Zadanie nr 3 na pracownię

Weryfikator dowodów logiki pierwszego rzędu

Celem pracowni jest napisanie weryfikatora dowodów w intuicjonistycznej logice pierwszego rzędu z elementami logiki drugiego rzędu. Wejściem weryfikatora jest lista aksjomatów oraz twierdzeń wraz z ich dowodami zapisanymi w postaci pewnego programu funkcyjnego. Weryfikacja dowodów w istocie polega na sprawdzeniu typów, dlatego weryfikator będzie mieć strukturę podobną do implementacji typowanego języka FUN z wykładu.

Przykład

Poniżej znajduje się przykładowy dowód, który jest poprawnym wejściem weryfikatora. Dla poprawnych dowodów weryfikator powinien się zakończyć bez błędu.

(* Niech R będzie relacją, która jest symetryczna i przechodnia. *)

```
axiom symm  : forall x y, R(x, y) -> R(y, x)
axiom trans : forall x y z, R(x, y) -> R(y, z) -> R(x, z)
```

(* Załóżmy, że element 0 jest w relacji (po lewej, lub po prawej) z każdym innym elementem. *)

```
axiom rel0 : forall x, R(0, x) or R(x, 0)
```

(* Wówczas, z symetryczności 0 jest w relacji po lewej oraz po prawej z każdym innym elementem. *)

```
theorem rel0_l : forall x, R(0, x)
proof
  fun [x] ->
    case rel0 [x] of
    | left  H -> H
```

```
    | right H -> symm [x] [0] H
qed

theorem rel0_r : forall x, R(x, 0)
proof
  fun [x] ->
    case rel0 [x] of
    | left  H -> symm [0] [x] H
    | right H -> H
qed
```

(* A zatem relacja jest zwrotna. *)

```
theorem refl : forall x, R(x, x)
proof
  fun [x] -> trans [x] [0] [x] (rel0_r [x]) (rel0_l [x])
qed
```

Termy, formuły i dowody

W składni rozróżniamy termy (t), formuły (φ) oraz dowody (e). Termy są albo zmienną termową, albo symbolem funkcyjnym zaaplikowanym do listy termów.

```
type term =
  | Var  of term_var                   (* x *)
  | Func of fun_symbol * term list  (* f(t_1,...,t_n) *)
```

W poprawnych dowodach wszystkie zmienne termowe powinny być związane, dlatego wprowadzimy relację dobrego sformowania termów nad zbiorem zmiennych Δ, zapisywaną jako $\Delta \vdash \mathbf{wft}(t)$. Relacja ta jest zdefiniowana poprzez następujący zestaw reguł.

$$\frac{x \in \Delta}{\Delta \vdash \mathbf{wft}(x)} \qquad \frac{\Delta \vdash \mathbf{wft}(t_1) \quad \dots \quad \Delta \vdash \mathbf{wft}(t_n)}{\Delta \vdash \mathbf{wft}(f(t_1, \dots, t_n))}$$

Podobną relację wprowadzimy dla formuł ($\Delta \vdash \mathbf{wff}(\varphi)$). Ponadto, o formułach możemy myśleć jak o typach dowodów, więc wprowadzimy relację również typowania ($\Delta; \Gamma \vdash e : \varphi$). Zauważ, że relacja typowania wiąże dwa środowiska: Δ jest zbiorem zmiennych termowych, a Γ funkcją częściową ze zmiennych dowodowych do ich typów (formuł). Te dwa rodzaje zmiennych tworzą rozłączne przestrzenie nazw, tzn. zmienne termowe i dowodowe o tej samej nazwie są traktowane jako różne zmienne. Reguły definiujące relację typowania i dobrego sformowania formuł wprowadzimy krok po kroku w kolejnych sekcjach, omawiając przy tym kolejne spójniki logiczne.

Relacje i zmienne

Formuła może mieć postać symbolu relacyjnego zaaplikowanego do listy termów.

```
type formula =
  ...
  | Rel of rel_symbol * term list  (* R(t_1, ..., t_n) *)
  ...
```

Taka formuła jest dobrze sformowana, jeśli wszystkie podtermy są dobrze sformowana.

$$\frac{\Delta \vdash \mathbf{wft}(t_1) \quad \dots \quad \Delta \vdash \mathbf{wft}(t_n)}{\Delta \vdash \mathbf{wff}(R(t_1, \dots, t_n))}$$

Dla formuł tej postaci nie ma reguł wprowadzania ani eliminacji, ale można je udowodnić (jak inne formuły) korzystając z założenia. W składni dowodów założeniom odpowiadają zmienne (dowodowe).

```
type expr =
  ...
  | EVar of prf_var   (* x *)
  ...
```

$$\frac{\Gamma(x) = \varphi}{\Delta; \Gamma \vdash x \,:\, \varphi}$$

Implikacja

Spójnik implikacji zachowuje się jak przestrzeń funkcyjna i przychodzi wraz z dwiema regułami wnioskowania, które odpowiadają które odpowiadają funkcji anonimowej oraz aplikacji funkcji.

```
type formula =
  ...
  | Imp of formula * formula  (* φ → ψ *)
  ...

type expr =
  ...
```

```
  | EFun of prf_var * formula * expr  (* fun (x:φ) -> e *)
  | EApp of expr * expr               (* e₁ e₂ *)
  ...
```

$$\frac{\Delta \vdash \mathbf{wff}(\varphi) \quad \Delta \vdash \mathbf{wff}(\psi)}{\Delta \vdash \mathbf{wff}(\varphi \to \psi)} \qquad \frac{\Delta \vdash \mathbf{wff}(\varphi) \quad \Delta;\Gamma[x \mapsto \varphi] \vdash e \ :\ \psi}{\Delta;\Gamma \vdash \texttt{fun}\ (x\!:\!\varphi)\ \texttt{->}\ e \ :\ \varphi \to \psi}$$

$$\frac{\Delta;\Gamma \vdash e_1 \ :\ \varphi \to \psi \quad \Delta;\Gamma \vdash e_2 \ :\ \varphi}{\Delta;\Gamma \vdash e_1\ e_2 \ :\ \psi}$$

Zwróć uwagę, że w regule wprowadzania implikacji pozwalamy na przesłanianie zmiennych dowodowych.

Koniunkcja

Spójnik koniunkcji odpowiada produktowi kartezjańskiemu, a jego reguły wnioskowania odpowiadają konstruktorowi par oraz projekcjom.

```
type formula =
  ...
  | And of formula * formula  (* φ ∧ ψ *)
  ...

type expr =
  ...
  | EPair of expr * expr      (* (e₁, e₂) *)
  | EFst  of expr             (* π₁ e *)
  | ESnd  of expr             (* π₂ e *)
  ...
```

$$\frac{\Delta \vdash \mathbf{wff}(\varphi) \quad \Delta \vdash \mathbf{wff}(\psi)}{\Delta \vdash \mathbf{wff}(\varphi \land \psi)} \qquad \frac{\Delta;\Gamma \vdash e_1 \ :\ \varphi \quad \Delta;\Gamma \vdash e_2 \ :\ \psi}{\Delta;\Gamma \vdash (e_1, e_2) \ :\ \varphi \land \psi}$$

$$\frac{\Delta;\Gamma \vdash e \ :\ \varphi \land \psi}{\Delta;\Gamma \vdash \pi_1\ e \ :\ \varphi} \qquad \frac{\Delta;\Gamma \vdash e \ :\ \varphi \land \psi}{\Delta;\Gamma \vdash \pi_2\ e \ :\ \psi}$$

Alternatywa

Spójnik alternatywy to odpowiednik typu `Either.t`. Ma dwa konstruktory (**`left`** i **`right`**), a eliminuje się go poprzez dopasowanie wzorca.

```
type formula =
  ...
```

```
  | Or of formula * formula  (* φ ∨ ψ *)
  ...

type expr =
  ...
  | ELeft  of expr * formula (* left  e : φ *)
  | ERight of expr * formula (* right e : φ *)
  | ECase  of expr * prf_var * expr * prf_var * expr
         (* case e of left x -> e₁ | right y -> e₂ *)
  ...
```

Zauważ, że konstruktory `left` i `right` jako parametr przyjmują formułę, która jest typem całego wyrażenia. Jest to potrzebne, ponieważ z typów podwyrażeń nie sposób uzyskać pełnej informacji o typie całego wyrażenia.

$$\frac{\Delta \vdash \mathbf{wff}(\varphi) \quad \Delta \vdash \mathbf{wff}(\psi)}{\Delta \vdash \mathbf{wff}(\varphi \vee \psi)}$$

$$\frac{\Delta \vdash \mathbf{wff}(\varphi) \quad \varphi = \varphi_1 \vee \varphi_2 \quad \Delta; \Gamma \vdash e \ : \ \varphi_1}{\Delta; \Gamma \vdash (\texttt{left}\ e\ :\ \varphi)\ :\ \varphi}$$

$$\frac{\Delta \vdash \mathbf{wff}(\varphi) \quad \varphi = \varphi_1 \vee \varphi_2 \quad \Delta; \Gamma \vdash e \ : \ \varphi_2}{\Delta; \Gamma \vdash (\texttt{right}\ e\ :\ \varphi)\ :\ \varphi}$$

$$\frac{\Delta; \Gamma \vdash e\ :\ \varphi_1 \vee \varphi_2 \quad \Delta; \Gamma[x \mapsto \varphi_1] \vdash e_1\ :\ \psi \quad \Delta; \Gamma[y \mapsto \varphi_2] \vdash e_2\ :\ \psi}{\Delta; \Gamma \vdash (\texttt{case}\ e\ \texttt{of left}\ x\ \texttt{->}\ e_1\ \texttt{|}\ \texttt{right}\ y\ \texttt{->}\ e_2)\ :\ \psi}$$

Podobnie jak w przypadku implikacji, reguła eliminacji alternatywy pozwala na przesłanianie zmiennych dowodowych.

Fałsz

Spójnik fałszu jest typem, który nie ma żadnych wartości. Nie ma on reguły wprowadzania, ale ma regułę eliminacji, która pozwala na dowodzenie dowolnej formuły.

```
type formula =
  ...
  | False  (* ⊥ *)
  ...

type expr =
```

```
  ...
  | EAbsurd of expr * formula  (* absurd e : φ *)
  ...
```

$$\frac{}{\Delta \vdash \mathbf{wff}(\bot)} \qquad \frac{\Delta; \Gamma \vdash e \;:\; \bot \quad \Delta \vdash \mathbf{wff}(\varphi)}{\Delta; \Gamma \vdash (\texttt{absurd}\ e \;:\; \varphi) \;:\; \varphi}$$

Kwantyfikator uniwersalny

Kwantyfikator uniwersalny jest typem funkcji, która przyjmuje term jako argument, a typ wyniku zależy od tego termu. W OCamlu nie ma dobrego odpowiednika takiego typu, ale z zależnością typu wyniku od wartości argumentu już się zetknęliśmy przy okazji funktorów. Reguły wprowadzania i eliminacji kwantyfikatora uniwersalnego odpowiadają funkcji anonimowej oraz aplikacji funkcji.

```
type formula =
  ...
  | Forall of term_var * formula  (* ∀x, φ *)
  ...

type expr =
  ...
  | ETermFun of term_var * expr (* fun [x] -> e *)
  | ETermApp of expr * term     (* e [t] *)
  ...
```

$$\frac{\Delta \cup \{x\} \vdash \mathbf{wff}(\varphi)}{\Delta \vdash \mathbf{wff}(\forall x,\ \varphi)} \qquad \frac{x \notin \Delta \quad \Delta \cup \{x\}; \Gamma \vdash e \;:\; \varphi}{\Delta; \Gamma \vdash \texttt{fun [}x\texttt{] ->}\ e \;:\; \forall x,\ \varphi}$$

$$\frac{\Delta; \Gamma \vdash e \;:\; \forall x,\ \varphi \quad \Delta \vdash \mathbf{wft}(t)}{\Delta; \Gamma \vdash e\ [t] \;:\; \varphi\{x \mapsto t\}}$$

Zapis $\varphi\{x \mapsto t\}$ oznacza podstawienie termu t za zmienną x w formule φ. Zauważ, że reguła dobrego sformowania kwantyfikatora (pierwsza reguła) dopuszcza przesłanianie zmiennych, natomiast reguła wprowadzania kwantyfikatora (druga reguła) wymaga, by zmienna termowa x była odpowiednio świeża. Zastanów się, dlaczego jest to konieczne.

Kwantyfikator egzystencjalny

Kwantyfikator egzystencjalny jest typem par złożonych z termu i dowodu. Jest to para zależna, tzn. typ dowodu zależy od termu. Kwantyfikator egzystencjalny eliminuje się poprzez dopasowanie wzorca.

```
type formula =
  ...
  | Exists of term_var * formula  (* ∃x, φ *)
  ...

type expr =
  ...
  | EPack   of term * expr * formula (* pack [t] e : φ *)
  | EUnpack of term_var * prf_var * expr * expr
            (* unpack [x] y from e₁ in e₂ *)
  ...
```

$$\frac{\Delta \cup \{x\} \vdash \mathbf{wff}(\varphi)}{\Delta \vdash \mathbf{wff}(\exists x,\ \varphi)}$$

$$\frac{\Delta \vdash \mathbf{wft}(t) \quad \Delta \vdash \mathbf{wff}(\varphi) \quad \varphi = \exists x,\ \psi \quad \Delta; \Gamma \vdash e\ :\ \psi\{x \mapsto t\}}{\Delta; \Gamma \vdash (\texttt{pack [}t\texttt{]}\ e\ :\ \varphi)\ :\ \varphi}$$

$$\frac{\Delta; \Gamma \vdash e_1\ :\ \exists x,\ \varphi \quad x \notin \Delta \quad \Delta \cup \{x\}; \Gamma[y \mapsto \varphi] \vdash e_2\ :\ \psi \quad \Delta \vdash \mathbf{wff}(\psi)}{\Delta; \Gamma \vdash (\texttt{unpack [}x\texttt{]}\ y\ \texttt{from}\ e_1\ \texttt{in}\ e_2)\ :\ \psi}$$

W ostatniej regule warunek $\Delta \vdash \mathbf{wff}(\psi)$ jest potrzebny, by zapewnić, że zmienna termowa x nie ucieka poza swój zakres. Można pokazać, że jeśli ten warunek zamienimy na słabszy, który mówi, że zmienna x nie ma wolnego wystąpienia w formule ψ, otrzymamy równoważną definicję.

Kwantyfikator drugiego rzędu

Logika pierwszego rzędu jest zbyt słaba, by dało się w skończony sposób zaksjomatyzować arytmetykę liczb naturalnych. Problematyczna jest zasada indukcji, która powinna być prawdziwa dla dowolnej własności (predykatu unarnego). By temu zaradzić, rozszerzymy naszą logikę o kwantyfikator uniwersalny drugiego rzędu, który będzie kwantyfikował nad predykatami unarnymi. Tego kwantyfikatora będziemy używać tylko w aksjomatach, więc wystarczy nam tylko reguła eliminacji[1]. Jak można się domyślić, będzie miała ona postać aplikacji funkcji.

```
type formula =
  ...
  | ForallRel of rel_symbol * formula  (* ∀{R}, φ *)
  ...

type expr =
  ...
  | ERelApp of expr * term_var * formula (* e {x|φ} *)
  ...
```

$$\frac{\Delta \vdash \mathbf{wff}(\varphi)}{\Delta \vdash \mathbf{wff}(\forall\{R\},\ \varphi)} \qquad \frac{\Delta;\Gamma \vdash e\ :\ \forall\{R\},\ \varphi \quad \Delta \cup \{x\} \vdash \mathbf{wft}(\psi)}{\Delta;\Gamma \vdash e\,\{x\,|\,\psi\}\ :\ \varphi\{R \mapsto \{x\,|\,\psi\}\}}$$

Aksjomaty i twierdzenia

Wejściem weryfikatora jest lista definicji, gdzie definicja jest aksjomatem lub twierdzeniem.

```
type def =
  | Axiom   of position * prf_var * formula
            (* axiom x : φ *)
  | Theorem of position * prf_var * formula * expr
            (* theorem x : φ proof e qed *)
```

Weryfikator powinien zaczynając od pustego środowiska kolejno sprawdzać definicje, rozszerzając środowisko o kolejne zmienne. Proces ten można opisać formalnie następującymi regułami.

$$\frac{\varnothing \vdash \mathbf{wff}(\varphi)}{\Gamma \vdash \texttt{axiom}\ x\ :\ \varphi \rightsquigarrow \Gamma[x \mapsto \varphi]}$$

$$\frac{\varnothing \vdash \mathbf{wff}(\varphi) \quad \varnothing;\Gamma \vdash e\ :\ \varphi}{\Gamma \vdash \texttt{theorem}\ x\ :\ \varphi\ \texttt{proof}\ e\ \texttt{qed} \rightsquigarrow \Gamma[x \mapsto \varphi]}$$

$$\frac{\varnothing \vdash d_0 \rightsquigarrow \Gamma_0 \quad \Gamma_0 \vdash d_1 \rightsquigarrow \Gamma_1 \quad \ldots \quad \Gamma_{n-1} \vdash d_n \rightsquigarrow \Gamma_n}{\vdash d_0\ d_1\ \ldots\ d_n}$$

[1] By poprawnie zaimplementować regułę wprowadzania, trzeba by podnieść symbole relacyjne do rangi zmiennych nowego rodzaju, co niepotrzebnie by skomplikowało implementację.

Zadanie

Celem zadania jest zaimplementowanie funkcji `check_defs : def list -> unit` znajdującej się w pliku `typeCheck.ml`. Funkcja ta powinna kończyć się bez błędu, jeśli podana lista definicji jest poprawna, lub zgłaszać wyjątek `Syntax.Type_error` w przypadku błędu. Wyjątek ten przyjmuje dwa argumenty: pozycję błędu oraz komunikat. Pozycja powinna wskazywać na miejsce błędu, które można pozyskać z konstruktora definicji lub wyrażenia. Jeżeli błąd znajduje się wewnątrz formuły albo termu, należy użyć pozycji definicji lub wyrażenia w którym formuła lub term się znajduje. Komunikaty błędów nie będą sprawdzane przez testy, ale powinny być zrozumiałe i pomocne. Można korzystać z funkcji znajdujących się w module `Syntax`. W szczególności znajduje się tam implementacja podstawień, α-równoważności oraz wypisywania formuł i termów.