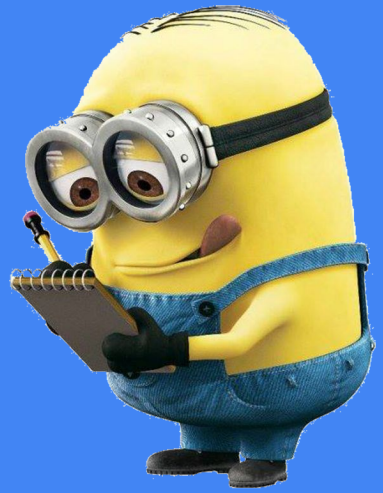
Teoría de Lenguajes
Teoría de la Programación

Clase 2: Introducción parte 2 - Modelo Declarativo

Ejercicios!



1) Devolver el máximo de una lista de enteros

2) A partir de una lista de números devolver una lista de los valores absolutos

Oz - Evaluación perezosa (Lazy)

Eager vs Lazy

En la evaluación perezosa los cálculos se hacen recién al momento en que se necesitan

```
fun lazy {Ints N}
   N|{Ints N+1}
end
```

Oz - Alto orden (high order programming)

Funciones como ciudadanos de primera clase (first class)

Poder pasar funciones como parámetro a otras funciones se conoce como programación de alto orden

Oz - Concurrencia

Threads

Comunicación entre threads

```
local A B C in
   thread  A = B + C  end
   C = 4
   thread  B = 10  end
   {Browse A}
end
```

Oz - Dataflow

¿Qué pasa si queremos usar una variable que todavía no está ligada a un valor?

- ¿Se lanza un error?
- ¿Se asume un valor por defecto?
- ¿Se detiene la ejecución?

Oz - Estado explícito

Explícito vs Implícito

Utilización de estados con celdas de memoria:

- NewCell
- @
- :=

```
C={NewCell 0} C:=@C+1 {Browse @C}
```

Oz - Objetos

¿Con los conceptos que vimos hasta ahora, cómo definirían un objeto como lo conocen?

Al objeto lo vamos a definir por ahora como una función con memoria interna

Oz - Clases

¿Cómo hacemos para tener más de una instancia de un objeto?

Definimos a la clase como una fábrica de objetos. Permite crear instancias.

Oz - Indeterminismo y atomicidad

Concurrencia + estados
=
Indeterminismos

Atomicidad de operaciones con Locks

Oz - Procedimientos

Los procedimientos son conceptos más básicos que las funciones.

Se utilizan variables no ligadas como parámetros

PARTE 2
MODELO COMPUTACIONAL

Modelo computacional

Sistema formal que define cómo se ejecutan los cálculos. Se define en términos de los conceptos que incluye. Sintaxis y semántica

Un modelo computacional es una definición más precisa de un paradigma de programación.

¿Qué nos permite estudiar un modelo computacional?

Modelo computacional

- Correctitud
- Complejidad temporal
- Complejidad espacial

Modelo declarativo

Modelo declarativo

- Evaluar funciones sobre estructuras de datos(parciales)
  - Sin estado (Stateless)
  - Paralelizable
  - Deterministico
  - Sin efectos secundarios
  - Fácil de probar

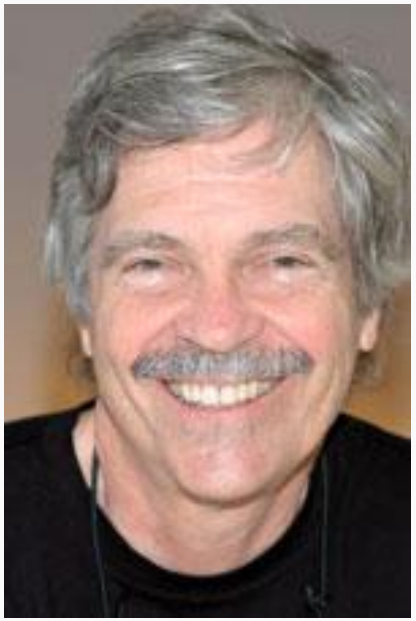
- Ideas principales del paradigma funcional y lógico

Modelo declarativo

Edsger Dijkstra (1930-2002)

“Object-oriented programming is an exceptionally bad idea which could only have originated in California”

Modelo declarativo



Alan Kay (1940)

Entre otras cosas, creador de Smalltalk

“You probably know that arrogance, in computer science, is measured in nanodijkstras”

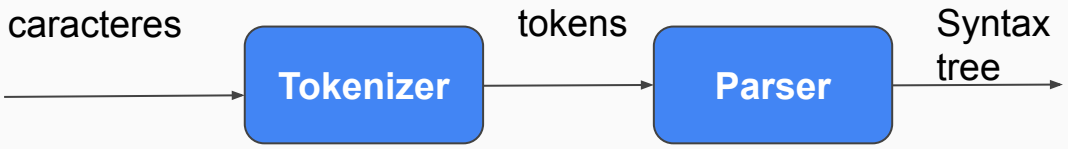
Modelo declarativo

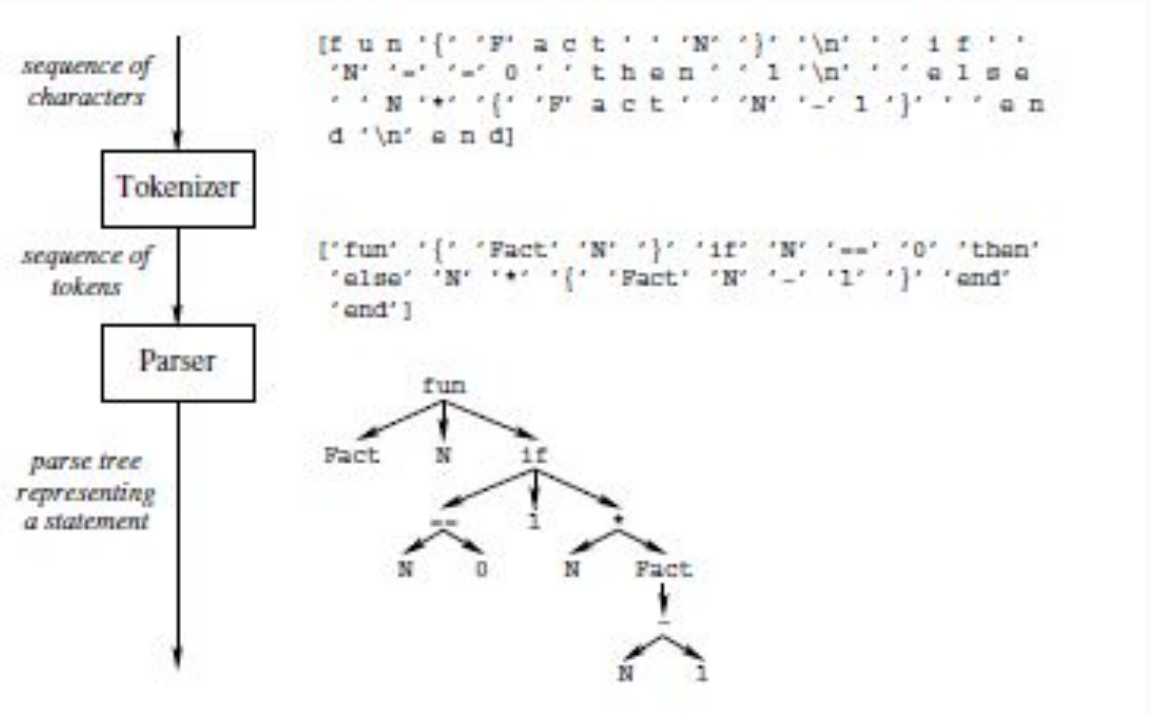
Una torta es 45 minutos de 200°C de calor aplicado a 200 mg de masa de torta final.
100 mg de masa de torta final es una mezcla de 99 mg de masa de torta etapa 2 y 1 mg de sal.
99 mg de masa de torta etapa 2 es una mezcla de 79 mg de harina y un huevo.

¿Cómo definimos un lenguaje?

Sintaxis

Qué es legal en un programa sin importar qué es lo que el programa esté haciendo





Sintaxis - EBNF

Extended Backus-Naur Form

```
<digit> ::== 0|1|2|3|4|5|6|7|8|9
<int> ::== <digit>{<digit>}
```

Semántica

Qué es lo que hace el programa al ejecutarse

Debe:

- Ser simple
- Permitir entender el programa en cuanto a correctitud y complejidad

Semántica - Kernel Language approach

- Set mínimo de instrucciones intuitivas
- Fácil de razonar, con semántica formal
- Se extiende a un lenguaje práctico a través de
  - Syntactic sugar
  - Linguistic abstraction

Lenguaje Kernel

Lenguaje Kernel - Valores

⟨v⟩	::=	⟨number⟩ | ⟨record⟩ | ⟨procedure⟩
⟨number⟩	::=	⟨int⟩ | ⟨float⟩
⟨record⟩, ⟨pattern⟩	::=	⟨literal⟩
	|	⟨literal⟩(⟨feature⟩$_1$: ⟨x⟩$_1$ $\cdots$ ⟨feature⟩$_n$: ⟨x⟩$_n$)
⟨procedure⟩	::=	**proc** { $ ⟨x⟩$_1$ $\cdots$ ⟨x⟩$_n$ } ⟨s⟩ **end**
⟨literal⟩	::=	⟨atom⟩ | ⟨bool⟩
⟨feature⟩	::=	⟨atom⟩ | ⟨bool⟩ | ⟨int⟩
⟨bool⟩	::=	**true** | **false**

Lenguaje Kernel - Statements

⟨s⟩ ::=	
skip	Empty statement
⟨s⟩$_1$ ⟨s⟩$_2$	Statement sequence
local ⟨x⟩ **in** ⟨s⟩ **end**	Variable creation
⟨x⟩$_1$=⟨x⟩$_2$	Variable-variable binding
⟨x⟩=⟨v⟩	Value creation
if ⟨x⟩ **then** ⟨s⟩$_1$ **else** ⟨s⟩$_2$ **end**	Conditional
case ⟨x⟩ **of** ⟨pattern⟩ **then** ⟨s⟩$_1$ **else** ⟨s⟩$_2$ **end**	Pattern matching
{ ⟨x⟩ ⟨y⟩$_1$ ... ⟨y⟩$_n$ }	Procedure application

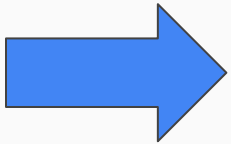
Expressions vs Statements

Una expresión es un valor. Es el resultado de una operación. Es algo que puede ser asignado a una variable

Un statement es una secuencia de operaciones que no devuelven valor

Múltiples declaraciones

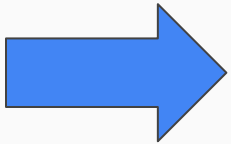
```
local A B in
   A = 4
   B = 5 + A
end
```



```
local A in
   local B in
      A = 4
      B = 5 + A
   end
end
```

Declaraciones sin asignación

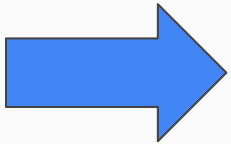
```
local X=<expression>
in
    <statement>
end
```



```
local X in
    X = <expression>
<statement>
end
```

Funciones a procedimientos

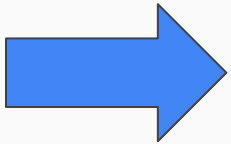
```
X = {F Y}
```



```
{F X Y}
```

Llamados anidados

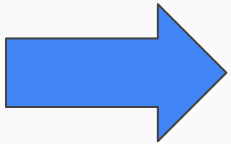
```
{P {F A X} Y}
```



```
local U in
    {F A X U}
    {P U Y}
end
```

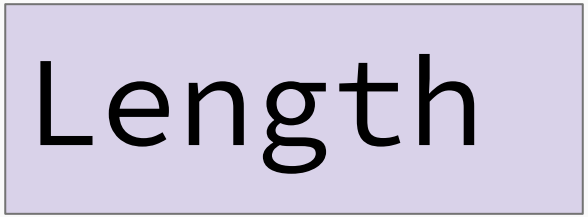
Condicionales booleanos

```
if X > Y then
    <statement>
end
```



```
local B in
    B = X > Y
    if B then
        <statement>
    end
end
```

Ejemplo



Scoping y procedures

Scoping - local scoping

```
local X in
        X=1
        local X in
            X=2
            {Browse X} % Muestra 2
        end
        {Browse X} % Muestra 1
    end
```

Scoping - static vs dynamic scoping

```
local P Q in
        proc {Q X} {Browse stat(X) end
        proc {P X} {Q X} end
        local Q in
            proc {Q X} {Browse dyn(X)} end
            {P ‘hola’}
        end
    end
```

Scoping - static vs dynamic scoping

```
local P Q in
        proc {Q X} {Browse stat(X) end
        proc {P X} {Q X} end
        local Q in
            proc {Q X} {Browse dyn(X)} end
            {P ‘hola’}
        end
    end
```

Abstraccion procedural

Cualquier statement puede convertirse en un procedimiento

En un statement un identificador es libre si no está definido en el mismo

Máquina abstracta

Single Assignment Store

σ

- Variables declarativas
- Value store
- Partial values
- Variables dataflow

Ej:

$\sigma = \{x_1=100, x_2=[1\ 2\ 3], x_3\}$

Entorno

E

- Identificadores

Notación:

<x> identificador de una variable

E(<x>) el valor en el store del identificador <x> según su entorno

Operaciones:

Adición: E’ = E + {<x>-> x}

Restricción: $E' = E|_{\{<x>,..<z>\}}$

STACK

ST

- Pila de semantic statements

Semantic statement es un par (<s>,E) siendo <s> un statement

Cómputo

- Estado de ejecución.
- Un cómputo define cómo se modifica el estado de ejecución de un programa.

$$(ST_0, \sigma_0) \rightarrow (ST_1, \sigma_1) \rightarrow (ST_2, \sigma_2) \rightarrow \ldots$$

Estados de ejecución

- Ejecutable
- Terminado
- Suspendido

Semántica

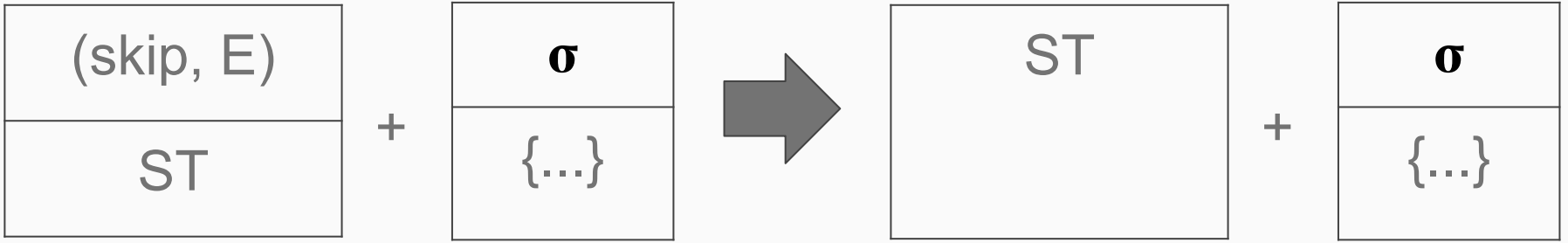
Statements

⟨s⟩ ::=	
skip	Empty statement
⟨s⟩$_1$ ⟨s⟩$_2$	Statement sequence
local ⟨x⟩ **in** ⟨s⟩ **end**	Variable creation
⟨x⟩$_1$=⟨x⟩$_2$	Variable-variable binding
⟨x⟩=⟨v⟩	Value creation
if ⟨x⟩ **then** ⟨s⟩$_1$ **else** ⟨s⟩$_2$ **end**	Conditional
case ⟨x⟩ **of** ⟨pattern⟩ **then** ⟨s⟩$_1$ **else** ⟨s⟩$_2$ **end**	Pattern matching
{ ⟨x⟩ ⟨y⟩$_1$ ... ⟨y⟩$_n$ }	Procedure application

Skip

En el tope del ST tenemos el siguiente semantic statement (skip, E)

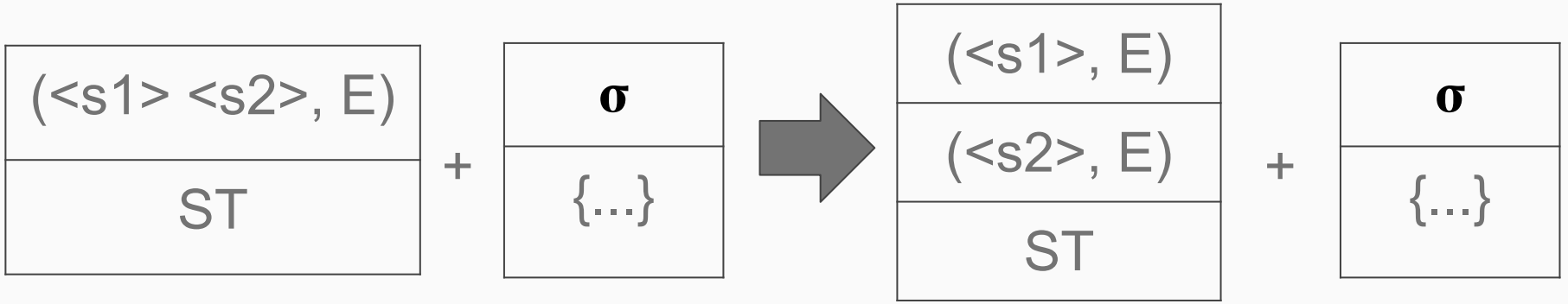
Se continua con el próximo paso



Composición secuencial

En el tope del ST tenemos el siguiente semantic statement (<s_1> <s_2>, E)

Se apila cada statement en el ST con el mismo entorno de la composición

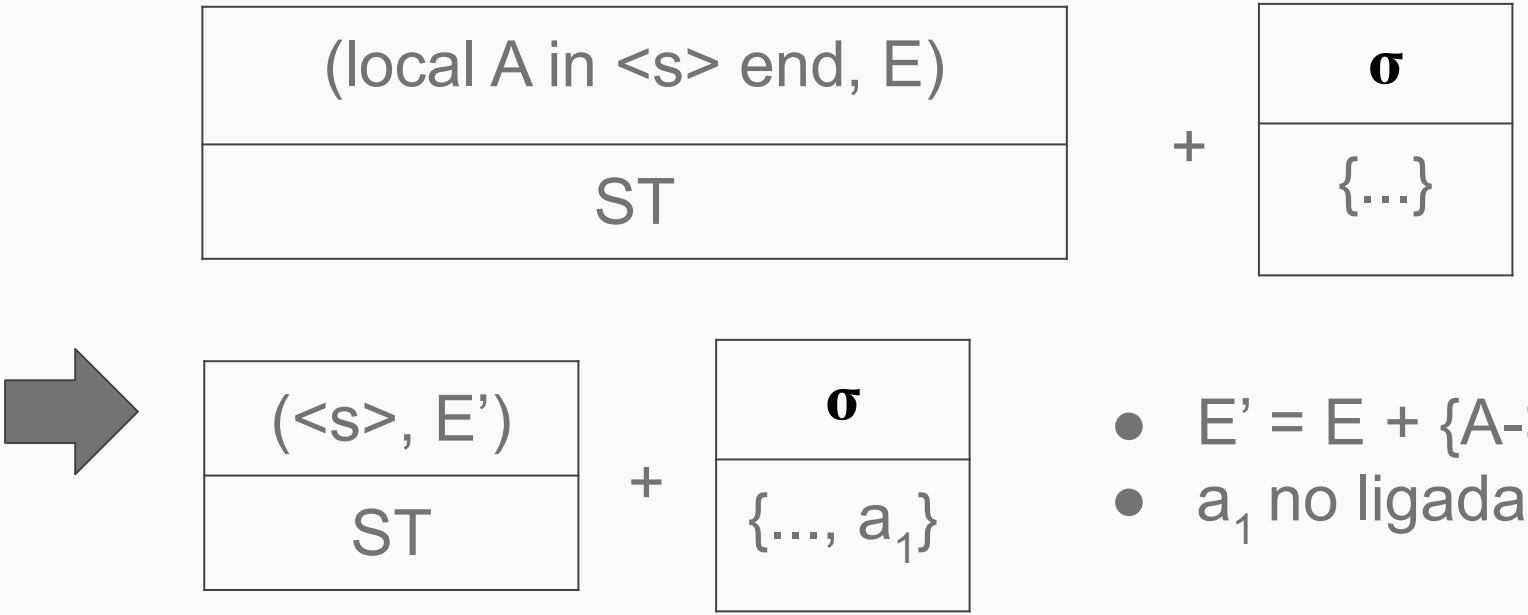


Declaración de variable

En el tope del ST tenemos el siguiente semantic statement
(local <x> in <s> end , E)

- Se crea la variable x en el store
- Se crea un ambiente E' = E + {<x> -> x}. Es decir un ambiente igual al anterior pero con el identificador <x> mapeando a la variable recién creada
- Se apila (<s>,E') al ST
- Se continua con la próxima ejecución

Declaración de variable

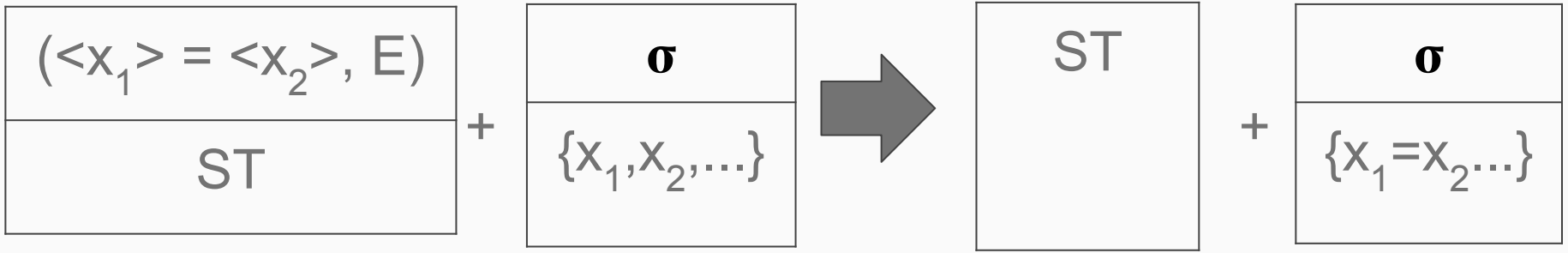


- E' = E + {A->a_1}
- a_1 no ligada en σ

Igualdad variable - variable

En el tope del ST tenemos el siguiente semantic statement ($\langle x_1 \rangle = \langle x_2 \rangle$, E)

Se hace un bind de E($\langle x_1 \rangle$) con E($\langle x_2 \rangle$) en el store

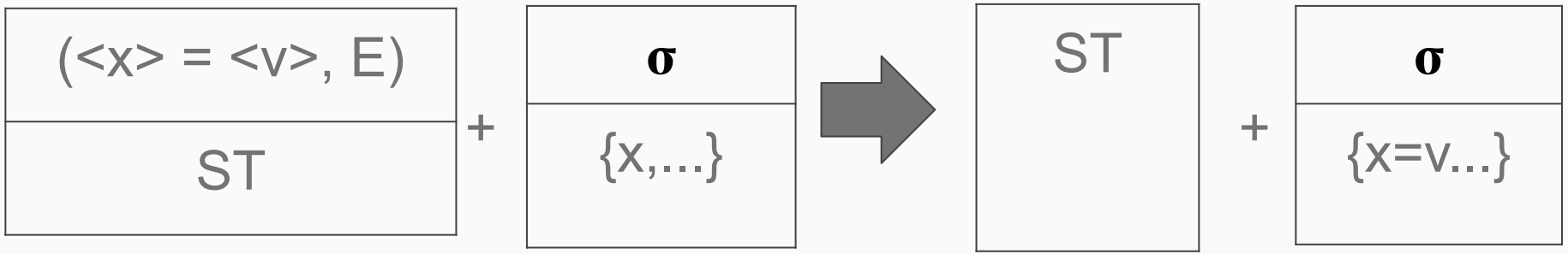


Igualdad variable - valor

En el tope del ST tenemos el siguiente semantic statement (<x> = <v>, E)

<v> es un record, un numero o un procedimiento

Se crea el valor y se liga a la variable <x> en el store



WHAAAT?

EJEMPLO

```
local X Y Z in
    X = 10
    Y = 40
    Z = X
    local X A in
        X = 30
        A = persona(nombre:'Lean' edad:X)
    end
end
```

Bibliografía

- **Concepts, Techniques, and Models of Computer Programming - Capítulo 2**, Peter Van Roy and Seif Haridi
- **Extras:**
  - **A practical introduction to functional programming (with Python)**

    https://maryrosecook.com/blog/post/a-practical-introduction-to-functional-programming

  - **Abstract syntax tree** https://en.wikipedia.org/wiki/Abstract_syntax_tree