DSLs, sets and von Neumann

Group A1.10 Martin Fredin

January 28, 2022

Part 1:

```
  -- abstract syntax of set expressions with variables of type v
data TERM v = Empty
  | Singleton (TERM v)
  | Union     (TERM v) (TERM v)
  | Intersection (TERM v) (TERM v)
  | Var       v
  deriving Show
  -- predicates over pure set expressions
data PRED v = Elem (TERM v) (TERM v)
  | Subset          (TERM v) (TERM v)
  | And             (PRED v) (PRED v)
  | Or              (PRED v) (PRED v)
  | Implies (PRED v) (PRED v)
  | Not             (PRED v)
  deriving Show
```

Part 2:

```
eval :: Eq v => Env v Set -> TERM v -> Set
eval env term = case term of
  Empty       -> S []
  Singleton t -> S [eval env t]
  Union     t t1 -> S $ f t `union` f t1
  Intersection t t1 -> S $ f t `intersect` f t1
  Var       v -> fromJust $ lookup v env
  where
  f = (\(S xs) -> xs) . eval env
  g Nothing = error "variable is not in enviroment"
  g (Just s) = s


check :: Eq v => Env v Set -> PRED v -> Bool
check env pred = case pred of
  Elem   t t1 -> eval env t `elem` g t1
```

```
  Subset t t1 → all (∈ g t1) $ g t
  And    p p1 → f p ∧ f p1
  Or     p p1 → f p ∨ f p1
  Implies p p1 → ¬ (f p) ∨ f p1
  Not    p → ¬ $ f p
  where
  f = check env
  g = (λ(S xs) → xs) ∘ eval env
```

Part 3:

```
vonNeumann :: Int → TERM v
vonNeumann 0 = Empty
vonNeumann n = Union (vonNeumann $ n − 1) (Singleton (vonNeumann $ n − 1))
claim :: Int → Int → Bool
claim i i1 | (#) n ⩽ (#) n = check env (Subset n n1)
  where
  (n, n1) = (vonNeumann i, vonNeumann i1)
  (#)     = (λ(S xs) → length xs) ∘ eval env
claim1 :: Int → Bool
claim1 i = n i ≡ S [n j | j ← [0 .. i − 1]]
  where
  n = eval env ∘ vonNeumann
```