싹수 분석을 활용하여 비트 벡터 프로그램을 양방향 합성하기

합성을 위한 분석, 분석에 맞는 합성

윤용호, 2022-05-27

성능

SyGuS Competition 벤치마크 해커의 기쁨(Hacker's delight) 문제*에 대해

* 총 44문제 중 hd-01-d1 ~ hd-08-d5 (23문제)는 부품 크기 7까지
단순 나열하는 중에도 1초 안에 정답이 발견되어 누구나 쉽게 풀 수 있어서 생략

합성 문제	초기 부품 크기	합성 시간 (초)
hd-09-* hd-13-* hd-14-d0 hd-14-d1 hd-15-d0 hd-15-d1 hd-17-d0	7	< 1
hd-14-d5	7	14.45
hd-15-d5	7	16.95
hd-17-d5	7	8.41
hd-19-d0	9	< 1
hd-19-d1	9	92초
hd-20-d0	9	< 3
hd-20-d1	9	< 8

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

참고: 3개월 전 SIGPL 발표 시점과 비교

상전벽해

문제	설정		비교군: 분석 없이 합성량		실험군: 싹수 분석 사용시 합성량						비교군 시간	실험군 시간		
	부품 크기	부품 수	미완성 수	완성 수	미완성 수 / %		싹수 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

개선 과정: 합성 방법 개선과 싹수 분석 개선의 선순환

- 합성 알고리즘 조율
  - 싹수 분석의 효과를 최대한 누릴 수 있도록
  - 합성을 멍청하게 하면: 싹수 분석의 효과가 과장되거나 무시되거나

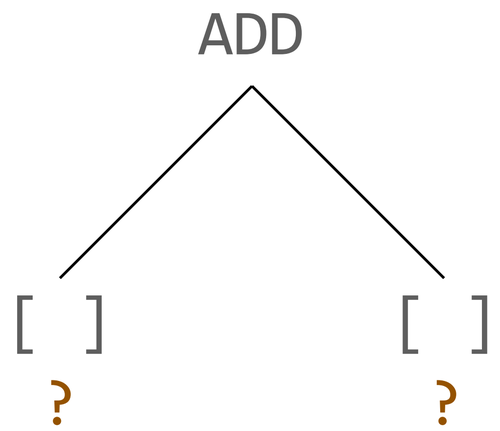
- 싹수 분석은 정교하게
  - 분석이 지나치게 느리지 않다면 결과가 정교할 수록 유리함
  - 기왕이면 합성 알고리즘에 도움이 되는 방향으로

합성 효율 개선

빈칸에 들어갈 값이 명확하면 즉시 부품 선택

조건: x = 3 -> f(x) = 11

후보: ADD([ ], [ ]) = 11



싹수 분석 결과

[부품 상자]

0: 0

1: 1

2: x - 1

3: x

4: x + 1

6: x << 1

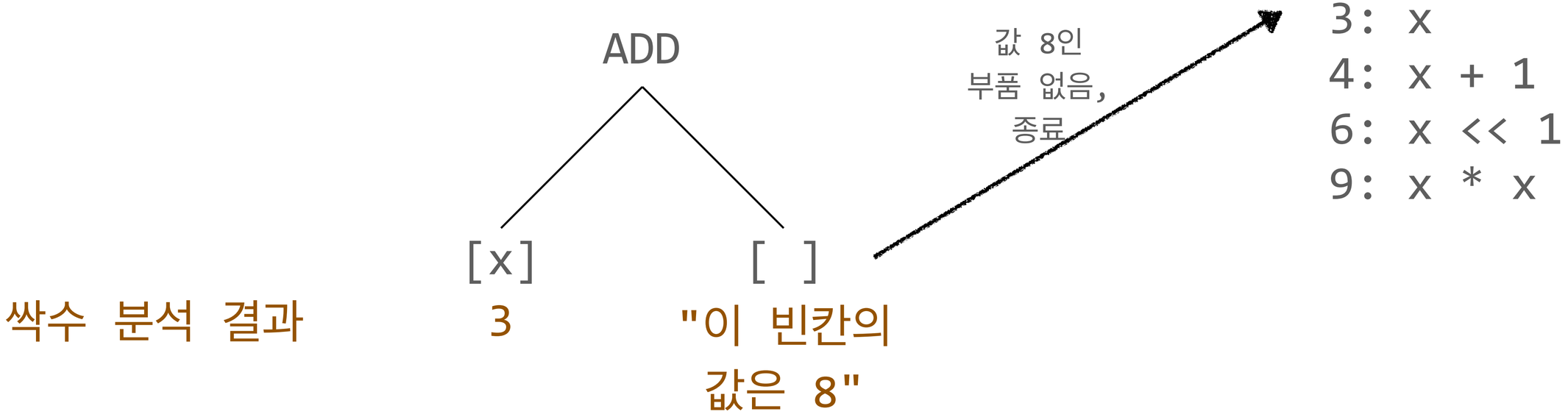
9: x * x

합성 효율 개선

빈칸에 들어갈 값이 명확하면 즉시 부품 선택

조건: x = 3 -> f(x) = 11

후보: ADD([x], [ ]) = 11



싹수 분석 결과

[부품 상자]

0: 0

1: 1

2: x - 1

3: x

4: x + 1

6: x << 1

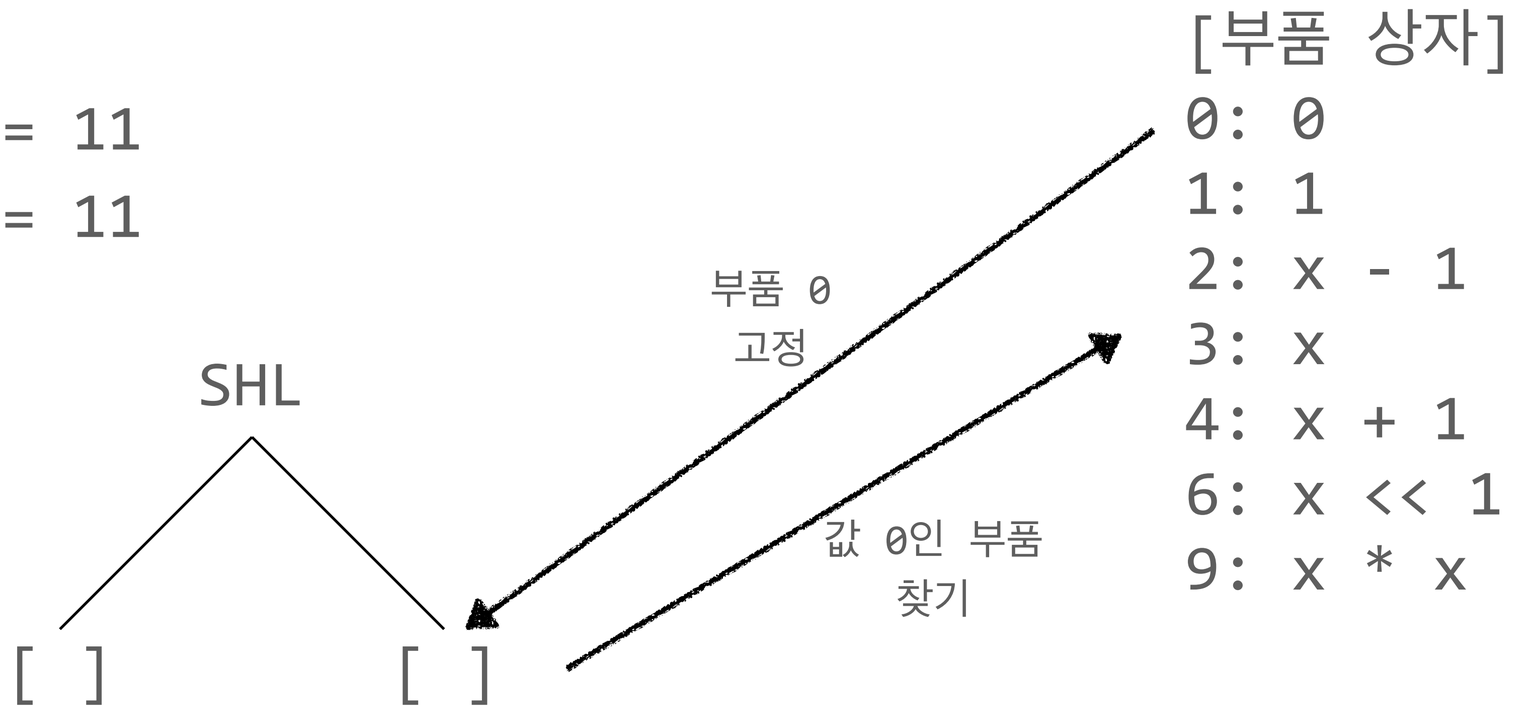
9: x * x

합성 효율 개선

빈칸에 들어갈 값이 명확하면 즉시 부품 선택

조건: x = 3 -> f(x) = 11

후보: SHL([ ], [ ]) = 11



싹수 분석 결과

?

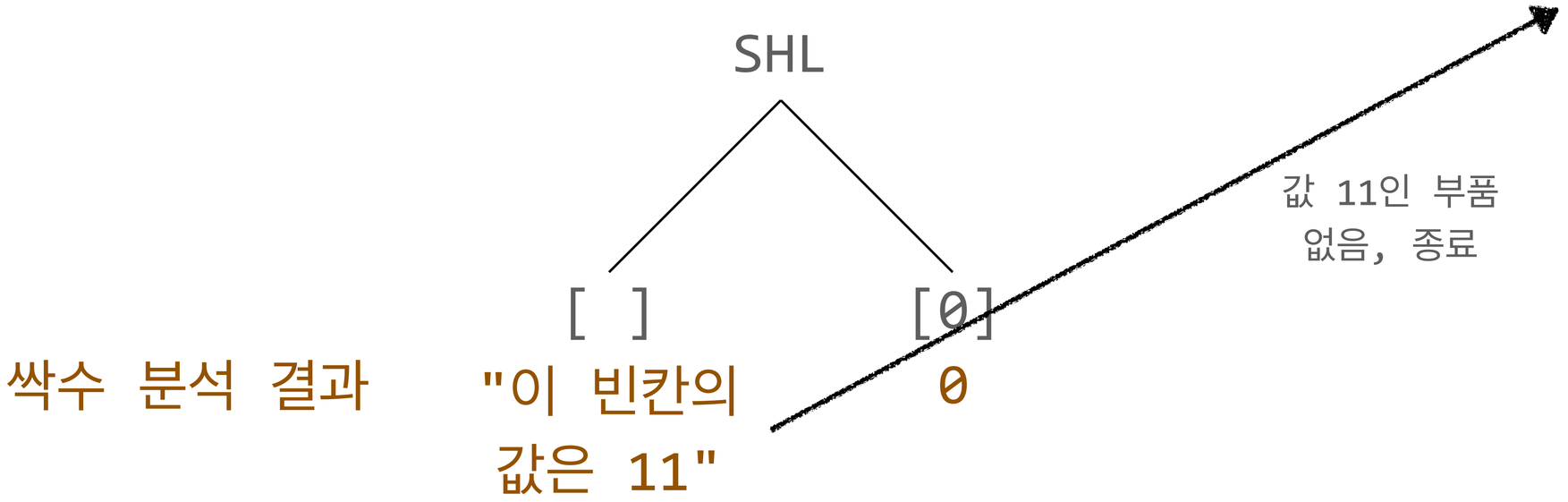
"이 빈칸의 값은 0"

합성 효율 개선

빈칸에 들어갈 값이 명확하면 즉시 부품 선택

조건: x = 3 -> f(x) = 11

후보: SHL([ ], 0) = 11



싹수 분석 결과

"이 빈칸의 값은 11"

[부품 상자]

0: 0

1: 1

2: x - 1

3: x

4: x + 1

6: x << 1

9: x * x

합성 효율 개선

빈칸에 들어갈 값이 명확하면 즉시 부품 선택

- 역함수가 명확히 존재하는 연산에 대해 빈칸 하나가 줄어드는 효과
  - 예를 들어, SUB([ ], [ ]) 의 결과값을 알고있고 부품이 1만개라면
    1만 * 1만 = 1억가지 조합 대신 빈칸 하나에 1만가지 조합만 시도하는 효과

- 운 좋게 역방향 분석이 매우 정확한 경우 빈칸 하나가 줄어드는 효과
  - 앞의 SHL(<<) 연산 예시처럼 양쪽이 모두 빈칸인데도 들어갈 값이 명확할 수 있음

싹수 분석을 더 정교하게

빈칸에 들어갈 값이 명확하면 좋겠다

- 기존 싹수분석 설계의 초점: 후보에 빈칸이 아직 남아있을 때 '싹수 없음' 판정 내리기
  - AND, OR, LSHR, ASHR, SHL 에서 강력한 효과

- 합성 효율 개선 결과 추가로 고려하게 된 초점: 아직 남은 빈칸에 들어갈 값을 상수로 딱 계산해내기
  - ADD, SUB, XOR, NOT, NEG 에서 강력한 효과

- 두 방향 모두 잘 분석하지 못하는 연산은 약점이 됨
  - **MUL**, DIV, REM

싹수 분석을 더 정교하게

빈칸에 들어갈 값이 명확하면 좋겠다

- 두 방향 모두 잘 분석하지 못하는 연산은 중요한 약점인가?
  - **MUL**, DIV, REM

- 실전에서 굉장히 중요한 병목
  - 예를들어, hd-20-d5 문제에서 초기 부품 크기를 9로 설정하면 50만개의 부품 생성
  - 앞의 "강력한 효과" 연산이 빈칸 있는 후보로 선택되면 수초~3분 안에 모든 케이스 검사 완료
  - 분석이 빈약한 MUL, DIV, REM 연산이 후보일 땐 모든 케이스 검사에 1시간 이상 걸림

싹수 분석을 더 정교하게
(비트 벡터)곱셈의 역방향 분석

- 단순히 생각하면 역함수가 있을 것 같은데?

```
6 * a = 24
  a = 4?
```

- 일반 정수가 아닌 비트 벡터 도메인이라 단순하지 않음

```
6 * a = 24 (mod 2^64)
  a = 4?   (mod 2^64)
```

비트 벡터 곱셈의 역방향 분석

예제(8비트로 단순화)

Q1: 6 * a = 24 (mod 256)

비트 벡터 곱셈의 역방향 분석

예제(8비트로 단순화)

Q1: 6 * a = 24 (mod 256)

변형된 같은 문제

Q2: 3 * a = 12 (mod 128)

비트 벡터 곱셈의 역방향 분석

예제(8비트로 단순화)

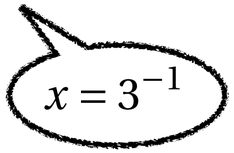
Q1: 6 * a = 24 (mod 256)

변형된 같은 문제

Q2: 3 * a = 12 (mod 128)

Q2를 풀기 위해 필요

Q3: 3 * x = 1 (mod 128)



비트 벡터 곱셈의 역방향 분석

예제(8비트로 단순화)

Q1: 6 * a = 24 (mod 256)

↓ 변형된 같은 문제

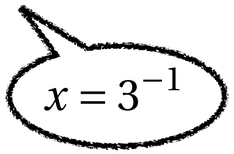
Q2: 3 * a = 12 (mod 128)

↓ Q2를 풀기 위해 필요

Q3: 3 * x = 1 (mod 128)

↓ 3과 128이 서로소일 때(3이 홀수일 때)
x는 유일하며
확장된 유클리드 알고리즘으로 계산 가능

x = 43 (mod 128)



비트 벡터 곱셈의 역방향 분석

예제(8비트로 단순화)

Q1: 6 * a = 24 (mod 256)

↓ 변형된 같은 문제

Q2: 3 * a = 12 (mod 128)

↓ Q2를 풀기 위해 필요

Q3: 3 * x = 1 (mod 128)

↓ 3과 128이 서로소일 때(3이 홀수일 때)
x는 유일하며
확장된 유클리드 알고리즘으로 계산 가능

x = 43 (mod 128)

$x = 3^{-1}$

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

Q2': 3 * 43 * a = 12 * 43 (mod 128)

비트 벡터 곱셈의 역방향 분석

예제(8비트로 단순화)

Q1: 6 * a = 24 (mod 256)

↓ 변형된 같은 문제

Q2: 3 * a = 12 (mod 128)

↓ Q2를 풀기 위해 필요

Q3: 3 * x = 1 (mod 128)

↓ 3과 128이 서로소일 때(3이 홀수일 때)
x는 유일하며
확장된 유클리드 알고리즘으로 계산 가능

x = 43 (mod 128)

Q2': 3 * 43 * a = 12 * 43 (mod 128)

Q2': a = 4 (mod 128)

$12 * 43 - 128 * 4 = 4$

비트 벡터 곱셈의 역방향 분석

예제(8비트로 단순화)

Q1: 6 * a = 24 (mod 256)

↓ 변형된 같은 문제

Q2: 3 * a = 12 (mod 128)

↓ Q2를 풀기 위해 필요

Q3: 3 * x = 1 (mod 128)

↓ 3과 128이 서로소일 때(3이 홀수일 때)
x는 유일하며
확장된 유클리드 알고리즘으로 계산 가능

x = 43 (mod 128)

Q2': 3 * 43 * a = 12 * 43 (mod 128)

Q2': a = 4 (mod 128)

a = 128 K + 4

Q1: a = 4 or 132 (mod 256)

```
  6 * 132
= 792
= 256 * 3 + 24
```

비트 벡터 곱셈의 역방향 분석

알고리즘

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

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

c1 >> k 를 홀수로 만드는 가장 작은 k 선택

비트 벡터 곱셈의 역방향 분석

알고리즘

```
                        c1 * [ ] = c2 mod 2^64

extended_eucledian  (c1 >> k) * [ ] = (c2 >> k) mod 2^(64-k)

      c1_inv * (c1 >> k) * [ ] = c1_inv * (c2 >> k) mod 2^(64-k)
```

비트 벡터 곱셈의 역방향 분석

알고리즘

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

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

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

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

$$[\ ] = c1\_inv * (c2 >> k) + N*2^{(64-k)} \bmod 2^{64}$$

$$[\ ] = \underline{TTTT}\ \{c1\_inv * (c2 >> k)\text{의 비트 표현}\}$$

k개의 Top 비트

비트 벡터 곱셈의 역방향 분석

실제 합성에 활용하려니

- 채워본 부품이 홀수일 때: 다른 빈칸의 정확한 값 계산 가능
  - => 적어도 홀수일 때는 명확한 값 분석을 할 수 있다
- 채워본 부품이 짝수일 때: 2로 몇번 나눌 수 있느냐(k)에 따라 정확도 변동

- 문제점: 입출력 쌍에 따라 부품의 값도 여러가지

부품 상자 구조

입출력 쌍마다 계산된 값

조건:

```
x = 3 -> f(x) = 12
x = 7 -> f(x) = 56
x = 4 -> 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
3	**3**	16	x*x
4	8	3	1+x
...			

*두 번째 이후 (4,8,3)은 중복되어 제거

부품 상자 구조

입출력 쌍마다 계산된 값

조건:

```
x = 3 -> f(x) = 12
x = 7 -> f(x) = 56
x = 4 -> f(x) = 20
```

착안:

한 칸쯤은 홀수가 있겠지

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
3	**3**	16	x*x
4	8	3	1+x
...			

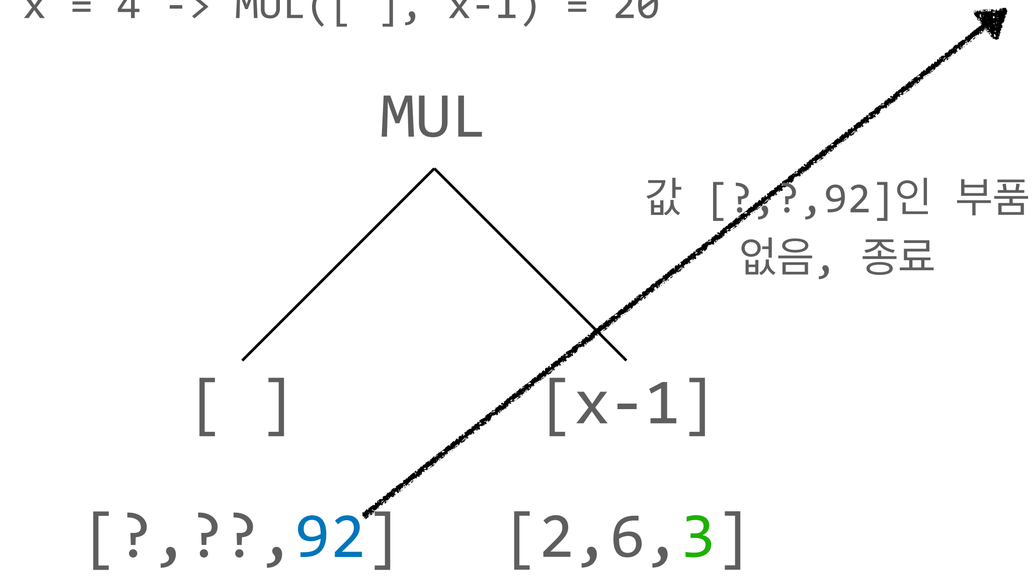
*두 번째 이후 (4,8,3)은 중복되어 제거

합성 효율 개선 ++

(일부 입출력 쌍에 대해서라도) 빈칸에 들어갈 값이 명확하면 맞는 부품 선택

후보:

```
x = 3 -> MUL([ ], x-1) = 12
x = 7 -> MUL([ ], x-1) = 56
x = 4 -> MUL([ ], x-1) = 20
```



? in {6, 128+6}

?? in {52, 128+52}

조건:

```
x = 3 -> f(x) = 12
x = 7 -> f(x) = 56
x = 4 -> 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
3	3	16	x*x
4	8	3	1+x
...			

합성 효율 개선 ++

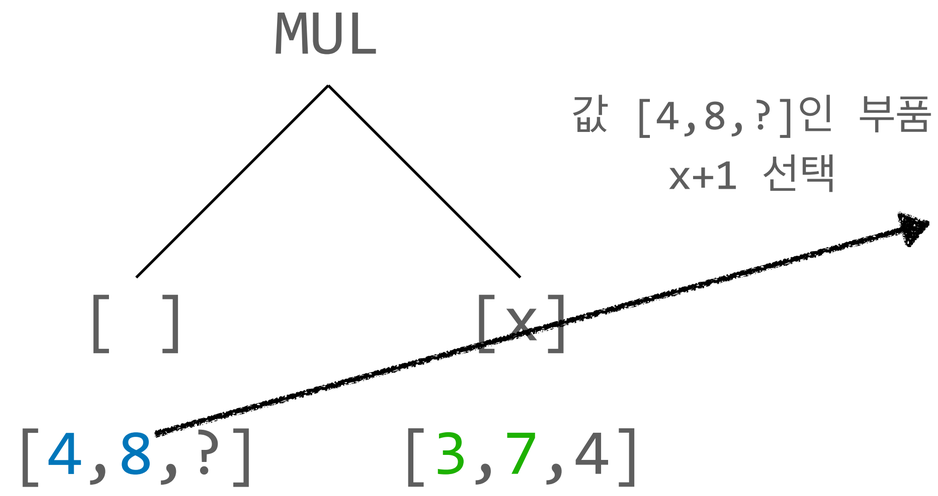
(일부 입출력 쌍에 대해서라도) 빈칸에 들어갈 값이 명확하면 맞는 부품 선택

후보:

```
x = 3 -> MUL([ ], x) = 12
x = 7 -> MUL([ ], x) = 56
x = 4 -> MUL([ ], x) = 20
```

조건:

```
x = 3 -> f(x) = 12
x = 7 -> f(x) = 56
x = 4 -> f(x) = 20
```



값 [4,8,?]인 부품
x+1 선택

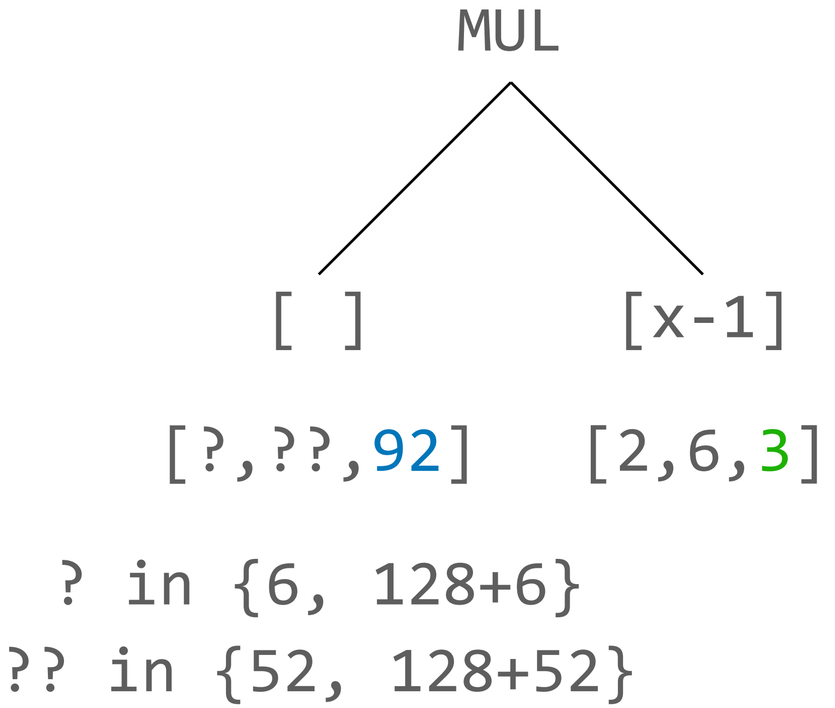
x=3	x=7	x=4	부품
0	0	0	0
3	7	4	x
2	6	3	x-1
4	8	5	x+1
1	3	2	x>>1
6	14	8	x<<1
3	3	16	x*x
4	8	3	1+x
...			

? in {5,64+5,128+5,196+5}

합성 효율 개선 ++
아쉬움

- 역방향 분석 결과 값이 정확히 둘 중 하나일 때 활용하지 못함
  - 특히 이처럼 한 비트만 0 또는 1일 때

- 요약값을 구체화한 집합이 너무 크지 않으면 활용하자



싹수 분석 도메인 활용법 개선

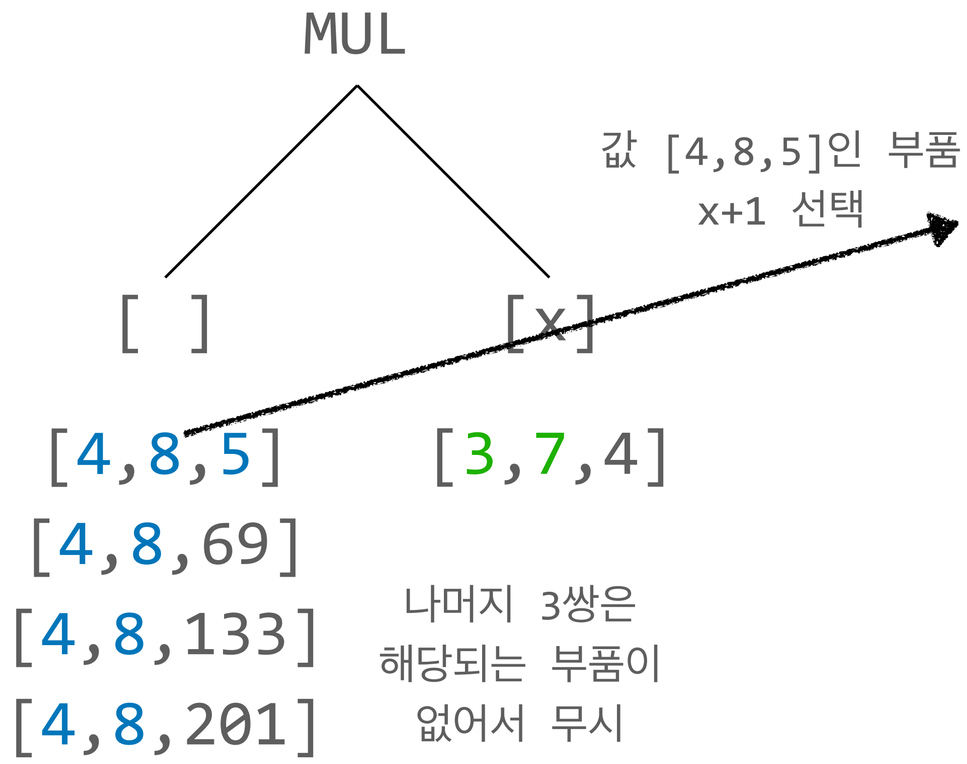
크기를 제한한 구체화

후보:

```
x = 3 -> MUL([ ], x) = 12
x = 7 -> MUL([ ], x) = 56
x = 4 -> MUL([ ], x) = 20
```

조건:

```
x = 3 -> f(x) = 12
x = 7 -> f(x) = 56
x = 4 -> 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
3	3	16	x*x
4	8	3	1+x
...			

싹수 분석 도메인 활용법 개선

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)

- 이렇게까지 해야하나? 해야함
  - 싹수 확인을 위해 수십만개의 부품을 단순히 순회해보는 것도 큰 시간 소모
  - 이 방법으로 싹수 없음을 더 일찍 확인할 수 있음

남은 고지

- 가장 어려운 두 문제 정복
  - 일반화된 방법으로 hd-19-d5, hd-20-d5를 일정 시간 이내에 풀기

- 왜 어려운가
  - 공통: 분석이 약한 연산 포함(bvrem, bvurem)
  - hd-19-d5: 입력 변수가 3개라 금방 부품이 폭발 (부품 크기 9까지 키우면 약 2천만개)
  - hd-20-d5: 한 쪽으로 쏠린 정답 모양