VERIFICACIÓN AUTOMÁTICA DE BASES DE CONOCIMIENTO
Demostración Automática versus Model Checking

Jose A. Alonso Jiménez[1], Joaquín Borrego Díaz[2], Fátima Olías Álvarez
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla. c/Tarfia s.n. 41012–Sevilla
e-mail: jborrego@cica.es

Resumen: *Presentamos un marco formal, utilizando lógicas temporales, para la verificación de sistemas basados en conocimiento, desarrollando ideas de* [Alonso *et al.* 99].

Introducción

En los últimos años se han diseñado gran número de lenguajes para especificar Sistemas Basados en Conocimiento (SBC)[van Eck *et al.* 00], con los que se especifica el conocimiento *declarativo* y se formalizan nociones acerca del método de resolución de la tarea, es decir, conocimiento *procedural*. Es evidente que en tipos muy importantes de SBC, las especificaciones basadas en la lógica de primer orden son importantes. Sin embargo, expresar ciertas propiedades de los SBC con una tal lógica es difícil. Entre las dificultades que se presentan, quizás la más importante es la representación adecuada de propiedades dinámicas del sistema, que relacionan la base de conocimiento con la ejecución de éste. Una solución consiste en aplicar técnicas similares a las de la Ingeniería de Software (IS). Aún existiendo diferencias importantes entre los sistemas estudiados en IS y los SBC, existen similitudes. En [Juristo y Morant 98] se justifica que los SBC son, simplemente, un caso de sistemas con *baja especificación*.

Este trabajo se engloba en el desarrollo de una fundamentación de la verificación automática de SBC, utilizando técnicas clásicas de la Ingeniería del Software; especificando con lógicas temporales y aplicando técnicas de deducción (tableros) y de *Model Checking*. De este modo, cuestiones acerca de la corrección del sistema se traducen en la validación de especificaciones temporales sobre la estructura asociada al sistema.

Sistemas basados en reglas

En este trabajo estudiaremos sistemas basados en reglas, utilizando como lógica base la lógica proposicional:

- Una *regla* r es una expresión de la forma $L_1, L_2, \ldots, L_n \rightarrow M$, donde $LHS(r) = \{L_1, \ldots, L_n\}$, y $RHS(r) = M$ son literales.
- Un *hecho* es un literal del lenguaje (una regla r, con $LHS(r) = \emptyset$).

Notaremos por $hechos(\mathcal{K})$ el conjunto de hechos de la base de conocimiento $\mathcal{K}$ y $reglas(\mathcal{K})$ al conjunto de reglas.

Un sistema basado en reglas (s.b.r.) $\mathcal{E}$ es una 3–upla $\langle E, R, W\rangle$ donde E es el motor de inferencia, R es el conjunto de reglas y W es el espacio de memoria donde se almacenan los hechos (la *memoria de hechos*). La *base de conocimiento* del sistema $\mathcal{E}$ es la unión de R y el conjunto de hechos almacenado en W. Desde este punto de vista, se puede determinar el estado de ejecución especificando al memoria de hechos[3].

Una *ejecución* de un s.b.r. $\mathcal{E}$ es una sucesión de disparos de reglas; una sucesión de estados que se relacionan entre sí por el disparo de una regla. Como nuestro objetivo es

[1]Financiado parcialmente por DGES PB96–0098–C04–04 y por PAI TIC-137
[2]Financiado parcialmente por DGES PB96-1345 y por PAI TIC-137.
[3]Esto presupone que no se modifica el conjunto de reglas. Sin embargo, algunos de los resultados de este trabajo son directamente aplicables a SBC que modifican tal conjunto.

la verificación de la base de conocimiento, no imponemos, ninguna restricción a la gestión de los disparos[4].

Anomalías

Un objetivo general de los s.b.r. es atrapar, mediante computaciones, los hechos que son consecuencia lógica de la base de conocimiento. Si denotamos por $\mathcal{K} \vdash_{\mathcal{E}} L$ si L se infiere mediante una ejecución de $\mathcal{E}$, a partir de $\mathcal{K}$, y por $\mathcal{K} \models L$ el concepto usual de consecuencia lógica, parece imprescindible determinar si es cierto que si $\mathcal{K} \vdash_{\mathcal{E}} L$, entonces $\mathcal{K} \models L$. En general, esta implicación no es una equivalencia.

El objetivo de la verificación y validación de SBC es detectar posibles fallos en la confección tanto de la base de conocimiento como del motor de inferencia. La *verificación* se preocupa de la corrección del sistema con respecto a las especificaciones. Es de carácter lógico. La *validación* consiste en decidir si el sistema resuelve el problema *real*, y requiere comprobación experimental. Algunas de las anomalías que afectan a una base de conocimiento $\mathcal{K}$ son:

Inconsistencia lógica: La base no admite un modelo (lógico).

Inconsistencia procedural: El sistema produce una inconsistencia (un par de literales complementarios, o un conjunto *no permisible*[5]). En general, debido a las limitaciones del sistema, la inconsistencia procedural no es equivalente a la inconsistencia lógica; la *consistencia procedural* es más débil que la *consistencia lógica*.

Redundancia: Una regla o hecho, E, es redundante si su eliminación no reduce el conjunto de hechos inferibles: $\forall L\,[\mathcal{K} \vdash_{\mathcal{E}} L \implies \mathcal{K} \setminus \{E\} \vdash_{\mathcal{E}} L]$

Existen distintos tipos de redundancia: reglas no disparables, reglas con consecuente no usable, literales redundantes, etc. (véase [Preece *et al.* 92] y [Du Zhang y Luqi 99]). La redundancia no es estrictamente una incorrección (no produce inconsistencia), es síntoma de una (posible) mala confección de la base de conocimiento.

Ambivalencia: Existen reglas cuyas consecuencias forman un conjunto no permisible (luego alguna de éstas debe ser no usable).

Circularidad: Un subconjunto de reglas provoca una ejecución infinita.

Incompletitud lógica: Existen hechos válidos en el modelo de estudio que no son consecuencia lógica de la base de conocimiento.

Incompletitud procedural: Existen hechos que son consecuencia lógica de la base de conocimiento pero el sistema no lo deduce.

En la última década, se han presentado diversas aproximaciones al estudio de estas anomalías, con mayor o menor grado de formalización. Véase la bibliografía de [Du Zhang y Luqi 99], el trabajo [Preece *et al.* 92] y [Gupta 91] para más información.

Sistemas reactivos y sistemas de transición

Un *sistema reactivo* está compuesto por procesos (agentes) que se relacionan entre ellos y con su entorno, y reaccionan a estímulos (mensajes) que reciben [Manna y Pnuelli 92]. A diferencia de los programas usuales con *resultado final*, estos sistemas no paran necesariamente[6].

El modelo formal asociado a este tipo de sistemas es un *sistema de transición* con un número finito o numerable de estados, que describen la situación de cada uno de los

[4] Es decir, el objetivo primordial de este trabajo no es especificar E.

[5] Un conjunto de literales es *no permisible* si ha sido declarado como tal. Los conjuntos con un par complementario representa sólo un tipo de éstos.

[6] Por ejemplo, los sistemas operativos, los sistemas de control de centrales nucleares, etc.

agentes que componen el sistema, en un instante determinado[7]. Cada vez que uno de los agentes realiza una acción se alcanza un nuevo estado.

No es necesario, en nuestro caso, los sistemas de transición asociados a los sistemas reactivos en general, basta con una versión simplificada, usual en el estudio de computación concurrente (véase, por ejemplo [Sifakis 82] y [Schnoebelen y Jorrand 89]). Un sistema de transición es una 4–upla $\mathcal{P} = \langle Q, L, \rightarrow, I \rangle$ donde $Q = \{q_1, q_2, \ldots\}$ es un conjunto no vacío de *estados*, L es un conjunto de *acciones* o *eventos*, $\rightarrow \subseteq Q \times L \times Q$ es la *relación de transición*, y $I \subseteq Q$ es el conjunto de estados *iniciales*.

Los SBC como modelos reactivos

Una forma de verificar la base de conocimiento de un SBC, incluyendo características procedurales, consiste en asociar un modelo formal de computación a la base, independientemente del *gestor*: se asocia a cada regla un agente, diseñado para disparar la regla, cuando sea posible [Alonso *et al.* 99][8]. El agente asociado a la regla r es, en pseudocódigo:

l_1: Lectura, en la memoria de hechos, de una k-upla de hechos no considerada, que siguen el patrón de las premisas de $LHS(r)$.
l_2: Chequeo de disparo. Si no es posible, ir a l_1.
l_3: Disparo de la regla. Obtención de $RHS(r)_d$.
l_4: *Sección crítica:* Insertar en la memoria de hechos $RHS(r)$, y eliminar $RHS(r)_d$.
l_5: volver a l_1.

Este tipo de agentes pueden mantenerse en espera (p. e. cuando esperan una nueva k-upla, o en la fase de escritura). Otras características son:

- Tanto en el caso proposicional, como en el caso de primer orden, la organización de la memoria de hechos es relativamente sencilla.
- El agente asociado a r sólo tiene conexión con la parte de la memoria donde se almacenan literales que pueden unificar con $LHS(r)$.

Lógicas temporales

Para integrar la ejecución del sistema en la verificación de la base de conocimiento, se utilizan lógicas que permiten manejar, de manera natural, el tiempo: las denominadas *lógicas temporales*, muy útiles en la verificación de sistemas reactivos[9]. Dependiendo del grado de especificación y del sistema, se usan distintas lógicas temporales.

La lógica $LTLP$ considera el tiempo como un flujo lineal y discreto, y está orientada hacia el futuro. Las fórmulas de $LTLP$ se obtienen mediante la gramática:

$$A ::= p \mid \neg A_1 \mid A_1 \vee A_2 \mid \Diamond A \mid \Box A \mid A_1 \mathcal{U} A_2 \mid \bigcirc A$$

(donde p es una variable proposicional). En un flujo lineal, una fórmula del tipo $\Diamond A$ expresa la validez de A en un instante posterior al actual, y las del tipo $\Box A$ en todo momento posterior, y $\bigcirc A$ en el instante siguiente.

La lógica $LTLP$ se interpreta como un subconjunto de CTL (*Computational Tree Logic*), una lógica que está diseñada para considerar el tiempo como ramificado y discreto (un instante tiene distintos sucesores), y es muy útil para la especificación de la ejecución

[7] Las máquinas con un número finito de estados pueden ser consideradas como un caso particular.

[8] El sistema tiene, como un modelo de transición asociado, el modelo de transición de $\mathcal{K}$ que definiremos más adelante.

[9] El trabajo pionero es [Pnuelli 77]. Véanse, por ejemplo, [Manna y Pnuelli 81] y [Manna y Pnuelli 92].

de sistemas reactivos (permite analizar todas las ejecuciones). La interpretación de $LTLP$ en CTL consiste en interpretar la validez de una $LTLP$–fórmula en un CTL–modelo como la validez en **todo** camino partiendo del estado actual.

Dos problemas básicos son el problema de la *satisfactibilidad* (dada una fórmula, decidir si es válida en algún modelo) y el de *model–Checking*: dadas una estructura temporal y una fórmula CTL, comprobar si dicha fórmula es válida en la estructurados (véanse [Ben Ari 93] y [Clarke *et al.* 00], respectivamente).

Deducción: Especificación temporal

Definición 1 *Sea $\mathcal{K}$ una base de conocimiento. La especificación temporal de $\mathcal{K}$, $Temp(\mathcal{K})$, es el conjunto de fórmulas formado por:*

1. $\{\Box L : L \in hechos(\mathcal{K})\} \cup \{L \rightarrow \Box L \ : \ L \in Leng_R(\mathcal{K})\}$
2. *Para cada $r \in reglas(\mathcal{K})$, $r \equiv L_1, \ldots, L_k \rightarrow M$, la fórmula $L_1 \wedge \cdots \wedge L_k \rightarrow \Diamond M$*

Con esta especificación se determina la consistencia de ciertas bases de conocimiento:

Teorema 2 *Sea $\mathcal{K}$ una base de conocimiento en la que toda regla r de $\mathcal{K}$ verifica que $LHS(r)$ está formado por literales positivos. Entonces $Temp(\mathcal{K})$ es consistente sii $\mathcal{K}$ es proceduralmente consistente.*

El teorema anterior es la base de esta sección. Nótese que como $Temp(\mathcal{K})$ es un conjunto de fórmulas de LTL, se puede aplicar el método del tablero para comprobar si es o no satisfactible, y consecuentemente comprobar si $\mathcal{K}$ es proceduralmente consistente.

Para los experimentos de esta sección, hemos implementado el método de tablero descrito en [Ben Ari 93], con algunas mejoras para el tratamiento de las *hojas preexistentes*.

Consideremos la base $\mathcal{K}_1$ formada por

$$hechos(\mathcal{K}_1) = \{P, Q\} \qquad reglas(\mathcal{K}_1) = \{r_1 \equiv P \rightarrow A, r_2 \equiv A \rightarrow \neg Q\}$$

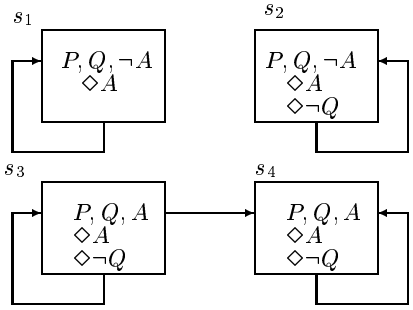
La especificación temporal de $Temp(\mathcal{K}_1)$ es:

$$\{\Box P, \Box Q, P \rightarrow \Diamond A, A \rightarrow \Diamond \neg Q, A \rightarrow \Box A, \neg A \rightarrow \Box \neg A, Q \rightarrow \Box Q, \neg Q \rightarrow \Box \neg Q\}$$

Para comprobar al consistencia de $Temp(\mathcal{K}_1)$, utilizaremos el programa que implementa el método del tablero. El resultado de:

```
>(es-satisfactible
'((ALWAYS P)(ALWAYS Q)
(P => (SOMETIME A)) (A => (SOMETIME (NOT Q)))
(A => (ALWAYS A)) ((NOT A) => (ALWAYS (NOT A)))
(Q => (ALWAYS Q)) ((NOT Q) => (ALWAYS (NOT Q))))
```

es `Fórmula Insatisfactible`, pues la estructura de Hintikka asociada al tablero es:

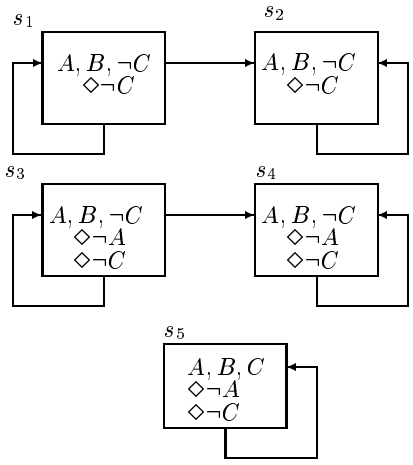


Obsérvese que en la estructura no hay ninguna sub-estructura *full-filling*[10], y por tanto el método de tablero determina la insatisfactibilidad de la fórmula. Sea $\mathcal{K}_2$ formada por $hechos(\mathcal{K}_2) = \{A, B\}$, y $reglas(\mathcal{K}_2) = \{r_1 \equiv B \rightarrow \neg C, r_2 \equiv B \wedge C \rightarrow \neg A, r_3 \equiv A \rightarrow \neg C\}$

[10] Toda fórmula del tipo $\Diamond A$ que aparece en un estado es válida en un camino partiendo de éste.

Entonces $Temp(\mathcal{K}_2) = \left\{ \begin{array}{l} \Box A, \Box B, B \rightarrow \Diamond\neg C, B \wedge C \rightarrow \Diamond\neg A, A \rightarrow \Diamond\neg C, A \rightarrow \Box A \\ \neg A \rightarrow \Box\neg A, C \rightarrow \Box C, \neg C \rightarrow \Box\neg C \end{array} \right.$

Aplicando nuestro programa a $Temp(\mathcal{K}_2)$, en este caso la respuesta es positiva, pues el conjunto $Temp(\mathcal{K}_2)$ es satisfactible (por tanto, $\mathcal{K}_2$ es proceduralmente consistente)[11]. La estructura de Hintikka asociada al tablero obtenido es la siguiente:



Observamos que la sub-estructura formada exclusivamente por s_1, (ó por s_2 ó por ambos) es *full-filling*, por tanto, el método responde que sí es satisfactible. La estructura con un sólo estado, validando A, B y $\neg C$, es un modelo de $Temp(\mathcal{K}_2)$.

Model–Checking: Modelos potenciales

Sea $Leng(\mathcal{K})$ el lenguaje asociado a $\mathcal{K}$, y $Leng_R(\mathcal{K})$ el asociado a $\{RHS(r) \ : \ r \in reglas(\mathcal{K})\}$.

La *extensión procedural* de $Leng(\mathcal{K})$, que notaremos por $Leng^e(\mathcal{K})$, se obtiene añadiendo a $Leng(\mathcal{K})$ un nuevo símbolo proposicional, p_d, por cada $p \in Leng_R(\mathcal{K})$. La extensión *entailment* del lenguaje se obtiene añadiendo una nueva variable proposicional p_L por cada literal del lenguaje. La *instancia deducida* de una fórmula A en el lenguaje $Leng(\mathcal{K})$, que notaremos por A_d, es la fórmula obtenida sustituyendo en A todo símbolo proposicional $p \in Leng_R(\mathcal{K})$ por p_d.

La siguiente definición recoge la idea de *modelo de ejecución* universal para $\mathcal{K}$ (es decir, toda computación de un SBC sobre $\mathcal{K}$):

Definición 3 *El* sistema potencial universal *de $\mathcal{K}$ es el sistema $\mathcal{M}^U_{\mathcal{K}} = \langle Q, L, \rightarrow_{\mathcal{K}}, I\rangle$, donde:*

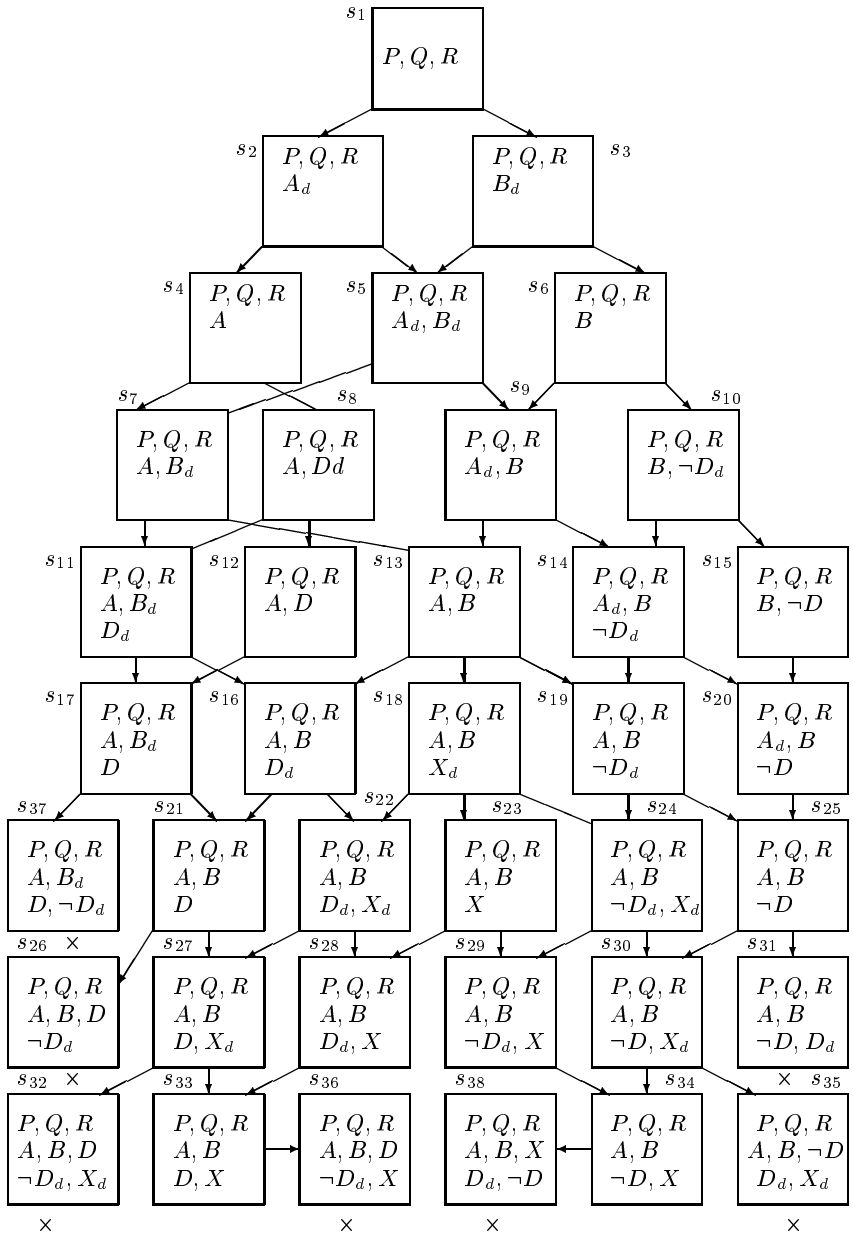
1. $L = reglas(\mathcal{K})$, $I = \{hechos(\mathcal{K})\}$
2. $Q \subseteq 2^{Form(Leng^e(\mathcal{K}))}$ *está formado por los subconjuntos sin par complementario de literales, en el lenguaje $Leng(\mathcal{K})$.*
3. *La relación $\rightarrow_{\mathcal{K}}$ está definida como sigue: Dados $S_1, S_2 \in Q$, y $r \in L$, se verifica que $S_1 \stackrel{r}{\rightarrow}_{\mathcal{K}} S_2$ si se tiene alguna de las siguientes condiciones:*
   - $RHS(r)_d \in S_1$, $RHS(r), \overline{RHS(r)} \notin S_1$, *y* $S_2 = (S_1 \setminus \{RHS(r)_d\}) \cup \{RHS(r)\}$.
   - $LHS(r) \subseteq S_1$, $RHS(r), RHS(r)_d \notin S_2$ *y* $S_2 = S_1 \cup \{RHS(r)_d\}$.
   - *Los estados con un par del tipo $\{A, \overline{A}_d\}$ son terminales.*

[11] Si se observa el conjunto de reglas y hechos que forman $\mathcal{K}_2$ notaremos que no es posible deducir un literal y su complementario. La regla r_2 no es disparable, pues no se deduce C.

El sistema potencial de $\mathcal{K}$, que notaremos por $\mathcal{M}_{\mathcal{K}}$, es el subsistema del sistema potencial universal formado por los estados accesibles desde el estado inicial. Los modelos potenciales de $\mathcal{K}$ se obtienen añadiendo al sistema potencial de $\mathcal{K}$ una interpretación $\mathcal{L}$, que asigna a cada estado s un conjunto de variables. Utilizaremos dos distintos:

- Modelo potencial CWA: $\mathcal{L}(s)$=$\{x$: con x literal positivo de $s\}$
- Modelo potencial *Entailment*: $\mathcal{L}(s)$=$\{P_x$: con x literal de $s\}$

Notése que en el Modelo CWA no hay diferencia entre los hechos negativos demostrados y los hechos indecidibles en un estado (una adaptación de la hipótesis del mundo cerrado, CWA). El modelo potencial de *entailment* considera válidos los hechos positivos y negativos demostrados, a través de la variable del tipo p_x. Ambos modelos describen toda posible ejecución de un s.b.r. sobre $\mathcal{K}$. En el caso de la lógica proposicional, tal modelo es una *máquina finita de estados*. Veamos, con ejemplos, la utilidad de estos modelos. Consideremos el sistema Potencial, $M_{\mathcal{K}}$:



Donde la base $\mathcal{K}$ está formada por $hechos(\mathcal{K}) = \{P, Q, R\}$ y $reglas(\mathcal{K})$ es el conjunto

$$\{r1 \equiv P \wedge Q \rightarrow A, r_2 \equiv R \wedge Q \rightarrow B, r_3 \equiv A \wedge B \rightarrow X, r_4 \equiv A \rightarrow D, r_5 \equiv B \rightarrow \neg D\}$$

En el modelo potencial CWA, se verifica que $\mathcal{L}(s_6) = \mathcal{L}(s_{10}) = \mathcal{L}(s_{15}) = \{P, Q, R, B\}$ Para los siguientes experimentos hemos implementado un algoritmo de Model–Checking distinto del clásico, debido a M. Fisher [Fisher 92], que en la práctica es más eficiente.

• Para comprobar que la fórmula $A_d \rightarrow \Diamond A$ es un invariante, basta comprobar que

$$(A_d \rightarrow \Diamond A) \rightarrow \bigcirc(A_d \rightarrow \Diamond A)$$

es válida en el modelo

• La fórmula $A \vee B \vee A_d \vee B_d$, que también es un invariante. Sin embargo, la fórmula $A_d \rightarrow \bigcirc A_d$ no lo es.

• *Persistencia*: A partir de cierto estado, A permanece en la base de hechos; la fórmula $\Diamond\Box A$, que hemos verificado con el programa.

• Redundancia: Es trivial comprobar que la regla $A \wedge B \rightarrow X$ es redundante para demostrar la inconsistencia procedural de $\mathcal{K}$. Una formalización del hecho de que la regla r_3 es redundante consiste en expresar que se llega a una inconsistencia independientemente de X:

$$(X \rightarrow \Diamond D_d) \rightarrow (\neg X \rightarrow \Diamond D_d), \quad (X \rightarrow \Diamond \neg D_d) \rightarrow (\neg X \rightarrow \Diamond \neg D_d)$$

Los tiempos para cada una de las fórmulas anteriormente descritas se resumen en la siguiente tabla, donde se pone de manifiesto la relación entre la complejidad algorítmica y la complejidad modal de la fórmula a verificar:

Fórmula	**Inicial**	**Salida**	**Tiempo(sg)**
$(A_d \rightarrow \Diamond A) \rightarrow \bigcirc(A_d \rightarrow \Diamond A)$	TODOS	T	0.970
$(A \vee B \vee A_d \vee B_d) \rightarrow \bigcirc(A \vee B \vee A_d \vee B_d)$	TODOS	T	0.520
$(A_d \rightarrow \bigcirc A_d) \rightarrow \bigcirc(A_d \rightarrow \bigcirc A_d)$	TODOS	NIL	0.020
$\Diamond\Box A$	s_1	T	458.020
$(X \rightarrow \Diamond D_d) \rightarrow (\neg X \rightarrow \Diamond D_d)$	s_1	T	3.420
$(X \rightarrow \Diamond \neg D_d) \rightarrow (\neg X \rightarrow \Diamond \neg D_d)$	s_1	T	0.010

Inconsistencia procedural.

Comprobaremos en esta sección que la base de conocimiento $\mathcal{K}$ que estamos tratando es proceduralmente inconsistente, es decir, produce en su ejecución un par de literales complementarios.

Utilizaremos el sistema descrito anteriormente, pero esta vez haremos uso del *Modelo Potencial entailment*. Ya comentamos que el Modelo Potencial CWA no distingue entre los hechos negativos demostrados y los hechos indecidibles. Para el caso que ahora tratamos, es necesario saber si un hecho negativo ha sido deducido, en especial debemos comprobar si por cualquier camino en el modelo, alguna vez se deduce D_d y $\neg D_d$. La fórmula que debe chequearse es, pues, $\Diamond p_{D_d} \wedge \Diamond p_{\neg D_d}$. Es decir, independientemente del orden en que se apliquen las reglas de la base de conocimiento -independientemente del gestor- siempre se llega a una inconsistencia. Como ocurre con la redundancia, sólo es necesario comprobar que es satisfactible en el estado inicial s_1, hecho que hemos comprobado con nuestro programa.

Conclusiones y futuro trabajo

Los experimentos que hemos realizado utilizando el marco formal aquí presentado demuestran que la verificación utilizando técnicas de deducción (tablero) es, en la práctica,

más compleja en coste que el *Model Checking*, si bien, en el desarrollo posterior de este trabajo nos encontraremos con el problema de la explosión combinatoria del número de estados del modelo, dificultad intrínseca a la técnica de *Model-Checking*.

El futuro trabajo estará dirigido a estudiar la utilización de lógicas temporales más expresivas (ramificadas, de primer orden, ...), otras estructuras temporales, estudiando la relación de este trabajo con dichas lógicas y estructuras (como en [Borrego y Kemme 94]), así como la utilización de lógicas temporales con mención explícita del tiempo (tiempo real, factible [Borrego *et al.* 98], etc.).

Referencias

[Alonso *et al.* 99] Alonso Jiménez, J.A.; Borrego Díaz, J.; Pérez Jiménez, M.: *Interpretación reactiva de SBC.* EOCOSIND'99, 187–193 (1999).

[Ben Ari 93] Ben-Ari, M. Mathematical Logic for Computer Science. Prentice Hall, 1993.

[Borrego *et al.* 98] Borrego Díaz, J.; Fernández Margarit, A.; Pérez Jiménez, M. J.: *Especificación y deducción de propiedades temporales factibles.* JIS'98, 351–362 (1998).

[Borrego y Kemme 94] Borrego Díaz, J.; Kemme, B.: *Aplicación de la lógica temporal lineal proposicional para la representación y deducción de conocimientos temporales.* Lenguajes Naturales y Lenguajes Formales X, 459–366 (1994).

[Clarke *et al.* 00] Clarke, E.; Grumberg, O.; Peled, D.: Model Checking. MIT, 2000.

[Du Zhang y Luqi 99] Du Zang; Luqi: *Approximate declarative semantics for rule base anomalies.* Knowledge–Based Systems 12, 341–353 (1999).

[van Eck *et al.* 00] van EcK, P.; Engelfriet, J.; Fensel, D.; van Harmelen, F.; Venema, Y.; Willems, M.: *A Survey of Languages for Specifying Dynamics.* Aparecerá en IEEE Transactions on Data and Knowledge Engineering, 2000.

[Fisher 92] Fisher, M.: *A model checker for linear temporal logic.* Formal Aspects of Computing 4, 299–319 (1992).

[Gupta 91] Gupta, U.: Validating and Verifying KBS. IEEE Press (1991).

[Juristo y Morant 98] Juristo, N,; Morant, J.L.: *Common Framework for the Evaluation process of KBS and Conventional Software.* Knowl. Based Systems, 11, 145-159 (1998).

[Manna y Pnuelli 81] Manna, Z.; Pnuelli, A.: *Verification of concurrent programs: The temporal framwork.* The Correctness Problem in Computer Science (R.S. Boyer y J.S. Moore, eds.), 215,273. Academic Press, 1981.

[Manna y Pnuelli 92] Manna, Z.; Pnuelli, A.: The temporal logic of reactive and concurrent systems. Springer–Verlag (1992).

[Pnuelli 77] Pnuelli, A.: *The Temporal Logic of Programs* 18th IEEE Symp. Found. of Comp. Sci., 46–57 (1997).

[Preece *et al.* 92] Preece, A.D.; Shinghal, R.; Batarekh, A.: *Principles and Practice in verifying Rule–Based Systems.* Knowledge Engineering Review, 7 (2), 115–141 (1992).

[Schnoebelen y Jorrand 89] Schnoebelen Ph.; Jorrand, Ph.: *Principles of FP2: Term Algebras for Specifications of Parallel Machines* en Languages for Parallel Architectures (J.W. de Bakker ed.). Wiley (1989).

[Sifakis 82] Sifakis, J.: *A unified approach for studying the properties of transition systems.* Theoretical Computer Science, 18, 227–258 (1982).