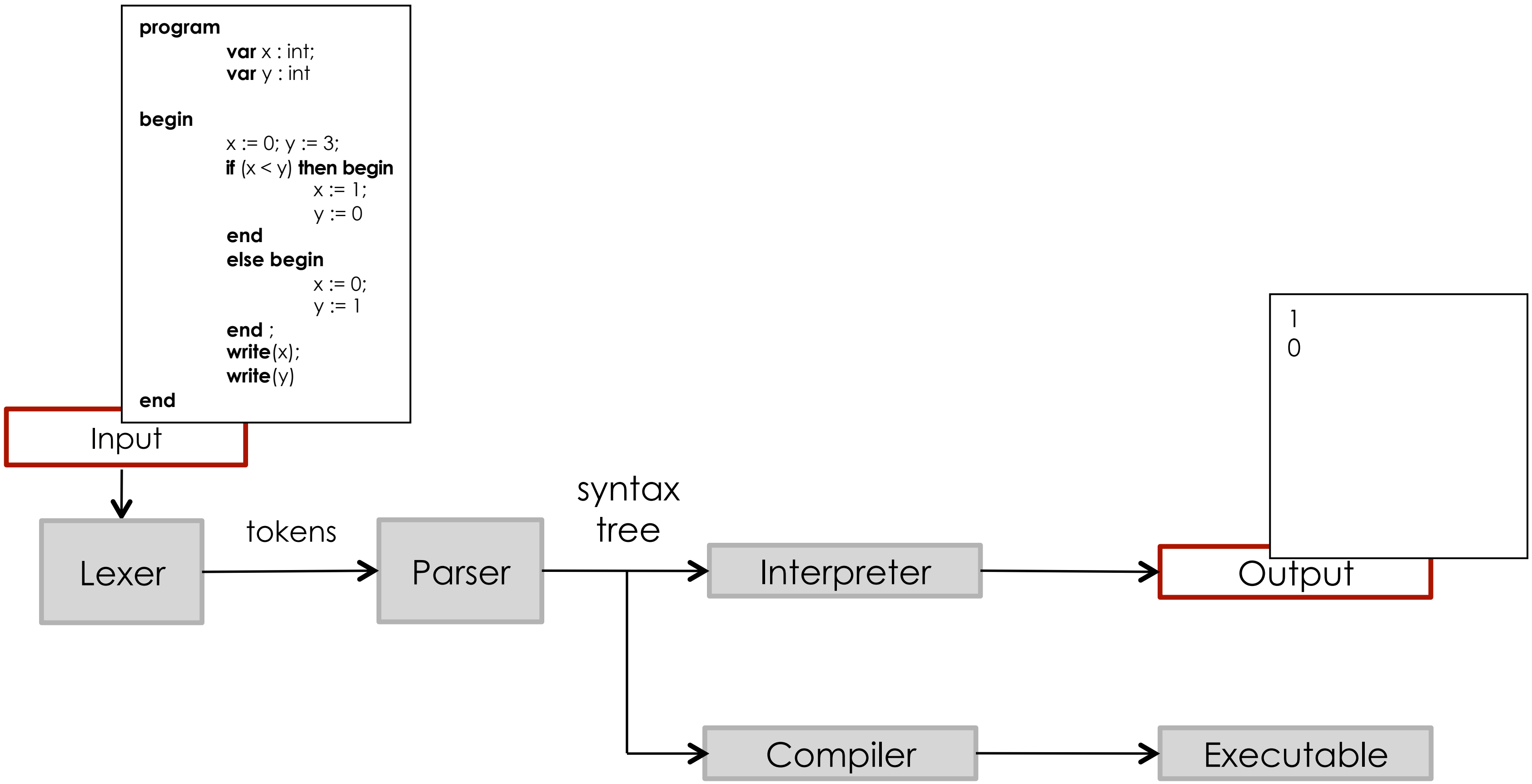
# Revisiting Interpreter

Lecture 15

Formal Languages and Compilers 2011

Nataliia Bielova

# Interpreter ≠ Compiler



# Front-end analysis (the same for interpreter and compiler)



Recognition of tokens (lexer)

↓

Checking the syntax (parser)

↓

Building the syntax tree (parser)

# 1 faze of interpreter

Syntax tree

↓

Execution of syntax tree

# How interpreter is made

- parser.mly: definition of tokens
- lexer.mll: regular expressions and creation of tokens
- syntaxtree.ml: declarations of types for the syntax tree
- parser.mly: language grammar and creation of the syntax tree
- mail.ml: starts lexer, parser, executes syntax tree
- interpreter_base.ml: functions for the execution of the syntax tree

# Definition of the memory and environment



- Formal definition:

$Store : Loc \rightarrow Val$

type store = loc -> value

$Env : Id -> (Loc \cup Val)$

type env = ide -> env_entry

- Updating the memory:

$$\mathrm{updatemem}(\mathrm{s}, \mathrm{l}, \mathrm{v})(x) = \begin{cases} \mathrm{v} & \text{if } x = \mathrm{l} \\ \mathrm{s}(x) & \text{if } x \neq \mathrm{l} \end{cases}$$

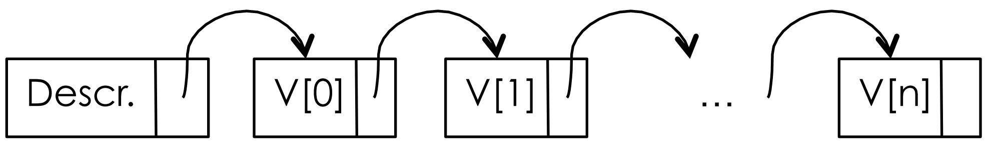
```
let updatemem((s:store), addr, (v:value)): store = function

x -> if (x = addr) then v else s(x)
```

# Implementation of vectors




☺ $\Lambda\|V[k]\| = B + O(k)$

☹ Ins. & Del.




☹ $\Lambda\|V[k]\|$ =scanning the whole list

☺ Ins. & Del.

# Vectors

- `var V:array [LB .. UB] of type`
- Address of the k-th element:

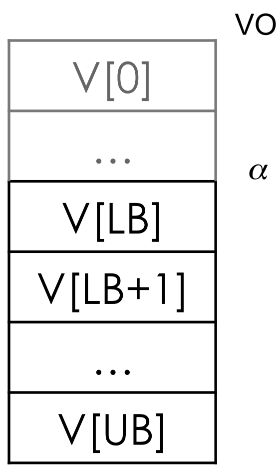
$$\Lambda\|\mathtt{V[k]}\| = \alpha + (\mathtt{k} - \mathtt{LB}) = (\alpha - \mathtt{LB}) + \mathtt{k} = \mathtt{V0} + \mathtt{k}$$

$$\mathtt{V0} = \alpha - \mathtt{LB} = \Lambda\|\mathtt{V[0]}\|$$

Descriptor:

| |
|---|
| VO |
| LB |
| UB |

Representation in the memory:




`var V : array[3..5] of int;`

$$\Lambda\|\mathtt{V[4]}\| = \alpha + (4 - 3) = \mathtt{V0} + 4$$

$$\mathtt{V0} = \alpha - 3 = \Lambda\|\mathtt{V[0]}\|$$

# Bidimensional matrices

$\texttt{var V : array[LB}_1\texttt{..UB}_1\texttt{, LB}_2\texttt{..UB}_2\texttt{] of } \textit{type}$

- Dimension of an element: $M_2$
- Dimension of a row: $M_1 = (UB_2 - LB_2 + 1) \times M_2$
- Virtual Origin: $VO = \alpha - LB_1 \times M_1 - LB_2 \times M_2$

$$\Lambda \| V[i, j] \| = VO + i \times M_1 + j \times M_2$$

# Virtual Origin (V.O.)

```
var v : array[5..8][2..6] of int;
```

$M_2 = \mathtt{sizeof(int)} = 1$

$M_1 = (UB_2 - LB_2 + 1) \times M_2 =$

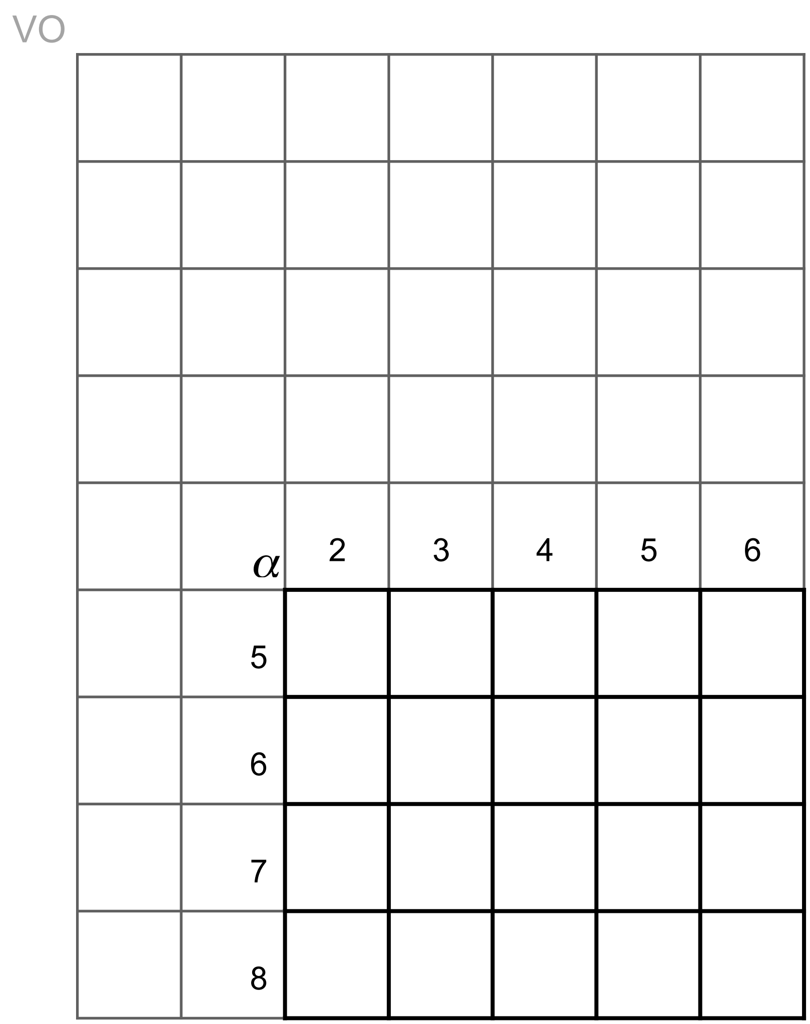
$= (6 - 2 + 1) \times M_2 = 5$

$VO = \alpha - LB_1 \times M_1 - LB_2 \times M_2 =$

$\alpha - 5 \times M_1 - 2 \times M_2 = \alpha - 27$

$\Lambda \| v[7,4] \| = VO + 7 \times M_1 + 4 \times M_2 =$

$= VO + 35 + 4$



# Virtual Origin (V.O.)

```
var v : array[5..8][2..6] of int;
```

$M_2 = \text{sizeof(int)} = 1$

$M_1 = (UB_2 - LB_2 + 1) \times M_2 =$

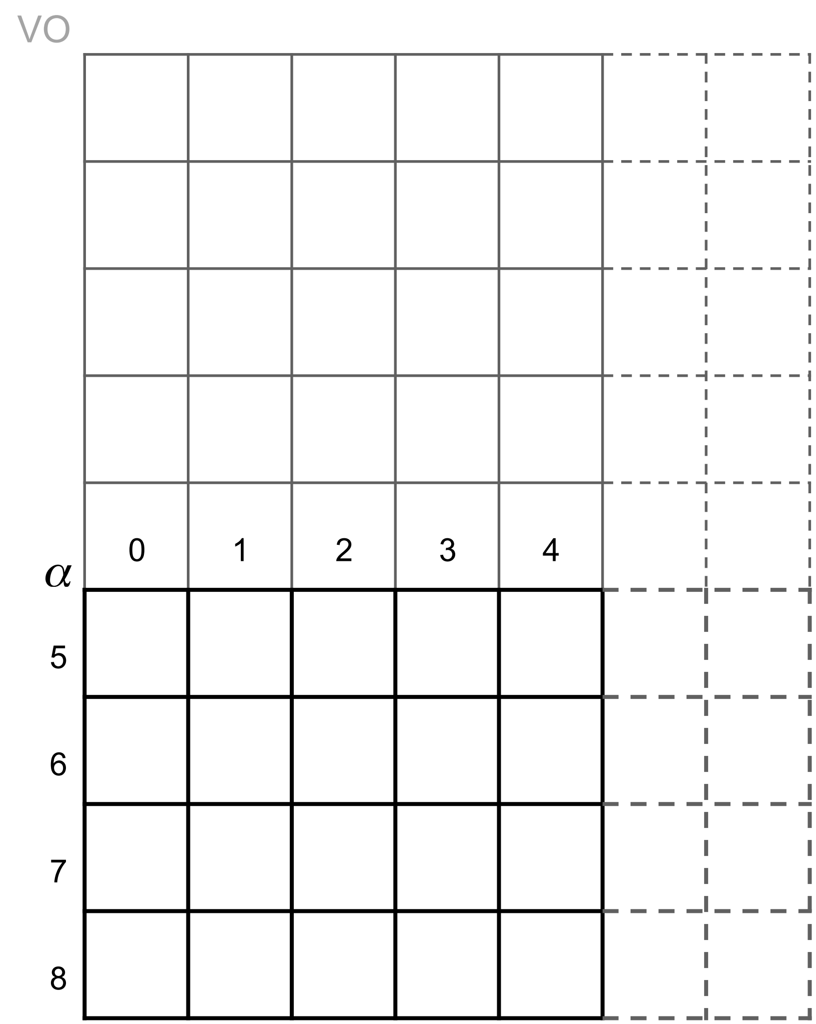
$= (6 - 2 + 1) \times M_2 = 5$

$VO = \alpha - LB_1 \times M_1 - LB_2 \times M_2 =$

$\alpha - 5 \times M_1 - 2 \times M_2 = \alpha - 27$

$\Lambda\|v[7,4]\| = VO + 7 \times M_1 + 4 \times M_2 =$

$= VO + 35 + 4$



# Virtual Origin (V.O.)

```
var v : array[5..8][2..6] of int;
```

$M_2 = \text{sizeof(int)} = 1$

$M_1 = (UB_2 - LB_2 + 1) \times M_2 =$

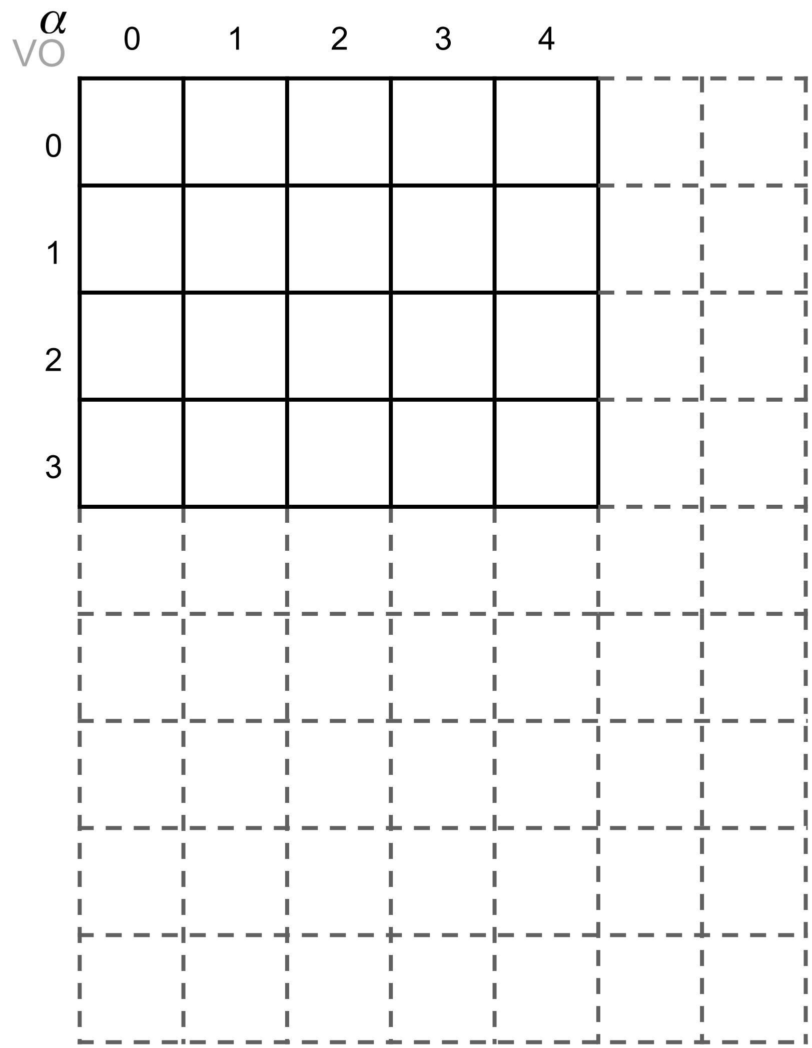
$= (6 - 2 + 1) \times M_2 = 5$

$VO = \alpha - LB_1 \times M_1 - LB_2 \times M_2 =$

$\alpha - 5 \times M_1 - 2 \times M_2 = \alpha - 27$

$\Lambda\|v[7,4]\| = VO + 7 \times M_1 + 4 \times M_2 =$

$= VO + 35 + 4$



# Multidimensional matrices

$$\texttt{var V : array[LB}_1\texttt{..UB}_1\texttt{, ..., LB}_n\texttt{..UB}_n\texttt{] of } \textit{type}$$

- Multipliers:

$$M_n = M$$

$$M_i = (UB_{i+1} - LB_{i+1} + 1) \times M_{i+1} \quad i \in [1, n-1]$$

$$VO = \alpha - \sum_{i=1}^{n} LB_i \times M_i$$

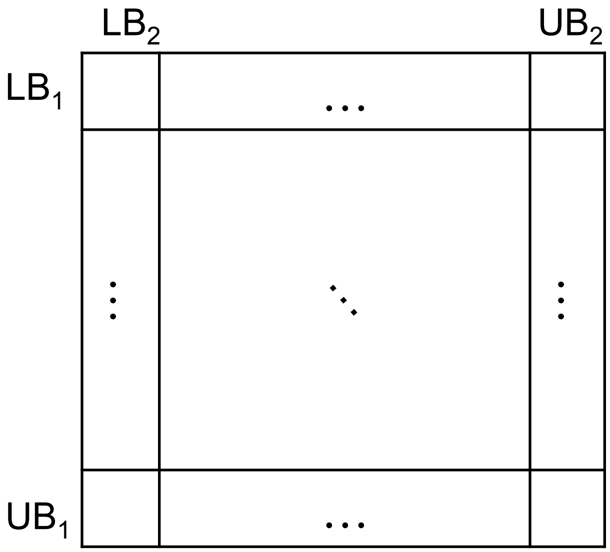
$$\Lambda \| V[k_1, \ldots, k_n] \| = VO + \sum_{i=1}^{n} k_i \times M_i$$

$$\texttt{array[LB}_1\texttt{..UB}_1\texttt{, LB}_n\texttt{..UB}_n\texttt{] of } \textit{type}$$

$$\approx$$

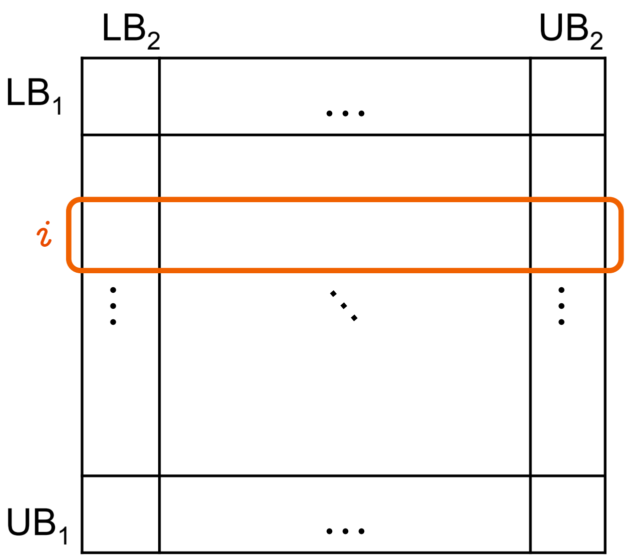
$$\texttt{array[LB}_1\texttt{..UB}_1\texttt{] of (array[LB}_2\texttt{..UB}_2\texttt{, LB}_n\texttt{..UB}_n\texttt{] of } \textit{type}\texttt{)}$$

# Slices of array




```
var v:array[LB1 ..UB1 ,LB2 ..UB2] of type;
var s: slice[i,*] of v;
```

# Slices of array




$$M = M_2$$

$$VO_I = VO_V + i \times (UB_2 - LB_2 + 1) \times M_2 =$$

$$VO_V + i \times M_1$$

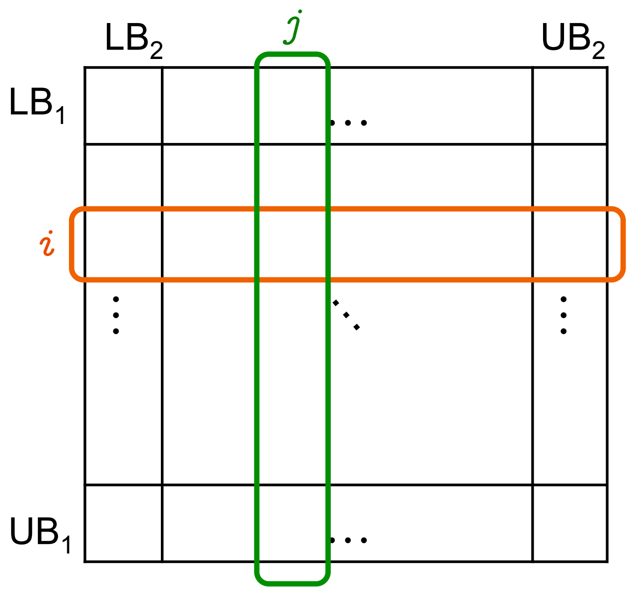
$$LB = LB_2$$

$$UB = UB_2$$

$$\Lambda \| I[k] \| = VO_I + k \times M$$

$$I = V[i][*] = \{V[i][LB_2], V[i][LB_2+1], \ldots, V[i][UB_2]\}$$

# Slices of array




$$M = (UB_2 - LB_2 + 1) \times M_2 = M_1$$

$$VO_J = VO_V + j \times M_2$$

$$LB = LB_1$$

$$UB = UB_1$$

$$\Lambda\|J[k]\| = VO_J + k \times M$$

$$I = V[i][*] = \{V[i][LB_2], V[i][LB_2 + 1], \ldots, V[i][UB_2]\}$$

$$J = V[*][j] = \{V[LB_1][j], V[LB_1 + 1][j], \ldots, V[UB_1][j]\}$$

# Types of passing the parameters



Notation `call P(x` $\Leftarrow_\alpha$ `e)` means that

- P is declared as **`proc P(x) …`**
- P is invoked as **`call P(e)`**
- $\alpha$ is type of passing the parameters

| | |
|---|---|
| Value | `call P(x` $\Leftarrow_{\mathtt{Val}}$ `e)` |
| Value-result | `call P(x` $\Leftarrow_{\mathtt{Val\text{-}res}}$ `y)` |
| Result | `call P(x` $\Leftarrow_{\mathtt{Res}}$ `y)` |
| Reference | `call P(x` $\Leftarrow_{\mathtt{Ref}}$ `y)` |
| Constant | `call P(x` $\Leftarrow_{\mathtt{Const}}$ `e)` |
| Name | `call P(x` $\Leftarrow_{\mathtt{Name}}$ `e)` |

Note: x, y are variables, e is an arithmetical expression

# Passing by name

$$\texttt{call P(x} \Leftarrow_{\texttt{Name}} \texttt{e)}$$

- create a new couple `<e, r>`, where `r` is an environment of the caller
- every time when `x` should be evaluated, `e` is getting evaluated instead in the environment `r` and put instead of `x`.
- `x` cannot be assigned values in `P`