LapsPython – Extend LAPS to Synthesize Python/R

- DC/LAPS synthesize code in the "built-in" λ-calculus (~LISP)
- More useful output would be code e.g. in Python
- Idea:
  - Create a "rule-based" translator from the λ-calculus to Python or R
  - Target useful domains, e.g. *data analysis* (Pandas), string processing, …
    - For each domain a set of suitable LIST primitives and examples are needed
  - Might address different target languages (Python, R, Typescript…)
- Limit research risk:
  - Start with a subset of Python and a corresponding simple set of primitives, then move to more advanced domain, e.g. Pandas
  - Provide sufficient many training examples (100+) and order them
- Challenges: requires understanding λ-calculus (~LISP) and some knowledge about "compiler technology" (for the translator)

```
MISS First letters of words (I)
MISS First letters of words (II)
MISS First letters of words (III)
MISS First letters of words (IIII)
MISS First letters of words (IIIII)
MISS First letters of words (IIIIII)
HIT Take first character and append '.' w/ (lambda (cons (car $0) (cons '.' empty))) ;
HIT Take first character and append ',' w/ (lambda (cons (car $0) (cons ',' empty))) ;
HIT Take first character and append ' ' w/ (lambda (cons (car $0) (cons SPACE empty)))
HIT Take first character and append '(' w/ (lambda (cons (car $0) (cons LPAREN empty)))
HIT Take first character and append ')' w/ (lambda (cons (car $0) (cons RPAREN empty)))
HIT Take first character and append '-' w/ (lambda (cons (car $0) (cons '-' empty))) ;
```

```
MISS First letters of words (I)
MISS First letters of words (II)
MISS First letters of words (III)
MISS First letters of words (IIII)
MISS First letters of words (IIIII)
MISS First letters of words (IIIIII)
HIT Take first character and append '.' w/ (lambda (cons (car $0) (cons '.' empty))) ;
HIT Take first character and append ',' w/ (lambda (cons (car $0) (cons ',' empty))) ;
HIT Take first character and append ' ' w/ (lambda (cons (car $0) (cons SPACE empty)))
HIT Take first character and append '(' w/ (lambda (cons (car $0) (cons LPAREN empty)))
HIT Take first character and append ')' w/ (lambda (cons (car $0) (cons RPAREN empty)))
HIT Take first character and append '-' w/ (lambda (cons (car $0) (cons '-' empty))) ;
```

```
Grammar after iteration 1:
-0.618106       t0      $_
0.000000        char -> char -> bool    char-eq?
0.000000        list(t0) -> bool        empty?
0.000000        int     1
-0.002220       int -> list(int)        range
-0.022608       list(t0) -> int length
-0.022608       int -> int -> int       +
-0.022608       int -> int -> int       -
-0.022608       int     0
-0.447027       char    '.'
-0.447027       char    ','
-0.447027       char    '-'
-0.447027       char    SPACE
-0.447027       char    RPAREN
-0.447027       char    LPAREN
-1.011074       t0 -> list(t0) -> list(t0)      cons
-1.185772       list(char)      STRING
-1.312380       list(t0)        empty
-1.444016       list(t0) -> t0  car
-1.511830       list(t0) -> list(t0)    cdr
-1.521691       (t0 -> t1) -> list(t0) -> list(t1)      map
-1.522244       t0 -> (t0 -> bool) -> (t0 -> t1) -> (t0 -> t0) -> list(t1)      unfold
-1.640042       int -> list(t0) -> t0   index
-1.661394       list(t0) -> t1 -> (t0 -> t1 -> t1) -> t1        fold
-1.662649       bool -> t0 -> t0 -> t0  if
-0.934457       list(t0) -> list(t0) -> list(t0)        #(lambda (lambda (fold $0 $1 (lambda (lambda (cons $1 $0))))))
```

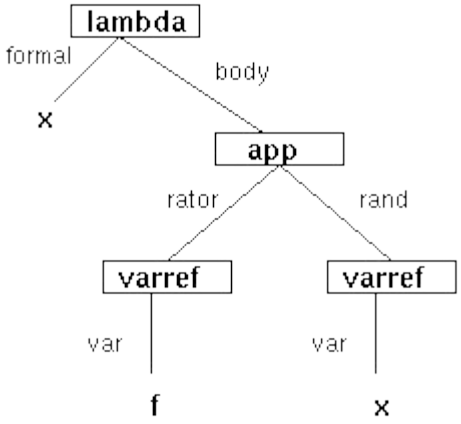
Requirements & Plan

1. Extract λ expressions (ASAP)
    - Where is the relevant code executed?
    - How are λ expressions and primitives stored internally?

Requirements & Plan

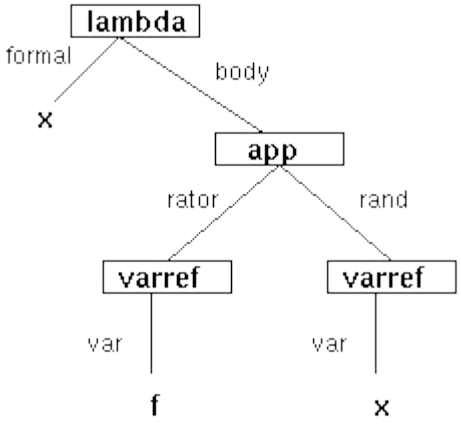
1. Extract λ expressions (ASAP)
   - Where is the relevant code executed?
   - How are λ expressions and primitives stored internally?

2. Parse lambda expressions
   - Abstract Syntax Tree
   - Generate 1 line of code for each intermediate node



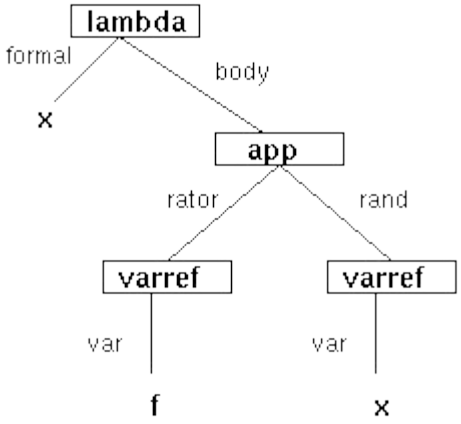
Requirements & Plan

1. Extract λ expressions (ASAP)
   - Where is the relevant code executed?
   - How are λ expressions and primitives stored internally?
2. Parse lambda expressions
   - Abstract Syntax Tree
   - Generate 1 line of code for each intermediate node
3. Rule-based translation for simple domains (May/June)
   - Rule DB
   - Tests!



Requirements & Plan

1. Extract λ expressions (ASAP)
   - Where is the relevant code executed?
   - How are λ expressions and primitives stored internally?
2. Parse lambda expressions
   - Abstract Syntax Tree
   - Generate 1 line of code for each intermediate node
3. Rule-based translation for simple domains (May/June)
   - Rule DB
   - Tests!
4. Extension (remaining time)
   - More complex domains (e.g. pandas)
   - Translation to R (lower priority)



Domains

- Start with list/string processing
  - Tasks are already implemented
- Later extend e.g. to pandas
  - Requires 100+ custom training samples
  - When translation works for list/string processing
- Combined domains
  - E.g. process string column in pandas dataframe
  - Questionable because LAPS separates domains
  - No priority so far