# Progetto di Linguaggi e Compilatori 2
# A.A. 2014-15

# Gruppo 0

**Attenzione: una volta letto il presente testo è *obbligatorio* consegnare alla scadenza quanto elaborato, indipendentemente dal fatto se lo si consideri adeguato o meno.**

Ogni gruppo deve inviare via email entro il termine stabilito la relazione in formato PDF (dal nome "`ProgettoLC2 Gruppo 0 Relazione`") assieme a tutti i sorgenti sviluppati (in modo che sia possibile verificare il funzionamento di quanto realizzato) all'interno di un file compresso, il cui nome sia "`ProgettoLC2 Gruppo 0`". In tale file compresso *non* devono comparire file oggetto e/o binari. La relazione può contenere immagini passate a scanner di grafici o figure (oppure si può concordare, al momento del ritiro, per una consegna del cartaceo delle figure senza doverle includere nel file PDF). Si richiede:

**Per tutti gli esercizi** descrizione dettagliata di tutte le tecniche non-standard impiegate (le tecniche standard imparate a lezione non vanno descritte).

**Per tutti gli esercizi** descrizione delle assunzioni fatte riguardo alla specifica, sia relativamente a scelte non previste espressamente dalla specifica stessa, che a scelte in contrasto a quanto previsto (con relative motivazioni).

**Per gli esercizi 7 e 9** descrizione sintetica generale della soluzione realizzata.

**Per gli esercizi 6, 7 e 9** commentare (molto) sinteticamente ma adeguatamente il codice prodotto.

**Per l'esercizio 9** fornire opportuni Makefile (compliant rispetto allo standard GNU make) per

- poter generare automaticamente dai sorgenti i files necessari all'esecuzione (lanciando `make`);
- far automaticamente una demo (lanciando `make demo`).

**Per gli esercizi 6 e 7** aggiungere un file di testo con alcune query di test (da poter lanciare a terminale). Oppure fornire un Makefile come al punto precedente.

**L'esercizio 9** *deve* essere implementato in Haskell.

1. Si rappresenti il codice della macchina di Warren relativo al seguente programma in Prolog.

```
% ***** ONLINE *****

nqueens(N, List):-
        length(List,N),
        fd_domain(List,1,N),
        constrain_queens(List),
        fd_labeling(List, [variable_method(ff)]).

constrain_queens([]).
constrain_queens([X|Y]):-
        safe(X,Y,1),
        constrain_queens(Y).

safe(_,[],_).
safe(X,[Y|T],K):-
        noattack(X,Y,K),
        K1 is K+1,
        safe(X,T,K1).

noattack(X,Y,K):-
        X #\= Y,
        Y #\= X+K,
        X #\= Y+K.
```

2. Data la grammatica

$$S \rightarrow aBDb$$
$$B \rightarrow Bc \mid d$$

$D \rightarrow EF$
$E \rightarrow e \mid \epsilon$
$F \rightarrow f \mid \epsilon$

1. Dare un'equivalente grammatica LL(1).
2. Costruirne il parser top-down.
3. Mostrare l'esecuzione di suddetto parser sull'input `abcdcb`, utilizzando opportuni passi di error recovery.

3. Data la grammatica

$S \rightarrow$ `id = id` | `id = num` | `while id > num beg` $L$ `end`
$L \rightarrow S \mid L$ `;` $S$

1. Costruire i parsers SLR e LALR.
2. Mostrare l'esecuzione di suddetti parsers sull'input

```
while x> 5
 beg
   a = x
   z = u ;
   y=
 end
```

utilizzando opportuni passi di error recovery. Qualora ci fossero conflitti nelle rispettive tabelle, si utilizzi la convenzione di prediligere lo shift sul reduce e di scegliere il reduce con regola testualmente precedente.

4. Si determinino opportuni termini $t_1, t_2, t_3$ affinché il seguente programma Curry sia sintatticamente corretto, ben tipato, la funzione $h$ sia induttivamente sequenziale ed inoltre la query $q =$ `f (h (Left x) y) x where x,y free` sia ammissibile.

```
f (Just _) (Right _)        = []
f x        (Left _)         = [x]
f _        (Right (Left y)) = [y]
f x        (Left y)         = x:y

h t1 [_,_] = Nothing
h t2 (y:_) = Just (t3,x)
```

Si calcoli l'albero di *needed* narrowing per $q$.

Per i rami terminati si scriva la risposta calcolata.

5. Si consideri il seguente frammento di codice sintatticamente ammissibile sia in Haskell che in Curry.

```
data T a b = V b | N a (T a b) (T a b)

h _ (V y) (Right z) = y=:=z
h x (N y _ _) (Left z) = 3*z=:=2+y
h x (N y _ _) (Left z) = (z < y)=:=True & z=:=x*y & y=:=2+x
```

Si assuma di aver esteso il Prelude di Haskell con le definizioni (=:=) = (==) e **True** & x = x.

Preso $t =$ `N 1 (V 'a') (V 'b')`, si descrivano le valutazioni nei due linguaggi delle queries h $(-1)$ $t$ (**Left** $(-1)$) e h 0 $t$ (**Left** 1).

Si mostri una rappresentazione dello spazio di ricerca di Curry per la query h $n$ $x$ $y$ con $n$ ground e $x$, $y$ variabili libere.

Se ha senso si riscriva la versione di un linguaggio per ottenere gli stessi risultati di quella dell'altro (su queries ammissibili da entrambi i linguaggi) o si spieghi perché ciò non sia possibile. Oppure si riscriva se avesse un qualche effetto pratico per queries ammissibili solo in un linguaggio.

6. Rappresentando Alberi "generici" con i costruttori `void/0` e `node/2`, dove `node` mantiene il dato di un nodo e la lista dei figli, si scriva una funzione `diameter/2` che determina il diametro di un albero. Il diametro di un albero è la lunghezza del massimo cammino fra due nodi, indipendentemente dall'orientamento degli archi.

   A titolo di esempio, `diameter( node(f, [ node(c, [node(b,[node(a,[])]), node(d, [node(e,[])]) ]) ]), D)` restituisce `D = 4`.

   Si discuta su cosa potrebbe essere fatto (qualora fosse possibile) per far funzionare il precedente predicato su termini non necessariamente ground.

7. Si vuole risolvere il seguente gioco: data una pulsantiera luminosa $n \times n$ con alcuni pulsanti accesi, si deve trovare (se esiste) un **insieme** di mosse che spenga l'intera pulsantiera. Una mossa consiste nel premere uno dei pulsanti. Premendo un pulsante viene invertito il suo stato (da acceso diventa spento e viceversa) e quello dei quattro pulsanti ad esso adiacenti (sopra, sotto, destra e sinistra), sempre che ci siano.

   A titolo d'esempio, la configurazione $\begin{bmatrix} \boxed{1} & 2 & 3 \\ 4 & 5 & 6 \\ \boxed{7} & \boxed{8} & 9 \end{bmatrix}$ ammette come soluzione l'insieme $\{1, 5, 8, 9\}$.

   Codificando i dati come si ritiene piú opportuno, si scriva una funzione Curry che, data una configurazione iniziale della pulsantiera, restituisce un **insieme** di mosse che risolvono il gioco.

8. Caratterizzare tutti gli (eventuali) overlap delle regole scritte negli esercizi 4, 5, 6 e 7. Nello specifico, per ogni coppia di regole specificare se non sono in overlap oppure (in caso lo siano) si fornisca il testimone più generale (unico a meno di varianza).

9. Si consideri un linguaggio costruito a *partire* dalla stessa *sintassi concreta* di **C**.

   In tale linguaggio (che non è C) le procedure son funzioni di tipo `void`. Si possono dichiarare funzioni, oltre che variabili, all'interno di qualsiasi blocco.

   Si hanno le operazioni aritmetiche, booleane e relazionali standard ed i tipi ammessi siano interi, booleani, real, caratteri, stringhe, array e puntatori (con i costruttori e/o le operazioni di selezione relativi).

   Non è necessario avere tipi di dato definiti dall'utente.

   Si hanno inoltre le funzioni predefinite `writeInt`, `writeFloat`, `writeChar`, `writeString`, `readInt`, `readFloat`, `readChar`, `readString`.

   Il linguaggio deve implementare—con la sintassi concreta di C qualora sia prevista—le modalità di passaggio dei parametri value e value-result (le altre a piacere), con i relativi vincoli di semantica statica.

   Il linguaggio deve ammettere le istruzioni `break` e `continue` (dentro il corpo dei cicli), con la sintassi concreta di C, qualora sia prevista.

   Per detto linguaggio (non necessariamente in ordine sequenziale):

   - Si progetti una opportuna sintassi astratta.
   - Si progetti un type-system (definendo le regole di tipo rispetto alla sintassi astratta generata dal parser, eventualmente semplificandone la rappresentazione senza però snaturarne il contenuto semantico).
   - Si implementi un lexer con Alex, un parser con Happy ed il corrispondente pretty printer (si suggerisce di utilizzare BNFC per costruire un primo prototipo iniziale da raffinare poi manualmente, ma non è obbligatorio).
   - Si implementi il type-checker e *tutti* gli altri opportuni controlli di semantica statica (ad esempio il rilevamento di utilizzo di r-expr illegali nel passaggio dei parametri). Si cerchi di fornire messaggi di errore che aiutino il più possibile a capire in cosa consiste l'errore, quali entità coinvolge e dove queste si collochino nel sorgente.
   - Dopo aver definito una opportuno data-type con cui codificare il three-address code (non una stringa di char!), si implementi il generatore di three-address code per il linguaggio considerato (non C), tenendo presente che:
     – gli assegnamenti devono valutare l-value prima di r-value;

- l'ordine delle espressioni e della valutazione degli argomenti si può scegliere a piacere (motivandolo);
- le espressioni booleane nelle guardie devono essere valutate con short-cut. In altri contesti si può scegliere a piacere (motivandolo).

Si predispongano dei test case significativi per una funzione che, dato un nome di file, esegua il parsing del contenuto, l'analisi di semantica statica, il pretty-print del sorgente ed (infine) generazione e pretty-print del three-address code.

Si tenga presente che del linguaggio C, da cui si trae ispirazione per la sintassi concreta, non si richiede di capire la semantica di funzionamento (irrilevante ai fini della soluzione) ma solo di utilizzare le stesse scelte di sintassi concreta relativamente ai costrutti (canonici) previsti dal testo.