Abstraction Techniques

Natalia Sidorova

More on abstractions

- abstract guards
- abstractions and deadlocks
- abstraction by restriction

Abstracting guards

We apply pos,0,neg abstraction for $\mathbb{Z}$.
We want to abstract

```
if
:: (x > 3) -> dosomething1
:: (x < -3) -> dosomething2
:: else -> dosomething3
fi
```

What do we get then?

Abstracting guards

We want to have more behaviour.

For all numbers abstracted to neg or 0, x > 3 is false.
There are numbers abstracted to pos such that x > 3 is true. So, x > 3 is abstracted to x == pos.

For all numbers abstracted to pos or 0, x < -3 is false.
There are numbers abstracted to neg such that x < -3 is true. So, x < -3 is abstracted to x == neg.

“else” means here $-3 \leq x \leq 3$.
There are numbers abstracted to pos, 0, and neg that satisfy it. So “else” is abstracted to true.

Abstracting guards

```
if
:: (x > 3) -> dosomething1
:: (x < -3) -> dosomething2
:: else -> dosomething3
fi
```

becomes

```
if
:: x == pos -> abstract-dosomething1
:: x == neg -> abstract-dosomething2
:: abstract-dosomething3
fi
```

Can you propose a (possibly) better abstraction for this example?

Abstracting guards

What about abstracting

```
if
:: (x == y) -> dosomething1
:: else -> dosomething2
fi
```

if you have pos, 0, neg abstraction?

Abstracting guards

What about abstracting

```
if
:: (x == y) -> dosomething1
:: else -> dosomething2
fi
```

if you have pos, 0, neg abstraction?

```
if
:: (x == y) -> abstract-dosomething1
:: !(x == 0 && y == 0) -> abstract-dosomething2
fi
```

Abstraction and deadlocks

```
if
:: x > 0 -> dosomething1
:: x < 0 -> dosomething2
fi
```

Note that there is no “else” option and we get deadlock when x == 0.

Now apply an abstraction that maps nonnegative numbers to nonneg and negative numbers to neg. What happens then?

Abstraction and deadlocks

```
if
:: x > 0 -> dosomething1
:: x < 0 -> dosomething2
fi
```

Note that there is no “else” option and we get deadlock when x == 0.

Now apply an abstraction that maps nonnegative numbers to nonneg and negative numbers to neg. What happens then?

Conclusion: Abstracting a system, we can lose deadlocks. Thus, the abstract system having no deadlock does not imply the concrete system has none.

Abstraction and deadlocks

Consider pos, 0, neg abstraction again and

```
/* invariant x == y here */
if
:: (x == y) -> dosomething1
:: else -> goto deadlock-state
fi
```

In the abstract system,

```
if
:: (x == y) -> abstract-dosomething1
:: !(x == 0 && y == 0) -> goto deadlock-state
fi
```

Abstraction and deadlocks

Consider pos, 0, neg abstraction again and

```
/* invariant x == y here */
if
:: (x == y) -> dosomething1
:: else -> goto deadlock-state
fi
```

In the abstract system,

```
if
:: (x == y) -> abstract-dosomething1
:: !(x == 0 && y == 0) -> goto deadlock-state
fi
```

Thus, we can introduce new deadlocks with abstraction, namely, unreachable deadlocks can become reachable.

Abstraction by restriction

Goal: Forbid some behaviour.

Useful for the debugging: to prove that some LTL property does not hold.

Done by removing states, transitions, or strengthening the guards.

More on abstractions

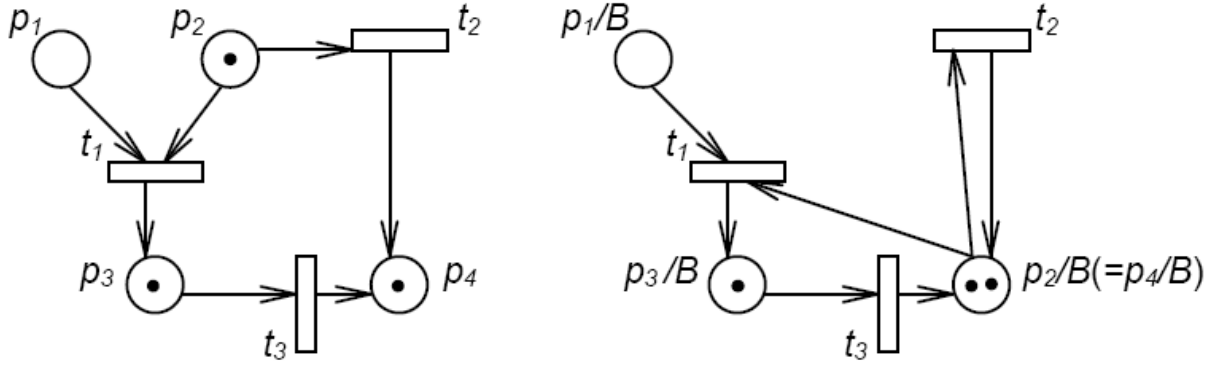
Chapter 11 of Berard et al., Systems and Software Verification

Abstraction for Petri nets

- Data abstraction for coloured Petri nets: the same as data abstraction we considered.
- Place fusion and addition of transitions to add behaviour
- Place replacement and removing of transitions to restrict the behaviour

Place fusion

Let $B \subseteq P \times P$ be an equivalence relation. Then N can be reduced by fusing B-equivalent places into a single place. We get a new net N/B.
The transitions are untouched but the arcs between places and transitions follow the fusion process:
if ${}^\bullet t$ (resp. $t^\bullet$) is $\{p_1, \ldots, p_k\}$ in N,
then in N/B, ${}^\bullet t$ (resp. $t^\bullet$) is $\{p_1/B, \ldots, p_k/B\}$.
A marking M in N is fused into a marking $m = M/B$.

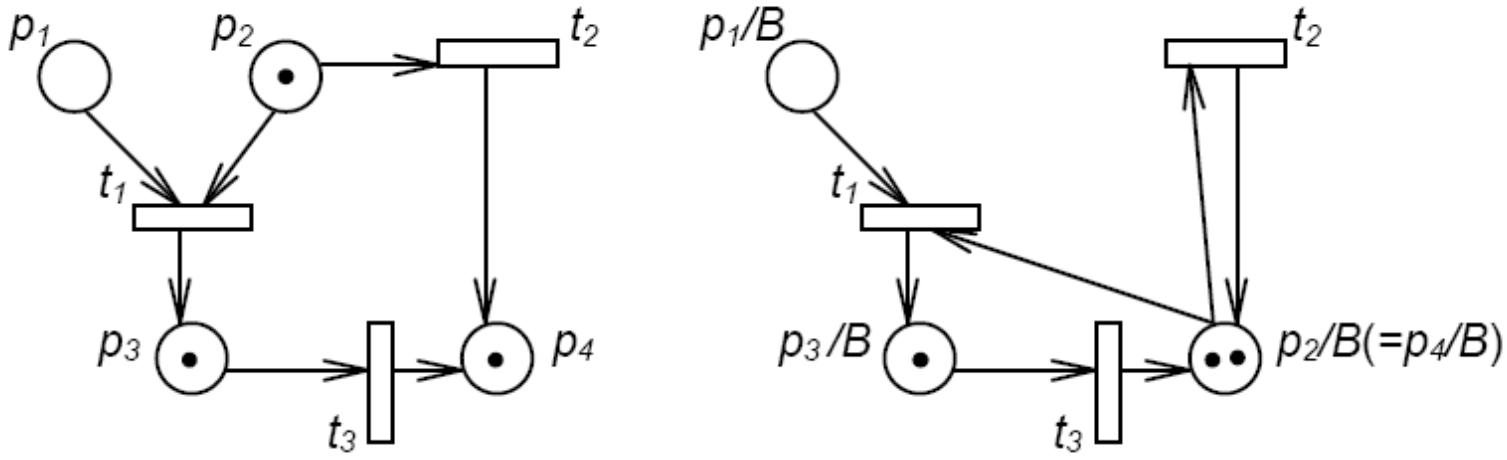


We fused places p_2 and p_4 here.

Place fusion

Fusing places only adds new behaviour.

If $M \xrightarrow{t} M'$ in N, then $M/B \xrightarrow{t} M'/B$ in N/B.



Place replacement

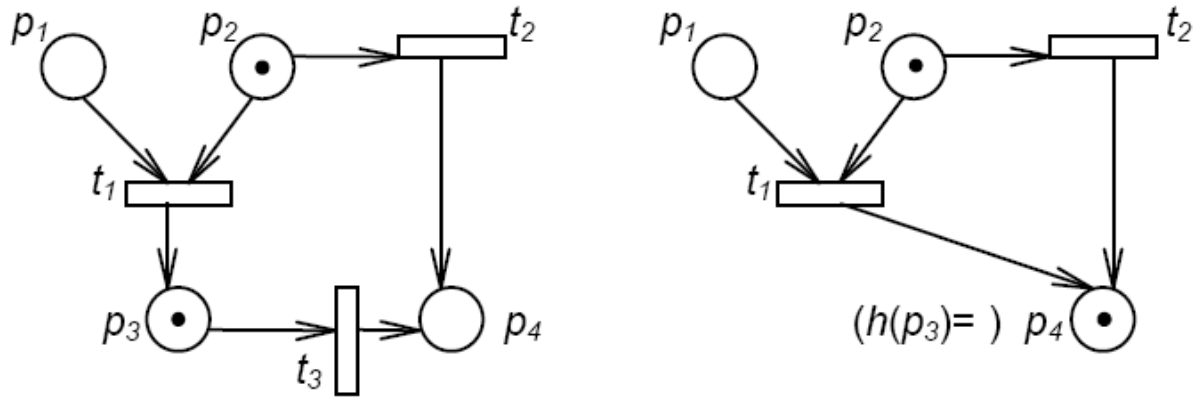
Let $h : P \to P$ be a projection, i.e. $\forall p \in P : h(h(p)) = h(p)$.
Place p can be replaced by $h(p)$ if $h(p) \neq p$.
The resulting net is denoted $h(N)$.

$h(N)$ is obtained by redirecting all output edges of transitions to place projections:
if $t^\bullet = M = \{p_1, \ldots, p_k\}$ in N, $h(M) = \{h(p_1), \ldots, h(p_k)\}$ in $h(N)$.

All places not in $h(P)$ are removed, all transitions lost one or more input places are removed.

A marking M in N yields a marking $m = h(M)$ in $h(N)$. m and M have the same number of tokens.

Place replacement



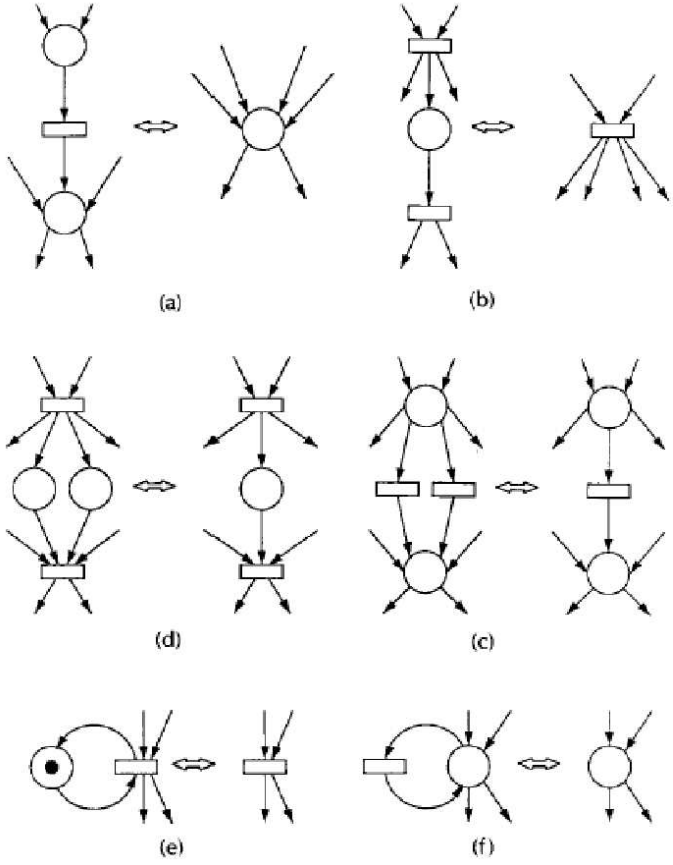
We replaced place p_3 by place p_4 here; t_3 is removed since it is an output transition of p_3.

Replacing places modify the behaviour in the following way:

If $m \xrightarrow{t} m'$ in $h(N)$, then $m \xrightarrow{t} M'$ in N for some M' s.t. $h(M') = m'$.

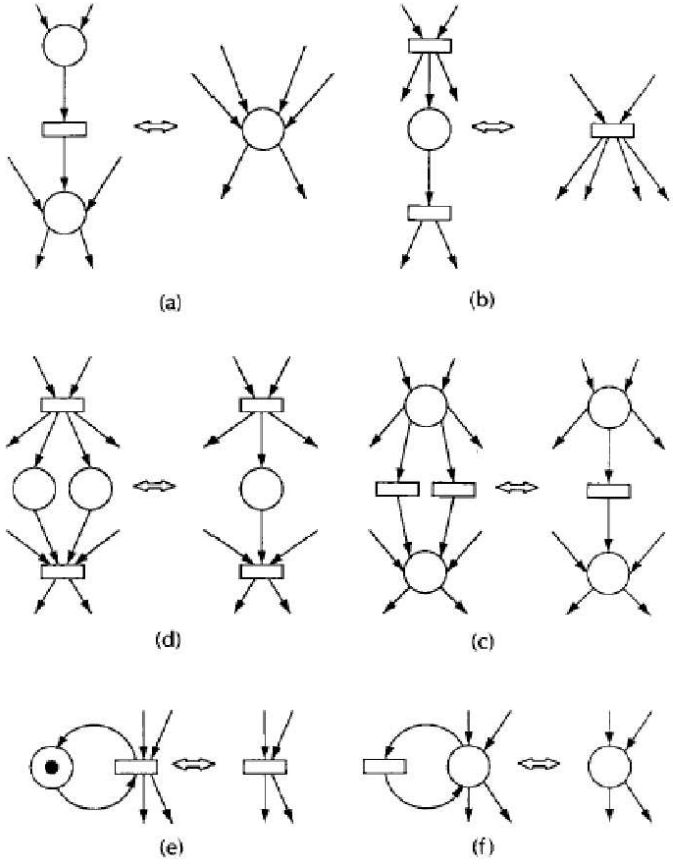
Petri net reduction techniques

Goal: to preserve such Petri net properties as liveness, safeness and boundedness.
The simplest transformations: (see [Murata1989])

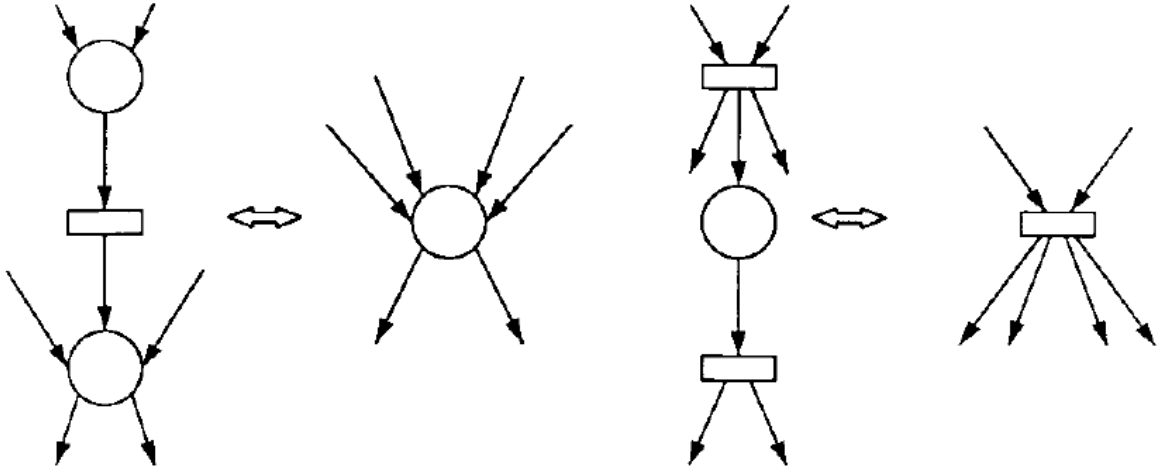


Petri net reduction techniques

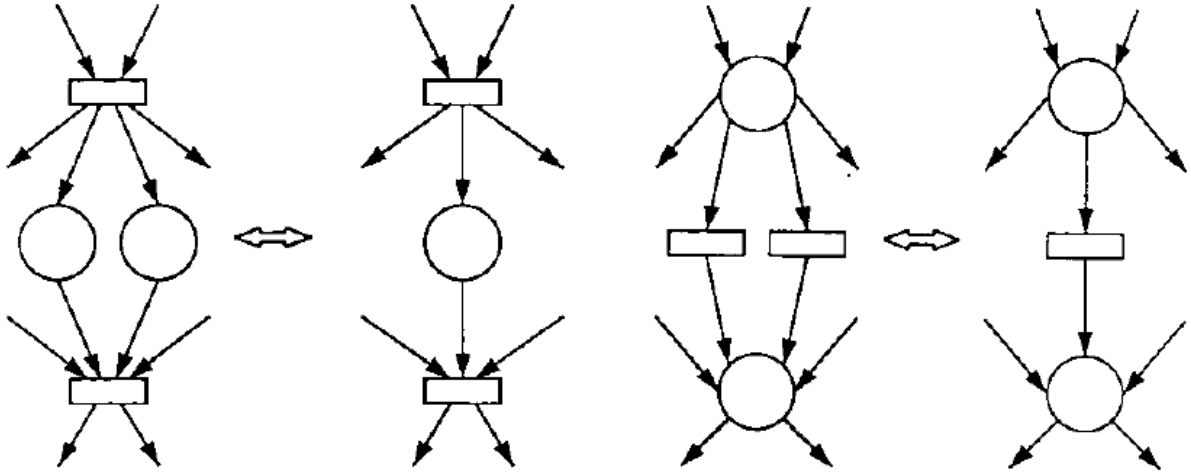
Goal: to preserve such Petri net properties as liveness, safeness and boundedness.
The simplest transformations: (see [Murata1989])



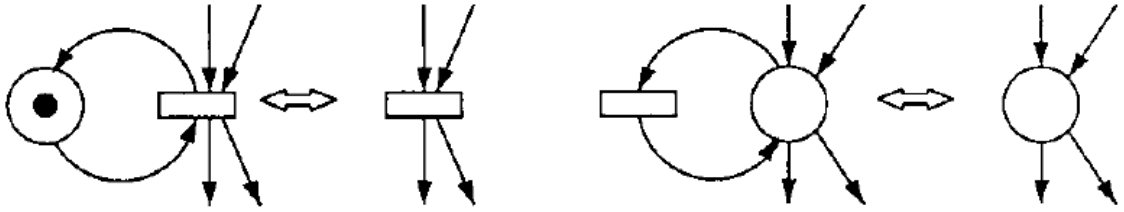
Fusion of series places/transitions



Fusion of parallel places/transitions



Elimination of self-loop places/trans.



Homework

Prove that the Petri net given in Fig. 5.1 (the upper net), p.90 of [Desel, Esparza] is live and bounded by applying reduction techniques preserving liveness, boundedness and safeness.