Progetto di Linguaggi di Programmazione

Anno Accademico 2017/2018

Gruppi

Il progetto deve essere fatto da gruppi di 3 - 4 persone. Ogni gruppo ha un *responsabile* che è anche colui/colei che deve comunicarmi, via email, i nomi dei componenti del gruppo. Ad ogni gruppo sarà attribuito un identificatore. Verrà creata anche una mailing list per far si che le varie comunicazioni possano arrivare a tutti. Il termine per la formazione dei gruppi è il 30 dicembre.

Consegna del progetto

Il termine per la consegna del progetto è il **4 febbraio** e non è prorogabile.

Il progetto deve essere contenuto in un tre file con estensione `.ml`: il primo deve chiamarsi `typing###.ml`, il secondo `eval###.ml` ed il terzo `evaltry###.ml` dove `###` è l'identificatore del gruppo. Potete consegnare anche una relazione illustrativa delle vostre scelte progettuali in formato `pdf`. I tre o quattro file devono essere contenuti in una cartella zippata con come nome della cartella il nome del gruppo, che deve essere consegnata dal responsabile del gruppo.

Il file che contiene il progetto non deve contenere le prove che le funzioni definite sono corrette.

Correzione

La correzione avviene in questo modo. Se un file non compila con `ocamlc` o il tipo delle funzioni da definire è diverso da quello richiesto allora quella parte viene valutata 0 punti. Se invece compila ed i tipi sono corretti verranno fatte un certo numero di test ed il punteggio viene determinato dal numero di test superati, quindi se li si supera tutti si hanno tutti i punti per la parte, se se ne supera il 50% la metà dei punti, secondo una scala sostanzialmente lineare.

I punti sono delle tre parti sono come segue: la prima parte vale 1,5 punti, la seconda 3,5 punti e la terza 1 punto.

Il linguaggio **Fun**

Considerate il linguaggio di programmazione, il cui scope è statico (il linguaggio riflette lo scope statico quando presenta esplicitamente un costrutto per gestire la ricorsione), e la cui sintassi è la seguente:

$$
\begin{aligned}
t \quad ::= \quad & x \mid n \mid c \mid \mathit{true} \mid \mathit{false} \mid \mathit{empty} \mid t_1 + t_2 \mid t_1 - t_2 \mid t_1 \times t_2 \mid t_1 \wedge t_2 \mid t_1 \vee t_2 \mid \neg t_1 \mid t_1 = t_2 \\
& \mid t_1 < t_2 \mid \mathit{cons}(t_1, t_2) \mid \mathit{head}(t) \mid \mathit{tail}(t) \mid \mathit{fst}(t) \mid \mathit{snd}(t) \mid \mathit{pair}(t_1, t_2) \mid \\
& \mid \mathit{if}\ t_0\ \mathit{then}\ t_1\ \mathit{else}\ t_2 \mid \mathit{fun}(x, t) \mid \mathit{apply}(t_1, t_2) \mid \mathit{let}\ x \Leftarrow t_1\ \mathit{in}\ t_2 \mid \mathit{rec}\ y.(\mathit{fun}(x, t))
\end{aligned}
$$

dove n sono interi e c caratteri, *empty* indica la lista vuota, *cons*, *head* e *tail* sono le ovvie operazioni sulle liste (costruzione, selezione del primo elemento e del resto di una lista), *pair* è il costruttore delle coppie e avete anche gli ovvi selettori (*fst* e *snd*), e le altre parti sono standard. Il predicato di uguaglianza è polimorfo, quindi lo potete usare in qualsiasi contesto abbia senso usarlo.

Considerate inoltre la seguente grammatica per i tipi

$$\tau ::= \ a \mid \mathsf{int} \mid \mathsf{bool} \mid \mathsf{char} \mid \tau \ \mathsf{list} \mid \tau_1 * \tau_2 \mid \tau_1 \to \tau_2$$

dove $a \in \mathsf{A}$ sono le variabili di tipo e le seguenti regole descrivono come si fa ad attribuire un tipo ad un termine.

$$\frac{}{x : type(x)} \qquad \frac{}{n : \mathsf{int}} \qquad \frac{}{c : \mathsf{char}} \qquad \frac{t_1 : \mathsf{int} \quad t_2 : \mathsf{int}}{t_1 \ op \ t_2 : \mathsf{int}} \ op \in \{+, -, \times\}$$

$$\frac{}{true : \mathsf{bool}} \qquad \frac{}{false : \mathsf{bool}} \qquad \frac{t_1 : \mathsf{bool} \quad t_2 : \mathsf{bool}}{t_1 \ op \ t_2 : \mathsf{bool}} \ op \in \{\wedge, \vee\} \qquad \frac{t_1 : \tau \quad t_2 : \tau}{t_1 \ = \ t_2 : \mathsf{bool}}$$

$$\frac{t : \mathsf{bool}}{\neg t : \mathsf{bool}} \qquad \frac{t_1 : \mathsf{int} \quad t_2 : \mathsf{int}}{t_1 < t_2 : \mathsf{bool}} \qquad \frac{t_1 : \tau_1 \quad t_2 : \tau_2}{pair(t_1, t_2) : \tau_1 * \tau_2}$$

$$\frac{t : \tau_1 * \tau_2}{fst(t) : \tau_1} \qquad \frac{t : \tau_1 * \tau_2}{snd(t) : \tau_2} \qquad \frac{}{empty : \tau \ \mathsf{list}} \qquad \frac{t_1 : \tau \quad t_2 : \tau \ \mathsf{list}}{cons(t_1, t_2) : \tau \ \mathsf{list}} \qquad \frac{t : \tau \ \mathsf{list}}{head(t) : \tau}$$

$$\frac{t : \tau \ \mathsf{list}}{tail(t) : \tau} \qquad \frac{t_0 : \mathsf{bool} \quad t_1 : \tau \quad t_2 : \tau}{if \ t_0 \ then \ t_1 \ else \ t_2 : \tau} \qquad \frac{x : \tau_1 \quad t : \tau_2}{fun(x, t) : \tau_1 \to \tau_2}$$

$$\frac{t_1 : \tau_1 \to \tau_2 \quad t_2 : \tau_1}{apply(t_1, t_2) : \tau_2} \qquad \frac{x : \tau_1 \quad t_1 : \tau_1 \quad t_2 : \tau_2}{let \ x \Leftarrow t_1 \ in \ t_2 : \tau_2} \qquad \frac{y : \tau \quad fun(x, t) : \tau}{rec \ y.(fun(x, t)) : \tau}$$

Assumiamo sempre che i termini che dobbiamo valutare siano *ben tipati*, nel senso che riusciamo sempre ad attribuire loro un tipo (che può contenere variabili di tipo).

Inferenza di tipo

L'algoritmo di inferenza di tipo usa le regole per l'attribuzione del tipo elencate prima per costruire un insieme di vincoli che poi dovrà essere risolto. I vincoli vengono costruiti nel seguente modo. Dato un termine t e la funzione *type* che associa ad ogni variabile del linguaggio un tipo τ della grammatica dei tipi, generiamo una coppia formata dal tipo di t e da un insieme di vincoli, quindi riscriviamo le regole in modo più operativo. La funzione $\mathsf{newvar}()$ *genera* una nuova variabile di tipo.

$$\frac{}{\langle\langle x, type\rangle\rangle \Rightarrow (type(x), \emptyset)} \qquad \frac{}{\langle\langle n, type\rangle\rangle \Rightarrow (\mathsf{int}, \emptyset)} \qquad \frac{}{\langle\langle c, type\rangle\rangle \Rightarrow (\mathsf{char}, \emptyset)}$$

$$\frac{\langle\langle t_1, type\rangle\rangle \Rightarrow (\mathsf{int}, T_1) \quad \langle\langle t_2, type\rangle\rangle \Rightarrow (\mathsf{int}, T_2)}{\langle\langle t_1 \ op \ t_2, type\rangle\rangle \Rightarrow (\mathsf{int}, \{(t_1, \mathsf{int}), (t_2, \mathsf{int})\} \cup T_1 \cup T_2)} \ op \in \{+, -, \times\} \qquad \frac{}{\langle\langle true, type\rangle\rangle \Rightarrow (\mathsf{bool}, \emptyset)}$$

$$\frac{\langle\langle t_1, type\rangle\rangle \Rightarrow (\mathsf{bool}, T_1) \quad \langle\langle t_2, type\rangle\rangle \Rightarrow (\mathsf{bool}, T_2)}{\langle\langle t_1 \ op \ t_2, type\rangle\rangle \Rightarrow (\mathsf{bool}, \{(t_1, \mathsf{bool}), (t_2, \mathsf{bool})\} \cup T_1 \cup T_2)} \ op \in \{\wedge, \vee\} \qquad \frac{}{\langle\langle false, type\rangle\rangle \Rightarrow (\mathsf{bool}, \emptyset)}$$

$$\frac{\langle\!\langle t_1, type\rangle\!\rangle \Rightarrow (\mathsf{int}, T_1) \quad \langle\!\langle t_2, type\rangle\!\rangle \Rightarrow (\mathsf{int}, T_2)}{\langle\!\langle t_1 < t_2, type\rangle\!\rangle \Rightarrow (\mathsf{bool}, \{(t_1, \mathsf{int}), (t_2, \mathsf{int})\} \cup T_1 \cup T_2)} \qquad \frac{\langle\!\langle t, type\rangle\!\rangle \Rightarrow (\mathsf{bool}, T)}{\langle\!\langle \neg t, type\rangle\!\rangle \Rightarrow (\mathsf{bool}, T)}$$

$$\frac{\langle\!\langle t_1, type\rangle\!\rangle \Rightarrow (\tau_1, T_1) \quad \langle\!\langle t_2, type\rangle\!\rangle \Rightarrow (\tau_2, T_2)}{\langle\!\langle t_1 = t_2, type\rangle\!\rangle \Rightarrow (\mathsf{bool}, \{(t_1, \tau_1), (t_2, \tau_2)\} \cup T_1 \cup T_2)}$$

$$\frac{\langle\!\langle t_1, type\rangle\!\rangle \Rightarrow (\tau_1, T_1) \quad \langle\!\langle t_2, type\rangle\!\rangle \Rightarrow (\tau_2, T_2)}{\langle\!\langle pair(t_1, t_2), type\rangle\!\rangle \Rightarrow (\tau_1 * \tau_2, \{(t_1, \tau_1), (t_2, \tau_2)\} \cup T_1 \cup T_2)}$$

$$\frac{\langle\!\langle t, type\rangle\!\rangle \Rightarrow (\tau_1 * \tau_2, T)}{\langle\!\langle fst(t), type\rangle\!\rangle \Rightarrow (\tau_1, \{(t, \tau_1 * \tau_2\} \cup T)} \qquad \frac{\langle\!\langle t, type\rangle\!\rangle \Rightarrow (\tau_1 * \tau_2, T)}{\langle\!\langle snd(t), type\rangle\!\rangle \Rightarrow (\tau_2, \{(t, \tau_1 * \tau_2\} \cup T)}$$

$$\frac{a = \mathsf{newvar}()}{\langle\!\langle empty, type\rangle\!\rangle \Rightarrow (a\ \mathsf{list}, \emptyset)} \qquad \frac{\langle\!\langle t_1, type\rangle\!\rangle \Rightarrow (\tau, T_1) \quad \langle\!\langle t_2, type\rangle\!\rangle \Rightarrow (\tau\ list, T_2)}{\langle\!\langle cons(t_1, t_2), type\rangle\!\rangle \Rightarrow (\tau\ list, \{(t_1, \tau), (t_2, \tau\ list)\} \cup T_1 \cup T_2)}$$

$$\frac{\langle\!\langle t, type\rangle\!\rangle \Rightarrow (\tau\ list, T)}{\langle\!\langle head(t), type\rangle\!\rangle \Rightarrow (\tau, \{(t, \tau list)\} \cup T)} \qquad \frac{\langle\!\langle t, type\rangle\!\rangle \Rightarrow (\tau\ list, T)}{\langle\!\langle tail(t), type\rangle\!\rangle \Rightarrow (\tau\ list, T)}$$

$$\frac{\langle\!\langle t_0, type\rangle\!\rangle \Rightarrow (\mathsf{bool}, T_0) \quad \langle\!\langle t_1, type\rangle\!\rangle \Rightarrow (\tau, T_1) \quad \langle\!\langle t_2, type\rangle\!\rangle \Rightarrow (\tau, T_2)}{\langle\!\langle if\ t_0\ then\ t_1\ else\ t_2, type\rangle\!\rangle \Rightarrow (\tau, \{(t_0, \mathsf{bool}), (t_1, t_2)\} \cup T_0 \cup T_1 \cup T_2)}$$

$$\frac{a = \mathsf{newvar}() \quad \langle\!\langle t, type[x/a]\rangle\!\rangle \Rightarrow (\tau_2, T)}{\langle\!\langle fun(x, t), type\rangle\!\rangle \Rightarrow (a \to \tau_2, T)}$$

$$\frac{a = \mathsf{newvar}() \quad \langle\!\langle t_1, type\rangle\!\rangle \Rightarrow (\tau_1 \to a, T_2) \quad \langle\!\langle t_2, type\rangle\!\rangle \Rightarrow (\tau_1, T_1)}{\langle\!\langle apply(t_1, t_2), type\rangle\!\rangle \Rightarrow (a, \{(t_1, \tau_1 \to a)\} \cup T_1 \cup T_2)}$$

$$\frac{a = \mathsf{newvar}() \quad \langle\!\langle t_1, type\rangle\!\rangle \Rightarrow (a, T_1) \quad \langle\!\langle t_2, type[x/a]\rangle\!\rangle \Rightarrow (\tau_2, T_2)}{\langle\!\langle let\ x \Leftarrow t_1\ in\ t_2, type\rangle\!\rangle \Rightarrow (\tau_2, \{(t_1, a)\} \cup T_1 \cup T_2)}$$

$$\frac{a = \mathsf{newvar}() \quad \langle\!\langle t, type[y/a]\rangle\!\rangle \Rightarrow (a, T)}{\langle\!\langle rec\ y.(fun(x, t)), type\rangle\!\rangle \Rightarrow (\tau, \{(y, a)\} \cup T)}$$

Una volta ottenuto l'insieme di vincoli, questi vanno *risolti* tramite un algoritmo che si basa sulla *sostituzione* e sull'*unificazione*. Una sostituzione è una funzione che associa ad ogni variabile di tipo un tipo (con eventuali variabili). Ovviamente l'identità è una sostituzione (ogni variabile di tipo viene *mandata* in se stessa) e le sostituzioni possono venire *composte*. Dato un vincolo, che è una coppia, si cerca la sostituzione che rende i due elementi della coppia uguali. Quindi, dato un insieme C di coppie, per trovare la sostituzione corretta si procede come segue, partendo dalla sostituzione *identica*. La sostituzione la chiamiamo s.

- se la coppia è fatta da tue termini uguali, la si elimina
- se la coppia è fatta da una variabile a ed un termine q in cui tale variabile non occorre allora si aggiorna la sostituzione s ponendo che $s(a) = q$ e si applica tale sostituzione a tutte le coppie in C

- se la coppia è fatta da due termini composti da costruttori (nel nostro caso coppie, liste e frecce) allora si verifica che i costruttori più esterni siano uguali, e se lo sono si aggiungono le coppie fatte dai componenti accoppiati secondo le regole dell'equivalenza e si elimina la coppia di partenza
- si ripetono i passi fino a che non si ottiene che C è vuoto (ed allora abbiamo la sostituzione cercata) oppure se si incontra una coppia che non è unificabile

Per chiarire, alcuni esempi: $\{(\mathsf{int}, \mathsf{int})\}$ unifica con la sostituzione identica, $\{(a, \mathsf{int})\}$ unifica con la sostituzione $s(a) = \mathsf{int}$, $\{(\mathsf{int}, a \to b)\}$ non unifica mentre $\{(a \to \mathsf{int}, \mathsf{bool} \to b)\}$ unifica con la sostituzione $s(a) = \mathsf{bool}$ e $s(b) = \mathsf{int}$. Osservate che da $\{(a \to \mathsf{int}, \mathsf{bool} \to b)\}$ si sono generati $\{(a, \mathsf{bool}), (\mathsf{int}, b)\}$.

Valutazione di un termine di Fun

La valutazione di un termine t in un ambiente δ, restituisce un *valore* in un insieme $\mathcal{V}$. Tale insieme è il più piccolo insieme che contiene gli interi, i booleani, i caratteri, le astrazioni funzionali chiuse, le liste finite di valori e le coppie di valori. Un'astrazione funzionale chiusa è una chiusura, quindi una coppia fatta da un'espressione $fun(x, t)$ e da un ambiente δ_f dove $FV(t) \subseteq \{x\} \cup dom(\delta_f)$.

L'insieme delle *free variables* del termine T, $FV(t)$, è definito per induzione strutturare sui termini nel seguente modo:

$$FV(t) = \begin{cases} \{x\} & \text{se } t \text{ è } x \\ \emptyset & \text{se } t \text{ è } n, c, true, false, empty \\ FV(t_1) \cup FV(t_2) & \text{se } t \text{ è } t_1 + t_2,\ t_1 - t_2,\ t_1 \times t_2, t_1 \wedge t_2, t_1 \vee t_2,\ t_1 = t_2, \\ & \quad t_1 < t_2, cons(t_1, t_2),\ pair(t_1, t_2), apply(t_1, t_2) \\ FV(t') & \text{se } t \text{ è } head(t'),\ \ tail(t'),\ fst(t'),\ snd(t') \\ FV(t') \setminus \{x\} & \text{se } t \text{ è } fun(x, t'),\ rec\ y.(t') \\ (FV(t_2) \setminus \{x\}) \cup FV(t_1) & \text{se } t \text{ è } let\ x \Leftarrow t_1\ in\ t_2 \\ FV(t_0) \cup FV(t_1) \cup FV(t_2) & \text{se } t \text{ è } if\ t_0\ then\ t_1\ else\ t_2 \end{cases}$$

mentre, data una funzione parziale h da un insieme A ad un insieme B (parziale significa che non è detto sia definita per ogni valore di $a \in A$), con $dom(h)$ indichiamo l'insieme dei valori di A per cui la funzione è definita. Se indichiamo con $h(a)\downarrow$ il fatto che la funzione è definita per a, cioè che esiste un $b \in B$ tale che $h(a) = b$ abbiamo che $dom(h) = \{a \in A \mid h(a)\downarrow\}$. Quindi, indicando con $\mathbb{N}$ gli interi, $\mathbb{B}$ i booleani, $\mathcal{C}$ i caratteri, $\mathcal{F}$ le chiusure, $\mathcal{L}$ le liste finite di valori (con la lista vuota indicata come $[\,]$ mentre se $\mathbf{v}$ è un valore e $\mathbf{v}'$ una lista, $\mathbf{v} :: \mathbf{v}'$ è una lista, e per le liste, quando necessario, si usa la notazione `ocaml`) e $\mathcal{P}$ le coppie di valori, abbiamo che $\mathcal{V} = \mathbb{N} \cup \mathbb{B} \cup \mathcal{C} \cup \mathcal{F} \cup \mathcal{L} \cup \mathcal{P}$ e questo insieme è ben definito dato che i sottoinsiemi che lo compongono sono disgiunti a due a due.

La semantica del linguaggio è data dalle seguenti regole, dove δ è l'ambiente che è una funzione parziale che associa agli identificatori un valore in $\mathcal{V}$, e $\delta[x/\mathbf{v}]$ è la funzione ottenuta da δ così definita: $\delta[x/\mathbf{v}](x) = \mathbf{v}$ e $\delta[x/\mathbf{v}](y) = \delta(y)$, mentre $t\{[x/t']\}$ è un termine del linguaggio ottenuto sostituendo ogni occorrenza di x in t con il termine t'.

$$\frac{}{\langle x, \delta\rangle \to \delta(x)} \qquad \frac{}{\langle n, \delta\rangle \to \mathbf{n}} \qquad \frac{}{\langle c, \delta\rangle \to \mathbf{c}} \qquad \frac{}{\langle true, \delta\rangle \to \mathbf{true}} \qquad \frac{}{\langle false, \delta\rangle \to \mathbf{false}}$$

$$\frac{}{\langle empty, \delta\rangle \to [\,]} \qquad \frac{\langle t_1, \delta\rangle \to \mathbf{v}_1 \quad \langle t_2, \delta\rangle \to \mathbf{v}_2}{\langle t_1\ op\ t_2, \delta\rangle \to \mathbf{v}_1\ \mathbf{op}\ \mathbf{v}_2}\ \mathbf{op}, op \in \{+, -, \times, \wedge, \vee\} \qquad \frac{\langle t, \delta\rangle \to \mathbf{v}}{\langle \neg t, \delta\rangle \to \neg\mathbf{v}}$$

$$\frac{\langle t_1, \delta\rangle \to \mathbf{v} \quad \langle t_2, \delta\rangle \to \mathbf{v}}{\langle t_1\ =\ t_2, \delta\rangle \to \mathbf{true}}\ \mathbf{v} \equiv \mathbf{v}' \qquad \frac{\langle t_1, \delta\rangle \to \mathbf{v} \quad \langle t_2, \delta\rangle \to \mathbf{v}'}{\langle t_1\ =\ t_2, \delta\rangle \to \mathbf{false}}\ \mathbf{v} \not\equiv \mathbf{v}'$$

$$\frac{\langle t_1,\delta\rangle \to \mathbf{v}_1 \quad \langle t_2,\delta\rangle \to \mathbf{v}_2}{\langle cons(t_1,t_2),\delta\rangle \to \mathbf{v}_1 :: \mathbf{v}_2} \qquad \frac{\langle t,\delta\rangle \to \mathbf{v} :: \mathbf{v}'}{\langle head(t),\delta\rangle \to \mathbf{v}} \qquad \frac{\langle t,\delta\rangle \to \mathbf{v} :: \mathbf{v}'}{\langle tail(t),\delta\rangle \to \mathbf{v}'}$$

$$\frac{\langle t_1,\delta\rangle \to \mathbf{v}_1 \quad \langle t_2,\delta\rangle \to \mathbf{v}_2}{\langle pair(t_1,t_2),\delta\rangle \to (\mathbf{v}_1,\mathbf{v}_2)} \qquad \frac{\langle t,\delta\rangle \to (\mathbf{v}_1,\mathbf{v}_2)}{\langle fst(t),\delta\rangle \to \mathbf{v}_1} \qquad \frac{\langle t,\delta\rangle \to (\mathbf{v}_1,\mathbf{v}_2)}{\langle snd(t),\delta\rangle \to \mathbf{v}_2}$$

$$\frac{\langle t_0,\delta\rangle \to \mathbf{true} \quad \langle\langle t_1,\delta\rangle \to \mathbf{v}}{\langle if\ t_0\ then\ t_1\ else\ t_2,\delta\rangle \to \mathbf{v}} \qquad \frac{\langle t_0,\delta\rangle \to \mathbf{false} \quad \langle\langle t_2,\delta\rangle \to \mathbf{v}}{\langle if\ t_0\ then\ t_1\ else\ t_2,\delta\rangle \to \mathbf{v}}$$

$$\frac{}{\langle fun(x,t),\delta\rangle \to (fun(x,t),\delta)} \qquad \frac{\langle t_1,\delta\rangle \to (fun(x,t),\delta') \quad \langle t_2,\delta\rangle \to \mathbf{v}_2 \quad \langle t,\delta[x/\mathbf{v}_2]\rangle \to v}{\langle apply(t_1,t_2),\delta\rangle \to \mathbf{v}}$$

$$\frac{\langle t_1,\delta\rangle \to \mathbf{v}_1 \quad \langle t_2,\delta[x/\mathbf{v}_1\rangle \to \mathbf{v}}{\langle let\ x \Leftarrow t_1\ in\ t_2,\delta\rangle \to \mathbf{v}} \qquad \frac{}{\langle rec\ y.(fun(x,t)),\delta\rangle \to \langle fun(x,t\{[y/rec\ y.(fun(x,t))]\}),\delta\rangle}$$

Il progetto

Il progetto si divide in tre parti.

Nella prima parte dovete scrivere una funzione in `Ocaml` che realizza la *type inference* che si chiama `typeinf` e che ha tipo `exp -> etype`. Il sistema di tipi illustrato serve per generare i vincoli che risolvete con l'unificazione descritta precedentemente.

Nella seconda parte dovere scrivere la funzione `sem` che valuta un'espressione e che ha tipo `exp -> env -> eval`. Deve implementare le regole per la semantica illustrate prima.

La terza parte progetto riguarda invece una modifica al linguaggio. Dovete aggiungere alla sintassi astratta le seguenti linee

```
| Try exp * ide * exp
| Raise ide
```

che potete vedere come un "try and catch". L'eccezione è lanciata con il `Raise ide` ed è trattata dal `Try` che ha il corretto identificatore. Quindi `Try (e1, Ide "x", e2)` è tale che se all'interno di `e1` viene eseguito `Raise Ide "x"` questo viene trattato dal gestore `e2`.

Il nuovo interprete si deve chiamare `semtry` ed ha lo stesso tipo di `sem`.

Di seguito sono fornite tutte le parti del progetto che devono tassativamente avere quella forma.

I tipi in `ocaml`

La sintassi astratta del linguaggio è la seguente:

```
type exp =                          | Eq of exp * exp
      Val of ide                    | Less of exp * exp
    | Eint of int                   | Cons of exp * exp
    | Echar of char                 | Head of exp
    | True                          | Tail of exp
    | False                         | Fst of exp
    | Empty                         | Snd of exp
    | Sum of exp * exp              | Pair of exp * exp
    | Diff of exp * exp             | Ifthenelse of exp * exp * exp
    | Times of exp * exp            | Let of ide * exp * exp
    | And of exp * exp              | Fun of ide * exp
    | Or of exp * exp               | Appl of exp * exp
    | Not of exp                    | Rec of ide * exp
```

mentre `ide` è il tipo `type ide = Ide of string`.

I valori restituiti sono rappresentati dal tipo

```
type eval =                     | Char of char
       Undefined                | List of eval list
     | Int of int               | Pair or eval * eval
     | Bool of bool             | Closure of exp * env
```

dove `env` è il tipo dell'ambiente ed è `type env = string -> eval`.

Le operazioni sull'ambiente sono tre: la creazione di un ambiente (che si deve chiamare `emptyenv`) in cui ad ogni elemento in `ide` è associato il valore `Undefined`, la `applyenv` che riceve una coppia `(r,x)` del tipo `(env,ide)` e restituisce il valore nell'ambiente `r` associato al nome `x` (quindi `applyenv` ha tipo `(ide -> eval) * ide -> eval`), mentre `bind` riceve una tripla `(r,x,e)` del tipo `(env,ide,eval)` ed restituisce un nuovo ambiente `r'` dove al nome `x` è associato `e` mentre a tutti i nomi `y` diversi da `x` è associato il valore `applyenv(r,y)`.

Il tipo in `Ocaml` per i tipi del linguaggio è il seguente:

```
type etype =                        | TPair of etype * etype
    TBool                           | TList of etype list
  | TInt                            | TFun of etype * etype;;
  | TVar of string
```

e per generare ogni volta una nuova variabile dovete usare il seguente codice:

```
let nextsym = ref (-1);;
let newvar = fun () -> nextsym := !nextsym + 1;
                       TVar ("?T" ^ string_of_int (!nextsym));;
```

Queste righe di codice `Ocaml` hanno una parte imperativa per generare un nuovo numero.

L'ambiente per i tipi (*type*) ha come come tipo `(ide * etype) list` ed ha una funzione per creare un ambiente per i tipi che si chiama `newtypenv`, una per ottenere il tipo associato ad un `ide` che si chiama `applytypenv` ed un modo per aggiungere o modificare un legame che è `bindtyp` che riceve un ambiente per tipi, un elemento di tipo `ide` ed un valore di tipo e restituisce l'ambiente per tipi aggiornato.

Questi devono avere quindi i seguenti tipi

```
val newtypenv : (ide * etype) list
val applytypenv : (ide * etype) list -> ide -> etype
val bindtyp : (ide * etype) list -> ide -> etype -> (ide * etype) list
```