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_s'|_{ẽ↦v}
20:            end if
21:            flag = true
22:        else
23:            M_s' ← M_s'|_{ẽ↦ite(ẽ=e,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
```

```
function LOAD(e, size):
    V = ⟨⟩
    for k = 0 to size − 1 do
        v_k = _LOAD(e + k)
        V = V · v_k
    end for
    return V
end function
```

```
function _LOAD(e):
    v = ⊥
    min_e = min(e)
    max_e = max(e)
    if min_e == max_e then
        if (min_e, ṽ) ∈ M_c then
            v = ṽ
        end if
    else
        for e_c = min_e to max_e do
            if (e_c, ṽ) ∈ M_c then
                v = ite(e_c = e, ṽ, v)
            end if
        end for
    end if
    for (ẽ, ṽ) ∈ M_s do
        if intersect(ẽ, e, π) then
            v = ite(ẽ = e, ṽ, v)
        end if
    end for
    return v
end function
```

```
function MERGE((M^1_c, M^1_s, π^1), (M^2_c, M^2_s, π^2), ...):
    M_c ← {},  M_s ← {}
    π ← ⋁_i pi^i

    for (e, v) ∈ M^i_c do
        if (e, v′) ∈ M_c then
            M_c ← M_c|_{e↦ite(π^i, v, v′)}
        else
            M_c ← M_c|_{e↦ite(π^i, v, ⊥)}
        end if
    end for

    for (e, v) ∈ M^i_s do
        M_s ← M_s|_{e↦ite(π^i, v, ⊥)}
    end for

    return (M_c, M_s, π)
end function
```