Parsing with Derivatives (Yacc is Dead)

Matt Might & David Darais
University of Utah
matt.might.net

WARNING

Not yet true.

What is parsing?

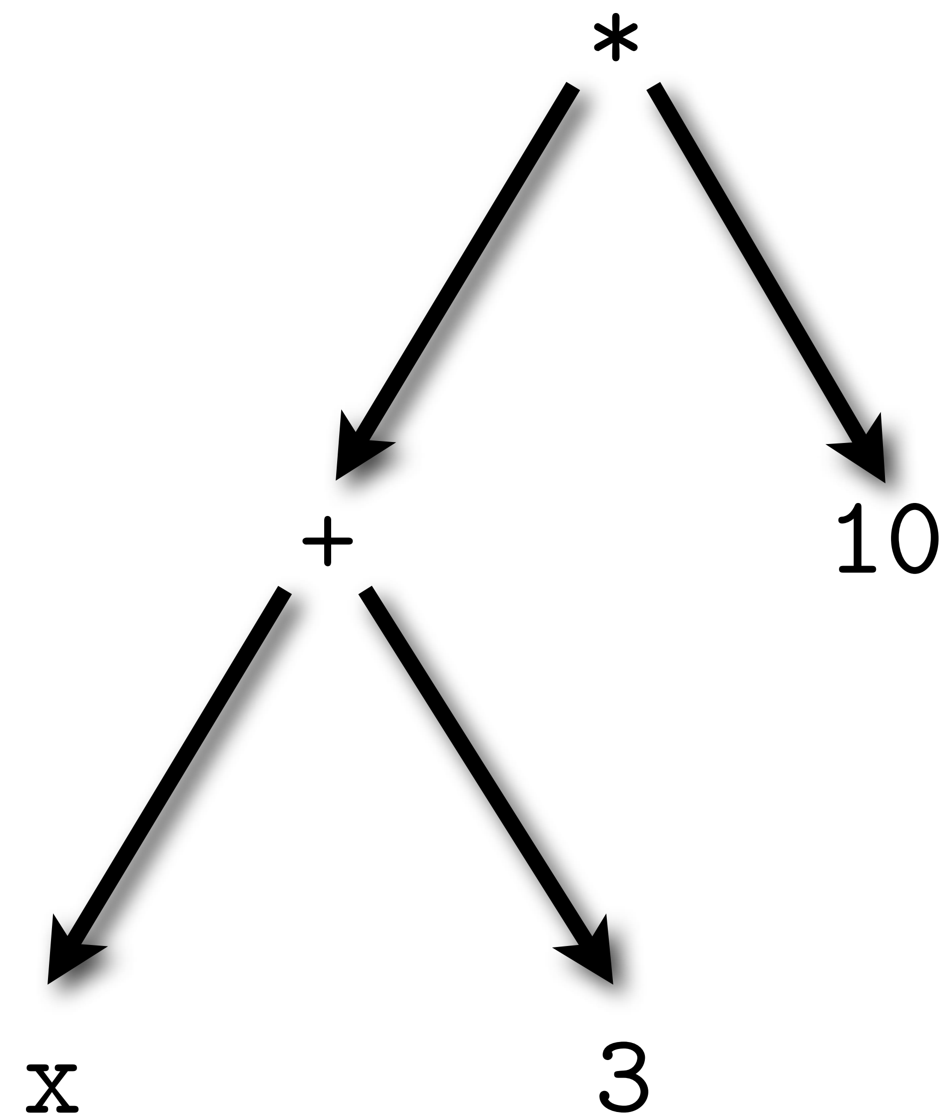
Unstructure to structure.

A transformation from a sequence to a tree.

```
(x + 3) * 10
```

```
(x + 3) * 10
```

```
(x + 3) * 10
```


*
+
10
x
3

“Parsing is ubiquitous.”

Trevor Jim

Compilers

Interpreters

Syntax highlighting

Natural language

Network protocols

Command lines

Configuration files

Serialization

Query languages

API design

Compilers

Interpreters

Syntax highlighting

Natural language

Network protocols

Command lines

Configuration files

Serialization

Query languages

API design

RFC 2616 (HTTP 1.1)

2.1 Augmented BNF

All of the mechanisms specified in this document are described in both prose and an augmented Backus-Naur Form (BNF) similar to that used by RFC 822 [9]. Implementors will need to be familiar with the notation in order to understand this specification. The augmented BNF includes the following constructs:

```
media-type   = type "/" subtype *( ";" parameter )
type         = token
subtype      = token
```

```
HTTP-Version = "HTTP" "/" 1*DIGIT "." 1*DIGIT
```

```
LWS = [CRLF] 1*( SP | HT )
```

```
separators = "(" | ")" | "<" | ">" | "@"
           | "," | ";" | ":" | "\" | <">
           | "/" | "[" | "]" | "?" | "="
           | "{" | "}" | SP | HT
```

```
http_URL = "http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]
```

```
Chunked-Body   = *chunk
                 last-chunk
                 trailer
                 CRLF

chunk          = chunk-size [ chunk-extension ] CRLF
                 chunk-data CRLF
chunk-size     = 1*HEX
last-chunk     = 1*("0") [ chunk-extension ] CRLF

chunk-extension= *( ";" chunk-ext-name [ "=" chunk-ext-val ] )
chunk-ext-name = token
chunk-ext-val  = token | quoted-string
chunk-data     = chunk-size(OCTET)
trailer        = *(entity-header CRLF)
```

```
HTTP-date    = rfc1123-date | rfc850-date | asctime-date
rfc1123-date = wkday "," SP date1 SP time SP "GMT"
rfc850-date  = weekday "," SP date2 SP time SP "GMT"
asctime-date = wkday SP date3 SP time SP 4DIGIT
date1        = 2DIGIT SP month SP 4DIGIT
               ; day month year (e.g., 02 Jun 1982)
date2        = 2DIGIT "-" month "-" 2DIGIT
               ; day-month-year (e.g., 02-Jun-82)
date3        = month SP ( 2DIGIT | ( SP 1DIGIT ))
               ; month day (e.g., Jun  2)
time         = 2DIGIT ":" 2DIGIT ":" 2DIGIT
               ; 00:00:00 - 23:59:59
wkday        = "Mon" | "Tue" | "Wed"
             | "Thu" | "Fri" | "Sat" | "Sun"
weekday      = "Monday" | "Tuesday" | "Wednesday"
             | "Thursday" | "Friday" | "Saturday" | "Sunday"
month        = "Jan" | "Feb" | "Mar" | "Apr"
             | "May" | "Jun" | "Jul" | "Aug"
             | "Sep" | "Oct" | "Nov" | "Dec"
```

RFC 3501 (IMAPv4)

```
address         = "(" addr-name SP addr-adl SP addr-mailbox SP
                  addr-host ")"

addr-adl        = nstring
                    ; Holds route from [RFC-2822] route-addr if
                    ; non-NIL

addr-host       = nstring
                    ; NIL indicates [RFC-2822] group syntax.
                    ; Otherwise, holds [RFC-2822] domain name

addr-mailbox    = nstring
                    ; NIL indicates end of [RFC-2822] group; if
                    ; non-NIL and addr-host is NIL, holds
                    ; [RFC-2822] group name.
                    ; Otherwise, holds [RFC-2822] local-part
                    ; after removing [RFC-2822] quoting

addr-name       = nstring
                    ; If non-NIL, holds phrase from [RFC-2822]
                    ; mailbox after removing [RFC-2822] quoting

append          = "APPEND" SP mailbox [SP flag-list] [SP date-time] SP
                  literal

astring         = 1*ASTRING-CHAR / string

ASTRING-CHAR   = ATOM-CHAR / resp-specials

atom            = 1*ATOM-CHAR

ATOM-CHAR       = <any CHAR except atom-specials>

atom-specials   = "(" / ")" / "{" / SP / CTL / list-wildcards /
                  quoted-specials / resp-specials

authenticate    = "AUTHENTICATE" SP auth-type *(CRLF base64)

auth-type       = atom
                    ; Defined by [SASL]

base64          = *(4base64-char) [base64-terminal]

base64-char     = ALPHA / DIGIT / "+" / "/"
                    ; Case-sensitive

base64-terminal = (2base64-char "==") / (3base64-char "=")

body            = "(" (body-type-1part / body-type-mpart) ")"

body-extension  = nstring / number /
                   "(" body-extension *(SP body-extension) ")"
                    ; Future expansion.  Client implementations
                    ; MUST accept body-extension fields.  Server
                    ; implementations MUST NOT generate
                    ; body-extension fields except as defined by
                    ; future standard or standards-track
                    ; revisions of this specification.

body-ext-1part  = body-fld-md5 [SP body-fld-dsp [SP body-fld-lang
                  [SP body-fld-loc *(SP body-extension)]]]
                    ; MUST NOT be returned on non-extensible
                    ; "BODY" fetch

body-ext-mpart  = body-fld-param [SP body-fld-dsp [SP body-fld-lang
                  [SP body-fld-loc *(SP body-extension)]]]
                    ; MUST NOT be returned on non-extensible
                    ; "BODY" fetch

body-fields     = body-fld-param SP body-fld-id SP body-fld-desc SP
                  body-fld-enc SP body-fld-octets

body-fld-desc   = nstring

body-fld-dsp    = "(" string SP body-fld-param ")" / nil

body-fld-enc    = (DQUOTE ("7BIT" / "8BIT" / "BINARY" / "BASE64"/
                  "QUOTED-PRINTABLE") DQUOTE) / string

body-fld-id     = nstring

body-fld-lang   = nstring / "(" string *(SP string) ")"

body-fld-loc    = nstring

body-fld-lines  = number

body-fld-md5    = nstring

body-fld-octets = number

body-fld-param  = "(" string SP string *(SP string SP string) ")" / nil

body-type-1part = (body-type-basic / body-type-msg / body-type-text)
                  [SP body-ext-1part]

body-type-basic = media-basic SP body-fields
                    ; MESSAGE subtype MUST NOT be "RFC822"

body-type-mpart = 1*body SP media-subtype
                  [SP body-ext-mpart]

body-type-msg   = media-message SP body-fields SP envelope
                  SP body SP body-fld-lines

body-type-text  = media-text SP body-fields SP body-fld-lines

capability      = ("AUTH=" auth-type) / atom
                    ; New capabilities MUST begin with "X" or be
                    ; registered with IANA as standard or
                    ; standards-track

capability-data = "CAPABILITY" *(SP capability) SP "IMAP4rev1"
                  *(SP capability)
                    ; Servers MUST implement the STARTTLS, AUTH=PLAIN,
                    ; and LOGINDISABLED capabilities
                    ; Servers which offer RFC 1730 compatibility MUST
                    ; list "IMAP4" as the first capability.

CHAR8           = %x01-ff
                    ; any OCTET except NUL, %x00

command         = tag SP (command-any / command-auth / command-nonauth /
                  command-select) CRLF
                    ; Modal based on state

command-any     = "CAPABILITY" / "LOGOUT" / "NOOP" / x-command
                    ; Valid in all states

command-auth    = append / create / delete / examine / list / lsub /
                  rename / select / status / subscribe / unsubscribe
                    ; Valid only in Authenticated or Selected state

command-nonauth = login / authenticate / "STARTTLS"
                    ; Valid only when in Not Authenticated state

command-select  = "CHECK" / "CLOSE" / "EXPUNGE" / copy / fetch / store /
                  uid / search
                    ; Valid only when in Selected state

continue-req    = "+" SP (resp-text / base64) CRLF

copy            = "COPY" SP sequence-set SP mailbox

create          = "CREATE" SP mailbox
                    ; Use of INBOX gives a NO error

date            = date-text / DQUOTE date-text DQUOTE

date-day        = 1*2DIGIT
                    ; Day of month

date-day-fixed  = (SP DIGIT) / 2DIGIT
                    ; Fixed-format version of date-day

date-month      = "Jan" / "Feb" / "Mar" / "Apr" / "May" / "Jun" /
                  "Jul" / "Aug" / "Sep" / "Oct" / "Nov" / "Dec"

date-text       = date-day "-" date-month "-" date-year

date-year       = 4DIGIT

date-time       = DQUOTE date-day-fixed "-" date-month "-" date-year
                  SP time SP zone DQUOTE

delete          = "DELETE" SP mailbox
                    ; Use of INBOX gives a NO error

digit-nz        = %x31-39
                    ; 1-9

envelope        = "(" env-date SP env-subject SP env-from SP
                  env-sender SP env-reply-to SP env-to SP env-cc SP
                  env-bcc SP env-in-reply-to SP env-message-id ")"

env-bcc         = "(" 1*address ")" / nil

env-cc          = "(" 1*address ")" / nil

env-date        = nstring

env-from        = "(" 1*address ")" / nil

env-in-reply-to = nstring

env-message-id  = nstring

env-reply-to    = "(" 1*address ")" / nil

env-sender      = "(" 1*address ")" / nil

env-subject     = nstring

env-to          = "(" 1*address ")" / nil

examine         = "EXAMINE" SP mailbox

fetch           = "FETCH" SP sequence-set SP ("ALL" / "FULL" / "FAST" /
                  fetch-att / "(" fetch-att *(SP fetch-att) ")")

fetch-att       = "ENVELOPE" / "FLAGS" / "INTERNALDATE" /
                  "RFC822" [".HEADER" / ".SIZE" / ".TEXT"] /
                  "BODY" ["STRUCTURE"] / "UID" /
                  "BODY" section ["<" number "." nz-number ">"] /
                  "BODY.PEEK" section ["<" number "." nz-number ">"]

flag            = "\Answered" / "\Flagged" / "\Deleted" /
                  "\Seen" / "\Draft" / flag-keyword / flag-extension
                    ; Does not include "\Recent"

flag-extension  = "\" atom
                    ; Future expansion.  Client implementations
                    ; MUST accept flag-extension flags.  Server
                    ; implementations MUST NOT generate
                    ; flag-extension flags except as defined by
                    ; future standard or standards-track
                    ; revisions of this specification.

flag-fetch      = flag / "\Recent"

flag-keyword    = atom

flag-list       = "(" [flag *(SP flag)] ")"

flag-perm       = flag / "\*"

greeting        = "*" SP (resp-cond-auth / resp-cond-bye) CRLF

header-fld-name = astring

header-list     = "(" header-fld-name *(SP header-fld-name) ")"

list            = "LIST" SP mailbox SP list-mailbox

list-mailbox    = 1*list-char / string

list-char       = ATOM-CHAR / list-wildcards / resp-specials

list-wildcards  = "%" / "*"

literal         = "{" number "}" CRLF *CHAR8
                    ; Number represents the number of CHAR8s

login           = "LOGIN" SP userid SP password

lsub            = "LSUB" SP mailbox SP list-mailbox

mailbox         = "INBOX" / astring
                    ; INBOX is case-insensitive.  All case variants of
                    ; INBOX (e.g., "iNbOx") MUST be interpreted as INBOX
                    ; not as an astring.  An astring which consists of
                    ; the case-insensitive sequence "I" "N" "B" "O" "X"
                    ; is considered to be INBOX and not an astring.
                    ;  Refer to section 5.1 for further
                    ; semantic details of mailbox names.

mailbox-data    =  "FLAGS" SP flag-list / "LIST" SP mailbox-list /
                   "LSUB" SP mailbox-list / "SEARCH" *(SP nz-number) /
                   "STATUS" SP mailbox SP "(" [status-att-list] ")" /
                   number SP "EXISTS" / number SP "RECENT"

mailbox-list    = "(" [mbx-list-flags] ")" SP
                   (DQUOTE QUOTED-CHAR DQUOTE / nil) SP mailbox

mbx-list-flags  = *(mbx-list-oflag SP) mbx-list-sflag
                  *(SP mbx-list-oflag) /
                  mbx-list-oflag *(SP mbx-list-oflag)

mbx-list-oflag  = "\Noinferiors" / flag-extension
                    ; Other flags; multiple possible per LIST response

mbx-list-sflag  = "\Noselect" / "\Marked" / "\Unmarked"
                    ; Selectability flags; only one per LIST response

media-basic     = ((DQUOTE ("APPLICATION" / "AUDIO" / "IMAGE" /
                  "MESSAGE" / "VIDEO") DQUOTE) / string) SP
                  media-subtype
                    ; Defined in [MIME-IMT]

media-message   = DQUOTE "MESSAGE" DQUOTE SP DQUOTE "RFC822" DQUOTE
                    ; Defined in [MIME-IMT]

media-subtype   = string
                    ; Defined in [MIME-IMT]

                    ; The Universal Time zone is "+0000".

media-text      = DQUOTE "TEXT" DQUOTE SP media-subtype
                    ; Defined in [MIME-IMT]

message-data    = nz-number SP ("EXPUNGE" / ("FETCH" SP msg-att))

msg-att         = "(" (msg-att-dynamic / msg-att-static)
                   *(SP (msg-att-dynamic / msg-att-static)) ")"

msg-att-dynamic = "FLAGS" SP "(" [flag-fetch *(SP flag-fetch)] ")"
                    ; MAY change for a message

msg-att-static  = "ENVELOPE" SP envelope / "INTERNALDATE" SP date-time /
                  "RFC822" [".HEADER" / ".TEXT"] SP nstring /
                  "RFC822.SIZE" SP number /
                  "BODY" ["STRUCTURE"] SP body /
                  "BODY" section ["<" number ">"] SP nstring /
                  "UID" SP uniqueid
                    ; MUST NOT change for a message

nil             = "NIL"

nstring         = string / nil

number          = 1*DIGIT
                    ; Unsigned 32-bit integer
                    ; (0 <= n < 4,294,967,296)

nz-number       = digit-nz *DIGIT
                    ; Non-zero unsigned 32-bit integer
                    ; (0 < n < 4,294,967,296)

password        = astring

quoted          = DQUOTE *QUOTED-CHAR DQUOTE

QUOTED-CHAR     = <any TEXT-CHAR except quoted-specials> /
                  "\" quoted-specials

quoted-specials = DQUOTE / "\"

rename          = "RENAME" SP mailbox SP mailbox
                    ; Use of INBOX as a destination gives a NO error

response        = *(continue-req / response-data) response-done

response-data   = "*" SP (resp-cond-state / resp-cond-bye /
                  mailbox-data / message-data / capability-data) CRLF

response-done   = response-tagged / response-fatal

response-fatal  = "*" SP resp-cond-bye CRLF
                    ; Server closes connection immediately

response-tagged = tag SP resp-cond-state CRLF

resp-cond-auth  = ("OK" / "PREAUTH") SP resp-text
                    ; Authentication condition

resp-cond-bye   = "BYE" SP resp-text

resp-cond-state = ("OK" / "NO" / "BAD") SP resp-text
                    ; Status condition

resp-specials   = "]"

resp-text       = ["[" resp-text-code "]" SP] text

resp-text-code  = "ALERT" /
                  "BADCHARSET" [SP "(" astring *(SP astring) ")" ] /
                  capability-data / "PARSE" /
                  "PERMANENTFLAGS" SP "("
                  [flag-perm *(SP flag-perm)] ")" /
                  "READ-ONLY" / "READ-WRITE" / "TRYCREATE" /
                  "UIDNEXT" SP nz-number / "UIDVALIDITY" SP nz-number /
                  "UNSEEN" SP nz-number /
                  atom [SP 1*<any TEXT-CHAR except "]">]

search          = "SEARCH" [SP "CHARSET" SP astring] 1*(SP search-key)
                    ; CHARSET argument to MUST be registered with IANA

search-key      = "ALL" / "ANSWERED" / "BCC" SP astring /
                  "BEFORE" SP date / "BODY" SP astring /
                  "CC" SP astring / "DELETED" / "FLAGGED" /
                  "FROM" SP astring / "KEYWORD" SP flag-keyword /
                  "NEW" / "OLD" / "ON" SP date / "RECENT" / "SEEN" /
                  "SINCE" SP date / "SUBJECT" SP astring /
                  "TEXT" SP astring / "TO" SP astring /
                  "UNANSWERED" / "UNDELETED" / "UNFLAGGED" /
                  "UNKEYWORD" SP flag-keyword / "UNSEEN" /
                    ; Above this line were in [IMAP2]
                  "DRAFT" / "HEADER" SP header-fld-name SP astring /
                  "LARGER" SP number / "NOT" SP search-key /
                  "OR" SP search-key SP search-key /
                  "SENTBEFORE" SP date / "SENTON" SP date /
                  "SENTSINCE" SP date / "SMALLER" SP number /
                  "UID" SP sequence-set / "UNDRAFT" / sequence-set /
                  "(" search-key *(SP search-key) ")"

section         = "[" [section-spec] "]"

section-msgtext = "HEADER" / "HEADER.FIELDS" [".NOT"] SP header-list /
                  "TEXT"
                    ; top-level or MESSAGE/RFC822 part

section-part    = nz-number *("." nz-number)
                    ; body part nesting

section-spec    = section-msgtext / (section-part ["." section-text])

section-text    = section-msgtext / "MIME"
                    ; text other than actual body part (headers, etc.)

select          = "SELECT" SP mailbox

seq-number      = nz-number / "*"
                    ; message sequence number (COPY, FETCH, STORE
                    ; commands) or unique identifier (UID COPY,
                    ; UID FETCH, UID STORE commands).
                    ; * represents the largest number in use.  In
                    ; the case of message sequence numbers, it is
                    ; the number of messages in a non-empty mailbox.
                    ; In the case of unique identifiers, it is the
                    ; unique identifier of the last message in the
                    ; mailbox or, if the mailbox is empty, the
                    ; mailbox's current UIDNEXT value.
                    ; The server should respond with a tagged BAD
                    ; response to a command that uses a message
                    ; sequence number greater than the number of
                    ; messages in the selected mailbox.  This
                    ; includes "*" if the selected mailbox is empty.

seq-range       = seq-number ":" seq-number
                    ; two seq-number values and all values between
                    ; these two regardless of order.
                    ; Example: 2:4 and 4:2 are equivalent and indicate
                    ; values 2, 3, and 4.
                    ; Example: a unique identifier sequence range of
                    ; 3291:* includes the UID of the last message in
                    ; the mailbox, even if that value is less than 3291.

sequence-set    = (seq-number / seq-range) *("," sequence-set)
                    ; set of seq-number values, regardless of order.
                    ; Servers MAY coalesce overlaps and/or execute the
                    ; sequence in any order.
                    ; Example: a message sequence number set of
                    ; 2,4:7,9,12:* for a mailbox with 15 messages is
                    ; equivalent to 2,4,5,6,7,9,12,13,14,15
                    ; Example: a message sequence number set of *:4,5:7
                    ; for a mailbox with 10 messages is equivalent to
                    ; 10,9,8,7,6,5,4,5,6,7 and MAY be reordered and
                    ; overlap coalesced to be 4,5,6,7,8,9,10.

status          = "STATUS" SP mailbox SP
                  "(" status-att *(SP status-att) ")"

status-att      = "MESSAGES" / "RECENT" / "UIDNEXT" / "UIDVALIDITY" /
                  "UNSEEN"

status-att-list =  status-att SP number *(SP status-att SP number)

store           = "STORE" SP sequence-set SP store-att-flags

store-att-flags = (["+" / "-"] "FLAGS" [".SILENT"]) SP
                  (flag-list / (flag *(SP flag)))

string          = quoted / literal

subscribe       = "SUBSCRIBE" SP mailbox

tag             = 1*<any ASTRING-CHAR except "+">

text            = 1*TEXT-CHAR

TEXT-CHAR       = <any CHAR except CR and LF>

time            = 2DIGIT ":" 2DIGIT ":" 2DIGIT
                    ; Hours minutes seconds

uid             = "UID" SP (copy / fetch / search / store)
                    ; Unique identifiers used instead of message
                    ; sequence numbers

uniqueid        = nz-number
                    ; Strictly ascending

unsubscribe     = "UNSUBSCRIBE" SP mailbox

userid          = astring

x-command       = "X" atom <experimental command arguments>

zone            = ("+" / "-") 4DIGIT
                    ; Signed four-digit value of hhmm representing
                    ; hours and minutes east of Greenwich (that is,
                    ; the amount that the given time differs from
                    ; Universal Time).  Subtracting the timezone
                    ; from the given time will give the UT form.
```

RFC 2812 (IRC)

The Augmented BNF representation for this is:

```
message    =  [ ":" prefix SPACE ] command [ params ] crlf
prefix     =  servername / ( nickname [ [ "!" user ] "@" host ] )
command    =  1*letter / 3digit
params     =  *14( SPACE middle ) [ SPACE ":" trailing ]
           =/ 14( SPACE middle ) [ SPACE [ ":" ] trailing ]

nospcrlfcl =  %x01-09 / %x0B-0C / %x0E-1F / %x21-39 / %x3B-FF
                ; any octet except NUL, CR, LF, " " and ":"
middle     =  nospcrlfcl *( ":" / nospcrlfcl )
trailing   =  *( ":" / " " / nospcrlfcl )

SPACE      =  %x20        ; space character
crlf       =  %x0D %x0A   ; "carriage return" "linefeed"

target     =  nickname / server
msgtarget  =  msgto *( "," msgto )
msgto      =  channel / ( user [ "%" host ] "@" servername )
msgto      =/ ( user "%" host ) / targetmask
msgto      =/ nickname / ( nickname "!" user "@" host )
channel    =  ( "#" / "+" / ( "!" channelid ) / "&" ) chanstring
              [ ":" chanstring ]
servername =  hostname
host       =  hostname / hostaddr
hostname   =  shortname *( "." shortname )
shortname  =  ( letter / digit ) *( letter / digit / "-" )
              *( letter / digit )
                ; as specified in RFC 1123 [HNAME]
hostaddr   =  ip4addr / ip6addr
ip4addr    =  1*3digit "." 1*3digit "." 1*3digit "." 1*3digit
ip6addr    =  1*hexdigit 7( ":" 1*hexdigit )
ip6addr    =/ "0:0:0:0:0:" ( "0" / "FFFF" ) ":" ip4addr
nickname   =  ( letter / special ) *8( letter / digit / special / "-" )
targetmask =  ( "$" / "#" ) mask
                ; see details on allowed masks in section 3.3.1
chanstring =  %x01-07 / %x08-09 / %x0B-0C / %x0E-1F / %x21-2B
chanstring =/ %x2D-39 / %x3B-FF
                ; any octet except NUL, BELL, CR, LF, " ", "," and ":"
channelid  = 5( %x41-5A / digit )   ; 5( A-Z / 0-9 )


user       =  1*( %x01-09 / %x0B-0C / %x0E-1F / %x21-3F / %x41-FF )
                ; any octet except NUL, CR, LF, " " and "@"
key        =  1*23( %x01-05 / %x07-08 / %x0C / %x0E-1F / %x21-7F )
                ; any 7-bit US_ASCII character,
                ; except NUL, CR, LF, FF, h/v TABs, and " "
letter     =  %x41-5A / %x61-7A       ; A-Z / a-z
digit      =  %x30-39                 ; 0-9
hexdigit   =  digit / "A" / "B" / "C" / "D" / "E" / "F"
special    =  %x5B-60 / %x7B-7D
                 ; "[", "]", "\", "`", "_", "^", "{", "|", "}"
```

Efficient parsing techniques exist.

LALR(k)

Earley

LL(k)

Operator precedence

GLR

SLR

CYK

LR(k)

Combinators

PEG

packrat

Parsing tools abound.

Yacc

Parsec

ANTLR

NLTK

Happy

Flex

Bison

CUPS

Ragel

PLY

State of the art?

`*buf++`

Apache

2,179 lines of C

lighttpd

1,211 lines of C

freenode IRCD

> 2000 lines of C

Courier IMAP

2,633 lines of C

Result?

CVE-ID

CVE-2004-0786 Learn more at National Vulnerability Databas

• Severity Rating • Fix Information • Vulnerable Software Ve

Desc

The I
(child

Refe

Note:

• G
• U
• M
• U

Vulnerability Details : CV

mod_access.c in lighttpd 1.4.

Publish Date : 2007-07-23 Last U

Collapse All E

Click here if yo

− CVSS Sco

Cvs

Cor

IRC buffer o
(IRC_Daem

About this sig

Proventia S

IRCnet IRCD Buffer Overflow

Secunia ID	SA9999
CVE-ID	CVE-2003-0864
Release Date	16 Oct 2003
Last Change	20 Oct 2003
Criticality	Not Critical
Solution Status	Vendor Patch
Software	IRCnet IRCD 2.x
Where	Local system

Details

Vulnerable systems:
* mIRC version 6.1 and prior

Immune systems:
* mIRC version 6.11

When mIRC is installed, it registers its own handler for URL of the type "irc". Calling "irc://irc.hackme.com" from our web browser causes mirc irc.hackme.com server. By inputting an overly long string to the "irc" protoc instruction pointer, thus controls the program's execution.

Example:
irc://[buffer]...... where's buffer >998 bytes

An attacker would be able to gain access to the target system if he was able Hence, he can have his code executed under the current user's privilege.

be modified,

e.)

Apache 1.3.37 h

From: "Matias Soler" <g
Date: Tue, 2 Jan 2007 17

Buffer over

```
Synopsis: Apache 1.3
Version: 1.3.37 (lat

Product
=======
Apache htpasswd util:

Issue
=====
A buffer overflow vilnerability has been found, it is dangerous only on
environment where the binary is suid root.

Details
=======
Incorrect validation on the size of user input allows to copy a string, via
strcpy, to a fixed size buffer.
File: htpasswd.c, Line 421.
```

Pr

Comp

V

Pla

Impo

Target Mile

Assigr

Keywords: FixedInTrunk, PatchAvailable

Depends on:

Blocks:

Show dependency tree

Mar 11 2004 12:00AM

Jul 12 2009 03:06AM

These issues were disclosed by the vendor.

Inter7 Courier-IMAP 2.2.1
Inter7 Courier-IMAP 2.2 .0
Inter7 Courier-IMAP 2.1.2
Inter7 Courier-IMAP 2.1.1

“High-performance network daemon”

“High-throughput exploit server”

Yet...

All of them also use lex and yacc.

(For config files.)

The tools exist.

They know how to use them.

So, why don’t they?

Compilers

Interpreters

Syntax highlighting

Natural language

Network protocols

Command lines

Configuration files

Serialization

Query languages

API design

Compilers

Interpreters

Syntax highlighting

Natural language

Network protocols

Command lines

Configuration files

Serialization

Query languages

API design

```
$ cat /etc/*
```

networks

protocols

man.conf

crontab

asl.conf

hosts

bind zones

fstab

sudoers

ftpd.conf

syslog.conf

passwd

services

ssh_config

notify.conf

gettytab

A tribute to false dichotomy.

Human

versus

Machine


There is no tradeoff.

XML: a compromise solution to a problem that does not exist.

Neither human nor machine-readable.

The tools exist.

So, why don’t we use them?

Compilers

Interpreters

Syntax highlighting

Natural language

Network protocols

Command lines

Configuration files

Serialization

Query languages

API design

Compilers

Interpreters

Syntax highlighting

Natural language

Network protocols

Command lines

Configuration files

Serialization

Query languages

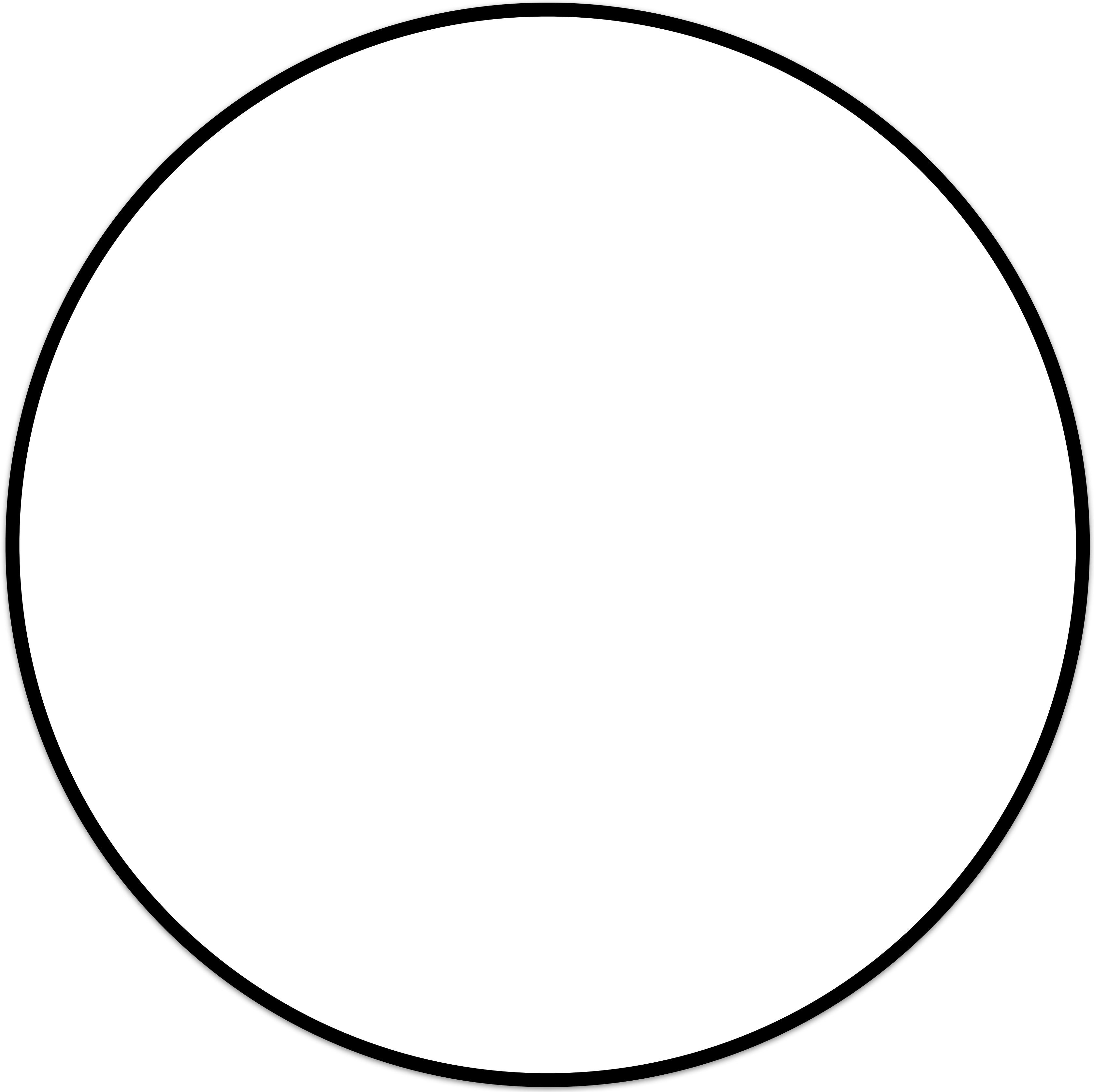
API design

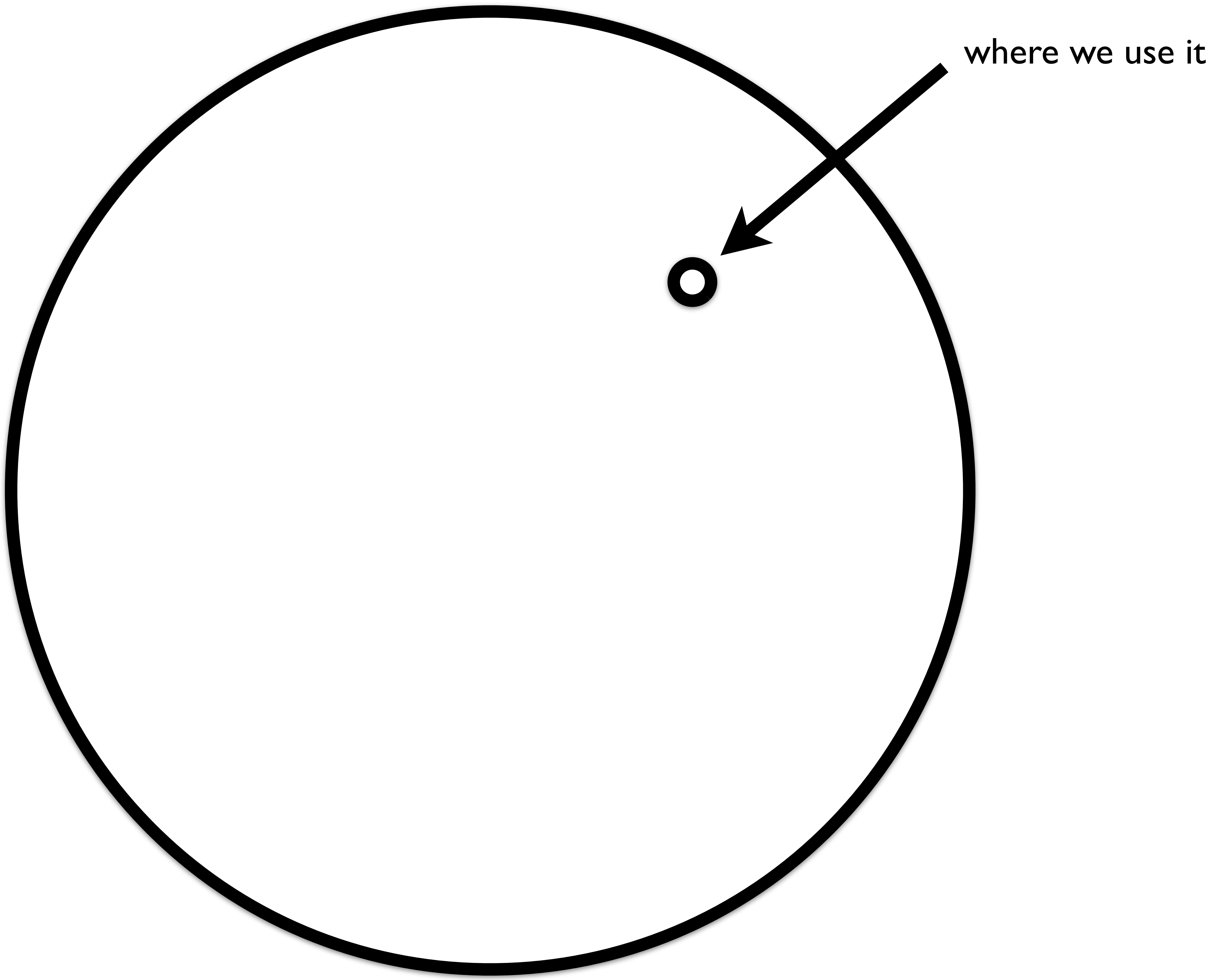
```c
#include <stdio.h>
#include <stdlib.h>


typedef unsigned int count ;

count parens1 ;
unsigned int parens2 ;
```

```c
#include <stdio.h>
#include <stdlib.h>


typedef unsigned int count ;

count parens1 ;
unsigned int parens2 ;
```

emacs, vi: over 30 years

WHY!?




where we use it

A list of grievances

Opportunity cost

Transaction cost

Barriers to entry

Opportunity cost

What do you sacrifice with a parser generator?

Performance

Thread-safety

Non-blocking I/O

~~Performance~~

Thread-safety

Non-blocking I/O

~~Performance~~

~~Thread safety~~

Non-blocking I/O

What if a server blocks on I/O?

DoS

Transaction cost

Time to set up lex & yacc

versus

Time to hack it with `strtok`

lex/yacc	`strtok()`

lex/yacc	`strtok()`
Create `.y` yacc file	

# lex/yacc	# `strtok()`
Create `.y` yacc file Define `%union lval`	

lex/yacc

Create `.y` yacc file

Define `%union lval`

Define tokens/types

`strtok()`

lex/yacc

Create `.y` yacc file

Define `%union lval`

Define tokens/types

Define yacc rules

`strtok()`

lex/yacc

Create `.y` yacc file

Define `%union lval`

Define tokens/types

Define yacc rules

Compile yacc spec

`strtok()`

lex/yacc

Create `.y` yacc file

Define `%union lval`

Define tokens/types

Define yacc rules

Compile yacc spec

Create .l lex file

`strtok()`

lex/yacc

Create `.y` yacc file

Define `%union lval`

Define tokens/types

Define yacc rules

Compile yacc spec

Create .l lex file

`#include y.tab.h`

`strtok()`

lex/yacc	`strtok()`
Create `.y` yacc file	
Define `%union lval`	
Define tokens/types	
Define yacc rules	
Compile yacc spec	
Create .l lex file	
`#include y.tab.h`	
Define states	

lex/yacc	`strtok()`
Create `.y` yacc file	
Define `%union lval`	
Define tokens/types	
Define yacc rules	
Compile yacc spec	
Create .l lex file	
`#include y.tab.h`	
Define states	
Define lex rules	

lex/yacc

Create `.y` yacc file

Define `%union lval`

Define tokens/types

Define yacc rules

Compile yacc spec

Create .l lex file

`#include y.tab.h`

Define states

Define lex rules

Deal with `yywrap()`

`strtok()`

lex/yacc	`strtok()`
Create `.y` yacc file	
Define `%union lval`	
Define tokens/types	
Define yacc rules	
Compile yacc spec	
Create .l lex file	
`#include y.tab.h`	
Define states	
Define lex rules	
Deal with `yywrap()`	
Call `yyparse()`	

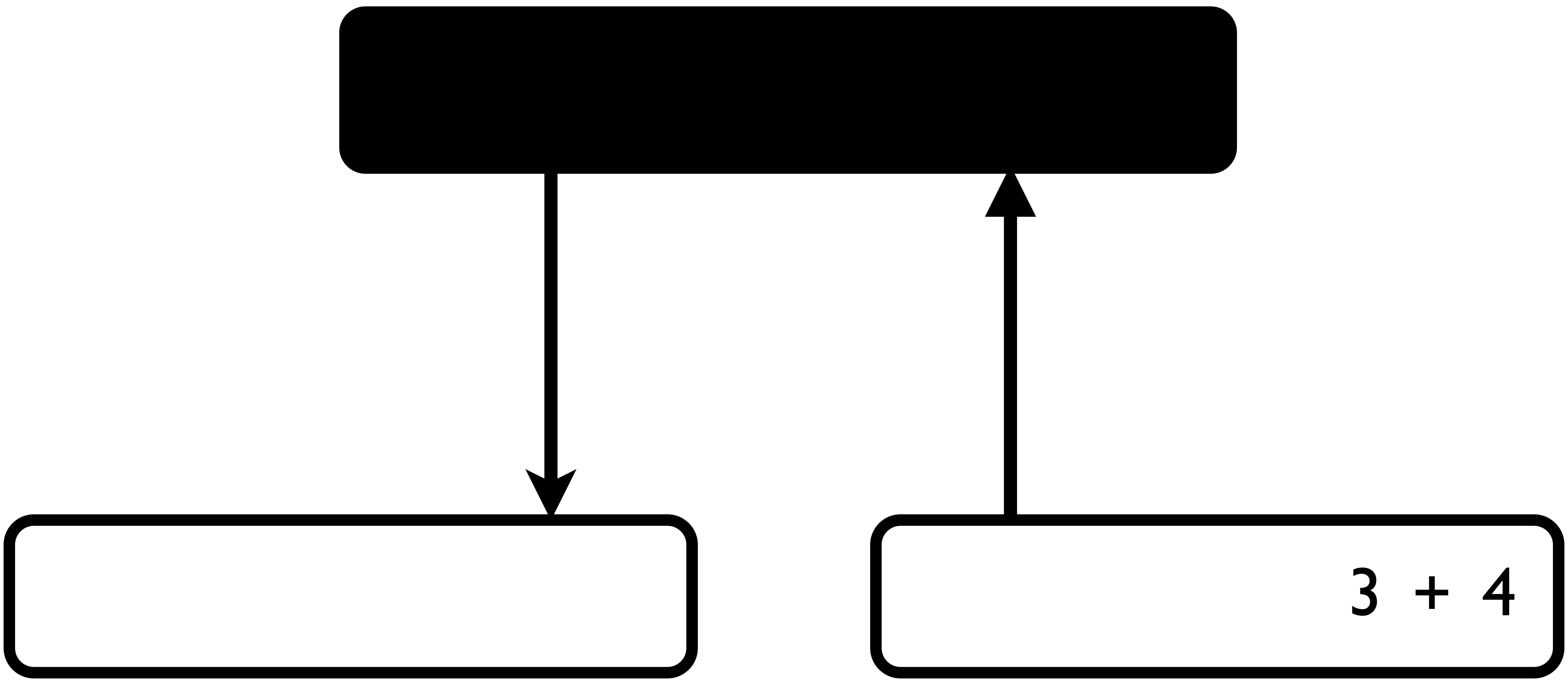
lex/yacc	`strtok()`
Create `.y` yacc file	
Define `%union lval`	
Define tokens/types	
Define yacc rules	
Compile yacc spec	
Create .l lex file	Call `strtok()`
`#include y.tab.h`	
Define states	
Define lex rules	
Deal with `yywrap()`	
Call `yyparse()`	

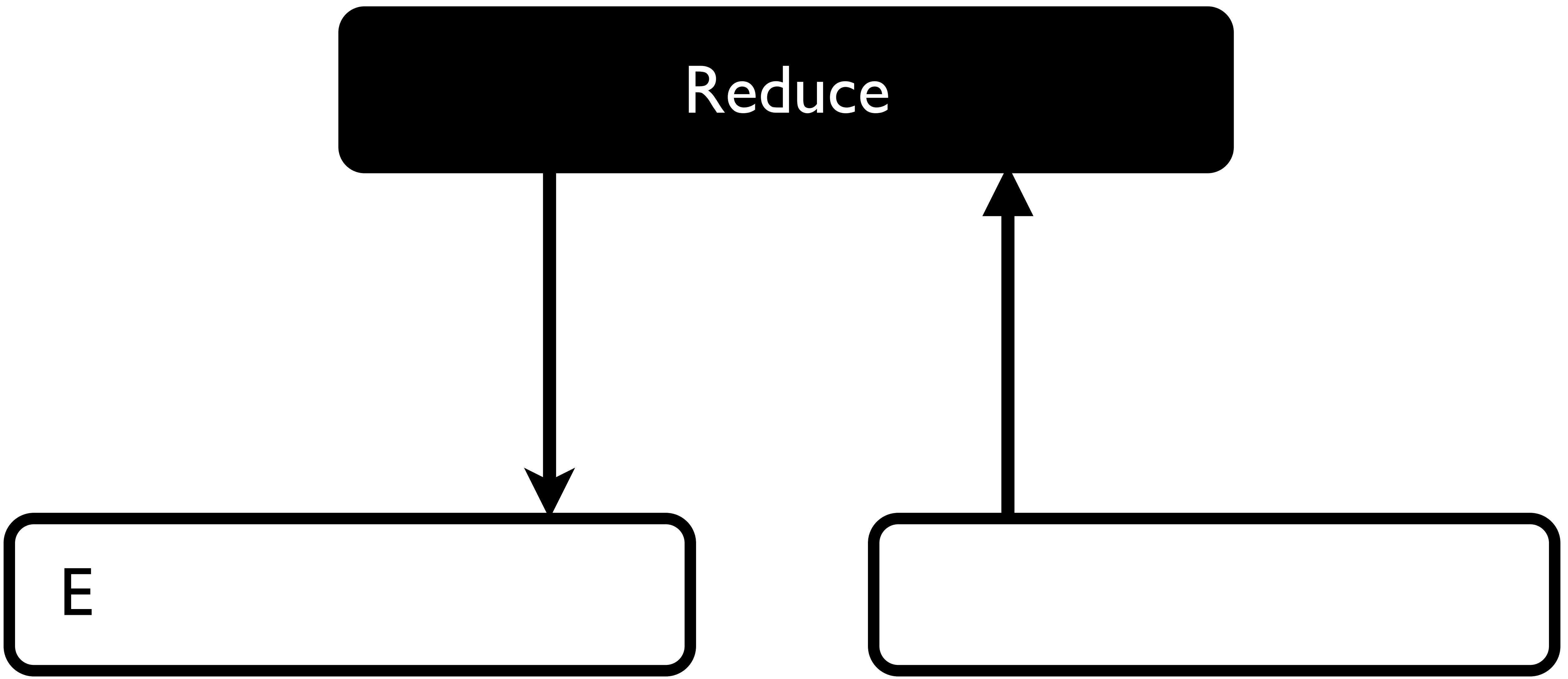
Barriers to entry

How does yacc work?

```
start : exp { printf("ans: %i\n", $1 ); }
      ;

exp : exp '+' term { $$ = $1 + $3 ; }
    | term         { $$ = $1 ; }
    ;

term : term '*' factor { $$ = $1 * $3 ; }
     | factor          { $$ = $1 ; }
     ;

factor : INT { $$ = $1 ; }
       | SYM { $$ = lookup($1) ; }
       | '(' exp ')' { $$ = $2 ; }
       ;
```

[redacted]


3 + 4



Reduce
E

All good until...

```
$ bison -d exp.y
exp.y: conflicts: 57 shift/reduce
```

```
$ bison -d exp.y
exp.y: conflicts: 57 shift/reduce
```

What's in the black box?

```
$ wc -l exp.output
    5109 exp.output
```

```
state 301:

    2 exp: exp . '+' exp  [$end, '+', '*', ')']
    2    | exp '+' exp .  [$end, '+', '*', ')']
    3    | exp . '*' exp

    '+'  shift, and go to state 8
    '*'  shift, and go to state 9

    '+'        [reduce using rule 2 (exp)]
    '*'        [reduce using rule 2 (exp)]
    $default  reduce using rule 2 (exp)
```

```
%left '+'
%left '*'
```

```
$ bison -d exp.y
exp.y: conflicts: 53 shift/reduce
```

Two days later...

```
%right VAR
%left '@'
%right FOR
%left WHILE
%nonassoc '-'
%left FUNCTION
%nonassoc RETURN
%right '?'
%left '+'
%left ','
%left '*'
%left '['
%right ']'
%left '.'
%left '{'
%right '}'
%nonassoc ';'
```

```
$ bison -d exp.y
exp.y: conflicts: 1 shift/reduce
```

```
$ bison -d exp.y
exp.y: conflicts: 1 shift/reduce
```

“Probably just dangling `else`.”

Barrier to entry:
One must *learn* each tool.

Barrier to entry:
Tool must exist for language.

How do we change the economics of parsing?

What we need

- Allow non-blocking I/O
- Act as library within the language
- Handle all CFGs: parse forest
- Draw on regular expressions
- Cacheable partial parses
- Be easy to implement

(Jim, Mandelbaum, Walker, POPL 2010)

Would be nice

- Dynamic reconfigurability
- Beyond context-free languages
- Built into the language
- Support compositionality

(Jim, Mandelbaum, Walker, POPL 2010)

Parsing with derivatives

Derivatives!?

Brzozowski, 1964

Derivatives of Regular Expressions

JANUSZ A. BRZOZOWSKI

Princeton University, Princeton, New Jersey†

Abstract. Kleene's regular expressions, which can be used for describing sequential circuits, were defined using three operators (union, concatenation and iterate) on sets of sequences. Word descriptions of problems can be more easily put in the regular expression language if the language is enriched by the inclusion of other logical operations. However, in the problem of converting the regular expression description to a state diagram, the existing methods either cannot handle expressions with additional operators, or are made quite complicated by the presence of such operators. In this paper the notion of a derivative of a regular expression is introduced and the properties of derivatives are discussed. This leads, in a very natural way, to the construction of a state diagram from a regular expression containing any number of logical operators.

A language is a set of strings.

Regular languages

$$\epsilon \qquad \equiv \{""\}$$

$$c \qquad \equiv \{c\}$$

$$\emptyset$$

Regular languages

$$L_1 \cup L_2$$

$$L_1 \cdot L_2$$

$$L_1^\star$$

```
(foo|bar)*
```

$\{\texttt{foo,bar}\}^*$

$$D_c L = \{w : cw \in L\}$$

The derivative

1. **Filter**: Keep every string starting with c.
2. **Chop**: Remove c from the start of each.

$$D_{\mathtt{f}}\{\mathtt{foo}, \mathtt{frak}, \mathtt{bar}\} = \{\mathtt{oo}, \mathtt{rak}\}$$

$$D_{\texttt{f}}\{\texttt{foo}, \texttt{frak}, \texttt{bar}\} = \{\texttt{oo}, \texttt{rak}\}$$

$$D_{\texttt{b}}\{\texttt{foo}, \texttt{frak}, \texttt{bar}\} = \{\texttt{ar}\}$$

$$D_{\mathtt{f}}\{\mathtt{foo}, \mathtt{frak}, \mathtt{bar}\} = \{\mathtt{oo}, \mathtt{rak}\}$$

$$D_{\mathtt{b}}\{\mathtt{foo}, \mathtt{frak}, \mathtt{bar}\} = \{\mathtt{ar}\}$$

$$D_{\mathtt{x}}\{\mathtt{foo}, \mathtt{frak}, \mathtt{bar}\} = \emptyset$$

$$D_{\mathtt{f}}\{\mathtt{foo}, \mathtt{bar}\}^{\star} =$$

$$D_{\mathtt{f}}\{\mathtt{foo},\mathtt{bar}\}^{\star} = \{\mathtt{oo}\} \cdot \{\mathtt{foo},\mathtt{bar}\}^{\star}$$

$$D_{\mathtt{f}}\{\mathtt{foo}, \mathtt{bar}\}^{\star} \cdot \{\mathtt{frak}\} =$$

$$D_{\mathtt{f}}\{\mathtt{foo}, \mathtt{bar}\}^{\star} \cdot \{\mathtt{frak}\} = \{\mathtt{oo}\} \cdot \{\mathtt{foo}, \mathtt{bar}\}^{\star} \cdot \{\mathtt{frak}\}$$
$$\cup \{\mathtt{rak}\}$$

Observation

$$cw \in L \text{ iff } w \in D_c(L).$$

Matching algorithm

- Derive with respect to each character.
- Does the derived language contain ε?

$$\texttt{foo} \in \{\texttt{foo}, \texttt{bar}\}^\star?$$

$$D_{\mathtt{f}}\{\mathtt{foo}, \mathtt{bar}\}^\star = \{\mathtt{oo}\} \cdot \{\mathtt{foo}, \mathtt{bar}\}^\star$$

$$D_{\mathtt{f}}\{\mathtt{foo},\mathtt{bar}\}^\star = \{\mathtt{oo}\}\cdot\{\mathtt{foo},\mathtt{bar}\}^\star$$

$$D_{\mathtt{o}}\{\mathtt{oo}\}\cdot\{\mathtt{foo},\mathtt{bar}\}^\star = \{\mathtt{o}\}\cdot\{\mathtt{foo},\mathtt{bar}\}^\star$$

$$D_{\mathtt{f}}\{\mathtt{foo}, \mathtt{bar}\}^{\star} = \{\mathtt{oo}\} \cdot \{\mathtt{foo}, \mathtt{bar}\}^{\star}$$

$$D_{\mathtt{o}}\{\mathtt{oo}\} \cdot \{\mathtt{foo}, \mathtt{bar}\}^{\star} = \{\mathtt{o}\} \cdot \{\mathtt{foo}, \mathtt{bar}\}^{\star}$$

$$D_{\mathtt{o}}\{\mathtt{o}\} \cdot \{\mathtt{foo}, \mathtt{bar}\}^{\star} = \{\mathtt{foo}, \mathtt{bar}\}^{\star}$$

Nullability

$$\delta(L) = \epsilon \text{ if } \epsilon \in L$$
$$\delta(L) = \emptyset \text{ if } \epsilon \notin L$$

$$\delta(\epsilon) = \epsilon$$

$$\delta(c) = \emptyset$$

$$\delta(\emptyset) = \emptyset$$

$$
\begin{aligned}
\delta(L_1 \cup L_2) &= \delta(L_1) \cup \delta(L_2) \\
\delta(L_1 \cdot L_2) &= \delta(L_1) \cdot \delta(L_2) \\
\delta(L_1^\star) &= \epsilon
\end{aligned}
$$

```
(define (δ L)
  (match L
    [(empty)         #f]
    [(eps)           #t]
    [(char _)        #f]

    [(rep _)         #t]
    [(or  L1 L2)     (or  (δ L1) (δ L2))]
    [(seq L1 L2)     (and (δ L1) (δ L2))]))
```

Case: Empty set

$$D_c\emptyset =$$

Case: Empty set

$$D_c\emptyset = \emptyset$$

Case: Empty string

$$D_c(\epsilon) =$$

Case: Empty string

$$D_c(\epsilon) = \emptyset$$

Case: Character

$$D_c\{c\} =$$

$$D_c\{c'\} =$$

Case: Character

$$D_c\{c\} = \epsilon$$

$$D_c\{c'\} = \emptyset \text{ if } c \neq c'$$

Case: Union

$D_c(L_1 \cup L_2)$

Case: Union

$$
\begin{aligned}
D_c(L_1 \cup L_2) &= \{w : cw \in L_1 \cup L_2\} \\
&= \{w : cw \in L_1 \text{ or } cw \in L_2\} \\
&= \{w : w \in D_c L_1 \text{ or } w \in D_c L_2\} \\
&= \{w : w \in D_c L_1\} \cup \{ w : w \in D_c L_2\} \\
&= D_c L_1 \cup D_c L_2.
\end{aligned}
$$

Case: Intersection

$$D_c(L_1 \cap L_2) =$$

Case: Intersection

$$D_c(L_1 \cap L_2) = D_c L_1 \cap D_c L_2.$$

Case: Difference

$$D_c(L_1 - L_2) =$$

Case: Difference

$$D_c(L_1 - L_2) = D_c L_1 - D_c L_2.$$

Case: Catenation

$$D_c(L_1 \cdot L_2) =$$

Case: Catenation

$$D_c(L_1 \cdot L_2) = (D_c L_1 \cdot L_2) \cup (\delta(L_1) \cdot D_c L_2)$$

Case: Complement

$D_c\overline{L}$

Case: Complement

$$\begin{aligned}
D_c\overline{L} &= \{w : cw \in \overline{L}\} \\
&= \{w : cw \notin L\} \\
&= \{w : \text{not } cw \in L\} \\
&= \overline{\{w : cw \in L\}} \\
&= \overline{\{w : w \in D_cL\}} \\
&= \overline{D_cL}.
\end{aligned}$$

Case: Option

$$D_c(L^?) =$$

Case: Option

$$D_c(L^?) = D_c L.$$

Case: Kleene star

$$D_c(L^\star) =$$

Case: Kleene star

$$D_c(L^\star) = (D_c L) \cdot L^\star$$

Case: Kleene plus

$$D_c(L^+) =$$

Case: Kleene plus

$$D_c(L^+) = (D_c L) \cdot L^\star$$

```
(define (D c L)
  (match L
    [(empty)      (empty)]
    [(eps)        (empty)]
    [(char c)     (eps)]
    [(char _)     (empty)]

    [(or L1 L2)                       (alt (D c L1)
                                           (D c L2))]
    [(seq (and (? nullable?) L1) L2)  (alt (D c L2)
                                           (cat (D c L1) L2))]
    [(seq L1 L2)                      (cat (D c L1) L2)]
    [(rep L1)                         (cat (D c L1) L)]))
```

How about context-free grammars?

What's a context-free grammar?

Recursive regular expressions.

$$P = (\cdot P \cdot) \cdot P$$

$$\cup \ \epsilon$$

The derivative?

Works the same, but...

Problem

$$L = L \cdot \mathtt{x}$$

$$\cup \quad \epsilon$$

Problem

$$D_{\mathtt{x}}L = D_{\mathtt{x}}L \cdot \mathtt{x}$$

$$\cup \; \epsilon$$

```
(define (D c L)
  (match L
    [(empty)      (empty)]
    [(eps)        (empty)]
    [(char c)     (eps)]
    [(char _)     (empty)]

    [(or L1 L2)                          (alt (D c L1)
                                              (D c L2))]
    [(seq (and (nullable?) L1) L2)       (alt (D c L2)
                                              (cat (D c L1) L2))]
    [(seq L1 L2)                         (cat (D c L1) L2)]
    [(rep L1)                            (cat (D c L1) L)]))
```

```
(define (D c L)
  (match L
    [(empty)        (empty)]
    [(eps)          (empty)]
    [(char c)       (eps)]
    [(char _)       (empty)]

    [(or L1 L2)                          (alt (D c L1)
                                              (D c L2))]
    [(seq (and (nullable?) L1) L2)       (alt (D c L2)
                                              (cat (D c L1) L2))]
    [(seq L1 L2)                         (cat (D c L1) L2)]
    [(rep L1)                            (cat (D c L1) L)]))
```

Problem

$$\delta(L) = \delta(L) \cdot \delta(\mathtt{x})$$
$$\cup \;\; \delta(\epsilon)$$

Solution?

Laziness

Memoization

Fixed points

Laziness delays computation until moment of use.

Memoization caches return values.

Fixed points solve the chicken-and-egg problem.

Laziness

```
(define-syntax cat
  (syntax-rules ()
    [(_ L1 L2)  (concatenation         L1           L2 )]))
```

Laziness

```
(define-syntax cat
  (syntax-rules ()
    [(_ L1 L2)  (concatenation (delay L1) (delay L2))]))
```

Memoization

```
(define        (D c L)

  (match L
    [(empty)      (empty)]
    [(eps)        (empty)]
    [(char c)     (eps)]
    [(char _)     (empty)]

    [(or L1 L2)                           (alt (D c L1)
                                               (D c L2))]
    [(seq (and (nullable?) L1) L2)        (alt (D c L2)
                                               (cat (D c L1) L2))]
    [(seq L1 L2)                          (cat (D c L1) L2)]
    [(rep L1)                             (cat (D c L1) L)]))
```

Memoization

```
(define/memoize (D c L)
  #:order ([L #:eq] [c #:equal])
  (match L
    [(empty)      (empty)]
    [(eps)        (empty)]
    [(char c)     (eps)]
    [(char _)     (empty)]

    [(or L1 L2)                        (alt (D c L1)
                                            (D c L2))]
    [(seq (and (nullable?) L1) L2)     (alt (D c L2)
                                            (cat (D c L1) L2))]
    [(seq L1 L2)                       (cat (D c L1) L2)]
    [(rep L1)                          (cat (D c L1) L)]))
```

Fixed points

```
(define    (δ L)

  (match L
    [(empty)          #f]
    [(eps)            #t]
    [(char _)         #f]

    [(rep _)          #t]
    [(or  L1 L2)      (or  (δ L1) (δ L2))]
    [(seq L1 L2)      (and (δ L1) (δ L2))]))
```

Fixed points

```
(define/fix (δ L)
  #:bottom #f
  (match L
    [(empty)         #f]
    [(eps)           #t]
    [(char _)        #f]

    [(rep _)         #t]
    [(or  L1 L2)     (or  (δ L1) (δ L2))]
    [(seq L1 L2)     (and (δ L1) (δ L2))]))
```

Now it works.

(For recognizing.)

What about parsing?

What is a parser?

$$\mathbb{P}(A, T) = A^* \to \mathcal{P}(T \times A^*)$$

$$\mathbb{P}(A, T) = \overset{\substack{\text{Input string} \\ \downarrow}}{A^*} \to \mathcal{P}(T \times A^*)$$

Input string

$$\mathbb{P}(A, T) = A^* \to \mathcal{P}(T \times A^*)$$

Parse tree

Input string

Remaining input

$$\mathbb{P}(A, T) = A^* \to \mathcal{P}(T \times A^*)$$

Parse tree

$$\lfloor \mathbb{P} \rfloor (A, T) = A^* \to \mathcal{P}(T)$$

Input string

$$\lfloor \mathbb{P} \rfloor (A, T) = A^* \to \mathcal{P}(T)$$

Input string

$$\lfloor \mathbb{P} \rfloor (A, T) = A^* \to \mathcal{P}(T)$$

Parse tree

$$p \in \mathbb{P}(A, T)$$

$$\lfloor p \rfloor(w) = \{t : (t, \epsilon) \in p(w)\}$$

Context-free parsers

$$w \equiv \lambda w'.\begin{cases}\{(w, w'')\} & w' = ww'' \\ \emptyset & \text{otherwise.}\end{cases}$$

$$\epsilon \equiv \lambda w.\{(\epsilon, w)\}$$

$$\emptyset \equiv \lambda w.\{\}$$

$$p \in \mathbb{P}(A, \boldsymbol{X})$$

$$q \in \mathbb{P}(A, \boldsymbol{Y})$$

$$p \cdot q \in \mathbb{P}(A, \boldsymbol{X} \times \boldsymbol{Y})$$

$$p \cdot q = \lambda w.\{((x, y), w'') : (x, w') \in p(w), (y, w'') \in q(w')\}$$

$$p \cdot q = \lambda w.\{((x, y), w'') : (x, w') \in p(w), (y, w'') \in q(w')\}$$

Input

$$p \cdot q = \lambda w.\{((x,y),w'') : (x,w') \in p(w), (y,w'') \in q(w')\}$$

Input

First parse

Left overs

$$p \cdot q = \lambda w.\{((x, y), w'') : (x, w') \in p(w), (y, w'') \in q(w')\}$$

Input

First parse

Left overs

$$p \cdot q = \lambda w.\{((x, y), w'') : (x, w') \in p(w), (y, w'') \in q(w')\}$$

Input

First parse

Second parse

Left overs Remainder

$$p \cdot q = \lambda w.\{((x, y), w'') : (x, w') \in p(w), (y, w'') \in q(w')\}$$

Input First parse Second parse

$$
\begin{aligned}
p &\in \mathbb{P}(A, X) \\
q &\in \mathbb{P}(A, X) \\
p \cup q &\in \mathbb{P}(A, X) \\
p \cup q &= \lambda w. p(w) \cup q(w)
\end{aligned}
$$

$$
\begin{aligned}
f &\in X \to Y \\
p &\in \mathbb{P}(A, X) \\
p \to f &\in \mathbb{P}(A, Y) \\
p \to f &= \lambda w.\{((f(x), w') : (x, w') \in p(w)\}
\end{aligned}
$$

Defining the derivative

$$D_c : \mathbb{L} \to \mathbb{L}$$

$$D_c : \mathbb{P}(A, T) \to \mathbb{P}(A, T)$$

$$D_c(p) = \lambda w.p(cw) - (\lfloor p \rfloor(\epsilon) \times \{cw\})$$

$$D_c(p) = \lambda w.p(cw) - (\lfloor p \rfloor(\epsilon) \times \{cw\})$$

$$p(cw) = D_c(p)(w) \cup (\lfloor p \rfloor(\epsilon) \times \{cw\})$$

$$\lfloor p \rfloor (cw) = \lfloor D_c(p) \rfloor (w)$$

Calculating the derivative

$$D_c(c) = \epsilon \rightarrow \lambda\epsilon.c$$
$$D_c(c') = \emptyset \text{ if } c \neq c'$$

$$D_c(p \cup q) = D_c(p) \cup D_c(q)$$

$$D_c(p \to f) = D_c(p) \to f$$

$$D_c(p \cdot q) = \begin{cases} D_c(p) \cdot q & \epsilon \notin \mathcal{L}(p) \\ D_c(p) \cdot q \cup (\epsilon \rightarrow \lambda\epsilon.\lfloor p \rfloor(\epsilon)) \cdot D_c(q) & \text{otherwise.} \end{cases}$$

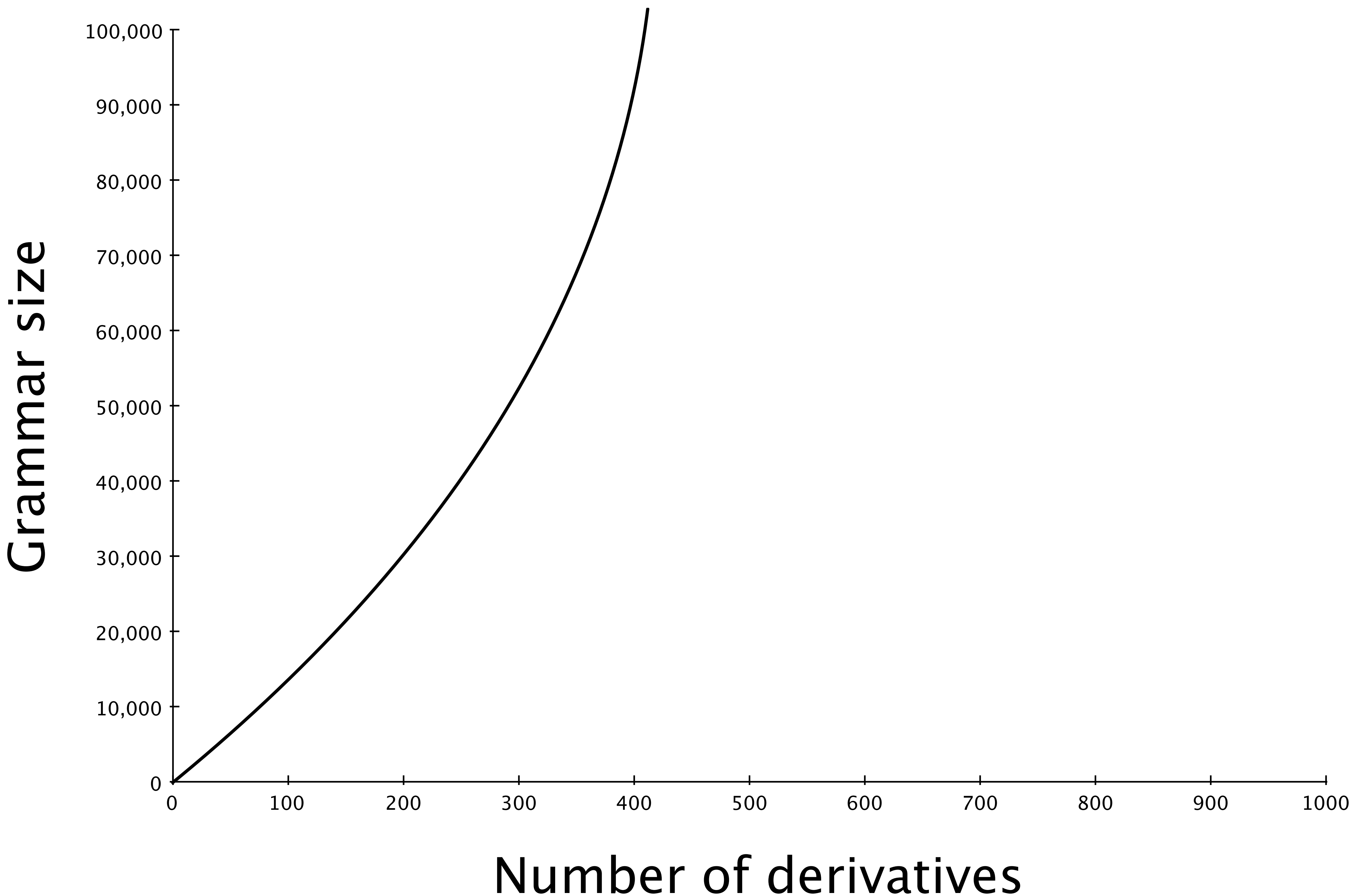
Code

```
; Derivative of a parser combinator:
(define/memoize (DP c L)
  #:order ([l #:eq] [c #:equal])
  (match L
    [(empty)              (empty)]
    [(eps)                (empty)]
    [(token pred class)  (if (pred c) (eps* (set c)) (empty))]

    [(orp L1 L2)                      (alt (DP c L1)
                                           (DP c L2))]
    [(seqp (and (nullp) l1) l2)       (cat (eps* (parse-null L1)) (DP c L2))]
    [(seqp (and (nullablep) l1) l2)   (alt (cat (eps* (parse-null l1)) (DP c L2))
                                           (cat (DP c L1) L2))]
    [(seqp l1 l2)                     (cat (DP c L1) L2)]
    [(repp l1)                        (cat (DP c L1) L)]
    [(redp l f)                       (red (DP c L) f)]))
```

```
(define/fix (parse-null l)
  #:bottom (set)
  (match l
    [(empty)        (set)]
    [(eps* S)       S]
    [(eps)          (set l)]
    [(token _ _)    (set)]
    [(repp (nullp)) (error "infinite parse-null")]
    [(repp _)       (set '())]
    [(orp  l1 l2)   (set-union (parse-null l1) (parse-null l2))]
    [(seqp l1 l2)   (for*/set ([t1 (parse-null l1)]
                               [t2 (parse-null l2)])
                              (cons t1 t2))]
    [(redp l1 f)    (for/set ([t (parse-null l1)])
                             (f t))]))
```

Performance?

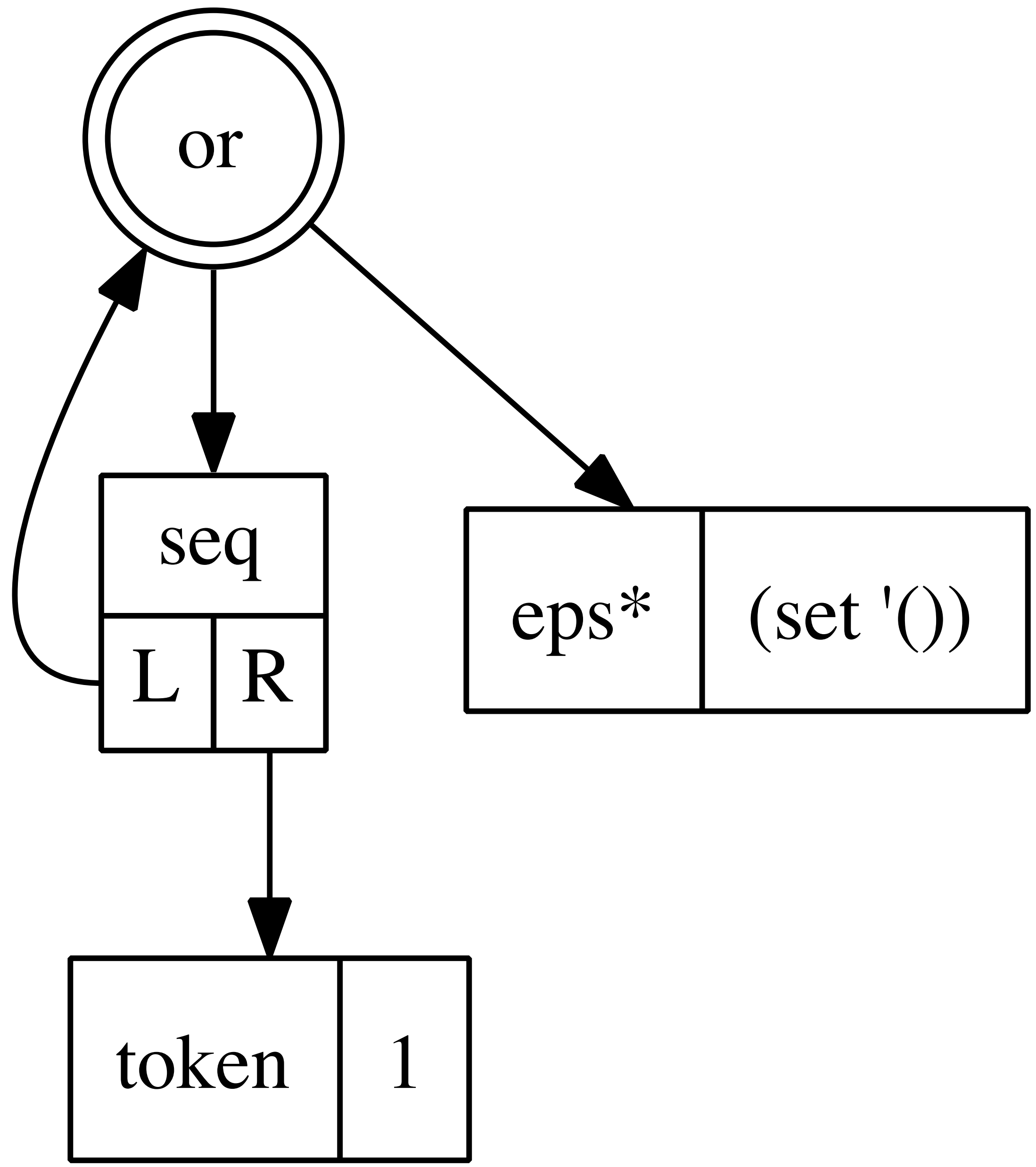
Atrocious.

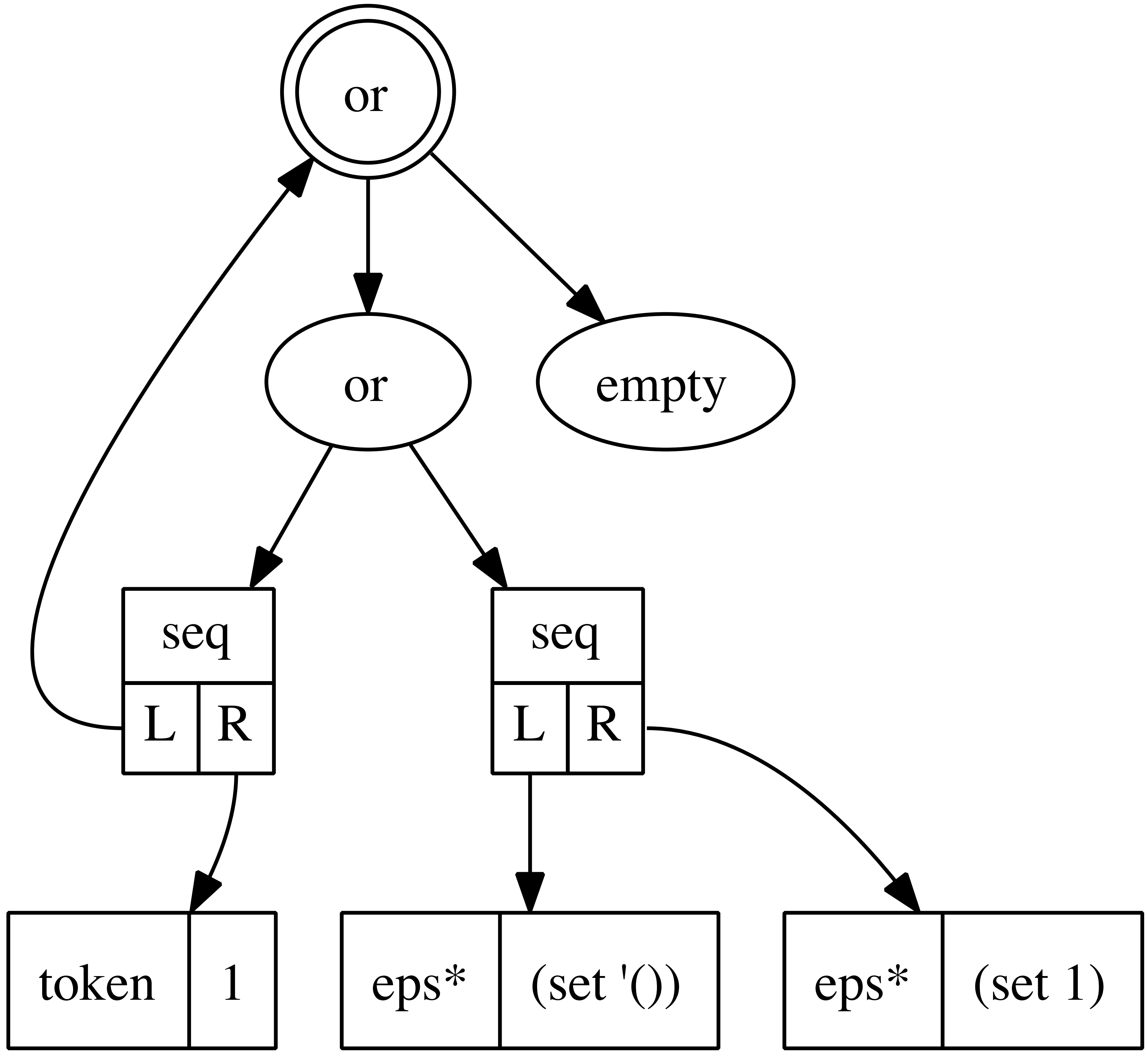

Grammar size
100,000
90,000
80,000
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
0
100
200
300
400
500
600
700
800
900
1000
Number of derivatives

Complexity?

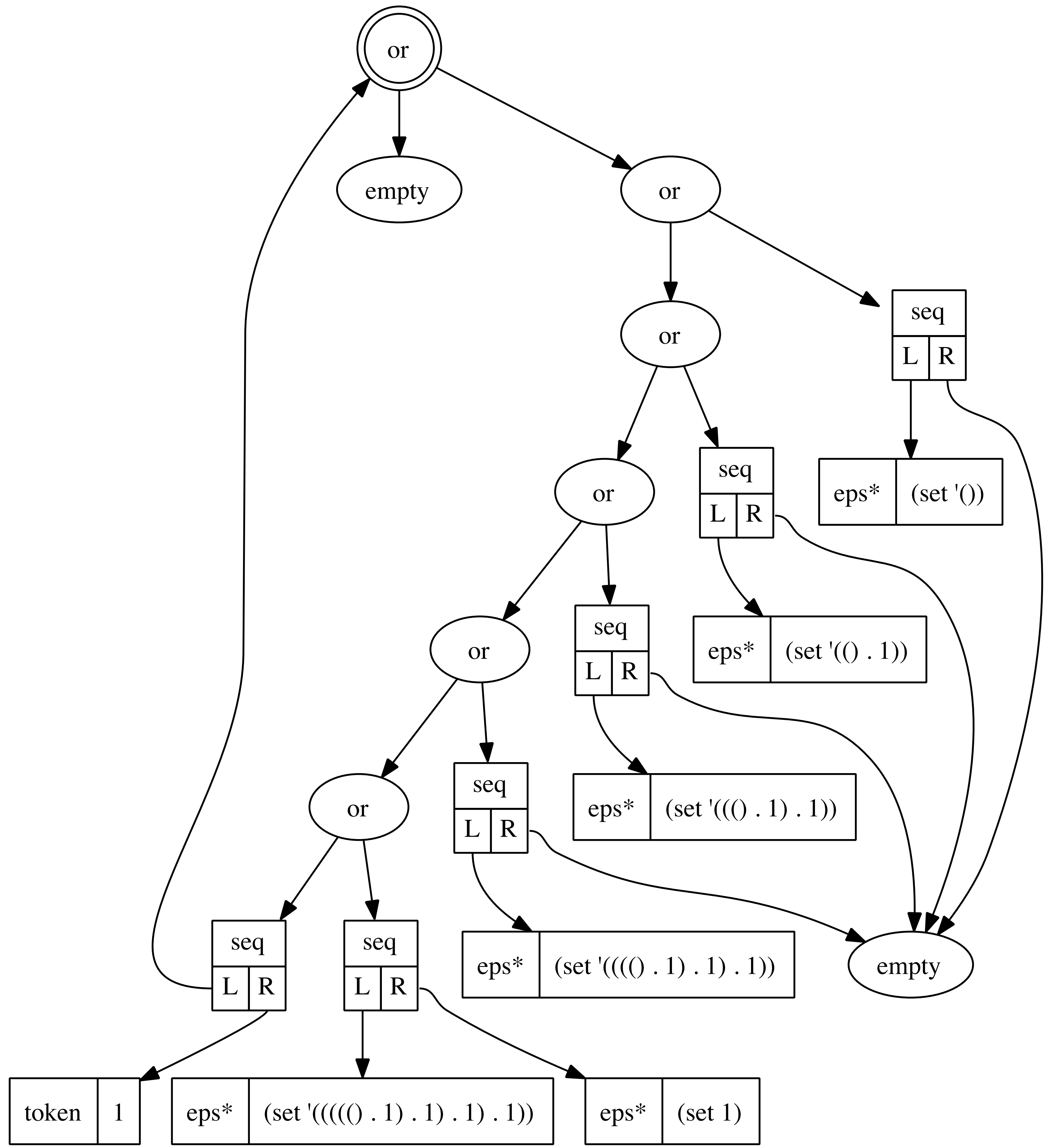
$$O(n2^n|G|)$$

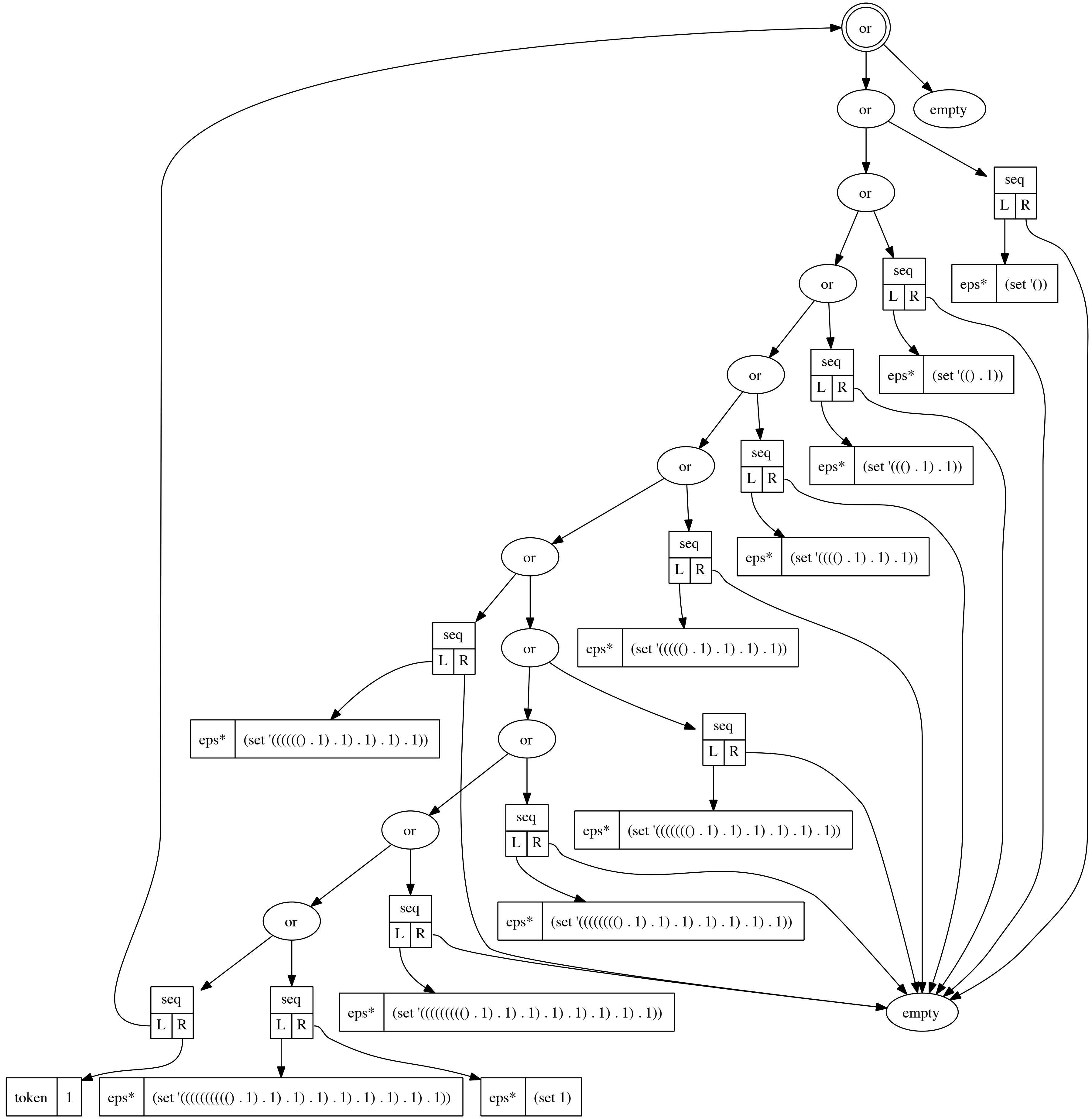
```
(define OneList (lang (or (seq OneList '1)
                          (eps* (set '()))))))
```


or
seq
L
R
eps*
(set '())
token
1



or
or
empty
seq
L
R
seq
L
R
token
1
eps*
(set '())
eps*
(set 1)


or
empty
or
or
seq
L
R
or
seq
L
R
eps*
(set '())
or
seq
L
R
eps*
(set '(() . 1))
or
seq
L
R
eps*
(set '((() . 1) . 1))
seq
L
R
seq
L
R
eps*
(set '(((() . 1) . 1) . 1))
empty
token
1
eps*
(set '((((() . 1) . 1) . 1) . 1))
eps*
(set 1)


or
or
empty
or
seq
L
R
or
seq
L
R
eps*
(set '())
or
seq
L
R
eps*
(set '(() . 1))
or
seq
L
R
eps*
(set '((() . 1) . 1))
or
seq
L
R
eps*
(set '(((() . 1) . 1) . 1))
seq
L
R
or
eps*
(set '((((() . 1) . 1) . 1) . 1))
eps*
(set '(((((() . 1) . 1) . 1) . 1) . 1))
or
seq
L
R
seq
L
R
eps*
(set '((((((() . 1) . 1) . 1) . 1) . 1) . 1))
or
eps*
(set '(((((((() . 1) . 1) . 1) . 1) . 1) . 1) . 1))
seq
L
R
or
eps*
(set '((((((((() . 1) . 1) . 1) . 1) . 1) . 1) . 1) . 1))
empty
seq
L
R
seq
L
R
token
1
eps*
(set '(((((((((() . 1) . 1) . 1) . 1) . 1) . 1) . 1) . 1) . 1))
eps*
(set 1)

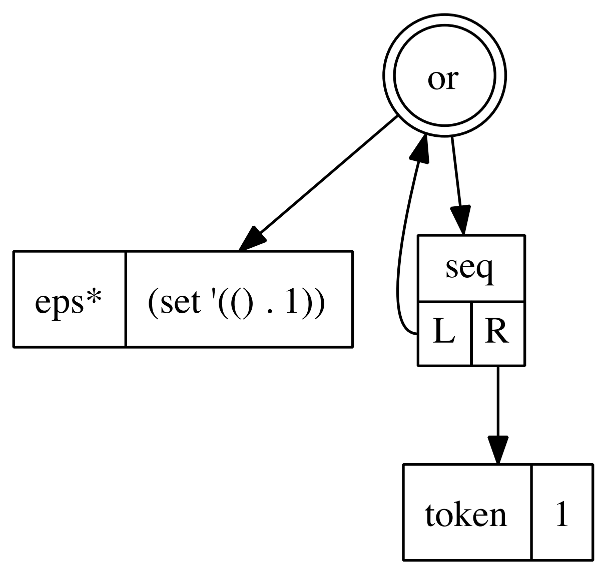
Needs compaction.

$$p \cdot \emptyset = \emptyset$$

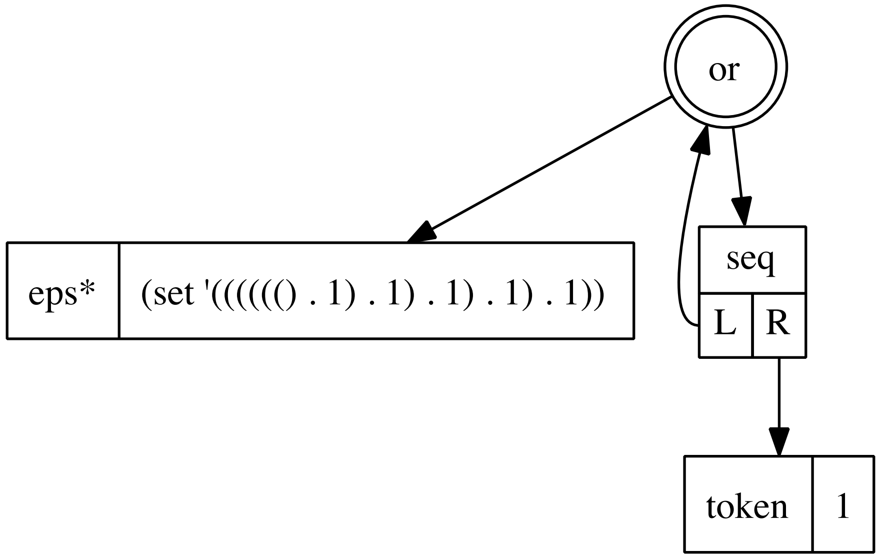
$$p \cdot q \to f = q \to \lambda x . f(\lfloor p \rfloor(\epsilon), x)$$
$$\text{if } \{\epsilon\} = \mathcal{L}(p)$$

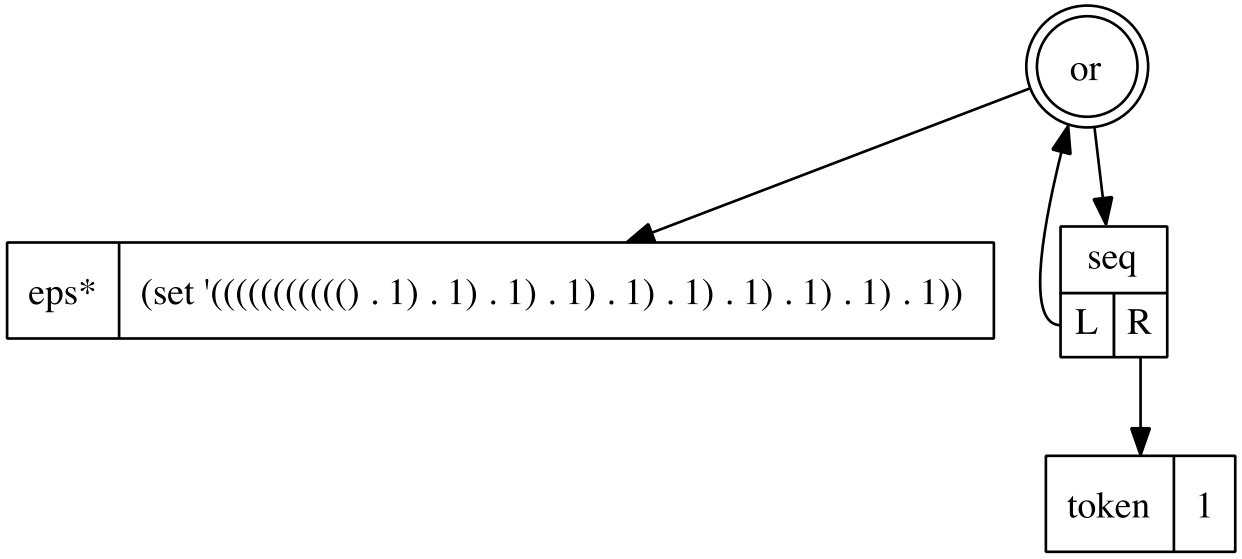
```
(define          (compact          )
  (match l
    [(empty)       l]
    [(eps)         l]
    [(emptyp)      (empty)]
    [(nullp)       (eps* (parse-null l))]
    [(token p c)   l]

    [(orp (emptyp) l2)  (compact l2)]
    [(orp l1 (emptyp))  (compact l1)]

    [(seqp (nullp t) l2)  (red (compact l2) (lambda (w2) (cons t w2)))]
    [(seqp l1 (nullp t))  (red (compact l1) (lambda (w1) (cons w1 t)))]

    [(orp l1 l2)    (alt (compact l1) (compact l2))]
    [(seqp l1 l2)   (cat (compact l1) (compact l2))]

    [(redp (and e (nullp)) f)
     (eps* (for/set ([t (parse-null e)]) (f t)))]

    [(redp (seqp (nullp t) l2) f)
     (red (compact l2) (lambda (w2) (f (cons t w2))))]

    [(redp (redp l f) g)
     (red (compact l) (lambda (w) (g (f w))))]

    [(redp l f)    (red (compact l) f)]))
```

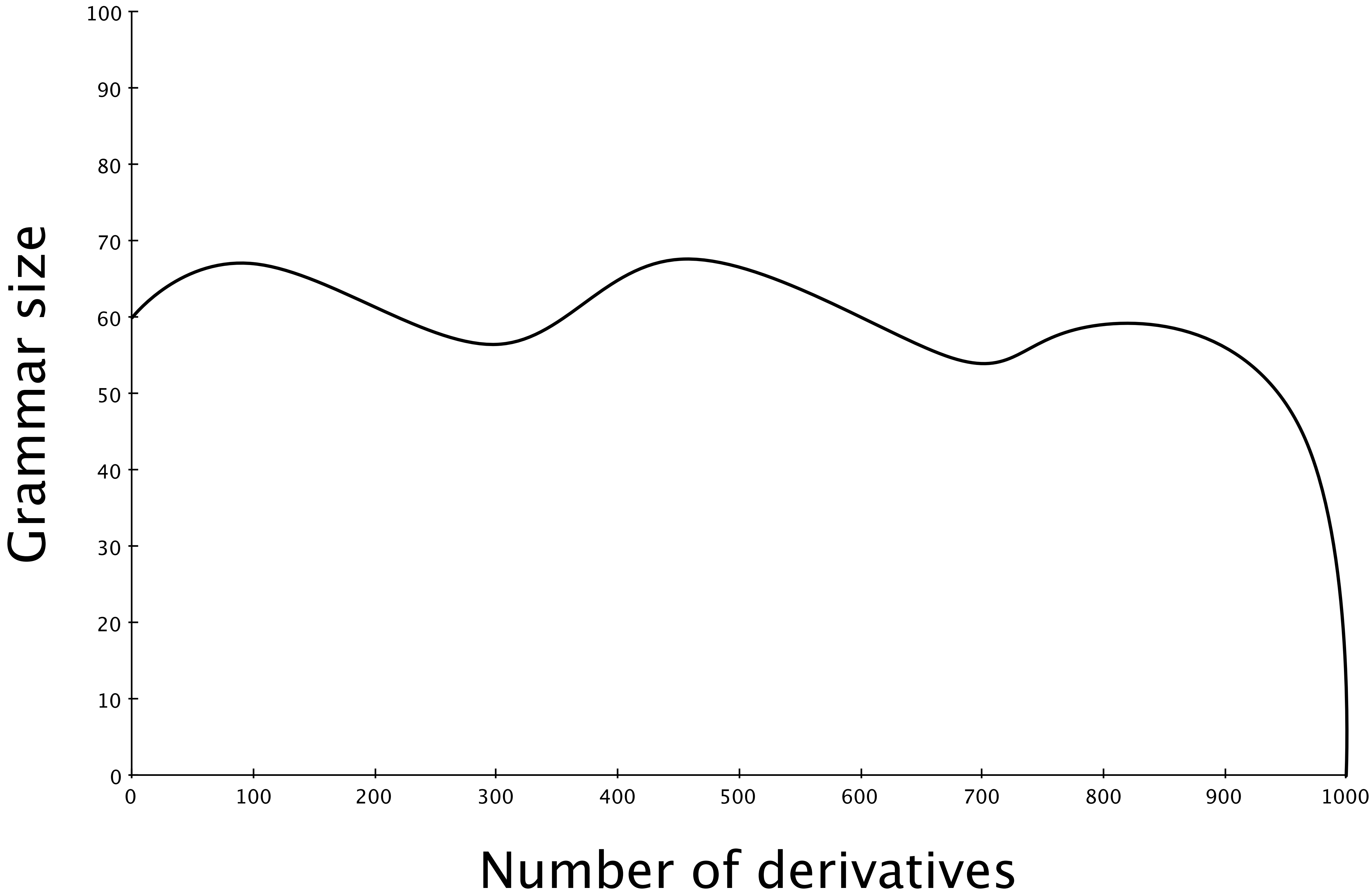
```
(define/memoize (compact [l #:eq])
  (match l
    [(empty)       l]
    [(eps)         l]
    [(emptyp)      (empty)]
    [(nullp)       (eps* (parse-null l))]
    [(token p c)   l]

    [(orp (emptyp) l2)  (compact l2)]
    [(orp l1 (emptyp))  (compact l1)]

    [(seqp (nullp t) l2)  (red (compact l2) (lambda (w2) (cons t w2)))]
    [(seqp l1 (nullp t))  (red (compact l1) (lambda (w1) (cons w1 t)))]

    [(orp l1 l2)   (alt (compact l1) (compact l2))]
    [(seqp l1 l2)  (cat (compact l1) (compact l2))]

    [(redp (and e (nullp)) f)
     (eps* (for/set ([t (parse-null e)]) (f t)))]

    [(redp (seqp (nullp t) l2) f)
     (red (compact l2) (lambda (w2) (f (cons t w2))))]

    [(redp (redp l f) g)
     (red (compact l) (lambda (w) (g (f w))))]

    [(redp l f)    (red (compact l) f)]))
```



or
eps*
(set '(() . 1))
seq
L
R
token
1


or
eps*
(set '((((((() . 1) . 1) . 1) . 1) . 1))
seq
L
R
token
1


or
eps*
(set '((((((((((() . 1) . 1) . 1) . 1) . 1) . 1) . 1) . 1) . 1) . 1))
seq
L
R
token
1


Grammar size
100
90
80
70
60
50
40
30
20
10
0
0
100
200
300
400
500
600
700
800
900
1000
Number of derivatives

Average complexity?

$$O\left(n|G|\right)?$$

What we need

- Allow non-blocking I/O
- Act as library within the language
- Handle all CFGs: parse forest
- Draw on regular expressions
- Cacheable partial parses
- Be easy to implement

What we need

- ✓ Allow non-blocking I/O
- Act as library within the language
- Handle all CFGs: parse forest
- Draw on regular expressions
- Cacheable partial parses
- Be easy to implement

What we need

- ✓ Allow non-blocking I/O
- ✓ Act as library within the language
- Handle all CFGs: parse forest
- Draw on regular expressions
- Cacheable partial parses
- Be easy to implement

What we need

- ✓ Allow non-blocking I/O
- ✓ Act as library within the language
- ✓ Handle all CFGs: parse forest
- Draw on regular expressions
- Cacheable partial parses
- Be easy to implement

What we need

- ✓ Allow non-blocking I/O
- ✓ Act as library within the language
- ✓ Handle all CFGs: parse forest
- ✓ Draw on regular expressions
- Cacheable partial parses
- Be easy to implement

What we need

- ✓ Allow non-blocking I/O
- ✓ Act as library within the language
- ✓ Handle all CFGs: parse forest
- ✓ Draw on regular expressions
- ✓ Cacheable partial parses
- Be easy to implement

What we need

- ✓ Allow non-blocking I/O
- ✓ Act as library within the language
- ✓ Handle all CFGs: parse forest
- ✓ Draw on regular expressions
- ✓ Cacheable partial parses
- ✓ Be easy to implement

Would be nice

- Dynamic reconfigurability
- Beyond context-free languages
- Built into the language
- Support compositionality

Would be nice

- ✔ Dynamic reconfigurability
- Beyond context-free languages
- Built into the language
- Support compositionality

Would be nice

- ✓ Dynamic reconfigurability
- ✓ Beyond context-free languages
- Built into the language
- Support compositionality

Would be nice

- ✓ Dynamic reconfigurability
- ✓ Beyond context-free languages
- ✓ Built into the language
- Support compositionality

Would be nice

- ✓ Dynamic reconfigurability
- ✓ Beyond context-free languages
- ✓ Built into the language
- ✓ Support compositionality

Further reading

- Brzozowski. JACM 1964.
- Owens, Reppy, Turon. JFP 2010.
- Danielsson. ICFP 2010.
- Might, Darais. arXiv 2010.

Thank you!

Want more?

See blog.might.net