Análisis de Sistemas Críticos en Teoría de Tipos

Carlos Daniel Luna
Instituto de Computación, Facultad de Ingeniería, Universidad de la República
Julio Herrera y Reissig 565 - Piso 5, (cp:11300), Montevideo, Uruguay
TE: (+598) (2) 7114244 (int. 115). Fax: (+598) (2) 7110469
cluna@fing.edu.uy

Resumen

Para el análisis de sistemas reactivos y de tiempo real se destacan dos enfoques formales: la verificación de modelos y el análisis deductivo basado en asistentes de pruebas. El primero se caracteriza por ser completamente automatizable pero presenta dificultades al tratar sistemas con un gran número de estados o que tienen parámetros no acotados. El segundo permite tratar con sistemas arbitrarios pero requiere la interacción del usuario. Este trabajo presenta formalizaciones en teoría de tipos de grafos temporizados para modelar sistemas reactivos y de tiempo real, y formalizaciones de las lógicas CTL y TCTL para razonar sobre estas clases de sistemas críticos. Asimismo, el artículo explora una metodología que permite compatibilizar el uso de un verificador de modelos como Kronos y el asistente de pruebas Coq en el análisis de sistemas reactivos y de tiempo real.

Palabras claves: Especificación y Análisis de Sistemas de Tiempo Real, Autómatas (Grafos) Temporizados, Lógicas TCTL y CTL, Verificación de Modelos, Teoría de Tipos y Coq, Verificación-Demostración de Corrección.

1. INTRODUCCIÓN

Cada vez son más frecuentes las aplicaciones donde el tiempo juega un rol importante. Por ejemplo en: protocolos de comunicación; controladores de robots, de procesos industriales automatizados y de dispositivos electrónicos; aplicaciones multimedia y de internet. En general éstas son aplicaciones críticas, en las cuales una falla o mal funcionamiento pueden acarrear consecuencias graves, tales como poner en juego vidas humanas y/o grandes inversiones económicas. El comportamiento de estos sistemas, llamados *sistemas de tiempo real*, no está determinado únicamente por la sucesión de acciones que se ejecutan, sino también por el momento en que las mismas ocurren y son procesadas. El tiempo de ejecución es "el" parámetro fundamental en el comportamiento de esta clase de sistemas y una gran parte, quizás la más importante, de los requerimientos de los mismos son temporales: "tal acción debe ejecutarse en un lapso de tiempo determinado" o "el tiempo transcurrido entre dos eventos o señales debe estar acotado por un valor constante", etc.

Es indiscutible hoy la influencia que tiene en la industria y en casi todos los ámbitos el uso del software. A pesar de su uso extensivo, uno de los costos más alto no se da en la producción del software, sino en la corrección de errores que son detectados posteriormente al desarrollo del sistema. En la actualidad, el método más usado para validar software es el "*testing*". No obstante, este método no garantiza la corrección del software analizado, por ser incompleto en la mayoría de los casos [15]. En aplicaciones críticas, que tratan con vidas humanas y/o grandes inversiones económicas, la *certeza de corrección* es, en general, un criterio indispensable. De un software correcto se espera que resuelva un problema determinado por una *especificación* y que exista una justificación formal –matemática– de que el programa la satisface. En los últimos años un gran esfuerzo de investigación se ha invertido en el desarrollo de métodos y herramientas para la especificación y el análisis de la corrección de sistemas de tiempo real. Sin embargo no hay un formalismo, una metodología o una herramienta claramente preferibles a otras en todas circunstancias. Para el análisis de la corrección de esta clase de sistemas dos importantes enfoques formales se destacan:

- *Verificación de corrección*. En este enfoque un sistema es considerado correcto cuando se prueba que *toda* ejecución posible satisface la especificación. Existen algunas técnicas bien conocidas que permiten recorrer de manera exhaustiva el espacio de ejecuciones posibles y herramientas que las implementan.
- *Demostración de corrección*. En este caso se construye o deriva una *prueba* matemática de que el sistema satisface su especificación. Aquí las herramientas asisten al programador en la construcción de la prueba.

1.1 Verificación de Modelos

El método de verificación llamado *verificación de modelos* ("*model checking*") fue desarrollado para analizar *sistemas reactivos* –aquellos que se comportan como una secuencia de estímulos-respuestas en relación al medio– y posteriormente ha sido extendido también a *sistemas de tiempo real*, en los cuales la corrección depende de las magnitudes de los retardos temporales. Usando esta metodología los sistemas se modelan como *grafos* y la especificación se expresa, generalmente, mediante fórmulas en una *lógica temporal* (por ejemplo, [5]). Un procedimiento eficiente se utiliza para determinar automáticamente si las especificaciones son satisfechas por los grafos. Esta técnica ha sido usada con éxito para detectar errores sutiles en distintos sistemas, en particular en protocolos de comunicaciones. Durante los últimos años el tamaño de los sistemas que pueden ser verificados de esta manera se ha incrementado notoriamente. Esto ha sido consecuencia del desarrollo de nuevas técnicas dentro de la misma estrategia, como el "*symbolic model checking*" y la verificación "*on the fly*". Entre las herramientas que implementan algunas de estas técnicas se destacan *Kronos*, *HyTech* y *Uppaal*.

1.2 Demostración de Corrección

Esta aproximación permite construir una *demostración* –en el sentido matemático del término– de que un sistema satisface una especificación. En este trabajo estamos interesados en herramientas basadas en *teorías constructivas de tipos* [6], las cuales han sido formuladas como fundamento de la Matemática Constructiva.

En la última década varios equipos de investigación han dedicado un considerable esfuerzo al diseño e implementación de editores de prueba interactivos basados en teorías de tipos. Ejemplos de estos sistemas son *ALF*, *Coq* y *LEGO*. Una de las principales características de los mismos es el carácter unificador de la teoría que implementan, en la cual pueden ser expresados programas, teoremas y pruebas de éstos. Otro punto destacable es que el usuario es guiado en forma interactiva por el sistema en el proceso de construcción de un programa o una prueba, siendo verificada inmediatamente la validez de cada paso del desarrollo. El principal objetivo de estos sistemas es convertirse en sofisticadas herramientas que asistan en la tarea del desarrollo incremental de programas correctos. Sin embargo, el marco conceptual necesario para desarrollar software verificado es de una muy alta complejidad y requiere cubrir muchos aspectos que en realidad escapan a la construcción de un asistente de pruebas. Estos sistemas disponen de un lenguaje de especificación de orden superior, permiten hacer pruebas en lógica de alto orden y proveen definiciones de tipos (co)inductivos. La experimentación desarrollada en su uso se ha enfocado principalmente en demostrar la corrección de programas secuenciales, pero dado su poder expresivo consideramos que pueden ser también adecuados

para razonar sobre sistemas reactivos y, en particular, de tiempo real. Algunas experiencias llevadas a cabo en esta dirección y particularmente en *Coq*, para sistemas reactivos, son [8, 9].

1.3 Complementariedad de los Enfoques

Los verificadores de modelos ("*model checkers*") son usados actualmente en la industria con éxito para verificar sistemas reactivos, paralelos y de tiempo real, ya que son fáciles de usar y no requieren la asistencia del usuario en el proceso de prueba. Sin embargo ellos en general presentan problemas al tratar con sistemas que involucran un gran número de estados o que tienen parámetros variables, no acotados. Los asistentes de prueba antes citados proveen una solución alternativa para estos casos, aunque la complejidad del proceso de análisis se incrementa muchas veces en forma considerable. No obstante, el enfoque deductivo presenta como ventajas comparativas, además de las nombradas, entre otras, las siguientes: (1) Al demostrar una propiedad de un sistema no sólo se tiene la certeza de que vale la propiedad, sino por qué esto es así. El desarrollo de una prueba induce a tener un conocimiento más profundo del sistema, muchas veces a descubrir nuevas propiedades o generalizarlas. Asimismo, en el proceso de prueba pueden llegarse a descubrir las causas por las cuales una propiedad no vale e incluso, algunas veces, las modificaciones necesarias del sistema para que la cumpla. (2) El proceso de construcción de las pruebas es incremental y composicional. Esto es, las propiedades demostradas pueden ser utilizadas en la prueba de otras, sin ser necesario hacer dicho proceso cada vez desde cero. Además, una demostración bien pensada permite reutilizar una estrategia de prueba en la demostración de otros teoremas. (3) En el método de verificación de modelos, cuando una modificación es introducida en el sistema todo el proceso de verificación debería ser ejecutado nuevamente desde el comienzo. Sin embargo muchas veces las pruebas, o algunas de ellas, o partes de las mismas, pueden mantenerse luego de un cambio, si la estrategia de demostración está bien estructurada (por ejemplo, al modificar ciertas constantes del problema o algunas relaciones entre ellas). (4) Las pruebas de un sistema pueden permitir generalizar la especificación del mismo preservando propiedades de interés.

Los dos enfoques referidos se consideraban al comienzo diametralmente opuestos para el análisis de sistemas reactivos y de tiempo real. Sin embargo en los últimos años surgió un interés creciente en combinarlos debido, en parte, a que los mismos pueden ser cooperativos y no sólo competitivos entre sí. La idea es compensar las desventajas de un enfoque con las ventajas del otro, aunque aún no está claro cómo lograr dicho objetivo. Algunos referentes en esta dirección son [16, 19], entre otros muchos.

Este trabajo estuvo inicialmente enmarcado en un proyecto de investigación que vinculó al grupo de Métodos Formales del Instituto de Computación de la Universidad de la República (Uruguay) y al proyecto *Coq* del INRIA –Rocquencourt (Francia). El proyecto, titulado "Integración de dos enfoques para la verificación formal de sistemas reactivos: Teoría de Tipos y Verificación de Modelos", tiene por objetivo estudiar la integración de los dos enfoques previamente referidos para la verificación de sistemas reactivos y de tiempo real. En este marco, nuestro objetivo consistió en dar los primeros pasos en una combinación entre ambos enfoques, estableciendo una metodología de trabajo que permita compatibilizar el uso de un *model checker* como *Kronos* [21] y el asistente de pruebas *Coq* en el análisis de sistemas reactivos y de tiempo real. A fin de lograr esto propusimos formalizar *grafos* (*autómatas*) *temporizados* [1, 12, 17] y la *lógica TCTL* (*timed computation tree logic*) [1, 12] en el *cálculo de construcciones (co)inductivas* de *Coq* [3]. Los grafos temporizados permiten describir sistemas de tiempo real, mientras que la lógica *TCTL* es un lenguaje adecuado y ampliamente usado para especificar requerimientos temporales. *Kronos* permite verificar si un grafo temporizado satisface una fórmula *TCTL*, siempre que los parámetros del sistema sean valores constantes. El enfoque deductivo permite trabajar con parámetros variables y estructuras infinitas, y de esta manera analizar sistemas más generales. En este contexto a las ventajas citadas del enfoque deductivo se suma una muy importante: la posibilidad de realizar *síntesis de programas* a partir de una formalización en teoría de tipos. Esta es una línea que no será explotada en el trabajo pero que justifica aún más el interés de esta experiencia.

1.4 Acerca de Coq

El asistente de pruebas *Coq* es una implementación del *cálculo de construcciones inductivas*, una lógica intuicionista de alto orden con tipos dependientes y tipos inductivos como objetos primitivos. El usuario introduce definiciones y hace demostraciones en un estilo de *deducción natural*, usando *tácticas*, las cuales son chequeadas mecánicamente por el sistema. Dicho formalismo permite especificar y probar en lógica intuicionista de alto orden. Esta lógica asocia una interpretación computacional a las pruebas, la noción de veracidad de una proposición corresponde a la existencia de una prueba. Además de los habituales tipos inductivos (o sea, conjuntos definidos inductivamente, como por ejemplo los números naturales o las listas finitas), *Coq* permite también la definición de tipos *co-inductivos*. Estos son tipos recursivos que pueden contener objetos infinitos, no bien fundados. Un ejemplo de tipos co-inductivos es el de las secuencias infinitas, usualmente llamadas *streams*, de elementos de un tipo dado. En este trabajo no estamos interesados en dar una descripción completa del cálculo de construcciones inductivas y co-inductivas, ni del sistema de *Coq* en

general. Por aspectos teóricos el lector puede referirse a [6] por el cálculo puro de construcciones, a [18] por tipos inductivos y a [7, 8] por tipos co-inductivos. Acerca de *Coq*, una buena descripción es el manual de referencia [3].

1.5 Organización del Trabajo

La organización del trabajo es como sigue. En la sección 2 introducimos los grafos temporizados, usados para describir sistemas de tiempo real, y la lógica *TCTL*, utilizada como lenguaje de especificación de propiedades temporales. Analizamos una discretización del dominio temporal continuo inherente a los grafos y la lógica considerados, en base a la cual obtenemos una semántica alternativa que asumiremos en el resto de este trabajo. En la sección 3 formalizamos en *Coq* grafos temporizados y la lógica *TCTL*. Incluimos además operadores de la lógica *CTL* (*computation tree logic*) [5], que permite razonar sobre sistemas reactivos, y algunas definiciones alternativas para los operadores que permiten expresar *invarianza* y *alcanzabilidad*, con tipos inductivos y co-inductivos. Asimismo, formulamos algunas propiedades generales de los operadores temporales. En la sección 4 exhibimos las conclusiones, algunos trabajos relacionados y los trabajos futuros. En particular, describimos una metodología de trabajo para la especificación y el análisis de sistemas reactivos y de tiempo real en función de las formalizaciones introducidas en este artículo y los desarrollos expuestos en [13].

2. Sistemas de Tiempo Real: Grafos Temporizados y TCTL

2.1 Grafos (Autómatas) Temporizados

Los grafos –también llamados autómatas– temporizados constituyen un modelo matemático-computacional en el cual pueden representarse de manera formal, simple, clara y modular muchos problemas del mundo real donde hay restricciones temporales que interfieren con transiciones discretas, las cuales representan acciones o eventos. Varias definiciones similares de grafos temporizados han sido propuestas, como es el caso de [1, 12, 17]. Un grafo temporizado es un autómata extendido con un conjunto finito de variables reales, llamadas relojes, cuyos valores se incrementan uniformemente con el paso del tiempo [1, 17]. Dichos relojes son utilizados para medir, por ejemplo, el tiempo transcurrido entre dos eventos, el tiempo de espera o la demora de una comunicación. Las restricciones temporales inherentes a las acciones del sistema son expresadas ligando condiciones de activación a cada una de las transiciones del autómata. Una transición está habilitada, es decir puede ser atravesada, cuando su condición asociada es satisfecha por los valores de los relojes. Un reloj puede ser puesto a cero en cualquier transición. A todo instante, el valor de un reloj es igual al tiempo transcurrido desde la última vez en que fue puesto a cero.

Definición. Sea A un conjunto global de *etiquetas*. Un *grafo temporizado* es una quíntupla $G = \langle L, X, E, l^0, I \rangle$:

- L es un conjunto finito de nodos llamados *locaciones*.
- X es un conjunto finito de *relojes*. Una *valuación* v de los relojes es una función que asigna un valor $v(x) \in R^+$ a cada reloj $x \in X$, donde R^+ son los números reales no negativos. V denota el conjunto de las valuaciones. $v + t$ denota la valuación v' tal que $v'(x) = v(x) + t$ para todos los relojes $x \in X$.
- E es un conjunto finito de aristas llamadas *transiciones*. Cada transición $e = \langle l, \alpha, \psi_X, \rho_X, l' \rangle$ consiste en una locación origen $l \in L$, una locación destino $l' \in L$, una etiqueta $\alpha \in A$, una condición ψ_X y un conjunto $\rho_X \subseteq X$ de relojes que son puestos a cero (reseteados) simultáneamente con la transición.
  Una condición es una combinación booleana de átomos de la forma $x \prec c$; donde, $x \in X$, $\prec$ es una relación binaria en el conjunto $\{<, \leq, =, \geq, >\}$ y c es una constante entera positiva. La transición $e = \langle l, \alpha, \psi_X, \rho_X, l' \rangle$ está habilitada en un estado $\langle l, v \rangle$ si v satisface la condición ψ_X. El estado $\langle l', v[\rho_X := 0] \rangle$ es el *sucesor discreto* del estado $\langle l, v \rangle$ por α, con $v[\rho_X := 0]$ la valuación v' tal que $v'(x) = 0$ si $x \in \rho_X$ y $v'(x) = v(x)$ en caso contrario.
- l^0 es la *locación inicial*. El estado inicial del sistema es $\langle l^0, v^0 \rangle$, con $v^0(x) = 0$ para todo $x \in X$, es decir la locación inicial con todos los relojes puestos en cero.
- I es el conjunto de *invariantes de las locaciones* del grafo. Para cada $l \in L$, I_l es el *invariante* de la locación. Cuando el *sistema* se encuentra en un estado $\langle l, v \rangle$, éste puede permanecer en la locación l dejando pasar el tiempo, mientras la valuación corriente de los relojes satisfaga el invariante I_l. El estado $\langle l, v + t \rangle$ es un *sucesor temporal* del estado $\langle l, v \rangle$, si $v + t$ satisface I_l.

La semántica formal de un grafo temporizado puede definirse en función de *un sistema de transiciones etiquetado* [17], donde los estados son pares de $L \times V$ y las transiciones son de dos tipos: *Temporales*. Los nodos del grafo representan una actividad continua, dada por el paso del tiempo. El paso de un tiempo t es representado por una transición etiquetada con t. *Discretas* (*instantáneas*). Las aristas del grafo representan acciones discretas. La ejecución de una acción α es una transición que lleva la etiqueta α.

2.1.1 Trazas de Ejecución

Una *ejecución* o *traza de ejecución* de un grafo temporizado G con estado inicial q_0 es una secuencia infinita de estados $q_0 \xrightarrow{t_0} q_0' \xrightarrow{\alpha_0} q_1 \xrightarrow{t_1} q_1' \xrightarrow{\alpha_1} q_2 \ldots$, donde $q_i \xrightarrow{t_i} q_i'$ representa una transición temporal y $q_i' \xrightarrow{\alpha_i} q_{i+1}$ es una transición discreta o una transición temporal con tiempo cero. Sea r una ejecución de G con estado inicial q_0, una *posición* π de r es un par $\langle i, t\rangle \in N \times R^+$, con $0 \le t \le t_i$. Escribimos $\sigma_r(\pi)$ para denotar al estado $\langle l, v + t\rangle$ con $q_i = \langle l, v\rangle$, y $\delta_r(\pi)$ para el tiempo $\Sigma_{j<i}\, t_j + t$ transcurrido desde el comienzo de la ejecución r. Una ejecución r de G es *divergente* si para cada $t \in R^+$ existe una posición π de r tal que $\delta_r(\pi) > t$. Las ejecuciones divergentes son aquellas en las cuales el tiempo avanza más allá de cualquier límite, diverge; en la literatura son denominadas "*non-Zeno*". Nosotros estamos interesados, al igual que los trabajos previos (por ejemplo [1, 12, 17]), en sistemas de tiempo real en los cuales todo prefijo finito de una ejecución de G es prefijo de una ejecución divergente de G. El comportamiento (cada traza de ejecución) de estos sistemas, que llamaremos *sistemas bien temporizados*, puede ser generado transición a transición, comenzando en un estado inicial y repetidamente eligiendo entre incrementar el tiempo y hacer una transición discreta.

2.1.2 Composición Paralela de Grafos Temporizados

Para facilitar la descripción modular de los sistemas se utiliza la *composición paralela* de grafos temporizados. La composición paralela de dos grafos temporizados es el producto de ambos, donde las transiciones que comparten etiquetas deben *sincronizar*. Es decir, las acciones correspondientes tienen que ejecutarse simultáneamente. Por cada par de dichas transiciones el sistema global tendrá una única transición tal que la etiqueta es la misma, la condición es la conjunción de las condiciones y el conjunto de relojes a resetear es la unión de los conjuntos correspondientes. Por más detalles ver [17, 13].

2.2 Lógica TCTL: "Timed Computation Tree Logic"

Muchas propiedades importantes de los sistemas encuentran una expresión natural en lógica temporal. La lógica temporal de tiempo real TCTL ("timed computation tree logic") [1, 12] extiende los operadores temporales $\exists\mu$ (*existe una ejecución*) y $\forall\mu$ (*para todas las ejecuciones*) de CTL ("computation tree logic") [5] con restricciones temporales que permiten un razonamiento "*cuantitativo*" del tiempo. Si bien CTL es una lógica temporal adecuada para sistemas reactivos, ésta permite razonamiento "*cualitativo*" del tiempo basado en la noción de *secuencialidad* en las ejecuciones, pero no es posible expresar en ella restricciones temporales cuantitativas. En CTL pueden expresarse propiedades tales como "inevitablemente sucederá el evento *e*" o "la propiedad *p* se satisface continuamente en todas las ejecuciones del sistema". Las fórmulas de TCTL permiten expresar, además, propiedades tales como "inevitablemente antes de un tiempo *t* sucederá el evento *e*" o "la propiedad *p* se satisface continuamente entre los tiempos t_i y t_f para toda ejecución del sistema".

Las fórmulas de TCTL son interpretadas sobre los estados de un sistema de tiempo real y se definen a partir de un conjunto de predicados básicos sobre los estados. El conjunto P de predicados sobre los estados de un grafo temporizado se define en [17]: $\varphi := @ = l \mid x_i \prec c \mid x_i - x_j \prec d \mid Init$. Donde $l \in L$, $x_i, x_j \in X$, $c \in \mathrm{N}$, $d \in \mathrm{Z}$ y $\prec \in \{<, \le, =, \ge, >\}$. Informalmente, *Init* caracteriza al estado inicial y $@ = l$ al conjunto de los estados cuya locación es l. Las fórmulas de TCTL se definen por la siguiente gramática: $\varphi := p \mid \neg\varphi_1 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \exists\mu_I \varphi_2 \mid \varphi_1 \forall\mu_I \varphi_2$. Donde $p \in P$ e I es un intervalo con extremos enteros positivos (I puede ser abierto o cerrado, acotado o no acotado). Intuitivamente, $\varphi_1 \exists\mu_I \varphi_2$ significa que existe una ejecución divergente del sistema tal que φ_2 se cumple en un estado de la misma, en un tiempo t dentro del intervalo I y que φ_1 se satisface continuamente en los estados previos. $\varphi_1 \forall\mu_I \varphi_2$ expresa que para todas las ejecuciones la propiedad anterior se cumple. La semántica formal de TCTL es descripta en [1, 17] y puede también ser consultada en [13].

2.2.1 Algunas Propiedades Relevantes

Usaremos abreviaturas típicas, como $\exists\Diamond_I\varphi$ (*posible* φ), $\forall\Diamond_I\varphi$ (*inevitable* φ) y $\forall\Box_I\varphi$ (*siempre*φ) en lugar de $true\exists\mu_I\varphi_2$, $true\forall\mu_I\varphi_2$ y $\neg\exists\Diamond_I\neg\varphi$, respectivamente. Para I=[0, ∞) omitiremos el subíndice I, caracterizando a fórmulas de la lógica CTL. La fórmula $\exists\Diamond_I\varphi$ es satisfecha por los estados a partir de los cuales existe una ejecución divergente del sistema tal que φ se cumple en un estado de la misma, en un tiempo t dentro del intervalo I. De esta manera se especifican problemas de *alcanzabilidad* acotada. $\forall\Diamond_I\varphi$ establece que φ es *inevitable*. Un estado q satisface esta fórmula si y sólo si a partir de toda ejecución divergente con estado inicial q existe un estado que satisface φ, dentro del intervalo I. $\forall\Diamond_{\le c}\varphi$ expresa la propiedad de tiempo real de respuesta en tiempo acotado, en la cual un evento debe ocurrir antes de un cierto tiempo c. Finalmente $\forall\Box_I\varphi$ especifica que φ es un *invariante*. Esto es, un estado q satisface $\forall\Box_I\varphi$ si y sólo si toda ejecución con estado inicial q satisface continuamente φ, dentro del intervalo I.

2.3 Verificación de Modelos: Grafos Temporizados y TCTL

Dado un grafo temporizado *G* y una fórmula φ de TCTL, el problema de decidir, algorítmicamente, si *G satisface* φ es una instancia del problema de *verificación de modelos*, solucionada por Alur, Courcoubetis y Dill [1]. Su solución está basada en la construcción explícita de un grafo cociente finito, llamado *grafo de regiones*, a partir del sistema de transiciones de estados *infinito* que caracteriza al grafo *G*. Las regiones están determinadas por una relación de equivalencia sobre el conjunto de los relojes del grafo que unifica configuraciones de los relojes desde las cuales los comportamientos futuros son esencialmente idénticos. Es decir, dos relojes x_1 y x_2 son equivalentes si las mismas secuencias de transiciones discretas son posibles desde x_1 y x_2. La construcción del grafo de regiones conduce a un algoritmo de verificación de modelos que es exponencial en el número de relojes de entrada y en el tamaño de la más grande constante de *G*. Sin embargo en la práctica es a menudo innecesario construir el grafo de regiones entero y consecuentemente pueden desarrollarse algoritmos más eficientes. Por más detalles ver [13].

2.4 Un Modelo de Tiempo Discreto

En [13] analizamos ventajas y desventajas de modelos de tiempo discreto –usando números enteros o naturales– y modelos de tiempo continuo –usando racionales o reales. En este trabajo estamos interesados en el análisis de sistemas de tiempo real para un dominio temporal discreto y particularmente en el marco de *grafos temporizados*. Numerosos trabajos previos consideran modelos de tiempo discreto que usan a los números naturales para representar el tiempo [2, 11]. Este modelo es apropiado para, por ejemplo, ciertas clases de circuitos digitales sincrónicos, donde los eventos ocurren a valores exactos de incremento del tiempo de un reloj global. En los casos en que ello no ocurre el enfoque puede resultar igualmente válido y requiere que el tiempo continuo sea aproximado por la elección de una determinada *granularidad*. La idea es que la elección de una granularidad pequeña resulta suficiente para verificar ciertas propiedades donde valores temporales menores son indistinguibles para el proceso de verificación [1]. En el enfoque de *relojes ficticios* los eventos (las transiciones) pueden ocurrir en tiempo continuo pero son registrados por un reloj global, en tiempo discreto. En este modelo se introduce generalmente una transición especial *tick* y el tiempo incrementa una unidad con cada *tick* [2, 11]. Si varios eventos ocurren entre dos instantes consecutivos de tiempo, ellos pueden ser distinguidos solamente por el ordenamiento temporal, no por el valor real del tiempo.

Para grafos temporizados es posible considerar una semántica alternativa basada en tiempo discreto, tomando valores en N, la cual ha sido discutida en algunos trabajos previos (ver, por ejemplo, [4]). De acuerdo a esta visión los pasos temporales son múltiplos de una constante (*ticks* de un reloj global) y a cada instante el autómata puede elegir entre incrementar el tiempo o hacer alguna transición discreta. Consideremos el fragmento de un autómata temporizado con 2 relojes, de la figura 1(a). El autómata puede permanecer en el estado dejando pasar el tiempo (es decir, los valores de x_1 y x_2 aumentan simultáneamente) mientras $x_1 \leq u$. Cuando x_1 alcanza un valor *k*, asumiendo que $k \leq u$, el autómata puede tomar una transición a otro estado y resetear x_1 a cero. Restringiendo el dominio del tiempo a N, las condiciones de permanencia en cada estado (los *invariantes* de las locaciones) pueden ser reemplazadas –interpretadas– por transiciones *tick* como se muestra en la figura 1(b).

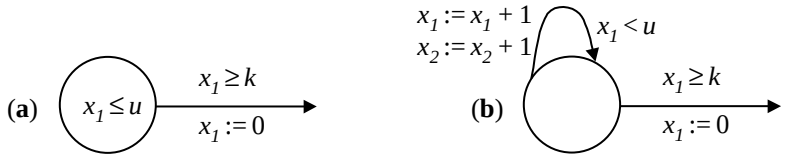

Fig. 1: Un autómata temporizado y su interpretación en tiempo discreto

Bajo esta interpretación los relojes son simplemente variables naturales acotadas, cuyos valores son incrementados simultáneamente por transiciones temporales y algunas de ellas reseteadas a cero por transiciones discretas. En particular, cualquier esquema de representación para una semántica densa, basada en desigualdades de relojes, puede ser especializado para una semántica discreta [4]. Dado que, sobre un orden discreto, cualquier desigualdad de la forma $x_i < c$ puede ser escrita como la desigualdad no estricta $x_i \leq c\text{-}1$, las *regiones* discretas pueden ser expresadas usando exclusivamente desigualdades no estrictas. Esta observación que ha sido aprovechada para mejorar la eficiencia de algoritmos de verificación modelos (ver, por ejemplo, [4]), es considerada con particular interés en el presente trabajo ya que permite resaltar que con una discretización del tiempo "no mucho se pierde" [4, 10, 11]. En particular, [10] construye dos discretizaciones de autómatas temporizados las cuales generan los mismos lenguajes "no temporizados" que sus versiones densas y satisfacen el mismo conjunto de propiedades TCTL. Con una discretización, una *ejecución discreta* puede ser una ligera variación de algunas de las *ejecuciones densas* que ella representa, donde algunas transiciones tienen que tomarse simultáneamente mientras que en una ejecución densa las transiciones son separadas por una pequeña cantidad de tiempo. En el caso de estudio que abordaremos en la sección 4, las propiedades verificadas sobre un dominio discreto valen en el dominio denso de los reales o los racionales, según los resultados establecidos en [4, 11].

3. Formalización de Grafos Temporizados y TCTL en Coq

3.1 Nociones Temporales

En la formalización asumimos a N como dominio temporal discreto, que corresponde al tipo inductivo `nat` de *Coq*. A continuación introducimos algunas definiciones básicas sobre nociones temporales:

```
Definition Instant := nat. Definition Clock := nat. Definition Ini_Ck := O.
Definition tick : Instant := (1). Definition Inc := [x:Clock] (plus x tick).
Definition Reset :=O. Definition time0:= O.
```

`Instant` y `Clock` corresponden al tipo de los valores temporales y las valuaciones de los relojes; `Ini_Ck` es el valor inicial de los relojes; `tick` es la *granularidad* del sistema; `Inc` es la operación que incrementa en un `tick` un reloj `x`; y, `Reset` y `time0` corresponden al valor de reseteo de los relojes y el tiempo inicial del sistema, respectivamente.

3.2 Definiciones Elementales de Grafos Temporizados

Sea $G = \langle Loc = \{loc_0, \ldots, loc_n\}, X, Trans, loc_0, Inv \rangle$ un grafo temporizado. El conjunto global `Label` de etiquetas puede definirse por un tipo inductivo, al igual que las locaciones del grafo *G*:

```
Inductive Label : Set := Lab₁ : Label | ... | Labₘ : Label | Tick : Label.
Inductive Loc : Set := Loc₀ : Loc | ...| Locₙ : Loc.
```

Lab_1, …, Lab_m y *Tick* son los constructores del tipo `Label`. Lab_1, …, Lab_m son las etiquetas que corresponden a las transiciones discretas; *Tick* es una etiqueta distinguida que caracteriza a las transiciones temporales.

El estado del sistema en un instante dado está determinado por una locación de `Loc` y una tupla de valores de tipo `Clock`. Asumiremos que `Clocks` representa el tipo de dicha tupla, correspondiente al conjunto *X*; `InicAll` es la tupla de relojes inicializados (todos puestos en cero); e `IncAll` es la función que incrementa todos los relojes de *X* con la función `Inc`.

```
Definition State := Loc * Clocks. Definition St_ini : State := (Loc₀,InicAll).
```

`St_ini` es el estado inicial del sistema, compuesto por la locación inicial y la tupla de relojes inicializados.

Notación. La cuantificación universal sobre un tipo `S` se escribe en *Coq* "`(x:S) P`". Sin embargo, usaremos la notación "`∀x∈S P`" en este trabajo para dar más claridad a las especificaciones.

Los *invariantes* de las locaciones del grafo pueden ser definidos por predicados sobre los estados, como sigue:

```
Inductive Inv : State -> Prop :=
   ILoc₀   : ∀x∈Clocks (Icond₀ x) -> (Inv (Loc₀,x))
   ...
   |ILocₙ  : ∀x∈Clocks (Icondₙ x) -> (Inv (Locₙ,x)).
```

$Icond_i$ es el predicado de permanencia en la locación Loc_i, para $0 \leq i \leq n$. El constructor $ILoc_i$ construye pruebas del invariante en la locación Loc_i, para los valores de los relojes que cumplen $Icond_i$. Esto es, dado un objeto `x` de tipo `Clocks` y una prueba `H` de $(Icond_i\ x)$, $(ILoc_i\ x\ H)$ es una prueba de $(Inv\ (Loc_i,x))$.

Las transiciones discretas y temporales de *G* son relaciones entre estados y etiquetas. Esto es,

```
Inductive Trans : State -> Label -> State -> Prop :=
  trans₁: ∀x∈Clocks (Tcond₁ x) -> (Inv (Loc_j,new_x₁)) -> (Trans (Loc_i,x) Lab_k (Loc_j,new_x₁))
  ...
  |trans_p:∀x∈Clocks (Tcond_p x) -> (Inv (Loc_j,new_x_p)) -> (Trans (Loc_i,x) Lab_k (Loc_j,new_x_p))
  |tTick: ∀x∈Clocks ∀l∈Loc (Inv (l,(IncAll x))) -> (Trans (l,x) Tick (l,(IncAll x))).
```

$Tcond_1$, …, $Tcond_p$ son los predicados asociados a cada una de las *p* transiciones del grafo; new_x_i es la tupla de `Clocks` donde cada reloj conserva su valor de *x* o es reseteado a cero, con la constante `Reset`; $trans_1$, …, $trans_p$ corresponden a transiciones discretas (del conjunto *Trans* de *G*) y *tTick* al conjunto de transiciones temporales (una por locación). $trans_i$ construye pruebas de transiciones de una locación Loc_i a otra Loc_j por una etiqueta Lab_k, para los valores de los relojes que satisfacen $Tcond_i$ e $(Inv\ (Loc_j,new\_x_i))$; *tTick* construye pruebas de transiciones de una locación a ella misma, incrementando el valor de los relojes en un `Tick` con `IncAll`, siempre que se cumple el predicado `Inv` para los nuevos valores de los relojes en la misma locación.

3.3 Operadores Temporales con Tipos Inductivos

En esta sección formalizamos dos operadores temporales con tipos inductivos que permiten especificar *invarianza* y *alcanzabilidad*. Incluimos la versión de CTL y la cuantitativa temporal de TCTL.

3.3.1 Estados Alcanzables

Los estados *alcanzables* desde un estado q_0 son estados pertenecientes a trazas de ejecución con estado inicial q_0. Sea `S` el tipo de los estados de un sistema de tiempo real y `tr` una relación de transición entre estados de `S` que respeta el formato dado en la sección previa. El conjunto de estados alcanzables desde uno inicial `Sini` a través de `tr` puede definirse inductivamente como sigue,

```
Variables S : Set; tr : S -> Label -> S -> Prop.
Inductive RState [Sini:S] : S -> Prop :=
   rsIni : (RState Sini Sini)
   |rsNext : ∀s1,s2∈S ∀l∈Label (RState Sini s1) -> (tr s1 l s2) -> (RState Sini s2).
```

Los estados alcanzables en tiempo *t* desde un estado `Sini` se formalizan generalizando la definición previa.

```
Inductive RState_T [Sini:S] : S -> Instant  -> Prop :=
   rsIni_T : (RState_T Sini Sini time0)
   |rsNoTime_T : ∀s1,s2∈S ∀l∈Label ∀t∈Instant
      (RState_T Sini s1 t) -> ~l=Tick -> (tr s1 l s2) -> (RState_T Sini s2 t)
   |rsTime_T : ∀s1,s2∈S ∀t∈Instant
      (RState_T Sini s1 t) -> (tr s1 Tick s2) -> (RState_T Sini s2 (Inc t)).
```

El estado inicial es alcanzable en tiempo `time0` (`rsIni_T`); las transiciones discretas no insumen tiempo (`rsNoTime_T`); y, las transiciones temporales representan el paso de un *tick* (`rsTime_T`).

3.3.2 Invarianza y Alcanzabilidad

- $\forall\Box P$ y $\forall\Box_I P$ permiten especificar propiedades invariantes e invarianza acotada.

```
Definition ForAll := [Sini:S; P:S->Prop] ∀s∈S (RState Sini s) -> (P s).
Definition ForAll_T := [Sini:S; P:S->Prop; bound:Instant->Prop]
   ∀s∈S ∀t∈Instant (bound t) -> (RState_T Sini s t) -> (P s).
```

`ForAll` especifica que todos los estados alcanzables desde un estado inicial `Sini` cumplen la propiedad `P` y `ForAll_T`, la propiedad anterior para todos los valores temporales `t` que satisfacen `(bound t)`, es decir `t`$\in I$.

- $\exists\Diamond P$ y $\exists\Diamond_I P$ permiten especificar problemas de alcanzabilidad no acotada y acotada.

```
Inductive Exists [Sini:S; P:S->Prop] : Prop :=
   exists : ∀s∈S (RState Sini s) -> (P s) -> (Exists Sini P).
Inductive Exists_T [Sini:S; P:S->Prop; bound:Instant->Prop] : Prop :=
   exists_T : ∀s∈S ∀t∈Instant (bound t) -> (RState_T Sini s t) -> (P s) ->
            (Exists_T Sini P bound).
```

La relación inductiva `Exists` especifica que existe un estado alcanzable desde un estado inicial `Sini` que cumple la propiedad `P`. `Exists_T` define la propiedad anterior para todos los valores temporales `t` que satisfacen `(bound t)`. Una prueba de $\exists\Diamond$P ($\exists\Diamond$IP) se obtiene a partir de un estado alcanzable que verifica P –y `(bound t)`– y consiste en la construcción de un camino desde el estado inicial. $\forall\Box$ y $\exists\Diamond$ pueden alternativamente definirse instanciando en las formalizaciones de $\exists\Diamond$I y $\forall\Box$I a `bound` con el predicado constante `True`.

3.3.3 Algunas Propiedades

En [13] probamos un conjunto de propiedades para los operadores temporales definidos en la sección previa. A continuación destacamos dos de ellas, a modo de ejemplo.

```
Theorem StepsEX : ∀s1,s2∈S ∀P∈(S->Prop)
   (RState s1 s2) -> (Exists s2 P) -> (Exists s1 P).
Theorem ForAll_EX_T : ∀Sini∈S ∀P∈(S->Prop) ∀bound∈(Instant->Prop)
   (ForAll_T Sini P bound) <-> ~(Exists_T Sini ( [s:S] ~(P s) ) bound).
```

El teorema `StepsEX` establece que si $\exists\Diamond P$ vale a partir de una estado inicial `s2` y `s2` es alcanzable a partir del estado `s1` entonces $\exists\Diamond P$ vale a partir del estado inicial `s1`. `Mon_I_T` permite probar un invariante `Pp` a partir del invariante `Pg` si `Pp` es consecuencia lógica de `Pg`. `ForAll_EX_T` es la equivalencia: $\forall\Box_I P \Leftrightarrow \neg\exists\Diamond_I\neg P$.

3.4 Necesidad de Tipos Co-Inductivos

Si bien muchas propiedades temporales cuantitativas pueden especificarse usando los operadores $\exists\Diamond_I$ y $\forall\Box_I$, otras requieren el uso de las versiones más generales de $\exists\mu_I$ y $\forall\mu_I$. Por ejemplo las propiedades de respuesta en tiempo acotado. El operador $\forall\Diamond_I$ no se formaliza, de forma natural, en base a solamente tipos inductivos y la noción de *estados alcanzables* en tiempo acotado. Es por esto que resulta necesario formalizar el concepto de *traza de ejecución* y consecuentemente, la definición de tipos *co-inductivos* para una descripción completa de todas las fórmulas de TCTL (y

de CTL). En esta sección presentamos, de manera reducida, una formalización en *Coq* de TCTL con tipos co-inductivos. Detalles adicionales y una formalización de CTL pueden consultarse en [13].

Sea `S` el tipo de los estados de un sistema de tiempo real y `tr` una relación de transición entre estados de `S`. Asumiremos además que la pertenencia de un valor temporal a un intervalo está representada por un predicado `bound:Instant->Prop`. Definimos las trazas de estados generadas por `tr` como un predicado co-inductivo `isTrace_T` sobre secuencias infinitas, *streams* (`SPath_T`), de pares (`State_T`) de estados de `S` y valores temporales. Cada elemento de estas trazas es un par que representa al estado del sistema en un punto dado y al valor del *reloj global* en dicho punto. Este reloj es transparente para los sistemas, no es reseteado nunca, y se utiliza para demarcar el tiempo de las ejecuciones. `IsTraceFrom_T` define a las trazas temporizadas que poseen comienzo en un estado inicial dado. `Cons` es el constructor del tipo `Stream`, que dado un elemento y un `Stream` genera un `Stream`. Asumiremos alternativamente una notación infija para `Cons` dada por el operador `^`, asociativo a derecha.

```
Variables S : Set; tr : S->Label->S->Set; bound : Instant->Prop.
Definition State_T := S * Instant.
Definition SPath_T := (Stream State_T).
CoInductive isTrace_T : SPath_T -> Prop :=
   isTraceDisc : ∀x∈SPath_T ∀s1,s2∈S ∀l∈Label (tr s1 l s2) ->
      ~l=Tick -> (isTrace_T (s2,t)^x) -> (isTrace_T ( (s1,t)^(s2,t)^x ))
   |isTraceTick : ∀x∈SPath_T ∀s1,s2∈S ∀t∈Instant (tr s1 Tick s2) ->
      (isTrace_T (s2,(Inc t))^x) -> (isTrace_T ( (s1,t)^(s2,(Inc t))^x )).
Definition isTraceFrom_T := [Sini:State_T; x:SPath_T] Sini=(hd x) /\ (isTrace_T x).
```

isTraceDisc corresponde a una transición discreta, la cual no insume tiempo e *isTraceTick* a una transición temporal etiquetada con `Tick`. El paso del tiempo está representado por `Inc`, que incrementa el *reloj global* del sistema en un *tick*. Asumimos que las propiedades tienen como dominio a *streams* de estados, a fin de permitir caracterizar operadores temporales composicionales [13], donde estos estados son pares de tipo `State_T`. Dado `x∈SPath_T`, las propiedades que sólo refieren a estados del sistema se expresan sobre el elemento `Fst(hd x)`, con `Fst` la función que retorna el primer componente de un par.

El operador μ acotado (μ_I) se define como un predicado inductivo de la siguiente manera:

```
Inductive μ_bound [P,Q:SPath_T->Prop]: SPath_T -> Prop :=
   μ_Further_bound : ∀s∈State_T ∀x∈SPath_T
      (P s^x) -> (μ_bound P Q x) -> (μ_bound P Q s^x)
   |μ_Here_bound : ∀s∈State_T ∀t∈Instant ∀x∈SPath_T
      (Q (s,t)^x) -> (bound t) -> (μ_bound P Q (s,t)^x).
```

A partir de la definición anterior, las formalizaciones de los operadores $\exists\mu_I$ ($P\exists\mu_I Q$) y $\forall\mu_I$ ($P\forall\mu_I Q$), sobre trazas con estado de origen dado, se obtienen como sigue:

```
Inductive EX_μ_bound [Sini:State_T; P,Q:SPath_T->Prop] : Prop :=
   Ex_μ_bound : ∀x∈SPath_T (isTraceFrom_T Sini x) ->
      (μ_bound P Q x) -> (EX_μ_bound Sini P Q).
Definition FA_μ_bound := [Sini:State_T; P,Q:SPath_T->Prop]
   ∀x∈SPath_T (isTraceFrom_T Sini x) -> (μ_bound P Q x).
```

Algunas de las fórmulas más comúnmente usadas de TCTL ($\exists\Diamond_I P$, $\forall\Diamond_I P$ y $\forall\Box_I P$) pueden definirse como abreviaturas de los operadores generales definidos previamente (ver sección 2.2.1). En [13] presentamos formalizaciones alternativas para las fórmulas anteriores que conducen a reducir la complejidad de las demostraciones y el tamaño de los términos de prueba. Asimismo probamos algunas propiedades de los operadores de TCTL que resultan útiles para simplificar ciertas pruebas, particularmente en casos de alcanzabilidad, invarianza y respuesta en tiempo acotado.

4. CONCLUSIONES

Una parte importante del trabajo la dedicamos a analizar cómo representar nociones temporales y razonar deductivamente en teoría de tipos sobre sistemas reactivos y de tiempo real, evaluando la utilidad de tipos inductivos y la necesidad de tipos co-inductivos. Formalizamos en *Coq* sistemas de tiempo real representados como grafos temporizados, asumiendo una semántica de tiempo discreto. A fin de especificar y demostrar requerimientos temporales sobre los sistemas formalizamos la lógica TCTL, y su restricción CTL para razonar sobre sistemas reactivos. Dimos algunas definiciones alternativas para los operadores que permiten expresar invarianza y alcanzabilidad, con tipos inductivos (bajo la noción de estados alcanzables) y co-inductivos (bajo la noción de trazas –infinitas– de ejecución). Estos últimos necesarios para una descripción completa de todas las fórmulas de las lógicas TCTL y CTL. Asimismo, formulamos y probamos en *Coq* propiedades generales de los operadores temporales que permiten simplificar ciertas pruebas, particularmente para propiedades de invarianza y alcanzabilidad. En [13] definimos dos representaciones genéricas de grafos temporizados en *Coq* para la semántica de tiempo discreto considerada, una de las cuales extendimos

a tiempo continuo. Estas representaciones permiten obtener sistemas como instancias particulares y simplifican el proceso de definición de sistemas compuestos con el uso de un operador genérico de composición.

En [14] analizamos un benchmark en el área de los sistemas de tiempo real: *the rail road crossing example* [12, 20]. Especificamos el sistema en *Coq* en base a las formalizaciones introducidas en la sección 3. Analizamos luego la demostración de invariantes, incluyendo la propiedad de seguridad esencial del sistema, y verificamos la divergencia del tiempo en las ejecuciones. Por más detalles ver [13, 14].

Las formalizaciones de grafos temporizados y las lógicas consideradas, traducidas en bibliotecas *Coq* constituyen los aportes de este trabajo y un punto de partida para la realización de nuevos trabajos de investigación en torno al análisis de sistemas de tiempo real en teoría de tipos. En http://coq.inria.fr/ están disponibles las bibliotecas como parte del proyecto Coq del INRIA, Rocquencourt, Francia. A continuación proponemos una metodología de trabajo para la especificación y el análisis de sistemas de tiempo real que relaciona el contenido de las secciones previas y busca compatibilizar el uso de un *model checker* (*MC*) con el desarrollo de sistemas en un ambiente de pruebas. Los objetivos de la misma son: definir un esquema de representación de los sistemas común a los dos enfoques; establecer un proceso de análisis de los sistemas que vincule los resultados de la verificación de propiedades para los enfoques considerados; permitir trabajar con sistemas de tiempo real con *parámetros variables*; y, abarcar una etapa de *síntesis* de sistemas de tiempo real. La metodología es esquematizada en la figura 2.

4.1 Una Metodología de Trabajo

El análisis de sistemas de tiempo real, representados por grafos temporizados, podría dividirse en dos etapas, no necesariamente independientes entre sí ni secuenciales. La primera –optativa y no profundizada en este trabajo– consiste en la especificación y verificación de ciertas propiedades de un sistema en estudio usando un *MC* automático, por ejemplo *Kronos*, *HyTech* o *Uppaal* ("delegación de pruebas" en la figura 2). Entre estas propiedades consideramos relevantes las de *liveness*, como *non-Zeno*, y las de *alcanzabilidad* que son en general más difíciles de demostrar deductivamente, aunque algunas de éstas pueden transformarse en otras equivalentes más simples (en [13] probamos equivalencias para propiedades de alcanzabilidad, invarianza y respuesta en tiempo acotado).

La segunda etapa consiste en la especificación y el análisis del sistema en el cálculo de construcciones (co)inductivas de *Coq* (otros sistemas de pruebas similares a *Coq* son, por ejemplo, *ALF* y *LEGO*). Para esta etapa podemos asumir una semántica de tiempo discreto, como en la mayor parte de este trabajo, o trabajar en un modelo de tiempo continuo, tal como analizamos en [13]. Una manera de incorporar las propiedades demostradas con un *MC* en el ambiente de teoría de tipos consiste en asumir a las mismas como axiomas. Como el lenguaje de especificación de los requerimientos temporales –la lógica TCTL– y el formalismo de descripción de los sistemas –los grafos temporizados– son los mismos en ambas etapas, no son necesarias traducciones significativas para compatibilizar el uso de un *MC* como *Kronos* con un ambiente de teoría de tipos como *Coq*.

La especificación de un sistema puede obtenerse de una representación genérica de los grafos y la composición paralela de los mismos, como describimos en [13], o a través de definiciones particulares para la composición. En [13] analizamos las ventajas de ambos enfoques. Los operadores lógicos de TCTL (y CTL), formalizados en la sección 3, permiten formular requerimientos temporales, es decir pueden ser utilizados para la verificación de propiedades sobre los sistemas especificados. Para propiedades que sólo involucran $\exists\Diamond_I$ y $\forall\Box_I$ ($\exists\Diamond$ y $\forall\Box$) podemos optar entre una formalización con tipos inductivos y una con tipos co-inductivos. Para propiedades arbitrarias la formalización requiere el uso de tipos co-inductivos y por lo tanto, el uso de co-inducción como mecanismo de prueba de ciertas propiedades. Las demostraciones de propiedades de *safety* y de *liveness* de un sistema se basan fuertemente en el conocimiento de un conjunto de propiedades invariantes del sistema. La deducción y prueba de invariantes permite incrementar el conocimiento de un sistema y consecuentemente probar otras propiedades con su ayuda [13].

A partir de las propiedades analizadas de un sistema y consideradas relevantes puede intentarse generalizar la especificación del sistema, asumiendo a las constantes del mismo como variables que deben satisfacer un conjunto de restricciones (deducidas en función de condiciones de prueba elementales entre los relojes y las constantes), tal como procedimos en el análisis del benchmark en [14]. Esta es una clara ventaja del enfoque deductivo utilizado que está condicionada respecto a la utilización de un *MC*. Para el sistema analizado en [14] todas las pruebas se hicieron exclusivamente en el ambiente de pruebas *Coq* y fue posible una generalización de la especificación del problema que preserva todas las propiedades analizadas. Si ciertas pruebas se hicieran con el auxilio de un *MC* –para determinados valores constantes de los parámetros–, la generalización completa no sería posible. Sin embargo, en función de las propiedades demostradas en el ambiente de pruebas podría ensayarse una generalización que especifique un superconjunto de valores posibles para los parámetros, que hagan verdaderas a estas propiedades. Siendo un subconjunto de éstos los valores que hacen verdaderas a todas las propiedades del sistema, incluyendo a las demostradas con el *MC*. Luego al elegir valores constantes para los parámetros del sistema, distintos a los usados para la verificación de las propiedades con el *MC*, deberían demostrarse todas estas propiedades nuevamente con el *MC*, pero no las demostradas

con el asistente de pruebas. Los nuevos valores constantes serían seleccionados a partir de las posibles instancias de los parámetros que satisfacen las propiedades demostradas deductivamente.

Finalmente, una etapa no abordada en este trabajo corresponde a la síntesis de programas, que creemos es una línea promisoria a seguir y en la cual un ambiente de teoría de tipos como *Coq* es particularmente adecuado, ya que permite expresar pruebas y programas en una teoría unificada.

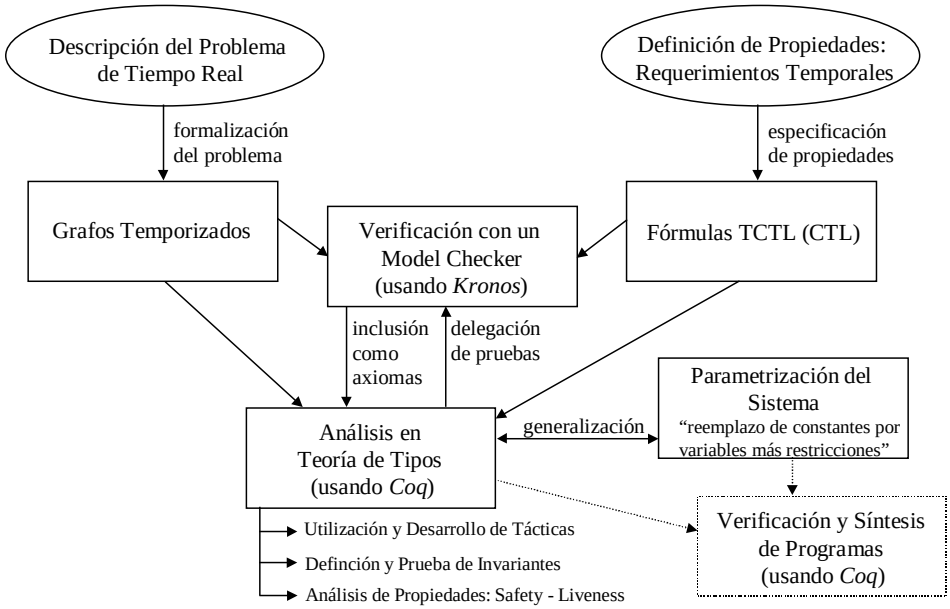

Fig. 2: Una metodología de trabajo para la especificación y el análisis de sistemas de tiempo real

4.2 Trabajos Relacionados

En la sección previa citamos trabajos compatibles con la metodología de trabajo propuesta. Respecto a la utilización "exclusiva" de un *model checker* en la verificación (como por ejemplo *Kronos*, *HyTech* o *Uppaal*), nuestro enfoque permite trabajar con parámetros variables y suma las ventajas de los métodos deductivos de análisis, descriptas en la sección 1.3. Además, los tamaños de las pruebas no dependen de los valores de las constantes del sistema, como ocurre en *model checking* (ver sección 2.3).

En [13] y [14] destacamos las diferencias con otros modelos de especificación y análisis deductivos (por ejemplo, *PVS* [20]) en el marco de un caso de estudio. Nosotros pensamos que la utilización de relojes en el modelo de especificación y la disponibilidad de un mecanismo de definición composicional facilitan el proceso de definición de los sistemas, su comprensión, aún por personas no expertas, y también su análisis, ya que resulta bastante fácil y natural la formulación de propiedades. Otra ventaja de la formalización del sistema en torno a grafos temporizados, en contraste con métodos axiomáticos para especificar restricciones temporales, reside en el contenido algorítmico de la especificación que permite simular el sistema con el control de los múltiples relojes [13]. Creemos que esta formalización además de permitirnos usar un *model checker* automático como asistente para la demostración de ciertas propiedades, es adecuada para la síntesis de programas, en el marco de teoría de tipos y en particular del cálculo de construcciones (co)inductivas de *Coq*. Algunos otros trabajos que buscan relacionar métodos deductivos de prueba y *model checking* son: [9, 16, 19]. En particular [9] analiza posibles combinaciones, en *Coq*, de métodos de demostración asistida de programas y *model checking* para la verificación de programas concurrentes sobre memorias compartidas. Sin embargo el *tiempo* no es considerado un parámetro relevante en la clase de problemas abordados.

4.3 Trabajos Futuros

En este trabajo exploramos una combinación de dos metodologías de análisis de sistemas de tiempo real. A partir de esta experiencia, en torno al ambiente de pruebas *Coq*, surgen algunas líneas promisorias a seguir: (1) desarrollar tácticas generales que permitan simplificar la prueba de propiedades sobre grafos temporizados obtenidos como instancias de representaciones genéricas; (2) analizar la construcción y corrección, en teoría de tipos, de *abstracciones* de sistemas de tiempo real, a fin de demostrar ciertas propiedades sobre sistemas más reducidos (abstractos), que puedan ser luego generalizadas a los sistemas originales. Un trabajo relacionado en esta dirección es [16]; (3) extender el lenguaje de descripción de sistemas de tiempo real con estructuras de datos y en particular aquellas con dominios infinitos. Asimismo, permitir trabajar con grafos temporizados más generales que consideren asignaciones a los relojes de valores no necesariamente constantes. En ambas extensiones los *model checkers* no pueden aplicarse directamente y

consecuentemente una metodología de análisis que incorpore un asistente de pruebas, como la expuesta en la sección 4.1, resulta adecuada; (4) estudiar mecanismos de síntesis de programas de tiempo real en teoría de tipos.

Respecto a la última propuesta, algunas ideas surgen a partir del trabajo [9]. Una noción –primitiva– de programa de tiempo real podría concebirse, inicialmente, como una función que dada una serie (un *stream*) de *eventos* construye una traza de ejecución del sistema. Sin embargo, a diferencia de [9], no toda posible serie de eventos produce una traza válida de ejecución del sistema, ya que el tiempo aquí es un parámetro a considerar. Es decir, la función que dado un estado del sistema y un evento devuelve el nuevo estado del sistema, no es una función total. Podríamos entonces considerar que el tipo de los estados (`State`) incluye un estado distinguido de error (`ErrState`), a partir del cual el sistema es incorrecto. La función que construye las trazas de ejecución –en realidad, *streams* de tipo `State`, donde se incluyen posibes trazas erroneas– sería entonces una función co-recursiva (`mkTrace`) que recibe un estado y una secuencia de eventos. `mkTrace` es en realidad un simulador del sistema. Esto es,

```
Definition Trace  := (Stream State). Definition Events := (Stream Event).

Parameter NexState : State -> Event -> State.

CoFixpoint mkTrace [s:State; evs:Events] : Trace :=

  (Cons s (mkTrace (NextState s (hd evs))) (tl evs)).
```

Donde, `Event` es el tipo de los eventos (para sistemas determinísticos éste podría ser el tipo `Label`) y `NextState` es una función que implementa la relación de transición. Dado un estado y un evento, `NextState` devuelve el nuevo estado del sistema, siendo éste `ErrState` cuando el estado del sistema no es válido para el evento dado o cuando es precisamente el estado de entrada (propagación del estado de error).

Las propiedades del sistema se demuestran para las trazas *válidas*, esto es, aquellas generadas a partir de secuencias de eventos que no generan en ningún punto el estado `ErrState`.

Referencias

[1] R. Alur, C. Courcoubetis, and D. Dill. “Model-checking for real-time systems”. In *Proc. 5th Symp on Logics in Computer Science,* pages 414-425. IEEE Computer Society Press, 1990.

[2] R. Alur and T. Henzinger. “A Really Temporal Logic”. *Journal of the ACM*, 41(1):181-204, 1994.

[3] B. Barras, S. Boutin, C. Cornes, J. Courant, Y. Coscoy, D. Delahaye, D. de Rauglaudre, J-C. Filliâtre, E. Giménez, H. Herbelin, G. Huet, H. Laulhère, C. Muñoz, Ch. Murthy, C. Parent-Vigouroux, P. Loiseleur, Ch. Paulin-Mohring, A. Saïbi, and B. Werner. “The Coq Proof Assistant. Reference Manual, Versión 6.2.4”. *INRIA*, 1999.

[4] M. Bozga, O. Maler, and S. Tripakis. “Efficient verification of timed automata using dense and discrete time semantics”. In L. Pierre and T. Kropf (Eds.), *Proc CHARME’99*, Springer-Verlag, 1999.

[5] E. Clarke, E. Emerson, and A. Sistla. “Automatic verification of finite-state concurrent systems using temporal logic specifications”. *ACM Transactions on Programming Languages and Systems*, 8(2):244-263, 1986.

[6] T. Coquand and G. Huet. “The calculus of constructions”. *Information and Computation*, 76(2/3), 1988.

[7] T. Coquand. “Infinite objects in type theory”. In H. Barendregt and T. Nipkow, editors, *Workshop on Types for Proofs and Programs*, number 806 in LNCS, pages 62-78. Springer-Verlag, 1993.

[8] E. Giménez. *A Calculus of Infinite Constructions and its application to the verification of communicating systems*. PhD thesis, Ecole Normale Supérieure de Lyon, 1996, Unité de Recherche Associée au CNRS No. 1398, 1996.

[9] E. Giménez. “Two Approaches to the Verification of Concurrent Programs in Coq”. To appear, 1999.

[10] A. Göllü, A. Puri, and P. Varaiya. “Discretization of timed automata”. *Proc. 33rd CDC*, Orlando, Florida, 1994.

[11] T. Henzinger, Z. Manna, and A. Pnueli. “What good are digital clocks?”. In *W. Kuich, editor, ICALP 92: Automata, Languages and Programming, LNCS* 623, pages 545-558. Springer-Verlag, 1992.

[12] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. “Symbolic model-checking for real-time systems”. In *Proc. 7th Symp on Logics in Computer Science*. IEEE Computer Society Press, 1992.

[13] C. Luna. *Especificación y análisis de sistemas de tiempo real en teoría de tipos. Caso de estudio: the railroad crossing example*. Master thesis, Technical Report 00-01, InCo, PEDECIBA Informática, Fac. de Ingeniería, U. de la República, Uruguay, Febrero de 2000. Disponible en http://www.fing.edu.uy/inco/pedeciba/bibliote/bases.html y también en http://www.fing.edu.uy/~cluna.

[14] C. Luna. “Verificación de Sistemas de Tiempo Real en Teoría de Tipos. Un Caso de Estudio: The RailRoad Crossing example in Coq”. En proceedings de la Conferencia Latinoamericana de Informática: CLEI'2002, Montevideo, Noviembre de 2002.

[15] D. Mandrioli, Carlo Ghezzi, and Mehdi Jazayeri. *Fundamentals of Software Engineering*. Prentice Hall, 1991.

[16] Olaf Müller and T. Nipkow. “Combining Model Checking and Deduction for I/O-Automata”. In *Tools and Algorithms for the Construction and Analysis of Systems*, *LNCS* 1019, pages 1-16, 1995.

[17] A. Olivero. *Modélisation et Analyse de Systèmes Temporisés et Hybrides*. PhD thesis, Institut National Polytéchnique de Grenoble. France, 1994.

[18] C. Paulin-Mohring. “Inductive definitions in the system Coq – rules and properties”. In M. Bezem and J. Groote, editors, *Proceeedings of the conference Typed Lambda Calculi and Aplications*, *LNCS* 664, 1993.

[19] S. Rajan, N. Shankar, and M. Srivas. “An integration of model checking with automated proof checking”. In *Computer-Aided Verification*, *CAV’95*. *LNCS* 939, Belgium, 1995.

[20] N. Shankar. “Verification of real-time systems using PVS”. In *CAV’93*, Greece. *LNCS* 697, pages 280-291, 1993.

[21] S. Yovine. “Kronos: A verification tool for real-time systems”. *Software Tools for Technology Transfer*, 1997.