Precisazioni in base ai vostri dubbi

1. **Q:** è corretto che tra i valori di tipo exp non compaia il valore List ?
   A: è corretto: il costruttore per la lista è **`Empty`** (lista vuota, come si evince dalla semantica) e poi il **`Cons of exp * exp`** che corrisponde ai :: dell'**`Ocaml`**.
   Q: Se si quale è il procedimento per implementare la lista?
   A: Come illustrato prima, ovviamente i selettori della testa e del resto della lista sono quelli ovvi.

2. **Q:** La formula dell'uguaglianza a pag 3 è corretta?
   Dopo un'attenta analisi sembrerebbe mancare la forzatura dell'uguaglianza tra il termine a sinistra ed il termine a destra dell'uguale.
   A: Avete ragione, tra i vincoli che si generano manca questo: (t_1,t_2)

3. **Q:** L'equivalente di List.hd con argomento la lista vuota deve restituire eccezione come la funzione di libreria oppure deve restituire la lista vuota?
   A: A vostro piacimento

4. **Q:** La regola di inferenza per il not a pag 3 è corretta?
   A: Si

5. **Q:** Può essere che manchi il valore **`TChar`** per il tipo **`etype`** per l'inferenza di **`Echar`**?
   A: Si. Manca. Aggiungetelo. Provvederò a creare un file con i tipi.

6. **Q:** Si può usare ai fini della modularità del progetto la keyword #use per utilizzare qualcosa definito in un altro file; nello specifico vorremmo collegare la typeinf ai controlli per la valutazione di sem?
   A: Se volete si.

7. **Q:** La definizione di env a pag 6 è corretta? Non dovrebbe essere **`type env = ide -> eval`**?
   A: dipende dall'implementazione. Potete quindi, se volete, ridefinire il tipo dell'ambiente come **`type env = ide -> eval.`** In effetti è più naturale in questo modo che non **`type env = string -> eval`**

8. **Q:** è corretto che Pair compaia con lo stesso nome sia in exp che in eval ? Ciò non potrebbe creare conflitti nella valutazione per interprete e compilatore
   A: nel tipo **`exp`** il costruttore per le coppie corretto è **`Epair`**

9. **Q:** Per quanto riguarda la funzione applyenv definita a pagina 6, è scritto che quest'ultima riceve una coppia di tipo (env,ide) ma è specificato anche che l'interpretazione della funzione deve restituire (ide -> eval) * ide -> eval . è corretto? la forzatura di r ad env produce il risultato env*ide -> eval, mentre la forzatura di r ad (ide -> eval) violerebbe quanto scritto.
   A: alla luce del punto 7, **`env`** e **`ide -> eval`** sono lo stesso tipo. Talvolta Ocaml protesta, però sono sostanzialmente lo stesso tipo, quindi il tipo della **`applyenv`** può essere visto come **`(ide -> eval) * ide -> eval`** o come **`env * ide -> eval`** dato che **`env`** e **`ide -> eval`** sono lo stesso tipo

10. **Q:** nella regola del tail a pag 2 del progetto il tail non dovrebbe avere tipo tau list invece di tau?
    A: Si, è corretto che deve essere τ list e non τ.

11. **Q:** qual è l'esatto tipo della funzione bind definita a pagina 6?
    A: **`env * ide * eval -> env`**

12. **Q:** nella regola a pag 5, la quale descrive la valutazione di una funzione nella sua chiusura, la sigma indicata a destra si riferisce a σF descritto a pagina 4?
    A: Dato che lo scope è statico, è l'ambiente al momento della dichiarazione di tale funzione. La condizione a pagina 4 si riferisce alle variabili libere nella funzione.

13. **Q:** la **`newtypenv: (ide*etype) list`** è da interpretare come **`newtypenv: unit -> (ide*etype) list`**?
    A: No. Dato che l'ambiente per i tipi lo si interpreta come una lista, deve in qualche modo forzare il tipo (ad esempio creando una lista del tipo corretto, inserendoci un elemento, e poi levandolo subito).

14. **Q:** **`Cons(Eint 1,Cons(Eint 2,Empty))`** dev'essere uguale a **`TList(TInt::[])`** oppure a **`TList(TInt::TInt::TInt)`**
    A: entrambe vanno bene, la seconda vi consente un controllo a posteriori, se ad esempio da **`Cons(Eint 1,Cons(True,Empty))`** trovate **`TList(TInt::TBool::TBool)`** allora potete inferire che l'espressione non è tipabile.

15. **Q:** Riguardando la funzione che tratta le free variables è sorto un dubbio : è corretto che not non compaia nella definizione di FV(t) a pagina 4?
    A: No, è una dimenticanza: dovete considerare anche che FV(not e) = FV(e).

16. **Q:** Qual è il tipo esatto di **`emptyenv`**?
    A: il tipo è **`ide -> eval`** (quindi un ambiente).

17. **Q:** Cosa deve restituire **`Try`** nel caso si verifichi un eccezione? se la valutazione di un espressione specifica restituisce un eccezione, **`Try`** deve restituire un valore specifico?
    A: Se avete **`Try(exp1,id,exp2)`** allora il risultato della valutazione di **`exp2`**.

18. **Q:** Il tipo **`Tchar`** è corretto con la c minuscola o dobbiamo usare lo stesso stile degli altri etype che utilizzano la seconda lettera dopo la T maiuscola.
    A: meglio usare lo stesso stile: ho corretto il file con i tipi.

19. **Q:** La parola **`Appl`** dei tipi **`exp`** è corretta? O la parola esatta è Apply?
    A: **`Appl`** va benissimo dato che è il costruttore nel tipo **`exp`**

20. **Q:** Dobbiamo effettuare l'eq pure sul tipo closure nella parte della semantica?
    A: No, troppo complesso

21. **Q:** Nella parte sulla semantica, manca la regola del **`Less`**. La dobbiamo comunque implementare?
22. **A:** Si, la dovete implementare. Ho comunque corretto il file su moodle e anche dal dokuwiki

23. **Q:** Possiamo utilizzare funzioni di libreria di ocaml? Quali **`List.hd`**, la fold right ecc? O è più opportuno costruire delle funzioni di supporto?
    A: A scelta vostra

24. **Q:** Al fine di Usare il **`Try`** e il **`Raise`** una funzione scritta in questo tipo è permessa dalla sintassi?

```
Try(
  Let(Ide "prova",
           Fun(Ide "x",
                   (Ifthenelse
                      ( Not (Eq (Val(Ide "x"), Empty)),
                        Eint 4,
                        (Raise (Ide "Funziona cosi'")))
                     )
              ),
```

```
      Appl(Val(Ide "prova"), Empty))
    ,(Ide "Funziona cosi'"),
  True);;
```

A: No: gli identificatori (compresi quelli del **`Try`**) devono essere in scope.

25. **Q:** Al fine di utilizzare il **`Rec`** è possibile usare questa sintassi

```
let list=
 Rec (Ide "y",
  Fun (Ide "x",
    Ifthenelse (Eq (Val (Ide "x"), Eint 0), Empty,
         Cons (Val (Ide "x"),
         Appl (Val (Ide "y"), Diff (Val (Ide "x"), Eint 1))))));;

sem (Appl(list, Eint 10)) emptyenv;;
```

A: No: Mentre il Rec è corretto, non lo è il "sem". Infatti list è un nome nell'ambiente dell'interprete, non nel linguaggio.

26. **Q:** Sia in sem che in typeinf è prevista l'uguaglianza di (in un espressione **`Eq`**): **`Let, Fun, Rec, liste, Appl, Ifthenelse`**? Ciò che crea il dubbio sono le due regole per l'uguaglianza a pagina 4. Se si, in che modo è definita l'uguaglianza tra queste espressioni (ad esempio, **`Eq(fun1,fun2)`** vale true se hanno parametro con stesso nome e stesso tipo e stesso corpo oppure ...)?
A: L'uguaglianza sulle coppie è ovvia: devono essere uguali le componenti (nel corretto ordine). Anche per le liste è ovvia. Per le funzioni non è ovvia, quindi non la dovete implementare. Sarebbe troppo riduttiva. Attenzione, che se io scrivo **`Eq(Appl(Val(Ide("f")),Eint 1),Val(Ide("f")),Eint 1))`** dove **`Ide("f")`** è una funzione che restituisce un intero, devo fare l'uguaglianza su interi.

27. **Q:** Nella regola per il not a pagina 3, è corretto che non compaia un vincolo che forzi l'espressione (chiamata t) del not ad essere booleana nella parte conclusione della regola (qualcosa tipo (t,bool))?
A: Se pensate che il vincolo manchi, aggiungetelo.

28. **Q:** Per una migliore comprensione del **`Try`**, in una espressione del tipo:

```
semtry(
  Try(
    (Try (
       (Ifthenelse( Eq(Val (Ide "x"), Echar 'c'),
        Raise (Ide "ecc1"),
        Raise (Ide "ecc2")
              )),
           Ide "ecc2",
           Eint 3)),
```

```
            Ide "ecc1",
            Eint 4)
)
(bind (emptyenv,(Ide "x"), Char 'c'));;
```

Deve essere restituito `Int 4, Int 3` o l'espressione non è corretta?

A: L'espressione è corretta ed ovviamente restituisce `Int 4`.