PEP Scala (5)

Email:	christian.urban at kcl.ac.uk
Slides & Code:	KEATS

Plan for Today

Being Lazy

Polymorphic Types

Immutable OOP

Making Fun about Scala

How To calcululate 100 Mio Collatz Series?

```
(1L to 100_000_000).map(collatz).max
```

Polyorphic Types

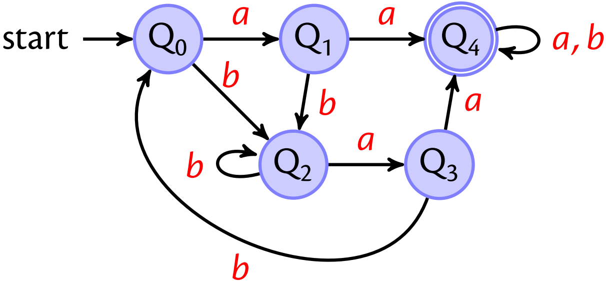
To be avoided:

```
def length_string_list(lst: List[String]): Int =
 lst match {
   case Nil => 0
   case x::xs => 1 + length_string_list(xs)
 }


def length_int_list(lst: List[Int]): Int =
 lst match {
   case Nil => 0
   case x::xs => 1 + length_int_list(xs)
 }
```

Polyorphic Types

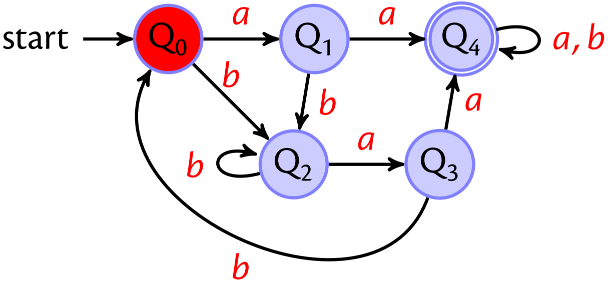
```
def length[A](lst: List[A]): Int = lst match {
  case Nil => 0
  case x::xs => 1 + length(xs)
}


length(List(”1”, ”2”, ”3”, ”4”))
length(List(1, 2, 3, 4))


def map[A, B](lst: List[A], f: A => B): List[B] =
 lst match {
   case Nil => Nil
   case x::xs => f(x)::map(xs, f)
 }
```

DFAs



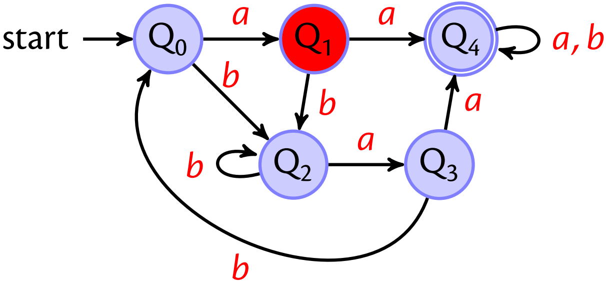
abaaa

DFAs



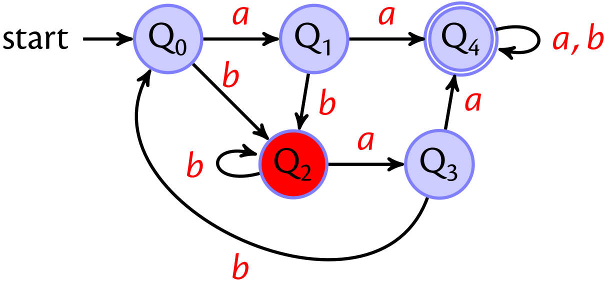
abaaa

DFAs



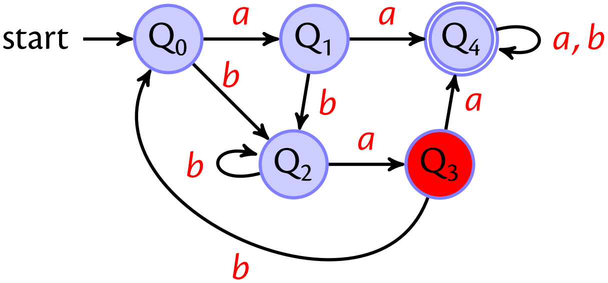
abaaa

DFAs



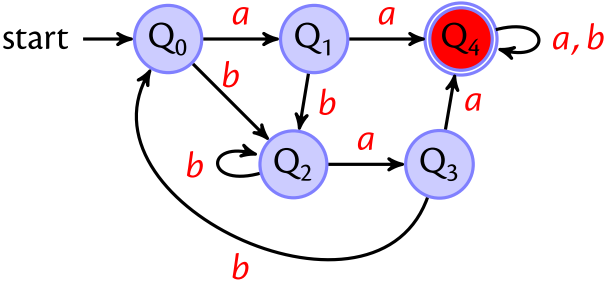
abaaa

DFAs



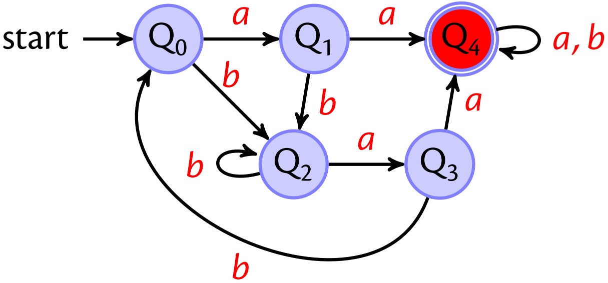
abaaa

DFAs



abaaa

DFAs



abaaa $\Rightarrow$ **yes**

DFAs

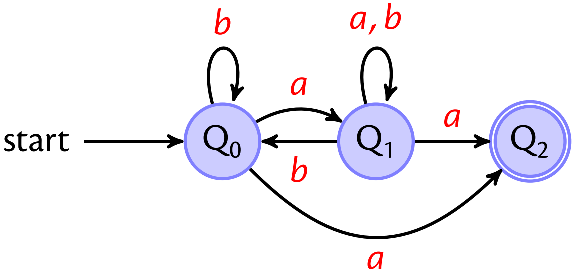
A **deterministic finite automaton** (DFA) consists of 5 things:

- an alphabet Σ
- a set of states Qs
- one of these states is the start state Q_0
- some states are accepting states F, and
- there is transition function δ

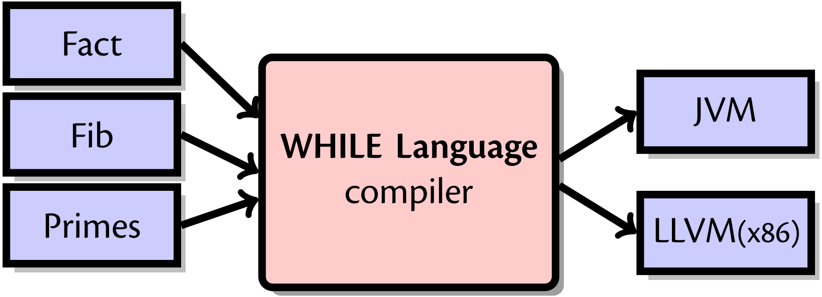
which takes a state and a character as arguments and produces a new state; this function might not be everywhere defined

$$A(\Sigma, Qs, Q_0, F, \delta)$$

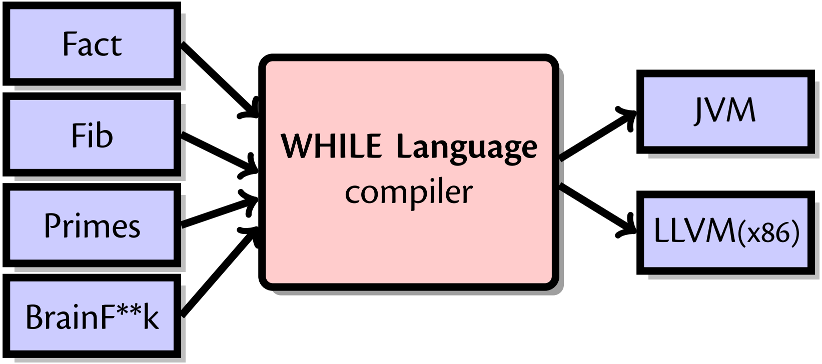
NFAs



Compilers 6CCS3CFL



Compilers 6CCS3CFL



Fact

Fib

Primes

BrainF**k

WHILE Language compiler

JVM

LLVM(x86)

Dijkstra on Testing

"Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence."

Proving Programs to be Correct

Theorem: There are infinitely many prime numbers.

Proof ...

similarly

Theorem: The program is doing what it is supposed to be doing.

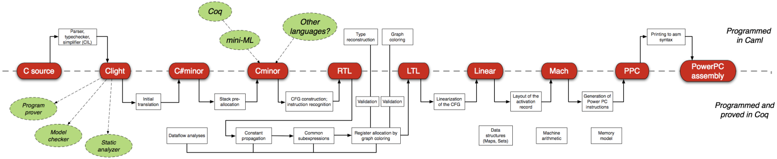
Long, long proof ...

This can be a gigantic proof. The only hope is to have help from the computer. 'Program' is here to be understood to be quite general (compiler, OS, ...).

Can This Be Done?

in 2011, verification of a small C-compiler (CompCert)

> “if my input program has a certain behaviour, then the compiled machine code has the same behaviour”
>
> is as good as `gcc -O1`, but much, much less buggy



Fuzzy Testing C-Compilers

tested GCC, LLVM and others by randomly generating C-programs

found more than 300 bugs in GCC and also many in LLVM (some of them highest-level critical)

about CompCert:

> "The striking thing about our CompCert results is that the middle-end bugs we found in all other compilers are absent. As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task."

seL4 / Isabelle

verified a microkernel operating system ($\approx$8000 lines of C code)

US DoD has competitions to hack into drones; they found that the isolation guarantees of seL4 hold up

CompCert and seL4 sell their code

seL4 / Isabelle

verified a microkernel c
lines of C code)

US DoD has competiti
found that the isolatio

CompCert and seL4 sell their code

Where to go on from here?

Martin Odersky (EPFL)...he is currently throwing out everything and starts again with the dotty compiler for Scala 3.0

Elm (`http://elm-lang.org`)...web applications with style

Haskell, Ocaml, Standard ML, Scheme, ...

Questions?

```
                               *
                              * *
                             *   *
                            * * * *
                           *       *
                          * *     * *
                         *   *   *   *
                        * * * * * * * *
                       *               *
                      * *             * *
                     *   *           *   *
                    * * * *         * * * *
                   *       *       *       *
                  * *     * *     * *     * *
                 *   *   *   *   *   *   *   *
                * * * * * * * * * * * * * * * *
               *                               *
              * *                             * *
             *   *                           *   *
            * * * *                         * * * *
           *       *                       *       *
          * *     * *                     * *     * *
         *   *   *   *                   *   *   *   *
        * * * * * * * *                 * * * * * * * *
       *               *               *               *
      * *             * *             * *             * *
     *   *           *   *           *   *           *   *
    * * * *         * * * *         * * * *         * * * *
   *       *       *       *       *       *       *       *
  * *     * *     * *     * *     * *     * *     * *     * *
 *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
++++++++[>+>++++<<-]>++>>
+<[-[>>+<<-]+>>]>+[-<<<[-
>[+[-]+>++>>>-<<]<[<]>>++
++++[<<+++++>>-]+<<++.[-]
<<]>.>+[>>]>+]
```

Mind-Blowing Programming Languages:
Scala ?