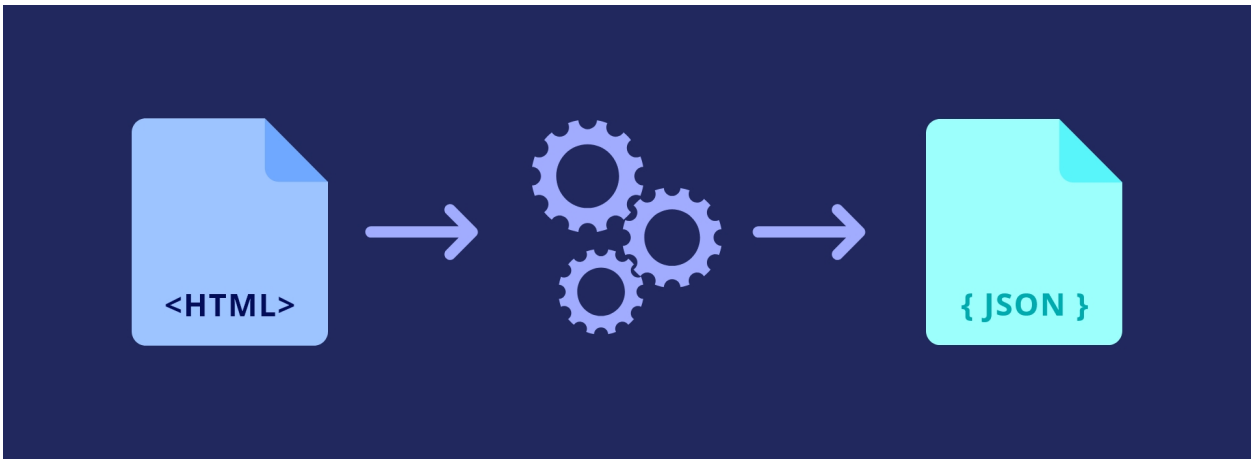


Parser Homework

Santiago Ahumada Lozano

Punto 1
Punto 2
Punto 3
Punto 4
Punto 5

Punto 1

Escribe una gramática para un subconjunto interesante de oraciones en español, incluyendo sustantivos, verbos, adjetivos, adverbios, conjunciones, subordinadas frases, etc. (Incluya solo algunos terminales en cada categoría para dar una idea.) ¿Es la gramática LL(1), LR(1) o ambigua? Explique por qué.

Solución. Definamos:

- S = Sujeto
- SU = Sustantivo
- D = Determinante
- P = Predicado
- S = Sustantivo
- FP = Frase preposicional
- PN = Pronombre
- V = Verbo

- O = Oración
- Ad = Adverbio

Por tanto, las producciones son las siguientes

1. P→VA
2. C→y|,
3. Ad→ϵ|”Interesante”| “Hermoso”
4. V→ “usó” | “cambió”
5. SU → “Hombre” | “Mujer” | “Santiago”
6. S→A SU Ad
7. O→ SP
8. O → OCO

Esta gramática no es LL1 pues presenta recursión izquierda en (8.).

Punto 2

Considere la siguiente gramática:

1. $P \rightarrow S$
2. $P \rightarrow S\ P$
3. $S \rightarrow$ if E then S
4. $S \rightarrow$ if E then S else S
5. $S \rightarrow$ while E S
6. $S \rightarrow$ begin P end
7. $S \rightarrow$ print E
8. $S \rightarrow E$
9. $E \rightarrow$ id
10. $E \rightarrow$ integer
11. $E \rightarrow E + E$

- En cuales aspectos la gramática no es LL(1):

  Solución: No es LL1 dada la existencia de ambigüedad en las reglas 3. y 4. También deja de ser LL1 al presentar recursión a izquierda en la regla 11.

- Escriba una gramática que acepte el mismo lenguaje pero que no tenga recursiones a la izquierda ni prefijos comunes a la izquierda

  Solución:

  1. T → id
  2. S → while E S
  3. P → S P’
  4. E’ → **ϵ**
  5. S → begin P end
  6. P’ → P
  7. S → if E then S’
  8. S’ → S
  9. T → int
  10. E → TE’
  11. S’ → I
  12. I → If E then S else S’
  13. P’ → **ϵ**
  14. S → print E
  15. S → E
  16. E’ → TE’

- Escriba los FIRST y los FOLLOW de la nueva gramática

  Solución:

  - FIRST:
    - S={int, id, print, begin, while, if}
    - P’={**ϵ,** int, id, print, begin, while, if}
    - P={int, id, print, begin, while, if}
    - T={int, id}
    - S’ = {int, id, print, begin, while, if}
    - I={if}
    - E={int, id}

        - E’={**ϵ}**
    - FOLLOW
        - S={$, end, int, id, print, begin, while, if, else}
        - P’={$, end}
        - P={$, end}
        - T={$, end, int, id, print, begin, while, if, else}
        - S’ = {$, end, int, id, print, begin, while, if, else}
        - I={$, end, int, id, print, begin, while, if, else}
        - E={$, end, int, id, print, begin, while, if, else}
        - E’={$, end, int, id, print, begin, while, if, else}
- Escriba un parser recursivo para la nueva gramática

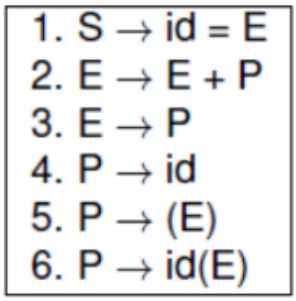
  Solución:

```
int parse_P(){
  return parse_P() && parse_P_prim();
}

int parse_P_prim(){
  t = scan_token();
  if(parse_P()){
    return 1;
  }
  else {
    put_back_token(t);
    return 1;
  }
}

int pase_S(){
  token t = scan_token()
  if(t=="if"){
    return parse_E() && expect_token("then") && parse_S_prime();
  }
  else if(t=="while"){
    return parse_E() && parse_S();
  }
  else if(t=="begin"){
    return parse_P() && expect_token("end");
  }
  else if(t=="print"){
    return parse_E();
  }
  else {
    putback_token(t);
    return parse_E();
  }
}
```

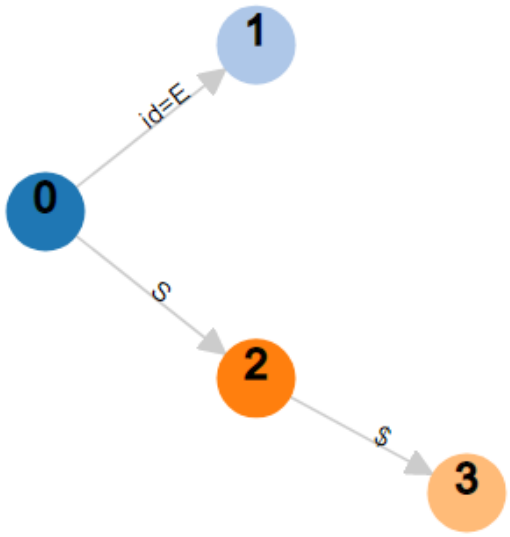
```
int parse_I(){
  token t = scan_token();
  if (t=="if"){
    return parse_E() && expect_token("then") && parse_S() && expect_token("else") && parse_S_prime();
  }
}

int parse_E(){
  t = scan_token();
  if(parse_P()){
    return 1;
  }
  else {
    put_back_token(t);
    return 1;
  }
}

int parse_E_prime(){
  return parse_E() && parse_E_prim();
}
int parse_T(){
  token t = scan_token();
  if (t=="if"){
    return parse_E() && expect_token("then") && parse_S() && expect_token("else") && parse_S_prime();
  }
}
```

Punto 3

Considere la siguiente gramática:



- Dibuje el autómata LR(0)



- Escriba la tabla de parsing SLR

State	Item set
0	{S' ::= • S $, S ::= • id=E}
1	{S ::= id=E • }
2	{S' ::= S • $}
3	{S' ::= S $ • }

Punto 4

Escriba una gramática para representar el lenguaje JSON
Escriba los first y los follow de la gramática
La gramática que usted escribió es LL(1), SLR o LR(1), o ninguna de ellas, explique.
Reescriba su gramática de la forma más simple posible.
Escriba una tabla de parse apropiada para la gramática

Solución:

1. **J → {E}**

2. E → E, E
3. E → “K”:V
4. K → nombre | id
5. V → Jorge | Laura | 101 | 203

FIRST

1. **J → {**
2. E → “
3. K → nombre ; id
4. V → Jorge ; Laura ; 101; 203

FOLLOW

1. **J → $**
2. E →]
3. K → ”
4. V → }

La gramática no es LL1 pues presenta recursión a izquierda en 2. Dado que podemos derivar cualquier cadena con `shift-reduce` entonces la gramática es LR1

Tabla:

Stck	Input	Action
{”K”:Jorge	, “id:203}$	reduce v→Jorge
{”K”:Jorge	, “id:203}$	reduce E→”K”,V
{”K”:V	, “id:203}$	shift
{E	, “id:203}$	shift, shift
{E,	:203}$	reduce K→id
{E, id	}$	shift, shift, shift
{E, “K”	}$	reduce E→”K”,V
{E,E	}$	reduce E→E,E
{E	}$	shift

Stck	Input	Action
{E}	$	reduce J→{E}
J	$	accept

Punto 5

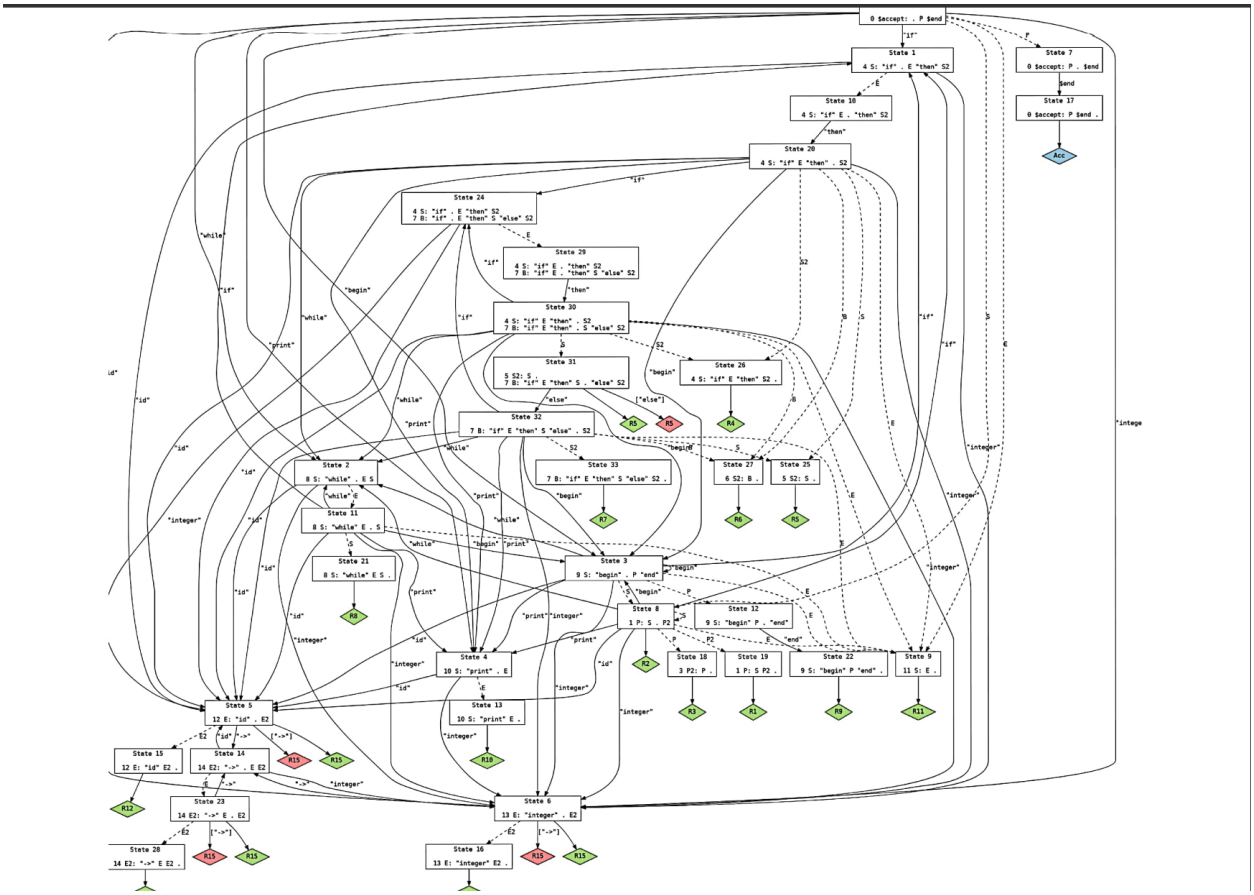
Consulte el manual de BISON y explique cómo se puede generar automáticamente el grafo asociado a la gramática, y genere los grafos asociados a las gramáticas de los puntos 2 y 3.

Solución:

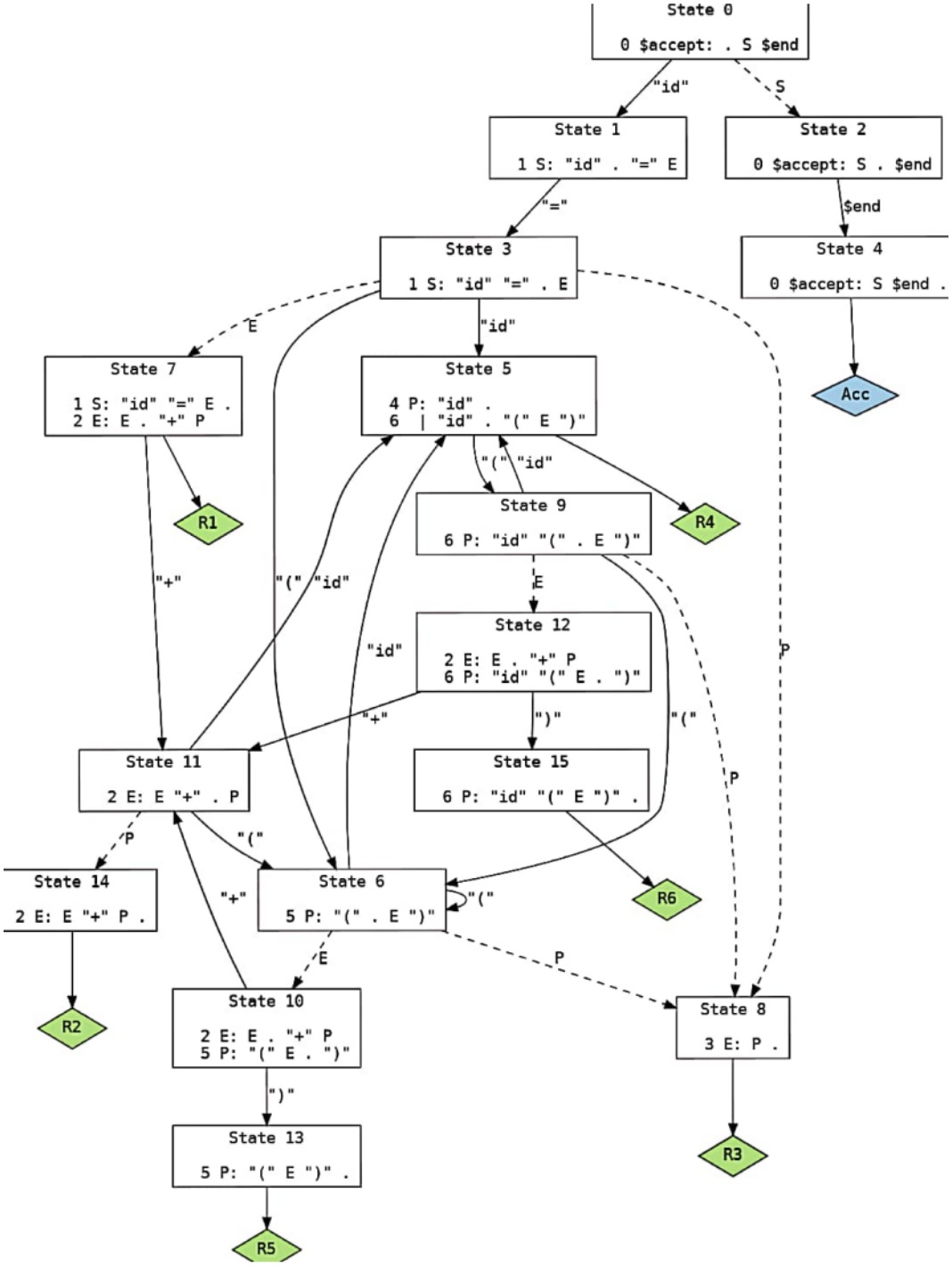
Según la fuente GNU BISON manual. Para generar de manera automática el gráfo asociado a una grámatica se tiene que tener una especificación de la opción `--graph`. El manual indica que si el archivo que define la gramática es, por ejemplo, `foo.y`, entonces, se creará de manera automática un archivo homónimo pero con la extensión de Graphviz, es decir, `foo.gv`, además, debemos generar una archivo con la extensión `.dot` de la siguiente forma:

```
bison my_file.y --graph
dot -Tpng my_file.dot > output.png
```

Grafo gramática punto 2



Grafo gramática punto 3


State 0
0 $accept: . S $end
"id"
S
State 1
1 S: "id" . "=" E
State 2
0 $accept: S . $end
"="
$end
State 3
1 S: "id" "=" . E
State 4
0 $accept: S $end .
E
"id"
Acc
State 7
1 S: "id" "=" E .
2 E: E . "+" P
State 5
4 P: "id" .
6 | "id" . "(" E ")"
"(" "id"
R1
State 9
6 P: "id" "(" . E ")"
R4
"+"
"(" "id"
E
State 12
2 E: E . "+" P
6 P: "id" "(" E . ")"
"id"
P
"+"
")"
"("
State 11
2 E: E "+" . P
State 15
6 P: "id" "(" E ")" .
P
"("
State 14
2 E: E "+" P .
"+"
State 6
5 P: "(" . E ")"
"("
R6
E
P
R2
State 10
2 E: E . "+" P
5 P: "(" E . ")"
State 8
3 E: P .
")"
State 13
5 P: "(" E ")" .
R3
R5