

```

module target where

-- Operator precedence and associativity
infix 4 _≤s_
infixl 6 _÷s_ _-s_

open import lib

-- Stack descriptor: (frames, displacement)
record SD : Set where
  constructor ⟨_,_⟩
  field
    f : ℕ
    d : ℕ

-- Stack descriptor operations
_+_s_ : SD → ℕ → SD
⟨ S_f , S_d ⟩ +_s n = ⟨ S_f , S_d + n ⟩

_÷s_ : SD → ℕ → SD
⟨ S_f , S_d ⟩ ÷s n = ⟨ S_f , S_d ÷ n ⟩

-- _-s_ : (sd : SD) → (n : ℕ) → n ≤ SD.d sd → SD
-- (⟨ S_f , S_d ⟩ -s n) p = ⟨ S_f , (S_d - n) p ⟩

_-s_ : (sd : SD) → (n : ℕ) → (p : n ≤ SD.d sd) → SD
(⟨ S_f , S_d ⟩ -s n) p = ⟨ S_f , (S_d - n) p ⟩

-- Stack descriptor lexicographic ordering
data _≤s_ : SD → SD → Set where
  <-f : ∀ {S_f S'_f S_d S'_d} → S_f < S'_f → ⟨ S_f , S_d ⟩ ≤s ⟨ S'_f , S'_d ⟩
  ≤-d : ∀ {S_f S_d S'_d} → S_d ≤ S'_d → ⟨ S_f , S_d ⟩ ≤s ⟨ S_f , S'_d ⟩

≤s-refl : ∀{sd : SD} → sd ≤s sd
≤s-refl {⟨ f , d ⟩} = ≤-d ≤s-refl

≤s-trans : ∀{sd sd' sd'' : SD} → sd ≤s sd' → sd' ≤s sd'' → sd ≤s sd''
≤s-trans (<-f f<f') (≤-d _) = <-f f<f'
≤s-trans (<-f f<f') (<-f f'<f'') = <-f (<-trans f<f' f'<f'')
≤s-trans (≤-d _) (<-f f'<f'') = <-f f'<f''
≤s-trans (≤-d d≤d') (≤-d d'≤d'') = ≤-d (≤-trans d≤d' d'≤d'')

-- Operator
data UnaryOp : Set where
  UNeg : UnaryOp

data BinaryOp : Set where
  BPlus : BinaryOp
  BMinus : BinaryOp
  BTimes : BinaryOp

```

```

data RelOp : Set where
  RLeq : RelOp
  RLt : RelOp

-- Nonterminals
-- Lefthand sides
data L (sd : SD) : Set where
  l-var : (sdv : SD) → sdv ≤s sd → 1 → L sd
  l-sbrs : L sd

-- Simple righthand sides
data S (sd : SD) : Set where
  s-l : L sd → S sd
  s-lit : ℤ → S sd

-- Righthand sides
data R (sd : SD) : Set where
  r-s : S sd → R sd
  r-unary : UnaryOp → S sd → R sd
  r-binary : S sd → BinaryOp → S sd → R sd

-- Instruction sequences
data I (sd : SD) : Set where
  stop : I sd
  assign_inc : (δ : ℕ) → L (sd +s δ) → R sd → I (sd +s δ) → I sd
  assign_dec : (δ : ℕ) → (p : δ ≤ SD.d sd) → L ((sd -s δ) p) → R sd → I
  ((sd -s δ) p) → I sd
  if-then-else_inc : (δ : ℕ) → S sd → RelOp → S sd → I (sd +s δ) → I (sd
+s δ) → I sd
  if-then-else_dec : (δ : ℕ) → (p : δ ≤ SD.d sd) → S sd → RelOp → S sd →
I ((sd -s δ) p) → I ((sd -s δ) p) → I sd
  adjustdisp_inc : (δ : ℕ) → I (sd +s δ) → I sd
  adjustdisp_dec : (δ : ℕ) → (p : δ ≤ SD.d sd) → I ((sd -s δ) p) → I sd
  poppto : (sd' : SD) → sd' ≤s sd → I sd' → I sd

```