I am certainly bad at theorem provers, so this didn’t get anywhere (it exploded into errors about type universes when I tried to prove a theorem), probably because of stupid mistakes. Nevertheless, this snippet (Lean 4) does show that the terminating subset of Nock (anything not involving operation 2) always terminates or else nonterminates in a detectable way (here represented as none

).

section abbrev Atom := Nat inductive Noun where | atom : Atom -> Noun | cell : Noun -> Noun -> Noun deriving Repr, DecidableEq open Noun

instance : OfNat Noun a where ofNat := Noun.atom a

abbrev Axis := Atom def axis_of : (a : Axis) -> (n : Noun) -> Option Noun | 0, _ => none | 1, n => n | _a, atom _ => none | 2, cell h _t => h | 3, cell _h t => t | (Nat.succ a'), n => let a := a' + 1 if a % 2 == 0 then let n' := axis_of (a / 2) n match n' with | none => none | atom _ => none | cell h _t => some h else let n' := axis_of (a / 2) n match n' with | none => none | atom _ => none | cell _h t => some t

def const_noun : Noun -> Noun := id

-- nock' is the terminating or provably nonterminating subset of nock def nock' : (subject : Noun) -> (formula : Noun) -> Option Noun | s, cell (cell l_h l_t) r => if let some res_l := nock' s (cell l_h l_t) then if let some res_r := nock' s r then some (cell res_l res_r) else none else none | s, cell 0 (atom a) => axis_of a s | *, cell 1 n => const_noun n | s, cell 3 subformula => match (nock' s subformula) with | none => none | atom _ => some 1 | cell _ _ => some 0 | s, cell 4 subformula => match (nock' s subformula) with | none => none | atom a => some (atom a.succ) | cell _ _ => none | s, cell 5 (cell sub_l sub_r) => if let some res_l := nock' s sub_l then if let some res_r := nock' s sub_r then if res_l == res_r then some 0 else some 1 else none else none* | , _ => none

-- fully nonterminating nock partial def nock : (subject : Noun) -> (formula : Noun) -> Noun | s, cell 2 (cell sub_s sub_f) => let res_s := nock s sub_s let res_f := nock s sub_f nock res_s res_f | s, f => match nock' s f with | none => nock s f | some r => r end