Pontifícia Universidade Católica
do Rio de Janeiro

How Much Does it Cost?

Roberto Ierusalimschy

Some Recent Proposals

- Alternative way of format/pack/unpack
- UTF-8–aware scanner
- new ':.' syntax
- new pattern `'%B'`
- `'__tostring'` can use `'__name'`

- adding `[]` as table creation syntax
- new array type
- `'table'` as fallback for tables
- `'__key'` metamethod
- light syntax for anonymous functions

UTF-8–aware scanner

```
,,,= print

function function  (function)
  if function == 0 then return 1
  else return function * function (function - 1)
  end
end

,(function (10))        --> 3628800
```

Varargs

Integers

Iterators

Goto

RAII
support

Unicode-aware scanner

What is the cost of a new feature in a programming language?

Integers

Coroutines

Safe navigation

Try-catch

Lines of Code

- Implementation effort
- “Why not? It's so easy!!”
- Quite relevant

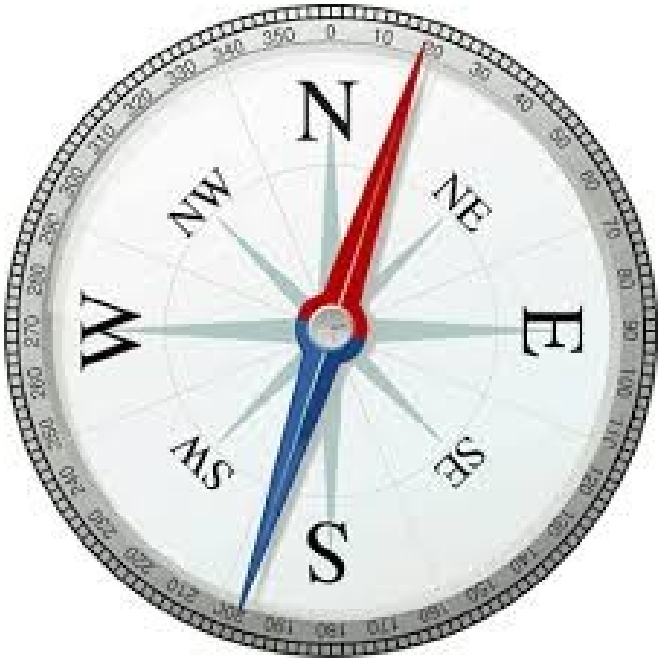
```
static StkId adjust_varargs (lua_State *L, Proto *p, int actual) {
  int i;
  int nfixargs = p->numparams;
  StkId base, fixed;
  lua_assert(actual >= nfixargs);
  /* move fixed parameters to final position */
  luaD_checkstack(L, p->maxstacksize);  /* check again for new 'base' */
  fixed = L->top - actual;  /* first fixed argument */
  base = L->top;  /* final position of first argument */
  for (i=0; i<nfixargs; i++) {
    setobjs2s(L, L->top++, fixed + i);
    setnilvalue(fixed + i);
  }
  return base;
}
```

Performance

- Does the feature help code that uses it?
- Does it impair code that does not use the feature?
- Does it create large performance variations for special cases?

Documentation

- Is it hard to explain?
- Is it hard to understand?
- Is it hard to guess?



Conceptual Integrity

- Does it fit well with other features?
- Does it follow *general principles*?

Generality

- Does it solve all problems in a class?
- Does it solve related problems?
- Does it solve unrelated problems?

Testing

- Is it hard to test?
- Is it hard to debug?
- Bugs can be severe?

Obstacle to Evolution

- Does it narrow the design space?
- Does it restrict alternative implementations?

Some Case Studies

Equality

- Lua 1 used '=' for comparisons.
  - syntax allowed the distinction between comparisons and assignments.
  - Lua 1 used `@{a = b}` for records and `@[a = b]` for sequences.
  - Lua 2 unified all constructors with `{}`.
  - Result was ambiguity in `{a = b}`.

Comments

- Comments with `'--'` preclude a C-style decrement operator.
  - which mostly precludes a C-style increment operator.
  - which precludes a new language: Lua++
- Comments with `'[[...]]'` have problems with `t[x[i]]`
  - that was before `'[===['`.

Multiple Returns

- Powerful
  - generic-call mechanism: `f(table.unpack(t))`
- Conceptual integrity
  - a second class data-structure mechanism hidden inside the language.
  - nil's become more significant.

Multiple Returns

- Restrictions on implementations
  - stack frame has variable size.
  - C functions must return an integer.
  - translations to other languages cannot use native return mechanism.
- Small cost even when not used
  - every return must check (and adjust) number of results.

Varargs (“new style”)

- Conceptual integrity
  - a second class data-structure mechanism hidden inside the language.
  - demonizes table creation.
- Small cost even when not used
  - the framework for supporting varargs is used in all Lua functions.

Incremental Garbage Collector

- Negligible costs in documentation
- High costs in testing
  - very hard to debug, aggravated by the C API.
  - bugs are severe.
  - very hard to test, due to uncontrolled interactions between the mutator and the collector.

Finalizers and Weak Tables

- Key ingredient in the Lua-C API and in several Lua idioms
- Big impact on alternative implementations
  - hard to use a native collector
- Some cost for the garbage collector
  - mainly when using the features

Length Operator (`#t`)

- Documentation
  - easy to explain
  - difficult to understand
- Conceptual integrity
  - fights with varargs and multiple returns: `{...}`, `{f(x)}`

String methods

- `s:len()`, `s:toupper()`
- Trivial implementation
- Trivial testing
- Conceptual integrity
  - only way in Lua to call library functions out of raw syntax
  - particularly bad for sandboxing

Coroutines

- Small cost even when not used
  - no way to know it won't be used!
- Implementation
  - spread over the interpreter
  - nasty interaction between coroutines, garbage collection, and closures
  - hard for JITs
- Hard to test
  - all situations where yields can occur

Integers

- Many lines of code
- Small price when not using it
  - one extra check before float arithmetic
- Some gains when using it
  - integer operations faster in several architectures
  - avoid conversions float $\Longleftrightarrow$ int

Integers

- Conceptual integrity
  - Several “integer types” already present in the language: arguments to library functions, concept of sequence, bitwise library, C API, etc.
  - Ill-defined limits
  - Ill-defined conversions

Final Remarks

- Many applications can rip big benefits from small and simple additions to a language. But each application needs different small and simple additions.
- It is hard to foresee all consequences of a new feature.
- To avoid bloating the language, features should be measured not by themselves, but against competing features.

Final Remarks

- We can err both by adding a bad feature or by blocking a good feature.

Final Remarks

- We can err both by adding a bad feature or by blocking a good one.
- One of these errors is much easier to fix than the other.

PONTIFÍCIA
UNIVERSIDADE
CATÓLICA
DO RIO DE JANEIRO