Appendix C
ABC machine specification

This appendix summarizes the machine instructions of the ABC machine. The most representative instructions (marked with •) are formally specified together with the used micro-instructions. The actual ABC machine (Koopman *et al.*,1990) has some facilities (such as directives and calling external functions) that are not included here.

C.1 The ABC instruction set

```
state = = (astack,bstack,cstack,graphstore,descrstore,pc,programstore,io)
instruction     = =   state -> state
```

a_dst	= =	nat	int	= =	num
a_src	= =	nat	instrid	= =	num
ap_entry	= =	instrid	name	= =	[char]
arg_nr	= =	nat	nat	= =	num
arity	= =	nat	nodeid	= =	nat
b_dst	= =	nat	nodeid_seq	= =	[nodeid]
b_src	= =	nat	nr_args	= =	nat
c_src	= =	nat	real	= =	num
descrid	= =	num	string	= =	[char]
graph	= =	[node]			

Instruction set

•	add_args	::	a_src -> nr_args -> a_dst -> instruction
•	create	::	instruction
•	del_args	::	a_src -> nr_args -> a_dst -> instruction
•	eq_descr	::	descrid -> a_src -> instruction
•	eq_descr_arity	::	descrid -> arity -> a_src -> instruction
•	eq_symbol	::	a_src -> a_src -> instruction
•	eqB	::	instruction
•	eqB_a	::	bool -> a_src -> instruction
•	eqB_b	::	bool -> b_src -> instruction
	eqC_a	::	char -> a_src -> instruction
	eqC_b	::	char -> b_src -> instruction

```
•  eqI               ::  instruction
•  eqI_a             ::  int -> a_src -> instruction
•  eqI_b             ::  int -> b_src -> instruction
   eqR_a             ::  real -> a_src -> instruction
   eqR_b             ::  real -> b_src -> instruction
   eqS_a             ::  string -> a_src -> instruction
•  fill              ::  descrid -> nr_args -> instrid -> a_dst -> instruction
•  fill_a            ::  a_src -> a_dst -> instruction
•  fillB             ::  bool -> a_dst -> instruction
•  fillB_b           ::  b_src -> a_dst -> instruction
   fillC             ::  char -> a_dst -> instruction
   fillC_b           ::  b_src -> a_dst -> instruction
•  fillI             ::  int -> a_dst -> instruction
•  fillI_b           ::  b_src -> a_dst -> instruction
   fillR             ::  real -> a_dst -> instruction
   fillR_b           ::  b_src -> a_dst -> instruction
   fillS             ::  string -> a_dst -> instruction
   fillS_symbol      ::  a_src -> a_dst -> instruction
•  get_descr_arity   ::  a_src -> instruction
•  get_node_arity    ::  a_src -> instruction
•  halt              ::  instruction
•  jmp               ::  instrid -> instruction
•  jmp_eval          ::  instruction
•  jmp_false         ::  instrid -> instruction
•  jmp_true          ::  instrid -> instruction
•  jsr               ::  instrid -> instruction
•  jsr_eval          ::  instruction
•  no_op             ::  instruction
•  pop_a             ::  nr_args -> instruction
•  pop_b             ::  nr_args -> instruction
•  print             ::  string -> instruction
•  print_symbol      ::  a_src -> instruction
•  push_a            ::  a_src -> instruction
•  push_ap_entry     ::  a_src -> instruction
•  push_arg          ::  a_src -> arity -> arg_nr -> instruction
•  push_arg_b        ::  a_src -> instruction
•  push_args         ::  a_src -> arity -> nr_args -> instruction
•  push_args_b       ::  a_src -> instruction
•  push_b            ::  b_src -> instruction
•  pushB             ::  bool -> instruction
•  pushB_a           ::  a_src -> instruction
   pushC             ::  char -> instruction
   pushC_a           ::  a_src -> instruction
•  pushI             ::  int -> instruction
•  pushI_a           ::  a_src -> instruction
   pushF_a           ::  a_src -> instruction
   pushR             ::  real -> instruction
   pushR_a           ::  a_src -> instruction
•  repl_args         ::  arity -> nr_args -> instruction
•  repl_args_b       ::  instruction
•  rtn               ::  instruction
•  set_entry         ::  instrid -> a_dst -> instruction
•  update_a          ::  a_src -> a_dst -> instruction
•  update_b          ::  b_src -> b_dst -> instruction

•  addI              ::  instruction
•  decI              ::  instruction
   divI              ::  instruction
•  gtI               ::  instruction
•  incI              ::  instruction
•  ltI               ::  instruction
•  mulI              ::  instruction
   modI              ::  instruction
```

```
• subI          :: instruction

  acosR         :: instruction
  addR          :: instruction
  asinR         :: instruction
  atabR         :: instruction
  cosR          :: instruction
  decR          :: instruction
  divR          :: instruction
  entierR       :: instruction
  expR          :: instruction
  incR          :: instruction
  gtR           :: instruction
  lnR           :: instruction
  log10R        :: instruction
  ltR           :: instruction
  mulR          :: instruction
  powR          :: instruction
  remR          :: instruction
  sinR          :: instruction
  sqrtR         :: instruction
  subR          :: instruction
  tanR          :: instruction

  catS          :: instruction
  cmpS          :: instruction
  indexS        :: instruction
  lenS          :: instruction
  sliceS        :: instruction
  updateS       :: instruction

  andBit        :: instruction
  notBit        :: instruction
  orBit         :: instruction
  xorBit        :: instruction
  shiftlBit     :: instruction
  shiftrBit     :: instruction

  andB          :: instruction
  notB          :: instruction
  orB           :: instruction

  gtC           :: instruction
  ltC           :: instruction

  convertCtoI   :: instruction
  convertItoC   :: instruction
  convertItoR   :: instruction
  convertRtoI   :: instruction

  endF     :: instruction
  getFC    :: instruction
  getFS    :: a_dst -> instruction
  openF    :: a_src->a_src->instruction
  putFC    :: instruction
  putFS    :: a_src -> instruction
  reopenF:: instruction
```

Specification of representative instructions

```
int_descr     = 0
bool_descr    = 1
rnf_entry     = 1

add_args a_src nr_args a_dst (as,bs,cs,gs,ds,pc,ps,io)
    = (as',bs,cs,gs',ds,pc,ps,io)
      where as'          = as_popn nr_args as
            gs'          = gs_update dstid (n_fill descrid entry newargs) gs
            dstid        = as_get a_dst as
            srcid        = as_get a_src as
            node         = gs_get srcid gs
            descrid      = n_descrid node
            entry        = n_entry node
            newargs      = n_args node arity ++ as_topn nr_args as
            arity        = n_arity node

create (as,bs,cs,gs,ds,pc,ps,io)
    = (as',bs,cs,gs',ds,pc,ps,io)
      where as'             = as_push nodeid as
            (gs',nodeid)    = gs_newnode gs

del_args a_src nr_args a_dst (as,bs,cs,gs,ds,pc,ps,io)
    = (as,bs,cs,gs',ds,pc,ps,io)
      where gs'          = gs_update dstid (n_fill descrid entry newargs) gs
            dstid        = as_get a_dst as
            srcid        = as_get a_src as
            node         = gs_get srcid gs
            descrid      = n_descrid node
            entry        = n_entry node
```

```
                newargs     =  n_nargs node (arity – nr_args) arity
                arity       =  n_arity node

eq_descr descrid a_src (as,bs,cs,gs,ds,pc,ps,io)
   =  (as,bs',cs,gs,ds,pc,ps,io)
      where  bs'          =  bs_pushB equal bs
             equal        =  n_eq_descrid node descrid
             node         =  gs_get nodeid gs
             nodeid       =  as_get a_src as

eq_descr_arity descrid arity a_src (as,bs,cs,gs,ds,pc,ps,io)
   =  (as,bs',cs,gs,ds,pc,ps,io)
      where  bs'          =  bs_pushB equal bs
             equal        =  n_eq_descrid node descrid & n_eq_arity node arity
             node         =  gs_get nodeid gs
             nodeid       =  as_get a_src as

eq_symbol a_src1 a_src2 (as,bs,cs,gs,ds,pc,ps,io)
   =  (as,bs',cs,gs,ds,pc,ps,io)
      where  bs'          =  bs_pushB equal bs
             equal        =  n_eq_symbol node1 node2
             node1        =  gs_get nodeid1 gs
             node2        =  gs_get nodeid2 gs
             nodeid1      =  as_get a_src1 as
             nodeid2      =  as_get a_src2 as

eqB (as,bs,cs,gs,ds,pc,ps,io)
   =  (as,bs',cs,gs,ds,pc,ps,io)
      where  bs'          =  bs_eqB bs

eqB_a bool a_src (as,bs,cs,gs,ds,pc,ps,io)
   =  (as,bs',cs,gs,ds,pc,ps,io)
      where  bs'          =  bs_pushB equal bs
             equal        =  n_eq_B (gs_get nodeid gs) bool
             nodeid       =  as_get a_src as

eqB_b bool b_src (as,bs,cs,gs,ds,pc,ps,io)
   =  (as,bs',cs,gs,ds,pc,ps,io)
      where  bs'          =  bs_eqBi bool b_src bs

eqI (as,bs,cs,gs,ds,pc,ps,io)
   =  (as,bs',cs,gs,ds,pc,ps,io)
      where  bs'          =  bs_eqI bs

eqI_a int a_src (as,bs,cs,gs,ds,pc,ps,io)
   =  (as,bs',cs,gs,ds,pc,ps,io)
      where  bs'          =  bs_pushB equal bs
             equal        =  n_eq_I (gs_get nodeid gs) int
             nodeid       =  as_get a_src as

eqI_b int b_src (as,bs,cs,gs,ds,pc,ps,io)
   =  (as,bs',cs,gs,ds,pc,ps,io)
      where  bs'          =  bs_eqIi int b_src bs

fill descr nr_args entry a_dst (as,bs,cs,gs,ds,pc,ps,io)
   =  (as',bs,cs,gs',ds,pc,ps,io)
      where  as'          =  as_popn nr_args as
             gs'          =  gs_update nodeid (n_fill descr entry args) gs
             nodeid       =  as_get a_dst as
             args         =  as_topn nr_args as

fill_a a_src a_dst (as,bs,cs,gs,ds,pc,ps,io)
   =  (as,bs,cs,gs',ds,pc,ps,io)
```

```
        where   gs'            =  gs_update nodeid_dst (n_copy node_src) gs
                node_src       =  gs_get nodeid_src gs
                nodeid_dst     =  as_get a_dst as
                nodeid_src     =  as_get a_src as

fillB bool a_dst (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs,cs,gs',ds,pc,ps,io)
        where   gs'       =  gs_update nodeid (n_fillB bool_descr rnf_entry bool) gs
                nodeid    =  as_get a_dst as

fillB_b b_src a_dst (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs,cs,gs',ds,pc,ps,io)
        where   gs'       =  gs_update nodeid (n_fillB bool_descr rnf_entry bool) gs
                bool      =  bs_getB b_src bs
                nodeid    =  as_get a_dst as

fillI int a_dst (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs,cs,gs',ds,pc,ps,io)
        where   gs'       =  gs_update nodeid (n_fillI int_descr rnf_entry int) gs
                nodeid    =  as_get a_dst as

fillI_b b_src a_dst (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs,cs,gs',ds,pc,ps,io)
        where   gs'       =  gs_update nodeid (n_fillI int_descr rnf_entry int) gs
                int       =  bs_getI b_src bs
                nodeid    =  as_get a_dst as

get_descr_arity a_src (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs',cs,gs,ds,pc,ps,io)
        where   bs'       =  bs_pushI arity bs
                arity     =  d_arity (ds_get descrid ds)
                descrid   =  n_descrid (gs_get nodeid gs)
                nodeid    =  as_get a_src as

get_node_arity a_src (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs',cs,gs,ds,pc,ps,io)
        where   bs'       =  bs_pushI arity bs
                arity     =  n_arity (gs_get nodeid gs)
                nodeid    =  as_get a_src as

halt (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs,cs,gs,ds,pc',ps,io)
        where   pc'       =  pc_halt pc

jmp address (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs,cs,gs,ds,pc',ps,io)
        where   pc'       =  pc_update address pc

jmp_eval (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs,cs,gs,ds,pc',ps,io)
        where   pc'       =  pc_update (n_entry (gs_get nodeid gs)) pc
                nodeid    =  as_get 0 as

jmp_false address (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs',cs,gs,ds,pc',ps,io)
        where   pc'       =  pc, if bool
                          =  pc_update address pc, otherwise
                bool      =  bs_getB 0 bs
                bs'       =  bs_popn 1 bs

jmp_true address (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs',cs,gs,ds,pc',ps,io)
        where   pc'       =  pc_update address pc, if bool
```

```
                    =  pc, otherwise
         bool       =  bs_getB 0 bs
         bs'        =  bs_popn 1 bs

jsr address (as,bs,cs,gs,ds,pc,ps,io)
  =  (as,bs,cs',gs,ds,pc',ps,io)
     where  pc'     =  pc_update address pc
            cs'     =  cs_push (pc_get pc) cs

jsr_eval (as,bs,cs,gs,ds,pc,ps,io)
  =  (as,bs,cs',gs,ds,pc',ps,io)
     where  pc'     =  pc_update (n_entry (gs_get nodeid gs)) pc
            nodeid  =  as_get 0 as
            cs'     =  cs_push (pc_get pc) cs

no_op state  =  state

pop_a n (as,bs,cs,gs,ds,pc,ps,io)
  =  (as',bs,cs,gs,ds,pc,ps,io)
     where  as'     =  as_popn n as

pop_b n (as,bs,cs,gs,ds,pc,ps,io)
  =  (as,bs',cs,gs,ds,pc,ps,io)
     where  bs'     =  bs_popn n bs

print string (as,bs,cs,gs,ds,pc,ps,io)
  =  (as,bs,cs,gs,ds,pc,ps,io')
     where  io'     =  io_print string io

print_symbol a_src (as,bs,cs,gs,ds,pc,ps,io)
  =  (as,bs,cs,gs,ds,pc,ps,io')
     where  io'     =  io_print_symbol node descr io
            nodeid  =  as_get a_src as
            node    =  gs_get nodeid gs
            descr   =  ds_get (n_descrid node) ds

push_a a_src (as,bs,cs,gs,ds,pc,ps,io)
  =  (as',bs,cs,gs,ds,pc,ps,io)
     where  as'     =  as_push nodeid as
            nodeid  =  as_get a_src as

push_ap_entry a_src (as,bs,cs,gs,ds,pc,ps,io)
  =  (as,bs,cs',gs,ds,pc,ps,io)
     where  cs'     =  cs_push (d_ap_entry (ds_get descrid ds)) cs
            descrid =  n_descrid (gs_get nodeid gs)
            nodeid  =  as_get a_src as

push_arg a_src arity arg_nr (as,bs,cs,gs,ds,pc,ps,io)
  =  (as',bs,cs,gs,ds,pc,ps,io)
     where  as'     =  as_push arg as
            arg     =  n_arg (gs_get nodeid gs) arg_nr arity
            nodeid  =  as_get a_src as

push_arg_b a_src (as,bs,cs,gs,ds,pc,ps,io)
  =  (as',bs,cs,gs,ds,pc,ps,io)
     where  as'     =  as_push arg as
            arg     =  n_arg (gs_get nodeid gs) arg_nr arity
            nodeid  =  as_get a_src as
            arg_nr  =  bs_getI 0 bs
            arity   =  bs_getI 1 bs

push_args a_src arity nr_args (as,bs,cs,gs,ds,pc,ps,io)
  =  (as',bs,cs,gs,ds,pc,ps,io)
```

```
        where   as'     =  as_pushn args as
                args    =  n_nargs (gs_get nodeid gs) nr_args arity
                nodeid  =  as_get a_src as

push_args_b a_src (as,bs,cs,gs,ds,pc,ps,io)
    =  (as',bs,cs,gs,ds,pc,ps,io)
        where   as'     =  as_pushn args as
                args    =  n_nargs (gs_get nodeid gs) nargs arity
                nargs   =  bs_getI 0 bs
                nodeid  =  as_get a_src as
                arity   =  bs_getI 1 bs

push_b b_src (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs',cs,gs,ds,pc,ps,io)
        where   bs'     =  bs_push basic bs
                basic   =  bs_get b_src bs

pushB bool (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs',cs,gs,ds,pc,ps,io)
        where   bs'     =  bs_pushB bool bs

pushB_a a_src (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs',cs,gs,ds,pc,ps,io)
        where   bs'     =  bs_pushB bool bs
                bool    =  n_B (gs_get nodeid gs)
                nodeid  =  as_get a_src as

pushI int (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs',cs,gs,ds,pc,ps,io)
        where   bs'     =  bs_pushI int bs

pushI_a a_src (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs',cs,gs,ds,pc,ps,io)
        where   bs'     =  bs_pushI int bs
                int     =  n_I (gs_get nodeid gs)
                nodeid  =  as_get a_src as

repl_args arity nr_args (as,bs,cs,gs,ds,pc,ps,io)
    =  (as',bs,cs,gs,ds,pc,ps,io)
        where   as'     =  as_pushn args (as_popn 1 as)
                args    =  n_nargs (gs_get nodeid gs) nr_args arity
                nodeid  =  as_get 0 as

repl_args_b (as,bs,cs,gs,ds,pc,ps,io)
    =  (as',bs,cs,gs,ds,pc,ps,io)
        where   as'     =  as_pushn args (as_popn 1 as)
                args    =  n_nargs (gs_get nodeid gs) nr_args arity
                nodeid  =  as_get 0 as
                arity   =  bs_getI 0 bs
                nr_args =  bs_getI 1 bs

rtn (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs,cs',gs,ds,pc',ps,io)
        where   pc'     =  pc_update (cs_get 0 cs) pc
                cs'     =  cs_popn 1 cs

set_entry entry a_dst (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs,cs,gs',ds,pc,ps,io)
        where   gs'     =  gs_update nodeid (n_setentry entry) gs
                nodeid  =  as_get a_dst as

update_a a_src a_dst (as,bs,cs,gs,ds,pc,ps,io)
    =  (as',bs,cs,gs,ds,pc,ps,io)
```

```
        where   as'      =  as_update a_dst nodeid as
                nodeid   =  as_get a_src as

update_b b_src b_dst (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs',cs,gs,ds,pc,ps,io)
       where   bs'      =  bs_update b_dst basic bs
               basic    =  bs_get b_src bs

addI  (as,bs,cs,gs,ds,pc,ps,io)  = (as,bs',cs,gs,ds,pc,ps,io) where bs' = bs_addI bs
decI  (as,bs,cs,gs,ds,pc,ps,io)  = (as,bs',cs,gs,ds,pc,ps,io) where bs' = bs_decI bs
gtI   (as,bs,cs,gs,ds,pc,ps,io)  = (as,bs',cs,gs,ds,pc,ps,io) where bs' = bs_gtI bs
incI  (as,bs,cs,gs,ds,pc,ps,io)  = (as,bs',cs,gs,ds,pc,ps,io) where bs' = bs_incI bs
ltI   (as,bs,cs,gs,ds,pc,ps,io)  = (as,bs',cs,gs,ds,pc,ps,io) where bs' = bs_ltI bs
mulI  (as,bs,cs,gs,ds,pc,ps,io)  = (as,bs',cs,gs,ds,pc,ps,io) where bs' = bs_mulI bs
subI  (as,bs,cs,gs,ds,pc,ps,io)  = (as,bs',cs,gs,ds,pc,ps,io) where bs' = bs_subI bs
```

C.2 Running ABC programs

Booting and instruction fetching

```
boot:: ([instruction],[descr]) -> state
boot (program,descriptors)
    =  (as,bs,cs,gs,ds,pc,ps,io)
       where   pc   =  pc_init
               as   =  as_init
               bs   =  bs_init
               cs   =  cs_init
               gs   =  gs_init
               ps   =  ps_init program
               io   =  io_init
               ds   =  ds_init descriptors

fetch_cycle:: state -> state
fetch_cycle (as,bs,cs,gs,ds,pc,ps,io)
    =  (as,bs,cs,gs,ds,pc,ps,io)                          , if pc_end pc
    =  fetch_cycle (currinstr (as,bs,cs,gs,ds,pc',ps,io))  , otherwise
       where   pc'         =  pc_next pc
               currinstr   =  ps_get (pc_get pc) ps
```

ABC assembly language

```
assembler:: assembly -> ([instruction], [descr])
assembler asm_prog
    =  (translate asm_prog loc_count sym_table, descr_table asm_prog sym_table)
       where   loc_count     =  0
               descr_count   =  0
               sym_table     =  collect asm_prog loc_count descr_count
```

The specification of translate, descr_table and collect that translate the ABC assembly program and produce its ABC instructions with its description table would require many (rather trivial) lines. For the full specification the reader is referred to Koopman *et al.* (1990).

The ABC assembly language is defined by the algebraic data structure given below. Most assembly statements are equal to their ABC instruction counterpart, but their names begin with an upper case character. The statements in *italic* are slightly different because symbolic names are used instead of addresses. In addition, the assembler allows the definition of labels and descriptors. Furthermore, Branch instructions are added to allow the specification of relative jumps. The assembler will convert them to jump instructions. These extra assembly statements are also shown in *italic*.

```
assembly      ==  [statement]
red_label     ==  label
descr_label   ==  label
label         ==  [char]
nr_instr      ==  int

statement
::= Label        label                              |  No_op                          |
    Descriptor descr_label red_label arity name     |  Pop_a         nr_args          |
    Br           nr_instr                           |  Pop_b         nr_args          |
    Br_false     nr_instr                           |  Print         string           |
    Br_true      nr_instr                           |  Print_symbol  a_src            |
    Add_args     a_src nr_args a_dst                |  Push_a        a_src            |
    Create                                          |  Push_ap_entry   a_src          |
    Del_args     a_src nr_args a_dst                |  Push_arg   a_src arity arg_nr  |
    Eq_descr     descr_label a_src                  |  Push_arg_b    a_src            |
    Eq_descr_arity descr_label arity a_src          |  Push_args a_src arity arg_nr   |
    EqB                                             |  Push_args_b   a_src            |
    EqB_a        bool a_src                         |  Push_b        b_src            |
    EqB_b        bool b_src                         |  PushB         bool             |
    EqI                                             |  PushB_a       a_src            |
    EqI_a        int a_src                          |  PushI         int              |
    EqI_b        int b_src                          |  PushI_a       a_src            |
    Fill      descr_label nr_args label a_dst       |  Repl_args     arity nr_args    |
    Fill_a       a_src a_dst                        |  Repl_args_b                    |
    FillB        bool a_dst                         |  Rtn                            |
    FillB_b      b_src a_dst                        |  Set_entry     label a_dst      |
    FillI        int a_dst                          |  Update_a      a_src a_dst      |
    FillI_b      b_src a_dst                        |  Update_b      b_src b_dst      |
    Get_descr_arity a_src                           |
    Get_node_arity  a_src                           |
    Halt                                            |  AddI                           |
    Jmp          label                              |  DecI                           |
    Jmp_eval                                        |  GtI                            |
    Jmp_false    label                              |  IncI                           |
    Jmp_true     label                              |  LtI                            |
    Jsr          label                              |  MulI                           |
    Jsr_eval                                        |  SubI
```

C.3 The ABC micro-instructions

```
abstype   astack, bstack, cstack, graphstore, descrstore, pc, programstore, io
  with

  ps_get        ::  instrid -> programstore -> instruction
  ps_init       ::  [instruction] -> programstore

  pc_end        ::  pc -> bool
  pc_get        ::  pc -> instrid
  pc_halt       ::  pc -> pc
  pc_init       ::  pc
  pc_next       ::  pc -> pc
  pc_update     ::  instrid -> pc -> pc
```

```
gs_get          :: nodeid -> graphstore -> node
gs_init         :: graphstore
gs_newnode      :: graphstore -> (graphstore,nodeid)
gs_update       :: nodeid -> (node -> node) -> graphstore -> graphstore

n_arg           :: node -> arg_nr -> arity -> nodeid
n_args          :: node -> arity -> nodeid_seq
n_arity         :: node -> arity
n_B             :: node -> bool
n_copy          :: node -> node -> node
n_descrid       :: node -> descrid
n_entry         :: node -> instrid
n_eq_arity      :: node -> arity -> bool
n_eq_B          :: node -> bool -> bool
n_eq_descrid    :: node -> descrid -> bool
n_eq_I          :: node -> int -> bool
n_eq_symbol     :: node -> node -> bool
n_fill          :: descrid -> instrid -> nodeid_seq -> node -> node
n_fillB         :: descrid -> instrid -> bool -> node -> node
n_fillI         :: descrid -> instrid -> int -> node -> node
n_I             :: node -> int
n_nargs         :: node -> nr_args -> arity -> nodeid_seq
n_setentry      :: instrid -> node -> node

ds_get          :: descrid -> descrstore -> descr
ds_init         :: [descr] -> descrstore

d_ap_entry      :: descr -> instrid
d_arity         :: descr -> arity
d_name          :: descr -> string

as_get          :: a_src -> astack -> nodeid
as_init         :: astack
as_popn         :: nr_args -> astack -> astack
as_push         :: nodeid -> astack -> astack
as_pushn        :: nodeid_seq -> astack -> astack
as_topn         :: nr_args -> astack -> nodeid_seq
as_update       :: a_dst -> nodeid -> astack -> astack

bs_get          :: b_src -> bstack -> basic
bs_getB         :: b_src -> bstack -> bool
bs_getI         :: b_src -> bstack -> int
bs_init         :: bstack
bs_popn         :: nr_args -> bstack -> bstack
bs_push         :: basic -> bstack -> bstack
bs_pushB        :: bool -> bstack -> bstack
bs_pushI        :: int -> bstack -> bstack
bs_update       :: b_dst -> basic -> bstack -> bstack

bs_addI         :: bstack -> bstack
bs_decI         :: bstack -> bstack
bs_eqB          :: bstack -> bstack
bs_eqI          :: bstack -> bstack
bs_eqBi         :: bool -> b_src -> bstack -> bstack
bs_eqIi         :: int -> b_src -> bstack -> bstack
bs_gtI          :: bstack -> bstack
bs_incI         :: bstack -> bstack
bs_ltI          :: bstack -> bstack
bs_mulI         :: bstack -> bstack
bs_subI         :: bstack -> bstack

cs_init         :: cstack
cs_get          :: c_src -> cstack -> instrid
```

```
cs_popn           :: nr_args -> cstack -> cstack
cs_push           :: instrid -> cstack -> cstack

io_init           :: io
io_print          :: string -> io -> io
io_print_symbol   :: node -> descr -> io -> io
```

Program store

```
programstore = =  [location]
location     :: = I instruction

ps_get target ps             =  instr        where   I instr   =  ps ! target

ps_init (instruction : rest) =  I instruction : ps_init rest
ps_init [ ]                  =  [ ]
```

Program counter

```
pc       = =  instrid

pc_end instrid          =  instrid < 0
pc_get instrid          =  instrid
pc_halt instrid         =  –1
pc_init                 =  0
pc_next instrid         =  instrid + 1
pc_update instrid pc    =  instrid
```

Graph store

```
graphstore    = =   (graph, nat)

gs_get nodeid (nds,free)  =  nds ! (nodeid – free – 1)
gs_init                   =  ([ ], store_size)

store_size:: nat
store_size  =  100            || some natural number indicating the size

gs_newnode (nds,0)        =  error "graph-store is full"
gs_newnode (nds,free)     =  ((Empty : nds, free – 1), free)

gs_update nid f (nodes,free) =  (upd (nid – free –1) nodes f, free)
                                where   upd 0 (nd : nds) f = f nd : nds
                                        upd n (nd : nds) f = nd : upd (n – 1) nds f
```

Nodes

```
node :: =   Node descrid instrid nodeid_seq | Basic descrid instrid basic | Empty
basic :: =  Int int    |   Bool bool

n_arg node n arity  =  args ! (n – 1)  where   args  =  n_args node arity
n_args (Node descrid entry args) arity = args  ||  arity can be used for optimizations

n_arity (Basic descrid entry basic)    =  0
n_arity (Node descrid entry args)      =  # args

n_B (Basic descrid entry (Bool b))     =  b
n_copy new old                         =  new
```

```
n_descrid (Node descrid entry args)      =  descrid
n_descrid (Basic descrid entry basic)    =  descrid

n_entry (Node descrid entry args)      =  entry
n_entry (Basic descrid entry basic)    =  entry

n_eq_arity  node n                     =  n_arity node = n
n_eq_B      node b                     =  n_B node = b
n_eq_descrid node descrid              =  n_descrid node = descrid
n_eq_I      node i                     =  n_I node = i
n_eq_symbol (Node descrid1 entry1 args1) (Node descrid2 entry2 args2)
                                       =  descrid1 = descrid2
n_eq_symbol (Basic descrid1 entry1 basic1) (Basic descrid2 entry2 basic2)
                                       =  descrid1 = descrid2 & basic1 = basic2
n_eq_symbol node1 node2                =  False
n_fill descr entry args node           =  Node descr entry args
n_fillB descr entry b node             =  Basic descr entry (Bool b)
n_fillI descr entry i node             =  Basic descr entry (Int i)
n_I (Basic descrid entry (Int i))      =  i
n_nargs node arg_count arity           =  take arg_count (n_args node arity)

n_setentry newentry (Node  descrid entry args)  =  Node descrid newentry args
n_setentry newentry (Basic descrid entry basic) =  Basic descrid newentry basic
```

Descriptor store

```
descrstore    = =    [descr]

ds_get target ds     =  ds ! target
ds_init descriptors  =  descriptors
```

Descriptors

```
descr    :: =    Descr ap_entry arity name

d_ap_entry    (Descr ap_entry arity name)    =  ap_entry
d_arity       (Descr ap_entry arity name)    =  arity
d_name        (Descr ap_entry arity name)    =  name
```

A-stack

```
astack   = =    [nodeid]

as_get target as          =  as ! target
as_init                   =  [ ]
as_popn n as              =  drop n as
as_push nodeid as         =  nodeid : as
as_pushn nodeids as       =  nodeids ++ as
as_topn n as              =  take n as

as_update 0 nodeid (a : x)    =  nodeid : x
as_update n nodeid (a : x)    =  a : as_update (n – 1) nodeid x
```

B-stack

```
bstack   = =    [basic]

bs_get target bs     =  bs ! target
bs_getB n bs         =  b      where    Bool b   =  bs_get n bs
```

```
bs_getI n bs          =  i      where   Int i      =  bs_get n bs
bs_init               =  [ ]
bs_popn n bs          =  drop n bs
bs_push basic bs      =  basic : bs
bs_pushB b bs         =  Bool b : bs
bs_pushI i bs         =  Int i : bs

bs_update 0 basic (b : x)  =  basic : x
bs_update n basic (b : x)  =  b : bs_update (n – 1) basic x

bs_addI    (Int i1 : Int i2 : r)      =  Int (i1 + i2) : r
bs_decI    (Int i : r)                =  Int (i – 1) : r
bs_eqB     (Bool b1 : Bool b2 : r)    =  Bool (b1 = b2) : r
bs_eqBi    b         n        bs      =  bs_push (Bool (b = bs_getB n bs)) bs
bs_eqI     (Int i1 : Int i2 : r)      =  Bool (i1 = i2) : r
bs_eqIi    i         n        bs      =  bs_push (Bool (i = bs_getI n bs)) bs
bs_gtI     (Int i1 : Int i2 : r)      =  Bool (i1 > i2) : r
bs_incI    (Int i : r)                =  Int (i + 1) : r
bs_ltI     (Int i1 : Int i2 : r)      =  Bool (i1 < i2) : r
bs_mulI    (Int i1 : Int i2 : r)      =  Int (i1 * i2) : r
bs_subI    (Int i1 : Int i2 : r)      =  Int (i1 – i2) : r
```

C-stack

```
cstack   = =   [instrid]

cs_init              =  [ ]
cs_get target cs     =  cs ! target
cs_popn n cs         =  drop n cs
cs_push c cs         =  c : cs
```

I/O channel

```
io  = =   [char]

io_init                    =  [ ]
io_print string output     =  output ++ string

io_print_symbol (Basic descrid ap_entry (Int i)) descr output      =  output ++ show i
io_print_symbol (Basic descrid ap_entry (Bool b)) descr output =  output ++ show b
io_print_symbol (Node descrid entry args) (Descr ap_entry arity name) output
                                                                   =  output ++ name
```