

```

01 pop :: DiceChoice
02     -> DiceVals
03     -> Maybe ((Bool, Integer), (DiceChoice, DiceVals))
04 pop [] [] = Nothing
05 pop (chosen:choices) (v:vs) = Just ((chosen, v),
06     (choices, vs))
07 pop ( : ) [] = error "Invariant violated: missing val"
08 pop [] ( : _) = error "Invariant violated: missing
09     choice"
10
11 allRolls :: DiceChoice
12     -> DiceVals
13     -> Integer
14     -> [ (DiceVals, Integer) ]
15 allRolls choices vs n = case pop choices vs of
16     Nothing -> [ ([], n-1) ]
17     Just ((chosen, v), (choices, vs)) ->
18         allRolls choices vs (error "Didn't expect to use")
19         >=> \ (roll, ) -> [ (d:roll, n-1)
20             | d <- rollList ]
21     where
22         rollList = if chosen then [v] else [ 1..6 ]
23
24 example =
25     let diceChoices = [ False, True, True, False, False
26         diceVals = [ 6, 4, 4, 3, 1 ]
27     in mapM_ print $ allRolls diceChoices diceVals 2
28

```

```

01 pop :: (DiceChoice, DiceVals)
02     -> Maybe ((Bool, Integer), (DiceChoice, DiceVals))
03 pop ([], []) = Nothing
04 pop (chosen:choices, v:vs) = Just ((chosen, v),
05     (choices, vs))
06 pop ( : , []) = error "Invariant violated: missing val"
07 pop ([], _:_ ) = error "Invariant violated: missing cho
08
09 allRolls :: (DiceChoice, DiceVals)
10     -> Integer
11     -> [ (DiceVals, Integer) ]
12 allRolls (choices, vs) n = case pop (choices, vs) of
13     Nothing -> [ ([], n-1) ]
14     Just ((chosen, v), (choices, vs)) ->
15         allRolls (choices, vs) (error "Didn't expect to us
16         >=> \ (roll, ) -> [ (d:roll, n-1)
17             | d <- rollList ]
18     where
19         rollList = if chosen then [v] else [ 1..6 ]
20
21 example =
22     let diceChoices = [ False, True, True, False, False
23         diceVals = [ 6, 4, 4, 3, 1 ]
24     in mapM_ print $ allRolls (diceChoices, diceVals) 2
25

```