**Ex. 1**, Cerință: Fie mulțimile A, B, C disjuncte, și complementare. Demonstrați că dacă A, B și C sunt recursiv-enumerabile, atunci acestea sunt și recursive.

Rezolvare: Aproape identică cu aplicația rezolvată 2, doar că acum parcurgem 3 șiruri în loc de 2.

Definim următorul program:

P(x) {

While (true) {

If (Ga() == x) return A;

If (Gb() == x) return B;

If (Gc() == x) return C;

}

}

Programul verifică simultan dacă elementul este în A, B sau C. Din moment ce orice element aparține uneia dintre mulțimi, unul din generatori este garantat să îl returneze după un număr finit de etape.

**Ex. 2**, Cerință: Fie mulțimea A recursivă și mulțimea B nerecursivă dar recursiv-enumerabilă. Este mulțimea *A reunit cu B* recursiv-enumerabilă?

Rezolvare:

Definim următorul program:

P(x) {

If (Pa(x)) return true;

Return Pb(x);

}

Știm că Pa se termină mereu iar Pb se termină măcar pe instanțele *adevărat*, deci și P se termină măcar pe instanțele *adevărat* și returnează corect pentru elementele din *A reunit cu B*. Deci *A reunit cu B* este recursiv-enumerabilă.

**Ex. 3** (bonus), Cerință: De la ex. 2, demonstrați că *A reunit cu B* poate fi și recusivă și nerecursivă.

Să presupunem că B este mulțimea Halt, adică mulțimea perechilor (program, intrare) pentru care programul se oprește pe respectiva intrare. Mulțimea Halt este demonstrabil nerecursivă dar recursiv enumerabilă. (la seminar fiți pregătiți să explicați pe scurt măcar de ce este recursiv enumerabilă). Natura lui *A reunit cu B* va fi decisă de cum alegem A.

Exemplu *A reunit cu B* nerecursivă: alegem pur și simplu pe A mulțimea vidă, a.î. *A reunit cu B* păstrează doar proprietățile lui B.

Exemplu *A reunit cu B* recursivă: alegem A întreg domeniul de definiție al problemei, adică toate perechile (program, intrare), astfel *A reunit cu B = A*, iar A este recursivă.

**Ex. 4**, Cerință: Mulțimea A este recursiv-enumerabilă dar nerecursivă. Ce putem spune despre mulțimea *B = N fără A*

Rezolvare: B nu este nici măcar recursiv enumerabilă. Este situația contradictorie față de exercițiul 2 rezolvat. Presupui că B e recursiv-enumerabilă, urmărești demonstrația de la 2 ca să deduci că și A și B sunt recursive => Contradicție => B nu e nici măcar recursiv enumerabilă.

P(x) {

While (true) {

If (Ga() == x) return A;

If (Gb() == x) return B;

}

}

**Ex. 5**, Cerință: Fie mulțimea A nici măcar recursiv-enumerabilă, și mulțimea B care e submulțime a lui A astfel încât *A fără B* este recursivă. Ce putem spune despre B?

Rezolvare:

Definim următorul program:

Pa(x) {

If (Pa\_fără\_b(x)) return true;

Return Pb(x);

}

Dacă B este recursiv-enumerabilă, Pa se termină pe toate instanțele *adevărat* deci A este recursiv-enumerabilă => Contradicție => B nu e nici măcar recursiv-enumerabilă.

**Ex. 6**, Cerință: Fie mulțimea A recursiv enumerabilă, și mulțimea B = {x + 1 | x aparține A}. Știind că mulțimile A și B sunt disjuncte, demonstrați că dacă *A reunit cu B* este recursivă atunci și A și B sunt recursive.

Rezolvare:

Pasul 1: Demonstrăm că și B este recursiv-enumerabilă. Fie Pb:

Pb(x) {

Return Pa(x-1);

}

Acest program se termină pe toate instanțele *adevărat* => B este recursiv-enumerabilă.

Pasul 2: Definim un program care se termină pe orice intrare și identifică elemente din A și din B.

P(x) {

If (!Pa\_reunit\_b(x)) return false;

While (true) {

If (Ga() == x) return A;

If (Gb() == x) return B;

}

}

Acest program se termină cât timp *A reunit cu B* e recursivă iar A și B sunt măcar recursiv-enumerabile. De aici rezultă că A și B sunt recursive.

**Ex. 7**, Cerință: Fie mulțimea A care nu are nicio pereche de elemente consecutive și mulțimea B = {x + 1 | x aparține A}. Demonstrați că dacă *A reunit cu B* este recursivă atunci și A și B sunt recursive.

Rezolvare: Trebuie să ne imaginăm cum sunt distribuite elementele din A respectiv din B în mulțimea *A reunit cu B*. Astfel putem trage două concluzii:

Dacă x nu aparține lui A dar x + 1 aparține *A reunit cu B* atunci x + 1 nu poate aparține lui B (că atunci x ar fi trebuit să aparțină lui A) deci x + 1 trebuie să aparțină lui A.

Dacă x aparține lui A și x + 1 aparține lui *A reunit cu B* atunci x + 1 nu poate aparține lui A (fiindcă A nu admite membrii consecutivi) deci x + 1 trebuie să aparține lui B.

Folosind aceste două reguli putem deduce inductiv toate elementele din A respectiv din B. Definim următorul program:

P(x) {

Prev = NONE;

For (int i = 0; i <= x; i++) {

If (!Pa\_reunit\_b(i)) prev = NONE;

If (prev == A) prev = B;

If (prev == B) prev = A;

If (prev == NONE) prev = A;

}

Return prev;

}