Algorithm Fully symbolic memory: naive implementation

Immutable objects:

M	$:= \{(e, v)\}$
e	:= an expression over symbols and concrete values
v	:= an expression over symbols and concrete values
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):
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):
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
```

```
1: function BRANCH(e):
2:     used = false
3:     if (e ∧ π) then
4:         π' = e ∧ π
5:         used = true
6:     end if
7:     if (¬e ∧ π) then
8:         π'' = ¬e ∧ π
9:         if used then
10:            fork execution using (π'', M)
11:        else
12:            π' = π''
13:        end if
14:    end if
15:    π = π'
16: end function
```