Lista zadań nr 6

Zadanie 1. (2 pkt)

Napisz funkcję, która sprawdza, czy wyrażenie języka LET z wykładu jest zamknięte (czyli nie ma w nim zmiennych wolnych):

```
closed (e : expr) : bool = ...
```

Zadanie 2. (1 pkt)

Zaproponuj składnię abstrakcyjną w postaci definicji typu w OCamlu (np. rozszerzenia typu wyrażeń z wykładu) dla następujących konkretnych notacji:

- `for i := n to m do ... end  (* pętla w Pascalu *)`
- $\int_k^n f(x)\,dx$ `  (* całka oznaczona *)`

Zadanie 3. (1 pkt)

Dodaj do języka LET typ unit (składnię abstrakcyjną, odpowiadającą wartość, parser, ewaluator). Unit to wyjątkowy typ, bo ma konstruktor, ale nie potrzebuje destruktorów (czy rozumiesz dlaczego?).

Zadanie 4. (2 pkt)

Dodaj do języka LET typ par (składnię abstrakcyjną, odpowiadającą wartość, parser, ewaluator):

- Składnia konkretna konstruktora par to `(e1,e2)`, gdzie `e1` i `e2` to dowolne wyrażenia,
- Konstrukcjami eliminującymi pary są `fst p` i `snd p`, które ujawniają odpowiednio pierwszy i drugi element pary, do której oblicza się wyrażenie `p`.

Zadanie 5. (1 pkt)

Zmodyfikuj język z poprzedniego zadania tak, by konstrukcją eliminującą parę było (płytkie) dopasowanie wzorca, tzn. miała ona składnię konkretną

```
match p with (x, y) -> e
```

gdzie `p` i `e` to wyrażenia, a `x` i `y` to identyfikatory.

Zadanie 6. (1 pkt)

Do składni abstrakcyjnej i konkretnej wyrażeń języka LET z wykładu dodaj konstrukcję `sum` o następującej składni konkretnej:

```
sum x = n to m in k
```

gdzie `sum`, `to` i `in` to słowa kluczowe, `x` to identyfikator, a `n`, `m` i `k` to wyrażenia. Rozszerz odpowiednio lekser i parser.

Zadanie 7. (2 pkt)

Rozszerz ewaluator języka LET o konstrukcję `sum`. Semantyką wyrażenia

```
sum x = n to m in k
```

jest:

1. Oblicz wartość wyrażenia `n` do liczby n (jeśli wyrażenie oblicza się do innego rodzaju wartości, zgłoś błąd typów).
2. Oblicz wartość wyrażenia `m` do liczby m (jeśli wyrażenie oblicza się do innego rodzaju wartości, zgłoś błąd typów).
3. Dla każdej liczby naturalnej i z zakresu $[n, m]$ oblicz wartość wyrażenia `k` z wartością i podstawioną za zmienną `x`. Jeśli któreś z tych wyrażeń nie oblicza się do liczby, zgłoś błąd typów.
4. Wartością wyrażenia `sum x = n to m in k` jest suma wszystkich wyrażeń z punktu 3.

Przykładowo, wartością wyrażenia

```
sum x = 3 - 2 to 5 in x * x
```

powinna być suma kwadratów liczb 1..5, czyli $\sum_{x=3-2}^{5} x^2 = 55$.

Zadanie 8. (2 pkt)

QBF (ang. *quantified boolean formulas*) to język formuł logicznych, które przypominają rachunek zdań, ale dodatkowo pozwalają na kwantyfikację zmiennych. Formuła QBF składa się więc z:

- stałych $\top$ oraz $\bot$,
- zmiennych,
- spójników $\vee$, $\wedge$, $\neg$,
- kwantyfikatorów $\forall$ i $\exists$.

Przykładowa formuła może więc wyglądać następująco:

$$\forall x.\exists y.\exists z.(x \vee y) \wedge z$$

Prawdziwość zamkniętych formuł możemy zdefiniować nieformalnie w następujący sposób:

- Stała $\top$ jest zawsze prawdziwa; stała $\bot$ nigdy nie jest prawdziwa;
- Formuła $\phi \vee \psi$ jest prawdziwa, jeśli prawdziwa jest formuła ϕ lub formuła ψ; odpowiednio dla formuł stworzonych przy użyciu spójników $\wedge$ i $\neg$;
- Formuła $\forall x.\phi$ jest prawdziwa, jeśl formuła ϕ jest prawdziwa zarówno dla $x =$ `true` i $x =$ `false`;
- Formuła $\exists x.\phi$ jest prawdziwa, jeśl formuła ϕ jest prawdziwa dla $x =$ `true` lub jest prawdziwa dla $x =$ `false`.

W OCamlu formuły QBF możemy reprezentować przy użyciu następującego typu danych:

```
type ident = string

type qbf =
  | Top                     (* ⊤ *)
  | Bot                     (* ⊥ *)
  | Var of ident            (* x *)
  | Disj of qbf * qbf       (* ∨ *)
  | Conj of qbf * qbf       (* ∧ *)
  | Not of qbf              (* ¬ *)
  | Forall of ident * qbf   (* ∀ *)
  | Exists of ident * qbf   (* ∃ *)
```

Zaimplementuj ewaluator formuł QBF, w którym wyrażeniami są wartości typu `qbf`, a wartościami wartości typu `bool`. Niech ewaluator działa przez podstawienie stałych `Top` i `Bot` za zmienne:

```
let subst (x : ident) (s : qbf) (f : qbf) : qbf = ...
let eval (f : qbf) : bool = ...
```