Mit jelent a rescue klóz? Mi a szerződése (Hoare-triple), ha nincs benne retry?

Egy Eiffel rutin végén állhat egy rescue klóz, mely akkor fut le, amikor a rutin törzsében kivétel lép fel. A rescue klóz célja, hogy a rutin úgy érhessen véget, hogy legalább az osztályinvariáns teljesüljön, ha már az utófeltételt nem sikerült elérni. (A retry a rescue klózban a rutin újrakezdését kezdeményezi.) A szerződést (utófeltételt) teljesíteni nem tudó, és ezért kivételt kiváltó törzset követve lefut a `rescue`, majd a fellépett kivételt propagálja a hívó felé, azaz a hívóban a hívás helyszínén fellép a szóban forgó kivétel. A hívó a szerződést teljesíteni nem tudó objektumot konzisztens állapotban látja (az osztályinvariáns teljesül rá).

Az r rutin $retry_r$ retry-klózának szerződése tehát:

$$\{true\}\ retry_r\ \{inv_C\}$$

ahol inv_C az r osztályának, azaz a C-nek az egyesített invariánsa (a bázistípusokban és a C-ben deklarált invariánsok konjunkciója).

Amennyiben egy rutin nem tartalmaz explicit retry-klózt, az ANY-ből megörökölt (és esetleg átnevezett) `default_retry` eljárás lesz implicit módon a rutin retry-klóza, melynek ANY-beli implementációja üres (SKIP utasítás). Egy lehetőség a retry-klózra egy creation-procedure végrehajtása (például a `default_retry` és a `default_create` ugyanolyan módon történő implementálása.)

Mit értünk multimethodon?

Bizonyos objektumorientált nyelvekben a dinamikus kötés nem *egyetlen* kitüntetett paraméter dinamikus típusán működik, hanem az összes paraméter dinamikus típusát figyelembe veszi. Például az Eiffelben az `a.b(c)` rutinhívásnál az `a` és a `c` változók statikus típusa alapján dől el, hogy a `b` rutin hívása értelmes-e, és az `a` változó dinamikus típusa alapján dől el, hogy a művelet mely osztályban definiált implementációja hajtódik végre. Egy multimethodokat (azaz multiple dispatchet) támogató nyelvben az `a` és a `c` dinamikus típusa alapján választódik ki a végrehajtandó kód (azaz az `a` statikus típusában, illetve leszármazottaiban több olyan `b` művelet is lehet, amelynek a paramétere a `c` statikus típusának valamilyen altípusába tartozik).

A multimethodokat támogató nyelvekben az úgynevezett *bináris műveletek* (azaz amikor a művelet paraméterének típusa a fogadó objektum típusával kell megegyezzen, mint mondjuk egyenlőségvizsgálatnál) megvalósítása megoldott, hiszen minden osztályban megírhatjuk az arra az osztályra jellemző (és a fogadó és paraméter objektumot ugyanazzal típusozó) implementációt, és a dispatch során a legjobban illeszkedő implementáció hívódik meg. A szimmetriáról a multiple dispatch gondoskodik.

Milyen lehetőségei vannak a constrained genericitynek Eiffelben?

Az Eiffelben a generikus típusoknak több (fix számú) típusparamétere lehet, és mindre tehetünk megszorításokat (az aktuális típusparamétereknek meg kell felelniük a formális típusparaméterekre tett megszorításoknak). Természetesen a generic típusdefiníción belül a típusparamétert olyan típusként használhatjuk, amely következik a rá adott megszorításokból. (Például olyan műveleteket használhatunk rá...)

Ha nem teszünk megszorítást, mint például a `class G[T]` esetben, az ekvivalens azzal, mintha azt írtuk volna, hogy `class G[T->ANY]`. Természetesen az `ANY` helyett más felső korlátot is adhatunk a típusparaméterre, pl: `class G[T <- HASHABLE]`.

Még érdekesebb, amikor egy típusparaméterre több megszorítást adunk: ebben az esetben az aktuális paramétertípusnak az összes feltüntetett típus altípusa kell lenni. Például `class G[ T <- {HASHABLE, NUMERIC} ]`.

Ha több típust is meghatározunk egy típusparaméter elvárt bázisaként, akkor ezeket szükséges lehet összehangolni, hogy konzisztensen lehessen használni a típusparamétert és a rá értelmezett műveleteket a generikus definícióban. Ez az összehangolás azt jelenti, hogy a különböző bázistípusokban lévő műveleteket átnevezhetjük, hogy a névütközést elkerüljük. Például `class G[T <- {B rename f as g end, C}]`. Ezek az átnevezések nem jelentik azt, hogy az aktuális típusparaméterben ilyen névvel kell legyenek deklarálva ezek a műveletek.

A típusparaméterre megadható az is, hogy egyedeket abból a típusból milyen creation procedure-ök segítségével lehet létrehozni. Ha például a `B` osztálynak van egy `make` nevű, paraméter nélküli művelete, akkor a `class G[T <- B create make end] azt jelenti, hogy a`T`-nek megfeleltetett típus csak a B leszármazottja lehet, és ebben a típusban a`B`-ból megörökölt (és esetleg átnevezett)`make‘ művelet egy creation procedure.

Lehet olyan megkötést is adni egy típusparaméterre, hogy az aktuális csak olyan attached típus lehessen, amely öninicializáló, azaz amelynek az `ANY`-ból megörökölt (és esetleg átnevezett) `default_create` egy creation procedure-je. Például `class G[?T]`.

Lehet olyan megkötést tenni, hogy a típusparaméter invariáns: `class G[frozen T]`. Ellenkező esetben a típusparaméter kovariáns.

Az Eiffelben nincs F-bounded polymorphism, azaz egy `T` típusparaméterre adott megszorításban nem hivatkozhatunk a `T`-re.

Strukturális altípusosság esetén milyen altípusrelációk kell, hogy teljesüljenek az A, A', B, B', C, C' típusokra, hogy matematikailag helyes legyen, hogy az $(A \to B) \to C$ altípusa az $(A' \to B') \to C'$ típusnak?

A paraméter kontravariáns, az eredmény kovariáns kell legyen, ezért $(A' \to B') <: (A \to B)$ és $C <: C'$ kell, továbbá az elsőhöz az kell, hogy $A <: A'$ és $B' <: B$.