LLM과 요약해석을 이용한 양방향 탐색기반 프로그램 합성 가속화

이 우 석

한양대학교 ERICA 컴퓨터학부

2025 소프트웨어재난연구센터 여름 정기 워크샵

2025. 7. 28

HANYANG UNIVERSITY

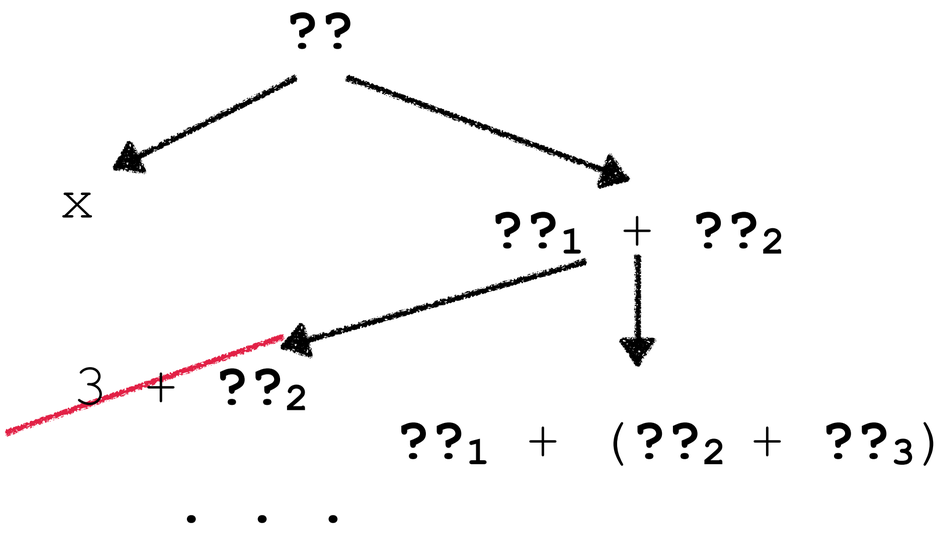
5차년도 상반기

- 양방향 탐색 기반 프로그램 합성 가속화 방법 연구
  - 고성능 합성기들 다수 양방향 탐색전략 사용 중
- 양방향 탐색기반 합성 가속화
  - LLM의 오답을 힌트로 활용 — 함수형언어 합성 문제에 적용 (조한결, 이제형)
  - 요약해석을 통한 탐색 순서 결정 — 표준(SyGuS) 합성 문제에 적용 (서울대 이도윤)
- 결과
  - LLM과 합성기 모두 못푸는 문제 다수 해결
  - SOTA SyGuS 합성기들보다 우위의 성능 달성

합성을 위한 두 탐색 전략

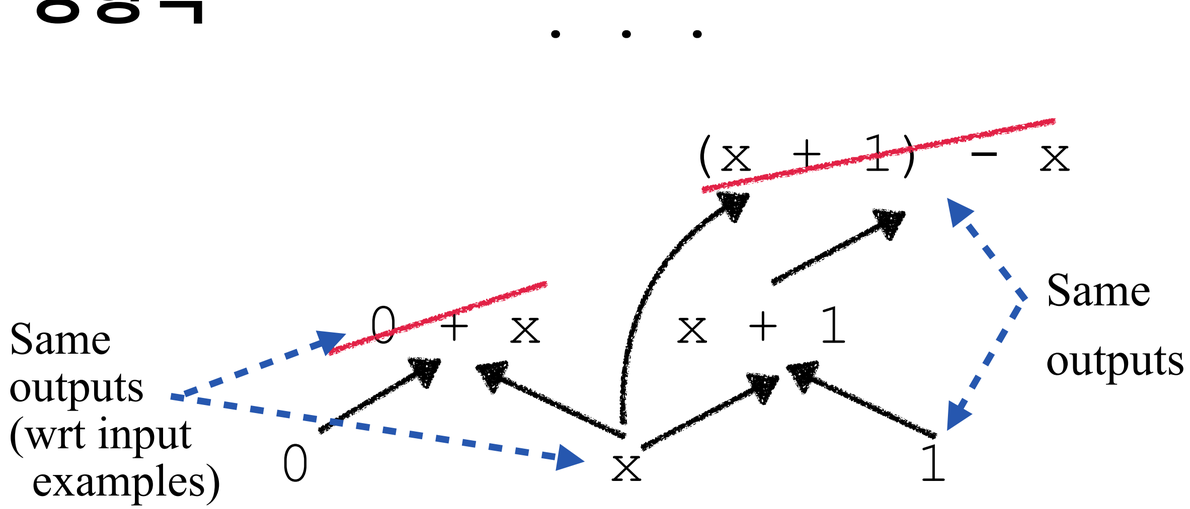
```
function f x = ?? (spec: 0 ⟼ 0, 1 ⟼ 2)
```

하향식



- 빈 프로그램에서 시작, 구멍을 점차 채움
- 싹수없는*infeasible* 프로그램 조기 제거

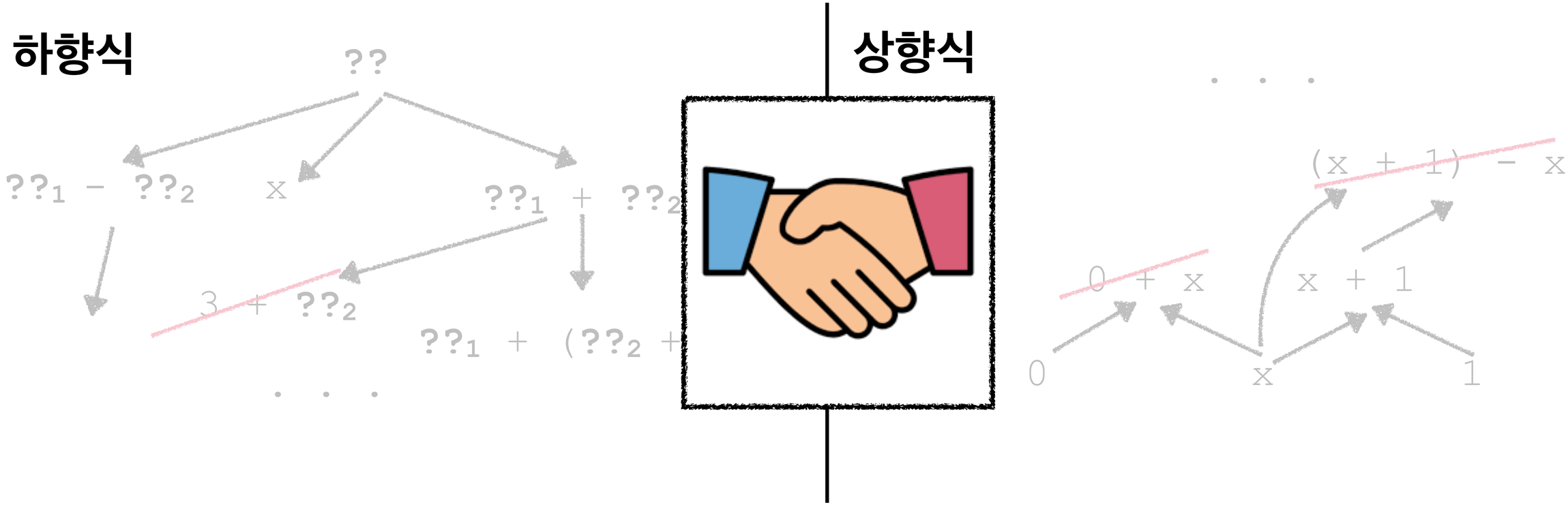
상향식



- 작은식들을 조합, 점차 큰 프로그램 생성
- 실행결과가 같은 중복*redundant* 식들 제거 (*observational equivalence pruning*)

양방향 탐색 전략

- 하향식으로 뼈대 생성 + 상향식으로 구멍을 채울 식 생성†
  - 싹수없는 뼈대 제거 + 중복되는 식 제거
- 다양한 합성기가 사용중 (Duet[POPL'21], Simba[PLDI'23], Flashfill++[POPL'23], Synthphonia[PLDI'25])



†Woosuk Lee, Combining the Top-Down Propagation and Bottom-Up Enumeration for Inductive Program Synthesis, POPL'21

일반적인 양방향탐색 기반 합성 알고리즘

양방향탐색 (문법 G, 스펙 Φ)

1 : C := ∅

2 : repeat

3 : C := **부품식추가**(G, C)

4 : Q := { □ }

5 : while Q ≠ ∅ do

6 : P := **후보선택**(Q)

7 : if P 만족 Φ then return P

8 : if P 싹수없음 then continue

9 : h := **구멍선택**(P)

10: Q := Q ∪ **구멍채우기**$_P$(h, G, C)

11 : done

12: until 시간제한 초과

양방향탐색 (문법 G, 스펙 Φ, LLM 오답 P_{LLM})

1 : C := ∅
2 : repeat
3 : C := **부품식추가**(G, C, P_{LLM})
4 : Q := {□}
5 : while Q ≠ ∅ do
6 : P := **후보선택**(Q, P_{LLM})
7 : if P 만족 Φ then return P
8 : if P 싹수없음 then continue
9 : h := **구멍선택**(P)
10: Q := Q ∪ **구멍채우기**$_P$(h, G, C)
11 : done
12: until 시간제한 초과

LLM 오답을 이용한 탐색 안내

LLM 오답과 비슷한 부품식 추가

LLM 오답과 비슷한 후보 먼저 탐색

양방향탐색 (문법 G, 스펙 Φ)

1 : C := ∅

2 : repeat

3 : C := **부품식추가**(G, C)

4 : Q := { □ }

5 : while Q ≠ ∅ do

6 : P := **후보선택**(Q)

7 : if P 만족 Φ then return P

8 : if P 싹수없음 then continue

9 : h := **구멍선택**(P)

10: Q := Q ∪ **구멍채우기**$_P$(h, G, C)

11 : done

12: until 시간제한 초과

요약해석을 통한 채울부분 선택순서 도출

가지치기 효과 극대화 시키는 방향으로 구멍 선택

LLM 오답을 이용한 양방향탐색 안내

예제 문제

- 두 정수 리스트를 받아서, 각 인덱스 마다 원소들을 비교하여 더 큰 것들의 합을 반환하는 함수 만들기. 리스트 길이가 다를 경우, 더 긴 리스트의 나머지 부분을 결과에 더해서 반환
- 입출력 예제:

```
[ ]          [ 0 ]          ->   0
[1, 2, 3]    [0, 1]         ->   6
[1, 2]       [0, 4, 2, 3]   ->   10
 …
```

- SOTA 합성기들 모두 2분내 합성 실패




```
let rec f x y =
    match x with
    | [] ->
        match y with
        | [] -> 0
        | n2 :: rest2 -> n2 + (f [] rest2)
    | n1 :: rest1 ->
        match y with
        | [] -> n1 + (f rest1 [])
        | n2 :: rest2 ->
            match (compare n1 n2) with
            | EQ -> n1 + (f rest1 rest2)
            | GT -> n1 + (f rest1 rest2)
            | LT -> n2 + (f rest1 rest2)
```

LLM에 질의

- GPT-4o-mini 사용. 틀릴 경우 피드백 제공. 3회까지 기회주기
- 그럼에도 오답 생성

```
let rec f x y =
    match x with
    | [] ->
        match y with
        | [] -> 0
        | n2 :: rest2 -> n2
    | n1 :: rest1 ->
        match y with
        | [] -> n1
        | n2 :: rest2 ->
            (compare n1 n2) + (f rest1 rest2)
```

LLM오답

```
let rec f x y =
    match x with
    | [] ->
        match y with
        | [] -> 0
        | n2 :: rest2 -> n2 + (f [] rest2)
    | n1 :: rest1 ->
        match y with
        | [] -> n1 + (f rest1 [])
        | n2 :: rest2 ->
            match (compare n1 n2) with
            | EQ -> n1 + (f rest1 rest2)
            | GT -> n1 + (f rest1 rest2)
            | LT -> n2 + (f rest1 rest2)
```

정답

틀렸지만 유용

- 전체적인 구조 유사
- 정답의 중요부분이 그대로 존재하거나, 약간 다르게 존재

```
1     let rec f x y =
2         match x with
3         | [] ->
4             match y with
5             | [] -> 0
6             | n2 :: rest2 -> n2
7         | n1 :: rest1 ->
8             match y with
9             | [] -> n1
10            | n2 :: rest2 ->
11                (compare n1 n2) + (f rest1 rest2)
12
13
14
```

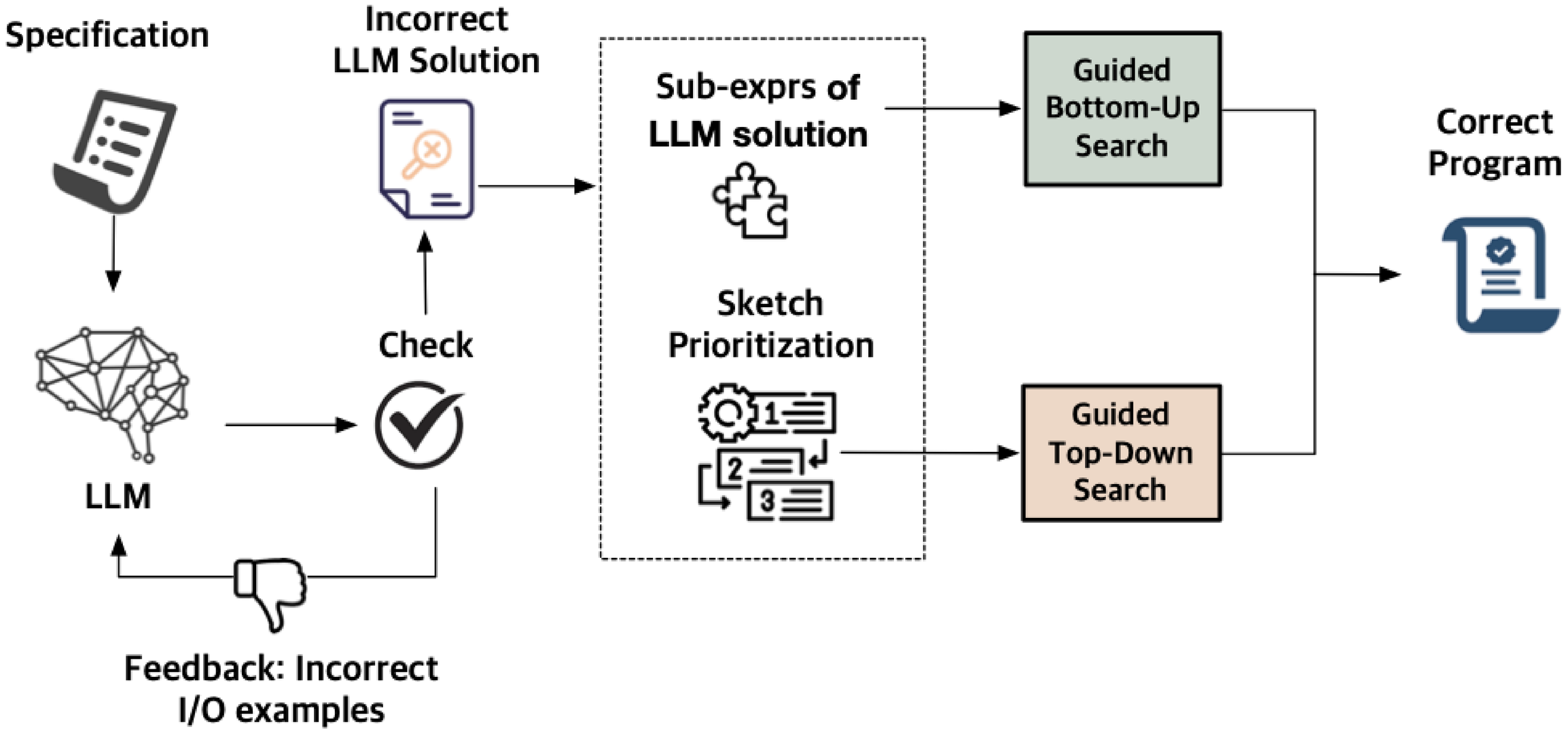
match 구조 유사

정답에 존재하는 식

```
let rec f x y =
    match x with
    | [] ->
        match y with
        | [] -> 0
        | n2 :: rest2 -> n2 + (f [] rest2)
    | n1 :: rest1 ->
        match y with
        | [] -> n1 + (f rest1 [])
        | n2 :: rest2 ->
            match (compare n1 n2) with
            | EQ -> n1 + (f rest1 rest2)
            | GT -> n1 + (f rest1 rest2)
            | LT -> n2 + (f rest1 rest2)
```

LLM오답에 없으나
(f rest1 rest2)와 유사

전체 개요


Specification
Incorrect
LLM Solution
Sub-exprs of
LLM solution
Sketch
Prioritization
Guided
Bottom-Up
Search
Guided
Top-Down
Search
Correct
Program
Check
LLM
Feedback: Incorrect
I/O examples

하향식 탐색 가이드

- 공통패턴 뽑기*anti-unification*: 두 프로그램의 공통부분만 남기고 불일치되는 부분을 변수로

```
let rec f x y =
    match x with
    | [] ->
        match y with
        | [] -> 0
        | n2 :: rest2 -> n2 + (f [] rest2)
    | n1 :: rest1 ->
        match y with
        | [] -> n1 + (f rest1 [])
        | n2 :: rest2 ->
            match (compare n1 n2) with
            | EQ -> n1 + (f rest1 rest2)
            | GT -> n1 + (f rest1 rest2)
            | LT -> n2 + (f rest1 rest2)
```

⊔

```
1   let rec f x y =
2       match x with
3       | [] ->
4           match y with
5           | [] -> 0
6           | n2 :: rest2 -> n2
7       | n1 :: rest1 ->
8           match y with
9           | [] -> n1
10          | n2 :: rest2 ->
11              (compare n1 n2) + (f rest1 rest2)
12
13
14
```

=

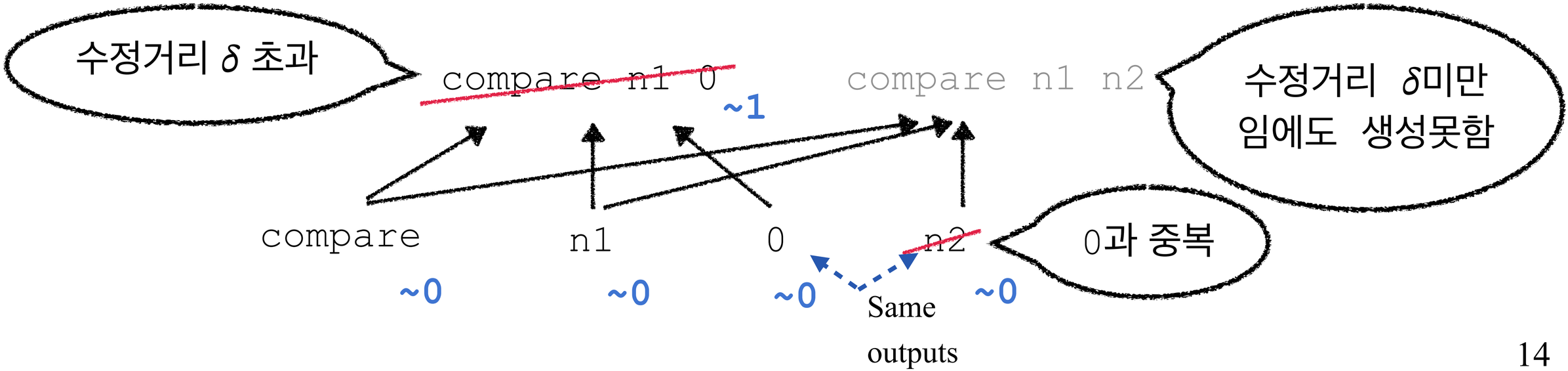
```
let rec f x y =
    match x with
    | [] ->
        match y with
        | [] -> 0
        | n2 :: rest2 -> X
    | n1 :: rest1 ->
        match y with
        | [] -> Y
        | n2 :: rest2 -> Z
```



- 프로그램 A와 LLM오답 사이 거리: LLM오답에서 패턴 변수에 해당하는 부분 크기의 합
- 거리가 짧은 순으로 후보 프로그램들 탐색 — 구멍은 특별히 취급 (어떤 프로그램도 될 수 있음)
- 복잡도 $O(n)$ 으로 $O(n^2)$ 이상의 복잡도의 문자열/트리수정거리 보다 많은 후보 탐색에 유리

상향식 탐색 가이드

- LLM 오답의 부분표현식들 중, 수정거리가 δ 미만인 것이 있는 표현식들 생성
  - δ : LLM에 대한 믿음의 정도 (낮을수록 LLM오답이 정답에 가까울거라 믿음)
- 중복 표현식 제거 시 수정거리가 δ 미만인 식을 생성하지 못하는 경우 발생
  - 예: δ = 1인 상황 (n1, n2 가 각각 첫째 둘째 인자 리스트의 첫원소를 지칭,
    ~k : LLM오답의 부분표현식 중 가장 유사한 것과의 수정거리가 **k**)



상향식 탐색 가이드

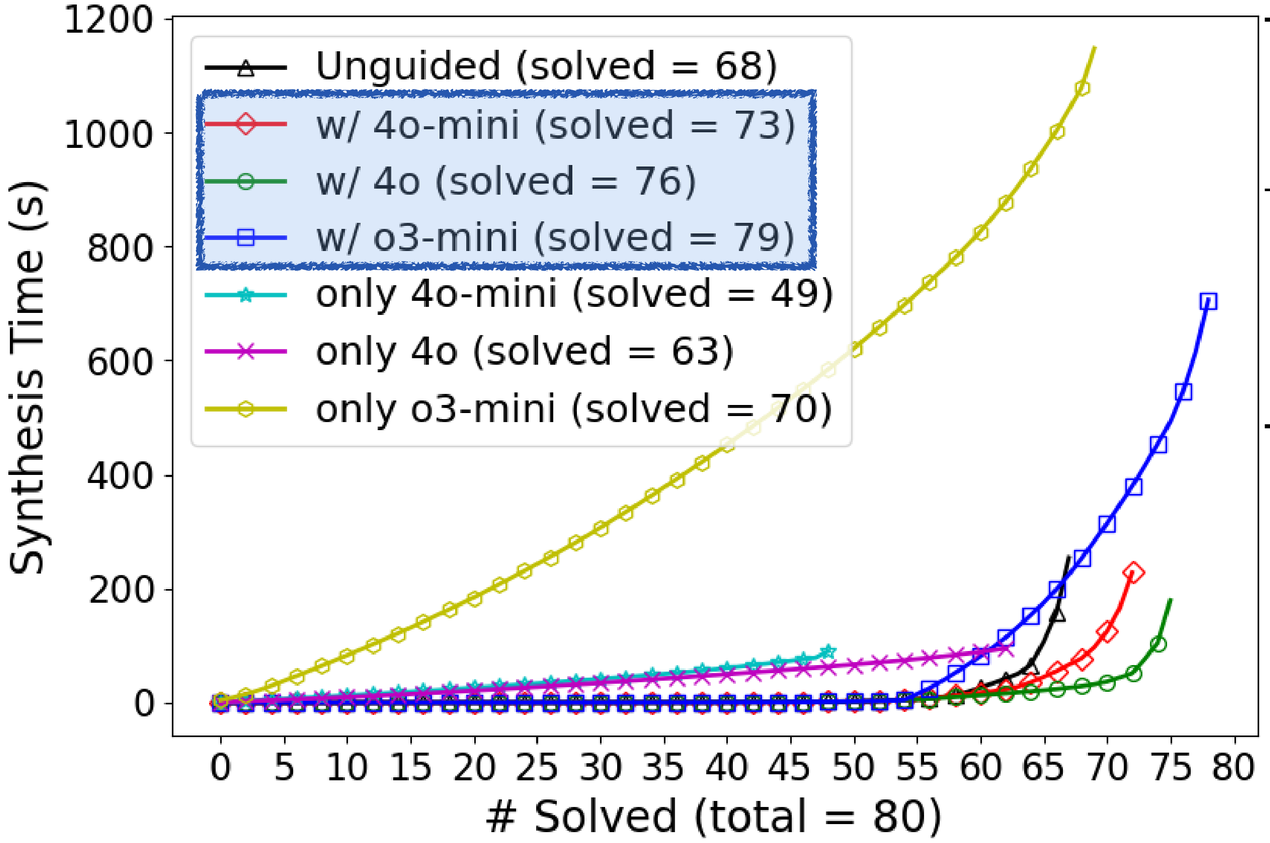
- 상향식 나열 중 LLM오답의 모든 부분표현식들과의 수정거리 리스트 계산
  - LLM오답의 부분표현식들: [`0, n2, n1, x, y, …`]
  - `n2`의 수정거리 리스트: [1, 0, 1, 1, 1, …]
  - `n1`의 수정거리 리스트: [1, 1, 0, 1, 1, …]
- 중복된 표현식이라도 (즉, 이미 생성된 다른 표현식과 출력이 같더라도) 수정거리 리스트가 파레토 최적이면 제거하지 않고 더 큰 표현식 조립을 위해 사용
  - 수정거리 리스트 L1, L2 에 대해, 모든 L1 원소가 같은 위치의 L2 원소이하면 $L1 \leq L2$
  - 어떤 표현식 e에 대해서, 지금까지 나열된 다른 표현식들의 수정거리 리스트 중 e의 것 이하인 수정거리 리스트를 가진 것이 없다면 e는 파레토 최적

실험

- 재귀호출 합성기 Trio 위에 구현, 80개 문제 대상
  - 기존 문제 60개 + 새로 만든 어려운 20개 문제
- 세팅
  - 1초 Trio 돌려보고 못풀면 LLM에 질의
  - LLM이 오답 생성 시 틀린부분 지적해주며 재생성 요구 (3번까지)
  - 끝내 LLM이 정답 생성 못할 경우, 3번 시도 중 가장 괜찮은 오답으로 탐색 가이드
- LLM 모델 : GPT-4o-mini (경량), 4o, o3-mini (추론)

실험

- 성능비교



- 상향/하향 가이드 효과

Methods	**# Solved**		
	4o-mini	4o	o3-mini
UNGUIDED	68	68	68
+ Top-Down Guidance	70 (↑ **2**)	72 (↑ **4**)	77 (↑ **9**)
+ Both Guidance	73 (↑ **5**)	76 (↑ **8**)	79 (↑ **11**)

- **LLM이나 합성기를 단독으로 쓸 때 못푸는 문제들 다수 해결**

실험

- 설계선택*Design choice* 정당화를 위한 실험
- $O(n)$인 공통패턴뽑기 대신 $O(n^3)$ 인 RTED (트리수정거리) 쓸 경우
  - 73 → 62 (↓9) (w/ 4o-mini)　　76 → 70 (↓6) (w/ 4o)
- LLM에게 100개의 답을 만들게 한 후 PCFG(probabilistic context-free grammar) 를 학습, A* 알고리즘으로 탐색 시
  - 73 → 67 (↓6) (w/ 4o-mini)　　76 → 67 (↓9) (w/ 4o)
  - 기존 논문들에서 흔히 LLM + 합성으로 사용하는 방법

[7] Y. Li, J. Parsert, and E. Polgreen, "Guiding enumerative program synthesis with large language models," in *International Conference on Computer Aided Verification* Springer, 2024, pp. 280–301.

[8] S. Barke, E. Anaya Gonzalez, S. R. Kasibatla, T. Berg-Kirkpatrick, and N. Polikarpova, "Hysynth: Context-free llm approximation for guiding program synthesis," *Advances in Neural Information Processing Systems*, vol. 37, pp. 15 612–15 645, 2024.

[10] Y. Li, J. W. d. S. Magalhães, A. Brauckmann, M. F. O'Boyle, and E. Polgreen, "Guided tensor lifting," *Proceedings of the ACM on Programming Languages*, vol. 9, no. PLDI, pp. 1984–2006, 2025.

요약해석을 통한 채울부분 선택순서 도출

Hole-Filling Strategy

예제 문제

- 성 x, 이름 y 를 받아서 이니셜을 만드는 함수 f
- 문법조건

$$\begin{array}{rcll} S & \rightarrow & x \mid y \mid \text{"."} \mid \text{"␣"} & \text{string parameters and constants} \\ & \mid & \texttt{ConCat}(S,S,S) \mid \texttt{CharAt}_0(S) & \text{string operators} \end{array}$$

- 입출력예제

$$f(\text{"}Jan\text{"}, \text{"}Kotas\text{"}) = \text{"}J.K\text{"}$$

- 솔루션

$$f(x,y) = \texttt{ConCat}(\texttt{CharAt}_0(x), \text{"."}, \texttt{CharAt}_0(y))$$

채울부분 선택

- 하향식으로 다음 뼈대를 만듦:

$$\texttt{ConCat}(\square, \square, \square)$$

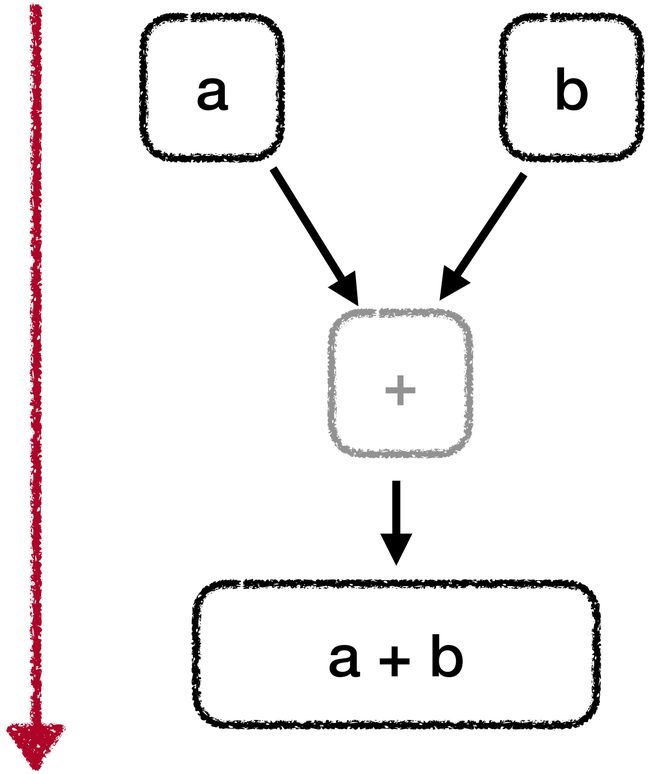
- 상향식으로 다음 부품식들을 만듦:

$$\{x, y, \text{"."}, \text{"␣"}, \texttt{CharAt}_0(x), \texttt{CharAt}_0(y)\}$$

- 합성기는 한 구멍에 부품들을 넣어서 새로운 뼈대들을 만들고, 이들은 탐색큐에 추가됨
- 전방분석: 싹수없는 후보 제거. 후방분석: 각 구멍에 필요전제조건*necessary precondition*유추
- 이하 전/후방 분석을 "싹수분석"이라고 지칭

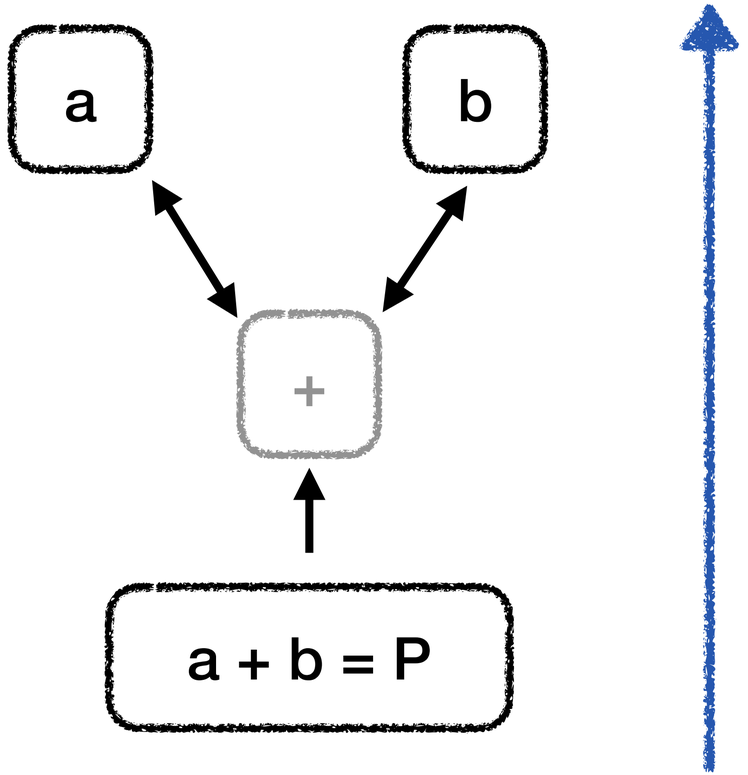
전방/후방 분석

정방향



a의 요약을 계산하고
b의 요약을 계산하고
그것으로 a+b의 요약을 계산

역방향



a+b가 만족해야할 조건 P가 주어졌을 때
a, b 각각의 요약이 만족해야할 조건을 계산
(a, b의 정방향 요약이 계산되어있으면 활용 가능)

채울부분 선택은 가지치기에 영향을 줌

- 합성기가 접미사*suffix* 요약도메인 D 사용, 싹수없는후보를 가지치기한다고 가정
  - 접미사 요약: 미완성 프로그램이 만들 가능성 있는 모든 문자열출력을 공통 접미사로 표현
- 가장 오른쪽 구멍을 "␣" 로 채운경우

$$\texttt{ConCat}(\square, \square, \text{"␣"})$$

  전방분석결과는 "␣" 로 바람직한 출력 "$J.K$" 와 겹치지 않으므로 싹수없음 판정

- 가장 왼쪽 구멍을 x 로 채운 경우

$$\texttt{ConCat}(x, \square, \square)$$

  전방분석: ⊤ (접미사 알수없음)으로 싹수없음 판정 불가.
  후방분석: 가장 오른쪽 구멍 필요전제조건 “K”로 유추 (“K”를 출력하는 부품식만 올 수 있음)

메타분석을 통한 선택전략

- 싹수분석 요약도메인 D 를 요약한 $D^{\#} = \{\texttt{Must}\top, \texttt{May}\not\top\}$ 사용하는 **메타분석** 도입
  - 메타분석: 싹수분석의 가능한 결과들을 포섭하는 분석 (**요약해석의 요약해석**)
  - $\texttt{Must}\top$: 싹수분석 결과가 무조건 $\top$ 일 것.
  - $\texttt{May}\not\top$: 싹수분석 결과가 $\top$ 이 아닐 수 있음.
- 메타분석도 싹수분석과 마찬가지로 전/후방 분석 수행
  - $\texttt{ConCat}(x, \square, \square)$ 의 메타 전방분석 결과는 $\texttt{Must}\top$ (싹수분석하면 싹수있다고 나올것)
  - $\texttt{ConCat}(\texttt{CharAt}_0(x), ".", \square)$ 의 메타 후방분석 결과는 $\texttt{May}\not\top$ (구멍의 필요전제조건은 $\top$이 아닌 것이 유추될 수 있음)

메타분석을 통한 선택전략

가능한 구멍채우기들 (⊡: 구멍에 뭘 채웠다고 치자)	전방 메타분석	후방 메타분석 결과 나머지 구멍들의 필요전제조건
ConCat(⊡, □, □)	Must⊤	모든 구멍 Must⊤
ConCat(□, ⊡, □)	Must⊤	가장 오른쪽 구멍 May⊬
ConCat(□, □, ⊡)	May⊬	가운데 구멍 May⊬

메타분석을 통한 선택전략

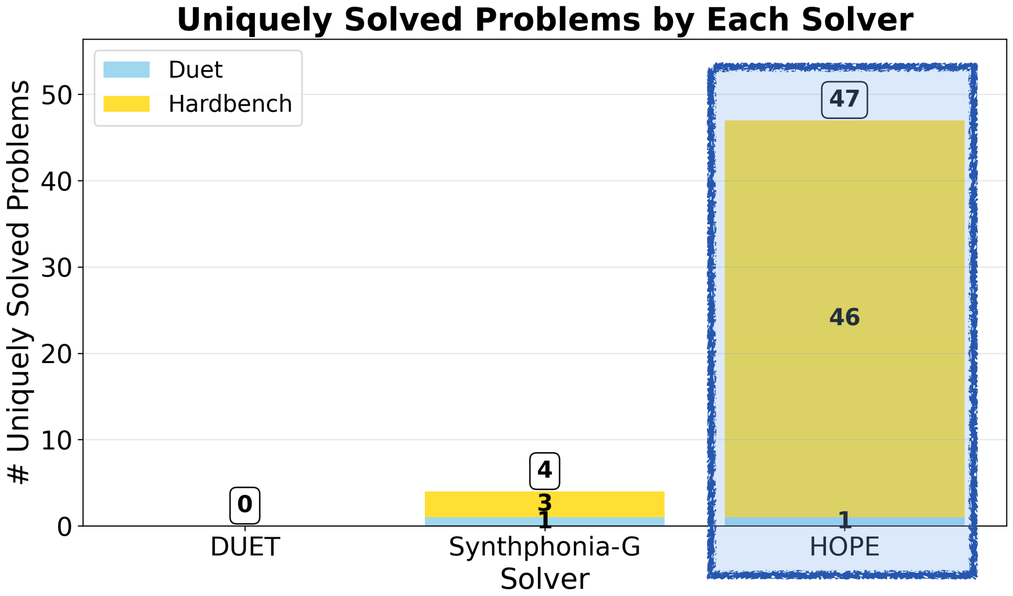
- 모든 구멍 각각에 대해 "가정적 구멍채우기" 후 메타분석 수행
- 각 구멍채우기에 대해 순위매기기
  - 1순위: 전방 메타분석 결과가 $\mathsf{May}\not\top$ — 싹수없음 판정 가능성 존재
  - 2순위: 후방 메타분석 결과 $\mathsf{May}\not\top$인 구멍 존재 — 의미있는 사전조건 유추가능성
  - 3순위: 전후방 메타분석 결과 모두 $\mathsf{Must}\top$
- 위 예제에서는 $\mathsf{ConCat}(\square, \square, \boxdot)$ 가 1순위이므로 가장 오른쪽 구멍을 선택
- 만약 모든 구멍이 3순위이면 임의의 구멍을 선택하고, 싹수분석을 수행하지 않음 (효과가 없을 것이 확실하므로. 이로인해 싹수분석에 드는 비용을 아낄 수 있음)

실험

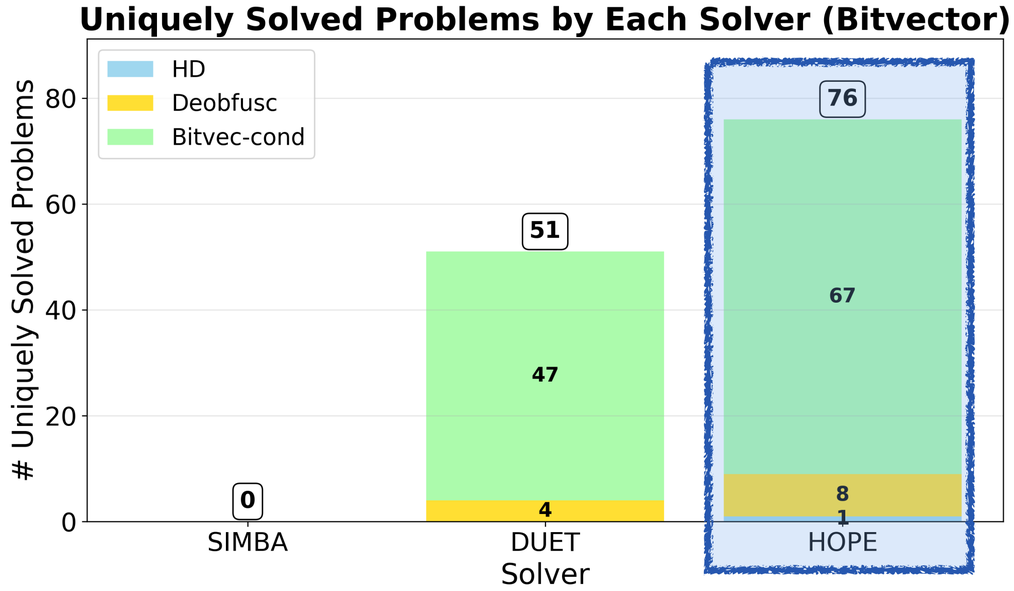
- 도구 : HOPE (**H**ole-filling **O**rder **P**rediction for **E**fficient synthesis)
- 벤치마크 : 총 1631개의 SyGuS 합성문제
  - 337개의 문자열 합성문제
  - 1294개의 비트정수 합성문제
- 경쟁도구
  - Duet [POPL'21]
  - Simba [PLDI'23]
  - Synthphonia [PLDI'25]

결과

- 유일하게 푼 문제 수



(a) # Uniquely solved in the string domain



(b) # Uniquely solved in the bitvec domain

결과 (문자열 합성)

- 푼 문제 수 및 평균 시간 모든면에서 최신도구 Synthphonia 보다 우위

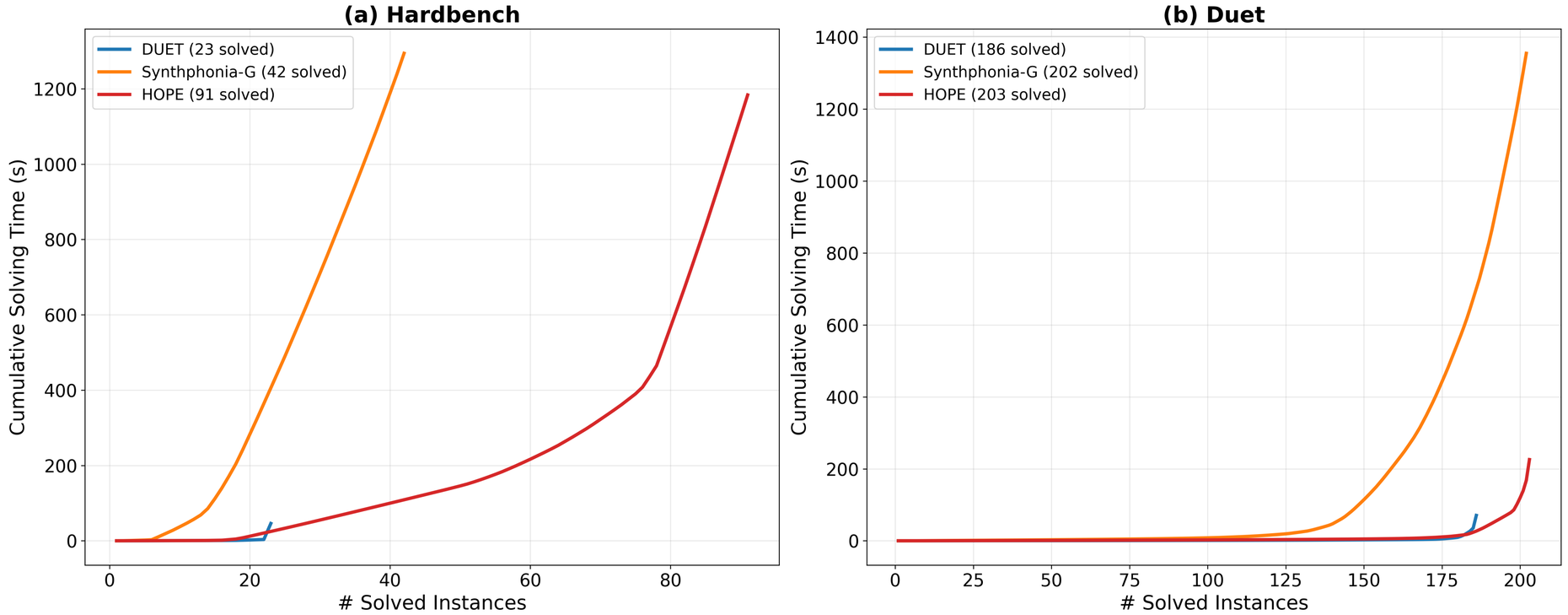


Fig. 8. Comparison between HOPE and the other baseline solvers in the string domain.

결과 (비트정수 합성)

- 푼 문제 수 및 평균 시간 모든면에서 우위

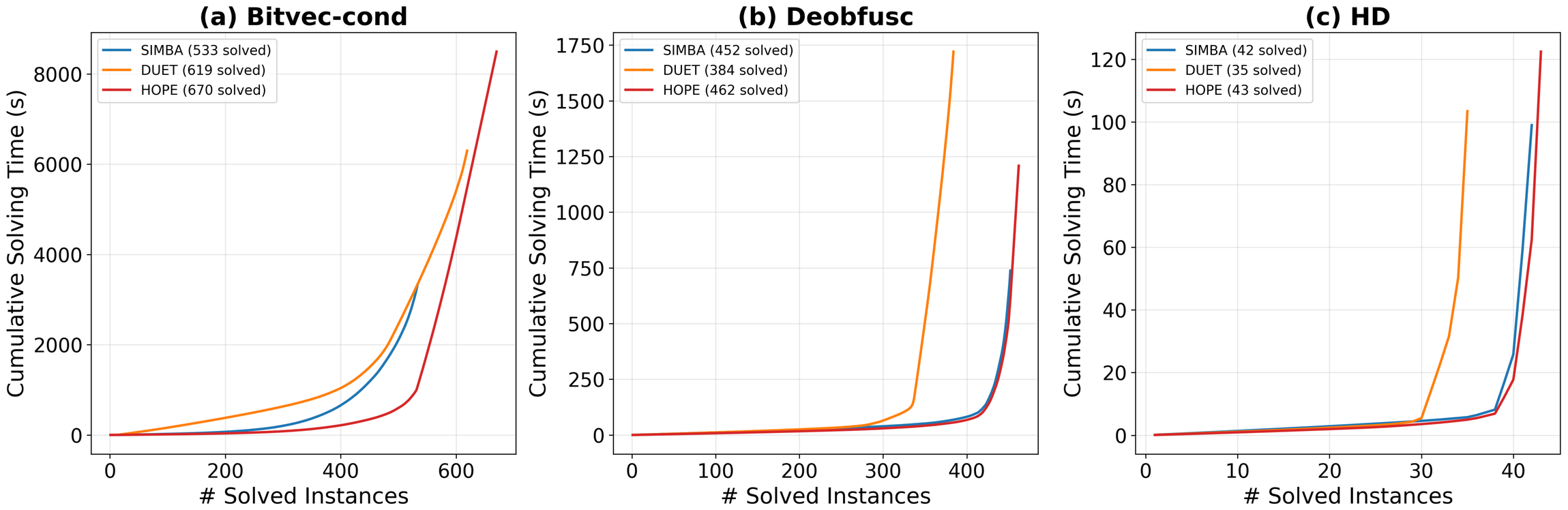


Fig. 9. Comparison between HOPE and the other baseline solvers in the bitvector domain.

그 외 최적화

- 여러 구멍 한꺼번에 채우기
  - 한개만 채워서는 여전히 높은 자유도로 인해 가지치기 효과가 제한적일때 쓸모있음
- 점진적 메타분석
  - 이전 메타분석 결과를 재활용하여 메타분석의 성능가속화



감사합니다!