Appendix D
PABC machine specification

The PABC machine is a parallel extension of the ABC machine. This extension has some consequences for the sequential ABC instructions presented in Appendix C. For most instructions only the machine state has to be extended. Together with the extension of the assembly language for the new instructions this is left to the reader.

For some sequential instructions a call to the operating system has to be included. This is fully specified together with the parallel instructions that are marked ("•") in the summary below. The instructions newP, randomP, currentP, neighbourP and channelP require some extensions of the specification (see Exercise 16.2).

D.1 The PABC instruction set

Sequential and parallel reducer instructions

```
instruction   ==  state -> state
state         ==  (local_state,global_state,opsys_callqueue)
local_state   ==  (astack,bstack,cstack,nqueue,pc,wlist)
global_state  ==  (admin,graphstore,descrstore,programstore)

•  halt              ::  instruction
•  jmp               ::  instrid -> instruction
•  jsr               ::  instrid -> instruction
•  print             ::  string -> instruction
•  print_symbol      ::  a_src -> instruction

   channelP          ::  a_src -> instruction
•  create_channel    ::  instruction
   currentP          ::  instruction
•  get_node          ::  instruction
•  getWL             ::  a_src -> instruction
•  is_empty_n        ::  instruction
   neighbourP        ::  instruction
```

```
• new_ext_reducer      :: instrid -> a_src -> instruction
• new_int_reducer      :: instrid -> a_src -> instruction
• newP                 :: instruction
  randomP              :: instruction
• release              :: instruction
• send_graph           :: a_src -> a_src -> instruction
• send_request         :: a_src -> instruction
• set_continue         :: a_src -> instruction
• set_wait             :: a_src -> instruction
• stop_reducer         :: instruction
• suspend              :: instruction
```

Operating system instructions

```
opsys_instruction   = =   opsys_state -> opsys_state
opsys_state = = (schedule_adm,in_msgs,out_msgs,global_state,opsys_callqueue)

• os_halt              :: opsys_instruction
• os_new_reducer       :: instrid -> nodeid -> opsys_instruction
• os_print             :: string -> opsys_instruction
• os_reducerprio       :: opsys_instruction
• os_release           :: wlist -> opsys_instruction
• os_schedule          :: opsys_instruction
• os_send_graph        :: nodeid -> nodeid -> opsys_instruction
• os_send_newchannel   :: procid -> nodeid -> opsys_instruction
• os_send_newreducer   :: instrid -> nodeid -> opsys_instruction
• os_send_request      :: nodeid -> opsys_instruction
• os_setwait           :: nodeid -> opsys_instruction
• os_stop_reducer      :: opsys_instruction
• os_suspend_reducer   :: opsys_instruction
```

Communication process instructions

```
comproc_instruction   = =   comproc_state -> comproc_state
comproc_state         = =   (in_msgs,out_msgs,global_state,opsys_callqueue)

• cp_accept_msg     ::  msg_contents -> comproc_instruction
```

Specification of sequential reducer instructions

```
halt ((as,bs,cs,nq,pc,wl),glob,cq)
   =  ((as,bs,cs,nq,pc,wl),glob,cq')
      where   cq'   =  os_call cq [os_stop_reducer, os_halt]

jmp address ((as,bs,cs,nq,pc,wl),glob,cq)
   =  ((as,bs,cs,nq,pc',wl),glob,cq')
      where   pc'   =  pc_update address pc
              cq'   =  os_call cq [os_schedule]

jsr address ((as,bs,cs,nq,pc,wl),glob,cq)
   =  ((as,bs,cs',nq,pc',wl),glob,cq')
      where   pc'   =  pc_update address pc
              cs'   =  cs_push (pc_get pc) cs
              cq'   =  os_call cq [os_schedule]

print string (loc,glob,cq)
   =  (loc,glob,cq')
      where   cq'   =  os_call cq [os_print string, os_schedule]

print_symbol a_src ((as,bs,cs,nq,pc,wl),(ad,gs,ds,ps),cq)
```

```
    = ((as,bs,cs,nq,pc,wl),(ad,gs,ds,ps),cq')
      where  cq'      =  os_call cq [os_print string, os_schedule]
             nodeid   =  as_get a_src as
             node     =  gs_get nodeid gs
             string   =  ds_get_repr node ds
```

Specification of parallel reducer instructions

```
create_channel ((as,bs,cs,nq,pc,wl),(ad,gs,ds,ps),cq)
    = ((as',bs',cs,nq,pc,wl),(ad,gs',ds,ps),cq')
      where  bs'         =  bs_popn 1 bs
             pid         =  bs_getP 0 bs
             (gs',nid)   =  gs_newchannel pid gs
             as'         =  as_push nid as
             cq'         =  os_call cq [os_send_newchannel pid nid,
                                        os_setwait nid, os_suspend_reducer]

get_node ((as,bs,cs,nq,pc,wl),glob,cq)
    = ((as',bs,cs,nq',pc,wl),glob,cq)
      where  as'  =  as_push n as
             n    =  nq_first nq
             nq'  =  nq_remove nq

getWL a_depth ((as,bs,cs,nq,pc,wl),(ad,gs,ds,ps),cq)
    = ((as,bs,cs,nq,pc,wl'),(ad,gs,ds,ps),cq)
      where  wl'  =  n_wl_get (gs_get nid gs)
             nid  =  as_get a_depth as

is_empty_n ((as,bs,cs,nq,pc,wl),glob,cq)
    = ((as,bs',cs,nq,pc,wl),glob,cq)
      where  bs'  =  bs_pushB (nq_empty nq) bs

new_ext_reducer code chan_depth ((as,bs,cs,nq,pc,wl),glob,cq)
    = ((as,bs,cs,nq,pc,wl),glob,cq')
      where  chanid  =  as_get chan_depth as
             cq'     =  os_call cq [os_send_newreducer code chanid]

new_int_reducer code a_depth ((as,bs,cs,nq,pc,wl),glob,cq)
    = ((as,bs,cs,nq,pc,wl),glob,cq')
      where  nid  =  as_get a_depth as
             cq'  =  os_call cq [os_new_reducer code nid]

newP ((as,bs,cs,nq,pc,wl),(ad,gs,ds,ps),cq)
    = ((as,bs',cs,nq,pc,wl),(ad',gs,ds,ps),cq)
      where  bs'        =  bs_pushP pid bs
             (ad',pid)  =  ad_new_procid ad

release ((as,bs,cs,nq,pc,wl),glob,cq)
    = ((as,bs,cs,nq,pc,wl),glob,cq')
      where  cq'  =  os_call cq [os_release wl]    , if ~ wl_empty wl
                  =  cq                            , otherwise

send_graph graph_depth chan_depth ((as,bs,cs,nq,pc,wl),glob,cq)
    = ((as,bs,cs,nq,pc,wl),glob,cq')
      where  chanid   =  as_get chan_depth as
             graphid  =  as_get graph_depth as
             cq'      =  os_call cq [os_send_graph graphid chanid]

send_request chan_depth ((as,bs,cs,nq,pc,wl),glob,cq)
    = ((as,bs,cs,nq,pc,wl),glob,cq')
      where  cq'     =  os_call cq [os_send_request chanid,os_reducerprio]
             chanid  =  as_get chan_depth as
```

```
set_continue depth ((as,bs,cs,nq,pc,wl),glob,cq)
    =  ((as,bs,cs,nq',pc,wl),glob,cq)
       where   n      =  as_get depth as
               nq'    =  nq_add n nq

set_wait a_depth ((as,bs,cs,nq,pc,wl),glob,cq)
    =  ((as,bs,cs,nq,pc,wl),glob,cq')
       where   nid    =  as_get a_depth as
               cq'    =  os_call cq [os_setwait nid,os_reducerprio]

stop_reducer ((as,bs,cs,nq,pc,wl),glob,cq)
    =  ((as,bs,cs,nq,pc,wl),glob,cq')
       where   cq'    =  os_call cq [os_stop_reducer]

suspend ((as,bs,cs,nq,pc,wl),glob,cq)
    =  ((as,bs,cs,nq,pc,wl),glob,cq')
       where   cq'    =  os_call cq [os_suspend_reducer]
```

Specification of operating system instructions

```
schedule_adm     ==   ([active], [passive])
active           ==   reducer
passive          ==   reducer
reducer          ==   (redid, local_state)

os_halt (rs,in,out,(ad,gs,ds,ps),instrs)
    =  (rs,in,out',(ad,gs,ds,ps),instrs)
       where   out'     =  out ++ [(my_pid, 0, Msg_Halt)]
               my_pid   =  ad_pid ad

os_new_reducer code nid (rs,in,out,(ad,gs,ds,ps),instrs)
    =  (rs',in,out,(ad',gs,ds,ps),instrs)
       where   rs'                       =  (act_reds ++ [new_red], pass_reds)
               (act_reds, pass_reds)     =  rs
               new_red                   =  (new_redid,(as,bs,cs,nq,pc,wl))
                                            where  as   =  as_init
                                                   bs   =  bs_init
                                                   cs   =  cs_init
                                                   nq   =  nq_new nid
                                                   pc   =  pc_update code pc_init
                                                   wl   =  wl_init
               (ad', new_redid)          =  ad_new_redid ad

os_print string (rs,in,out,(ad,gs,ds,ps),instrs)
    =  (rs,in,out',(ad,gs,ds,ps),instrs)
       where   out'     =  out ++ [(my_pid, 0, Msg_Print string)]
               my_pid   =  ad_pid ad

os_reducerprio (rs,in,out,(ad,gs,ds,ps),instrs)
    =  (rs,in,out,(ad',gs,ds,ps),instrs)
       where   ad'      =  ad_new_prio Prio_Reducer ad

os_release wl (rs,in,out,(ad,gs,ds,ps),instrs)
    =  (rs',in,out',(ad,gs,ds,ps),instrs)
       where   (reds,reqs,nid)           =  (wl_reds wl, wl_reqs wl, wl_nid wl)
               (old_acts, old_pass)      =  rs
               graph          =  gs_copy nid my_pid ds gs
               rs'            =  (new_acts, new_pass)
               new_acts       =  old_acts ++ rel_reds
               new_pass  =  old_pass -- rel_reds
               rel_reds       =  [(rid,loc) | (rid,loc) <- old_pass; rid' <- reds; rid' = rid]
```

```
              out'            =  out ++ [(my_pid, n_pid_of_global dst,
                                             Msg_Graph graph dst) | dst <- reqs]
              my_pid          =  ad_pid ad

os_schedule ((act_reds,pass_reds),in,out,glob,instrs)
   =  ((act_reds,pass_reds),in,out,glob,instrs)      , if act_reds = [ ]
   =  (rs',in,out,glob,instrs)                       , otherwise
      where   rs'                 =  (new_act_reds, pass_reds)
              new_act_reds        =  rest_act_reds ++ [red]
              red : rest_act_reds =  act_reds

os_send_graph graphid chanid (rs,in,out,(ad,gs,ds,ps),instrs)
   =  (rs,in,out',(ad,gs,ds,ps),instrs)
      where   out'    =  out ++ [(my_pid, pid, Msg_Graph graph' dstid)]
              graph   =  gs_copy graphid my_pid ds gs
              graph'  =  gs_root_update (n_setdescr defer_descr) graph
              pid     =  n_pid_of_global dstid
              dstid   =  n_dest (gs_get chanid gs)
              my_pid  =  ad_pid ad

os_send_newchannel pid chanid (rs,in,out,(ad,gs,ds,ps),instrs)
   =  (rs,in,out',(ad,gs,ds,ps),instrs)
      where   out'      =  out ++ [(my_pid, pid, Msg_CreateChannel chan_eid)]
              chan_eid  =  n_global my_pid chanid
              my_pid    =  ad_pid ad

os_send_newreducer code chanid (rs,in,out,(ad,gs,ds,ps),instrs)
   =  (rs,in,out',(ad,gs,ds,ps),instrs)
      where   chan    =  n_dest (gs_get chanid gs)
              out'    =  out ++ [(my_pid, pid, Msg_NewReducer chan code)]
              pid     =  n_pid_of_global chan
              my_pid  =  ad_pid ad

os_send_request chanid (rs,in,out,(ad,gs,ds,ps),instrs)
   =  (rs,in,out',(ad,gs,ds,ps),instrs)
      where   out'    =  out ++ [(my_pid, pid, Msg_Request srcid dstid)]
              srcid   =  n_global my_pid chanid
              pid     =  n_pid_of_global dstid
              dstid   =  n_dest (gs_get chanid gs)
              my_pid  =  ad_pid ad

os_setwait nid (rs,in,out,(ad,gs,ds,ps),instrs)
   =  (rs,in,out,(ad,gs',ds,ps),instrs)
      where   gs'                       =  gs_update nid (n_wl_add_red rid) gs
              (rid, loc)                =  red
              (red : reds, pass_reds)   =  rs

os_stop_reducer (rs,in,out,glob,instrs)
   =  (rs',in,out,glob,instrs)
      where   rs'                           =  (act_reds, pass_reds)
              (red : act_reds, pass_reds)   =  rs

os_suspend_reducer (rs,in,out,glob,instrs)
   =  (rs',in,out,glob,instrs)
      where   rs'                           =  (act_reds, red : pass_reds)
              (red : act_reds, pass_reds)   =  rs
```

Specification of communication process instructions

```
in_msgs        = =   [msg_contents]
out_msgs       = =   [message]
message        = =   (proc_src, proc_dst, msg_contents)
```

```
msg_contents
 :: =   Msg_CreateChannel ext_nid         | Msg_Channel ext_nid ext_nid
    |   Msg_Request ext_nid ext_nid       | Msg_Graph graph ext_nid
    |   Msg_NewReducer ext_nid instrid    | Msg_Print string         |   Msg_Halt
proc_src          = =   procid
proc_dst          = =   procid
ext_nid           = =   (procid, nodeid)

cp_accept_msg (Msg_CreateChannel src_eid) (in,out,(ad,gs,ds,ps),instrs)
    =  (in,out',(ad,gs',ds,ps),instrs)
       where  (gs', rqid)  =  gs_newnode gs
              out'         =  out ++ [(my_pid, pid, Msg_Channel rq_eid src_eid)]
              rq_eid       =  n_global my_pid rqid
              pid          =  n_pid_of_global src_eid
              my_pid       =  ad_pid ad
cp_accept_msg (Msg_Channel rq_eid dst_eid) (in,out,(ad,gs,ds,ps),instrs)
    =  (in,out,(ad,gs',ds,ps),instrs')
       where  nid      =  n_local dst_eid
              gs'      =  gs_update nid (n_putchannel rq_eid) gs
              wl       =  n_wl_get (gs_get nid gs)
              instrs'  =  os_call instrs [os_release wl]
cp_accept_msg (Msg_NewReducer rq_eid code) (in,out,(ad,gs,ds,ps),instrs)
    =  (in,out,(ad,gs,ds,ps),instrs')
       where  instrs'  =  os_call instrs [os_new_reducer code (n_local rq_eid)]
cp_accept_msg (Msg_Request chan_eid rq_eid) (in,out,(ad,gs,ds,ps),instrs)
    =  (in,out',(ad,gs',ds,ps),instrs)
       where  rqid     =  n_local rq_eid
              rnf      =  n_entry (gs_get rqid gs) = rnf_entry
              out'     =  out ++ [(my_pid, pid, Msg_Graph graph chan_eid)]  , if rnf
                       =  out                                               , otherwise
                          where  pid     =  n_pid_of_global chan_eid
                                 graph   =  gs_copy rqid my_pid ds gs
                                 my_pid  =  ad_pid ad
              gs'      =  gs_update rqid (n_wl_add_req chan_eid) gs   , if ~ rnf
                       =  gs                                          , otherwise
cp_accept_msg (Msg_Graph gr dst_eid) (in,out,(ad,gs,ds,ps),instrs)
    =  (in,out,(ad,gs',ds,ps),instrs')
       where  gs'      =  gs_store nid gr gs
              wl       =  n_wl_get (gs_get nid gs)
              nid      =  n_local dst_eid
              instrs'  =  os_call instrs [os_release wl]
```

D.2 Running PABC programs

Booting and instruction fetching

```
boot:: nat -> ([instruction],[descr]) -> network
boot nr_proc (program,descriptors)    =   nw_init nr_proc (program,descriptors)

machine_cycle:: network -> output
machine_cycle nw
              =  output                                    , if op_halt output
              =  op_append output (machine_cycle nw')      , otherwise
                 where  output  =  nw_output nw
                        nw'     =  nw_exec proc_cycle (nw_distribute_msgs nw)

proc_cycle:: processor -> processor
proc_cycle     =  opsys_cycle . red_cycle . opsys_cycle . comproc_cycle

comproc_cycle:: processor -> processor
```

```
comproc_cycle (rs,in,out,(ad,gs,ds,ps),cq)
    =  (rs,in,out,(ad',gs,ds,ps),cq)        , if ad_prio ad = Prio_Reducer
       where   ad'    =  ad_new_prio No_Prio ad
comproc_cycle (rs,[ ],out,glob,cq)
    =  (rs,[ ],out,glob,cq)
comproc_cycle (rs,(msg:msgs),out,glob,cq)
    =  comproc_cycle (rs,in',out',glob',cq')
       where   (in',out',glob',cq')     =  cp_accept_msg msg (msgs,out,glob,cq)

opsys_cycle:: processor -> processor
opsys_cycle (rs,in,out,glob,cq)
    =  (rs,in,out,glob,cq)                                        , if os_cq_empty cq
    =  opsys_cycle (os_first_call cq (rs,in,out,glob,os_rest_calls cq)), otherwise

red_cycle:: processor -> processor
red_cycle ((act_reds,pass_reds),in,out,glob,cq)
    =  ((act_reds,pass_reds),in,out,glob,cq)                      , if act_reds = [ ]
    =  (rs',in,out,glob',cq')                                     , otherwise
       where   rs'                  =  ((rid,loc') : reds, pass_reds)
               (rid,loc) : reds     =  act_reds
               (loc',glob',cq')     =  fetch_cycle (loc,glob,cq)

fetch_cycle:: state -> state
fetch_cycle ((as,bs,cs,nq,pc,wl),(ad,gs,ds,ps),cq)
    =  ((as,bs,cs,nq,pc,wl),(ad,gs,ds,ps),cq)                 , if ~ os_cq_empty cq
    =  fetch_cycle (currinstr ((as,bs,cs,nq,pc',wl),(ad,gs,ds,ps),cq)), otherwise
       where   pc'          =  pc_next pc
               currinstr    =  ps_get (pc_get pc) ps
```

D.3 The new micro-instructions

```
abstype    astack, bstack, cstack, nqueue, pc , wlist,
           admin, graphstore, descrstore, programstore,
           opsys_callqueue, network, output
   with    …
   gs_copy              ::  nodeid -> procid -> descrstore -> graphstore -> graph
   gs_newchannel        ::  procid -> graphstore -> (graphstore,nodeid)
   gs_root_update       ::  (node -> node) -> graph -> graph
   gs_store             ::  nodeid -> graph -> graphstore -> graphstore

   n_dest               ::  node -> dest
   n_global             ::  procid -> nodeid -> ext_nid
   n_local              ::  ext_nid -> nodeid
   n_pid_of_global      ::  ext_nid -> procid
   n_putchannel         ::  ext_nid -> node -> node
   n_setdescr           ::  descrid -> node -> node
   n_wl_add_red         ::  redid -> node -> node
   n_wl_add_req         ::  request -> node -> node
   n_wl_clear               ::  node -> node
   n_wl_get             ::  node -> wlist
   n_wl_put             ::  wlist -> node -> node

   ds_get_repr          ::  node -> descrstore -> string

   bs_getP              ::  b_src -> bstack -> procid
   bs_pushP             ::  procid -> bstack -> bstack

   nq_add               ::  nodeid -> nqueue -> nqueue
   nq_first             ::  nqueue -> nodeid
   nq_init              ::  nqueue
   nq_empty             ::  nqueue -> bool
```

```
nq_new               ::  nodeid -> nqueue
nq_remove            ::  nqueue -> nqueue

wl_clear             ::  wlist -> wlist
wl_empty             ::  wlist -> bool
wl_init              ::  wlist
wl_new               ::  nodeid -> wlist
wl_nid               ::  wlist -> nodeid
wl_reds              ::  wlist -> [redid]
wl_reqs              ::  wlist -> [request]

ad_init              ::  procid -> procid -> admin
ad_new_redid         ::  admin -> (admin,redid)
ad_new_procid        ::  admin -> (admin,procid)
ad_new_prio          ::  schedule_priority -> admin -> admin
ad_nr_pids           ::  admin -> procid
ad_pid               ::  admin -> procid
ad_prio              ::  admin -> schedule_priority

os_call         ::  opsys_callqueue -> [opsys_instruction] -> opsys_callqueue
os_clear        ::  opsys_callqueue
os_first_call   ::  opsys_callqueue -> opsys_instruction
os_init         ::  procid -> procid -> [instruction] -> [descr] -> opsys_state
os_cq_empty     ::  opsys_callqueue -> bool
os_rest_calls   ::  opsys_callqueue -> opsys_callqueue

nw_distribute_msgs   ::  network -> network
nw_exec              ::  (processor -> processor) -> network -> network
nw_init              ::  procid -> ([instruction],[descr]) -> network
nw_output            ::  network -> output

op_append            ::  output -> output -> output
op_halt              ::  output -> bool
```

Graph store

```
gs_copy top pid ds gs
  =  copy
     where   (copy,bindings)  =  copy_nodes [top] pid ds gs 0 empty_bindings

gs_newchannel pid (nds,0)
  =  error "cannot create new channel; graph-store is full"
gs_newchannel pid (nds,free)
  =  ((nds++[new_channel],free-1),nid)
     where   nid            =  store_size-free
             new_channel    =  Channel (wl_new nid) new_channel_entry (pid,(-1))

gs_root_update f (n : r)   =  f n : r

gs_store nodeid graph (old_graph,free)
  =  error "cannot store graph; graph-store is full"   , if size_graph > free
  =  (new_graph,free-size_graph+1)                     , otherwise
     where   size_graph       =  # graph
             size_old_graph   =  # old_graph
             offset           =  size_old_graph – 1
             (top:rest)       =  graph
             new_graph        =  upd nodeid old_graph (n_copy' top) ++
                                           [adjust_nids node | node<-rest]
             n_copy' (Node descrid wl entry args) old
                        =  Node descrid (n_wl_get old) entry [arg+offset|arg<-args]
             n_copy' (Basic descrid wl entry basic) old
                        =  Basic descrid (n_wl_get old) entry basic
```

```
n_copy' (Channel wl instrid dest) old
        =  Channel (n_wl_get old) instrid dest
upd 0 (nd : nds) f     =  f nd : nds
upd n (nd : nds) f     =  nd : upd (n - 1) nds f
adjust_nids (Node descrid (rd,rq,nid) entry args)
   =  Node descrid (wl_new (nid+offset)) entry [arg+offset|arg<-args]
adjust_nids (Basic descrid (rd,rq,nid) entry basic)
   =  Basic descrid (wl_new (nid+offset)) entry basic
adjust_nids (Channel (rd,rq,nid) instrid dest)
   =  Channel (wl_new (nid+offset)) instrid dest
```

The lazy copying algorithm

```
bindings = =   nodeid -> nodeid

copy_nodes:: [nodeid] -> procid -> descrstore -> graphstore -> nodeid -> bindings
                                                                 -> (graph, bindings)
copy_nodes [ ] pid ds gs n bindings
   =  ([ ],bindings)
copy_nodes (nid : rest) pid ds gs n bindings
   =  copy_nodes rest pid ds gs n bindings     , if ~ bindings_is_empty bindings nid
   =  (nodes,bindings'')                       , otherwise
      where   node                 =  gs_get nid gs
              nodes                =  new_node : nodes'
              (new_node,args)      =  copy_node pid ds n node bindings''
              bindings'            =  bind nid n bindings
              (nodes',bindings'')
                              =  copy_nodes (args++rest) pid ds gs (n+1) bindings'

copy_node:: procid -> descrstore -> nodeid -> node -> bindings -> (node, args)
copy_node pid ds nid (Node descrid wl entry args) bindings
   =  ((Channel (wl_new nid) new_channel_entry rq_eid),[ ])
                                  , if descrid = defer_descr ∨ entry = reserved_entry
   =  ((Node descrid (wl_new nid) new_entry new_args),args)              , otherwise
      where   new_entry         =  d_node_entry (ds_get descrid ds)
              new_args          =  map bindings args
              rq_eid            =  (pid,wl_nid wl)
copy_node pid ds nid (Basic descrid wl entry basic) bindings
   =  ((Basic descrid (wl_new nid) entry basic),[ ])
copy_node pid ds nid (Channel wl entry dest) bindings
   =  ((Channel (wl_new nid) entry dest),[ ])

bind:: nodeid -> nodeid -> bindings -> bindings
bind src dst bindings n     =  dst          , if n  =  src
                            =  bindings n  , otherwise

empty_bindings:: bindings
empty_bindings nodeid    =  –1

bindings_is_empty::bindings -> nodeid -> bool
bindings_is_empty b n      =  b n = –1
```

Nodes

```
node     :: =   Node descrid wlist entry nodeid_seq |   Channel wlist entry dest   |
                Basic descrid wlist entry basic     |   Empty
basic    :: =   Int int          |   Bool bool       |   Pid procid
entry    = =    instrid
dest     = =    (procid,nodeid)

reserved_entry       =  3
```

```
new_channel_entry =  4
defer_descr       =  3

n_dest (Channel wl instrid dest)    =  dest
n_global pid nid                    =  (pid, nid)
n_local (pid, nid)                  =  nid
n_pid_of_global (pid, nid)          =  pid
n_putchannel ext_nid (Channel wl entry dest) = Channel (wl_clear wl) entry ext_nid
n_setdescr new_descr (Node descr wl entry args) = Node new_descr wl entry args

n_wl_add_red rid (Node descr (rids,reqs,nid) entry args )
                                        =  Node descr (rid:rids,reqs,nid) entry args
n_wl_add_red rid (Basic descr (rids,reqs,nid) entry basic )
                                        =  Basic descr (rid:rids,reqs,nid) entry basic
n_wl_add_red rid (Channel (rids,reqs,nid) entry dest)
                                        =  Channel (rid:rids,reqs,nid) entry dest

n_wl_add_req req (Node descr (rids,reqs,nid) entry args )
                                        =  Node descr (rids,req:reqs,nid) entry args
n_wl_add_req req (Basic descr (rids,reqs,nid) entry basic )
                                        =  Basic descr (rids,req:reqs,nid) entry basic
n_wl_add_req req (Channel (rids,reqs,nid) entry dest)
                                        =  Channel (rids,req:reqs,nid) entry dest

n_wl_clear (Node descr wl entry args )  =  Node descr (wl_clear wl) entry args
n_wl_clear (Basic descr wl entry basic) =  Basic descr (wl_clear wl) entry basic
n_wl_clear (Channel wl entry dest)      =  Channel (wl_clear wl) entry dest

n_wl_get (Node descr wl entry args)     =  wl
n_wl_get (Basic descr wl entry basic)   =  wl
n_wl_get (Channel wl entry dest)        =  wl

n_wl_put (reds,reqs,nid) (Node descr (reds',reqs',nid') entry args )
                                        =  (Node descr (reds,reqs,nid') entry args )
n_wl_put (reds,reqs,nid) (Basic descr (reds',reqs',nid') entry basic )
                                        =  (Basic descr (reds,reqs,nid') entry basic )
n_wl_put (reds,reqs,nid) (Channel (reds',reqs',nid') entry dest)
                                        =  (Channel (reds,reqs,nid') entry dest)
```

Descriptor store

```
descr       :: =  Descr ap_entry node_entry arity name
node_entry  = =   instrid

d_node_entry (Descr ap_entry node_entry arity name)    =  node_entry
ds_get_repr (Basic descrid wl entry (Int i)) ds      =  show i
ds_get_repr (Basic descrid wl entry (Bool b)) ds     =  show b
ds_get_repr (Node descrid wl entry args) ds          =  d_name (ds_get descrid ds)
```

B-stack

```
bs_getP n bs    =  p where Pid p = bs_get n bs
bs_pushP p bs   =  (Pid p):bs
```

N-queue

```
nqueue  = =   [nodeid]

nq_add     n nq      =  nq ++ [ n ]
nq_first   (n : nq)  =  n
```

```
nq_init                    =  [ ]
nq_empty  (n : q)          =  False
nq_empty  [ ]              =  True
nq_new    n                =  [ n ]
nq_remove    (n : nq)      =  nq
```

Waiting lists

```
wlist      = =   ([redid], [request],nodeid)
request  = =   ext_nid

wl_clear (reds,reqs,nid)    =  ([ ],[ ],nid)
wl_empty ([ ],[ ],nid)      =  True
wl_empty wl                 =  False
wl_init                     =  ([ ],[ ], –1)
wl_new nid                  =  ([ ],[ ],nid)
wl_nid (reds,reqs,nid)      =  nid
wl_reds (reds,reqs,nid)     =  reds
wl_reqs (reds,reqs,nid)     =  reqs
```

Global administration

```
admin    = =   (nr_procids,new_procid,this_procid,redid,schedule_priority)
nr_procids            = =   procid
new_procid            = =   procid
this_procid           = =   procid
procid                = =   num
redid                 = =   num
schedule_priority     :: =  No_Prio | Prio_Reducer

ad_init nr_pids pid   =  (nr_pids, 0, pid, 0, No_Prio)
ad_new_redid (nr_pids,new_pid,pid,rid,prio)
                      =  ((nr_pids,new_pid,pid, rid',prio), rid')
                         where    rid'     =  rid + 1
ad_new_procid (nr_pids,new_pid,pid, rid,prio)
                      =  ((nr_pids,new_pid',pid, rid,prio),new_pid')
                         where    new_pid'    =  1 + (new_pid mod nr_pids)
ad_new_prio new_prio (nr_pids,new_pid,pid, rid,prio)
                      =  (nr_pids,new_pid,pid, rid,new_prio)
ad_nr_pids    (nr_pids,new_pid,pid, rid,prio)  =  nr_pids
ad_pid        (nr_pids,new_pid,pid, rid,prio)  =  pid
ad_prio       (nr_pids,new_pid,pid, rid,prio)  =  prio
```

Operating system interface

```
opsys_callqueue       = =   os_cq
os_cq                 :: =  OS_cq [opsys_instruction]

os_call (OS_cq instr) new              =  OS_cq (instr ++ new)
os_clear                               =  OS_cq [ ]
os_first_call (OS_cq (first : rest))   =  first
os_cq_empty (OS_cq (first : rest))     =  False
os_cq_empty (OS_cq [ ])                =  True
os_rest_calls (OS_cq (first : rest))   =  OS_cq rest

os_init nr_pids procid program descriptors
   =  (rs,in,out,glob,instrs)
      where    rs        =  ([(rid_main, (as,bs,cs,nq,pc,wl))], [ ])       , if procid = 1
                         =  ([ ], [ ])                                      , otherwise
                            where    as      =  as_init
```

```
                              bs     =  bs_init
                              cs     =  cs_init
                              nq     =  nq_init
                              wl     =  wl_init
                              pc     =  pc_init
          (ad_main,rid_main)     =  ad_new_redid (ad_init nr_pids 1)
          glob      =  (ad,gs,ds,ps)
                       where  ad     =  ad_main                  , if procid = 1
                                     =  ad_init nr_pids procid  , otherwise
                              gs     =  gs_init
                              ps     =  ps_init program
                              ds     =  ds_init descriptors
          in        =  [ ]
          out       =  [ ]
          instrs    =  os_clear
```

The communication network

```
network            = =  [network_element]
network_element    :: = P processor
processor          = =  opsys_state
output             = =  [msg_contents]

nw_distribute_msgs processors
   =  [ P (rs,in ++ select_msgs (ad_pid ad) msgs,out,(ad,gs,ds,ps),instrs) |
                                 P (rs,in,out,(ad,gs,ds,ps),instrs) <- new_procs]
      where   msgs = [msg |
                        P (rs,in,proc_out,glob,instrs)<-processors;msg<-proc_out]
              new_procs = [ P (rs,in,[ ],glob,instrs) |
                                 P (rs,in,proc_out,glob,instrs) <- processors]
              select_msgs pid msgs = [cts | (src,dst,cts) <- msgs; dst = pid]

nw_exec proc_cycle processors
   =  [ P (proc_cycle processor) | P processor<-processors]

nw_init nr_procids (program,descriptors)
   =  [ P (os_init nr_procids procid program descriptors) | procid<-[1..nr_procids]]

nw_output processors
   =  [ cts | P (rs,in,proc_out,glob,instrs)<-processors;(src,dst,cts)<-proc_out;dst=0]

op_append o1 o2    =  o1 ++ o2
op_halt msgs       =  foldr (\/) False (map (= Msg_Halt) msgs)
```