תרגום SMV ל-CDL

תרגום של השמה בודדת מ-SMV ל-CDL יתבצע באופן הבא:

SMV:

```
next(x) :=
        case
                Cond_X1 : Target_X1;
                Cond_X2 : Target_X2;
                :
                :
                Cond_Xn : Target_Xn;
        esac;
```

CDL:

```
Transition T_x1:
        enable: Cond_X1;
        transition: x'=Target_X1;

Transition T_x2:
        enable: !Cond_X1/\ Cond_X2;
        transition: x'=Target_X2;
:
:
Transition T_xn:
        enable: !Cond_X1/\ !Cond_X2/\ … /\ Cond_Xn;
        transition: x'=Target_Xn;
```

יש לשים לב לכך שב-CDL לא מוגדר סדר בין ההשמות כפי שמוגדר ב-SMV (התנאי הראשון שמתקיים יתבצע) לפיכך יש צורך ליצור סדר ביניהם על ידי שלילת יתר התנאים הקודמים.

השוני המהותי בין מודל המתואר ב-SMV למודל המתואר ב-CDL בא לידי ביטוי כאשר רוצים לבטא השמות של מספר משתנים.
ב- SMV מתאימים למשתנה מסוים השמות למצבים שונים.
ב- CDL מתאימים למצב מסוים השמות למשתנים שונים.

למשל:

```
next(x) := case
                t=1 : 5;
                u>5 : 10;
                1 : x+1;
            esac;


Transition tr_1 :
        Enable: u>5;
        Assign: x'=x+1 /\ u`=5 /\ t'=0;
```

לכן, כאשר רוצים לתאר מודל SMV ב-CDL יש צורך לרכז את כל ההשמות המתאימות לתנאי מסוים, כך שכולם יתבצעו בצורה מסונכרנת.

עבור ההשמות הבאות במודול כלשהו:

```
Module main()
Var
        x : TypeX ;
        y : TypeY ;

Assign
next(x) :=
        case
                Cond_x1 : Target_x1;
                Cond_x2 : Target_x2;
                :
                :
                Cond_xn : Target_xn;
        esac;

next(y) :=
        case
                Cond_y1 : Target_y1;
                Cond_y2 : Target_y2;
                :
                :
                Cond_ym : Target_ym;
        esac;
```

קיימות שתי אפשרויות לביצוע התרגום

האפשרות הראשונה – מכפלה ווקטורית של התנאים:

```
Module Main()
{

Transition T_x1_y1:
        enable: Cond_x1/\ Cond_y1;
        transition: x'=Target_x1/\y'=Target_y1;

Transition T_x2_y1:
        enable: !Cond_x1/\Cond_x2/\Cond_y1;
        transition: x'=Target_x2/\y'=Target_y1;
:
:
Transition T_xn_y1:
        enable: !Cond_x1/\!Cond_x2/\ … /\Cond_xn/\Cond_y1;
        transition: x'=Target_xn/\ y'=Target_y1;

Transition T_x1_y2:
        enable: Cond_x1/\ !Cond_y1/\Cond_y2;
        transition: x'=Target_x1/\y'=Target_y2;

Transition T_x2_y2:
        enable: !Cond_x1/\Cond_x2/\!Cond_y1/\Cond_y2;
```

```
        transition: x'=Target_{x2}/\y'=Target_{y2};
:
:
Transition T_xn_y2:
        enable: !Cond_{x1}/\!Cond_{x2}/\ … /\Cond_{xn}/\!Cond_{y1}/\Cond_{y2};
        transition: x'=Target_{xn}/\ y'=Target_{y2};
:
:
Transition T_xn_ym:
        enable: !Cond_{x1}/\!Cond_{x2}/\ … /\Cond_{xn}/\!Cond_{y1}/\!Cond_{y1}/\…/\Cond_{ym};
        transition: x'=Target_{xn}/\ y'=Target_{ym};
}
```

כפי שניתן לראות, אפשרות זו כרוכה בכמות עצומה של שורות.
למשל עבור מודל שבו 10 משתנים ולכל משתנה 4 תנאים נקבל 4^{10} תנאים כלומר כ-3 מיליון שורות קוד !

האפשרות השנייה (האפשרות בה נשתמש):
נייצר שתי רשימות מעברים בלתי תלויות אחת בשניה – אחת לכל משתנה (כפי שתואר בהתחלה), כל אחת מהן נמקם במודול משלה ונסכרן את המודולים סנכרון מלא.
כלומר:

```
Module v_X(x : TypeX, …)
{
        Transition T_x1:
                enable: Cond_{x1};
                transition: x'=Target_{x1};

        Transition T_x2:
                enable: !Cond_{x1}/\ Cond_{x2};
                transition: x'=Target_{x2};
        :
        :
        Transition T_xn:
                enable: !Cond_{x1}/\ !Cond_{x2}/\ … /\ Cond_{xn};
                transition: x'=Target_{xn};
}

Module v_Y(y:  TypeY, …)
{
Transition T_y1:
        enable: Cond_{y1};
        transition: y'=Target_{y1};

Transition T_y2:
        enable: !Cond_{y1}/\ Cond_{y2};
        transition: y'=Target_{y2};
:
:
Transition T_ym:
```

```
        enable: !Cond_y1/\ !Cond_y2/\ … /\ Cond_yn;
        transition: y'=Target_yn;
}

Module Main()
{
        y : TypeA;
        x : TypeX;

        (v_Y(y) || v_X(x));
}
```

במקרה זה כפי שניתן לראות, קיימים n+m מעברים.
הפעם עבור מודל שבו 10 משתנים ולכל משתנה 4 תנאים נקבל 4*10 תנאים כלומר כ-120 שורות קוד.