Subtipos y subclases

Roni Guillermo Apaza Aceituno

Universidad Nacional de San Agustin

rapazaac@unsa.edu.pe

September 26, 2018

Introducción

- Los sistemas de tipo estático serían ventajosos en los lenguajes orientados a objetos sobre los procedimentales.
- Los primeros intentos eran inseguros o más inflexibles de lo que uno podia desear.
- En algunos casos la dependencia de los moldes de tipos hacian esto muy rigido.

Introducción

- Estos moldes erán controlados en tiempo de ejecución.
- Estos moldes no eran controlados en tiempo de ejecución.
- Todo con el fin de obtener una buena expresividad.

Introducción

- El sistema de tipos des muy flexible.
- Lo cual hace que la verificación de tipos se haga en tiempo de ejecución.
- También puede hacerlo en el tiempo de enlace.

La verificación de tipos

- Para no hacer operaciones como $5/hello$.
- Verifica los tipos de operandos.
- Así aparece los lenguajes de tipo estático y dinámico.

La verificación de tipos

- Un lenguaje dinámicamente tipado tiene su verificación de tipo en tiempo de ejecución justo antes de realizarse.
- Un lenguaje estaticamente tipado tiene su verificación de tipo en tiempo de compilación.
- El sistema de tipos puede garantizar que cada expresión tenga un tipo compatible con el tipo de expresión definida.

La verificación de tipos

- Revisar estaticamente en un lenguaje de programación tiene ventajas como proporcionar información anterior sobre errores de programación.
- Ventajas como tener documentación sobre las interfaces de los componentes (por ejemplo procedimientos, funciones y paquetes)
- Esto tiene la ventaja de verificar los tipos en tiempo de ejecución.

La verificación de tipos

- Hacer esto en tiempo de ejecución puede retardar la ejecución del programa.
- Lo cual hace menos optimo al compilador.
- Claro que existen desventajas.

La verificación de tipos

- Una desventaja es que la verificación de tipos estaticos son conservadoras.
- Esto puede provocar el rechazo de un programa que este bien escrito.
- Esto hace del lenguaje de programación menos expresivos.

La verificación de tipos

- Lenguajes procedimentales como Pascal, CLU, Modula-2 y Ada.
- Junto con lenguajes funcionales como ML y Haskell.
- Presentan el tipado estático seguro.

La verificación de tipos

- Aun existiendo algunos agujeros inseguros como por ejemplo registro variables en Pascal.
- La seguridad de los sistemas de tipos estan presentes.
- El polimorfismo ha sido útil para aumentar la expresividad de los lenguajes de programación imperativos y funcionales.

La verificación de tipos

- El problema central de muchos lenguajes de programación se basan en la idea del mensaje enviado a un objeto.
- En otras palabras si el receptor de dicho lenguaje tiene un método para ejecutar dicho mensaje.
- Aun así existen problemas similares.

La verificación de tipos

- Es aqui donde podemos mencionar un problema sobre el subtipado.
- Junto con el uso de pseudovariables.
- Para la representación del objeto ejecutado en el método.

La verificación de tipos

- La subtipificación hace un cambio en los tipos de los parámetros reales de los métodos.
- También ocurre para los procedimientos y funciones.
- Puede generar un tipo diferente al especificado en la declaración formal de los parametros correspondientes.

La verificación de tipos

- La herencia permite que los métodos definidos en una clase puedan definirse en una subclase.
- Debe crearse alguna forma de evitar agujeros al usar estos pasos.
- Especialmente si hablamos de la escritura del programa.

La verificación de tipos

- Esto represento un problema par la comprobación de tipos en lenguajes orientados a objetos.
- Tenemos algunos problemas de sobre estos sistemas de comprobación.
- Sobre lenguajes de programación orientados a objetos o hibridos.

La verificación de tipos

- Algunos no muestran mucho respeto por el tipado estático como Smalltalk.
- Algunos tienen sistemas de tipo estático relativamente flexibles que requieren moldes de tipo para superar deficiencias de tipo. Pueden estar desmarcados como en C++ o Object Pascal.
- También puede haber verificación en tiempo de ejecución como en Java.

La verificación de tipos

- Algunos proveen mecanismos como declaraciones **typecase** para permitir la busqueda de mas tipos de datos para determinar el sistema de tipo.
- De esa forma esta Modula-3, Simula-67 y Beta.
- Algunos permiten asignaciones inversas de superclases a subclases, lo cual requiere verificaciones en tiempo de ejecución. Por ejemplo en Beta y Eiffel.

La verificación de tipos

- Algunos requieren que los parametros de métodos anulados en subclases tengan los mismos tipos que en las superclases de manera exacta. Como por ejemplo en C++, Java, Object Pascal y Modula-3.
- Esto resulta en mas flexibilidad, aun cuando no sea muy deseable.
- En cambio otros permiten flexibilidad al cambiar los tipos de parámetros o variables de instancia.

La verificación de tipos

- Esta flexibilidad exige verificadores adicionales en tiempo de ejecución.
- O verificadores en tiempo de enlace para detectar errores.
- Como ejemplo tenemos a Beta y Eiffel.

La verificación de tipos

- Por lo tanto todos los programadores deben tratar las deficiencias de tipo.
- Es necesario una verificación de tipo en tiempo de ejecución.
- O en su defecto permiten los errores en tiempo de ejecución.

La verificación de tipos

- Funciones y verificaciones en tiempo de ejecución o asignaciones inversas son necesarias para manejar problemas dificiles con estructuras de datos heterogéneas.
- Es preferible tener sistema de tipo que permitan la programación de manera mas natural posible.
- Todo esto mientras se intenta atrapar la mayor cantidad de errores en tiempo de compilación como sean posibles.

La verificación de tipos

- Existen problemas con la configuración de tipos en la clase.
- Existen problemas con la falta de correspondencia de la jerarquia en la herencia y el subtipado.
- Esto parece mejorar hacia una combinación de mayor seguridad y mayor expresividad en los sistemas de tipos.

Tipos y subtipos

- Las nociones de tipo y clase a menudo se confunden dentro de los lenguajes de programación orientados a objetos.
- Tienen papeles distintos dentro de este paradigma de programación.
- Pueden ser útilmente distinguidos unos de otros.

Tipos y subtipos

- Los tipos proporcionan información de la interfaz definidiendo cuando ciertas operaciones son validas.
- Las clases proporcionan información de la implementación incluidos nombre y valores iniciales de las variables de las instancias.
- De las variables delos nombres y de las variables de los cuerpos de los métodos.

Tipos y subtipos

- Un tipo en un lenguaje de programación representa un conjunto de valores.
- También representa las operaciones y las relaciones que pueden ser aplicadas los valores representados por este tipo.
- Asi por ejemplo tenemos el tipo entero.

Tipos y subtipos

- Un tipo de objeto punto representa un punto.
- También representa a la colección de puntos.
- También representa los mensajes que se pueden enviar a los objetos puntuales.

Tipos y subtipos

- Los lenguajes fuertemente tipados tienen mecanismos de verificación de tipo que garantizan las correctas operaciones de los tipos.
- También evitan las operaciones sin sentido.
- Se dicen que estos lenguajes son inseguros.

Tipos y subtipos

- Asi por ejemplo un lenguaje fuertemente tipado garantiza la operaciones sobre caracteres.
- No transforma la operacion de dos caracteres en uno solo como si fueran números.
- Estas comprobaciones podrían hacerse en tiempo de ejecución.

Tipos y subtipos

- En los lenguajes de tipo estático como C y Pascal.
- Las expresiones son asignadas a tipos por el verificador de tipos.
- Un conjunto de reglas para el tipo basadas en la estructura de las expresiones son usadas para construir el tipo estático de cada expresión.

Tipos y subtipos

- El verficador de tipo garantiza que si una expresión tiene un tipo estático T, entonces en tiempo de ejecución, el resultado de dicha expresión dara un valor de tipo T.
- Un sistema de verificación de tipo estático determina en que contexto ocurre legalmente una expresión.
- Es lo que garantiza un lenguaje de tipo estático.

Tipos y subtipos

- En programación orientada a objetos la operación más importante es enviar un mensaje a un objeto.
- Un objetivo de la verificación de tipos es garantizar que los mensajes inapropiados no se envíen a los objetos.
- Al enviarse un mensaje a un objeto se debe garantizar un metodo, un parametro y el tipo de retorno.

Tipos y subtipos

- El objeto debe tener un método con el mismo nombre.
- También debe haber un parámetro formal.
- El tipo de retorno debe ser compatible con los de la llamada.

Tipos y subtipos

- Desde el punto de vista del tipo, un mensaje enviado a un objeto es similar a extraer un campo de un registro.
- El campo del registro resulta ser una función.

Tipos y subtipos

- Integer, Character, Real son tipos de datos básicos conocidos para lenguajes imperativos y funcionales.
- Acompañados de operadores para crear nuevos tipos.
- Asi por ejemplo: Tipos de registros, tipos de arreglos y tipos de funciones y procedimientos entre otros.

Tipos y subtipos

- Lenguajes orientados a objetos reemplazan los tipos mencionados con tipos de objetos.
- Como todo objeto proporcionan información acerca de los nombres y métodos (y sus tipos) admitidos por el objetos del tipo.

Tipos y subtipos

- Un tipo S es un subtipo de un tipo T (escrito S $<:$ T) si una expresión de tipo S puede usarse en cualquier contexto que espere un elemento de tipo T.
- Si S $<:$ T y la expresión e tiene el tipo S, e también tiene el tipo T.
- Si a una expresión se le puede asignar un tipo T, también se le puede asignar cualquier tipo que sea un supertipo de T.

Tipos y subtipos

- En la mayoria de lenguajes de programación sin subtipificación solo se puede agregar un tipo.
- Por ejemplo, sea x una variable de tipo T. Si e es una expresión de tipo T, entonces $x := e$ es una declaración de asignación legal.
- Si S es un subtipo de T y e' tiene tipo S, entonces $x := e'$ también será una declaración de asignación legal.

Tipos y subtipos

- Un parámetro real de tipo S puede utilizarse en una llamada a función o procedimiento cuando el tipo de parámetro formal correspondiente se declara como T.
- Los lenguajes de programación orientados a objetos puros se da este soporte, manteniendo los objetos como referencias implicitas.
- Interpretan los párametros de asignación y paso como vinculos para nombres nuevos de objetos existentes. Esto se conoce como formas de compartición.

Tipos y subtipos

- ¿Cómo podemos determinar cuándo un tipo es un subtipo de otro?
- En este documento se presentara cuando dos tipos son subtipos.

Tipos de registro

- Registra valores asociados a etiquetas particulares.
- El tipo de registro especifica el tipo del valor correspondiente a cada etiqueta.
- Para simplificar, tratamos aquí solo con registros inmutables (o "read-only ") del tipo encontrado en los lenguajes de programación funcional como ML.

Tipos de registro

- No hay operaciones disponibles que actualicen campos particulares de estos registros.
- Uno solo puede crearlos como un todo y extraer los valores de campos particulares.

Tipos de funciones

- Las funciones asignan valores de un tipo a elementos de otro tipo.
- El tipo de función se determina por el tipo de argumentos a los que se puede aplicar la función y el tipo de valores devueltos por la función.
- Los tipos de procedimiento, Proc(S) escritos, pueden subtipificarse como si fueran tipos de funciones degeneradas.

Tipos de variables

- Las variables de tipo T tienen propiedades diferentes de las expresiones ordinarias de tipo T.
- Por ejemplo, si fv es una variable de tipo FoodType, y apple es un valor de tipo FoodType.
- El enunciado $fv := apple$ es una afirmación de tipo correcto, mientras que $apple := fv$ claramente no es correcto.

Tipos de variables

- Nos referimos a los tipos de variables como tipos de referencia
- Una variable declarada para tener el tipo T en realidad tiene el tipo ref T. 2 .
- Una variable de tipo T en realidad denota una ubicación (o referencia) en la que uno puede almacenar un valor de tipo T.

Tipos de objetos

-
-
-

Clases y subclases

-
-
-

Subtipos y subclases

-
-
-

Varianza

- Herencia de conveniencia.
- La linea hereda de un punto.
- Punto (coordenadas)
- Linea (dirección)

Varianza

- La varianza designa el hecho de utilizar un tipo de objetos no correspondiente con el esperado.
- Para que esto ocurra el tipo utilizado y el tipo esperado deben formar parte de la misma jerarquia de clases.
- Si el tipo utilizado es un supertipo del tipo esperado.
- Un subtipo del tipo esperado.

Covarianza

- Si el tipo utilizado es un subtipo del tipo esperado (tipo derivado)
- Se aplica el termino covarianza
- La clase Cliente es usado donde se espera la clase persona.

Contravarianza

- Si el tipo utilizado es un supertipo del tipo esperado (tipo menos derivado)
- Se aplica el termino contravarianza
- Se usa la clase persona donde se espera la clase cliente.

Preguntas