PNPEq: Verification of Scheduled Conditional Behavior in Embedded Software using Petri Nets

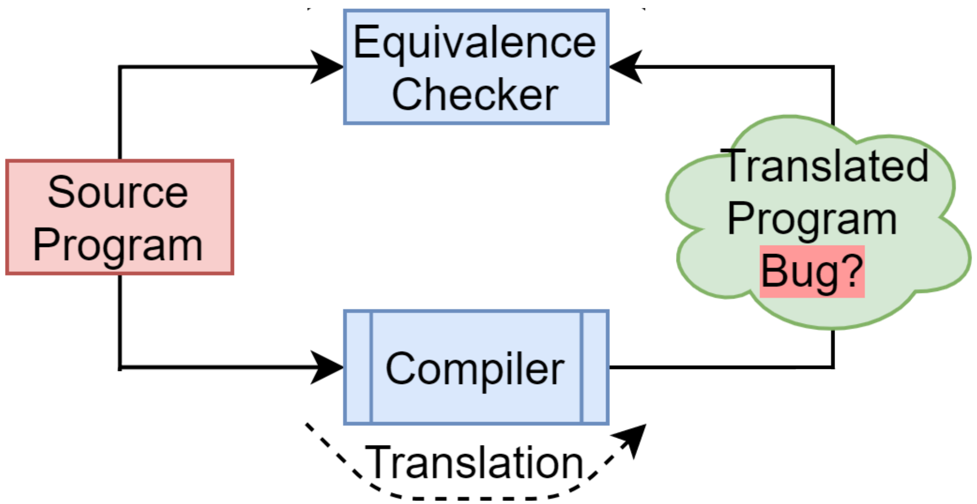
[1,2]Rakshit Mittal, [1,3]Dominique Blouin, [2,3]Soumyadip Bandyopadhyay

[1]Telecom Paris, Institut Polytechnique de Paris, France

[2]Birla Institute of Technology & Science Pilani, Goa, India

[3]Hasso Plattner Institut, Potsdam, Germany

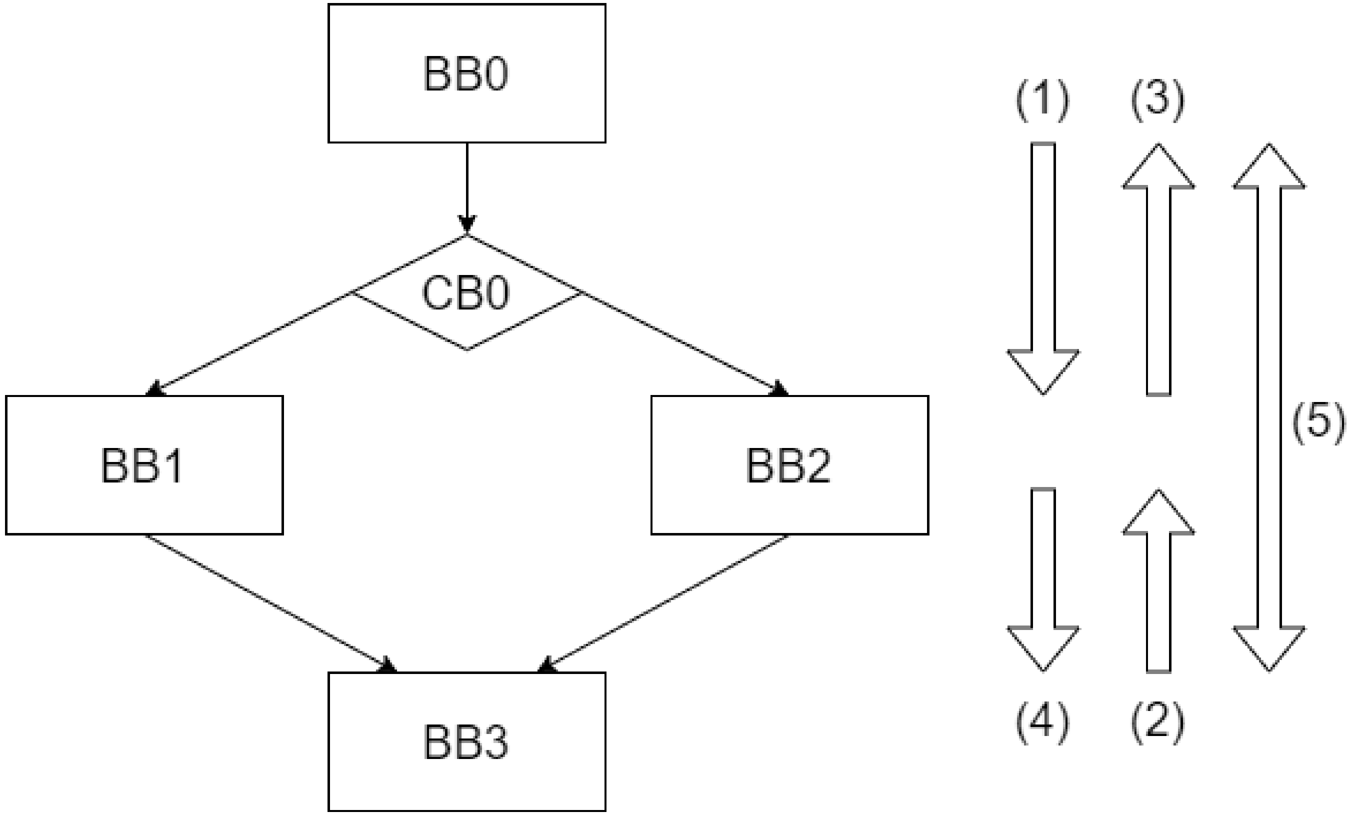
Background



Q. Is the compiler translation correct?
Q. Are there any bugs in the translated program?
Q. Are the source and translated program equivalent?

Background

Optimizing Transformations



1) Duplicating down
2) Duplicating up
3) Boosting up
4) Boosting down
5) Useful Move

Bacon et al, “Compiler Transformations for High-Performance Computing, ACM Computing Survey, 1994

State-of-the-Art

Equivalence Checking

- **Bisimulation-Based Methods**
  - Proposed by Amir Pnueli [TACAS 1998]
  - Enhanced by Necula et al [PLDI 2000] and Rinard et al [MIT 2000]
  - Modified by Kundu et al [CAV 2008]

- **Inductive Inference-Based Methods** {Only scalar-handling problems}
  - Matthias et al [ASE 2014]

{Termination not guaranteed}

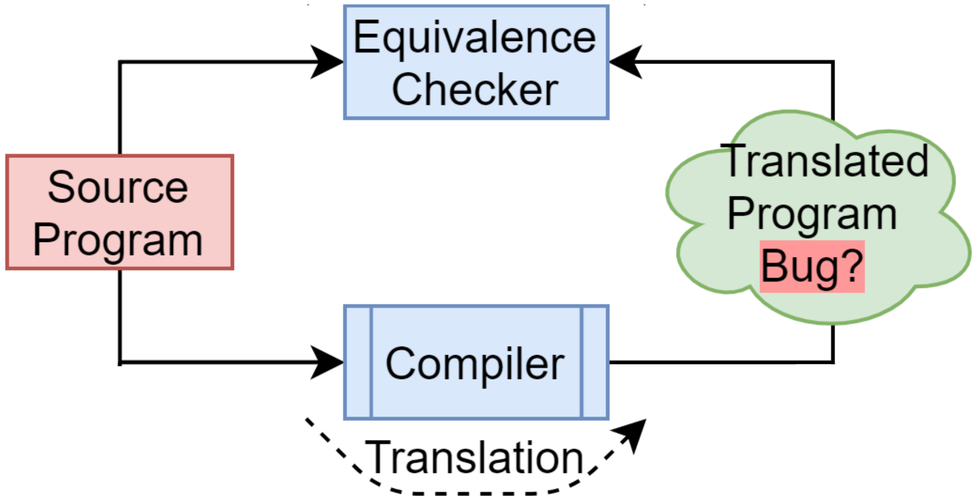
State-of-the-Art

Equivalence Checking

- **CDFG Path-Based Methods**
  - Karfa et al verify transformations of the SPARK compiler, control structure of program altered [TCAD 2012]
  - Modified by Banerjee et al [TCAD 2014] (value-propagation) and Chouksey et al [TCAD 2019] (extended value-propagation)

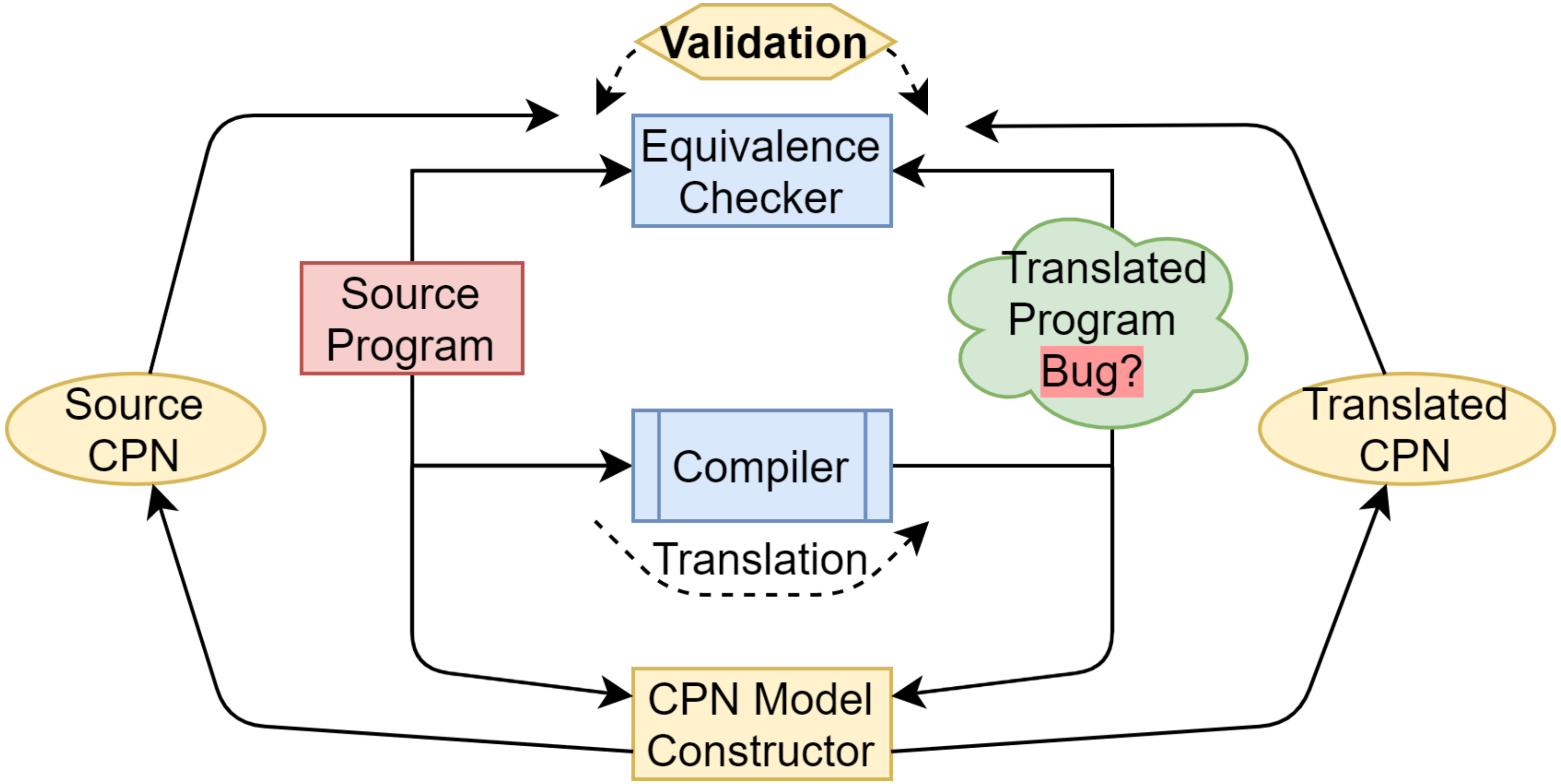
- **Petri Net Path-Based Methods**
  - SamaTulyata [ATVA 2017] {Larger model size}
  - SamaTulyata2 [PNSE 2020] {Large model size}
  - Mittal et al [ICSOFT 2021] {Path explosion}

Background



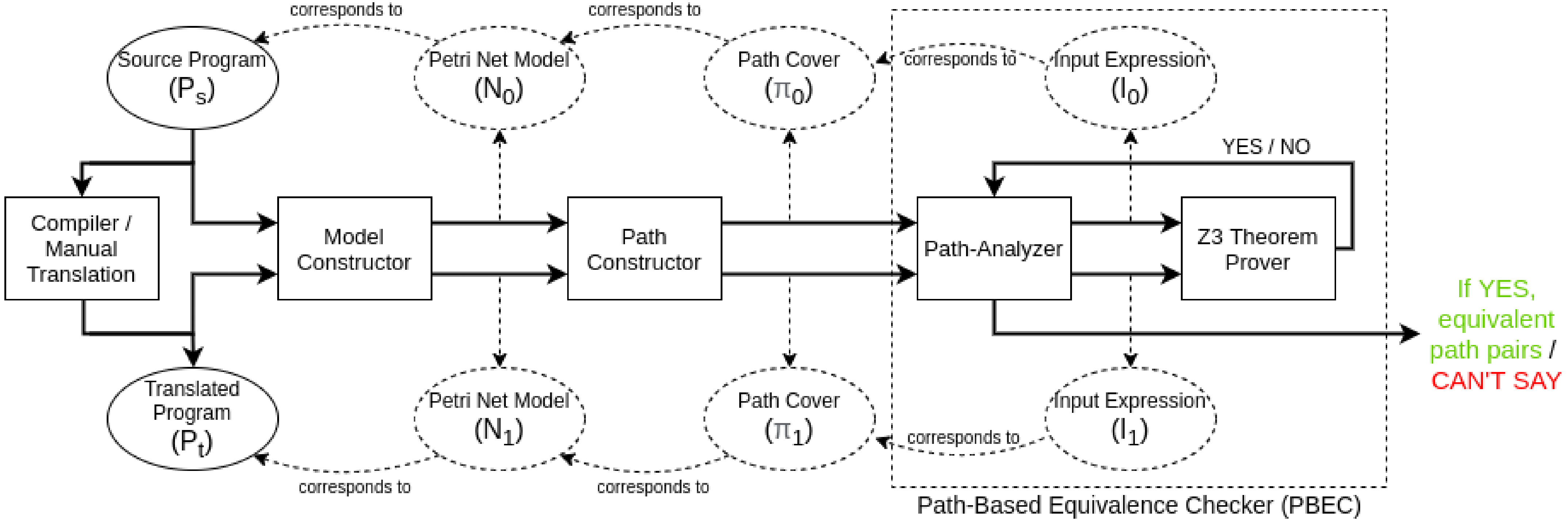
Q. Is the compiler transformation correct?
Q. Are there any bugs in the translated program?
Q. Are the source and translated program equivalent?

Motivation



Q. Are the source and translated CPN Equivalent?
PNPEq : Petri Net Program Equivalence

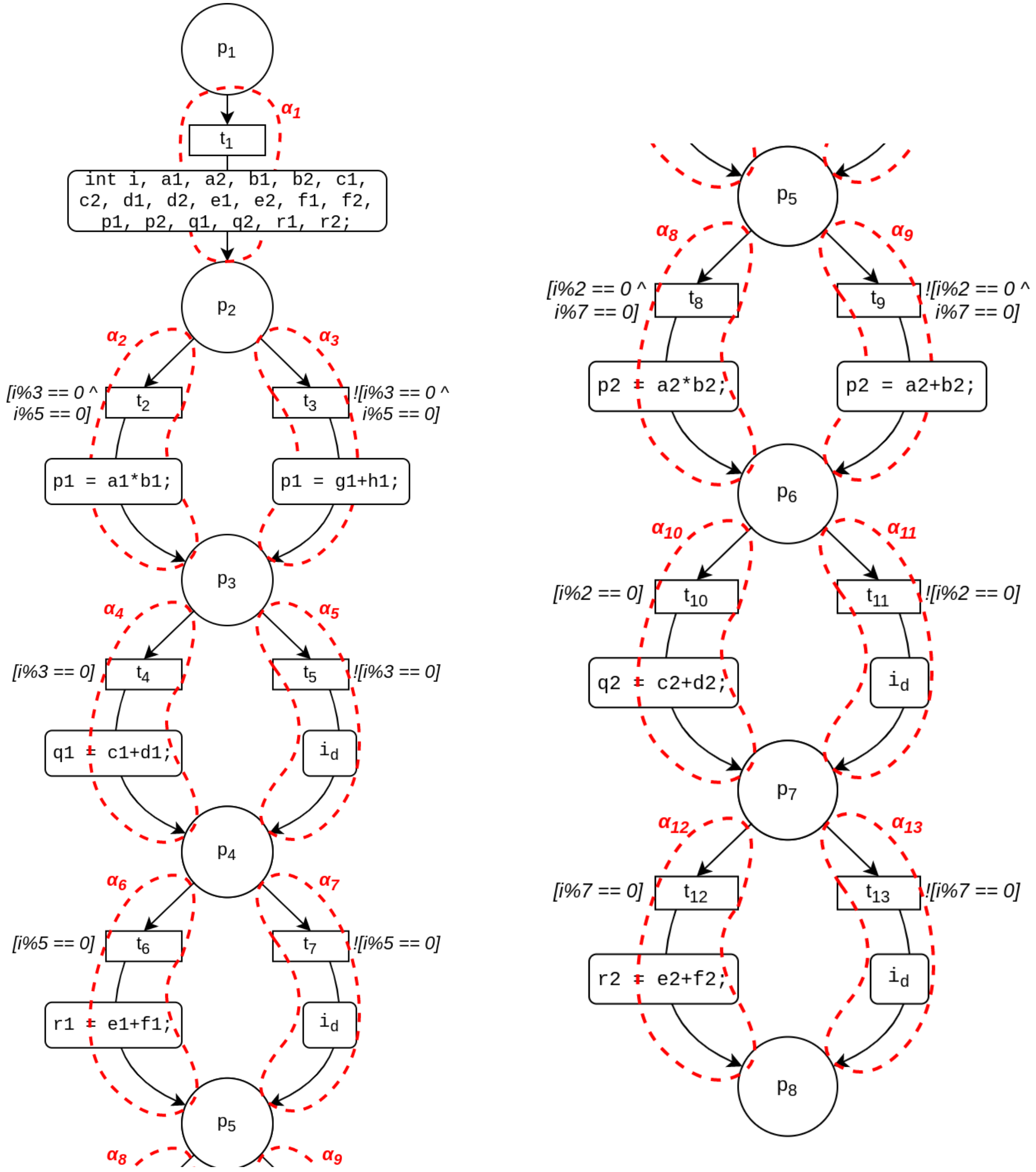
Proposed Toolchain



Cut-point: In-port / Out-port / Place with back-edge / Bifurcation point

Path Construction: From cut-point to cut-point

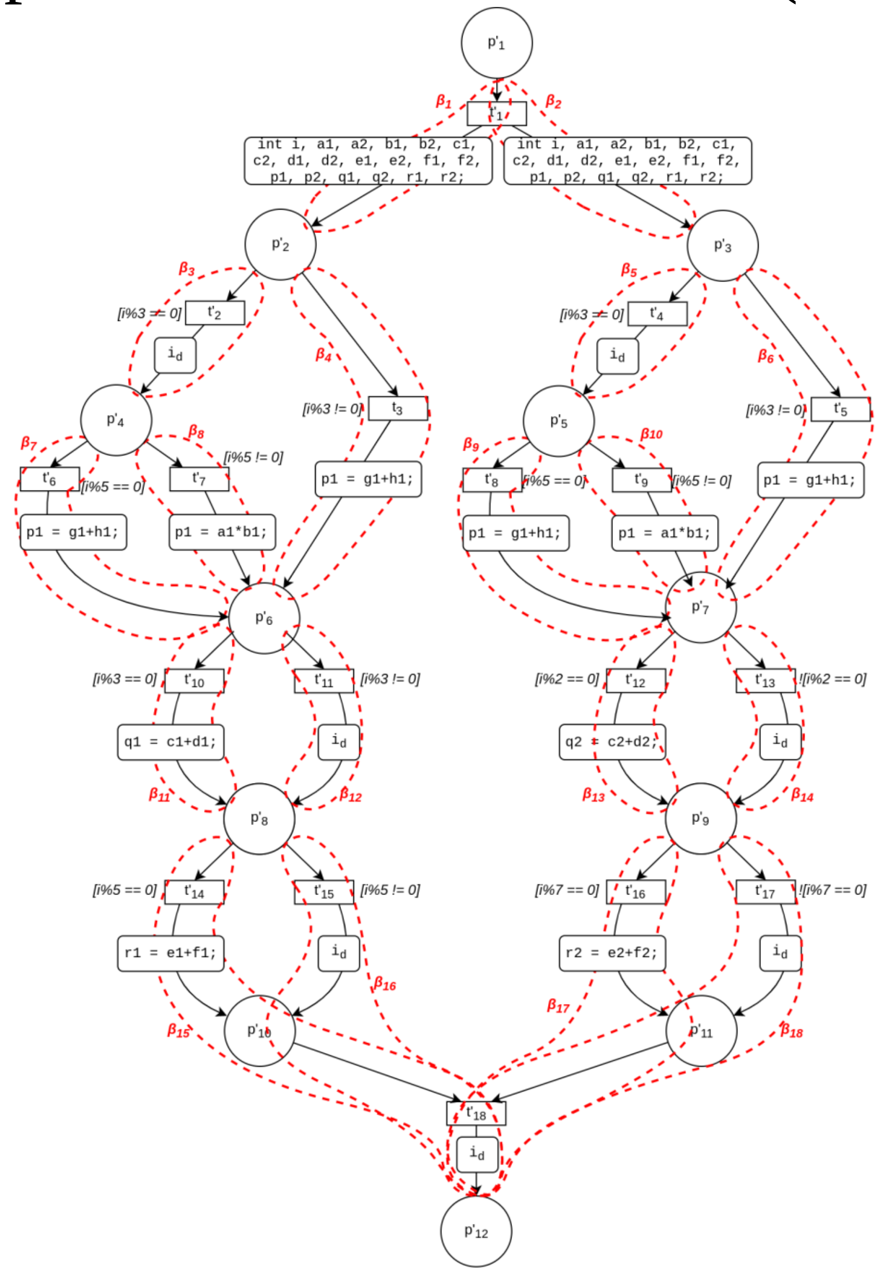
Example - Source Petri Net (6 cycles)



```
int i,a1,a2,b1,b2,c1,c2,
d1,d2,e1,e2,f1,f2,p1,p2,
q1,q2,r1,r2;
if (i%3 == 0 && i%5 == 0)
{ p1 = a1 * b1; }
else
{ p1 = g1 + h1; }
if (i%3 == 0)
{ q1 = c1 + d1; }
if (i%5 == 0)
{ r1 = e1 + f1; }
if (i%2 == 0 && i%7 == 0)
{ p2 = a2 * b2; }
else
{ p2 = g2 + h2; }
if (i%2 == 0)
{ q2 = c2 + d2; }
if (i%7 == 0)
{ r2 = e2 + f2; }
```

Source Program

Example - Translated Petri Net (2 cycles)



```
int i,a1,a2,b1,b2,c1,c2,
d1,d2,e1,e2,f1,f2,p1,p2,
q1,q2,r1,r2;
#parbegin scop
if (i%3 == 0)
{if (i%5 == 0)
{p1 = a1 * b1;}
else
{p1 = g1 + h1;}}
else
{ p1 = g1 + h1; }
if (i%3 == 0)
{ q1 = c1 + d1; }
if (i%5 == 0)
{ r1 = e1 + f1; }
||
if (i%2 == 0)
{if (i%7 == 0)
{p2 = a2 * b2;}
else
{p2 = g2 + h2;}}
else
{ p2 = g2 + h2; }
if (i%2 == 0)
{ q2 = c2 + d2; }
if (i%7 == 0)
{ r2 = e2 + f2; }
#parend scop
```

Translated Program

Path Concatenation (·), Equivalence Checking

Path Extension
The conditions of execution are conjuncted and data transformations are substituted.

Path Merging
The data transformation remains the same but the condition of execution is disjuncted.

Equivalence Checking Principle
For all paths in N_1 there exists path in N_0 such that the paths are equivalent and vice versa, implies $N_0 \simeq N_1$

Two paths are equivalent if R_{path} : condition of execution and r_{path} : data transformation match for both the paths. That is,

$$R_\alpha \simeq R_\beta \quad \text{and} \quad r_\alpha = r_\beta$$

$\beta_1 \simeq \alpha_1$
$\beta_2 \simeq \alpha_1$
$(\beta_3 \cdot \beta_7) \simeq \alpha_2$ (extension)
$(\beta_5 \cdot \beta_9) \simeq \alpha_8$ (extension)
$(\beta_4 \cdot \beta_8) \simeq \alpha_3$ (merging)
$(\beta_6 \cdot \beta_{10}) \simeq \alpha_9$ (merging)
$\beta_{11} \simeq \alpha_4$
$\beta_{12} \simeq \alpha_5$
$\beta_{13} \simeq \alpha_{10}$
$\beta_{14} \simeq \alpha_{11}$
$\beta_{15} \simeq \alpha_6$
$\beta_{16} \simeq \alpha_7$
$\beta_{17} \simeq \alpha_{12}$
$\beta_{18} \simeq \alpha_{13}$

Experimentation

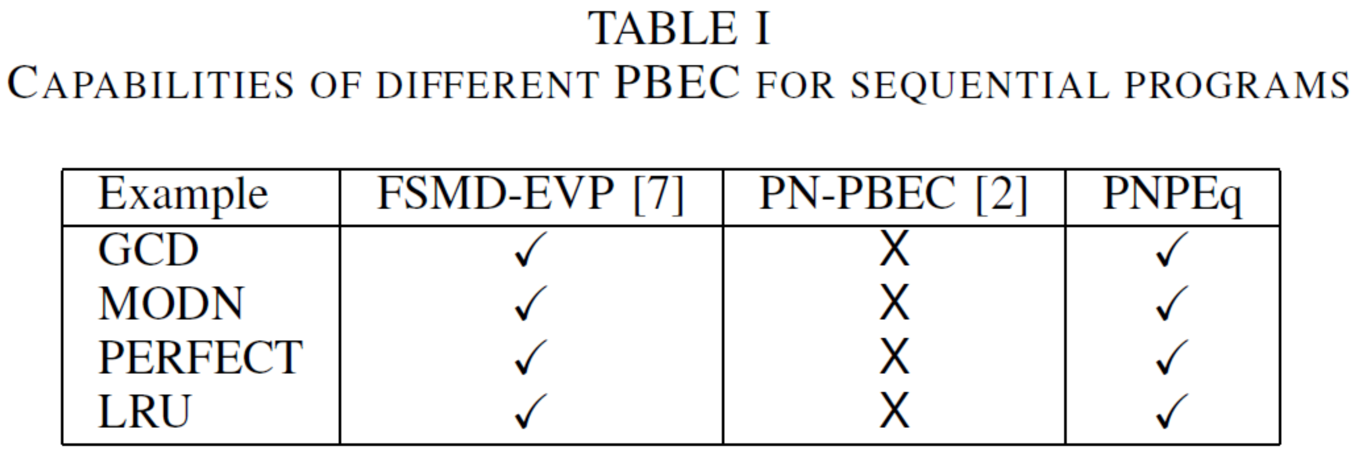


TABLE I

CAPABILITIES OF DIFFERENT PBEC FOR SEQUENTIAL PROGRAMS

Example	FSMD-EVP [7]	PN-PBEC [2]	PNPEq
GCD	✓	X	✓
MODN	✓	X	✓
PERFECT	✓	X	✓
LRU	✓	X	✓

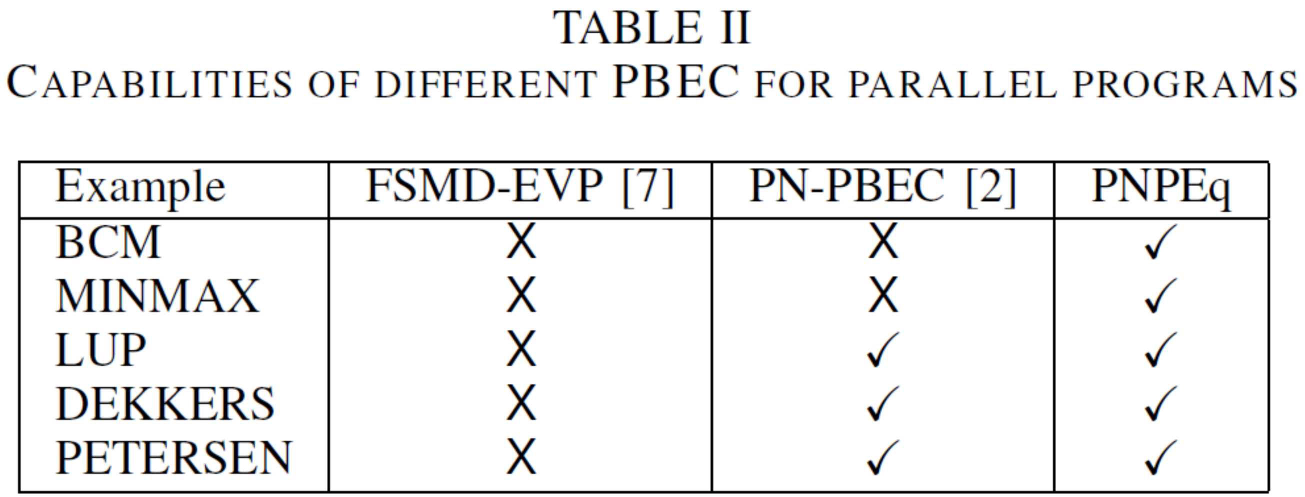


TABLE II

CAPABILITIES OF DIFFERENT PBEC FOR PARALLEL PROGRAMS

Example	FSMD-EVP [7]	PN-PBEC [2]	PNPEq
BCM	X	X	✓
MINMAX	X	X	✓
LUP	X	✓	✓
DEKKERS	X	✓	✓
PETERSEN	X	✓	✓

[2] Mittal et al “Towards an approach for translation validation of thread-level parallelizing transformations using colored petri nets,” in 16th ICSOFT, 2021.

[7] Chouksey et al, “Translation validation of code motion transformations involving loops,” IEEE TCADICS, 2019.

Limits and Capabilities

Cannot validate

Array-handling programs
Software pipelining transformations
Loop reversal transformations
Invariant assertions based transformations

Can validate

Uniform and non-uniform code transformations
Loop swapping transformation
Thread-level parallelizing transformations
for scalar-handling programs

Thank You

Questions?

e-mail: rakshit.mittal@telecom-paris.fr (myself)
dominique.blouin@telecom-paris.fr (Dominique)
soumyadipb@goa.bits-pilani.ac.in (Soumyadip)