Recursion

Semantic

D.E ZEGOUR

École Supérieure d'Informatique

ESI

Recursion / Semantic

Introduction

Recursion is a powerful tool.

What exactly happens in RAM when a recursive call is made?

Anticipation of compilation techniques

Transformation of recursive algorithms into iterative algorithms

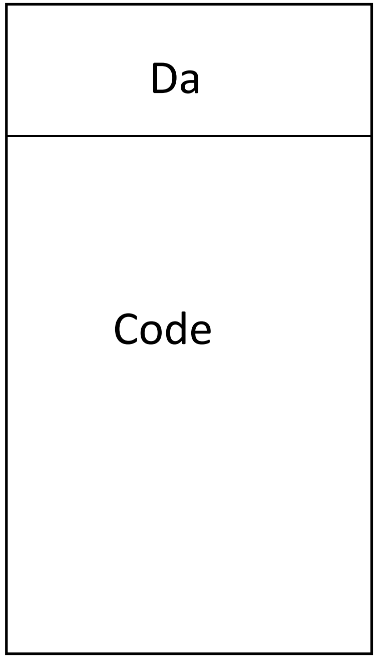
Recursion / Semantic

Compilation concepts

Each procedure is associated with:

- A data area (Da) (local variables, parameters, etc.)

- Code

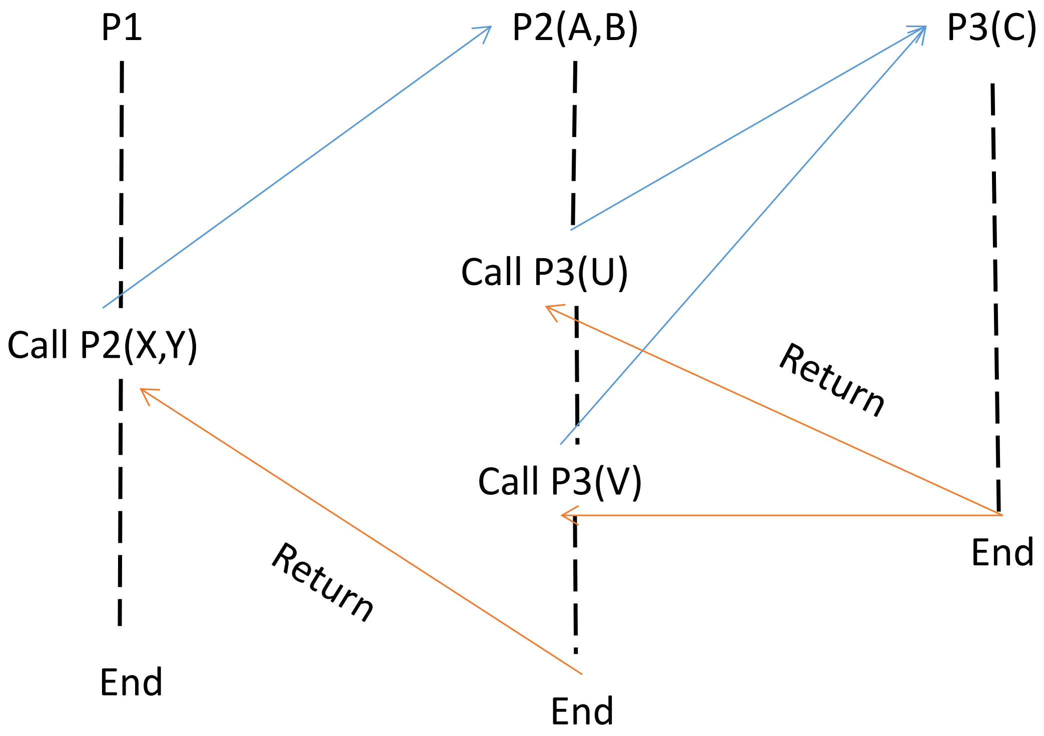


Da

Code

Recursion / Semantic

Compilation concepts



Scenario: P1 calls P2

P2 calls P3 twice.

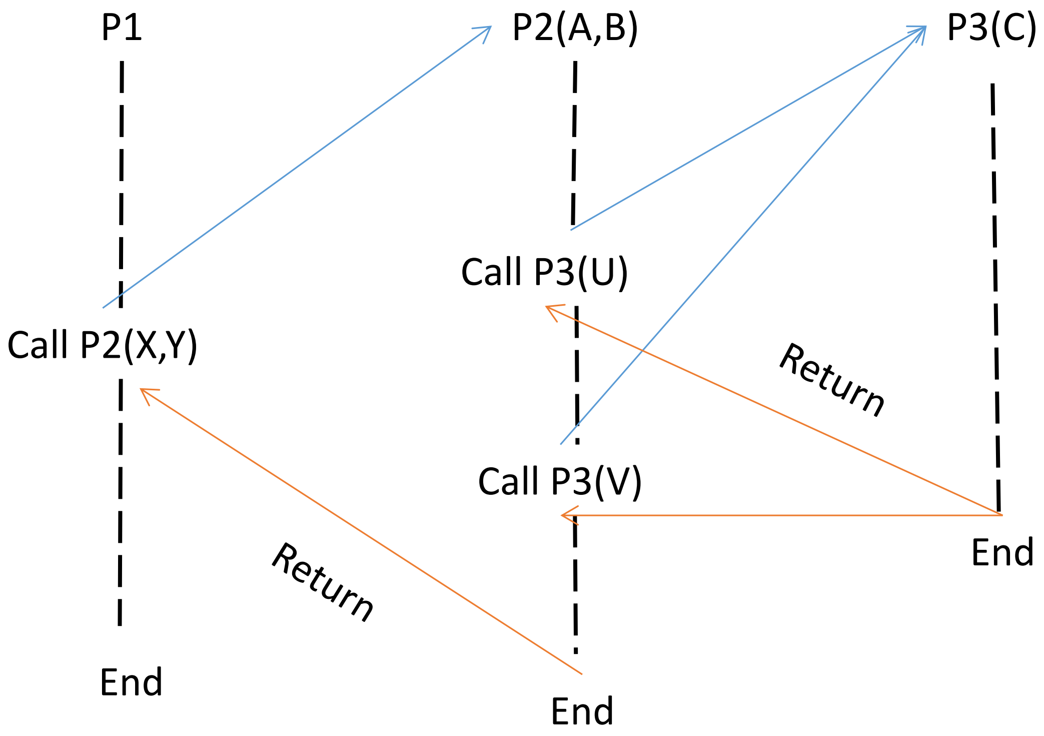
Caller: the procedure making the call.

Callee: the called procedure.

X, Y, U, and V: actual parameters
A, B, C: formal parameters

Recursion / Semantic

Compilation concepts



The data areas (Da) are managed as a stack.

At each call: save the Data Area (Da) of the caller (Push onto the stack).

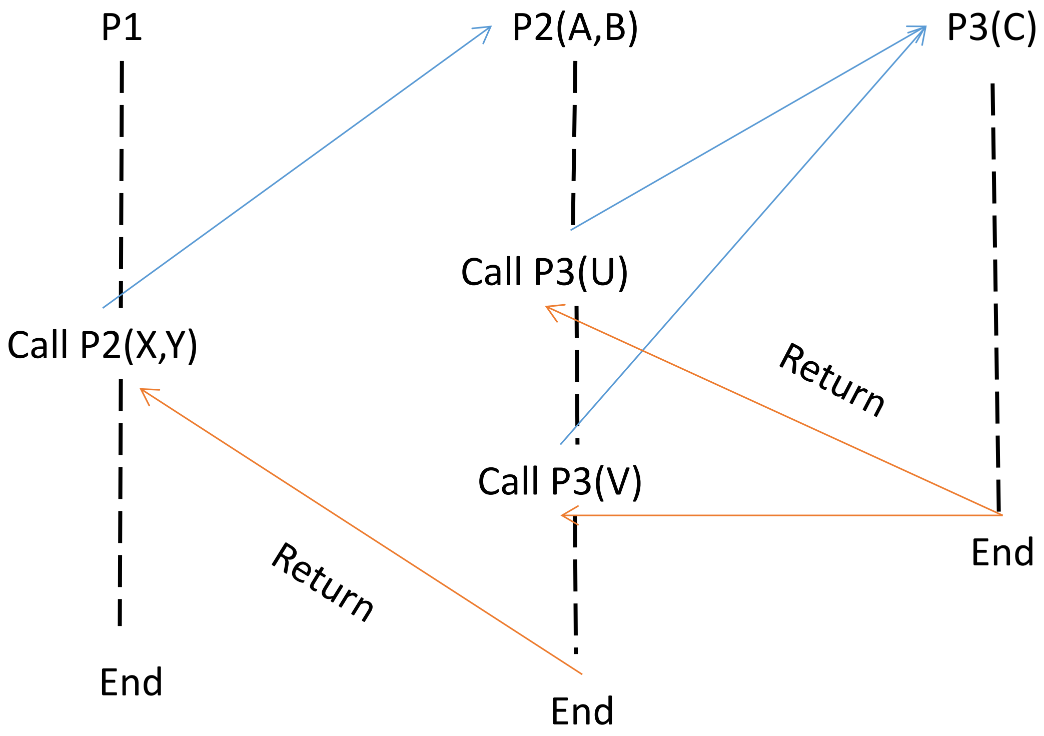
At each return: restore the data area of the caller.

The callee always returns to the last caller.

The stack:
Before the call to P2: empty stack
Call P2(X, Y): Da(P1)
Call P3(U): Da(P2), Da(P1)
Return from P3: Da(P1)
Call P3(V): Da(P2), Da(P1)
Return from P3: Da(P1)
Return from P2: empty stack

Recursion / Semantic

Compilation concepts



At the time of a call:

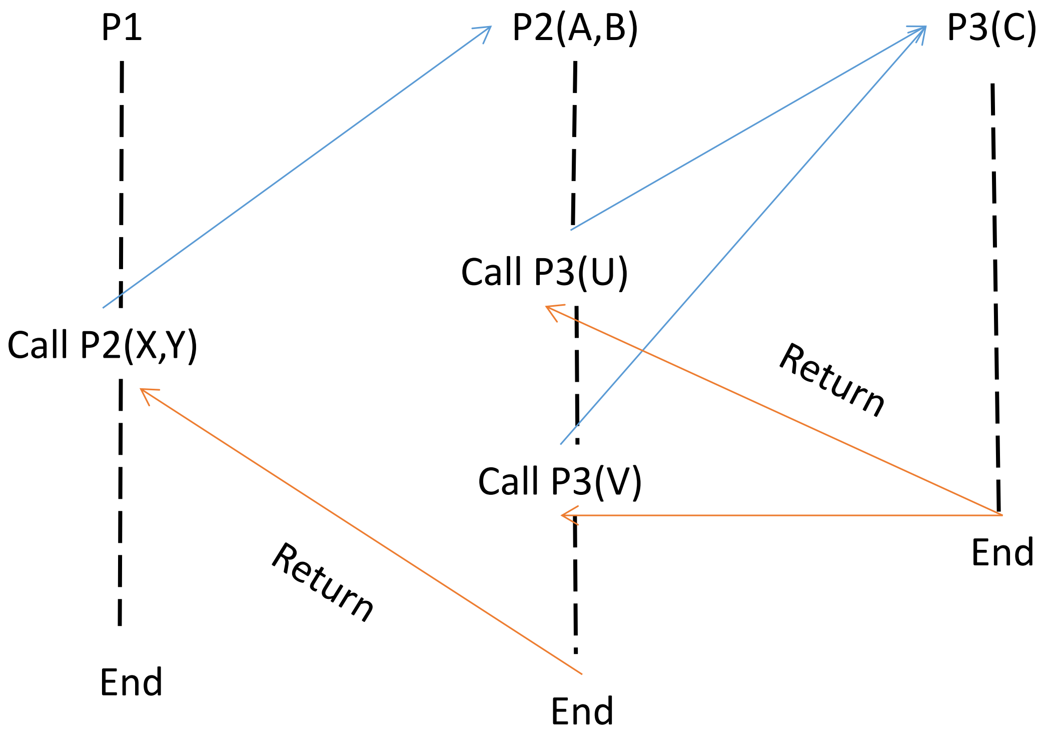
- Save the data area of the caller
- Transmit parameters in the data area of the callee
- Save the return address in the data area of the callee
- Branch to the beginning of the called procedure.

The return address at the caller's level must also be saved.

Place it in the data area of the called procedure.

Recursion / Semantic

Compilation concepts

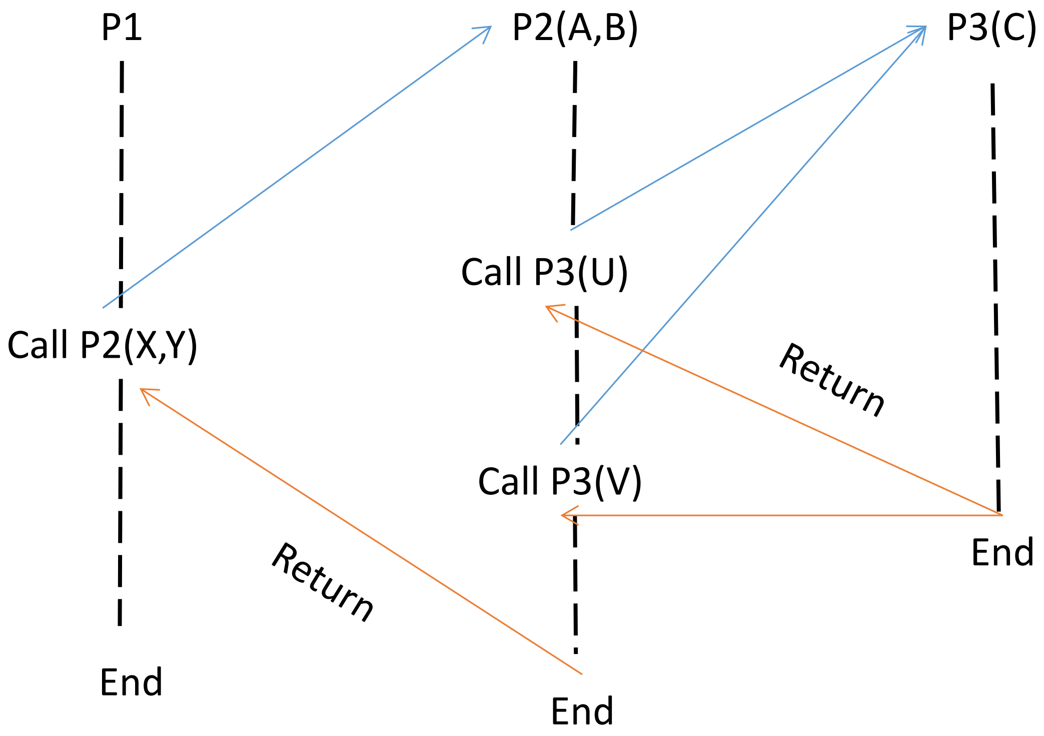


At the time of a return:

- Retrieve the return address (Ret) from the data area of the callee
- Restore the data area of the caller
- Branch to Ret in the caller

Recursion / Semantic

Compilation concepts



Parameter passing:

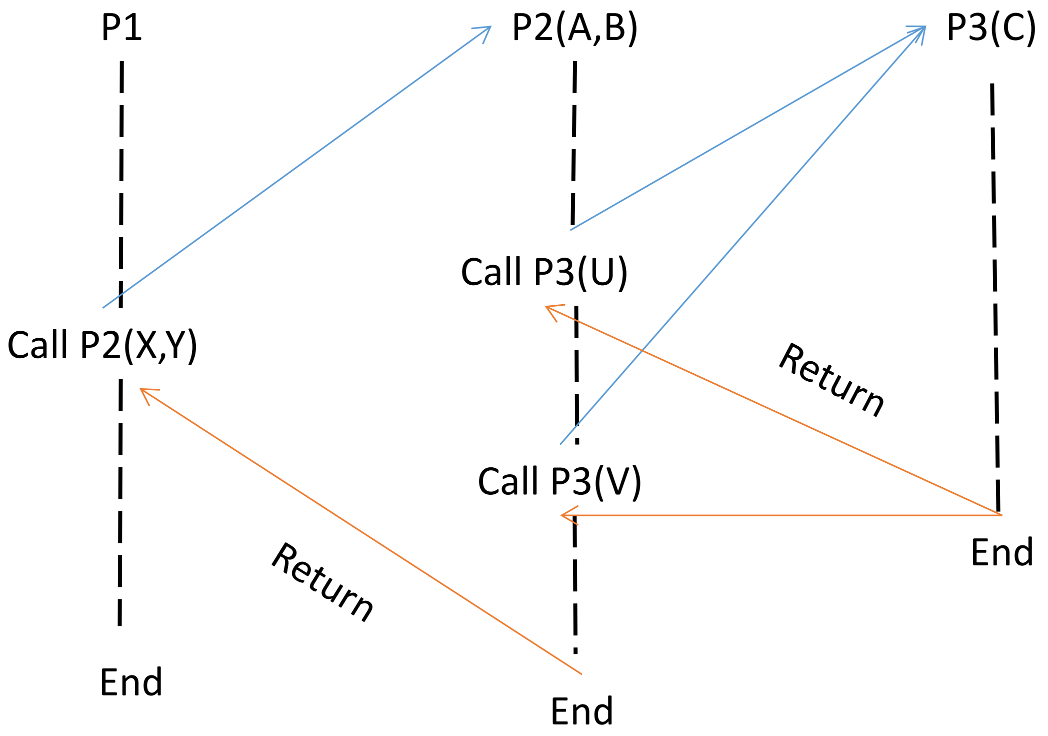
- By value
- By reference (or address)

By value: Store the values of the actual parameters in the data area of the callee.

By reference: Store the addresses of the actual parameters in the data area of the callee.
The callee accesses, indirectly, the data area of the caller, which is at the top of the stack.

Recursion / Semantic

Compilation concepts



Case of recursive procedures

Multiple copies of the data area (Da) and code.

The data area (Da) is associated with an execution.

Stacking and unstacking of data areas (Da) related to executions.

Recursion / Semantic

Transformation Technique

Case of functions (all parameters are called by value).

Main program
(First call)

Recursive function
containing recursive calls

Main program:

```
Read(n)
Write(Fact(n))
```

Recursive function

```
Fact(N)
X and Y local variables
IF N = 0
     Fact := 1
ELSE
      X := N-1;
      Y := Fact(X);
     Fact := N * Y
ENDIF
```

Recursion / Semantic

Transformation Technique

1. Define the data area (Da): local variables + parameters + Address field.

2. Define the call and return points.
There is always an initial call in the main program.

3. Call Translation
- Push the current data area onto the stack.
- Prepare the callee's data area:
  a) Pass parameters.
  b) Save the return address (call point).
- Branch to the beginning of the function.

4. Return Translation
- Retrieve the return address (Ret) from the current data area.
- Pop a data area.
- Branch to Ret.

5. Push a dummy data area at the beginning!

Recursion / Semantic

Transformation Technique

Main program:

```
Read(n)
Write(Fact(n))
```

Recursive function

```
Fact(N)
X and Y local variables
IF N = 0
    Fact := 1
ELSE
     X := N-1;
     Y := Fact(X);
    Fact := N * Y
ENDIF
```

What does the data area contain?
N, X, Y, and A (return address).
Cda denotes the current data area

There are two call points:

- Writing Fact(n) in the main program (Label 1:)
- Assignment of Fact(X) to Y (Label 2:)

There are two return points:

- After Fact := 1
- After Fact := N*Y

Recursion / Semantic

Transformation Technique

Main program

```
→ Read(n)
Write( A Fact(n))
```

Recursive function :

```
Fact(N)
X an Y are local variables
IF N = 0
    Fact := 1
     R
ELSE
     X := N-1;
     Y := A Fact(X);
    Fact := N * Y
     R
ENDIF
```

```
Read(n) ; Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
```

Recursion / Semantic

Transformation Technique

```
Programme principal:
Lire(n)
Ecrire( A  Fact(n))

Fonction récursive:
Fact(N)
X et Y variables locales
SI N = 0
    Fact := 1
     R
 SINON
     X := N-1;
     Y := A Fact(X);
    Fact := N * Y
     R
FSI
```

```
Read(n); Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N  ;
Cda.ReturnAddress := 1
```

Recursion / Semantic

Transformation Technique

```
Programme principal:
Lire(n)
Ecrire( A  Fact(n))

Fonction récursive:
Fact(N)
X et Y variables locales
→ SI N = 0
      Fact := 1
       R
   SINON
       X := N-1;
       Y := A Fact(X);
      Fact := N * Y
       R
FSI
```

```
Read(n); Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N
Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
```

Recursion / Semantic

Transformation Technique

```
Programme principal:
Lire(n)
Ecrire( A  Fact(n))

Fonction récursive:
Fact(N)
X et Y variables locales
SI N = 0
→      Fact := 1
       R
  SINON
       X := N-1;
       Y := A Fact(X);
      Fact := N * Y
       R
FSI
```

```
Read(n); Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N
Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
            Fact := 1
```

Recursion / Semantic

Transformation Technique

```
Programme principal:
Lire(n)
Ecrire( A  Fact(n))

Fonction récursive:
Fact(N)
X et Y variables locales
SI N = 0
    Fact := 1
→    R
 SINON
     X := N-1;
     Y := A Fact(X);
    Fact := N * Y
     R
FSI
```

```
Read(n); Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N
Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
            Fact := 1
            {  Simulate the return }
            I := Cda.A;  Pop(P,Cda)
            IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
```

Recursion / Semantic

Transformation Technique

Programme principal:

```
Lire(n)
Ecrire( A  Fact(n))
```

Fonction récursive:

```
Fact(N)
X et Y variables locales
SI N = 0
     Fact := 1
      R
 SINON
→     X := N-1;
      Y := A Fact(X);
     Fact := N * Y
      R
FSI
```

⇒

```
Read(n); Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N
Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
              Fact := 1
              {  Simulate the return }
              I := Cda.A;  Pop(P,Cda)
              IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
     Else
              Cda.X = Cda.N - 1
```

Recursion / Semantic

Transformation Technique

```
Programme principal:
Lire(n)
Ecrire( A  Fact(n))

Fonction récursive:
Fact(N)
X et Y variables locales
SI N = 0
     Fact := 1
      R
  SINON
      X := N-1;
→     Y := A Fact(X);
     Fact := N * Y
      R
FSI
```

⇒

```
read(n); Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N
Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
            Fact := 1
            {  Simulate the return }
            I := Cda.A;  Pop(P,Cda)
            IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
    Else
            Cda.X = Cda.N - 1
            {  Simulate the recursive call }
            Push(P,Cda);  Cda.N := Cda.X ;    Cda.A:= 2
            GOTO 10
    Endif
2:   Cda.Y := Fact ;
```

Recursion / Semantic

Transformation Technique

```
Programme principal:
Lire(n)
Ecrire( A  Fact(n))

Fonction récursive:
Fact(N)
X et Y variables locales
SI N = 0
     Fact := 1
      R
  SINON
      X := N-1;
      Y := A Fact(X);
→    Fact := N * Y
      R
FSI
```

⇒

```
read(n); Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N
Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
              Fact := 1
              {  Simulate the return }
              I := Cda.A;  Pop(P,Cda)
              IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
     Else
              Cda.X = Cda.N - 1
              {  Simulate the recursive call }
              Push(P,Cda);  Cda.N := Cda.X ;    Cda.A:= 2
              GOTO 10
     Endif
2:   Cda.Y := Fact ;
     Fact := Cda.N * Cda.Y
```

Recursion / Semantic

Transformation Technique

```
Programme principal:
Lire(n)
Ecrire( A  Fact(n))

Fonction récursive:
Fact(N)
X et Y variables locales
SI N = 0
     Fact := 1
      R
 SINON
      X := N-1;
      Y := A Fact(X);
     Fact := N * Y
→     R
FSI
```

```
Read(n); Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N
Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
            Fact := 1
            {  Simulate the return }
            I := Cda.A;  Pop(P,Cda)
            IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
    Else
            Cda.X = Cda.N - 1
            {  Simulate the recursive call }
            Push(P,Cda);  Cda.N := Cda.X ;    Cda.A:= 2
            GOTO 10
    Endif
2:  Cda.Y := Fact ;    Fact := Cda.N * Cda.Y
    { Simulate the return }
    I:= Cda.A;  Pop(P,Cda)
    IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
```

Recursion / Semantic

Transformation Technique

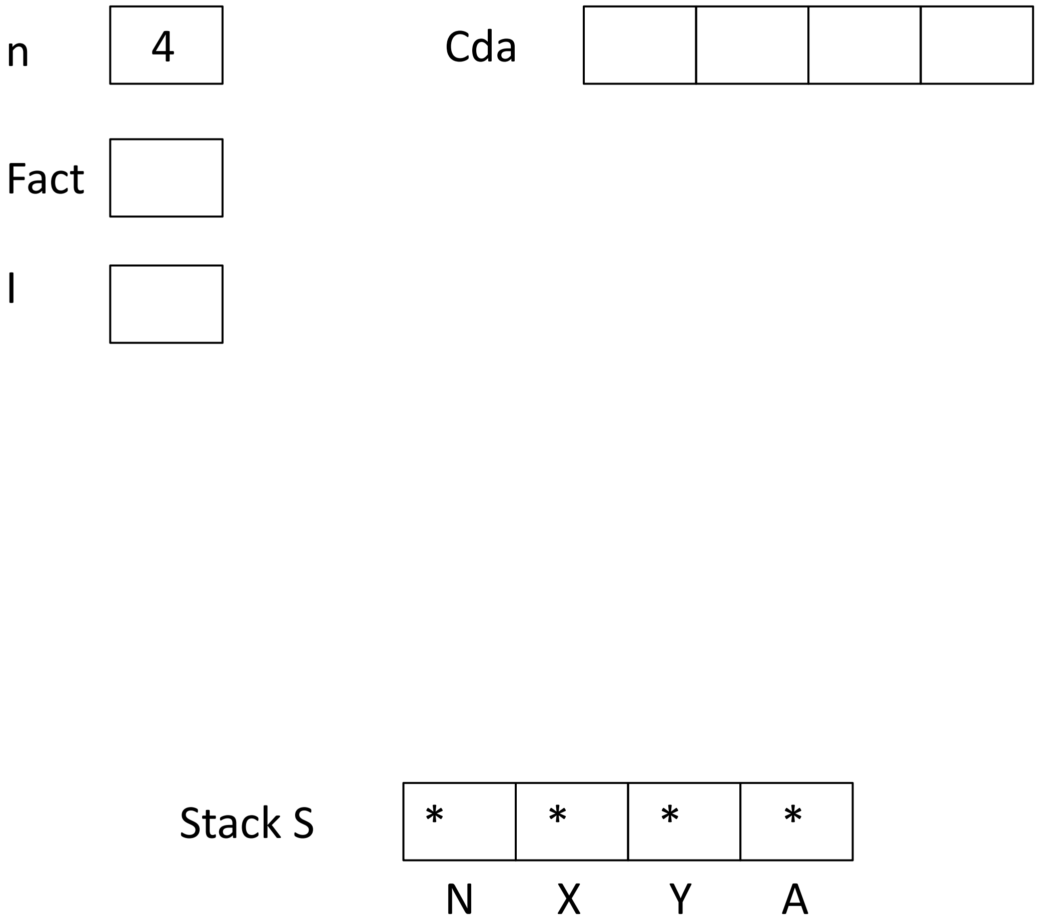
```
Programme principal:
Lire(n)
→ Ecrire( A  Fact(n))

Fonction récursive:
Fact(N)
X et Y variables locales
SI N = 0
     Fact := 1
      R
  SINON
      X := N-1;
      Y := A Fact(X);
     Fact := N * Y
      R
FSI
```

Rule 5 : Case n=0

```
Read(n) ; Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N
Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
              Fact := 1
              {  Simulate the return }
              I := Cda.A;  Pop(P,Cda)
              IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
     Else
              Cda.X = Cda.N - 1
              {  Simulate the recursive call }
              Push(P,Cda);  Cda.N := Cda.X ;    Cda.A:= 2
              GOTO 10
     Endif
2:   Cda.Y := Fact ;    Fact := Cda.N * Cda.Y
     { Simulate the return }
     I:= Cda.A;  Pop(P,Cda)
     IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
1:  Write (Fact)
```

Recursion / Semantic

Transformation Technique


n 4

Cda

Fact

I

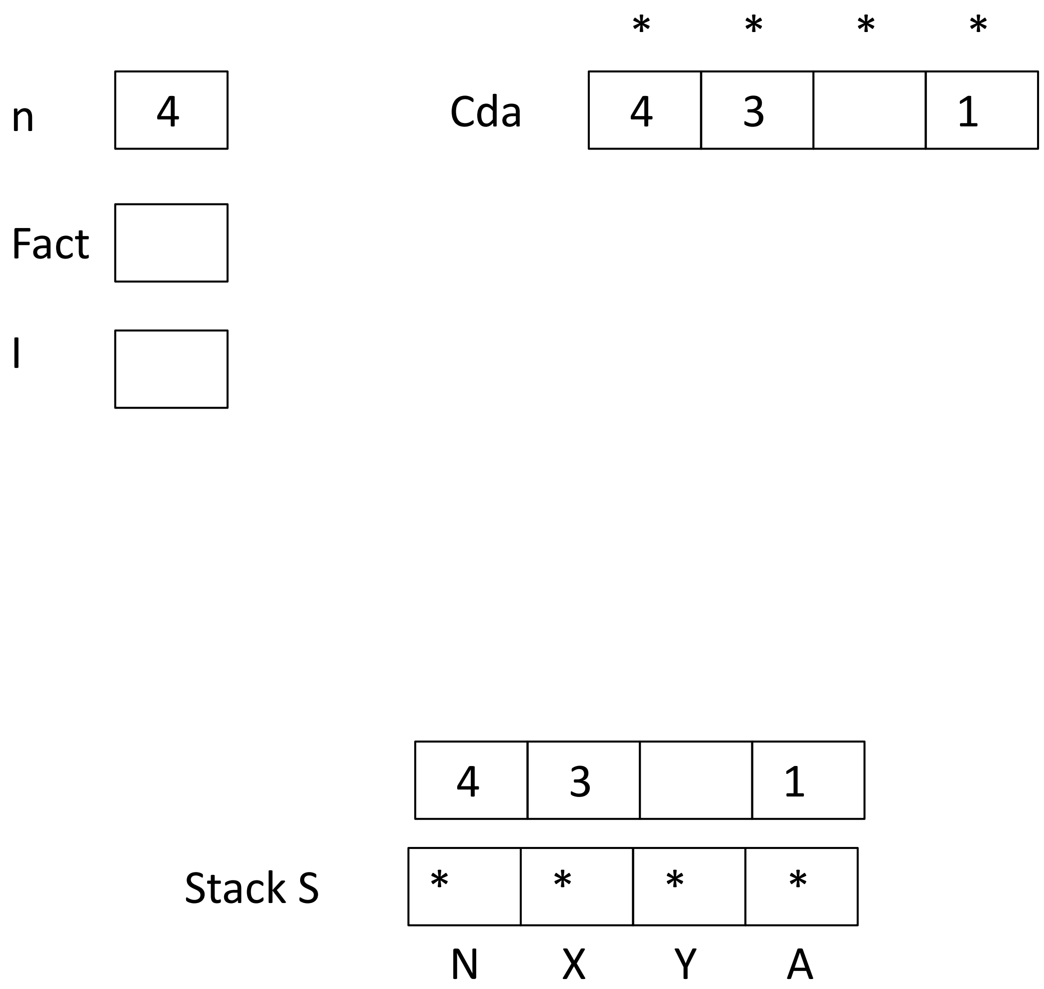
Stack S

N	X	Y	A
*	*	*	*

```
Read(n)
Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N; Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
              Fact := 1
              {  Simulate the return }
              I := Cda.A;  Pop(P,Cda)
              IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
     Else
              Cda.X = Cda.N - 1
              {  Simulate the recursive call }
              Push(P,Cda);  Cda.N := Cda.X ;    Cda.A:= 2
              GOTO 10
     Endif
2:   Cda.Y := Fact ;   Fact := Cda.N * Cda.Y
     { Simulate the return }
     I:= Cda.A;  Pop(P,Cda)
     IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
1:  Write (Fact)
```

Recursion / Semantic

Transformation Technique



n: 4

Fact:

I:

	*	*	*	*
Cda	4	3		1

Stack S

N	X	Y	A
4	3		1
*	*	*	*

```
Read(n)
Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N; Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
              Fact := 1
              {  Simulate the return }
              I := Cda.A;  Pop(P,Cda)
              IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
     Else
              Cda.X = Cda.N - 1
              {  Simulate the recursive call }
              Push(P,Cda);  Cda.N := Cda.X ;    Cda.A:= 2
              GOTO 10
     Endif
2:   Cda.Y := Fact ;    Fact := Cda.N * Cda.Y
     { Simulate the return }
     I:= Cda.A;  Pop(P,Cda)
     IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
1:   Write (Fact)
```

Recursion / Semantic

Transformation Technique

n	4
Fact	
I	

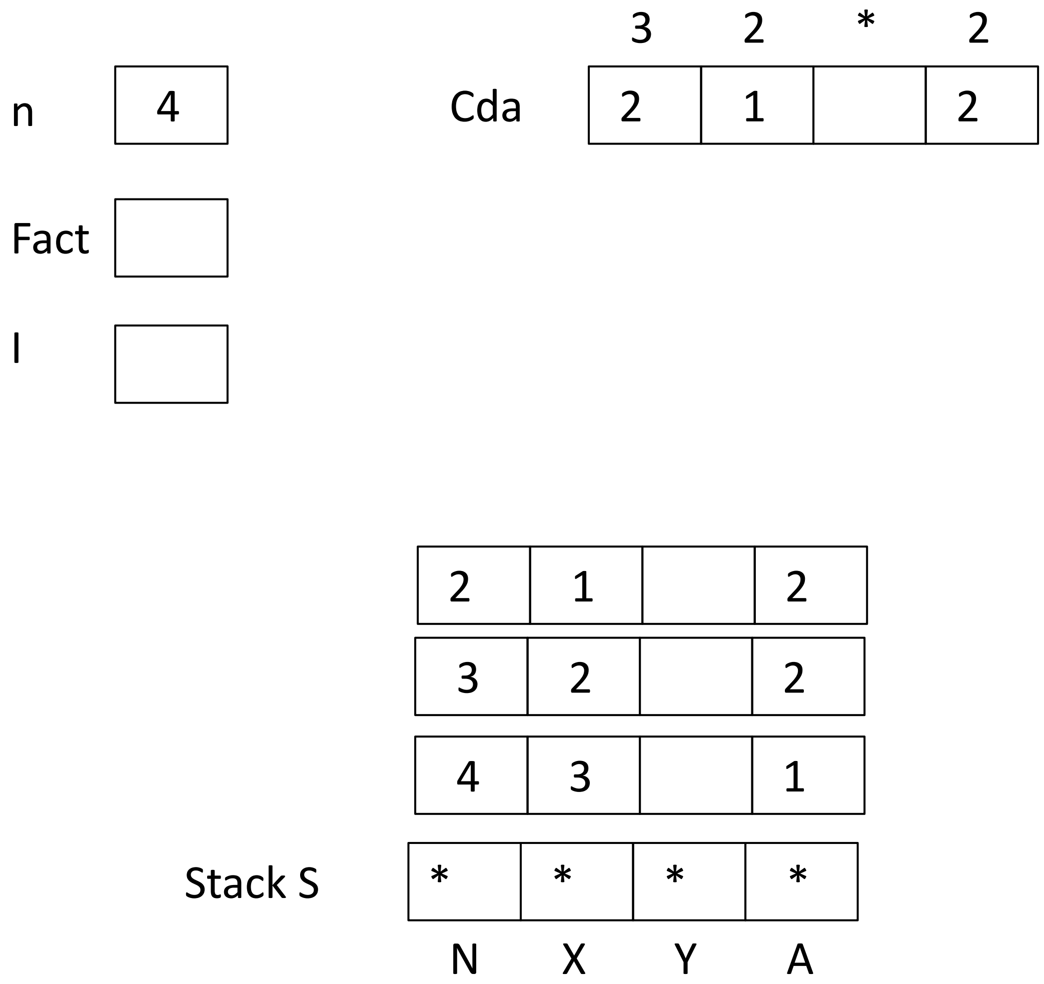
	4	3	*	1
Cda	3	2		2

	3	2		2
	4	3		1
Stack S	*	*	*	*
	N	X	Y	A

```
Read(n)
Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N; Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
            Fact := 1
            {  Simulate the return }
            I := Cda.A;  Pop(P,Cda)
            IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
    Else
            Cda.X = Cda.N - 1
            {  Simulate the recursive call }
            Push(P,Cda);  Cda.N := Cda.X ;     Cda.A:= 2
            GOTO 10
    Endif
2:  Cda.Y := Fact ;   Fact := Cda.N * Cda.Y
    { Simulate the return }
    I:= Cda.A;  Pop(P,Cda)
    IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
1:  Write (Fact)
```

Recursion / Semantic

Transformation Technique



n 4

Fact

I

Cda				
	3	2	*	2
	2	1		2

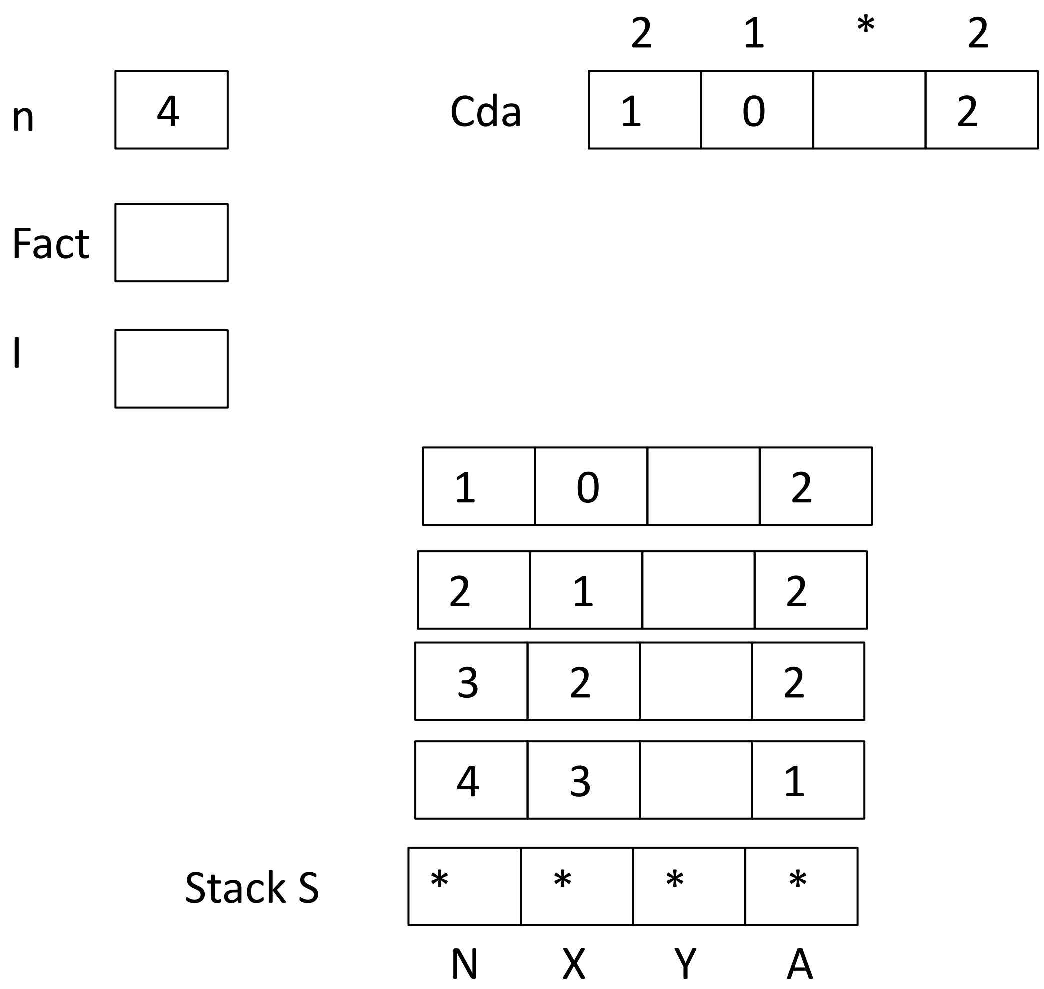
Stack S

N	X	Y	A
2	1		2
3	2		2
4	3		1
*	*	*	*

```
Read(n)
Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N; Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
              Fact := 1
              {  Simulate the return }
              I := Cda.A;  Pop(P,Cda)
              IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
     Else
              Cda.X = Cda.N - 1
              {  Simulate the recursive call }
              Push(P,Cda);  Cda.N := Cda.X ;     Cda.A:= 2
              GOTO 10
     Endif
2:    Cda.Y := Fact ;    Fact := Cda.N * Cda.Y
     { Simulate the return }
     I:= Cda.A;  Pop(P,Cda)
     IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
1:   Write (Fact)
```

Recursion / Semantic

Transformation Technique



n	4
Fact	
I	

Cda

2	1	*	2
1	0		2

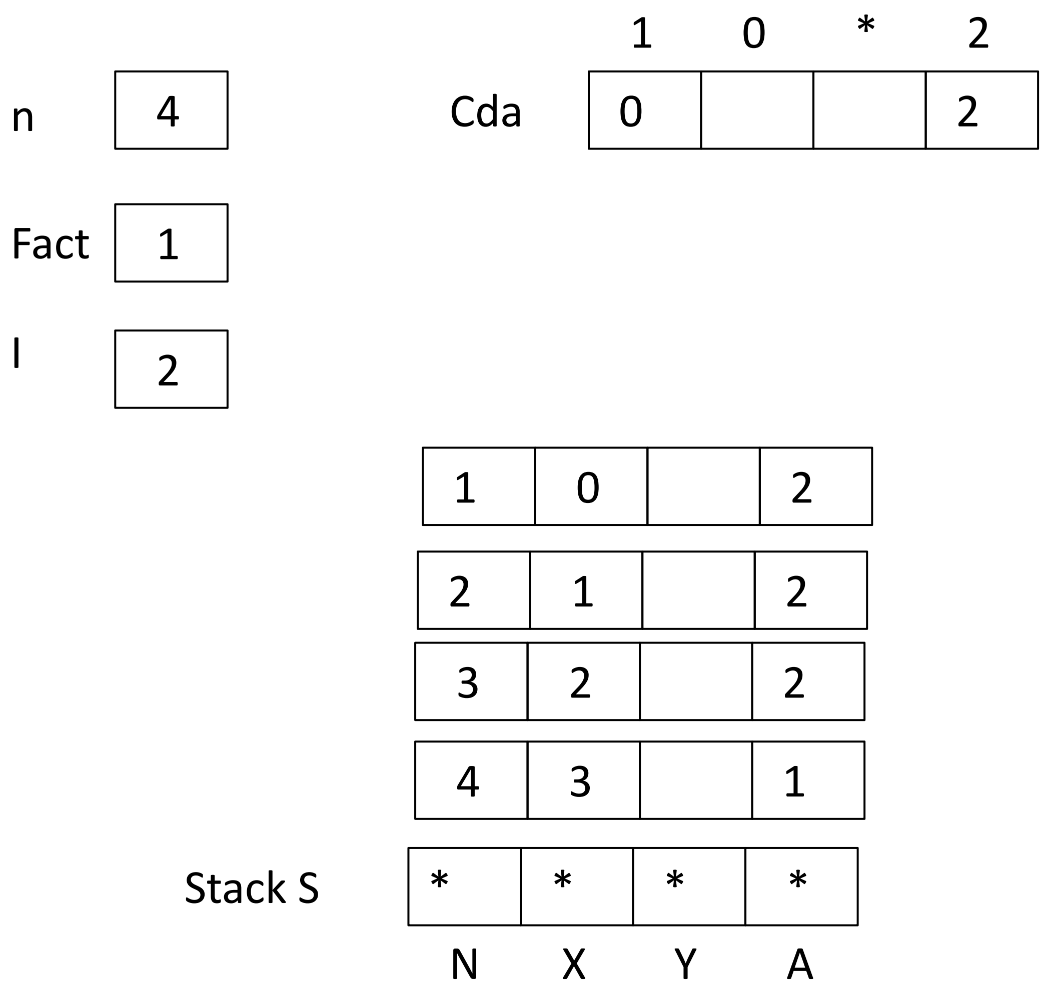
Stack S

N	X	Y	A
1	0		2
2	1		2
3	2		2
4	3		1
*	*	*	*

```
Read(n)
Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N; Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
            Fact := 1
            {  Simulate the return }
            I := Cda.A;  Pop(P,Cda)
            IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
      Else
            Cda.X = Cda.N - 1
            {  Simulate the recursive call }
            Push(P,Cda);  Cda.N := Cda.X ;    Cda.A:= 2
            GOTO 10
      Endif
2:    Cda.Y := Fact ;    Fact := Cda.N * Cda.Y
      { Simulate the return }
      I:= Cda.A;  Pop(P,Cda)
      IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
1:   Write (Fact)
```

Recursion / Semantic

Transformation Technique



1 0 * 2

n 4

Cda 0 | | | 2

Fact 1

I 2

1 | 0 | | 2

2 | 1 | | 2

3 | 2 | | 2

4 | 3 | | 1

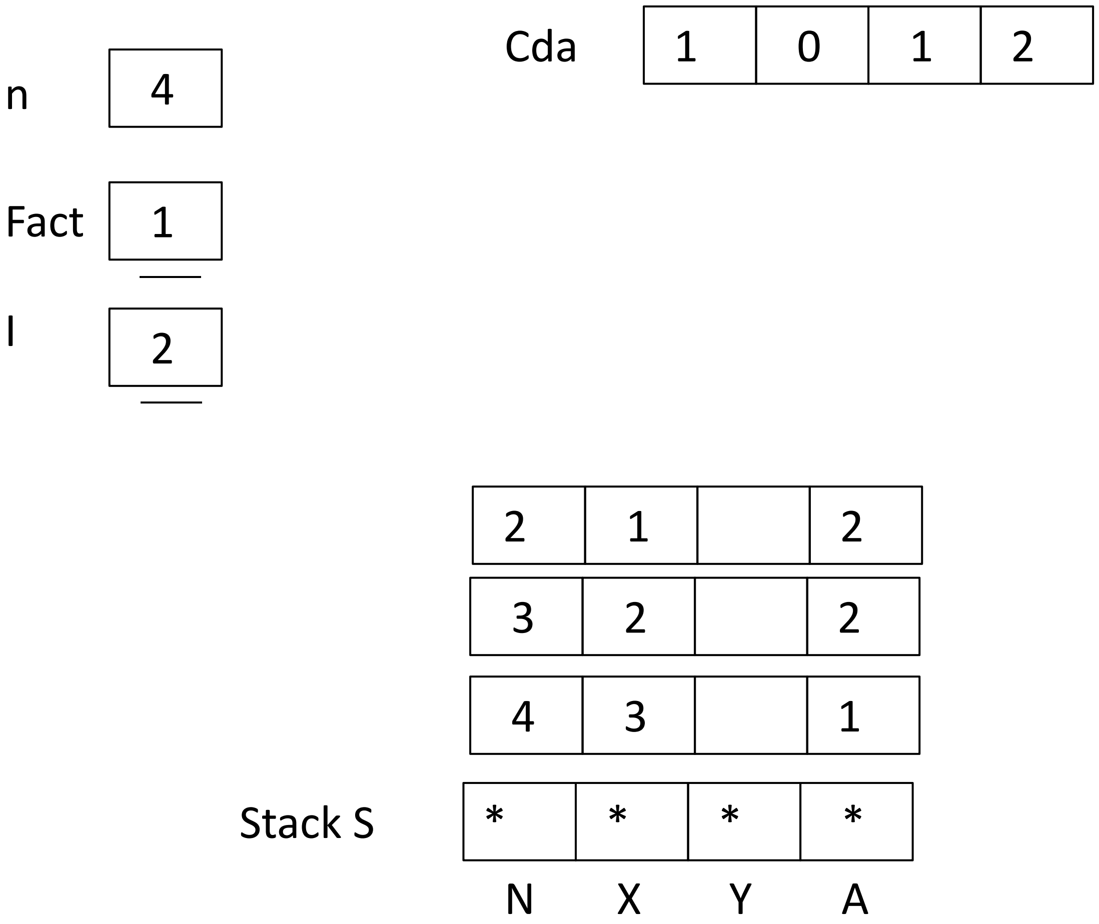
Stack S * | * | * | *

N X Y A

```
Read(n)
Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N; Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
            Fact := 1
            {  Simulate the return }
            I := Cda.A;  Pop(P,Cda)
            IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
     Else
            Cda.X = Cda.N - 1
            {  Simulate the recursive call }
            Push(P,Cda);  Cda.N := Cda.X ;    Cda.A:= 2
            GOTO 10
     Endif
2:   Cda.Y := Fact ;    Fact := Cda.N * Cda.Y
     { Simulate the return }
     I:= Cda.A;  Pop(P,Cda)
     IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
1:  Write (Fact)
```

Recursion / Semantic

Transformation Technique



n: 4

Fact: 1

I: 2

Cda: 1 | 0 | 1 | 2

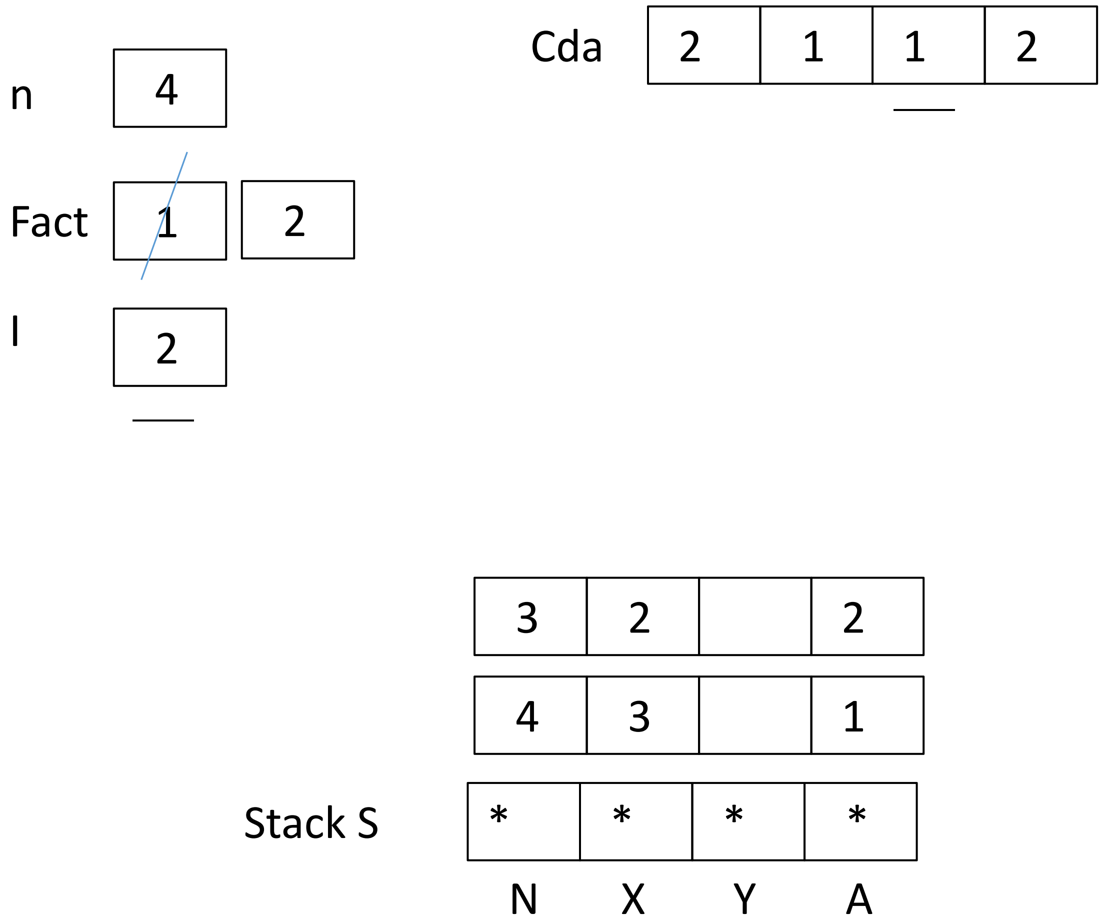
Stack S

2	1		2
3	2		2
4	3		1
*	*	*	*
N	X	Y	A

```
Read(n)
Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N; Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
            Fact := 1
            {  Simulate the return }
            I := Cda.A;  Pop(P,Cda)
            IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
    Else
            Cda.X = Cda.N - 1
            {  Simulate the recursive call }
            Push(P,Cda);  Cda.N := Cda.X ;    Cda.A:= 2
            GOTO 10
    Endif
2:   Cda.Y := Fact ;    Fact := Cda.N * Cda.Y
     { Simulate the return }
     I:= Cda.A;  Pop(P,Cda)
     IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
1:   Write (Fact)
```

Recursion / Semantic

Transformation Technique



Cda: 2 | 1 | 1 | 2

n: 4

Fact: ~~1~~ | 2

I: 2

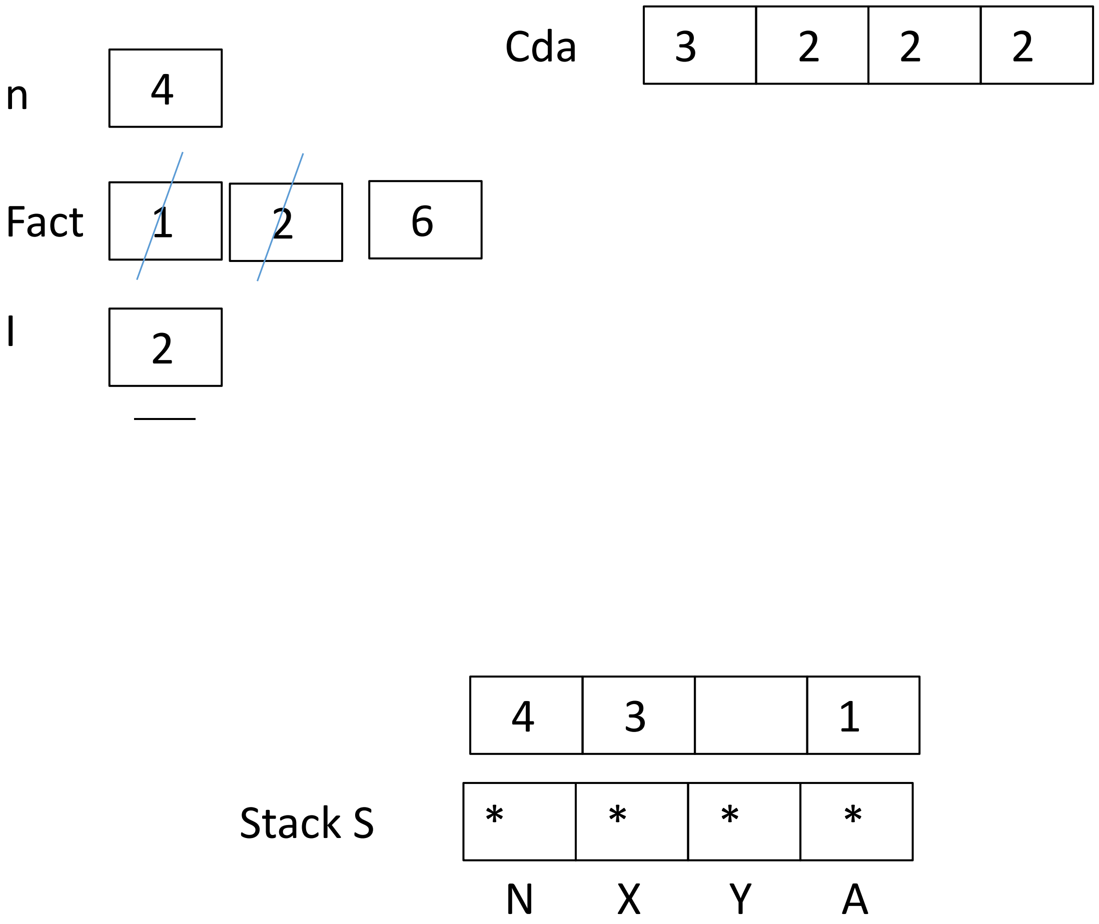
Stack S

N	X	Y	A
3	2		2
4	3		1
*	*	*	*

```
Read(n)
Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N; Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
                Fact := 1
                {  Simulate the return }
                I := Cda.A;  Pop(P,Cda)
                IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
      Else
                Cda.X = Cda.N - 1
                {  Simulate the recursive call }
                Push(P,Cda);  Cda.N := Cda.X ;    Cda.A:= 2
                GOTO 10
      Endif
2:    Cda.Y := Fact ;    Fact := Cda.N * Cda.Y
      { Simulate the return }
      I:= Cda.A;  Pop(P,Cda)
      IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
1:   Write (Fact)
```

Recursion / Semantic

Transformation Technique



```
Read(n)
Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N; Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
            Fact := 1
            {  Simulate the return }
            I := Cda.A;  Pop(P,Cda)
            IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
    Else
            Cda.X = Cda.N - 1
            {  Simulate the recursive call }
            Push(P,Cda);  Cda.N := Cda.X ;    Cda.A:= 2
            GOTO 10
    Endif
2:  Cda.Y := Fact ;   Fact := Cda.N * Cda.Y
    { Simulate the return }
    I:= Cda.A;  Pop(P,Cda)
    IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
1:  Write (Fact)
```

Recursion / Semantic

Transformation Technique

n: 4

Cda: 4 | 3 | 6 | 1

Fact: ~~1~~ | ~~2~~ | ~~6~~ | 24

I: ~~2~~ | 1

Stack S

N	X	Y	A
*	*	*	*

```
Read(n)
Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N; Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
            Fact := 1
            {  Simulate the return }
            I := Cda.A;  Pop(P,Cda)
            IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
    Else
            Cda.X = Cda.N - 1
            {  Simulate the recursive call }
            Push(P,Cda);  Cda.N := Cda.X ;    Cda.A:= 2
            GOTO 10
    Endif
2:   Cda.Y := Fact ;    Fact := Cda.N * Cda.Y
    { Simulate the return }
    I:= Cda.A;  Pop(P,Cda)
    IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
1:  Write (Fact)
```

Recursion / Semantic

Transformation Technique

Cda * * * *

n 4

Fact ~~1~~ ~~2~~ ~~6~~ 24

I ~~2~~ 1

Writing result
Fact(4) = 24

Stack S

N X Y A

```
Read(n)
Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N; Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
            Fact := 1
            {  Simulate the return }
            I := Cda.A;  Pop(P,Cda)
            IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
    Else
            Cda.X = Cda.N - 1
            {  Simulate the recursive call }
            Push(P,Cda);  Cda.N := Cda.X ;    Cda.A:= 2
            GOTO 10
    Endif
2:  Cda.Y := Fact ;    Fact := Cda.N * Cda.Y
    { Simulate the return }
    I:= Cda.A;  Pop(P,Cda)
    IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
1:  Write (Fact)
```

Recursion / Semantic

Refine Space

Transformation Technique

Do we need to use the variables X and Y in the data area (Da)?

Only include in the data area (Da) the relevant information after the call point.

X and Y are not necessary in the data area (Da):
- Y is never defined before the call
  (Cda.X = Cda.N - 1)
- X is not used after the call point
  (2: Cda.Y := Fact ; Fact := Cda.N * Cda.Y)

Rule: If there is only one call in the recursive module, it can be eliminated from the data area (Da).

How?
Replace the operation
Pop(P, Cda) with
Pop(P, Cda, Possible),
Possible = false if Popping an empty stack.

Consequence: Avoid pushing the "dummy" data area at the beginning.

Recursion / Semantic

```
Read(n)
Createstack(S)
{ Push a dummy data area onto the stack }
Push(S, Cda)
{ Initialize Cda }
Cda.Param := N; Cda.ReturnAddress := 1
{ Beginning of the simulated function }
10: IF Cda.Param = 0
            Fact := 1
            {  Simulate the return }
            I := Cda.A;  Pop(P,Cda)
            IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
    Else
            Cda.X = Cda.N - 1
            {  Simulate the recursive call }
            Push(P,Cda);  Cda.N := Cda.X ;    Cda.A:= 2
            GOTO 10
    Endif
2:   Cda.Y := Fact ;    Fact := Cda.N * Cda.Y
    { Simulate the return }
    I:= Cda.A;  Pop(P,Cda)
    IF I=1 GOTO 1 ELSE GOTO 2 ENDIF
1:   Write (Fact)
```

Refine Space

```
Read(N)
Createstack(S)
{ Initialize Cda }
Cda := N
10: IF Cda = 0
        Fact := 1
       {  Simulate the return }
        Pop(S, Cda, Possible)
        IF NOT Possible GOTO 1 ELSE GOTO 2 ENDIF
   ENDIF
   X := Cda - 1
  {  Simulate the recursive call }
   Push(S, Cda); Cda := X
  GOTO 10
2: Y := Fact
  Fact := Cda * Y
        {  Simulate the return }
        Pop(S, Cda, Possible)
        IF NOT Possible GOTO 1 ELSE GOTO 2 ENDIF
1: { End of the algorithm }
   Write(Fact)
```

Recursion / Semantic

```
Read(N)
Createstack(S)
{ Initialize Cda }
Cda := N
10: IF Cda = 0
          Fact := 1
         {  Simulate the return }
          Pop(S, Cda, Possible)
          IF NOT Possible GOTO 1 ELSE GOTO 2 ENDIF
    ENDIF
    X := Cda - 1
   {  Simulate the recursive call }
    Push(S, Cda); Cda := X
   GOTO 10
2: Y := Fact
   Fact := Cda * Y
          {  Simulate the return }
          Pop(S, Cda, Possible)
          IF NOT Possible GOTO 1 ELSE GOTO 2 ENDIF
1: { End of the algorithm }
    Write(Fact)
```

Refine Space

X and Y can be easily eliminated.

Rename Cda to X.

```
Read(N)
Createstack(S)
{ Initialize X }
X:= N
10: IF X= 0
          Fact := 1
         {  Simulate the return }
          Pop(S, X, Possible)
          IF NOT Possible GOTO 1 ELSE GOTO 2 ENDIF
    ENDIF
    X := X- 1
   {  Simulate the recursive call }
    Push(S, X);
   GOTO 10
2: Fact :=  Fact  * X
          {  Simulate the return }
          Pop(S, X, Possible)
          IF NOT Possible GOTO 1 ELSE GOTO 2 ENDIF
1: { End of the algorithm }
    Write(Fact)
```

Recursion / Semantic

```
Read(N)
Createstack(S)
{ Initialize X }
X:= N
10: IF X= 0
          Fact := 1
         {  Simulate the return }
          Pop(S, X, Possible)
          IF NOT Possible GOTO 1 ELSE GOTO 2 ENDIF
    ENDIF
    X := X- 1
   {  Simulate the recursive call }
    Push(S, X);
   GOTO 10
2: Fact :=  Fact  * X
          {  Simulate the return }
          Pop(S, X, Possible)
          IF NOT Possible GOTO 1 ELSE GOTO 2 ENDIF
1: { End of the algorithm }
    Write(Fact)
```

Eliminating Go to

The sequence

```
Pop(P, X, Possible)
IF Possible
   Go to 2
ELSE
   Go to 1
ENDIF
```

Write it only once

⇨

```
         Read(N)
         Createstack(S)
          X := N
10:       IF X = 0
                Fact:= 1
           ELSE
                Push(S, X)
                X := X - 1
               GOTO 10
           ENDIF
2:        Pop(S, X, Possible)
           IF NOT Possible
               GOTO 1
           ELSE
              Fact := X * Fact
              GOTO 2
          ENDIF
1:       Write(Fact)
```

Recursion / Semantic

```
     Read(N)
     Createstack(S)
      X := N
10:   IF X = 0
         Fact:= 1
      ELSE
         Push(S, X)
         X := X - 1
        GOTO 10
      ENDIF
2:    Pop(S, X, Possible)
      IF NOT Possible
        GOTO 1
      ELSE
        Fact := X * Fact
        GOTO 2
      ENDIF
1:   Write(Fact)
```

Eliminating Go to

Presence of two independent loops.

⇒

```
Read(N)
Createstack(S)
X := N
WHILE X <> 0 :
    Push(S, X)
    X := X - 1
ENDWHILE
Fact := 1
Pop(S, X, Possible)
WHILE Possible
    Fact := X * Fact
    Pop(S, X, Possible)
ENDWHILE
Write(Fact)
```

Recursion / Semantic

Eliminating the stack

```
Read(N)
Createstack(S)
X := N
WHILE X <> 0 :
        Push(S, X)
        X := X - 1
ENDWHILE
Fact := 1
Pop(S, X, Possible)
WHILE Possible
       Fact := X * Fact
       Pop(S, X, Possible)
ENDWHILE
Write(Fact)
```

First loop: stacks the first n natural numbers.
Second loop: retrieves the elements.

Solution: generate the numbers through a "FOR" loop.

```
Read(n)
Fact := 1
FOR X = 1 , n :
      Fact := Fact * X
ENDFOR
Write(Fact)
```

Recursion / Semantic

Other rules

Case of procedures

- Only input parameters should be placed in the data area.
- Output parameters are considered as global variables.

P(E1, E2, ..., S1, S2, ...)

Utilisation des variables globales
il est conseillé de mettre les tableaux comme variables globales

Avoid arrays as 'Value' parameters:
Treat them as 'Reference' parameters.

Example:
Several arrays T1, T2
Sum (Ti)
(Sum: recursive function)

Calling at the end of the procedure.
Code reduction:
- Change the values in the data area with the new parameters.
- Branch to the beginning of the procedure.

Otherwise:
- Push the data area of the caller.
- Prepare the data area of the callee.
- Go to the beginning of the callee.
- Retrieve the address.
- Pop the data area of the caller.
- Go to the retrieved address.