Inferencia de Tipos (Algoritmo W)
Semántica y Verificación Formal
Proyecto Final

Lissarrague Berumen Iker, Enriquez Mendoza Javier

Introducción

Las expresiones válidas en un Lenguaje son definidas a partir de reglas sintácticas y de tipos.
Tomemos como ejemplo Haskell, varias veces en nuestra experiencia programando en este lenguaje el intérprete nos arroja errores de tipo. Para que esto sea posible Haskell deduce el tipo de una expresión a partir de la sintaxis, esto es lo que conocemos como inferencia de tipos.
El problema que surge cuando se diseña un lenguaje tipado pero sin anotaciones de tipo es :

- Existen Γ, T tales que $\Gamma \vdash e^a$: T
- $erase(e^a) = e$, donde $erase$ es la función que elimina todas las anotaciones de tipos de una expresión

Para este proyecto nosotros usaremos un lenguaje cercano a ML. La certificación de nuestra implementación del Algoritmo W se realiza en dos pasos: probar la correctud y completes de W respecto a las lenguas de Tipado del lenguaje

El Lenguaje

La mayoría de los lenguajes de programación funcionales tienen un núcleo común el lenguaje conocido como Mini-ML, que es el cálculo lambda con tipado simple, enriquecido con expresiones let (definiciones locales de valores polimórficos).
Las expresiones válidas de nuestro lenguaje son :

- Constantes Numéricas
- Variables (identificadores)
- Funciones lambda (λx.e)
- Aplicación (e e')
- Let

Definido en coq como un tipo inductivo.
Sería trivial agregar a nuestro lenguaje otros constructores (como condicionales, listas, etc.) pero esto sólo aumentaría la complejidad y longitud de las pruebas sin ninguna aportación real.

Los Tipos

Un tipo es una clasificación asociada a los datos que le dice al compilador o intérprete con qué intención el programador usará estos datos.
intuitivamente cada expresión del lenguaje está asociada a un tipo. La información del tipo de las variables la encontramos en el contexto. Pero el constructor **let** introduce funciones polimórficas, es decir que pueden ser aplicadas a objetos de distinto tipo y que el tipo de estas expresiones depende completamente del parámetro que reciba al momento de la aplicación. Así que el tipo de estas funciones contiene variables cuantificadas. Para esto usamos el tipo *scheme.*
Los tipos que vamos a considerar son:

- Number
- Var, que recibe un stamp (nat)
- Arrow, type -> type

El tipo Scheme

Un tipo scheme se define como un tipo cuantificado universalmente sobre un conjunto finito de variables de tipo, ∀α1 … αn.τ donde τ es un tipo, las variables cuantificadas se conocen como *genéricas.* Pero un tipo scheme puede también tener variables libres. Si un tipo scheme no tiene variables genéricas se le conoce como trivial

Tipo instancia y sustitución

El tipo instancia está definido con la función de sustitución de la siguiente manera: un tipo τ' es tipo instancia del tipo τ si existe una sustitución *s* tal que *s(*τ) = τ'

Instancia y sustitución genérica

Un tipo τ' es una instancia genérica del tipo *scheme* ∀α1 … αn.τ si y sólo si existe una sustitución genérica *sg* para las variables genéricas tal que *sg(τ)* = τ'. Esta sustitución no afecta a las variables libres del tipo *scheme.*

Las reglas de Tipado

Ambiente

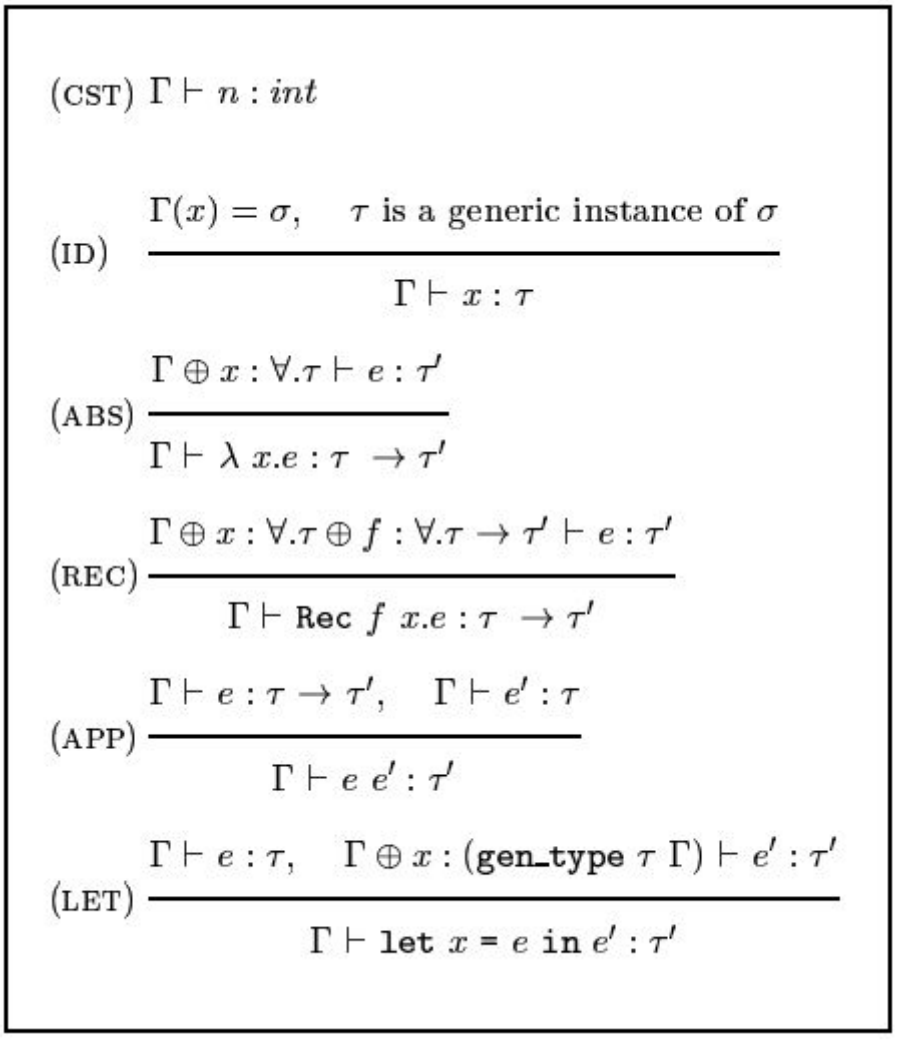
Para conocer los tipos de las variables en nuestro lenguaje es necesario que tengamos un contexto de tipos en el cual se va a asociar cada nombre de variable con su tipo correspondiente. Nuestra implementación de este contexto es una lista de pares (*id, scheme*)

Generalización de tipos

Como ya habíamos mencionado anteriormente el constructor **let** introduce en el contexto identificadores con tipos polimórficos, i. e. tipos *scheme* no triviales.
EL problema surge cuando dos tipos diferentes se convierten en tipos schemes equivalentes. Por ejemplo, si α y β denotan dos identificadores distintos no libres en el contexto, los tipos $\alpha \to \alpha$ y $\beta \to \beta$ introducen respectivamente los tipos scheme $\forall \alpha.\alpha \to \alpha$ y $\forall \beta.\beta \to \beta$ que en coq son dos términos sintácticamente diferentes. El sistema de tipos y el algoritmo de inferencia no necesitan decidir si dos tipos scheme son equivalentes, pero para demostrar la correctud del algoritmo si es necesario.

Reglas de Inferencia

Las reglas están descritas en la siguiente imagen, usamos la notación $\Gamma \vdash e : \tau$ lo que quiere decir que la expresión *e* tiene tipo τ bajo el contexto Γ.



Tipo Principal

El sistema de tipos propuesto permite que para una sola expresión haya más de una derivación de tipos. Por ejemplo Γ ⊢ λx.x : *number* → *number* y tambien Γ ⊢ λx.x : α → α con α ligada en Γ. Entonces decimos que el tipo principal de una expresión es el tipo más general asociado a dicha expresión y el resto de los tipos asociados son instancias del tipo principal. Por lo tanto α → α es el tipo principal de λx.x

El algoritmo de inferencia de tipos (W)

Para este proyecto utilizaremos una implementación del algoritmo de inferencia de tipos de Damas-Milner (Algoritmo W) que obtiene el tipo principal de una expresión. Este algoritmo es uno de los más utilizados para la inferencia de tipos, tiene su origen en el algoritmo de tipos para cálculo lambda tipificado propuesto po Haskell B. Curry y Robert Feys en 1958. En 1969 Roger Hindley demostró que ese algoritmo siempre deduce el tipo más general. En 1978 Robin Milner propuso un algoritmo equivalente para el lenguaje ML. En 1985 Luis Damas demostró que el algoritmo propuesto por Milner es completo y lo extendió para dar soporte a la noción de referencia polimórfica.
Este algoritmo usa el mecanismo clásico de unificación para obtener el unificador más general para dos tipos τ1 y τ2 si son unificables, en él caso contrario entonces el algoritmo falla.

El algoritmo

El algoritmo *W* que dado un término sin anotaciones de tipo e, devuelve un contexto Γ, un término con anotaciones e^a y un tipo τ, tales que se cumple el tipado más general posible Γ ⊢ e^a : τ, se define:

- Variables: $W(x) = x : X \vdash x : X$ donde X es una variable de tipo nueva.
- Aplicación: $W(e_1\ e_2) = (\Gamma_1 \cup \Gamma_2)\mu \vdash (e_1^a e_2^a)\mu : X\mu$ donde
  - $W(e_1) = \Gamma_1 \vdash e_1^a : T_1$
  - $W(e_2) = \Gamma_2 \vdash e_2^a : T_2$
  - $\mu = umg(\{S_1 = S_2 \mid x : S_1 \in \Gamma_1,\ x : S_2 \in \Gamma_2\} \cup \{T_1 = T_2 \to X\})$ con X una variable de tipo nueva
- Abstracción: Si $W(e) = \Gamma\ e^a : S$ entonces
  - Si Γ tiene una declaración para x, digamos $x : R \in \Gamma$, entonces $W(\lambda x.e) = \Gamma \setminus \{x : R\} \vdash \lambda x : R.e^a : R \to S$
  - Si Γ no tiene una declaración para x, $W(\lambda x.e) = \Gamma \vdash \lambda x : X.e^a : X \to S$ siendo X una variable nueva.