VERIFICACIÓN DE PROCESOS CONCURRENTES. UN MÉTODO FORMAL Y UN CASO DE APLICACIÓN

M. GARCÍA HOFFMANN

En este trabajo se propone una nueva técnica para la verificacion de programas concurrentes. Para realizar la verificación se parte de la descripción LDP de los procesos.

El método requiere expresar en una relación de simulación las propiedades que se desea probar. El algoritmo de verificación construye mediante ejecución - simbólica un árbol de prueba para cada punto de paro de la relación de simulación. La verificación se reduce así a la prueba de las condiciones de verificación generadas.

Como caso de aplicación se presenta la verificación de un conocido protocolo de comunicación: el protocolo de bit alternante.

1. INTRODUCCION

En la prueba estática de corrección de programas se han empleado técnicas diversas ---/20/. La mayor parte de los esfuerzos se han dirigido hacia los programas secuenciales, basándose los métodos en las ideas de Goldstine y Von Neumann más tarde formalizadas por Naur /21/ y Floyd /8/. El método de las aserciones permite demostrar si un programa es correcto o no respecto a un conjunto de predicados sobre las variables del programa. La verificación requiere el establecimiento de una aserción inicial y otra final, de modo que si la aserción inicial se cumple al comienzo de la ejecución del programa, deba cumplirse la aserción final al término de la misma. Para facilitar la prueba se añaden aserciones inductivas en puntos intermedios del programa, reduciéndose la verificación de la corrección parcial a la prueba de las denominadas condiciones de verificación. Diversos autores han seguido esta línea proponiendo sistemas automáticos de verificación /7,12,16,18,23/.

La extensión del método de las aserciones a la prueba de programas concurrentes encuentra aún mayores dificultades ya que el resultado de la ejecución depende del orden impredictible en que se entrelazan las acciones de los diversos procesos. Sin embargo, dentro de la programación concurrente existen determinadas aplicaciones cuyas peculiares características hacen más asequible la verificación. Uno de estos casos lo constituyen los protocolos de comunicación en los cuales, normalmente, se manejan estructuras de datos simples, operaciones sencillas y procesos cíclicos. Los trabajos más relevantes en el campo de la verificación automática de protocolos se deben a Bremer y Danthine /4,5,6/,- Brand y Joyner /3/ y Hajek /13/. En /10/ se presentan dos aspectos de una nueva metodología de diseño de protocolos -la descripción y la validación- que se completa con las técnicas de verificación desarrolladas en este trabajo y las de realización práctica de protocolos discutidos en /11/. Sin embargo, dado que el planteamiento teórico y la solución propuesta al problema de la verifica ción son absolutamente generales, se expondrá el método en el marco más amplio de los procesos concurrentes para señalar posteriormente sus limitaciones y su aplicación al caso particular de los protocolos de comunicación.

2. VERIFICACION DE PROCESOS SECUENCIALES DESCRITOS EN LDP

En /10/ se presenta un lenguaje descriptivo de alto nivel y se muestra su aptitud para modelar los diversos aspectos presentes en la programación concurrente. En esencia, el

- M. García Hoffmann, de la Cátedra de Mètodes Informàtics de la ETSEIB. Av. Diagonal, 647. Barcelona-28

- Article rebut el Setembre de 1980.

LDP es una readaptación y simplificación del lenguaje propuesto por Hoare /17/ realizada con el fín de ajustarlo a las peculiaridades de una familia particular de procesos concurrentes: los protocolos de comunicación. La generalización de la técnica de verificación que se propone para el LDP al lenguaje de Hoare o Hansen /14/ es inmediata.

Diversos autores han empleado la ejecución simbólica en la verificación de programas secuenciales /7,15,19/. La ejecución simbólica es una extensión natural del concepto de ejecución de un programa. Los valores iniciales se convierten en símbolos, las variables en el curso de la ejecución simbólica toman valores simbólicos que serán expresiones compuestas de operadores, constantes y los símbolos iniciales, la verificación requiere la prueba de teoremas en cálculo de predicados de primer orden. En un programa secuencial determinista, fijados los valores de entrada queda determinado el camino de ejecución y, por tanto, el resultado. Las descripciones LDP no son deterministas por lo que los resultados pueden ser diferentes para un mismo vector inicial. Por otro lado, si los valores iniciales son símbolos representando todos los valores iniciales posibles, la ejecución simbólica debe permitir determinar todos los posibles caminos de ejecución.

El objeto de la ejecución simbólica de descripciones en LDP es construir el denominado árbol de prueba. Cada nudo del árbol está formado por tres elementos.

(a) Punto de control: es la etiqueta que indica la posición alcanzada en la ejecución simbólica de la descripción.

(b) Vector de estado: está formado por los valores simbólicos de todas las variables de la descripción. Un valor simbólico es una expresión formada por los operadores del lenguaje, constantes y los símbolos iniciales.

(c) Lista de predicados: conteniendo el conjunto de condiciones que los símbolos iniciales han de satisfacer para que el control pueda llegar al nudo.

El nudo inicial o raíz refleja todas las situaciones en que puede encontrarse inicialmente la descripción. Su punto de control es la etiqueta de la primera sentencia, el vector de estado contiene los valores simbólicos iniciales de todas las variables y la lista de predicados las restricciones que deban cumplir los símbolos iniciales.

La ejecución simbólica de una sentencia de asignación del tipo <variable : = expresión> tiene por efecto la asignación a la variable de la expresión simbólica resultante de sustituir las variables de la expresión por sus valores simbólicos. Las sentencias <variable booleana : =#> generan dos sucesores, uno asignando el valor falso a la variable booleana, el otro asignándole el valor cierto.

En la ejecución simbólica de las sentencias alternativas y repetitivas se evaluan primero las guardias. Para cada guardia que evalue un valor cierto se genera un nudo sucesor. La evaluación de las guardias de cada comando con guardia se realiza como sigue:

(a) Se sustituyen las variables de la guardia por sus valores simbólicos.

(b) Si la lista de predicados implica que la guardia sea falsa, la lista de sentencias del comendo con guardia no se ejecutará nunca a partir del nudo que se está considerando. En este caso es innecesario generar un sucesor que considere la ejecución del comando con guardia.

(c) Si la lista de predicados implica un valor cierto de la guardia, se genera un nudo sucesor idéntico al que se está considerando, pero con el punto de control conteniendo la etiqueta de la primera sentencia de la lista de sentencia del comando.

(d) Si la lista de predicados no implica que la guardia se cierta ni que sea falsa, se crea un sucesor que contemple la posibilidad de que la guardia evalue a cierto. El nuevo nudo tiene el mismo vector de estado que su antecesor; su lista de predicados se obtiene añadiendo la guardia a la lista de predicados de éste y el control se avanza a la lista de sentencias del comando con guardia.

La primera "guardia" que debe considerarse - es la que conduce al fin de la ejecución de la sentencia, es decir, la negación de la -- reunión booleana de las guardias de cada comando con guardia. Si la lista de predica-- dos no implica que esta guardia sea falsa se genera un nudo sucesor añadiendo a la lista de predicados la condición que hace posible este camino. En el caso de la sentencia alternativa este nuevo nudo describe las situaciones en que la descripción LDP es errónea (estas situaciones conducen a la aborción). Si se trata de una sentencia repetitiva el - punto de control es la etiqueta de la si---- guiente sentencia que ha de ejecutarse simbólicamente y el nuevo nudo no es más que la - descripción de las situaciones en que termina la ejecución de la sentencia repetitiva.

Obsérvese que en este apartado se considera un único proceso, por lo que se ignoran aquí las sentencias de asignación externa.

Se denominarán puntos de paro a los lugares de las descripciones en los que insertan --- aserciones. Estas se establecen en forma de una relación de simulación /2/. Cada componente de la relación de simulación consiste en un punto de paro y la aserción correspondiente.

El procedimiento de verificación propuesto - consiste en generar para cada punto de paro el árbol de la prueba que se obtiene ejecu-- tando simbólicamente la descripción LDP desde el punto de paro inicial hasta que se encuentra el punto de paro siguiente. En la - raíz del árbol el punto de control es la etiqueta del punto de paro inicial, el vector - de estado contiene símbolos arbitrarios y la lista de predicados restringe los valores -- simbólicos iniciales de modo que se cumpla - la aserción especificada en la relación de - simulación. Las hojas del árbol serán puntos de paro y en ellos ha de probarse que la aserción es cierta.

Se exponen a continuación dos ejemplos que - clarifican el procedimiento a seguir.

Ejemplo 1: Verificar el siguiente algoritmo que pretende dejar en la variable x el máximo de a, b y c

$$1:[a\geq b \rightarrow 2:[a\geq c \rightarrow 3:x:=a \,[]\, a<c \rightarrow 4:x:=c] \,[]\, a<b \rightarrow 5:[b\geq c \rightarrow 6:x:=b \,[]\, b<c \rightarrow 7:x:=c]];8:$$

La aserción inicial es cierto, al no haber - ninguna restricción sobre los valores iniciales de las variables a, b, c y x. La aser-- ción final debe reflejar el comportamiento - deseado del algoritmo, es decir, que x = --- maxim(a,b,c). Esta aserción será $(x=a \vee x=b \vee x=c) \wedge (x\geq a) \wedge (x\geq b) \wedge (x\geq c)$. La figura 1 muestra el árbol de prueba del algoritmo. La -- raíz del árbol queda definida por

(a) Punto de control: etiqueta de la primera sentencia

(b) Vector de estado: valores simbólicos iniciales de las variables ($a=\alpha$, $b=\beta$, $c=\gamma$, $x=\delta$).

(c) Lista de predicados: vacío (ó cierto).

El resto de los nudos se obtiene ejecutando simbólicamente la descripción. En los puntos de paro a que se llega debe probarse la aserción final, sustituyendo las variables - de ésta por sus valores simbólicos. Concretamente, es necesario demostrar que:

$$(\alpha\geq\beta)\wedge(\alpha\geq\gamma)\supset(\alpha=\alpha\vee\alpha=\beta\vee\alpha=\gamma)\wedge(\alpha\geq\alpha)\wedge(\alpha\geq\beta)\wedge(\alpha\geq\gamma)$$
$$(\alpha\geq\beta)\wedge(\alpha\geq\gamma)\supset(\gamma=\alpha\vee\gamma=\beta\vee\gamma=\gamma)\wedge(\gamma\geq\alpha)\wedge(\gamma\geq\beta)\wedge(\gamma\geq\gamma)$$
$$(\alpha\geq\beta)\wedge(\beta\geq\gamma)\supset(\beta=\alpha\vee\beta=\beta\vee\beta=\gamma)\wedge(\beta\geq\alpha)\wedge(\beta\geq\beta)\wedge(\beta\geq\gamma)$$
$$(\alpha\geq\beta)\wedge(\beta\geq\gamma)\supset(\gamma=\alpha\vee\gamma=\beta\vee\gamma=\gamma)\wedge(\gamma\geq\alpha)\wedge(\gamma\geq\beta)\wedge(\gamma\geq\gamma)$$

Obviamente, los cuatro predicados son ciertos por lo que se puede concluir que el algoritmo es parcialmente correcto respecto a -- las aserciones dadas, La prueba de corrección total requeriría demostrar que el algoritmo termina.

Ejemplo 2: Verificar la siguiente descrip--- ción LDP del algoritmo de Euclides

$$1:x:=m;\ y:=m;$$
$$2:*[x>y \rightarrow 3:x:=x-y \,[]\, y>x \rightarrow 4:y:=y-x];5:$$

que pretende dejar en la variable x el máximo común divisor de dos enteros positivos m y n. Insertaremos tres aserciones en la descripción, la aserción inicial $(m>0)\wedge(n>0)$, que restringe los valores iniciales de las - variables m y n; la aserción final x = (m,n) que expresa el resultado esperado, y la aser

ALGORITMO

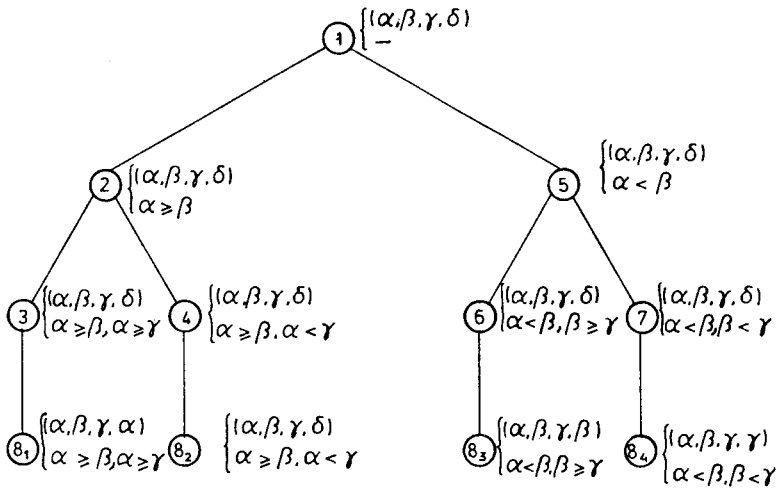
$$1:\Big[a\geqslant b \to 2:\Big[a\geqslant c \to 3: x:=a \;\Big[\Big]\; a<c \to 4: x:=c\Big]$$
$$\Big[\Big]\; a<b \to 5:\Big[b\geqslant c \to 6: x:=b \;\Big[\Big]\; b<c \to 7: x:=c\Big]\Big]; 8:$$

RELACION DE SIMULACION

1; cierto

8; $(x=a \vee x=b \vee x=c) \wedge (x\geqslant a) \wedge (x\geqslant b) \wedge (x\geqslant c)$

ARBOL DE PRUEBA



PRUEBA EN LOS PUNTOS DE PARO FINALES

En 8_1: $(\alpha\geqslant\beta)\wedge(\alpha\geqslant\gamma)\supset(\alpha=\alpha\vee\alpha=\beta\vee\alpha=\gamma)\wedge(\alpha\geqslant\alpha)\wedge(\alpha\geqslant\beta)\wedge(\alpha\geqslant\gamma)$

En 8_2: $(\alpha\geqslant\beta)\wedge(\alpha\geqslant\gamma)\supset(\gamma=\alpha\vee\gamma=\beta\vee\gamma=\gamma)\wedge(\gamma\geqslant\alpha)\wedge(\gamma\geqslant\beta)\wedge(\gamma\geqslant\gamma)$

En 8_3: $(\alpha\geqslant\beta)\wedge(\beta\geqslant\gamma)\supset(\beta=\alpha\vee\beta=\beta\vee\beta=\gamma)\wedge(\beta\geqslant\alpha)\wedge(\beta\geqslant\beta)\wedge(\beta\geqslant\gamma)$

En 8_4: $(\alpha\geqslant\beta)\wedge(\beta\geqslant\gamma)\supset(\gamma=\alpha\vee\gamma=\beta\vee\gamma=\gamma)\wedge(\gamma\geqslant\alpha)\wedge(\gamma\geqslant\beta)\wedge(\gamma\geqslant\gamma)$

Figura 1: Verificación de la descripción LDP de un algoritmo para determinar el máximo de tres números

ción inductiva $(m,n)=(x,y)\wedge(x>0)\wedge(y>0)$, situada al principio del bucle.

La figura 2 muestra el árbol de prueba que comienza en el punto de paro 1 y acaba en el 2. El nudo inicial tiene el punto de control en 1, el vector de estado $(m=\alpha, n=\beta, x=\gamma, y=\delta)$ y la lista de predicados$_{\alpha>0,\beta>0}$. En el punto final ha de probarse que

$$(\alpha>0)\wedge(\beta>0)\supset((\alpha,\beta)=(\alpha,\beta))\wedge(\alpha>0)\wedge(\beta>0),$$

predicado trivialmente cierto.

A partir del punto de paro 2 se construye el árbol de prueba de la figura 2. El punto de control de la raíz es 2, el vector de estado $(m=\alpha, n=\beta, x=\gamma, y=\delta)$ y la lista de predicados $(\alpha,\beta)=(\gamma,\delta)\wedge(\gamma>0)\wedge(\delta>0)$. En el punto de paro final debe probarse el predicado.

$$((\alpha,\beta)=(\gamma,\delta))\wedge(\gamma=\delta)\supset\gamma=(\alpha,\beta)$$

En los puntos 2_1 y 2_2 la prueba toma la forma de los predicados de la figura 2.

Los tres predicados pueden probarse ciertos

ALGORITMO

$1{:}x{:=}m;y{:=}n;2{:}*\left[x>y \rightarrow 3{:}x{:=}x-y \,\square\, y>x \rightarrow 4{:}y{:=}y-x\right];5{:}$

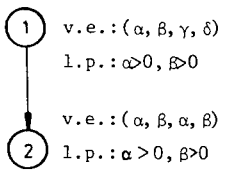
RELACION DE SIMULACION

1; $(m>0)\wedge(n>0)$

2; $((m,n)=(x,y))\wedge(x>0)\wedge(y>0)$

3; $x=(m,n)$

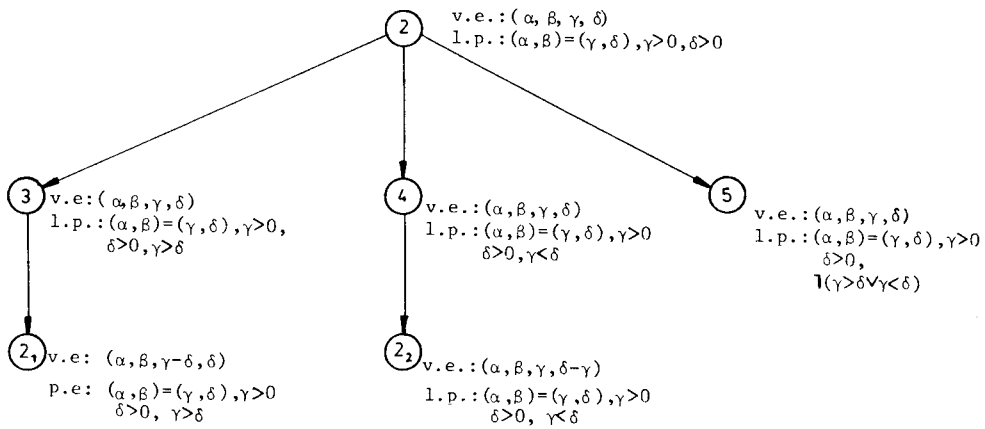
ARBOL DE PRUEBA DESDE 1



PRUEBA EN EL PUNTO DE PARO 2

$(\alpha>0)\wedge(\beta>0)\supset((\alpha,\beta)=(\alpha,\beta))\wedge(\alpha>0)\wedge(\beta>0)$

ARBOL DE PRUEBA DESDE 2



PRUEBA EN LOS PUNTOS DE PARO 5, 2_1, 2_2

En 5 $((\alpha,\beta)=(\gamma,\delta))\wedge(\gamma>0)\wedge(\delta>0)\wedge(\neg(\gamma>\delta\vee\gamma<\delta))\supset\gamma=(\alpha,\beta)$

En 2_1 $((\alpha,\beta)=(\gamma,\delta))\wedge(\gamma>0)\wedge(\delta>0)\wedge(\gamma>\delta)\supset((\alpha,\beta)=(\gamma-\delta,\delta))\wedge(\gamma-\delta>0)\wedge(\delta>0)$

En 2_2 $((\alpha,\beta)=(\gamma,\delta))\wedge(\gamma>0)\wedge(\delta>0)\wedge(\gamma<\delta)\supset((\alpha,\beta)=(\gamma,\delta-\gamma))\wedge(\gamma>0)\wedge(\delta-\gamma>0)$

Figura 2: Verificación de la descripción LDP del algoritmo de Euclides

haciendo uso de los axiomas que definen el máximo común divisor de dos enteros positivos,

$$(a,a) = a$$
$$(a,b) = (b,a)$$
$$(a,b) = (a+b,b)$$

La descripción dada del algoritmo de Euclides es por tanto parcialmente correcta respecto a las aserciones consideradas.

3. VERIFICACION DE PROCESOS CONCURRENTES DESCRITOS EN LDP

El método de verificación de descripciones LDP propuesto en el apartado anterior puede generalizarse de un modo simple para verificar descripciones de varios procesos concurrentes. En este caso las aserciones no se asocian a puntos concretos de cada proceso sino que hacen referencia a un estado global de la descripción, es decir, las aserciones

deben probarse cuando cada uno de los procesos se encuentra en un punto determinado. -- Los puntos de paro de la relación de simulación hacen referencia, por tanto, a un conjunto de etiquetas, una de cada proceso.

Establecida la relación de simulación, la -- verificación consiste en construir un árbol de prueba para cada punto de paro que considere todos los posibles caminos de ejecución. En la generación del árbol de prueba puede - considerarse, como en la validación /10/, un tiempo variable, fijo o nulo para la ejecución de las distintas transiciones. En lo - que sigue se considerará la verificación con tiempo de ejecución de las transiciones fijo. La verificación sin considerar el tiempo de ejecución es un caso particular en el que el tiempo de ejecución de todas las transiciones se considera nulo. La generalización al caso de tiempo de ejecución variable es inmediata.

En la búsqueda de los caminos de ejecución - del árbol de prueba se utilizará un algoritmo semejante al propuesto en /10/ para la validación de protocolos. Allí se construía - un árbol, denominado de ejecución temporal, cuyos nudos están formados por el estado del sistema y las perturbaciones parciales dinámicas, V_i. El árbol de ejecución temporal - determina todos los caminos de ejecución que puede seguir el protocolo, sin tomar en cuenta la estructura de datos. En la generación del árbol de prueba han de observarse las -- mismas reglas que en la del arbol de ejecución temporal, más las que se derivan de tomar en cuenta la estructura de datos. A continuación se exponen las características del árbol de prueba y el procedimiento a seguir para generarlo.

Los nudos del árbol de prueba están formados por cuatro elementos

(a) Punto de control: conjunto de etiquetas que caracteriza la posición alcanzada en la ejecución simbólica de la descripción.

(b) Perturbaciones parciales dinámicas: son los conjuntos de transiciones parciales posibles de cada proceso y el tiempo que necesitan para realizarse.

(c) Vector de estado: formado por los valores simbólicos de las variables (expresiones formadas por los operadores del - LDP, constantes y símbolos iniciales).

(d) Lista de predicados: conjunto de condiciones sobre los símbolos iniciales para que el control pueda llegar al nudo.

Los nudos sucesores de uno dado, n, se obtienen siguiendo los pasos que a continuación - se indican

(a) Buscar en V_i^n la transición que requiere un tiempo menor para ejecutarse, t_{min}.

(b) Buscar los estados sucesores (puntos de control) de $\bar{e}$ al cabo del tiempo t_{min}.- Para ello

(1) Formar los conjuntos
$W_i=\{\bar{f} \mid (\bar{f},t_{min})\in V_i^n\}$
Los W_i contienen las transiciones -- parciales del estado $\bar{e}$ que puede ejecutar el proceso i en el tiempo t_{min}.

(2) Tomando un único elemento de cada W_i formar las transiciones compatibles de los procesos cuya ejecución conduciría a los nuevos estados $\bar{g}\in G$, - sucesores del $\bar{e}$. De estas transiciones sólo podrán ejecutarse aquellas que sean consistentes con el vector de estado y la lista de predicados. Los estados a que se llega ejecutando estas transiciones son $\bar{h}\in H\subset G$.

(c) Los nudos sucesores del n quedan caracterizados por

(1) Punto de control: $\bar{h}\in H$

(2) Perturbaciones parciales dinámicas - V_i^m, formadas a partir de las V_i^n del nudo, haciendo

(i) Para todo i tal que $e_i \neq h_i$, hacer $V_i^m = \{\emptyset\}$

(ii) Para todo i tal que $e_i = h_i$, hacer $V_i^m = V_i^n$ y decrementar todos - los tiempos de V_i^m en t_{min}.

(iii) Para todo i, añadir a V_i^n los elementos de $U_i(\bar{h})$ que no correspondan a estados siguientes del sistema ya - contenidos en algún elemento de V_i^m.

(3) Vector de estado: se obtiene ejecutando simbólicamente los procesos que intervienen en la transición de $\bar{e}$ a $\bar{h}$ considerada.

(4) Lista de predicados: obtenida como en (3).

La ejecución simbólica de una transición conduce al sistema del estado $\bar{e}$ al $\bar{h}$ comporta la ejecución simbólica de cada proceso desde el punto e_i al h_i. Esta se realiza tal y como se ha indicado en el apartado anterior, con la salvedad de las sentencias de asignación externa. La ejecución simbólica de una sentencia de salida en un proceso se realiza "simultáneamente" con la ejecución simbólica de una sentencia de entrada en otro proceso. El efecto de esta ejecución es la asignación de la expresión simbólica de la sentencia de salida a la variable de la entrada.

Sabiendo cómo construir un árbol de prueba el algoritmo de verificación es inmediato. Dado un protocolo descrito en LDP y una relación de simulación que refleje fielmente las propiedades que se desean probar, se genera para cada punto de paro un árbol de prueba. La raíz del árbol se inicializa de forma que

(a) Punto de control: punto de paro inicial $(\bar{e}_o)$

(b) Perturbaciones parciales dinámicas: $V_i = U_i(\bar{e}_o)$

(c) Vector de estado: valores simbólicos iniciales de todas las variables de la descripción.

(d) Lista de predicados: condiciones que han de cumplir los valores iniciales para que se cumpla la aserción que la relación de simulación asocia al punto de paro $\bar{e}_o$.

La verificación se lleva a cabo mediante el algoritmo siguiente:

(a) Definir una sucesión de conjuntos A(i) de nudos del árbol de prueba. El conjunto A(0) contiene el nudo inicial o raíz. Hacer i = 0.

(b) Tomar un nudo de A(i) cuyo conjunto de nudos sucesores no haya sido aún determinado e ir a (c). De no existir tal nudo mirar si $A(i+1) = \{\emptyset\}$. En este caso la verificación a partir del punto de paro que se está considerando ha concluído; en caso contrario, hacer i:=i+1 y volver al principio de (b).

(c) Determinar los nudos sucesores del nudo considerado y

(1) Si algún nudo sucesor es un punto de paro especificado en la relación de simulación, intentar probar la aserción asociada.

(2) Si algún nudo sucesor se encuentra ya en algún A(j), para j=1,...,i+1, no considerarlo más.

(3) Añadir a A(i+1) el resto de los nudos sucesores.

(4) Ir a (b).

El algoritmo de verificación propuesto permite deterctar diversos errores en la descrip-LDP de un protocolo, entre ellos:

(a) Si en el apartado (c) el nudo considerado carece de nudos sucesores se ha localizado un estado terminal o un bloqueo.

(b) Si en el apartado (c) punto (1) se prueba que la aserción es falsa, el protocolo no funcionará correctamente respecto a la relación de simulación dada.

(c) Si en el apartado (c) punto (2) el nudo considerado es igual a un antecesor suyo, se ha detectado un bucle.
Del exámen de la descripción se determina si se trata de un bucle normal o si por el contrario es un bloqueo temporal.

A continuación se ilustra el funcionamiento del algoritmo de verificación mediante un ejemplo. La figura 3 es la descripción LDP del protocolo de bit alternante /1,10/ en la que se ha supuesto que el medio puede introducir errores, pero no pérdidas de la información. En la figura 3 se muestra asimismo, la relación de simulación mediante la cual se expresan las propiedades que se desean verificar. En este ejemplo outdata(a) = indata(b) significará la igualdad de los elemen-

```
[S::SENDER || R::RECEIVER]

SENDER
1:ea:=ack:=seq:=false;pt:=1;
*[true →
  2:[¬ea∧ack=seq →3:datas:=indata(pt);pt:=pt+1;seq:=¬seq
    [] ea∨ack≠seq →skip];4:R:=(datas,seq);5:ack:=R;6:ea:=#
 ]


RECEIVER
1:exp:=false;ps:=1
*[true →
  2:(datar,seqr):=S;em:=#;
    4:[¬em∧seqr≠exp →5:outdata(ps):=datar;ps:=ps+1;exp:=¬exp
      [] em∨seqr=exp →skip];6:S:=exp
 ]
```

RELACION DE SIMULACION

1-1; outdata(0)=indata(0)

3-2; seq=exp∧ps=pt∧outdata(ps-1)=indata(pt-1)

Figura 3: Descripción LDP del protocolo de bit alternante y relación de simulación para la verificación.

tos de los vectores outdata e indata de índice menor o igual que a y b, respectivamente. La primera componente de la relación de simulación establece la igualdad del elemento 0 de estos dos vectores antes de la inicialización. La segunda componente expresa que --- cuando el proceso emisor está en situación - de tomar un nuevo mensaje de indata para --- transmitir y el proceso receptor espera la - recepción de un mensaje, deben cumplirse las condiciones seq = exp, ps= pt y outdata(ps-1) = indata(pt-1).

La figura 4 recoge la descripción gráfica -- del protocolo a partir de la cual se construyen fácilmente los dos árboles de prueba que requiere la verificación, uno para cada punto de paro.

El primer árbol de prueba se encuentra en la figura 5. La raíz (nudo 1) fija el punto de control en la primera sentencia de cada proceso y establece el tiempo de ejecución 0 para las transciciones de ambos. Su vector de estado adopta valores simbólicos arbitrarios y la lista de predicados contiene un único - elemento que restringe los valores simbólicos x_o e y_o para que se cumpla la aserción - asociada a este punto de paro (los valores - simbólicos de outdata (m) e indata (n) son - respectivamente, y_m y x_n). La generación del árbol termina en el nudo 3, al ser su punto de control el 3-2, punto de paro especificado en la relación de simulación. En este -- punto debe probarse la aserción.

seq = exp ∧ pt = ps∧outdata(ps-1)=indata(pt-1).

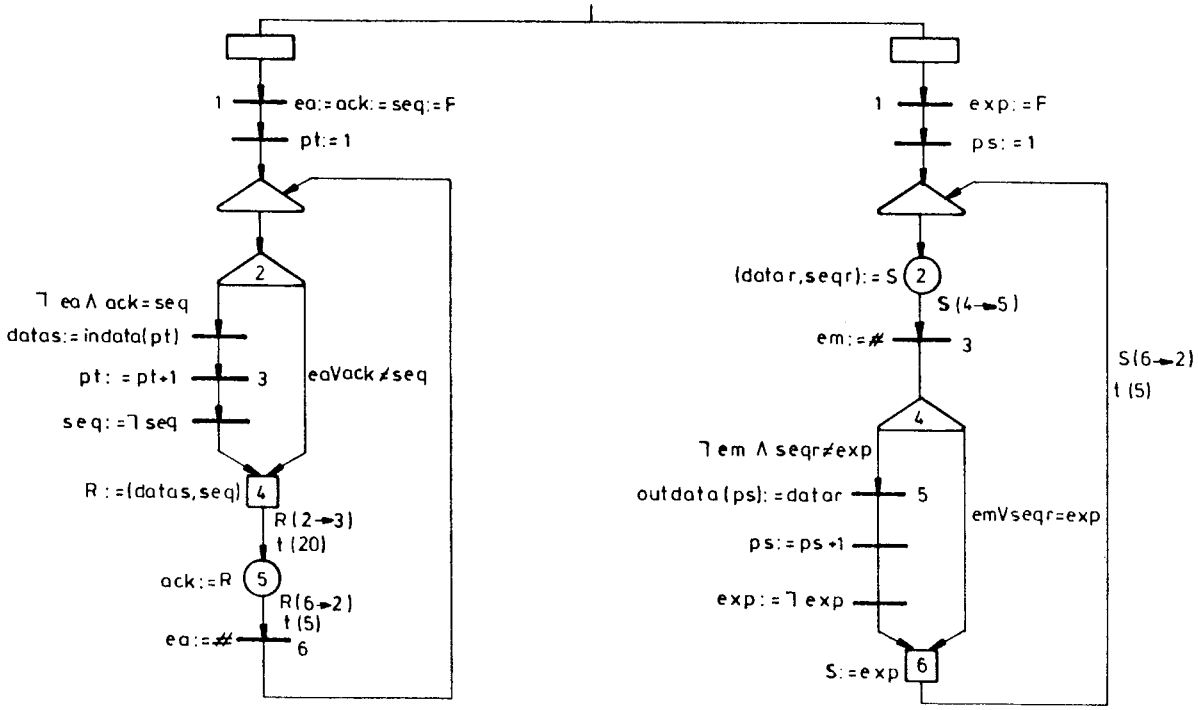


Figura 4: Descripción gráfica LDP del protocolo de la figura 3

que en este caso se convierte en el predicado

$$y_o = x_o \supset F = F \wedge 1 = 1 \wedge y_o = x_o ,$$

trivialmente cierto.

Las figuras 6 , 7 muestran el árbol de prueba correspondiente a la segunda componente de la relación de simulación. El punto de control de la raíz es el 3-2, en el que únicamente el proceso S puede evolucionar. Las variables del vector de estado toman valores simbólicos y la lista de predicados impone a éstos las condicones que hacen que se cumpla la aserción correspondiente a este punto. Para facilitar el seguimiento de la construcción de este árbol se ha escrito junto a los nudos de la figura 6 el punto de control del nudo y el tiempo necesario para ejecutar la siguiente transición y, junto a éstas, las sentencias que ejecuta simbólicamente cada proceso.

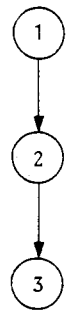
En la figura 6 se observa que el subárbol que comienza en el nudo 5, correspondiente a la recepción errónea de un mensaje, conduce a los estados 15 ó 16. Ambos estados son iguales al estado 2, por lo que esta situación conduce a la retransmisión del mensaje. El subárbol que se inicia en el nudo 4 corresponde a la recepción correcta del mensaje. En la transición del nudo 6 al 8 el mensaje se pasa al vector outdata y en la de 8 a 10 se envía el ack. Si el ack llega correctamente se llega al nudo 17 que es un punto de paro. En él ha de probarse la aserción (seq=exp)∧(pt=ps)∧(outdata(ps-1)=indata(pt-1)) que se traduce en el predicado siguiente

$$(c=i) \wedge (d=j) \wedge (y_{j-1}=x_{d-1}) \supset (c=i) \wedge (d+1=j+1) \wedge (y_j=x_d) ,$$

que, dado que el elemento j de outdata se ha hecho igual al d de indata, es siempre cierto. El subárbol que comienza en el nudo 14 corresponde a la recepción incorrecta del ack. Tras retransmitir el mensaje se llega al nudo 22 ó 23 que son idénticos al 8, a partir del cual se retransmite el ack.

De la verificación del protocolo de bit alternante se concluye que su comportamiento es cíclico, carece de bloqueos y tiene dos bloqueos temporales. Efectivamente, si la línea que transmite los mensajes de S a R produce siempre errores, el protocolo sigue los ciclos 2-3-5-7-9-10-2 ó 2-3-5-7-4-12-2. Igualmente, si la línea que transmite el ack de R a S introduce sistemáticamente errores, el protocolo entra en los ciclos 8-10-14-18-

ARBOL



DESCRIPCION DE LOS NUDOS

Nudo	Estado	ea	ack	seq	pt	datas	datar	seqr	em	exp	ps
1	1 1 0 0	a	b	c	d	e	f	g	h	i	j
2	2 2 0 -	F	F	F	1	e	f	g	h	F	1
3	3 2 0 -	F	F	F	1	e	f	g	h	F	1

LISTA DE PREDICADOS IGUAL EN LOS TRES NUDOS

$$\alpha = y_0 = x_0$$

PRUEBA

$$y_0 = x_0 \supset F = F \wedge 1 = 1 \wedge y_0 = x_0$$

Figura 5: Arbol de prueba y condición de verificación correspondiente al primer punto de paro de la relación de simulación.

19-20-8, 8-10-14-18-19-21-8. Fuera de estos casos triviales de funcionamiento incorrecto, el protocolo se comporta correctamente respecto a la relación de simulación dada, lo que garantiza que el protocolo transfiere una única copia de cada mensaje de indata a outdata y en el debido orden.

4. OBSERVACIONES Y CONCLUSIONES

El algoritmo de verificación propuesto permite generar sistemáticamente las condiciones de verificación de una descripción LDP respecto a una relación de simulación dada. La prueba de estas condiciones pertenece al cálculo de predicados de primer orden, que carece de la propiedad de ser decidible (no existe ningún procedimiento de decisión que permita determinar si un predicado de primer orden es o no válido). Esto supone que puede resultar imposible demostrar si una condición es válida o no. Por tanto, el método puede fallar de dos maneras en la verificación; no siendo capaz de probar la corrección de una descripción correcta o la incorrección de una descripción incorrecta. Si el algoritmo es capaz de decidir si una descripción LDP determinada es correcta (o incorrecta), puede afirmarse sin lugar a dudas que la descripción es correcta (o incorrecta) respecto a la relación de simulación considerada. Esta limitación es común a todos los sistemas de verificación de programas basados en el cálculo de predicados de primer orden.

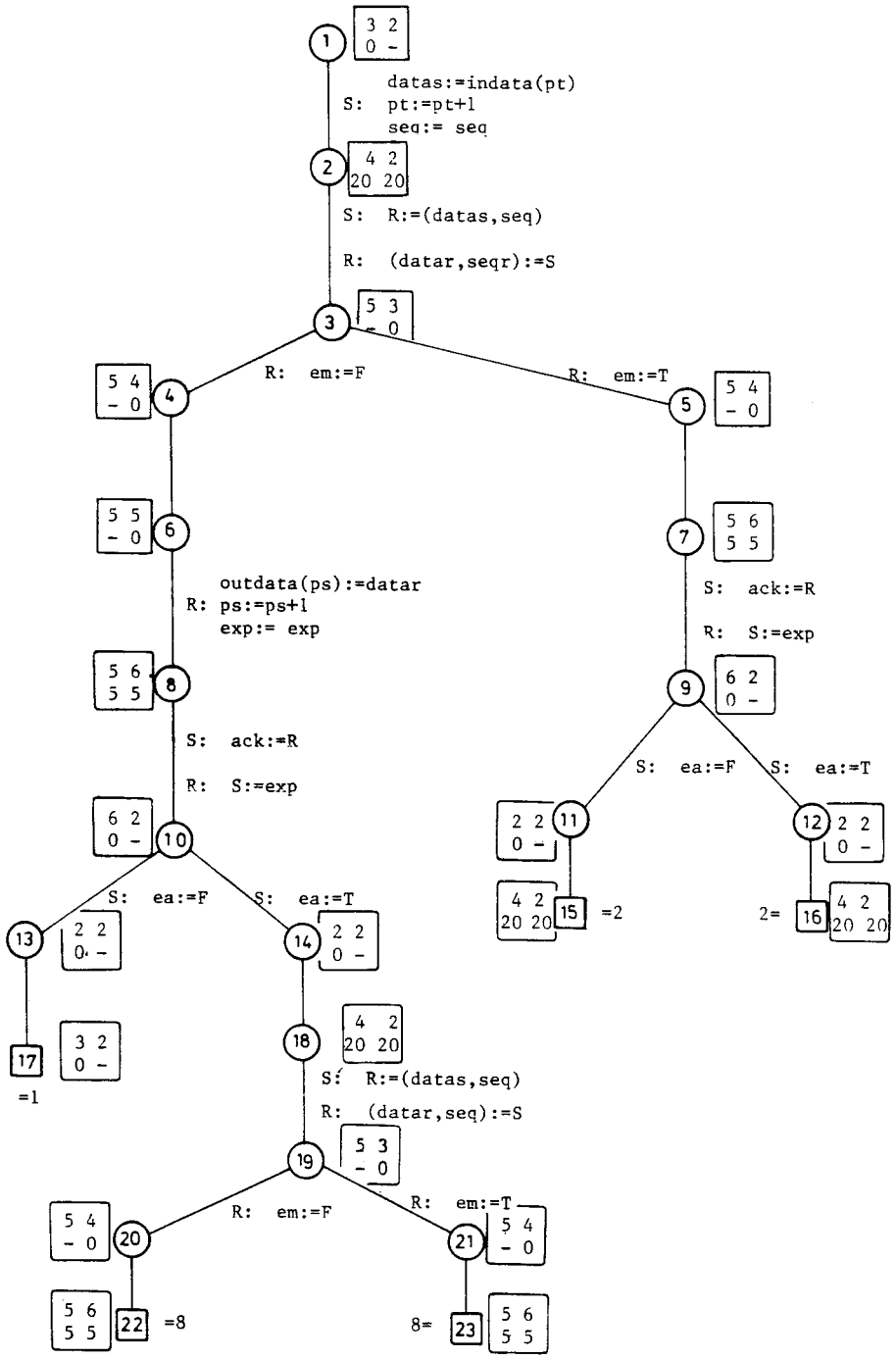


Figura 6: Arbol de prueba del protocolo correspondiente a la segunda componente de la relación de simulación.

Una limitación adicional es la posibilidad - de que el algoritmo no termine. Efectivamente, si las variables de una descripción LDP pueden tomar valores arbitrariamente grandes, resulta imposible garantizar que la verificación termine.

Estas limitaciones teóricas impiden el poder afirmar que el algoritmo de verificación permita siempre probar la corrección o incorrección de procesos concurrentes descritos en - LDP. Sin embargo, si puede garantizarse la limtación en el credimiento del número de estados del árbol de prueba y la complejidad - de las condiciones de verificación son moderadas, la prueba podrá llevarse a cabo. Tal como se avanzó en el apartado de introduc--- ción,estas condiciones suelen darse en la verificación de protocolos de comunicación y - en muchos otros procesos concurrentes de características análogas.

Nudo	Estado	ea	ack	seq	pt	datas	datar	seqr	em	exp	ps	
1	3 2 0 -	a	b	c	d	e	f	g	h	i	j	
2	4 2 20 20	a	b	¬c	d+1	x_d	f	g	h	i	j	
3	5 3 - 0	a	b	¬c	d+1	x_d	x_d	¬c	h	i	j	
4	5 4 - 0	a	b	¬c	d+1	x_d	x_d	¬c	F	i	j	
5	5 4 - 0	a	b	¬c	d+1	x_d	x_d	¬c	T	i	j	
6	5 5 - 0	a	b	¬c	d+1	x_d	x_d	¬c	F	i	j	
7	5 6 5 5	a	b	¬c	d+1	x_d	x_d	¬c	T	i	j	
8	5 6 5 5	a	b	¬c	d+1	x_d	x_d	¬c	F	¬i	j+1	$y_j=x_d$
9	6 2 0 -	a	i	¬c	d+1	x_d	x_d	¬c	T	i	j	
10	6 2 0 -	a	¬i	¬c	d+1	x_d	x_d	¬c	F	¬i	j+1	
11	2 2 0 -	F	i	¬c	d+1	x_d	x_d	¬c	T	i	j	
12	2 2 0 -	T	i	¬c	d+1	x_d	x_d	¬c	T	i	j	
13	2 2 0 -	F	¬i	¬c	d+1	x_d	x_d	¬c	F	¬i	j+1	
14	2 2 0 -	T	¬i	¬c	d+1	x_d	x_d	¬c	T	¬i	j+1	
15	4 2 20 20	F	i	¬c	d+1	x_d	x_d	c	T	i	j	
16	4 2 20 20	T	i	¬c	d+1	x_d	x_d	c	T	i	j	
17	3 2 0 -	F	¬i	¬c	d+1	x_d	x_d	¬c	F	¬i	j+1	
18	4 2 20 20	T	¬i	¬c	d+1	x_d	x_d	¬c	T	¬i	j+1	
19	5 3 - 0	T	¬i	¬c	d+1	x_d	x_d	¬c	T	¬i	j+1	
20	5 4 - 0	T	¬i	¬c	d+1	x_d	x_d	¬c	F	¬i	j+1	
21	5 4 - 0	T	¬i	¬c	d+1	x_d	x_d	¬c	T	¬i	j+1	
22	5 6 5 5	T	¬i	¬c	d+1	x_d	x_d	¬c	F	¬i	j+1	
23	5 6 5 5	t	¬i	¬c	d+1	x_d	x_d	¬c	T	¬i	j+1	

Figura 7: Descripción de los nudos de la figura 6

La verificación manual de descripciones de - cierta complejidad resulta inabordable. Puede pensarse en la construcción de un sistema de verificación automático, o semiautomático (interactivo), que genere las condiciones de verificación y las pruebe. La realización - de este tipo de programas constituye un tema permanente de investigación. Las dos limitaciones fundamentales de estos sistemas automáticos son la dificultad de encontrar aserciones inductivas adecuadas y la falta de potencia de los probadores de teoremas que, en muchos casos, son incapaces de demostrar las condiciones de verificación generadas /20/.

Resulta interesante comparar el sistema de - descripción y verificación desarrollado con otros métodos existentes. La figura 8 recoge, en términos similares a los de /22/ las características de las técnicas más impor-- tantes.

Autor	Formalismo de modelación	Técnicas de análisis	Aspectos considerados	Propiedades estudiadas	Aspectos complicados	Trabajo adicional necesario
DANTHINE BREMER	Modelo local de estados más algoritmo	Caminos compatibles	Control	Bloqueos	Definición del protocolo	Función de transferencia, bloqueo temporal, asimetría, tiempo
BRAND JOYNER	Algoritmos	Ejecución simbólica, aserciones	Función de transferencia	Bloqueos, bloqueos temporales, f. transf	Definición de aserciones	Inicializaciones, medios complejos, tiempo
RUDIN WEST ZAFIROPULO	Modelo local de estados	Caminos compatibles, estado global	Control	Bloqueos, compleción	Definición del protocolo	Medios complejos, bloqueo temporal, tiempo, f. transf
BOCHMANN	Modelo local de estados con variables y algoritmos	Estado global aserciones y estados adjuntos	Ambos	Bloqueos, bloqueo temporal, vida	Definición protocolo, prueba aserciones	Automatización, medios complejos, tiempo
HAJEK	Algoritmos	Estado global	Ambos	Bloqueos temporales, terminación	Definición algoritmos, aserciones	Medios complejos, control nº estados, tiempo
HARANGOZO	Gramáticas formales		Ambos		Definición gramática	Verificación, tiempo
GOUDA	Modelo local de estados	Estado global, caminos compatibles	Control	Bloqueos, acotación	Definición protocolo, prueba	Medios y protocolos complejos, automatizac.,tiempo
VALIDACION PROPUESTA EN /10/	Algoritmos en LDP	Estado global, análisis de alcanzabilidad	Control	Bloqueos, bloqueo temporal, comportamiento cíclico,tiempo		Automatización
VERIFICACION PROPUESTA	Algoritmos en LDP	Ejecución simbólica, aserciones	Ambos	Bloqueos, bloqueo temporal, comp. cíclico, tiempo, f. de transferencia	Definición y prueba de aserciones	Automatización

Figura 8: Comparación de las técnicas de verificación más significativas (referenciadas en /9/)

De la observación de la figura puede concluirse que el sistema propuesto en esta tésis se compara favorablemente con las técnicas mas significativas que existen en la actualidad.

5. REFERENCIAS

/1/ BARTLETT, K.A. et al.: "A Note on Reliable Full-Duplex Transmission over Half-Duplex Links". Comm. ACM, v.12, nº5, -- May 1969, pp. 260-261.

/2/ BIRMAN, A., JOYNER, W.H.: "A Problem - Reduction Approach to Proving Simulation Between Programs". IEEE Trans. on Soft Eng., v.SE-2,nº2, June 1976, pp. - 87-96.

/3/ BRAND, D., JOYNER, W.H.: "Verification of Protocols Using Symbolic Execution". Proc. Symp. on Computer Network Protocols, Liege 1978, pp. F2-1/F2-7.

/4/ BREMER; J.: "Representation Axiomatique d'un Protocol Description du Programme Reduction". Univ. Liege, SART 76/19/10, Sept. 1976.

/5/ BREMER, J.: "Verification de la Logique d'un Protocol. Descripcion du Programm Verify". Univ. Liege, SART 77/03/10, -- Jan. 1977.

/6/ DANTHINE, A.S., BREMER, J.: "Modelling and Verification of End-to-End Transport Protocols". Proc. Symp. on Computer Network Protocols, Liege 1978. pp. F5-1/F5-12.

/7/ DEUTSCH, P.: "An Interactive Program -- Verifier". Ph.D dissertation, Univ. California, Berkeley, 1973.

/8/ FLOYD, R.W.: "Assigning Meaning to Programs". Proc. Symp. in Aplied Mathematics, v 19, 1967, pp. 19-32.

/9/ GARCIA HOFFMANN,M.: "Tecnicas para la - Descripción Formal y Verificación de -- Protocolos". Qüestió. v 3. nº4, 1979, - pp. 219-229.

/10/ GARCIA HOFFMANN,M.: "Un Método para la Especificación y Validación de la Estructura Lógica de los Protocolos de Comunicación". Qüestió, v.4, nº 1, 1980, pp. 23-45.

/11/ GARCIA HOFFMANN,M.: "Hardware Implementation of Communication Protocols: A -- Formal Approach". Proc. of the Seventh. Inter. Symp. on Computer Architecture, La Baule, May 1980, pp. 253-263.

/12/ GOOD, D.I., et al.: "An Interactive Program Verification Method". IEEE Trans. on Soft. Eng., v. SE-1, Mar.1975, pp.-- 59-67.

/13/ HAJEK, J.: "Automatically Verified Data Transfer Protocols". Proc. of ICCC - 78, 1978, pp. 749-756.

/14/ HANSEN, P.B.: "Distributed Processes: - A Concurrent Programming Concept". Comm. ACM, v. 21, nº 11, Nov. 1978, pp. 934-941.

/15/ HANTLER, S.L., KING, J.C.: "An Introduction to Proving the Correctness of - Programs". ACM Computing Surveys, v.8,- nº 3, Sept. 1976, pp. 331-353.

/16/ Von HENKE, F.W., LUCKHAM, D.C.: "A Methodology for Verifying Programs". Proc. Int. Joint Conf. Reliable Software, Apr. 1975, pp.156-164.

/17/ HOARE, C.A.R.; "Communicating Sequential Processes". Comm. ACM, v.21, nº 8. Aug. 1978, pp.666-677.

/18/ KING, J.C.: "A Program Verifier". Proc. IFIP, Aug. 1971, pp. 235-249.

/19/ KING, J.C.: "Symbolic Execution and --- Program Testing". Comm. ACM, v.19, nº7. July 1976, pp. 385-394.

/20/ MANNA, Z., WALDINGER, R.: "The Logic of Computer Programming". IEEE Trans. on - Soft. Eng., v.SE-4, nº 3. May 1978, pp. 199 - 229.

/21/ NAUR, P.: "Proof of Algorithms by General Snapshots". BIT, v.6, 1966, pp. 310 -316.

/22/ SUNSHINE, C.A.: "Survey of Protocol Definition and Verification Techniques". Proc. Symp. on Computer Network Protocols, Liege, 1978, pp. F1-1/F1-4.

/23/ WALDINGER, R.J., LEVITT, K.N.: "Reasoning About Programs". Artificial Intelligence, v.5, Fall 1974, pp. 235-316.