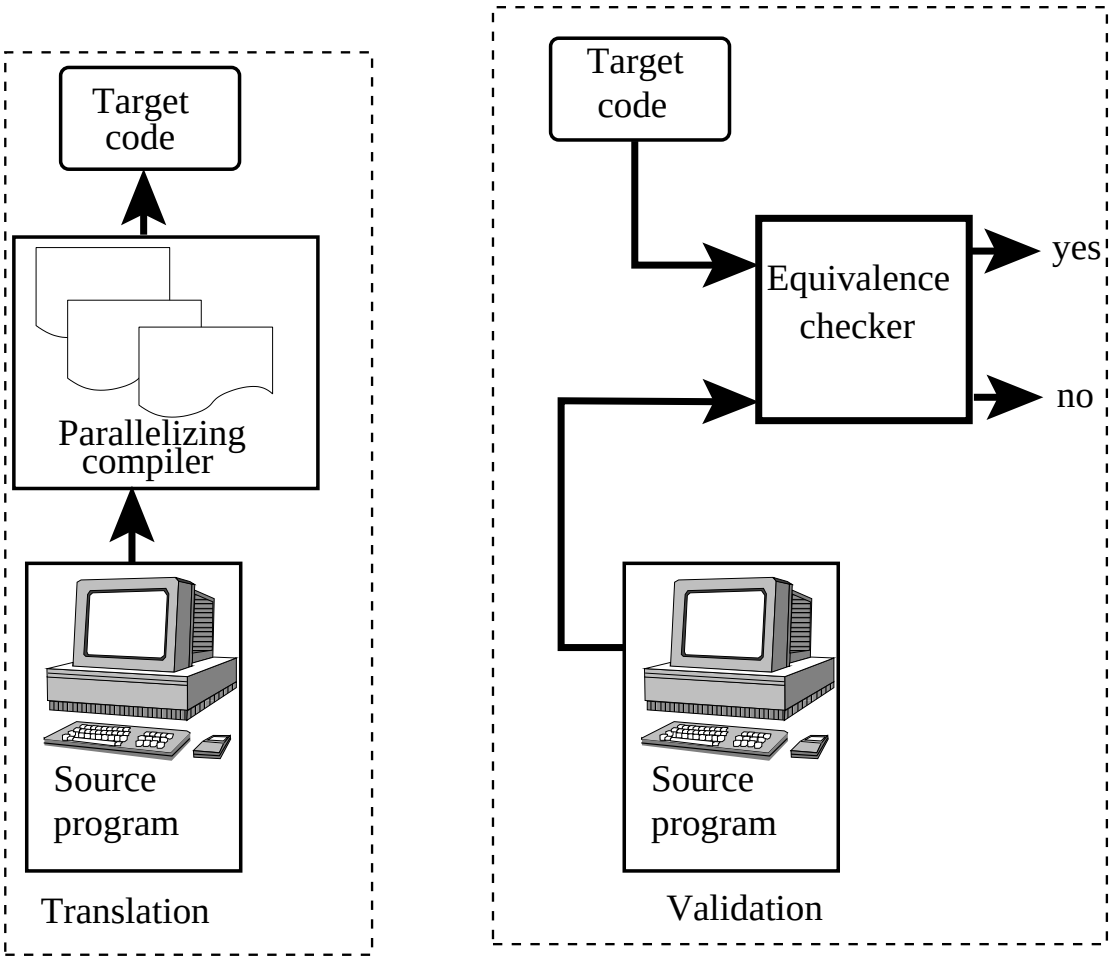
Translation Validation of Loop involving Code Optimizing Transformations using Petri Net based Models of Programs (Work in Progress)

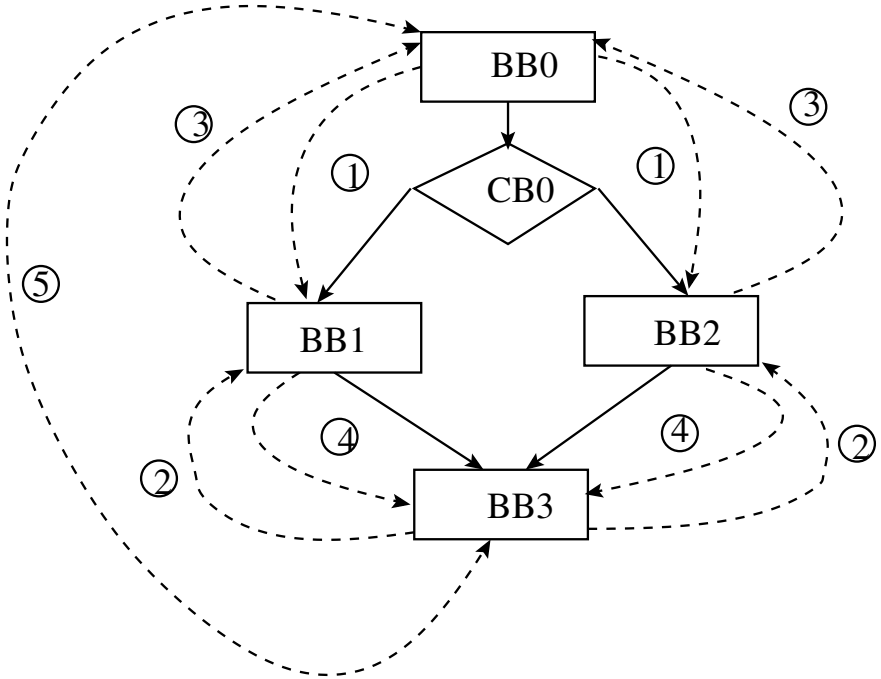
Soumyadip Bandyopadhyay, Rakshit Mittal, Rochishnu Banerjee

BITS Pilani K K Birla Goa Campus, India
Telecom Paris, France

Overview


Target code
Parallelizing compiler
Source program
Translation
Target code
Equivalence checker
yes
no
Source program
Validation

Related work



- (1) Duplicating Down, (2) Duplicating up, (3) Boosting up, (4) Boosting down, (5) Useful move. [1]

[1] ACM Computing Survey 1990; Becon et. al.

Related work

- **Bisimulation-Based Methods**
  - First proposed by Amir Pnueli [TACAS 1998]
  - Enhanced by Necula et. al. [PLDI-2000] and Rinard et. al. [Technical Report MIT 2000]
  - Modified by Kundu et. al. [CAV 2008]
- **Inductive Inference based Methods**
  - Matthias et. al. [ASE-2014]
- **Path based Methods**
  - Karfa et. al. verify transformations carried out by SPARK; where the control structure of program is altered [TCAD-2012]
  - Modified by Banerjee et. al. [TCAD-2014]
  - Further modified by Chouksey et. al. [TCAD 2019]
- **Coq–Prove the correctness**

Related work

```
int i = 1, j = 1;
int k;
while ( i*7 <=100)
  {
    i++  ;
  }
while ((j+1)*11 <=100)
  {
   j++;
  }
 k = i+j;
```

(a)

```
int i = 1, j = 1;
int k;
while ((j+1)*11 <=100)
  {
    j++;
  }
while ( i*7 <=100)
  {
   i++  ;
  }
k = i+j;
```

(b)

```
int i = 1, j ;
int k;
j = i;
#parbegin
   while ( i*7 <=100)
   {
        i++  ;
    }
||
    while ( (j+1)*11 <=100)
    {
      j++;
    }
#parend
 k = i+j;
```

(c)

- Loop swapping
- Thread level parallelizing transformations [2]

[2] ACM Computing Survey

Observation

- Bandyopadhyay et. al [ATVA 2017, Acta informatica 2019] handles the above transformations.
- Major limitations and disadvantages
  - Huge Model size
  - Correctness of Model Constructor is not proved
  - Cannot handle loop involving Code Optimizing Transformations

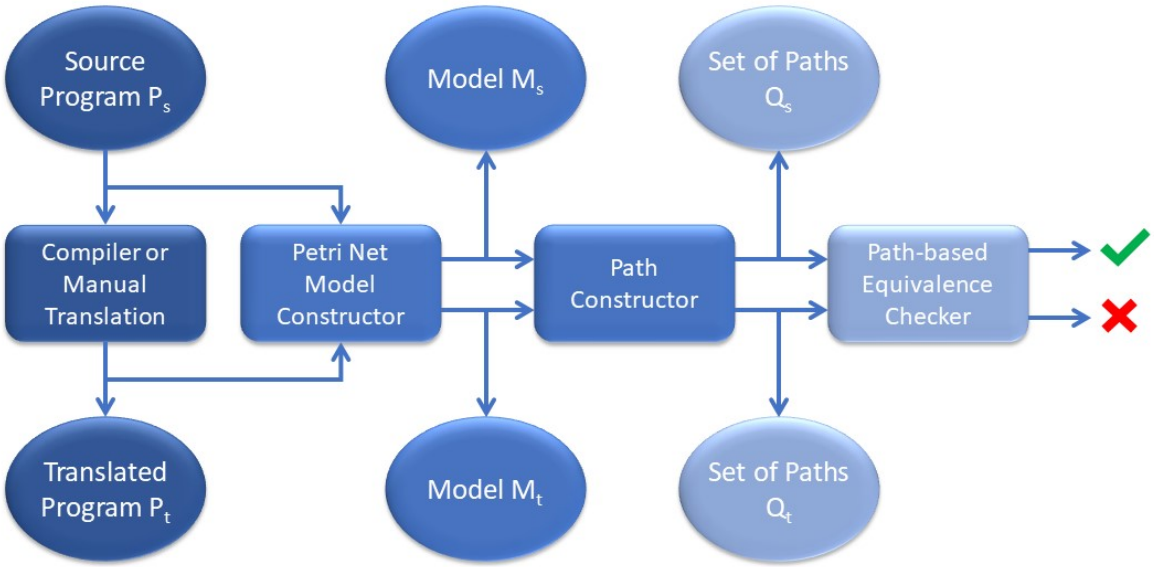
Observation

- Bandyopadhyay et. al [ATVA 2017, Acta informatica 2019] handles the above transformations.
- Major limitations and disadvantages
  - Huge Model size
  - Correctness of Model Constructor is not proved
  - Cannot handle loop involving Code Optimizing Transformations

Observation

- Bandyopadhyay et. al [ATVA 2017, Acta informatica 2019] handles the above transformations.
- Major limitations and disadvantages
  - Huge Model size
  - Correctness of Model Constructor is not proved
  - Cannot handle loop involving Code Optimizing Transformations

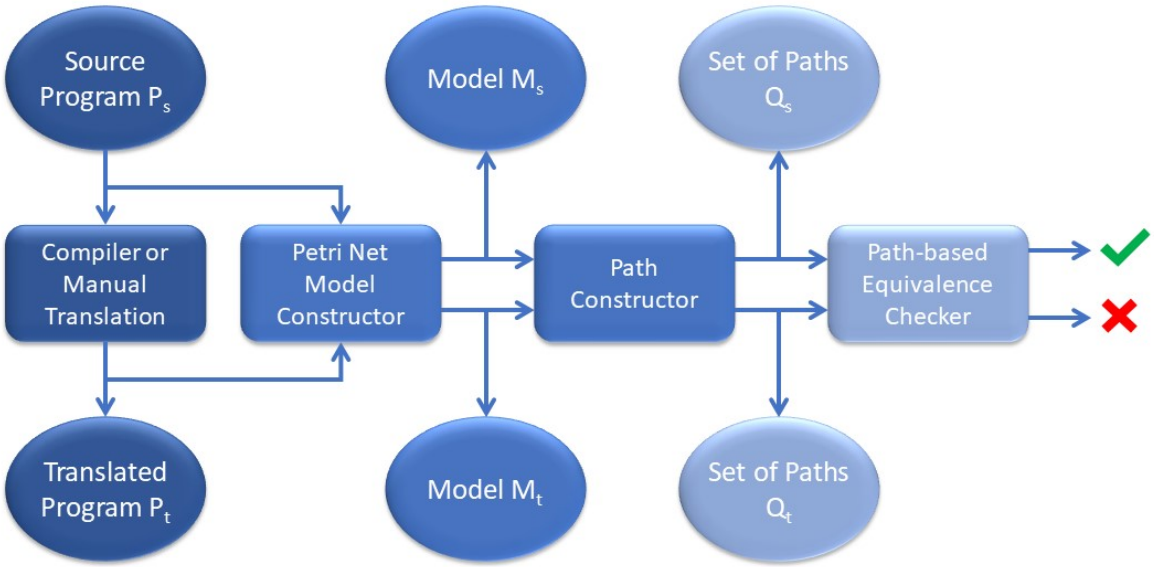
Problem definition



Tool: Prototype

- https://github.com/raks0009/AOFM/blob/master/SamaTulyata.zip
- SimPres https://www.ida.liu.se/labs/eslab/publications/pap/db/NC99.pdf

Problem definition



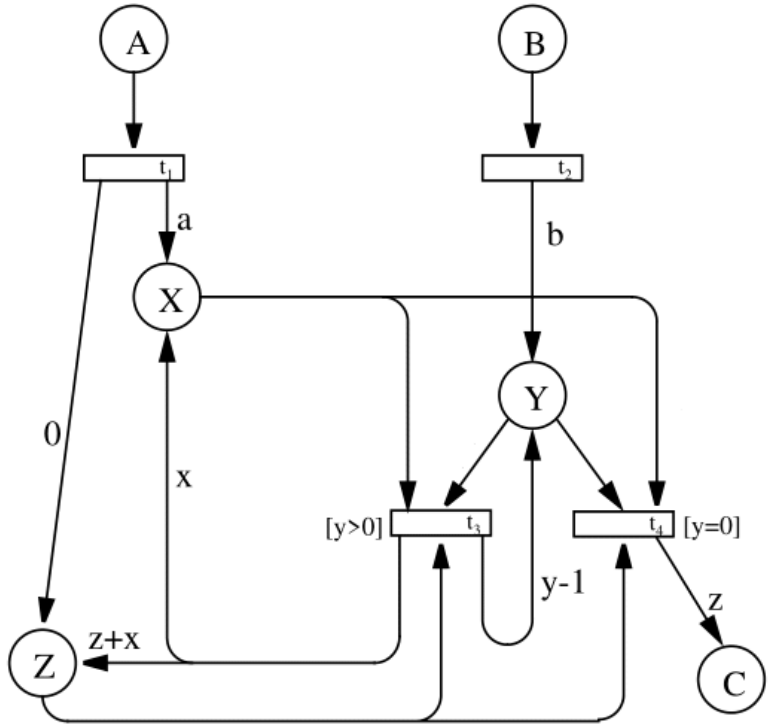
Tool: Prototype

- https://github.com/raks0009/AOFM/blob/master/SamaTulyata.zip
- SimPres
  https://www.ida.liu.se/labs/eslab/publications/pap/db/NC99.pdf

Model Construction

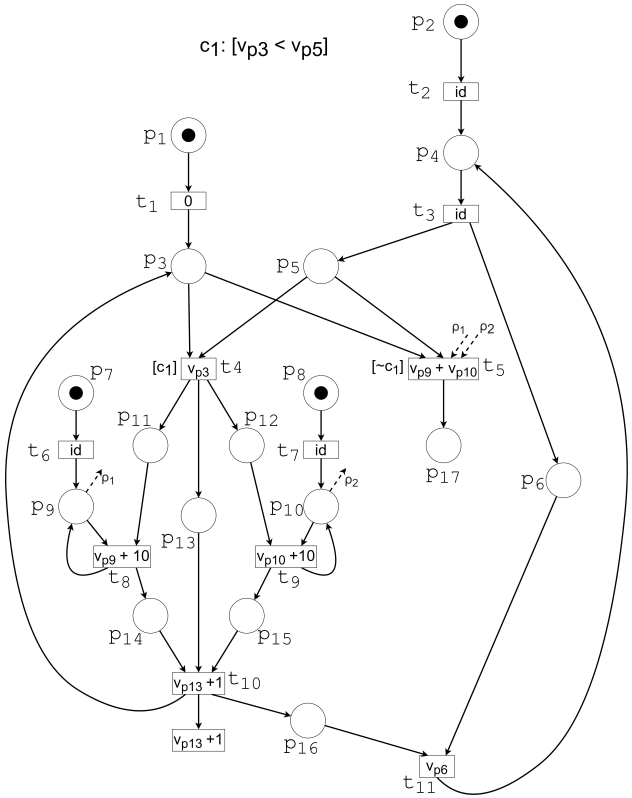
```
int mult(int a,int b)
{
    int x,y,z;
    x=a;
    y=b;
    z=0;
    while (y>0) {
        z=z+x;
        y=y-1;
    }
    return z;
}


c=mult(a,b);
```

(a)

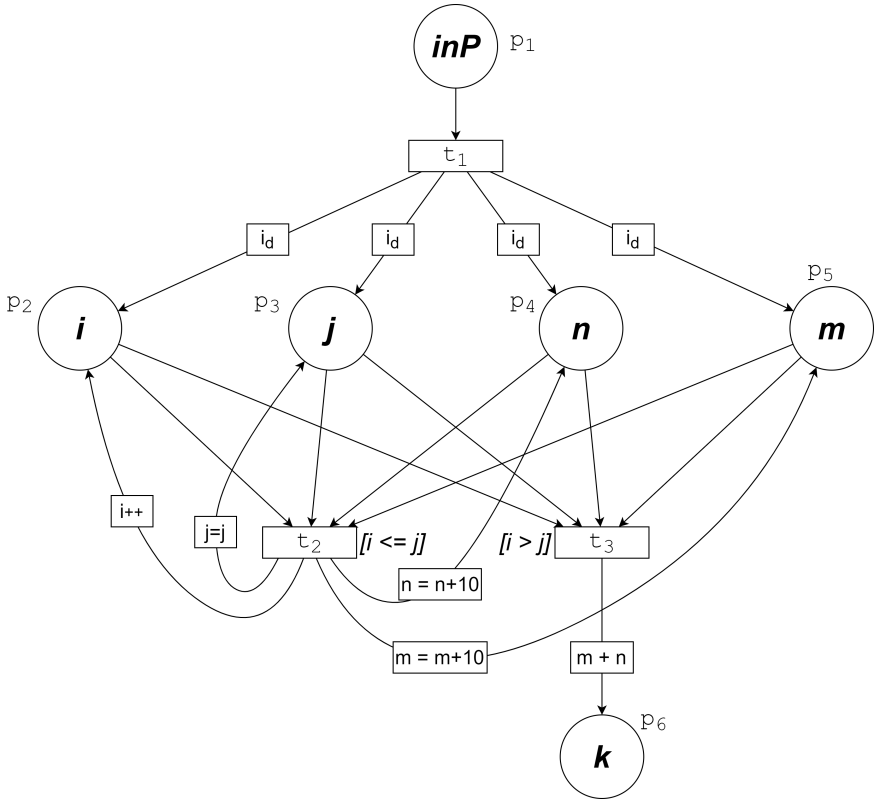


(b)

New CPN Model




```
int i, j, k, n, m;
while(i <=j){
m = m+10;
n = n+10;
i++;
}
k = m+n;
```

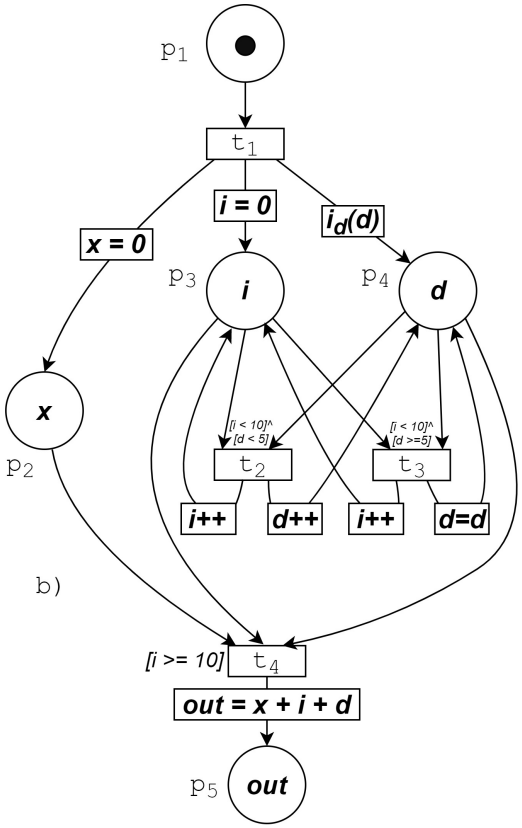


- Function is associated with every transition. (ATVA 2017)
- Function is associated with outgoing edges.

Loop involving code optimizing transformation

```
int i = 0;
int x = 0;
int d;
while (i<10)
 {while (d < 5)
  {d++;}
  i++;}
out = x + i + d;
```

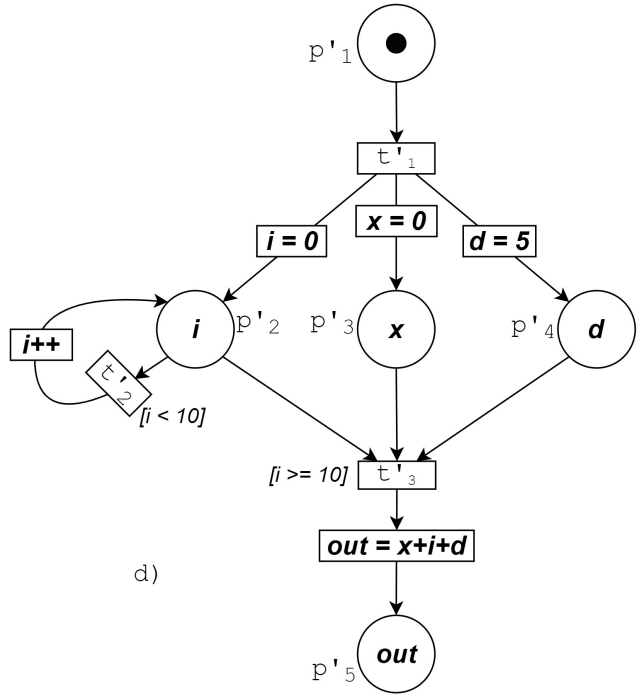
a)



b)

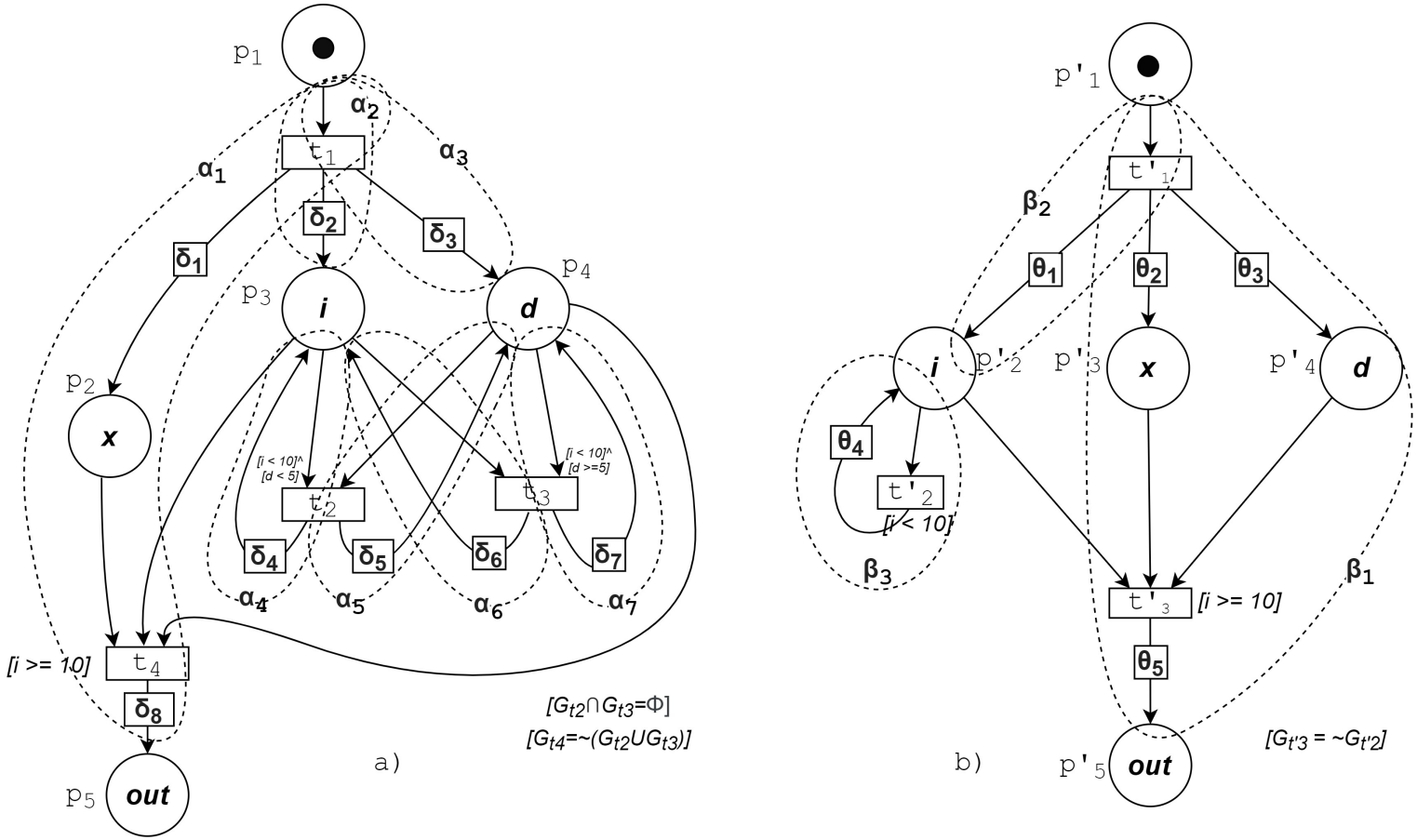
```
int i' = 0;
int x' = 0;
int d' = 5;
while (i'<10)
   {i'++;}
out' = x'+i'+d';
```

c)



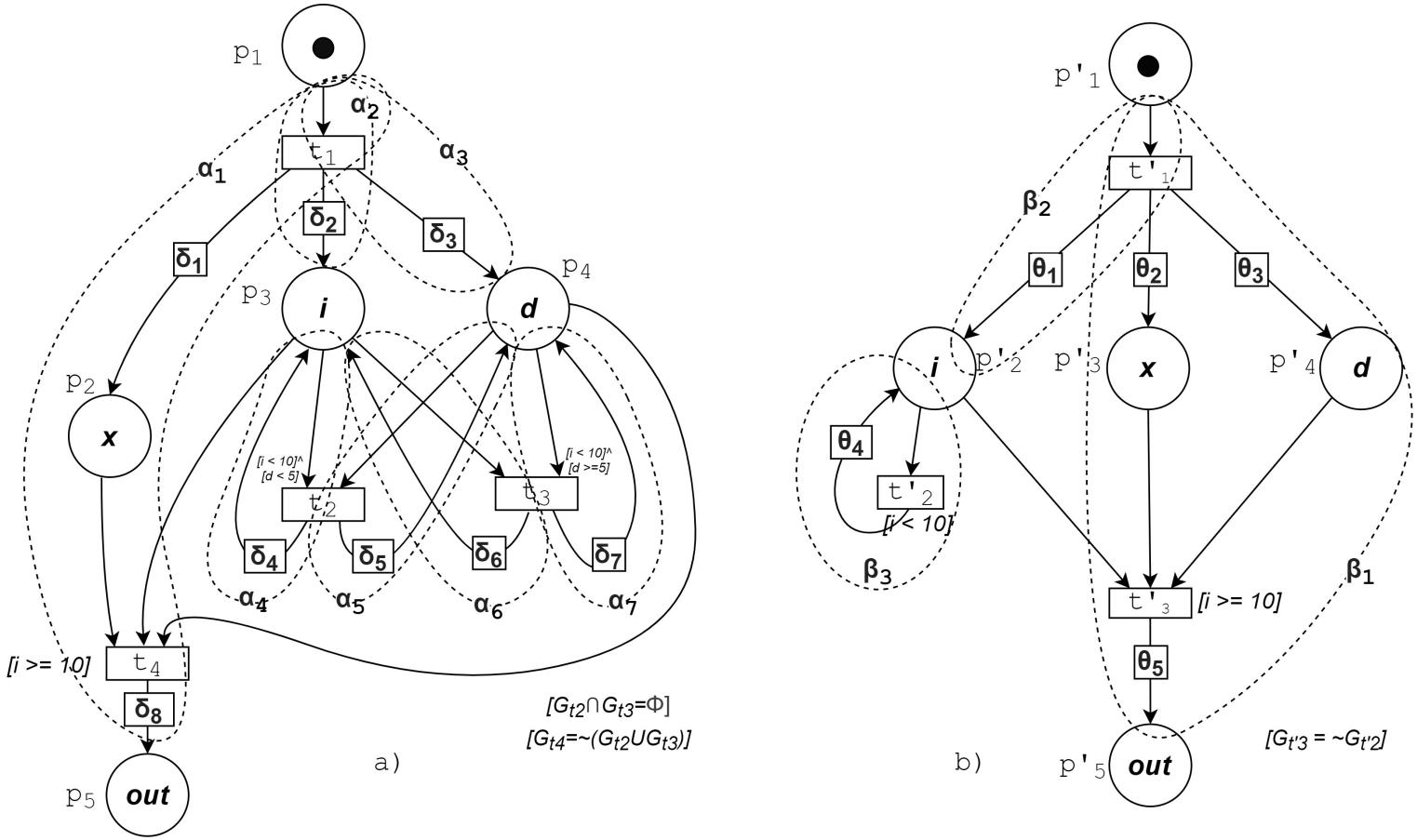
d)

Path Based Equivalence Checking Method



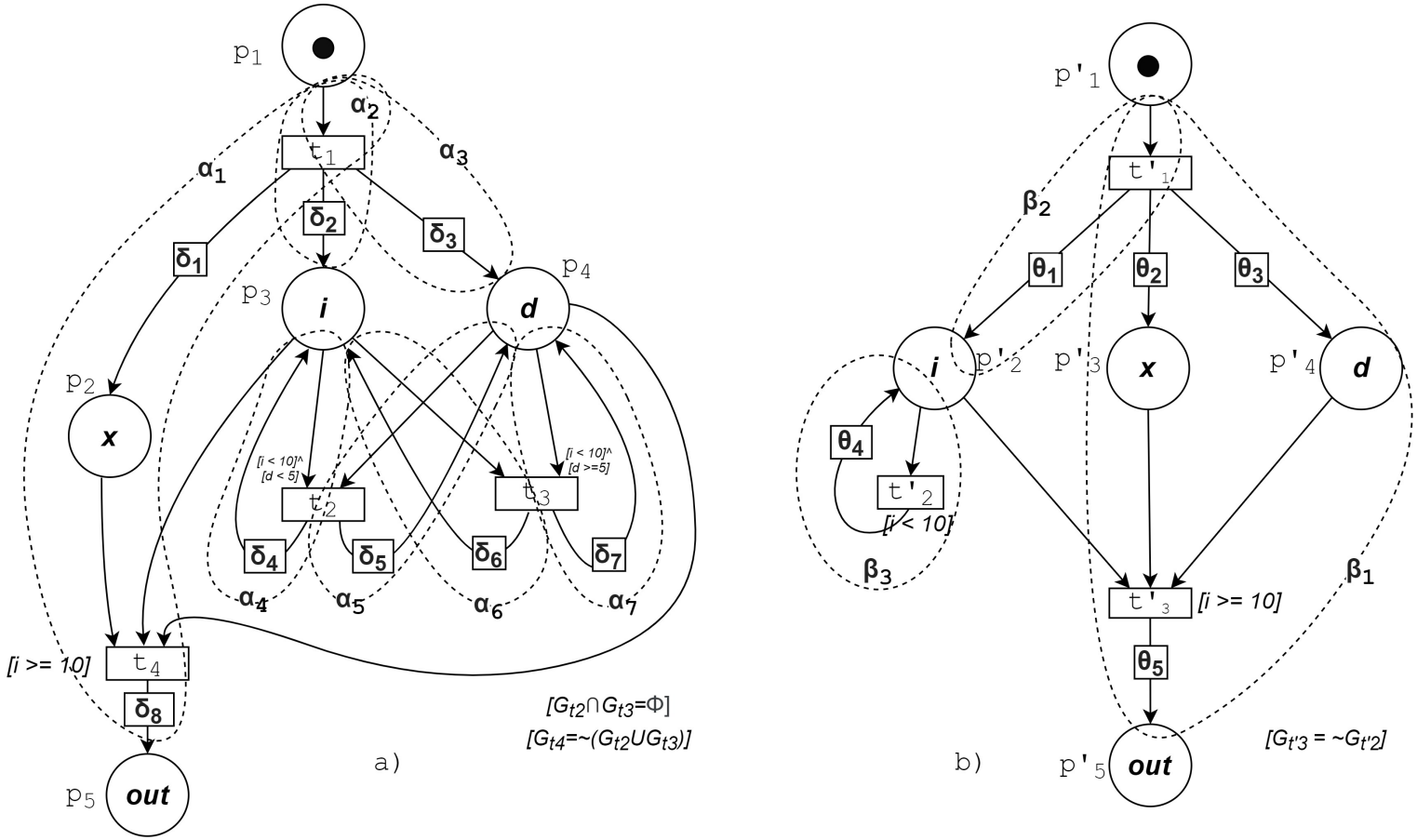
- $\mu_{p_5} = \langle \{t_1\}, \{t_2\}^n, \{t_3\}^m, \{t_4\} \rangle$
- $\mu_{p_5} = \langle \{\delta_1, \delta_2, \delta_3\}, \{\delta_4, \delta_5\}^n, \{\delta_6, \delta_7\}^m, \delta_8 \rangle$
- $\mu^r_{p_5} = \langle \rangle$

Path Based Equivalence Checking Method



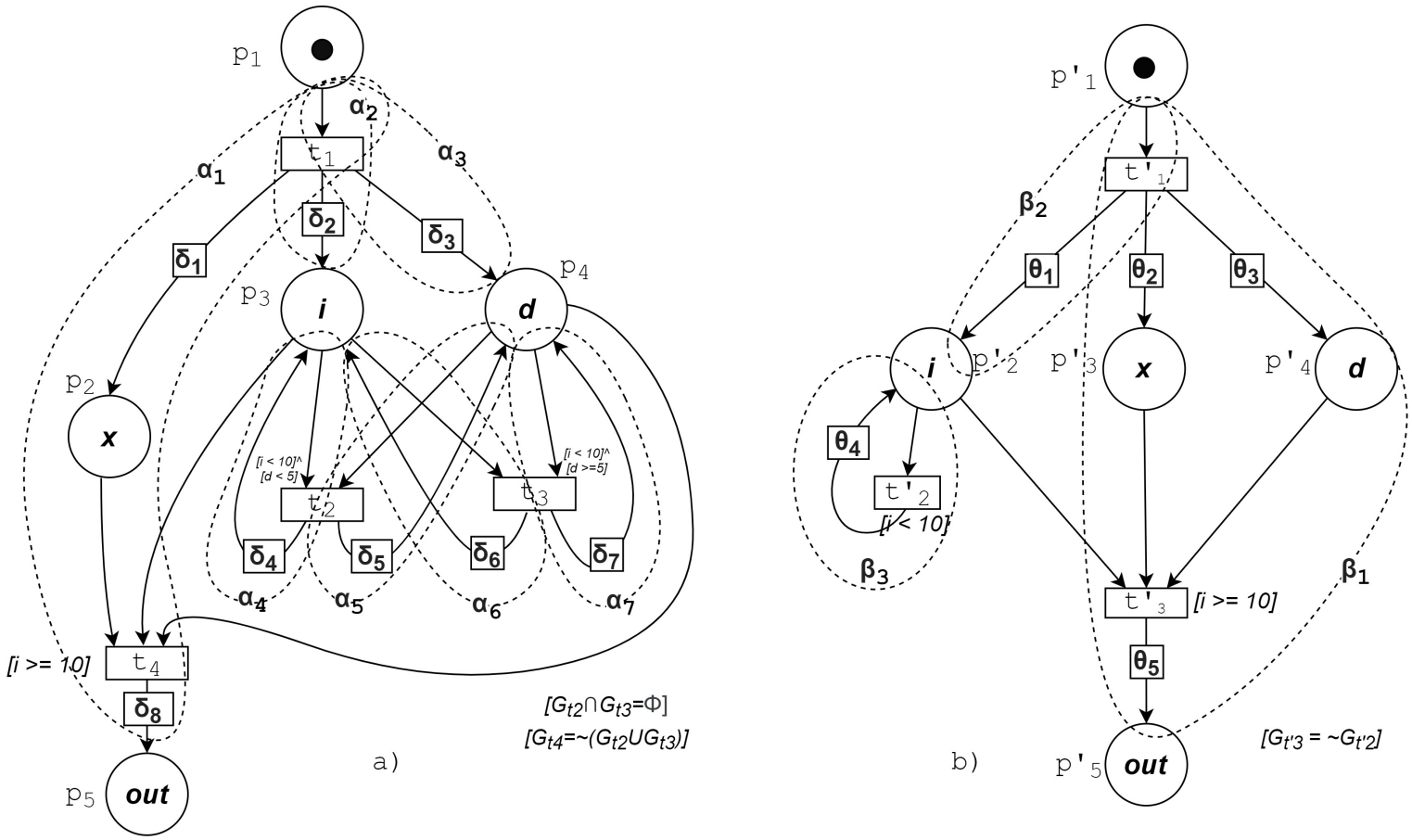
- $\mu_{p_5} = \langle \{t_1\}, \{t_2\}^n, \{t_3\}^m, \{t_4\} \rangle$
- $\mu_{p_5} = \langle \{\delta_1, \delta_2, \delta_3\}, \{\delta_4, \delta_5\}^n, \{\delta_6, \delta_7\}^m, \delta_8 \rangle$
- $\mu^r_{p_5} = \langle \rangle$

Path Based Equivalence Checking Method



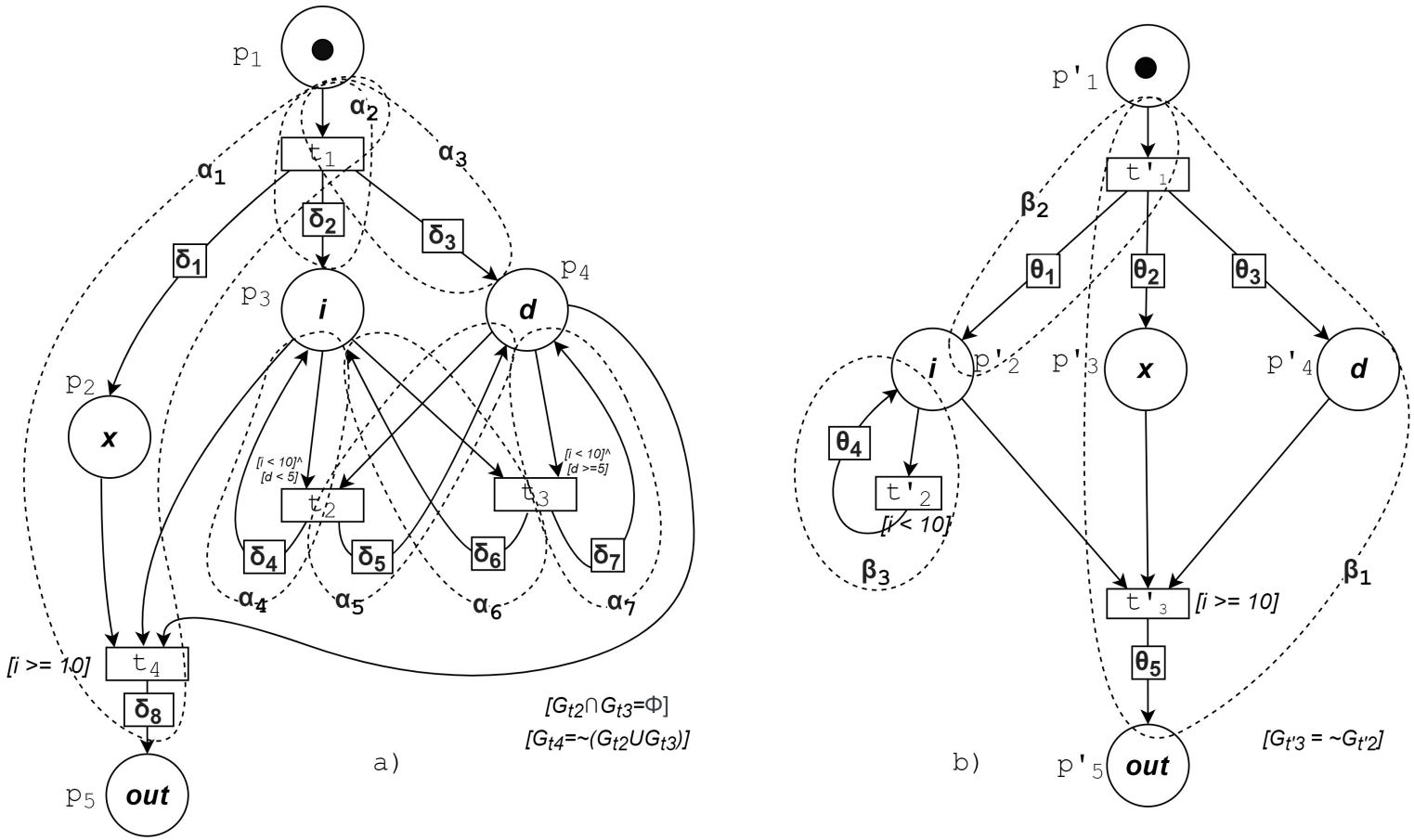
- $\mu_{p_5} = \langle \{t_1\}, \{t_2\}^n, \{t_3\}^m, \{t_4\} \rangle$
- $\mu_{p_5} = \langle \{\delta_1, \delta_2, \delta_3\}, \{\delta_4, \delta_5\}^n, \{\delta_6, \delta_7\}^m, \delta_8 \rangle$
- $\mu^r_{p_5} = \langle \rangle$

Path Based Equivalence Checking Method



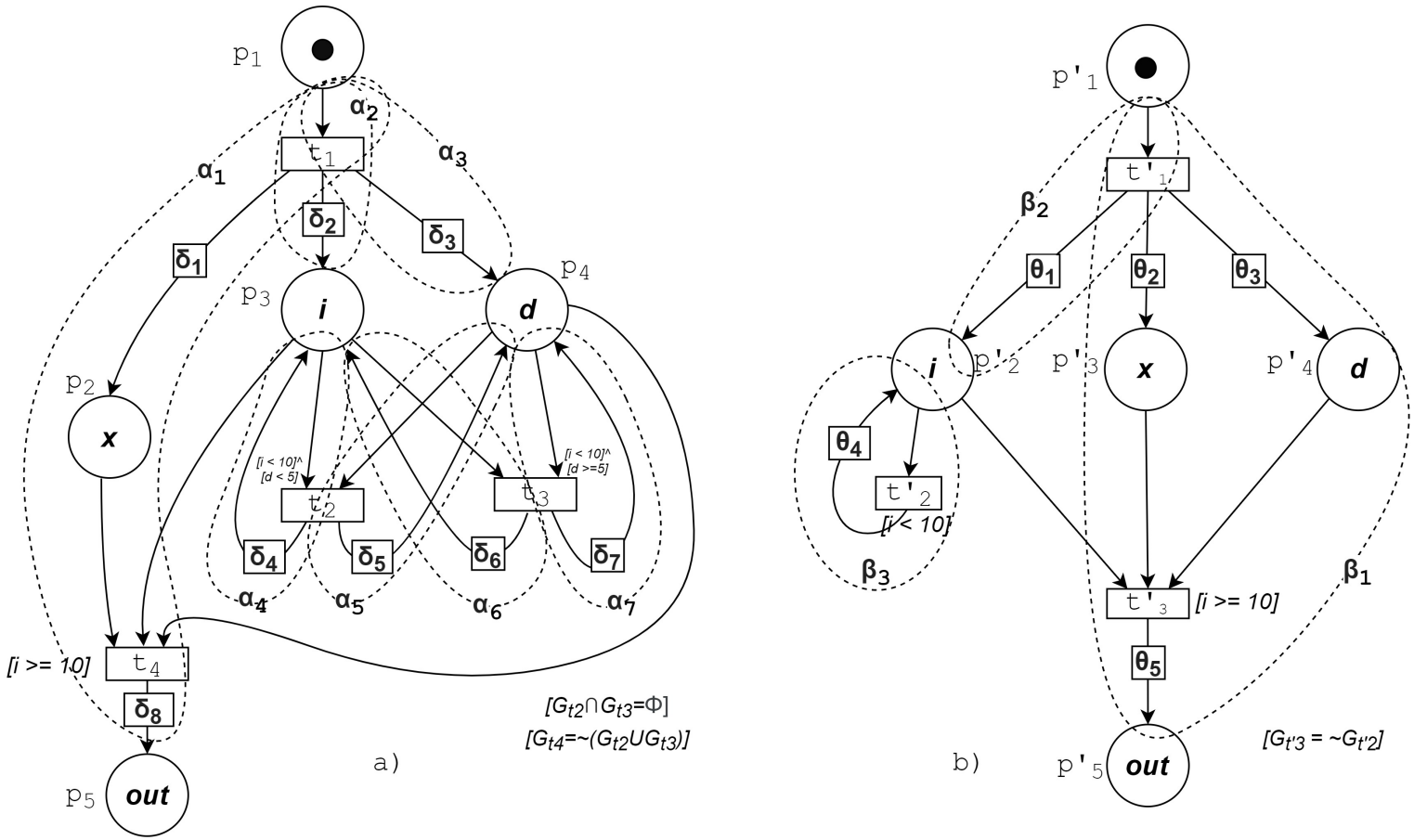
- $\mu_{p_5} = \langle \{\delta_2, \delta_3\}, \{\delta_4, \delta_5\}^n, \{\delta_6, \delta_7\}^m \rangle$
- $\mu^r_{p_5} = \langle \alpha_1 \rangle$.

Path Based Equivalence Checking Method



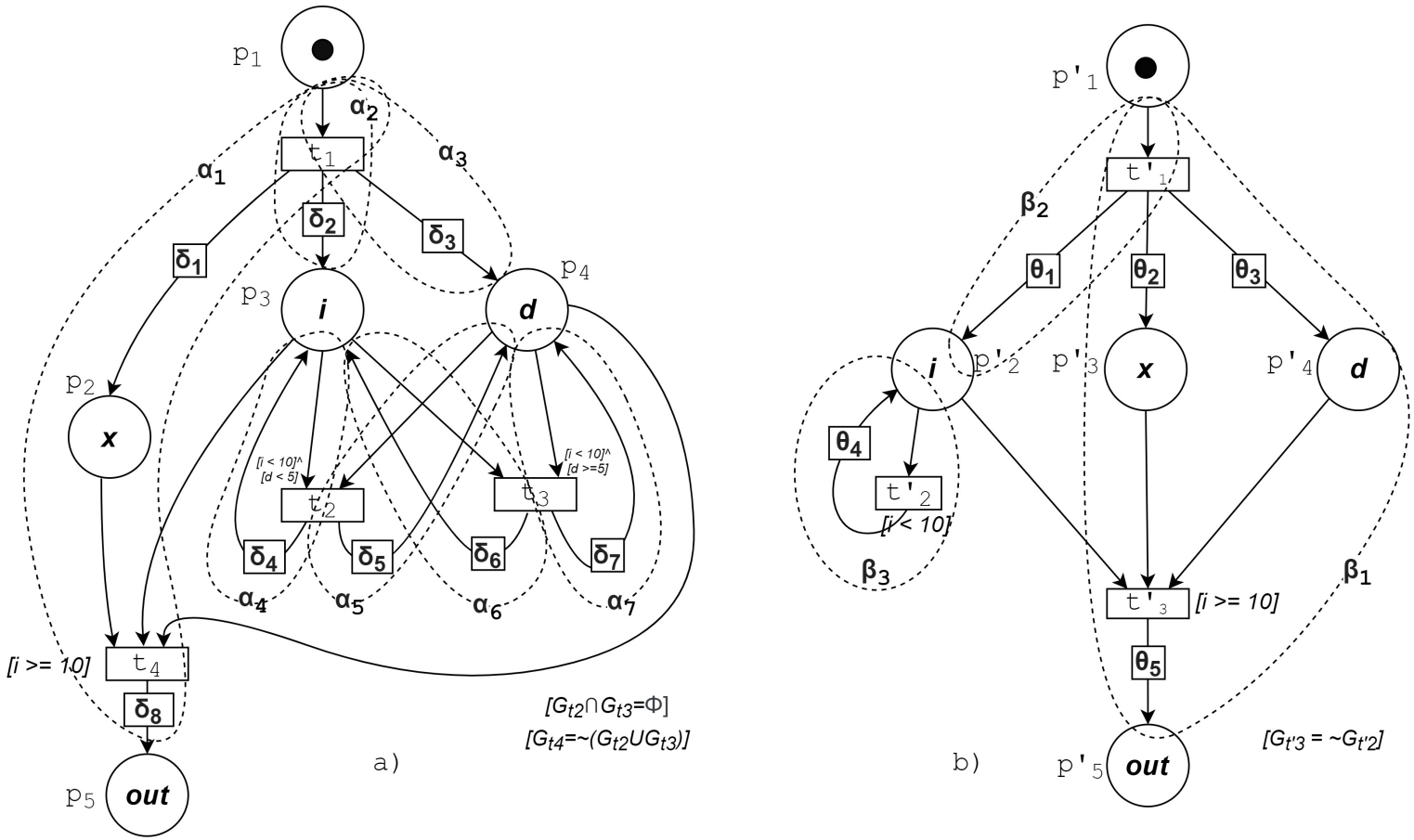
- $\mu_{p_5} = \langle \{\delta_2, \delta_3\}, \{\delta_4, \delta_5\}^n, \{\delta_6, \delta_7\}^m \rangle$
- $\mu^r_{p_5} = \langle \alpha_1 \rangle$.

Path Based Equivalence Checking Method



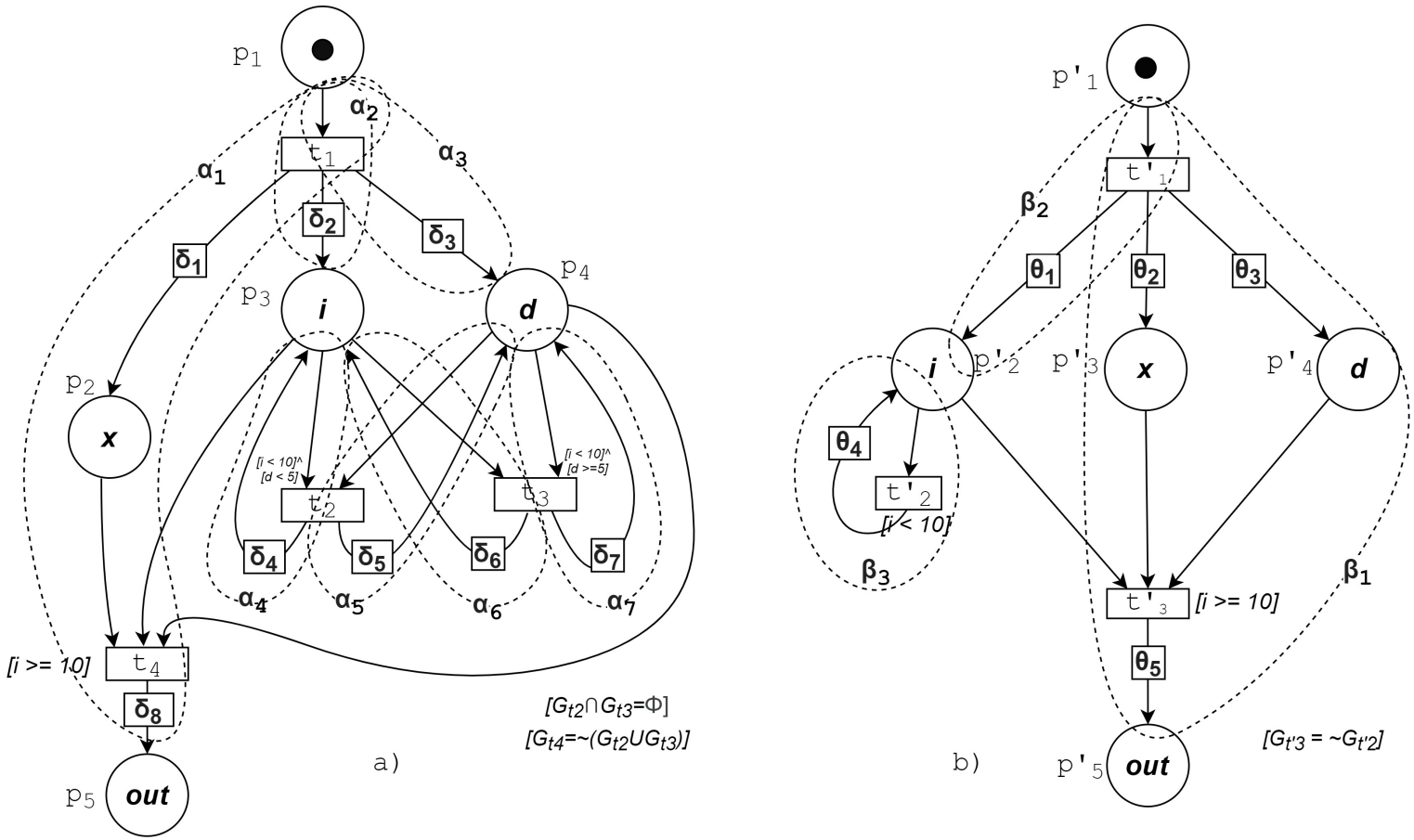
- $\mu_{p_5} = \langle \{\delta_2, \delta_3\}, \{\delta_4, \delta_5\}^n \rangle$
- $\mu^r_{p_5} = \langle \{\alpha_6 \parallel \alpha_7\}^m . \alpha_1 \rangle$.

Validity of Path Based Equivalence Checking Method



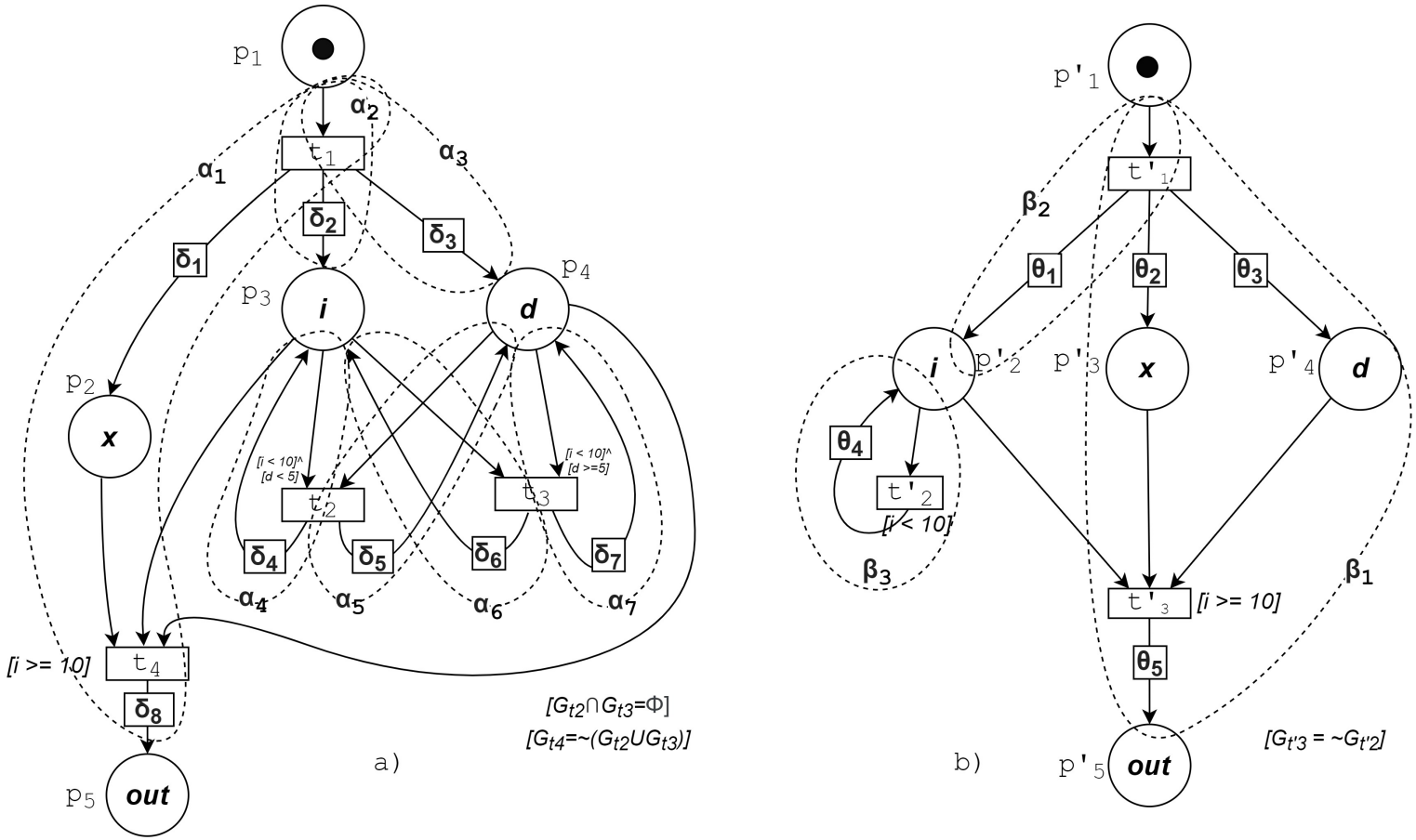
- $\mu_{p_5} = \langle \{\delta_2, \delta_3\} \rangle$
- $\mu^r_{p_5} = \langle \{\alpha_4 \parallel \alpha_5\}^n . \{\alpha_6 \parallel \alpha_7\}^m . \alpha_1 \rangle$.

Validity of Path Based Equivalence Checking Method



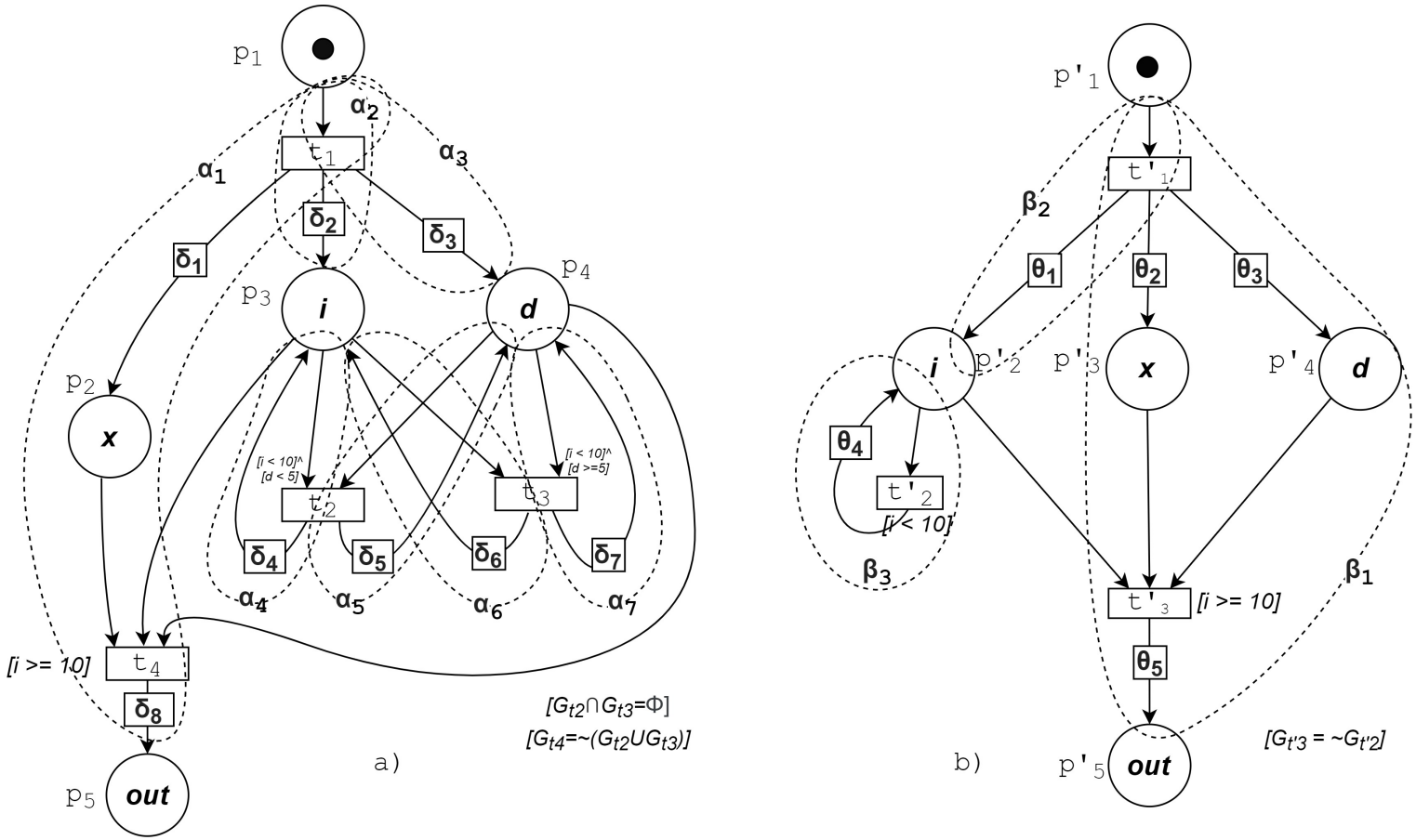
- $\mu_{p_5} = \langle\rangle$
- $\mu^r_{p_5} = \langle\{\alpha_2 \parallel \alpha_3\}.\{\alpha_4 \parallel \alpha_5\}^n.\{\alpha_6 \parallel \alpha_7\}^m.\alpha_1\rangle.$

Equivalence Checking Method



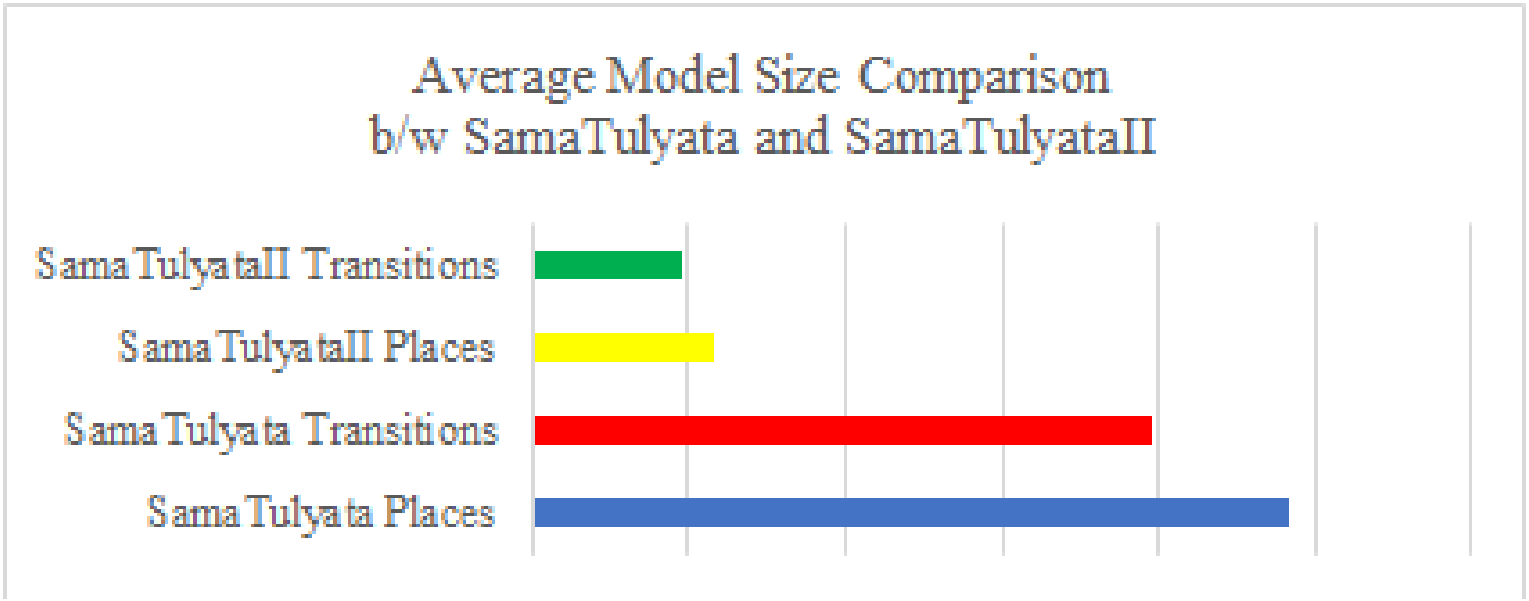
- $\alpha_2 \cong \beta_2$
- $(\alpha_1 \parallel (\alpha_3 \cdot (\alpha_5 \parallel \alpha_7))) \cong \beta_1$ [Path extension]
- $(\alpha_4 \parallel \alpha_6) \cong \beta_3$ [Path merging]

Equivalence Checking Method



- $\alpha_2 \cong \beta_2$
- $(\alpha_1 \parallel (\alpha_3 \cdot (\alpha_5 \parallel \alpha_7))) \cong \beta_1$ [Path extension]
- $(\alpha_4 \parallel \alpha_6) \cong \beta_3$ [Path merging]

Result Section



Class of Supported Transformations and Limitations

Code optimizing transformations

- Uniform and non-uniform code transformations
- code motion across loop and loop swapping

Parallelizing transformations:

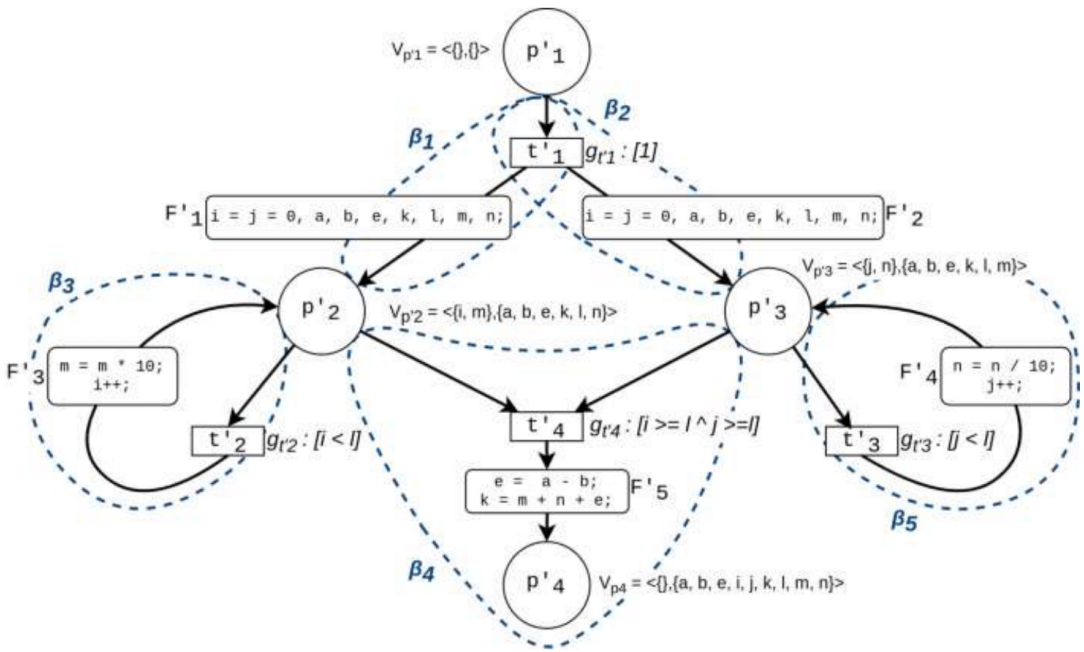
- Several thread level parallelizing transformations such as loop interchanging, loop splitting and merging, data locality etc.

Limitations:

- Loop-shifting [CAV 2008], software pipelining based transformations.
- Several code motions for array handling programs.
- Several loop transformations for any array handling programs.
- Equivalence checker is not proved complete with respect to these transformation.

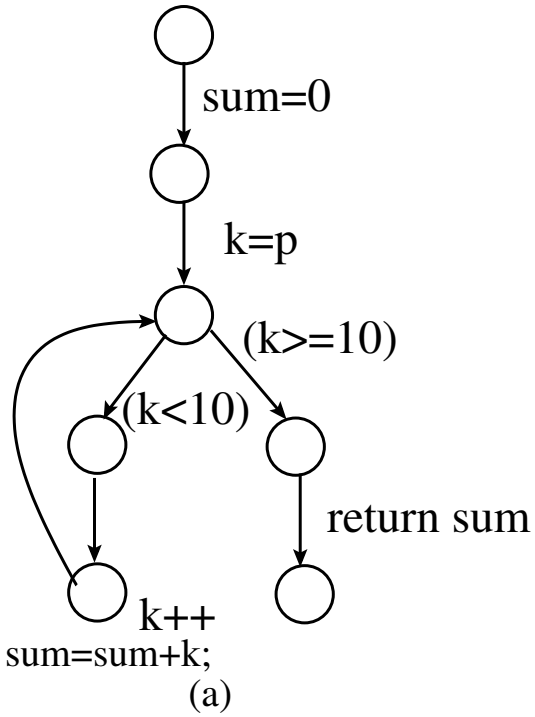
Future scope

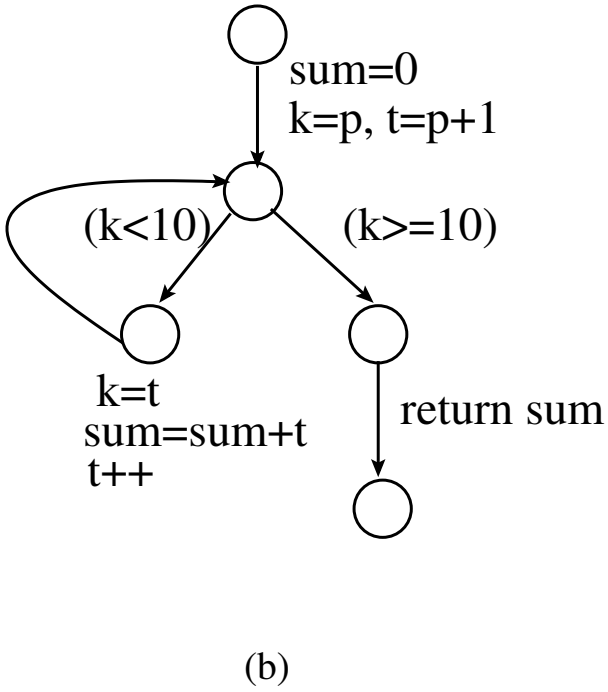
New iteration of Petri Net model

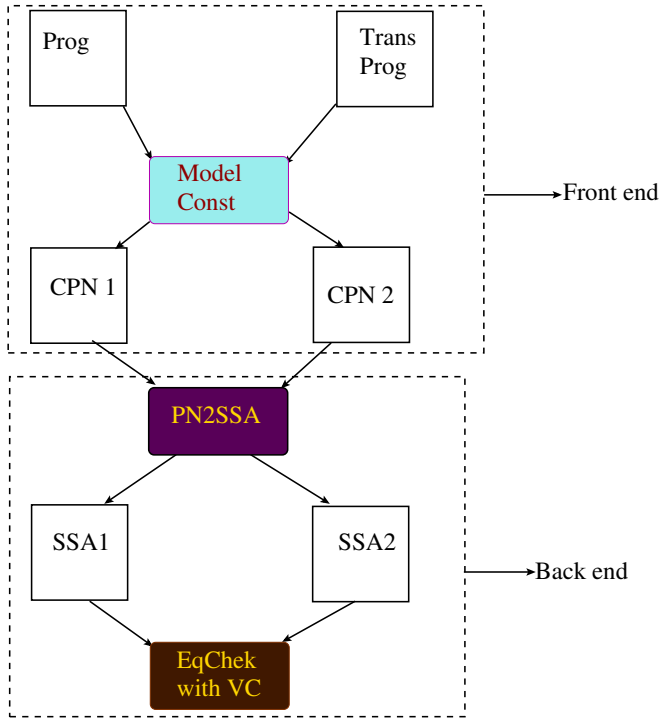



```
int i = j = 0,a,b,e,k,l,m,n;
scanf("%f,%f,%f,%f,%f",
       &a,&b,&l,&m,&n);

#parbegin scop
while( i < l ) {
  m = m * 10;
  i++;
}
||
while( j < l ) {
n = n / 10;
j++;}
#parend scop

e = a - b;
k = m + n + e;
```

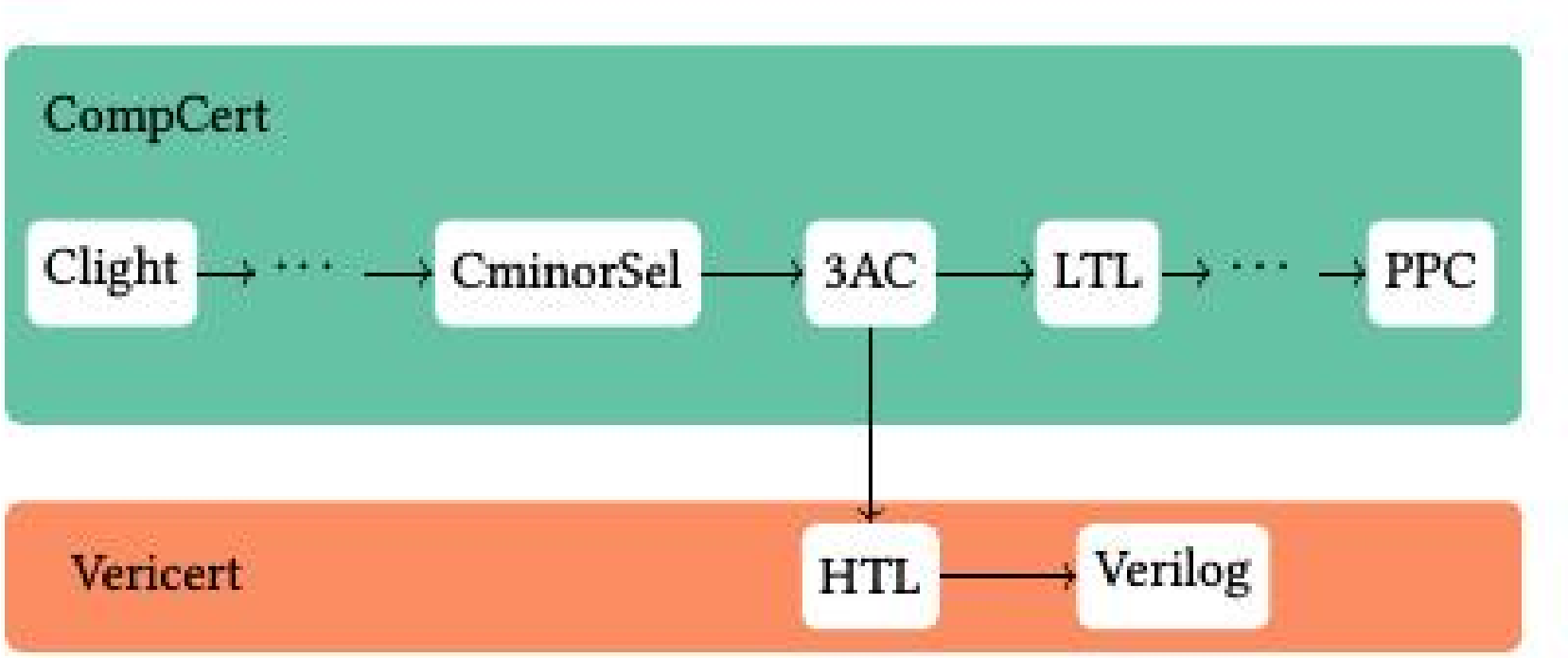
Future scope







Future scope



https://yannherklotz.com/docs/drafts/formal_hls.pdf

Future scope

1. Use of equivalence checker for automated evaluation in CS1.[3]

[3] https://github.com/soumyadipcsis/autoval

Thank You !!!