

싹수 분석을 활용하여 비트 벡터 프로그램 양방향 합성 잘 하기

윤용호, 2022. 07. 22.

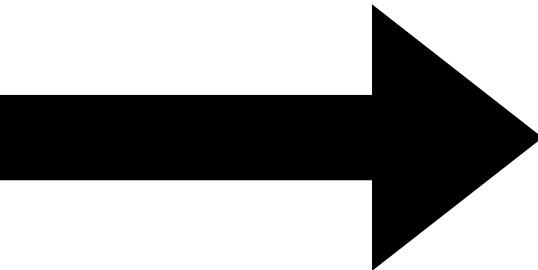
배경 지식

싹수분석을 활용하여
비트 벡터 **프로그램**
양방향 **합성** 잘 하기

프로그램 합성

조건을 만족하는 프로그램을 자동으로 만들기

$$\begin{aligned}f(1) &= 3, \\f(2) &= 6, \\f(x) &= ?\end{aligned}$$

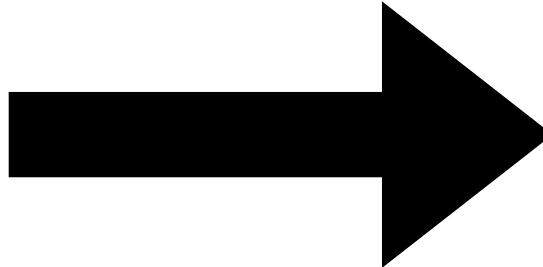


$$f(x) = ?$$

프로그램 합성

조건을 만족하는 프로그램을 자동으로 만들기

$$\begin{aligned}f(1) &= 3, \\f(2) &= 6, \\f(x) &=?\end{aligned}$$

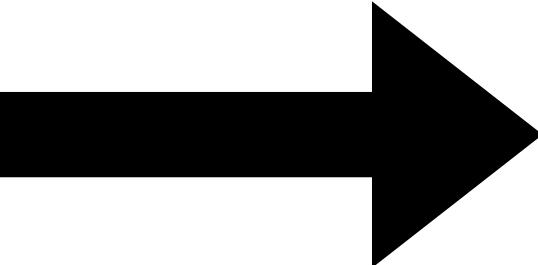


$$f(x) = 3 * x$$

프로그램 합성

조건을 만족하는 프로그램을 자동으로 만들기

$$\begin{aligned}f(1) &= 3, \\f(2) &= 6, \\f(x) &=?\end{aligned}$$

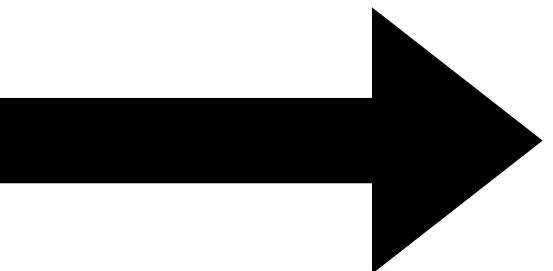


$$\begin{aligned}f(x) &= 3 * x ? \\f(x) &= x * x + 2 ?\end{aligned}$$

프로그램 합성

조건을 만족하는 프로그램을 자동으로 만들기

$$\begin{aligned}f(1) &= 3, \\f(2) &= 6, \\f(3) &= 9, \\f(x) &=?\end{aligned}$$

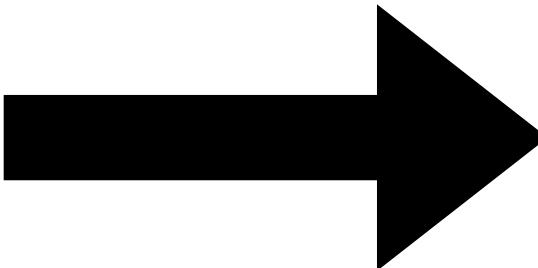


$$\begin{aligned}f(x) &= 3 * x \\f(x) &= x * x + 2 ?\end{aligned}$$

프로그램 합성

조건을 만족하는 프로그램을 자동으로 만들기

$$\begin{aligned}f(1) &= 3, \\f(2) &= 6, \\f(3) &= 11, \\f(x) &=?\end{aligned}$$



$$\begin{aligned}f(x) &= 3 * x \\f(x) &= x * x + 2 ?\end{aligned}$$

문법에 맞게 프로그램 합성하기

Syntax-Guided Synthesis*

- 프로그램 합성 대상 프로그램 문법과 문제를 표현하는 언어(SyGuS Language)와 적당한 규모의 벤치마크 제시
- 프로그램 합성 대회도 6회에 걸쳐 개최(2014~2019)**
- 충분히 다양한 합성 문제들을 표현할 수 있을 뿐 아니라, 서로 다른 합성기들의 성능을 공평하게 비교할 수 있어서 합성 분야에서 널리 사용됨
- 이번 연구도 SyGuS Language로 표현된 합성 문제를 입력으로 사용

*Rajeev Alur et al., "Syntax-guided synthesis", FMCAD'13 (<https://sygus.org/>)

** <https://sygus.org/comp/>

입출력 예제 기반 프로그래밍

Programming By Example (PBE)

"주어진 입출력 쌍들을 만족하는 프로그램 만들기"

- 주어진 예시를 만족하는 프로그램은 무수히 많을 수 있지만 그 중 어느 하나만 찾아도 OK
- 간단한 프로그램을 원한다면 입출력 2~3쌍으로도 원하던 답을 얻을 수도 있음 (예: MS Excel의 자동 채우기*)
- 대개 입출력 예시를 만족하는 가장 간단하고 작은 프로그램을 선호 (오컴의 면도날)

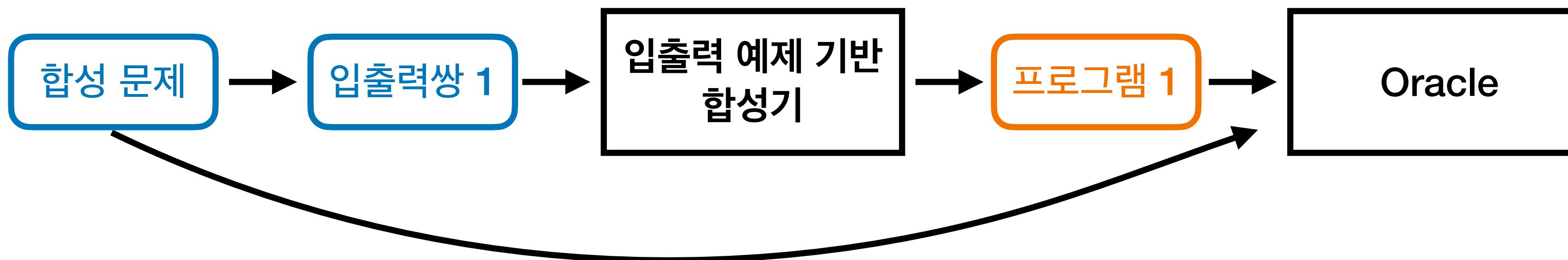
*Sumit Gulwani, "Automating String Processing in Spreadsheets using Input-Output Examples", POPL'11

<https://www.microsoft.com/en-us/research/publication/automating-string-processing-spreadsheets-using-input-output-examples/>

반례를 활용한 프로그램 합성

CounterExample-Guided Inductive Synthesis (CEGIS)*

- 아무 프로그램 합성 문제를 입출력 예제 기반 합성 문제로 바꾸어 풀 수 있는 "틀"
- 단, 답을 냈을 때 "정답" / "오답+반례" 중 하나를 알려주는 판정기(Oracle) 필요
- 말 됨: 엑셀 자동채우기를 쓸때도, 결과가 마음에 안 들면 입출력 쌍을 하나 더 넣어봄



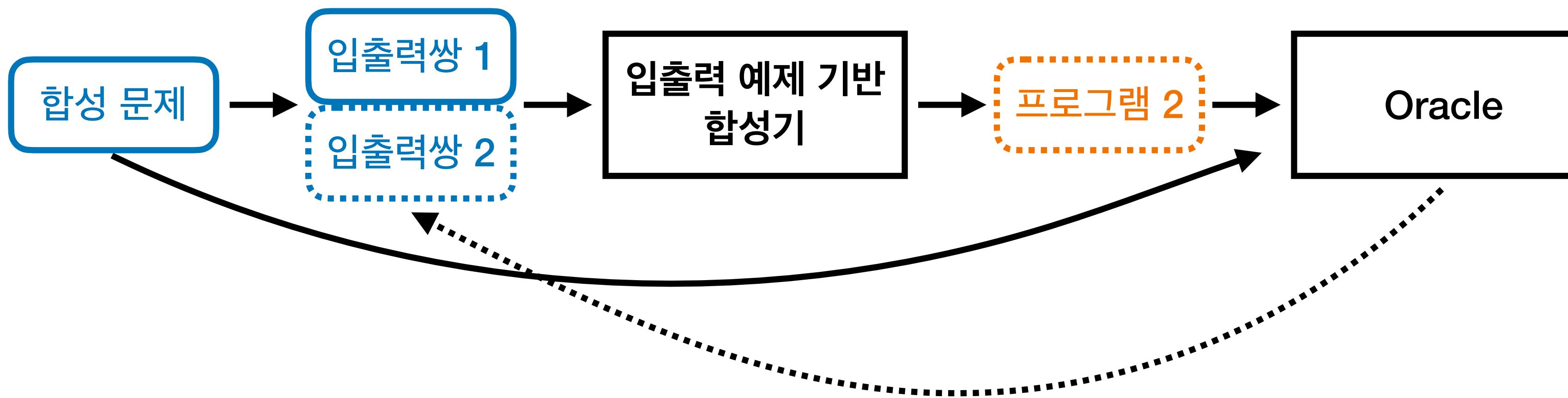
*Susmit Jha et al., "Oracle-Guided Component-Based Program Synthesis", ICSE'10

<https://susmitjha.github.io/papers/icse10.pdf>

반례를 활용한 프로그램 합성

CounterExample-Guided Inductive Synthesis (CEGIS)

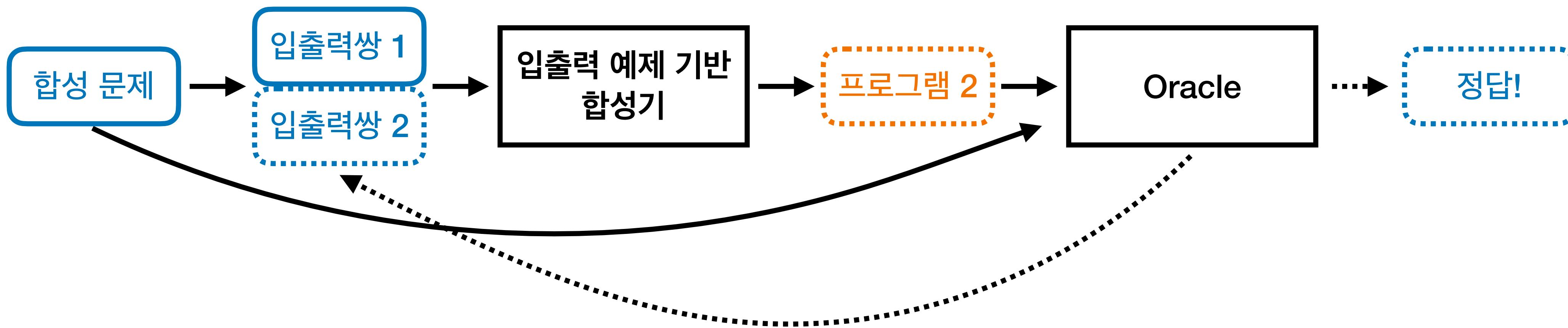
- 아무 프로그램 합성 문제를 입출력 예제 기반 합성 문제로 바꾸어 풀 수 있는 "틀"
- 단, 답을 냈을 때 "정답" / "오답+반례" 중 하나를 알려주는 판정기(Oracle) 필요
- 말 됨: 엑셀 자동채우기를 쓸때도, 결과가 마음에 안 들면 입출력 쌍을 하나 더 넣어봄



반례를 활용한 프로그램 합성

CounterExample-Guided Inductive Synthesis (CEGIS)

- 아무 프로그램 합성 문제를 입출력 예제 기반 합성 문제로 바꾸어 풀 수 있는 "틀"
- 단, 답을 냈을 때 "정답" / "오답+반례" 중 하나를 알려주는 판정기(Oracle) 필요
- 말 됨: 엑셀 자동채우기를 쓸때도, 결과가 마음에 안 들면 입출력 쌍을 하나 더 넣어봄



반례를 활용한 프로그램 합성

CounterExample-Guided Inductive Synthesis (CEGIS)

- 아무 프로그램 합성 문제를 입출력 예제 기반 합성 문제로 바꾸어 풀 수 있는 "틀"
- 단, 답을 냈을 때 "정답" / "오답+반례" 중 하나를 알려주는 판정기(Oracle) 필요
- 말 됨: 엑셀 자동채우기를 쓸때도, 결과가 마음에 안 들면 입출력 쌍을 하나 더 넣어봄
- 판정기로는 대부분 SMT Solver 사용
- 입출력 예제 기반 합성기만 잘 만들어도 SMT Solver에 기대어 많은 문제를 풀 수 있게 됨

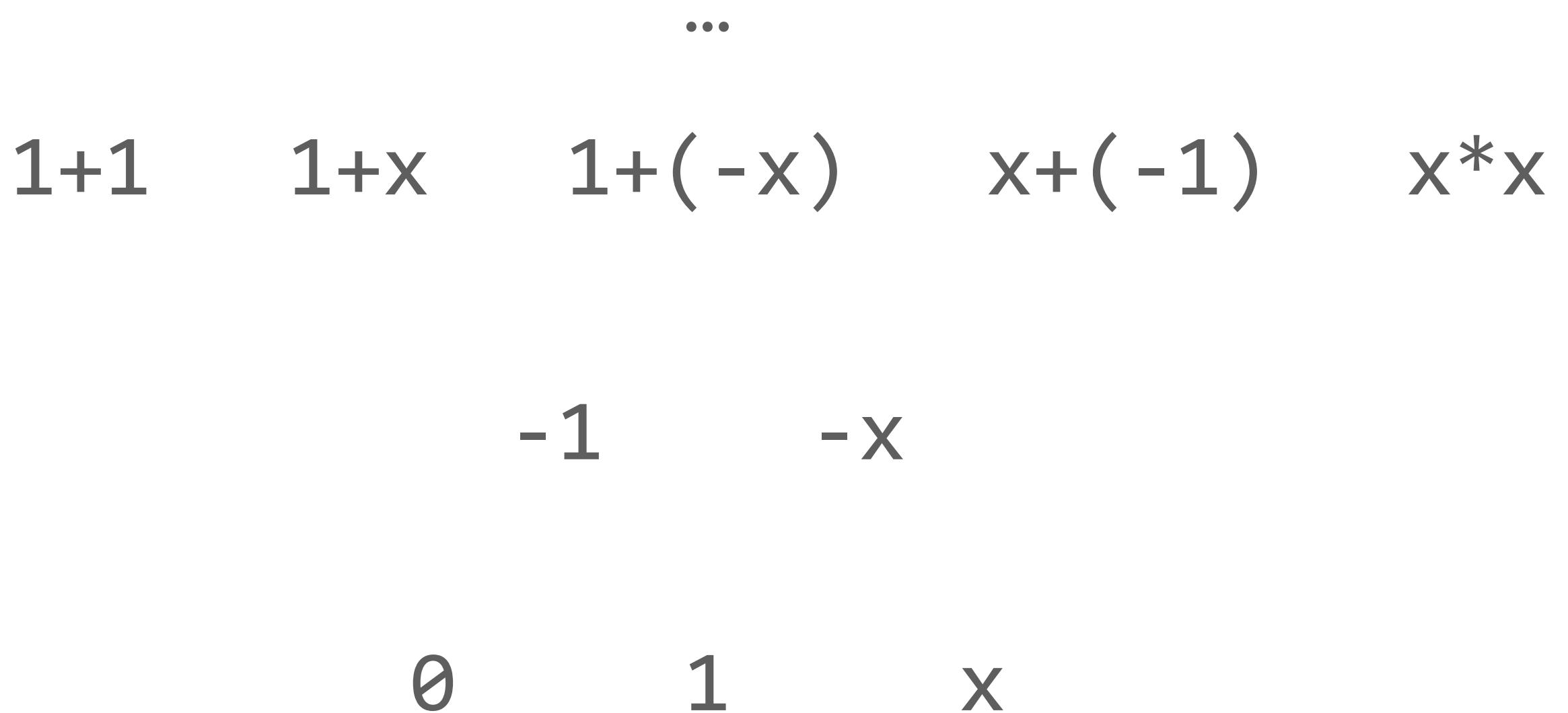
배경 지식

싹수분석을 활용하여
비트 벡터 **프로그램**
양방향 합성 잘 하기

기계적으로 프로그램 합성하기

작은 부품부터 조립해나가기(Bottom up)

↑
너무 많은 부품을 조립해봐야한다
절대 안 쓰일 부품을 판단하기는 매우 어려움



E	\rightarrow	0
		1
		x
		$E + E$
		$E * E$
		$-E$

기계적으로 프로그램 합성하기

상위 문법 규칙부터 나열해나가기(Top down)

E					
0	1	x	E+E	E*E	-E
1+1	1+x	x+x	E*(-E)		
-E+E	E*E+E	E*(E+E)			
	...				

$E \rightarrow 0$
1
x
E + E
E * E
-E

중간중간 짜수 없는 뼈대를 잘라낼 수 있지만
여전히 너무 많은 공간을 탐색해야한다

기계적으로 프로그램 합성하기

양방향으로 합성하기(Bidirectional)*

E						E	→	0
0	1	x	E+E	E*E	-E			
1+1	1+x	x+x	E*(-E)					
-E+E	E*E+E	E*(E+E)						
...								
뼈대는 위에서 아래로 뼈대의 빈칸을 부품으로 채워본다								
부품은 아래서 위로								
1+1	1+x	1+(-x)	x+(-1)	x*x				
-1	-x							
0	1	x						

배경 지식

싹수**분석**을 활용하여
비트 벡터 프로그램
양방향 합성 잘 하기

프로그램 정적 분석, 혹은 요약 해석

아주 짧은 설명

- 프로그램이 실행될 때 일어날 일들을 실행 전에 안전하게 확인하는 방법
 - 프로그램 실제 실행을 (수학적으로) 정의
 - 그 실행의 **요약**을 '잘' 정의
 - '잘' = 실제 실행을 안전하게 포섭하도록
 - 그 요약된 세계 안에서 분석 수행
 - 분석 결과를 해석

프로그램 정적 분석, 혹은 요약 해석

아주 짧은 설명

- 10의 자리와 1의 자리 홀/짝 여부가 궁금하다면? 홀짝 요약 세계(Domain)에서 분석
- 두 자리를 각각 '홀/짝/다/없' 중 하나로 표현
 - 다: 다 가능 / 없: 가능한 값이 없음
- x 는 입력에 뭐든 들어올 수 있으므로 "다다"
- y 는 00 또는 01이므로 "짝다"
- z 는 00 또는 10이므로 "다짝"
- w 는 01 또는 11이므로 "다홀"
- k 는 계산될 수 없으므로 "없없"

$f(x)$:

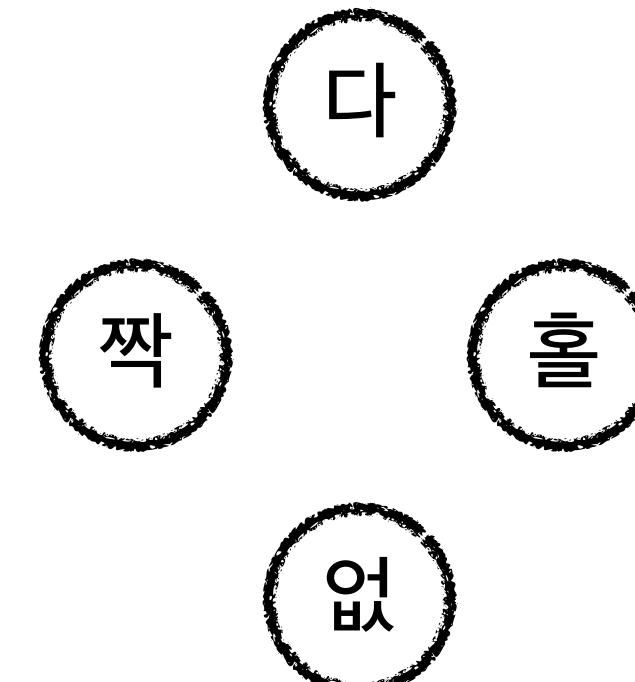
$$y = x \bmod 2$$

$$z = y * 10$$

$$w = z + 1$$

print w

$$k = w / 0$$



연구 내용

싹수분석을 활용하여
비트 벡터 프로그램
양방향 합성 잘 하기

프로그램 양방향 합성을 더 잘해볼 아이디어

정적 분석을 활용하여

- “싹수가 없는” 뼈대를 빨리 판단해서 버리면 유리함
 - “싹수가 없는” = 아직 미완성 프로그램이지만 빈 칸에 어떤 부품을 끼워봐도 조건을 만족하지 못할 것이 확실한
- 정적 분석으로 잘 할 수 있지 않을까?
 - 분석 대상: 조건(입출력 쌍들)과 미완성 프로그램
 - 분석 결과: “아직 모름” or “싹수 없음”

합성에 짝수 분석을 활용할 수 있는 예

조건:

$f(x)$ = “ x 가 8개의 비트 나열일 때,
 x 의 가장 낮은 자리부터 처음으로 0이 나오기 직전까지
쪽 연속된 1만 그대로 남기고 그 위는 모두 0으로 채운 값”

정답:

$((x + 1) \wedge x) \gg 1$

정방향 분석($x=b_11001111$):

[]	:	TTTTTTTT
x	:	11001111
[] x	:	11TT1111
([] x) >> 1	:	T11TT111

T = 임의의 비트가 가능
. = 어떤 비트도 불가능
0 = 이 위치엔 0만 가능
1 = 이 위치엔 1만 가능

입출력 예제:

$f(b_{1100\text{_____}}) = b_{0000\text{_____}}$
 $f(b_{01010\text{_____}}) = b_{00000\text{_____}}$

후보 미완성 프로그램:

$([] | x) \gg 1$

합성에 썩수 분석을 활용할 수 있는 예

조건:

$f(x)$ = “ x 가 8개의 비트 나열일 때,
 x 의 가장 낮은 자리부터 처음으로 0이 나오기 직전까지
쪽 연속된 1만 그대로 남기고 그 위는 모두 0으로 채운 값”

정답:

$((x + 1) \wedge x) \gg 1$

정방향 분석($x=b_11001111$):

입출력 예제:

$f(b_{1100\text{underline}1111}) = b_{0000\text{underline}1111}$

$f(b_{01010\text{underline}111}) = b_{00000\text{underline}111}$

[]	:	TTTTTTTT
x	:	11001111
[] x	:	11TT1111
([] x) >> 1	:	T11TT111

후보 미완성 프로그램:

$([] | x) \gg 1$

역방향 분석($x=b_{11001111}$):

([] x) >> 1	:	T11TT111	$\wedge 00001111 = 0..01111$	<- 썩수 없음
[] x	:	11TT1111		
x	:	11001111		
[]	:	TTTTTTTT		

합성에 썩수 분석을 활용할 수 있는 예

조건:

$f(x) = "x가 8개의 비트 나열일 때,
x의 가장 낮은 자리부터 처음으로 0이 나오기 직전까지
쪽 연속된 1만 그대로 남기고 그 위는 모두 0으로 채운 값"$

정답:

$((x + 1) \wedge x) \gg 1$

정방향 분석($x=b_11001111$):

입출력 예제:

$f(b_{1100\text{1111}}) = b_{0000\text{1111}}$
 $f(b_{01010\text{111}}) = b_{00000\text{111}}$

[]	:	TTTTTTTT
x	:	11001111
[] & x	:	TT00TTTT
([] & x) >> 1	:	TTT00TTT

후보 미완성 프로그램:

$([] \& x) \gg 1$

역방향 분석($x=b_{11001111}$):

$([] \& x) \gg 1$:	TTT00TTT	$\wedge 00001111 = 0000.111$	<- 썩수 없음
[] & x	:	TT00TTTT		
x	:	11001111		
[]	:	TTTTTTTT		

합성에 썩수 분석을 활용할 수 있는 예

조건:

$f(x) = "x가 8개의 비트 나열일 때,
x의 가장 낮은 자리부터 처음으로 0이 나오기 직전까지
쪽 연속된 1만 그대로 남기고 그 위는 모두 0으로 채운 값"$

정답:

$((x + 1) ^ x) \gg 1$

정방향 분석($x=b_11001111$):

입출력 예제:

$f(b_{1100\text{1111}}) = b_{0000\text{1111}}$

$f(b_{01010\text{111}}) = b_{00000\text{111}}$

[]	:	TTTTTTTT
x	:	11001111
[] / x	:	TTTTTTTT
([] / x) >> 1	:	TTTTTTTT

후보 미완성 프로그램:

$([] / x) \gg 1$

역방향 분석($x=b_{11001111}$):

$([] / x) \gg 1$:	TTTTTTTT	$\wedge 00001111 = 00001111$
[] / x	:	TTTTTTTT	$\wedge 0001111T = 0001111T$ (30 or 31)
x	:	11001111	$\wedge 0000TTTT ([0,8]) = ..001111 <- 썩수 없음$
[]	:	TTTTTTTT	

합성에 썩수 분석을 활용할 수 있는 예

조건:

$f(x) = "x가 8개의 비트 나열일 때,
x의 가장 낮은 자리부터 처음으로 0이 나오기 직전까지
쪽 연속된 1만 그대로 남기고 그 위는 모두 0으로 채운 값"$

정답:

$((x + 1) ^ x) \gg 1$

정방향 분석($x=b_11001111$):

입출력 예제:

$f(b_{1100\text{1111}}) = b_{0000\text{1111}}$
 $f(b_{01010\text{111}}) = b_{00000\text{111}}$

[]	:	TTTTTTTT
x	:	11001111
[] ^ x	:	TTTTTTTT
([] ^ x) >> 1	:	TTTTTTTT

후보 미완성 프로그램:

$([] ^ x) \gg 1$

역방향 분석($x=b_11001111$): 아직 모름 => 마저 진행

([] ^ x) >> 1	:	TTTTTTTT	$\wedge 00001111 = 00001111$
[] ^ x	:	TTTTTTTT	$\wedge 0001111T = 0001111T$
x	:	11001111	
[]	:	TTTTTTTT	$\wedge 1101000T = 1101000T$

연구 내용

싹수분석을 활용하여
비트 벡터 프로그램
양방향 합성 잘 하기

합성 및 정적 분석 대상 언어

- SyGuS Language로 표현 가능한 합성 대상 언어 중 BitVector 프로그램
 - = CPU에서 연산 가능한 정수 프로그램

- 값: 64비트 정수(64개의 비트 나열)
- 연산: 널리 쓰이는 산술 및 비트 연산
- []: 프로그램의 아직 완성되지 않은 부분
- 모든 변수는 함수의 매개변수(let-선언 없음)

$$\begin{array}{c} E \rightarrow 0 \mid 1 \mid -1 \\ | \\ x \\ | \\ \diamond E \\ | \\ E \circ E \\ | \\ [] \\ \diamond \rightarrow \sim \mid - \\ \circ \rightarrow \wedge \mid \vee \mid \oplus \mid \ll \mid \gg \mid \ggg \\ | \\ + \mid - \mid * \mid / \mid \% \mid /_s \mid \%_s \end{array}$$

벤치 마크: 해커의 기쁨 문제

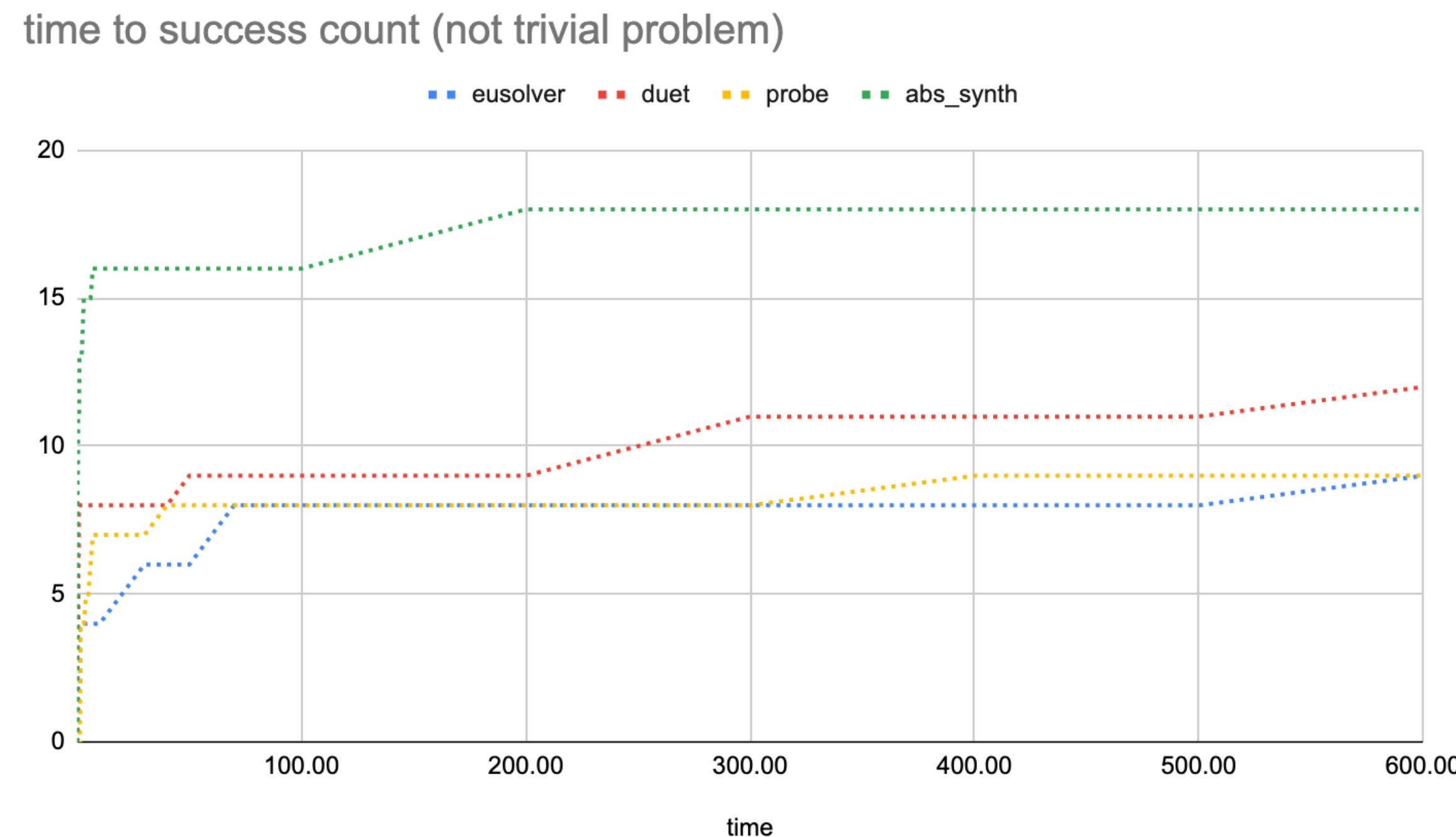
SyGuS-Comp 2018 Benchmark: BV/Hacker's Delight

- SyGuS 합성 경진대회 (2018) 공식 벤치마크* 중 해커의 기쁨 문제(hd-xx-dy)
 - 여러가지 기기묘묘한 비트 조작 함수 최적화 문제
 - 단, 벤치마크 원본은 32비트 문제지만 우리는 64비트로 확장하여 사용중
 - 곧 임의의 비트수 대상으로 합성/분석기 확장 예정
- 44가지 BitVector 합성 문제
- 동일한 목표에 대해서도 다양한 대상 언어 문법으로 난이도 다양화
 - 예: 정답에 포함되는 연산자만 포함 vs 모든 가능한 연산자 포함
 - xx 번호가 높을수록 목표가 어려운 편, y 번호가 높을수록 대상 언어 문법 복잡한 편

*https://github.com/SyGuS-Org/benchmarks/tree/master/comp/2018/General_Track

얼마나 잘 기존 합성 도구와 비교

- 21*개의 해커의 기쁨 벤치마크로 3가지 도구와 비교 **eusolver[1]**, **duet[2]**, **probe[3]**



*44개 문제 중 단순 나열로도 금방 답이 나오는 쉬운 문제 23개 제외

[1] eusolver: Rajeev Alur et al., "Scaling Enumerative Program Synthesis via Divided and Conquer, TACAS'17

[2] duet: Woosuk Lee, POPL'21

[3] probe: Shraddha Barke et al., "Just-in-time learning for bottom-up enumerative synthesis", OOPSLA'20

얼마나 잘

기존 합성 도구와 비교

- timeout \geq 1시간
- error = 합성 도중 처리하지 못한 예외 발생
- 총 16개의 문제에서 다른 도구보다 빨리 합성
- 1시간 시간제한 안에 18문제 합성에 성공
(2위: duet의 12문제)

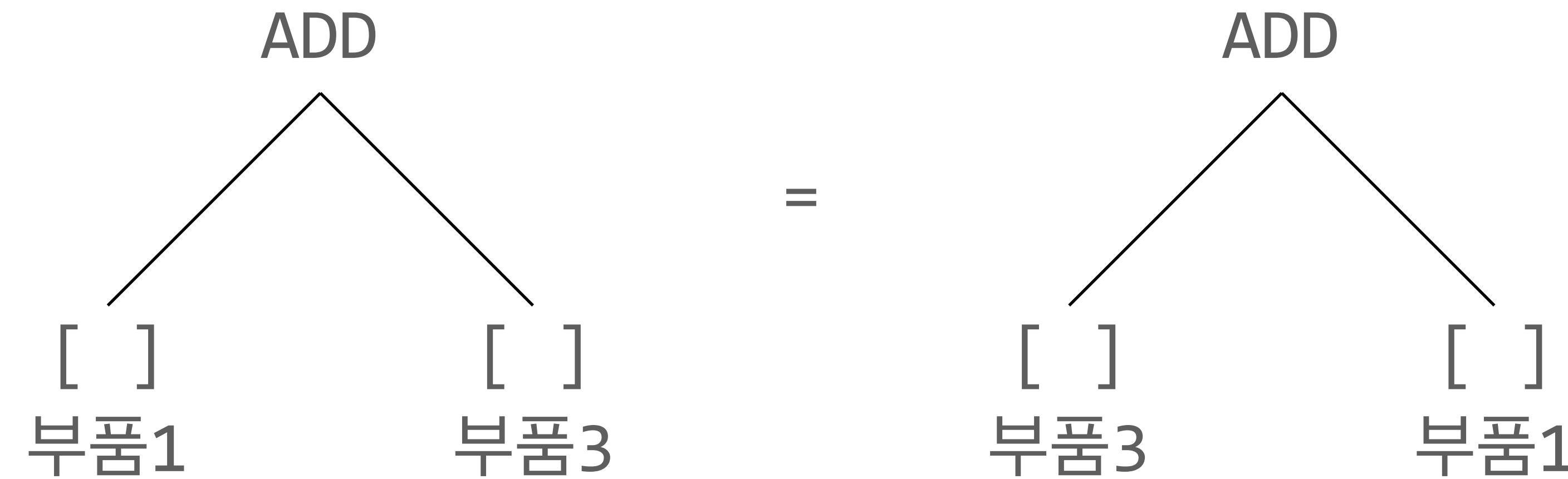
problem	eusolver	duet	probe	abs_synth
hd-09-d0-prog	0.27	0.15	error	0.09
hd-09-d1-prog	10.62	timeout	error	0.13
hd-09-d5-prog	timeout	timeout	error	0.44
hd-13-d0-prog	1.35	0.08	error	0.10
hd-13-d1-prog	69.75	0.19	error	0.18
hd-13-d5-prog	timeout	0.78	error	0.24
hd-14-d0-prog	52.04	0.59	1.41	0.22
hd-14-d1-prog	timeout	232.51	6.26	0.99
hd-14-d5-prog	timeout	timeout	5.21	2.99
hd-15-d0-prog	26.80	43.85	3.53	0.16
hd-15-d1-prog	timeout	timeout	38.18	0.98
hd-15-d5-prog	timeout	timeout	error	6.88
hd-17-d0-prog	0.52	0.60	1.43	0.15
hd-17-d1-prog	1.44	0.28	1.22	0.18
hd-17-d5-prog	599.13	timeout	1.39	2.98
hd-19-d0-prog	timeout	0.74	318.29	0.37
hd-19-d1-prog	timeout	timeout	error	179.18
hd-19-d5-prog	timeout	timeout	timeout	timeout
hd-20-d0-prog	timeout	234.13	error	191.89
hd-20-d1-prog	timeout	560.91	timeout	timeout
hd-20-d5-prog	timeout	timeout	error	timeout

어떻게 잘

- 합성을 잘
 - 불필요한 탐색 덜 하기
 - 부품을 필요할 때 꺼내기 좋게 갈무리
- 분석을 잘
 - 아주 정교한 분석
 - 상호 보완하는 복합 도메인, 정교한 역방향 분석 => 최대한 정확한 요약값 추출
 - 요약값의 구체화까지 적극 활용

합성: 불필요한 조합 덜 하기

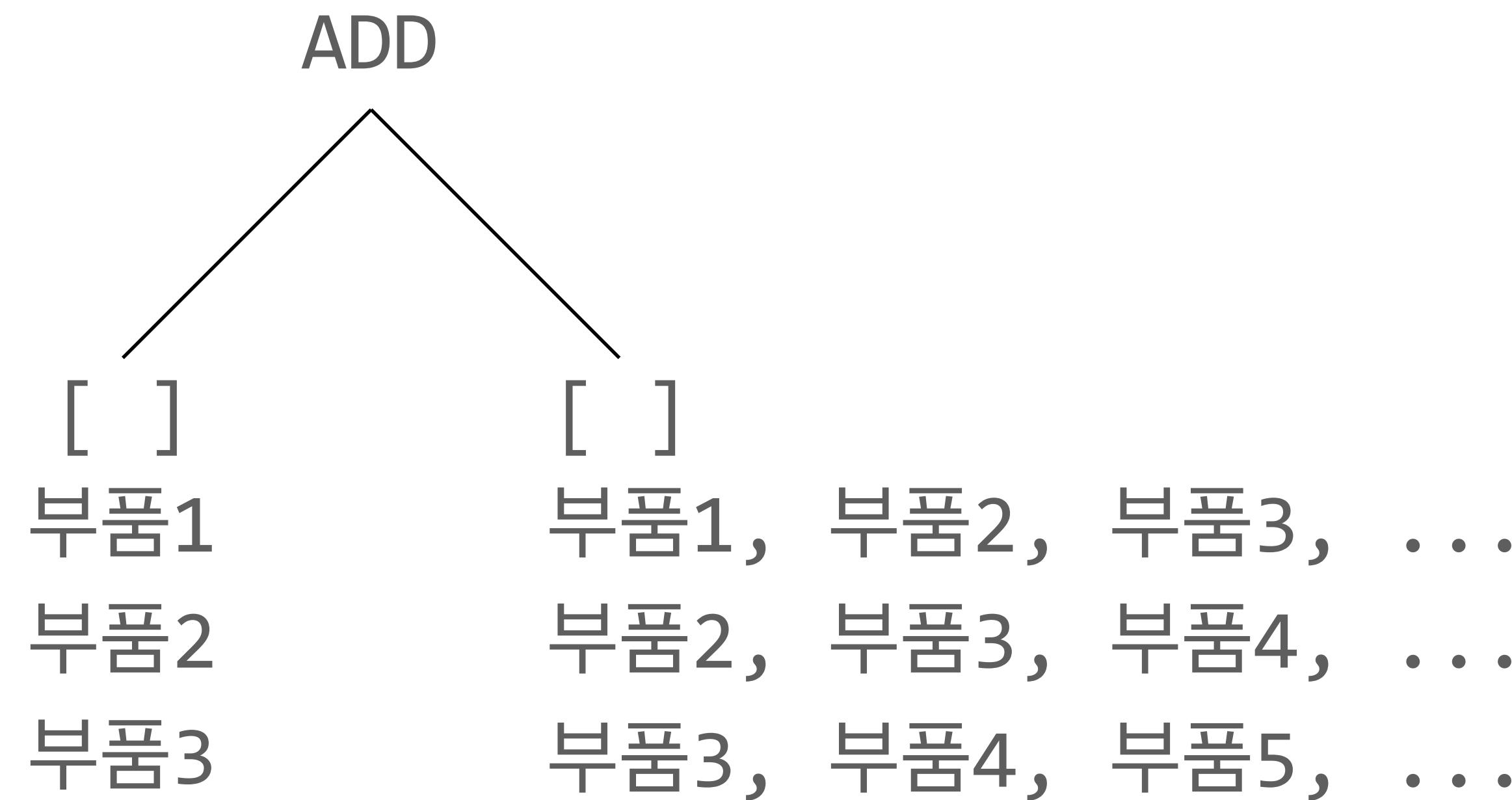
교환 법칙 성립 연산에서 결과 같아질 조합 덜 넣어보기



합성: 불필요한 조합 덜 하기

교환 법칙 성립 연산에서 결과 같아질 조합 덜 넣어보기

- 교환 법칙이 성립하는 연산의 양쪽이 모두 빈칸일 때만 적용
- 단순하지만, 적용 가능한 미완성 프로그램에 대해선 탐색 과정을 절반정도 단축



합성: 만든 부품 저장 방법

3가지 표로 저장

조건:

$$\begin{aligned}x &= 3 \rightarrow f(x) = 12 \\x &= 7 \rightarrow f(x) = 56 \\x &= 4 \rightarrow f(x) = 20\end{aligned}$$

크기 \leftrightarrow 부품들

size	부품
1	0, 1, x
2	-x, -1
3	$x>>1, x<<1, x+x, x-1, \dots$
4	$-(x+x), (-x)>>1, \dots$
5	$(x+x)^*x, (1<<x)/x, x^*(x-1), \dots$
6	$(1<<x)^*(-x), -(x+x)^*x, x^*(-x-1)$
7	...
4	$1+x$
	...

모든 입력 계산 결과값 \leftrightarrow 부품

x=3	x=7	x=4	부품
0	0	0	0
3	7	4	x
2	6	3	$x-1$
4	8	5	$x+1$
1	3	2	$x>>1$
6	14	8	$x<<1$
9	49	16	x^*x
4	8	3	$1+x$
			...

각 입력별 계산 결과값 \leftrightarrow 부품들

	x=3 부품	x=7 부품	x=4 부품
0	$0, x>>(1+1)$
1	$1, x>>1, x-(1+1)$
2	$x-1, x>>1+1$
3	x
4	$x+1, (x-1)<<1$
5	$x<<1-1$
6	$x<<1, x^*(x-1)$
4	$1+x$

분석: 상호 보완하는 복합 도메인 활용

Reduced Product Domain

- 부호 없는 구간, 부호 있는 구간, 요약 비트 나열 도메인을 모두 활용
 - 구체화 함수는 각 부분 도메인의 구체화 함수 결과의 교집합
 - 예: {b11111100, b11111110} 을 요약하면 ([252, 254], [-4, -2], b111111T0)
- 각 부분 도메인이 서로의 정확도를 보완 (Reduction Operator)
 - 비트 연산 결과에서는 구간 값, 산술 연산 결과에서는 비트 값의 정확도가 떨어지는데, 떨어진 정확도를 정확도 높은 도메인의 결과를 이용해 보완할 수 있다
 - 예: 요약 연산 결과가 (Top, Top, bT0001T1T) 이었다면 ([10, 143], [-118, 15], bT0001T1T) 로 구간 값을 살릴 수 있다

상호 보완 함수: 비트 나열 도메인 보완

Reduction Operator

- 부호 없는 구간 도메인 -> 비트 나열 도메인

- lb, ub를 비트로 표현했을 때의 앞부터 연속으로 동일한 부분(common prefix)은 그대로 사용, 처음 달라지는 위치부터 끝까지 모두 T으로 채움

$$[10, 30] = [b00001010, b00011110] \Rightarrow \begin{array}{c} 00001010 \\ 00011110 \end{array} \Rightarrow b\mathbf{000}TTTTT$$

- 부호 있는 구간 도메인 -> 비트 나열 도메인

- 양끝값의 부호가 같으면 부호 없는 구간 도메인과 같은 함수 적용
- 양끝값 부호가 다르면 활용 포기 (구간에 111...1 과 000...0 을 포함하게 되므로)

$$[-5, -11] = [b11111011, b11110101] \Rightarrow \begin{array}{c} 11111011 \\ 11110101 \end{array} \Rightarrow b\mathbf{1111}TTTT$$

$$\gamma_s[-5, 3] = \{-5, \dots, -1, 0, \dots, 3\} \Rightarrow \alpha_b\{-5, \dots, -1, 0, \dots, 3\} = \alpha_b\{b00000000, b11111111, \dots\} = bTTTTTTT = \text{Top}$$

상호 보완 함수: 부호 없는 구간 도메인 보완

Reduction Operator

- 비트 나열 도메인 -> 부호 없는 구간 도메인
 - T 비트를 모두 0으로 만들면 최소, 1로 만들면 최대

$$b001\textcolor{red}{TT}00\textcolor{red}{T} \Rightarrow [b001\textcolor{red}{00000}, b001\textcolor{red}{11001}] = [32, 57]$$

- 부호 있는 구간 도메인 -> 부호 없는 구간 도메인
 - 양끝값 부호가 같으면 양끝값의 비트 표현을 부호없이 해석해서 사용
 - 양끝값 부호가 다르면 활용 포기 (구간에 -1과 0을 포함하게 되므로)

$$[-20, -15] = [b11101100, b11110001] \Rightarrow [236, 241]$$

$$\begin{aligned}\gamma_s[-5, 3] &= \{-5, \dots, -1, 0, \dots, 3\} \Rightarrow \alpha_u\{-5, \dots, -1, 0, \dots, 3\} \\ &= \alpha_u\{b11111111, b00000000, \dots\} \\ &= [0, 255] = \text{Top}\end{aligned}$$

상호 보완 함수: 부호 있는 구간 도메인 보완

Reduction Operator

- 부호 없는 구간 도메인 -> 부호 있는 구간 도메인
 - 양끝값을 비트로 표현했을 때 최상위 비트가 동일하면: 양끝값 비트 표현을 부호 있는 것으로 해석하여 사용
 - 최상위 비트가 다르면 활용 포기 (구간에 100....0 과 011...1 을 포함하게 되므로)

$$[236, 241] = [b11101100, b11110001] \Rightarrow [-20, -15]$$

$$\alpha_s\{120, \dots, 127, 128, \dots, 140\}$$

$$\begin{aligned}\gamma_u[120, 140] &= \{120, \dots, 127, 128, \dots, 140\} \Rightarrow = \alpha_s\{b01111111, b10000000, \dots\} \\ &= \alpha_s\{127, -128, \dots\} \\ &= [-128, 127] = \text{Top}\end{aligned}$$

상호 보완 함수: 부호 있는 구간 도메인 보완

Reduction Operator

- 비트 나열 도메인 -> 부호 있는 구간 도메인
 - 최상위 비트가 0이나 1이면 부호 없는 구간 도메인과 같은 함수 적용

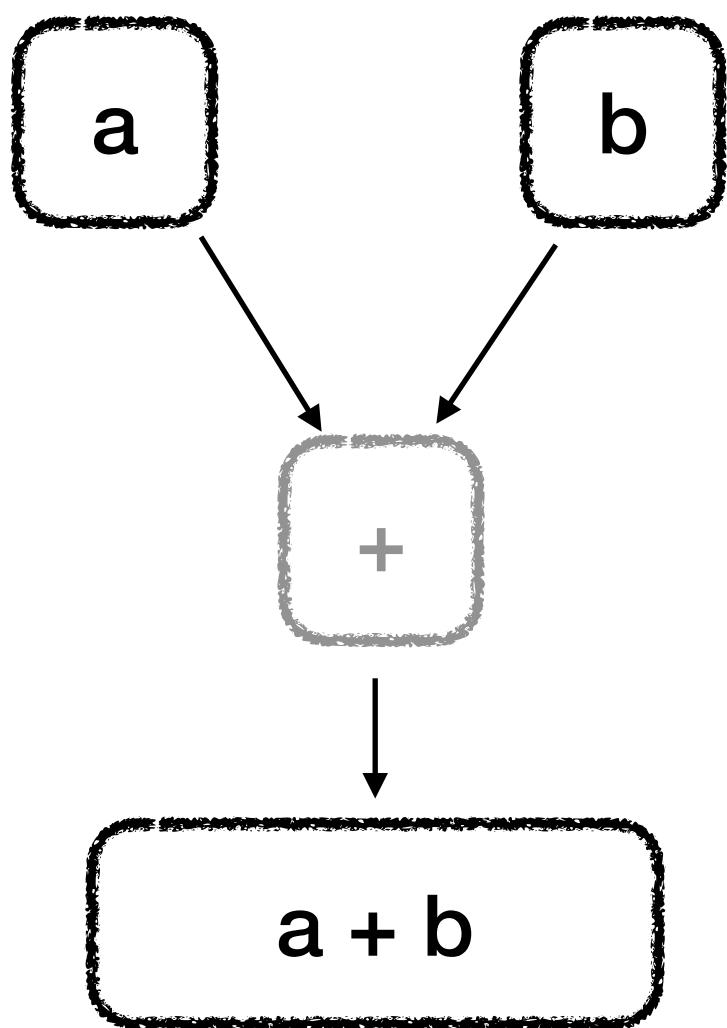
$$b101\textcolor{red}{TT}00T \Rightarrow [b101\textcolor{red}{00000}, b101\textcolor{red}{11001}] = [-96, -71]$$

- 최상위 비트가 T이면
 - 최소: 최상위비트는 1, 나머지 T 비트는 0으로 채운 것 (포함하고 있는 음수 중 가장 작은 값)
 - 최대: 최상위 비트는 0, 나머지 T 비트는 1로 채운 것 (포함하고 있는 양수 중 가장 큰 값)

$$b\textcolor{red}{T}01\textcolor{red}{TT}00T \Rightarrow [\textcolor{red}{b10100000}, \textcolor{red}{b00111001}] = [-96, 57]$$

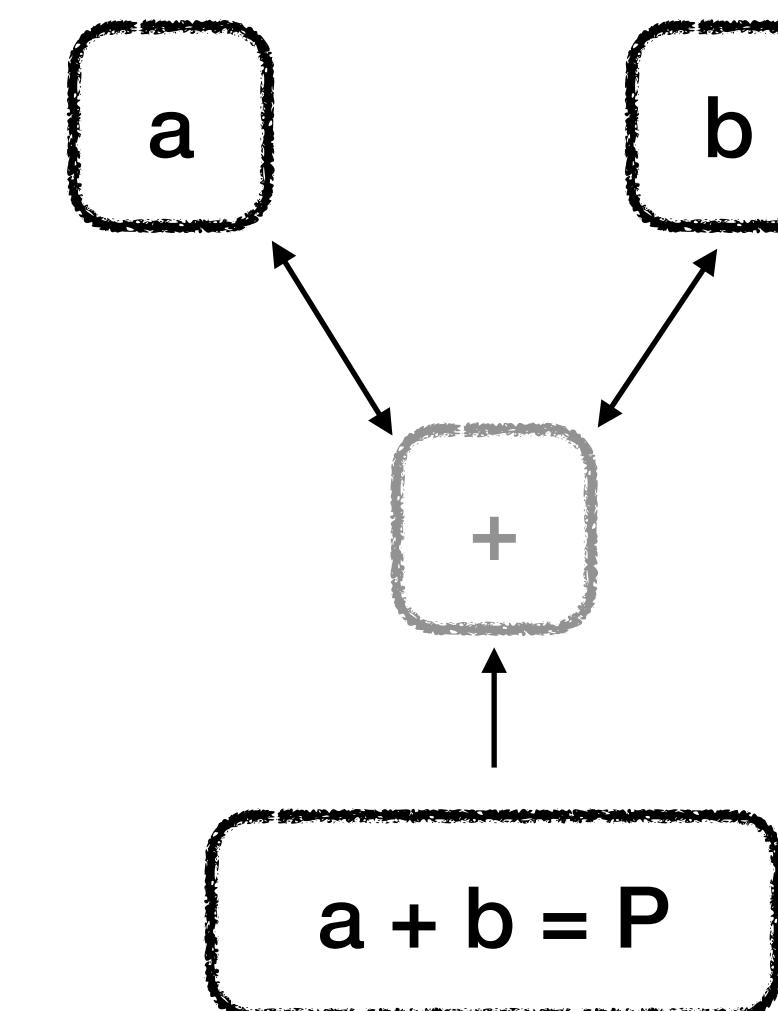
분석: 정교한 역방향 분석

정방향



a의 요약을 계산하고
b의 요약을 계산하고
그것으로 $a+b$ 의 요약을 계산

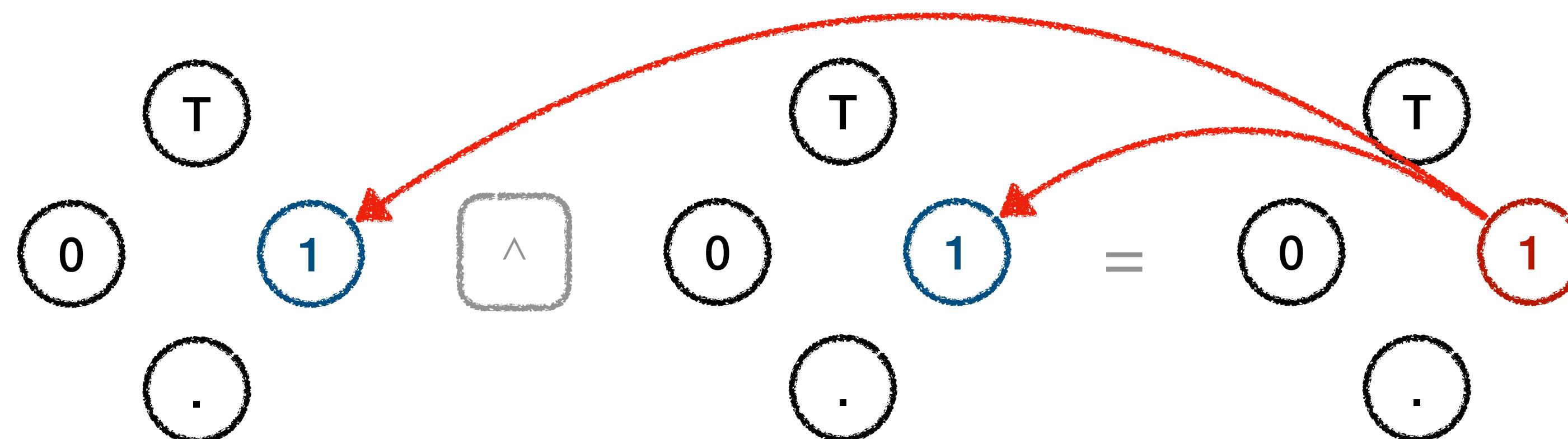
역방향



$a+b$ 가 만족해야할 조건 P가 주어졌을 때
a, b 각각의 요약이 만족해야할 조건을 계산
(a, b의 정방향 요약이 이미 계산되어있다면 활용 가능)

역방향 분석 예: and(\wedge)

각 비트 자리마다



역방향 결과

$T10TTT1T$

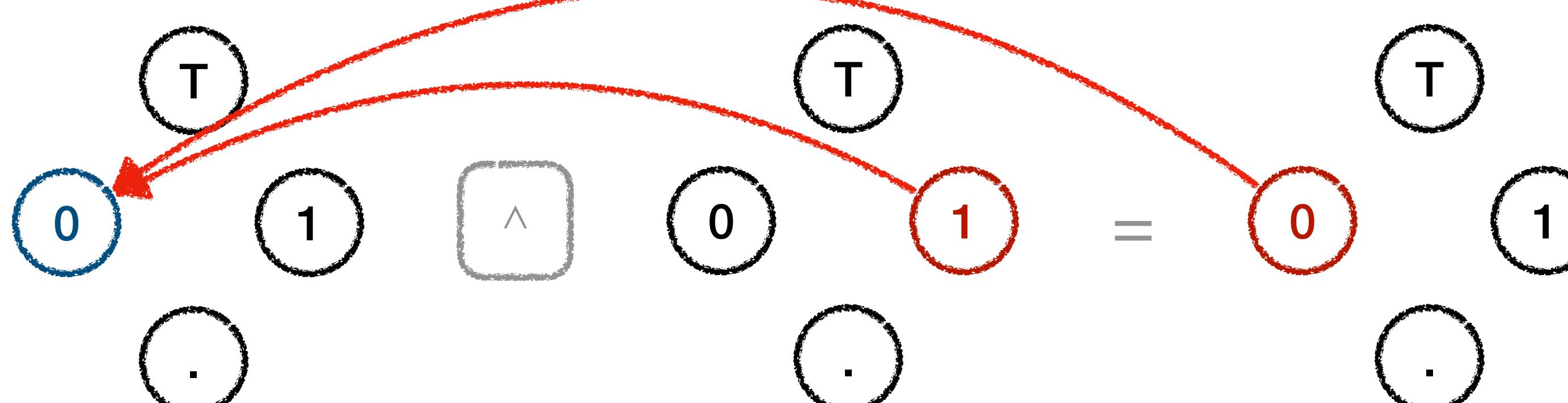
정방향 결과

$x=TTTTTTTT$

$T110101T$
 $y=TT10101T$

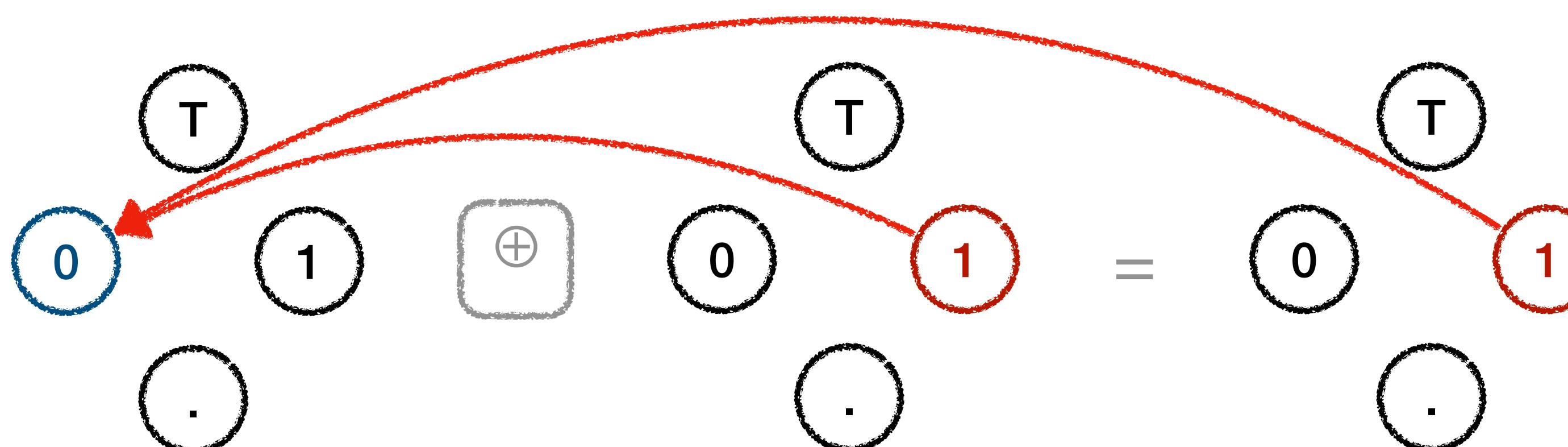
and

$P=T100T01T$



역방향 분석 예: xor(\oplus)

각 비트 자리마다



역방향 결과

T010T01T

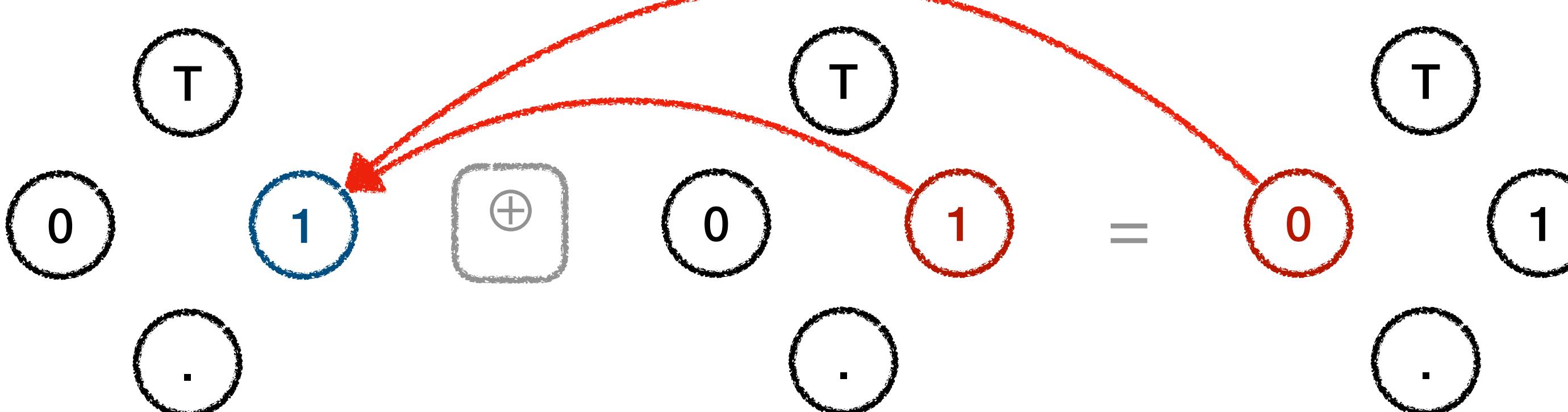
정방향 결과

x=T0T0TT1T

T110100T
y=TT10100T

xor

P=T100T01T



역방향 분석 예: lshr(>>>)

- $a >>> b = P$ 가 주어졌을 때
- P 의 1이 아닌 연속된 상위 비트 갯수로 b 의 최댓값 제한
 - logical shift right를 N 번 했다면 상위 N 개 비트는 반드시 0이어야 하므로
- b 를 구체화한 값들 만큼씩 P 를 왼쪽으로 shift 한 결과들을 모두 뭉친 결과로 a 의 비트 나열을 보완

정방향	$\overrightarrow{TTTTTTTT \quad >> \quad [2,4] \quad = \quad 00TTTTTT}$
역방향	$T\overleftarrow{11TTTTT} \quad >>> \quad [2,3] \quad = \quad 000111T0$
a =	$0111T0TT$
a =	$111T0TTT$
	$a \gg 2 = 000111T0$
	$a \gg 3 = 000111T0$

역방향 분석 예: mul

빈칸이 하나 남은 곱셈의 역방향 분석

- 단순히 생각하면 역함수가 있을 것 같은데?

$$\begin{aligned} 6 * a &= 24 \\ a &= 4? \end{aligned}$$

- 일반 정수가 아닌 비트 벡터 도메인이라 단순하지 않음

$$\begin{aligned} 6 * a &\equiv 24 \pmod{2^{64}} \\ a &\equiv 4? \pmod{2^{64}} \end{aligned}$$

비트 벡터 곱셈의 역방향 분석

예제(8비트로 단순화)

$$Q1: 6 * a = 24 \pmod{256}$$

비트 벡터 곱셈의 역방향 분석

예제(8비트로 단순화)

$$Q1: 6 * a = 24 \pmod{256}$$



변형된 같은 문제

$$Q2: 3 * a = 12 \pmod{128}$$

비트 벡터 곱셈의 역방향 분석

예제(8비트로 단순화)

$$Q1: 6 * a = 24 \pmod{256}$$



변형된 같은 문제

$$Q2: 3 * a = 12 \pmod{128}$$



Q2를 풀기 위해 필요

$$Q3: 3 * x = 1 \pmod{128}$$

$$x = 3^{-1}$$

비트 벡터 곱셈의 역방향 분석

예제(8비트로 단순화)

$$Q1: 6 * a = 24 \pmod{256}$$



변형된 같은 문제

$$Q2: 3 * a = 12 \pmod{128}$$



Q2를 풀기 위해 필요

$$Q3: 3 * x = 1 \pmod{128}$$



3과 128이 서로소일 때(3이 홀수일 때)

x는 유일하며

확장된 유clidean 알고리즘으로 계산 가능

$$x = 43 \pmod{128}$$

$x = 3^{-1}$

비트 벡터 곱셈의 역방향 분석

예제(8비트로 단순화)

$$Q1: 6 * a = 24 \pmod{256}$$



변형된 같은 문제

$$Q2': 3 * 43 * a = 12 * 43 \pmod{128}$$

$$43 = 3^{-1} \pmod{128}$$

$$Q2: 3 * a = 12 \pmod{128}$$



Q2를 풀기 위해 필요

$$Q3: 3 * x = 1 \pmod{128}$$



3과 128이 서로소일 때(3이 홀수일 때)

x는 유일하며

확장된 유clidean 알고리즘으로 계산 가능

$$x = 43 \pmod{128}$$

$$x = 3^{-1}$$

비트 벡터 곱셈의 역방향 분석

예제(8비트로 단순화)

$$Q1: 6 * a = 24 \pmod{256}$$



변형된 같은 문제

$$Q2': 3 * 43 * a = 12 * 43 \pmod{128}$$

$$Q2': a = 4 \pmod{128}$$

$$12 * 43 - 128 * 4 = 4$$

$$Q2: 3 * a = 12 \pmod{128}$$



Q2를 풀기 위해 필요

$$Q3: 3 * x = 1 \pmod{128}$$



3과 128이 서로소일 때(3이 홀수일 때)

x는 유일하며

확장된 유clidean 알고리즘으로 계산 가능

$$x = 43 \pmod{128}$$

비트 벡터 곱셈의 역방향 분석

예제(8비트로 단순화)

$$Q1: 6 * a = 24 \pmod{256}$$



변형된 같은 문제

$$Q2: 3 * a = 12 \pmod{128}$$



Q2를 풀기 위해 필요

$$Q3: 3 * x = 1 \pmod{128}$$



3과 128이 서로소일 때(3이 홀수일 때)

x는 유일하며

확장된 유clidean 알고리즘으로 계산 가능

$$x = 43 \pmod{128}$$

$$Q2': 3 * 43 * a = 12 * 43 \pmod{128}$$

$$Q2': a = 4 \pmod{128}$$

$$a = 128K + 4$$

$$Q1: a = 4 \text{ or } 132 \pmod{256}$$

$$\begin{aligned} 6 * 132 \\ = 792 \\ = 256 * 3 + 24 \end{aligned}$$

비트 벡터 곱셈의 역방향 분석

알고리즘

$$c1 * [] = c2 \bmod 2^{64}$$

$$(c1 \gg k) * [] = (c2 \gg k) \bmod 2^{(64-k)}$$

$c1 \gg k$ 를 홀수로 만드는 가장 작은 k 선택 (= 비트 표현에서 낮은자리에 연속되는 0의 갯수)

비트 벡터 곱셈의 역방향 분석

알고리즘

$$c1 * [] = c2 \bmod 2^{64}$$

$$\text{extended_euclidian} \quad (c1 \gg k) * [] = (c2 \gg k) \bmod 2^{(64-k)}$$

$$c1_inv * (c1 \gg k) * [] = c1_inv * (c2 \gg k) \bmod 2^{(64-k)}$$

비트 벡터 곱셈의 역방향 분석

알고리즘

$$c1 * [] = c2 \bmod 2^{64}$$

$$(c1 \gg k) * [] = (c2 \gg k) \bmod 2^{(64-k)}$$

$$c1_inv * (c1 \gg k) * [] = c1_inv * (c2 \gg k) \bmod 2^{(64-k)}$$

$$[] = c1_inv * (c2 \gg k) \bmod 2^{(64-k)}$$

$$[] = c1_inv * (c2 \gg k) + \underline{N*2^{(64-k)}} \bmod 2^{64}$$

$$[] = \underline{\text{TTTT}} \{c1_inv * (c2 \gg k) \text{의 비트 표현}\}$$

k개의 Top 비트

구체화 결과 적극 활용

값이 명확하면 부품 선택, 맞는 부품 없으면 짹수 없음

조건: $x = 3 \rightarrow f(x) = 11$

후보: $\text{ADD}([], []) = 11$

[부품 상자]

0: 0

1: 1

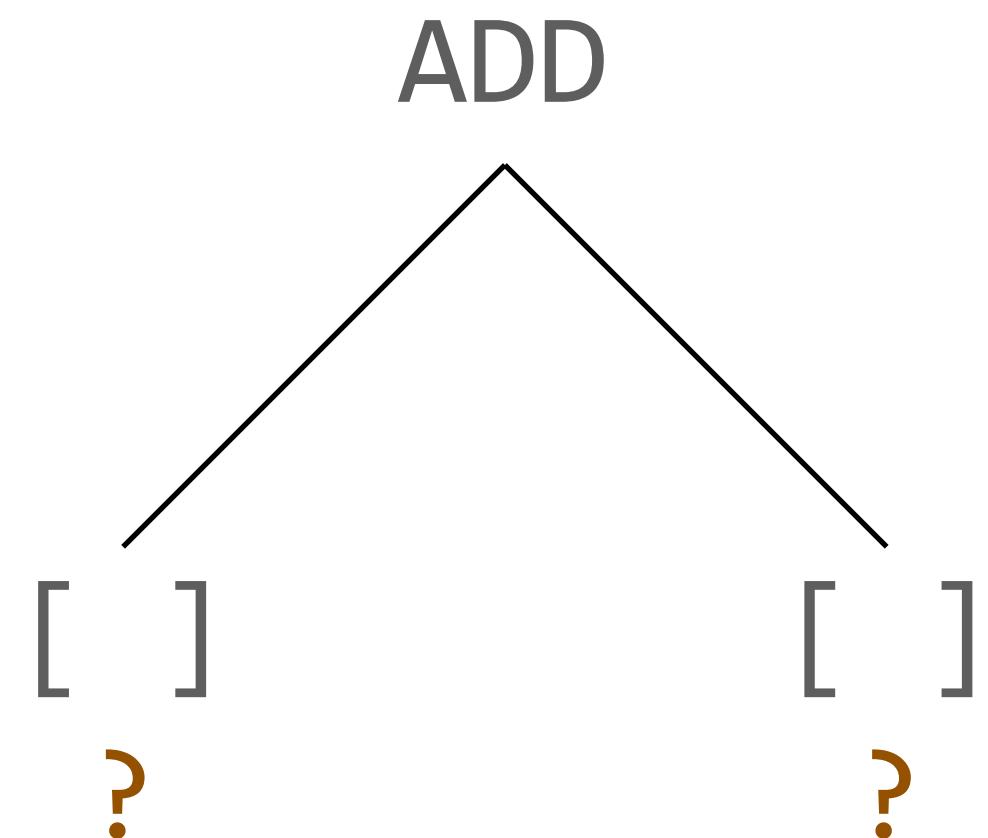
2: $x - 1$

3: x

4: $x + 1$

6: $x \ll 1$

9: $x * x$



짬수 분석 결과

구체화 결과 적극 활용

값이 명확하면 부품 선택, 맞는 부품 없으면 짹수 없음

조건: $x = 3 \rightarrow f(x) = 11$

후보: $\text{ADD}([x], []) = 11$

[부품 상자]

0: 0

1: 1

2: $x - 1$

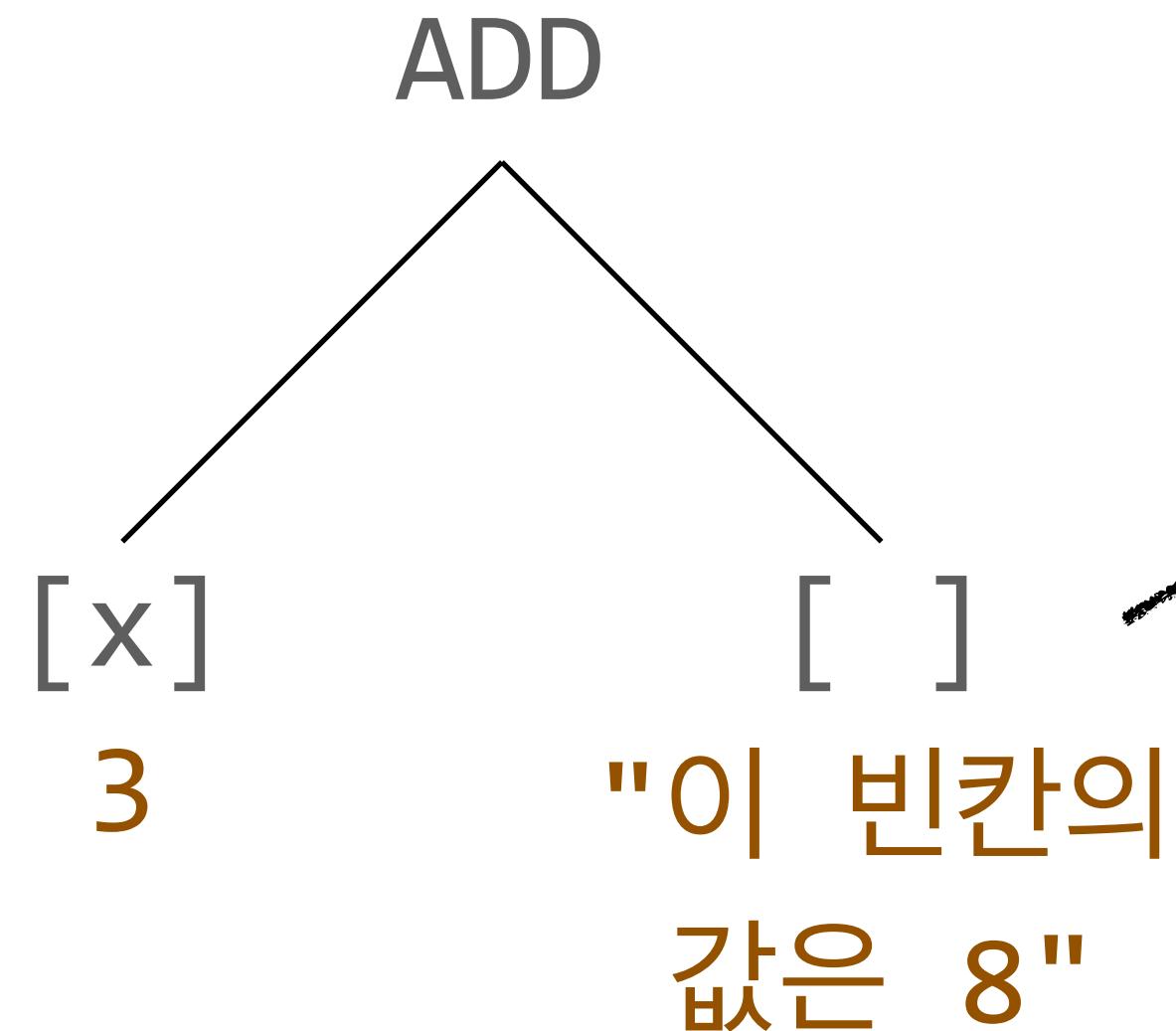
3: x

4: $x + 1$

6: $x \ll 1$

9: $x * x$

짬수 분석 결과



구체화 결과 적극 활용

값이 명확하면 부품 선택, 맞는 부품 없으면 싹수 없음

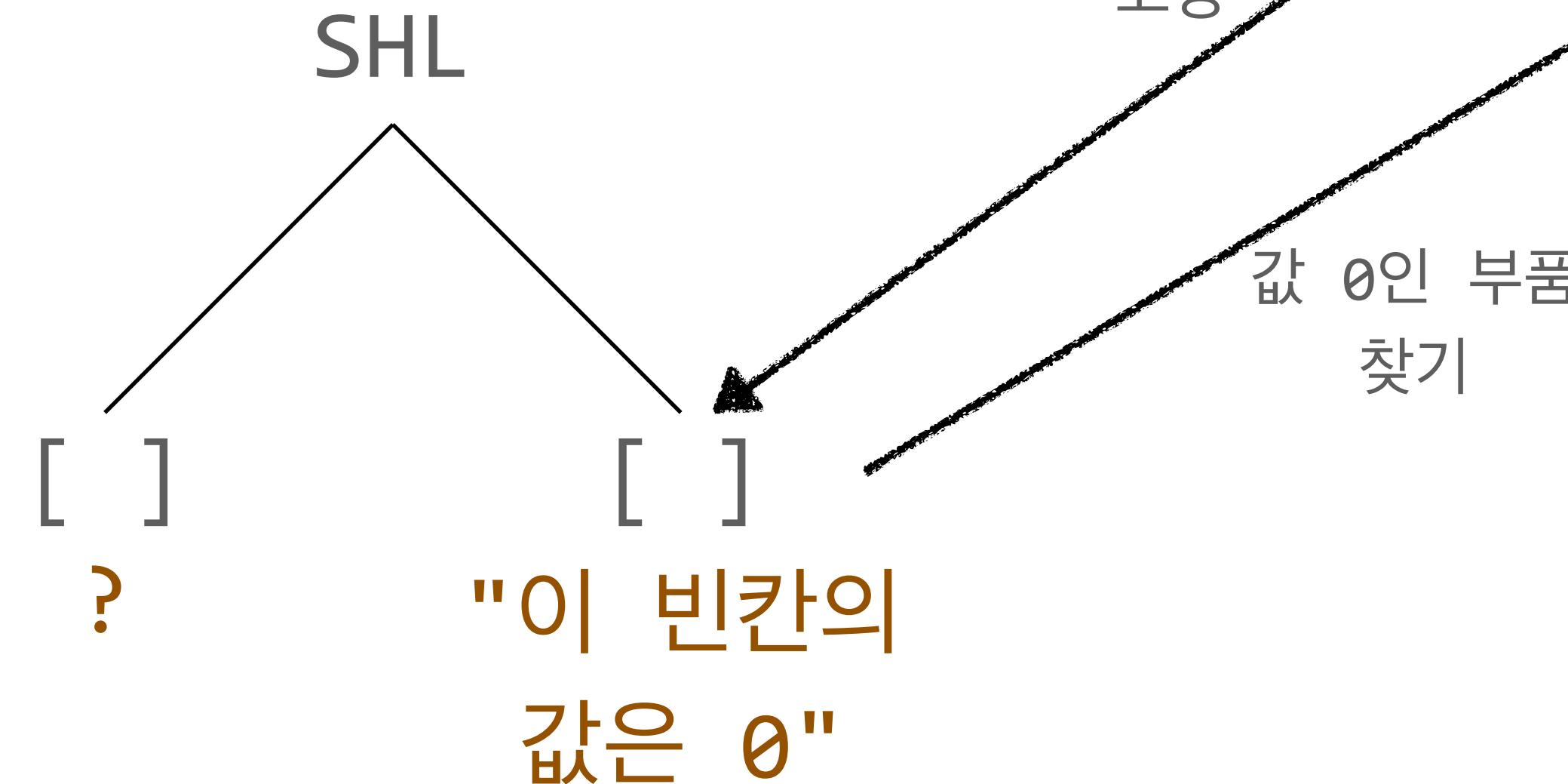
조건: $x = 3 \rightarrow f(x) = 11$

후보: $\text{SHL}([], []) = 11$

[부품 상자]

0:	0
1:	1
2:	$x - 1$
3:	x
4:	$x + 1$
6:	$x \ll 1$
9:	$x * x$

싹수 분석 결과



구체화 결과 적극 활용

값이 명확하면 부품 선택, 맞는 부품 없으면 짹수 없음

조건: $x = 3 \rightarrow f(x) = 11$

후보: $\text{SHL}([], 0) = 11$

[부품 상자]

0: 0

1: 1

2: $x - 1$

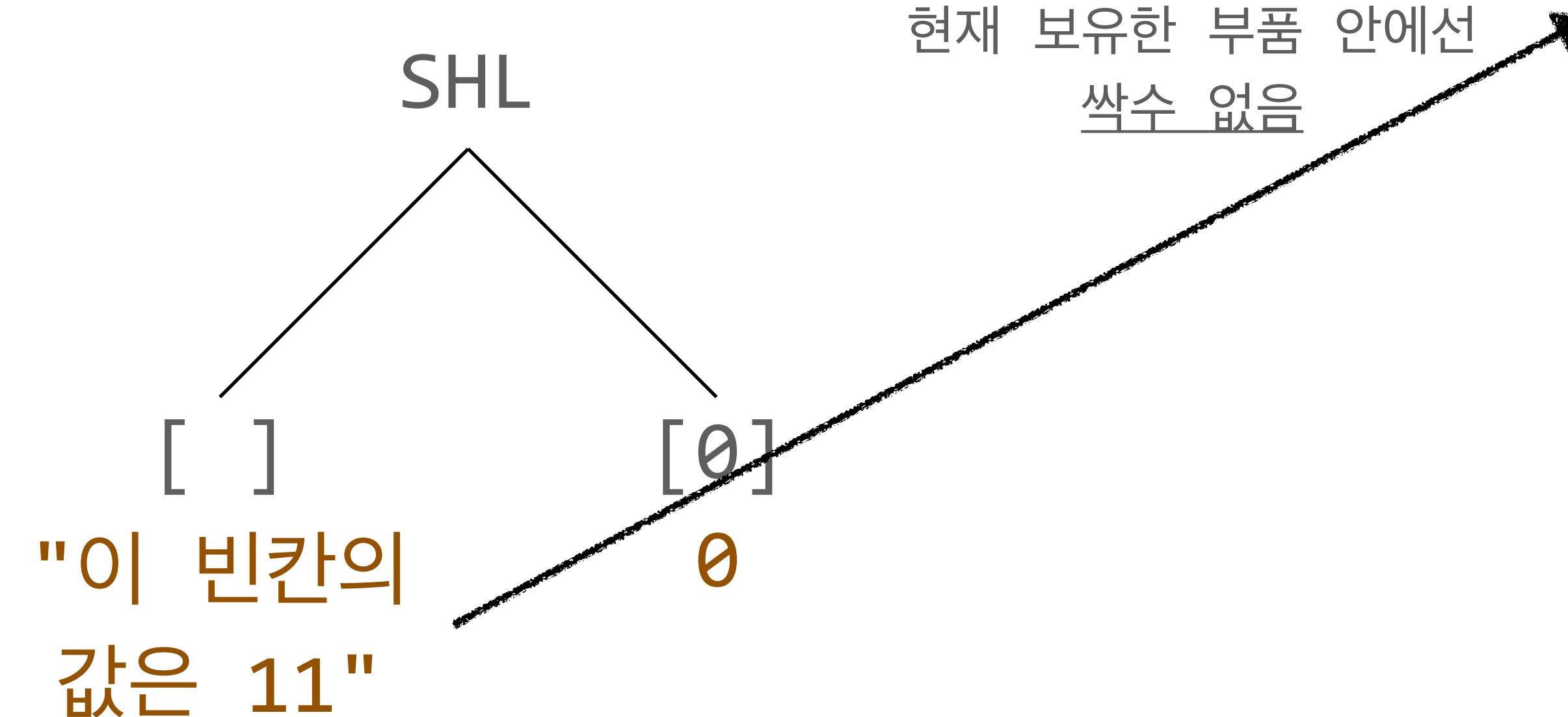
3: x

4: $x + 1$

6: $x \ll 1$

9: $x * x$

짬수 분석 결과



구체화 결과 적극 활용

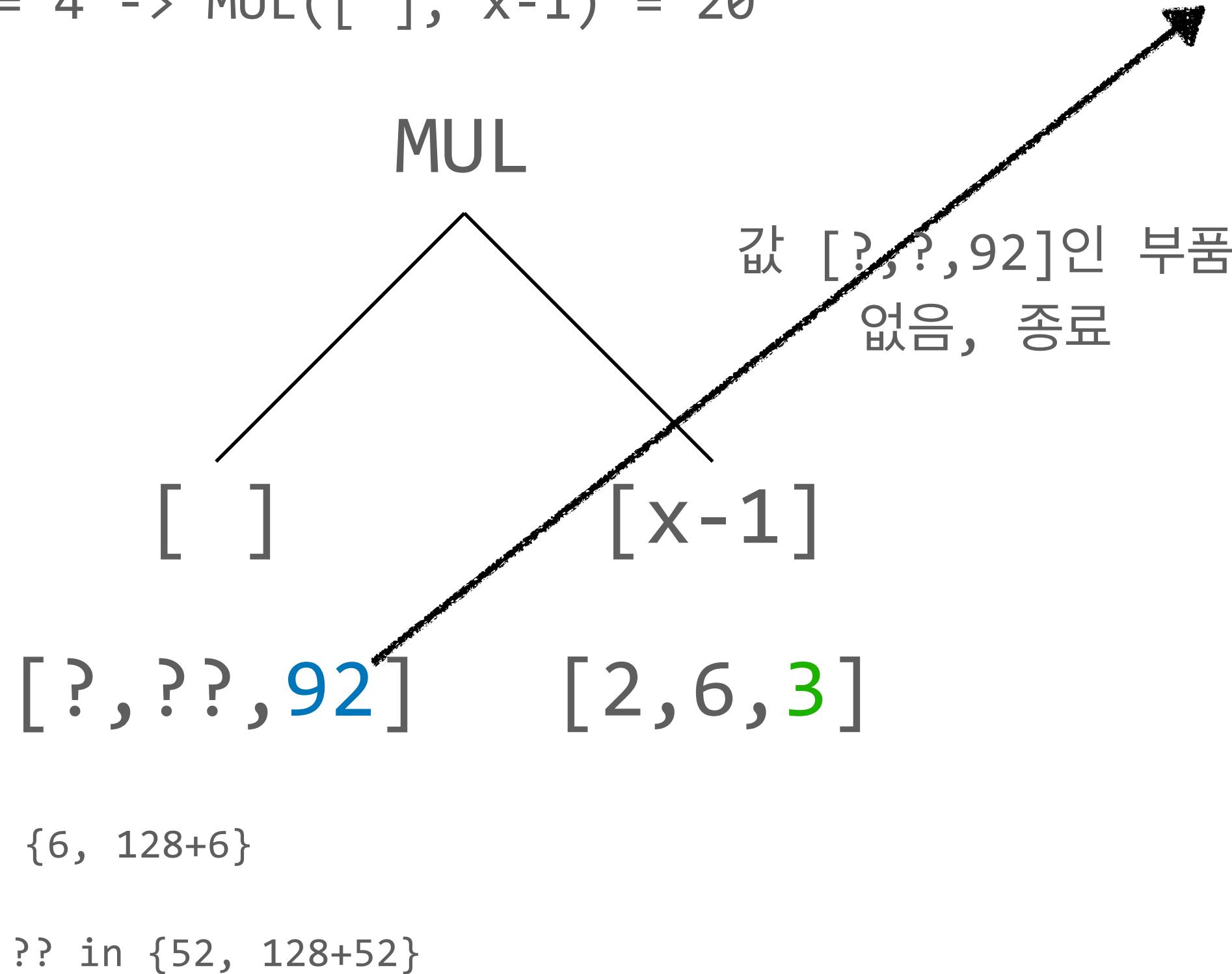
일부 입출력 쌍에 대해서라도

후보:

$$x = 3 \rightarrow \text{MUL}([], x-1) = 12$$

$$x = 7 \rightarrow \text{MUL}([], x-1) = 56$$

$$x = 4 \rightarrow \text{MUL}([], x-1) = 20$$



조건:

$$x = 3 \rightarrow f(x) = 12$$

$$x = 7 \rightarrow f(x) = 56$$

$$x = 4 \rightarrow f(x) = 20$$

x=3	x=7	x=4	부품
0	0	0	0
3	7	4	x
2	6	3	x-1
4	8	5	x+1
1	3	2	x>>1
6	14	8	x<<1
9	49	16	x*x
4	8	3	1+x
...			

구체화 결과 적극 활용

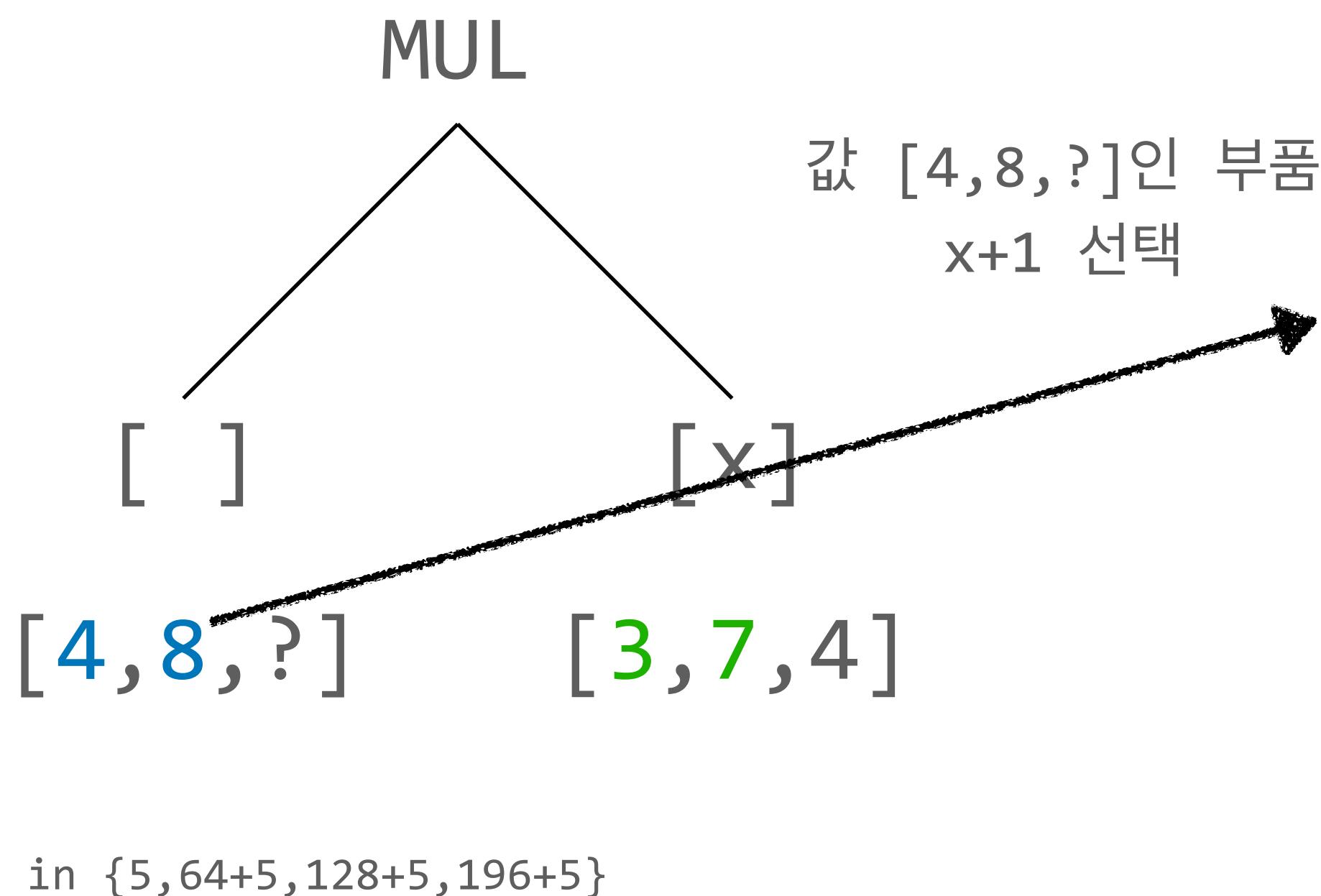
일부 입출력 쌍에 대해서라도

후보:

$$x = 3 \rightarrow \text{MUL}([], x) = 12$$

$$x = 7 \rightarrow \text{MUL}([], x) = 56$$

$$x = 4 \rightarrow \text{MUL}([], x) = 20$$



조건:

$$x = 3 \rightarrow f(x) = 12$$

$$x = 7 \rightarrow f(x) = 56$$

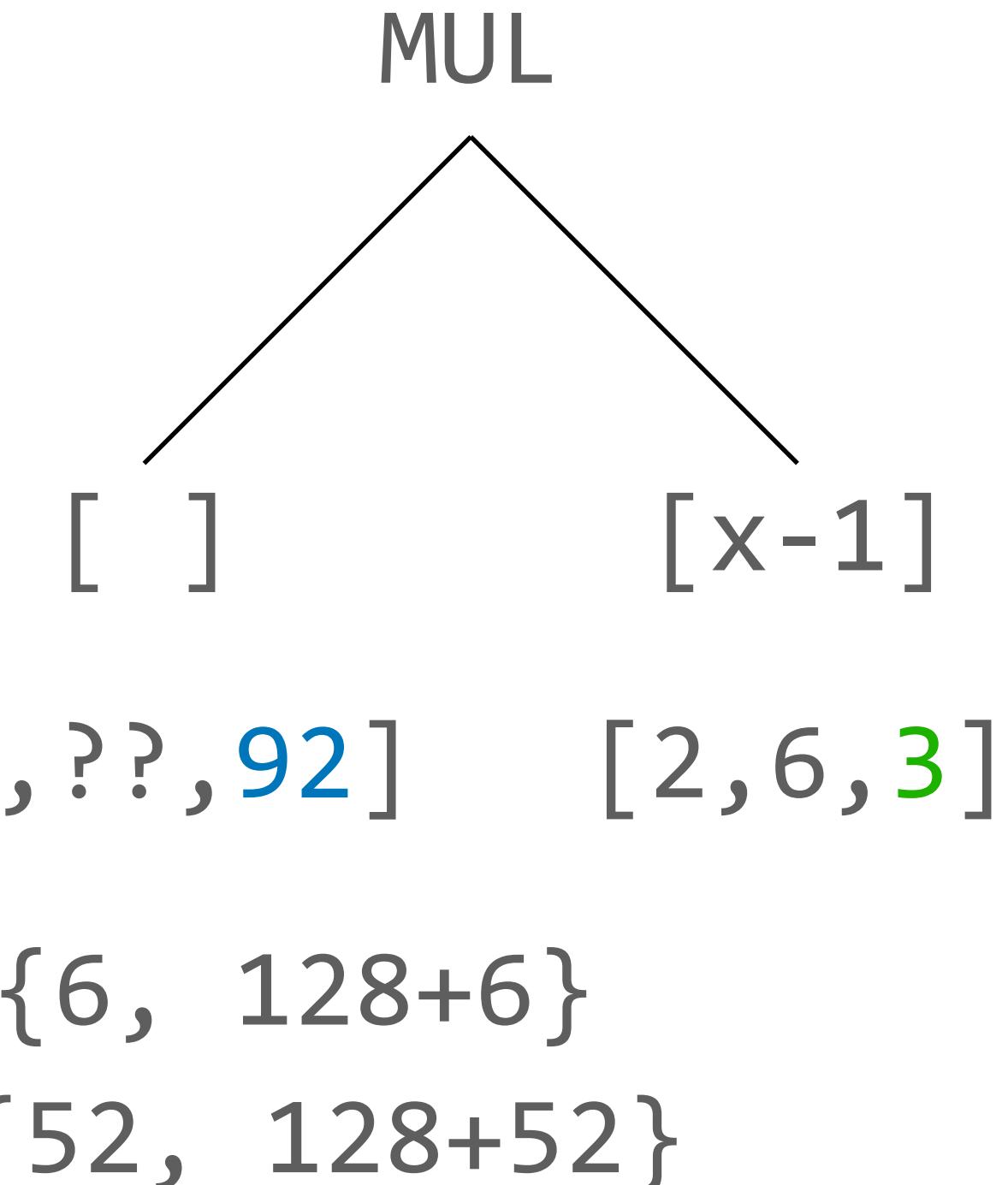
$$x = 4 \rightarrow f(x) = 20$$

x=3	x=7	x=4	부품
0	0	0	0
3	7	4	x
2	6	3	x-1
4	8	5	x+1
1	3	2	x>>1
6	14	8	x<<1
9	49	16	x*x
4	8	3	1+x
...			

구체화 결과 적극 활용

크기를 제한한 구체화

- 구체화 결과 값이 단 하나일때만 사용하면 아쉬움
 - 특히 이처럼 한 비트만 0 또는 1일 때
- 요약값을 구체화한 집합이 너무 크지 않으면 활용한다



구체화 결과 적극 활용

크기를 제한한 구체화

- 정해진 집합 크기만큼만 구체화를 시도 (현재 4로 설정)
 - 성공하면 그 정보를 바탕으로 빈칸에 넣어도 싹수가 있을 부품만 추릴 수 있음
 - 구체화 된 집합 크기가 너무 크면 포기하고 모든 부품 순회
 - 구체화 시도 크기는 적당해야: 지나치게 크게 설정하면 입출력 쌍 수에 대해 지수적으로 복잡도가 늘어남(4개까지 구체화 할 때 입출력이 5쌍이면 최대 1024가지 조합)
- 이 방법으로 싹수 없음을 더 일찍 확인할 수 있음

구체화 결과 적극 활용

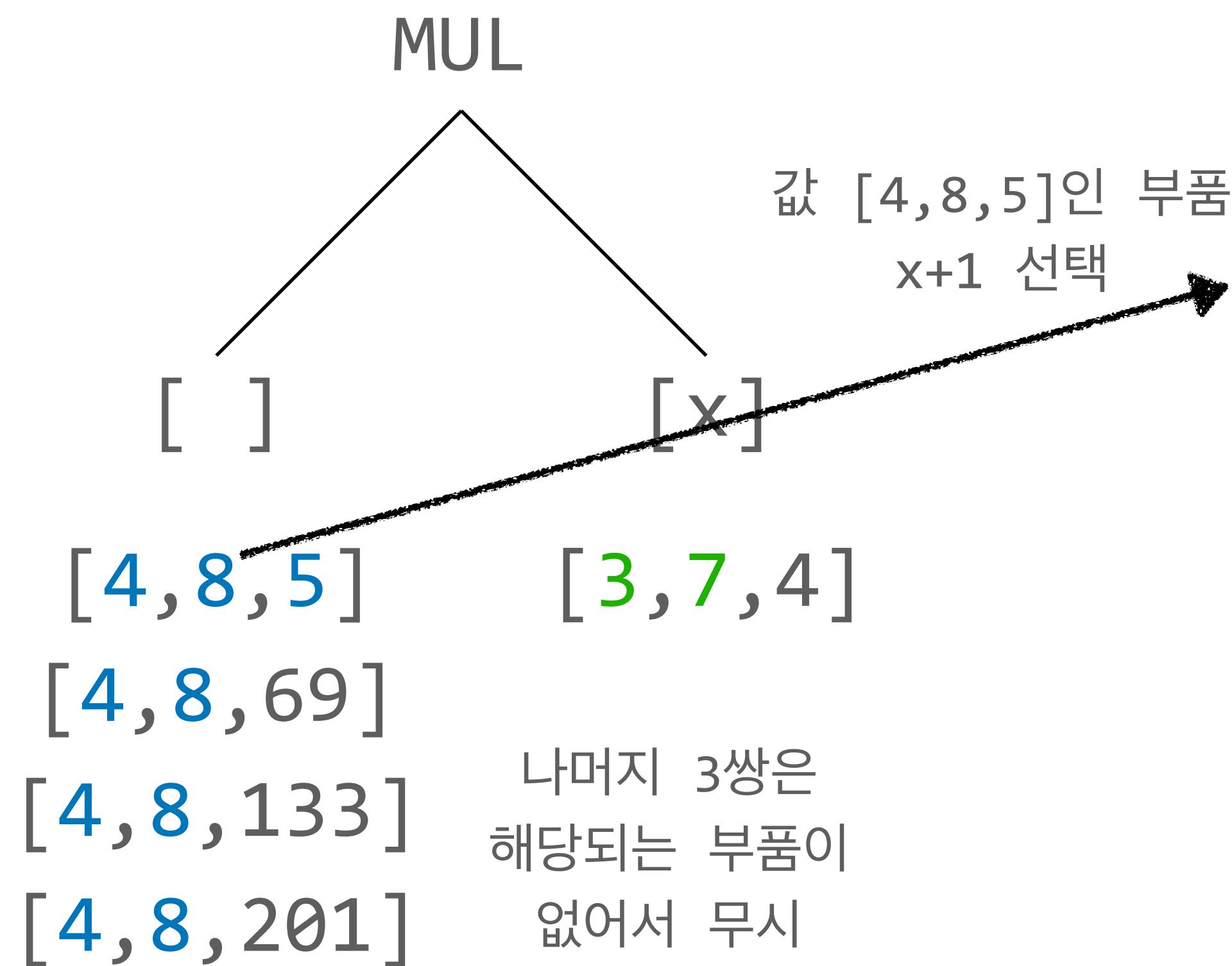
크기를 제한한 구체화

후보:

$x = 3 \rightarrow \text{MUL}([], x) = 12$

$x = 7 \rightarrow \text{MUL}([], x) = 56$

$x = 4 \rightarrow \text{MUL}([], x) = 20$



조건:

$x = 3 \rightarrow f(x) = 12$

$x = 7 \rightarrow f(x) = 56$

$x = 4 \rightarrow f(x) = 20$

x=3	x=7	x=4	부품
0	0	0	0
3	7	4	x
2	6	3	x-1
4	8	5	x+1
1	3	2	x>>1
6	14	8	x<<1
9	49	16	x*x
4	8	3	1+x
...			

남은 작업

- 합성 및 분석 대상 확장
 - 현재 64비트 벡터 도메인만 지원
 - 1비트(=Circuit 도메인) 및 일반화된 비트수로 구현 확장
- 적절한 Top-Down 시점 결정하는 방법 고안
 - 현재 고정값 사용
- 벤치마크 확장
 - SyGuS 공식 벤치마크 내 Circuit 도메인 문제 추가
 - (한양대학교에서) SuGuS Language로 표현된 비트 조작 역난독화 문제 확보 진행중