target

```
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 ≤-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 ∸ₛ 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
    popto : (sd' : SD) → sd' ≤ₛ sd → I sd' → I sd
```