```
type DiceChoice = [ Bool ]
type DiceVals   = [ Integer ]
type DiceTurn  = (DiceChoice, DiceVals)

pop :: DiceTurn
    -> Maybe ((Bool, Integer), DiceTurn)
pop ([], []) = Nothing
pop (chosen:choices, v:vs) = Just ((chosen, v),
  (choices, vs))
pop (_:_, []) = error "Invariant violated: missing
  val"
pop ([], _:_) = error "Invariant violated: missing
  choice"

allRolls :: DiceTurn
         -> Integer

example =
  let diceChoices = [False, True, True, False, False]
      diceVals = [ 6, 4, 4, 3, 1 ]
  in mapM_ print $ allRolls (diceChoices, diceVals) 2
```

```
type DiceVals   = [ Integer ]
type DiceTurn  = [(Bool, Integer)]

pop :: DiceTurn
    -> Maybe ((Bool, Integer), DiceTurn)
pop [] = Nothing
pop (a:as) = Just (a, as)



allRolls :: DiceTurn
         -> Integer

example =
  let diceChoices = [False, True, True, False, False]
      diceVals = [ 6, 4, 4, 3, 1 ]
  in mapM_ print $ allRolls
     (zip diceChoices diceVals) 2
```