APLICACIONES

Computacionales de la Demostración Asistida de Teoremas usando COQ

APLICACIONES

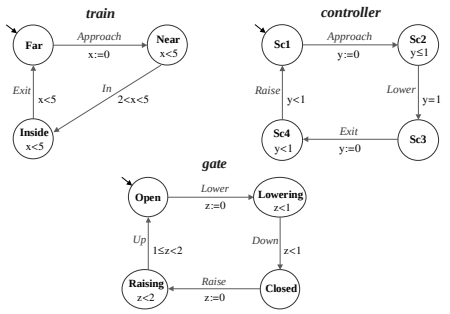
- **Especificación y Análisis de Sistemas Reactivos y de Tiempo Real**
- **Desarrollos en curso en la región**
- **Otras aplicaciones**

Especificación y Análisis de Sistemas Reactivos y de Tiempo Real en Teoría de Tipos

Carlos Daniel Luna
cluna@fing.edu.uy
http://www.fing.edu.uy/~cluna

InCo, U. de la República, Uruguay

Especificación y Análisis de Sistemas de Tiempo Real



Esta charla...

- **Motivaciones y objetivos del trabajo**
- **Lenguajes utilizados:**
  - **Para describir los sistemas**
  - **Para especificar los requerimientos temporales**
- **Formalización de los lenguajes en *Coq***
- **Especificación y análisis de un caso de estudio**
- **Conclusiones, trabajos relacionados y trabajos pendientes**

Motivaciones

- Crecimiento de las aplicaciones de tiempo real
- Interés actual en combinar verificación de modelos y métodos deductivos de prueba
- Utilidad de un asistente de pruebas basado en teoría de tipos para la especificación y el análisis de sistemas de tiempo real

Objetivos

- Especificar y analizar sistemas de tiempo real en teoría de tipos (con *Coq*)
  - Uso de tipos inductivos y co-inductivos
- Estudiar una metodología de trabajo que vincule verificación de modelos y métodos deductivos de prueba
- Validar a través de un caso de estudio (*benchmark* [AD94]) la metodología propuesta

Los Lenguajes

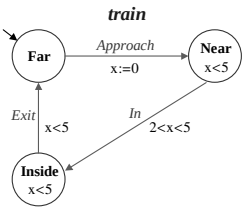
Sistemas de Tiempo Real

Descripción de los sistemas:
Grafos Temporizados

Especificación de requerimientos temporales:
Lógica TCTL

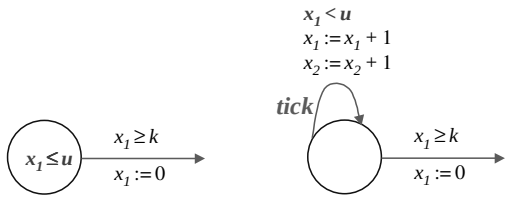
Grafos Temporizados

$G = \langle L, X, E, I^0, I \rangle$



$(Far,0) \xrightarrow{t=1} (Far,1) \xrightarrow{Approach} (Near,0) \xrightarrow{t=3} (Near,3) \xrightarrow{In} (Inside,3)....$

Grafos Temporizados con Tiempo Discreto



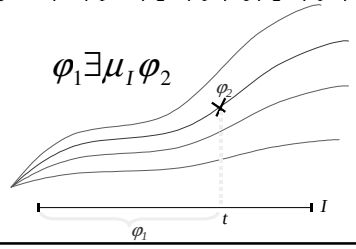
con tiempo continuo

con tiempo discreto

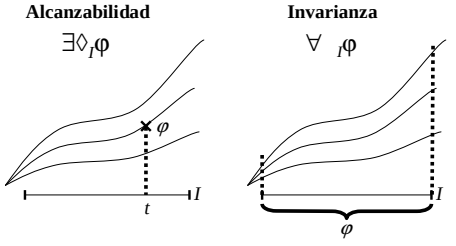
TCTL (timed computation tree logic)

"si el tren está cruzando el paso a nivel la barrera está baja"
"la barrera no permanece baja más de k unidades de tiempo"

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \exists\mu_I \varphi_2 \mid \varphi_1 \forall\mu_I \varphi_2$$



Alcanzabilidad e Invarianza



Liveness (vivacidad)

Safety (seguridad)

Formalizaciones en Coq

- Nociones Temporales:
  - Dominio: ***N***
  - Operadores lógicos habituales (<, =, …)
  - Primitivas de relojes (inicialización, reseto e incremento)

```
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.
```

Formalizaciones en Coq (cont.)

- Grafos Temporizados:
  - Etiquetas (+Tick), Locaciones, Estados, Invariantes de Locaciones y Transiciones
  - En *Coq*: se formalizan con tipos inductivos

Inductive **Label** : Set := Lab_1 : Label | … | Lab_m : Label | *Tick* : Label.

Inductive **Loc** : Set := Loc_0 : Loc | …| Loc_n : Loc.

Definition **State** := Loc * Clocks.

Definition **St_ini** : State := (Loc_0, InicAll).

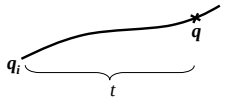
Formalizaciones en Coq (cont.)

Inductive **Inv** : State -> Prop :=
ILoc$_0$: ∀x∈Clocks (Icond$_0$ x) -> (Inv (Loc_0,x))
…
|*ILoc*$_n$: ∀x∈Clocks (Icond$_n$ x) -> (Inv (Loc_n,x)).

Inductive **Trans** : State -> Label -> State -> Prop :=
trans$_1$: ∀x∈Clocks (Tcond$_1$ x) -> (Inv (Loc$_{j1}$,new_x$_1$)) ->
(Trans (Loc$_{i1}$,x) Lab$_{k1}$ (Loc$_{j1}$,new_x$_1$))
…
|*trans*$_p$:∀x∈Clocks (Tcond$_p$ x) -> (Inv (Loc$_{jp}$,new_x$_p$)) ->
(Trans (Loc$_{ip}$,x) Lab$_{kp}$ (Loc$_{jp}$,new_x$_p$))
|*tTick*: ∀x∈Clocks ∀l∈Loc (Inv (l,(IncAll x))) ->
(Trans (l,x) Tick (l,(IncAll x))).

(A) Operadores Temporales con Tipos Inductivos

Conjunto de estados alcanzables



En Coq: con tipos inductivos

Alcanzabilidad ($\exists\Diamond$ - $\exists\Diamond_I$)

"se prueba la existencia de un camino"

Invarianza ($\forall\Box$ - $\forall\Box_I$)

"las pruebas son por inducción en los estados alcanzables"

(A) Operadores Temporales con Tipos Inductivos en Coq

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).

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)).

(A) Operadores Temporales con Tipos Inductivos

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).

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).

Algunas Propiedades de los Operadores Temporales

- Propiedades para invarianza ($\forall\Box$, $\forall\Box_I$):
  - conjunción y monotonía
- Propiedades para alcanzabilidad ($\exists\Diamond$, $\exists\Diamond_I$):
  - transitividad y monotonía
- Equivalencias: $\forall\Box\varphi \Leftrightarrow \neg\exists\Diamond\neg\varphi$
  $\forall\Box_I\varphi \Leftrightarrow \neg\exists\Diamond_I\neg\varphi$

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).

(B) Operadores Temporales con Tipos Co-Inductivos

- Tipos inductivos permiten formalizar alcanzabilidad e invarianza
- "la barrera no permanece baja más de k unidades de tiempo" no se formaliza naturalmente con operadores inductivos
- Necesidad de tipos co-inductivos para formalizar toda la lógica TCTL (y CTL)
  - Trazas de ejecución (en Coq, con *streams*)
    - Definición de los operadores CTL: μ - $\exists\mu$ - $\forall\mu$
    - $\forall\Box\varphi = \neg(true\,\exists\mu\neg\varphi)$!!!
    - Definición de $\exists$ y $\forall$ sobre *streams*
      - Operadores habituales: $\exists\Diamond$ - $\forall\Diamond$ - $\exists\Box$ - $\forall\Box$

Algunas Propiedades de Operadores Temporales con Tipos Co-Inductivos

- Equivalencias: (definiciones de operadores habituales)
  $\exists\Diamond\varphi \Leftrightarrow true\,\exists\mu\varphi$
  $\forall\Diamond\varphi \Leftrightarrow true\,\forall\mu\varphi$
  $\forall\Box\varphi \Leftrightarrow \neg\exists\Diamond\neg\varphi$
  $\exists\Box\varphi \Leftrightarrow \neg\forall\Diamond\neg\varphi$
- conjunción y monotonía de invariantes
- transitividad y monotonía para alcanzabilidad

Extensión a TCTL

- Definición de trazas de ejecución temporizadas
  - Formalización de los operadores: μ_I - $\exists\mu_I$ - $\forall\mu_I$
  - Definición de los operadores habituales:
    - $\exists\Diamond_I$ (alcanzabilidad acotada)
    - $\forall\Diamond_I$ (respuesta acotada)
    - $\forall\Box_I$ (invarianza acotada) y $\exists\Box_I$
- Propiedades de los operadores: análogas a las analizadas para CTL

(B) Operadores Temporales con Tipos Co-Inductivos en Coq

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).

(B) Operadores Temporales con Tipos Co-Inductivos (cont.)

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).

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).

(B) Operadores Temporales con Tipos Co-Inductivos (cont.)

CoInductive **ForAllS** [P:SPath->Prop] : SPath->Prop :=
ForallS : ∀x∈SPath ∀s∈S (P s^x) -> (ForAllS P x) ->
(ForAllS P s^x).

Inductive ***ExistsS*** [P:SPath->Prop] : SPath->Prop :=
Further : ∀x∈Spath ∀S∈S (ExistsS P x) -> (ExistsS P s^x)
|*Here* : ∀x∈Spath (P x) -> (Exists P x).

(B) Operadores Temporales con Tipos Co-Inductivos (cont.)

Definimos algunas de las abreviaturas más comúnmente usadas a través de los operadores ExistsS y ForAllS: $\exists\Diamond_I P$, $\forall\Diamond_I P$, $\exists\Box_I P$ y $\forall\Box_I P$.

Inductive **Posible_T** [Sini:State_T; P:SPath_T->Prop] : Prop :=
posible_T : ∀x∈SPath_T (isTraceFrom_T Sini x) ->
(ExistsS ([s:SPath_T] (bound (Snd (hd s))) /\ (P s)) x) -> (PosibleS_T Sini P).

Definition **Inevitable_T** := [Sini:State_T; P:SPath_T->Prop]
∀x∈SPath_T (isTraceFrom_T Sini x) ->
(ExistsS ([s:SPath_T] (bound (Snd (hd s))) /\ (P s)) x).

Inductive **SafePath_T** [Sini:State_T; P:SPath_T->Prop]: Prop :=
safePath_T : ∀x∈SPath_T (isTraceFrom_T Sini x) ->
(ForAllS ([s:Path_T] (bound (Snd (hd s))) -> (P s)) x) -> (SafePath_T Sini P).

Definition **Always_T** := [Sini:State_T; P:SPath_T->Prop]
∀x∈SPath_T (isTraceFrom_T Sini x) ->
(ForAllS ([s:SPath_T] (bound (Snd (hd s))) -> (P s)) x).

(B) Operadores Temporales con Tipos Co-Inductivos (cont.)

Ejemplos

- Sea *P* una propiedad básica sobre los estados del sistema e *IsSini* el predicado característico de un estado *Sini*.
  La fórmula $IsSini \Rightarrow \forall\Box_I P$ se especifica: (Always_T tr boundI (Sini,time0) P')

  donde, tr es la relación de transición del sistema, boundI es el predicado que corresponde a la pertenencia del valor del reloj global al intervalo *I* y P' es el predicado "[x:SPath_T] (P (Fst (hd x)))" –con SPath_T el tipo (Stream S*Instant), S el tipo de los estados (Sini∈S) y time0 el instante de tiempo inicial.
- Asimismo, una fórmula compuesta $IsSini \Rightarrow \forall\Diamond_J\exists\Diamond_I P$ puede especificarse como sigue: (Inevitable_T tr boundI (Sini,time0) *Q*)

  donde, *Q* es el predicado "[x:SPath_T] (Posible_T tr boundJ' (hd x) P')", con P' el predicado anterior y boundJ' la actualización del predicado boundJ "[t:Instant] (boundJ (plus_Ck t (Snd (hd x))))" (boundJ es el predicado que corresponde a la pertenencia de un valor temporal al intervalo *J*).

(B) Operadores Temporales con Tipos Co-Inductivos (cont.)

Las abreviaturas dadas pueden ser demostradas como teoremas, ahora para las versiones más generales de los operadores $\exists\mu_I$ y $\forall\mu_I$.

Theorem Equiv1_T : ∀Sini∈State_T ∀P∈(SPath_T->Prop)
(Posible_T Sini P) <-> (EX_Until_bound Sini ([_:SPath_T] True) P).

Theorem Equiv2_T : ∀Sini∈State_T ∀P∈(SPath_T->Prop)
(Inevitable_T Sini P) <-> (FA_Until_bound Sini ([_:SPath_T] True) P).

Theorem Equiv3_T : ∀Sini∈State_T ∀P∈(SPath_T->Prop)
(Always_T Sini P) <-> ~(Posible_T Sini ([s:SPath_T] ~(P s))).

Theorem Equiv4_T : ∀Sini∈State_T ∀P∈(SPath_T->Prop)
(SafePath_T Sini P) <-> ~(Inevitable_T Sini ([s:SPath_T] ~(P s))).

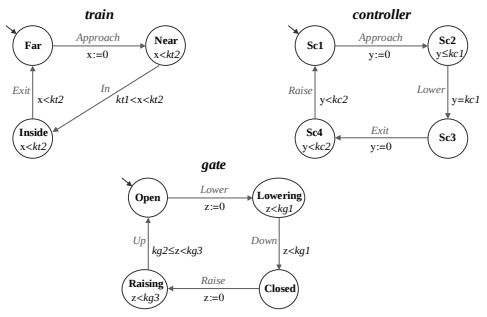
Equiv3_T y Equiv4_T se demuestran en lógica clásica, con el uso del principio del tercero excluido. Las pruebas para estos dos teoremas son por co-inducción, mientras que Equiv1_T y Equiv2_T se demuestran por inducción.

(B) Operadores Temporales con Tipos Co-Inductivos (cont.)

NOTA:

Las formalizaciones para operadores temporales no acotados (CTL) resultan de las presentadas previamente asumiendo que los intervalos son no acotados; es decir, cada predicado bound:Instant->Prop es True para todo t∈Instant.

Control de un Paso Nivel de Tren "the railroad crossing example" [AD94]



Especificación y Análisis del Caso de Estudio

- Definición del sistema en *Coq*
  - Etiquetas, Locaciones, Estados, Invariantes de Locaciones, Transiciones (simples y compuestas)
- Análisis:
  (basado en operadores formalizados con tipos inductivos)
  - Invariantes
  - Safety
  - Liveness *Non-Zeno*

Invariantes del Sistema

- *Init* $\Rightarrow \forall\Box\varphi$
  - propiedades cualitativas
    φ_1: $Loc_T = Far \Rightarrow Loc_G = Open \vee Loc_G = Raising \Rightarrow Loc_C = Sc1$
  - propiedades que expresan restricciones cuantitativas
    φ_2: $Loc_C = Sc2 \Rightarrow z \geq y$
- Pruebas:
  - Inducción
  - Desarrollo de tácticas (estrategias de prueba) para simplificar las pruebas por inducción

```
Tactic Definition BeginForAll :=
[<:tactic:< Unfold ForAll; Induction 1; [Simpl; Intros; Easy | Idtac] >>].
```

Safety y Non-Zeno

- *Safety*
  - "si el tren está cruzando el paso a nivel la barrera está baja"
  - *Init* $\Rightarrow \forall\Box(Loc_T = Inside \Rightarrow Loc_G = Closed)$
- *Non-Zeno*
  - "el tiempo no se bloquea"
  - *Init* $\Rightarrow \forall\Box\exists\Diamond_{=1}$ *True*
  - Prueba:
    - Uso de Invariantes
    - Simplificaciones y lemas auxiliares
  - El sistema [AD94] no cumple *Non-Zeno* !!!

Parametrización del Sistema

- Constantes $\Rightarrow$ Variables + Restricciones
  - Abstracción de propiedades básicas en las pruebas
    - $\forall x \in$ Clock x > *kt1* $\Rightarrow$ x > *kc1*
    - *kc2* > (Inc Reset)
  - *Safety*: $kt1 - kc1 + 1 \geq kg1$

en *Coq*: Constantes $\Rightarrow$ Parámetros
Restricciones $\Rightarrow$ Axiomas

Representaciones Genéricas de Grafos Temporizados

- Grafo (S) = < Trans, Sini, Inv, Inc >
  - Definición de composición paralela de grafos
    - Simplifica el proceso de definición de sistemas compuestos

```
Record Tgraph [S:Set] : Type :=  mkTG {
   trans: S->Label->S->Prop; sini: S; inv: S->Prop; inc: S->Instant->S }.

Definition no_label := [S:Set; tr:S->Label->S->Prop; l:Label]
   ∀s,s'∈S ~(tr s l s').

Variables S1,S2:Set; tr1:S1->Label->S1->Prop; tr2:S2->Label->S2->Prop.
```

Representaciones Genéricas de Grafos Temporizados (cont.)

```
Variables S1,S2:Set; tr1:S1->Label->S1->Prop; tr2:S2->Label->S2->Prop.

Inductive tr_c : (S1*S2)->Label->(S1*S2)->Prop :=
tr_c_syn : ∀s1,s1'∈S1 ∀s2,s2'∈S2 ∀l∈Label
   (tr1 s1 l s1') -> (tr2 s2 l s2') -> (tr_c (s1,s2) l (s1',s2'))
|tr_c_nosyn1 : ∀s1,s1'∈S1 ∀s2,s2'∈S2 ∀l∈Label
   (tr1 s1 l s1') -> (no_label tr2 l) -> (tr_c (s1,s2) l (s1',s2))
|tr_c_nosyn2 : ∀s1,s1'∈S1 ∀s2,s2'∈S2 ∀l∈Label
   (tr2 s2 l s2') -> (no_label tr1 l) -> (tr_c (s1,s2) l (s1,s2')).

Definition inv_c := [S1,S2:Set; inv1:S1->Prop; inv2:S2->Prop; s:(S1*S2)]
   Cases s of (s1,s2) => (inv1 s1) /\ (inv2 s2) end.
```

Representaciones Genéricas de Grafos Temporizados (cont.)

Definition **inc_c** :=
[S1,S2:Set; inc1:S1->Instant->S1; inc2:S2->Instant->S2; s:(S1*S2); t:Instant]
Cases s of (s1,s2) => ((inc1 s1 t),(inc2 s2 t)) end.

Definition **Parallel_composition** :=
[S1,S2:Set; G1:(Tgraph S1); G2:(Tgraph S2)]
(mkTG (tr_c (trans G1) (trans G2))
((sini G1),(sini G2))
(inv_c (inv G1) (inv G2))
(inc_c (inc G1) (inc G2))).

Representaciones Genéricas de Grafos Temporizados (cont.)

CoInductive **isTraceTG_T** [S:Set; G:(Tgraph S)] :
(Stream S*Instant)->Prop :=
isTraceDisc_T: ∀x∈(Stream S*Instant) ∀s1,s2∈S ∀l∈Label ∀t∈Instant
(trans G s1 l s2) -> (inv G s2) -> (isTraceTG_T S G (s2,t)^x) ->
(isTraceTG_T S G ((s1,t)^(s2,t)^x))
|*isTraceTick_T*: ∀x∈(Stream S*Instant) ∀s∈S ∀t∈Instant
(inv G (inc G s tick)) -> (isTraceTG_T S G ((inc G s tick),(Inc t))^x) ->
(isTraceTG_T S G ((s,t)^((inc G s tick),(Inc t))^x)).

Representaciones Genéricas de Grafos Temporizados (cont.)

Una Representación Alternativa

Una representación genérica más simple de grafos temporizados se obtiene si consideramos que las transiciones están representadas por una relación de tipo S->Label->S->Prop, con S el tipo de los estados del grafo y Tick una etiqueta distinguida que representa a las transiciones temporales. De esta manera un grafo es simplemente un par.

Record **Tgraph** [S:Set] : Type :=
mkTG { trans : S->Label->S->Prop; sini : S }.

La composición paralela de dos grafos temporizados es el grafo que compone las transiciones y los estados iniciales. A diferencia que en la formalización previa, el constructor tr_c_syn de tr_c permite sincronizar, además de transiciones discretas, transiciones temporales con Tick.

Definition **Parallel_composition** :=
[S1,S2:Set; G1:(Tgraph S1); G2:(Tgraph S2)]
(mkTG (tr_c (trans G1) (trans G2)) ((sini G1),(sini G2))).

Representaciones Genéricas de Grafos Temporizados

- Extensión a tiempo continuo (***Q***, ***R***)
  - Nociones temporales
  - Trazas de ejecución
  - Análisis de las limitaciones de *Coq* para trabajar con ***Q*** y ***R***

Los tipos Instant y Clock son abreviaciones del tipo T, donde T representa el dominio temporal continuo elegido: Q o R. Los operadores considerados se redefinen en función de T, excepto tick e Inc que no son considerados en el dominio continuo.

Representaciones Genéricas de Grafos Temporizados

La definición de traza de ejecución dada debería ser modificada como sigue:

CoInductive **isTraceTG_T**
[S:Set; G:(Tgraph S)]: (Stream S*Instant)->Prop :=
IsTraceDisc_T: ∀x∈(Stream S*Instant) ∀s1,s2∈S ∀l∈Label ∀t∈Instant
.....
|*IsTraceTick_T*: ∀x∈(Stream S*Instant) ∀s∈S ∀t,tg∈Instant
(∀t'∈Instant (0 <= t') -> (t' <= t) -> (inv G (inc G s t'))**)** ->
(isTraceTG_T S G ((inc G s t),(plus_Ck tg t))^x) ->
(isTraceTG_T S G ((s,tg)^((inc G s t),(plus_Ck tg t))^x)).

Conclusiones

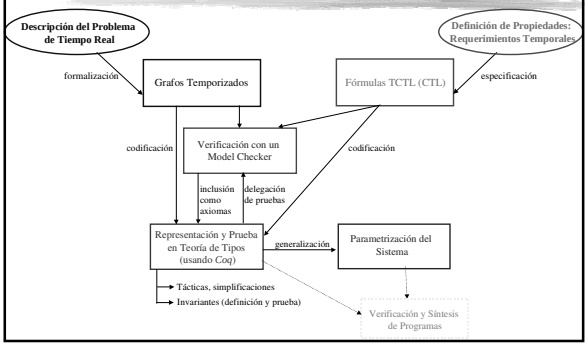
- Formalización de sistemas de tiempo real: Grafos Temporizados
  - Dominio temporal (***N*** — ***Q***, ***R***)
  - Representaciones alternativas
- Análisis de propiedades:
  - Formalización de TCTL (y CTL)
    - Definiciones y propiedades
  - Tipos inductivos y tipos co-inductivos
    - Análisis de metodologías de prueba

Conclusiones (Cont.)

- Análisis de un caso de estudio
  - Formalización del sistema
  - Se probaron invariantes
    - una propiedad de *safety*
    - la propiedad de *liveness Non-Zeno*
    - se desarrollaron algunas tácticas de prueba…
    - se utilzaron algunas tácticas automáticas de *Coq*
  - Parametrización del sistema

http://www.fing.edu.uy/~cluna
http://coq.inria.fr/contribs-eng.html

Metodología de Trabajo



Trabajos Relacionados

- "Verification of real-time systems using PVS" [Sha93]
  - **Axiomatiza el tiempo por fuera del sistema**
  - **Las especificaciones son axiomáticas**
  - **Las pruebas no tienen asociada una interpretación computacional**
- "Two approaches to the verification of concurrent programs in Coq" [Gim99]
  - **Analiza programas concurrentes sobre memoria compartida.**
    El tiempo no es un parámetro de interés

Trabajos Pendientes

- **Extender las bibliotecas de CTL y TCTL**
- **Desarrollar tácticas generales y procedimientos de decisión**
- **Analizar abstracciones de sistemas de tiempo real para la prueba de ciertas propiedades**
  $S \subseteq_P S'$ y $P(S')$
- **Extender el lenguaje de los sistemas de tiempo real con estructuras de datos**
- **Estudiar la síntesis de programas de tiempo real**

Más Trabajos Regionales Recientes y en Curso

• Verificación formal de una extensión "segura" de un file system Unix-compatible
Maximiliano Cristia <mcristia@fceia.unr.edu.ar>
gidis-info@fceia.unr.edu.ar, UNR Argentina

• Una semántica formal de primitivas de modificación de estados de sistemas orientados a objetos
Andres Vignaga <avignaga@fing.edu.uy>, InCo Uruguay

• Especificación formal de OCL
Daniel Perovich <perovich@fing.edu.uy>, InCo Uruguay

• Especificación y verificación de un microcontrolador de un marcapasos
Paula Echenique <paulaechenique@hotmail.com>, Uruguay

Formal verification of an extension of a secure, compatible UNIX file system

Lisex es un sistema operativo "seguro".
Incorpora:
• un modelo de seguridad multinivel (MLS) en el núcleo del sistema, y
• métodos formales en el desarrollo.

El modelo MLS que incorpora Lisex 0.0, llamado Lisex Security Model 0.0, es una adaptación del modelo introducido por Bell y LaPadula en 1973.

LSM0.0 ha sido incorporado únicamente al sistema de archivos del núcleo 2.4.14 de Linux y desarrollado utilizando el entorno de pruebas Coq.

Formal verification of an extension of a secure, compatible UNIX file system

El desarrollo de LSM0.0 incluye especificación y verificación formal. Futuras versiones de Lisex implementarán otros modelos equivalentes en otras porciones del núcleo.

Además, Lisex 0.0 implementa listas de control de acceso (ACL) para mejorar el mecanismo DAC tradicional de Linux y otras características de seguridad menores.

Formal verification of an extension of a secure, compatible UNIX file system

Se cuenta con una **especificación** formal de:

- cada una de las operaciones del modelo de cómputo
- cada una de las propiedades que se espera que el modelo verifique.

Verificación:

(s,t:Estado) · P(s) /\ TR(s,Op,t) -> P(t)

Propiedades: Seguridad discrecional; Seguridad simple; Confinamiento; Cambio de atributos de seguridad

Operaciones: aclstat; addUsrGrpToAcl; chmod; chobjsc; chown; chsubsc; close; create; delUsrGrpFromAlc; mkdir; open; oscstat; owner_close; read; readdir; rmdir; sscstat; stat; unlink; write

Una semántica formal de primitivas de modificacion de estados de sistemas orientados a objetos

El problema central abordado en este trabajo es la propuesta de una semántica formal, con un alto nivel de abstracción, para un conjunto de cinco primitivas de modificación de estados:
la creación y destrucción de objetos, la creación y destrucción de links, y la actualización de valores de atributos.

Una semántica formal permite eliminar ambigüedades y propicia la aplicación de métodos formales en etapas tempranas del desarrollo de sistemas orientados a objetos.

En este trabajo se muestra que programas que se expresan como secuencias de cambios de estados son lo suficientemente poderosos como para representar operaciones de nivel de sistemas no triviales.

Una semántica formal de primitivas de modificacion de estados de sistemas orientados a objetos

Se propone además un framework para razonar rigurosamente acerca de estos programas, donde es posible construir pruebas de correctitud para los mismos usando el asistente de pruebas Coq.

El framework es aplicado a un sistema de información realista que muestra la factibilidad del enfoque propuesto.

Especificación formal de OCL

En los paradigmas de orientación a objetos y de orientación a componentes los modelos de información y los contratos de software juegan un rol preponderante.

A los primeros se les asocia invariantes y los segundos pueden expresarse mediante pre- y poscondiciones.

El estándar UML provee los diagramas de clases como la herramienta adecuada para expresar los modelos de información. A su vez, propone el lenguaje formal OCL para expresar los invariantes, pre- y poscondiciones.

OCL ha evolucionado en sucesivas versiones desde el estándar (desarrollado en 1995 por IBM), y distintos autores han propuesto una semántica formal para el mismo.

Especificación formal de OCL

Estos trabajos, en cambio, no permiten utilizar dicha semántica como herramienta para la prueba de propiedades sobre modelos de información en forma automática-asistida.

Este trabajo presenta una especificación del lenguaje OCL. Esta especificación presenta dos características fundamentales:

- es ejecutable (i.e. permite evaluar expresiones para un estado particular)
- es adecuada para realizar pruebas de propiedades generales sobre modelos particulares como ser instanciabilidad, simplificación de invariantes y verificación de correctitud de operaciones.

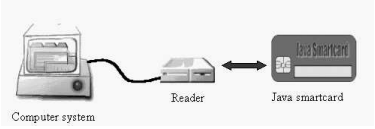
Especificación formal de OCL

Ideas claves:

- Estudiar el nivel de metamodelos ayuda a la elaboración de la especificación
- La elección de un asistente de pruebas como base permite realizar especificaciones utilizables
- Es factible y útil especificar OCL formalmente dado que permite:
  - Entender (y definir) todos los pormenores del lenguaje
  - Probar propiedades de modelos
  - Determinar instanciabilidad
  - Verificar realización de contratos de software

MAS APLICACIONES...

Smart Cards – Java Cards



• **IST Project VerifiCard: Tool-assisted specification and verification of JavaCard**. (http://www.verificard.org/)
The project was completed in november 2003
http://www-sop.inria.fr/lemme/personnel/Gilles.Barthe/

Smart Cards – Java Cards

• **Formal verification techniques for JavaCard**: Gilles Barthe (INRIA). International Winter School on Semantics and Applications (WSSA'2003) 21-31 July, 2003. Montevideo, Uruguay.
http://www-sop.inria.fr/lemme/personnel/Gilles.Barthe/
http://www.fing.edu.uy/inco/eventos/wssa/

Trusted Logic announces
(press release of November 18th, 2003)

that the DCSSI (Direction centrale de la sécurité des systèmes d'information) has successfully evaluated its security methodology applied to the Java Card™ System at the Common Criteria EAL7 level.

Coq is the proof engine used by Trusted Logics, and was chosen for its expressiveness.

(http://coq.inria.fr/)
(http://www.trusted-logic.fr/)

Smart Cards – Java Cards

• **FORMAVIE: Formal Modelling and Verification of the Java Card 2.1.1 Security Architecture (eSmart 2002)**

Trusted Logic (G. Betarte, E. Gimenez, C. Loiseaux)
&
SclumbergerSema (B. Chetali)

El principal objetivo de este proyecto consitió en el desarrollo de una plataforma formal para la verificación de propiedades de seguridad concernientes al proceso de verificación, carga y ejecución de aplicaciones en la plataforma JavaCard 2.1.1.
(http://www.trusted-logic.fr/)

Otros desarrollos...

• Specification of a program generator.

Description: This project consisted in specifying Verilog's compiler L2C. L2C translates Lustre programs into C. Lustre is a declarative language used to describe reactive systems, like the flying commands of Airbus aircrafts. The model included the specification of the abstract syntax and the semantics of both Lustre and the sub set of C used as target language, as well as the algorithm used by the compiler. The correctness proof of the compiler was also sketched (for any program, its semantics are preserved by the compiler).

The Partners: Aerospatiale, Verilog, INRIA, Dassault Aviation
Contact: emmanuel.ledinot@dassault-aviation.fr
Date: 1999

Otros desarrollos...

• Philipps probó hace algunos años la corrección del protocolo BRP (bounded repetition protocol), que se usa en algunos de los sistemas de audio comercializados por Philipps (ver LNCS no. 807, pagina 127: Proof Checking a Data Link Protocol, Helmink, Sellink, Vaandrager).

• Dominique Bolignano probó el protocolo de comercio electronico C-SET (ver http://www.ercim.org/publication/Ercim_News/enw30/bolignano.html y http://citeseer.nj.nec.com/142731.html).

• Specification and verification of Binary Decision Diagram (BDD) algorithms. The application domain was aircraft dependability studies using fault tree analysis. Industrial partners : Dassault Aviation.
Contact : emmanuel.ledinot@dassault-aviation.fr

• .....

Materiales y Contactos

Sitio Oficial de COQ (http://coq.inria.fr/)
- **Documentos (manuales, tutoriales, artículos, monografías, libros)**
- **Software (versiones de coq, interfases-ambientes de trabajo, herramientas utilitarias)**
- **Bibliotecas, contribuciones de usuarios, aplicaciones**
- **Contactos, novedades**
- **Herramientas relacionadas**

Curso de COQ (http://www.fing.edu.uy/inco/grupos/mf/TPPSF/)
- **Gustavo Betarte (http://www.fing.edu.uy/~gustun/)**
- **Carlos Luna (http://www.fing.edu.uy/~cluna/)**

Instituto de Computación, Fac. de Ingeniería, U. de la República, Uruguay