EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

A C++ Standard Template Library helyességvizsgálata

TDK Dolgozat

Pataki Norbert
programtervező matematikus V.

Témavezető: Porkoláb Zoltán
Programozási Nyelvek és Fordítóprogramok
Tanszék

Budapest, 2004

Készült az OTKA T037742
sz. pályázat támogatásával.

Tartalomjegyzék

I. Bevezetés **4**

II. Alapok **6**
II.1. Generikus programozás 6
II.2. A Standard Template Library 7
II.3. Ismert STL specifikációs módszerek és eszközök 10
II.4. Helyességbizonyítások 14
II.5. A Hoare-módszer 15
II.6. Aszimptotikus függvények 18

III. A formalizmus bővítése **19**

IV. Példák **22**
IV.1. Első példa 22
IV.1.1. Első megoldás 22
IV.1.2. Második megoldás 24
IV.1.3. Harmadik megoldás 26
IV.2. Második példa 26
IV.2.1. Megoldás 28

V. Továbbfejlesztési tervek **31**
V.1. A programszintézis lehetősége 31
V.2. Más nyelvek 33
V.3. Egyéb tervek 33

VI. Összefoglalás **34**

A. Specifikációk **36**
A.1. Alapvető concept-ek 36
A.1.1. Equality Comparable 36
A.1.2. LessThen Comparable 37
A.2. Iterátorok specifikációja 37

A.2.1. Iterátorok hierarchiája 37
A.2.2. Típusok és kategóriák 37
A.2.3. Olvasó iterátorok specifikációja 38
A.2.4. Előre haladó iterátorok specifikációja 40
A.2.5. Kétirányú iterátorok specifikációja 40
A.2.6. Közvetlen elérésű iterátorok specifikációja 41
A.2.7. Iterátorok deklarációja 43
A.3. Konténerek specifikációja 45
A.3.1. A vector osztály 45
A.3.2. A stack osztály 49
A.3.3. A queue osztály 51
A.3.4. A priority queue osztály 53
A.3.5. A set osztály 54
A.3.6. A multiset osztály 60
A.4. Algoritmusok specifikációja 63

I. fejezet

Bevezetés

A professzionális C++ programok nagymértékben használják a szabványos könyvtár részét képező STL-t. Az STL a generikus programozási paradigmára alapuló, absztrakt adatszerkezetekből, és ezeken működő algoritmusokból álló (sablon-)osztálykönyvtár. A C++ nyelvi szabvány szövegesen adja meg az adatszerkezetek és algoritmusok definícióját, ami nem kívánt értelmezési problémákhoz vezethet.

Jelen dolgozat célja egy olyan formalizmus kialakítása, amely lehetővé teszi az STL komponensek absztrakt formalizmuson alapuló definícióját. Ezekkel a formalizációkkal helyességbizonyító eljárások végezhetőek, mind STL implementációk, mind STL-t használó programok bizonyítására is.

Általában egy módszertan nagy előnyének tekintik, a programozási nyelv- illetve környezet-függetlenséget, ilyen például [5]. Ezzel szemben itt egy nagyon is nyelvfüggő módszertanhoz jutunk el. A módszereknek figyelemmel kell lennie a C++ nyelvi sajátosságaira (például: *const*-tal való tagfüggvény túlterhelés, kivételkezelés, nem definiált változó értéke, stb.). Viszont ezt a nyelvfüggőséget több tény is inspirálja:

1. A C++ nyelv filozófiája sokszor nem fér bele az általános módszertanok kereteibe, gondoljunk például arra, hogy a legtöbb módszertan az „:=”-ség fogalmat triviálisnak veszi, a C++ filozófiájában viszont nem magától értetődő az *operator=* függvény illetve kezdeti értékadás esetén a másoló konstruktor helyessége.

2. A C++ – többek között az STL miatt – jóval bővebb szolgáltatásokat nyújt a programozó számára, mint a legtöbb imperatív nyelv, így hatékonyabb lehet egy specifikusabb módszertan.

3. Mivel az STL-t a generikus programozás előfutárának tartják, szinte nyilvánvaló, hogy egy általános *generikus programozási módszertan*t is

ezen kell kidolgozni első lépésként.

Porkoláb Zoltán [4] megemlíti, hogy egy általános generikus módszertant a korábbi procedurális és objektum-orientált alapok felett kell megvalósítani.

Mivel a Hoare-módszer[1] az axiómákat és a következtetési szabályokat a nyelv szemantikája alapján származtatja, ezért ideális kiinduló pont egy ilyen jellegű feladat során.

II. fejezet

Alapok

II.1. Generikus programozás

A generikus programozási paradigma egy új fajta megközelítés. Ennek az a lényege, hogy a programozási feladatokat visszavezetjük általános, újrafelhasználható komponensekre és könyvtárakra, amelyeket sablonként korábban már megvalósítottak[20]. A sablonok előnye, hogy sok célra felhasználhatóak, továbbá, hogy a hatékonyság csökkenése nélkül implementálhatók. Ezeket a sablonokat úgy példányosítjuk, hogy az adott feladatot minél precízebben oldja meg. Ezt a megközelítést használva nagymértékben csökkenthető egy szoftverkönyvtár bonyolultsága. A generikus programozást használva csökken a programok hossza, és a lehetséges programozói hibák száma is kevesebb lesz, ugyanakkor újrafelhasználhatóságot, és könnyebben érthető kódot is kapunk. A [20] cikkben is példaként szerepel, hogy egy könyvtár, amely tartalmaz n adatszerkezetet, mindegyik k alaptípussal és m algoritmussal, annak a bonyolultsága $O(n * k * m)$ az objektum-orientált paradigmát alkalmazva, míg a generikus paradigmával a bonyolultsága: $O(n + m)$.

A generikus programozásra a *generatív programozás* jelentős mértékben épít, és a paradigma végcélja, hogy a programkód teljesen automatikusan jöjjön létre [29].

A generikus paradigma az utóbbi időben annyira népszerű lett, hogy például a Java programozási nyelv esetén, mely korábban nem támogatta a sablonokat, 2003-ban a szabványt módosították, hogy lehessen sablonokat létrehozni[17].

Hasonlóan járt a C# nyelv is. A C# nyelv, amit a Microsoft alkotott meg a C++ és Java modernebb utódjaként, először nem támogatta a generikus programozást, de az újabb változatában már megtalálható ez a lehetőség is [27].

Ugyanakkor [28] cikk szerint a Java és a C# nem ad olyan komoly lehetőséget a generikus programozásnak, mint a C++ vagy a Haskell. A gond mindkét nyelv esetén a típusrendszerrel van, mivel a sablon metódus megkötéseiből nem tud következtetni a típus paraméterre. Emiatt ezek a nyelvek éppen csak megengedik a típussal való paraméterezhetőséget, de egyenlőre nem igazán alkalmasak a generikus programozásra. A C++ template mechanizmusa viszont bizonyítottan Turing-teljes [35].

II.2. A Standard Template Library

Az STL a C++ standard könyvtárának egy része, amely első megközelítésben három részből épül fel: tárolókból (containers), ezeken működő algoritmusokból, és bejárókból (iterators), amelyek biztosítják azt, hogy a tárolók és az algoritmusok együttműködjenek.

Néhány szerző finomítja ezt felosztást. Jeremy Gibbons[7] kibővíti a fenti felosztást egy függvényobjektumokból álló résszel és ezt a négyest egy irányított körmentes gráffal reprezentálja. Stroustrup[3] ezeket a következőképpen definiálja: azon osztályok objektumait, melyekhez elkészítettük a függvényhívó operátort, függvényszerű objektumnak, funktornak vagy egyszerűen függvényobjektumnak nevezzük. Előnyük, hogy sokszor gyorsabban futnak, mint a szokásos függvények, mert könnyebben lehet optimalizálni. Akkor hasznosak a függvényobjektumok, amikor valamely algoritmust saját függvénnyel szeretnénk példányosítani. Ugyanezt a felosztást használja Matthew H. Austern[21] is. Changqing Wang[9] még tovább megy az említett négyest még két komponenssel bővíti: allokátorok, melyek a memóriakezelést segítik, és az átalakítók (adapters), melyek új interfészt biztosítanak egy komponensnek.

A tárolókat és az algoritmusokat úgy találták ki, hogy elég általános célúak legyenek. Ezt házasították a C++ template eszközeivel, ezáltal egy hatékony, és sokszor használható eszköz legyen a programozó kezében.

Scott Meyers[2] az STL-t a szabványkönyvtár legforradalmibb részének tartja. Szerinte a felépítése, a rugalmassága, bővíthetősége, a szabvány miatti hatékonysága teszi nagyon jól használhatóvá. Szerinte az STL nem szoftver, hanem, *konvenciók* halmaza, és emiatt forradalmi. Dewhurst[8] viszont azt a tulajdonságát emeli ki, hogy a tárolók szerkezetükkel és működésükkel kapcsolatos döntések már fordítási időben megszületnek. Emiatt hatékony és kicsi kód készül, mely teljesítmény tekintetében pontosan alkalmazkodik az adott felhasználási módhoz. Bruce Eckel[19] az STL-nek azt a jó tulajdonságát is kiemeli, hogy teljesen platformfüggetlen.

A tárolók kapcsán Eckel[19] azt írja, hogy alapvető komponensei már az

objektum-orientált programozásnak is. Az STL adatszerkezeteinek további előnye az is, hogy a memóriafolyások (*memory leaks*) lehetősége is csökken, mivel a memóriakezelés gondját az STL készítői átveszik a programozótól [2]. A konténereket három további alosztályba lehet sorolni: szekvenciális, átalakító, és asszociatív tárolók. A vector, a deque, és a list tartozik az első kategóriába. Átalakítók a stack, a queue, és a priority queue osztályai. Az asszociatív tárolókhoz a map, a set, a multimap és a multiset tartozik. Nagyon fontos megjegyezni, hogy az átalakítóknak nincsenek iterátoraik, mivel ezzel szűkített interfésszel is elegendő szolgáltatásokat nyújtanak[3]. Emiatt az általános algoritmusok sem alkalmazhatóak ezeken a konténereken, mivel az algoritmusok iterátorokat használnak.

A vector osztály a hagyományos tömbök kiterjesztései, például a megszokott indexelés is használható. Lényegesen kényelmesebb a használatuk a megszokott C++ dinamikus tömbjeihez képest.

A deque a kettős végű sorok osztálya. Neve a double-ended queue szóból származik. A sor elején és végén képes konstans időben beszúrást és törlést végezni.

A list osztály a lista adatszerkezetet valósítja meg.

A stack a szokásos veremosztály megfelelője.

A priority_queue osztály a prioritásos sor megfelelője. Mivel össze kell hasonlítani az elemeket a prioritás miatt, szükséges, hogy legyen az osztályon egy $<$ reláció. Alapértelmezés szerint ehhez az `operator<` tagfüggvényt használja, de ez módosítható. Általában egy kupaccal[6] (heappel) reprezentálják az STL implementációk.

Mind a négy asszociatív tároló index szerint rendezve tárolja az elemeit. Ehhez szükséges, hogy legyen rendezés ($<$ reláció) értelmezve azon az osztályon, amit indexelésre használunk. A relációt – mint a priority_queue osztály esetében is – alapértelmezés szerint az `operator<` tagfüggvény definiálja.

A map tulajdonképpen az asszociatív tömbök osztálya. Index-érték párokat reprezentál. A garanciák miatt elég hatékonynak kell lennie, ezért általában egy kiegyensúlyozott keresőfával reprezentálják a különböző STL implementációk. Pontosabban elterjedt a piros-fekete fákkal [6] reprezentált map. Stroustrup[3] szerint ez az egyik leghasznosabb felhasználói típus. Egy indexhez csak egy érték tartozhat.

A set adatszerkezet a hagyományos halmaz fogalomnak felel meg, tehát az adatokat multiplicitás nélkül tárolja. Stroustrup[3] úgy definiálja ezt az osztályt, hogy egy olyan map, amelyben nincs érték, csak a map indexhalmaza.

A multimap egy olyan asszociatív tömb,amelyben egy indexhez több érték is tartozhat, tehát egy multiplicitást kezelő map. Bjarne Stroustrup[3] szerint

a multimap használata általában tisztább és elegánsabb megoldást ad, mint a mapé.

A multiset – a multimaphez hasonlóan – olyan set, amely multiplicitást is kezeli. Ezt szokás zsáknak is nevezni.

Önmagukban a tárolók is hasznosak, de az algoritmusokkal együtt nyújtanak igazán jó teljesítményt. Ezeket az algoritmusokat függvénysablonként valósítják meg. Az STL számos ilyen algoritmust biztosít. Az algoritmusok együttműködnek a konténerekkel, ezek alól leginkább az átalakítók képeznek kivételt. Az STL az iterátorok segítségével valósítja meg azt, hogy ugyanaz a függvénysablon fut le tárolótól függetlenül.

Az algoritmusoknak is több típusa van: nem módosító sorozatműveletek (például: mismatch(), find_if(), for_each()), sorozatmódosító műveletek: replace_if(), reverse(), replace_if(), stb., rendezett sorozatok műveletei: merge(), binary_search(), sort(), stb., halmazműveletek: set_symmetric_difference(), includes(), set_intersection(),stb., kupacműveletek: make_heap(), push_heap(), pop_heap(), sort_heap(), a minimum és maximum műveletei: min_element(), max_element(), lexicographical_compare(), stb. és a permutációk: next_permutation() és prev_permutation(). Egészen pontosan 60 darab algoritmusa van az STL-nek [3].

A fenti osztályok és függvények az std névtérben találhatóak, a dolgozat során végig feltesszük, hogy ezeket használjuk, és eltekintünk a névtér kiírásától.

Az iterátorok biztosítják a különböző tárolókon való bejárást. Gibbons[7] és Austern[21] az iterátorokat a pointerek absztrakciójaként interpretálja. Ezzel szemben Stroustrup nem osztja ezt a véleményt, inkább a tömböknél alkalmazott mutatókkal próbálja párhuzamba állítani a bejárokat. Eckel[19] pedig úgy fogalmazza ezt meg, hogy az iterátorok azok az absztrakciók, amelyek megengedik, hogy a kód generikus lehessen, mivel azok semmit sem tudnak a tárolók szerkezetéről. Levonhatjuk azt a következtetést, hogy az iterátorokat még informálisan is nehéz meghatározni. Három nagyon fontos fogalom kötődik az iterátorokhoz: az indirekció, a növelés, és az egyenlőségvizsgálat.

Az indirekció (* vagy -> szintaxissal) tudja az iterátor által mutatott elemet használni.

A növelés (++ jelöléssel) a következő elemre lépteti az iterátort.

Az egyenlőségvizsgálattal (==) két iterátor egyezőségét lehet összehasonlítani. Ez egy ekvivalenciarelációt határoz meg.

Az iterátorokat öt különböző kategóriába lehet sorolni: író, olvasó, előre haladó, kétirányú, közvetlen elérésű kategóriák léteznek. Ezek a kategóriák meghatározzák, azt hogy milyen műveletet lehet végezni a bejáróval. Ez a kategorizálás egy hierarchiát is eredményez. Minden kategória esetén értelmezik

a const_iteratort és az iteratort. A const_iterator abban különbözik az iteratortól, hogy azon keresztül a konténer nem módosítható. Ebből az is rögtön következik, hogy egy const_iterator nem konstans. Az író és olvasó iterátorokat leginkább az STL algoritmusai használják

A különböző konténerek iterátorait különböző típusként értelmezi a C++ nyelvi szabvány. A különböző konténerek bejárói meghatározott kategóriába esnek, például a vector-nak és a deque-nak véletlen elérésű iterátorai vannak, a list-nek csak kétirányú bejárói vannak, az összes asszociatív konténernek pedig kétirányú const_iterator-ai vannak. Ezeknek azért kell const_iteratoroknak lenniük, hogy rajtuk keresztüli értékadás ne ronthassa el a tároló rendezettségét.

II.3. Ismert STL specifikációs módszerek és eszközök

Az STL formális eszközökkel definiálásával a legtöbbet David R. Musser foglalkozott, például [11],[12],[13] [14],[15],[16]. Ő és a vele közreműködő kutatók több módszert is kidolgoztak: a Tecton rendszert [11],[12],[13], [14], amely az algebrai specifikációt[1] veszi alapul, egy új, könnyűsúlyú módszert ad a formális elemzéshez, amit dinamikus verifikációnak neveztek.

A Tecton nyelv definíciója megtalálható a [11] munkában. A Tecton rendszerben Musser megadta a konténerek és az iterátorok formális leírását [12]. A rendszerben precízen megtalálhatóak az STL komponensek definíciója, mégis az algebrai specifikáció körülményes volta miatt nehezen kezelhető. Szintén probléma a Tecton jellegű algebrai specifikációkkal, a bizonyítások is komplikáltabbak az elő-, utófeltételes bizonyításokhoz képest. Gyakori módszer a *struktúrális indukció* [1], amely az STL méreteit figyelembe véve, kényelmetlenül használható. Emiatt kidolgoztak rajta egy bizonyítási rendszert, amelyet felkészítettek az automatikus helyességbizonyításra is [14]. Ez a rendszer a bizonyításokat fákként reprezentálja. A rendszer a ,,közönséges" és a ,,Hoare"-logikát is használja, az előbbit a kapcsolatok leírására, az utóbbi pedig a program dinamikus viselkedését írja le, azáltal, hogy összekapcsolja a programrészeket és a logikai formulákkal.

A módszert egy MELAS-nak (MEta-Level program Analysis System - metaszintű programelemző rendszer) nevezett rendszerrel támogatták meg. A rendszer támogatja az STL dinamikus verifikációját és tesztelését metaszinten. A módszert a tesztelés ötlete adta, és a compileren és a gdb debuggeren keresztül valósítják meg. Ezt úgy valósítják meg, hogy a tesztelendő algoritmust először Prolog nyelven specifikálják. Ezt követően ebből

a specifikációból C++ nyelven készítenek egy sablon osztályt, amelynek három tagfüggvénye van: precond, aminek a paraméterei megegyeznek a tesztelendő algoritmus paramétereivel és logikai értéket ad vissza, a postcond tagfüggvény, ugyanezzel a prototípussal, és az update függvény. A precond függvény ellenőrzi, hogy az előfeltétel igaz-e, a postcond függvény az utófeltétellel teszi ugyanezt. Az update függvény elvégzi a specifikáció végrehajtását az eredeti függvény helyett. Egy programban ezt példányosítják, ellátják a teszteléshez szükséges kiegészítésekkel. A programot lefordítják, és a gdb debuggeren keresztül megkezdődik a verifikáció. Először a MELAS-sal kapcsolatos események hajtódnak végre, például betölti az elemző adatbázist és a tény adatbázist. Innentől kezdve ciklikusan működik a rendszer. Meghívja a példányosított objektum precond tagfüggvényét, hogy eltárolja a megelőző állapotot az elemzés végéig. Ellenőrzi, hogy igaz-e az előfeltétel, ha igen folytatódik a verifikáció, ha nem igaz, akkor leáll. Ha ismeretlen lenne az eredménye, azt igaznak veszi. Ezután a ciklus feltételt értékeli a program, ehhez vagy a felhasználó adja meg input adatot, vagy pedig az esetelemző modul. Ezután az utófeltételt ellenőrzi a rendszer. Ha igaz, akkor következő input adattal folytatja az elemzést. Ha hamis az utófeltétel, akkor ciklust nem fogadják el. Ezt követően meghívja az update függvényt, és elmentik az aktuális állapotot. Ezzel ellenőrzik az indukciós feltevést. Végül újra kiértékelik az utófeltételt. A merge() függvény ilyen jellegű tesztelése megtalálható [9] dolgozatban, a copy() függvényé pedig a [16] cikkben.

A problémákat a Prolog nyelvre vezetik vissza, ami mivel hogy *logikai* nyelv a helyességbizonyítás lényegesen egyszerűbb. A Prolog nyelvet az automatikus helyességbizonyítás megvalósítása miatt hozták létre. A Prolog nyelv előnye továbbá az is, hogy *specifikációs* és *deklaratív* nyelvként is használják [18].

David Musser és Changqing Wang is kiterjesztette a Hoare-módszert, és megadta az elő-, utófeltételes specifikáció lehetőségét a [15] cikkben. Megadtak egy lehetséges axiómarendszert következtetési szabályokkal, amivel bizonyítható egy STL implementáció. Changqing Wang[9] munkájában a különböző módszerek összehasonlítása is megtalálható.

Austern is elő-, utófeltételes módszerekkel írta le az STL-t a könyvében [21]. De ő nem precíz matematikai formulákat használt, hanem informális módszereket alkalmazott. Például nézzük meg hogy, hogyan írta le az asszociatív tárolók intervallum törlési műveletét:

a.erase(p,q)
Visszatérési érték típusa: void
Előfeltétel: [p,q) érvényes intervallum az a objektumban.
Szemantika: Megszünteti a [p,q) intervallumba eső elemeket, és kiveszi őket

az a objektumból.
Utófeltétel: Az a objektum mérete csökken distance(p,q)-val.

Vagy például a szekvenciális tárolók push_back() művelete:

a.push_back(t)
Típus követelmény: az a objektum módosítható legyen
Visszatérési érték típusa: void
Szemantika: ekvivalens a.insert(a.end(), t) utasítással.
Az ilyen jellegű módszerek nagy hibája, hogy nem alkalmasak helyességbizonyításra.

Az invariánsokat is elég körülményesen határozta meg Austern, például a vector konténerre az alábbiakat követeli meg:

- *[a.begin(),a.end()) érvényes intervallum*
- *a.size()==distance(a.begin(), a.end())*
- *minden algoritmus, ami az [a.begin(),a.end()) intervallumon fut, az a objektum minden elemét pontosan egyszer éri el.*
- *ha egy push_back műveletet egy pop_back követ, akkor ez így együtt egy nullműveletet eredményez.*

Jeremy Gibbons is megadott egy lehetséges formalizmust az STL komponensekhez [7]. Ő erre a problémára egy olyan jellegű megoldást adott, ami az algebrai specifikációra épít, felhasználva a Haskell funkcionális nyelvet is. Ennek a módszernek előnye a formalizmus matematikai eszközökkel való elemezhetősége, beleértve a helyességbizonyítást is. Az eddigi munkája látványos, bár eddig a formalizmussal csak Haskell adatszerkezeteket írt le a publikációiban. Sajnos, helyességbizonyítási módszerekre sem adott példát, emiatt félő, hogy ő a nehézkes struktúrális indukciót választja bizonyítási módszerként. Nagyon szép gondolat, hogy ő a *tervmintákat* (design patterns) is beleveszi a formalizálandó témákhoz. Az ötlete onnan származik, hogy a C++ STL iterátorai és az iterator tervminta nagyon közel állnak egymáshoz. Az ő végcélja is megcsinálni egy módszertant a generikus programozáshoz.

Douglas Gregor írt egy programot az STL hibáinak felderítésére, amit *STLlint*-nek nevezett [26]. Bizonyos hibák kiszűrhetőek a program alkalmazásával, amelyekre a C++ fordítóprogramok nem jeleznek hibát, de használatuknak nem kívánatos mellékhatásai lehetnek. A program a következő jellegű hibákat tudja észrevenni: intervallumok érvényessége, rendezéssel kapcsolatos tulajdonságok, heap-pel kapcsolatos tulajdonságok, eltávolítással

kapcsolatos esetek, az STL komponenseinek alapvető szemantikai tulajdonságai. A program az interneten online módon használható: el kell küldeni a C++ kódot, és a program azonnal megadja a hibákat.

Az STLlint úgy működik, hogy a C++ kódban az összes STL-es utasítás helyére egy konkrét implementációt helyettesít. Az így átalakított kódot kiegészítik a vizsgálathoz szükséges ellenőrzéssekkel. Ehhez a C/C++ *assert*-jeit használta Douglas Gregor. Ez egy olyan utasítás, amivel egy logikai kifejezéstől függően vagy folytatódik a program végrehajtása, vagy egy hibaüzenettel leáll.

A program figyelmeztetést adott arra, amikor egy üres sor front-jának próbáltam értéket adni, észrevette azt a hibát is, amikor egy iterátort inkrementáltam mielőtt értéket adtam volna a bejárónak. Viszont a következő utasítást jónak tartotta az STLlint: `s.erase(s.end());`. Itt s egy set<int> objektum volt, tehát egy egészeket tartalmazó halmaz. Az utasítás viszont nem helyes, eredménye nem definiált. Érdekes probléma, hogy a for_each algoritmussal, amely sorozat nem módosító típusú, megoldható, hogy módosítsa a sorozatot. Ez a nyelvi szabványnak ellentmond, de az STLlint és a különböző fordítóprogramok viszont minden megjegyzés nélkül elfogadják, sőt Austern sem specifikálja helyesen[21]. Hasonló igaz, például a find_if algoritmusra is, de ott az STLlint hibajelzést adott. Ilyen esetek miatt az STLlint még nem tökéletes, folyamatosan javítja Douglas Gregor.

A [26] cikk tartalmaz egy érdekes statisztikát arról, hogy melyek a leggyakrabban elkövetett hibás utasítások az STL használatakor. A cikk szerint legnagyobb százalékban az a probléma, amikor egy iterátor egy algoritmuson keresztül kap értéket (például find-dal) és ezt követően anélkül dereferálják a bejárót, hogy megvizsgálnák, hogy az nem mutat-e a bejáró végére.

Jeremy Siek és Andrew Lumsdaine írt egy könyvtárat, aminek a neve *Boost Concept Check Library* (BCCL) [33]. Ez a Boost rendszer része, amely sok hasznos különböző könyvtárat tartalmaz. Ezek a könyvtárak ingyenesen letölthetőek a Boost honlapjáról.

A BCCL sok szempontból megkönnyíti az STL-t használó programozók dolgát, például a nehezen érthető hibaüzenetek helyett, szebb és érthetőbb formában közli a fordítási hiba mibenlétét. Ezenkívül a könyvtár lehetőséget ad *concept*-ek specifikálására, template paraméterek, mint concept verifikálására, így az STL conceptjeire is.

Egy concept követelményeknek egy adott halmaza. Ez magába foglalja az érvényes kifejezéseket, szemantikus invariánsokat, műveletigény garanciákat stb.. Egy conceptnek ezeket teljesítenie kell, ezért hogy egy adott osztály vagy objektum teljseíti-e ezeket, ellenőrizni kell. Például, az STL különböző kategóriájú iterátorai, más-más concept-nek foghatók fel. Egy véletlen elérésű bejárónak több dolgot kell teljesítenie, mint egy előrehaladónak. Az STL

különböző konténereinek és algoritmusainak használatához ezeket teljesíteni kell, például, hogy a stable_sort függvény véletlen elérésű bejárót kap inputként vagy, hogy a set konténer esetében a példányosítandó osztály elemei összehasonlíthatóak.

A BCCL a conceptek ellenőrzését úgy végzi, hogy template paraméterként megkapja azt az osztályt, amit verifikálni próbál. A concepteket *struct*-ként lehet megvalósítani. Ezeknek a concept osztályoknak van egy constraints() nevű tagfüggvénye, ez tartalmazza az érvényes kifejezéseket. Ha egy osztályt próbálunk verifikálni, akkor ezt ellenőrzi. Ellenőrizni tudja a függvénykövetelményeket is, ezt a function_requires() függvénnyel lehet megtenni. A könyvtár természetesen tartalmazza az összes ilyen STL-lel kapcsolatos conceptet. Egy hasonló elképzelésről olvashatunk a [34] cikkben is.

Egy UML (Unified Modeling Language) alapú specifikáció található a [10] jelentésben. Itt specifikálták STL minden részletét egy kiterjesztett UML alapú jelölésrendszerrel. Leírtak egy STL implementációt ugyanezzel a jelölésrendszerrel, és a kettőt összehasonlították. Ez a módszer – sajnos – nem ad lehetőséget a precíz bizonyításra. STL-t használó programok helyességéről sem ad semmilyen információt. Az UML nyelv ugyanis nem támogatja a procedurális elemeket.

II.4. Helyességbizonyítások

Helyességbizonyítás alatt egy *program* adott *specifikáció* helyességének matematikai eszközökkel való bizonyítást értünk.

Rendszerint, két alapvetően különböző típusú eljárás létezik: az egyik esetben a specifikációból indulunk ki, a másik esetben pedig a programból. Mindkét esetben nagyon fontos a pontos és minél teljesebb specifikáció, de általában a gyakorlat azt mutatja, hogy mindig van olyan eset, amit a specifikáció nem vesz figyelmbe[22].

A számítástudomány keretein belül több ilyen eljárást is kidolgoztak: például a Floyd módszerét[1], a Hoare-módszert[1], a relációkon alapuló programok helyességét[5], stb..

A módszerek előnyeinek tekintik[23], hogy a specifikáció formális elemezhetőségét, a pontos megoldást biztosít az absztrakt szinten a programkészítés korai fázisában és a program helyességének bizonyíthatósága az absztrakt leírás szerint.

Helyességbizonyításhoz általában három fontos eszköz kell: program (egy adott nyelven), specifikáció (ezek általában a predikátumkalkulus[18] formulái) és a kalkulus.

A legtöbb esetben a program nyelve általános sémák halmaza szokott lenni, például: stuktogramm, vagy a blokkstruktúra nyelv.

Egy kalkulus esetén fontos kérdés annak helyessége és teljessége[18]. Helyesség alatt azt értjük, hogy minden, amit le lehet vezetni a kalkulussal, az helyes is, és teljesség pedig azt jelenti, hogy minden ami helyes, az le is vezethető a kalkulus szabályaival. A helyesség nyilvánvalóan fontos, ahhoz, hogy jól működjön a kalkulus, tehát ennek feltétlenül teljesülnie kell, ellenben a teljességgel. Például [5] egy olyan kalkulust mutat be, amire csak a helyesség áll fenn. Ott ugyanis a ciklus levezetési szabályának megfordítása csak bizonyos további feltételek fennállása esetén igaz.

Hasonlóan működik a helyességbizonyítás nem imperatív nyelvek esetén is, de teljesen eltérő eszközöket alkalmaznak. Például (párhuzamos és elosztott) funkcionális nyelvek esetén szokás alkalmazni a temporális logikát. Az Erlang nyelv esetén a modális μ-kalkulust használják specifikációra, bár a nyelv nem tisztán funkcionális[25], [24].

II.5. A Hoare-módszer

A Hoare-módszer[1] lényege, hogy a matematikai logikában tételek bizonyítására használt deduktív módszert alkalmazza a programok helyességének bizonyítására. Ez azt jelenti, hogy a programok helyességére vonatkozó tételek az axiómákból következtetési szabályok segítségével bebizonyíthatók. Az axiómákat és a következtetési szabályokat a nyelv szemantikája alapján származtatjuk. A bizonyításokat az elő-, utófeltételes formával végezhetjük el.

Axiómák:

Üres utasítás (skip) axiómája:

$$\{P(x,y)\}skip\{P(x,y)\}$$

Az értékadás axiómája:

$$\{P(x,g(x,y))\}y \leftarrow g(x,y)\{P(x,y)\}$$

Következtetési szabályok:

Az S_1;S_2 szekvencia következtetési szabálya:

$$\frac{\{P\}S_1\{Q_1\} \text{ és } \{Q_1\}S_2\{Q\}}{\{P\}S_1;S_2\{Q\}}$$

Az if α then S_1 else S_2 fi elágazás szabálya:

$$\frac{\{P \wedge \alpha\}S_1\{Q\} \text{ és } \{P \wedge \neg\alpha\}S_2\{Q\}}{\{P\}if\ \alpha\ then\ S_1\ else\ S_2\{Q\}}$$

A while α do S od iteráció következtetési szabálya:

$$\frac{\{P \wedge \alpha\}S\{P\} \text{ és } (P \wedge \neg\alpha) \Rightarrow Q}{\{P\}while\ \alpha\ do\ S\ od\{Q\}}$$

A következmény szabálya:

$$\frac{(P \Rightarrow P_1) \text{ és } \{P_1\}S\{Q_1\} \text{ és } (Q_1 \Rightarrow Q)}{\{P\}S\{Q\}}$$

További szabályok:

Az értékadás következtetési szabálya az axióma és a következmény szabályának felhasználásával a következő formára hozható:

$$\frac{P(x,y) \Rightarrow Q(x,g(x,y))}{\{P(x,y)\}y \leftarrow g(x,y)\{Q(x,y)\}}$$

A szekvencia következtetési szabályának egy általános formája a következmény szabályának felhasználásával:

$$\frac{P \Rightarrow P_1 \text{ és } \{P_1\}S_1\{Q_1\} \text{ és } Q_1 \Rightarrow P_2 \text{ és } \{P_2\}S_2\{Q_2\} \text{ és } Q_2 \Rightarrow Q}{\{P\}S_1; S_2\{Q\}}$$

Az elágazás következtetési szabályának általános formája:

$$\frac{P \Rightarrow P_1 \text{ és } \{P_1 \wedge \alpha\}S_1\{Q_1\} \text{ és } \{P_1 \wedge \neg\alpha\}S_2\{Q_1\} \text{ és } Q_1 \Rightarrow Q}{\{P\}if\ \alpha\ then\ S_1\ else\ S_2\ fi\{Q\}}$$

Az elágazás következtetési szabálya speciális esetben:

$$\frac{P \Rightarrow P_1 \text{ és } \{P_1 \wedge \alpha\}S\{Q_1\} \text{ és } (P_1 \wedge \neg\alpha) \Rightarrow Q_1 \text{ és } Q_1 \Rightarrow Q}{\{P\}if\ \alpha\ then\ Sfi\{Q\}}$$

Az iterációnál bevezetve a ciklus invariánsát a következmény szabályának felhasználásával kapjuk a következő általános formát:

$$\frac{P(x,y) \Rightarrow I(x,y) \text{ és } \{I(x,y) \wedge \alpha(x,y)\}S\{I(x,y)\} \text{ és } I(x,y) \wedge \neg\alpha(x,y) \Rightarrow Q(x,y)}{\{P(x,y)\}while\ \alpha(x,y)\ do\ S\ od\{Q(x,y)\}}$$

A teljes helyesség bizonyításának következtetési szabálya:

$$P(x,y) \Rightarrow I(x,y) \text{ és } I(x,y) \Rightarrow E(x,y) \in W_< \text{ és}$$

$$\{I(x,y) \wedge \alpha(x,y) \wedge E = E(x,y)\} S \{I(x,y) \wedge E(x,y) < E\} \text{ és}$$

$$\frac{I(x,y) \wedge \neg\alpha(x,y) \Rightarrow Q(x,y)}{\{P(x,y)\} while\ \alpha(x,y)\ do\ S\ od \{Q(x,y)\}}$$

Az iteráció következtetési szabályának gyakran azt a formáját használjuk, amikor a W=$\mathbb{N}_0$ és E(x,y)=i, ami a ciklus számlálót jelenti. Ekkor a következtetési szabály:

$$P(x,y) \Rightarrow I(x,y,0) \text{ és } I(x,y,i) \Rightarrow i < k(x) \text{ és}$$

$$\{I(x,y,i) \wedge \alpha(x,y)\} S \{I(x,y,i+1\} \text{ és}$$

$$\frac{I(x,y,i) \wedge \neg\alpha(x,y) \Rightarrow Q(x,y)}{\{P(x,y)\} while\ \alpha(x,y)\ do\ S\ od \{Q(x,y)\}}$$

Tétel: A Hoare-módszer következtetési szabályai helyesek.
Bizonyítás: A tételt nem bizonyítjuk. A bizonyítás megtalálható [1]-ben.

Legyen adott az $S(x,y)$ struktúrált program, amelynek egy tetszőleges részprogramját $s(x,y)$ jelöli. Tegyük fel, hogy igaz a következő tétel minden ilyen $s(x,y)$ részprogramra:

$$\{Q(x) \wedge y = I_s(x)\} s(x,y) \{Q(x) \wedge y = O_s(x)\},$$

ahol

$$f_s(x, I_s(x)) = O_s(x),$$

azaz $I_s(x)$ jelöli az $s(x,y)$ függvény bemenő és $O_s(x)$ a hozzá tartozó eredmény adatát a program x bemenő paraméterei mellett.

Tétel: Minden ilyen tulajdonságú részprogramra vonatkozó fenti tétel, a Hoare-módszer segítségével bebizonyítható.
Bizonyítás: A tételt nem bizonyítjuk. A bizonyítás megtalálható [1]-ben.
Ez utóbbi tétel mondja ki a Hoare-módszer teljességét.

Owicki-Gries módszer néven ismert a Hoare-módszer párhuzamos változata. Nekünk most nincs rá szükségünk, mivel a standard C++ nem támogatja a párhuzamos programok írását. Ennek a módszernek a leírása is megtalálható [1]-ben.

A különböző STL implementációk helyességének bizonyításához is ad módszert [1]: Procedurálisan adott konkrét specifikáció elő-, és utófeltételekkel adott absztrakt specifikáció helyessége című fejezetben található a következő tétel:
Adottak a d_a és a d_c specifikációk közös szignatúrával:

$$d_a = (A, F, E); ahol f_i \in F;$$
$$\{true\}a = f_0\{post_{f_0}(a)\},$$
$$\{pre_{f_i}(a)\}b = f_i(a)\{post_{f_i}(a,b)\} \in E_a, i = 1, 2, \ldots, n;$$
$$d_c = (C, G, E_C); Q_{g_i} \in E_C, g_i \in G, i = 1, 2, \ldots, n;$$

Legyen az absztrakt invariáns

$$A = \{a|\mathcal{I}_a(a)\},$$

a konkrét invariáns

$$C = \{c|\mathcal{I}_c(c)\}.$$

Legyenek a konkrét szemantika eljárásai a következők:
procedure g_0 begin Q_0 end;
procedure g_i begin Q_i end; i=1,2,...,n;
A reprezentációs függvény:

$$\varphi: C \to A.$$

Ha a következő tételek teljesülnek:
1. $(\forall c \in C)(\mathcal{I}_c(c) \Rightarrow \mathcal{I}_a(\varphi(c))$;
2. $\{true\}Q_0\{post_{f_0}(\varphi(c)) \wedge \mathcal{I}_c(c)\}$;
3. $(\forall f \in F) : \{pre_{f_i}(\varphi(c)) \wedge \mathcal{I}_c(c)\}Q_i\{post_{f_i}(\varphi(c), \varphi(c')) \wedge \mathcal{I}_c(c')\}$;
ahol a 2. és 3. a teljes helyességi tételek, akkor a d_c konkrét specifikáció helyes a d_a absztrakt specifikáció szerint.
Bizonyítás: a tételt nem bizonyítjuk. A bizonyítás megtalálható [1]-ben.

II.6. Aszimptotikus függvények

A C++ nyelv szabványa *garanciákat* biztosít az STL-ben szereplő függvények aszimptotikus műveletigényére. Ezért fontos a következő definíció:

$f(n) = O(g(n)),\ ha\ \exists n_0\ \ és\ \ c \in \mathbb{N}\ \ úgy,\ \ hogy\ \forall n > n_0 : f(n) \leq cg(n)$

A többi aszimptotikus függvény definíciója, és tulajdonságaik leírása megtalálható [6]-ban.

III. fejezet

A formalizmus bővítése

Az STL formális specifikációja előtt, be kell vezetni egy egységes formális eszköztárat, amivel az leírható.

Először bemutatom az elő-, utófeltételes specifikáció[1] leírását:

$$\{P\}S\{Q\}$$

jelöli a továbbiakban, azt hogy P *előfeltétel* (precondition) esetén S program hatására a Q *utófeltételhez* (postcondition) jutunk. A [30] jegyzet fogalmát használva a Q a P S-re vonatkozó *legszigorúbb utófeltétele*. A P és Q leírásához az elsőrendű logikai[18] kifejezéseket használunk. Az S program a továbbiakban szintaktikusan helyes C++ utasítás vagy utasítássorozat. Mint már jeleztem, a `std::` névtér kiírásától eltekintünk.

Nézzünk egy egyszerű példát!

$$\{true\}int\ a = 1; \{a = 1\}$$

$$\{a = 1\} + +a; \{a = 2\}$$

A C++ nyelv több olyan nyelvi konstrukciót tartalmaz, amelyet le kell tudnunk írni:

A deklarált, de nem definiált változó leírására a ? szimbólumot használjuk:

$$\{true\}int\ a; \{a =?\}$$

Szükség esetén ezek indexelhetők is:

$$\{true\}$$

$$int\ a;$$

$$int\ b = a;$$

$$int\ c;$$

$$\{a =?_1 \wedge b =?_1 \wedge c =?_2\}$$

$Undef$ szimbólummal jelöljük azt, hogy az utasításnak nincsen értelmes eredménye, lehet, hogy a program futás idejű hibával elszáll, de az is lehetséges, hogy valamilyen eredményt ad, aminek nincs értelme. Ilyenkor nem definiált a program működése.

Ha a program olyan utasítást hajt végre, ami egy kivételt(exception) vált ki, azt $Exc(a)$ módon jelöljük, ahol a a kivétel.

Ha egy utófeltételben több függvény szekvenciális végrehajtását kell leírni, azt a $SEQ(f(a), g(b))$ kifejezés fogja leírni, tehát ez azt jelenti, hogy először az $f(a)$ függvény lefut, aztán pedig a $g(b)$. Ez például a for_each() függvény specifikálásánál lesz fontos.

Egy adott osztály esetén $\mathcal{I}$-vel jelöljük az osztályhoz tartozó típus- vagy osztályinvariánst.

A definiáló egyenlőség fogalma a C++-ban eléggé eltér a más nyelvekben megszokott konstrukcióktól. Amikor az `=`-t írunk C++-ban, két különböző függvény hajtódhat végre a háttérben a környezettől függően. A másoló konstruktor fut le, amikor deklarálunk és értéket is adunk az objektumnak vagy változónak, egyszerű értékadás esetén pedig az `operator=` tagfüggvény. Mivel nem POD[1] osztály esetén ezek alapértelmezés szerint nem helyesek, és ha valaki nem írja meg helyesen a fenti függvényeket, az értékadások nem működnek helyesen. Mivel a template-ek fogalma erősen kötődik a típussal való paraméterezhetőséghez, és a modellünk statikus, ezért nem tehetjük fel, hogy az értékadások helyesek. Az STL osztályainál és a beépített egyszerű típusok esetén a problémát megoldották, ezért ott nem kell figyelembe venni. A problémát úgy hidaljuk át, hogy a fenti függvényekre vezetjük vissza az `=` műveleteket. A $operator = (param)$ esetén a T típus `operator=` függvénye szerinti értékadást értjük, a $cpctor(param)$ esetén pedig a másoló konstruktor függvény szerinti értékadást jelezzük.

Használjuk továbbá a χ függvényt. Ennek definíciója:

$$\chi(l) = \begin{cases} 1 & \text{ha } l \text{ igaz} \\ 0 & \text{különben} \end{cases}$$

ahol l tetszőleges logikai kifejezés.

Az $x_1, x_2, \ldots, x_n$ elemek összes permutációjának a halmazát $perm(x_1, x_2, \ldots, x_n)$-nel jelöljük.

[1]POD (*Plain Old Data)* osztály: olyan osztály, amely az adattagjai bitenkénti másolásával pontosan az eredeti objektum másolatát kapjuk.

Az iterátorok esetén alkalmazott jelölések

A bejáróknak sok fajtája van, ennek a formális leírásban meg kell jelennie. A $const\_it(it)$ jelöli, azt hogy az it azonosítójú iterátor const_iterator. Ha egy it bejáró nem const_iterator, azt emiatt úgy jelöljük, hogy $\neg const\_it(it)$. Hasonlóképpen $rev\_it(it)$ az iterátor haladási irányát írja le. Ebben az esetben az it iterátor hátrafelé halad, és az előrefele haladó iterátorra $\neg rev\_it(it)$ jelölést használjuk.

A bejárók legfontosabb tulajdonsága, az hogy egy konténer elemére mutatnak. Az it nevű iterátor által jelölt elemet $*it = x_j$ kifejezés írja le. Ha a konténer végére mutat: $*it = x.end$, itt x a konténer objektum azonosítója. Ha hátrafele halad a bejáró: $*it = x.rend$

A bejáróknak létezik kategóriája is, ezt $cat(it)$ függvény fogja kifejezni. Ennek a függvénynek 5 értéke van: In/Out/For/Bi/Ran. Az In (input) olvasó iterátorok kategóriája, az Out (output) az író bejáróké. A For (forward) iterátorok kategoriája, azaz az előre haladó iterátorok, a Bi (bidirectional) a kétirányú iterátorok és végül a Ran (random-access) pedig a véletlen elérésű iterátorok kategóriája.

A C++ erősen típusos nyelv, élesen megkülönbözteti a különböző tárolótípushoz tartozó bejárótípusokat. Emiatt egy bejáróról azt is tudni kell, hogy milyen típusú tárolóhoz tartozik. A $type(it)$ függvénnyel fejezzük ki ezt.

A tárolók esetén alkalmazott jelölések

A tárolókat arra találták ki, hogy egy adott típushoz tartozó objektumok legyenek bennük. Más szóval objektumok egy sorozata egy tároló. Az $< x_1, x_2, \ldots, x_n >$ jelöl egy n elemből álló sorozatot. Az üres sorozat jele értelemszerűen: $<>$.

Egy tárolóobjektum lehet konstans és nemkonstans is. A C++ nyelv megengedi a *const*-tal való tagfüggvény túlterhelést, azaz más függvény futhat le egy konstans és egy nemkonstans objektumon azonos függvényhívás esetén is. Ha egy x objektum konstans, azt így fogjuk jelölni: $const(x)$, és $\neg const(x)$, ha nem az.

A map és a multimap tárolók olyan konténerek, amelyek párokat tartalmaznak. A pároknak van egy index komponense és egy érték komponense. Ezeket így fogjuk jelölni: $m =< p_1, p_2, \ldots, p_n >$ jelöli az n darab párt. Egy $p_j = i_j : x_j$, ahol i_j az indexe a j-edik párnak, és x_j az értéke a párnak. Egy iterátor ebben az esetben egy párra mutat. A $(*it).first$ jelöli a pár index komponsensét, és $(*it).second$ az értékét.

IV. fejezet

Példák

IV.1. Első példa

Adott a következő probléma! Adott egy szó, amelyet szeretnénk megfordítani és a megfordított szó betűit egy vector-ban tárolni. A betűket a felhasználó adja meg inputként, és sorvége jelzi a szó végét. Az általánosság megszorítása nélkül feltehető, hogy a szó $n \geq 0$ betűből áll.

IV.1.1. Első megoldás

Első megoldásnak az juthat az eszünkbe, hogy egy verem és egy ciklus segítségével könnyen meg lehet fordítani a szó betűit.

Tekintsük az alábbi megoldást!

```
#include <stack>
#include <vector>
#include <iostream>
using namespace std;

int main() {

  char betu;
  cin.get(betu);
  stack<char> original;
  vector<char> megford;

  // Betuk beolvasa:
  while (betu!='\n') {
     original.push(betu);
```

```
    cin.get(betu);
  }

  // Megfordítás:
  while (!original.empty()) {
    megford.push_back(original.top());
    original.pop();
  }

  return 0;
}
```

Most a megoldás első részével a szó beolvasásával nem foglalkozunk, könnyen meggondolható, annak a helyessége.

Lényegében a program előfeltétele:
$\varphi = \{original =< b_1, b_2, \ldots, b_n > \wedge \neg const(original) \wedge megford =<> \wedge$
$\wedge \neg const(megford) \wedge n \geq 0\}$
És az utófeltétele:
$\psi = \{megford =< b_n, b_{n-1}, \ldots, b_1 > \wedge original =<>\}$
Először bizonyítsuk be a parciális helyességet!
Ehhez kell a ciklus szabályának a bizonyítása:
A ciklus parciális helyességének bizonyításához invariánsra van szükségünk:

$\mathcal{I} = \{original =< b_1, b_2, \ldots, b_i > \wedge \neg const(original) \wedge$
$\wedge megford =< b_n, b_{n-1}, \ldots, b_{i+1} > \wedge \neg const(megford)\}$

Ekkor a következő nyilvánvalóan igaz:
$\varphi \Rightarrow \mathcal{I}$ és
$\{\mathcal{I} \wedge original.empty()\} \Rightarrow \psi$ -t sem nehéz belátni:
$\{original =<>\} l = original.empty()\{l = true\}$ specifikáció miatt az invariáns és a $original.empty()$ csak úgy teljesülhet egyszerre, ha $i = 0$. Viszont pontosan ezt fejezi ki a ψ.

Még azt kell belátni, hogy a ciklusmag megtartja az invariánst:

$\{original =< b_1, b_2, \ldots, b_i > \wedge megford =< b_n, b_{n-1}, \ldots, b_{i+1} > \wedge$
$\wedge \neg const(original) \wedge \neg const(megford)\}$
$megford.push\_back(original.top())$;
$original.pop()$;
$\{original =< b_1, b_2, \ldots, b_{i-1} > \wedge megford =< b_n, b_{n-1}, \ldots, b_i > \wedge$
$\wedge \neg const(original) \wedge \neg const(megford)\}$

Alkalmazzuk a szekvencia következtetési szabályát!

Beépített adattípusokon, mint például a `char` mind a másoló konstruktor, mind az `operator=` helyesen működik. Ezzel egyszerűsíthető a specifikáció ebben az esetben.
$\{original =< b_1, b_2, \ldots, b_i > \wedge \neg const(original)\} original.top(); \{b_i \wedge$
$\wedge \{original =< b_1, b_2, \ldots, b_i > \wedge \neg const(original)\}$ miatt:

$\{original =< b_1, b_2, \ldots, b_i > \wedge megford =< b_n, b_{n-1}, \ldots, b_{i+1} > \wedge$
$\wedge \neg const(original) \wedge \neg const(megford)\}$
$megford.push\_back(original.top());$
$\{original =< b_1, b_2, \ldots, b_i > \wedge megford =< b_n, b_{n-1}, \ldots, b_{i+1}, b_i > \wedge$
$\wedge \neg const(original) \wedge \neg const(megford)\}$
és
$\{original =< b_1, b_2, \ldots, b_i > \wedge megford =< b_n, b_{n-1}, \ldots, b_{i+1}, b_i > \wedge$
$\wedge \neg const(original) \wedge \neg const(megford)\}$
$original.pop();$
$\{original =< b_1, b_2, \ldots, b_{i-1} > \wedge megford =< b_n, b_{n-1}, \ldots, b_i > \wedge$
$\wedge \neg const(original) \wedge \neg const(megford)\}$

Ezzel a parciális helyesség bizonyítását befejeztük. Még bizonyítani kell a teljes helyességet is.

Terminálló függvény: $t = i$ és korlátja: $k = n + 1$. Nyilvánvaló, hogy: $\varphi \Rightarrow t = 0$ és az is triviális, hogy az invariáns teljesülése esetén $t < k$. Szintén nyilvánvaló a fenti bizonyításból és terminálófüggvény definíciójából, hogy a ciklusmag 1-gyel növeli a terminálófüggvény értékét.
Ezzel a bizonyítás teljesen készen van.

IV.1.2. Második megoldás

Mivel a stack egy átalakító, ezért nincsenek bejárói. Azt gondolhatjuk, hogy egy olyan megoldás elegánsabb, ha egy hátrafele haladó iterátorral járunk be egy konténert szintén egy ciklus segítségével.

```
#include <vector>
#include <list>
#include <iostream>
using namespace std;

int main()
{
  char betu;
  cin.get(betu);
```

```
  vector<char> v;
  list<char> l;

  while (betu!='\n') {
    l.push_back(betu);
    cin.get(betu);
  }
  list<char>::reverse_iterator it=l.rbegin();
  while(it!=l.rend()) {
     betu=*it;
     v.push_back(betu);
     ++it;
  }
  return 0;
}
```

$\varphi = \{l =< x_1, x_2, \ldots, x_n > \wedge \neg const(l) \wedge v =<> \wedge \neg const(v) \wedge betu = x_n\}$ a program előfeltétele
$\psi = l =< x_1, x_2, \ldots, x_n > \wedge \neg const(l) \wedge v =< x_n, x_{n-1}, \ldots, x_1 >\}$ pedig a program utófeltétele.

Bizonyítás:

Az előző bizonyításhoz nagyon hasonló gondolatmenetet lehet alkalmazni, csak a különbségekre hívjuk fel a figyelmet.

Először látható, hogy $type(it) = list < char > \Rightarrow cat(it) = Bi \Rightarrow cat(it) = For \Rightarrow cat(it) = In$.

A ciklus előtti állapot: $\{rev\_it(it) \wedge cat(it) = In \wedge v =<> \wedge \neg const(v) \wedge l =< x_1, x_2, \ldots, x_n > \wedge \neg const(l) \wedge *it = x_n \wedge \neg const\_it(it)\}$.

Különbség az előző megoldáshoz képest, hogy itt az eredeti objektumból (l-ből) nem töröljük ki az elemet. Természetesen, itt a ciklusban az it mutat x_i-re és nem egy stack teteje, tehát $*it = x_i \wedge rev\_it(it)$ kell az invariánsba. így ennyiben módosul a ciklus invariánsa.

Nyilvánvalóan, a ciklus is megtartja ezt az invariánst, hiszen a ciklusmag lefutása után $it = x_{i-1} \wedge \neg rev\_it(it) \wedge v =< x_1, x_2, \ldots, x_i > \wedge \neg const(v)\}$ lesz, mivel v-hez konkatenáltuk x_i-t a *betu* változón keresztül, és a inkrementáltuk it-t, amely hátrafelé halad, azaz x_i helyett x_{i-1}-re mutat.

Az előző bizonyítás terminálló függvénye, és a korlátja megfelel most is.

A ciklus utáni állapot pedig: $\{l =< x_1, x_2, \ldots, x_n \wedge v =< x_n, x_{n-1}, \ldots, x_1 > \wedge \neg const(l) \wedge \neg const(v) \wedge *it = x_1 \wedge rev\_it(it) \wedge \neg const\_it(it) \wedge betu = x_1 \wedge type(it) = list < char >\}$. Ebből pedig már következik ψ. Ezzel ennek a megoldásnak a helyességét is beláttuk.

IV.1.3. Harmadik megoldás

Mivel az előző két megoldásban ciklusokat kellett írni, amit könnyebben el lehet rontani, mint egy függvényhívást, ezért a harmadik megoldásunkban még hatékonyabb és elegánsabb megoldást adunk, a reverse algoritmus használatával. Ennek a megoldásnak előnyei a kevesebb memóriahasználat, gyorsabb végrehajtás, és a programkód egyszerűsége. Bjarne Stroustrup[3] is többször javasolja, hogy ciklusok helyett az algoritmusokat preferáljuk.

```
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
  char betu;
  cin.get(betu);
  vector<char> v;

  while (betu!='\n') {
    v.push_back(betu);
    cin.get(betu);
  }

  reverse(v.begin(), v.end());

  return 0;
}
```

A program elő-, és utófeltétele:
$\varphi = \{v =< b_1, b_2, \ldots, b_n > \wedge \neg const(v)\}$
$\psi = \{v =< b_n, b_{n-1}, \ldots, b_1 > \wedge \neg const(v)\}$

Ennek a programnak a helyessége viszont nyilvánvalóan igaz a reverse függvény és az iterátorok specifikációja miatt.

IV.2. Második példa

A következő feladat, amit megpróbálunk megoldani a következő: A program parancssori argumentumként megkap $n = argc$ darabszámú egész számot, és

határozzuk meg közülük a legnagyobb prímet. Ha az input adatok között nincsen prím, akkor az eredmény legyen 0.

Először elkészítjük az isprime függvényt, ami egy számról eldönti, hogy az prím-e. Ennek a helyességét bebizonyítjuk, mivel a megoldás ezt fel fogja használni.

```
bool isprime(int x) {
  int j=1;
  bool l=(x != 1);
  while (l && j < x-1) {
     l=((x%(j+1))!=0);
     ++j;
  }
  return l;
}
```

Az alprogram előfeltétele: $\varphi = \{true\}$ és az utófeltétele:
$\psi = \{l = prime(x)\}$, ahol $prime(x) = x \neq 1 \wedge \forall k \in [2..x-1] : x \; mod \; k \neq 0$.
Az $x \; mod \; k$ kifejezést C++-ban a `x % k` kifejezés írja le.

Bizonyítsuk be a függvény helyességét! Alkalmazzuk kétszer a szekvencia szabályát:

$\{true\} int \; j = 1; \{j = 1\}$ és
$\{j = 1\} bool \; l = (x \; != \; 1); \{j = 1 \wedge l = x \neq 1\} =: \varphi_1$

Ezután a ciklus szabályát kell alkalmazni:

$\mathcal{I} = \{j \in [1..x] \wedge (l = (x \neq 1) \wedge (\forall k \in [2..j] : x \; mod \; k \neq 0))\}$ lesz az invariáns.

Triviális, hogy $\varphi_1 \Rightarrow \mathcal{I}$

Be kell látni, hogy $\mathcal{I} \wedge (\neg l \vee j \geq x - 1) \Rightarrow \psi$:
Felbontva a zárójelet: $((\mathcal{I} \wedge \neg l) \vee (\mathcal{I} \wedge j \geq x - 1)) \Rightarrow \psi$.

Belátjuk, hogy $(\mathcal{I} \wedge \neg l) \Rightarrow \psi$ és
hogy $(\mathcal{I} \wedge j \geq x - 1) \Rightarrow \psi$.

Az első eset szerint: $j \in [1..x] \wedge (x = 1) \vee \exists k \in [2..j] : x \; mod \; k = 0$. Ez pontosan azt fejezi ki, hogy $l = prime(x) = hamis$. Tehát ebből következik ψ.

A második eset: $j \geq x - 1 \wedge j \in [1..x] \wedge \forall k \in [2..j] : x \; mod \; k \neq 0$. Vagyis $j = x - 1 \wedge \forall k \in [2..x-1] : x \; mod \; k \neq 0$. Ebből nyilvánvalóan következik ψ és az is, hogy $l = prime(x) = igaz$.

Be kell látni, hogy a ciklusmag lefutása után is igaz az invariáns:

$\{j \in [1..x] \wedge (l = (x \neq 1) \wedge (\forall k \in [2..j] : x \; mod \; k \neq 0))\}$
$l = ((x\%(j+1))! = 0);$
$\{j \in [1..x] \wedge (l = (x \neq 1) \wedge (\forall k \in [2..j+1] : x \; mod \; k \neq 0))\}$

$\{j \in [1..x] \wedge (l = (x \neq 1) \wedge (\forall k \in [2..j+1] : x \; mod \; k \neq 0))\}$
$++j;$
$\{j+1 \in [1..x] \wedge (l = (x \neq 1) \wedge (\forall k \in [2..j+1] : x \; mod \; k \neq 0))\} = \mathcal{I}$

Tehát a ciklusmag megtartja az invariánst. Ezzel a parciális helyességet bizonyítottuk.

A teljes helyességhez: terminálló függvény: $t = j - 1$ és a korlátja: $k = x + 1$.

Ezekkel $\varphi_1 \Rightarrow t = 0$ továbbá az is teljesül, hogy az $\mathcal{I} \Rightarrow t < k$. Teljesen triviális, hogy a terminálófüggvény értéke 1-gyel nő a ciklusmag lefutása esetén.

Ezzel az isprime függvény teljes helyességét beláttuk.

IV.2.1. Megoldás

```
#include <vector>
#include <algorithm>
using namespace std;

bool isprime(int x) {
  int j=1;
  bool l=(x != 1);
  while (l && j < x-1) {
     l=((x%(j+1))!=0);
     ++j;
  }
  return l;
}

bool jobb(int a,int b) {
  return(a<b && isprime(b));
}

int main(int argc, char *argv[]) {
  vector<int> v;
  for(int i=1;i<argc;i++) v.push_back(atoi(argv[i]));
```

```
  vector<int>::iterator it=max_element(v.begin(),v.end(),jobb);

  int maxpr;
  if (isprime(*it)) maxpr=*it;
  else maxpr=0;

  return 0;
}
```

Tehát az ötlet az az, hogy egy olyan reláció szerint keresünk maximumot, amely azt jelenti, hogy ahhoz, hogy az egyik szám jobb legyen, mint a másik, nem elég az, hogy nagyobb, azonkívül még prím is legyen. A végén az elágazásra azért van szükség, mert, ha nincs prím az adatokban, akkor is valamelyik elemet a ,,legjobbnak" fogja venni.

Valójában ezt a megoldást, könnyen lehet tetszőleges δ tulajdonságra általánosítani, ezáltal megkapva a feltételes maximumkeresés programozási tétel STL-es, ciklus nélküli, helyes megoldását.

Persze a problémát más úton is meg lehetet volna közelíteni: például a vektor csökkenő sorrendben való rendezése után az első isprime tulajdonságú elemet keresésével (find_if algoritmussal) vagy az adatok feldolgozásakor egyből csak az isprime tulajdonságú elemeket egy set adatszerkezetbe tárolni, ahol a rendezettség miatt, könnyű megkeresni a legnagyobb elemet, akár $O(1)$ időben is.

Nézzük a helyesség bizonyítását:

A program előfeltétele (az adatok feldolgozásától ismét eltekintünk): $\varphi = \{v =< v_1, v_2, \ldots, v_n > \wedge n \geq 1\}$.

A program utófeltétele: $\psi = \{(\forall j : 1 \leq j \leq n : \neg isprime(v_j) \Rightarrow maxpr = 0) \wedge \exists j : 1 \leq j \leq n : isprime(v_j) \Rightarrow maxpr = v_k \wedge isprime(maxpr) \wedge \forall j : (v_j \leq maxpr \vee \neg isprime(v_j))\}$.

Ahhoz, hogy belássuk a program megfelel a specifikációnak először is vegyünk észre a következőket:

- $LTC(int, jobb)$. Nyilvánvaló ugyanis, hogy két egész számot össze lehet hasonlítani a $jobb$ függvénnyel.
- $type(it) = vector < int > \Rightarrow cat(it) = Ran \Rightarrow cat(it) = Bi \Rightarrow cat(it) = For$.
- $type(v.begin()) = vector < int > \Rightarrow cat(it) = For$ és $type(v.end()) = vector < int > \Rightarrow cat(it) = For$ a fenti következtetések alapján.

Ekkor a `vector<int> it=max_element(v.begin(),v.end(),jobb);` függvényhívás után, a max_element definíciója alapján, a $\{v =< v_1, v_2, \ldots, v_n > \wedge \neg const(v) \wedge *it = v_k \wedge k \in [1,n] \wedge \forall l \in [1,n] : jobb(v_l, v_k)\}$ állapotba jutunk.

A $jobb(v_l, v_k) \Leftrightarrow v_l < v_k \wedge isprime(v_k)$.
Ezt visszahelyettesítve az állapotba:
$\{v =< v_1, v_2, \ldots, v_n > \wedge \neg const(v) \wedge *it = v_k \wedge k \in [1,n] \wedge \forall l \in [1,n] : v_l < v_k \wedge isprime(v_k)\}$ kapjuk.
Változó deklarálás után: $\{v =< v_1, v_2, \ldots, v_n > \wedge \neg const(v) \wedge *it = v_k \wedge k \in [1,n] \wedge \forall l \in [1,n] : v_l < v_k \wedge isprime(v_k) \wedge maxpr =?\} =: P$.

Ezután az elágazás következik: $\{P \wedge isprime(*it)\} maxpr = *it; \{maxpr = v_k \wedge isprime(maxpr) \wedge v =< v_1, v_2, \ldots, v_n > \wedge \neg const(v) \wedge *it = v_k \wedge k \in [1,n] \wedge \forall l \in [1,n] v_l < v_k\}$ Ebből már következik ψ-ben a konjunkció második tagja.
$P \wedge \neg isprime(*it)$ csak akkor lehet igaz, ha $\forall i \in [1,n] : \neg isprime(v_i)$. A `maxpr=0;` értékadás miatt pedig innen következik ψ-ben a konjunkció első tagja.
Ezzel pedig beláttuk, hogy a program megoldja a specifikált feladatot.

V. fejezet

Továbbfejlesztési tervek

V.1. A programszintézis lehetősége

A *programszintézis* alatt azt értjük, hogy egy adott specifikációból kiindulva, egy algoritmus segítségével megkapunk egy lehetséges helyes programot, amely megoldja a specifikált feladatot.

Több módszertannal ellentétben ebben az esetben itt nem egy absztrakt megoldást kapnánk, hanem egyből a C++ kódot. Ez nagyon sok esetben hasznos lehetne, ezért érdemes elgondolkodni a lehetőségen.

Az előző fejezetben láttuk, hogy egy pontosan specifikált feladatnak nagyon sok, lényegesen eltérő helyes megoldása lehet. A szómegfordító programra adtunk három teljesen más eszközöket használó, de helyes programot. Az első megoldás egy vermet használt, a második egy listát és hátrafele haladó iterátorokat, mégis végül a legjobb megoldás az lett, hogy egy sorozatmódosító algoritmust hívtunk segítségül.

Ez azért van, mert a megszokott módszertanokon belül a három programkonstrukció (szekvencia, elágazás és ciklus) és kevés elemi program (utasítás) van. Ezzel szemben a generikus programozás sokkal bővebb utasításhalmazzal rendelkezik, csak a programkonstrukciók maradtak ugyanazok. Ez gyakorlati szempontból nagyon hasznos, de elméletben nehezebben kezelhető.

Gondoljunk bele, hogy egy saját osztályt szeretnénk megvalósítani az STL segítségével, például egy telefonkönyvet! A telefonkönyv egy bejegyzése tulajdonképpen egy név és egy telefonszámból áll. A név egyértelműen egy string, a telefonszámot szintén vehetjük egy string-nek. A telefonkönyvhöz szeretnénk a következő tagfüggvényeket használni: új bejegyzés, keresés név alapján, törlés, bejegyzés módosítása. Gondoljuk végig, hogy az STL különböző konténereivel, milyen megoldások adhatók!

Első megoldásnak az tűnik, hogy elkészítjük a két string adattagot tartal-

mazó Bejegyzés osztályt, és valamelyik szekvenciális tárolót példányosítjuk ezzel a Bejegyzés osztállyal. A szekvenciális tárolókban lineáris idejű keresést lehet végrehajtani, de konstans időben lehet beszúrni. Ha viszonylag kevés adatra számítunk, jó döntés lehet. Ezekben az esetekben nem gond az sem, ha egy névhez több szám tartozik. De melyiket célszerű példányosítani?

A list akkor célszerű, ha lista műveleteket szeretnénk végezni, például két telefonkönyvet összefűzni. Esetleg szóba jöhetne egy rendezett listát használó megoldás is, a hatékonyság növelése érdekében.

A vektor előnye, hogy *O(1)* idejű elem elérése, tehát, ha tudjuk, hogy a konténer 4. elemének telefonszám adattagjára vagyunk kíváncsiak. Ilyen lehetőség megtalálható például mobiltelefonok telefonkönyv szolgáltatásában.

A deque előnye, hogy mindkét végén *O(1)* műveletigénye van az elérésnek és az insert műveletnek is. Ez a szempont elég elhanyagolható egy telefonkönyv kapcsán.

Nézzük meg mit kezdhetünk az asszociatív tárolókkal! Tulajdonképpen a multiset és a multimap konténerekkel érdemes foglalkozni. A multiset-et szintén a Bejegyzés osztállyal lehet példányosítani, csak még biztosítani kell a rendezést. Ezt nem nehéz megtenni: meg kell valósítani az `operator<`-t, ami visszaadja név adattag szerinti `operator<` eredményét, ami azt jelenti, hogy név szerint lexikografikusan lesznek rendezve az adataink. A multimap-et pedig úgy lehetne példányosítani, hogy az indextípus is és a kulcs típusa is string. Ez a két megoldás elég ekvivalensnek tekinthető. Ezeknek az az előnye, hogy a keresés már $O(\log(n))$-es rendezett tárolás miatt. Viszont emiatt beszúrás költsége is megnő $O(\log(n))$-re. A valódi, nyomtatott telefonkönyvek ilyen szisztémát mutatnak be. Ha sok keresést kell majd elvégezni a telefonkönyvön, akkor ajánlott ez a megoldás. Viszont sok adat bejegyezése és kevés keresés esetén ez a megoldás nem tekinthető optimálisnak.

Viszont vegyük észre, hogy a specifikációból semmire sem következtethetünk a telefonkönyv keresés-új bejegyzés arányával kapcsolatban! Ahhoz, hogy eldönthessük, hogy melyik reprezentáció a leghasznosabb nincs igazán jó algoritmus! Lehetne szűkíteni a szóbajöhető konténereket és algoritmusokat, ezzel a megoldással az a gond, hogy könnyen elveszíthetjük azt a flexibilitást, amit az STL nyújt. Viszont [30] figyelmeztet arra, hogy a szintetizáló algoritmusok általában nagyon rossz hatékonyságúak, a megoldás előállításához a specifikáció hosszával exponenciálisan arányos időre van szükség. Ahhoz, hogy ez az algoritmus megszülethessen a mesterséges intelligencia eszközeit kell majd igénybe venni, például heurisztikák[31], amelyek pontosabb lehetőséget adnak, hogy jobb reprezentációt és hatékonyabb műveleteket kapjunk vagy genetikus algoritmusokat[32] lehetne alkalmazni, amivel ugyanehhez jutnánk közelebb.

Továbbá, érdekes lehetne egy olyan programmanipuláló algoritmus meg-

adása, amellyel egy helyes programból, más adatszerkezetet illetve algoritmust használó programot lehetne létrehozni, ami ugyanúgy megoldaná a specifikált feladatot, de valamilyen szempont szerint hatékonyabban vagy optimálisabban. Például lehetne optimalizálni (aszimptotikus) futásidőre, vagy memóriahasználatra.

V.2. Más nyelvek

Láttuk, hogy az utóbbi időben több nyelv is kiegészült a generikus programozáshoz szükséges nyelvi eszközökkel, ilyen például a C# és a Java. Java Generics-hez (a Java sablonokat kezelő bővített változata) már az STL ottani megfelelője is elkészült, ennek neve *Collection Framework*.

Hasznos lenne, ezekre a nyelvekre is, és az itt használatos könyvtárakra kiterjeszteni a módszert. Nyilvánvalóan ez megoldható feladat, hiszen csak a C++ nyelvi konstrukciót át kell formalizálni a C# vagy Java szemantikájának megfelelően és hasonló eszközökkel a könyvtárakat is specifikálni kell. Mivel a Hoare-módszer ilyen imperatív nyelvekhez egyszerűen módosítható, ezekkel nincsen probléma. Így megkaphatjuk az adott nyelv esetén a generikus módszertan alapjait.

V.3. Egyéb tervek

Nyilvánvalóan célszerű lenne a teljes STL formális specifikációját megadni, az esetleges hibákat kijavítani, és a további concept-eket is leírni. Továbbá hasznos lenne a rendszert minél inkább *függetleníteni* [18].

Ilyen kalkulusok esetén szokás vizsgálni a helyesség és teljesség kérdését[18]. A mi esetünkben ezek megválaszolása nem nyilvánvaló. Mivel a Hoare-módszer helyes és teljes kalkulus, ezért ennek a módszernek a helyessége is igaz, feltéve, hogy a specifikáció nem tartalmaz hibákat. Ezt a tényt ezzel a módszerrel kellene belátni, tehát ez egy nagyon hosszú bizonyítás lenne, szinte elvégezhetetlen. Még bonyolultabb a válasz a teljesség problémájára. Ennek a kérdésnek egyébként is csak akkor lesz létjogosultsága, amikor az egész STL-t specifikáltuk, addig ugyanis könnyű ellenpéldát adni. Valószínűleg nem teljesül az, hogy minden helyes programhoz megkonstruálható, annak a bizonyítása, hogy az valóban helyes, az STL mérete miatt. Eddig sem a teljesség bizonyítását, sem pedig ellenpéldát nem sikerült megadni, csak sejtjük, hogy a teljesség nem igaz a módszerre. Fontos lenne, ennek a kérdésnek a pontos eldöntése, és precíz megválaszolása.

VI. fejezet

Összefoglalás

Bemutattuk a C++ szabványkönyvtár egy nagyon fontos részét, az STL-t. Bemutattunk már létező eszközöket, amivel már leírták formális eszközökkel az STL-t, és elemeztünk olyan szoftvereket, amivel az STL használata precízebbé és könnyebbé válik. Emellett sikerült olyan problémákat találnunk, ahol a különböző rendszerek és a nyelvi szabvány nem egyezik. A dolgozatban sikerült megadni egy formalizmust, amivel az STL leírható. Ezzel megtettük az első lépéseket egy generikus programozási módszertan irányába, amely matematikai alapokon nyugszik. A módszerünk tulajdonképpen a Hoare-módszert veszi alapul, és kiterjeszti azt, oly módon, hogy a szükséges C++ nyelvi elemek és maga az STL formalizálható legyen. Az STL nagy részét megadtuk a formalizmussal és helyességbizonyításokra példákat is hoztunk.

A módszer előnyei:

- Ismert módszer kiterjesztése, nincs szükség túl sok új eszköz megismerésére.
- Viszonylag nem túl mély matematikai eszközök használata.
- A specifikáció elemezhetősége
- Precíz helyességbizonyítás, mind az STL implementációk irányába, mind konkrét felhasználás irányába
- A helyességbizonyítás statikus, független a program futásától és környezetétől.
- Nincsenek absztakt programok, C++ kódokkal dolgozik a modell.
- A módszer könnyen terjeszthető ki párhuzamos programok esetére is.

A módszer hátrányai:

- A teljes STL formális leírása hosszú, ezért egy helyességbizonyítás folyamata több időbe telhet, mire a programozó kikeresi az összes használt függvény leírását, mint hagyományos módszertanok esetén. A módszer azonban támogatható szoftver eszközökkel, melyek ezt a hátrányt csökkenthetik vagy kiküszöbölhetik.
- A nagy számú specifikáció hibákat rejthet.
- A szintézisre egyenlőre nincs módszer.
- A kalkulus helyessége nem bizonyított, a teljessége nem eldöntött.

A. Függelék

Specifikációk

Ebben a részben megadjuk az STL nagy részének a formális specifikációját. Először néhány alapvető concept-et adunk meg, amelyet mind a konténerek, mind az algoritmusok használnak. Ezt követi az iterátorok formális leírása. Ez majdnem teljesen kész van, csak az író kategóriájú bejárókat hagytuk ki, illetve 1-2 művelet kimaradt. Ezután a konténerek következnek: a szekvenciális adatszerkezetekből a vector-t mutatjuk be, a többi ehhez nagy mértékben hasonlít. A három átalakítót leírjuk, mivel ezeknek a specifikációja rövidebb, és elég nagy különbségek vehetők észre közöttük. Ezt követően az asszociatív tárolók közül a set-et és a multiset-et adjuk meg formálisan, ez alapján, már könnyen elképzelhető a map és a multimap is. Végezetül néhány alapvető algoritmust is specifikálunk.

A.1. Alapvető concept-ek

A.1.1. Equality Comparable

Ez a concept azt fejezi ki, hogy a T osztályon értelmezett egy $f : T \times T \mapsto \mathbb{L}$ függvény, és ez egy ekvivalenciarelációt határoz meg. Alapértelmezésben f neve `operator==`, ilyenkor értelmes az `!=` kifejezés is.

$EC(T, f)$:

$\mathcal{I} = \{(\forall x \in T : f(x, x) = true) \wedge (\forall x, y \in T : f(x, y) \Rightarrow f(y, x)) \wedge (\forall x, y, z \in T : f(x, y) \wedge f(y, z) \Rightarrow f(x, z))\}$

$$\{x = x_1 \wedge y = y_1 \wedge x = y\} bool\ l = f(x, y); \{x = x_1 \wedge y = y_1 \wedge l = true\}$$

$$\{x = x_1 \wedge y = y_1 \wedge \neg x = y\} bool\ l = f(x, y); \{x = x_1 \wedge y = y_1 \wedge l = false\}$$

$$\{x = x_1 \wedge y = y_1 \wedge x = y\} bool\ l = !f(x, y); \{x = x_1 \wedge y = y_1 \wedge l = false\}$$

$$\{x = x_1 \wedge y = y_1 \wedge \neg x = y\} bool\ l = !f(x, y); \{x = x_1 \wedge y = y_1 \wedge l = true\}$$

A.1.2. LessThen Comparable

Ez a concept azt fejezi ki, hogy a T osztály elemei rendezhetőek. Itt az alapértelmezett név az `operator<`. Ennek legalább parciális rendezésnek kell lennie.

$LTC(T, f)$:

$\mathcal{I} = \{(\forall x \in T : f(x, x) = false) \wedge \forall x, y \in T : f(x, y) \Rightarrow \neg f(y, x) \wedge \forall x, y, z \in T : f(x, y) \wedge f(y, z) \Rightarrow f(x, y)\}$

$$\{x = x_1 \wedge y = y_1 \wedge x < y\} bool\ l = x < y; \{l = true \wedge x = x_1 \wedge y = y_1\}$$

$$\{x = x_1 \wedge y = y_1 \wedge x \geq y\} bool\ l = x < y; \{l = false \wedge x = x_1 \wedge y = y_1\}$$

A.2. Iterátorok specifikációja

A.2.1. Iterátorok hierarchiája

$$cat(it) = Ran \Rightarrow cat(it) = Bi$$
$$cat(it) = Bi \Rightarrow cat(it) = For$$
$$cat(it) = For \Rightarrow cat(it) = In$$
$$cat(it) = For \Rightarrow cat(it) = Out$$

A.2.2. Típusok és kategóriák

$$type(it) = list < T > \Rightarrow cat(it) = Bi$$
$$type(it) = deque < T > \Rightarrow cat(it) = Ran$$
$$type(it) = vector < T > \Rightarrow cat(it) = Ran$$

$$type(it) = set < T > \Rightarrow (cat(it) = Bi \wedge const\_it(it))$$
$$type(it) = mset < T > \Rightarrow (cat(it) = Bi \wedge const\_it(it))$$
$$type(it) = map < I, T > \Rightarrow (cat(it) = Bi \wedge const\_it(it))$$
$$type(it) = mmap < I, T > \Rightarrow (cat(it) = Bi \wedge const\_it(it))$$

A.2.3. Olvasó iterátorok specifikációja

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x_j\} T\ e = *it; \{e = e.cpctor(x_j) \wedge *it = x_j \wedge \\ \wedge cat(it) = In \wedge x =< x_1, x_2, \ldots, x_n >\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x_j\} e = *it; \{e = e.operator = (x_j) \wedge *it = x_j \wedge \\ \wedge cat(it) = In \wedge x =< x_1, x_2, \ldots, x_n >\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x.end\} T\ e = *it; \{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x.end\} e = *it; \{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x_j \wedge j < n \wedge \neg rev\_it(it)\} ++it; \\ \{*it = x_{j+1} \wedge x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge \neg rev\_it(it)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x_n \wedge \neg rev\_it(it)\} ++it; \{*it = x.end \wedge \\ \wedge x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge \neg rev\_it(it)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x.end \wedge \neg rev\_it(it)\} ++it; \{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x_j \wedge j < n \wedge \neg rev\_it(it)\} T\ e = *it++; \\ \{*it = x_{j+1} \wedge e = e.cpctor(x_j) \wedge x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge \neg rev\_it(it)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x_n \wedge \neg rev\_it(it)\} T\ e = *it++; \\ \{e = e.cpctor(x_n) \wedge *it = x.end \wedge x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x.end \wedge \neg rev\_it(it)\} T\ e = *it++; \{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x_j \wedge j < n \wedge \neg rev\_it(it)\} e = *it{+}{+};$$

$$\{e = e.operator = (x_j) \wedge *it = x_{j+1} \wedge x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge \neg rev\_it(it)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x_n\} e = *it + +; \{*it = x.end \wedge$$
$$\wedge e = e.operator = (x_n) \wedge x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x.end\} e = *it + +; \{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x_j \wedge 1 < j \wedge rev\_it(it)\} + +it;$$
$$\{*it = x_{j-1} \wedge x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge rev\_it(it)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x_1 \wedge rev\_it(it)\} + +it; \{*it = x.rend \wedge$$
$$\wedge x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge rev\_it(it)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x.rend \wedge \neg rev\_it(it)\} + +it; \{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x_j \wedge 1 < j \wedge rev\_it(it)\} T\ e = *it{+}{+};$$
$$\{*it = x_{j-1} \wedge e = e.cpctor(x_j) \wedge x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge rev\_it(it)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x_1 \wedge rev\_it(it)\} T\ e = *it + +;$$
$$\{e = e.cpctor(x_1) \wedge *it = x.rend \wedge x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge rev\_it(it)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x.rend \wedge rev\_it(it)\} T\ e = *it{+}{+}; \{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x_j \wedge 1 < j \wedge rev\_it(it)\}e = *it{+}{+};$$

$$\{*it = x_{j-1} \wedge e = e.operator = (x_j) \wedge x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge rev\_it(it)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x_1 \wedge rev\_it(it)\}e = *it + +;$$

$$\{e = e.operator = (x_1) \wedge *it = x.rend \wedge x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge rev\_it(it)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = In \wedge *it = x.rend \wedge rev\_it(it)\}e = *it{+}{+}; \{Undef\}$$

$$\{cat(it) = In \wedge *it =?\} + +it; \{Undef\}$$
$$\{cat(it) = In \wedge *it =?\}e = *it + +; \{Undef\}$$
$$\{cat(it) = In \wedge *it =?\}T\ e = *it + +; \{Undef\}$$

Ezen az absztrakt szinten nincs értelme külön definiálni postinkrementálást (it++), lényeges eltérés nincs a formalizmusban. Valójában a kettő nem ekvivalens! Austern[21] a könyvében rámutat, hogy valójában az `it++;` utasítás a `(void)++it;` utasítással ekvivalens.

A.2.4. Előre haladó iterátorok specifikációja

Az előre haladó iterátorok nem tartalmaznak új műveleteket, ami miatt mégis különböznek az input iterátoroktól az az, hogy a forward iterátorok engedélyezik azt, hogy inkrementáláskor a régi értéket másolja, és biztosítja azt a tulajdonságot is, hogy ha i és j iterátorok dereferálhatók és $i == j$, akkor $++i == ++j$. Emiatt az előre haladó iterátorokkal többször is bejárható egy objektum, míg az előzőekkel ez nem volt megtehető [21].

A.2.5. Kétirányú iterátorok specifikációja

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = Bi \wedge \neg rev\_it(it) \wedge *it = x_j \wedge j > 1\} - -it;$$
$$\{*it = x_{j-1} \wedge cat(it) = Bi \wedge x =< x_1, x_2, \ldots, x_n > \wedge \neg rev\_it(it)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = Bi \wedge \neg rev\_it(it) \wedge *it = x_1\} - -it; \{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = Bi \wedge \neg rev\_it(it) \wedge *it = x.end\} - -it;$$
$$\{*it = x_n \wedge x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = Bi \wedge \neg rev\_it(it)\}$$

$$\{cat(it) = Bi \wedge *it =?\} - -it; \{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = Bi \wedge rev\_it(it) \wedge *it = x_j \wedge j < n\} - -it;$$
$$\{*it = x_{j+1} \wedge cat(it) = Bi \wedge x =< x_1, x_2, \ldots, x_n > \wedge rev\_it(it)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = Bi \wedge rev\_it(it) \wedge *it = x.rend\} - -it;$$
$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it) = Bi \wedge rev\_it(it) \wedge *it = x_1\}$$

A.2.6. Közvetlen elérésű iterátorok specifikációja

$$\{cat(it) = Ran \wedge x =< x_1, x_2, \ldots, x_n > \wedge *it = x_i \wedge \neg rev\_it(it) \wedge 0 \leq k \leq n-i \wedge$$
$$\wedge const\_it(it) = L\} it+ = k; \{cat(it) = Ran \wedge *it = x_{i+k} \wedge \neg rev\_it(it) \wedge$$
$$\wedge x =< x_1, x_2, \ldots, x_n > \wedge const\_it(it) = L\}$$

$$\{cat(it) = Ran \wedge const\_it(it) = L \wedge *it = x_i \wedge \neg rev\_it(it) \wedge -(i-1) \leq k < 0 \wedge$$
$$\wedge x =< x_1, x_2, \ldots, x_n >\} it+ = k; \{cat(it) = Ran \wedge *it = x_{i-k} \wedge \neg rev\_it(it) \wedge$$
$$\wedge x =< x_1, x_2, \ldots, x_n > \wedge const\_it(it) = L\}$$

$$\{cat(it) = Ran \wedge x =< x_1, x_2, \ldots, x_n > \wedge *it = x_i \wedge rev\_it(it) \wedge 0 \leq k \leq i-1 \wedge$$
$$\wedge const\_it(it) = L\} it+ = k; \{cat(it) = Ran \wedge *it = x_{i-k} \wedge rev\_it(it) \wedge$$
$$\wedge x =< x_1, x_2, \ldots, x_n > \wedge const\_it(it) = L\}$$

$$\{cat(it) = Ran \wedge x =< x_1, x_2, \ldots, x_n > \wedge *it = x_i \wedge rev\_it(it) \wedge -(n-i) \leq k < 0 \wedge$$
$$\wedge x =< x_1, x_2, \ldots, x_n > \wedge const\_it(it) = L\} it+ = k; \{cat(it) = Ran \wedge *it = x_{i+k} \wedge$$

$$\wedge x =< x_1, x_2, \ldots, x_n > \wedge const\_it(it) = L \wedge rev\_it(it)\}$$

$$\{cat(it) = Ran \wedge x =< x_1, x_2, \ldots, x_n > \wedge *it = x_i \wedge \neg rev\_it(it) \wedge k = n+1-i \wedge$$
$$\wedge const\_it(it) = L\} it+ = k; \{cat(it) = Ran \wedge *it = x.end \wedge \neg rev\_it(it) \wedge$$
$$\wedge x =< x_1, x_2, \ldots, x_n > \wedge const\_it(it) = L\}$$

$$\{cat(it) = Ran \wedge x =< x_1, x_2, \ldots, x_n > \wedge *it = x_i \wedge rev\_it(it) \wedge k = i-n-1 \wedge$$
$$\wedge const\_it(it) = L\} it+ = k; \{cat(it) = Ran \wedge *it = x.end \wedge rev\_it(it) \wedge$$
$$\wedge x =< x_1, x_2, \ldots, x_n > \wedge const\_it(it) = L\}$$

$$\{cat(it) = Ran \wedge x =< x_1, x_2, \ldots, x_n > \wedge *it = x_i \wedge \neg rev\_it(it) \wedge k > n+1-i\}$$
$$it+ = k; \{Undef\}$$

$$\{cat(it) = Ran \wedge x =< x_1, x_2, \ldots, x_n > \wedge * it = x_i \wedge rev\_it(it) \wedge k \geq i\}$$
$$it+ = k; \{Undef\}$$

$$\{cat(it) = Ran \wedge x =< x_1, x_2, \ldots, x_n > \wedge *it = x_i \wedge rev\_it(it) \wedge k < i-n-1\}$$
$$it+ = k; \{Undef\}$$

$$\{cat(it) = Ran \wedge x =< x_1, x_2, \ldots, x_n > \wedge * it = x_i \wedge \neg rev\_it(it) \wedge k \leq (-i)$$
$$it+ = k; \{Undef\}$$

$$\{cat(it) = Ran \wedge *it =?\} it+ = k; \{Undef\}$$

A.2.7. Iterátorok deklarációja

$\{true\}vector < T >:: iterator\ it; \{type(it) = vector < T > \wedge *it =? \wedge \neg const\_it(it) \wedge \neg rev\_it(it)\}$

$\{true\}vector < T >:: const\_iterator\ cit; \{type(cit) = vector < T > \wedge *cit =? \wedge const\_it(cit) \wedge$
$\wedge \neg rev\_it(cit)\}$

$\{true\}vector < T >:: reverse\_iterator\ rit; \{rev\_it(rit) \wedge *rit =? \wedge \neg const\_it(rit) \wedge$
$\wedge type(rit) = vector < T >\}$

$\{true\}vector < T >:: const\_reverse\_iterator\ crit; \{type(crit) = vector < T > \wedge *crit =? \wedge$
$\wedge const\_it(crit) \wedge rev\_it(crit)\}$

$\{true\}list < T >:: iterator\ it; \{type(it) = list < T > \wedge *it =? \wedge \neg const\_it(it) \wedge \neg rev\_it(it)\}$

$\{true\}list < T >:: const\_iterator\ cit; \{type(cit) = list < T > \wedge *cit =? \wedge const\_it(cit) \wedge$
$\wedge \neg rev\_it(cit)\}$

$\{true\}list < T >:: reverse\_iterator\ rit; \{type(rit) = list < T > \wedge *rit =? \wedge \neg const\_it(rit) \wedge$
$\wedge rev\_it(rit)\}$

$\{true\}list < T >:: const\_reverse\_iterator\ crit; \{type(crit) = list < T > \wedge *crit =? \wedge$
$\wedge const\_it(crit) \wedge rev\_it(crit)\}$

$\{true\}deque < T >:: iterator\ it; \{type(it) = deque < T > \wedge *it =? \wedge \neg const\_it(it) \wedge \neg rev\_it(it)\}$

$\{true\}deque < T >:: const\_iterator\ cit; \{type(cit) = deque < T > \wedge *cit =? \wedge const\_it(cit) \wedge$
$\wedge \neg rev\_it(cit)\}$

$\{true\}deque < T >:: reverse\_iterator\ rit; \{type(rit) = deque < T > \wedge *rit =? \wedge \neg const\_it(rit) \wedge$
$\wedge rev\_it(rit)\}$

$\{true\}deque < T >:: const\_reverse\_iterator\ crit; \{type(crit) = deque < T > \wedge *crit =? \wedge$
$\wedge const\_it(crit) \wedge rev\_it(crit)\}$

$\{true\}set < T >:: iterator\ it; \{type(it) = set < T > \wedge *it =? \wedge const\_it(it) \wedge \neg rev\_it(it)\}$

$\{true\}set < T >:: const\_iterator\ cit; \{type(cit) = set < T > \wedge *cit =? \wedge const\_it(cit) \wedge$

$$\wedge\neg rev\_it(cit)\}$$

$$\{true\}set < T >:: reverse\_iterator\ rit; \{type(rit) = set < T > \wedge *rit =? \wedge const\_it(rit) \wedge \wedge rev\_it(rit)\}$$

$$\{true\}set < T >:: const\_reverse\_iterator\ crit; \{type(crit) = set < T > \wedge *crit =? \wedge \wedge const\_it(crit) \wedge rev\_it(crit)\}$$

$$\{true\}multiset < T >:: iterator\ it; \{type(it) = mset < T > \wedge *it =? \wedge const\_it(it) \wedge \wedge \neg rev\_it(it)\}$$

$$\{true\}multiset < T >:: const\_iterator\ cit; \{type(cit) = mset < T > \wedge *cit =? \wedge const\_it(cit) \wedge \wedge \neg rev\_it(cit)\}$$

$$\{true\}multiset < T >:: reverse\_iterator\ rit; \{type(rit) = mset < T > \wedge *rit =? \wedge \wedge const\_it(rit) \wedge \wedge rev\_it(rit)\}$$

$$\{true\}multiset < T >:: const\_reverse\_iterator\ crit; \{type(crit) = mset < T > \wedge *crit =? \wedge \wedge const\_it(crit) \wedge rev\_it(crit)\}$$

$$\{true\}map < Key, T >:: iterator\ it; \{type(it) = map < Key, T > \wedge const\_it(it) \wedge \wedge \neg rev\_it(it) \wedge (*it) =?\}$$

$$\{true\}map < Key, T >:: const\_iterator\ cit; \{type(cit) = map < Key, T > \wedge const\_it(cit) \wedge \wedge \neg rev\_it(cit) \wedge (*cit) =?\}$$

$$\{true\}map < Key, T >:: reverse\_iterator\ rit; \{type(rit) = map < Key, T > \wedge const\_it(rit) \wedge \wedge rev\_it(rit) \wedge (*rit) =?\}$$

$$\{true\}map < Key, T >:: const\_reverse\_iterator\ crit; \{type(crit) = map < Key, T > \wedge \wedge const\_it(crit) \wedge rev\_it(crit) \wedge *crit =?\}$$

$$\{true\}multimap < Key, T >:: iterator\ it; \{type(it) = mmap < Key, T > \wedge const\_it(it) \wedge \wedge \neg rev\_it(it) \wedge *it =?\}$$

$$\{true\}multimap < Key, T >:: const\_iterator\ cit; \{type(cit) = mmap < Key, T > \wedge \wedge const\_it(cit) \wedge \neg rev\_it(cit) \wedge *cit =?\}$$

$$\{true\}multimap < Key, T >:: reverse\_iterator\ rit; \{type(rit) = mmap < Key, T > \wedge \wedge const\_it(rit) \wedge rev\_it(rit) \wedge *rit =?\}$$

$$\{true\}multimap < Key, T >:: const\_reverse\_iterator\ crit; \{type(crit) = mmap < Key, T > \wedge \wedge const\_it(crit) \wedge rev\_it(crit) \wedge (*crit) =?\}$$

A.3. Konténerek specifikációja

A.3.1. A vector osztály

$\mathcal{I} = \{T \neq bool\}$[1]

$$\{true\}vector < T > v; \{v =<> \wedge \neg const(v)\}$$
$$\{true\}const\ vector < T > v; \{v =<> \wedge const(v)\}$$

$$\{true\}vector < T > v(n,t); \{\neg const(v) \wedge v =< v_0, v_1, \ldots, v_{n-1} > \wedge$$
$$\wedge \forall i \in [0..n-1] : v_i = t\}$$

$$\{true\}const\ vector < T > v(n,t); \{const(v) \wedge v =< v_0, v_1, \ldots, v_{n-1} > \wedge$$
$$\wedge \forall i \in [0..n-1] : v_i = t\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge \neg const(v)\}v.push\_back(x);$$
$$\{v =< v_0, v_1, \ldots, v_n, x > \wedge \neg const(v)\}$$

$$\{v =< v_0, v_1, \ldots, v_n >\}s = v.size(); \{s = n+1 \wedge v =< v_0, v_1, \ldots, v_n >\}$$
$$\{v =<>\}s = v.size(); \{s = 0 \wedge v =<>\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge i \leq n \wedge \neg const(v)\}T\ t = v[i];$$
$$\{v =< v_0, v_1, \ldots, v_n > \wedge t = t.cpctor(v_i) \wedge \neg const(v)\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge i \leq n \wedge const(v)\}T\ t = v[i];$$
$$\{v =< v_0, v_1, \ldots, v_n > \wedge t = t.cpctor(v_i) \wedge const(v)\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge i \leq n \wedge \neg const(v)\}t = v[i];$$

[1] A *vector* < *bool* > nem ez az osztály. Hatékonysági okokból a bitvektor külön osztályként van megvalósítva, de annak specifikációja ugyanaz, mint ez $T = bool$ helyettesítéssel.

$$\{v =< v_0, v_1, \ldots, v_n > \wedge t = t.operator = (v_i) \wedge \neg const(v)\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge i \leq n \wedge const(v)\} t = v[i];$$
$$\{v =< v_0, v_1, \ldots, v_n > \wedge t = t.operator = (v_i) \wedge const(v)\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge i \leq n \wedge \neg const(v)\} v[i] = c;$$
$$\{v = v_0, v_1, \ldots, v_{i-1}, c, v_{i+1}, \ldots, v_n > \wedge \neg const(v)\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge i > n \wedge \neg const(v)\} v[i] = c; \{Undef\}$$

$$\{v =<> \wedge \neg const(v)\} v.pop\_back(); \{Undef\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge \neg const(v)\} v.pop\_back();$$
$$\{v =< v_0, v_1, \ldots, v_{n-1} \wedge \neg const(v)\}$$

$$\{v =<> \wedge \neg const(v)\} v.front() = c; \{Undef\}$$
$$\{v =<> \wedge \neg const(v)\} v.back() = c; \{Undef\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge \neg const(v)\} v.front() = c; \{\neg const(v) \wedge$$
$$\wedge v =< c, v_1, v_2, \ldots, v_n >\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge \neg const(v)\} v.back() = c; \{\neg const(v) \wedge$$
$$\wedge v =< v_0, v_1, v_2, \ldots, c >\}$$

$$\{v =< v_0, v_1, \ldots, v_n \wedge \neg const(v)\} t = v.front(); \{\neg const(v) \wedge$$
$$\wedge v =< v_0, v_1, \ldots, v_n \wedge t = t.operator = (v_0)\}$$

$$\{v =< v_0, v_1, \ldots, v_n \wedge \neg const(v)\} T\ t = v.front(); \{\neg const(v) \wedge$$

$$\wedge v =< v_0, v_1, \ldots, v_n \wedge t = t.cpctor = (v_0)\}$$

$$\{v =< v_0, v_1, \ldots, v_n \wedge const(v)\}t = v.front(); \{const(v) \wedge$$
$$\wedge v =< v_0, v_1, \ldots, v_n \wedge t = t.operator = (v_0)\}$$

$$\{v =< v_0, v_1, \ldots, v_n \wedge const(v)\}T\ t = v.front(); \{const(v) \wedge$$
$$\wedge v =< v_0, v_1, \ldots, v_n \wedge t = t.cpctor = (v_0)\}$$

$$\{v =< v_0, v_1, \ldots, v_n \wedge \neg const(v)\}t = v.back(); \{\neg const(v) \wedge$$
$$\wedge v =< v_0, v_1, \ldots, v_n \wedge t = t.operator = (v_n)\}$$

$$\{v =< v_0, v_1, \ldots, v_n \wedge \neg const(v)\}T\ t = v.back(); \{\neg const(v) \wedge$$
$$\wedge v =< v_0, v_1, \ldots, v_n \wedge t = t.cpctor = (v_n)\}$$

$$\{v =< v_0, v_1, \ldots, v_n \wedge const(v)\}t = v.back(); \{const(v) \wedge$$
$$\wedge v =< v_0, v_1, \ldots, v_n \wedge t = t.operator = (v_n)\}$$

$$\{v =< v_0, v_1, \ldots, v_n \wedge const(v)\}T\ t = v.back(); \{const(v) \wedge$$
$$\wedge v =< v_0, v_1, \ldots, v_n \wedge t = t.cpctor = (v_n)\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge *it = v_i \wedge type(it) = vector < T > \wedge \neg const(v)\}v.erase(it);$$
$$\{v =< v_0, v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n > \wedge * it = v_{i+1} \wedge \neg const(v)\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge(*it = v.end \vee *it =?) \wedge \neg rev\_it(it) \wedge \neg const(v)\}$$
$$v.erase(it); \{Undef\}$$

$$\{v =<>\}l = v.empty(); \{l = true \wedge v =<>\}$$
$$\{v =< v_0, v_1, \ldots, v_n >\}l = v.empty(); \{l = false \wedge v =< v_0, v_1, \ldots, v_n >\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge \neg const(v) type(it) = vector < T > \wedge *it = v_i\} v.insert(it, a);$$

$$\{\neg const(v) \wedge v =< v_0, v_1, \ldots, v_{i-1}, a, v_i, \ldots, v_n > \wedge *it = a \wedge type(it) = vector < T >\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge type(it) = vector < T > \wedge *it =?\} v.insert(it, k, t); \{Undef\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge \neg const(v) type(it) = vector < T > \wedge *it =?\} v.insert(it, a); \{Undef\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge \neg const(v) \wedge type(it_1) = vector < T > \wedge type(it_2) = vector < T > \wedge$$
$$\wedge * it_1 = v_i \wedge *it_2 = v_j \wedge i < j \wedge \neg rev\_it(it_1) \wedge \neg rev\_it(it_2)\}$$
$$v.erase(it_1, it_2);$$
$$\{v =< v_0, v_1, \ldots, v_{i-1}, v_j, v_{j+1}, \ldots, v_n > \wedge \neg const(v) \wedge \neg rev\_it(it_1) \wedge$$
$$\wedge * it_1 = v_j \wedge *it_2 = v_j \wedge \neg rev\_it(it_1) \wedge \neg rev\_it(it_2)\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge \neg const(v) \wedge type(it_1) = vector < T > \wedge type(it_2) = vector < T > \wedge$$
$$\wedge * it_1 = v_i \wedge *it_2 = v_j \wedge i < j \wedge rev\_it(it_1) \wedge rev\_it(it_2)\}$$
$$v.erase(it_2, it_1);$$
$$\{v =< v_0, v_1, \ldots, v_{i-1}, v_i, v_{j+1}, \ldots, v_n > \wedge \neg const(v) \wedge \neg rev\_it(it_1) \wedge$$
$$\wedge * it_1 = v_j \wedge *it_2 = v_j \wedge \neg rev\_it(it_1) \wedge \neg rev\_it(it_2)\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge \neg const(v) \wedge type(it_1) = vector < T > \wedge type(it_2) = vector < T > \wedge$$
$$\wedge (*it_1 =? \vee *it_2 =?)\} v.erase(it_1, it_2); \{Undef\}$$

$$\{v =< v_0, v_1, \ldots, v_n > \wedge \neg const(v) \wedge type(it) = vector < T > \wedge *it = v_i \wedge \neg rev\_it(it)\}$$

$$v.insert(it, k, t); \{v =< v_0, v_1, \ldots, v_{i-1}, t_1, t_2, \ldots, t_k, v_i, v_{i+1}, \ldots, v_n > \wedge \neg const(v) \wedge$$
$$\wedge \forall i \in [1..k] : t_i = t \wedge type(it) = vector < T > \wedge * it = v_i \wedge \neg rev\_it(it)\}$$

$$\{x =< x_0, x_1, \ldots, x_n > \wedge y =< y_0, y_1, \ldots, y_k > \wedge (n \neq k \vee \exists i \in [0..n] : x_i \neq y_i) \wedge$$

$$\wedge EC(T, operator ==)\}$$

$$l = x == y;$$

$$\{x =< x_0, x_1, \ldots, x_n > \wedge y =< y_0, y_1, \ldots, y_k > \wedge l = false\}$$

$$\{x =< x_0, x_1, \ldots, x_n > \wedge y =< y_0, y_1, \ldots, y_k > \wedge EC(T, operator ==)$$

$$\wedge \neg((x == y) = false)\} l = x == y; \{x =< x_0, x_1, \ldots, x_n > \wedge y =< y_0, y_1, \ldots, y_k > \wedge l = true\}$$

$$\{x =< x_0, x_1, \ldots, x_n > \wedge y =< y_0, y_1, \ldots, y_k > \wedge (n < k \wedge \forall i \in [0..n] : x_i = y_i \vee$$

$$\vee \exists j \in [0..n] : x_j < y_j \wedge \forall s \in [0..j-1] : x_s = y_s) \wedge LTC(T, operator <)\}$$

$$l = x < y; \{l = true \wedge x =< x_0, x_1, \ldots, x_n > \wedge y =< y_0, y_1, \ldots, y_k >\}$$

$$\{x =< x_0, x_1, \ldots, x_n > \wedge y =< y_0, y_1, \ldots, y_k > \wedge \neg(x < y) \wedge LTC(T, operator <)\}$$

$$l = x < y;$$

$$\{l = true \wedge x =< x_0, x_1, \ldots, x_n > \wedge y =< y_0, y_1, \ldots, y_k >\}$$

A.3.2. A stack osztály

$\mathcal{I} = \{true\}$

$$\{true\} stack < T > st; \{st =<> \wedge \neg const(st)\}$$

$$\{true\} stack < T > \ const\ st; \{st =<> \wedge const(st)\}$$

$$\{st =< x_1, x_2, \ldots, x_n > \wedge \neg const(st)\} st.push(y); \{st =< x_1, x_2, \ldots, x_n, y > \wedge \neg const(st)\}$$

$$\{st =< x_1, x_2, \ldots, x_n > \wedge \neg const(st)\} c = st.top(); \{st =< x_1, x_2, \ldots, x_n > \wedge$$
$$\wedge c = c.operator = (x_n); \wedge \neg const(st)\}$$

$$\{st =< x_1, x_2, \ldots, x_n > \wedge \neg const(st)\}\ T\ c = st.top(); \{st =< x_1, x_2, \ldots, x_n > \wedge$$

$$\wedge c = c.cpctor(x_n); \wedge \neg const(st)\}$$

$$\{st =< x_1, x_2, \ldots, x_n > \wedge const(st)\}c = st.top(); \{st =< x_1, x_2, \ldots, x_n > \wedge$$
$$\wedge c = c.operator = (x_n) \wedge const(st)\}$$
$$\{st =< x_1, x_2, \ldots, x_n > \wedge const(st)\}T\ c = st.top(); \{st =< x_1, x_2, \ldots, x_n > \wedge$$
$$\wedge c = c.cpctor(x_n) \wedge const(st)\}$$
$$\{st =< x_1, x_2, \ldots, x_n > \wedge \neg const(st)\}st.pop(); \{st =< x_1, x_2, \ldots, x_{n-1} > \wedge \neg const(st)\}$$
$$\{st =<> \wedge \neg const(st)\}c = st.pop(); \{st =<> \wedge \neg const(st)\}$$
$$\{st =<>\}m = st.size(); \{st =<> \wedge m = 0\}$$
$$\{st =< x_1, x_2, \ldots, x_n > \wedge \neg const(st)\}st.top = c; \{st =< x_1, x_2, \ldots, x_{n-1}, c > \wedge \neg const(st)\}$$
$$\{st =<> \wedge const(st)\}c = st.top(); \{Undef\}$$
$$\{st =<> \wedge const(st)\}st.top() = c; \{Undef\}$$
$$\{st =<> \wedge \neg const(st)\}c = st.top(); \{Undef\}$$
$$\{st =<> \wedge \neg const(st)\}st.top() = c; \{Undef\}$$
$$\{st =< x_1, x_2, \ldots, x_n >\}m = st.size(); \{st =< x_1, x_2, \ldots, x_n > \wedge m = n\}$$
$$\{st =<>\}l = st.empty(); \{st =<> \wedge l = true\}$$
$$\{st =< x_1, x_2, \ldots, x_n >\}l = st.empty(); \{st =< x_1, x_2, \ldots, x_n > \wedge l = false\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge (n \neq k \vee \exists i \in [1..n] : x_i \neq y_i) \wedge$$
$$\wedge EC(T, operator ==)\}$$
$$l = x == y;$$
$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge l = false\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge EC(T, operator ==)$$
$$\wedge \neg((x == y) = false)\}l = x == y; \{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge l = true\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge (n < k \wedge \forall i \in [1..n] : x_i = y_i \vee$$
$$\vee \exists j \in [1..n] : x_j < y_j \wedge \forall s \in [1..j-1] : x_s = y_s) \wedge LTC(T, operator <)\}$$
$$l = x < y; \{l = true \wedge x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k >\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge \neg(x < y) \wedge LTC(T, operator <)\}$$
$$l = x < y;$$
$$\{l = true \wedge x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k >\}$$

ahol: $x_i, y, a_i, b_i, c \in T, i \in \mathbb{N}, l \in bool, m \in int, st, y, x \in stack < T >$.

A.3.3. A queue osztály

$\mathcal{I} = \{true\}$

$$\{true\}queue < T > \ que; \{que =<> \wedge \neg const(que)\}$$

$$\{true\}queue < T > \ const\ que; \{que =<> \wedge const(que)\}$$

$$\{que =< x_1, x_2, \ldots, x_n > \wedge \neg const(que)\}que.pop(); \{que =< x_2, x_3, \ldots, x_n > \wedge \neg const(que)\}$$

$$\{que =<>\}l = que.empty(); \{que =<> \wedge l = true\}$$

$$\{que =< x_1, x_2, \ldots, x_n >\}l = que.empty(); \{l = false \wedge que =< x_1, x_2, \ldots, x_n >\}$$

$$\{que =< x_1, x_2, \ldots, x_n >\}s = que.size(); \{s = n \wedge que =< x_1, x_2, \ldots, x_n >\}$$

$$\{que =< x_1, x_2, \ldots, x_n > \wedge \neg const(que)\}que.push(e); \{que =< x_1, x_2, \ldots, x_n, e > \wedge$$
$$\wedge \neg const(que)\}$$

$$\{que =< x_1, x_2, \ldots, x_n > \wedge \neg const(que)\}que.front() = e; \{que =< e, x_2, \ldots, x_n > \wedge$$
$$\wedge \neg const(que)\}$$

$$\{que =<> \wedge \neg const(que)\}que.front() = e; \{Undef\}$$

$$\{que =< x_1, x_2, \ldots, x_n > \wedge const(que)\}T\ e = que.front(); \{const(que) \wedge$$
$$\wedge que =< x_1, x_2, \ldots, x_n > \wedge e = e.cpctor(x_1)\}$$

$$\{que =< x_1, x_2, \ldots, x_n > \wedge const(que)\}e = que.front(); \{const(que) \wedge$$
$$\wedge que =< x_1, x_2, \ldots, x_n > \wedge e = e.operator = (x_1)\}$$

$$\{que =< x_1, x_2, \ldots, x_n > \wedge \neg const(que)\}e = que.front(); \{\neg const(que) \wedge$$
$$\wedge e = e.operator = (x_1) \wedge que =< x_1, x_2, \ldots, x_n\}$$

$$\{que =< x_1, x_2, \ldots, x_n > \wedge \neg const(que)\}T\ e = que.front(); \{\neg const(que) \wedge$$
$$\wedge e = e.cpctor(x_1) \wedge que =< x_1, x_2, \ldots, x_n\}$$

$$\{que =<> \wedge const(que)\}T\ e = que.front(); \{que =<> \wedge e = e.cpctor(?)\}$$

$$\{que =<> \wedge const(que)\}e = que.front(); \{que =<> \wedge e = e.operator = (?)\}$$

$$\{que =<> \wedge \neg const(que)\}T\ e = que.front(); \{que =<> \wedge e = e.cpctor(?)\}$$

$$\{que =<> \wedge \neg const(que)\}e = que.front(); \{que =<> \wedge e = e.operator = (?)\}$$

$$\{que =< x_1, x_2, \ldots, x_n > \wedge \neg const(que)\}e = que.front(); \{\neg const(que) \wedge$$
$$\wedge e = e.operator = (x_1) \wedge que =< x_1, x_2, \ldots, x_n\}$$

$$\{que =< x_1, x_2, \ldots, x_n > \wedge \neg const(que)\}T\ e = que.front(); \{\neg const(que) \wedge$$
$$\wedge e = e.cpctor(x_1) \wedge que =< x_1, x_2, \ldots, x_n\}$$

$$\{que =< x_1, x_2, \ldots, x_n > \wedge \neg const(que)\}que.back() = e; \{que =< x_1, x_2, \ldots, x_{n-1}, e > \wedge$$
$$\wedge \neg const(que)\}$$

$$\{que =<>\}T\ e = que.back(); \{Undef\}$$
$$\{que =<>\}e = que.back(); \{Undef\}$$
$$\{que =<> \wedge \neg const(que)\}que.back() = e; \{Undef\}$$

$$\{que =< x_1, x_2, \ldots, x_n > \wedge const(que)\}T\ e = que.back(); \{const(que) \wedge$$
$$\wedge que =< x_1, x_2, \ldots, x_n > \wedge e = e.cpctor(x_n)\}$$

$$\{que =< x_1, x_2, \ldots, x_n > \wedge const(que)\}e = que.back(); \{const(que) \wedge$$
$$\wedge que =< x_1, x_2, \ldots, x_n > \wedge e = e.operator = (x_n)\}$$

$$\{que =< x_1, x_2, \ldots, x_n > \wedge \neg const(que)\}e = que.back(); \{\neg const(que) \wedge$$
$$\wedge e = e.operator = (x_n) \wedge que =< x_1, x_2, \ldots, x_n >\}$$

$$\{que =< x_1, x_2, \ldots, x_n > \wedge \neg const(que)\} T\ e = que.back(); \{\neg const(que) \wedge$$

$$\wedge e = e.cpctor(x_n) \wedge que =< x_1, x_2, \ldots, x_n >\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge (n \neq k \vee \exists i \in [1..n] : x_i \neq y_i) \wedge$$

$$\wedge EC(T, operator ==)\}$$

$$l = x == y;$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge l = false\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge EC(T, operator ==)$$

$$\wedge \neg((x == y) = false)\} l = x == y; \{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge l = true\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge (n < k \wedge \forall i \in [1..n] : x_i = y_i \vee$$

$$\vee \exists j \in [1..n] : x_j < y_j \wedge \forall s \in [1..j-1] : x_s = y_s) \wedge LTC(T, operator <)\}$$

$$l = x < y; \{l = true \wedge x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k >\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge \neg(x < y) \wedge LTC(T, operator <)\}$$

$$l = x < y;$$

$$\{l = true \wedge x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k >\}$$

A.3.4. A priority queue osztály

$\mathcal{I} = \{LTC(T, operator <)\}$[2]

$$\{true\} priority\_queue < T > prq; \{prq =<> \wedge \neg const(prq)\}$$

$$\{true\} priority\_queue < T > \ const\ prq; \{prq =<> \wedge const(prq)\}$$

[2]Ennek részletesebb indoklása a set osztálynál megtalálható.

$$\{prq =< x_1, x_2, \ldots, x_n > \wedge \neg const(prq)\} prq.push(e); \{prq =< x_1, x_2, \ldots, x_n, e > \wedge \neg const(prq)\}$$

$$\{prq =< x_1, x_2, \ldots, x_n >\} m = prq.size(); \{m = n \wedge prq =< x_1, x_2, \ldots, x_n >\}$$

$$\{prq =<>\} l = prq.empty(); \{prq =<> \wedge l = true\}$$

$$\{prq =< x_1, x_2, \ldots, x_n >\} l = prq.empty(); \{prq =< x_1, x_2, \ldots, x_n > \wedge l = false\}$$

$$\{prq =< x_1, x_2, \ldots, x_n >\} e = prq.top(); \{prq =< x_1, x_2, \ldots, x_n > \wedge e = e.operator = (x_i) \wedge$$
$$\wedge \forall (j < i) : x_j < x_i \wedge \forall (k > i) : \neg (x_i < x_k)\}$$

$$\{prq =< x_1, x_2, \ldots, x_n >\} T\ e = prq.top(); \{prq =< x_1, x_2, \ldots, x_n > \wedge e = e.cpctor(x_i) \wedge$$
$$\wedge \forall (j < i) : x_j < x_i \wedge \forall (k > i) : \neg (x_i < x_k)\}$$

$$\{prq =<>\} e = prq.top(); \{Undef\}$$
$$\{prq =<>\} T\ e = prq.top(); \{Undef\}$$

$$\{prq =< x_1, x_2, \ldots, x_n > \wedge x_i = prq.top() \wedge \neg const(prq)\} prq.pop();$$
$$\{prq =< x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n > \wedge \neg const(prq);\}$$

A.3.5. A set osztály

$\mathcal{I} = \{LTC(T, operator <)\}$

A C++ nyelv ún. „lusta példányosítást” használ [29]. Ennek lényege, hogy csak akkor próbálkozik egy adott kódrészlet sablon szerinti létrehozásával, ha arra hivatkozás történik. Megfordítva: ha egy sablonra nem hivatkozunk, a fordítónak tilos azt példányosítania.

Ennek egyik következmenyeként a fordítók elfogadnák egy olyan `set<T>` létrehozását, amely T paraméter nem rendelkezik `operator<` művelettel. Egy ilyen set-be persze egyetlen elemet sem helyezhetünk el, hiszen az `operator<`-et felhasználó `insert()` példányosulása minden esetben szintaktikus hibát eredményezne.

Módszerünkkel ez az ellentmondás kiszűrhető, és már a fejlesztés korai szakaszaiban javítani tudjuk a hibát.

$$\{true\} set < T > \ s; \{s =<> \wedge \neg const(s)\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge \neg const(s) \wedge \exists j : 1 \leq j \leq n : a = x_j\} pair < set < T >:: iterator, bool >$$
$$p = s.insert(a); \{s =< x_1, x_2, \ldots, x_n > \wedge \neg const(s) \wedge p.second = false \wedge$$
$$*(p.first) = x_j \wedge \neg rev\_it(p.first) \wedge \neg const\_it(p.first) \wedge type(p.first) = set < T >\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge \neg const(s) \wedge \forall i : 1 \leq i \leq n : x_i \neq a\} pair < set < T >:: iterator, bool >$$
$$p = s.insert(a); \{p.second = true \wedge \neg const(s) \wedge type(p.first) = set < T > \wedge$$
$$\wedge s =< x_1, x_2, \ldots, x_j, a, x_{j+1}, \ldots, x_n > \wedge x_j < a \wedge a < x_{j+1} \wedge$$
$$\wedge (p.first) = a \wedge \neg rev\_it(p.first) \wedge \neg const\_it(p.first)\}$$

$$\{s =<>\} l = s.empty(); \{s =<> \wedge l = true\}$$
$$\{s =< x_1, x_2, \ldots, x_n >\} l = s.empty(); \{s =< x_1, x_2, \ldots, x_n > \wedge l = false\}$$
$$\{s =< x_1, x_2, \ldots, x_n >\} d = s.size(); \{s =< x_1, x_2, \ldots, x_n > \wedge d = n\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge \neg const(s) \wedge \exists j : 1 \leq j \leq n : a = x_j \wedge EC(T, operator ==)\}$$
$$i = s.erase(a); \{i = 1 \wedge \neg const(s) \wedge s =< x_1, x_2, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n >\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge \neg const(s) \wedge \forall i : 1 \leq i \leq n : x_i \neq a \wedge EC(T, operator ==)\}$$
$$i = s.erase(a); \{s =< x_1, x_2, \ldots, x_n > \wedge \neg const(s) \wedge i = 0\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge \exists j : 1 \leq j \leq n : a = x_j \wedge EC(T, operator ==)\} d = s.count(a);$$
$$\{d = 1 \wedge s =< x_1, x_2, \ldots, x_n >\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge \forall i : 1 \leq i \leq n : x_i \neq a \wedge EC(T, operator ==)\}$$
$$d = s.count(a); \{d = 0 \wedge s =< x_1, x_2, \ldots, x_n >\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge \exists j : 1 \leq j \leq n : a = x_j \wedge type(it) = set < T > \wedge \neg rev\_it(it) \wedge$$
$$\wedge const\_it(it) = C\} it = s.lower\_bound(a); \{\neg rev\_it(it) \wedge$$
$$\wedge s =< x_1, x_2, \ldots, x_n > \wedge * it = x_j \wedge type(it) = set < T > \wedge const\_it(it) = C\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge \forall i : 1 \leq i \leq n : x_i \neq a \wedge \neg rev\_it(it) \wedge$$
$$\wedge const\_it(it) = C \wedge a < x_n \wedge type(it) = set < T >\} it = s.lower\_bound(a); \{\neg rev\_it(it) \wedge$$
$$\wedge s =< x_1, x_2, \ldots, x_n > \wedge * it = x_k \wedge type(it) = set < T > \wedge const\_it(it) = C \wedge$$
$$\wedge x_k > a \wedge x_{k-1} < a\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge a > x_n \wedge \neg rev\_it(it) \wedge type(it) = set < T > \wedge$$
$$\wedge const\_it(it) = C\} it = s.lower\_bound(a); \{\neg rev\_it(it) \wedge type(it) = set < T > \wedge$$
$$\wedge s =< x_1, x_2, \ldots, x_n > \wedge * it = s.end \wedge const\_it(it) = C\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge \neg rev\_it(it) \wedge a < x_n \wedge type(it) = set < T > \wedge$$
$$\wedge const\_it(it) = C\} it = s.upper\_bound(a); \{\neg rev\_it(it) \wedge type(it) = set < T > \wedge$$
$$\wedge s =< x_1, x_2, \ldots, x_n > \wedge * it = x_k \wedge const\_it(it) = C \wedge$$
$$\wedge x_k > a \wedge x_{k-1} < a\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge a > x_n \wedge \neg rev\_it(it) \wedge type(it) = set < T > \wedge$$
$$\wedge const\_it(it) = C\} it = s.upper\_bound(a); \{\neg rev\_it(it) \wedge type(it) = set < T > \wedge$$
$$\wedge s =< x_1, x_2, \ldots, x_n > \wedge * it = s.end \wedge const\_it(it) = C\}$$

$$\{s =< x_1, x_2, \ldots, x_n >\} pair < set < T >:: iterator, set < T >:: iterator > \ p =$$
$$s.equal\_range(a); \{s =< x_1, x_2, \ldots, x_n > \wedge p.first = s.lower\_bound(a) \wedge$$
$$\wedge p.second = s.upper\_bound(a)\}$$

$$\{s =< x_1, x_2, \ldots, x_n >\} pair < set < T >:: const\_iterator, set < T >:: iterator > \ p =$$
$$s.equal\_range(a); \{s =< x_1, x_2, \ldots, x_n > \wedge p.first = s.lower\_bound(a) \wedge$$
$$\wedge p.second = s.upper\_bound(a)\}$$

$$\{s =< x_1, x_2, \ldots, x_n >\} pair < set < T >:: iterator, set < T >:: const\_iterator > \ p =$$
$$s.equal\_range(a); \{s =< x_1, x_2, \ldots, x_n > \wedge p.first = s.lower\_bound(a) \wedge$$
$$\wedge p.second = s.upper\_bound(a)\}$$

$$\{s =< x_1, x_2, \ldots, x_n >\} pair < set < T >:: const\_iterator, set < T >:: const\_iterator > \ p =$$
$$s.equal\_range(a); \{s =< x_1, x_2, \ldots, x_n > \wedge p.first = s.lower\_bound(a) \wedge$$
$$\wedge p.second = s.upper\_bound(a)\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge \exists j : 1 \leq j \leq n : a = x_j \wedge type(it) = set < T > \wedge$$
$$\wedge const\_it(it) = c \wedge rev\_it(it) = R \wedge EC(T, operator ==)\} it = s.find(a); \{*it = x_j \wedge$$
$$\wedge s =< x_1, x_2, \ldots, x_n > \wedge const\_it(it) = c \wedge rev\_it(it) = R \wedge type(it) = set < T >\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge \forall i : 1 \leq i \leq n : x_i \neq a \wedge \neg rev\_it(it) \wedge$$
$$\wedge const\_it(it) = c \wedge EC(T, operator ==) \wedge type(it) = set < T >\} it = s.find(a);$$
$$\{*it = s.end \wedge type(it) = set < T > \wedge s =< x_1, x_2, \ldots, x_n > \wedge$$
$$\wedge const\_it(it) = c \wedge \neg rev(it)\}$$

$$\{a =< x_1, x_2, \ldots, x_n > \wedge \neg const(a) \wedge b =< y_1, y_2, \ldots, y_k > \wedge$$
$$\wedge cat(it1) = In \wedge cat(it2) = In \wedge const\_it(it1) = C_1 \wedge const\_it(it2) = C_2 \wedge$$
$$\wedge * it1 = y_j \wedge *it2 = y_m \wedge rev\_it(it1) = rev\_it(it2) \wedge rev\_it(it1) = R\}$$
$$a.insert(it1, it2); \{a =< c_1, c_2, \ldots, c_t > \wedge \neg const(a) \wedge b =< y_1, y_2, \ldots, y_k > \wedge$$

$$\wedge\forall i : 1 \leq i \leq t-1 : c_i < c_{i+1} \wedge \forall i : 1 \leq i \leq t : c_i = y_s (1 \leq s \leq n \vee$$
$$\vee c_i = x_s (j \leq s < m) \wedge cat(it1) = In \wedge cat(it2) = In \wedge *it1 = y_j \wedge$$
$$\wedge * it2 = y_m \wedge const\_it(it1) = C_1 \wedge const\_it(it2) = C_2 \wedge rev\_it(it1) = R \wedge$$
$$\wedge rev\_it(it2) = R\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge type(it1) = set < T > \wedge type(it2) = set < T > \wedge \neg rev\_it(it1) \wedge$$
$$\wedge rev\_it(it2) \wedge *it1 = x_j \wedge *it2 = x_k \wedge \neg const(s) \wedge const\_it(it1) = C_1 \wedge$$
$$const\_it(it2) = C_2\} s.erase(it1, it2); \{type(it1) = set < T > \wedge type(it2) = set < T > \wedge$$
$$\wedge s =< x_1, x_2, \ldots, x_{j-1}, x_k, \ldots, x_n > \wedge \neg const(s) \wedge *it2 = x_k \wedge$$
$$\wedge *it1 =? \wedge \neg rev\_it(it1) \wedge \neg rev\_it(it2) \wedge const\_it(it1) = C_1 \wedge const\_it(it2) = C_2\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge type(it) = set < T > \wedge \neg rev\_it(it) \wedge *it = x_k \wedge$$
$$\wedge const\_it(it) = C \wedge \neg const(s)\} s.erase(it); \{\neg const(s) \wedge *it =? \wedge$$
$$\wedge s =< x_1, x_2, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n > \wedge \neg rev\_it(it) \wedge const\_it(it) = C$$
$$\wedge type(it) = set < T >\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge type(it) = set < T > \wedge \neg rev\_it(it) \wedge *it = s.end \wedge$$
$$\wedge const\_it(it) = C \wedge \neg const(s)\} s.erase(it); \{Undef\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge type(it) = set < T > \wedge \neg rev\_it(it) \wedge *it =?\}$$
$$s.erase(it); \{Undef\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge \neg const(s) \wedge type(it1) = set < T > \wedge \neg rev\_it(it1) \wedge$$
$$\wedge const\_it(it1) = C_1 \wedge type(it2) = set < T > \wedge \neg rev\_it(it2) \wedge const\_it(it2) = C_2 \wedge$$
$$\wedge \exists j : 1 \leq j \leq n : v = x_j \wedge *it1 = x_k\} it2 = s.insert(it1, v); \{*it2 = x_j \wedge type(it1) = set < T > \wedge$$
$$\wedge s =< x_1, x_2, \ldots, x_n > \wedge \neg const(s) \wedge type(it2) = set < T > \wedge \neg rev\_it(it1) \wedge$$

$$\wedge const\_it(it1) = C_1 \wedge const\_it(it2) = C_2 \wedge \neg rev\_it(it2) \wedge *it1 = x_k\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge \neg const(s) \wedge type(it1) = set < T > \wedge \neg rev\_it(it1) \wedge$$
$$\wedge const\_it(it1) = C_1 \wedge type(it2) = set < T > \wedge \neg rev\_it(it2) \wedge const\_it(it2) = C_2 \wedge$$
$$\wedge \forall i : 1 \leq i \leq n : x_i \neq v \wedge *it1 = x_k\} it2 = s.insert(it1, v); \{\neg const(s) \wedge$$
$$\wedge s =< x_1, x_2, \ldots, x_j, v, x_{j+1} > \wedge *it2 = v \wedge x_j < v \wedge v < x_{j+1} \wedge *it1 = x_k \wedge$$
$$\wedge type(it1) = set < T > \wedge type(it2) = set < T > \wedge \neg rev\_it(it1) \wedge \neg rev\_it(it2) \wedge$$
$$\wedge const\_it(it1) = C_1 \wedge const\_it(it2) = C_2\}$$

$$\{s =< x_1, x_2, \ldots, x_n > \wedge \neg const(s) \wedge type(it1) = set < T > \wedge \neg rev\_it(it1) \wedge$$
$$\wedge *it1 =? \wedge type(it2) = set < T > \wedge \neg rev\_it(it2)\} it2 = s.insert(it1, v); \{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge (n < k \wedge \forall i \in [1..n] : x_i = y_i \vee$$
$$\vee \exists j \in [1..n] : x_j < y_j \wedge \forall s \in [1..j-1] : x_s = y_s)\} l = x < y; \{l = true \wedge$$
$$\wedge x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k >\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge \neg(x < y)\} l = x < y;$$
$$\{l = true \wedge x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k >\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge EC(T, operator ==) \wedge$$
$$\wedge (n \neq k \vee \exists i \in [1..n] : x_i \neq y_i\}$$
$$l = x == y;$$
$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge l = false\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge \neg((x == y) = false) \wedge EC(T, operator ==)\}$$
$$l = x == y; \{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge l = true\}$$

A.3.6. A multiset osztály

$\mathcal{I} = \{LTC(T, operator <)\}$[3]

$$\{true\}multiset < T > x; \{x =<> \wedge \neg const(x)\}$$
$$\{true\}multiset < T > \ const\ x; \{x =<> \wedge const(x)\}$$

$$\{x =<>\} l = x.empty(); \{l = true \wedge x =<>\}$$
$$\{x =< x_1, x_2, \ldots, x_n >\} l = x.empty(); \{l = false \wedge x =< x_1, x_2, \ldots, x_n >\}$$
$$\{x =< x_1, x_2, \ldots, x_n >\} s = x.size(); \{s = n \wedge x =< x_1, x_2, \ldots, x_n >\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge \neg rev\_it(it) \wedge type(it) = mset < T > \wedge const\_it(it) = C\}$$
$$it = x.insert(v); \{*it = v \wedge \neg rev\_it(it) \wedge const\_it(it) = C \wedge type(it) = mset < T > \wedge$$
$$\wedge x =< x_1, x_2, \ldots, x_k, v, x_{k+1}, \ldots, x_n > \wedge x_k \leq v < x_{k+1}\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge \neg rev\_it(it1) \wedge type(it1) = mset < T > \wedge const\_it(it1) = C_1 \wedge$$
$$\wedge type(it2) = mset < T > \wedge \neg rev\_it(it2) \wedge *it2 = x_j \wedge const\_it(it2) = C_2\}$$
$$it1 = x.insert(it2, v); \{type(it2) = mset < T > \wedge \neg rev\_it(it2) \wedge const\_it(it2) = C_2 \wedge$$
$$\{\wedge * it2 = x_j \wedge x =< x_1, x_2, \ldots, x_k, v, x_{k+1}, \ldots, x_n > \wedge x_k \leq v < x_{k+1} \wedge$$
$$\wedge * it1 = v \wedge type(it1) = mset < T > \wedge \neg rev\_it(it1) \wedge const\_it(it1) = C_1\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge type(it) = mset < T > \wedge \neg rev\_it(it) \wedge *it =?\}$$
$$x.erase(it); \{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge type(it) = mset < T > \wedge \neg rev\_it(it) \wedge *it = x_k \wedge$$
$$\wedge const\_it(it) = C \wedge \neg const(x)\} x.erase(it); \{\neg const(x) \wedge *it =? \wedge$$
$$\wedge x =< x_1, x_2, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n > \wedge \neg rev\_it(it) \wedge const\_it(it) = C$$

[3]Ennek részletesebb indoklása a set osztálynál megtalálható.

$$\wedge type(it) = mset < T >\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge type(it) = mset < T > \wedge \neg rev\_it(it) \wedge *it = x.end \wedge$$
$$\wedge const\_it(it) = C \wedge \neg const(x)\} x.erase(it); \{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge type(it1) = mset < T > \wedge type(it2) = mset < T > \wedge \neg rev\_it(it1) \wedge$$
$$\wedge rev\_it(it2) \wedge *it1 = x_j \wedge *it2 = x_k \wedge \neg const(x) \wedge const\_it(it1) = C_1 \wedge$$
$$const\_it(it2) = C_2\} x.erase(it1, it2); \{type(it1) = mset < T > \wedge type(it2) = mset < T > \wedge$$
$$\wedge x =< x_1, x_2, \ldots, x_{j-1}, x_k, \ldots, x_n > \wedge \neg const(x) \wedge *it2 = x_k \wedge$$
$$\wedge *it1 =? \wedge \neg rev\_it(it1) \wedge \neg rev\_it(it2) \wedge const\_it(it1) = C_1 \wedge const\_it(it2) = C_2\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge x_k = x_{k+1} = \ldots = x_{k+m} = v \wedge \neg const(x) \wedge (EC(T, operator ==)\}$$
$$d = x.erase(v); \{\neg const(x) \wedge x =< x_1, x_2, \ldots, x_{k-1}, x_{k+m+1}, \ldots, x_n > \wedge d = m{+}1\}$$

$$\{x =< x_1, x_2, \ldots, x_n >\} a = count(v); \{x =< x_1, x_2, \ldots, x_n > \wedge a = \sum_{k=1}^{n} \chi(x_i = v)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge type(it) = mset < T > \wedge \neg rev\_it(it) \wedge const\_it(it) = C \wedge$$
$$\wedge x.count(a) > 0\} it = x.find(a); \{x =< x_1, x_2, \ldots, x_n > \wedge type(it) = mset < T > \wedge$$
$$\wedge * it = x_k \wedge x_k = a \wedge x_{k-1} < a \vee *it = x_1 \wedge const\_it(it) = C \wedge \neg rev\_it(it)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge type(it) = mset < T > \wedge \neg rev\_it(it) \wedge const\_it(it) = C \wedge$$
$$\wedge x.count(a) = 0\} it = x.find(a); \{x =< x_1, x_2, \ldots, x_n > \wedge type(it) = mset < T > \wedge$$
$$\wedge * it = x.end \wedge \neg rev\_it(it) \wedge const\_it(it) = C\}$$

$$\{x =< x_1, x_2, \ldots, x_n >\} pair < multiset < T >:: const\_iterator,$$

$$multiset < T >:: const\_iterator > \ p = x.equal\_range(a); \{x =< x_1, x_2, \ldots, x_n > \wedge$$
$$\wedge p.first = x.lower\_bound(a) \wedge p.second = x.upper\_bound(a)\}$$

$$\{x =< x_1, x_2, \ldots, x_n >\} pair < multiset < T >:: iterator,$$
$$multiset < T >:: const\_iterator > \ p = x.equal\_range(a); \{x =< x_1, x_2, \ldots, x_n > \wedge$$
$$\wedge p.first = x.lower\_bound(a) \wedge p.second = x.upper\_bound(a)\}$$

$$\{x =< x_1, x_2, \ldots, x_n >\} pair < multiset < T >:: const\_iterator,$$
$$multiset < T >:: iterator > \ p = x.equal\_range(a); \{x =< x_1, x_2, \ldots, x_n > \wedge$$
$$\wedge p.first = x.lower\_bound(a) \wedge p.second = x.upper\_bound(a)\}$$

$$\{x =< x_1, x_2, \ldots, x_n >\} pair < multiset < T >:: iterator,$$
$$multiset < T >:: iterator > \ p = x.equal\_range(a); \{x =< x_1, x_2, \ldots, x_n > \wedge$$
$$\wedge p.first = x.lower\_bound(a) \wedge p.second = x.upper\_bound(a)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge (n < k \wedge \forall i \in [1..n] : x_i = y_i \vee$$
$$\vee \exists j \in [1..n] : x_j < y_j \wedge \forall s \in [1..j-1] : x_s = y_s)\} l = x < y; \{l = true \wedge$$
$$\wedge x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k >\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge \neg(x < y)\} l = x < y;$$
$$\{l = true \wedge x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k >\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge (n \neq k \vee \exists i \in [1..n] : x_i \neq y_i\}$$
$$l = x == y;$$
$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge l = false\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge \neg((x == y) = false)\} l = x == y;$$
$$\{x =< x_1, x_2, \ldots, x_n > \wedge y =< y_1, y_2, \ldots, y_k > \wedge l = true\}$$

A.4. Algoritmusok specifikációja

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it_1) = In \wedge cat(it_2) = In \wedge cat(it_3) = In \wedge *it_1 = x_i \wedge$$
$$\wedge * it_2 = x_j \wedge \exists k \in [i, j) : x_k = a \wedge \forall m \in [i, k) : x_m \neq a \wedge$$
$$\wedge \neg rev\_it(it1) \wedge \neg rev\_it(it_2) \wedge rev\_it(it_3) = L \wedge EC(T, operator ==)\}$$
$$it_3 = find(it_1, it_2, a);$$
$$\{*it_3 = x_k \wedge rev\_it(it_3) = L \wedge cat(it_1) = In \wedge cat(it_2) = In \wedge cat(it_3) = In \wedge$$
$$\{x =< x_1, x_2, \ldots, x_n > \wedge \neg rev\_it(it1) \wedge \neg rev\_it(it_2) \wedge *it_2 = x_j \wedge *it_1 = x_i\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it_1) = In \wedge cat(it_2) = In \wedge cat(it_3) = In \wedge *it_1 = x_i \wedge$$
$$\wedge * it_2 = x_j \wedge \forall k \in [i, j) : x_k \neq a \wedge \neg rev\_it(it1) \wedge$$
$$\wedge \neg rev\_it(it_2) \wedge rev\_it(it_3) = L \wedge EC(T, operator ==)\}$$
$$it_3 = find(it_1, it_2, a);$$
$$\{*it_3 = x_j \wedge x =< x_1, x_2, \ldots, x_n > \wedge \neg rev\_it(it1) \wedge \neg rev\_it(it_2) \wedge *it_2 = x_j \wedge$$
$$\wedge * it_1 = x_i \wedge rev\_it(it_3) = L \wedge cat(it_1) = In \wedge cat(it_2) = In \wedge cat(it_3) = In\}$$

$$\{cat(it_1) = In \wedge cat(it_2) = In \wedge cat(it_3) = In \wedge (*it_2 =? \vee *it_1 =?) \wedge$$
$$\wedge EC(T, operator ==)\}$$
$$it_3 = find(it_1, it_2, a);$$
$$\{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it_1) = Bi \wedge cat(it_2) = Bi \wedge *it_1 = x_k \wedge$$
$$\wedge * it_2 = x_l \wedge \neg const(x)\}$$
$$reverse(it_1, it_2);$$
$$\{x =< x_1, x_2, \ldots, x_{k-1}, x_{l-1}, x_{l-2}, \ldots, x_k, x_l, x_{l+1}, \ldots, x_n > \wedge \neg const(x) \wedge$$
$$\wedge * it_1 = x_l \wedge *it_2 = x_{l-1} \wedge cat(it_1) = Bi \wedge cat(it_2) = Bi\}$$

$$\{cat(it_1) = Bi \wedge cat(it_2) = Bi \wedge (*it_1 =? \vee *it_2 =?)\}$$

$$reverse(it_1, it_2);$$
$$\{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it_1) = In \wedge cat(it_2) = In \wedge *it_1 = x_k \wedge *it_2 = x_l \wedge$$
$$\wedge \neg rev\_it(it_1) \wedge \neg rev\_it(it_2)\}$$
$$for\_each(it_1, it_2, f);$$
$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it_1) = In \wedge cat(it_2) = In \wedge *it_1 = x_k \wedge *it_2 = x_l \wedge$$
$$\wedge SEQ(f(x_k), f(x_{k+1}), \ldots, f(x_{l-1})) \wedge \neg rev\_it(it_1) \wedge \neg rev\_it(it_2)\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it_1) = In \wedge cat(it_2) = In \wedge *it_1 = x_l \wedge *it_2 = x_k \wedge$$
$$\wedge rev\_it(it_1) \wedge rev\_it(it_2)\}$$
$$for\_each(it_1, it_2, f);$$
$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it_1) = In \wedge cat(it_2) = In \wedge *it_1 = x_l \wedge *it_1 = x_k \wedge$$
$$\wedge SEQ(f(x_l), f(x_{l-1}), \ldots, f(x_{k+1})) \wedge rev\_it(it_1) \wedge rev\_it(it_2)\}$$

$$\{cat(it_1) = In \wedge cat(it_2) = In \wedge *it_1 =? \wedge *it_2 =?\}$$
$$for\_each(it_1, it_2, f);$$
$$\{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge cat(it_1) = In \wedge cat(it_2) = In \wedge *it_1 = x_k \wedge *it_2 = x_l \wedge i = c \wedge$$
$$\wedge EC(T, operator ==)\}$$
$$count(it_1, it_2, v, i);$$
$$\{i = c + \sum_{m=k}^{l-1} \chi(x_m = v) \wedge x =< x_1, x_2, \ldots, x_n > \wedge cat(it_1) = In \wedge cat(it_2) = In \wedge$$
$$\wedge * it_1 = x_k \wedge *it_2 = x_j\}$$

$$\{cat(it_1) = In \wedge cat(it_2) = In \wedge (*it_1 =? \vee *it_2 =?) \wedge EC(T, operator ==)\}$$

$$count(it_1, it_2, v, i);$$
$$\{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge *it_1 = x_i \wedge *it_2 = x_j \wedge cat(it_1) = Ran \wedge cat(it_2) = Ran \wedge$$
$$\wedge LTC(T, operator <) \wedge \neg const(x) \wedge \neg rev\_it(it_1) \wedge \neg rev\_it(it2)\}$$
$$sort(it_1, it_2);$$
$$\{x =< x_1, x_2, \ldots, x_{i-1}, x'_i, x'_{i+1}, \ldots, x'_{j-1}, x_j, \ldots, x_n > \wedge *it1 = x'_i \wedge *it2 = x_j \wedge$$
$$\wedge \{x'_i, \ldots, x'_{j-1}\} \in perm(x_i, \ldots, x_{j-1}) \wedge \forall k \in [i, j-1] : x'_k < x'_{k+1} \wedge$$
$$\wedge \neg const(x) \wedge \neg rev\_it(it_1) \wedge \neg rev\_it(it2) \wedge cat(it_1) = Ran \wedge cat(it_2) = Ran\}$$

$$\{cat(it_1) = Ran \wedge cat(it_2) = Ran \wedge (*it_1 =? \vee *it_2 =?) \wedge LTC(T, operator <) \wedge$$
$$\wedge \neg const(x) \wedge x =< x_1, x_2, \ldots, x_n >\}$$
$$sort(it_1, it_2);$$
$$\{Undef\}$$

$$\{x =< x_1, x_2, \ldots, x_n > \wedge *it_1 = x_i \wedge *it_2 = x_j \wedge cat(it_1) = For \wedge cat(it_2) = For$$
$$\wedge cat(it_3) = For \wedge LTC(T, operator <)\}$$
$$it_3 = max\_element(it_1, it_2);$$
$$\{*it_3 = x_k \wedge k \in [i, j) \wedge \forall l \in [i, j) : x_l < x_k \wedge cat(it_3) = For \wedge$$
$$\wedge x =< x_1, x_2, \ldots, x_n > \wedge *it_1 = x_i \wedge *it_2 = x_j \wedge cat(it_1) = For \wedge cat(it_2) = For\}$$

$$\{cat(it_1) = For \wedge cat(it_2) = For \wedge cat(it_3) = For \wedge LTC(T, operator <) \wedge$$
$$\wedge (*it_1 =? \vee *it_2 =?)\}$$
$$it_3 = max\_element(it_1, it_2);$$
$$\{Undef\}$$

Irodalomjegyzék

[1] Kozma László-Varga László: A szoftvertechnológia elméleti kérdései, ELTE Eötvös Kiadó, 2003

[2] Scott Meyers: Hatékony C++, Scolar Kiadó, 2003

[3] Bjarne Stroustrup:A C++ programozási nyelv, Kiskapu Kiadó, 2001

[4] Porkoláb Zoltán:Programok Struktúrális Bonyolultsági Mérőszámai, 2002, http://gsd.web.elte.hu/Publications/Metrics/metrika.pdf

[5] Gregorics Tibor:A programozás gondolkodási formái – Egy programozási módszertan, 2004, http://people.inf.elte.hu/gt/prog/konyv/konyv.html

[6] Cormen-Leiserson-Rivest: Algoritmusok, Műszaki Könyvkiadó, 2001

[7] Jeremy Gibbons:Patterns in Datatype-Generic Programming, 2003 http://web.comlab.ox.ac.uk/oucl/work/jeremy.gibbons/publications/patterns.pdf

[8] Stephen C. Dewhurst:C++ hibaelhárító, Kiskapu Kiadó, 2003

[9] Changqing Wang:Integrating tools and methods for rigorous analysis of C++ generic library Components, 1996, http://www.cs.rpi.edu/~musser/Tecton/, ftp://ftp.cs.rpi.edu/pub/adv-prog/Changqing-Wang-thesis.ps.gz

[10] Holger Eichelberger-German Tischler-Jürgen Wolff von Gudenberg: Comprehensive Graphical Description of the STL, 2001 http://www-info2.informatik.uni-wuerzburg.de/staff/eichelberger/reports/stlAbstract.html

[11] David R. Musser: The Tecton Concept Description Language, 1998 http://www.cs.rpi.edu/~musser/Tecton/

[12] David R. Musser: Tecton Description of STL Container and Iterator Concepts, 1998
http://www.cs.rpi.edu/~musser/Tecton/

[13] Deepak Kapur-David R. Musser: Tecton: a framework for specifying and verifying generic system components, 1992
http://www.cs.rpi.edu/~musser/Tecton/

[14] Deepak Kapur - David R. Musser - Xumin Nie: The Tecton Proof System, 1992
http://www.cs.rpi.edu/~musser/Tecton/

[15] David R. Musser - Changqing Wang: A Basis for Formal Specification and Verification of Generic Algorithms in the C++ Standard Template Library, 1995
http://www.cs.rpi.edu/~musser/Tecton/

[16] David R. Musser-Changqing Wang:Dynamic Verification of C++ Generic Algorithms, 1996,
http://www.cs.rpi.edu/~musser/Tecton/

[17] Martin Büchi: Generics in Java and Beyond, 2001
http://www.abo.fi/~mbuechi/download/JavaGenerics.pdf

[18] Pásztorné Varga Katalin - Várterész Magda: A matematikai logika alkalmazásszemléletű tárgyalása, Panem Kiadó, 2003

[19] Bruce Eckel:Thinking in C++ Volume I,II, Prentice Hall, 2000
http://www.BruceEckel.com

[20] Ákos Frohner - Zoltán Porkoláb - Dr. László Varga: Code Generation from UML models, Acta Cybernetica, 2000
http://gsd.web.elte.hu/Temak/cscs.zip

[21] Matthew H. Austern: Generic Programming and the STL, Addison-Wesley,1998

[22] Nyékyné Gaizler Judit szerk.: Az Ada95 programozási nyelv, ELTE Eötvös Kiadó, 1998

[23] Sike Sándor - Varga László: Szoftvertechnológia és UML, ELTE Eötvös Kiadó, 2001

[24] Horváth Zoltán - Pásztorné Varga Katalin - Tejfel Máté: Funkcionális programok helyessége, 2002
http://aszt.inf.elte.hu/~fun_ver/2002/papers/veriffp.ps

[25] Horváth Zoltán: Az ErlVer és a modális μ-kalkulus vizsgálata
http://aszt.inf.elte.hu/~fun_ver/2002/report/helyesseg.ps

[26] Douglas Gregor: STLlint: Static Checking for the C++ Standard Template Library, 2004
http://www.cs.rpi.edu/~gregod/STLlint/STLlint.html

[27] Soczó Zsolt: Generikus programozás C#-ban, 2004
http://www.netacademia.net/tudastar/articlepage.aspx?upid=717

[28] Roland Garcia - Jaako Jarvi - Andrew Lumsdaine - Jeremy Siek - Jeremiah Willcock: A comparative study of language support for generic programming, 2003
http://www.osl.iu.edu/publications/pubs/2003
/comparing_generic_programming03.pdf

[29] Nyékyné Gaizler Judit szerk.: Programozási nyelvek, Kiskapu Kiadó, 2003

[30] Horváth Zoltán: A párhuzamos programozás alapjai, 1994
http://people.inf.elte.hu/hz/parh/jegyzet.ps

[31] Fekete István - Gregorics Tibor - Nagy Sára: Bevezetés a mesterséges intelligenciába, LSI, 1990,1999

[32] Futó Iván szerk.: Mesterséges intelligencia, Aula Kiadó, 1999

[33] The Boost Concept Check Library (BCCL), 2000
http://www.org/libs/concept_check/concept_check.htm

[34] Zólyomi István, Porkoláb Zoltán: Towards a general template introspection library, 2004
http://gsd.web.elte.hu/Publications/GPCE_04/introspection.pdf

[35] Todd L. Veldhuizen: C++ Templates are Turing Complete, 2003
http://osl.iu.edu/~tveldhui/papers/2003/turing.pdf