Structural counter abstraction

Proving fair-termination of depth bounded systems

Kshitij Bansal[1]

with Eric Koskinen[1], Thomas Wies[1], Damien Zufferey[2]

[1]New York University [2]IST Austria

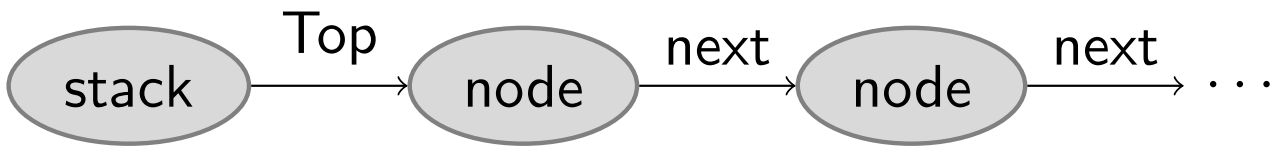
March 18, 2013

TACAS, Rome, Italy

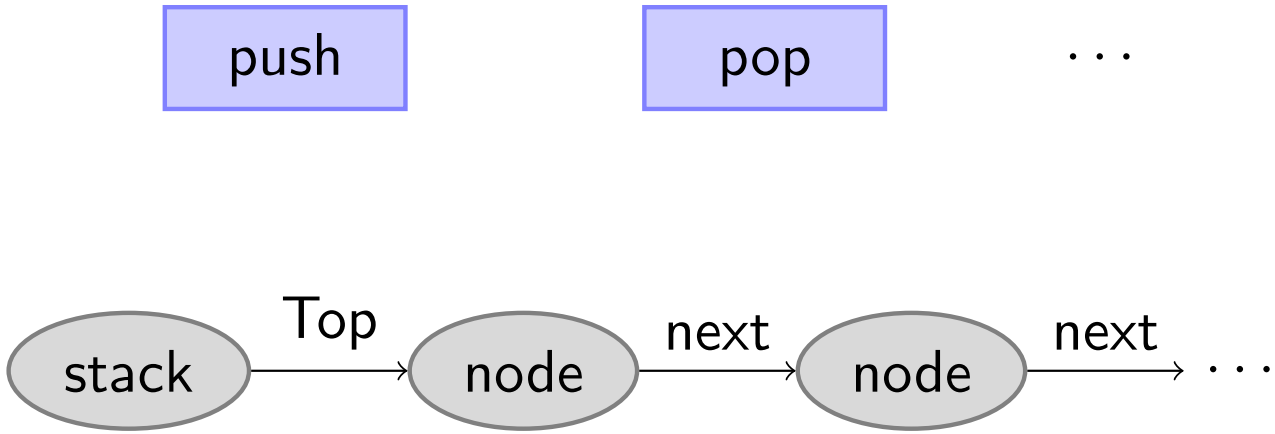
Introduction

- **Model: Depth-bounded systems**
  Graph-rewrite based transition systems, which can be used to model concurrent heap-manipulating algorithms, as well as distributed systems.
- **Problem: Fair-termination problem**
  Fairness: If a transition is continuously enabled after some point, it is taken infinitely often.
- Application: Proving progress properties of concurrent and distributed systems.

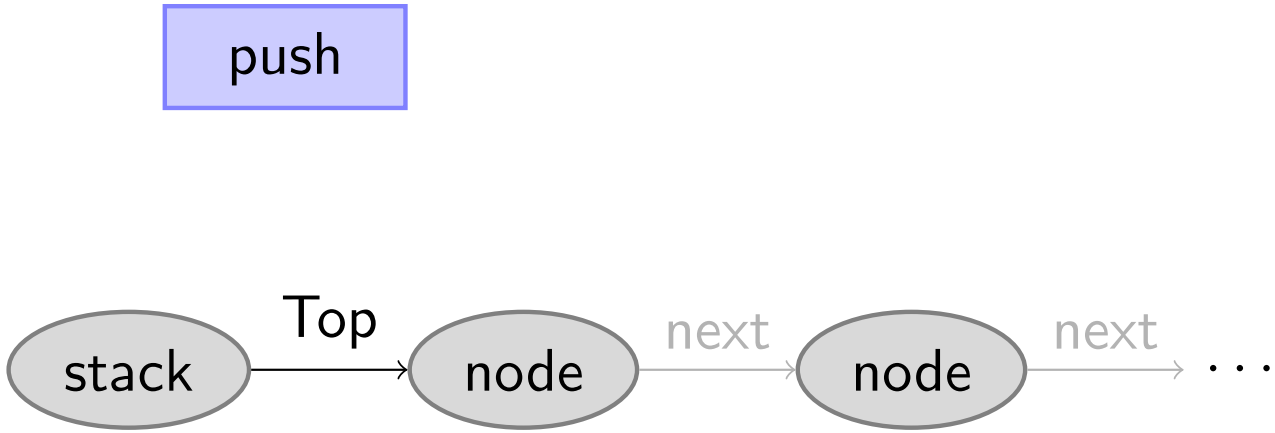
Treiber stack



Treiber stack

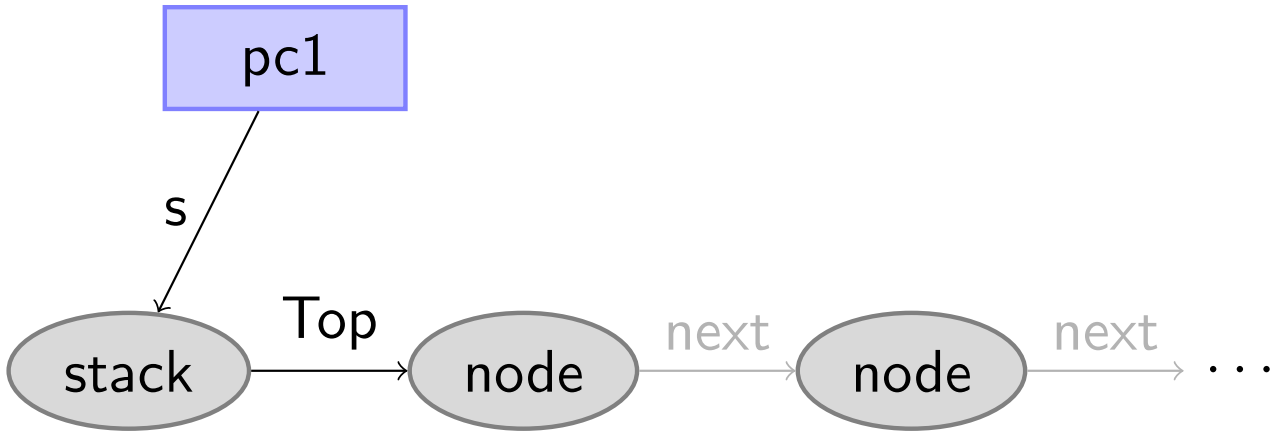


Treiber stack



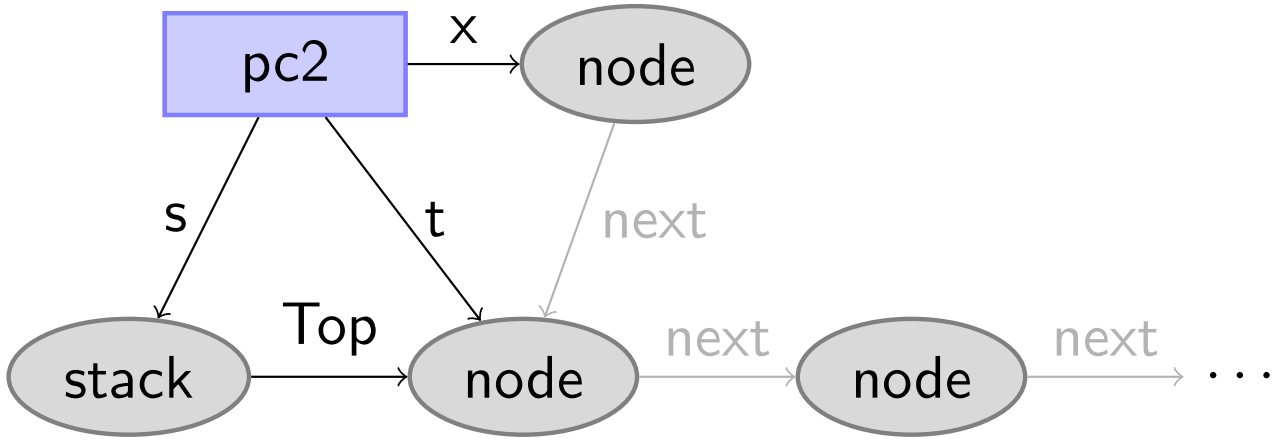

```
        push(s, data):
          do {
pc1:        t = s->top; x = new node(t, data);
pc2:      }while( !CAS(s->top, t, x) );
```

Treiber stack



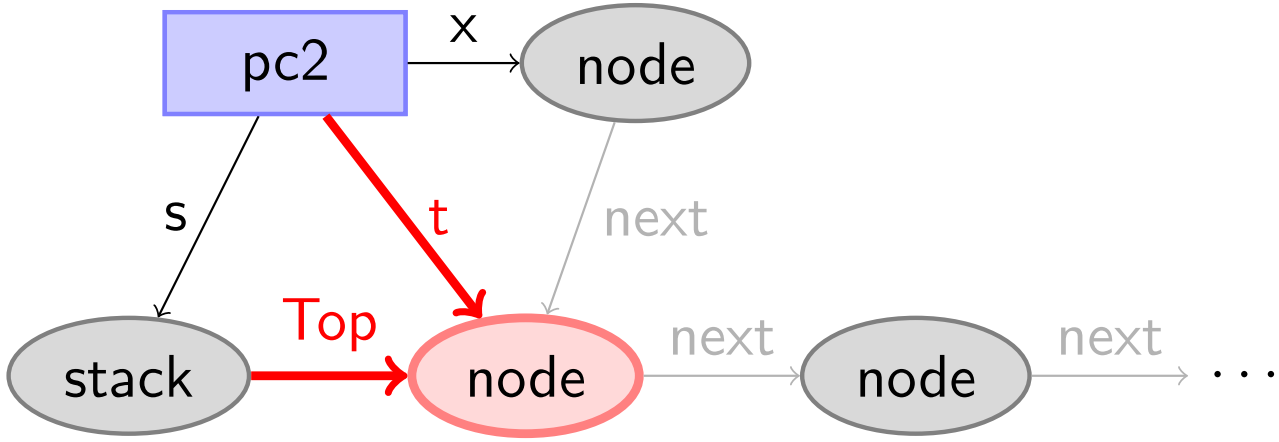

```
        push(s, data):
          do {
pc1:        t = s->top; x = new node(t, data);
pc2:      }while( !CAS(s->top, t, x) );
```

Treiber stack



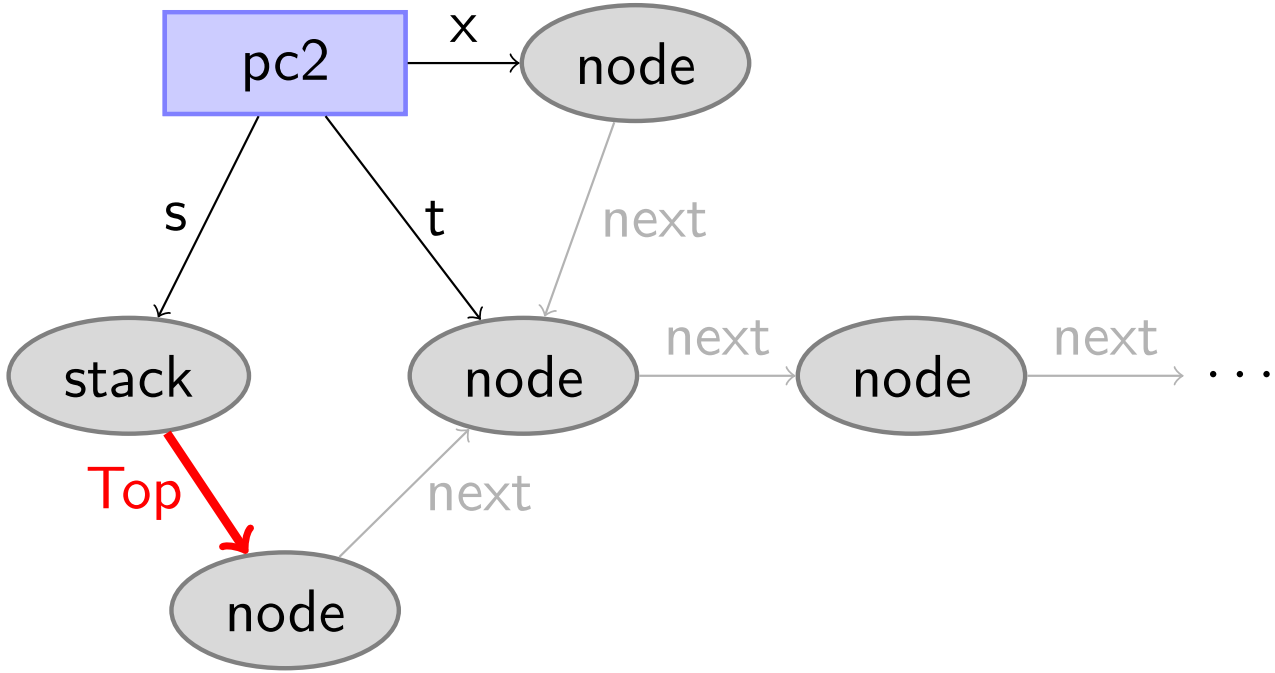

```
        push(s, data):
          do {
pc1:        t = s->top; x = new node(t, data);
pc2:      }while( !CAS(s->top, t, x) );
```

Treiber stack



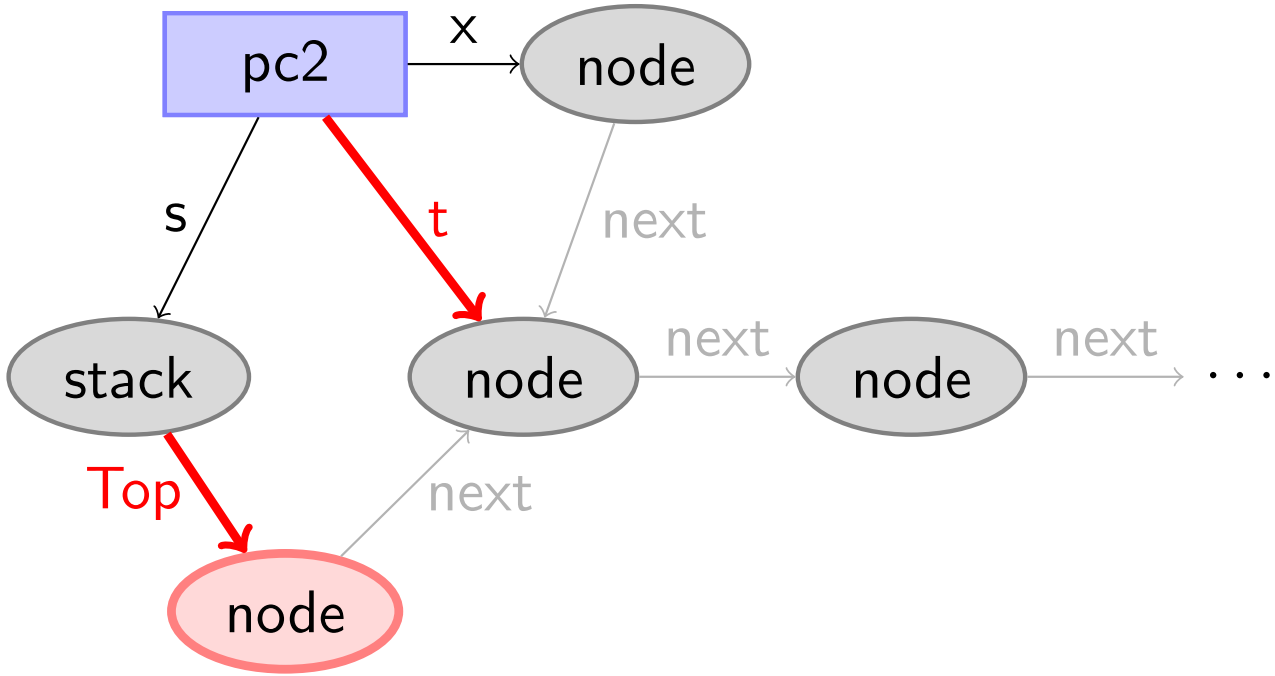

```
        push(s, data):
          do {
pc1:        t = s->top; x = new node(t, data);
pc2:      }while( !CAS(s->top, t, x) );
```

Treiber stack



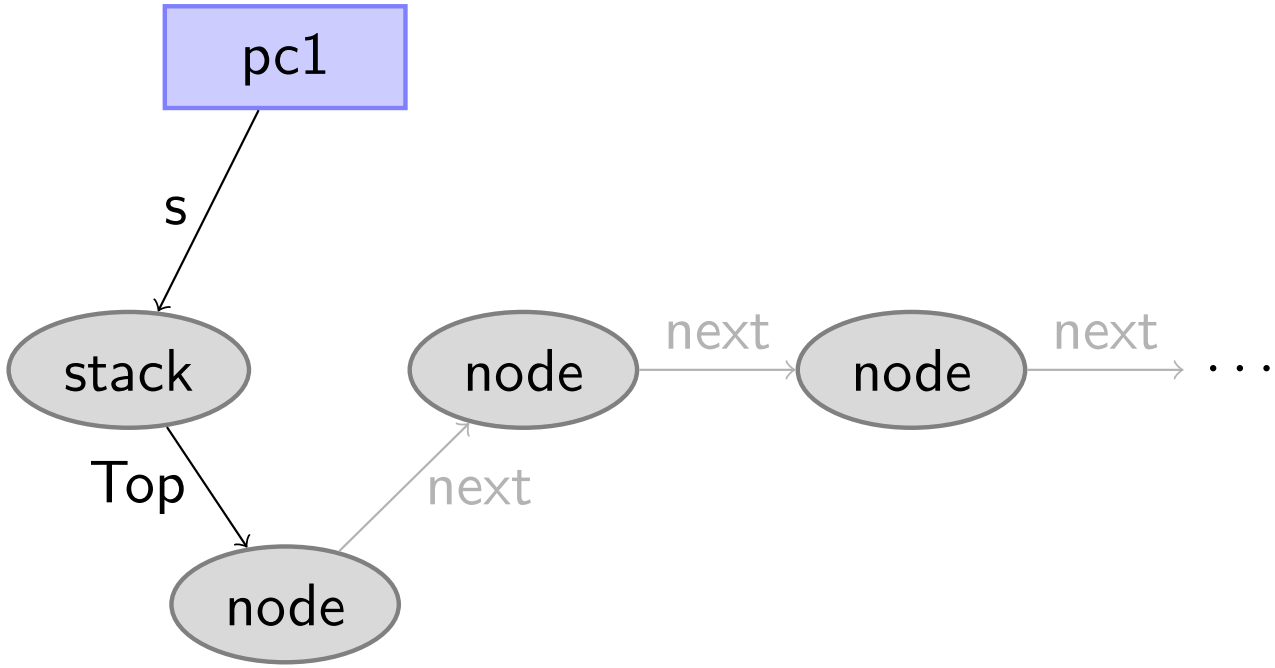

```
        push(s, data):
          do {
pc1:        t = s->top; x = new node(t, data);
pc2:      }while( !CAS(s->top, t, x) );
```

Treiber stack




```
        push(s, data):
          do {
pc1:        t = s->top; x = new node(t, data);
pc2:      }while( !CAS(s->top, t, x) );
```

Treiber stack




```
        push(s, data):
          do {
pc1:        t = s->top; x = new node(t, data);
pc2:      }while( !CAS(s->top, t, x) );
```

Lock freedom as fair termination

- Treiber stack is lock-free.
  - guarantees global progress: some thread will finish
  - individual threads might starve
- Reduced to termination problem where arbitrarily many but finite number of threads are present.
- A transition which spawns processes at will, along with a fairness constraint can be used to encode this.

Lock freedom as fair termination

- Treiber stack is lock-free.
  - guarantees global progress: some thread will finish
  - individual threads might starve
- Reduced to termination problem where arbitrarily many but finite number of threads are present.
- A transition which spawns processes at will, along with a fairness constraint can be used to encode this.

- **Challenge:** Unbounded number of heap objects and thread objects.

Contribution

- Work with symbolic graphs which can model structures that arise commonly in these systems, and required to be tracked to prove termination.
- **Contribution:** We introduce a counter abstraction derived from these, thus called **structural counter abstraction**. It is sufficiently refined to be able to prove progress properties like lock-freedom of Treiber stack.

Related work

Counter abstraction for concurrent systems

- A. Pnueli, J. Xu, and L. D. Zuck. *Liveness with (0, 1, ∞)-counter abstraction.* In *CAV*, 2002.
- G. Basler, M. Mazzucchi, T. Wahl, and D. Kroening. *Symbolic counter abstraction for concurrent software.* In *CAV*, 2009.

Related work

Counter abstraction for concurrent systems

- A. Pnueli, J. Xu, and L. D. Zuck. *Liveness with (0, 1, ∞)-counter abstraction.* In *CAV*, 2002.
- G. Basler, M. Mazzucchi, T. Wahl, and D. Kroening. *Symbolic counter abstraction for concurrent software.* In *CAV*, 2009.

Graph-based analysis

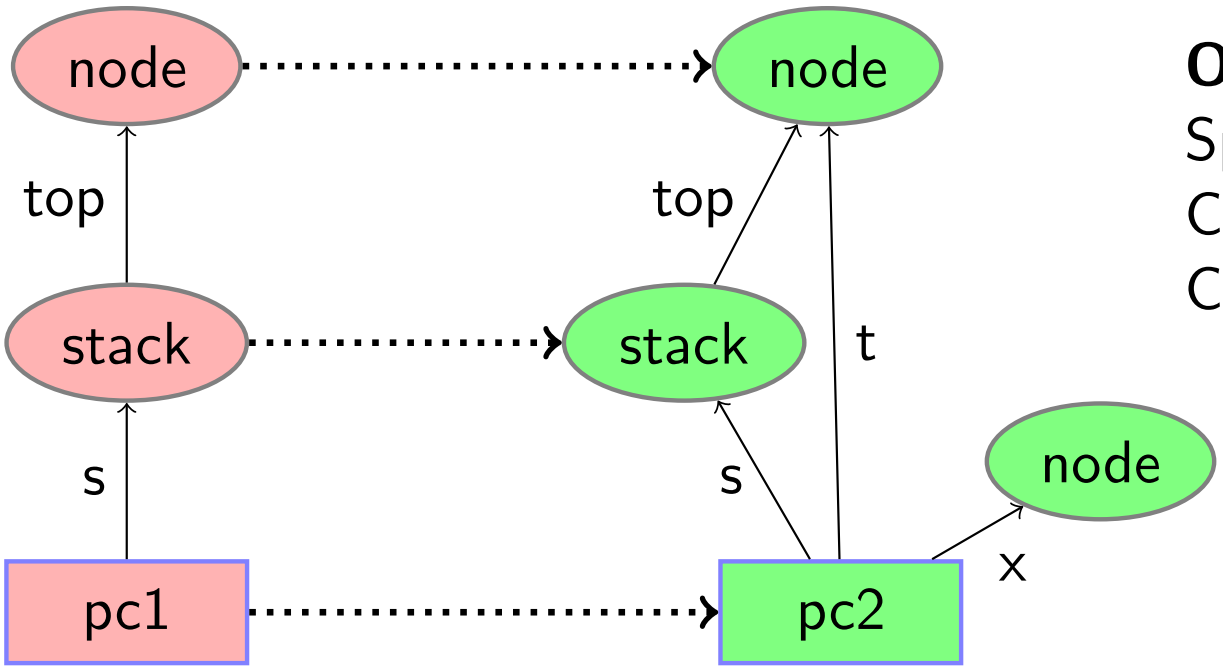
- J. Berdine, B. Cook, D. Distefano, and P. W. O'Hearn. *Automatic termination proofs for programs with shape-shifting heaps.* In *CAV*, 2006.
- S. Gulwani, T. Lev-Ami, and M. Sagiv. *A combination framework for tracking partition sizes.* In *POPL*, 2009.

Outline

Model

Graph Transformation Systems

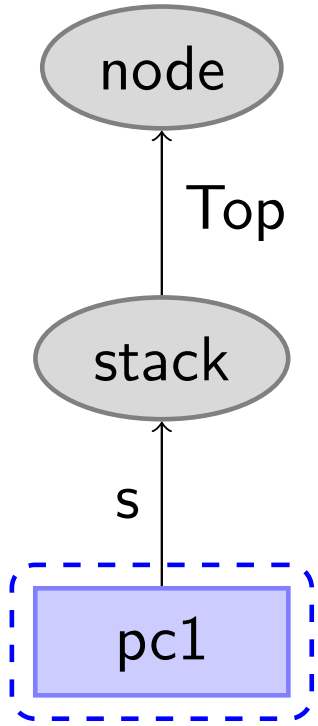
- States: graphs. In our case, symbolic graphs (on next slide).
- Rules: rewrite one subgraph with another.



Other rules:
Spawn
CAS succeed
CAS fail

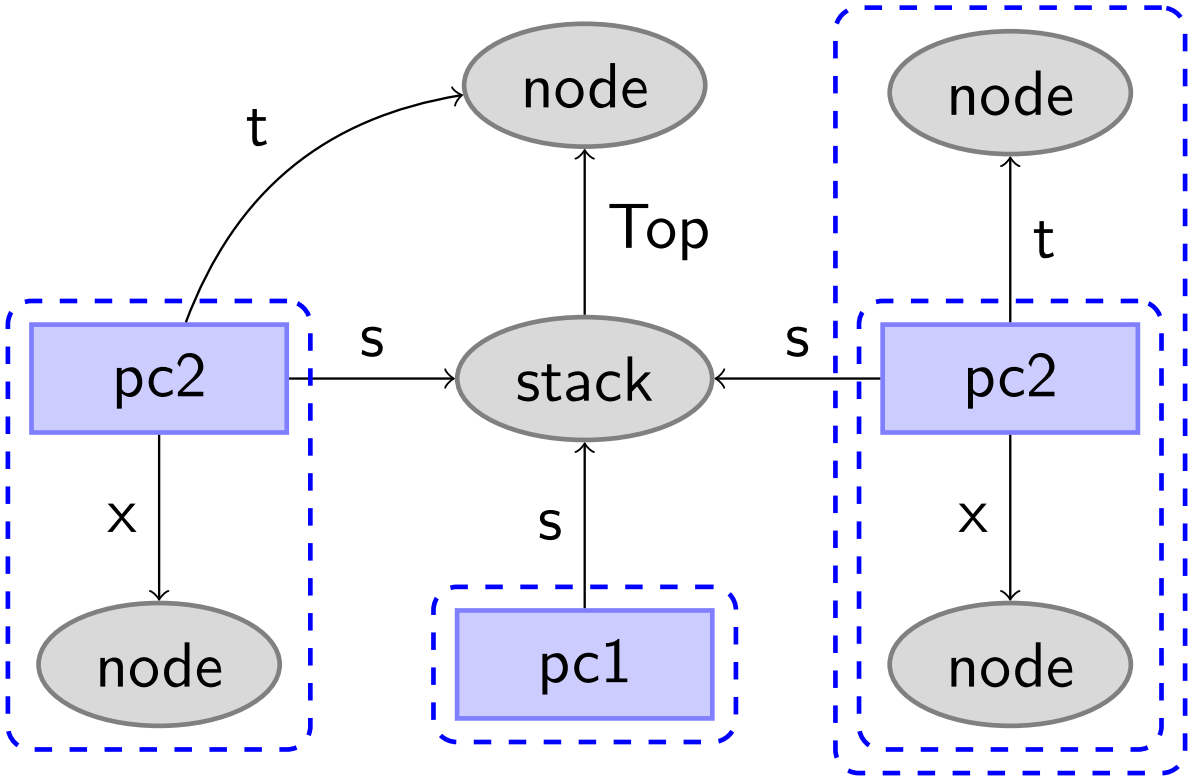
Prepare rule

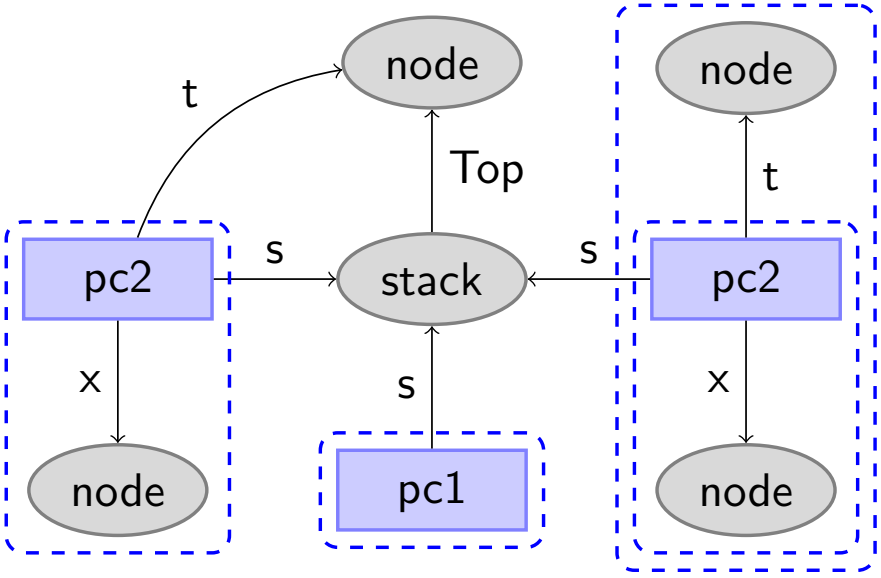
Nested graphs



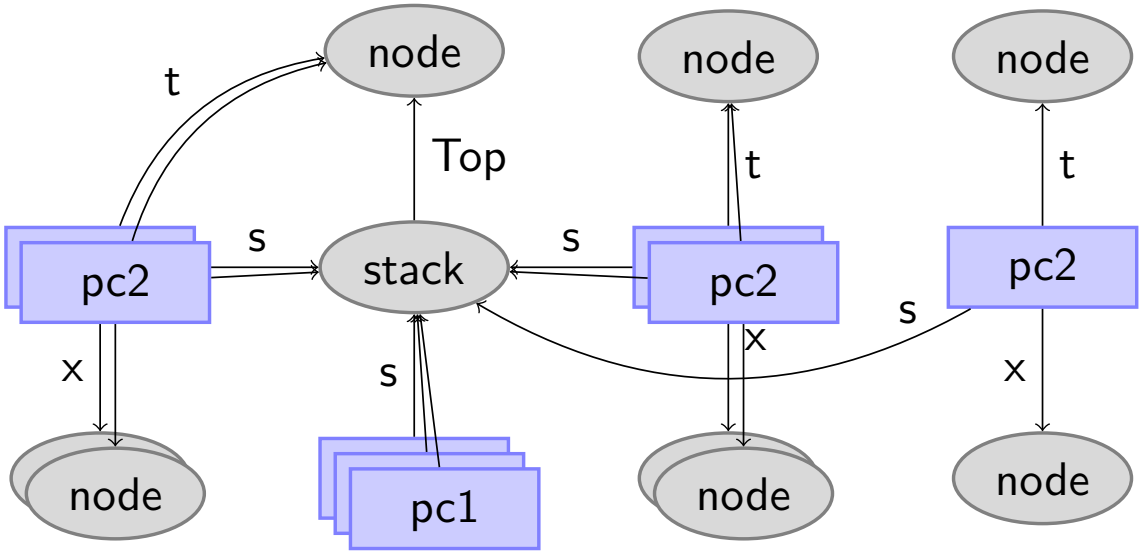
represent arbitrary number of copies.

Nested graphs

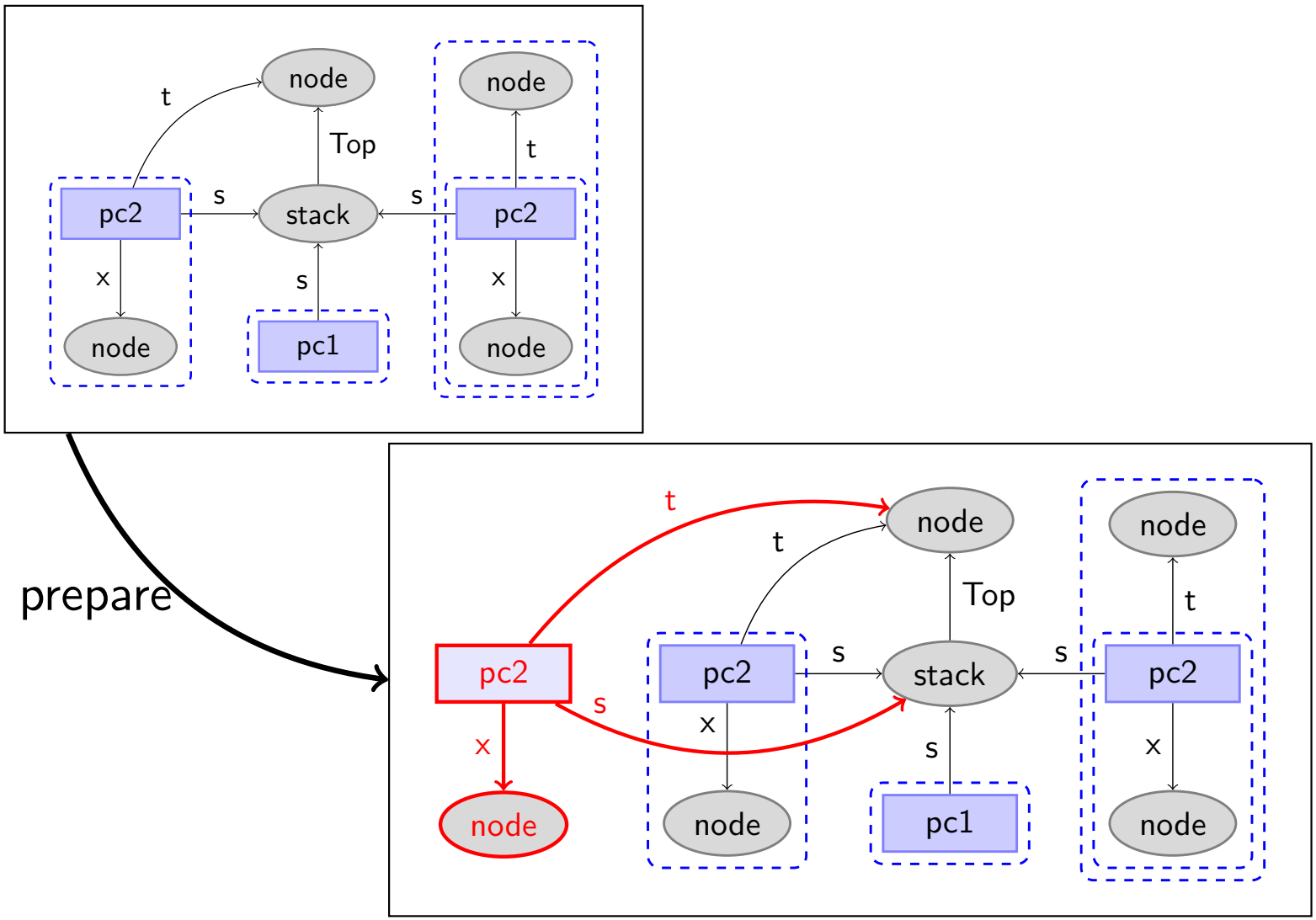




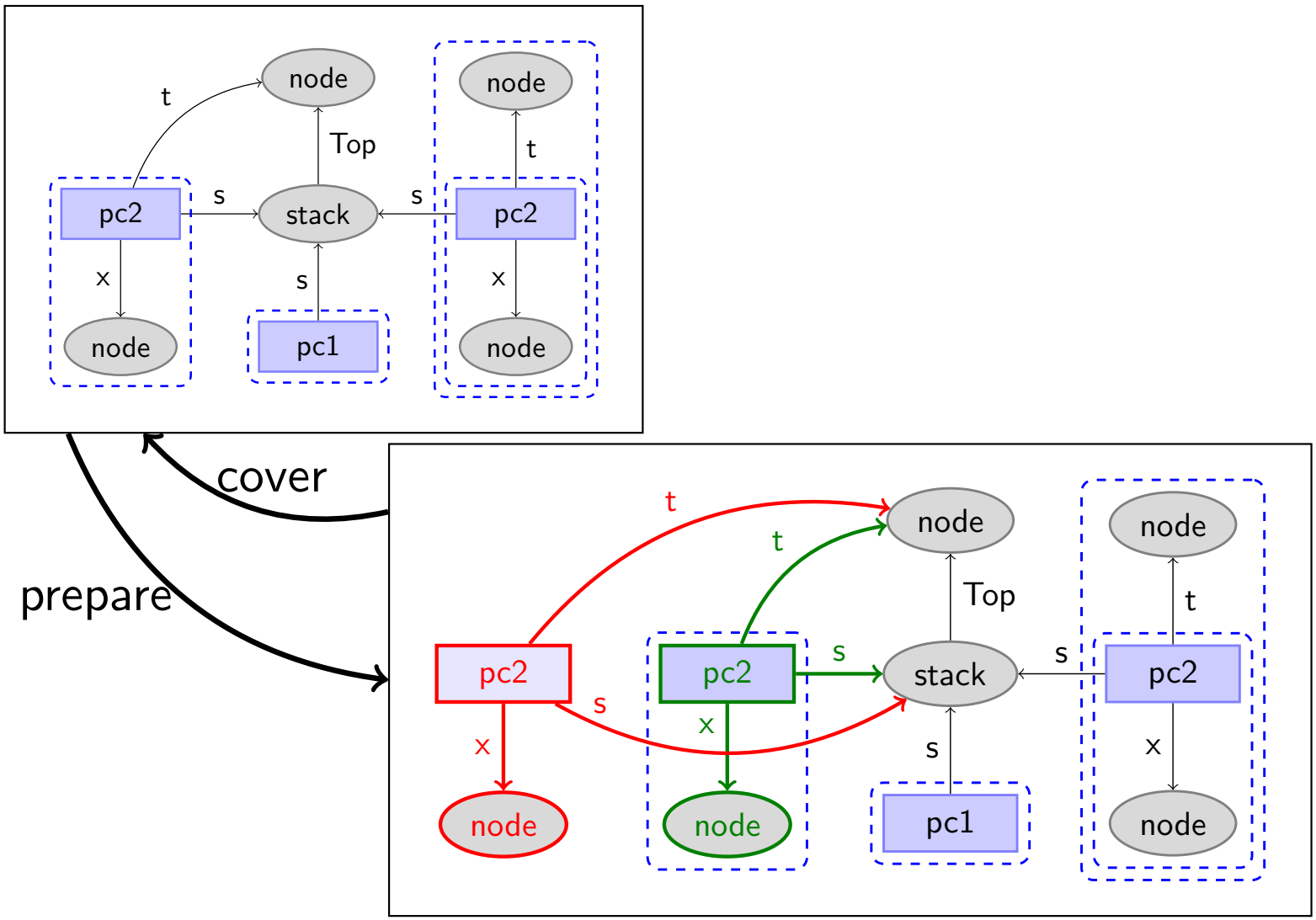
Nested subgraphs represent arbitrary number of copies of the subgraphs



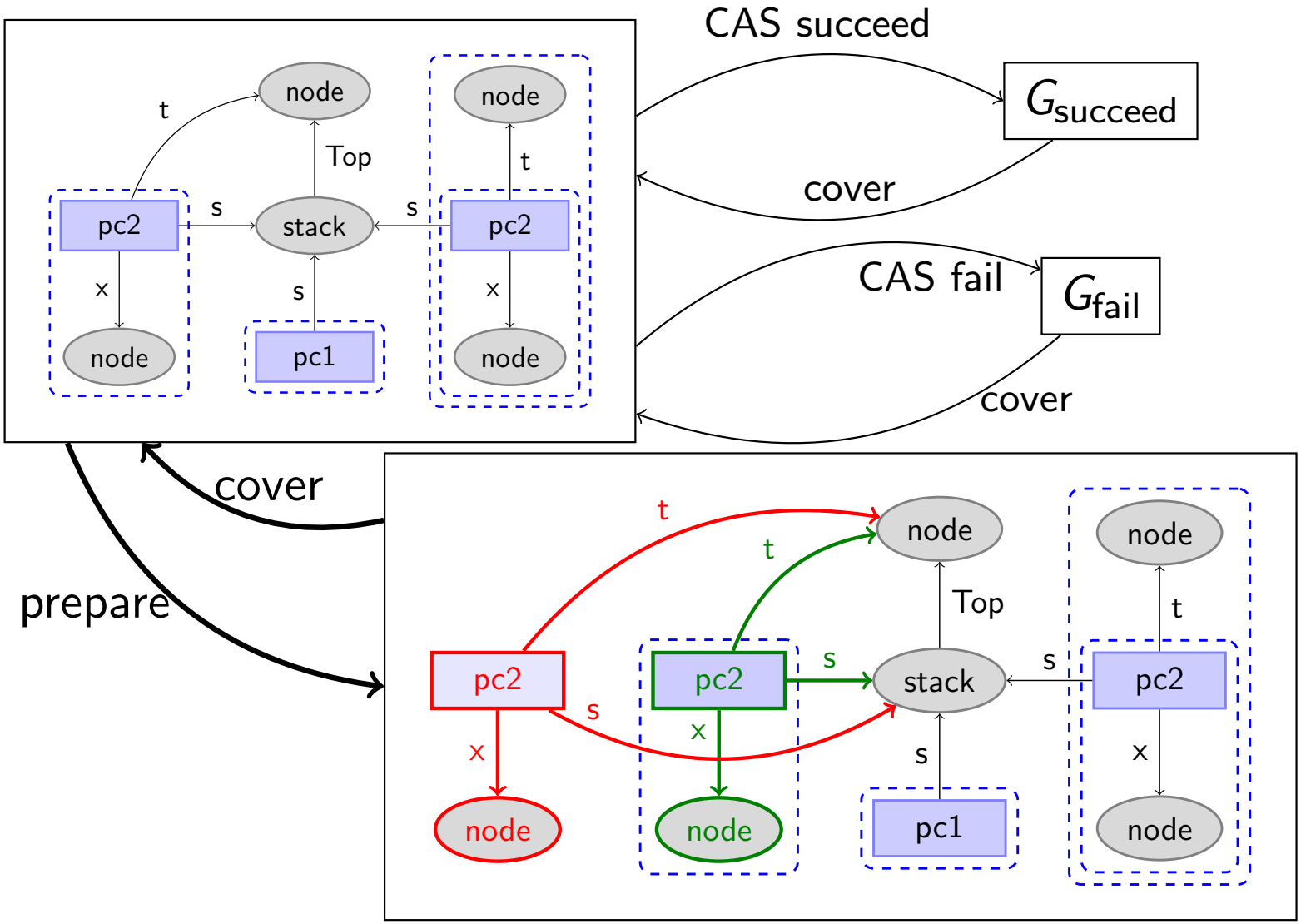
Inductive invariant



Inductive invariant



Inductive invariant

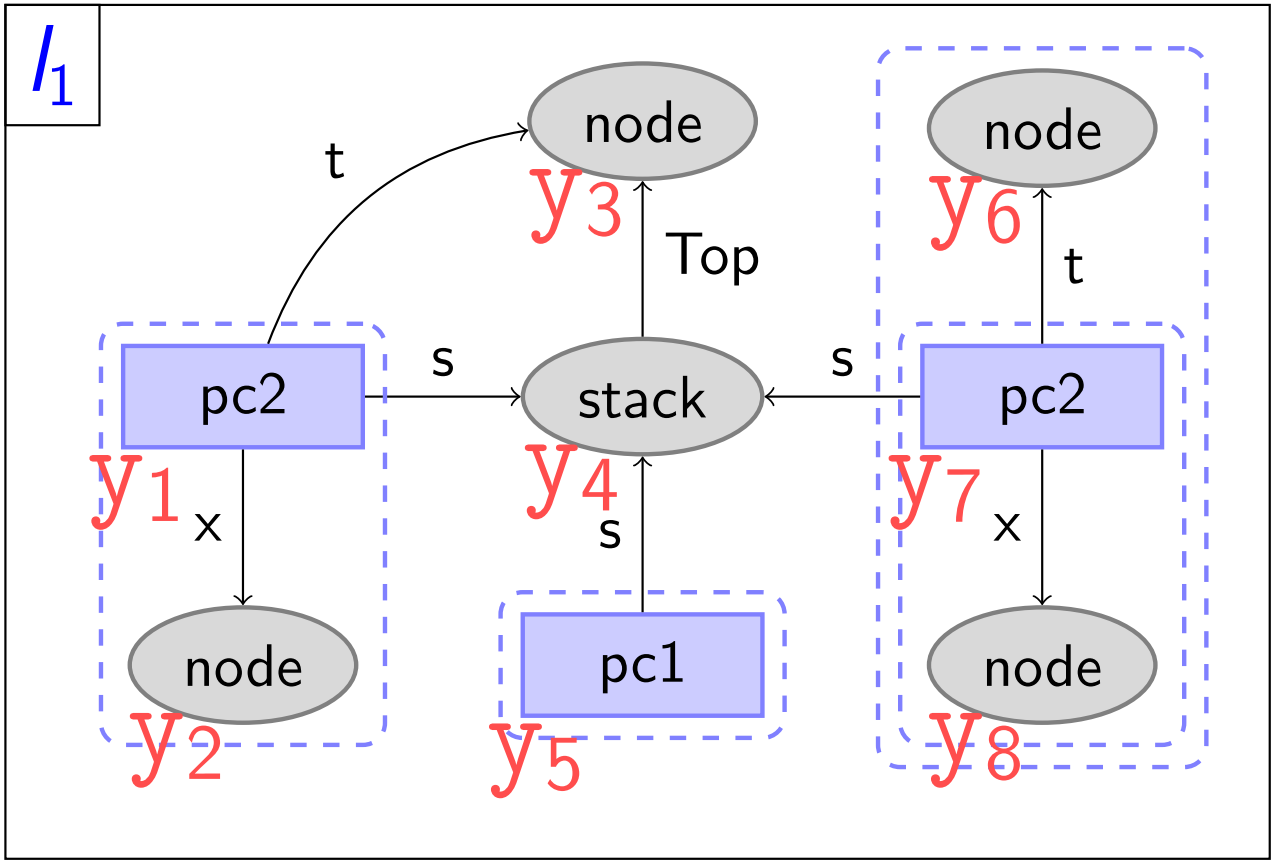


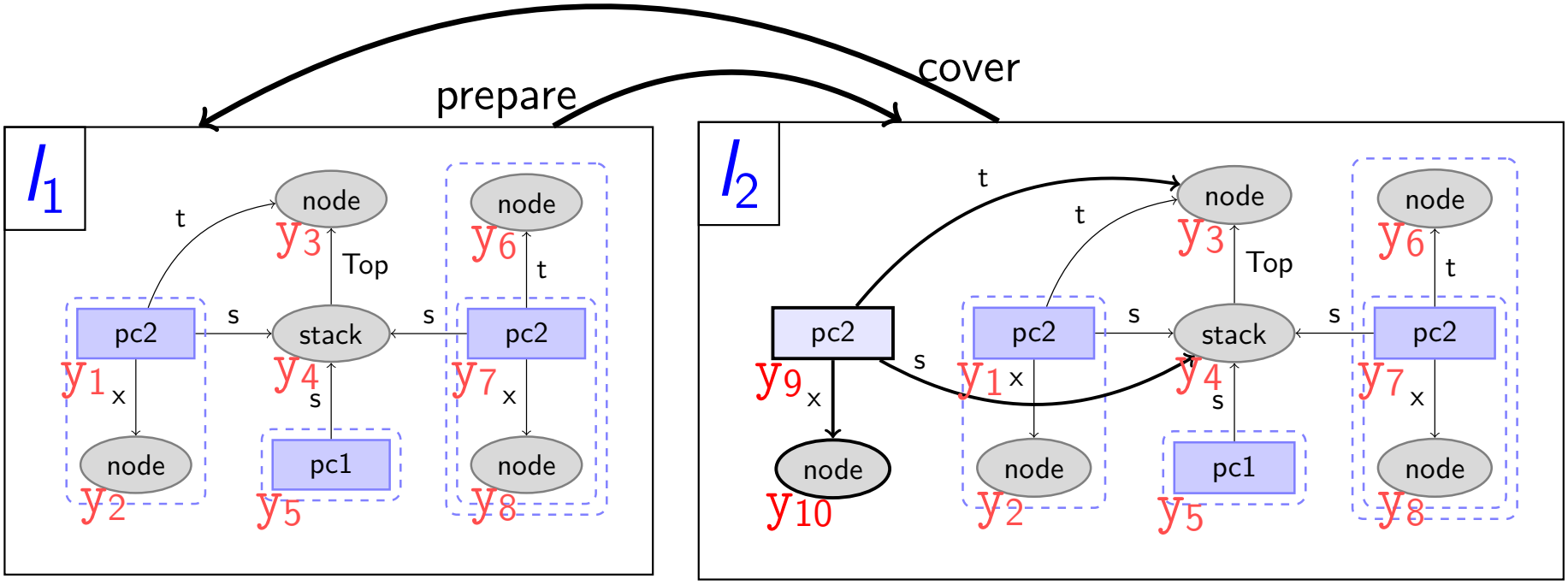
Structural counter abstraction

Input: Rewrite-rules,
Inductive invariant as nested graphs
Output: Counter system

Graph system	$\rightarrow$	**Counter system**
Nested graphs	$\rightarrow$	Control locations
Nodes in nested graphs	$\rightarrow$	Counters
Rule applications	$\rightarrow$	Counter updates

Soundness. If the graph transition system has a fair non-terminating run, then the counter system will have a fair non-terminating run.


l_1
node
t
y_3
Top
node
y_6
t
pc2
s
stack
s
pc2
y_1
x
y_4
s
y_7
x
node
pc1
node
y_2
y_5
y_8



prepare: $(l_1, \{\ y_9' = 1, y_{10}' = 1, y_5' = y_5 - 1,$ identity on rest $\}\ , l_2)$

cover: $(l_2, \{\ y_1' = y_1 + y_9, y_9' = 0, y_2' = y_{10} + y_2, y_{10}' = 0,$ identity on rest $\}\ , l_1)$

Computing Inductive Invariant

- Depth bounded systems: class of well-structured transition systems [Meyer, 2008]. It says if the length of the longest simple path is bounded, then system is well-structured with the ordering given by subgraph homomorphism.
- Analysis to compute over-approximation of set of reachable states of the WSTS [Ideal abstraction, Zufferey, Wies, Henzinger, 2012]. This overapproximation is a downward closed set, also inductive, given as finite union of states represented by the nested graphs.
- Many concurrent and distributed process can be modeled as depth-bounded processes for proving termination (Treiber stack without next, etc.)

Implementation

Input: Graph rewrite system.

1. Picasso[1] computes the inductive invariant as nested graphs[ZWH'12].
2. Picasso extended to compute the counter abstraction from the invariant[this work].
3. Counter program is fed to termination prover for counter systems, ARMC [Andrey Rybalchenko, Andreas Podelski].

We also use Z3 [Leonardo de Moura, Nikolaj Bjorner] and Princess[Philipp Rümmer] for variable elimination to optimize counter abstraction.

[1] `http://pub.ist.ac.at/~zufferey/picasso/`

Experimental Results

Example	#loc	#v	#t	$\widehat{\mathcal{I}}$	$\mathcal{N}$	ARMC	**Total**
Split/merge	4	3	9	1.5	6.8	0.1	8.4
Work stealing, 3 processors	4	4	20	1.7	13.1	0.2	15.0
Work stealing, parameterized	2	3	4	1.5	5.6	0.1	6.2
Compute server job queue	2	5	4	1.6	6.1	0.1	7.8
Chat room	5	34	80	9.8	61.3	5 min	6 min
Map reduce	6	10	15	2.0	8.8	0.2	11.0
Map reduce with failure	6	15	21	2.3	11.1	0.9	14.3
Treiber's stack (coarse-grained)	2	6	4	1.9	7.2	0.2	9.3
Treiber's stack (fine-grained)	3	14	13	2.7	14.2	1.2	17.1
Herlihy/Wing queue	3	16	25	3.8	24.9	6.5	34.2
Michael/Scott queue (dequeue only)	4	7	23	2.8	13.0	0.6	16.4
Michael/Scott queue (enqueue only)	7	15	53	3.8	43.7	7.6	55.1
Michael/Scott queue	9	31	224	25.0	265.0	3 wks	3 wks

Table : The columns show the number of locations, variables, and transitions in the counter abstraction, and the running times, in seconds, for computing the inductive invariant, constructing the abstraction, and for proving termination.

Related work

- R. Meyer. *On boundedness in depth in the π-calculus.* In *Fifth Ifip International Conference On Theoretical Computer Science–Tcs 2008*, 2008.
- A. Pnueli, J. Xu, and L. D. Zuck. *Liveness with (0, 1, ∞)-counter abstraction.* In *CAV*, 2002.
- G. Basler, M. Mazzucchi, T. Wahl, and D. Kroening. *Symbolic counter abstraction for concurrent software.* In *CAV*, 2009.
- S. Gulwani, T. Lev-Ami, and M. Sagiv. *A combination framework for tracking partition sizes.* In *POPL*, 2009.
- S. Joshi and B. König. *Applying the graph minor theorem to the verification of graph transformation systems.* In *CAV*, 2008.
- A. Gotsman, B. Cook, M. J. Parkinson, and V. Vafeiadis. *Proving that non-blocking algorithms don't block.* In *POPL*, 2009.

Conclusion

- Novel technique for proving fair termination of DBS that can be used to prove progress properties of concurrent data structures and distributed systems.
- An analysis that is both practical and sufficiently precise built on top of existing termination provers for counter systems.