Un sistema de inferencia de tipos extendidos con información de terminación

Jorge F. Salas O. *
Universidad Central de Venezuela
Escuela de Computación
Apartado 47002, Caracas 1041, Venezuela
E-mail: jsalas@usb.ve

Ascánder Suárez
Universidad Simón Bolívar
Dpto. de Computación
Apartado 89000, Caracas 1080, Venezuela
E-mail: suarez@usb.ve.

Resumen

Se presenta un sistema de tipos extendidos que permite garantizar la terminación de un conjunto no trivial de programas que contemplan definiciones de función por casos, aplicaciones, variables y estructuras de datos.

La consistencia semántica del sistema se garantiza por medio de un teorema que afirma que si la decoración del tipo de una expresión es la de terminación entonces existe una cota superior para el número de pasos de evaluación que reducen la expresión a su forma normal.

1 Introducción

Como es bien conocido, el problema de terminación de programas, en el caso general, es indecidible [Cut80]. De aquí que cualquier intento de abordarlo necesariamente debe partir

*Este trabajo ha sido financiado por el Consejo de Desarrollo Científico y Humanístico de la Universidad Central de Venezuela. Proyecto No. 03.13.3426.95.

del supuesto de que solamente se podrá resolver parcialmente el mismo.

En este trabajo abordaremos el problema de terminación de programas no recursivos con estructuras de datos considerando un lenguaje de expresiones que es un subconjunto no trivial del lenguaje ML [MTH90, W+90].

Los programas no recursivos con tipos terminan. Sin embargo, agregando estructuras de datos se pierde la terminación en general como se muestra en el siguiente programa CAML:

$$\begin{array}{l}\textbf{type}\ \ t = T\ \textbf{of}\ \ t \rightarrow t;;\\ \textbf{let}\ \ apply = \textbf{fun}\ \ (T\ f) \rightarrow \textbf{fun}\ \ a \rightarrow f\ a;;\\ \textbf{let}\ \ \Delta = T(\textbf{fun}\ \ x \rightarrow apply\ x\ x);;\\ apply\ \Delta\ \Delta;;\end{array}$$

Se puede verificar (por ejemplo aplicando la semántica propuesta en la sección 2.7) que luego de varios pasos de reducción, la expresión *apply* Δ Δ se evalúa en sí misma. Este ejemplo muestra que, en general, los programas no recursivos no pueden ser optimizados a tiempo de compilación llevándolos a forma normal.

El objetivo del presente trabajo es diseñar un sistema de inferencia de que permita determinar estáticamente si un programa no recursivo con estructuras de datos efectivamente terminará al ser ejecutado con cualquier dato de entrada válido para su especificación.

El método empleado está fundamentado en los sistemas de inferencia de tipos propuesto por Hindley [Hin69] y por Milner [Mil78, Dam85] y más concretamente, en los sistemas de tipos extendidos de Guzmán y Suárez [GS94b]. En particular, definimos un sistema de tipos extendidos que decora los tipos de los programas con información acerca de su terminación. Se demuestra que todo programa cuyo tipo esté decorado con una etiqueta que indique la propiedad de terminación efectivamente termina.

Antecedentes

Los sistemas de tipos han sido utilizados desde los años 40 para demostrar propiedades de programas. Por ejemplo, en el Cálculo Lambda de Church [Chu41] se usaban para restringir el conjunto de frases que correspondían a cálculos que terminan. Más recientemente, se han utilizado en lenguajes como Algol y Pascal para que el programador muestre explícitamente la coherencia entre sus procedimientos y sus datos, sin atacar el problema de terminación.

Normalmente, los sistemas de tipos se utilizan para demostrar propiedades de corrección en los programas, y tienen como característica la de estar subyacentes a éstos, de manera que el programador no está obligado a hacer él mismo el cálculo de tipos sino que esta tarea es dejada a los compiladores.

Lucassen y Gifford [LG88] han utilizado el algoritmo de reconstrucción de tipos de Hindley-Milner como soporte para un análisis de propiedades en los lenguajes de programación. Más recientemente, los sistemas de tipos han sido empleados como herramientas para la demostración de otras propiedades de los programas. En particular, los trabajos de Guzmán y Suárez en analisis de vida, mutación de estados y excepciones [Guz93, GS94a].

Estas investigaciones han demostrado la viabilidad de la metodología de tipos extendidos para el análisis de propiedades de los programas tal como el objetivo planteado en el presente trabajo.

2 Definiciones sintácticas y semánticas

2.1 Sintaxis de las expresiones de terminación

Sea UV un conjunto de etiquetas de terminación (variables booleanas). La sintaxis del lenguaje de expresiones de terminación U está dada por las producciones:

$$U ::= \quad b \mid u \mid u_1 \wedge u_2 \qquad \text{donde } b \in Bool,\ u \in UV,\ u_1, u_2 \in U.$$

En el caso especial cuando la expresión de terminación sea la constante booleana t (verdadero), se interpretará que el sistema está indicando que la expresión es de evaluación finita.

2.2 Sintaxis de las expresiones de tipo

Sean TC un conjunto de constantes de tipo y TV un conjunto de identificadores de tipo. La sintaxis del lenguaje de las expresiones de tipo T está dada por las siguientes producciones:

$$T ::= (\tau_1, \ldots, \tau_m)c^u \mid \alpha^u \mid (\tau_1 \xrightarrow{v} \tau_2)^u \mid (\tau_1 \times \tau_2)^u$$

donde $\tau_1, \ldots, \tau_m \in T$, $m \geq 0$, $c \in TC$, $u, v \in U$, $\alpha \in TV$. La decoración de un tipo es una etiqueta de terminación. Las decoraciones de un tipo funcional $(\tau_1 \xrightarrow{v} \tau_2)^u$ son las etiquetas: v de terminación de la aplicación de la función, y u de terminación de la evaluación de la expresión a un valor funcional.

La sintaxis de los esquemas de tipo S viene dada por las producciones:

$$S ::= \quad \forall\alpha_1 \ldots \alpha_n.\tau \mid \tau \qquad \text{donde} \quad \alpha_1, \ldots, \alpha_n \in TV,\ n \geq 1,\ \tau \in T.$$

La relación de instanciación $Inst(\sigma, \tau)$ se cumple cuando τ se obtiene al sustituir en el esquema σ las variables cuantificadas por tipos cualesquiera. Para los fines del presente trabajo, sólo se usarán esquemas de tipos en las estructuras de datos.

2.3 Sintaxis de los patrones

Sean V un conjunto de identificadores de variables y C un conjunto de constantes. La sintaxis del lenguaje de patrones P está dada por las siguientes producciones:

$$P ::= \quad c \mid x \mid c(p) \mid (p_1, p_2) \qquad \text{donde} \quad c \in C,\ x \in V,\ p, p_1, p_2 \in P.$$

2.4 La relación $\Rightarrow$

Dados un ambiente de constantes $\mathcal{C}$, un ambiente de variables Γ y un patrón p, definimos la relación $\Rightarrow$ de la forma siguiente $\mathcal{C}, \Gamma, p \Rightarrow \tau^t, \Gamma'$ indicando que en esos ambientes se puede inferir que el tipo de p es τ, y que por consecuencia del apareamiento se enriquece el ambiente Γ generándose un nuevo ambiente Γ'. Note que la decoración asociada al tipo τ es t pues los patrones no se evalúan. Normalmente esta relación la abreviamos así $\mathcal{C}, \Gamma, p \Rightarrow \tau, \Gamma'$.

2.5 Sintaxis de las expresiones del lenguaje

La sintaxis del lenguaje de expresiones E está dada por las producciones:

$$E ::= c \mid x \mid \textbf{fun}\ p_1 \rightarrow e_1\ \text{"|"} \cdots \text{"|"}\ p_n \rightarrow e_n \mid e_1 e_2 \mid c(e) \mid (e_1, e_2) \mid \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \mid e_1 + e_2$$

donde $c \in C$, $x \in V$, $p_i \in P$, $e_i \in E$, $1 \leq i \leq n$. Notemos que la definición funcional es por casos utilizando apareamiento de patrones.

2.6 Conjunto de formas normales

Las formas normales son aquellas expresiones del lenguaje que no admiten reducción, es decir, que no se evalúan v.g. los valores o $ERROR$. Sintácticamente, el conjunto FN de formas normales está determinado por las siguientes producciones:

$$FN ::= c \mid x \mid \textbf{fun}\ p_1 \rightarrow e_1\ \text{"|"} \cdots \text{"|"}\ p_n \rightarrow e_n \mid c(v) \mid (v_1, v_2) \mid ERROR$$

donde $c \in C$, $x \in V$, $p_i \in P$, $e_i \in E$, $1 \leq i \leq n$, $v, v_1, v_2 \in FN$.

Note el lector que hemos incluido el caso de $ERROR$ por razones de completitud. Sin embargo, este caso será ignorado en lo que resta del trabajo, pues su tratamiento se haría por otra vía (tipos extendidos para excepciones [GS94a]). Además, para nuestros fines, es irrelevante ya que si una expresión se evalúa a $ERROR$, trivialmente podemos afirmar que ella termina. Por lo tanto, sólo consideraremos las expresiones que nunca se reducen a $ERROR$.

2.7 Semántica por reescritura

A continuación presentaremos las reglas de evaluación de las diferentes construcciones que forman nuestro lenguaje de expresiones. En estas reglas, v, v_1 y v_2 denotan formas normales.

Note el lector que las constantes y las variables no se evalúan y que las definiciones funcionales no se evalúan hasta que su argumento sea un valor. Los casos restantes se encuentran en la figura 1. En cualquiera otra situación, si la expresión e no está en forma normal y ninguna de las reglas se aplica, $e \longrightarrow ERROR$. Un ejemplo de ésto se presentaría cuando en una aplicación e_1 no se evaluará a una función.

En la regla APP_3, la expresión $e[x \leftarrow v_2]$ denota el resultado de la sustitución en e de las ocurrencias libres de la variable x por el valor denotado por v_2. En la regla SUM_3, v denota al valor resultado al efectuar la operación aritmética con los valores denotados por v_1 y v_2.

$$\frac{e_1 \longrightarrow e_1'}{e_1 e_2 \longrightarrow e_1' e_2}\ APP_1 \qquad \frac{e_2 \longrightarrow e_2'}{v_1 e_2 \longrightarrow v_1 e_2'}\ APP_2 \qquad (\textbf{fun}\ x \rightarrow e)\, v_2 \longrightarrow e[x \leftarrow v_2]\ APP_3$$

$$\frac{e_1 \longrightarrow e_1'}{c(e_1) \longrightarrow c(e_1')}\ CEX_1 \qquad \frac{e_1 \longrightarrow e_1'}{(e_1, e_2) \longrightarrow (e_1', e_2)}\ PEX_1 \qquad \frac{e_2 \longrightarrow e_2'}{(v_1, e_2) \longrightarrow (v_1, e_2')}\ PEX_2$$

$$\frac{e_1 \longrightarrow e_1'}{\textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \longrightarrow \textbf{if}\ e_1'\ \textbf{then}\ e_2\ \textbf{else}\ e_3}\ IF_1$$

$$\textbf{if}\ true\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \longrightarrow e_2\ \ IF_2 \qquad \textbf{if}\ false\ \textbf{then}\ e_2\ \textbf{else}\ e_3 \longrightarrow e_3\ \ IF_3$$

$$\frac{e_1 \longrightarrow e_1'}{e_1 + e_2 \longrightarrow e_1' + e_2}\ SUM_1 \qquad \frac{e_2 \longrightarrow e_2'}{v_1 + e_2 \longrightarrow v_1 + e_2'}\ SUM_2 \qquad v_1 + v_2 \longrightarrow v\ \ SUM_3$$

Figura 1: Semántica Operacional del Lenguaje

En la sección 4 se presenta el esquema de la prueba de consistencia semántica del sistema de tipos extendidos con respecto a la semántica propuesta.

3 Tipos extendidos con terminación

A continuación presentamos en las figuras 2 y 3 las reglas de inferencia del sistema de tipos extendidos con información de terminación.

$$\frac{\Gamma(x) = \tau}{\mathcal{C}, \Gamma \vdash x : \tau^t}\ VAR \qquad \frac{\mathcal{C}(c) = \sigma \quad Inst(\sigma, \tau)}{\mathcal{C}, \Gamma \vdash c : \tau^t}\ CTE$$

$$\frac{\mathcal{C}, \Gamma, p_i \Rightarrow \tau_1, \Gamma_i \quad \mathcal{C}, \Gamma_i \vdash e_i : \tau^{u_i} \quad (i = 1, \ldots, n)}{\mathcal{C}, \Gamma \vdash \textbf{fun}\ p_1 \to e_1 \mid \cdots \mid p_n \to e_n : (\tau_1 \xrightarrow{u_1 \wedge \ldots \wedge u_n} \tau)^t}\ FUN$$

$$\frac{\mathcal{C}, \Gamma \vdash e_1 : (\tau_1 \xrightarrow{w} \tau)^u \quad \mathcal{C}, \Gamma \vdash e_2 : \tau_1^v}{\mathcal{C}, \Gamma \vdash e_1 e_2 : \tau^{u \wedge v \wedge w}}\ APP$$

$$\frac{\mathcal{C}(c) = \sigma \quad Inst(\sigma, \tau_1 \to \tau) \quad \mathcal{C}, \Gamma \vdash e : \tau_1^u}{\mathcal{C}, \Gamma \vdash c(e) : \tau^u}\ CEX$$

$$\frac{\mathcal{C}, \Gamma \vdash e_1 : bool^{u_1} \quad \mathcal{C}, \Gamma \vdash e_2 : \tau^{u_2} \quad \mathcal{C}, \Gamma \vdash e_3 : \tau^{u_3}}{\mathcal{C}, \Gamma \vdash \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 : \tau^{u_1 \wedge u_2 \wedge u_3}}\ IF$$

$$\frac{\mathcal{C}, \Gamma \vdash e_1 : \tau_1^u \quad \mathcal{C}, \Gamma \vdash e_2 : \tau_2^v}{\mathcal{C}, \Gamma \vdash (e_1, e_2) : (\tau_1 \times \tau_2)^{u \wedge v}}\ PEX$$

$$\frac{\mathcal{C}, \Gamma \vdash e_1 : num^u \quad \mathcal{C}, \Gamma \vdash e_2 : num^v}{\mathcal{C}, \Gamma \vdash e_1 + e_2 : num^{u \wedge v}}\ SUM$$

Figura 2: Reglas para las Expresiones

VAR Si en el ambiente de variables, un identificador tiene asociado un tipo, entonces podemos concluir que el identificador tiene ese tipo, y que la evaluación de una variable siempre termina pues una variable ya denota un valor.

CTE Si en el ambiente de constantes c tiene tipo τ, entonces podemos inferir que el tipo de esa constante tiene decoración t. Es decir, la evaluación de las constantes termina en tiempo constante.

FUN Esta regla de inferencia se usa para la definición de funciones por casos utilizando apareamiento de patrones. Note que la etiqueta de terminación asociada al tipo funcional resultante es la conjunción de las etiquetas de terminación asociadas a las expresiones en cada caso de la función.

APP En la regla para la aplicación, la terminación viene dada por la conjunción de la terminación de la evaluación del valor funcional (u), la terminación de la evaluación del argumento de la función (v), y la terminación de la aplicación del valor funcional al argumento evaluado (w).

CEX La terminación de un constructor aplicado a una expresión depende únicamente de la terminación de la expresión.

PEX La terminación de un par de expresiones está determinada por la conjunción de la terminación de cada expresión.

IF Podemos asegurar la terminación de la construcción condicional si se garantiza la terminación de la expresión booleana de selección, y la terminación de las expresiones asociadas con cada alternativa. Como se trata de un análisis estático, sólo se considera que terminan aquellos programas en los que las tres subexpresiones terminen.

SUM La terminación de una operación aritmética depende únicamente de la terminación de las expresiones operandos.

PCT Para los patrones constantes, el tipo se obtiene por instanciación del esquema que tenga asociado en el ambiente de constantes. El ambiente de variables no se altera.

PVA En el caso del patrón variable, al identificador se le asocia un tipo. Este se hará más preciso después por unificación [Rob65]. El ambiente de variables se enriquece con la asociación de tipo que se le ha hecho al identificador.

$$\frac{\mathcal{C}(c) = \sigma \quad Inst(\sigma, \tau)}{\mathcal{C}, \Gamma, c \Rightarrow \tau, \Gamma}\ PCT \qquad \frac{\mathcal{C}(c) = \sigma \quad Inst(\sigma, \tau_1 \to \tau) \quad \mathcal{C}, \Gamma, p \Rightarrow \tau_1, \Gamma_1}{\mathcal{C}, \Gamma, c(p) \Rightarrow \tau, \Gamma_1}\ CPA$$

$$\mathcal{C}, \Gamma, x \Rightarrow \tau, \Gamma + \{x : \tau\}\ \ PVA \qquad \frac{\mathcal{C}, \Gamma, p_1 \Rightarrow \tau_1, \Gamma_1 \quad \mathcal{C}, \Gamma_1, p_2 \Rightarrow \tau_2, \Gamma_2}{\mathcal{C}, \Gamma, (p_1, p_2) \Rightarrow (\tau_1 \times \tau_2), \Gamma_2}\ PPA$$

Figura 3: Reglas para los patrones

CPA En la regla del constructor aplicado a un patrón, el patrón enriquece al ambiente de variables y el tipo funcional del constructor debe ser consistente con el tipo del patrón.

PPA En la regla del par de patrones, el primer patrón enriquece al ambiente de variables, y este ambiente enriquecido se vuelve a enriquecer por el segundo patrón.

Ejemplo

Consideremos la deducción de tipo para la función (TWICE) $\lambda g.\lambda x.g(g\,x)$.

$$\frac{\dfrac{}{\mathcal{C}, \emptyset, g \Rightarrow \tau_1, (\Gamma_1 =)\{g : \tau_1 = (\tau \xrightarrow{u} \tau)^t\}}\ PVA \qquad \dfrac{\dfrac{}{\mathcal{C}, \Gamma_1, x \Rightarrow \tau, (\Gamma_2 =)\Gamma_1 + \{x : \tau\}}\ PVA \quad \alpha}{\mathcal{C}, \Gamma_1 \vdash \textbf{fun}\ x\ \to g(g\,x) : (\tau \xrightarrow{u} \tau)^t}\ FUN}{\mathcal{C}, \emptyset \vdash \textbf{fun}\ g\ \to \textbf{fun}\ x\ \to g(g\,x) : ((\tau \xrightarrow{u} \tau) \xrightarrow{t} (\tau \xrightarrow{u} \tau))^t}\ FUN$$

donde la derivación α es:

$$\frac{\dfrac{\Gamma_2(g) = \tau_1}{\mathcal{C}, \Gamma_2 \vdash g : (\tau \xrightarrow{u} \tau)^t}\ VAR \qquad \dfrac{\dfrac{\Gamma_2(g) = \tau_1}{\mathcal{C}, \Gamma_2 \vdash g : (\tau \xrightarrow{u} \tau)^t}\ VAR \qquad \dfrac{\Gamma_2(x) = \tau}{\mathcal{C}, \Gamma_2 \vdash x : \tau^t}\ VAR}{\mathcal{C}, \Gamma_2 \vdash g\,x : \tau^u}\ APP}{\mathcal{C}, \Gamma_2 \vdash g(g\,x) : \tau^u}\ APP$$

Como se ve en la deducción, la terminación de TWICE va a depender de la etiqueta u que indica la terminación de la evaluación del valor funcional asociado a g cuando se aplica a su argumento.

El siguiente teorema garantiza la consistencia semántica del sistema propuesto.

4 Teorema de consistencia semántica

Si para una expresión no recursiva e se puede deducir un tipo extendido con decoración t, entonces existe un n tal que la expresión se reduce a forma normal en n pasos de evaluación:

$$\emptyset \vdash e : \tau^t \Longrightarrow \exists n \; [e \xrightarrow{n} v \;\; \wedge \; v \in FN]$$

Esquema de la demostración. Hemos dicho que cuando la etiqueta del tipo de una expresión es t queremos indicar que la expresión es de evaluación finita. Esto nos conduce a pensar que la etiqueta de terminación t está asociada a un número máximo n de pasos de reducción de forma tal que si la expresión se somete a n pasos de evaluación se garantiza que ella queda reducida a su forma normal. Por lo tanto, se puede especificar un sistema de tipos extendidos con cota superior de pasos de reducción de manera tal que el tipo de una expresión tendrá etiqueta n si a lo sumo en n pasos de evaluación la expresión queda reducida a forma normal. La demostración del teorema se realiza en tres etapas: (1) probar la equivalencia entre ambos sistemas; (2) demostrar que en el sistema de tipos extendidos con cota superior de pasos de reducción los tipos se preservan durante la evaluación; y (3) demostrar que la propiedad $\emptyset \vdash e : \tau^n$ implica que el número de pasos de evalución de la expresión e es a lo sumo n. Los detalles completos de la demostración se encuentran en [Sal95].

5 Conclusiones

Hemos presentado un mecanismo simple y poderoso a través del cual se puede determinar estáticamente, para una clase no trivial de programas, si la ejecución de un programa terminará.

El enfoque utilizado consiste en extender el sistema de tipos de Hindley-Milner decorando los tipos con información acerca de la terminación de los programas. Esto permite una integración natural de la extensión a los lenguajes de la familia ML [MTH90, W+90], por

ejemplo HASKELL [HW90]. En efecto, la extensión del sistema de tipos propuesta en este trabajo se puede incorporar a la fase de síntesis de tipos de los procesadores de esos lenguajes, y de esta manera se podría ofrecer al programador información importante adicional acerca del comportamiento de sus programas.

El utilizar los tipos como transportes de información sobre la terminación de los programas es una aplicación genuina y original de la metodología general de tipos extendidos. Por ésto, el presente trabajo se puede considerar como un aporte a dicha teoría.

A nivel práctico, a pesar de la indecidibilidad del caso general, la incorporación de un sistema de estas características en los lenguajes de programación funcional contribuiría significativamente a aumentar la confiablidad de los programas: en muchos casos, se podrá tener certeza de inexistencia de errores causados por problemas de terminación.

Referencias

[Chu41] A. Church. *The Calculi of Lambda Conversion.* Princeton University Press, 1941. Reprinted 1963 by University Microfilms Inc., Ann Arbor, Michigan.

[Cut80] N. Cutland. *Computability : an introduction to recursive function theory.* Cambridge University Press, Bristol, 1980.

[Dam85] L. Damas. *Type Assignment in Programming Languages.* PhD thesis, CST 33-85, University of Edinburgh, Scotland, 1985.

[GS94a] J.C. Guzmán and A. Suárez. An extended type system for exceptions. In *Record of the 5^{th} ACM SIGPLAN workshop on ML and it's Applications*, pages 127–135, BP 105 - 78153 Le Chesnay Cedex, France, June 1994. INRIA, Rapport de Recherche No 2265. http://www.usb.ve/~suarez/PAPERS/except.ps.gz

[GS94b] J.C. Guzmán and A. Suárez. Viewing type systems as information carriers. In *Proceedings of the 10^{th} Workshop on the Mathematical Foundations of Programming Semantics*, Kansas, USA, 3 1994. http://www.usb.ve/~suarez/PAPERS/tsic.ps.gz

[Guz93] J.C. Guzmán. *On expressing the mutation of state in a functional programming language.* PhD thesis, Yale University, May 1993.

[Hin69] R. Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.

[HW90] P. Hudak and P. Wadler. Report on the programming language haskell. Technical Report YALEU/DCS/RR-777, Yale University, New Heaven, 1990.

[LG88] J. Lucassen and D. Gifford. Polymorphic effect systems. In *Proceedings of the 15th ACM Symposium on Principles of Programming Languages*, 1988.

[Mil78] R. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348–375, 1978.

[MTH90] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML.* The MIT Press, Cambridge, Massachusetts, 1990.

[Rob65] J. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[Sal95] J. Salas. Demostración por tipos extendidos de la terminación de programas recursivos simples. Tesis de maestría, Universidad Simón Bolívar, Caracas, 1995.

[W+90] P. Weis et al. The caml reference manual. Technical Report 121, INRIA, Paris, September 1990.