Model Checking for Nominal Calculi[⋆]

Gianluigi Ferrari, Ugo Montanari, and Emilio Tuosto

Dipartimento di Informatica, Largo Bruno Pontecorvo 3, 56127 Pisa – Italy

Abstract. Nominal calculi have been shown very effective to formally model a variety of computational phenomena. The models of nominal calculi have often infinite states, thus making model checking a difficult task. In this note we survey some of the approaches for model checking nominal calculi. Then, we focus on *History-Dependent automata*, a syntax-free automaton-based model of mobility. History-Dependent automata have provided the formal basis to design and implement some existing verification toolkits. We then introduce a novel syntax-free setting to model the symbolic semantics of a nominal calculus. Our approach relies on the notions of reactive systems and observed borrowed contexts introduced by Leifer and Milner, and further developed by Sassone, Lack and Sobocinski. We argue that the symbolic semantics model based on borrowed contexts can be conveniently applied to web service discovery and binding.

1 Summary

Model checking has been shown very effective for proving properties of system behaviour whenever a finite model of it can be constructed. The approach is convenient since it does not require formal proofs and since the same automaton-like model can accommodate system specification languages with substantially different syntax and semantics. Among the properties which can be checked, behavioural equivalence is especially important for matching specifications and implementations, for proving the system resistant to certain attacks and for replacing the system with a simpler one with the same properties.

Names have been used in process calculi for representing a variety of different informations concerning addresses, mobility links, continuations, localities, causal dependencies, security keys and session identifiers. When an unbound number of new names can be generated during execution, the models tend to be infinite even in the simplest cases, unless explicit mechanisms are introduced to allocate and garbage collect names, allowing the same states to be reused with different name meanings.

We review some existing syntax-free models for name-passing calculi and focus on *History-Dependent automata* (HD-automata), introduced by Montanari and Pistore in 1995 [61]. HD-automata [61, 62, 70] have been shown a suitable automata-based model for representing Petri nets, CCS with causality and localities and some versions of π-calculus [58, 74].

Different versions of HD-automata have been defined. The simplest version can be easily translated to ordinary automata, but possibly with a larger number of states. In a

[⋆] Work supported by European Union project PROFUNDIS, Contract No. IST-2001-33100.

second version, the states are equipped with name symmetries which further reduce the size of the automata. Furthermore, a theory based on coalgebras in a category of ”named sets” can be developed for this kind of HD-automata, which extends the applicability of the approach to other nominal calculi and guarantees the existence of the minimal automaton within the same bisimilarity class [63, 34].

HD-automata also constitute the formal basis upon which several verification toolkits have been defined and implemented. The front end towards the π-calculus and the translation algorithm for the simplest version of HD-automata have been implemented in the HAL tool [31, 32], which relies on the JACK verification environment [7] for handling the resulting ordinary automata. The minimisation algorithm, naturally suggested by the coalgebraic framework, has been implemented in the Mihda toolkit [35, 36] within the European project PROFUNDIS. Other versions of HD-automata can be equipped with algebraic operations, and are based on a algebraic-coalgebraic theory [60].

Here we propose a further instance handling the symbolic versions of nominal calculi, where inputs are represented as variables which are instantiated only when needed. As it is the case for logic programming unification, one would like the variables to be instantiated only the least possible, still guaranteeing that all behaviours are eventually explored. The approach we follow relies on the notion of reactive system and of observable borrowed contexts introduced by Leifer and Milner [52, 51] and further developed by Sassone, Lack and Sobocinski [75, 76, 49] using G-categories and adhesive categories. The reduction semantics of reactive systems is extended in order to introduce as borrowed contexts both the variable instantiations needed in the transitions and the ordinary π-calculus actions. It is argued that the symbolic semantics model based on borrowed contexts can be conveniently applied to web service discovery and binding.

2 Verification via Semantics Equivalence

In the last thirty years the application of formal methods to software engineering has generated techniques and tools to deal with the various facets of the software development process (see e.g. [19] and the references therein). One of the main advantages of exploiting formal techniques consists of the possibility of constructing *abstractions* that approximate behaviours of the system under development. Often, these abstractions are amenable to automatic verification of properties thus providing a support to the certification of software quality.

Among the different proposals, *verification via semantics equivalence* provides a well established framework to deal with the checking of behavioural properties. In this approach, checking behavioural properties is reduced to the problem of contrasting two system abstractions in order to determine whether their behaviours coincide with respect to a suitable notion of semantics equivalence. For instance, it is possible to verify whether an abstraction of the implementation is consistent with its abstract specification. Another example is provided by the *information leak* detection; in [39] the analysis of information flow is done by verifying that the abstraction of the system P is equivalent to another abstraction obtained by suitably restricting the behaviour of P. A similar idea has been exploited in [1] for the analysis of cryptographic protocols.

Bisimilarity [68] has been proved to be an effective basis for verification based on semantics-equivalence of system abstractions described in some process calculus, i.e. Milner's Calculus of Communicating Systems (CCS) [57]. Bisimilarity is a *co-inductive* relation defined over a special class of automata called *labelled transition systems*. A generic labelled transition system (LTS) describes the evolution of a system by its interactions with the external environment. The co-inductive nature of bisimulation provides an effective proof method to establish semantics equivalence: it is sufficient to exhibit a bisimulation relating the two abstractions. Bisimulation-based proof methods have been exploited to establish properties of a variety of systems such as communication protocols, hardware designs and embedded controllers. Moreover, they have been incorporated in several toolkits for the verification of properties. Indeed, finite state verification environments have enjoyed substantial and growing use over the last years. Here, we mention the Concurrency WorkBench [21], the Meije-FC2 tools [8] and the JACK toolkit [7] to cite a few. Several systems of considerable complexity have been formalised and proved correct by exploiting these semantics-based verification environments.

The advent of mobile computing and wireless communication together with the development of applications running over the Internet (*Global Computing Systems*) have introduced software engineering scenarios that are much more dynamic than those handled with the techniques discussed above. Indeed, finite state verification of global computing systems is much more difficult: in this case, even simple systems can generate infinite state spaces. An illustrative example is provided by the π-calculus [58, 74]. The π-calculus primitives are simple but expressive: channel names can be created, communicated (thus giving the possibility of dynamically reconfiguring process acquaintances) and they are subjected to sophisticated scoping rules. The π-calculus is the archetype of name passing or nominal process calculi. Nominal calculi emphasise the principle that name mechanisms (e.g. local name generation, name exchanges, etc.) provide a suitable abstraction to formally explain a wide range of phenomena of global computing systems (see e.g. [78, 41]). Moreover, nominal calculi provide a basic programming model that has been incorporated in suitable libraries or novel programming languages [22, 4]. Finally, the usefulness of names has been also emphasised in practice. For instance, Needham [65] pointed out the role of names for the security of distributed systems. The World Wide Web provides an excellent (perhaps the most important) example of the power of names and name binding/resolution.

Nominal calculi have greater expressive power than ordinary process calculi, but the possibility of dynamically generating new names leads also to a much more complicated theory. In particular, bisimilarity is not always a congruence even for the strong bisimilarity. Moreover, the ordinary, underlying LTSs are infinite-state and infinite branching, thus making verification via semantics equivalence a difficult task.

Bisimulation-based proof techniques for nominal calculi can be roughly divided into two main families. The first consists of the *syntax-based* approaches while the second refers to the *syntax-free* approaches. The former line of development represents the states of the LTS with their syntactic denotation, while in the latter the states are just items characterised by their properties and connections. We recall a few of the approaches of both families without the ambition of being exhaustive.

Among the syntax-based, the most efficient approaches for finite-state verification rely on symbolic semantics. Symbolic semantics [42, 6, 53], generalise standard operational semantics by keeping track of equalities among names: transitions are derived in the context of such constraints. The main advantage of the symbolic semantics is that it yields a smaller transition system. The idea of symbolic semantics has been exploited to provide a convenient characterisation of *open bisimilarity* [73] and in the design of the corresponding bisimulation checker, the *Mobility WorkBench* (MWB) [81]. The MWB adapts to the case of the π-calculus the *on-the-fly* approach of [30], where the bisimulation relation is constructed during the state space generation. The MWB checks for open bisimilarity in the case of (finite-control) π-calculus processes and has also been reworked to deal with the Fusion calculus [69]. To gain efficiency, the MWB has been extended in [44] with modules implementing certain bisimulation-preserving program transformations, the *up-to-techniques* (introduced in [72]). Symbolic semantics has been also exploited in the design of the MCC model checker for the π-calculus [82]. The key idea of the approach is to provide an encoding of π-calculus symbolic semantics as a logic programming system. It is important to emphasise that all the constructions of the symbolic semantics rely on an *external* metalanguage and on a theory to describe and reason about name equalities.

A different approach is the definition of semantic-based techniques where names have a central role and are explicitly dealt with. Basically, in these frameworks it is possible to allocate and garbage collect names, allowing the same names to be reused with different meanings. This alternative line of research explores models of name-passing calculi, regardless of their syntactic details and aims at providing uniform theories that can be used to handle a variety of calculi and semantics. A well studied approach is based on the so-called permutation model, whose ingredients are a set of names and an action of its group of permutations (renaming substitutions) on an abstract set [37, 40, 47, 63]. In this setting, transition systems for nominal calculi are constructed via suitable functors over the underlying category of names and permutations: the internal theory of names.

It is important to notice that these approaches are *syntax-free* and provide the abstract framework to capture the notions of name abstraction and fresh name that are needed to describe and reason about nominal calculi. The HD-automata [34, 63, 70] and indexed LTSs [17] are examples of syntax-free models of name process calculi developed following the permutation approach.

3 Model Checking

Probably, the most successful formal technique applied in practice in the verification of systems is *model checking* (we refer to [18] for a detailed introduction to this field). Roughly speaking, model checking is used to determine whether a system abstraction (expressed as an automata or a term of a process calculus) satisfies a property (expressed as a modal or temporal logic formula). In order to model check a system with respect to a given formula it is necessary to prove that the system is a model of the formula. Tools supporting model checking techniques have matured to be used in practice (e.g. the SPIN model checker [45, 46] and SMV [56]). Recently, these techniques have been adopted to verify properties of programs written in high level programming languages like C++ and Java (e.g. JavaPathFinder [10], BANDERA [23], SLAM [3] and BLAST [43]).

Model checking presents several advantages. It is completely automatic, provided that finiteness of the system (the model) is guaranteed. Usually, it provides counterexamples when a system does not satisfy the property. This gives information on the design choices that have lead to the implementation errors. Finally, it is possible to obtain very high efficiency by exploiting refined data structures (e.g. BDDs), or symbolic techniques.

While modal and temporal logics have been proved suitable to express many properties of interest of concurrent systems, similar logics for global computing systems are still lacking. Only recently a new class of modal logics, *spatial logics* [15, 16], has been introduced to address the characterising issues of global computing. In our opinion, this explains why traditionally model checking has been exploited on foundational models for global computing only for limited fields and has not been fully applied to the general setting.

Without the ambition of being exhaustive, we now review some of the approaches to model check properties of nominal calculi. The MWB provides a model checking functionality. This is based on the implementation of the tableau-based proof system [25, 26] for the $\pi - \mu$-calculus, an extension of the propositional μ-calculus in which it is possible to express name parameterisation and quantifications over names. The MCC system also provides a model checking facility for the $\pi - \mu$-calculus.

The *HD-automata Laboratory* (HAL) [32] supports verification by model checking of properties expressed as formulae of a suitable modal logic, a high level logic with modalities indexed by π-calculus actions. This logic, although expressive enough to describe interesting safety and liveness properties of π-calculus specifications, is less expressive than the $\pi - \mu$-calculus. The construction of the HAL model checker takes direct advantage of the finite representation of π-calculus specifications presented in [61]. In particular, a HAL module translates these logical formulae into classical modal logic formulae and the translation is driven by the finite state representation of the system (the π-calculus process) to be verified.

The most relevant examples of application of model checking techniques and nominal calculi are those of the verification of security protocols [55, 20]. Several prototypical tools based on nominal calculi have been in fact designed and implemented [59, 54, 27, 38]. Indeed, nominal calculi provide a solid formal context for expressing many facets of cryptographic protocols in natural way. For instance, many authentication protocols rely on *nonce-challenges* where a fresh sequence of bit must be generated; the correctness of these protocols relies on the uniqueness of the nonces used in a given session. This can be easily modelled in nominal calculi, e.g. the π-calculus, where freshly generated names can be expressed and dealt with. An advantage of using model checking is that, when the protocol does not satisfy the security property, then the counterexample is the attack that an intruder could perform.

The main drawback of these approaches is that they require a finite state space while, in general, the generation of fresh names easily leads to infinite state spaces, if no countermeasure for garbage-collecting and reusing names is adopted. In practice, this problem has been faced by imposing strong conditions that limit the generality of the analysis. In particular, *finitary* systems, namely systems with infinite behaviour which can be finitely represented, are not considered. For instance, the analysis are performed on instances of protocols where only a limited number of participants is *apriori* fixed and in general recursion or iteration is forbidden. Hence, model checking security properties for nominal calculi can only deal with protocol sessions where a finite number of participants run in parallel and all the participants are non-recursive processes. Recently, symbolic ad-hoc model checkers have been proposed to overcome these issues e.g., [5, 80, 9, 2]. Despite the technical differences, all these approaches check a given property by generating a "symbolic" state space, where states collect constraints over the names involved in the execution. If there is a reachable state that violates the property, but whose constraints hold, then an attack is found. The symbolic techniques exploited in these approaches enforce efficiency both in the size of the generated state space and in the visit of it, but they still require finite state space.

4 History-Dependent Automata

History Dependent automata (HD-automata in brief) are one of the proposal based on the syntax-free approach. HD-automata are an operational model for history dependent formalisms, namely those formalisms accounting for systems whose behaviour at a given time might be influenced by some "historical" information which is too expensive to be included explicitly in the states. HD-automata allow for a compact representation of agent behaviour by collapsing states differing only for the renaming of local names and encompass the main characteristics of name-passing calculi, namely creation/deallocation of names. Basically, HD-automata associate a "history" to the names of the states appearing in the computation, in the sense that it is possible to reconstruct the associations which have led to the state containing the name. Clearly, if a state is reached in two different computations, different histories could be assigned to its names. Process calculi exhibiting causality, localities and mobility, and Petri nets, can be translated (preserving bisimilarity) to HD-automata [70].

Different versions of HD-automata have been defined [70, 62, 63, 34]. When handling causality, locality and the link mobility exhibited by the synchronous π-calculus without matching, the simplest version can be easily translated to ordinary automata. However, in general, a larger number of states is necessary for representing HD-automata with ordinary automata. The front-end towards the π-calculus and the translation algorithm have been implemented in the HAL toolkit, which relies on the JACK verification environment for handling the resulting ordinary automata.

In a second version, states of HD-automata are equipped with name symmetries which further reduce the size of the automata [63] and which guarantee the existence of the minimal realization. The minimal automata are computed using a partition refinement algorithm [34]. They have a very important practical fall-out: for instance, the problem of deciding bisimilarity is reduced to the problem of computing the minimal transition system [66, 29, 48]. Moreover, the minimal automaton is indistinguishable from the original one with respect to many behavioural properties (e.g., bisimilarity) and properties expressed in most modal or temporal logics. The minimisation algorithm, naturally suggested by the coalgebraic framework, has been implemented in the Mihda toolkit [36] within the European project PROFUNDIS. Other versions of HD-automata can be equipped with algebraic operations [60], thus relying on an algebraic-coalgebraic, namely bialgebraic, theory.

Similarly to ordinary automata, HD-automata consist of states and labelled transitions, their peculiarity being that states and transitions are equipped with names which are no longer dealt with as syntactic components of labels, but become an explicit part of the operational model. Noteworthy, names in states of HD-automata have *local meaning* which requires a mechanism for describing how names correspond each other along transitions.

Graphically, we can represent such correspondences using "wires" that connect names of label, source and target states of transitions as in Figure 1, where a transition from source state s to destination state d is depicted. State s has three names, $s1$, $s2$ and $s3$ while d has two names $d1$ and $d2$ which correspond to name $s1$ of s and to the new name $\star$, respectively. The transition is labelled by *lab* and exposes two names: name $l1$ and $\star$ the former corresponding to name $s2$ of s and the latter to a fresh name denoted as $\star$. Notice that name $s3$ is "deallocated" along such transition.

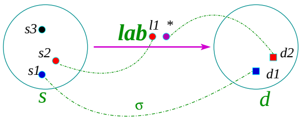


Fig. 1. A HD-automaton transition

4.1 Minimising HD-automata: an informal presentation

We report the formal definitions for *named sets* and *named functions* for representing finite HD-automata. These are the basic concepts upon which the partition refinement algorithm for HD-automata has been defined. For the sake of conciseness we give here only an incomplete definition. The interested reader is referred to [36] for a full presentation.

Definition 1 (Named sets). *Let $\mathcal{N}$ be a denumerable set of names. A* named set *is a pair $\langle Q, \mathrm{g} \rangle$ where Q is a totally-ordered set and $\mathrm{g} : Q \to \bigcup_{N \in \wp_{fin}(\mathcal{N})} \mathrm{sym}(N)$ assigns a (finite) group of permutations over a finite set of names to elements in Q. For $q \in Q$, $|q|$ denotes the* carrier *of q defined as $\mathrm{dom}(\rho)$, where $\rho \in \mathrm{g}(q)$.*

Definition 2 (Named functions). *Let $\mathcal{N}^\star$ be $\mathcal{N} \cup \{\star\}$ where $\star$ is an element not in $\mathcal{N}$. Given two named sets $\langle Q, \mathrm{g} \rangle$ and $\langle Q', \mathrm{g}' \rangle$, a* named function *$H : \langle Q, \mathrm{g} \rangle \to \langle Q', \mathrm{g}' \rangle$ consists of a pair of functions $\langle h, \Sigma \rangle$ where $h : Q \to Q'$ and $\Sigma : Q \to \wp_{fin}(\mathcal{N} \to \mathcal{N}^\star)$ such that for all $q \in Q$ and $\sigma \in \Sigma(q)$*

- *σ is injective, $\sigma(|h(q)|) \subseteq |q| \cup \{\star\}$ and $\sigma|_{\mathcal{N} \setminus |h(q)|}$ is the identity;*
- *$\sigma; \mathrm{g}(q) \subseteq \Sigma(q)$;*
- *$\mathrm{g}(h(q)); \sigma = \Sigma(q)$.*

Named sets and functions form a category, ***NS***, since named functions can be composed and identity named functions can be easily defined (see [36] for details). Given a set of labels L, if $\wp_{\text{fin}}(\_)$ is the finite power set functor on category Set, we define the functor $\wp_L$ on named sets as $\wp_L(\langle Q, \mathrm{g}\rangle) = \langle \wp_{\text{fin}}(L \times Q), \mathrm{g}'\rangle$ where $\mathrm{g}'(B)$ contains all those permutations ρ such that $B\rho = B$ ($B\rho$ is element-wise application of ρ to B).

Definition 3. *A* HD-automaton *over L is a coalgebra for the functor $\wp_L$.*

The most important operation for minimising HD-automata is the *normalisation* which removes *redundant* transitions. In nominal calculi, redundancy is strictly connected to the concept of *active names*. A name n is *inactive* for an agent P if it is not used in the future behaviour of P.

In π-calculus, if P is bisimilar to $(\nu\, n)P$ we say that n is inactive in P (otherwise n is *active* in P) and a transition $P \xrightarrow{xn} Q$ is *redundant* (in the early semantics of π-calculus) when n is inactive in P. Deciding whether a name is active is as difficult as deciding bisimilarity. The importance of redundancy emerges when we try to establish the equivalence of states that have different numbers of free names. For instance consider $P \overset{\text{def}}{=} x(u).\nu\, v(\bar{v}z + \bar{u}y)$ and $Q \overset{\text{def}}{=} x(u).\bar{u}y$, which differ only for a deadlocked alternative. They are bisimilar only if, for any name substituted for u, their continuations remain bisimilar. However, the input transition $P \xrightarrow{xz}$ cannot be matched by Q when considering only the *necessary* input transitions of agents, namely those where the acquired name is either one of the free names of the agent or a fresh name (as required for a finite representation of the transition system). Thus, unless the above transition of P is recognised as redundant and removed, the automata for P and Q would not be bisimilar. Redundant transitions occur when LTSs of π-calculus processes are compiled to HD-automata and are removed during the minimisation algorithm, since it is not possible to leave them out at compiling time[1].

The minimisation algorithm relies on functor T consisting of the composition of the normalisation functor and $\wp_L$. Consider a T-coalgebra $\langle D, K : D \to T(D)\rangle$, the minimisation algorithm is defined by the two equations below.

$$H_{(0)} \overset{\text{def}}{=} \langle q \mapsto \bot, q \mapsto \emptyset\rangle, \quad \text{where } \mathrm{dom}(H_{(0)}) = D \tag{1}$$

$$H_{(i+1)} \overset{\text{def}}{=} K; T(H_i). \tag{2}$$

In words, all the states of automaton K are initially considered equivalent, indeed, the kernel of H_0 gives rise to a single equivalence class containing the whole $\mathrm{dom}(K)$. At the generic $(i+1)$-th iteration, the image through T of the i-th iteration is composed with K as prescribed in (2). The algorithm stops when the fixpoint $\overline{H}$ of (2) is reached. Then $\overline{H}$ is the unique final coalgebra morphism and states mapped together by it are bisimilar.

Theorem 1 (Convergence [36]). *The iterative algorithm described by (1) and (2) is convergent on finite state automata.*

[1] In general, deciding whether a free input transition is redundant or not is equivalent to decide whether a name is active or not; therefore, it is as difficult as deciding bisimilarity.

4.2 The PROFUNDIS web

In the last years distributed applications over the World-Wide Web, have attained wide interest. Recently, the Web is exploited as a *service distributor* and applications are no longer monolithic but rather made of components (i.e., services). Applications over the Web are developed by combining and integrating Web services. The Web service framework has emerged as the standard and natural architecture to realize the so called *Service Oriented Computing* (SOC) [24, 67]. In [33] a Web-service infrastructure was developed integrating verification toolkits for checking properties of mobile systems and related higher-level toolkits for verifying security protocols. The development of the verification infrastructure has been performed inside the PROFUNDIS project (see URL `http://www.it.uu.se/profundis`) within the Global Computing Initiative of the European Union. For this reason we called it the *PROFUNDIS WEB*, PWeb for short. The current prototype implementation of the PWeb infrastructure can be exercised on-line at the URL `http://jordie.di.unipi.it:8080/pweb`.

Beyond the current prototype implementation, we envisage the important role that will be played by PWeb service coordination. Indeed, service coordination provides several benefits:

- *Model-based verification*. The coordination rules impose constraints on the execution flow of the verification session thus enabling a *model-based* verification methodology where several descriptions are manipulated together.
- *Modularity*. The verification of the properties of a large software system can be reduced to the verification of properties over subsystems of manageable complexity: the coordination rules reflect the semantic modularity of system specifications.
- *Flexibility*. The choice of the verification toolkits involved in the verification session may depend on the specific verification requirements.

The PWeb implementation has been conceived to support reasoning about the behaviour of systems specified in some dialect of the π-calculus. It supports the dynamic integration of different verification techniques (e.g. standard bisimulation checking and symbolic techniques for cryptographic protocols). The PWeb integrates several independently-developed toolkits, e.g., Mihda [35, 36] and several tools for verifying cryptographic protocols, like TRUST [80] and STA [5]. The PWeb has been designed by targeting also the goal of extending available verification environments (Mobility Workbench [81], HAL [31, 32]) with new facilities provided as Web services.

The core of the PWeb is a *directory service*. A PWeb directory service is a component that maps the description of the Web services into the corresponding network addresses and has two main facilities: the `publish` facility, invoked to make a toolkit available as Web service, and the `query` facility, used to discover available services. For instance, Mihda publishes the `reduce` service which accepts a (XML description of) HD-automaton describing the behaviour of a π-calculus agent. Once invoked, `reduce` performs the minimisation of the HD-automaton.

The service discovery mechanisms are exploited by the `trader` engine which manipulates pools of services distributed over several PWeb directory services. It can be used to obtain a Web service of a certain type and to bind it inside the application. The `trader` engine gives to the PWeb directory service the ability of finding and binding web services at run-time without "hard-coding" the name of the web service inside the application code. The following code describes the use of a simple trader for the PWeb directory.

```
import Trader
offers = Trader.query( "reducer" )
mihda = offers[ 0 ]
```

The code asks the trader for a reduce service and selects the first of them. The trader engine allows one to hide network details in the service coordination code. A further benefit is given by the possibility of replicating the services and maintaining a standard access modality to the Web services under coordination.

The fundamental technique enabling the dynamic integration of services is the separation between the service facilities (what the service provides) and the mechanisms that coordinate the way services interact (service coordination). An example of service coordination for checking whether a process A is a model for a formula F is as follows

```
hd = mihda.compile( A )
reduced_hd = mihda.reduce( hd )
reduced_hd_fc2 = mihda.Tofc2( reduced_hd )
aut = hal.unfold( reduced_hd_fc2 )
if hal.check( aut, F ):
    print 'ok'
else:
    print 'ko'
```

Variables mihda and hal have been linked by the trader engine to the required services (acquired as illustrated before). Now, the compile service of mihda is invoked yielding an HD-automaton (stored in hd). Next, hd is minimised by invoking the service reduce of Mihda; and afterward it is transformed into the FC2 format by a HAL service. Finally, the HAL service unfold generates an ordinary automaton from the FC2 representation of the automaton and prints a message which depends on whether the system satisfies the formula F or not. This is obtained by invoking the HAL model checking facility check.

5 A Borrowed Context Semantics for the Open π-calculus

The version of the π-calculus implemented in the Mihda toolkit does not rely on a symbolic semantics. This fact makes unnecessarily large the number of states, due to the existence of different input transitions for different instantiations of the input variable. While a symbolic semantics for a syntax-based version of HD-automata for the open π-calculus has been defined in [71], it might be convenient to define a symbolic semantics for the ordinary syntax-free HD-automata. More generally, in Service Oriented Computing (SOC) [24, 67] one would like to have more sophisticated mechanisms than service call and parameter passing for modelling the phase of service discovery and binding. The SOC paradigm is the emerging technology to design and develop global computing systems: several research activities have addressed the theoretical foundations of the SOC paradigm by exploiting formal frameworks based on process calculi [12, 50, 14, 11] (see also [79] for an informal presentation on the usefulness of nominal calculi to design workflow business processes).

When looking for a generalisation of parameter passing, logic programming unification comes to mind, or rather constraint programming, when service level agreements involve nonfunctional issues. When the binding occurs, not only the callee is instantiated, but also the caller. The instantiation that must be applied to the caller is formally analogous to a missing context that must be borrowed by a process in order to undergo a reduction. In this line of thought, some recent works about systematic methods for deriving LTSs from reduction rules look relevant. In particular, the approach we follow relies on the notion of reactive system, introduced by Leifer and Milner [52, 51] and further developed by Sassone, Lack and Sobocinski [75, 76, 49] using G-categories and adhesive categories.

In this section we will consider a simplified version of open π-calculus and we will develop a semantics for it using the notion of reactive systems. While the corresponding bisimilarity semantics turns out to be finer, we think that this exercise shows the feasibility of employing context borrowing for modelling symbolic semantics. The generality of the reactive system approach gives some hope that interesting abstractions of the SOC paradigm could also be modelled that way. Note however that the transition system which can be derived from reactive rules in our development is not really suitable for a HD-automata implementation, since new names are never forgotten, thus making the transition systems infinite in all but the most trivial cases. We comment in Section 6 about possible solutions of this problem.

5.1 Open π-calculus

One of main peculiarities of the π-calculus is the richness of its observational semantics. Initially, it came equipped with the *early* and the *late* observational semantics [58] which differ each other in the way they deal with name instantiation. Symbolic semantics [42] generalises standard operational semantics by keeping track of equalities among names: transitions are derived in the context of such constraints. The main advantage of the symbolic semantics is that it yields a smaller transition system. The idea of symbolic semantics has been exploited to provide a finitary characterisation of *open bisimilarity* [73] which, differently from the early and the late semantics, is a congruence with respect to the contexts of the π-calculus.

We consider a subset of the π-calculus without neither matching nor restriction operators. Given a numerable infinite and totally ordered set of *names* $\mathcal{N} = \{a_1, a_2, \ldots\}$, the set **P** of π_{-} *processes* is defined by the grammar

$$p, q ::= \mathbf{0} \;\big|\; \mu.p \;\big|\; p \mid q \;\big|\; p + q \;\big|\; p! \qquad\qquad \alpha ::= \overline{a}b \;\big|\; a(b).$$

As usual, name a is free in $\overline{a}b$ and $a(b)$, while b is free just in the former case and bound in the latter. Moreover, a is called the *subject* and b the *object* of the action. Considering $a(b).p$, the occurrences of b in p are bound, *free names* are defined as usual and $\mathrm{fn}(p)$ indicates the set of free names of process p. Differently than in the full π-calculus, only

$$(\textsc{pre})\ \alpha.p \xrightarrow{\alpha} p \qquad\qquad (\textsc{sum})\ \frac{p \xrightarrow{\mu} p'}{p+q \xrightarrow{\mu} p'}$$

$$(\textsc{par})\ \frac{p \xrightarrow{\mu} p'}{p \mid q \xrightarrow{\mu} p' \mid q}\ \text{if } \mathrm{bn}(\mu)\cap\mathrm{fn}(q)=\emptyset \qquad (\textsc{com})\ \frac{p \xrightarrow{\overline{a}b} p' \quad q \xrightarrow{a'(c)} q'}{p \mid q \xrightarrow{a=a'} p' \mid q'\{^b/_c\}}$$

$$(\textsc{rep})\ \frac{p \mid p! \xrightarrow{\mu} q}{p! \xrightarrow{\mu} q}$$

Table 1. Semantics of π_-

the input prefix binds names. Processes are considered equivalent up-to α-renaming of bound names.

The operational rules for the semantics of π_- are those reported in Table 1 together with the symmetric rules for (PAR) and (SUM). The rules specify an LTS whose labels (denoted as μ) are either actions or *fusions*. The only non-standard rule is (COM) which states that an output $\overline{a}b$ and an input $a'(c)$ can synchronise provided that a and a' are fused. Notice that, if a and a' are the same, $a = a'$ is the identity fusion, denoted as ε, which corresponds to the usual silent action τ.

The transition system of π_- resulting from specification rules in Table 1 is the same as the one obtained by applying the LTS rules of [73] to π_-. The only differences between the two LTSs are in the syntax of the labels and in the rule (COM). In [73] the labels are pairs (M,μ) or (M,τ) where M are sequences of fusions. It is easy to see that our label μ corresponds to (μ,τ) if μ is a fusion label and to $(\emptyset,\mu)$ if it is an action label. The communication rule of [73] is

$$\frac{p \overset{(M,\overline{a}b)}{\longmapsto} p' \qquad q \overset{(N,a'(c))}{\longmapsto} q'}{p \mid q \overset{(L,\tau)}{\longmapsto} p' \mid q'\{^b/_c\}} \qquad\qquad L = \begin{cases} MN[a=a'], & \text{if } a \neq a' \\ MN, & \text{if } a = a' \end{cases}$$

which resembles rule (COM) of Table 1. However it is considerably more complex since it must also collect the fusions due to matchings.

Proposition 1. *Let $p \in \mathbf{P}$ be a π_- process, then $p \xrightarrow{\mu} q$ if, and only if, the same transition can be derived from the transition system in [73] (changing μ with the corresponding label of [73]).*

Proof. The ($\Rightarrow$) part trivially follows by induction on the length of the proof of $p \xrightarrow{\mu} q$. The ($\Leftarrow$) part follows by observing that the length of the fusions in labels of [73] is one, since π_- lacks the matching operator. □

We recast the definition of *open bisimulation* given in [73] for π_-.

Definition 4 (Open bisimulation). *A symmetric relation $S \subseteq \mathbf{P} \times \mathbf{P}$ is an* open bisimulation *if whenever pSq,*

- *if $p \xrightarrow{\alpha} p'$ then there is q' such that $q \xrightarrow{\alpha} q'$ and $p'Sq'$;*
- *if $p \xrightarrow{\varepsilon} p'$ then there is q' such that $q \xrightarrow{\varepsilon} q'$ and $p'Sq'$;*
- *if $p \xrightarrow{a=b} p'$ then there is q' such that $(q \xrightarrow{a=b} q' \vee q \xrightarrow{\varepsilon} q') \wedge \sigma_{a=b}(p')S\sigma_{a=b}(q')$,*

where $\sigma_{a=b}$ is a substitution that maps a to b (or viceversa*) and leaves the other names unchanged. Two processes p and q are* open bisimilar*, written $p \sim q$, when there is an open bisimulation relating them.*

In order to compare the ordinary bisimilarity $\sim$ with the one arising from the Leifer and Milner approach, it is convenient to introduce an additional bisimilarity for π_{-}.

Definition 5 (Syntactical bisimilarity). *The* syntactical *bisimilarity relation $\backsimeq$ for π_{-} is obtained by simplifying the last condition of Definition 4 with*

$$\textit{if } p \xrightarrow{a=b} p' \textit{ then there is } q' \textit{ such that } q \xrightarrow{a=b} q' \textit{ and } \sigma_{a=b}(p') S \sigma_{a=b}(q').$$

It is immediate to see that $\backsimeq$ is finer than or equal to $\sim$. In fact its conditions for matching transition labels are more demanding than those for $\sim$.

Theorem 2 (Open vs. syntactical bisimilarity). *We have $\backsimeq \subseteq \sim$.*

An equivalence relation relating terms of an algebra is said a *congruence* if it is preserved by all the operation of the algebra, or, equivalently, if it is preserved in all the contexts of the language. In [73], $\sim$ has been proven to be a congruence for the π-calculus.

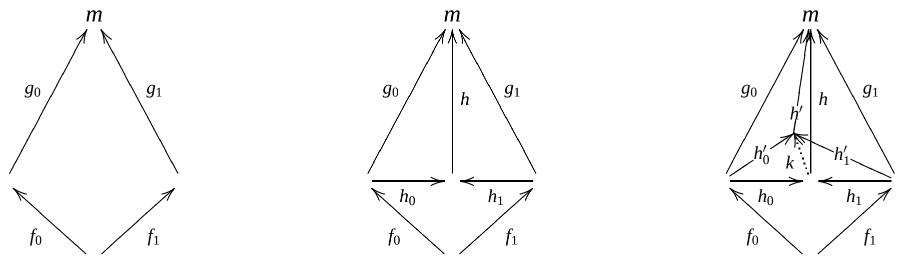
5.2 Reactive systems

A systematic method for deriving bisimulation congruence from reduction rules has been proposed by Leifer and Milner in [52, 51], on turn inspired by [77], where the idea of interpreting $p \xrightarrow{c} q$ as "in the context c, p reacts and becomes q" has been proposed. Also, the approach of observing contexts imposed on agents at each step has been introduced in [64], yielding the notion of *dynamic bisimilarity*. Following [28], we will call *borrowed context* the context c. The basic idea of [52, 51] is to express "minimality" conditions for electing the context c among the (possibly infinite) ones that allow p to react. These conditions have been distilled by [52] in the notion of *relative push-out* (RPO) in categories of *reactive systems*. The RPO construction is reminiscent of the unification process of logic programming, which in fact can be given an interactive semantics in much the same style [13].

We want to apply this approach to a reduction semantics of π_{-} that reflects its LTS semantics, therefore, we collect here the main definitions and results of the RPO approach. We remark that Definitions 6, 7, 8 and Theorem 3 are borrowed from [52, 51] (aside from some minor notational conventions).

Let $\mathbf{C}$ be an arbitrary category whose arrows are denoted by f, g, h, k and whose objects by m, n. Hereafter, $f;g$ will indicate arrow composition.

Definition 6 (Relative push-out and Idem push-out). *Consider the commuting diagram in Figure 2(a) consisting of $f_0;g_0 = f_1;g_1$. A triple $\langle h_0, h_1, h \rangle$ is an* RPO *if diagram in Figure 2(b) commutes and for any triple $\langle h'_0, h'_1, h' \rangle$ satisfying $f_0;h'_0 = f_1;h'_1$ and $h'_i;h' = g_i$, for $i = 0,1$ there exists a unique k such that diagram Figure 2(c) commutes. Diagram (a) is an* idem push-out (IPO) *if $\langle g_0, g_1, id \rangle$ is an RPO.*



(a) (b) (c)

Fig. 2. Diagrams for Definitions 6

Definition 7 (Reactive system). *A* reactive system *is a category* **C** *with the following extra components:*

- *a distinguished (not necessarily initial) object* $\star$*;*
- *a set of pairs of arrows* $(l : \star \to m, r : \star \to m)$ *called* reaction rules*;*
- *a subcategory* **D** *of* reactive contexts *with the property that if* $d;d'$ *is an arrow of* **D***, then both* d *and* d' *are arrows in* **D***.*

The IPO construction yields the definition of labelled transition out of a reduction semantics and the corresponding observational semantics.

Definition 8 (Labelled transition and bisimulation). *We write* $a \xrightarrow{f} a'$ *iff there exist a reaction rule* (l,r) *and a reactive context* d *such that*

$$\begin{tikzcd} & {} & \\ {} \arrow[ur, "f"] & & {} \arrow[ul, "d"'] \\ & \star \arrow[ul, "a"] \arrow[ur, "l"'] & \end{tikzcd}$$

is an IPO and $a' = r;d$*.*

A symmetric binary relation $S \subseteq \bigcup_m \mathbf{C}[\star,m] \times \bigcup_m \mathbf{C}[\star,m]$*, where* $C[x,y]$ *is the set of all the arrows from* x *to* y *of category* C*, is a* bisimulation *over* $\xrightarrow{f}$ *iff for* $(a,b) \in S$*, if* $a \xrightarrow{f} a'$ *then there is* b' *such that* $b \xrightarrow{f} b'$ *and* $(a',b') \in S$*.*

The central result of [52] can be stated as follows:

Theorem 3. *The largest bisimulation over* $\xrightarrow{f}$ *is a congruence provided that* **C** *has all* redex-RPOs.

The category **C** has all *redex-RPOs* when for all reaction rules (l,r), all arrow a, f and all contexts d such that $a;f = l;d$ then the square

$$\begin{tikzcd} & {} & \\ {} \arrow[ur, "f"] & & {} \arrow[ul, "d"'] \\ & \star \arrow[ul, "a"] \arrow[ur, "l"'] & \end{tikzcd}$$

has an RPO.

5.3 A Reactive system for the open π-calculus

We shall specify a reactive system semantics for π_- taking actions and name substitutions as reactive contexts and by defining rules in such a way that the LTS will be essentially the same as the one defined in Section 5.1. However, the observational semantics resulting from the RPO approach considers labels as purely syntactical items and transitions can match only if they have identical labels. In the definition of open bisimilarity, instead, a proper fusion can be matched by an ε label. Thus it cannot be expected that the two bisimilarity relations coincide. In fact, we will show that the bisimilarity arising from the RPO approach is finer than open bisimilarity.

The reduction semantics of π_- is specified with rules of the form $P;\mu \rightarrow q$, where μ is an action or a fusion, $q \in \mathbf{P}$ and P is a *normalised process* (formally defined below). A rule $P;\mu \rightarrow q$ corresponds to a π_- transition $P \xrightarrow{\mu} q$, the only difference being that in the reactive system approach processes must be typed by (a natural number larger or equal than) the largest index of their free variables. Normalised processes can be thought of as being processes where all the occurrences of free variables are replaced by different variables $\{a_1,\dots,a_n\}$ ordered in some standard way. Normalised processes give a logic programming flavour to the reduction semantics. In fact, they are reminiscent of predicate symbols, while processes correspond to goals: as goals are instantiations of predicate symbols, any process $p \in \mathbf{P}$ can be regarded as the instantiation of a normalised process P. This amounts to say that, whenever p and $P;\mu$ (i.e., the instance and the head of the clause) unify, then a transition for p can be deduced. They unify whenever P is the normalised process of p. Moreover, the label is the borrowed context, which turns out either to be μ whenever μ is an action or to be a fusion not implied by the substitution mapping P to p, or else to be ε if it is implied.

Let $p \in \mathbf{P}$, we assume given two functions $\hat{p}$ and σ_p such that

$$\mathrm{fn}(\hat{p}) = \{a_1,\dots,a_n\}, \quad \hat{\hat{p}} = \hat{p}, \quad p = \sigma_p(\hat{p}), \quad p = \sigma(q) \implies \hat{p} = \hat{q} \wedge \sigma_p = \sigma \circ \sigma_q,$$

where $\sigma_p : \mathrm{fn}(\hat{p}) \rightarrow \mathrm{fn}(p)$ and $\sigma : \mathrm{fn}(q) \rightarrow \mathrm{fn}(p)$ are surjective name substitutions homomorphically extended to π_- agents ($\sigma(\_)$ stands for the extension of σ to agents). It is easy to show that $\hat{p}$ is a linear process, namely each free variable occurs exactly once. Indeed, let $x \in \mathcal{N}$ occur twice in $p \in \mathbf{P}$ and assume by absurd that $\hat{p} = p$. Now, consider $p' \in \mathbf{P}$ to be the term obtained by replacing in p the first and the second occurrence of x with y and z, respectively. Then $p = \sigma(p')$, where $\sigma = \{y \mapsto x, z \mapsto x\}$, thus by definition, $\hat{p} = \widehat{p'}$. But there is no $\sigma_{p'}$ such that $p' = \sigma_{p'}(\widehat{p'}) = \sigma_{p'}(p)$.

Notice that $\hat{\_}$ and $\sigma_\_$ only involve syntactical aspects of agents, therefore they can be easily defined on the syntax trees of π_-. For instance, $\hat{p}$ might be defined as the agent having the same syntax tree of p where the i-th leaf is named by a_i, assuming that leaves are ordered according to a depth-first visit: substitution σ_p is defined accordingly. The order of leaves is arbitrary and different definitions might be possible, however, all of them differ only for a permutation of (the indexes of) $\mathrm{fn}(\hat{p})$.

Definition 9 (Normalised processes). *The processes that are fixpoints of $\hat{\_}$ are the* normalised *processes and are ranged over by P.*

Before defining **PAC**, the category we work with, we specify its (basic) arrows where the underlying objects are elements of the set $\omega_\star = \omega \cup \{\star\}$ consisting of the natural numbers plus a distinguished element $\star$.

Definition 10 (Basic arrows). *We define the following* basic arrows.
A normalised agent arrow $P_m : \star \to m$ *is a pair consisting of a normalised process* P *and a natural number* $m \in \omega$ *such that, for any* $a_n \in \mathrm{fn}(P)$*,* $n \leq m$*. We write* P *instead of* P_m *when* $\mathrm{fn}(P)$ *contains exactly* m *names.*
A fusion arrow from m to n *is a surjective substitution from* $\{a_1, \ldots, a_m\}$ *to* $\{a_1, \ldots, a_n\}$ *written as* $\sigma : m \to n$*.*
Action arrows *are* π_- *actions parameterised on* ω*, more precisely*

$$\overline{a}_i^m a_j : m \to m \qquad a_i^m : m \to m+1 \qquad i, j \leq m$$

that respectively correspond to output and input transitions with the object name in the latter case being a_{m+1}*.*
A sequence arrow $\gamma : m_0 \to m_1$ *is a tuple* $\langle \mu_1, \ldots, \mu_k, \sigma \rangle$ *where* $k \geq 0$*, for each* $0 < i \leq k$*,* $\mu_i : m_{i-1} \to m_i$ *is an action arrow and* $\sigma : m_k \to m'$ *is a fusion arrow. In addition, we require that, if* $\sigma(a_i) = \sigma(a_j)$ *with* $i < j$*, then name* a_j *does not appear in actions* $\mu_1, \ldots, \mu_k$*. Notice that for* $k = 0$ *we obtain fusion arrows while for* $k = 1$ *and* $\sigma = \mathrm{id}_m$ *we obtain action arrows.*
A process arrow $p : \star \to m$ *is a tuple* $\langle P, \mu_1, \ldots, \mu_k, \sigma \rangle$ *where* $P : \star \to m_0$ *is a normalised agent arrow and* $\langle \mu_1, \ldots, \mu_k, \sigma \rangle$ *is a sequence arrow such that* $\mathrm{dom}(\mu_1) = m_0$*. Notice that for* $k = 0$*, and* $\sigma = id_{m_0}$ *we obtain normalised agent arrows.*

Definition 11 (Process-Action-Context category). *The* process-action-context category **PAC** *is the category having as objects elements of* $\omega_\star$ *and as morphisms:*

1. *the identity arrows* $id_\star : \star \to \star$ *and* $id_m : m \to m$*, the latter being the identity substitution on* $\{a_1, \ldots, a_m\}$*;*
2. *the normalised agent arrows, the fusion arrows and the action arrows as generators; and*
3. *the arrows freely generated by 2 under the composition operation* $\_;\_$ *subject to the usual associativity and identity axioms and, in addition, to the following axioms:*

$$\frac{\sigma : n \to m \quad a_i^m : m \to m+1}{\sigma; a_i^m = a_h^n; \sigma'},\ h = \min_l \{\sigma(a_l) = a_i\} \quad \sigma' = \sigma[n+1 \mapsto m+1]$$

$$\frac{\sigma : n \to m \quad \overline{a}_i^m a_j : m \to m}{\sigma; \overline{a}_i^m a_j = \overline{a}_h^n a_k; \sigma},\ h = \min_l \{\sigma(a_l) = a_i\} \quad k = \min_l \{\sigma(a_l) = a_j\}$$

($\sigma[n+1 \mapsto m+1]$ *stands for the function that behaves as* σ *for any* $a \in \{a_1, \ldots, a_n\}$ *and maps* a_{n+1} *to* a_{m+1}*).*

The arrows of **PAC** can be given an intuitive standard representation that will be useful later in the proofs.

Proposition 2. *The arrows of* **PAC** *are exactly the process arrows, the sequence arrows and the identity arrow* $id_\star$.

Proof. First, observe that: (a) a normalised agent arrow is a process arrow with an empty sequence of actions and an identity substitution. (b) A fusion arrow σ is a sequence arrow with no action arrows and with σ as the fusion arrow; this also yields the identities id_m where $m \in \omega$. (c) Similarly, action arrows are sequence arrows with a single action arrow and the identity substitution. Now, we prove that the composition of a process (resp. sequence) arrow with a sequence arrow yields a process (resp. sequence) arrow. Consider $p : \star \to m$ and $\gamma : m \to n$ be the process arrow $\langle P, \mu_1, \ldots, \mu_h, \sigma \rangle$ and the sequence arrow $\langle \mu'_1, \ldots, \mu'_k, \sigma' \rangle$. By definition $p;\gamma = \langle P, \mu_1, \ldots, \mu_h, \sigma, \mu'_1, \ldots, \mu'_k, \sigma' \rangle$, and, observing that the two last axioms in 3 of Definition 11, allows to "exchange" a fusion arrow with an action arrow, we trivially conclude that $p;\gamma = \langle P, \mu_1, \ldots, \mu_h, \mu''_1, \ldots, \mu''_k, \sigma''; \sigma' \rangle$, for suitable $\mu''_1, , \ldots, \mu''_k$ and σ''. We remark that if, at any stage, two names are fused, say a_i and a_j with $i < j$, then a_j is replaced by a_i by definition and this guarantees that $\langle P, \mu_1, \ldots, \mu_h, \mu''_1, \ldots, \mu''_k, \sigma''; \sigma' \rangle$ is a process arrow. The prove is the same when considering composition between two sequence arrows.

The proof is concluded by showing that different arrows cannot be equated by axioms. In other words, we prove that the standard representation of an arrow is unique (up to identities). Indeed, by inspecting the initial part of the proof we see that equality between two arrow can be proved only by shifting back and forth fusion arrows or introducing/cancelling identities. In the former case, any shift uniquely determines both the action and the fusion arrow of the equated arrows (Definition 11). □

As already mentioned, in the above definitions we have introduced typed versions (the type is a natural number m) of normalised agents and actions (substitutions are already typed), such that their names are in $\{a_1, \ldots, a_m\}$. This is apparently required by the "box and wires" structure of category **PAC**. We continue defining typed versions of ordinary processes and of fusions.

Given a π_- agent p and a natural number m such that $m \geq max\{k \mid a_k \in \mathrm{fn}(p)\}$, we denote as $p_m : \star \to m$ the arrow $\hat{p}_n;\sigma$ where $n = |\mathrm{fn}(\hat{p})| + m - |\mathrm{fn}(p)|$ and $\sigma : n \to m$ is defined as:

- $\sigma(a_i) = \sigma_p(a_i)$, if $i \in \mathrm{fn}(\hat{p})$,
- σ bijective and index monotone when restricted to $i \notin \mathrm{fn}(\hat{p})$ (where σ is *index monotone* if $\sigma(a_i) = \sigma(a_h)$, $\sigma(a_j) = \sigma(a_k)$ and $i \leq j$ implies $h \leq k$).

Basically, p_m represents the agent p in terms of a normalised process with n variables. Given a fusion $a_i = a_j$ and $m \in \omega$, with $i < j \leq m$, the substitution $[a_i = a_j]_m : m \to m-1$ is defined as follows:

$$[a_i = a_j]_m(a_k) = \begin{cases} a_k, & k < j \\ a_i, & k = j \\ a_{k-1}, & j < k \leq m \end{cases}$$

In words, $[a_i = a_j]_m$ maps the initial m names to the initial $m-1$ by replacing a_j with a_i and mapping the names greater that a_j to their predecessors.

$$\begin{tikzcd} & {} & \\ {} \arrow[ur, "\gamma'"] \arrow[r, "\hat{\gamma}_1"'] & {} \arrow[u, "\hat{\gamma}_3"'] & {} \arrow[ul, "\gamma"'] \arrow[l, "\hat{\gamma}_2"] \\ & \star \arrow[ul, "p"] \arrow[ur, "q"'] & \end{tikzcd} \qquad \begin{tikzcd} & {} & \\ {} \arrow[ur, "\gamma'"] \arrow[r, "\gamma'_1"'] & {} \arrow[u, "\gamma'_3"'] & {} \arrow[ul, "\gamma"'] \arrow[l, "\gamma'_2"] \\ & \star \arrow[ul, "p"] \arrow[ur, "q"'] & \end{tikzcd} \qquad \begin{tikzcd} & {} & \\ {} \arrow[ur, "\gamma'"] \arrow[r, "\hat{\gamma}_1"'] & {} \arrow[u, "\hat{\gamma}_3"'] & {} \arrow[ul, "\gamma"'] \arrow[l, "\hat{\gamma}_2"] \\ & {} \arrow[ul, "\gamma_1"] \arrow[ur, "\gamma_2"'] & \end{tikzcd}$$

(a) (b) (c)

Fig. 3. Diagrams for proofs in Theorem 4

Definition 12 (PAC reaction rules). *The* reaction rules *are those generated by the following inference rules where $m \geq |\mathrm{fn}(P)|$:*

$$\frac{P \xrightarrow{\overline{a}b} q}{P_m;\overline{a}^m b \implies q_m} \qquad \frac{P \xrightarrow{a(a_{m+1})} q}{P_m;a^m \implies q_{m+1}}$$

$$\frac{P \xrightarrow{a_i=a_j} q \quad i \neq j}{P_m;[a_i=a_j]_m \implies q_m;[a_i=a_j]_m} \qquad \frac{P \xrightarrow{\varepsilon} q}{P_m \implies q_m}$$

Definition 12 specify the reduction rules of **PAC** which rely on the LTS semantics of $\pi_{=}$. Take the first rule; it states that, if a normalised process P makes an output transition to q, then, in **PAC**, the corresponding arrow composed with the (output) action arrow (considered in at least $|\mathrm{fn}(P)|$ variables m) reduces to the arrow representing q in m variables. Basically, the same can be said for the input and fusion transitions, aside that the former introduces the new variable a_{m+1} while the latter eliminates a variable. The last rule is just the special case of fusing a name with itself (i.e., $P;id$ is the lhs of the reduction).

Theorem 4. **PAC** *has redex relative pushouts (RPOs).*

Proof. We must prove that, given a reaction rule $q \implies r$, for any process arrow p and any sequence arrows γ, γ' such that $p;\gamma' = q;\gamma$, there exist three sequence arrows $\hat{\gamma}_1$, $\hat{\gamma}_2$ and $\hat{\gamma}_3$ that satisfy the following conditions:

a. the diagram in Figure 3(a) commutes, and
b. for any sequence arrows γ'_1, γ'_2 and γ'_3 such that the diagram in Figure 3(b) commutes, there is a unique $\hat{\gamma}$ such that both
$$\begin{tikzcd} & {} & \\ {} \arrow[ur, "\gamma'_1"] \arrow[r, "\hat{\gamma}_1"'] & {} \arrow[u, "\hat{\gamma}"'] & {} \arrow[ul, "\gamma'_2"'] \arrow[l, "\hat{\gamma}_2"] \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} & {} \\ {} \arrow[ur, "\hat{\gamma}_3"] \arrow[r, "\hat{\gamma}"'] & {} \arrow[u, "\gamma'_3"'] \end{tikzcd}$$
commute.

Let us remark that the reduction contexts are all the arrows of **PAC**, however, for redex RPOs, γ and γ' can only be sequence arrows. Moreover, since $p;\gamma' = q;\gamma$, for Proposition 2, p and q are process arrows that are the composition of the *same* normalised linear arrow, say P with two sequence arrows. Hence, without loss of generality, it suffices to prove that there are arrows $\hat{\gamma}_1$, $\hat{\gamma}_2$ and $\hat{\gamma}_3$ forming an RPO for any diagram as in Figure 3(c).

The proof continues by case analysis.

- First assume that γ_2 is an identity fusion arrow and consider the commuting diagram below.

  $$\begin{tikzcd} & {} & \\ {} \arrow[ur, "\gamma'"] & & {} \arrow[ul, "\gamma"'] \\ & {} \arrow[ul, "\gamma_1"] \arrow[ur, "id"'] & \end{tikzcd}$$

  We prove that $\hat{\gamma}_1 = id$, $\hat{\gamma}_2 = \gamma_1$ and $\hat{\gamma}_3 = \gamma'$ is an RPO. Indeed, condition a) trivially holds because the external square commutes by hypothesis. Consider three sequence arrows γ'_1, γ'_2 and γ'_3 such that $\gamma' = \gamma'_1;\gamma'_3$, $\gamma = \gamma'_2;\gamma'_3$ and $\gamma_1;\gamma'_1 = \gamma'_2$. Then, assuming $\hat{\gamma} = \gamma'_3$ we obtain that the commutativity of the triangles corresponding to condition b) holds. Finally, uniqueness of $\hat{\gamma}$ is guaranteed by observing that $\hat{\gamma}_1$ is the identity.
- Let γ_2 is a generic fusion arrow σ. By Proposition 2, there is a sequence arrow γ'' such that $\sigma;\gamma = \gamma''$. Hence, we can equivalently prove that $$\begin{tikzcd} & {} & \\ {} \arrow[ur, "\gamma'"] & & {} \arrow[ul, "\gamma''"'] \\ & {} \arrow[ul, "\gamma_1"] \arrow[ur, "id"'] & \end{tikzcd}$$ has an RPO, which hold by the previous case.

- Finally, assume that γ_2 is an action arrow μ. By hypothesis, $\gamma_1;\gamma' = \mu;\gamma$, then, by Proposition 2,
  - either $\gamma_1 = \mu;\gamma'_1$
  - or γ_1 is the identity and $\gamma' = \mu;\gamma''$.

In the former case, the proof reduces to show that $$\begin{tikzcd} & {} & \\ {} \arrow[ur, "\gamma'"] & & {} \arrow[ul, "\gamma''"'] \\ & {} \arrow[ul, "\gamma'_1"] \arrow[ur, "id"'] & \end{tikzcd}$$ has an RPO, which hold by the previous case. While, in the latter case, the redex diagram is $$\begin{tikzcd} & {} & \\ {} \arrow[ur, "\gamma'"] & & {} \arrow[ul, "\gamma"'] \\ & {} \arrow[ul, "id"] \arrow[ur, "\mu"'] & \end{tikzcd}$$ and, proceeding as before, it is easy to see that μ, id and γ constitute an RPO.

□

Definition 13 (Labelled transitions). *The diagram in Figure 3(a) is an IPO when it is an RPO and $\hat{\gamma}_1 = \gamma'$, $\hat{\gamma}_2 = \gamma$ and $\hat{\gamma}_3 = id$. We write $p \xrightarrow{\gamma'} r;\gamma$ when there is a reduction rule $q \Longrightarrow r$ and the diagram Figure 3(a) is an IPO. This defines a LTS. The corresponding bisimilarity according to Definition 8 is denoted as $\bumpeq$.*

The results in [52] and Theorem 12 guarantee the following corollary.

Corollary 1. *Bisimilarity relation $\bumpeq$ is a congruence.*

The LTS of Definition 13 is essentially the same as in Section 5.1 indeed, the states are π_{-} processes and it is possible to show that the IPOs of **PAC** characterise the transitions of [73]. Thus bisimilarity relation $\bumpeq$ essentially coincides with syntactic bisimilarity $\backsimeq$.

Theorem 5 ($\bumpeq$ is $\backsimeq$). *Relation $\bumpeq$, which is defined on process arrows p_m, when restricted to those p_m with $m = max\{k \mid a_k \in \mathrm{fn}(p)\}$, coincides with $\backsimeq$.*

Notice that, due to the missing restriction operator, two agents with different sets of free names cannot be bisimilar. Thus, observing actions or typed actions does not make a difference.

From Theorem 2 we know that $\bumpeq$ is finer than or equal than $\sim$. It is easy to see that it is finer from this example. Consider the following processes

$$p = (\overline{a}b \mid a'(c)) + (\overline{d}e \mid d(f)) \qquad\qquad q = \overline{a}b.a'(c) + a'(c).\overline{a}b + (\overline{d}e \mid d(f)),$$

then $p \sim q$ because the synchronisation between $\overline{a}b$ and $a'(c)$ in a context that identifies a and a' is matched by the (unique) synchronisation of q. On the contrary, $p \not\bumpeq q$ because the transition $p \xrightarrow{a=a'}$ cannot be matched by q. We can thus conclude the following fact.

Theorem 6. *Relation $\bumpeq$ when restricted to those p_m with $m = max\{k \mid a_k \in \mathrm{fn}(p)\}$, is finer than $\sim$.*

6 Conclusions

In the paper we surveyed some of the approaches for model checking nominal calculi, focusing on HD-automata and on the existing toolkits for handling them. We also introduced a simplified version of open π-calculus and we proposed a bisimilarity semantics for it based on a reactive system with observed borrowed contexts. This approach has been proposed by Leifer and Milner [52, 51] and further developed by Sassone, Lack and Sobocinski [75, 76, 49] using G-categories and adhesive categories. The generality of the reactive system approach gives some hope that interesting abstractions of the SOC paradigm could also be modelled that way.

However we noticed that the transition system we obtain in this manner is not really suitable for a HD-automata implementation, since new names are never forgotten. To avoid this problem, it might be necessary of taking advantage of the extended theory developed by Sassone, Lack and Sobocinski [75, 76, 49]. In particular, the actions of nominal calculi which forget names could be represented as cospans of suitable adhesive categories. In fact several expressive graph-like structures can be represented by adhesive categories and the existing theory guarantees that the categories of their cospans have the *all redex-RPOs* property.

Acknowledgements

The authors thank Vladimiro Sassone and Pawel Sobocinski for their helpful comments on an earlier draft of this paper.

References

1. M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. and Comp.*, 148(1):1–70, January 1999.
2. G. Baldi, A. Bracciali, G. Ferrari, and E. Tuosto. A coordination-based methodology for security protocol verification. In *2nd International Workshop on Security Issues with Petri Nets and other Computational Models*, ENTCS, Bologna, Italy, June 2004. Elsevier. To appear.
3. T. Ball and S. Rajamani. The slam toolkit. In G. Berry, H. Comon, and A. Finkel, editors, *CAV*, volume 2102 of *LNCS*, pages 260–264. Springer, 2001.
4. N. Benton, L. Cardelli, and C. Fournet. Modern concurrency abstractions for C#. *TOPLAS*, 26(5):269–304, Sept. 2004.
5. M. Boreale and M. Buscemi. A framework for the analysis of security protocols. In L. Brim, P. Jančar, M. Křetinský, and A. Kučera, editors, *CONCUR*, volume 2421 of *LNCS*, pages 483–498. Springer, Aug. 2002.
6. M. Boreale and R. De Nicola. A symbolic semantics for the pi-calculus. *Inf. and Comp.*, 126(1):34–52, April 1996.
7. A. Bouali, S. Gnesi, and S. Larosa. The integration project for the jack environment. In *EATCS Bull.*, volume 54, pages 207–223. Centrum voor Wiskunde en Informatica (CWI), 1994.
8. A. Bouali, A. Ressouche, V. Roy, and R. de Simone. The FC2TOOLS set. In R. Alur and T. Henzinger, editors, *CAV*, volume 1102 of *LNCS*, pages 441–445, New Brunswick, NJ, USA, 1996. Springer.
9. A. Bracciali, A. Brogi, G. Ferrari, and E. Tuosto. Security issues in component based design. In U. Montanari and V. Sassone, editors, *ConCoord: International Workshop on Concurrency and Coordination*, volume 54 of *ENTCS*, Lipari Island - Italy, July 2001. Elsevier.
10. G. Brat, K. Havelund, S. Park, and W. Visser. Model checking programs. *Automated Software Engineering*, 10(2):203–232, 2003.
11. R. Bruni, C. Laneve, and U. Montanari. Orchestrating transactions in join calculus. In L. Brim, P. Jancar, M. Kretinsky, and A. Kucera, editors, *CONCUR*, volume 2421 of *LNCS*, pages 321–336. Springer, 2002.
12. R. Bruni, H. Melgratti, and U. Montanari. Theoretical foundations for compensations in flow composition languages. In *POPL*, 2005. To appear.
13. R. Bruni, U. Montanari, and F. Rossi. An interactive semantics of logic programming. *Theory and Practice of Logic Programming.*, 1(6):647–690, 2001.
14. M. Butler and C. Ferreira. An operational semantics for stac, a language for modelling long-running business transactions. In R. De Nicola, G. Ferrari, and G. Meredith, editors, *COORDINATION*, volume 2949 of *LNCS*, pages 87–104. Springer, 2004.
15. L. Caires and L. Cardelli. A spatial logic for concurrency (part I). *Inf. and Comp.*, 186, 2003.
16. L. Caires and L. Cardelli. A spatial logic for concurrency II. *TCS*, 322(3):517–565, Sept. 2004.
17. G. Cattani and P. Sewell. Models for name-passing processes: Interleaving and causal (extended abstract). *Inf. and Comp.*, 190(2):136–178, May 2004.
18. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
19. E. Clarke and J. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, December 1996.
20. S. Clarke, Edmund M. Jha and W. Marrero. Using state space exploration and a nautural deduction style message derivation engine to verify security protocols. In *In Proc.* IFIP Working Conference on Programming Concepts and Methods (PROCOMET), 1998.

21. R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems. *TOPLAS*, 15(1):36–72, Jan. 1993.
22. S. Conchon and F. Le Fessant. Jocaml: Mobile agents for objective-cam. In *International Symposium on Agent Systems and Applications*, pages 22–29, Palm Springs, California, Oct. 1999.
23. J. Corbett, M. Dwyer, J. Hatcliff, S. Laubach, S. Corina, Robby, and H. Zheng. Bandera: Extracting finite-state models from java source code. In *International Conference on Software Engineering*, pages 439–448, Limerick, Ireland, June 2000.
24. F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The next step in web services. *CACM*, 46(10):29–34, 2003.
25. M. Dam. Model checking mobile processes. *Inf. and Comp.*, 129(1):35–51, 1996.
26. M. Dam. Proof systems for π-calculus logics. *Logic for concurrency and synchronisation*, pages 145–212, 2003.
27. G. Denker and J. Millen. Capsl integrated protocol environment. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, 1999.
28. H. Ehrig and B. König. Deriving bisimulation congruences in the DPO approach to graph rewriting. In I. Walukiewicz, editor, *FoSSaCS*, volume 2987, pages 151–166. LNCS, 2004.
29. J. Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science of Computer Programming*, 13(2–3):219–236, May 1990.
30. J. Fernandez and L. Mounier. ”On-the-fly” verification of behavioural equivalences and preorders. In K. Larsen and A. Skou, editors, *CAV*, volume 575 of *LNCS*, pages 181–191. Springer, July 1991.
31. G. Ferrari, G. Ferro, S. Gnesi, U. Montanari, M. Pistore, and G. Ristori. An automata based verification environment for mobile processes. In E. Brinksma, editor, *TACAS*, volume 1217 of *LNCS*, pages 275–289. Springer, April 1997.
32. G. Ferrari, S. Gnesi, U. Montanari, and M. Pistore. A model checking verification environment for mobile processes. *TOPLAS*, 12(4):1–34, 2004.
33. G. Ferrari, S. Gnesi, U. Montanari, R. Raggi, G. Trentanni, and E. Tuosto. Verification on the web. In J. Augusto and U. Ultes-Nitsche, editors, *VVEIS*, pages 72–74, Porto, Portugal, April 2004. INSTICC Press.
34. G. Ferrari, U. Montanari, and M. Pistore. Minimizing transition systems for name passing calculi: A co-algebraic formulation. In M. Nielsen and U. Engberg, editors, *FOSSACS 2002*, volume 2303 of *LNCS*, pages 129–143. Springer, 2002.
35. G. Ferrari, U. Montanari, and E. Tuosto. From co-algebraic specifications to implementation: The Mihda toolkit. In F. de Boer, M. Bonsangue, S. Graf, and W. de Roever, editors, *FMCO*, volume 2852 of *LNCS*, pages 319 – 338. Springer, November 2002.
36. G. Ferrari, U. Montanari, and E. Tuosto. Coalgebraic minimisation of HD-automata for the π-calculus in a polymorphic λ-calculus. *TCS*, 2004. To appear.
37. M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding (extended abstract). In *LICS*, pages 193–202, Trento, Italy, July 1999. IEEE.
38. P. Fiore and M. Abadi. Computing symbolic models for verifying cryptographic protocols. In *Computer Security Foundations Workshop*, CSFW, pages 160–173, Cape Breton, Nova Scotia, Canada, June 2001. IEEE.
39. R. Focardi and R. Gorrieri. A classification of security properties. *J. of Computer Security*, 3(1), 1995.
40. M. Gabbay and A. Pitts. A new approach to abstract syntax involving binders. In G. Longo, editor, *LICS*, pages 214–224, Trento, Italy, July 1999. IEEE.
41. A. Gordon. Notes on nominal calculi for security and mobility. In R. Focardi and R. Gorrieri, editors, *FOSAD*, volume 2171 of *LNCS*, pages 262–330. Springer, September 2002.
42. M. Hennessy and H. Lin. Symbolic bisimulations. *TCS*, 138(2):353–389, February 1995.

43. T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *POPL*, pages 58–70. ACM Press, 2002.
44. D. Hirschkoff. On the benefits of using the up-to techniques for bisimulation verification. In W. Cleaveland, editor, *TACAS*, volume 1579 of *LNCS*, pages 285–299, Amsterdam, March 1999. Springer.
45. G. Holzmann. The Model Checker Spin. *TSE*, 23(5):279–295, May 1997.
46. G. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, Sept. 2003.
47. K. Honda. Elementary structures in process theory (1): Sets with renaming. *MSCS*, 10(5):617–663, 2000.
48. P. Kanellakis and S. Smolka. CCS expressions, finite state processes and three problem of equivalence. *Inf. and Comp.*, 86(1):272–302, 1990.
49. S. Lack and P. Sobocinski. Adhesive categories. In I. Walukiewicz, editor, *FoSSaCS*, volume 2987 of *LNCS*, pages 273–288, Barcelona, March 2004. Springer.
50. C. Laneve and G. Zavattaro. Foundations of web transactions. In *FoSSaCS*, LNCS, 2005. To appear.
51. J. Leifer. *Operational Congruences for Reactive Systems*. PhD thesis, Computer Laboratory, University of Cambridge, Cambridge, UK, 2001.
52. J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In C. Palamidessi, editor, *CONCUR*, volume 1877 of *LNCS*, pages 243–258, University Park, PA, USA, August 22-25 2000. Springer.
53. H. Lin. Complete inference systems for weak bisimulation equivalences in the pi-calculus. *Inf. and Comp.*, 180(1):1–29, January 2003.
54. G. Lowe. Towards a completeness result for model checking of security protocols. In *CSFW*. IEEE, 1998.
55. W. Marrero, E. Clarke, and S. Jha. Model checking for security protocols. In *Formal Verification of Security Protocols*, 1997.
56. K. McMillan. *Symbolic Model Checking*. Kluwer Academic Publisher, 1993.
57. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
58. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Inf. and Comp.*, 100(1):1–40,41–77, September 1992.
59. J. Mitchell, M. Mitchell, and U. Ster. Automated analysis of cryptographic protocols using murϕ. In *CSFW*, pages 141–151. IEEE, 1997.
60. U. Montanari and M. Buscemi. A first order coalgebraic model of π-calculus early observational equivalence. In L. Brim, P. Jančar, M. Křetinský, and A. Kučera, editors, *CONCUR*, volume 2421 of *LNCS*, pages 449–465. Springer, Aug. 2002.
61. U. Montanari and M. Pistore. Checking bisimilarity for finitary π-calculus. In I. Lee and S. Smolka, editors, *CONCUR*, volume 962 of *LNCS*, pages 42–56, Philadelphia, PA, USA, Aug. 1995. Springer.
62. U. Montanari and M. Pistore. History dependent automata. Technical report, Computer Science Department, Università di Pisa, 1998. TR-11-98.
63. U. Montanari and M. Pistore. π-calculus, structured coalgebras, and minimal HD-automata. In M. Nielsen and B. Roman, editors, *MFCS*, volume 1983 of *LNCS*. Springer, 2000. An extended version will be published on Theoretical Computer Science.
64. U. Montanari and V. Sassone. Dynamic congruence vs. progressing bisimulation for CCS. *Fundamenta Informaticae*, 16:171–196, 1992.
65. R. Needham. *Names*. Addison-Wesley (Mullender Ed.), 1989.
66. R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, December 1987.
67. M. Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Web Information Systems Engineering (WISE'03)*, LNCS, pages 3–12. Springer, 2003.

68. D. Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science, 5th GI-Conf.*, volume 104 of *LNCS*, pages 167–183. Springer, Karlsruhe, March 1981.
69. J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *LICS*. IEEE, 1998.
70. M. Pistore. *History dependent automata*. PhD thesis, Computer Science Department, Università di Pisa, 1999.
71. M. Pistore and D. Sangiorgi. A partition refinement algorithm for the π-calculus. *Inf. and Comp.*, 164(2):467–509, 2001.
72. D. Sangiorgi. On the bisimulation proof method (extended abstract). In J. Wiedermann and P. Hájek, editors, *MFCS*, volume 969 of *LNCS*, pages 479–488, Prague, August-September 1995. Springer.
73. D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33(1):69–97, 1996.
74. D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2002.
75. V. Sassone and P. Sobocinski. Deriving bisimulation congruences using 2-categories. *Nordic J. of Computing*, 10(2), 2003.
76. V. Sassone and P. Sobocinski. Locating reaction with 2-categories. *TCS*, 2004. To appear.
77. P. Sewell. From rewrite rules to bisimulation congruences. *LNCS*, 1466, 1998.
78. P. Sewell. Applied π – a brief tutorial. Technical Report 498, Computer Laboratory, University of Cambridge, Aug. 2000.
79. H. Smith and P. Fingar. Workfow is just a pi process. Available at http://www.bpm3.com/picalculus, 2003.
80. V. Vanackére. The trust protocol analyser. automatic and efficient verification of cryptographic protocols. In *VERIFY02*, 2002.
81. B. Victor and F. Moller. The Mobility Workbench — A Tool for the π-Calculus. In D. Dill, editor, *Proceedings of CAV '94*, volume 818 of *LNCS*, pages 428–440. Springer, 1994.
82. P. Yang, C. Ramakrishnan, and S. Smolka. A logical encoding of the pi-calculus: Model checking mobile processes using tabled resolution. *STTT*, 6(1):38–66, July 2004.