Bienvenidos a MayCEy

MayCEy es el sistema experto creado para la gestión del Aeropuerto de Ingeniería en Computadores del TEC a partir de programación lógica.

Desarrolladores:

David Pereira Jiménez

Randall Samuel Méndez Loría

Jonathan Gonzalez Sánchez

Manual de usuario

MayCEy

Requisitos

Sistema Operativo: Linux

SWI-Prolog 7.7.20 o mayor

Instrucciones de inicio

Se tiene que consultar los tres archivos:

- MayCEy
- DataBase

- sentenceRecognition

En ese orden especifico para evitar errores.

Instrucciones de uso

Para iniciar MayCEy, se tiene que escribir `conectar.`. Una vez iniciada se debe escribir `"hola".` *Para iniciar solicitud* o `"mayday".` *Para iniciar emergencia*.

Sección de solicitud

En esta sección hay dos posibles acciones:

- aterrizar, un ejemplo `"Solicito aterrizar".`
- despegar, un ejemplo `"Solicito despegar".`

A partir de que se a elegido alguna de estas dos acciones las dos se van siguientes preguntas se van a comportar de la misma forma en los dos casos, en orden seria:

- Matricula, posible respuesta `"Mi matricula es tango".`
- Tipo de aeronave, posible respuesta `"Soy un cessna".`
- Dirección, posible respuesta `"Mi direccion es EsteOeste".`

Al responder todas las preguntas de manera satisfactoria se presentara la pista asignada y a la hora que tiene que llegar. Para terminar la sección se escribe `"gracias".`(por ejemplo).

Sección de emergencia

En esta sección lo primero que se pregunta es la emergencia que posee, un ejemplo de respuesta seria `"perdida de Motor".`.

Después se seguirá con las preguntas habituales:

- Matricula, posible respuesta `"Mi matricula es delta".`
- Tipo de aeronave, posible respuesta `Soy un boeing717.`
- Dirección, posible respuesta `"Mi direccion es NorteSur".`

Al igual que la sección de solicitud, se presentara la pista asignada y la hora, mas un añadido que es la acción a tomar según la emergencia que se definió. Para terminar la sección se escribe `"gracias".`(por ejemplo).

Consideraciones Finales

- Cada ves que se llama a `conectar.`, se liberan todas las pista, haciendo entender que a pasado el suficiente tiempo como para que los aviones hagan las acciones programadas.
- Las pistas solo se asignan durante "5 minutos".
- Para terminar la conexión con MayCEy se debe escribir `"cambio".`, al momento inicio de una nueva conversación.

Descripción de Hechos y reglas implementadas

Hechos

Los hechos en este proyecto fueron implementados como tablas, las cuales siguen un formato específico según la necesidad de su uso, estas se exponen una a una a continuación.

Tabla Aeronaves

El formato escogido para el manejo de las aeronaves fue el siguiente: `aeronave(tipo, naves)`. En este caso el tipo hace referencia a la clasficiación de la nave según su envergadura, ya sea pequeña, mediana o grande, mientras que naves corresponde a una lista que contiene todas las naves de la envergadura respectiva. Se escogió implementar la tabla de esta manera a causa del incremento en eficiencia que represantaba, ya que con este método se evitaba tener que definir una a una la aeronave con su respectivo tipo.

Tabla Pistas

En el caso de las pistas, la tabla sigue el formato que se muestra: `pista(numero, largo, tipoNaves, direcciones, disponibilidad, horaAsig, Placa)`, donde numero hace referencia al numero de pista correspondiente, largo indica la longitud de la pista en kilometros, tipoNave es una lista con las envergaduras permitidas en la pista, direccion contiene una lista con las direcciones de aterrizaje o despegue aceptadas en la pista, disponibilidad indica si la pista esta disponible o si se encuentra ocupada por una nave especifica, horaAsig corresponde a la hora en la que se asigno la pista en caso de estar ocupada y la placa contiene el número de

matrícula de la Nave que la ocupa en caso de ser así. Es importante mencionar que esta tabla posee un atributo dinámico, es decir, será modificada durante la ejecucipon del programa a conveniencia.

Se utilizaron listas tanto para el tipo de Naves como para las direcciones, debido a que algunas pistas podían recibir naves de distintas envergaduras y distintas direcciones, y resultaba más eficiente que definirlas una por una.

Tabla Emergencias

La tabla de emergencias fue definida junto con su atención respectiva de la siguiente forma: `respuestaE(emergencia, respuesta)`. Se realizó de esta forma con el propósito de reducir la cantidad de código y hacer el programa más eficiente. De esta forma, para una emergencia dada, se obtendrá de inmediato la respuesta necesaria.

Tabla Condiciones de Vuelo

Por último, la tabla para las condiciones de vuelo posee la siguiente estructura: `condicion(cond, valor)`, donde cond corresponde a la condición de vuelo, ya sea viento, velocidad, peso o distancia, mientras que el valor, como su nombre lo indica hace referencia al valor respectivo de cada condición.

Reglas

En cuanto reglas se estableció lo siguiente:

Miembro

Esta regla se encarga de verificar si un elemento dado se encuentra dentro de una lista, de igual manera, dada. La regla posee la estructura `miembro(elemento, lista)`. Retorna un valor de true si lo encuentra y false en caso contrario.

TipoNave

La regla del tipo de nave se encarga de verificar a que envergadura corresponde la nave que se pasa. La estructura de la regla es `tipoNave(TNave, Nave)`, su objetivo es pasar el primer parámetro como variable para que este sea el valor de retorno, es

decir el tipo de la nave, y el segundo valor, el cual corresponde al nombre del avión, es dado por el usuario.

Asignar Pista

La regla encargada de asignar pistas sigue dos formatos distintos, por un lado se encuentra: `asignarPista(Nave, Dir, Placa)`, la cual recibe el nombre de la nave que va a ocupar la pista, la dirección en la que se acerca, y la placa o matrícula de esta misma; esta primera implementación se ejecuta cuando la solicitud de la pista no recibe una hora de asignación. Por otro lado, se tiene el formato `asignarPista(Nave, HoraAsig, Dir, Placa)`, como se observa la diferencia radica en la presencia de la hora de asignación como parametro adicional, este se ejecuta cuando el usuario sí provee la hora en la que va a ocupar la pista específica.

LiberarPista

Liberar pista, como regla, se encarga de eliminar del registro una pista que se encuentre como ocupada, reasignando los valores a su valor por defecto. De esta forma una nueva aeronave podrá disponer de ella con libertad. Su estructura es `liberarPista(Nave, HoraAsig, Dir, Placa)`, utilizando los mismos parametros de asignar, con el mismo objetivo. A partir de estos valores accede a la pista que se va a liberar y elimina su registro.

LiberarTodo

La regla de liberar todo limpia por completo el registro de pistas y reasigna los valores por defecto de todas y cada una de ellas. Esta regla no recibe parámetros como tal pues no los requiere.

ObtHora

La función de esta regla consiste en consultar la hora actual en la región donde se ubique el usuario, su estructura es `obtHora(H,M)`, donde H corresponde a la hora, y M a los minutos. El propósito de esta regla es que H y M sean variables que serán retornadas para manipularse a conveniencia.

HoraActual

Hora actual es una regla encargada de tomar los valores que retorna `obtHora(H,M)` y convertirlos a formato militar para que puedan ser manipulados como enteros. Su

formato es `horaActual(R)`, donde R corresponde al valor de retorno de la regla correspondiente a la hora transformada al formato militar.

Validar Hora

Esta regla tiene como objetivo verificar la validez de la hora que devuelve `horaActual(R)` ya que al ser un valor entero puede tomar valores como 1163hrs, cuando debería ser 1203hrs, si ese es el caso de la hora recibida la regla se encarga de transformarla al formato correcto. La estructura corresponde a `validarHora(H, NH)`, donde H es la hora para validar y NH es la hora validada que se va a retornar.

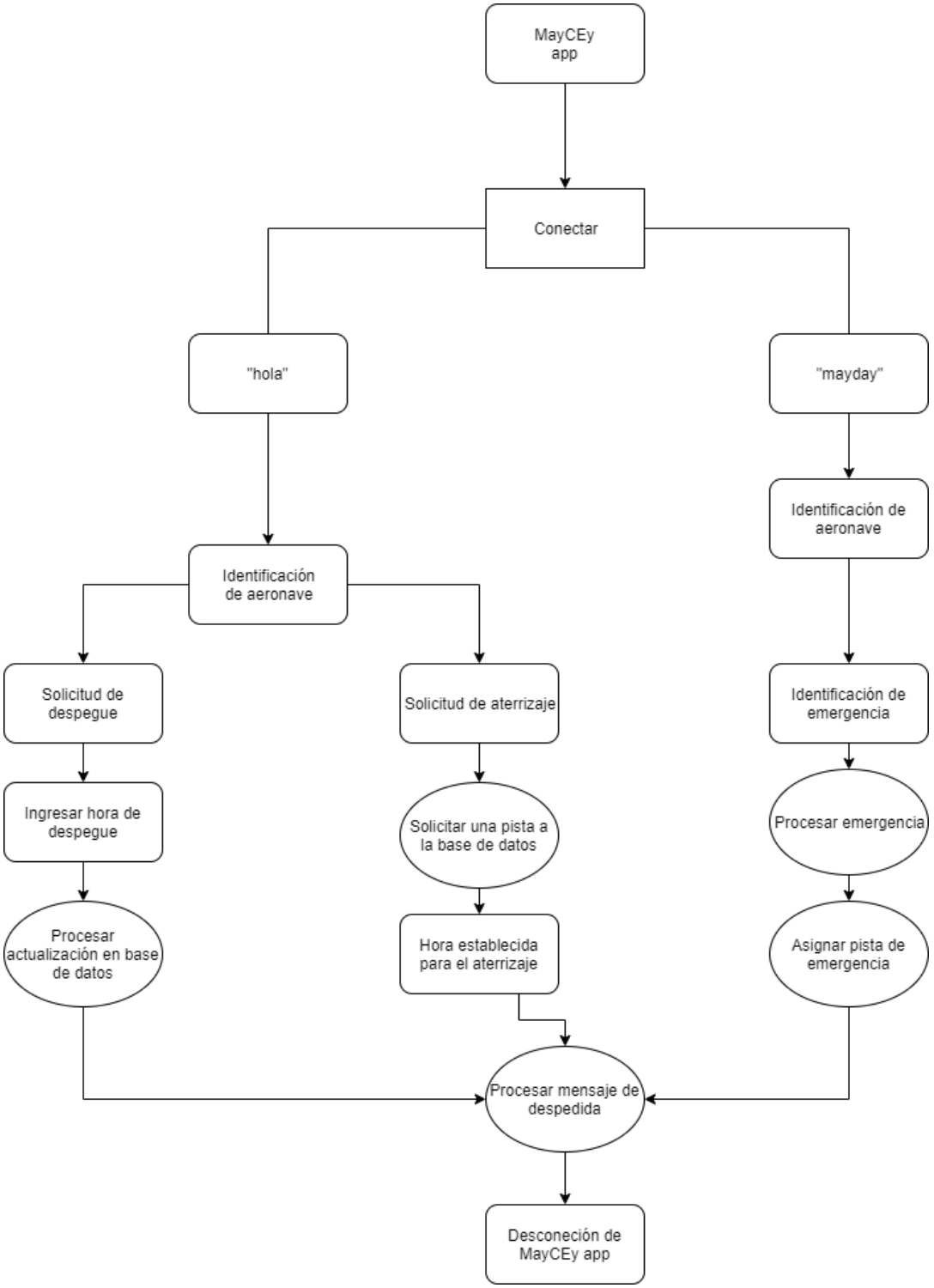
Descripción de las estructuras de datos utilizadas

Listas

Para el proyecto en cuestión, como estructuras de datos, se trabajó únicamente con listas dada su accesible forma de manipular la información. Estas mismas fueron utilizadas en los hechos `aeronave(tipoNave, naves)` y `pista(numeroPista, largoPista, tiposNaves, direcciones, disponibilidad, horaAsig, placa)`, de forma más específica en los atributos `naves`, `tiposNaves` y `direcciones`. En el caso de `naves`, se almacenan todas las naves de la envergadura correspondiente al tipo dentro de la lista, mientras que en `tiposNaves` y `direcciones`, se guardan los tipos de naves que acepta la pista correspondiente y las direcciones permitidas, respectivamente.

Al utilizar estas estructuras con el fin mencionado se logró eficientizar el código de mejor manera, ya que evitó la definición innecesaria de una cantidad de hechos abrumadora.

Descripción detallada de los algoritmos desarrollados



Problemas conocidos

Posible "caídas"

A pesar de que se espera que el programa no se "caiga", si se cumplen algunas condiciones se puede "caer", estas serian:

- Al momento en que el usuario tiene que escribir alguna oración, si el escribe `.` hará que el `read(X)` genere error
- En algunos `read(X)` específicos si se ingresa informa errónea, hará que la conversación se "caiga", lo cual genera que se reinicie la conversación sin "matar" el programa.

Plan de Actividades realizadas por estudiante

Actividad	Descripción	Tiempo	Responsable
Interfaz	Desarrollo de la interfaz con la que el usuario interactuará.	8 días	Randall
Hechos	Desarrollo de la base de conocimiento.	3 días	David
Reglas	Desarrollo de las funciones que trabajaran según el conocimiento que posea.	5 días	David
Vocabulario	Identifica y crea el posible vocabulario que se puede usar.	2 días	Jonathan
DCG	Desarrollo de un reconocedor de oraciones según el vocabulario.	4 días	Jonathan
Fecha de entrega	DeadLine	7/11 a las 11:59	Randall, David, Jonathan

Problemas encontrados

Tratamiento de cortes en la interfaz

Cuando se combino la interfaz con el DCG, apareció una duda la cual seria:

- ¿Que pasa cuando lo que escribió el usuario no es una oración?

Lo normal seria que tirar false pero si hace esto comenzaría a hacer false las demás reglas, lo cual llevaría a que el programa se "caiga". Entonces se opto por tirar un `"-1"` en caso de fallo esto se pasara a una verificación de existencia y después le sigue un corte `!`, entonces en el momento que la verificación falle por corte va tirar a la siguiente regla que pueda cumplir lo pedido.

Modificación de hechos establecidos

Mientras se desarrollaba la base de datos, se presentó la necesidad de modificar la información que contenía los hechos `pistas`, sin embargo, dada la poca experiencia con el lenguaje ProLog, se desconocía como conseguir está modificación. Investigando en internet, se da con una opción que indicaba utilizar las funciones primitivas `assert` y `recall`. Se procedió a intentar por ese camino pero el programa presentaba errores a la hora de ejecutarse. Luego de una nueva investigación se da con la solución final, los hechos debían ser declarados cómo dinámicos mediante la palabra reservada `dynamic/1`, donde el número final debe ser cambiado por la cantidad de argumentos que poseen los hechos. Al implementar esta solución se consiguió modificar con éxito cada parámetro de los hechos.

Pistas podían ser utilizadas por distintos tipos de naves según su envergadura

Por otra parte, dentro del desarrollo de la base de datos de igual manera, se presenta en forma de inconveniente la característica de que las pistas pueden ser utilizadas por naves de distinta envergadura según su tamaño, por lo que debía ser validado en la regla `asignarPista`, por lo que se intentó realizar variaciones de la regla para cada tipo de envergadura, pero además de ser muy poco eficiente, no resultó exitoso el intento. Por lo tanto, mediante trabajo en equipo, pensando en como solucionarlo, se llegó a la decisión de implementar las listas dentro de los hechos `pista`, esto permitió validar si la envergadura pertenecía a la pista utilizando la regla `miembro`.

Conclusiones

Del presente proyecto se pueden obtener un par de conclusiones. En primer lugar, se encuentra una gran modularidad dentro del paradigma lógico debido a la independencia existente entre hechos y reglas, lo cual facilita en gran manera la manipulación del código sin riesgos de afectar significativamente el resto del programa. Además, se observa que es un lenguaje que ayuda a simplificar la expresión de los problemas, y a hacer su interpetación más precisa, esto gracias a la sencillez que posee para implementar las distintas estructuras.

Recomendaciones

En cuanto a la utilización del lenguaje lógico, se recomienda realizar un estudio conciente sobre el funcionamiento del backtracking antes de iniciarse en este paradigma, ya que todo su funcionamiento esta basado en este algoritmo, por lo que una buena comprensión de él ayudara a resolver de forma más eficaz los problemas que se presenten en este lenguaje. Por otra parte, un repaso a los operadores lógicos y a la recursividad no estaría de más para aquel que desee aprender programación lógica, el primero es el fundamento de todo este paradigma y el segundo resulta de bastante utilidad en este tipo de programación.

Bibliografía

- SISTEMAS EXPERTOS: Fundamentos, Metodologías y Aplicaciones
- Sistema Experto PROLOG
- Grammar Rules in Prolog
- Using Definite Clause Grammars in SWI-Prolog
- how to read a file in prolog?
- Arithmetic in Prolog
- Reference manual
- Programming in Prolog: Using the ISO Standard
- Prolog

Bitácora

Bitácora David

Actividad	Descripción	Fecha
Primera Reunión	Se estudia el enunciado y se asignan las actividades corrrespondientes a cada compañero	25/10/2019
Comienza desarrollo base de datos	Se inicia definiendo las tablas necesarias para el programa	28/10/2019
Investigación como modificar hechos	Se requiere modificación de los hechos, se comienza a investigar	29/10/2019
Segunda Reunión	Se comentan avances y complicaciones del proyecto	30/10/2019
Se finaliza implementación de los hechos	Los hechos quedan definidos, sujetos a cambios	31/11/2019
Inicia implementación reglas	Se comienza a desarrollar las reglas necesarias	2/11/2019
Tercera Reunión	Se definen detalles finales del proyecto, prioridades a terminar, entre otros	3/11/2019
Inconveniente Pistas	Pistas pueden ser ocupadas por distintas envergaduras de naves, se investiga solución	4/11/2019

Modificación Pistas	Se modifcan los hechos pistas para incluir listas de envergaduras y direcciones	5/11/2019
Implementación hora asignación	Se comienza implementación de la hora asignación con tiempo real	6/11/2019
Finaliza implementación reglas	Se completa implementación de reglas	7/11/2019

Bitácora Jonathan

Actividad	Descripción	Fecha
Primera Reunión	Se estudia el enunciado y se asignan las actividades corrrespondientes a cada compañero	25/10/2019
Comienza investigación BNF	Investigación sobre posibles formas de implementar los BNFs	29/10/2019
Segunda Reunión	Se comentan avances y complicaciones del proyecto	30/10/2019
Cambio BNF por DCG	De la investigación surge una mejor alternativa llamada DCG, inicia implementación	1/11/2019
Creación base vocabulario	Se comienza a crear la base para el vocabulario requeido para el programa	2/11/2019
Tercera Reunión	Se definen detalles finales del proyecto, prioridades a terminar, entre otros	3/11/2019

Investigación clasificar oraciones	Se investiga como clasificar oraciones según sujeto, verbo y predicado	5/11/2019
Finaliza Base vocabulario	Se completa la base del vocabulario para el programa	6/11/2019
Implementación final	Implementación final vocabulario e interfaz, se conectan ambas partes junto Samuel	7/11/2019

Bitácora Samuel

Actividad	Descripción	Fecha
Primera Reunión	Se estudia el enunciado y se asignan las actividades corrrespondientes a cada compañero	25/10/2019
Inicia desarrollo interfaz	Se comienza a desarrollar una interfaz básica de prueba	27/10/2019
Se implementan oraciones a conversacion	Se añaden oraciones predeterminadas a la conversación para avanzar con la interfaz	29/10/2019
Segunda Reunión	Se comentan avances y complicaciones del proyecto	30/10/2019
Implementación aterrizaje y despegue	Se implementa solicictud de aterrizaje y de despegue en la conversación	2/11/2019
Tercera Reunión	Se definen detalles finales del proyecto, prioridades a terminar, entre otros	3/11/2019
Implementación Mayday	Implementada la conversación del Mayday en la interfaz	4/11/2019
Validación de errores en la conversación	Se validan algunos errores presentes en la interfaz, programa aun se cae con punto	5/11/2019

Finaliza implementación interfaz	Se completó el desarrollo de la interfaz, error con un punto sin solución	6/11/2019
Implementación final	Implementación final vocabulario e interfaz, se conectan ambas partes junto Jonathan	7/11/2019