uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Informática

2018-2019

Trabajo Fin de Grado

Aplicaciones de la Programación Lógica Probabilística usando Problog2

Christian Carrasco Vega

Tutor/es

Susana Fernández Arregui

Leganés, 2019

Agradecimientos

A mi tutora del TFG Susana, que sin su orientación no hubiera podido ponerle un rumbo al trabajo.

A mis padres y a mi hermana, que me apoyaron sin entender qué estaba haciendo.

A Rubén, por tantas horas al otro lado del teléfono entre audio y audio.

Y a Silvia, por soportarme todos estos meses y aguantarme tantos y tantos días.

Índice de contenidos

Índice de ilustraciones

Índice de tablas

CAPÍTULO 1. INTRODUCCIÓN

"La teoría de la probabilidad es en el fondo nada más que sentido común reducido a cálculo; nos permite apreciar con exactitud aquello que las mentes rigurosas pueden sentir con una especie de instinto que a veces no pueden explicar; nos enseña a evitar las ilusiones que con frecuencia nos engañan... no hay ciencia más digna de nuestra contemplación".

Pierre-Simon Laplace, 1985

En este capítulo haremos una breve introducción al documento, situando el contexto inicial en el que nos encontramos, las motivaciones que nos han llevado a realizar este proyecto, los objetivos que queremos cumplir, el marco regulador en el que se sitúa el documento y una breve explicación de la estructura que tendrá.

1.1 Contexto inicial

En el mundo actual en el que nos encontramos el uso del aprendizaje para resolver problemas que se presentan en la vida real tiene cada vez un impacto mayor. Desde hace siglos tenemos la capacidad de resolver los problemas que se nos plantean, ya sea una toma de decisiones en un momento oportuno o la solución al problema planteado.

Esta capacidad para resolver problemas la adquirimos desde que somos niños a través de la enseñanza y el perfeccionamiento del conocimiento durante nuestra madurez. Dentro de esta enseñanza una de las partes más importantes será la lógica, que nos permitirá sacar conclusiones del problema presentado, sin importar que sea verdadero o falso [1], por lo que adquiriremos conocimiento sobre él para un problema futuro con las mismas características. A este concepto se le conoce como aprendizaje.

Por otra parte, esta necesidad de resolver problemas junto a los avances tecnológicos de los últimos años hace que surja el concepto de la programación, a partir del cual buscamos generar modelos y programas que aprendan las soluciones a nuestros problemas de manera autónoma.

¿Qué problema presenta la programación?

Es cierto que la programación, en su definición como desarrollo de aplicaciones informáticas y tal y como la conocemos en la actualidad, tiene infinitas capacidades que a medida que pasa el tiempo se van actualizando y perfeccionando, pero la capacidad que en nuestro caso nos interesa es la de resolver problemas que se puedan presentar en el mundo real y que tengan que ser resueltos por un ser humano.

Enfocándolo de esta manera es más complicado su uso si no tenemos en cuenta el pensamiento lógico humano. Este es necesario para resolver ciertos problemas que el programa pueda presentarlos como una solución posible, mientras que llevándolo al mundo real no se puede realizar porque quizá supere las capacidades humanas.

Por ello es necesario presentar la programación lógica, un concepto de programación donde se unifica la programación declarativa clásica con los conceptos de lógica formal que se conocen hasta la fecha. De esta manera en vez de proporcionar al programa una serie de instrucciones que debe realizar para conseguir el objetivo deseado, se proporciona al programa el problema que se quiere llegar a resolver y a partir de mecanismos de inferencia (basados en reglas) [2] con la información proporcionada, se ejecuta buscando la solución al problema, imitando al comportamiento humano.

¿La programación lógica resuelve todos nuestros problemas?

Por desgracia, no. Es cierto que la programación lógica nos proporciona las capacidades necesarias para resolver problemas de la vida real, pero sigue sin adecuarse de manera exacta a estos problemas. Esta razón se debe a que de alguna manera en la vida real nunca puedes estar seguro de que algún hecho vaya a ocurrir siempre. Aunque dependa de alguna condición que su aparición sea extremadamente difícil que ocurra, por pequeña que sea esa su probabilidad de que no ocurra, se debe tener en cuenta esta incertidumbre presente en el mundo real.

La manera de tratar la incertidumbre mediante el uso de la programación lógica es mediante la adicción a esta del concepto de probabilidad, generando en este caso la programación lógica probabilística.

Con la programación lógica probabilística se seguirá el mismo funcionamiento que en la programación lógica, pero teniendo en cuenta que los hechos presentados no tienen por qué ocurrir siempre, como ocurre en la vida real.

Para ver las capacidades que tiene esta programación se puede hacer uso del lenguaje *Problog2*, una extensión del lenguaje de *Prolog* que basa su funcionamiento en este, pero incluyendo el trato con la incertidumbre. A lo largo del trabajo veremos el funcionamiento de este lenguaje y las capacidades que presenta junto a otros lenguajes de programación lógica probabilística.

Aplicaciones de la programación lógica probabilística

Entre todas las aplicaciones que tiene la programación lógica probabilística para resolver problemas reales para un ser humano, nos centraremos sobretodo en las técnicas de reconocimiento de actividades en el tiempo, aquellas que dada cierta información del problema realiza el reconocimiento de lo que ocurre en él.

Este reconocimiento funciona de tal manera que a partir de cierta información clasificada con una serie de datos de entrada es capaz de generar predicciones sobre qué tipo de salida, en este caso la actividad que se realiza, se ejecutará previsiblemente. Combinando todas las técnicas vistas, un programa con todas estas características sería capaz de predecir actividades de la manera más real posible.

1.2 Motivación

La motivación principal por la que se ha elegido este tema para la realización del trabajo de fin de grado es la profundización en el ámbito de estudio de la Inteligencia Artificial, estudiada a lo largo de la carrera y que en la actualidad tiene una importancia considerable gracias a los continuos avances que surgen, incluso a día de hoy, que hacen que sea difícil establecer un límite de las capacidades que presenta.

Centrándonos en la programación lógica probabilística, se han desarrollado muchas herramientas para su utilización, como son PHA, PRISM, SLP o MLN. Todas estas tienen la capacidad de generar probabilidades en sus fórmulas lógicas, sin embargo, todas cuentan con limitaciones y restricciones utilizadas para facilitar su computación que no permiten al algoritmo desarrollar su funcionalidad de la manera más precisa.

Por otro lado, *Problog2* tiene un uso muy simple que permite modelar y obtener respuestas rápidamente además de la capacidad de tratar conjuntos de datos muy grandes. Una de las principales razones por la que ocurre esto, a diferencia de los anteriores, es que *Problog2* basa su inferencia en un mundo cerrado, es decir, se centra en representaciones de alto nivel simbólicos donde las variables siempre están definidas, lo que genera esta capacidad de ejecución rápida y eficaz.

Además de esto, otra motivación importante para el estudio de *Problog2* es que es uno de los lenguajes más modernos que tratan la programación lógica probabilística, y por ello cuentan con un blog en el que están presentes los desarrolladores del lenguaje [3] y en el que se discuten aspectos importantes a tratar para mejorar las capacidades del lenguaje, por lo que podemos decir que el lenguaje en sí está 'vivo' a día de hoy y se sigue mejorando y perfeccionando.

Por esta razón compararemos este lenguaje con las capacidades que presentan otros para decidir si el uso de *Problog2* para desarrollar nuestro modelo es el correcto. Nuestro modelo para ver las aplicaciones que tienen la programación lógica probabilística se basará en el reconocimiento de actividades, pero esto engloba muchas posibilidades. Nos centraremos en el ámbito del reconocimiento de actividades en vídeos de vigilancia. Las actividades que ocurren en estos vídeos están bastante acotadas por lo que no se tendrá que divagar en exceso para saber qué actividad se está realizando.

Como hemos dicho, el modelo será una aproximación lo más real posible a lo que sería un sistema capaz de reconocer actividades en tiempo real. Nuestro modelo se basará en el reconocimiento de actividades a partir de datos ya procesados de bases de datos de vídeos de vigilancia, ya que el desarrollo de un sistema para el tratamiento de datos sería mucho más costoso y quedaría fuera del ámbito en el que se centra este trabajo, ya que está mas centrado en Deep Learning.

1.3 Objetivos del trabajo

El objetivo a grandes rasgos que se plantea de este trabajo será, tal y como indica el título del documento, el de investigar las aplicaciones que tiene la programación lógica probabilística a partir del uso del lenguaje de programación *Problog2*.

Por lo tanto, los objetivos principales que seguiremos para su desarrollo, estructurados en varios pasos, serán los siguientes.

1. Investigar acerca de las capacidades de la lógica probabilística y del lenguaje de programación *Problog2*.
2. Comparar *Problog2* con otros lenguajes de programación lógica probabilística.
3. Desarrollar una de sus aplicaciones creando un sistema que reconozca actividades de seres humanos en vídeos de vigilancia, desarrollando este modelo en lenguaje *Problog2*.
4. Analizar los resultados del modelo a partir de las actividades que reconoce y obtener conclusiones acerca de si este lenguaje es realmente utilizable.

1.4 Marco regulador

En este apartado se hablará de la legislación aplicable al desarrollo del trabajo que se va a realizar. Entre las regulaciones que se deben tener en cuenta destacamos que este trabajo se encontraba sujeto a la Ley Orgánica de Protección de Datos 15/1999 (LOPD), la cual fue revisada por el Real Decreto 1720/2007, garantizando la seguridad de los datos personales de los ciudadanos a la hora de aplicarlos al sistema informático utilizado.

Al haber sido desarrollado este trabajo hasta el mes de junio de 2019, esta ley se vio afectada por la normativa europea RGPD (Reglamento General de Protección de Datos), aprobada el 14 de abril de 2016 y entrando en vigor el 25 de mayo de 2018. Esta normativa, que afecta a todos los países de la Unión Europea, hizo que en España se derogara la ley anterior y se aprobara la Ley Orgánica 3/2018 de Protección de Datos Personales y garantía de los derechos digitales (LOPD-GDD) [4], aprobada el 18 de octubre de 2018 y con vigencia desde el 7 de diciembre de 2018, a partir de la cual está sujeta desde entonces nuestro proyecto.

Entre los datos personales en los que se garantiza su seguridad destacan los siguientes.

- Datos que pueden identificar a una persona física.
- Datos que pueden identificar las tareas o el empleo que realiza una persona.
- Datos que pueden identificar rasgos personales de una persona.
- Datos que pueden identificar afinidades ideológicas de una persona.

Debido a que se realiza el trabajo en la Universidad Carlos III de Madrid, se recogerá la normativa de utilización de las aulas informáticas por los alumnos de primer y segundo ciclo de enseñanzas universitarias (Resolución 7/93 del 3 de marzo) [5]

Conclusiones regulación del Trabajo de Fin de Grado

Vistas las legislaciones presentes que afectan al desarrollo de este trabajo, no hay impedimentos para su realización siempre que se respeten las leyes descritas. Para la utilización de los datos de las personas en los vídeos de vigilancia bastará con no publicar su nombre, tratándolos como anónimos y diferenciándolos unos de otros mediante identificadores, para respetar sus datos personales y hacer un uso de ellos estadístico y analítico.

1.5 Estructura del documento

El documento está estructurado en nueve grandes capítulos diferenciados.

1. **Introducción**: contiene el contexto del que parte el documento, la motivación y los objetivos establecidos en él, además del marco regulador en el que se encuentra el proyecto.
2. **Estado del arte**: primer acercamiento que se hace al problema, estableciendo la visión general que se le da a cada una de las partes que tratamos en el trabajo además de algunas investigaciones que las usan. También se comparan distintas herramientas para elegir cuales usar para realizar el trabajo y para saber hacia dónde queremos llegar y desde el punto que partimos.
3. **Diseño y modelado del sistema**: contiene la explicación en detalle de la estrategia y diseño seguido para realizar el modelo, las transformaciones necesarias para su implementación, el desarrollo de los datos de la base de datos de imágenes utilizada y, por último, su implementación en el lenguaje *Problog2*.
4. **Análisis de los resultados**: estudia que se ha realizado sobre los resultados obtenidos en la ejecución del modelo, comparando distintos experimentos.
5. **Gestión del proyecto**: contiene la planificación y los costes asociados al trabajo realizado, junto al impacto socio-económico que tiene su elaboración.
6. **Conclusiones y líneas futuras**: aquí se recogen las consideraciones finales que se han tenido sobre el estudio realizado además de las líneas futuras a tener en cuenta sobre este proyecto.
7. **English competitions**: resumen del proyecto realizado en el que se demuestran las competencias adquiridas en inglés.
8. **Bibliografía:** lista con todas las fuentes de las que se ha extraído información para la realización del trabajo.
9. **Anexo**: aquí se recoge el código que se ha utilizado para la realización y ejecución del sistema modelado.

CAPÍTULO 2. ESTADO DEL ARTE

En este capítulo vamos a hablar del estado del arte en relación a nuestro trabajo. Se pondrá en contexto cada una de las partes que vamos a tratar, el punto en el que se encuentran actualmente con los avances que existen y la comparación entre las distintas posibilidades de elección para su uso. Posteriormente, sacaremos conclusiones de todo lo investigado y definiremos el punto del que partirá nuestra investigación.

2.1 Visión general

En este apartado presentaremos en qué punto se encuentran cada una de las partes que trataremos en nuestro trabajo, para establecer el contexto en el que nos encontramos y de lo que estamos hablando junto a su correspondiente explicación. Para ello diferenciaremos entre las siguientes partes: la lógica probabilística, el lenguaje de programación lógica probabilística Problog2 y su aplicación en vídeos de vigilancia.

2.1.1 Lógica probabilística

Antes de hablar de lo que entendemos como lógica probabilística haremos un primer acercamiento al tema hablando de en lo que consiste la lógica como concepto principal.

La lógica, en el contexto que nos ocupa [6], se trata de la ciencia que estudia el por qué de las cosas, los métodos y las razones que explican la verdad de las ideas, distinguiendo las correctas de las incorrectas. Como concepto general es fácil de entender, pero en ella surge una necesidad clara que hay que abordar y que hay que tener en cuenta, que es la incertidumbre.

La incertidumbre tiene una relevancia destacable en nuestro trabajo y por ello es importante tener claro cómo es su funcionamiento. Hay diversas formas de medir la incertidumbre [7]. La medida más antigua que existe y que la mayoría de la gente conoce por ser de las más intuitivas es la probabilidad. Esta consiste, tal y como dice Lindley [8], en lo siguiente:

> *"la única descripción satisfactoria de la incertidumbre es la probabilidad. Al decir esto se hace referencia a que toda afirmación incierta debe estar en la forma de una probabilidad; que varias incertidumbres deben ser combinadas usando las reglas de la probabilidad; y que el cálculo de probabilidades es adecuado para manejar todas las situaciones con incertidumbre. En particular, las descripciones alternativas de incertidumbre son innecesarias".*

Pero al igual que tiene defensores que apoyan esta idea, hay detractores que no apoyan su uso. Para el desarrollo de sistemas expertos existen muchas más medidas de la incertidumbre igual de válidas, como pueden ser las siguientes:

- El uso de la teoría de la evidencia desarrollado por Dempster-Shafer [9], que tiene diversas aplicaciones en Inteligencia Artificial.
- La lógica difusa, desarrollada por Lotfi A. Zadeh [10], que se basa en experiencias anteriores para dar valor a sus hechos, los cuales además de tomar valores numéricos, pueden tomar valores de un atributo definido previamente. Por ejemplo, en vez de decir que la altura de una persona es de 0.877 se puede reemplazar por el valor de "muy alta".
- Los factores de certeza, que consisten en asegurarnos la veracidad de algo, asegurándonos que es cierto lo que estamos tratando con un cierto grado dado, el cual será mayor si la seguridad que tenemos en la idea es mayor, y viceversa. Este método ha sido utilizado por ejemplo para el desarrollo del sistema experto **MYCIN** [11] [12], uno de los sistemas expertos pioneros más importante. Este sistema marcó un antes y un después y su uso en la medicina posibilita la realización de diagnósticos de infecciones en la sangre [13]. Su funcionamiento actual se basa en el uso de un motor de inferencia con usa serie de premisas simples (con más de 500 reglas) a las que el paciente debe contestar las preguntas que se le formulan con un sí o un no. El resultado del sistema presenta una serie de posibles enfermedades que el paciente podría tener ordenadas según la probabilidad de que ocurran, basándose en las respuestas que había dado. La probabilidad con la que se marca cada diagnóstico se basa en el uso de factores de certeza, que marcan el grado de creencia que se tiene en cada enfermedad. Aunque obtenía buenos resultados, su uso se basó exclusivamente en casos experimentales.

Todas estas opciones son aceptadas para tratar la incertidumbre, pero no se suelen usar por la complejidad que presentan en su entendimiento en comparación con la probabilidad, que será en la que nos centraremos en nuestra investigación.

Sin embargo, los modelos que se basan en la probabilidad tienen una dificultad clara destacable relacionada con su complejidad computacional. Esto es, el aumento del número de parámetros con los que estemos tratando, lo cual implicará que aumente considerablemente su complejidad, haciendo que la carga de trabajo en algunos casos sea casi imposible de tratar.

Por esta necesidad de tratar con la incertidumbre surge el concepto de lógica probabilística, que fue definido por primera vez por Nils Nilsson [14]. Esto consiste en que una serie de conclusiones adquieren un valor probabilístico de premisas predefinidas anteriormente y que se escogen al ser más probable que se cumplan que otras. Esto quiere decir que no tiene por qué ser verdad la solución propuesta, sino que será verdad en el mayor de los porcentajes presentados. Además de esto, crearon un método para tratar con la complejidad computacional que tienen los sistemas expertos basados en casos reales. Esta complejidad viene dada por la inconsistencia de ciertos valores proporcionados y el método se encarga de aproximarlos a valores consistentes para poder ser tratados.

Por ello el objetivo de esta lógica consiste en lidiar con los dos temas que estamos tratando: el uso de la lógica deductiva para obtener las conclusiones a partir de las premisas predefinidas combinado con la capacidad de tratar la incertidumbre que aparece en los problemas del mundo real. Esto nos lleva a definir la inferencia.

La inferencia [15], tal y como hemos dicho antes, consiste en sacar conclusiones a partir de premisas. Podemos diferenciar varias clases de inferencia, pero las más destacables que nos interesan son las siguientes:

- La inferencia **inductiva**, a través de la cual generamos normas generales a partir de premisas creadas por la información extraída del mundo observado.
- La inferencia **deductiva** que, al contrario que en la inductiva, conseguimos crear hipótesis o ideas a partir de normas generales ya definidas.

De esta manera, el razonamiento probabilístico lo obtendremos de los valores que se dan a ese conjunto de premisas, a partir de las cuales obtendremos la probabilidad de que ocurran las conclusiones. Una de las formas más utilizadas para representar este razonamiento como modelo probabilístico son las redes bayesianas [16], que consisten en generar nodos ordenados, cada uno de ellos haciendo referencia a unas premisas, en dirección a otros, que se comportarán como conclusiones extraídas de estas premisas. Las relaciones entre ellos estarán marcadas por sus conexiones, creando como se puede ver en la imagen inferior, padres e hijos. Esto hará que la probabilidad de ejecución de los hijos esté condicionada únicamente por sus parientes más cercanos, y más exactamente, por la probabilidad de que ocurran estos.

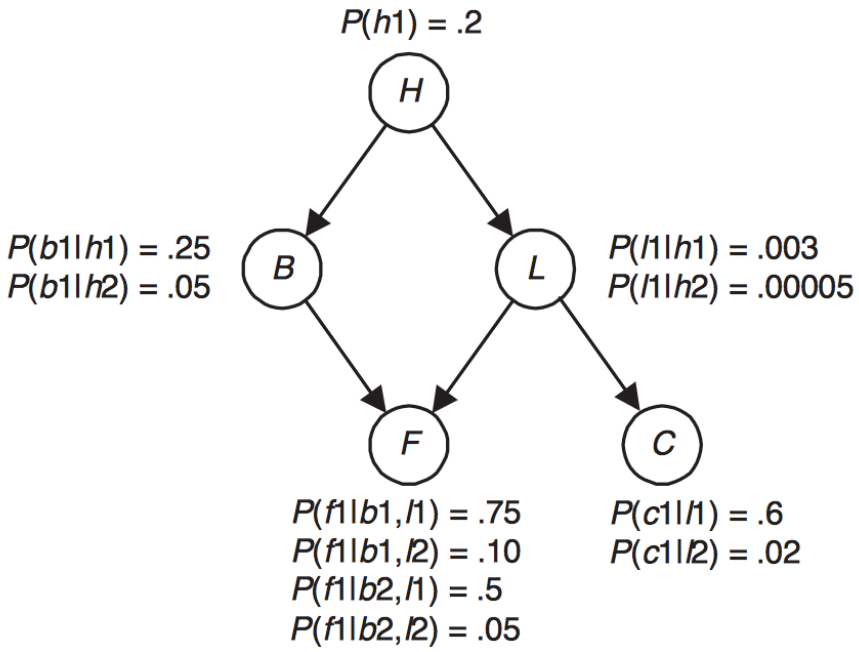

Ilustración 1. Ejemplo red bayesiana [17]

Un ejemplo claro de red bayesiana sería el Modelo Oculto de Markov, HMM (Hidden Markov Model) [18], el cual tiene la característica de que algunos de sus parámetros son desconocidos, y su objetivo es descubrirlos a partir de los estados conocidos y los parámetros que sí son observables. Estos estados cuentan con valores que representan la probabilidad de transición a otros estados o directamente hacia una salida.

Actualmente está en auge el uso de la lógica probabilística porque nos permite tratar con la incertidumbre, un problema a tener en cuenta para tratar con sistemas que se quieren utilizar en la vida real. Por ello, vamos a destacar algunos de los usos que tienen en diferentes investigaciones.

Aprendizaje lógico probabilístico

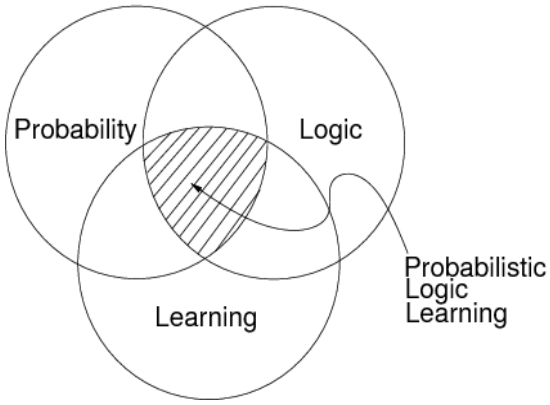

Ilustración 2. Aprendizaje Lógico Probabilístico [19]

El enfoque que le dan al aprendizaje probabilístico Raedt y Kersting [19] se basa en la combinación de los principios del razonamiento probabilístico junto a las representaciones lógicas y el aprendizaje estadístico, buscando la coherencia entre ellos. Las técnicas usadas se basan sobretodo en una perspectiva lógica o mediante el uso de la programación lógica inductiva y, gracias a esto, se ven las diferencias y similitudes entre los distintos formalismos presentados.

En su presentación diferencian entre modelos teóricos y pruebas teóricas. Los modelos teóricos especifican qué mundos son posibles a partir del uso de la lógica, mientras que las pruebas teóricas definirán, a partir de un objetivo marcado, aquellas pruebas que tendrán mayor probabilidad de que ocurran. Esta distinción proporciona la relación que existe entre las actualizaciones en las redes bayesianas y las gramáticas y, además de esto, información relevante acerca del aprendizaje. Por eso, el aprendizaje para estos dos formalismos se diferencia de la siguiente manera.

- El aprendizaje de los modelos teóricos se basa principalmente en interpretaciones.
- El aprendizaje de las pruebas teóricas se basa en la conexión y similitud que presentan entre ellas.
- Además de estas dos, el aprendizaje que se realiza con un punto intermedio entre los dos teoremas se basa en el recorrido seguido por el modelo.

Por último, los parámetros y la estructura de las lógicas probabilísticas que se han tenido en cuenta se han aprendido a partir de los principios del aprendizaje estadístico y de la programación lógica inductiva, de este modo la representación de los hechos será mucho más compacta a la vez que utilizable.

Sistemas expertos

En el punto anterior hablamos de MYCIN y su uso en el área de la medicina. En la actualidad en este mismo ámbito nos encontramos con el sistema GIDEON [20], usado mundialmente y que permite la detección de más de 300 enfermedades infecciosas con un porcentaje de acierto muy alto (94%). Sin embargo, al igual que ocurría con MYCIN, su uso se basa exclusivamente en ayudar al experto en su diagnóstico mejorando la exactitud, no para que tome la decisión por él de qué hacer.

Por otro lado, Shapiro demostró en uno de sus trabajos [21] que el uso de la programación lógica probabilística haciendo uso de la incertidumbre para el desarrollo de sistemas expertos de los que no se tiene la información completa de su modelo tiene mucho sentido. Esto es así porque la semántica simple con la que cuentan estos programas permite al experto ajustar las probabilidades para su ejecución según desee a partir del conocimiento que tiene del sistema, sin importar la metodología que siga este sistema para realizar la inferencia, por lo que es algo bastante útil.

2.1.2 Problog2

El lenguaje *Problog2* se trata de una extensión del lenguaje de programación lógica *Prolog,* por lo que antes de adentrarnos en él veremos en qué consiste este lenguaje en primer lugar y la relación que tiene con *Problog2*.

Lenguaje *Prolog*

El nombre de *Prolog* proviene de Programación Lógica [2] y es un lenguaje que se basa en la lógica de las relaciones entre dos objetos y en resolver problemas que presentan estos. Por ello, la programación en este lenguaje [22] se fundamenta en poder declarar hechos como verdades absolutas sobre los objetos que está tratando, definir una serie de reglas sobre ellos, tanto individualmente como en conjuntos relacionados y la capacidad de realizar consultas. A continuación, explicaremos como se declaran los elementos de un programa en *Prolog*.

Entre los elementos que tiene un programa de *Prolog* podemos diferenciar entre hechos y reglas. Los primeros son relaciones que siempre son ciertas. Se declaran con el nombre que se le atribuye a la relación y entre paréntesis todos los objetos que relaciona. Algunos ejemplos, como los declarados a continuación, serían que Miguel pertenece a la relación definida como ‘persona’, que Bob y Mary pertencen a la relación ‘hijo_de’ o que Tom, John y 26 pertenecen a la relación ‘diferencia_edad’.

Definiciones de hechos
persona(miguel).
Hijo_de(bob, mary).
Diferencia_edad(tom, □ohn, 26).

Tabla 1. Definición hechos en Prolog

En estos ejemplos podemos observar varios detalles. Las declaraciones que se le atribuyen a las relaciones son infinitas, pueden tener tanto un atributo individual como varios, según deseemos declarar, y estos a su vez pueden ser tanto cualitativos como cuantificativos. Además, como veremos a continuación, estos atributos también pueden declararse como variables, pero en el caso de que no lo sean, deberán empezar por minúscula para indicar que se tratan de variables constantes. Por último, la definición del significado de estos hechos es propia y el programa debe entenderla tal y como la entendemos nosotros. El significado de un hecho con un único atributo individual no tiene mucho misterio, pero en cambio, como en el ejemplo visto anteriormente, el hecho 'hijo_de' habría que saber especificar cual de los dos atributos es hijo de quién, porque en este caso, si Bob es hijo de Mary, el hecho de que Mary es hija de Bob no nos interesa ya que sería falso, o como el último ejemplo, donde el atributo numérico, según el hecho donde está declarado, se define como la diferencia de edad de los dos primeros atributos cualitativos.

Para dar forma a estos hechos se puede hacer uso de las reglas. Las reglas consisten en una función condicional que expresa que, si cada uno de los objetivos incluidos, que pueden tratarse de hechos o de alguna condición numérica, se cumplen, entonces el hecho que define la regla también será verdadero. La sintaxis utilizada para esto, junto a su expresión desde una visión lógica sería la siguiente:

Expresión	Ejemplo
Visión en Prolog	*padre(X, Y)* **:-** *masculino(Y), madre(Z, Y), pareja(X, Z).*
Visión lógica	$masculino(Y) \land madre(Z, Y) \land pareja(X, Z) \rightarrow padre(X, Y)$
Cláusula de Horn	$\neg masculino(Y) \lor \neg madre(Z, Y) \lor \neg pareja(X, Z) \lor padre(X, Y)$

Tabla 2. Diferentes formas de expresión reglas

Podemos observar como los objetivos incluidos en cada regla se representan como conjunciones entre ellos. De esto podemos deducir que su comportamiento se basa en el uso de cláusulas de Horn [23]. Como vemos en el ejemplo anterior, se crea una única cláusula incluyendo todos los literales, tanto el objetivo de la regla como los hechos que la componen, y se representan en forma de disyunciones. De esta manera, para que se cumpla la cláusula como mucho un único literal deberá ser cierto. Al estar todos los

objetivos de la regla representados como una negación, su comportamiento será el esperado, ya que, si no se cumple uno de esos hechos, la cláusula será cierta porque el hecho objetivo tampoco se cumplirá. En cambio, si se cumplen todos los hechos, al estar negados el único literal que será cierto será el del hecho objetivo de la regla.

Por otro lado, tal y como hemos dicho antes, se hace uso de variables para sustituir a los argumentos no conocidos. De esta manera, esas condiciones tratadas con variables, harán que se comprueben todos los hechos definidos para esa relación, como por ejemplo en el caso anterior: para la regla en la que se comprueba si X es el padre de Y, al estar definida como condición 'masculino(Y)', se comprobarán todos los hechos (con un solo argumento por cada comprobación) que hayamos definido previamente para este hecho y junto a las demás condiciones, se ejecutará la regla para ver si se cumple. De la misma manera, si en la definición de la regla tenemos alguna variable, en las condiciones de esa regla deberemos incluir el valor que adquirirá para el cual se cumplen todas las condiciones declaradas, y de esta manera, el hecho con el valor definido se sustituye por la variable y se incluirá en la lista de predicados que se cumplen en nuestro programa. Véase también que la definición de estas variables debe hacerse siempre en mayúscula, para indicar que se trata de una variable y no de un valor constante.

La estructura general del programa consiste en la mayoría de los casos en la siguiente: en primer lugar, declarar los hechos con sus argumentos válidos que sabemos que son verdad, los cuales adquieren un valor conceptual de datos de entrada del programa; después, una serie de reglas, cada una con sus condiciones, que se deben cumplir para generar nuevos hechos o cumplimentar las consultas realizadas; y por último, estas consultas que realizaremos para hacer las comprobaciones que deseemos acerca de los hechos que componen nuestro programa, y así comprobar si estas consultas son ciertas o no, o para que serie de hechos se cumple la consulta realizada.

La sintaxis de estas consultas es simple, basta con poner el símbolo de pregunta seguido de un guion ('?-') y a continuación la consulta que queremos realizar. Dependiendo de la consulta que queramos realizar, la respuesta puede cambiar: si la consulta realizada es un hecho simple sin variables de por medio la consulta se corresponde a si ese hecho es válido o no; en cambio, si incluimos alguna variable en la consulta realizada, haciendo que en este caso la consulta responda con los valores que puede adquirir esta variable, haciendo el hecho definido válido. Ejemplos reales de estas consultas con sus posibles respuestas son los que vemos a continuación:

Consulta	Respuesta posible a la consulta.
¿- adulto(X)	X = Miguel; X = Bob; X = Tom.
¿- adulto(Miguel)	yes.
¿- edad(Bob, X)	X = 32.

Tabla 3. Ejemplos de consultas en Prolog

En el primer ejemplo vemos que la consulta que se realiza son los valores que adquiere una variable en el conjunto de adultos, consultando quiénes de todos los argumentos establecidos cumplen este hecho, por lo que la respuesta será todas aquellas personas que sean adultas, tantas como haya o queramos mostrar. Al contrario de este ejemplo, en la segunda vemos que se realiza una consulta de un hecho sin variables, haciendo una pregunta directa a la cual el programa responderá si ese hecho consultado es verdadero o falso con la dupla de valores booleanos *yes*/*no*. También se puede hacer una consulta condicionada como se ve en el último ejemplo, donde se pregunta la edad de todas aquellas personas (ya que se trata de una variable) que estén en la base de hechos, pero con la condición de que se llamen 'Bob'. En este caso seguramente sólo contemos con una única respuesta, pero si tenemos varias personas que se llamen de la misma manera se podrían mostrar las edades de todas ellas.

De esta manera la deducción que sigue *Prolog* para resolver problemas dependiendo del caso en el que se encuentre se puede encasillar en tres marcos distintos:

- Según una **lógica proposicional**: caso en el que sólo hay una única respuesta y sólo se utilizan proposiciones sin valores variables.
- Según una **lógica relacional**: en este caso ya contamos con variables además de constantes, relaciones y cuantificadores.
- Según una **lógica funcional**: como el caso anterior, pero añadiendo funciones en sus reglas.

Por otro lado, la estrategia que sigue *Prolog* para resolver las consultas que se realizan es la siguiente:

1. Partimos de que la consulta está compuesta por varios hechos. Esto significa que para confirmar la consulta se deben cumplir todos y cada uno de ellos. Todos estos hechos que se tienen que confirmar se almacenarán en una lista.
2. Para confirmar estos hechos se buscarán reglas que logren satisfacer cada uno de ellos. De la misma manera, para satisfacer que se cumplen estas reglas, habrá que confirmar también los hechos que las componen. Por ello, si una regla confirma uno de los hechos de la lista, este hecho se sustituirá por todos aquellos hechos que compongan la regla.
3. Esto se repetirá continuamente hasta llegar a un hecho que pertenece a la lista que se sabe que es verdadero por estar así declarado. En este caso, este hecho se eliminará de la lista, lo que significa que se ha confirmado.
4. Si todos los hechos que componen la lista se pueden confirmar de la misma manera, al eliminarse cada uno de estos hechos de la lista acabaremos con un conjunto vacío, lo que significa que hemos confirmado la consulta realizada. En cambio, si hemos recorrido todos los hechos y reglas de nuestro programa y sigue habiendo hechos que pertenecen a la lista, implicará que la consulta no es verdadera.

Esta estrategia ejemplificada sería la siguiente: contamos con una consulta que para que sea válida tendrá que confirmar los tres hechos que la componen (X_1, X_2, X_3). El primero de ellos se confirma ya que está definido (X_2, X_3). El segundo de ellos para que sea válido se debe cumplir cierta regla (Y :- y_1, y_2, y_3), por lo que la lista cambiará sustituyendo el segundo hecho por todos los hechos que componen la lista, los cuales si se cumplen, confirmarán este hecho también (y_1, y_2, y_3, X_3). Finalmente, todos los hechos que quedan en la lista se confirman uno a uno, por lo que al final quedará un conjunto vacío. Al quedar un conjunto vacío, la consulta será válida.

Esta estrategia de resolución tiene el nombre de *SLD (Selective Linear Definite)* [24]. Podemos destacar que al recorrerse cada uno de los hechos y las reglas según aparezcan, importará el orden en el que estén estas definidas, ya que la última que aparezca será la última que se ejecutará. Por esta razón es recomendable siempre escribir en primer lugar los hechos que sabemos que son verdaderos y seguidamente las reglas que se ejecutarán con más frecuencia, para así aumentar el rendimiento durante la búsqueda. Además, esta búsqueda se realizará recorriendo estas reglas y hechos que se disponen en forma de árbol y su forma de recorrerlo será en **profundidad**, donde se recorrerá cada rama del árbol hasta encontrar el conjunto vacío (éxito) o un camino sin salida (fracaso) y después volver hacia atrás (mediante **backtracking**) a la siguiente rama. Así, siempre se recorrerá todo el árbol y obtendremos todas las posibles respuestas válidas a nuestra consulta.

En general, lo más representativo de este lenguaje lo hemos descrito anteriormente, pero además tiene muchas más capacidades [25], como por ejemplo la creación de listas, que hacen el programa mucho más robusto. Algunas de estas funcionalidades las veremos al hablar de su extensión probabilística *Problog2*, que es la que realmente nos importa saber cómo es su funcionamiento.

Lenguaje Problog2

Como ya hemos dicho, la mayoría de programas que tratan la lógica probabilística tienen una alta complejidad en cuanto al nivel de interacciones que deben realizar para resolver el problema. Además, la mayoría de las situaciones de la vida real no son siempre ciertas, por lo que debemos tener en cuenta la incertidumbre que pueden presentar. Teniendo esto en cuenta, presentamos un lenguaje que tiene en cuenta estas limitaciones y del que veremos a continuación sus características, que es el lenguaje *Problog2*.

Problog2 es la versión mejorada del lenguaje *Problog* y es una extensión del lenguaje que hemos desarrollado en el punto anterior. Se basa sobretodo en programación lógica, distribuciones semánticas y modelos probabilísticos y gráficos. El modelado de *Problog2* [26] se puede resumir diferenciando dos puntos de vista:

- Punto de vista **probabilístico**, donde se definen una serie de probabilidades sobre ciertos valores verdaderos de un subconjunto de átomos del programa. Estos se definen como *p:: fact*, donde 'p' es la probabilidad y 'fact' el hecho a tratar.
- Punto de vista **lógico**, que deriva los valores verdaderos de los átomos restantes usando un razonamiento similar al de *Prolog*. Para estos casos se usan las cláusulas definidas anteriormente.

Como podemos observar, la diferencia principal con *Prolog* es que *Problog2* usa un razonamiento probabilístico para dar las respuestas. Sus respuestas no se basan en contestar con algo que será siempre verdadero o falso, sino que proporciona una probabilidad, comprendida entre 0 y 1, de que lo que está contando es cierto. De esta manera, para definir un hecho que es cierto en casi la mayoría de los casos, pero no en todos, se le puede establecer una probabilidad un poco más baja que 1 y en el caso contrario en el que un hecho casi nunca es válido, se le puede atribuir una probabilidad un poco más alta que 0 y así, tratar fácilmente la incertidumbre. En adicción a que cada cláusula cuente con una probabilidad de que cada cláusula tenga una probabilidad, todas ellas son mutuamente independientes [27], es decir, que se cumpla o no un hecho no influye en la probabilidad de otro, a no ser que este esté condicionado por el primero (mediante reglas). En el caso de contener variables, si tienen sus valores definidos también serán independientes.

Los aspectos más destacables que nos permite desarrollar este lenguaje [26] son los siguientes:

- Permite la inferencia marginal, es decir, calcular las probabilidades marginales de las consultas realizadas a partir de ciertas evidencias.
- Hace posible el aprendizaje de parámetros del programa partir de datos en forma de interpretaciones parciales.
- Resuelve problemas de decisión teóricos.
- Permite el muestreo de todos los mundos posibles (sampling).
- Interacciona con el lenguaje Python [26].

La estructura de *Problog2* en comparación con la que ya hemos visto de *Prolog* es muy similar al tener un funcionamiento parecido. Las diferencias más destacables las veremos a continuación.

En este lenguaje, tanto los hechos como las reglas, al tener la capacidad de definirse a partir de una probabilidad de que sean válidos, contarán al principio de su definición con un número que indica el grado de probabilidad que presenta esa definición. Se representa con el número seguido de dos puntos (::), tal y como se ve a continuación.

Definiciones de hechos y reglas con probabilidad
0.95:: adulto(X) :- persona(X), edad(X, E), E>18.
0.25:: persona(tom).
persona(john).
1/6:: tirada(X,1); 1/6:: tirada(X,2); 1/6:: tirada(X,3); 1/6:: tirada(X,4); 1/6:: tirada(X,5); 1/6:: tirada(X,6) :- dado(X).

Tabla 4. Definición hechos y reglas en Problog2

Como vemos en los ejemplos anteriores, en el primer caso se trata de una regla, que la adicción de la probabilidad implica que, cuando las definiciones de la regla sean válidas, el objetivo de la regla lo será también en un 95% de los casos. Por otro lado, el segundo ejemplo indica que en el 25% de los casos Tom pertenecerá al conjunto de 'personas'. A pesar de esta capacidad, podemos operar sin probabilidad como hacíamos en *Prolog*, como se ve en el tercer ejemplo en el que no incluimos la probabilidad, que se traduce como que ese hecho será siempre válido. También podría haber una asignación de diferentes objetivos en una regla para el caso en el que se cumpliera, como podemos ver en el último ejemplo, donde en vez decidir si el objetivo se cumple o no, habría que decidir entre un conjunto de posibles asignaciones; si todas ellas suman una probabilidad de '1' siempre se cumplirá una de ellas, si no lo hicieran, podría darse el caso de que la regla no activara ninguna evidencia nueva.

Al igual que se produce una asignación de probabilidades en hechos y preguntas, lo verdaderamente importante es su asignación en las consultas realizadas. Por ello, se definen dos nuevos hechos para tratar las consultas y las evidencias.

Las consultas se realizan gracias a la definición de **query**(Q), donde Q será la consulta a realizar. El programa ante esta definición calculará la probabilidad de que esta consulta ocurra. La manera en la que se realiza la veremos a continuación. Las consultas además pueden verse influenciadas por algunas evidencias definidas. Son de la forma **evidence**(E), donde E será la evidencia que para nuestro problema queremos que sea siempre verdadera. Esto hará que algunas probabilidades, dada esa evidencia, cambien. Esto se puede comprobar mediante consultas.

Veamos un ejemplo sobre esto:

```
1   % Regla 1
2   0.25:: bola(B,1); 0.25:: bola(B,2); 0.25:: bola(B,3); 0.25:: bola(B,4) :- coger_bola(B).
3   % Regla 2
4   suma(S) :- bola(1,A), bola(2,B), S is A+B.
5
6   % Hechos
7   coger_bola(1).
8   coger_bola(2).
9
10  % Consulta
11  query(suma(_)).
```

Ilustración 3. Ejemplo programa en Problog2

La finalidad de este programa es sacar dos bolas de una bolsa, las cuales están marcadas con números del 1 al 4 y nuestra consulta consiste en saber la probabilidad de la suma de ambos números. El resultado de la consulta, será el siguiente:

```
suma(2):        0.0625
suma(3):        0.125
suma(4):        0.1875
suma(5):        0.25
suma(6):        0.1875
suma(7):        0.125
suma(8):        0.0625
```

Ilustración 4. Resultados programa ejemplo Ilustración 3

Podemos ver que, al realizar la consulta, en vez de incluir el valor numérico que queremos comprobar incluimos '_'. Esto hace que la consulta realizada mostrará todas las posibilidades de respuesta que tiene. Las que tienen una probabilidad de 0 (como sumar 1) no se muestran y, además, la suma de todas ellas será 1. Este tipo de consulta se conoce en *Problog* como **probabilidad marginal**. Si a este programa le incluimos la evidencia de que la primera bola sacada será siempre 4, como se ve en la imagen de a continuación, el resultado de la consulta cambiará viéndose afectado por esta evidencia.

```
1  % Regla 1
2  0.25:: bola(B,1); 0.25:: bola(B,2); 0.25:: bola(B,3); 0.25:: bola(B,4) :- coger_bola(B).
3  % Regla 2
4  suma(S) :- bola(1,A), bola(2,B), S is A+B.
5
6  % Hechos
7  coger_bola(1).
8  coger_bola(2).
9
10 % Evidencias y consulta
11 evidence(bola(1,4)).
12 query(suma(_)).
```

Ilustración 5. Ejemplo programa Problog2 con evidencias

```
suma(5):        0.25
suma(6):        0.25
suma(7):        0.25
suma(8):        0.25
```

Ilustración 6. Resultados programa Ilustración 5

La consulta cambiará tanto para las respuestas obtenidas como para las probabilidades de estas, ya que al verse condicionada con que la primera bola es 1, los hechos suma(2), suma(3) y suma(4) ya no son posible, y las probabilidades restantes consistirán únicamente en escoger la segunda bola, tal y como ya está definido el programa. Este tipo de consulta se conoce como **probabilidad condicionada**.

Un último estilo de consulta sería **MPE inference** (Most Probable Explanation), que consiste en buscar el mundo en el que es más probable que sea válida la consulta realizada.

En cuanto a la parte probabilística de este lenguaje todo lo descrito anteriormente es lo más destacable a tener en cuenta, todo lo demás está comprendido en la parte lógica, lo cual su estructura y funcionamiento es similar a como se realiza en *Prolog*.

La inferencia que se realiza en *Problog2* [28] [29] para obtener las probabilidades de sus consultas, explicada por pasos, es la siguiente:

- Se reúnen todas las definiciones que influyen en la consulta realizada, buscándolas en el árbol de pruebas SLD.
- Una vez localizadas, se representan todas las pruebas como una DNF (Disjunctive Normal Form).
- Gracias a un script ya desarrollado en el lenguaje, se traduce el DNF en un BDD (Binary Decision Diagrams).
- Finalmente, se calcula la probabilidad a partir de este BDD.

La probabilidad de una consulta se puede transformar en la probabilidad de que se cumpla ese DNF, que es simplemente una transformación de las probabilidades del SLD, pero eliminando redundancias y compactándolo todo en una única fórmula, con forma de disyunción de conjunciones, separando cada una de las pruebas. Así la probabilidad será igual a la probabilidad de que se cumpla al menos una de estas pruebas. La complejidad de la fórmula DNF es *NP-hard*.

Al tratarse de esta manera podemos pensar en asumir la probabilidad como la suma del producto de todas las pruebas involucradas en la consulta, pero no se puede plantear de esta manera ya que cada prueba por separado no representa mundos mutuamente exclusivos, siempre contará con alguna prueba que se repita y, por tanto, se compute su probabilidad redundantemente. Para resolver este problema, se podría mejorar las conjunciones de cada prueba añadiendo los literales negativos que no se tienen en cuenta en la consulta, para excluir aquellos mundos en los que dicha prueba ya se haya computado. Este problema se conoce como *'disjoint-sum'* y su resolución es *#P-hard* [30].

Los BDD's son una forma de representar las fórmulas DNF de una forma más compacta. Se trata de un árbol de decisión en el que los nodos redundantes son eliminados. Cada nodo de este árbol son los hechos probabilísticos de nuestro programa y cada uno de ellos tiene dos salidas, una positiva y otra negativa. El camino que sigue cada uno dependerá de cómo esté definido el programa, por lo que su camino positivo tendrá la probabilidad establecida para ese hecho mientras que el negativo será el complemento de su probabilidad, ya que no influirá en la resolución de la consulta. Los caminos llegan a dos nodos finales, con valores de '1' o '0', dependiendo de si se ha llegado a la consulta válida o no. Veamos un ejemplo de esto:

$$P_S(q) = P\,((f1 \wedge f2 \wedge f3 \wedge f5) \vee (f1 \wedge f2 \wedge f4))$$

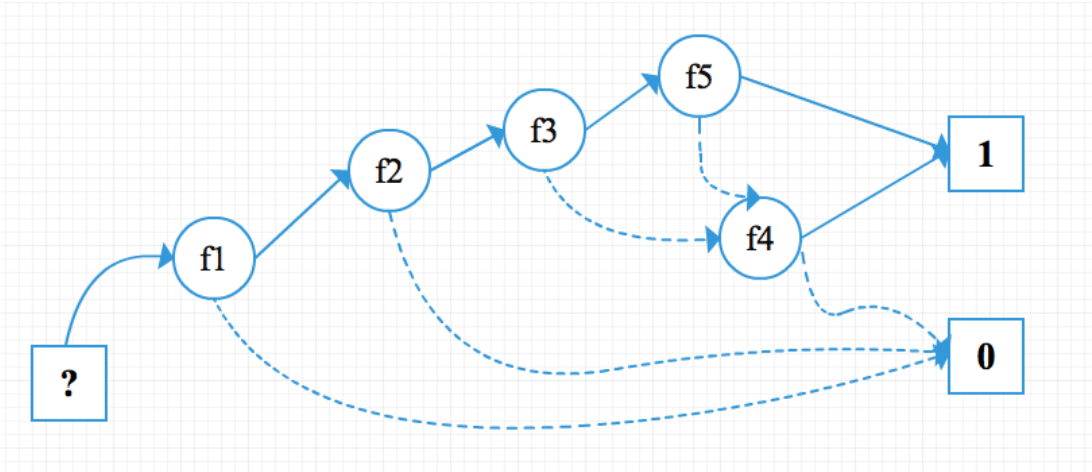

Ilustración 7. Diagrama de Decisión Binaria (BDD)

En el diagrama anterior vemos como el DNF se compone de dos disyunciones y su BDD sería de esa manera. Por ejemplo, en el caso de que f1 no se cumpliera, al pertenecer a ambas cláusulas, iría directo al nodo terminal no válido (0), al igual que ocurre con f2. Si f3 no se cumpliera, aún podría cumplirse la segunda disyunción establecida, por lo que el camino del nodo será en dirección a este nodo de la segunda disyunción.

Como se puede observar, es fácil obtener la probabilidad de las consultas de esta manera ya que no es necesario hacer backtracking para recorrer el árbol. Por esta razón, entre las ventajas que tienen estos diagramas destaca su simplicidad y su flexibilidad para plantear consultas complejas, además de ser muy eficiente ya que permite consultas con miles de pruebas.

Herramientas de uso en Problog2

Problog2 como herramienta autónoma, permite su ejecución de diferentes formas, según el uso que le queramos dar a su motor de ejecución. En este apartado vamos a diferenciar cada una de ellas y más adelante explicaremos como utilizarlas para su ejecución.

- **Inferencia en *Problog* estándar**: en este modo *Problog* ejecuta el modelo que se le proporciona y computa la probabilidad de sus consultas, mostrándolas como output de la ejecución. Este modo es el que siempre se ejecuta por defecto. Este cálculo de probabilidades se puede realizar mediante **SDD** (Sentencial Decision Digram), que es más completo, o con **d-DNNF** (Deterministic Decomposable Negation Normal Form).
- **Muestreo**: en este modo *Problog* selecciona un conjunto de individuos que se ejecutarán como muestra de la población para obtener la probabilidad de las consultas realizadas reduciendo la carga computacional de trabajo, ya que sólo se ejecutará parte del total. Se puede seleccionar cuantos muestreos realizar.
- **Muestreo basado en inferencia**: en este caso se realizarán muestreos de la misma manera que en el caso anterior, pero se ejecutarán según una cantidad de muestras o un tiempo de ejecución y el output del sistema será una media de todas las probabilidades obtenidas.
- **Most Probable Explanation**: este caso, conocido como **MPE** y como su propio nombre indica, se computa el mundo más probable en el que todas las consultas y evidencias descritas sean verdad, obteniendo como output todos los outputs, ya sean válidos o nulos, junto a la probabilidad de que se cumplan todos ellos.
- **Aprendizaje mediante interpretaciones**: en este modo, conocido como **lfi** (learning from interpretations), en vez de calcular la probabilidad de que se cumplan las consultas realizadas a partir de las probabilidades de los hechos conocidos como en los casos anteriores, se hace la acción contraria. En este caso se calculan las probabilidades de ciertos hechos de los que se desconoce su probabilidad a partir de ciertas evidencias declaradas. La probabilidad de estos hechos inicialmente puede ser totalmente desconocida o inicializarse con algún valor.

En este caso las probabilidades no se calcularán arbitrariamente, sino que se establecerán según las evidencias declaradas, las cuales harán que la probabilidad de estos hechos aumente o disminuya según el grado de implicación que tengan en las evidencias. Esta herramienta es interesante para saber cuánto de importante son ciertos hechos en un modelo o establecerlos si se desconocen.

- **Decisión teórica**: este modo se conoce como DTProblog, que es una extensión de *Problog* que difiere de este en que no cuenta con evidencias ni consultas, en los hechos hay que decidir la probabilidad con la que se ejecutarán, mediante el uso de **decision**, y, por último, los átomos se declaran mediante su utilidad, que indica su contribución para la ejecución final del programa. Esto último se declara mediante el uso de **utility**.
- **Inferencia MAP**: similar a MPE, pero en este caso es necesario que los hechos a los que se le asignan las probabilidades sean consultas explícitas (formuladas mediante una query).
- **Modo explicativo**: en este modo se reformula el modelo creado con las pruebas de cada consulta por separado y con la probabilidad de que se ejecute, además de la probabilidad de cada una de estas consultas, sumando las probabilidades de todas las pruebas anteriormente declaradas. Así, tal y como se indica, se explica cómo se ejecuta el programa.
- **Otros modos**: *Problog* cuenta con muchas más alternativas de ejecución, como puede ser grounding, donde se ejecuta el programa sin variables, shell interactiva para ejecutar y realizar modelo en tiempo real, representación de redes bayesianas con programas externos y servidores web.

La forma de ejecución para usar cada una de estas herramientas las explicaremos más adelante.

2.1.3 Vídeos de vigilancia: Reconocimiento e interpretación

La base principal de nuestro trabajo se centra en el reconocimiento de las actividades que llevan a cabo ciertas personas, tanto individual como conjuntamente. Para poder realizar este reconocimiento necesitaremos obtener estas actividades de algún tipo de base de datos. Por ello, haremos uso de vídeos de vigilancia en los cuales se realicen este tipo de actividades que queremos reconocer, De esta manera innovaremos en este ámbito introduciéndolo en la inteligencia artificial para hacer de él un uso mucho mas avanzado.

El reconocimiento de actividades que ocurren en vídeos de vigilancia permitirá un uso más práctico de estos, ya que se tendrá la capacidad de detectar las actividades que se realizan, en el momento exacto en el tiempo en el que ocurren y, además, si tenemos un buen modelado del problema, detectar quién está realizando esas actividades. El coste económico de estos avances se vería reducido considerablemente ya que se obtendrían respuestas casi instantáneas. A su vez, si contamos con una gran memoria de almacenamiento, nos permitirá administrar grandes cantidades de esta información en el

tiempo que, dependiendo del campo de uso de estas actividades, puede ser algo a tener muy en cuenta.

La pregunta entonces es cómo obtenemos estas actividades de los vídeos de vigilancia. Para realizar esta labor se hace uso del Deep Learning [31], una clase de técnica de aprendizaje novedosa que permite el desarrollo de abstracciones de alto nivel para obtener de ellos datos procesados de una manera mucho más sencilla. Con las técnicas antiguas con las que se creaban estos sistemas de reconocimiento de actividades se precisaba un dominio del campo bastante alto además de una sofisticada labor de programación para procesar estos datos, cosa que se evita gracias al Deep Learning.

El método de uso es simple. A partir de una red de neurona artificial con varios niveles el aprendizaje se realiza tomando la información del primero de estos niveles, en el siguiente nivel se realiza una combinación de esta información de manera más compleja para hacerla más completa y así en los sucesivos niveles de la red, hasta hacer la información lo suficientemente completa como para aprender lo que deseamos. Esta metodología es de lo más eficaz, ya que obtiene un porcentaje de éxito muy elevado incluso cuando se realiza mediante un aprendizaje no supervisado.

En nuestro caso no vamos a hacer uso del Deep Learning ya que obtendremos directamente las actividades que ocurren en los vídeos de vigilancia directamente de una base de datos, por lo que en el trabajo nos centraremos únicamente en el reconocimiento de actividades a partir de estas actividades ya procesadas.

2.2 Lenguajes de programación lógica probabilística

En este apartado nos centraremos en ver las capacidades que presentan algunos de los lenguajes de programación lógica probabilística que existen y compararlos entre ellos, para ver cuáles son las ventajas de su uso con respecto a otras y las deficiencias que presentan.

Para poder abordar la comparación entre los distintos lenguajes de programación lógica probabilística, podemos observar un ejemplo donde se presenten varias de estas herramientas en las que se explique su funcionamiento con el problema que tratan, como es el caso de [27], donde se realiza un algoritmo de aproximación para calcular las probabilidades en un problema típico de la vida real, como es el descubrimiento de enlaces en redes biológicas.

Estas redes biológicas se relacionan entre sí mediante relaciones probabilísticas independientes, por ello en este caso el uso de *Problog2* es eficiente, ya que computa probabilidades mutuamente independientes entre los hechos que relaciona, a diferencia de otras herramientas como pueden ser **PRISM** y **PHA**.

El problema que presentan tanto PRISM como PHA es que sólo tratan probabilidades sobre hechos específicos, sin el uso de variables, además de que se excluye la posibilidad de que ciertas combinaciones de hechos sean verdaderas simultáneamente. Esta restricción facilita la computación del algoritmo, pero limita su aprendizaje considerablemente.

Por otro lado, aparte de estas dos herramientas, también existe otra llamada '**Probabilistic Datalog'** (**pD**) [32], que se trata de una extensión del lenguaje de Datalog en la que se incluyen las negaciones. Su funcionamiento es muy similar a Problog, ya que no impone estas restricciones de simultaneidad que imponen las otras herramientas, pero también tiene ciertas limitaciones que hacen que no sea preferible su uso. En primer lugar, el poder computacional de pD no es Turing completo, es decir, no alcanza el que tiene la máquina de Turing universal para resolver problemas. Por otro lado, posee una mayor complejidad de entendimiento, que hace que su uso sea más complejo, tanto para el programador como para el usuario al que esté dirigido. Por último, su uso tiene una perspectiva más ligada a las bases de datos que a las consultas de probabilidad lógica, haciendo que pierda el sentido su uso en este ámbito.

El problema de las aplicaciones usadas para sucesos en la vida real relacionadas con la programación lógica probabilística es su dificultad en cuanto al entendimiento y al manejo para el usuario. Esto *Problog* lo soluciona gracias a sus probabilidades independientes, que permiten describir las relaciones del problema fácilmente.

La semántica de *Problog* se extiende en que permite el cálculo de probabilidades en cláusulas, no sólo en hechos predefinidos como hacen las demás herramientas. En estos casos hay que tener precaución porque si dos cláusulas son verdaderas, no implica que tengan que ser independientes una de la otra, como en el caso de que una deriva de la otra: la probabilidad de la segunda siempre estará condicionada con lo que le ocurra a la primera, por lo que habrá que tenerlo en cuenta. Otros sistemas como PRISM y PHA este problema lo solucionan mediante restricciones que no permiten que se ejecuten estos casos especiales donde varios hechos se quiere que sean simultáneamente verdaderos, haciendo que el programa ejecute las probabilidades como una suma de productos en forma de disyunciones. Quizá estas restricciones no se traten como una característica del todo inválida, pero en realidad sí. Esto es porque, al realizar estas restricciones, se provoca que el programa pierda parte de la capacidad que tenía para aprender, haciendo que su rendimiento y eficacia disminuya.

Finalmente, comparando *Problog2* con otras herramientas para su uso en el ámbito del reconocimiento de actividades, este lenguaje contará con la capacidad de inferir basándose en mundos cerrados. Esto significa que *Problog2* basa su reconocimiento en entornos en los que no hay variables sin definir, por lo que siempre elegirá sus respuestas a partir de las que ya tiene en memoria. Esta capacidad hace que el programa sea más eficiente ya que acota sus posibilidades de ejecución y hace que su tiempo de respuesta sea mucho más rápido. Además de esto, generará representaciones simbólicas que permitirán al usuario que utilice el programa con este lenguaje de programación entender más fácilmente su uso ya que utilizará una semántica entendible, a diferencia del resto de lenguajes que utilizan representaciones sub-simbólicas, con semántica que en la mayoría de los casos no entiende el usuario que la utiliza.

2.3 Estudio bases de datos de actividades

En este apartado nos centraremos en diferentes bases de datos que existen en las que se recogen datos sobre actividades, ya sean de vídeos de vigilancia o de otros escenarios, y cómo tratan cada una de ellas los datos, para ver si podemos hacer uso de ellas para nuestro problema.

Además, se hablará de cómo obtienen los datos que las componen y la manera de utilizarlos para el modelo que queremos crear.

PIROPO DATASET

En esta base de datos llamada PIROPO (People in Indoor ROoms with Perspective and Omnidirectional cameras) [33] se almacenan datos relacionados con la grabación de vídeos en dos habitaciones distintas. Estos vídeos se componen de más de 100.000 *frames*, una cantidad considerable a tener en cuenta para su uso en el aprendizaje de modelos, pero no tanto para el reconocimiento de actividades.

En los vídeos que se graban se representa a personas haciendo acciones tales como caminar, sentarse o permaneciendo quietas. Por ello, los valores que se almacenarán en la base de datos a partir de estas situaciones serán únicamente valores que representan las coordenadas en las que se encuentra la persona que se está grabando junto al framen en el que aparece. Si en un mismo instante de tiempo aparecen varias personas se almacenará también sus coordenadas, de la siguiente manera.

(40, 12, 20)

(1800, 20, 30, 22, 30)

Véase que las coordenadas se establecen no según la posición en la que aparezcan en la habitación, sino la posición en la que estén captadas por el vídeo. En el primer ejemplo se observa como una persona se encuentra en las coordenadas (12, 20) en el *frame* 40, mientras que en el segundo hay dos personas, en el (20,30) y (22,30) en el *frame* 1800.

Cada uno de estos conjuntos de datos se almacenarán en filas de un archivo *.csv*. Para el uso que queremos darle a nuestro modelo, esta base de datos no es del todo interesante, ya que tendríamos que transformar estos datos para que el lenguaje de programación que utilicemos los reconozca. Además, carecen de significado ya que aportan poca información para nuestro problema.

CAVIAR DATASET

CAVIAR (Context Aware Vision Image Active Recognition) es un proyecto en el que, de la misma manera que en la base de datos anterior, se recogen imágenes grabadas de distintos escenarios con la idea de procesar lo que ocurra en estas imágenes para así obtener una serie de datos que poder almacenar en su base de datos. Para realizar esto los vídeos que utilizan están divididos en dos conjuntos, uno grabado en un laboratorio y el segundo en un centro comercial.

El proceso que siguen para tratar los vídeos a partir de los *frames* de cada uno de ellos y procesar e inferir los datos se recoge en su página web [34] y su uso es público.

Los tipos de datos que se recogen en los vídeos procesados de CAVIAR se centran básicamente en el reconocimiento de personas andando, saludándose entre ellos, abriendo ventanas, entrando y saliendo de tiendas, peleándose, desmayándose y dejando objetos en lugares públicos. Además del reconocimiento de estas acciones, cada actividad deberá contar con otros datos como la orientación y las coordenadas que tienen estas personas para ponerlas en situación. Estos últimos datos se procesarán según las posiciones de los píxeles de la cámara de vigilancia, por lo que las coordenadas de cierta persona se codificarán según sea la posición del vídeo en el que esté siendo grabado, y de igual manera ocurrirá con el grado de orientación.

Cada uno de los vídeos grabados pertenece a un estilo de actividad de reconocimiento, por lo que en un solo conjunto de *frames* no se recogerán todas las actividades. De entre los vídeos del primer conjunto de datos grabados en el laboratorio se distinguen a su vez otro conjunto de 28 vídeos, en los que se especifica que actividades ocurren en cada uno. Se distinguen estas actividades:

- Tres vídeos donde se recoge que una persona está andando.
- Seis vídeos donde las personas están buscando ciertos objetos.
- Cuatro vídeos donde las personas se encuentran sin realizar movimientos (descansando o desmayadas).
- Cinco vídeos en los que las personas dejan objetos.
- Seis vídeos donde las personas se saludan, andan juntas y se separan.
- Por último, cuatro vídeos donde las personas discuten.

Como podemos observar, esta base de datos es muy completa para su uso en nuestro modelo ya que trata las características que queremos tomar como datos de entrada de este además de una amplia variedad en las actividades que se reconocen. El único problema es que el segundo conjunto de vídeos trata únicamente escenarios en los que aparecen y desaparecen personas andando y estas características se quedan escuetas para utilizarlas en nuestro modelo.

Otra característica de esta base de datos a tener en cuenta es que dentro de la propia base de datos cuenta con estos datos transformados en lenguaje de programación lógica, por lo que si utilizamos esta base de datos en nuestro modelo no hará falta procesar los datos de entrada de los vídeos de vigilancia, característica muy a tener en cuenta.

2.4 Conclusiones obtenidas

Una vez investigadas algunas de las herramientas que se pueden utilizar para el tratamiento de la programación lógica probabilística en modelos de reconocimiento de actividades, el uso de Problog2, reafirmando lo que comentábamos en la motivación de este documento, es la más acertada.

Problog2, a diferencia del resto de herramientas, proporciona una gran flexibilidad para realizar consultas complejas, ya que no tiene las restricciones o limitaciones con las que cuentan las demás. Como ya hemos visto, estas restricciones se hacen para aumentar sus capacidades de ejecución y reconocimiento siendo mucho más exhaustivos, pero de esta manera se ve afectado su rendimiento ya que el tiempo de respuesta será mayor.

Como la meta será poder realizar un sistema de reconocimiento de actividades que proporcione las respuestas lo más rápido posible, *Problog2* al tratar representaciones simbólicas su uso es preferible, ya que ocurrirá lo contrario que con el resto, su capacidad se verá afectada pero su tiempo de respuesta será mejor.

Para compensar la capacidad de reconocimiento de la herramienta elegida bastará con que el problema a tratar no sea muy complejo y, como dijimos en la motivación, la elección de actividades que ocurren en vídeos de vigilancia nos ayuda con este problema ya que estas actividades están acotadas por lo que a la hora de reconocer la actividad que se realiza el recorrido empleado no será muy costoso.

En conclusión, usaremos *Problog2* ya que el tiempo de respuesta a la hora de reconocer actividades no es muy alto y es la herramienta que más se asemeja a una representación simbólica de problemas y para nuestro caso, buscamos sobretodo eso, que las respuestas obtenidas se puedan representar para que cualquiera pueda utilizar el modelo de reconocimiento.

Como nuestro modelo se va a tratar de una versión simplificada de reconocimiento de actividades donde sólo queremos demostrar las capacidades de ejecución que podría tener *Problog2* en un sistema real, el problema de eficacia mencionado anteriormente no lo tendremos ya que nos basaremos en reconocer un reducido número de actividades presentes en la base de datos de vídeos de vigilancia utilizada en nuestro caso.

En cuanto a la base de datos de actividades en vídeos de vigilancia a utilizar no cabe duda que se tratará de CAVIAR, por las características que ya henos mencionado, tanto por su completitud como por sus datos ya procesados en lenguaje de programación lógica *(.pl)* para su uso directo sin necesidad de transformaciones.

CAPÍTULO 3. DISEÑO Y MODELADO DEL SISTEMA

En este capítulo se habla del diseño y el modelado que se ha seguido para desarrollar el sistema de reconocimiento de actividades a partir de vídeos de vigilancia según los criterios establecidos que se quieren abordar, así como la implementación de este propio sistema en lenguaje de *Problog2*.

Además de esto, se explica el uso de los datos de la base de datos de vídeos de vigilancia utilizada que trataremos como entradas para hacer posible la ejecución del sistema, el reconocimiento de actividades que ocurren en ella y, posteriormente en el siguiente capítulo, su análisis completo.

3.1 Diseño del sistema y estrategia

En este punto se recoge el diseño del sistema realizado con su fundamentación, así como la estrategia que seguirá el programa para resolver el problema.

3.1.1 Fundamentación

El problema que se quiere abordar en nuestro sistema como ya hemos mencionado anteriormente consiste en el reconocimiento de actividades a partir de imágenes de vídeos de vigilancia. Esta funcionalidad será bastante útil para el uso de este sistema en la vida cotidiana.

Si tratamos de reconocer las acciones de ciertas personas en vídeos de vigilancia con una finalidad de mantener el orden público será interesante poder reconocer acciones tales como que cierta persona se mueva o que en un momento determinado esté discutiendo con alguien; en el caso de tratarse de una tienda de venta de productos podría ser interesante reconocer si cierta persona está robando o simplemente observando sus productos; por otro lado, si se tratara de vídeos de vigilancia de una exposición de arte quizá nos interese conocer la distancia en tiempo real a la que se encuentren las personas que visitan la galería de las obras expuestas, para conocer si se está respetando la distancia de seguridad establecida, distancia que también podemos definir según nuestro criterio.

Siguiendo la estrategia que empleará nuestro sistema podríamos llegar a diseñar cualquiera de estas situaciones que hemos detallado, pero en nuestro desarrollo únicamente nos vamos a centrar en el primer ejemplo descrito, donde la elección de las actividades a reconocer se basará sobretodo en los movimientos que realicen las personas que aparecen en los vídeos de vigilancia a partir de los datos que proporciona la base de datos de CAVIAR.

3.1.2 Estrategia a seguir

La estrategia que seguiremos para modelar este problema se basará en el uso de la programación lógica y más concretamente centrándonos en el apartado probabilístico de esta. La razón de esto es porque al tratarse de un modelo extrapolable para su uso en la vida real, nunca sabremos si tanto las actividades modeladas como datos de entrada de cada *frame* de los vídeos como las actividades que son reconocidas por el programa son

ciertas en todos los casos, siempre hay una cierta probabilidad de que no ocurran. El uso en este caso de la incertidumbre es más que necesario. De esta manera podremos tratar los casos en los que no esté del todo claro si son ciertos o no, aportando a las condiciones en las que aparezcan de una probabilidad acorde a estas condiciones de validación.

Por ello, enmarcando el problema en la programación lógica probabilística, la estrategia que se seguirá paso a paso será la siguiente:

1. **Diseño del modelo** que se creará para posteriormente realizar nuestro sistema.
2. **Implementación del modelo** diseñado usando el lenguaje de *Problog2,* con las reglas y hechos necesarios para su funcionamiento.
3. **Elección de los datos de la base de datos de CAVIAR** que el programa enmarcará para reconocer las actividades a partir de ellos.
4. **Tratar estos datos como entradas** del programa siendo hechos evidentes.
5. **Realizar las consultas** pertinentes según las actividades que queramos comprobar si se cumplen a partir de estos datos.
6. Procesar la petición a partir de estas consultas, **recorriendo las reglas** que influyen en estas consultas en los *frames* de tiempo correspondientes.
7. **Mostrar los resultados tras la ejecución del modelo.** En estos resultados se especificará además para qué persona o personas se cumple o no la consulta junto al momento temporal en el que ocurre.
8. **Analizar los resultados obtenidos comprobando si se cumplen o no las consultas realizadas**.
9. **Analizar las distintas experimentaciones y sacar conclusiones de ellas**.

Los pasos del 3 al 7 se repetirá tantas veces como ejecuciones queramos realizar para cada conjunto de *frames*. Si tenemos en cuenta probabilidades en los datos de entrada o en las reglas a ejecutar, las respuestas obtenidas no indicarán si son ciertas o no, sino la probabilidad con la que serán válidas.

Para el desarrollo de esta estrategia el punto del que partiremos para desarrollarla será teniendo en cuenta las siguientes condiciones.

- Usaremos una plantilla base de predicados que nos ayudará en el reconocimiento de actividades en el tiempo, llamada Event Calculus que veremos a continuación.
- Las reglas del modelo se desarrollarán según el entendimiento que tenemos nosotros de las condiciones que deben tener para que se ejecuten cada una de ellas.
- Las actividades que compondrán nuestro modelo estarán basadas en la base de datos de CAVIAR, que es la que utilizaremos, ya que la inclusión de actividades distintas a las proporcionadas por CAVIAR provocaría que no se ejecute nunca esta actividad o la necesidad de cambiar los datos de la base de datos para que se generen estas actividades (labor que realizamos con uno de las actividades, que veremos más adelante).

Una vez claras todas estas condiciones, partiremos de ellas para el desarrollo por nuestra cuenta del modelo a generar para reconocer actividades.

3.1.3 Diseño

A partir de la estrategia de ejecución definida, que se debe seguir para realizar el reconocimiento de las actividades en nuestro problema, el diseño deberá estar ligado a esta para que funcione tal y como queremos. Por esta razón tendremos distintos tipos de predicados a tener en cuenta.

Todos los predicados de nuestro modelo se definirán a partiendo de la base del uso de Event Calculus [35]. Esto es un conjunto de predicados lógicos que se utilizan sobretodo para el reconocimiento de los efectos que provoca una acción a lo largo del tiempo. Estos efectos de las acciones se reconocen gracias a los cambios que experimentan de un momento a otro en el tiempo. Además, estos efectos, según el evento o los eventos que se estén realizando, podrán cambiar a lo largo del tiempo. Este funcionamiento es interesante ya que nos permitirá modelar nuestro problema basándonos en estos predicados para la definición de diferentes valores en el tiempo de nuestras actividades.

Las reglas que utilizaremos basándonos en las que componen a Event Calculus las veremos a continuación. Debido a su funcionamiento, en el que se distinguen acciones y sus efectos, deberemos distinguir para la generación de nuestro modelo entre dos conjuntos diferentes para clasificar la información, que son 'eventos' y 'fluentes'.

Los **eventos** consisten en acciones que ocurren prefijadas, que pueden ser tanto internas como externas al sistema. En nuestro caso trataremos como eventos de nuestro sistema al conjunto de actividades proporcionadas por CAVIAR. Estos datos serán a partir de los cuales realizaremos el reconocimiento de actividades ya que los trataremos como predicados y entre todos ellos distinguimos los siguientes valores que pueden obtener:

- Las **STA (Short Term Activities)**, que serán las actividades de las que están compuestas la base de datos de vídeos de vigilancia que estamos utilizando. Estas serán las actividades *active*, *inactive*, *walking* y *running.*
- La apariencia que adquieren estos entes en el vídeo, que sirve para especificar si las personas o los objetos están siendo 'trackeados' en el vídeo o no, en el caso de que desaparezcan de la imagen. En este caso serán los predicados *appear* y *disappear*.

Los **fluentes** serán aquellos eventos que ocurren pero que no son fijos ya que pueden cambiar a lo largo del tiempo debido a distintas acciones que ocurran sobre ellos. Esta definición se puede observar claramente que hace referencia a los datos de salida de nuestro programa, los cuales se definen mediante el uso de **LTA (Long Term Activities**). Estas actividades serán las consultas que queremos comprobar si se cumplen o no en nuestro programa y para ello se definirán unas series de reglas que veremos en el punto siguiente.

Existe un caso especial que no sigue las reglas anteriormente definidas, ya que se codifica como un fluente, pero es tratado como un dato de entrada del sistema. Estos son las

coordenadas y la orientación de todos aquello que se identifica en el vídeo, tanto personas como objetos. La razón por la que ocurre esto es porque como cualquier otra LTA, puede cambiar su valor a lo largo del tiempo, por lo que debe tratarse como fluente y, por otro lado, es necesario su uso como dato de entrada ya que se necesita conocer para identificar los distintos LTA que ocurren en el vídeo.

Una característica a tener en cuenta es que los datos de entrada del sistema son mutuamente excluyentes, para cada tipo de predicado definido anteriormente sólo puede ocurrir uno de ellos en cada momento de tiempo definido. En cambio, los datos de salida del programa (LTA) no lo son, por lo que en el mismo instante de tiempo puede haber más de una actividad ocurriendo simultáneamente, siempre y cuando se cumplan las condiciones de ejecución para cada una de ellas. Tiene sentido que esto sea así ya que no puede darse el caso en el que alguien este andando y corriendo a la vez en el mismo instante de tiempo, pero, sin embargo, puede ser factible que alguien este saludando y andando junto a otra persona en el mismo instante de tiempo

A continuación, veremos los distintos predicados con los que contaremos en cada conjunto de datos según el tipo del que se traten.

Event Calculus

Los predicados que compondrán este bloque serán aquellos que nos permitirán relacionar los diferentes tipos de actividades a tratar. Al ser un sistema que reconoce actividades en el tiempo, estos predicados serán una versión parecida a la que compone Event Calculus, pero más reducida. La declaración de los predicados y su breve funcionamiento es la siguiente:

Predicados	Significado
holdsAt(F = V, T)	indica que el valor del fluente F será V en el tiempo T.
broken(F = V, Ts, T)	indica que el valor de V del fluente F se ha quebrado entre los puntos de tiempo Ts y T.
initially(F = V)	indica que el valor del fluente F es V en el tiempo inicial (0).
iniciatedAt(F = V, T)	indica que en T se ha iniciado un fluente F con un valor de V.
terminatedAt(F = V, T)	al contrario que el anterior, en T termina un fluente F con valor de V.
happensAt(E, T)	reconoce que ocurre un evento E en el tiempo T.

Tabla 5. Predicados de Event Calculus

El funcionamiento que siguen en resumen es que *happens* se encarga de indicar que un evento ocurre, el efecto de estos eventos se recoge mediante el uso de *iniciatedAt* y *terminatedAt* y, por último, los valores de los fluentes se asignan mediante *initially* y *holdsAt*.

Inputs del sistema

En este apartado, como su propio nombre indica, estarán aquellas actividades que utilizaremos como datos de entrada del sistema, que serán aquellas actividades de las que estará compuesto CAVIAR. Esta base de datos cuenta con muchos tipos de datos distintos, pero en los que nosotros nos centraremos son en los de la siguiente tabla.

Cabe destacar que los objetos que se identifiquen, únicamente pueden adquirir el valor de STA de *inactive*, los demás valores sólo pueden ser adquiridos por personas. También decir que CAVIAR no detectará el predicado *abrupt* ya que no está incluido en su base de datos junto a los demás STA, pero es interesante su inclusión para el reconocimiento de algunos LTA.

Predicados	Significado
active(id)	Indica que id está activo, es decir, no está realizando movimientos abruptos con el cuerpo en el lugar donde se encuentra.
inactive(id)	Indica que id está inactivo. Esto significa que se mantiene quieto en donde se encuentra.
walking(id)	Indica que id está andando.
running(id)	Indica que id está corriendo.
abrupt(id)	Indica que id está realizando movimientos abruptos.
appear(id)	Indica que id aparece en el vídeo por primera vez. En este momento se le asigna el identificador que le caracterizará.
disappear(id)	Indica que id desaparece de la escena y, por lo tanto, se libera la asignación identificativa con la que contaba. Esto ocurrirá cuando id deje de aparecer en el vídeo o cuando este finalice.
coord(id) = (x, y)	Se trata de un fluente e indica que id cuenta con unas coordenadas definidas como los píxeles del vídeo de x e y.
orientation(id) = A	Fluente que indica que id cuenta con una orientación que forma un ángulo de A con el eje de abscisas (eje x) del vídeo.

Tabla 6. Predicados para entradas del modelo

Outputs del sistema

En este apartado se encuentran los predicados que nuestro sistema deberá reconocer a partir de los datos de entrada que le proporcionemos. Serán los que deberemos analizar y comprobar que su funcionamiento sea el idóneo en las reglas desarrolladas más adelante. Los predicados que reconocerá nuestro sistema estarán centrados en la interacción entre personas, junto a la interacción de ellos con objetos, que serán los de la siguiente tabla.

Predicados	Significado
leaving_object(P, Obj)	Indica que la persona P deja el objeto Obj en algún lugar.
meeting(P_1, P_2)	Indica que están interaccionando entre ellas P_1 y P_2, saludándose o hablando.
moving(P_1, P_2)	Indica que las personas P_1 y P_2 están moviéndose juntas.
fighting(P_1, P_2)	Indica que las personas P_1 y P_2 están discutiendo entre ellas e incluso peleándose.
person(id)	Indica que id le pertenece a una persona.
close(ID_1, ID_2, Limite)	Indica que la distancia que existe ente los entes a los que les pertenece ID_1 e ID_2 (previsiblemente personas) no excede el límite establecido (Limite).
distance(ID_1, ID_2) = D	Completando el predicado anterior, este calcula el valor de la distancia entre ID_1 e ID_2 mediante la distancia euclídea a partir de sus coordenadas.

Tabla 7. Predicados salidas del modelo

Las consultas que se realizarán serán únicamente las cuatro primeras, los demás predicados también se tratan como fluentes pero nuestro modelo no busca que se realice un reconocimiento sobre ellos, están definidos para completar estos predicados principales.

Predicados auxiliares

En este grupo estarán aquellos predicados necesarios para hacer consistentes nuestras reglas, haciéndolas más intuitivas y jerarquizadas. La lista de todos ellos es la siguiente:

Predicados	Significado
negate(Fact)	Simplemente indica que el hecho Fact estará negado.
leaving_object_mult(P, Obj)	Indica que una inicialización recursiva de el predicado *leaving_object*
moving_mult(P_1, P_2)	Indica que una inicialización recursiva de el predicado *moving*
fighting_mult(P_1, P_2)	Indica que una inicialización recursiva de el predicado *fighting*.

Tabla 8. Predicados auxiliares del modelo

La razón por la que utilizaremos estos predicados la veremos en el apartado 3.4.

3.2 Estructura y modelo del sistema

A partir de los predicados definidos en el apartado anterior, en este punto recogeremos la estructura que seguirá el programa, con alguna de las definiciones de sus reglas junto al funcionamiento de Event Calculus que permite ejecutar el programa correctamente. Las reglas definidas estarán escritas en lenguaje lógico, en el punto 3.4 definiremos cómo se deben implementar en lenguaje *Problog.*

Es importante destacar que el modelo diseñado está realizado como un modelo determinista. Su funcionamiento en *Problog2* es el correcto, pero en este caso no cuenta con ninguna probabilidad definida, por lo que simplemente se ejecutará como si se tratara de un modelo de *Prolog*. En la sección de Análisis explicaremos qué deberemos hacer para tratar la incertidumbre en este modelo.

3.2.1 Estructura del sistema

Como mencionamos en alguno de los puntos descritos anteriormente de cómo es la organización de un programa *Problog2*, el nuestro seguirá exactamente la misma estructura:

- En primer lugar, tendremos los hechos que trataremos como los datos de entrada de nuestro programa obtenidos por la base de datos de CAVIAR. En cada ejecución del programa elegiremos cada conjunto de hechos que queremos ejecutar y más adelante veremos cómo se definen cada valor que adquieren.
- En segundo lugar, aparecerán las reglas que formarán el modelo en el que nos basaremos para inferir las actividades que queremos reconocer. Cada regla contará con las condiciones necesarias para que se cumpla su función y además su disposición se hará basándose en Event Calculus. En el siguiente apartado veremos alguna de estas reglas y para acceder a todas ellas podemos ir al apartado de Anexo al final del documento.
- Por último, contaremos con las consultas que queremos realizar para inferir las actividades de reconocimiento.

3.2.2 Definición de las reglas utilizadas

La base de todas las reglas que utilizaremos para definir nuestro modelo cuenta con los predicados de Event Calculus, por ello en primer lugar veremos cómo es su funcionamiento. A partir de ellos se definirán tanto los eventos como los fluentes que utilizaremos para cada una de nuestras reglas, que veremos a continuación.

Definiciones para Event Calculus y STA

Las reglas de los predicados de Event Calculus son los siguientes.

$$\textbf{holdsAt}(F = V, T) \leftarrow$$
$$\textbf{initially}(F = V) \wedge$$
$$\textbf{negate}(\textit{broken}(F = V, 0, T)$$

holdsAt$(F = V, T) \leftarrow$

iniciatedAt($F = V, T_s) \wedge$

negate$(broken(F = V, T_s, T)$

Vemos como holdsAt, el predicado que utilizaremos para comprobar si se cumple una actividad o no, será verdadero cuando se inicie la actividad en algún momento determinado y desde ese momento, hasta el tiempo actual en el que nos encontremos, no se haya quebrado. De igual manera, si esta actividad se inicia en el momento 0 se utiliza la primera regla.

Si se cumple que haya algún fluente que esté broken en un intervalo de tiempo será porque ha terminado esa actividad en algún momento dentro de ese intervalo de tiempo o porque se ha iniciado en ese intervalo la misma actividad, pero con distinto valor, en ese caso consideramos que la actividad se ha quebrado.

broken$(F = V, T_s, T) \leftarrow$

terminatedAt($F = V, T_f) \wedge$

$T_s < T_f < T$

broken$(F = V_1, T_s, T) \leftarrow$

iniciatedAt($F = V_2, T_f) \wedge$

$V_1 \neq V_2 \wedge$

$T_s < T_f < T$

Las reglas de iniciatedAt y terminatedAt se definirán según las actividades que se quieran reconocer. Dentro de cada una de ellas contaremos con una serie de condiciones además del uso de happens. Este último predicado está compuesto de eventos y es el que se encarga de definir como estarán escritos los datos de entrada.

happensAt$(running(id1), 280)$

happensAt$(appear(id2), 5000)$

El significado de estas definiciones es que la persona id1 está corriendo en el *frame* 280 y que la persona id2 aparece por primera vez en el vídeo en el *frame* 5000. Esto se realiza tanto para los STA como para appear/disappear.

holdsAt$(coord(Id2) = (20,45), 10500)$

holdsAt$(orientation(Id2) = 120, 10500)$

La orientación y las coordenadas, que también son datos de entrada, pero se definen de otra manera, se generarían como podemos ver abajo, donde decimos que en el *frame* 10500 la persona id2 se encuentra en las coordenadas (20, 45) formando un ángulo de 120 grados con el eje X.

Definiciones para LTA

La definición de la actividad **leaving_object** será la siguiente:

initiatedAt(*leaving_object(P, O)* = true, *T*) ←

happens(*appear(O), T)* ∧

happens(*inactive(O), T)* ∧

holdsAt(*close(P, O, 30)* = true, *T)* ∧

holdsAt (*person(P)* = true, *T)*

terminatedAt(*leaving_object(P, O)* = true, *T*) ←

happens(dis*appear(O), T)*

Aquí podemos observar que para que se inicie la actividad de dejar un objeto por parte de una persona se debe cumplir que en ese mismo momento aparezca el objeto y se encuentre inactivo (si está inactivo se podrá tratar de un objeto) y además que la distancia entre la persona y el objeto sea pequeña, en este caso 30. Sabremos que el identificador P pertenece a una persona gracias al fluente ***person***, que se activará siempre y cuando se detecte que ese id está realizando eventos relacionados con personas, en este caso estar activo, andando, corriendo o moviéndose abruptamente.

Esta funcionalidad terminará cuando la persona desaparezca, al igual que ocurre con leaving_object, que se lanzará su terminación cuando se deje de detectar el objeto en el vídeo, es decir, se lance el evento disappear. Este caso sólo cuenta con una única inicialización, por lo que deberán tener un trato especial que veremos en el siguiente punto.

En cuanto al cálculo de la distancia simplemente necesitaremos una regla que calcula la distancia a la que se encuentran ambos id a tratar. Esta distancia se calculará mediante la distancia euclídea de sus coordenadas y se comprobará si supera o no el límite que hemos establecido.

holdsAt*(close(ID1, ID2, Limite)* = true, *T)* ←

holdsAt*(distance(ID1, ID2, Distancia)* = true, *T)* ∧

Distancia < *Limite*

holdsAt(*distance(ID1, ID2)* = Dist, *T)* ←

holdsAt(*coord(ID1)* = *(X1, Y1), T)* ∧

holdsAt(*coord(ID2)* = *(X2, Y2), T)* ∧

$Dist = \sqrt{(X2 - X1)^2 + (Y2 - Y1)^2}$

Las demás iniciaciones y terminaciones siguen una tendencia parecida a las de leaving_object. Las actividades de **meeting** se iniciarán cuando los identificadores tratados sean personas y además, al menos uno de ellos esté activo o inactivo y el otro que no corra ni se mueva abruptamente, es decir, no puede ser que ambos estén andando, como mucho sólo uno de ellos. Además de esto la distancia entre ambos tiene que ser 25, más cercana que la anterior. Esta actividad terminará debido a una larga serie de condiciones, por lo que se crearán en total seis reglas. Estas condiciones será que alguno de ellos esté corriendo, moviéndose abruptamente o desaparezca, que ninguno esté ni activo ni inactivo o que la distancia entre ellos sea mayor a 25. Un ejemplo de este último sería.

terminatedAt(*meeting* (P_1, P_2) = true, *T*) ←

holdsAt (*person*(P_1) = true, *T)* ∧

holdsAt (*person*(P_2) = true, *T)* ∧

holdsAt(*close*(P_1, P_2, *25)* = false, *T)*

Si no declaramos en ningún momento ninguna condición donde se de por hecho que el id que estamos tratando es una persona deberemos comprobar que se trata de una directamente. Esto ocurre cuando no se declara ningún caso o cuando se declara inactive, que también puede tratarse de un objeto.

Para el caso de la actividad **moving**, es muy parecido al caso anterior. La iniciación de esta actividad, que es única, se producirá cuando ambas personas estén andando, su distancia sea menor a 34 y la orientación que forman la diferencia de sus orientaciones no supere los 45 grados. Esta última condición es necesaria para diferenciar aquellos casos donde las personas se encuentren cerca una de la otra mientras andan, pero la orientación que llevan es completamente contraria, por lo que no están andando juntos.

Para finalizar esta actividad podrá ocurrir que una de las personas esté andando, pero la distancia con la otra sea mayor al límite establecido, que simplemente no se cumpla la distancia entre ellas sin importar lo que hagan, que uno de ellos esté corriendo, moviéndose abruptamente, activo o desaparezca de la escena y, por último, que el ángulo que formen sea mayor a 45.

inticiatedAt$(moving(P_1, P_2) = \text{true}, T) \leftarrow$

happens($walking(P_1), T) \wedge$

happens($walking(P_2), T) \wedge$

holdsAt($close(P_1, P_2, 34) = \text{true}, T) \wedge$

holdsAt ($orientation(P_1) = Or_1, T) \wedge$

holdsAt ($orientation(P_2) = Or_2, T) \wedge$

$|Or_1 - Or_2| < 45$

Por último, **fighting** se iniciará si al menos una de las personas involucradas realiza movimientos abruptos, la distancia euclídea entre ellos es menor a 44 y la otra persona no está inactiva. Esto quiere decir que precisaremos del STA abrupt en la base de datos así que lo crearemos de forma manual para su análisis posterior.

terminatedAt$(fighting(P_1, P_2) = \text{true}, T) \leftarrow$

negate(happens($abrupt(P_1), T)) \wedge$

negate(happens($abrupt\ (P_2), T)) \wedge$

holdsAt ($person(P_1) = \text{true}, T) \wedge$

holdsAt ($person(P_2) = \text{true}, T)$

Su finalización, como podemos ver arriba en un ejemplo, ocurrirá cuando al menos una de las personas esté andando, corriendo, moviéndose abruptamente, desaparezca o esté inactiva, además de que no se produzca en ninguno de los dos movimientos abruptos o que la distancia entre ellos independientemente de el evento que realicen, sea mayor al límite de 44.

Como ya hemos dicho, el modelado de todas las reglas utilizadas en el programa podemos verlas en el anexo del documento.

3.3 Modificaciones necesarias

En este apartado comentaremos las modificaciones que se deben realizar en el modelo realizado para que su ejecución sea la idónea usando el lenguaje de *Problog2* junto a los predicados de reconocimiento de eventos de Event Calculus. Estos cambios se dividen en dos grandes apartados, la transformación de los hechos negados para su uso válido y el control de las inicializaciones.

3.3.1 Tratamiento de hechos negados

En nuestras reglas como hemos podido comprobar se hace uso de la negación de los átomos probabilísticos. Esta negación en el lenguaje de *Problog* no es elemental, ya que influirá que el que el hecho pertenezca a la base de conocimiento o no.

Como nuestro sistema hace uso de la lógica probabilística, el simple uso de la negación mediante los operadores '\+' o 'not' provocará que no se cumpla nunca ese hecho, tenga la probabilidad que tenga. Su uso se hace únicamente para los hechos que no pertenecen a la base de conocimiento y, por lo tanto, al no pertenecer y negarlo, su probabilidad será de 1, como debe ser. Para el caso en el que pertenezca a la base de conocimiento, la probabilidad que deberá tener el hecho negado será el complemento de la que tiene sin negar, de la siguiente manera:

$$P\big(negacion(q)\big) = 1 - P(q)$$

Esta labor habría que realizarla mediante el uso de ***problog_not(Fact)***, incluido en las librerías de *Problog.* El problema de este predicado es que no es capaz de tratar los hechos que no pertenecen a la base de conocimiento, complementariamente a lo que ocurre con los operadores, así que para su uso habrá que crear una nueva regla, llamada ***negate$_1$(fact),*** la cual se active cuando se cumpla alguno de los predicados descritos.

$$negate_1(F) \leftarrow not\ F$$

$$negate_1(F) \leftarrow problog_{not(F)}$$

De la misma manera, habrá que realizar esto para átomos derivados, que generen otras reglas, por lo que de igual manera se puede hacer uso del predicado (perteneciente a Problog) llamado ***problog_neg(Goal)***, que funciona de igual manera que el anterior, por lo que se podrá crear un segundo conjunto de reglas llamados ***negate$_{goal}$(Goal)***, con problog_neg y el uso de '\+ Goal', de la siguiente manera:

$$negate_2(G) \leftarrow not\ G$$

$$negate_2(G) \leftarrow problog_{neg(G)}$$

Esta modificación es necesaria para el correcto funcionamiento del programa, pero finalmente no hemos realizado su implementación ya que no es necesaria. La razón de esto es porque para la versión utilizada de *Problog* en nuestro programa (2.1.0.36) ya se trata este problema con la negación de manera uniforme, por lo que no son necesarias las distinciones sintácticas de las primeras versiones de *Problog.*

3.3.2 Control de inicializaciones

Dependiendo del número de inicializaciones que tenga la consulta que se está realizando, podremos realizar diferentes modificaciones en la forma de ejecutarse para hacer más eficaz su comportamiento.

Múltiples iniciaciones.

Al realizar una consulta de una actividad en un *frame* determinado, mediante el uso de holdsAt, vemos que como dice la regla, se comprueban todas las iniciaciones que se realizan de esta actividad desde el tiempo 0 al *frame* que se está comprobando. Como en este caso contamos para la actividad consultada con varias iniciaciones que se pueden disparar en diferentes momentos en el tiempo, cada vez que hagamos una consulta de este tipo se creará una fórmula DNF con tantas disyunciones como iniciaciones haya, para la elección exacta de la probabilidad de la consulta realizada.

Este sería su comportamiento básico, pero tiene un problema, y es el caso en los que una iniciación se consulte redundantemente y provoque un coste computacional alto. Si se trata de dos o tres iniciaciones quizá el coste no sea del todo considerable, pero si hay un número mayor y el número de comprobaciones en el tiempo que se realiza es bastante alto, seguramente sea un problema a tener en cuenta.

Por ello en estos casos para evitar esta redundancia se realiza una técnica para consultar la consulta que estamos tratando pero exactamente en el tiempo anterior (*holdsAt(F = V, (T-1))*) y comprobamos como las iniciaciones y las terminaciones afectan a su probabilidad para, a partir de ella, asumir como debe ser la misma en el tiempo T.

Para realizar esta labor es necesario que el sistema reciba los *frames* de vídeo ordenados en el tiempo, porque sino esta técnica será defectuosa.

Una vez hecho esto, se generarán las diferentes consultas haciendo esta suposición y veremos como a medida que pasa el tiempo la probabilidad de que se cumpla una actividad, si no hay ninguna terminación de por medio, irá aumentando gracias a esta transformación. Tiene sentido ya que, aunque pueda ser que no se cumpla para todos los casos, si se ha iniciado una actividad en algún tiempo determinado y en el siguiente no ha habido ninguna condición que haga que esta actividad termine, para el tiempo posterior en el que nos encontramos su probabilidad aumentará con respecto al anterior.

Imaginemos que tenemos dos iniciaciones de alguna actividad con una probabilidad de 0.8 y 0.5 cada una. Si en un *frame* posterior la probabilidad de que se inicie en el esa actividad tiene que tener en cuenta esas dos iniciaciones, su probabilidad será mayor que tratándolos por separado, como podemos ver en su fórmula DNF:

$$P(X) = P(init_1 \vee init_2) = P(init_1) + P(init_2) - P(init_1 \wedge init_2)$$
$$= 0.8 + 0.5 - 0.8 \times 0.5 = 0.9$$

Esto ocurre de igual manera con la terminación, que irá disminuyendo a medida que pase el tiempo si no se realiza ninguna iniciación.

Única iniciación.

En este caso trataremos aquellos reconocimientos de actividades en los que sólo cuentan con una iniciación. En estos no importa el momento en el que se realice la consulta, al contar con una única iniciación durante toda la ejecución su probabilidad no cambiará nunca. Esto es un problema ya que si está iniciación se ha realizado con una probabilidad alta, siempre que se realice esta consulta obtendremos la misma probabilidad, incluso cuando ya no se realice la actividad. De la misma manera si la probabilidad de iniciación es baja, sus consultas siempre tendrán una probabilidad baja, a pesar de la posibilidad de que en un momento posterior si que se cumple la actividad.

Por ello generamos una nueva regla donde iniciemos múltiples actividades para aquellas que tengan una única iniciación. Se activará cuando detecte que se cumple cierta actividad con holdsAt y así poder modificar su probabilidad a lo largo del tiempo según ocurra que finalice (bajando su probabilidad) o siga ocurriendo (aumentando su probabilidad). La regla será de la siguiente manera:

$$\textbf{iniciatedAt}(\textit{leaving\_object\_multi(P,Obj)} = \text{true}, \textit{T}) \leftarrow$$
$$\textbf{holdsAt}(\textit{leaving\_object(P, Obj)} = \text{true}, \textit{T})$$

3.3.3 Creación del STA abrupt

Como ya hemos comentado, el uso del STA abrupt es muy útil para la generación del LTA fighting. Sin su uso habría que generar fighting a partir de los STA walking, active e inactive, y de esta forma quizá no sea todo lo preciso que queremos. El STA abrupt tiene un significado de que la persona que adquiere este STA está realizando movimientos abruptos en cierto punto de tiempo. Si se realiza este evento por parte de alguien ante otra persona, además de las otras condiciones que ya hemos comentado en puntos anteriores, tiene sentido que se genere la actividad de fighting.

Por ello para generar los STA abrupt realizaremos una modificación en la base de datos de CAVIAR para que aparezca este STA, ya que CAVIAR no lo tiene implementado. La manera de hacerlo de la forma más exacta es accediendo a diferentes archivos pertenecientes de la base donde se generan datos acerca de la situación en la que se encuentran, diferente a la actividad que se realiza. Esto se encuentra en el archivo de *SituationGrp.pl* y la regla que generaremos para que aparezca abrupt será la siguiente.

$$\textbf{happens}(\textit{abrupt(P1), T}) \leftarrow$$
$$\textbf{happens}\ (\textit{walking(P1), T}) \wedge$$
$$\textbf{happens(}\text{fighting(Gr,[P1,P2])}, \text{T})$$

De este archivo de situaciones buscamos aquellas situaciones de grupo enmarcadas en el tiempo como fighting. Si alguno de nuestros identificadores está relacionado con esta situación y, además, observando los STA generados está andando, generaremos el STA abrupt en este punto de tiempo. Lo haremos a partir de walking porque es el que más sentido tiene de que se sustituya por abrupt de todos los STA con los que contamos.

Finalmente, para poder hacer uso de estos hechos será necesario cargar junto a los archivos mencionados anteriormente, el archivo *SituationGrp.pl* en el que se encuentran estos hechos, Se realizará de la siguiente manera.

:- use_module('CAVIAR_DATASET/set01/VIDEO_FIGHTING/ SituationGrp.pl').

Gracias a esto podremos generar los LTA de fighting sin ningún problema según el modelo generado.

3.3.4 Condiciones del código

En este punto veremos una lista con algunas condiciones a tener en cuenta en *Problog2* al generar el modelo para no tener problemas a la hora de ejecutarlo.

- Al tratar variables necesitaremos en cada regla diferenciar unas con otras mediante el uso de '*ID1 \== ID2*', para que no realice la regla para el mismo valor, ya que no tiene sentido decir que una persona está peleando o andando consigo misma. Esta condición tendremos que tenerla en cuenta en todas las reglas en las que aparezcan las dos variables excepto al calcular la distancia entre los dos ids, ya que entre ellos ya lo comprobaremos.
- Como para comprobar que se trata de una persona, la orientación de cada id y sus coordenadas hacemos uso de holdsAt, para evitar que compruebe esta regla cuando suponemos que estos hechos son tratados como datos de entrada, incluimos en la regla holdsAt una condición de que el fluente tratado no sea ninguno de estos, de la manera que vemos a continuación.

  F \= orientation(X)

- Al tratar de una manera diferente *Problog2* la negación, será necesario que la condición de que sea una persona (person) se realice en las reglas que aparece antes que la negación de algún STA. Es decir, antes de comprobar que no se trata de algún tipo de STA, debe saber antes que se trata de una persona, para conocer qué condiciones le pueden corresponder.
- Para evitar el uso de otra regla para el cálculo de la orientación o de la distancia entre coordenadas cuando no se cumpla, simplemente llamaremos a la misma regla ya definida para cuando queremos que sea válida, pero negándola y así su funcionamiento será el óptimo.

3.4 Datos de entrada: CAVIAR

En la base de datos de CAVIAR contamos con una infinidad de datos relacionados con las interpretaciones de los vídeos de vigilancia. De todos ellos nos interesará únicamente aquellos que tendremos en cuenta para la ejecución de nuestro modelo. En la base de datos contamos tanto con los datos sin procesar, que para nuestro caso no nos interesan, como con los datos ya transformados en lenguaje de programación lógica, que serán los que utilizaremos como datos de entrada de nuestro programa.

Dentro de cada ejemplo contaremos con varios archivos con datos, pero solamente utilizaremos de cada ejemplo dos archivos:

- *AppearenceIndv.pl*: en estos archivos obtendremos la información relacionada con la orientación además de appear/disappear, según comience a ser 'trackeado' o no.
- *MovementIndv.pl*: en estos archivos contaremos con la información acerca de los STA que se producen además de las coordenadas.

En el script de ejecución de estos datos la forma de llamarlos será mediante el uso de *use_module*. Para la ejecución de los eventos pertenecientes a los vídeos de *cose1gt*, la forma de llamarlos sería la siguiente:

```
:- use_module('CAVIAR_DATASET/set02/cose1gt/AppearenceIndv.pl').
:- use_module('CAVIAR_DATASET/set02/cose1gt/MovementIndv.pl').
```

En cuanto a qué vídeos utilizaremos, del primer set de vídeos nos interesarán casi todos, sobretodo aquellos en los que se anda, se dejan objetos, se saludan y discuten. En cuanto a los del segundo set, nos interesarán aquellos en los que las personas aparecen y desaparecen al salir o entrar de una tienda y en los que se mueven unos con otros.

Especificando estos del primer set de ejemplos que serán los verdaderamente importantes, utilizaremos los siguientes.

Actividades	Archivos utilizados
MOVING, WALKING	*wk1gt, wk2gt, wk3gt*
LEAVING_OBJECT	*lb1gt, lb2gt, lbbcgt, lbgt, lbpugt*
MEETING, WALKING	*mwt1gt, mwt2gt, mws1gt, ms3ggt, mc1gt, spgt*
FIGHTING	*fra1gt, fra2gt, fomdgt1, fomdgt2, fomdgt3, fcgt*

Tabla 9. Actividades que ocurren en cada vídeo de CAVIAR

Mencionar que la realización de estas actividades es teórica, puede ocurrir que se genere en alguno de ellos otra actividad no mencionada dependiendo de lo que se realice en dichos vídeos, cosa que se puede comprobar ejecutando el programa.

3.5 Manual de uso de Problog2

En este apartado detallaremos todo lo que necesitaremos para la ejecución de nuestro programa mediante el uso del lenguaje *Problog2*. Para ello detallaremos los requisitos necesarios para su instalación, además de la manera de ejecutar cada una de sus herramientas.

3.5.1 Instalación

Para la instalación del programa necesitaremos contar en nuestro ordenador con Python, ya que *Problog* se trata de una extensión de este.

La versión que instalaremos de *Problog* será la más moderna que existe a fecha de 2018-05-30 (2.1.0.36). Para la instalación de la versión de esta extensión deberemos contar con una versión de Python superior a la 3.6, por lo que utilizaremos para ello la versión 3.7 de este lenguaje, pudiéndose descargar de su web oficial.

Además de esto para una correcta instalación, actualizaremos pip a su versión más reciente (19.1.1) ya que su instalación se realizará mediante el uso de este comando.

pip3.7 install problog

Una vez realizado esto, ya podremos ejecutar el programa llamándolo desde la carpeta bin de esta versión de Python.

$/3.7/bin/problog ejemplo.pl

De esta manera se ejecutará el programa del archivo ejemplo.pl de la manera que tiene *Problog* por defecto. En el siguiente punto veremos cómo ejecutar todas estas formas de ejecución de este lenguaje.

3.5.2 Formas de ejecución

Como hemos dicho anteriormente, *Problog* tiene muchas formas de ejecución, que cada uno hace que se realicen distintas peticiones. La forma de ejecutar todas ellas es modificando el primer argumento de la llamada realizada la siguiente.

Ejecución	Herramienta utilizada
$- problog ejemplo.pl	Sin argumentos, inferencia por defecto con SDD.
$- problog sample ejemplo.pl -N 10	Muestreo con 10 ejemplos a realizar.
$- problog sample –estimate ejemplo.pl -N 10	Muestreo con estimación media de la probabilidad.
$- problog mpe ejemplo.pl	Most Probable Explanation.
$- problog lfi ejemplo.pl	Aprendizaje basado en interpretaciones.
$- problog dt ejemplo.pl	Uso de *DTProlog*.
$- problog map ejemplo.pl	Inferencia basada en MAP.

Tabla 10. Formas de ejecución Problog2

Además de estas también podemos ejecutar el modo explicativo (*explain*), grounding (*ground*), con shell interactiva (*shell*), redes bayesianas (*bn*) o ejecutando un servidor web (*web*). Todas estas escribiendo su llamada correspondiente mediante argumentos según se ha visto en la tabla anterior.

Además de esto, para almacenar su resultado en un archivo externo se puede hacer uso del comando siguiente.

$- problog lfi ejemplo.pl -O ejemploAprendido.pl

Esto es interesante en casos como este ejemplo donde el aprendizaje mediante interpretaciones genera como resultado un nuevo modelo con las probabilidades especificadas, por lo que así se genera un nuevo modelo en su archivo correspondiente de manera inmediata para su posterior uso en otro caso.

3.6 Implementación en Problog2

Una vez instalado el programa y conocidas todas sus opciones de uso, bastará con crear un buen modelo para que se ejecute correctamente y esto se logrará implementado el modelo creado anteriormente usando el lenguaje instalado.

Claramente, el programa debe estar escrito en un lenguaje lógico que pueda reconocer *Problog2*, que como ya mencionamos anteriormente, será como el lenguaje *Prolog* con el uso de probabilidades, consultas y evidencias. La estructura que debe tener ya la mencionamos anteriormente, por lo que vamos a ver de qué se compone cada parte.

Los datos de entrada se proporcionan mediante CAVIAR y con el uso de use_module, como hemos dicho en el punto anterior. La forma que adquieren estos datos para un *frame* de tiempo es la siguiente.

happens(walking(id0), 960).

holdsAt(coord(id0) = (182, 126), 960).

holdsAt(orientation(id0) = 149, 960).

holdsAt(appearance(id0) = visible, 960).

Siempre se generarán los cuatro hechos para cada *frame*, además de estar todos ellos ordenados en el tiempo.

Las reglas se escribirán siguiendo la estructura mencionada en el punto 3.2.2 pero transformándolo de la siguiente manera.

iniciatedAt(leaving_object(P, Obj) = true, T) :-

happens(appear(Obj), T),

happens(inactive(Obj), T),

holdsAt(close(P, Obj, 30) = true, T),

holdsAt(person(P) = true, T).

Las flechas que indica el hecho que se genera se cambia por '**:-**', entre cada hecho que compone la regla se escribirá una coma y para indicar que la regla finaliza un punto. También se hace uso de ***is*** para sustituir a la igualdad o el uso de algunos predicados implementados en el lenguaje para hacer su ejecución más fácil, como el calculo de raíces mediante ***sqrt*** o el valor absoluto con la función ***abs***. La lista de todos estos predicados que se pueden utilizar los tenemos en la página web de *Problog* [36].

query(holdsAt(moving(id0,id1)=true, 840))

Por último, las consultas de cada actividad se realizan mediante el uso de query, como podemos ver en la parte superior. Podemos especificar en la consulta tanto los identificadores como el *frame* en el que se realiza la actividad o mediante el uso de variables sin especificar los *frames* donde tiene que ejecutarlo para que el programa genere directamente los *frames* donde aparece esta actividad. Es interesante el uso de variables especificando en que momento ejecutar cada *frame* para además de obtener los *frames* donde se genera, generar aquellos en los que no ocurre, para su posterior análisis hacerlo más completo.

Después de esto, según la ejecución que hayamos utilizado, generaremos las probabilidades de que se cumplan las actividades consultadas, pero también podremos

dadas las evidencias de que cierta actividad se cumpla en cierto *frame* de tiempo, obtener las probabilidades de que los hechos sean ciertos o incluso la generación de un modelo mejorado especificando ciertas probabilidades de este mediante aprendizaje.

CAPÍTULO 4. ANÁLISIS DE LOS RESULTADOS

En este capítulo realizaremos el análisis del modelo creado a partir de los datos proporcionados por CAVIAR. Para ello diferenciaremos varios apartados donde hablaremos cómo se va a realizar este análisis, las características que tendrán cada uno de ellos y los resultados obtenidos analizados comparándolos con los datos proporcionados por la base de datos, para comprobar si se realiza un correcto reconocimiento de actividades.

4.1 Introducción al análisis

En este apartado analizaremos todo aquello que deberemos tener en cuenta para la realización del modelo, distinguiendo los tipos de experimentos que se realizarán, cómo se realizarán los experimentos para evaluar los resultados y qué valores utilizaremos para realizar estos experimentos.

4.1.1 Experimentos a realizar

Como ya sabemos, el uso de *Problog2* nos permite tratar la incertidumbre a la hora de realizar experimentos de programación lógica. Nuestro diseño está modelado de manera que se permiten realizar experimentos donde se acepte esta incertidumbre en la elección o distinción de actividades, pero de igual manera se puede ejecutar sin realizar esta distinción, tratándose de un programa meramente lógico.

Por esta razón distinguiremos entre estos dos tipos de experimentos:

- **Experimentación sin ruido artificial**. En este caso se tratará de un problema de programación lógica, donde tanto los eventos como las actividades a reconocer se ejecutarán suponiendo que no cuentan con una probabilidad de que ocurran, es decir, siempre se tratarán como ciertas. Por ello, cada vez que se genere una actividad será con una probabilidad de 1, ya que siempre será cierta y, en caso contrario, será de 0 ya que no lo será nunca.
- **Experimentación con ruido artificial**. En este caso se tratará de un problema de programación lógica probabilística, donde ciertos datos del problema, que la elección de ellos la veremos en su correspondiente apartado, se tratarán para ejecutarse con cierta probabilidad, para así generar esa incertidumbre que tenemos sobre las actividades. De igual manera, las actividades reconocidas también contarán con cierta probabilidad.

4.1.2 Ejecución del modelo

Como dijimos en el capítulo anterior, en vez de ejecutar el programa para saber en qué momentos en el tiempo se ejecuta cierta actividad, será interesante poder ejecutar todos los *frames* de tiempo y contar con los resultados que se obtienen en cada *frame*, tanto si se cumplen o no como la probabilidad de que se ejecuten en todos ellos. Incluso cuando sea de 0.

Para recorrer exactamente todos los *frames*, ya que en la base de datos estos se representan de 40 en 40, se puede hacer uso de la siguiente regla de ejecución para únicamente recorrer los *frames* con estos valores.

query*(CONSULTA) :-*
between*(0, LIMITE_FRAMES, C),*
T is *(C* * 40*)*.

El valor de CONSULTA será el de aquellos LTA expresados con el predicado holdsAt de Event Calculus que se quieren reconocer en cada caso, de la siguiente manera.

Posibles valores variable CONSULTA
holdsAt(meeting(P1, P2) = true, T)
holdsAt(moving(P1, P2) = true, T)
holdsAt(leaving_object(P1, P2) = true, T)
holdsAt(fighting(P1, P2) = true, T)

Tabla 11. Posibles valores atributo CONSULTA

Se tratarán de variables ya que queremos saber tanto cuándo se generarán y por parte de quién. El valor que deberá adquirir LIMITE*_FRAMES* dependerá del set de datos que utilicemos para realizar el análisis, ya que para ejecutar el número de *frames* exacto tendremos que conocer hasta que *frame* compone este conjunto, por lo que lo especificaremos este valor en el siguiente punto.

Otra forma de ejecución del modelo sería sin especificar la actividad que se producirá en cada *frame*, para que reconozca todas las posibilidades con las que cuenta el sistema: quién realiza la actividad, cuándo lo hace y de qué actividad se trata. El problema de esta ejecución es que sería mucho más costosa, ya que el recorrido de los datos sería mayor además de que el modelo no está codificado para permitir esta consulta, por ello se debe especificar siempre qué actividad queremos consultar.

La ejecución que se realizará de entre todas las herramientas con las que cuenta *Problog2* será la que se ejecuta por defecto (inferencia), ya que lo que buscamos es que dados ciertos hechos y una serie de reglas se generen las actividades que hemos consultado. Para saber si su comportamiento es el correcto se compararán con las que genera la base de datos de CAVIAR que ya tenemos proporcionada.

Estas actividades que compararemos con nuestros resultados se encuentran dentro de cada carpeta de vídeos de CAVIAR en los archivos *ContextGrp.pl* y *SituationGrp.pl*. Habrá que analizar ambos archivos ya que en el segundo de ellos se almacenarán los LTA cuando se produzcan dos actividades simultáneas. Si sólo se produce una actividad se encontrará en el primer archivo de los mencionados.

Las actividades reconocidas del modelo se mostrarán según la siguiente matriz de confusión.

	CAVIAR Positivos	**CAVIAR Negativos**	
Modelo Positivos	*TP*	*FP*	*Suma actividades modelo positivas*
Modelo Negativos	*FN*	*TN*	*Suma actividades modelo negativas*
	Suma actividades positivas CAVIAR	*Suma actividades negativas CAVIAR*	*SUMA TOTAL*

Tabla 12. Plantilla matriz de confusión

En ella podemos observar las actividades que se reconocen en la base de datos de CAVIAR junto a los reconocidos por nuestro modelo. En cada celda de la tabla tendremos diferentes tipos de datos. Por ejemplo, para los datos que nuestro modelo ha reconocido, diferenciará entre aquellos que realmente son positivos, en este caso **TP** (True Positives), o aquellos que son negativos, **FP** (False Positives), en cuanto a su comparación con los de CAVIAR. De manera similar ocurre lo mismo para los datos que nuestro modelo no ha reconocido como positivos, diferenciando entre **FN** (False Negatives), si se tratan en verdad de actividades que deberían haber sido reconocidas, y **TN** (True Negatives), si están bien ejecutadas ya que estas serán aquellas actividades que no ocurren ni en CAVIAR ni en nuestro modelo. En los extremos de la tabla contaremos con la suma de cada tipo de actividades. Cabe destacar que cuantos más datos se encuentren en la diagonal de 'True', con mayor precisión contará nuestro modelo para ese tipo de datos que esté tratando.

Por último, mencionar que para cada tipo de actividad mostraremos una matriz de confusión que muestre las actividades totales en cada una de las ejecuciones junto a una gráfica donde se muestren cuánto de preciso ha sido el modelo a la hora de predecir las actividades. Después de esto se procederá a hacer un análisis del modelo total con todos los LTA.

4.1.3 Conjunto de datos a analizar

Los datos que utilizaremos para generar el análisis del problema, junto a una explicación de lo que ocurre en este set y el valor que deberá adquirir LIMITE_*FRAMES* en la consulta será el siguiente

Vídeo	Actividad que se realiza	LIMITE_FRAMES
mwt2gt	moving y meeting	827
mwt1gt	moving y meeting	707
fomdgt1	fighting	959
fcgt	fighting	431
wk2gt	moving y meeting	1055
wk1gt	moving y meeting	610
lb1gt	moving, meeting y leaving_object	1421
lb2gt	leaving_object	1115
lbpugt	leaving_object	1151
fomdgt3	fighting	803

Tabla 13. Actividades de cada vídeo y LIMITE_FRAMES

Se puede observar como en algunos vídeos se realizan más de una actividad y esto es posible por lo que dijimos anteriormente de que los LTA que se reconocen no son mutuamente independiente, en el mismo *frame* se pueden producir varias actividades.

Los escenarios que se recogen en cada uno de estos vídeos explicados con detalle para saber exactamente lo que ocurre en cada uno de ellos, son los siguientes.

Vídeo	Escenarios grabados
mwt2gt	Dos personas se saludan, empiezan a andar juntas y al final se separan
mwt1gt	Igual que el anterior, dos se saludan, empiezan a andar y se separan al final
fomdgt1	Dos personas se encuentran, empiezan a discutir y se pelean, una cae al suelo y la otra sale corriendo de la situación
fcgt	Dos personas se encuentran, se pelean y empiezan a perseguirse una a la otra
wk2gt	Una persona anda de un lado a otro del escenario mientras que dos personas están hablando al fondo de la imagen,
wk1gt	Al igual que antes, una persona se mueve de un lado a otro mientras dos hablan en el fondo de la imagen, moviéndose mínimamente.
lb1gt	Dos personas se saludan y empiezan a andar juntas y después, otra persona deja una mochila en el suelo
lb2gt	Una persona deja una mochila en el suelo, desaparece de la imagen y posteriormente vuelve a aparecer para recogerla.
lbpugt	Una persona deja una mochila en el suelo y posteriormente la recoge.
fomdgt3	Dos personas se encuentran, empiezan a discutir y se pelean, una cae al suelo y la otra sale corriendo de la situación

Tabla 14. Descripción escenarios vídeos de CAVIAR

Si se quiere acceder a la base de datos completa de vídeos de vigilancia de CAVIAR o simplemente ver los vídeos en los que nos basamos para realizar el análisis, se puede visitar la página web de CAVIAR [35].

4.2 Primer experimento: Datos sin ruido artificial

Análisis del experimento sin ruido artificial, con las características y los resultados especificados en el punto anterior.

4.2.1 Contexto experimentación sin ruido

Como hemos dicho anteriormente, este experimento se realizará sin tener en cuenta probabilidades en ninguno de sus datos, por lo que sus datos de entrada serán siempre ciertos con una probabilidad de 1, las reglas se ejecutarán siempre y cuando se ejecuten todos y cada uno de los predicados que la componen y, por último, se generará la actividad consultada con una probabilidad de 1 también.

Esto será así ya que al no contar con probabilidades las condiciones de que se ejecute algo, al tener que cumplirse todos sus hechos, su probabilidad condicionada será siempre

de 1 o de 0, por lo que para representar que se genere una actividad contaremos con que tiene una probabilidad de 1 y no se genera cuando tiene una probabilidad de 0.

Su ejecución, al tratar de esta manera la probabilidad, se podrá ejecutar mediante *Problog2*, imitando al funcionamiento que se podría realizar mediante *Prolog*, programación lógica sin probabilidad.

4.2.2 Resultados obtenidos experimentación sin ruido

Los resultados obtenidos en la experimentación sin ruido en los datos de entrada se mostrarán aquí. Para cada actividad a reconocer especificaremos que vídeos hemos utilizado de la tabla anterior para su análisis, para evitar la comprobación de vídeos en los que no se produce esa actividad y ahorrar trabajo.

Esta labor se puede realizar gracias a que en la base de datos, además de los datos de entrada que utilizaremos para ejecutar nuestro modelo, contamos con los datos que previsiblemente debe reconoce, por lo que podremos realizar este análisis ya que contaremos con los resultados que el modelo debe generar y así, compararlos con las respuestas que genere para comprobar si el modelo es bueno o no.

Mencionar que los datos estadísticos relacionados con la precisión del reconocimiento de cada actividad los tendremos en el apartado final de esta sección donde relacionaremos todas las actividades entre sí para poder hacer una comparación más completa.

Análisis Moving

Para la actividad de moving utilizaremos los vídeos almacenados en *mwt2gt*, *mwt1gt*, *wk2gt*, *wk1gt* y *lb1gt*. La matriz de confusión y su gráfica correspondiente son las siguientes.

	CAVIAR Positivos	**CAVIAR Negativos**	
Modelo Positivos	801	46	847
Modelo Negativos	297	3530	3827
	1098	3576	4674

Tabla 15. Matriz confusión MOVING experimento sin ruido

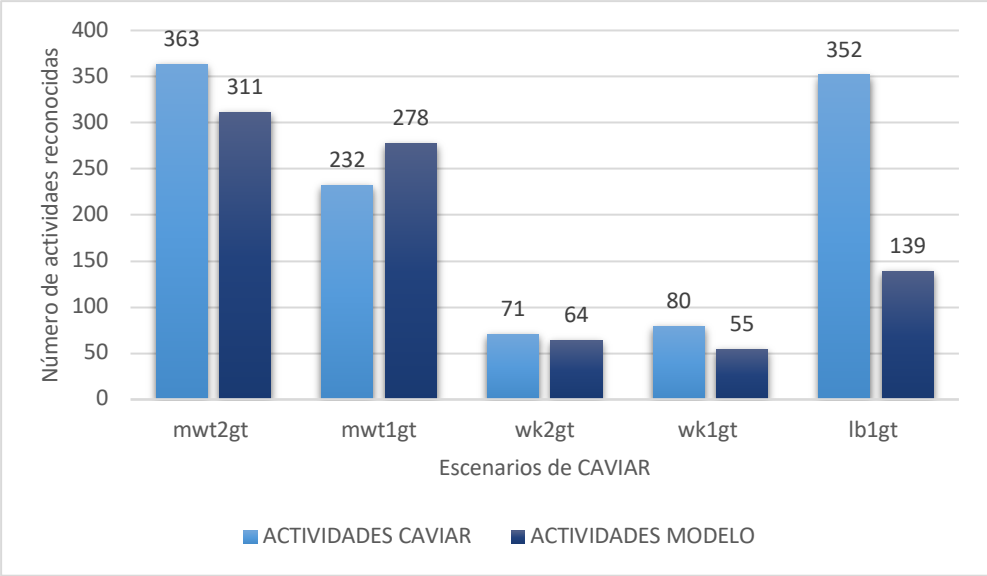

Ilustración 8. Gráfica actividades MOVING experimento sin ruido

Tanto en la matriz de confusión como en el eje Y de la gráfica podemos observar el número de actividades de moving que podrían ocurrir a lo largo de la ejecución de cada vídeo en nuestro modelo, generando una actividad por cada *frame* del vídeo. Los datos relacionados con la precisión los veremos en el análisis total.

Tras la ejecución de los datos obtenidos de estos vídeos de vigilancia podemos observar como la actividad moving tiene una precisión alta incluso generando falsos positivos, ya que se reconocen de los 4674 *frames* de vídeos ejeecutados, 847 en los que ocurre esta actividad, siendo el total reconocidos por CAVIAR de 1098.

Aún y así vemos una diferencia considerable de unos vídeos a otros. La razón de esto es sobretodo por la manera en la que está realizada la asignación de actividades en la base de datos. En el caso de los archivos de datos de *wk2gt* y *wk1gt*, el número de generaciones de la actividad es baja ya que son vídeos que si observamos se tratarán de gente andando simplemente, por lo que generará la actividad únicamente cuando detecte que las personas que aparecen estén juntas una de la otra, acción que no ocurrirá en la mayoría de los casos.

Por otro lado, para el archivo de datos del vídeo lb1gt, vemos como la precisión desciende, ya que se reconocen 139 actividades de las 352 que se debían haber reconocido. Esto es así porque en este vídeo se basan en mostrar como una persona deja una mochila en el suelo, pero además de esto aparecen dos personas juntas, por lo que en algún momento considera que se activa esta actividad, pero no están moviéndose como tal ya que en múltiples momentos se paran y miran cada uno para un lado (haciendo que la condición de diferencia de orientación mayor de 45º se active para que termine esta actividad), mientras que estas consideraciones no las tiene la base de datos de CAVIAR y, aunque su orientación sea la contraria o estén parados, en la mayoría de los *frames* detecta que están realizando la actividad moving, generando esta gran cantidad de FN y haciendo que descienda su precisión.

Análisis Meeting

Para la actividad de meeting utilizaremos los vídeos almacenados en *mwt2gt*, *mwt1gt*, *wk2gt*, *wk1gt* y *lb1gt*. La matriz de confusión y su gráfica correspondiente son las siguientes.

	CAVIAR Positivos	**CAVIAR Negativos**	
Modelo Positivos	355	0	355
Modelo Negativos	2651	1613	4264
	3006	1613	4619

Tabla 16. Matriz confusión MEETING experimento sin ruido

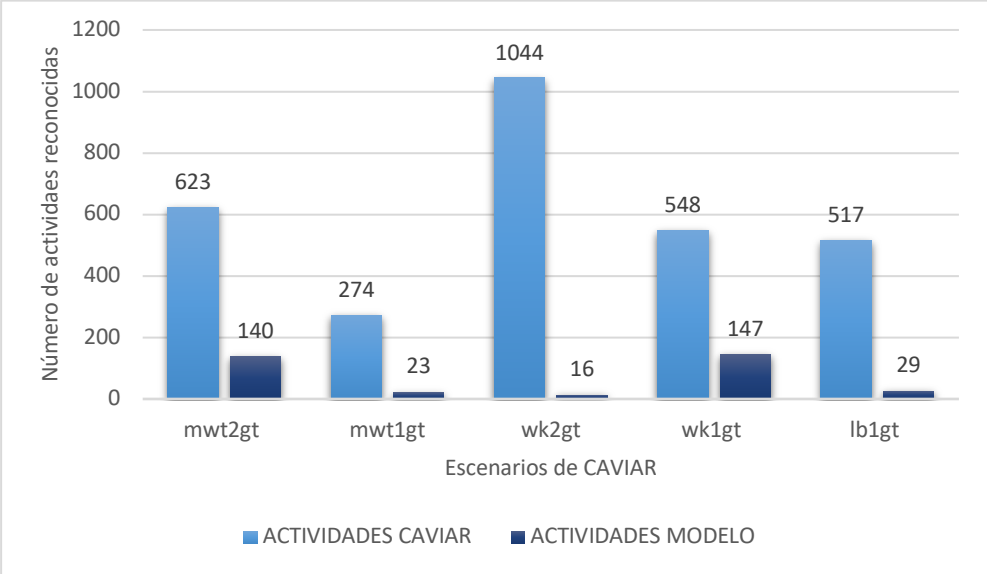

Ilustración 9. Gráfica actividades MEETING experimento sin ruido

Tanto en la matriz de confusión como en el eje Y de la gráfica podemos observar el número de posibles actividades de meeting que ocurren por cada *frame* del vídeo a lo largo de su ejecución. Los datos relacionados con la precisión los veremos en el análisis total.

De los 4619 *frames* de vídeo que se revisan, en 3006 de ellos ocurre meeting y nuestro sistema reconoce de estos solamente 355 *frames* de vídeo, lo que es una precisión bastante baja. El desglose por vídeos tratados para comprobar esta actividad se puede ver en la gráfica.

Para el reconocimiento de esta actividad podemos observar como el problema del archivo de datos de la actividad anterior se propaga en este caso para la mayoría de casos. De la

misma manera, se generan muchas actividades de meeting (4619) en la base de datos de CAVIAR, pero en nuestro caso la mayoría de estas actividades no se reconocen (355) ya que nuestras reglas son muy restrictivas para su reconocimiento.

Contamos con muchas condiciones de terminación de la actividad meeting y al no tratar la incertidumbre de que puedan no ocurrir alguna de sus condiciones, siempre de alguna u otra manera se confirmará una regla que provocará la terminación de la actividad y, posteriormente, su no generación como respuesta del sistema. Por ejemplo, al detectarse un movimiento abrupto, si dudar de que pueda ocurrir de verdad o no, hay una condición de terminación de la actividad para este caso. Esto en la vida real no es así, porque, por ejemplo, para considerar que se termina el saludo entre dos personas no se puede considerar un movimiento abrupto de una de ellas, hay más condiciones a tener en cuenta.

La solución de este tipo de problemas se realiza gracias al uso de la probabilidad, ya que de esta manera las condiciones de terminación no serán tan restrictivas, y se basarán en un cierto valor de probabilidad, que dependiendo del caso y de la cantidad de condiciones de terminación que se generen, será mayor o menor su probabilidad de ejecución.

Análisis Fighting

Para la actividad de fighting utilizaremos los vídeos almacenados en *fomdgt1*, *fcgt* y *fomdgt3.* La matriz de confusión y su gráfica correspondiente son las siguientes.

	CAVIAR Positivos	**CAVIAR Negativos**	
Modelo Positivos	262	0	262
Modelo Negativos	215	1716	1931
	477	1716	2193

Tabla 17. Matriz confusión FIGHTING experimento sin ruido

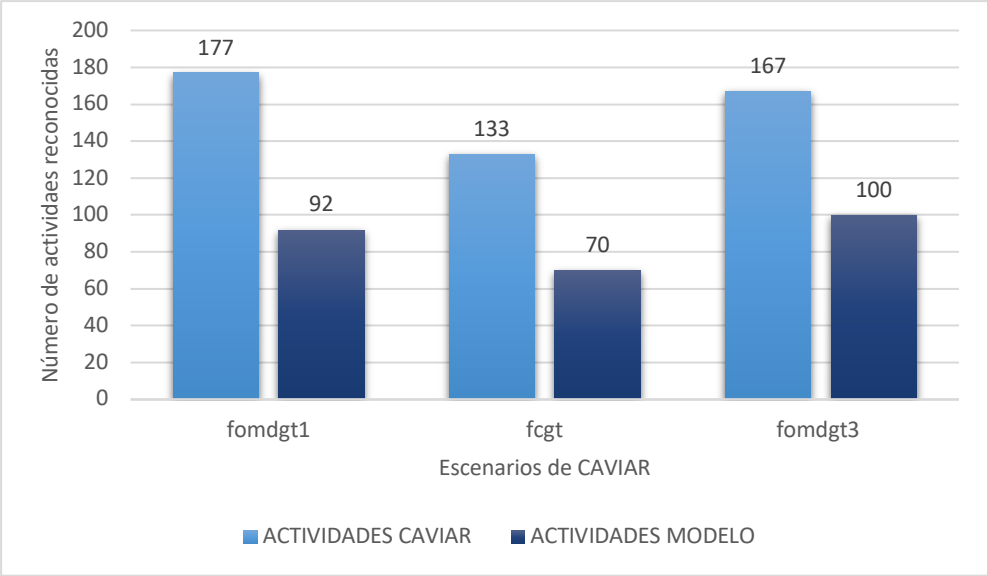

Ilustración 10. Gráfica actividades FIGHTING experimento sin ruido

Al igual que ocurre con meeting y moving, el eje Y de la gráfica y la matriz de confusión se componen de las posibles actividades de fighting que se producen a lo largo de cada *frame* de los vídeos tratados. Los datos relacionados con la precisión los veremos en el apartado de análisis total.

En este caso, de los 2193 *frames* tratados, se reconoce en CAVIAR la actividad de fighting en 477 de ellos, de los cuales nuestro modelo reconoce 262, un poco más de la mitad. En la gráfica observamos como se comporta para cada vídeo tratado este reconocimiento.

Para el reconocimiento de la actividad fighting la precisión vuelve a subir en comparación con la actividad de meeting. Se debe básicamente a la misma razón que en el apartado anterior, pero esta vez aumenta ya que las condiciones de iniciación y terminación no son tan restrictivas como en el apartado anterior. Además de esto, los vídeos utilizados para el análisis de esta actividad se basan únicamente en encuentros entre dos personas en los que no ocurre otra actividad.

Esto quiere decir que, en otros vídeos tratados, aparte de la actividad que se está intentando reconocer, se generan muchas más actividades por parte de las mismas personas u otras que también aparecen en la acción. En cambio, en este caso, los vídeos que tratan la actividad de fighting se basan básicamente en generar la actividad y terminarla, esto es que dos personas se encuentran, se pelean y se separan, siendo los momentos de iniciación y terminación de la actividad muy claros para su posterior reconocimiento.

Análisis Leaving_object

Para la actividad de leaving_object utilizaremos los vídeos almacenados en *lb1gt*, *lb2gt* y *lbpugt.* La matriz de confusión y su gráfica correspondiente son las siguientes.

	CAVIAR Positivos	CAVIAR Negativos	
Modelo Positivos	168	0	168
Modelo Negativos	127	3043	3170
	295	3043	3338

Tabla 18. Matriz confusión LEAVING_OBJECT experimento sin ruido

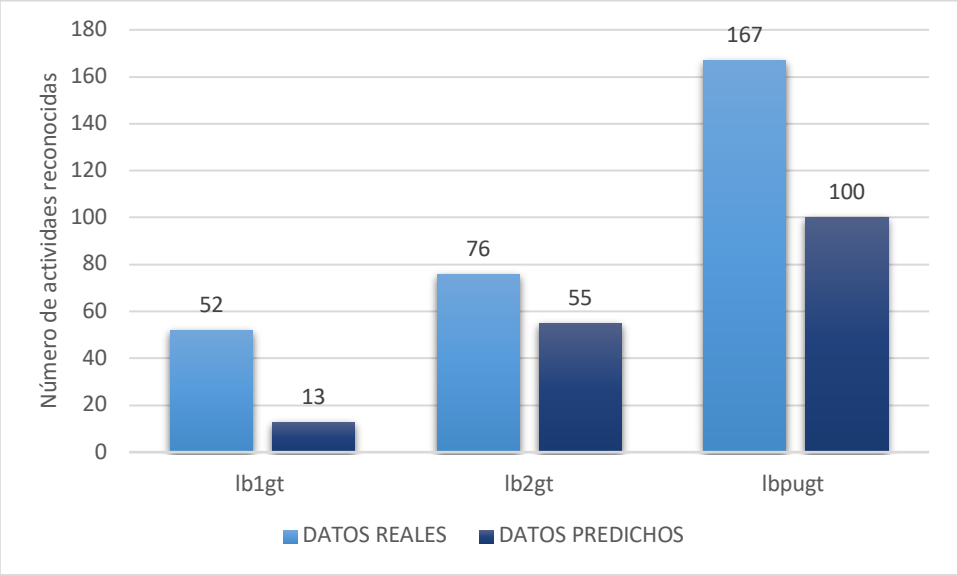

Ilustración 11. Gráfica actividades LEACING_OBJECT experimento sin ruido

En este caso la matriz de confusión y el eje Y de la gráfica se componen del número de posibles actividades de leaving_object que podrían ocurrir durante la ejecución de cada vídeo, siendo una por cada *frame* de estos. Además, la precisión de estos datos la veremos en el análisis total.

En este último caso contamos con 3338 *frames* de vídeo, en los cuales sólo hay 295 en los que se produce la actividad de leaving_object y de todos estos sólo se reconocen 168 *frames* de estos vídeos. Una vez más, la gráfica representa los datos divididos por cada vídeo tratado.

Para el caso del reconocimiento de la actividad de leaving_object el proceso quizá es un poco más complicado, ya que aquí entra en consideración lo que entendemos nosotros como realización de esta actividad y lo que entiende la base de datos de CAVIAR actividad. CAVIAR parece considerar que esta actividad termina cuando la persona que ha dejado el objeto desaparece de la escena del vídeo, mientras que nosotros esta condición no la tenemos en cuenta, ya que, aunque la persona desaparezca del vídeo, la acción de haber dejado un objeto se debería seguir generando, ya que no termina de ser

falsa. Las condiciones de terminación que nosotros tenemos en cuenta para esta actividad se basan en que el objeto vuelva a ser recogido o que el objeto desaparezca de la escena. En este caso sí que tiene sentido la terminación de la actividad ya que generar una actividad sobre algo que no existe en el momento en el que se genera no tiene sentido. Por esta razón en la mayoría de los *frames* esta actividad no se reconoce como debería.

Por otro lado, comparando la precisión de acierto con la de las anteriores actividades, en estos archivos aumenta el reconocimiento de la actividad. Esto se debe a que para esta actividad se tratan vídeos donde el escenario principal es el de dejar un objeto, por lo que las condiciones de iniciación de la actividad a reconocer ocurrirán claramente, favoreciendo su reconocimiento. Aún y así, su reconocimiento positivo seguirá dependiendo tanto de cuando considere la base de datos que ha terminado de reconocer a la persona que ha dejado el objeto (generando posibles FP) como cuando deje de reconocer al objeto, aún apareciendo en el vídeo (generando FN).

Análisis total

Una vez analizados todos los datos para cada actividad de LTA, rellenaremos una tabla con el análisis total de nuestro modelo, junto a una gráfica representando todos los datos anteriores en los que especificar la precisión que ha obtenido el modelo a la hora de reconocer las actividades que ocurren en cada *frame* de vídeo.

En la tabla tendremos tres tipos distintos de predicción, todos ellos generados como números normalizados para saber cuántas de estas actividades se han reconocido correctamente en función del total que se considere en cada caso. Los tipos de predicción que trataremos de aquí en adelante serán los siguientes:

- ***Precisión Total****:* como su nombre indica, será la precisión total del modelo, en la que se indicará de todas las actividades, tanto reconocidas positivamente como negativamente, cuántas de ellas se han asignado bien en función al reconocimiento base que tiene CAVIAR. En este caso, sólo se tendrá en cuenta que la acotación de los datos sea la correcta. Si esta precisión es baja, no implica que el reconocimiento no sea bueno, sino que la asignación de los TP y TN para ese reconocimiento no es correcta, generando FP y FN.
- ***Recall***: este valor de precisión indicará que de todas las actividades que CAVIAR reconoce que ocurren en sus vídeos, cuantas de estas actividades son reconocidas realmente por nuestro modelo. Este valor será muy interesante para saber si nuestro modelo reconoce bien las actividades o no.
- ***PAV*** (Precisión Actividades Válidas): esta precisión hará referencia al número de actividades reconocidas por nuestro modelo que están correctamente reconocidas. En este caso se obtendrá una precisión de 1 siempre y cuando no haya en el reconocimiento ejecutado falsos positivos (FP).

Siguiendo la plantilla de la matriz de confusión y usando sus valores, su definición será la siguiente.

Precisión total	Recall	Precisión Actividades Válidas (PAV)
$\frac{TP + TN}{SUMA\ TOTAL}$	$\frac{TP}{Suma\ actividades\ positivas\ CAVIAR}$	$\frac{TP}{Suma\ actividades\ modelo\ positivas}$

Tabla 19. Explicación datos precisión

Una vez especificado esto, la gráfica en la que representaremos las actividades reconocidas por nuestro modelo, además de las que componen la base de datos, será la siguiente, viéndose estas actividades en el eje Y de la gráfica. Además de la gráfica a continuación representamos la tabla con todos los datos tratados junto a la precisión de cada uno de ellos, siguiendo la siguiente disposición.

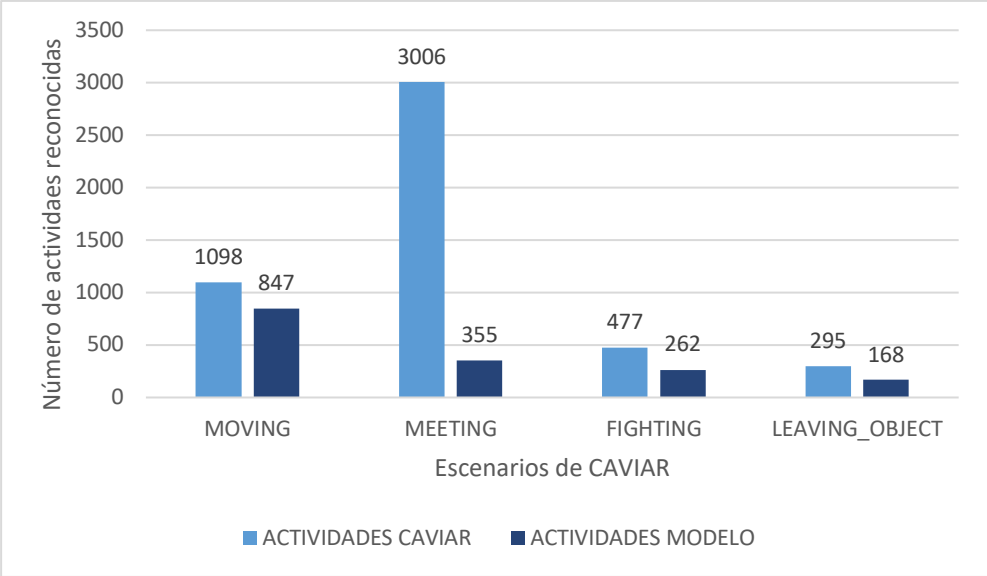

Ilustración 12. Gráfica actividades totales experimento sin ruido

LTA	Valores *frames* consultados			Porcentaje precisión		
	TP	*TN*	*FN*	*Precisión total*	*Recall*	*PAV*
moving	801	3530	297	0.927	0.730	0.946
meeting	355	1613	2651	0.426	0.118	1.000
fighting	262	1716	215	0.902	0.549	1.000
leaving_object	168	3043	127	0.962	0.569	1.000

Tabla 20. Actividades reconocidas y porcentaje de precisión

En líneas generales como hemos dicho en el análisis de cada una de las actividades, el reconocimiento que realiza nuestro modelo en la experimentación sin ruido tiene algunas deficiencias a pesar de reconocer en algunos de los casos muchas actividades.

Observando la *Precisión Total* del modelo, reflejada en la tabla anterior, es bastante alto para todas las actividades excepto para la actividad de meeting. Para todos ellos supera el 90% de acierto, pero como ya dijimos, esta precisión es un tanto ficticia. La única conclusión que se puede sacar de esta precisión es que el funcionamiento del modelo es correcto ya que la mayoría de actividades las ordena bien.

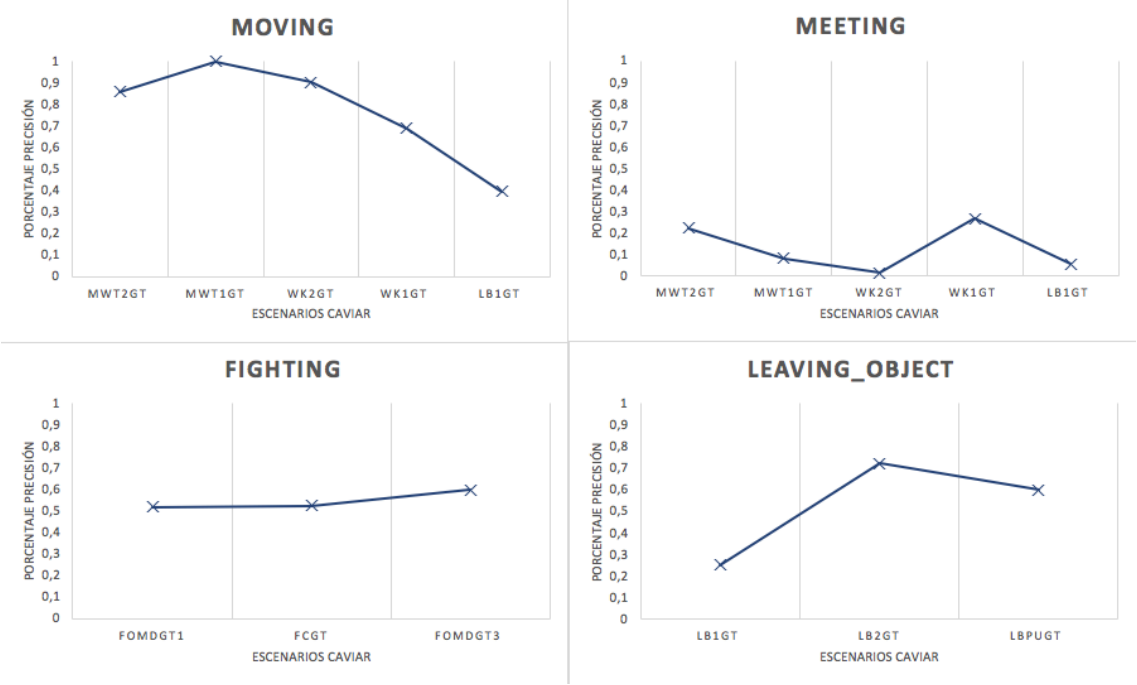

Ilustración 13. Gráficas con la precisión de RECALL para cada actividad en experimentación sin ruido

Para el caso que nos ocupa, la precisión que de verdad nos importa es la precisión de *Recall*, la cual podemos observar en las gráficas anteriores para cada uno de los escenarios tratados y una media de todos estos en la tabla de precisiones anteriormente mencionada.

En este caso observamos qué tan bueno es el reconocimiento de cada una de las actividades que tratamos en el modelo y podemos observar claramente que su precisión no se asemeja a la que se espera de un buen modelo. Para el caso de meeting no supera el 12% mientras que para el resto ronda el 60-70%. La principal razón de esto, como dijimos anteriormente, es la falta de precisión por parte de la base de datos a la hora de generar las actividades en algunos casos junto a las restricciones tan estrictas que se generan en nuestro sistema de reconocimiento, lo cual hace que este porcentaje de precisión descienda para todos los LTA. Se vuelve a demostrar por lo tanto la necesidad de la incertidumbre para no generalizar en cuanto a las actividades que ocurren en cada momento, pudiendo generar esa duda necesaria para resolver problemas reales.

Por último, observamos como la precisión *PAV*, refiriéndose a aquellas actividades que están correctamente reconocidas, su porcentaje de precisión es total en todas excepto para el caso de moving, que desciende mínimamente a 94,6%.

Esto puede ser una casualidad con respecto a los vídeos que hemos utilizado para el análisis, pero en el fondo tiene sentido. Debido a la gran cantidad de restricciones generadas y al bajo número de valores que pueden tomar las actividades en el sistema. Recordemos que las condiciones iniciales del modelo constan únicamente de cinco posibles atributos, sin contar con los relacionados con la posición y la apariencia, y las actividades finales pueden ser sólo cuatro. Esto provoca que la ejecución de casos en los que el modelo no se sepa qué se está realizando baje, por lo que la probabilidad de que reconozca una actividad cuando en realidad no está pasando es poco probable.

En definitiva, no se puede esperar que este experimento reconozca un alto número de actividades debido a su baja similitud con un problema del mundo real. Como ya hemos mencionando, para hacerlo más preciso bastaría con incluir, como hemos visto a lo largo de este trabajo, probabilidades en la ejecución del modelo, para tratarlo como un problema más real donde las condiciones de iniciación o terminación de una actividad no sean tan estrictas.

4.3 Segundo experimento: Datos con ruido artificial

En este experimento se cuenta con ruido artificial y se especificarán los resultados según la forma descrita en el punto anterior.

4.3.1 Contexto experimentación con ruido

El experimento anterior, aunque no sea del todo preciso, es válido, pero no se asemeja del todo a un problema que nos podamos encontrar en la vida real, porque no la certeza de que algo siempre vaya a ocurrir no ocurre en problemas reales. Por esta razón, haremos uso de la incertidumbre. La experimentación en este caso contará con ruido en una serie de datos de nuestro modelo para generar esta incertidumbre.

Este ruido se generará mediante valores probabilísticos en algunos de los hechos de nuestro modelo, lo que lo transformará en un problema de programación lógica probabilística, finalidad para la que se ha elegido Problog2, en el que las actividades a reconocer también se generarán con cierta probabilidad.

La manera de incluir esta probabilidad en los datos puede ser muy diversa, pero cada una de ellas tiene algún problema por el que su uso no es trivial, que veremos a continuación. En primer lugar, habrá que especificar en qué datos del modelo se incluirían estas probabilidades, los cuales serán en los hechos tratados como la entrada del modelo, en las reglas de ejecución que hacen que se cumpla la actividad que se realiza o incluso en ambas dos.

Por otro lado, las distintas posibilidades de ejecución del problema para incluir las probabilidades serían las siguientes.

- Se podrían generar los valores de probabilidad de manera aleatoria con valores entre 0 y 1. El problema de este método es que el modelo no seguiría ningún patrón para asignar las probabilidades, ya que cada probabilidad estaría establecida sin seguir ningún tipo de criterio.
- Se podrían establecer estos valores aleatoriamente también, pero acotándolos a los valores proporcionados por distribuciones estadísticas, como podría ser por ejemplo una distribución gamma, la cual seguirá generando valores aleatorios, pero ya con algún sentido de ejecución, además de poder establecer manualmente el rango en el que se encuentren estas probabilidades.
- Otra opción claramente sería el uso de datos de una base de datos que ya estén ponderados con ciertas probabilidades, algo que se prevé improbable que exista alguna base de datos que funcione de esta manera.
- Otra opción sería la inclusión manual de estas probabilidades, haciendo que sus valores se establezcan según nuestro propio criterio y sentido de una ejecución de ese estilo en la vida real.

De entre todas estas posibilidades parece que la más correcta sería la segunda, en la que se podría ejecutar varios ejemplos con distintos valores de los atributos de la distribución para ver cómo se comporta el modelo, pero tanto este tipo como el primero ha sido imposible llevarlas a cabo ya que la versión utilizada de Problog2 no permite tratar con valores aleatorios, ya sean números o los proporcionados por distribuciones estadísticas. En adicción a esto, la tercera opción tampoco es posible por la falta de una base de datos con estas características. En consecuencia, haremos uso de la última opción descrita.

El problema de esta opción es que la inclusión en todos los hechos de las probabilidades manualmente es una labor muy costosa ya que contamos con un gran número de *frames* a tratar, por lo que únicamente se incluirán manualmente las probabilidades en las reglas que componen nuestro modelo.

De esta manera el funcionamiento tampoco será el esperado ya que, si únicamente las reglas obtienen probabilidades, seguirán generando las mismas actividades sólo que cambiando la probabilidad de que ocurran. Para solucionar esto, deberemos incluir probabilidades en los hechos de entrada automáticamente, según el caso que queramos reconocer, como veremos a continuación.

Esta probabilidad en los hechos de entrada se basará básicamente en que, si ocurre en cierto *frame* de vídeo una actividad STA, estableceremos una probabilidad baja a que ocurra también otra actividad distinta. De esta manera las reglas que no se generan por no ocurrir cierto STA (al computarse la probabilidad de una regla como la multiplicación de todas sus condiciones, si una de ellas es ‘0’ la regla será ‘0’), ahora si que se generarán, con una probabilidad baja pero lo suficiente para que genere el reconocimiento de más actividades que en el primer experimento.

Al contar ahora con probabilidades para las reglas, los STA y, posteriormente, los LTA, habrá que establecer algún mecanismo para diferenciar entre las actividades que se reconocen y las que no. Para ello haremos uso de un *umbral de reconocimiento.* Este

umbral consiste en que aquellos LTA que se han reconocido con un valor probabilístico superior al umbral establecido serán catalogados como valores positivos, mientras que, en el caso contrario, los LTA que no superen el umbral, incluidos los que obtengan una probabilidad de '0', serán tratados como negativos. Estableceremos el umbral en un valor de 0.5, punto medio para establecer la misma probabilidad de que se reconozca una actividad como que no se haga

En el siguiente punto veremos cómo funciona la ejecución del modelo con esta experimentación.

4.3.2 Resultados obtenidos experimentación con ruido

La ejecución de esta experimentación será como la primera que se ha realizado, pero incluyendo probabilidades en las reglas del modelo y la capacidad de generar otras actividades STA con baja probabilidad para tenerlas en cuenta en el reconocimiento de las LTA.

Esta probabilidad aleatoria en las actividades STA seguirá la siguiente estructura.

Probabilidades en actividades STA	Condición inicial
0.2 :: happens(abrupt(P), T); 0.25 :: happens(walking(P), T); 0.1 :: happens(running(P), T); 0.25 :: happens(inactive(P), T)	:- happens(active(P), T).
0.2::happens(running(P), T); 0.15::happens(inactive(P),T); 0.2::happens(abrupt(P), T)	:- happens(walking(P), T).
0.05::happens(inactive(P),T); 0.2::happens(abrupt(P), T)	:- happens(running(P), T).
0.05::happens(inactive(P),T)	:- happens(abrupt(P), T).

Tabla 21. Probabilidades establecidas manualmente en actividades STA

Como vemos se establece la condición de que ocurra cada STA, y para cada uno de ellos en el *frame* en el que ocurra se crean nuevas actividades STA con una probabilidad de ejecución baja. De esta manera, como hemos dicho anteriormente, las reglas que no se activaban por la falta de estas condiciones ahora sí lo harán, pero no afectarán en exceso al reconocimiento ya que su probabilidad de aparición es baja en comparación con las actividades que se reconocen realmente.

Para establecer cada probabilidad manualmente se ha utilizado el sentido común de lo que podría ocurrir en cada caso en la vida real, haciendo que tengan más probabilidades aquellas STA que tienen más sentido que ocurran según el STA que en realidad se reconoce. Por ejemplo, para el caso en el que se detecte el STA active, tendrá más sentido que se haya confundido con una acción en la que la persona esté inactiva o andando (las cuales tienen una probabilidad de 25%) a que en realidad esté corriendo (10%), y esto se repite con todas. La razón por la que se reduce el número de actividades que se pueden

generar en cada STA según avanzamos es para evitar la ejecución en bucle del modelo y de esta manera se ejecuta correctamente.

Las probabilidades de cada regla las veremos en su correspondiente análisis de reconocimiento de LTA.

Análisis meeting

Para el reconocimiento de la LTA meeting, la definición de las probabilidades de las reglas que afectan a esta LTA se ha seguido un proceso similar al de los hechos. Se han establecido cada probabilidad según las posibilidades de que se generen estas reglas en la vida real, dando más posibilidades a las más obvias. Por ejemplo, tiene más sentido que termina la actividad de meeting si una de las dos personas empieza a correr (95%) que si una persona desaparece de la imagen (75%) porque para este último caso se debe tener en cuenta que puede haber desaparecido o no, ya que puede haberse activado el hecho de que haya desaparecido por error o porque de verdad haya desaparecido de la imagen, pero en realidad sigan realizando la LTA meeting.

Las probabilidades establecidas para las reglas de esta LTA, además de las reglas de las siguientes LTA, podemos verlas en los anexos del documento.

Una vez establecidas las probabilidades se procederá a la ejecución de algunos de los escenarios de vídeos para comprobar su comportamiento. En primer lugar, se ejecutará el escenario wk2gt, en el que se produce esta LTA y, en la experimentación anterior, ha tenido la precisión más baja de todos los escenarios ejecutados.

Una vez ejecutado este ejemplo, de la misma manera que se explicó al principio del capítulo, un ejemplo de su ejecución sería el siguiente.

```
holdsAt(meeting(id2,id1)=true,2720):   0.54
holdsAt(meeting(id2,id1)=true,2760):   0.54
 holdsAt(meeting(id2,id1)=true,280):   0.08256
holdsAt(meeting(id2,id1)=true,2800):   0.54
holdsAt(meeting(id2,id1)=true,2840):   0.54
holdsAt(meeting(id2,id1)=true,2880):   0.54
holdsAt(meeting(id2,id1)=true,2920):   0.54
holdsAt(meeting(id2,id1)=true,2960):   0.54
holdsAt(meeting(id2,id1)=true,3000):   0.54
 holdsAt(meeting(id2,id1)=true,320):   0.08256
 holdsAt(meeting(id2,id1)=true,360):   0.08256
holdsAt(meeting(id2,id1)=true,36880):   0.48
holdsAt(meeting(id2,id1)=true,36920):   0.48
holdsAt(meeting(id2,id1)=true,36960):   0.48
```

Ilustración 14. Ejemplo ejecución modelo con ruido

Vemos como se generan las probabilidades descritas anteriormente tal y como dijimos. Observando el escenario por completo, la matriz de confusión de las actividades que se reconocen en la ejecución de este escenario será la siguiente.

	CAVIAR Positivos	CAVIAR Negativos	
Modelo Positivos	80	0	80
Modelo Negativos	964	11	975
	1044	11	1055

Tabla 22. Matriz confusión escenario wk2gt con ruido.

En la matriz de confusión hemos representado todas las actividades que reconoce nuestro modelo, sin importar el umbral de reconocimiento, y vemos como aumenta en comparación con la experimentación sin ruido, para la cual los TP que se generaban eran 16. Estos resultados tienen sentido ya que ahora se tienen en cuenta ciertas reglas que antes no se ejecutaban por la falta de incertidumbre en la aparición de diversos STA.

Aún y así, los resultados siguen sin asemejarse a los que proporciona CAVIAR, porque incluso con las probabilidades en los hechos STA y las reglas el modelo sigue precisando de una asignación de las probabilidades más real. Además, si tenemos en cuenta el umbral de reconocimiento establecido anteriormente, el reconocimiento de la LTA meeting desciende a los mismos 16 *frames* reconocidos en la ejecución sin el uso de las probabilidades.

Análisis moving

Para el análisis de este LTA se utilizará el escenario *mwt2gt*, para el cual se obtuvo una precisión de Recall de 85,67% (311*frames* reconocidos de los 363 que había por reconocer), para comprobar como afecta la inclusión de la probabilidad en un caso con alta precisión sin el uso de la incertidumbre.

Tras su ejecución, la matriz de confusión obtenida para este ejemplo será la siguiente.

	CAVIAR Positivos	CAVIAR Negativos	
Modelo Positivos	363	84	447
Modelo Negativos	0	464	975
	363	464	827

Tabla 23. Matriz de confusión escenario mwt2gt con ruido

Podemos observar como los resultados obtenidos para este escenario, sin tener en cuenta el umbral de reconocimiento, ocurre como en el caso anterior, ya que las reglas que antes no se ejecutaban ahora lo hacen y, para este caso en el que la precisión ya de por sí era alta, se generan incluso FP. Como en el caso anterior teniendo en cuenta el umbral de reconocimiento, los TP generados serán 311, los mismo que en el escenario sin el uso de la incertidumbre.

Para los demás escenarios y las distintas actividades LTA con las que contamos, el funcionamiento es parecido al ya descrito en estos escenarios, por lo que obviamos su definición ya que en todas ellas se generarán mayor número de actividades reconocidas gracias a la adicción de actividades STA con baja probabilidad que antes no estaban presentes. Sin embargo, al tener todas ellas una baja probabilidad, la actividad reconocida también la tendrá, por lo que no superará el umbral de reconocimiento necesario para tratar la actividad como reconocida positivamente.

Esto se debe básicamente a que la importancia de las actividades STA en las reglas definidas en nuestro modelo son claves para la ejecución de estas. Por ello, para el buen funcionamiento del modelo usando la incertidumbre, es imprescindible una correcta definición de las probabilidades de todas las actividades STA presentes en la base de datos, para así conseguir una correcta ejecución de las reglas y el reconocimiento de la mayor cantidad de actividades LTA posibles.

4.4 Comparación de resultados en experimentaciones anteriores

Como ya hemos dicho, la definición de las probabilidades manualmente no proporcionaba una mejora significativa en el modelo realizado para el reconocimiento de actividades en los vídeos de vigilancia que se han utilizado. Aún y así, podemos sacar una serie de conclusiones de su ejecución, además de intentar comprender como sería su buen funcionamiento con el uso de la incertidumbre y compararlos con la ejecución del modelo en programación lógica sin el uso de la probabilidad.

En cuanto a los experimentos descritos anteriormente, podemos ver como no hay una diferencia significativa entre ambas ejecuciones debido a la manera de incluir la probabilidad en el segundo de estos experimentos, ya que su inclusión no afecta en los resultados obtenidos. La conclusión que se extrae de esto es que es importante la probabilidad en las actividades STA para que se tengan en cuenta todas las reglas en las que aparezcan y poder ejecutar cada escenario teniendo en cuenta la incertidumbre de que estas STA puedan aparecer o no. Como ya hemos dicho, para que se ejecuten las reglas de iniciación o terminación de una actividad será necesario que estas actividades aparezcan, y si su probabilidad es baja, la de la regla también lo será, al igual que ocurre en el caso en el que la probabilidad sea alta.

Por todo esto, la probabilidad de las STA, aparte de ser importantes para que se ejecuten las reglas, también lo son para obtener buenos resultados de reconocimiento, así que para el correcto funcionamiento del programa será indispensable que las probabilidades en los STA estén establecidas de la manera mas eficaz posible, a diferencia de lo que ocurre en

nuestra experimentación, ya que su inclusión manualmente no afecta favorablemente en el reconocimiento realizado.

También destacar que la definición manual de las distintas probabilidades en las actividades STA de los escenarios ejecutados, provoca que cada ejecución del modelo tenga que establecer estas probabilidades y así cada vez que se ejecute. Esto provoca que su velocidad de ejecución disminuya considerablemente, haciendo que uno de los beneficios que nos proporcionaba el uso del lenguaje Problog2 se vea afectado. Por ello es preferible la adicción de estas probabilidades tanto de manera más eficaz como antes de la ejecución de cada uno de los escenarios.

Suponiendo que se solucionan los problemas de funcionamiento del modelo usando la incertidumbre, podríamos obtener varias conclusiones comparándolos con el modelo del primer experimento.

Sabemos teóricamente que el experimento con ruido es mejor ya que se asemeja más a un problema del mundo real tratando la incertidumbre, pero a la hora de su ejecución, la pregunta que surge es que dónde se ven diferenciados un modelo del otro. La respuesta es en el número de conjuntos que utilicemos para reconocer cada actividad.

Para el caso en el que no se cuentan con múltiples iniciaciones de un LTA y no se trata la probabilidad, se ejecutará una de las reglas de iniciación que se cumpla, o ninguna de ellas en algunos casos; en cambio, al tratar la probabilidad, nunca se dejará una regla sin ejecutar. Esto es así ya que el reconocimiento de la actividad LTA se computará como la suma de todas las probabilidades de que ocurran cada una de las iniciaciones que tiene esa LTA, ya tengan una probabilidad de 0 o muy cercana a ese valor.

Por otro lado, también influye la cantidad de conjuntos probabilísticos de los que depende cada regla de iniciación, ya que, si depende de un número elevado de condiciones para la ejecución de cada regla, la probabilidad de que ocurra esta regla dependerá del producto de todas ellas. Al tratarse de un producto de hechos con probabilidades, cuanto mayor sea el número de conjuntos probabilísticos, menor será la probabilidad de que se ejecute esa regla, ya que por cada producto que se realice descenderá la probabilidad resultante. Este comportamiento en el caso del modelo sin ruido no se verá afectado, ya que únicamente tendrá una probabilidad de 1 si se cumplen todos los conjuntos pertenecientes a la regla y de 0 si al menos uno de ellos no se cumple.

En resumen, cada uno de los experimentos seguidos tienen sus ventajas y dependiendo del caso a tratar se preferirá el uso de uno por encima del otro. La elección dependerá de lo siguiente.

- Será preferible la **ejecución del modelo con incertidumbre** cuando tengamos una única iniciación de las actividades a reconocer y un pequeño número de conjuntos probabilísticos en sus reglas de ejecución.
- Por otro lado, será preferible la **ejecución del modelo sin técnicas probabilísticas** cuando contemos con múltiples iniciaciones para generar el reconocimiento de las actividades LTA, además de un número alto de conjuntos probabilísticos en cada una de las reglas que lo componen.

CAPÍTULO 5. GESTIÓN DEL PROYECTO

En este capítulo especificaremos la planificación que se ha seguido para el desarrollo de este trabajo, diferenciando entre cada una de las fases involucradas en el proyecto. Además, se incluirá el coste total de la realización del trabajo.

5.1 Planificación del proyecto

Para estudiar la planificación del proyecto realizado tendremos que tener en cuenta la duración que ha tenido el mismo y cada una de las fases que lo han compuesto.

5.1.1 Esfuerzo y duración del proyecto

El proyecto se comenzó a mediados del mes de marzo de 2019 y finalizó a principios de junio de 2019, por lo que se estima que la duración del proyecto fue de 80 días.

El esfuerzo dedicado por día será de una media de 3 horas diarias durante los primeros 40 días y una media de 5 horas diarias durante los siguientes 40 días, por lo que el total de esfuerzo estimado es de 320 horas de trabajo.

5.1.2 Fases del proyecto

Las fases de las que se compone el proyecto, a grandes rasgos, son las siguientes:

1. **Estudio e investigación**: en esta fase se incluirá un análisis exhaustivo del problema que se iba a tratar junto al estudio de las herramientas precisas para el desarrollo del sistema que se iba a realizar.
2. **Diseño y desarrollo:** en esta fase se realizaron todas las labores relacionadas con el sistema a desarrollar, que podemos dividir en las siguientes fases.
   - **2.1.** Instalación de los componentes software necesarios para el desarrollo del modelo.
   - **2.2.** Desarrollo del modelo de reconocimiento de actividades.
   - **2.3.** Pruebas y modificaciones necesarias para su correcto funcionamiento.
   - **2.4.** Análisis de los resultados obtenidos por parte del modelo.
   - **2.5.** Conclusiones obtenidas acerca de su funcionamiento.
3. **Documentación:** en esta fase se realiza la redacción de la memoria del trabajo.

5.1.3 Planificación realizada

FASE DEL PROYECTO	INICIO DEL PLAN	DURACIÓN DEL PLAN	INICIO REAL	DURACIÓN REAL	PORCENTAJE COMPLETADO
1. ESTUDIO E INVESTIGACIÓN	1	25	1	25	100%
1.1 Análisis del problema	1	20	1	20	100%
1.2 Estudio herramientas	21	5	21	5	100%
2. DISEÑO Y DESARROLLO	26	36	26	36	100%
2.1 Instalación software	26	2	26	2	100%
2.2 Desarrollo modelo	28	15	28	15	100%
2.3 Pruebas y modificaciones	43	4	43	4	100%
2.4 Análisis resultados	47	10	47	10	100%
2.5 Conclusiones obtenidas	57	5	57	5	100%
3. DOCUMENTACIÓN	62	19	62	19	100%

Ilustración 15. Fases del proyecto con día de inicio, duración de la fase y porcentaje completado

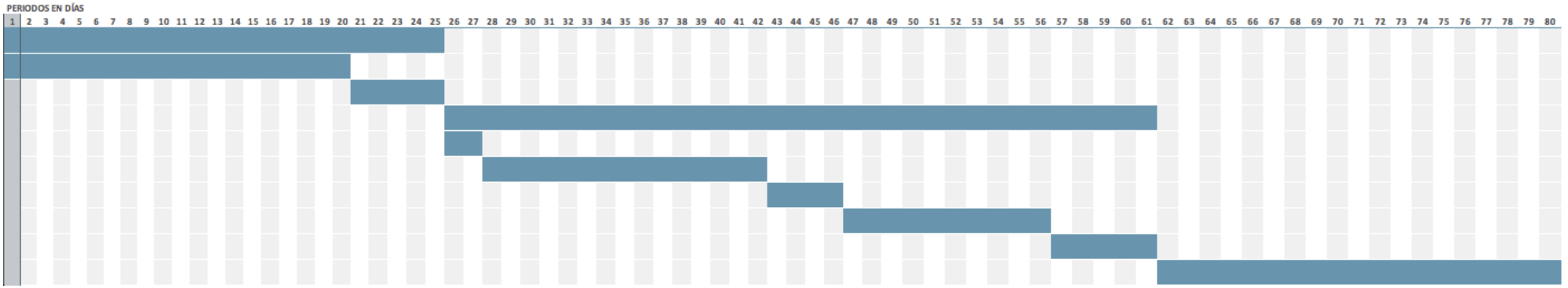

Ilustración 16. Diagrama de Gantt

Como hemos podido observar en el diagrama de Gantt anterior, la duración del trabajo se divide por cada día desarrollado según las líneas descritas, correspondiéndole al estudio una duración de 25 días, al desarrollo del modelo una duración de 36 días y concluyendo finamente con la confección de este documento durante 19 días.

5.2 Costes del proyecto

Para la planificación del de los costes del proyecto deberemos diferenciar los diferentes presupuestos invertidos durante el proyecto, siendo estos presupuestos del personal y el presupuesto invertido en las herramientas utilizadas para la realización del proyecto.

5.2.1 Presupuesto del personal

Teniendo en cuenta las labores que se deben realizar para hacer este proyecto y el sueldo medio anual de un desarrollador que lo lleve a cabo, se estima que el gasto que se tiene para el presupuesto del personal es de 20€ la hora de trabajo realizada.

Por esta razón y teniendo en cuenta que se ha trabajo durante 320 horas, pero durante los primeros días se han invertido menos horas, el gasto invertido en el personal será el que se muestre en la siguiente tabla.

Fase del proyecto	Horas invertidas	Coste en euros (€)
Estudio e investigación	75 horas	1500 €
Diseño y desarrollo	150 horas	3000 €
Documentación	95 horas	1900 €
TOTAL	320	**6400 €**

Tabla 24. Presupuesto del personal

Donde se han invertido para la primera fase 25 días (75 horas) para la segunda 39 días (45 horas los primeros 15 días y 105 horas los siguientes 21 días) y par la última 19 días (95 horas).

5.2.2 Presupuesto del material utilizado

El material utilizado para la realización de este proyecto consta únicamente de un ordenador, a través del cual se ha realizado el código y las pruebas del sistema junto a la redacción de este documento.

Si consideramos que el ordenador tiene una vida útil de 3 años y que su coste ha sido de 1099 €, el coste que se le asemeja al ordenador según el uso de 80 días que le hemos dado es el siguiente.

Descripción	Coste	Vida útil	Coste mensual	Uso	Coste por uso
Ordenador	1099€	3 años	30,52€	2,6 meses	79,35€
				TOTAL	**79,35€**

Tabla 25. Presupuesto del material utilizado

No se consideran otros gastos ya que las licencias de uso en el ordenador y la obtención de la base de datos no supusieron ningún coste.

5.2.3 Coste imputable total

Teniendo en cuenta los costes anteriores, el coste imputable que le corresponde a la realización de este proyecto, considerando que generará un beneficio del 20%, es el siguiente.

Descripción	Coste en euros (€)
Coste del personal	6400 €
Coste del material	79,35 €
Beneficio (20%)	1295,87 €
IVA 21%	1632,80 €
TOTAL	**9408.02 €**

Tabla 26. Coste total del proyecto

El coste total del proyecto será de **NUEVE MIL CUATROCIENTOS OCHO EUROS Y DOS CÉNTIMOS.**

5.3 Impacto socio-económico

El impacto socio-económico de este proyecto, considerándolo como una aproximación o primer acercamiento a un sistema real, apenas contará con un impacto considerable. El impacto económico que tendrá será mínimo y todo ello derivado del beneficio económico que proporciona la realización de este proyecto. En cuanto al beneficio social, tampoco se tendrá muy en cuenta porque no tendrá un gran uso en la vida real.

Considerando la aplicación que podría llegar a tener este proyecto si se desarrollara por completo como ya hemos mencionado, su impacto sería mucho mayor. Además de que el beneficio económico que proporcionaría sería mucho más alto, tendría una repercusión social mucho mayor ya que:

- Permitiría mejorar el desarrollo de los trabajos de vigilantes de seguridad e incluso, en casos más avanzados, sustituirles.
- Dependiendo del uso que se le de a este sistema y en qué casos se utilizará para detectar actividades, mejoraría la seguridad de las personas a las que les afecta directamente.

De igual manera, si uso es velar por la seguridad, también tendrá un impacto ético hacia las personas a las que les afecte positivamente el uso de este sistema.

CAPÍTULO 6. CONCLUSIONES Y FUTUROS TRABAJOS

En este capítulo hablaremos de las conclusiones obtenidas acerca de la realización de la investigación durante el trabajo, el modelo creado y ejecutado y los resultados obtenidos tras su ejecución, además de incluir las líneas futuras en las que se puede enmarcar este trabajo.

6.1 Conclusiones obtenidas

Para desarrollar las conclusiones de este Trabajo de Fin de Grado de una manera más amplia y ordenada diferenciaremos por un lado los objetivos a los que se querían llegar en un inicio de su desarrollo, estudiando si hemos cumplido con lo esperado o no y las razones en cada caso, además de unas conclusiones generales que engloben todo el trabajo. Después especificaremos las conclusiones personales a las que hemos llegado y las asignaturas cursadas durante el grado que nos han ayudado a realizar este proyecto.

6.1.1 Capacidades lógica probabilística y lenguaje de programación Problog2

Una vez observado el uso que se le da a la lógica para abordar problemas y resolverlos haciendo uso del sentido común, y las capacidades que tiene esto para ser llevado a un sistema que sea capaz de resolverlo automáticamente, nos damos cuenta de que no existe un límite al que llegar a la hora de desarrollar un modelo. Sus capacidades son infinitas y a medida que hay avances profundiza y mejora cada vez más sus facultades.

Y como hemos visto, el hacer uso de la programación lógica probabilística para que sus aptitudes sean mejoradas gracias a estos avances en el tiempo y, a su vez, tengamos la posibilidad de que se comporten como si se tratara de pensamiento humano, hacen que para ciertos tipos de problemas su resolución sea mucho más eficiente que incluso la capacidad del mejor de los pensamientos humanos.

Para el desarrollo de esta labor se ha comprobado también que el uso del lenguaje *Problog2* es más que aceptable ya que nos permite desarrollar los problemas de la manera más eficiente posible haciendo uso de la incertidumbre, presente en problemas en el mundo real.

6.1.2 Comparación Problog2 con otros lenguajes

Viendo las capacidades que tiene *Problog2* en comparación con las que presentan otros lenguajes de programación lógica probabilística, se ha llegado a la conclusión de que el uso de este lenguaje es correcto gracias a las capacidades que presenta, a su eficiencia e incluso su simplicidad. Quizá en algunos problemas su uso no sea el más correcto, pero para las aplicaciones vistas durante este proyecto su utilización nos ha ayudado bastante.

Además, la característica de tratarse de un lenguaje relativamente nuevo en el tiempo y que a día de hoy se siga mejorando y perfeccionando gracias a una comunidad por Internet en la que se exponen y se resuelven problemas hace que su uso tenga un mayor impacto social, ya que se trata de una herramienta que no se sabe hasta qué punto será capaz de llegar.

En el caso de este Trabajo de Fin de Grado, gracias a su simplicidad, la realización del modelo ha sido muy intuitiva, además de tratarse de una extensión del lenguaje de programación lógica *Prolog*, visto en alguna asignatura de la carrera, por lo que algunas de sus definiciones ya las conocíamos.

6.1.3 Desarrollo modelo reconocimiento actividades usando Problog2

El desarrollo del modelo para el reconocimiento de actividades no fue fácil debido a la falta de información que hay acerca de este lenguaje de programación, ya que no es del todo conocido, incluyendo además la falta de información en cuanto a vídeos de vigilancia y datos acerca de ellos.

Se pensó en abordar el trabajo anotando manualmente los datos de entrada o intentando obtener los datos a partir de un sistema de aprendizaje que tratara las imágenes de los vídeos, pero esta labor era demasiado costosa y su realización no tenía mucho que ver con el trabajo que se estaba realizando, además de no contar tampoco con los valores probabilísticos en sus definiciones. Por ello adaptamos en cierta medida los datos de entrada de nuestro modelo para que se asemejaran a los que proporcionaba la base de datos de vídeos de vigilancia CAVIAR.

De esta manera el desarrollo del modelo fue mucho más intuitivo y nos permitió la generación de un modelo que se ejecutaba correctamente y que, como es este caso, a partir de los datos de entrada proporcionados, reconocía qué actividades se realizaban en cada caso, tanto para una versión del modelo sin datos probabilísticos como para la modificación en la que contar con esta incertidumbre presente en la vida real.

6.1.4 Lenguaje útil a partir del análisis del modelo realizado

El análisis de los resultados obtenidos a partir del modelo desarrollado nos permitió ver que el sistema tiene potencial para desarrollarse de manera completa para ser capaz de reconocer actividades que ocurren en vídeos de vigilancia mediante el uso de *Problog2*, pero para que sea útil y tenga un rendimiento óptimo es necesario un proyecto mucho más elaborado, el cual cuente con un modelo mucho más exhaustivo que tenga en cuenta muchas más condiciones, una base de datos de la misma manera mucho más completa y quizá un método de reconocimiento de imágenes instantáneo, para que su uso sea a tiempo real.

Además, a todo lo descrito anteriormente se debe incluir un método para establecer las probabilidades según la importancia que tengan cada hecho que se trate en cada momento para el problema a tratar, para hacer que el modelo se ejecute correctamente teniendo en cuenta la incertidumbre.

6.1.5 Conclusiones generales

En general, el modelo creado nos ha permitido comprobar las capacidades que tienen los sistemas de reconocimiento mediante el uso de la programación lógica probabilística, como es este caso, donde simplemente con una base de datos completa, pero sin una gran cantidad de datos diferenciados entre sí, y una relación de reglas y hechos en la que se engloben estos datos, hemos sido capaces de reconocer actividades correctamente, incluso con un acercamiento mediante el uso de probabilidades tratando así la incertidumbre en sistemas de este estilo.

Como hemos dicho anteriormente, si contáramos con una base de datos mayor, que englobara más reglas, y mediante el uso correcto de la probabilidad, el sistema podría ser utilizado por cualquier empresa que precisara de un sistema de reconocimiento de actividades de este tipo, para su uso en seguridad, mejorar la calidad de los trabajadores que se dediquen a esta labor o simplemente con un fin estadístico.

El uso de *Problog2* para el desarrollo del modelo ha facilitado su realización, ya que su sintaxis es muy clara, asemejándose fácilmente a la lógica formal, y todas las herramientas con las que cuenta para su ejecución facilitan y permiten un entendimiento del problema mucho mayor, otorgando la posibilidad de ejecutar un mismo problema de formas distintas, según lo que requerimos en cada momento. Por ello, la elección de *Problog2* para desarrollar el modelo en este trabajo ha sido efectiva.

A partir de este punto hay que buscar la manera de progresar en el reconocimiento de actividades para innovar y mejorar lo que se está haciendo hasta día de hoy, ya que para que un modelo de este estilo sea usable no importa en cierta medida el coste que pueda llegar a suponer el realizarlo. Lo verdaderamente importante será la eficiencia y rapidez de respuestas con la que cuenta, que es lo que realmente importa en este tipo de sistemas, en los que se buscan respuestas válidas y rápidas, y por lo que la gente en definitiva acaba utilizándolos y prefiriéndolos a otros que pueda encontrar en el mercado.

6.1.6 Conclusiones personales

Gracias a este trabajo y a las investigaciones realizadas para su desarrollo he aprendido como se desarrolla el tratamiento de imágenes de vídeos para su posterior uso en forma de datos, y el uso de estos datos para el reconocimiento de patrones. Esto también me sirvió para descubrir lo fácil que resulta el reconocimiento de actividades si se cuenta con un buen modelo y una buena base de datos y, sobretodo, un buen lenguaje de programación.

Finalmente, gracias a *Problog2* he aprendido las capacidades que tiene un lenguaje de programación lógica probabilística y todo lo que proporciona, ya que ha medida que realicé este trabajo se me ocurrieron muchas aplicaciones en las que usar este lenguaje para generar sistemas de reconocimiento y otras muchas aplicaciones.

6.1.7 Relación del proyecto con asignaturas cursadas

Las asignaturas que he cursado a lo largo del grado de Ingeniería Informática que me han permitido la realización de este Trabajo de Fin de Grado son las siguientes:

- **Lógica**: proporcionó las bases de los conceptos de la lógica formal.
- **Técnicas de búsqueda y uso de la información y expresión oral y escrita**: necesario para la correcta escritura y realización del trabajo.
- **Estadística**: necesario para saber tratar los resultados obtenidos durante el análisis.
- **Fundamentos de gestión empresarial**: necesario para la realización de la gestión del proyecto.
- **Teoría de autómatas y lenguajes formales**: proporcionó las nociones básicas para entender la complejidad computacional.
- **Inteligencia artificial**: proporcionó conocimientos de inteligencia artificial para el desarrollo del trabajo.
- **Aprendizaje automático**: necesario para entender de qué forma reconocía actividades el modelo.
- **Redes de neuronas artificiales**: necesario para entender en qué consisten las redes de neuronas y cómo se utilizan para el aprendizaje.
- **Ingeniería del conocimiento**: para entender cómo funcionan los sistemas expertos y cómo utilizan el conocimiento.

6.2 Problemas encontrados

Durante la realización del trabajo nos hemos encontrado con una serie de problemas, los cuales algunos hemos conseguido subsanar y otros no. Entre todos ellos destacamos las siguientes.

En primer lugar, la utilización del modelo sin contar con la base de datos de CAVIAR. En una primera aproximación del trabajo se intentó ejecutar el modelo sin un uso externo de estos datos, incluyéndolos todos ellos a mano, pero para que se realizara una buena ejecución del problema es necesario una gran cantidad de estos datos, por lo que finalmente se recurrió al uso de esta base de datos.

Por otro lado, fue un problema la falta en la versión de *Problog* utilizada de la capacidad de generar variables aleatorias continuas o de distribuciones estadísticas para desarrollar el problema completo en programación lógica probabilística. Al no contar con esto, como se proponía en el párrafo anterior, se tuvo que realizar la asignación de probabilidades a mano, siendo esta meramente una versión simplificada del modelo completo.

También fue un problema la poca variedad de atributos en la base de datos de CAVIAR, que hacía que las reglas utilizadas no fueran nada permisivas, ya que no contaba con una

amplia variedad de posibilidades para su ejecución en cada regla, por lo que el reconocimiento en este caso se vio perjudicado.

Por último, lo desconocido que es en la actualidad el lenguaje de *Problog* hace que sea complicada la búsqueda de soluciones cuando se tiene un problema. Aún y así, a favor de esto cabe destacar que los desarrolladores del lenguaje cuentan con un blog en el que participan activamente muchos integrantes del equipo desarrollador además de programadores que utilizan el lenguaje en diferentes partes del mundo a los cuales se les puede realizar consultas sobre algún problema que tengamos y a las que contestarán lo más rápido posible, razón a favor de su uso a pesar de no ser muy conocido mundialmente.

6.3 Trabajos futuros

A partir del punto en el que se ha dejado el proyecto se mencionan los siguientes desarrollos que se podrían realizar para continuar este trabajo en un futuro.

- Uso de datos de entrada del modelo con probabilidad establecida con sentido en todos ellos para la ejecución probabilística del modelo de la manera más real posible.
- Diferentes tipos de ruido para que el problema sea más completo, como la adicción de datos de entrada falsos o incluir probabilidades en otros datos extra como la orientación o las coordenadas.
- Uso de bases de datos más completa a partir de la cual poder caracterizar más tipos de atributos, tanto para usar como datos de entrada como para el reconocimiento de actividades.
- Ejecución del problema como un reconocimiento de actividades en tiempo real, mediante el uso de técnicas más avanzadas que traten tanto el reconocimiento de imágenes como el de actividades más eficientemente.
- El uso de herramientas de Problog, como por ejemplo ‘lfi’, aprendizaje mediante interpretaciones, para designar las probabilidades a los hechos de entrada a partir de evidencias de datos. Así se podría establecer qué hechos son importantes, usando esta probabilidad generada como el peso de cada uno de los hechos del problema.

CAPÍTULO 7. ENGLISH COMPETITIONS

This chapter will summarize the work that has been done along this document, including the parts of which it is composed and specifying the introduction, objectives, results obtained and conclusions reached.

7.1 Introduction

In the current world, the use of learning to solve problems that arise in real life has an ever-greater impact. For centuries we have the ability to solve the problems that arise, whether decision making at an opportune time or the solution to a problem.

To solve these problems, we must use logic, which will allow us to draw conclusions from the problems that appear, whether they are true or false, so we will acquire knowledge about these problems for a future problem with the same characteristics. This concept related to logic is known as learning.

The need to solve problems together with the technological advances of recent years makes programming appear, which is capable of finding solutions to our problems in an autonomous way.

The impediment to programming is that there are problems that can't be solved without the use of human logic. This is why logical programming appears, a concept of programming where classical declarative programming is unified with the concepts of formal logic that are known today.

It's true that logical programming provides us with the necessary capabilities to solve real-life problems, but it doesn't look like a real problem. This is because in real life you can never be sure that something will ever happen, you must consider the uncertainty.

The way to deal with uncertainty through the use of logic programming is adding the concept of probability, causing the probabilistic logic programming to be defined.

To see the capabilities of probabilistic logic programming, the Problog2 language is used. This is an extension of Prolog's language and its operation is based on it, but including dealing with uncertainty. Throughout the document we will see the functioning of this language and the competences that it presents.

The main motivation for this topic has been chosen for the realization of the TFG is to deepen the study of Artificial Intelligence, studied throughout the university degree and that currently has a considerable importance thanks to the continuous advances that appear, even to this day, that make it difficult to establish a limit of the capacities that it presents.

In addition to this, another important motivation for the study of Problog2 is that it's one of the most modern languages that deal with probabilistic logic programming, and for that reason they have a blog in which the language developers are and in which problems are discussed to improve the language capabilities until today, so we can say that the language is 'alive' today and continues to improve and perfecting.

Our model will be based on the recognition of activities from data already processed from surveillance video databases. The main objectives to be reached are the following.

1. Investigate the capabilities of probabilistic logic and the programming language Problog2.
2. Compare Problog2 with other probabilistic logic programming languages.
3. Develop one of its applications by creating a system that recognizes human activities in surveillance videos, developing this model in the Problog2 language.
4. Analyze the results of the model from the activities that the model recognizes and get conclusions about whether this language is really usable.

For the correct development of the work, its structure will be as follows.

1. **Introduction**: the work is presented.
2. **State of the art**: study to know where our project is and from what point we have to develop it.
3. **Design and modeling of the system**: explanation of the strategy and design followed to make the model, the database of images used and finally, its implementation in Problog2.
4. **Analysis of the results**: study of what has been done on the results obtained in the execution of the model.
5. **Conclusions and future lines**: final considerations of the project and future lines to consider.

7.2 State of the art

Logic is the science that studies the why of things, the methods and the reasons for the truth of ideas, distinguishing right from wrong. Within the logic we must know how to deal with the uncertainty that can be generated in the problems that are addressed. To deal with uncertainty, methods such as:

- Dempster-Shafer theory of evidence.
- Fuzzy logic based on previous experiences to add value to the facts presented by the problem.
- Factors of certainty, to make sure that something will be true in an established degree.

The use of logic to generate conclusions from a series of premises combined with the ability to deal with the uncertainty that appears in real life, makes it appear the concept of **inference**, ability to create conclusions from premises. There are two types of inference:

- **Inductive Inference**: rules are generated from the premises of the observed world.
- **Deductive inference**: hypothesis or ideas based on general norms already defined.

The probabilistic reasoning will be obtained from the values given to that set of premises and from them we will obtain the probability that the conclusions will occur.

For the development of probabilistic logic programming we will use the language of Problog2. This is an extension of Prolog so we will define this language first.

Prolog is a logical programming language and is based on the following:

- Declare facts as absolute truths about the objects you are dealing with.
- Define a series of rules about the facts.
- Ability to perform queries.

The elements that a Prolog program has are facts and rules. The facts are logical relations that are always true. They are declared with the name that is attributed to the relation and between parentheses all the objects that it relates, which can be individual or multiple, as well as having qualitative or quantitative values.

The rules consist of a conditional function that expresses that, if each of the included objectives, which may be facts or some numerical condition, are fulfilled, then the fact that defines the rule will also be true.

The rules consist of a conditional function that expresses that, if each of the included objectives, which may be facts or some numerical condition, are fulfilled, then the fact that defines the rule will also be true.

The objectives included in each rule are represented as conjunctions between them, so their behavior will be based on the use of **Horn clauses**. On the other hand, variables are used to replace the unknown arguments of the rules. In this way, those conditions that are treated with variables will cause that all the facts defined for that relationship will be checked and will verify for which of them the rule is fulfilled.

The general structure of the program follows the following organization.

- In the first place, the facts are declared with valid arguments.
- Afterwards, a series of rules are declared with their conditions that must be met.
- Finally, the queries are declared to make the verifications that we want from our program.

The deduction that Prolog follows to solve problems can be differentiated in this way:

- According to a propositional logic: case in which there is only a single answer and only propositions without variable values are used.
- According to a relational logic: we have variables besides constants, relationships and quantifiers.

- According to a functional logic: as in the previous case, but adding functions in the rules.

The strategy that Prolog follows to solve the queries that are made is the following:

1. The queries that is carried out is made up of several facts and to confirm the consultation all of them must be fulfilled. All these facts that have to be confirmed will be stored in a list.
2. In order to confirm these facts, rules that satisfy each of them will be sought. In the same way, to satisfy that these rules are met, we must also confirm the facts that compose them. If a rule confirms one of the facts on the list, this fact will be replaced by all those facts that make up the rule.
3. This will be repeated continuously until you reach a fact that belongs to the list that is known to be true and then it will be removed from the list, which means that it has been confirmed.
4. If all the facts that make up the list can be confirmed in the same way as in the previous point, by eliminating each of these facts from the list we will end up with an empty set, which means that we have confirmed the query made. On the other hand, if we have gone through all the facts and rules of our program and there are still facts that belong to the list, it will imply that the query is not true.

This resolution is known as SLD (Selective Linear Definite). We emphasize that when each of the facts and the rules are covered as they appear, it will import the order in which they are defined, since the last one that appears will be the last one that will be executed. Furthermore, this path will be carried out through these rules and facts arranged in the form of a tree and its way of traversing it will be in **depth**, where each branch of the tree will be traversed until finding the empty set (success) or a dead end (failure) and after go back (**backtracking**) to the next branch.

In this way, the whole tree will always be traversed and we will obtain all possible valid answers to our query.

Once presented the operation of Prolog, we will see its extension, which is what we will use in our model.

Problog2 is the modernized improvement of the Problog language and is an extension of the language that we have developed in the previous point. It is based on logical programming, semantic distributions and probabilistic models and graphs. The modeling of Problog2 has two points of view.

- **Probabilistic**, where probabilities are defined on certain true values of a subset of program atoms.
- **Logical**, which derives the true values of the remaining atoms using a reasoning similar to that of Prolog. For these cases the clauses defined above are used.

The main difference with Prolog is that Problog2 uses a probabilistic reasoning to give the answers, which are not true or false, but have a degree of probability that they are true.

The aspects that define the skills of Problog2 are the following:

- Allows marginal inference, that is, calculate the marginal probabilities of the queries based on certain evidences.
- It makes it possible to learn program parameters based on data in the form of partial interpretations.
- Solve theoretical decision problems.
- Allows sampling of all possible worlds.
- Interacts with the Python language.

The structure of Problog2 compared to Prolog's is very similar. In this language, both the facts and the rules have the ability to be defined from a probability that they are valid, they will have a number that indicates the degree of probability that presents.

0.25 :: person (tom).

This language also changes when defining the program's queries.

The queries are made thanks to the definition of the predicate ***query(Q),*** where Q will be the query from which the probability of its occurrence will be shown. The queries can also be influenced by evidences of the form ***evidence(E***), where E will be the evidence that we want to be always true.

The inference made in Problog2 to obtain the probabilities of the queries, step by step, is the following:

1. All the definitions that influence the query are gathered, looking for them in the **SLD** tree.
2. Once located, all the tests are represented as a **DNF** (Disjunctive Normal Form).
3. Through a script already developed in the language, the DNF is transformed into a **BDD** (Binary Decision Diagrams).
4. Finally, the probability is calculated from this BDD.

The DNF’s are a transformation of the probabilities of the SLD, but eliminating redundancies and compacting everything in a single formula, in the form of disjunction of conjunctions. Its probability will be equal to the probability that at least one of them will be made.

The BDD's are a way to represent the DNF formulas in a more compact way. It is a decision tree in which the redundant nodes are eliminated. Each node of this tree is the

probabilistic facts of our program and each of them has two outputs, one positive and one negative, for each possible output.

Problog2 as an autonomous tool, allows its execution in different ways depending on the use we want to give to its execution engine. In this section we will differentiate each of them.

- **Standard inference in Problog**: in this way Problog executes the model that is provided to it and computes the probability of its queries, showing them as output of the execution. This mode is executed by default. This probability calculation can be performed by SDD (Sentential Decision Diagram).
- **Sampling**: in this mode Problog selects a set of individuals that will run as a sample of the population to obtain the probability of the queries made by reducing workload, since only part of the total will be executed.
- **Sampling based on inference**: in this case, samplings will be carried out in the same way as in the previous case, but they will be executed according to some conditions and the result will be the average of all of them.
- **Most Probable Explanation (MPE**): The most probable world is computed in which all the queries and evidences described are true.
- **Learning from interpretations**: in this mode, the probabilities of certain events whose probability is unknown are determined based on certain declared evidences. The probability of these events may initially be totally unknown or initialized with some value.
- **Theoretical decision**: extension of Problog called **DTProblog** that differs from this in that it does not have evidence or queries. The facts must decide the probability that these will be implemented and atoms are declared according to their usefulness in the program, indicating its contribution to the final execution of the program.
- **MAP inference**: similar to MPE, but in this case it is necessary that the facts with probabilities are specified as explicit queries.

Problog2 addition there are many other programming languages probabilistic logic that we can use but have limitations that affect their usefulness.

For example, there are **PRISM** and **PHA**, which behave like Problog2, but only handle probabilities on specific facts, without the use of variables, besides excluding certain combinations of facts from being true simultaneously.

On the other hand, there is **pD** (**Probabilistic Datalog**), that its operation is very similar to Problog2, but it is more complex to understand, both for the programmer and for the user who uses the language. In addition, its use has a perspective more linked to databases than logical probability queries.

The problem with these languages is their difficulty in understanding and handling for the user. Problog solves this problem thanks to its independent probabilities, which allow describing the problem's relations easily. Another characteristic unlike the rest is that it allows the calculation of probabilities in clauses, not only in predefined facts. This problem PRISM and PHA solve it by restricting the execution of these special cases.

Once we have seen the different languages, we will see the use of the databases from which we will obtain the activities to be recognized.

In our case we will use CAVIAR (Context Aware Vision Image Active Recognition). This is a project in which recorded images of different scenarios are collected with the idea of processing what happens in these images to obtain a series of data that can be stored in the database.

The different data collected in the CAVIAR videos are basically focused on the recognition of people walking, greeting each other, opening windows, entering and leaving stores, fighting, fainting and leaving objects in public places. In addition, each activity will have data such as the orientation and coordinates of these people.

The recorded videos will have the following situations:

- Three videos showing that a person is walking.
- Six videos where people are looking for certain objects.
- Four videos where people do not move (resting or fainting).
- Five videos in which people leave objects.
- Six videos where people greet each other, walk together and separate.
- Finally, four videos where people discuss among themselves.

In conclusion, for the development of the model we will use the Problog2 language and the CAVIAR database to recognize the activities carried out by the people who appear in their videos.

7.3 Design and model

The strategy that will be followed step by step to carry out the program will be the following:

1. **Design of the model**.
2. **Implementation of the mode**l designed using the language of Problog2, with the necessary rules and facts.
3. **Choice of data from the CAVIAR database** that we will use.
4. **Handle these data as inputs** to the program.
5. **Carry out appropriate queries** according to the activities we want to check.

6. Process the request from these queries, **going through the rules**.
7. **Show the results** after the execution of the model.
8. **Analyze the results** obtained by checking whether the queries carried out are fulfilled or not.
9. **Analyze the different experiments and draw queries from them**.

For the development of this strategy we will start from the following points:

- We will use a base template of predicates that will help us in the recognition of activities over time, called Event Calculus.
- The rules of the model will be developed according to the understanding we have of the conditions that must have for them to be executed.
- The activities that will be in our model will be based on the CAVIAR database.

Event Calculus is a set of predicates that are used to recognize the effects that actions cause over time. For the generation of our model using Event Calculus we will differentiate between events and fluent.

The ***events*** consist of actions that are always executed, used as system inputs, which are:

- **STA (Short Term Activities**). Activities of which the surveillance video database is composed. These will be: *active, inactive, walking, running* and *abrupt*. We will generate the latter manually.
- The predicates *appear* and *disappear*, which indicate if something begins to be tracked in the video or stops being tracked.

The ***fluentes*** will be those events that are executed but that are not fixed since they can change over time due to different actions that are executed on them. This definition refers to the outputs of our program, defined as **LTA (Long Term Activities**), which will be the activities of *moving, meeting, leaving_object* and *fighting.*

The structure that the program will have will be the following:

1. The facts that we will treat as input data of our program, obtained from CAVIAR. In each execution of the program we will choose each set of facts.
2. The rules that will form the model on which we will base ourselves to infer the activities that we want to recognize, as each activity is initialized or finalized in each time frame.
3. Finally, the queries we want to make to infer recognition activities.

The complete structure can be seen in the annexes of the document.

The facts, provided by CAVIAR, will be obtained from the following files of the database downloaded from its website:

- *AppearenceIndv.pl*: orientation information and the predicates appear and disappear.
- *MovementIndv.pl*: information about STAs and coordinates.

The videos that we will use for each activity will be the following:

- MOVING, WALKING: wk1gt, wk2gt and wk3gt.
- LEAVING_OBJECT: lb1gt, lb2gt, lbbcgt, lbgt and lbpugt.
- MEETING, WALKING: mwt1gt, mwt2gt, mws1gt, ms3ggt, mc1gt and spgt.
- FIGHTING: fra1gt, fra2gt, fomdgt1, fomdgt2, fomdgt3 and fcgt.

For the execution of Problog2 we will download the most modern version (2.1.0.36) and to install this version we will need a Python version higher than 3.6 (we will use 3.7). It will be installed using the command:

pip3.7 install problog

Once installed, the tool can be used, executing the model generated from all the facts and rules defined throughout the document, implemented in the language of Problog2.

7.4 Analysis of experiments

To check how the model works we will analyze two different experiments:

- **Experimentation without artificial noise**. We will treat a problem of logical programming, where both the events and the activities to be recognized will run without probabilities, executing as true or false.

- **Experimentation with artificial noise**. In this case it will be a probabilistic logic programming problem, where certain data of the problem will be processed to run with probability, to generate this uncertainty that we have about the activities. Similarly, recognized LTA activities will also have some probability. The data in which the probabilities will be included manually will be the following:
    - The rules defined in the model.
    - The facts that aren't executed in each video frame, with a low probability since the generated STA predominates.

The execution of each experiment will be done by specifying the videos that will be executed in each case and the *frames* that will be recognized for each execution, based on the real LTA activities that should be recognized in each of them. This information is also provided by CAVIAR.

Once both experiments are executed, we will obtain the following conclusions about the analysis:

- **First experiment**: many activities aren't recognized due to their low similarity with a real-world problem, since uncertainty isn't dealt with and in this way, the rules are very restrictive to allow the execution of several of them.
- **Second experiment**: manually including the probabilities, it will be executed as a probabilistic logic program, but it won't follow the desired operation since the probabilities aren't well weighted, besides the rules are still restrictive. Therefore, a greater number of recognized activities will be generated, but the accuracy will remain the same, taking into consideration the recognition threshold that is established to differentiate recognized activities from those that aren't recognized.

In addition, observing its behavior, the execution of the **model with uncertainty** will be preferable when we have a single initiation of activities and a small number of probabilistic sets in its rules, while the execution of the mod**el without probabilistic techniques** will be better when we have multiple initiations and a large number of sets in each probabilistic rule.

7.5 Conclusion and future work

Concluding, the model created has allowed us to verify the competences that recognition systems have by using probabilistic logic programming, using probabilities and thus treating the uncertainty in systems of this style.

If we had a larger database, which includes more rules and through the correct use of probability, the system could be used by any company that needs a system to recognize activities of this type, for its use in security, improve the quality of the workers who dedicate themselves to it or simply with a statistical purpose.

The use of Problog2 for the development of the model has facilitated the work, since its syntax is very clear, easily resembling formal logic, and all the tools it has for its execution facilitate and allow a better understanding of the problem, granting the possibility of executing the same problem in different ways.

From this point we must look for ways to progress in the recognition of activities to innovate and improve what is being done to this day. What needs to be improved is the efficiency and speed of response it has, which is what really matters in this type of systems in which valid and fast answers are sought, and for what people end up using them.

Future work to follow this project could be the following:

- Use of model input data with probability established in all of them for the probabilistic execution of the model in the most real way possible.
- Different types of noise to make the problem more complete, such as the addition of false input data or to include probabilities in other extra data such as orientation or coordinates.
- Use of more complete databases from which to include more types of attributes.
- Execution of the problem as a recognition of activities in real time, through the use of more advanced techniques that treat both the recognition of images and the activities more efficiently.
- The use of tools such as learning through interpretations (lfi) of Problog2 to designate the probabilities to input facts from data evidence. This could establish what facts are important, using this probability generated as the weight of each of the facts of the problem.

BIBLIOGRAFÍA

[1] R. Kowalski, *Logic for problem solving*, vol. 7. Ediciones Díaz de Santos, 1979.

[2] J. W. Lloyd, *Foundations of logic programming*. Springer Science & Business Media, 2012.

[3] "ListServ 16.0 PROBLOG Home Page." [Online]. Disponible: https://ls.kuleuven.be/cgi-bin/wa?A0=PROBLOG. [Acceso: 10-Jun-2019].

[4] A. Sánchez, "El impacto de la nueva LOPD en el ámbito laboral: nuevos derechos digitales," Prisacom, Madrid, 15-Jan-2019.

[5] "Normativa de utilización aulas informáticas," 03-Mar-1993. [Online]. Disponible: https://bit.ly/2QyBhCi. [Acceso: 10-Jun-2019].

[6] A. Deaño, *Introducción a la lógica formal*. Alianza, 1975.

[7] E. Álvarez Saiz, *Modelos probabilísticos para utilización en sistemas expertos*. Universidad de Cantabria, 2011.

[8] D. V. Lindley, "The probability approach to the treatment of uncertainty in artificial intelligence and expert systems," vol. 2, no. 1, pp. 17–24, 1987.

[9] G. A. Sharer, "A mathematical theory of evidence. Princeton," vol. 1, p. 976, 1976.

[10] L. A. Zadeh, "Fuzzy sets," vol. 8, no. 3, pp. 338–353, 1965.

[11] E. H. Shortliffe and B. G. Buchanan, "A model of inexact reasoning in medicine," vol. 23, no. 3–4, pp. 351–379, 1975.

[12] E. H. Shortliffe, R. Davis, S. G. Axline, B. G. Buchanan, C. C. Green, and S. N. Cohen, "Computer-based consultations in clinical therapeutics: explanation and rule acquisition capabilities of the MYCIN system," vol. 8, no. 4, pp. 303–320, 1975.

[13] E. Shortliffe, *Computer-based medical consultations: MYCIN*, vol. 2. Elsevier, 2012.

[14] N. J. Nilsson, "Probabilistic logic," vol. 28, no. 1, pp. 71–87, 1986.

[15] G. Casella and R. L. Berger, *Statistical inference*, vol. 2. Duxbury Pacific Grove, CA, 2002.

[16] L. E. Sucar, "Redes bayesianas," pp. 77–100, 2006.

[17] R. E. Neapolitan, *Learning bayesian networks*, vol. 38. Pearson Prentice Hall Upper Saddle River, NJ, 2004.

[18] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," vol. 77, no. 2, pp. 257–286, 1989.

[19] L. De Raedt and K. Kersting, "Probabilistic logic learning," vol. 5, no. 1, pp. 31–48, 2003.

[20] E. Bottieau, J. Moreira, J. Clerinx, R. Colebunders, A. Van Gompel, and J. Van den Ende, “Evaluation of the GIDEON expert computer program for the diagnosis of imported febrile illnesses,” vol. 28, no. 3, pp. 435–442, 2008.

[21] E. Y. Shapiro, “Logic Programs With Uncertainties: A Tool for Implementing Rule-Based Systems.,” presented at the Ijcai, 1983, vol. 83, pp. 529–532.

[22] W. F. Clocksin and C. S. Mellish, *Programming in Prolog: Using the ISO standard*. Springer Science & Business Media, 2012.

[23] K. Ueda, “Guarded horn clauses,” presented at the Conference on Logic Programming, 1985, pp. 168–179.

[24] H. Boström and P. Idestam-Almquist, “Specialization of logic programs by pruning SLD-trees,” presented at the Proc. of the Fourth International Workshop on Inductive Logic Programming (ILP-94) Bad Honnef/Bonn Germany September, 1994.

[25] V. S. Costa, R. Rocha, and L. Damas, “The yap prolog system,” vol. 12, no. 1–2, pp. 5–34, 2012.

[26] A. Dries *et al.*, “Problog2: Probabilistic logic programming,” presented at the Joint European Conference on Machine Learning and Knowledge Discovery in Databases, 2015, pp. 312–315.

[27] L. De Raedt, A. Kimmig, and H. Toivonen, “ProbLog: A Probabilistic Prolog and Its Application in Link Discovery.,” presented at the IJCAI, 2007, vol. 7, pp. 2462–2467.

[28] D. S. Shterionov, A. Kimmig, T. Mantadelis, and G. Janssens, “DNF sampling for problog inference,” 2010.

[29] A. Skarlatidis, A. Artikis, J. Filippou, and G. Paliouras, “A probabilistic logic programming event calculus,” vol. 15, no. 2, pp. 213–245, 2015.

[30] L. G. Valiant, “The complexity of enumeration and reliability problems,” vol. 8, no. 3, pp. 410–421, 1979.

[31] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” vol. 521, no. 7553, p. 436, 2015.

[32] N. Fuhr, “Probabilistic Datalog: Implementing logical information retrieval for advanced applications,” vol. 51, no. 2, pp. 95–110, 2000.

[33] “PIROPO database.” [Online]. Disponible: https://sites.google.com/site/piropodatabase/. [Acceso: 10-Jun-2019].

[34] R. Fisher, “CAVIAR Test Case Scenarios,” 2007. [Online]. Disponible: https://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1/. [Acceso: 10-Jun-2019].

[35] M. Shanahan, *The event calculus explained*. Springer, 1999, pp. 409–430.

[36] L. De Raedt, “ProbLog: Probabilistic Programming.” [Online]. Disponible: https://dtai.cs.kuleuven.be/problog/. [Acceso: 10-Jun-2019].

ANEXOS

ANEXO A: Código modelo lógico probabilístico (segundo experimento)

```
/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        DATOS ENTRADA CAVIAR_DATASET
                -Llamada a los archivos de la base de datos de CAVIAR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*/

                % set datos análisis
                :- use_module('script_dataset_result.pl').


                % Añadir probabilidades a las actividades STA
                0.2::happens(abrupt(P), T);
                0.25::happens(walking(P), T);
                0.1::happens(running(P), T);
                0.25::happens(inactive(P), T)
                  :- happens(active(P), T).
                0.2::happens(running(P), T);
                0.15::happens(inactive(P),T);
                0.2::happens(abrupt(P), T)
                  :- happens(walking(P), T).
                0.05::happens(inactive(P),T);
                0.2::happens(abrupt(P), T)
                  :- happens(running(P), T).
                0.05::happens(inactive(P),T)
                  :- happens(abrupt(P), T).


/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        REGLAS EVENT CALCULUS ∫
                -Reglas necesarias para ejecutar programa de reconocimiento en el tiempo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*/
% holdsAt: comprobar si está ocurriendo una actividad
                holdsAt(F = V, T) :-
                        initially(F = V),
                        negate(broken(F = V, 0, T)).
                holdsAt(F = V, T) :-
                        F \= orientation(X),
                        F \= person(X2),
                        iniciatedAt(F = V, Tinicio),
                        Tinicio is T - 40,
                        negate(broken(F = V, Tinicio, T)).
```

```
% broken: comprobar si se ha terminado una actividad.
		broken(F = V, Tinicio, T) :-
			terminatedAt(F = V, Tfinal),
			Tinicio < Tfinal,
			Tfinal < T.
		broken(F = V1, Tbroke, T) :-
			iniciatedAt(F = V2, Tinicio),
			V1 \= V2,
			Tbroke < Tinicio,
			Tinicio < T.
		%.....

% negate: predicado que engloba la negación de un hecho o regla.
		negate(F) :- \+ F.

/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
		TRANSFORMACIONES EN CODIGO
		-Necesarias para correcto funcionamiento del modelo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*/
% Transformación apariencia para reconocer CAVIAR_DATASET
		happens(appear(Id), T) :-
			holdsAt(appearance(Id)=appear,T).
		happens(disappear(Id), T) :-
			holdsAt(appearance(Id)=disappear,T).
% Transformación para crear abrupt
		happens(abrupt(P1), T) :-
			happens(walking(P1), T),
			happens(fighting(Gr,[P1,P2]), T).
		happens(abrupt(P2), T) :-
			happens(active(P2), T),
			happens(active(P1), T),
			happens(fighting(Gr,[P1,P2]), T).
		happens(abrupt(P1), T) :-
			happens(active(P2), T),
			happens(active(P1), T),
			happens(fighting(Gr,[P1,P2]), T).
```

```
% Si no se cumple que las fluents se hayan iniciado o terminado, sus valores serán false en esos casos
		iniciatedAt(F = false, T) :-
			negate(iniciatedAt(F = true, T)).
		terminatedAt(F = false, T) :-
			negate(terminatedAt(F = true, T)).
/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
		REGLAS PARA RECONOCIMIENTO ACTIVIDADES
		-iniciatedAt y terminatedAt de las LTA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*/
%%%%
% LEAVING_OBJECT: una persona deja un objeto.
%%%%
		0.88::iniciatedAt(leaving_object(P, Obj) = true,  T) :-
			happens(appear(Obj), T),
			happens(inactive(Obj), T),
			holdsAt(close(P, Obj, 30) = true, T),
			holdsAt(person(P) = true, T).
		0.88::iniciatedAt(leaving_object(P, Obj) = true,  T) :-
			iniciatedAt(leaving_object(P, Obj) = true,  Tanterior),
			happens(inactive(Obj), T),
			holdsAt(close(P, Obj, 30) = true, T),
			holdsAt(person(P) = true, T),
			Tanterior < T.
		%.....................................
		0.88::terminatedAt(leaving_object(P, Obj) = true, T) :-
			happens(disappear(Obj),T),
			happens(inactive(Obj),T).
%%%%
% MEETING: dos personas se saludan o interactúan entre si.
%%%%
		0.80::iniciatedAt(meeting(P1, P2) = true, T) :-
			happens(inactive(P1), T),
			holdsAt(close(P1, P2, 25) = true, T),
			%holdsAt(person(P1) = true, T),
			holdsAt(person(P2) = true, T),
			negate(happens(running(P2), T)),
			negate(happens(abrupt(P2),T)),
			negate(happens(active(P2),T)).
```

```
0.90::iniciatedAt(meeting(P1, P2) = true, T) :-
        happens(active(P1), T),
        holdsAt(close(P1, P2, 25) = true, T),
        holdsAt(person(P2) = true, T),
        negate(happens(running(P2), T)),
        negate(happens(abrupt(P2), T)).
%.....................................
0.95::terminatedAt(meeting(P1, P2) = true, T) :-
        happens(running(P1), T),
        holdsAt(person(P2) = true, T).
0.75::terminatedAt(meeting(P1, P2) = true, T) :-
        happens(abrupt(P1), T),
        holdsAt(person(P2) = true, T).
0.75::terminatedAt(meeting(P1, P2) = true, T) :-
        happens(disappear(P1), T),
        holdsAt(person(P2) = true, T).
0.40::terminatedAt(meeting(P1, P2) = true, T) :-
        holdsAt(person(P1) = true, T),
        holdsAt(person(P2) = true, T),
        negate(happens(inactive(P1), T)),
        negate(happens(active(P1), T)),
        negate(happens(inactive(P2), T)),
        negate(happens(active(P2), T)),
        P1 \== P2.
0.80::terminatedAt(meeting(P1, P2) = true, T) :-
        holdsAt(person(P1) = true, T),
        holdsAt(person(P2) = true, T),
        negate(holdsAt(close(P1, P2, 25) = true, T)),
        P1 \== P2.
%.....................................................................................
%%%%
% MOVING: dos personas se están moviendo juntas
%%%%
0.95::iniciatedAt(moving(P1, P2) = true, T) :-
        happens(walking(P1), T),
        happens(walking(P2), T),
        holdsAt(close(P1, P2, 34) = true, T),
        holdsAt(closeOr(P1, P2, 45) = true, T).
%.....................................
```

```
0.85::terminatedAt(moving(P1, P2) = true, T) :-
        happens(walking(P1), T),
        holdsAt(person(P2) = true, T),
        negate(holdsAt(close(P1, P2, 34) = true, T)),
        P1 \== P2.

0.80::terminatedAt(moving(P1, P2) = true, T) :-
        holdsAt(person(P1) = true, T),
        holdsAt(person(P2) = true, T),
        negate(holdsAt(close(P1, P2, 34) = true, T)),
        P1 \== P2.
0.80::terminatedAt(moving(P1, P2) = true, T) :-
        holdsAt(person(P1) = true, T),
        holdsAt(person(P2) = true, T),
        negate(happens(walking(P1), T)),
        P1 \== P2.
0.70::terminatedAt(moving(P1, P2) = true, T) :-
        holdsAt(person(P1) = true, T),
        holdsAt(person(P2) = true, T),
        negate(holdsAt(closeOr(P1, P2, 45) = true, T)),
        P1 \== P2.
%.....
%%%%%
% FIGHTING: dos personas se están peleando
%%%%%
0.88::iniciatedAt(fighting(P1, P2) = true, T) :-
        happens(abrupt(P1), T),
        holdsAt(close(P1, P2, 44) = true, T),
        holdsAt(person(P2) = true, T),
        negate(happens(inactive(P2), T)).
%.....
0.88::terminatedAt(fighting(P1, P2) = true, T) :-
        happens(walking(P1), T),
        holdsAt(person(P2) = true, T),
        P1 \== P2.
0.88::terminatedAt(fighting(P1, P2) = true, T) :-
        happens(running(P1), T),
        holdsAt(person(P2) = true, T),
        P1 \== P2.
```

```
        0.88::terminatedAt(fighting(P1, P2) = true, T) :-
                happens(disappear(P1), T),
                holdsAt(person(P2) = true, T),
                P1 \== P2.
        0.88::terminatedAt(fighting(P1, P2) = true, T) :-
                happens(inactive(P1), T),
                holdsAt(person(P2) = true, T),
                P1 \== P2.
        0.88::terminatedAt(fighting(P1, P2) = true, T) :-
                holdsAt(person(P1) = true, T),
                holdsAt(person(P2) = true, T),
                negate(holdsAt(close(P1, P2, 44) = true, T)),
                P1 \== P2.
        0.88::terminatedAt(fighting(P1, P2) = true, T) :-
                holdsAt(person(P1) = true, T),
                holdsAt(person(P2) = true, T),
                negate(happens(abrupt(P1), T)),
                negate(happens(abrupt(P2), T)),
                P1 \== P2.
/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        REGLAS AUXILIARES          -Necesarias para reconocer LTA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*/
% close: Distancia entre dos ids es menor que la establecida por parámetro.
        holdsAt(close(ID1, ID2, Limite) = true, T) :-
                holdsAt(distance(ID1, ID2, Distancia) = true, T),
                Distancia < Limite.
% closeOr: Angulo entre dos ids es menor que el establecida por parámetro.
        holdsAt(closeOr(P1, P2, LimAngle) = true, T) :-
                holdsAt(orientation(P1) = Orient1, T),
                holdsAt(orientation(P2) = Orient2, T),
                abs(Orient1 - Orient2) < LimAngle,
                P1 \== P2.
% distance: distancia euclidea entre las dos personas a partir de sus coordenadas.
        holdsAt(distance(ID1, ID2, Dist) = true, T) :-
                holdsAt(coord(ID1) = (X1, Y1), T),
                holdsAt(coord(ID2) = (X2, Y2), T),
                Dist is sqrt(((X2-X1)^2)+((Y2-Y1)^2)),
                ID1 \== ID2.
```

```
		%.....................................................................
% person: indicada si el id pertenece a una persona (inactive puede ser de un objeto)
% Su regla holdsAt es diferente a los fluentes
		iniciatedAt(person(P) = true, T) :-
			happens(active(P), T).
		iniciatedAt(person(P) = true, T) :-
			happens(walking(P), T).
		iniciatedAt(person(P) = true, T) :-
			happens(running(P), T).
		iniciatedAt(person(P) = true, T) :-
			happens(abrupt(P), T).
		terminatedAt(person(P) = true, T) :-
			happens(disappear(P), T).
		holdsAt(person(P) = true, T) :-
			iniciatedAt(person(P) = true, T),
			negate(terminatedAt(person(P) = true, T)).
		%.....................................................................
% Multiples iniciaciones para que la probabilidad de ejecucion aumente
		iniciatedAt(moving_multi(P1, P2) = true, T) :-
			holdsAt(moving(P1, P2) = true, T).
		iniciatedAt(fighting_multi(P1, P2) = true, T) :-
			holdsAt(fighting(P1, P2) = true, T).
/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
	CONSULTAS
		-Queries para comprobar si se cumplen las actividades, por quién y cuando
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*/
		query(holdsAt(meeting(X1, X2) = true, T)) :-
			between(0,1054,C),
			T is C*40.
		query(holdsAt(moving(X1, X2) = true, T)) :-
			between(0,826,C),
			T is C*40.
		query(holdsAt(fighting(X1, X2) = true, T)) :-
			between(0,958,C),
			T is C*40.
		query(holdsAt(leaving_object(X1, X2) = true, T)) :-
			between(0,1151,C),
			T is C*40.
```

ANEXO B: Código modelo lógico (primer experimento)

```
/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        DATOS ENTRADA CAVIAR_DATASET
                -Llamada a los archivos de la base de datos de CAVIAR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*/
                % set datos análisis
                :- use_module('script_dataset_result.pl').


/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        REGLAS EVENT CALCULUS ∫
                -Reglas necesarias para ejecutar programa de reconocimiento en el tiempo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*/
% holdsAt: comprobar si está ocurriendo una actividad
                holdsAt(F = V, T) :-
                        initially(F = V),
                        negate(broken(F = V, 0, T)).
                holdsAt(F = V, T) :-
                        F \= orientation(X),
                        F \= person(X2),
                        iniciatedAt(F = V, Tinicio),
                        Tinicio is T - 40,
                        negate(broken(F = V, Tinicio, T)).
% broken: comprobar si se ha terminado una actividad.
                broken(F = V, Tinicio, T) :-
                        terminatedAt(F = V, Tfinal),
                        Tinicio < Tfinal,
                        Tfinal < T.
                broken(F = V1, Tbroke, T) :-
                        iniciatedAt(F = V2, Tinicio),
                        V1 \= V2,
                        Tbroke < Tinicio,
                        Tinicio < T.
                %..........................................
% negate: predicado que engloba la negación de un hecho o regla.
                negate(F) :- \+ F.
```

```
/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
			TRANSFORMACIONES EN CODIGO
			-Necesarias para correcto funcionamiento del modelo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*/

% Transformación apariencia para reconocer CAVIAR_DATASET
			happens(appear(Id), T) :-
				holdsAt(appearance(Id)=appear,T).
			happens(disappear(Id), T) :-
				holdsAt(appearance(Id)=disappear,T).

% Transformación para crear abrupt
			happens(abrupt(P1), T) :-
				happens(walking(P1), T),
				happens(fighting(Gr,[P1,P2]), T).
			happens(abrupt(P2), T) :-
				happens(active(P2), T),
				happens(active(P1), T),
				happens(fighting(Gr,[P1,P2]), T).
			happens(abrupt(P1), T) :-
				happens(active(P2), T),
				happens(active(P1), T),
				happens(fighting(Gr,[P1,P2]), T).

% Si no se cumple que las fluents se hayan iniciado o terminado, sus valores serán false en esos casos
			iniciatedAt(F = false, T) :-
				negate(iniciatedAt(F = true, T)).
			terminatedAt(F = false, T) :-
				negate(terminatedAt(F = true, T)).
/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
			REGLAS PARA RECONOCIMIENTO ACTIVIDADES
			-iniciatedAt y terminatedAt de las LTA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*/
```

```
%%%%
% LEAVING_OBJECT: una persona deja un objeto.
%%%%
                iniciatedAt(leaving_object(P, Obj) = true,  T) :-
                        happens(appear(Obj), T),
                        happens(inactive(Obj), T),
                        holdsAt(close(P, Obj, 30) = true, T),
                        holdsAt(person(P) = true, T).
                iniciatedAt(leaving_object(P, Obj) = true,  T) :-
                        iniciatedAt(leaving_object(P, Obj) = true,  Tanterior),
                        happens(inactive(Obj), T),
                        holdsAt(close(P, Obj, 30) = true, T),
                        holdsAt(person(P) = true, T),
                        Tanterior < T.
                %.....................................
                terminatedAt(leaving_object(P, Obj) = true, T) :-
                        happens(disappear(Obj),T),
                        happens(inactive(Obj),T).
%%%%
% MEETING: dos personas se saludan o interactúan entre si.
%%%%
                iniciatedAt(meeting(P1, P2) = true, T) :-
                        happens(inactive(P1), T),
                        holdsAt(close(P1, P2, 25) = true, T),
                        %holdsAt(person(P1) = true, T),
                        holdsAt(person(P2) = true, T),
                        negate(happens(running(P2), T)),
                        negate(happens(abrupt(P2),T)),
                        negate(happens(active(P2),T)).
                iniciatedAt(meeting(P1, P2) = true, T) :-
                        happens(active(P1), T),
                        holdsAt(close(P1, P2, 25) = true, T),
                        holdsAt(person(P2) = true, T),
                        negate(happens(running(P2), T)),
                        negate(happens(abrupt(P2), T)).
                %.....................................
                terminatedAt(meeting(P1, P2) = true, T) :-
                        happens(running(P1), T),
                        holdsAt(person(P2) = true, T).
```

```
terminatedAt(meeting(P1, P2) = true, T) :-
        happens(abrupt(P1), T),
        holdsAt(person(P2) = true, T).
terminatedAt(meeting(P1, P2) = true, T) :-
        happens(disappear(P1), T),
        holdsAt(person(P2) = true, T).
terminatedAt(meeting(P1, P2) = true, T) :-
        holdsAt(person(P1) = true, T),
        holdsAt(person(P2) = true, T),
        negate(happens(inactive(P1), T)),
        negate(happens(active(P1), T)),
        negate(happens(inactive(P2), T)),
        negate(happens(active(P2), T)),
        P1 \== P2.
terminatedAt(meeting(P1, P2) = true, T) :-
        holdsAt(person(P1) = true, T),
        holdsAt(person(P2) = true, T),
        negate(holdsAt(close(P1, P2, 25) = true, T)),
        P1 \== P2.
%.....................................................................................
%%%%
% MOVING: dos personas se están moviendo juntas
%%%%
iniciatedAt(moving(P1, P2) = true, T) :-
        happens(walking(P1), T),
        happens(walking(P2), T),
        holdsAt(close(P1, P2, 34) = true, T),
        holdsAt(closeOr(P1, P2, 45) = true, T).
%.....................................

terminatedAt(moving(P1, P2) = true, T) :-
        happens(walking(P1), T),
        holdsAt(person(P2) = true, T),
        negate(holdsAt(close(P1, P2, 34) = true, T)),
        P1 \== P2.
terminatedAt(moving(P1, P2) = true, T) :-
        holdsAt(person(P1) = true, T),
        holdsAt(person(P2) = true, T),
        negate(holdsAt(close(P1, P2, 34) = true, T)),
        P1 \== P2.
```

```
terminatedAt(moving(P1, P2) = true, T) :-
        holdsAt(person(P1) = true, T),
        holdsAt(person(P2) = true, T),
        negate(happens(walking(P1), T)),
        P1 \== P2.
terminatedAt(moving(P1, P2) = true, T) :-
        holdsAt(person(P1) = true, T),
        holdsAt(person(P2) = true, T),
        negate(holdsAt(closeOr(P1, P2, 45) = true, T)),
        P1 \== P2.
%.....................................................................................
%%%%%
% FIGHTING: dos personas se están peleando
%%%%%
iniciatedAt(fighting(P1, P2) = true, T) :-
        happens(abrupt(P1), T),
        holdsAt(close(P1, P2, 44) = true, T),
        holdsAt(person(P2) = true, T),
        negate(happens(inactive(P2), T)).
%.....................................
terminatedAt(fighting(P1, P2) = true, T) :-
        happens(walking(P1), T),
        holdsAt(person(P2) = true, T),
        P1 \== P2.

terminatedAt(fighting(P1, P2) = true, T) :-
        happens(running(P1), T),
        holdsAt(person(P2) = true, T),
        P1 \== P2.

terminatedAt(fighting(P1, P2) = true, T) :-
        happens(disappear(P1), T),
        holdsAt(person(P2) = true, T),
        P1 \== P2.

terminatedAt(fighting(P1, P2) = true, T) :-
        happens(inactive(P1), T),
        holdsAt(person(P2) = true, T),
        P1 \== P2.
```

```
		terminatedAt(fighting(P1, P2) = true, T) :-
			holdsAt(person(P1) = true, T),
			holdsAt(person(P2) = true, T),
			negate(holdsAt(close(P1, P2, 44) = true, T)),
			P1 \== P2.
		terminatedAt(fighting(P1, P2) = true, T) :-
			holdsAt(person(P1) = true, T),
			holdsAt(person(P2) = true, T),
			negate(happens(abrupt(P1), T)),
			negate(happens(abrupt(P2), T)),
			P1 \== P2.
/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
		REGLAS AUXILIARES		-Necesarias para reconocer LTA
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*/
% close: Distancia entre dos ids es menor que la establecida por parámetro.
		holdsAt(close(ID1, ID2, Limite) = true, T) :-
			holdsAt(distance(ID1, ID2, Distancia) = true, T),
			Distancia < Limite.
% closeOr: Angulo entre dos ids es menor que el establecida por parámetro.
		holdsAt(closeOr(P1, P2, LimAngle) = true, T) :-
			holdsAt(orientation(P1) = Orient1, T),
			holdsAt(orientation(P2) = Orient2, T),
			abs(Orient1 - Orient2) < LimAngle,
			P1 \== P2.
% distance: distancia euclidea entre las dos personas a partir de sus coordenadas.
		holdsAt(distance(ID1, ID2, Dist) = true, T) :-
			holdsAt(coord(ID1) = (X1, Y1), T),
			holdsAt(coord(ID2) = (X2, Y2), T),
			Dist is sqrt(((X2-X1)^2)+((Y2-Y1)^2)),
			ID1 \== ID2.
		%.....................................................................
% person: indicada si el id pertenece a una persona (inactive puede ser de un objeto)
% Su regla holdsAt es diferente a los fluentes
		iniciatedAt(person(P) = true, T) :-
			happens(active(P), T).
		iniciatedAt(person(P) = true, T) :-
			happens(walking(P), T).
```

```
		iniciatedAt(person(P) = true, T) :-
			happens(running(P), T).
		iniciatedAt(person(P) = true, T) :-
			happens(abrupt(P), T).
		terminatedAt(person(P) = true, T) :-
			happens(disappear(P), T).
		holdsAt(person(P) = true, T) :-
			iniciatedAt(person(P) = true, T),
			negate(terminatedAt(person(P) = true, T)).
		%.....
% Multiples iniciaciones para que la probabilidad de ejecucion aumente
		iniciatedAt(moving_multi(P1, P2) = true, T) :-
			holdsAt(moving(P1, P2) = true, T).
		iniciatedAt(fighting_multi(P1, P2) = true, T) :-
			holdsAt(fighting(P1, P2) = true, T).
/*
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
	CONSULTAS
		-Queries para comprobar si se cumplen las actividades, por quién y cuando
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
*/
		query(holdsAt(meeting(X1, X2) = true, T)) :-
			between(0,1054,C),
			T is C*40.
		query(holdsAt(moving(X1, X2) = true, T)) :-
			between(0,826,C),
			T is C*40.
		query(holdsAt(fighting(X1, X2) = true, T)) :-
			between(0,958,C),
			T is C*40.
		query(holdsAt(leaving_object(X1, X2) = true, T)) :-
			between(0,1151,C),
			T is C*40.
```

ANEXO C: Código script datos de entrada

```
% SITUACIONES WALKING
% wk1gt:
                :- use_module('CAVIAR_DATASET/set01/wk1gt/AppearenceIndv.pl').
                :- use_module('CAVIAR_DATASET/set01/wk1gt/MovementIndv.pl').
% wk2gt:
                :- use_module('CAVIAR_DATASET/set01/wk2gt/AppearenceIndv.pl').
                :- use_module('CAVIAR_DATASET/set01/wk2gt/MovementIndv.pl').

% SITUACIONES LEAVING_OBJECT
% lb1gt:
                :- use_module('CAVIAR_DATASET/set01/lb1gt/AppearenceIndv.pl').
                :- use_module('CAVIAR_DATASET/set01/lb1gt/MovementIndv.pl').
% lb2gt:
                :- use_module('CAVIAR_DATASET/set01/lb2gt/AppearenceIndv.pl').
                :- use_module('CAVIAR_DATASET/set01/lb2gt/MovementIndv.pl').
% lbpugt:
                :- use_module('CAVIAR_DATASET/set01/lbpugt/AppearenceIndv.pl').
                :- use_module('CAVIAR_DATASET/set01/lbpugt/MovementIndv.pl').

% SITUACIONES MEETING WALKING
% mwt1gt:
                :- use_module('CAVIAR_DATASET/set01/mwt1gt/AppearenceIndv.pl').
                :- use_module('CAVIAR_DATASET/set01/mwt1gt/MovementIndv.pl').
% mwt2gt:
                :- use_module('CAVIAR_DATASET/set01/mwt2gt/AppearenceIndv.pl').
                :- use_module('CAVIAR_DATASET/set01/mwt2gt/MovementIndv.pl').

% SITUACIONES FIGHTING
% fomdgt1:
                :- use_module('CAVIAR_DATASET/set01/fomdgt1/AppearenceIndv.pl').
                :- use_module('CAVIAR_DATASET/set01/fomdgt1/MovementIndv.pl').
                :- use_module('CAVIAR_DATASET/set01/fomdgt1/SituationGrp.pl').
% fomdgt3:
                :- use_module('CAVIAR_DATASET/set01/fomdgt3/AppearenceIndv.pl').
                :- use_module('CAVIAR_DATASET/set01/fomdgt3/MovementIndv.pl').
                :- use_module('CAVIAR_DATASET/set01/fomdgt3/SituationGrp.pl').
```

% fcgt:

```
:- use_module('CAVIAR_DATASET/set01/fcgt/AppearenceIndv.pl').
:- use_module('CAVIAR_DATASET/set01/fcgt/MovementIndv.pl').
:- use_module('CAVIAR_DATASET/set01/fcgt/SituationGrp.pl').
```

ANEXO D: Ejemplo datos de entrada

```
% Frame number: 17
    holdsAt( orientation( id0 )=0, 680  ).
    holdsAt( appearance( id0 )=appear,  680 ).
    happens( walking( id0 ), 680 ).
    holdsAt( coord( id0 )=( 262, 285 ), 680 ).
% Frame number: 18
    holdsAt( orientation( id0 )=0, 720  ).
    holdsAt( appearance( id0 )=visible,  720 ).
    happens( walking( id0 ), 720 ).
    holdsAt( coord( id0 )=( 262, 284 ), 720 ).
% Frame number: 19
    holdsAt( orientation( id0 )=45, 760  ).
    holdsAt( appearance( id0 )=visible,  760 ).
    happens( walking( id0 ), 760 ).
    holdsAt( coord( id0 )=( 260, 284 ), 760 ).
% Frame number: 20
    holdsAt( orientation( id0 )=45, 800  ).
    holdsAt( appearance( id0 )=visible,  800 ).
    happens( walking( id0 ), 800 ).
    holdsAt( coord( id0 )=( 260, 284 ), 800 ).
```

Aspectos relevantes

- Para cada consulta y ejecución de los scripts de datos se debe elegir un script y una consulta a realizar, según la actividad que se quiera reconocer.
- La jerarquía de carpetas debe seguirse según use_module o cambiarse para ejecutar según se desee.
- Los datos de entrada completos los tenemos en la página web de CAVIAR, citada anteriormente en la bibliografía.
- El Anexo A pertenece a la experimentación del problema con datos probabilísticos usado en el segundo experimento, mientras que el Anexo B pertenece al modelo sin el uso de probabilidad utilizado en el primer experimento.