Sistemas de tipos

Conjunto de reglas utilizado por un lenguaje para estructurar y organizar su coleccion de tipos (Ghezzi, p. 159)

- Integrado por los métodos utilizados para construir tipos, por las reglas de equivalencia y compatibilidad entre tipos, la inferencia de tipos, entre otros (Louden, p. 331)
- El objetivo de un sistema de tipos es escribir programas seguros

Clasificación según cuando se liguen los tipos

Si las ligaduras y el chequeo se realizan en tiempo de compilación, es de **tipado estático**. Caso contrario, es de **tipado dinámico**. Durante el chequeo de tipos se aplican reglas como las de compatibilidad y equivalencia (Sebesta, p. 302)

Tipado Estático

Según el libro de Ghezzi... *Un lenguaje **tipado estáticamente** requiere que los tipos de todas las expresiones se conozcan en tiempo de compilación.* (p. 162)

Según el libro de Louden... "*La alternativa al tipificado dinámico es el **tipificado estático**: los tipos de expresiones y de objetos se determinan del texto del programa, y el intérprete lleva a cabo la verificación de tipos antes de la ejecución.*" (p. 205, español)

Ventajas:

- La información que provee el estado estático le permite a los compiladores alocar memoria eficientemente y generar machine code que manipule los datos eficientemente, por lo que aumenta la **eficiencia** (de ejecución)
- Los tipos estáticos pueden ser usados por un compilador para reducir la cantidad de código que se necesita compilar, mejorando la **eficiencia** (de compilación)
- El chequeo de tipado estático permite que varios errores comunes de programación se capturen rápido, mejorando la **seguridad**

- El tipado explícito mejora la **legibilidad** y reduce la **ambiguedad**, al documentar el diseño de los datos

Tipado Dinámico

Clasificación según la seguridad de tipos

- Un **error de tipo** es cualquier intento de manipular objetos por medio de operaciones ilegales
- Un programa tiene **seguridad de tipos** si está garantizado si ninguna operación produce un error de tipos (Ghezzi, p. 160)

Tipado Fuerte

La posta es que los libros no se ponen de acuerdo. La cátedra usa la definición de Ghezzi

- Definición del libro de Ghezzi → *"A un sistema de tipos se le dice **fuerte** si garantiza la seguridad de tipos; los programas escritos siguiendo las restricciones establecidas por el sistema de tipos tienen la certeza que no tendrán errores de tipo ... si el lenguaje es fuertemente tipado, la ausencia de errores de tipo en los programas **puede** ser garantizada por el compilador"* (p. 161)
- Definición del libro de Louden → *"Si la definición de un lenguaje de programación especifica un sistema de tipos completo que pueden ser aplicado estáticamente y que garantiza que todos los errores de corrupción de datos (no intencionales) en un programa serán detectados en el menor tiempo posible, entonces se dice que el lenguaje es **fuertemente tipado**. Esencialmente, esto quiere decir que todos los errores de tipos son detectados en tiempo de compilación, con la excepción de algunos errores que solo se pueden detectar durante ejecución."* (p. 331)
    - Además, Louden conecta el tipado fuerte con el estático:

      > *"En un lenguaje con **tipificación fuerte**, deben localizarse todos los errores de tipos antes del tiempo de ejecución, por lo cual estos lenguajes deben efectuar tipificado estático, y los errores de tipo se reportan como*

mensajes de error de compilación que impiden la ejecución. Sin embargo, la definición de un lenguaje puede no ser especifica con respecto a si se debe utilizar tipificado dinámico o estático." (p. 205)

- Definición del libro de Sebesta → "Un *lenguaje de programación es* ***fuertemente tipado*** *si siempre se detectan los errores de tipo. Esto requiere que los tipos de todos los operandos se puedan determinar en tiempo de compilación o ejecución. La importancia del tipado fuerte radica en su habilidad para detectar cualquier mal uso de las variables que resulte en un error de tipo. Un lenguaje con tipado fuerte también permite la deteccion, en tiempo de ejecución, de los usos de valores incorrectos de tipos en las variables que pueden guardar valores que pertenezcan a mas de un tipo.*" (p. 304)
    - Cómo ejemplo de lenguajes con tipado fuerte habla de Ada, Java, C#, F# y ML/I. No permiten que ocurran errores de tipo, por lo menos de forma implícita.

Tipado Débil

En general, un lenguaje es **débilmente tipado** si ocurre lo contrario, osea, no previene cualquier error de tipo. Los autores en general consideran a C y C++ como ejemplos de lenguajes con tipado débil, debido a que incluyen los tipos uniones (que son inseguros porque fácilmente pueden causar errores de tipos), los valores de los punteros pueden cambiar de tipo a los valores sin que salte error (de float a entero por ejemplo), entre otros

Reglas de resolución

Reglas y mecanismos involucrados en el proceso de chequeo de tipos que hace el lenguaje

Compatibilidad de Tipo

Reglas semánticas que determinan si el tipo de un objeto es válido en un contexto particular. "*Dictan qué tipos de operandos son aceptables para cada operador y, por lo tanto, especifican los posibles errores de tipo del lenguage.*" (Sebesta, p. 304).

- Un tipo es **compatible** con otro si es **equivalente** y **se puede convertir**

Equivalencia de Tipo

La equivalencia entre tipos se puede implementar de dos maneras:

- **Equivalencia por nombre** → Dos variables son del mismo tipo si y solo si están declaradas juntas o si están declaradas con el mismo nombre de tipo
- **Equivalencia por estructura** → Dos variables son del mismo tipo si los componentes de su tipo son iguales

Conversión de Tipos

- **Widening** → Cada valor del dominio tiene su correspondiente valor en el rango
- **Narrowing** → Cada valor del dominio puede no tener su correspondiente valor en el rango. En este caso algunos lenguajes producen un mensaje avisando la pérdida de información
- **Cláusula de casting** → Conversiones explicitas, se fuerza a que se convierta

Inferencia de Tipo

Permite que el tipo de una entidad declarada se “infiera” en lugar de ser declarado. La inferencia puede realizarse de acuerdo al tipo de: un operador predefinido, un operando, un argumento, o el tipo del resultado.

Falta monomorfismo y polimorfismo (no entra en la práctica)