UNIVERSITAT DE VALÈNCIA
BIBLIOTECA CIÈNCIES

Nº Registre 4849
DATA 26·5·93
SIGNATURA T.D. 211
BIBLIOTECA
Nº LIBIS: 19491323

→ FISICAS.

30 cms.

UNIVERSITAT DE VALENCIA
FACULTAT DE FISICA
Departament d'informàtica i electrònica

FORMALIZACION DEL RAZONAMIENTO CUALITATIVO Y TEMPORAL PARA EL TRATAMIENTO DE PROBLEMAS ASOCIADOS AL CONTROL DE SISTEMAS FISICOS CON REPRESENTACION MULTIDIMENSIONAL

Tesis Doctoral
presentada por:
Salvador Moreno Picot.
1993

UMI Number: U607743

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI U607743
Published by ProQuest LLC 2014. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

DEPARTAMENTO DE INFORMATICA Y ELECTRONICA.
FACULTAD DE CIENCIAS FISICAS.
UNIVERSITAT DE VALENCIA.

FORMALIZACION DEL RAZONAMIENTO CUALITATIVO Y TEMPORAL PARA EL TRATAMIENTO DE PROBLEMAS ASOCIADOS AL CONTROL DE SISTEMAS FISICOS CON REPRESENTATION MULTIDIMENSIONAL

DON GREGORIO MARTIN QUETGLAS, PROFESOR TITULAR DEL AREA DE CONOCIMIENTO DE CIENCIAS DE LA COMPUTACION E INTELIGENCIA ARTIFICIAL DE LA UNIVERSIDAD DE VALENCIA, Y DON FRANCISCO TOLEDO LOBO, PROFESOR TITULAR DE UNIVERSIDAD INTERINO DEL AREA DE CONOCIMIENTO DE CIENCIAS DE LA COMPUTACION E INTELIGENCIA ARTIFICIAL DE LA UNIVERSIDAD JAUME I,

CERTIFICAN:que la presente TESIS DOCTORAL titulada "FORMALIZACION DEL RAZONAMIENTO CUALITATIVO Y TEMPORAL PARA EL TRATAMIENTO DE PROBLEMAS ASOCIADOS AL CONTROL DE SISTEMAS FISICOS CON REPRESENTATION MULTIDIMENSIONAL", ha sido realizada bajo nuestra dirección por D. Salvador Moreno Picot, Licenciado en Físicas.

y para que conste a todos los efectos oportunos, se extiende la presente certificación, en el lugar y la fecha indicados.

VALENCIA, 26 de Marzo de 1993

Fdo:Dr. D.Francisco Toledo Lobo.

Fdo.:Dr. D.Gregorio Martín Quetglás.

AGRADECIMIENTOS:

A Gregorio Martín Quetglás, sin cuya inestimable colaboración el presente trabajo nunca hubiera visto la luz.

A Francisco Toledo Lobo, cuya valiosísima ayuda y consejo ha sido decisiva en la realización del presente trabajo.

A Victoriano Sánchez-Barcaiztegui por su valiosa colaboración y ayuda.

A todos mis amigos y compañeros del L.I.S.I.T.T, sin cuya ayuda no me hubiera sido posible realizar la implementación computacional del formalismo presentado en este trabajo.

Al grupo de Investigación de la Universidad Politécnica de Madrid dirigido por Dr. D. José Cuena, por la valiosa ayuda prestada durante mi formación en las materias que forman la base del presente trabajo.

Al L.I.S.I.T.T, por los medios que ha puesto a mi alcance para la realización de este trabajo.

Al Servicio de Tráfico y Transportes del Ayuntamiento de Valencia, por las facilidades prestadas para la realización del presente trabajo.

LA REALIZACION DEL PRESENTE TRABAJO HA SIDO POSIBLE GRACIAS AL APOYO FINANCIERO DE UNA BECA DE LA CONSELLERIA DE CULTURA, EDUCACIO I CIENCIA DE LA **GENERALITAT VALENCIANA** Y DEL CONVENIO DE INVESTIGACION ESTABLECIDO CON LA EMPRESA **ETRA S.A.**, PARA EL DESARROLLO DEL PROYECTO ESPRIT II "EQUATOR".

DEDICATORIA:

A mis padres.

A Maria Teresa.

A Nicolás.

RESUMEN

Esta memoria presenta un nuevo formalismo de razonamiento cualitativo y temporal para el tratamiento de problemas asociados al control de sistemas físicos con representación multidimensional. Este formalismo generaliza los aspectos más importantes de los paradigmas de razonamiento cualitativo de Kuipers, De Kleer y Forbus, como son el tratamiento cualitativo de funciones, la representación orientada a componentes y una representación jerárquica del conocimiento asociado a los procesos de cambio, e incluye el soporte adecuado para la representación y razonamiento con parámetros multidimensionales. La investigación que ha conducido al desarrollo de este formalismo ha sido principalmente realizada para el proyecto ESPRIT II EQUATOR. El formalismo presentado es muy adecuado para su utilización en sistemas de Control de Tráfico Urbano y otros dominios donde se requiera la representación de parámetros multidimensionales, entre los que analizamos el problema del sistema de cubas, extraído de la literatura.

En primer lugar se analizan las deficiencias presentadas por los paradigmas de razonamiento cualitativo más representativos del campo en el tratamiento de sistemas con representación multidimensional; seguidamente se propone una arquitectura de control basada en un formalismo de razonamiento cualitativo con representación multidimensional y un segundo formalismo de representación cualitativa y temporal de patrones de situaciones problemáticas asociadas al control de sistemas físicos con diferentes niveles de agregación. A continuación se desarrolla de forma teórica el formalismo de razonamiento cualitativo propuesto, y se aplica a la representación del Tráfico Urbano, previo análisis de la naturaleza multidimensional del mismo. Seguidamente se aplica a un problema extraído de la literatura, el problema del Sistema de Cubas, donde demostramos su adecuación. Presentamos a continuación los principios del formalismo de representación de patrnos de situaciones problemáticas propuesto en la arquitectura presentada. Por último, se presenta el desarrolla del motor de inferencia en Prolog y en un lenguaje de representación del conocimiento que demuestra las ventajas de escribir el formalismo sobre un lenguaje que incorpore en forma homogénea elementos de programación declarativa, programación orientada a objetos y programación lógica por restricciones.

Se adjunta a la memoria una serie de anexos, entre los que se pueden encontrar el listado completo del motor de inferencia en Prolog del formalismo, y los programas en Prolog de los dos casos analizados.

ABSTRACT

This thesis introduces a new qualitative and temporal reasoning formalism for the treatment of problems associated to the control of physical systems with multidimensional representation. This formalism generalizes the most important aspects of the qualitative reasoning paradigms of Kuipers, De Kleer and Forbus, as the qualitative treatment of functions, the component-oriented representation and a hiearchical representation of the knowledge associated to change processes, and includes a proper support for representing and reasoning with multidimensional parameters. The research from which this formalism has been developed, belongs to the work done in the EQUATOR ESPRIT-II project. The presented formalism is very suitable to be used in Urban Traffic Control Systems and other domains demanding multidimensional parameter representation. As an example of a different domain of this kind, we choose the Many Tanks Problem.

First of all, the deficiencies presented by the most representative qualitative reasoning paradigms in the field of multidimensional representation are analized. After this, we propose a control architecture based on a qualitative reasoning formalism with multidimensional representacion and a second formalism for the qualitative and temporal representation of patterns of problematical situations with different levels of aggregation. Later, the theory of the proposed qualitative and temporal reasoning formalism is developed, and then it is applied to the representation of Urban Traffic, due to the multidimensional nature of traffic. The formalism is after applied to the Many Tanks Problem, showing again its suitability in this new application domain. Afterwards, the principles of the pattern representation formalism of problematical situations is presented. By last, the inference engine of the proposed formalism is given in two different languages: Prolog and a representation language, able to represent in an homogeneous way elements of declarative programing, object-oriented programming and constraint-logic programming.

The thesis also includes several annexes, containing the whole listing of the inference engine of the formalism in Prolog, toguether with the programs in Prolog of the two application domains the formalism has been applied to.

Indice

Organización de la Tesis Doctoral

El capítulo I propone una arquitectura funcional de un sistema de control basado en modelización cualitativa, que esta actualmente siendo desarrollada en el proyecto ESPRIT II EQUATOR, con objeto de mostrar las ventajas que supone para un sistema de control la incorporación de un modelo del mundo real y de una definición de alto nivel de lo que se considera incidente. Este capítulo enmarca, por tanto, el formalismo de razonamiento cualitativo con representación multidimensional que se presenta en el capítulo II y el formalismo de abstracción de acontecimientos que se presenta en el capítulo IV como partes integrantes de una innovadora arquitectura de sistemas de control, enmarcable dentro de los sistemas expertos de segunda generación, que constituye un prometedor campo de trabajo para futuras investigaciones.

El capítulo II presenta un formalismo de razonamiento cualitativo para sistemas físicos con representación multidimensional, que surge como respuesta a las carencias de los paradigmas existentes de razonamiento cualitativo en la representación de sistemas con parámetros multidimensionales. El formalismo (λ,t) puede verse como una generalización de los paradigmas de Kuipers, De Kleer y Forbus que incorpora parámetros multidimensionales. Sus rasgos más característicos son la representación de funciones del tipo F(t) tomada del paradigma de Kuipers, la descomposición jerárquica del sistema en componentes tomada del paradigma de De Kleer, la jerarquía de ecuaciones físicas y confluencias, que podemos considerar como relacionada a los procesos cualitativos de Forbus y a las confluencias de De Kleer, y por último, la representación de parámetros multidimensionales, que ha tratado de desarrollarse como una extensión de la representación de funciones del tipo F(t) según Kuipers. El resultado final de la aplicación del formalismo a un sistema físico consiste en una base de conocimiento que lo describe, compuesta de las siguientes cinco secciones: jerarquía de objetos, jerarquía de relaciones, jerarquía

de parámetros, comportamiento de objetos y comportamiento de relaciones. Esta base de conocimiento tiene doble particularidad de unir un gran poder de representación con una separación del conocimiento del dominio de aplicación del conocimiento sobre el sistema concreto, lo que hace particularmente sencillo modelizar varios sistemas pertenecientes al mismo dominio de aplicación, o modificar el conocimiento sobre un sistema dado. Al final se propone un algoritmo para el motor de inferencia del formalismo (λ,t). La versión en Prolog de este algoritmo se analizará en el capítulo VI.

En el capítulo III tratamos de demostrar la utilidad del formalismo definido en el capítulo anterior para la modelización de sistemas físicos con parámetros multidimensionales. Para ello tomamos dos dominios de aplicación distintos: el Control de Tráfico Urbano y un problema tomado de la literatura, el Problema del Sistema de Cubas. Ambos dominios tienen en común el manejar procesos de cambio continuo, como puede ser la evolución de una cola en una calle o el proceso de llenado de una cuba, cuya representación puede hacerse de forma más natural mediante la utilización de parámetros multidimensionales. La no utilización de los mismos conduciría a una aproximación burda del elemento como un único parámetro del tipo F(t), o a una innecesaria discretización del mismo en un conjunto de funciones Fi(t) hasta lograr el nivel de detalle requerido.

El dominio del Control de Tráfico Urbano, considerado en primer lugar, merece una mención especial debido a ser el origen del formalismo considerado. En efecto, el formalismo (λ,t) nació como un intento de generalizar un formalismo anterior que desarrollamos en este dominio con objeto de poder manejar de forma adecuada las características multidimensionales del tráfico, durante el desarrollo de un prototipo del VANESA (Valencia Area Network Expert System Advisor). En la sección III.2.1 demostramos la naturaleza esencialmente multidimensional del tráfico urbano, y en la sección III.2.2 analizamos un escenario simple de tráfico mediante el formalismo (λ,t), obteniendo su correspondiente Base de Conocimiento.

El dominio del Sistema de Cubas, tomado del artículo original de Murray Shanahan sobre Cálculo de Acontecimientos para Cambio Continuo, ha sido elegido como ejemplo documentado de un sistema diferente que implica procesos de cambio continuo. De forma análoga, en la sección III.3.1 se propone su representación utilizando parámetros multidimensionales, por el mayor poder descriptivo del sistema de los mismos, y la sección III.3.2 analiza el sistema con el formalismo (λ,t), llegando de forma análoga a una Base de Conocimiento que lo representa.

La conclusión fundamental de este capítulo es mostrar la adecuación del formalismo propuesto para la representación de sistemas con parámetros multidimensionales, con respecto a la utilización alternativa de paradigmas clásicos de modelización cualitativa.

El capítulo IV presenta el módulo de Abstracción de Acontecimientos, indispensable en la arquitectura propuesta en el capítulo I. Su objetivo es el de poder reconocer situaciones problemáticas a partir de los acontecimientos de la Base de Datos Temporal generada por el formalismo (λ,t), por medio de razonamiento cualitativo y temporal con diferentes niveles de agregación. En el capítulo se propone un formalismo de representación de patrones de situaciones problemáticas que un sistema de control debe tratar de evitar o de paliar. Este formalismo define clases de situaciones problemáticas, que pueden agruparse en tres niveles, según su nivel de agregación temporal: situaciones problemáticas elementales, situaciones problemáticas temporales y situaciones problemáticas abstractas. Las primeras definen patrones de situaciones problemáticas instantáneas, que se dan en un único objeto por concurrencia de determinadas condiciones sobre determinados parámetros (por ejemplo, en una esclusa de aire, tendremos un problema si ambas compuertas están simultáneamente abiertas). Las segundas definen patrones de situaciones problemáticas temporales, que se dan en un único objeto mediante una sucesión de condiciones sobre los parámetros que lo definen (por ejemplo, en la esclusa de aire, podemos definir un problema temporal cuya gravedad es proporcional al tiempo que ambas compuertas estén simultáneamente abiertas). Las terceras definen patrones de situaciones problemáticas de gran extensión temporal, que se dan en un grupo de objetos vecinos (por ejemplo, la detección de un "dead-lock", o congestión circular, en un anillo de una red urbana).

El formalismo de representación de situaciones problemáticas aquí presentado combina su gran poder de representación cualitativo y temporal con una gran adaptabilidad a muy diversos tipos de dominios, y además la representación obtenida puede ser convertida fácilmente en una representación clausal directamente ejecutable en Prolog, haciendo uso del Cálculo de Acontecimientos para Cambio Continuo de Shanahan.

El capítulo V tiene por objetivo dar una muestra de la simplicidad y claridad de representación que se alcanza expresando el dominio de aplicación de Tráfico Urbano en el formalismo (λ,t) si para ello se utiliza un lenguaje de representación del

conocimiento de alto nivel con características especiales. Las características básicas del lenguaje escogido pueden verse en el Anexo C, y pueden resumirse en la capacidad que el lenguaje ofrece para la representación declarativa del conocimiento incorporando programación lógica por restricciones y programación orientada a objetos.

El capítulo VI presenta una implementación en Prolog del motor de inferencia del formalismo (λ,t). Esta implementación está realizada utilizando técnicas de programación orientada a objetos en Prolog, y su relación con el algoritmo teórico para el motor de inferencia propuesto el capítulo II está claramente detallada a través de un análisis comparativo de los predicados más importantes del programa.

El Anexo A muestra una implementación ejecutable del formalismo (λ,t) sobre un escenario de Tráfico Urbano. Esta implementación se compone de la Base de Conocimiento del escenario, cuya obtención fue explicada en el capítulo III, y de un motor de inferencia que la interpreta. El motor de inferencia mostrado en el ejemplo, corresponde a una implementación del motor de inferencia más primitiva que la mostrada en el capítulo VI, y puede verse cómo, aunque tienen idéntica función, el código mostrado en el capítulo VI es mucho más compacto y fácil de leer, debido a la utilización en el mismo de técnicas de programación orientada a objetos.

El Anexo B muestra una implementación ejecutable del formalismo (λ,t) sobre un sistema de cubas. Esta implementación se compone, de forma análoga al caso anterior, de la Base de Conocimiento del sistema, ya explicada en el capítulo III, y de un motor de inferencia que la interpreta. El motor de inferencia mostrado es el mismo que se explica en el capítulo VI.

El Anexo C muestra las características fundamentales de un lenguaje de representación del conocimiento de alto nivel. Este lenguaje se ha escogido en el capítulo V para mostrar la claridad de representación que puede llegar a alcanzar el formalismo (λ,t) sobre un lenguaje apropiado.

El Anexo D muestra el listado en Prolog del motor de inferencia del formalismo (λ,t) que se comenta en el capítulo VI.

El Anexo E muestra los predicados del Cálculo de Acontecimientos de Kowalski y Sergot. La sección E.1 muestra los 8 predicados originales, mientras que la sección

E.2 muestra la extensión del Event Calculus para poder representar acontecimientos y propiedades con diferente nivel de agregación temporal.

El Anexo F muestra una implementación en C del motor de inferencia del formalismo (λ,t) para la Aplicación de Tráfico Urbano. Esta implementación es la mas rápida de que disponemos, pero adolece de grandes defectos comparada con la implementación en Prolog expuesta en el anexo D. El primero es su dependencia del dominio: mientras la implementación en Prolog puede aplicarse a diferentes dominios de aplicación, la implementación en C únicamente puede representar escenarios de Tráfico Urbano. El segundo es su complejidad, que se pone de manifiesto comparando la longitud de ambos listados, y tratando de entender su contenido. El tercero es la dificultad de modificación de su contenido: mientras el programa en Prolog permitiría añadir fácilmente nuevos tipos de parámetros ahora no considerados, o realizar cualquier otro cambio de esta índole, el programa en C requeriría un esfuerzo enorme de modificación y depuración de errores antes de producir un resultado comparable.

El anexo G presenta un estado del arte sobre el estado actual de los sistemas de control de Tráfico Urbano.

El anexo H muestra un resumen del Cálculo de Acontecimientos para Cambio Continuo de Murray Shanahan.

Introducción

i.1 CONTROL DE PROCESOS

i.1.1 Definición

Los métodos de diseño de Sistemas de Control tradicionales, se basan en la construcción de un modelo determinista del proceso a controlar [Smith,85]. Esto implica hacer unas suposiciones básicas de partida acerca del comportamiento del sistema dentro de un cierto rango de operación que se ve prefijado de antemano. Generalmente se presupone la linearidad y/o la invarianza temporal, y utilizando el cálculo diferencial se obtiene una representación del sistema en forma de ecuaciones diferenciales o en diferencias finitas. En muchas ocasiones se necesita un modelo perfectamente formulado para proceder a partir de él; sin embargo, en gran cantidad de procesos, es dificultoso sino imposible dar una representación adecuada sobre el rango completo de condiciones de operación, y en la práctica aún cuando un proceso pueda formularse matemáticamente, el tiempo y esfuerzo requerido para extraer el modelo asociado, lo hace inviable. Por otra parte, dada la complejidad de muchos de estos modelos matemáticos, la extracción de consecuencias a partir de los mismos es tan costosa, que impide la toma de decisiones en condiciones operativas.

El concepto de Control de Procesos (CP) que se asume en este trabajo, es el de aquellas aplicaciones que reúnen las siguientes características [Toledo,90]:

a) Trabajan en paralelo con el mundo real, con el cual interaccionan por medio de los sensores y efectores correspondientes; como consecuencia, el sistema que

UNIVERSITAT DE VALÈNCIA BIBLIOTECA DE CIÈNCIES

manejamos entra dentro del conjunto de los llamados DAC (Digital Adquisition and Control System) tal como vienen definidos en textos clásicos [Smith,85][1].

b) Incluyen un modelo del sistema, a partir del cual toman decisiones. En base a la utilización de este modelo, que es de carácter algorítmico, el sistema está en condiciones de tomar decisiones sobre el control de su evolución.

c) En el vértice de la jerarquía de la toma de decisiones, existe un operador humano, al cual se le reconoce expertez sobre el dominio y conocimiento acerca de las limitaciones del modelo sobre el cual se ejerce el control de forma automática. Esta característica distingue nuestra acepción de Control de Procesos de la más tradicional de Control automático de Procesos.

d) Por su naturaleza, se considera que el sistema algorítmico no está en condiciones óptimas de tomar decisiones bajo ciertas condiciones, cosa que sí sabe llevar a cabo el experto humano.

En resumen, los sistemas que manejaremos serán más genéricos que aquellos que son susceptibles de llegar a ser controlados por medios totalmente automáticos. El ejemplo del Control de Tráfico Urbano, es un caso representativo de tal situación, pues existen una serie de variables (p.e. las demandas) que no pueden ser objeto de un control automático.

i.1.2 De los Sistemas Tradicionales a los Sistemas Expertos en el Control de Procesos

Son muchos los trabajos que se han centrado en el desarrollo de métodos de control adaptativos [Bristol,77], [Hetthessy,83], [Radke,84], [Keviczky,85], [Lim,85] (en el sentido de que son capaces de fijar on-line, los parámetros conocidos de un modelo cuya estructura se ha asumido). Esta aproximación extiende el campo de aplicabilidad de los modelos analíticos, pero sin embargo, la bondad de tales sistemas sigue siendo dependiente de las simplificaciones hechas en la estructura del modelo. Los métodos adaptativos requieren conocimiento de teoría de control de

[1]Smith et al. resumen los principios de los Sistemas de Control por las tres letras M, D, A. M referida a las medidas de las variables que afectan al proceso; D referida a las decisiones que se toman sobre la base de las medidas; A referida a las acciones a tomar.

procesos (lo que podríamos llamar meta-conocimiento) más que conocimiento acerca del proceso en sí[2].

Otro gran inconveniente que presentan los sistemas de control tradicionales es que el conocimiento humano del proceso y su descripción matemática convencional son de naturaleza totalmente distinta, y de este hecho se deriva que el Interface hombre/máquina sea poco efectivo, e incluso la carencia de tal interface (sobre este tema volveremos en i.1.3); por tanto no puede proporcionarse una buena explicación de las decisiones de control, y el operador humano no es capaz de utilizar su conocimiento experimental (su expertez) para modificar el programa de control; la necesidad de que el operador humano actúe sobre el sistema lleva frecuentemente a la construcción de un sistema de monitorización del proceso que resulta costoso y cuya información en muchas aplicaciones no es utilizable por el sistema de control.

De todo lo comentado anteriormente, podemos destacar que la automatización de procesos de control acarrea problemas de vigilancia y control en sistemas complejos, a los que es difícil dar una buena solución que sirva para múltiples situaciones; generalmente, los sistemas de control tradicionales no dan una solución satisfactoria en situaciones críticas y por tanto, aún cuando se disponga de un sistema de control de procesos, a su frente, desde los más simples hasta los muy complejos, debe haber un experto humano que contraste la actuación del sistema y que actúe en el momento en que las soluciones del sistema de control dejan de ser las óptimas. Por añadidura, la exigencia de que estos sistemas funcionen en paralelo con el mundo real, plantea múltiples problemas [Sibille,87] que analizaremos posteriormente.

Los Sistemas Expertos [Cuena,87], ofrecen al Control de Procesos la oportunidad de "incorporar en su estructura una síntesis tanto del conocimiento que define las respuestas del modelo, como la forma de utilizarlas para la definición de decisiones por las personas expertas, de manera que es posible tener 'en línea' no sólo imágenes de datos más o menos agregados como en las bases de datos, sino previsiones razonadas de la evolución de los aspectos problema y propuestas valoradas con grado de idoneidad de la distintas decisiones... La construcción de este conocimiento se hace por la aplicación estructurada de técnicas de aprendizaje automático a partir de experiencias previas, cuantificadas, generadas mediante un proceso ordenado de pasadas de los modelos de simulación [Cuena,83]. Un sistema de este tipo puede generar propuestas de reglas que, supervisadas, refinadas y complementadas con

[2]Cuando nos refiramos al Control del Tráfico Urbano, esta actuación aparecerá con bastante claridad, al analizar el control de un cruce con respecto al control de toda la red.

UNIVERSITAT DE V... BIBLIOTEC... DE CIEN...

unidades de conocimiento personales por expertos en el tema, pertimirían producir la base de hechos definitiva. Este tipo de enfoque es criticable en el sentido de que desciende al nivel de detalle cuantitativo exigido por los modelos matemáticos para, en una etapa posterior, abstraer parte de estos detalles y referirlos con estructura de regla a escala cualitativa necesaria al nivel de decisión, y para integración de las distintas fuentes de conocimiento (modelos, experiencia en utilización de los modelos y los juicios de los expertos). Este inconveniente a la hora de operar, está paliado por el hecho de que el análisis cuantitativo es algo de que ya se dispone como consecuencia de la larga tradición en este tipo de análisis. Sin embargo, estas ideas justificarían plantearse directamente un razonamiento de tipo cualitativo que evitara este recorrido de ida y vuelta (de especificación y abstracción)".

Los sistemas de control de procesos son utilizados a menudo en las situaciones críticas, más aún, en situaciones hostiles en las que el usuario debe tener una maestría (que debe aplicar durante un cierto tiempo) de la situación y tomar una decisión rápida, teniendo el mejor conocimiento de causa posible. Por esto, hay que poner una atención especial a la hora de definir las especificaciones ergonómicas de un sistema de CP. Más concretamente, el interface hombre/máquina juega un papel muy importante, y se debe encargar de realizar la presentación de la información pertinente de la forma más sintética e inteligible posible.

Como hemos comentado en el apartado anterior, el interface hombre/máquina producido por los Sistemas de Control de Procesos tradicionales es poco efectivo, y sería deseable construir un sistema de control que lo mejorase. Este interface debe superar la presentación de los niveles de "timbre de alarma", para llegar a dar indicaciones que faciliten las tomas de decisiones del operador, con la criticidad de tiempo que cada aplicación requiera.

Un caso ejemplificador puede ser el CTU, en el que el casi inexistente interface hombre/máquina en los sistemas desarrollados hasta el momento, es la causa de que resulte difícil poder dedicar los suficientes recursos humanos a la tarea de supervisar el control automático, y la de que esta tarea resulte ingrata por la cualificación y la excesiva atención que requiere [Martín,88].

Por tanto, podemos concluir que la nueva aproximación al CP, genera unas nuevas necesidades de interface hombre/máquina, ya que sería de desear que el operador pudiera definir un modelo informativo orientado a unos determinados objetivos de decisión, y en su caso, la facilitación de la acción a tomar.

i.1.3 Toma de decisiones en Sistemas de Control

De los comentarios hechos en i.1.2 podría deducirse que muchos procesos son de hecho incontrolables, pero la experiencia demuestra que no sólo esto no es cierto, sino que además muchos de los sistemas "complejos" son controlados adecuadamente y a veces con relativa facilidad utilizando técnicas manuales o semiautomáticas. Este control no automático ofrece ventajas significativas sobre las metodologías de control tradicionales, pero desafortunadamente también tiene desventajas considerables, entre las que cabe citar:

- Es aplicable sólo a procesos con respuesta relativamente lenta, y se degrada bajo situaciones de tensión del operador, tanto más señaladas cuanto más crítico sea el tiempo de respuesta que se demanda.

- Aún alcanzando un aceptable nivel de calidad, muchas veces no se hace una utilización óptima de los recursos disponibles.

- Factores psicológicos tales como aburrimiento, cansancio, falta de motivación, etc., pueden llevar también a un deterioro en su efectividad (una clara desventaja en relación con los sistema automáticos).

- Resulta evidente que en un proceso que sea controlado a lo largo del tiempo por diferentes operadores humanos, las estrategias de control empleadas difieren notablemente de unos operadores a otros. Podríamos hablar pues, de una cierta "inconsistencia" en la estrategia de control.

Por lo apuntado hasta el momento, parece adecuado pensar en el desarrollo de un sistema automático de control "inteligente", que de alguna manera pueda combinar la capacidad de procesamiento cualitativo humano con la velocidad, capacidad y "seguridad" de las máquinas y así superar las limitaciones del control automático convencional.

i.1.4 Aspectos de Tiempo Real en Control de Procesos

Según Tebbs y Collins [Tebbs,77] un Sistema en Tiempo Real (abreviadamente STR) es aquel en el que se proporciona un servicio al usuario a través del uso de terminales unidas directamente al computador; los datos o informaciones de entrada

se reciben en los periféricos por el usuario sin que para ello tenga que para ello tenga que aplicar programas especiales.

El desarrollo de STR es un área en la que se ha pasado rápidamente de la fase pionera a la fase operativa y de uso intensivo con un buen nivel de elaboración, no sólo por que hay disponible hardware y software, sino también porque se han desarrollado y probado técnicas y procedimientos muy relevantes. Incluso la definición de "tiempo real" se ha concretado más. Según S.J. Young [Young,87] "En su sentido más amplio, el tiempo real puede utilizarse para describir cualquier actividad de proceso de información o sistema, que deba responder a estímulos de entrada generados externamente dentro de un intervalo especificable y finito". Evidentemente esta definición amplía la de Tebbs y Collins.

Un ordenador multiusuario dedicado a cálculo científico podría ser un ejemplo paradigmático; en él el tiempo de respuesta no es crítico. Sin embargo en CP es absolutamente necesario obtener una respuesta en un tiempo especificado; más generalmente esta es una necesidad de los "embebbed systems" y para manejar este tipo de sistemas se han diseñado diferentes lenguajes (RTL/2, Modula, Ada, etc.).

Desafortunadamente, las técnicas complejas de hardware y software específicas tienden a oscurecer los aspectos que son comunes a diferentes STR, y así, es dificultoso aplicar en otros ámbitos la metodología ya desarrollada.

En un sistema que funcione en tiempo real, como se desprende de su definición, la adquisición de datos por parte del sistema viene determinada por el mundo real, y no por el programador o usuario, como sucede en muchas de las aplicaciones de gestión actuales. Esto hace que el STR tenga que tener en cuenta más o menos simultáneamente y con restricciones de tiempo, acontecimientos que en principio pueden ser independientes, y por tanto debe de estar preparado para solucionar conflictos de prioridad y para filtrar adecuadamente las informaciones que va adquiriendo; a esto hay que añadir que la interacción del STR con el usuario o con el entorno, generalmente no tiene un carácter secuencial, y de aquí la necesidad de un tratamiento en "paralelo"del sistema, que conlleva la toma en cuenta de puntos de sincronización, tratamiento simultáneo de tareas con prioridades diferentes, y el mantenimiento de la coherencia en las operaciones de actualización.

Además de los problemas expuesto anteriormente que son comunes a los sistemas de control en tiempo real, al intentar incorporarle un Sistema Experto, surgen otros problemas adicionales [Sibille,87], [Laffey,88], [Repsol,92]:

i) Validez temporal de la reglas. Para superar esta limitación, nos vemos enfrentados a diversos problemas:

   1) Representación de los conocimientos temporales y utilización de un tipo de razonamiento distinto al empleado en los Sistemas Expertos "clásicos", y que ha de incorporar una componente temporal que afecta también a los hechos conocidos en un determinado momento.

   2) No monotonía del razonamiento, problema que se presenta en multitud de aplicaciones de SS EE.

   3) Se debe realizar Razonamiento por Defecto, como ya hemos mencionado y estudiaremos en su momento.

   4) Interrupción del razonamiento por el entorno externo que puede obligar a interrumpir procesos de deducción, para retomarlos más tarde.

   5) Los requerimientos de interface hombre/máquina señalados en i.1.2.

ii) La gestión de la memoria dinámica: El tratamiento de la información simbólica se vuelve eficaz gracias al reparto implícito de bloques de memoria para la representación de estructuras de datos, y a la recuperación automática de zonas de memoria que no se utilizarán.

Pero además de mencionar los problemas que surgen al incorporar el tiempo real a los sistemas expertos, también hay que destacar las ventajas de la utilización de un S.E. en Control de Procesos en tiempo real frente a un sistema de Control que no esté basado en el conocimiento; entre ellas caben destacar [Toledo,90]:

a) Utilización de la expertez humana que se puede extraer en múltiples campos: ayuda al diagnóstico, ayuda a la decisión, auto-control inteligente,...

b) Mejoría del interface hombre/máquina: en el soporte de la comunicación (tratamiento de la palabra, comprensión del lenguaje natural, comunicación a base

de imágenes,...) y en la utilización de un interface "inteligente" (en cuento que puede hacer una representación sinóptica y una explicación "on-line").

c) Una forma más fácil de representar y manejar información imprecisa, incierta, o errónea.

d) Una gran facilidad a la hora de actualizar el sistema para hacerle "más experto" o para que se ocupe de nuevos mecanismos de control.

i.2 MODELIZACION DE SISTEMAS FISICOS PARA SU CONTROL

La modelización de los sistemas físicos es una de las técnicas fundamentales empleadas por científicos e ingenieros en su trabajo. Para ellos el modelo de un sistema físico es la herramienta fundamental para comprender el funcionamiento del sistema y razonar acerca del mismo. Esto puede entenderse mejor considerando que el trabajo de los científicos y de los ingenieros puede describirse en lineas generales como el intento de comprender las diferencias entre los sistemas físicos y sus modelos. Los ingenieros generalmente asumen que sus modelos son correctos y tratan de encontrar malos comportamientos en los sistemas físicos basándose en estas diferencias. Los científicos refinan un modelo sucesivamente basándose en datos empíricos durante el proceso de formación de teorías [De Kleer,84].

La importancia de los modelos es tan grande que muchas veces olvidamos que el modelo no es más que una abstracción de la realidad, y lo tomamos como la realidad misma. Sin embargo, hay ocasiones en las que su utilidad estriba precisamente en que es una abstracción. La pregunta clave es entonces qué nivel de abstracción es necesario para una tarea determinada [Widman,89].

Esta idea es importante, pues indica que el modelo de un sistema no depende sólo del sistema, sino que depende tanto del sistema físico que se pretende modelar como del propósito específico al que se destina el modelo. Un mismo sistema físico puede ser modelado a través de varios modelos distintos, correspondiendo cada uno de ellos a cada uno de los diferentes propósitos a los que se destina el modelo, y siendo cada modelo inadecuado para los propósitos a los que no esta destinado [Leitch,90].

Por ejemplo, el modelo físico más preciso que describe el funcionamiento de un diodo es un modelo de electrodinámica cuántica que requiere conocer la disposición espacial de los componentes del diodo y de todos los elementos del universo que lo

puedan alterar por medio de alguna interacción (p.ej.: ondas electromagnéticas emitidas por componentes próximos). Sin embargo, este modelo no es adecuado para hacer un diseño de un circuito electrónico, pues es demasiado complejo. En su lugar se utiliza un modelo en el que las únicas interacciones del diodo con el exterior son la corriente y la tensión, y el diodo se describe por una función exponencial entre ambas, definida por dos constantes del diodo. Estas dos constantes es lo único que se utiliza para la representación de la estructura espacial (y química) del diodo. Naturalmente el comportamiento del mismo diodo según ambos modelos es distinto, pero sus diferencias se encuentran por debajo del nivel de detalle que un diseño electrónico requiere. Por consiguiente el modelo simple es el más adecuado para el propósito de diseño de un circuito electrónico, mientras que el modelo complejo es adecuado para el propósito de optimización de las características del diodo que pueda querer hacer el fabricante.

Para comprender la importancia de disponer de un modelo del control de un sistema físico, hagamos un pequeño análisis sobre la forma en que un experto humano trabaja con uno de estos sistemas de control. La tarea de un experto en un sistema de este tipo puede ser reducida a lo siguiente:

- Cuando controlamos un proceso normalmente pretendemos que su evolución temporal siga una línea determinada de antemano. Si esto no es posible nos contentamos con que su evolución temporal se mueva dentro de una región determinada.
- Cada vez que la evolución del sistema sale de esta región permitida, o da indicios de que esta a punto de salir de ella, estamos frente a un potencial problema. Lo que suele hacer el experto humano es identificar este tipo de situaciones con el objeto de encontrar una secuencia de acciones que puedan encauzar de nuevo la evolución del sistema dentro de los límites de la región anteriormente citada.

Si tomamos una aplicación de control de procesos cualquiera y anotamos todos los problemas que surjan junto con las soluciones que les fueron dadas por el experto, veremos que esta lista puede dividirse en dos partes: por un lado están aquellos problemas que el experto ya conocía y se limita a reconocerlos y aplicar la solución, y por el otro aquellos que el experto no conoce y debe razonar para encontrarles solución. En este segundo tipo de problemas el razonamiento se hace sobre un modelo del sistema que el humano tiene implícitamente en mente.

Tenemos pues dos tipos de razonamiento, uno basado en un conocimiento superficial y el otro en un conocimiento profundo (basado en un modelo del sistema que permite razonar sobre el mismo para encontrar soluciones a problemas no "programados"). Obviamente este "modelo" no es un modelo cuantitativo, sino es lo mas cualitativo posible.

En la construcción de Sistemas Expertos (ó Sistemas Basados en el Conocimiento), se han utilizado ambos tipos de conocimiento: A continuación mostramos la experiencia en la utilización de ambos.

i.2.1 Conocimiento Superficial

El Conocimiento Superficial puede definirse como un tipo de conocimiento que relaciona directamente las premisas que pueden conocerse sobre un sistema con las conclusiones que pueden obtenerse de las mismas. En una aplicación de Control, podemos decir que está constituido por un conjunto de elementos de conocimiento que representan una relación directa entre los patrones de problemas conocidos y las acciones necesarias para solucionarlos. En la construcción de estas Bases de Conocimiento Superficiales, el ingeniero del conocimiento introduce reglas elaboradas por él, pero no introduce ningún elemento perteneciente al proceso de razonamiento que le condujo a la elaboración de dichas reglas ni el auténtico *por qué* de las reglas [Salmerón,92].

La mayor utilidad de este tipo de Bases de Conocimiento consiste en su capacidad de ser gestionadas muy rápidamente, debido a su estructura planar, sin apenas interdependencias entre los distintos elementos de razonamiento que la componen; sin embargo, los sistemas expertos basados en este tipo de conocimiento presentan un gran número de limitaciones, enumeradas por [Widman,89], que están causadas por la concepción del mismo como un gran conjunto de reglas de aplicación directa sin base justificativa. [Winston,87] bautizó a este tipo de sistemas basados en reglas como *sabios idiotas*, capaces de almacenar una gran cantidad de conocimiento en su interior pero incapaces de identificar los razonamientos elementales que se ocultan tras sus propias reglas. Una importante consecuencia de este hecho consiste en su incapacidad para explicar razonablemente la razón por la que se toma una determinada acción para solucionar un problema, puesto que su conocimiento no contiene esta información (y si la contiene, es como añadido a la Base de Conocimiento, sin relación con la misma). Esta afirmación puede ilustrarse con el siguiente ejemplo de explicación, típico en un sistema de este tipo: "Se toma la

acción 23 para solucionar el problema 12 porque la regla 53 dice: SI problema_12 ENTONCES acción_23". Evidentemente este ejemplo es muy simple, pero ayuda a ilustrar el problema del conocimiento superficial.

i.2.2 Conocimiento Profundo

La principal conclusión que cabe extraer del apartado anterior sobre las Bases de Conocimiento Superficial, es que éstas contienen únicamente la manifestación externa o superficial del conocimiento real que un experto tendría sobre el tema tratado [Salmerón,92] [Bonissone,85]. Este conocimiento real al que aludimos, y que es la base que explica las reglas utilizadas en una Base de Conocimiento Superficial, utiliza otras estructuras de razonamiento diferentes, mas próximas al razonamiento real de un experto. Estas estructuras son las teorías sobre el dominio de aplicación: conocimiento sobre la estructura interna, funcionamiento y/o composición de los objetos de estudio (conocimiento profundo) [Forbus,88] y conocimientos generales sobre resolución de problemas. Esta clasificación del conocimiento permite su estructuración en dos niveles [Salmerón,92]: a) un nivel inferior con conocimiento que explica el comportamiento del sistema sobre el que se razona y b) un nivel superior, nivel de control, con conocimiento sobre la forma de usar el nivel inferior para buscar soluciones al problema que se plantea. Las reglas heurísticas que aparentemente constituyen el conocimiento de un experto sobre un tema no son más que las conclusiones de un proceso de razonamiento de carácter general guiado por los conocimientos específicos de un dominio profesional que son las teorías que lo explican.

Originalmente, las implementaciones de sistemas expertos se basaron en representaciones de conocimiento superficial, el cual era generado por expertos a partir de su conocimiento profundo. Por ejemplo, una regla superficial en Control de Tráfico Urbano puede reducir el tiempo de verde en una calle como respuesta a una congestión a gran distancia del lugar. Por debajo de esta regla superficial subyace el modelo de tráfico que explica la forma de generarse una congestión en una red urbana. Una congestión se genera cuando el flujo entrante en una zona excede a su capacidad máxima de desagüe, y la causa suele ser la concurrencia sobre la zona congestionada de flujos ligeramente elevados generados en puntos bastante alejados. Por consiguiente, debemos restringir el paso de vehículos (en lugares donde sea posible hacerlo sin producir males mayores) para relajar la presión sobre la zona problema. El ingeniero de tráfico conoce estos hechos, y los desarrolla generando reglas superficiales de aplicación rápida, como la enunciada previamente. Por eso

puede llegar a decirse que un modelo superficial no es más que la compilación de un modelo profundo [Salmerón,92] [Chandrasekaran,82].

Veamos ahora las principales ventajas e inconvenientes que puede traer la utilización del conocimiento profundo en vez del superficial.

La principal ventaja que el conocimiento profundo tiene estriba en la amplitud de su campo de actuación, el cual ya no se ve limitado a la solución de problemas prototípicos, como en el caso anterior. A igualdad de tamaño de la base de conocimiento es capaz de solucionar un número de problemas bastante superior que un sistema basado en conocimiento superficial, debido a que analiza los problemas y deduce las acciones a tomar, en lugar de tener almacenadas todas las relaciones problema-acción. Ademas es capaz de explicar porqué ha tomado una determinada acción para solucionar un problema dado en función del comportamiento del modelo, con lo que su explicación es bastante más completa que en el caso anterior.

La principal desventaja de la utilización del conocimiento profundo es que requiere tiempos de computación elevados, aunque esta limitación tiene un carácter muy relativo, habida cuenta del ritmo al que evoluciona la capacidad de proceso de los ordenadores.

En vista de lo anteriormente comentado sobre el conocimiento profundo y superficial, podemos concluir abogando por la utilización de conocimiento profundo en los sistemas expertos actuales, lo que nos llevará a una **segunda generación de sistemas expertos** [Salmerón,92], cuya característica principal será la estructuración de sus bases de conocimiento en distintos niveles de representación que cooperen para la consecución de los objetivos del sistema, aumentando de esta forma la claridad y la capacidad declarativa de sus bases de conocimiento.

i.2.3 Técnicas de Modelización

La modelización de sistemas físicos se ha hecho con diversas técnicas que pueden agruparse en dos: **cuantitativas** (originadas en el campo de las matemáticas aplicadas y de la estadística) y **cualitativas** (originadas en el campo de la I.A.). De las primeras, puesto que no son el objetivo de la tesis, sólo vamos a presentar brevemente sus principios, y estudiar las ventajas y desventajas que reporta su utilización.

i.2.3.1 Técnicas cuantitativas

Las técnicas cuantitativas de modelización son aquellas que modelan el mundo real por medio de un conjunto de funciones numéricas, cuya evolución temporal está generalmente regida por un sistema de ecuaciones diferenciales o ecuaciones en diferencias finitas.

Debido a que las teorías físicas más exactas que conocemos en la actualidad se expresan principalmente a través de ecuaciones diferenciales, los modelos más precisos suelen ser los realizados mediante técnicas cuantitativas. Esta precisión es la mayor virtud obtenida con este tipo de técnicas.

Pero sin embargo tienen bastantes desventajas, de entre las cuales citamos las siguientes:

a) El tiempo de cálculo que precisan es generalmente muy elevado, lo cual limita su utilización en una aplicación que deba funcionar en tiempo cercano al real.

b) La descripción del estado inicial requiere un conocimiento completo del estado del mundo real que normalmente no se posee, debido a que el número de sensores es limitado. Esto implica que sólo puede conocerse una parte del estado inicial, y que debe completarse la información a partir de supuestos razonables. Sin embargo pueden derivarse consecuencias graves de este proceder si se trabaja con un sistema de baja estabilidad, en el que un pequeño error en el estado inicial se amplifica a lo largo del proceso de simulación.

c) El almacenamiento en memoria suele ser muy elevado, tanto para la descripción del estado inicial del simulador como para el cálculo de su evolución. Este hecho supone una limitación a su utilización en ciertas aplicaciones. Bien es cierto que con la rápida evolución en la tecnología de ordenadores, este punto cada vez supone una limitación menor.

d) Es muy difícil para un humano el entender los diversos aspectos del funcionamiento del sistema a partir de los datos generados por un modelo cuantitativo, puesto que el humano trabaja a un nivel de abstracción superior al del modelo.

i.2.3.2 Técnicas cualitativas

La razón que ha motivado el desarrollo de una física cualitativa ha sido la identificación del conocimiento subyacente a la intuición física. Las personas aparentemente utilizan un cálculo cualitativo casual para razonar sobre el entorno físico y lo hacen de una forma rápida y eficaz. Por ejemplo, las ecuaciones que describen el movimiento de un vehículo en carretera son de una complejidad bastante elevada; entre ellas se incluyen ecuaciones de termodinámica, rozamientos, vibraciones, aerodinámica, gravedad, etc; y deben incluírse los efectos exteriores (viento, lluvia, etc.) y de los vehículos adyacentes. A pesar de ello conducir es relativamente fácil. Cuando se entra en una curva se calcula cualitativamente la velocidad a la que se debe circular, evaluando rápidamente los factores involucrados. El tipo de razonamiento que nos permite hacer esto es, evidentemente, muy diferente de la física clásica. Este tipo de razonamiento y muchos otros como él forman la base de lo que se conoce como física cualitativa [Salmerón,92], y en la actualidad existen ya numerosas técnicas que abordan el estudio de los sistemas físicos de forma cualitativa, en oposición o como complemento a las técnicas tradicionales.

Hay muchas razones que aconsejan utilizan técnicas de modelización cualitativas en ciertas aplicaciones, de entre las cuales citaremos aquí algunas de las más importantes:

a) El tiempo de cálculo que precisan es bastante más reducido que en las cuantitativas, lo que convierte a este tipo de técnicas en las únicas utilizables en los casos en que las técnicas cuantitativas son temporalmente prohibitivas.

b) El almacenamiento en memoria es más reducido que en las técnicas cuantitativas.

c) La descripción del estado inicial consiste en una descripción incompleta del estado del mundo real, por lo que está más en concordancia con la incompletitud de los datos recibidos, motivada por la finitud del número de sensores. Por consiguiente, aunque el simulador sea mas inexacto que en el caso anterior, se encuentra alimentado con datos suficientes para él.

d) Puesto que las técnicas cualitativas trabajan a un nivel de abstracción cercano al de la mente de un experto humano, es mucho más sencillo para un experto humano el entender los datos generados por el mismo.

Los intentos realizados para conseguir razonar sobre el comportamiento de sistemas físicos han conducido al desarrollo de la física cualitativa, nombre por el que se conoce un grupo de teorías sobre representación y razonamiento sobre sistemas físicos que presentan varios rasgos en común.

En palabras de De Kleer, uno de los fundadores del campo, los objetivos comunes de todas estas teorías son los siguientes [De Kleer,84]:

1- Ser más simple que los modelos físicos basados en métodos cuantitativos, pero reteniendo sin embargo las distinciones importantes sin invocar las matemáticas de variables continuas y ecuaciones diferenciales (aunque en algunas aproximaciones la modelización matemática está subyacente).

2- Producir explicaciones causales de los mecanismos, y que éstas sean fáciles de comprender.

3- Proporcionar los fundamentos de modelos de sentido común para la próxima generación de Sistemas Basados en el Conocimiento.

Vamos a proceder a una revisión y puesta al día de las técnicas de razonamiento cualitativo más utilizadas. Entre las mismas he escogido las técnicas de Kuipers, De Kleer y Forbus como las más significativas (aunque sin embargo podemos encontrar también otras aproximaciones en la literatura: [Williams,84], [Leitch,90], y muchas más). Pasaremos seguidamente a analizarlas con más detalle.

i.2.3.2.1 Método de Kuipers

La técnica de modelización cualitativa desarrollada por Kuipers [Kuipers,86] está basada en un tratamiento cualitativo de las ecuaciones diferenciales utilizadas para describir un sistema según la física tradicional.

Partimos del modelo físico de un sistema formado por un conjunto de ecuaciones. El primer paso es reescribir estas ecuaciones en forma de restricciones aritméticas y funcionales. Seguidamente se hace un proceso de interpretación cualitativa que acaba generando un grafo de transiciones entre estados cuyos caminos describen los posibles comportamientos del sistema a lo largo del tiempo [Cuena,89].

Describiremos seguidamente el marco conceptual de la representación. Un sistema físico se caracteriza por un conjunto de atributos, los cuales son funciones reales del tiempo (F(t)). Si el intervalo temporal de estudio es [a,b], se exige que las funciones:

- sean continuas en [a,b]
- sean continuamente diferenciables en (a,b)
- tengan un número finito de puntos críticos (mínimos, máximos, etc...) en [a,b]
- Tengan derivada por la derecha en a y por la izquierda en b

Nótese que las funciones más usuales utilizadas en un sistema real cumplen estas condiciones.

Se definen **puntos hito** de una función real definida en un intervalo [a,b] a los puntos: a, b, máximos, mínimos, cortes con el eje de ordenadas e infinitos; y **momentos relevantes** al tiempo en que ocurren.

Las funciones reales son cuantificadas con un espacio cuántico formado por puntos (valores hito) e intervalos (intervalos entre valores hito). El estado cualitativo de cada función toma valor de su espacio cuántico correspondiente. El estado del sistema se compone por los estados cualitativos de todas sus funciones.

Este proceso de representación cualitativa de las funciones reales puede verse en la figura i.1:

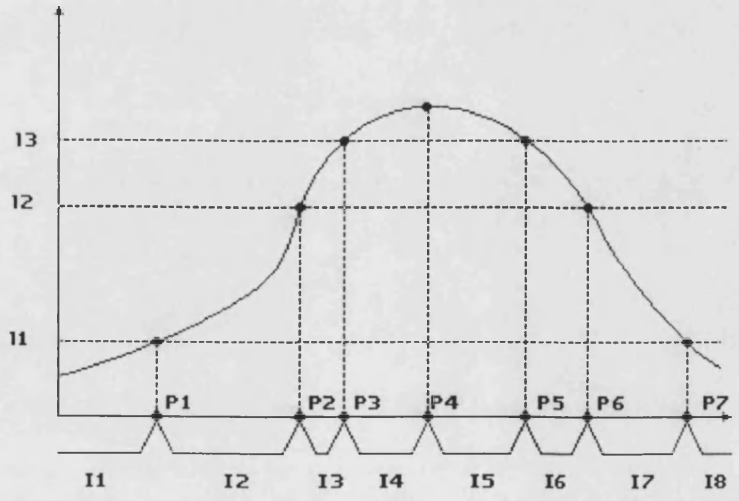

Figura i.1

P1	QS(f,t1)	<l1,cre>
I2	QS(f,t1,t2)	<(l1,l2),cre>
P2	QS(f,t2)	<l2,cre>
I3	QS(f,t2,t3)	<(l2,l3),cre>
P3	QS(f,t3)	<l3,cre>
I4	QS(f,t3,t4)	<(l3,l4),cre>
P4	QS(f,t4)	<(l3,l4),est>
I5	QS(f,t4,t5)	<(l3,l4),dec>
P5	QS(f,t5)	<l3,dec>
I6	QS(f,t5,t6)	<(l2,l3),dec>
P6	QS(f,t6)	<l2,dec>
I7	QS(f,t6,t7)	<(l1,l2),dec>
P7	QS(f,t7)	<l1,dec>

Tabla i.2

Las funciones imponen unas restricciones generales a la evolución del sistema (por aplicación del Teorema del valor medio). Si a estas restricciones se les añaden las restricciones obtenidas al representar las ecuaciones diferenciales obtenemos todas las restricciones que deben cumplirse entre las funciones.

Las restricciones que define Kuipers son las siguientes:

- aritméticas:
  ADD(f,g,h) h = f + g
  MULT(f,g,h) h = f * g
  MINUS(f,g) f = -g
- funcionales:
  M+(f,g) signo(f') = signo(g')
  M-(f,g) signo(f') = - signo(g')
- derivadas:
  DERIV(f,g) g = f'

Y todas las ecuaciones diferenciales se escriben en función de ellas.
Por ejemplo:
  u'' + u' + arctan ku = 0

Definiendo: f1 = u' ; f2 = f1' ; f3 = ku ; f4 = arctan f3;

entonces:

DERIV(u,f1)
DERIV(f1,f2)
MULT(k,u,f3)
M+(f3,f4)

sería el conjunto de restricciones cualitativas que representaría la ecuación anterior.

Escojamos un estado inicial para el sistema. Definamos una lista de nodos activos que contenga sólo el estado inicial; entonces, para generar el grafo de posibles transiciones entre estados se obra de la manera siguiente:

Para cada nodo de la lista activos, usar el conjunto de restricciones para determinar las transiciones posibles desde ese nodo a otros nodos. Borrar el nodo de la lista. Aplicar un filtro a estos estados sucesores para evitar ciclos, e insertar los que queden en la lista activos. Repetir hasta que la lista activos este vacía.

Este algoritmo recorre el grafo que buscamos. Basta almacenar las transiciones a medida que las calcula para que al final del algoritmo tengamos el grafo. Básicamente lo que hace el algoritmo es calcular sucesivamente estados sucesores a partir de un estado inicial, formando el grafo paso a paso. El final está asegurado porque el número de estados del sistema es finito.

i.2.3.2.2 Método de De Kleer

La física cualitativa basada en confluencias es la teoría mas cercana a la visión que un ingeniero tiene del mundo real. Las confluencias fueron propuestas por De Kleer y Brown [De Kleer,83] y permiten obtener una visión basada en componentes de la modelización cualitativa. Esta aproximación puede resumirse como sigue:

1- Un sistema consiste de partes físicamente distintas interconectadas,

La investigación de De Kleer y Brown en el Centro de Investigación de Xerox en Palo Alto se encuentra entre los trabajos mas importantes sobre razonamiento cualitativo en sistemas físicos. Su trabajo es motivado por el deseo de identificar el conocimiento central que sirve de soporte a las intuiciones físicas de las personas. En

este trabajo se intenta describir los fenómenos físicos de una forma mas simple que en la física clásica pero conservando todos los aspectos importantes de los procesos físicos.

El resultado de este trabajo fue la definición del proceso conocido como ENVISION, que predice el comportamiento de un dispositivo como una secuencia de posibles estados futuros junto con un análisis causal completo de su comportamiento. Para ello requiere conocer: La estructura física del dispositivo, las reglas de comportamiento de cada componente, y una entrada suministrada al dispositivo.

Modelo del Dispositivo:

El proceso de ENVISION utiliza una visión orientada a componentes de un dispositivo. El dispositivo debe pues ser dividido en un conjunto de componentes físicamente disjuntos interconectados a través de "conductos".

Definimos los conductos como partes físicas de un dispositivo cuya única función es transmitir información entre componentes sin alterar la información. Esto es una idealización y puede no ser estrictamente verdad en la realidad, pero es una asunción razonable en muchas situaciones.

Por otro lado los componentes tienen un comportamiento más complejo. Estos se encuentran agrupados en clases, y el comportamiento de cada clase (y por ende, de todos los objetos que pertenecen a ella) viene determinado por un conjunto de reglas. Este conjunto de reglas se encuentra codificado en forma de lo que De Kleer denomina confluencias.

Para describir un dispositivo siguiendo el esquema de De Kleer, su especificación debe cumplir dos principios fundamentales:

1- **Principio de localidad:** La descripción de un componente no debe estar referida a ninguna otra parte del dispositivo (de esta manera nos aseguramos que la información entre componentes sólo pueda pasar a través de los conductos).

2- **Principio de no función en la estructura:** La descripción de un componente no debe presuponer el funcionamiento del todo (si un componente realiza una función particular en el dispositivo, esta función no debe formar parte

integrante de la descripción del componente, pues presupone que ese componente esta conectado a ese dispositivo en particular, y la definición no es transportable a otros dispositivos).

Vamos a representar el dispositivo mostrado en la figura siguiente utilizando la aproximación de ENVISION para ilustrarla.

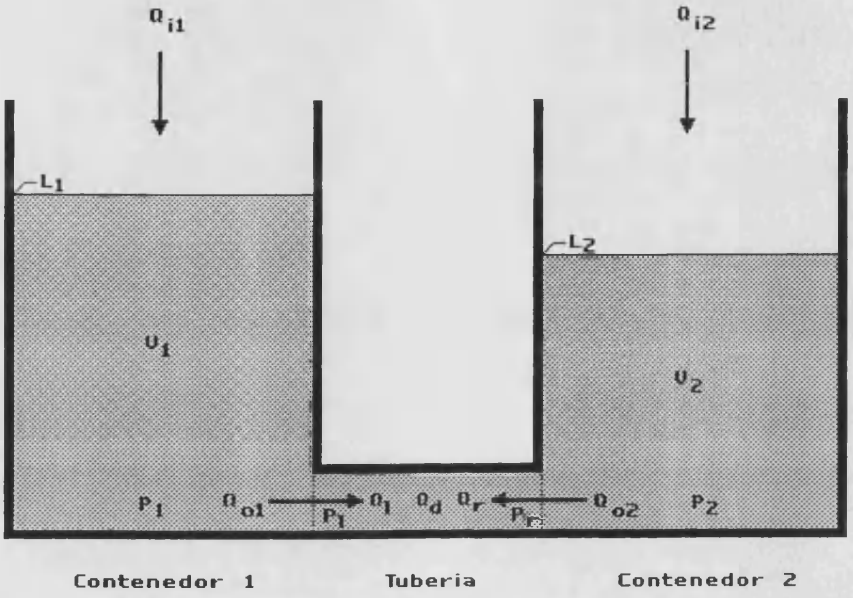

Figura i.3

Este dispositivo representa el famoso problema de los vasos comunicantes. El dispositivo se compone de dos tanques conectados por una tubería. Siguiendo la nomenclatura de De Kleer y Brown, los componentes son los dos tanques y la tubería es el conducto. Existen pues dos componentes, ambos pertenecientes a la misma clase. Esta clase es la clase de los contenedores de liquido con un agujero en el fondo, cuyas reglas de comportamiento debemos codificar en forma de confluencias.

Para describir la clase tanque, el primer paso es especificar las variables que necesitamos para ello:

P- Presión en el fondo del contenedor.
L- Nivel de liquido en el contenedor.
V- Volumen de liquido en el contenedor.
Qi- Flujo de entrada de liquido en el contenedor desde arriba.
Qo- Flujo de salida del liquido del contenedor a través del agujero del fondo.

Debido al significado físico de cada variable, tenemos varias restricciones a los valores que pueden tomar, que son: P>=0, L >=0, V >=0, Qi >=0.

Por otro lado, las ecuaciones de la física clásica imponen una serie de restricciones entre los valores de las variables que son susceptibles de ser codificadas de manera cualitativa:

a) La presión es proporcional al nivel del agua.

En física clásica esta restricción es del tipo P = L x C, siendo C una constante positiva.
Si derivamos con respecto al tiempo y la convertimos en ecuación cualitativa (expresando cada variable x por [x] = signo de x , Dominio[x] = {-,0,+}) obtenemos una confluencia:

[dP] = [dL]

Esta confluencia indica que el signo de la derivada temporal de P es igual al signo de la derivada temporal de L, que es como decir que hay una relación monótona estrictamente creciente entre ellas, por lo que si una crece, la otra también lo hace, y si una decrece, la otra decrece también.

b) El nivel del agua es una función monótona estrictamente creciente del volumen.

[dL] = [dV]

c) La diferencia entre flujo de entrada y salida es el cambio del volumen.

[Qi] - [Qo] = [dV]

Para describir el conducto, necesitamos definir cinco variables:

Ql, Qr : Flujos de entrada al conducto desde los dos extremos del mismo.
Pl, Pr : Presión en los dos extremos del conducto.
Pd : diferencia entre las presiones a los extremos del conducto.

Entre estas variables tenemos las siguientes restricciones:

d) Conservación del material.

[dQl] = - [dQr]

e) El flujo es proporcional a la diferencia de presiones. [dQl] = [dPd]

f) Por definición.

[dPd] = [dPl] - [dPr]

Una vez tenemos todas las confluencias, podemos realizar la representación de este dispositivo según el proceso de ENVISION:

- Restricciones para el contenedor C1:
  1) [dP1] = [dL1]
  2) [dL1] = [dV1]
  3) [Qi1] - [Qo1] = [dV1]

- Restricciones para el contenedor C2:
  4) [dP2] = [dL2]
  5) [dL2] = [dV2]
  6) [Qi2] - [Qo2] = [dV2]

- Restricciones para el conducto:
  7) Ql = - Qr
  8) [dQl] = - [dQr]
  9) [dQl] = [dPd]
  10) [dPd] = [dPl] - [dPr]

- Restricciones de conexión entre el conducto y el contenedor C1:
  11) [Qo1] = [Ql]
  12) [dQo1] = [dQl]
  13) [dP1] = [dPl]

- Restricciones de conexión entre el conducto y el contenedor C2:
  14) [Qo2] = [Qr]
  15) [dQo2] = [dQr]
  16) [dP2] = [dPr]

Una vez representado el sistema, vamos a estudiar cómo se predice el comportamiento del mismo utilizando la técnica de De Kleer.

Definimos un estado del sistema como el conjunto de valores cualitativos de todas las variables del sistema. Dentro de un estado ninguna variable puede cambiar de valor cualitativo, pues un cambio de valor de una variable marcaría un cambio de estado del sistema.

De Kleer y Brown definen dos tipos distintos de comportamiento, uno referido a las relaciones entre las variables dentro de un mismo estado y otro referido a las relaciones entre las variables de estados contiguos:

1 - Comportamiento interno del estado:

El comportamiento interno de un estado se determina propagando los valores de los parámetros cuyos valores son conocidos a otras variables a través de restricciones cualitativas para determinar sus valores.

En este proceso de propagar valores, el programa debe a menudo hacer asunciones sobre los valores. Para ello se utilizan algunas reglas heurísticas, como por ejemplo: "Si en una ecuación hay variables que están cambiando y variables desconocidas, asumir que las variables desconocidas no cambian".

2 - Causalidad Mítica:

El proceso de propagación de valores a otras variables a través de restricciones tiene un significado causal. Por ejemplo, si [a] = + y [a] + [b] = 0, se deduce que [b] = -, y se considera [a] = + como la causa de que [b] = -. Pero como el proceso de propagación de valores se asume instantáneo, no se trata del concepto usual de causalidad. A este concepto de causalidad instantánea le llama De Kleer causalidad mítica.

Retomando el ejemplo de los vasos comunicantes, vamos a calcular los valores de las variables sabiendo que estamos añadiendo agua al contenedor 1 y que no entra agua por arriba al contenedor 2:

17) [Qi2] = 0 dado

18) [Qi1] = + dado
19) [Qo1] = 0 dada la condición inicial
20) [dV1] = + (3),(18),(19)
21) [dL1] = + (2),(20)
22) [dP1] = + (1),(21)
23) [dPl] = + (13),(22)
24) [dPd] = + (10),(23),
asumiendo dPr despreciable
25) [Ql] = 0 (11),(19)
26) [Qr] = 0 (7),(24)
27) [Qo2] = 0 (14),(25)
28) [dV2] = 0 (6),(17),(26)
29) [dL2] = 0 (5),(27)
30) [dP2] = 0 (4),(28)
31) [dPr] = 0 (16),(29)
32) [dQl] = + (9),(31)
33) [dQo1] = + (11),(32)
34) [dQr] = - (8),(32)
35) [dQo2] = - (15),(34)

La única asunción hecha aquí está en la línea 24, y es suponer dPd despreciable[3]. No es la única asunción que podíamos hacer. Si se introduce una asunción en el proceso de propagar valores, ésta puede conducir a una lista de valores para las variables que satisfagan todas las combinaciones, o puede conducir a una contradicción. En este último caso, la asunción es incorrecta y debe ser eliminada.

Es posible que al final obtengamos distintos conjuntos de valores para las variables, todos ellos consistentes tanto con los valores dados como con las restricciones; en este caso todos ellos son explicaciones causales míticas validas. Como el número de explicaciones distintas que obtenemos depende del número de asunciones introducidas, y es posible hacer un número enorme de asunciones, el proceso de ENVISION dispone de reglas heurísticas para limitar las asunciones que se hagan en diferentes situaciones, con lo que se consigue reducir el número de interpretaciones causales míticas producidas por el sistema.

[3] Podemos hacer la asunción de que una variable es despreciable, y más tarde descubrir que una perturbación le asigna una valor no nulo. ENVISION interpreta esto como indicación de un posible bucle de realimentación. Si en la descripción estructural del dispositivo hay un auténtico bucle de realimentación, se interpreta como tal, pero si no lo hay, ENVISION decide que el bucle es ficticio.

Una vez se ha determinado completamente el comportamiento interestado, se calculan los estados futuros del dispositivo. En el ejemplo de los vasos comunicantes, hay cinco estados posibles del sistema que cumplan [Qi1] = + y [Qi2] = 0; y un grafo de transiciones posibles entre estos estados.

i.2.3.2.3 Método de Forbus

La Teoría Cualitativa de Procesos (Qualitative Process Theory, QPT) fue desarrollada por Ken Forbus en el MIT [Forbus,83][Forbus,84]. Su objetivo es la comprensión del razonamiento de sentido común sobre los procesos físicos. Para llevar a la práctica la QPT Forbus desarrollo un programa, el Qualitative Process Engine (QPE), que utiliza conocimiento general sobre los procesos y objetos físicos en el mundo, para inferir el comportamiento temporal del sistema (deduce los procesos que ocurrirán, que efectos tendrán, y cuando pararán).

En la QPT, las situaciones físicas son modeladas como conjuntos de objetos, de relaciones entre objetos y de procesos. Los objetos tienen variables para representar sus propiedades, y los procesos actúan a lo largo del tiempo cambiando estas propiedades. Los procesos son los únicos agentes del cambio, razón por la que la descripción del comportamiento temporal de un sistema se da en términos de procesos y de sus efectos sobre el sistema.

La QTP proporciona dos herramientas complementarias de trabajo:

1) Un lenguaje de representación de procesos físicos, en el que los procesos se definen en función de sus precondiciones y sus efectos.

2) Un motor de inferencia para predicción (Qualitative Process Engine, QPE), que utiliza una teoría física concreta descrita con el lenguaje de la QPT para predecir el comportamiento futuro de un sistema.

El funcionamiento del QPE se divide en cuatro fases:

1) Dado un conjunto de objetos y de conocimiento general de procesos, se decide qué instancias de procesos pueden existir en la situación dada.

UNIVERSITAT DE VALÈNCIA BIBLIOTECA DE CIENCIES

2) Determinar qué instancias de procesos están activas examinando si sus condiciones se satisfacen.

3) Determinar qué cambios serán causados por los procesos activos. Cuando varios procesos afectan a una variable, se intenta determinar el efecto conjunto sobre la variable.

4) Predecir el comportamiento a lo largo del tiempo. Los cambios causados por los procesos activos pueden satisfacer las precondiciones de otros procesos que estaban inactivos, o hacer falsas las de procesos que estaban activos. Esto provoca la activación o desactivación de nuevos procesos como resultado del comportamiento temporal de cada proceso. El QPE produce un informe sobre la actividad de los procesos a lo largo del tiempo.

A continuación describiremos la representación en la QPT de objetos y procesos, e ilustraremos el procedimiento de "envision" del QPE con un ejemplo.

i.2.3.2.3.1 Representación de Objetos

Una situación física se describe como un conjunto de objetos y de sus relaciones. Los objetos en el mundo real pueden mostrar características muy dispares dependiendo de las condiciones a que están sometidos,incluso estando formados por la misma sustancia. Por ejemplo, el agua puede estar en forma sólida, líquida o gaseosa. Incluso el agua líquida se considera de forma diferente si esta completamente contenida en un depósito o si es parte de una masa mayor de líquido por la que puede moverse en cualquier dirección. Las vistas de objetos que se centran en las condiciones de sus características de comportamiento se llaman **vistas individuales** en la QPT.

Pueden haber vistas individuales para tipos genéricos de objetos, objetos bajo condiciones específicas (comprimidos, evaporados, ...), y combinaciones de objetos con interrelaciones particulares. Algunos ejemplos de vistas individuales son sólidos, líquidos, gaseosos, objetos elásticos, objetos plásticos, objetos comprimidos, líquidos contenidos, etc. Las vistas individuales se organizan en una jerarquía en la base de conocimiento de QPE.

Para definir una vista individual hay que especificar las siguientes cuatro partes:

1) **Individuales:** Los objetos involucrados. Esta parte especifica el conjunto de objetos que deben existir para que la vista individual pueda mantenerse. Por ejemplo, las vistas individuales de objeto comprimido y objeto relajado se darán para unos objetos y no para otros (los no elásticos).

2) **Condiciones de Cantidad:** Las inecuaciones que deben cumplir las cantidades de los "individuales", y las vistas individuales y procesos que deben estar activos para que exista esta vista individual. Por ejemplo, para que pueda estar activa la vista individual "objeto comprimido", debe cumplirse la inecuación: longitud objeto < longitud del objeto en reposo.

3) **Precondiciones:** Condiciones que deben cumplir los "individuales" y que están relacionadas a sus cantidades. Por ejemplo, para que la vista individual de "objeto comprimido" pueda mantenerse, es necesario que el objeto está hecho de una sustancia elástica.

4) **Relaciones:** Relaciones entre atributos de los "individuales" que se mantienen mientras la vista individual esté activa. Por ejemplo, si la vista individual de gas está activa, también lo está la relación PV = nRT entre sus atributos temperatura, presión y volumen.

Los "individuales", las precondiciones y las condiciones de cantidad especifican las condiciones suficientes para que la vista individual se mantenga, mientras que las relaciones muestran las consecuencias producidas por la vista individual, así como las propiedades de los "individuales".

Con objeto de ilustrar la definición de vista individual, mostramos seguidamente un ejemplo de definición de vista individual, correspondiente a un líquido contenido en un recipiente:

Vista individual: liquido_contenido
Individuales:
con: un contenedor sub: un líquido
Precondiciones:
con puede contener sub
Condiciones de Cantidad:
La cantidad de sub en con es mayor que cero.
Relaciones:

Hay un P tal que:
- P es una porción de materia
- la substancia de P es sub
- P esta en con
- la cantidad de p es igual a la cantidad de sub en con

i.2.3.2.3.2 Representación de Procesos

Los únicos agentes del cambio sobre los objetos son los procesos. La representación de un proceso es similar a la de una vista individual excepto que la del proceso debe describir sus propios efectos. La descripción de un proceso incluye pues: los objetos que deben existir para que el proceso ocurra, las circunstancias en las que ocurrirá el proceso, y los cambios que serán producidos por el proceso.

Como ejemplo vamos a ver la descripción del proceso de intercambio de calor entre dos cuerpos:

Proceso: flujo_de_calor
Individuales:
src: un objeto con una cantidad, calor_fuente.
dst: un objeto con una cantidad, calor_destino.
camino: un camino por donde el calor pueda viajar de src a dst.
Precondiciones:
camino no debe estar obstruido.
Condiciones de Cantidad:
temperatura_fuente > temperatura_destino.
Relaciones:
Sea velocidad_de_flujo una cantidad.
velocidad_de_flujo > 0.
velocidad_de_flujo AQ+ (temperatura_fuente - temperatura_destino).
Influencias:
I-(calor_fuente,velocidad_de_flujo).
I+(calor_destino,velocidad_de_flujo).

En esta definición de proceso, src, dst y camino son variables que toman el valor de vistas individuales o de otros procesos cuando se crea una instancia del proceso.

Ahora debemos explicar la notación de la QPT para describir relaciones matemáticas cualitativas entre cantidades. En la QPT, I+(a,b) y I-(a,b) denotan una influencia directa positiva o negativa de b sobre a. I+(a,b) significa que a es incrementado en una cantidad b, y también que a depende causalmente de b. De forma similar, a AQ+ b y a AQ- b denotan una relación de proporcionalidad cualitativa entre a y b, y que a depende causalmente de b.

La definición de un proceso consta de cinco partes:

1) **Individuales:** los "individuales" involucrados, que pueden ser objetos o instancias de vistas individuales.

2) **Precondiciones:** Condiciones no cuantitativas que deben cumplirse para que el proceso pueda mantenerse.

3) **Condiciones de cantidad:** Condiciones sobre las cantidades de los "individuales" que deben ser satisfechas para que el proceso pueda mantenerse.

4) **Relaciones:** El conjunto de relaciones que el proceso impone sobre los "individuales", y también las nuevas entidades que son creadas por actuación del proceso. Por ejemplo, en el proceso "hervir" se crea la entidad "vapor".

5) **Influencias:** Las influencias directas causadas por el proceso sobre las variables de los "individuales". La diferencia entre Relaciones e Influencias consiste en que las influencias son las causas primarias del cambio producido por un proceso, mientras que las relaciones no son más que condiciones que deben cumplirse entre las cantidades mientras el proceso tiene lugar.

i.2.3.2.3.3 Predicción del Comportamiento con la QPT

Dada una descripción del mundo real compuesta por un conjunto de objetos y sus relaciones mutuas, el QPE busca primero qué vistas individuales son válidas en la situación presente. Si los objetos en la escena satisfacen las condiciones de definición de una vista individual (individuales, precondiciones y condiciones de cantidad), se crea una instancia de esa vista individual. De forma similar se crean las instancias de los procesos: éstas son declaradas activas si sus condiciones son satisfechas.

Las vistas individuales y los procesos imponen relaciones funcionales cualitativas sobre las variables de los objetos. Además los procesos activos especifican qué variables cambian y cómo. Dadas estas relaciones y cambios, la QPT predice el curso futuro de los acontecimientos en términos de procesos que suceden y se detienen, provocando cambios en la situación.

La QPT utiliza la representación de Allen del tiempo [Allen,84]. En ella el tiempo se compone de intervalos, que están relacionados por medio de ciertas relaciones como antes, después y durante. La representación en la QPT del comportamiento, utiliza la noción de Hayes de historia [Hayes,**]. El comportamiento se representa como un conjunto de fragmentos, llamados episodios, donde cada fragmento describe una condición particular de una parte de la situación en algún intervalo de tiempo. Semejantes fragmentos pueden tratar sobre el cambio de valor cualitativo de alguna variable, la activación de un proceso o la de una vista individual. La diferencia entre episodios y acontecimientos consiste en que los episodios se mantienen durante periodos de tiempo no nulos mientras que los acontecimientos son instantáneos.

La historia es pues una secuencia de acontecimientos y episodios que describe la evolución posible de la situación. Las historias describen el comportamiento en términos de cambios de valores de variables, de comienzo de procesos, de fin de procesos, y de los momentos en que las vistas individuales son validas o dejan de serlo. Una historia predicha no es necesariamente una secuencia lineal de sucesos y episodios puesto que en muchas ocasiones existen varias líneas de comportamiento distintas predecibles.

Mostramos aquí algunos ejemplos de episodios y acontecimientos:

- La temperatura t de un objeto A está aumentando (un episodio de la variable t).
- La instancia hf1 del proceso flujo_de_calor está activa (un episodio de la instancia hf1).
- La posición x del objeto A es 1 (un episodio o un acontecimiento según x sea 1 por un intervalo de tiempo o por un instante).

Puesto que los procesos son la única causa de cambio, el identificar los procesos activos es lo que permite al QPT producir cambios. Si el valor de una variable p está aumentando debido a un proceso activo, se crea el episodio EP1, definido como:

EP1: [dp] = +

Por otro lado, si el valor inicial de p, p0, se sabe que es menor que otro valor p1, y si no hay otro valor significativo entre p0 y p1, el episodio EP1 puede ser seguido por un suceso (o un episodio, según si p sigue cambiando después) en el que p sea igual a p1. Este suceso puede disparar otros procesos o finalizar procesos activos.

Este proceso de mirar hacia qué valores significativos se mueve una variable (o de qué valores se está alejando) se denomina en la QPT análisis de límites. Si un análisis de límites muestra que el valor de una variable se mueve hacia el valor significativo más próximo, se genera un nuevo acontecimiento definido por el momento en el que la variable alcanza ese valor, como continuación del episodio actual.

i.2.3.2.3.4 Espacio de cantidades del QPT

El espacio de valores que puede tomar cada variable se especifica por medio de un número finito de valores significativos y de unas relaciones ordinales entre los mismos. Los valores cualitativos de una variable pueden ser o un valor significativo o un intervalo entre dos valores significativos que no contenga ningún valor significativo entre ambos. Como consecuencia, el número de valores cualitativos que puede tomar una variable es finito.

Un detalle importante es que el conjunto de valores significativos no necesita estar completamente ordenado, por lo que es válida toda ordenación total posible de estos valores que no se contradiga con las relaciones ordinales conocidas. La QPT debe incluirlos todos ellos.

A modo de ejemplo, imaginemos una variable x definida en (-inf,+inf) con tres valores significativos, v0, v1 y v2, tales que v0 > v1 y v0 > v2. En este caso, la relación entre v1 y v2 es desconocida, por lo que hay tres posibles conjuntos de valores cualitativos, uno por cada posible relación entre v1 y v2:

v1 > v2: (-inf,v2),v2,(v2,v1),v1,(v1,v0),v0,(v0,+inf)
v1 = v2: (-inf,v2),v2,(v2,v0),v0,(v0,+inf)
v1 < v2: (-inf,v1),v1,(v1,v2),v2,(v2,v0),v0,(v0,+inf)

El espacio de cantidades de los valores cualitativos de x incluye todos estos posibles valores cualitativos.

He escogido estas tres aproximaciones como las más significativas dentro del campo del razonamiento cualitativo, sin embargo podemos encontrar también [].

i.2.4 Estudio Comparativo de las Técnicas de Razonamiento Cualitativo

En este apartado realizaremos un estudio comparativo de las técnicas de razonamiento cualitativo analizadas anteriormente.

a) **Método de Kuipers:** Desarrollado con posterioridad al método de De Kleer, puede ser considerado como una formalización del mismo que hace especial hincapié en la interpretación cualitativa de las características matemáticas de las funciones que definen el sistema (continuidad, derivabilidad, y puntos singulares, especialmente) pero pierde en el proceso el significado físico y la relación de las mismas con el sistema físico que representan. La mayor inconveniencia que este sistema puede tener consiste en su forma de tratar el sistema como un conjunto, y de considerar las posibles evoluciones temporales del mismo como el desarrollo del producto cartesiano de las posibles evoluciones temporales de cada función componente, que conlleva a problemas de eficiencia computacional muy graves, incluso en sistemas simples.

b) **Método de De Kleer:** El método de De Kleer, aunque supuestamente mejorado por el método de Kuipers, es colocado después del mismo debido a la gran declaratividad de su base de conocimientos. En la misma, los objetos físicos se presentan ordenados en una jerarquía de objetos y clases de objetos, mientras que su conocimiento asociado se expresa a nivel de clase utilizando el concepto de confluencias, desarrollado por De Kleer para mejor representar el conocimiento cualitativo sobre una clase de objetos. Sin embargo, la declarativiidad mostrada a nivel de especificación de la Base de Conocimiento no es compartida por el motor de inferencia. Este es muy similar al de Kuipers, pues de hecho deriva uno del otro, y adolece de las mismas deficiencias computacionales del de Kuipers,, debido a considerar todas las funciones del sistema en el mismo nivel en el motor de inferencia, sin tratar de agruparlas por objetos o afinidades, provocando por tanto una explosión combinatoria en su proceso de razonamiento.

c) **Método de Forbus:** El método de Forbus tiene como principal novedad la incorporación del concepto de Proceso. Este concepto es muy eficaz, pues da una estructura jerárquica al conocimiento sobre los cambios de un sistema físico, de la

que los modelos anteriores carecen, sin embargo, pierde las ventajas de la jerarquía de componentes de De Kleer, al mezclar conocimiento de diferentes clases de objetos involucrados bajo un mismo proceso.

Como conclusión, cabe decir que todas estas aproximaciones al razonamiento cualitativo tienen en común el hecho de que sólo se han aplicado a problemas juguete; es decir, a sistemas físicos con un número pequeño de componentes y de bucles realimentados. Los intentos para modelar sistemas complejos con paradigmas cualitativos chocan con las ineficiencias señaladas en los modelos anteriores, y que podríamos resumir en la siguiente lista:

- La representación temporal utilizada no es adecuada; el intento de imponer una ordenación total a componentes que no tienen relación entre sí, introduce una explosión de posibilidades para cubrir la indeterminación asociada que va contra la eficiencia del sistema.
- La no representación de parámetros multidimensionales en estos paradigmas de razonamiento cualitativo, obliga a discretizar aquellas funciones de naturaleza multidimensional que podamos encontrar, como conjuntos de funciones unidimensionales, de forma completamente innecesaria y además gravosa en tiempo de ejecución para el sistema (como ejemplo podemos considerar la modelización de una calle, que evidentemente es bidimensional, pues los parámetros de tráfico varían en el tiempo y en el espacio, pero cuya consideración en los paradigmas anteriores obligaría a dividir la calle en secciones, con objeto de representar los parámetros de tráfico en cada sección y eliminar así la dimensión espacial al problema). Esta discretización, como vemos, se debe a las limitaciones de los paradigmas considerados y no a una característica del sistema.

De acuerdo con lo anterior, se justifica que uno de los objetivos (ver sección siguiente) del presente trabajo sea la elaboración de un formalismo de razonamiento cualitativo que trate de eliminar parte de estas limitaciones, mediante la incorporación de parámetros multidimensionales, y agregue además las mejores características de cada modelo (el tratamiento cualitativo de funciones temporales de Kuipers, la jerarquía de objetos de De Kleer y la jerarquía de procesos de Forbus).

Objetivos

Las limitaciones de los métodos actuales de razonamiento cualitativo, analizadas en la Introducción, para la representación de sistemas físicos con parámetros multidimensionales, nos han conducido a desarrollar un formalismo de razonamiento cualitativo que incorpore, junto con las características básicas de los métodos estudiados en la introducción, un soporte adecuado para la representación y razonamiento con parámetros multidimensionales. Posteriormente, este formalismo es aplicado a dos dominios distintos de aplicación que incorporan tales tipos de parámetros, con objeto de mostrar la claridad alcanzada en la representación de su conocimiento y las ventajas computacionales que se obtienen con respecto a otros métodos. Los dominios de aplicación escogidos son dos: un sistema con flujos discretos, el Control del Tráfico Urbano, y otro con flujos contínuos, el Problema del Sistema de Cubas, sacado de la literatura.

En consecuencia, la presente memoria pretende abordar los siguientes objetivos:

- Estudio de los actuales métodos de razonamiento cualitativo, evaluando sus limitaciones en lo que respecta a su aplicación a sistemas físicos con parámetros de difícil representación.

- Definición de un formalismo de representación del conocimiento que permita modelizar el conocimiento de sistemas con dinámica compleja, con el objetivo de superar las limitaciones encontradas en los estudios anteriores.

- Construcción de un motor de inferencia que trabaje sobre el conocimiento expresado con el formalismo anterior, y que permita razonar cualitativamente en un cierto nivel de agregación, sobre la dinámica de los sistemas.

- Discusión de la utilidad del modelo construído con el formalismo y el motor de inferencia anteriormente citados, dentro de la arquitectura funcional de un sistema de control.

- Aplicación a la modelización del tráfico urbano y del problema del sistema del sistema de cubas obteniendo el razonamiento sobre el futuro cercano, discutiendo las ventajas y desventajas respecto a la utilización de otras técnicas de razonamiento cualitativo.

- Estudio de los requerimientos de implementación del formalismo y motor de inferencia en un lenguaje CLP, que aporte un gran nivel de declarativid.

- Generalización de la implementación del formalismo y motor de inferencia para hacerlo independiente del dominio, y mostrar su potencial generalidad.

Capítulo I

ARQUITECTURA FUNCIONAL DE UN SISTEMA DE CONTROL BASADO EN LA UTILIZACION DE UN MODELO CUALITATIVO

I.1 INTRODUCCION

En el presente capítulo vamos a proponer una arquitectura funcional para un sistema de control basado en la utilización de un modelo cualitativo. La arquitectura propuesta ha sido desarrollada con objeto de obtener un sistema de control que razone de forma cualitativa sobre una representación multidimensional del sistema objeto del control. El objetivo de este capítulo es mostrar la utilidad de disponer de un modelo del sistema y a partir de sus principios generales tratar de extenderla sin grandes dificultades a cualquier otro dominio de aplicación donde la necesidad de considerar parámetros multidimensionales juegue un papel primordial.

Un Sistema de Control de Procesos [Repsol,92] puede describirse como un sistema jerárquico de distintos niveles, que van desde el nivel más bajo, compuesto por la capa de sensores y efectores del sistema, hasta el más elevado, donde aparece el control inteligente del proceso, pasando por una serie de niveles de sofisticación creciente, normalmente automatizables. El requerimiento básico que se le pide a un sistema de control de procesos es que sea capaz de actuar en *tiempo real*, lo que obliga a considerar diferentes requerimientos sobre la velocidad del sistema de control, dependiendo de las características temporales del proceso y del nivel de monitorización seleccionado. La velocidad del sistema decrece según aumenta el nivel de control, de forma que el control inteligente no suele precisar una respuesta

rápida, y normalmente es realizado por un operador humano; mientras que los niveles inferiores requieren una mayor velocidad.

Lukas [Lukas,89] ofrece una enumeración de las áreas del control de procesos que presentan características tales que sugieren su abordamiento mediante técnicas de inteligencia artificial; entre las que podemos destacar los sistemas de diagnóstico [Chandrasekaran,87], [Salmerón,92]. Estos sistemas suelen disponer de un modelo del mundo real sobre el que trabajan [De Kleer,87], y son una muestra de la importancia de la utilización de modelos en los sistemas expertos en control de procesos.

Además de la conveniencia de incorporar un modelo, Laffey [Laffey,88] propone unos requerimientos que son deseables en los Sistemas Expertos para el Control de Procesos en tiempo real, entre los que destacaremos la necesidad de focalización de la atención y de respuesta en un tiempo prefijado.

Por otro lado, diversos autores [Leitch,91], [Cuena, 91], [Cain, 91] han identificado la necesidad de considerar la utilización de modelos del mundo real en diferentes niveles de detalle según la aplicación a la que estén destinados, lo que redunda en claridad y eficiencia, pero implica una consideración del mundo real en diferentes niveles de agregación.

Pasaremos seguidamente a enumerar las ventajas que la incorporación de un modelo cualitativo del mundo real puede traer a un sistema de control, obtenidas a partir de un estudio de los trabajos realizados por los autores anteriormente citados:

En primer lugar, puede ser utilizado para la predicción del comportamiento del mundo real bajo el plan de control vigente, con lo que nos ofrece la posibilidad de anticiparnos a los efectos de posibles errores en la elaboración del plan de control por el sistema. Esta capacidad aumenta la calidad del control ofrecido por el sistema, puesto que, en principio, nos permite:

a) Prevenir muchas situaciones problemáticas provocadas que puedan darse como consecuencia de medidas de control inadecuadas antes de que aquellas lleguen a producirse.

b) Aumentar la velocidad del sistema de control, al descargarlo del esfuerzo computacional de detectar y hallar remedio a estas situaciones problemáticas, que siempre es mayor que el esfuerzo de prevenirlas.

Estas consideraciones nos sugieren incorporar un **módulo de detección de problemas futuros por razonamiento basado en modelo**, como parte integrante de un sistema de control. Este proceso puede ser caracterizado como razonamiento profundo del sistema de control sobre el mundo real, con lo que entramos dentro del campo de los Sistemas Expertos de Segunda Generación.

En segundo lugar, la utilización de un modelo puede ayudar al sistema de control a discernir de forma precoz entre las situaciones problemáticas causadas por una mala regulación y accidentes imprevisibles que requieren una rápida actuación, ayudando con ello a acelerar la respuesta del sistema frente a situaciones inesperadas. En efecto, la utilización de un modelo nos permite modelizar el comportamiento presente del mundo real a partir de datos pasados, con lo que los incidentes imprevisibles son fácilmente reconocidos como bruscas desviaciones del comportamiento del mundo real con respecto al modelo; mientras que las situaciones problemáticas debidas a un mal control no suponen desviación alguna, puesto que pueden ser modelizadas. Esta capacidad podría ser desarrollada por un sistema de control mediante la inclusión en su estructura de lo que llamaremos un **módulo de identificación de causas de problemas a través de diagnósticos basados en el modelo.**

En tercer lugar, el sistema de control, frente a la identificación de un conjunto de posibles soluciones a las situaciones problemáticas planteadas, la utilización de un modelo, permite discernir entre las mismas, permitiendo razonar sobre la bondad de los futuros correpondientes a cada posible solución. Este sistema puede funcionar durante un tiempo prefijado [Sibille,87] mediante su inclusión en un ciclo que pueda ser interrumpido por una señal horaria, momento en el que se aplicaría la mejor solución encontrada hasta entonces. La bondad de la solución obtenida vendría entonces dada por el tiempo del que se ha dispuesto para calcularla. Este ciclo podría ser realizado por un conjunto de **módulos de resolución** que incorporen un modelo del mundo real junto con un sistema capaz de proponer soluciones a las situaciones problemáticas planteadas.

En cuarto lugar, la descripción de la estructura del mundo real que un modelo ofrece puede servir de base a un sistema de focalización eficaz, el cual determinaría

rápidamente la región del mundo real que va a ser objeto de un análisis más detallado para encontrar soluciones a las situaciones problemáticas planteadas. Esta función puede ser realizada por un **módulo de identificación de zonas problema** basado en la estructura del sistema y un modelo más simplista del comportamiento del mundo real, con un nivel de detalle inferior a los otros modelos utilizados hasta ahora.

Considerando las diferentes aplicaciones que un modelo cualitativo puede tener en el diseño de un sistema de control y las múltiples ventajas que es capaz de aportar al mismo, pensamos que sería deseable diseñar una arquitectura para un sistema de control que estuviera basada en la utilización de un modelo cualitativo. Con este objetivo, proponemos una arquitectura funcional de un sistema de control basada en la utilización de un modelo cualitativo que incorpora las funcionalidades que acabamos de describir, con objeto de lograr tanto una mejor calidad del control realizado por el sistema como una mayor rapidez en la respuesta del mismo frente a potenciales problemas críticos, en las siguientes secciones desarrollamos en profundidad los diferentes módulos que compondrían una tal arquitectura.

I.2 ANALISIS DE LA ARQUITECTURA PROPUESTA

La arquitectura de control propuesta corresponde a un sistema experto de segunda generación, debido a que es un sistema de control basado en el conocimiento, que hace uso de un modelo del sistema para razonar sobre el mismo. Este modelo del sistema se representa en la base de conocimiento mediante el formalismo (λ,t) que describiremos en el capítulo II, y se utiliza a través de los predicados de consulta que ofrece el formalismo.

Siguiendo una descripción de carácter tradicional, vamos a describir la interacción entre el sistema y el mundo real objeto de nuestro discurso en los siguientes componentes:

1) Sensorización
2) Preprocesamiento
3) Detección
4) Diagnóstico/Resolución
5) Refinamiento
6) Acciones sobre el Mundo Real

Puesto que las fases 1 y 6 dependen de las características específicas de cada sistema objeto del control, aquí nos vamos a centrar en las fases 2,3,4 y 5, que definen la estructura cognitiva y de razonamiento que nos interesan en este capítulo, con independencia de las aplicaciones concretas a las que nos referiremos en capítulos posteriores.

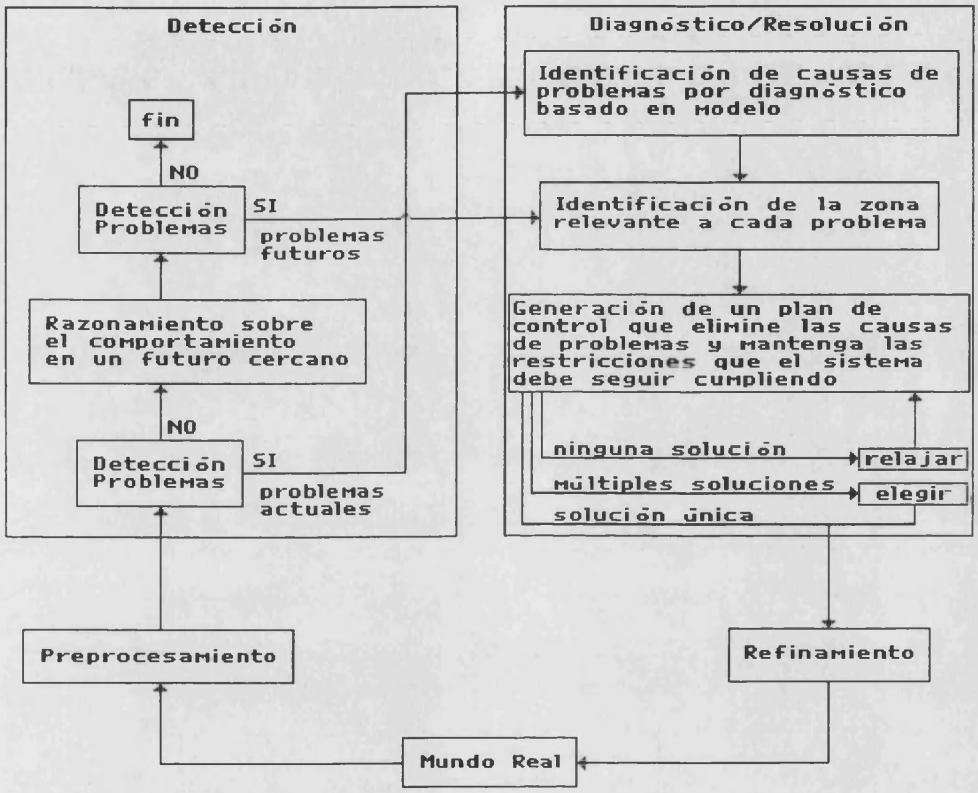

Figura 1.1

Comentaremos seguidamente la estructura de dicha arquitectura de control, mostrada en la figura 1.1, para analizar en las secciones siguientes cada uno de los módulos que la componen.

Primeramente, los datos del mundo real entran en el sistema de control y son sometidos a una etapa de **preprocesamiento**, donde se filtran los datos recibidos de los sensores con objeto de extraer la información realmente útil para el sistema de control. Un proceso importante que se lleva a cabo en esta etapa es el de convertir los valores cuantitativos de entrada en valores cualitativos que puedan ser utilizados por el sistema.

En segundo lugar, los datos preprocesados se ven sometidos a un proceso de **detección de situaciones problemáticas** por reconocimiento de patrones. Si se detecta la existencia de situaciones problemáticas en el momento actual, se pasa la ejecución al módulo de **diagnóstico y resolución** para que trate de solucionarlas; si no se detecta la existencia de situaciones problemáticas en el momento actual, se intenta averiguar si la estrategia actual de control puede producir problemas en un futuro cercano, para lo cual se realiza un proceso de **razonamiento sobre el comportamiento futuro del sistema**, utilizando para ello el formalismo (λ,t), y los datos obtenidos se someten nuevamente a un proceso de **detección de situaciones problemáticas**. Si se detecta la existencia de situaciones problemáticas en el comportamiento futuro del sistema, se pasa la ejecución al módulo de **diagnóstico y resolución**, para que tome las acciones preventivas necesarias, tal y como muestra la figura 1.1; en caso contrario, el sistema deduce que no existe motivo alguno de alarma, ni en el momento presente ni en un futuro cercano, por lo que espera a la llegada de más datos desde el mundo real.

En tercer lugar, los informes sobre situaciones problemáticas actuales o previstas entran en el módulo de **diagnóstico y resolución**. Los informes sobre situaciones problemáticas entran en un módulo de **identificación de causas de problemas por razonamiento basado en modelo**, donde utilizamos de nuevo el formalismo (λ,t) con objeto de averiguar si el problema detectado se debe a un incidente no previsible o a una mala regulación del sistema; de este módulo se pasa, siempre según la figura 1.1, a un módulo de **identificación de la zona relevante a cada problema**, donde se decide qué subconjunto del mundo real consideramos para encontrar solución a las situaciones problemáticas consideradas. Es de notar que las situaciones problemáticas detectadas en el comportamiento futuro del sistema (ver apartado anterior) pasan directamente a este módulo sin pasar por el anterior, debido a que ya sabemos que únicamente pueden deberse a una mala regulación del sistema (puesto que si se debieran a incidentes no previsibles, no habríamos podido preveerlos). Finalmente pasamos al **módulo de generación de soluciones**, que permite generar soluciones a un sistema de restricciones temporales que representa todos los requisitos de control que deben cumplirse en el subconjunto del mundo real considerado, más las causas de problemas que deben ser evitadas. Si el generador de soluciones no genera ninguna solución, se relajan las restricciones hasta que lo haga; si genera excesivas soluciones, se escoje la mejor, para lo que también se utiliza el formalismo (λ,t), y si genera una única solución, se pasa directamente al módulo siguiente.

En cuarto lugar, la solución obtenida en el módulo de **diagnóstico y resolución**, pasa al módulo de **refinamiento**, donde se realiza un proceso de refinamiento de la solución obtenida, en el que se trata de optimizar el funcionamiento de las partes del sistema vecinas a aquellas afectadas por el nuevo plan de control, ajustándolas lo mejor posible a las nuevas condiciones. Es importante destacar que el plan de control final no es tan sólo el nuevo estado de los efectores para el momento siguiente, sino toda una evolución temporal de los mismos a lo largo de los próximos minutos. Esto es necesario pues muchos incidentes no pueden ser solucionados por cambios inmediatos de los efectores, sino mediante una secuencia de cambios que se desarrolla a lo largo de varios minutos y que debe considerarse como un todo a efectos de razonamiento de control.

I.3 MODULO DE PREPROCESAMIENTO

El módulo de preprocesamiento tiene por objeto extraer a los sensores la información de utilidad para el sistema de control que éstos puedan proporcionar. Para ello, se realiza en primer lugar un filtrado de los valores cuantitativos que el sensor registra, y en segundo lugar se convierten los valores cuantitativos filtrados en valores cualitativos que puedan ser utilizados por el sistema. En este punto conviene distinguir tres tipos de sensores, según el preprocesado que cada uno necesita:

-1) El primer tipo estaría formado por aquellos sensores que se ajustan a la descripción que acabamos de dar, como por ejemplo la medida de la densidad de tráfico en una entrada a una ciudad, que se somete a un filtrado y a una transformación directa sobre una escala cualitativa de valores.

-2) El segundo tipo lo formarían aquellos sensores que se utilizan para medir parámetros multidimensionales sin ser ellos mismos de naturaleza multidimensional, como por ejemplo el utilizar un sensor de bucle (sensor puntual) para registrar el estado de una calle (función bidimensional); estos sensores requieren una etapa posterior de preprocesado en la que se generan hipótesis sobre la evolución del parámetro multidimensional que se pretende medir que sean compatibles con las medidas puntuales obtenidas del sensor, para lo cual se utiliza el formalismo (λ,t).

-3) El tercer tipo lo formarían aquellos sensores cuya naturaleza impida transformarlos en valores cualitativos y deban seguir siendo considerados valores cuantitativos, como por ejemplo la medida del tiempo restante de verde en un semáforo.

I.4 MODULO DE DETECCION

I.4.1 Módulo de detección de situaciones problemáticas actuales

El módulo de detección de situaciones problemáticas actuales, que puede verse en la figura 1.1 como el elemento que recibe datos del preprocesador, tiene por objeto la detección de situaciones problemáticas en el momento presente mediante un proceso de reconocimiento de patrones. Este proceso utiliza un formalismo presentado en el capítulo IV para representación de patrones situaciones problemáticas en diferentes niveles de abstracción espacial y temporal, llamado Abstractor de Acontecimientos. Las situaciones problemáticas que este módulo es capaz de detectar pueden dividirse en dos clases: incidentes y problemas detectables por el sistema. Los incidentes son aquellos problemas de carácter no predecible y que únicamente pueden ser reconocidos como desviaciones del comportamiento del sistema con respecto al modelo, y los problemas detectables por el sistema son problemas debidos a una mala regulación y que, por tanto, pueden ser predichos por el modelo. Este módulo se encarga de detectar ambos tipos de problema, sin discernirlos, y transmite esta información al módulo de **identificación de causas de problemas por razonamiento basado en modelo**, sito en el módulo de **diagnóstico y resolución**, donde se disciernen ambos tipos de situaciones problemáticas merced a la utilización del modelo.

I.4.2 Módulo de razonamiento sobre el comportamiento futuro

El módulo de razonamiento sobre el comportamiento futuro del sistema se activa únicamente si no existen situaciones problemáticas a considerar en el momento presente, y tiene por objeto predecir el comportamiento del sistema en un futuro cercano según la actual estrategia de control, como paso previo necesario para averiguar si la estrategia actual de control puede producir problemas detectables por el sistema en el futuro cercano. Para realizar el proceso de razonamiento sobre el comportamiento futuro del sistema, se utiliza el formalismo (λ,t). El capítulo VI muestra una implementación del motor de inferencia del formalismo, que es capaz de realizar esta tarea. Como datos base para el motor de inferencia se utilizan los datos actuales como estado inicial, y la estrategia de control como evolución posterior de los elementos que afectan al sistema (dominio exterior, según la terminología definida en el capítulo II).

I.4.3 Módulo de detección de situaciones problemáticas futuras

El módulo de detección de situaciones problemáticas futuras es un subconjunto del módulo de detección de situaciones problemáticas actuales, que tratamos aquí por separado por razones de claridad expositiva. Este módulo tiene por objeto la detección de situaciones problemáticas en el futuro cercano, actuando para ello sobre la Base de Datos Temporal que describe el comportamiento previsto del sistema en un futuro cercano según el resultado del módulo anterior. El reconocimiento de las situaciones problemáticas se realiza utilizando el Abstractor de Acontecimientos, pero únicamente es posible reconocer un tipo de situaciones problemáticas, los problemas detectables por el sistema, razón por la que anteriormente afirmamos que este módulo es tan sólo un subconjunto del módulo de detección de situaciones problemáticas actuales, descrito en I.3.1. En efecto, no podemos reconocer situaciones problemáticas de tipo incidente porque éstas se obtienen de la divergencia entre los datos reales y la evolución del sistema prevista por el modelo. Puesto que en el futuro cercano todavía no disponemos de datos reales con los que comparar el comportamiento previsto, nos vemos limitados, por tanto, a la consideración de los problemas detectables por el sistema. Este tipo de problemas, pasan directamente al módulo de **identificación de zonas problema** sin pasar por el módulo de **identificación de causas de problemas por razonamiento basado en modelo**, puesto que ya sabemos que la causa de los mismos es una mala regulación.

I.5 MODULO DE DIAGNOSTICO Y MODULO DE RESOLUCION

I.5.1 Módulo de identificación de causas de problemas por razonamiento basado en modelo

El módulo de identificación de causas de situaciones problemáticas por razonamiento basado en modelos, o más brevemente módulo de diagnóstico, se utiliza únicamente en el caso de que se hayan detectado situaciones problemáticas en el momento actual. Su misión consiste en discriminar entre las dos clase posibles de situaciones problemáticas: incidentes y problemas detectables por el sistema, tarea ésta que realiza merced a la utilización del formalismo (λ,t). Los incidentes, causados por sucesos imprevisibles, son reconocidos como desviaciones del comportamiento del sistema con respecto al modelo, y los problemas detectables por el sistema, causados por una mala regulación, son reconocidos como consecuencias del plan de control vigente, según el modelo. Pasemos a analizar ambos tipos de situaciones problemáticas con más profundidad:

Los **incidentes** son situaciones problemáticas que no pueden ser previstas por el modelo del sistema (como por ejemplo un accidente de tráfico en una red urbana). Los incidentes se reconocen procediendo de la siguiente forma: en primer lugar, se realiza un razonamiento sobre el comportamiento futuro del sistema basado en datos del pasado cercano, utilizando para ello el formalismo (λ,t), con el objetivo de obtener el estado de los sensores en el momento presente. Si el estado actual de los sensores coincide prácticamente con el estado predicho por el modelo, entonces el estado presente no ha sufrido desviación apreciable con respecto al modelo del sistema, por lo que no puede haber ninguna situación problemática de tipo incidente. Si, por el contrario, existen diferencias importantes entre ambos estados, se realiza un proceso de reconocimiento de patrones sobre las diferencias para identificar las situaciones problemáticas de tipo incidente que las puedan causar, utilizando para ello el Abstractor de Acontecimientos. En el caso de que hayan diferencias importantes entre ambos estados y, sin embargo, el Abstractor de Acontecimientos no reconozca ningún incidente, debemos entender que las diferencias observadas no constituyen ningún incidente a solucionar por el sistema de control.

Los **problemas detectables por el sistema** son situaciones problemáticas que pueden ser previstas por el modelo del sistema (como por ejemplo, el preveer que una situación de congestión en una avenida acabará afectando a todas las calles afluentes), y generalmente indican una falta de adecuación de la estrategia de control actual a la evolución actual del mundo real. Estos problemas se reconocen por una doble vía: por reconocimiento de patrones sobre los datos actuales y por reconocimiento de patrones sobre datos previstos, como explicamos en las secciones I.3.1 y I.3.3. Sin embargo, es conveniente señalar que este tipo de problemas no se reconoce propiamente sobre los datos reales, sino que más bien se vigilan, puesto que el sistema de control ya tiene conocimiento de la existencia del problema, que pudo preveer tiempo atrás como consecuencia de la estrategia de control que estaba siendo aplicada. Queda tan sólo considerar las razones por las que un problema de este tipo puede existir, puesto que se podría pensar que si el problema pudo ser predicho, debiera haber sido evitado. Existen dos razones que impidan proceder de esta manera: la primera consiste en que no exista solución inmediata al problema, por lo que debemos solucionarlo a lo largo de un intervalo de tiempo considerable, durante el cual el problema será detectado de nuevo con cada nueva iteración del bucle de control hasta que se solucione, y la segunda consiste en aquellos problemas que nosotros causamos deliberadamente con objeto de solucionar otros problemas de mayor índole. En efecto, toda acción de control tomada para solucionar un problema

es susceptible de crear otro problema secundario, y en el caso (bastante común en Control de Tráfico Urbano) de que no exista ninguna secuencia concebible de acciones de control que puedan solucionar un problema dado sin producir otro secundario, nos vemos obligados a hacerlo. En este caso tendremos deliberadamente un problema predecible que no debemos solucionar, puesto que hacerlo antes de haber solucionado el problema original no conduciría a ningún sitio, pero que sin embargo será detectado en cada nueva iteración del bucle de control, y que debemos ignorar. Todas estas consideraciones se hacen en el módulo de generación de planes de control.

I.5.2 Módulo de identificación de zonas problema

El módulo de identificación de zonas relevantes a las situaciones problemáticas enviadas desde el módulo de detección para su análisis y consideración en la elaboración de una nueva estrategia de control, tiene por objeto identificar un área de estudio para cada situación problemática a considerar, de tal manera que la solución a la misma se encuentre a través de la modificación de la estrategia de control vigente en el área. Podemos considerar este módulo como un proceso de focalización del sistema de control sobre las situaciones problemáticas consideradas, con objeto tanto de aumentar la eficiencia computacional, como de facilitar el diseño del sistema, al proveer de una aproximación modular al estudio de situaciones problemáticas.

I.5.3 Módulo de generación de planes de control

El módulo de generación de planes de control tiene por objeto encontrar planes de control que eviten las acciones de control causantes de problemas que se dedujeron en el módulo anterior, sin caer por ello en inconsistencias de las acciones de control que puedan ser causa de nuevos problemas. La estructura de este módulo cae fuera de los objetivos de esta tesis, y aquí lo consideraremos como una especie de caja negra cuya labor consiste en encontrar soluciones a un conjunto de restricciones de control. Baste decir que este módulo se está desarrollando en el proyecto EQUATOR [EQUATOR,93] como parte de la arquitectura general que este proyecto propone para sistemas de control de procesos que hagan uso de elementos cualitativos y temporales.

Este módulo incluye todas aquellas reglas de control cuyo incumplimiento pueda ser causante de problemas en forma de ligaduras de obligado cumplimiento, e incluye

como inconsistencias a evitar todas las acciones de control que, según los resultados del módulo anterior, deban ser evitadas. El módulo trata de resolver el sistema de restricciones así planteado sobre las acciones de control permitidas, llegando a tres posibles resultados que consideraremos aquí:

-1) Solución única: El sistema tiene una única solución, que es directamente el plan de control adecuado a la situación considerada. En este caso se pasa el nuevo plan de control al módulo de refinamiento. Un detalle importante sobre el plan de control que conviene no olvidar es que el plan de control no está únicamente compuesto del estado de los efectores en el siguiente momento del tiempo, sino que se trata de una evolución temporal a lo largo de un tiempo más o menos largo. Recordemos que ciertos problemas sólo pueden resolverse creando un problema secundario en el proceso que se resuelve más tarde. Este tipo de procesos requieren especificar un plan de acción a lo largo de un tiempo largo, o de lo contrario se perdería la noción de que el problema secundario es deliberado, y se trataría de resolver de forma continua, anulando así la eficacia de este método de resolución de problemas, muy común en Control de Tráfico Urbano, por ejemplo.

-2) Ninguna solución: El sistema no es capaz de dar ningún plan de control capaz de solucionar todas las situaciones problemáticas consideradas, por lo que habrá que relajar las restricciones hasta que aparezca una solución. Nótese que relajar las restricciones significa causar un problema secundario de forma deliberada para poder encontrar solución a los problemas más acuciantes. Este problema secundario se resolverá cuando los problemas principales lo hayan hecho, lo que notaremos porque el sistema de mantenimiento de la verdad temporal ya es capaz de dar soluciones al sistema. Este proceso puede verse en la figura 1.1 bajo el nombre de **relajar**.

-3) Múltiples soluciones: El sistema encuentra varios planes de control que son capaces de dar solución a las situaciones problemáticas consideradas, por lo que se presenta el problema de escojer una. Para ello se utiliza el formalismo (λ,t) con objeto de averiguar el comportamiento futuro del sistema bajo cada plan de control, y se aplica el Abstractor de Acontecimientos con objeto de calibrar la bondad de cada una de las soluciones hasta encontrar la mejor, o hasta que las necesidades de control en tiempo real del sistema nos obligan a dar una respuesta, en cuyo caso escojemos el mejor plan de control que se haya encontrado hasta el momento. En cualquiera de ambos casos, el plan de control seleccionado se envía al módulo de refinamiento, último elemento de la arquitectura de control. Este proceso queda reflejado en la figura 1.1 bajo el nombre de **elegir**.

I.6 MODULO DE REFINAMIENTO

El módulo de refinamiento tiene por objeto optimizar el funcionamiento de la parte del sistema no afectada directamente por el plan de control propuesto por el módulo de diagnóstico y resolución. En efecto, la introducción de un cambio en el plan de control para solucionar las situaciones problemáticas consideradas hará que muchos parámetros de control cuyos valores no podían mejorarse bajo el antiguo plan de control, ya puedan ser mejorados, proceso éste que lleva a cabo el módulo tratado.

La razón que hace posible que existan parámetros de control que pueden ser mejorados tras cada cambio del plan de control vigente, tenemos que buscarla en el hecho de que un sistema no puede ser nunca controlado de forma localmente óptima, debido a que el óptimo de cualquier parámetro suele ser inconsistente con el óptimo de sus vecinos. Por ello, el control óptimo de un sistema tiene carácter global, y se da por un constante tira y afloja de muchos parámetros, por lo que el cambiar el valor de algunos parámetros de control viene seguido, casi inevitablemente, por la relajación de la presión que éstos ejercían sobre algunos de sus parámetros vecinos, mientras recrudece la presión sobre otros. El módulo de refinamiento detecta estos parámetros cuyas restricciones han quedado relajadas y los optimiza localmente en lo posible sin violar ninguna restricción que los ligue a sus vecinos.

I.7 CONCLUSIONES

En la Arquitectura funcional presentada en el presente capítulo se ha podido identificar un potencial uso de un modelo (en nuestro caso construído con el formalismo (λ,t), que se presentará en el próximo capítulo) del sistema a controlar en diversos módulos y con los diferentes objetivos que nos habíamos planteado en el principio del capítulo:

- Diagnóstico.
- Evolución futura, para detección de problemas futuros.
- Elección de una de las varias propuestas de control posibles.
- Refinamiento.

Como puede observarse, la utilización del modelo puede ser intensiva, por lo que un requerimiento importante es el hecho de que pueda obtenerse una implementación eficiente del mismo. Esta eficiencia es un punto clave para dar por buena la

arquitectura ahora propuesta, y será por tanto uno de los hilos conductores de los capítulos que siguen. La validación de la bondad de la aproximación esquematizada en la figura 1.1 vendrá dada por las implementaciones llevadas a cabo (ver capítulo VI y anexos A, B y D) y que nos han justificado la racionalidad de la arquitectura que acabamos de proponer y que está en línea con las desarrolladas en la literatura por los autores referenciados.

Capítulo II

EL FORMALISMO (λ,T)

II.1 INTRODUCCION

A principios de los ochenta empezaron a confirmarse las limitaciones de los paradigmas de razonamiento existentes para las tareas de control y monitorización de sistemas dinámicos (ver anexo G). La razón de estas carencias hay que buscarlas en la falta de conocimiento específico que estas aproximaciones incorporan sobre el sistema físico objeto del control, que constituye el mundo real sobre el que funciona el sistema. Estas limitaciones del modelo para que refleje el conocimiento sobre el mundo real obliga a expresar el conocimiento del dominio de forma superficial, fragmentaria e incompleta, que aparece mezclado con el conocimiento propio de la función de control. Además presenta el inconveniente de su no reusabilidad, ya que habitualmente debe ser reescrito en su casi totalidad al alterar ó expandir el conocimiento de control.

Con el objetivo de superar estos inconvenientes, varios autores trataron de formalizar el conocimiento sobre el mundo real para hacerlo independiente del resto del conocimiento. La representación de este tipo de conocimiento y las técnicas de razonamiento asociado se inscribe en lo que llamamos Razonamiento Cualitativo [Shapiro,92]. En esta línea hay que situar los trabajos de Kuipers, De Kleer y Forbus, ya comentados en la Introducción.

En todos estos trabajos pueden encontrarse características comunes, tales como:

- 1) El mundo real se suele descomponer en función de objetos simples de comportamiento conocido, lo que hace posible analizar sistemas complejos con relativa sencillez a partir de sus componentes.

- 2) Cada objeto del sistema tiene asociados unos parámetros cuyos valores en cada momento determinan el estado del objeto. Consecuentemente, la integración de los estados de todos los objetos que componen un cierto sistema, acaban determinando su estado.

- 3) Los parámetros utilizados son funciones del tiempo que toman valor en un espacio continuo, por lo que pueden ser representados como funciones del tipo $F_i(t)$.

Al tener que trasladar los parámetros a una representación cualitativa, debe definirse un espacio cuántico que tenga en cuenta el significado de cada parámetro dentro del objeto (el espacio cuántico se intenta construir de manera que cada uno de sus valores corresponda a un comportamiento cualitativamente distinto del objeto). Por ejemplo, si el objeto es el agua contenida en un cazo puesto a calentar y el parámetro es la temperatura, un espacio cuántico adecuado puede ser el siguiente:

Estado cualitativo	**Significado**
A	Temperatura menor que 0^oC
B	Temperatura 0^oC
C	Temperatura entre 0^o y 100^oC
D	Temperatura 100^oC
E	Temperatura mayor que 100^oC

Tabla 2.1

El espacio cuántico así definido, QS = {A,B,C,D,E}, representa de forma cualitativa todos los posibles valores que el parámetro temperatura puede tomar. Nótese además que cada uno de los valores cualitativos del espacio cuántico representa un estado cualitativamente distinto del objeto (agua en el ejemplo de la tabla 2.1).

Cuando se representa un parámetro a través de un espacio cuántico, la evolución temporal del parámetro puede ser representada como un conjunto de intervalos temporales. En cada uno de estos intervalos el valor cualitativo del parámetro es constante, como puede verse en la figura 2.2.

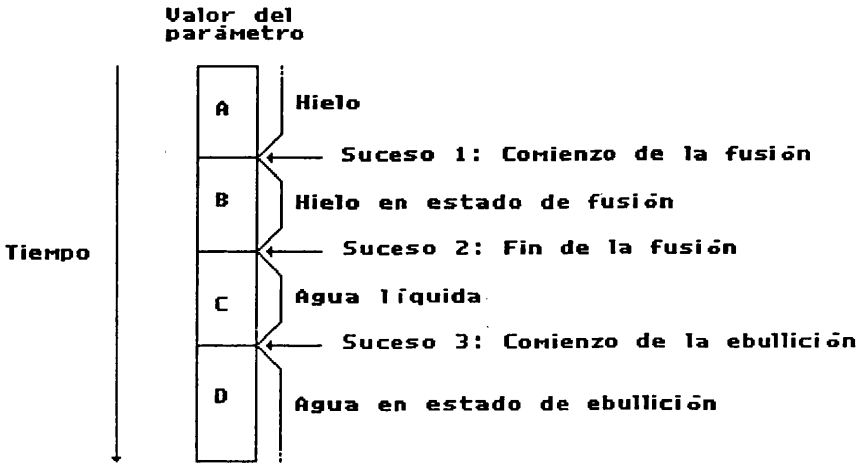

Figura 2.2

- 4) La evolución del sistema se obtiene, en general, en forma de un grafo dirigido, cuyos nodos son estados del sistema y cuyos arcos son cambios de estado. Cada estado viene definido por un conjunto de valores para los parámetros del sistema, y cada cambio de estado corresponde al cambio de valor cualitativo de uno o más parámetros del sistema. Existen dos clases posibles de evolución del sistema según el número de estados siguientes y posibles a uno dado: tiempo lineal y tiempo ramificado. Hablamos de tiempo ramificado cuando existe incertidumbre en la determinación del estado siguiente a uno dado, y como consecuencia se obtienen varios estados siguientes como posibles; en este caso la evolución real del sistema podría seguir por cualquiera de estos estados siguientes, correspondientes cada uno a un futuro distinto para el sistema. El grafo de evolución del sistema toma entonces el aspecto de árbol, correspondiendo cada rama a un futuro distinto. En contraposición, diremos que estamos ante un caso de tiempo lineal si el estado siguiente a uno dado es único y el grafo de evolución del sistema toma el aspecto de una cadena de estados.

- 5) La evolución temporal del sistema se calcula a partir de las restricciones existentes entre los parámetros y entre sus derivadas temporales[1]. Estas restricciones pueden ser aisladas (Kuipers, De Kleer) o pueden estar agrupadas en un conjunto según el proceso físico al que correspondan (Forbus).

Los paradigmas de razonamiento cualitativo de Kuipers, De Kleer y Forbus son capaces de:

[1] Esto es válido si no consideramos como un comportamiento posible la creación y destrucción de objetos físicos. En este caso tendríamos que considerar la creación y destrucción de los parámetros, además de su comportamiento, para calcular la evolución de un sistema.

-a) Utilizar las restricciones y/o procesos que controlan el sistema para predecir su comportamiento.

-b) Detectar un comportamiento anómalo del sistema y analizar sus posibles causas. Esta capacidad es necesaria para la detección de incidentes que perturban el funcionamiento normal del sistema pero que no son observables directamente.

-c) Explicar el comportamiento del mundo real, identificando:

c.1.El estado cualitativo en el que cada objeto se encuentra (aunque este estado no sea observable por ningún sensor de entre los disponibles por el operador).

c.2.Las restricciones y/o los procesos que definen el comportamiento físico del sistema.

A pesar de las sensibles mejoras epistemológicas que suponen estas aproximaciones, una revisión de la literatura, nos permite afirmar que no parece que exista ningún paradigma de razonamiento cualitativo que sea capaz de realizar una simulación cualitativa de un sistema físico que contenga parámetros del tipo $F_i(\lambda,t)$. Esto es, parámetros que sean función del tiempo y de otra(s) variable(s) continua(s). La dificultad estriba básicamente en que mientras un parámetro del tipo $F_i(t)$ es una función de un parámetro que puede representarse dividiendo el eje temporal en intervalos, un parámetro $F_i(\lambda,t)$ debe representarse dividiendo el plano (o variedad lineal) de coordenadas (λ,t) en regiones poligonales (o n-dimensionales, puesto que F puede ser función de varios λ_i además del tiempo, adoptando en general la forma: $F(\lambda_1,...,\lambda_{n-1},t))$, lo que hace su tratamiento más complejo.

El objetivo a perseguir, sería estudiar la evolución del sistema, teniendo en cuenta sólo los momentos representativos en los que el "comportamiento" del sistema sufre variación. Como este estudio de la evolución se hace con respecto al valor de la función (sea ésta unidimensional ó bidimensional), en el caso de un parámetro que viniera determinado por F(x,t), la definición de un espacio cuántico para él implica la división del plano (x,t) en regiones:

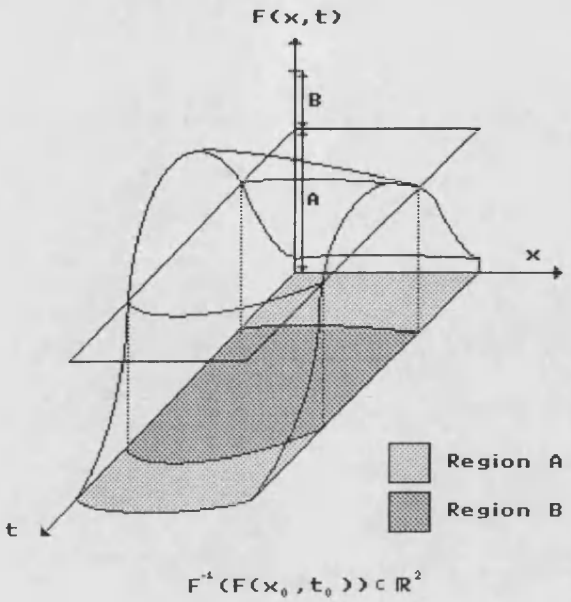

Figura 2.3

Mientras que en el ejemplo de la tabla 2.1, la definición del espacio cuántico con cinco valores implica la división del eje temporal en una serie de intervalos:

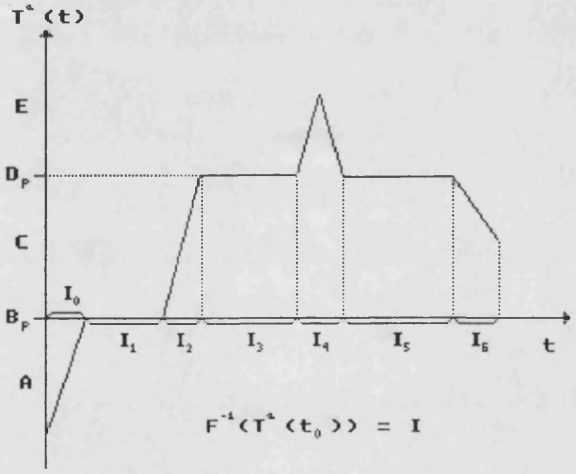

Figura 2.4

Volviendo a nuestro ejemplo de Tráfico, la densidad de vehículos en un carril de una calle es una función del tipo D(x,t). Aquí los argumentos del parámetro densidad son dos: espacio y tiempo, lo que significa que el parámetro toma valor en un plano de coordenadas (x,t), con lo que la representación espacio-temporal de D es un conjunto

de regiones poligonales cualitativamente distintas, como puede observarse en la figura 2.5.

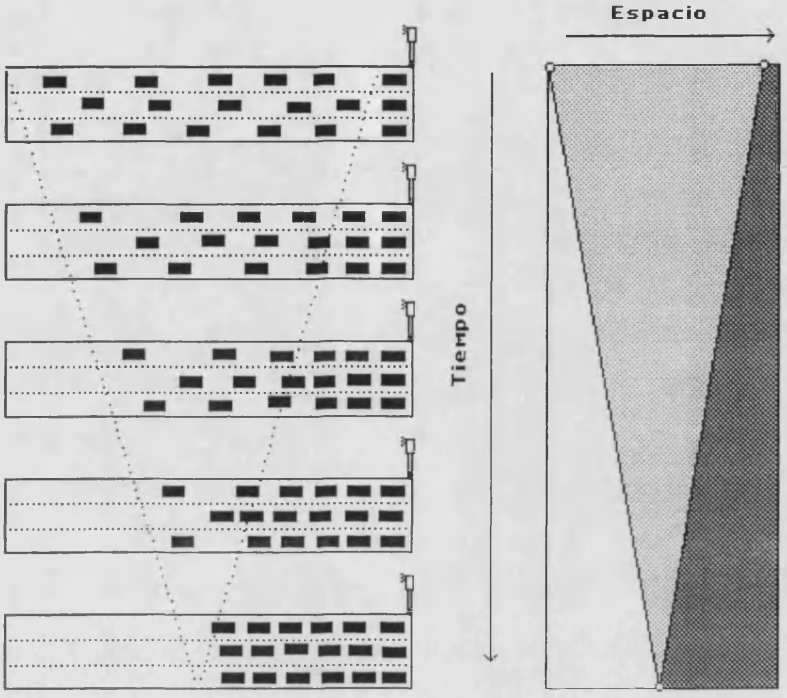

Figura 2.5

El formalismo de razonamiento cualitativo que presentaremos en el presente capítulo es una primera aproximación hacia el desarrollo de un formalismo de representación cualitativa que pueda ser capaz de tratar con sistemas físicos solamente describibles a través de parámetros multidimensionales. Además, el formalismo debe permitir razonar sobre el comportamiento de un sistema, a través de la consideración de parámetros unidimensionales y bidimensionales. Este formalismo puede permitir, entre otras cosas, la simulación de una manera eficiente de sistemas físicos complejos tales como: Circuitos hidráulicos (propagación de ondas de choque y cambios de nivel), Redes de Tráfico Urbano, Circuitos de microondas (ondas estacionarias), etc...

II.2 EL FORMALISMO (λ,T)

Llamamos formalismo (λ,t) a un paradigma de razonamiento cualitativo desarrollado para tratar con sistemas físicos solamente describibles a través de parámetros multidimensionales y cuyas características se mostrarán más adelante. Esencialmente puede considerarse como una generalización de los paradigmas de Forbus

(Qualitative Process Theory) y de De Kleer (Envision) que utiliza no solamente los parámetros puntuales que estos utilizan ($F_i(t)$) sino también parámetros multidimensionales ($F_i(\lambda_1,...,\lambda_t)$). El formalismo ofrece dos herramientas fundamentales:

-1) Un método de representación jerárquica del conocimiento sobre el sistema físico que se pretende modelar. Este conocimiento se representa de una forma granular desde las especificaciones generales del dominio de aplicación, hasta las características particulares del sistema tratado. Este método es independiente del dominio de aplicación al que pertenece el sistema físico.

-2) Un motor de inferencia capaz tanto de manejar el conocimiento obtenido por el método anterior, como de contestar a preguntas que se puedan hacer sobre el comportamiento del sistema.

Vamos a empezar presentando el método de representación del conocimiento, mientras que su manejo se describirá en próximas secciones. En Capítulos siguientes trataremos aplicaciones completas del uso del formalismo.

El método en cuestión consiste en una serie de especificaciones sobre cómo representar la descripción física del sistema a un formato adecuado para que el motor de inferencia pueda trabajar con ella.

El primer paso del método consiste en la creación de un árbol jerárquico de clases cuyas instancias serán los objetos componentes del sistema físico a ser modelado.

En este árbol cada clase debe especificar: los atributos estáticos y los parámetros dinámicos comunes a los objetos de la clase.

Los atributos estáticos son constantes que toman valor de un espacio cuántico (dominio cualitativo de valores), por lo que se especifica el espacio cuántico (y el valor del atributo si es común a todos los objetos de la clase). Los parámetros dinámicos son atributos dependientes del tiempo que especifican el tipo de parámetro (Si el parámetro es $F(t)$ o $F(\lambda,t)$) y su correspondiente espacio cuántico en el que toma valores.

En un segundo paso hay que definir las ecuaciones físicas y sus condiciones de contorno, que deben ser formuladas como ecuaciones cualitativas. Estas ecuaciones

cualitativas toman la forma de restricciones entre los valores de los parámetros de cada objeto; cada restricción posee un conjunto de inecuaciones cualitativas cuyo cumplimiento indica que la restricción está activa.

Las ecuaciones cualitativas están ordenadas jerárquicamente de acuerdo al proceso físico al que pertenecen, y pueden incluir relaciones diferenciales e integrales. Este árbol de ecuaciones cualitativas está asociado a las clases de objetos de tal manera que cada clase de objetos tendrá definidas las ecuaciones cualitativas que pueden estar activas entre sus objetos.

El tercer paso es describir la estructura estática del sistema físico que se pretende modelar (hasta ahora sólo hemos estado describiendo el dominio de aplicación). De acuerdo con la orientación a objeto, basta instanciar las clases previamente definidas que componen el dominio de aplicación (clases de objetos, relaciones y ecuaciones cualitativas). Los objetos que componen el sistema se instancian de las clases de objetos, las relaciones topológicas entre ellos se instancian a las clases de relaciones y las ecuaciones cualitativas que describen el comportamiento de cada objeto son heredadas de la clase del objeto.

En un cuarto paso, finalmente, debe describirse tanto el estado inicial del sistema como el comportamiento temporal de los atributos variables externos al sistema. Esto último se debe al hecho de que la evolución temporal del resto del universo ajeno al sistema bajo control puede afectar a este (si no está aislado), pero esta evolución temporal no puede ser deducida, por estar fuera del sistema que se representa, por lo tanto debe darse su representación. Por ejemplo, en un sistema controlado debemos describir como dato para cualquier simulación la secuencia de órdenes que el sistema de control le dará al sistema, pues de lo contrario no podría realizarse la simulación. En el caso de que la secuencia de órdenes del sistema de control dependa de la evolución del sistema no hay más remedio que considerar como sistema a simular el conjunto formado por el sistema original y su sistema de control. La figura 2.6 muestra la estructura de la base de datos temporal del sistema, donde se muestra el estado inicial del sistema (en la figura, el estado inicial del dominio interno) y el comportamiento temporal de los atributos variables externos al sistema (en la figura, el estado inicial del dominio externo y su evolución temporal); El resultado final de la ejecución del motor de inferencia sobre estos datos permite obtener la descripción de la evolución temporal del sistema (en la figura, la evolución temporal del dominio interno). Los términos utilizados en la figura se definen más adelante en la Taxonomía de Objetos (sección II.2.1.1).

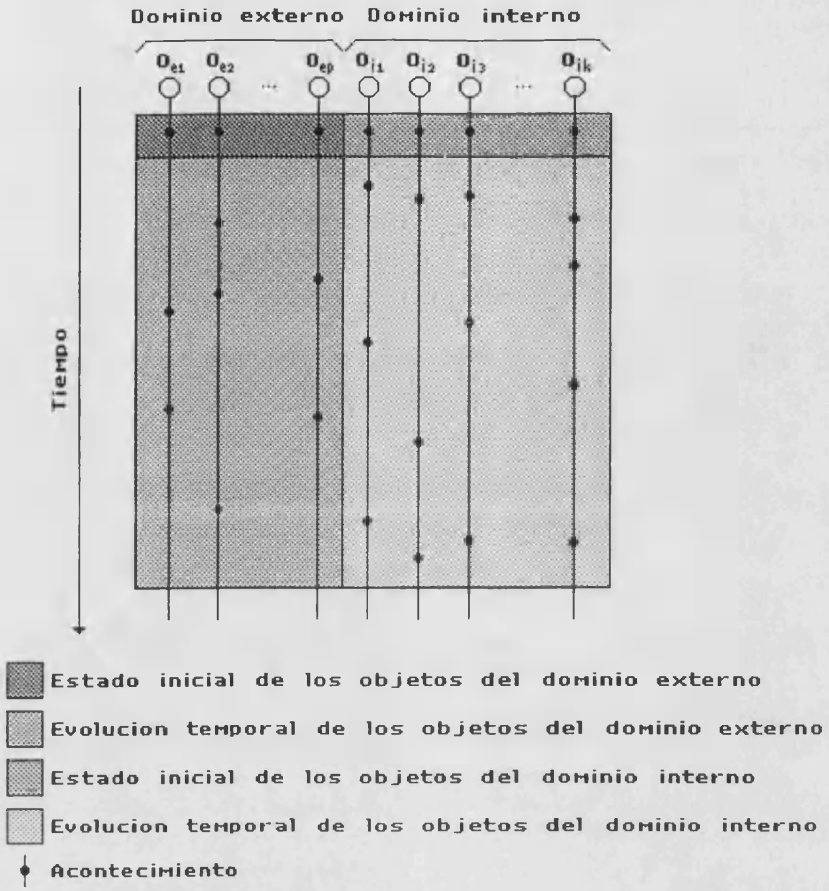

Figura 2.6

Con el objetivo de evitar cualquier ambigüedad, vamos a definir con mayor precisión los términos que utilizaremos:

-1) **Dominio de Aplicación**: Es un subconjunto del conjunto de todos los posibles sistemas físicos, caracterizado por el hecho de que el conocimiento necesario para modelarlo es común a todos sus elementos y a su vez está claramente diferenciado del conocimiento sobre el resto de los sistemas físicos (por ejemplo, el Control de Tráfico Urbano es un Dominio de Aplicación, pues toda red de tráfico urbano es modelable a través de las ecuaciones de tráfico en cruces y calles).

-2) **Sistema Físico**: Región del espacio que se pretende estudiar, separada del resto del universo por medio de una frontera imaginaria. Si esta frontera no puede ser atravesada por ningún flujo de información, decimos que el sistema es aislado. Por ejemplo, en el caso del CTU, un sistema físico podría ser la red urbana de la Ciudad de Valencia.

-3) **Sistema de Control**: Parte del resto del universo que interacciona con el sistema físico con el objetivo de hacerlo evolucionar de acuerdo a alguna pauta deseable de

comportamiento (por ejemplo, siguiendo con el ejemplo del CTU, el sistema de control de tráfico de la Ciudad de Valencia comprendería la sala de control de tráfico, la red semafórica, varios paneles variables, la red de sensores de bucle y la red de cámaras).

-4) **Parámetro**: Función del tiempo (puede depender o no además de otras variables no temporales) utilizada como base para describir el comportamiento temporal de un sistema (por ejemplo, el flujo de vehículos en una entrada a un cruce de la red de tráfico urbano) .

-5) **Parámetro fijo**: Es un parámetro (y por lo tanto una función del tiempo) cuya evolución temporal se fija de antemano y no puede ser modificada por ningún elemento del sistema (estos parámetros se utilizan para describir condiciones de contorno que cambian con el tiempo). Por ejemplo, los efectores del sistema de control sobre el sistema físico y las acciones del resto del universo sobre el sistema físico).

Una vez definidos los términos básicos, estamos en condiciones de estudiar de una manera más detallada el método de representación del conocimiento.

II.2.1 Representacion del Conocimiento del Sistema

De acuerdo con lo ya visto, la representación del sistema se obtiene tras la aplicación del método de especificación jerárquica del conocimiento al sistema físico que se pretende estudiar. En ella, el conocimiento, sobre el sistema se representa de una manera granular y jerárquica, desde las especificaciones del dominio de aplicación hasta las características particulares del sistema.

Una vez obtenida la representación del sistema, ésta constituye la base de conocimiento, y es susceptible de ser utilizada directamente por el motor de inferencia para razonar sobre el sistema físico al que representa.

El contenido de esta base de conocimiento está organizado en una estructura que puede dividirse en cinco partes:

- taxonomía de objetos
- taxonomía de relaciones
- taxonomía de parámetros
- comportamiento de objetos
- comportamiento de relaciones

II.2.1.1 Taxonomía de Objetos

La taxonomía de objetos consiste en la especificación de todas las clases de objetos físicos que son relevantes en cada dominio de aplicación, ordenadas de forma jerárquica.

En la taxonomía de objetos, las clases dependen del dominio de aplicación, y sus instancias del sistema físico particular que se pretende estudiar. Por ejemplo, en CTU, el grafo de clases es el mismo para cualquier posible red urbana, mientras que sus instancias dependen de su topología.

En esta taxonomía predefinimos tres clases (ver fig. 2.7), que encabezan el grafo de clases de cualquier sistema físico:

- **Dominio de Aplicación**: Superclase de todos los objetos (Ej.: Una red de tráfico urbano).
- **Clases Exteriores**: Subclase de "Dominio de Aplicación" que agrupa todos aquellos objetos que representan las condiciones de contorno del sistema fisico que está siendo representado (Ej.: entradas y salidas de una red de tráfico urbano).
- **Clases Interiores**: Subclase de "Dominio de Aplicación" que agrupa todas las clases no incluidas en "Clases Exteriores" (Ej.: calles interiores[2] y cruces que componen la red urbana).

Las subclases "Clases Interiores" y "Clases Exteriores" constituyen una partición de la clase "Dominio de Aplicación". El hecho de dividir las clases según esta partición presenta ventajas computacionales, debido a que permite dar un tratamiento diferente a las condiciones de contorno de la simulación, cosa especialmente útil al tratar con aquellas condiciones que cambian con el tiempo.

[2] Calle interior es una calle cuyos extremos pertenecen ambos a la red urbana. Una acceso a la ciudad , por ejemplo, no es una calle interior.

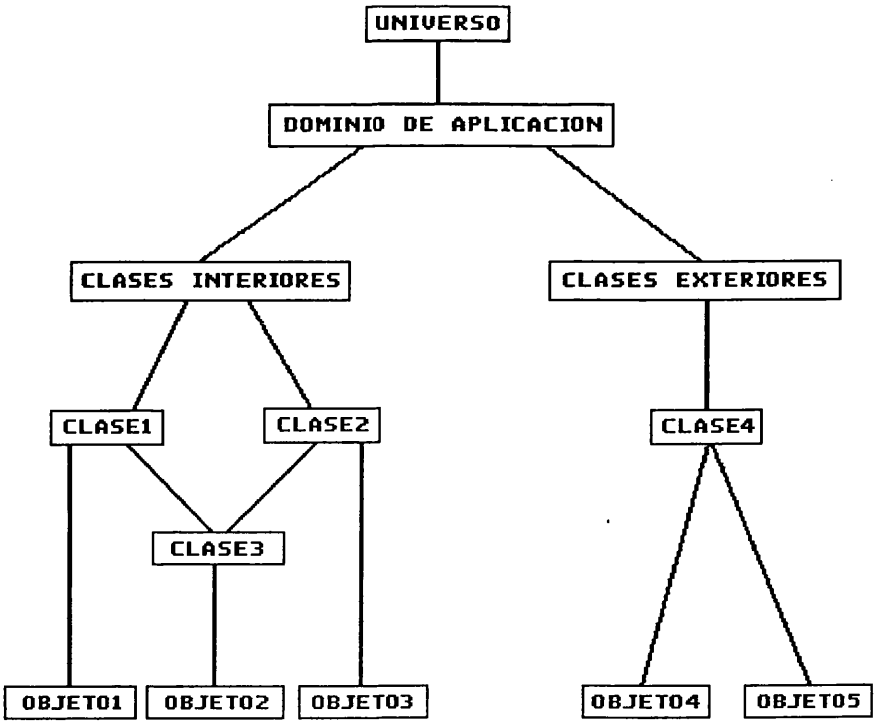

Figura 2.7

Como en toda orientación a objetos, se permite herencia múltiple entre las clases de objetos: Una clase puede ser subclase directa de más de una clase (esto implica que la estructura de clases no puede representarse por medio de un árbol). En la figura 2.7 representamos los nombres de todos los objetos que componen el sistema físico como instancias de estas clases.

La utilidad de la taxonomía de objetos es la de servir de marco principal para representar el resto del conocimiento tanto sobre el dominio de aplicación como sobre el sistema físico. De esta manera podemos definir en la especificación de una clase de forma simultánea los parámetros de estado que ésta precisa, y el conjunto de ecuaciones que determinan la evolución de los mismos.

La especificación de una clase de objetos contiene los siguientes elementos:

- Nombre de la clase.
- Nombre de todos los parámetros utilizados.
- Tipo de cada uno de los parámetros (F, F(t) ó $F(\lambda,t)$).
- Espacio cuántico donde toma valor cada parámetro.
- Ecuaciones cualitativas que ligan los parámetros entre sí.

Por su parte, la especificación de un objeto contiene los siguientes elementos:

- Nombre del objeto.
- Clase a la que pertenece.
- Nombre de los parámetros que no pueden ser especificados a nivel de clase, debido a que cambian de un objeto a otro.
- Valores de todos los parámetros que sean constantes.
- Base temporal que representa la evolución del objeto, señalando los cambios de valor de sus respectivos parámetros F(t) y F(λ,t).

Como ejemplo de lo anterior veamos la definición de la clase de objetos "calle" y del objeto "calle1":

Clase **calle**:
- atributo: longitud
  tipo: constante
  dominio: numeros naturales (N)
- atributo: numero_de_carriles
  tipo: constante
  dominio: numeros naturales (excepto el cero) (N^*)
- atributo: densidad
  tipo: F(λ,t)
  dominio: {d1,d2,d3,d4,d5,d6,d7,stop}
- ecuación: lambda_mínimo(densidad) = 0
  ecuación: lambda_máximo(densidad) = valor(longitud)

Objeto **calle1**:
- subclase_de: calle
- atributos_de_objeto: no tiene.
- valor(longitud): 100
- valor(numero_de_carriles): 2
- base_temporal:
  event(E1,state(densidad,[[d1,0,100]]))
  time(E1,0)
  event(E2,state(densidad,[[d4,0,0],[d1,0,100]]))
  time(E2,2)
  event(E3,state(densidad,[[d4,0,100]]))
  time(E3,11)

...

En el ejemplo anterior, el único parámetro que evoluciona en el tiempo es "densidad", por lo que la base temporal de la "calle1" sólo contiene la evolución de este parámetro. La explicación detallada de la representación de parámetros (λ,t) se verá más adelante. La base de datos temporal, que se comentará más adelante, se compone únicamente de acontecimientos, y se deduce el estado del objeto en los intervalos entre acontecimientos utilizando la propiedad de persistencia temporal, de forma análoga a como lo hace el Cálculo de Acontecimientos para Cambio Contínuo [Shanahan,90]. Un breve resumen del mismo puede encontrarse en el anexo H.

II.2.1.2 Taxonomía de Relaciones

La taxonomía de relaciones consiste en la especificación jerárquica de las clases de relaciones causales que pueden existir entre los objetos de cualquier sistema de un dominio de aplicación.

Definimos una relación entre dos objetos como la representación conceptual de un flujo directo de información entre los mismos (por ejemplo, una relación podría ser el intercambio de información que se produce entre una calle y un cruce con objeto de transmitir las variaciones en el flujo de vehículos). A efectos prácticos esto significa que los cambios de valor de las variables de uno de los dos objetos pueden afectar al valor de las variables de otro objeto, sin la intervención de ningún otro objeto intermedio.

Conviene aclarar en este punto que existe una gran diferencia entre el concepto de relación tal y como se acaba de definir y el de mensaje, tal y como se utiliza en la Programación Orientada a Objetos. En efecto, una relación representa la conexión que permite un intercambio de información entre dos objetos del sistema físico, mientras que un mensaje es una orden directa dada por un objeto a otro objeto en tiempo de ejecución. Por otro lado, también existen analogías entre ambos conceptos, siendo la más importante el que el motor de inferencia del formalismo (λ,t) mantiene la existencia de una relación mediante un intercambio de mensajes entre objetos, que obviamente siguen la estructura de la relación.

Las relaciones se agrupan en clases, dependiendo de las clases de objetos que relacionan y de las variables de los mismos que definen el flujo de información. Las

conexiones topológicas están representadas en esta taxonomía de relaciones por tres clases (fig.2.8):

- **Relaciones**: Superclase de todas las relaciones.
- **Acciones**: Superclase de todas las clases que relacionan dos o más objetos por medio de flujos de información unidireccionales (por ejemplo, la orden de parada que un semáforo en rojo impone a una calle puede ser considerada como una acción del objeto semáforo sobre el objeto calle).
- **Restricciones**: Superclase de todas las clases que relacionan dos o más objetos por medio de flujos de información bidireccionales (por ejemplo, la relación que existe entre el flujo de vehículos en una entrada a la ciudad y la calle a la que desemboca puede ser considerada como una restricción entre el objeto entrada y el objeto calle: en ella es posible tanto que la entrada deje de suministrar vehículos a la calle como que la calle se colapse y deje de permitir la entrada a la ciudad).

Nótese que "Acciones" y "Restricciones" son subclases de "Relaciones" de la que forman una partición. Esta taxonomía también permite herencia múltiple. En la figura 2.8 puede verse la estructura topológica del sistema físico expresada a través de las instancias de estas clases de relaciones.

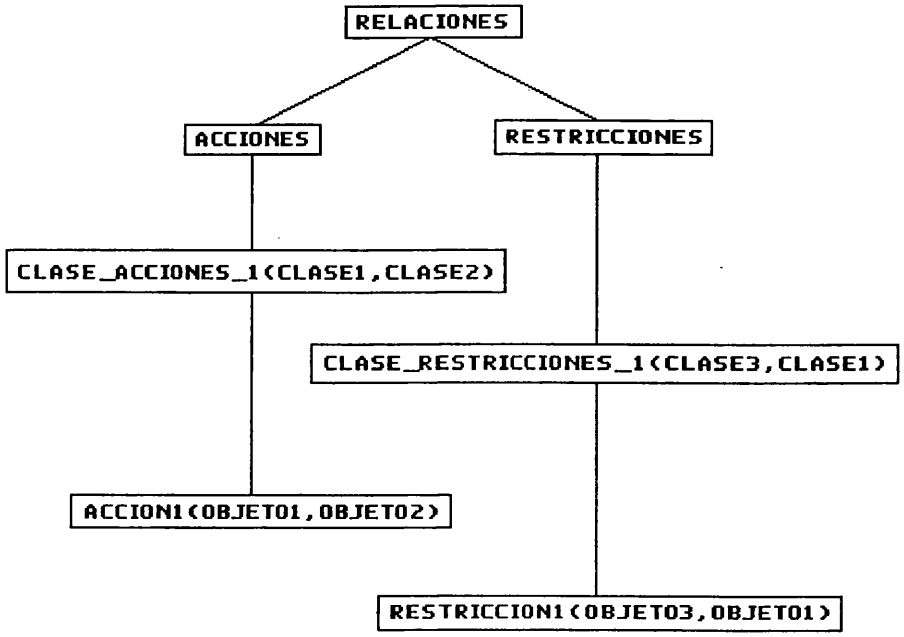

Figura 2.8

Los componentes de una relación son:

- Los nombres de los objetos que relaciona.
- Los nombres de las variables de cada objeto que afecta.
- Las igualdades impuestas entre las variables de estos objetos (Excepto igualdades entre dos variables de un mismo objeto, que deben figurar en el conocimiento de la clase del objeto).

Como ejemplo práctico de relación, veamos la relación establecida entre una tubería de conducción de agua y un grifo (ver figura 2.9). Estos dos objetos componen un sistema físico cuyo estado es describible mediante cuatro parámetros (dos por cada objeto) del tipo (F(λ,t)): La presión y el flujo en la tubería y la presión y el flujo en el grifo.

La restricción entre ambos puede ser expresada como sigue (P:Presión, F:Flujo):

$P_{grifo}(x_0,t) = P_{tubo}(x_f,t)$ (1)
$F_{grifo}(x_0,t) = F_{tubo}(x_f,t)$ (2)

Este sistema puede verse en la figura siguiente:

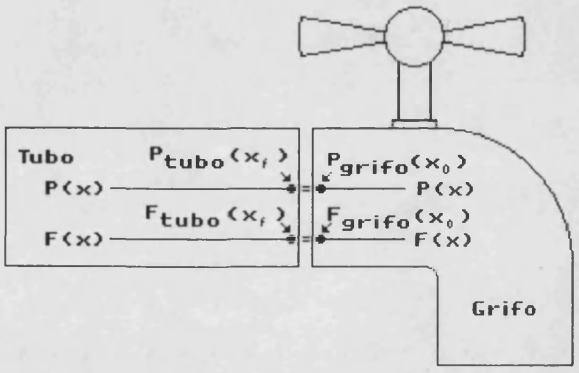

Figura 2.9

Es interesante remarcar que, en el ejemplo anterior, la relación entre las instancias de grifo1 y tubo1 viene expresada por medio de dos restricciones (1), (2), siempre válidas, que permiten bidireccionalidad en la propagación de los cambios de presión a través de la misma variable.

En un sistema tradicional, cada objeto debe tener un conjunto fijo de entradas y otro de salidas para describirlo como un autómata finito. Las relaciones a través de restricciones permiten crear conexiones entre objetos que son simultáneamente entradas y salidas, lo que evita la innecesaria duplicación de los parámetros de los objetos, duplicación que habríamos tenido que hacer para expresar la bidireccionalidad de los cambios (por ejemplo: cerrar el grifo propaga un aumento de presión desde el grifo 1 hacia el tubo 1, mientras que cortar el flujo de entrada del tubo 1 propaga un descenso de presión desde el tubo 1 hacia el grifo 1).

II.2.1.3 Taxonomía de Parámetros

Teniendo en cuenta que uno de los objetivos del formalismo es permitir la representación de parámetros bidimensionales (o n-dimensionales) de los objetos junto con los parámetros temporales unidimensionales, es necesario definir nuevas primitivas para la utilización de ambos tipos de parámetros. Sus respectivas representaciones en la base de datos temporal del simulador son diferentes, por lo que es conveniente agruparlos en clases distintas. La taxonomía de parámetros tiene cuatro clases especiales:

- **Parámetros**: Superclase de todos los parámetros.
- **F**:Agrupa los parámetros que no varían a lo largo del tiempo.
- **F(t)**:Agrupa los parámetros que son funciones únicamente del tiempo.
- **F(λ,t)**: Agrupa los parámetros que son funciones del tiempo y de otra variable continua.

Las instancias de los parámetros no se definen, puesto que se definen de forma automática al instanciarse los objetos que las necesitan. Un ejemplo de taxonomía de parámetros puede ser observado en la figura 2.10.

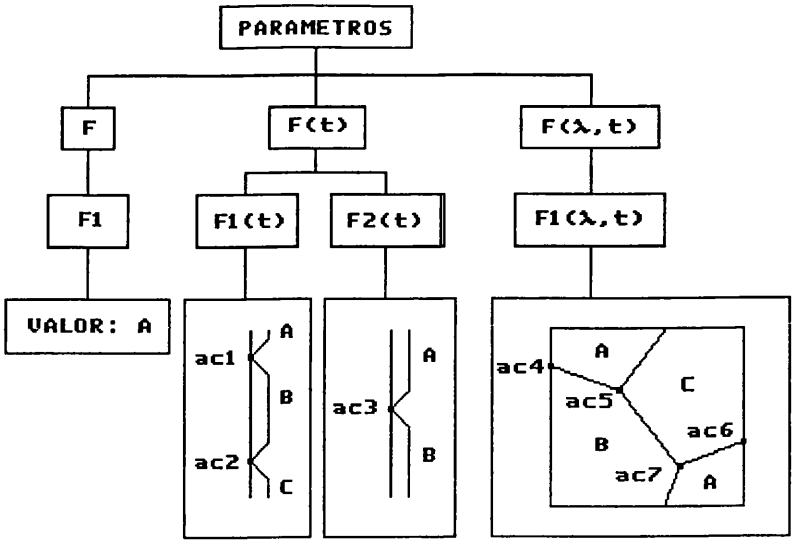

Figura 2.10

II.2.1.4 Comportamiento de Objetos

El comportamiento dinámico de los objetos viene determinado a través de un grupo de ecuaciones entre los parámetros de cada objeto, que están definidos a nivel de clase como ya hemos indicado. Una ecuación se define como una restricción entre parámetros. Veamos un ejemplo: Siendo A,B,C y D parámetros, las siguientes expresiones son consideradas como ecuaciones por el motor de inferencia:

ACCIONES:
[A] = [B] + [C]
IF [A] = Q1 THEN [B] = Q3
IF [A] IN {Q1,Q2} THEN [B] = Q4

RESTRICCIONES:
[A] = [B]
[A] = [B]*[C] + [D]

Donde [A],[B],[C] y [D] son los valores cualitativos de A,B,C y D expresados en un espacio cuántico, y Q1, Q2, Q3 y Q4 son cuatro elementos del espacio cuántico.

El conjunto de ecuaciones de un objeto no puede contener parámetros de otros objetos. Las ecuaciones que relacionan parámetros de dos objetos diferentes siempre pueden separarse en ecuaciones internas y una o más conexiones entre variables de objetos

diferentes. Estas conexiones siempre se pueden especificar en la taxonomía de relaciones, definida en la sección II.2.1.2.

Por otro lado las ecuaciones existentes pueden separarse en dos grupos:

- **Acciones**: Una ecuación pertenece a este conjunto cuando indica que un parámetro es una función unidireccional de otros. Esto significa que existe una relación de causalidad entre las entradas a la función y la salida del parámetro, por lo que su valor sólo puede ser cambiado por una acción de esta función (por ejemplo, en el caso de un objeto de tipo grifo, el flujo de fluido que lo atraviesa es una función unidireccional de la diferencia de presiones a ambos lados y del estado de apertura de la llave del grifo).
- **Restricciones**: El conjunto de ecuaciones cuyos elementos no son acciones (por ejemplo, la densidad de vehículos en una entrada de un cruce está relacionada con la densidad de vehículos en las demás entradas y salidas del cruce por una serie de ecuaciones bidireccionales, en las que tan pronto se propaga una congestión desde las salidas hacia las entradas, como un descenso en la afluencia de vehículos en la dirección opuesta).

Nótese que las acciones y restricciones aquí descritas no son las mismas que aparecen en la taxonomía de relaciones. Mientras aquéllas se refieren a los flujos de información entre objetos, éstas se refieren al comportamiento interno de los mismos.

Debe señalarse que aun cuando una ecuación indique que un parámetro es una función explícita de otros, esto no implica que la ecuación sea unidireccional. Por ejemplo, en una intersección urbana el flujo de salida es igual al flujo de entrada, sin embargo la relación es bidireccional (si el flujo de salida decrece, debido a una congestión, fuerza al flujo de entrada a decrecer, y si el flujo de entrada decreciera por falta de vehículos, haría decrecer al flujo de salida). En la literatura [Williams.84] puede encontrarse un análisis detallado de las diferencias entre las acciones unidireccionales y bidireccionales.

II.2.1.5 Comportamiento de Relaciones

Las relaciones, tal como las hemos definido en la sección II.2.1.2, son entidades cuya función es actuar de canales de comunicación entre objetos, sin capacidad de procesar la información que por ellos circula (este hecho no restringe la generalidad del concepto de relación, puesto que si un elemento de comunicación entre objetos

necesita procesar la información que por él circula, también se le considera un objeto). Sin embargo, aunque la sección II.2.1.2 establece los elementos necesarios para representar y clasificar una relación entre dos objetos determinados, no establece los medios para analizar la forma concreta como esta relación transfiere la información entre ambos. Es por ello que en esta sección tratamos sobre la forma particular como se transmite la información en las relaciones existentes en la "taxonomía de relaciones" (II.2.1.2) de un sistema dado. Esta separación entre la clasificación de las relaciones y la descripción de su estructura es similar a la realizada con los objetos físicos, donde existe una "taxonomía de objetos" (II.2.1.1) y un "comportamiento de objetos" (II.2.1.4).

La forma como una relación determinada entre dos objetos transfiere la información entre ambos es mediante una serie de ligaduras entre sendos parámetros de cada objeto; donde cada ligadura, compuesta por dos parámetros, uno de cada objeto, obliga a ambos a mantener el mismo valor cualquiera que sean los cambios a los que se vean sometidos. De esta forma, la descripción del comportamiento de una relación se reduce a una enumeración de las ligaduras individuales que la componen.

Un ejemplo de comportamiento de una relación entre objetos es el siguiente: Imaginemos una relación "conexión_tubo_grifo" entre objetos de la clase "tubo" y objetos de la clase "grifo", y supongamos que la forma como transmite la información entre ambas clases es a través del parámetro "presión" y del parámetro "flujo", en este caso, la descripción del comportamiento de la relación se realizaría a través de dos ligaduras, en la forma:

conexión_tubo_grifo(tubo,grifo)
    ligadura: valor(presión_al_final,tubo) = valor(presión_al_comienzo,grifo)
    ligadura: valor(flujo_al_final,tubo) = valor(flujo_al_comienzo,grifo)

II.2.2 Representación de Parámetros en la Base de Datos Temporal

El simulador trabaja sobre una base de datos temporal (BDT) donde se almacena la evolución temporal de los parámetros. Estos parámetros toman valor en un espacio cuántico escogido de acuerdo a criterios físicos adecuados, tanto para los de tipo F(t) como para los de tipo $F(\lambda,t)$.

La representación de los parámetros de tipo F(t) y $F(\lambda,t)$ en la BDT se realiza de la siguiente manera:

- Representación de los parámetros F(t):

La representación del valor de F(t) en un espacio cuántico, permite considerar la evolución temporal del parámetro como un eje temporal en donde se señala cada cambio en el valor cualitativo de F como un acontecimiento. De esta manera, el eje temporal se divide de una manera natural en intervalos temporales (donde F mantiene un valor cualitativo constante) y acontecimientos puntuales (que señalan los momentos en que F cambia de valor). La representación de F(t) puede verse en la figura 2.11.

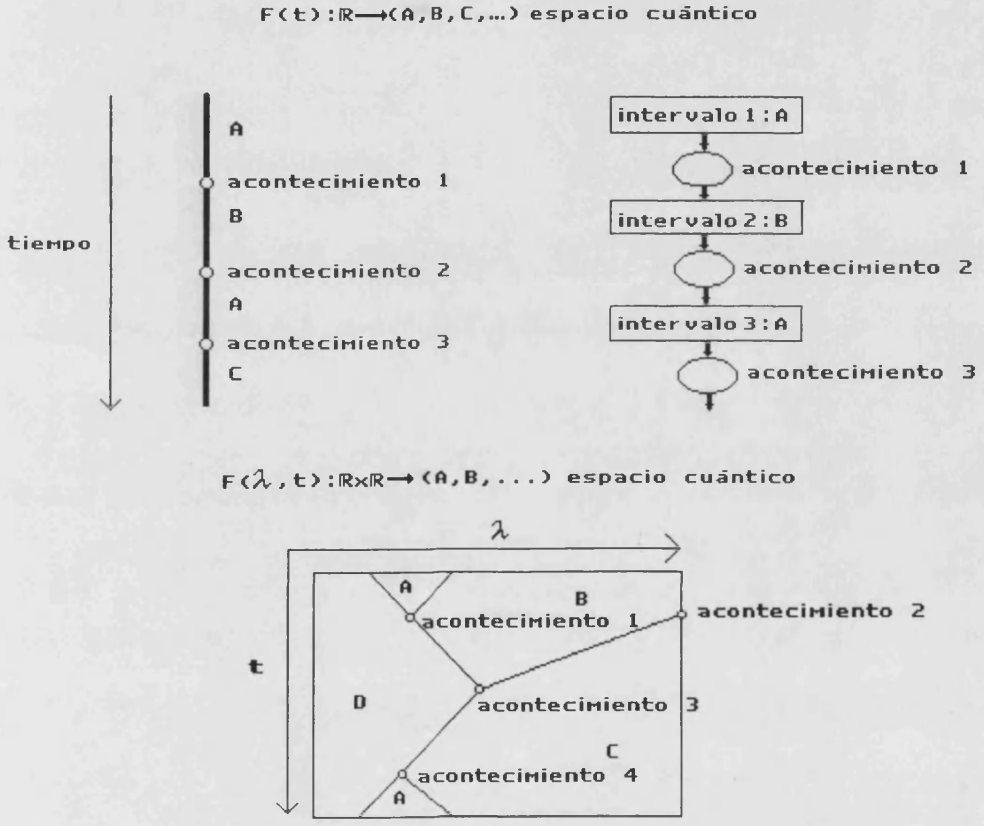

Figura 2.11

- Representación de los parámetros F(λ,t):

Los parámetros F(λ,t) son funciones bidimensionales, por lo que se representan sobre una superficie bidimensional (ver figura 2.11). Si cuantizamos F, sobre un espacio cuántico adecuado, la superficie se divide en diferentes regiones. Como primera aproximación, se introduce una nueva restricción: las fronteras entre regiones pueden ser sólo lineas rectas.

Esta restricción no es tan importante como pueda parecer a primera vista, puesto que si las ecuaciones físicas que rigen la evolución interna de nuestro parámetro F(λ,t) son homogéneas (no dependientes del origen de lambda y de tiempo), las fronteras serían líneas rectas por necesidad, puesto que de no serlo tendríamos un comportamiento distinto de la frontera ante las mismas regiones, según las coordenadas. Si tenemos en cuenta que prácticamente todas las ecuaciones físicas son homogéneas, podemos concluir que la aproximación tomada no es excesivamente restrictiva, y nos permite tratar los sistemas físicos con bastante generalidad.

Por otro lado, al trabajar con etiquetas cualitativas para definir las regiones se introduce una ligera indeterminación (una etiqueta cualitativa puede corresponder a cualquier valor de la función dentro de un intervalo) cuya principal consecuencia es que la pendiente de las fronteras también adquiere una ligera indeterminación; haciendo que la pendiente pueda tomar cualquier valor arbitrario entre una pendiente mínima y una pendiente máxima. Esta indeterminación en la pendiente implica que las fronteras dejan de ser líneas rectas, aunque están comprendidas entre dos líneas rectas bastante próximas: la construída con la pendiente mínima para esa frontera y la construída con la pendiente máxima. Esto nos obliga a introducir una segunda restricción: las fronteras son líneas rectas cuya pendiente es la media entre las pendientes máxima y mínima permitidas.

Tomando estas dos restricciones, la representación de F(λ,t) se convierte en un conjunto de regiones poligonales separadas por fronteras rectilíneas.

Definimos como acontecimientos aquellos puntos del plano donde las líneas frontera empiezan o terminan (fig. 2.11). Puesto que las regiones pueden definirse a partir de sus fronteras, y éstas a partir de los acontecimientos que las generan o terminan, se deduce que la evolución temporal de un parámetro F(λ,t) en la BDT se reduce a un grupo de acontecimientos puntuales en lambda y en el tiempo.

Los acontecimientos significativos representan hechos con auténtico significado físico. Por ejemplo, en el dominio del CTU, una cola en una calle se vacía si entran menos coches en ella de los que salen. Si esto continúa, la cola acabará desapareciendo en un determinado punto del espacio y del tiempo. Este punto se representa como un acontecimiento en la representación F(λ,t), pues tiene un significado cualitativamente importante en el comportamiento de la calle (ver figura 2.5).

II.3 RAZONAMIENTO CON EL FORMALISMO (λ,T)

El procedimiento de razonamiento con el formalismo (λ,t) se basa en el Cálculo de Acontecimientos para Cambio Contínuo (Event Calculus for Continuous Change, abreviadamente ECCC) propuesto inicialmente por Shanahan [Shanahan,89] (ver anexo H). Las razones de esta selección son las siguientes:

- La representación temporal utilizada en nuestra aproximación está basada en acontecimientos, razón por la que el Cálculo de Acontecimientos parece adecuado.

- La representación temporal utilizada en los parámetros F(λ,t) está también basada en acontecimientos, sin embargo necesita propiedades que describan un cambio contínuo a lo largo del tiempo.

El razonamiento realizado sobre los parámetros del tipo F(t) es similar al utilizado por los paradigmas de De Kleer [2] y Forbus [6], por lo que no lo discutiremos aquí. Nos centraremos en el razonamiento realizado con parámetros F(λ,t), ya que constituyen la novedad principal de esta aproximación. En estos parámetros, la evolución temporal está descrita en términos de acontecimientos significativos y propiedades generadas y terminadas por estos acontecimientos, según los principios del ECCC.

Recordemos que la representación de un parámetro F(λ,t) se basa en los acontecimientos, puesto que de ellos se deducen las fronteras y de éstas las regiones. Pongamos entonces un acontecimiento en la base de datos temporal por cada acontecimiento significativo en F(λ,t). Si hay varios acontecimientos significativos simultáneos, los agruparemos en un solo acontecimiento por razones de eficiencia (Basta colocar en el lugar reservado a la descripción del acontecimiento significativo una lista conteniendo las descripciones individuales de cada acontecimiento significativo). Podemos definir ahora dos propiedades a partir de los acontecimientos de la base temporal: la estructura cualitativa y la posición de fronteras. Estas propiedades emergen de forma natural al considerar la evolución temporal de F(λ,t).

Si cortamos el plano F(λ,t) en un tiempo t dado, obtenemos una línea compuesta por intervalos y puntos (ver figura 2.11). Cada intervalo representa el corte con una región cualitativa y se caracteriza por su etiqueta cualitativa, mientras que cada punto representa el corte con una frontera y se caracteriza por su coordenada lambda. Llamamos estructura cualitativa en el tiempo t a la lista que contiene las etiquetas

cualitativas de los intervalos que aparecen en el corte, ordenados según el sentido creciente de lambda, y llamamos posición de fronteras a una lista similar que contiene las coordenadas lambda de los puntos.

A continuación se muestra un ejemplo de las reglas que describen su interrelación [Moreno,90]:

```
nuevo_acontecimiento(X,T) :-
        intervalo(Ij),
        velocidad_de_frontera_al_principio(V1,Ij),
        velocidad_de_frontera_al_final(V2,Ij),
        V is V2 - V1,
        V < 0,
        calcular(X,T,Ij,V).
```

donde Ij denota la región j, X es la coordenada lambda, T es el tiempo transcurrido entre el estado actual y el nuevo acontecimiento, y V1 y V2 representan las velocidades de las fronteras. Esta regla sólo podrá cumplirse si el intervalo Ij termina en un período finito de tiempo (como puede verse en la figura 2.12), devolviendo en ese caso las coordenadas (X,T) del nuevo acontecimiento significativo. La figura 2.12 muestra cómo estas reglas permiten deducir nuevos acontecimientos a partir de los anteriores a lo largo del plano de representación de $F(\lambda,t)$. Estas reglas se mezclan con las ecuaciones cualitativas con el objeto de lograr un cuerpo compacto de conocimiento con el que razonar sobre la representación cualitativa temporal del comportamiento dinámico del sistema.

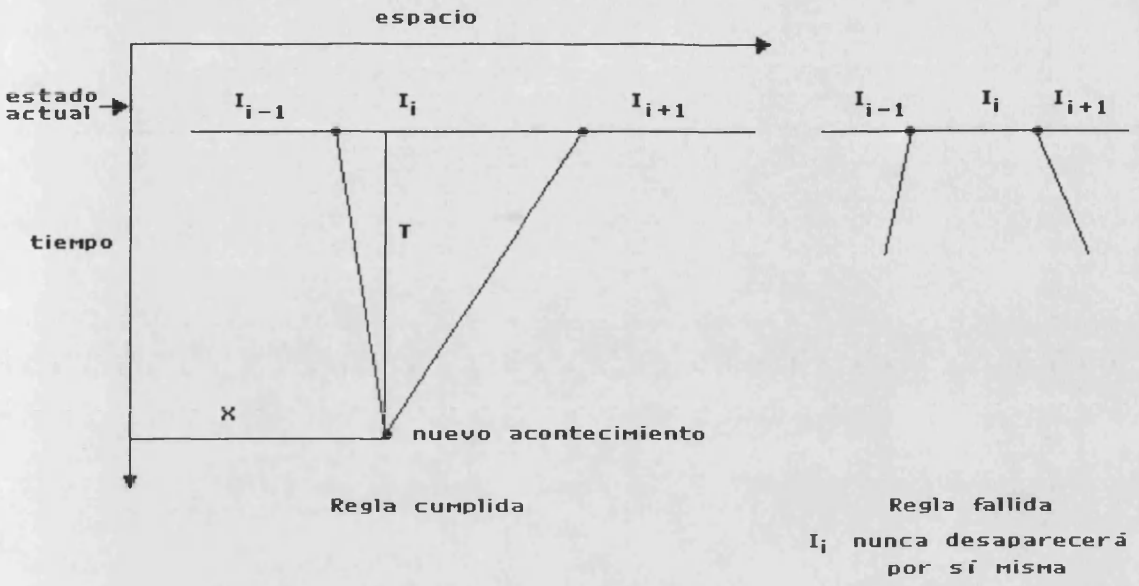

Figura 2.12

II.4 MOTOR DE INFERENCIA

El ciclo de funcionamiento del motor de inferencia es como sigue:

-1) Sean $O_1...O_n$ los objetos del sistema. Sea t_i el tiempo en el que se produjo el último suceso que tenemos almacenado en O_i. Sea $t_{min} = \min\{t_1...t_n\}$ (Para cada uno de los objetos del sistema, tomamos el tiempo del último suceso que tenemos almacenado sobre el mismo y escojemos el tiempo más pequeño de la serie así obtenida). Es fácil ver que antes de este tiempo, todos los objetos tienen su comportamiento temporal perfectamente definido, pues lo contrario violaría las condiciones bajo las que hemos obtenido este tiempo. También ocurre que después de ese tiempo el comportamiento temporal del sistema no está completamente definido (al menos el objeto al que corresponde ese tiempo tiene su último suceso conocido en ese tiempo, con lo que su comportamiento posterior está indefinido y es susceptible de ser simulado).

-2) Se forma el conjunto de sucesos que tienen la condición de suceder después o simultáneamente al tiempo anteriormente calculado. Fijarse que sólo los sucesos de este conjunto pueden generar nuevos sucesos. Los sucesos pueden darse de dos maneras:

Internamente, donde un suceso en un objeto puede causar otro suceso en el mismo objeto posteriormente.

Externamente, donde un suceso en un objeto puede causar otro suceso en otros objetos.

-3) Cada suceso es analizado con el objetivo de establecer hipótesis de posibles estados futuros que pueden ser derivados del conjunto previamente calculado.

-4) Se analiza el conjunto de hipótesis para descartar aquellas que son inconsistentes y quedarse sólo con las hipótesis consistentes, que serán convertidas en sucesos reales y guardadas en la base de datos.

-5) Si el tiempo obtenido en el paso 1 resulta ser inferior al tiempo de finalización de la simulación, volver al paso 1.

El ciclo de funcionamiento del algoritmo puede verse en la figura 2.13.

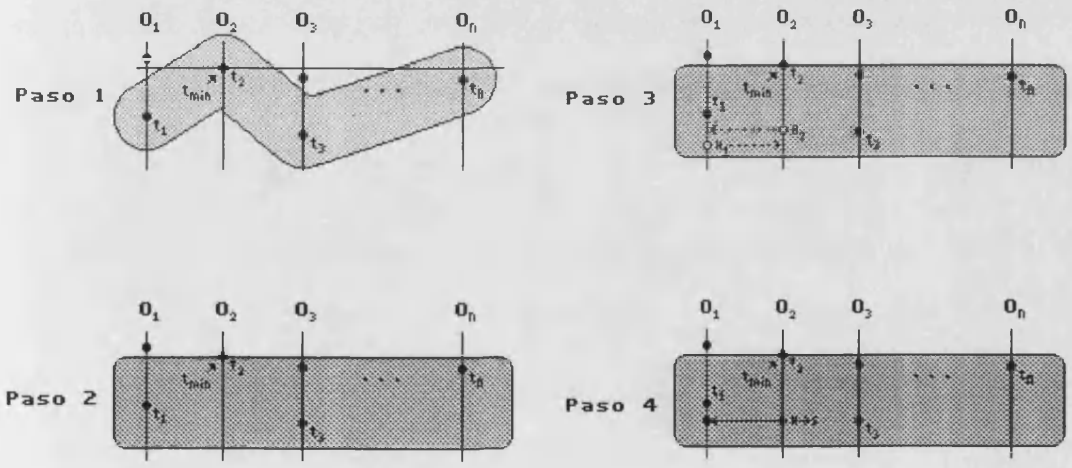

Figura 2.13

II.5 CONCLUSIONES

El formalismo (λ,t) generaliza el concepto de parámetro cualitativo de un sistema físico utilizado en los paradigmas de De Kleer, Forbus y Kuipers. Los parámetros (λ,t) son multidimensionales y están muy bien adaptados a describir sistemas que no pueden ser descritos de una forma localizada, y es necesario recurrir a una forma distribuída.

UNIVERSITAT DE VALÈNCIA BIBLIOTECA DE CIÈNCIES

Por otro lado, el formalismo (λ,t) permite hacer inferencias utilizando valores cualitativos con una referencia temporal, tal y como veremos en el capítulo III, donde aplicaremos el formalismo al Control de Tráfico Urbano y al Problema del Sistema de Tanques.

La BDT generada por el formalismo (λ,t), presenta dos características importantes:

- Es particularmente compacta (puesto que el simulador trabaja sólo sobre acontecimientos significativos). Esto permite una fácil representación gráfica y un rápido análisis de la BDT.

- Los cambios de los valores cualitativos de los parámetros del sistema se almacenan con una etiqueta temporal que se utiliza para eliminar parte de la indeterminación existente en una simulación cualitativa.

Capítulo III

REPRESENTACION DEL CONOCIMIENTO UTILIZANDO EL FORMALISMO (λ,T): APLICACION A SISTEMAS CON FLUJOS DISCRETOS Y CONTINUOS

III.1 INTRODUCCION

En el capítulo anterior hemos definido el formalismo (λ,t), con el objetivo de superar con él las limitaciones conceptuales que presentan paradigmas como De Kleer, Forbus y Kuipers. Utilizando este formalismo, el conocimiento de un sistema puede representarse por medio de parámetros del tipo F, F(t), F(λ,t), utilizando un espacio cuántico para expresar sus valores dentro de una escala cualitativa, objetos, relaciones, etc.

En el presente capítulo vamos a aplicar el formalismo (λ,t) para representar el conocimiento de dos sistemas: uno que presente flujos discretos (control de tráfico urbano) y otro que presente flujos continuos (el problema del sistema de cubas, tomado de la literatura [Shanahan,89]).

III.2 REPRESENTACION DEL CONOCIMIENTO DEL CONTROL DE TRAFICO URBANO EN EL FORMALISMO (λ,T).

Para representar conocimiento de CTU en el formalismo (λ,t), primeramente vamos a estudiar el comportamiento de los flujos de vehículos en una red de tráfico urbano, con objeto de determinar el tipo de parámetros más adecuado para su representación.

III.2.1 Comportamiento de los flujos de vehículos en una red de tráfico urbano: una aproximación cualitativa.

En esta sección vamos a partir del estudio detallado (microscópico) del flujo de tráfico, para justificar la abstracción a un modelo de más alto nivel (macroscópico), y veremos como con la introducción de tres hipótesis, podemos representar el comportamiento del flujo de tráfico utilizando un parámetro del tipo (λ,t).

III.2.1.1 Descripción de un carril aislado

Supongamos un carril aislado por el que circulan vehículos en un solo sentido, tal como se ve en la figura 3.1:

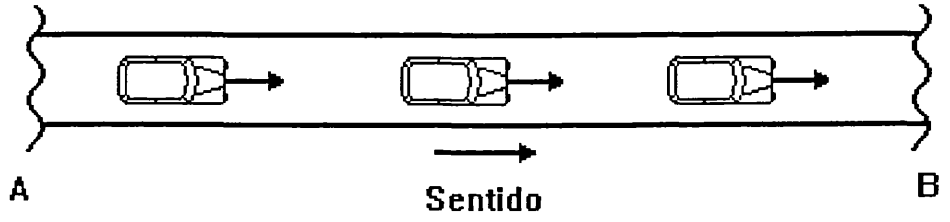

Figura 3.1

Vamos a estudiar el comportamiento del tráfico entre los puntos A y B de este carril, con objeto de tener caracterizado un segmento de estudio, de longitud finita. Nótese que para especificar completamente el estado del segmento AB en un instante dado, es suficiente con conocer la posición y velocidad de todos los vehículos que se encuentren en él. Un ejemplo puede verse en la figura 3.2:

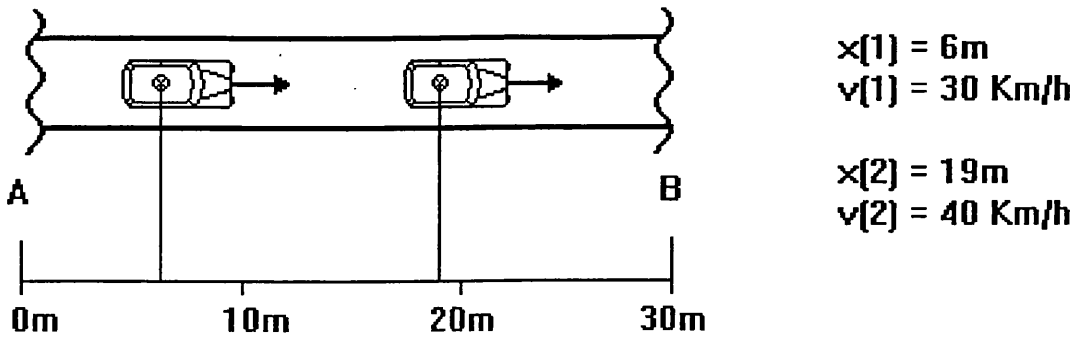

Figura 3.2

A partir de esta información se puede construir un modelo con un gran nivel de detalle, sim embargo éste no es necesario a efectos de Control de Tráfico, por ello hay que estudiar cómo simplificarlo manteniendo tanta información sobre el comportamiento dinámico como sea posible. Para ello hagamos las siguientes consideraciones:

A)- En el comportamiento real de los vehículos en un segmento como el considerado, existe una relación bien conocida entre la velocidad de un vehículo y la distancia que lo separa del siguiente:

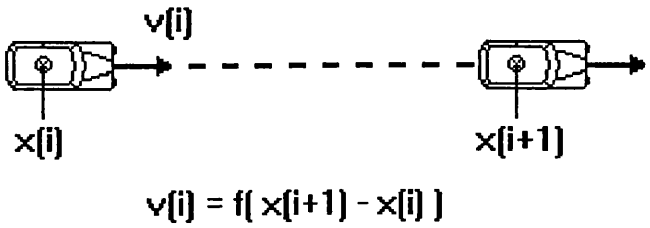

Figura 3.3

Realmente, la velocidad de un vehículo no depende únicamente de su distancia al precedente, sino también de la velocidad del mismo, por lo que la relación real sería:

$$v(i) = g(x(i+1)-x(i),v(i+1))$$

Sin embargo, v(i) es prácticamente igual a v(i+1), siendo sus diferencias de un orden de magnitud muy inferior a la influencia de la distancia x(i+1)-x(i), por lo que podemos aproximar esta fórmula por:

$$v(i) = g(x(i+1)-x(i),v(i))$$

Esta última fórmula, que contiene v(i) en ambos miembros, impone una relación entre v(i) y x(i+1)-x(i) que puede ser reescrita explícitamente en la forma:

$$v(i) = f(x(i+1)-x(i))$$

siendo f la relación mostrada en la figura 3.3.

Sobre esta relación debemos hacer las siguientes consideraciones, antes de continuar:

A.1) La relación exacta depende del vehículo y de su conductor. Existen conductores que circulan a 30 Km/h con una distancia de seguridad de 10 metros con el vehículo precedente, mientras otros pueden hacerlo a 2 metros; por otro lado, un vehículo largo obligará al vehículo que lo sigue a mantener una distancia mayor, dado que las propias longitudes de los vehículos están incluidas en x(i+1) - x(i) (ver figura 3.4):

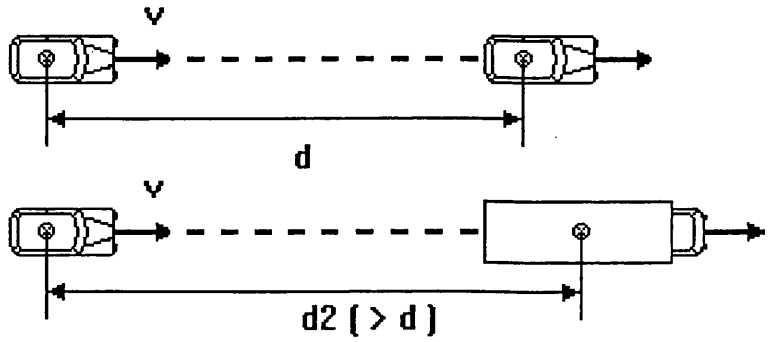

Figura 3.4

A.2) La relación entre v(i) y (x(i+1) - x(i)) no es instantánea. Un cambio en x(i+1) - x(i) tarda un tiempo en verse reflejado en v(i). Este hecho puede verse en la figura 3.5, donde se representa la evolución temporal de ambos términos para un vehículo dado. En la figura puede verse cómo la relación se cumple en zonas estacionarias, pero cuando cambia (x(i+1) - x(i)) (por ejemplo, por motivo de frenado del vehículo anterior), v(i) tarda un tiempo en ajustarse al nuevo valor de (x(i+1)-x(i)). Durante este tiempo, existe un intervalo de transición donde la relación anterior no se ha cumplido. A este hecho se lo conoce en control de procesos como inercia.

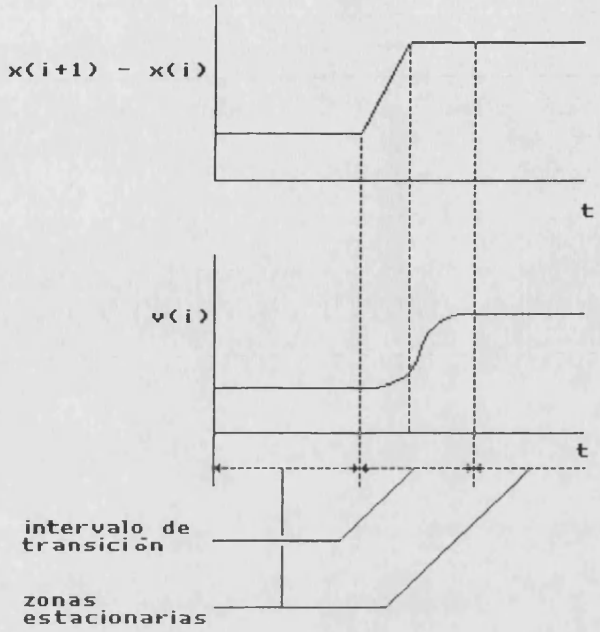

Figura 3.5

Es importante destacar que este cambio se produce en un período de tiempo no nulo, que depende del conductor.

B)- A pesar de que la relación entre x(i+1) - x(i) y v(i) no es determinista, los datos experimentales nos dicen que si representamos la relación entre v(i) y x(i+1)-x(i) para distintos conductores y/o en distintos momentos, obtenemos una gráfica como la mostrada en la figura 3.6:

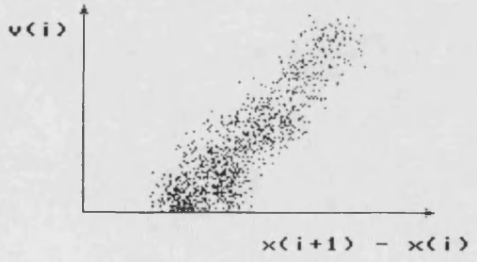

Figura 3.6

En esta gráfica podemos observar que la distribución de puntos no es completamente aleatoria: existe una clara relación entre las variables v(i) y (x(i+1) - x(i)), a pesar de

la dispersión aleatoria producida por las diferencias de comportamiento de los conductores y las diferencias de tamaño entre los vehículos.

Basándonos en las anteriores consideraciones, podemos definir la densidad del flujo de vehículos en el punto x(i) como la cantidad:

$$d(i) = \frac{1}{x(i+1) - x(i)}$$

Una justificación intuitiva de esta definición puede obtenerse del estudio de la figura 3.7

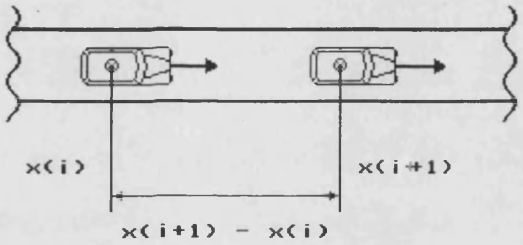

Figura 3.7

III.2.1.2 La relación densidad-velocidad

Una vez definida d(i), podemos hablar de una relación entre v(i) y d(i), cuya gráfica para distintos valores puede verse en la figura 3.8:

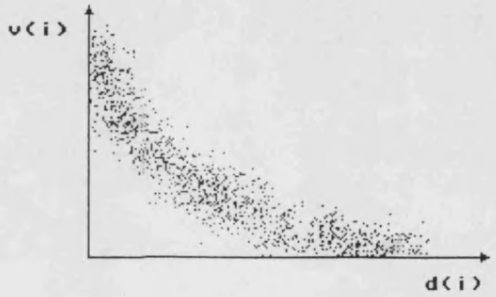

Figura 3.8

Esta gráfica de la figura 3.8, puede obtenerse a partir de la figura 3.6 por medio de una transformación matemática. Como puede verse, es una relación monótona decreciente y su gráfica "tiene el aspecto" de forma cóncava; estos dos rasgos sobreviven a la dispersión aleatoria de las muestras, por lo que pueden considerarse estos rasgos como comunes a todos los vehículos.

Retomando a la definición de d(i), y considerando ahora un escenario más general en el que haya más de dos vehículos (como el mostrado en la figura 3.9):

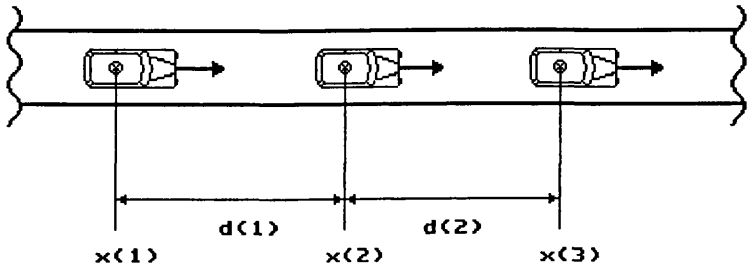

Figura 3.9

Si analizamos esta función d(i) vemos que en un instante de tiempo t, podemos hablar de una función d=d(x), donde estos puntos d(i) podrían ser representados, como se muestra en la figura 3.10:

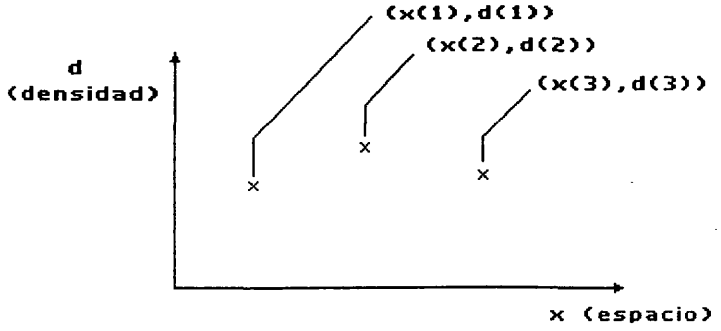

Figura 3.10

Estos puntos (x(i),d(i)) sobre el plano x-d representan el estado de la calle (pues en ellos se ven reflejadas las variables x(i) y las v(i), estas últimas a través de la relación v(i) = f(d(i))). A partir de los puntos (x(i),d(i)), podemos interpolar una función d(x) que pase por todos ellos (ver figura 3.11):

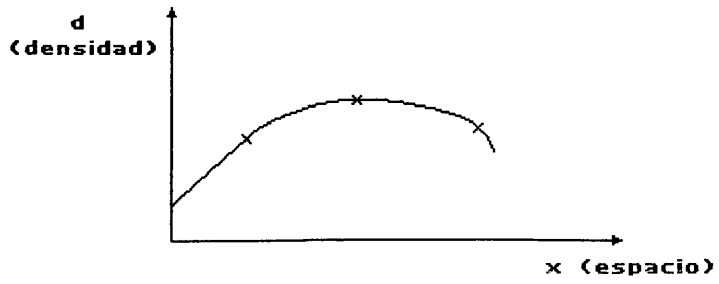

Figura 3.11

Conviene hacer notar que al definir una función continua d(x) que describa el estado de la calle en un instante t, obviamos la existencia de los vehículos; por tanto, podemos decir que esta función marca la transición de un modelo microscópico (en el que se tienen en cuenta todos y cada uno de los vehículos) a uno macroscópico (en el que no se maneja el concepto de vehículo individual).

Todo lo anterior lo hemos hecho para un instante de tiempo. Si consideramos ahora la introducción de la variable tiempo (t), obtendremos una función d(x,t) de dos variables que representa la "densidad continua" en un punto x del carril y en un tiempo t.

De igual manera, expresamos por v(x,t) una función que representa la "velocidad continua" en un punto x y en un tiempo t.

III.2.1.3 Aproximaciones y Ecuaciones Básicas

A partir de ambas funciones, podemos definir el flujo de vehículos, en un punto x y un tiempo t (lo representaremos por f(x,t)) de la siguiente forma:

$$f(x,t) = d(x,t) \cdot v(x,t)$$

con lo que basta con conocer f(x,t) y d(x,t) para conocer el estado del carril y su evolución temporal.

Puesto que los vehículos individuales no se crean ni se destruyen, el "fluido de vehículos" que éstos constituyen, cumplirá la ecuación de conservación de la materia, de forma análoga a como lo hace cualquier fluido físico. Esta ecuación de conservación impone la siguiente relación entre el flujo y la densidad:

$$\frac{\partial f(x,t)}{\partial x} + \frac{\partial d(x,t)}{\partial t} = 0$$

por otro lado, teniendo en cuenta la relación entre velocidad y densidad (ver figura 3.8), el flujo f(x,t) puede escribirse como una relación

$$f(x,t) = F(d(x,t))$$

La estructura de F puede obtenerse por vía experimental:

1) La relación estática flujo-densidad se obtiene realizando medidas experimentales de pares (flujo,densidad) en un punto de una calle donde se cumpla que d(x,t) = constante en un entorno espacio-temporal del punto de medida, y también puede deducirse a partir de la relación experimental entre v(i) y (x(i+1)-x(i)) de la figura 3.8 utilizando la definición que hemos dado para el flujo y la densidad. Esta relación puede verse en la figura 3.12:

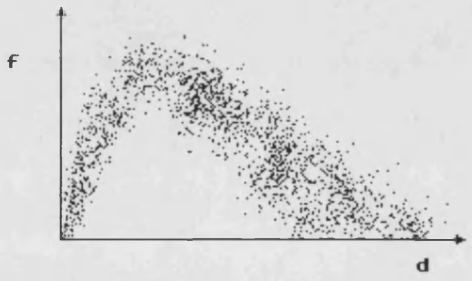

Figura 3.12

2) Si, por el contrario, suponemos que no se cumple d(x,t) = constante en un entorno del punto de medida de los pares (flujo,densidad), la figura 3.12 quedará distorsionada por el retraso que la inercia introduce en la relación flujo-densidad (recuérdese las consideraciones realizadas al analizar la influencia de la inercia en la relación entre v(i) y (x(i+1)-x(i))). Sin embargo, la inercia se debe al tiempo de reacción de los conductores, y éste corresponde a un tiempo muy pequeño, por lo que tiene sentido tomar la aproximación de que es nulo.

ASUNCION 1: tiempo de reacción = 0

El error cometido por esta aproximación es bastante pequeño, y el beneficio obtenido es grande: al hacer que el tiempo de reacción sea nulo, F no contiene integración temporal alguna sobre d(x,t), por lo que la relación estática entre f y d vista en el párrafo anterior es válida también dinámicamente.

Por tanto, la teoría macroscópica de tráfico explica el comportamiento del mismo a través de las dos siguientes ecuaciones:

(I) Ecuación de conservación:

$$\frac{\partial f(x,t)}{\partial x} + \frac{\partial d(x,t)}{\partial t} = 0$$

(II) Ecuación dinámica:

$$f(x,t) = F(d(x,t))$$

Adoptando F una forma como la mostrada en la figura 3.13:

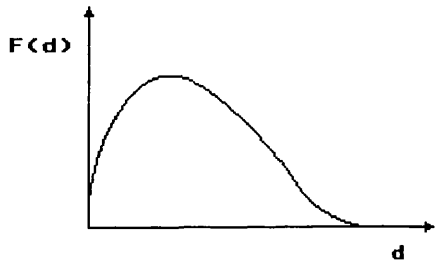

Figura 3.13

La forma particular que adopta F identifica el modelo macroscópico particular de que se trate. De todas formas, la experiencia nos dice que F tiene forma convexa, tal como se ve en la figura 3.13, y tiene por tanto una zona creciente, un máximo, y una zona decreciente; esto es común a todos los modelos, por lo que tomaremos como base la descripción cualitativa resultante de dividir el dominio de valores de la densidad en tres regiones: $[0,d_m)$,d_m y $(d_m,d_{maximo}]$, correspondiendo respectivamente a las regiones que hacen F(d) creciente, máxima y decreciente.

La figura 3.14 muestra el efecto que tiene esta descripción de F sobre la densidad, tomada como función del espacio en un momento del tiempo t.

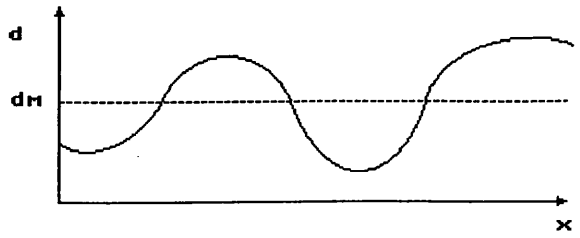

Figura 3.14

La ecuación (II), al relacionar f y d, nos permite representar el estado del tráfico únicamente por d(x,t) (puesto que f(x,t) puede deducirse de d(x,t)).

Por otra parte, si sustituímos en la ecuación (I) el valor de f(x,t) dado por la ecuación (II), obtenemos:

$$\frac{\partial F(d(x,t))}{\partial x} = -\frac{\partial d(x,t)}{\partial t}$$

$$\frac{\partial F}{\partial d} \cdot \frac{\partial d(x,t)}{\partial x} = -\frac{\partial d(x,t)}{\partial t}$$

lo que nos conduce a una tercera ecuación que llamaremos:

(III) Ecuación de la densidad

$$\frac{\partial d(x,t)}{\partial t} = -\frac{\partial d(x,t)}{\partial x} \cdot \frac{\partial F(d)}{\partial d}$$

Esta última ecuación representa la evolución temporal de d(x,t). Conociendo el estado de la calle en un tiempo t_0, $d(x,t_0)$, y la evolución temporal en los extremos del carril, podemos calcular d(x,t) para todo $t > t_0$.

Supongamos que en t=0 el estado de la calle viene representado por d(x,0), y que la relación entre F y d viene determinada por la gráfica de la figura 3.15 (donde d_{sat} representa el valor máximo posible de la densidad, correspondiente a una cola, y d_m es el valor de la densidad correspondiente al máximo de F(d)):

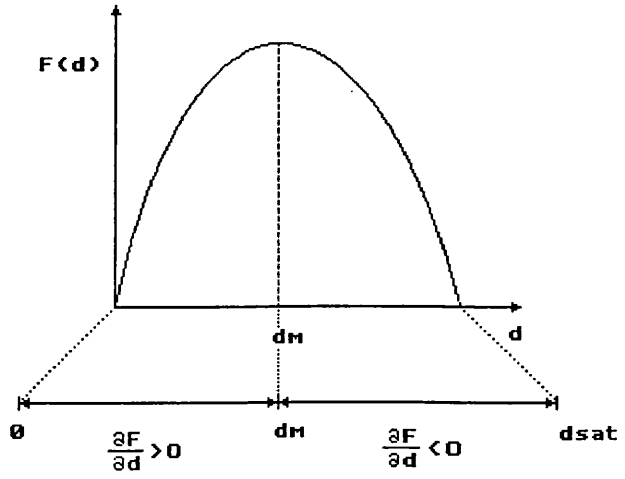

Figura 3.15

Donde d_m es el valor de d que cumple $\partial F/\partial d = 0$ (Es el máximo de F(d)).

Si analizamos la evolución de d(x,t) para distintos valores de x, podemos observar la influencia que F tiene en esta ecuación (ver figura 3.16).

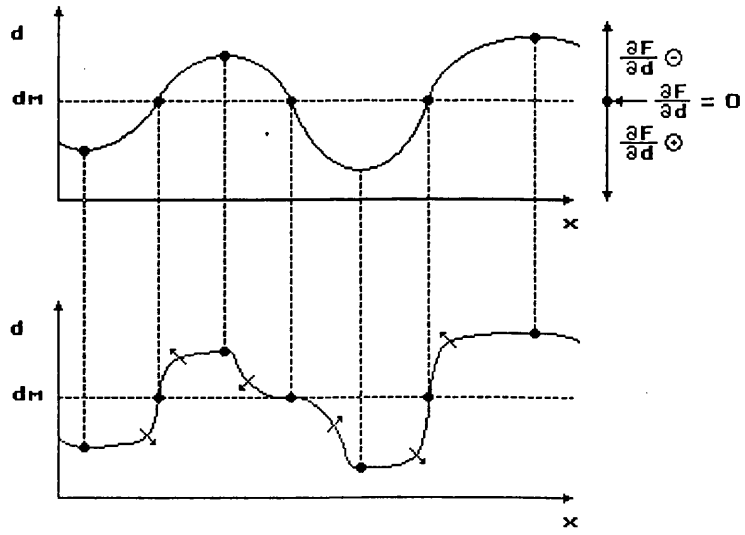

Figura 3.16

La figura 3.16 muestra cómo para cualquiera que sea la forma inicial de d(x,t) (diagrama superior de la figura 3.16), en poco tiempo evoluciona a una forma compuesta por mesetas y escalones (diagrama inferior de la figura 3.16), siguiendo un proceso de estratificación que se aclara en la figura 3.17. Conviene aclarar que esta evolución se produce de forma independiente a cualquier discretización cualitativa, como consecuencia directa de la forma de la función F.

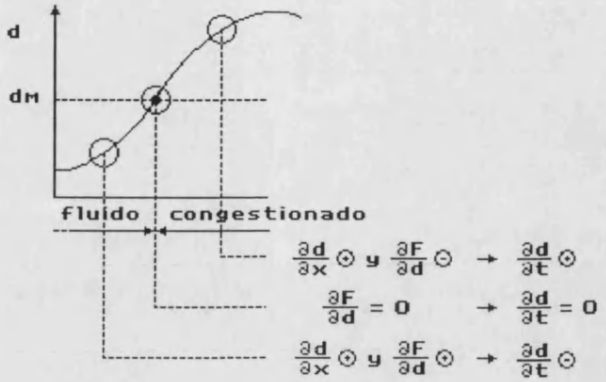

Figura 3.17

El proceso de estratificación mostrado en la figura 3.17 y que se aclara más adelante, evoluciona hasta llegar a una forma final de función escalón para d(x,t) en todos aquellos puntos donde d(x) pasa de la zona fluida ($d < d_m$) a la zona congestionada ($d > d_m$) (ver figura 3.18):

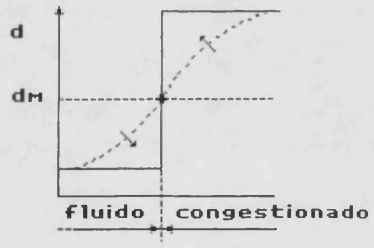

Figura 3.18

Expliquemos este proceso de estratificación, analizando la evolución temporal de cada fragmento de la curva d(x,t) en función de x (ver figura 3.17), que presenta tres zonas diferenciadas:

- Centro: Corresponde al punto $d = d_m$, en el que $\partial F/\partial d = 0$, mientras $\partial d/\partial x$ sea finito. Por la ecuación (III), deducimos que $\partial d/\partial t = 0$, lo que significa que este punto permanecerá constante a lo largo del tiempo.

- Derecha: Corresponde a la zona $d > d_m$, el valor de $\partial F/\partial d$ es negativo, y como $\partial d/\partial x > 0$ (ver figura 3.17), se deduce que $\partial d/\partial t$ es positivo, según la ecuación (III). Esto significa que en toda la parte derecha de la curva, las densidades crecen con el

tiempo. Sin embargo, están acotadas por el valor límite de d por la derecha (puesto que crecer más que él supone invertir el signo de la derivada espacial).

- Izquierda: de forma análoga, se ve que las densidades decrecen en esta zona, con el límite impuesto por el valor límite de d por la izquierda.

En la realidad, el escalón de la figura 3.18 no llega a formarse completamente, debido a la aceleración finita de los vehículos, que distorsiona la posición del máximo de F(d) en casos de frenada. Sin embargo la diferencia entre la realidad y la función escalón es muy pequeña y no afecta al comportamiento global del tráfico, como puede verse en la figura 3.19:

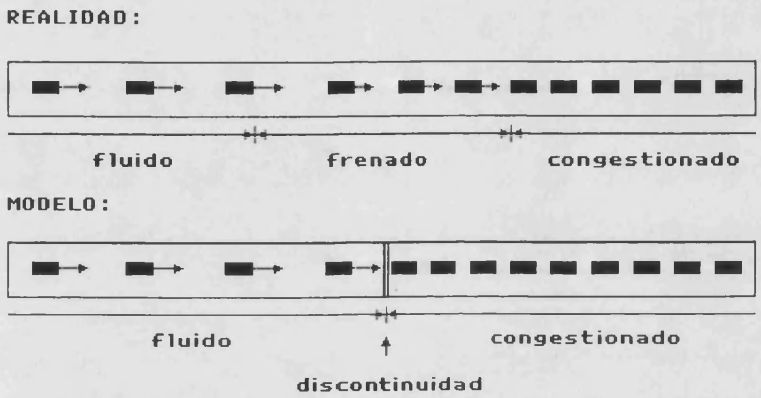

Figura 3.19

Demostraremos ahora que la discontinuidad se forma en un tiempo finito, y además que éste es muy corto, lo que nos permitirá considerar la función d(x,t) como compuesta únicamente de mesetas y discontinuidades.

Sabemos que la zona de transición del estado fluido a congestionado permanece fija mientras d(x,t) sea continua y se pueda aplicar la ecuación (III), tal y como hemos visto al analizar la evolución de la figura 3.17. Por otro lado, el flujo correspondiente al estado fluido en general es distinto del flujo correspondiente al estado congestionado, por lo que el fragmento de carril considerado en la figura 3.17 tiene un flujo resultante no nulo. Este flujo ha de ser compensado de alguna manera, por la ecuación de conservación de vehículos, y la única forma de hacerlo es forzando la formación de la discontinuidad, puesto que el número de vehículos en el fragmento de carril considerado no es el mismo con d(x,t) continua (figura 3.17) que con d(x,t) discontínua (figura 3.18). Por otro lado, el tiempo máximo en el que puede suceder

esta transición será la diferencia de vehículos entre ambos estados de la función d(x,t), dividido por el flujo resultante que hay que compensar. De esta forma comprobamos que la discontinuidad se forma en tiempo finito, y cálculos basados en la experiencia nos dicen que además este tiempo es despreciable (del orden del segundo).

Una vez generado el escalón en la función d(x,t), no cambiará de forma por sí mismo, puesto que se trata de una forma estable. Sin embargo, lo que si ocurrirá es que comenzará a desplazarse con velocidad uniforme a lo largo del tramo, como veremos a continuación.

Puesto que el estado fluido no tiene en general el mismo flujo que el estado congestionado, el flujo resultante de vehículos a ambos lados del escalón no es generalmente nulo, sin embargo, la ecuación de conservación (I) exige que los vehículos se conserven. Este flujo resultante debe compensarse de alguna manera, y no puede hacerlo cambiando la forma del escalón porque la ecuación (III) no lo permite. La única posibilidad de compensarlo consiste en mover el escalón con velocidad uniforme, hecho que puede deducirse de la ecuación de conservación de vehículos en su forma integral. Efectivamente, puesto que las densidades de vehículos a ambos lados del escalón son diferentes, el movimiento del escalón implica un aumento o disminución de la cantidad de vehículos del tramo de carril considerado, que basta para igualar al flujo que pretendíamos compensar. Además, de esta igualación podemos deducir la velocidad y el sentido de movimiento del escalón en función de los estados de densidad de vehículos a la derecha y a la izquierda del mismo.

Recordemos que la aparición de la discontinuidad es consecuencia de haber supuesto que el tiempo de reacción de los conductores es nulo. En la realidad todo vehículo necesita un tiempo para pasar de una zona de densidad a otra, y por tanto si no se hace esta suposición, se llega a que en la realidad aparecen unas zonas de frenado de tamaño no nulo en lugar de estas discontinuidades puntuales; sin embargo, como las dos se comportan de la misma manera, nuestra aproximación puede considerarse como una aceptable aproximación al comportamiento real.

Analicemos ahora la evolución de d(x) en las zonas de transición de una zona congestionada a otra fluída:

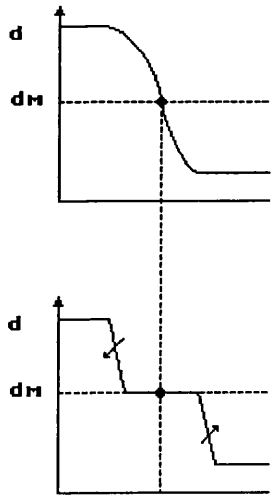

Figura 3.20

Vemos que aquí ocurre el fenómeno inverso: La zona de transición se allana progresivamente, tendiendo a aparecer una planicie con d = dm que se va extendiendo hacia el exterior. Podemos considerar que la función original d(x) tiende hacia una función con dos escalones que se alejan.

Resumiendo lo presentado hasta ahora, podemos ver que la evolución temporal de las funciones d(x) tienden a una función compuesta por mesetas y discontinuidades, que aunque no llega a alcanzarse en la realidad, tampoco constituye una aproximación muy diferente de la misma. Esta es la razón que justifica la introducción de una segunda hipótesis:

ASUNCION 2: Suponemos que d(x,t) es una función escalón que evoluciona en el tiempo de acuerdo a la ecuación de conservación de vehículos.

Asumiendo esta aproximación, la gráfica de una posible función d(x,t) en un tiempo t podría ser la mostrada en la figura 3.21.

Definimos d(x,t) de la siguiente manera:
Sea la función característica de un intervalo ($I \subseteq R$),

$$\chi_I = \begin{cases} 1 & x \in I \\ 0 & x \notin I \end{cases}$$

Tomaremos d(x,t) como una función escalón:

$$d(x,t) = \sum_{i \in J_X} \alpha_i \chi_{I_i}$$

Donde J_X es un conjunto finito de índices (en principio distinto para cada x) .

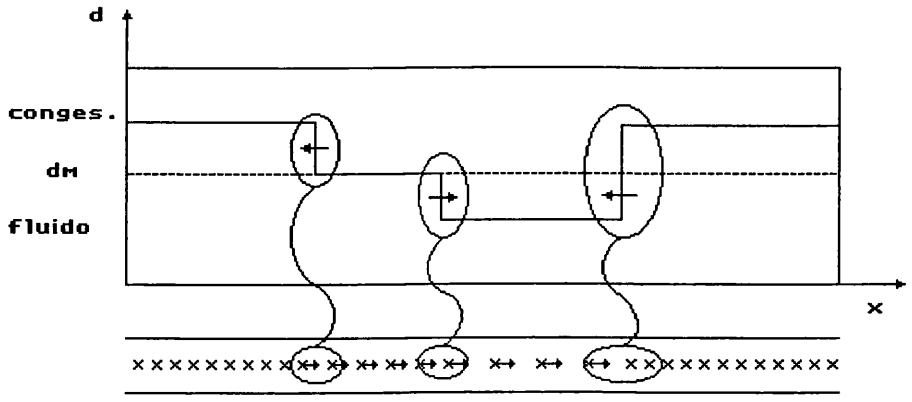

Figura 3.21

Por el apartado anterior, sabemos que las discontinuidades de la gráfica 3.21, se desplazan a una velocidad constante en virtud de la ecuación integral de conservación de vehículos; por ello, para cada discontinuidad δ de la función d(x,t), podemos definir el concepto de velocidad de frontera:

Definición:

Sea $d(x,t) = \sum_{i \in J_X} \alpha_i \chi_{I_i}$

Sea δ una discontinuidad de d(x,t), llamaremos velocidad de la frontera δ:

$$V_{\text{frontera } \delta} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon,t) - f(x-\varepsilon,t)}{d(x+\varepsilon,t)-d(x-\varepsilon,t)}$$

Una idea intuitiva puede verse en la figura 3.22:

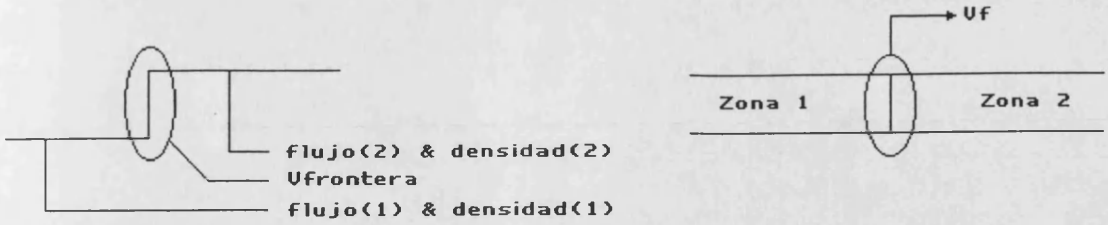

Figura 3.22

La velocidad de frontera depende, por consiguiente, del flujo y la densidad a ambos lados de la discontinuidad; como el flujo en un punto es función de la densidad, según la ecuación (II), tenemos que la velocidad de frontera es función de la densidad en la zona 1 y de la densidad en la zona 2 (ver figura 3.22). La figura 3.23 muestra el signo de la velocidad de frontera como función de ambas densidades:

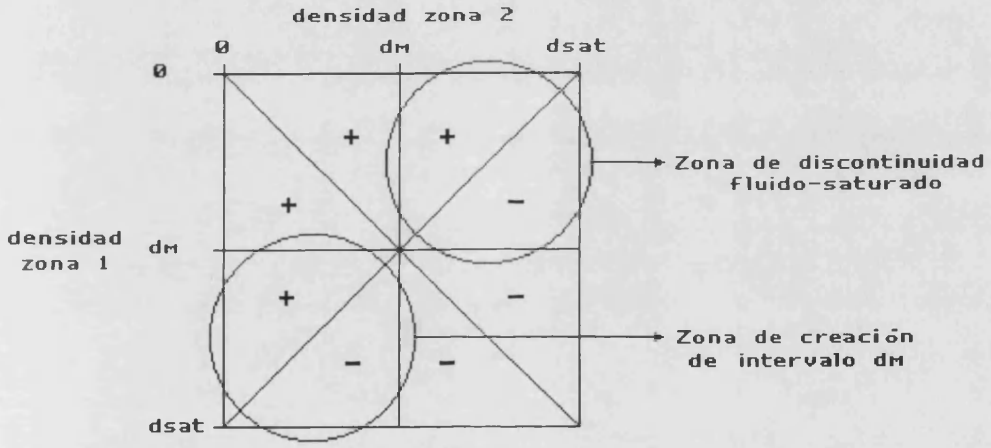

Figura 3.23

Anteriormente hemos visto cómo una transición fluído-congestionado (ver figura 3.17) se convertía en una discontinuidad, mientras que una transición congestionado-fluído se convertía en una planicie de valor dm flanqueada por dos discontinuidades que se alejaban entre sí (ver figura 3.20). La región de aparición de ambos fenómenos está señalada en la figura 3.23. Es de destacar que la figura 3.23 es independiente de cualquier discretización cualitativa de la densidad de vehículos, debido a que en ella, las densidades en las zonas 1 y 2 son funciones contínuas de x y t.

III.2.1.4 El Proceso de Discretización

El último paso consiste en la discretización de las densidades; para ello hay que definir un dominio cuántico adecuado. Por ejemplo, un dominio posible sería el siguiente:

Densidad				
cero	fluido	máximo	congestionado	saturado
0	(0,dm)	dm	(dm,dsat)	dsat

Tabla 3.24

Este espacio cuántico puede tener más o menos elementos de acuerdo con el tratamiento que vayamos a hacer.

A partir de ahora, el estado de una calle viene dado por un conjunto de intervalos espaciales caracterizados por una etiqueta cualitativa (correspondiente al valor cualitativo de la densidad), como puede verse en la figura 3.25:

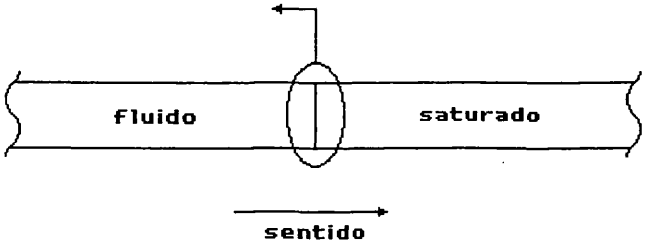

Figura 3.25

Sin embargo, debemos tener en cuenta que al discretizar d(x,t), el dominio de valores de la velocidad de frontera (ver figura 3.23) se hace igualmente discreto. En efecto, la definición de velocidad de frontera que fue expuesta en la sección anterior muestra cómo la velocidad de frontera en función de los valores de flujo y densidad a ambos lados de la frontera, y como los valores de flujo y densidad pueden tomar cualquier valor dentro de un intervalo de valores posibles, correspondientes a los valores cualitativos de densidad, obtenemos como resultado que la velocidad de frontera podrá tomar cualquier valor entre un mínimo Vf- y un máximo Vf+, dependientes de los valores cualitativos de la densidad a ambos lados de la frontera. Este hecho se ilustra en la figura 3.26:

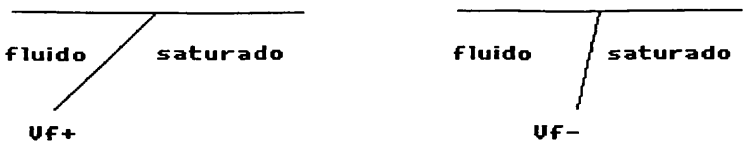

Figura 3.26

ASUNCION 3: Tomamos como valor de la velocidad de la frontera el centro del intervalo de valores [Vf-, Vf+] que puede tomar, con objeto de eliminar la incertidumbre.

La introducción de esta tercera hipótesis no es tan restrictiva como puede parecer en una primera aproximación, como podemos ver analizandos los dos casos posibles:

1) Las regiones de densidad cualitativas que la frontera separa son muy diferentes (por ejemplo, un estado cola con un estado fluido): en este caso, si analizamos la definición de velocidad de frontera, comprobamos que el intervalo [Vf-,Vf+] se hace muy pequeño, debido a la relación inversa entre la diferencia de densidades y la velocidad de frontera, por lo que la aproximación tomada resulta razonable.

2) Las regiones de densidad cualitativas que la frontera separa son similares (por ejemplo, dos estados fluidos ligeramente diferentes): En este caso, el intervalo [Vf-,Vf+] se hace muy grande, y el error aumenta de forma considerable; sin embargo, la importancia de una frontera que separa dos estados similares es muy pequeña, y su influencia en el comportamiento del sistema, casi nula, por lo que también parece razonable tomar esta aproximación. En el caso extremo, ambas regiones son iguales, en cuyo caso la velocidad está completamente indeterminada, pero no importa porque la frontera entre dos estados idénticos no influye en absoluto sobre el comportamiento de la calle.

III.2.1.5 El comportamiento del Tráfico representado por el formalismo (λ,t)

Como resumen final, obtenemos que la evolución temporal de la función bidimensional d(x,t), puede representarse de la forma ejemplificada en la figura 3.27, y queda codificada en la descripción cualitativa que puede verse a su izquierda, donde se observa que ha quedado reducida a unos cuantos puntos significativos que marcan los choques entre fronteras y la consiguiente creación o desaparición de fronteras.

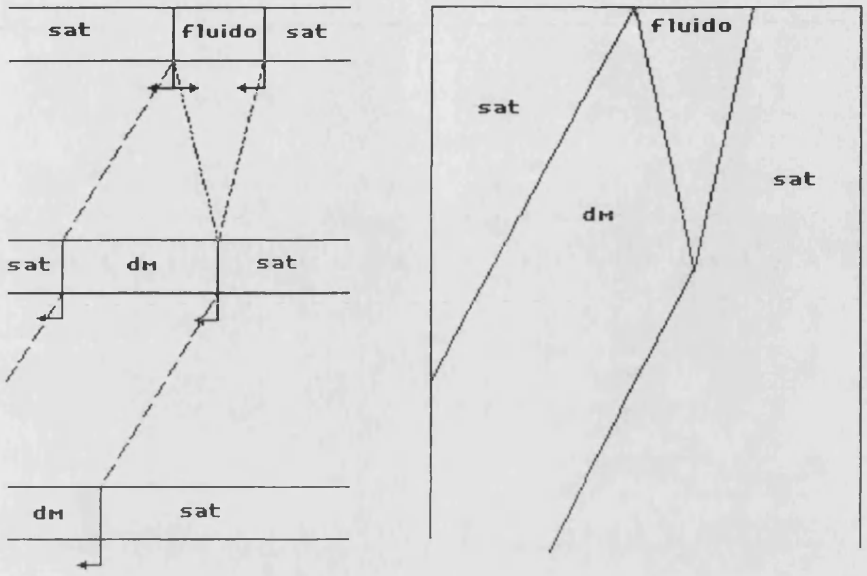

Figura 3.27

La representación mostrada en la figura 3.27 de la densidad de vehículos como función bidimensional del espacio y del tiempo corresponde a la representación de un parámetro de tipo (λ,t), descrito en el capítulo anterior, por lo que queda plenamente justificado aplicar el formalismo (λ,t) al problema de representación de los flujos de vehículos en una red de tráfico urbano. A continuación hemos de introducir en él, el correspondiente razonamiento asociado a su control.

III.2.2 Representación del Control de Tráfico Urbano con el Formalismo (λ,T).

El objetivo de esta sección es aplicar el uso del formalismo (λ,t) para la representación del conocimiento de un sistema real, con el doble objetivo de ilustrar con un ejemplo el formalismo (λ,t) (cuyo desarrollo ha sido hasta ahora abstracto), y de comprobar la utilidad del mismo como herramienta de análisis de situaciones complejas del mundo real. El sistema es el CTU, un sistema con flujos no discretos, cuya naturaleza y posible representación hemos estudiado en la sección III.2.1.

Supongamos un escenario de trafico sencillo (ver figura 3.28), compuesto por un cruce y varias calles adyacentes a él. Naturalmente no podemos, ni en éste ni en ningun otro sistema, hacer una representación total del sistema. Debemos escojer un nivel de granularidad adecuado para el objetivo que pretendemos alcanzar. Este nivel

se ha fijado teniendo en cuenta la finalidad a la que esta destinado el modelo: realizar tareas de razonamiento del comportamiento del tráfico urbano especialmente en el caso que éste pueda considerarse como "congestionado", situación en la cual dejan de ser óptimos los sistemas algorítmicos de CTU y se empieza a necesitar un razonamiento heurístico apropiado para cada situación.

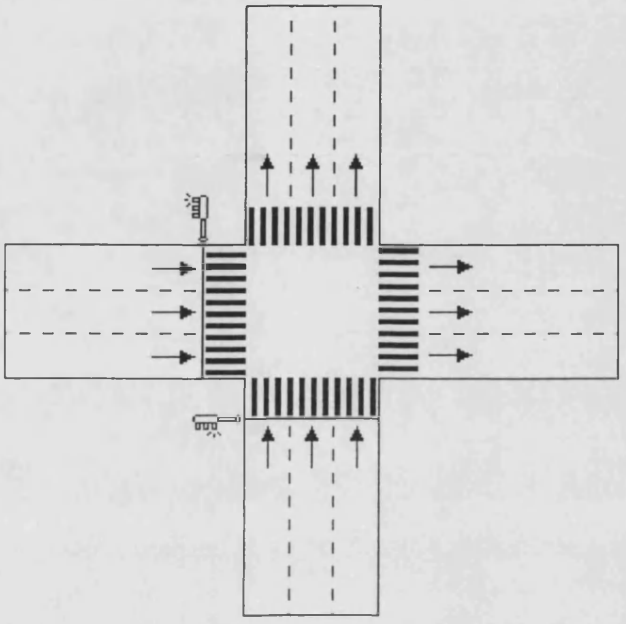

Figura 3.28

En los lenguajes de programación imperativos, la elección de una u otra estructura de datos para representar información sobre una aplicación, está muy relacionada con el algoritmo que trabajará sobre ella y con los objetivos de la aplicación [Wirth,XX]. De igual forma, la primera tarea para la representación del conocimiento es tener establecido cuál es el objetivo de la misma (que tratamiento se va ha hacer del conocimiento). Esto es necesario para evitar una especificación de objetos, relaciones y variables que no sean necesarias para realizar la tarea objetivo y que incluso podrían no ser deseables, por motivos de claridad y eficiencia.

III.2.2.1 Descomposición Orientada a Objetos

La primera tarea a considerar es decidir qué objetos son realmente necesarios para describir el sistema. Por ejemplo, una representación gráfica del sistema escogido puede verse en la figura 3.29:

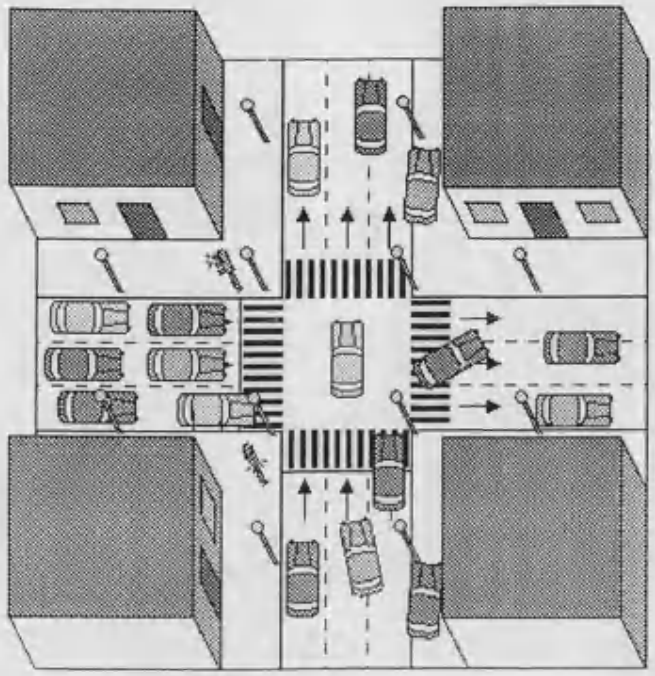

Figura 3.29

los objetos componentes del sistema son: calles, aceras, manzanas de casas, cruces, farolas, coches, alcantarillas, papeleras, semáforos y peatones.

Para empezar, esta claro que esto no es una representación completa del sistema, pues para ello habría que incluir otro tipo de objetos como tiestos puestos en las ventanas de las fincas, etc.

Sin embargo, resulta evidente que muchos de estos objetos no van a ser tenidos en cuenta en el proceso de razonamiento, para predicción del comportamiento futuro del sistema realizado por el experto de tráfico. Los únicos objetos que este utiliza en su proceso de razonamiento son:

- calles
- cruces
- semáforos

Resulta evidente que el resto de los objetos no son necesarios en el proceso de razonamiento, excepto los coches. La razón por la que los coches no representan un tipo de objetos en el sistema es que su consideración implica una conceptualización de muy bajo nivel, que estaría muy alejada de un proceso de razonamiento sobre conceptos cognitivos que tengan un significado en la evolución del sistema.

Una descomposición jerárquica orientada a objetos del escenario de la figura 3.29 puede verse en la figura 3.30:

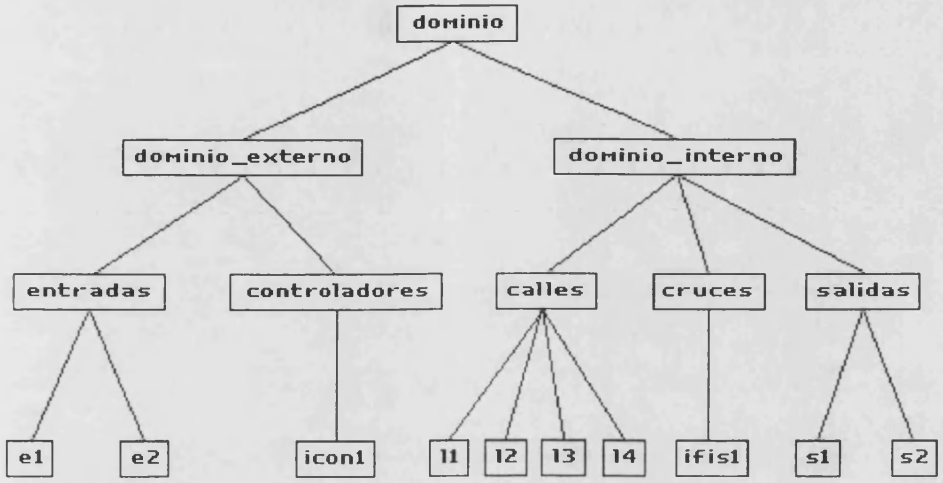

Figura 3.30

La representación del sistema en función de estos objetos puede verse en la figura 3.31.

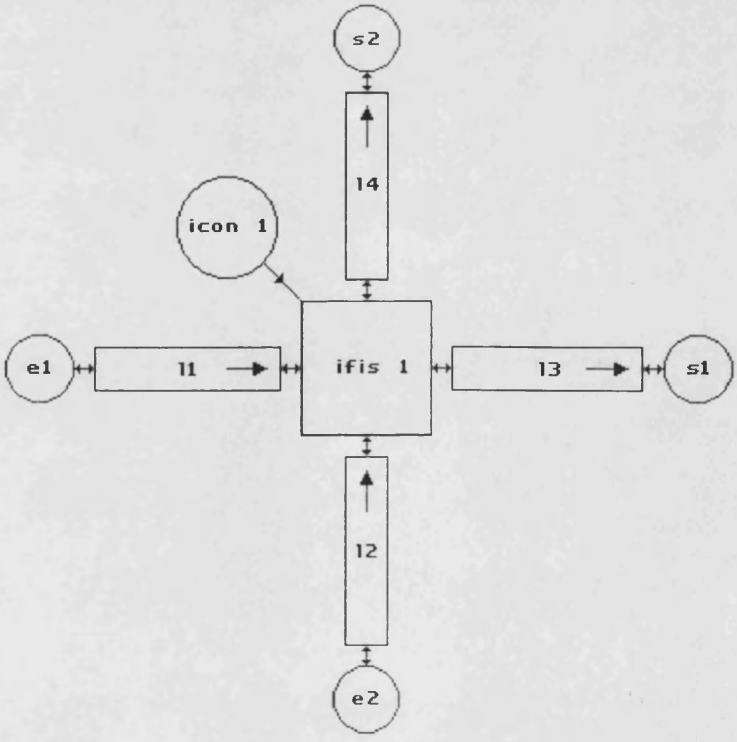

Figura 3.31

Las clases de objetos identificadas en la representación del conocimiento del escenario de la figura 3.31 han sido las calles, el cruce, el controlador del cruce (dispositivo que controla los semáforos del cruce), las entradas y las salidas del sistema. Los objetos que componen el sistema son instancias de estas clases, como puede observarse en el árbol de clases (figuras 3.30 y 3.31). Estas clases pueden a su vez ser agrupadas en clases del **dominio externo** y clases del **dominio interno**. Esta división es bastante simple, y tiene sobre todo ventajas de representación computacional que se verán mas adelante. El sistema que se pretende simular es una porción del universo aislada por una frontera conceptual del resto del mismo. Este sistema puede interactuar con el resto del universo, por lo que la información debe poder atravesar esta frontera conceptual. El dominio interno agrupa aquellas clases de objetos que no tienen contacto con la frontera conceptual, y por tanto son inaccesibles de forma directa al resto del universo. El dominio externo agrupa el resto de clases, que son las que mantienen contacto con la frontera conceptual, gracias a lo cual son accesibles directamente desde el resto del universo, tanto para entrada como para salida de información.

Finalmente, el dominio externo y el interno se agrupan en la clase dominio, que constituye el grado máximo de generalidad en la jerarquía.

Nótese que en el árbol jerárquico de la figura 3.30, el árbol de clases pertenece al dominio de aplicación Tráfico Urbano, mientras que las instancias son las que realmente representan los objetos significativos que componen el sistema actual.

De acuerdo con el Capítulo II, la representación orientada a objetos correspondiente a la figura 3.31 es la siguiente:

```
% DOMINIO DE APLICACION

clase(dominio).

clase(dominio_externo,[dominio]).
clase(dominio_interno,[dominio]).

clase(entradas,[dominio_externo]).
clase(salidas,[dominio_externo]).
clase(controladores,[dominio_externo]).
```

```
clase(calles,[dominio_interno]).
clase(cruces,[dominio_interno]).

% DESCRIPCION DEL SISTEMA

instancia(e1,entradas).
instancia(e2,entradas).
instancia(s1,salidas).
instancia(s2,salidas).
instancia(l1,calles).
instancia(l2,calles).
instancia(l3,calles).
instancia(l4,calles).
instancia(ifis1,cruces).
instancia(icon1,controladores).
```

III.2.2.2 Descomposición Orientada a Relaciones

Una vez tenemos descompuesto el sistema en sus objetos, el siguiente paso es el de realizar una descomposición del sistema orientada a relaciones, expresable mediante un árbol de clases. Recordar que entendemos que existe una relación entre dos objetos cuando el comportamiento de uno de ellos puede afectar al del otro directamente, sin la intervención de un tercer objeto, y que estas relaciones suelen repetirse entre distintos objetos pero con idénticas características. Esto es lo que sugiere ordenarlas según un árbol jerárquico de relaciones, que para el caso actual quedaría en la forma:

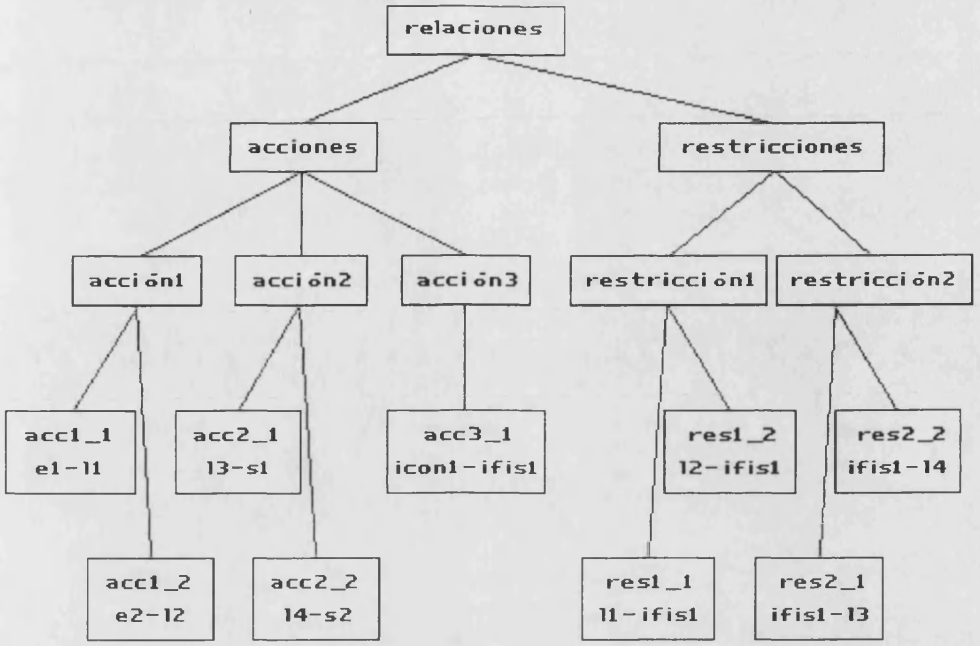

Figura 3.32

III.2.2.2.1 Representación Declarativa en Prolog

En el análisis del sistema hemos identificado cinco relaciones básicas, que llamaremos acción1, acción2, acción3, restricción1 y restricción2.

Las cinco clases de relaciones son binarias (afectan sólo a dos objetos), estando definida cada una de ellas por las clases de objetos que puede relacionar. Por ejemplo, que la clase acción1 relacione las clases de objetos entradas con las clases de objetos calles significa que todas sus instancias relacionan objetos-entradas con objetos-clases; las siguientes cláusulas Prolog corresponden a dos instancias de la clase acción1, "acc1_1" y "acc1_2", donde la primera relaciona la entrada "e1" con la calle "l1", y la segunda relaciona la entrada "e2" con la calle "l2":

```
acc1_1(e1,l1)
acc1_2(e2,l2)
```

Las instancias de estas cinco clases forman todas las relaciones del sistema. A su vez, como sabemos, estas cinco clases pueden clasificarse en dos grupos: **acciones** y **restricciones**.

Agrupamos en la clase acciones a todas aquellas relaciones en las que un objeto afecte a otro, pero no se dé la relación inversa. Por ejemplo, el controlador de intersección afecta al cruce (a través de los semáforos), pero el cruce no afecta al controlador de intersección.

En la clase restricciones se encuentran todas aquellas relaciones en las que hay bidireccionalidad. O sea, que un objeto afecte a otro y viceversa. Por ejemplo, la relación "acción1", entre las clases de objetos-entradas y las clases de objetos-calles, es bidireccional porque la falta de tráfico en la entrada puede afectar a la calle, mientras que la congestión en la calle puede afectar a la entrada, por lo que tenemos bidireccionalidad en la propagación de estados de tráfico.

La representación de este árbol en (λ,t) queda de la forma:

```
% DOMINIO DE APLICACION
clase(relaciones).

clase(acciones,[relaciones]).
clase(restricciones,[relaciones]).

clase(acción1,[acciones]).
atributo(acción1,objeto1,entradas).
atributo(acción1,objeto2,calles).

clase(acción2,[acciones]).
atributo(acción2,objeto1,calles).
atributo(acción2,objeto2,salidas).

clase(acción3,[acciones]).
atributo(acción3,objeto1,controladores).
atributo(acción3,objeto2,cruces).

clase(restricción1.[restricciones]).
atributo(restricción1,objeto1,calles).
atributo(restricción1,objeto2,cruces).

clase(restricción2.[restricciones]).
atributo(restricción2,objeto1,cruces).
```

```
atributo(restricción2,objeto2,calles).

% DESCRIPCION DEL SISTEMA

instancia(acc1_1,acción1).
atributo(acc1_1,objeto1,e1).
atributo(acc1_1,objeto2,l1).

instancia(acc1_2,acción1).
atributo(acc1_2,objeto1,e2).
atributo(acc1_2,objeto2,l2).

instancia(acc2_1,acción2).
atributo(acc2_1,objeto1,l3).
atributo(acc2_1,objeto2,s1).

instancia(acc2_2,acción2).
atributo(acc2_2,objeto1,l4).
atributo(acc2_2,objeto2,s2).

instancia(acc3_1,acción3).
atributo(acc3_1,objeto1,icon1).
atributo(acc3_1,objeto2,ifis1).

instancia(res1_1,restricción1).
atributo(res1_1,objeto1,l1).
atributo(res1_1,objeto2,ifis1).

instancia(res1_2,restricción1).
atributo(res1_2,objeto1,l2).
atributo(res1_2,objeto2,ifis1).

instancia(res2_1,restricción2).
atributo(res2_1,objeto1,ifis1).
atributo(res2_1,objeto2,l3).

instancia(res2_2,restricción2).
atributo(res2_2,objeto1,ifis1).
```

atributo(res2_2,objeto2,l4).

Una vez expresada la estructura del sistema, podemos estudiar sus variables de estado.

III.2.2.3 Descomposición de Parámetros

Una variable se caracteriza porque los valores que toma varian o pueden variar en el tiempo, por lo que la variable se representa siempre como una función del tiempo.

III.2.2.3.1 Tipos de Variables

Existen varios tipos de variables, según esté discretizado o no el tiempo y según esté discretizado o no el espacio de llegada.

1) Variables discretas: son aquellas funciones del tiempo sobre un espacio que es discreto por naturaleza. Por ejemplo, en el sistema dado tenemos dos semáforos que necesitan sendas variables discretas. Estas variables tienen como espacio imagen el {rojo,verde} (Puesto que el ámbar puede tomarse como verde, que por otra parte es lo que desgraciadamente sucede en muchas de nuestras ciudades). Este espacio imagen no es resultado de discretizar un espacio continuo, sino que es discreto por naturaleza[1]. Las variables discretas que definen los semáforos son del tipo:

F:R --> {rojo,verde}

donde R representa el tiempo de forma continua.

2) Variables continuas: Son funciones del tiempo sobre la recta real. Son por tanto del tipo:

F:R --> R

donde la primera R representa el tiempo, y la segunda la variable de que se trate. Por ejemplo, la función densidad de tráfico en una entrada al sistema sería una variable continua.

---

[1] Nótese que es posible tener dominios que son contínuos pero que interesa tratarlos de manera discreta, puesto que su manejo en forma contínua no es necesario para los objetivos de la aplicación. Es conveniente diferenciar estos casos en que una variable contínua es discretizada por necesidad del modelo, y aquellos en que la variable es realmente de naturaleza discreta.

3) Variable continua bidimensional: Son funciones del tiempo y de otra variable sobre la recta real. Son por lo tanto del tipo:

F:R x R --> R

Donde la primera R representa el tiempo, la segunda otra dimensión cualquiera, y la tercera la variable de que se trate. Por ejemplo, la función d(x,t) sería una variable continua bidimensional, donde la otra dimensión sería el espacio.

Para representar un sistema también hacen falta constantes, que pueden considerarse conceptualmente como un caso especial de variables (constante es una variable que no varia con el tiempo).

Teniendo en cuenta las observaciones anteriores, podemos hacer un árbol de clases para ordenar los distintos tipos de variables de estado que se utilizan en la representación del sistema. Bastan cuatro clases:

a) Clase de constantes (una constante la representaremos por la letra: "a").

b) Clase de las variables F:R -> R (funciones continuas del tiempo, que representaremos por la letra "f"); estas variables tienen su comportamiento temporal completamente descrito en el momento de representar el sistema, pues representan condiciones de contorno del mismo que no pueden ser deducidas. En nuestro sistema, estas variables son las que representan las afluencias de coches por las entradas a la red (obsérvese que se han de conocer necesariamente para poder controlar al sistema).

Para representar una función continua de una forma que permita razonar con ella, la técnica que utilizaremos será discretizar el dominio de la función; p.ej. podemos escojer el dominio siguiente:

Discreto	Continuo
d1	[0,0]
d2	(0,0.01]
d3	(0.01,0.03]
d4	(0.03,0.1]
d5	(0.1,0.15]
d6	(0.15,0.2]
d7	[0.2,0.2]

Tabla 3.33

En nuestro escenario, un ejemplo de este tipo de variables es la densidad de tráfico. Asumiendo que un vehículo mide 5 metros (es la asunción usual en tráfico urbano, considerando el espacio que debe dejar libre por adelante y por detrás), la densidad de trafico en coches por metro y por carril es una variable continua que toma valor en [0,0.2], y podemos definir un dominio discreto que cubra este intervalo. Por tanto, al tener una función del tipo F:R -> {d1,d2,...,d7}, ya podemos representarla de forma computacional de la manera que puede verse en la siguiente figura:

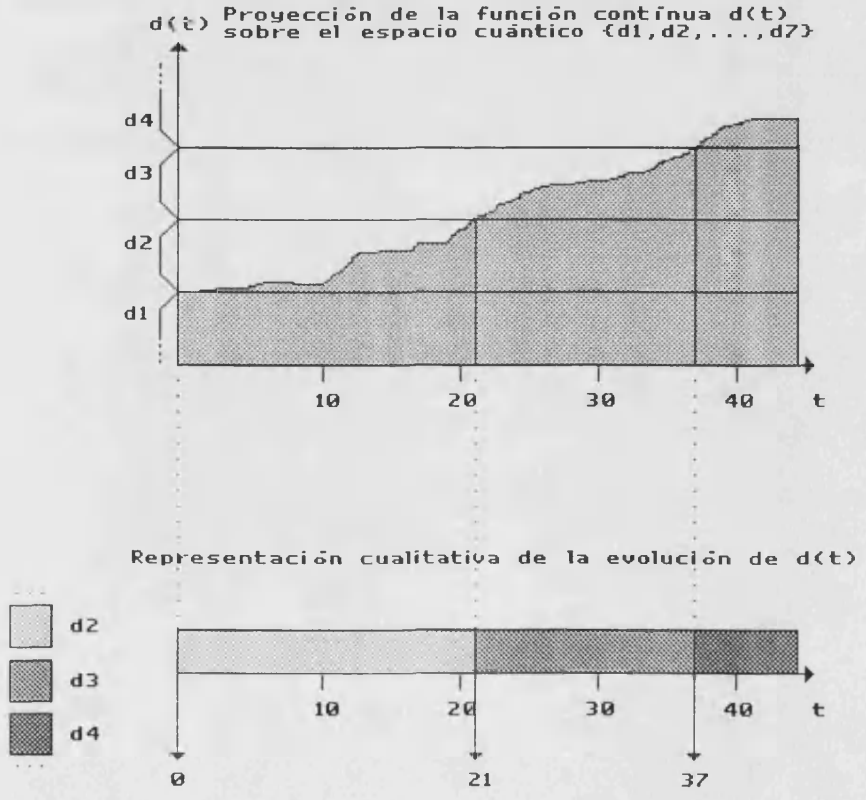

Figura 3.34

En resumen, la representación de la figura queda como:

event(cambio_a(d2),0).
event(cambio_a(d3),21).
event(cambio_a(d4),37).

considerando como hecho significativo cada cambio de valor de la variable, y almacenando un event por cada hecho significativo, tenemos ya toda la conducta temporal de la variable como un numero finito de acontecimientos perfectamente computables.

El valor de la función entre dos acontecimientos contiguos es el valor del event inferior del intervalo, debido a la evolución continua de la función (no puede cambiar de valor en un intervalo porque si lo hiciera los dos events ya no serían contiguos: el cambio de valor sería un nuevo event).

c) Clase de variables del tipo F:R -> R que representaremos por "ft", y que son similares a las del tipo f, pero con la diferencia de que representan variables cuyo valor no es necesario fijar de antemano. Su comportamiento temporal va siendo deducido del resto de la información.

d) Clase de variables del tipo F:R x R -> R que representaremos por "flt", y que llamaremos variables (λ,t). Estas variables toman valor en un plano continuo, uno de cuyos ejes es el tiempo y el otro una coordenada cualquiera (en nuestro caso la longitud de la calle), ambos necesarios para describir el estado de las calles, según hemos justificado en la sección III.2.1 (ver figura 3.35):

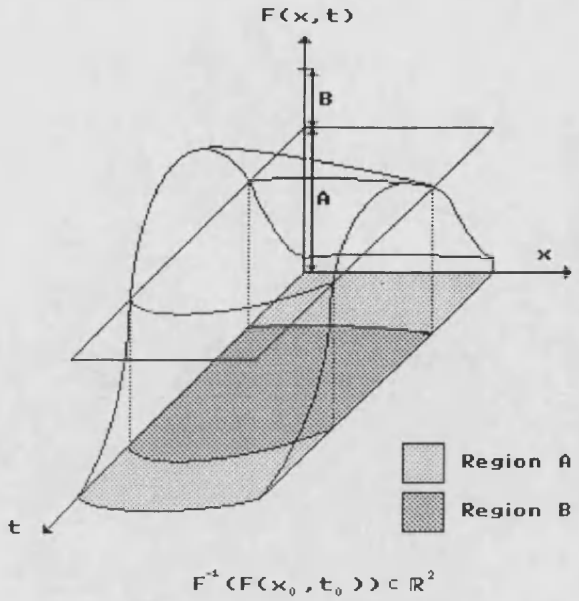

Figura 3.35

El espacio de llegada de este tipo de variables lo discretizaremos en la línea conceptual seguida por la teoría de razonamiento cualitativo. Así, estas funciones quedan del tipo R: R x R -> $\{a_1,...,a_n\}_e$ con ello, el plano origen queda dividido en regiones de igual valor de $F(\lambda,t)$, donde la evolución interna de la función es homogénea y por tanto las fronteras entre regiones son líneas rectas.

Un ejemplo de este tipo de funciones puede verse en la siguiente figura:

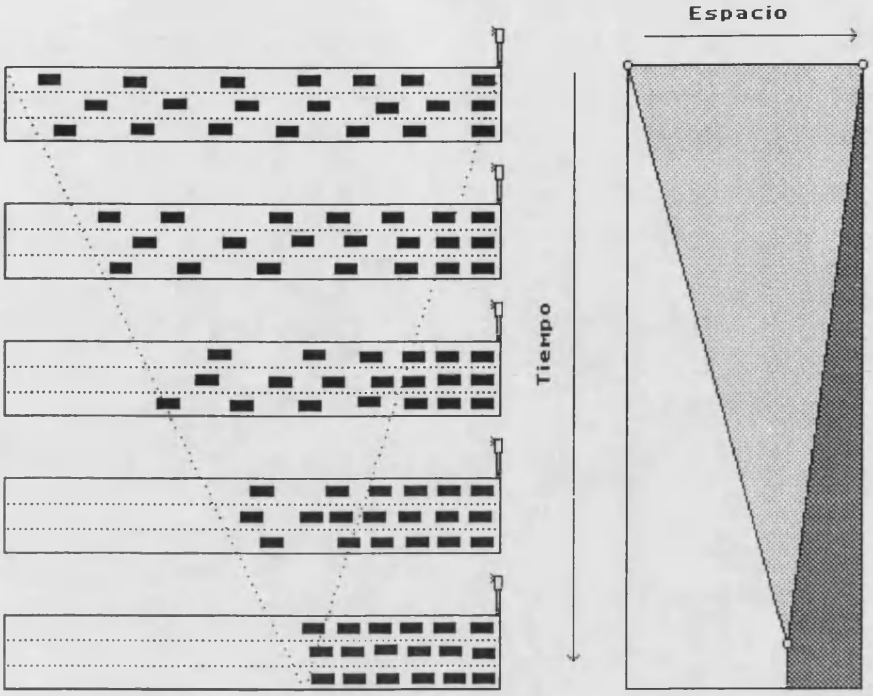

Figura 3.36

El principal problema a resolver en la definición de la flt, es la discretización a hacer.

En la representación de la figura 3.36 puede verse la economía en la representación de la densidad de vehículos en la calle. Lo que se ve en la imagen no es más que el crecimiento de una cola en una calle debido al semáforo que se pone en rojo en t=0 (parte de arriba de la figura). El plano completo es representado por tres events y el estado de la calle en t=0.

Razones computacionales aconsejan agrupar los events según el tiempo en que suceden, con el objeto de poder utilizar el cálculo de acontecimientos para cambio continuo de Shanahan [Shanahan,89]. En este caso los dos primeros events quedan agrupados en uno solo y el último en otro, con lo que la representación en Prolog será del tipo:

```
event(cambio_a([[d1,0,0],[d4,0,100],[stop,100,100]]),0).
event(cambio_a([[d1,0,70],[stop,70,100]],13).
```

El primer event, ocurrido en tiempo cero, indica que la calle se divide en tres intervalos, de principio a fin son: d1 (calle vacia) de 0 metros, d4 (coches en marcha) de 100 metros y stop (cola) de 0 metros.

El segundo event muestra el siguiente cambio significativo ocurrido en t = 13, donde ha desaparecido el intervalo de coches en movimiento, quedándose un estado d1 (calle vacía) de 70 metros y otro de stop (cola) de 30 metros.

El estado de la calle en todos los instantes intermedios puede calcularse con los predicados de Shanahan para propiedades que cambian contínuamente en el tiempo, puesto que en el intervalo entre ambos events el estado de la calle es el mismo que en el primer event, variando continuamente las fronteras de los intervalos.

Una vez hemos visto las cuatro clases de variables que vamos a utilizar (a, f, ft, flt), pasemos a estudiar las variables que están asociadas a la dinámica de cada objeto:

- objetos entradas: Su comportamiento es describible con dos variables:

1) f:densidadf es un parámetro que indica la densidad de tráfico prevista en una entrada como dato previo.
2) ft:densidad es un parámetro que indica la densidad real de tráfico existente en la entrada (Esta densidad no tiene por qué coincidir con la anterior, pues si la red urbana se encuentra congestionada, la densidad real superará a la prevista).

Realmente el estado de la entrada lo da el parámetro ft:densidad, y es dependiente tanto de la afluencia de vehículos como del estado de congestión de la calle. La afluencia de vehículos viene dada por f:densidadf, que representa la densidad de vehículos que existiría si la calle no interfiriera la circulación (por ejemplo, colas), y la ft:densidad se calcula a partir de f:densidadf y del estado de la calle.

- objetos salidas: Su comportamiento es describible con una variable:

1) ft:densidad es un parámetro que indica la densidad de tráfico existente en la salida.

El estado de la salida es fácil de calcular, pues aunque debiera depender de la afluencia de vehículos por la calle y del estado de congestión existente en el punto del resto del universo al que la salida va conectada, tomamos la suposición de que el

resto del universo no opone resistencia alguna a la salida de vehículos del sistema, por lo que el estado de la salida depende simplemente del estado de la calle a la que pertenece.

- objetos calles: Su comportamiento es describible con tres variables:

1) flt:densidad es un parámetro bidimensional que representa la densidad de trafico en cada punto de la calle y en cada momento.
2) a:carriles es una constante que indica el número de carriles de una calle
3) a:longitud es otra constante que indica la longitud de la calle en metros.

- objetos controladores: Su comportamiento es describible con un número de variables igual al número de semáforos de las calles de entrada al cruce que controlan:

1) f:($calle)semáforo son un conjunto de parámetros de tipo f que representan los semáforos de las calles de entrada conectadas al cruce. Hay un parámetro para cada calle de entrada al cruce que el controlador regula. Su dominio de valores es discreto por naturaleza, no es una variable continua discretizada por requisitos computacionales), y es {rojo,verde}.

En nuestro caso, las variables de este tipo son 11semáforo y 12semáforo, que significan, respectivamente, semáforo de la calle 1 (11) y semáforo de la calle 2 (12).

- objetos cruces: Su comportamiento es describible con un número de variables igual al número de calles del cruce, más el número de semáforos del mismo:

1) ft:($calle)densidad es un conjunto de parámetros, donde cada parámetro corresponde a una calle del cruce. Su valor es el valor de la densidad de tráfico en el punto por donde la calle se une al cruce.

2) ft:($calle)semáforo son un conjunto de parámetros de tipo ft que representan los semáforos de las calles de entrada al cruce. Tienen el mismo significado que las variables del controlador, sólo que son de naturaleza deducible (ft, no f).

Explicaré para qué sirven estos parámetros ordenandolos por las clases de objetos a los que pertenecen:

objetos entradas
 f :densidadf
 ft:densidad

objetos salidas
 ft:densidad

objetos calles
 flt:densidad
 a:carriles,longitud

objetos controladores
 f:[para toda calle de entrada al cruce ->
   ($calle)semáforo]

objetos cruces
 ft:[para toda calle de entrada -> ($calle)semáforo],
   [para toda calle -> ($calle)densidad]
 a:[para todo par calle entrada-calle salida ->
   ($calle ent)($calle sal)%]

En el escenario que estamos trabajando (ver figura 3.29), las variables a utilizar para representar el sistema son:

< formato clase_variable: objeto.variable,... >

f: e1.densidadf,e2.densidadf,
  icon1.l1semáforo,icon1.l2semáforo.

ft:e1.densidad,e2.densidad,
  ifis1.l1semáforo,ifis1.l2semáforo,
  ifis1.l1densidad,ifis1.l2densidad,
  ifis1.l3densidad,ifis1.l4densidad,
  s1.densidad,s2.densidad.

flt:l1.densidad,l2.densidad,
  l3.densidad,l4.densidad.

a:l1.carriles,l1.longitud,
l2.carriles,l2.longitud,
l3.carriles,l3.longitud,
l4.carriles,l4.longitud,
ifis1.1113%,
ifis1.1114%,
ifis1.1213%,
ifis1.1214%.

Si hacemos un repaso a lo hecho hasta ahora, tenemos tres taxonomías distintas: una de objetos, otra de relaciones entre objetos y una tercera de variables. Analizando sus interdependencias mutuas, vemos que la de objetos es completa, en el sentido de que no depende de las otras dos. La de relaciones está supeditada a la de objetos, pues para definir una relación necesitamos incluir las clases de objetos que relaciona; y la de variables depende de la de objetos tan íntimamente que las variables pueden ser consideradas como atributos de los objetos, con la salvedad de que además se ordenan en una jerarquía propia de variables (f,ft,flt,a) que les sirve para heredar propiedades dinámicas de los objetos (como veremos más adelante).

La representación completa del sistema propuesto utilizando el formalismo (λ,t) puede verse en el anexo A.

III.2.2.4 Comportamiento de Objetos

Siguiendo el capítulo II, el punto siguiente en la representación del conocimiento asociado al sistema considerado es la representación del comportamiento de las clases de objetos que componen el mismo. En el caso de nuestro escenario de Tráfico Urbano, puede verse una muestra más de la potencia del formalismo (λ,t) para la representación de dominios de aplicación con parámetros multidimensionales en el hecho de que la representación del comportamiento de las calles ya no es necesaria. De hecho, en este apartado, la única clase de objetos cuyo comportamiento se ha de representar es la intersección física.

En efecto, las entradas, salidas y controladores semafóricos no requieren representar conocimiento sobre su comportamiento interno, puesto que se trata de condiciones frontera al sistema. Las calles tampoco lo necesitan, puesto que su comportamiento interno es el de una función bidimensional, y será deducido automáticamente por el sistema a partir de la taxonomía de parámetros, donde se indica las características de

la función bidimensional citada (como por ejemplo, su tabla cualitativa de valores y pendientes).

Las intersecciones físicas son la única clase de objetos que requiere conocimiento para representar su comportamiento interno. Este conocimiento, en el caso de De Kleer, se expresaría por una serie de confluencias a nivel de objeto sin relación entre sí, y en el caso de Forbus, se agruparía bajo las etiquetas de los distintos procesos físicos involucrados en el comportamiento de cada objeto, pero sin considerar la modularidad por componentes típica de De Kleer; Kuipers, por último, considera el sistema como un todo y no divide las restricciones de comportamiento ni por componentes ni por procesos. En el formalismo (l,t) hemos utilizado los rasgos más importantes de cada uno de ellos; así, el conocimiento sobre el comportamiento se especifica a nivel de objeto, siguiendo la idea de De Kleer, pero internamente al objeto, se agrupa en relación a los procesos físicos que los originan, siguiendo a Forbus. De esta forma creamos una jerarquía de procesos interna a las clases de objetos, teniendo por tanto todas las ventajas de modularidad y herencia de los mismos, de los que Forbus carece.

En el caso de la intersección urbana, existen dos tipos de procesos que contribuyen al comportamiento interno de la misma: la conservación del número de vehículos y el comportamiento de los conductores. Ambos son de naturaleza totalmente distinta y no tendría sentido unirlos en una única representación del conocimiento sobre el comportamiento de una intersección. Por ejemplo, en el caso de niebla o lluvia, la conservación de vehículos y sus restricciones asociadas (flujo entrante en la intersección es igual al flujo saliente) no son alteradas, pero sí lo serían aquellas debidas al comportamiento de los conductores (salida más lenta de los semáforos y alteración de la curva de tráfico por variación de la relación entre la distancia de seguridad y la velocidad).

Por otro lado, el comportamiento de los conductores puede dividirse en dos tipos de conocimiento: el asociado a la curva de tráfico correspondiente (que, por ejemplo, variará en función de características climatológicas) y el asociado a la matriz origen-destino de los mismos, que determina los porcentajes de giro en el cruce.

En resumen, el conocimiento sobre el comportamiento interno de una intersección puede dividirse en tres partes distintas, según su naturaleza; cada una de ellas aporta un número de confluencias (restricciones cualitativas entre los valores de los parámetros del objeto, según De Kleer), y la unión de todas ellas forma el cuerpo

completo de conocimiento sobre el objeto considerado. Mostraremos seguidamente un ejemplo de cada una de ellas:

Parte 1: Conservación del número de vehículos

Suma de flujos de entrada = Suma de flujos de salida

Parte 2: Matriz Origen-Destino, parte_de Comportamiento de conductores:

Número de vehículos en salida(i) =

Suma de (porcentaje giro entrada(j)-salida(i))*(Número de vehículos en entrada(j)

Parte 3: Curva de Tráfico, parte de Comportamiento de conductores:

Relación cualitativa densidad-flujo en caso de tiempo normal.

Relación cualitativa densidad-flujo en caso de tiempo lluvioso.

...

III.2.2.5 Comportamiento de Relaciones

El último punto de la representación del conocimiento del sistema, siempre siguiendo el capítulo II, es la especificación del comportamiento de la relaciones entre objetos. Una relación entre objetos no tiene capacidad de proceso propia, y se limita a establecer una serie de restricciones de igualdad entre pares de parámetros de objetos distintos. Por comportamiento de relaciones, por tanto, entendemos el conjunto de igualaciones que una relación establece entre dos o más objetos.

En nuestro ejemplo, tendremos las siguientes igualaciones:

relación entrada(i)-calle(j):

densidad_entrada_(t) = densidad_calle(comienzo,t)

relación calle(i)-intersección(j):

densidad_calle(fin,t) = densidad_entrada_i(t)

relación intersección(i)-calle(j):

densidad_salida_j(t) = densidad_calle(comienzo,t)

relación calle(i)-salida(j):

densidad_calle(fin,t) = densidad_salida(t).

relación controlador_semaforico-intersección:

estado_semaforos(t) = estado_semaforos(t)

Donde la clase controlador aparece el listado de la Base de Conocimiento como icon y la de intersecciones como ifis.

III.3 APLICACION DEL FORMALISMO (λ,T) A SISTEMAS CON FLUJOS CONTINUOS

El paradigma de razonamiento cualitativo (λ,t) constituye una herramienta computacional de carácter general, susceptible de ser aplicada a cualquier dominio de aplicación que sea describible a través de funciones de una o dos dimensiones. Con objeto de ilustrar esta afirmación, hemos escogido un dominio de aplicación distinto al Control de Tráfico Urbano (CTU) que contiene procesos que cambian continua y linealmente y, por ende, no puede ser tratado sino con herramientas que incorporen un tratamiento para el cambio continuo, como el paradigma (λ,t).

III.3.1 Comportamiento de un fluido en un sistema de cubas: una aproximación cualitativa

El dominio de aplicación escogido para la demostración consiste en un sistema de cubas, como se muestra en la figura 3.37.

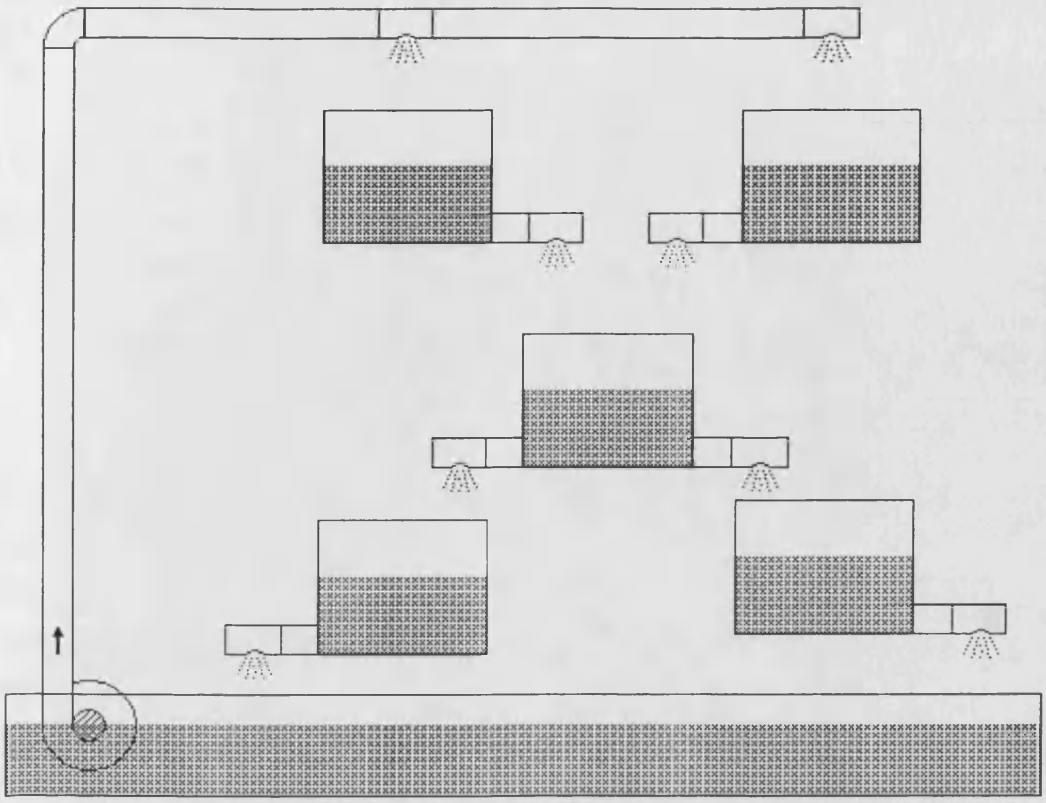

Figura 3.37

Este sistema consta de seis cubas y una bomba hidráulica. La bomba impulsa el agua de la cuba inferior hacia las dos cubas superiores, desde donde el agua va cayendo a través de las cubas restantes hasta retornar a la cuba inferior. A pesar de la aparente simplicidad del sistema, los cambios cualitativos de comportamiento motivados por el vaciado o rebose de las cubas introducen una complejidad en la evolución del sistema que lo hacen bastante adecuado para estudiar la aplicabilidad del paradigma (λ,t) a procesos de cambio continuo.

Podemos formular de una manera más rigurosa el comportamiento del sistema de la siguiente manera: El sistema se compone de un conjunto de cubas, de cañerías y de grifos. Supondremos que la velocidad de salida del líquido de una cuba es constante independientemente de la altura del líquido (como si la cuba tuviera un regulador de caudal en su desagüe). Si la cuba rebosa, consideraremos perdido el fluido sobrante. El estado de una cuba vendrá caracterizado por la masa de agua que contenga, cantidad que estará acotada entre cero y la capacidad total de la cuba. El estado de una cañería vendrá caracterizado por el caudal que ésta tenga, y el de un grifo por un espectro discreto de valores (cerrado,cuarto,medio,trescuartos,abierto). La bomba se

ignora al representar el sistema, como si fuera posible que de la cuba inferior pudiera caer agua sobre las cubas superiores.

Una vez formulado el problema, vemos que las ecuaciones que lo definen son lineales con respecto al tiempo, salvo en los momentos en que sucede algún acontecimiento significativo, en estos momentos, el sistema pasa a comportarse bruscamente de acuerdo a un sistema distinto de ecuaciones lineales.

III.3.2 Representación del Sistema de Cubas con el Formalismo (λ,t).

El primer paso es caracterizar las variables dinámicas que aparecen en el problema. Aparecen tres tipos: El estado de apertura o cierre de los grifos, el flujo que circula por las cañerías y la masa de fluido en cada cuba; de los tres, el más difícil de representar es la masa de fluido en cada cuba, debido a que varía contínuamente aunque las demás variables no lo hagan (esto se debe a que esta variable se relaciona con las demás a través de una relación integral). Para representarlo, utilizaremos una variable de tipo F(λ,t) con dos estados: Presencia o Ausencia de fluido, y en la que la pendiente de la frontera entre estados esté ligada al flujo total resultante sobre la cuba (suma de flujos entrantes menos suma de flujos salientes). El aspecto de este parámetro será el que muestra la figura 3.38.

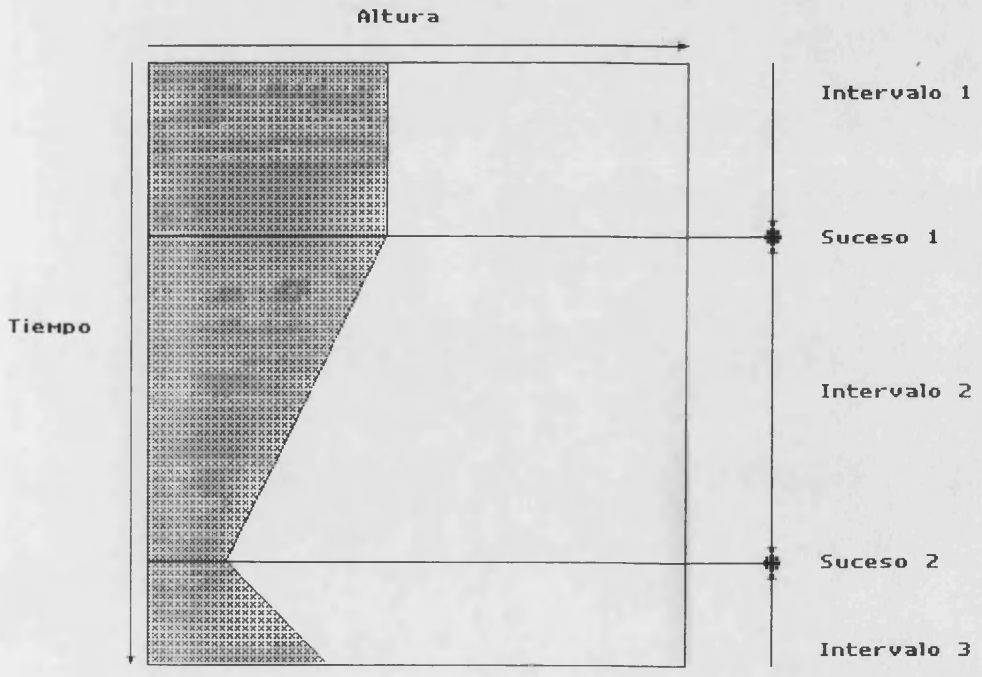

Figura 3.38

En la figura podemos ver la evolución temporal de la masa de agua contenida en una cuba. El eje vertical es el tiempo, mientras que el eje horizontal representa la masa de agua contenida en la cuba. La zona sombreada representa el fluido, y puede verse cómo su movimiento es lineal en los tres intervalos representados, cambiando únicamente como respuesta a los sucesos externos, de los que se representan dos.

La interpretación de la figura sería la siguiente: en el primer intervalo de tiempo, la cantidad de fluido contenido en la cuba permanece constante, hasta que sucede el suceso 1. Este suceso provoca el descenso lineal de la cantidad de fluido contenido en la cuba, durante el intervalo 2. De seguir el proceso, la cuba se habría vaciado en un tiempo posterior, provocando un cambio sobre el resto del sistema, pero esto no llega a suceder debido a la ocurrencia del suceso 2, que provoca un llenado paulatino de la cuba, durante el intervalo 3.

Una vez hemos analizado la manera de representar el sistema, vamos a definir las variables concretas que caracterizan nuestro sistema.

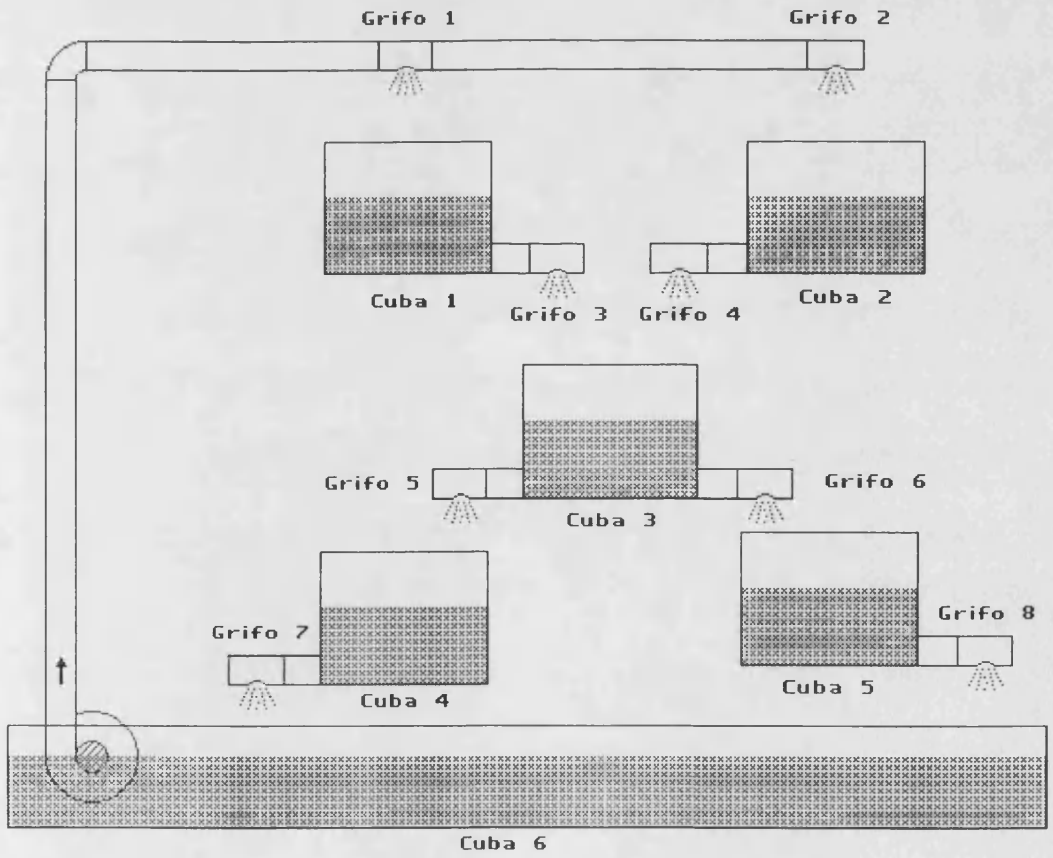

Figura 3.39

Como podemos ver en la figura 3.39, nuestro sistema tiene 6 cubas, 7 cañerías, 8 grifos y una bomba. A efectos de análisis del sistema, la bomba no necesita ser considerada, pues la gravedad no aparece en nuestro modelo del sistema y puede considerese que la cuba 6 desagua directamente sobre las cubas 1 y 2.

Supondremos ahora que las 6 cubas tienen la misma capacidad total, de 1000 litros, excepto la cuba 6, que supondremos de 10000 litros. Supondremos también que todas las cañerías tienen un flujo máximo de 100 litros por segundo.

Veamos ahora la base de conocimiento que describiría nuestro sistema:

III.3.2.1 Descomposición Orientada a Objetos

La primera tarea a considerar en la representación del sistema de cubas propuesto con el formalismo (λ,t) es, de forma análoga al caso del Tráfico Urbano, realizar una descomposición orientada a objetos del sistema propuesto (ver figura 3.39).

Las únicas clases de objetos relevantes que pueden ser identificadas en el sistema de cubas, de acuerdo con el capítulo II, son las siguientes:

- cubas
- tuberías
- grifos

Donde la bomba no ha sido incluida porque la no inclusión de la gravedad en el modelo del sistema la hace innecesaria, como se dijo anteriormente.

La descomposición jerárquica orientada a objetos del sistema de cubas de la figura 3.39 puede verse en la figura 3.40:

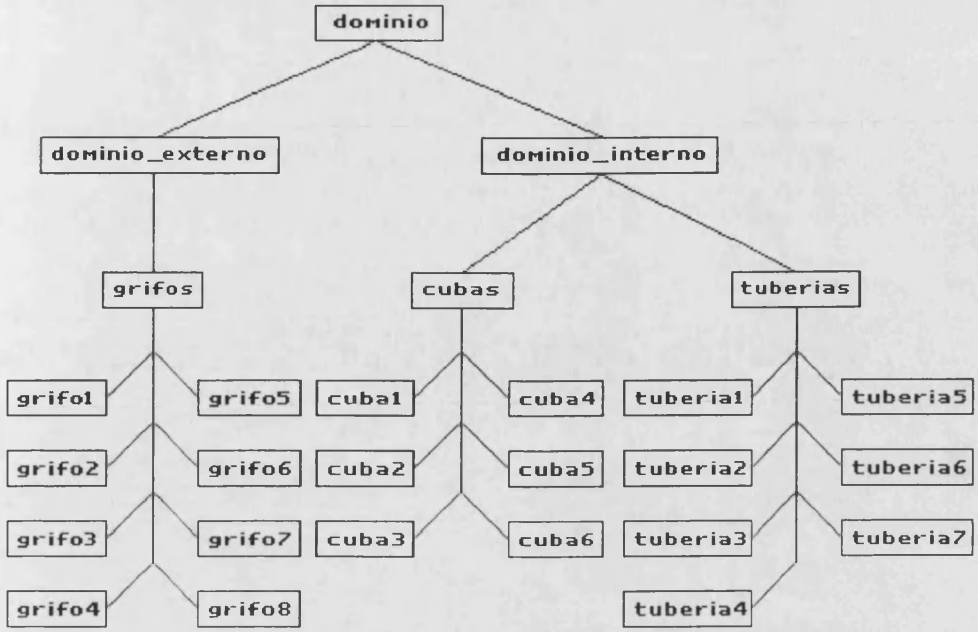

Figura 3.40

Las clases de objetos identificadas en la representación del conocimiento del escenario considerado han sido las cubas, las tuberías y los grifos. Los objetos que componen el sistema son instancias de estas clases, como puede observarse en el árbol de clases (ver figura 3.40). Estas clases pueden a su vez ser agrupadas, siguiendo el formalismo (I,t), en clases del **dominio externo** y clases del **dominio interno**. Recordemos que el sistema considerado podía ser considerado como una porción del universo, separada del resto por medio de una frontera conceptual. Definíamos entonces el dominio interno como la clase que agrupaba todas las clases de objetos cuyas instancias no tenían contacto con la frontera conceptual, y el dominio externo como el resto de clases, que sí mantienen contacto con la frontera conceptual. En nuestro caso, las clases cubas y tuberías pertenecen al dominio interno, puesto que su comportamiento está completamente definido y no interactúan con elementos externos al sistema, mientras que la clase grifos pertenece al dominio externo, puesto que sus instancias requieren contacto con algún ente externo que indique cuándo han de abrirse o cerrarse.

La representación orientada a objetos del sistema de cubas es la siguiente:

```
% OBJETOS

clase(dominio).
```

```
clase(dominio_interno,[dominio]).
clase(dominio_externo,[dominio]).

clase(tubería,[dominio_interno]).

instancia(tubería1,tubería).
instancia(tubería2,tubería).
instancia(tubería3,tubería).
instancia(tubería4,tubería).
instancia(tubería5,tubería).
instancia(tubería6,tubería).
instancia(tubería7,tubería).

clase(cuba,[dominio_interno]).

instancia(cuba1,cuba).
instancia(cuba2,cuba).
instancia(cuba3,cuba).
instancia(cuba4,cuba).
instancia(cuba5,cuba).
instancia(cuba6,cuba).

clase(grifo,[dominio_externo]).

instancia(grifo1,grifo).
instancia(grifo2,grifo).
instancia(grifo3,grifo).
instancia(grifo4,grifo).
instancia(grifo5,grifo).
instancia(grifo6,grifo).
instancia(grifo7,grifo).
instancia(grifo8,grifo).
```

El listado en Prolog de la representación orientada a objetos del sistema de cubas que acabamos de mostrar, se incluye como una parte de la base de conocimiento del sistema de cubas. Esta base de conocimiento, junto con el motor de inferencia del formalismo (λ,t), forman una implementación ejecutable del problema, que se

muestra en el anexo B. Con objeto de comprender mejor la parte ejecutable del formalismo, puede ser interesante buscar este listado dentro del programa del anexo B, pero se debe tener en cuenta que se ha cambiado el nombre de varios predicados con objeto de facilitar la escritura del programa. En el caso de la descomposición orientada a objetos, se ha cambiado clase/2 por c/2 e instancia/2 por i/2.

III.3.2.2 Descomposición Orientada a Relaciones

De acuerdo con el capítulo II, una vez descompuesto el sistema en sus objetos componentes, el siguiente paso consiste en realizar una descomposición del sistema orientada a relaciones. La jerarquía de relaciones así obtenida es expresable mediante un árbol de clases, de forma análoga a como se hizo en la sección III.2.2.2 para el escenario de Tráfico Urbano, sin embargo, en el caso del sistema de cubas, resulta demasiado compleja para expresarla de igual manera, por lo que nos limitaremos a analizar y mostrar su representación declarativa en Prolog.

III.3.2.2.1 Representación Declarativa en Prolog

En el análisis del sistema de cubas hemos identificado tres relaciones básicas, que llamaramos tubería_cuba, grifo_tubería y cuba_tubería. tubería_cuba representa el desagüe de una tubería en una cuba, grifo_tubería representa la relación entre un grifo y la tubería en la que está colocado, y cuba_tubería representa el desagüe de una cuba en una tubería.

Las tres clases de relaciones son binarias (afectan sólo a dos objetos), y están definidas por la pareja de clases de objetos que pretenden relacionar, de forma análoga a como ocurría en el apartado III.2.2.2.1.

Recordemos que las relaciones se dividían en acciones y restricciones. En este caso, tubería_cuba y grifo_cuba son acciones, y cuba_tubería es una restricción. La clase de relaciones tubería-cuba es una acción porque es unidireccional, esto es, porque la tubería puede verter en la cuba, pero la cuba no tiene forma directa de afectar a la tubería. De igual forma, la clase de relaciones grifo-tubería es una acción porque mientras que el grifo puede afectar a la tubería, ésta no puede afectar directamente al grifo. La última clase, cuba-tubería, es un poco más complicada, porque la relación es bidireccional: si la tubería tiene un grifo cerrado, impedirá desaguar a la cuba, y si la cuba se vacía, la tubería se verá afectada; por ello esta clase es una restricción.

UNIVERSITAT DE VALÈNCIA BIBLIOTECA DE CIÈNCIES

La representación de este árbol de restricciones queda en la forma:

```
% DOMINIO DE APLICACION

clase(relaciones).

clase(acciones,[relaciones]).
clase(restricciones,[relaciones]).

clase(tubería_cuba,[acciones]).
atributo(tubería_cuba,objeto1,tubería).
atributo(tubería_cuba,objeto2,cuba).

clase(grifo_tubería,[acciones]).
atributo(grifo_tubería,objeto1,grifo).
atributo(grifo_tubería,objeto2,tubería).

clase(cuba_tubería,[restricciones]).
atributo(cuba_tubería,objeto1,cuba).
atributo(cuba_tubería,objeto2,tubería).

% DESCRIPCION DEL SISTEMA

%  Instancias de la clase tubería_cuba

instancia(t1_c1,tubería_cuba).
atributo(t1_c1,objeto1,tubería1).
atributo(t1_c1,objeto2,cuba1).

instancia(t1_c2,tubería_cuba).
atributo(t1_c2,objeto1,tubería1).
atributo(t1_c2,objeto2,cuba2).

instancia(t2_c3,tubería_cuba).
atributo(t2_c3,objeto1,tubería2).
atributo(t2_c3,objeto2,cuba3).

instancia(t3_c3,tubería_cuba).
```

```
atributo(t3_c3,objeto1,tubería3).
atributo(t3_c3,objeto2,cuba3).

instancia(t4_c4,tubería_cuba).
atributo(t4_c4,objeto1,tubería4).
atributo(t4_c4,objeto2,cuba4).

instancia(t5_c5,tubería_cuba).
atributo(t5_c5,objeto1,tubería5).
atributo(t5_c5,objeto2,cuba5).

instancia(t6_c6,tubería_cuba).
atributo(t6_c6,objeto1,tubería6).
atributo(t6_c6,objeto2,cuba6).

instancia(t7_c6,tubería_cuba).
atributo(t7_c6,objeto1,tubería7).
atributo(t7_c6,objeto2,cuba6).

% Instancias de la clase grifo_tubería

instancia(g1_t1,tubería_cuba).
atributo(g1_t1,objeto1,grifo1).
atributo(g1_t1,objeto2,tubería1).

instancia(g2_t1,tubería_cuba).
atributo(g2_t1,objeto1,grifo2).
atributo(g2_t1,objeto2,tubería1).

instancia(g3_t2,tubería_cuba).
atributo(g3_t2,objeto1,grifo3).
atributo(g3_t2,objeto2,tubería2).

instancia(g4_t3,tubería_cuba).
atributo(g4_t3,objeto1,grifo4).
atributo(g4_t3,objeto2,tubería3).

instancia(g5_t4,tubería_cuba).
```

```
atributo(g5_t4,objeto1,grifo5).
atributo(g5_t4,objeto2,tubería4).

instancia(g6_t5,tubería_cuba).
atributo(g6_t5,objeto1,grifo6).
atributo(g6_t5,objeto2,tubería5).

instancia(g7_t6,tubería_cuba).
atributo(g7_t6,objeto1,grifo7).
atributo(g7_t6,objeto2,tubería6).

instancia(g8_t7,tubería_cuba).
atributo(g8_t7,objeto1,grifo8).
atributo(g8_t7,objeto2,tubería7).

% Instancias de la clase cuba_tubería

instancia(c1_t2,cuba_tubería).
atributo(c1_t2,objeto1,cuba1).
atributo(c1_t2,objeto2,tubería2).

instancia(c2_t3,cuba_tubería).
atributo(c2_t3,objeto1,cuba2).
atributo(c2_t3,objeto2,tubería3).

instancia(c3_t4,cuba_tubería).
atributo(c3_t4,objeto1,cuba3).
atributo(c3_t4,objeto2,tubería4).

instancia(c3_t5,cuba_tubería).
atributo(c3_t5,objeto1,cuba3).
atributo(c3_t5,objeto2,tubería5).

instancia(c4_t6,cuba_tubería).
atributo(c4_t6,objeto1,cuba4).
atributo(c4_t6,objeto2,tubería6).

instancia(c5_t7,cuba_tubería).
```

```
atributo(c5_t7,objeto1,cuba5).
atributo(c5_t7,objeto2,tubería7).

instancia(c6_t1,cuba_tubería).
atributo(c6_t1,objeto1,cuba6).
atributo(c6_t1,objeto2,tubería1).
```

El listado que acabamos de mostrar constituye la representación en Prolog de la jerarquía de relaciones del sistema de cubas, y es parte integrante de la Base de Conocimiento del mismo. Sin embargo, no está directamente representado el la implementación ejecutable del problema del anexo B. La razón consiste en que la Base de Conocimiento del programa del anexo B, que corresponde a las especificaciones del motor de inferencia definido en el capítulo VI, no sigue exactamente la estructura dada por el capítulo II, aunque contiene toda su información. En el caso que nos ocupa, la jerarquía de relaciones se ha subsumido en las reglas de comportamiento de relaciones, con objeto de facilitar la escritura del programa, pero perdiendo declarativiad. Un punto importante a aclarar aquí es que aunque puedan fundirse los predicados de la jerarquía de relaciones con los del comportamiento de las mismas, ambos son cosas distintas y no podemos, por tanto, eliminar ninguna de los dos del proceso de razonamiento.

La información de la jerarquía de relaciones está subsumida como parte del conocimiento de relaciones almacenado en el predicado mismo/2, así que daremos un ejemplo del mismo:

```
mismo(var(tubería1,salida_cuba1),var(cuba1,entrada_tubería1)).
```

Este predicado, que puede encontrarse en el programa del anexo B, no sólo indica la forma como la relación entre la tubería 1 y la cuba1 se lleva a cabo, sino que indica además la propia existencia de la relación, por tanto, está subsumiendo la instancia siguiente:

```
instancia(t1_c1,tuberia_cuba).
atributo(t1_c1,objeto1,tubería1).
atributo(t1_c1,objeto2,cuba1).
```

Esto ocurre igualmente con el resto de predicados mismo/2.

III.3.2.3 Descomposición de Parámetros

En el capítulo II, y más recientemente, en la sección III.2.2.3, se vieron con detalle los cuatro tipos distintos de variables que componían la jerarquía de parámetros, por lo que no repetiremos de nuevo estos conceptos; en cambio, pasaremos directamente a la clasificación de los parámetros del sistema identificados:

- objetos cubas: Su comportamiento se describe con un número de variables igual al número total de tuberías que empiezan o terminan en ellos, más dos variables propias.

1) ft:entrada_($cuba), son un conjunto de parámetros que representan los flujos de las cubas que vierten a la tubería.
2) ft:salida_($cuba), son un conjunto de parámetros que representan los flujos de las cubas que nacen de la tubería.
3) ft:presion: es un parámetro propio de la cuba, y representa la presión en el fondo de la misma.
4) flt:cantidad: es un parámetro bidimensional que representa la altura que alcanza el fluido en la cuba con respecto al tiempo.

- objetos tuberias: Su comportamiento viene descrito por las siguientes variables:

1) ft:entrada_($tuberia), son un conjunto de parámetros que representan los flujos de las tuberías que desaguan a la cuba.
2) ft:salida_($tuberia), son un conjunto de parámetros que representan los flujos de las tuberías que nacen de la cuba.
3) ft:presion_($tuberia), son un conjunto de parámetros que representan las presiones de entrada del fluido desde cada cuba.
4) ft:estado_($grifo), son un conjunto de parámetros que representan el estado de los grifos que actúan sobre la tubería.

- objetos grifos: Su comportamiento viene descrito por una única variable:

1) f:estado, es un parámetro que indica el estado del grifo, y cuya evolución temporal se introduce como paso previo al proceso de razonamiento.

Las variables citadas componen las clases de variables de la jerarquía de parámetros del sistema de cubas. El desarrollo completo de estas variables, así como el de

cualquier otro elemento de la Base de Conocimiento del problema de Cubas, puede encontrarse en el anexo B.

III.3.2.4 Comportamiento de Objetos

De acuerdo con el capítulo II, el punto siguiente de la representación del conocimiento sobre el sistema de cubas es la representación del comportamiento interno de las clases de objetos consideradas.

Las clases de objetos cubas y grifos no requieren esta representación, debido a que los grifos son condiciones de contorno al sistema considerado y a que el comportamiento de las cubas es el de su parámetro bidimensional cantidad, por lo que realiza automáticamente a partir de la taxonomía de parámetros.

La clase tuberías, sin embargo, requiere la especificación del conocimiento asociado a su comportamiento interno, lo que puede verse en la Base de Conocimiento del problema expresada en el anexo B. Esta clase requiere conocimiento sobre las relaciones físicas que existen entre el flujo y la presión en una tubería, sometida a la acción de uno o varios grifos y conduciendo agua de una o más cubas.

Para empezar, la cantidad de fluido en la cuba(s) que la alimenta determina el régimen de flujo de la tubería. Si la cuba(s) que la alimenta tiene cantidad no nula, se supone que la tubería tendrá un flujo igual a su flujo máximo teórico, en el caso de que todos sus grifos estén abiertos al 100%. En caso contrario, el flujo de salida será igual al que recibe la cuba, e inferior al anterior (si fuera superior, la cuba se llenaría lentamente y pasaríamos al caso anterior).

En uno u otro caso, tendremos que existe una presión de entrada en la tubería, y ninguna presión en la salida de la misma (o en las salidas, de ser varias). Entonces tendremos que el flujo de salida de fluido de la tubería por cada una de sus salidas (si tuviera más de una, como la tubería 1), depende de la diferencia de presión en los grifos, y esta depende del estado de los mismos. Unas ecuaciones hidrodinámicas simples describen por tanto el comportamiento de la tubería, y estas se encuentran codificadas por medio del predicado causa/4, que puede verse en el listado.

Un último detalle a tener en cuenta es que en la representación del comportamiento de las tuberías, mezclamos parámetros cualitativos con parámetros cuantitativos. Aunque el formalismo (λ,t) sea de razonamiento cualitativo, ello no le impide manejar ecuaciones cuantitativas, si no se ve objeto alguno en su cualificación. El

manejo de las mismas se realiza por medio de un rudimentario pero eficaz sistema de manejo de restricciones, que funciona de forma incremental hasta ajustar los valores de las variables por debajo de un mínimo de tolerancia especificado en el cumplimiento de las restricciones propuestas. Baste decir que de igual manera podríamos haber tratado el flujo cualitativamente, sustituyendo estas ecuaciones por confluencias hidrodinámicas.

III.3.2.5 Comportamiento de Relaciones

El último punto de la representación del conocimiento del sistema es, acorde siempre al capítulo II, la representación del conocimiento sobre el comportamiento de relaciones. Recordemos que esta representación consiste en la especificación exacta de las restricciones de igualdad que las relaciones imponen entre sendas variables de objetos relacionados.

Estas restricciones pueden verse expresadas en el listado completo del programa del anexo B en el predicado mismo/2, de la forma siguiente:

```
mismo(var(tubería1,salida_cuba1),var(cuba1,entrada_tubería1)).
```

Este predicado indica que la variable salida_cuba1 del objeto tubería1 ha de ser igual a la variable entrada_tubería1 del objeto cuba1, y es una de las restricciones que componen la relación tubería1-cuba1 (en este caso, es la única, pero no tiene porqué serlo, como por ejemplo la relación cuba6-tubería1, compuesta de una restricción de flujo, y otra de presión).

III.4 CONCLUSION

La conclusión principal que podemos extraer del presente capítulo es la adecuación del formalismo (λ,t) a los dos diferentes dominios de aplicación presentados en este capítulo. La claridad de representación de ambos dominios, junto con la elevada velocidad obtenida en la ejecución del motor de inferencia sobre ambos casos de estudio, confirman la generalidad del formalismo para la representación de sistemas multidimensionales y hacen pensar que puede constituir la base para desarrollar la arquitectura funcional para sistemas de control que fue propuesta en el capítulo I.

Capítulo IV

ABSTRACTOR DE ACONTECIMIENTOS. SU APLICACION A SISTEMAS CON FLUJOS DISCRETOS Y CON FLUJOS CONTINUOS

IV.1. INTRODUCCION

En los capítulos anteriores hemos presentado el formalismo (λ,t) y hemos estudiado su aplicación a un ejemplo con flujos discretos y a otro con flujos continuos. En ambos casos el motor de inferencia utilizado, creaba una Base de Datos (BD) que contenía toda información deducida a partir del modelo. Esta BD presenta un cierto nivel de agregación, en el sentido de que está formada por acontecimientos que llevan asociado el tiempo, en el que la dinámica de los parámetros que representan el sistema presenta un cambio cualitativo. Este nivel de agregación es la principal diferencia (en lo que a resultados se refiere) que nuestro modelo tiene respecto a la utilización de modelos cuantitativos tradicionales, al tiempo que es la principal fuente de la rapidez y expresividad del proceso de razonamiento. Es importante hacer notar que la utilización del formalismo (λ,t) dentro de la arquitectura de control presentada en el capítulo 1, exige que los resultados del proceso de razonamiento tengan un nivel de agregación superior al registrado en la Base de Datos, produciendo inferencias del mismo nivel cognitivo que las que realizaría un humano. Así, por ejemplo, en el caso de la aplicación al Tráfico Urbano no estamos interesados en los acontecimientos de bajo nivel generados por el formalismo (λ,t), sino en procesos de más alto nivel como puede ser el saber si un cruce queda colapsado, o si un itinerario está congestionado, o si se ha producido un incidente, etc.

En el presente capítulo presentamos un módulo, llamado Abstractor de Acontecimientos (AA), que permitirá razonar en términos cognitivos. Para ello, el AA tendrá la misión de analizar los acontecimientos generados por el formalismo (λ,t) para deducir acontecimientos de "alto nivel" (en adelante los llamaremos situaciones problemáticas). Este análisis se llevará a cabo por medio de reglas que puedan implicar razonamiento temporal y representaciones cualitativas de los objetos a distintos niveles de granularidad.

En el caso de Control de Tráfico, el objetivo del AA no es tan simple como pudiera parecer a primera vista, debido a la definición "borrosa" de incidente, que está normalmente más relacionada al conocimiento sobre el dominio de los expertos de control humanos que al propio Mundo Real (MR). El concepto de situación problemática se define como una perturbación concreta del funcionamiento normal del MR, y el concepto de funcionamiento normal del MR se caracteriza por los resultados que los humanos esperan del funcionamiento del MR. Por ello la definición de situación problemática puede ser considerada como un acontecimiento importante para los sistemas de control, pero no para el MR. Bajo este punto de vista. resulta más fácil comprender la complejidad del Abstractor de Acontecimientos, ya que por un lado debe manejar conocimiento físico del MR y por otro debe ser capaz de representar conocimiento del experto humano que constituye la mayor parte de la complejidad del AA.

Para representar el conocimiento y la definición de situaciones problemáticas en el AA, debe tenerse en cuenta la estructura de este conocimiento, ya que éste es normalmente cualitativo, e implica diferentes niveles de abstracción espacial y temporal. En otras palabras, varias situaciones problemáticas pueden ser definidas en regiones pequeñas del mundo real y con duraciones cortas, y otras pueden ser más grandes en el tiempo o en el espacio. Una situación problemática puede también estar contenida en otra situación problemática a un nivel superior de abstracción. Para representar este tipo de conocimiento, el AA debe ser capaz de manejar acontecimientos y propiedades cualitativas, y proveer un mecanismo temporal capaz de definir las situaciones problemáticas a partir de los acontecimientos producidos por el formalismo (λ,t).

A lo largo del capítulo, se presentan dos ejemplos para facilitar la comprensión de las funcionalidades del AA y de su aplicabilidad.

IV.2. NECESIDAD DEL ABSTRACTOR DE ACONTECIMIENTOS: GRANULARIDAD ESPACIAL Y TEMPORAL EN LA DETECCION DE SITUACIONES PROBLEMATICAS

Una buena forma de entender la necesidad del Abstractor de Acontecimientos es a través de un conjunto de ejemplos adecuado. Elegiremos los ejemplos de la aplicación del CTU ya que en ella es fácil escoger ejemplos intuitivos que pueden ilustrar muy bien tanto la representación granular como el razonamiento temporal.

Supongamos una intersección cualquiera. El objetivo que un experto de tráfico tiene en su cabeza acerca de una intersección aislada es tratar de mantener despejado el espacio crítico de la misma para asegurar un flujo de tráfico óptimo bajo distintas condiciones. Por ello una situación problemática puede definirse a este nivel como cualquier causa capaz de obstruir el espacio crítico. Por ejemplo, una cola bloqueando una intersección que no ha desaparecido en el momento en que cambia el semáforo, constituye una definición de situación problemática a nivel de intersección: Existe un incidente en una intersección dada si se satisface la condición anterior. La intensidad del incidente será proporcional al porcentaje de tiempo de ciclo en que esta situación se mantiene [1]. Es de notar la baja abstracción espacial de este incidente (implica una única intersección) junto con su abstracción temporal "media" (su duración puede ser considerada la de un ciclo semafórico completo, ya que todo él es perturbado por la situación problemática). Por otro lado, los acontecimientos del mundo real a partir de los que la situación problemática es detectada tienen un nivel de abstracción espacial y temporal muy bajo (Sobre 10 metros de anchura y 10 segundos de longitud). Puede observarse que la definición de esta situación problemática abstrae temporalmente un orden de magnitud, mientras que la noción de espacio desaparece completamente (la situación problemática sucede en la intersección, tomada como un todo).

Como segundo ejemplo de situación problemática, imaginemos un anillo cerrado compuesto por calles. Si alguna de estas calles se encuentra en un estado congestionado durante el tiempo suficiente para que la congestión se propague a través del anillo, éste puede convertirse en lo que se llama un "dead lock". En esta situación, cada calle del anillo está bloqueada por la anterior, por lo que la congestión no puede ser resuelta con facilidad. Esta situación requiere de un tremendo esfuerzo para ser resuelta, por ello es importante detectarla lo antes posible

[1]Tiempo de Ciclo: Periodo de tiempo que tardan los semáforos de la intersección en retornar al mismo estado.

para aplicar rápidamente las acciones de control más adecuadas. Sin embargo, el reconocer este incidente es una tarea ardua: implica tener en cuenta el estado de todas las calles del anillo, las entradas de tráfico al anillo y el estado de las posibles salidas del mismo. El nivel de abstracción espacial es muy alto, típicamente de varios centenares de metros. También lo es la abstracción temporal, debido a que los tiempos de reacción del anillo suelen ser de varios minutos como mínimo. Por supuesto, como en el caso anterior, estas abstracciones deben ser alcanzadas partiendo de los acontecimientos suministrados por el formalismo (λ,t), cuyo nivel de abstracción temporal promedio es de unos 10 segundos y el espacial de unos 10 metros.

Obsérvese que el segundo ejemplo implica un grado de abstracción superior al primero. En este caso no es de interés conocer que uno o varios cruces presentan una situación problemática, ya que todo ello es parte de una situación problemática de mayor nivel (el dead-lock).

Podrían estudiarse más ejemplos, pero todos ellos tienen en común la dificultad de ser observados a través de detectores aislados, y normalmente su detección requiere realizar un razonamiento de red sobre un gran conjunto de acontecimientos generados por el formalismo (λ,t), partiendo de una región grande de la ciudad durante grandes períodos de tiempo. Por ello puede concluirse que la definición de semejante clase de incidentes implica una gran cantidad de razonamiento temporal a varios niveles de granularidad, y es por tanto lo bastante complejo como para justificar el esfuerzo dedicado al AA.

IV.3. EL ABSTRACTOR DE ACONTECIMIENTOS

El Abstractor de Acontecimientos consiste en una herramienta de definición de situaciones problemáticas que es capaz de trabajar junto con el formalismo (λ,t). Como se dijo con anterioridad, la complejidad del AA viene de la borrosidad de la definición de situación problemática. La situación problemática es un concepto introducido por los expertos humanos de control para razonar acerca del dominio, pero este concepto no proviene del mundo real (MR) (así, el mismo hecho del MR puede tener diferentes significados para un experto dependiendo del dominio). El concepto de situación problemática puede definirse como una perturbación significativa y concreta del funcionamiento normal del MR, y este concepto de funcionamiento normal se define por los resultados que los humanos esperan del mismo. Los expertos humanos no sólo definen (comúnmente si darse cuenta) lo que

debe ser considerado como funcionamiento normal, sino también definen qué perturbaciones deben ser consideradas como situaciones problemáticas, qué perturbaciones deben ser descompuestas en varias situaciones problemáticas y qué perturbaciones deben ser ignoradas. En resumen, las definiciones de situación problemática deben ser consideradas como parte de la base de conocimiento del experto humano, por lo que es posible hablar de un conocimiento de definición de situaciones problemáticas.

Teniendo en cuenta lo expuesto en el párrafo anterior, podemos considerar al Abstractor de Acontecimientos como un sistema basado en el conocimiento que razona sobre el conocimiento de definición de la situación problemática, por lo que debe tener un motor de inferencia capaz de manejar adecuadamente esta tarea. Las particularidades de este conocimiento deben ser previamente discutidas para poder abordar las funcionalidades del AA.

El conocimiento de definición de situaciones problemáticas es una clase de conocimiento de lo que se conoce en la literatura como "Identification Problem" (problema de identificación), y que tiene en cuenta el tiempo. El AA deduce el dominio temporal de existencia de una situación problemática a partir de los acontecimientos del formalismo (λ,t) contenidos en una ventana espacio-temporal. Los acontecimientos de esta ventana son los hechos que se utilizarán para deducir el dominio temporal de existencia de acontecimientos de orden superior a través de elementos de conocimiento temporal, y estas deducciones se encadenan hasta determinar el dominio temporal de existencia del incidente, junto con la evolución temporal de su nivel de intensidad.

Veamos la forma general de un acontecimiento dado por el formalismo (λ,t):

aconteciemiento(Objeto,Tiempo,NuevoEstado)

El acontecimiento representa la presencia de un acontecimiento instantáneo (en una escala temporal de segundos) en el tiempo dado para un objeto determinado.

La base de datos temporal no contiene el dominio temporal de existencia de ninguna propiedad: Si éste es necesario, se deduce a partir de los acontecimientos en una forma similar a la del Cálculo de Acontecimientos [Kowalski,86]

El AA comienza a partir de estos acontecimientos instantáneos, y utilizará varios predicados del ECCC (Event Calculus for Continuous Change, (Cálculo de Acontecimientos para Cambio Continuo) [Shanahan,90]) para lograr su objetivo. La razón que hace necesario el ECCC es el comportamiento temporal de los parámetros $F(\lambda,t)$, y a continuación vamos a justificar su conveniencia:

Un parámetro $F(\lambda,t)$ es un parámetro cualitativo multidimensional. Es decir, un parámetro cualitativo que toma valor en un espacio multidimensional, una de cuyas dimensiones es el tiempo. Tiene la restricción de que todas las fronteras entre hiperregiones cualitativas son hiperplanos. En el caso del CTU, el único parámetro de tipo $F(\lambda,t)$, la densidad de vehículos, es bidimensional. Por esta razón las fronteras que separan los estados cualitativos son líneas. La evolución temporal de la densidad de vehículos consistirá en un conjunto de regiones de densidad que están variando continuamente de forma continua, hecho éste que no podemos representar por medio de sucesos convencionales. Representaremos como sucesos los puntos del tiempo en los que aparezca o desaparezca una región cualitativa, y deduciremos el estado de la densidad de vehículos en cualquier momento por medio de un predicado "trayectoria" [Shanahan,90]. Este predicado calcula el movimiento de las fronteras a partir de la representación (λ,t), en la que las fronteras se representan como líneas rectas. Esto es similar al ECCC, por lo que parece en principio buena idea utilizar los predicados del ECCC en el AA, con objeto de manejar los parámetros (λ,t) de forma adecuada.

El conocimiento utilizado por el Abstractor de Acontecimientos se basa pues en tres pilares:

a) En los sucesos instantáneos deducidos por el proceso de razonamiento sobre el comportamiento del sistema.
b) En los predicados "trayectoria" de cambio contínuo de los parámetros $F(\lambda,t)$.
c) En el ECCC.

Cada elemento de conocimiento del Abstractor de Acontecimientos se codifica en la forma de una cláusula prolog, cuyo antecedente se compone de una serie de subobjetivos que pueden incluir predicados del ECCC, sucesos instantáneos del proceso de razonamiento sobre el comportamiento de sistema, y cuyo consecuente es el dominio temporal de existencia de una propiedad.

El Abstractor de Acontecimientos provee un marco de trabajo para definir situaciones problemáticas. Podemos definir tres clases de situaciones problemáticas:

1. **Situaciones Problemáticas Elementales:** Situaciones Problemáticas definidas sobre un objeto simple en un momento del tiempo dado a partir de sucesos generados por el proceso de razonamiento cualitativo o de propiedades activas en ese momento del tiempo.
2. **Situaciones Problemáticas Temporales:** Situaciones Problemáticas definidas sobre un objeto simple durante un intervalo de tiempo obtenido de la abstracción temporal de varias situaciones problemáticas elementales, en momentos diferentes del tiempo por medio de reglas de razonamiento temporal.
3. **Situaciones Problemáticas Temporales Abstractas:** Situaciones Problemáticas definidas sobre un conjunto de objetos durante un intervalo de tiempo, obtenido de la abstracción temporal de varias situaciones problemáticas temporales en diferentes intervalos del tiempo por medio de reglas de razonamiento temporal.

A continuación estudiaremos con más detalle cada tipo de situación problemática:

IV.3.1 Situaciones Problemáticas Elementales

Las Situaciones Problemáticas Elementales son la clase más simple de situación problemática que puede definirse. Se componen de los siguientes elementos:

- **Nombre del Objeto:** El nombre del objeto sobre el que actúa la situación problemática.
- **Atributo del Objeto 1:**

  ...
- **Atributo de Objeto N:** Los atributos del objeto que son necesarios para deducir la situación problemática.
- **Restricción de Atributos 1**

  ...
- **Restricción de Atributos N:** Las Restricciones que los atributos deben satisfacer para deducir la presencia de la situación problemática.
- **Nivel de Importancia:** Un valor definido por el operador que representa la importancia de la situación problemática. Es utilizado por el Interfaz Hombre-Máquina para lograr una gestión óptima de avisos de situaciones problemáticas

(es importante no sobrecargar al operador con avisos no significativos, con objeto de no dificultar la comprensión de aquellos realmente importantes).

- **Nivel de Activación:** Representa el nivel de activación de la situación problemática. que se deduce a partir de los atributos del objeto a través de la satisfacción de las restricciones de los mismos.

Obsérvese que cuando la simple presencia de la situación problemática no es suficiente para realizar una caracterización adecuada de la situación, las restricciones pueden incluir varios grados cualitativos de intensidad, de los que podemos deducir un "nivel de activación". El nivel de activación representa la intensidad de la situación problemática en un momento dado del tiempo, y el procedimiento de inferencia calculará entonces el dominio temporal de existencia del nivel de activación.

El procedimiento de inferencia para esta clase de situaciones problemáticas es el siguiente: El dominio temporal de satisfacción de cada restricción de atributo se calcula a partir de la evolución temporal de cada atributo del objeto dado, y el dominio temporal de la presencia de la situación problemática se deduce realizando una operación lógica "Y" sobre los dominios temporales de satisfacción de todas las restricciones.

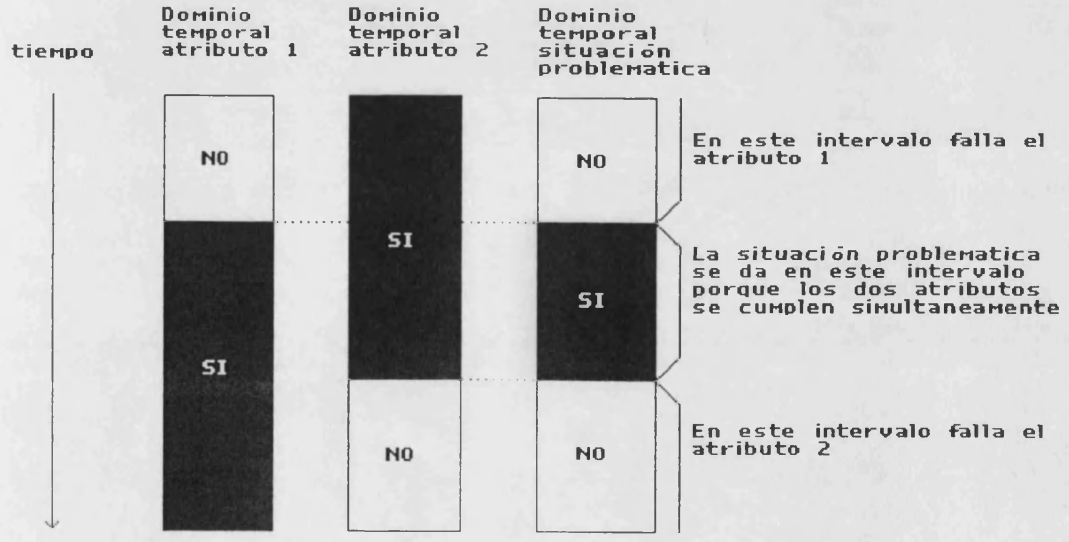

Figura 4.1

Puede verse que esta clase de situaciones problemáticas únicamente permite definir situaciones problemáticas cuyas precondiciones deban suceder en un único momento del tiempo. Si queremos definir situaciones problemáticas con precondiciones en diferentes momentos del tiempo, debemos utilizar las "Situaciones Problemáticas Temporales".

IV.3.2 Situaciones Problemáticas Temporales

Las Situaciones Problemáticas Temporales se definen sobre un único objeto a partir de valores de sus atributos tomados en diferentes momentos del tiempo, y a partir de situaciones problemáticas elementales. Se componen de los siguientes elementos:

- **Nombre del Objeto:** El nombre del objeto sobre el que actúa la situación problemática.
- **Atributo del Objeto 1:**

  ...
- **Atributo del Objeto N:** Los atributos del objeto que son necesarios para deducir la presencia de la situación problemática.
- **Incidente Elemental 1:**

  ...
- **Incidente Elemental M:** El nivel de activación de las situaciones problemáticas elementales que son necesarias para deducir la presencia de la situación problemática.
- **Restricción Temporal 1:**

  ...
- **Restricción Temporal N:** Las restricciones temporales que deben satisfacer los atributos y las situaciones problemáticas elementales para poder deducir la presencia de la situación problemática. Pueden incluir predicados temporales, sucesos y propiedades.
- **Nivel de Importancia:** Un valor definido por el operador que representa la importancia de la situación problemática.
- **Nivel de Activación:** Representa el nivel de activación de esta situación problemática, que se deduce a partir de los atributos y las situaciones problemáticas elementales a través de la satisfacción de las restricciones temporales.

El procedimiento de inferencia para esta clase de incidente es más complejo que en el caso anterior, debido a la introducción de las restricciones temporales. De todas

formas, la complejidad en el manejo temporal se solventa utilizando el ECCC. Las restricciones temporales permiten definir la presencia de situaciones problemáticas a partir de propiedades y sucesos en tiempos diferentes (por ejemplo, puede definirse que una situación problemática sucederá si cierto suceso ocurre después de que una propiedad determinada haya terminado).

IV.3.3 Situación Problemática Temporal Abstracta

La Situación Problemática Temporal Abstracta actúa sobre un conjunto de objetos, y se define a partir de los valores de sus atributos, de sus situaciones problemáticas elementales y de sus situaciones problemáticas temporales. Se compone de los siguientes elementos:

- **Nombre del Objeto 1:**

  ...

- **Nombre del Objeto K:** El nombre de los objetos sobre los que actúa la situación problemática.
- **Atributo del Objeto 1:**

  ...

- **Atributo del Objeto N:** Los atributos de los objetos que son necesarios para deducir la situación problemática.
- **Incidente Elemental 1:**

  ...

- **Incidente Elemental M:** El nivel de activación de las situaciones problemáticas elementales que sean necesarias para deducir la presencia de la situación problemática.
- **Incidente Temporal 1:**

  ...

- **Incidente Temporal O:** El nivel de activación de las situaciones problemáticas temporales que son necesarias para deducir la presencia de la situación problemática.
- **Restricción Temporal 1:**

  ...

- **Restricción Temporal N:** Las restricciones temporales que deben satisfacer los atributos, las situaciones problemáticas elementales y las situaciones problemáticas temporales para poder deducir la presencia de la situación problemática. Pueden incluir predicados temporales, sucesos y propiedades.

- **Nivel de Importancia:** Un valor definido por el operador que representa la importancia de la situación problemática.
- **Nivel de Activación:** Representa el nivel de activación de esta situación problemática, que se deduce de los atributos, de las situaciones problemáticas elementales y de las situaciones problemáticas temporales a través de la satisfacción de las restricciones temporales.

IV.4. REQUERIMIENTOS TEMPORALES DEL ABSTRACTOR DE ACONTECIMIENTOS

Como hemos mencionado en la sección anterior, el Abstractor de Acontecimientos utiliza los predicados extendidos del Cálculo de Acontecimientos provistos por el ECCC, debido a la presencia de los parámetros $F(\lambda,t)$. En particular, utiliza el predicado "trayectoria", definido a partir de las características de los parámetros $F(\lambda,t)$, el predicado generalizado "holds-at", para el cálculo de instantáneas, y el predicado de autoterminación, para tratar interacciones de objetos que contienen parámetros $F(\lambda,t)$.

El Abstractor de Acontecimientos también utiliza varios de los predicados del Cálculo de Acontecimientos. En particular los predicados "initiates", "terminates", "happens", "time", "broken_during", "holds_at", "broken_after", "holds_forever_from", "Mholds" e "Iholds_for". Las restricciones temporales también son necesarias para realizar una ordenación causal de sucesos instantáneos y propiedades (por ejemplo, T1 > T2 ó T3 = T4).

IV.5. EJEMPLOS DE APLICACION

Se han escogido cuatro ejemplos para ilustrar las funcionalidades del Abstractor de Acontecimientos. Estos ejemplos muestran la granularidad espacio-temporal del Abstractor de Acontecimientos y el alto nivel de abstracción de los sucesos deducidos. Los cuatro ejemplos proceden de la aplicación del CTU:

1) El primer ejemplo define el dominio temporal de la situación problemática "Bloqueo de cruce", que consiste en una situación problemática simple que puede definirse por medio de una única regla temporal:

```
congestion(Cruce,DT3) :-
        entradacruce(A,Cruce),
```

salidacruce(B,Cruce),
normal(Cruce,A,B),
cola_al_final(B,DT1),
semáforo(A,verde,DT2),
intersección(DT1,DT2,DT3).

La regla define el dominio temporal (DT3) de la situación problemática "Bloqueo de cruce" en un cruce dado. Esta situación problemática se deduce a partir de sus antecedentes, que explicaremos seguidamente:

- **"entradacruce"** y **"salidacruce"** son predicados "topológicos" que relacionan las calles y las intersecciones de una red urbana.
- **"normal"** es un predicado "topológico" que se utiliza para conocer si dos calles son "perpendiculares" entre sí (si el flujo de una calle cruza el flujo de la otra en fases semafóricas distintas).
- **"cola_al_final"** da el dominio temporal en el que se verifica que "La cola en la calle dada ha llegado hasta el final de la calle".
- **"semáforo"** da el dominio temporal de existencia de la luz verde al final de una calle dada,
- **"intersección"** genera un nuevo dominio temporal por intersección de los dos dominios temporales dados.

El significado de la regla completa, sería el siguiente: "Decimos que hay bloqueo en un cruce dado si hay una cola bloqueando el cruce cuando el semáforo de alguna calle perpendicular a la cola se encuentra verde". Los predicados **"entradacruce"** y **"salidacruce"** encuentran todos los pares calle de entrada - calle de salida del cruce (variables A y B), el predicado **"normal"** restringe estos pares a aquellos capaces de provocar un bloqueo; el predicado **"cola_al_final"** encuentra el dominio temporal de existencia del hecho "cola al final de la calle B", y el predicado **"semáforo"** encuentra el dominio temporal del hecho "semáforo de la calle A en verde". El predicado **"intersección"** encuentra el dominio temporal de la situación problemática "Bloqueo en cruce" como todos aquellos puntos del tiempo que satisfagan todas las restricciones sobre los atributos simultáneamente (en este caso dos: "cola_al_final" para B y "semáforo" para A). Una aclaración a hacer es que aunque pueda parecer que el dominio temporal de existencia de la situación problemática se deduzca comprobando la satisfacción de estas dos restricciones punto a punto a lo largo del eje temporal, en realidad esto no se hace así, sino que se

utilizan predicados del Cálculo de Acontecimientos para definir el dominio temporal de existencia de la situación problemática a partir de los puntos en que comienzan o terminan las propiedades "existencia de cola al final de B" y "semáforo en verde en A".

En la figura 4.2 puede verse el dominio temporal de existencia del bloqueo del cruce. El eje vertical de los tres rectángulos representa el tiempo. El rectángulo de la izquierda representa la evolución temporal de la densidad de tráfico en la calle B (calle 3 en la figura) y el rectángulo central representa la evolución temporal del estado del semáforo al final de la calle A (semáforo 1 en la figura). El rectángulo de la derecha representa el dominio temporal deducido del bloqueo en el cruce. Las líneas punteadas horizontales representan el predicado temporal **"intersección"**. Esta es una representación gráfica de la regla temporal que acabamos de analizar: puede verse que la presencia de la situación problemática tiene valor "SI" cuando el semáforo está verde y la cola está al final de la calle 3 simultáneamente.

Como puede observarse, este primer ejemplo corresponde al tipo "situación problemática elemental".

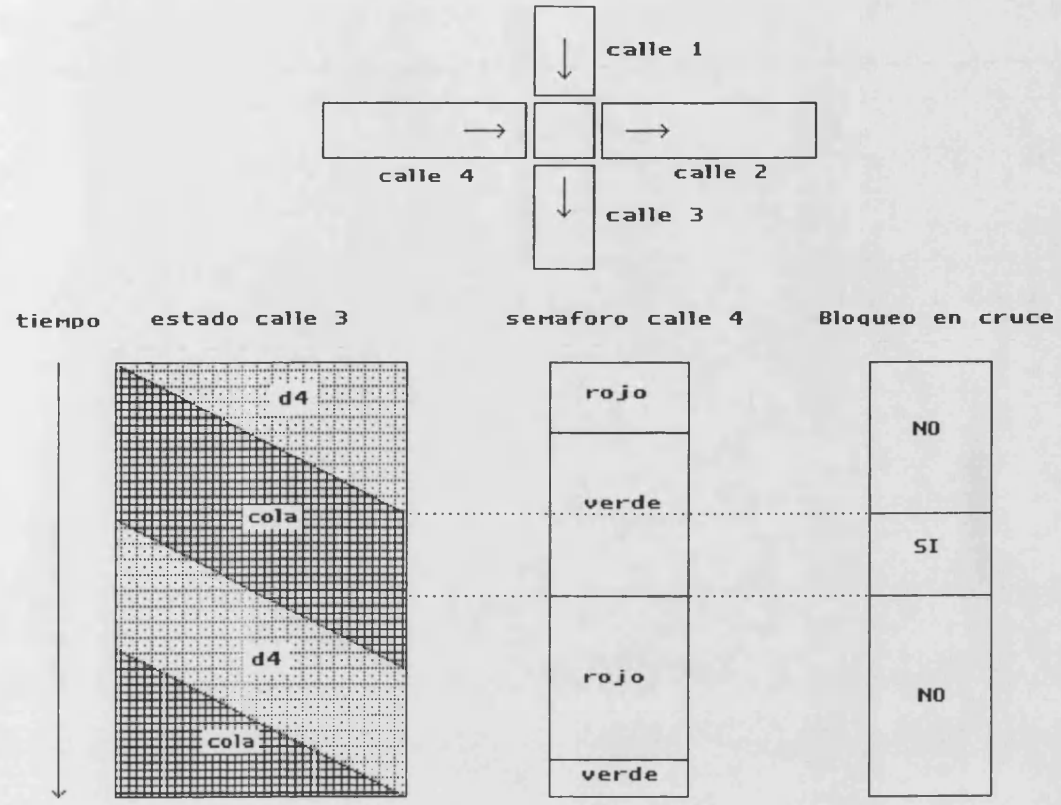

Figura 4.2

2) El segundo ejemplo que hemos elegido está muy relacionado con el primero. Su antecedente es la situación problemática elemental "Bloqueo en cruce" tal y como se ha definido en el primer ejemplo, y su conclusión consiste en la caracterización de la congestión en una escala temporal de 5 a 10 minutos. Se trata principalmente de un proceso de abstracción temporal. El proceso puede comprenderse perfectamente a partir de la figura 4.3. Como en el ejemplo anterior, el eje vertical de los tres rectángulos representa el tiempo. El rectángulo de la izquierda representa los ciclos de los semáforos del cruce: cada línea en el rectángulo representa el tiempo en el que comienza un nuevo ciclo. El rectángulo del centro representa la presencia de congestión tal y como se definió en el ejemplo anterior, donde el color negro representa "SI". El rectángulo de la derecha se deduce de los anteriores a través de la siguiente regla temporal: La caracterización de la congestión será "ninguna" (blanco) durante todos aquellos ciclos de semáforo en los que en ningún momento se ha presentado el incidente "Bloqueo en cruce". Será "media" (gris) en todos los ciclos de semáforo en los que el incidente "Bloqueo en cruce" se ha presentado en una parte del ciclo, y será "total" en todos aquellos ciclos de semáforo en los que haya presencia del incidente durante toda su duración.

Este segundo ejemplo corresponde al tipo "situación problemática temporal" cuya precondición es la "situación problemática elemental" definida en el primer ejemplo.

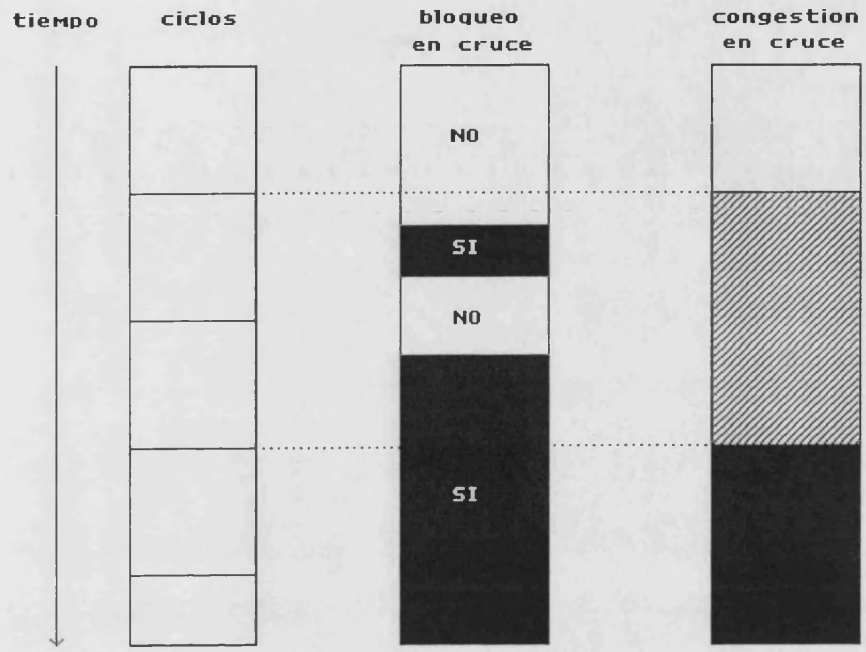

Figura 4.3

3) El tercer ejemplo no es exactamente una definición de situación problemática, sino parte de ella. Tan sólo deduce una abstracción temporal de las instrucciones de control de un cruce aislado, como primera fase para realizar un análisis más profundo del comportamiento de la intersección en altos niveles de granularidad temporal (sobre 10 minutos). El ejemplo se representa en la figura 4.4, en la que, como en los ejemplos anteriores, el eje vertical representa el tiempo. El rectángulo de la izquierda representa los periodos de verde (blanco) y de rojo (negro) de un semáforo. El rectángulo central se deduce a partir del primero, y representa el porcentaje de tiempo que el semáforo ha estado en verde durante un ciclo. Los ciclos son el período de tiempo que el semáforo tarda en repetir su comportamiento, y en la figura están separados por líneas discontínuas. El porcentaje de tiempo que el semáforo ha estado en verde durante un ciclo se conoce como "reparto", y aunque tiene infinitos valores por tratarse de un parámetro contínuo, en la práctica se utiliza un número pequeño de valores (en el ejemplo utilizaremos una escala desde s1 hasta s8, ordenados por tiempo de rojo). En la figura puede verse que los dos primeros ciclos del rectángulo de la izquierda son iguales, y corresponden a un porcentaje bajo de rojo, por lo que los caracterizaremos por un valor bajo de la escala cualitativa (s2

en el ejemplo). De igual forma, los tres últimos ciclos tienen un porcentaje de tiempo de rojo más elevado, por lo que los caracterizaremos por un valor alto de la escala (s5 en el ejemplo). Por consiguiente, el rectángulo central queda como cinco intervalos de tiempo, correspondientes a los cinco ciclos semafóricos, etiquetados con valores cualitativos representando el "reparto". El rectángulo de la derecha representa la evolución temporal del "reparto", y únicamente indica los momentos de cambio de reparto. Representa un concepto de abstracción temporal de varios minutos, y es de gran utilidad para razonar sobre incidentes de red.

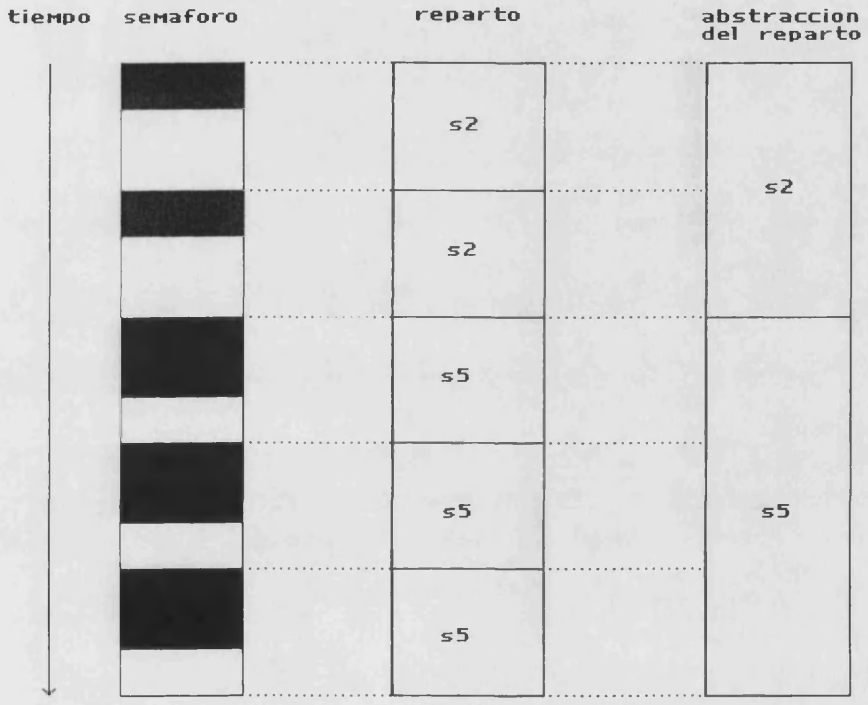

Figura 4.4

4) El cuarto ejemplo corresponde al tipo "situación problemática temporal abstracta", consiste en la definición de una "congestión circular" en un anillo de calles. Esta es una situación problemática que involucra varios objetos (las calles y cruces del anillo), y cuyas precondiciones son "situaciones problemáticas elementales" definidas sobre estos objetos. Un conjunto de restricciones temporales permiten definir la presencia de la situación problemática a partir de la concurrencia de estas situaciones problemáticas elementales, y también definen el nivel de activación de la situación problemática comenzando a partir del nivel de activación de las situaciones problemáticas elementales. Por ejemplo, la figura 4.5 representa el cálculo del Dominio Temporal de existencia de la situación problemática temporal abstracta "Congestión circular" definida sobre el anillo de calles "calle1-calle2-calle3-calle4".

Toma como base cuatro situaciones problemáticas temporales definidas sobre cada calle involucrada, del tipo "Congestión en calle X", que no definiremos aquí y que corresponden a situaciones de congestión sobre las calles que las definen, y define el dominio temporal de existencia de la congestión circular como la intersección de los dominios temporales de las situaciones problemáticas tomadas como base, tal y como muestra la figura 4.5. Es destacar aquí que la definición real de una congestión circular es un poco más complicada temporalmente de lo aquí expuesto, debido al tiempo de propagación de las congestiones y la influencia de las calles vecinas, pero creemos que el presente ejemplo captura la esencia del problema sin perder por ello claridad.

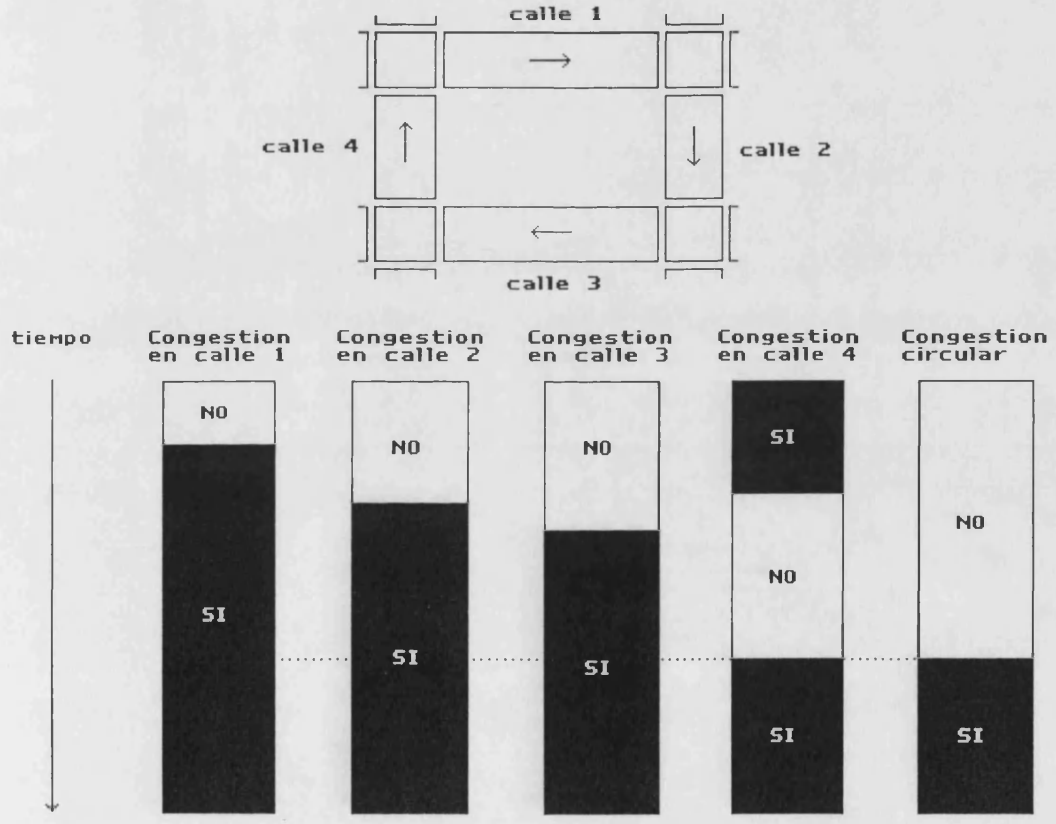

Figura 4.5

IV.6. CONCLUSIONES

El Abstractor de Acontecimientos es un módulo de razonamiento indispensable en la arquitectura propuesta en el capítulo I. Su objetivo es el poder reconocer situaciones problemáticas a partir de los acontecimientos de la Base de Datos Temporal generados por el formalismo (λ,t), por medio de razonamiento cualitativo y temporal

con diferentes niveles de agregación. Hemos adoptado el uso de ECCC para realizar el razonamiento temporal necesario en el AA, por la adecuación de los principios del CC a la representación de la evolución de los parámetros (λ,t).

Como ventaja adicional del uso del ECCC podemos reseñar que al ser representable en forma clausal, su implementación en Prolog resulta totalmente integrada con el formalismo (λ,t). Por otra parte, al ser el ECCC una extensión del EC, resulta lo suficientemente conocido como para que su utilización no resulte extraña.

Los principios del AA presentados en este capítulo permiten construir un formalismo de representación de situaciones problemáticas para cada dominio de aplicación particular. En este sentido es de destacar la facilidad para plasmar mediante el uso del AA, el concepto de situación problemática, debido a la elevada declaratividad que contribuye a una representación de alto nivel, y debido además a la representación cualitativa y temporal utilizada.

El AA permite razonar en términos cognitivos de alto nivel, similares a los utilizados por los humanos, presentando la ventaja de que la identificación de problemas puede hacerse a distintos grados de granularidad espacial y temporal. Es de destacar que al separar el AA del formalismo (λ,t), se consigue que el uso del modelo de comportamiento del sistema alcance unas altas cotas de generalidad, ya que su explotación se hace independientemente del modelo, y además permite ser adaptado a diferentes dominios de aplicación.

Capítulo V

IMPLEMENTACION EN LDOOR

V.1. INTRODUCCION

Uno de los frutos que ha dado nuestro trabajo en el desarrollo del formalismo (λ,t), ha sido la identificación de una serie de deficiencias en la capacidad de representación declarativa del Prolog para implementar los elementos básicos del formalismo. Estas deficiencias consisten en la incapacidad de manejar de forma adecuada los paradigmas de programación orientada a objetos y de programación por restricciones. El objetivo de este capítulo es dar una muestra de la simplicidad y claridad de representación que se alcanzaría expresando el dominio de tráfico urbano en el formalismo (λ,t), si para ello se utilizase un lenguaje de representación del conocimiento de alto nivel que incorporase las características anteriormente citadas. Nosotros utilizaremos un Lenguaje Declarativo Orientado a Objetos y con Restricciones (de ahora en adelante, LDOOR).

El LDOOR es un potente lenguaje de especificación desarrollado en el proyecto ESPRIT-II EQUATOR, con el objetivo de servir como lenguaje de especificación y como lenguaje de implementación (una vez desarrollado), especialmente en entornos que requieran razonamiento cualitativo y temporal, objetivo último del proyecto.

Lamentablemente, no existe todavía, en el momento de escribir esta memoria, ninguna implementación práctica del LDOOR, por lo que este trabajo queda como una potente representación teórica del problema, pero actualmente sin posibilidad de estudiar los resultados de su implementación. La razón para la inclusión de este capítulo hay que enmarcarla en los esfuerzos actuales invertidos en la consecución

práctica de un tal lenguaje, que viene dado por los requisitos de muchas aplicaciones, incluida ésta.

Las características especiales del LDOOR dentro de EQUATOR [FCSL,90] [SYSECA,90] están adaptadas a ciertas necesidades de la representación del conocimiento de tráfico, en especial a su representación bidimensional, y como podrá verse, el código LDOOR de esta aplicación es bastante más corto y legible que el obtenido en otros intentos de codificar el problema como puede ser utilizando PROLOG (Ver Capítulo VI) o en C (Ver Anexo G).

V.2. LDOOR

El LDOOR es un lenguaje de propósito general orientado a objetos, lógico, y de restricciones diseñado primariamente para su utilización en aplicaciones interactivas que precisan razonamiento temporal con componentes parcialmente ordenados.

El diseño del LDOOR cumple los siguientes requisitos:

1. Soportar la representación declarativa del conocimiento, por razones de expresividad y reusabilidad.

2. Capacidad para razonar con conocimiento incompleto. El sistema debe ser capaz de caracterizar las soluciones razonando directamente sobre conjuntos de posibles valores para ellas (su dominio). Una aproximación basada en restricciones es una respuesta directa a esta necesidad.

3. Conseguir economía de expresión. El lenguaje debe explotar una organización jerárquica y orientada a objetos de las restricciones que representan el conocimiento.

4. Posibilidad para poder utilizarse en aplicaciones industriales que requieren una potente herramienta para razonar con estructuras compuestas, donde los componentes estén parcialmente ordenados, y donde los atributos tomen sus valores en métricas de diferentes granularidades. La aproximación basada en restricciones también resuelve este punto.

Otros lenguajes que combinan una aproximación orientada a objetos con la programación lógica son lenguajes híbridos. La aproximación basada en restricciones

del LDOOR está diseñada para dar la potencia de ambos desde un entorno de trabajo homogéneo.

En LDOOR, la especificación de tipos e instancias constituye una estructura jerárquica orientada al objeto, que permite herencia múltiple de restricciones sobre los atributos. Los tipos son considerados como restricciones a resolver, como en otras aproximaciones basadas en (o relativamente relacionadas con) CLP ([Ait-Kaci y Masr,1986],[Beringer y Porcher, 1989]). Pero a diferencia de la familia de lenguajes CLP ([Jaffar y Lassez, 1987], [Van Hentenrick, 1989], [Colmerauer, 1986]) LDOOR no fuerza una partición de los predicados en dos conjuntos: los que pueden ser simplificados y los que pueden ser probados. En vez de ello, LDOOR trata los predicados de forma uniforme como restricciones, y cualquier restricción sobre un atributo puede ser probada o simplificada. La resolución de restricciones añade el poder del Prolog a la simplificación de restricciones. La semántica de la resolución está dentro de la lógica de la resolución de restricciones.

Se requiere un rico conjunto de diferentes ordenaciones de las restricciones para conseguir este poder expresivo, sin embargo, las restricciones primitivas se escojen con gran economía: LDOOR es capaz no sólo de representar pertenencias a dominios, sino también de representar la no pertenencia. Además, un dominio puede ser no solamente un tipo, sino también una unión de tipos o instancias o dominios de otras variables. Otros conjuntos de restricciones, por ejemplo las aritméticas, se añaden sobre la base de las restricciones primitivas.

V.3 APLICACION DEL LDOOR A LA REPRESENTACION DEL FORMALISMO (λ,T) EN EL DOMINIO DEL TRAFICO URBANO

V.3.1 Descripción del Escenario

A lo largo de las siguientes páginas, ilustraremos un caso práctico de aplicación del LDOOR a la representación del formalismo (λ,t) . El escenario elegido para la representación en LDOOR del conocimiento sobre el dominio de aplicación es el objeto con comportamiento más complejo en el Control de Tráfico Urbano. Este se compone de una única calle con un solo sentido de circulación terminada en un semáforo y de un solo carril. La afluencia de vehículos a la calle la supondremos constante y de valor d2, mientras que la salida de la calle, pasado el semáforo, supondremos que es capaz de absorber cualquier flujo de tráfico sin retención alguna (o sea, que es un sumidero de capacidad superior al máximo flujo entrante desde la

calle dada), por lo que podemos evitar considerarlo en el escenario. Este escenario puede verse en la figura siguiente:

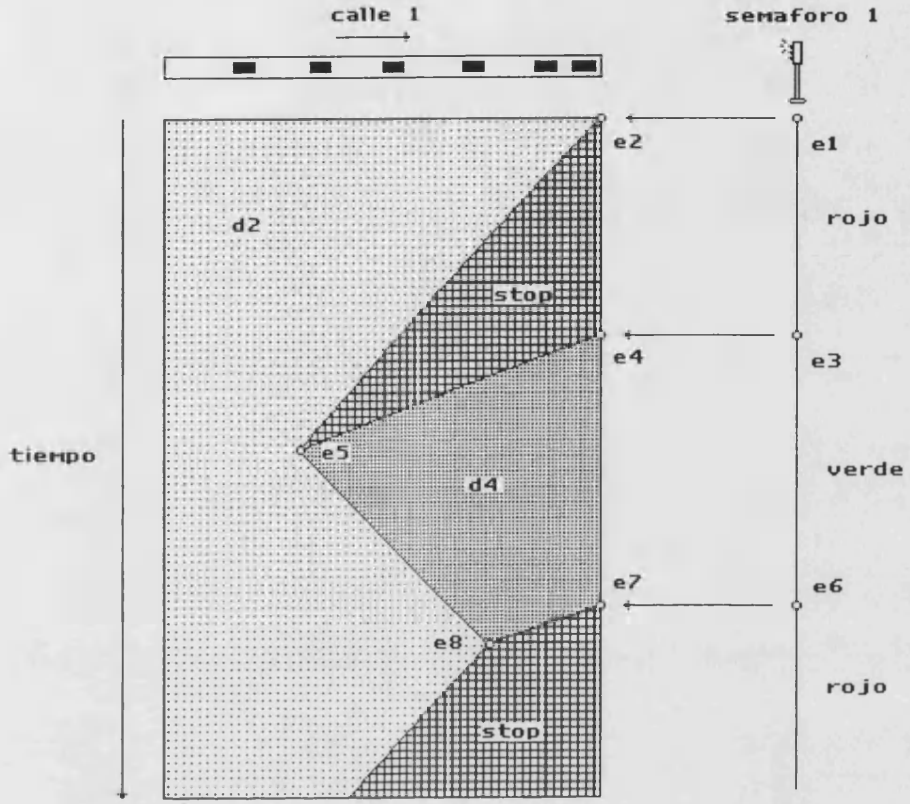

Figura 5.1

En la figura, el estado del semáforo se representa en la línea vertical de la derecha. Allí pueden verse reflejados los sucesos originados por el semáforo: e1, e3 y e6.

Los datos iniciales para la simulación cualitativa son:

- Evolución temporal de los elementos no predecibles durante el periodo de simulación (En nuestro ejemplo, debe darse la evolución temporal del semáforo, de la entrada y de la salida de la calle. Puesto que hemos supuesto que la entrada y la salida de la calle no influyen en el comportamiento del sistema, no se incluye su evolución temporal, pero sí debemos incluir los cambios de estado del semáforo, por lo que debemos introducir los acontecimientos e1, e3 y e6 como datos de entrada para simulación).

- Estado inicial de los componentes cuyo comportamiento es predecible. (En este caso el único es el estado de la calle 1, cuyo estado inicial suponemos un valor cualitativo de densidad d2 a lo largo de la calle).

El resultado final de la simulación es el conjunto de acontecimientos que describen el comportamiento cualitativo a lo largo del tiempo de los elementos del sistema durante el intervalo de tiempo considerado. Este conjunto de acontecimientos es en nuestro caso e2, e4, e5, e7 y e8.

V.3.2 Modelo Conceptual del Escenario

El problema puede ser simplificado realizando la descripción cualitativa y temporal de la calle de forma similar al Cálculo de Acontecimientos para Cambio Continuo (Event Calculus for Continuous Change, ECCC) [Shanahan'89], como primer paso hacia su reescritura en LDOOR. Explicaremos seguidamente este concepto.

Consideremos la descripción cualitativa y temporal del comportamiento de la calle (ver figura 5.2). Este comportamiento puede ser dividido a lo largo del eje temporal en una serie de intervalos temporales llamados ec1, ec2, ec3,... El primer intervalo temporal (ec1) comienza a partir del estado inicial y termina en el primer instante en que se produce el acontecimiento de la calle 1 (e4). El resto de intervalos temporales comienzan al final del intervalo anterior y terminan en el momento en que se produce el próximo acontecimiento que tenga lugar en la calle 1.

Cada intervalo temporal tiene una propiedad interesante que permanece constante en todos sus puntos: la estructura cualitativa de la calle. Esta estructura se define como una lista que contiene los valores cualitativos de las secciones de la calle en un momento del tiempo.

En el ejemplo de la figura 5.2 la estructura cualitativa de cada intervalo temporal es la siguiente:

TI	**EC**
ec1	[d2,stop]
ec2	[d2,stop,d4]
ec3	[d2,d4]
ec4	[d2,d4,stop]
ec5	[d2,stop]

En cada una de estas estructuras cualitativas se hace necesaria una propiedad de segundo orden: el conjunto de fronteras. Una propiedad de segundo orden, según la nomenclatura del ECCC [Shanahan,89], es una propiedad que representa un valor de cambio continuo durante un intervalo de tiempo. De igual forma, la lista formada por las posiciones de las fronteras entre las secciones de calle con diferentes estados cualitativos en un momento del tiempo varía linealmente con el tiempo a lo largo del intervalo temporal.

Para tratar el comportamiento del conjunto de fronteras en LDOOR, se divide cada estructura cualitativa en un conjunto infinito de instantáneas (en expresión anglosajona, snapshots) del estado de la calle. Cada instantánea representa el estado de la calle en un momento del tiempo, por lo que el conjunto de fronteras tendrá un valor único en cada una.

En la figura 5.3 puede verse el árbol jerárquico que representa la descripción cualitativa y temporal de la calle. La figura representa un árbol hasta profundidad dos. El nodo raíz, denominado calle 1, representa el comportamiento cualitativo y temporal completo de la calle 1. El primer nivel está formado por nodos etiquetados como :estructura_cualitativa, donde cada nodo representa una estructura cualitativa válida durante un intervalo de tiempo. En la figura 5.3 se representan los nodos correspondientes a las estructuras cualitativas ec1 y ec2 de la figura 5.2.

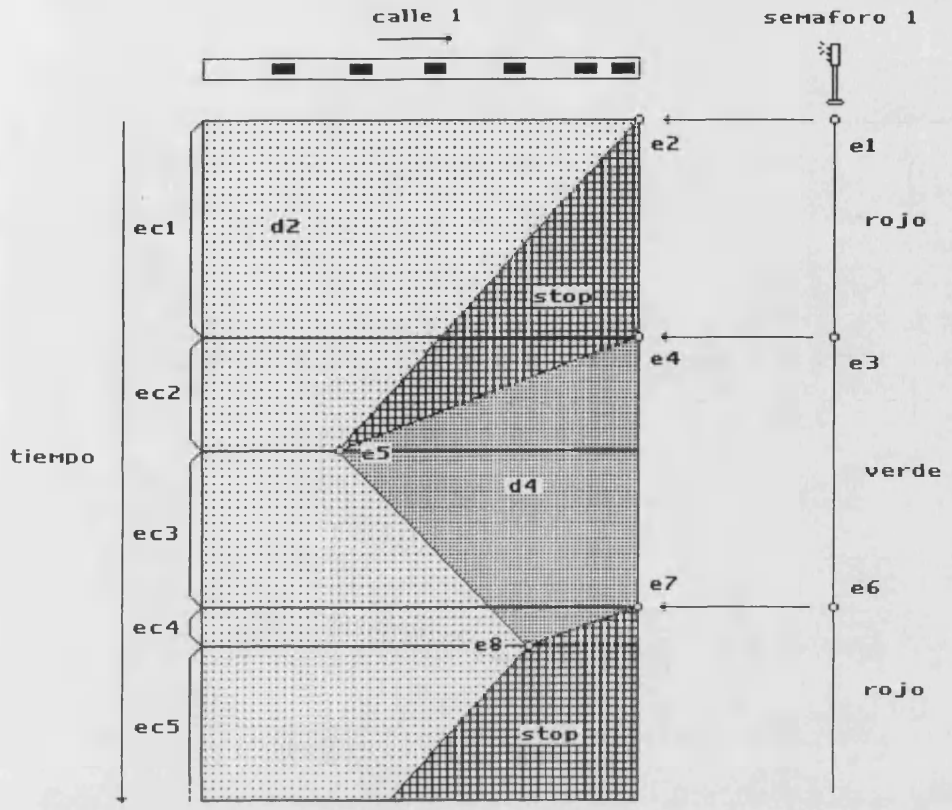

Figura 5.2

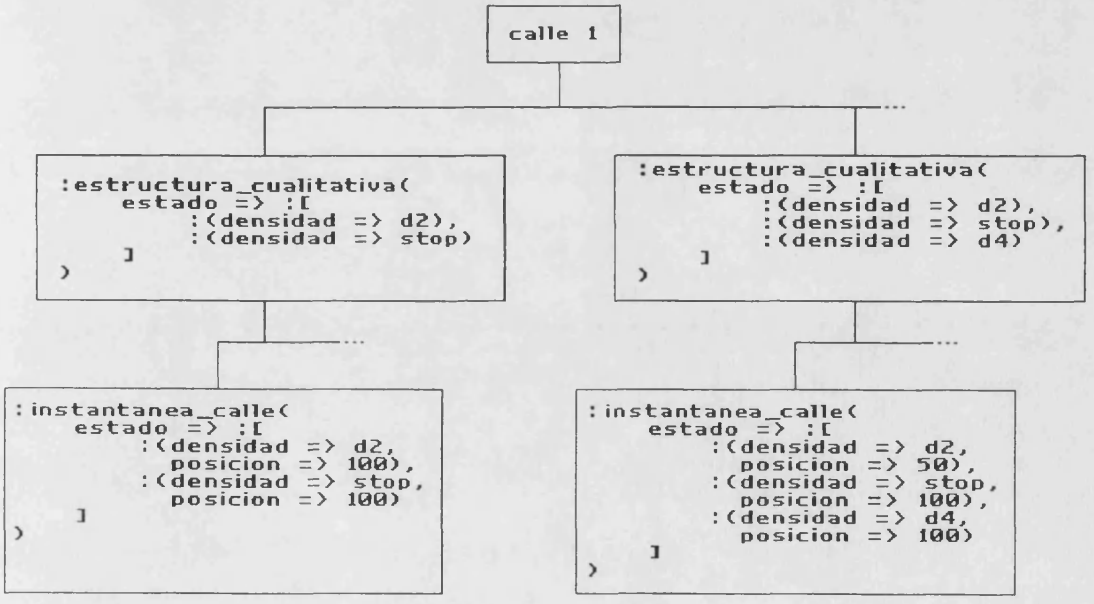

Figura 5.3

Estas estructuras cualitativas se dividen en un número infinito de instantáneas, cada una representando un instante del tiempo. Como no es posible representar las infinitas instancias de cada estructura, sólo se representa la primera de cada una. Esta primera instantánea representa el estado exacto de la calle en el momento del tiempo en que comienza la estructura cualitativa. Todas las demás instantáneas pueden deducirse fácilmente a partir de la primera.

Por razones de simplicidad en la explicación, escojeremos una evolución temporal más simple que la mostrada en la figura 5.2 sobre la misma escena de trabajo, y que puede verse en la figura 5.4. La evolución escogida consiste sólo en dos estructuras cualitativas (en el periodo de tiempo estudiado) que se representan en la tabla siguiente:

TI	EC
ec1	[d2,d4,stop]
ec2	[d2,stop]

La división de estas nuevas estructuras cualitativas en instantáneas puede verse en la figura 5.5. En la representación de las estructuras mostrada, se muestra una característica importante del LDOOR: la especificación progresiva del dominio de atributos desde los objetos más generales a los más particulares.

Analicemos el siguiente caso: el atributo "estado" representa el estado de la calle. Este atributo no aparece en el nodo principal (calle 1), lo que quiere decir que su dominio a este nivel es el universo completo (En LDOOR, todo atributo no especificado en un objeto tiene como dominio de valores el universo completo, lo que significa que no disponemos de información para restringir su dominio de valores). En cada nodo del primer nivel puede encontrarse una estructura cualitativa dada, por lo que el valor del atributo estado se restringe con esta estructura (en nuestro caso, estado =>:[:(densidad => d2),:(densidad => d4),:(densidad => stop)] para la estructura cualitativa ec1 y estado => :[:(densidad => d2),;(densidad => stop)] para la estructura cualitativa ec2). Sin embargo, la posición de las fronteras entre las diferentes secciones cualitativas que componen cada estructura cualitativa permanece todavía desconocida. Esto es debido al cambio continuo de posición de las mismas a lo largo del intervalo temporal de existencia de la estructura cualitativa (ver figura 5.4).

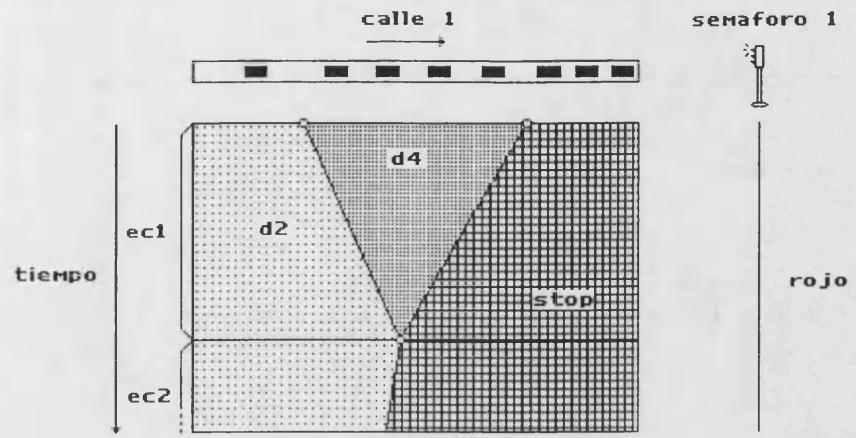

Figura 5.4

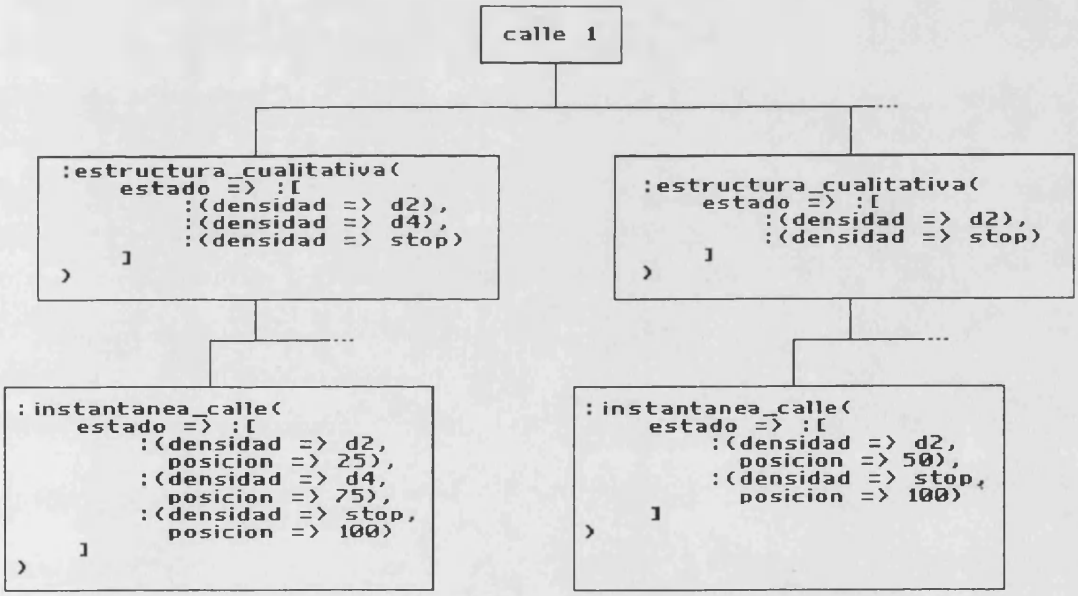

Figura 5.5

Cada nodo del segundo nivel representa una instantánea de la estructura cualitativa a la que pertenece, y como en una instantánea la posición de las fronteras está definida, se restringe aún mas el valor del atributo estado indicando estas posiciones, tal y como puede verse en la figura 5.5.

Tomemos la primera instantánea:

```
:instantánea_calle(
        estado => :[
                :(densidad => d2,
                  posición => 25),
                :(densidad => d4,
                  posición => 75),
                :(densidad => stop,
                  posición => 100)
        ]
).
```

Aquí el atributo estado determina, no sólo la estructura cualitativa correspondiente sino también la posición exacta de las fronteras entre los sectores cualitativos de la calle. El atributo posición indica la posición espacial del extremo final de la sección cualitativa a la que pertenece el atributo (contando a partir del comienzo de la calle, que se define como el punto de entrada de los vehículos en la calle). Es de notar que el extremo inicial de cada sección cualitativa no está indicado con el objeto de evitar redundancias, ya que es igual al extremo final de la sección de calle anterior, (excepto para la primera sección, donde vale cero).

El atributo posición de la última sección cualitativa de la calle es redundante, puesto que su valor es igual a la longitud total de la calle, que se hereda de las especificaciones estáticas de la calle.

En resumen, la interpretación de la expresión

```
:[
        :(densidad => d2,
          posición => 25),
        :(densidad => d4,
          posición => 75),
        :(densidad => stop,
          posición => 100)
]
```

sería la siguiente: La calle estaría dividida en tres secciones cualitativas:

- La primera, de valor d2, desde la posición 0 hasta 25.
- La segunda, de valor d4, desde 25 hasta 75.
- La tercera, de valor stop, desde 75 hasta 100.

Este es pues el estado de la calle en la primera instantánea, que puede verse de una forma gráfica en la figura 4 (estado de la calle al comienzo de ec1).

V.3.3 Código LDOOR

La simulación cualitativa del comportamiento de esta calle consiste en generar la secuencia de estructuras cualitativas únicamente a partir de la primera instantánea y de las influencias externas al sistema (p.ej. cambios de semáforos).

El programa se ha simplificado al máximo para facilitar su exposición, sin embargo su extensión al tratamiento completo del problema requiere poco conocimiento adicional y no afecta sustancialmente su legibilidad.

Primeramente estudiaremos la estructura de los datos y las restricciones asociadas que describen los componentes temporales de una calle:

V.3.3.1 Base de Conocimientos del Escenario en LDOOR

El conocimiento general sobre las calles, y el conocimiento particular sobre la calle 1, se codifican en la regla mostrada a continuación, que en esencia significa lo siguiente: Los objetos instancia de la clase **calle** tienen dos atributos que los determinan: su longitud L (longitud_calle) y su composición temporal (componentes(tiempo)). La composición temporal de la calle debe ser un conjunto de estructuras cualitativas, que a su vez se componen de un estado cualitativo común S y de un conjunto de instantáneas. Las instantáneas se especifican a través de una lista de posiciones de fronteras en cada momento del tiempo, de las que la última es siempre igual a la longitud de la calle, lo que se especifica igualándola a L. Por último, el conocimiento particular sobre la calle 1 está colocado como restricción[1] a la clase calle, y puede entenderse así: "Cualquier objeto de la clase calle que se llame calle 1, ha de tener su atributo longitud igual a 100 metros". El listado de este predicado puede verse a continuación:

[1] Recordemos que el formato general de un predicado en CRL es: [objeto]:clase[:superclase](lista de pares atributo => valor) / lista de restricciones a cumplir por el objeto (ver anexo C).

```
:calle(
        longitud_calle => L,
        componentes(tiempo) => :(
                todo_en_dominio =>
                        :estructura_cualitativa(
                                estado => S,
                                componentes(tiempo) => :(
                                        todo_en_dominio => instantánea_calle(
                                                estado => S1:(
                                                        ultimo_mb => :(
                                                                posicion => L
                                                        )
                                                )
                                        )
                                )
                        )
        )
)

/

calle1:calle(longitud_calle => 100).
```

El conocimiento sobre la relación que liga los posibles estados en que pueden encontrarse las instantáneas de una misma estructura cualitativa, y que en esencia consiste en que la estructura cualitativa de densidades es común a todas ellas, se condensa en el siguiente predicado, que indica que todos los objetos X de una clase **misma_densidad**, pueden ser o bien conjuntos de listas cuyos elementos contienen un atributo densidad que es común a todas, o bien listas vacías (en el anexo C se muestra cómo las llaves { } denotan conjuntos de elementos en LDOOR). El listado de esta predicado puede verse a continuación:

```
X:
        misma_densidad /
        X:{
                :(
                        lista1 => L1:(cabeza => :(densidad => D), cola => COLA1),
                        lista2 => L2:(cabeza => :(densidad => D), cola => COLA2)
```

```
        );
        :(lista1 => [], lista2 => [])
    },
    :misma_densidad(
        lista1 => COLA1,
        lista2 => COLA2
    ), L1 <> [], L2 <> [].
```

El estado de las fronteras de las distintas instantáneas que componen una misma estructura cualitativa es distinto, pero puede ser calculado conociendo el estado de la primera instantánea de la estructura cualitativa, conociendo las ecuaciones de evolución temporal de las fronteras. Este conocimiento es descrito por el siguiente predicado de la siguiente forma: todo objeto de la clase **instantánea_calle** forma parte de una estructura cualitativa, y toda estructura cualitativa es iniciada por una instantánea específica, que aquí denotamos por S0. Entonces los estados de ambas instantáneas están relacionados por medio del predicado posicion_frontera_segun_tiempo, que se describirá más adelante. El listado del predicado considerado es mostrado a continuación:

```
:instantánea_calle:instantánea(
        parte_de => :estructura_cualitativa(iniciada_por => S0))
/
:posicion_frontera_segun_tiempo(
        instantánea_calle0 => S0,
        instantánea_calle => S
).
```

Las ecuaciones de evolución temporal de las fronteras que ligan entre sí los estados de la instantánea inicial de una estructura cualitativa y de cualquier otra instantánea de la misma se expresan en el predicado siguiente, cuyo significado viene a ser el siguiente: para todo objeto X del tipo **posicion_frontera_segun_tiempo** existen dos atributos, instantánea_calle0 e instantánea_calle, que representan los estados de la instantánea inicial y otra instantánea cualquiera de la estructura cualitativa. Cada estado contiene dos atributos, uno llamado tiempo, que contiene el tiempo de ocurrencia de la instantánea especificada, y otro llamado estado, que contiene una lista con los pares (posición de frontera,valor de la densidad) para cada sección cualitativamente distinta de la calle. Estas listas pueden estar ambas vacías o ambas

llenas, y en este último caso el atributo posición de la cabeza de ambas listas ha de responder a la fórmula lineal P = P0 + K*(T-T0), donde P0 y P son las posiciones es la posiciones de la frontera especificada en T0 (inicial) y T, respectivamente, y el coeficiente K se obtiene por medio del predicado tabla_busqueda, que permite obtener la velocidad de una frontera a partir de los estados cualitativos de densidad a ambos lados (D0a y D0b en el predicado). Este razonamiento en el primer elemento de la lista estado se aplica a todos los demás elementos de la lista llamando recursivamente al mismo predicado. El listado completo del predicado puede verse a continuación:

```
X:posicion_frontera_segun_tiempo
/
X:{
        :(
                instantanea_calle0 => :(
                        tiempo => TP0:(tp => T0),
                        estado => :(
                                cabeza => :(posicion => P0, densidad => D0a),
                                cola => L0:(cabeza => :(densidad => D0b))
                                )
                ),
                instantanea_calle => :(
                        tiempo => TP:(tp => T),
                        estado => :(cabeza => :(posicion => P),cola => L)
                ),
        );
        :(
                :instantanea_calle0 => :(estado => :(cola => [])),
                :instantanea_calle => :(estado => :(cola => []))
        )
},
L0 <> [], L <> [],
:tabla_busqueda(d1 => D0a,d2 => D0b,k => K),
P = P0 + K*(T-T0),
:posicion_frontera(
        instantanea_calle0 => :(tiempo => TP0, estado => L0),
        instantanea_calle => :(tiempo => TP, estado => L)
).
```

El predicado que representa la tabla que permite obtener la velocidad de la frontera entre dos secciones cualitativamente distintas de calle a partir de los valores cualitativos de densidad de las mismas puede verse a continuación:

```
T:tabla_busqueda
/
T:{
        :(d1 => d1, d2 => d2, k => 50);
        :(d1 => d1, d2 => d3, k => 40);
        .....
        :(d1 => d7, d2 => stop, k => -10)
}.
```

La propiedad de autoterminación de las estructuras cualitativas, similar a la propiedad de autoterminación definida por Shanahan en su ECCC [Shanahan,89], consiste para las calles en detectar el posible choque entre dos fronteras cualitativas y señalarlo como fin de la estructura cualitativa. Esta propiedad define por tanto un acontecimiento de terminación de la estructura que ocurrirá únicamente si la estructura no ha terminado anteriormente por la acción de algún otro acontecimiento. El predicado que representa este conocimiento lo hace de la siguiente forma: Toda instantánea de una estructura cualitativa X en cuyo estado M puedan encontrarse dos fronteras de dos secciones distintas con el mismo valor P, hará terminar a la estructura cualitativa X, y hará comenzar a una nueva estructura cualitativa, cuya primera instantánea será una copia de la instantánea de autoterminación, quitando la sección cualitativa que ha desaparecido como consecuencia del choque de fronteras. Una aclaración que conviene hacer es que es innecesario indicar que para que haya autoterminación, ningún acontecimiento debe terminar la estructura cualitativa con antelación, puesto que si eso llegara a suceder, la instantánea de autoterminación jamás se instanciaría, por no poderse satisfacer la condición de choque de fronteras en el intervalo de validez que el acontecimiento de terminación ha impuesto para la estructura. El listado completo de este predicado puede verse a continuación:

```
:estructura_cualitativa_autoterminante:instantanea_calle(
        tiempo => :(tp => T),
        parte_de => X:estructura_cualitativa(
                estado => :(mb => M),
                iniciado_por => :(tiempo => :(tp => T0))
```

```
        ),
        termina => X,
        inicia => X1:estructura_cualitativa(
                estado => :(mb => M1 \:(indice => I1),
                componentes => :(
                        cabeza => :instantanea_calle(
                                estado => :(mb => Di1 \:(indice => I1))
                        )
                )
        ),
        estado => :(mb => Di)
)
/
:Di(indice => I, posicion => P),
:Di(indice => I1, posicion => P),
I1 = I + 1,
T > T0,
M1:M,
Di1:Di.
```

Las condiciones iniciales que se necesitan en nuestro escenario para poder realizar razonamiento sobre el comportamiento del mismo, consisten en la descripción de la instantánea inicial de la primera estructura cualitativa de la calle1, que se representa en el siguiente predicado:

```
estructura_cualitativa_1:estructura_cualitativa_inicial(
        calle => calle1,
        estado => :[
                :(densidad => d2, posicion => 25),
                :(densidad => d4, posicion => 75),
                :(densidad => stop, posicion => 100)
        ],
        tiempo => :(tp => 0),
        sucede_en => mundo
).
```

Por otro lado, necesitamos un predicado que relacione la primera instantánea de una estructura cualitativa, con la propia estructura cualitativa, y este predicado es el siguiente:

```
X:estructura_cualitativa_inicial:instantanea_calle(
        inicia => :estructura_cualitativa(
                componentes(tiempo) => :(cabeza => X)
        )
).
```

Los predicados citados hasta ahora completan la descripción del escenario en LDOOR, que utilizaremos en las secciones posteriores.

V.3.3.2 Cálculo de Acontecimientos en LDOOR

Antes de describir como será utilizado este conocimiento, estudiaremos cómo codificar el Cálculo de Acontecimientos en el LDOOR:

```
X:(
        tiempo => :(desde => T1, hasta => T2),
        sucede_en => :{H1;H2},
        iniciado_por => E1:(
                inicia => X,
                tiempo => T1,
                sucede_en => H1
        ),
        terminado_por => E2
)
/
:(
        termina => X,
        sucede_en => H2,
        después => :(mb(_) => E1),
        lista => :(elmasantiguo => E2:(tiempo => T2))
).
```

Como necesitamos considerar el cambio continuo, debemos incluir una definición de los componentes de mayor finura temporal de una propiedad (i.e. estructuras cualitativas), a los que llamaremos instantáneas:

```
:instantanea(
        sucede_en => :H,
        parte_de(tiempo) => :(
                iniciado_por => E1:(sucede_en => H),
                terminado_por => E2),
        después => E1,
        antes => E2
).
```

La definición anterior especifica que una instantánea sucede si es parte (temporal) de una propiedad que es iniciada por un acontecimiento que sucede.

V.3.3.3 Utilización de la Base de Conocimientos en LDOOR

Ahora estamos en posición de describir de qué manera se empleará la Base de Conocimiento. La pregunta que mostraremos a continuación pide los componentes temporales de la calle 1, que es equivalente a pedir su evolución temporal. Estos componentes forman una lista (L) de estructuras cualitativas, cuyos atributos son:

1. El intervalo de tiempo en el que son válidas.
2. La lista de densidades.

El simbolo ' después de calle 1 indica que los atributos de calle 1 deben ser reconocidos en la Base de Conocimiento. Esto significa, teniendo en cuenta la pregunta formulada, que debe probarse que sucede cada estructura cualitativa que compone la evolución del sistema. En otras palabras, las estructuras cualitativas que representan la evolución posterior del escenario se derivan como consecuencia lógica de la Base de Conocimiento que lo describe y de la pregunta formulada. Para obtener explícitamente los atributos tiempo => y estado => de cada miembro de L, se utiliza el atributo predefinido "lista(explicita([tiempo,estado])) => ".

La pregunta a la Base de Conocimiento que hemos escogido como ejemplo es la siguiente:

```
?- calle1':(
  componentes(tiempo) => L:(
        todo_en_dominio => :(sucede_en => mundo),
        mb => :(lista(explicita => ([tiempo,estado])) => L)
  )
).
```

Y la respuesta que el motor de inferencia del LDOOR debería dar a esta pregunta es la lista de estructuras cualitativas que componen la evolución futura del escenario:

```
L:[
  :estructura_cualitativa(
        tiempo => :(desde => :(tp => 0), hasta => :(tp => 13)),
        estado => :[
              :(densidad => d2),
              :(densidad => d4),
              :(densidad => stop)
        ]
  ),
  :estructura_cualitativa(.....),
  .....,
  :estructura_cualitativa(
        tiempo => :(desde => :(tp => 50), hasta => :(tp => infinito)),
        estado => :[:(densidad => stop)]
  )
].
```

Se utilizará el Cálculo de Acontecimientos para deducir que la primera estructura cualitativa sucede debido a que es iniciada por estructura_cualitativa_1, la cual tiene las siguientes características:

1. Es definida como que sucede en el instante de tiempo 0.
2. Es de tipo :estructura_cualitativa_inicial, que se define como iniciadora de una estructura cualitativa.

Esta primera estructura cualitativa, y aquellas que la siguen, terminarán automáticamente cuando dos de sus líneas frontera se crucen, merced al predicado de autoterminación (ver figura 4). Cada vez que una estructura cualitativa autotermine,

iniciará automáticamente una nueva estructura cualitativa cuya primera instantánea tendrá los mismos elementos que la última instantánea de la estructura anterior, pero sin la región que ha desaparecido (que es la que causa la autoterminación). Esta transformación está expresada en la clase :estructura_cualitativa_autoterminante, anteriormente mostrada.

En nuestro ejemplo, como no hay cambios en las luces del semáforo, la última autoterminación será cuando la cola llene completamente la calle. Esta última autoterminación iniciará una estructura cualitativa de duración indefinida ("hasta => :(tp => infinito)") y con una única densidad (" estado => :[:(densidad => stop)] ").

Explicaremos seguidamente la organización de la Base de Conocimiento mostrada en la sección V.3.3.1 en líneas generales:

La frase ":calle" describe las restricciones comunes a todas las calles. Esto significa que cualquier calle debe tener un atributo longitud restringido al valor de la calle en metros y un atributo compuesto cuyo valor debe ser una lista de estructuras cualitativas. Cada una de estas estructuras cualitativas tiene un atributo estado que describe la lista de secciones de calle que los definen. La última de estas secciones tiene la restricción de que su extremo final (posicion => atributo) debe ser igual a la longitud de la calle (o sea, que la última sección termina donde lo hace la calle).

Cada estructura cualitativa tiene un atributo componente cuyo valor es la lista de instantáneas que pertenecen a la estructura, y cuya última sección de calle tiene igualmente restringido su atributo posicion => al valor de la longitud de la calle.

La última restricción significa que la lista de densidades que caracteriza cada estructura cualitativa debe ser igual a la lista de densidades de cada una de sus instancias. Esta restricción está representada por ":misma_densidad".

Las frases calle1 y estructura_cualitativa_1 describen la escena sobre la cual razonamos y el estado inicial del sistema. Pueden ser interpretadas de la siguiente manera: La calle 1 tiene 100 metros de longitud, y su estado en la primera instancia de la primera estructura cualitativa es :[:(densidad => d2, posicion => 25), :(densidad => d4, posicion => 75), :(densidad => stop, posicion => 100)], y sucede en tiempo 0.

La frase :instantanea_calle es la que establece, para cada instantánea, la posición de cada una de las fronteras entre las secciones cualitativas que dividen la calle, dentro de la estructura cualitativa a la que pertenecen.

Las fronteras entre regiones cualitativas adyacentes son líneas rectas dentro de cada estructura cualitativa (ver figuras 2 y 4), o sea que el movimiento de las fronteras no experimenta cambio alguno y se realiza a velocidad constante. Para calcular el estado de cada calle basta conocer:

1. El estado de la calle en la primera instancia de la estructura cualitativa a la que pertenece.

2. La velocidad de cada frontera. La cual a su vez cual depende de los estados cualitativos que esta frontera separa (recuérdese el predicado tabla_busqueda de la sección V.3.3.1).

3. Los puntos de tiempo de ambas instancias.

La restricción entre la lista de fronteras de cada instancia está representada en la frase :posicion_frontera_segun_tiempo.

La frase :estructura_cualitativa_autoterminante, describe la condición que una instancia debe cumplir para terminar la estructura cualitativa a la que pertenece e iniciar la estructura siguiente.

Esta estructura cualitativa que autotermina es una instancia en cuyo estado :(mb => Di) debe haber una sección de calle de longitud 0 (lo que quiere decir que ha ocurrido una colisión de fronteras, por lo que se inicia una nueva estructura cualitativa). La instancia autotermina si en la lista :(mb => Di) hay dos Di adyacentes con el mismo valor del atributo pos (lo que equivale a decir que la instancia autotermina si dos fronteras chocan, tal y como vimos en la definición del predicado estructura_cualitativa_autoterminante del la sección V.3.3.1).

En una estructura cualitativa pueden haber colisiones de fronteras en periodos de tiempos diferentes, pero únicamente el primero termina la estructura cualitativa, con lo que los siguientes nunca llegan a suceder, al menos en la misma estructura cualitativa.

La instantánea que termina una estructura cualitativa comienza otra nueva (inicia => X1) cuya primera instantánea tiene un estado idéntico al de la instantánea autoterminante, pero eliminando la sección cualitativa de calle de longitud cero originada por la colisión de fronteras (o sea, :(mb => Di1 /:(index => I1)).

V.4 CONCLUSIONES

La conclusión que se obtiene de este capítulo consiste en la idoneidad, para la implementación del formalismo (λ,t), de disponer de un lenguaje declarativo orientado a objetos y basado en tecnicas de programacion logica basada en restricciones. La sencillez y claridad con que ha quedado expresado el formalismo en LDOOR a lo largo de las páginas anteriores demuestran esta afirmación. La especificacion de este lenguaje, que se ha hecho dentro del proyecto ESPRIT II EQUATOR, recogiendo los requerimientos presentados en esta memoria, esta completada, pero sin embargo no se ha comenzado su implementacion, tema en el que creemos que se debe trabajar en corto espacio de tiempo.

Capítulo VI

IMPLEMENTACION INDEPENDIENTE DEL DOMINIO DEL FORMALISMO (λ,T)

VI.1. INTRODUCCION

En el capítulo II se ha presentado el formalismo (λ,t), y a lo largo del capítulo III se ha estudiado su aplicación a dos problemas con diferentes particularidades. Sin embargo, para mostrar la generalidad del formalismo se hace necesario extender la implementación a otras aplicaciones. Por otra parte, también es necesario dar una explicación con todo detalle de los mecanismos de funcionamiento del motor de inferencia del formalismo (λ,t).

En el presente capítulo presentamos una implementación del formalismo (λ,t) en Prolog, cuyo listado completo viene en el anexo D. Esta implementación se hace de la forma más general posible, i.e. independiente del dominio de aplicación, lo que puede dar una idea más aproximada del nivel de abstracción del formalismo. Los principios teóricos del motor de inferencia del formalismo (λ,t) se han explicado, junto con el formalismo (λ,t) en el capítulo II. En este capítulo recordaremos los aspectos más importantes de funcionamiento teórico y analizaremos la forma en que han sido codificados en Prolog.

VI.2. MOTOR DE INFERENCIA

En el capítulo II se mencionó que el formalismo (l,t) consta de dos partes diferenciadas: Un método de representación del conocimiento de los sistemas a tratar, y un motor de inferencia que trabaja sobre este conocimiento. La primera parte fue

presentada con detalle en el capítulo II, y de la segunda se presentaban allí los principios básicos. En este capítulo vamos a tratar la implementación del motor de inferencia con todo tipo de detalle, para mostrar la viabilidad computacional de esos principios básicos.

Recordemos el ciclo básico de funcionamiento del motor de inferencia, explicado en la sección II.4. Este ciclo sigue los siguientes pasos:

-1) Obtención de t_{min}: Sean $O_1...O_n$ los objetos del sistema. Sea t_i el tiempo en el que se produjo el último suceso que tenemos almacenado correspondiente al objeto O_i. Sea $t_{min} = \min\{t_1...t_n\}$. Antes del tiempo t_{min} los objetos $O_1,...,O_n$ tienen su comportamiento perfectamente definido (por la definición de t_{min}).

-2) Obtención del conjunto de sucesos potencialmente activos: Se forma el conjunto de sucesos que tienen la condición de suceder después o simultáneamente al tiempo anteriormente calculado (recordemos que sólo los sucesos de este conjunto pueden originar nuevos sucesos).

-3) Generación de hipótesis de nuevos sucesos: Cada suceso del conjunto anterior es analizado con el objetivo de establecer hipótesis de posibles estados futuros que pueden ser derivados del mismo.

-4) Comprobación de hipótesis: Se analiza el conjunto de hipótesis para descartar aquellas que son inconsistentes y quedarse sólo con las hipótesis consistentes, que dejarán de ser hipótesis para convertirse en sucesos y guardarse en la base de datos.

-5) Condición de finalización: Si t_{min} es inferior al tiempo de finalización de la simulación, volver al paso 1; en caso contrario, finalizar el ciclo.

El ciclo de funcionamiento del algoritmo puede verse en la figura 2.11.

VI.3. CICLO PRINCIPAL DEL MOTOR DE INFERENCIA

La implementación en Prolog del motor de inferencia del formalismo (λ,t) que presentamos en este capítulo, se diferencia del algoritmo dado mostrado en la sección anterior en un punto importante de carácter más bien formal, pero que conviene aclarar con objeto de evitar confusiones. En este capítulo adoptaremos como elemento base en la representación del conocimiento del sistema una

representación orientada a objetos del formalismo (λ,t). El motor de inferencia ha sufrido las pertinentes modificaciones, de forma que ya no hay un algoritmo base que realice operaciones sobre los objetos, sino que estas operaciones son realizadas por los mismos objetos, a través de unas funciones comunes heredadas por todos los objetos del sistema. Estas funciones calculan las hipótesis de evolución posible del objeto y se comunican con los objetos vecinos para saber cuáles son posibles. Como puede observarse, no se ha alterado el algoritmo del motor de inferencia recordado en la sección anterior, sino tan sólo la forma de ejecutarlo.

El punto de entrada del motor de inferencia lo constituye el predicado main/0, cuya función es iniciar la ejecución cíclica del predicado analiza_objeto/3 hasta que se cumpla una determinada condición. A tal efecto, se define un hecho en la base de datos, testigo/0, con el objeto de saber si la condición de parada se cumple. El algoritmo principal que sigue la función main/0 será pues el siguiente:

1. Borrar el testigo de parada (anular_objeto/0).

2. Llamar al predicado analiza_objeto(Objeto,Clase,Tiempo) para cada objeto que pertenezca al dominio interior (clases de objetos sin contacto con el mundo exterior, cuyo comportamiento ha de predecirse. Recordemos que los objetos en contacto con el mundo exterior tenían su comportamiento fijado de antemano). Si este predicado decide que el objeto que lo llama ha de provocar aún un cambio en la base de datos temporal, se levanta el testigo de parada (analizar_objetos(Tiempo final de simulación)).

3. Si el testigo de parada se ha levantado, se repite el ciclo, en caso contrario, significa que ningún objeto producirá ya más cambios en la base de datos temporal, en el intervalo considerado, por lo que termina la ejecución del programa. El intervalo temporal considerado va desde el tiempo inicial especificado en la base de datos temporal inicial, 0 en este caso, hasta el tiempo dado como argumento en el predicado analizar_objetos/1 del punto anterior.

Podemos decir que la función de main/0 es equivalente al punto cinco del algoritmo anterior, mientras que analiza_objeto/3 realizaría el trabajo de los otros cuatro puntos.

El predicado main/0 y sus predicados asociados pueden verse a continuación:

```
% PROGRAMA

main :- repeat,
      anular_testigo,
      analizar_objetos(1000),
      cerrar_bucle,!.

anular_testigo :- abolish(testigo/0).

analizar_objetos(Tf) :-
       i(O,C),subclase(C,dominio_interior),
       analiza_objeto(O,C,Tf),fail.
analizar_objetos(_).

cerrar_bucle :- not testigo.
```

Este predicado es el único no incluído en el paradigma de programación orientada al objeto. analiza_objeto/3, por el contrario, es un mensaje dirigido a un objeto específico, subclase del dominio interior del sistema. Todos los demás predicados del motor de inferencia se disparan mediante el código de analiza_objeto/3 o mediante mensajes enviados por causa del mismo.

VI.4 ANALISIS DE LA POSIBLE MODIFICACION DE LA BASE DE DATOS TEMPORAL POR UN OBJETO

El predicado analizar_objeto(Objeto,Clase,Tiempo) tiene como objetivo analizar si el objeto especificado tiene alguna posibilidad de generar un nuevo acontecimiento en la base de datos temporal antes del tiempo especificado como final del periodo de estudio del sistema, y de forma que no pueda ser alterado posteriormente por el comportamiento de ningún otro objeto; en caso afirmativo, levanta el testigo de parada anteriormente mencionado.

Un hecho muy importante a destacar aquí es que este predicado realiza la función de los puntos uno al cuatro del algoritmo de la sección VI.2 para cada objeto del sistema. Sin embargo, no es posible hacerlo sin tener en cuenta la respuesta de los demás objetos del sistema a los cambios producidos por este predicado sobre un objeto. La forma en que se ha solucionado este tema es fragmentando la ejecución

del predicado. En efecto, si llamamos a este predicado sobre un objeto dado, se ejecutará hasta donde pueda hacerlo (en tiempo) sin requerir la respuesta de los objetos vecinos a los cambios por él producidos, momento en el que pasará la ejecución a otros objetos. En el próximo ciclo de main/0, cuando se vuelva a llamar al predicado sobre el objeto original, ya se tendrán disponible las respuestas de los objetos vecinos a los cambios por él propuestos, por lo que continuará la ejecución en el punto en el que la dejó hasta que el fenómeno se repita. De esta forma, el predicado analizar_objeto/3 se ejecuta sobre cada objeto hasta completar los puntos uno al cuatro del algoritmo de la sección anterior, aunque para hacerlo pueda requerir un número mayor de ciclos de main/0, debido al necesario intercalamiento de las ejecuciones de analizar_objeto/3 sobre cada objeto. Las interrupciones de ejecución del predicado permiten dividir la ejecución del predicado en pasos, que corresponden aproximadamente a los pasos del algoritmo original del la sección VI.2:

1.-La primera vez que llamamos a este predicado a propósito de un objeto determinado, trata de calcular si existe alguna posibilidad de que este objeto evolucione generando un nuevo estado cualitativo, bajo la hipótesis de que los objetos vecinos no evolucionan. Este resultado se almacena como un hecho del tipo hevent(Objeto,Tiempo). Puesto que el objetivo de este primer paso es calcular t_{min}, que definiremos como el tiempo de los hevent supervivientes en el próximo ciclo de main/0, debe hacerse lo siguiente: Si el hevent calculado corresponde a un tiempo anterior a todos los hevents almacenados en la base de datos temporal correspondientes a otros objetos, se borran todos ellos y queda tan sólo el hevent calculado, y si el hevent calculado corresponde a un tiempo posterior al de algún hevent almacenado en la base de datos temporal, ni siquiera lo guardamos en ella. De esta forma nos aseguramos que los hevent supervivientes para el próximo ciclo de main/0 correspondan al tiempo t_{min}, y hemos realizado el paso 1 del algoritmo de la sección VI.2.

2.- La segunda vez que llamamos a este predicado sobre un objeto dado, pueden ocurrir dos cosas: que exista un hevent/2 del objeto o que no exista. Si existe, indica que el objeto tiene un cambio de estado en t_{min}, y si no existe, indica que el objeto no es relevante en el tiempo considerado y puede ser descartado. La razón de que los objetos con hevent/2 hayan de suceder, es que no existe ningún hevent anterior en el tiempo que los pueda volver inconsistentes (recordemos que un hevent era una hipótesis de suceso que ocurriría si los objetos vecinos al objeto dado no experimentaban cambios antes del tiempo del hevent, luego si no existe ningún

hevent anterior, no pueden experimentar cambios). Por consiguiente, en este paso deberíamos convertir la hipótesis de acontecimiento hevent/2 en un nuevo suceso en la base de datos temporal; sin embargo, queda por considerar el hecho de que las acciones que el nuevo suceso origine sobre los objetos vecinos se vean reflejadas en el mismo momento sobre él mismo, con lo que se requeriría un doble cambio de estado en el mismo momento para este objeto. Este tema se soluciona de la siguiente forma: el nuevo suceso se almacena de forma provisional (con el predicado hevent/3), y se realiza la propagación de acciones sobre objetos vecinos, (por el predicado hacc/3). En el paso siguiente se comprueba si los objetos vecinos han aceptado o no estas acciones, consolidando o modificando en consecuencia el nuevo suceso para su inclusión definitiva en la base de datos temporal.

3.- La tercera vez que llamamos a este predicado se realiza una de las siguientes dos funciones según se hayan recibido o no acciones de los objetos vecinos al objeto considerado. Si no se han recibido acciones, es porque el cambio de estado para este objeto propuesto en el hevent ha sido aceptado por los objetos vecinos, por lo que el hevent puede ya convertirse en un auténtico suceso almacenable en la base de datos. Si se recibe alguna acción , se indica con ello que el nuevo estado del mismo propuesto en hevent no es compatible con el estado de algún objeto vecino y que se requiere algún cambio. Por tanto se alterará el hevent del objeto en consecuencia y se enviarán nuevas acciones a los objetos vecinos informando de la nueva modificación.

Como puede verse, un único ciclo de main/0 no tiene porqué resolver completamente el paso 3, debido a que todo objeto que haya recibido alguna acción proveniente del paso 2 generará nuevas acciones que han de resolverse antes de volver al paso 1, por tanto, el paso 3 se ejecuta de forma cíclica mientras exista alguna acción que haya sido enviada a algún objeto. La forma en que se realiza este ciclo es la siguiente: la ejecución del predicado analiza_objeto/3 permanecerá en el paso 3 para todo ciclo de main/0 mientras exista un solo hacc/3, sea del objeto que sea. Cuando no existe ningún hacc/3, significa que los nuevos estados de los objetos en t_{min} han sido finalmente aceptados por sus vecinos, por lo que puede darse por concluido el ciclo de ejecución y volver al paso 1 para reiniciar el proceso.

El código de analiza_objeto/3 se muestra en las siguientes líneas:

```
% ANALIZAR OBJETO

analiza_objeto(O,C,Tf) :-
```

```
ultimoevent(O,Ti,Estado),
(\+hevent(O,_),
 detectar_inconsistencia(O,C,Ti,Tf,Estado,T),
 colocahev(O,T),colocahac(O,T),
 assert(testigo);
hevent(O,T),
 calculaestado(O,C,Ti,Tf,Estado,T,NEst),
 assert(testigo),
 (\+ hevent(O,T,_),assert(hevent(O,T,NEst)),
       colocahac(O,T);
  hevent(O,T,NE),
  (NEst = NE,quitahac(O,T),
        afirmar(O,T,NEst);
   NEst \= NE,retract(hevent(O,T,NE)),
        assert(hevent(O,T,NEst)),
        colocahac(O,T)))),!.
```

Los tres pasos anteriormente comentados pueden observarse en el código de analiza_objeto/3:

- El paso 1 se ejecuta cuando no existe ningún hevent/2, y consta de los siguientes predicados:

```
detectar_inconsistencia(O,C,Ti,Tf,Estado,T),
colocahev(O,T),colocahac(O,T),
assert(testigo);
```

La función de detectar_inconsistencia es averiguar si el objeto O, cuyo último cambio de estado se produjo en Ti, puede generar un nuevo cambio de estado antes del tiempo de finalización de la simulación (Tf), bajo la hipótesis de que los objetos vecinos no evolucionarán mientras tanto. Si puede hacerlo, se devuelve el tiempo (T) en el que sucederá esta hipótesis de acontecimiento.

El predicado colocahev(O,T) tiene por objeto borrar todos los hevent de la base de datos temporal cuyo tiempo sea mayor que T, y almacenar hevent(O,T) en la misma sólo si no hay ningún hevent con tiempo anterior a T.

El predicado colocahac(O,T) tiene por objeto borrar todas las hipótesis de acción que pudieran haber quedado del ciclo anterior. Podemos decir que tiene una función de seguridad, pues no deberían quedar hipótesis de acción al llegar a este punto en la ejecución del motor de inferencia.

La instrucción final, assert(testigo), levanta el indicador para saber que el ciclo de main/0 no ha terminado aún (puesto que sólo se alcanza este punto si se sabe que al menos un objeto evolucionará en el intervalo de tiempo estudiado).

- El paso 2 del ciclo teórico del motor de inferencia (ver II.4), se realiza también en el mismo predicado analiza_objeto/3, pero únicamente cuando existe hevent/2, y no existe ningún hevent/3 para el objeto especificado (o sea, cuando no se ha calculado aún el nuevo suceso al que corresponde hevent/2), y consta de los siguientes predicados:

```
calculaestado(O,C,Ti,Tf,Estado,T,NEst),
assert(testigo),
(\+ hevent(O,T,_),
assert(hevent(O,T.NEst)),
colocahac(O,T);
```

El predicado calculaestado tiene como objeto calcular el nuevo suceso del objeto O en el tiempo T (dado por hevent/2); el nuevo estado del objeto tras el suceso considerado se almacena en la variable NEst.

La condición not hevent(O,T,_) sucede por definición del paso 2.

assert(hevent(O,T.NEst)) almacena de forma provisional el nuevo suceso calculado, a la espera de que en el paso 3 se confirme de forma definitiva o se modifique según las acciones enviadas por los objetos vecinos.

El predicado colocahac(O,T) envía mensajes hacc/3 a todos los objetos vecinos que sean afectados por el nuevo suceso (O,T,NEst), con objeto de comprobar posteriormente si son aceptados o no estos cambios.

- El paso 3 se ejecuta cuando existe simultáneamente un hevent/2 y un hevent/3 para el objeto considerado. A él corresponde el siguiente código:

```
calculaestado(O,C,Ti,Tf,Estado,T,NEst),
assert(testigo),
...
hevent(O,T,NE),
(NEst = NE,quitahac(O,T),
        afirmar(O,T,NEst);
NEst \= NE,retract(hevent(O,T,NE)),
        assert(hevent(O,T,NEst)),
        colocahac(O,T)))),!.
```

El predicado calculaestado calcula el nuevo suceso del objeto O en el tiempo T, pero tiene en cuenta todos los hacc/3 recibidos de objetos vecinos para hacerlo (este hecho no fue indicado en el paso anterior porque en él aún no existían los hacc/3). El nuevo estado del objeto se almacena en la variable NEst.

El predicado hevent(O,T,NE), aparte de servir de condición para el paso 3, consulta el nuevo estado provisional que fue calculado en el paso anterior para este objeto.

La condición NEst = NE se cumple si los objetos circundantes aceptan el nuevo estado del objeto O, pues indica que el nuevo estado no ha sido modificado por las acciones de los objetos vecinos. En este caso, se llaman a los predicados quitahac/2 y afirmar/3, que tiene por objeto eliminar todas las hacc/3 y convertir hevent(O,T,NEst) en un nuevo suceso de la base de datos temporal.

La condición NEst \= NE indica que algún objeto circundante ha enviado un mensaje hacc/3 indicando una incompatibilidad con el valor actual del nuevo estado. En este caso se sustituye el nuevo estado NE por NEst para adecuarse al hacc/3 problema y se envian (con colocahac/2) hacc/3 a los objetos circundantes para informarles del cambio.

En lo comentado hasta el momento, han aparecido cuatro predicados importantes que pasamos a describir con más detalle:

VI.4.1 El predicado detectar_inconsistencia/6

El predicado **detectar_inconsistencia/6** tiene la misión de enviar un mensaje a un objeto dado del sistema (O) y con el objetivo de averiguar si el objeto dado, cuyo último cambio de estado se produjo en Ti, puede generar un nuevo cambio de estado

antes del tiempo de finalización de la simulación (Tf), bajo la hipótesis de que los objetos vecinos no evolucionarán mientras tanto. Si puede hacerlo, se devuelve el tiempo en el que sucederá esta hipótesis de acontecimiento (T).

Este predicado, por consiguiente, realiza una pregunta cuya respuesta implica consultar tanto la base de datos temporal del sistema como la base de conocimiento de la clase que describe el comportamiento del objeto considerado. Para responder a esta pregunta, de forma análoga al procedimiento de refutación de Robinson, lo que hacemos es negarla. Si añadimos el hecho "El objeto O no evoluciona durante el intervalo de tiempo de estudio (desde el último suceso registrado para este objeto hasta el tiempo de finalización, Tf)" a la base de datos temporal y demostramos que produce una inconsistencia en la misma a partir de un tiempo T determinado, habremos demostrado que debe ocurrir algún suceso en el tiempo T, que solucione la inconsistencia; puesto que trabajamos bajo la hipótesis de que los objetos vecinos no evolucionan, debemos concluir que el suceso ocurre en el tiempo T y en el objeto O. Si, por el contrario, no se produce ninguna inconsistencia, significa que el objeto dado no puede producir un nuevo suceso por sí mismo, por lo que el predicado falla.

La forma de responder a la pregunta que el predicado formula, por tanto, es suponer que no hay ningún nuevo suceso generado por el objeto considerado y explorar el eje temporal hasta encontrar la inconsistencia más antigua en el tiempo que esto supone, o hasta demostrar que no hay ninguna inconsistencia. La implementación de detectar_inconsistencia/6 consiste, como puede verse en el código mostrado a continuación, en un predicado findall que busca todas las posibles inconsistencias temporales que la anterior suposición pueda producir, y un predicado minimo que encuentra aquella que es anterior en el tiempo a todas las demás, devolviendo su tiempo en T. Si no hay ninguna inconsistencia, el predicado findall devolverá en L una lista vacía y el predicado minimo fallará, causando el fallo de detectar_inconsistencia/6.

```
detectar_inconsistencia(O,C,Ti,Tf,E,T) :-
  findall(T,d_inconsistencia(O,C,Ti,Tf,E,T),L),minimo(T,L).
```

El predicado **d_inconsistencia/6** tiene por objeto encontrar (por backtraking) todas las inconsistencias que la suposición anterior pueda producir, y devuelve el tiempo en que sucede cada inconsistencia. Existen dos tipos distintos de inconsistencias: las producidas internamente en un objeto y las producidas por desajuste con el estado de los objetos vecinos. Por ejemplo, en el caso estudio de los muchos tanques

presentado en el capítulo III, una cuba que se está vaciando producirá una inconsistencia interna bajo la suposición de que no genera ningún nuevo suceso en la BDT, puesto que en el momento en el que la cuba se vacíe debe haber un suceso en la BDT que lo indique. Sin embargo, una tubería con un flujo constante gobernada por un grifo del que un suceso en la BDT nos indica que se cerrará en un futuro cercano, producirá una inconsistencia externa bajo la suposición de que no genera ningún nuevo suceso en la BDT, puesto que sabemos que en el momento en el que el grifo se cierra, el flujo en la tubería cambiará y debe haber un nuevo suceso que indique el hecho. Las inconsistencias internas se encuentran mediante el predicado di_interior/6 y las externas mediante el predicado di_exterior/6.

```
d_inconsistencia(O,C,Ti,Tf,E,T) :-
  di_interior(O,C,Ti,Tf,E,T);
  di_exterior(O,C,Ti,Tf,E,T).
```

El predicado **di_interior/6** busca las inconsistencias internas que puedan producirse en el objeto O bajo la asunción de que no necesita nuevos sucesos en la BDT. Para ello llama al predicado di_inte/6, cuya función es encontrar (por backtraking) todas las posibles inconsistencias internas que pueden producirse, y luego llamá a minimo/2 para escojer aquella que sea anterior en el tiempo y devolver en T el tiempo en que se produce.

```
di_interior(O,C,Ti,Tf,E,T) :-
  findall(Tj,di_inte(O,C,Ti,Tf,E,Tj),L),minimo(T,L).
```

El predicado **di_inte/6** tiene por objeto encontrar por backtraking todas las posibles inconsistencias internas que puedan producirse en el objeto O y el tiempo en el que éstas se producen. Las inconsistencias internas de un objeto se producen siempre debidas a un cambio cualitativo en un parámetro de tipo $F(\lambda,t)$, debido a que éstos parámetros pueden encontrarse experimentando un proceso de cambio continuo que se prolonga hasta que se viola alguna condición (por ejemplo, una cola que crece progresivamente acabará encontrando el final de la calle y requiriendo un nuevo suceso del objeto para expresar este hecho; una cuba cuyo nivel disminuye progresivamente acabará vaciándose, etc.). En consecuencia, el predicado di_inte/6 explorará uno por uno todos los parámetros de tipo $F(\lambda,t)$ del objeto considerado, y analizará para cada uno si existe alguna posibilidad de que el parámetro produzca alguna inconsistencia. La sintaxis del predicado di_inte/6 es:

```
di_inte(O,C,Ti,Tf,E,T) :-
  v(O,Var,flt),localize(E,Var,[_,Valor]),
  flt_vera([O,Var],Valor,Di),Di < Tf - Ti,T is Ti + Di.
```

El predicado **v(O,Var,flt)** pregunta el nombre de todas las variables de tipo "flt" (F(λ,t)) que tenga el objeto O (de hecho, simplemente pregunta por el valor del atributo múltiple "flt" del objeto O, hablando en términos de Programación Orientada a Objetos).

El predicado **flt_vera/3** trata de averiguar el próximo cambio cualitativo que tendrá el parámetro considerado por evolución interna. Fallará si su estado actual es estable, y si no lo es, devolverá en T el tiempo de su próximo cambio.

El predicado **di_exterior/6**, de forma análoga al predicado di_interior/6, busca las inconsistencias externas que puedan producirse en el objeto O bajo la asunción de que no necesita nuevos sucesos en la BDT. Para ello llama al predicado di_exte/6, cuya función es encontrar (por backtraking) todas las posibles inconsistencias externas que pueden producirse, y luego llama a minimo/2 para escojer aquella que sea anterior en el tiempo y devolver en T el tiempo en que se produce.

```
di_exterior(O,C,Ti,Tf,E,T) :-
  findall(Tj,di_exte(O,C,Ti,Tf,E,Tj),L),minimo(T,L).
```

El predicado **di_exte/6** tiene por objeto, como acabamos de decir, encontrar por backtraking todas las posibles inconsistencias externas que puedan producirse en el objeto O y el tiempo en el que éstas se producen. Las inconsistencias externas de un objeto se producen siempre debido a que el objeto recibe en un futuro próximo un mensaje de cambio de valor de alguna de sus variables por causa de otro objeto vecino. Al respecto cabe hacer las siguientes observaciones:

a) Un objeto vecino puede afectar el valor de una variable del objeto O tan sólo si hay definida una relación entre ambos objetos que liga esa variable a otra variable del objeto vecino.

b) Dos variables ligadas se ven forzadas a mantener el mismo valor, por lo que un objeto distinto del objeto O puede cambiar el valor de una variable V de este. Esto sucederá si se produce un cambio en el valor de una de sus variables que esté ligada

a la variable V del objeto O (supuesto de que este objeto tenga alguna variable ligada a V).

c) En consecuencia, el predicado di_exte/6 explorará una por una todas las ligaduras entre las variables del objeto O y las de sus objetos vecinos, dadas por las relaciones existentes entre ellos, y analizará para cada una si existe alguna posibilidad de que la variable del objeto vecino sufra algún cambio, hecho que produciría una inconsistencia externa con la suposición de que el objeto O no cambia.

El predicado elmismo(var(O,V),var(O2,V2)) pregunta por todas las variables V del objeto O que tengan alguna ligadura con alguna variable (V2) de algún objeto vecino (O2), y los predicados (event(O2,T,E2);hevent(O2,T,E2)),T > Ti,T < Tf pretenden averiguar si el objeto vecino O2 considerado tiene algún suceso o hipótesis de suceso previstos para el intervalo [Ti,Tf] (desde el último suceso del objeto O hasta el final del período de estudio), si lo tiene, localize(E2,V2,[_,Val2]) localiza el valor Val2 que toma la variable V2 en el nuevo estado E2 del objeto O2, y si no coincide con el valor de la variable V en ese momento, el predicado tiene resultado exitoso, demostrando la existencia de una inconsistencia externa en el tiempo T (tiempo del suceso de O2 que la produjo).

```
di_exte(O,C,Ti,Tf,E,T) :-
 elmismo(var(O,V),var(O2,V2)),
 (event(O2,T,E2);hevent(O2,T,E2)),T > Ti,T < Tf,
 localize(E2,V2,[_,Val2]),
 localize(E,V,[_,Val]),
 (number(Val),number(Val2),Val =\= Val2;
  Val \= Val2).
```

Finalmente, veremos el listado conjunto de todos los predicados explicados, que forman la implementación de la función detectar_inconsistencia/6:

```
% ANALIZA INCONSISTENCIA

detectar_inconsistencia(O,C,Ti,Tf,E,T) :-
 findall(T,d_inconsistencia(O,C,Ti,Tf,E,T),L),minimo(T,L).
```

```
d_inconsistencia(O,C,Ti,Tf,E,T) :-
 di_interior(O,C,Ti,Tf,E,T);
 di_exterior(O,C,Ti,Tf,E,T).

di_interior(O,C,Ti,Tf,E,T) :-
 findall(Tj,di_inte(O,C,Ti,Tf,E,Tj),L),minimo(T,L).

di_inte(O,C,Ti,Tf,E,T) :-
 v(O,Var,flt),localize(E,Var,[_,Valor]),
 flt_vera([O,Var],Valor,Di),Di < Tf - Ti,T is Ti + Di.

di_exterior(O,C,Ti,Tf,E,T) :-
 findall(Tj,di_exte(O,C,Ti,Tf,E,Tj),L),minimo(T,L).

di_exte(O,C,Ti,Tf,E,T) :-
 elmismo(var(O,V),var(O2,V2)),
 (event(O2,T,E2);hevent(O2,T,E2)),T > Ti,T < Tf,
 localize(E2,V2,[_,Val2]),
 localize(E,V,[_,Val]),
 (number(Val),number(Val2),Val =\= Val2;
  Val \= Val2).
```

VI.4.2 El predicado colocahev/2

El predicado **colocahev/2** tiene por función colocar una marca en la BDT indicando que el objeto O producirá un nuevo suceso en el tiempo T bajo la suposición de que los objetos vecinos no evolucionan desde el último suceso de O hasta T. Su objetivo ha sido explicado en el paso 1 del motor de inferencia, y en esencia consiste en una parte del cálculo de la hipótesis de acontecimiento más antiguo que puede producirse en la BDT en un ciclo dado del motor. Para lograr este objetivo, el predicado colocahev(O,T) realiza dos pasos: primero, comprueba si existe alguna marca anterior en la BDT, (hevent(_,T1),T1 < T), en cuyo caso termina, puesto que este hecho indicaría que existe una hipótesis de suceso (de otro objeto) anterior en el tiempo a la considerada, con lo que ésta no puede ser la más primitiva; segundo, si no se cumple la condición anterior, se coloca la marca de hipótesis de nuevo suceso para el objeto O (assert(hevent(O,T))) y se borran todas aquellas marcas hevent/2 que sean posteriores en el tiempo a T, puesto que no tienen posibilidad alguna de ser

las más primitivas en la BDT. A continuación se muestra el listado del predicado considerado:

```
colocahev(O,T) :- hevent(_,T1),T1 < T,!.
colocahev(O,T) :- assert(hevent(O,T)),fail.
colocahev(O,T) :- hevent(O2,T2),T2 > T,retract(hevent(O2,T2)),fail.
colocahev(_,_).
```

VI.4.3 El predicado colocahac/2

El predicado **colocahac/2** tiene por función colocar una marca en la BDT indicando que el objeto O (del que ya sabemos que tendrá un nuevo suceso en T, pero aún ignoramos el nuevo estado resultante), envía mensajes a sus objetos vecinos informando de los cambios sufridos por las variables de O que puedan estar ligadas a variables en los objetos vecinos. Estas marcas se utilizan en el paso 3 del motor de inferencia para saber si el nuevo estado de un objeto puede considerarse definitivo. Cuando estas marcas dejan de existir, indica que todos los objetos han aceptado los nuevos estados de los demás y por tanto no se requieren ulteriores modificaciones de los mismos, momento en el que se estos sucesos se asientan definitivamente en la BDT y comienza otro ciclo del motor de inferencia.

El predicado colocahev(O,T) realiza dos pasos: primero, comprueba si existe alguna marca anterior en la BDT, (hacc(_,T1),T1 < T), en cuyo caso termina, puesto que este hecho indicaría que existe una hipótesis de acción de algún otro objeto anterior en el tiempo a la considerada, y como no podemos hacer razonamiento alguno sobre una hipótesis de acción que no tiene su pasado completamente determinado, no se considera; segundo, si no se cumple la condición anterior, se coloca la marca de hipótesis de nueva acción para el objeto O (assert(hacc(O,T))) y se borran todas aquellas marcas hacc/2 que sean posteriores en el tiempo a T, puesto que, por la razón anterior, no pueden ser procesadas. A continuación se muestra el listado del predicado considerado:

```
colocahac(O,T) :- hacc(_,T1),T1 < T,!.
colocahac(O,T) :- assert(hacc(O,T)),fail.
colocahac(O,T) :- hacc(O2,T2),T2 > T,retract(hacc(O2,T2)),fail.
colocahac(_,_).
```

VI.4.4 El predicado calculaestado/7

El predicado **calculaestado/7** envía un mensaje a un objeto dado del sistema (O) y tiene como finalidad averiguar el nuevo estado en el que el objeto dado se encuentra en el tiempo T, teniendo en cuenta tanto su evolución interna como las acciones externas a las que se ve sometido, bajo la hipótesis de que no se reciben acciones externas en el intervalo [Ti,T). En caso de recibir acciones externas hace el mismo trabajo únicamente en el tiempo T. Para ello utiliza como base el último cambio de estado del objeto O, llamado Estado y producido en Ti, la clase C a la que pertenece el objeto O, el tiempo de finalización del periodo de estudio considerado Tf, y las hipótesis de acción que los objetos vecinos puedan haberle enviado en el tiempo T, que se encuentran en la BDT. El nuevo estado del objeto se devuelve en la variable N3E.

Este predicado, por consiguiente, constituye el núcleo del motor de inferencia, pues engloba todos los procesos necesarios para calcular los nuevos estados a incluir en la BDT. El cálculo del nuevo estado del objeto O, teniendo en cuenta su evolución interna desde Ti (tiempo del último suceso registrado de O) hasta T (tiempo preguntado) y teniendo en cuenta las acciones externas que puedan llegar en T, se realiza dividiendo la tarea en tres partes, que son realizadas por tres predicados distintos:

La primera parte, realizada por el predicado evolucion_interna/4, tiene por objetivo calcular el estado al que llega el objeto O en el tiempo T teniendo únicamente en cuenta la evolución de sus parámetros internos, y este estado se devuelve en la variable NEstado.

La segunda parte, realizada por el predicado consistencia_externa/4, consiste en recoger todas las acciones externas recibidas por el objeto O en el tiempo T como cambios directos de las variables del estado NEstado, obteniendo un nuevo estado llamado N2Estado, que en general es inestable, puesto que el objeto O no tiene porqué aceptar los cambios propuestos por las acciones de sus objetos vecinos.

La tercera parte, realizada por el predicado consistencia_externa/4, tiene por objeto transformar el estado N2Estado, generalmente inestable, en un estado estable que cumpla las reglas internas de comportamiento del objeto, y trate de conservar en lo posible los cambios producidos por las acciones externas; naturalmente, esta última parte puede aceptar o rechazar los cambios producidos por las acciones externas, en cuyo caso se enviará un mensaje de hipótesis de acción al objeto cuya acción haya

sido rechazada (de esto se encarga el predicado colocarhac/2, anteriormente analizado). El nuevo estado definitivo se devuelve en la variable N3E, y el listado del predicado considerado puede verse a continuación:

```
calculaestado(O,C,Ti,Tf,Estado,T,N3E) :-
  D is T - Ti,
  evolucion_interna(O,Estado,D,NEstado),
  consistencia_externa(O,NEstado,T,N2Estado),
  consistencia_interna(O,N2Estado,N3E).
```

Vamos a analizar con más detalle cada uno de estos tres predicados:

El predicado **evolucion_interna/4** tiene por objeto, como acabamos de decir, calcular el estado al que llega el objeto O en el tiempo T teniendo únicamente en cuenta la evolución de sus parámetros internos y sin considerar las acciones externas que recibe en el tiempo T. Este estado estará compuesto por una lista de pares de valores [Variable,Valor], que se obtiene a partir del estado Estado en el tiempo Ti, analizando por separado la evolución que cada variable haya podido experimentar desde Ti hasta T. La razón de que las variables puedan ser consideradas por separado es que no han podido interactuar entre sí en el intervalo [Ti,T), puesto que de haberlo hecho, habría un suceso en la BDT indicándolo entre Ti y T, con lo que Ti ya no correspondería al tiempo del último suceso registrado para O. Por consiguiente, el predicado evolucion_interna/4 se implementa como un predicado recursivo que recorre todos los elementos de la lista Estado, correspondientes a los valores en Ti de las variables, analizando por separado cada variable y pasando a la siguiente. De ahí la estructura:

```
evolucion_interna(<Objeto>,[[<Variable>,<Valor en Ti>]|Resto_Estado],T-Ti,
                  [[<Variable>,<Nuevo Valor en T>]|Resto_Nuevo_Estado]):-
        {calculo de <Nuevo Valor en T>},
        evolucion_interna(<Objeto>,Resto_Estado,T-Ti,Resto_Nuevo_Estado).
evolucion_interna(_,[],_,[]).
```

El nuevo valor que cada variable toma en T depende del tipo de variable de que se trate. Las variables ft no pueden evolucionar entre Ti y T, debido a que para hacerlo precisan de un suceso en dicho intervalo y no lo hay porque de haberlo Ti ya no sería el tiempo del último suceso de O; por tanto, su nuevo valor coincide con el antiguo, lo que representamos por v(O,V,ft),V2l = Vl (si la variable V del objeto O es del tipo

ft, su nuevo valor V2l coincide con el antiguo Vl). Las variables flt, en cambio, sí pueden evolucionar, pero sin causar ni recibir cambios externos entre Ti y T, puesto que de hacerlo precisarían de un nuevo suceso en dicho intervalo, cosa que no ocurre. Se trata pues tan sólo de una evolución lineal de cada variable flt, que se calcula por medio del predicado nst/4. Este predicado, que forma parte del conocimiento de la clase de parámetros flt, tiene por objeto calcular el valor de una variable flt en un momento cualquiera del tiempo, teniendo una función similar al predicado trajectory del ECCC de Shanahan. El valor de la variable considerada en T viene dado por V2l. Un listado completo del predicado evolucion_interna/4 puede verse en las siguientes líneas:

```
evolucion_interna(O,[[V,Vl]|E],D,[[V,V2l]|NE]) :-
  (v(O,V,ft),V2l = Vl;
   v(O,V,flt),nst([O,V],Vl,V2l,D)),!,
  evolucion_interna(O,E,D,NE).
evolucion_interna(_,[],_,[]).
```

El predicado **consistencia_externa/4** tiene por objeto recoger todas las acciones externas recibidas por el objeto O en el tiempo T, como cambios directos de las variables del estado obtenido en el paso anterior, obteniendo un nuevo estado que en general es inestable, puesto que el objeto O no tiene porqué aceptar los cambios propuestos por las acciones de sus objetos vecinos. El proceso de cálculo se realiza, análogamente al caso anterior, a nivel de variable; por consiguiente, el predicado consistencia_externa/4 tiene forma recursiva, procesando cada variable por separado en cada recursión. El proceso realizado con cada variable es el siguiente: primero, se busca si la variable V del objeto O está ligada a alguna variable V2 de algún objeto vecino O2, hecho que se realiza con el predicado elmismo/2; si no está ligada, se concluye que la variable permanece inalterada (V2l = Vl) y se pasa a otra variable. Si está ligada, se comprueba si el objeto O2 al que está ligada tiene algún suceso o hipótesis de suceso en T, en cuyo caso se examina el nuevo valor de la variable V2 (localize(ES,V2,[_,V2l]), que se sustituye directamente por el valor de V, ya que si son diferentes, debe cambiarse V como respuesta a la acción de O2, y si son iguales, da igual sustituir V por V2 que no hacerlo. El listado completo del predicado consistencia_externa/4 se muestra a continuación:

```
consistencia_externa(O,[[V,Vl]|E],T,[[V,V2l]|E2]) :-
  (elmismo(var(O,V),var(O2,V2)),
     (event(O2,T2,ES);hevent(O2,T2,ES)),T2 =:= T,
```

```
    localize(ES,V2,[_,V2l]);
  V2l = Vl),
  !,consistencia_externa(O,E,T,E2).
consistencia_externa(O,[],T,[]).
```

El predicado **consistencia_interna/3** tiene por objeto transformar el estado obtenido en el predicado consistencia_externa/4, generalmente inestable, en un estado estable que cumpla las reglas internas de comportamiento del objeto, y trate de conservar en lo posible los cambios producidos por las acciones externas. Estos cambios pueden ser aceptados o rechazados, según indique el comportamiento interno del objeto. Si son rechazados se enviará un mensaje de hipótesis de acción al objeto cuya acción haya sido rechazada, pero esta función ya no es realizada por este predicado, tal y como se dijo anteriormente.

El predicado realiza su trabajo por medio de un bucle de propagación interna de cambios de variables del objeto basados en el conocimiento sobre su comportamiento. Este bucle toma el estado supuesto para el objeto (que en su primera iteración será el estado obtenido en el predicado consistencia_externa/4), analiza si cumple las reglas de comportamiento del objeto, y si encuentra alguna que no se cumpla, se modifica ligeramente el estado para que la cumpla, tratando en lo posible de no violar las reglas de comportamiento que ya se cumplían; el nuevo estado se emplea como base en una nueva iteración del bucle, y el proceso continúa hasta que el estado sea estable (es decir, que el estado obtenido no viole ninguna regla de comportamiento interno).

La forma como este algoritmo es llevado a cabo es la siguiente: el predicado baja primero un testigo necesario para encontrar la condición de parada del bucle (abolish(con_cyc/0)), coloca después el estado en el que se encuentra el objeto en ese momento (abolish(con_ins/1),assert(con_ins(E1))), llama al predicado cons_inte(O), que realiza el bucle, y por último consulta el hecho con_ins(E2), que contiene el estado final del objeto, que debe ser estable (o no habría terminado el bucle). El nuevo estado definitivo se devuelve en la variable E2, y el listado del predicado considerado puede verse a continuación:

```
consistencia_interna(O,E1,E2) :-
  abolish(con_ins/1),abolish(con_cyc/0),assert(con_ins(E1)),
  cons_inte(O),con_ins(E2).
```

El predicado **cons_inte/1** forma el bucle comentado en el apartado anterior, y su único objeto es formar la instrucción de bucle sobre el predicado cons_inte2/1, que es el cuerpo del bucle. El predicado cons_inte2/1 se evalúa a verdad si el bucle aún no ha terminado y es falso si el bucle termina, por lo que la misión de cons_inte/1 es repetir indefinidamente cons_inte2 mientras sea cierto, tal y como puede verse a continuación:

```
cons_inte(O) :- repeat,not cons_inte2(O),!.
```

El predicado **cons_inte2/1** forma el cuerpo del bucle, y tiene por objeto comprobar la consistencia del estado actual del objeto, almacenado en el hecho con_ins(E), con todas las reglas de comportamiento del objeto, accesibles a través del predicado causa/4. Si son consistentes, el predicado falla, causando la terminación del bucle, si no lo son, se encuentra un nuevo estado que corrija la inconsistencia encontrada y se levanta el testigo con_cyc/0 para que el predicado sea cierto y el bucle continue ejecutándose. La función del predicado se realiza de la siguiente forma: un predicado causa/4 explora por backtraking todas las reglas de comportamiento del objeto O. Para cada regla, existe un atributo Cond que describe una condición cuyo cumplimiento por parte del estado actual del objeto indica que la regla de comportamiento ha sido violada. Se llama al hecho con_ins(E) para averiguar el estado actual del objeto (E), y se llama al predicado ci_condicion/2 con la condición y el estado, para saber si la regla considerada es violada por el estado actual del objeto. Si no lo es, el predicado falla y por backtraking se pasa a buscar otra regla de comportamiento; y si lo es, entonces se llama al predicado ci_calcula/3 para encontrar un nuevo valor de la variable Var que produjo la inconsistencia, de forma que sea consistente. El resto de predicados tienen por objeto sustituir el valor antiguo de la variable dada en el estado inconsistente por el nuevo valor calculado, obteniendo un nuevo estado Ep que se coloca en con_ins/1 como el nuevo estado propuesto para el objeto O. Entonces se levanta la bandera con_cyc y termina el predicado, forzando con ello a continuar el bucle. El listado del predicado es mostrado a continuación:

```
cons_inte2(O) :-
        causa(O,Cond,Var,Valor),con_ins(E),
        [!ci_condicion(E,Cond),
          ci_calcula(E,Valor,V2),
          localize(E,Var,[_,V1]),
          (number(V1),number(V2),V1 =\= V2;
```

```
        V1 \= V2),
        sustituye(E,Var,[Var,V2],Ep),
        abolish(con_ins/1),abolish(con_cyc/0),
        assert(con_ins(Ep)),assert(con_cyc)!],fail.
cons_inte2(O) :- con_cyc,abolish(con_cyc/0),!.
```

El predicado **ci_condicion/2** tiene por objeto comprobar si el estado actual del objeto (E) cumple o no una condición Cond dada. El cumplimiento de dicha condición indicaría la existencia de una inconsistencia con la regla de comportamiento del objeto a la que Cond corresponde. Las condiciones se componen de conjunciones o disyunciones entre relaciones sobre valores de parámetros del objeto considerado. Las relaciones permitidas entre variables son igual(V1,V2), mayor(V1,V2), nomayor(V1,V2) y distinto(V1,V2), y la implementación del predicado puede verse en las siguientes líneas:

```
ci_condicion(E,y(C1,C2)) :- !,ci_condicion(E,C1),ci_condicion(E,C2).
ci_condicion(E,o(C1,C2)) :- !,(ci_condicion(E,C1);ci_condicion(E,C2)).
ci_condicion(E,igual(V1,V2)) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),
                        (number(A),number(B),A =:= B;
                         A = B).
ci_condicion(E,mayor(V1,V2)) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),
                        number(A),number(B),A - B > 1e-8.
ci_condicion(E,nomayor(V1,V2)) :- !,not ci_condicion(E,mayor(V1,V2)).
ci_condicion(E,distinto(V1,V2)) :- \+ ci_condicion(E,igual(V1,V2)),!.
```

El predicado **ci_valor/3** tiene por objeto calcular el valor de la variable o constante dada. Una constante se expresa en el formato v(Constante), por ejemplo v(2) ó v(hola), y tiene como valor el argumento de v/1, mientras que una variable se expresa por su nombre, y tiene como valor el valor que tenga en el estado considerado, que se localiza mediante el predicado localize/3, como puede verse a continuación:

```
ci_valor(_,v(A),A) :- !.
ci_valor(E,V,A) :- localize(E,V,[_,A]).
```

El predicado **ci_calcula/3** tiene por objeto evaluar una expresión matemática sobre variables del objeto, para ello descompone la expresión en sus operaciones componentes y realiza cada una por separado. Puesto que se trata de variables

cualitativas, para realizar estas operaciones se necesitan las tablas de valores cualitativos que las describen. Estas tablas no se consideran aquí puesto que en general forman parte del conocimiento de la clase a la que pertenece el objeto en cuestión, y se llaman cuando sean necesarias. Sin embargo, y puesto que el formalismo (λ,t) no está limitado a considerar únicamente valores cualitativos, sino que permite tratar también valores cuantitativos (aunque no sean el objeto fundamental del formalismo), se incluyen por defecto en la definición de ci_calcula/3 las cuatro operaciones matemáticas básicas sobre valores cuantitativos: suma, resta, multiplicación y división, como puede verse en las siguientes líneas:

```
ci_calcula(E,V1*V2,C) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),C is A*B.
ci_calcula(E,V1/V2,C) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),C is A/B.
ci_calcula(E,V1+V2,C) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),C is A+B.
ci_calcula(E,V1-V2,C) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),C is A-B.
ci_calcula(E,V,A) :- ci_valor(E,V,A).
```

El listado completo de los predicados que componen la función calculaestado/7 puede verse a continuación:

```
% CALCULA ESTADO EN TIEMPO T

calculaestado(O,C,Ti,Tf,Estado,T,N3E) :-
  D is T - Ti,
  evolucion_interna(O,Estado,D,NEstado),
  consistencia_externa(O,NEstado,T,N2Estado),
  consistencia_interna(O,N2Estado,N3E).

evolucion_interna(O,[[V,V1]|E],D,[[V,V2I]|NE]) :-
  (v(O,V,ft),V2I = V1;
   v(O,V,flt),nst([O,V],V1,V2I,D)),!,
  evolucion_interna(O,E,D,NE).
evolucion_interna(_,[],_,[]).

consistencia_externa(O,[[V,V1]|E],T,[[V,V2I]|E2]) :-
  (elmismo(var(O.V),var(O2,V2)),
     (event(O2,T2.ES);hevent(O2,T2,ES)),T2 =:= T,
     localize(ES,V2,[_,V2I]);
   V2I = V1),
```

```
  !,consistencia_externa(O,E,T,E2).
consistencia_externa(O,[],T,[]).

consistencia_interna(O,E1,E2) :-
  abolish(con_ins/1),abolish(con_cyc/0),assert(con_ins(E1)),
  cons_inte(O),con_ins(E2).

cons_inte(O) :- repeat,not cons_inte2(O),!.

cons_inte2(O) :-
  causa(O,Cond,Var,Valor),con_ins(E),
  [!ci_condicion(E,Cond),
   ci_calcula(E,Valor,V2),
   localize(E,Var,[_,V1]),
   (number(V1),number(V2),V1 =\= V2;
    V1 \= V2),
   sustituye(E,Var,[Var,V2],Ep),
   abolish(con_ins/1),abolish(con_cyc/0),
   assert(con_ins(Ep)),assert(con_cyc)!],fail.
cons_inte2(O) :- con_cyc,abolish(con_cyc/0),!.

ci_condicion(E,y(C1,C2)) :- !,ci_condicion(E,C1),ci_condicion(E,C2).
ci_condicion(E,o(C1,C2)) :- !,(ci_condicion(E,C1);ci_condicion(E,C2)).
ci_condicion(E,igual(V1,V2)) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),
                        (number(A),number(B),A =:= B;
                         A = B).
ci_condicion(E,mayor(V1,V2)) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),
                        number(A),number(B),A - B > 1e-8.
ci_condicion(E,nomayor(V1,V2)) :- !,not ci_condicion(E,mayor(V1,V2)).
ci_condicion(E,distinto(V1,V2)) :- \+ ci_condicion(E,igual(V1,V2)),!.

ci_valor(_,v(A),A) :- !.
ci_valor(E,V,A) :- localize(E,V,[_,A]).

ci_calcula(E,V1*V2,C) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),C is A*B.
ci_calcula(E,V1/V2,C) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),C is A/B.
ci_calcula(E,V1+V2,C) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),C is A+B.
ci_calcula(E,V1-V2,C) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),C is A-B.
```

```
ci_calcula(E,V,A) :- ci_valor(E,V,A).
```

VI.5 CONCLUSIONES

A lo largo de este capítulo hemos podido ver la estructura del motor de inferencia del formalismo (λ,t) en Prolog, cuyo listado completo aparece en el anexo D. La principal conclusión que cabe extraer del motor de inferencia del formalismo es la gran eficiencia computacional que presenta. Es perfectamente capaz de manejar sistemas con gran número de objetos componentes a una velocidad considerablemente elevada, lo que nos plantea el pensar en serio sobre las posibilidades de implementación de una arquitectura de sistemas de control de procesos como la planteada en el capítulo I.

Conclusiones

El formalismo (λ,t) se planteo en la introducción de esta memoria con el objetivo de ampliar el conjunto de sistemas de control de procesos a los que son aplicables los paradigmas de razonamiento cualitativo desarrollados en la literatura. El formalismo utiliza las ideas fundamentales de los paradigmas de DeKleer, Forbus y Kuipers, permitiendo representar parámetros multidimensionales y permitiendo asociar una referencia temporal a los comportamientos que tienen una importancia destacada en la evolución del sistema.

Las aplicaciones que se han descrito en este trabajo tanto para el caso de sistemas con flujos discretos como para sistemas con flujos continuos, permiten asegurar la utilidad práctica del formalismo. Los resultados obtenidos y el poco tiempo de ejecución empleado en conseguirlos hacen que podamos pensar en la utilización de la modelización de un sistema de control con el formalismo (λ,t) dentro de los distintos papeles reflejados en la arquitectura funcional presentada en el tema I.

Es de destacar que a pesar de la naturaleza no determinista del motor de inferencia del formalismo (λ,t) y a pesar de haber sido aplicado a problemas de una cierta entidad (como es el caso del control de tráfico urbano), el tiempo de ejecución empleado es muy bajo. Además el crecimiento lineal de la complejidad del método empleado, hace que se pueda extrapolar su utilidad a problemas de gran envergadura, en los que sea necesario suministrar respuesta en cortos espacios de tiempo. Por otra parte, hay que destacar que el funcionamiento del motor de inferencia es fácilmente paralelizable, por lo que queda abierta la puerta al estudio de este tema.

El formalismo (λ,t) responde a los objetivos marcados al comienzo de esta memoria, pudiéndose considerar en cierta forma como una generalización de paradigmas presentados en la literatura. En particular es destacable las propiedades de la base de

datos temporal que se genera, que han sido estudiadas a lo largo de la memoria. Por otra parte es de destacar que algunos autores han resaltado la idoneidad del formalismo (λ,t) e incluso lo han adoptado en sus trabajos [Wild, 92], [Sauthier, 92], [Sugimoto,92].

En otro orden de cosas, es necesario mencionar que la declaratividad del formalismo (λ,t) queda encapsulada incluso por la utilización de un lenguaje declarativo como el PROLOG. En la implementacion independiente del dominio que se ha presentado en esta memoria, se puede observar claramente esta perdida de declarativiad, por lo que hemos investigado los posibles requerimientos que pueden plantearse a un lenguaje de programación para que soporte de una forma declarativa y eficiente modelizaciones como las presentadas. Así, hemos concluido con la idoneidad de disponer de un lenguaje declarativo orientado a objetos y basado en técnicas de programación lógica basada en restricciones. La especificación de este lenguaje, que se ha hecho dentro del proyecto ESPRIT II EQUATOR, recogiendo los requerimientos presentados en esta memoria, esta completada, pero sin embargo no se ha comenzado su implementacion, tema en el que creemos que se debe trabajar en corto espacio de tiempo.

Por ultimo remarcar que el trabajo relacionado con el formalismo (λ,t) no queda cerrado en esta memoria. La generalidad de su formulación abre unos interesantes campos de aplicación. Por otra parte se plantean temas de trabajo relacionados entre los que cabe destacar los relacionados con la paralelizacion, con la implementacion del lenguaje antes mencionado, con la introducción de mayor indeterminismo en el formalismo, y con la granularidad de agregación en el razonamiento. En este momento se están abordando algunos de estos trabajos, esperando obtener buenos resultados.

Anexo A

REPRESENTACION DE UN SISTEMA DE TRAFICO URBANO CON EL PARADIGMA (λ,T).

```
% **REPRESENTACION ORIENTADA A VARIABLES

% *CLASES DE VARIABLES

clase(a).
clase(f).
clase(ft).
clase(flt).

% *INSTANCIAS DE VARIABLES

clase(dominio).

clase(dominio_externo,[dominio]).
clase(dominio_interno,[dominio]).

clase(entradas,[dominio_externo]).
clase(salidas,[dominio_externo]).
clase(controladores,[dominio_externo]).

clase(calles,[dominio_interno]).
clase(cruces,[dominio_interno]).
```

```
%   *DESCRIPCION DEL SISTEMA

instancia(e1,entradas).
instancia(e2,entradas).
instancia(s1,salidas).
instancia(s2,salidas).
instancia(l1,calles).
instancia(l2,calles).
instancia(l3,calles).
instancia(l4,calles).
instancia(ifis1,cruces).
instancia(icon1,controladores).

---------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% PREDICADOS BASEDATOS
prepararbasedatos :- events,times,attrib.

attrib :- aa(A,B,C),assert(a(A,B,C)),fail.
attrib.
events :- e(A,B,C),assert(event(A,B,C)),event2(B),fail.
events.
times :- t(A,B),assert(time(A,B)),fail.
times.

event2(C) :- not(event(A)),assert(event(C)).
event2(C) :- event(A),A < C,retract(event(A)),assert(event(C)).
event2(C) :- event(A),A >= C.

% BASE DATOS

c(trafficdensity1,[valuedomains]).
aa(trafficdensity1,values,[free,congestive]).
aa(trafficdensity1,free,[d1,d2,d3,d4]).
aa(trafficdensity1,congestive,[d5,d6,d7,stop]).
```

```
c(trafficdensity2,[trafficdensity1]).
aa(trafficdensity2,values,[d1,d2,d3,d4,d5,d6,d7,stop]).      7)3
aa(trafficdensity2,d1,[70,0,0]).
aa(trafficdensity2,d2,[60,0.03,1.8]).
aa(trafficdensity2,d3,[50,0.07,3.5]).
aa(trafficdensity2,d4,[40,0.1,4]).
aa(trafficdensity2,d5,[30,0.13,3.9]).
aa(trafficdensity2,d6,[20,0.15,3]).
aa(trafficdensity2,d7,[10,0.17,1.7]).
aa(trafficdensity2,stop,[0,0.2,0]).
aa(trafficdensity2,flowfree,[[d1,0,0.9],[d2,0.9,2.65],[d3,2.65,3.75],[d4,3.75,4.3]]).
aa(trafficdensity2,flowcong,[[stop,0,0.85],[d7,0.85,2.35],[d6,2.35,3.45],[d5,3.45,4.3]
]).
aa(trafficdensity2,zeroflow,d1).
aa(trafficdensity2,zeroflow,stop).

c(lineartrafficdensity,[trafficdensity2,flt]).
aa(lineartrafficdensity,borderspeed,[D,E,F]) :- borspeed(D,E,F).
aa(lineartrafficdensity,expandir,[C,D,d4]) :- exp(C,D).

%CONTROL
c(control,[outerdomain]).

i(control1,control).
aa(control1,ciclo,[1,80]).
aa(control1,ciclo,[2,100]).
aa(control1,ciclo,[3,120]).
e(control1,1,[[cci1,newcycle,2],[icon1,newcoord,0],[icon1,newsplit,5]]).
t(1,[0,0,0]).


%ENTRADAS
c(input,[outerdomain]).
i(e1,input).
e(e1,2,d4).
t(2,[0,0,0]).

i(e2,input).
```

```
e(e2,3,d4).
t(3,[0,0,0]).

%SALIDAS
c(output,[innerdomain]).
i(s1,output).
e(s1,4,[state,d1]).
t(4,[0,0,0]).

i(s2,output).
e(s2,5,[state,d1]).
t(5,[0,0,0]).

%CCI
c(area,[outerdomain]).
i(cci1,area).
e(cci1,6,[[newcycle,2],[state,100,2],[pulse,2]]).
t(6,[0,0,0]).

%CONT       7)3  c(cont,[innerdomain]).
i(icon1,cont).
aa(icon1,cci,cci1).
aa(icon1,fase,[1,[l1,l3,50],[l1,l4,50]]).
aa(icon1,fase,[2,[l2,l3,50],[l2,l4,50]]).
aa(icon1,reparto,[1,100,900]).
aa(icon1,reparto,[2,200,800]).
aa(icon1,reparto,[3,300,700]).
aa(icon1,reparto,[4,400,600]).
aa(icon1,reparto,[5,500,500]).
aa(icon1,reparto,[6,600,400]).
aa(icon1,reparto,[7,700,300]).
aa(icon1,reparto,[8,800,200]).
e(icon1,7,[[pulse,2],[newcoord,0],[newsplit,5],
    [state,2,[0,0,0],0,5,[[1,50],[2,50]]],[phase,1]]).
t(7,[0,0,0]).

%CRUCE
c(cross,[innerdomain]).
```

```
i(ifis1,cross).
aa(ifis1,controlador,icon1).
e(ifis1,8,[[state,1,[[l2,d1],[l1,d1]],[[l4,d1],[l3,d1]]]]).
t(8,[0,0,0]).

%LINK
c(link,[innerdomain]).

i(l1,link).
aa(l1,length,100).
aa(l1,lanes,2).
e(l1,9,[[state,[d4,0.0],[d1,0,100]]]).
t(9,[0,0,0]).

i(l2,link).
aa(l2,length,100).
aa(l2,lanes,2).
e(l2,10,[[state,[d4,0,0],[d1,0,100]]]).
t(10,[0,0,0]).

i(l3,link).
aa(l3,length,100).
aa(l3,lanes,2).
e(l3,11,[[state,[d1.0,100]]]).
t(11,[0,0,0]).

i(l4,link).
aa(l4,length,100).
aa(l4,lanes,2).
e(l4,12,[[state,[d1.0,100]]]).
t(12,[0,0,0]).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%RELACIONES
c(relacion).

c(entradacalle,[relacion]).     7)3
i([e1,l1],entradacalle).
```

```
e([e1,l1],13,d4).
t(13,[0,0,0]).

i([e2,l2],entradacalle).
e([e2,l2],14,d4).
t(14,[0,0,0]).

c(callecruce,[relacion]).

i([l1,ifis1],callecruce).
e([l1,ifis1],15,d1).
t(15,[0,0,0]).

i([l2,ifis1],callecruce).
e([l2,ifis1],16,d1).
t(16,[0,0,0]).

c(crucecalle,[relacion]).

i([ifis1,l3],crucecalle).
e([ifis1,l3],17,d1).
t(17,[0,0,0]).

i([ifis1,l4],crucecalle).
e([ifis1,l4],18,d1).
t(18,[0,0,0]).

c(callesalida,[relacion]).

i([l3,s1],callesalida).
e([l3,s1],19,d1).
t(19,[0,0,0]).

i([l4,s2],callesalida).
e([l4,s2],20,d1).
t(20,[0,0,0]).

c(contcruce,[relacion]).
```

```
i([icon1,ifis1],contcruce).
e([icon1,ifis1],21,[]).
t(21,[0,0,0]).

c(ccicont,[relacion]).

i([cci1,icon1],ccicont).
e([cci1,icon1],22,[]).
t(22,[0,0,0]).

c(controlcci,[relacion]).

i([control1,cci1],controlcci).
e([control1,cci1],23,[]).        7)3  t(23,[0,0,0]).

c(controlcont,[relacion]).
i([control1,icon1],controlcont).
e([control1,icon1],24,[]).
t(24,[0,0,0]).
```

Anexo B

REPRESENTACION DE UN SISTEMA DE CUBAS CON EL PARADIGMA (λ,T)

```
% DESCRIPCION PROBLEMA

% OBJETOS

c(tuberia,[dominio_interior]).

i(tuberia1,tuberia).
i(tuberia2,tuberia).
i(tuberia3,tuberia).
i(tuberia4,tuberia).
i(tuberia5,tuberia).
i(tuberia6,tuberia).
i(tuberia7,tuberia).

c(cuba,[dominio_interior]).

i(cuba1,cuba).
i(cuba2,cuba).
i(cuba3,cuba).
i(cuba4,cuba).
i(cuba5,cuba).
i(cuba6,cuba).
```

```
c(grifo,[dominio_exterior]).

i(grifo1,grifo).
i(grifo2,grifo).
i(grifo3,grifo).
i(grifo4,grifo).
i(grifo5,grifo).
i(grifo6,grifo).
i(grifo7,grifo).
i(grifo8,grifo).

% VARIABLES

v(tuberia1,entrada_cuba6,ft).
v(tuberia1,presion_cuba6,ft).
v(tuberia1,salida_cuba1,ft).
v(tuberia1,salida_cuba2,ft).
v(tuberia1,estado_grifo1,ft).
v(tuberia1,estado_grifo2,ft).

v(tuberia2,entrada_cuba1,ft).
v(tuberia2,presion_cuba1,ft).
v(tuberia2,salida_cuba3,ft).
v(tuberia2,estado_grifo3,ft).

v(tuberia3,entrada_cuba2,ft).
v(tuberia3,presion_cuba2,ft).
v(tuberia3,salida_cuba3,ft).
v(tuberia3,estado_grifo4,ft).

v(tuberia4,entrada_cuba3,ft).
v(tuberia4,presion_cuba3,ft).
v(tuberia4,salida_cuba4,ft).
v(tuberia4,estado_grifo5,ft).

v(tuberia5,entrada_cuba3,ft).
v(tuberia5,presion_cuba3,ft).
v(tuberia5,salida_cuba5,ft).
```

v(tuberia5,estado_grifo6,ft).

v(tuberia6,entrada_cuba4,ft).
v(tuberia6,presion_cuba4,ft).
v(tuberia6,salida_cuba6,ft).
v(tuberia6,estado_grifo7,ft).

v(tuberia7,entrada_cuba5,ft).
v(tuberia7,presion_cuba5,ft).
v(tuberia7,salida_cuba6,ft).
v(tuberia7,estado_grifo8,ft).

v(grifo1,estado,ft).

v(grifo2,estado,ft).

v(grifo3,estado,ft).

v(grifo4,estado,ft).

v(grifo5,estado,ft).

v(grifo6,estado,ft).

v(grifo7,estado,ft).

v(grifo8,estado,ft).

v(cuba1,entrada_tuberia1,ft).
v(cuba1,salida_tuberia2,ft).
v(cuba1,presion,ft).
v(cuba1,cantidad,flt).

v(cuba2,entrada_tuberia1,ft).
v(cuba2,salida_tuberia3,ft).
v(cuba2,presion,ft).
v(cuba2,cantidad,flt).

```
v(cuba3,entrada_tuberia2,ft).
v(cuba3,entrada_tuberia3,ft).
v(cuba3,salida_tuberia4,ft).
v(cuba3,salida_tuberia5,ft).
v(cuba3,presion,ft).
v(cuba3,cantidad,flt).

v(cuba4,entrada_tuberia4,ft).
v(cuba4,salida_tuberia6,ft).
v(cuba4,presion,ft).
v(cuba4,cantidad,flt).

v(cuba5,entrada_tuberia5,ft).
v(cuba5,salida_tuberia7,ft).
v(cuba5,presion,ft).
v(cuba5,cantidad,flt).

v(cuba6,entrada_tuberia6,ft).
v(cuba6,entrada_tuberia7,ft).
v(cuba6,salida_tuberia1,ft).
v(cuba6,presion,ft).
v(cuba6,cantidad,flt).

% COMPORTAMIENTO

causa(tuberia1,y(igual(estado_grifo1,v(ST)),
     y(mayor(presion_cuba6,v(X)),
       distinto(salida_cuba1,v(X)))),salida_cuba1,v(X)) :-
  (ST = abierto,X = 100;
   ST = trescuartos, X = 75;
   ST = medio, X = 50;
   ST = cuarto, X = 25;
   ST = cerrado, X = 0).
causa(tuberia1,y(igual(estado_grifo2,v(ST)),
     y(mayor(presion_cuba6,v(X)),
       distinto(salida_cuba2,v(X)))),salida_cuba2,v(X)) :-
  (ST = abierto,X = 100;
   ST = trescuartos, X = 75;
```

```
   ST = medio, X = 50;
   ST = cuarto, X = 25;
   ST = cerrado, X = 0).
causa(tuberia1,y(igual(estado_grifo1,v(ST)),
      y(nomayor(presion_cuba6,v(X)),distinto(salida_cuba1,presion_cuba6))),
     salida_cuba1,presion_cuba6) :-
  (ST = abierto,X = 100;
   ST = trescuartos, X = 75;
   ST = medio, X = 50;
   ST = cuarto, X = 25;
   ST = cerrado, X = 0).
causa(tuberia1,y(igual(estado_grifo2,v(ST)),
      y(nomayor(presion_cuba6,v(X)),distinto(salida_cuba2,presion_cuba6))),
     salida_cuba2,presion_cuba6) :-
  (ST = abierto,X = 100;
   ST = trescuartos, X = 75;
   ST = medio, X = 50;
   ST = cuarto, X = 25;
   ST = cerrado, X = 0).
causa(tuberia1,distinto(entrada_cuba6,salida_cuba1+salida_cuba2),
           entrada_cuba6,salida_cuba1+salida_cuba2).

causa(tuberia2,y(igual(estado_grifo3,v(ST)),
     y(mayor(presion_cuba1,v(X)),
       distinto(entrada_cuba1,v(X)))),entrada_cuba1,v(X)) :-
  (ST = abierto,X = 100;
   ST = trescuartos, X = 75;
   ST = medio, X = 50;
   ST = cuarto, X = 25;
   ST = cerrado, X = 0).
causa(tuberia2,y(igual(estado_grifo3,v(ST)),
      y(nomayor(presion_cuba1,v(X)),distinto(entrada_cuba1,presion_cuba1))),
     entrada_cuba1,presion_cuba1) :-
  (ST = abierto,X = 100;
   ST = trescuartos, X = 75;
   ST = medio, X = 50;
   ST = cuarto, X = 25;
   ST = cerrado, X = 0).
```

```
causa(tuberia2,distinto(entrada_cuba1,salida_cuba3),
          salida_cuba3,entrada_cuba1).

causa(tuberia3,y(igual(estado_grifo4,v(ST)),
     y(mayor(presion_cuba2,v(X)),
      distinto(entrada_cuba2,v(X)))),entrada_cuba2,v(X)) :-
  (ST = abierto,X = 100;
   ST = trescuartos, X = 75;
   ST = medio, X = 50;
   ST = cuarto, X = 25;
   ST = cerrado, X = 0).
causa(tuberia3,y(igual(estado_grifo4,v(ST)),
      y(mayor(v(X),presion_cuba2),distinto(entrada_cuba2,presion_cuba2))),
     entrada_cuba2,presion_cuba2) :-
  (ST = abierto,X = 100;
   ST = trescuartos, X = 75;
   ST = medio, X = 50;
   ST = cuarto, X = 25;
   ST = cerrado, X = 0).
causa(tuberia3,distinto(entrada_cuba2,salida_cuba3),
          salida_cuba3,entrada_cuba2).

causa(tuberia4,y(igual(estado_grifo5,v(ST)),
     y(mayor(presion_cuba3,v(X)),
      distinto(entrada_cuba3,v(X)))),entrada_cuba3,v(X)) :-
  (ST = abierto,X = 100;
   ST = trescuartos, X = 75;
   ST = medio, X = 50;
   ST = cuarto, X = 25;
   ST = cerrado, X = 0).
causa(tuberia4,y(igual(estado_grifo5,v(ST)),
      y(mayor(v(X),presion_cuba3),distinto(entrada_cuba3,presion_cuba3))),
     entrada_cuba3,presion_cuba3) :-
  (ST = abierto,X = 100;
   ST = trescuartos, X = 75;
   ST = medio, X = 50;
   ST = cuarto, X = 25;
   ST = cerrado, X = 0).
```

```
causa(tuberia4,distinto(entrada_cuba3,salida_cuba4),
            salida_cuba4,entrada_cuba3).

causa(tuberia5,y(igual(estado_grifo6,v(ST)),
     y(mayor(presion_cuba3,v(X)),
       distinto(entrada_cuba3,v(X)))),entrada_cuba3,v(X)) :-
  (ST = abierto,X = 100;
   ST = trescuartos, X = 75;
   ST = medio, X = 50;
   ST = cuarto, X = 25;
   ST = cerrado, X = 0).
causa(tuberia5,y(igual(estado_grifo6,v(ST)),
      y(mayor(v(X),presion_cuba3),distinto(entrada_cuba3,presion_cuba3))),
     entrada_cuba3,presion_cuba3) :-
  (ST = abierto,X = 100;
   ST = trescuartos, X = 75;
   ST = medio, X = 50;
   ST = cuarto, X = 25;
   ST = cerrado, X = 0).
causa(tuberia5,distinto(entrada_cuba3,salida_cuba5),
            salida_cuba5,entrada_cuba3).

causa(tuberia6,y(igual(estado_grifo7,v(ST)),
     y(mayor(presion_cuba4,v(X)),
       distinto(entrada_cuba4,v(X)))),entrada_cuba4,v(X)) :-
  (ST = abierto,X = 100;
   ST = trescuartos, X = 75;
   ST = medio, X = 50;
   ST = cuarto, X = 25;
   ST = cerrado, X = 0).
causa(tuberia6,y(igual(estado_grifo7,v(ST)),
      y(mayor(v(X),presion_cuba4),distinto(entrada_cuba4,presion_cuba4))),
     entrada_cuba4,presion_cuba4) :-
  (ST = abierto,X = 100;
   ST = trescuartos, X = 75;
   ST = medio, X = 50;
   ST = cuarto, X = 25;
   ST = cerrado, X = 0).
```

```
causa(tuberia6,distinto(entrada_cuba4,salida_cuba6),
          salida_cuba6,entrada_cuba4).
causa(tuberia7,y(igual(estado_grifo8,v(ST)),
    y(mayor(presion_cuba5,v(X)),
     distinto(entrada_cuba5,v(X)))),entrada_cuba5,v(X)) :-
 (ST = abierto,X = 100;
  ST = trescuartos, X = 75;
  ST = medio, X = 50;
  ST = cuarto, X = 25;
  ST = cerrado, X = 0).
causa(tuberia7,y(igual(estado_grifo8,v(ST)),
     y(mayor(v(X).presion_cuba5),distinto(entrada_cuba5,presion_cuba5))),
    entrada_cuba5.presion_cuba5) :-
 (ST = abierto,X = 100;
  ST = trescuartos. X = 75;
  ST = medio, X = 50;
  ST = cuarto, X = 25;
  ST = cerrado, X = 0).
causa(tuberia7,distinto(entrada_cuba5,salida_cuba6),
          salida_cuba6,entrada_cuba5).

causa(cuba1,y(distinto(cantidad,v([[vacio,0,1000]])),
          distinto(presion,v(100))),presion,v(100)).
causa(cuba1,y(igual(cantidad,v([[vacio,0,1000]])),
          distinto(presion,entrada_tuberia1)),
         presion,entrada_tuberia1).

causa(cuba2,y(distinto(cantidad,v([[vacio,0,1000]])),
          distinto(presion,v(100))),presion,v(100)).
causa(cuba2,y(igual(cantidad,v([[vacio,0,1000]])),
          distinto(presion,entrada_tuberia1)),
         presion,entrada_tuberia1).

causa(cuba3,y(distinto(cantidad,v([[vacio,0,1000]])),
          distinto(presion,v(100))),presion,v(100)).
causa(cuba3,y(igual(cantidad,v([[vacio,0,1000]])),
          y(mayor(salida_tuberia4+salida_tuberia5,
               entrada_tuberia2+entrada_tuberia3),
```

```
          o(nomayor(salida_tuberia4,presion),
            nomayor(salida_tuberia5,presion)))),
        presion,presion-(salida_tuberia4+salida_tuberia5
            -entrada_tuberia2-entrada_tuberia3)/v(3)).
causa(cuba3,y(igual(cantidad,v([[vacio,0,1000]])),
        y(mayor(entrada_tuberia2+entrada_tuberia3,
            salida_tuberia4+salida_tuberia5),
          o(nomayor(presion,salida_tuberia4),
            nomayor(presion,salida_tuberia5)))),
        presion,presion-(salida_tuberia4+salida_tuberia5
            -entrada_tuberia2-entrada_tuberia3)/v(3)).

causa(cuba4,y(distinto(cantidad,v([[vacio,0,1000]])),
        distinto(presion,v(100))),presion,v(100)).
causa(cuba4,y(igual(cantidad,v([[vacio,0,1000]])),
        distinto(presion,entrada_tuberia4)),
      presion,entrada_tuberia4).

causa(cuba5,y(distinto(cantidad,v([[vacio,0,1000]])),
        distinto(presion,v(100))),presion,v(100)).
causa(cuba5,y(igual(cantidad,v([[vacio,0,1000]])),
        distinto(presion,entrada_tuberia5)),
      presion,entrada_tuberia5).

causa(cuba6,y(distinto(cantidad,v([[vacio,0,1000]])),
        distinto(presion,v(100))),presion,v(100)).
causa(cuba6,y(igual(cantidad,v([[vacio,0,1000]])),
        y(mayor(salida_tuberia1,
            entrada_tuberia6+entrada_tuberia7),
          nomayor(salida_tuberia1,presion))),
        presion,presion-(salida_tuberia1
            -entrada_tuberia6-entrada_tuberia7)/v(3)).
causa(cuba6,y(igual(cantidad,v([[vacio,0,1000]])),
        y(mayor(entrada_tuberia6+entrada_tuberia7,
            salida_tuberia1),
          nomayor(presion,salida_tuberia1))),
        presion,presion-(salida_tuberia1
            -entrada_tuberia6-entrada_tuberia7)/v(3)).
```

```
pendiente(cuba1,cantidad,[fluido,vacio],
        resta(entrada_tuberia1,salida_tuberia2)).

pendiente(cuba2,cantidad,[fluido,vacio],
        resta(entrada_tuberia1,salida_tuberia3)).

pendiente(cuba3,cantidad,[fluido,vacio],
        resta(suma(entrada_tuberia2,entrada_tuberia3),
            suma(salida_tuberia4,salida_tuberia5))).

pendiente(cuba4,cantidad,[fluido,vacio],
        resta(entrada_tuberia4,salida_tuberia6)).

pendiente(cuba5,cantidad,[fluido,vacio],
        resta(entrada_tuberia5,salida_tuberia7)).

pendiente(cuba6,cantidad,[fluido,vacio],
        resta(suma(entrada_tuberia6,entrada_tuberia7),
            salida_tuberia1)).

% RELACIONES

elmismo(A,B) :- mismo(A,B);mismo(B,A).

mismo(var(tuberia1,entrada_cuba6),var(cuba6,salida_tuberia1)).
mismo(var(tuberia1,presion_cuba6),var(cuba6,presion)).
mismo(var(tuberia1,salida_cuba1),var(cuba1,entrada_tuberia1)).
mismo(var(tuberia1,salida_cuba2),var(cuba2,entrada_tuberia1)).
mismo(var(tuberia1,estado_grifo1),var(grifo1,estado)).
mismo(var(tuberia1,estado_grifo2),var(grifo2,estado)).

mismo(var(tuberia2,entrada_cuba1),var(cuba1,salida_tuberia2)).
mismo(var(tuberia2,presion_cuba1),var(cuba1,presion)).
mismo(var(tuberia2,salida_cuba3),var(cuba3,entrada_tuberia2)).
mismo(var(tuberia2,estado_grifo3),var(grifo3,estado)).

mismo(var(tuberia3,entrada_cuba2),var(cuba2,salida_tuberia3)).
```

```
mismo(var(tuberia3,presion_cuba2),var(cuba2,presion)).
mismo(var(tuberia3,salida_cuba3),var(cuba3,entrada_tuberia3)).
mismo(var(tuberia3,estado_grifo4),var(grifo4,estado)).

mismo(var(tuberia4,entrada_cuba3),var(cuba3,salida_tuberia4)).
mismo(var(tuberia4,presion_cuba3),var(cuba3,presion)).
mismo(var(tuberia4,salida_cuba4),var(cuba4,entrada_tuberia4)).
mismo(var(tuberia4,estado_grifo5),var(grifo5,estado)).

mismo(var(tuberia5,entrada_cuba3),var(cuba3,salida_tuberia5)).
mismo(var(tuberia5,presion_cuba3),var(cuba3,presion)).
mismo(var(tuberia5,salida_cuba5),var(cuba5,entrada_tuberia5)).
mismo(var(tuberia5,estado_grifo6),var(grifo6,estado)).

mismo(var(tuberia6,entrada_cuba4),var(cuba4,salida_tuberia6)).
mismo(var(tuberia6,presion_cuba4),var(cuba4,presion)).
mismo(var(tuberia6,salida_cuba6),var(cuba6,entrada_tuberia6)).
mismo(var(tuberia6,estado_grifo7),var(grifo7,estado)).

mismo(var(tuberia7,entrada_cuba5),var(cuba5,salida_tuberia7)).
mismo(var(tuberia7,presion_cuba5),var(cuba5,presion)).
mismo(var(tuberia7,salida_cuba6),var(cuba6,entrada_tuberia7)).
mismo(var(tuberia7,estado_grifo8),var(grifo8,estado)).

% BASE TEMPORAL

% ESTADO INICIAL

event(tuberia1,0,[[entrada_cuba6,100],
                  [presion_cuba6,100],
                  [salida_cuba1,50],
                  [salida_cuba2,50],
                  [estado_grifo1,abierto],
                  [estado_grifo2,abierto]]).

event(tuberia2,0,[[entrada_cuba1,100],
                  [presion_cuba1,100],
                  [salida_cuba3,100],
```

```
        [estado_grifo3,abierto]]).

event(tuberia3,0,[[entrada_cuba2,100],
        [presion_cuba2,100],
        [salida_cuba3,100],
        [estado_grifo4,abierto]]).

event(tuberia4,0,[[entrada_cuba3,100],
        [presion_cuba3,100],
        [salida_cuba4,100],
        [estado_grifo5,abierto]]).

event(tuberia5,0,[[entrada_cuba3,100],
        [presion_cuba3,100],
        [salida_cuba5,100],
        [estado_grifo6,abierto]]).

event(tuberia6,0,[[entrada_cuba4,100],
        [presion_cuba4,100],
        [salida_cuba6,100],
        [estado_grifo7,abierto]]).

event(tuberia7,0,[[entrada_cuba5,100],
        [presion_cuba5,100],
        [salida_cuba6,100],
        [estado_grifo8,abierto]]).

event(grifo1,0,[[estado,abierto]]).

event(grifo2,0,[[estado,abierto]]).

event(grifo3,0,[[estado,abierto]]).

event(grifo4,0,[[estado,abierto]]).

event(grifo5,0,[[estado,abierto]]).

event(grifo6,0,[[estado,abierto]]).
```

```
event(grifo7,0,[[estado,abierto]]).

event(grifo8,0,[[estado,abierto]]).

event(cuba1,0,[[entrada_tuberia1,50],
        [salida_tuberia2,100],
        [cantidad,[[fluido,0,500],[vacio,500,1000]]],
        [presion,100]]).

event(cuba2,0,[[entrada_tuberia1,50],
        [salida_tuberia3,100],
        [cantidad,[[fluido,0,500],[vacio,500,1000]]],
        [presion,100]]).

event(cuba3,0,[[entrada_tuberia2,100],
        [entrada_tuberia3,100],
        [salida_tuberia4,100],
        [salida_tuberia5,100],
        [cantidad,[[fluido,0,500],[vacio,500,1000]]],
        [presion,100]]).

event(cuba4,0,[[entrada_tuberia4,100],
        [salida_tuberia6,100],
        [cantidad,[[fluido,0,500],[vacio,500,1000]]],
        [presion,100]]).

event(cuba5,0,[[entrada_tuberia5,100],
        [salida_tuberia7,100],
        [cantidad,[[fluido,0,500],[vacio,500,1000]]],
        [presion,100]]).

event(cuba6,0,[[entrada_tuberia6,100],
        [entrada_tuberia7,100],
        [salida_tuberia1,100],
        [cantidad,[[fluido,0,500],[vacio,500,10000]]],
        [presion,100]]).
```

```
% EVOLUCION POSTERIOR

event(grifo3,5,[[estado,trescuartos]]).

event(grifo3,10,[[estado,medio]]).

event(grifo3,15,[[estado,cuarto]]).

event(grifo3,20,[[estado,cerrado]]).

event(grifo3,40,[[estado,abierto]]).
```

Anexo C

LAS CARACTERISTICAS BASICAS DEL LDOOR

C.1 Objetos, Atributos y Restricciones

En LDOOR, el conocimiento se expresa como restricciones sobre los dominios de las variables. Una restricción es una descripción del dominio. El dominio de una variable es el conjunto de descripciones del objeto que puede denotar. Un objeto es una cosa en el mundo; puede ser cualquier elemento de información relevante al problema considerado. Una instancia es el nombre de un objeto. Una misma instancia no puede denotar más que a un solo objeto. Cada objeto tiene un conjunto de atributos, y los atributos son asimismo objetos.

Una pregunta es un conjunto de restricciones sobre un conjunto de variables. Cada restricción describe un dominio, y el conjunto de restricciones para una variable dada denota un conjunto de dominios para esa variable. El dominio resultante de la variable viene dado por la intersección de todos los dominios en este conjunto. Dada una pregunta, LDOOR devuelve un conjunto simplificado de restricciones describiendo el dominio resultante de cada variable en la pregunta. Si no actúa ninguna restricción sobre una variable, su dominio es el dominio universal.

Conviene señalar que el LDOOR no manipula los dominios como tales, sino las descripciones de los mismos en forma de restricciones. Puesto que las restricciones no tienen que ser completamente reducidas al nivel de los dominios, es posible razonar en un alto nivel de abstracción. El LDOOR puede simplificar un conjunto de restricciones sobre una variable, incluso si las posibles instancias de esa variable no pueden ser determinadas.

Un tipo representa un dominio predefinido. La base de conocimiento es una colección de tipos y definiciones de instancia. Cada definición de tipo define un dominio a través de un número de restricciones. Las restricciones de tipo están simplificadas a los más generales subtipos comunes encontrados en la base de conocimiento: la simplificación de tipos no implica la búsqueda del conjunto de instancias para las que el tipo es válido.

Una restricción sobre una variable X tiene la siguiente forma:

- X debe o no debe ser un miembro de una union de dominios. Cada dominio D se describe como sigue:

    - D es un conjunto atómico de una instancia
    - D es una clase
    - D es el dominio resultante de otra variable
    - Uno o más atributos del dominio D se restringen entre sí.

Consideremos un ejemplo de geometría escolar. Los tres tipos de objetos que nos interesan son: Rombos, Cuadrados y Rectángulos. Hay dos atributos relevantes a estos objetos: Su número de ángulos rectos y su número de lados iguales. Pueden haber cualquier número de instancias de estos tipos: rombo12, cuadrado14, etc.

La definición de tipo en la base de conocimiento para la restricción de "rombo" consiste en que el atributo "número de lados iguales" sea "cuatro". La definición para la restricción de "rectangulo" consite en que el "número de ángulos rectos" sea "cuatro". La definición para las restricciones de "cuadrado" son ser del tipo "rectángulo" y del tipo "rombo".

Supóngase que presentamos la siguiente pregunta: X es tanto un rectángulo como un rombo. Estas dos restricciones se simplificarán a la restricción siguiente: X es un cuadrado.

Seguidamente mostraremos la representación de esta base de conocimiento y esta pregunta en LDOOR. Los tipos se escriben con dos puntos (por ejemplo, :cuadrado), a diferencia de las instancias (por ejemplo, cuadrado43). Cuando no se escribe ningún tipo detrás de un signo de dos puntos, se entiende implícitamente ":universo" (El dominio que contiene todas las posibles descripciones de objeto), por ello :rombo

es subtipo de :universo (primera línea) y :cuadrado es subtipo de :rombo (tercera línea).

```
:rombo:('número de lados iguales' => 4).
:rectangulo:('número de ángulos-rectos' => 4).
:cuadrado:rombo.
:cuadrado:rectangulo.
rombo12:rombo.
cuadrado43:cuadrado.
etc...

?- X:rectangulo, X:rombo.

X:cuadrado.
```

Los atributos son simplemente valuados. Los valores multi atributo se representan como estructuras dinámicas (listas, por ejemplo). En el caso de que un atributo semejante no tenga valor, es una estructura vacía (una lista vacía por ejemplo). Notar que una estructura vacía no representa un dominio vacío.

Si el dominio de la variable es no vacío , entonces los dominios de cada uno de sus atributos deben ser también no vacíos. Esto se sigue de la definición de dominios (dada más adelante con la semántica formal).

Si un atributo no es relevante a un objeto o no tiene sentido para el mismo, en vez de decir que su dominio es vacío, en LDOOR:

- Si el atributo no es relevante al problema considerado, su dominio es el universo.
- Si el atributo no tiene sentido para el objeto, se le califica con el tipo :inconsistente en la base de conocimiento, o cualquier otro tipo que no tenga un subtipo común con otro tipo. De esta forma aseguramos que el conjunto de posibles instancias para este atributo es vacío.

C.2 Tipos y Taxonomías

Un tipo representa un dominio predefinido. Una restricción de tipo aplicada a un objeto especifica que el objeto pertenece al dominio denotado por el tipo. Los tipos se utilizan para subsumir todas las restricciones contenidas en su definición y por

tanto permiten una cierta economía de expresión. Por ejemplo, los tipos :rectangulo y :rombo subsumen todas las restricciones pertenecientes a cuadriláteros (cuatro lados, cuatro ángulos,...), a trapecios (dos lados, paralelos,...) y a paralelogramos (lados opuestos paralelos). Como se explicó anteriormente, el tipo :rectangulo especifica además de lo dicho, que los cuatro ángulos son rectos, mientras que :rombo especifica además que los cuatro lados son iguales.

Los tipos permiten definiciones recursivas que no podrían ser expresadas únicamente como conjuntos de instancias y variables. Por ejemplo, la definición de :persona especifica que el apellido del padre de una persona es el mismo que su propio apellido y que el padre es también una persona ([Ait-Kaci,1986]). Esta restricción sobre el apellido de una persona se aplica además a sus ancestros de forma recursiva.

```
:persona:(      apellido => S,
          padre => :persona(apellido => S)).
```

Los tipos también permiten una cierta economía de ejecución (un aumento en el rendimiento), puesto que la simplificación del conjunto de restricciones que representan puede ser realizada en una forma predeterminada y potencialemnte más eficiente. Además, como en el ejemplo de la sección anterior, uno puede definir (pre-definir) que si un objeto es :rectangulo y a la vez es :rombo, entonces la simplificación de estos tipos da :cuadrado directamente, sin tener que procesar todas las restricciones aritméticas detalladas que esos tipos subsumen. Gracias a los tipos, la simplificación de restricciones se puede realizar a un nivel de abstracción más alto, y por consiguiente más rápidamente. Además el resultado obtenido es también más informativo.

Los tipos se definen desde los más generales a los más específicos. El tipo más general es :universo. Por motivos de economía de expresión, cada tipo se define en términos de los tipos más específicos que lo incluyan. Por ejemplo, :trapecio sería definido en términos de :cuadrilatero, :paralelogramo en términos de :trapecio, :rectangulo y :rombo en términos de :paralelogramo, y :cuadrado en términos de rectángulo y :rombo. A cada definición se le añaden una o más restricciones suplementarias, sobre uno o más atributos.

El conjunto de definiciones de tipo forma una taxonomía. Esto constituye una estructura jerárquica - más precisamente, un grafo. Los atributos de cada tipo heredan las restricciones que se especifican para la misma etiqueta sobre los tipos

que son más generales. Estos últimos son sus supertipos. Por definición, el dominio de un supertipo de un tipo T incluye el dominio de T.

En otras palabras, una taxonomía consiste en conjuntos de restricciones, cada uno definiendo un tipo (totalmente o parcialmente), que están organizados en un grafo permitiendo herencia múltiple de restricciones para cada etiqueta. Cada tipo T denota el dominio caracterizado por los conjutnos de restricciones que definen T y sus supertipos en la jerarquía.

En LDOOR puro, todo el conocimientode los dominios de aplicación es expresado en la taxonomía.- Esto incluye la definición de instancias y puede verse a la vez como una base de conocimiento, como una base de datos (una instancia es un dato concreto, un tipo es un dato abstracto), y como un programa (unica y totalmente declarativo). Por ello, en LDOOR, la taxonomía es la base de conocimiento.

C.3 Semántica formal

Consideremos un programa y una pregunta que mencionan m atributos a_1 a a_m. En LDOOR, las preguntas pueden ser sobre varias variables, en cuyos dominios resultantes estamos interesados. Llamemos "variable principal" a una variable que denota un objeto que no es un atributo en la pregunta. En esta sección consideraremos preguntas sobre una única variable principal. Las preguntas sobre varias variables principales pueden considerarse como compuestas de varias preguntas sobre una variable principal.

Una pregunta es un conjutno de descripciones del dominio (restricciones) sobre variables. Un dominio es un conjunto de descripciones de objeto. Una descripción de objeto es una m+1 tupla $(i,v_1...v_m)$, donde i es una instancia y v_1a v_m son descripciones de objeto correspondientes a los atributos a_1 a a_m.

Una descripción del dominio para una variable dada es o un par $(\in,S)$ o un par $(\notin,S)$, donde S es un conjunto de pares (I,A), donde I es una instancia o un tipo o una variable, y A es una m-tupla de descripciones del dominio, una por cada atributo.

Sea $C_X(Q)$ el conjunto de restricciones (descripciones del dominio) sobre la variable x en la pregunta Q. Entonces, este conjunto de restricciones denotan el dominio $M1(C_X(Q))$, donde M1 (función de significado) se define de la siguiente forma:

$$M1(C) = \text{universo} \cap \bigcap_{c \in C} M2(c)$$

$$M2((\in, S)) = \bigcup_{s \in S} M3(s)$$

$$M2((\notin, S)) = \text{universo} - \bigcup_{s \in S} M3(s)$$

$$M3((I,A)) = M4(I) \cap M5(A)$$

$M4(I) = M1(L(I))$ si I es una instancia o un tipo
$M4(I) = M1(C_I(Q))$ si I es una variable

$$M5((a_1 \ldots a_m)) = \text{universo} \times \underset{i=1..m}{X} M1(C_{a_i}(Q))$$

La función L(I) proyecta una instancia o tipo sobre el conjunto de descripciones del dominio que la definen en la base de conocimiento.

La función M1 representa el significado denotador de un conjutno de restricciones. El LDOOR, sin embargo, no genera una representación extensional de un dominio, sino que transforma conjuntos de restricciones en conjuntos más simples de restricciones con la misma denotación, siguiendo un método que llamamos simplificación cuya explicación detallada cae fuera del ámbito de este capítulo.

C.3.1 Ejemplo con instancias y variables

Para ilustrar la semántica formal definimos un universo de 6 objetos: rectángulo, cuadrado, rombo, 0, 2 y 4. Hay dos atributos: número de lados iguales y número de ángulos rectos, respectivamente a_1 y a_2. Además, sabemos que un rectángulo tiene cuatro ángulos rectos y dos o cuatro lados iguales, que un cuadrado tiene cuatro ángulos rectos y cuatro lados iguales, y que un rombo tiene cuatro lados iguales y cero o cuatro ángulos rectos. Por ello hay ocho posibles formas de descripciones de objeto (3-tupla), como se muestra en la siguiente tabla que representa la semántica del contenido de la base de conocimiento.

instancia	lados	angulos
(rectangulo,	(2,_,_),	(4,_,_))
(rectangulo,	(4,_,_),	(4,_,_))
(cuadrado,	(4,_,_)	(4,_,_))

(rombo,	(4,_,_)	(0,_,_))
(rombo,	(4,_,_)	(4,_,_))
(0,	_,	_)
(2,	_,	_)
(4,	_,	_)

Es importante señalar que esta representación no es la del LDOOR, sino que se introduce tan sólo para especificar e ilustrar la semántica del LDOOR. Formalmente hablando, todos los objetos tienen todos los atributos, y todo atributo es un objeto. Hay por consiguiente muchos atributos irrelevantes para cualquier objeto, y estos son representados por el carácter '_'. Nótese que todas las descripciones de objeto tienen el mismo número de atributos, 2 en este caso. Además, toda descripción de objeto es una 3-tupla (con 1 elemento para la instancia y 2 elementos para las descripciones de atributos).

Ahora consideremos la siguiente pregunta Q, que trata de averiguar si X es o bien un rectángulo o un rombo y tiene cero o cuatro lados iguales e Y no tiene cuatro ángulos rectos y X es o bien Y o bien un cuadrado.

CX(Q) = {(∈,{ (rectangulo,({0,4},universo)),
(rombo,({0,4},universo))}),
(∈,{Y,cuadrado})}
CY(Q) = {(∉,{(universo,(universo,{4}))})}

En sintaxis LDOOR Q se escribiría:

?- X:{rectangulo;rombo}(lados => {0;4}),
Y\:(angulos => 4),
X:{Y;cuadrado}.

Claramente, el dominio resultante para Y debe ser la descripción del rombo sin 4 ángulos rectos, puesto que Y no puede ser un rectángulo o un cuadrado, y X tendrá el mismo dominio resultante que Y, ya que X no puede ser un cuadrado. Veamos cómo nuestra función de significado M1 da este resultado.

El dominio para X dado por la primera restricción en $C_X(Q)$ es

M2((∈,{ (rectangulo,({0,4},universo)),

(rombo,({0,4},universo))}))

que es

M3((rectangulo,({0,4},universo))) ∪
M3((rombo,({0,4},universo)))

que, utilizando M4 (que consulta la base de conocimiento) y M5, es

({(rectangulo,(2,_,_),(4,_,_)),(rectangulo,(4,_,_),(4,_,_))} ∩ universo x {0,4} x universo)
∪
({(rombo,(4,_,_),(0,_,_)),(rombo,(4,_,_),(4,_,_))} ∩
universo x {0,4} x universo)

que es

(rectangulo,(4,_,_),(4,_,_))
(rombo,(4,_,_),(0,_,_))
(rombo,(4,_,_),(4,_,_))

De forma similar, el dominio para Y dado por $C_Y(Q)$ es

(rombo,(4,_,_),(0,_,_))

que es unido al dominio del cuadrado para dar el dominio para X de la segunda restricción de $C_X(Q)$:

(cuadrado,(4,_,_),(4,_,_))
(rombo,(4,_,_),(0,_,_))

Utilizando M1, la intersección de los dos dominios para X da el dominio resultante

(rombo,(4,_,_),(0,_,_))

Un ejemplo simple con tipos

Consideremos los tipos :rectangulo y :rombo. La base de conocimiento especifica que las descripciones de objeto correspondientes a :rectangulo son

instancia	lados	ángulos
(rectangulo,	(2,_,_),	(4,_,_))
(rectangulo,	(4,_,_)	(4,_,_))
(cuadrado,	(4,_,_)	(4,_,_))

Y los correspondientes al tipo :rombo son:

instancia	lados	ángulos
(rombo,	(4,_,_),	(0,_,_))
(rombo,	(4,_,_)	(4,_,_))
(cuadrado,	(4,_,_)	(4,_,_))

Supongamos ahora que hacemos la pregunta:

CX(Q) = {(∈,{:rectangulo}),
(∈,{:rombo})}

que en LDOOR es

?- X:rectangulo, X:rombo.

que debería devolver X = cuadrado, puesto que éste es el único objeto en los dominios denotado por los tipos dados en la pregunta.

La descripción del dominio para X dada por la primera restricción en $C_X(Q)$ es

M2((∈,{(:rectangulo,(universo,universo))}))

que es

M3(:rectangulo,(universo,universo))

que, usando M4 y M5, es

{ (rectangulo, (2,_,_), (4,_,_)),

(rectangulo, (4,_,_), (4,_,_)),
(cuadrado, (4,_,_), (4,_,_))}

De forma similar la segunda restricción representa:

{ (rombo, (4,_,_), (0,_,_)),
(rombo, (4,_,_), (4,_,_)),
(cuadrado, (4,_,_), (4,_,_))}

La intersección por M1 de estos dos dominios da entonces la única descripción de objeto, correspondiente a la instancia "cuadrado", tal como era de esperar.

C.4 Sintaxis del LDOOR

Esta sección introduce informalmente la sintaxis del LDOOR. Puesto que tratamos de integrar LDOOR en un entorno Prolog, la sintaxis del LDOOR será tan similar como sea posible a la sintaxis del Prolog de Edimburgo, que actualmente representa un standard.

Al igual que en el Prolog de Edimburgo, una variable comienza con una letra mayúscula o con un signo '_'. Los tipos y las instancias no pueden comenzar con una letra mayúscula a menos que estén entre comillas, por ejemplo:

juan, coche_123 y 'lápiz de Juan' (cualquier átomo de Prolog) son *instancias*.

COCHE, Jaime, y _123 (cualquier variable de Prolog) son *variables*.

Nótese que las variables anónimas pueden ser omitidas. Los tipos se distinguen de las instancias por tener un signo de dos puntos a su izquierda. Por ejemplo:

:coche, :barco_volante, :'barco-volante' y :'Barco volante' son tipos.

El tipo que denota el dominio universal puede ser omitido.

La notación de los *atributos* es muy parecida a la de LOGIN. Unicamente se escriben los atributos que se encuentran realmente restringidos, el resto puede omitirse. Los atributos se escriben en cualquier orden y separados por comas. Su etiqueta indica qué característica del objeto denotan. Los atributos pueden encontrarse embutidos.

En la siguiente restricción, Y es el atributo etiqueta_1 de X, e Y es del tipo :tipo_de_Y y tiene el atributo Z para etiqueta_2:

```
X:tipo_de_X( etiqueta_1 =>
                    Y:tipo_de_Y(etiqueta_2 => Z:tipo_de_Z)).
```

Las mismas variables pueden aparecer en varios lugares, por ejemplo:

```
:persona:(    apellido => S,
                    padre => :persona(apellido => S)).
```

Nótese que en el ejemplo anterior el tipo de S es implícitamente :universe aunque no se haya mencionado.

Consideremos ahora la forma general de la sintaxis de restricciones, que incluye la sintaxis dada anteriormente. Tal y como se indicó en la sección anterior, una restricción sobre una variable X tiene la siguiente forma:

- X debe o no debe ser un miembro de una union de dominios. Cada dominio D se describe como sigue:

    - D es un conjunto atómico de una instancia
    - D es una clase
    - D es el dominio resultante de otra variable
    - Uno o más atributos del dominio D se restringen entre sí.

La unión de dominios se representa con llaves y punto y coma. "X es *miembro* de la union de dominios desde $dominio_1$ hasta $dominio_n$" se representa de la siguiente forma:

$X:\{dominio_1;...;dominio_n\}$

"X *no es miembro* de la union de dominios desde $dominio_1$ hasta $dominio_n$" se representa de la siguiente forma:

$X\backslash:\{dominio_1;...;dominio_n\}$

Para representar por ejemplo un dominio que sea la unión de un conjunto de una instancia simple coche_123, un tipo :coche_veloz y el dominio de una variable COCHE que está restringida a tener su atributo peso de tipo :pesado, escribimos:

{coche_123;:coche_veloz;COCHE:(peso => :pesado)}

peso => :pesado es la notación abreviada para peso => _:{:pesado}. Como regla general, las llaves pueden omitirse para uniones de un único dominio.

También es posible expresar que algún dominio de una variable no es un subconjunto de algún otro dominio. Por ejemplo, :X\:t significa que el dominio de X no está incluído en el dominio representado por el tipo :t. Esta no es una restricción primitiva, puesto que puede ser expresada mediante la relación de no pertenecia.

Una pregunta es un conjunto de restricciones escritas detrás de un ?-, separadas por comas, y terminadas en punto. Por ejemplo:

```
?-    X:{rectangulo123;triangulo456}(area => A),
      X\:(area => 16).
```

La sintaxis para una definición de tipo, comparada a la sintaxis para una restricción sobre una variable, tiene dos elementos adicionales. El tipo es seguido directamente por un supertipo, y los atributos (opcionales). Es posible añadir al final, precedido por una barra inclinada, un conjunto de restricciones adicionales, con idéntica sintaxis a las preguntas; por ejemplo:

X:comienzo123:suceso(agente => juan)/:(inicia => X, tiempo => 12).

La línea anterior define el tipo :comienzo123. Es subtipo de :suceso, su atributo agente se instancia a juan, y nalgún objeto (de tipo :universo) lo inicia en el tiempo 12.

Pueden darse varias definiciones de un mismo tipo. LDOOR las combinará automáticamente, simplificando todas las restricciones sobre las mismas etiquetas.

En algunos casos nos gustaría especificar algunas restricciones que deben mantenerse para *todos los objetos,*, entonces debemos escribirlas en una definición de :universo.

Esta definición se escribe detrás de un doble dos puntos "::(...)/..."; que es una forma abreviada de escribir ":universo:universo(...)/...". Por ejemplo:

::(naturaleza => concreta).

Esta línea indicaría que todos los objetos tendrían naturaleza concreta.

Anexo D

LISTADO PROLOG DEL MOTOR DE INFERENCIA DEL (λ,T) INDEPENDIENTE DEL DOMINIO

```
% PROGRAMA

main :- repeat,
      anular_testigo,
      analizar_objetos(1000),
      cerrar_bucle,!.

anular_testigo :- abolish(testigo/0).

analizar_objetos(Tf) :-
       i(O,C),subclase(C,dominio_interior),
       analiza_objeto(O,C,Tf),fail.
analizar_objetos(_).

cerrar_bucle :- not testigo.

% ANALIZAR OBJETO

analiza_objeto(O,C,Tf) :-
        ultimoevent(O,Ti,Estado),
        (\+hevent(O,_),
          detectar_inconsistencia(O,C,Ti,Tf,Estado,T),
          colocahev(O,T),colocahac(O,T),
```

```
        assert(testigo);
        hevent(O,T),
         calculaestado(O,C,Ti,Tf,Estado,T,NEst),
         assert(testigo),
         (\+ hevent(O,T,_),assert(hevent(O,T,NEst)),
               colocahac(O,T);
          hevent(O,T,NE),
          (NEst = NE,quitahac(O,T),
                  afirmar(O,T,NEst);
           NEst \= NE,retract(hevent(O,T,NE)),
                   assert(hevent(O,T,NEst)),
                   colocahac(O,T)))),!.

colocahev(O,T) :- hevent(_,T1),T1 < T,!.
colocahev(O,T) :- assert(hevent(O,T)),fail.
colocahev(O,T) :- hevent(O2,T2),T2 > T,retract(hevent(O2,T2)),fail.
colocahev(_,_).

colocahac(O,T) :- hacc(_,T1),T1 < T,!.
colocahac(O,T) :- assert(hacc(O,T)),fail.
colocahac(O,T) :- hacc(O2,T2),T2 > T,retract(hacc(O2,T2)),fail.
colocahac(_,_).

quitahac(O,T) :- retract(hacc(O,T)),fail.
quitahac(O,T) :- hacc(O2,T2),T2 > T,retract(hacc(O2,T2)),fail.
quitahac(_,_).

afirmar(O,T,NEst) :- \+ hacc(_,_),!,retract(hevent(O,T)),
 retract(hevent(O,T,_)),assert(event(O,T,NEst)),
 nl,write($Afirmando nuevo estado$),nl,write(event(O,T,NEst)),nl.
afirmar(_,_,_).

% ANALIZA INCONSISTENCIA

detectar_inconsistencia(O,C,Ti,Tf,E,T) :-
 findall(T,d_inconsistencia(O,C,Ti,Tf,E,T),L),minimo(T,L).

d_inconsistencia(O,C,Ti,Tf,E,T) :-
```

```
 di_interior(O,C,Ti,Tf,E,T);
 di_exterior(O,C,Ti,Tf,E,T).

di_interior(O,C,Ti,Tf,E,T) :-
 findall(Tj,di_inte(O,C,Ti,Tf,E,Tj),L),minimo(T,L).

di_inte(O,C,Ti,Tf,E,T) :-
 v(O,Var,flt),localize(E,Var,[_,Valor]),
 flt_vera([O,Var],Valor,Di),Di < Tf - Ti,T is Ti + Di.

di_exterior(O,C,Ti,Tf,E,T) :-
 findall(Tj,di_exte(O,C,Ti,Tf,E,Tj),L),minimo(T,L).

di_exte(O,C,Ti,Tf,E,T) :-
 elmismo(var(O,V),var(O2,V2)),
 (event(O2,T,E2);hevent(O2,T,E2)),T > Ti,T < Tf,
 localize(E2,V2,[_,Val2]),
 localize(E,V,[_,Val]),
 (number(Val),number(Val2),Val =\= Val2;
  Val \= Val2).

% CALCULA ESTADO EN TIEMPO T

calculaestado(O,C,Ti,Tf,Estado,T,N3E) :-
  D is T - Ti,
  evolucion_interna(O,Estado,D,NEstado),
  consistencia_externa(O,NEstado,T,N2Estado),
  consistencia_interna(O,N2Estado,N3E).

evolucion_interna(O,[[V,Vl]|E],D,[[V,V2l]|NE]) :-
  (v(O,V,ft),V2l = Vl;
   v(O,V,flt),nst([O,V],Vl,V2l,D)),!,
  evolucion_interna(O,E,D,NE).
evolucion_interna(_,[],_,[]).

consistencia_externa(O,[[V,Vl]|E],T,[[V,V2l]|E2]) :-
  (elmismo(var(O,V),var(O2,V2)),
     (event(O2,T2,ES);hevent(O2,T2,ES)),T2 =:= T,
```

```
      localize(ES,V2,[_,V2l]);
    V2l = Vl),
   !,consistencia_externa(O,E,T,E2).
consistencia_externa(O,[],T,[]).

consistencia_interna(O,E1,E2) :-
   abolish(con_ins/1),abolish(con_cyc/0),assert(con_ins(E1)),
   cons_inte(O),con_ins(E2).

cons_inte(O) :- repeat,not cons_inte2(O),!.

cons_inte2(O) :-
   causa(O,Cond,Var,Valor),con_ins(E),
   [!ci_condicion(E,Cond),
     ci_calcula(E,Valor,V2),
     localize(E,Var,[_,V1]),
     (number(V1),number(V2),V1 =\= V2;
      V1 \= V2),
     sustituye(E,Var,[Var,V2],Ep),
     abolish(con_ins/1),abolish(con_cyc/0),
     assert(con_ins(Ep)),assert(con_cyc)!],fail.
cons_inte2(O) :- con_cyc,abolish(con_cyc/0),!.

ci_condicion(E,y(C1,C2)) :- !,ci_condicion(E,C1),ci_condicion(E,C2).
ci_condicion(E,o(C1,C2)) :- !,(ci_condicion(E,C1);ci_condicion(E,C2)).
ci_condicion(E,igual(V1,V2)) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),
                        (number(A),number(B),A =:= B;
                         A = B).
ci_condicion(E,mayor(V1,V2)) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),
                        number(A),number(B),A - B > 1e-8.
ci_condicion(E,nomayor(V1,V2)) :- !,not ci_condicion(E,mayor(V1,V2)).
ci_condicion(E,distinto(V1,V2)) :- \+ ci_condicion(E,igual(V1,V2)),!.

ci_valor(_,v(A),A) :- !.
ci_valor(E,V,A) :- localize(E,V,[_,A]).

ci_calcula(E,V1*V2,C) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),C is A*B.
ci_calcula(E,V1/V2,C) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),C is A/B.
```

```
ci_calcula(E,V1+V2,C) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),C is A+B.
ci_calcula(E,V1-V2,C) :- !,ci_calcula(E,V1,A),ci_calcula(E,V2,B),C is A-B.
ci_calcula(E,V,A) :- ci_valor(E,V,A).

% ATRIBUTOS PENDIENTE

a([O,Var],borderspeed,[S1,S2,P]) :-
  pendiente(O,Var,[S1,S2],Calcular),
  uevent(O,_,E),resolver(E,Calcular,P).

resolver(E,resta(A,B),C) :- !,resolver(E,A,C1),resolver(E,B,C2),C is C1 - C2.
resolver(E,suma(A,B),C) :- !,resolver(E,A,C1),resolver(E,B,C2),C is C1 + C2.
resolver(E,X,C) :- !,localize(E,X,[_,C]).

% ULTIMOEVENT

ultimoevent(O,T,St) :- retract(uevent(O,_,_)),fail.
ultimoevent(O,T,St) :- ultevent(O,T,St),assert(uevent(O,T,St)).

ultevent(O,T,St) :- findall(Ti,event(O,Ti,_),L),maximo(T,L),event(O,T,St).

% MISCELANEA

mayor(inf,_).
mayor(X,Y) :- number(X),number(Y),X > Y.

maximo(X,[X]) :- !.
maximo(X,[Y|L]) :- maximo(Z,L),(mayor(Y,Z),X = Y;X = Z),!.

minimo(X,[X]) :- !.
minimo(X,[Y|L]) :- minimo(Z,L),(mayor(Y,Z),X = Z;X = Y),!.

localize([[X|Y]|Z],X,[X|Y]) :- !.
localize([_|Y],X,Z) :- !,localize(Y,X,Z).

sustituye([[X|Y]|Z],X,Y2,[Y2|Z]) :- !.
sustituye([X|Y],A,B,[X|Y2]) :- !,sustituye(Y,A,B,Y2).
```

```
% PARAMETRO FLT

% LAMBDAT PREDICATES

generar(LTF,C1,F,Ti) :-
   expandirall(LTF,C1,C),flt_vera(LTF,C,T),
   nst(LTF,C,F,T),entero(Ti,T),!.

entero(Ti,T) :- not(T = inf),Ti is integer(round(T,0)).
entero(inf,inf).

expandirall(LTF,[A],[A]).
expandirall(LTF,[[A,B,C],[D,E,F]|G],[[A,B,C],[D,E,F]|G1]) :-
  not a(LTF,expandir,[A,D,_]),
     expandirall(LTF,[[D,E,F]|G],[[D,E,F]|G1]).
expandirall(LTF,[[A,B,C],[D,E,F]|G],[[A,B,C],[ND,C,C],[D,E,F]|G1]) :-
  a(LTF,expandir,[A,D,ND]),
     expandirall(LTF,[[D,E,F]|G],[[D,E,F]|G1]).

% FLT: Averiguar el tiempo que tardará el estado C en cambiar.

flt_vera(LTF,C,Min) :- ver(LTF,C,nada,L),minimo(Min,L).

ver(LTF,[A,B|C],Pre,[Dur|Lis]) :- durac(LTF,Pre,A,B,Dur),ver(LTF,[B|C],A,Lis).
ver(LTF,[A],Pre,[Dur]) :- durac(LTF,Pre,A,nada,Dur).

durac(LTF,nada,[Bst,Bmin,Bmax],[Cst,Cmin,Cmax],Dur) :-
   a(LTF,borderspeed,[Bst,Cst,V]),
   calcula(V,Bmax - Bmin,Dur).
durac(LTF,[Ast,Amin,Amax],[Bst,Bmin,Bmax],[Cst,Cmin,Cmax],Dur) :-
   a(LTF,borderspeed,[Ast,Bst,V1]),
   a(LTF,borderspeed,[Bst,Cst,V2]),
   V is V2 - V1,
   calcula(V,Bmax - Bmin,Dur).
durac(LTF,[Ast,Amin,Amax],[Bst,Bmin,Bmax],nada,Dur) :-
   a(LTF,borderspeed,[Ast,Bst,V]),
   calcula(- V,Bmax - Bmin,Dur).
durac(LTF,nada,B,nada,inf).
```

```
calcula(V,D,Dur) :- V >= 0,Dur = inf.
calcula(V,D,Dur) :- V < 0,Dur is - (D / V).

nst(LTF,C,F,T) :- ns(LTF,C,nada,F,T),(F =
[[fluido,0,0]|_],write(LTF),nl,write(C),nl,write(T),nl,read(JI);true),!.
ns(LTF,[A,B|C],Pre,[Ai|F],T) :- deduce(LTF,Pre,A,B,T),newst(LTF,Pre,A,B,T,Ai),
                ns(LTF,[B|C],A,F,T).
ns(LTF,[A,B|C],Pre,F,T) :- not deduce(LTF,Pre,A,B,T),ns(LTF,[B|C],A,F,T).
ns(LTF,[A],Pre,[Ai],T) :- deduce(LTF,Pre,A,nada,T),newst(LTF,Pre,A,nada,T,Ai).
ns(LTF,[A],Pre,[],T) :- not deduce(LTF,Pre,A,nada,T).


deduce(LTF,A,B,C,T) :- durac(LTF,A,B,C,D),not iguales(D,T).
deduce(LTF,A,B,C,inf).

iguales(D,T) :- number(D),number(T),!,D - T < 1e-9,T - D < 1e-9.
iguales(D,T) :- D = T.

newst(LTF,nada,[Bst,Bmin,Bmax],[Cst,Cmin,Cmax],T,[Bst,Bmin,Dmax]) :-
   a(LTF,borderspeed,[Bst,Cst,V]),
   Dmax is integer(round(Bmax + (V * T),0)).
newst(LTF,[Ast,Amin,Amax],[Bst,Bmin,Bmax],[Cst,Cmin,Cmax],T,[Bst,Dmin,Dm
ax]) :-
   a(LTF,borderspeed,[Ast,Bst,V1]),
   a(LTF,borderspeed,[Bst,Cst,V2]),
   Dmin is integer(round(Bmin + (V1 * T),0)),
   Dmax is integer(round(Bmax + (V2 * T),0)).
newst(LTF,[Ast,Amin,Amax],[Bst,Bmin,Bmax],nada,T,[Bst,Dmin,Bmax]) :-
   a(LTF,borderspeed,[Ast,Bst,V]),
   Dmin is integer(round(Bmin + (V * T),0)).
newst(LTF,nada,B,nada,_,B).

% SUBCLASE

subclase(C1,C2) :- c(C1,L),miembro_de(C2,L),!.
subclase(C1,C2) :- c(C1,L),miembro_de(X,L),subclase(X,C2).
```

```
miembro_de(X,[X|_]) :- !.
miembro_de(X,[_|L]) :- !,miembro_de(X,L).
```

Anexo E

EL CALCULO DE ACONTECIMIENTOS

Los axiomas originales del Cálculo de Acontecimientos de Kowalski y Sergot son en total 8, pero posteriormente se completaron hasta un total de 21. En este anexo mostraremos el conjunto completo de axiomas:

E.1 Los Axiomas Originales del Cálculo de Acontecimientos

(1)

```
Mcierto_para(p,<comienzo,fin>) :-
        sucede(e),tiempo(e,comienzo),inicia(e,p),
        sucede(e'),tiempo(e',fin),termina(e',p),fin > comienzo,
        no interrumpido_durante(p,<comienzo,fin>).
```

(2)

```
cierto_para_siempre_desde(p,t) :-
        sucede(e),tiempo(e,t),
        inicia(e,p),no interrumpido_despues(p,t).
```

(3)

```
Icierto_para(p,<comienzo,fin>) :-
        Mcierto_para(p,<a,b>),comienzo >= a,fin <= b.
```

(4)

```
Icierto_para(p,<comienzo,fin>) :-
        cierto_para_siempre_desde(p,t),comienzo >= t,comienzo < fin.
```

(5)

```
cierto_en(p,t) :-
        cierto_para_siempre_desde(p,t'),t' < t.
```

(6)

```
cierto_en(p,t) :-
        Mcierto_para(p,<comienzo,fin>),t > comienzo,t < fin.
```

(7)

```
interrumpido_durante(p,<comienzo,fin>) :-
        sucede(e),tiempo(e,t),comienzo < t,
        fin > t,termina(e,p).
```

(8)

```
interrumpido_despues(p,t) :-
        sucede(e),tiempo(e,t'),
        t < t',termina(e,p).
```

E.2 Los Axiomas Extendidos del Cálculo de Acontecimientos

(9)

```
Mcierto_para(p,<comienzo,fin>) :-
        sucede(e),tiempo(e,comienzo),inicia(e,p),
        sucede(e'),tiempo(e',fin),termina(e',p),
        despues(fin,comienzo),
        no interrumpido_durante(p,<comienzo,fin>).
```

(10)

```
cierto_para_siempre_desde(p,t) :-
        sucede(e),tiempo(e,t),inicia(e,p),
        no interrumpido_despues(p,t).
```

(11)

```
Icierto_para(p,<comienzo,fin>) :-
        Mcierto_para(p,<a,b>),
        igual_o_despues(comienzo,a),
        igual_o_despues(b,fin).
```

(12)

```
Icierto_para(p,<comienzo,fin>) :-
        cierto_para_siempre_desde(p,t),
        igual_o_despues(comienzo,t),despues(fin,comienzo).
```

(13)

```
cierto_en(p,t) :-
        cierto_para_siempre_desde(p,t'),despues(t,t').
```

(14)

```
cierto_en(p,t) :-
        Mcierto_para(p,<comienzo,fin>),
        despues(t,comienzo),igual_o_despues(fin,t).
```

(15)

```
interrumpido_durante(p,<comienzo,fin>) :-
        sucede(e),tiempo(e,t),
        despues(t.comienzo),despues(fin,t),
        termina(e,p).
```

(16)

```
interrumpido_despues(p,t) :-
        sucede(e),tiempo(e,t'),
        despues(t',t),termina(e,p).
```

(17)

```
tiempo(e,t1) :-
        tiempo(e,t2),grano_grueso_de(t1,t2).
```

(18)

```
despues(tiempo1,tiempo2) :-
        valor_metrico(tiempo1,metrica1,valor1),
        valor_metrico(tiempo2,metrica2,valor2),
        factor_de_conversion(metrica1,metrica2,factor),
        valor1 > (valor2*factor),
        no grano_grueso_de(tiempo2,tiempo1).
```

(19)

```
despues(tiempo1,tiempo2) :-
        valor_metrico(tiempo1,metrica,valor1),
        valor_metrico(tiempo2,metrica,valor2),
        valor1 > valor2.
```

(20)

```
igual_o_despues(tiempo,tiempo).
```

(21)

```
igual_o_despues(tiempo1,tiempo2) :-
        despues(tiempo1,tiempo2).
```

Anexo F

LISTADO EN C DEL MOTOR DE INFERENCIA DEL (λ,T) EN EL DOMINIO DEL CONTROL DE TRAFICO URBANO

```
/*************************************************************************
*                                                                        *
*          SIMULADOR CUALITATIVO DE TRAFICO VERSION 1.0                  *
*                                                                        *
*            Autor: Enrique Vicente Bonet Esteban                        *
*                                                                        *
*                  OCT/LISITT 1991                                       *
*                                                                        *
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <netdb.h>

#define INF 0xFFFF
#define MAX_S 100
```

```
typedef struct ESTADO{
  unsigned valor; /*Estado del tramo 1 .. 8*/
  float fin; /*Posicion final del trozo*/
  struct ESTADO *sig; /*Puntero al siguiente estado*/
};

typedef struct CALLE{
  unsigned codigo; /*Numero de la calle*/
  int longitud; /*Longitud de la calle*/
  unsigned carriles; /*Numero de carriles de la calle*/
  unsigned nodo1; /*Cruce inicial de la calle*/
  unsigned nodo2; /*Cruce final de la calle*/
  struct ESTADO *estado; /*Puntero a la lista que guarda el estado de la
                calle*/
};

typedef struct ENTRADA{
  unsigned codigo; /*Numero del tramo asociado a esta entrada*/
  unsigned valor; /*Estado de la entrada 1 .. 8*/
};

typedef struct SALIDA2{
  unsigned codigo; /*Numero del tramo asociado a esta salida*/
};

typedef struct CICLO{
  unsigned ciclo; /*Numero de ciclo*/
  unsigned duracion; /*Duracion en segundos del ciclo*/
};

typedef struct CCI{
  unsigned codigo; /*Numero de cci*/
  unsigned ciclo; /*Ciclo actual de trabajo*/
  unsigned duracion; /*Duracion del ciclo actual*/
  unsigned modif; /*0 No se ha modificado el ciclo, 1 se ha modificado*/
  unsigned tiempo; /*Tiempo actual ejecutado del ciclo*/
  unsigned num_cruce; /*Numero de cruces asociados a la cci*/
```

```
  unsigned *cruce; /*Puntero a la lista de cruces asociados a la cci*/
};

typedef struct FASE{
  unsigned fase; /*Numero de fase*/
  unsigned duracion; /*Duracion de la fase en % por mil.*/
};

typedef struct REPARTO{
  unsigned reparto; /*Numero de reparto*/
  unsigned num_fase; /*Numero de fases*/
  struct FASE *fase; /*Puntero a las fases del reparto*/
};

typedef struct CRUCE{
  unsigned codigo; /*Numero de cruce*/
  unsigned ciclo_actual; /*Numero de ciclo*/
  unsigned coord_actual; /*Coordinacion del cruce*/
  unsigned reparto_actual; /*Reparto del cruce*/
  unsigned modif; /*0 No se ha modificado, 1 se ha modificado*/
  unsigned num_reparto; /*Numero de repartos por cruce*/
  struct REPARTO *reparto; /*Puntero a la lista de repartos del cruce*/
};

typedef struct FASE2{
  unsigned fase; /*Numero de fase*/
  unsigned duracion; /*Duracion de la fase en segundos*/
};

typedef struct ICON{
  unsigned codigo; /*Numero del cruce*/
  unsigned num_fase; /*Numero de fases*/
  unsigned fase_actual; /*Fase actual en ejecucion*/
  unsigned tiempo; /*Tiempo actual ejecutado de la fase*/
  struct FASE2 *fase; /*Puntero a las fases del icon*/
};

typedef struct COEF_ENT{
```

```
  unsigned calle; /*Numero de la calle conectada al cruce*/
  unsigned estado; /*Estado del tramo final de dicha calle*/
  unsigned porcentaje; /*Porcentaje de giro respecto a la salida relacionada*/
  float coef; /*Coeficiente de giro respecto a la salida relacionada*/
  float demanda; /*Demanda de la entrada al cruce*/
};

typedef struct SALIDA{
  unsigned calle; /*Numero de la calle conectada al cruce*/
  unsigned estado; /*Estado del tramo inicial de dicha calle*/
  float demanda; /*Demanda de la entrada al cruce*/
  unsigned num_entrada; /*Numero de entradas a esta salida*/
  struct COEF_ENT *coef; /*Puntero a la estructura de las entradas al cruce*/
};

typedef struct SEMAFORO{
  unsigned calle; /*Numero de la calle*/
  unsigned estado; /*Estado de la calle*/
  unsigned semaforo;/*Estado del semaforo, 0 rojo, 1 verde*/
};

typedef struct FASE3{
  unsigned fase; /*Numero de la fase*/
  unsigned num_semaforo; /*Numero de semaforos existentes*/
  unsigned num_salida; /*Numero de salidas existentes*/
  struct SEMAFORO *semaforo; /*Puntero a la estructura de los semaforos*/
  struct SALIDA *salida; /*Puntero a la estructura de las salidas*/
};

typedef struct IFIS{
  unsigned codigo; /*Numero del cruce*/
  unsigned fase_actual; /*Fase actual del ifis*/
  unsigned num_fase; /*Numero de fases del cruce*/
  struct FASE3 *fase; /*Puntero a la estructura de las fases*/
};

typedef struct TIEMPO{
  unsigned hora; /*Hora de la simulacion*/
```

```
  unsigned min; /*Minutos de la simulacion*/
  unsigned seg; /*Segundos de la simulacion*/
};

float tabla[8][8]={
#include "tabla.h"   /*Tabla con las velocidades de frontera entre estados*/
};

int sock;
int n_msg=0;
int n_global;
int msgsock[MAX_S];
int stop=0;

/*************************************************************************
*              RUTINAS DE TRATAMIENTO DE ERRORES                        *
*************************************************************************/

/*-------------------------------------------------------------------
Muestra un mensaje de error, cierra los ficheros, etc., y termina la
ejecucion del programa.

Parametros:   unsigned char tipo   Tipo de error sucedido.
          FILE *fichero   Fichero a cerrar (si es necesario).
          char *nombre   Nombre del fichero donde sucedio el error.
Return:   Ninguno.
-------------------------------------------------------------------*/

void Error(tipo,fichero,nombre)

unsigned char tipo;
FILE *fichero;
char *nombre;

{
  (void)fprintf(stderr,"\nError en programa Sim_cual.\n");

  switch(tipo)
```

```
  {
    case 0:(void)fprintf(stderr,
             "Error reservando memoria en fichero: %s.\n",nombre);
           (void)fclose(fichero);
           break;
    case 1:(void)fprintf(stderr,"Error abriendo el fichero: %s\n",nombre);
           break;
    case 2:(void)fprintf(stderr,"Error leyendo el fichero: %s\n",nombre);
           (void)fclose(fichero);
           break;
    case 3:(void)fprintf(stderr,"Error cerrando el fichero: %s\n",nombre);
           break;
    default:(void)fprintf(stderr,"Error desconocido.\n");
            break;
  }

  exit(0);
}

/*************************************************************************
*            RUTINAS DE CARGA DE LAS BASES DE DATOS                     *
*************************************************************************/

/*-------------------------------------------------------------------
Lee las calles del fichero con todas sus datos.

Parametros:  char *nombre   Nombre del fichero que contiene las
                            propiedades de las calles.
             unsigned *num_calle   Puntero al entero que guardara el
                                   numero de calles.
Return:  struct CALLE *   Puntero al array que contiene las calles.
-------------------------------------------------------------------*/

struct CALLE *CargarCalles(nombre,num_calle)

char *nombre;
unsigned *num_calle:
```

```
{
  struct CALLE *calle;
  FILE *fichero;
  unsigned i;

  if ((fichero=fopen(nombre,"r"))==NULL)
    Error(1,fichero,nombre);

  if (fscanf(fichero,"CALLES: %u\n",num_calle)!=1)
    Error(2,fichero,nombre);

  if ((calle=(struct CALLE *)calloc(*num_calle,sizeof(struct CALLE)))==NULL)
    Error(0,fichero,nombre);

  for(i=0;i<*num_calle;i++)
  {
    if (fscanf(fichero,"%u:%d:%d:%u:%u\n",&calle[i].codigo,&calle[i].longitud,
      &calle[i].carriles,&calle[i].nodo1,&calle[i].nodo2)!=5)
      Error(2,fichero,nombre);
    calle[i].estado=NULL;
  }

  if (fclose(fichero)!=0)
    Error(3,fichero,nombre);

  return(calle);
}

/*-------------------------------------------------------------------------
Lee la entradas del fichero con todos sus datos.

Parametros:  char *nombre   Nombre del fichero que contiene las
                            propiedades de las entradas.
             unsigned *num_entrada   Puntero al entero que contendra el
                            numero de entradas.
Return:  struct ENTRADA *   Puntero al array que contiene las
                            entradas.
-------------------------------------------------------------------------*/
```

```
struct ENTRADA *CargarEntradas(nombre,num_entrada)

char *nombre;
unsigned *num_entrada;

{
  struct ENTRADA *entrada;
  FILE *fichero;
  unsigned i;

  if ((fichero=fopen(nombre,"r"))==NULL)
    Error(1,fichero,nombre);

  if (fscanf(fichero,"ENTRADAS: %u\n",num_entrada)!=1)
    Error(2,fichero,nombre);

  if ((entrada=(struct ENTRADA *)calloc(*num_entrada,sizeof(struct ENTRADA)))
    ==NULL)
    Error(0,fichero,nombre);

  for(i=0;i<*num_entrada;i++)
    if (fscanf(fichero,"%u:%u\n",&entrada[i].codigo,&entrada[i].valor)!=2)
      Error(2,fichero,nombre);

  if (fclose(fichero)!=0)
    Error(3,fichero,nombre);

  return(entrada);
}

/*---------------------------------------------------------------------
Lee las salidas del fichero con todos sus datos.

Parametros:  char *nombre   Nombre del fichero que contiene las
                            propiedades de las salidas.
             unsigned *num_salida   Puntero al entero que contendra el
                            numero de salidas.
```

```
Return:   struct SALIDA2 *   Puntero al array que contiene las salida.
-------------------------------------------------------------------------*/

struct SALIDA2 *CargarSalidas(nombre,num_salida)

char *nombre;
unsigned *num_salida;

{
  struct SALIDA2 *salida;
  FILE *fichero;
  unsigned i;

  if ((fichero=fopen(nombre,"r"))==NULL)
    Error(1,fichero,nombre);

  if (fscanf(fichero,"SALIDAS: %u\n",num_salida)!=1)
    Error(2,fichero,nombre);

  if ((salida=(struct SALIDA2 *)calloc(*num_salida,sizeof(struct SALIDA2)))
    ==NULL)
    Error(0,fichero,nombre);

  for(i=0;i<*num_salida;i++)
    if (fscanf(fichero,"%u\n",&salida[i].codigo)!=1)
      Error(2,fichero,nombre);

  if (fclose(fichero)!=0)
    Error(3,fichero,nombre);

  return(salida);
}

/*-------------------------------------------------------------------------
Carga los ciclos de la red.

Parametros:   char *nombre   Nombre del fichero que contiene los ciclos.
              unsigned *num_ciclo   Puntero al entero que guarda el numero
```

```
                        de ciclos.
Return:  struct CICLO *  Puntero al array que guarda los ciclos.
-------------------------------------------------------------------------*/

struct CICLO *CargarCiclo(nombre,num_ciclo)

char *nombre;
unsigned *num_ciclo;

{
  struct CICLO *ciclo;
  FILE *fichero;
  int i;

  if ((fichero=fopen(nombre,"r"))==NULL)
    Error(1,fichero,nombre);

  if (fscanf(fichero,"CICLOS: %u\n",num_ciclo)!=1)
    Error(2,fichero,nombre);

  if ((ciclo=(struct CICLO *)calloc(*num_ciclo,sizeof(struct CICLO)))==NULL)
    Error(0,fichero,nombre);

  for(i=0;i<*num_ciclo;i++)
    if (fscanf(fichero,"%u:%u\n",&ciclo[i].ciclo,&ciclo[i].duracion)!=2)
      Error(2,fichero,nombre);

  if (fclose(fichero)!=0)
    Error(3,fichero,nombre);

  return(ciclo);
}

/*-------------------------------------------------------------------------
Carga las cci de la red con sus cruces asociados.

Parametros:  char *nombre  Nombre del fichero que contiene los datos
                        de las cci.
```

```
        unsigned *num_cci   Puntero al entero que contiene el numero
                            de cci existentes.
Return:   struct CCI *   Puntero al array que contiene las cci.
-------------------------------------------------------------------------*/

struct CCI *CargarCci(nombre,num_cci)

char *nombre;
unsigned *num_cci;

{
  struct CCI *cci;
  FILE *fichero;
  int i,j;

  if ((fichero=fopen(nombre,"r"))==NULL)
    Error(1,fichero,nombre);

  if (fscanf(fichero,"CCIS: %u\n",num_cci)!=1)
    Error(2,fichero,nombre);

  if ((cci=(struct CCI *)calloc(*num_cci,sizeof(struct CCI)))==NULL)
    Error(0,fichero,nombre);

  for(i=0;i<*num_cci;i++)
  {
    if (fscanf(fichero,"%u:%u\n",&cci[i].codigo,&cci[i].num_cruce)!=2)
      Error(2,fichero,nombre);
    cci[i].ciclo=0;
    cci[i].duracion=0;
    cci[i].modif=0;
    cci[i].tiempo=0;
    if ((cci[i].cruce=(unsigned *)calloc(cci[i].num_cruce,
      sizeof(unsigned)))==NULL)
      Error(0,fichero,nombre);
    for(j=0;j<cci[i].num_cruce;j++)
      if (j<(cci[i].num_cruce-1))
      {
```

```
        if (fscanf(fichero,"%u:",&cci[i].cruce[j])!=1)
          Error(2,fichero,nombre);
      }
      else
        if (fscanf(fichero,"%u\n",&cci[i].cruce[j])!=1)
          Error(2,fichero,nombre);
  }

  if (fclose(fichero)!=0)
    Error(3,fichero,nombre);

  return(cci);
}

/*---------------------------------------------------------------------------
Carga los cruces de la red.

Parametros:  char *nombre   Nombre del fichero a cargar.
             unsigned *num_cruce   Puntero al entero que guarda el numero
                                   de cruces.
             struct ICON **icon   Puntero al puntero que guardar la
                                  estructura de los ICON.
Return:  struct CRUCE *   Puntero al array que guarda la estructura de
                          los cruces.
---------------------------------------------------------------------------*/

struct CRUCE *CargarCruce(nombre,num_cruce,icon)

char *nombre;
unsigned *num_cruce;
struct ICON **icon;

{
  struct CRUCE *cruce;
  FILE *fichero;
  unsigned i,j,k;

  if ((fichero=fopen(nombre,"r"))==NULL)
```

```
  Error(1,fichero,nombre);

if (fscanf(fichero,"CRUCES: %u\n",num_cruce)!=1)
  Error(2,fichero,nombre);

if (((*icon)=(struct ICON *)calloc(*num_cruce,sizeof(struct ICON)))==NULL)
  Error(0,fichero,nombre);

if ((cruce=(struct CRUCE *)calloc(*num_cruce,sizeof(struct CRUCE)))==NULL)
  Error(0,fichero,nombre);

for(i=0;i<*num_cruce;i++)
{
  if (fscanf(fichero,"%u:%u\n",&cruce[i].codigo,&cruce[i].num_reparto)!=2)
    Error(2,fichero,nombre);
  cruce[i].ciclo_actual=0;
  cruce[i].coord_actual=0;
  cruce[i].reparto_actual=0;
  cruce[i].modif=0;
  (*icon)[i].codigo=cruce[i].codigo;
  (*icon)[i].num_fase=0;
  (*icon)[i].fase_actual=0;
  (*icon)[i].tiempo=0;
  (*icon)[i].fase=NULL;
  if ((cruce[i].reparto=(struct REPARTO *)calloc
    (cruce[i].num_reparto,sizeof(struct REPARTO)))==NULL)
    Error(0,fichero,nombre);
  for(j=0;j<cruce[i].num_reparto;j++)
  {
    if (fscanf(fichero,"%u:%u\n",&cruce[i].reparto[j].reparto,
      &cruce[i].reparto[j].num_fase)!=2)
      Error(2,fichero,nombre);
    if ((cruce[i].reparto[j].fase=(struct FASE *)calloc
      (cruce[i].reparto[j].num_fase,sizeof(struct FASE)))==NULL)
      Error(0,fichero,nombre);
    for(k=0;k<cruce[i].reparto[j].num_fase;k++)
      if (fscanf(fichero,"%u:%u\n",&cruce[i].reparto[j].fase[k].fase,
        &cruce[i].reparto[j].fase[k].duracion)!=2)
```

```
        Error(2,fichero,nombre);
    }
  }

  if (fclose(fichero)!=0)
    Error(3,fichero,nombre);

  return(cruce);
}

/*---------------------------------------------------------------------
Carga los ifis de la red.

Parametros:  char *nombre   Nombre del fichero a cargar.
             unsigned num_cruce   Numero de cruces que existen en la red.
Return:  struct IFIS *   Puntero al array de estructuras que guarda
                         los ifis.
---------------------------------------------------------------------*/

struct IFIS *CargarIfis(nombre,num_cruce)

char *nombre;
unsigned num_cruce;

{
  struct IFIS *ifis;
  FILE *fichero;
  unsigned i,j,k,l;

  if ((fichero=fopen(nombre,"r"))==NULL)
    Error(1,fichero,nombre);

  if ((ifis=(struct IFIS *)calloc(num_cruce,sizeof(struct IFIS)))==NULL)
    Error(0,fichero,nombre);

  for(i=0;i<num_cruce;i++)
  {
    if (fscanf(fichero,"%u:%u\n",&ifis[i].codigo,&ifis[i].num_fase)!=2)
```

```
  Error(2,fichero,nombre);
ifis[i].fase_actual=1;
if ((ifis[i].fase=(struct FASE3 *)calloc(ifis[i].num_fase,
  sizeof(struct FASE3)))==NULL)
  Error(0,fichero,nombre);
for(j=0;j<ifis[i].num_fase;j++)
{
  if (fscanf(fichero,"%u:%u:%u\n",&ifis[i].fase[j].fase,
    &ifis[i].fase[j].num_semaforo,&ifis[i].fase[j].num_salida)!=3)
    Error(2,fichero,nombre);
  if ((ifis[i].fase[j].semaforo=(struct SEMAFORO *)calloc(
    ifis[i].fase[j].num_semaforo,sizeof(struct SEMAFORO)))==NULL)
    Error(0,fichero,nombre);
  for(k=0;k<ifis[i].fase[j].num_semaforo;k++)
  {
    if (fscanf(fichero,"%u:%u\n",&ifis[i].fase[j].semaforo[k].calle,
      &ifis[i].fase[j].semaforo[k].semaforo)!=2)
      Error(2,fichero,nombre);
    ifis[i].fase[j].semaforo[k].estado=1;
  }
  if ((ifis[i].fase[j].salida=(struct SALIDA *)calloc(
    ifis[i].fase[j].num_salida,sizeof(struct SALIDA)))==NULL)
    Error(0,fichero,nombre);
  for(k=0;k<ifis[i].fase[j].num_salida;k++)
  {
    if (fscanf(fichero,"%u:%u\n",&ifis[i].fase[j].salida[k].calle,
      &ifis[i].fase[j].salida[k].num_entrada)!=2)
      Error(2,fichero,nombre);
    ifis[i].fase[j].salida[k].estado=1;
    ifis[i].fase[j].salida[k].demanda=0;
    if ((ifis[i].fase[j].salida[k].coef=(struct COEF_ENT *)calloc(
      ifis[i].fase[j].salida[k].num_entrada,sizeof(struct COEF_ENT)))
      ==NULL)
      Error(0,fichero,nombre);
    for(l=0;l<ifis[i].fase[j].salida[k].num_entrada;l++)
    {
      if (fscanf(fichero,"%u:%u\n",
        &ifis[i].fase[j].salida[k].coef[l].calle,
```

```
            &ifis[i].fase[j].salida[k].coef[l].porcentaje)!=2)
            Error(2,fichero,nombre);
          ifis[i].fase[j].salida[k].coef[l].estado=1;
          ifis[i].fase[j].salida[k].coef[l].demanda=0;
          ifis[i].fase[j].salida[k].coef[l].coef=0;
        }
      }
    }
  }

  if (fclose(fichero)!=0)
    Error(3,fichero,nombre);

  return(ifis);
}

/*************************************************************************
*                    RUTINAS DE LAS CALLES                    *
*************************************************************************/

/*-------------------------------------------------------------------
Calcula el tiempo minimo necesario para que se produzca un evento en la
calle.

Parametros:  struct CALLE *calle  Puntero a la calle de la que
                                  queremos calcular el tiempo.
Return:  unsigned  Tiempo hasta el proximo evento en la calle.
-------------------------------------------------------------------*/

unsigned CalculaTiempoEventoCalle(calle)

struct CALLE *calle;

{
  struct ESTADO *p,*q,*r;
  unsigned tiempo=INF,t;

  r=NULL;
```

```
 p=calle->estado;
 q=p->sig;

 if (q!=NULL)
   while (p!=NULL)
   {
     if (r==NULL)
       t=(tabla[p->valor-1][q->valor-1]<0) ?
         (unsigned)(p->fin/-tabla[p->valor-1][q->valor-1]) : INF;
     else
       if (q==NULL)
         t=(-tabla[r->valor-1][p->valor-1]<0) ?
           (unsigned)((p->fin-r->fin)/tabla[r->valor-1][p->valor-1])
           : INF;
       else
         t=(tabla[p->valor-1][q->valor-1]-tabla[r->valor-1][p->valor-1]<0)
           ? (unsigned)((p->fin-r->fin)/(tabla[r->valor-1][p->valor-1]
             -tabla[p->valor-1][q->valor-1]))
           : INF;
     if (t<tiempo)
       tiempo=t;
     r=p;
     p=q;
     if (q!=NULL)
       q=q->sig;
   }

 if (tiempo==0)
   tiempo++;

 return(tiempo);
}

/*---------------------------------------------------------------------------
Calcula el estado de una calle en un tiempo dado.

Parametros:  struct CALLE *calle  Puntero a la calle que queremos
                                  calcular su estado.
```

```
        unsigned tiempo   Tiempo que queremos avanzar la calle.
Return:  Ninguno.
-------------------------------------------------------------------------*/

void CalculaEstadoCalle(calle,tiempo)

struct CALLE *calle;
unsigned tiempo;

{
  struct ESTADO *p,*q;

  p=calle->estado;
  q=p->sig;

  while (q!=NULL)
  {
    p->fin+=tabla[p->valor-1][q->valor-1]*tiempo;

    if (p->fin<0) p->fin=0;
    if (p->fin>(float)calle->longitud) p->fin=(float)calle->longitud;

    p=q;
    q=q->sig;
  }

  q=NULL;
  p=calle->estado;
  while (p!=NULL)
  {
    if (q==NULL)
      if (p->fin<=0)
      {
        calle->estado=p->sig;
        free((void *)p);
        p=calle->estado;
      }
      else
```

```
      {
        q=p;
        p=p->sig;
      }
    else
      if (p->fin<=q->fin)
      {
        q->sig=p->sig;
        free((void *)p);
        p=q->sig;
      }
      else
      {
        q=p;
        p=p->sig;
      }
  }

  return;
}

/*****************************************************************************
*                    RUTINAS DE LAS ENTRADAS                    *
*****************************************************************************/

/*---------------------------------------------------------------------
Introduce el estado de la entrada en su tramo, siempre que ello sea
posible.

Parametros:  struct ENTRADA *entrada  Puntero a la entrada a simular.
             struct CALLE *calle  Puntero a la calle relacionada con
                                  la entrada.
Return:  Ninguno.
---------------------------------------------------------------------*/

void CalculaEntrada(entrada,calle)

struct ENTRADA *entrada;
```

```
struct CALLE *calle;

{
  struct ESTADO *p;

  if (tabla[entrada->valor-1][calle->estado->valor-1]>0)
  {
    if ((p=(struct ESTADO *)malloc(sizeof(struct ESTADO)))==NULL)
    {
      (void)fprintf(stderr,"\nError en programa Sim_cual.\n");
      (void)fprintf(stderr,"Error reservando memoria.\n");
      exit(0);
    }

    p->valor=entrada->valor;
    p->fin=0;
    p->sig=calle->estado;
    calle->estado=p;
  }

  return;
}

/*************************************************************************
*                    RUTINAS DE LAS SALIDAS                             *
*************************************************************************/

/*-------------------------------------------------------------------
Introduce el estado de la salida en un tramo, siempre que ello sea posible.

Parametros:    struct CALLE *calle   Puntero a la calle relacionada con
                                     la salida.
Return:  Ninguno.
-------------------------------------------------------------------*/

void CalculaSalida(calle)

struct CALLE *calle;
```

```
{
  struct ESTADO *p,*q;

  p=calle->estado;
  while (p->sig!=NULL)
    p=p->sig;

  if (tabla[p->valor][3]<0)
  {
    if ((p=(struct ESTADO *)malloc(sizeof(struct ESTADO)))==NULL)
    {
      (void)fprintf(stderr,"\nError en programa Sim_cual.\n");
      (void)fprintf(stderr,"Error reservando memoria.\n");
      exit(0);
    }

    p->valor=4;
    p->fin=(float)calle->longitud;
    p->sig=NULL;
    q=calle->estado;
    while (q->sig!=NULL)
      q=q->sig;
    q->sig=p;
  }

  return;
}

/*************************************************************************
*                    RUTINAS DE LAS CCI                    *
*************************************************************************/

/*-------------------------------------------------------------------
Calcula el tiempo que falta hasta que suceda un evento en una cci.

Parametros:  struct CCI *cci  Puntero a la cci de la que queremos
                              calcular el tiempo.
```

```
Return:   unsigned   Tiempo hasta el evento.
-------------------------------------------------------------------------*/

unsigned CalculaTiempoEventoCci(cci)

struct CCI *cci;

{
  return(cci->duracion-cci->tiempo);
}

/*-------------------------------------------------------------------------
Cambia los parametros de una cci si ello es necesario.

Parametros:   struct CCI *cci   Puntero a la cci que queremos cambiar.
          struct CICLO *ciclo   Puntero al array de estructuras que
                                contiene los ciclos.
          struct CRUCE *cruce   Puntero al array de estructuras que
                                contiene los cruces.
Return:   Ninguno.
-------------------------------------------------------------------------*/

void CalculaCambioCci(cci,ciclo,cruce)

struct CCI *cci;
struct CICLO *ciclo;
struct CRUCE *cruce;

{
  unsigned i;

  if (cci->modif)
  {
    cci->modif=0;
    cci->duracion=ciclo[cci->ciclo-1].duracion;
    for(i=0;i<cci->num_cruce;i++)
    {
      cruce[cci->cruce[i]-1].modif=1;
```

```
      cruce[cci->cruce[i]-1].ciclo_actual=cci->ciclo;
    }
  }

  return;
}

/*---------------------------------------------------------------------------
Simula la cci hasta el instante actual.

Parametros:  struct CCI *cci   Puntero a la cci que queremos simular.
             struct CICLO *ciclo   Puntero al array de estructuras
                                   que contiene los ciclos.
             struct CRUCE *cruce   Puntero al array de estructuras que
                                   contiene los cruces.
             unsigned tiempo   Tiempo a simular.
Return:  Ninguno.
---------------------------------------------------------------------------*/

void SimularCci(cci,ciclo,cruce,tiempo)

struct CCI *cci;
struct CICLO *ciclo;
struct CRUCE *cruce;
unsigned tiempo;

{
  cci->tiempo+=tiempo;

  if (cci->tiempo>=cci->duracion)
  {
    cci->tiempo-=cci->duracion;
    CalculaCambioCci(cci,ciclo,cruce);
  }

  return;
}
```

```
/*************************************************************************
*                    RUTINAS DE LOS CRUCES                              *
*************************************************************************/

/*-------------------------------------------------------------------------
Calcula el tiempo que falta hasta que suceda un evento en un cruce.

Parametros:  struct ICON *icon   Puntero al cruce del que queremos
                                 calcular el tiempo que falta hasta el
                                 evento.
Return:  unsigned   Tiempo que falta hasta el evento.
-------------------------------------------------------------------------*/

unsigned CalculaTiempoEventoCruce(icon)

struct ICON *icon;

{
  return(icon->fase[icon->fase_actual-1].duracion-icon->tiempo);
}

/*-------------------------------------------------------------------------
Cambia los parametros de un cruce si ello es necesario.

Parametros:  struct ICON *icon      Puntero al cruce.
             struct CRUCE *cruce    Puntero a la estructura que
                                    contiene los datos de todos los
                                    repartos, etc., del cruce
             struct CICLO *ciclo    Puntero al array de estructuras
                                    que contiene los ciclos.
Return:  Ninguno.
-------------------------------------------------------------------------*/

void CalculaCambioCruce(icon,cruce,ciclo)

struct ICON *icon;
struct CRUCE *cruce;
struct CICLO *ciclo;
```

```
{
  unsigned i;
  float jf,error;

  if (cruce->modif)
  {
    cruce->modif=0;
    if (icon->fase!=NULL)
      free((void *)icon->fase);
    icon->num_fase=cruce->reparto[cruce->reparto_actual-1].num_fase;
    if ((icon->fase=(struct FASE2 *)calloc(icon->num_fase,sizeof(struct
      FASE2)))==NULL)
    {
      (void)fprintf(stderr,"\nError en programa Sim_cual.\n");
      (void)fprintf(stderr,"Error reservando memoria.\n");
      exit(0);
    }

    icon->tiempo=ciclo[cruce->ciclo_actual-1].duracion-(unsigned)((float)
      cruce->coord_actual/1000*ciclo[cruce->ciclo_actual-1].duracion+0.5);
    icon->fase_actual=1;

    error=0;
    for(i=0;i<icon->num_fase;i++)
    {
      jf=(float)cruce->reparto[cruce->reparto_actual-1].fase[i].duracion
        /1000*ciclo[cruce->ciclo_actual-1].duracion;
      icon->fase[i].duracion=(int)(jf+error+0.5);
      error=error+jf-icon->fase[i].duracion;
      icon->fase[i].fase=i+1;
    }

    for(i=0;i<icon->num_fase;i++)
      if (icon->fase[i].duracion>icon->tiempo)
      {
        icon->fase_actual=i+1;
        break;
```

```
      }
      else
        icon->tiempo-=icon->fase[i].duracion;
  }

  return;
}

/*---------------------------------------------------------------------------
Simula el cruce hasta el instante actual.

Parametros:  struct ICON *icon   Puntero al cruce a simular.
             struct CRUCE *cruce   Puntero a la estructura que contiene
                                   los datos de todos los repartos, etc.,
                                   del cruce.
             struct CICLO *ciclo   Puntero al array de estructuras que
                                   contiene los ciclos.
             unsigned tiempo   Tiempo a simular.
Return:  Ninguno.
---------------------------------------------------------------------------*/

void SimularCruce(icon,cruce,ciclo,tiempo)

struct ICON *icon;
struct CRUCE *cruce;
struct CICLO *ciclo;
unsigned tiempo;

{
  icon->tiempo+=tiempo;

  if (icon->tiempo>=icon->fase[icon->fase_actual-1].duracion)
  {
    icon->tiempo-=icon->fase[icon->fase_actual-1].duracion;
    if (icon->fase_actual<icon->num_fase)
      icon->fase_actual++;
    else
    {
```

```
      icon->fase_actual=1;
      CalculaCambioCruce(icon,cruce,ciclo);
    }
  }

  return;
}

/*************************************************************************
*                    RUTINAS DE LOS IFIS                    *
*************************************************************************/

/*-------------------------------------------------------------------------
Completa los datos de los ifis antes de simularlos.

Parametros:  struct IFIS *ifis   Puntero al ifis a completar
             struct CALLE *calle   Puntero al array que guarda las
                                   calles.
             struct ICON *icon   Puntero al icon relacionado con el
                                 ifis.
Return:  Ninguno.
-------------------------------------------------------------------------*/

void CompletarIfis(ifis,calle,icon)

struct IFIS *ifis;
struct CALLE *calle;
struct ICON *icon;

{
  struct ESTADO *p;
  struct FASE3 *fase;
  struct SALIDA *salida;
  struct COEF_ENT *coef;
  struct SEMAFORO *semaforo;
  unsigned i,j;

  ifis->fase_actual=icon->fase_actual;
```

```
  fase=&ifis->fase[ifis->fase_actual-1];
  for(i=0;i<fase->num_salida;i++)
  {
    salida=&fase->salida[i];
    salida->estado=calle[salida->calle-1].estado->valor;
    for(j=0;j<salida->num_entrada;j++)
    {
      coef=&salida->coef[j];
      p=calle[coef->calle-1].estado;
      while (p->sig!=NULL)
        p=p->sig;
      coef->estado=p->valor;
    }
  }

  for(i=0;i<fase->num_semaforo;i++)
  {
    semaforo=&fase->semaforo[i];
    p=calle[semaforo->calle-1].estado;
    while (p->sig!=NULL)
      p=p->sig;
    semaforo->estado=p->valor;
  }

  return;
}

/*--------------------------------------------------------------------------
Simula el comportamiento de los ifis.

Parametros:   struct IFIS *ifis   Puntero al ifis a simular.
          struct CALLE *calle   Puntero al array que guarda las
                                calles.
          struct ICON *icon   Puntero al icon.
Return:   Ninguno.
--------------------------------------------------------------------------*/
```

```
void SimularIfis(ifis,calle,icon)

struct IFIS *ifis;
struct CALLE *calle;
struct ICON *icon;

{
  static float flujo[2][8]={0,0.9,2.65,3.75,4.3,3.45,2.35,0.85,
                    0.9,2.65,3.75,4.3,3.45,2.35,0.85,0};

  struct ESTADO *p,*q;
  struct FASE3 *fase;
  struct SEMAFORO *semaforo;
  struct SALIDA *salida;
  struct COEF_ENT *coef;
  unsigned i,j,k,l,m,num_calle,solucion[2];
  float demanda;
  unsigned char salir;

  CompletarIfis(ifis,calle,icon);

  fase=&ifis->fase[ifis->fase_actual-1];
  for(i=0;i<fase->num_semaforo;i++)
  {
    semaforo=&fase->semaforo[i];
    if (semaforo->semaforo==0 && tabla[semaforo->estado-1][7]<0)
    {
      if ((p=(struct ESTADO *)malloc(sizeof(struct ESTADO)))==NULL)
      {
        (void)fprintf(stderr,"\nError en programa Sim_cual.\n");
        (void)fprintf(stderr,"Error reservando memoria.\n");
        exit(0);
      }
      q=calle[semaforo->calle-1].estado;
      while (q->sig!=NULL)
        q=q->sig;
      p->sig=NULL;
      p->valor=8;
```

```
    p->fin=q->fin;
    q->sig=p;
  }
}

for(i=0;i<fase->num_salida;i++)
{
  salida=&fase->salida[i];
  if (salida->estado<=4)
    salida->demanda=flujo[1][3];
  else
    salida->demanda=flujo[1][salida->estado-2];
  for(j=0;j<salida->num_entrada;j++)
  {
    coef=&salida->coef[j];
    if (coef->estado<=4)
      coef->demanda=flujo[1][coef->estado-1];
    else
      coef->demanda=flujo[1][3];
  }
}

for(i=0;i<fase->num_salida;i++)
{
  salida=&fase->salida[i];
  demanda=0;
  for(j=0;j<salida->num_entrada;j++)
    demanda+=salida->coef[j].demanda*salida->coef[j].coef;
  if (demanda>salida->demanda)
    for(j=0;j<salida->num_entrada;j++)
      salida->coef[j].demanda*=salida->demanda/demanda;
}

for(i=0;i<fase->num_semaforo;i++)
{
  semaforo=&fase->semaforo[i];
  if (semaforo->semaforo)
  {
```

```
    num_calle=semaforo->calle;
    demanda=flujo[1][3];

    for(j=0;j<fase->num_salida;j++)
    {
      salida=&fase->salida[j];
      for(k=0;k<salida->num_entrada;k++)
      {
        coef=&salida->coef[k];
        if ((num_calle==coef->calle) && (demanda>coef->demanda))
          demanda=coef->demanda;
      }
    }

    for(j=0;j<fase->num_salida;j++)
    {
      salida=&fase->salida[j];
      for(k=0;k<salida->num_entrada;k++)
      {
        coef=&salida->coef[k];
        if (num_calle==coef->calle)
          coef->demanda=demanda;
      }
    }
  }
}

for(i=0;i<fase->num_salida;i++)
{
  salida=&fase->salida[i];
  demanda=0;
  for(j=0;j<salida->num_entrada;j++)
  {
    coef=&salida->coef[j];
    demanda+=coef->demanda*coef->coef;
  }
  salida->demanda=demanda;
}
```

UNIVERSITAT DE VALÈNCIA BIBLIOTECA DE CIÈNCIES

```
for(i=0;i<fase->num_salida;i++)
{
  salida=&fase->salida[i];
  k=0;
  for(j=0;j<8;j++)
  {
    if (flujo[0][j]<flujo[1][j])
    {
      if ((flujo[0][j]<salida->demanda) && (flujo[1][j]>=salida->demanda))
        solucion[k++]=j+1;
    }
    else
      if ((flujo[1][j]<=salida->demanda) && (flujo[0][j]>salida->demanda))
        solucion[k++]=j+1;
  }
  j=0xffff;
  if (k==1)
  {
    if (salida->estado!=solucion[0])
      j=0;
  }
  else
    if (salida->estado<=4)
      j=0;
    else
      if (salida->estado==solucion[1])
        j=1;
      else
        j=0;

  if ((j!=0xffff) && (tabla[solucion[j]-1][salida->estado-1]>0))
  {
    if ((p=(struct ESTADO *)malloc(sizeof(struct ESTADO)))==NULL)
    {
      (void)fprintf(stderr,"\nError en programa Sim_cual.\n");
      (void)fprintf(stderr,"Error reservando memoria.\n");
      exit(0);
```

```
    }
    p->fin=0;
    p->valor=solucion[j];
    p->sig=calle[salida->calle-1].estado;
    calle[salida->calle-1].estado=p;
  }
}

for(i=0;i<fase->num_semaforo;i++)
{
  semaforo=&fase->semaforo[i];
  if (semaforo->semaforo)
  {
    num_calle=semaforo->calle;
    salir=1;

    for(j=0;j<fase->num_salida && salir;j++)
    {
      salida=&fase->salida[j];
      for(k=0;k<salida->num_entrada && salir;k++)
      {
        coef=&salida->coef[k];
        if (num_calle==coef->calle)
        {
          salir=0;
          m=0;
          for(l=0;l<8;l++)
          {
            if (flujo[0][l]<flujo[1][l])
            {
              if ((flujo[0][l]<coef->demanda) &&
                (flujo[1][l]>=coef->demanda))
                solucion[m++]=l+1;
            }
            else
              if ((flujo[1][l]<=coef->demanda) &&
                (flujo[0][l]>coef->demanda))
                solucion[m++]=l+1;
```

```
        }
        l=0xffff;
        if (m==1)
        {
          if (coef->estado!=solucion[0])
            l=0;
        }
        else
          if (coef->estado>=5)
            l=1;
          else
            if (coef->estado==solucion[0])
              l=0;
            else
              l=1;

        if ((l!=0xffff) && (tabla[coef->estado-1][solucion[l]-1]<0))
        {
          if ((p=(struct ESTADO *)malloc(sizeof(struct ESTADO)))
            ==NULL)
          {
            (void)fprintf(stderr,"\nError en programa Sim_cual.\n");
            (void)fprintf(stderr,"\nError reservando memoria.\n");
            exit(0);
          }

          p->fin=(float)calle[coef->calle-1].longitud;
          p->valor=solucion[l];
          p->sig=NULL;
          q=calle[coef->calle-1].estado;
          while (q->sig!=NULL)
            q=q->sig;
          q->sig=p;
        }
      }
    }
  }
}
```

```
  }

  return;
}

/*************************************************************************
*          RUTINAS DE INICIALIZACION DE LA SIMULACION          *
*************************************************************************/

/*-------------------------------------------------------------------
Carga el fichero que contiene los estados iniciales de cada uno de los
objetos que tenemos.

Parametros:  char *nombre   Nombre del fichero que contiene los datos
                            iniciales de los objetos.
             struct CALLE *calle   Puntero a las calles.
             unsigned num_calle   Numero de calles.
             struct CCI *cci   Puntero a las cci.
             unsigned num_cci   Numero de cci.
             struct CRUCE *cruce   Puntero a los cruces.
             struct ICON *icon   Puntero a los icon.
             struct IFIS *ifis   Puntero a los ifis.
             unsigned num_cruce   Numero de cruces.
             struct CICLO *ciclo   Puntero a los ciclos.
Return:  Ninguno.
-------------------------------------------------------------------*/

void CargarEstadoInicial(nombre,calle,num_calle,cci,num_cci,cruce,icon,ifis,
  num_cruce,ciclo)

char *nombre;
struct CALLE *calle;
unsigned num_calle;
struct CCI *cci;
unsigned num_cci;
struct CRUCE *cruce;
struct ICON *icon;
struct IFIS *ifis;
```

```
unsigned num_cruce;
struct CICLO *ciclo;

{
  FILE *fichero;
  unsigned i,j,k,l;

  if ((fichero=fopen(nombre,"r"))==NULL)
    Error(1,fichero,nombre);

  for(i=0;i<num_calle;i++)
  {
    if ((calle[i].estado=(struct ESTADO *)malloc(sizeof(struct ESTADO)))
      ==NULL)
      Error(0,fichero,nombre);
    calle[i].estado->fin=(float)calle[i].longitud;
    calle[i].estado->sig=NULL;
    if (fscanf(fichero,"%u\n",&calle[i].estado->valor)!=1)
      Error(2,fichero,nombre);
  }

  for(i=0;i<num_cci;i++)
  {
    cci[i].modif=1;
    if (fscanf(fichero,"%u\n",&cci[i].ciclo)!=1)
      Error(2,fichero,nombre);
  }

  for(i=0;i<num_cruce;i++)
  {
    cruce[i].modif=1;
    if (fscanf(fichero,"%u:%u\n",&cruce[i].coord_actual,
      &cruce[i].reparto_actual)!=2)
      Error(2,fichero,nombre);
  }

  if (fclose(fichero)!=0)
    Error(3,fichero,nombre);
```

```
  for(i=0;i<num_cruce;i++)
    for(j=0;j<ifis[i].num_fase;j++)
      for(k=0;k<ifis[i].fase[j].num_salida;k++)
        for(l=0;l<ifis[i].fase[j].salida[k].num_entrada;l++)
          ifis[i].fase[j].salida[k].coef[l].coef=0.01*
            ifis[i].fase[j].salida[k].coef[l].porcentaje*
            calle[ifis[i].fase[j].salida[k].coef[l].calle-1].carriles/
            calle[ifis[i].fase[j].salida[k].calle-1].carriles;

  for(i=0;i<num_cci;i++)
    CalculaCambioCci(&cci[i],ciclo,cruce);

  for(i=0;i<num_cruce;i++)
    CalculaCambioCruce(&icon[i],&cruce[i],ciclo);

  for(i=0;i<num_cruce;i++)
    SimularIfis(&ifis[i],calle,&icon[i]);

  return;
}

/*************************************************************************
*                RUTINAS GENERALES DEL PROGRAMA                 *
*************************************************************************/

/*-------------------------------------------------------------------
Rutina que convierte un unsigned en una cadena de caracteres.

Parametros:  unsigned numero   Numero a convertir a cadena de caracteres.
             char *cadena   Cadena de caracteres donde se va a guardar el
                            numero convertido.
Return:  char *   Puntero a la cadena de caracteres donde se ha guardado
                  el numero convertido.
-------------------------------------------------------------------*/

char *ITOA(numero,cadena)
```

```
unsigned numero;
char *cadena;

{
  char auxiliar[15];
  int i=sizeof(auxiliar)-1,j=0;

  do
  {
    auxiliar[i--]=(char)(numero%10)+'0';
    numero/=10;
  }
  while (numero);

  while (i<(sizeof(auxiliar)-1))
    cadena[j++]=auxiliar[++i];
  cadena[j]='\0';

  return(cadena);
}

/*---------------------------------------------------------------------------
Rutina que detiene la ejecucion de la simulacion.

Parametros:  Ninguno.
Return:  Ninguno.
---------------------------------------------------------------------------*/

void Parar()

{
  stop=!stop;

  return;
}

/*---------------------------------------------------------------------------
Rutina que elimina el socket usado por un proceso que termina.
```

```
Parametros:  Ninguno.
Return:  Ninguno.
-------------------------------------------------------------------------*/

void Continuar()

{
  int i;

  (void)close(msgsock[n_global]);

  for(i=n_global;i<n_msg;i++)
    msgsock[i]=msgsock[i+1];

  n_msg--;

  return;
}

/*-------------------------------------------------------------------------
Rutina que mira si algun proceso esta intentando conectarse al socket.

Parametros:  Ninguno.
Return:  Ninguno.
-------------------------------------------------------------------------*/

void Conexion()

{
  int num_s,n;
  fd_set ready;
  struct timeval tiempo;

  (void)listen(sock,MAX_S);
  FD_ZERO(&ready);
  tiempo.tv_sec=0;
  tiempo.tv_usec=1;
```

```
  FD_SET(sock,&ready);
  if ((num_s=select(FD_SETSIZE,&ready,(fd_set *)0,(fd_set *)0,&tiempo))<0)
    return;

  for(n=n_msg;(n<(n_msg+num_s)) && (n<MAX_S);n++)
    if ((msgsock[n]=accept(sock,(struct sockaddr *)0,(int *)0))==-1)
      return;

  n_msg+=num_s;

  return;
}

/*-------------------------------------------------------------------------
Rutina que escribe un dato de salida en los sockets existentes.

Parametros:  char *buf   Puntero al string que contiene los datos de
                         salida.
             int l_buf   Longitud de los datos a escribir.
Return:  Ninguno.
-------------------------------------------------------------------------*/
void Escribir(buf,l_buf)

char *buf;
int l_buf;

{
  for(n_global=0;n_global<n_msg;n_global++)
    (void)write(msgsock[n_global],buf,l_buf);

  return;
}

/*-------------------------------------------------------------------------
Rutina que mira si se esta intentado conectar algun proceso y llama a la
funcion que escribe los datos en el socket.

Parametros:  char *buf   Puntero a los datos a escribir en el socket.
```

```
Return:  Ninguno.
-------------------------------------------------------------------------*/

void EscribirSocket(buf)

char *buf;

{
  while (stop)
    Conexion();

  Escribir(buf,strlen(buf));

  return;
}

/*-------------------------------------------------------------------------
Rutina que cambia el tiempo de segundos a horas, minutos y segundos.

Parametros:  unsigned long tiempo_total   Tiempo total de la simulacion.
          struct TIEMPO *tiempo   Puntero a la estructura que contiene
                                  la hora en formato de hh:mm:ss.
Return:  Ninguno.
-------------------------------------------------------------------------*/

void CambiarFormatoTiempo(tiempo_total,tiempo)

unsigned long tiempo_total;
struct TIEMPO *tiempo;

{
  tiempo->hora=(unsigned)(tiempo_total/3600);
  tiempo_total-=tiempo->hora*3600;
  tiempo->min=(unsigned)(tiempo_total/60);
  tiempo_total-=tiempo->min*60;
  tiempo->seg=(unsigned)tiempo_total;

  return;
```

```
}

/*---------------------------------------------------------------------
Rutina que muestra el estado de los elementos de la simulacion.

Parametros:  struct CALLE *calle   Puntero al array que guarda las calles.
             unsigned num_calle    Numero de calles existentes.
             struct IFIS *ifis     Puntero al array que guarda los ifis.
             unsigned num_cruce    Numero de cruces existentes.
             unsigned_long tiempo_total   Tiempo total simulado hasta el
                                          momento.
Return:  Ninguno.
---------------------------------------------------------------------*/

void MostrarSimulacion(calle,num_calle,ifis,num_cruce,tiempo_total)

struct CALLE *calle;
unsigned num_calle;
struct IFIS *ifis;
unsigned num_cruce;
unsigned long tiempo_total;

{
  unsigned i,j;
  struct TIEMPO tiempo;
  struct ESTADO *p,*q;
  struct SEMAFORO *semaforo;
  char salida[1000],cadena[15];

  Conexion();

  CambiarFormatoTiempo(tiempo_total,&tiempo);
  (void)strcpy(salida,"0 ");
  (void)strcat(salida,ITOA(tiempo.hora,cadena));
  (void)strcat(salida," ");
  (void)strcat(salida,ITOA(tiempo.min,cadena));
  (void)strcat(salida," ");
  (void)strcat(salida,ITOA(tiempo.seg,cadena));
```

```
(void)strcat(salida,"\n");
EscribirSocket(salida);

for(i=0;i<num_calle;i++)
{
  p=calle[i].estado;
  q=NULL;
  while (p!=NULL)
  {
    (void)strcpy(salida,ITOA(calle[i].codigo,cadena));
    (void)strcat(salida," ");
    if (q==NULL)
      (void)strcat(salida,"0 ");
    else
    {
      (void)strcat(salida,ITOA((unsigned)q->fin,cadena));
      (void)strcat(salida," ");
    }
    (void)strcat(salida,ITOA((unsigned)p->fin,cadena));
    (void)strcat(salida," ");
    (void)strcat(salida,ITOA((unsigned)calle[i].longitud,cadena));
    (void)strcat(salida," ");
    (void)strcat(salida,ITOA(p->valor,cadena));
    (void)strcat(salida,"\n");
    q=p;
    p=p->sig;
    EscribirSocket(salida);
  }
}

for(i=0;i<num_cruce;i++)
{
  semaforo=ifis[i].fase[ifis[i].fase_actual-1].semaforo;
  for(j=0;j<ifis[i].fase[ifis[i].fase_actual-1].num_semaforo;j++)
  {
    (void)strcpy(salida,"100 ");
    (void)strcat(salida,ITOA(semaforo[j].calle,cadena));
    if (semaforo[j].semaforo)
```

```
        (void)strcat(salida," 3 0 0\n");
      else
        (void)strcat(salida," 0 0 0\n");
      EscribirSocket(salida);
    }
  }

  return;
}

/*---------------------------------------------------------------------------
Libera la memoria asignada en el programa.

Parametros:  Punteros a las estructuras creadas en el programa, y tamano
             de aquellas que contienen en su interior punteros.
Return:  Ninguno.
---------------------------------------------------------------------------*/

void LiberarMemoria(calle,entrada,salida,ciclo,cci,cruce,icon,ifis,
  num_calle,num_cci,num_cruce)

struct CALLE *calle;
struct ENTRADA *entrada;
struct SALIDA2 *salida;
struct CICLO *ciclo;
struct CCI *cci;
struct CRUCE *cruce;
struct ICON *icon;
struct IFIS *ifis;
unsigned num_calle,num_cci,num_cruce;

{
  struct ESTADO *p,*q;
  unsigned i,j,k;

  for(i=0;i<num_calle;i++)
  {
    p=calle[i].estado;
```

```
  while (p!=NULL)
  {
    q=p;
    p=p->sig;
    free((void *)q);
  }
}
free((void *)calle);

free((void *)entrada);

free((void *)salida);

free((void *)ciclo);

for(i=0;i<num_cci;i++)
  free((void *)cci[i].cruce);
free((void *)cci);

for(i=0;i<num_cruce;i++)
{
  for(j=0;j<cruce[i].num_reparto;j++)
    free((void *)cruce[i].reparto[j].fase);
  free((void *)cruce[i].reparto);
}
free((void *)cruce);

for(i=0;i<num_cruce;i++)
  free((void *)icon[i].fase);
free((void *)icon);

for(i=0;i<num_cruce;i++)
{
  for(j=0;j<ifis[i].num_fase;j++)
  {
    for(k=0;k<ifis[i].fase[j].num_salida;k++)
      free((void *)ifis[i].fase[j].salida[k].coef);
    free((void *)ifis[i].fase[j].salida);
```

```
    }
    free((void *)ifis[i].fase);
  }
  free((void *)ifis);

  return;
}

/*************************************************************************
*                    RUTINAS DE LOS SOCKETS                    *
*************************************************************************/

/*-------------------------------------------------------------------------
Rutina que crea un socket.

Parametros:  int puerto  Puerto en que queremos crear el socket.
Return:  int  Numero de puerto donde se ha creado el socket.
-------------------------------------------------------------------------*/

int CrearSocket(puerto)

int puerto;

{
  struct sockaddr_in server;
  int length=sizeof(server);

  if ((sock=socket(AF_INET,SOCK_STREAM,0))<0)
  {
    (void)fprintf(stderr,"\nError en programa Sim_cual.\n");
    (void)fprintf(stderr,"Error creando el socket.\n");
    exit(0);
  }

  server.sin_family=AF_INET;
  server.sin_addr.s_addr=INADDR_ANY;
  server.sin_port=puerto;
  if (bind(sock,(struct sockaddr *)&server,length)<0)
```

```
  {
    (void)fprintf(stderr,"\nError en programa Sim_cual.\n");
    (void)fprintf(stderr,"Error creando el socket.\n");
    exit(0);
  }

  if (getsockname(sock,(struct sockaddr *)&server,&length)<0)
  {
    (void)fprintf(stderr,"\nError en programa Sim_cual.\n");
    (void)fprintf(stderr,"Error en numero de puerto.\n");
    exit(0);
  }

  return(ntohs(server.sin_port));
}

/*-------------------------------------------------------------------------
Rutina que inicializa las senales, y llama a la rutina que crea el socket.

Parametros:  int *sock   Puntero al entero que guardara el numero de
                         socket.
Return:  Ninguno.
-------------------------------------------------------------------------*/

void InicializarSocket()

{
  FILE *fichero;

  (void)signal(SIGPIPE,Continuar);

  if ((fichero=fopen("./DatosProlog","w"))==NULL)
  {
    (void)fprintf(stderr,"\nError en programa Sim_cual.\n");
    (void)fprintf(stderr,"Error creando el fichero de datos.\n");
    exit(0);
  }
```

```
  (void)fprintf(fichero,"%d\n%d\n",getpid(),CrearSocket(0));
  (void)fclose(fichero);

  return;
}

/*************************************************************************
*                RUTINA PRINCIPAL DEL PROGRAMA                *
*************************************************************************/

main(argc,argv)

int argc;
char **argv;

{
  struct CALLE *calle;
  struct ENTRADA *entrada;
  struct SALIDA2 *salida;
  struct CICLO *ciclo;
  struct CCI *cci;
  struct CRUCE *cruce;
  struct ICON *icon;
  struct IFIS *ifis;
  unsigned num_entrada,num_salida,num_calle,num_ciclo,num_cci,num_cruce;
  unsigned valor,tiempo;
  unsigned long tiempo_total=0,tiempo_pedido;
  unsigned i;

  if (argc!=10)
  {
    (void)fprintf(stderr,"\nError, argumentos necesarios no pasados.\n");
    (void)fprintf(stderr,"Argumentos necesarios: \n");
    (void)fprintf(stderr,"\tNombre del fichero de las calles.\n");
    (void)fprintf(stderr,"\tNombre del fichero de las entradas.\n");
    (void)fprintf(stderr,"\tNombre del fichero de las salidas.\n");
    (void)fprintf(stderr,"\tNombre del fichero de los ciclos.\n");
    (void)fprintf(stderr,"\tNombre del fichero de las cci.\n");
```

```
  (void)fprintf(stderr,"\tNombre del fichero de los cruces.\n");
  (void)fprintf(stderr,"\tNombre del fichero de los ifis.\n");
  (void)fprintf(stderr,"\tNombre del fichero de inicializacion.\n");
  (void)fprintf(stderr,
    "\tTiempo total de simulacion deseado (en segundos).\n");
  exit(0);
}

(void)signal(SIGUSR1,Parar);
InicializarSocket();

calle=CargarCalles(argv[1],&num_calle);
entrada=CargarEntradas(argv[2],&num_entrada);
salida=CargarSalidas(argv[3],&num_salida);
ciclo=CargarCiclo(argv[4],&num_ciclo);
cci=CargarCci(argv[5],&num_cci);
cruce=CargarCruce(argv[6],&num_cruce,&icon);
ifis=CargarIfis(argv[7],num_cruce);

CargarEstadoInicial(argv[8],calle,num_calle,cci,num_cci,cruce,icon,ifis,
  num_cruce,ciclo);

tiempo_pedido=atol(argv[9]);

MostrarSimulacion(calle,num_calle,ifis,num_cruce,tiempo_total);

do
{
  tiempo=INF;

  for(i=0;i<num_entrada;i++)
    CalculaEntrada(&entrada[i],&calle[entrada[i].codigo-1]);

  for(i=0;i<num_salida;i++)
    CalculaSalida(&calle[salida[i].codigo-1]);

  for(i=0;i<num_calle;i++)
    if ((valor=CalculaTiempoEventoCalle(&calle[i]))<tiempo)
```

```
        tiempo=valor;

    for(i=0;i<num_cci;i++)
      if ((valor=CalculaTiempoEventoCci(&cci[i]))<tiempo)
        tiempo=valor;

    for(i=0;i<num_cruce;i++)
      if ((valor=CalculaTiempoEventoCruce(&icon[i]))<tiempo)
        tiempo=valor;

    tiempo_total+=tiempo;

    if (tiempo<INF)
    {
      for(i=0;i<num_calle;i++)
        CalculaEstadoCalle(&calle[i],tiempo);
      for(i=0;i<num_cci;i++)
        SimularCci(&cci[i],ciclo,cruce,tiempo);
      for(i=0;i<num_cruce;i++)
        SimularCruce(&icon[i],&cruce[i],ciclo,tiempo);
      for(i=0;i<num_cruce;i++)
        SimularIfis(&ifis[i],calle,&icon[i]);
    }

    MostrarSimulacion(calle,num_calle,ifis,num_cruce,tiempo_total);
  }
  while (tiempo_total<tiempo_pedido);

  LiberarMemoria(calle,entrada,salida,ciclo,cci,cruce,icon,ifis,num_calle,
    num_cci,num_cruce);

  return(0);
}
```

Anexo G

CONTROL DE TRAFICO URBANO

Para que un sistema de regulación de tráfico urbano resulte eficaz, se debe efectuar un análisis permanente del tráfico, ya que las decisiones a tomar dependen del estado del tráfico en cada instante, a modo de ejemplo, en los casos 1 y 2 de la figura G.1, se muestra de qué forma una decisión tan simple como es el encadenar la puesta en verde de dos semáforos consecutivos, cambia totalmente de unos casos a otros; en el primer caso, se debe poner en verde en primer lugar el semáforo anterior (en el sentido de circulación) y después el posterior, de forma que la cola de éste pueda atravesar el semáforo antes de que sea alcanzada por los vehículos que provienen del semáforo anterior (en el argot de tráfico este tipo de coordinación se llama una onda verde). En el segundo caso de la figura G.1 la puesta en verde de los semáforos se debe hacer en orden inverso (lo que se conoce como onda roja).

La recogida de datos es posible gracias a los detectores repartidos sobre la red, agrupados en unos puntos de medida estratégicamente situados, y que son capaces de detectar la presencia de vehículos y proporcionar la medida de diversos parámetros como son velocidad media, número de vehículos en un intervalo de tiempo, tiempo de ocupación, etc. Para asegurar una coherencia en la toma de decisiones, es necesario centralizar el conjunto de todas estas informaciones, y esta centralización se lleva a cabo en la sala de control de tráfico.

Un sistema centralizado de regulación de tráfico urbano tiene dos funciones básicas:

a) Analizar la situación del tráfico y adoptar las decisiones más adecuadas en cada caso.

b) Gestionar la transmisión de estas decisiones entre la sala de control y los reguladores de tráfico (semáforos, señales orientables, agentes de policía, carriles reversibles, etc.).

Estas funciones se llevan a cabo gracias a una serie de soluciones de hardware y de software especialmente diseñadas para este tipo de aplicación. En la figura G.2 se puede observar la arquitectura del sistema de regulación de tráfico utilizado en la ciudad de Valencia, a modo de ejemplo de los sistema instalados en Europa. Desde la sala de control de tráfico a los semáforos instalados en las calles, hay diferentes elementos de control que tienen el objetivo de que en caso de que aparezca algún tipo de fallo, el sistema pueda funcionar con los requerimientos de seguridad básicos.

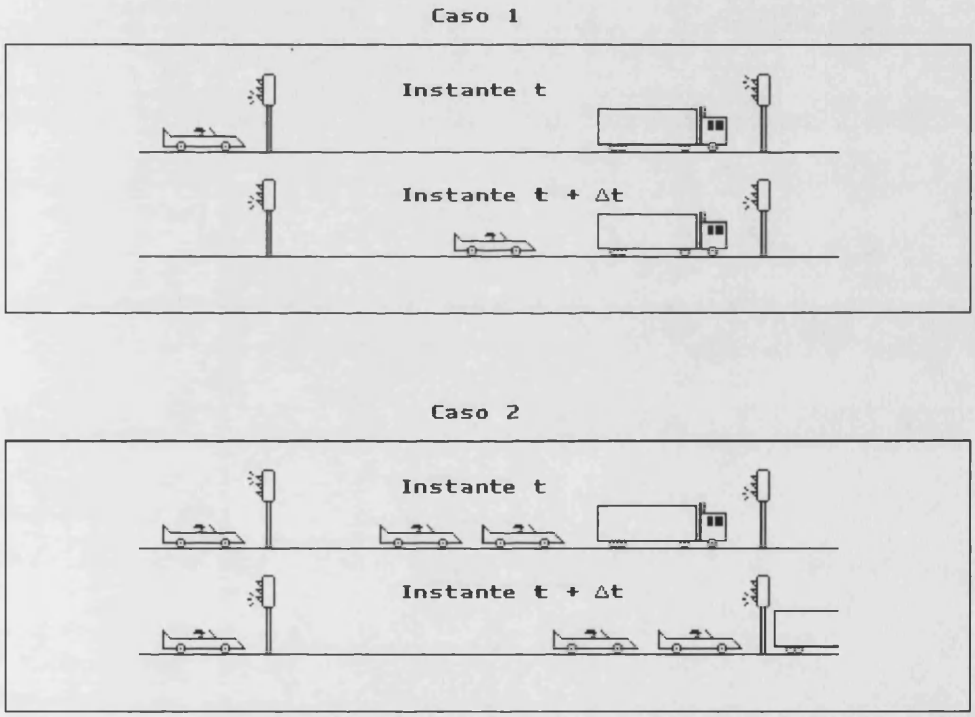

Figura G.1

En algunas salas de control, sobre todo en grandes ciudades, la actuación de estos sistemas se complementa con la supervisión permanente por parte de expertos humanos,

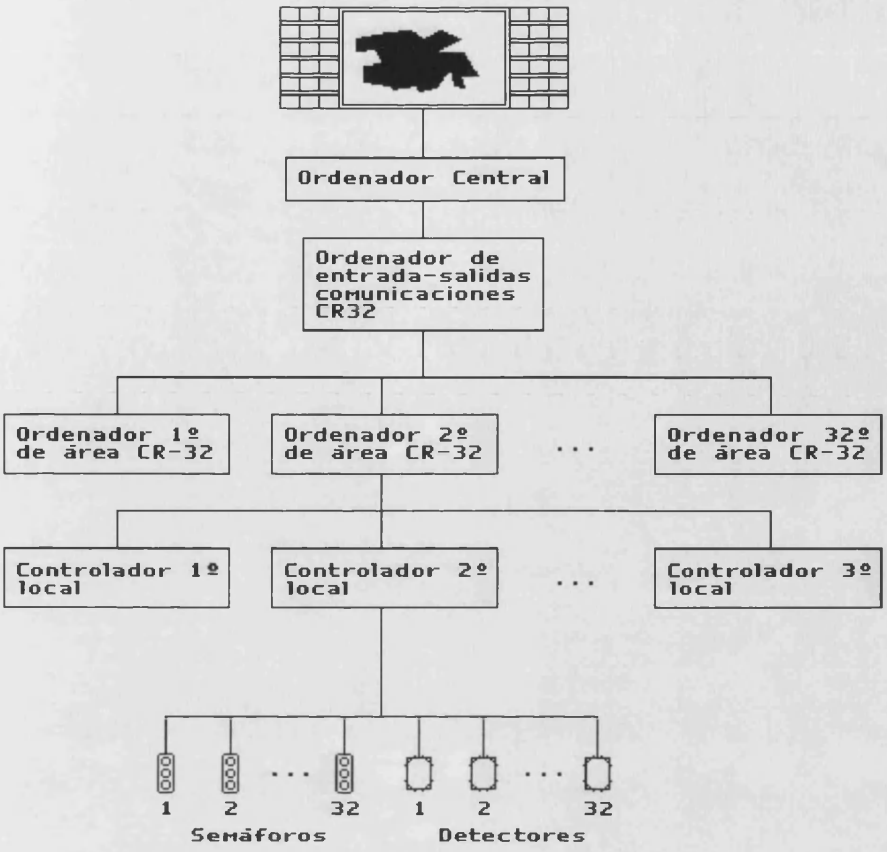

Figura G.2

que pasan a actuar cuando el sistema se muestra ineficiente; para que éstos actúen, la sala de control dispone además de un equipo de pantallas de TV con las que se pueden seleccionar imágenes provenientes de cualquiera de las cámaras de vídeo que se encuentran estratégicamente distribuídas a lo largo y ancho de la red a controlar.

Con la finalidad de introducir dos conceptos fundamentales en el control de tráfico urbano como son el ciclo y el reparto, vamos a proceder a su definición y a la explicación de cómo se manejan. Utilizando un sistema centralizado de control de tráfico, podemos actuar sobre cada intersección de dos formas básicas:

- Cambiando el ciclo.

- Cambiando el reparto.

El ciclo es el número de segundos que un semáforo de un cruce tarda en repetir su comportamiento.

El reparto es la distribución del tiempo de rojo, verde y ámbar de cada semáforo (incluyendo los de los peatones). La suma del tiempo de rojo, verde y ámbar de un semáforo, da su ciclo. Cuando el operador impone un reparto en un cruce, se dice que el cruce "está forzado" o que "tiene una forzadura".

El ciclo puede tomar cualquier valor, pero en la práctica está entre 60" y 120". Ciclos más cortos dan muy poco tiempo de verde, y consiguientemente dan muy poco paso a los vehículos (más todavía teniendo en cuenta que cada uno de los vehículos emplea un cierto tiempo del verde en empezar a moverse); ciclos más largos dan un excesivo tiempo en rojo (especialmente a los peatones), lo que puede provocar una desobediencia civil generalizada.

Dentro de un ciclo pueden seleccionarse un gran número de repartos, aunque en la práctica sólo se utilizan unos pocos (p.e. en Valencia se utilizan ocho repartos para cada cruce y en algunos cruces, entre esos ocho hay repetidos). La forma de asignar los repartos es compleja, por que se tiene que tener en cuenta no sólo que dos direcciones de tráfico que se intersectan no pueden tener verde al mismo tiempo, sino también que un semáforo destinado a vehículos se tiene que poner verde un cierto tiempo después de que haya dejado de estar verde en el cruce el de los peatones cuyo paso se encuentre intersectado por la dirección de los vehículos.

Este "cierto tiempo"es variable, dependiendo de la anchura de la calle. Por esta complejidad a la hora de asignar los repartos, es por lo que están precalculados y almacenados en un histórico.

G.1 Revisión y puesta al día de los sistemas de CTU tradicionales

Llamaremos sistemas de CTU "tradicionales" a aquellos que se basan en métodos algorítmicos. Para estudiar los más representativos, se pueden seguir dos clasificaciones:

i) La desarrollada en el proyecto UTCS del U.S. Department of Transportation [TSCC,84]:

   - Sistemas de control de primera generación (1G): realizan una optimización off-line de la señalización de la red, formando una librería de planes fijos entre los que se selecciona el adecuado.

- Sistemas de control de segunda generación (2G): realizan una optimización on-line, utilizando una predicción de tráfico basada en valores de flujo históricos y actuales.

- Sistemas de control de tercera generación (3G): realizan una optimización on-line y el control es sensible a las condiciones actuales de tráfico.

ii) La otra posible clasificación es la que hace el British Transportation and Road Research Laboratory:

- Sistemas que funcionan con planes fijos, basados en datos históricos y calculados off-line (se corresponde con 1G).

- Sistemas coordinados, mediante un plan fijo básico, y con respuesta al tráfico local. (Equivale a 1G con un control de intersecciones críticas).

- Sistemas que emplean estrategias de respuesta a las condiciones actuales del tráfico en las cuales la señalización se calcula on-line (abarca 2G y 3G).

Hilando un poco más fino se puede hacer la siguiente clasificación que es unión de las dos anteriores, y que es la que emplearemos:

- Sistemas de control de primera generación.

- Sistemas de control de primera generación con control de intersecciones críticas.

- Sistemas de control de segunda generación.

- Sistemas de control de tercera generación.

Con el fin de fijar la nomenclatura, es importante distinguir entre métodos (algorítmicos) de cálculo de temporizaciones de señales de tráfico y Sistemas de CTU tradicionales. Los primeros no son sistemas de control de tráfico, sino que, como su nombre indica, son herramientas que permiten fijar (generalmente off-line) las temporizaciones de las señales de tráfico, y sus resultados son utilizados por algunos de los sistemas de CTU tradicionales.

A continuación vamos a hacer la revisión y puesta al día de los sistemas de CTU tradicionales:

- De primera generación: Son sistemas de control basados en el precálculo de las temporizaciones. Como ejemplo cabe citar el sistema empleado en Glasgow, CITRAC; para este sistema los planes fueron calculados mediante el TRANSYT-8, poniéndose en acción un plan según horario, según intervención manual de un operador o bien según las condiciones medidas cada 15 minutos.

- De primera generación con control de intersecciones críticas: Se basan en estrategias en las que cada señal puede responder individualmente al tráfico detectado en sus accesos, pero sujeto a un esquema de coordinación y ciclo básico prefijado por algún método de primera generación. Como ejemplos podemos citar FLEXIPROG y EQUISAT.

- De segunda generación: Son sistemas de control que calculan e implementan en "tiempo cercano al real" (a intervalos de unos 15 minutos) planes de tráfico, a partir de una predicción basada en medidas externas y en datos históricos. Como ejemplo tenemos el sistema ASCOT (desarrollado por el Standford Research Institute en 1977) y el RTOP (desarrollado por el metropolitan Toronto Roads and Traffic Department en 1976).

- De tercera generación: En este apartado se incluyen aquellas estrategias que realizan una optimización on-line en respuesta a la situación actual del tráfico. Las longitudes del ciclo, reparto y desfase pueden variar entre intersecciones y de ciclo en ciclo, a fin de obtener la optimización global de la red. Utilizan datos recogidos por detectores, y a partir de ellos realizan unas predicciones de los volúmenes de tráfico que llegarán a los semáforos, mediante algún modelo de flujo de tráfico. Dentro de esta categoría se encuentran PLIDENT, SCOOT [Hunt,81], [Hunt,82], [Clowes,82], [Walmsley,82], OPAC [Gartner,83], y PRODYN [Henry,83], [Henry,84].

Ante la existencia de estos sistemas surge la pregunta de si éstos son o no satisfactorios; efectivamente no lo son, como ejemplo veamos las limitaciones que tiene SCOOT, uno de los más utilizados:

- En condiciones de tráfico ligero no proporciona mejoras en relación a un sistema de intersecciones actuadas por vehículos [Bretherton,82].

- Tampoco se puede asegurar que obtenga repartos apropiados en condiciones de congestión. esto es debido a que SCOOT no tiene información más allá del detector, por tanto no detecta la presencia de colas, y si se produce un error en la modelización de éstas (p.e. por un fallo en la predicción de la descarga o por que la cola alcanza al detector y el flujo de llegada parece decrecer), el reparto que proporciona es incorrecto [Robertson,87].

- Debido a los requerimientos del SCOOT (detectores en todos los accesos, densidad de semáforos alta,...) normalmente SCOOT se aplica a subredes, trabajando en cooperación con otros sistemas de control, lo cual crea problemas de interfaz entre el área controlada por el SCOOT y las restantes áreas [Robertson,87].

La complejidad del problema del control de tráfico urbano hace muy difícil el cálculo de planes de tráfico en tiempo real para una red suficientemente grande. Si bien se han hecho tentativas en este sentido [Henry,84], las técnicas actuales se basan en la existencia de una biblioteca [Petit,82] de planes de tráfico preestablecidos, y previstos para un conjunto de situaciones más o menos típicas: horas punta, ciclos horarios,.... La adaptación a las condiciones de la circulación se efectúa por medio de algoritmos de selección (método del vector,...) o de modulación (SCOOT [Hunt,81] [Hunt,82]) de los planes de tráfico. Por este motivo, de forma un tanto esquemática, podemos clasificar en dos tipos las formas de gestionar el tráfico que se dan en las ciudades españolas y europeas (consideramos que la problemática que se da en ciudades concebidas con mentalidad moderna como la mayor parte de ciudades de EE.UU., es distinta de la que se da sobre ciudades de la vieja Europa, con crecimiento progresivo en varias capas sobre el núcleo de un casco antiguo):

a) Sistemas computerizados que gestionan el tráfico en base a una biblioteca de planes o con sistemas autoadaptativos de generación de los mismos, que controlan la red semafórica sin intervención humana sobre la actuación del sistema. En general, los planes de tráfico que se manejan trabajan con criterios de optimización. Esta optimización es tal mientras no se llegue a situaciones críticas de congestión que caen fuera del dominio en el cual pueden empezar a mostrarse ineficientes.

b) Sistemas computerizados que actúan sobre el tráfico, pero que han sido concebidos para que además un experto de la sala de control pueda actuar

(forzando repartos y ciclos, efectuando acciones sobre la demanda[1], etc.) siguiendo estrategias que obedecen a su propia experiencia y conocimiento del medio. Esta aproximación presenta el inconveniente, de que siempre debe haber un experto (en realidad un grupo de ellos), dispuesto a actuar en corto espacio de tiempo y con unas herramientas suficientes que le garanticen la eficacia de sus medidas.

G.2 Revisión y puesta al día de los sistemas de CTU no tradicionales

Una reciente búsqueda bibliográfica computerizada con las palabras clave de entrada "Sistemas basados en el conocimiento" y "Control de tráfico", mostró que más del 70% de la literatura relacionada con estos temas hacen referencia a problemas relacionados con el tráfico aéreo, cerca del 20% con el tráfico ferroviario, y sólo el 10% restante a temas relacionados con el control de tráfico urbano. Estos datos bibliométricos junto con otros informes realizados en USA y Canadá, concluyen que la aplicación de Sistemas Basados en el Conocimiento a problemas de transporte, en general son bastante escasas (1% del total de realizaciones)[2].

Vamos a describir varios de los sistemas novedosos que se están desarrollando en la actualidad, y que son aplicables al tráfico de vehículos con diferentes objetivos:

- Entre los sistemas que manejan el tráfico, en algún sentido, estudiaremos cinco, estando todos ellos construídos con sistemas basados en el conocimiento:

    - El primero, AURA, es un sistema experto que utiliza simuladores para la predicción y para la adquisición de conocimiento profundo, y su campo de aplicación es el control del tráfico de acceso a las ciudades.

    - El segundo, SAGE, es un sistema experto concebido para gestionar embotellamientos. El sistema utiliza un razonamiento temporal rudimentario, tras darse cuenta sus autores de esta necesidad después de construir un primer prototipo.

[1]Se puede restringir la demanda, haciendo que el reparto de un semáfor tenga un porcentaje muy elevado de tiempo en rojo, lo que en el argot de tráfico se conoce como "gating".
[2]Esta situación está cambiando en los últimos meses, ejemplo de ello es la reciente 10th International Workshop de Avignon [Avignon,90] y los dos proyectos DRIVE relacionados con el tema (V-1015 y V-1022) [DRIVE,90].

- El tercero, VANESA, es interesante porque en su desarrollo se ha demostrado que es posible hacer ingeniería del conocimiento en el campo del control de tráfico urbano. En la actualidad se encuentra en desarrollo en Valencia.

- El cuarto, TRALI, está concebido para actuar sobre intersecciones aisladas.

- El quinto,en el dominio del control de tráfico urbano, ha sido propuesto por M.C. Bell y P.T. Martín [Bell,90] muy recientemente en la última conferencia de IEE celebrada en Londres.

- Por otra parte estudiaremos otros dos intentos de incorporar nuevas técnicas para la mejor gestión del tráfico:

  - Un simulador de tráfico que se está desarrollando en Barcelona.

  - Un sistema de guiado de vehículos (ALI-SCOUT), integrado con el sistema de CTU.

1) **AURA**: (Accesos Urbanos Regulados Automáticamente). Es un sistema basado en ingeniería del conocimiento para el control de accesos a las grandes ciudades. Está siendo desarrollado por J. Cuena [Cuena,89] en Madrid, y utiliza simuladores para predecir y para completar el conocimiento extraído del experto humano. Sus principales funciones son:

- Una interpretación de los datos recibidos en tiempo real para obtener una detección automática de incidentes y congestiones.

- Valoración del nivel de servicio y su distribución en la autopista.

- Capacidad de predecir la evolución del estado de los accesos, para predecir situaciones futuras de congestión.

- El sistema debe ser capaz de producir propuestas de control en dos niveles:

  - Control general: Tales como carriles reversibles o mostrar itinerarios alternativos.

- Control local: Tales como control de ciclos y repartos de los semáforos situados en los accesos.

- Capacidad de explicación y justificación de respuestas, así como aclaración de detalles.

La representación del conocimiento en este sistema considera tres tipos de conocimiento diferentes:

- Conocimiento de predicción:

  Está dividido en dos bases de conocimiento:

  - Un conjunto de restricciones que describen un simulador cualitativo, y cuyas salidas son las predicciones.

  - Un conjunto de reglas para la interpretación del estado actual y del pasado próximo, cuyas salidas son las entradas para el simulador.

- Conocimiento de interpretación:

  Está también dividido en dos bases de conocimiento, y toman como entradas las salidas de conocimiento de predicción:

  - Un conjunto de reglas por sección para deducir existen incidentes en cada carril y el grado de congestión en cada carril.

  - Un conjunto de reglas para cada zona (conjunto de carriles) para deducir la evaluación del grado de congestión medio y el grado de reversibilidad.

- Conocimiento de control:

  También está dividido en dos tipos de reglas:

  - Un conjunto de reglas para deducir decisiones generales (como cambiar el sentido de un carril).

- Otro conjunto de reglas para tomar decisiones locales como cambiar ciclo y reparto en un punto.

2) **SAGE**: (systeme d'Aide a la Gestion des Emboteillages).

Está siendo desarrollado por B. Forasté y G. Scenama [Forasté,86], [Forasté,87] en el Institute de Recherche des Transports de Francia desde 1985. La primera implementación de SAGE fue hecha en SNARK1, un motor de inferencia escrito en Pascal que utiliza lógica de primer orden y no tiene en cuenta el contexto temporal del problema. Sus autores llegaron a la conclusión de que el proceso de razonamiento necesitaba tener en cuenta nuevos factores como el tiempo, y reescribieron el nuevo SAGE en el lenguaje declarativo SNARK2 [Laurière,86], y ya se incluye un simple tratamiento de aspectos temporales en el proceso de razonamiento por medio de reglas que reconocen situaciones de presaturación (esto se debe a que el sistema no reacciona cuando la situación del embotellamiento aparece, sino un poco después). Así, el sistema SAGE puede evitar la formación de embotellamientos, que es algo más deseable y fácil que deshacerlos cuando se han producido. Las funciones del sistema son las siguientes:

- Observar y analizar la situación del tráfico en la red.

- Diagnosticar las causas de la congestión.

- Proponer y gestionar posibles soluciones a la congestión.

3) **VANESA**: (Valencia Area Network Expert System Aids) es un prototipo de sistema experto en control de tráfico urbano desarrollado en Valencia [Martín,88] y basado sobre reglas y lógica de primer orden, utilizando razonamiento hacia adelante. La primera versión de este sistema estuvo escrita en LISP y su objetivo fue, utilizando las lecturas de los detectores, dar avisos que permitan evitar situaciones críticas (especialmente congestión y ocupación de intersecciones por largas colas de vehículos). En el capítulo 8 estudiaremos su evolución, situación actual y planteamiento dentro del marco de CP. Es un ejemplo de que es posible realizar una ingeniería del conocimiento a partir de un equipo de ingenieros. Este sistema considera la problemática temporal del sistema y no ha trabajado todavía on-line.

4) **TRALI**: Es un sistema experto concebido para ayudar a la regulación de intersecciones aisladas. Ha sido desarrollado por Carlos Zozaya-Gorostiza y Chris

Hendrickson [Zozaya-Gorostiza,87], [Hendrickson,87] en la Carnegie-Mellon University en 1985. Está escrito en el entorno de sistemas expertos OPS5 [Browmston,85], con las rutinas de análisis escritas como funciones LISP, y sus principales funciones son:

- Identificación de los pares de flujos que provocan un conflicto en la intersección.

- Propuesta de una distribución de fases.

- Determinación de la longitud del ciclo y reparto de verde óptimos.

- Cálculo de índices de funcionamiento: evalúa la demora media por carril, longitud media de colas y demora total por ciclo mediante un proceso algorítmico.

- Una vez evaluada la distribución propuesta, el ingeniero puede modificar datos y resultados, y repetir el conjunto de tareas hasta que se considere aceptable la solución identificada.

5) **Sistema experto de control de tráfico**: M.C. Bell y P.T. Martín han propuesto recientemente [Bell,90], en paralelo con la finalización de esta memoria, los requerimientos para una integración completa de los sistemas de control de señales (estén aisladas o agrupadas en grupos coordinados, sean activadas por vehículos o no) con los sistemas de monitorización de tráfico. Esta integración sus autores proponen que se realice con un sistema experto con los siguientes requerimientos:

- Tienen que incorporar una completa descripción de las señales, y otros dispositivos para el control de tráfico instalados sobre la red de tráfico de la ciudad y de sus alrededores.

- Un conjunto comprensible de criterios y reglas que definen las restricciones del tráfico.

- Un análisis y monitorización contínuos de los datos recogidos del tráfico, no sólo dentro de la ciudad, sino en un área más amplia.

- Análisis estadístico de los datos "off-line" para construir un modelo probabilístico de la situación "normal" del tráfico y para construir un histórico de los acontecimiento que se presentan.

- Definición de umbrales de los diversos parámetros de tráfico para presentar "on-line" situaciones de congestión y los cambios significativos en las condiciones de tráfico.

- Modelización de un interface para predecir rutas alternativas con capacidad disponible.

- Calibración de las herramientas de modelización utilizando los datos de los detectores.

- Una biblioteca de planes de tráfico estratégicos que puedan ser activados cuando se requiera.

- Desarrollo de procedimientos para actualizar "on-line" datos históricos.

- Definición de la base de reglas para tomar decisiones, diagnosticar y explicar los problemas de tráfico, así como para seleccionar de la biblioteca el plan adecuado para remediarlos.

- Un método para evaluar la bondad del sistema.

- Integración de todas las estrategias de control de señales.

- Un interfaz con sistemas de autoguiado.

El sistema experto planteado por Bell y Martin empleará los sistemas de detección de tráfico de dos formas:

- "on-line" para actividades como monitorización, evaluación del sistema, detección de congestión, y asesoramiento de rutas con capacidad libre.

- "off-line" para calibración de modelos de tráfico, predicción de rutas de tráfico y para obtener estrategias de control ante situaciones conflictivas.

6) **Simuladores de Tráfico Urbano**: El Departamento de Investigación Operativa y Estadística de la Facultad de Informática de Barcelona, está desarrollando un proyecto para el Ayuntamiento de Barcelona, consistente en la construcción de un simulador de tráfico urbano [Barceló,88a], [Barceló,88b] teniendo entre otras las siguientes características:

- Capacidad para generar automáticamente los datos del simulador.

- Capacidad para manipular interactivamente los datos.

- Capacidad para generar automáticamente el simulador de la zona de la red de tráfico a estudiar.

- Capacidad interactiva de ejecución y diseño de experiencias de simulación.

El simulador se encuentra ya construído, y se investiga actualmente en una nueva versión del mismo que incorpora un sistema de Base de Conocimiento mejorado y dos nuevas características [Barceló,91]:

- Módulo de planes de control de tráfico.

- Módulo de diagnóstico y evaluación.

Estas características se incorporan para cubrir los objetivos básicos del simulador, entre los que podemos destacar los siguientes:

- Ser una herramienta de análisis del comportamiento del tráfico.

- Ayudar al diseño y cálculo de planes de control en tráfico:

    - Ayudando a la definición de estrategias.
    - Reduciendo los tiempos de ajuste de parámetros.
    - Agilizando la evaluación de planes alternativos.

- Ser un banco de pruebas de planes de control.

- Predecir el efecto de las estrategias de control.

- Permitir la evaluación del impacto de las medidas de la ingeniería de tráfico.

7) **ALI-SCOUT**: Es un proyecto basado en la intercomunicación baliza/coche desarrollado por SIEMENS, que tiene como objetivo el desarrollo de sistemas de autoguiado. En este momento su integración con los Sistemas de Control de Tráfico está en pleno desarrollo[3] y por ello se puede considerar de una nueva generación. Este sistema de autoguiado no sólo aconseja qué itinerario debe seguirse para llegar al punto de destino (lo cual es útil cuando no se conoce el terreno), si no que además aconseja el itinerario mejor en cada momento, considerando la evolución temporal del tráfico (tanto la situación actual como las previsiones de tráfico en todo el recorrido a seguir); esta componente temporal modifica el problema transformándolo en uno mucho más complejo en cuanto que:

- Lo convierte en un problema "tridimensional", como puede verse en la figura i.5.

- Presupone un constante análisis del tráfico, y un constante cálculo de itinerarios óptimos.

Para alcanzar sus objetivos, se propone instalar en los semáforos unas balizas que permitan la comunicación, mediante infrarrojos, entre los automóviles que lo soliciten y un potente ordenador que centraliza los datos y hace los cálculos pertinentes. El conductor recibe la información del ordenador en el cuadro de instrumentos o en una pantalla que tiene instalada en el salpicadero.

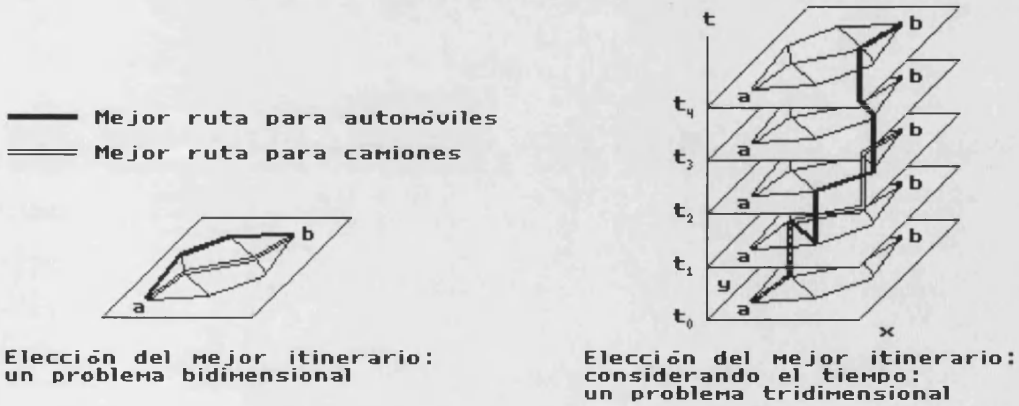

Figura G.3

[3]Proyecto DRIVE V-1011 [DRIVE,90].

Como se puede observar, pocas de las anteriores aproximaciones tienen como objetivo el control "on-line" del tráfico urbano; por ejemplo, SAGE, incide sobre él pero no en la parte de control a fin de administrar los recursos disponibles de la mejor forma posible, sino que ejerce el control cuando se han de deshacer embotellamientos(aunque en los últimos estudios [Schnetzler,90] se plantean el problema del control). En la quinta, se plantea el objetivo del control, en base a la activación de planes de tráfico precalculados, pero aún se encuentra en una fase primaria. La última, ALI-SCOUT, permite dar una información muy valiosa al conductor, para que no se mueva "a ciegas" por la ciudad, lo que debería complementarse con mecanismos que permitan controlar, además de informar. Aquí surge un nuevo campo de trabajo como es la cooperación entre sistemas de autoguiado y sistemas de CTU basados en el conocimiento, y que parece abordable sobre la base de la información de Origen/Destino y tiempos de viaje que proporcionan los sistemas de autoguiado.

Como se puede observar, se ha activado un intenso campo de trabajo en los últimos dos-tres años alrededor del problema del control del tráfico, tanto urbano como interurbano. Un ejemplo de esta situación es el programa DRIVE de la CEE, que pretende la utilización de las nuevas tecnologías de la Información y de las Comunicaciones a los problemas del tráfico y de los transportes; entre estas nuevas tecnologías, la IA ha sido objeto de especial dedicación [DRIVE,90]. En este sentido, los tres últimos capítulos de esta memoria (en especial el último) se dedican a particularizar lo desarrollado en los capítulos anteriores al caso del Control del tráfico urbano.

Anexo H

EL CALCULO DE ACONTECIMIENTOS PARA CAMBIO CONTINUO

H.1 INTRODUCCION

El Cálculo de Acontecimientos de Kowalski y Sergot [Kowalski,86] es un formalismo de representación de un mundo cambiante mediante acontecimientos y propiedades iniciadas por los mismos. Una propiedad que ha sido iniciada continúa válida por defecto hasta que suceda algún acontecimiento que pueda terminarla (asunción de persistencia temporal). Pueden representarse acontecimientos simultáneos y parcialmente ordenados, pero sin embargo, todos los cambios son discretos. Este formalismo necesita ser extendido para poder representar cantidades de cambio continuo, tales como la altura de un objeto que cae o el nivel de fluido en una cuba que se está llenando. El Cálculo de Acontecimientos para Cambio Continuo de Shanahan presenta esta extensión [Shanahan,89].

En él se presenta una versión simplificada del Cálculo de Acontecimientos de Kowalski, e introduce una extensión para manejar el cambio continuo. Se introduce la idea de *autoterminación* (Si un período de cambio continuo dura lo bastante, él mismo puede ser capaz de generar el acontecimiento que lo termine, como por ejemplo cuando una cuba que se está llenando rebosa, o cuando un objeto que cae alcanza el suelo). El Cálculo de Acontecimientos para Cambio Continuo es aplicado entonces a varios ejemplos de sistemas con cambio continuo compuestos de cubas que se llenan o vacían, componiendo lo que Shanahan denomina el Problema del Sistema de Cubas.

H.2 LOS AXIOMAS DEL CALCULO DE ACONTECIMIENTOS PARA CAMBIO CONTINUO. CASO CUANTITATIVO

Para representar el concepto de cambio continuo de una cantidad cualquiera, definimos una propiedad especial que contenga el valor de es cantidad en cualquier momento del tiempo durante el que sea válida. Por ejemplo, en el caso de una cuba llena de fluido cuyo nivel varía continuamente, nivel(N) representa la propiedad que indica que el nivel del fluido es N. Entonces, para preguntar por el valor en un tiempo T2 dado de una propiedad P que representa una cantidad cualquiera que cambia continuamente, utilizaremos el siguiente predicado general:

cierto_en(P,T2) si
    sucede(E) y tiempo(E,T1) y
    T1 < T2 y inicia(E,Q) y
    no interrumpido(T1,Q,T2) y trayectoria(Q,T1,P,T2).

Donde Q es otra propiedad que representa la naturaleza del proceso de cambio continuo que está afectando a la cantidad descrita por P. Aquí se introduce la idea de trayectoria de una cantidad continuamente variable, que es un camino dibujado contra el tiempo a través del correspondiente espacio de cantidades. Por ejemplo, el nivel de fluido en una cuba (**nivel(N)**), aumenta linealmente a partir del comienzo de un periodo de llenado (**llenado**). Entonces la fórmula trayectoria(Q,T1,P,T2) representa que la propiedad P (nivel(N)) es cierta en el tiempo T2 sobre la trayectoria del período de cambio continuo Q (llenado) que comienza en el tiempo T1, como puede verse a continuación:

trayectoria(llenado,T1,nivel(N2),T2) if
    cierto_en(nivel(N1),T1) y L2 = L1 + 2*(T2-T1).

La idea de autoterminación puede introducirse añadiendo los axiomas mostrados a continuación al Cálculo de Acontecimientos. En ella, el acontecimiento fin(E,Q) es definido para denotar el acontecimiento que terminará el período de cambio continuo que fue iniciado por el acontecimiento E. En el ejemplo de la cuba, la propiedad **llenado** autoterminará si la cuba llega a rebosar, hecho que indicaríamos como fin(E,llenado) en lugar de rebosa(E). Los axiomas de autoterminación son:

sucede(fin(E,Q)) si
    sucede(E) y tiempo(E,T1) y inicia(E,Q) y
    autotermina(P,Q) y trayectoria(Q,T1,P,T2) y

no interrumpido(T1,Q,T2).

tiempo(fin(E,Q),T2) si
sucede(E) y tiempo(E,T1) y inicia(E,Q) y
autotermina(P,Q) y trayectoria(Q,T1,P,T2) y
no interrumpido(T1,Q,T2).

termina(fin(E,Q),Q).

La fórmula autotermina(P,Q) representa el hecho de que el período de cambio continuo Q es terminado si la propiedad P perdura; para el caso de la cuba:

autotermina(nivel(L),llenado) si rebosa(L).

H.3 LOS AXIOMAS DEL CALCULO DE ACONTECIMIENTOS PARA CAMBIO CONTINUO. CASO CUALITATIVO

A veces puede ocurrir que las ecuaciones cuantitativas no sean conocidas, pero que aún dispongamos de información suficiente como para deducir cosas útiles, por ejemplo, el flujo de llenado puede variar erráticamente, y no linealmente, pero esto no altera el hecho de que la cuba acabará rebosando si no se termina la propiedad **llenado** de otro modo. Estos hechos pueden axiomatizarse sustituyendo la definición cuantitativa de trayectoria por una cualitativa.

Un espacio de cantidades cualitativo puede ser representado mediante la utilización de un conjunto ordenado de valores hito [Kuipers,86], compuesto por todos aquellos valores que sean importantes por alguna razón. La cantidad que varía continuamente puede valer entonces uno de ellos o estar entre dos valores adyacentes. En el ejemplo mostrado, los únicos puntos de interés son el fondo de la cuba y su parte superior. Escribiremos sigue(P1,P2,Q) para expresar que P1 y P2 son valores adyacentes en la trayectoria del período de cambio continuo Q. También escribiremos en(P,Q) para expresar que la propiedad P está en la trayectoria del período de cambio continuo Q, por lo que tendremos para nuestro ejemplo:

en(nivel(fondo),llenado).
en(nivel(borde_superior),llenado).

sigue(nivel(fondo),nivel(borde_superior),llenado).

rebosa(borde_superior).

Los siguientes axiomas definen entonces la versión cualitativa del predicado trayectoria, a partir de los hechos anteriores:

trayectoria(Q,T,P,tiempo_de(P,Q,T)) si en(P,Q).

trayectoria(Q,T,entre(P1,P2),T2) si
    tiempo_de(P1,Q,T) < T2 < tiempo_de(P2,Q,T) y
    sigue(P2,P1,Q).

tiempo_de(P,Q,T) > T si en(P,Q).

tiempo_de(P2,Q,T) > tiempo_de(P1,Q,T) si sigue(P2,P1,Q).

Referencias

[Ambrosino,88] **G.Ambrosino, G.Boccassi, M.Boero and A.Valcada:** "Knowledge-based techniques intraffic control systems", Proc. Colloquium on Applications of Expert Systems in Road Transportation, I.E.E., London, 1988, pp.6/1,6/4.

[Avignon,90] "Specialized Conference Artificial Intelligence & Transportation", 10th International Workshop Expert Systems & Their Applications, EC2, Avignon, 1990.

[Bahamonde,88] **A. Bahamonde y J. Secundino López:** "Diseño y arquitectura de un sistema inteligente para el control del tráfico urbano", Tecno-Traffic-88, 1988, Madrid.

[Barceló,88a] **J. Barceló:** "Inteligencia Artificial y simulación: aplicaciones a la simulación de tráfico urbano", E. y S. Municipales, Sep.-Oct. 1988, pp. 9-18.

[Barceló,88b] **J. Barceló:** "Inteligencia Artificial y Simulación: Aplicaciones a la Simulación de Tráfico Urbano", Ruta Automática e Instrumentaciones 81, pp. 239-249, 1988.

[Barceló,91] **J.Barceló:** "Simulation of Urban Traffic: Software Environments", Concise Encyclopedia of Traffic & Transportation Systems, PapaGeorgiou, Pergamon press 1991, pp.483-491.

[Barr,89] **A. Barr, P.R. Cohen and E.A. Feigenbaum eds.:** "The handbook of artificial intelligence", Addison-Wesley, 1989.

[Bell,90] **M.C. Bell and P.T. Martin:** "Use of Detector Data for Traffic Control Strategies", Third International Conference on Road and Traffic Control. 1-3 May 1990. Conference Publication Nr. 320, IEE 1990, pp. 86-89.

[Bonissone,85] **Bonissone, P.P., Valavani, K.P.:** "A comparative study of different approaches to qualitative phisics theories" en IEEE, pg 236, 1985.

[Bretherton,82] **R.D. Bretherton and G.I. Rai:** "The use of SCOOT in low flow conditions". Traffic Engineering & Control, 1982, pp. 574-576.

[Bristol,77] **E.H. Bristol:** "Pattern recognition: an alternative to parameter identification in adaptative control", Automatica, vol.13, 1977, pp. 197-202.

[Browmston,85] **L. Browmston, R. Farrell, E. Kant and N. Martin:** "Programming Expert Systems in OPS5. An introduction to Rule-Based Programming", Addison-Wesley Publishing Co. Inc, Reading, Mass, 1985.

[Cain,91] **Cain D. et al:** "PLEXSYS: A specialized knowledge-based tool in perspective" en Preprints of the European Conference on Industrial Applications of Knowledge-Based Diagnosis, CISE S.p.A., Milán, Italia, 17-10-1991.

[Chandrasekaran,82] **Chandrasekaran, B., Mittal,S.:** "Deep versus compiled knowledge approaches to diagnostic problem-solving" en Proceedings of the National Conference of Artificial Intelligence, AAAI-82, Pittsburgh, PA, USA, 1982.

[Chandrasekaran,87] **Chandrasekaran B.:** "Towards a functional architecture for intelligence based on generic information processing tasks" en Proceedings of the 10th International Joint Conference on Artificial Intelligence, IJCAI-87, pags. 1183-1192, Morgan Kaufmann Publishers, Milán, Italia, 1987.

[Clowes,82] **D.J. Clowes:** "Real-time area traffic control- the user's viewpoint", Traffic Engineering & Control, 1982, pp. 194-196.

[Cuena,83] **J.Cuena:** "The use of simulation models and human advice to build an expert system for the defense and control of river floods". Proc. IJCAI-83. Kalsruhe 1983.

[Cuena,87] **J.Cuena:** "Sistemas Expertos para ingeniería y gestión". En: Inteligencia Artificial. Conceptos, Técnicas y Aplicaciones, Ed. Marcombo, pp.101-106, 1987.

[Cuena,89] **J.Cuena:** "AURA: A Second Generation Expert System for Traffic Control in Urban Motorways", Proc. 9th International Workshop on Expert Systems with Applications, Avignon 1989.

[Cuena,89] **J.Cuena:** "El Sistema Experto AURA para Control de Trafico en Accesos Urbanos". Jose Cuena. Civil Engineering Expert Systems. Civil Engineering European Cursus. COMETT Programme. February 6-10th 1989. Madrid.

[Cuena,91] **J.Cuena, A. Salmerón:** "Arquitecturas de segunda generación para el diagnóstico profundo en instalaciones industriales" en las Actas de la IV reunión Técnica de la Asociación Española para la Inteligencia Artificial, AEPIA-91, Madrid, 23-10-1991.

[De Kleer,83] **J. De Kleer, J.S. Brown:** "Assumption and Ambiguities in Mechanistic Mental Models", in D.Genter and A.S.Stevens (eds.) Mental Models, Erlbaum, Hillsdale,NJ (1983).

[De Kleer,84] **De Kleer, J. and Brown, J.S.:** "A Qualitative Physics based on Confluences", Artificial Intelligence 24 (1984) pp. 7-83.

[De Kleer,87] **De Kleer, J. and Williams, B.C.:** "Diagnosing multiple faults", en Artificial Intelligence 32, pags. 97-130. Elsevier Science Publishers B.V. North-Holland, 1987.

[DRIVE,90] **Commision of the European Communities:** "R+D Advanced Road Transport Telematics in Europe. DRIVE'90", DG XIII, March 1990.

[EQUATOR,89a] **ETRA:** "Catalogue of Test Cases". EQUATOR ESPRIT II project, D113-2, August 1989.

[EQUATOR,89b] **ETRA:** "Detailed Domain Analysis". EQUATOR ESPRIT II project, D113-1, August 1989.

[FCSL,90] **Ferranti:** "General Representation Formalism". FCSL. Deliverable of EQUATOR ESPRIT II Project. February 1990.

[Forasté,86a] **B. Forasté et G. Scemana:** "Surveillance and congested traffic control in Paris by expert system", 2nd International Conference on Road Traffic Control, April 1986, London, pp. 91-94.

[Forasté,86b] **B. Forasté et G.Scemana:** "Système Expert Temps Reel D'Aide a la Gestion du Trafic", 6th International Workshop on Expert Systems and their Applications. Avignon. France. 1986. 28-30 April. Vol 2.

[Forasté,87] **B. Forasté et G. Scemana:** "Intelligent Traffic surveillance system", 5th annual meeting of Institution of Transportation Engineers, New York, 1987.

[Forbus,83] **Forbus K.D.:** "Qualitative Process Theory", MIT Report, AI, Memo No.664A (Revised May 1983), Cambridge, MA, USA, 1983.

[Forbus,84] **Forbus K.D.:** "Qualitative Process Theory", Artificial Intelligence 24 (1984) pp. 85-168.

[Gartner,83] **N.H. Gartner:** "A demand-responsive strategy for traffic signal control", Transportation Research Record 906, 1983, pp. 75-81.

[Hendrickson,87] **C. Hendrickson, C. Zozaya-Gorostiza and S. McNeill:** "A knowledge based expert system architecture for computer aided analisys and design of intersections", Proc. 10th International Symposium on Transport and Traffic Flow Theory, Boston, MA, 1987, pp. 1-18.

[Henry,83] **J.J. Henry, J.L. Farges and J. Tuffal:** "The PRODYN real time traffic algorithm", IFAC Control in Transportation Systems. Baden-Baden, 1983, pp. 305-310.

[Henry,84] **J.J. Henry and J.L. Farges:** "Commande des feux en temps réel PRODYN", T.E.C. n° 64, Mai-Juin 1984.

[Hetthessy,83] **J. Hetthessy, L.Kaviczky and Cs Banyasz:** "On a class of adaptative PID regulators", IFAC workshop on Adaptative Systems in Control and Signal Processing", S. Francisco, 1983.

[Hunt,81] **P.B. Hunt, D.I. Robertson, R.D. Bretherton and R.I. Winton:** "SCOOT: a traffic responsive method of co-

ordinating signals", TRRL Report LR1014, Transport and Road Research Laboratory, Crowthorne, 1981.

[Hunt,82] **P.B. Hunt, D.I. Robertson, R.D. Bretherton and R.I. Winton:** "The SCOOT on-line traffic signal optimization technique", Traffic Engineering & Control, 1982, pp. 190-193.

[Keviczky,85] **L. Keviczky, I. Vajk and J. Hetthessy:** "Intellicon: an industrial multiloop adaptative regulator", IFAC Symposium on Identification and System Parameter Estimation", York, 1985.

[Kowalski,86] **Kowalski, R.A. and Sergot, M.J.:** "A logic-based calculus of events", New Generation Computing, Ohmaha ltd and Springer Verlag, 1986, pp.67-96.

[Kuipers,86] **Kuipers B.:** "Qualitative Simulation", Artificial Intelligence 29 (1986) pp. 289-338.

[Laffey,88] **Th.J.Laffey, P.A.Cox,J.L.Schmidt, S.M. Kao & Jackson Y.Read:** "Real Time Knowledge Based Systems". AI Magazine, Spring 1988, pp.27-45.

[Laurière,86] **J.L. Laurière:** "Un langage déclaratif: SNARK", Technique et Science Informatiques, vol. 5, n° 3, 1986, pp. 141-172.

[Leitch, 90] **Leitch R.R., Wiegand M.E: and Quek H.C.:** "Coping with Complexity in Physical System Modelling", AICOM Vol.3 No. 2, June 1990, pp. 48-57.

[Leitch,91] **Leitch R. et al:** "ARTIST: A methodological approach to specifying model based diagnostic systems" en Preprints of the European Conference on Industrial Applications of Knowledge-Based Diagnosis, CISE.p.A., Milán, Italia, 17-10-1991.

[Leutzbach,88] **Leutzbach W.:** "Introduction to the Theory of the Traffic Control", Springer Verlag, Berlin, 1988.

[Lim,85] **K.W. Lim and C.C. Hang:** "Performance studies of an adaptative PID controlles", 4th Yale Workshop on Adaptative Control, 1985.

[Lukas,89] **M.P. Lukas, R.A. Oyen, M.A. Keyes and A. Kaya:** "Evolution of Experts Systems for Real Time Process Management: A Case Study on Motor Control", Artificial Intelligence in Real-Time Control, Shenyang, PCR, 1989, pp.79-84.

[Maim,90] **Maim E.:** "Common Representation Language". SYSECA. Second Review of EQUATOR Project. March 1990.

[Martín,88] **G. Martín, F. Toledo, V. Sanchez Barcaiztegui , J.C. de la Rosa:** "VANESA: una aplicación a los sistemas basados en el conocimiento a la gestión del tráfico urbano de Valencia. Informe Preliminar". Tecnotrafic-88 (8,9,10 Mayo 1988).

[Martín,88] **G.Martín, F.Toledo, V. Sánchez Barcaiztegui, J.C. de la Rosa:** "Vanesa: una aplicación de los sistema basados en el conocimiento a la gestión del tráfico urbano de Valencia". Proc. TecnoTraffic-88, 1988, Madrid.

[Michalopoulos,86] **Panos G. Michalopoulos:** "Macroscopic Models of Traffic Flow for Simulation and Control". Symposium Sistemas de Control de Tráfico. Proceedings Traffic Control Systems. May 13th-15th 1986. ICDT, Generalitat de Catalunya, Barcelona.

[Montanari,89] **Montanari A., Ratto E. and Tomada L.:** "Event Recogniser. Draft for WS2.2". CISE. In EQUATOR ESPRIT II project.

[Moreno,90] **Moreno S., Toledo F., Rosich F. and Martin G.:** "Qualitative Simulation for Temporal Reasoning in Urban Traffic Control", Proc. 10th International Workshop on Expert Systems and their Applications, Avignon 1990, pp. 97-111.

[Moreno,90] **S. Moreno, F.Toledo and G.Martín:** "A Qualitative Model for UTC". ETRA. Presented at the Second Review of EQUATOR ESPRIT II Project. March 1990.

[Radke,84] **F. Radke and R. Isermann:** "A parameter-adaptative PID-controller with stepwise parameter optimisation", IFAC 9th World Congress, Budapeswt, Hungary, 1984.

[Repsol,92] **REPSOL-Instituto de Ingeniería del Conocimiento:** "La Inteligencia Artificial y el Control en tiempo real", UIMP, Madrid 92, pp. 15-55.

[Robertson,87] **G.D. Robertson:** "Handling congestion with SCOOT", Traffic Engineering & Control, 1987, pp. 228-232.

[Salmerón,92] **Antonio Salmerón Cabañas:** "Tesis Doctoral", Universidad Politécnica de Madrid, Facultad de Informática, 1992, pp 23-28.

[Sauthier,92] **Eric Sauthier and Boi Faltings:** "Model-Based Traffic Control", International Congress of Artificial Intelligence applied to Transportation, Buenaventura, California, 1992, pp.133-152..

[Shanahan,89] **Murray Shanahan:** "Representing Continuous Change in The Event Calculus". Imperial College. Department of Computing. London. June 1989.

[Shapiro,92] **Shapiro:** "Enciclopedia of Artificial Intelligence", Second Edition, Ed. John Wiley & Sons, Inc, Printed in USA, 1992.

[Sibille,87] **E. Sibille:** "Systemes Experts et Temps Reel" Genie Logiciel, n°8, 1987, pp.62-70.

[Smith,85] **C.A. Smith and A.B. Corripio:** "Principles and practice of Automatic Process Control", Ed. J. Wiley&sons, 1985.

[Sugimoto,92] **Tsutomu Sugimoto, Satoru Nonaka and Hiroki Ooama:** "Traffic prediction and qualitative reasoning", International Congress of Artificial Intelligence applied to Transportation, Buenaventura, California, 1992, pp.3-18.

[SYSECA,90] **E.Maim:** "Common Representation Language". SYSECA. Second Review of EQUATOR ESPRIT II Project. March 1990.

[Tebbs,77] **D.Tebbs and G.Collins:** "Real Time Systems", Mc GrawHill 1977.

[Toledo,90] **Toledo F., Moreno S., Rosich F. and Martin G.:** "Análisis de dos modelos basados en confluencias para el razonamiento cualitativo en una red de tráfico urbano", Procc. IBERAMIA 1990, Morelia, Méjico, pp. 549-568.

[Toledo,90] **Toledo F.:** "Aspectos Temporales y de Incertidumbre en Sistemas Basados en el Conocimiento en Control de Procesos: La Gestión de una Red de Tráfico Urbano", Tesis Doctoral, Junio 1990.

[TSCC,84] **Traffic Signals Systems Commitee:** "Developments in traffic signal systems", Transportation research Circular, n° 282, 1984.

[Turner,86] **R.Turner:** "Logiques pour l'intelligence artificielle", ed. Masson, Paris, 1986.

[Walmsley,82] **J.Walmsley:** "The practical implementation of SCOOT traffic control system", Traffic Engineering & Control, 1982, pp. 196-199.

[Widman,89] **Widman L.E., Loparo K.A., Nielsen N.R.:** "Artificial Intelligence, Simulation, and Modelling", Ed. John Wiley & Sons, USA, 1989, pg 75-76.

[Wild,92] **Wild B.:** " SAPPORO: Towards an Intelligent Integrated Traffic Management System", International Congress of Artificial Intelligence applied to Transportation, Buenaventura, California, 1992, pp. 19-38.

[Williams,84] **Williams B.C.:** "Qualitative Analysis of MOS Circuits", Artificial Intelligence 24 (1984) pp. 281-346.

[Winston,87] **Winston, P.H.:** "The commercial debut of artificial intelligence" en Applications of expert systems, edición al cuidado de Quinlan, J.R., Addison Wesley, 1987.

[Young,87] **S.J. Young:** "Real Time Languages: Design and Development": Ellis Horwood Limited Eds. 1987.

[Zozaya-Gorostiza,87] **C. Zozaya-Gorostiza and C. Hendrickson:** "Expert system for traffic signal setting assistance", Journal of Transportation Engineering, vol. 113, n° 2, 1987, pp. 108-126.

UNIVERSITAT DE VALÈNCIA — FACULTAD DE CIÈNCIES FÍSIQUES

Reunit el Tribunal que subscriu en el dia de la data, acordà d'atorgar, per unanimitat, a la Tesi Doctoral d'En/ Na/ N' SALVADOR MORENO PICOT la qualificació d APTO CUM LAUDE

València a 6 d MAYO de 1993

El Secretari,

El President

UNIVERSITAT DE VALÈNCIA
BURJASSOT
FACULTAT DE CIÈNCIES FÍSIQUES

UNIVERSITAT DE VALÈNCIA
BIBLIOTECA
DE CIÈNCIES