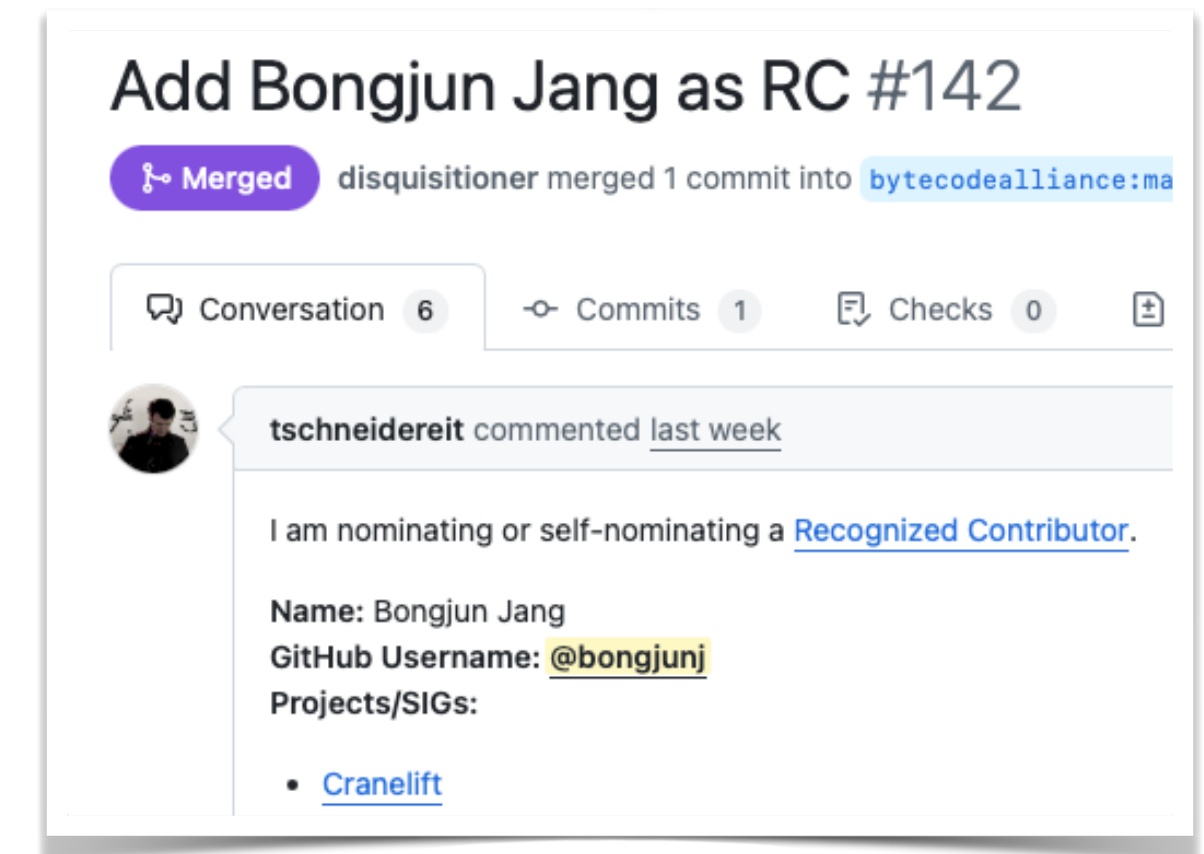
- Cranelift에 없는 규칙을 골라 현재까지 최적화 규칙 97개 추가
- 알아주는 개발자(Recognized Contributor)로 선정
- 아주 단순한 규칙도 구현되지 않은 경우
 - $(X - Y) + Y \Rightarrow X, (X \oplus Y) \oplus Y \Rightarrow X$



오픈소스 기여: Cranelift에 최적화 규칙 이식

- Cranelift에 없는 규칙을 골라 현재까지 최적화 규칙 97개 추가
- 알아주는 개발자(Recognized Contributor)로 선정
- 아주 단순한 규칙도 구현되지 않은 경우
 - $(X - Y) + Y \Rightarrow X, (X \oplus Y) \oplus Y \Rightarrow X$

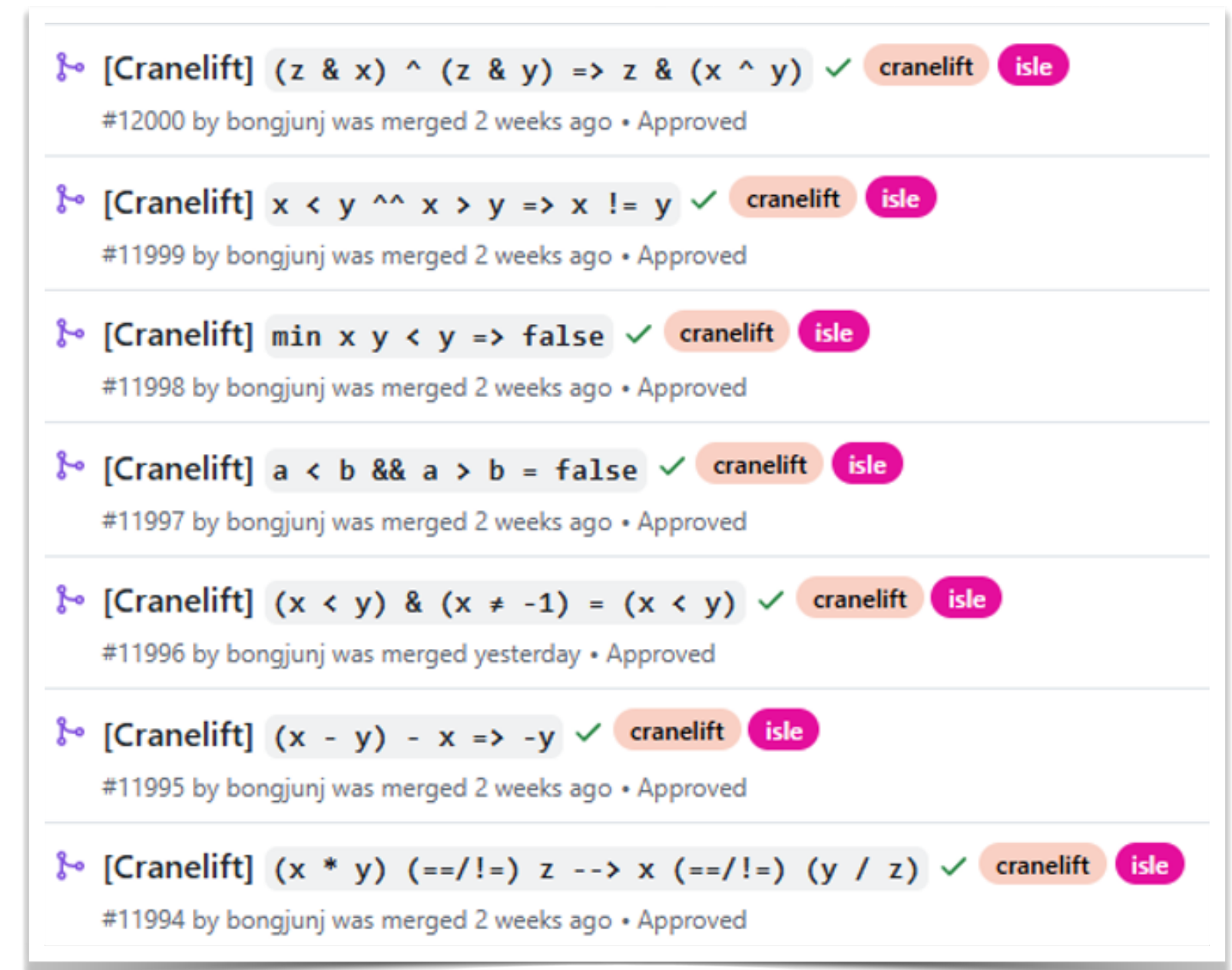
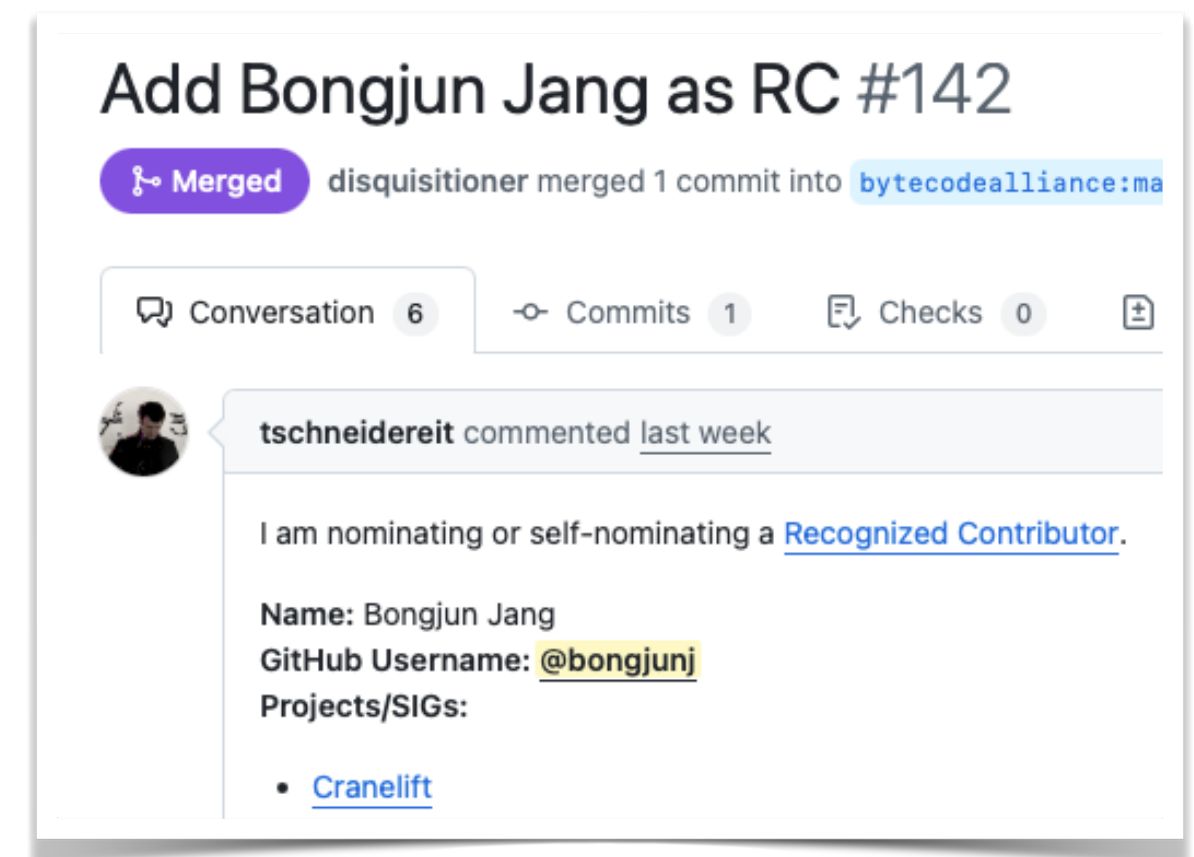
최적화 규칙을 일일이
작성하는 일은 어렵다



오픈소스 기여: Cranelift에 최적화 규칙 이식

- Cranelift에 없는 규칙을 골라 현재까지 최적화 규칙 97개 추가
- 알아주는 개발자(Recognized Contributor)로 선정
- 아주 단순한 규칙도 구현되지 않은 경우
 - $(X - Y) + Y \Rightarrow X, (X \oplus Y) \oplus Y \Rightarrow X$
- 복잡한 규칙도 추가
 - $(X | C) - C \Rightarrow X \& \sim C$
 - $X / (\text{select } P \ 2^C \ 2^D) \Rightarrow X \gg (\text{select } P \ C \ D)$

최적화 규칙을 일일이
작성하는 일은 어렵다



평가: 이식 후 성능 변화

평가: 이식 후 성능 변화

- CPU 사이클 10회 반복 측정 평균

평가: 이식 후 성능 변화

- CPU 사이클 10회 반복 측정 평균
- Cranelift(WASM 컴파일러) 최신버전 기준

평가: 이식 후 성능 변화

- CPU 사이클 10회 반복 측정 평균
- Cranelift(WASM 컴파일러) 최신버전 기준
- 컴파일된 프로그램 실행 시간 및 컴파일에 걸리는 시간

평가: 이식 후 성능 변화

- CPU 사이클 10회 반복 측정 평균
- Cranelift(WASM 컴파일러) 최신버전 기준
- 컴파일된 프로그램 실행 시간 및 컴파일에 걸리는 시간
- 벤치마크 프로그램: 해시 함수, 압축 알고리즘, 파서, 정규식 계산, 자바스크립트 런타임

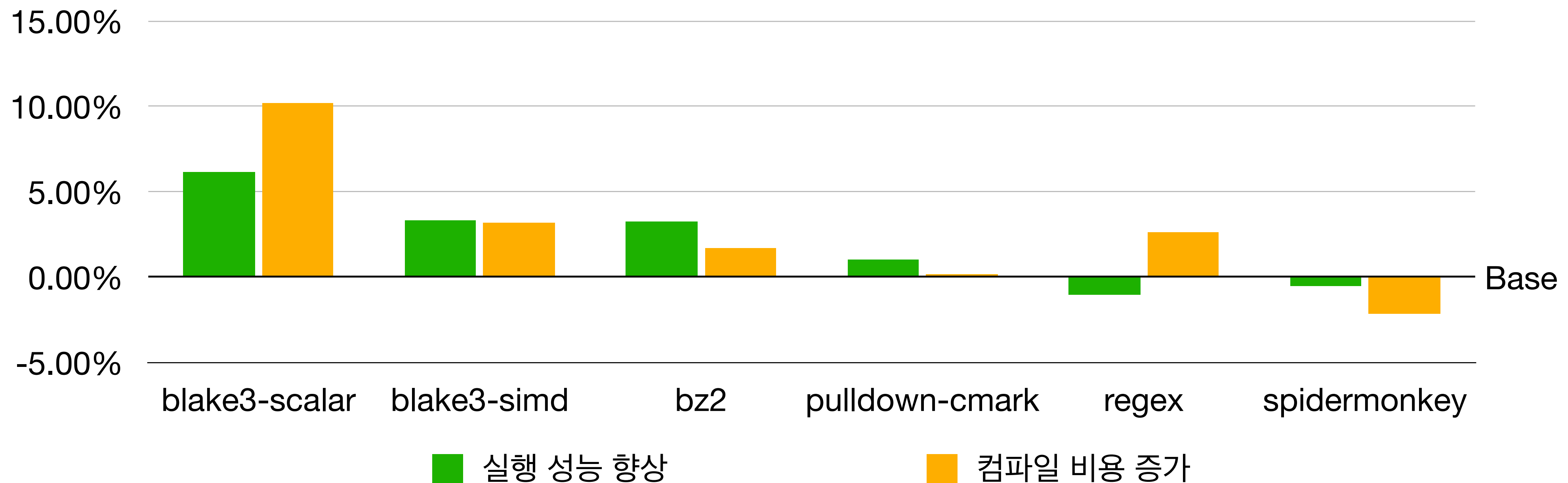
평가: 이식 후 성능 변화

- CPU 사이클 10회 반복 측정 평균
- Cranelift(WASM 컴파일러) 최신버전 기준
- 컴파일된 프로그램 실행 시간 및 컴파일에 걸리는 시간
- 벤치마크 프로그램: 해시 함수, 압축 알고리즘, 파서, 정규식 계산, 자바스크립트 런타임



평가: 이식 후 성능 변화

- CPU 사이클 10회 반복 측정 평균
- Cranelift(WASM 컴파일러) 최신버전 기준
- 컴파일된 프로그램 실행 시간 및 컴파일에 걸리는 시간
- 벤치마크 프로그램: 해시 함수, 압축 알고리즘, 파서, 정규식 계산, 자바스크립트 런타임



요약



컴파일러 최적화를 자동으로 이식하는 방법

장봉준, 허기홍

배경: 컴파일러 최적화 개발

컴파일러 최적화란?

- 프로그래름을 더 빠른 프로그램으로 바꿈: $X \div 2 \Rightarrow X \gg 1$
- 프로그램 성능 향상을 위한 컴파일러의 필수적인 요소
 - LLVM 반복문 최적화: 성능 14% 향상 [PLDI 2025]
 - NVCC 최적화 적용시 (-O0 vs -O3) 성능 약 4배 차이 [TACO 2024]

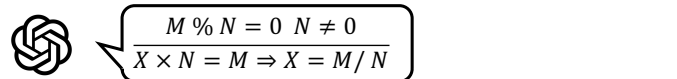
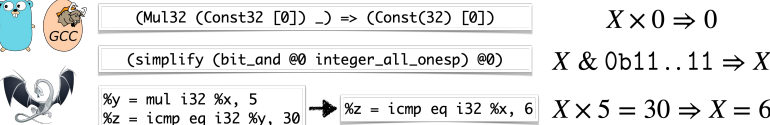
컴파일러 최적화 개발의 문제점

- 신생 언어의 등장으로 새로운 컴파일러가 요구됨
 - WebAssembly 컴파일러: Cranelift
 - AI 컴파일러: CUDA (NVIDIA), Triton (OpenAI)
- 하지만 컴파일러 최적화를 작성하는 것은 어려움
 - 성능 향상의 기회 찾기, 올바른 보장
 - WASM vs LLVM 컴파일러 성능 차이: 17.15% [ISSTA 2023]

스케치 합성: 최적화 아이디어 추출

언어모델을 이용한 최적화 스케치 작성

- 언어모델이 성숙한 컴파일러를 참고해 최적화 스케치 작성
 - LLVM: 최적화기 인출력 예제 참고
 - GCC/Go: DSL로 작성된 재작성 규칙 참고
- 언어모델을 통해 상수간 관계, 전제조건 추론



$$S ::= \frac{P^*}{P \Rightarrow P}$$
$$P ::= P \text{ binop } P \mid \text{unop } P \mid P \text{ cmp } P \mid n \mid X \mid Y \mid Z \mid \dots$$
$$\text{binop} ::= + \mid - \mid \times \mid \% \mid \div \mid \& \mid \mid \mid \oplus \mid \gg \mid \ll \mid \dots$$

문법 검사를 위한 스케치 언어

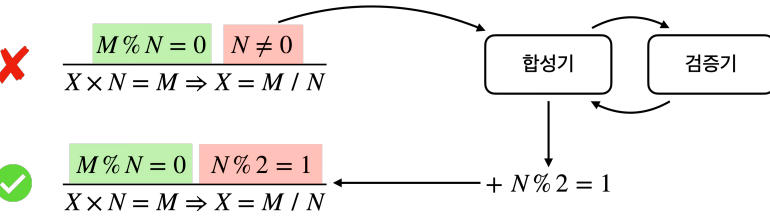
규칙 검증: 올바른 최적화 조건 합성

최적화 전제조건 합성

- 나열식 합성기와 검증기를 결합한 CEGIS (반례 기반 합성)
- 언어모델이 작성한 스케치 중 전제조건 부분을 이용해 탐색 가속

규칙의 동등성 검사

- 좌변과 우변이 모든 입력에 대해 같은 출력을 내는지 검사



검증된 규칙 후처리

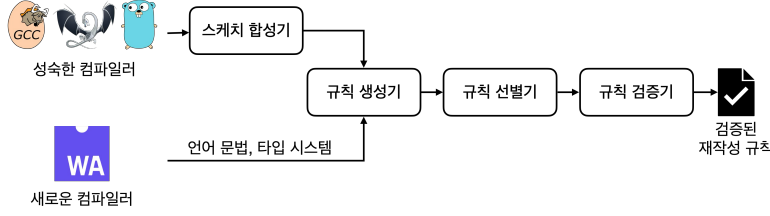
- 교환법칙, 대칭성을 이용한 규칙 추가 탐색
- 최적화 규칙의 좌변에 교환법칙 및 대칭성을 적용
- 더 많은 패턴 매칭 가능 → 최적화 기회 증가
- 변수명 정규화 및 중복 규칙 제거

$(X - Y) + Y \Rightarrow X$ $\text{select } C \ X \ Y \Rightarrow \dots$
 $Y + (X - Y) \Rightarrow X$ $\text{select } \neg C \ Y \ X \Rightarrow \dots$

해결책: 최적화 규칙 자동 이식

컴파일러 개발의 대자부, 최적화를 자동 이식하자!

- 서로 다른 컴파일러라도 최적화 아이디어는 같음
- 성숙한 컴파일러는 새로운 컴파일러에 비해 최적화가 많이 구현됨



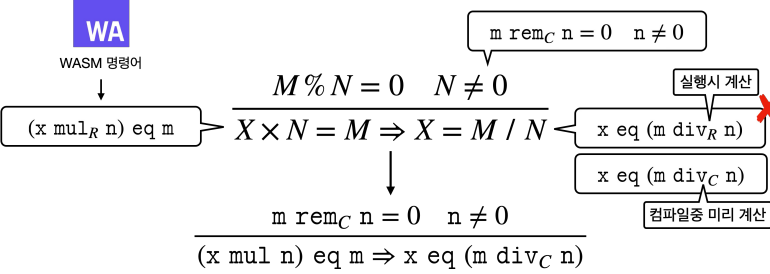
규칙 생성 및 선별: 스케치를 재작성 규칙으로 구체화

최적화 규칙을 실행 가능한 재작성 규칙으로 변환

- 스케치의 요소를 대상 컴파일러의 명령어로 대응
- 계산 시점 분화를 통해 컴파일러 최적화 기회 탐색
- 컴파일 중에 미리 계산 가능하면 실행 성능 향상

실행 성능을 향상시키는 규칙만 고르기

- 비용 모델 정의: 명령어 → 비용
- 우변의 비용이 좌변의 비용보다 작은 경우만 선택



성능 평가 및 Cranelift 기여

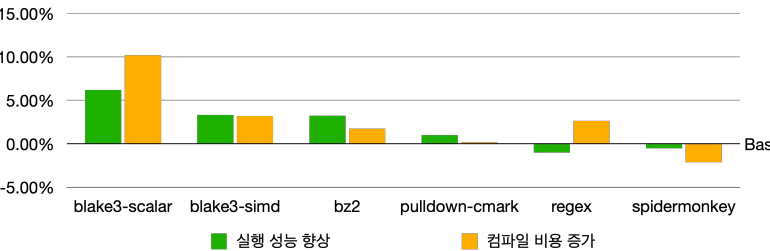
LLVM/GCC/Go의 최적화 규칙을 Cranelift로 이식

- 성숙한 컴파일러에서 최적화 3,050개 수집
- 후처리 및 중복 제거 후 이식 가능한 규칙 6,266개 생성
- 최적화 규칙 977개 Cranelift에 통합 완료
- Bytecode Alliance Recognized Contributor (알아주는 개발자) 선정

컴파일러	참고 대상		규칙 생성기		규칙 선별기		규칙 검증기	
	규칙	스케치	규칙	스케치	규칙	스케치	규칙	스케치
LLVM	1,995	1,814	462,050	1,712	354,652	1,145	28,138	
GCC	482	659	130,716	577	93,981	397	7,838	
Go	573	430	115,968	285	32,794	190	1,427	

Cranelift 성능 영향

- CPU 사이클 10회 반복 측정 평균값
- 컴파일된 프로그램 실행 시간 및 컴파일에 걸리는 시간
- 벤치마크: 해시, 압축 알고리즘, 파서, 정규표현식, 자바스크립트 런타임



요약

- 새로운 컴퓨팅 환경을 위한 새로운 언어 등장



컴파일러 최적화를 자동으로 이식하는 방법

장봉준, 허기홍

KAIST Programming Systems Laboratory

배경: 컴파일러 최적화 개발

컴파일러 최적화란?

- 프로그래름을 더 빠른 프로그램으로 바꿈: $X \div 2 \Rightarrow X \gg 1$
- 프로그램 성능 향상을 위한 컴파일러의 필수적인 요소
 - LLVM 반복문 최적화: 성능 14% 향상 [PLDI 2025]
 - NVCC 최적화 적용시 (-O0 vs -O3) 성능 약 4배 차이 [TACO 2024]

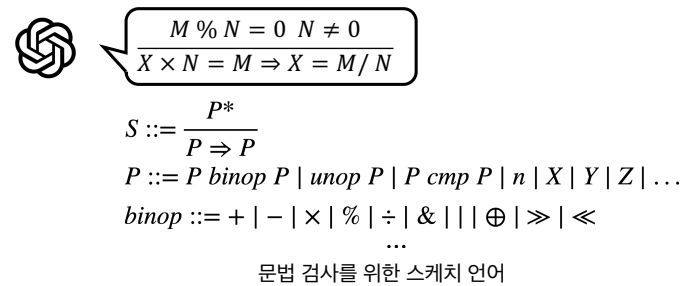
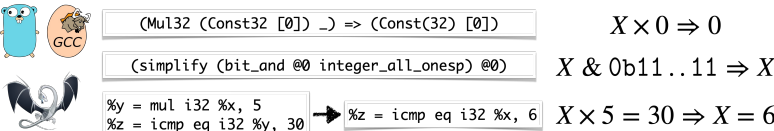
컴파일러 최적화 개발의 문제점

- 신생 언어의 등장으로 새로운 컴파일러가 요구됨
 - WebAssembly 컴파일러: Cranelift
 - AI 컴파일러: CUDA (NVIDIA), Triton (OpenAI)
- 하지만 컴파일러 최적화를 작성하는 것은 어려움
 - 성능 향상의 기회 찾기, 올바른 보장
 - WASM vs LLVM 컴파일러 성능 차이: 17.15% [ISSTA 2023]

스케치 합성: 최적화 아이디어 추출

언어모델을 이용한 최적화 스케치 작성

- 언어모델이 성숙한 컴파일러를 참고해 최적화 스케치 작성
 - LLVM: 최적화기 인출력 예제 참고
 - GCC/Go: DSL로 작성된 재작성 규칙 참고
- 언어모델을 통해 상수간 관계, 전제조건 추론



문법 검사를 위한 스케치 언어

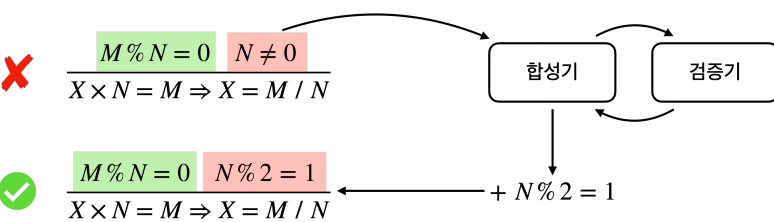
규칙 검증: 올바른 최적화 조건 합성

최적화 전제조건 합성

- 나열식 합성기와 검증기를 결합한 CEGIS (반례 기반 합성)
- 언어모델이 작성한 스케치 중 전제조건 부분을 이용해 탐색 가속

규칙의 동등성 검사

- 좌변과 우변이 모든 입력에 대해 같은 출력을 내는지 검사



검증된 규칙 후처리

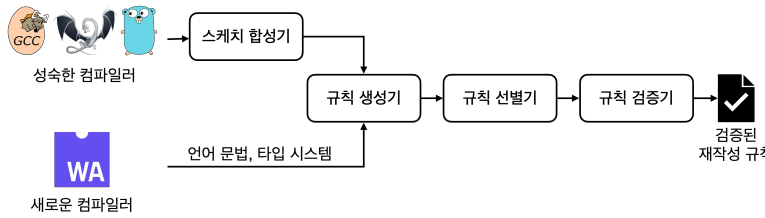
- 교환법칙, 대칭성을 이용한 규칙 추가 탐색
- 최적화 규칙의 좌변에 교환법칙 및 대칭성을 적용
- 더 많은 패턴 매칭 가능 → 최적화 기회 증가
- 변수명 정규화 및 중복 규칙 제거

$(X - Y) + Y \Rightarrow X$ $\text{select } C \ X \ Y \Rightarrow \dots$
 $Y + (X - Y) \Rightarrow X$ $\text{select } \neg C \ Y \ X \Rightarrow \dots$

해결책: 최적화 규칙 자동 이식

컴파일러 개발의 대자부, 최적화를 자동 이식하자!

- 서로 다른 컴파일러라도 최적화 아이디어는 같음
- 성숙한 컴파일러는 새로운 컴파일러에 비해 최적화가 많이 구현됨



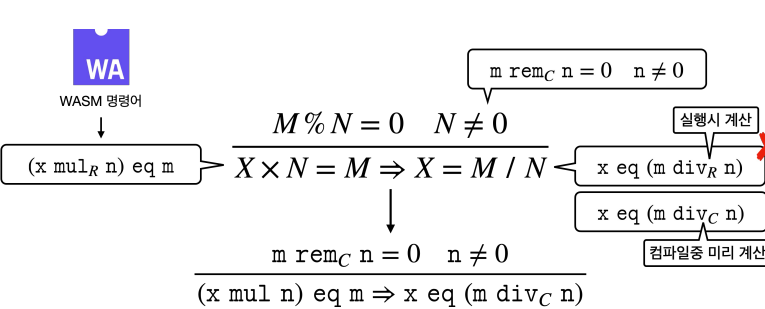
규칙 생성 및 선별: 스케치를 재작성 규칙으로 구체화

최적화 규칙을 실행 가능한 재작성 규칙으로 변환

- 스케치의 요소를 대상 컴파일러의 명령어로 대응
- 계산 시점 분화를 통해 컴파일러 최적화 기회 탐색
- 컴파일 중에 미리 계산 가능하면 실행 성능 향상

실행 성능을 향상시키는 규칙만 고르기

- 비용 모델 정의: 명령어 → 비용
- 우변의 비용이 좌변의 비용보다 작은 경우만 선택



성능 평가 및 Cranelift 기여

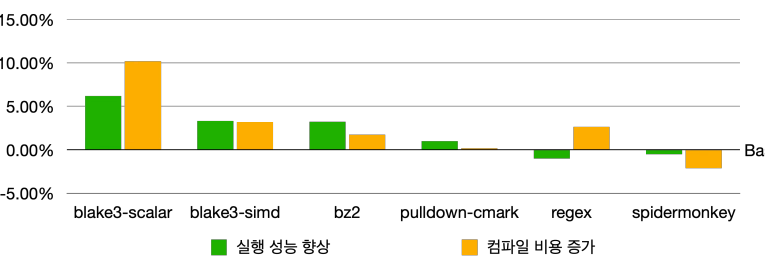
LLVM/GCC/Go의 최적화 규칙을 Cranelift로 이식

- 성숙한 컴파일러에서 최적화 3,050개 수집
- 후처리 및 중복 제거 후 이식 가능한 규칙 6,266개 생성
- 최적화 규칙 977개 Cranelift에 통합 완료
- Bytecode Alliance Recognized Contributor (알아주는 개발자) 선정

참고 대상 컴파일러	규칙 대상		규칙 생성기		규칙 선별기		규칙 검증기	
	규칙	스케치	규칙	스케치	규칙	스케치	규칙	스케치
LLVM	1,995	1,814	462,050	1,712	354,652	1,145	28,138	
GCC	482	659	130,716	577	93,981	397	7,838	
Go	573	430	115,968	285	32,794	190	1,427	

Cranelift 성능 영향

- CPU 사이클 10회 반복 측정 평균값
- 컴파일된 프로그램 실행 시간 및 컴파일에 걸리는 시간
- 벤치마크: 해시, 압축 알고리즘, 파서, 정규표현식, 자바스크립트 런타임



요약

- 새로운 컴퓨팅 환경을 위한 새로운 언어 등장
- 하지만 컴파일러 최적화를 다시 구현하는 일은 어려움



컴파일러 최적화를 자동으로 이식하는 방법

장봉준, 허기훈

KAIST Programming Systems Laboratory

배경: 컴파일러 최적화 개발

컴파일러 최적화란?

- 프로그램을 더 빠른 프로그램으로 바꿈: $X \div 2 \Rightarrow X \gg 1$
- 프로그램 성능 향상을 위한 컴파일러의 필수적인 요소
 - LLVM 반복문 최적화: 성능 14% 향상 [PLDI 2025]
 - NVCC 최적화 적용시 (-O0 vs -O3) 성능 약 4배 차이 [TACO 2024]

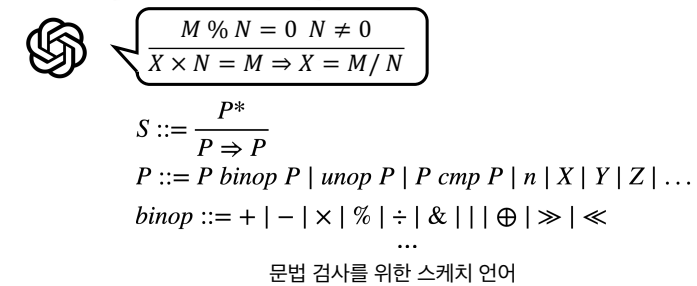
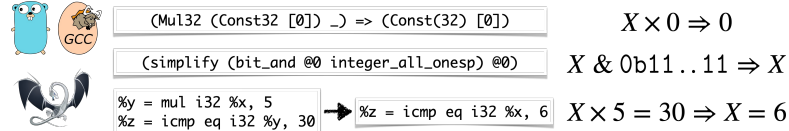
컴파일러 최적화 개발의 문제점

- 신어 언어의 등장으로 새로운 컴파일러가 요구됨
 - WebAssembly 컴파일러: Cranelift
 - AI 컴파일러: CUDA (NVIDIA), Triton (OpenAI)
- 하지만 컴파일러 최적화를 작성하는 것은 어려움
 - 성능 향상의 기회 찾기, 올바른 보장
 - WASM vs LLVM 컴파일러 성능 차이: 17.15% [ISSTA 2023]

스케치 합성: 최적화 아이디어 추출

언어모델을 이용한 최적화 스케치 작성

- 언어모델이 성숙한 컴파일러를 참고해 최적화 스케치 작성
- LLVM: 최적화기 입출력 예제 참고
- GCC/Go: DSL로 작성된 재작성 규칙 참고
- 언어모델을 통해 상수간 관계, 전제조건 추론



문법 검사를 위한 스케치 언어

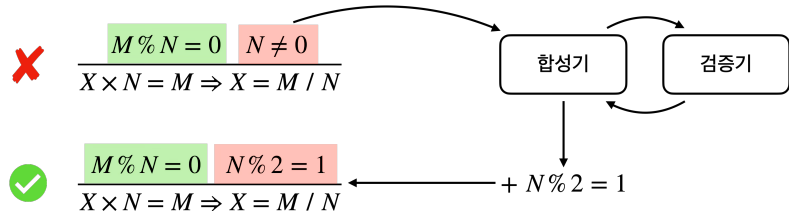
규칙 검증: 올바른 최적화 조건 합성

최적화 전제조건 합성

- 나열식 합성기와 검증기를 결합한 CEGIS (반례 기반 합성)
- 언어모델이 작성한 스케치 중 전제조건 부분을 이용해 탐색 가속

규칙의 동등성 검사

- 좌변과 우변이 모든 입력에 대해 같은 출력을 내는지 검사



검증된 규칙 후처리

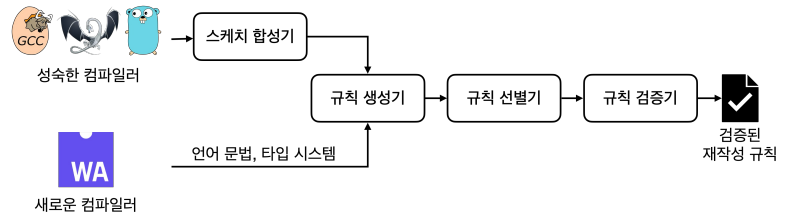
- 교환법칙, 대칭성을 이용한 규칙 추가 탐색
- 최적화 규칙의 좌변에 교환법칙 및 대칭성을 적용
- 더 많은 패턴 매칭 가능 → 최적화 기회 증가
- 변수명 정규화 및 중복 규칙 제거

$(X - Y) + Y \Rightarrow X$ $\text{select } C \ X \ Y \Rightarrow \dots$
 $Y + (X - Y) \Rightarrow X$ $\text{select } \neg C \ Y \ X \Rightarrow \dots$

해결책: 최적화 규칙 자동 이식

컴파일러 개발의 대자류, 최적화를 자동 이식하자!

- 서로 다른 컴파일러라도 최적화 아이디어는 같음
- 성숙한 컴파일러는 새로운 컴파일러에 비해 최적화가 많이 구현됨



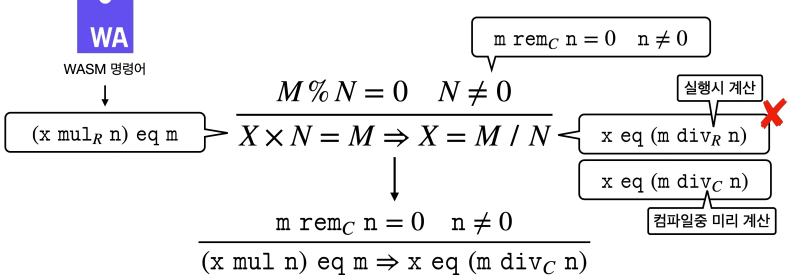
규칙 생성 및 선택: 스케치를 재작성 규칙으로 구체화

최적화 규칙을 실행 가능한 재작성 규칙으로 변환

- 스케치의 요소를 대상 컴파일러의 명령어로 대응
- 계산 시점 분해를 통해 컴파일러 최적화 기회 탐색
- 컴파일 중에 미리 계산 가능하면 실행 성능 향상

실행 성능을 향상시키는 규칙만 고르기

- 비용 모델 정의: 명령어 → 비용
- 우변의 비용이 좌변의 비용보다 작은 경우만 선택



성능 평가 및 Cranelift 기여

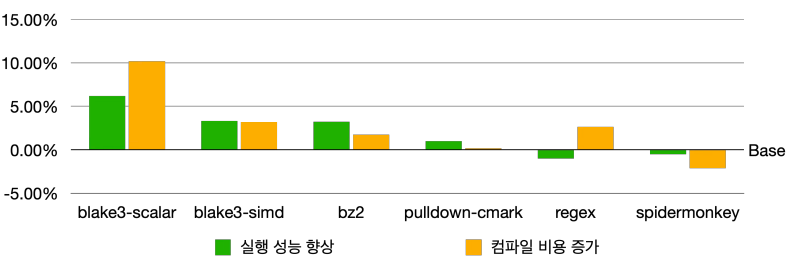
LLVM/GCC/Go의 최적화 규칙을 Cranelift로 이식

- 성숙한 컴파일러에서 최적화 3,050개 수집
- 후처리 및 중복 제거 후 이식 가능한 규칙 6,266개 생성
- 최적화 규칙 977개 Cranelift에 통합 완료
- Bytecode Alliance Recognized Contributor (알아주는 개발자) 선정

컴파일러	참고 대상		규칙 생성기		규칙 선택기		규칙 검증기	
	규칙	스케치	규칙	스케치	규칙	스케치	규칙	스케치
LLVM	1,995	1,814	462,050	1,712	354,652	1,145	28,138	
GCC	482	659	130,716	577	93,981	397	7,838	
Go	573	430	115,968	285	32,794	190	1,427	

Cranelift 성능 영향

- CPU 사이클 10회 반복 측정 평균값
- 컴파일된 프로그램 실행 시간 및 컴파일에 걸리는 시간
- 벤치마크: 해시, 압축 알고리즘, 파서, 정규표현식, 자바스크립트 런타임



요약

- 새로운 컴퓨팅 환경을 위한 새로운 언어 등장
 - 하지만 컴파일러 최적화를 다시 구현하는 일은 어려움
- TransOpt: 성숙한 컴파일러로부터 최적화 규칙 자동 이식



컴파일러 최적화를 자동으로 이식하는 방법

장봉준, 허기훈

KAIST Programming Systems Laboratory

배경: 컴파일러 최적화 개발

컴파일러 최적화란?

- 프로그램을 더 빠른 프로그램으로 바꿈: $X \div 2 \Rightarrow X \gg 1$
- 프로그램 성능 향상을 위한 컴파일러의 필수적인 요소
 - LLVM 반복문 최적화: 성능 14% 향상 [PLDI 2025]
 - NVCC 최적화 적용시 (-O0 vs -O3) 성능 약 4배 차이 [TACO 2024]

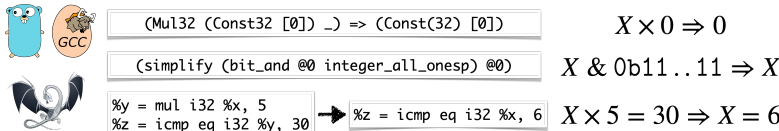
컴파일러 최적화 개발의 문제점

- 신어 언어의 등장으로 새로운 컴파일러가 요구됨
 - WebAssembly 컴파일러: Cranelift
 - AI 컴파일러: CUDA (NVIDIA), Triton (OpenAI)
- 하지만 컴파일러 최적화를 작성하는 것은 어려움
 - 성능 향상의 기회 찾기, 올바른 보장
 - WASM vs LLVM 컴파일러 성능 차이: 17.15% [ISSTA 2023]

스케치 합성: 최적화 아이디어 추출

언어모델을 이용한 최적화 스케치 작성

- 언어모델이 성숙한 컴파일러를 참고해 최적화 스케치 작성
- LLVM: 최적화기 인출력 예제 참고
- GCC/Go: DSL로 작성된 재작성 규칙 참고
- 언어모델을 통해 상수간 관계, 전제조건 추론



$$\begin{cases} M \% N = 0 \\ X \times N = M \Rightarrow X = M / N \end{cases}$$

$$\begin{aligned} S &::= \frac{P^*}{P \Rightarrow P} \\ P &::= P \text{ binop } P \mid \text{unop } P \mid P \text{ cmp } P \mid n \mid X \mid Y \mid Z \mid \dots \\ \text{binop} &::= + \mid - \mid \times \mid \div \mid \% \mid \& \mid \mid \mid \oplus \mid \gg \mid \ll \end{aligned}$$

문법 검사를 위한 스케치 언어

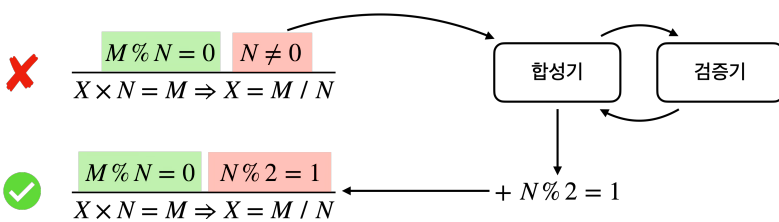
규칙 검증: 올바른 최적화 조건 합성

최적화 전제조건 합성

- 나열식 합성기와 검증기를 결합한 CEGIS (반례 기반 합성)
- 언어모델이 작성한 스케치 중 전제조건 부분을 이용해 탐색 가속

규칙의 동등성 검사

- 좌변과 우변이 모든 입력에 대해 같은 출력을 내는지 검사



검증된 규칙 후처리

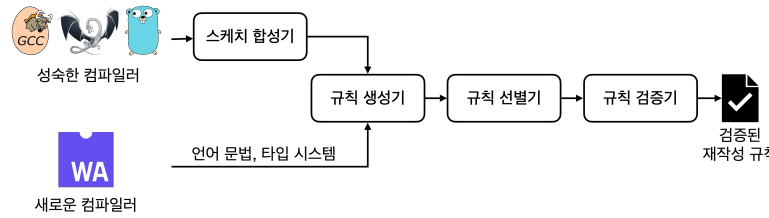
- 교환법칙, 대칭성을 이용한 규칙 추가 탐색
- 최적화 규칙의 좌변에 교환법칙 및 대칭성을 적용
- 더 많은 패턴 매칭 가능 -> 최적화 기회 증가
- 변수명 정규화 및 중복 규칙 제거

$$\begin{aligned} (X - Y) + Y &\Rightarrow X & \text{select } C \ X \ Y &\Rightarrow \dots \\ Y + (X - Y) &\Rightarrow X & \text{select } \neg C \ Y \ X &\Rightarrow \dots \end{aligned}$$

해결책: 최적화 규칙 자동 이식

컴파일러 개발의 대자부, 최적화를 자동 이식하자!

- 서로 다른 컴파일러라도 최적화 아이디어는 같음
- 성숙한 컴파일러는 새로운 컴파일러에 비해 최적화가 많이 구현됨



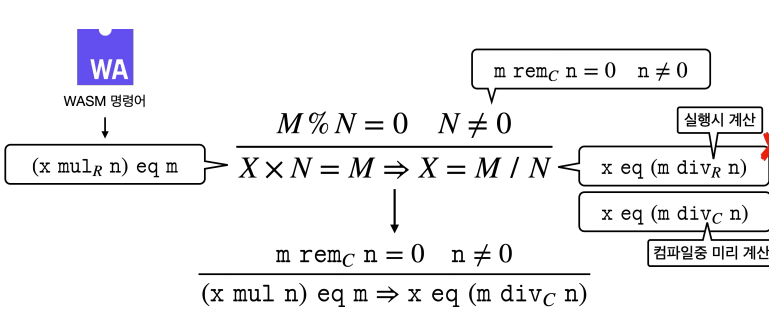
규칙 생성 및 선택: 스케치를 재작성 규칙으로 구체화

최적화 규칙을 실행 가능한 재작성 규칙으로 변환

- 스케치의 요소를 대상 컴파일러의 명령어로 대응
- 계산 시점 분해를 통해 컴파일러 최적화 기회 탐색
- 컴파일 중에 미리 계산 가능하면 실행 성능 향상

실행 성능을 향상시키는 규칙만 고르기

- 비용 모델 정의: 명령어 -> 비용
- 우변의 비용이 좌변의 비용보다 작은 경우만 선택



성능 평가 및 Cranelift 기여

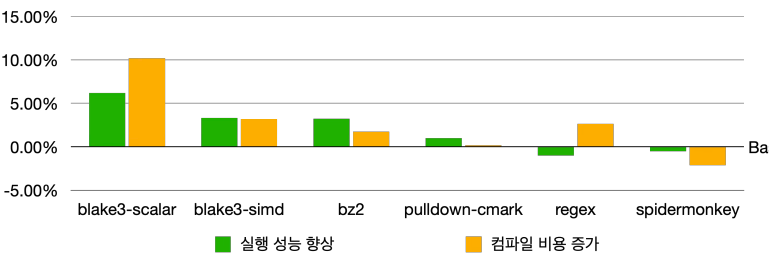
LLVM/GCC/Go의 최적화 규칙을 Cranelift로 이식

- 성숙한 컴파일러에서 최적화 3,050개 수집
- 후처리 및 중복 제거 후 이식 가능한 규칙 6,266개 생성
- 최적화 규칙 977개 Cranelift에 통합 완료
- Bytecode Alliance Recognized Contributor (알아주는 개발자) 선정

컴파일러	참고 대상		규칙 생성기		규칙 선택기		규칙 검증기	
	규칙	스케치	규칙	스케치	규칙	스케치	규칙	스케치
LLVM	1,995	1,814	462,050	1,712	354,652	1,145	28,138	
GCC	482	659	130,716	577	93,981	397	7,838	
Go	573	430	115,968	285	32,794	190	1,427	

Cranelift 성능 영향

- CPU 사이클 10회 반복 측정 평균값
- 컴파일된 프로그램 실행 시간 및 컴파일에 걸리는 시간
- 벤치마크: 해시, 압축 알고리즘, 파서, 정규표현식, 자바스크립트 런타임



요약

- 새로운 컴퓨팅 환경을 위한 새로운 언어 등장
 - 하지만 컴파일러 최적화를 다시 구현하는 일은 어려움
- TransOpt: 성숙한 컴파일러로부터 최적화 규칙 자동 이식
 - 언어모델, 문법 규칙, 검증기를 결합



컴파일러 최적화를 자동으로 이식하는 방법

장봉준, 허기홍

KAIST Programming Systems Laboratory

배경: 컴파일러 최적화 개발

컴파일러 최적화란?

- 프로그램을 더 빠른 프로그램으로 바꿈: $X \div 2 \Rightarrow X \gg 1$
- 프로그램 성능 향상을 위한 컴파일러의 필수적인 요소
 - LLVM 반복문 최적화: 성능 14% 향상 [PLDI 2025]
 - NVCC 최적화 적용시 (-O0 vs -O3) 성능 약 4배 차이 [TACO 2024]

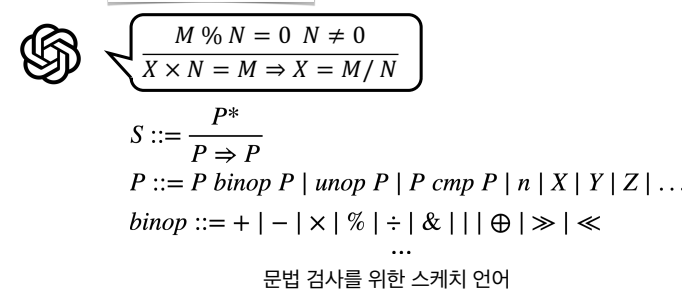
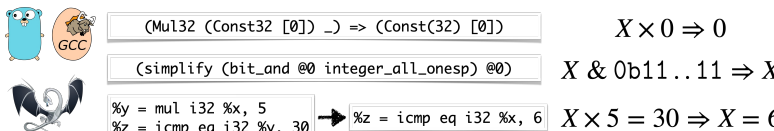
컴파일러 최적화 개발의 문제점

- 신어 언어의 등장으로 새로운 컴파일러가 요구됨
 - WebAssembly 컴파일러: Cranelift
 - AI 컴파일러: CUDA (NVIDIA), Triton (OpenAI)
- 하지만 컴파일러 최적화를 작성하는 것은 어려움
 - 성능 향상의 기회 찾기, 올바른 보장
 - WASM vs LLVM 컴파일러 성능 차이: 17.15% [ISSTA 2023]

스케치 합성: 최적화 아이디어 추출

언어모델을 이용한 최적화 스케치 작성

- 언어모델이 성숙한 컴파일러를 참고해 최적화 스케치 작성
- LLVM: 최적화기 인출력 예제 참고
- GCC/Go: DSL로 작성된 재작성 규칙 참고
- 언어모델을 통해 상수간 관계, 전제조건 추론



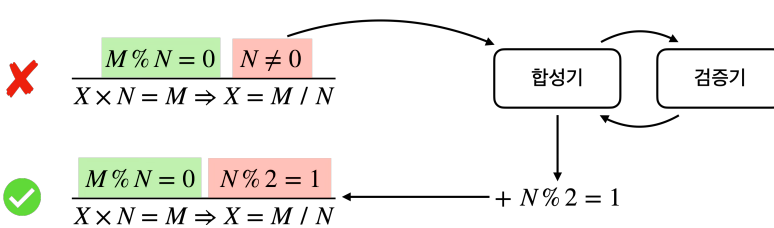
규칙 검증: 올바른 최적화 조건 합성

최적화 전제조건 합성

- 나열식 합성기와 검증기를 결합한 CEGIS (반례 기반 합성)
- 언어모델이 작성한 스케치 중 전제조건 부분을 이용해 탐색 가속

규칙의 동등성 검사

- 좌변과 우변이 모든 입력에 대해 같은 출력을 내는지 검사



검증된 규칙 후처리

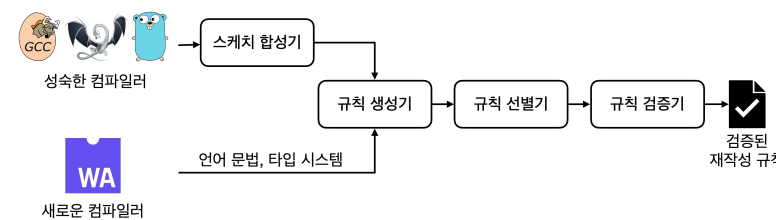
- 교환법칙, 대칭성을 이용한 규칙 추가 탐색
- 최적화 규칙의 좌변에 교환법칙 및 대칭성을 적용
- 더 많은 패턴 매칭 가능 → 최적화 기회 증가
- 변수명 정규화 및 중복 규칙 제거

$(X - Y) + Y \Rightarrow X$ $\text{select } C \ X \ Y \Rightarrow \dots$
 $Y + (X - Y) \Rightarrow X$ $\text{select } \neg C \ Y \ X \Rightarrow \dots$

해결책: 최적화 규칙 자동 이식

컴파일러 개발의 대자류, 최적화를 자동 이식하자!

- 서로 다른 컴파일러라도 최적화 아이디어는 같음
- 성숙한 컴파일러는 새로운 컴파일러에 비해 최적화가 많이 구현됨



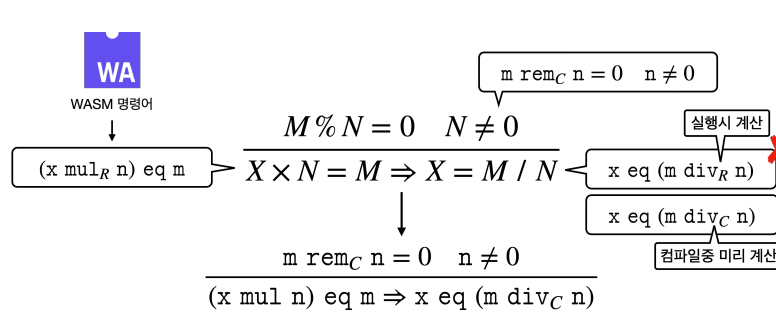
규칙 생성 및 선별: 스케치를 재작성 규칙으로 구체화

최적화 규칙을 실행 가능한 재작성 규칙으로 변환

- 스케치의 요소를 대상 컴파일러의 명령어로 대응
- 계산 시점 분해를 통해 컴파일러 최적화 기회 탐색
- 컴파일 중에 미리 계산 가능하면 실행 성능 향상

실행 성능을 향상시키는 규칙만 고르기

- 비용 모델 정의: 명령어 → 비용
- 우변의 비용이 좌변의 비용보다 작은 경우만 선택



성능 평가 및 Cranelift 기여

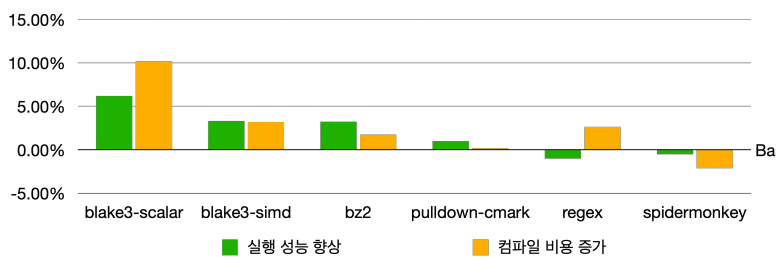
LLVM/GCC/Go의 최적화 규칙을 Cranelift로 이식

- 성숙한 컴파일러에서 최적화 3,050개 수집
- 후처리 및 중복 제거 후 이식 가능한 규칙 6,266개 생성
- 최적화 규칙 977개 Cranelift에 통합 완료
- Bytecode Alliance Recognized Contributor (알아주는 개발자) 선정

참고 대상 컴파일러	규칙 생성기		규칙 선별기		규칙 검증기	
	규칙	스케치	규칙	스케치	규칙	스케치
LLVM	1,995	1,814	462,050	1,712	354,652	1,145
GCC	482	659	130,716	577	93,981	397
Go	573	430	115,968	285	32,794	190

Cranelift 성능 영향

- CPU 사이클 10회 반복 측정 평균값
- 컴파일된 프로그램 실행 시간 및 컴파일에 걸리는 시간
- 벤치마크: 해시, 압축 알고리즘, 파서, 정규표현식, 자바스크립트 런타임



요약

- 새로운 컴퓨팅 환경을 위한 새로운 언어 등장
 - 하지만 컴파일러 최적화를 다시 구현하는 일은 어려움
- TransOpt: 성숙한 컴파일러로부터 최적화 규칙 자동 이식
 - 언어모델, 문법 규칙, 검증기를 결합
 - 이식 대상 컴파일러에 맞는 최적화 규칙 효과적으로 생성



컴파일러 최적화를 자동으로 이식하는 방법

장봉준, 허기홍

KAIST Programming Systems Laboratory

배경: 컴파일러 최적화 개발

컴파일러 최적화란?

- 프로그램을 더 빠른 프로그램으로 바꿈: $X \div 2 \Rightarrow X \gg 1$
- 프로그램 성능 향상을 위한 컴파일러의 필수적인 요소
 - LLVM 반복문 최적화: 성능 14% 향상 [PLDI 2025]
 - NVCC 최적화 적용시 (-O0 vs -O3) 성능 약 4배 차이 [TACO 2024]

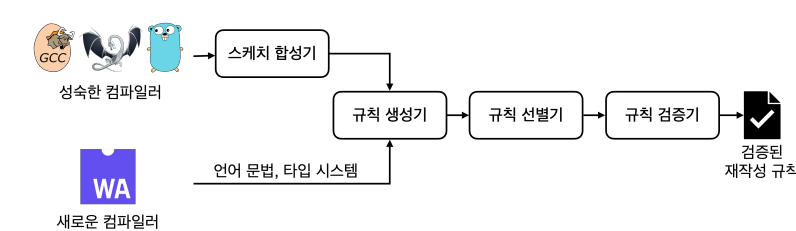
컴파일러 최적화 개발의 문제점

- 신어 언어의 등장으로 새로운 컴파일러가 요구됨
 - WebAssembly 컴파일러: Cranelift
 - AI 컴파일러: CUDA (NVIDIA), Triton (OpenAI)
- 하지만 컴파일러 최적화를 작성하는 것은 어려움
 - 성능 향상의 기회 찾기, 올바른 보장
 - WASM vs LLVM 컴파일러 성능 차이: 17.15% [ISSTA 2023]

해결책: 최적화 규칙 자동 이식

컴파일러 개발의 대자류, 최적화를 자동 이식하자!

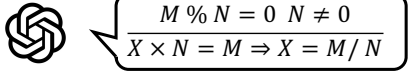
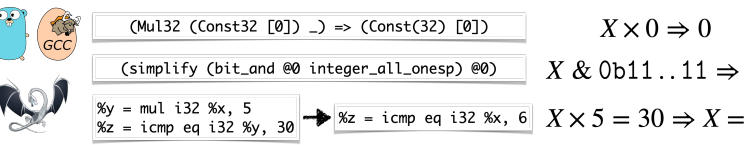
- 서로 다른 컴파일러라도 최적화 아이디어는 같음
- 성숙한 컴파일러는 새로운 컴파일러에 비해 최적화가 많이 구현됨



스케치 합성: 최적화 아이디어 추출

언어모델을 이용한 최적화 스케치 작성

- 언어모델이 성숙한 컴파일러를 참고해 최적화 스케치 작성
- LLVM: 최적화기 인출력 예제 참고
- GCC/Go: DSL로 작성된 재작성 규칙 참고
- 언어모델을 통해 상수간 관계, 전제조건 추론


$$S ::= \frac{P^*}{P \Rightarrow P}$$
$$P ::= P \text{ binop } P \mid unop P \mid P \text{ cmp } P \mid n \mid X \mid Y \mid Z \mid \dots$$
$$binop ::= + \mid - \mid \times \mid \div \mid \& \mid \mid \mid \oplus \mid \gg \mid \ll \mid \dots$$

문법 검사를 위한 스케치 언어

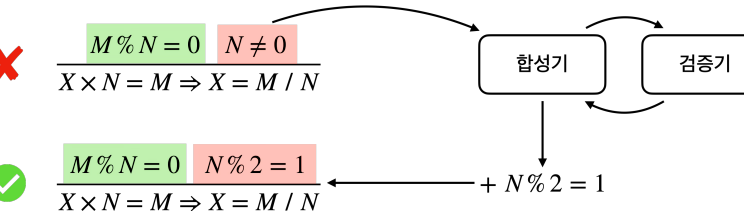
규칙 검증: 올바른 최적화 조건 합성

최적화 전제조건 합성

- 나열식 합성기와 검증기를 결합한 CEGIS (반례 기반 합성)
- 언어모델이 작성한 스케치 중 전제조건 부분을 이용해 탐색 가속

규칙의 동등성 검사

- 좌변과 우변이 모든 입력에 대해 같은 출력을 내는지 검사



검증된 규칙 후처리

- 교환법칙, 대칭성을 이용한 규칙 추가 탐색
- 최적화 규칙의 좌변에 교환법칙 및 대칭성을 적용
- 더 많은 패턴 매칭 가능 → 최적화 기회 증가
- 변수명 정규화 및 중복 규칙 제거

$$(X - Y) + Y \Rightarrow X \quad \text{select } C \ X \ Y \Rightarrow \dots$$
$$Y + (X - Y) \Rightarrow X \quad \text{select } \neg C \ Y \ X \Rightarrow \dots$$

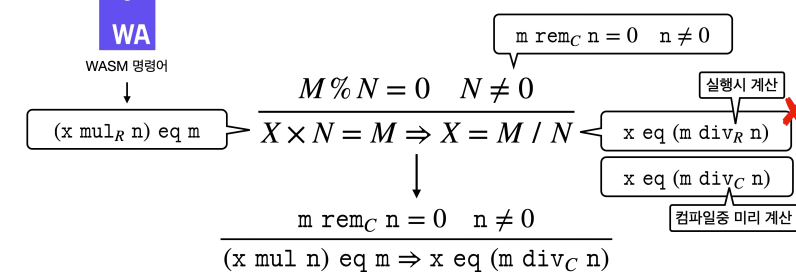
규칙 생성 및 선별: 스케치를 재작성 규칙으로 구체화

최적화 규칙을 실행 가능한 재작성 규칙으로 변환

- 스케치의 요소를 대상 컴파일러의 명령어로 대응
- 계산 시점 분해를 통해 컴파일러 최적화 기회 탐색
- 컴파일 중에 미리 계산 가능하면 실행 성능 향상

실행 성능을 향상시키는 규칙만 고르기

- 비용 모델 정의: 명령어 → 비용
- 우변의 비용이 좌변의 비용보다 작은 경우만 선택



성능 평가 및 Cranelift 기여

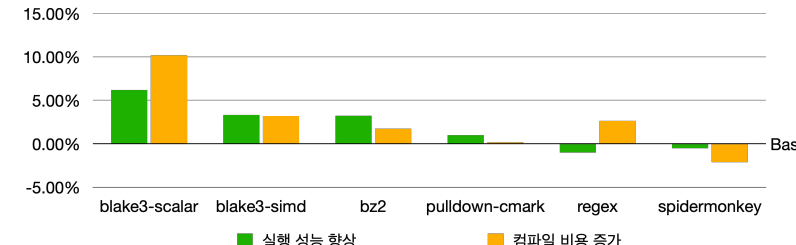
LLVM/GCC/Go의 최적화 규칙을 Cranelift로 이식

- 성숙한 컴파일러에서 최적화 3,050개 수집
- 후처리 및 중복 제거 후 이식 가능한 규칙 6,266개 생성
- 최적화 규칙 977개 Cranelift에 통합 완료
- Bytecode Alliance Recognized Contributor (알아주는 개발자) 선정

컴파일러	참고 대상		규칙 생성기		규칙 선별기		규칙 검증기	
	규칙	스케치	규칙	스케치	규칙	스케치	규칙	
LLVM	1,995	1,814	462,050	1,712	354,652	1,145	28,138	
GCC	482	659	130,716	577	93,981	397	7,838	
Go	573	430	115,968	285	32,794	190	1,427	

Cranelift 성능 영향

- CPU 사이클 10회 반복 측정 평균값
- 컴파일된 프로그램 실행 시간 및 컴파일에 걸리는 시간
- 벤치마크: 해시, 압축 알고리즘, 파서, 정규표현식, 자바스크립트 런타임



요약

- 새로운 컴퓨팅 환경을 위한 새로운 언어 등장
 - 하지만 컴파일러 최적화를 다시 구현하는 일은 어려움
- TransOpt: 성숙한 컴파일러로부터 최적화 규칙 자동 이식
 - 언어모델, 문법 규칙, 검증기를 결합
 - 이식 대상 컴파일러에 맞는 최적화 규칙 효과적으로 생성
- Cranelift에 적용 가능한 재작성 규칙 6,266개 생성



컴파일러 최적화를 자동으로 이식하는 방법

장봉준, 허기훈

KAIST Programming Systems Laboratory

배경: 컴파일러 최적화 개발

컴파일러 최적화란?

- 프로그램을 더 빠른 프로그램으로 바꿈: $X \div 2 \Rightarrow X \gg 1$
- 프로그램 성능 향상을 위한 컴파일러의 필수적인 요소
 - LLVM 반복문 최적화: 성능 14% 향상 [PLDI 2025]
 - NVCC 최적화 적용시 (-O0 vs -O3) 성능 약 4배 차이 [TACO 2024]

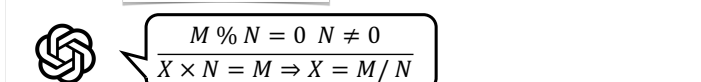
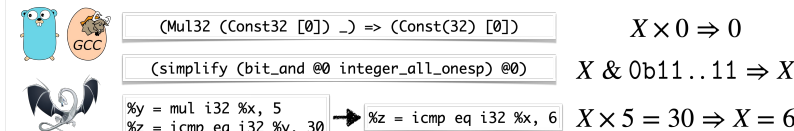
컴파일러 최적화 개발의 문제점

- 신어 언어의 등장으로 새로운 컴파일러가 요구됨
 - WebAssembly 컴파일러: Cranelift
 - AI 컴파일러: CUDA (NVIDIA), Triton (OpenAI)
- 하지만 컴파일러 최적화를 작성하는 것은 어려움
 - 성능 향상의 기회 찾기, 올바른 보장
 - WASM vs LLVM 컴파일러 성능 차이: 17.15% [ISSTA 2023]

스케치 합성: 최적화 아이디어 추출

언어모델을 이용한 최적화 스케치 작성

- 언어모델이 성숙한 컴파일러를 참고해 최적화 스케치 작성
- LLVM: 최적화기 인출력 예제 참고
- GCC/Go: DSL로 작성된 재작성 규칙 참고
- 언어모델을 통해 상수간 관계, 전제조건 추론



문법 검사를 위한 스케치 언어

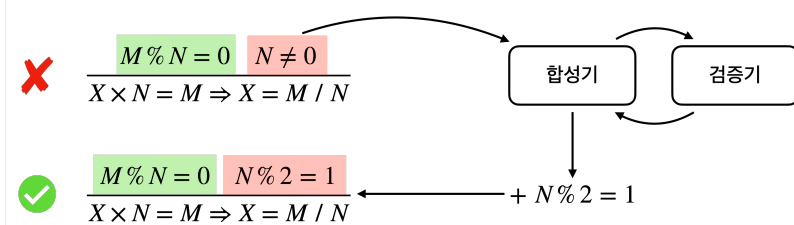
규칙 검증: 올바른 최적화 조건 합성

최적화 전제조건 합성

- 나열식 합성기와 검증기를 결합한 CEGIS (반례 기반 합성)
- 언어모델이 작성한 스케치 중 전제조건 부분을 이용해 탐색 가속

규칙의 동등성 검사

- 좌변과 우변이 모든 입력에 대해 같은 출력을 내는지 검사



검증된 규칙 후처리

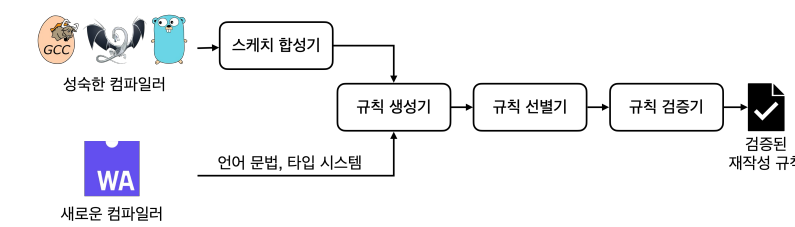
- 교환법칙, 대칭성을 이용한 규칙 추가 탐색
- 최적화 규칙의 좌변에 교환법칙 및 대칭성을 적용
- 더 많은 패턴 매칭 가능 → 최적화 기회 증가
- 변수명 정규화 및 중복 규칙 제거

$(X - Y) + Y \Rightarrow X$ $\text{select } C \ X \ Y \Rightarrow \dots$
 $Y + (X - Y) \Rightarrow X$ $\text{select } \neg C \ Y \ X \Rightarrow \dots$

해결책: 최적화 규칙 자동 이식

컴파일러 개발의 대자류, 최적화를 자동 이식하자!

- 서로 다른 컴파일러라도 최적화 아이디어는 같음
- 성숙한 컴파일러는 새로운 컴파일러에 비해 최적화가 많이 구현됨



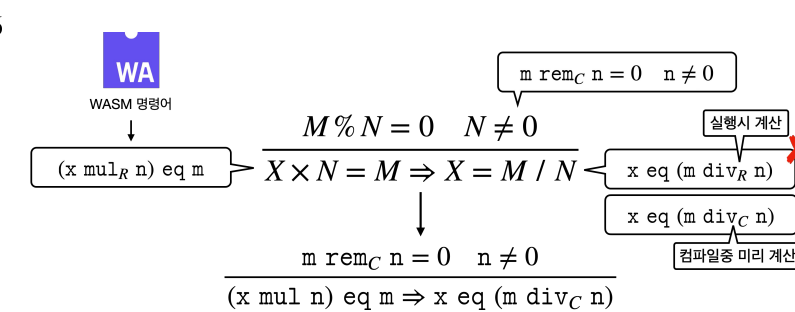
규칙 생성 및 선별: 스케치를 재작성 규칙으로 구체화

최적화 규칙을 실행 가능한 재작성 규칙으로 변환

- 스케치의 요소를 대상 컴파일러의 명령어로 대응
- 계산 시점 분해를 통해 컴파일러 최적화 기회 탐색
- 컴파일 중에 미리 계산 가능하면 실행 성능 향상

실행 성능을 향상시키는 규칙만 고르기

- 비용 모델 정의: 명령어 → 비용
- 우변의 비용이 좌변의 비용보다 작은 경우만 선택



성능 평가 및 Cranelift 기여

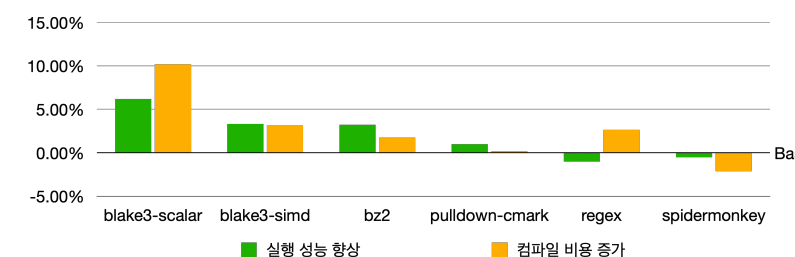
LLVM/GCC/Go의 최적화 규칙을 Cranelift로 이식

- 성숙한 컴파일러에서 최적화 3,050개 수집
- 후처리 및 중복 제거 후 이식 가능한 규칙 6,266개 생성
- 최적화 규칙 977개 Cranelift에 통합 완료
- Bytecode Alliance Recognized Contributor (알아주는 개발자) 선정

참고 대상 컴파일러	규칙	규칙 생성기 스케치	규칙 선별기 스케치	규칙 검증기 스케치
LLVM	1,995	1,814	462,050	1,712
GCC	482	659	130,716	577
Go	573	430	115,968	285

Cranelift 성능 영향

- CPU 사이클 10회 반복 측정 평균값
- 컴파일된 프로그램 실행 시간 및 컴파일에 걸리는 시간
- 벤치마크: 해시, 압축 알고리즘, 파서, 정규표현식, 자바스크립트 런타임



요약

- 새로운 컴퓨팅 환경을 위한 새로운 언어 등장
 - 하지만 컴파일러 최적화를 다시 구현하는 일은 어려움
- TransOpt: 성숙한 컴파일러로부터 최적화 규칙 자동 이식
 - 언어모델, 문법 규칙, 검증기를 결합
 - 이식 대상 컴파일러에 맞는 최적화 규칙 효과적으로 생성
- Cranelift에 적용 가능한 재작성 규칙 6,266개 생성
 - 새로운 최적화 규칙 97개 이식, 성능 최대 6% 향상



컴파일러 최적화를 자동으로 이식하는 방법

장봉준, 허기훈

KAIST Programming Systems Laboratory

배경: 컴파일러 최적화 개발

컴파일러 최적화란?

- 프로그램을 더 빠른 프로그램으로 바꿈: $X \div 2 \Rightarrow X \gg 1$
- 프로그램 성능 향상을 위한 컴파일러의 필수적인 요소
 - LLVM 반복문 최적화: 성능 14% 향상 [PLDI 2025]
 - NVCC 최적화 적용시 (-O0 vs -O3) 성능 약 4배 차이 [TACO 2024]

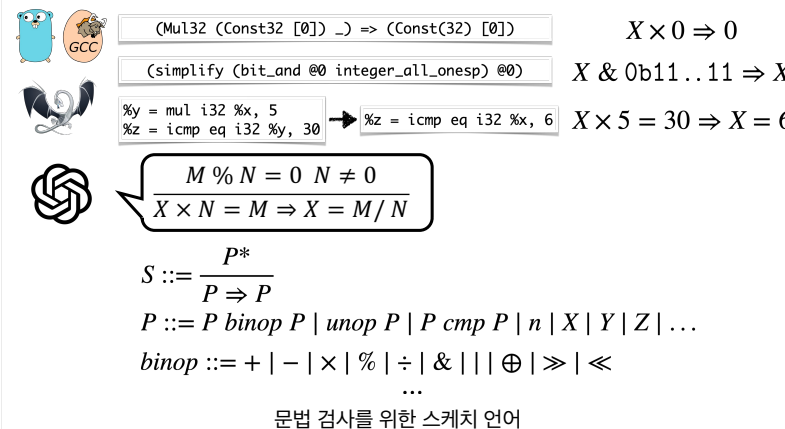
컴파일러 최적화 개발의 문제점

- 신생 언어의 등장으로 새로운 컴파일러가 요구됨
 - WebAssembly 컴파일러: Cranelift
 - AI 컴파일러: CUDA (NVIDIA), Triton (OpenAI)
- 하지만 컴파일러 최적화를 작성하는 것은 어려움
 - 성능 향상의 기회 찾기, 올바른 보장
 - WASM vs LLVM 컴파일러 성능 차이: 17.15% [ISSTA 2023]

스케치 합성: 최적화 아이디어 추출

언어모델을 이용한 최적화 스케치 작성

- 언어모델이 성숙한 컴파일러를 참고해 최적화 스케치 작성
- LLVM: 최적화기 인출력 예제 참고
- GCC/Go: DSL로 작성된 재작성 규칙 참고
- 언어모델을 통해 상수간 관계, 전제조건 추론



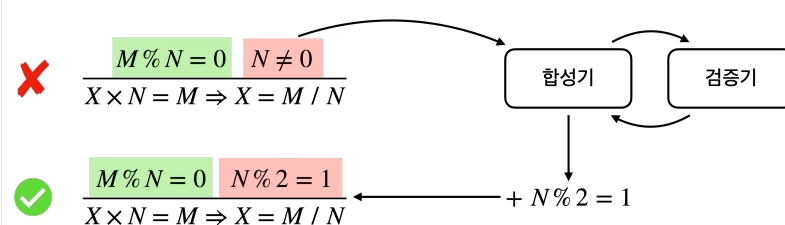
규칙 검증: 올바른 최적화 조건 합성

최적화 전제조건 합성

- 나열식 합성기와 검증기를 결합한 CEGIS (반례 기반 합성)
- 언어모델이 작성한 스케치 중 전제조건 부분을 이용해 탐색 가속

규칙의 동등성 검사

- 좌변과 우변이 모든 입력에 대해 같은 출력을 내는지 검사



검증된 규칙 후처리

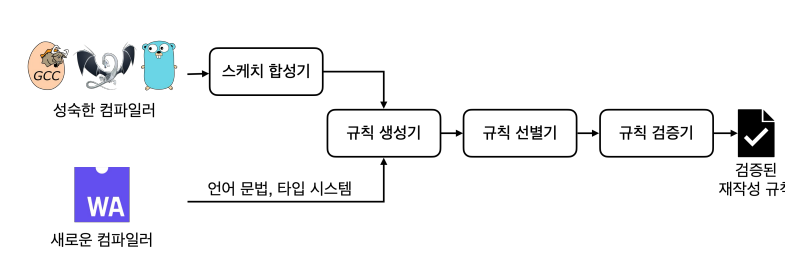
- 교환법칙, 대칭성을 이용한 규칙 추가 탐색
- 최적화 규칙의 좌변에 교환법칙 및 대칭성을 적용
- 더 많은 패턴 매칭 가능 → 최적화 기회 증가
- 변수명 정규화 및 중복 규칙 제거

$(X - Y) + Y \Rightarrow X$ $\text{select } C \ X \ Y \Rightarrow \dots$
 $Y + (X - Y) \Rightarrow X$ $\text{select } \neg C \ Y \ X \Rightarrow \dots$

해결책: 최적화 규칙 자동 이식

컴파일러 개발의 대자부, 최적화를 자동 이식하자!

- 서로 다른 컴파일러라도 최적화 아이디어는 같음
- 성숙한 컴파일러는 새로운 컴파일러에 비해 최적화가 많이 구현됨



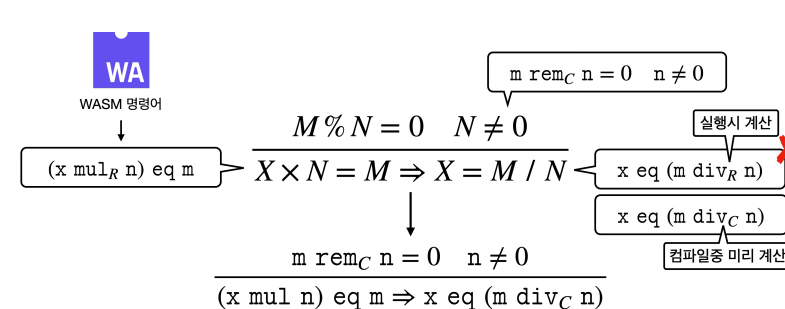
규칙 생성 및 선택: 스케치를 재작성 규칙으로 구체화

최적화 규칙을 실행 가능한 재작성 규칙으로 변환

- 스케치의 요소를 대상 컴파일러의 명령어로 대응
- 계산 시점 분해를 통해 컴파일러 최적화 기회 탐색
- 컴파일 중에 미리 계산 가능하면 실행 성능 향상

실행 성능을 향상시키는 규칙만 고르기

- 비용 모델 정의: 명령어 → 비용
- 우변의 비용이 좌변의 비용보다 작은 경우만 선택



성능 평가 및 Cranelift 기여

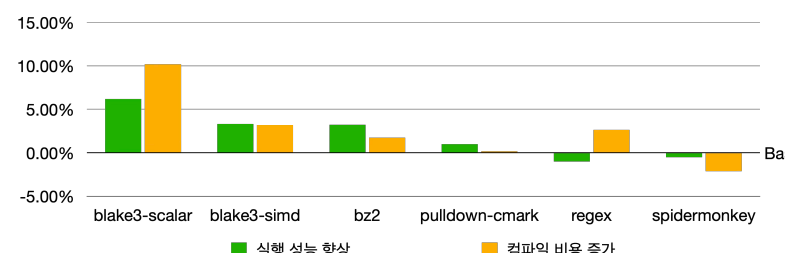
LLVM/GCC/Go의 최적화 규칙을 Cranelift로 이식

- 성숙한 컴파일러에서 최적화 3,050개 수집
- 후처리 및 중복 제거 후 이식 가능한 규칙 6,266개 생성
- 최적화 규칙 97개 Cranelift에 통합 완료
- Bytecode Alliance Recognized Contributor (알아주는 개발자) 선정

참고 대상 컴파일러	규칙	규칙 생성기 스케치	규칙 선택기 스케치	규칙 검증기 스케치
LLVM	1,995	1,814	462,050	1,712
GCC	482	659	130,716	577
Go	573	430	115,968	285

Cranelift 성능 영향

- CPU 사이클 10회 반복 측정 평균값
- 컴파일된 프로그램 실행 시간 및 컴파일에 걸리는 시간
- 벤치마크: 해시, 압축 알고리즘, 파서, 정규표현식, 자바스크립트 런타임



Appendix: N0이 홀수여야 하는 이유

- 4비트 정수 연산 가정
- $M = 4, N = 4$
- 좌변: $X \times 4 = 4$
 - $X = 1$ or $X = 5 \pmod{2^4}$
 - $5 \times 4 = 20 = 4 \pmod{16}$
- 우변: $X = 1$
 - $X = 1$

Appendix: N0이 홀수여야 하는 이유

- 4비트 정수 연산 가정
- $M = 4, N = 4$
- 좌변: $X \times 4 = 4$
 - $X = 1$ or $X = 5 \pmod{2^4}$
 - $5 \times 4 = 20 = 4 \pmod{16}$
- 우변: $X = 1$
 - $X = 1$

$$\frac{M \% N = 0 \quad N \neq 0}{X \times N = M \Rightarrow X = M / N}$$