Algorithm Fully symbolic memory: naive implementation

Immutable objects:

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:     a = min(e)
3:     b = max(e)
4:     t ← t + 1
5:     INSERT((a, b), (e, v, t, true)))
6: end function
```

```
1: function INSERT((a, b), (e, v, t, δ))):
2:     for x ∈SEARCH(a, b): do
3:         if equiv_sup(e, x(e)) then
4:             x(v) ← v
5:             x(t) ← t
6:             x(δ) ← δ
7:             return
8:         end if
9:     end for
10:    ADD((a, b), (e, v, t, δ)))
11: end function
```

```
1: function SEARCH(a, b)):
2:     return {x ∈ M_s | x(a, b) ∩ [a, b] ≠ ∅}
3: 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:     a = min(e)
3:     b = max(e)
4:     P ← {(ẽ, ṽ, t̃, δ̃) | (ẽ, ṽ, t̃, δ̃) ∈ SEARCH(a, b)}
5:     P′ ← SORT_BY_INCREASING_TIMESTAMP(P)
6:     v ← ⊥
7:     for (ẽ, ṽ, t̃, δ̃) ∈ P′ do
8:         v ← ite(e = ẽ ∧ δ̃, ṽ, v)
9:     end for
10:    return v
11: end function
```

```
1: function MERGE((S_1, δ_1), (S_2, δ_2)):
2:     return
3: end function
```