

## 8.1. A priedas. *CGIF2XML* proceso programinis kodas

Procesas parašytas Perl programavimo kalba. Programinį kodą sudaro 834 programinio kodo eilutės.

```
#!/perl
#
#
# Paleidimo budas:
#
# c:\>perl pa3.pl parametras
#
# Kur:
# "parametras" -- failo (be prapletimo) pavadinimas
#
# Dabar pagal pagal nurodyta parametra ieskomi du failai:
# "parametras".cgf -- CGIF'as su taisykle
# "parametras".str -- CGIF'as su lenteliu/stulpeliu informacija
#
# Prielaidos:
# - tarpu CGIF'e buti negali !
#
#Algoritmas:
# I) Is "str" failo istraukti lenteliu/stulpeliu informacija
# II) Is "cgf" failo istraukti ir konvertuoti CGIF taisykle
# 1. Visa CGIF'o taisykle sudeti i viena eilute
# 2. Isgraibyti konceptus, kiekvienam issaugot kokiam blokui jis priklauso
# 3. Isgraibyti conteptual relations
# 4. isgraibyti actor'ius
# 5. Su'generuoti tarpine struktura %HASH'a
# III) Sugeneruoti XML

use Data::Dumper;
use XML::Twig;

if ( $#ARGV < 0 ){
    die "Nenurodytas paremtas!"
}

$PrjName = $ARGV[0];

#
# HASH'ai, kuriuose bus saugomi tarpiniai duomenys
#
%Concepts = ();
%ConRelations = ();
%Actors = ();

%COLUMNS = (); #Hash'as, kuriame saugoma informacija apie stulpelius
%TABLES = ();
%RELATIONS = ();

#
# Rezultatines strukturos
#

%Rule = (); #Galutinis HASH'as, is kurio bus genruojamas XML
%Event = (); #Event
%Condition = (); #Condition dalis
%Blocks = (); #Action Block's

&get_table_data();

&get_cgif_data();

&generate_rezult_hash();

&generate_xml();
```

```

#####
#
# SUBROUTINES
#
#####

#####
#
# get_table_data()
#
# Is "failas.str" nuskaitoma informacija - koks laukas priklauso kokiai lentelei
#
sub get_table_data(){

    $/ = undef; #Nustatom i undef, kad per viena karta galima butu nuskaityti visas
failo eilutes

    open(IN,"$PrjName.str") || die "Failo \"$PrjName.str\" atidaryti nepavyko !($!!)\n";

    #
    # Visa faila sudedam i viena eilute
    #

    $sFile = <IN>; #Nuskaitom faila i $sFile

    $sFile =~ s/\s//g; #Ismetam visus tarpus

    $sTemp = $sFile;

    #
    # Nuskaitom "Konceptus"
    #

    while( $sTemp =~ s/\[[^\]]+\](\*\w)?:'([^\]]*)'\]?)?//){

        $sCName = ( $1 ) ? $1 : '';
        $sCalias = ( $2 ) ? $2 : '';

        $sCalias =~ s/\*/g; #Isvalom "*"

        $hTemp{$sCalias} = $sCName;

    }

    #
    # Pagal rysius "ID" surasim lenteliu alias'us
    #
    $sIDs = $sFile;

    while ( $sIDs =~ s/ID?(\\w)?(\\w)\\?)?//){

        if ( exists $hTemp{$1} && exists $hTemp{$2}){

            # $1 - objektas yra lenteles pav. aliasas

            $TABLES{$1} = $hTemp{$1};

        }

    }

    #
    # Surenkam lenteliu stulpelius
    #
    $sAttrs = $sFile;
    while ( $sAttrs =~ s/((\\w+)?(\\w)?(\\w)\\?)?//){

        if ( $1 eq 'attr' || $1 eq 'ID' ){

            #
            # bandysim iskart suformuoti HASH'a, kurio raktai bus stulpelio pavadinimai,
            # o reiksmes lenteliu, kurioms tie stulepliai priklauso, pavadinimai

```



```

        &create_actors_hash();

    close(IN);
}

#####
#
# create_concepts_hash()
#

sub create_concepts_hash(){

    #
    #
    # Graibom konceptus
    #
    $sConcepts = $sFile;

    $sBlock = "Rule";

    while( $sConcepts =~ s/\[([^\[]\[]+?) (\*\w)?:' ([^\[]\[]*) '\]?//){

        #
        # Parenkam bloka
        #
        if ($1 eq "Event") {
            $sBlock = $1;
        }elseif ( $1 eq 'If' ) {
            $sBlock = $1;
        }elseif( $1 eq 'Then'){
            $sBlock = $1;
        }

        $sCtype = ( $1 ne '' ) ? $1 : '';
        $sCalias = ( $2 ne '' ) ? $2 : '';
        $sCval = ( $3 ne '' ) ? $3 : '';

        $sCalias =~ s/\*//g; #Isvalom "*"

        $Concepts{$sCalias}->{'type'} = $sCtype;
        $Concepts{$sCalias}->{'val'} = $sCval;
        $Concepts{$sCalias}->{'block'} = $sBlock;
    }

    #print Dumper(\%Concepts)."\n";
}

#####
#
# create_concepts_relations_hash()
#

sub create_concepts_relations_hash(){

    #
    #
    # Graibom Conceptinius Relationus
    #

    $sConsReIs = $sFile;

    while ( $sConsReIs =~ s/\((\w+)\)? (\w)\? (\w)\)//){

        $ConRelations{"$2$3"}->{'type'} = $1;
        $ConRelations{"$2$3"}->{'obj'} = $2;
        $ConRelations{"$2$3"}->{'subj'} = $3;
    }

    #print Dumper(\%ConRelations);
}

```



```

sub create_event(){

# Ieskom atliekamos Event metu atliekamos operacijas:
# - tarp Concepts iekom kur 'block' yra 'Event'
# - tada CR ieskom su tipu 'obj', kur 'obj' ar 'subj' bus lygus vienam is rastu CA
#

@EventCAs = ();
foreach $sCA (sort keys %Concepts){

    if ( $Concepts{$sCA}->{'block'} eq 'Event' ){
        push @EventCAs, $sCA;
    }

}

$sColNameAlias = '';
$sOperationAlias = '';

LCR: foreach $sCRA (sort keys %ConRelations){

    if ( $ConRelations{$sCRA}->{'type'} eq 'obj' ){

        for ($i = 0; $i <= $#EventCAs; $i++){
            if ($EventCAs[$i] eq $ConRelations{$sCRA}->{'obj'} ||
                $EventCAs[$i] eq $ConRelations{$sCRA}->{'subj'} ){
                #
                # radom bent vieno is Event'o Concept'u alias'a
                # tai ir bus CR'as, kuris nusako Event
                $sColNameAlias = $ConRelations{$sCRA}->{'subj'};
                $sOperationAlias = $ConRelations{$sCRA}->{'obj'};
                last LCR;
            }
        }

    }

}

$operation = $Concepts{$sOperationAlias}{'type'};

if ( uc($operation) eq 'INSERT' ){

    $column = '';
    $mainTable = $Concepts{$sColNameAlias}{'type'};

}elseif ( uc($operation) eq 'UPDATE' ){

    $column = '';
    $mainTable = $COLUMNS{$Concepts{$sColNameAlias}{'type'}};

    if ( exists $COLUMNS{$Concepts{$sColNameAlias}{'type'}} ){

        $column = $Concepts{$sColNameAlias}{'type'};
        $mainTable = $COLUMNS{$Concepts{$sColNameAlias}{'type'}}{'TN'};

    }

}

%Event = ('Operation' => $operation,
          'MainTable' => $mainTable,
          'Column' => $column,
          'Value' => '');

}

#####
#
# create_action_blocks()
#

```

```

sub create_action_blocks(){

#
# Action (Then)
#

# pirma surenkam visu concept'u aliasus, kuriu 'block' = "Then"

@ThenCAs = ();
foreach $sCA (sort keys %Concepts){

    if ( $Concepts{$sCA}->{'block'} eq 'Then' ){
        push @ThenCAs, $sCA;
    }

}

# dabar tarp CR'u ieskom CA (is @ThenCAs), kur CR'o tipas yra 'obj'
foreach $sCRA (sort keys %ConRelations){

    if ( $ConRelations{$sCRA}->{'type'} eq 'obj' ){
        $bFound = 0; #Not Found
        for ($i = 0; $i <= $#ThenCAs; $i++){
            if ($ThenCAs[$i] eq $ConRelations{$sCRA}->{'obj'} ||
                $ThenCAs[$i] eq $ConRelations{$sCRA}->{'subj'} ){
                #
                # radom bent vieno is Then'o Concept'u alias'a
                # tai ir bus CR'as, kuris nusako Block'a

                $sColNameAlias = $ConRelations{$sCRA}->{'subj'};
                $sOperAlias = $ConRelations{$sCRA}->{'obj'};

                $bFound = 1;
                last;
            }
        }
        if ( $bFound){

            $Blocks{$sCRA}->{'Operation'} = $Concepts{$sOperAlias}->{'type'};
            $Blocks{$sCRA}->{'Column'} = $Concepts{$sColNameAlias}->{'type'};

            $value = $Concepts{$sColNameAlias}->{'val'};

            $valueStruct{'Value'} = $value; # $value reiksme
            #
            # bandysim nustatyti $value tipa
            #
            if ( exists ( $COLUMNS{$value} ) ){
                # cia stulpelis...
                $valueStruct{'Type'} = 'Column';
            }elseif ( $value =~ /\^d+$/ ){
                # cia skaicius...
                $valueStruct{'Type'} = 'Number';
            }elseif ( $value =~ /\(\)\$/ ){
                # cia f-ja, nes baigiasi skliaustais...
                $valueStruct{'Type'} = 'Function';
            }elseif ( $value eq '' ){
                #kitaip eilute, bet...
                $valueStruct{'Type'} = 'String';

                #... cia gali buti ir Actor, tai ir reikia patikrinti...

                foreach $a (sort keys %Actors) {

                    if ( $Actors{$a}{'out'} eq $sColNameAlias ){
                        # ar einamojo Actor'iaus 'out' (rezultatas) ir yra Action'o
rezultato stulpelis

```

```

        $$sIn1A = $Actors{$a}{'in1'};
        $$sIn2A = $Actors{$a}{'in2'};
        $$sOutA = $Actors{$a}{'out'};

        $valueStruct{'Type'} = 'Actor';

        $valueActor{'Name'} = $a;

        $valueActor{'Type'} = $Actors{$a}{'type'};

        $valueActor{'In1'}{'type'} = $Concepts{$sIn1A}{'type'};
        $valueActor{'In1'}{'name'} = '';#$Concepts{$sIn1A}->{'name'};
        $valueActor{'In1'}{'val'} = $Concepts{$sIn1A}{'val'};

        $valueActor{'In2'}{'type'} = $Concepts{$sIn2A}{'type'};
        $valueActor{'In2'}{'name'} = '';#$Concepts{$sIn1A}->{'name'};
        $valueActor{'In2'}{'val'} = $Concepts{$sIn2A}{'val'};

        $valueActor{'Out'}{'type'} = $Concepts{$sOutA}{'type'};
        $valueActor{'Out'}{'name'} = '';#$Concepts{$sIn1A}->{'name'};
        $valueActor{'Out'}{'val'} = $Concepts{$sOutA}{'val'};

        $valueStruct{'Value'} = \%valueActor;
    }
}

}

        %{$Blocks{$sCRA}->{'Value'}} = %valueStruct;
    }
}

}

# print Dumper(\@Blocks);
}

#####
#
# create_condition()
#

sub create_condition(){

    #
    # Suformuojam %Condition:
    # 1. Is %Actors reikia istraukti visus elementus, kurie priklauso If daliai:
    # - begam per visus visus %Actors elementus ir pagal concept'u aliasus
    # pasikreipe i%Concepts galim suzinoti kokie concepta'ai yra reikalingi
    #

    foreach $$sAA ( sort keys %Actors ) {

        $$sIn1A = $Actors{$sAA}->{'in1'};
        $$sIn2A = $Actors{$sAA}->{'in2'};
        $$sOutA = $Actors{$sAA}->{'out'};

        if ( $Concepts{$sIn1A}->{'block'} eq 'If' ||
            $Concepts{$sIn2A}->{'block'} eq 'If' ||
            $Concepts{$sOutA}->{'block'} eq 'If' ) {

            $Condition{$sAA}->{'type'} = $Actors{$sAA}->{'type'};
            $Condition{$sAA}->{'In1'}->{'type'} = $Concepts{$sIn1A}->{'type'};
            $Condition{$sAA}->{'In1'}->{'name'} = '';#$Concepts{$sIn1A}->{'name'};
            $Condition{$sAA}->{'In1'}->{'val'} = $Concepts{$sIn1A}->{'val'};
        }
    }
}

```



```

#
#
foreach $sAA (sort keys %{$Rule{'Condition'}} ){

    my $xActor = XML::Twig::Elt->new('Actor');

    $xActor->set_att('Name',$sAA); #Actors' Name/Alias

    $xActor->set_att('Type',$Rule{'Condition'}{$sAA}{'type'}); #Actors' Type

    my $xIn1 = XML::Twig::Elt->new('In1');
    $xIn1->set_att('Type',$Rule{'Condition'}{$sAA}{'In1'}{'type'});
    $xIn1->set_att('Name',$Rule{'Condition'}{$sAA}{'In1'}{'name'});
    $xIn1->set_att('Value',$Rule{'Condition'}{$sAA}{'In1'}{'val'});

    my $xIn2 = XML::Twig::Elt->new('In2');
    $xIn2->set_att('Type',$Rule{'Condition'}{$sAA}{'In2'}{'type'});
    $xIn2->set_att('Name',$Rule{'Condition'}{$sAA}{'In2'}{'name'});
    $xIn2->set_att('Value',$Rule{'Condition'}{$sAA}{'In2'}{'val'});

    my $xOut = XML::Twig::Elt->new('Out');
    $xOut->set_att('Type',$Rule{'Condition'}{$sAA}{'Out'}{'type'});
    $xOut->set_att('Name',$Rule{'Condition'}{$sAA}{'Out'}{'name'});
    $xOut->set_att('Value',$Rule{'Condition'}{$sAA}{'Out'}{'val'});

    $xOut->paste($xActor);

    $xIn2->paste($xActor);

    $xIn1->paste($xActor);

    $xActor->paste($xCondt);
}

#
# Action stuff
#
#
foreach $sAA ( sort keys %{$Rule{'Action'}} ){

    my $xBlock = XML::Twig::Elt->new('Block');

    $xBlock->set_att('Column',$Rule{'Action'}{$sAA}{'Column'});
    $xBlock->set_att('Operation',$Rule{'Action'}{$sAA}{'Operation'});

    $xValue = XML::Twig::Elt->new('Value');
    $xValue->set_att('Type',$Rule{'Action'}{$sAA}{'Value'}{'Type'});

    if ( $Rule{'Action'}{$sAA}{'Value'}{'Type'} eq 'Actor' ){

        $valueStruct = %{$Rule{'Action'}{$sAA}{'Value'}{'Value'}};

        my $xActor = XML::Twig::Elt->new('Actor');

        $xActor->set_att('Name',$valueStruct{'Name'}); #Actors' Name/Alias

        $xActor->set_att('Type',$valueStruct{'Type'}); #Actors' Type

        my $xIn1 = XML::Twig::Elt->new('In1');
        $xIn1->set_att('Type',$valueStruct{'In1'}{'type'});
        $xIn1->set_att('Name',$valueStruct{'In1'}{'name'});
        $xIn1->set_att('Value',$valueStruct{'In1'}{'val'});

        my $xIn2 = XML::Twig::Elt->new('In2');
        $xIn2->set_att('Type',$valueStruct{'In2'}{'type'});
        $xIn2->set_att('Name',$valueStruct{'In2'}{'name'});
        $xIn2->set_att('Value',$valueStruct{'In2'}{'val'});

        my $xOut = XML::Twig::Elt->new('Out');

```

```

    $xOut->set_att('Type', $valueStruct{'Out'}{'type'});
    $xOut->set_att('Name', $valueStruct{'Out'}{'name'});
    $xOut->set_att('Value', $valueStruct{'Out'}{'val'});

    $xOut->paste($xActor);

    $xIn2->paste($xActor);

    $xIn1->paste($xActor);

    $xActor->paste($xValue);

} else {

    $xValue->set_att('Value', $Rule{'Action'}{$sAA}{'Value'}{'Value'});

}

$xValue->paste($xBlock);

#foreach $sBA ( sort keys %{$Rule{'Action'}{$sAA}} ) {
#
#     $xBlock->set_att($sBA, $Rule{'Action'}{$sAA}{'$sBA'});
#
#}

$xBlock->paste($xAction);
}

$xAction->paste($xRule);

$xCondt->paste($xRule);

$xEvent->paste($xRule);

$xRule->paste($xCGIF);

#
# Sutvarkom relations
#
$xRelations = XML::Twig::Elt->new('Relations');

foreach $r ( sort keys %RELATIONS ) {

    $xRelation = XML::Twig::Elt->new('Relation');
    $xRelation->set_att('Name', $r);
    $xRelation->set_att('Father', $RELATIONS{$r}{'F'});
    $xRelation->set_att('Child', $RELATIONS{$r}{'C'});

    $xRelation->paste($xRelations);
}

$xRelations->paste($xCGIF);
#

#
# Sutvarkom Tables
#

$xTables = XML::Twig::Elt->new('Tables');

$sTable = ''; # "Einamoji" lentele

#
# %Columns isrusiuojam pagal reiksmes (t.y. lenteliu pavadinimus),
# mazejimo tvarka, kad XML'e butu atvirksčiai

foreach $sColumn ( sort { $COLUMNS{$b}{'TN'} cmp $COLUMNS{$a}{'TN'} } keys %COLUMNS
) {

```

```

if ( $$sTable ne $COLUMNS{$sColumn}{'TN'} ){
  #Jeigu pasikeite lenteles pav...
  if ( $$sTable ne '' ){
    #ir jeigu tai nera pirma Table (pirma butu jeigu $$sTable butu lygi ''),
    #tai cia vadinasi - "Table" pabaiga, tad ja reikia prijungti
    # prie "Tables"
    $xTable->paste($xTables);

  }

  $xTable = XML::Twig::Elt->new('Table');
  $xTable->set_att('Name',$COLUMNS{$sColumn}{'TN'});

  $$sTable = $COLUMNS{$sColumn}{'TN'};
}
#
# Sukuriam ir prie Table pridedam Column
#
$xColumn = XML::Twig::Elt->new('Column');
$xColumn->set_att('Name',$sColumn);
$xColumn->set_att('Type',$COLUMNS{$sColumn}{'CT'});
$xColumn->paste($xTable);

}
#
# jeigu lenteliu buvo ($sTable nelygus ''), tai paskutine "Table" prie "Tables"
# prideta nebuvo,
# nes foreach ciklas pasibaige...tad ja pridedam cia
#
if ( $$sTable ne '' ){
  $xTable->paste($xTables);
}

$xTables->paste($xCGIF);

$xCGIF->print;
}

```

## 8.2. B priedas. XML2SQL proceso programinis kodas

Procesas parašytas Perl programavimo kalba. Programinį kodą sudaro 573 programinio kodo eilutės.

```
#!/perl
#
#
# Paleidimo budas:
#
# c:\>perl pa3.pl parametras

use Data::Dumper;
use XML::Simple;

%ACTORTYPE = (
    'AND' => ' AND ',
    'OR'  => ' OR ',

    'EQUAL' => ' = ',
    'NOT EQUAL' => ' <> ',
    'GREATERTHAN' => ' > ',
    'LESSTHAN' => ' < ',
    'GREATEREQUAL' => ' >= ',
    'LESSEQUAL' => ' <= ',

    'SUM' => ' + ',
    'SUB' => ' - ',
    'MULTIPLY' => ' * ',
    'DIVIDE' => ' / '
);

if ( $#ARGV < 0 ){
    die "Nenurodytas paremtas!"
}

$sFileName = $ARGV[0];

my $hXML = XMLin($sFileName);

#print Dumper($hXML);

%COLUMNSTABLES = ();
%RELATIONS = ();

&create_columnstables_hash();

&create_relations_hash();

$RULENAME = $hXML->{'Rule'}->{'Name'};
$MAINTABLE = $hXML->{'Rule'}->{'Event'}->{'MainTable'};
$OPERATION = $hXML->{'Rule'}->{'Event'}->{'Operation'};
$COLUMN = $hXML->{'Rule'}->{'Event'}->{'Column'};

if ( uc($OPERATION) eq 'UPDATE' && $COLUMN ne '' ) {

    $EXTRA .= "\nIF Update($COLUMN) ";

}

&parse_actors();
```

```

foreach $sA ( sort keys %HACTORS ){

    $sExpr = $HACTORS{$sA}{EXP};
    $sReplStr = "<%$sA%>";

    foreach $sAA ( sort keys %HACTORS ){

        $HACTORS{$sAA}{EXP} =~ s/$sReplStr/$sExpr/x;

    }

}

$sActorRoot = &find_actor_root();
$sIfPart = $HACTORS{$sActorRoot}{EXP};
$sActionBlock = &create_action_block();

print qq(

CREATE trigger $RULENAME ON $MAINTABLE
FOR $OPERATION
AS $EXTRA
BEGIN
    IF ( $sIfPart )
        BEGIN
            $sActionBlock
        END
END

);

#
# Create structure
#
sub create_columnstables_hash(){
    #
    # Jeigu <Tables><Table> yra tik viena, tai XMLin masyvo nepagamina
    #
    my @TABLES = ();

    my %hTables = %{$hXML->{'Tables'}};

    if ( &what_is_it( $hTables{'Table'} ) eq 'ARRAY' ){

        @TABLES = @{$hTables{'Table'}};

    }else{

        $TABLES[0] = $hTables{'Table'};

    }

    #
    # Suformuojam hash'a
    #
    for ($i=0; $i<=#TABLES; $i++){

        my %hTable = %{$TABLES[$i]};

        my $sTableName = $hTable{'Name'};
        my @aCols = (); #Einamosios lenteles stulpeliai

        if ( &what_is_it( $hTable{'Column'} ) eq 'ARRAY' ){

            @aCols = @{$hTable{'Column'}};

```

```

    }else{
        $aCols[0] = $hTable{'Column'};
    }

    for (my $j=0; $j<=$#aCols; $j++){
        $COLUMNSTABLES{$aCols[$j]{'Name'}}{'TN'} = $sTableName;
        $COLUMNSTABLES{$aCols[$j]{'Name'}}{'CT'} = $aCols[$j]{'Type'};
    }
}
#
#
#
#
# Create structure
#
sub create_relations_hash(){
    #
    # Jeigu <Relations><Relation> yra tik viena, tai XMLin masyvo nepagamina
    #
    my @RELATIONS = ();

    my %hRelations = %{$hXML->{'Relations'}};

    if ( &what_is_it( $hRelations{'Relation'} ) eq 'ARRAY' ){
        @RELATIONS = @{$hRelations{'Relation'}};
    }else{
        $RELATIONS[0] = $hRelations{'Relation'};
    }

    #
    # Suformuojam hash'a
    #
    for ( my $i=0; $i<=$#RELATIONS; $i++){
        my %hRelation = %{$RELATIONS[$i]};

        my $relationName = $hRelation{'Name'};

        $RELATIONS{$relationName}{'F'} = $hRelation{'Father'};
        $RELATIONS{$relationName}{'C'} = $hRelation{'Child'};
    }
}
#
#
#
sub create_action_block(){
    #
    # Action Block
    #
    my $sActionBlock = '';

    my %hAction = %{$hXML->{'Rule'}->{'Action'}};

    my @aBlocks = ();

    if ( &what_is_it( $hAction{'Block'} ) eq 'ARRAY' ){
        @aBlocks = @{$hAction{'Block'}};
    }else{

```

```

        $aBlocks[0] = $hAction{'Block'};
    }
    for ( my $i = 0; $i <= $#aBlocks ; $i++ ){
        if ( uc($aBlocks[$i]{'Operation'}) eq 'UPDATE'){
            #
            #
            #
            my $sActionColumn = $aBlocks[$i]{'Column'};
            my %actionValueHash = %{$aBlocks[$i]{'Value'}};
            my $sActionValue = '';
            if ( $actionValueHash{'Type'} eq 'Number' ||
                $actionValueHash{'Type'} eq 'String' ){
                $sActionValue = "".$actionValueHash{'Value'}."";
            }elseif ( $actionValueHash{'Type'} eq 'Function' ){
                $sActionValue = $actionValueHash{'Value'};
            }elseif ( $actionValueHash{'Type'} eq 'Column' ){
                $sActionValue = " (
".&get_subselect_to_get_column_value($actionValueHash{'Value'}) ." ) ";
            }elseif ( $actionValueHash{'Type'} eq 'Actor' ){
                $sActionValue = " ( " . &get_actor_exp( \%{$actionValueHash{'Actor'}} ) .
" ) ";
            }
            }
            my $sActionColumnTable = $COLUMNSTABLES{$sActionColumn}{'TN'};
            my $idColumn = &get_id_column($sActionColumnTable);
            my $where = &get_subselect_to_get_record($sActionColumnTable);
            $sActionBlock .= qq(
UPDATE $sActionColumnTable SET $sActionColumn = $sActionValue
WHERE $idColumn = ( $where )
);
        }
    }
    return $sActionBlock;
}
#
#
#
sub parse_actors(){
    # Condition ref
    my %hCondition = %{$hXML->{'Rule'}->{'Condition'}};

    # Actors array
    my @Actors = ();

    if ( &what_is_it( $hCondition{'Actor'} ) eq 'ARRAY' ){

```

```

        @Actors = @{$hCondition{'Actor'}};
    }else{
        $Actors[0] = $hCondition{'Actor'};
    }

    #
    # Begsim per kiekviena aktoriu ir sukursim jo israiska
    #
    for ( my $i = 0; $i <= $#Actors; $i++ ){

        my $sActorType = $Actors[$i]->{'Type'};
        my $sActorName = $Actors[$i]->{'Name'};

        # 1-o operando israiska
        #my $sIn1Exp = &return_expression( $Actors[$i]->{'In1'} );

        # 2-o operando israiska
        #my $sIn2Exp = &return_expression( $Actors[$i]->{'In2'} );

        # Actor'iaus israiska
        #my $sActorExp = " $sIn1Exp" . $ACTORTYPE{uc($sActorType)} . "$sIn2Exp ";

        #print "$sActorName --> $sActorExp\n";

        #Galutinis Actor'iu hash'as, kolkas dabar su kintamaisiais (t.y. ne galutines
        israiskos)

        $HACTORS{$sActorName}{IN1} = $Actors[$i]->{'In1'}->{'Name'};
        $HACTORS{$sActorName}{IN2} = $Actors[$i]->{'In2'}->{'Name'};
        $HACTORS{$sActorName}{EXP} = &get_actor_exp( \%{$Actors[$i]} ); # $sActorExp;

    }

}

#
#
#
#

sub return_expression(){
    my ($sOp) = @_;

    my $sOpType = $sOp->{'Type'};
    my $sOpName = $sOp->{'Name'};
    my $sOpVal = $sOp->{'Value'};

    #
    # Jeigu nurodytas operando vardas tai cia tures buti israiska,
    # tad kolkas cia idedam kintamaji kuri reiks keisti
    #
    if ( $sOpName ne '' ){

        return " ( <%%$sOpName%> ) ";

    }

    if ( uc($sOpType) eq 'TEXT' ){

        return " '$sOpVal' ";

    }elseif ( uc($sOpType) eq 'NUMBER' ){

        return " $sOpVal ";

    }else{

        return " ( ".$&get_subselect_to_get_column_value($sOpType). " ) ";

    }

}

```

```

    }
}

#
# HACTORS hash'e ieskosim pacio pirmojo (sakninio) aktoriaus -
# galutines IF dalies israiskos
#
# Algoritmas:
# 1. begam per visus hash elementus...
# 1.1 dar karta begam per visus to paties hash'o elementus
# 1.1.1 jeigu einamojo elemento "raktas"
#
sub find_actor_root(){
    foreach my $sA ( sort keys %HACTORS){
        my $bThisOne = 1; #Darom prielaida, kad sis yra sakninis
        foreach my $sAA ( sort keys %HACTORS){
            next if ( $sA eq $sAA );
            if ( $sA eq $HACTORS{$sAA}{'IN1'} || $sA eq $HACTORS{$sAA}{'IN2'} ) {
                $bThisOne = 0; #Kadangi jis yra panaudotas kaip In1 ar In2, tai tada jis
ne sakninis!
                last;
            }
        }
        if ( $bThisOne ){
            return $sA; #Radom !
        }
    }
}

#
#
#
sub get_actor_exp(){
    my ( $actorHashRef ) = @_;
    %actorHash = %$actorHashRef;
    my $sActorType = $actorHash{'Type'};
    # 1-o operando israiska
    my $sIn1Exp = &return_expression( $actorHash{'In1'} );
    # 2-o operando israiska
    my $sIn2Exp = &return_expression( $actorHash{'In2'} );
    return " $sIn1Exp " . $ACTORTYPE{uc($sActorType)} . "$sIn2Exp ";
}

#
# Sugeneruoja subselect'a, kag galima butu gauti
# $table irasa, pagal $MAINTABLE (t.y. paprastai tai bus naudojama WHERE dalyje)
sub get_subselect_to_get_record(){
    my ( $table ) = @_;

```

```

my $subSelect = '';
if ( $table eq $MAINTABLE ) {
    my $idColumn = &get_id_column($table);
    return " SELECT $idColumn FROM inserted ";
};
my @fathers = &find_fathers($MAINTABLE,$table);
my $fatherTable = pop @fathers;
my $i = 0;
foreach my $t ( reverse @fathers ){
    $i++;
    if ( $t ne $table ) {
        my $idColumn = &get_id_column($t);
        $subSelect .= " SELECT ".$fatherTable."ID FROM $t WHERE $idColumn = ( ";
        if ( $t eq $MAINTABLE ){
            my $idColumn = &get_id_column($t);
            $subSelect .= " SELECT $idColumn FROM inserted " ;
            last;
        }
    }
    $fatherTable = $t;
}
for ( my $j = 0; $j < $i; $j++ ){
    $subSelect .= " )";
}
return $subSelect;
}
#
# Sugeneruoja subselect'a, kad paimti nurodyta stulpeli.
#
sub get_subselect_to_get_column_value(){
    my ( $column ) = @_;
    my $columnTable = $COLUMNSTABLES{$column}{'TN'};
    if ( $columnTable eq $MAINTABLE ){
        return " SELECT $column FROM inserted ";
    }
    my $idColumn = &get_id_column( $columnTable );
    my $where = &get_subselect_to_get_record($columnTable);

```

```

        return " SELECT $column FROM $columnTable WHERE $idColumn = ( $where ) ";
    }
    #
    # Pagalbinis SUB'as rysiu masyvo, tarp dviejų lentelių, gavimui
    #
    sub find_fathers(){
        my ($child,$father) = @_;

        my @tablesArray = ();

        foreach my $t (sort keys %RELATIONS){

            if ( $RELATIONS{$t}{C} eq $child && $RELATIONS{$t}{F} eq $father ){

                push @tablesArray,$child,$father;

                return @tablesArray;

            }elseif( $RELATIONS{$t}{C} eq $child ){

                my @fathers = &find_fathers( $RELATIONS{$t}{F}, $father);

                if ( $#fathers > -1 ) {

                    push @tablesArray, $child, @fathers;
                    return @tablesArray;

                }

            }

        }

        return @tablesArray;
    }

    #
    # Grazina nurodytos lenteles ID stulpeli
    #
    sub get_id_column(){

        my ( $table ) = @_;

        foreach my $c ( sort keys %COLUMNSTABLES ){

            if ( $COLUMNSTABLES{$c}{'TN'} eq $table &&
                $COLUMNSTABLES{$c}{'CT'} eq 'ID' ){

                return $c;

            }

        }

        return undef;
    }

    #
    # Pasako kox kintamojo tipas (is Dumper.pm)
    #
    sub what_is_it(){

        my $ref = ref $_[0];
        my $rval = $ref ? $_[0] : \"$_[0]";

        #print overload::StrVal($rval);

        if (overload::StrVal($rval) =~ /^(?:([^\=]+)=)?([A-Z]+\ (0x(?:[^\ ]+))\ )$/ ) {
            # $class = $1;

```

```
    $type = $2;  
    # $id = $3;  
  }  
  return $type;  
}
```