Recreating goto in Python; how and why

CITRENZ 2023 Auckland

Carl Cerecke

carl.cerecke@nmit.ac.nz

https://github.com/cdjc/goto

September 27–29, 2023

Outline

- Memories
- Motivation
- Mechanism
- Measurement
- Musing

Memories: A short history of goto

- In the beginning was the `goto`
- 1958 Heinz Zemanek expresses doubts about goto at pre-ALGOL meeting.
- 1968 Edsgar Dijkstra "GOTO Considered Harmful"
- 1974 Don Knuth "Structured Programming with go to statements"
- 1987 Frank Rubin " 'GOTO Considered Harmful' Considered Harmful"
- 1995 Java first major language with no `goto` statement.

Motivation: Why add goto to Python?

It seemed like a good idea at the time...

Also useful for:

1. Translating goto-filled code to python
2. Finite state machines
3. Generating python code programmatically
4. Breaking out of a nested loop

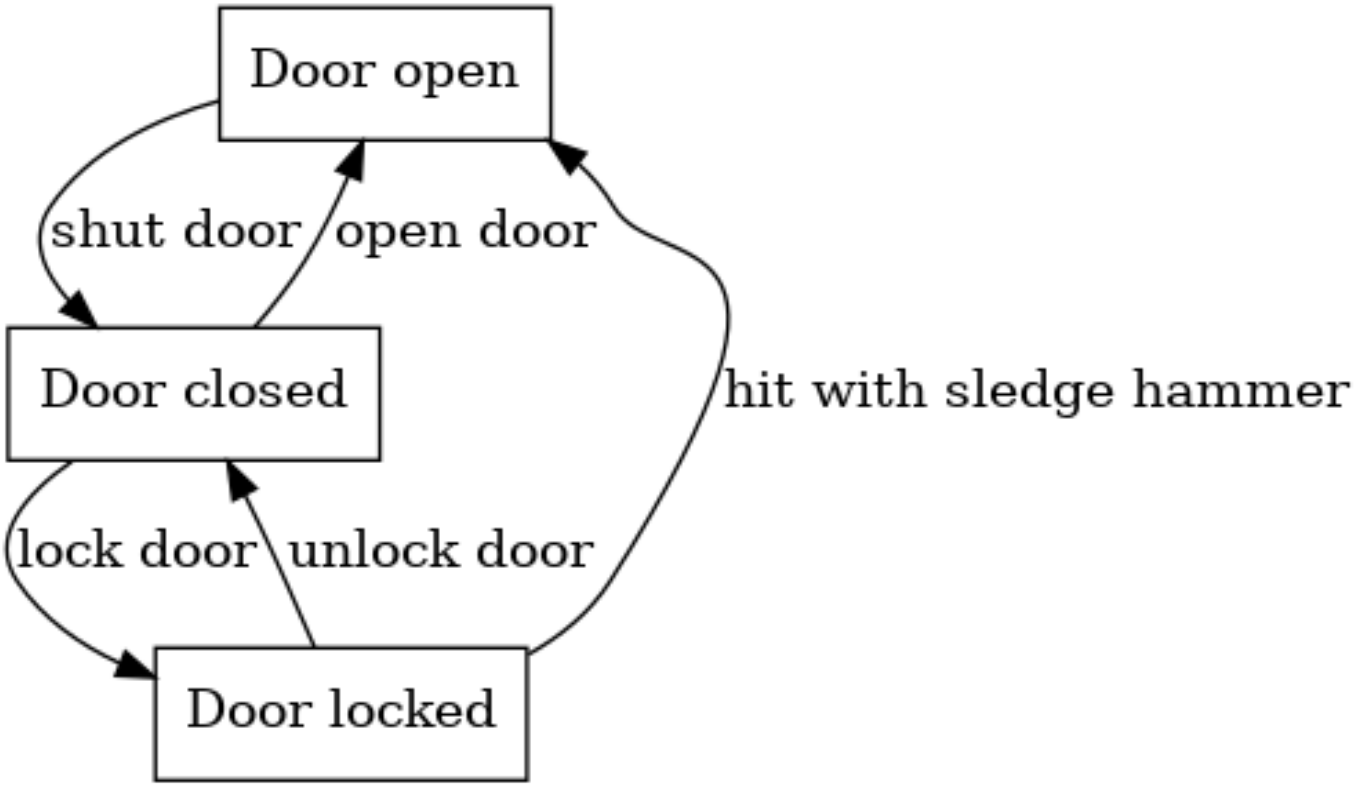
Motivation: An extract from `Hamurabi.bas` (1970's)

From David Ahl's *101 Basic Computer Games*, 1973

```
320 PRINT "HOW MANY ACRES DO YOU WISH TO BUY";
321 INPUT Q: IF Q<0 THEN 850
322 IF Y*Q<=S THEN 330
323 GOSUB 710
324 GOTO 320
330 IF Q=0 THEN 340
331 A=A+Q: S=S-Y*Q: C=0
334 GOTO 400
```

Motivation: State Machines

- A finite set of states. Transitions from one state to another on some input/event.
- Can model real world mechanisms: (Traffic lights, doors, game AI, etc.)



Mechanism: Goto using bytecode manipulation

Within a function that uses gotos:

1. (Mis)use attribute access for labels and goto.
   e.g. `label .found`
2. Extract a function's bytecode
3. Replace bytecode for `goto .found` with relative jump to label
4. Check for/Avoid/Mitigate various Bad Things
5. Replace the function's bytecode with the new goto-ified bytecode

Mechanism: Simple example function

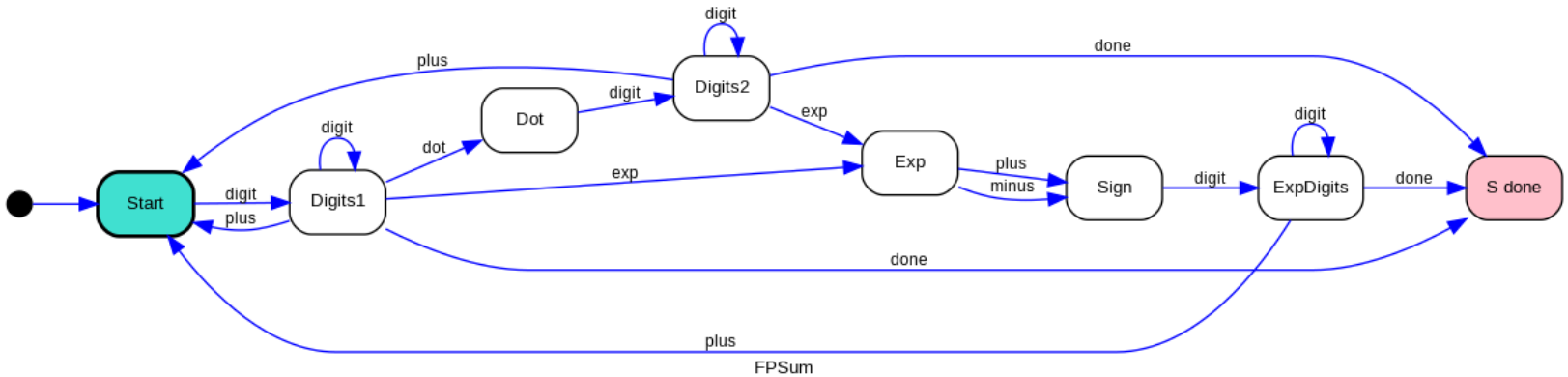
```
from goto import goto

@goto          # the goto decorator rewrites bytecode
def find_value(rectangle, n):
    for line in rectangle:
        for value in line:
            if value == n:
                rval = "found
                goto .found    # jump down to label
    rval = "not found"
    label .found               # a no-op at runtime
    # ... other code here ...
    return rval
```

Mechanism: Problems!

- At for-loop start, python pushes an iterator.
  Must pop iterators when jumping out of a loop
- Jumps more than 255 instructions require `EXTENDED_ARG` opcode
- Illegal:
  - Jump into a for-loop.
  - Jump into/out of `try`, `except`, `finally`, `with`
  - Multiple identical labels (or missing label)
  - Jump out of nested for loops more than 10 deep.
  - Very long jumps (65536 bytecodes)

Measurement: Non-trivial state machine

Recognises valid sum expression of floating point numbers



Example: `123.45+60e+3+123.45e-67+12`

Measurement: State machine implementation methods

Compare:

- Use existing `python-statemachine` library
- Use `for` loop with a `match` statement
- Use a regular expression:
  `r'\d+(\.\d+)?(e[+-]\d+)?(\+\d+(\.\d+)?(e[+-]\d+)?)*\$'`
- Use gotos for transition between states

Using valid 30,000 character strings

Measurement: results

Method	Recognition time	Slowdown
python-statemachine	504ms	180x
for-loop with match	21ms	7.5x
regular expression	3.1ms	1.1x
goto	2.8ms	1x

Musings

- Bytecode rewriting is powerful
- Non-standard: CPython only
- Fast but fragile: Bytecode can change between python versions
- Historic re-enactment and fast state machines
- Don't use in production!

Questions?