

Processadors de Llenguatges

Pràctica II – Eines de suport a la generació d'analitzadors sintàctics

Curs 20/21

Objectius

L'interès de la pràctica és que l'alumne formalitzi les regles sintàctiques per a llenguatges formals i utilitzi una eina de generació automàtica d'analitzadors sintàctics basats en la tècnica LR(1), concretament la tècnica LALR(1), com per exemple *yacc* o *bison*, per a la seva implementació.

Exercicis de programació

Exercici 1. Avaluació d'expressions aritmètiques.

Obteniu una especificació sintàctica (*yacc*) que simuli una petita calculadora. La calculadora tindrà 52 registres (26 de tipus enter etiquetats amb les lletres de la **a** a la **z** i 26 de tipus real etiquetats amb les lletres de la **A** a la **Z**) i acceptarà expressions enteres i reals. Podeu considerar un sistema de tipus estricte o un sistema de tipus que incorpori conversions implícites de tipus, és a dir, podeu considerar erroni combinar enters i reals en una mateixa expressió o podeu incorporar la conversió automàtica dels tipus específic (enter) al tipus real.

- Utilitzareu l'assignació, operador **=**, per introduir valors en els registres els quals podreu emprar en posteriors expressions.
- Els operadors aritmètics i de manipulació de bits que heu de considerar són els següents:
 - Operadors aritmètics binaris d'enters: **+**, **-**, *****, **div**, **mod** (residu). Considereu els formats lèxics que vulgueu.
 - Operadors aritmètics unaris d'enters: **+**, **-** (canvi de signe unari)
 - Operadors aritmètics binaris de reals: **+**, **-**, *****, **/**
 - Operadors aritmètics unaris de reals: **+**, **-**
 - Operadors de desplaçament de bits (únicament definits per als enters): **<<**, **>>**
 - Operadors de bit (únicament definits per als enters): **~** (complement a u), **&** (AND entre bits), **|** (OR inclusiu entre bits), **^** (OR exclusiu entre bits)
- Els operands poden ser registres i constants, ambdós de tipus enter o real. Per a les constants, considereu els formats lèxics que vulgueu.
- Considereu un operador de conversió explícita de tipus d'enter a real. Considereu el format lèxic que vulgueu.

- El final d'una expressió vindrà indicat pel caràcter ';' i el final de tractament per la marca de final d'arxiu. Un cop calculat el resultat d'una expressió l'escriurem a la sortida estàndard, excepte si es tracta d'una expressió d'assignació. En aquest darrer cas, actualitzareu el valor del registre intern referenciat.
- Desprecieu els caràcters blancs, tabuladors i nova línia existents a l'entrada.
- Verifiqueu que el llenguatge acceptat per l'analitzador sintàctic és el desitjat. És a dir, verifiqueu que elimineu tots els conflictes mitjançant la definició, per a cada operador, del corresponent nivell de precedència i associativitat. Normalment els operadors aritmètics són més prioritaris que els operadors de desplaçament de bits i aquests, són més prioritaris que els operadors de manipulació de bits.
- Incorporeu el tractament d'errors, seguint l'estratègia del mode pànic, a nivell sintàctic. Els possibles errors lèxics seran tractats a nivell sintàctic. Indiqueu el número de línia on es detecta l'error.
- Al concluir el processat es mostrarà el valor dels registres definits.

Per exemple, amb l'entrada següent:

```
a = 2;
b = -3;
c = a+b;
a-b;
c*8;
A=(real)a + 2.1;
```

La sortida pot ser la següent:

```
5
-8
a=2
b=-3
c=-1
A=4.1
```

Exercici 2. Conversor de notació infix a notació sufixa.

Escriviu una especificació sintàctica (*yacc*) que accepti com a entrada una expressió aritmètica entera en notació infix i que generi com a sortida l'expressió aritmètica entera equivalent en notació postfixa. Recordeu que en la notació postfixa amb operadors binaris són necessaris els parèntesis.

- Els operadors binaris mínims que heu de considerar són els següents: $+$, $-$, $*$, *div*, *mod*. Considereu els formats lèxics que vulgueu.
- Els operands podran ser qualsevol constant entera. Considereu els formats lèxics que vulgueu.
- Les expressions podran estar parentitzades.
- El caràcter ';' actuarà com a marca de final d'expressió i la marca de final d'arxiu com a marca de fi d'entrada.

- Desprecieu els caràcters blancs, tabuladors i nova línia existents a l'entrada.
- Verifiqueu que el llenguatge acceptat per l'analitzador sintàctic és el desitjat. És a dir, verifiqueu que elimineu tots els conflictes mitjançant la definició, per a cada operador, del corresponent nivell de precedència i associativitat. Recordeu que de major (1) a menor (2) prioritats tenim els operadors següents:

1. $*$, div , mod i
2. $+$, $-$.

A més, tots són associatius per l'esquerra. D'altra banda, els parèntesis permeten modificar la prioritats associada amb els operadors.

- Incorporeu el tractament d'errors, seguint l'estratègia del mode pànic, a nivell sintàctic. Els possibles errors lèxics seran tractats a nivell sintàctic. Indiqueu el número de línia on es detecta l'error.
- Per resoldre el problema, podeu escriure l'expressió resultant en notació postfixa directament a la sortida estàndard, És a dir, no cal construir l'arbre sintàctic a memòria i recorre'l en postordre.

Per exemple, l'expressió aritmètica infix següent:

$(3 + 4) * (5 - 6);$

es transforma en:

$3\ 4\ +\ 5\ 6\ -\ *$

Exercici 3. Eliminar els parèntesis redundants.

Escriviu una especificació sintàctica (*yacc*) que accepti com a entrada una expressió aritmètica entera en notació infix i que generi com a sortida l'expressió aritmètica entera equivalent però havent-hi eliminat els possibles parèntesis redundants. L'especificació de les expressions aritmètiques enteres en notació infix vàlides és la mateixa que a l'exercici anterior.

Per exemple, l'expressió aritmètica entera infix següent:

$((3 * 4) - ((5 / 6 * 4) * (2 + 3)))$;

pot ser convertida en l'expressió aritmètica entera infix equivalent següent:

$3 * 4 - 5 / 6 * 4 * (2 + 3)$

A l'igual que a l'exercici anterior, per resoldre el problema, no cal construir l'arbre sintàctic a memòria i recorre'l avaluant la necessitat o no de parèntesis. Podeu anar construint l'expressió resultant com un string que va creixent amb el processament de l'entrada, la qual un cop acabat el processat (expressions acabades en $;$) es copia a la sortida estàndard.

Exercici 4. Construcció de Thompson.

L'algorisme de Ken Thompson permet construir l'AFN amb λ -transicions reconeixedor del llenguatge denotat per una expressió regular. Trobareu descrit l'algorisme als apunts de l'assignatura, Secció 2.6.1 Construcció de Thompson. Escriuiu una especificació sintàctica (*yacc*) que accepti com a entrada una expressió regular definida sobre l'alfabet $\Sigma = \{a, b, c, d\}$ i produeixi com a sortida una representació de l'AFN amb λ -transicions associat. Podeu limitar el nombre d'estats de què consta l'AFN amb λ -transicions a construir.

- A nivell lèxic considerarem les expressions regulars definides a l'Exercici 3 (Lenguatge de les expressions regulars) de la 1a pràctica del curs (lex).

- El caràcter ';' actuarà com a marca de final d'expressió regular i la marca de final d'arxiu com a marca de fi d'entrada.
- Desprecieu els caràcters blancs, tabuladors i nova línia existents a l'entrada.
- Verifiqueu que el llenguatge acceptat per l'analitzador sintàctic és el desitjat. És a dir, verifiqueu que elimineu tots els conflictes mitjançant la definició, per a cada operador, del corresponent nivell de precedència i associativitat. Recordeu que de major (1) a menor (3) prioritats tenim els operadors següents:
 1. *, +, ?,
 2. . (concatenació) i
 3. | (alternativa).

A més, *, +, ? són unaris i tots els operadors del llenguatge són associatius per l'esquerra (els unaris i els binaris). D'altra banda, els parèntesis permeten modificar la prioritats associada amb els operadors.

- Incorporeu el tractament d'errors, seguint l'estratègia del mode pànic, a nivell sintàctic. Els possibles errors lèxics seran tractats a nivell sintàctic. Indiqueu el número de línia on es detecta l'error.
- Per mostrar l'AFN amb λ -transicions, podeu emprar un format textual senzill.

Per exemple, l'expressió regular següent:

(a | b) . c;

és reconeguda pel següent AFN amb λ -transicions:

Descripció del AF:

```
Estats numerats del 1 al 8
[Estat 1, Simbol a] Go to 2
[Estat 2, Lambda] Go to 6
[Estat 3, Simbol b] Go to 4
[Estat 4, Lambda] Go to 6
[Estat 5, Lambda] Go to 1
[Estat 5, Lambda] Go to 3
[Estat 6, Lambda] Go to 7
[Estat 7, Simbol c] Go to 8
Estat inicial: 5
Estat final: 8
```

Exercici 5. Eliminar les implicacions i dobles implicacions.

Escriviu una especificació sintàctica (*yacc*) que accepti com a entrada una fórmula proposicional definida sobre les lletres en majúscula (rang [A-Z]) i generi com a sortida la fórmula proposicional equivalent sense les connectives d'implicació i doble implicació.

- A nivell lèxic considerarem l'especificació definida a l'Exercici 2 (Llenguatge de la lògica proposicional) de la 1a pràctica del curs (lex).
- El caràcter ';' actuarà com a marca de final de fórmula proposicional i la marca de final d'arxiu com a marca de fi d'entrada.
- Desprecieu els caràcters blancs, tabuladors i nova línia existents a l'entrada.

- Verifiqueu que el llenguatge acceptat per l'analitzador sintàctic és el desitjat. És a dir, verifiqueu que elimineu tots els conflictes mitjançant la definició, per a cada operador lògic, del corresponent nivell de precedència i associativitat. Recordeu que de major (1) a menor (4) prioritat tenim els operadors lògics següents:

1. la negació !,
2. la conjunció \wedge ,
3. la disjunció \vee i
4. la implicació representada amb el string " $- >$ ", i la doble implicació representada amb el string " $< - >$ ".

A més, la negació ! és un operador unari associatiu per la dreta i, la resta són binaris i associatius per l'esquerra. D'altra banda, els parèntesis permeten modificar la prioritat associada amb els operadors.

- Incorporeu el tractament d'errors, seguint l'estratègia del mode pànic, a nivell sintàctic. Els possibles errors lèxics seran tractats a nivell sintàctic. Indiqueu el número de línia on es detecta l'error.
- Per transformar les fòrmules recordeu:

$$A \rightarrow B \equiv \neg(A) \vee (B)$$

$$A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A) \equiv (\neg(A) \vee (B)) \wedge (\neg(B) \vee (A))$$

Per exemple, en el nostre llenguatge la fòrmula següent:

$$P \wedge Q < - > (!R \vee (Q - > T));$$

pot ser transformada en la fòrmula equivalent següent:

$$(!((P \wedge Q) \vee ((!R \vee (!(Q) \vee (T)))))) \wedge (!((!R \vee (!(Q) \vee (T)))) \vee (P \wedge Q))$$

- A l'igual que a l'exercici anterior, per resoldre el problema, no cal construir l'arbre sintàctic a memòria i recorre'l avaluant les transformacions. Podeu anar construint la fòrmula equivalent com un string que es va completant amb el processament de l'entrada, la qual un cop acabada (fòrmules acabades en ;) es copia a la sortida estàndard.

Exercici 6. Exercici opcional (+ 0,5 punts)

Amplieu la solució a l'exercici 5 amb l'eliminació dels parèntesis redundants i propagant la negació a nivell de variables proposicionals. És a dir, quan la negació afecta a una fòrmula aquesta s'ha de transformar aplicant la propietat de la doble negació i la transformació de Morgan fins que únicament la negació afecti a variables (literals positius i negats). Recordeu que si A i B denoten fòrmules:

- $\neg\neg A \equiv A$,
- $\neg(A \wedge B) \equiv \neg(A) \vee \neg(B)$ i
- $\neg(A \vee B) \equiv \neg(A) \wedge \neg(B)$.

Lliurament

La documentació a lliurar per a cada exercici de programació és la següent:

1. Especificació *lex* i *yacc*.
2. Mòduls auxiliars emprats en la implementació de la solució global.
3. Joc de proves utilitzat per a la validació de l'exercici.
4. Si ho considereu oportú, un fitxer README amb les particularitats pròpies de la vostra implementació.

Lliurament de la pràctica al **cv.udl.cat** dins d'activitats. Lliurar un arxiu comprimit que agrupi mitjançant la utilitat **tar**, els fitxers fonts (extensions **.l**, **.y** i **.c**) dels exercicis de programació, els jocs de proves utilitzats per a la validació dels exercicis, el README i un arxiu **makefile** que compili correctament els fitxers font per a cada exercici de programació.

Avaluació

- La pràctica la podeu realitzar de manera individual o en grups de 2 o 3 persones.
- El pes d'aquesta pràctica és d'un 15% sobre la nota final de l'assignatura.
- La data límit per lliurar la pràctica és el 3 de maig per a l'avaluació contínua. Pels que no la tingueu acabada el 3 de maig, la podeu lliurar fins el dilluns 14 de juny (data de l'examen del 2n parcial de l'assignatura).

Lliurament

Lliurament de la pràctica al **cv.udl.cat** dins d'activitats. Lliurar un arxiu que agrupi els fitxers fonts (extensions **.l** i **.y**) dels exercicis de programació, els jocs de proves utilitzats per a la validació dels exercicis i els arxius **makefile** que compilin correctament els fitxers font per a cada exercici de programació.

Avaluació

- El pes d'aquesta pràctica és d'un 20% sobre la nota final de l'assignatura.
- La data límit per lliurar la pràctica és el 23 d'abril per a l'avaluació contínua. Pels que no la tingueu acabada el 23 d'abril, la podeu lliurar fins el 23 de juny (avaluació final de l'assignatura).