UNIVERSIDAD AUTONOMA DE TAMAULIPAS

Ingeniería en Sistemas Computacionales

9° G

Programación de Sistemas de Base 2

(3:00PM – 4:00PM)

Documentación Técnica

Gutiérrez Garcés Edwin

Garcés Hernández Jesús Eduardo

Moreno Reyes Luis Daniel

Ruiz Yáñez Aaron Alfonso

Guerrero Martínez Jesús

Cd. Madero, Tamaulipas a 2025

Contenido

1. Descripción del lenguaje

El lenguaje diseñado en este proyecto es un **lenguaje específico de dominio (DSL)** orientado al control de robots. Su sintaxis está inspirada en lenguajes imperativos como C o Java, pero centrado en instrucciones de movimiento, sensores, control condicional y temporización.

1.1 Estructura general del programa

Un programa válido está formado por una o más instrucciones, ya sean declaraciones, asignaciones, estructuras de control o llamadas a funciones.

```
<programa> ::= <instruccion_list>
<instruccion_list> ::= <instruccion> | <instruccion_list> <instrucción>
```

1.2 Declaraciones de variables

El lenguaje permite declarar variables de tipo int, float o boolean, seguidas del identificador y punto y coma.

```
<declaracion> ::= "int" IDENTIFIER ";"
                | "float" IDENTIFIER ";"
                | "boolean" IDENTIFIER ";"
```

Ejemplos válidos:

```
int x;
float velocidad;
boolean listo;
```

1.3 Asignaciones

Una variable puede recibir una expresión si fue previamente declarada y si el tipo de dato es compatible.

```
<asignacion> ::= IDENTIFIER "=" <expresion> ";"
```

Ejemplos válidos:

```
x = 10;
velocidad = 3.5;
listo = TRUE;
```

Errores detectables:

```
x = "hola";      // Error: string no es int
z = 5;           // Error: variable no declarada
```

1.4 Expresiones

El lenguaje soporta expresiones numéricas, booleanas, lógicas y comparativas. Las operaciones son evaluadas y tipificadas semánticamente.

```
<expresion> ::= IDENTIFIER
              | NUMBER
              | STRING
              | "TRUE" | "FALSE"
              | <expresion> OP <expresion>
```

Operadores válidos:

- Aritméticos: +, -, *, /
- Relacionales: ==, !=, >, <
- Lógicos: AND, OR

Ejemplo:

```
x = (a + b) * 2;
if (x > 10 AND listo) THEN { ... }
```

1.5 Condicionales (IF)

Permiten ejecutar instrucciones solo si la condición evaluada es verdadera. La condición debe ser de tipo boolean.

```
<condicional> ::= "IF" "(" <expresion> ")" "THEN" "{" <instruccion_list>
"}"
```

Ejemplo válido:

```
IF (distancia < 5) THEN {
    PRINT("Demasiado cerca");
    MOVE_BACKWARD(3);
}
```

Errores comunes:

- Condición no booleana
- Llaves mal cerradas

1.6 Ciclos (WHILE)

Permiten ejecutar instrucciones de forma repetitiva mientras una condición se mantenga verdadera.

```
<ciclo> ::= "WHILE" "(" <expresion> ")" "THEN" "{" <instruccion_list> "}"
```

Ejemplo válido:

```
WHILE (sensor_activado) THEN {
    WAIT(1);
    SCAN(1);
}
```

1.7 Funciones

Las funciones representan instrucciones que ejecuta el robot, clasificadas en:

Funciones sin argumento:

```
<funcion> ::= FUNCION ";"
```

Ejemplos:

```
SHUTDOWN;
ACTIVATE_SENSOR;
TOGGLE_LIGHT;
```

Funciones con argumento:

```
<funcion> ::= FUNCION "(" <expresion> ")" ";"
```

Ejemplos:

```
MOVE_FORWARD(5);
WAIT(2);
PRINT("Listo");
```

El parser valida si la función existe, si requiere o no argumentos y si el argumento tiene el tipo correcto. El analizador semántico también agrega una anotación funcional como:

```
PRINT: Imprime un mensaje en pantalla
```

1.8 Comentarios

El lenguaje permite incluir comentarios con //, los cuales no afectan la ejecución pero son extraídos por el sistema para generar anotaciones sugeridas con NLP.

Ejemplo:

```
// Iniciar secuencia de movimiento
MOVE_FORWARD(5);
```

Resultado procesado:

```
{
  "comentario": "Iniciar secuencia de movimiento",
  "sugerencia": "Sugerencia simulada: revisa la intención de 'Iniciar
secuencia de movimiento'."    }
```

2. Decisiones de Diseño

2.1 Elección del Lenguaje y Herramientas

El proyecto fue implementado completamente en **Python**, una decisión tomada por varias razones técnicas y estratégicas:

- Permite una rápida integración con bibliotecas modernas para análisis léxico y sintáctico sin depender de herramientas externas como JFlex o ANTLR.
- Posee una amplia compatibilidad con Hugging Face, especialmente a través de la biblioteca transformers y la Inference API.
- Es un lenguaje ampliamente utilizado en compiladores educativos, prototipado y en proyectos de inteligencia artificial.

La implementación léxica y sintáctica se realizó con analizadores hechos a mano en lugar de generadores automáticos (como PLY o ANTLR), para tener un mayor control sobre la estructura del DSL, lo cual era fundamental para poder anotar y transformar instrucciones robóticas en descripciones funcionales.

2.2 Elección del modelo de NLP: CodeBERT sobre GPT-2 y Falcon

Se consideraron varios modelos de lenguaje para generar sugerencias de error o interpretar comentarios:

Modelo	Evaluación
GPT-2	Buena generación libre de texto, pero sin foco en código estructurado.
Falcon	Más potente en tamaño, pero requiere recursos considerables y mayor tiempo de respuesta en Hugging Face.
CodeBERT	Entrenado específicamente en lenguaje de programación y descripciones en lenguaje natural. Ofrece equilibrio entre contexto de código y generación textual.

Decisión:

Se optó por utilizar **CodeBERT (huggingface/CodeBERTa-language-id)** como modelo base para sugerencias de errores porque fue entrenado con grandes volúmenes de código fuente (Java, Python, JavaScript, etc.) y pares de lenguaje natural, lo cual lo hace ideal para tareas como:

- Interpretar mensajes de error y sugerir soluciones semánticas
- Transformar comentarios en pseudocódigo o descripciones formales

Además, CodeBERT está disponible en Hugging Face con compatibilidad total con Inference API, lo que facilitó su integración estructurada, incluso cuando el modelo no respondió directamente en el entorno gratuito.

2.3 Uso de Inference API vs transformers locales

Inicialmente, se intentó utilizar transformers localmente para ejecutar el modelo directamente en el entorno del Space. Sin embargo, esta aproximación fue descartada por las siguientes razones:

- **Limitaciones de hardware** en Spaces gratuitos: modelos como Falcon y CodeBERT requieren GPU o al menos un entorno sin restricciones de RAM.
- **Problemas de latencia** al intentar cargar modelos pesados en tiempo de ejecución.
- **Complejidad operativa**: se necesitaría crear un contenedor personalizado, lo cual excedía el alcance del proyecto semestral.

Por estas razones, se adoptó el uso de la **Hugging Face Inference API**, que permite enviar texto al modelo y recibir una sugerencia procesada. En caso de que la API no esté disponible, se tenía implementada una alternativa de sugerencias simuladas para garantizar la continuidad del análisis, pero se optó por no implementar esta opción, ya que no representaría una sugerencia genuina del modelo y de requería modificar el código una vez que se actualizara el tipo de plan del Space de Huggin Face.

2.4 Optimización del Código Intermedio con Anotaciones NLP

Aunque la optimización algorítmica no era parte del objetivo principal, el código intermedio fue diseñado para ser fácilmente interpretable y complementado con anotaciones generadas por el sistema NLP. Por ejemplo:

```
FUNCION: MOVE_FORWARD(5)
 -Código Intermedio:
    PARAM 5
    CALL MOVE_FORWARD
    STORE t0
 -Anotación:
    El robot se moverá hacia adelante durante 5 unidades
```

Este diseño permite, en versiones futuras, utilizar las sugerencias NLP no solo como ayuda visual sino como metainformación para planificar rutas, validar comportamiento esperado o verificar consistencia lógica en simulaciones de robots.

3. Consideraciones finales y justificación técnica

Durante el desarrollo del proyecto se tomaron decisiones técnicas estratégicas que permitieron cumplir los objetivos centrales propuestos en la especificación, aunque con algunos ajustes razonables debido a las limitaciones del entorno. Esto implicó una evolución importante en su enfoque y herramientas.

En su etapa inicial, el compilador fue diseñado en Java utilizando JFlex para el análisis léxico y CUP para el análisis sintáctico. Esta arquitectura funcionaba correctamente a nivel local, pero representaba una limitación crítica al momento de integrar capacidades modernas de procesamiento de lenguaje natural (NLP), especialmente en entornos desplegables en la nube.

Con base en los requisitos de la asignación particularmente la integración con modelos de Hugging Face, se tomó la decisión de migrar completamente el proyecto a Python, lo que

permitió aprovechar bibliotecas como re para expresiones regulares, estructuras flexibles para el parser y compatibilidad directa con la Inference API de Hugging Face.

Este cambio fue justificado por las siguientes razones técnicas:

- Hugging Face Spaces no ofrece soporte para ejecutar Java directamente ni facilitar conexiones con modelos NLP desde esa plataforma.
- Python permite una integración sencilla con requests, transformers, y Gradio para construir una interfaz de entrada directa.
- El tiempo de desarrollo y prueba se redujo considerablemente sin sacrificar funcionalidad.

Uno de los principales ajustes fue la integración parcial de modelos NLP. Aunque el sistema fue diseñado para interactuar directamente con modelos como CodeBERT o GPT-2, el entorno gratuito de Hugging Face Spaces no permite ejecutar inferencias en tiempo real con modelos grandes. Por ello, se implementó un esquema híbrido: se mantuvo la arquitectura preparada para respuestas reales, pero se integraron sugerencias simuladas para garantizar la continuidad y funcionalidad del flujo completo de análisis.

En cuanto al diseño del lenguaje personalizado, se priorizó la claridad y la coherencia en la definición de tokens, estructuras de control (IF, WHILE) y llamadas a funciones con o sin argumentos. La gramática fue formalizada en BNF y el analizador semántico valida tipos, condiciones, uso correcto de variables y funciones, generando además anotaciones funcionales legibles por humanos como parte del análisis semántico.

Por último, el proyecto logró reflejar el espíritu de la especificación: no solo construir un compilador, sino también transformar la experiencia del usuario mediante descripciones interpretativas, sugerencias inteligentes y anotaciones funcionales. Este enfoque convirtió un

análisis técnico tradicional en una herramienta interactiva, educativa y extensible, alineada con el futuro de los compiladores asistidos por IA..