Petruchio: From Dynamic Networks to Nets

Roland Meyer[1] and Tim Strazny[2]

[1] LIAFA & CNRS
[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.

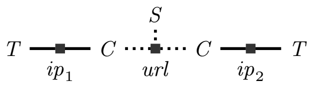
Run on a series of benchmarks, we routinely translate systems of two hundred lines of π-calculus code into Petri nets of around 1k places within seconds. The computability threshold lies around 90k transitions, which is in turn beyond the capabilities of latest net verification tools [13]. A concurrency 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].

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



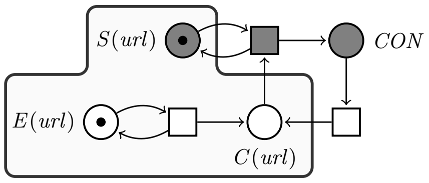


Fig. 1. π-calculus model *Bnk* of an online banking service (top left) and a reachable state represented as interconnection topology (bottom left). The structural semantics $\mathcal{N}_{\mathcal{S}}[\![Bnk]\!]$ is depicted to the right, *CON* abbreviates $\nu ip.(ip(dat).C(url) \mid \overline{ip}\langle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

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 $\mathcal{N}_{\mathcal{S}}[\![Bnk]\!]$ in Figure 1.

An *isomorphism* between the transition systems, $Sys =_{iso} \mathcal{N}_{\mathcal{S}}[\![Sys]\!]$, 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 $\mathcal{N}_{\mathcal{S}}[\![Sys]\!] = \mathcal{N}_{\mathcal{S}}[\![Sys']\!]$, 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, $\mathcal{N}_{\mathcal{S}}[\![Sys]\!] = \mathit{lfp}(\phi_{Sys})$. Thanks to continuity, we can compute it by iterating the function on the empty net, $\mathit{lfp}(\phi_{Sys}) = \sqcup\{\phi^n_{Sys}(\mathcal{N}_\emptyset) \mid n \in \mathbb{N}\}$. 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* $\mathit{cl}(F)$. 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 $F' \in \mathit{cl}(F)$ and $G' \in \mathit{cl}(G)$. 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* $\mathit{sig}(F)$. 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*, $\mathit{sig}(F) := \alpha(F).\beta(F)\ldots$ 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 $\mathit{sig}(F) = \mathit{sig}(G)$ 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**	$\|\mathbf{P}\|$	$\|\mathbf{T}\|$	$\|\mathbf{E}\|$	**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. Abdulla, P.A., Bouajjani, A., Cederberg, J., Haziza, F., Rezine, A.: Monotonic abstraction for programs with dynamic memory heaps. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 341–354. Springer, Heidelberg (2008)
2. Abdulla, P.A., Čerans, K., Jonsson, B., Tsay, Y.-K.: Algorithmic analysis of programs with well quasi-ordered domains. Inf. Comp. 160(1-2), 109–127 (2000)
3. Busi, N., Gorrieri, R.: Distributed semantics for the π-calculus based on Petri nets with inhibitor arcs. JLAP 78(1), 138–162 (2009)
4. Devillers, R., Klaudel, H., Koutny, M.: A compositional Petri net translation of general π-calculus terms. For Asp. Comp. 20(4-5), 429–450 (2008)
5. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! TCS 256(1-2), 63–92 (2001)
6. HAL, `http://fmt.isti.cnr.it:8080/hal/`
7. Hirschkoff, D.: An extensional spatial logic for mobile processes. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 325–339. Springer, Heidelberg (2004)
8. Joshi, S., König, B.: Applying the graph minor theorem to the verification of graph transformation systems. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 214–226. Springer, Heidelberg (2008)
9. Khomenko, V., Koutny, M., Niaouris, A.: Applying Petri net unfoldings for verification of mobile systems. In: MOCA, Bericht FBI-HH-B-267/06, pp. 161–178. University of Hamburg (2006)
10. Khomenko, V., Meyer, R.: Checking π-calculus structural congruence is graph isomorphism complete. In: ACSD, pp. 70–79. IEEE, Los Alamitos (2009)
11. König, B., Kozioura, V.: Counterexample-guided abstraction refinement for the analysis of graph transformation systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 197–211. Springer, Heidelberg (2006)
12. König, B., Kozioura, V.: 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. Meyer, R.: A theory of structural stationarity in the π-calculus. Acta Inf. 46(2), 87–137 (2009)
15. Meyer, R., Gorrieri, R.: On the relationship between π-calculus and finite place/transition Petri nets. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 463–480. Springer, Heidelberg (2009)
16. Meyer, R., Khomenko, V., Strazny, T.: 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. Orava, F., Parrow, J.: An algebraic verification of a mobile network. For. Asp. Comp. 4(6), 497–543 (1992)
19. Petruchio, `http://petruchio.informatik.uni-oldenburg.de`
20. Rensink, A.: Canonical graph shapes. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 401–415. Springer, Heidelberg (2004)
21. Sagiv, S., Reps, T.W., Wilhelm, R.: Parametric shape analysis via 3-valued logic. ACM TOPLAS 24(3), 217–298 (2002)
22. Saksena, M., Wibling, O., Jonsson, B.: Graph grammar modeling and verification of ad hoc routing protocols. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 18–32. Springer, Heidelberg (2008)
23. SPATIAL LOGIC MODEL CHECKER, `http://ctp.di.fct.unl.pt/SLMC/`
24. Wies, T., Zuffrey, D., Henzinger, T.A.: Forward analysis of depth-bounded processes. In: Ong, L. (ed.) FOSSACS 2010. LNCS, vol. 6014, pp. 94–108. Springer, Heidelberg (2010)