Algorithm Fully symbolic memory: naive implementation

$formula\_types = \{ft_{concrete}, ft_{symbolic}, ft_{memset}, ft_{memcpy}\}$
M_s := memory of state s
$M_s.concrete$:= $\{e \rightarrow (ft, e, v, t, \delta)\}$ where e must be a concrete expression
$M_s.symbolic$:= $\{(ft, a, b, e, v, t, \delta)\}$
$M_s.memcpy$:= $\{(ft, \alpha, \beta, size, t, \delta)\}$
$M_s.memset$:= $\{(ft, e, size, v, t, \delta)\}$

Immutable objects:
t_{pos} := timestamp (initially set to 0)
t_{neg} := timestamp (initially set to 0)
e := an expression over symbols and concrete values
v := a 1-byte expression over symbols and concrete values
V := ordered set of 1-byte values
π := set of assumptions

function STORE(e, V, $size$):
  for $k = 0$ to $size - 1$ **do**
    _STORE($e + k$, v_k)
  end for
end function

function _STORE(e, v):
  $a \leftarrow min(e)$
  $b \leftarrow max(e)$
  $t_{pos} \leftarrow t_{pos} + 1$
  if $a == b$ **then**
    $M_s.concrete[a] \leftarrow M_s.concrete[a] \cup \{(ft_{concrete}, e, v, t_{pos}, true)\}$
  else
    $M_s.symbolic \leftarrow M_s.symbolic \cup \{(ft_{symbolic}, a, b, e, v, t_{pos}, true)\}$
  end if
end function

function SEARCH(a, b, t):
  $P \leftarrow \varnothing$
  for $k \leftarrow a$ to b **do**
    $P \leftarrow P \cup \{x \mid x \in M_s.concrete[k] \wedge x(t) \leq t\}$
  end for
  $P \leftarrow P \cup \{x \mid x \in M_s \wedge [x(a), x(b)] \cap [a, b] \neq \varnothing \wedge x(t) \leq t\}$
  $P \leftarrow P \cup \{x \mid x \in M_s.memcpy \wedge x(t) \leq t\}$
  $P \leftarrow P \cup \{x \mid x \in M_s.memset \wedge x(x) \leq t\}$
  return P
end function

function LOAD(e, $size$):
$V = \langle\rangle$
for $k = 0$ to $size - 1$ **do**
$v_k = \_LOAD(e + k, t_{pos})$
$V = V \cdot v_k$
end for
return V
end function

function _LOAD(e, t):
$a \leftarrow min(e)$
$b \leftarrow max(e)$
$P \leftarrow$ SEARCH(a, b, t)
$P \leftarrow$ SORT_BY_INCREASING_TIMESTAMP(P)
$v \leftarrow$ SYMBOLIC_INPUT() ▷ unintialized memory
$t_{neg} \leftarrow t_{neg} - 1$
$M_s.symbolic \leftarrow M_s.symbolic \cup \{(a, b, e, v, t_{neg}, true)\}$ ▷ implicit store
for $x \in P$ **do**
if $x(ft) == ft_{concrete} \vee x(ft) == ft_{symbolic}$ **then**
$v \leftarrow ite(x(e) = e \wedge x(\delta), x(v), v)$
else if $x(ft) == ft_{memset}$ **then**
$v \leftarrow ite(x(e) \leq e \leq x(e) + x(size) \wedge x(\delta), x(v), v)$
else if $x(ft) == ft_{memcpy}$ **then**
$v \leftarrow ite(x(\beta) \leq e \leq x(\beta) + x(size) \wedge x(\delta), \_LOAD(e - \beta + \alpha, x(t)), v)$
end if
end for
return v
end function

function MEMCPY(e_{dst}, e_{src}, $size$):
$M_s.memcpy \leftarrow M_s.memcpy \cup \{(ft_{memcpy}, e_{src}, e_{dst}, size, t, true)\}$
end function

function MEMSET(e, $size$, v):
$M_s.memset \leftarrow M_s.memset \cup \{(ft_{memset}, e, size, v, t, true)\}$
end function

function MERGE$((S_1, \delta_1), (S_2, \delta_2), S_{ancestor})$: ▷ $S_1 := self$
$t^{anc}_{pos} = S_{ancestor}.t_{pos}$
$t^{anc}_{neg} = S_{ancestor}.t_{neg}$
$M_s \leftarrow S_{ancestor}.M_s$
for $\{x \in S_1.M_s \mid (x(t) > 0 \wedge x(t) > t^{anc}_{pos}) \vee (x(t) < 0 \wedge x(t) < t^{anc}_{neg})\}$ **do**
$x(\delta) = x(\delta) \wedge \delta_1$
M_s.ADD$((x(a), x(b)), x))$
end for
for $\{x \in S_2.M_s \mid (x(t) > 0 \wedge x(t) > t^{anc}_{pos}) \vee (x(t) < 0 \wedge x(t) < t^{anc}_{neg})\}$ **do**
$x(\delta) = x(\delta) \wedge \delta_2$
M_s.ADD$((x(a), x(b)), x))$
end for
$t_{pos} = max(S_1.t_{pos}, S_2.t_{pos})$
$t_{neg} = min(S_1.t_{neg}, S_2.t_{neg})$
end function