To *Iterate* is Human, To *Recurse* is Divine

Mapping Iterative Python to Recursive SQL

Tim Fischer

.tim.fischer@uni-tuebingen.de
What’s the Problem?

```python
def march(current: Point) -> list[Point]:
goal : Point | None = None
track: bool = False
march: list[Point] = []

while not track or current != goal:
square: Squares = SQL(
    "SELECT s FROM squares AS s WHERE s.xy = $1",
    [current],
)

dir: Directions = SQL(
    "",
    SELECT d
    FROM directions AS d
    (   d.ll,   d.lr,   d.ul,   d.ur)
    "",
    [square],
)

if not track and dir.track: track, goal = True, current
if track: march.append(current)
current += dir.dir

return march
```
Too Many Round Trips!

Invoke from Python

Inline into SQL
Compiler Stages

Frontend

- **Python** (source code)
- **Type Checking**
- **Desugaring**
- **Lowering → SSA**
  - `Python` → `AST_\tau` → `AST_s` → `SSA + SQL` → `SQL`
  - Source code → Fully typed → Simplified → `SSA + SQL` → `SQL`

Backend

- **Python** (iterative)
- **Lowering → ANF**
- **Trampolining**
- **Code Generation**
  - `Python` → `SSA` → `ANF` → `ANF + SQL` → `SQL`
  - Iterative → GOTO → Recursive → Single Loop → `SQL`
Types of Control Flow

def virtual_machine(source):
    ip: int = 0
    regs: list[int] = []

while True:
    inst = load_inst(ip)
    match inst.op:
        case "lod": ...
        case "mov": ...
        case "add": ...
        case "sub": ...
        case "jeq": ...
        case "HLT": ...

Non-Branching Linear Control Flow in SQL

```
SELECT ... 
FROM LATERAL (SELECT 1 :: int ) AS let_1(ip), 
LATERAL (SELECT ARRAY[] :: int[]) AS let_2(regs)
```
WITH RECURSIVE
loop("done?", ...) AS (  

| SELECT False, ...  |
| UNION |
| SELECT ...  |
FROM loop AS state  |
WHERE NOT state "done?"  |
)

SELECT ...
FROM loop  |
WHERE "done?"
Branching Linear Control Flow in SQL

```
SELECT next_state.*
FROM loop AS state,
     LATERAL (SELECT load_inst(ip)) AS let_3(inst),
     LATERAL (
         SELECT ... WHERE inst.op = 'lod'
             UNION ALL
         SELECT ... WHERE inst.op = 'mov'
             UNION ALL
         SELECT ... WHERE inst.op = 'add'
             UNION ALL
         SELECT ... WHERE inst.op = 'sub'
             UNION ALL
         SELECT ... WHERE inst.op = 'jeq'
             UNION ALL
         SELECT ... WHERE inst.op = 'HLT'
     ) AS next_state
WHERE NOT "done?"
```
Join us in our effort to marry complex computations with SQL!

Fully-Funded PhD or Postdoc Position

Database Systems Group @ Uni Tübingen

Approach Torsten Grust, who is around this week.