TyCL v1.0

Typed Command Language

Andres Buss

Otlet Technologies

What is TyCL?

- Is attempt to create a compiler for the Tcl/Tk language.
- Is a runtime-interpreter.
- Is the “extended” syntax of the language that the compiler understands. (Mostly the addition of “optionally” direct type declarations in the source code)

The “painful” evolution of TyCL

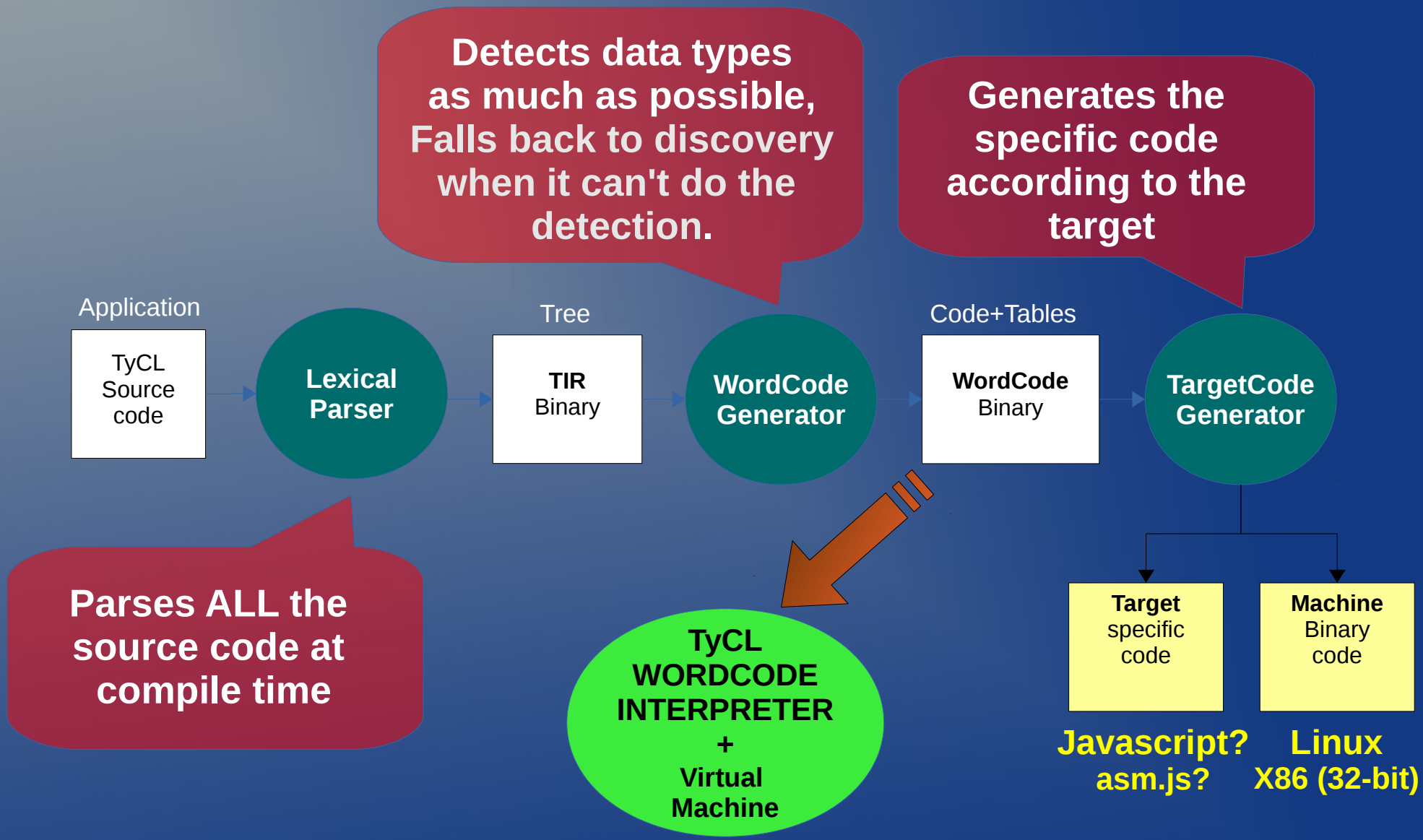
- It started as an exercise to add a prototype based OOP to Tcl. *(TyCL v.-2.x)*
- Then, as an standalone interpreter written in C. *(TyCL v.-1.x)* – Never completed
- Then, as an standalone interpreter and later a compiler, written in Tcl8.6 *(TyCL v.0.x)*
- And finally, as an standalone compiler written in *(TyCL v.+1.x)*

The “painful” evolution of TyCL

- It started as an exercise to add a prototype based OOP to Tcl. *(TyCL v.-2.x)*
- Then, as an standalone interpreter written in C. *(TyCL v.-1.x)* – Never completed
- Then, as an standalone interpreter and later a compiler, written in Tcl8.6 *(TyCL v.0.x)*
- And finally, a standalone compiler written in *TyCL (v.+1.x)*

BOOTSTRAPPING

Architecture



FORBIDS ANY DYNAMIC EVALUATION AT RUNTIME … no eval, no source commands

A TyCL application

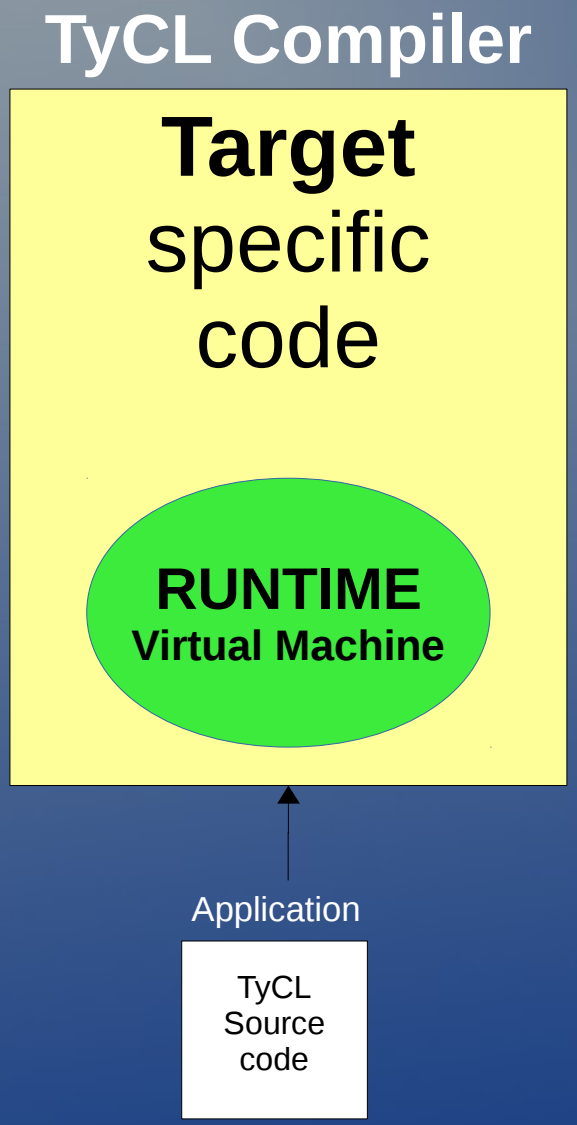
WordCode
Binary

TyCL WORDCODE INTERPRETER
+
Virtual Machine

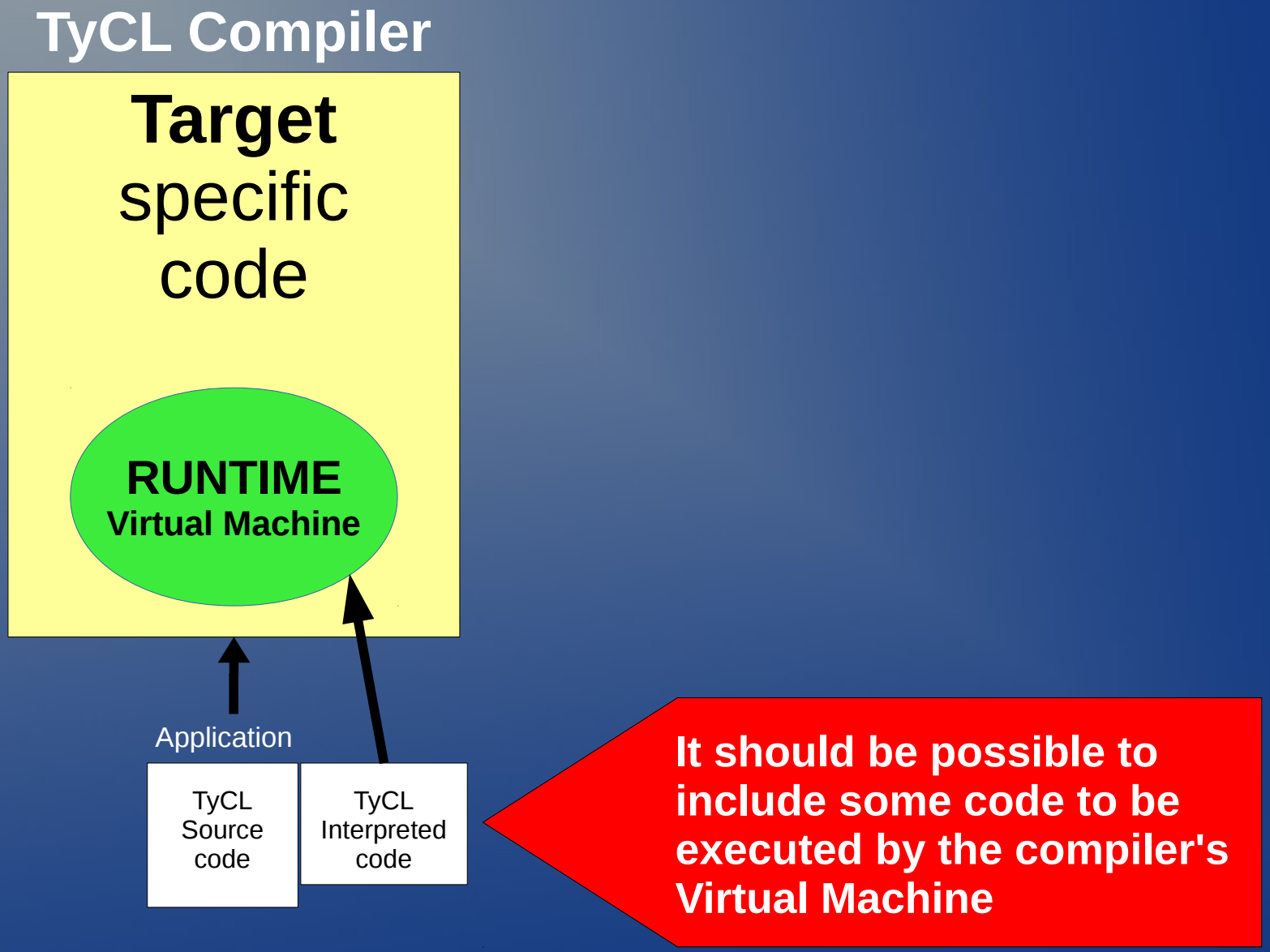
Target
specific
code

RUNTIME
Virtual Machine

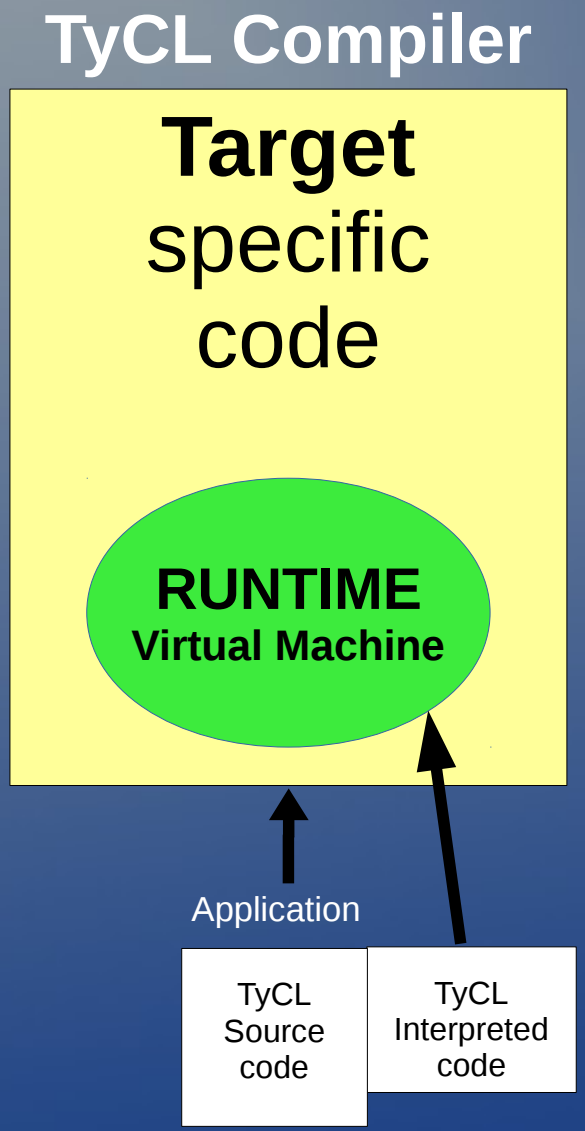
TyCL as a TyCL application



TyCL as a TyCL application



TyCL as a TyCL application



If this is allowed … the whole compiler could be modified at compile time!!

It should be possible to include some code to be executed by the compiler's Virtual Machine

How to run code in compile time?

How to run code in compile time?

Percent commands

How to run code in compile time?

Percent commands

%puts

%set

%incr

%proc

%eval

%source

%include

%exit

Other percent commands

%if : Conditional selection of source-code to compile.

Example:

```
%if {$size == 8} {
    define x i8:0

} elseif {$size == 16} {
    define x i16:0

} else {
    define x i32:0
}
```

Other percent commands

%macro : Defines an “active-macro”.

Syntax:

```
%macro  IDENTIFIER  PARAMETERS  "TEXT_SUBSTITUTE"
```

Examples:

```
%macro FREE {o} "\[.MEM.free $o\]"

%proc .foo {x} { return “0.$x” }    ;# Creates a runtime-function
%macro BAR {i} “[.foo $i]”          ;# Calls the 'foo' function
```

Calling/Using a macro

Macro's substitution

Syntax :

$<IDENTIFIER ARG1 ARG2 . . . ARGn>

Examples:

if {$v < 1} { set r $<FREE $p> }

if {$v < 1} { set r [.MEM.free $p] }

set v $<BAR 55>

set v 0.55

Now, something more interesting ….

Now, something more interesting ….

If we can run code at compile time, we should be able to change the compiler as it compiles the application's source code... variables and functions could be added or modified.

Now, something more interesting ….

If
s
a
c
be added or modified.

The only problem is that the compiler (and all its components) have to be coded in way that allows these changes as less dificult as posible.

Now, something more interesting ….

The only proble… …he
compiler (a… …s)
have t… …t allows
the… …less dificult as
possi…

And … as you can imagine right now …
it was not the case!

If…
s…
a…
c…
be added or modified.

The parser

- Originally “hardwired” coded
- Now … rewritten in v1.0

The parser uses a table of declarative rules and a set of functions that operate over those rules...

Example: This is the main TyCL rule:

PARSER.addRule "LANG" "all" {@STATEMENT}*

The parser

Current syntax:

```
PARSER.addRule                                          \
    RULE_NAME                                           \
    FUNCTION                                            \
    DESCRIPTION                                         \
    ?-tokid TOKNAME?                                    \
    ?-errmsg ERROR_MESSAGE_IF_FAILED_MATCH?
```

Examples:

```
PARSER.addRule   "LANG"   "all"   {@STATEMENT*}

PARSER.addRule   "STATEMENT"  "any" \
      {";"  @SPACENL+~  @COMMENT~ @NATCMD @COMMAND  @POPSTATE}

PARSER.addRule   "COMMAND"    "all"                     \
    {@CMDNAME   @ARGUMENT*  @SPACE*~  @EOCMD}         \
    -tokid "COMMAND" -errmsg "Invalid command"
```

The parser

Current Available functions:

- *all { DESC }*
  → Match all of the elements
- *any { DESC }*
  → Match any of the elements
- *if {STRING {TRUE_DESC} {FALSE_DESC}}*
  → If STRING exists match TRUE_DESC or else FALSE_DESC
- *blk {LEFT_STRING {DESC} RIGHT_STRING}*
  → Matches left & right strings …. besides the descriptor
- *exec FUNCTION_NAME*
  → Execute the function
- *push TOKID*
  → pushes a token into the token-stack of id TOKID if TOKID is provided
- *pop TOKID*
  → pops a token from the token-stack of id TOKID if TOKID is provided

The parser

Current Available functions:

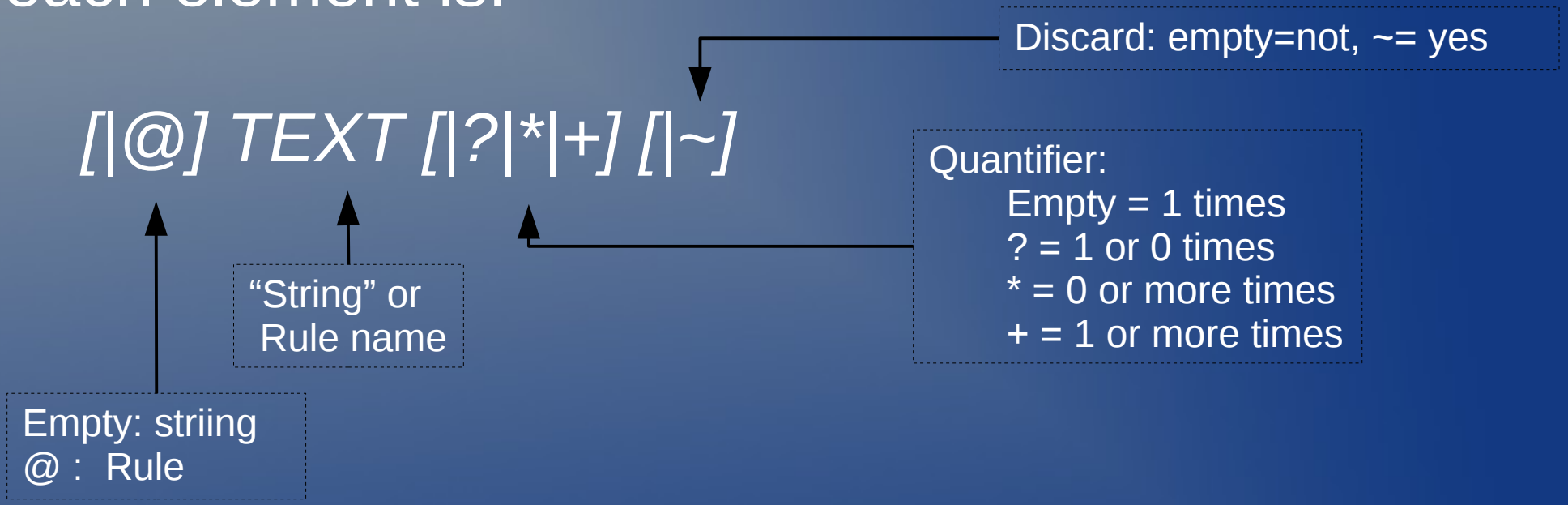
More functions can be dynamically created

- *all { DESC }*
  → Match all of the elements
- *any { DESC }*
  → Match any of the elements
- *if {STRING {TRUE_DESC} {FALSE_DESC}}*
  → If STRING exists match TRUE_DESC or else FALSE_DESC
- *blk {LEFT_STRING {DESC} RIGHT_STRING}*
  → Matches left & right strings …. besides the descriptor
- *exec FUNCTION_NAME*
  → Execute the function
- *push TOKID*
  → pushes a token into the token-stack of id TOKID if TOKID is provided
- *pop TOKID*
  → pops a token from the token-stack of id TOKID if TOKID is provided

The parser

The descriptors (*DESC*)

List of elements: *{ ELEM0 ELEM1 … ELEMn }* Where each element is:



Examples:

- Strings: “if” , “{“~ , “3”*~
- Rules: @COMMAND , @VAL?

The TyCL syntax description (1/2)

```
PARSER.addRule   "LANG"          "all"    {@STATEMENT*}
PARSER.addRule   "STATEMENT"     "any"    {";"  @SPACENL+~  @COMMENT~ @NATCMD @COMMAND  @POPSTATE}

PARSER.addRule   "POPSTATE"      "exec"   "_POPSTATE"
PARSER.addRule   "NATCMD"        "exec"   "_NATCMD"

PARSER.addRule   "COMMAND"       "all"    {@CMDNAME   @ARGUMENT*  @SPACE*~  @EOCMD}        -tokid "COMMAND"      -errmsg "Invalid command"
PARSER.addRule   "CMDNAME"       "all"    {@VARPATH}                                                              -errmsg "Invalid command name"

PARSER.addRule   "ARGUMENT"                "all"          {@SPACE+~ @ARGELEM}
PARSER.addRule   "ARGELEM"       "any"    {@NUMBER @ARGFLAG @ARGNAMED @WORD}
PARSER.addRule   "ARGFLAG"       "if"     {"--"  {"--" @VARSTR @EOW} }                    -tokid "FLAGVAR"      -errmsg "Invalid flag-argument"
PARSER.addRule   "ARGNAMED"                "eif"          {"-"  {"-" @VARSTR @SPACE+~ @WORD} }    -tokid "NAMEDVAR"     -errmsg "Invalid named-argument (bad name or missing value)"

PARSER.addRule   "EOCMD"         "exec"   "_EOCMD"
PARSER.addRule   "EOW"           "exec"   "_EOW"

PARSER.addRule   "VARPATH"       "any"    {@GLBVAR @OBJVAR @LOCVAR}                                               -errmsg ">>>Invalid variable"
PARSER.addRule   "GLBVAR"        "if"     {"."  {"." @VARDESC} }                -tokid "GFIND"
PARSER.addRule   "OBJVAR"        "if"     {"my."  {"my." @VARDESC} }            -tokid "OFIND"
PARSER.addRule   "LOCVAR"        "all"    {@VARDESC}                            -tokid "LFIND"

PARSER.addRule   "VARDESC"       "all"    {@VARNAME  @VARITEM*}                                                   -errmsg "Invalid member, extra characters"
PARSER.addRule   "VARNAME"       "all"    {@VARSTR}                                                               -errmsg "Invalid variable/command  name"
PARSER.addRule   "VARITEM"       "any"    {@MEMBER @INDRNG}                                                       -errmsg ">>>> Invalid member name"
PARSER.addRule   "MEMBER"        "all"    {"." @MEMBERNAME}
PARSER.addRule   "MEMBERNAME"    "any"    {@QTEXT @BTEXT @MTEXT}
PARSER.addRule   "MTEXT"         "exec"   "_MTEXT"


PARSER.addRule   "INDRNG"        "blk"    {"(" {@SPACE*~ @PUSHEXPR @SPACE*~ @RANGE} ")"}                                                  -errmsg "Extra characters or missing close-parenthesis"
PARSER.addRule   "RANGE"         "if"     {".." {@POPEXPR  ".."  @SPACE*~  @EXPR  @SPACE*~} {@INDEX} }    -tokid "RANGE"
PARSER.addRule   "INDEX"         "if"     {"" {@POPEXPR  @SPACE*~}}                                       -tokid "INDEX"        -errmsg "Invalid index value"
PARSER.addRule   "PUSHEXPR"      "push"   "EXPR"                                  -errmsg "Invalid value"
PARSER.addRule   "POPEXPR"       "pop"    ""                                      -errmsg "Missing value"

PARSER.addRule   "EXPR"          "exec"   "_EXPR"
PARSER.addRule   "EXPR_VAL"      "any"    {@SUBEXPR  @GETVAL  @SUBCMD  @QTEXT @BTEXT @NUMBER  @EFUNC @EVTEXT}
PARSER.addRule   "SUBEXPR"       "blk"    {"(" @EXPR  ")"}
PARSER.addRule   "EVTEXT"        "exec"   "_EVTEXT"

PARSER.addRule   "EFUNC"         "any"    {ninguno}     "EFUNC" ""

PARSER.addRule   "WORD"                    "any"          {@GETVAL  @SUBCMD  @PEXPR @QTEXT @BTEXT @NUMBER  @CASTVAL @TEXT}
PARSER.addRule   "VALWORD"       "any"    {@GETVAL  @SUBCMD  @PEXPR @QTEXT @BTEXT @NUMBER  @TEXT}
PARSER.addRule   "STRONLY"       "any"    {@QTEXT @BTEXT @TEXT}

PARSER.addRule   "CASTVAL"       "exec"   "_CASTVAL"
PARSER.addRule   "CASTVAR"       "all"    {@VARSTR ":" @VARSTR}                   -tokid "DEFVAR"
PARSER.addRule   "CASTVAR_PTR"   "all"    {@VARSTR ":@" @VARSTR}                  -tokid "DEFVARPTR"
PARSER.addRule   "CASTVAR2"      "all"    {@VARSTR ":" @SPACE*~  @VARSTR}         -tokid "DEFVAR"
PARSER.addRule   "CASTVAR_PTR2"  "all"    {@VARSTR ":" @SPACE*~  "@"  @VARSTR}    -tokid "DEFVARPTR"

PARSER.addRule   "GETVAL"        "if"     {"\$" {"\$" @VARPATH} }                 -tokid "GETVAL"

PARSER.addRule   "SUBCMD"        "blk"    {"\[" {@SPACE*~  @BCOMMAND  @SPACE*~}  "\]"}    -tokid "SUBCMD"
PARSER.addRule   "BCOMMAND"      "any"    {@NATCMD @COMMAND}

PARSER.addRule   "PEXPR"         "if"     {"(" @SUBEXPR}

PARSER.addRule   "BODY"          "blk"    {"\{"  {@STATEMENT*} "\}"}              -tokid "BLOCK"        -errmsg "Expecting a block/body of code"
PARSER.addRule   "ARGEXPR"       "if"     {"\{" @BEXPR @EXPR}
PARSER.addRule   "BEXPR"         "blk"    {"\{" @EXPR "\}"}

PARSER.addRule   "ARGFONLY"                "any"          {@ARGFLAG @NOARGNAMED}                                  -errmsg "Missing flag"
PARSER.addRule   "ARGNAMEONLY"   "any"    {@ARGNAMED @NOARGFLAG}                                                  -errmsg "Missing named-argument"
PARSER.addRule   "ARGNAMEFLAG"   "any"    {@ARGFLAG @ARGNAMED}                                                    -errmsg "Missing flag/named argument"
PARSER.addRule   "NOARGFLAG"     "eif"    {"--"  {}}                                                              -errmsg "Invalid flag-argument"
PARSER.addRule   "NOARGNAMED"    "eif"    {"-"  {}}                                                               -errmsg "Invalid named-argument"
```

The TyCL syntax description (2/2)

```
PARSER.addRule    "COMMENT"       "any"     {@COMMENTLN @COMMENTBLK}
PARSER.addRule    "COMMENTLN"     "exec"    "_COMMENTLN"
PARSER.addRule    "COMMENTBLK"    "exec"    "_COMMENTBLK"

PARSER.addRule    "SPACE"         "exec"    "_SPACE"
PARSER.addRule    "SPACENL"       "exec"    "_SPACENL"

PARSER.addRule    "VARSTR"        "exec"    "_VARSTR"

PARSER.addRule    "BTEXT"         "exec"    "_BTEXT"
PARSER.addRule    "QTEXT"         "exec"    "_QTEXT"
PARSER.addRule    "TEXT"          "exec"    "_TEXT"

PARSER.addRule    "NUMBER"        "exec"    "_NUMBER"
PARSER.addRule    "INTEGER"       "exec"    "_INTEGER"
PARSER.addRule    "REAL"          "exec"    "_REAL"


PARSER.addRule    "PARAMETERS"    "if"      {"\{"  @PARAMSOPT  @VARSTR }                    -tokid "PARAMS"      -errmsg "Invalid parameter-descriptor"
PARSER.addRule    "PARAMSOPT"     "any"     {@NOPARAMS @PARAMSDESC}
PARSER.addRule    "NOPARAMS"      "blk2"    {"\{"  @SPACE*~  "\}"}
PARSER.addRule    "PARAMSDESC"    "blk"     {"\{"  {@PARAM0 @PARAM* @SPACENL*~} "\}" }                                -errmsg "Invalid parameter"
PARSER.addRule    "PARAM0"        "all"     {@SPACENL*~ @PARAMELEM}                                                   -errmsg "Invalid parameter"
PARSER.addRule    "PARAM"         "all"     {@SPACENL+~ @PARAMELEM}                                                   -errmsg "Invalid parameter"
PARSER.addRule    "PARAMELEM"     "any"     {@ARGFLAG @PARAMNO @VARSTR}                                               -errmsg "Invalid paramater"
PARSER.addRule    "PARAMNO"       "blk2"    {"\{"  @PARAMNAMED  "\}" }
PARSER.addRule    "PARAMNAMED"    "if"      {"-"  {"-" @VARSTR @SPACE+~ @WORD @SPACE*~} @PARAMOPT}    -tokid "NAMEDVAR"    -errmsg "Invalid parameter"
PARSER.addRule    "PARAMOPT"      "all"     {@VARSTR @SPACE+~ @WORD @SPACE*~}                         -tokid "OPTVAR"      -errmsg "Invalid parameter"


PARSER.addRule    "ALLPARAMS"     "any"     {@ALLPARAMS_ @NOPARAMS_}
PARSER.addRule    "NOPARAMS_"     "all"     {@SPACE*~}
PARSER.addRule    "ALLPARAMS_"    "all"     {@PARAM0 @PARAM* @SPACENL*~}


PARSER.addRule    "ALLARGS"       "all"     {@ARGELEM @ARGSEP @ALLARGS?}
PARSER.addRule    "ARGSEP"        "if"      {" " @SPACE+~ @ARGTAB}
PARSER.addRule    "ARGTAB"        "if"      {"\t" @SPACE+~ @OK}

PARSER.addRule    "WORDSEPNL"     "if"      {"\n" @SPACENL+~  @WORDSEP}
PARSER.addRule    "WORDSEP"       "if"      {" " @SPACE+~  @WORDTAB}
PARSER.addRule    "WORDTAB"       "if"      {"\t" @SPACE+~  @OK}


PARSER.addRule    "OK"                      "exec"        "_OK"
```

Extending the language

Compiled commands:

- PARSER.parseCmd NAME {DESC} BODY_GENERATOR
- PARSER.command NAME {PARAMETERS} BODY_GENERATOR

Example:

```
PARSER.command "puts" {--nonewline {str ""} } {
    if $VAR(nonewline) {
        PUTS $VAR.str
    } else {
        PRINT $VAR.str
    }
    return -1
}
```

Extending the language

Compiled commands:

```
PARSER.addrule "_ELSEIF_"    if {"elseif" {"elseif" @SPACE+~ @ARGEXPR
@SPACE+~ @BODY @SPACE*~ @_ELSEIF_?} @_ELSE_?}          "IF"

PARSER.addrule "TyCL" "_ELSE_"  if {"else" {"else" @SPACE+~ @BODY}}


LANG.parseCmd  "if"  {@ARGEXPR @BODY ?_ELSEIF_?} {

      if {$__ntoks__  == 2} {
            return [__IF  $__rPtr__  $TOK(1)  $TOK(2) ]
      } elseif {$__ntoks__  == 3} {
            return [__IF  $__rPtr__  $TOK(1)  $TOK(2)  $TOK(3)]
      } else {
            error $TOK(0).line $TOK(0).pos $TOK(0).filen
      }
}
```

Extending the language

C - compiled functions:

TyCL has the ability to call c-compiled functions directly without any wrappers.

Example:

Having a C-function: int sum(int a, int b) in a library.

```
cproc sum sum {i32:* i32:a i32:b}

set a 3
set b 5
puts “$a + $b = [sum $a $b]
```

Extending the compiler

- Creating new native types
- Modifying the Word-Code Generator
  - Creating new opcodes
  - Adding a “Debugging” set of opcodes
  -
- Modifying de Target-Code Generator
  - Creating new ways to transform the opcodes
  - Adding new targets

Roadmap for v2.0

- Have some “infrastructure” stabilization
- Add support for X86_64 (New assembler)
- Add official support for Javascript (asm.js)
- Have some documentation and a WEB page
- Release de source code (BSD licence)

More information:

Andres Buss

aabuss@otlettech.com

aabuss@gmail.com