Caso de Estudio

Alejandro del Hoyo Abad
Sergio Pozuelo Martín-Consuegra

Escribir procedimientos en lenguaje C para manipular matrices y crear una librería que pueda integrarse en un programa SWI-Prolog

¿En qué consiste?

SWI-Prolog ⟷ Foreign Language Interface ⟷ **C**

Tipos de enlace a bibliotecas

Estáticos

- Enlazado en Tiempo de Compilación
- Predicados Disponibles sin Inicialización
- Portabilidad y Depuración Simplificada

Dinámicos

- Enlazado en Tiempo de Ejecución
- Necesidad de Cargar Bibliotecas Externas
- Protección de datos

Compilación

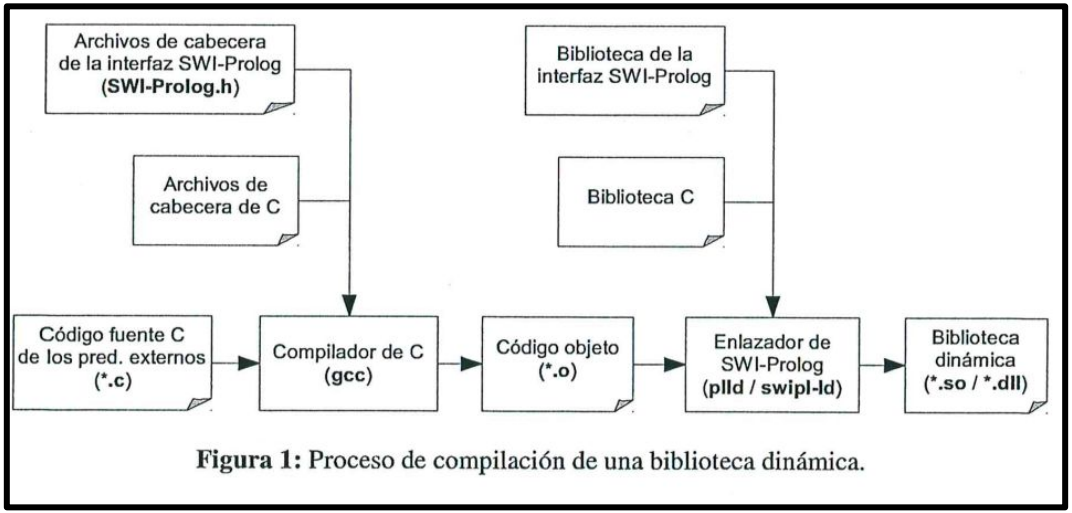


Figura 1: Proceso de compilación de una biblioteca dinámica.

1

```
$ swipl-ld -I/include file.c -o file.o
```

2

```
$ swipl-ld  -o -shared dynamic_library.so -o file.o
```

*.dll en windows

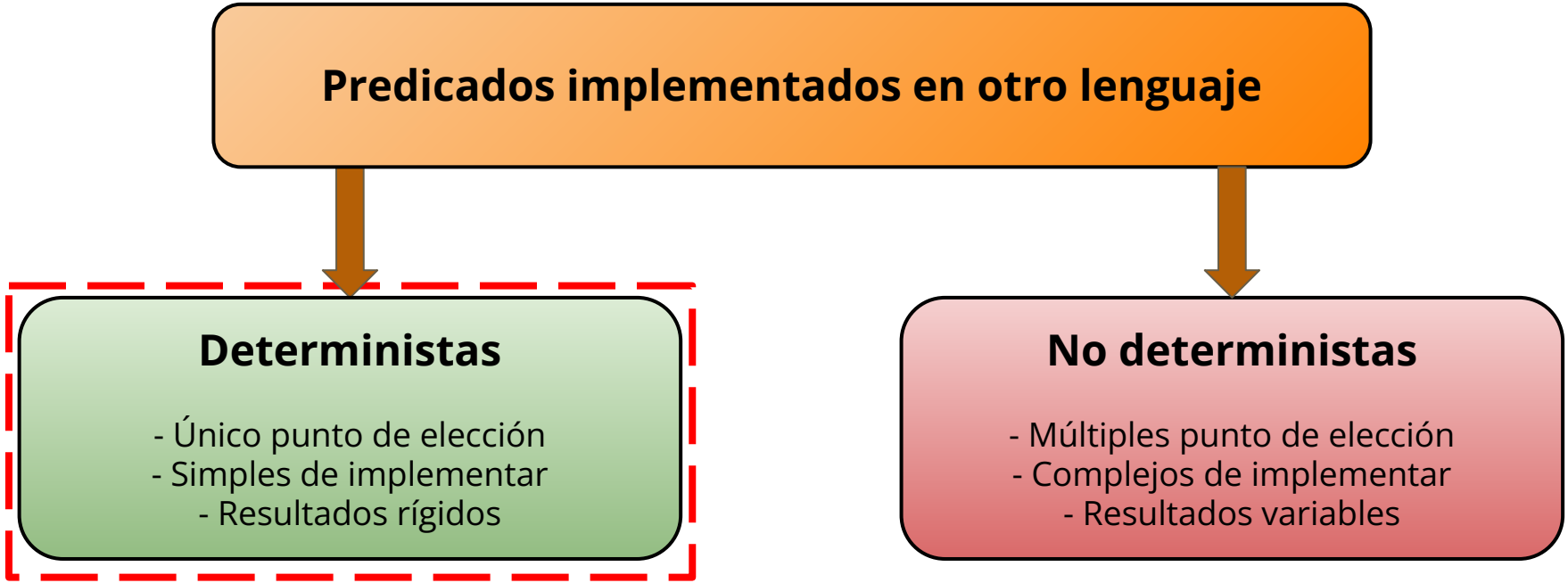
Ejecución

Proceso de conversión de términos de entrada a variables que maneja C y convertirlos de nuevo a términos que entienda Prolog tras realizar los respectivos cómputos.

?- load_foreign_library(+library)

```
?- sumar_matrices([[1,2,3],[2,4,2.42]],[[1,2,23.2],[1.23,2.2,3]],R).
R = [[2.0, 4.0, 26.2], [3.23, 6.2, 5.42]].
```

Predicados externos

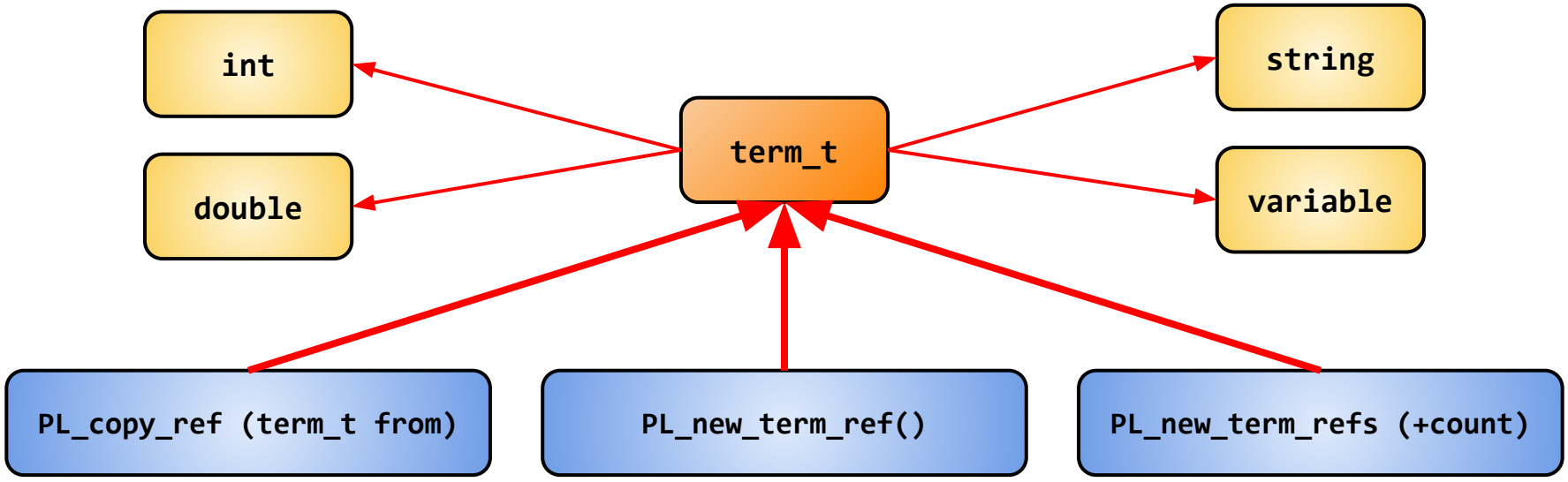


Predicados implementados

- Suma
- Resta
- Multiplicación
- Traspuesta
- Producto escalar
- Valor Máximo
- Comprobar Diagonal
- Multiplicar por factor
- Dividir por factor
- Sumar elementos
- Comprobar Diagonal superior
- Comprobar dimensiones iguales

```
PL_register_foreign("sumar_matrices", 3 , pl_matric
PL_register_foreign("restar_matrices", 3 , pl_matri
PL_register_foreign("multiplicar_matrices", 3, pl_m
PL_register_foreign("transponer", 2, pl_matrices_tr
PL_register_foreign("producto_escalar", 3, pl_vecto
PL_register_foreign("obtener_valor_maximo", 2, pl_o
PL_register_foreign("es_diagonal", 1, pl_is_diagona
PL_register_foreign("multiplicar_matriz_por_factor"
PL_register_foreign("dividir_matriz_por_factor", 3,
PL_register_foreign("sumar_elementos_de_matriz", 2,
PL_register_foreign("es_matriz_diagonal_superior",
PL_register_foreign("matrices_mismas_dimensions", 2
```

Tipos de datos de interfaz

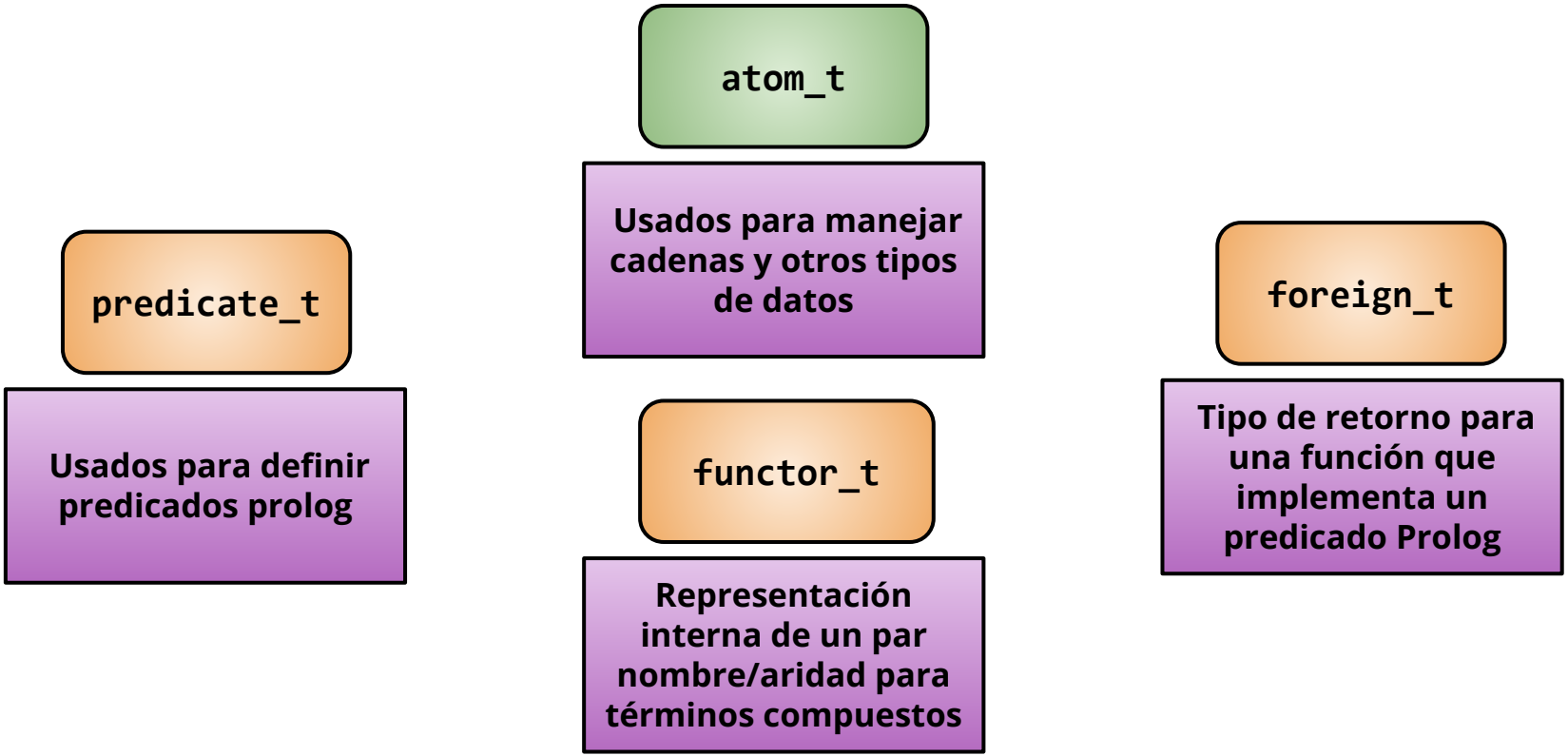


Copia un término

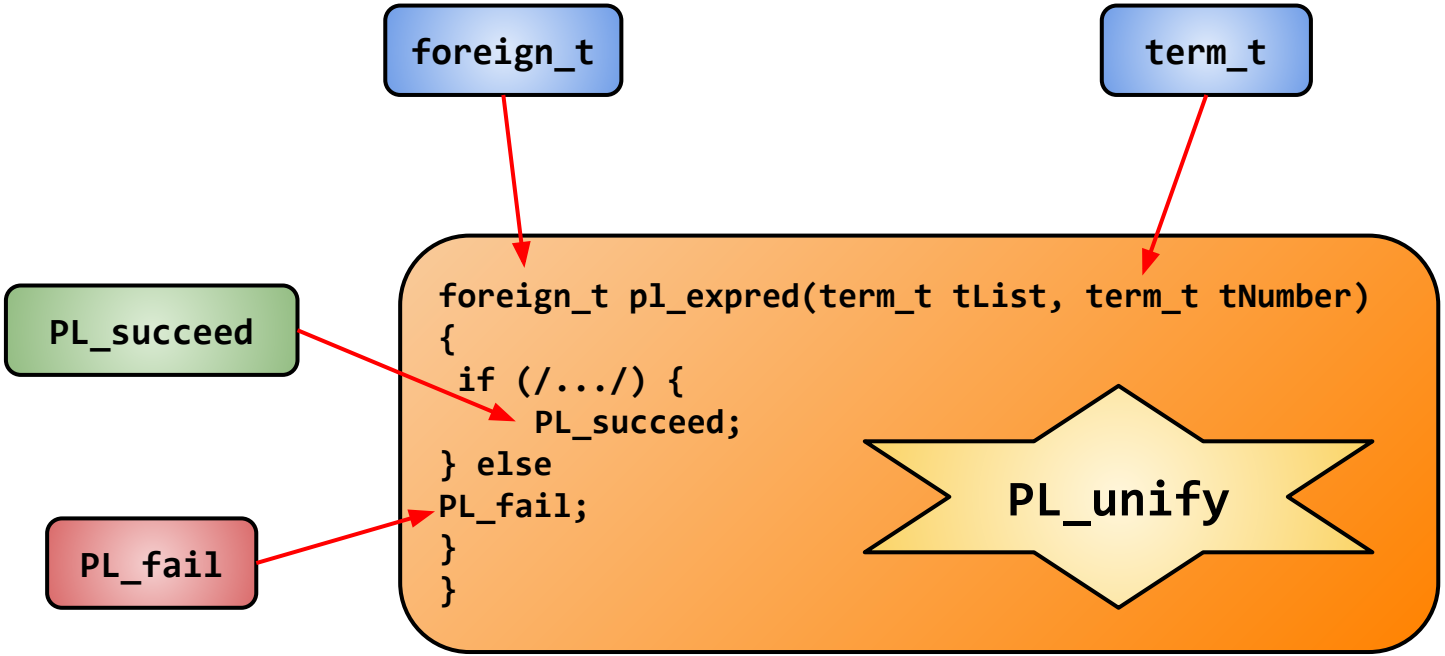
Crea un término

Lista de términos

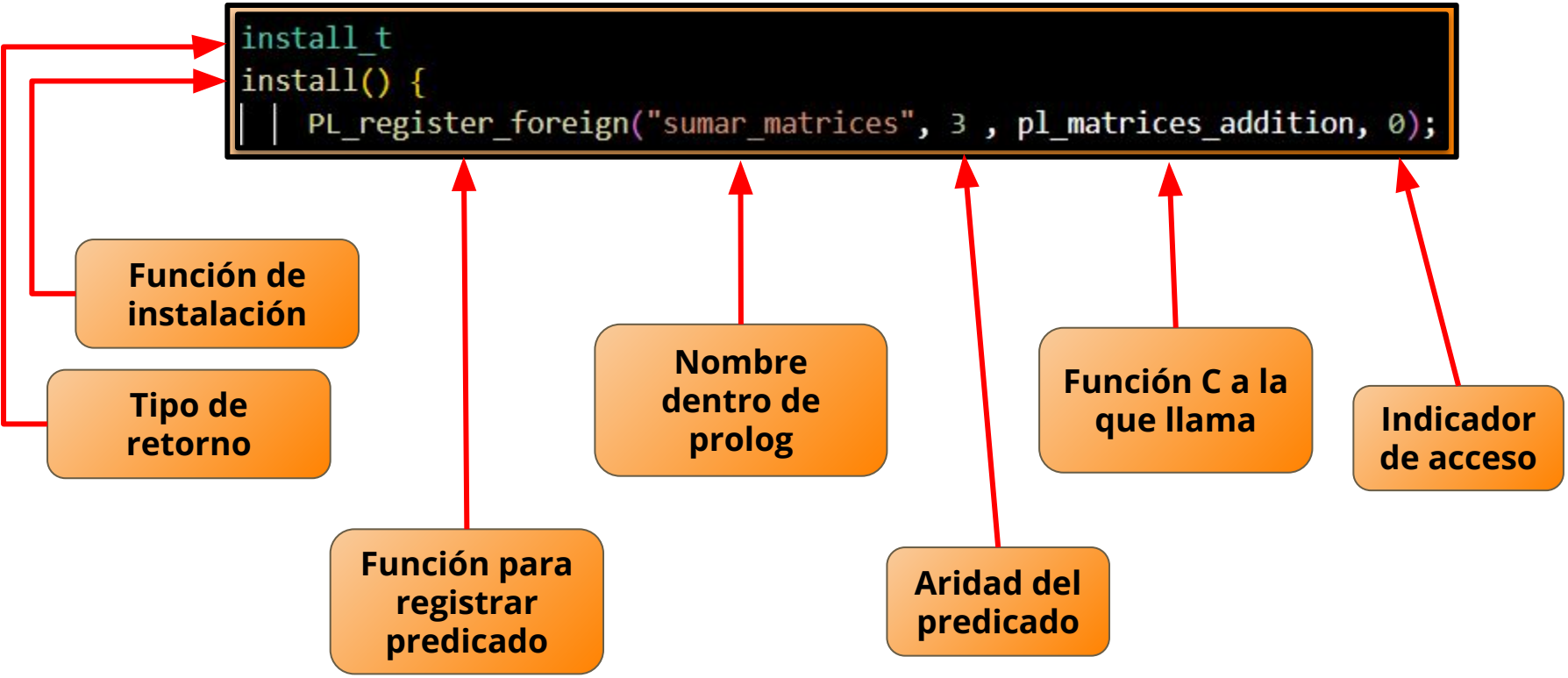
Otros tipos de datos de interfaz



Definición de predicados externos



Instalación de predicados



Análisis y construcción de términos

Análisis

- PL_is_atom (+term)
- PL_is_number (+term)
- PL_is_float (+term)
- PL_is_string (+term)
- PL_is_compound (+term)

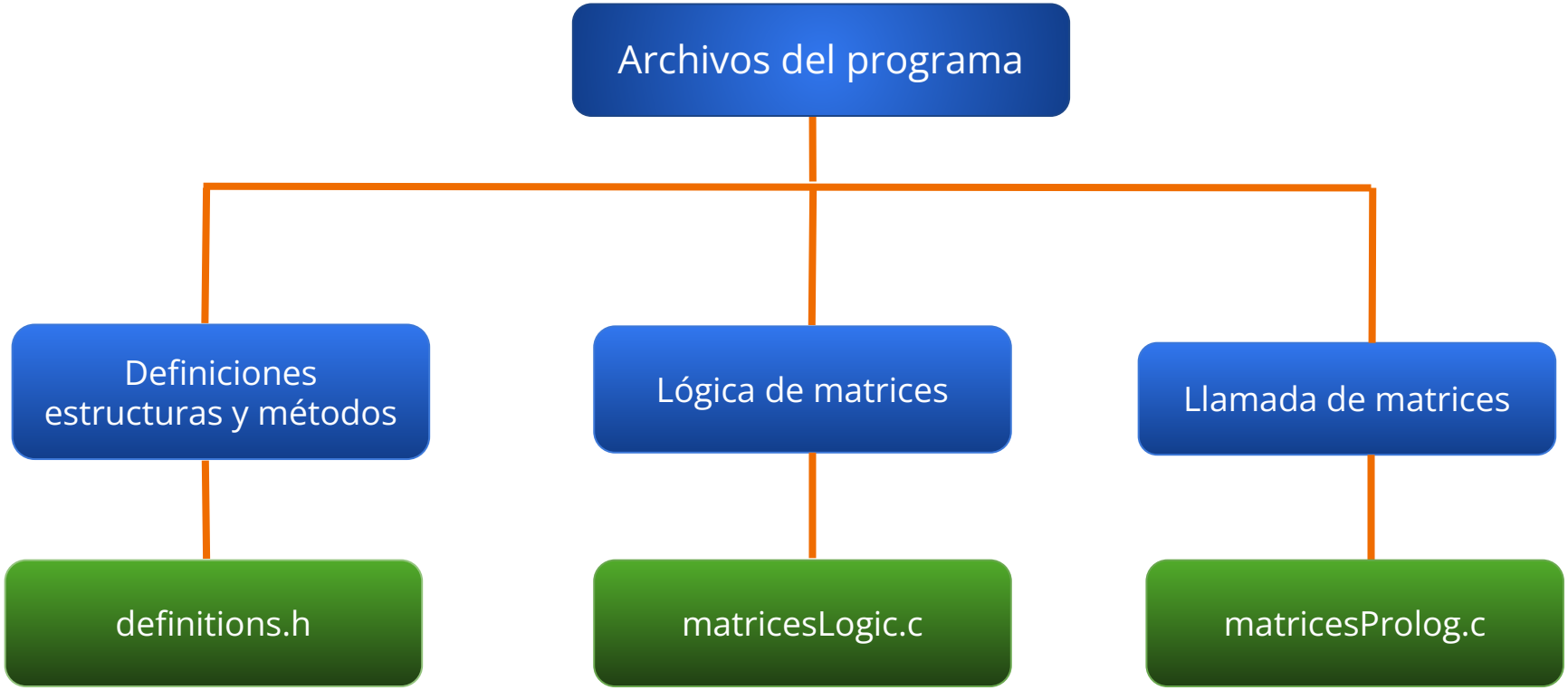
Construcción

- PL_get_float (+term, -double_pointer)
- PL_get_integer (+term, -integer_pointer)
- PL_get_atom_chars (+term, -char_pointer)
- PL_put_<tipo_variable> (-term, +<tipo variable>)

Manipulación de listas

`PL_get_list (+list, -head, -tail)`	`pl_get_list(List, Head, Tail) :- List = [Head | Tail].`
`PL_put_nil (+term)`	`List = [].`
`PL_get_nil (+term)`	`is_empty_list(Term) :- Term = [].`
`PL_cons_list (-list, +head, +tail)`	`cons_list(List, Head, Tail) :- List = [Head | Tail].`

Implementación de los métodos de matrices



definitions.h

```
typedef struct {
    int rows ;
    int columns;
    double* data;
} Matrix;
```

Número de filas

Número de columnas

Puntero unidimensional para acceder a elementos de la matriz

Macro para acceso y modificación de elementos de una matriz

```
# define ACCESS(matrix, row, column) matrix -> data[column * matrix -> rows + row]
```

Métodos destacables matricesLogic.c I

```c
Matrix* parse_list_of_lists_into_matrix(term_t tList) {
    int number_rows = 0;
    int number_columns = 0;

    if (get_correct_dimensions(tList, &number_rows, &number_columns) ==  FAILURE){
        return NULL;
    }
     Matrix* matrix = new_matrix(number_rows, number_columns);
    // Check that the matrix is not null
    if (!matrix) {
        return NULL;
    }
    term_t tTail = PL_copy_term_ref(tList);
    term_t tHead = PL_new_term_ref();
    int current_row = 0;

    while (PL_get_list(tTail, tHead, tTail)) {

        if (!PL_is_list(tHead)) { // Check if the current element is a list
        printf("No se está pasando correctamente una lista de elementos\n");
        return NULL;
            }

        term_t inner_tail = PL_copy_term_ref(tHead);
        term_t inner_head = PL_new_term_ref();

        int integer_value; // The input could be an integer
        double double_value; // The input could be a double
        int current_column = 0;

        while (PL_get_list(inner_tail, inner_head, inner_tail)) {
            if (PL_get_integer(inner_head, &integer_value)) {
                ACCESS(matrix, current_row, current_column) = (double) integer_value;
                current_column++;

            } else if (PL_get_float(inner_head, &double_value)) {
                ACCESS(matrix, current_row, current_column) = double_value;
                current_column++;

            } else {
                printf("Asegúrate que todos los valores que se introduce a la matriz s
                return NULL;
            }
        }
        current_row++;
        }

        return matrix;
    }
```

```c
int parse_matrix_into_list_of_lists(Matrix* matrix, term_t resutltListofLists) {
    if (!matrix || !matrix->data || !matrix->rows || !matrix->columns) {
        return FAILURE;
    }
    term_t tLists = PL_new_term_refs(matrix->rows); // These are consecutive references for all the lists

    for (int current_row = 0; current_row < matrix->rows; current_row++) {
        if (assign_matrix_row_values_to_list(matrix, tLists + current_row, current_row) == FAILURE) {
            return FAILURE;
        }
    }
    term_t allLists = PL_new_term_ref(); // This concatenates all the lists of for the rows into a single
    PL_put_nil(allLists); // Make the the empty list
    for (int i = matrix->rows - 1; i >= 0; i--){
        if (!PL_cons_list(allLists, tLists + i, allLists)) {
            return FAILURE;
        }
    }
    return PL_unify(resutltListofLists, allLists);
}
```

Métodos destacables matricesLogic.c II

```c
int assign_matrix_row_values_to_list(Matrix* matrix, term_t tList, int current_row) {
    if (!matrix) {
        return FAILURE;
    }
    PL_put_nil(tList); // Concatenate the list of values for a row
    term_t current_value = PL_new_term_ref(); // Current double value of a row of the matrix
    for (int current_column = matrix -> columns - 1; current_column >= 0; current_column--) {
        if (!PL_put_float(current_value, ACCESS(matrix, current_row, current_column))) {
            return FAILURE;
        }
        if (!PL_cons_list(tList, current_value, tList)) {
            return FAILURE;
        }
    }
    return SUCCESS;
}
```

```c
int get_correct_dimensions(term_t tList, int* rows, int* columns) {
    int total_rows = 0;
    int total_columns = 0;

    term_t head = PL_new_term_ref();
    term_t tail = PL_copy_term_ref(tList);

    // Check if the outer term is a list
    if (!PL_is_list(tList)) {
        printf("No se trata de una lista\n");
        return FAILURE;
    }
    // Iterate over the list
    while (PL_get_list(tail, head, tail)) {

        if (!PL_is_list(head)) {
            printf(" No se trata de una list \n");
            return FAILURE;
        }

        int current_columns = 0;
        term_t inner_tail = PL_copy_term_ref(head);
        term_t inner_head = PL_new_term_ref();

        // Iterate over the inner list to count the number of columns
        while (PL_get_list(inner_tail, inner_head, inner_tail)) {
            current_columns++;
        }
        // Update for the first columns
        if (total_rows == 0) {
            total_columns = current_columns;
        }
        // If the current row has a different number of columns than previous rows, it's no
        if (current_columns != total_columns) {
            printf("La matriz no tiene el mismo número de columnas en todas las filas\n");
            return FAILURE;
            }

        total_rows++;
    }
    if (rows != NULL && columns != NULL) {
        *rows = total_rows;
        *columns = total_columns;
    } else {
        return FAILURE;
    }
    return SUCCESS;
}
```

Ejemplo de un método en MatricesProlog.c

```
foreign_t pl_matrices_addition(term_t matrix1, term_t matrix2, term_t result){

    Matrix* m1 = parse_list_of_lists_into_matrix(matrix1);
    Matrix* m2 = parse_list_of_lists_into_matrix(matrix2);
    if (!m1 || !m2) {
        PL_fail;
    }
    Matrix* matrix_result = new_matrix(m1->rows, m1->columns);
    if (!matrix_result) {
        PL_fail;
    }

    if (matrices_addition(m1 ,m2 ,matrix_result) == FAILURE) {
        PL_fail;
    }
    term_t matrix_list = PL_new_term_ref();

    if(parse_matrix_into_list_of_lists(matrix_result, matrix_list) == FAILURE) {
        PL_fail;
    }
    return PL_unify(result, matrix_list);
}
```

Ejemplos al ejecutar librería

Comprobación de dimensiones de matrices satisfactoria

```
?- matrices_mismas_dimensions([[1,2,3,4],[4,6,5,7]],[[3,4,6,5],[3,2,41,5]]).
true.
```

Suma matrices satisfactoria

```
?- sumar_matrices([[1,2,3],[2,4,2.42]],[[1,2,23.2],[1.23,2.2,3]],R).
R = [[2.0, 4.0, 26.2], [3.23, 6.2, 5.42]].
```

Multiplicación matrices

```
?- multiplicar_matrices([[1,4,0]],[[2],[-1],[5]],R).
R = [[-2.0]].
```

Obtención de valor máximo de matriz satisfactoria

```
?- obtener_valor_maximo([[1,3,2,5],[2.5,3.4,68.45,4]],M).
M = 68.45.
```

Posibles fallos en los parámetros de las matrices

```
?- sumar_matrices([[4,2,3],[1,5.4,2],[[2,3,2],[3.2,4,2]],R).
ERROR: Syntax error: Illegal start of term
ERROR: sumar_matrices([[4,2,3],[1,5.4,2],[[2,3,2],[3.2,4,2]],
ERROR: ** here **
ERROR: R) .
```

Error por falta de cierre en uno de los corchetes

```
?- sumar_matrices([[1,2,3],[2,3,2.3]],[[1,2],[3,4]],R).
Para realizar la suma, asegúrate que ambas matrices tienen el mismo número de filas que de columnas
false.
```

```
?- sumar_matrices([[2,3,4,a]],[[1,2,3,4]],R).
Asegúrate que todos los valores que se introduce a la matriz son valores numéricos
false.
```

```
?- multiplicar_matrices([[2,3,4,2]],[[1,2,3,4]],R).
Para realizar la multiplicación, asegúrate de que el número de columnas de la primera matriz: 4 sea igual al número de filas de la segunda: 1
false.
```

Conclusión