LLM 솔루션을 참고하는 프로그램 합성

Guided Recursive Program Synthesis referencing LLM solutions

한양대학교 프로그래밍시스템 연구실

조한결

합성 방식의 장단점 Neural vs Basic Search

	Correctness Guarantee	Scalability	Method
Basic search	O	Low	Search algorithm
Neural	X	High	Large Language Model (LLM)

합성 방식의 장단점 Neural vs Basic Search

	Correctness Guarantee	Scalability	Method
Basic search	O	Low	Search algorithm
Neural	X	High	Large Language Model (LLM)

ChatGPT의 오답 예시

- Q : give me ocaml function that takes the last n elements of a list l.
  f [1;2;3;4;5] 3 = [3;4;5],
  f [1;2;3] 1 = [3]
  f [3;2;1] 2 = [2;1]

ChatGPT의 오답 예시

- Q : give me ocaml function that takes the last n elements of a list l.
  f [1;2;3;4;5] 3 = [3;4;5],
  f [1;2;3] 1 = [3]
  f [3;2;1] 2 = [2;1]
- A :

```
let rec f n l =
        match l with
        | [] -> []
        | hd :: tl ->
          if n <= 0 then l
          else f (n - 1) tl
```

ChatGPT의 오답 예시

- Q : give me ocaml function that takes the last n elements of a list l.
  f [1;2;3;4;5] 3 = [3;4;5],
  f [1;2;3] 1 = [3]
  f [3;2;1] 2 = [2;1]

- A :
```
let rec f n l =
        match l with
        | [] -> []
        | hd :: tl ->
          if n <= 0 then l
          else f (n - 1) tl
```

l n

Correct solution

```
if n = 0 then []
else if List.length tl < n
then l
else f tl n
```

Key Idea

- LLM이 만드는 솔루션 활용하기
  - 함수형 프로그램 합성을 위해 LLM에 자연어 기술 + 입출력 예제를 제공 시 **올바르지 못한 솔루션**이 종종 도출됨
  - 비록 올바르지 못한 솔루션이지만 **정답 프로그램과 유사한 구조**를 갖춘 경우가 빈번

Key Idea

- LLM이 만드는 솔루션 활용하기
  - 함수형 프로그램 합성을 위해 LLM에 자연어 기술 + 입출력 예제를 제공 시 **올바르지 못한 솔루션**이 종종 도출됨
  - 비록 올바르지 못한 솔루션이지만 **정답 프로그램과 유사한 구조**를 갖춘 경우가 빈번
- LLM이 준 솔루션을 탐색 기반 합성 가이드에 사용
  - 두 합성 방법의 장점 결합하기
    LLM 기반 합성의 Scalability + 탐색 기반 합성의 Correctness Guarantee
  - LLM 솔루션과 비슷한 후보 우선 탐색하기

양방향 합성 과정 - 후보 나열

Bottom-up Enumerator

Candidate Generator

Component = { x, y, [], hd(x), tl(x), ...}

Size < n 인 부품들 생성

양방향 합성 과정 - 후보 나열

Candidate = { _ , match x with | [] -> _ | hd::tl -> _ match x with | [] -> y | hd::tl -> _ ...}

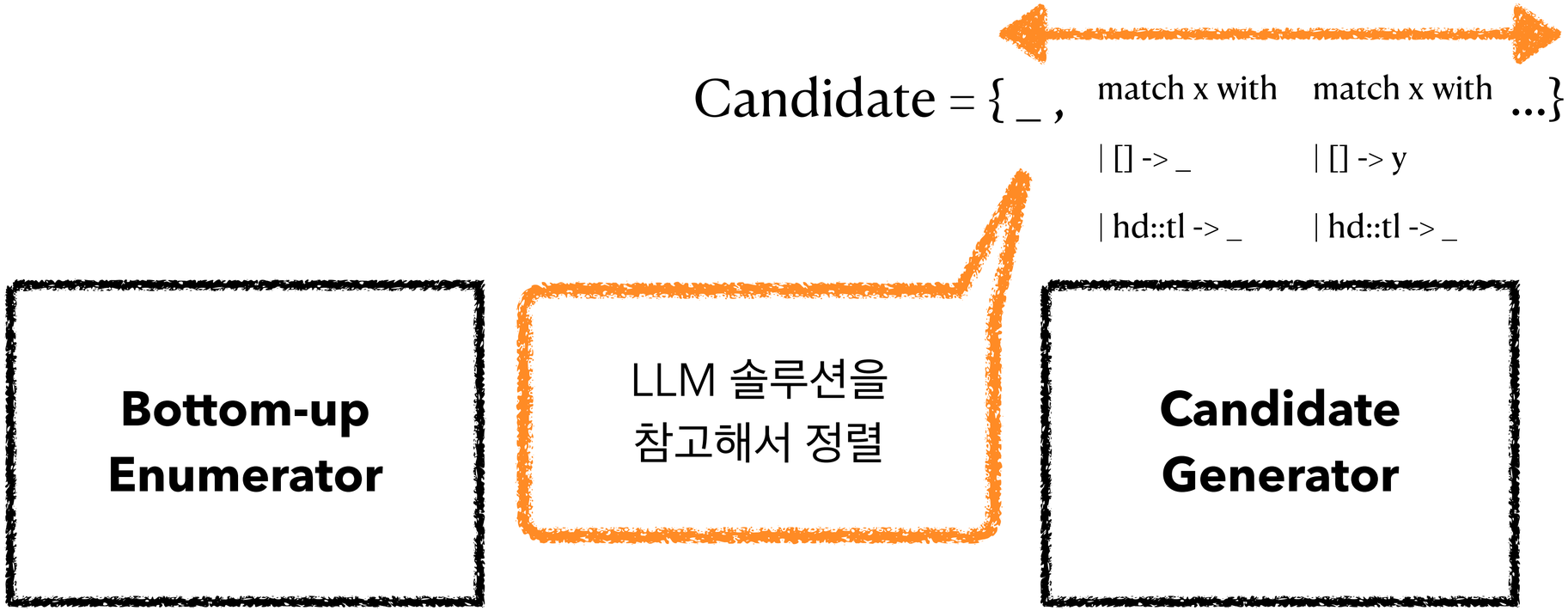
Bottom-up Enumerator

Candidate Generator

Component = { x, y, [], hd(x), tl(x), ...}

_ 에서 시작해 후보 프로그램 나열,
미완성 부분에 부품들을 넣어 완성

양방향 합성 과정 - 후보 나열



Component = { x, y, [], hd(x), tl(x), ...}

방법: 탐색 시 LLM의 솔루션과 유사한 것 먼저 고려하기

- 합성 중 생성하는 후보를 LLM 솔루션과 유사도를 측정해 정렬

- 유사 측정: Tree edit differencing Algorithm
  - Tree edit distance(TED): 원본 트리에서 목적 트리로 (1)노드 삽입, (2)노드 삭제, (3) 노드 수정 세 가지 명령을 통해 변경하기 위한 최단 명령 수 계산
  - Gumtree algorithm: TED + (4)노드 이동 네가지 명령을 통해 변경하기 위한 최단 명령 수 계산

- 미완성 후보가 가지는 미완성부분(_) 을 고려하도록 설계

Tree edit distance

- Tree edit distance(TED)
  원본 트리에서 목적 트리로 (1)노드 삽입, (2)노드 삭제, (3)노드 수정 세 가지 명령을 통해 변경하기 위한 최단 명령 수

$$P = \{add,\ del,\ replace\}$$

$add(t)$: insert node in right-most of t
$del(t)$: delete right-most node of t
$rep(t)$: rename the label of right-most node of t

$$TED(T_1, T_2) = \begin{cases} 0, & T_1 = T_2 \\ min_{a \in P}\{cost(a) + TED(a(T_1), T_2)\}, & \text{otherwise} \end{cases}$$

Tree edit distance

$P = \{add, del, replace\}$

$add(t)$: insert node in right-most of t
$del(t)$: delete right-most node of t
$rep(t)$: rename the label of right-most node of t

$$TED(T_1, T_2) = \begin{cases} 0, & T_1 = T_2 \\ min_{a \in P}\{cost(a) + TED(a(T_1), T_2)\}, & \text{otherwise} \end{cases}$$

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> match y with
              | [] -> x
              | hd2::tl2 -> hd :: f tl _
                            hd :: f tl y
```

Del x 8
Add x 5
Distance = 13

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> hd :: f tl y
```

미완성 후보 고려하기

- 미완성 부분(_)을 가지는 입력도 계산 가능하도록 TED 확장
- 미완성 부분은 목적 트리와의 거리가 가장 작도록 낙관적으로 계산

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> match y with
              | [] -> x
              | hd2::tl2 -> hd :: f tl _
                            hd :: f tl _
```

Del x 7
Add x 4
(_ x 1)
Distance = 11

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> hd :: f tl y
```

LLM 솔루션과 유사도로 탐색하기

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> hd :: f tl y
```

LLM 솔루션

TED distance: 11

TED distance: 11

후보 1

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> match tl with
              | [] -> x
              | hd2::tl2 -> hd2 :: f x _
```

Timeout

후보 2

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> match y with
              | [] -> x
              | hd2::tl2 -> hd :: f tl _
```

Gumtree algorithm δ

- Gumtree algorithm δ
  - 원본 트리에서 목적 트리로 (1)노드 삽입, (2)노드 삭제, (3)노드 수정, **(4)노드 이동** 네 가지 명령을 통해 변경하기 위한 최단 명령 수
  - 동일 서브트리를 찾아 mapping set M에 저장해 노드 이동 수행
  - $dice$ 함수를 사용해 서브 트리가 같은 것 중 이동 시 동일 비율이 더 좋아지는 것 선택

$P = \{add,\ del,\ replace\}$

$add(t)$: insert node in right-most of t
$del(t)$: delete right-most node of t
$rep(t)$: rename the label of right-most node of t

$$TED(T_1, T_2) = \begin{cases} 0, & T_1 = T_2 \\ min_{a \in P}\{cost(a) + TED(a(T_1), T_2)\}, & \text{otherwise} \end{cases}$$

$$\delta(T_1, T_2, M) = min \begin{cases} TED(T_1, T_2) \\ \{\delta(T_1 - p_1, T_2 - p_2, M \backslash \{\langle t_1, p_1, p_2 \rangle\}) + 1 \mid \langle t_1, p_1, p_2 \rangle \in M, t_1 = T_1|p_1, t_2 = T_2|p_2\} \end{cases}$$

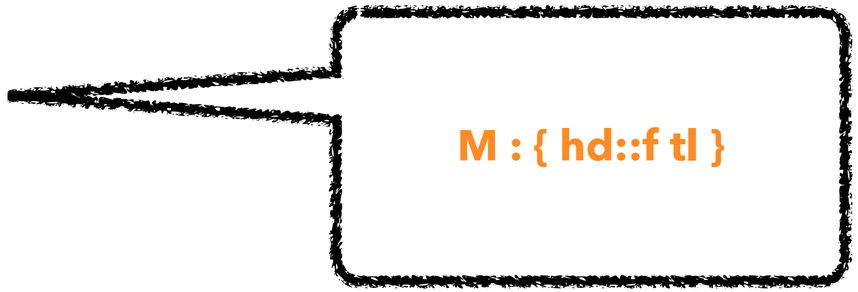
$$dice(t_1, t_2, M) = \frac{2 \times |\{t_1 \in s(t_1) \mid (t_1, t_2) \in M\}}{|s(t_1)| + |s(t_2)|}, \quad s(t) : \text{set of descendants of } t$$

Gumtree algorithm δ

- 원본 트리와 목적트리에 존재하는 모든 동일 서브트리를 찾아 mapping set M에 저장

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> match y with
              | [] -> x
              | hd2::tl2 -> hd :: f tl _
```

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> hd :: f tl y
```



Gumtree algorithm δ

- 찾은 M을 제외한 원본 트리에 TED로 목적 트리와 같게 만들기 위한 최단 명령 계산

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> match y with
              | [] -> x
              | hd2::tl2 -> M
```

Del x 3

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> M
```

M : { hd::f tl }

Gumtree algorithm δ

- M을 이용한 이동 명령으로 목적 트리와 같게 만들기 위한 최단 명령 계산

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl ->
```

hd :: f tl _

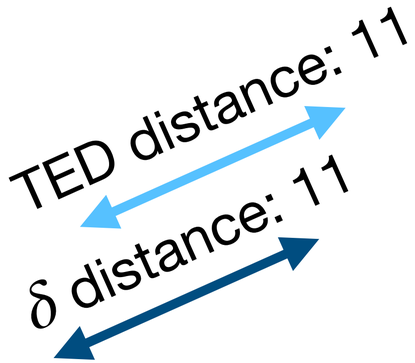
Del x 3
Move x 1
(_ x 1)
Distance = 4

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> hd :: f tl y
```

LLM 솔루션과 유사도로 탐색하기

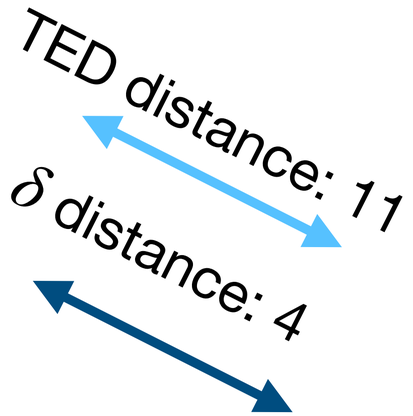
```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> hd :: f tl y
```

LLM 솔루션



후보 1

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> match tl with
              | [] -> x
              | hd2::tl2 -> hd2 :: f x _
```



후보 2

```
let f (x:list) (y:list) : list =
  match x with
  | [] -> y
  | hd::tl -> match y with
              | [] -> x
              | hd2::tl2 -> hd :: f tl _
```

실험

- ChatGPT가 오답을 주고, 기존 합성기 Trio가 합성하지 못한 8개 문제 실험
  - Timeout 5m, X: incorrect, O: correct

	ChatGPT solution	Trio	Guided Trio (TED)	Guided Trio (Gumtree)
list_append_u	**X**	**timeout**	**timeout**	**timeout**
list_last_n	**X**	**X**	**timeout**	**O (290s)**
list_pairwise	**X**	**X**	**O**	**O (0.33s)**
tree_notexist	**X**	**X**	**X**	**O (249s)**
list_drop	**X**	**X**	**O**	**O (1.14s)**
list_last	**X**	**X**	**O**	**O (6.75s)**
list_nth2	**X**	**X**	**O**	**O (7.68s)**
list_seconds	**X**	**timeout**	**timeout**	구현 문제 error

TED 를 사용하는 경우,
4 문제 합성 성공

실험

- ChatGPT가 오답을 주고, 기존 합성기 Trio가 합성하지 못한 8개 문제 실험
  - Timeout 5m, X: incorrect, O: correct

	ChatGPT solution	Trio	Guided Trio (TED)	Guided Trio (Gumtree)
list_append_u	X	timeout	timeout	timeout
list_last_n	X	X	timeout	O (290s)
list_pairwise	X	X	O	O (0.33s)
tree_notexist	X	X	X	O (249s)
list_drop	X	X	O	O (1.14s)
list_last	X	X	O	O (6.75s)
list_nth2	X	X	O	O (7.68s)
list_seconds	X	timeout	timeout	구현 문제 error

Gumtree 경우,
2개 문제에 추가 합성,
총 **6 문제 성공**

실험

- ChatGPT가 오답을 주고, 기존 합성기 Trio가 합성하지 못한 8개 문제 실험
  - Timeout 5m, X: incorrect, O: correct

	ChatGPT solution	Trio	Guided Trio (TED)	Guided Trio (Gumtree)
list_append_u	**X**	**timeout**	**timeout**	**timeout**
list_last_n	**X**	**X**	**timeout**	**O (290s)**
list_pairwise	**X**	**X**	**O**	**O (0.33s)**
tree_notexist	**X**	**X**	**X**	**O (249s)**
list_drop	**X**	**X**	**O**	**O (1.14s)**
list_last	**X**	**X**	**O**	**O (6.75s)**
list_nth2	**X**	**X**	**O**	**O (7.68s)**
list_seconds	**X**	**timeout**	**timeout**	구현 문제 error

같은것 하나만 남기기 때문에 발생하는 문제 발견

같은것 하나만 남기기 문제

- list_append_u 문제에서 필요한 부품을 만들지 않는 것을 확인

f [3;4;5;6] [1;2;3;4] = [1;2;3;4;5;6]
f [1;2;3;4] [3;4;5;6;7] = [3;4;5;6;7;1;2]
f [4;5;6;7] [1;2;3;4] = [1;2;3;4;5;6;7]

주어진 입출력 예제

필요한 부품: **(is_member hd(x) y)**

필요한 부품의 signature: **[true; false; true]**

입력 예제 실행 결과

signature를 **[true; false; true]** 로 가지는 부품이 존재: **(is_member 1 y)**

같은것 하나만 남기기 문제

- list_append_u 문제에서 필요한 부품을 만들지 않는 것을 확인

원인: 같은것 하나만 남기기(Observational Equivalence)

- 탐색 기반 합성에 쓰이는 최적화 기법으로 입력에 대해 결과가 같은 부품 중 하나만 남기기
- 필요한 부품을 생성해도 이미 같은 결과를 가지는 부품이 존재하는 경우 무시
- 입출력 예제가 적을수록 발생하기 쉬움

TODO **같은것 하나만 남기기 시 LLM 솔루션을 참고하여 부품 남기기**

결론

- LLM 솔루션을 탐색 가이드에 활용하여 합성 가속화
- 올바름을 보장해주지 못하는 LLM 기반 솔루션을 활용하여 올바른 솔루션 합성
- 기존에 하지 못하는 문제에 대해 성공적으로 합성
- Future work…
  - 같은것 하나만 남기기 시 LLM 솔루션을 참고하여 부품 남기기
  - Challenge: 탐색 공간에 영향을 덜 주어야 함, LLM이 틀리는 경우 필요없는 부품을 생성