Compiling Techniques
ECOTE – Macrogeneration

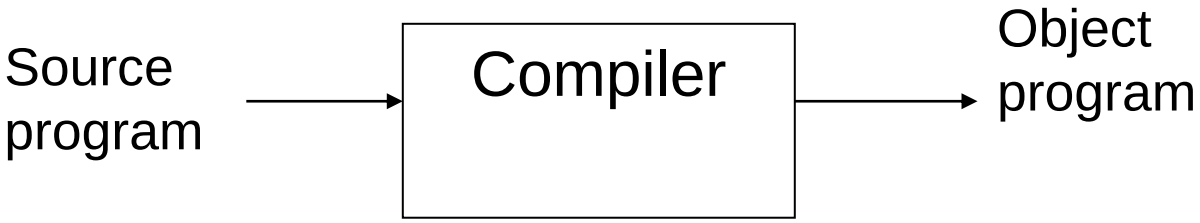
DSc Dr Ilona Bluemke

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!

EUROPEAN UNION
EUROPEAN
SOCIAL FUND

Project is co-financed by European Union within European Social Fund

Introduction to compilers



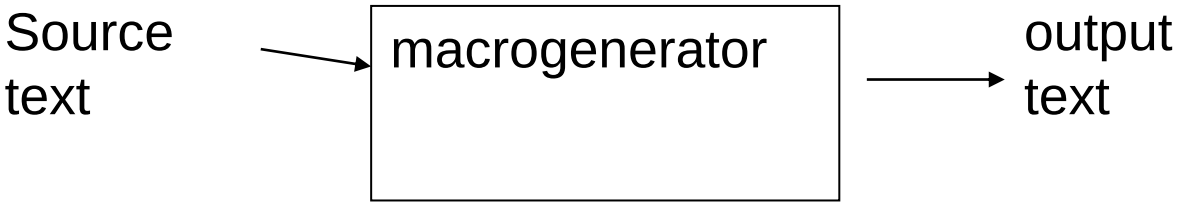
Objectives

- Organisation and modularization the process of compilation
- Systematic techniques for handling tasks that occur during compilation
- Software tools facilitating the implementation of compilers

Other translators

- ***Interpreter –*** transforms program into ***intermediate code,*** which can be directly executed (eg. command languages, Basic), smaller than compiler, longer execution time
- ***Assembler*** – source program is assembly language
- ***Preprocessor*** – takes program in one language and transforms into another (PL/1, C, ...)

macrogenerator

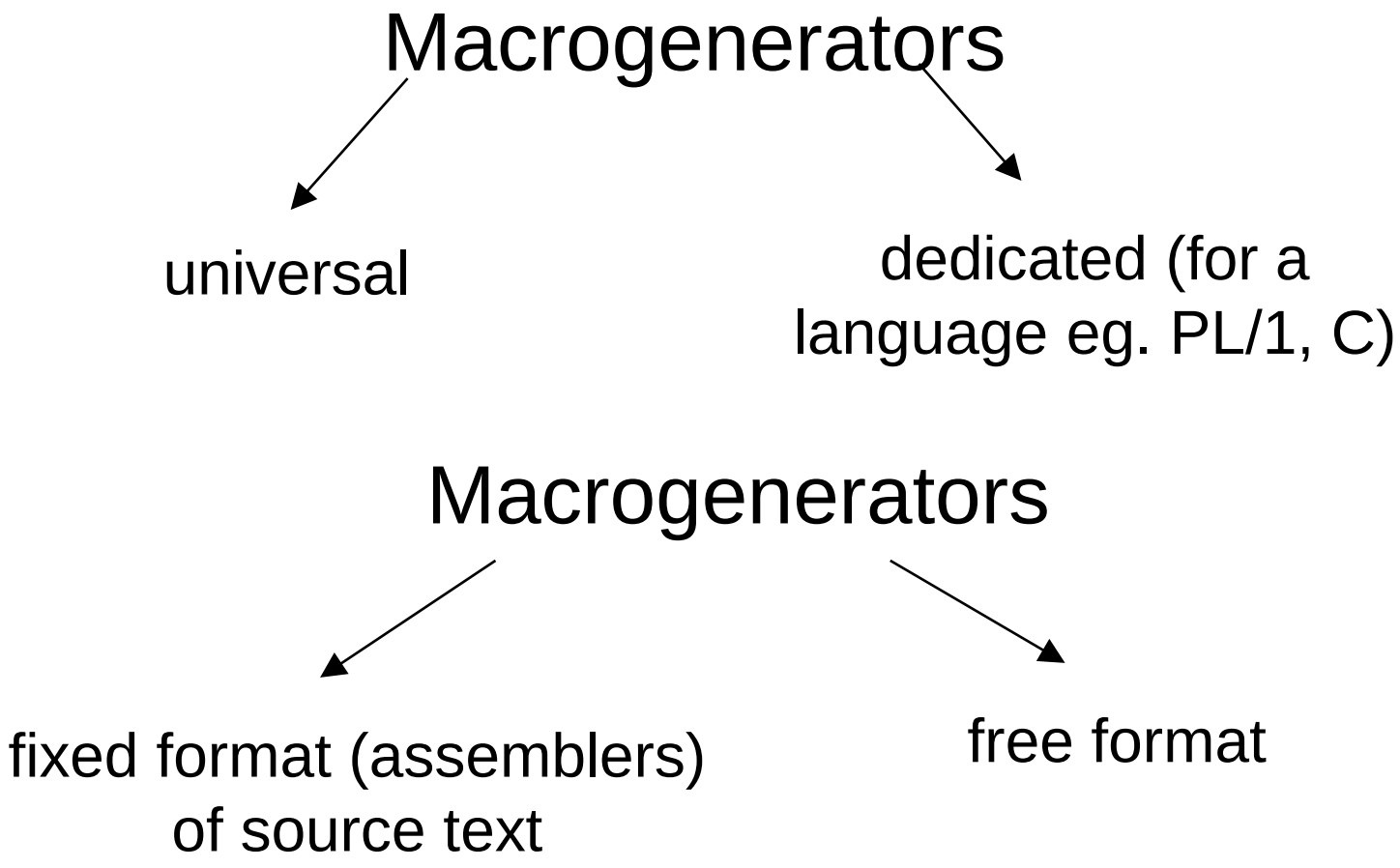


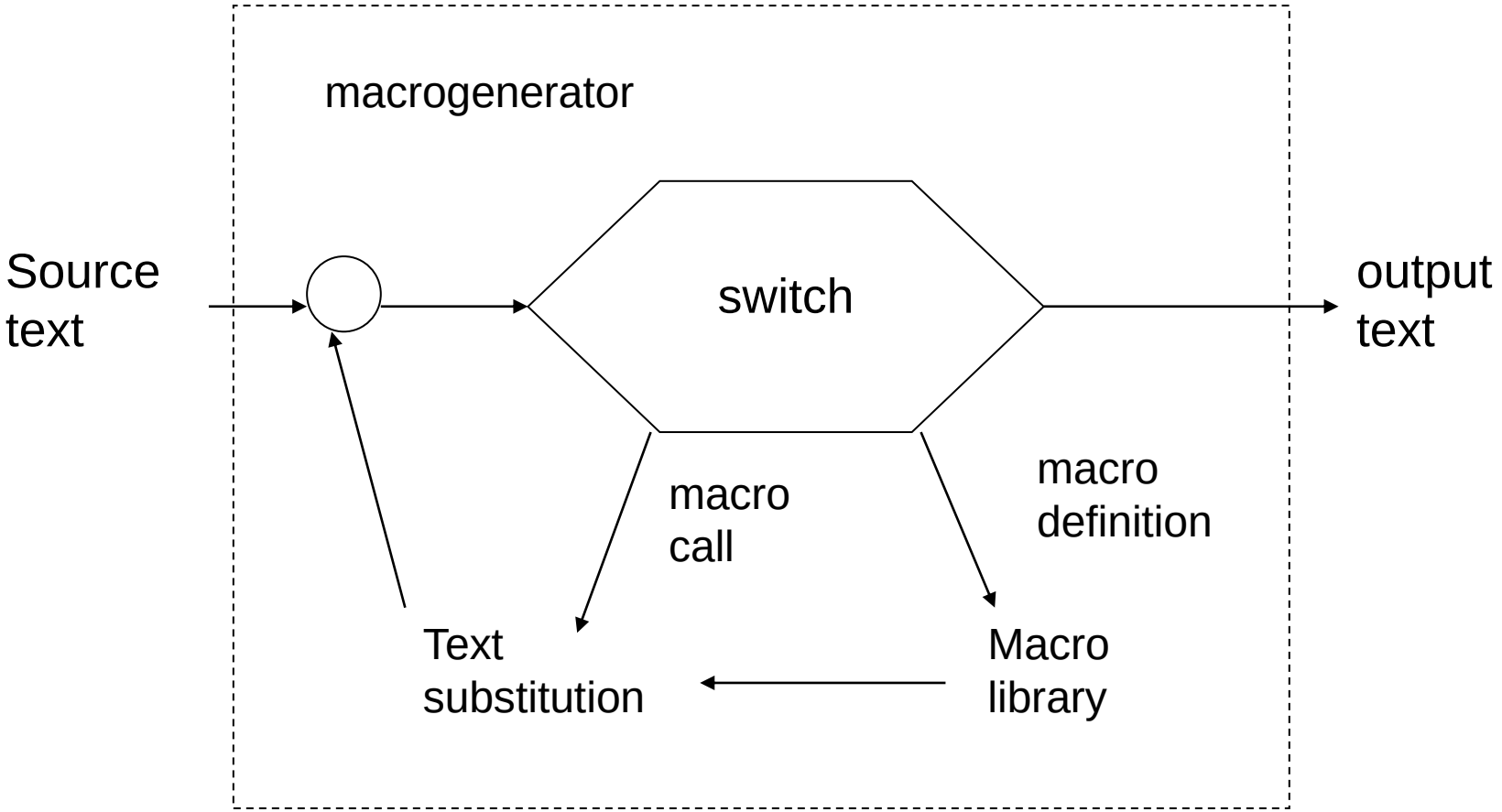
- Text replacement, text transformation
- Macro definition – transformation rules
- Macro call (use)

Macro definition:

- in the source text (**dynamic** transformation)
- inside the macrogenerator (**static** transformation)

Types of macrogenerators




macrogenerator
Source text
switch
output text
macro call
macro definition
Text substitution
Macro library

Switch – takes decisions

“**free**” text

- **macro definition (MD)** inserted into library
- **macro call (MC)** substituted text taken from library becomes input for the switch, source text is blocked during macro substitution

macrogenerators with:

- **discriminant** (special character distinguishing definition and call)
- **template** (call distinguished after comparing with templates defined in library)

macro library scope:

- **global** (all MD visible)
- **local** (MD at current level)

Text substitution

direct

Identifier given to a sequence of characters, MD – free text, not substituted

parameters (formal parameters substituted with actual ones)

- number of actual parameters fixed or variable
- connections between formal and actual parameters
- if the parameter can be substituted when the parameter substitution is performed:
  - when parameter recognised in MD
  - before switching to MD

Example

@M [(A&1B)]

$ - MC discriminant

@ - MD discriminant

& - formal parameter discriminant

$M[x] $M[y] $M[z]

substituted as: (AxB) (AyB) (AzB)

conditional substitution

if <condition> **then** <seq1> **else** <seq2> **fi**

if, then, else, fi – delimiters (not appearing in the output text)

<condition>::= <s1><op><s2>

<s1>, <s2> sequence of characters

<op> operator eg. =, <, >

hierarchical (multilevel) substitution

Example

(A(BxC)*(ByC)*D)
(A(BvC)+(BwC)+D)(BzC)

@Q [(B&1C)]

@P [(A$Q[&1]&2$Q[&3]&2D)]

$P[x,*,y] $P[v,+,w] $Q[z]

recursive substitution

inside MD are MC of the same macro

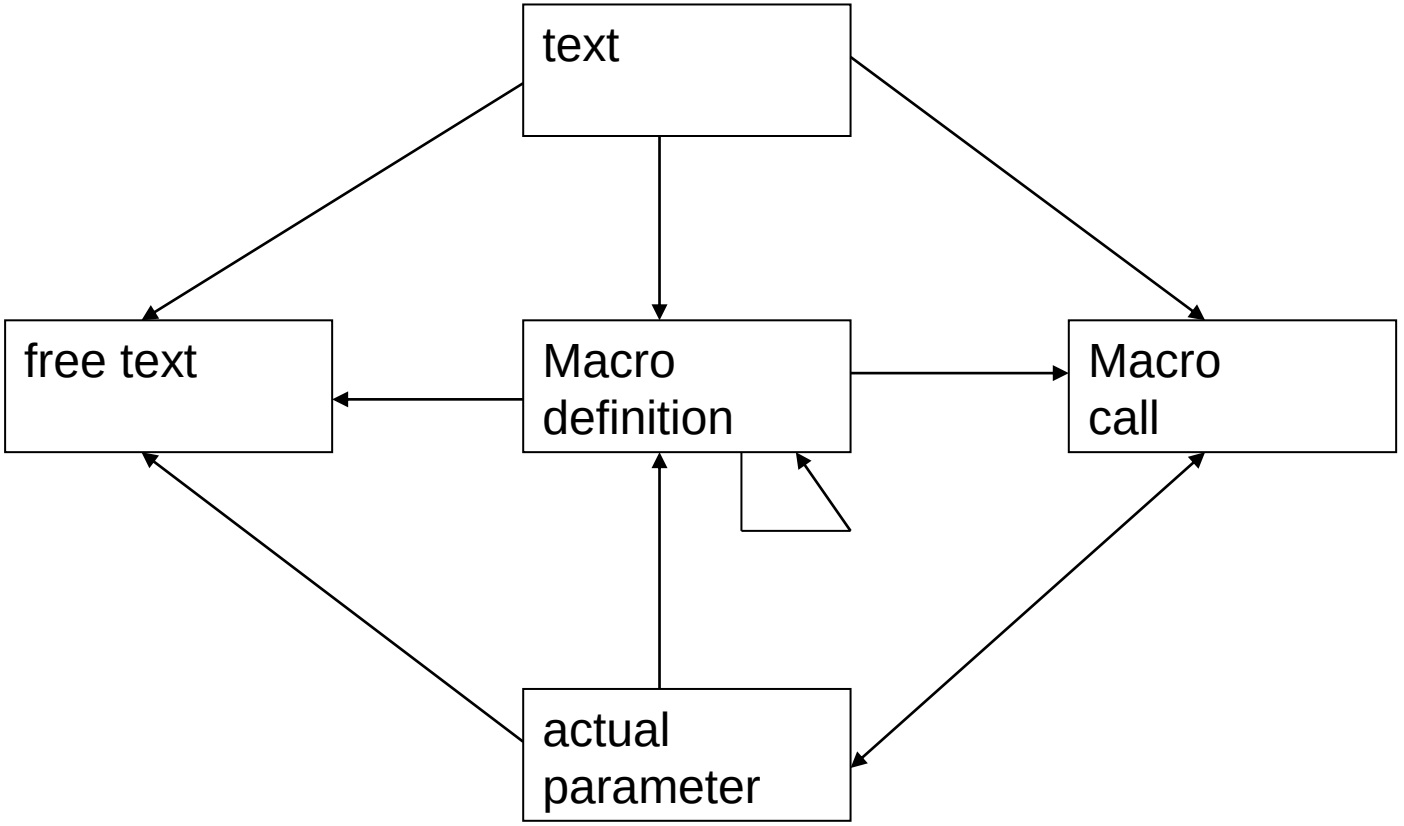
Example

@V [&1$V[&1]]

$V[x]

to finish the recursion some conditional substitution must be provided

Static text composition rules



Text level

0 for source text

+1 MC starts

parameter substitution starts

-1 MC finished

parameter substitution finished

0 -> 1 only MC

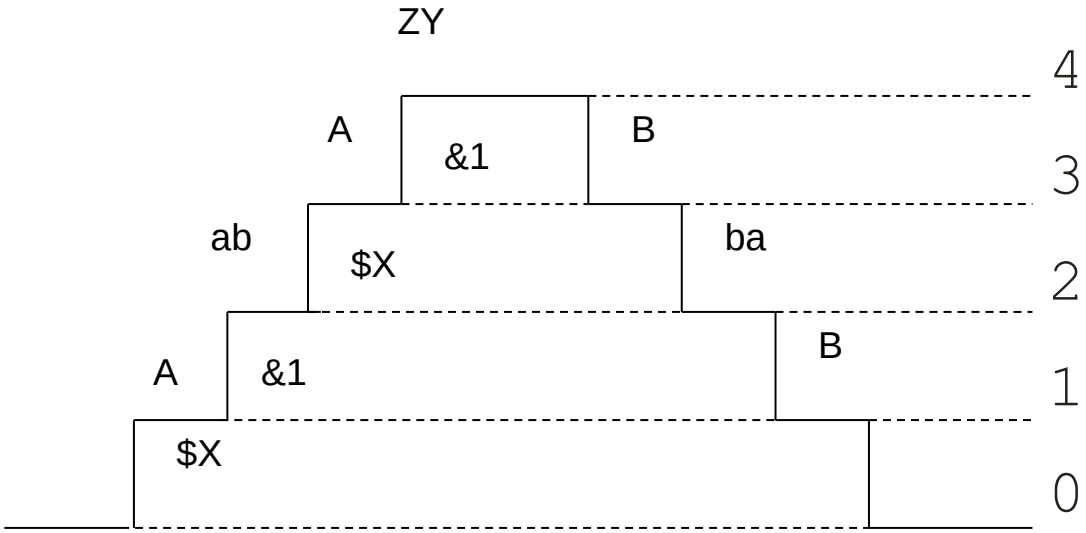
1 -> 0 only end of MC

Text level diagram

Example

@X[A&1B]

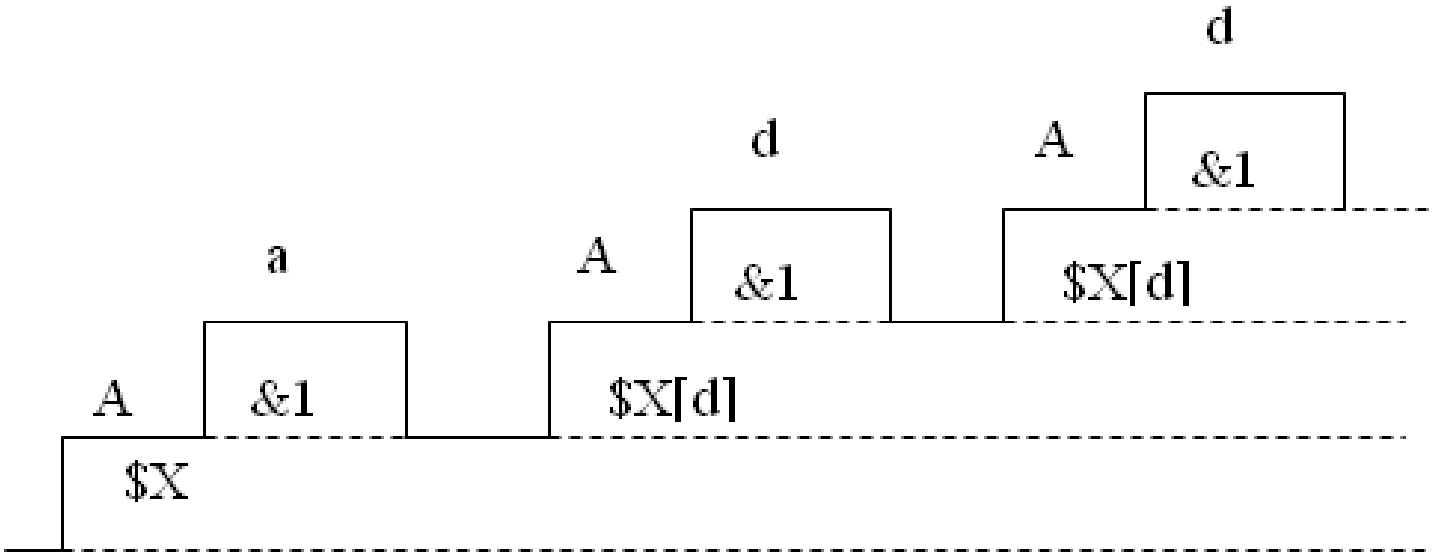
$X [ab$X[ZY]ba]



Output: AabAZYBbaB

@X[A&1$X[d]]

$X[a]



Organisation of macro library

- **global** (all MD are visible, ”new” MD covers “old”)
- **hierarchical** (MD defined at level *l*, *l>0*, are available at level *m*, *m>=l,* could be “covered” by definition of a homonym, the destruction of level *m,* destroys all the MD introduced at this level)

  Example

  @SUB[procedure @SUB[link_to_procedure]]

  $SUB

  compare different library organisations

Implementation of a macrogenerator MG

- hierarchical organisation of macro library
- 0-9 formal parameters (called by “name”)
- text composition rules as shown on a graph
- delimiters:
  - [] cover macro “body”, actual parameters
  - , actual parameters
  - # end of text (parameter, macro body)
- discriminants:
  - **$** - MC
  - @ - MD
  - **&** - formal parameter

Macrogeneration

- on each text level macro status descriptor
  - source of input text
  - library filled up
  - receiver of the output text
  - links to descriptors of lower levels
  - place of actual parameters
- on the same text level
  - copy “free” text from input to output
  - recognise MD, put MD in library, change library filled up

Increased text level

- finding new status descriptor

$ - MC

  - complete macro identifier,
  - search macro library
  - new input text = macro “body”

- MC has actual parameters. Parameters unchanged are stored, the position of parameters is put in the status descriptor
- status descriptor is stored, new status descriptor becomes current (levels are switched)

&- parameter

Current status descriptor gives information about the position of actual parameters

- current text level **MC**

  text level defines parameters

  &i found, **i**th parameter among parameters defined on this level

- current text level – **parameters substitution**

  no parameters at this level

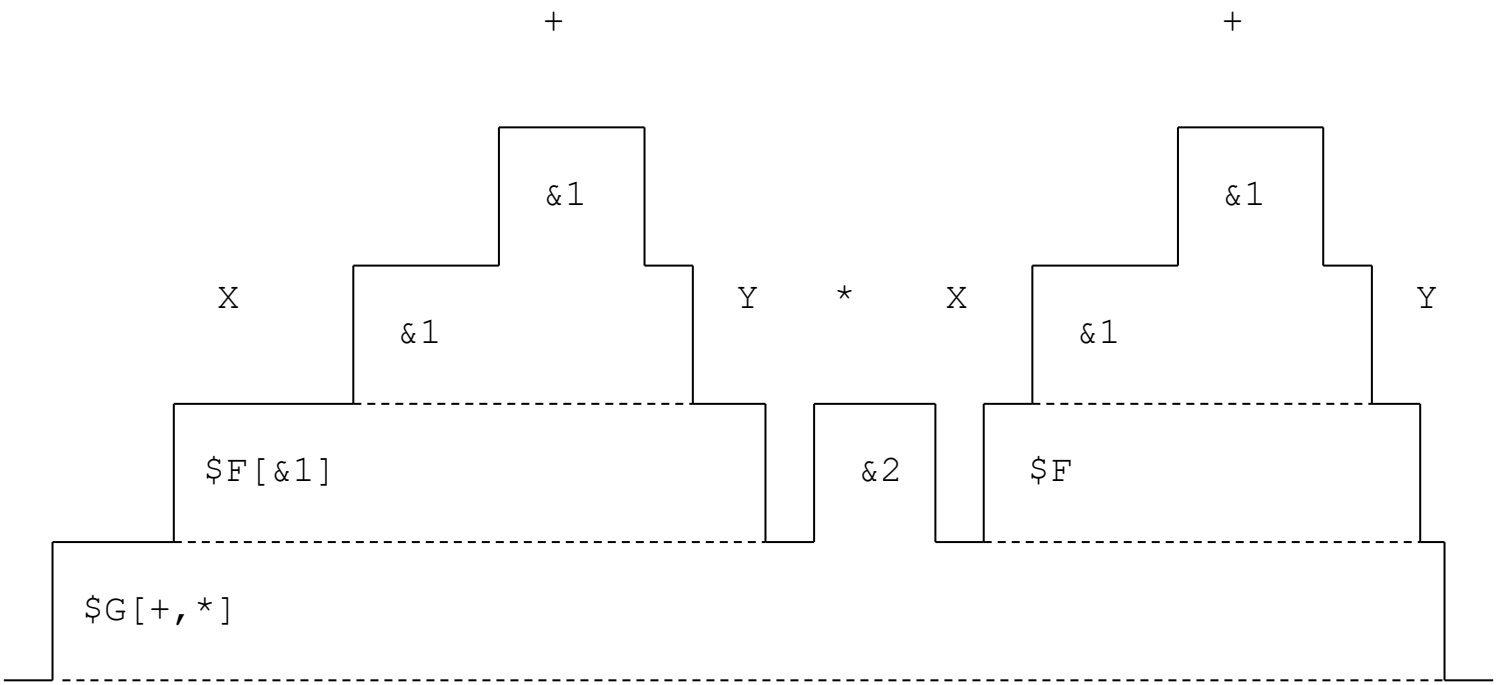
  &i found, looking for **i**th parameter among parameters defined on previous level – “**dragging**” parameter

Decreased text level

recognised

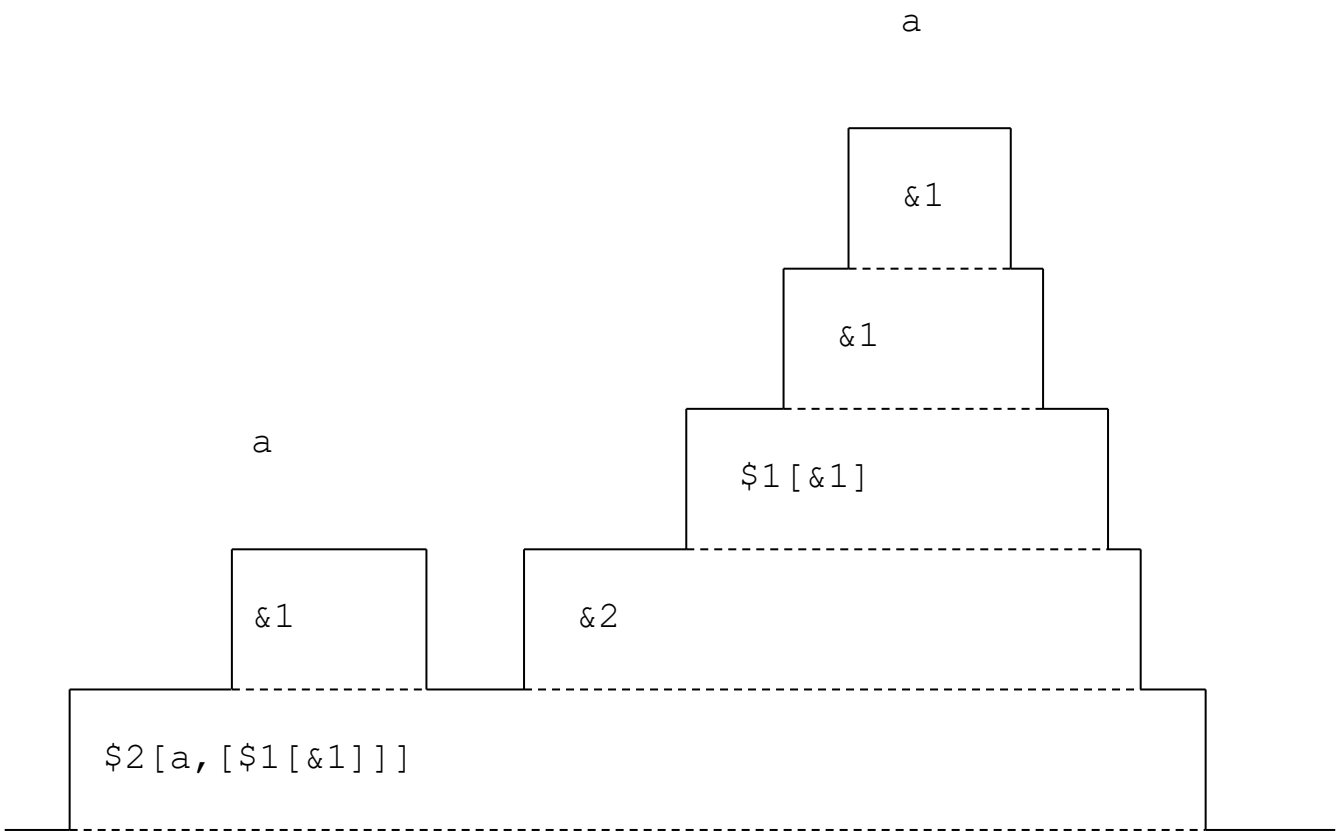
Current status descriptor is changed- taken is the descriptor from the lower level.

Example : @F[X&1Y] @G[$F[&1] &2 $F[&1]] $G[+,*]



Output: X+Y * X+Y

@1[&1] @2[&1&2] $2[a,[$1[&1]]]



Data structures

Tables:

- TEXT text information (identifiers, macro body, parameters)
- MNAME macro names locations
- MBODY macro body locations
- ARGS parameters locations

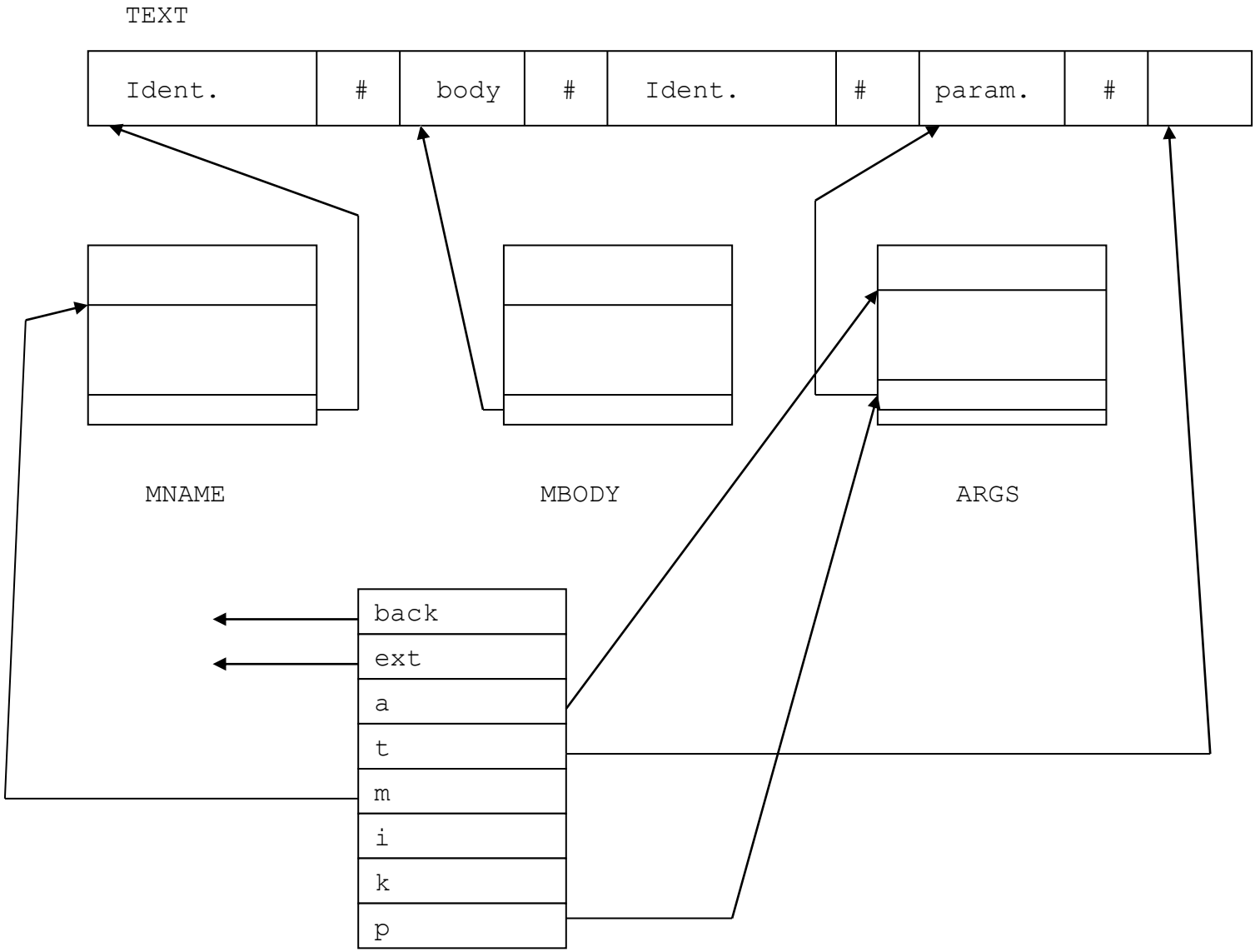
Status descriptor (class MGStatus)

Stack of status descriptors:

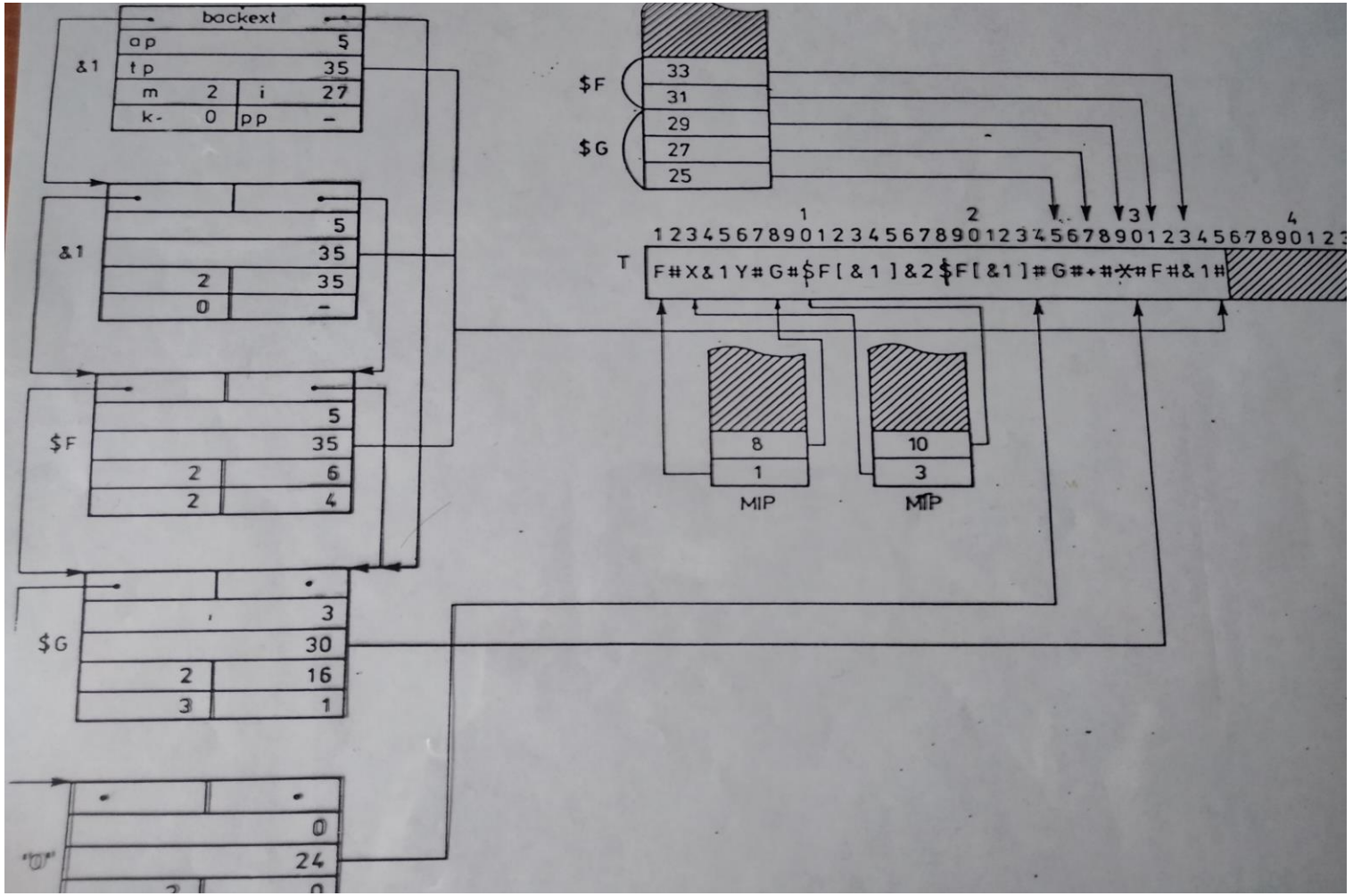
- back, ext - pointers to previous, external (defining parameters) descriptors
- a - number of used positions in ARGS
- t - number of used positions in TEXT
- m- number of available macros
- i – source of input character

  i=0 source text

  i>0 position in TEXT
- k – number of actual parameters
- p – position of the first parameter in ARGS


TEXT
Ident.
#
body
#
Ident.
#
param.
#
MNAME
MBODY
ARGS
back
ext
a
t
m
i
k
p

@F[X&1Y] @G[$F[&1] &2 $F[&1]] $G[+,*]



Differences between MG and GPM (GENERAL PURPOSE MACROGENERATOR)

	MG	GPM
Parameters	literally (by name)	substituted (by value)
MC	not in macro id	macro id generated
MD	fixed	generated

GPM macro definition:

internal macro **DEF**

first argument = **name**

second arg. = “**body**”

GPM syntax

discriminants:

- **$** - MC
- **&** - formal parameter
- **<**quotation**>** not interpreted, external **< >** taken away

delimiters:

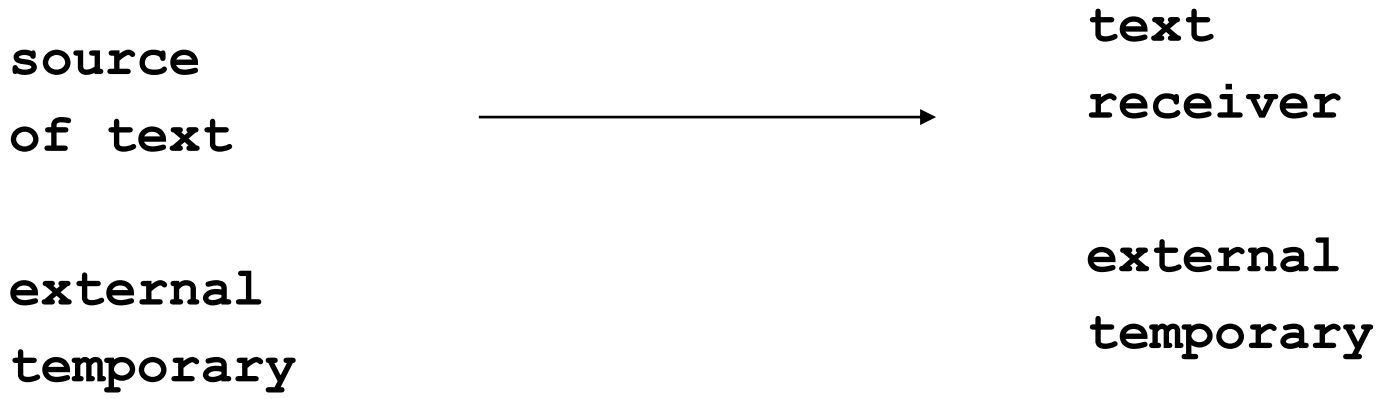
- **;** MC
- **,** parameters

“paired”:

- $;
- < . >

GPM interpretation

- through input text from left to right once
- immediate substitution



- **“free”** text copied from source to receiver,
  source and receiver unchanged
  text **without $, <>**

“simple” text text **without , ;**

- <quotation> external **< >** taken away, copied,checked number of < >

GPM Macro Call

1. Generation of all parameters (0,1,..to ;)

   For each parameter new, internal text receiver is created

   New receiver - , detected

   MC inside parameter generated at once

   Simple text copied to current receiver

   Quotation - external **<** **>** taken away, copied,

2. Parameter **zero – macro identifier**

- Table with macro definitions is searched, from top to bottom.
- At the bottom internal – system macro **DEF**
- Macro found – MD becomes **new source** of text
- receiver – the same as at **$** recognition

3. Text found in **step 2** is interpreted from left to right, interpretation rules as above

- ***&i*** detected , ***i***th parameter (**step 1**) is substituted, substitution is literal
- error if improper parameter number

For system macro

$DEF p1, p2;

step 3 contains different activities:

new entry in the **table with macro definitions** is added :

macro **identifier** **p1**

macro **body** **p2**

4. End of MC

- Parameters generated in step 1 and MD added in step 1 and 3 are removed from internal GPM structures
- Switch to source of text obligatory at $ detection

During GPM generation a **stack of text receivers** is maintained.

Some of them are “closed” (with completed text), some are “open”- awaiting some text.

START

1 *2*

$**X, **$DEF,X,<&2&1&2>**;**KAD,$X, BR, A **;** **;**

END

Text receivers

1 2

A
BR
X
KAD
X

ABRA
KAD
X

MD table

X	&2&1&2
DEF	

OUTPUT:

START ABRAKADABRA END

GPM examples

1.

$DEF,SUC,<$1,2,3,4,5,6,7,8,9, $DEF, 1,<&>&1; ;>;

$SUC, 4;

Substitution 10th parameter defines macro name 1 and body &4, the 4th parameter is 5 (the result)

2.

internal macros ***, **+**, **-**, ***/***

$+, 7, -3;

$*, $+, 8, 1; , $+, 8, -1; ;

GPM - factorial

```
$DEF, FACTORIAL, <$&1, $DEF, &1, <$*, &0,
$FACTORIAL, $-, &0, 1; ; ; > ; $DEF, 0,<1>; ;>;

$FACTORIAL, 3;
calls $3,;
which results in: $*, 3, $FACTORIAL, 2;;
.....
$FACTORIAL,0;  two MD of macro 1
the second with body <1> is taken (last)
```

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!

EUROPEAN UNION
EUROPEAN
SOCIAL FUND

Project is co-financed by European Union within European Social Fund