



**UNIVERSIDAD DE SANTIAGO DE CHILE  
FACULTAD DE INGENIERÍA  
DEPARTAMENTO DE INGENIERÍA EN INFORMÁTICA**

**LABORATORIO 2  
PARADIGMAS DE PROGRAMACIÓN**

Gabriel Gaete L.

Profesores: Roberto González  
Daniel Gacitúa  
Víctor Flores

Fecha de Entrega: 21 de Mayo del 2018

Santiago de Chile

1 - 2018

# TABLA DE CONTENIDO

<b><i>CAPÍTULO 1. INTRODUCCIÓN .....</i></b>	<b><i>4</i></b>
<b><i>CAPÍTULO 2. MARCO TEÓRICO .....</i></b>	<b><i>5</i></b>
2.1 PARADIGMA LÓGICO .....	5
2.2 PREDICADO .....	5
2.3 CLÁUSULA DE HORN .....	5
<b><i>CAPÍTULO 3. DESCRIPCIÓN DEL PROBLEMA .....</i></b>	<b><i>6</i></b>
3.1 ANÁLISIS.....	7
<b><i>CAPÍTULO 4. DESCRIPCIÓN DE LA SOLUCIÓN.....</i></b>	<b><i>8</i></b>
<b><i>CAPÍTULO 5. RESULTADOS .....</i></b>	<b><i>11</i></b>
<b><i>CAPÍTULO 6. CONCLUSIONES.....</i></b>	<b><i>14</i></b>
<b><i>CAPÍTULO 7. REFERENCIAS .....</i></b>	<b><i>15</i></b>

## TABLA DE FIGURAS

Figura 1 Representación de un mensaje .....	8
Figura 2 Código fuente de beginDialog .....	8
Figura 3 Código fuente de endDialog .....	9
Figura 4 Código fuente de sendMessage .....	9
Figura 5 Código fuente de logToStr .....	9
Figura 6 Código fuente de test .....	10
Figura 7 Ejemplo de beginDialog .....	11
Figura 8 Ejemplo de sendMessage .....	12
Figura 9 Ejemplo de endDialog .....	12
Figura 10 Ejemplo de logToStr .....	13
Figura 11 Ejemplo de test .....	13

# CAPÍTULO 1. INTRODUCCIÓN

Un *chatbot* o un bot conversacional es “*un robot capaz de hablar e interactuar imitando el comportamiento humano, ya sea oral o por escrito, respondiendo a las preguntas y reclamos de los usuarios.*” (Herrero, s.f.) En otras palabras, es un programa que permite simular una conversación con una persona, entregando respuestas automáticas a entradas hechas por un usuario. Esta conversación, habitualmente se establece a través de texto, sin embargo, hay modelos que disponen de una interfaz multimedia. También se han desarrollado chatbots que utilizan conversores de texto a sonido, dotando de mayor realismo a la interacción con el usuario.

El presente informe tiene por objetivo principal ser una referencia a una línea de pensamiento y a un contexto de desarrollo del código fuente que lo acompaña para la presentación del segundo laboratorio del curso “*Paradimas de programación*”. En esta oportunidad, se hará uso del paradigma lógico bajo el lenguaje de programación *Prolog*.

El paradigma lógico corresponde a un paradigma dentro de la programación declarativa, y se basa principalmente en el concepto de predicado o relación entre elementos, de esta forma, la interacción con este paradigma es a través de consultas de carácter booleano, es decir, verdaderas o falsas, e intenta responder a consultas de la siguiente forma: ¿Es correcta la relación (predicado) entre X e Y?, o bien ¿para qué valores de Y es correcta la relación con X?. Todas estas consultas son respondidas a partir de una base de conocimientos previamente definida.

El problema a resolver en este laboratorio es el desarrollo de un *chatbot*, con el cual se deberá mantener una conversación básica, protocolar, en la que exista un flujo conversacional coherente. El contexto para este chatbot será una venta de pasajes a capitales regionales dentro de Chile.

La solución implementada trabaja en base a un conjunto de predicados ya definidos, y cláusulas de Horn para realizar consultas que permitan simular un flujo de conversación entre un usuario y un chatbot. Esto se realiza en base a trabajo sobre listas en las cuales se mantiene una base de conocimientos para poder determinar respuestas.

Se establece entonces, como objetivo en este laboratorio, el aplicar y demostrar los conocimientos adquiridos en cátedra con respecto a lo que es el paradigma de programación lógico, utilizando el lenguaje de programación *Prolog*.

## CAPÍTULO 2. MARCO TEÓRICO

### 2.1 PARADIGMA LÓGICO

Un razonamiento al momento de resolver problemas en matemáticas se basa en lo que es la lógica de primer orden. El conocimiento básico de las matemáticas se puede representar en la lógica en forma de axiomas, a los cuales se añaden reglas formales para deducir cosas verdaderas, o también conocidas como *teoremas*. Gracias al trabajo de matemáticos de finales del siglo pasado y de principios de éste, es que se encontró la manera de representar computacionalmente el razonamiento lógico, permitiendo el nacimiento de lenguajes de programación lógicos. También se conoce a éstos como lenguajes *declarativos*, porque lo que un programador debe hacer para resolver un problema bajo este paradigma, es describir éste a través de axiomas y reglas de deducción.

De esta manera, en los lenguajes lógicos se utiliza el formalismo de la lógica de primer orden para representar el conocimiento sobre un problema, para posteriormente hacer preguntas que, si se demuestra que se pueden deducir a partir del conocimiento dado a través de los axiomas y de las reglas de deducción estipuladas, se vuelven teoremas. El conocimiento del problema se expresa en forma de predicados (axiomas) que establecen relaciones entre los símbolos que representan los datos del dominio del problema.

### 2.2 PREDICADO

Corresponde a una estructura o aserción compuesta de términos, en la que se unen, o se representa una relación entre estos.

### 2.3 CLÁUSULA DE HORN

Una cláusula de Horn corresponde a un conjunto de predicados que hace que una afirmación sea verdadera. Un programa lógico se compone de *cláusulas de Horn* y de *hechos*, los cuales corresponden a casos especiales de cláusulas de Horn que son verdaderas sin condiciones.

## CAPÍTULO 3. DESCRIPCIÓN DEL PROBLEMA

Se solicita crear un programa en el lenguaje de programación *Prolog* que simule un bot conversacional (chatbot). Para este laboratorio, el tema del chatbot será una venta de pasajes a capitales regionales de Chile. Este debe funcionar en base a una estructura chatbot, un *log* o historial de conversaciones, y una semilla (seed), para realizar operaciones aleatorias. El programa debe ser capaz de realizar las siguientes funcionalidades:

1. **Implementación de un TDA:** Haciendo uso de la estructura TDA de 6 capas, implementar abstracciones adecuadas al problema.
2. **beginDialog(Chatbot, InputLog, Seed, OutputLog):** Predicado a través del cual se puede determinar el mensaje de bienvenida que se da por parte del Chatbot hacia el usuario.
3. **sendMessage(Msg, Chatbot, InputLog, Seed, OutputLog):** Predicado que permite mantener el flujo de conversación entre el usuario y el chatbot. El flujo de conversación se determina a través del resultado obtenido en el OutputLog.
4. **endDialog(Chatbot, InputLog, Seed, OutputLog):** Predicado a través del cual se pone fin a la conversación, entregando la despedida dentro de un log completo que contiene toda la conversación entre un usuario y un chatbot.
5. **logToStr(Log, SreRep):** Predicado que permite transformar un log desde su representación (lista) a un string, con un formato que permite ser impreso por pantalla y que sea de más fácil entendimiento para el usuario.
6. **test(User, Chatbot, InputLog, Seed, OutputLog):** Predicado que permite simular una conversación entre un usuario y un chatbot. El resultado de esta conversación simulada se obtiene en el OutputLog.

Estas funcionalidades tienen un formato definido para su implementación y desarrollo, trabajan con estructuras específicas (principalmente listas). Se tiene la libertad de definir e implementar éstas estructuras acorde a las necesidades de la implementación deseada.

Las cláusulas de Horn, trabajan en base a predicados (hechos) definidos tanto para identificar los mensajes enviados por el usuario, como para determinar la respuesta que debe dar el Chatbot frente a un determinado mensaje. Estos hechos se encuentran dentro de un archivo llamado "hechos.pl", en la misma carpeta del código fuente.

### 3.1 ANÁLISIS

El principal problema que se presenta en este laboratorio radica en cómo definir los predicados requeridos, ya que no es posible modificar la firma de estos, haciendo que la definición de los componentes de cada una de ellas sea esencial para una correcta implementación. Debido a su naturaleza, *prolog* trabaja de manera recursiva, lo que lleva a que las condiciones de borde puedan ayudar a reducir el número de recursiones que el programa realiza. Por otra parte, y teniendo en cuenta que el problema principal de este laboratorio es la construcción de condiciones para que se cumplan ciertos predicados, es que se debe procurar mantener definiciones lógicas consistentes, con el fin de evitar por ejemplo la cláusula **not**, para no escaparse de las definiciones realizadas en la base de conocimiento.

## CAPÍTULO 4. DESCRIPCIÓN DE LA SOLUCIÓN

Como se mencionó en el capítulo anterior, puesto que los predicados ya están definidos y sus firmas no son modificables, es que el principal desafío recae en la implementación de las cláusulas.

Así, definiendo una implementación en *Prolog* para los mensajes, sería una lista de tres elementos, donde cada uno corresponde a un string que representa la fecha, el autor del mensaje, y el contenido del mensaje, respectivamente. A continuación, se presenta un ejemplo de la representación en *Prolog* de un mensaje.

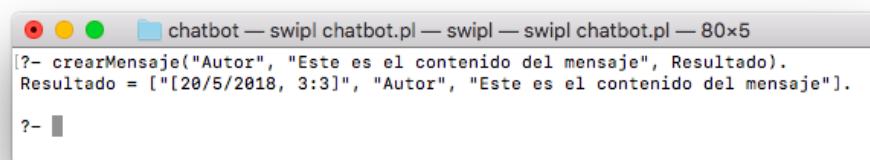


Figura 1 Representación de un mensaje

De esta forma, al momento de crear una representación para el log, se puede acceder a toda la información de los mensajes a través de los selectores para este TDA, logrando que los predicados que deben retornar un log actualizado (**beginDialog**, **sendMessage**, **endDialog**) tengan una manera directa de trabajar con este. Las siguientes figuras muestran el código fuente de **beginDialog** y de **endDialog**, cuya lógica y funcionamiento es similar.

```
beginDialog(Chatbot, InputLog, Seed, OutputLog):-
    chatbot(Chatbot),
    nth0(0, Chatbot, Greetings),
    set_random(seed(Seed)),
    NumberRandom is random(Seed),
    length(Greetings, GreetingsLength),
    Position is NumberRandom mod GreetingsLength,
    nth0(Position, Greetings, Content),
    nth0(2, Chatbot, ListIDRates),
    reverse(ListIDRates, ReversedListIDRates),
    nth0(0, ReversedListIDRates, PairIDRate),
    nth0(0, PairIDRate, ID),
    createDate(Date),
    append(InputLog, [[Date, "BeginDialog", "ID:", ID]], BeginDialogLog),
    crearMensaje("Bot:", Content, Result),
    messageToLog(BeginDialogLog, Result, OutputLog), !.
```

Figura 2 Código fuente de beginDialog



```

endDialog(Chatbot, InputLog, Seed, OutputLog):-
    chatbot(Chatbot),
    nth0(7, Chatbot, GoodbyeList),
    set_random(seed(Seed)),
    NumberRandom is random(Seed),
    length(GoodbyeList, LengthGoodbyeList),
    Position is NumberRandom mod LengthGoodbyeList,
    nth0(Position, GoodbyeList, GoodbyeText),
    crearMensaje("Bot:", GoodbyeText, GoodbyeMessage),
    messageToLog(InputLog, GoodbyeMessage, ModifiedLog),
    nth0(2, Chatbot, ListIDRates),
    reverse(ListIDRates, ReversedListIDRates),
    nth0(0, ReversedListIDRates, PairIDRate),
    nth0(0, PairIDRate, ID),
    createDate(Date),
    append(ModifiedLog, [[Date, "EndDialog", "ID", ID]], OutputLog).

```

*Figura 3 Código fuente de endDialog*

Por otro lado, **sendMessage** requiere pasos adicionales. Para lograr interpretar los mensajes que entrega el usuario, con el fin de entregar una respuesta que se acomode al flujo de la conversación, se trata al contenido del mensaje del usuario como una lista de strings; luego estos son intersectados con una series de palabras para interpretar el mensaje. Una vez interpretado el mensaje, se determina la respuesta que debe entregar el chatbot. La siguiente figura muestra el código fuente de este predicado.

```

sendMessage(Msg, Chatbot, InputLog, Seed, OutputLog):-
    chatbot(Chatbot),
    set_random(seed(Seed)),
    NumberRandom is random(Seed),
    crearMensaje("Usuario:", Msg, Result2),
    messageToLog(InputLog, Result2, ModifiedLog2),
    determineAnswer(Msg, Chatbot, NumberRandom, ModifiedLog2, Answer2),
    messageToLog(ModifiedLog2, Answer2, OutputLog).

```

*Figura 4 Código fuente de sendMessage*

Una vez realizados los predicados descritos anteriormente, la implementación de **logToStr** se reduce a iterar recursivamente sobre el log, concatenando directamente cada TDA Mensaje presente en su interior a un string. Para eso, se utiliza un predicado auxiliar que permita la concatenación, como lo muestra la siguiente figura.

```

logToStr(Log, StrRep):-
    logToStrAux(Log, "", StrRep).

```

*Figura 5 Código fuente de logToStr*

Teniendo todas estas funcionalidades, el predicado **test** ya puede ser implementado. La siguiente figura, muestra el código fuente de dicho predicado.

```
test(User, Chatbot, InputLog, Seed, OutputLog):-
    chatbot(Chatbot),
    user1(User),
    beginDialog(Chatbot, InputLog, Seed, BeginLog),
    recursiveTest(User, Chatbot, BeginLog, Seed, OutputLog);

    chatbot(Chatbot),
    user2(User),
    beginDialog(Chatbot, InputLog, Seed, BeginLog),
    recursiveTest(User, Chatbot, BeginLog, Seed, OutputLog);

    chatbot(Chatbot),
    user3(User),
    beginDialog(Chatbot, InputLog, Seed, BeginLog),
    recursiveTest(User, Chatbot, BeginLog, Seed, OutputLog);

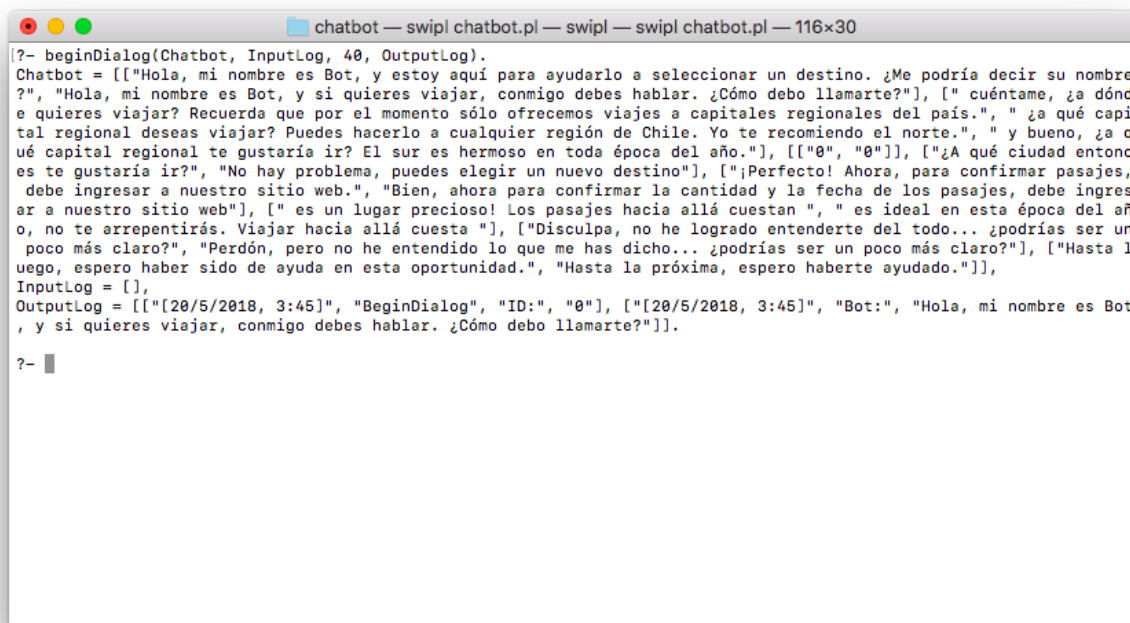
    chatbot(Chatbot),
    beginDialog(Chatbot, InputLog, Seed, BeginLog),
    recursiveTest(User, Chatbot, BeginLog, Seed, OutputLog).
```

*Figura 6 Código fuente de test*

Nótese que en la figura 6, se tienen distintas definiciones de usuario. Esto se da porque por enunciado, se pide entregar al menos 3 ejemplos de usuarios distintos. Sin embargo, se agrega la posibilidad de que el usuario ingrese una lista de strings “personalizada”, es decir, que no esté dentro de la base de conocimientos del programa.

## CAPÍTULO 5. RESULTADOS

Debido a la naturaleza de lo que es un chatbot, este no responde de manera booleana (Verdadero o Falso), sino que debe ser capaz de generar texto con el cual se permita la conversación con el usuario. Se puede apreciar en la siguiente figura, los resultados que entrega la utilización del predicado **beginDialog**. Cabe destacar, que como el usuario no conoce la representación interna de la estructura Chatbot, esta no es necesaria de ingresar, por lo que *prolog* también mostrará, aparte el *log* de salida, la correspondiente estructura chatbot utilizada para llevar a cabo la conversación.



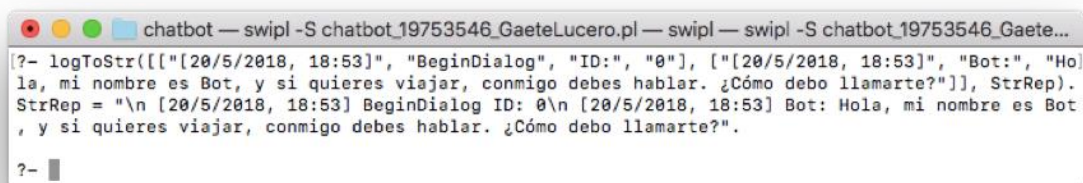
```
?- beginDialog(Chatbot, InputLog, 40, OutputLog).
Chatbot = [{"Hola, mi nombre es Bot, y estoy aquí para ayudarlo a seleccionar un destino. ¿Me podría decir su nombre?", "Hola, mi nombre es Bot, y si quieres viajar, conmigo debes hablar. ¿Cómo debo llamarte?"}, {"cuéntame, ¿a dónde quieres viajar? Recuerda que por el momento sólo ofrecemos viajes a capitales regionales del país.", "¿a qué capital regional deseas viajar? Puedes hacerlo a cualquier región de Chile. Yo te recomiendo el norte.", "y bueno, ¿a qué capital regional te gustaría ir? El sur es hermoso en toda época del año."}, [{"0", "0"}], [{"¿A qué ciudad entonces te gustaría ir?", "No hay problema, puedes elegir un nuevo destino"}], [{"¡Perfecto! Ahora, para confirmar pasajes, debe ingresar a nuestro sitio web.", "Bien, ahora para confirmar la cantidad y la fecha de los pasajes, debe ingresar a nuestro sitio web"}], [{"es un lugar precioso! Los pasajes hacia allá cuestan ", "es ideal en esta época del año, no te arrepentirás. Viajar hacia allá cuesta "}, {"Disculpa, no he logrado entenderte del todo... ¿podrías ser un poco más claro?"}, {"Perdón, pero no he entendido lo que me has dicho... ¿podrías ser un poco más claro?"}, {"Hasta luego, espero haber sido de ayuda en esta oportunidad.", "Hasta la próxima, espero haberte ayudado."}],
InputLog = [],
OutputLog = [{"[20/5/2018, 3:45]", "BeginDialog", "ID:", "0"}, {"[20/5/2018, 3:45]", "Bot:", "Hola, mi nombre es Bot, y si quieres viajar, conmigo debes hablar. ¿Cómo debo llamarte?"}].
```

Figura 7 Ejemplo de *beginDialog*

Por otra parte, para **sendMessage**, se necesita el resultado del OutputLog del predicado anterior. En la siguiente figura, se ha copiado el outputLog de la Figura 7, produciendo el siguiente resultado.



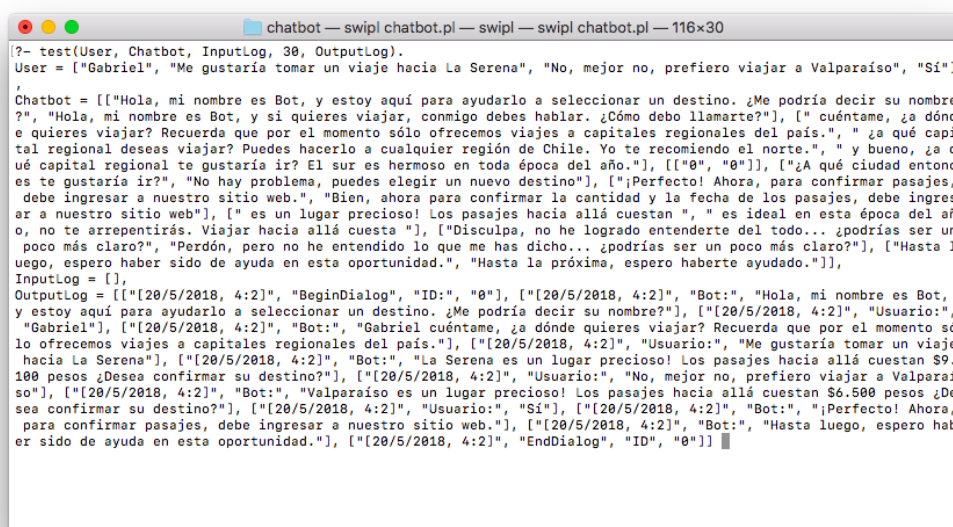
La siguiente figura, muestra el resultado generado por la funcionalidad **logToStr**, dado cierto log de entrada.



```
chatbot — swipl -S chatbot_19753546_GaeteLucero.pl — swipl — swipl -S chatbot_19753546_Gaete...
[?- logToStr([["[20/5/2018, 18:53]", "BeginDialog", "ID:", "0"], ["[20/5/2018, 18:53]", "Bot:", "Ho
la, mi nombre es Bot, y si quieres viajar, conmigo debes hablar. ¿Cómo debo llamarte?"]], StrRep).
StrRep = "\n [20/5/2018, 18:53] BeginDialog ID: 0\n [20/5/2018, 18:53] Bot: Hola, mi nombre es Bot
, y si quieres viajar, conmigo debes hablar. ¿Cómo debo llamarte?".
?- ]
```

Figura 10 Ejemplo de logToStr

Por último, el predicado **test** se muestra en la siguiente figura. Nótese que, en caso de no especificarse entrada de User, permite simular las tres conversaciones previamente definidas por el programador.



```
chatbot — swipl chatbot.pl — swipl — swipl chatbot.pl — 116x30
[?- test(User, Chatbot, InputLog, 30, OutputLog).
User = ["Gabriel", "Me gustaría tomar un viaje hacia La Serena", "No, mejor no, prefiero viajar a Valparaíso", "Sí"]
,
Chatbot = [{"Hola, mi nombre es Bot, y estoy aquí para ayudarlo a seleccionar un destino. ¿Me podría decir su nombre
?", "Hola, mi nombre es Bot, y si quieres viajar, conmigo debes hablar. ¿Cómo debo llamarte?"}, {"cuéntame, ¿a dónde
quieres viajar? Recuerda que por el momento sólo ofrecemos viajes a capitales regionales del país."}, {"¿a qué capi
tal regional deseas viajar? Puedes hacerlo a cualquier región de Chile. Yo te recomiendo el norte."}, {"y bueno, ¿a q
ué capital regional te gustaría ir? El sur es hermoso en toda época del año."}, [{"0", "0"}], [{"¿A qué ciudad entonc
es te gustaría ir?", "No hay problema, puedes elegir un nuevo destino"}, {"¡Perfecto! Ahora, para confirmar pasajes,
debe ingresar a nuestro sitio web."}, {"Bien, ahora para confirmar la cantidad y la fecha de los pasajes, debe ingres
ar a nuestro sitio web"}, {"es un lugar precioso! Los pasajes hacia allá cuestan "}, {"es ideal en esta época del añ
o, no te arrepentirás. Viajar hacia allá cuesta "}, {"Disculpa, no he logrado entenderte del todo... ¿podrías ser un
poco más claro?", "Perdón, pero no he entendido lo que me has dicho... ¿podrías ser un poco más claro?"}, {"Hasta l
uego, espero haber sido de ayuda en esta oportunidad."}, {"Hasta la próxima, espero haberte ayudado."}],
InputLog = [],
OutputLog = [{"[20/5/2018, 4:2]", "BeginDialog", "ID:", "0"}, {"[20/5/2018, 4:2]", "Bot:", "Hola, mi nombre es Bot,
y estoy aquí para ayudarlo a seleccionar un destino. ¿Me podría decir su nombre?"}, {"[20/5/2018, 4:2]", "Usuario:",
"Gabriel"}, {"[20/5/2018, 4:2]", "Bot:", "Gabriel cuéntame, ¿a dónde quieres viajar? Recuerda que por el momento só
lo ofrecemos viajes a capitales regionales del país."}, {"[20/5/2018, 4:2]", "Usuario:", "Me gustaría tomar un viaje
hacia La Serena"}, {"[20/5/2018, 4:2]", "Bot:", "La Serena es un lugar precioso! Los pasajes hacia allá cuestan $9.
100 pesos ¿Desea confirmar su destino?"}, {"[20/5/2018, 4:2]", "Usuario:", "No, mejor no, prefiero viajar a Valparaí
so"}, {"[20/5/2018, 4:2]", "Bot:", "Valparaíso es un lugar precioso! Los pasajes hacia allá cuestan $6.500 pesos ¿De
sea confirmar su destino?"}, {"[20/5/2018, 4:2]", "Usuario:", "Sí"}, {"[20/5/2018, 4:2]", "Bot:", "¡Perfecto! Ahora,
para confirmar pasajes, debe ingresar a nuestro sitio web."}, {"[20/5/2018, 4:2]", "Bot:", "Hasta luego, espero hab
er sido de ayuda en esta oportunidad."}, {"[20/5/2018, 4:2]", "EndDialog", "ID", "0"}]
```

Figura 11 Ejemplo de test



## CAPÍTULO 6. CONCLUSIONES

El paradigma lógico tiene la ventaja de ser simple, los “algoritmos” pueden mejorarse al modificar las componentes de control sin necesariamente cambiar la lógica del programa. Este paradigma presenta además una sencillez para representar estructuras de datos complejas, sin tener la necesidad definir estas estructuras, como se podría hacer en otros paradigmas, por lo que se presenta una increíble potencia para resolver problemas a gran escala, sólo con definir relaciones entre los elementos relevantes. Sin embargo, por lo mismo, puede pasar a ser una mala opción para la resolución de algunos problemas, puesto que se deben definir **ABSOLUTAMENTE TODAS** las relaciones en la base de conocimiento para poder establecer las reglas y trabajar en base a éstas, teniendo en cuenta que si esta base de conocimiento no posee la información suficiente, las respuestas pueden retornar valores no esperados.

Dado que un chatbot se basa en el reconocimiento de lenguaje natural, permitiendo procesar respuestas frente a mensajes de un usuario, es que en particular *Prolog* es una buena herramienta, ya que en la base de conocimientos se pueden definir las palabras que el chatbot puede llegar a reconocer, siendo este un buen lenguaje para el procesamiento de lenguaje natural.

Al ser comparado con el paradigma funcional, para este laboratorio, resulta ser una mejor herramienta el paradigma lógico, por lo expresado anteriormente.

Finalmente, dado que se ha logrado desarrollar un *chatbot* o *bot conversacional* como ha sido requerido, de paso demostrando los conocimientos expuestos en la cátedra de la clase y se ha obtenido el resultado esperado, es posible concluir que se ha logrado cumplir con el objetivo de este laboratorio.

## **CAPÍTULO 7. REFERENCIAS**

Herrero, C. (s.f.). No son mis cookies. Recuperado el 21 de Abril de 2018, de No son mis cookies: <http://nosinmiscookies.com/que-es-un-chatbot/>