

씩수 분석으로 프로그램 합성 보조하기

윤용호

서울대학교 프로그래밍연구실 / (주)스패로우

2022-02-12, SIGPL 겨울학교

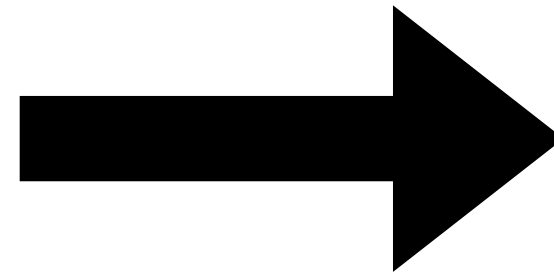
내용

- 배경지식: 프로그램 합성에 대한 짧은 설명
- 아이디어: 프로그램 합성에 정적 분석을 활용하는 방법
 - “씩수 분석”
- 썩수 분석의 효과와 분석 방법

프로그램 합성?

조건을 만족하는 프로그램을 자동으로 만들기

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

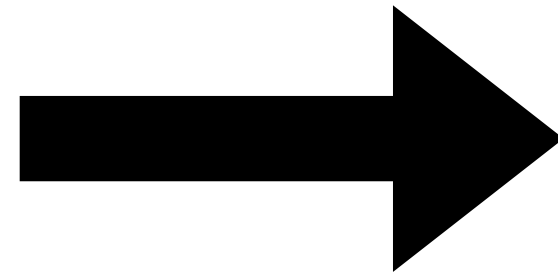


$$f(x) = ?$$

프로그램 합성?

조건을 만족하는 프로그램을 자동으로 만들기

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

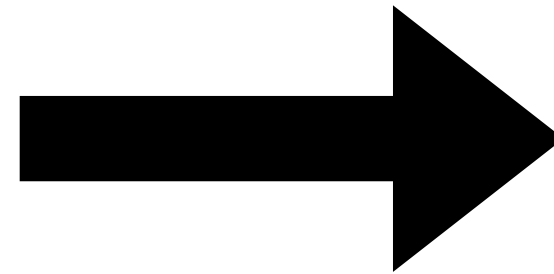


$$f(x) = 3 * x$$

프로그램 합성?

조건을 만족하는 프로그램을 자동으로 만들기

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

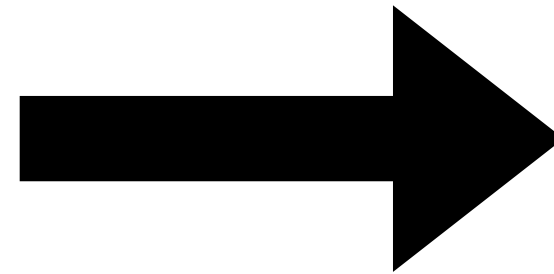


$$\begin{aligned}f(x) &= 3 * x ? \\f(x) &= x * x + 2 ?\end{aligned}$$

프로그램 합성?

조건을 만족하는 프로그램을 자동으로 만들기

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



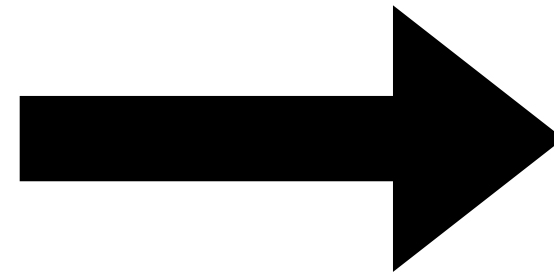
$$f(x) = 3 * x$$

~~$$f(x) = x * x + 2 ?$$~~

프로그램 합성?

조건을 만족하는 프로그램을 자동으로 만들기

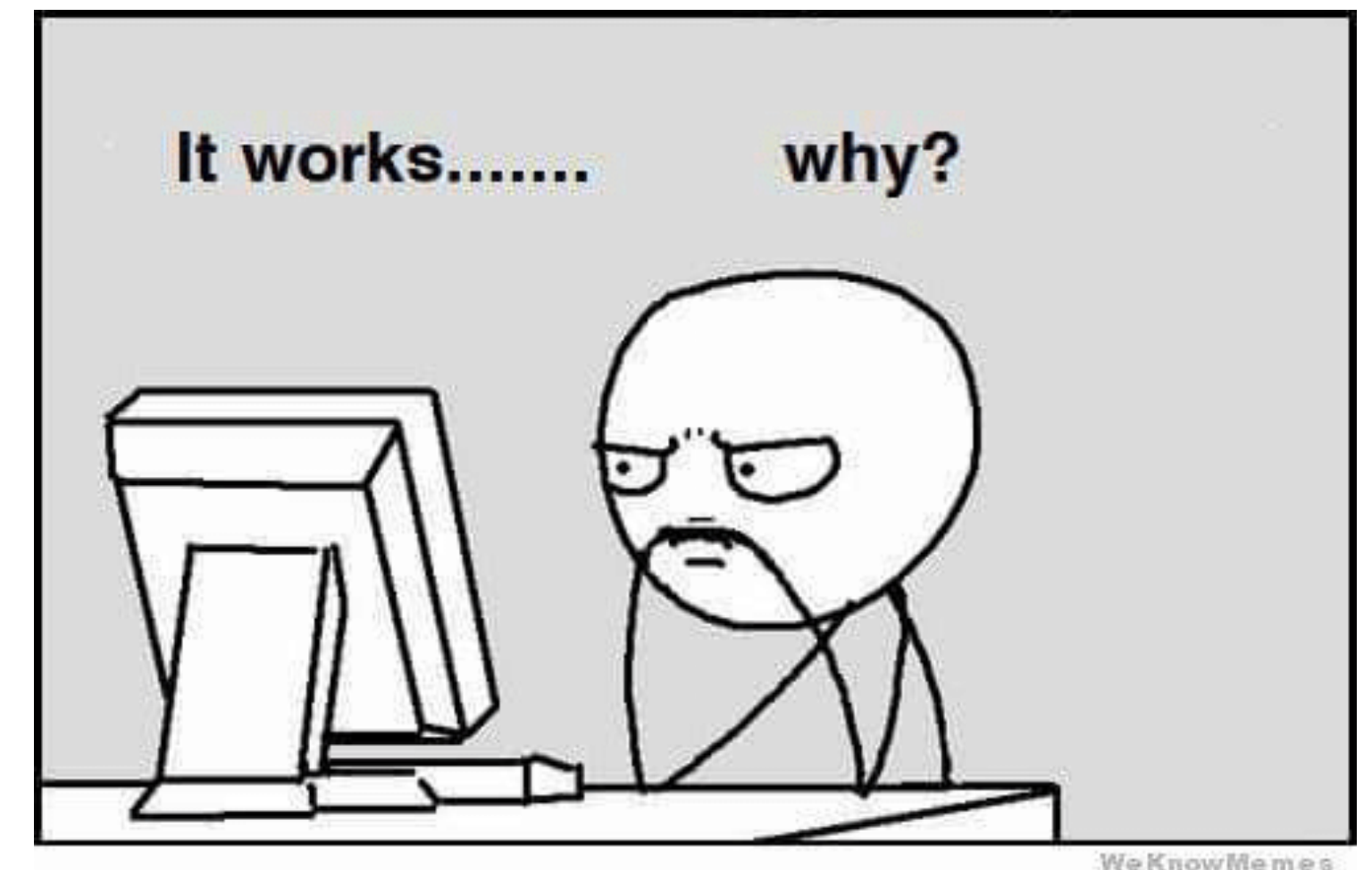
$$\begin{aligned}f(1) &= 3, \\f(2) &= 6, \\f(3) &= 11, \\f(x) &= ?\end{aligned}$$



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

입출력 예제 기반 프로그램 합성

- 합성 조건이 언제나 꼭 입출력 예제여야만 하는 것은 아니지만, 꽤 유용함
 - 입출력 예제 기반으로도 합성 문제를 꽤 많이 의도한대로 풀 수 있다
- 이게 왜 되지?
 - 평범한 사람들이 프로그램이 맞는지 확인할 때도 대개 몇 가지 테스트 케이스에 대해 잘 동작하는지 확인해봄
 - “오컴의 면도날”



기계적으로 프로그램 합성하기

작은 부품부터 조립해나가기(Bottom up)

너무 많은 부품을 조립해봐야한다

...

$1+1$

$1+x$

$1+(-x)$

$x+(-1)$

$x*x$

-1

$-x$

\emptyset


1

x

$$\begin{array}{lcl} E & \rightarrow & 0 \\ & | & 1 \\ & | & x \\ & | & E + E \\ & | & E * E \\ & | & -E \end{array}$$

기계적으로 프로그램 합성하기

상위 문법 규칙부터 나열해나가기(Top down)



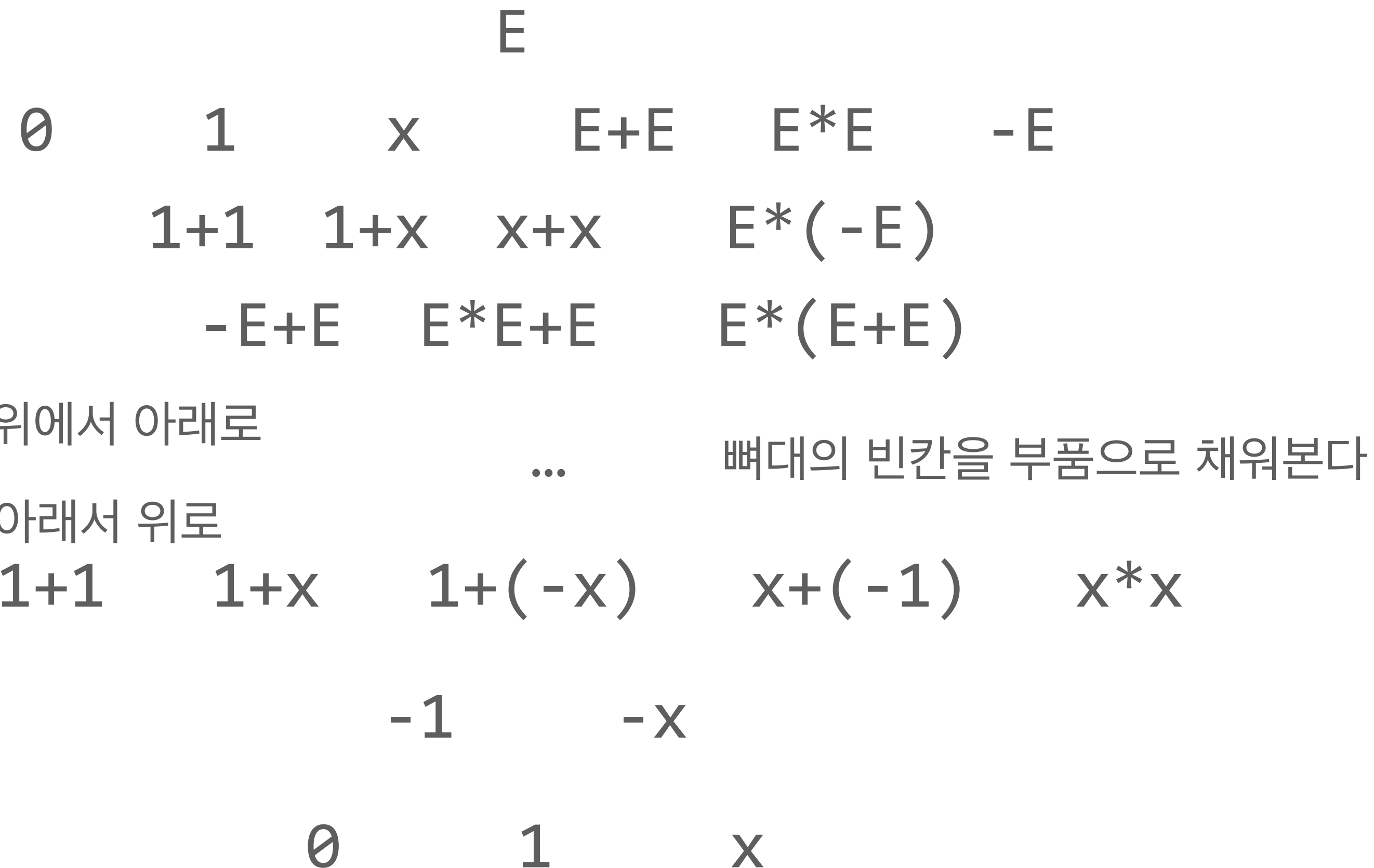
E					
\emptyset	1	x	$E + E$	$E * E$	$-E$
	1+1	1+x	x+x	$E * (-E)$	
	-E+E	$E * E + E$	$E * (E + E)$		
			...		

중간중간 불가능한 후보를 잘라낼 수 있지만
여전히 너무 많은 공간을 탐색해야한다

$$\begin{array}{lcl}
 E & \rightarrow & 0 \\
 & | & 1 \\
 & | & x \\
 & | & E + E \\
 & | & E * E \\
 & | & -E
 \end{array}$$

기계적으로 프로그램 합성하기

양방향으로 합성하기(Bidirectional)*



$$\begin{array}{c}
 E \rightarrow 0 \\
 | \\
 1 \\
 | \\
 x \\
 | \\
 E + E \\
 | \\
 E * E \\
 | \\
 -E
 \end{array}$$

기계적으로 프로그램 합성을 더 잘해볼 아이디어

정적 분석을 활용하여

- “삭수가 없는” 뼈대를 빨리 판단해서 버리면 유리함
 - “삭수가 없는” = 아직 미완성 프로그램이지만 빈 칸에 어떤 부품을 끼워봐도 조건을 만족하지 못할 것이 확실한
- 정적 분석으로 잘 할 수 있지 않을까?
 - 입력: 조건(입출력 쌍)과 미완성 프로그램
 - 출력: “아직 모름” or “삭수 없음”

썹수 분석의 현재 효과

아직은 만족스럽진 못하지만

- SyGuS Benchmarks 2018 Hacker’s Delight 문제 중 일부를 대상으로 파일럿 실험

문제	설정		비교군: 분석 없이 합성량		실험군: 썹수 분석 사용시 합성량						비교군 시간	실험군 시간		
	부품 크기	부품 수	미완성 수	완성 수	미완성 수 / %		썹수 없음 수 / %		완성 수 / %		합성 시간	분석 자체	분석 제외	전체
hd-09-d1	3	47	180560	7975896	154922	85.8	26551	17.14	5611989	70.4	10.3	11.4	7.7	19.1
hd-09-d1	4	47	180560	7975896	154922	85.8	26551	17.14	5611989	70.4	10.3	11.4	7.8	19.2
hd-09-d5	3	71	1223181	81132799	1109713	90.7	138970	11.62	64733373	79.8	106.1	69.2	89.5	158.7
hd-09-d5	4	255	1469169	353705597	1469169	100.0	42352	2.88	343369766	97.1	531.4	141.0	522.7	663.7
hd-13-d1	3	50	193455	9028233	162175	83.8	23013	14.19	6430534	71.2	12.2	14.4	9.5	23.9
hd-13-d5	3	78	879529	64031338	755777	85.9	167612	22.18	42474827	66.3	94.8	70.8	68.6	139.5
hd-13-d5	5	1753	13434	21524797	13434	100.0	3167	23.58	16490616	76.6	33.4	1.1	25.9	27.0
hd-14-d1	4	205	238961	48489143	238961	100.0	1835	0.77	48150432	99.3	109.1	43.0	110.6	153.5
hd-14-d5	3	120	122474	14040852	122474	100.0	1158	0.95	13915642	99.1	24.5	17.8	25.3	43.1
hd-14-d5	4	618	2057450	1254449998	2057450	100.0	5262	0.26	1251565155	99.8	2902.3	379.9	2927.9	3307.8
hd-15-d0	3	20	1042639	19350759	649676	62.3	11710	1.80	11807247	61.0	40.4	87.7	28.4	116.0
hd-15-d0	4	20	1042639	19350759	649676	62.3	11710	1.80	11807247	61.0	40.3	89.0	28.6	171.6

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

썩수 분석의 현재 효과 희망

- 썩수 없는 미완성 프로그램을 일찍 잘 버리면 완성 프로그램 절감 효과가 더 큼

문제	설정		비교군: 분석 없이 합성량		실험군: 썩수 분석 사용시 합성량						비교군 시간	실험군 시간		
	부품 크기	부품 수	미완성 수	완성 수	미완성 수 / %		썩수 없음 수 / %		완성 수 / %		합성 시간	분석 자체	분석 제외	전체
hd-09-d1	3	47	180560	7975896	154922	85.8	26551	17.14	5611989	70.4	10.3	11.4	7.7	19.1
hd-09-d1	4	47	180560	7975896	154922	85.8	26551	17.14	5611989	70.4	10.3	11.4	7.8	19.2
hd-09-d5	3	71	1223181	81132799	1109713	90.7	138970	11.62	64733373	79.8	106.1	69.2	89.5	158.7
hd-09-d5	4	255	1469169	353705597	1469169	100.0	42352	2.88	343369766	97.1	531.4	141.0	522.7	663.7
hd-13-d1	3	50	193455	9028233	162175	83.8	23013	14.19	6430534	71.2	12.2	14.4	9.5	23.9
hd-13-d5	3	78	879529	64031338	755777	85.9	167612	22.18	42474827	66.3	94.8	70.8	68.6	139.5
hd-13-d5	5	1753	13434	21524797	13434	100.0	3167	23.58	16490616	76.6	33.4	1.1	25.9	27.0
hd-14-d1	4	205	238961	48489143	238961	100.0	1835	0.77	48150432	99.3	109.1	43.0	110.6	153.5
hd-14-d5	3	120	122474	14040852	122474	100.0	1158	0.95	13915642	99.1	24.5	17.8	25.3	43.1
hd-14-d5	4	618	2057450	1254449998	2057450	100.0	5262	0.26	1251565155	99.8	2902.3	379.9	2927.9	3307.8
hd-15-d0	3	20	1042639	19350759	649676	62.3	11710	1.80	11807247	61.0	40.4	87.7	28.4	116.0
hd-15-d0	4	20	1042639	19350759	649676	62.3	11710	1.80	11807247	61.0	40.3	89.0	28.6	171.6

썩수 분석의 현재 효과 희망

- 구멍이 하나뿐인 미완성 프로그램이라도 부품이 아주 많을땐 썩수 분석의 효과 좋음

문제	설정		비교군: 분석 없이 합성량		실험군: 썩수 분석 사용시 합성량						비교군 시간	실험군 시간		
	부품 크기	부품 수	미완성 수	완성 수	미완성 수 / %		썩수 없음 수 / %		완성 수 / %		합성 시간	분석 자체	분석 제외	전체
hd-09-d1	3	47	180560	7975896	154922	85.8	26551	17.14	5611989	70.4	10.3	11.4	7.7	19.1
hd-09-d1	4	47	180560	7975896	154922	85.8	26551	17.14	5611989	70.4	10.3	11.4	7.8	19.2
hd-09-d5	3	71	1223181	81132799	1109713	90.7	138970	11.62	64733373	79.8	106.1	69.2	89.5	158.7
hd-09-d5	4	255	1469169	353705597	1469169	100.0	42352	2.88	343369766	97.1	531.4	141.0	522.7	663.7
hd-13-d1	3	50	193455	9028233	162175	83.8	23013	14.19	6430534	71.2	12.2	14.4	9.5	23.9
hd-13-d5	3	78	879529	64031338	755777	85.9	167612	22.18	42474827	66.3	94.8	70.8	68.6	139.5
hd-13-d5	5	1753	13434	21524797	13434	100.0	3167	23.58	16490616	76.6	33.4	1.1	25.9	27.0
hd-14-d1	4	205	238961	48489143	238961	100.0	1835	0.77	48150432	99.3	109.1	43.0	110.6	153.5
hd-14-d5	3	120	122474	14040852	122474	100.0	1158	0.95	13915642	99.1	24.5	17.8	25.3	43.1
hd-14-d5	4	618	2057450	1254449998	2057450	100.0	5262	0.26	1251565155	99.8	2902.3	379.9	2927.9	3307.8
hd-15-d0	3	20	1042639	19350759	649676	62.3	11710	1.80	11807247	61.0	40.4	87.7	28.4	116.0
hd-15-d0	4	20	1042639	19350759	649676	62.3	11710	1.80	11807247	61.0	40.3	89.0	28.6	171.6

썩수 분석의 현재 효과 장애물

- 아직 대체로 배보다 배꼽이 커서 분석 효율화 고민중

문제	설정		비교군: 분석 없이 합성량		실험군: 썩수 분석 사용시 합성량						비교군 시간	실험군 시간		
	부품 크기	부품 수	미완성 수	완성 수	미완성 수 / %		썩수 없음 수 / %		완성 수 / %		합성 시간	분석 자체	분석 제외	전체
hd-09-d1	3	47	180560	7975896	154922	85.8	26551	17.14	5611989	70.4	10.3	11.4	7.7	19.1
hd-09-d1	4	47	180560	7975896	154922	85.8	26551	17.14	5611989	70.4	10.3	11.4	7.8	19.2
hd-09-d5	3	71	1223181	81132799	1109713	90.7	138970	11.62	64733373	79.8	106.1	69.2	89.5	158.7
hd-09-d5	4	255	1469169	353705597	1469169	100.0	42352	2.88	343369766	97.1	531.4	141.0	522.7	663.7
hd-13-d1	3	50	193455	9028233	162175	83.8	23013	14.19	6430534	71.2	12.2	14.4	9.5	23.9
hd-13-d5	3	78	879529	64031338	755777	85.9	167612	22.18	42474827	66.3	94.8	70.8	68.6	139.5
hd-13-d5	5	1753	13434	21524797	13434	100.0	3167	23.58	16490616	76.6	33.4	1.1	25.9	27.0
hd-14-d1	4	205	238961	48489143	238961	100.0	1835	0.77	48150432	99.3	109.1	43.0	110.6	153.5
hd-14-d5	3	120	122474	14040852	122474	100.0	1158	0.95	13915642	99.1	24.5	17.8	25.3	43.1
hd-14-d5	4	618	2057450	1254449998	2057450	100.0	5262	0.26	1251565155	99.8	2902.3	379.9	2927.9	3307.8
hd-15-d0	3	20	1042639	19350759	649676	62.3	11710	1.80	11807247	61.0	40.4	87.7	28.4	116.0
hd-15-d0	4	20	1042639	19350759	649676	62.3	11710	1.80	11807247	61.0	40.3	89.0	28.6	171.6

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:

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

입출력 예제:

$f(b_11001111) = b_00001111$

$f(b_01010111) = b_00000111$

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

후보 미완성 프로그램:

$([] | x) \gg 1$

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:

$$((x + 1) ^ x) >> 1$$

정방향 분석($x=b_11001111$):

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

역방향 분석($x=b_11001111$):

```
([ ] | x) >> 1 : T11TT111  ^ 00001111 = 0..01111 <- 짝수 없음
[ ] | x       : 11TT1111
x             : 11001111
[ ]           : TTTTTTTT
```

입출력 예제:

$f(b_11001111) = b_00001111$
 $f(b_01010111) = b_00000111$

후보 미완성 프로그램:

$([] | x) >> 1$

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:

$$((x + 1) ^ x) >> 1$$

정방향 분석($x=b_11001111$):

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

입출력 예제:

$f(b_11001111) = b_00001111$
 $f(b_01010111) = b_00000111$

후보 미완성 프로그램:

([] & x) >> 1

역방향 분석($x=b_11001111$):

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

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:

$$((x + 1) ^ x) >> 1$$

정방향 분석($x=b_11001111$):

입출력 예제:
 $f(b_11001111) = b_00001111$
 $f(b_01010111) = b_00000111$

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

후보 미완성 프로그램:
([] / x) >> 1

역방향 분석($x=b_11001111$):

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

합성에 짝수 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:
 $((x + 1) \wedge x) \gg 1$

입출력 예제:
f(b_11001111) = b_00001111
f(b_01010111) = b_00000111

후보 미완성 프로그램:
 $([] \wedge x) \gg 1$

정방향 분석(x=b_11001111):

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

역방향 분석(x=b_11001111): 아직 모름 => 정답으로 가보자

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

정적 분석을 기반으로 좋은 싹수 분석을 만들려면

- 미완성 프로그램을 분석할 수 있어야 (당연히)
- 역방향 분석(Backward Analysis)이 필수이자 핵심: 정방향 분석은 미완성 부분에 의해 Top이 많이 발생
- 최대한 빠르고 정확한 분석으로 싹수 없는 후보를 쳐내야
 - **최대한 정확한 분석: 도메인과 (역방향)요약 연산을 정교하게 정의해야**
 - 최대한 빠른 분석: 온갖 부품을 마구 끼워서 직접 계산해보는 것보다 싹수 분석이 더 비싸면 손해
- 희망: 분석 정확도를 높이기엔 편리한 분야 특성
 - 현대의 기술로 풀 수 있는 합성 문제는 정답 프로그램의 크기가 비교적 작음
 - 아직은 합성 대상 언어 문법에 반복문이 없음

미완성 프로그램 정적 분석 대상 언어

- SyGuS (Syntax-Guided Synthesis) 에서 지원하는 합성 대상 언어 중 BitVector 관련 부분
 - SyGuS는 합성 분야의 사실상 표준
 - BitVector 에서 잘 풀리면 String 등으로 확장 예정
- 값: 64비트 정수(64개의 비트 나열)
- 연산: 널리 쓰이는 산술 및 비트 연산
- `[]`: 프로그램의 아직 완성되지 않은 부분
- 모든 변수는 입력값 (let 변수 선언 없음)

$$\begin{array}{lcl} E & \rightarrow & 0 \mid 1 \mid -1 \\ & & \mid x \\ & & \mid \diamond E \\ & & \mid E \circ E \\ & & \mid \text{if } B \ E \ E \\ & & \mid [] \\ B & \rightarrow & E = E \\ & & \mid \text{not } B \\ & & \mid B \text{ and } B \\ & & \mid B \text{ or } B \\ & & \mid [] \\ \diamond & \rightarrow & \sim \mid - \\ \circ & \rightarrow & \wedge \mid \vee \mid \oplus \mid \ll \mid \gg \mid \ggg \\ & & \mid + \mid - \mid * \mid / \mid \% \mid /_s \mid \%_s \end{array}$$

상호 보완하는 복합 도메인

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: {b_11111100, b_11111110} 을 요약하면 ([252, 254], [-4, -2], 111111T0)
- 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, T0001T1T) 이었다면 ([10, 143], [-118, 15], T0001T1T) 로 구간 값을 살릴 수 있다

상호 보완 함수

Reduction Operator

- 비트 나열 도메인
 - 부호 없는 구간 도메인에서
 - 양끝값을 비트로 표현한 후 최상위 위치부터 순서대로 동일한 비트들은 그 값으로, 처음으로 달라지는 위치부터 이후 더 낮은 자리는 모두 1 으로
 - 부호 있는 구간 도메인에서
 - 양끝값의 부호가 같으면 부호 없는 구간 도메인과 같은 방법
 - 양끝값 부호가 다르면 활용 포기 (구간에 111...1 과 000...0 을 포함하게 되므로)

상호 보완 함수

Reduction Operator

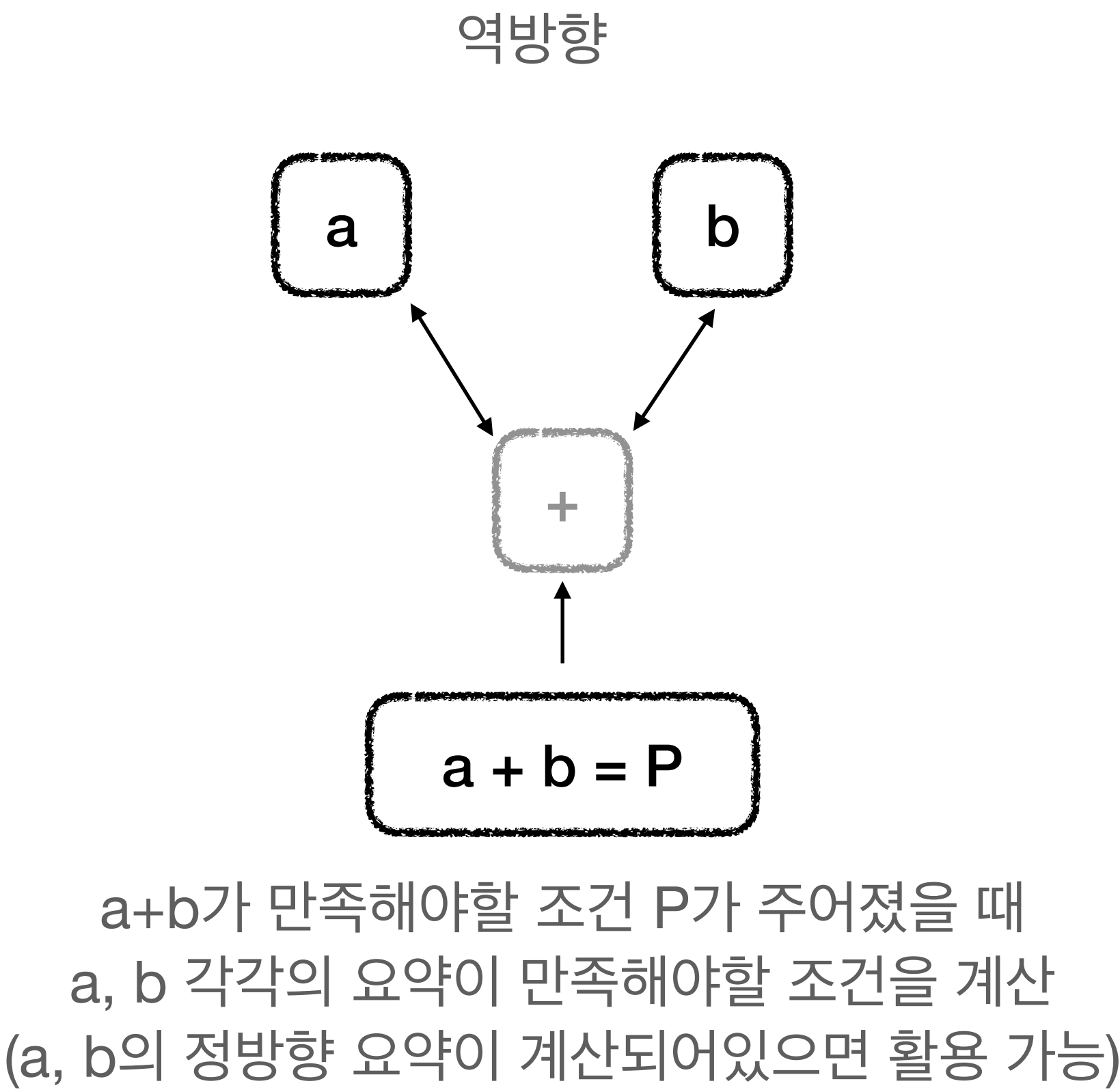
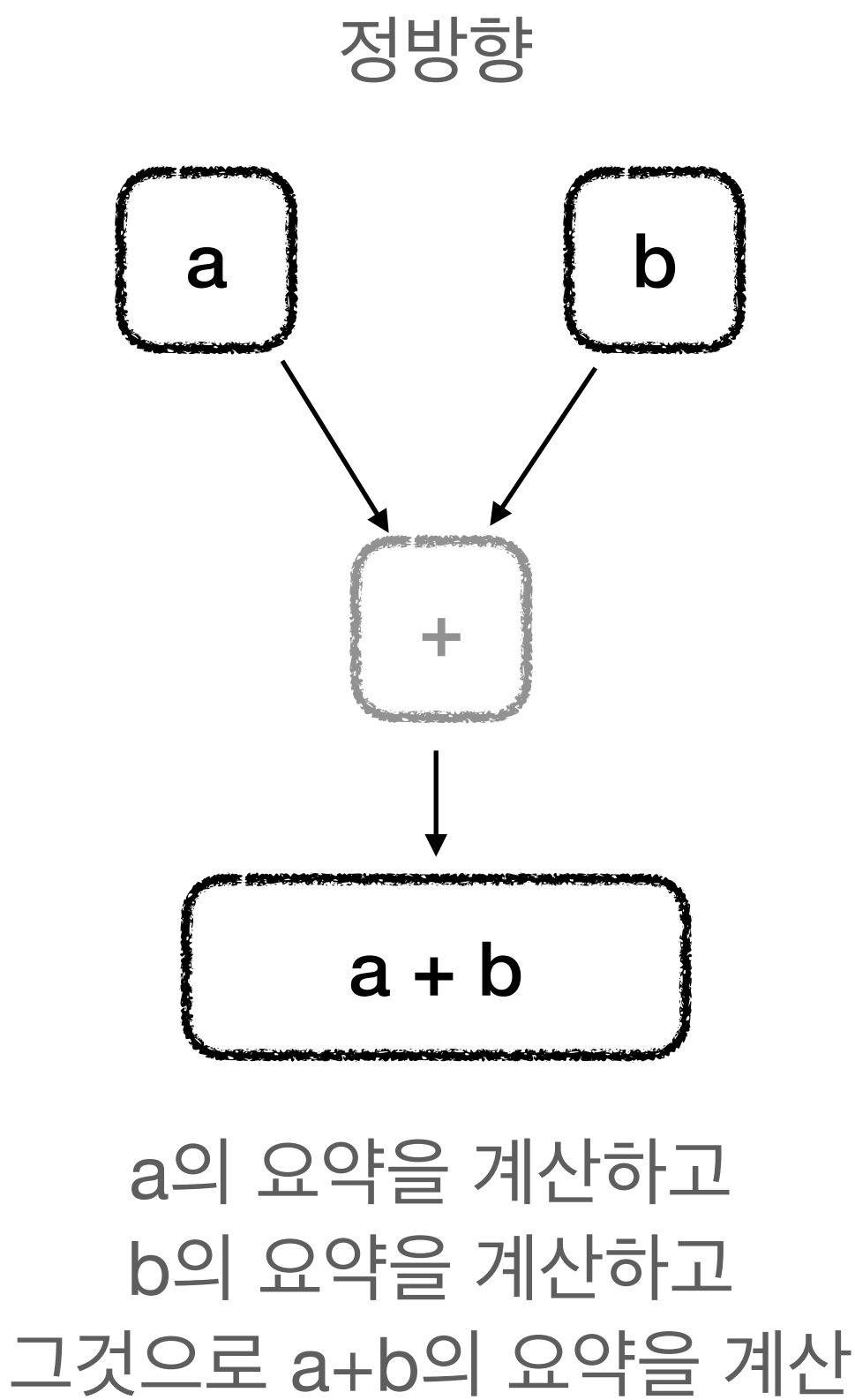
- 부호 없는 구간 도메인
 - 비트 나열 도메인에서: T 비트를 모두 0으로 만들면 최소, 1로 만들면 최대
- 부호 있는 구간 도메인에서
 - 양끝값 부호가 동일하면 양끝값의 비트 표현을 부호없이 해석해서 사용
 - 양끝값 부호가 다르면 활용 포기 (구간에 111...1 과 000...0 을 포함하게 되므로)

상호 보완 함수

Reduction Operator

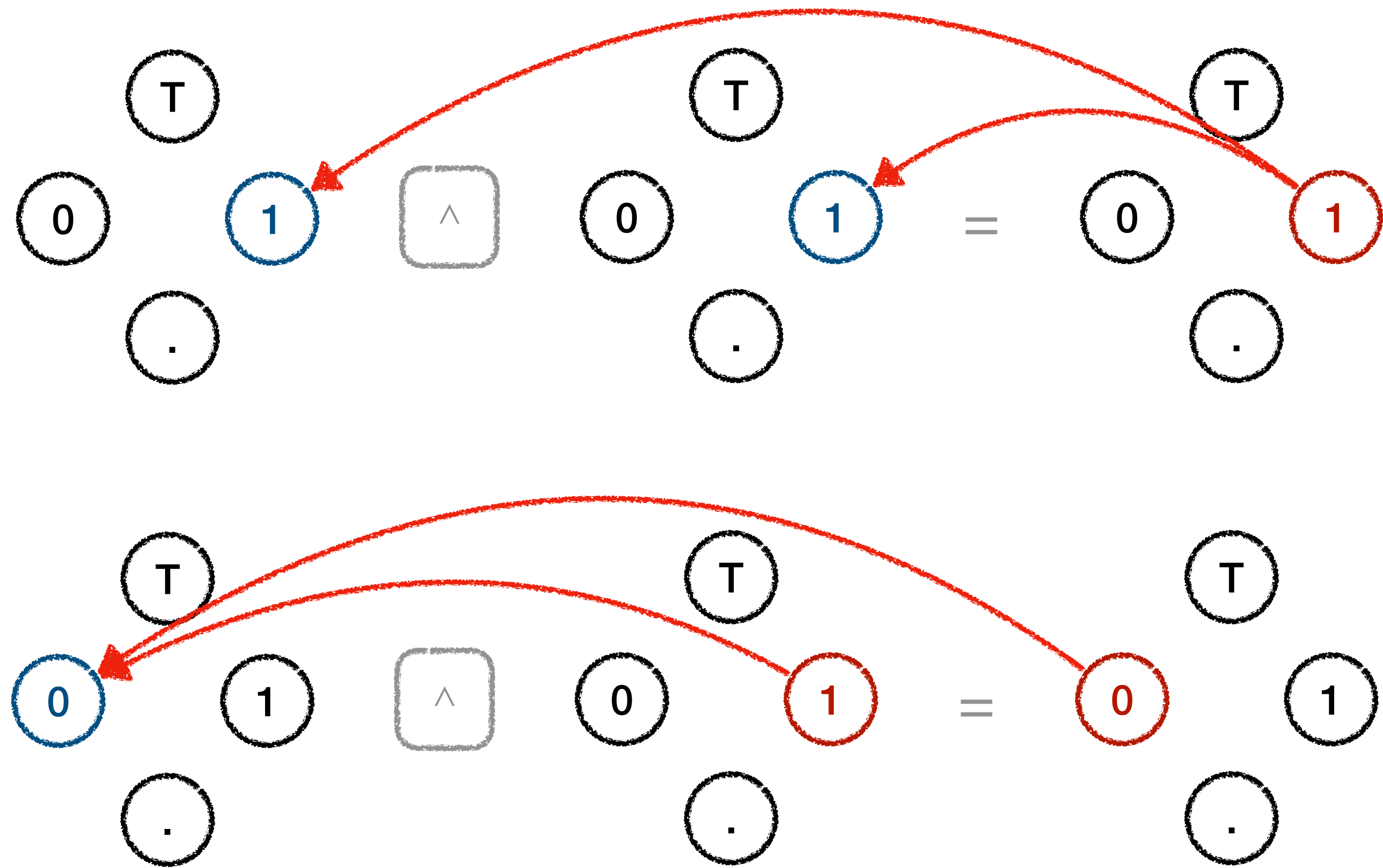
- 부호 있는 구간 도메인
 - 비트 나열 도메인에서
 - 최상위 비트가 0이나 1이면 부호 없는 구간 도메인과 똑같이
 - 최상위 비트가 T이면
 - 최소: 최상위비트는 1, 나머지 T 비트는 0으로 채운 것
 - 최대: 최상위 비트는 1, 나머지 T 비트는 1로 채운 것
- 부호 없는 구간 도메인에서
 - 양끝값을 비트로 표현했을 때 최상위 비트가 동일하면: 양끝값 비트 표현을 부호 있는 것으로 해석하여 사용
 - 최상위 비트가 다르면 활용 포기 (구간에 100....0 과 011...1 을 포함하게 되므로)

정방향/역방향 분석



역방향 분석 예: and(\wedge)

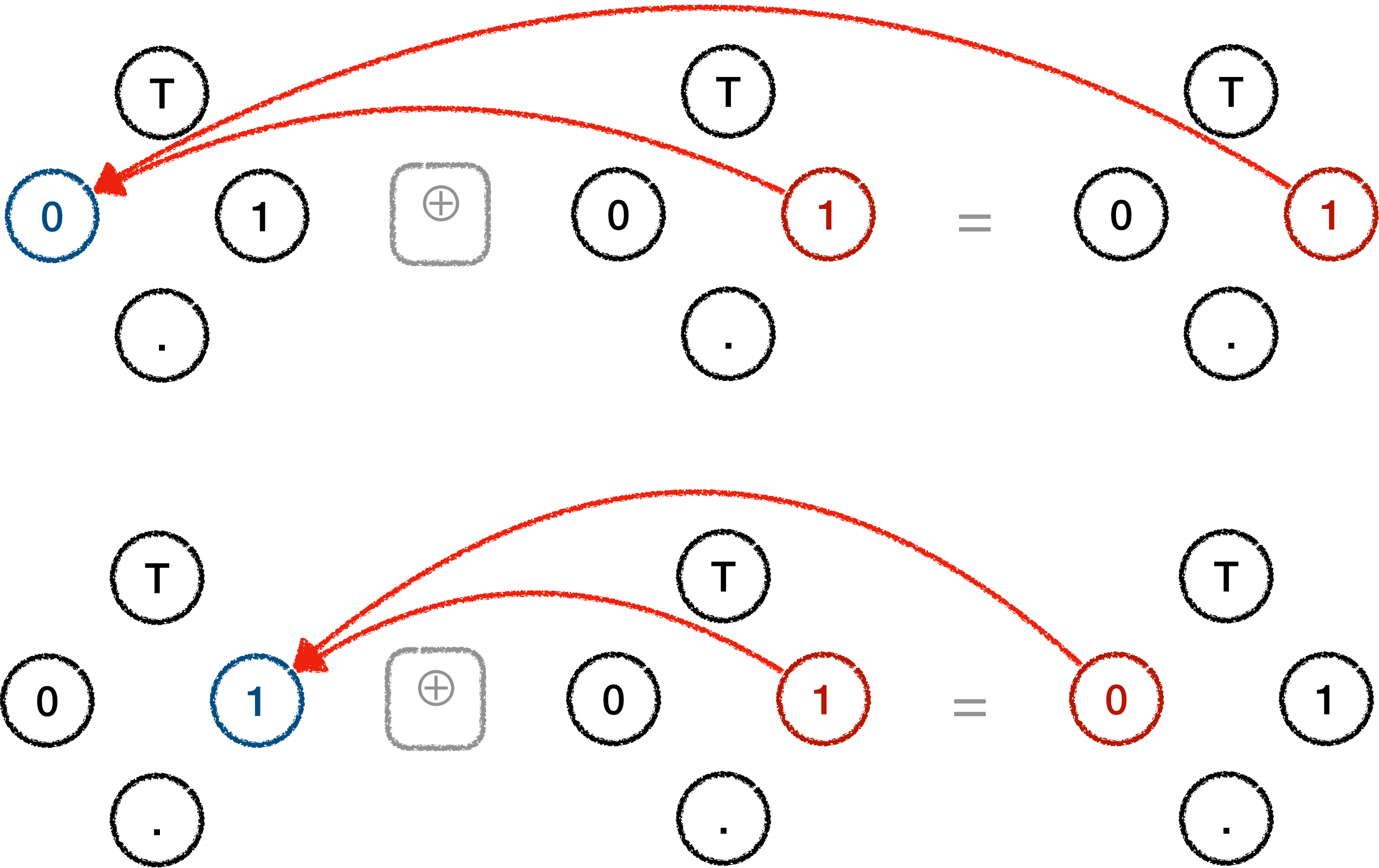
각 비트 자리마다



정방향 결과	역방향 결과
TTTTTTTTT	T10T0T1T
TT10101T	T110101T
	T100T01T

역방향 분석 예: xor(\oplus)

각 비트 자리마다



정방향 결과

역방향 결과

T0T0TT1T

T0**1**0TT1T

TT10100T

T**1**10100T

T100T01T

역방향 분석 예: lshr(>>>)

- $a \ggg b = P$ 가 주어졌을 때
- P의 1이 아닌 연속된 상위 비트 갯수로 b의 최댓값 제한
 - b가 그보다 크다면 그 위치에 1이 올 수 없음
- b를 구체화한 값들 만큼씩 P를 왼쪽으로 shift 한 결과들을 모두 뭉친 결과로 a의 비트 나열을 보완

정방향 결과

TTTTTTTT

[2,4]

00TTTTTT

역방향 결과

T11TTTTT

[2,3]

000111T0

$a \gg 2 = P$ 0111T0TT

$a \gg 3 = P$ 111T0TTT

앞으로의 방향

- 분석 효율화
 - 비슷비슷한 분석을 수백~수천만 번 반복하므로 재활용 가능성
 - 힐끔 보고 자신 있는 프로그램만 분석하기
- 더 다양한 상황에서 효과 확인
 - 부품 크기와 갯수를 더욱 공격적으로 늘려볼 것

감사합니다