Towards an Efficient Contextual Unfolder

Stefan Schwoon

LSV, ENS Cachan & CNRS, INRIA

ACTS III, Chennai, 27.01.2011

Opening remarks

Last year at ACTS:

- Theory behind construction of contextual unfolding
- Lots of open algorithmic questions

This year:

- Progress on algorithms and implementation
- Some experimental results

Work in progress, jointly with:

- César Rodríguez, author of Cunf tool
- Baldan, Bruni, Corradini, König

Outline

Motivation

Challenges

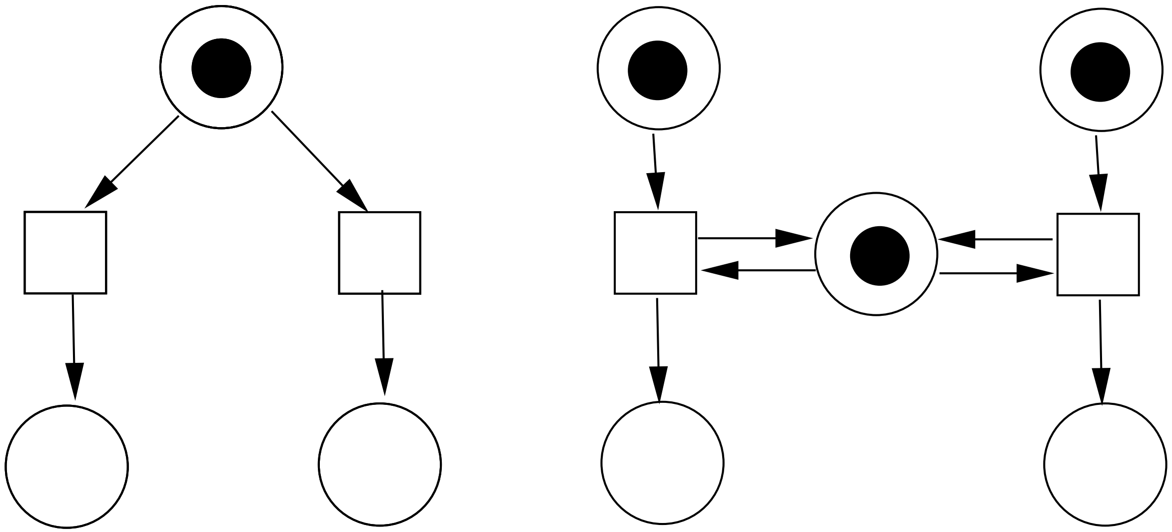
Solutions

Results

Motivation

Petri nets

Model for distributed, concurrent system:



Expresses independence, conflict, causality, . . .

Petri net unfoldings

Acyclic data structure that completely represents the behaviour of a Petri net; exploits concurrency inherent in the Petri net model.

Size between that of Petri net and that of reachability graph; once unfolding is computed, reachability queries become easier.

Large body of work on using unfoldings in verification.

> reachability, LTL model checking, diagnosis, . . .

Construction of an unfolding

The unfolding U of a Petri net N is an *acyclic*, infinite Petri net.

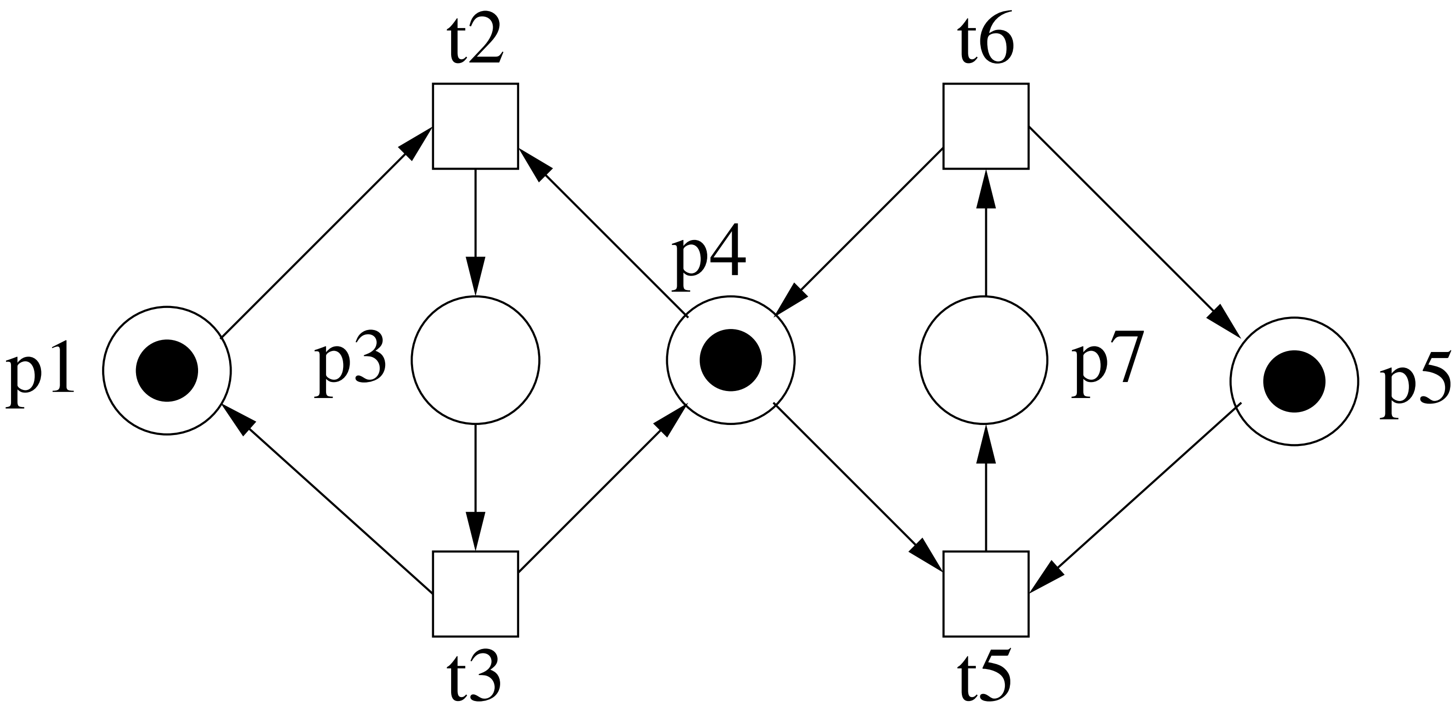
Places in U (called conditions) are labelled with places of N.

Transitions in U (called events) are labelled with transitions of N.

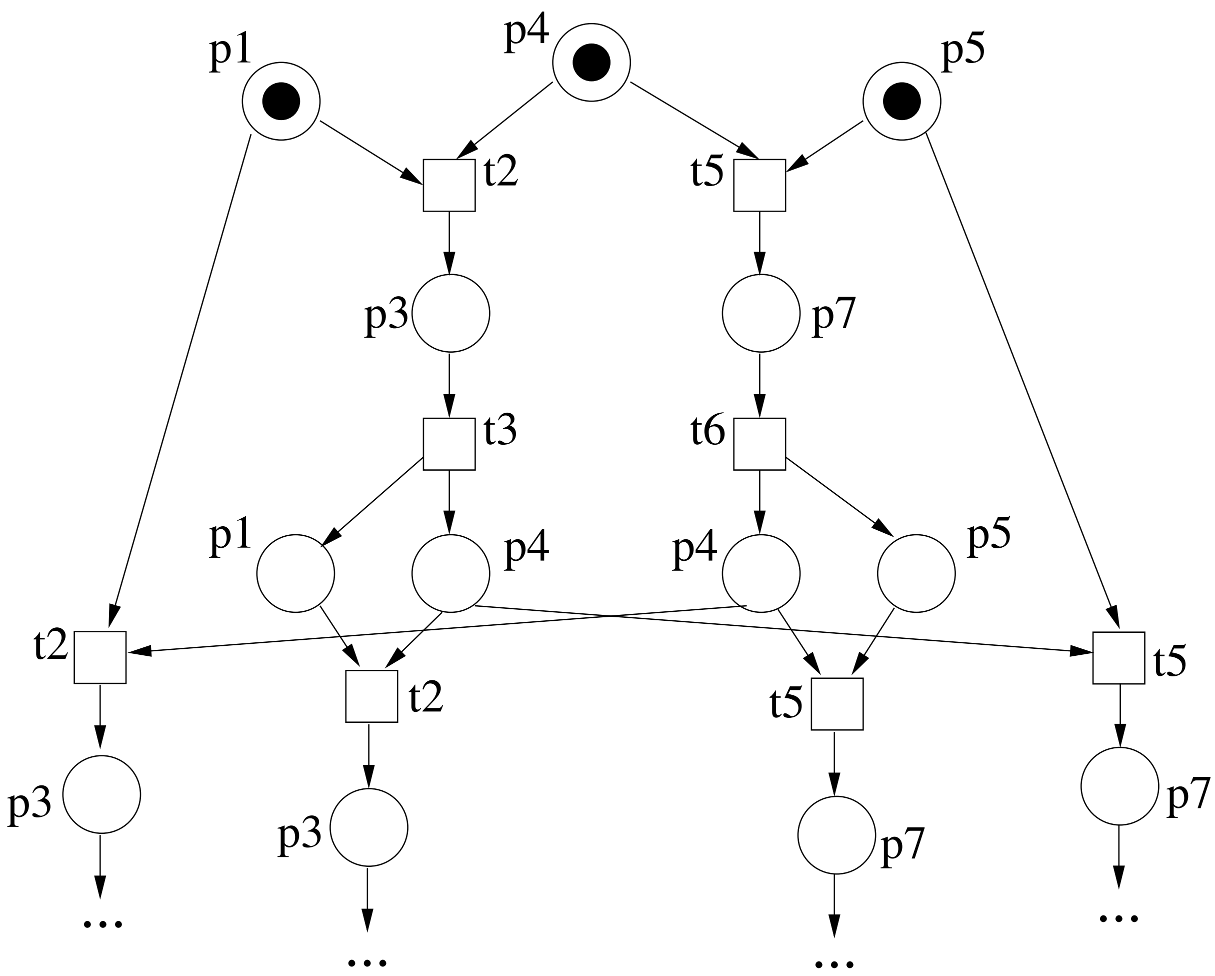
Modulo the labelling, the unfolding has the same behaviours and the same reachable states.

Construction: Start with "copies" of initially marked places; for every coverable marking in U whose labelling enables a transition in *N*, add a "copy" of that transition with *fresh copies* of the output places.

Example: Petri net. . .



. . . and its unfolding



Complete prefixes

In the following, we consider only nets that are 1-safe:

In any reachable marking, any place holds at most one token.

Sometimes guaranteed by construction (e.g., communicating FA).

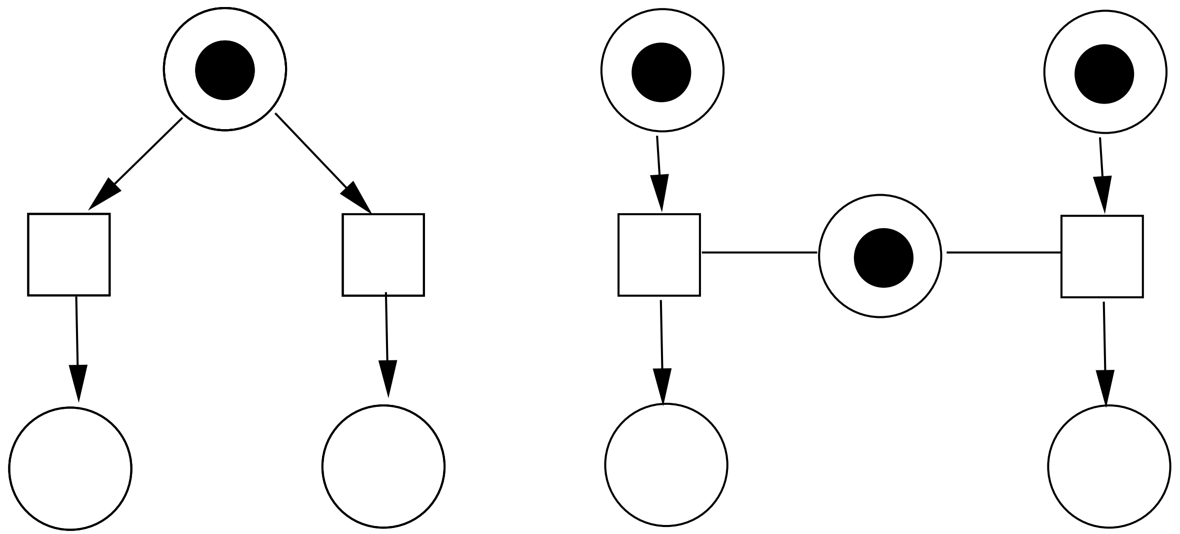
Even the unfolding of a 1-safe Petri net is (in general) infinite!

Possible to construct a *finite*, complete prefix P of U:
for every marking m reachable in N there exists a marking m' in P whose labelling equals m.

Technique: declare certain events as cut-offs.

Contextual nets

Explicit modelling of “read/test” actions (arcs without arrows):



Intuition: The read arc does not consume or touch the token, it merely verifies its presence. For any transition t, we distinguish its preset ${}^\bullet t$, its context $\underline{t}$, and its postset $t^\bullet$.

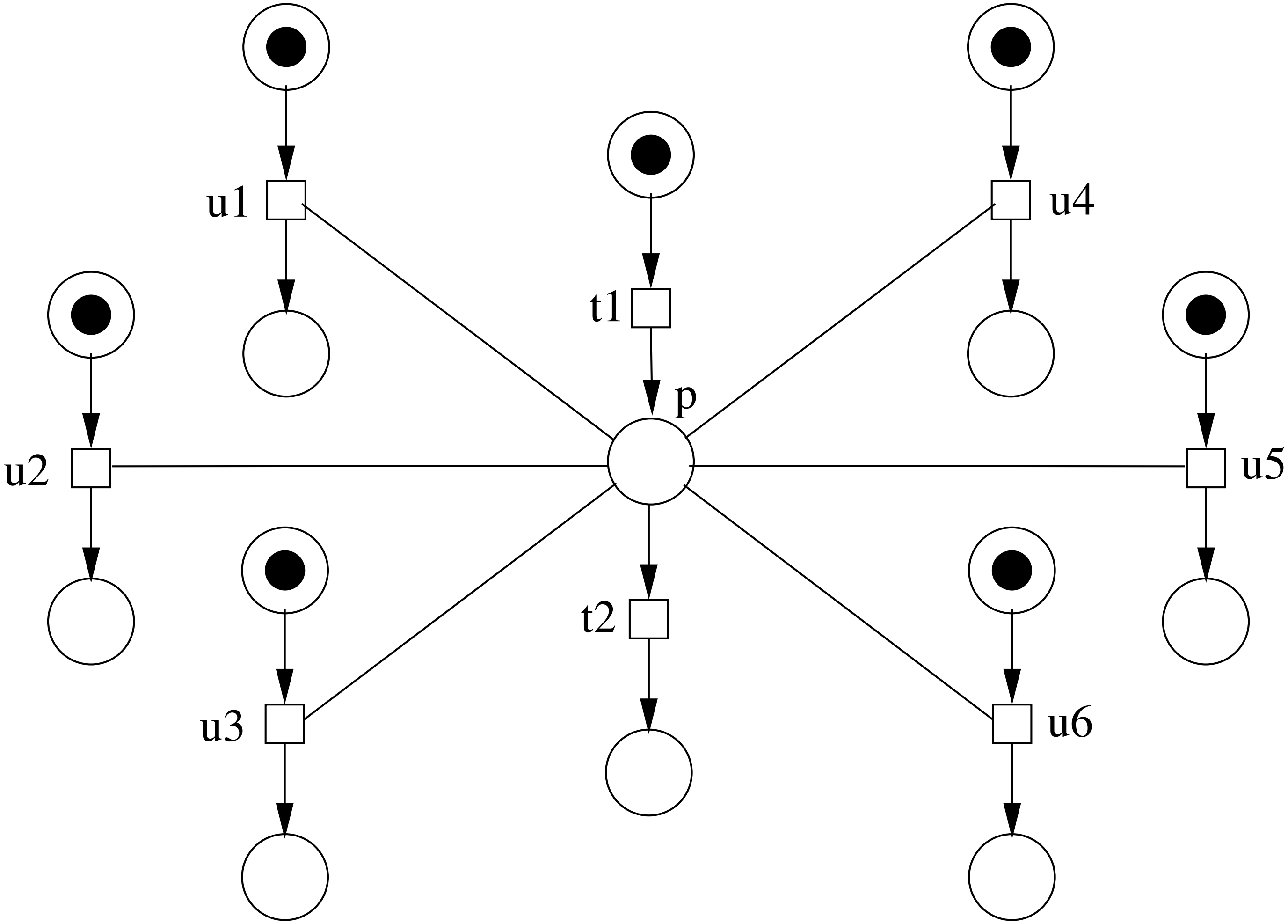
Contextual unfoldings

The unfolding U of a *contextual* net N is an *acyclic*, infinite *contextual* net.

Contextual nets faithfully model concurrent read accesses;
$\Longrightarrow$ better exploitation of concurrency
$\Longrightarrow$ smaller unfoldings

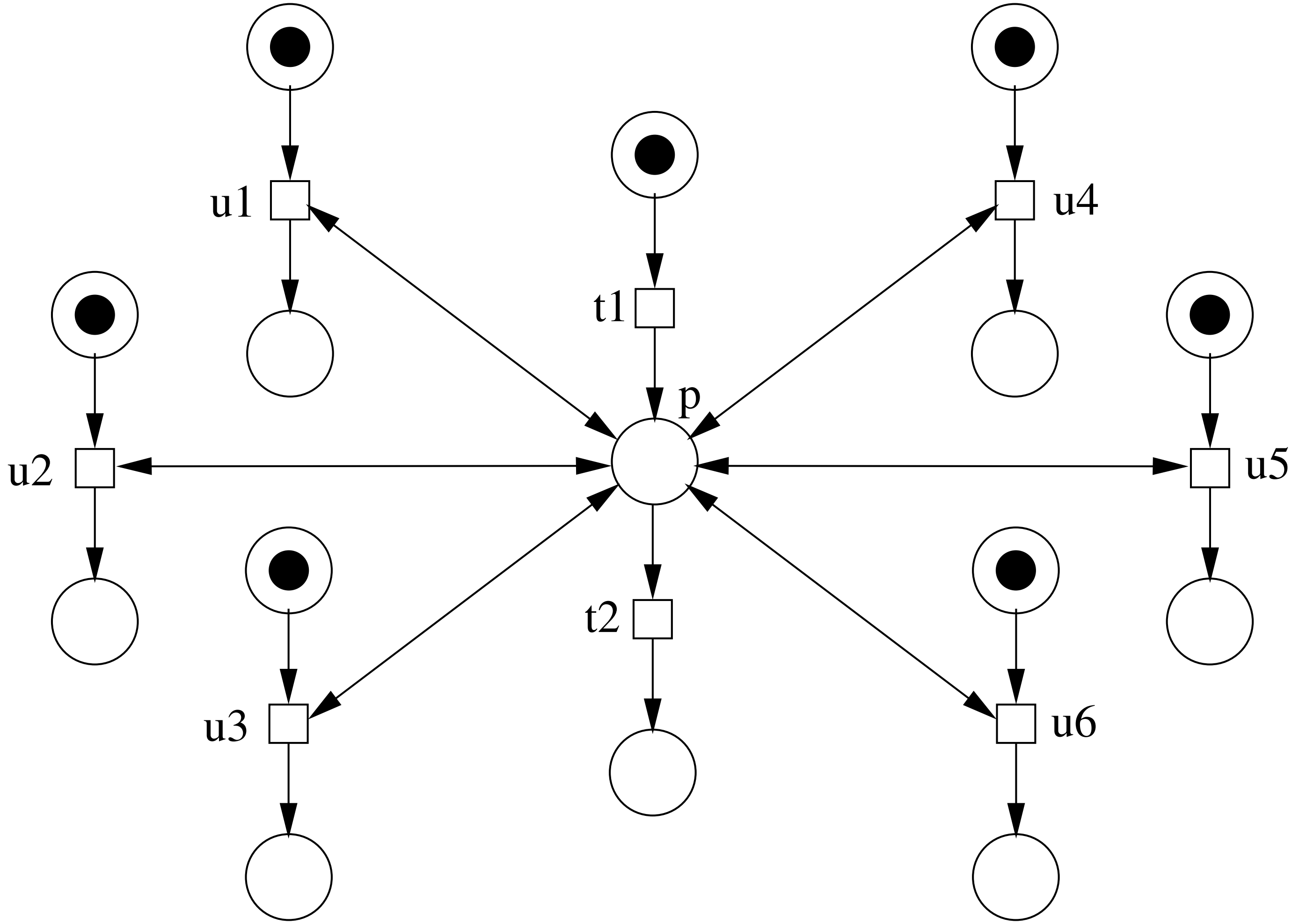
Example: Contexual unfolding

Consider the contextual net shown below (six readers):



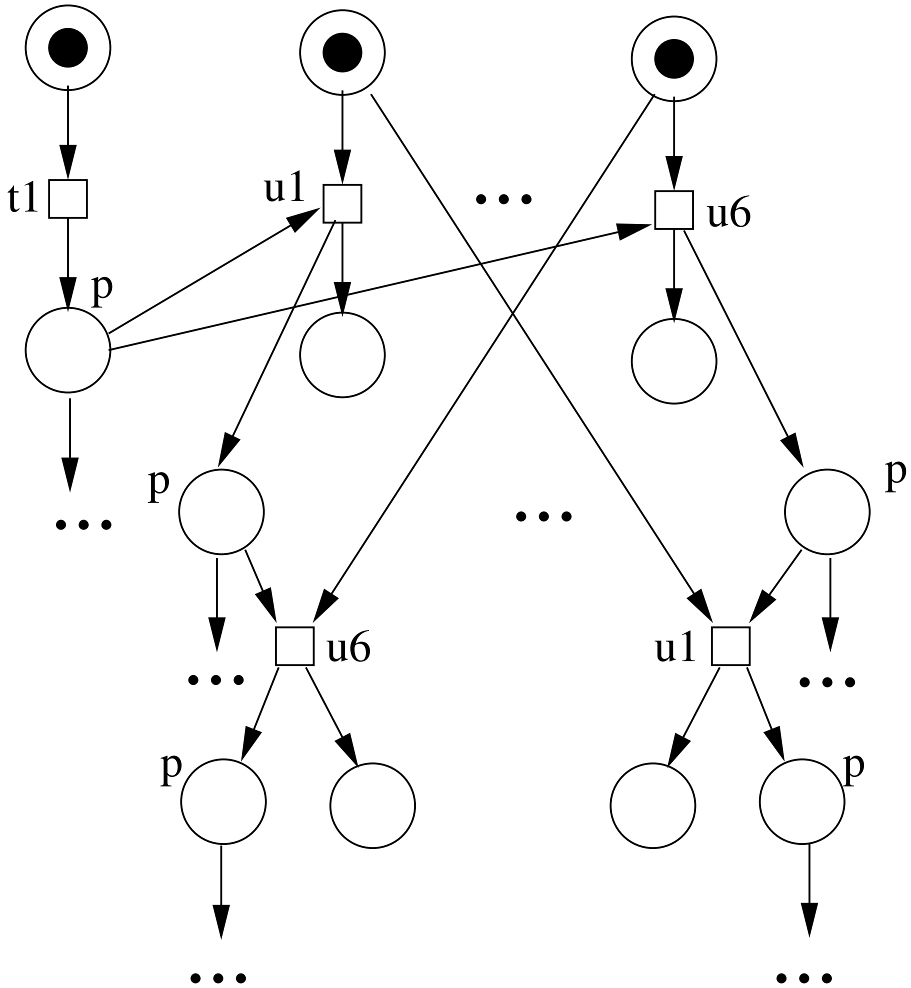
Naïve encoding into Petri nets

Why not replace read arcs by double arrows and unfold normally?



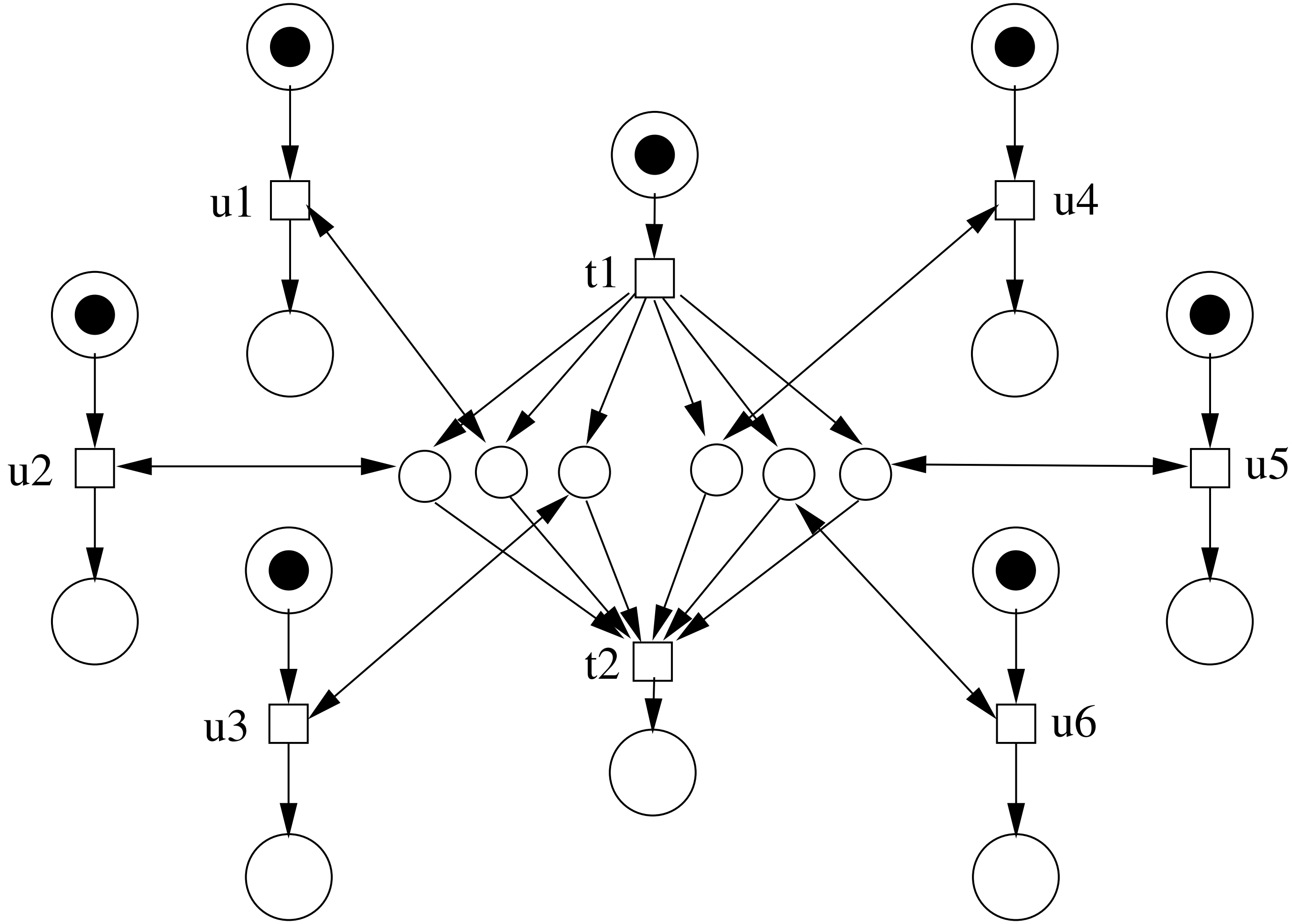
Naïve encoding into Petri nets

Resulting unfolding: 6! u_i-labelled events, no exploitation of concurrency!



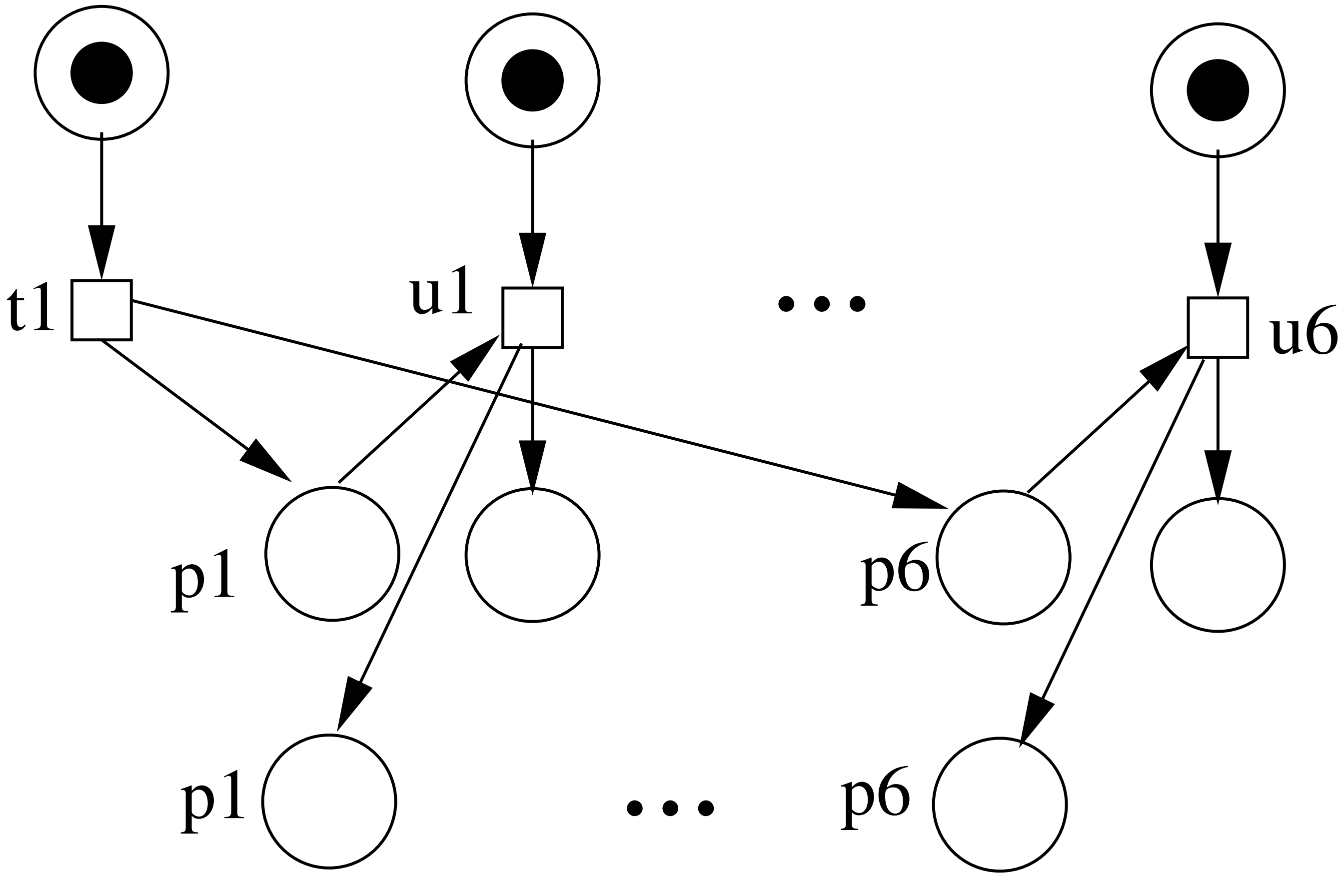
Place-replication encoding into Petri nets

(PR-encoding) Replace *p* by six copies, one for each reader.



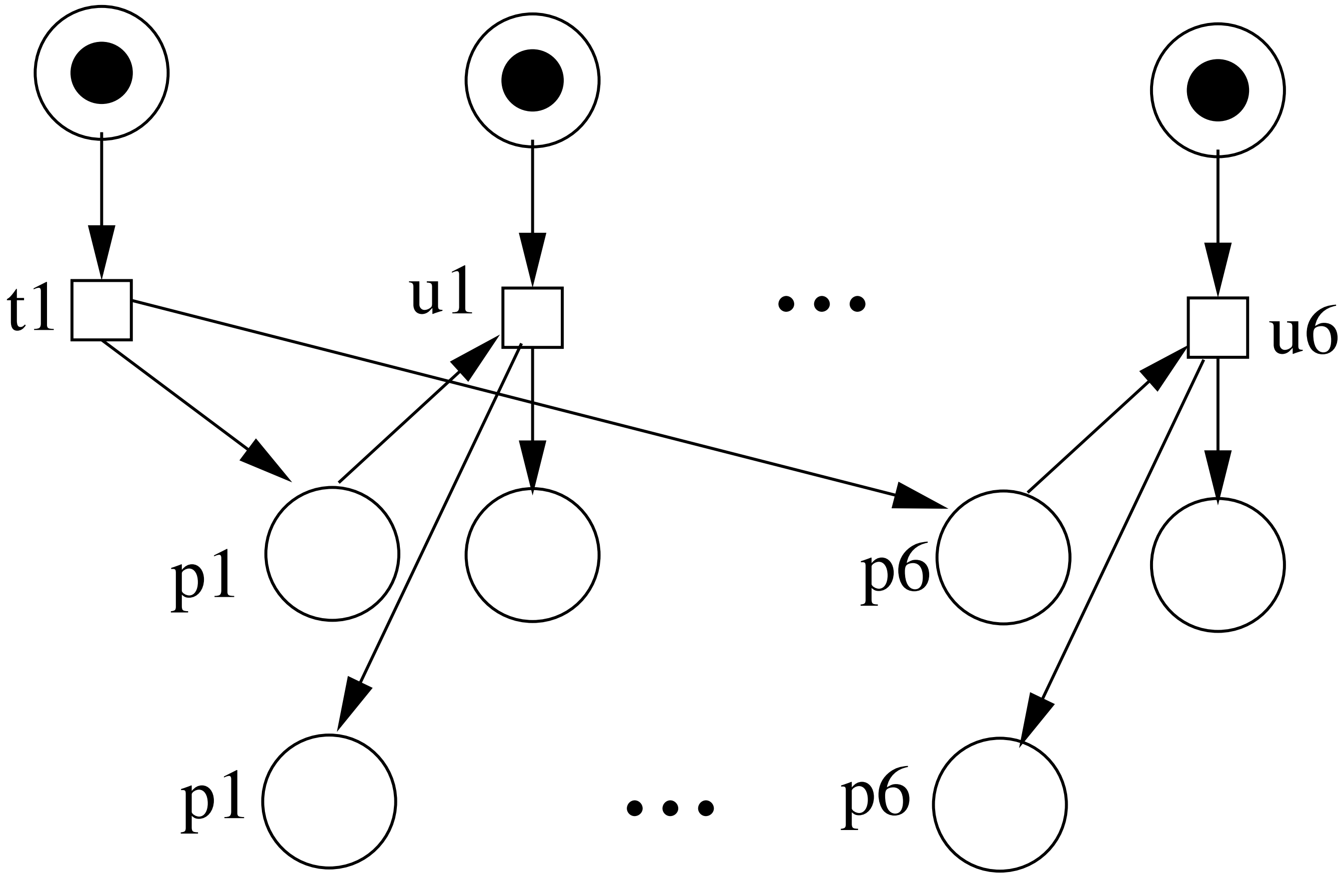
Place-replication encoding into Petri nets

Resulting unfolding: just one copy each of $u_1, \ldots, u_6$!



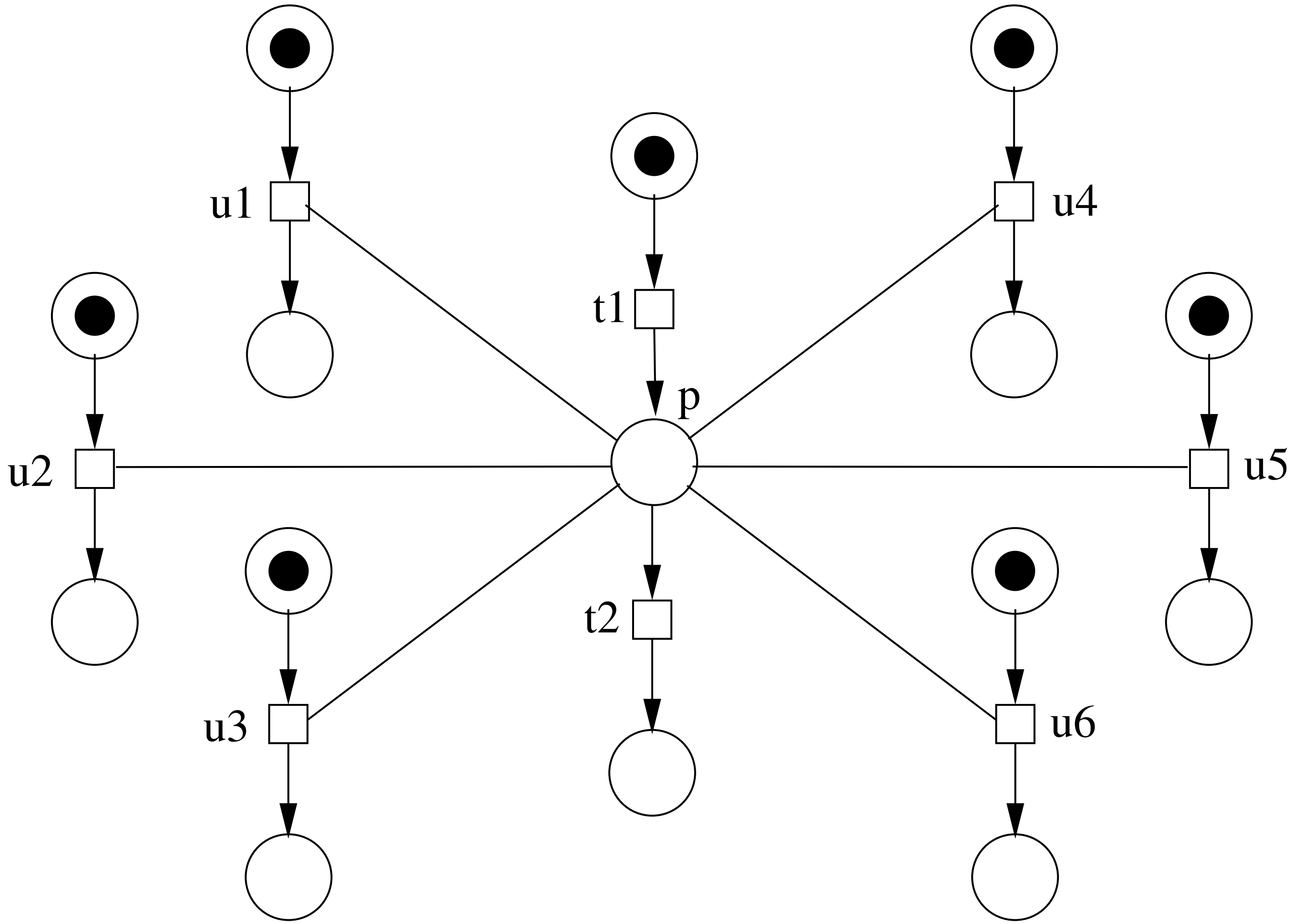
Place-replication encoding into Petri nets

However, we will still have 2^6 copies of t_2.



Direct contextual unfolding

Here: unfolding identical to net, no blowup at all!



Contextual unfoldings

The unfolding U of a *contextual* net N is an *acyclic*, infinite *contextual* net.

Contextual nets faithfully model concurrent read accesses;

$\Longrightarrow$ better exploitation of concurrency

$\Longrightarrow$ smaller unfoldings

Construction of a finite prefix still possible (see last year's talk).

Contextual unfoldings

The unfolding U of a *contextual* net N is an *acyclic*, infinite *contextual* net.

Contextual nets faithfully model concurrent read accesses;
$\Longrightarrow$ better exploitation of concurrency
$\Longrightarrow$ smaller unfoldings

Construction of a finite prefix still possible (see last year's talk).

Disadvantage: construction becomes rather more complex

Challenges

Algorithmic problems

The two principal problems in unfolding are:

Problem 1: Decide (efficiently) whether a set of places is coverable.

$\rightarrow$ decision required whenever the unfolding is extended

Problem 2: Decide with events are cut-offs.

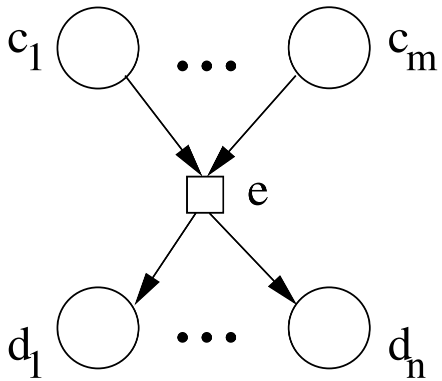
$\rightarrow$ to obtain finite complete prefix

Revisiting Problem 1 on Petri nets

Two conditions c, c' are called concurrrent ($c \parallel c'$) iff there exists a firing sequence that marks them both.

Fact I: A set S of conditions is coverable iff $c \parallel c'$ for all $c, c' \in S$.

Fact II: (Non-)Concurrency is inherited by causal successors:

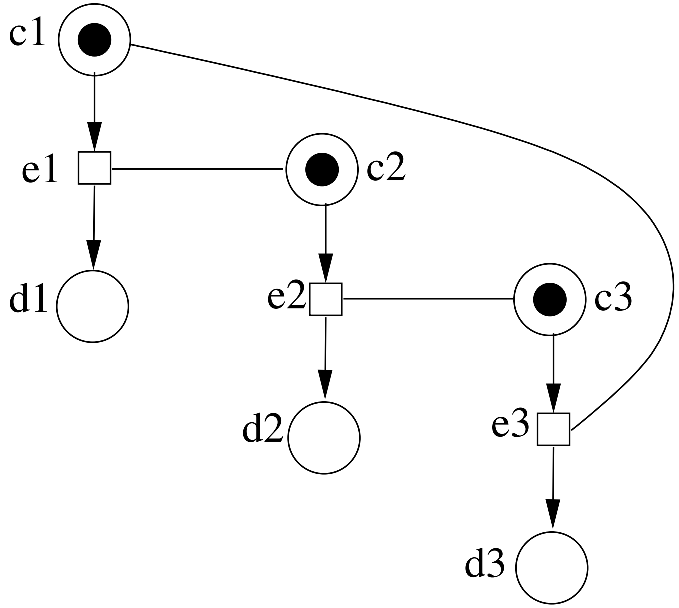


For any condition d_i (where $1 \leq i \leq n$) and c' we have:

$$d_i \parallel c' \iff c' \in e^\bullet \vee \left(c' \notin {}^\bullet e \wedge \bigwedge_{j=1}^{m} c_j \parallel c' \right)$$

Concurrency on contextual nets

Bad news (from last time): Fact I no longer holds:



Any pair $\{d_1, d_2\}$, $\{d_2, d_3\}$, $\{d_2, d_3\}$ is coverable, but $\{d_1, d_2, d_3\}$ is not.

$\Rightarrow$ key element from Petri unfolding algorithms unavailable

Revisiting Problem 2 on Petri nets

Let $<$ (causality) be the transitive closure of the relation $\{ (x, y) \mid x \in {}^\bullet y \}$.

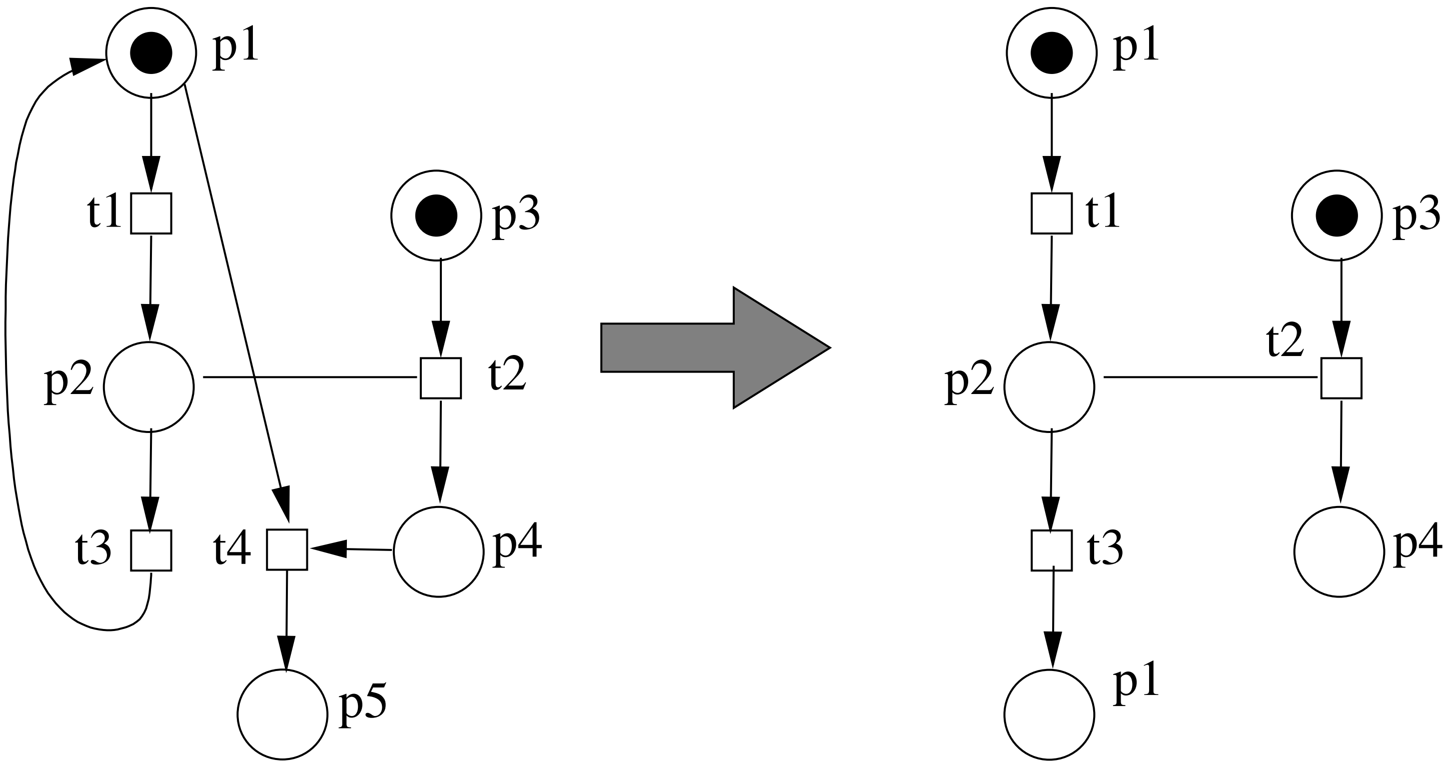
We write $\lfloor e \rfloor := \{ e' \mid e' < e \}$ (the causal-predecessor events of e.

Each event is associated with the marking M_e generated by firing the events in $\lfloor e \rfloor$.

If M_e equals the initial marking, or if there already exists an event e' with $M_e = M_{e'}$, then e is declared a cut-off.

Cut-offs in contextual nets

Read arcs do not fit into this scheme:



Should event t_2 be considered a causal predecessor of event t_3?

Asymmetric conflict

Let e, e' be distinct events. They are in asymmetric conflict, written $e \nearrow e'$ iff $e^\bullet \cap {}^\bullet e' \neq \emptyset$, or ${}^\bullet e \cap {}^\bullet e' \neq \emptyset$, or $\underline{e} \cap {}^\bullet e' \neq \emptyset$.

Intuition: “If both e and e' happen, then e happens first.”

Let C be a *finite* set of events in a contextual unfolding.
We call C a configuration iff:

(i) $e \in C$ and $e' < e$ imply $e' \in C$ (i.e., C is causally closed);

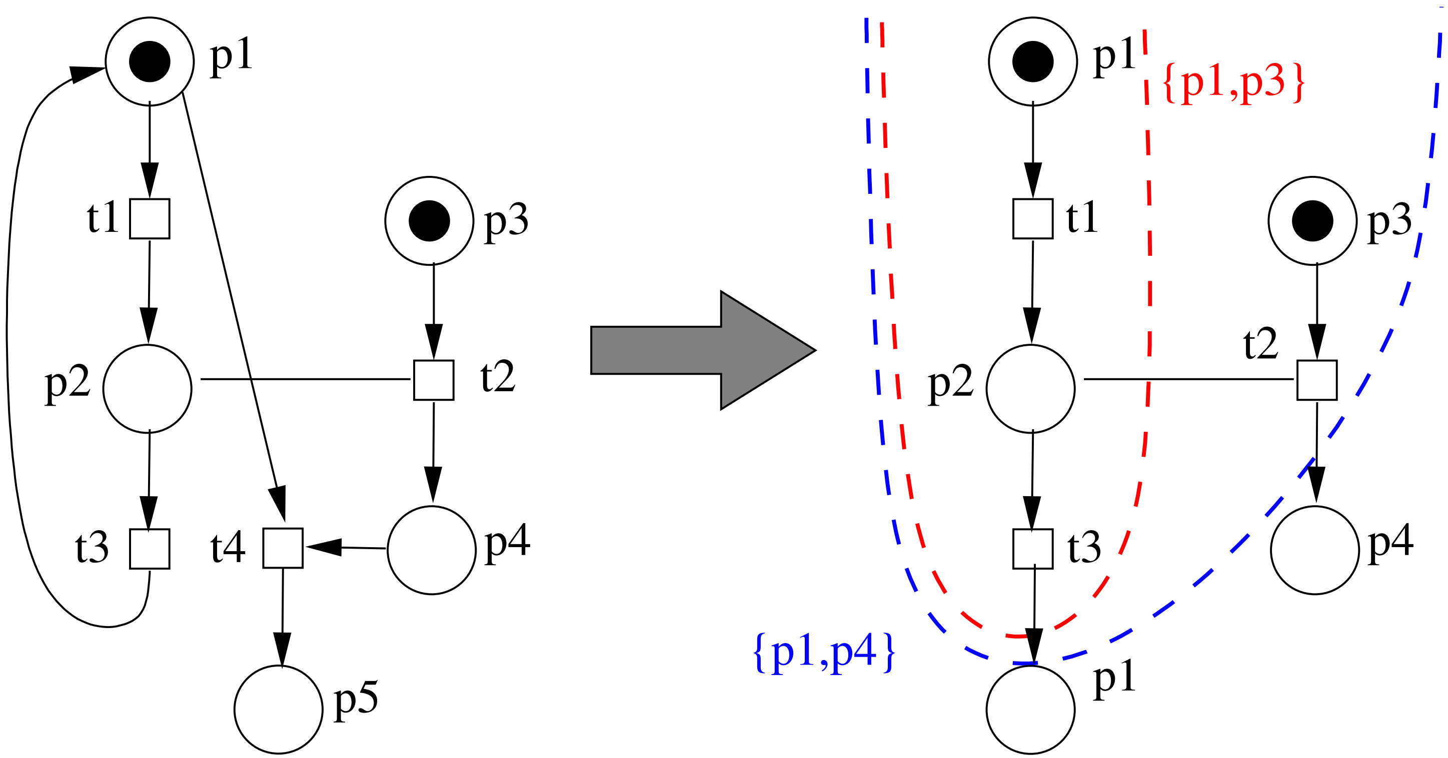
(ii) $\nearrow \cap \, (C \times C) =: \, \nearrow_C$ does not contain any cycles;

The marking associated with C is $M_C = (M_0 \cup C^\bullet) \setminus {}^\bullet C$, where M_0 is the initial marking.

Let C be a configuration and $e \in C$ an event. The history of e in C is the configuration $C[\![e]\!] := \{\, e' \in C \mid e' \nearrow_C^* e \,\}$.

Example: Histories

Below, two histories for t_3 and their markings are shown:



Cut-offs for contextual nets

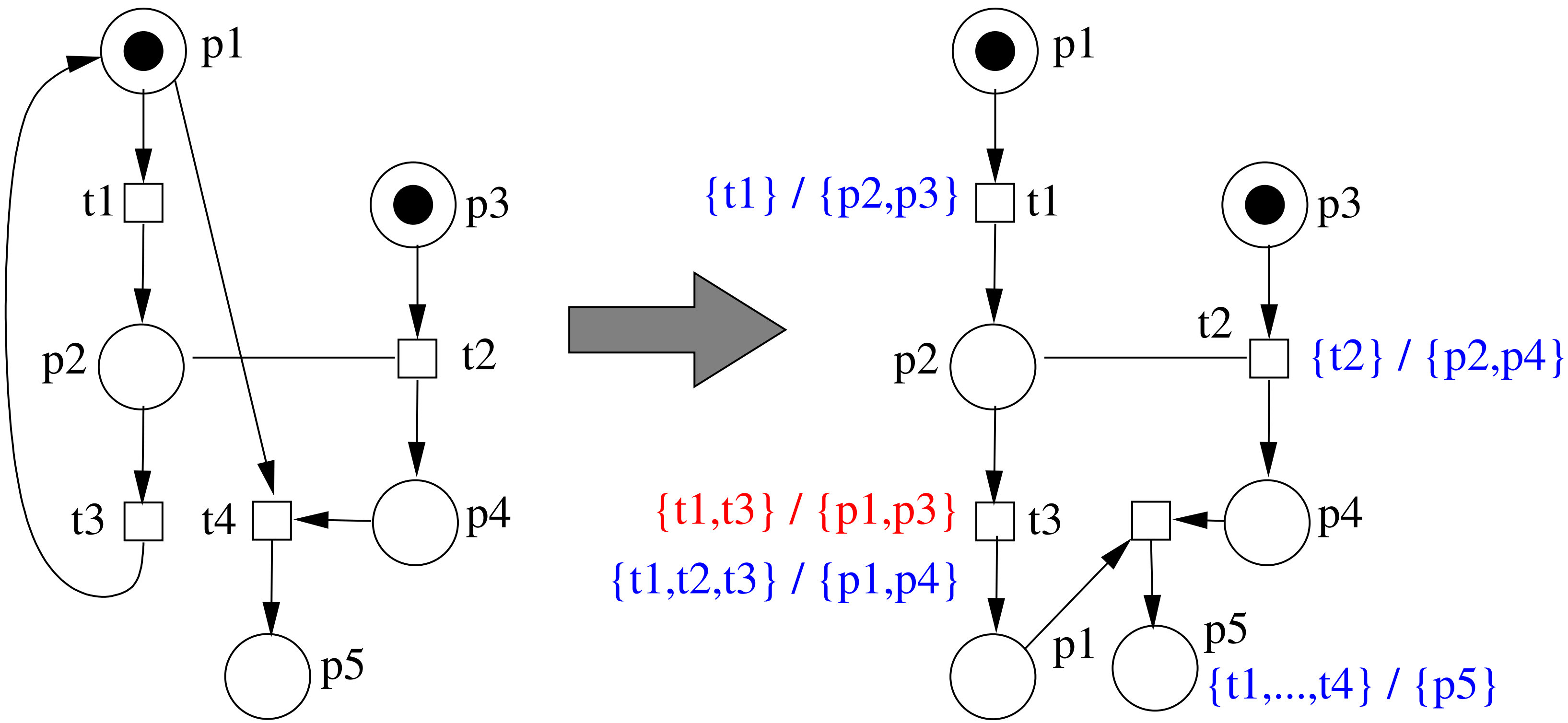
We shall annotate events with a relevant subset of their histories.

The cutoff criterion is lifted to *histories* (rather than events); the future of an event is explored if it has at least one non-cutoff history.

How to choose that relevant subset: see last year's talk.

Example: Prefix with cut-offs

Unfolding with annotated histories:



t_3 has one cut-off history (marked red) and one non-cutoff history.

Solutions

Goal

Implement the (abstract) algorithm presented last year

- existing implementation for Petri nets not-reusable due to presence of asymmetric conflict and histories

Motivation: generate small unfoldings, efficiency unclear a priori

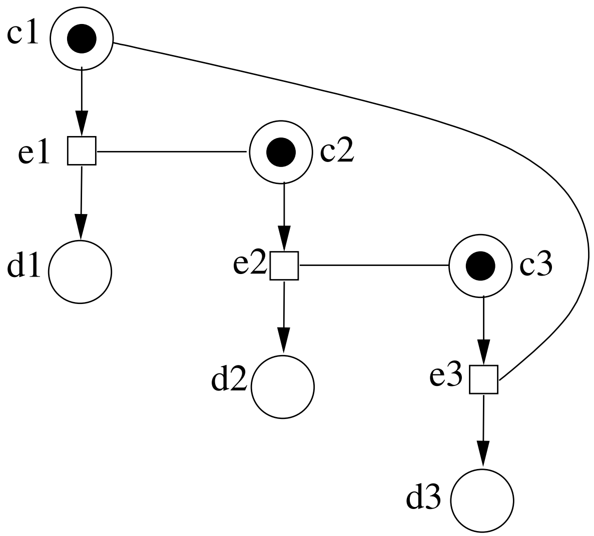
Problems to overcome (among others):

- Efficiently find coverable sets
- Data structures to deal with histories

Conflicting histories

Let C_1, C_2 be two configurations. We say that C_1 and C_2 are in conflict ($C_1 \# C_2$) iff there exist events $e \in C_2 \setminus C_1$ and $e' \in C_1$ such that $e \nearrow e'$ (or vice versa).

If $C_1 \# C_2$, then C_1 and C_2 have “diverged”; they cannot both be extended to a common, bigger configuration. However, if $\neg(C_1 \# C_2)$, then $C_1 \cup C_2$ is a configuration (the other direction does not hold in general.)



e.g., $\{e_1\} \# \{e_2\}$, but $\neg(\{e_1\} \# \{e_1, e_2\})$

Enriched conditions

Let c be a condition.

if ${}^\bullet c = \{e\}$ and H is a history of e, then H is a generating history of c; if ${}^\bullet c = \emptyset$, then $\emptyset$ is.

if $e \in \underline{c}$ and H is a history of e, then H is a reading history of c.

A history of c is any

– generating or reading history of c;

– union $H_1 \cup H_2$ of non-conflicting histories of c.

We call $\langle c, H \rangle$ an enriched condition.

Example: $\{e_2\}$ is a generating history for d_2 and a reading history for c_3.

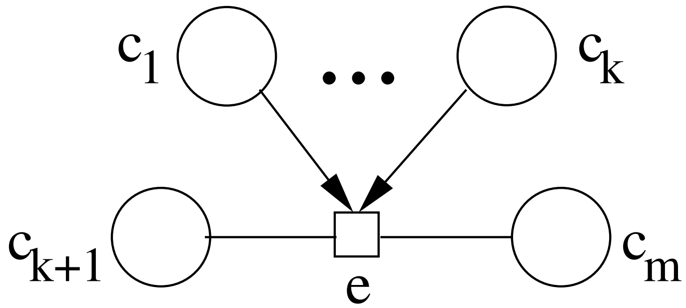
Composing histories

Let e be an event such that $^\bullet e = \{c_1, \ldots, c_k\}$ and $\underline{e} = \{c_{k+1}, \ldots, c_m\}$. Then H is a history of e iff there exist *arbitrary* histories $H_1, \ldots, H_k$ for $c_1, \ldots, c_k$ and *generating* histories $H_{k+1}, \ldots, H_m$ for $c_{k+1}, \ldots, c_m$ such that:

$H = \{e\} \cup \bigcup_{i=1}^{m} H_i$

$\nearrow_H$ is free of cycles

$c_1, \ldots, c_m \in M_H$



Provides a strategy for unfolding procedure: start with generating histories for initial conditions; for every new enriched condition, use above theorem to construct new event/condition histories.

A concurrency relation for contextual nets

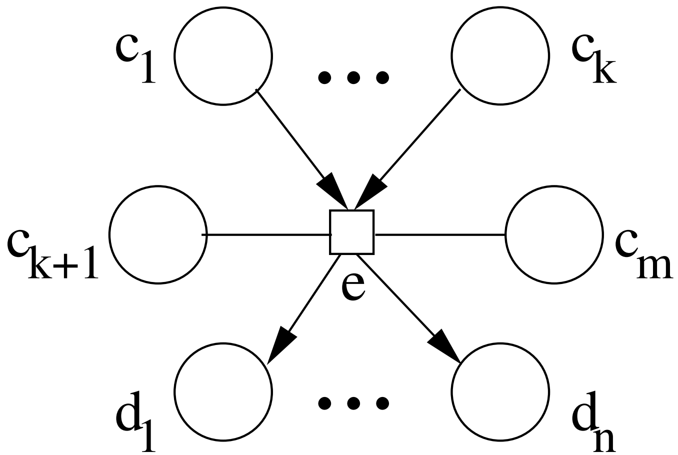
Let $\rho = \langle c, H \rangle$ and $\rho' = \langle c', H' \rangle$ be two enriched conditions. We say $\rho \parallel \rho'$ (that is, ρ, ρ' are concurrent) iff:

$$\neg(H \# H') \qquad \text{and} \qquad c, c' \in M_{H \cup H'}$$

Fact I: A set $S = \{c_1, \ldots, c_k\}$ is coverable iff there exist histories $H_1, \ldots, H_k$ such that $(c_i, H_i) \parallel (c_j, H_j)$ for all $1 \leq i < j \leq k$.

Computing the concurrency relation

Fact II: Let H be a newly discovered history for e composed from histories H_j for c_j, where $1 \leq j \leq m$; we let $\rho_j := \langle c_j, H_j \rangle$.



Moreover, let $\rho = \langle c, H \rangle$ for some $c \in e^\bullet$ and $\rho' = \langle c', H' \rangle$ any existing enriched history. Then:

$$\rho \parallel \rho' \iff (c' \in e^\bullet \wedge H = H') \vee \left(c' \notin {}^\bullet e \wedge \bigwedge_{j=1}^{m} (\rho_j \parallel \rho') \wedge \underline{({}^\bullet e)} \cap H' \subseteq H \right)$$

Notes

The condition $\underline{({}^\bullet e)} \cap H' \subseteq H$ can be implemented with reasonable efficiency:

No need to traverse H' completely when checking $\rho \parallel \rho'$: can remember candidate events when computing the marking $M_{H'}$.

A similar result exists for general (i.e., composed) histories of conditions.

Efficient detection of coverable sets akin to Petri net method.

History graph

We call $\langle e, H \rangle$, where H is a history of e, an enriched event.

Let $\mathcal{H}$ be the node-labelled directed graph whose nodes are the enriched events and an edge $\langle e, H \rangle \to \langle e', H' \rangle$ exists iff $e' \nearrow e$ and $H' = H[\![e']\!]$. Node $\langle e, H \rangle$ is labelled by e.

Notes:

- $\mathcal{H}$ can be incrementally constructed during the construction of the unfolding.

- The history H of a node can be recovered by recursively following the outgoing edges of the node and reading the labels.

- All required operations on histories can be implemented as simple neighbourhood queries on $\mathcal{H}$, for instance, finding all events in $H \cap \underline{({}^{\bullet}e)}$ in the computation of concurrency.

Results

Benchmarks

Benchmarks used: Corbett’s set of examples

- Standard benchmarks in unfolding literature
- Derived from concurrent finite automata, hence 1-safe
- Different characteristics, fairly sure to exhibit implementation flaws
- Not specifically geared towards contextual nets

Experiments I: Mole vs Cunf

	events	Mole	Cunf
bds_1	12900	0.47	0.52
buf100	5051	2.85	2.10
byzagr4	14724	3.04	3.40
dpd_7	10457	0.93	0.87
dph_7	37272	0.79	0.99
elevator_4	16856	2.00	2.01
fifo20	41792	4.89	4.14
ftp_1	83889	76.02	77.09
furnace_3	25394	1.22	1.10
key_4	67954	1.80	2.18
q_1.sync	10722	1.36	1.22
rw_12.sync	98361	2.89	3.98
rw_1w3r	15401	0.30	0.39
rw_2w1r	9241	0.23	0.29

Mole is an (efficient) unfolder for Petri nets.

Cunf is the new contextual unfolder.

We run both tools on the original Petri nets (no read arcs!) $\Rightarrow$ results are the same

Times given in seconds.

Conclusion: implementation of Cunf is reasonably efficient (factors 0.7 to 1.4 w.r.t. Mole)

Conclusion: histories handled gracefully

Experiments II: Naïve vs contextual

	events	Naïve	Context.	c-events
bds_1	12900	0.52	0.16	4032
buf100	5051	2.10	2.17	5051
byzagr4	14724	3.40	2.59	8044
dpd_7	10457	0.87	0.94	10457
dph_7	37272	0.99	0.99	37272
elevator_4	16856	2.01	1.30	16856
fifo20	41792	4.14	4.14	41792
ftp_1	83889	77.09	34.60	50928
furnace_3	25394	1.10	0.62	16893
key_4	67954	2.18	9.35	21742
q_1.sync	10722	1.22	1.20	10722
rw_12.sync	98361	3.98	3.14	98361
rw_1w3r	15401	0.39	0.43	14982
rw_2w1r	9241	0.29	0.36	9241

Contextual nets obtained by converting read/write loops to read arcs.

Cunf used on both tools.

3 examples w/o read arcs (in italics).

Some savings on time and size (not always on both).

Inefficiency detected in key_4 example, we're working on fixing it.

Experiments III: Contextual vs PR-encoding

	Context.	PR-enc.
bds_1	0.16	0.27
buf100	2.17	2.16
byzagr4	2.59	5.30
dpd_7	0.94	0.98
dph_7	0.99	1.00
elevator_4	1.30	557.06
fifo20	4.14	4.12
ftp_1	34.60	113.71
furnace_3	0.62	0.96
key_4	9.35	4.28
q_1.sync	1.20	2.18
rw_12.sync	3.14	7.66
rw_1w3r	0.43	0.70
rw_2w1r	0.36	8.86

PR-encoding obtained from contextual nets.

Cunf used on both tools.

#histories in contextual = #events in PR

Nonetheless, contextual is consistently better (except key_4, for now).

Explanation: combinatorial problems in PR resulting from larger transition pre-sets.

Conclusions and future work

Contextual unfolding feasible and efficient

Beats PR-encoding

Work in progress, further ideas for optimization

Will look at more extensive benchmarks

To do: look at the applications in verification, diagnosis, etc.