Dynamické vlastnosti

Dynamické vlastnosti jazyků

- Základem dynamických vlastností je obvykle pojem **objekt**
- **Plně dynamické jazyky (duck-typing)**
  - Objekty mohou měnit své vlastnosti za běhu
    - Včetně přidávání **nových** datových položek a metod
- **Jazyky s prototypy**
  - Objekty sdílejí prototyp, popisující vlastnosti těchto objektů
    - Metody a datové položky jsou deklarovány v prototypu
    - Prototyp může být měněn za běhu (obvykle pro budoucí objekty)
    - Implementace metody je někdy součástí objektu, ne prototypu
- **Polymorfismus ve staticky typovaných jazycích**
  - Tentýž kód dokáže pracovat s objekty různých typů
    - Podmínkou bývá společný předek (interface) těchto typů
    - Společný kód dokáže přímo pracovat pouze s položkami/metodami staticky deklarovanými u tohoto předka
    - Různé typy mohou mít **různé implementace** těchto metod

Dynamické vlastnosti jazyků

- Přidávání nových datových položek a metod
  - Znemožňuje pevný layout datových položek/tabulek metod
  - Každý objekt má své mapování identifikátorů na položky/metody
    - Obvykle není možné přetěžování identifikátorů funkcí
    - Mapování lze zrychlit globální konverzí identifikátorů na binární GUID
  - Objekt je fyzicky reprezentován (hashovací) tabulkou
- Změna implementace existující metody za běhu
  - Objekt může mít staticky nebo prototypem určený layout dat i tabulky metod
  - Změna implementace realizována změnou záznamu v tabulce metod
    - Tabulka bývá součástí každého objektu
- Polymorfismus
  - Objekt daného typu může mít staticky určený layout dat/tabulky metod
  - Tabulka metod má pro daný typ fixní obsah
    - Tabulka sdílena všemi objekty téhož typu

Dynamické vlastnosti jazyků

- Plně dynamické jazyky
  - Objekt je fyzicky reprezentován (hashovací) tabulkou
  - Číselné typy obvykle **boxovány** jako objekty
  - Všechny proměnné/datové položky jsou **ukazatele** na objekty
    - Vyžaduje garbage collector
- Jazyky s prototypy nebo statickými typy
  - Objekt je fyzicky reprezentován pevným layoutem datových položek a tabulkou (virtuálních) metod
    - layout může být určen staticky nebo dynamicky
    - během života daného objektu se layout nemění
    - součástí objektu je ukazatel na nějakou formu typové informace
      - prototyp nebo class objekt – dostupný ze zdrojového jazyka
      - tabulka virtuálních metod – ze zdrojového jazyka neviditelná
    - tabulka metod může být součástí objektu nebo typové informace
  - Číselné typy obvykle v **nativní** podobě
  - Proměnné a datové položky jsou těchto typů
    - Ukazatele na objekty (často garbage-collected)
    - Nativní číselné typy
    - (Někdy) speciální typy (string, kontejnery...)
    - (C++, C#) Struktury/pole složené z těchto typů

Dynamické vlastnosti jazyků

- Běhová identifikace typů
  - Plně dynamické jazyky
    - Má smysl pouze jako detekce/lokalizace požadovaných vlastností (datových položek/metod) – duck-typing
    - Založena na dotazech do hashovací tabulky reprezentující objekt
  - Jazyky s explicitním pojmem typu
    - Polymorfní typy je možné realizovat pouze nepřímo - ukazatelem
    - Typovým identifikátorem je prototyp/class-object/tabulka virt. metod
    - Překladač/runtime musí být schopen typový identifikátor v objektu najít
      a) Jednotná hlavička všech typů obsahující typovou informaci
         - Obvykle implikuje jednoduchou dědičnost a univerzální společný předek (Object)
      b) Poloha a význam typové informace pro každý objekt jiná
         - (C++) Pouze některé objekty podporují běhovou identifikaci typu
         - Lokalizace založena na layoutu staticky deklarovaného předka
- Garbage collection
  - Vyžaduje schopnost identifikace typu pro každý dynamicky alokovaný objekt
    - Vyžaduje jednotnou hlavičku všech objektů
    - Obvykle znemožňuje násobnou dědičnost a ukazatele dovnitř objektů

Static analysis

of dynamic features

Static analysis of dynamic features

- Purpose of static analysis
  - Verification: Discovering errors sooner than at run-time
  - Optimization:
    - Eliminating run-time checks where possible
    - Eliminating/simplifying run-time branching/indirect calls
    - Improving outcome of other optimizations by decreasing uncertainty
    - Allowing procedure integration/specialization
    - Allowing cheaper allocation for objects of predictable lifetime
- Features to analyze
  - Dynamic type of an expression
    - Upper/lower bounds wrt. inheritance hierarchy
    - Not applicable in duck-typed languages
  - Presence/location/definition of a data field or method
    - In true dynamic languages
  - Existence of aliasing

Static analysis of dynamic features

- Features to analyze
  - Dynamic type of an expression
    - Upper/lower bounds wrt. inheritance hierarchy
    - Not applicable in duck-typed languages
  - Presence/location/definition of a data field or method
    - In true dynamic languages
  - Existence of aliasing

  - Expressions are ultimately computed from variables and objects
  - Analyzing expressions involves analyzing objects and vice versa
  - Objects are usually referenced indirectly, via pointers/references stored in variables and other objects
  - For any non-trivial static analysis, the links between variables and objects and among objects must be analyzed
    - **“Points-to” analysis**
  - Often, the points-to analysis must be done simultaneously with the analysis of other dynamic features (e.g. dynamic types)