Intermediate Code Generation using Syntax directed translation schemes (SDTS)

Semantic Analysis

- The semantics of the source code is verified after the syntax is checked.

- Semantic analysis validates the meaning of the code by checking if the sequence of tokens:
    - is meaningful and correct
    - is associated with the correct type
    - is consistent and correct in the way in which control structures and data types are used.

- The semantics of the language is validated with the help of semantic rules.

- usually combined with other phases like the parser.

Semantic Analysis

- Semantic rules can be attached to grammar to perform type checking.

- Semantic rules are a collection of procedures called at appropriate times by the parser as the grammar requires.

- Semantic rules can be applied to the grammar by attaching attributes to the CFG.

- CFG+ sematic rules Attribute grammar

Grammar Attributes

- Attribute grammar
  - E □E + T {E.value = E.value + T.value}

- Based on the way the attributes obtain their values they are divided into two categories:

  - Synthesized – obtain the values from child nodes

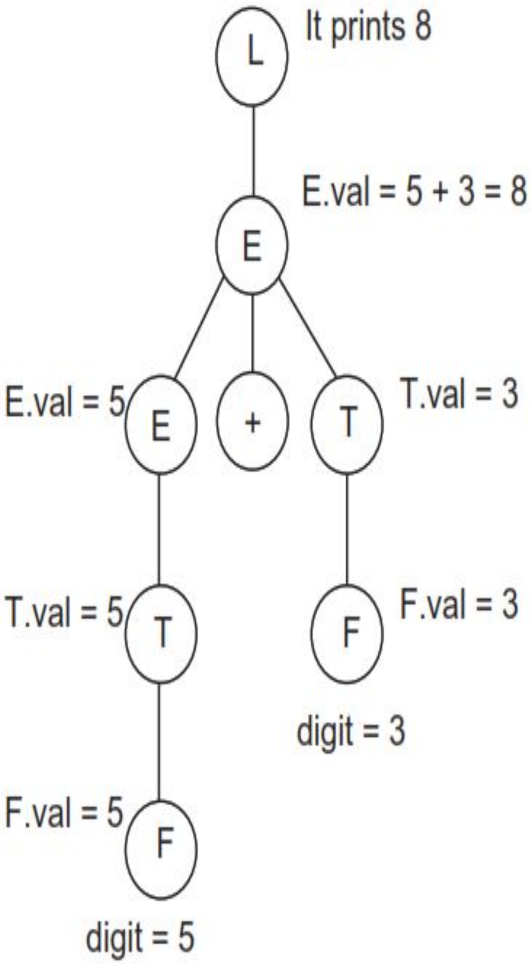
  - Inherited - obtain the values from parents or siblings

Synthesized attributes

- The attributes that obtain values from the attribute values of their child nodes

Production	Semantic Rules
L□E	print(E.val)
E□E1+T	E.val=E1.val+T.val
E□ T	E.val = T.val
T□T*F	T.val = T1.val * F.val
T□ F	T.val = F.val
F□digit	F.val = digit.lexval

Example: 5+3

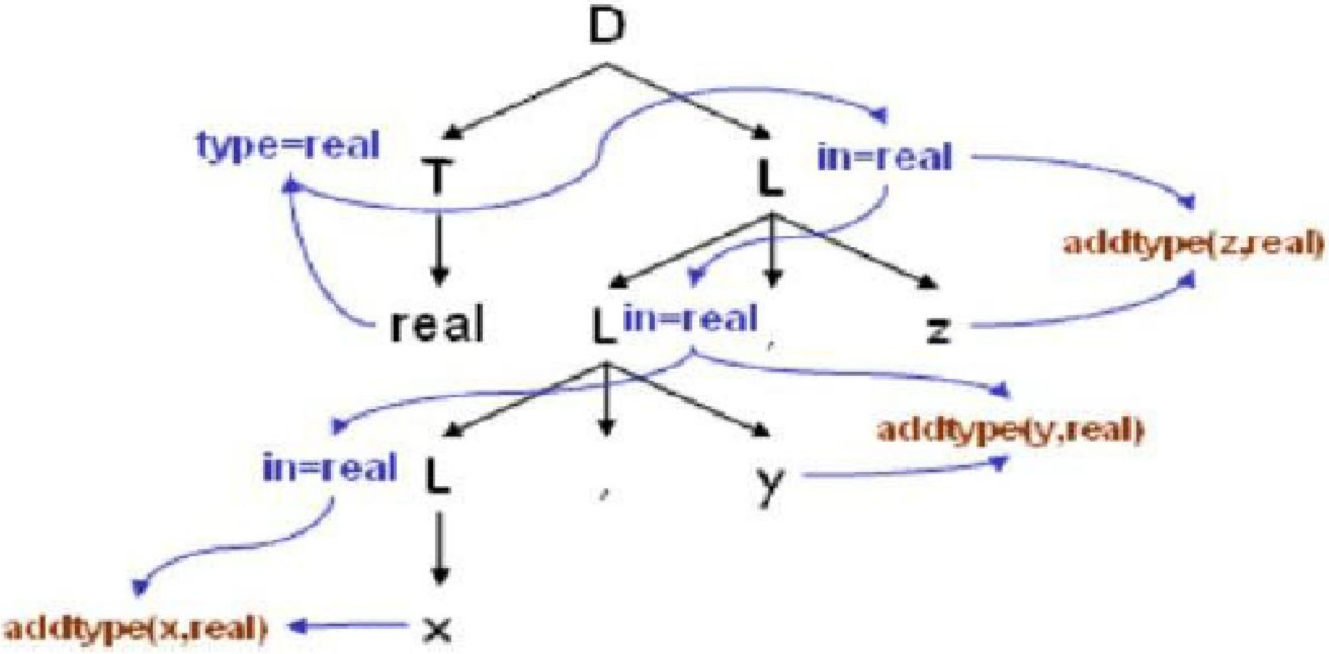


Inherited attributes

- Inherited attributes take values from parents and/or siblings.

Parse tree for real x, y, z

D → T L	L.in = T.type
T → real	T.type = real
T → int	T.type = int
L → L_1 , id	L_1 .in = L.in; addtype(id.entry, L.in)
L → id	addtype (id.entry,L.in)



Intermediate Code Representation- Example

X= (a+b) * - c / d

Quadruple

	op	x (operand1)	y (operand2)	z (result)
(1)				
(2)				
(3)				
(4)				
(5)				

Intermediate Code Representation- Example

X= (a+b) * - c / d

Triple

	op	x (operand1)	y (operand2)
(1)			
(2)			
(3)			
(4)			
(5)			

Intermediate Code Representation- Example

$X= (a+b) * - c / d$

Indirect Triple

			op	x (operand1)	y (operand2)
(1)					
(2)					
(3)					
(4)					
(5)					

Introduction to Intermediate Code Generation

- Many compilers convert the code to an intermediate representation.
- get machine.
- The benefits of using machine-independent intermediate code are as follows:
- It reduces the number of optimisers and code generators.
- It is easy to generate and translate code into the target program.
- It enhances portability.
- It is easy to optimise as compared to machine-dependent code.
- The representation of intermediate code can be directly executed using a program, which is referred to as the interpreter.

Intermediate Code Representation

- Graphical representations can be parse trees, abstract syntax trees, DAG, etc.
- Linear representations are non-graphical like three-address code (TAC), static single assignment (SSA), etc.
- Representation of TACs
  - Quadruples
  - Triples
  - Indirect triples

Syntax-directed Translation into Three-address Code – Principle

- To translate any construct of a programming language, its syntax structure must be specified

- Semantic actions should be defined in the production rules of the grammar.

- The syntax-directed translation (SDT) scheme is used to generate the TAC. In SDT, the parsing process and parse trees are used.

Syntax-directed translation scheme to convert infix to postfix

Grammar	**Semantic rule**
E1E2+T	E1.string = E2.string || T.string || ‘+’
E1 T	E1.string = T.string
T1T2*F	T1.tring = T2.string || F.string || ‘*’
TF	T.string = F.string
F(E)	F.string = E.string
Fnum	F.string = num.string

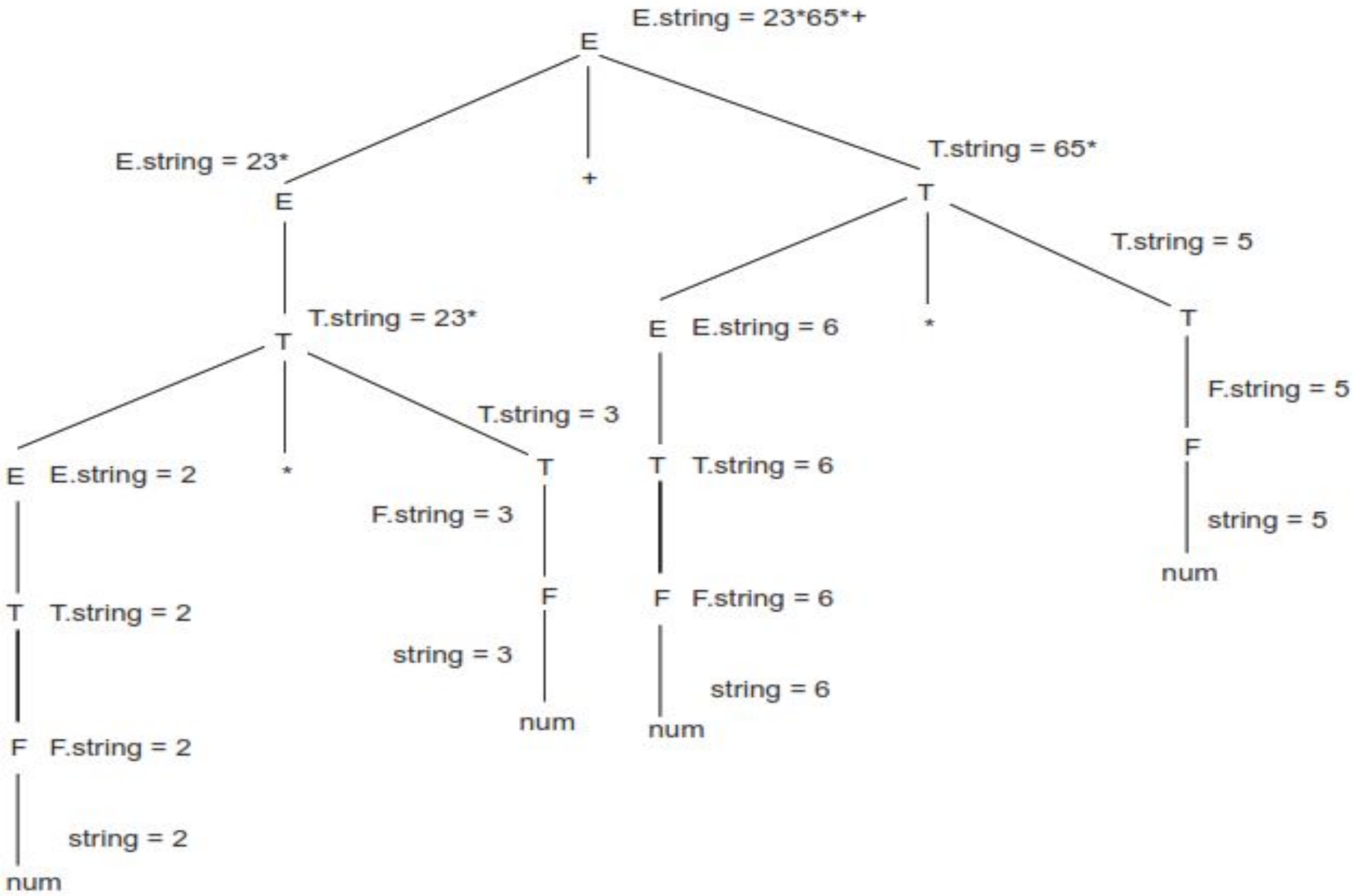
Syntax-directed translation scheme to convert infix to postfix

Annotated parse tree for the input string 2*3+6*5

Grammar	Semantic rule
E1□E2+T	E1.string = E2.string || T.string || ‘+’
E1□ T	E1.string = T.string
T1□E*T	T1.tring = T2.string || F.string || ‘*’
T□F	T.string = F.string
F□(E)	F.string = E.string
F□num	F.string = num.string

Syntax-directed translation scheme to convert infix to postfix

Annotated parse tree for the input string 2*3+6*5



Practice Example

(5+3)* 12 + 7

Grammar	**Semantic rule**
E1□E2+T	E1.string = E2.string || T.string ||’+’
E1□ T	E1.string = T.string
T1□T2*F	T1.tring = T2.string || F.string || ‘*’
T□F	T.string = F.string
F□(E)	F.string = E.string
F□num	F.string = num.string

SDTS of Assignment Statement into Three Address Code (TAC)

A□ id = E
E □E+E | E*E | E-E | (E) | id

Production	Semantic Rule
A □id = E	gen(id.place ‘ =’ E . place)

E□ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E□ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

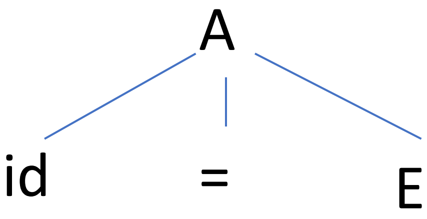
E□ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

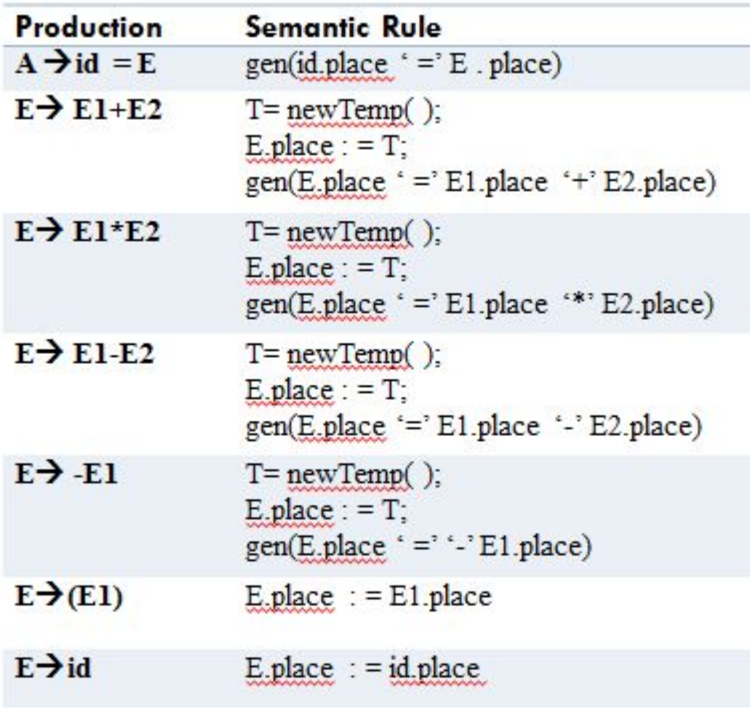
E□ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E□(E1)	E.place : = E1.place
E□id	E.place : = id.place

Example 1

p=(q+r)*(s+t)





Production	Semantic Rule
A →id =E	gen(id.place ‘ =’ E . place)

E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

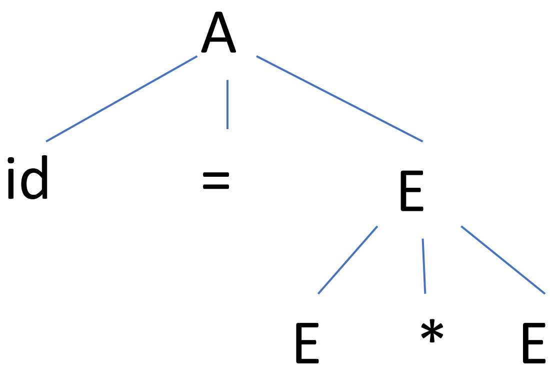
E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

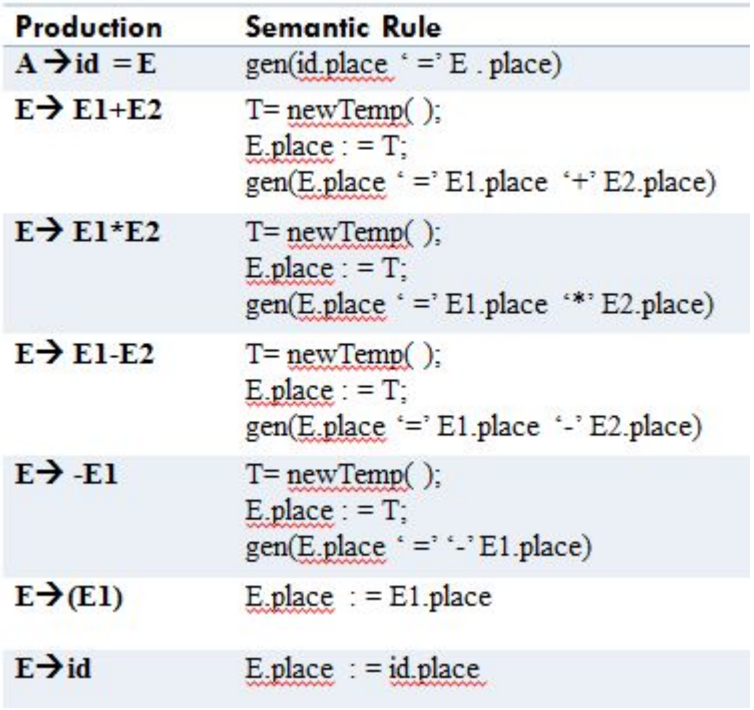
E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place

Example 1

p=(q+r)*(s+t)





Production	Semantic Rule
A→id =E	gen(id.place '=' E . place)

E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place '=' E1.place '+' E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place '=' E1.place '*' E2.place)

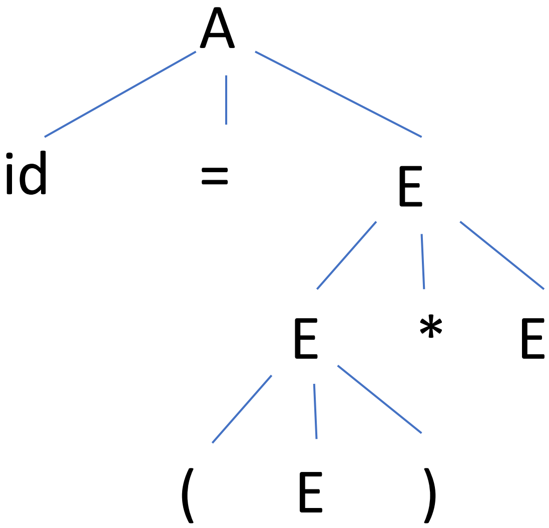
E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place '=' E1.place '-' E2.place)

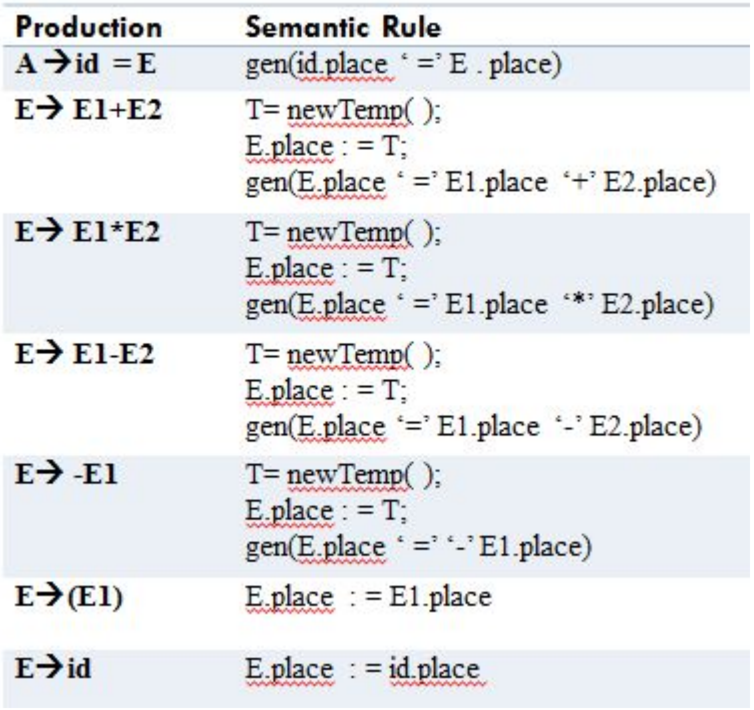
E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place '=' '-' E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place

Example 1

p=(q+r)*(s+t)





Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

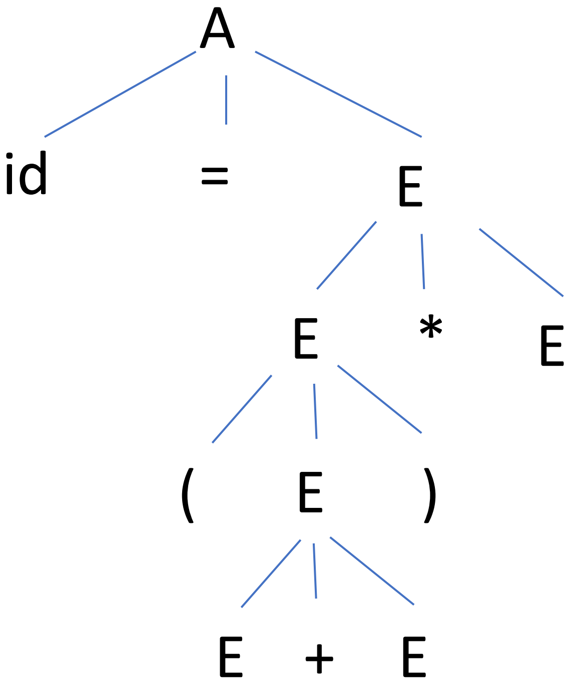
E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

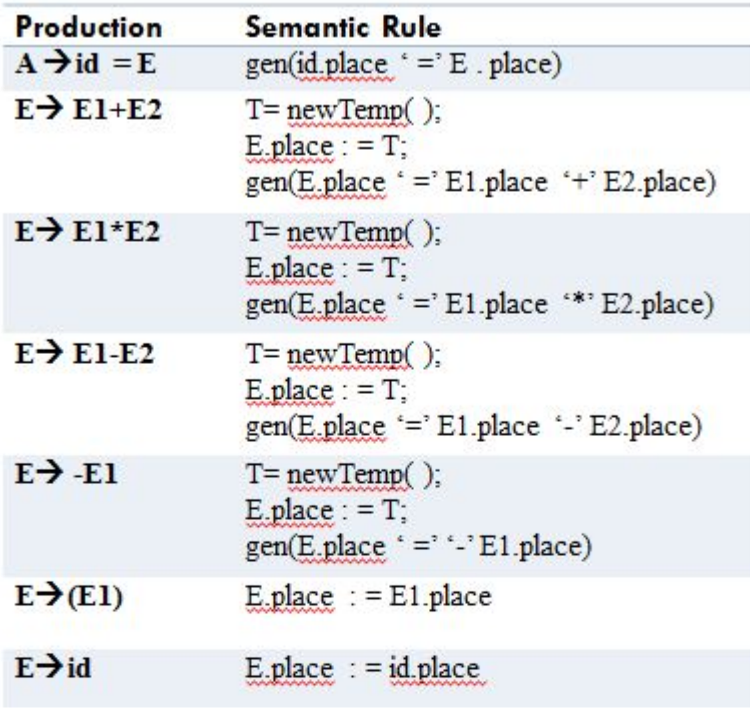
E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place

Example 1

p=(q+r)*(s+t)





Production	Semantic Rule
A→id = E	gen(id.place ‘ =’ E . place)

E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

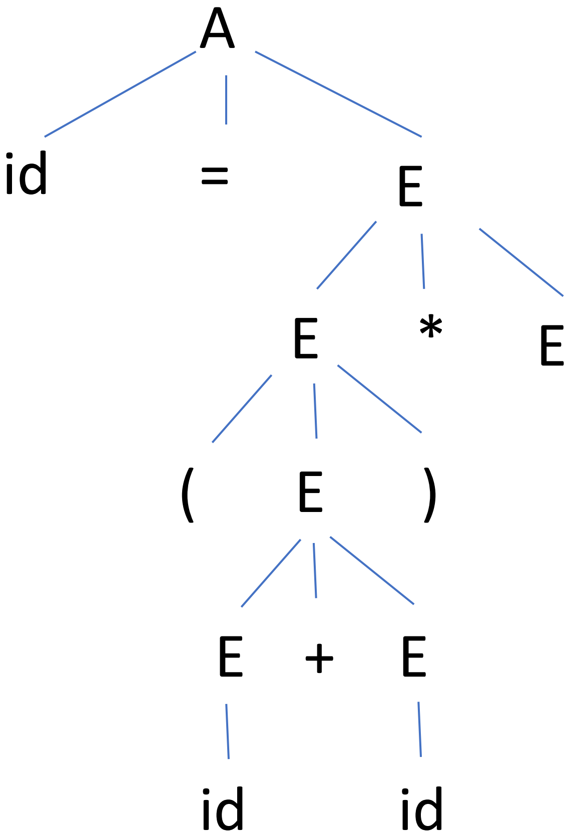
E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

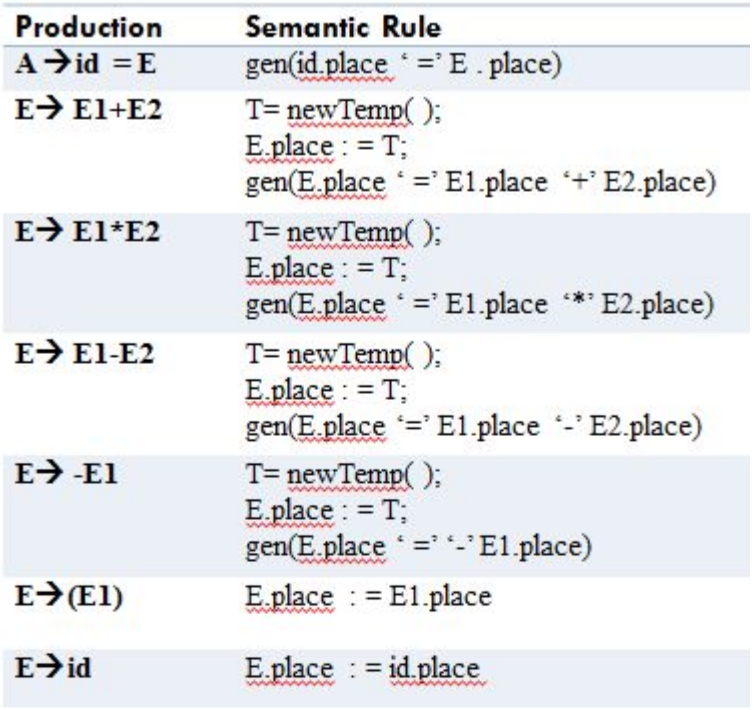
E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place

Example 1

p=(q+r)*(s+t)





Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

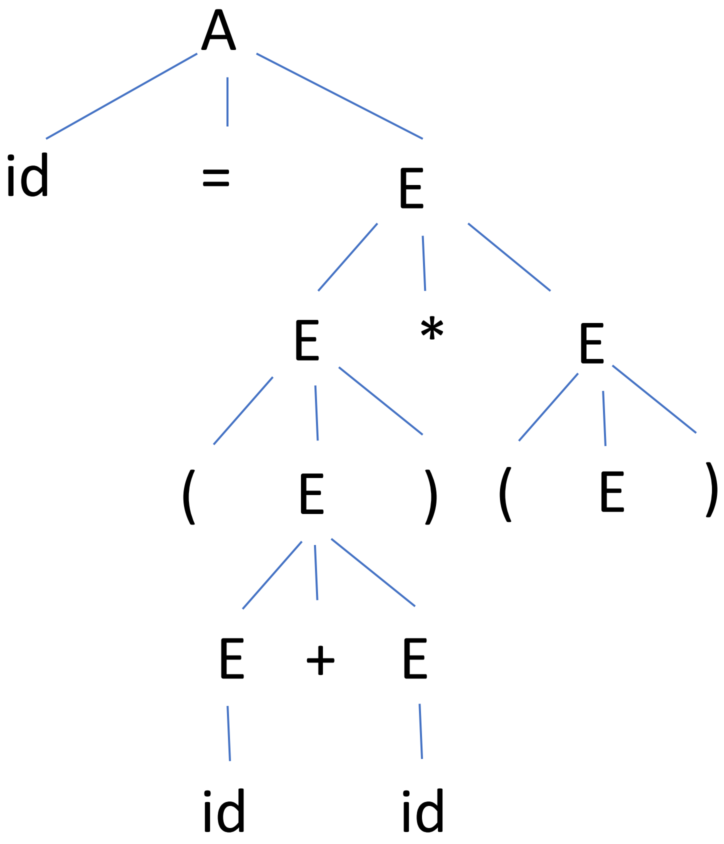
E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

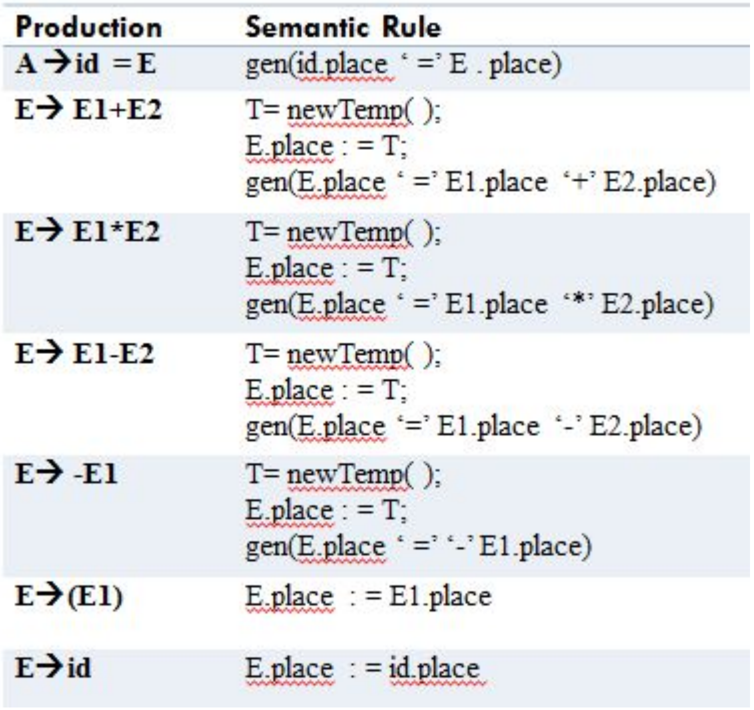
E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place

Example 1

p=(q+r)*(s+t)





Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

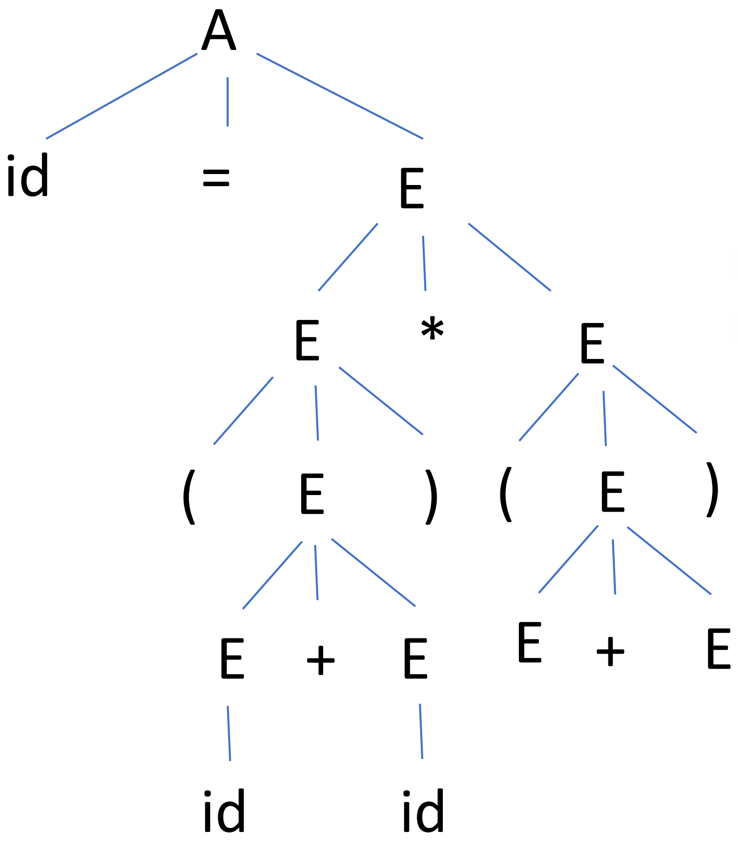
E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

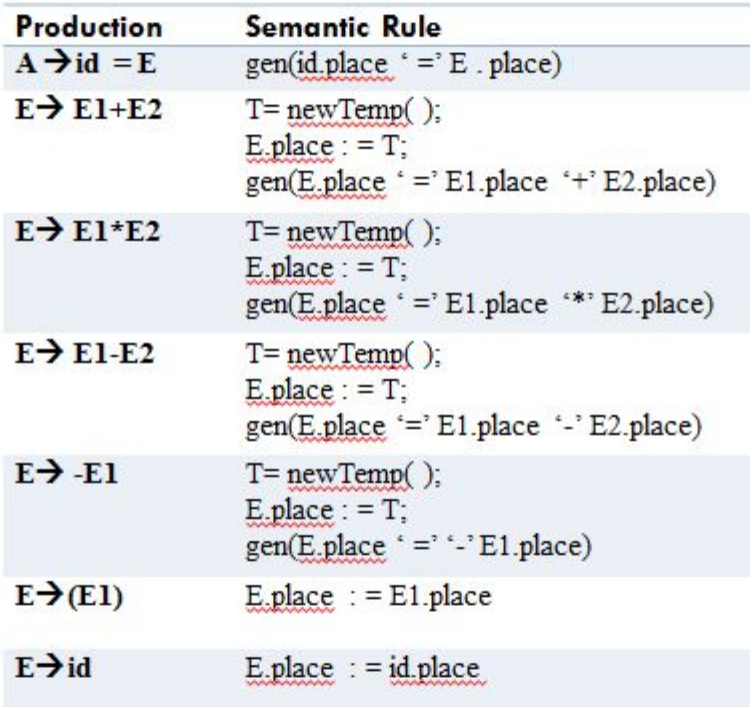
E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place

Example 1

p=(q+r)*(s+t)





Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

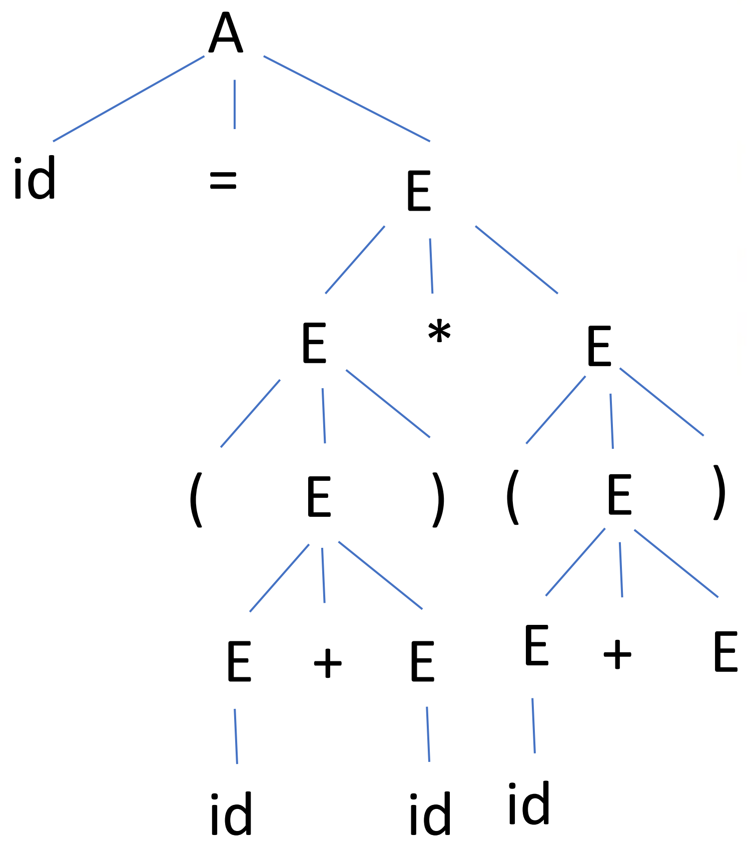
E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

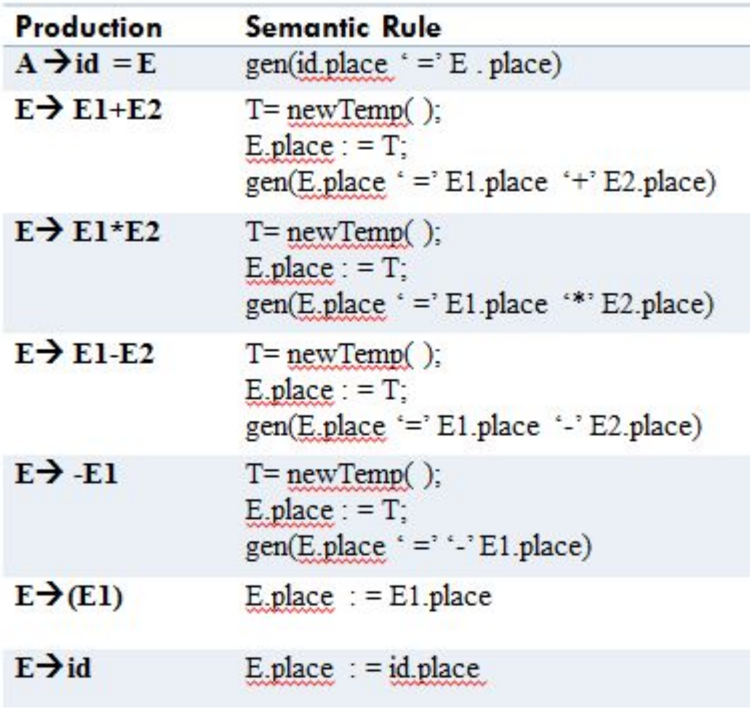
E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place

Example 1

p=(q+r)*(s+t)





Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

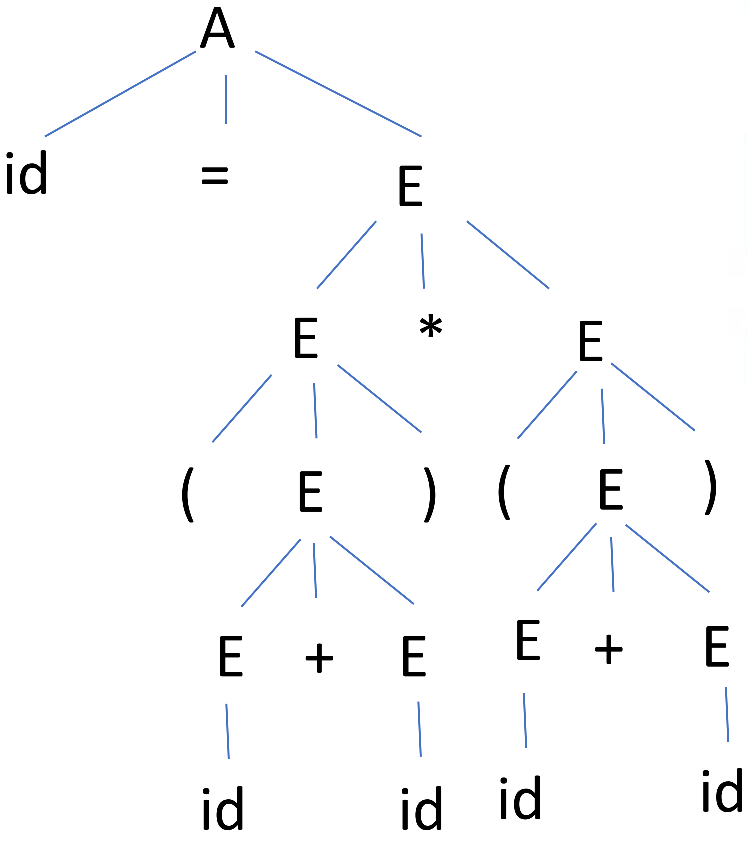
E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

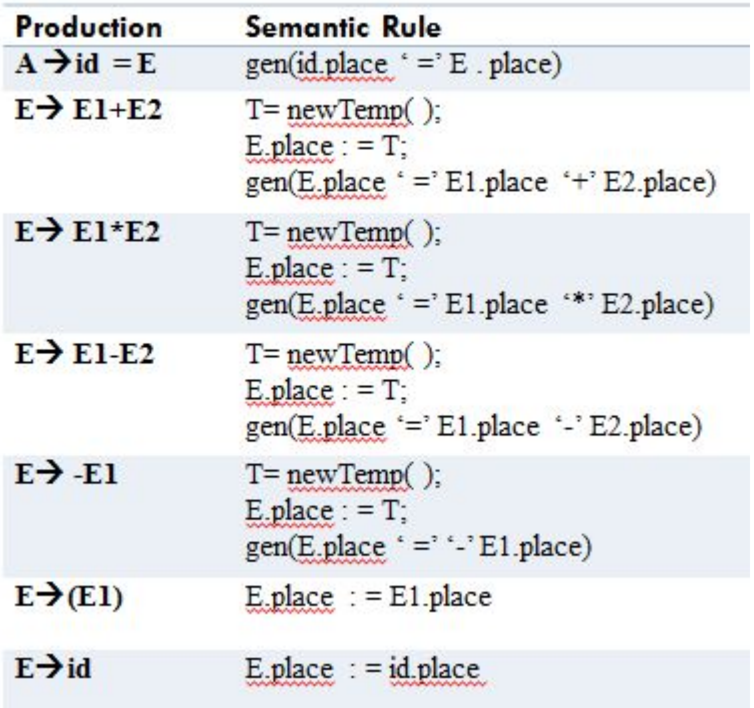
E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place

Example 1

p=(q+r)*(s+t)





Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

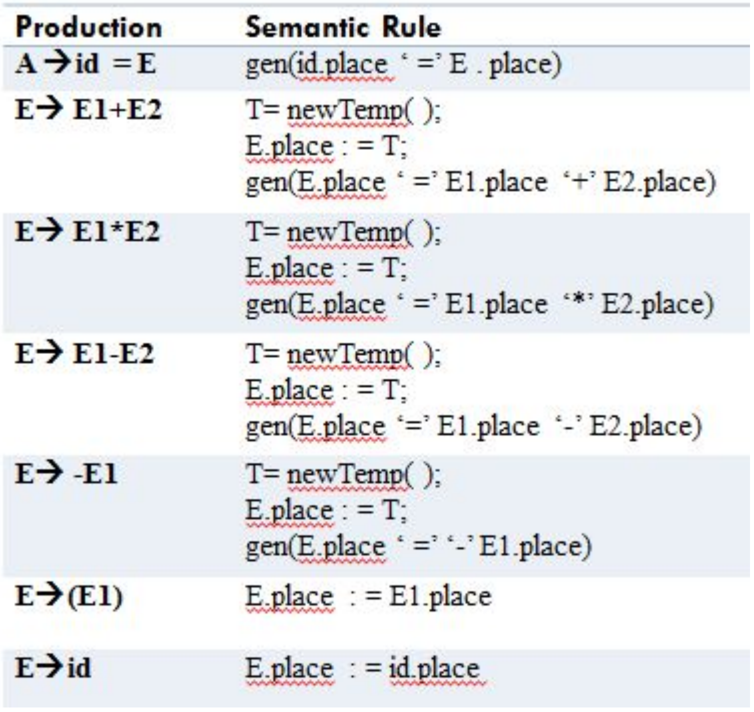
E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place

Example 1

p=(q+r)*(s+t)



Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

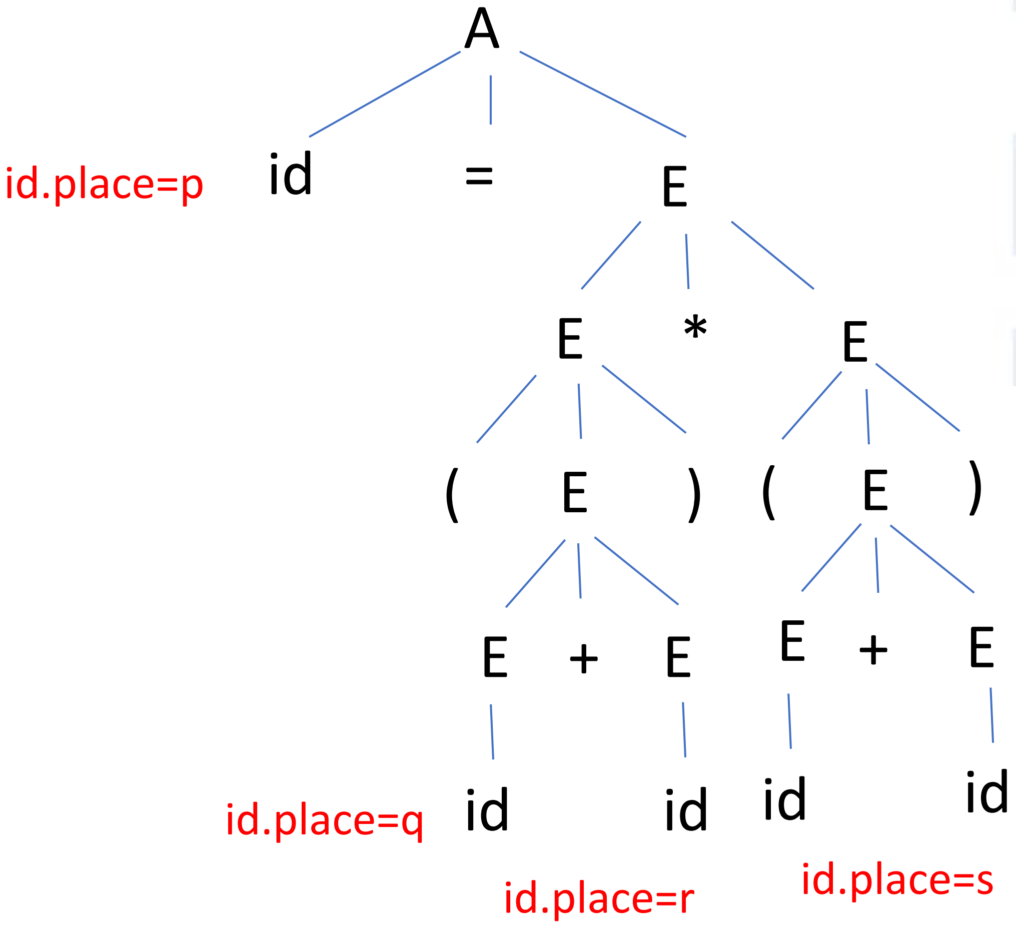
E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

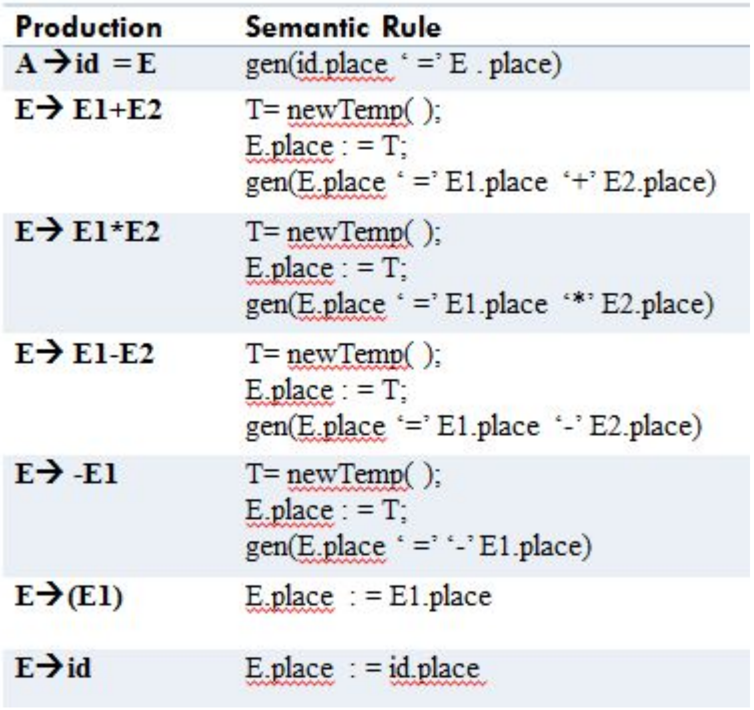
E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place



Example 1

p=(q+r)*(s+t)



Production	Semantic Rule
A→id = E	gen(id.place ‘ =’ E . place)

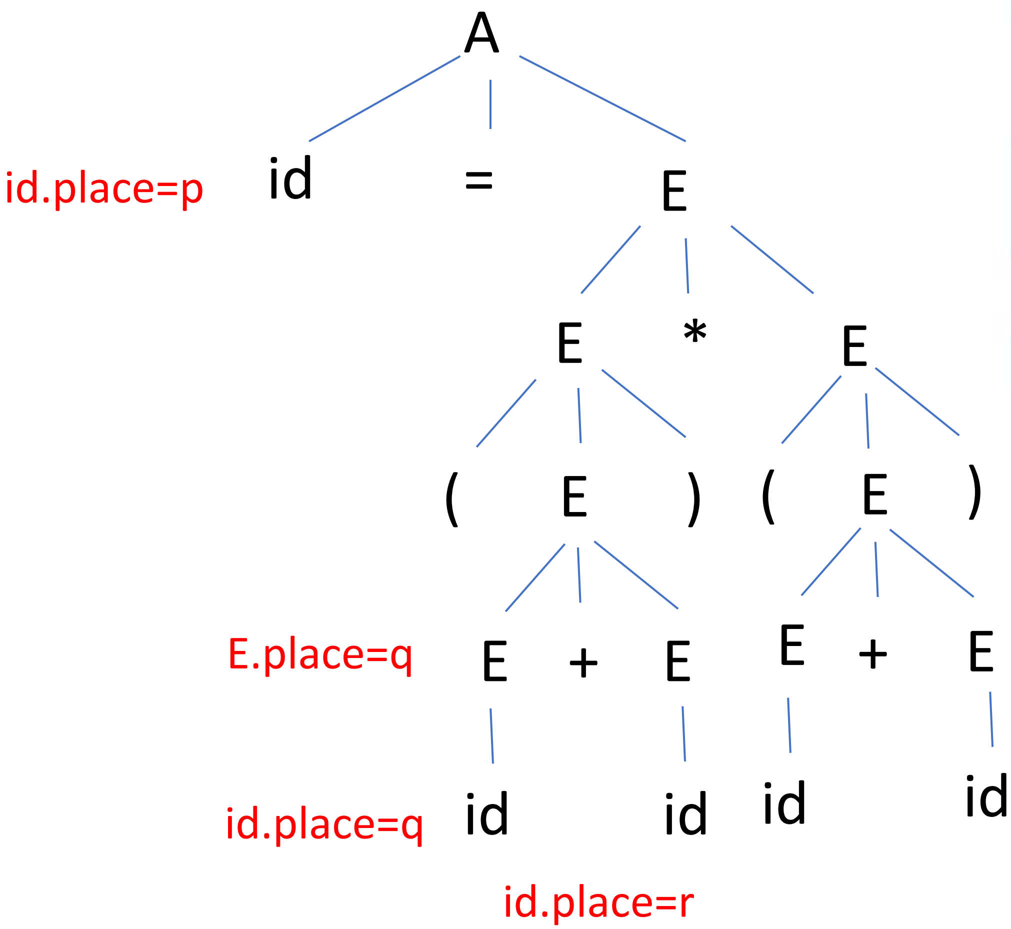
E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

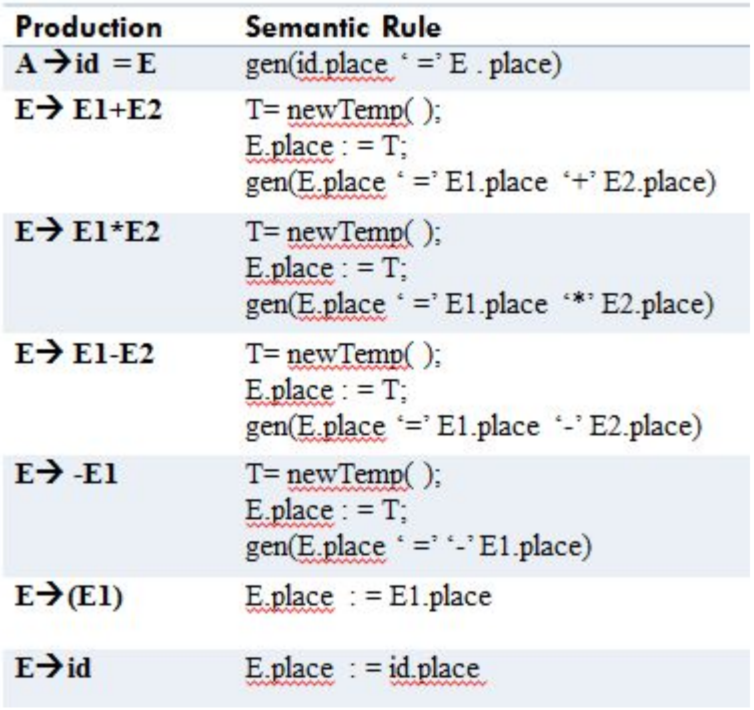
E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place



Example 1

p=(q+r)*(s+t)



Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

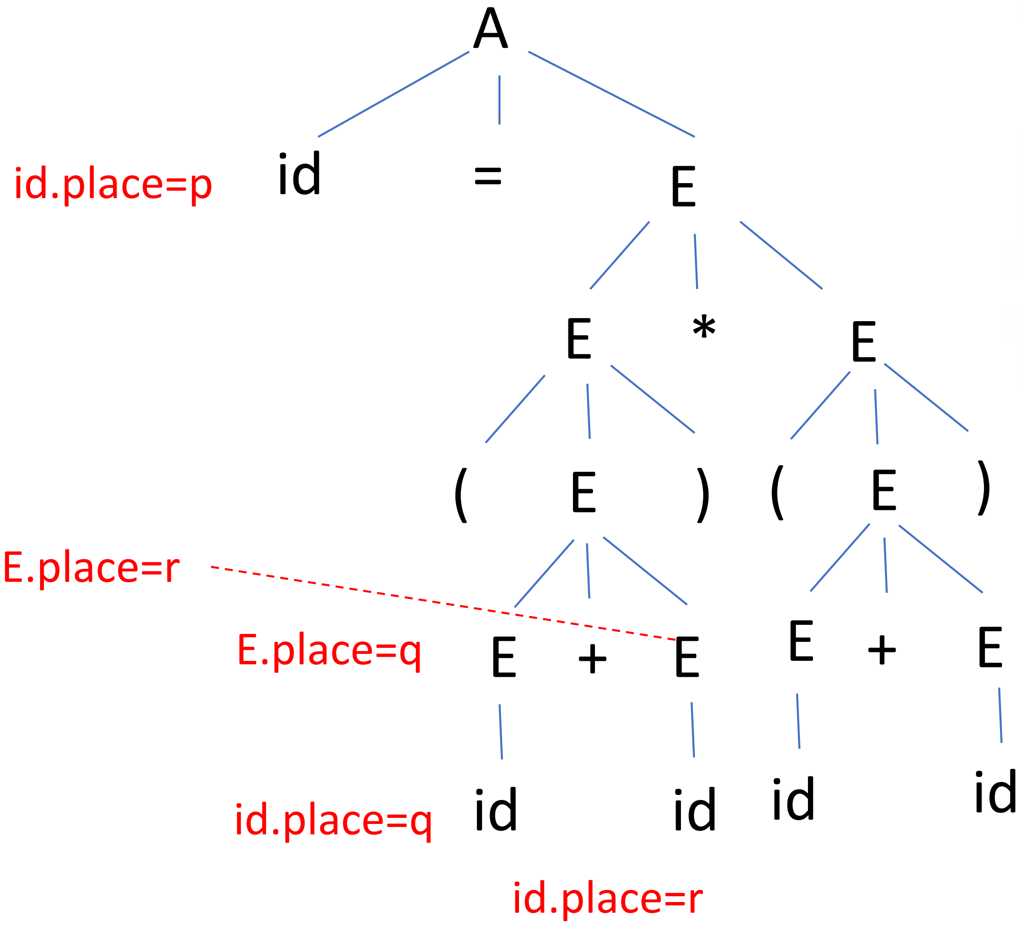
E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

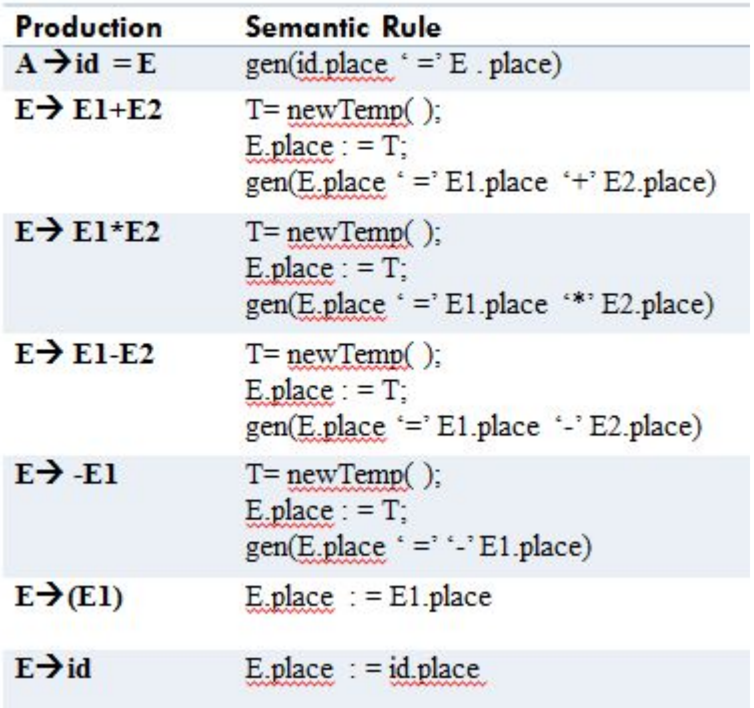
E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place



Example 1

p=(q+r)*(s+t)



Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

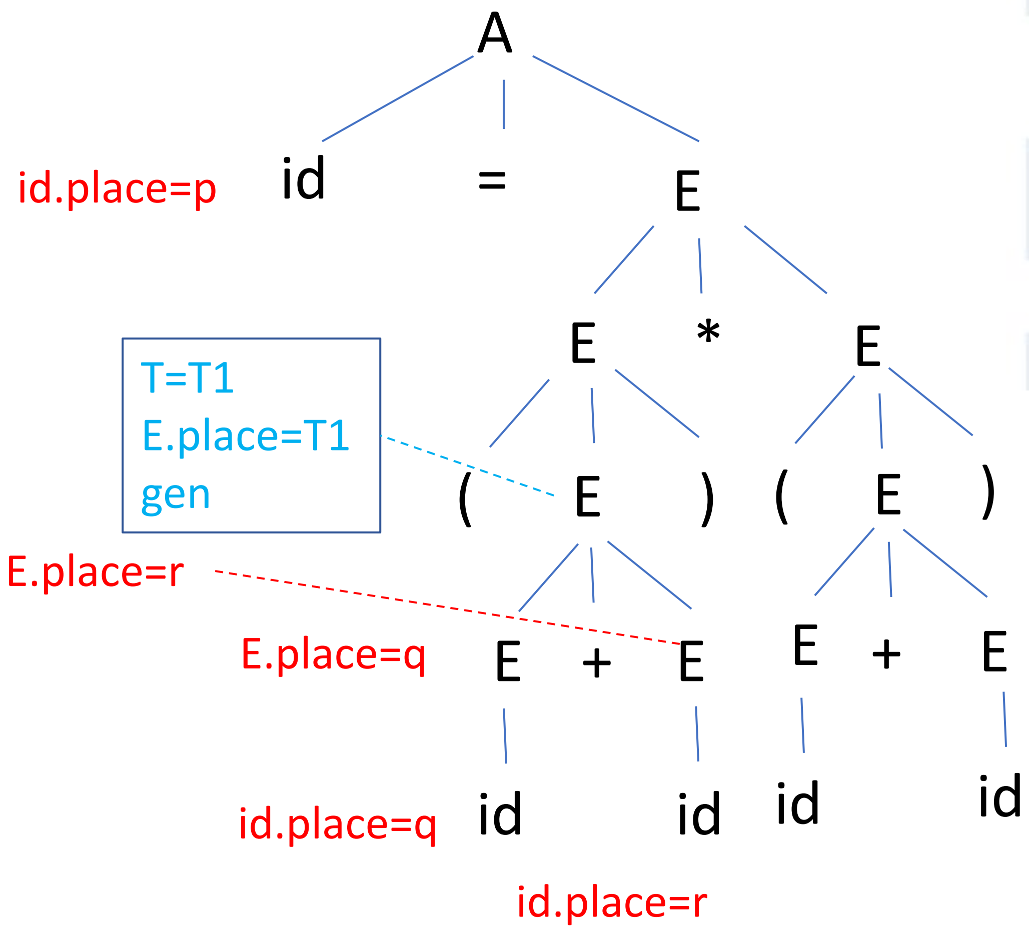
E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place

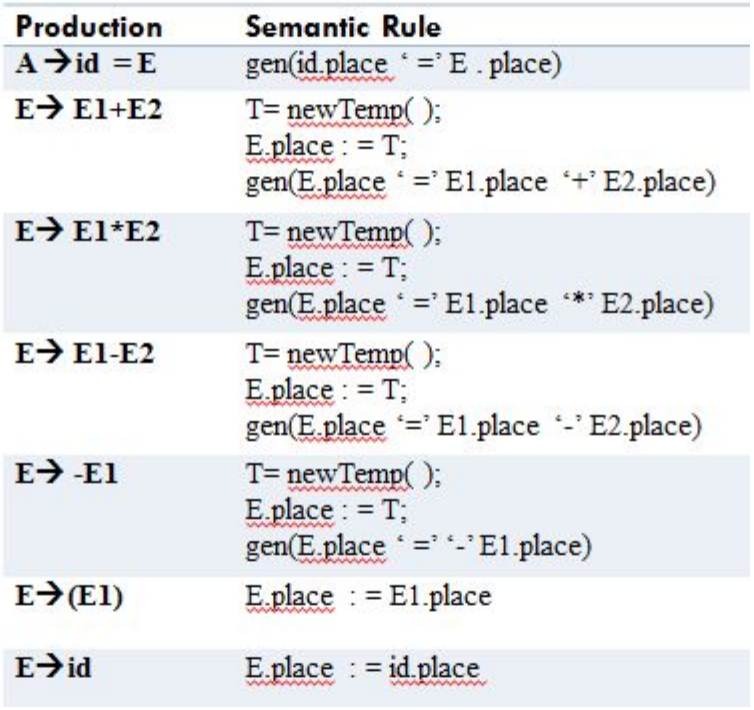


Three address code:

100) T1=q+r

Example 1

p=(q+r)*(s+t)



Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

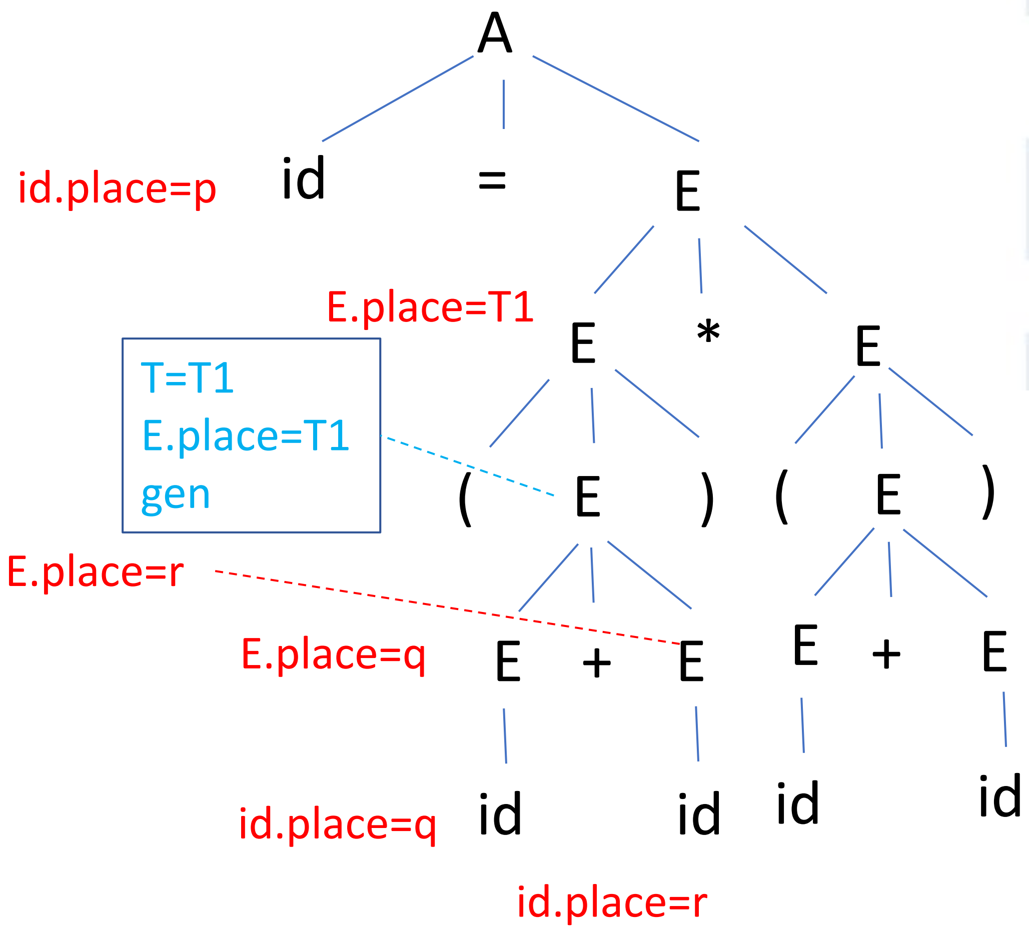
E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place

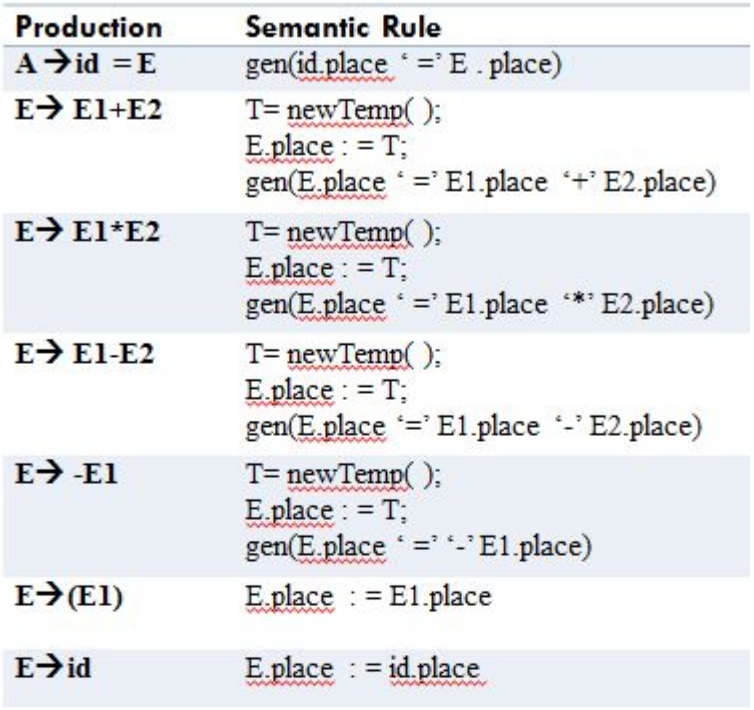


Three address code:

100) T1=q+r

Example 1

p=(q+r)*(s+t)



Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

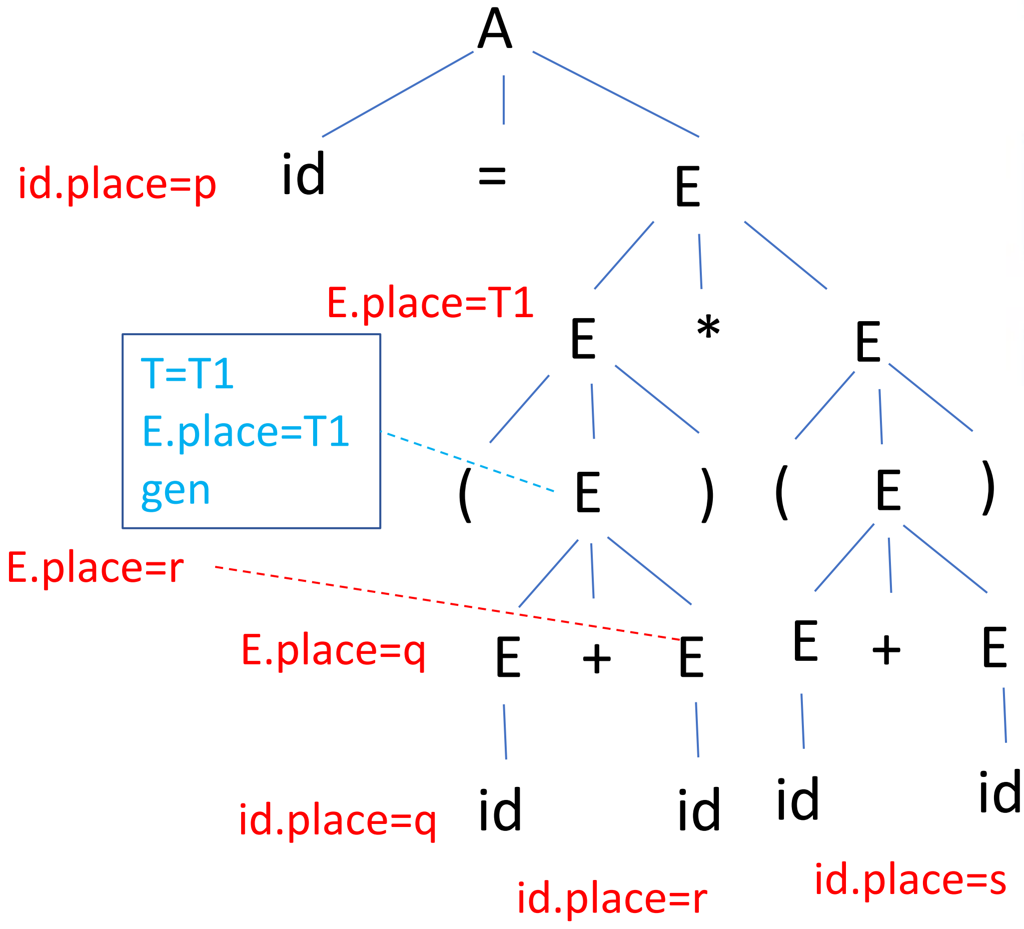
E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place

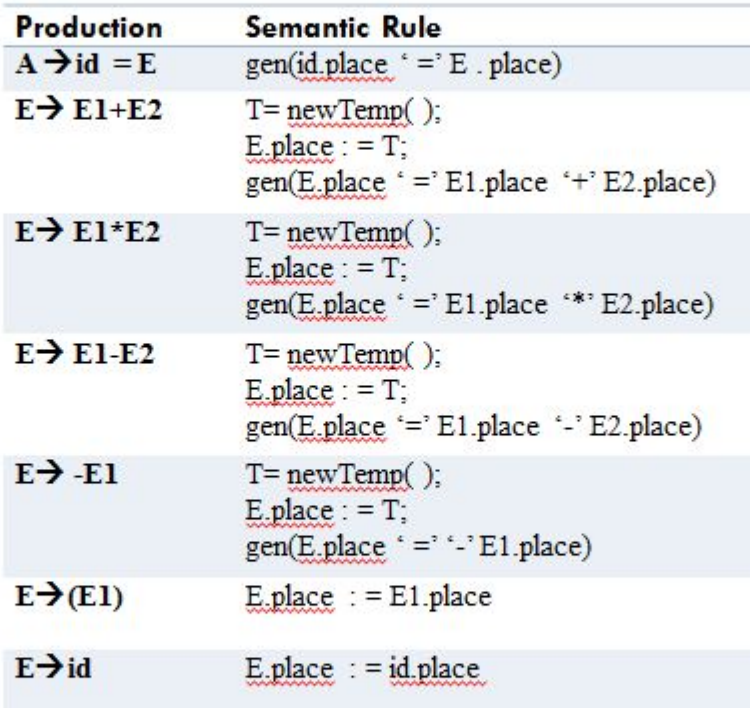


Three address code:

100) T1=q+r

Example 1

p=(q+r)*(s+t)



Production	Semantic Rule
A→id =E	gen(id.place ' =' E . place)

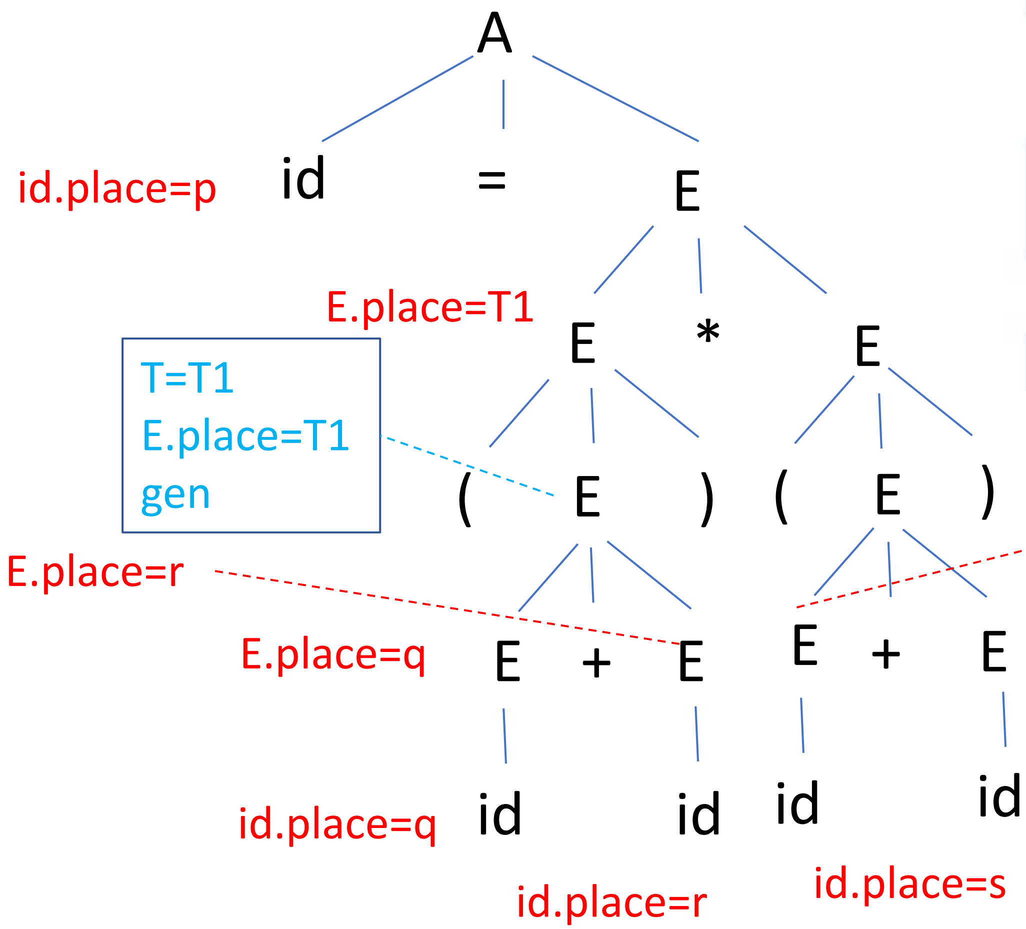
E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ' =' E1.place '+' E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ' =' E1.place '*' E2.place)

E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place '=' E1.place '-' E2.place)

E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ' =' '-' E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place

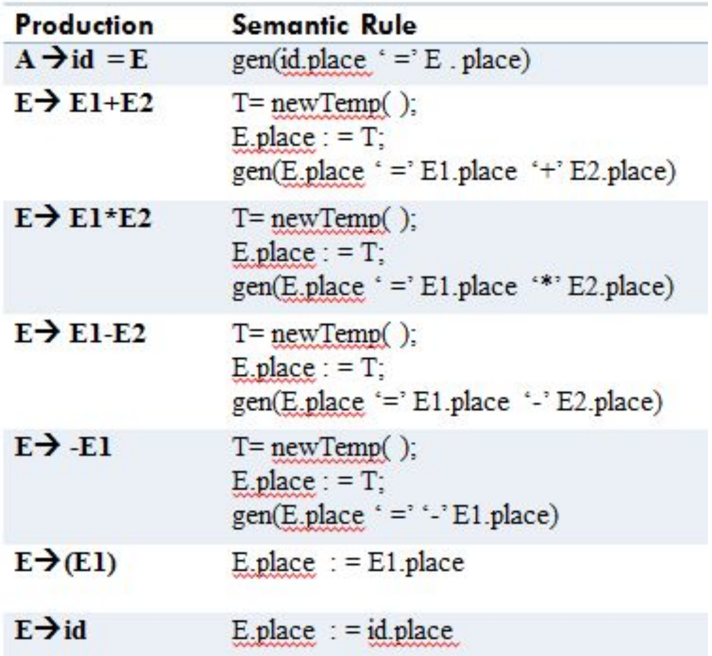


Three address code:

100) T1=q+r

Example 1

p=(q+r)*(s+t)



Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

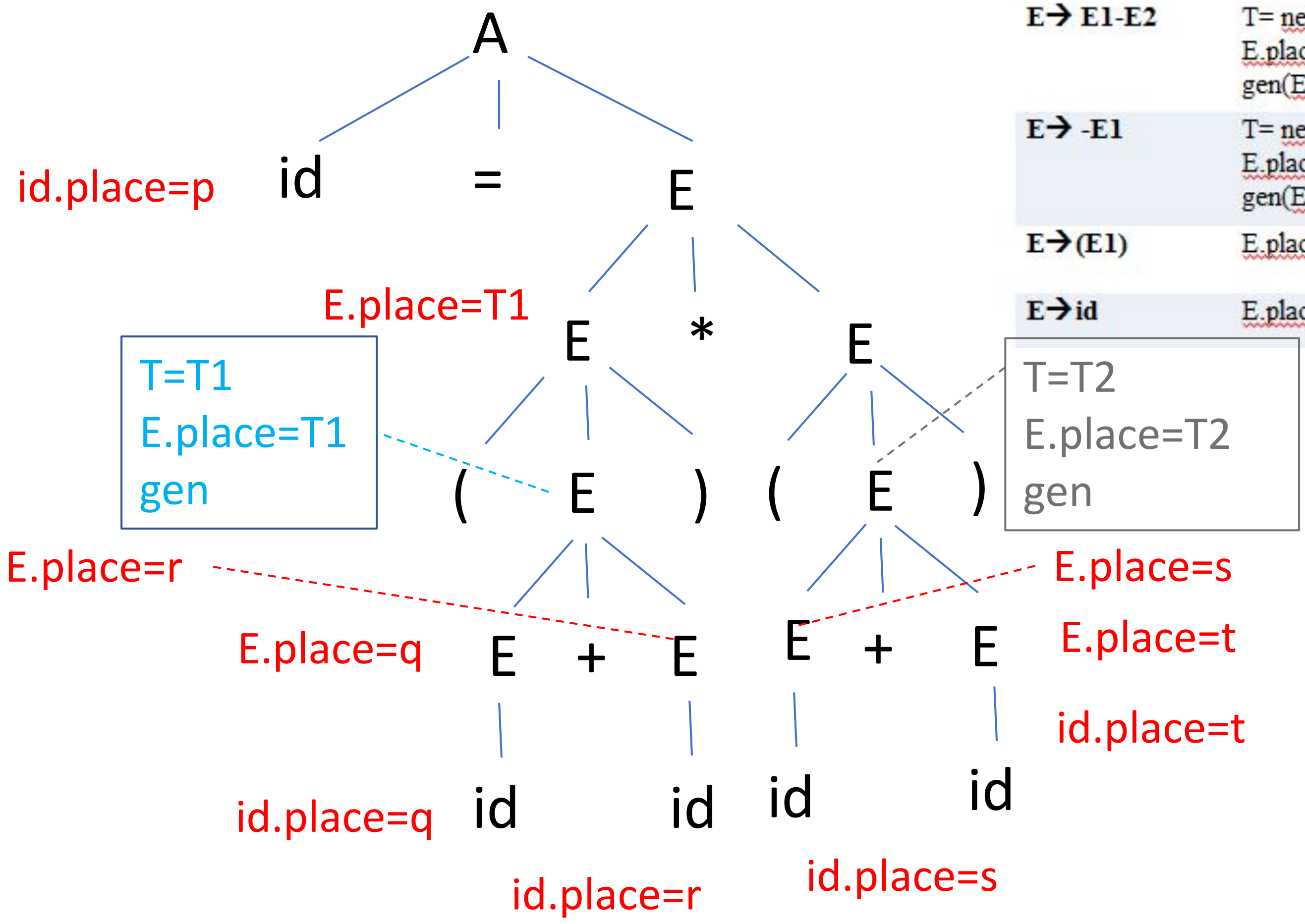
E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place



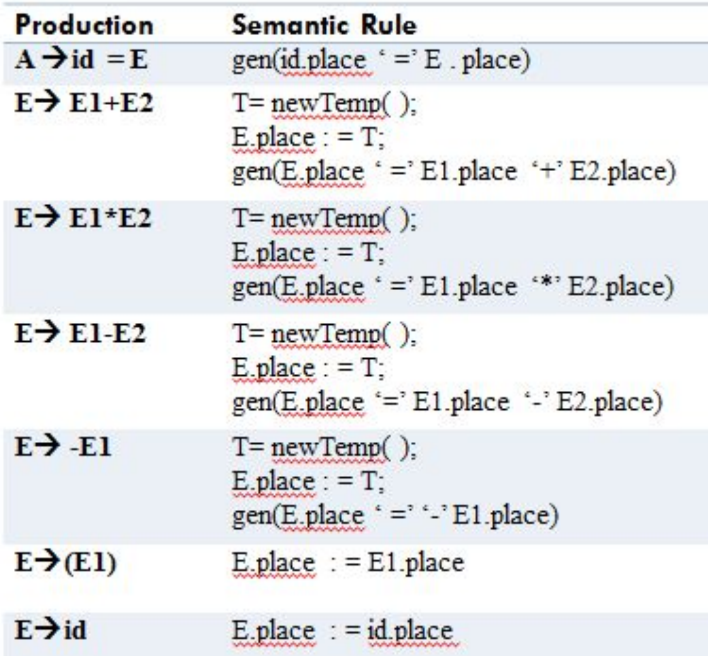
Three address code:

100) T1=q+r

101) T2=s+t

Example 1

p=(q+r)*(s+t)



Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

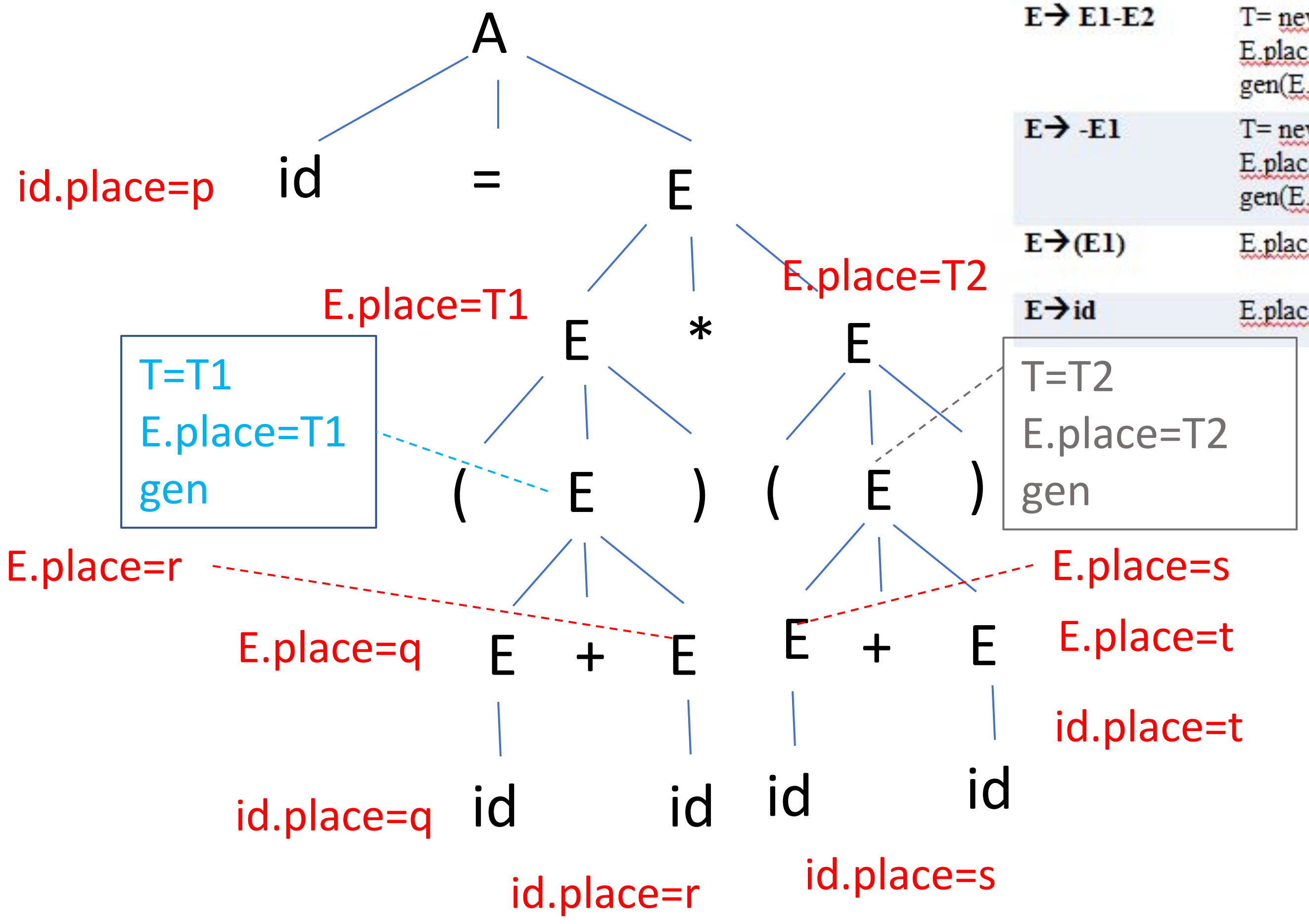
E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place



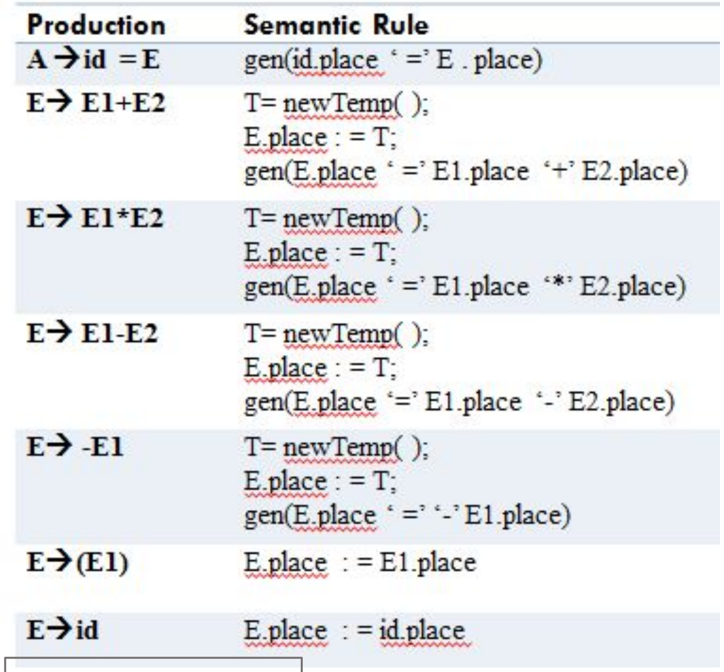
Three address code:

100) T1=q+r

101) T2=s+t

Example 1

p=(q+r)*(s+t)



Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

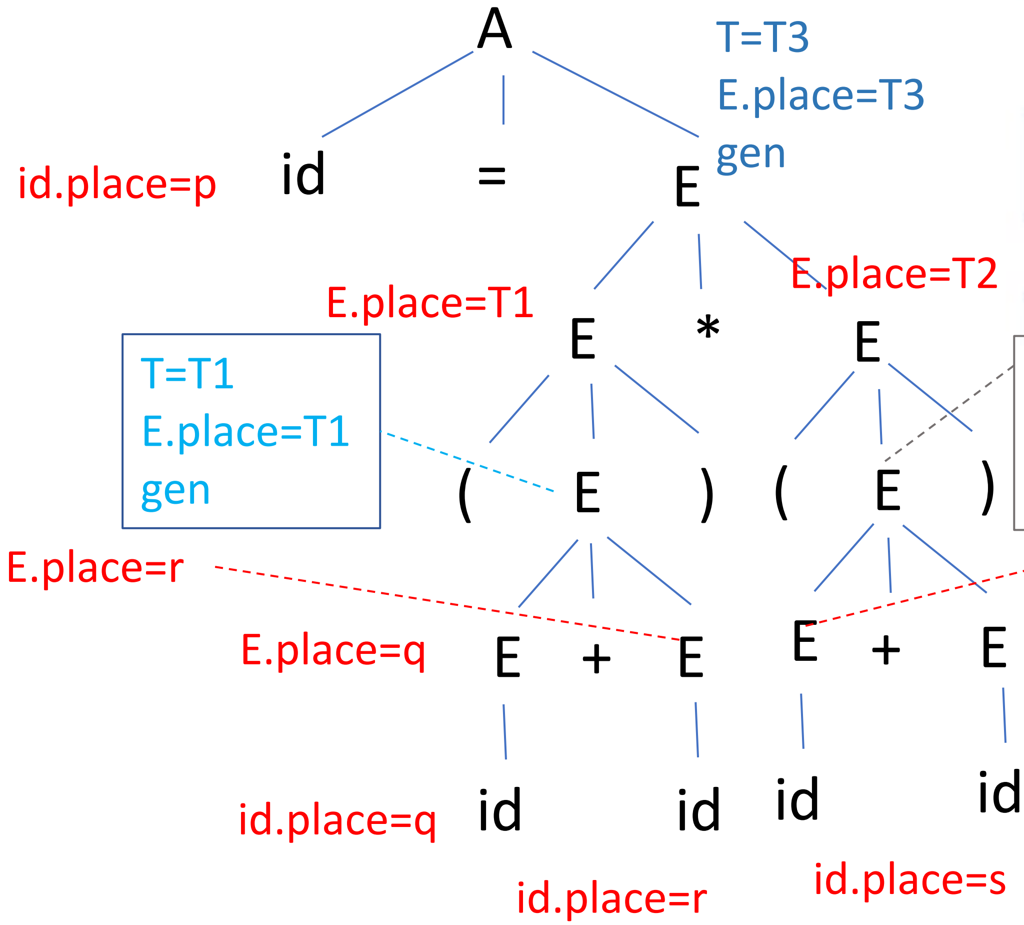
E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place



T=T2
E.place=T2
gen

E.place=s

E.place=t

id.place=t

Three address code:

100) T1=q+r

101) T2=s+t

102) T3=T1*T2

Example 1

p=(q+r)*(s+t)

Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

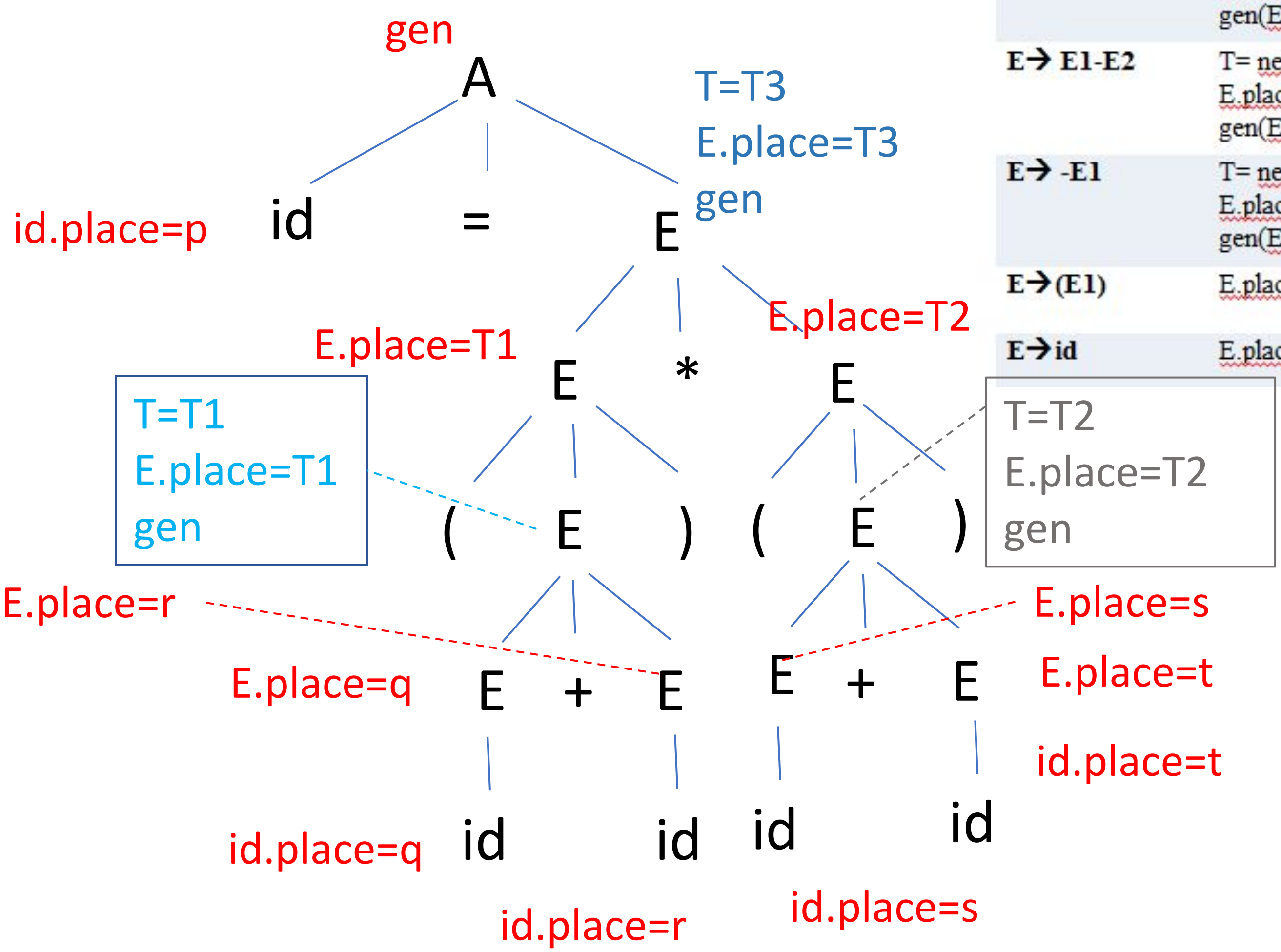
E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place



Three address code:

100) T1=q+r

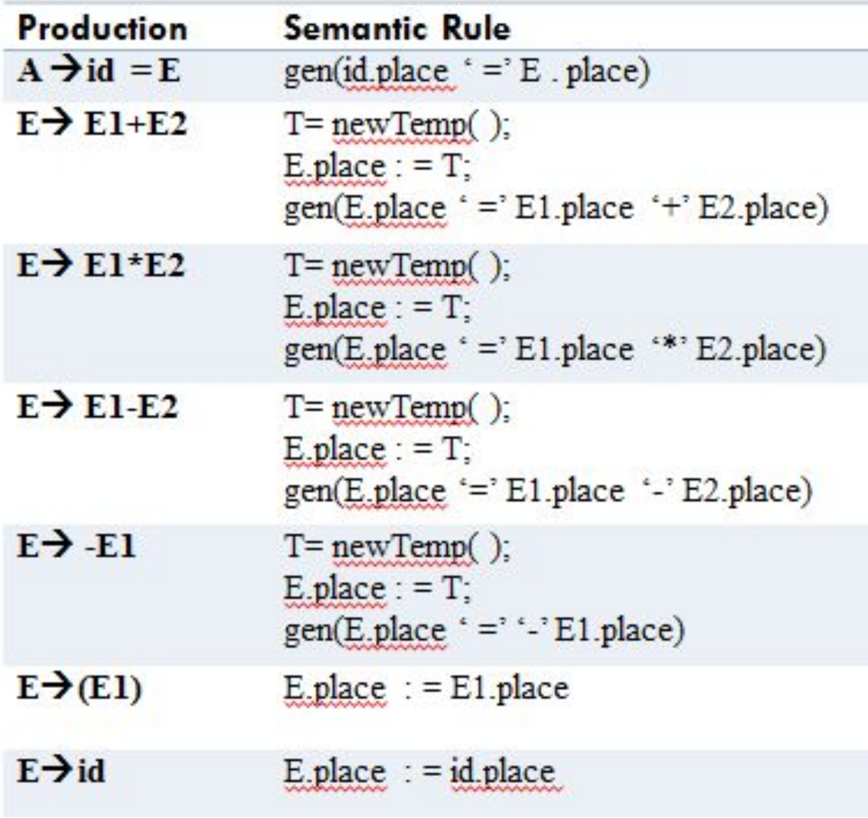
101) T2=s+t

102) T3=T1*T2

103) p=T3

Example 2

a= -c*(d+e)



Production	Semantic Rule
A→id =E	gen(id.place ‘ =’ E . place)

E→ E1+E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘+’ E2.place)

E→ E1*E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ E1.place ‘*’ E2.place)

E→ E1-E2	T= newTemp( );
	E.place : = T;
	gen(E.place ‘=’ E1.place ‘-’ E2.place)

E→ -E1	T= newTemp( );
	E.place : = T;
	gen(E.place ‘ =’ ‘-’ E1.place)

E→(E1)	E.place : = E1.place
E→id	E.place : = id.place

Example 2

a= -c*(d+e)

Three address Code:

```
100) T1= - c
101) T2=d+e
102) T3=T1*T2
103) a=T3
```