Befunge-93 in SQL

(Ab-)Using SQLs Turing Completeness

Tim Fischer

Eberhard Karls Universität Tübingen
t.fischer@student.uni-tuebingen.de

SQL is a Programming Language
January 3, 2023

Befunge TL;DR

Befunge is a...

- stack-based
- Turing complete
- two-dimensional
- self-modifying
- imperative
- esoteric
- programming language.

Befunge Commands

Commands	Description
>v<^	Set direction of program counter
#	Skip the next command
?	Set direction of program counter at random
_|	Vertical and horizontal branching
0123456789	Put number on stack
:$\	Stack modification
+-*/%	Arithmetic operators
.,	Output either numeric or ASCII value to stdout
&~	Take either numeric or ASCII value from stdin
gp	Get or put ASCII value from the grid
"	Toggle string mode
@	Halt program execution

Simple Examples

```
"!dlroW ,olleH">:#v_@
                ^ ,<
```

Figure: Hello World

```
&>:1-:v v *_$.@
 ^     _$>\:^
```

Figure: Factorial

Examples

```
211p&01p>121p          >21g1+21p 11g21g-v>11g21g%#v_v
                                         >|
                  v,,,,, ,,,.g11"is prime"<
>       ^         >   v ^                      <
^_@#-g10p11:+1g11,*25<,,,,,,,,,,,,.g11"is not prime"<
```

Figure: Calculating Prime Numbers

Interpreter Pseudocode

```
def interpreter(source):
  program = preprocess(source)
  state   = make_inital_state(program)

  while not state.done():
    match state.mode:
      case "⚙️": ...
      case "〰️": ...

    match state.direction:
      case "➡️": ...
      case "⬇️": ...
      case "⬅️": ...
      case "⬆️": ...

  return state.result
```

Types of Control Flow

```
def interpreter(source):
    program = preprocess(source)
    state   = make_inital_state(program)

    while not state.done():
        match state.mode:
            case "⚙": ...
            case "↗": ...

        match state.direction:
            case "➡": ...
            case "⬇": ...
            case "⬅": ...
            case "⬆": ...

    return state.result
```

Non-Branching Linear Control Flow in SQL

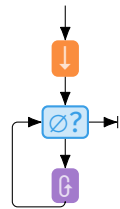
```
SELECT ...
FROM LATERAL (
       preprocess(source)
     ) AS _1(program),
     LATERAL (
       make_inital_state(program)
     ) AS _2(state)
```

Condition-controlled Non-Linear Control Flow in SQL

```
WITH RECURSIVE
  loop(...) AS (
    SELECT ↦ init

      UNION

    SELECT ↦ body
    FROM     loop AS state
    WHERE  ↦ NOT state.done()
  )
SELECT ...
FROM   loop
WHERE  ...
```



Branching Linear Control Flow in SQL

```
SELECT ...
FROM   loop AS state,
LATERAL (
  SELECT ... WHERE mode='⚙'
    UNION ALL
  SELECT ... WHERE mode='↻'
) AS next,
LATERAL (
  SELECT ... WHERE direction='→'
    UNION ALL
  SELECT ... WHERE direction='↓'
    UNION ALL
  SELECT ... WHERE direction='←'
    UNION ALL
  SELECT ... WHERE direction='↑'
) AS move
```