Programación Declarativa
Seminario 2
Sistema Experto

Daniel Toledo Daniel Machado Osvaldo Moreno José Antonio Concepción
C-311

23 de febrero de 2024

Índice

1. Definición del sistema y sus características.

Un **sistema experto** es un programa informático diseñado para resolver problemas complejos en un dominio específico, utilizando la experiencia y el conocimiento de un experto humano. Estos sistemas simulan el proceso de razonamiento de un experto en un campo particular, emulando su capacidad para tomar decisiones y ofrecer soluciones a problemas específicos.

1.1. Características principales.

- **Base de conocimiento:** Almacena información y hechos relevantes al dominio del problema, incluyendo reglas, casos de estudio y heurísticas.
- **Motor de inferencia:** Aplica las reglas y el conocimiento de la base de datos para analizar nueva información y generar nuevas reglas, soluciones o recomendaciones.
- **Interfaz de usuario:** Permite la interacción entre el usuario y el sistema, facilitando la introducción de datos, la consulta de información y la obtención de resultados.
- **Explicaciones:** El sistema puede explicar las razones detrás de sus decisiones y recomendaciones, mejorando la transparencia y la confianza del usuario.
- **Capacidad de aprendizaje:** Algunos sistemas expertos pueden aprender de la experiencia y mejorar su desempeño con el tiempo, incorporando nuevos conocimientos y adaptándose a nuevas situaciones.

1.2. Beneficios.

- **Mejora la toma de decisiones:** Ofrecen soluciones más precisas y consistentes que las basadas en la intuición o la experiencia individual.
- **Ahorro de tiempo y recursos:** Automatizan tareas repetitivas y permiten a los expertos enfocarse en tareas más complejas.
- **Acceso a la experiencia:** Permiten que personas sin experiencia en un dominio específico puedan acceder a conocimiento experto.
- **Mejora la comunicación:** Facilitan la comunicación entre expertos y no expertos.

1.3. Limitaciones:

- **Costo de desarrollo:** El desarrollo de un sistema experto puede ser costoso y complejo. Dificultad de adquisición de conocimiento: Extraer el conocimiento de los expertos y codificarlo en el sistema puede ser un proceso desafiante.
- **Falta de flexibilidad:** Los sistemas expertos pueden ser rígidos y no adaptarse bien a cambios en el entorno o en el problema a resolver.
- **Falta de creatividad:** No son capaces de generar ideas o soluciones nuevas e innovadoras.

2. Historia.

2.1. Década de los 50-60.

Después del surgimiento de las computadoras modernas a fines de la década de 1940 y principios de la década de 1950, los investigadores empezaron a reconocer el vasto potencial que estas máquinas tenían para la sociedad contemporánea. Un desafío inicial consistió en dotar a estas máquinas de la capacidad de "pensar"de manera similar a los seres humanos, específicamente, otorgándoles la habilidad de tomar decisiones importantes de forma autónoma. El ámbito de la medicina y la atención médica presentó un desafío intrigante al permitir que estas máquinas realizaran diagnósticos médicos.

Así, hacia finales de la década de 1950, justo en la era plena de la información, los investigadores comenzaron a explorar la viabilidad de emplear la tecnología informática para emular la toma de decisiones humanas. Por ejemplo, los investigadores biomédicos iniciaron el desarrollo de sistemas informáticos para aplicaciones diagnósticas en medicina y biología. Estos sistemas tempranos utilizaban datos como los síntomas de los pacientes y los resultados de pruebas de laboratorio como insumos para generar diagnósticos. Estos sistemas fueron considerados como precursores de los sistemas expertos. Sin embargo, se notaron limitaciones significativas al emplear métodos tradicionales como diagramas de flujo, coincidencia de patrones estadísticos o teoría de la probabilidad.

Este escenario condujo gradualmente al desarrollo de sistemas expertos que se basaban en enfoques fundamentados en el conocimiento. Ejemplos notables de sistemas expertos en medicina incluyen MYCIN, Internist-I y, posteriormente, en la década de 1980, CADUCEUS.

Los sistemas expertos fueron formalmente introducidos alrededor de 1965 por el Proyecto de Programación Heurística de Stanford, liderado por Edward Feigenbaum, quien a menudo es denominado el "padre de los sistemas expertos". Otros colaboradores clave fueron Bruce Buchanan y Randall Davis. Este equipo de investigadores de Stanford se propuso identificar áreas donde la experiencia humana era altamente valorada y compleja, como el diagnóstico de enfermedades infecciosas (MYCIN) y la identificación de moléculas orgánicas desconocidas (Dendral). La noción de que "los sistemas inteligentes derivan su poder del conocimiento que poseen, más que de los formalismos y esquemas de inferencia específicos que utilizan", según Feigenbaum, marcó un avance significativo, ya que la investigación previa se había centrado en métodos computacionales heurísticos y en el desarrollo de solucionadores de problemas de propósito general.

La investigación sobre sistemas expertos también fue activa en Europa, con enfoques diferentes a los de Estados Unidos. Mientras que en Estados Unidos se enfocaron en el uso de sistemas de reglas de producción, principalmente en sistemas codificados en el entorno de programación LISP y en shells de sistemas expertos desarrollados por compañías como Intellicorp, en Europa se centraron más en sistemas y shells de sistemas expertos basados en Prolog, que empleaban una forma de programación basada en reglas fundamentada en la lógica formal.

Uno de los primeros shells de sistemas expertos basados en Prolog fue APES. Un caso de uso temprano de Prolog y APES se dio en el ámbito jurídico, específicamente en la codificación de una parte importante de la Ley de Nacionalidad Británica. Este trabajo demostró la eficacia del uso de técnicas y tecnologías de inteligencia artificial para formalizar la ley estatutaria en una representación lógica basada en computación.

2.2. Década de los 80

En la década de 1980, los sistemas expertos se volvieron más comunes. Las universidades comenzaron a ofrecer cursos sobre sistemas expertos, y dos tercios de las empresas incluidas en la lista Fortune 500 aplicaban esta tecnología en sus operaciones diarias. El interés en los sistemas expertos era global, con el proyecto de Quinta Generación de Computadoras en Japón y un aumento en la financiación de la investigación en Europa.

En 1981, con la presentación del primer IBM PC y el sistema operativo PC DOS, se produjo un cambio significativo en el equilibrio del panorama informático. Los chips utilizados en los PCs, aunque relativamente potentes, eran mucho más económicos que la costosa capacidad de procesamiento ofrecida por los mainframes, que en ese momento dominaban el ámbito de la informática empresarial. Esta disparidad condujo a la emergencia de un nuevo paradigma arquitectónico para la informática empresarial: el modelo cliente-servidor.

La llegada de los PCs permitió realizar cálculos y razonamientos a una fracción del costo de un mainframe. Además, este modelo facilitaba a las unidades de negocio eludir la dependencia de los departamentos de TI de la empresa y crear sus propias aplicaciones directamente. Este cambio tuvo un impacto significativo en el mercado de los sistemas expertos.

Los sistemas expertos ya eran poco comunes en el mundo empresarial, ya que requerían habilidades especializadas que muchos departamentos de TI no poseían o no estaban dispuestos a desarrollar. Sin embargo, resultaban ideales para las nuevas plataformas basadas en PC que prometían empoderar a los usuarios finales y a los expertos en la creación de aplicaciones. Hasta ese momento, el entorno principal de desarrollo para los sistemas expertos había sido las costosas máquinas Lisp de empresas como Xerox, Symbolics y Texas Instruments.

Con la popularización de los PCs y la adopción generalizada de la informática cliente-servidor, empresas como Intellicorp e Inference Corporation reorientaron sus esfuerzos hacia el desarrollo de herramientas basadas en PC. Además, surgieron numerosos nuevos proveedores, frecuentemente respaldados por capital de riesgo, como Aion Corporation, Neuron Data, Exsys, entre otros.

El primer sistema experto utilizado en el diseño de un producto a gran escala fue el programa de software SID (Synthesis of Integral Design), desarrollado en 1982. Escrito en LISP, SID generó el 93 de las puertas lógicas de la CPU VAX 9000. Las reglas para este programa fueron creadas por varios diseñadores lógicos expertos. SID amplió estas reglas y generó rutinas de síntesis lógica de software que eran significativamente más extensas que las propias reglas. A pesar de ciertas controversias, el programa fue utilizado debido a restricciones presupuestarias, y fue desmantelado tras la finalización del proyecto VAX 9000.

3. Aplicación en la actualidad.

En la actualidad, los sistemas expertos tienen múltiples aplicaciones en diferentes campos. Uno de los principales usos de estos sistemas es en el área de la salud, donde se utilizan como sistemas de diagnóstico de enfermedades. También se utilizan en la industria y los negocios empresariales para realizar tareas de análisis de préstamos, optimización de almacenes logísticos, toma de decisiones financieras, gestión de datos, evaluación, control de procesos, entre otros. Los sistemas expertos son programas informáticos que tienen el objetivo de solucionar un problema concreto y utilizan la inteligencia artificial y técnicas de razonamiento lógico para simular el conocimiento y las habilidades analíticas de un experto humano en un campo o área de conocimiento en concreto. Estos sistemas pueden ser utilizados por no-expertos para mejorar sus habilidades en la resolución de problemas y también pueden ser utilizados como asistentes por expertos.

Unos ejemplos de sistema experto son:

- **MYCIN**: Un sistema experto para el diagnóstico de enfermedades infecciosas.
- **DENDRAL**: Un sistema experto para la identificación de compuestos químicos.
- **PUFF**: Un sistema experto para la interpretación de pruebas de función pulmonar.
- **XCON**: Un sistema experto para la configuración de computadoras.
- **CLIPS**: Un lenguaje de programación para el desarrollo de sistemas expertos.

4. Ejemplo de un caso de aplicación y una solución usando Prolog.

Proponemos un juego interactivo y original que hace uso de un sistema experto de una manera sencilla. En este juego, el sistema experto interroga al usuario con el objetivo de adivinar el personaje de la saga de Harry Potter en el que el usuario está pensando. Sin embargo, el usuario solo puede responder con un 'sí.' o un 'no.' a las preguntas formuladas por el sistema experto. Es crucial que el usuario responda de manera precisa y honesta, ya que cualquier respuesta incorrecta podría impedir que el sistema experto adivine correctamente el personaje.

```
% Personaje identification rules
% To run, type      go.
go :- hypothesize(Personaje),
      write('El personaje es: '),
      write(Personaje), nl, undo.

 /* hypotheses to be tested */
hypothesize(harry_potter) :- harry_potter, !.
hypothesize(ron_weasley) :- ron_weasley, !.
hypothesize(ginny_weasley) :- ginny_weasley, !.
hypothesize(hermione_granger) :- hermione_granger, !.
hypothesize(dumbledore) :- dumbledore, !.
hypothesize(sirius_black) :- sirius_black, !.
hypothesize(remus_lupin) :- remus_lupin, !.
hypothesize(snape) :- snape, !.
hypothesize(voldemort) :- voldemort, !.
hypothesize(bellatrix_lestrange) :- bellatrix_lestrange, !.
hypothesize(molly_weasley) :- molly_weasley, !.
hypothesize(draco_malfoy) :- draco_malfoy, !.
hypothesize(lucius_malfoy) :- lucius_malfoy, !.
hypothesize(dobby) :- dobby, !.
hypothesize(hagrid) :- hagrid, !.
hypothesize(luna_lovegood) :- luna_lovegood, !.
hypothesize(cho_chang) :- cho_chang, !.
hypothesize(cedric_diggory) :- cedric_diggory, !.
hypothesize(nymphadora_tonks) :- nymphadora_tonks, !.
hypothesize(muggle) :- muggle, !.
hypothesize(no_identificado). /* no diagnosis */

/* Personaje identification rules */

harry_potter :- masculino, gryffindor, ejercito_de_dumbledore, (tiene_cicatriz ; habil
ron_weasley :- masculino, gryffindor, weasley, ejercito_de_dumbledore.
ginny_weasley :- femenino, gryffindor, weasley, ejercito_de_dumbledore.

hermione_granger :- femenino, gryffindor, (inteligente ; no_se_dice_leviosa).
```

```
dumbledore :- masculino, gryffindor, orden_del_fenix, profesor, inteligente, muere_en_

sirius_black :- masculino, gryffindor, orden_del_fenix, muere_en_alguna_pelicula, veri
remus_lupin :- masculino, gryffindor, orden_del_fenix, muere_en_alguna_pelicula, lobo.

snape :- masculino, slytherin, profesor, mortifago, muere_en_alguna_pelicula, orden_de
voldemort :- masculino, slytherin, mortifago, muere_en_alguna_pelicula, (magia_oscura
bellatrix_lestrange :- femenino, slytherin, mortifago, magia_oscura, muere_en_alguna_p

molly_weasley :- femenino, gryffindor, weasley, verify(es_madre), orden_del_fenix.
draco_malfoy :- masculino, slytherin, malfoy, not(es_mayor), magia_oscura, mortifago.
lucius_malfoy :- masculino, slytherin, malfoy, magia_oscura, mortifago.

dobby :- masculino, verify(necesitaba_un_calcetin).
hagrid :- masculino, gryffindor, semigigante, profesor, orden_del_fenix.

luna_lovegood :- femenino, ravenclaw, inteligente, ejercito_de_dumbledore.
cho_chang :- femenino, ravenclaw, verify(aparece_en_el_libro_4), inteligente, ejercito
cedric_diggory :- masculino, hufflepuff, verify(aparece_en_el_libro_4).

nymphadora_tonks :- femenino, hufflepuff, orden_del_fenix, verify(no_le_gusta_su_prime

/* classification rules */

masculino :- verify(es_masculino), !.
femenino :- not(masculino).

lobo :- verify(se_transforma_en_lobo), !.

inteligente :- verify(inteligente), !.

:- dynamic no_pertenece_a_ninguna_casa/1.
no_pertenece_a_ninguna_casa(true) :- !.

gryffindor :- no_pertenece_a_ninguna_casa(true), verify(pertenece_a_gryffindor), retra
slytherin :- no_pertenece_a_ninguna_casa(true), verify(pertenece_a_slytherin), retract
ravenclaw :- no_pertenece_a_ninguna_casa(true), verify(pertenece_a_ravenclaw), retract
hufflepuff :- no_pertenece_a_ninguna_casa(true), verify(pertenece_a_hufflepuff), retra
muggle :- verify(es_muggle), !.

ejercito_de_dumbledore :- verify(pertenece_a_ejercito_de_dumbledore), !.
orden_del_fenix :- verify(pertenece_a_orden_del_fenix), !.
mortifago :- verify(pertenece_a_mortifago), !.
auror :- verify(auror), !.

bueno :- auror.
```

```
bueno :- ejercito_de_dumbledore.
bueno :- not(mortifago), not(orden_del_fenix).
bueno :- orden_del_fenix.

malo :- mortifago.
malo :- not(orden_del_fenix).
malo :- not(ejercito_de_dumbledore).

semigigante :- verify(semigigante), !.


profesor :- verify(es_profesor), !.
% alumno :- not(profesor).

weasley :- verify(tiene_el_pelo_rojo), !.
malfoy :- verify(tiene_el_pelo_rubio), !.
gordo :- verify(es_gordo), !.

es_mayor :- verify(es_mayor), !.
muere_en_alguna_pelicula :- verify(muere_en_alguna_pelicula), !.

% mas personales
tiene_cicatriz :- verify(tiene_cicatriz), !.
magia_oscura :- verify(practica_magia_oscura), !.
habilidad_parlante_parsel :- verify(tiene_habilidad_parlante_parsel), !.
no_se_dice_leviosa :- verify(no_se_dice_leviosa), !.
no_se_puede_nombrar :- verify(no_se_puede_nombrar), !.


/* how to ask questions */
ask(Question) :-
      write('El personaje tiene la siguiente caracteristica: '),
      write(Question), write('? '),
       read(Response), nl,
       ( (Response == si ; Response == s; Response == yes; Response == y)
       -> assert(yes(Question)) ;
       assert(no(Question)), fail).
:- dynamic yes/1,no/1.
/* How to verify something */
verify(S) :- (yes(S) -> true ; (no(S) -> fail ; ask(S))).
/* undo all yes/no assertions */
undo :- retract(yes(_)),fail.
undo :- retract(no(_)),fail.
undo.
```

5. Explicar ventajas y desventajas de Prolog en la creación de este tipo de sistema.

5.1. Ventajas

- **Facilidad de uso**: Prolog tiene una sintaxis simple y clara, lo que facilita su aprendizaje y uso.
- **Lógica declarativa**: Prolog se basa en la lógica declarativa, lo que permite expresar reglas y hechos de manera natural.
- **Inferencia potente**: Prolog tiene un poderoso motor de inferencia que puede deducir nuevas reglas y hechos a partir de los existentes.
- **Amplia comunidad**: Prolog tiene una comunidad activa de usuarios y desarrolladores que ofrecen soporte y recursos.
- **Código abierto**: Prolog es un lenguaje de código abierto, lo que significa que es gratuito y que cualquiera puede contribuir a su desarrollo.

5.2. Desventajas

- **Eficiencia**: Prolog no es el lenguaje más rápido para ejecutar programas, lo que puede ser un problema para sistemas que requieren un alto rendimiento.
- **Escalabilidad**: Prolog puede tener dificultades para manejar grandes conjuntos de datos o sistemas complejos.
- **Mantenimiento**: El código Prolog puede ser difícil de mantener y actualizar, especialmente para sistemas grandes.
- **Dependencia de la plataforma**: No todas las plataformas tienen implementaciones de Prolog disponibles.
- **Falta de bibliotecas**: En comparación con otros lenguajes, Prolog tiene una menor cantidad de bibliotecas disponibles para tareas específicas.

6. Otro punto de interés...

Nos inspiramos en el juego Akinator para hacer el problema sugerido. `https://en.akinator.com/`. Akinator es un videojuego desarrollado por la empresa francesa Elokence. Se creó en 2007 y se popularizó en 2015. El juego trata de un genio que puede adivinar a qué personaje estás pensando con solo hacerte algunas preguntas. Hay muchos tutoriales en internet que explican cómo funciona el algoritmo de Akinator o cómo hacer un juego similar, pero oficialmente no se ha revelado cómo funciona el algoritmo.