M4 by example

https://github.com/jkubin/m4_by_example

https://github.com/jkubin/L-system

OpenAlt 2018

Josef Kubín

Contents

- Introduction to M4
- Examples with M4 usage cases
- A brief history
- Prerequisites for understanding M4
- M4 fundamentals
- Common patterns in M4 code
- C preprocessor and M4
- Explanation of M4 examples

Introduction to M4

- M4 is general purpose macro-processor
- Not tied with a particular language or software
- GNU Build System – **autotools**
  configure, Makefile, config.h, …
- Sometimes used as a web template system
- SELinux policy macros
  (M4 replaced by CIL)
- Source code generators in my examples
  source.mc → target.{java,csv,xml,html,c,h,mc,m4,json,...}

A brief history

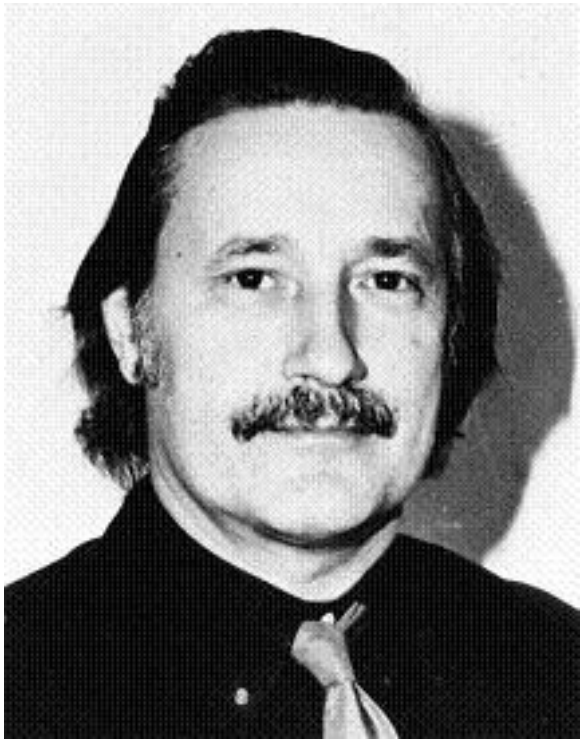
- **GPM** (**G**eneral **P**urpose **M**acrogenerator)
  - written by **Christopher Strachey**
  - GPM fit into 250 machine instructions!
- **M3** written by **Dennis Ritchie** for the AP-3 minicomputer
- **M4** for the original versions of UNIX, written by
  - **Dennis Ritchie**
  - **Brian Kernighan**
  - Jim E. Weythman

```
divert(-1)
define(…)
…
divert
```

  - Rick Becker
  - John Chambers
  - Doug McIlroy
  - …

Authors of M4

GPM (1965)

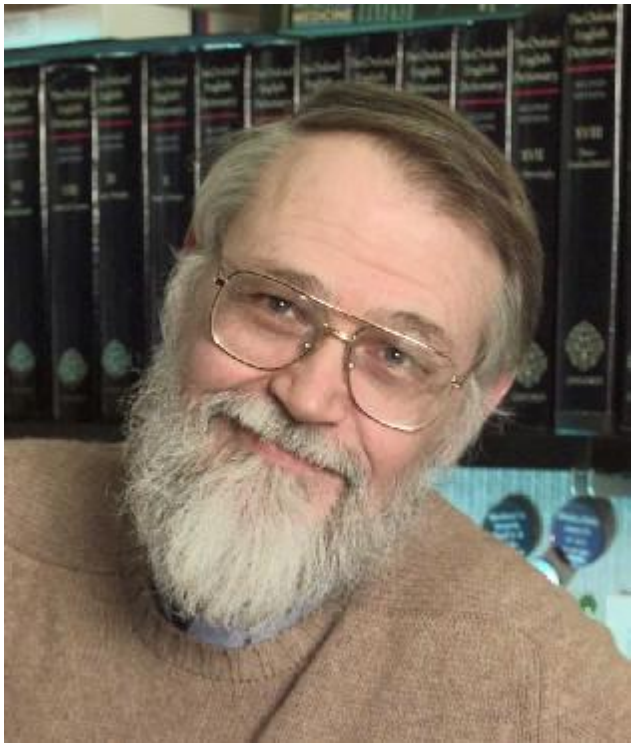


Christopher Strachey

C (1973), M3, M4 (1977)

Dennis Ritchie

C, M4, AWK



Brian Kernighan

and several other authors ...

Prerequisites for understanding M4

- A good plain text editor
- Foundation of **automata and grammars**
- GNU Make
- A lot of working, commented examples
- A lot of time to learn and understand
  - If you have IT education you have no time
  - If you have no IT education you have a lot of time
  - https://www.gnu.org/software/m4/manual/m4.html
    - Describes each keyword in detail, but **NO** fundamentals (context-free grammar, automata and **β** rules)
      - Therefore M4 is nearly forgotten

Writing M4 scripts

- Abbreviations for M4 keywords (avoid bracket hell)
- Placeholders **%%%** in keywords arguments (Vim config)

```
~/.vim/ftplugin/m4/m4_maps.vim
...
inoremap <buffer> ;; <c-o>/%%%<cr><del><del><del>
...
```

- Familiarity with Vim text-objects (keyboard shortcuts)
  - See **:help text-objects**
    ci(
    da(
    ci[
    ya[
    ...
- **%** is a shortcut to jump over (…) or […]
  - See **:help %**
- Shortcuts to effectively comment-out M4 code

Formal Grammar (Chomsky type)

G = (N, Σ, P, S)

N: fin. set of non-terminal symbols

Σ: fin. set of terminal symbols

$\mathbf{N} \cap \mathbf{\Sigma} = \mathbf{\varnothing}$

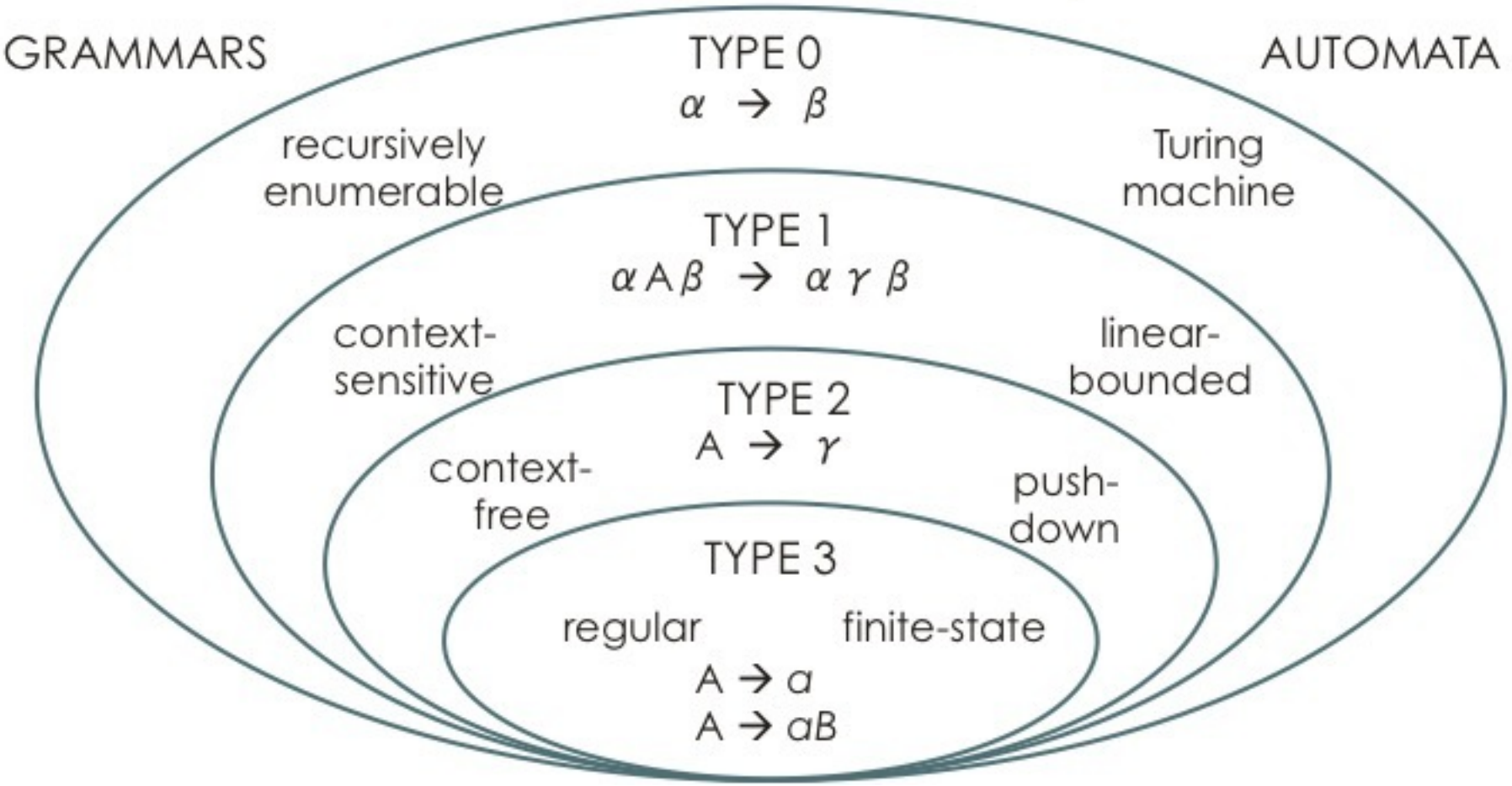
P: fin. set of production (rewrite) rules

$\mathbf{(N \cup \Sigma)^* N (N \cup \Sigma)^* \rightarrow (N \cup \Sigma)^*}$

S: is the start symbol

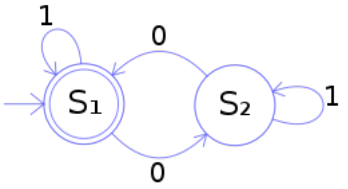
$\mathbf{S \in N}$

Chomsky Hierarchy



Fundamentals of M4

- **Context-free grammar (CFG)**
  - **Σ**: Terminal symbols
  - **N**: Non-terminal symbols
  - **P**: Production (rewrite) rules
    - **A → β**
- **Every M4 define(...)** can be seen as **a CFG production rule**
  - **Even all built-in keywords!**
- **Branching** by macro expansion, without **ifelse(...)** keyword
- **Automata** (and β rules)
  - Significantly reduces M4 code complexity!
- **Stacks**
  - Nearly unlimited number and size (limited by your RAM size)
- **Buffers** (queues) for processed data
  - Nearly unlimited size (limited by your HDD)
  - Keywords: **divert(N)**, **undivert(N)**
  - Total number of buffers is $2^{31} - 1$



Production rules in M4

- Production rules in context-free grammar (type 2)
  P: A → β
  A ∈ N
  β ∈ (N ∪ Σ)*

- Productions rules in M4
  define(`**A**', `**β**')

Production rules in M4

$A \rightarrow \beta$

Σ = {a, ε}; N = {S, A};

A → ε

```
define(`A') or define(`A', ``')
```

A → a

```
define(`A', `a')
```

A → aA

```
define(`A', `a``'A')
```

A → Aa

```
define(`A', `A``'a')
```

S → aSb

```
define(`S', `a``'S``'b')
```

Symbols in M4

- Terminal symbols
  - Are symbols that go out to **stdout**, **buffers**, **/dev/null**
- Non-terminal symbols (or macro names) **`[_a-zA-Z][_a-zA-Z0-9]*`**
  - Are **recursively** expanded to terminal symbols
    - Unwanted expansion is protected by quotation marks ` ', [ ], …
    - See keyword: **changequote**
    - **changequote([,])** ← change quotation marks to [ ]
    - **changequote(,)** ← completely disables quotation marks
- No data types, symbols/tokens only
- Everything has global scope
  - Non-terminal symbol temporarily hide **pushdef()**

Two types of finite loops in M4

A → β

1. right-recursive (A → aA | a | ε)

```
define(`A', `ifelse(`$1', `0', `0', `$1$0(decr($1))')')
A(9)
9876543210
define(`B', `ifelse(`$*', `', `', `$1`'$0(shift($@))')')
B(a, b, c, d)
abcd
```

2. left-recursive (A → Aa | a | ε)

generally not recommended (if produces hundreds of MiB)

```
define(`A', `ifelse(`$1', `0', `0', `$0(decr($1))$1')')
A(9)
0123456789
define(`B', `ifelse(`$*', `', `', `$0(shift($@))$1')')
B(a, b, c, d)
dcba
```

Two types of finite loops in M4

A → β

2.1 left-recursive (S → aSb | ε)

```
define(`S', `ifelse(`$1', `0', ``', `a`'$0(decr($1))b')')
S(9)
aaaaaaaaabbbbbbbbb
```

Branching in M4

A → β

```
$ m4
changequote([,])
define([A_0], [zero])
define([A_1], [one])
define([A_2], [two])
define([A_3], [three])
define([A_4], [four])
define([A_5], [five])
define([A_6], [six])
define([A_7], [seven])
define([A_8], [eight])
define([A_9], [nine])
define([A], [A_$1])
A(3)
three
#because: A(3) → A_$1 → A_3 → three
define([A], [$0_$1])
A(3)
three
#because: A(3) → $0_$1 → A_3 → three
```

Branching in M4

A → β

```
...
define([A], [A_[]eval($1 % 10)])
A(33)
A_3
#because: A(33) → A_[]3 → A_3
define([B], [$1])
B(A(33))
three
#because: B(A(33)) → B(A_3) → A_3 → three
define([A], [B(A_[]eval($1 % 10))])
A(33)
three
#because: A(33) → B(A_3) → A_3 → three
```

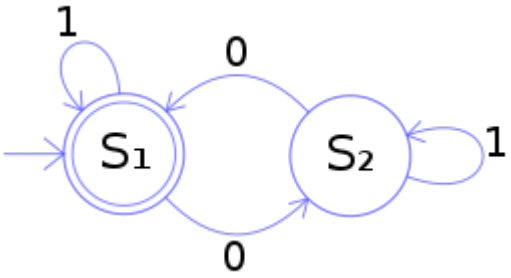
Using **ifelse(…)**

1. Small branches
2. End condition of a loop
3. Automaton (transition/evaluation)

NOT FOR GENERAL PURPOSE BRANCHING!

- Insufficient code decomposition
  - Big ball of mud
    https://en.wikipedia.org/wiki/Anti-pattern

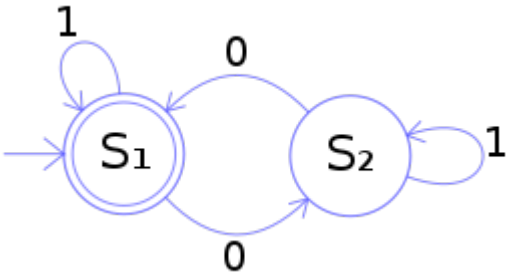
Automaton in M4



https://github.com/jkubin/m4_by_example/automaton/even_0.m4

```
$ cat automaton/even_0.m4
# β rule
define([S1], [
        ifelse(
                [$*], [0], [define([$0], defn([S2]))],
                [$*], [1], [],
                [NOT_IN_ALPHABET($@)]
        )
])
# β rule
define([S2], [
        ifelse(
                [$*], [0], [define([$0], defn([S1]))],
                [$*], [1], [],
                [NOT_IN_ALPHABET($@)]
        )
])
# A → β
define([EVEN], defn([S1]))
```

Automaton in M4




```
...
m4wrap([
        divert(0)dnl
ifelse(defn([EVEN]), defn([S1]), [accept], [reject])
])

EVEN(0)
EVEN(1)
EVEN(0)
EVEN(1)

$ m4 common.m4 even_0.m4
accept
```

Automaton in M4

```
$ m4
changequote([,])
#      __________        _________
# --->/ EPSILON \---->/ SHADOW \
#     \__________/<----\_________/
#
# β rules
define([SHADOW], [ class="shadow"define([$0], defn([EPSILON]))])
define([EPSILON], [define([$0], defn([SHADOW]))])

# A → β
define([CLASS], defn([EPSILON]))

<tr[]CLASS>
<tr>
<tr[]CLASS>
<tr class="shadow">
<tr[]CLASS>
<tr>
<tr[]CLASS>
<tr class="shadow">
```

Automaton in M4

```
$ m4
changequote([,])
#      __________       ___
# --->/ <table> \--->/ ε \---,
#     \__________/    \___/<--'
#
define([TABLE], [<table>
define([$0])])

TABLE
<table>
TABLE
TABLE
TABLE
```

Automaton in M4

```
$ m4
changequote([,])
#       ___        __________
# --->/ ε \--->/ <table> \---,
#      \___/     \__________/<--'
#
define([TABLE], [define([$0], [<table>
])])

TABLE
TABLE
<table>
TABLE
<table>
TABLE
<table>
```

Special arguments to macros $#, $*, $@, $0, $1, $2, $3, …

- Similar meaning in the following languages
  - Bash
  - Perl
  - AWK
  - bc

Special arguments to macros **$#**, $*, $@, $0, $1, $2, $3, ...

```
$ m4
define(`A', `$#')
A(a, b, c)
3
A
0
A()
1
```

Special arguments to macros $#, $*, $@, $0, $1, $2, $3, ...

```
$ m4
define(`A', ``                  A(a, b, c)
$#                              3
$*                              a,b,c
$@                              `a',`b',`c'
$0                              A
$1                              a
$2                              b
$3                              c
…                               …
'')
```

Arguments in M4
$#, $*, $@, **$0**, $1, $2, $3, …

```
$ m4
# β rule
define(`BETA', `$0_SUFFIX')
BETA
BETA_SUFFIX

define(`A', defn(`BETA'))
define(`B', defn(`BETA'))
A
A_SUFFIX
B
B_SUFFIX
```

Counters in M4

```
$ m4
changequote([,])
# init counter
define([COUNTER], 1)
COUNTER
1
define([COUNTER], incr(COUNTER))
COUNTER
2
define([COUNTER], incr(COUNTER))
COUNTER
3
...
```

Counters in M4

```
$ m4
changequote([,])
# β rule
define([COUNT_UP], [dnl
    define([$0_COUNTER], $1)dnl
    define([$0], [$0_COUNTER[]define([$0_COUNTER], incr($0_COUNTER))])dnl
])
# β rule
define([COUNT_DOWN], [dnl
    define([$0_COUNTER], $1)dnl
    define([$0], [$0_COUNTER[]define([$0_COUNTER], decr($0_COUNTER))])dnl
])
# A → β
define([A], defn([COUNT_DOWN]))
define([B], defn([COUNT_UP]))
define([C], defn([COUNT_DOWN]))
define([D], defn([COUNT_UP]))
```

Counters in M4

```
...
# init counters
A(3)
B(-3)
C(0)
D(2147483645)
A B C D
3 -3 0 2147483645
A B C D
2 -2 -1 2147483646
A B C D
1 -1 -2 2147483647
A B C D
0 0 -3 -2147483648
A B C D
-1 1 -4 -2147483647
```

Stack(s) in M4

```
$ m4
define(`A', `Alice')
pushdef(`A', `Annie')
pushdef(`A', `Amy')
A
Amy
popdef(`A')
A
Annie
popdef(`A')
A
Alice
popdef(`A')
A
A
ifdef(`A', `yes', `no')
no
```

Stack(s) in M4
How to define “locals”

```
$ m4
changequote([,])
define([LAST], [pushdef([$0], [$$#])$0($@)[]popdef([$0])])
LAST(a, b, c)
c
#because: pushdef([$0], [$$#])
# $$# → $3
# pushdef([LAST], [$3])
# LAST(a,b,c) → c

LAST([a, b, c], [x, y, z])
x, y, z

define([LAST_BUT_ONE], [pushdef([$0], $decr($#))$0($@)[]popdef([$0])])
LAST_BUT_ONE(a, b, c)
b
#because: $decr($#) → $decr(3) → $2 → b
```

Temporary buffers/queues in M4

divert(-1) send to ***/dev/*null**

divert(0) send to **stdout**

divert(1) send to buffer #1

divert(2) send to buffer #2

divert(3) send to buffer #3

…

divert(2147483647) send to buffer #2147483647

Dumping the buffers (destructive)

undivert(1) dump buffer #1 to the ***current*** data stream

undivert(2) dump buffer #2 to the ***current*** data stream

...

undivert(2147483647) recall from #2147483647

undivert without args dumps all buffers in numeric order 1 to 2147483647

Big advantage!

Dumping the buffers on M4 exit

- Undivert files to the **current** data stream

```
$ m4
divert(-1)
divert(2)
undivert(
        `foo.txt',
        `bar.txt',
        `baz.txt',
)dnl
divert(1)dnl
this is different kind of include (no expansion)
Ctrl-d
```

C preprocessor (CPP) and M4

- CPP is not recursive, **NO** iteration
- CPP cannot expand following macro
  - char name[] = "... CPP_MACRO ...";
  - But M4 can
    - CPP solves it by stringification
- char letter = 'A';
  - M4 ignore: **'**
    - Which is right M4 quotation mark by default
  - M4 recognize: `
    - Left quotation mark by default
    - A rare occurence in C source code (can be hidden as \x68)

C preprocessor (CPP) and M4

- Most CPP keywords are different from M4
  - #**include** <...>
  - #**define** ...
  - #if defined(...)
  - #if, #**ifdef**, #ifndef, #else, #elif a #endif
- Lines with CPP directives starts by #
  - Comments for M4
- `#'define CPP_MACRO ...
  - Visible to M4
  - But “**define**” CPP keyword is M4 keyword!
    - Ignored, because of space (not round bracket) next to

C preprocessor (CPP) and M4

https://github.com/jkubin/m4_by_example/array

```
# ← comments the rest of line for M4
`#' ← is NOT comment for M4
#define define(a, b, c)     comment for M4
#if  …
#include <stdio.h>          comment for M4
`#'define define(a, b, c)   processed by M4, error
`#'define define (a, b, c)   processed by M4, no error
```

Bug or Feature?

https://mbreen.com/m4.html (Michael Breen)

```
$ m4
define(`eng', `engineering')
...
substr(`engineer', 0, 3)
engineering
```

Feature!

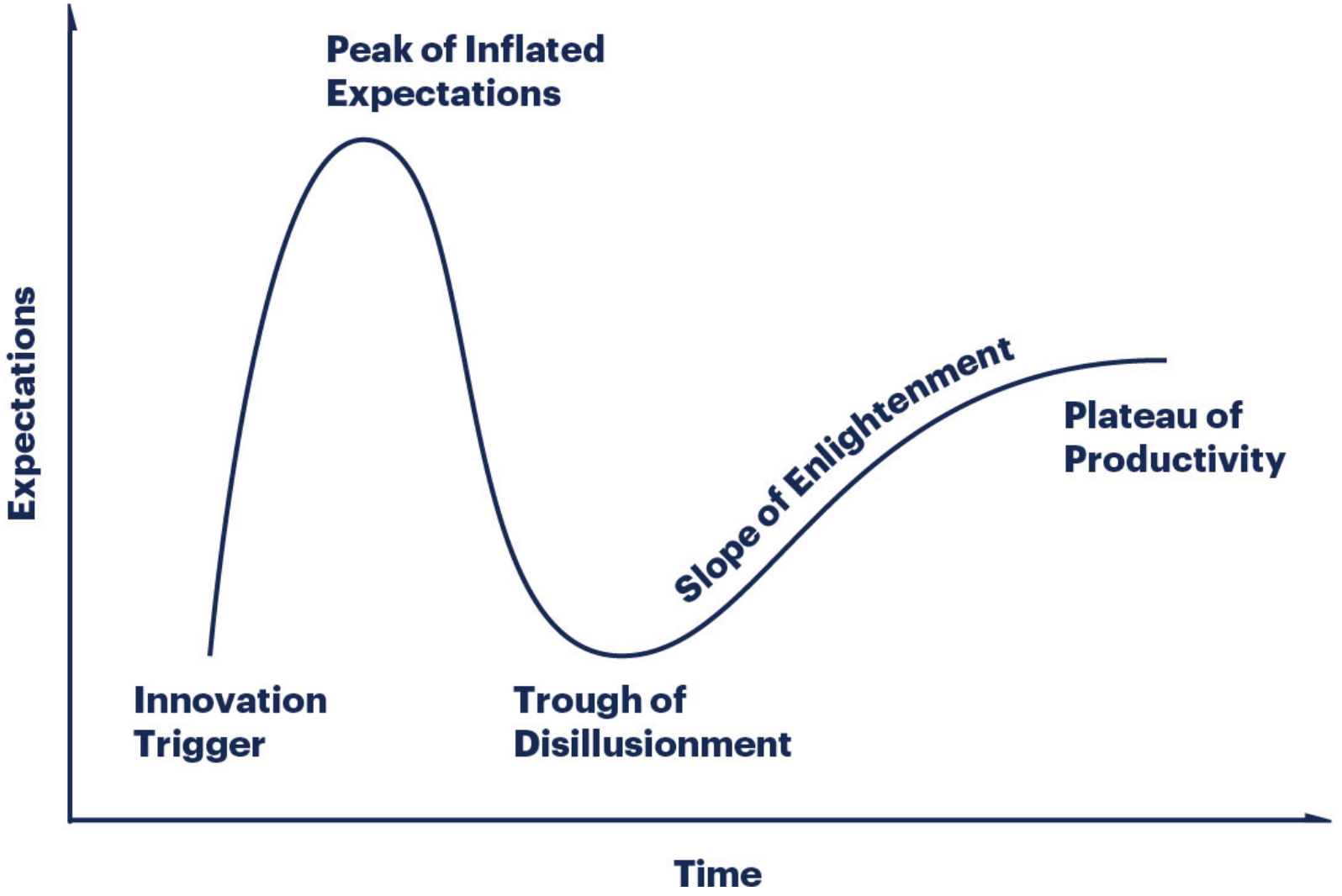
https://mbreen.com/m4.html (Michael Breen)

```
$ m4
define(`eng', `engineering')
...
substr(`engineer', 0, 3)
engineering
```

- Because of context-free grammar rule:
  A → β
- Therefore:
  eng → engineering

Hype cycle

- Angry at the M4? Keep calm and carry on!



References

- https://www.gnu.org/software/m4/manual/m4.html
- https://www.gartner.com/en/research/methodologies/gartner-hype-cycle
- http://mbreen.com/m4.html
- https://www.cs.cmu.edu/~mihaib/kernighan-interview/
- https://pc.zoznam.sk/novinka/zomrel-tvorca-unixu-jazyka-c
- https://www.computerhope.com/people/christopher_strachey.htm

Děkuji za pozornost!

https://github.com/jkubin/m4_by_example

https://github.com/jkubin/L-system

Nezapomeňte vyplnit anketu!
OpenAlt 2018

Josef Kubín