Universidad Católica Argentina
Facultad de Ingeniería y Ciencias Agrarias

TRABAJO FINAL INTEGRADOR

Materia: Algorítmica y Lógica Computacional

Autor:
Juan Cruz Di Meglio

Docente:
Antonio Quintero Rincón

14 de febrero de 2026

Resumen

En este trabajo presento el desarrollo de **CarBot**, un sistema experto de arquitectura híbrida de mi autoría, diseñado para optimizar la toma de decisiones en el sector automotriz. Integré un motor de inferencia en **Programación Lógica (Prolog)** con una interfaz de control en **Python**. El enfoque principal fue la aplicación práctica de los modelos teóricos de la computación: modelé la interacción del usuario mediante un **Autómata Finito Determinista (AFD)** con capacidad de retroceso; implementé una **Máquina de Turing** para la validación de contactos; utilicé **Gramáticas de Cláusulas Definidas (DCG)** aplicando restricciones semánticas para patentes; y abordé la logística (TSP) mediante el análisis de **Complejidad Computacional**, implementando un selector adaptativo entre un algoritmo exacto ($O(N!)$) y uno heurístico ($O(N^2)$) para garantizar la trazabilidad del sistema.

Palabras Clave: Sistemas Expertos, Prolog, Autómatas Finitos, Máquina de Turing, DCG, TSP, NP-Completo, Complejidad Asintótica.

Índice

1. Introducción

1.1. Contexto y Problemática

En los sistemas de gestión tradicionales (ERPs) de concesionarias automotrices identifiqué una problemática recurrente: son sistemas rígidos basados en bases de datos relacionales que carecen de capacidad deductiva. No pueden razonar"sobre los datos, validar entradas complejas de forma declarativa, ni resolver problemas de optimización logística sin recurrir a extensos y poco eficientes códigos imperativos.

1.2. Solución Propuesta y Objetivos

Para solucionar esto, diseñé y desarrollé un **Sistema Experto Híbrido**. Mi objetivo fue desacoplar la lógica de negocio del control de flujo utilizando la herramienta más adecuada para cada paradigma:

- **Capa de Conocimiento (Prolog):** Donde programé las reglas lógicas, la inferencia mediante cláusulas de Horn y los algoritmos formales (Turing/-Gramáticas).
- **Capa de Interfaz (Python):** Donde gestioné la interacción I/O, la limpieza del dataset (`Autosdef.csv` vía Pandas) y la comunicación FFI (Foreign Function Interface) usando la librería `PySwip`.

De esta forma, busqué demostrar la aplicación directa y empírica de los fundamentos teóricos abordados durante la cursada en un entorno de software funcional.

2. Aplicación de Modelos Teóricos e Implementación

En lugar de depender de sentencias condicionales clásicas (`if/else`) o expresiones regulares, opté por fundamentar la lógica de *CarBot* directamente en la teoría de autómatas, lenguajes formales y complejidad.

2.1. Control de Flujo: Autómata Finito Determinista (AFD)

Diseñé el control de diálogo del sistema experto como un AFD. Esto me permitió dotar al sistema de persistencia de estado y rutas no lineales. La característica más importante que implementé fue un **bucle de retroalimentación** en el estado de recomendación. Si el motor lógico no satisface la necesidad del usuario, el autómata transiciona hacia atrás, evitando bucles muertos o reinicios abruptos.

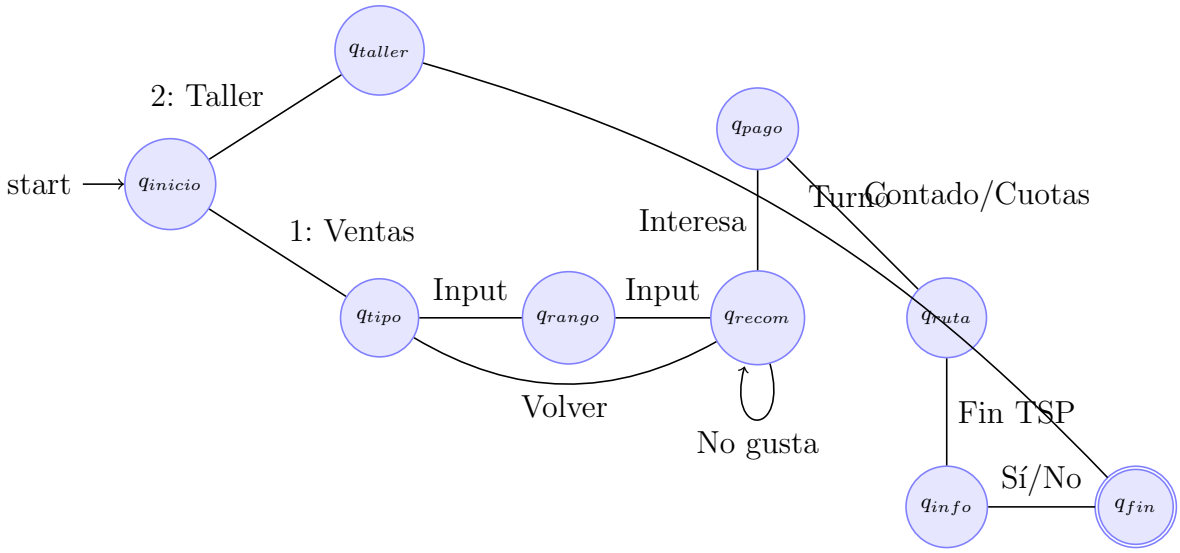


Figura 1: Diseño propio del AFD. Implementado en Prolog mediante aserciones dinámicas `transicion/3`.

2.2. Validación Dinámica: Máquina de Turing

Para la validación de números telefónicos, necesitaba un algoritmo capaz de adaptar su comportamiento según la entrada. Implementé una Máquina de Turing en Prolog que simula un no-determinismo aparente: lee el primer carácter de la cinta y bifurca su estado interno (q_{amba} vs $q_{interior}$). Esto dicta si el conteo recursivo posterior debe exigir 8 o 7 dígitos restantes, algo inviable en un autómata sin memoria auxiliar.

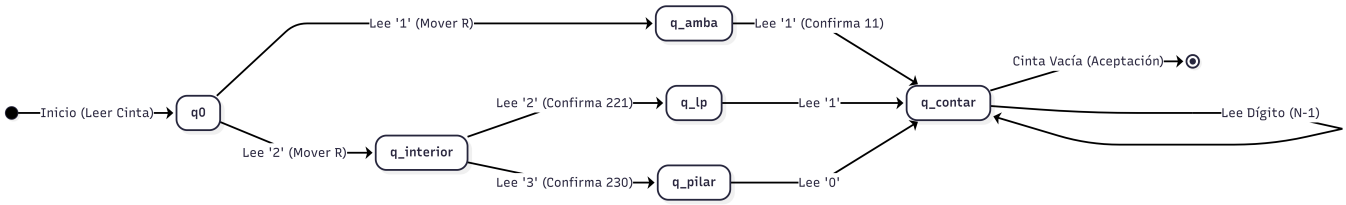


Figura 2: Diagrama de transición de la Máquina de Turing para validación de celulares. Diseñé este modelo para bifurcar la lógica según el código de área detectado.

Figura 3: Modelo de la MT implementada para reconocer lenguajes dependientes del prefijo.

2.3. Parsing y Semántica: Gramáticas DCG

Para analizar la sintaxis de las patentes utilicé Gramáticas de Cláusulas Definidas (DCG). Para dotar al sistema de rigor profesional, no me limité a una validación estructural, sino que implementé una **restricción semántica**. Creé reglas de producción específicas (`letra_mercosur_inicio`) que rechazan automáticamente caracteres que aún no han sido emitidos en la realidad argentina (e.g., patentes que

inician con 'Z'), demostrando la potencia deductiva de Prolog frente a expresiones regulares clásicas.

2.4. Logística, TSP y Complejidad Asintótica

Durante la programación del módulo de *Test Drives*, abordé el Problema del Viajante de Comercio (TSP) buscando un **Ciclo Hamiltoniano** perfecto.

Al probar mi primer algoritmo basado en permutaciones (`ruta_optima`), experimenté empíricamente la naturaleza **NP-Hard** del problema: al solicitar rutas de más de 8 destinos, el sistema entraba en inanición. A través del análisis de complejidad, determiné que el orden de crecimiento era $O(N!)$, exigiendo millones de operaciones para instancias pequeñas.

Para solucionarlo, diseñé un **Selector Algorítmico Adaptativo**:

- **Cota inferior ($N \leq 3$):** Utilizo el algoritmo de Fuerza Bruta, garantizando el óptimo global ya que el costo computacional ($3! = 6$) es imperceptible.
- **Cota superior ($N > 3$):** El sistema conmuta automáticamente a una heurística Voraz (Greedy), reduciendo la complejidad a un tiempo polinomial $O(N^2)$.

Nodos (N)	**Op. Exactas ($O(N!)$)**	**Op. Greedy ($O(N^2)$)**	**Decisión del Sistema**
3	6	9	Fuerza Bruta (Óptimo)
5	120	25	Heurística (Greedy)
10	3,628,800	100	Heurística (Greedy)

Tabla 1: Justificación empírica de mi estrategia de selección algorítmica.

3. Conclusiones

El desarrollo individual de **CarBot** me permitió confirmar que la integración de lenguajes lógicos e imperativos resulta vital para resolver problemas complejos de ingeniería de software. Al dejar de lado sentencias imperativas rígidas y estructurar mi código sobre Autómatas, Máquinas de Turing y Gramáticas, logré un sistema altamente robusto ante fallos de usuario. Asimismo, la experiencia directa con la intratabilidad del TSP me demostró la importancia crítica de aplicar el análisis de Complejidad Asintótica antes de desplegar algoritmos en producción, justificando matemáticamente el uso de heurísticas adaptativas.

4. Anexos

4.1. Anexo A: Manual de Instalación y Requisitos

Para ejecutar el sistema desarrollado, el entorno requiere:

- **Motor Lógico:** `SWI-Prolog` instalado y agregado a las variables de entorno (PATH).
- **Entorno Python:** Versión 3.11.9 (utilizada en desarrollo) sobre Visual Studio Code con soporte Jupyter.
- **Dependencias:** Mediante `pip install pyswip pandas networkx matplotlib`.

Nota Técnica: Debido a la manipulación de buffers de I/O entre Jupyter y Prolog, la consola puede requerir presionar [ENTER] en la barra de input para forzar la actualización del estado si ocurre una pausa en el renderizado del diálogo.

4.2. Anexo B: Evidencia de Ejecución

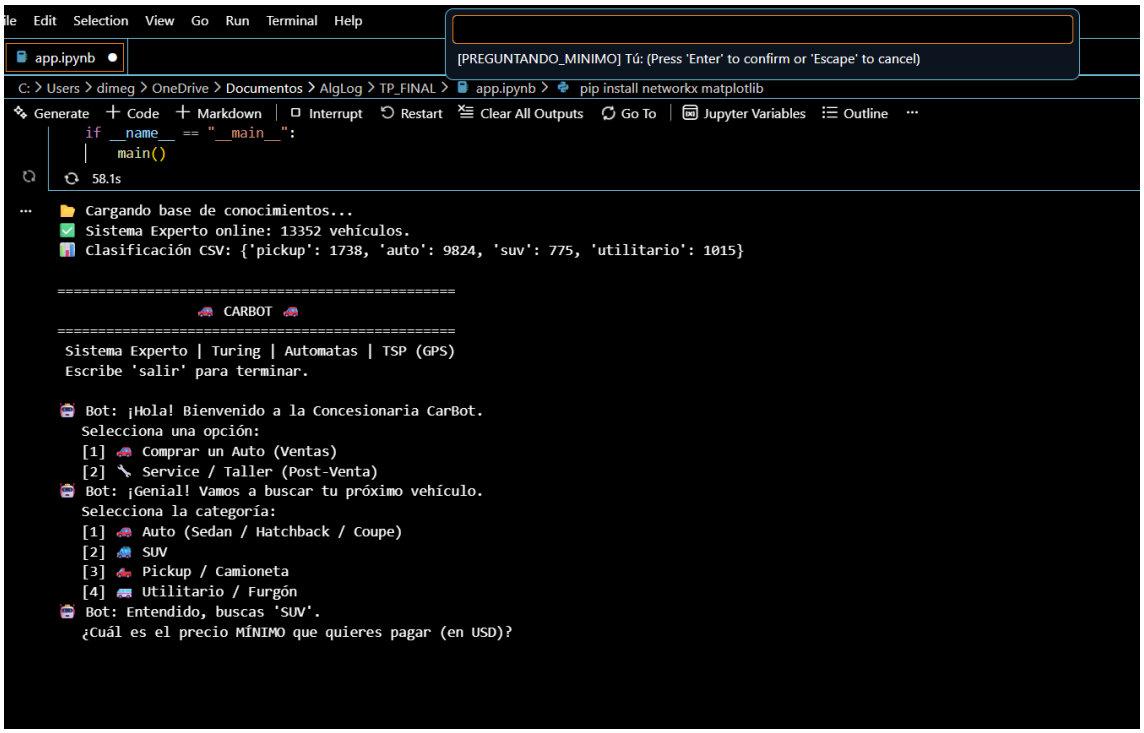
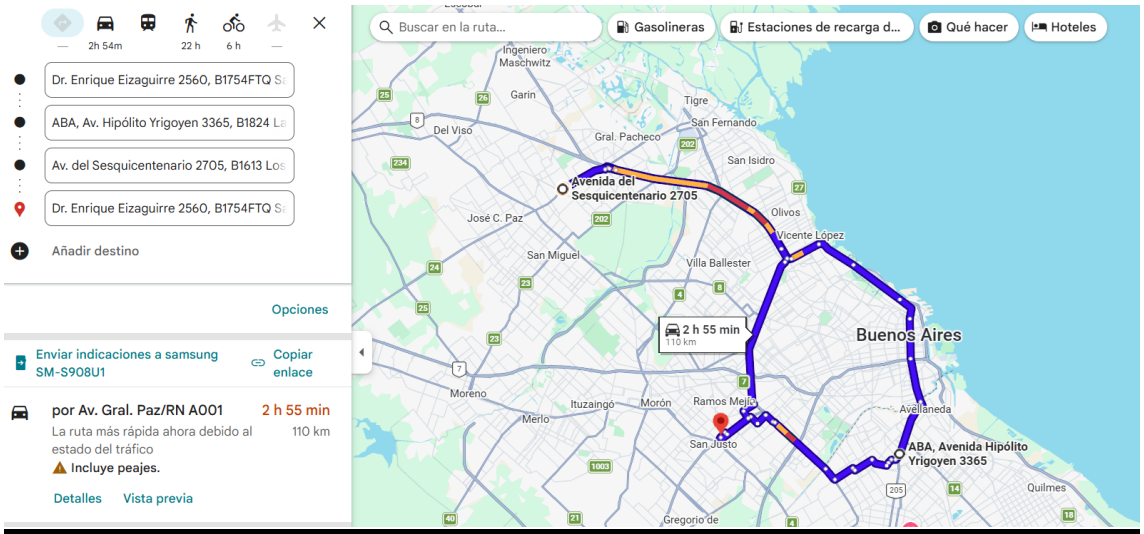


Figura 4: Consola interactiva ejecutando el motor de inferencia híbrido.




```
Bot: Calculando logística...

RUTA SUGERIDA (ruta_optima):
   Del Oeste (San Justo)
   Del Sur Autos (Lanús)
   Autodelta
   Del Oeste (San Justo)
Distancia Real Total: 96 km

Generando enlace de navegación GPS...
Link generado: https://www.google.com/maps/dir/Dr.%20Enrique%20Eizaguirre%202560
Abriendo Google Maps en tu navegador...
```

Figura 5: Ciclo Hamiltoniano de la ruta óptima generado exitosamente.

5. Bibliografía

Referencias

[1] Merritt, D. (2000). *Building Expert Systems in Prolog*. Springer-Verlag.

[2] Cormen, T. H., et al. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.