Testing Ling

August 29, 2012

Ling is a reincarnation of Erlang VM and as such should exhibit behaviour sufficiently similar to its widely used predecessor, BEAM. An extensive testing was conducted to demonstrate the similarity. The overview of the results of the testing are summarized in the present document.

A subset of 25 test suites were selected from the BEAM emulator testing framework:

- lists_SUITE
- tuple_SUITE
- list_bif_SUITE
- binary_SUITE
- bs_bincomp_SUITE
- bs_bit_binaries_SUITE
- bs_construct_SUITE
- bs_match_bin_SUITE
- bs_match_int_SUITE
- bs_match_misc_SUITE
- bs_match_tail_SUITE
- bs_utf_SUITE
- exception_SUITE
- big_SUITE
- float_SUITE
- fun_SUITE
- guard_SUITE
- num_bif_SUITE
- ref_SUITE
- ets_SUITE
- file_SUITE
- gen_tcp_api_SUITE
- gen_tcp_echo_SUITE
- re_SUITE
- decode_packet_SUITE

The subset addresses functions of a generic Erlang VM. Certain functions, such as driver subsystem, call tracing, etc are intentionally were left out as these functions are implemented differently in Ling or excluded completely.

The source code of the test suites was not modified, except for file_SUITE and gen_tcp_echo_SUITE. All modifications to the source code are noted in the corresponding section below. The summary of the results can be found in the Table 1.

Test suite	Total	Ok	Failed	Skipped
lists SUITE	68	68	-	-
tuple SUITE	12	12	-	-
list bif SUITE	6	6	-	-
binary SUITE	32	27	2	3
bs bincomp SUITE	6	5	-	1
bs bit binaries SUITE	11	10	-	1
bs construct SUITE	21	19	1	1
bs match bin SUITE	3	3	-	-
bs match int SUITE	8	8	-	-
bs match misc SUITE	16	16	-	-
bs match tail SUITE	3	3	-	-
bs utf SUITE	7	7	-	-
exception SUITE	10	10	-	-
big SUITE	14	13	-	1
float SUITE	11	8	1	2
fun SUITE	18	10	1	7
guard SUITE	6	5	1	-
num bif SUITE	9	9	-	-
ref SUITE	1	1	-	-
ets SUITE	108	79	12	17
file SUITE	65	50	5	10
gen tcp api SUITE (9p)	10	6	4	-
gen tcp echo SUITE (9p)	9	9	-	-
gen tcp api SUITE (lwIP)	10	9	1	-
gen tcp echo SUITE (lwIP)	9	9	-	-
re SUITE	15	15	-	-
decode packet SUITE	9	9	-	-

Table 1: The summary of test results.

binary SUITE

1. `bad_binary_to_term_2` attempts to start a new node (skipped).
2. `obsolete_funs` tests obsolete funs represented as binaries (skipped).
3. `gc_test` uses undocumented process info(Pid, binary) call (skipped).
4. `terms_float` relies on snprintf() when converting a double to an external term format. The Ling's snprintf() implementation produces a slightly different string that results in mismatch after unpacking. The possible cause are compiler options related to the floating-point math (fails).
5. `deep` creates a structure a million levels deep. The routine copying terms between heaps is recursive and thus unable to handle the structure. A system limit exception is thrown (fails).

bs_bincomp_SUITE

1. `tracing` uses call tracing not supported in Ling. A simpler call tracing is possible in the new VM (skipped).

bs_bit_binaries_SUITE

1. `append` uses undocumented and unsupported erts debug() calls (skipped).

bs_construct_SUITE

1. `bs_add` uses a compiler. While it is possible to import all compiler modules into Ling, it seems an overkill to make a few tests run. All iterpreter modules are imported as many more tests use them (skipped).
2. `huge_binary` is unable to build a binary because the maximum number of bits for a binary in Ling is 7 bits smaller than in BEAM (4294967288 vs 4294967295). Smaller maximum bit size means the ability to use 32-bit size representation in many places (fails).

big_SUITE

1. `big_literals` uses compiler (skipped).

float_SUITE

1. `fp_drv` uses driver subsystem (skipped).
2. `fp_drv_thread` uses driver subsystem (skipped).
3. `cmp_bignum` exhibits different rounding behaviour not related to Erlang VM. For instance, an integer value of 288230376151711744 is (automatically) rounded in C to 288230376151711750.0 in Ling and to 288230376151711740.0 in BEAM. It may be dependent on the compiler options (fails).

fun_SUITE

1. `fun_to_port` uses unsupported port command() call (skipped).
2. `refc` uses undocumented fun info(Fun, refc) call. It may represent an implementation detail related to funs (skipped).
3. `refc_ets` see `refc` (skipped).
4. `refc_dist` see `refc` (skipped).
5. `const_propagation` see `refc` (skipped).
6. `t_arity` starts a new node (skipped).
7. `t_fun_info` see `refc` (skipped).

8. `t_is_function2` treats certain 2-tuples as funs. Ling does not support obsolete funs (fails).

guard SUITE

1. `guard_bifs` directly tests two floating-point values for equality. The values appear to be (slightly) different (fails).

ets SUITE

1. `t_delete_object` uses undocumented `ets:info(T, stats)` call to make an implementation-dependent check; works if the check is removed.
2. `t_delete_all_objects` see t delete object.
3. `smp_insert` uses undocumented `ets:info(T,stats)` call.
4. `smp_select_delete` same as smp insert.
5. `memory` is a joke; it uses hardcoded values for various undocumented parameters.
6. `heir` uses undocumented `erts_debug:get_internal_state()` to reset the pid numbering to ease reuse of pids; hangs because the function is not supported; other bits of the case work.
7. `t_match_spec` starts a node to get external things, works if these small dependencies are removed.
8. `types` starts a node to make external things; this useful case should be reimplemented without the dependency.
9. `fixtable_insert` fails only the last small case that verifies that semi-deleted entries should be deleted immediately when the table is unfixed. This requires additional full-scan through the table or maintaining a list of semi-deleted entries. Ling uses keeps semi-deleted entries when the table is unfixed and removes them opportunistically.
10. `partly_bound` verifies the perfromance gain for key bound match specs; faster code for such match specs are not implemented yet - they still use the full scan.
11. `t_whitebox` uses a continuation generated by `ets:match/3` in the call to `ets:select/1`; such use is not supported.
12. `t_repair_continuation` tests a function that is not required for Ling as a match spec gets compiled every time before use; to be stubbed later to make Mnesia, etc happy.

file SUITE

1. `exclusive` O EXCL mode is not supported by diod (fails).

2. `truncate` Ling allows truncating a file opened read-only. It may be possible to implement the restriction but the potential gain vs the overhead involved is small (fails).

3. `file_info_times` the access and modification times need timezone information that is not yet handled in Ling properly (fails).

4. `delayed_write` delayed write option is not supported by Ling (fails).

5. `read_ahead` read ahead option is not supported by Ling (fails).

6. `read_line_` see `read_ahead` (fails).

   The modifications to the source code of file SUITE.erl are mostly related to the fact that Ling allows deleting the current directory (if empty) and that `os:type()` returns 'ling'. Also 'enotempty' is returned instead of 'eexist' when renaming a directory to an existing non-empty directory. Several modifications were made to decrease the number of iterations and operate on smaller data segments to avoid 'no memory' exceptions. The test suite was also modified to avoid use of regular expressions that are not supported by Ling yet.

gen tcp api SUITE 9p

1. `t_shutdown_write, t_shutdown_both, t_shutdown_error` test unsupported inet:shutdown() function (fails).

2. `t_fdopen` tries to retrieve a file descriptor to base a socket on (fails).

gen tcp echo SUITE 9p

The source code of the test suite was modified to exclude checks for the correct packing/unpacking for the following packet types: sunrm, cdr, line, http, http bin, asn1. The support for some of the excluded packet types, such as http/http bin, will be added later.

gen tcp api SUITE lwIP

1. `t_fdopen` creates a socket from a file desriptor. Ling does not have file descriptors (fails).

gen tcp echo SUITE lwIP

The source code of the test suite to exclude larger packet sizes (65531, 65535, and 70000). In lwIP the len parameter of various API calls has uint16 t type. Therefore, the maximum packet size tested is 65000.

re SUITE

The source code of the test suite was modified to exclude the checks for locale-dependent regular expressions. Such regular expressions are unsupported by Ling as there is no concept of locale. Another small modification deals with easing of a check for an error returned when compilation space overflows.