Una aproximación lógica a la verificación de propiedades de programas lógico-funcionales

José Miguel Cleva, Javier Leach, Francisco J. López-Fraguas

Departamento de Sistemas Informáticos y Programación.
Universidad Complutense de Madrid
[jcleva, leach, fraguas]@sip.ucm.es

Resumen Este trabajo es un resumen de [2,3], donde se aborda el tema de la verificación de propiedades de programas lógico-funcionales desde un punto de vista lógico. Para ello se parte de un semántica bien conocida para este tipo de lenguajes como es la proporcionada por el marco lógico *CRWL*, y se asocia a cada *CRWL*-programa un programa lógico que servirá como base para la demostración de las propiedades del programa original. Se analizan algunos inconvenientes que se encuentran en la práctica con este enfoque, y se proponen un par de refinamientos que mejoran la efectividad de la aproximación.

1. Introducción

En los lenguajes de programación lógico-funcional como Curry [5] y Toy [6], existe la posibilidad de definir funciones perezosas mediante sistemas de reescritura no confluentes y no terminantes lo que, en particular, impide poder llevar a cabo un razonamiento puramente ecuacional sobre dichos sistemas.

La lógica *CRWL* (Constructor based ReWriting Logic) [4] proporciona una lógica alternativa para este tipo de lenguajes. A partir de ella, la semántica de un programa viene dada por una relación de reducción $e \to t$ entre expresiones y términos construidos parciales.

En este trabajo buscamos una base lógica para probar propiedades de programas *CRWL*. Las propiedades de interés son aquellas válidas en el modelo inicial que se corresponde generalmente bien con el modelo "pretendido" del programa.

La idea básica es asociar a cada programa *CRWL* un programa lógico (por tanto una teoría de primer orden) que exprese la relación $e \to t$. Así, intentaremos probar propiedades en esta teoría de primer orden por medio de mecanismos de deducción lógica.

2. La lógica CRWL y su traducción como programa lógico

Consideramos una signatura $\Sigma = DC_\Sigma \cup FS_\Sigma$, donde $DC_\Sigma = \bigcup_{n \in \mathbb{N}} DC^n_\Sigma$ es un conjunto de símbolos de constructor y $FS_\Sigma = \bigcup_{n \in \mathbb{N}} FS^n_\Sigma$ es un conjunto de símbolos de función. También se considera un conjunto numerable de variables

$\mathcal{V}$. Para poder razonar sobre expresiones parciales, se considera la extensión de la signatura $\Sigma_\perp$ con una nueva constante $\perp$. Se denominan *c-términos* aquellos que sólo usan constructores o variables y *c-términos parciales*($CTerm_\perp$) si además se considera $\perp$. Los c-términos parciales sin variables se denominan *términos cerrados*. Un programa *CRWL* será un conjunto finito de reglas de la forma $f(\bar{t}) \to e$ donde f es un símbolo de función, $\bar{t}$ es una tupla de c-términos donde cada variable aparece una sola vez (i.e, lineal), y e es una expresión cualquiera.

Las reglas del cálculo *CRWL* que hemos considerado son:

$$\textbf{(BT)}\ \frac{}{e \to \perp} \quad \textbf{(DC)}\ \frac{e_1 \to t_1, ..., e_n \to t_n}{c(e_1, ..., e_n) \to c(t_1, ..., t_n)}\ c \in DC^n, t_i \in CTerm_\perp$$

$$\textbf{(FR)}\ \frac{e_1 \to t_1, ..., e_n \to t_n \quad e \to t}{f(e_1, ..., e_n) \to t} \qquad \text{si } t \not\equiv \perp, f(t_1, ..., t_n) \to e \in [P]_\perp$$

donde el conjunto $[P]_\perp$ en (**FR**) está formado por todas las instancias $(l \to r)\sigma$ de reglas de P, donde $l \to r \in P$ y σ reemplaza variables por c-términos parciales. Este cálculo es una simplificación del original [4], pero esencialmente equivalente. Usamos $P \vdash_{CRWL} e \to t$ para indicar la *CRWL*-derivabilidad de $e \to t$ a partir del programa P. Como ejemplo, las reglas $coin \to 0$, $coin \to s(0)$, $0 + Y \to Y$, $s(X) + Y \to s(X + Y)$, $double(X) \to X + X$ forman un *CRWL*−programa no confluente, para el que se tiene $P \vdash_{CRWL} double(coin) \to 0$ y $P \vdash_{CRWL} double(coin) \to s(s(0))$, pero $P \nvdash_{CRWL} double(coin) \to s(0)$, lo que refleja una semántica de *call time choice* para el indeterminismo [4].

Sea P un programa *CRWL* con signatura $\Sigma = DC \cup FS$. El programa lógico P_L asociado se obtiene mediante la transformación directa a cláusulas de las reglas de *CRWL* para cada constructor y regla de función del programa P. Se comprueba que la reducciones $e \to t$ válidas en el programa lógico P_L son las mismas que las que se deducen por medio de *CRWL* para el programa original.

Las propiedades que consideramos son fórmulas φ de primer orden sobre la relación de reducción $e \to t$. Consideramos las siguientes teorías en lógica de primer orden: $T_{P_L} = \{\varphi \mid P_L \models \varphi\}$, $T_{Comp(P_L)} = \{\varphi \mid Comp(P_L) \models \varphi\}$ y $T_{M_P} = \{\varphi \mid M_{P_L} \models \varphi\}$ donde $Comp(P_L)$ es la compleción de Clark de P_L y M_{P_L} es el modelo mínimo de Herbrand de P_L. Las propiedades que nos interesan son aquéllas que son válidas en el modelo mínimo de P_L, es decir, las que pertenecen a T_{M_P}. Como M_{P_L} es modelo tanto del programa lógico como de su compleción, tenemos la cadena de inclusiones: $T_{P_L} \subseteq T_{Comp(P_L)} \subseteq T_{M_P}$. Así, podemos usar el programa lógico y su compleción para obtener propiedades de M_{P_L} mediante deducción en lógica de primer orden. Por ejemplo, en el programa anterior se tiene $P_L \models coin \to 0$ y $Comp(P_L) \models coin \not\to s(0)$. De todos modos estas dos aproximaciones no permiten demostrar muchas propiedades interesantes de M_{P_L} que son de naturaleza inductiva, como sucede en el ejemplo anterior con $\forall X, T. X \to T \Rightarrow X + 0 \to T$. Por ello extendemos la compleción del programa lógico con los axiomas de inducción estructural. Llamando a esta extensión $CompInd(P_L)$ y $T_{CompInd(P_L)} = \{\varphi \mid CompInd(P_L) \models \varphi\}$, obtenemos la siguiente cadena de inclusiones (estrictas, como no es difícil comprobar): $T_{P_L} \subset T_{Comp(P_L)} \subset T_{CompInd(P_L)} \subset T_{M_P}$

Hemos trasladado esta aproximación a diversos sistemas de demostración como ITP [1], LPTP [8] e Isabelle [7] y hemos analizado las distintas dificultades que se plantean a la hora de verificar algunas de las propiedades que hemos considerado. Muchos de los problemas detectados son comunes en todas las herramientas consideradas. En primer lugar, cuando se tratan propiedades sobre reducciones inalcanzables como, por ejemplo, $double(coin) \twoheadrightarrow s(0)$, se debe explorar todo el espacio de posibles reducciones que se pueden dar a partir de una determinada expresión, esto hace que las pruebas crezcan de una forma considerable. Por otra parte surgen problemas dependientes del sistema que se considera, ya que al traducir las distintas teorías a los sistemas en los que demostrar las propiedades, se produce en muchos casos una pérdida de automatismo en las pruebas que se realizan.

Para resolver alguno de estos problemas hemos considerado *CRWL'* una versión de *CRWL* en la que se elimina la regla **BT**. Esta nueva versión tiene como ventaja que se generan menos reducciones que en la versión original y además hace que el razonamiento sobre términos sea determinista, lo que, en particular, podría ser interesante ya que podría permitir una separación de las partes determinista y no determinista del programa. Se comprueba que si consideramos reducciones a términos totales, ambos cálculos son equivalentes.

3. Axiomatización de la derivabilidad

Como dijimos anteriormente en la cadena de inclusiones de las distintas teorías, las inclusiones son estrictas. En particular, se tiene $T_{CompInd(P_L)} \subsetneq T_{M_P}$. Consideremos la función con la única regla `loop` $\rightarrow$ `loop`. Es fácil ver que $loop \twoheadrightarrow 0$ es válido en M_{Loop_L}, pero $CompInd(Loop_L) \not\models loop \twoheadrightarrow 0$.

Para demostrar propiedades como la anterior, una posibilidad es poder razonar sobre las derivaciones que se obtienen en *CRWL*. De este modo vamos a asociar al programa *CRWL* original un programa lógico $Der(P)$ que defina la relación de derivabilidad por medio de una relación ternaria $d \vdash e \rightarrow t$, donde d representa términos de primer orden para las *CRWL*-derivaciones. Consideraremos tanto la compleción como la compleción inductiva de $Der(P)$, teniendo en cuenta que, en este caso, hay dos tipos de construcciones: expresiones y derivaciones, por lo que podremos razonar por inducción sobre las derivaciones.

El siguiente resultado relaciona $Der(P)$ con el cálculo original.

Proposición 1 *Para cualquier programa CRWL P, expresión e y término t: $Der(P) \models \exists D.D \vdash e \rightarrow t \Leftrightarrow P \vdash_{\mathrm{CRWL}} e \rightarrow t$ y $Comp(Der(P)) \models \nexists D.D \vdash e \rightarrow t \Rightarrow P \nvdash_{\mathrm{CRWL}} e \rightarrow t$*

Si consideramos la traducción $\hat{\varphi}$ de una fórmula φ de P_L donde cada aparición de $e \rightarrow t$ se sustituye por $\exists D.D \vdash e \rightarrow t$, se obtiene el siguiente resultado:

Proposición 2 *Sea φ una fórmula de primer orden sobre la relación $\rightarrow$, entonces $M_{P_L} \models \varphi \Leftrightarrow M_{Der(P)} \models \widehat{\varphi}$*

Análogamente se tiene el resultado para $Comp(P_L)$ y $Comp(Der(P))$. Esto quiere decir que incorporar las derivaciones no añade nada nuevo al programa lógico ni a su compleción, la diferencia se encuentra cuando consideramos la compleción inductiva ya que utilizando ahora un razonamiento inductivo sobre las derivaciones, se puede demostrar la propiedad $loop \nrightarrow 0$.

4. Conclusiones y trabajo futuro

En estos trabajos hemos dado una primera aproximación a la verificación de propiedades de lenguajes lógico-funcionales. Nos hemos basado en el marco *CRWL* y, para estos programas, hemos considerado aquellas propiedades que son válidas en el modelo inicial. Para probar estas propiedades hemos dado una traducción del programa original a un programa lógico a partir del cual demostrar propiedades válidas en el modelo mínimo del mismo por medio de mecanismos de deducción en lógica de primer orden. Además, hemos trasladado esta aproximación a diversos sistemas de demostración automática, analizando las limitaciones prácticas de este enfoque. Hemos mejorado la eficiencia por medio de dos refinamientos: *CRWL'* y la axiomatización de la derivabilidad que permite probar más propiedades mediante deducción en lógica de primer orden.

Como trabajo futuro y desde el punto de vista práctico, intentaremos trasladar este enfoque a otros sistemas de demostración automática y, además, extenderemos el repertorio de ejemplos considerando otras propiedades no triviales. Desde el punto de vista teórico, queremos mejorar la aproximación en dos sentidos, considerando lógica de primer orden con géneros ordenados y considerando distintas extensiones de *CRWL* que incorporen orden superior o fallo.

Referencias

1. M. Clavel. *The ITP tool*. Proc. of the 1st Workshop on Logic and Language, Kronos, 55–62, 2001. Sistema en `http://www.ucm.es/info/dsip/clavel/itp`.
2. J.M. Cleva, J. Leach, F.J.López-Fraguas. *A logical approach to the verification of functional-logic programs*. Proc. of the 13th International Workshop of Functional and Logic Programming, RWTH Aachen Technical Report, pp. 33–48, 2004.
3. J.M. Cleva, J. Leach, F.J.López-Fraguas. *A logic programming approach to the verification of functional-logic programs*. Proc. Principles and Practice of Declarative Programming (PPDP'04), ACM Press, pp. 9–19, 2004.
4. J.C. González-Moreno, M.T. Hortalá-González, F.J. López-Fraguas, M. Rodríguez-Artalejo. *An Approach to Declarative Programming Based on a Rewriting Logic*. Journal of Logic Programming 40(1), pp. 47–87, 1999.
5. M. Hanus (ed.), *Curry: an Integrated Functional Logic Language*, Version 0.8, April 15, 2003. `http://www-i2.informatik.uni-kiel.de/~curry/`.
6. F.J. López Fraguas, J. Sánchez Hernández. *TOY: A Multiparadigm Declarative System*. Proc. RTA'99, Springer LNCS 1631, pp 244–247, 1999.
7. T.Nipkow, L.C. Paulson, M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higer-Order Logic*. Springer LNCS 2283, 2002.
8. R.F. Stäerk. *The theoretical foundations of LPTP (A logic program theorem prover)*. Journal of Logic Programming 36, pp. 241–269, 1998.