Activeness and Responsiveness in Mobile Processes

Maxime Gamboni[*] [‡], António Ravara[‡]

[‡]SQIG, Instituto de Telecomunicações and Mathematics Department, IST, Technical University of Lisbon
Tel. +351-218417148 Fax. +351-218499242
E-mail: maxime@gamboni.org, amar@math.ist.utl.pt

Abstract — **In mobile processes, a channel can be *active* (able of sending/receiving) or *responsive* (provide a reply to, or parameters for, a request), or both. It has a *multiplicity* such as ω (usable arbitrarily many times) or plain (unrestricted behaviour).**

The context the process will be running in is similarly described, specifying to what extent third-party processes may interfere.

We outline semantic definitions of the properties, and a sound, compositional, type system enjoying subject-reduction, which verifies that, in a given process, channels behave as specified, and indicates, when applicable, what should be provided to the process before the properties hold.

I. Introduction

When reasoning about process behaviour one often needs to know (or enforce) certain properties of (to) given names, but not to all of them, and moreover, different names may require different properties. Furthermore, imposing properties on names instead of on processes allows more liberal process behaviour (e.g., a process may have deterministic names without being deterministic itself). This approach eases the verification of liveness properties, and is, for instance, often crucial to ensure full abstraction results when defining calculi encodings, like for checking the correctness of an encoding of non-deterministic processes handling deterministic values. The setting of our work is the (polyadic) π-calculus [MPW92, Mil93]. Specifically, channel properties described in the current work are *activeness* (ability to send or receive) and *responsiveness* (ability to provide a reply to, or parameters for, a request).

In addition to non-homogeneity (having processes interact both at highly constrained and at unconstrained channels), a process may interact on channels whose remote side are assumed to be provided by the environment (through process composition), which means that we need a description of the environment behaviour before any guarantees can be given about the process. Such a description is included in *process types* and *channel types*. Each channel may also have *multiplicity constraints* on both input and output polars. For instance, channels to be used as uniform servers are ω (used uniformly and an arbitrary number of times) on input and plain (not having any constraint) on output. Finally, to each channel is associated a contract specifying both what to expect of the remote side and what should be provided locally. Given a process P, a set of channel types Σ and optional multiplicity information, the type system either fails (in case the process doesn't match the requirements), or builds a *process type* Γ containing Σ and a *dependency network* Ξ, that indicates which channels are active (resp., responsive), and, when applicable, what is additionally required from the environment. In all cases, that dependency network satisfies the semantic definitions of activeness and responsiveness.

Our main contributions are: (1) a channel-based approach, providing a non-homogeneous combination of properties of channels; (2) the possibility of describing the intended behaviour of a process context to guarantee a (conditional) property; (3) a notion of *activeness*, which provides stronger guarantees than ((non-)uniform) receptiveness of Amadio et al. and Acciai and Boreale's responsiveness, as well as a notion of *conditional* activeness which generalises them. (4) a sound type checking system that is compositional (in the sense that applying to a process a constructor does not imply to retype the whole process). We illustrate the main technical notions with running examples that we incrementally analyse. These examples also highlight the expressiveness of our approach since they are not fully or correctly analysed by related approaches. The interested reader may find the precise definitions, results and proofs in a technical report [GR09].

II. Calculus

A. Processes

Our target calculus is the choice-free synchronous polyadic π-calculus, where processes P have inputs $a(\tilde{y}).P$, outputs $\overline{a}\langle\tilde{x}\rangle.P$, parallel composition $P_1|P_2$, binding $(\boldsymbol{\nu}x)\,P$, replication $!\,P$ and the idle process $\mathbf{0}$. We use an early labelled transition system adapted from [SW01],

[*]This work is partially supported by SQIG - IT and IST, Portugal, by Fundação para a Ciência e a Tecnologia (via CLC and the project Space-Time-Types, POSC/EIA/55582/2004), as well as Sensoria (IST-2005-16004).

and labels, ranged over by μ, are inputs $a(\tilde{x})$, outputs $(\boldsymbol{\nu}\tilde{z})\,\overline{a}\langle\tilde{x}\rangle$ and reductions τ. Consider the following running example:

$$P_1 = a.\overline{b}\langle tu\rangle \,|\, t.\overline{u}.\overline{s} \tag{1}$$

We show in the next sections that s is (output) *active* provided that, in the process environment, either a is output active and b input active and responsive, or t is output active and u input active.

B. Types

Channel Types describe the behaviour of a process at a channel, and process types describe the behaviour of a process globally. To a name a is associated a channel type recording the parameter types and *multiplicities*, as well as a *protocol* describing how a's input and output may negotiate receivers (and senders) at the parameters. A *resource* (ranged by α, β, γ) is a property k at a port p, and is written p_k, k being one of $\mathbf{A}$ (*activeness* — a receiver or sender is available at the port) and $\mathbf{R}$ (*responsiveness* — inputs respond to the queries they receive, and outputs provide receivers at their parameters). In (1), $\overline{s}_{\mathbf{A}}$ becomes available only after $\overline{t}_{\mathbf{A}}$ and $u_{\mathbf{A}}$ are provided. This information is written $(\overline{t}_{\mathbf{A}}u_{\mathbf{A}}) < \overline{s}_{\mathbf{A}}$. A channel type (ranged over by σ) is written $(\tilde{\sigma};\xi_I;\xi_O)$, where $\tilde{\sigma}$ is the channel types of the parameters and ξ_I and ξ_O form the protocol, describing what is provided by the input and the output side of the channel, respectively. For example (1), a, t, u and s have the *parameterless type* $\sigma_0 = (\emptyset;\emptyset;\emptyset)$, and b has the type $\sigma_b = \Big(\sigma_0\sigma_0; \overline{1}^{\bigstar}_{\mathbf{A}}, 2^{\bigstar}_{\mathbf{A}}; 1, \overline{2}\Big)$. As the parameters on b's input side are declared plain and active, any b input must provide *at least one* (t-output and u-input).

A *process type* is very similar to a channel type, except that it refers to channels by their name rather than by numbers. Indeed, the type of a process P is similar to what would be the type of channel z in $z(\tilde{x}).P$, where $\tilde{x} = \mathrm{fn}(P)$ and $z \notin \tilde{x}$. The process corresponds to the input side of the z channel, and the output side of that channels abstracts the environment of the process. The process type for example (1) is Γ_1, equal to

$$\begin{gathered}((a,t,u,s){:}\sigma_0, b{:}\sigma_b\ ;\ a_{\mathbf{A}}, \overline{a}_{\mathbf{A}}{<}\overline{b}_{\mathbf{A}}, \overline{a}_{\mathbf{A}}b_{\mathbf{AR}}{<}\overline{t}_{\mathbf{A}},\\ \overline{a}_{\mathbf{A}}b_{\mathbf{AR}}{<}u_{\mathbf{A}}, (\overline{t}_{\mathbf{A}}u_{\mathbf{A}})|(\overline{a}_{\mathbf{A}}b_{\mathbf{AR}}){<}\overline{s}_{\mathbf{A}}\ ;\ (a\overline{b}t\overline{u}\overline{s})^0 b^{\omega}\overline{a}s)\end{gathered} \tag{2}$$

where the dependency statement for $\overline{s}_{\mathbf{A}}$ formalises the description of (1) given in the introduction.

A *product* operator for process types, describes the effect of composing two processes. For instance, composing (2) with a type for

$$\overline{a}.!\,b(xy).\overline{x}.y \tag{3}$$

yields a process type where $\overline{s}_{\mathbf{A}}$'s dependency is 0.

C. Typed Transition System

We lift the transition system on *typed processes*, to rule out transitions that force the process to behave badly, and obtain good safety properties for labelled transitions, as well as predict the effect of a transition on a process. The predicted effect of a μ-transition on a process with type Γ is given by $\Gamma \setminus \mu$. A *typed process* being a pair formed by a process type and a process, we write $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$ if $P \xrightarrow{\mu} P'$ and $\Gamma' = \Gamma \setminus \mu$ is well-defined. Taking example (1) but (as will be justified later) with $\overline{a}_{\mathbf{A}}$ and $b_{\mathbf{AR}}$ added in the remote side of Γ_1 (2). $(\Gamma_1; P_1) \xrightarrow{a} (\Gamma_1'; \overline{b}\langle tu\rangle \,|\, t.\overline{u}.\overline{s})$, where Γ_1' is like Γ_1 but with $\neg a^0_{\mathbf{A}}$ and $\neg\overline{a}^0_{\mathbf{A}}$ replacing $a_{\mathbf{A}}$ and $\overline{a}^1_{\mathbf{A}}$ in the local and remote components, respectively. $(\Gamma_1'; \overline{b}\langle tu\rangle \,|\, t.\overline{u}.\overline{s}) \xrightarrow{\overline{b}\langle tu\rangle} (\Gamma_1''; t.\overline{u}.\overline{s})$, where $\overline{b}$'s local multiplicities change from 1 to 0 and $\overline{b}_{\mathbf{A}}$ drops. Then, $\overline{t}_{\mathbf{A}}, u_{\mathbf{A}}$ moves from the local to the remote side: it's no longer a *local* assumption of remote behaviour, but an actual *remote* behaviour, and we obtain $\Gamma_1'' = \Big(\Sigma; (\overline{t}_{\mathbf{A}}u_{\mathbf{A}})|(\overline{a}_{\mathbf{A}}b_{\mathbf{AR}}){<}\overline{s}_{\mathbf{A}}, \neg(\overline{t}_{\mathbf{A}}u_{\mathbf{A}}); (a\overline{a}bt\overline{u}\overline{s})^0 b^{\omega}\overline{t}^{\bigstar}_{\mathbf{A}}u^{\bigstar}_{\mathbf{A}}s\Big)$.

III. Semantic Properties

A. Activeness

An *activeness strategy* for a port p is a function mapping processes to transition label-process pairs (if $f(P) = (\mu; P')$ then $(\Gamma; P) \xrightarrow{\mu} (\Gamma'; P')$) that "shows the way" to a process where it is immediately available. We say that p is *active* in P if there is a strategy f such that, in any transition sequence that follows f an infinite number of times, p eventually becomes immediately available. To prove availability of a resource γ *conditional* on some ε we modify the process type to declare that ε is available in the remote side, then prove that γ is available without conditions in the resulting typed process. For example, to prove the $\overline{a}_{\mathbf{A}}\&b_{\mathbf{AR}} < \overline{s}_{\mathbf{A}}$ statement in (2), let $\Gamma_{\overline{s}}$ be equal to Γ_1 but with $0\overline{a}_{\mathbf{A}}$ and $0b_{\mathbf{AR}}$ added to Ξ_R. Then f defined as follows is a strategy for $\overline{s}$. $\forall Q \in \{t.\overline{u}.\overline{s}\,,\ \overline{u}.\overline{s}\,,\ \overline{s}\}$: $f(a.\overline{b}\langle tu\rangle|Q) = (a; \overline{b}\langle tu\rangle|Q)$ and $f(\overline{b}\langle tu\rangle|Q) = (\overline{b}\langle tu\rangle; Q)$. Then, $f(t.\overline{u}.\overline{s}) = (t; \overline{u}.\overline{s})$, $f(\overline{u}.\overline{s}) = (\overline{u}; \overline{s})$ and $f(P') = (\emptyset; P')$ in all other cases. The strategy is to first send the request on $\overline{b}$ and then consume $\overline{s}$'s prefixes. It might be tempting to try consuming $\overline{s}$'s prefixes directly but $\overline{t}_{\mathbf{A}}$ and $u_{\mathbf{A}}$ are only made available in Ξ_R by the $\overline{b}\langle tu\rangle$ transition, as seen in the previous section.

B. Responsiveness

To show responsiveness of a port p we replace the entire dependency network of the process type by just $0p_{\mathbf{R}}$, and require the type to be valid for any transition sequence starting from that type. The effect of the transitions $\tilde{\mu}$ will extend that statement to include all the expected behaviour of P at p. Well-chosen transition sequences $\tilde{\mu}'$ permit "exploring" any part of the channel type. For example, (taking $a : \sigma_a$), b is responsive in (3) — the effect of the transition sequence $\xrightarrow{\bar{a}}\xrightarrow{b(tu)}$ expands (using σ_a's input side) $0b_{\mathbf{R}}$ into $\{0\overline{t}_{\mathbf{A}}, t_{\mathbf{A}} < \bar{u}_{\mathbf{A}}\}$, and both statements can be verified on the resulting process $\overline{t}.u$.

IV. Type System

A. Labelled Dependencies

Responsiveness must be computed *assuming the remote side is available and respects the channel type*. So in (1), when computing $\overline{b}_{\mathbf{R}}$'s dependencies, $\overline{t}_{\mathbf{A}}$ is assumed to be available, although, when considered on its own, $\overline{t}_{\mathbf{A}}$ depends on both $b_{\mathbf{A}}$ and $b_{\mathbf{R}}$. We use the *label annotations* $l:$ and $\neg l:$ to describe this. A *labelled dependency* $l : \varepsilon$ is equivalent to ε on its own except that, inside ε, dependencies marked with a *conditional dependency* $\neg l : \varepsilon'$ are irrelevant and treated like 0. Reciprocally, that conditional dependency is equivalent to ε' unless found in a region marked with l, in which case it is equivalent to 0.

Example (1) may now be described with $l : (t_{\mathbf{A}}\bar{u}_{\mathbf{A}}) \leq \overline{b}_{\mathbf{R}}$, $\neg l : (\bar{a}_{\mathbf{A}}b_{\mathbf{AR}}) < \overline{t}_{\mathbf{A}}$, $(\overline{t}_{\mathbf{A}}\&\neg l : (\bar{a}_{\mathbf{A}}b_{\mathbf{AR}})) < u_{\mathbf{A}}$, $0t_{\mathbf{A}}$ and $\overline{t}_{\mathbf{A}} < \bar{u}_{\mathbf{A}}$. Substituting $t_{\mathbf{A}}$ by 0 and $\bar{u}_{\mathbf{A}}$ by $\overline{t}_{\mathbf{A}}$ in the output responsiveness statement gives $l : (\overline{t}_{\mathbf{A}}) \leq \overline{b}_{\mathbf{R}}$ as before. Substituting $\overline{t}_{\mathbf{A}}$ by $\neg l : b_{\mathbf{AR}}$ yields $l : (\neg l : b_{\mathbf{AR}}) < \overline{b}_{\mathbf{R}}$ which is equivalent (by simplifying the labels) to $0\overline{b}_{\mathbf{R}}$, i.e. b is output responsive in the process.

B. Typing

The typing derivation for (1) is as follows: Process $\overline{b}\langle tu\rangle$ has type $\big(b : \sigma_b;;b^1\overline{b}^1\big) \odot \overline{b}_{\mathbf{A}} \odot l : 0\overline{b}_{\mathbf{R}} \odot \neg l : b_{\mathbf{AR}} < (\overline{t}^{\bigstar}_{\mathbf{A}}, u^{\bigstar}_{\mathbf{A}}) \odot \neg l : b_{\mathbf{A}}.\emptyset$, where the terms respectively stand for b's type and multiplicities, b's output activeness, output responsiveness (vacuously satisfied as σ_b doesn't have properties on the output side), the expected answer (obtained from σ_b and depending on remote activeness and responsiveness), and the (empty) continuation. Composing the factors yields $\Big(b : \sigma_b; \overline{b}_{\mathbf{A}}, l : 0\overline{b}_{\mathbf{R}}, \neg l : b_{\mathbf{AR}} < (\overline{t}^{\bigstar}_{\mathbf{A}}, u^{\bigstar}_{\mathbf{A}}); t^1\overline{t}^0u^0\bar{u}^1b^1\overline{b}^0\Big)$. For $a.\overline{b}\langle tu\rangle$, the type system just adds a statement $0a_{\mathbf{A}}$, and adds a $\bar{a}_{\mathbf{A}} <$ dependency to $\overline{b}_{\mathbf{A}}$, $\overline{t}_{\mathbf{A}}$ and $u_{\mathbf{A}}$. We similarly obtain $\big(tus : \sigma_0; t_{\mathbf{A}}, \overline{t}_{\mathbf{A}} < \bar{u}_{\mathbf{A}}, \overline{t}_{\mathbf{A}}u_{\mathbf{A}} < \bar{s}_{\mathbf{A}}; s^0\bar{s}^1\big)$ for $t.\bar{u}.\bar{s}$. The multiplicities for t and u are automatically obtained thanks to $\overline{b}\langle tu\rangle$. Finally, composing the two statements we get Γ_1 as given in (2). The dependencies for $\bar{s}_{\mathbf{A}}$ are obtained by substituting $\overline{t}_{\mathbf{A}} <$ with $(\overline{t}_{\mathbf{A}} <)|(a_{\mathbf{A}}\&\neg l : b_{\mathbf{AR}} <)$ and similarly for $u_{\mathbf{A}}$.

C. Properties

In addition to type soundness, the type system enjoys weak subject reduction for labelled transitions. Given $(\Gamma; P) \xrightarrow{\mu} (\Gamma_2; P')$, if $(\Gamma; P)$ is accepted by the type system, there is a strengthening Γ' of Γ_2 (with possibly less dependencies) that is accepted by the type system for P'.

V. Conclusions and Related Work

In this paper we outlined a notation for channel and process types in the π-calculus, semantics for activeness and responsiveness, and a (sound) type system analysing such process behaviour. Dependency networks in process types accurately specify the interface of the process with the environment, so that having typed P and Q independently, their types can be composed to obtain $P|Q$'s type without needing to type check from scratch. Labelled dependencies permit improving the accuracy of a type checking analysis by breaking dependency transitivity where it is not relevant. One strong point of our work is a high expressiveness — not only the types of all papers we surveyed (except *channel usages* [Kob02]) can be encoded in our notation, it allows a more detailed specification of the protocol used on a channel and capabilities being transmitted. Another strong point is detailed process types constructed by the type system — rather than merely rejecting or accepting a process, the type system will say separately for each resource if it is available or not, or if it requires additional resources to be provided by a third-party process.

In brief, we compare our work to other closely related.

Sangiorgi's uniform receptiveness [San99]. A name x is uniformly receptive for a process P if: (1) at any time P is ready to accept an input at x, at least as long as processes could send messages at x; (2) the input offer at x is functional. A type discipline checks linear and persistent names. Receptive names corresponds, in our terminology, to input active names. The type system is typing what we call strong activeness (only identity strategies are considered), and typability in our type system strictly implies in his.

Amadio et al.'s type discipline of non-uniform receptiveness entails message deliverability (every emitted message has the possibility of being received) [ABL02]. Type discipline enforces every resource to be persistent. All names are (in our terminology) plain output and non-uniform strong active ω input. Their work is mainly interesting in the distributed setting [ABL03] — restricting it to a local setting would reduce to the essentially syntactic check that all outputs have at least one corresponding unguarded input. However, *non-uniform* ω-activeness can't be presently characterised in our type system.

Acciai and Boreale's responsiveness [AB08a]. An agent uses a channel name r responsively if a communication along r is guaranteed eventually to take place. Responsiveness is achieved by combining techniques for deadlock and livelock avoidance with linearity and receptiveness. Their setting is simpler and less expressive than ours, in that it works on monadic synchronous π, I/O alternating and doesn't consider combinations of active and non-active names. On the other hand, their type system is more powerful, in that it handles unbounded recursion.

Acciai and Boreale's spatial and behavioural type system [AB08b] combines ideas from spatial logics and from [IK01]. Their systems relies on spatial model checking, but properties — both safety and liveness ones — are checked against types rather than against processes. Implementation and complexity issues, as well as the degree of completeness of the approach, are not treated, but naturally one expects a type checking system to be simple to implement and be of low complexity.

Kobayashi's TyPiCal [Kob08] is an implementation of a lock-freedom type system. Although it also performs termination and information flow analysis we are particularly interested in its lock-freedom analysis. This property is related to activeness: p is livelock-free if and only if the complement port $\bar{p}$ is active. Channel usages are more general than our multiplicities, and tell for a particular channel how many times the input and output ports are used, and in what order. There is no subsumption relation either way between our system and the one implemented by TyPiCal. On the one hand, the usage information is strictly more expressive than multiplicities. This permits for instance TyPiCal to handle locks correctly, unlike our system that would dismiss locks as plain and unreliable names. On the other hand, labelled dependencies permit an accurate analysis of processes such as (1). TyPiCal incorrectly marks the b input as unreliable (not livelock-free). Finally, TyPiCal does not handle recursive types that would be required to analyse processes like $\overline{a}\langle a\rangle$ or $!\, a(x).\overline{x}\langle a\rangle$ but we believe it would be a rather simple extension, as was the case for our system.

Igarashi and Kobayashi's Generic Type System [IK01] is a framework for type-checking various safety properties such as deadlock-freedom or race-freedom. It works with *abstract processes* — a simplified form of the target process — and soundness theorems establishing that if the abstraction is well-behaved then so is the actual process. It is particularly useful for *safety* properties as subject reduction is proven once and for all. In contrast, our type system is focused on *liveness* properties like activeness or termination so that showing the validity of a dependency analysis done on the abstract process and the correspondence between activeness in the abstract process and the actual one would likely require the same amount of work as starting from scratch.

REFERENCES

[AB08a] L. Acciai and M. Boreale. Responsiveness in process calculi. *Theoretical Computer Science*, 409(1):59–93, 2008.

[AB08b] L. Acciai and M. Boreale. Spatial and Behavioral Types in the Pi-Calculus. In *Proceedings of the 19th international conference on Concurrency Theory*, volume LNCS 5201, pages 372–386. Springer, 2008.

[ABL02] R. M. Amadio, G. Boudol and C. Lhoussaine. On message deliverability and non-uniform receptivity. *Fundamentæ Informatica*, 53(2):105–129, 2002.

[ABL03] R. M. Amadio, G. Boudol and C. Lhoussaine. The receptive distributed π-calculus. *ACM Transactions on Programming Languages and Systems*, 25(5):549–577, 2003.

[GR09] M. Gamboni and A. Ravara. Non-Homogeneous Channel Behaviour in Mobile Processes. Technical Report, SQIG — IT and IST, Technical University of Lisbon, Portugal, 2009. http://gamboni.org/files/incremental.pdf.

[IK01] A. Igarashi and N. Kobayashi. A generic type system for the Pi-calculus. *ACM SIGPLAN Notices*, 36(3):128–141, 2001.

[Kob02] N. Kobayashi. Type systems for concurrent programs. In *Proceedings of UNU/IIST 10th Anniversary Colloquium*, volume 2757 of *LNCS*, pages 439–453. Springer, 2002.

[Kob08] N. Kobayashi. TyPiCal 1.6.2, 2008. http://www.kb.ecei.tohoku.ac.jp/ koba/typical/.

[Mil93] R. Milner. The Polyadic π-Calculus: A Tutorial. In *Logic and Algebra of Specification, Proceedings of the International NATO Summer School (Marktoberdorf, Germany, 1991)*, volume 94 of *NATO ASI Series F*. Springer, 1993.

[MPW92] R. Milner, J. Parrow and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–77, 1992.

[San99] D. Sangiorgi. The Name Discipline of Uniform Receptiveness. *Theoretical Computer Science*, 221(1–2):457–493, 1999.

[SW01] D. Sangiorgi and D. Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.