Algorithm Fully symbolic memory: naive implementation

Immutable objects:

M_c	$:= \{(e_c, v)\}$
e_c	:= an expression over concrete values
M_s	$:= \{(e, v)\}$
e	:= an expression over symbols and concrete values
v	:= a 1-byte expression over symbols and concrete values
V	:= ordered set of v
π	:= set of assumptions
$equiv(e, \widetilde{e}, \pi)$	$:= (e \neq \widetilde{e} \wedge \pi) == UNSAT$
$disjoint(e, \widetilde{e}, \pi)$	$:= (e = \widetilde{e} \wedge \pi) == UNSAT$
$intersect(e, \widetilde{e}, \pi)$	$:= (e = \widetilde{e} \wedge \pi) == SAT$

```
1: function STORE(e, v, size):
2:     for k = 0 to size − 1 do
3:         _STORE(e + k, v_k)
4:     end for
5: end function
```

```
1: function _STORE(e, V):
2:     min_e = min(e)
3:     max_e = max(e)
4:     flag = false
5:     constant = false
6:     M_c' ← M_c
7:     M_s' ← M_s
8:     if min_e == max_e then
9:         constant = true
10:        M_c' ← M_c'|_{e↦v}
11:    end if
12:    for (ẽ, ṽ) ∈ M_s do
13:        if disjoint(ẽ, e, π) then
14:            continue
15:        else if equiv(ẽ, e, π) then
16:            if constant then
17:                M_s' ← M_s' − (ẽ, v)
18:            else
19:                M_s' ← M'|_{ẽ↦v}
20:            end if
21:            flag = true
22:        else
23:            M_s' ← M_s'|_{ẽ↦ite(ẽ=e∧π,v,ṽ)}
24:        end if
25:    end for
26:    if ¬flag ∧ ¬constant then
27:        M_s' ← M_s'|_{e↦v}
28:    end if
29:    M_c ← M_c'
30:    M_s ← M_s'
31: end function
```

```
1: function LOAD(e, size):
2:     V = []
3:     for k = 0 to size − 1 do
4:         v_k = _LOAD(e + k)
5:         V = V ∪ v_k
6:     end for
7:     return V
8: end function
```

```
1: function _LOAD(e):
2:     v = ⊥
3:     min_e = min(e)
4:     max_e = max(e)
5:     if min_e == max_e then
6:         if (min_e, ṽ) ∈ M_c then
7:             v = ṽ
8:         end if
9:     else
10:        for e_c = min_e to max_e do
11:            if (e_c, ṽ) ∈ M_c then
12:                v = ite(e_c = e, ṽ, v)
13:            end if
14:        end for
15:    end if
16:    for (ẽ, ṽ) ∈ M do
17:        if intersect(ẽ, e, π) then
18:            v = ite(ẽ = e ∧ π, ṽ, v)
19:        end if
20:    end for
21:    return v
22: end function
```