Modal μ-Types for Processes

Marino Miculan
Dipartimento di Informatica – Università di Udine
Via delle Scienze, 206 – I-33100 Udine – Italy
E-mail: miculan@dimi.uniud.it

Fabio Gadducci*
Dipartimento di Informatica – Università di Pisa
Corso Italia, 40 – I-56100 Pisa – Italy
E-mail: gadducci@di.unipi.it

Abstract

We introduce a new paradigm for concurrency, called behaviours-as-types*. In this paradigm, types are used to convey information about the* behaviour *of processes: while terms corresponds to processes, types correspond to behaviours.*

We apply this paradigm to Winskel's Process Algebra. Its types are similar to Kozen's modal μ-calculus; hence, they are called modal μ-types*. We prove that two terms having the same type denote two processes which behave in the same way, that is, they are* bisimilar*. We give a sound and complete compositional typing system for this language. Such a system naturally recovers the notion of bisimulation also on open terms, allowing us to deal with processes with undefined parts in a compositional manner.*

1 Introduction

In Computer Science, the notion of *type* is one of the most important and fertile inheritances of Logics. Since it has been introduced by Church for the simple typed λ-calculus [9], the idea of typing has been applied to a variety of languages and formalisms. Types are usually syntactic objects which can be *assigned* to terms of the considered language, according to a given set of rules (the *typing system*). If t is a term and a type A is assigned to t, then we say that "t has type A", or "t inhabits A", we denote this by "$t : A$".

Types are intended to convey some information about the terms. The kind of information varies a lot from a system to another. One of the most common is about the way terms *interface* each other. For instance, in the simply typed λ-calculus, t_1 of type A can be applied to t_2 of type B only if A is "functional", that is, of the form $B \to C$.

This interpretation of types has yielded the well-known Curry-Howard isomorphism (the so-called *propositions-as-types* paradigm). It states that formulæ of the Intuitionistic Propositional Logic and their Natural Deduction proofs are isomorphic to types and terms of the simply typed λ-calculus, respectively. Proving the formula A is equivalent to write a program which meets the specification A. Normalization of proofs corresponds to evaluation of λ-terms.

Recently, a paradigm similar to Curry-Howard's between Milner's π-calculus and Girard's Linear Logic has been proposed [1, 8]. The fundamental idea of the π-calculus is that of *communication*, or *name passing*, among processes which evolve in a common environment. Communications take place through *ports*, and only ports of matching types can be connected. One-side sequents of Linear Logic provide such a typing: a proof(-net) of the sequent $\vdash A_1, \ldots, A_n$ corresponds to a process with n ports whose types are $A_1, \ldots, A_n$. The CUT rule $\frac{\vdash \Gamma, A \quad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta}$ can be seen as the connection of the plug of shape A of the left device to the corresponding socket (of shape $A^\perp$) of the right device. The other ports become the ports of the resulting device. Normalization of proofs corresponds to message passing.

Such a paradigm sticks Curry-Howard's isomorphism: all the terms having the same type denote processes which interface in the same way with the environment. However, we do not know anything about what the processes do: types say nothing about how processes *behave*. This, indeed, is the topic of this paper. We investigate the use of types in conveying information about the *behaviours* of terms, instead of their interfacing. Thus, while terms correspond to processes, types correspond to behaviours. This paradigm is hence called *behaviours-as-types*.

We present this paradigm directly on a given language, following a general path of Logics and Type Theory. We choose as object language the process algebra due to Winskel, both for its expressive power and its neat categorical interpretation in the category of Labeled Transition Systems [16, 3] (Section 2).

*Supported by the EEC Basic Research Action No.6454, *Confer*.

Then, we define the "behavioural types" for this algebra (Section 3.1). They turn to be very close to the *propositional modal μ-calculus*; for this reason they have been called *modal μ-types*. This is not so surprising, because the modal μ-calculus, introduced by Kozen as an extension of Dynamic Logic [12], seems to be very suited for expressing properties of concurrent systems [3, 4, 15].

However, the meaning of modal μ-types is *not* the same of formulæof the modal μ-calculus: the meaning of a type should be, roughly, a class of processes whose behaviour is that described by the type. The model for these types will be a category $\mathbf{U}$, whose objects are classes of "equally-behaving" LTS. That is, $\mathbf{U}$ has to point out the relevant notion of "behaviour" we want to represent by types.

Now, stating precisely a satisfactory notion of "behaviour" is very difficult, in general. It is easier, and often sufficient, to be able to *compare* behaviours, that is, to say when two terms will behave in the same way. In non-concurrent systems, it is customary speaking of *operational equivalence*; in concurrency, the notion of "equivalent behaviour" is usually taken to be the well-known *bisimilarity* [13]. Therefore, the construction of $\mathbf{U}$ is supposed to yield some link between the notion of "having the same type" and that of "being bisimilar". In fact, it will be proved that *two terms are bisimilar if and only if they share a type*. This result will be called *behaviours-as-types principle*,[1] and it will proved in Section 4.

While the objects of $\mathbf{U}$ are classes of bisimilar LTS, morphisms turn out to be *inclusions*. This means that the same behaviour can be described in several different ways, more or less generally. Hence, morphisms form a *subtyping relation*. This leads us to introduce the syntactic judgment of subtyping, "$A \leq B$", beside the typing judgment "$t : A$".

Having defined the typing and subtyping judgments and their meaning, our aim is now to develope a *typing system*. It will be a Natural-Deduction style proof system for assigning types to terms starting from a given set of hypotheses, that is, for proving sequents of the form $\Gamma \vdash t : A$. For giving an interpretation to these sequents, we need to introduce the *logical consequence* $\Gamma \models t : A$ (Section 3.2). The typing system will be sound and complete with respect to $\models$ (Section 5). This means that, for instance, a proof of $\emptyset \vdash t : A$ guarantees that t has the behaviour A (in every context). Such a system can be used, therefore, for *verification* of terms, that is, concurrent systems.

[1] A *principle* is something we state and prove, like a theorem, while a *paradigm* is a "meta-level" interpretation.

Moreover, the use of a Natural Deduction style system allows us to extend naturally the notion of typing to terms with free process names. According to the behaviour-as-types paradigm, a context can be seen as a declaration of "external modules": the sequent $\Gamma \vdash t : A$ can be read "if, for each $x_i : A_i$ in Γ, we are given a system x_i whose behaviour is A_i, then t has the behaviour A". This holds because the system enjoys a fundamental property of every "good" typing system, that is, the *substitution lemma* (Section 6). It states that names in the hypotheses have to be seen as placeholders for more complex, but with the same type, terms. As a consequence of this, the type system is *compositional*. As pointed out in [4], compositionality is a desiderable property of verification systems because it yields

– *modularity:* if we change part of a system we have certified, only the verification of the changed part have to be redone;

– *abstraction:* during the verification or the synthesis of a system, we can have also undefined parts; all we need is to assume their behaviour.

– *process transformations:* we can replace subparts of a system with other equally-behaving modules. For instance, optimizations rely on this feature;

– *decomposition:* the verification of a given term can be reduced to the verification of smaller terms;

– *reusability:* a system which has been verified once, can be reused in the synthesis or the verification of larger systems. In this way we can build incrementally a *library* of verified processes.

For these reasons, compositional proof systems for verifying properties of concurrent systems are very important and studied in literature. We refer for such a system for the modal μ-calculus (which has been of some inspiration for this work) and for further references to [4]. However, our approach is radically different, because in our interpretation the meaning of a formula is not the usual set of closed terms which verify it. Indeed, the failure of the naïf subject reduction theorem (Section 7) points out that the behaviours-as-types paradigm is "orthogonal" to Curry-Howard-like paradigms. The reason of this is in the different nature of the information carried by types and formulæ: behaviours are not invariant under computations. This point also out that "process computation" is not the right "reduction notion" for behavioural types. In Section 7 we outline a more suited Subject Reduction Theorem, where the notion of reduction adopted is that of *normalization of process terms*. This approach is directly suggested by comparing the theory of behavioural types with that of typed λ-calculus.

2 Winskel's Process Algebra

In this section we recall the process algebra due to Winskel. The following definitions are from [16].

Definition 1 (*Proc*) *Given a set of basic actions Act, the set Proc of* process terms *is as follows:*

$$t ::= nil \mid a.t \mid t_0 + t_1 \mid t_0 \times t_1 \mid t\restriction\Lambda \mid t\{\Xi\} \mid recx.t \mid x$$

where $a \in Act$, $\Lambda \subset Act_*$, x ranges over the set of process names *Nam, Ξ on the set of* relabelings. *We require x to be guarded in $recx.t$. Let* $\mathrm{FN}(t)$ *denote the set of free process names in t.*

Each constructor of this algebra has a direct categorical interpretation in the following category of *labeled transition systems* (see [16]):

Definition 2 (T) *A* labeled transition system *is a 4-tuple $\langle S, i, L, tran\rangle$, where S is a set of* states, *with initial state i; L is a set of* labels*; $tran \subseteq S \times L \times S$ is a* transition relation. *Let $T_j = \langle S_j, i_j, L_j, tran_j\rangle$, where $j = 1, 2$, be two transition systems: a* ts-morphism *(simply morphism) $f : T_1 \to T_2$ is a pair $\langle\sigma, \lambda\rangle$ such that $\sigma : S_1 \to S_2$ is a set morphism (preserving initial object), $\lambda : L_1 \to L_2$ is a partial set morphism and together they satisfy*

1. *if $(s, a, s') \in trans_1$ and $\lambda(a)$ is defined, then $(\sigma(s), \lambda(a), \sigma(s')) \in Trans_2$;*
2. *if $(s, a, s') \in trans_1$ and $\lambda(a)$ is undefined, then $\sigma(s) = \sigma(s')$.*

The category $\mathbf{T}$ *has labeled transition systems as objects, and ts-morphisms as arrows.*

In the following we will consider just the subcategory $\mathbf{T}_R$, such that its objects are finitely-branching LTS's where every state is *reachable* from the initial one. $\mathbf{T}_R$ has a very rich categorical structure: as an example, it admits both binary products and coproducts, corresponding respectively to pairing and disjoint union (with the merging of the initial objects, in the second case). Moreover, classical constructions on transition systems such as renaming, restriction and fixpoint have a nice categorical interpretation: we refer to [16] for more details. Here, we are just interested in stressing that these constructions presents not only an intuitive, set-theoretic definition, but they can actually be given in terms of (co)limits over $\mathbf{T}_R$: this fact will play a crucial rôle when defining the notion of *interpretation* for terms and types.

Since we are interested in modeling *Proc* for a given set of actions, we will just consider a *fibre* of the category of LTS's, that's to say, the subcategory such that all its objects have the same set of label, and label morphisms always are the identities. Let us denote $\mathbf{L}$ the fibre over the label set Act_*.

Definition 3 (Interpretation of process terms) *Let $\rho : Nam \to \mathbf{L}_O$ be a process assignment; then the* interpretation of the process term t on ρ *is as follows:*

$$\begin{array}{ll} [\![nil]\!]\rho = nil & [\![t_1 \times t_2]\!]\rho = [\![t_1]\!]\rho \times [\![t_2]\!]\rho \\ [\![x]\!]\rho = \rho(x) & [\![t_1 + t_2]\!]\rho = [\![t_1]\!]\rho + [\![t_2]\!]\rho \\ [\![a.t]\!]\rho = a.[\![t]\!]\rho & [\![t\{\Xi\}]\!]\rho = ([\![t]\!]\rho)\{\Xi\} \\ [\![t\restriction\Lambda]\!]\rho = ([\![t]\!]\rho)\restriction\Lambda & [\![recx.t]\!]\rho = \mu\tau.[\![t]\!](\rho[\tau/x]) \end{array}$$

where nil denotes the LTS $\langle\{i\}, i, Act_*, \emptyset\rangle$.

The one state LTS *nil* is initial in $\mathbf{L}$; the prefix operation $a.T$ simply adds a new state i_0 to T, makes it initial and inserts an arc from i_0 to the previous initial state; the restriction operator $\restriction\Lambda$ deletes all the arcs such that their label are not in Λ (and deletes also all the nodes that are no more reachable); the fixpoint operator $\mu\tau$ just "unfolds" a given T with respect to the variable τ, producing an infinite LTS; and so on. We refer the reader to [16] for further details.

Two interpretations can differ just for open terms. On closed terms, any interpretation assigns to a process the same LTS, providing a description of its "behaviour". Usually, such a description is too intensional, making different processes that intuitively behave in the same way. The standard way out of this *empasse* is given by the notion of *bisimilarity*: a (maximal) equivalence over (usually closed) terms, equating those with the "same observational behaviour". We refer the reader to [13] for further details. Here we simply recall some definitions and results from [16, 11], showing how bisimilarity can be expressed in terms of properties of the morphisms in $\mathbf{L}$.

Definition 4 (Open map) *A morphism $\sigma : T \to T_1$ in* $\mathbf{L}$ *is an* open map *iff for all the states s of T, if $(\sigma(s), a, s') \in tran_{T_1}$ then exists u in T such that $(s, a, u) \in tran_T$ and $\sigma(u) = s'$.*

This is not the most general definition of open map (see [11]), but it is proved to be equivalent as far as we consider only a fibre of $\mathbf{T}_R$ ([11, Proposition 1]).

Definition 5 (L-bisimilarity) *Two transition systems T_1, T_2 in* $\mathbf{L}$ *are* L-bisimilar *if and only if there exists a transition system T and a pair of open maps from T to, respectively, T_1 and T_2.*

L-bisimilarity is an equivalence relation. It is clearly reflexive and symmetric. Transitivity follows from the existence of pullbacks in the fibre $\mathbf{T}_R$. Given three LTS's $T_j = \langle S_j, i_j, L_j, tran_j \rangle$ for $j = 1 \ldots 3$ and two open maps $\sigma_j : T_j \to T_3$, the associated pullback is the triple $pb(\sigma_1, \sigma_2) = \langle T, \pi_1, \pi_2 \rangle$: $T = \langle S, (i_1, i_2), Act_*, tran \rangle$, where $S = \{(s_1, s_2) \mid \sigma_1(s_1) = \sigma_2(s_2)\}$ and $((s_1, s_2), a, (s_1', s_2')) \in tran \iff (s_1, a, s_1') \in tran_1$ and $(s_2, a, s_2') \in tran_2$, while π_1, π_2 are the obvious projections.

Also pushouts exist in $\mathbf{L}_{open}$ for any pair of coinitial open maps. Given three LTS's $T_j = \langle S_j, i_j, L_j, tran_j \rangle$ for $j = 1 \ldots 3$ and two open maps $\sigma_j : T_3 \to T_j$, the associated pushout is the triple $po(\sigma_1, \sigma_2) = \langle T, \eta_1, \eta_2 \rangle$, $T = \langle S, i, Act_*, tran \rangle$, where
– S is the pushout of σ_1 and σ_2 in **Set**, that is, $S = (S_1 + S_2)/R$ where R is the least equivalence relation containing the pairs $\{\langle (1, \sigma_1(s)), (2, \sigma_2(s)) \rangle \mid s \in S_3\}$;
– $tran = \{\langle [s], a, [s'] \rangle \mid \exists \bar{s} \in [s], \bar{s}' \in [s'].(\bar{s}, a, \bar{s}') \in tran_1 \cup tran_2\}$;
– $\eta_i : S_i \to S$ are the obvious injections modulo R: $\eta_i(s) = [s]$.

Moreover, whenever there exists an open map from T to T' then those two LTS's are L-bisimilar, since identity morphisms are always open maps. All this is summed up in the next proposition, exactly relating L-bisimilarity on transition systems and strong bisimilarity over terms.

Proposition 1 ([11, Theorem 2]) *Given two (not necessarily closed) terms t_1, t_2, then they are strongly bisimilar according to [13] iff for each interpretation ρ then $[\![t_1]\!]\rho$ and $[\![t_2]\!]\rho$ are L-bisimilar.*

3 Modal μ-Types

3.1 Syntax and Semantics

In this section we introduce the *modal μ-types*. They are called so because they are close to the modal μ-calculus [12]. Actually, this language is a subset of the *extended modal μ-calculus* adopted in [4].

Definition 6 (MT) *Given a set of basic actions Act, the set MT of* modal μ-types *is as follows:*

$$A ::= 0 \mid A_1 \vee A_2 \mid A_1 \wedge A_2 \mid \langle a \rangle A \mid A \restriction \Lambda \mid A\{\Xi\} \mid p \mid \mu p.A$$

where $a \in Act$, $p \in TV$ (type variables). We require p to be guarded in $\mu p.A$. Let $\mathrm{FV}(A)$ denote the set of free type variables in A.

As remarked in the introduction, types are intended to denote classes of processes which share the same behaviour, where the notion of "same behaviour" is intuitively described by bisimilarity. Hence, each type has to be interpreted as an object of a suitable category $\mathbf{U}$, such that the arrows of $\mathbf{U}$ could be considered as "bisimulation preserving" morphisms.

Definition 7 (Connected components) *Let T be an LTS. We define the* connected component $C(T)$ *of T as the set of objects of $\mathbf{L}_{open}$ reachable from T. T is called a* generator *of $C(T)$.*

Let $C(\mathbf{L}_{open})$ be the collection of connected components of the objects of $\mathbf{L}_{open}$. Its elements are sets, obviously ordered by set-inclusion: $\langle C(\mathbf{L}_{open}), \subseteq \rangle$ is a partial order. Next proposition states that this ordering reflects the existence of suitable open maps.

Proposition 2 *Let T_1, T_2 be LTS's. Then $C(T_1) \subseteq C(T_2)$ iff there exists an open map $f : T_2 \to T_1$.*

Since pushouts are defined in $\mathbf{L}_{open}$, next result easily follows from the previous proposition.

Proposition 3 *Let T_1, T_2 be bisimilar LTS's. Then there exist LTS's T_g, T_l such that $C(T_l) \subseteq C(T_i)$ and $C(T_i) \subseteq C(T_g)$ for $i = 1, 2$.*

Note that, in general, for an element u of $C(\mathbf{L}_{open})$ there could be more than one generator. Anyway, it can be shown that these generators are isomorphic.

Proposition 4 *Let T_1, T_2 be LTS's. $C(T_1) = C(T_2)$ iff T_1, T_2 are isomorphic in $\mathbf{L}_{open}$.*

It is well known that any given partial order $\langle P, \leq \rangle$ induces a category: its objects are the elements of P, while the arrows are freely generated from the ordering relation. In the following, we denote with $\mathbf{U}$ the category associated to the partial order $\langle C(\mathbf{L}_{open}), \subseteq \rangle$.

Definition 8 (Interpretation of modal μ-types) *Let $\xi : TV \to \mathbf{U}_O$ be a type assignment; the* interpretation of types on ξ *is defined as follows:*

$$\begin{array}{ll} [\![0]\!]\xi = C(nil) & [\![\langle a \rangle A]\!]\xi = C(a.\gamma([\![A]\!]\xi)) \\ [\![p]\!]\xi = \xi(p) & [\![A \restriction \Lambda]\!]\xi = C(\gamma([\![A]\!]\xi) \restriction \Lambda) \\ [\![A_1 \wedge A_2]\!]\xi = C(\gamma([\![A_1]\!]\xi) \times \gamma([\![A_2]\!]\xi)) & \\ [\![A_1 \vee A_2]\!]\xi = C(\gamma([\![A_1]\!]\xi) + \gamma([\![A_2]\!]\xi)) & \\ [\![A\{\Xi\}]\!]\xi = C(\gamma([\![A]\!]\xi)\{\Xi\}) & \\ [\![\mu p.A]\!]\xi = C(\mu T.\gamma([\![A]\!]\xi[T/p])) & \end{array}$$

where γ associates to each object of $\mathbf{U}$ one of its generators.

The definition is well-given, since all the categorical constructions involved are defined up-to-isomorphism: the not-uniqueness of generators is then ininfluential.

3.2 The Theory of Types

In this section we develope further the theory of modal μ-types types. We define two syntactic judgments:

the **typing judgment,** which is of the form "$t : A$", read "t has type A"; roughly, it means "t belongs to the type A", or better, t *has the behaviour* A. Thus, this judgment investigates the *inside* of objects of $\mathbf{U}$;

the **subtyping judgment,** which is of the form "$A \leq B$", read "A is a subtype of B"; roughly, it means "the meaning of A is a subset of the meaning of B," or better, *the behaviour* B *is a generalization of the behaviour* A. Thus, this judgment investigates the *morphisms* among objects of $\mathbf{U}$.

As usual in Logics, the meaning of judgments is given formally by Tarski's *satisfaction*:

Definition 9 (Satisfaction) *Let* $\xi \in TV \to \mathbf{U}_O$, $A, B \in MT$, $\rho \in Nam \to \mathbf{L}_O$, $x \in Nam$. *We say that*
1. ξ satisfies $A \leq B$ ($\xi \models A \leq B$) *iff* $[\![A]\!]\xi \subseteq [\![B]\!]\xi$;
2. ξ, ρ satisfy $t : A$ ($\xi, \rho \models t : A$) *iff* $[\![t]\!]\rho \in [\![A]\!]\xi$.

If a judgment is satisfied by every assignment, then it is said to be *valid*. Validity is the usual notion of "theoremhood" of Logics, and it is usually adopted in Hilbert style proof systems. However, since our aim is to introduce a Natural Deduction style typing system (Section 5), the fundamental notion is that of *consequence*: *which* assertions follow from a given set of *hypotheses*. (This is also one of the reasons for using other styles of proof systems, such as Natural Deduction or Sequent Calculus; for a deeper discussion we refer to [6, 10]). For defining the notion of logical consequence, we need to introduce *contexts*:

Definition 10 (*Ctxt*) *The set Ctxt of* contexts *is defined as follows:*

- $\emptyset \in Ctxt$ *(the empty context);*
- *if* $\Gamma \in Ctxt$, $x \in Nam$ *does not appear in* Γ, *and* $A \in PT$ *is a type, then* $\Gamma, x : A \in Ctxt$;
- *if* $\Gamma \in Ctxt$, $A \in MT$, $p \in TV$ *does not appear free in* Γ, *then* $\Gamma, p \leq A \in Ctxt$ *and* $\Gamma, A \leq p \in Ctxt$.

We will denote by $\mathrm{DN}(\Gamma)$ *the set of process names declared by* Γ, *by* $\mathrm{FN}(\Gamma)$ *the set of process names occurring free*[2] *in* Γ, *and by* $\mathrm{FV}(\Gamma)$ *the set of type variable occurring free in the types in* Γ.

[2]Notice that $\mathrm{DN}(\Gamma) \subseteq \mathrm{FN}(\Gamma)$, because x is free in $x : t$.

Therefore, a context declares types of process variables and the "value" of type variables. Satisfaction is straightforwardly extended to contexts:

Definition 11 *Let* $\xi \in TV \to \mathbf{U}_O$, $\rho \in Nam \to \mathbf{L}_O$, $\Gamma \in Ctxt$. *We say that* ξ satisfies Γ *($\xi \models \Gamma$) iff for all* $(A \leq B) \in \Gamma : \xi \models A \leq B$, *and* ξ, ρ satisfy Γ *($\xi, \rho \models \Gamma$) iff* $\xi \models \Gamma$ *and for all* $(x : A) \in \Gamma : \xi, \rho \models x : A$.

Finally, we formalize the notion of "logical consequence" by defining the *consequence relation* between contexts and judgments:

Definition 12 ($\models$) *Let* $\Gamma \in Ctxt$, $A, B \in MT$, $t \in Proc$. *We say that*

– $A \leq B$ *is a* consequence *of* Γ ($\Gamma \models A \leq B$) *if for all* ξ, *if* $\xi \models \Gamma$ *then* $\xi \models A \leq B$;

– $t : A$ *is a* consequence *of* Γ ($\Gamma \models t : A$) *if for all* ξ, ρ, *if* $\xi, \rho \models \Gamma$ *then* $\xi, \rho \models t : A$.

That is, if we have an interpretation of variables and names which satisfies the assumptions, then also the consequence has to be satisfied. Hence, contexts could be seen as constraints on type and term assignments. This allow us to reason also on *open* process terms and types. We will come back on this in Section 6.

Proposition 5 *For* $\Gamma \in Ctxt$, J *a judgment:*

1. *if* $\Delta \supset \Gamma$ *and* $\Gamma \models J$ *then* $\Delta \models J$;
2. *if* $\Gamma \models t : A$ *and* $\Delta = \{A' \leq B' \mid (A' \leq B') \in \Gamma\} \cup \{(x : A') \in \Gamma \mid x \in \mathrm{FN}(t)\}$ *then* $\Delta \models t : A$.

Obviously, each term has some behaviour (at least, the behaviour of the corresponding LTS), hence it is always typable in a proper context:

Proposition 6 (Typability) *For* $t \in Proc, \Gamma \in Ctxt$, *if* $\mathrm{FN}(t) \subseteq \mathrm{DN}(\Gamma)$ *then there is a type* A *such that* $\Gamma \models t : A$, *and, for* ξ, ρ *such that* $\xi, \rho \models \Gamma$: $[\![A]\!]\xi = C([\![t]\!]\rho)$.

We call such A the *canonic type of* t *in the context* Γ.

On the other hand, each object of $\mathbf{U}$ is generated by some term, hence each type should be inhabited by at least one term, in a proper context:

Proposition 7 (Type inhabitation) *Let* $A \in MT$, $\Gamma \in Ctxt$ *such that for all* $p \in \mathrm{FV}(A)$ *there is* $(x : p) \in \Gamma$. *Then, there is a process* t *such that* $\Gamma \models t : A$, *and, for* ξ, ρ *such that* $\xi, \rho \models \Gamma$: $[\![A]\!]\xi = C([\![t]\!]\rho)$.

We call such t the *canonic term of* A *in the context* Γ.

The following result follows from Proposition 2, and states that $\leq$ is a subtyping relation:

Proposition 8 (Subtyping) *If* $\Gamma \models t : A_1$ *and* $\Gamma \models A_1 \leq A_2$ *then* $\Gamma \models t : A_2$

4 Behaviours-As-Types

In this section we state and prove the *behaviours-as-types* principle.

Giving a general notion of "behaviour" is very difficult. It is much easier to compare behaviours. The notion of "equivalent behaviour" usually adopted in concurrency is Milner's bisimilarity [13]. Hence, we introduce a new syntactic judgment between terms, "$t_1 \approx t_2$" (read "t_1 behaves as t_2"), for comparing term behaviours within contexts:

Definition 13 *For $t_1, t_2 \in Proc, \Gamma \in Ctxt$, we say that $t_1 \approx t_2$ is a consequence of Γ ($\Gamma \models t_1 \approx t_2$) iff for all ξ, ρ, if $\xi, \rho \models \Gamma$ then $[\![t_1]\!]\rho \sim [\![t_2]\!]\rho$.*

Notice that the meaning of $\approx$ is *not* defined by using types or the category $\mathbf{U}$; in fact, it can be seen as the generalization of bisimulation to open terms.

The link between bisimulation and types is estabilished by the "behaviours-as-types" principle. As we have said before, types should be intended as descriptions of the behaviour of their terms. Hence, two terms which share *at least one* type, no matter which one, behave in the same way. This is proved in the following main theorem.

Theorem 1 (Behaviours-As-Types) *Let $\Gamma \in Ctxt$, $t_1, t_2 \in Proc$. The following are equivalent:*

1. *exists A such that $\Gamma \models t_1 : A$ and $\Gamma \models t_2 : A$*
2. *$\Gamma \models t_1 \approx t_2$.*

Proof. (Sketch) ($1 \Rightarrow 2$) If two LTS's T_1, T_2 belong to the same u in $\mathbf{U}$, then there are two open maps $h_i : \gamma(u) \to T_i$, $i = 1, 2$, and hence T_1, T_2 are bisimilar.

($2 \Rightarrow 1$) For Proposition 6, let A_i be the canonic type of t_i, that is $\Gamma \models t_i : A_i$. Define $A \stackrel{\text{def}}{=} ((A_1 \times A_2)\restriction\Delta)\{\Xi\}$ where $\Delta = \{a \times a \mid a \in Act_*, a \neq *\}$ and $\Xi(b) = \begin{cases} a & \text{if } b = a \times a \\ b & \text{otherwise} \end{cases}$. For Proposition 8, it is sufficient to prove that $\Gamma \models A_i \leq A$ for $i = 1, 2$. We prove it for $i = 1$, the other case being similar.

It is easy to see that the canonic term of A is $t \stackrel{\text{def}}{=} ((t_1 \times t_2)\restriction\Delta)\{\Xi\}$. Now, take ξ, ρ such that $\xi, \rho \models \Gamma$; for canonicity, we need only to prove that there is a open map $f_1 : T \to T_1$, where $T_i \stackrel{\text{def}}{=} [\![t_i]\!]\rho$ and $T \stackrel{\text{def}}{=} [\![t]\!]\rho = ((T_1 \times T_2)\restriction\Delta)\{\Xi\}$. Now, by definition states of T are pairs (s_1, s_2) where s_i is a state of T_i. Define, hence, $f_1((s_1, s_2)) = s_1$. Trivially this is a morphism; we prove it is a open map, that is:

> for all (s_1, s_2) reachable in T, if $(s_1, a, s_1') \in tran_{T_1}$ then there is s_2' in T_2 such that $((s_1, s_2), a, (s_1', s_2')) \in tran_T$.

$$\textsc{Prod}\ \frac{t_1 : A_1 \quad t_2 : A_2}{t_1 \times t_2 : A_1 \wedge A_2} \qquad \textsc{Rlb}\ \frac{t : A}{t\{\Xi\} : A\{\Xi\}}$$

$$\textsc{Sum}\ \frac{t_1 : A_1 \quad t_2 : A_2}{t_1 + t_2 : A_1 \vee A_2} \qquad \textsc{Sub}\ \frac{t : A \quad A \leq B}{t : B}$$

$$\textsc{Rstr}\ \frac{t : A}{t\restriction\Lambda : A\restriction\Lambda} \qquad \textsc{Pref}\ \frac{t : A}{a.t : \langle a \rangle A} \qquad \textsc{Nil}\ \frac{}{nil : 0}$$

$$\textsc{Rec}\ \frac{t : A}{recx.t : \mu p.A}\ (x : p)$$

x, p do not appear free in any undischarged assumption

Figure 2: The Typing System $T_{\mu K}$.

By definition of T, this is equivalent to prove that

> for all (s_1, s_2) reachable in T, if $(s_1, a, s_1') \in tran_{T_1}$ then there is s_2' in T_2 such that $(s_2, a, s_2') \in tran_{T_2}$.

This is easily proved by induction on the length of the path which leads to (s_1, s_2), because T_1 and T_2 are bisimilar by hypothesis. □

Indeed, in Theorem 1 one can see the coinductive nature of bisimulation. We need to find *one* type for both t_1 and t_2, in order to state $t_1 \approx t_2$, as well as we need to find *one* bisimilarity relation R between t_1 and t_2 for stating $t_1 \sim t_2$.

5 The Typing System

The relation $\models$ is a purely semantic notion: it is defined by means of the interpretation of terms and types. Following the general path of Logics, in this section we introduce a sound and complete proof system for representing $\models$ via syntactic objects, i.e. *proofs*.

The system is a Natural Deduction style typing system composed by two parts:

- the system $S_{\mu K}$ (Fig.1) which derives the subtyping judgment "$\leq$", and
- the system $T_{\mu K}$ (Fig.2) which derives the typing judgment ":".

The following definition recalls the standard proof-theoretic notion of natural deduction proof (see [14]):

Definition 14 ($\vdash$) *Let J be a typing or subtyping judgment, and $\Gamma \in Ctxt$. A* (natural deduction) proof π of J from Γ *(denoted by $\pi : \Gamma \vdash J$) is a finite tree built and labeled according to the rules of $T_{\mu K}$ and $S_{\mu K}$, and such that the label of the root is J and those of the leaves are in Γ.*

$A \leq A$ TRAN $\dfrac{A \leq B \quad B \leq C}{A \leq C}$ OR $\dfrac{C \leq A \quad C \leq B}{C \leq A \vee B}$ CNGR $\dfrac{A \leq B}{C\,[A/p] \leq C\,[B/p]}$ no captured variable occurs free in any assumption

μ-I $\dfrac{(p \leq A)}{\dfrac{A_1 \leq A_2}{\mu p.A_1 \leq \mu p.A_2}}$ $\dfrac{(A \leq p)}{\dfrac{A_1 \leq A_2}{\mu p.A_1 \leq \mu p.A_2}}$ p does not occur free in any undischarged assumption, and A_1, A_2 are unwindings of A on p

$0 \star A = A \qquad (A \star B) \star C = A \star (B \star C) \qquad A \star B = B \star A \qquad \star \in \{\vee, \wedge\}$

$\mu p.A = A\,[\mu p.A/p] \qquad (\mu p.A)\restriction\Lambda = \mu p.(A\restriction\Lambda)$

$\Gamma \vdash A\restriction\Lambda = A \;\; if \;\; Acts_\Gamma(A) \subseteq \Lambda \qquad A\{Id\} = A$

$\Gamma \vdash A\{\Xi_1\} = A\{\Xi_2\} \;\; if \;\; \Xi_1\restriction Acts_\Gamma(A) = \Xi_2\restriction Acts_\Gamma(A) \qquad (\mu p.A)\{\Xi\} = \mu p.(A\{\Xi\})$

$(A_1 \vee A_2)\restriction\Lambda = (A_1\restriction\Lambda) \vee (A_2\restriction\Lambda) \qquad (A\{\Xi_1\})\{\Xi_2\} = A\{\Xi_1;\Xi_2\}$

$(A_1\restriction\Lambda_1) \wedge (A_2\restriction\Lambda_2) = (A_1 \wedge A_2)\restriction(\Lambda_1 \times_* \Lambda_2) \qquad (A \vee B)\{\Xi\} = A\{\Xi\} \vee B\{\Xi\}$

$(\langle a\rangle A)\restriction\Lambda = \langle a\rangle(A\restriction\Lambda) \; if \; a \in \Lambda \qquad (A\restriction\Lambda_1)\restriction\Lambda_2 = A\restriction(\Lambda_1 \cap \Lambda_2)$

$(\langle a\rangle A)\restriction\Lambda = 0 \; if \; a \notin \Lambda \qquad (A\{\Xi\})\restriction\Lambda = (A\restriction(\Xi^{-1}(\Lambda)))\{\Xi\}$

$(A_1 \wedge A_2)\{\Xi\} = A_1\{\Xi_1\} \wedge A_2\{\Xi_2\} \;\; if \;\; \Xi(a \times b) = \Xi_1(a) \times \Xi_2(b)$

$$P \wedge Q = \left(\bigvee_i (\langle a_i\rangle P_i)\{\Xi_i\}\right) \wedge \left(\bigvee_j (\langle b_j\rangle Q_j)\{\Phi_j\}\right) \leq$$
$$\left(\bigvee_i (\langle c_i\rangle (P_i\{\Xi_i\} \wedge Q))\{\Xi'_i\}\right) \vee \left(\bigvee_j (\langle d_j\rangle (P \wedge Q_j\{\Phi_j\}))\{\Phi'_j\}\right) \vee \left(\bigvee_{(i,j)} (\langle e_{(i,j)}\rangle (P_i\{\Xi_i\} \wedge Q_j\{\Phi_j\}))\{\Psi_{(i,j)}\}\right)$$

where $\Phi'_j(c) = \begin{cases} * \times \Phi_j(b_j) & \text{if } c = d_j \\ c & \text{otherwise} \end{cases}$ (analogously for Ξ'_i); $\Psi_{(i,j)}(c) = \begin{cases} \Xi_i(a_i) \times \Phi_j(b_j) & \text{if } c = e_{(i,j)} \\ c & \text{otherwise} \end{cases}$;
moreover, the $c_i, d_j, e_{(i,j)}$'s are all different and neither appear in P nor in Q.

Figure 1: The Subtyping System $S_{\mu K}$.

We say that J is derivable *from Γ ($\Gamma \vdash J$) if there is a proof $\pi : \Gamma \vdash J$.*

Some explanations are in order. The equality (actually, congruence) symbol is a syntactic shorthand for both directions of $\leq$. For instance, a proof of $\Gamma \vdash A = B$ means that both $\Gamma \vdash A \leq B$ and $\Gamma \vdash B \leq A$. However, it may be easier to think of "=" as a real judgment.

In $S_{\mu K}$, $Acts_\Gamma(A)$ denotes the set of possible actions which can take place in the behaviour A, within the context Γ. Its definition is the following:

$$\begin{array}{l} Acts_\Gamma(0) = \emptyset \\ Acts_\Gamma(A \vee B) = Acts_\Gamma(A) \cup Acts_\Gamma(B) \\ Acts_\Gamma(A \wedge B) = Acts_\Gamma(A) \times_* Acts_\Gamma(B) \\ Acts_\Gamma(\langle a\rangle A) = \{a\} \cup Acts_\Gamma(A) \\ Acts_\Gamma(A\restriction\Lambda) = Acts_\Gamma(A) \cap \Lambda \\ Acts_\Gamma(A\{\Xi\}) = (Acts_\Gamma(A))\{\Xi\} \\ Acts_\Gamma(\mu p.A) = Acts_\Gamma(A[0/p]) \\ Acts_\Gamma(p) = \begin{cases} Acts_\Gamma(A[0/p]) & \text{if } (p \leq A) \in \Gamma \\ & \text{or } (A \leq p) \in \Gamma \\ Act_* & \text{otherwise} \end{cases} \end{array}$$

The context Γ may constraint the meaning of free variables, because an assumption like $p \leq A$ declares that p and A denote the same behaviour. Therefore, the possible actions of p are all Act_*, if there are no assumptions on p; otherwise, it is the possible action set of the assumption on p.

The rules μ-I could be explained as follows. Let A_1 and A_2 be unwindings of A on p. If p does not appear free in A, then $A_1 = A_2 = A$ and hence, for the unfolding axiom, $\mu p.A_1 = \mu p.A_2$. Othewise, let $p \in \mathrm{FV}(A)$. Then, if we prove that the two unfolding denote the same behaviour whenever p and A are assumed to be the same, then $\mu p.A_1 = \mu p.A_2$. In fact, the assumption $p \leq A$ and $A \leq p$ should be seen as fixpoint equations.

The rule REC could be understood as follows. Let consider the recursive term $recx.t$; following the idea of least fixpoint, each occurrence of x in t should denote the process $recx.t$ itself. If we denote by p the behaviour of x, p has to be the same of the whole term $recx.t$ — that is, the least fixpoint $\mu p.A$. The side condition, similar to the *Eigenvariable Condition* for $\forall$-intro [10, 14], avoids unsound derivations like

$$\dfrac{\dfrac{x : p}{ax : \langle a\rangle p}}{recy.ax : \mu p.\,\langle a\rangle p} \quad \text{that is,} \quad \dfrac{\dfrac{x : p, y : p \vdash x : p}{x : p, y : p \vdash ax : \langle a\rangle p}}{x : p \vdash recy.ax : \mu p.\,\langle a\rangle p}$$

The side condition can also be explained in purely proof-theoretic terms: it is needed in the proof of the *substitution lemma* (Theorem 5, Section 6).

Example 1 We prove that the term $recx.aax$ has

type $\mu p.\,\langle a\rangle p$ in the empty context:

$$\textsc{Sub}\,\dfrac{\textsc{Rec}\,\dfrac{\textsc{Pref}\,\dfrac{\textsc{Pref}\,\dfrac{(x:p)_1}{ax:\langle a\rangle\langle a\rangle p}}{aax:\langle a\rangle\langle a\rangle p}}{recx.aax:\mu p.\,\langle a\rangle\langle a\rangle p}1 \qquad \mu\text{-I}\,\dfrac{\textsc{Cngr}\,\dfrac{(\langle a\rangle p\le p)_2}{\langle a\rangle\langle a\rangle p\le\langle a\rangle p}}{\mu p.\,\langle a\rangle\langle a\rangle p\le\mu p.\,\langle a\rangle p}2}{recx.aax:\mu p.\,\langle a\rangle p}$$

□

It should be noticed that the rules and axioms of $S_{\mu K}$ resemble Milner's *equational laws* for strong bisimulation [13]. Actually, many of them are also similar to rules and axioms of the modal μ-calculus [12]: if the relation $\le$ is intended as a "consequence", or "implication", then several rules become "logical"; e.g. Tran is the cut rule. However, this parallel cannot be pushed too far, since many logical implications (such as "$A\wedge B\le A$") do not hold, and there are no type constructors corresponding to "implication" and "negation"; moreover, 0 is the unity for both $\wedge$ and $\vee$.

The relation $\vdash$ is a "concrete" consequence relation between contexts and judgments. It should be thought as the syntactic counterpart of $\models$. Indeed, the properties we have proved *semantically* for $\models$ in Section 3.2 can be restated and proved *syntactically* for $\vdash$, without need of semantic notions.

Proposition 9 *For $\Gamma\in Ctxt$, J a judgment:*

1. *if $\Delta\supset\Gamma$ and $\Gamma\vdash J$ then $\Delta\vdash J$;*
2. *if $\Gamma\vdash A\le B$ and $\Delta=\{A'\le B'\mid(A'\le B')\in\Gamma\}$ then $\Delta\vdash A\le B$;*
3. *if $\Gamma\vdash t:A$ then $\mathrm{FV}(A)\subseteq\mathrm{FV}(\Gamma)$ and $\mathrm{FN}(t)\subseteq\mathrm{DN}(\Gamma)$;*
4. *if $\Gamma\vdash t:A$ and $\Delta=\{A'\le B'\mid(A'\le B')\in\Gamma\}\cup\{(x:A')\in\Gamma\mid x\in\mathrm{FN}(t)\}$, then $\Delta\vdash t:A$.*

Proof. By structural induction on the proofs. □

Proposition 10 (Canonic typability) *For $t\in Proc$, $\Gamma\in Ctxt$, if $\mathrm{FN}(t)\subseteq\mathrm{DN}(\Gamma)$ then there is A such that $\Gamma\vdash t:A$. Moreover, in this derivation* Sub *is never applied, and hence A is the canonic type of t.*

Proof. By induction on the syntax of t. □

Proposition 11 (Canonic type inhabitation) *Let $A\in MT$, $\Gamma\in Ctxt$, such that for all $p\in\mathrm{FV}(A)$ there is $(x:p)\in\Gamma$. Then, there is t such that $\Gamma\vdash t:A$. Moreover, in this derivation* Sub *is never applied, and hence t is the canonic term of A.*

Proof. By induction on the syntax of A. □

The following results relate the syntactic notion of derivability ($\vdash$) to its semantic counterpart ($\models$):

Theorem 2 $\Gamma\vdash A\le B\iff\Gamma\models A\le B$.

Proof. ($\Rightarrow$) Each rule is sound.

($\Leftarrow$) Lenghty proof (omitted). It can be shown proceeding by induction on the syntax of canonic types A, B, deriving for each finite type a normal form as a disjunction of sequential types (i.e., obtained simply as a sequence of applications of rules Pref, Rlb and Rstr). The rules μ-I and Cngr allow us to deal with recursive types which cannot be sequentialised. □

Theorem 3 $\Gamma\vdash t:A\iff\Gamma\models t:A$.

Proof. ($\Rightarrow$) Each rule is sound, and the subsystem $S_{\mu K}$ is sound for Theorem 2.

($\Leftarrow$) Notice that the canonic type of t is either A or a subtype of A. Therefore, for the Sub rule and the completeness of $S_{\mu K}$ (Theorem 2), we can consider only canonic types. In this case, the thesis is proved by induction on the syntax of t. We will see the most difficult case.

Suppose $\Gamma\models recx.t:\mu p.A$; for the inductive hypothesis and the Rec rule, we have to prove that $\Gamma,x:p\models t:A$ where x,p do not appear free in Γ. Let ξ,ρ such that $\xi,\rho\models\Gamma$ and $\rho(x)\in\xi(p)$; we prove that $[\![t]\!]\rho\in[\![A]\!]\xi$, that is, $C([\![t]\!]\rho)\subseteq[\![A]\!]\xi$.

By induction on A it can be proved that if $u_1\subseteq u_2$ then $[\![A]\!](\xi[u_1/p])\subseteq[\![A]\!](\xi[u_2/p])$. Since $\rho(x)\in\xi(p)$, surely $C(\rho(x))\subseteq\xi(p)$, then

$$\begin{aligned}C([\![t]\!]\rho) &\subseteq [\![A]\!](\xi[p\mapsto C(\rho(x))])\\ &\subseteq [\![A]\!](\xi[p\mapsto\xi(p)])=[\![A]\!]\xi\end{aligned}$$

where the first inclusion holds for canonicity and for the monotonicity of the interpretation (which can be easily proved by induction on t,A). Note that for each type variable q in A there is a y in the correspondent place in t, and an assumption $y:q$ in $\Gamma,x:p$; hence, $\rho(y)\in\xi(q)$. □

Finally, the BAT principle can be syntactically formalized by introducing the following Bat *rule*:

$$\textsc{Bat}\ \frac{t_1:A\quad t_2:A}{t_1\approx t_2}\quad\text{that is,}\quad\frac{\Gamma\vdash t_1:A\quad\Gamma\vdash t_2:A}{\Gamma\vdash t_1\approx t_2}$$

The syntactic and semantic notions of "same behaviour" coincide, which yields the system to be sound and complete:

Theorem 4 (BAT Adequacy) *For all Γ,t_1,t_2:*

$$\Gamma\vdash t_1\approx t_2\iff\Gamma\models t_1\approx t_2$$

Proof. Follows from Theorem 1 and Theorem 3. □

In the case that no hypothesis on process variables and subtyping is assumed, we recover the bisimulation between terms:

Corollary 1 $\emptyset \vdash t_1 \approx t_2 \iff t_1 \sim t_2$.

Example 2 We prove $\emptyset \vdash recx.ax \approx recx.aax$:

$$\textsc{Bat}\dfrac{\textsc{Rec}\dfrac{\textsc{Pref}\dfrac{(x:p)_1}{ax:\langle a\rangle p}}{recx.ax:\mu p.\langle a\rangle p}1 \qquad \begin{matrix}\pi\\ recx.aax:\mu p.\langle a\rangle p\end{matrix}}{recx.ax \approx recx.aax}$$

where π is the proof of Example 1. In the classical theory of bisimilarity, such equivalence is inferred from Milner's lemma of unicity of solutions for (weakly) guarded terms. □

6 Open terms, compositionality and the Cut Rule

As pointed out before, also consequences $\Gamma \vdash t_1 \approx t_2$ where t_1, t_2 have free process names, are given a precise meaning (Definition 13). This allows us to reason also on the behaviour of open terms, in a compositional way (see Section 1).

Compositionality is reflected both in the semantics and in the syntax of our system. Semantically, compositionality corresponds to *transitivity* of the consequence relation $\models$:

Proposition 12 *For $\Gamma \in Ctxt$, $t_1, t_2 \in Proc$, $A_1, A_2 \in MT$, $x \in Nam$, if $\Gamma \models t_1 : A_1$ and $\Gamma, x : A_1 \models t_2 : A_2$ then $\Gamma \models t_2\,[t_1/x] : A_2$.*

More interesting is the syntactic counterpart of transitivity. In proof theory, the "plugging" of terms and proof is represented by the well-known Cut rule:

$$\textsc{Cut}\ \frac{\Gamma \vdash t_1 : A_1 \quad \Gamma, x : A_1 \vdash t_2 : A_2}{\Gamma \vdash t_2\,[t_1/x] : A_2}$$

or, in Natural Deduction style,

$$\frac{\begin{matrix}(x:A_1)\\ t_1 : A_1 \quad t_2 : A_2\end{matrix}}{t_2\,[t_1/x] : A_2}.$$

However, we *do not* need to add this rule to the typing system $T_{\mu K}$, because $T_{\mu K}$ enjoys the following fundamental property of proof and typing systems [7]:

Theorem 5 (Substitution Lemma) *If $\pi_1 : \Gamma \vdash t_1 : A_1$ and $\pi_2 : \Gamma, x : A_1 \vdash t_2 : A_2$ then there is a proof $\pi : \Gamma \vdash t_2\,[t_1/x] : A_2$.*

Proof. By structural induction on π_2.

Base case: $t_1 = y$ for some y and $(y : A_1) \in \Gamma, x : A_2$. Now, if $y \neq x$, then we can just drop the assumption $x : A_2$; otherwise, take $\pi_2' = \pi_1$.

Inductive step: By cases on the last rule applied. We see only two significant cases.

If the last rule is Sub, then π_2 appears as follows:

$$\pi_2 = \begin{cases} \Gamma, x : A_1 \qquad \Gamma \\ \pi_{21} \qquad \pi_{22} \\ t_2 : A \qquad A \leq A_2 \\ \hline t_2 : A_2 \end{cases}$$

We can drop the assumption $(x : A_1)$ for Proposition 9. By the inductive hypothesis, there is $\pi_{21}' : \Gamma \vdash t_2\,[t_1/x] : A$. By applying the Sub rule to π_{21}', π_{22} we obtain $\pi : \Gamma \vdash t_2\,[t_1/x] : A_2$

If the last rule is Rec (which discharges the hypothesis $y : p$), then π_2 is as follows:

$$\pi_2 = \begin{cases} \Gamma, x : A_1, (y : p) \\ \pi_{21} \\ t : A \\ \hline recy.t : \mu p.A \end{cases}$$

where t_2 is $recy.t$ and A_2 is $\mu p.A$. For the side condition of Rec, y has to be different from x. Hence, for the inductive hypothesis, there is a proof $\pi_{21}' : \Gamma, y{:}p \vdash t\,[t_1/x] : A$. By applying the Rec rule to π_{21}' we obtain the proof $\pi : \Gamma \vdash recy.(t\,[t_1/x]) : \mu p.A$. Now, we have to prove that $recy.(t\,[t_1/x])$ is syntactically equal to $(recy.t)\,[t_1/x]$. This is true because $x \neq y$ and, by Proposition 5, $\mathrm{FN}(t_1) \subseteq \mathrm{DN}(\Gamma)$. For the side condition of Rec, $y \notin \mathrm{DN}(\Gamma)$, hence y does not occur free in t_1. □

As a consequence of this, free names in the conclusion of $\Gamma \vdash t_1 \approx t_2$ can be seen as "external modules" whose behaviour is declared by the assumptions in the context. If we instantiate the free names in t_1, t_2 with closed process terms which fulfill these assumptions, we obtain two closed terms whose behaviour is still the same (Theorem 4).

Moreover, if we have proved that in a certain context Γ, a process t has the behaviour A, then we can use this proof as a rule in the inference of behaviours of more complex processes. Equivalentely, we can put t in place of each x if the assumed behaviour of x is supposed to have is the same of what has been previously proved for t. Therefore, the substitution lemma allows

us for *libraries of certified processes*, which can be used in developing modularly larger processes. An application of this can be seen also in Example 2 above, where we reused the proof of the previous Example 1.

7 Final Remarks

Comparison with the Typed λ-calculus. In the paradigm *behaviours-as-types* we have presented, terms correspond to processes and types correspond to behaviours. Comparing with the propositions-as-types paradigm, we lack a connection between invariant properties. In the theory of typed λ-calculus, this invariant property is highlighted by the *subject reduction theorem* (SRT):

$$\text{if } \Gamma \vdash M : A \text{ and } M \to_\beta N \text{ then } \Gamma \vdash N : A$$

This means that the information carried by types is "orthogonal" to the information reduced by the β-rule. However, in the behaviours-as-types paradigm the corresponding naïf SRT

$$\text{if } \Gamma \vdash t : A \text{ and } t \xrightarrow{a} t' \text{ then } \Gamma \vdash t' : A$$

does not hold. This is intuitively correct, because computations are not "orthogonal" to behaviours: in general, t' behaves differently from t.

There are (at least) two solutions to this problem. The first is to restrict our attention to those processes which satisfy the SRT above mentioned. In this way, we committ ourselves to processes whose behaviour is always the same even if they are evolving, like an operating system.

The other approach is to adopt a different notion of process reduction, not necessarely with operational contents. A closer look to the theory of λ-calculus suggests that it should be a notion of *normalization*. In fact, the β-reduction is the reduction of λ-terms in head normal form. Similarly, we should choose as reduction a notion of *normalization of process terms*. For instance, we could define a rewriting relation $\rhd \subseteq \mathit{Proc} \times \mathit{Proc}$ by applying the technique introduced in [2] for finite-state processes: from each bisimulation equation we could obtain a rewriting rule. Hence, the "right" SRT is the following:

$$\text{If } \emptyset \vdash t : A \text{ and } t \rhd t' \text{ then } \emptyset \vdash t' : A$$

Traditionally, in rewriting only closed terms are concerned. As in the case of bisimulation, we think that, within the type-based approach presented in this paper, rewriting can be extended smoothly to open terms where free process variables may have a declared behaviour. We can introduce a new syntactic judgment "$\rhd$" between terms, analoguosly to β-reduction: the intended meaning of $\Gamma \vdash t \rhd t'$ should be "in the context Γ, t is rewritten to t', whose complexity is not greater than those of t." Due to lack of space, we cannot elaborate further on this topic; however, we just point out that the Substitution Lemma suggests immediately the following rewriting rule for taking care of behaviour declaration of free process variables:

$$\frac{\Gamma \vdash t_1 : A_1 \quad \Gamma, x : A_1 \vdash t \rhd t'}{\Gamma \vdash t\,[t_1/x] \rhd t'\,[t_1/x]}$$

or, in Natural Deduction style,

$$\frac{t_1 : A_1 \qquad \overset{(x : A_1)}{t \rhd t'}}{t\,[t_1/x] \rhd t'\,[t_1/x]}$$

Another link between modal μ-types and typed λ-calculus regards recursive types. Typed λ-calculus with recursive types has been thoroughly studied — see e.g. [7]. Although the theory of recursive types for the λ-calculus is quite different from that of modal μ-types, they share the following type-theoretical results:

– *type inhabitation* is trivially decidible;
– *typability* is trivially decidible;
– *type checking* is not decidible;
– *subject reduction* holds.

Type inhabitation and typability are always verified (Propositions 10, 11). Decidibily of type-checking would yield decidibility of bisimilarity. In fact, for t_1, t_2 closed terms, $t_1 \sim t_2$ iff $\emptyset \vdash t_i : A$, $i = 1, 2$, where A is the type defined in the proof of the BAT principle (Theorem 1). Actually, the undecidibility of the ":" judgment is due to the undecidibility of the "$\leq$" judgment.

Other models. The notion of *behavioural types* we have introduced is not strictly limited to the semantic interpretation of processes in the category of LTS's. Actually, we could define the category $\mathbf{U}$ in many different ways, viz. by using comma subcategories, or by choosing as starting point the category of synchronization trees or event structures [16], or even by defining $\mathbf{U} = \mathbf{L}_{open}$. In fact, all the results of the paper could be recovered in these alternative settings: we have preferred our characterization for its set-theoretical flavour.

The fact that there are so many and only apparently different models for concurrency, seems to say

that the right notions of "process denotation" and "process behaviour" are still lacking. A promising new model (the *presheaf model*), strictly related to the notion of open map, has been introduced in [11]; its applications to our paradigm are under investigation.

Another topic related to the lack of a satisfactory model of behaviours is the absence of the "arrow" type constructor. From a constructive point of view, terms of type $A \supset B$ should have a "functional behaviour", that is, they denote higher-order processes. However, the categories adopted in usual models for concurrency are not closed, and hence they do not allow for exponentials: we refer to [5] for an account of this topic, and further references, with regard to the applications to Petri nets.

Acknowledgments

We are most grateful to Martin Abadi for his useful remarks on the preliminar version of this paper. Corrado Priami has been our on-line bibliography about CCS. Bob Kane and many others have inspired the name for the BAT principle.

References

[1] S. Abramsky. Computational interpretation of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.

[2] L. Aceto, B. Bloom, and F. W. Vaandrager. Turning SOS rules into equations. In *Proceedings of the 7th LICS Symposium*, pages 113–124, Santa Cruz, California, 1992. IEEE.

[3] H. R. Andersen. *Verification of Temporal Properties of Concurrent Systems.* Daimi PB-445, Computer Science Department, Århus University, June 1993.

[4] H. R. Andersen, C. Stirling, and G. Winskel. A compositional proof system for the modal μ-calculus. In *Proceedings of the 9th LICS Symposium*, pages 144–153, Paris, July 1994. IEEE.

[5] A. Asperti, G. Ferrari, and R. Gorrieri. Implicative formulæ in the *proofs as computations* analogy. In *Proc. of the* 17^{th} *Symposium on Principles of Programming Languages*, pages 59–71, San Francisco, Jan. 1990. ACM.

[6] A. Avron. Simple consequence relations. *Information and Computation*, 92:105–139, Jan. 1991.

[7] H. Barendregt. *The lambda calculus: its syntax and its semantics.* Studies in Logic and the Foundations of Mathematics. North-Holland, 1984.

[8] G. Bellin and P. J. Scott. On the π-calculus and linear logic. In *Proceedings of MFPS 8*, 1994. To appear in Theoretical Computer Science.

[9] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[10] G. Gentzen. Investigations into logical deduction. In M. Szabo, editor, *The collected papers of Gerhard Gentzen*, pages 68–131. North Holland, 1969.

[11] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. BRICS Report RS-94-7, Department of Computer Science, University of Århus, May 1994.

[12] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27, 1983.

[13] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[14] D. Prawitz. *Natural Deduction.* Almqvist & Wiksell, Stockholm, 1965.

[15] C. Stirling. Modal and Temporal Logics. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 477–563. Oxford University Press, 1992.

[16] G. Winskel and M. Nielsen. Models for concurrency. BRICS Report RS-94-12, Århus University, May 1994. 144 pp. To appear as a chapter in the *Handbook of Logic and the Foundations of Computer Science*, Oxford University Press.