Loginis reiškinys

Reiškinį nagrinėjame nuo pradžios po vieną simbolį. Atskiriame skaičius, kintamuosius, logines bei lyginimo operacijas. Visus skaičius, kintamuosius bei operacijas įsimename, o pabaigę nagrinėti reiškinį, suskaičiuojame kiek buvo skirtingų skaičių, kintamųjų bei operacijų.

Sprendžiant šį uždavinį reikia atkreipti dėmesį į du dalykus:

1) kintamųjų vardai gali būti užrašyti ir mažosiomis ir didžiosiomis raidėmis: sk1 **or** SK1, todėl verta visas raides reiškinyje pakeisti didžiosiomis;

2) yra vienintelis atvejis, kai dvi operacijos gali būti viena šalia kitos: tai operacija **and** arba **or** ir operacija **not**; operacija **not** gali prasidėti ir pats reiškinys:

```
not a or b
a or not b
```

Beje, reiškinyje gali būti skliaustų, kurie iš esmės nekeičia uždavinio sprendimo. Juos paprasčiausiai galima pakeisti tarpais, pavyzdžiui, jei turime reiškinį:

```
(a <= b) or (cd <> 6) and ((-14) < 13))?
```

tai pašalinę skliaustus, gauname:

```
a <= b or cd <> 6 and -14 < 13?
```

Kiek kebliau yra su ženklais ‘+’ ir ‘-‘, kurie gali eiti tiek prieš neigiamus skaičius, tiek prieš kintamuosius:

```
(-5 < 6) or (-a <= b)
```

Į šiuos ženklus galima iš viso nekreipti dėmesio, išskyrus vienintelį atvejį – kai prieš skaičių parašytas minusas. Tuomet jį reikia įsidėmėti kartu su skaičiumi.

```
const max = 200;
      max_ilg = 20; { maksimalus kintamojo vardo ar skaičiaus ilgis }
type lentelė = array [1..max] of string[max_ilg];
     loginės = (ne, ir, arba);
     lyginimo = (lygu, nelygu, daugiau, mažiau,
                 dauglygu, mažlygu);
     log_aibė = set of loginės; { loginių operacijų aibė }
     lyg_aibė = set of lyginimo; { lyginimo operacijų aibė }

procedure tarp_plius (var s: string);
{ pašalina tarpus ir ženklą '+' iš reiškinio s pradžios }
begin
  while (s[1] = ' ') or (s[1] = '+')do
    delete (s, 1, 1)
end; { tarp_plius }

procedure skaičius (s: string; var ilgis: integer);
{ s – reiškinys, prasidedantis skaičiumi, procedūra randa jo ilgį }
begin
 { ši procedūra sutampa su analogiška 98 užd. procedūra }
end; { skaičius }

procedure kintamasis (s: string; var ilgis: integer);
{ s – reiškinys, prasidedantis kintamuoju, procedūra randa jo ilgį }
{ ši procedūra sutampa su analogiška 98 užd. procedūra }
begin
  ...
end; { kintamasis }

procedure operacija(s: string;  var ilgis: integer;
                    var login: log_aibe; var lygin:
lyg_aibė);
{ s – reiškinys, prasidedantis operacija }
{ procedūra randa pačią operaciją ir jos ilgį }
  var op: lyginimo;
begin
  case s[1] of
    'N': begin                { not }
           ilgis := 3;
           login := login + [ne];
         end;
    'A': begin                { and }
           ilgis := 3;
           login := login + [ir];
```

```
            end;
    'O': begin                  { or }
            ilgis := 2;
            login := login + [arba]
            end;
     else if s[2] in ['>', '=']   { <>, <=, >= }
            then begin
                   ilgis := 2;
                   if s[1] = '>'
                   then op := dauglygu
                   else if (s[1] = '<') and (s[2] = '>')
                           then op := nelygu
                           else op := mažlygu;
                   lygin := lygin + [op];
                 end
            else begin          { +, <, > }
                   ilgis := 1;
                   if s[1] = '='
                   then op := lygu
                   else if s[1] = '<'
                           then op := mažiau
                           else op := daugiau;
                   lygin := lygin + [op];
                 end
  end
end;  { operacija }

procedure kiek_skirt (skaič, kintam: lentelė;
                      login: log_aibė;
                      lygin: lyg_aibė;
                      var sk, kint, lyg, log: integer);
{ randa skirtingų skaičių, kintamųjų bei operacijų kiekį }
  const epsilon = 0.1e-10;  { realieji skaičiai lyginami tokiu tikslumu }
  var ly: lyginimo;
      lo: loginės;
      a, b, kodas, kiek, i, j: integer;
      lygus, real_a, real_b: boolean;
      areal, breal: real;
begin
 { rasime kiek yra skirtingų skaičių }
  kiek := 0;
  i := 1;
  while i <= sk do
    begin
      { įtraukiame i-ąjį skaičių }
      kiek := kiek + 1;
```

```
    j := i + 1;
    while j <= sk do { randame skaičius, sutampančius su i-tuoju }
      begin
        { nustatysime, ar skaičiai yra vienodų tipų }
        real_a := (pos('.', skaič[i]) > 0) or
                  (pos('E', skaič[i]) > 0);
        real_b := (pos('.', skaič[j]) > 0) or
                  (pos('E', skaič[j]) > 0);
        lygus := false;
        if real_a and real_b
        then begin
               val(skaič[i], areal, kodas);
               val(skaič[j], breal, kodas);
               lygus := abs(areal-breal) < epsilon;
             end
        else if not real_a and not real_b
                then begin
                       val(skaič[i], a, kodas);
                       val(skaič[j], b, kodas);
                       lygus := a = b
                     end;
          if lygus { panaikiname j-ąjį skaičių }
          then begin
                 skaič[j] := skaič[sk];
                 sk := sk - 1;
               end
          else j := j + 1
      end;
    i := i + 1
  end;
{ rasime skirtingų kintamųjų skaičių }
kiek := 0;
 for i := 1 to kint do
   if kintam[i] <> ''   { jei šis kintamasis dar nepriskaičiuotas }
      then begin
             kiek := kiek + 1;
             for j := i + 1 to kint do
               if kintam[i] = kintam[j]
                  then kintam[j] := ''
           end;
kint := kiek;
{ rasime skirtingų lyginimo operacijų skaičių }
lyg := 0;
for ly := lygu to mažlygu do
  if ly in lygin
     then lyg := lyg + 1;
```

```
  { rasime skirtingų loginių operacijų skaičių }
  log := 0;
  for lo := ne to arba do
    if lo in login
       then log := log + 1
end;  { kiek_skirt }

procedure skaičiuoti (s: string;  { pradinis reiškinys }
                      var sk, kint, lyg, log: integer);
{ suskaičiuoja kiek reiškinyje yra skaičių, kintamųjų ir operacijų }
  var i, ilgis: integer;
      skaič, kintam: lentelė;
      login: log_aibė;
      lygin: lyg_aibė;
begin
  { visos mažosios raidės reiškinyje keičiamos didžiosiomis }
  for i := 1 to length(s) do
    s[i] := upcase (s[i]);
  { jei tarp ženklo '-' ir skaičiaus yra tarpas (ai), tie tarpai panaikinami, jei po '-' }
 { eina kintamasis, tuomet panaikinamas pats ženklas }
  i := 1;
  while i <= length(s) do
    if s[i] = '-'
       then begin
              while s[i+1] = ' ' do
                delete (s, i+1, 1);
              if not (s[i + 1] in ['0'..'9'])
                 then delete (s, i, 1)
                 else i := i + 1
            end
       else i := i + 1;
    end;
  { skliaustai pakeičiami tarpais }
  for i := 1 to length(s) do
    if s[i] in ['(', ')']
       then s[i] := ' ';
  tarp_plius (s);  { pašalinami tarpai ir ženklas '+' reiškinio priekyje }
  sk := 0;
  kint := 0;
  lygin := [];
  login := [];
  while s <> '?' do   { kol neišnagrinėjome viso reiškinio }
    begin
      { reiškinys prasidės arba skaičiumi, arba kintamojo vardu }
      if s[1] in ['0'..'9', '-']
         then begin
```

```
                skaičius (s, ilgis);
                sk := sk + 1;
                skaič[sk] := copy (s, 1, ilgis);
            end
        else begin
                kintamasis (s, ilgis);
                { reikia patikrinti, ar tai tikrai ne operacija not }
                if copy (s, 1, ilgis) <> 'NOT'
                then begin
                        kint := kint + 1;
                        kintam[kint] := copy (s, 1, ilgis)
                    end
                else ilgis := 0
            end;
        { pašaliname tą skaičių ar kintamąjį }
        delete (s, 1, ilgis);
        tarp_plius (s);
        { jei tai ne reiškinio pabaiga, randame operaciją }
        if s <> '?'
        then begin
                operacija (s, ilgis, login, lygin);
                delete (s, 1, ilgis)
            end;
        tarp_plius (s)
    end;
  kiek_skirt(skaič, kintam, login, lygin, sk, kint, lyg, log)
end;  { skaičiuoti }
```

Aukščiau pateiktas algoritmas yra gan ilgas. Jį galima gerokai sutrumpinti, bet algoritmas taps sunkiau suprantamas. Todėl palikome ilgesnį sprendimą.