Singapore 2021

33RD INTERNATIONAL OLYMPIAD IN INFORMATICS SINGAPORE

Look Ma, Backtracking without Recursion

Tom Verhoeff
Netherlands

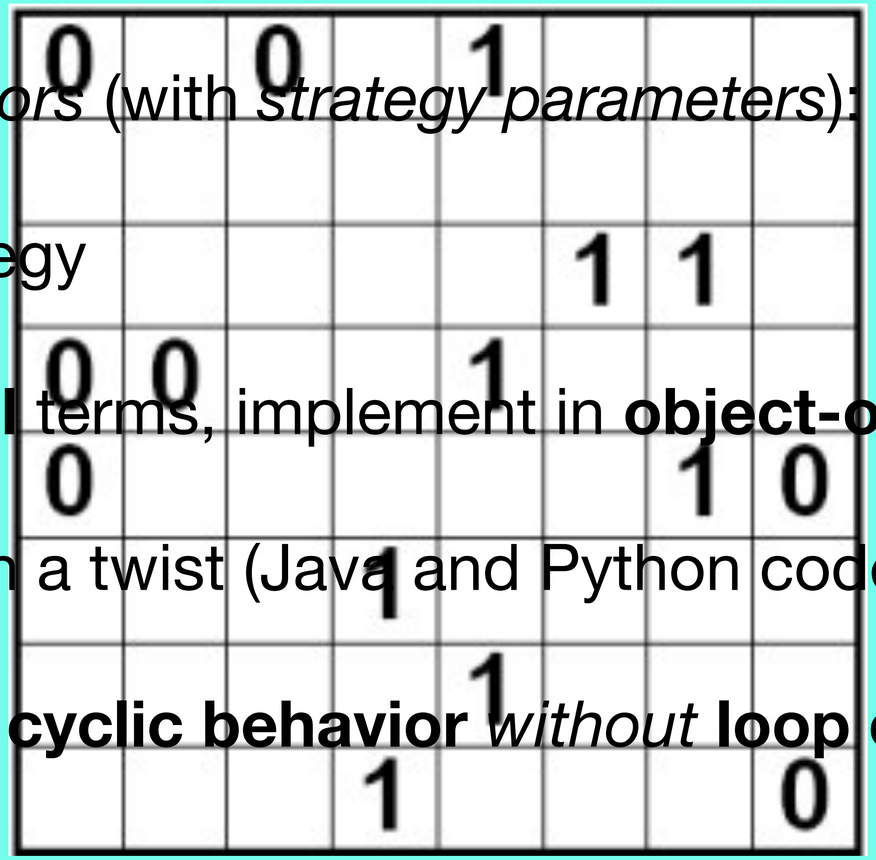
TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Insights

- How to discover **backtracking** *naturally* (when not even looking for it)
- How to implement backtracking **without recursion** (in traditional sense)
  - Also *no loop+explicit stack*, but using **self-application**
- Connect **functional** and **object-oriented** programming
  - OO design pattern for **partial function application**
- Power & dangers of **polymorphism**; need for *contractual reasoning*

Article Overview by Section

2. Simplified **binary puzzles** and reasoning **strategies**
   - Strategy *combinators* (with *strategy parameters*): repeat to fixpoint, pair
   - **Self-applied** strategy
3. Design in **functional** terms, implement in **object-oriented** language
   - It really works, with a twist (Java and Python code provided in Git repo)
4. Minimal example of **cyclic behavior** *without* **loop** or **recursion**



Self-Application

Self-Application Gone Wrong

- **Lambda Calculus**: *variables*, *lambda abstractions*, and *applications*
  - **λ x. x x** is the (nameless) function of variable x, which applies x to x
  - One computational rule: *substitution* (beta reduction)
    - **(λ x. T) U** β-reduces to **T[x := U]**, i.e. T with free x's replaced by U
  - **(λ x. x x)(λ x. x x)** β-reduces to …
    - **(λ x. x x)(λ x. x x)** i.e. an infinite loop

Self-Application in C++14

- **λ x. x x**
  - `[](auto x) -> int { return x(x); }  // generic lambda expression`
- **(λ x. x x)(λ x. x x)**
  - `[](auto x) -> int { return x(x); }`
    `( [](auto x) -> int { return x(x); } )  // segmentation fault`
- Compare x to DNA
  - Treated as **code**: *transcribe* (execute), cf. the call of **x** in **x(x)**
  - Treated as **data**: *replicate* (copy), cf. the argument **x** in **x(x)**

Self-Application Put to Good Use

```
cout <<
   [](auto dna, int n) -> long {
     return (n == 0) ? 1 : n * dna(dna, n - 1);
   }( [](auto dna, int n) -> long {
        return (n == 0) ? 1 : n * dna(dna, n - 1);
      },
      10
    );
```

- Blue: code executed (DNA transcribed)
- Orange: data copied (DNA replicated)

Lambdas – Closures – Objects

Polymorphic: could be from *subclass*

Subclass can *override* operator( )

```
class prefac {
  public:
    virtual long operator()(prefac * dna, int n) {
      return (n == 0) ? 1 : n * (*dna)(dna, n - 1);
    }
};
```

Contractual reasoning needed

```
prefac * pf = new prefac();
std::cout << (*pf)(pf, 3);
```

This creates a *cycle*

`(*pf)(dna, n) == n!`

if

- `n >= 0`, and
- `n > 0` implies `(*dna)(dna, n-1) == (n-1)!`

No recursion!

`pf` satisfies its contract, by induction

so, `(*pf)(pf, n) == n!`

Conclusion

My Related Articles

- "*A Master Class on Recursion*", in **LNCS** Vol. 11011, 2018.
  - The basics of recursion, and more examples of self-application
- "*An Enticing Environment for Programming*", in **IOI Journal**, Vol. 4, 2010.
  - *Tom's JavaScript Machine*, with *self-referential* programming challenge
- *From Callbacks to Design Patterns*, on **ResearchGate.net**, 2012.
  - OO programming techniques discovered from a functional view

33RD INTERNATIONAL
OLYMPIAD IN INFORMATICS
SINGAPORE

Thanks!
Questions?

Tom Verhoeff
Netherlands

TU/e EINDHOVEN
UNIVERSITY OF
TECHNOLOGY