Referências Bibliográficas

[Atkins88] ATKINS, M. C.; NACKMAN, L. R.. **The active deallocation of objects in object-oriented systems**. Software: Practice and Experience, 18(11):1073–1089, 1988. 1

[Bloch01] BLOCH, J.. **Effective Java Programming Language Guide**. Prentice Hall, first edition, June 2001. 1

[Boehm91] BOEHM, H.-J.; DEMERS, A. J. ; SHENKER, S.. **Mostly parallel garbage collection**. In: PROCEEDINGS OF THE ACM SIGPLAN 1991 CONFERENCE ON PROGRAMMING LANGUAGE DESIGN AND IMPLEMENTATION, p. 157–164. ACM Press, June 1991. 2.1.1

[Boehm03] BOEHM, H.-J.. **Destructors, finalizers, and synchronization**. In: PROCEEDINGS OF THE ACM SYMPOSIUM ON PRINCIPLES OF PROGRAMMING LANGUAGES, p. 262–272. ACM Press, 2003. 1, 3.2, 3.2.2, 3.2.2

[Brownbridge85] BROWNBRIDGE, D. R.. **Cyclic reference counting for combinator machines**. In: PROCEEDINGS OF THE ACM CONFERENCE ON FUNCTIONAL PROGRAMMING LANGUAGES AND COMPUTER ARCHITECTURE, p. 273–288, New York, NY, USA, 1985. Springer Verlag. 2.1.3, 3.1, 3.1.1, 3.2.3

[CLS07] **C# language specification**. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csspec/html/CSharpSpecStart.asp, último acesso em 2 de abril de 2007. 3

[Cheney70] CHENEY, C. J.. **A nonrecursive list compacting algorithm**. Communications of the ACM, 13(11):677–678, November 1970. 2.1.1

[Christiansen03] CHRISTIANSEN, T.; TORKINGTON, N.. **Perl Cookbook**. O'Reilly, second edition, August 2003. 3.2, 3.2.3

[Cohen83] COHEN, J.; NICOLAU, A.. **Comparison of compacting algorithms for garbage collection**. ACM Transactions on Programming Languages and Systems, 5(4):532–553, October 1983. 2.1.1

[Collins60] COLLINS, G. E.. **A method for overlapping and erasure of lists**. Communications of the ACM, 2(12):655–657, December 1960. 2.1.1

[Deutsch76] DEUTSCH, L. P.; BOBROW, D. G.. **An efficient, incremental, automatic garbage collector**. Communications of the ACM, 19(9):522–526, September 1976. 2.1.1

[Dijkstra78] DIJKSTRA, E. W.; LAMPORT, L.; MARTIN, A. J.; SCHOLTEN, C. S. ; STEFFENS, E. F. M.. **On-the-fly garbage collection: An exercise in cooperation**. Communications of the ACM, 21(11):966–975, November 1978. 2.1.1, 2.1.1

[Dybvig93] DYBVIG, R. K.; BRUGGEMAN, C. ; EBY, D.. **Guardians in a generation-based garbage collector**. In: SIGPLAN CONFERENCE ON PROGRAMMING LANGUAGE DESIGN AND IMPLEMENTATION, p. 207–216, 1993. 1

[Fenichel69] FENICHEL, R. R.; YOCHELSON, J. C.. **A LISP garbage collector for virtual memory computer systems**. Communications of the ACM, 12(11):611–612, November 1969. 2.1.1

[GCS07] **Google code search**. http://www.google.com/codesearch, último acesso em 2 de abril de 2007. 3

[GHC07] TEAM, T. H.. **Hugs/ghc extension libraries: Weak**. http://www.dcs.gla.ac.uk/fp/software/ghc/lib/hg-libs-15.html, último acesso em 2 de abril de 2007. 4.3

[Gamma95] GAMMA, E.; HELM, R.; JOHNSON, R. ; VLISSIDES, J.. **Design Patterns: Elements of Reusable Objetct-Oriented Software**. Addison Wesley, 1995. 3.1.4

[Haskell07] **The Glasgow Haskell Compiler user's guide, version 6.2**. http://www.haskell.org/ghc, último acesso em 2 de abril de 2007. 1

[Hayes92] HAYES, B.. **Finalization in the collector interface**. In: IWMM '92: PROCEEDINGS OF THE INTERNATIONAL WORKSHOP ON MEMORY MANAGEMENT, p. 277–298, London, UK, 1992. Springer Verlag. 1, 2.1.2, 3.2.4

[Hayes97] HAYES, B.. **Ephemerons: a new finalization mechanism**. In: OOPSLA '97: PROCEEDINGS OF THE 12TH ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS, p. 176–183, New York, NY, USA, 1997. ACM Press. 1, 2.1.2, 2.1.3, 2.3.1, 3.1, 5, 5.1

[Homing93] HOMING, J.; KALSOW, B.; MCJONES, P. ; NELSON, G.. **Some useful modula-3 interfaces**. Technical Report 113, Digital Equipment Corporation, Systems Research Center, December 1993. 2.1.3

[Ierusalimschy06] IERUSALIMSCHY, R.. **Programming in Lua**. Lua.org, second edition, 2006. 1, 1.1, 3.1.2

[Jensen91] JENSEN, K.; WIRTH, N.; MICKEL, A. B. ; MINER, J. F.. **Pascal User Manual and Report: ISO Pascal Standard**. Springer Verlag, fourth edition, September 1991. 2.1

[Jones96] JONES, R.; LINS, R.. **Garbage Collection: Algorithms for Automatic Dynamic Memory Management**. Wiley, first edition, 1996. 1, 2, 2.1, 2.1.1

[Jones00] JONES, S. P.; MARLOW, S. ; ELLIOTT, C.. **Stretching the storage manager: Weak pointers and stable names in Haskell.** In: IMPLEMENTATION OF FUNCTIONAL LANGUAGES, 11TH INTERNATIONAL WORKSHOP, volumen 1868 de **Lecture Notes in Computer Science**, p. 37–58. Springer Verlag, 2000. 3.1.2, 4.3

[Kernighan88] KERNIGHAN, B. W.; RITCHIE, D. M.. **The C Programming Language**. Prentice Hall, second edition, 1988. 2.1

[Krugle07] **Krugle - code search for developers**. http://www.krugle.com, último acesso em 2 de abril de 2007. 3

[LML07] **Lua mailing list**. http://www.lua.org/lua-l.html, último acesso em 2 de abril de 2007. 3

[Leal05] LEAL, M. A.. **Finalizadores e Referências Fracas: Interagindo com o Coletor de Lixo**. PhD thesis, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, 2005. 1, 2.2, 3, 3.1, 3.2, 4.1

[Liu00] LIU, J. W. S.. **Real-Time Systems**, chapter 2, p. 26–33. Prentice Hall, first edition, April 2000. 2.1.1

[McCarthy60] MCCARTHY, J.. **Recursive functions of symbolic expression and their computation by machine, part I**. Communications of the ACM, 3(4):184–195, April 1960. 2.1.1

[Modula07] **Critical Mass Modula-3 5.1 documentation**. http://www.elegosoft.com/cm3/doc, último acesso em 2 de abril de 2007. 2.1.3, 4.1

[Muhammad06] MUHAMMAD, H. H.. **Estudo sobre APIs de linguagens de script**. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, August 2006. 3.2.1

[Rees84] REES, J. A.; ADAMS, N. I. ; MEEHAN, J. R.. **The T Manual**. Yale University Computer Science Department, fourth edition, January 1984. 1

[Richter02] RICHTER, J.. **Applied Microsoft .NET Framework Programming**. Microsoft Press, first edition, January 2002. 3

[Rossum06] VAN ROSSUM, G.. **Python library reference**, March 2006. Release 2.4.3. 2.1.3

[Rovner85] ROVNER, P.. **On adding garbage collection and runtime types to a strongly-typed, statically-checked, concurrent language**. Technical report CSL-84-7, Xerox PARC, Palo Alto, CA, 1985. 1

[SUN04] MICROSYSTEMS, S.. **Java 2 plataform standard edition 5.0: API specification**, 2004. 1, 3.2.1

[Schwartz81] SCHWARTZ, R.; MELLIAR-SMITH, P. M.. **The finalization operation for abstract types**. In: ICSE'81: 5TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, p. 273–282. IEEE Press, 1981. 3.2.4

[Schwartz05] SCHWARTZ, R.; PHOENIX, T. ; D'FOY, B.. **Learning Perl**. O'Reilly, fourth edition, July 2005. 3

[Sebesta02] SEBESTA, R. W.. **Concepts of Programming Languages**. Addison Wesley, fifth edition, 2002. 3.2.4

[Stroustrup97] STROUSTRUP, B.. **The C++ Programing Language**. Addison Wesley, third edition, June 1997. 2.1.2

[TCG07] **Tecgraf, computer graphics technology**. http://www.tecgraf.puc-rio.br/, último acesso em 2 de abril de 2007. 3

[Venners98] VENNERS, B.. **Object finalization and cleanup**. JavaWorld, June 1998. 3.2.4

[Wandler90] WANDLER, P.. **Comprehending monads**. In: 1990 ACM CONFERENCE ON LISP AND FUNCTIONAL PROGRAMMING, p. 61–78. ACM Press, 1990. 1

[Wandler95] WANDLER, P.. **Monads for functional programming**. Lecture Notes in Computer Science, 925:24–52, 1995. 1

[Wilson92] WILSON, P. R.. **Uniprocessor garbage collection techniques**. In: PROCEEDINGS OF THE 1992 INTERNATIONAL WORKSHOP ON MEMORY MANAGEMENT, volumen 637, p. 1–42, Saint-Malo (France), 1992. Springer Verlag. 2, 2.1, 2.1.1

[Wise77] WISE, D. S.; FRIEDMAN, D. P.. **The one-bit reference count**. BIT Numerical Mathematics, 17(3):351–359, September 1977. 2.1.1

[Xerox85] Xerox Palo Alto Research Center (PARC), Palo Alto, CA. **InterLISP Reference Manual**, October 1985. 1

A
Modificações Realizadas no Coletor de Lixo da Linguagem Lua

Neste apêndice, apresentamos as modificações feitas nas funções originais do coletor de lixo da linguagem Lua assim como as novas funções implementadas a fim de fornecer suporte ao mecanismo de ephemerons e à tabela de notificações. A seguir, listamos o código referente à implementação das mais relevantes novas funções para o mecanismo de ephemerons, `convergeephemerons` e `traverseephemerons`.

```
static void convergeephemerons(global_State *g, GCObject *l){
  while(traverseephemerons(g, l)) propagateall(g);
}

static int traverseephemerons(global_State *g, GCObject *l){
  int marked = 0;
  while(l){
    Table *h = gco2h(l);
    if(testbit(h->marked, EPHEMERONBIT)){
      int i = sizenode(h);
      while (i--) {
        Node *n = gnode(h, i);
        if (!ttisnil(gval(n)) && /* non-empty entry? */
            !iscleared(key2tval(n), 1) && iscleared(gval(n), 0)) {
          markvalue(g, gval(n));
          marked = 1;
        }
      }
    }
    l = h->gclist;
  }
  return marked;
}
```

A próxima função, `traversetable`, foi modificada a partir de sua implementação original. As linhas 11 a 19 testam se a string contida no campo `__mode` da metatabela contém o caractere "e", o que classifica a tabela como uma tabela de ephemerons. Em seguida, esse pedaço de código marca a tabela como sendo uma tabela de ephemerons e a insere na lista `weak`. O próximo pedaço de código referente à implementação de ephemerons está na linha 39. Essa linha testa se a tabela não é uma tabela de ephemerons, pois quando o é a parte hash não deve ser percorrida.

```
1:   static int traversetable (global_State *g, Table *h) {
2:     int i;
3:     int weakkey = 0;
4:     int weakvalue = 0;
5:     int isephemeron = 0;
6:     const TValue *mode;
7:     if (h->metatable)
8:       markobject(g, h->metatable);
9:     mode = gfasttm(g, h->metatable, TM_MODE);
10:    if (mode && ttisstring(mode)) {  /* is there a weak or ephemeron mode? */
11:      isephemeron = (strchr(svalue(mode), 'e') != NULL);
12:      weakkey = !isephemeron && (strchr(svalue(mode), 'k') != NULL);
13:      weakvalue = !isephemeron && (strchr(svalue(mode), 'v') != NULL);
14:      if (isephemeron) {
15:        h->marked &= ~EPHEMERON;
16:        h->marked |= cast_byte(isephemeron << EPHEMERONBIT);
17:        h->gclist = g->weak;
18:        g->weak = obj2gco(h);
19:      }
20:      else if (weakkey || weakvalue) {  /* is really weak? */
21:        h->marked &= ~(KEYWEAK | VALUEWEAK);  /* clear bits */
22:        h->marked |= cast_byte((weakkey << KEYWEAKBIT) |
23:                               (weakvalue << VALUEWEAKBIT));
24:        h->gclist = g->weak;  /* must be cleared after GC, ... */
25:        g->weak = obj2gco(h);  /* ... so put in the appropriate list */
26:      }
27:    }
28:    if (weakkey && weakvalue) return 1;
29:
30:    /* mark the array part, even if it's an ephemeron */
31:    if (!weakvalue) {
```

```
32:       i = h->sizearray;
33:       while (i--){
34:         markvalue(g, &h->array[i]);
35:       }
36:     }
37:
38:     /* only mark the hash part if it's not an ephemeron */
39:     if( !isephemeron) {
40:       i = sizenode(h);
41:       while (i--) {
42:         Node *n = gnode(h, i);
44:         lua_assert(ttype(gkey(n)) != LUA_TDEADKEY || ttisnil(gval(n)));
45:         if (ttisnil(gval(n)))
46:           removeentry(n);  /* remove empty entries */
47:         else {
48:           lua_assert(!ttisnil(gkey(n)));
49:           if (!weakkey) markvalue(g, gkey(n));
52:           if (!weakvalue) markvalue(g, gval(n));
55:         }
56:       }
57:     }
58:     return (weakkey || weakvalue) || isephemeron;
59:   }
```

A função `cleartable` mostrada a seguir foi modificada a fim de oferecer suporte à tabela de notificações. Como função auxiliar de `cleartable` implementamos a função `marknotify` cujo código é mostrado após `cleartable`. As linhas 6 a 8, testam se o campo `__notify` foi definido e atribuem a variável local a tabela de notificações. A seguir, nas linhas 19 a 24, antes de remover o valor da parte array da tabela, ele é copiado para a tabela de notificações. E por último, nas linhas 36 a 41, antes de remover o valor da parte hash da tabela, ele é copiado também para a tabela de notificações.

```
1:   static void cleartable (lua_State *L, GCObject *l) {
2:     while (l) {
3:       Table *h = gco2h(l);
4:
5:       global_State *g = G(L);
6:       const TValue *notify = gfasttm(g, h->metatable, TM_NOTIFY);
```

```
7:      Table *notifications = NULL;
8:      if(notify && ttistable(notify)) notifications = hvalue(notify);
9:
10:     lua_assert(testbit(h->marked, VALUEWEAKBIT) ||
11:                testbit(h->marked, KEYWEAKBIT) ||
12:                testbit(h->marked, EPHEMERONBIT));
13:
14:     int i = h->sizearray;
15:     if (testbit(h->marked, VALUEWEAKBIT)) {
16:       while (i--) {
17:         TValue *o = &h->array[i];
18:         if (iscleared(o, 0)){  /* value was collected? */
19:           if(notifications != NULL){
20:             /* mark notify if it's not already marked and insert the object
21:             in the array part */
22:             marknotify(g, notify);
23:             setobj2t(L, luaH_setnum(L, notifications, i+1), o);
24:           }
25:           setnilvalue(o);  /* remove value */
26:         }
27:       }
28:     }
29:     i = sizenode(h);
30:     while (i--) {
31:       Node *n = gnode(h, i);
33:       if (!ttisnil(gval(n)) &&  /* non-empty entry? */
34:           (iscleared(key2tval(n), 1) || iscleared(gval(n), 0))) {
35:
36:         if(notifications != NULL){
37:           /* mark notify if it's not already marked and insert the object
38:            in the hash part */
39:           marknotify(g, notify);
40:       setobj2t(L, luaH_set(L, notifications, key2tval(n)), gval(n));
41:     }
42:         setnilvalue(gval(n));  /* remove value ... */
43:         removeentry(n);  /* remove entry from table */
44:       }
45:     }
46:     l = h->gclist;
```

```
47:    }
48:  }

static void marknotify(global_State *g, const TValue *notify){
  GCObject *gcnotify = gcvalue(notify);
  if(!isgray(gcnotify)){
    makewhite(g, gcnotify);
    markvalue(g, notify);
  }
}
```