Algorithm Fully symbolic memory: naive implementation

Immutable objects:

M	$:= \{(e, v)\}$
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:     M' ← M
3:     for (ẽ, ṽ) ∈ M do
4:         if disjoint(ẽ, e, π) then
5:             continue
6:         else if equiv(ẽ, e, π) then
7:             M' ← M'|_{ẽ↦v}
8:             flag = true
9:         else
10:            M' ← M'|_{ẽ↦ite(ẽ=e∧π,v,ṽ)}
11:        end if
12:    end for
13:    if ¬flag then
14:        M' ← M'|_{e↦v}
15:    end if
16:    M ← M'
17: 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:     for (ẽ, ṽ) ∈ M do
4:         if intersect(ẽ, e, π) then
5:             v = ite(ẽ = e ∧ π, ṽ, v)
6:         end if
7:     end for
8:     return v
9: end function
```