Universität Rostock

Traditio et Innovatio

# Petri Net (versus) State Spaces

Karsten Wolf

# My experience with state spaces

-INA     Integrated Net Analyzer

-LoLA   A Low Level Analyzer

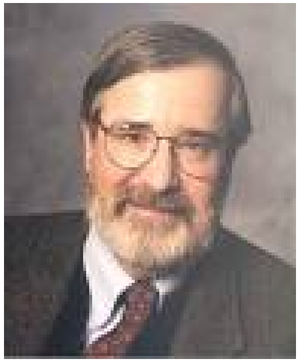
-The service-technology.org tool family

Case studies and applications:

-Finding hazards in a GALS wrapper
-Integration into Pathway Logic Assistent
-Soundness check for 700+ industrial business process models in (avg) 2 msec
-Verification of web service choreographies
-Verification of parameterized Boolean programs
-Solving AI planning challenges
-Integration into BP related tools like ProM, Oryx
-Integration into model checking platforms (MC Kit, PEP, CPN-AMI,…)
-….To be continued

Universität Rostock Traditio et Innovatio

Why state spaces?



Why Petri nets?

Verification based on state space

Universität Rostock Traditio et Innovatio

Why state spaces?

-Consider asynchronously communicating components rather than global state changes

-Consider causality of events rather than their ordering in time!

# Petri net principles

Monotonicity of firing

Presence or absence of ressources rather than reading / writing variables

Linearity of firing rule

-Consider asynchronously communicating components rather than global state changes

Locality

Partially ordered event structures

-Consider causality of events rather than their ordering in time!

# Petri net specific verification

Monotonicity of firing

Coverability graphs
Siphons / traps

Linearity of firing rule

invariants

Locality

Net reduction

Partially ordered
event structures

Branching prefixes

# State space generation

1. Checking enabledness

2. Firing a transition

3. Backtracking

4. Managing the visited states

# State space generation

1. Checking enabledness

Monotonicity

After firing, only check:

previously enabled transitions which have lost tokens

previously disabled transitions which have gained tokens

... managed through explicitly stored lists

… typical: reduction from linear to constant time

Locality

2. Firing a transition

3. Backtracking

4. Managing the visited states

# State space generation

1. Checking enabledness

2. Firing a transition

Marking changed via list of pre-, list of post-places
- effort does not depend on size of net
- Typically: constant effort


3. Backtracking

4. Managing the visited states

# State space generation

1. Checking enabledness

2. Firing a transition

3. Backtracking

   In depth-first search: fire transition backwards

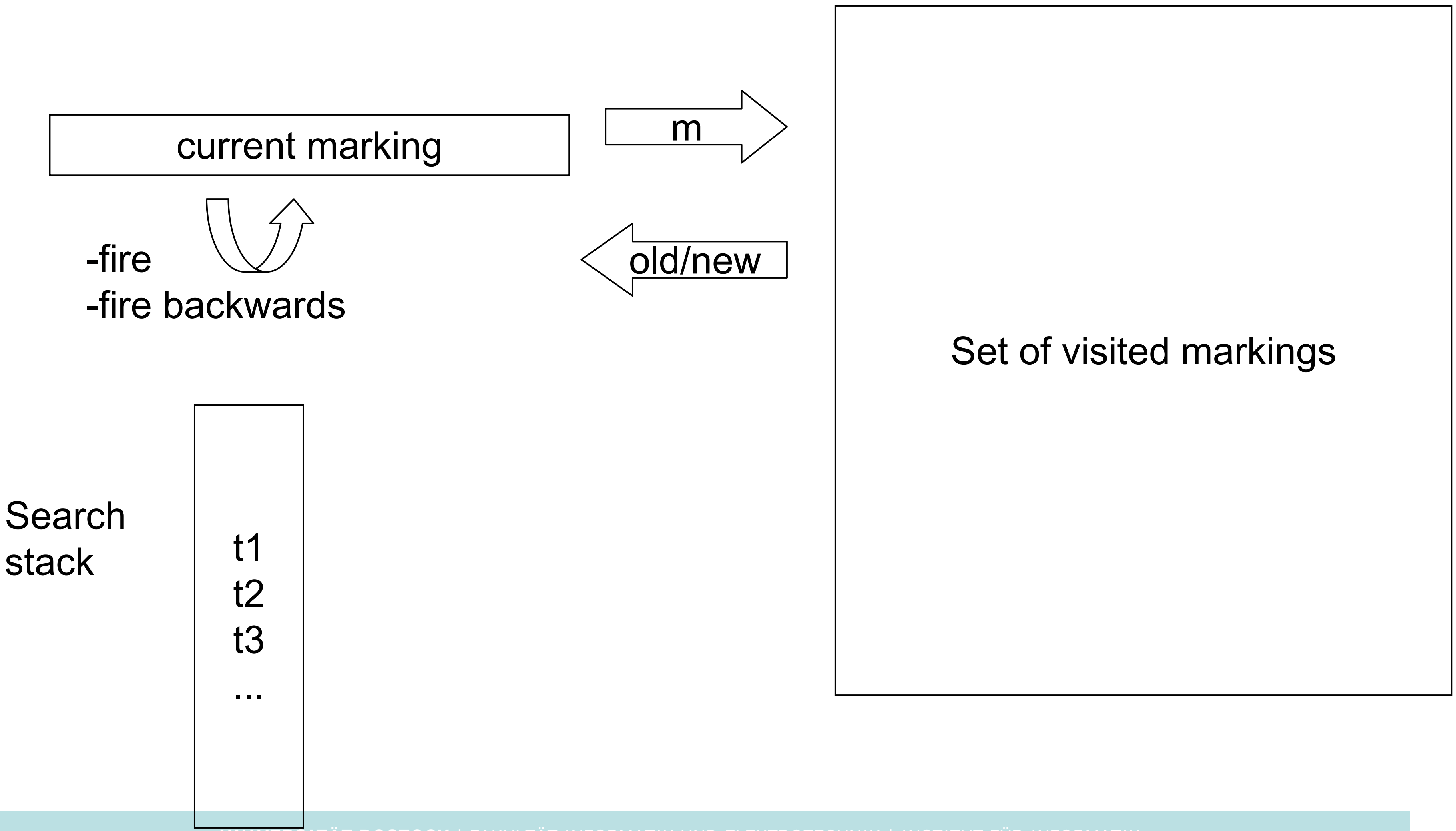
   In breadth-first search: implemented as incremental depth-first search

Linearity

Locality

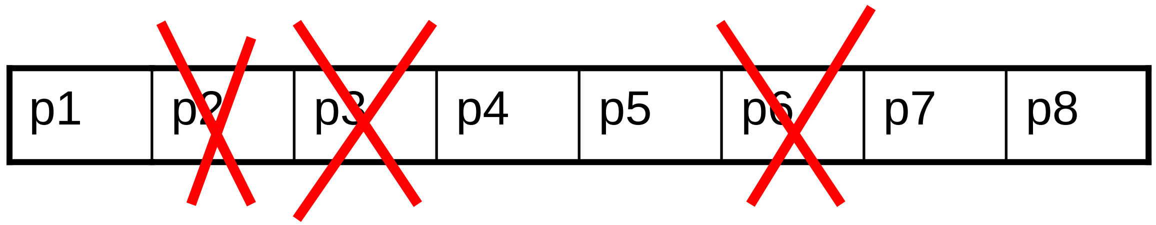
4. Managing the visited states

# Consequence: „write-only“ storage of markings



# 4. Managing the visited states

only performed actions: search, insert





a1 p1 + a2 p2 + a3 p3 = const.

b2 p2 + b4 p4 + b6 p6 = const.

c3 b3 + c7 p7 + c8 p8 = const.

} Place invariants

30-60% less memory
preprocessing <1sec
run time gain: 30-60%

# Reduction techniques

1. Linear Algebra
2. The Sweep-Line Method
3. Symmetries
4. Stubborn Sets

Universität Rostock Traditio et Innovatio

# 1. Linear algebra

- The invariant calculus
  - originally invented for *replacing* state spaces
  - in LoLA: used for *optimizing* state spaces

Already seen: place invariants

Transition invariant: firing vector of a potential cycle

# Transition invariants

Linearity

for termination sufficient: store one state per cycle of occurrence graph

implementation in LoLA:

transition invariants

- set of transitions that occur in every cycle
- store states where those transitions enabled

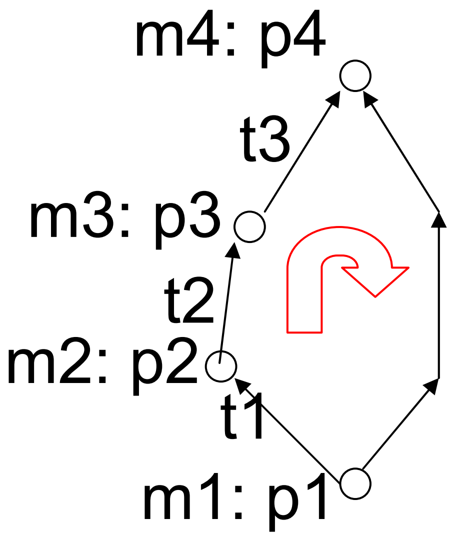
saves space, if applied in connection with stubborn sets, costs time

# 2. The sweep-line method

- Relies on progress measure

LoLA computes measure automatically:


$p2=p1+\Delta t1$
$p3 = p2+\Delta t2$
...




transition invariant

# 3. The symmetry method


LoLA: A symmetry = a graph automorphism of the PT-Net

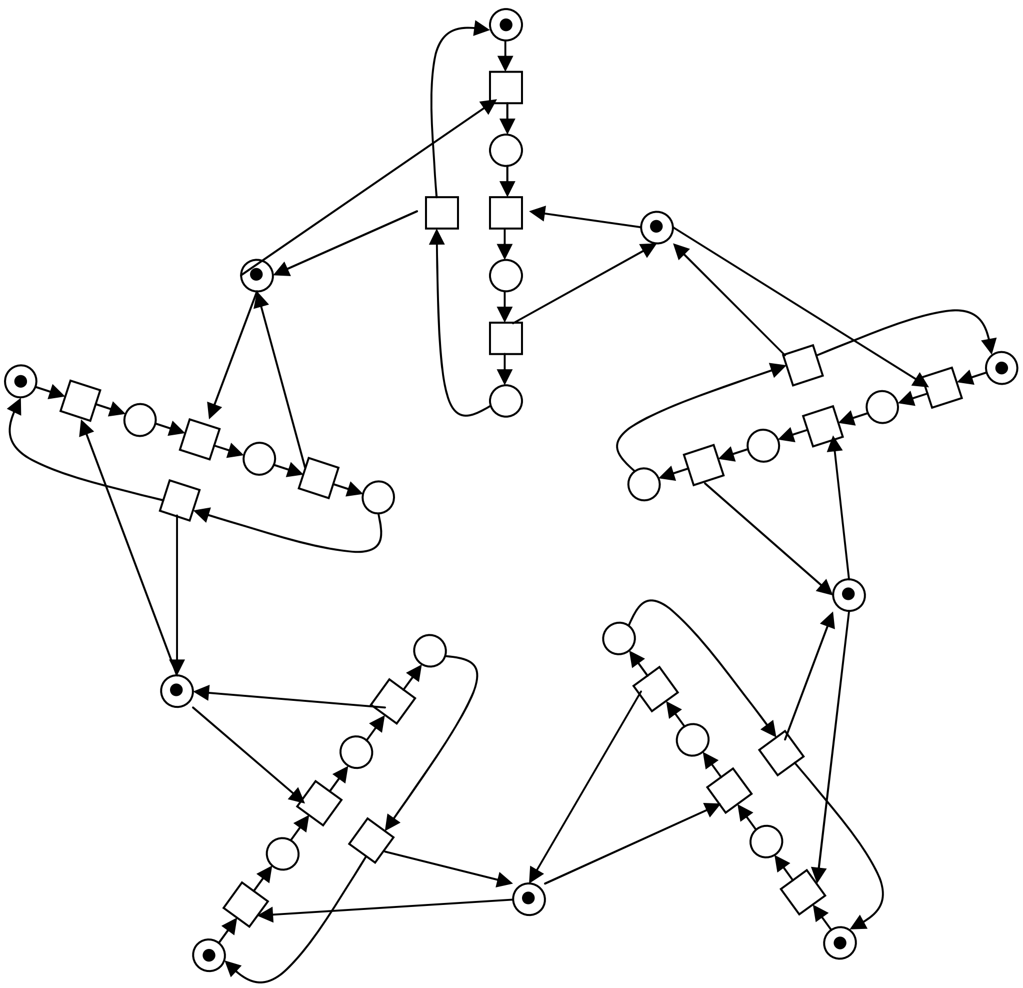
All graph automorphisms = a group (up to exponentially many members)

- stored in LoLA: polynomial generating set

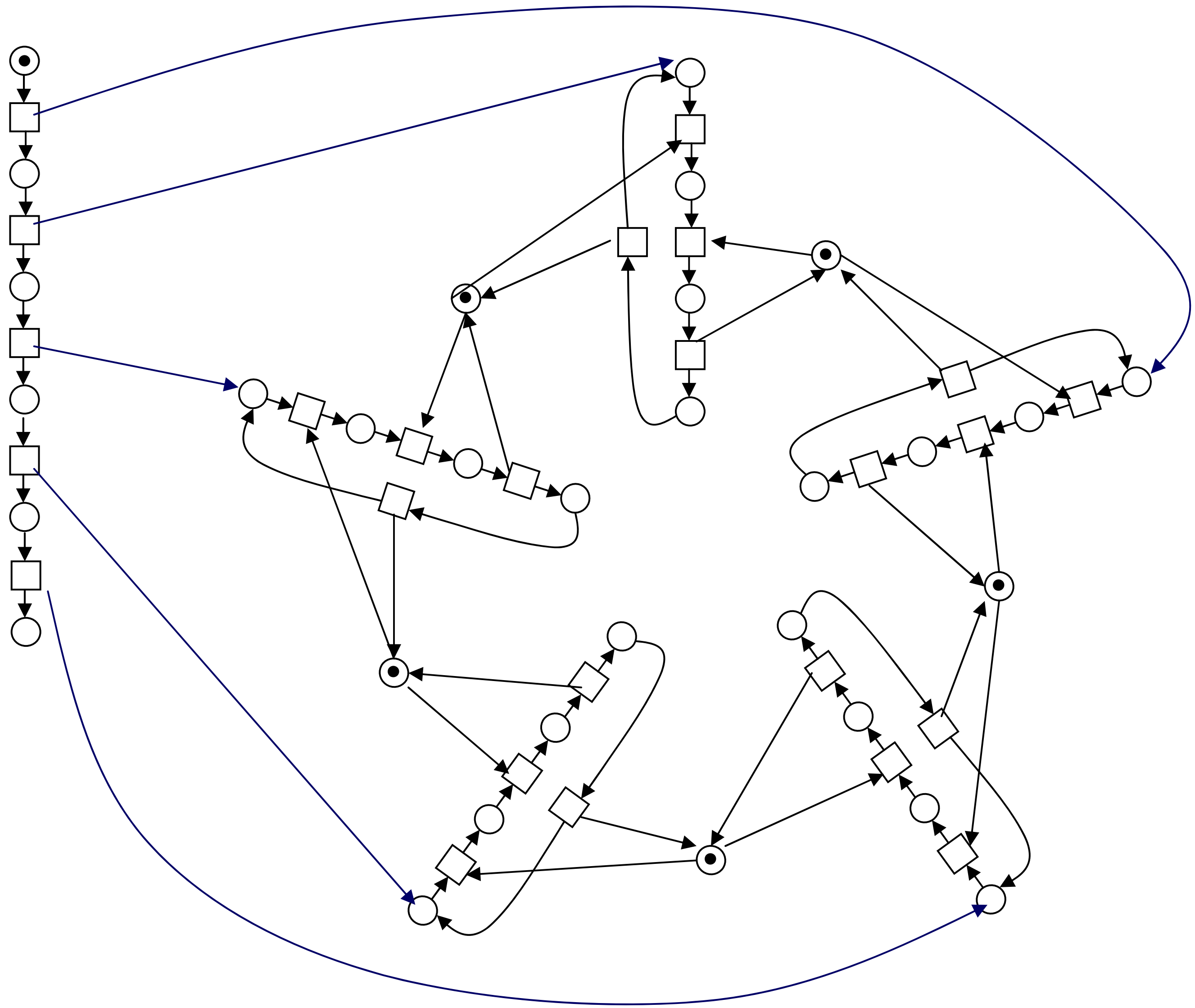
A marking class: all markings that can be transformed into each other by a symmetry

- executed in LoLA: polynomial time approximation

# Example



# … as derived from a program



# 4. Stubborn set method

- Dedicated method for each supported property

traditional LTL-preserving method:

- one enabled transition
- the basic stubborness principal
- only *invisible* transitions
- at least once, on every cycle, all enabled transitions

LoLA:

- can avoid some of the criteria, depending on property

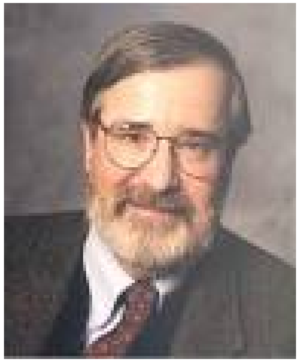
Locality

PN Tradition

# Conclusion

Why state spaces?

That‘s why



Why Petri nets?

Further reading:

- Tools: www.service-technology.org
- Group / Papers: www.informatik.uni-rostock.de/tpp/