3ª Aula Prática de Compiladores: Análise Semântica com ANTLR 4

Roteiro

- Análise Semântica
- Checagem de Tipos
- Checagem de Tipos com Visitor
- Geração de Arquivos
- Exercícios Práticos

Análise Semântica

- Verifica aspectos relacionados ao significado das instruções
- Última chance para o compilador descartar programas inválidos
- Algumas tarefas dessa etapa:
  - Checagem de tipos
  - Verificação do fluxo de controle
  - Verificação de unicidade

Análise Semântica

- Por exemplo:

```
x = a + b;
```

  - O identificador x foi declarado?
  - O identificador x é uma variável?
  - Qual é o escopo da variável x?
  - Qual é o tipo da variável x?
  - O tipo da variável x é compatível com os demais identificadores e operadores da expressão?

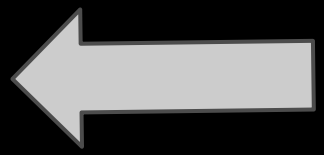
- Vamos nos focar na tarefa de Checagem de Tipos na aula de hoje

Checagem de Tipos

```
//Utilizando a semântica da linguagem Java
int a = 10;
float b = 20.5;

float c = a + b;
int d = a + b;
```

Checagem de Tipos

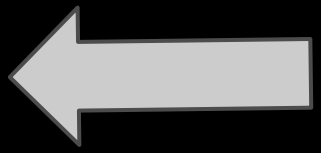
```
//Utilizando a semântica da linguagem Java
int a = 10;
float b = 20.5;

float c = a + b;
int d = a + b;
```



Conversão ampliadora:
int é promovido para float automaticamente

Checagem de Tipos

```
//Utilizando a semântica da linguagem Java
int a = 10;
float b = 20.5;

float c = a + b;
int d = a + b;
```



Conversão redutora:
float é rebaixado para int, porém essa conversão não pode ser realizada automaticamente

Checagem de Tipos

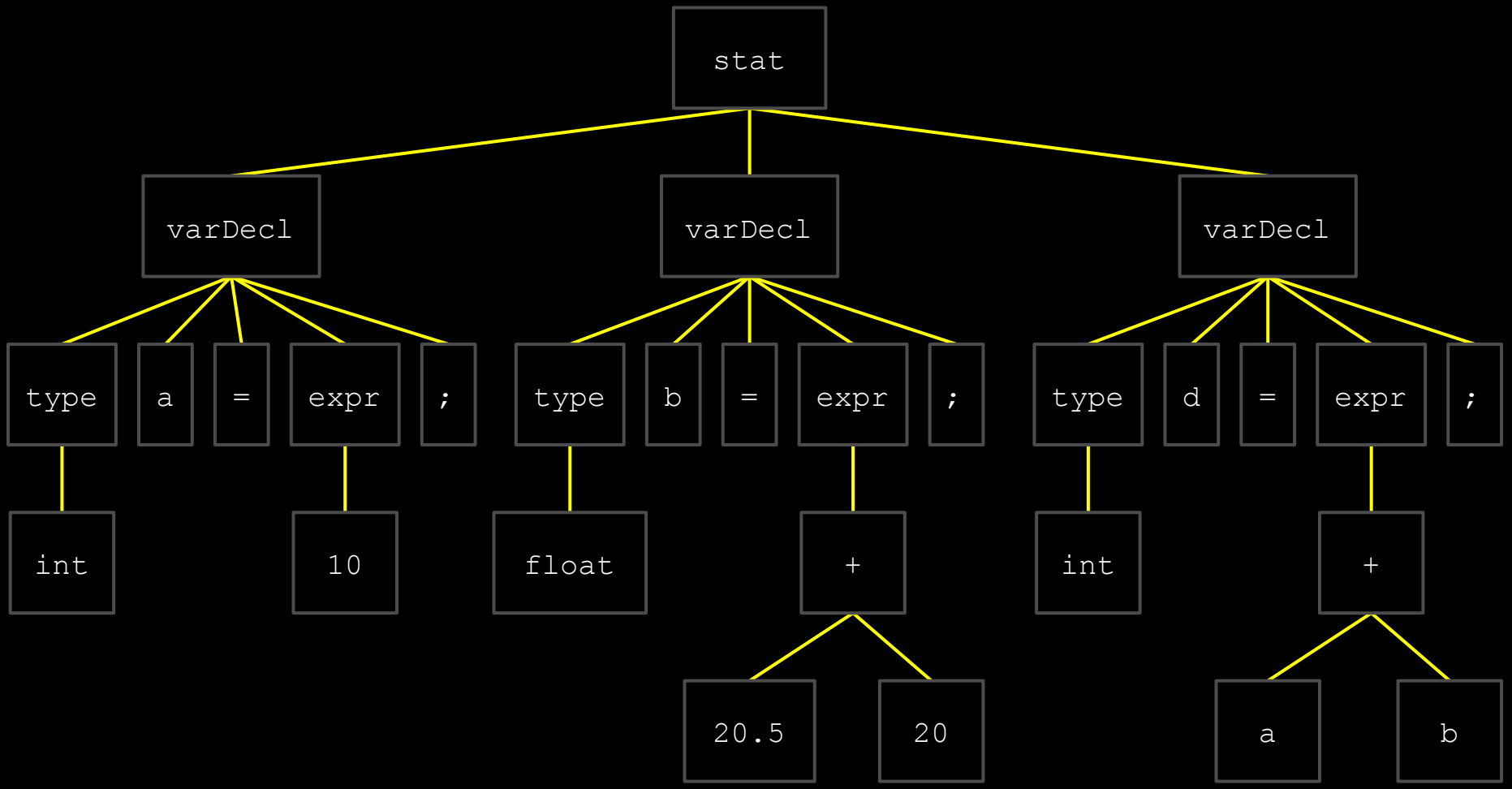
```
//Utilizando a semântica da linguagem Java
int a = 10;
float b = 20.5;

float c = a + b;  //OK
int d = a + b;    ERRO!
```

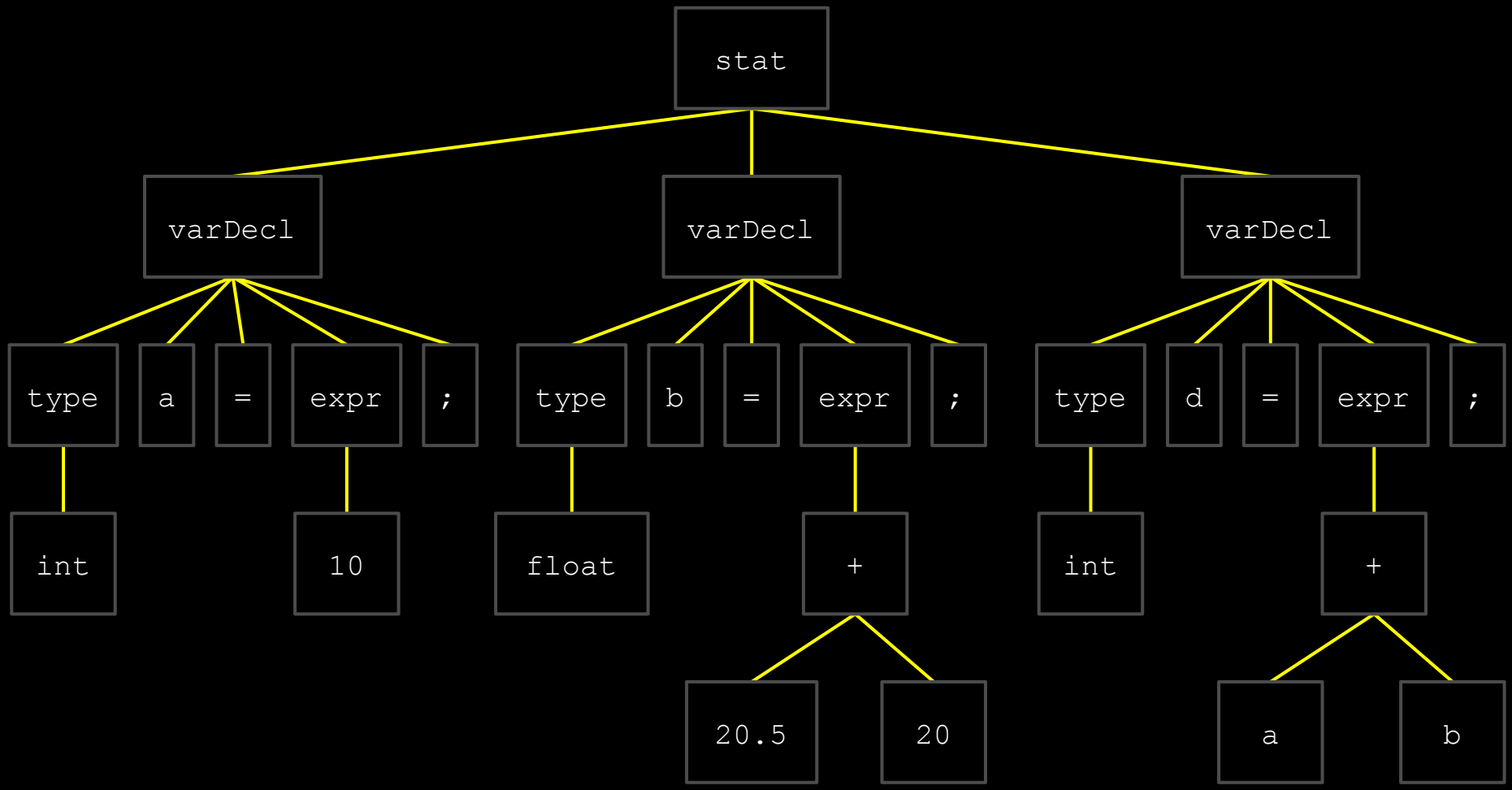
Checagem de Tipos com Visitor

- Utilizando a gramática Cymbol e o trecho de código abaixo como exemplo:

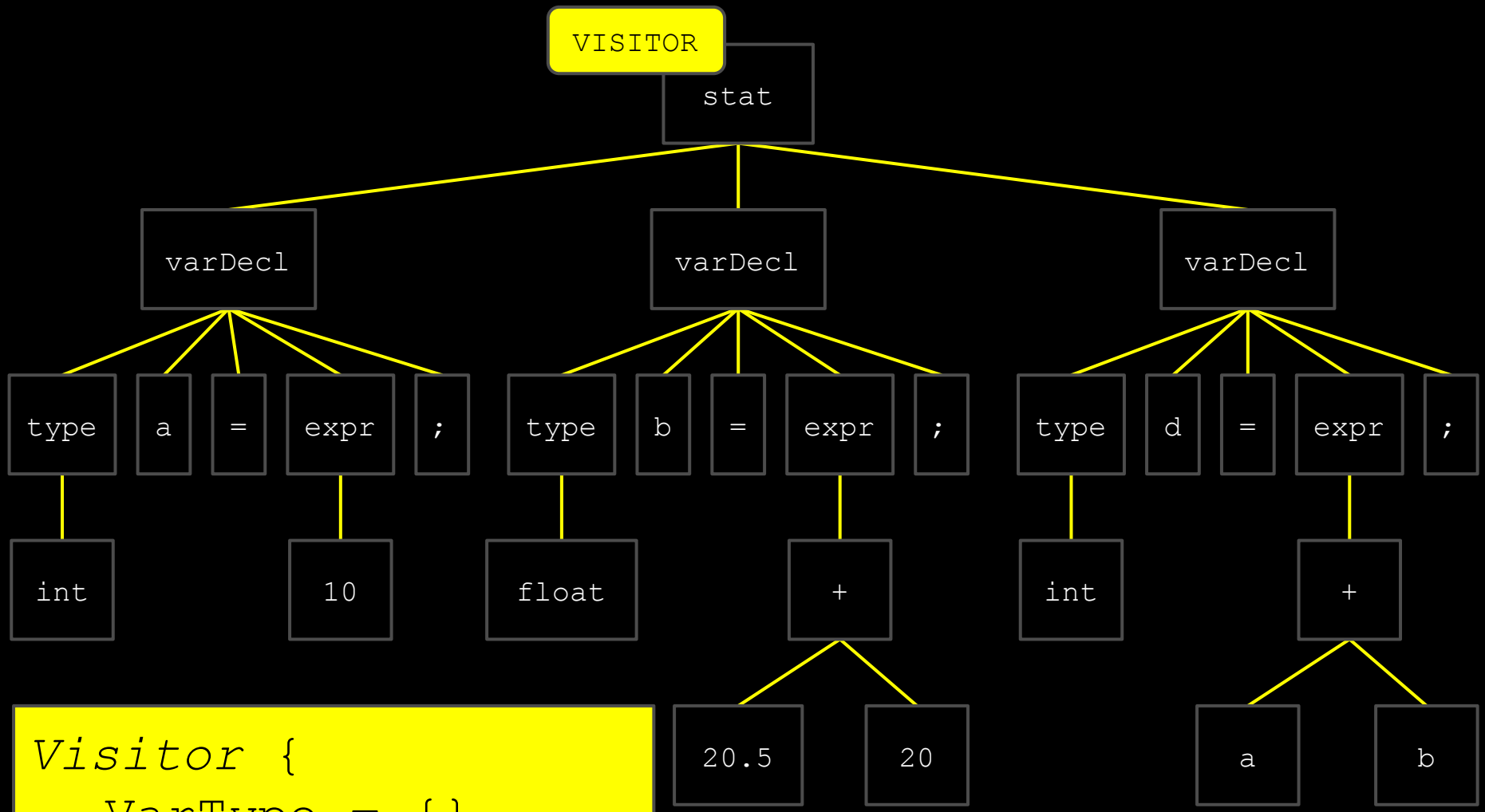
```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```

Checagem de Tipos com Visitor



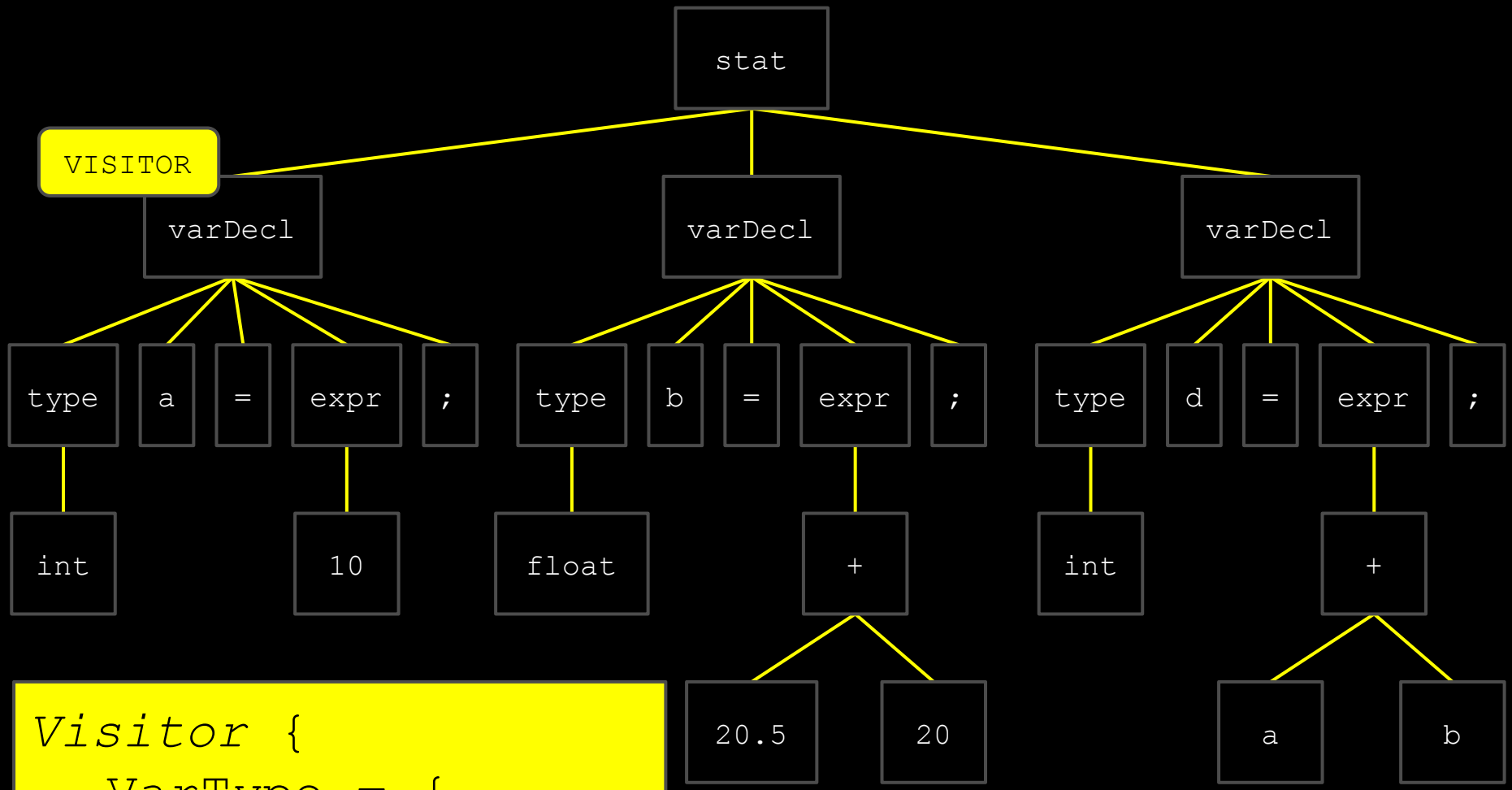



```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```



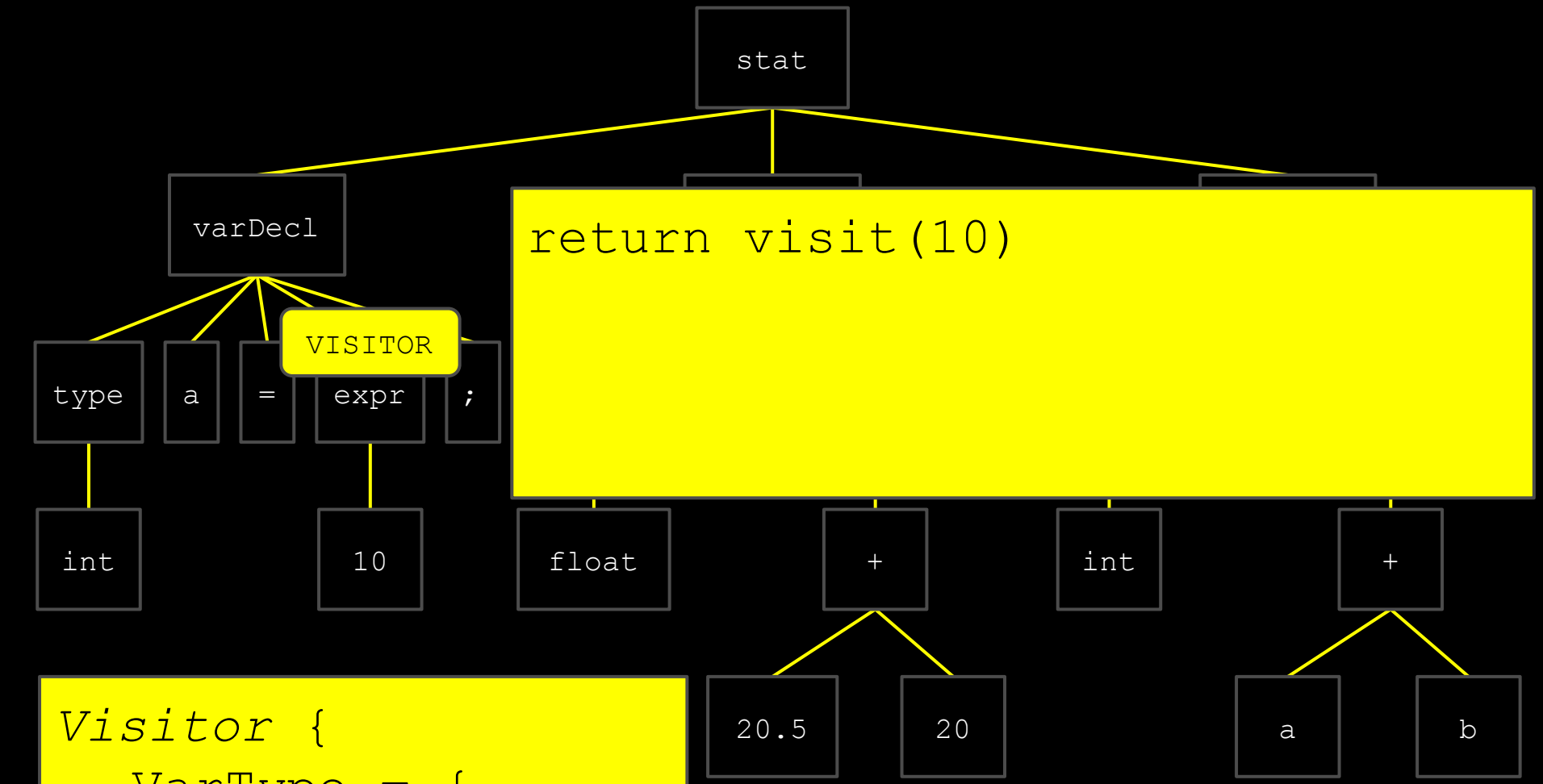

```
Visitor {
   VarType = {}
}
```

```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```



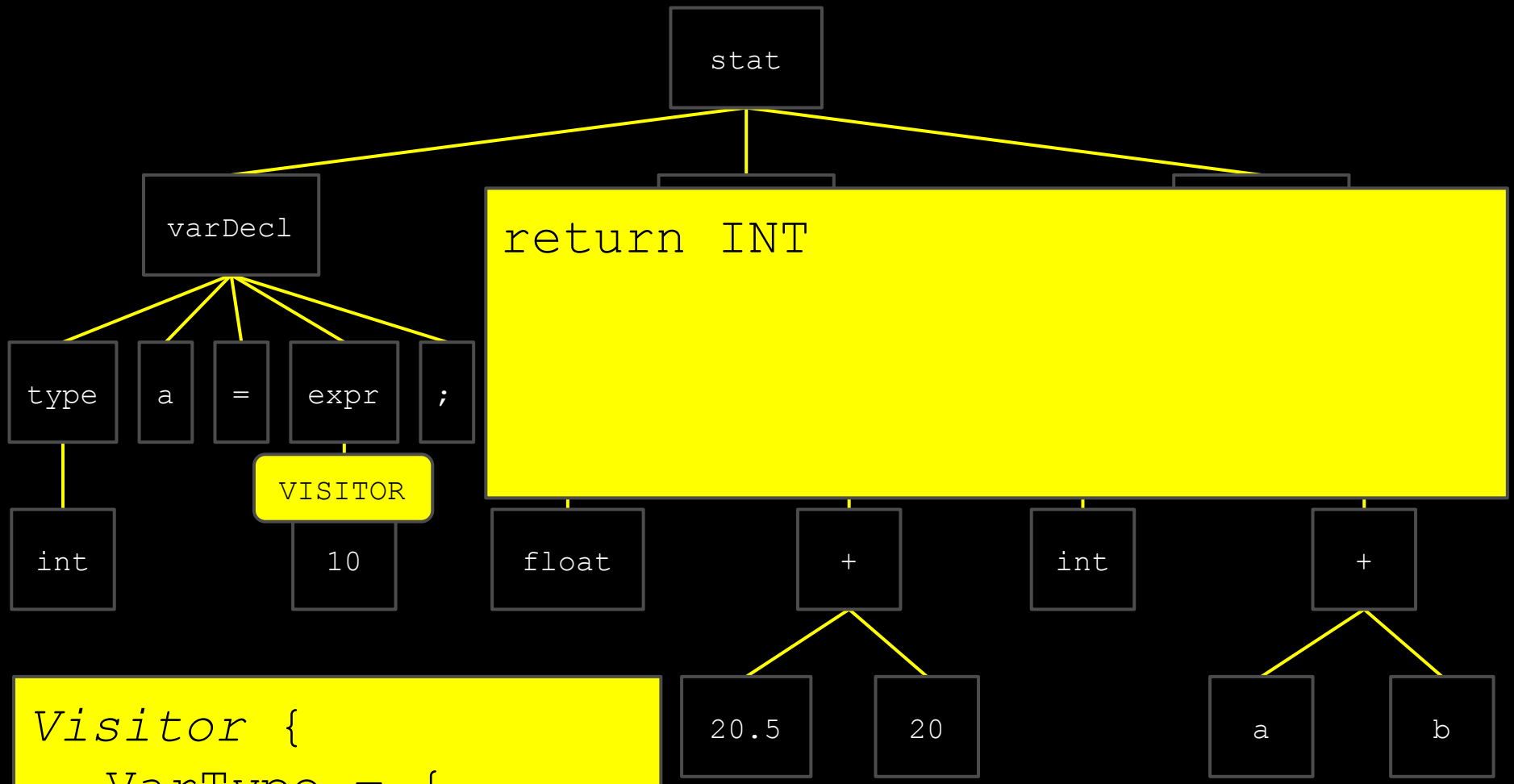

```
Visitor {
   VarType = {
      a → int
   }
}
```

```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```



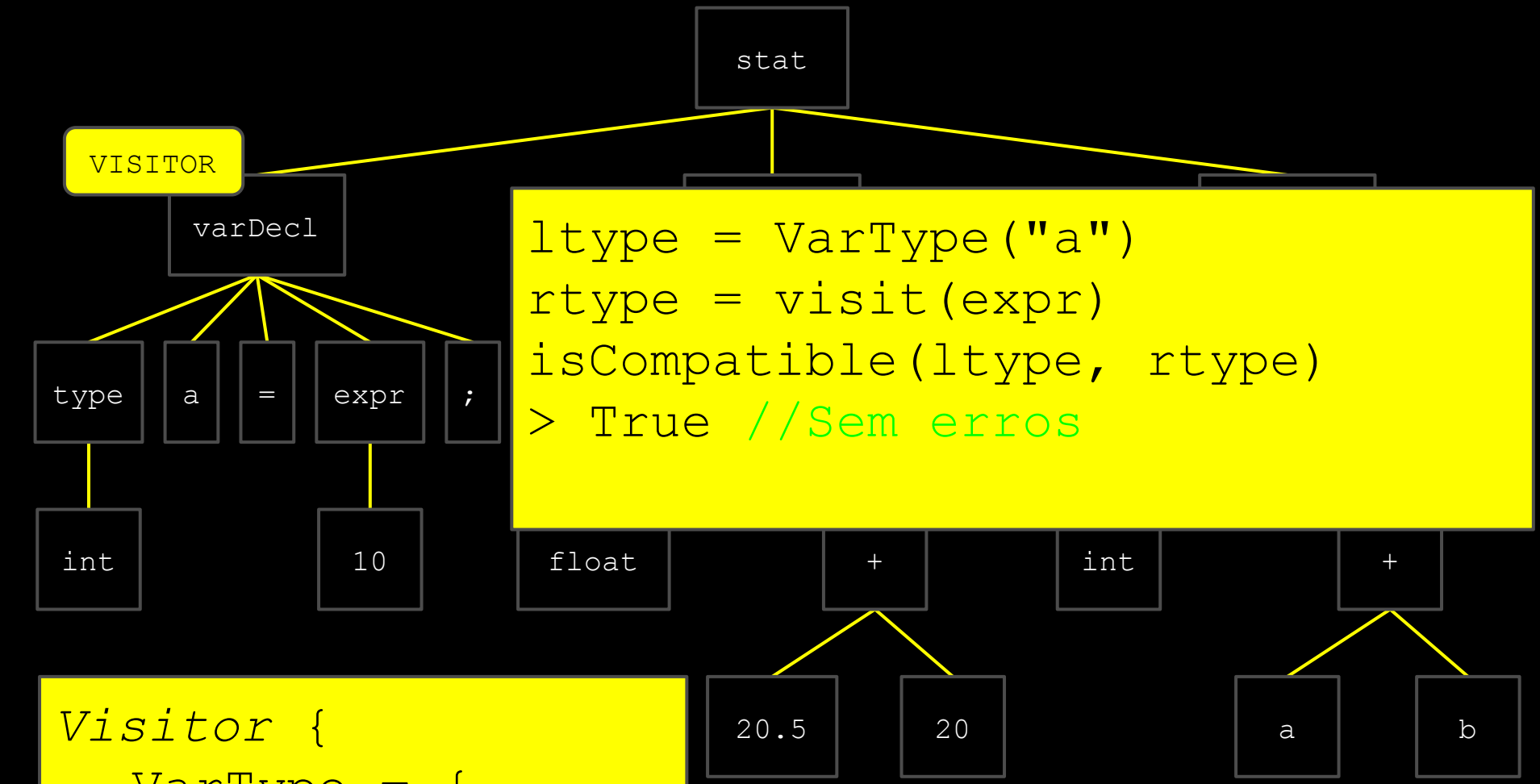

```
Visitor {
   VarType = {
      a → int
   }
}
```

```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```




```
Visitor {
   VarType = {
      a → int
   }
}
```

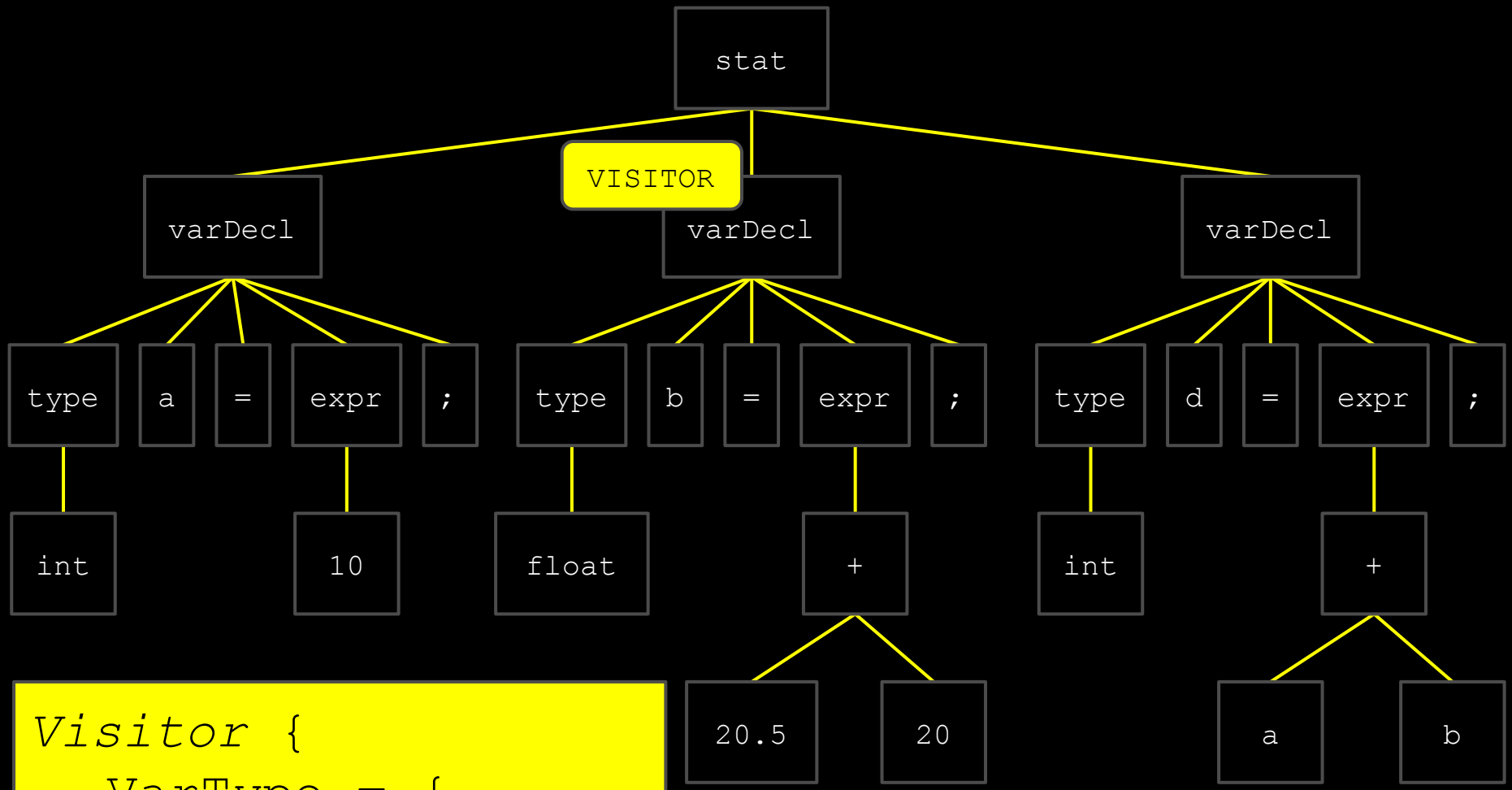
```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```




```
ltype = VarType("a")
rtype = visit(expr)
isCompatible(ltype, rtype)
> True //Sem erros
```

```
Visitor {
    VarType = {
        a → int
    }
}
```

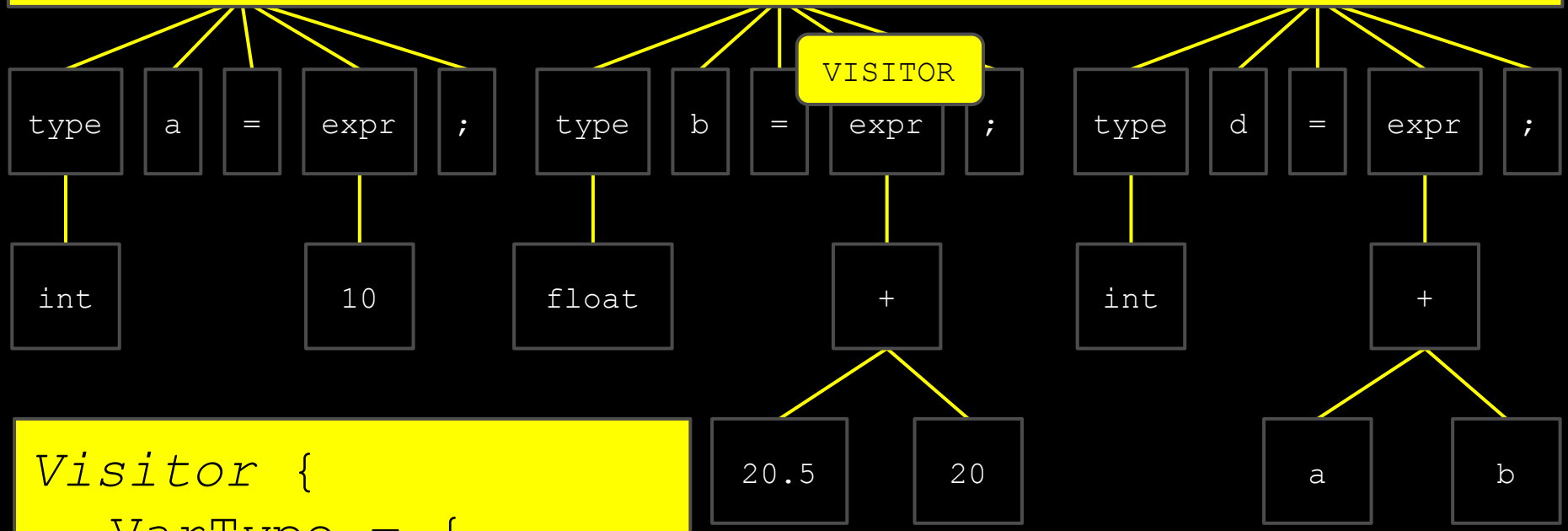
```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```




```
Visitor {
   VarType = {
      a → int,
      b → float
   }
}
```

```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```

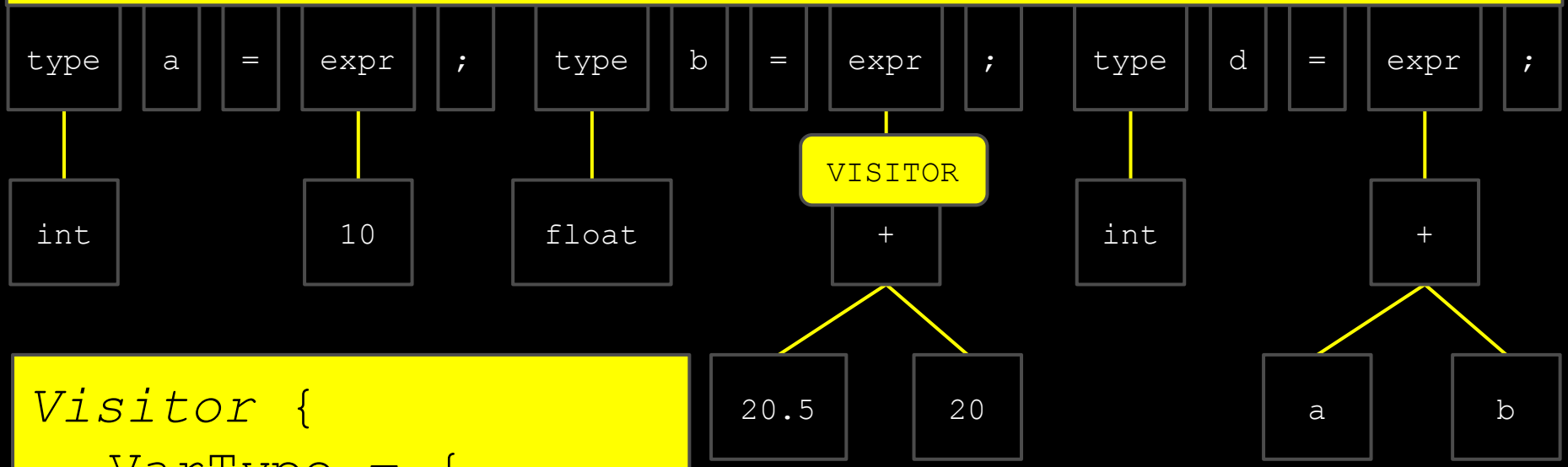
```
return visit(+)
```




```
Visitor {
   VarType = {
      a → int,
      b → float
   }
}
```

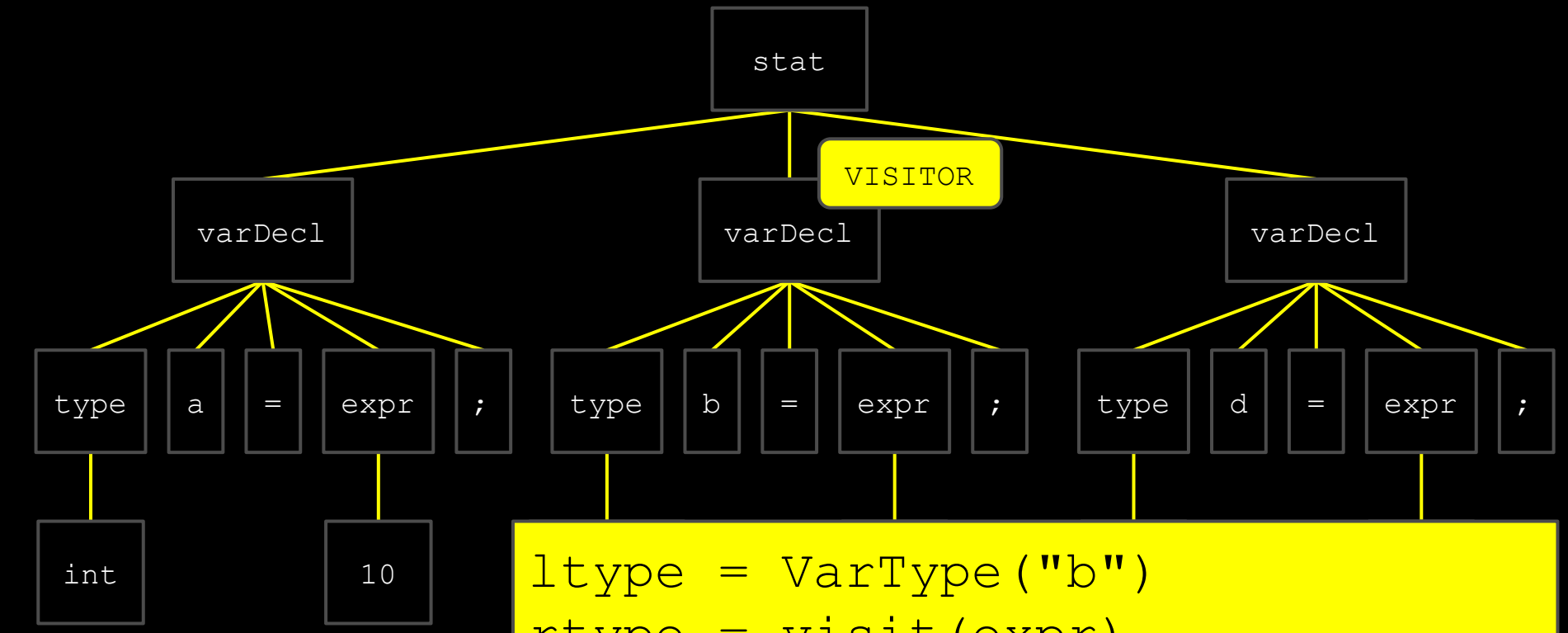
```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```

```
ltype = visit(20.5) //Retorna o tipo FLOAT
rtype = visit(20)   //Retorna o tipo INT
//FLOAT + INT → FLOAT
return checkAdd(ltype, rtype)
```




```
Visitor {
   VarType = {
      a → int,
      b → float
   }
}
```

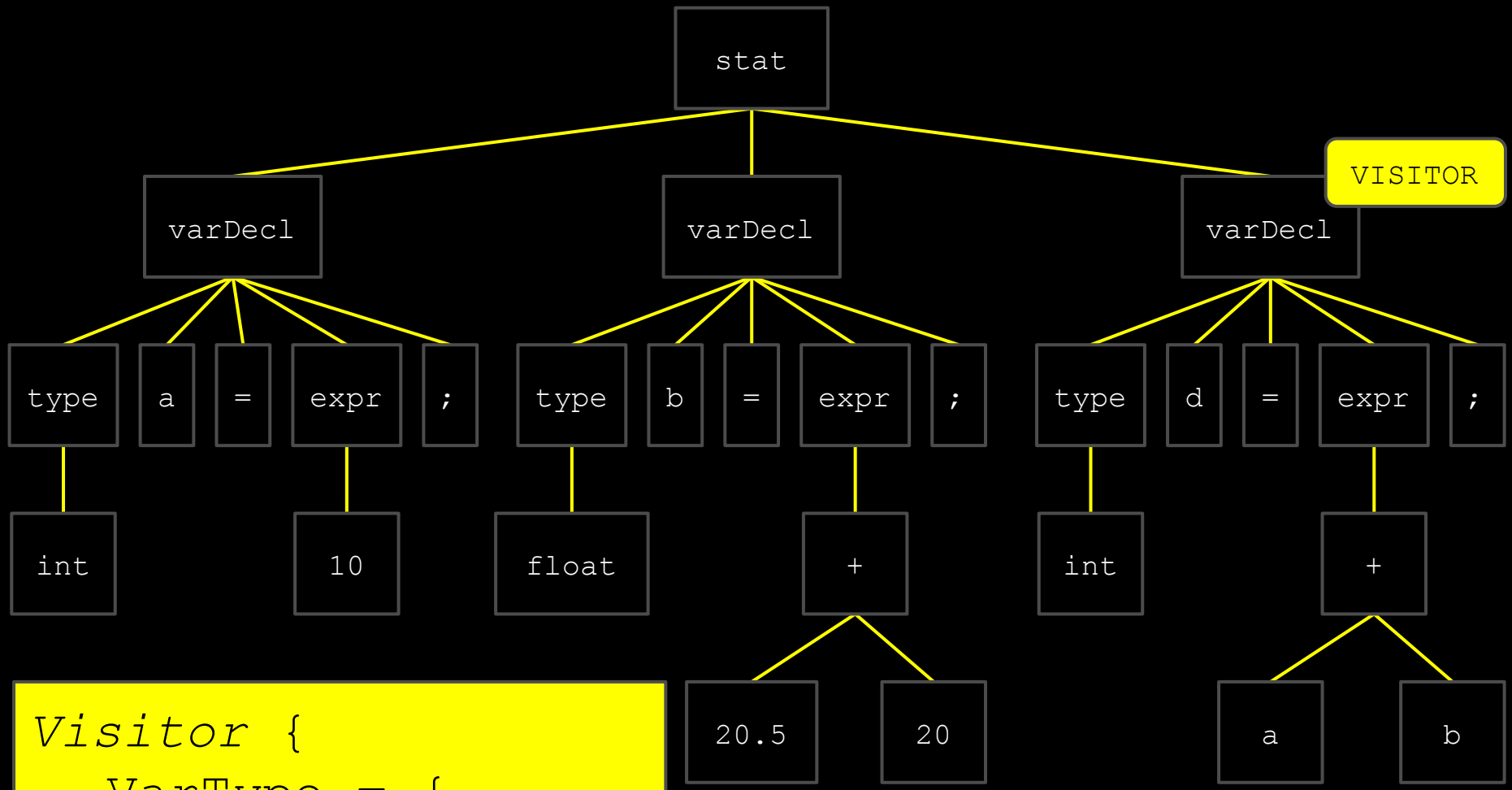
```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```




```
Visitor {
    VarType = {
        a → int,
        b → float
    }
}
```

```
ltype = VarType("b")
rtype = visit(expr)
isCompatible(ltype, rtype)
> True //Sem erros
```

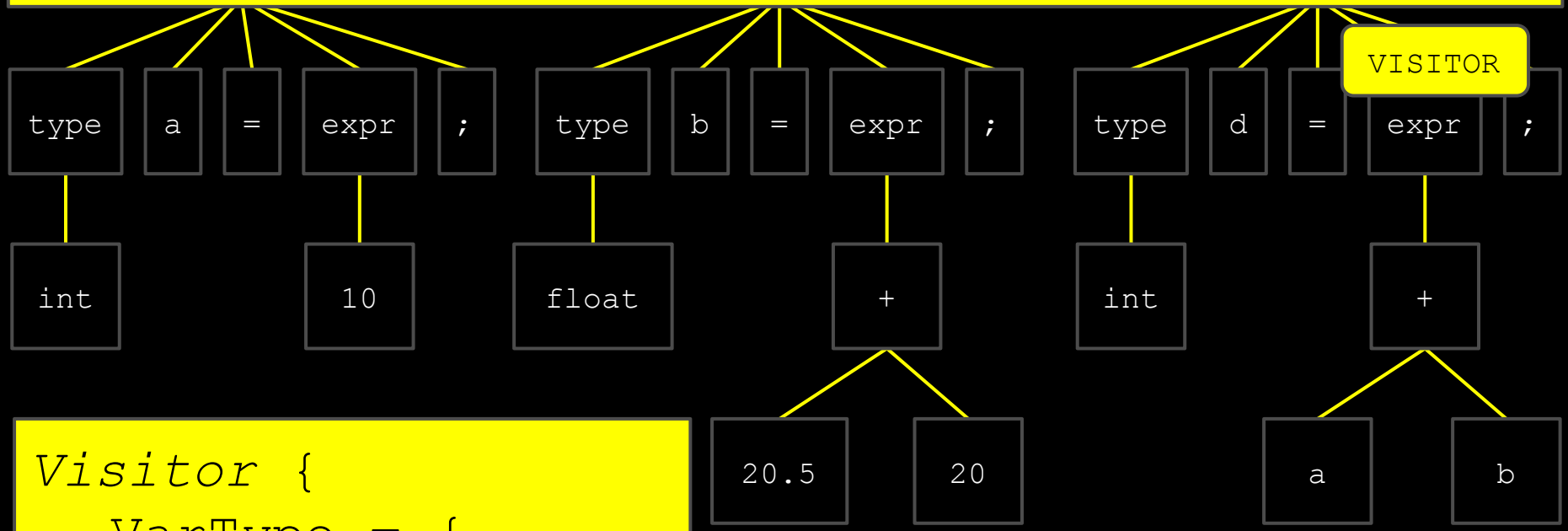
```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```




```
Visitor {
   VarType = {
      a → int,
      b → float,
      d → int
}}
```

```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```

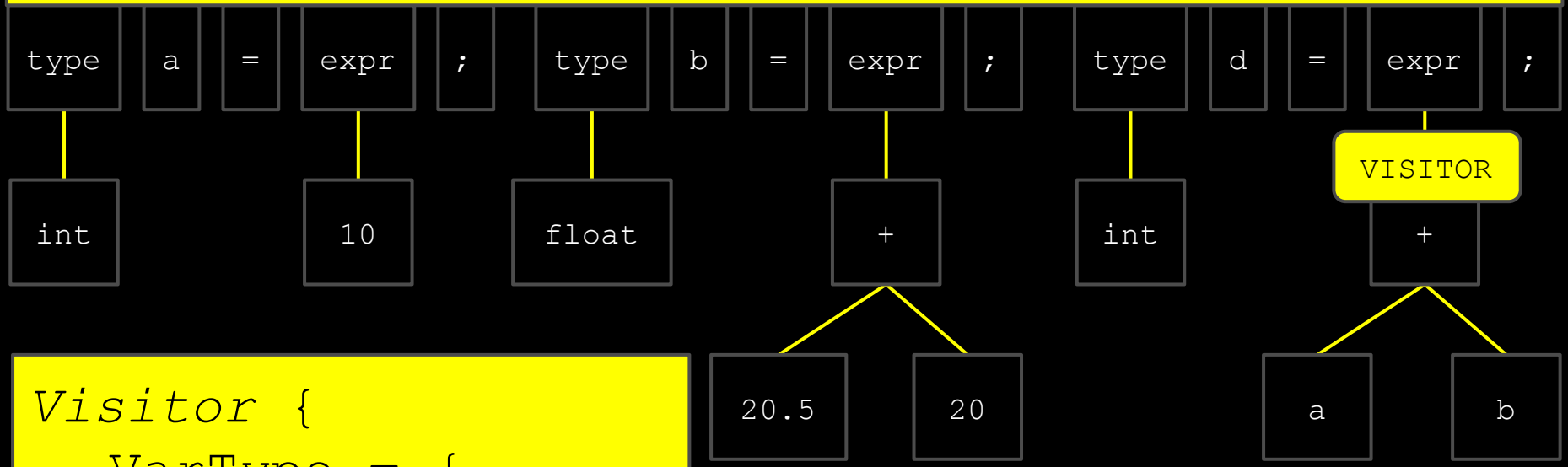
```
return visit(+)
```




```
Visitor {
   VarType = {
      a → int,
      b → float,
      d → int
}}
```

```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```

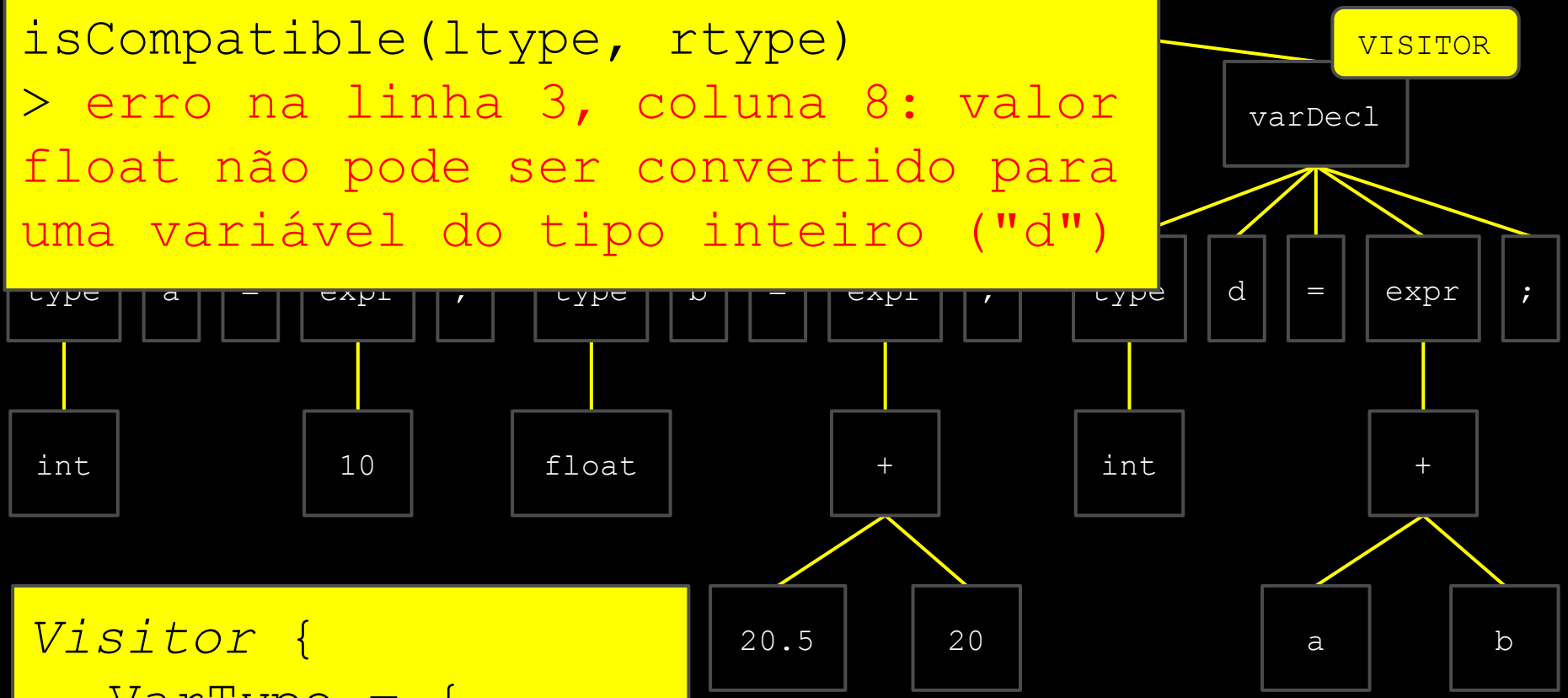
```
ltype = VarType("a")
rtype = VarType("b")
//INT + FLOAT → FLOAT
return checkAdd(ltype, rtype)
```




```
Visitor {
   VarType = {
      a → int,
      b → float,
      d → int
}}
```

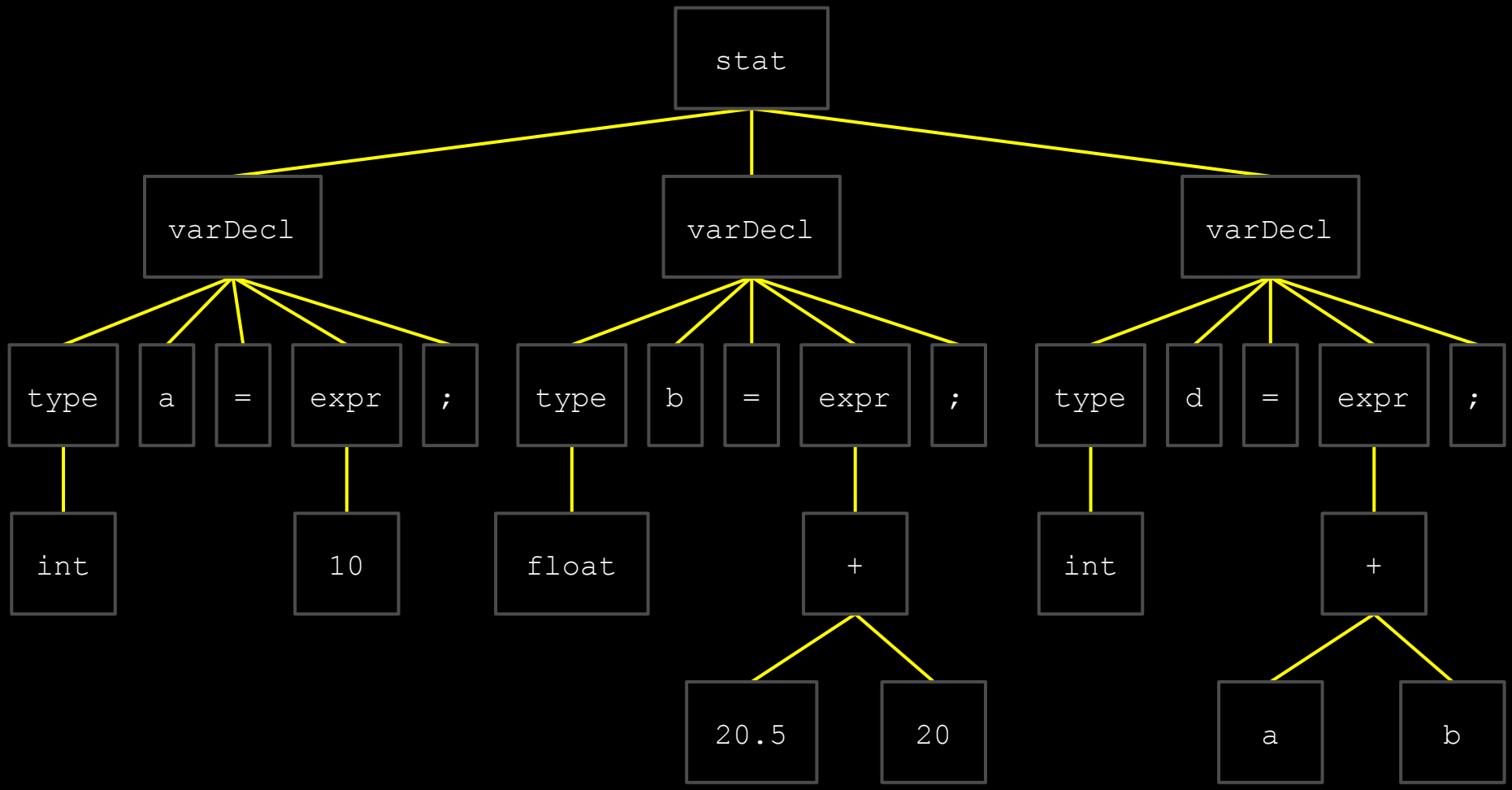
```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```

```
ltype = VarType("d")
rtype = visit(expr)
isCompatible(ltype, rtype)
> erro na linha 3, coluna 8: valor
float não pode ser convertido para
uma variável do tipo inteiro ("d")
```




```
Visitor {
   VarType = {
      a → int,
      b → float,
      d → int
}}
```

```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```




```
int a = 10;
float b = 20.5 + 20;
int d = a + b;
```

Geração de Arquivos

- Façam o download do ANTLR Versão 4.5.3 a partir do link abaixo: http://www.antlr.org/download/antlr-4.5.3-complete.jar
- Confiram as instruções do arquivo README.txt na pasta compilers-cin\aulas-praticas\ap3\ do repositório da disciplina

Exercícios Práticos

1. Modifique a gramática Cymbol para dar suporte:

- Tipos booleanos, floats e strings
- Operadores lógicos (&&, ||)
- Concatenação de strings (+)

Ex.:

```
boolean a = true;
string foo(int x, boolean b, string s) {
    if (x == 0 || b) return "miss";
    return "result: " + x + b + s;
}
```

Exercícios Práticos

Em seguida, modifique o visitor CymbolCheckerVisitor para dar suporte à linguagem definida pela nova gramática Cymbol. Caso o visitor detecte uma inconsistência de tipos, o programa deverá abortar sua execução e exibir uma mensagem de erro indicando a linha e a coluna na qual o problema foi encontrado.

Exercícios Práticos

```
int x = 10;
boolean y = (10 < x) + x;
int z = y;
```

```
erro: operador '+' na linha 2, coluna 13
indefinido para os tipos boolean e int
```

Exercícios Práticos

Utilize as mesmas regras semânticas da linguagem Java:

- int + float → float
- string + int → string
- etc.

Exercícios Práticos

- O exercício prático deve ser realizado individualmente ou em dupla e enviado por e-mail com o assunto “EXERCÍCIOS PRÁTICOS 03” para monitoria-if688-l@cin.ufpe.br até as 23:59 da quarta-feira (27.09.2017)
- A resolução do exercício prático deve estar em um arquivo comprimido com o nome “Q1.zip” e deve conter a gramática Cymbol (Cymbol.g4) alterada e os arquivos de código fonte CymbolCheckerVisitor.java e Type.java