Petruchio: From Dynamic Networks to Nets*

Roland Meyer[1] and Tim Strazny[2]

[1] LIAFA & CNRS and [2] University of Oldenburg

Abstract. We introduce PETRUCHIO, a tool for computing Petri net translations of dynamic networks. To cater for unbounded architectures beyond the capabilities of existing implementations, the principle fixed-point engine runs interleaved with coverability queries. We discuss algorithmic enhancements and provide experimental evidence that PETRUCHIO copes with models of reasonable size.

1 Introduction

PETRUCHIO computes Petri net representations of dynamic networks, as they are the basis to automatic-verification efforts [19]. As opposed to static networks where the topology is fixed, in dynamic networks the number of components as well as connections changes at runtime. Whereas earlier tools covered only finite state models [6, 23, 9], PETRUCHIO features the unbounded interconnection topologies needed when tackling software. Theoretically, the implementation rests upon recent insights on the relationship between dynamic networks and Petri nets [15, 14]. Practically, the heart of our algorithm is an unconventional fixed-point computation interleaved with coverability queries.

Run on a series of benchmarks, we routinely translate systems of two hundred lines of π-calculus code into Petri nets of around 1k places within seconds. The computability threshold lies around 90k transitions, which is in turn beyond the capabilities of latest net verification tools [13]. A concurrency bug found in an automated manufacturing system and automatic verification of the gsm benchmark underline the practicability of our tool [16].

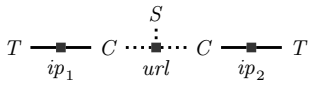
Related Work There has been recent interest in translation-based network verification [4, 3, 16], PETRUCHIO puts these efforts into practice. Besides, the well-structured transition system framework [2, 5, 8, 1, 24] as well as abstraction-based verification techniques [21, 20, 11, 22] have been applied.

2 Foundations behind Petruchio

Online banking services are typical dynamic networks where failures have severe consequences and thus verification is required. We model this example in the π-calculus and for simplicity explain the implementation of the Petri net translation from [14]. Based on similar algorithmic ideas, the fixed-point engine in PETRUCHIO also handles the more involved translations from [3, 15].

* The first author was supported by the French ANR projects Averiss and Veridyc. Appeared in Proc. of CAV, volume 6174 of LNCS, pages 175-179. Springer, 2010.

$$S(url) = url(y).(\overline{y}\langle bal\rangle \mid S(url))$$
$$C(url) = \nu ip.\overline{url}\langle ip\rangle.ip(dat).C(url)$$



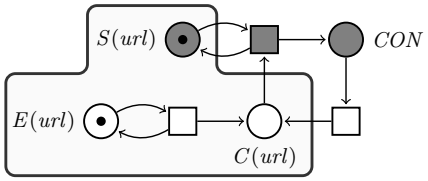


Fig. 1. π-calculus model *Bnk* of an online banking service (top left) and a reachable state represented as interconnection topology (bottom left). The structural semantics $\mathcal{N}_S[\![Bnk]\!]$ is depicted to the right, *CON* abbreviates $\nu ip.(ip(dat).C(url) \mid \overline{ip}\langle bal\rangle)$.

The overall functionality of the banking system *Bnk* is a login of the client, which spawns a new thread that displays the account balance. We detail the π-calculus model in Figure 1. The bank server $S(url)$ is located at some url and ready to receive the ip-address of a customer, $url(y)$. Upon reception, a new thread is spawned (parallel composition $\mid$). It transmits the balance, $\overline{y}\langle bal\rangle$, and terminates. The server itself is immediately ready for new requests. To guarantee proper interaction, the client sends its private (indicated by a νip quantifier) ip-address $\overline{url}\langle ip\rangle$ and waits on this channel for data. We assume an environment $E(url)$ that generates further customers.

Translation Although the banking service exhibits an unbounded number of connection topologies, there exists a finite basis of connection *fragments* they are built from. Fragments are maximal subgraphs induced by private channels and can be determined in linear time by minimising the scopes of the quantifiers. For instance, a private connection between client and thread is fragment $\nu ip.(ip(dat).C(url) \mid \overline{ip}\langle bal\rangle)$. It is present twice in the example state in Figure 1.

For verification purposes, the *structural semantics* translates dynamic networks into Petri nets. Every reachable fragment yields a place, communications inside and between fragments determine the transitions, and the initial state is the decomposition of the system's initial state into fragments. The running example is represented by the Petri net $\mathcal{N}_S[\![Bnk]\!]$ in Figure 1.

An *isomorphism* between the transition systems, $Sys =_{iso} \mathcal{N}_S[\![Sys]\!]$, proves the net representation suitable for model checking purposes. In fact, it is a lower bound on the information required for verifying topological properties. This follows from a *full abstraction* result wrt. syntactic equivalence, $Sys \equiv Sys'$ iff $\mathcal{N}_S[\![Sys]\!] = \mathcal{N}_S[\![Sys']\!]$, and the descriptive power of topological logics [7].

3 Algorithmic Aspects

The declarative definition of the structural semantics leaves the problem of its computability open. Taking a classical view from denotational semantics, we understand it as an unconventional least fixed-point on a particular set of nets. A

dynamic network *Sys* gives rise to a function ϕ_{Sys} on nets. As an example, consider the subnet N shown in the box in Figure 1. An application $\phi_{Bnk}(N)$ extends it by the communication between client and server (dark). The least fixed-point of such a ϕ_{Sys} is in fact the structural semantics, $\mathcal{N}_{\mathcal{S}}[\![Sys]\!] = lfp(\phi_{Sys})$. Thanks to continuity, we can compute it by iterating the function on the empty net, $lfp(\phi_{Sys}) = \sqcup\{\phi^n_{Sys}(\mathcal{N}_\emptyset) \mid n \in \mathbb{N}\}$. The algorithm terminates precisely on systems with a finite structural semantics. They are *completely characterised* by the existence of a finite basis of fragments [14].

Leading yardstick to a practical implementation is the efficient *computation of extensions* and the quick *insertion of places*.

Computing Extensions An application of function ϕ_{Sys} determines the set of transitions the net has to be extended with. Transitions between fragments rely on pairs (F, G) of *potential communication partners*. Hashing the leading communication channels, they can be determined in constant time. Each such pair then needs a *semantic confirmation* of F and Gs simultaneous reachability. We reduce it to a coverability problem in the Petri net built so far and implement strategies to *avoid* unnecessary queries and *speed-up* coverability checks.

To reduce the number of checks, PETRUCHIO augments the breadth-first fixed-point computation with dedicated depth-first searches. Whenever fragments F and G are found simultaneously markable, we build their *internal closure* $cl(F)$. It consists of all fragments reachable from F with internal communications. By definition, containment in the internal closure is a semantic confirmation for all potential communication partners $F' \in cl(F)$ and $G' \in cl(G)$. Their transitions can be added without further coverability queries.

Despite the advantage of incremental computability [12], Karp and Miller graphs turned out impractical for coverability checks due to their size. Instead, we perform independent backwards searches [2] that we prune with knowledge about place bounds. These bounds are derived from place invariants, and we currently use an incomplete cubic time algorithm. Our experiments show that already non-optimal bounds dramatically speed-up the backwards search.

Inserting Places Every newly discovered fragment F in $\phi_{Sys}(N)$ has to be compared for syntactic equivalence $\equiv$ with the places in the original net N. Since these checks $F \equiv G$ are graph isomorphism complete [10], we implemented a technique in PETRUCHIO to minimise their number.

We abstract fragments to so-called *signatures* $sig(F)$. As equality of these signatures is necessary for syntactic equivalence, they allow us to quickly refute non-equivalent pairs $F \not\equiv G$. Technically, the theory rests upon functions α that are invariant under syntactic equivalence, $F \equiv G$ implies $\alpha(F) = \alpha(G)$. A signature is a combination of these *indicator values*, $sig(F) := \alpha(F).\beta(F)\ldots$ We use ten values, ranging from number of free names to sequences of input and output actions. All of them are computable in linear time.

As all indicator values stem from totally ordered domains, the lexicographic order on signatures is total. When a new fragment is inserted, we can thus rely

on a (logarithmic) binary search for candidates $sig(F) = sig(G)$ that need to be checked for syntactic equivalence. The check itself is implemented in PETRUCHIO and we provide the option to hand over larger instances to a *graph isomorphism solver* that we integrated in black-box fashion [10, 17].

Experimental Evaluation The implementation encapsulates coverability checker and fixed-point engine into separate threads that run loosely coupled. We demonstrate its efficiency on the gsm handover procedure [18] and an automatic manufacturing system [16]. Note that HTS_P with a parametric number of transport vehicles yields a smaller net than the concrete model HTS_C with six of them, underpinning the need for unbounded verification techniques. For each model, we give loc, Petri net size (places, transitions, edges), and compile-time on an AMD Athlon 64 X2 Dual Core with 2.5 GHz.

Model	LOC	$\lvert\mathbf{P}\rvert$	$\lvert\mathbf{T}\rvert$	$\lvert\mathbf{E}\rvert$	t[s]
GSM	84	131	263	526	1.55
HTS_P	194	903	1103	3482	3.24
HTS_C	195	1912	3515	11881	15.7

References

1. P. A. Abdulla, A. Bouajjani, J. Cederberg, F. Haziza, and A. Rezine. Monotonic abstraction for programs with dynamic memory heaps. In *CAV*, volume 5123 of *LNCS*, pages 341–354. Springer, 2008.
2. P. A. Abdulla, K. Čerans, B. Jonsson, and Y.-K. Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Inf Comp*, 160(1–2):109–127, 2000.
3. N. Busi and R. Gorrieri. Distributed semantics for the π-calculus based on Petri nets with inhibitor arcs. *JLAP*, 78(1):138–162, 2009.
4. R. Devillers, H. Klaudel, and M. Koutny. A compositional Petri net translation of general π-calculus terms. *For Asp Comp*, 20(4–5):429–450, 2008.
5. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *TCS*, 256(1–2):63–92, 2001.
6. HAL. http://fmt.isti.cnr.it:8080/hal/.
7. D. Hirschkoff. An extensional spatial logic for mobile processes. In *CONCUR*, volume 3170 of *LNCS*, pages 325–339. Springer, 2004.
8. S. Joshi and B. König. Applying the graph minor theorem to the verification of graph transformation systems. In *CAV*, volume 5123 of *LNCS*, pages 214–226. Springer, 2008.
9. V. Khomenko, M. Koutny, and A. Niaouris. Applying Petri net unfoldings for verification of mobile systems. In *MOCA*, Bericht FBI-HH-B-267/06, pages 161–178. University of Hamburg, 2006.
10. V. Khomenko and R. Meyer. Checking π-calculus structural congruence is graph isomorphism complete. In *ACSD*, pages 70–79. IEEE, 2009.
11. B. König and V. Kozioura. Counterexample-guided abstraction refinement for the analysis of graph transformation systems. In *TACAS*, volume 3920 of *LNCS*, pages 197–211. Springer, 2006.
12. B. König and V. Kozioura. Incremental construction of coverability graphs. *IPL*, 103(5):203–209, 2007.
13. MODEL CHECKING KIT. http://www.fmi.uni-stuttgart.de/szs/tools/mckit/.
14. R. Meyer. A theory of structural stationarity in the π-calculus. *Acta Inf*, 46(2):87–137, 2009.

15. R. Meyer and R. Gorrieri. On the relationship between π-calculus and finite place/transition Petri nets. In *CONCUR*, volume 5710 of *LNCS*, pages 463–480. Springer, 2009.
16. R. Meyer, V. Khomenko, and T. Strazny. A practical approach to verification of mobile systems using net unfoldings. *Fund Inf*, 94(3–4):439–471, 2009.
17. NAUTY. http://cs.anu.edu.au/~bdm/nauty/.
18. F. Orava and J. Parrow. An algebraic verification of a mobile network. *For Asp Comp*, 4(6):497–543, 1992.
19. Petruchio. http://petruchio.informatik.uni-oldenburg.de.
20. A. Rensink. Canonical graph shapes. In *ESOP*, volume 2986 of *LNCS*, pages 401–415. Springer, 2004.
21. S. Sagiv, T. W. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM TOPLAS*, 24(3):217–298, 2002.
22. M. Saksena, O. Wibling, and B. Jonsson. Graph grammar modeling and verification of ad hoc routing protocols. In *TACAS*, volume 4963 of *LNCS*, pages 18–32. Springer, 2008.
23. Spatial Logic Model Checker. http://ctp.di.fct.unl.pt/SLMC/.
24. T. Wies, D. Zuffrey, and T. A. Henzinger. Forward analysis of depth-bounded processes. In *FoSSaCS*, volume 6014 of *LNCS*, pages 94–108. Springer, 2010.

A Details on the Algorithms

We provide more details on the fixed-point computation and the internal closure.

Fixed-Point Algorithm Fragments are particular π-calculus processes F, G, H from a set of fragments $\mathcal{F}$ [14]. The definition of the structural semantics relies on *Petri nets over fragments*, formalised as pairs $N = (S, T)$ from an overall set $\mathcal{PN}_{\mathcal{F}}$. Unlike standard ones, these nets have fragments as places, $S \subseteq \mathcal{F}$, and two kinds of transitions, $T = T_1 \cup T_2$. Transitions in T_1 are pairs (F, Q). They model an internal communication within F, which results in state Q. Transitions of the second form are triples (F_1, F_2, Q) that reflect interactions between F_1 and F_2.[1] Arcs between places and transitions are determined by their names, e.g. there is one arc from place F to (F, Q).

Petri nets over fragments $N = (S, T)$ and $N' = (S', T')$ are ordered by inclusion, $N \subseteq N'$ if $S \subseteq S'$ and $T \subseteq T'$. This turns $\mathcal{PN}_{\mathcal{F}}$ into a complete partial-order $(\mathcal{PN}_{\mathcal{F}}, \subseteq)$. The minimal element is the empty net, $\mathcal{N}_\emptyset = (\emptyset, \emptyset)$, and the least upper bound of a directed set of nets A is their union, $\sqcup A = \bigcup_{N \in A} N$, with $N \cup N' = (S \cup S', T \cup T')$.

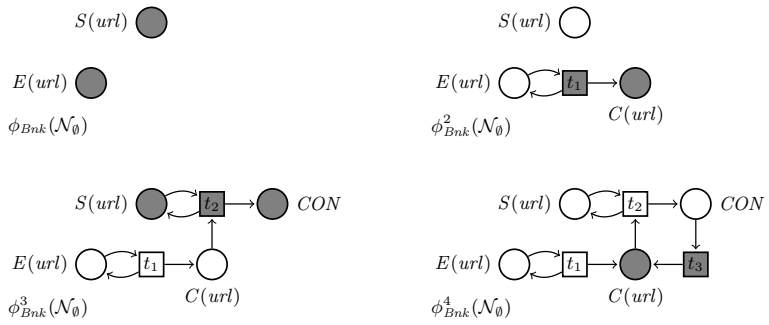


Fig. 2. Kleene iteration computing the structural semantics of the online banking service. Dark parts have been added in the last application of ϕ_{Bnk}.

In the structural semantics $\mathcal{N}_{\mathcal{S}}[\![Sys]\!] = (S, T)$, the places are the fragments reachable in *Sys* and the transitions are their communications. We imitate the translation by a continuous function $\phi_{Sys} : \mathcal{PN}_{\mathcal{F}} \to \mathcal{PN}_{\mathcal{F}}$ on nets over fragments. Given a net, the function adds the decomposition of the initial system state to properly start the Kleene iteration below. Then, for every fragment F that evolves to Q a transition (F, Q) is added. Similarly, a communication between F_1 and F_2 yields a transition (F_1, F_2, Q), provided the fragments are

[1] There are side conditions that avoid symmetric transitions but they are not important for the development here.

simultaneously markable from the decomposition of the initial state. Since ϕ_{Sys} is continuous, Kleene's theorem shows its least fixed-point to be the least upper bound of the sequence

$$\mathcal{N}_\emptyset \qquad \phi_{Sys}(\mathcal{N}_\emptyset) \qquad \phi_{Sys}(\phi_{Sys}(\mathcal{N}_\emptyset)) = \phi^2_{Sys}(\mathcal{N}_\emptyset) \qquad \dots$$

It is not difficult to show that this fixed-point is the structural semantics. Figure 2 illustrates the computation on the banking service. Initially, the server and the environment are present. They are added by an application of ϕ_{Bnk} to the empty net. The second application of ϕ_{Bnk} lets the environment generate a new client, which gives transition t_1. The server has no internal communication. In $\phi^2_{Bnk}(\mathcal{N}_\emptyset)$, client and server are *potential communication partners* $(C(url), S(url))$. A coverability check confirms they are simultaneously markable with one token on the environment and one on the server place. Their interaction is reflected by t_2. Then the thread inside $CON = \nu ip.(ip(dat).C(url) \mid \overline{ip}\langle bal\rangle)$ sends the balance to the client, t_3. Further applications of ϕ_{Bnk} do not change the net, $\phi_{Bnk}(\phi^4_{Bnk}(\mathcal{N}_\emptyset)) = \phi^4_{Bnk}(\mathcal{N}_\emptyset)$. The fixed point is found.

Internal Closure The internal closure avoids coverability checks by depth-first accelerations of the fixed-point computation. Formally, $cl(F)$ contains all places reachable from fragment F by internal communication, i.e., by transitions of the form (F', Q). In Figure 3, the closures of F and G are depicted light and dark, respectively. Place H happens to be in both sets.

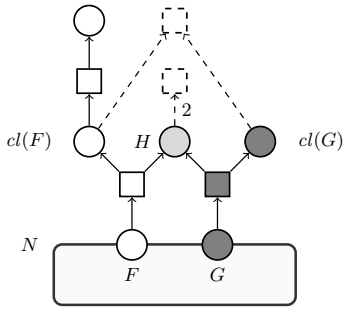


Fig. 3. Illustration of the internal closure acceleration.

If fragment F is reachable, all fragments in its internal closure are reachable and have to be added to the net. Furthermore, if F and G are simultaneously markable in the net N, then so are all fragment $F' \in cl(F)$ and $G' \in cl(G)$. Hence, communicating transitions between F' and G' can be added without further coverability queries. In Figure 3, they are drawn dashed and their postsets are omitted for simplicity. Note that from H there is an arc weighted two to a new transition (H, H, Q) that represents a communication between identical fragments. Since H is in two closures, it can in fact be covered by two tokens.