



잘게 나눠 기억해두어 알뜰하게 실행하기

서장협(서울대학교 프로그래밍연구실)

2026년 8월 26일

프로그래밍언어연구회 2026 여름학교 번개 발표

기억해두기 *memoization*를 함수보다 작은 단위로 할 수 없을까?

문제 상황

- 기억해두기 *memoization*는 보통 함수 단위로 적용

문제 상황

- 기억해두기 *memoization*는 보통 함수 단위로 적용
- 함수가 복잡해질수록 기억해두기를 적용하기는 어려워짐

문제 상황

- 기억해두기 *memoization*는 보통 함수 단위로 적용
- 함수가 복잡해질수록 기억해두기를 적용하기는 어려워짐
  - 많은 매개변수, 딴짓 *side effect*, ...

문제 상황: 예시

```
let fn f(x, y) =>
  if x / 4 = 0 then 1 else y
in
f(0, 2) + f(0, 3) + f(1, 3)
```

문제 상황: 예시

```
let fn f(x, y) =>
  if x / 4 = 0 then 1 else y
in
f(0, 2) + f(0, 3) + f(1, 3)
```

- 함수가 전달받는 모든 인자의 값이 같아야 함
  - `f(0, 2)` 이후 `f(0, 3)`: `y`가 바뀐 건 사실 상관 없음

문제 상황: 예시

```
let fn f(x, y) =>
  if x / 4 = 0 then 1 else y
in
f(0, 2) + f(0, 3) + f(1, 3)
```

- 함수가 전달받는 모든 인자의 값이 같아야 함
  - `f(0, 2)` 이후 `f(0, 3)`: `y`가 바뀐 건 사실 상관 없음
- 함수가 전달받는 모든 인자의 값이 같아야 함
  - `f(0, 3)` 이후 `f(1, 3)`: `x`가 바뀐 건 사실 상관 없음 (`x / 4` 동일)

문제 상황: 예시

```
let fn f(x, y) =>
  if x / 4 = 0 then 1 else y
in
f(0, 2) + f(0, 3) + f(1, 3)
```

- 함수가 전달받는 모든 인자의 값이 같아야 함
  - `f(0, 2)` 이후 `f(0, 3)`: `y`가 바뀐 건 사실 상관 없음
- 함수가 전달받는 모든 인자의 값이 같아야 함
  - `f(0, 3)` 이후 `f(1, 3)`: `x`가 바뀐 건 사실 상관 없음 (`x / 4` 동일)

→ '인자', '함수' 라는 코드의 겉모양에 집중하고 있어서 생기는 문제

기억해두기의 단위를 더 잘게 나누어 식 단위로 바라본다면?

잘게 나눠 기억해두기 *fine-grained memoization*

- 어떤 식의 값에 정말로 영향을 미치는 식들을 실행 전에 분석

잘게 나눠 기억해두기 *fine-grained memoization*

- 어떤 식의 값에 정말로 영향을 미치는 식들을 실행 전에 분석
- 모든 식을 기억해 둘 수는 없음 (너무 많아!)

잘게 나눠 기억해두기 *fine-grained memoization*

- 어떤 식의 값에 정말로 영향을 미치는 식들을 실행 전에 분석
- 모든 식을 기억해 둘 수는 없음 (너무 많아!)
- 실행 전 분석을 통해 기억해 둘 만한 식 선정

잘게 나눠 기억해두기 *fine-grained memoization*

- 어떤 식의 값에 정말로 영향을 미치는 식들을 실행 전에 분석
- 모든 식을 기억해 둘 수는 없음 (너무 많아!)
- 실행 전 분석을 통해 기억해 둘 만한 식 선정
- 기억해두기의 타율이 좋아지지 않을까?

많은 관심 부탁드립니다!