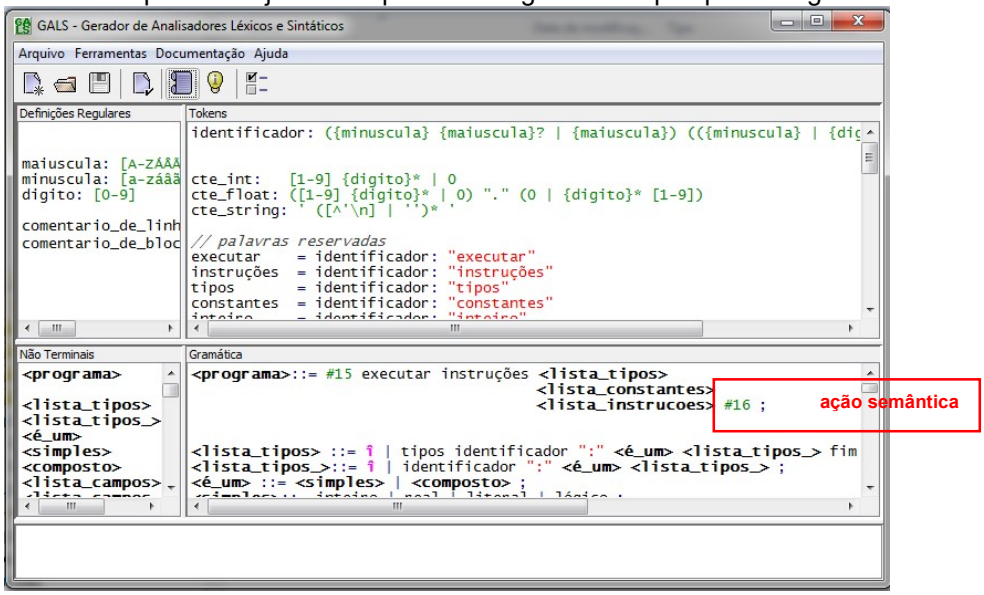
ESQUEMA DE TRADUÇÃO nº1 - parcial
(para implementação do analisador semântico e gerador de código)

1º passo: abra o arquivo que contém as especificações dos *tokens* e das regras sintáticas da linguagem (gramática usada para a implementação do analisador sintático).

2º passo: se necessário, efetue as correções na especificação dos *tokens* e das regras sintáticas, conforme indicado no relatório de avaliação dos analisadores léxico e sintático.

3º passo: coloque a numeração das ações semânticas na gramática, considerando o esquema de tradução nº1 - parcial, que possui não determinismo à esquerda. Veja o exemplo de uma gramática qualquer na figura abaixo.




```
<programa>::= #15 def <definição_tipos> <definição_constantes> <declaração_variáveis>
              execute <lista_comandos> #16
```

```
...
<lista comandos>    ::= <comando> | <comando> <lista_comandos>
<comando>           ::= ... | <saída> | ...
...
<saída>             ::= print ( <lista_expressões> ) | println ( <lista_expressões> ) #17
<lista_expressões>  ::= <expressão> #14 | <expressão> #14 , <lista_expressões>
```

```
<expressão>         ::= <elemento> <expressão_>
<expressão_>        ::= ε | && <elemento> #18 <expressão_> | || <elemento> #19 <expressão_>
<elemento>          ::= <relacional> | true #11 | false #12 | ! <elemento> #13
<relacional>        ::= <aritmetica> <relacional_>
<relacional_>       ::= ε | <operador_relacional> #9 <aritmetica> #10
<operador_relacional>::= = | != | < | <= | > | >=
<aritmética>        ::= <termo> <aritmética_>
<aritmética_>       ::= ε | + <termo> #1 <aritmética_> | - <termo> #2 <aritmética_>
<termo>             ::= <fator> <termo_> ;
<termo_>            ::= ε | * <fator> #3 <termo_> | / <fator> #4 <termo_>
<fator>             ::= identificador   |
                        identificador get ( identificador ) |
                        cte_int   #5    |
                        cte_float #6    |
                        cte_str   #20   |
                        ( <expressão> ) |
                        + <fator> #7    |
                        - <fator> #8
```

4º passo: uma vez que a gramática esteja alterada e as ações semânticas corretamente colocadas, gere novamente os analisadores léxico, sintático e semântico para refletir na implementação as alterações feitas. Observa-se que, em geral, o único código alterado pelo GALS é o das constantes (em Java - `ScannerConstants.java`, `ParserConstants.java`, `Constants.java`).

5º passo: implemente os registros semânticos e as ações semânticas que constituem o analisador semântico e o gerador de código, conforme explicado em aula. Algumas das ações semânticas (de `#1` a `#16`), especificadas em sala, **deverão ser alteradas** para atender a semântica da linguagem 2018.1, conforme descrito a seguir, principalmente no que diz respeito à verificação de tipos.

6º passo: valide o código objeto gerado. Para tanto, utilize o `ilasm` para gerar o executável a partir do código objeto gerado e, em seguida, execute o arquivo executável.

DESCRIÇÃO DOS REGISTROS SEMÂNTICOS: para executar a análise semântica e a geração de código é necessário fazer uso de registros semânticos (outros podem e devem ser definidos, bem como os descritos abaixo podem ser alterados, conforme a implementação das ações semânticas). Tem-se:

- **`operador`**: usado para armazenar o operador relacional reconhecido pela **ação #9**, para uso posterior na **ação #10**.
- **`código`**: usado para armazenar o código objeto gerado.
- **`pilha_tipos`**: usada para determinar o tipo de uma `<expressão>`.

DESCRIÇÃO DAS VERIFICAÇÕES SEMÂNTICAS: tem-se:

✓ O tipo de uma `<expressão>` deve ser determinado da seguinte forma:

operando$_1$	operando$_2$	operador	tipo da expressão resultante
`cte_int`			**`int64`**
`cte_float`			**`float64`**
`cte_str`			**`string`**
`true`			**`bool`**
`false`			**`bool`**
`int64`		`operadores unários : + -`	**`int64`**
`int64`	**`int64`**	`operadores binários: + - * /`	**`int64`**
`float64`		`operadores unários : + -`	**`float64`**
`int64` ou **`float64`**	**`int64`** ou **`float64`**	`operadores binários: + - * /` pelo menos um operando do tipo **`float64`**	**`float64`**
`int64` ou **`float64`**	**`int64`** ou **`float64`**	**`= != < <= > >=`**	**`bool`**
`str`	**`str`**	**`= != < <= > >=`**	**`bool`**
`bool`		**`!`** (não)	**`bool`**
`bool`	**`bool`**	**`&&`** (e) **`||`** (ou)	**`bool`**

Operadores e tipos não previstos na tabela anterior indicam que a operação correspondente não pode ser executada. Assim, por exemplo, **`10 =`** `"oi"` deve gerar um erro semântico (*tipos incompatíveis em expressão relacional)*.

DESCRIÇÃO DA SEMÂNTICA: tem-se:

✓ A semântica de uma expressão (`<expressão>`) é a seguinte:
- para as constantes (`cte_int` – ação #5, `cte_float` – ação #6, `cte_str` – ação #20, **`true`** – ação #11, **`false`** – ação #12): (1) empilhar o tipo da constante na **`pilha_tipos`**; (2) gerar código para carregar o valor da constante,
- para os operadores (lógicos, relacionais, aritméticos), (1) efetuar a verificação de tipos conforme descrito na tabela anterior; (2) gerar código para efetuar a operação correspondente.

✓ A semântica do comando **`print`** é a seguinte: gerar código para escrever (na saída padrão) o resultado da avaliação de cada `<expressão>` da `<lista_expressões>`.

✓ A semântica do comando **`println`** é a seguinte (ou seja, a ação #17 deve): (1) gerar código para escrever (na saída padrão) o resultado da avaliação de cada `<expressão>` da `<lista_expressões>`, (2) gerar código para escrever `\n` (na saída padrão).

EXEMPLO DE PROGRAMA FONTE / OBJETO: programa fonte (**`teste_01.txt`**) x programa objeto (**`teste_01.il`**)

programa fonte	programa objeto

`def`
`execute`
`  print   (1, " ", 1,5)`
`  println (" ---> olá mundo!")`
	`.assembly extern mscorlib {}`
	`.assembly _codigo_objeto{}`
	`.module   _codigo_objeto.exe`
	
	`.class public _UNICA{`
	`.method static public void _principal() {`
	`  .entrypoint`
	`  ldc.i8 1`
	`  conv.r8`
	`  conv.i8`
	`  call void [mscorlib]System.Console::Write(int64)`
	`  ldstr " "`
	`  call void [mscorlib]System.Console::Write(string)`
	`  ldc.r8 1.5`
	`  call void [mscorlib]System.Console::Write(float64)`
	`  ldstr " ---> olá mundo!"`
	`  call void [mscorlib]System.Console::Write(string)`
	`  ldstr "\n"`
	`  call void [mscorlib]System.Console::Write(string)`
	`  ret`
	`  }`
	`}`