Linearity and Bisimulation

Nobuko Yoshida[1,⋆], Kohei Honda[2,⋆⋆], and Martin Berger[2,⋆⋆]

[1] University of Leicester, U.K.
[2] Queen Mary, University of London, U.K.

Abstract. We introduce a theory of weak bisimilarity for the π-calculus with linear type structure [35] in which we abstract away not only τ-actions but also non-τ actions which do not affect well-typed environments. This gives an equivalence far larger than the standard bisimilarity while retaining semantic soundness. The congruency of the bisimilarity relies on a liveness property at linear channels ensured by typing. The theory is consistently extendible to settings which involve nontermination, nondeterminism and state. As an application we develop a behavioural theory of secrecy for the π-calculus which ensures secure information flow for a strictly greater set of processes than the type-based approach in [20, 23].

1 Introduction

Linearity is a fundamental concept in semantics with many applications to both sequential and concurrent computation. This paper studies how a linear type structure, close to those of Linear Logic [12] and game semantics [4, 22, 24], can be used to give a powerful extension of a basic process equivalence, bisimilarity. We use a linear π-calculus, introduced in [35], which, among others, satisfies a basic liveness property in linear interaction: actions on linear channels always eventually fire. A central idea of our construction is that observables, an underpinning of any behavioural semantics, can be given a radical change in the presence of this liveness and other properties ensured by linear typing: a class of visible interactions with the environment which a typed process is actually engaged in, can be completely abstracted (neglected) away in terms of their semantic effects.

Let us briefly explain the key technical ideas, using a process encoding of a λ-calculus. We first recall that the linear π-calculus in [35] can fully abstractly embed $\lambda_{()\times+}$, the simply typed λ-calculus with unit, products and sums. The encoding $[\![M : \alpha]\!]_u$ for a λ-term $M : \alpha$ in [35] is a typed version of Milner's encoding [26]. We also recall that in $\lambda_{()\times+}$, the following equation is semantically sound: $\Gamma \vdash M_1 = M_2 : \mathtt{unit}$ for any $\Gamma \vdash M_{1,2} : \mathtt{unit}$. In particular, any term of this type is always equated with its unique constant, which we write $\star$. As an example we have the following equation.

$$y : \mathtt{unit} \Rightarrow \mathtt{unit} \vdash \ (y\star) = \star : \mathtt{unit}$$

[⋆] Partially supported by EPSRC grant GR/R33465/01.
[⋆⋆] Partially supported by EPSRC grant GR/N/37633.

If we apply the encoding in [35] to this, we obtain the following two processes:

$$[\![(y\star)]\!]_u \overset{\text{def}}{=} !u(c).\overline{y}(e)e.\overline{c} \qquad\qquad [\![\star]\!]_u \overset{\text{def}}{=} !u(c).\overline{c}.$$

Here $x(y)$ is an input of y via x, $\overline{x}(y)$ is an (asynchronous) output of a fresh name y via x, and ! indicates replication. Thus, the process $[\![(y\star)]\!]_u$, when invoked at u with a continuation c, first asks at y and, after receiving an answer at e, returns to c; while $[\![\star]\!]_u$ immediately answers at the continuation after the invocation. Because of the obvious difference in these actions, we know $[\![(y\star)]\!]_u \not\approx [\![\star]\!]_u$ where $\approx$ is the standard weak bisimilarity. However, since the encoding is fully abstract, the contextual equivalence $\cong_\pi$ in [35] for the linear π-calculus does equate them. Intuitively, this is because the linear type structure allows us to abstract away the additional non-τ-actions in the following way:

1. The action $\overline{y}(e)$ is typed as an output to replication: thus it just replicates a process in the environment without affecting it.
2. The action e is typed as a linear input: hence it necessarily receives its dual output, neither receiving nor emitting non-trivial information.

For these reasons, the additional actions in $[\![(y\star)]\!]_u$ never affect the environment in a way well-typed observers could detect, and are automatically executable, so they behave "as if they were τ-actions", allowing them to be neglected. This suggests the following principle of behavioural semantics in linear processes.

Categorise some of the typed actions as "non-affecting", and abstract away non-affecting actions as if they were τ-actions.

The type structure plays a crucial role in this principle.

Following [6, 11, 15, 18, 33], the linear π-calculus in [35] includes branching and selection, which correspond to sums in the λ-calculus and additives in Linear Logic [12]. A *branching* is an input with I-indexed branches of form $x[\&_{i\in I}(\vec{y}_i).P_i]$, while a *selection* is an output of form $\overline{x}\mathtt{in}_i(\vec{z})Q$. These constructs have the following dynamics: $x[\&_i(\vec{y}_i).P_i]|\overline{x}\mathtt{in}_j(\vec{y}_j)Q \longrightarrow (\boldsymbol{\nu}\,\vec{y}_j)(P_j|Q)$. Now consider another equation in $\lambda_{()\times+}$, which uses sums this time. Let $\mathtt{bool} \overset{\text{def}}{=} \mathtt{unit}+\mathtt{unit}$ below.

$$y : \mathtt{bool} \vdash \ \mathtt{case}\ y\ \mathtt{of}\ \{\mathtt{in}_i() : \mathtt{in}_1(\star)\}_{i\in\{1,2\}} \ = \ \mathtt{in}_1(\star)\ :\ \mathtt{bool}$$

These terms are translated as follows:

$$[\![\mathtt{case}\ y\ \mathtt{of}\ \{\mathtt{in}_i() : \mathtt{in}_1(\star)\}_{i\in\{1,2\}}]\!]_u \overset{\text{def}}{=} !u(c).\overline{y}(e)e[\&_{1,2}.\overline{c}\mathtt{in}_1]$$
$$[\![\mathtt{in}_1(\star)]\!]_u \overset{\text{def}}{=} !u(c).\overline{c}\mathtt{in}_1.$$

Both processes are equated by $\cong_\pi$. Intuitively this is because an input at e in the first process surely arrives (due to liveness at the linear channel e), and regardless of a selected branch, it leads to $\overline{c}\mathtt{in}_1$. We can thus augment the previous principle as follows.

We may abstract away linear branching inputs as far as they lead to the same action in all possible branches.

The precise formulation of this idea is given in Section 2.

Applications. The bisimilarity based on these ideas, which we hereafter call linear bisimilarity, can justify the equations mentioned above, as well as many equations used for definability arguments to prove full abstraction of $\lambda_{()\times+}$ in [35]. As another application, Section 5 discusses a behavioural theory of secure information flow for the π-calculus, which uses a secrecy-sensitive bisimilarity built on the top of linear bisimilarity. The theory ensures secrecy through semantic means for a strictly larger set of processes than the syntactic theory in [23], which itself is powerful enough to embed representative secrecy calculi such as [3, 32]. As a simple example, the theory can justify the safety of the following λ-term via encoding ($\top$ and $\bot$ are high and low secrecy levels, respectively).

$$\texttt{case}\ y^{\top}\ \texttt{of}\ \{\texttt{in}_i() : \texttt{in}_1(\star)\}_{i\in\{1,2\}} : \texttt{bool}^{\bot}$$

which is untypable in standard secrecy typing systems, cf. [3, 32].

One of the technical contributions of the present work is a proof technique for establishing congruency of linear bisimilarity which, among others, uses liveness at linear channels. The proof method is applicable when we extend linear bisimilarity to other type structures involving nontermination [6], nondeterminism [20] and state [23], as well as to their secrecy enhancement. Such extensions are briefly discussed at the end of in Section 5.

Related Work. Since the introduction of Linear Logic [12], linearity has been studied in various semantic and syntactic contexts. In the setting of the π-calculus, linearity and its relationship to contextual equivalences [21, 27] are studied in [25, 30, 34]. In each case, it is shown that linearity induces a strictly larger contextual equivalence than the standard bisimilarities. [30] as well as [5] study typed bisimilarities in which two processes whose actions are equivalent up to forwarding of names are equated. [19] studies an untyped bisimulation in which visible actions can be ignored due to asynchronous observables. [13] studies a process equivalence in which certain actions are ignored due to capabilities assigned to channels via subtyping. While sharing the common orientation towards a larger equality by a refined treatment of observables, the nature of abstraction offered by the present theory differs from these works in two aspects. First, the introduced behavioural equivalence allows us to treat visible interactions which do take place between the process and the environment as if they were internal (silent) actions. Since the use of liveness in linear actions is essential for this abstraction (as shown by the examples above), it would be difficult to apply these existing techniques to obtain the same effect. The second significance of using linear type structures for a behavioural equivalence is that it enables precise embedding of semantics of language constructs including functional [6, 35] and imperative ones [23], which is important for applications. This direction may not have been pursued in the foregoing studies. Combination of existing techniques and the present one would be an interesting subject for further study.

In the analysis of secure information flow, equality over programs often plays a central role, cf. [2, 3, 10, 14, 31]. Among them, [9, 10] present a bisimulation for cryptographic protocols where high-level actions are abstracted away, preceding the behavioural theory of secrecy in Section 5. The main difference from our approach is that [9,10] are based on CCS without using type structure, which would limit expressiveness of the resulting theory. [1] establishes a secrecy theorem for the spi-calculus based on may-equivalence, using type information to control the interface of the attacker. Our approach differs in that it first uses linearity to limit environments, then applies it to information flow analysis by a simple elaboration of channels with security levels. [14] uses a secrecy-sensitive may-equivalence for noninterference in the π-calculus. The use of linear type structure in the present work is the main difference. Finally, the first two present authors proposed, in [20] (with Vasconcelos) and in [23], type systems for the π-calculus which ensures secrecy. The present paper gives a semantic theory of secrecy, extending and complementing the syntactic approach in [20,23].

Outline of the paper. Section 2 briefly reviews the linear π-calculus in [35]. Section 3 introduces linear bisimilarity, whose proof of congruency is given in Section 4. Section 5 discusses an application of the linear bisimilarity to secure information flow analysis. For details of the results and motivation of the syntax and types used in the paper, the reader may refer to [6,35]. The omitted proofs and definitions of this paper can be found in [36].

Acknowledgements. The authors thank Martin Abadi and anonymous referees for their useful comments and suggestions on an early version of this paper.

2 Preliminaries

2.1 Processes and Channel Types

The set of *processes* is given by the following grammar [6,35]. Below and henceforth $x, y, \ldots$ range over a countable set of names.

$$P \ ::= \ x(\vec{y}).P \mid \overline{x}(\vec{y})P \mid x[\&_{i\in I}(\vec{y_i}).P_i] \mid \overline{x}\mathtt{in}_i(\vec{y})P \mid P|Q \ \mid \ (\boldsymbol{\nu}\, x)P \ \mid \ \mathbf{0} \ \mid \ !P.$$

$x(\vec{y}).P$ (resp. $\overline{x}(\vec{y})P$) is a unary input (resp. unary output), while $x[\&_{i\in I}(\vec{y_i}).P_i]$ (resp. $\overline{x}\mathtt{in}_i(\vec{y})P$) is a branching (resp. selection). Bound name passing has essentially equivalent expressive power as free name passing [17,29], and is convenient for obtaining precise correspondence with functional type structures [6,35]. $P|Q$ is a parallel composition, $(\boldsymbol{\nu}\, x)P$ is a restriction, and $!P$ is a replication. In $!P$ we assume P is either a unary or branching input. The reduction relation $\longrightarrow$ is generated from the following rules, closed under output prefix, restriction and parallel composition modulo $\equiv$ [36]. We also set $\twoheadrightarrow \overset{\text{def}}{=} \longrightarrow^* \cup \equiv$.

$$\begin{aligned}
x(\vec{y}).P|\overline{x}(\vec{y})Q &\longrightarrow (\boldsymbol{\nu}\, \vec{y})(P|Q)\\
!x(\vec{y}).P|\overline{x}(\vec{y})Q &\longrightarrow !x(\vec{y}).P|(\boldsymbol{\nu}\, \vec{y})(P|Q)\\
x[\&_i(\vec{y_i}).P_i]|\overline{x}\mathtt{in}_j(\vec{y_j})Q &\longrightarrow (\boldsymbol{\nu}\, \vec{y_j})(P_j|Q)\\
!x[\&_i(\vec{y_i}).P_i]|\overline{x}\mathtt{in}_j(\vec{y_j})Q &\longrightarrow !x[\&_i(\vec{y_i}).P_i]|(\boldsymbol{\nu}\, \vec{y_j})(P_j|Q)
\end{aligned}$$

Action modes, ranged over by $p, \ldots$, are from the sets $\{\downarrow, \mathbf{!}\}$ (written $p_{\mathtt{I}}, \ldots$), and $\{\uparrow, \mathbf{?}\}$ (written $p_{\mathtt{O}}, \ldots$). $\downarrow$ (resp. $\uparrow$) represents linear input (resp. output).

$\mathbf{!}$ means a unique server associated with input replication. Dually $\mathbf{?}$ represents client requests to $\mathbf{!}$. We also use the mode $*$ to indicate uncomposability. The *dual* $\overline{p}$ of p is given by $\overline{\downarrow} = \uparrow$, $\overline{\mathbf{!}} = \mathbf{?}$ and $\overline{\overline{p}} = p$. Then *channel types* are given by the following grammar. For simplicity we assume indices i range over $\{1, 2\}$.

$$\tau \ ::= \ \tau_{\mathtt{I}} \mid \tau_{\mathtt{O}} \mid * \qquad \tau_{\mathtt{I}} \ ::= \ (\vec{\tau})^{p_{\mathtt{I}}} \mid [\&_i \vec{\tau_i}]^{p_{\mathtt{I}}} \qquad \tau_{\mathtt{O}} \ ::= \ (\vec{\tau})^{p_{\mathtt{O}}} \mid [\oplus_i \vec{\tau_i}]^{p_{\mathtt{O}}}$$

Above, $\vec{\tau}$ denotes a vector of channel types. A branching type is sometimes written $[\tau_1 \& \tau_2]^p$ and similarly for selection. We define $\overline{\tau}$, the *dual* of τ, by dualising the action modes and exchanging $\oplus$ and $\&$ in τ. $\mathsf{md}(\tau)$ is $*$ if $\tau = *$, otherwise the outermost action mode of τ.

On types $\odot$ is the least commutative partial operation such that

$$(1)\ \tau \odot \overline{\tau} = * \ \ (\mathsf{md}(\tau) = \downarrow) \qquad (2)\ \ \tau \odot \tau = \tau \ \text{ and } \ \tau \odot \overline{\tau} = \overline{\tau} \ \ (\mathsf{md}(\tau) = \mathbf{?}).$$

Intuitively, (1) says once we compose input-output linear channels it becomes uncomposable (for example, $x.\mathbf{0} \,|\, \overline{x}$ has mode $*$ at x, which is uncomposable with any process which has x). (2) says that a server should be unique, to which an arbitrary number of clients can request interactions (for example, $!x.\mathbf{0} \mid !x.\mathbf{0}$ is never typed because of $()^{\mathbf{!}} \not\asymp ()^{\mathbf{!}}$, while $\overline{x} \,|\, \overline{x}$ is typable by $()^{\mathbf{?}}$ at x, and $!x.\mathbf{0} \,|\, \overline{x} \,|\, \overline{x}$ is so by $()^{\mathbf{!}}$ at x). If $\tau \odot \tau'$ is defined we say they *compose*.

Following [6, 22, 35], we assume the following *sequentiality constraint* (IO-alternation and a unique answer $\uparrow$ in each server type $\mathbf{!}$), which comes from game semantics [4, 22, 24]. We state the constraint only for unary types: for branching/selection types, we require the same constraint for each summand.

- In $(\vec{\tau})^{\downarrow}$, $\mathsf{md}(\tau_i) = \mathbf{?}$ for each $1 \leq i \leq n$. Dually for $(\vec{\tau})^{\uparrow}$.
- In $(\vec{\tau})^{\mathbf{!}}$, $\mathsf{md}(\tau_i) \in \mathbf{?}$ for each $1 \leq i \leq n$ except at most one j for which $\mathsf{md}(\tau_j) = \uparrow$. Dually for $(\vec{\tau})^{\mathbf{?}}$.

2.2 Typing and Typed Processes

An *action type* is a finite acyclic directed graph whose nodes have the form $x : \tau$ such that no names occur twice and each edge is of form $x : \tau \rightarrow x' : \tau'$ with either $\mathsf{md}(\tau) = \downarrow$ and $\mathsf{md}(\tau') = \uparrow$, or $\mathsf{md}(\tau) = \mathbf{!}$ and $\mathsf{md}(\tau') = \mathbf{?}$. We write $A(x)$ for the channel type assigned to x occurring in A. The partial operator $A \odot B$ is defined iff channel types with common names compose and the adjoined graph does not have a cycle. This avoids divergence. For example, $x : \tau_1 \rightarrow y : \tau_2$ and $y : \overline{\tau_2} \rightarrow x : \overline{\tau_1}$ are not composable, hence a process such as $!x.\overline{y} \mid !y.\overline{x}$ is untypable.[1] $\mathsf{fn}(A)$ and $\mathsf{md}(A)$ denote the sets of free names and modes in A, respectively. $A \asymp B$ indicates $A \odot B$ is defined.

Sequents of the linear typing system have the form $\vdash P \triangleright A$.[2] The rules are given in Appendix A. If $\vdash P \triangleright A$ is derivable, we say P *is typable with* A. We sometimes write P^A instead of $\vdash P \triangleright A$. Typable processes are often called *linear processes*.

[1] See Section 2 in [35] or Appendix B in [36] for detailed examples and definitions.

[2] In [35], the main sequent has the shape $\Gamma \vdash P \triangleright A$. This is equivalent to the present one by adjoining Γ to the right-hand side.

Example 1. (linear processes)

1. $\vdash \overline{x} \triangleright x : ()^{\uparrow}$ and $\vdash x.\mathbf{0}|\overline{x} \triangleright x : *$. The former can also be typed with $x : ()^{\mathbf{?}}$.
2. $\vdash !u(c).\overline{c} \triangleright u : (()^{\uparrow})^{\mathbf{!}}$ and $\vdash !u(c).\overline{x}(e)e.\overline{c} \triangleright u : (()^{\uparrow})^{\mathbf{!}} \to x : (()^{\downarrow})^{\mathbf{?}}$.
3. Let $\mathbb{B} = [\varepsilon \oplus \varepsilon]^{\uparrow}$ (where ε is the empty vector). Then $\vdash !u(c).\overline{x}(e)e[.\overline{c}\mathtt{in}_1 \& .\overline{c}\mathtt{in}_2] \triangleright u : (\mathbb{B})^{\mathbf{!}} \to x : (\overline{\mathbb{B}})^{\mathbf{?}}$. Other terms typable with this type include $!u(c).\overline{c}\mathtt{in}_1$ and $!u(c).\overline{x}(e)e[.\overline{c}\mathtt{in}_1 \& .\overline{c}\mathtt{in}_1]$ as well as their symmetric variants.

The following properties of typed terms are from [35]. (2) is a consequence of strong normalisability of linear processes and will play an important role later.

Proposition 1. *1.* (subject reduction) *If* $\vdash P \triangleright A$ *and* $P \twoheadrightarrow Q$ *then* $\vdash Q \triangleright A$.
2. (liveness) *Let* $\vdash P \triangleright A \otimes x : \tau$ *with* $\mathsf{md}(\tau) = \uparrow$ *and* $\mathsf{md}(A) \subseteq \{\mathbf{!}, *\}$ *($\otimes$ is disjoint graph union). Then* $P \twoheadrightarrow P'$ *such that* $P' \equiv \overline{x}(\vec{y})R$ *or* $P' \equiv \overline{x}\mathtt{in}_j(\vec{y})R$.

2.3 Contextual Congruence and Bisimilarity

A relation $\mathcal{R}$ over typed processes is *typed* when $P_1^{A_1} \,\mathcal{R}\, P_2^{A_2}$ implies $A_1 = A_2$. We write $P_1 \,\mathcal{R}^A\, P_2$ when P_1^A and P_2^A are related by a typed relation $\mathcal{R}$. A *typed congruence* is a typed relation which is an equivalence, closed under all typed contexts. The *contextual congruence* $\cong_\pi$ is the maximum typed congruence satisfying the following condition ($\mathbb{B}$ appeared in Example 1).

$$\text{If } P \Downarrow_x^i \text{ and } P \cong_\pi^{x:\mathbb{B}} Q, \text{ then } Q \Downarrow_x^i \qquad (i = 1, 2)$$

where $P \Downarrow_x^i$ means $P \twoheadrightarrow \overline{x}\mathtt{in}_i(\vec{y})P'$. $\cong_\pi$ is maximally consistent in the sense that any addition of equations leads to inconsistency. A more restricted and tractable equality is obtained by labelled transition. Let $l, l', \ldots$ be given by:

$$l \ ::= \ \boldsymbol{\tau} \mid x(\vec{y}) \mid \overline{x}(\vec{y}) \mid x\mathtt{in}_i(\vec{y}) \mid \overline{x}\mathtt{in}_i(\vec{y})$$

If $l \neq \boldsymbol{\tau}$, we write $\mathsf{sbj}(l)$ for the initial free name of l. Using these labels, the typed transition $P^A \xrightarrow{l} Q^B$ is defined as in Appendix B. The weak bisimilarity induced by the transition is denoted $\approx$.

As indicated in the introduction, $\cong_\pi$ is strictly greater than $\approx$. One of the aims of the present work is to fill the gap between $\approx$ and $\cong_\pi$, at least partially, without losing the ease of reasoning of $\approx$.

3 Linear Bisimilarity

3.1 Categorising Actions

We begin our path towards the definition of linear bisimilarity with classifying types according to the following criteria: whether actions typed with these types affect the environment non-trivially; and whether the typed actions are guaranteed to take place. One of the significant aspects of our linear type structure is that types are informative enough to allow this classification.

Definition 1. (affecting and enabled types)

1. τ is *affecting* iff there exist $\vdash P_{1,2} \triangleright x : \tau$ and a typed context $C[\,\cdot\,]$ such that $\vdash C[P_i] \triangleright u : \mathbb{B}$, $C[P_1] \Downarrow^1_u$ and $C[P_2] \Downarrow^2_u$.
2. τ is *enabling* iff $\vdash P \triangleright x : \tau$ implies $P \twoheadrightarrow P' \xrightarrow{l}$ such that $\mathsf{sbj}(l) = x$. τ is *enabled* if $\overline{\tau}$ is enabling.

Example 2. (affecting and enabled types)

1. $\mathbb{B}$, $(\mathbb{B})^{\mathbf{!}}$ and $((\mathbb{B})^{\mathbf{!}})^{\uparrow}$ are affecting but $((\mathbb{B})^{\mathbf{!}})^{\mathbf{?}}$ and $(((\mathbb{B})^{\mathbf{!}})^{\mathbf{?}})^{\downarrow}$ are not. It is notable that no τ such that $\mathsf{md}(\tau) \in \{\mathbf{?}, \downarrow\}$ is affecting.
2. Any τ such that $\mathsf{md}(\tau) \in \{\downarrow, \uparrow, \mathbf{!}\}$ is enabling, while any τ such that $\mathsf{md}(\tau) =$ $\mathbf{?}$ is not. Hence all and only enabled types are τ such that $\mathsf{md}(\tau) = \{\downarrow, \uparrow, \mathbf{?}\}$.

As suggested in the above example, we have an easy rule to determine whether a type is affecting or not, based on the shape of types.

Proposition 2. *Define* Aff *as the smallest set of types generated by:*

- $[\oplus_{1,2}\vec{\tau_i}]^{\uparrow} \in \mathsf{Aff}$.
- $(\tau_1..\tau_n)^{\uparrow} \in \mathsf{Aff}$ *and* $(\tau_1..\tau_n)^{\mathbf{!}} \in \mathsf{Aff}$ *when* $\tau_i \in \mathsf{Aff}$ *for some* i $(1 \le i \le n)$.
- $[\&_{1,2}\tau_{i1}..\tau_{in_i}]^{\mathbf{!}} \in \mathsf{Aff}$ *when* $\tau_{ij} \in \mathsf{Aff}$ *for some* i *and* j $(i \in \{1, 2\}, 1 \le j \le n_i)$.

Then τ *is affecting iff* $\tau \in \mathsf{Aff}$.

Using the classification of types given above, we can classify actions of well-typed processes. First, an action annotated with an action type, say l^A, is called a *typed action* if the shape of l conforms to A. For example, if $l = \overline{x}\mathtt{in}_1$ then l^A is a typed action iff $A(x) = \mathbb{B}$. $\boldsymbol{\tau}^A$ is a typed action for an arbitrary A. If $l \neq \boldsymbol{\tau}$ and l^A is typed, the *type of* l^A is $A(\mathsf{sbj}(l))$. Then we say:

Definition 2. (affecting and enabled typed actions) l^A is *affecting* if $l \neq \boldsymbol{\tau}$ and the type of l^A is affecting; l^A is *non-affecting* if it is not affecting. Further l^A is *enabled* if $l = \boldsymbol{\tau}$ or the type of l^A is enabled.

Table 1 illustrates the classification of actions, writing $\boldsymbol{\tau}$, $\downarrow()$, $\uparrow()$, $\downarrow\&$, $\uparrow\oplus$, $\mathbf{!}$ and $\mathbf{?}$ for (respectively) the $\boldsymbol{\tau}$-action, unary linear input, unary linear output, linear branching, linear selection, replicated unary/branching input, and its dual.[3]

Table 1. Classification of Actions

	$\boldsymbol{\tau}$	$\downarrow()$	$\uparrow()$	$\downarrow\&$	$\uparrow\oplus$	$\mathbf{!}$	$\mathbf{?}$
affecting	no	no	yes	no	yes	yes	no
enabled	yes	yes	yes	yes	yes	no	yes

We can now introduce invisibility under the linear type structure which dictates the "$\boldsymbol{\tau}$-like" nature of certain non-$\boldsymbol{\tau}$-actions in the typed setting. Below and henceforth $\Delta, \Gamma, \ldots$ range over finite sets of names. $\mathsf{fn}(l)$ is the set of free names in l while $\mathsf{bn}(l)$ is the set of free names in l.

[3] We classify unary linear outputs $\uparrow()$ as affecting in Table 1 even though they are sometimes not, as is seen from Proposition 2. A simplest example is $()^{\uparrow}$.

Definition 3. 1. (invisible actions) A typed action l^A is *Δ-invisible* (Δ-i.) when either $\mathsf{fn}(l) \cap \Delta = \emptyset$ or, if not, l^A is an output which is non-affecting.[4] If l^A is Δ-invisible and is enabled, then l^A is *Δ-strongly invisible* (Δ-s.i.).

2. (abstracted transitions) $P^A \xrightarrow{\hat{l}}_\Delta Q^B$ when either: (1) $P^A \xrightarrow{l} Q^B$ or (2) $B = A$, $Q = P$ and l^A is Δ-invisible. $P^A \Longrightarrow_\Delta Q^B$ denotes $P^A \xrightarrow{l_1 \ldots l_n} Q^B$ ($n \geq 0$) where each l_i is strongly Δ-invisible; then $P^A \overset{l}{\Longrightarrow}_\Delta Q^B$ denotes $P^A \Longrightarrow_\Delta \xrightarrow{l}_\Delta \Longrightarrow_\Delta Q^B$; finally $P^A \overset{\hat{l}}{\Longrightarrow}_\Delta Q^B$ denotes either $P^A \overset{l}{\Longrightarrow}_\Delta Q^B$ or $P^A \Longrightarrow_\Delta Q^B$ where l is invisible and $\mathsf{fn}(B) \cap \mathsf{bn}(l) = \emptyset$. if $P^A \Longrightarrow_\Delta Q^B$ is induced by $P^A \xrightarrow{l_1 \ldots l_n} Q^B$, we say the latter *underlies* the former.

Note the standard abstracted transitions are a special case of those defined above.

3.2 Semi-typed Relation and Branching Closure

The invisibility of non-τ-actions necessitates one fundamental change in the notion of bisimulation. As an illustration we go back to the initial example in the introduction. The two typed processes concerned were $!x(c).\overline{c}^A$ and $!x(c).\overline{y}(e)e.\overline{c}^A$ with $A = x : (()^{\uparrow})^{!} \to y : (()^{\downarrow})^{?}$. After the common initial action, the typing becomes $A \otimes c : ()^{\uparrow}$. But if $\overline{y}(e)e.\overline{c}^A$ has an output action (which should and can be abstracted away), then e becomes free in the residual and appears in its type environment. This state should be related to the other process which still has type $A \otimes c : ()^{\uparrow}$. Consequently, a bisimulation needs to relate processes with distinct action types.

Definition 4. A relation $\mathcal{R}$ on typed processes is *semi-typed* when $P^A \mathcal{R} Q^B$ implies that the projections of A and B on $\mathsf{fn}(A) \cap \mathsf{fn}(B)$ coincide. We write $P^A \mathcal{R}^\Delta Q^B$ if $\mathcal{R}$ is semi-typed and $\mathsf{fn}(A) \cap \mathsf{fn}(B) = \Delta$, in which case we say *$P^A$ and Q^B are related by $\mathcal{R}$ at Δ*. The maximum typed subrelation of a semi-typed $\mathcal{R}$ is called its *centre*.

Using semi-typed relation, a natural way to define a bisimulation would be as follows: a semi-typed $\mathcal{R}$ such that, whenever $P_1^{A_1} \mathcal{R} P_2^{A_2}$ with $\Delta = \mathsf{fn}(A_1) \cap \mathsf{fn}(A_2)$, we have the following and its symmetric case:

whenever $P_1^{A_1} \xrightarrow{l} Q_1^{B_1}$, there is $P_2^{A_2} \overset{\hat{l}}{\Longrightarrow}_\Delta Q_2^{B_2}$ such that $Q_1^{B_1} \mathcal{R} Q_2^{B_2}$.

However the following shows that congruency is lost if we allow branching.

Example 3. $\overline{x}\mathtt{in}_1^{x:\mathbb{B}}$ and $\overline{y}\mathtt{in}_2^{y:\mathbb{B}}$ are bisimilar at $\emptyset$ in the above definition. Similarly $x[\&_{1,2}\overline{z}\mathtt{in}_i]$ and $y[\&_{1,2}\overline{z}\mathtt{in}_i]$ are bisimilar at z. However when we compose them in pairs, $(\overline{x}\mathtt{in}_1|x[\&_{1,2}\overline{z}\mathtt{in}_i])$ and $(\overline{y}\mathtt{in}_2|y[\&_{1,2}\overline{z}\mathtt{in}_i])$ are *not* bisimilar: in fact these terms can be regarded as, up to redundant reduction, $\overline{z}\mathtt{in}_1$ and $\overline{z}\mathtt{in}_2$, which would not be equated under any reasonable semantic criteria.

[4] We can consistently abstract away linear input at Δ. For simplicity and because this may not significantly change the resulting equality, we use the present definition.

The problem in this example is in the second equation: intuitively, $x[\&_{1,2}\overline{z}\mathtt{in}_i]$ and $y[\&_{1,2}\overline{z}\mathtt{in}_i]$ cannot be equated because, at disparate interfaces (here x and y), we should expect anything can happen: thus it is possible, at x, the first process receives the left selection, while, at y, the second process receives the right selection (which is precisely what happens in the composition). This indicates that we should say "for every possible branching at disparate channels, the behaviours of two processes at common channels coincide." This idea is formalised in the following definition. Below $t, t_i, \ldots$ range over sequences of typed transitions. A *branching variant* of, say, $x\mathtt{in}_1(\vec{y})$ is $x\mathtt{in}_2(\vec{z})$ (conforming to the given typing), taken up to α-equality.

Definition 5. (branching closure) A set $\{P^A \xrightarrow{t_i} Q_i^{B_i}\}_{i\in I}$ of sequences of typed transitions is *Δ-branching closed* (Δ-b.c.) iff: whenever $t_i = sls' \in S$ with l being a linear branching input such that $\mathsf{fn}(l) \cap \mathsf{fn}(\Delta) = \emptyset$, there is $t_j = sl's''$ $(j \in I)$ for each branching variant l' of l.

Accordingly we say $\{P^A \overset{\hat{l}}{\Longrightarrow}_\Delta Q_i^{B_i}\}_{i\in I}$ is Δ-branching closed if there exists a Δ-branching closed set $\{P^A \xrightarrow{t_i} Q_i^{B_i}\}_{i\in I}$ where $P^A \xrightarrow{t_i} Q_i^{B_i}$ underlies $P^A \overset{\hat{l}}{\Longrightarrow}_\Delta Q_i^{B_i}$ for each i. Similarly for other forms of abstracted transitions.

3.3 Linear Bisimulation

We can now introduce a bisimilarity on linear processes.

Definition 6. (linear bisimulation) A semi-typed $\mathcal{R}$ is a *linear bisimulation* when $P_1^{A_1} \mathcal{R} P_2^{A_2}$ with $\Delta = \mathsf{fn}(A_1) \cap \mathsf{fn}(A_2)$ implies the following and its symmetric case: whenever $P_1^{A_1} \xrightarrow{l} Q_1^{B_1}$, there is a Δ-closed $\{P_2^{A_2} \overset{\hat{l}}{\Longrightarrow}_\Delta Q_{2i}^{B_{2i}}\}_{i\in I}$ such that $Q_1^{B_1} \mathcal{R} Q_{2i}^{B_{2i}}$ for all $i \in I$. The maximum linear bisimulation exists, denoted $\approx_{\mathrm{L}}$.

Simple examples of (non-)bisimilarity follow. Below and henceforth we omit obvious type annotations, assuming all processes are well-typed. We often annotate $\approx_{\mathrm{L}}$ as $\approx_{\mathrm{L}}^{x,y}$ (which follows Definition 4) to make intersecting channels explicit.

Example 4. 1. $x.\mathbf{0} \approx_{\mathrm{L}}^{\emptyset} \mathbf{0}$ and $!x.\mathbf{0} \approx_{\mathrm{L}}^{\emptyset} \mathbf{0}$ and $\overline{x}|\overline{x} \approx_{\mathrm{L}}^{x} \overline{x} \approx_{\mathrm{L}}^{x} \mathbf{0}$.
2. $x.\overline{y}\mathtt{in}_1 \approx_{\mathrm{L}}^{y} \overline{y}\mathtt{in}_1$. Intuitively this is because an output at x will surely arrive in which case the former process has the same observable as the latter.
3. Because of the lack of branching closure, we have $x[\&_{1,2}\overline{z}\mathtt{in}_i] \not\approx_{\mathrm{L}}^{z} y[\&_{1,2}\overline{z}\mathtt{in}_i]$. On the other hand, we have $x[\&_{1,2}\overline{z}\mathtt{in}_1] \approx_{\mathrm{L}}^{z} y[\&_{1,2}\overline{z}\mathtt{in}_1] \approx_{\mathrm{L}}^{z} \overline{z}\mathtt{in}_1$.

We prove the following result in the next section.

Theorem 1. *The centre of* $\approx_{\mathrm{L}}$ *is a congruence.*

Since an action of $\mathbb{B}$-type is always visible, we immediately obtain:

Corollary 1. *The centre of* $\approx_{\mathrm{L}}$ *is a subrelation of* $\cong_\pi$.

We give simple applications of the linear bisimilarity. Below in (1) $?A$ in the condition indicates $\mathsf{md}(A) = ?$. (1) says processes which are entirely typed with $?$-types (which in particular includes $\mathbf{0}$) are mutually equated with each other even if they may be engaged in arbitrarily long interactions with the environments. (2) says there is essentially a unique inhabitant in the translation of the unit type of $\lambda_{()\times+}$. (3) uses Theorem 1 and (2), combined with Theorem 5.9 (full abstraction) in [35], to derive the equality in $\lambda_{()\times+}$.

Proposition 3. *1.* (innocuous actions [20]) *If* $\vdash P_{1,2} \triangleright ?A$ *then* $P_1 \approx_{\text{L}}^{\mathsf{fn}(A)} P_2$.
2. (unit inhabitation, 1) $\vdash P \triangleright A$ *with* $A = x : (()^{\uparrow})^{!} \rightarrow A_0$ *implies* $P \approx_{\text{L}}^{x} !x(c).\overline{c}$.
3. (unit inhabitation, 2) *If* $\Gamma \vdash M$: $\mathtt{unit}$ *in* $\lambda_{()\times+}$ *then* $\Gamma \vdash M \cong \star : \mathtt{unit}$ *where* $\cong$ *is the standard contextual equivalence in* $\lambda_{()\times+}$.

4 Congruency of Linear Bisimilarity

The purpose of this section is to briefly illustrate a proof method for congruency of $\approx_{\text{L}}$. A central difficulty of the proof lies in the existence of strong invisibility when we prove a closure of parallel compositions. This is overcome by the analysis of the operational structures ensured by linear typing, which involve liveness.

Suppose we wish to prove the relation $\mathcal{R} \stackrel{\text{def}}{=} \{(P_1|Q_1,\ P_2|Q_2) \mid P_1 \approx P_2,\ Q_1 \approx Q_2\}$ to be a bisimulation in order to show that $\approx$ is closed under $|$. Assume $P_1 \mid Q_1 \xrightarrow{l} P_1' \mid Q_1$; then by assumption, there exists $P_2 \stackrel{\hat{l}}{\Longrightarrow} P_2' \approx P_1'$, hence in the standard proof, we easily have: $P_2 \mid Q_2 \stackrel{\hat{l}}{\Longrightarrow} P_2' \mid Q_2$, and $P_1' \mid Q_1 \ \mathcal{R}\ P_2' \mid Q_2$. However, due to strong invisibility, the same reasoning does not work for $\approx_{\text{L}}$ even in the above trivial case. Recall the example in the Introduction, $P_1 \stackrel{\text{def}}{=} !u(x).\overline{x}\mathtt{in}_1$ and $P_2 \stackrel{\text{def}}{=} !u(x).\overline{y}(e)e.\overline{x}\mathtt{in}_1$. Then we know $P_1 \approx_{\text{L}} P_2$ because $\overline{y}(e)$ and e are both invisible, so we have $P_1 \xrightarrow{u(x)}_{uy} P_1' \stackrel{\text{def}}{=} \overline{x}\mathtt{in}_1 \mid P_1$ and $P_2 \stackrel{u(x)}{\Longrightarrow}_{uy} P_2' \stackrel{\text{def}}{=} \overline{x}\mathtt{in}_1 \mid P_2$ with P_1' and P_2' bisimilar. Suppose we compose them with $Q \stackrel{\text{def}}{=} !y(e).Q_0$ for some Q_0 such that $P_1 \mid Q$ and $P_2 \mid Q$ are typable. Then we have $P_1 \mid Q \xrightarrow{u(x)}_{uy} P_1' \mid Q$, while we cannot have $P_2 \mid Q \stackrel{u(x)}{\Longrightarrow}_{uy} P_2' \mid Q$, because the only possible transition is $P_2 \mid Q \xrightarrow{u(x)} \xrightarrow{\tau} (\boldsymbol{\nu}\, e)(e.\overline{x}\mathtt{in}_1 \mid Q_0) \mid P_2 \mid Q$. In order to achieve $P_2' \stackrel{\text{def}}{=} \overline{x}\mathtt{in}_1 \mid P_2$ from this process, $e.\overline{x}\mathtt{in}_1$ needs an acknowledgement $\overline{e}$ from Q_0. At this point, however, we can use a liveness property which extends Proposition 1 (2): if Q_0 has a linear output type at e, then there *always* exist a finite sequence of strong invisible transitions to emit e such that $Q_0 \stackrel{\overline{e}}{\Longrightarrow}_{uy} Q_0'$ and $Q \approx_{\text{L}} Q_0' \mid Q$.

In the following we define such a chain, called *call-sequence*. Let us assume $P \stackrel{l_1 \cdot l_2}{\Longrightarrow} Q$. We write: $l_1 \curvearrowright_{\mathsf{b}} l_2$ (l_1 *binds* l_2) when the subject of l_2 is bound by l_1 (e.g. $x(y) \curvearrowright_{\mathsf{b}} \overline{y}$) and $l_1 \curvearrowright_{\mathsf{p}} l_2$ (l_1 *prefixes* l_2) when the action l_2 is input-prefixed by l_1 (e.g. $x(y) \curvearrowright_{\mathsf{p}} \overline{z}$ in $x(y).\overline{z}$). Define $\curvearrowright = \curvearrowright_{\mathsf{b}} \cup \curvearrowright_{\mathsf{p}}$. We write $\boldsymbol{\tau} \curvearrowright l_2$ if $P \stackrel{l_2}{\Longrightarrow} Q$ and P has subterms Q_1 and Q_2 such that $Q_1 \stackrel{l}{\Longrightarrow} Q_1'$ and $Q_2 \stackrel{\overline{l} \cdot l_2}{\Longrightarrow} Q_2'$ with $\overline{l} \curvearrowright l_2$; similarly we define $l_1 \curvearrowright \boldsymbol{\tau}$; we extend this to a chain $l_1 \curvearrowright \boldsymbol{\tau}^* \curvearrowright l_2$ and denote it

$l_1 \curvearrowright^+ l_2$ ([6, Appendix F] gives a detailed definition using occurrences of terms). Then a *call-sequence (c.s.) to l under A* is a sequence of actions which has the following shape.

$$(l_0 \curvearrowright^+)\ l_1 \curvearrowright_{\mathsf{b}} l_2 \curvearrowright^+ l_3 \curvearrowright_{\mathsf{b}} l_4 \curvearrowright^+ \cdots \curvearrowright^+ l_{2n-1} \curvearrowright_{\mathsf{b}} l_{2n} \curvearrowright_{\mathsf{p}} l$$

where $\mathsf{md}(l_{2k-1}^A) = \mathbf{?}$ and $\mathsf{md}(l_{2k}^A) = \downarrow$.

Lemma 1. *1.* (shortest c.s.) *Suppose $\vdash P \triangleright A$ and $P \stackrel{l}{\Longrightarrow}_{\Gamma}$ with l output. Then there is a shortest c.s. $l_1 \curvearrowright \cdots \curvearrowright l_n$ to l under A such that $P \xrightarrow{l_1 \cdots l_n} \xrightarrow{l}$.*

2. (extended liveness) *If $\vdash P \triangleright A \otimes e : \tau$ with $\mathsf{md}(\tau) = \uparrow$, then $P \Longrightarrow_{A\otimes e:\tau} \xrightarrow{l}$ with $\mathsf{sbj}(l) = e$.*

Note (2) does not restrict the shape of A, cf. Proposition 1 (2). Together with (1), we know there is always a shortest strongly invisible call sequence to each linear output.

We now turn to the congruency of $\approx_{\text{L}}$. First, reflexivity and symmetry of $\approx_{\text{L}}$ are immediate by definition; it also satisfies transitivity on the centre ($P_1^{A_1} \approx_{\text{L}}^{\Gamma} P_2^{A_2}$ and $P_2^{A_2} \approx_{\text{L}}^{\Delta} P_3^{A_3}$ with $\mathsf{fn}(A_1) \cap \mathsf{fn}(A_3) = \Gamma \cap \Delta$ imply $P_1^{A_1} \approx_{\text{L}}^{\Gamma\cap\Delta} P_3^{A_3}$). Hence $\approx_{\text{L}}$ is an equivalence. For compatibility, we use the following characterisation of $\approx_{\text{L}}$, which reduces the conditions needed for a bisimulation closure. The form of the resulting relation is similar to the branching bisimulation studied in (untyped) confluent processes [28].

Lemma 2. (context lemma) *Suppose $\mathcal{R}$ is semi-typed such that $P_1^{A_1} \mathcal{R} P_2^{A_2}$ with $\Delta = \mathsf{fn}(A_1) \cap \mathsf{fn}(A_2)$ implies the following and its symmetric case:*

- *whenever $P_1^{A_1} \xrightarrow{l} Q_1^{B_1}$ where l is Δ-invisible and $\mathsf{fn}(l) \cap \Delta = \emptyset$ then $Q_1^{B_1} \mathcal{R} P_2^{A_2}$.*
- *whenever $P_1^{A_1} \xrightarrow{l} Q_1^{B_1}$ with l input such that $\mathsf{sbj}(l) \in \Delta$, then there is $P_2^{A_2} \xrightarrow{l}_{\Delta} Q_2^{B_2}$ such that $Q_1^{B_1} \mathcal{R} Q_2^{B_2}$.*
- *whenever $P_1^{A_1} \xrightarrow{l} Q_1^{B_1}$ with l Δ-visible linear output such that $\mathsf{sbj}(l) \in \Delta$, then there is a Δ-closed call sequence to l $\{P_2^{A_2} \Longrightarrow_{\Delta} \xrightarrow{l} Q_{2i}^{B_{2i}}\}_{i\in I}$ such that $Q_1^{B_1} \mathcal{R} Q_{2i}^{B_{2i}}$ for all $i \in I$.*
- *whenever $P_1^{A_1} \xrightarrow{l} Q_1^{B_1}$ with $\mathsf{md}(l^{A_1}) = \mathbf{?}$ and $\mathsf{sbj}(l) \in \Delta$, there is a Δ-closed call sequence to l, $\{P_2^{A_2} \Longrightarrow_{\Delta} Q_{2i}^{B_{2i}}\}_{i\in I}$, such that either $Q_1^{B_1} \mathcal{R} Q_{2i}^{B_{2i}}$ or $Q_{2i}^{B_{2i}} \xrightarrow{l} {Q'}_{2i}^{B'_{2i}}$ such that $Q_1^{B_1} \mathcal{R} {Q'}_{2i}^{B'_{2i}}$.*

Then the maximum such relation, denoted by $\stackrel{\bullet}{\approx}_{\text{L}}$, coincides with $\approx_{\text{L}}$.

Using the characterisation, the closure under prefixes and restriction is easy. For parallel composition, using extended liveness repeatedly, we can prove if whenever $P \xrightarrow{l}_{\Delta} P'$ and $P\,|\,Q$ is typable, there exists a Δ-b.c. $\{P\,|\,Q \Longrightarrow_{\Delta} \xrightarrow{l}_{\Delta} (\boldsymbol{\nu}\,\vec{v}_i)(P_i''|Q_i'')\}_{i\in I}$ such that $P_i'' \approx P'$ and $Q_i'' \approx Q$. Using this and Lemma 2, we can prove the closure under the parallel composition on the centre ($P_1^A \approx_{\text{L}} P_2^A$ and $Q_1^B \approx_{\text{L}} Q_2^B$ with $A \asymp B$ imply $P_1|Q_1^{A\odot B} \approx_{\text{L}} P_2|Q_2^{A\odot B}$). Thus the centre of $\approx_{\text{L}}$ is a congruence. Finally $\equiv \subset \approx_{\text{L}}$ is proved easily by using weakening and strengthening of base A of $\approx_{\text{L}}^A$. Since the observability predicate given in § 2.3 is easily satisfied by $\approx_{\text{L}}$, we conclude that $\approx_{\text{L}} \subseteq \cong_{\pi}$.

5 Applications to Secrecy

In linear bisimulations, we abstract away non-affecting typed actions as if they were τ-actions. If we assign a secrecy level to each channel and stipulate a level of observation, then we can further abstract away those actions which should not be visible from the stipulated level. For example, from a low-level viewpoint, actions at high-level channels should be invisible. The technical development of this secrecy enhancement closely follows that of the linear bisimilarity, and offers a powerful tool for reasoning about secrecy in processes.

Assume given a complete lattice of *secrecy levels* $(s, s', \ldots)$ with the ordering $\sqsubseteq$. $\top$ (the most secret) and $\bot$ (the most public) denote the top and bottom of the lattice, respectively. Channel types are annotated with these levels:

$$\tau \ ::= \ \tau_{\mathtt{I}} \mid \tau_{\mathtt{O}} \mid *_s \qquad \tau_{\mathtt{I}} \ ::= \ (\vec{\tau})^{p_{\mathrm{I}}}_s \mid [\&_{i\in I}\vec{\tau}_i]^{p_{\mathrm{I}}}_s \qquad \tau_{\mathtt{O}} \ ::= \ (\vec{\tau})^{p_{\mathrm{O}}}_s \mid [\oplus_{i\in I}\vec{\tau}_i]^{p_{\mathrm{O}}}_s$$

The same sequential constraints (cf. §2.1) apply to channel types. In $\overline{\tau}$, we require each dualised occurrence to own identical secrecy levels. Action types are given precisely as before, using secrecy annotated types. Then we set:

1. l^A is *s-affecting* if it is affecting in the preceding sense and, if l is a linear selection, then $\mathsf{sec}(A(x)) \sqsubseteq s$ ($\mathsf{sec}(\tau)$ is the outermost secrecy level of τ).
2. l^A is *s-Δ-invisible* when either $\mathsf{fn}(l)\cap\Delta = \emptyset$ or, if not, l^A is an output which is not s-affecting. If l^A is s-Δ-invisible and, moreover, is enabled, then l^A is s-Δ *strongly invisible*. The abstracted transitions $P^A \Longrightarrow_{\Delta,s} Q^B$, $P^A \stackrel{l}{\Longrightarrow}_{\Delta,s} Q^B$ and $P^A \stackrel{\hat{l}}{\Longrightarrow}_{\Delta,s} Q^B$ are defined accordingly.

In (1), we only count linear selections among secrecy-sensitive observable actions since in the linear type structure no other typed actions directly emit information (note, in Proposition 2, a type is affecting only when it is or contains linear selection). We can now introduce a secrecy-sensitive bisimilarity.

Definition 7. (s-bisimulation) A semi-typed relation $\mathcal{R}$ is a *s-bisimulation* when $P_1^{A_1}\,\mathcal{R}\,P_2^{A_2}$ with $\Delta = \mathsf{fn}(A_1)\cap\mathsf{fn}(A_2)$ implies the following and its symmetric case: whenever $P_1^{A_1} \xrightarrow{l} Q_1^{B_1}$, there is a Δ-closed $\{P_2^{A_2} \stackrel{\hat{l}}{\Longrightarrow}_{\Delta,s} Q_{2i}^{B_{2i}}\}$ such that $Q_1^{B_1}\,\mathcal{R}\,Q_{2i}^{B_{2i}}$. The maximum s-bisimulation exists for each s, which we write $\approx_s$.

By definition, $P^A \approx_{\mathrm{L}} Q^B$ implies $P^A \approx_s Q^B$ for any s. Further $P^A \approx_\top Q^B$ implies $P^A \approx_{\mathrm{L}} Q^B$. A simple example of s-bisimilarity:

Example 5. (s-bisimilarity) $\vdash \overline{x}\mathtt{in}_1 \triangleright x : \mathbb{B}_\top \not\approx_\top \vdash \overline{x}\mathtt{in}_2 \triangleright x : \mathbb{B}_\top$ but we have $\vdash \overline{x}\mathtt{in}_1 \triangleright x : \mathbb{B}_\top \approx_\bot \vdash \overline{x}\mathtt{in}_2 \triangleright x : \mathbb{B}_\top$.

A basic observation on $\approx_s$ is that it alone does not form a coherent notion of process equivalence.

Fact 1. *Suppose $\vdash P \triangleright A$ is derived as Section 2 using secrecy annotated types. Then the centre of $\approx_s$ is not closed under parallel composition.*

Proof. Take $\overline{x}\mathtt{in}_i^{x:\tau_1}$ $(i = 1, 2)$ with $\tau_1 = \mathbb{B}_\top$. Then $\overline{x}\mathtt{in}_1 \approx_\bot \overline{x}\mathtt{in}_2$. However if we compose these processes with $x[.\overline{u}\mathtt{in}_1\&.\overline{u}\mathtt{in}_2]^{x:\tau_2 \to u:\tau_3}$ where $\tau_2 = \overline{\tau_1}$ and $\tau_3 = \mathbb{B}_\bot$, then $(\boldsymbol{\nu}\, x)(P_1|Q)^{u:\tau_3} \not\approx_\bot (\boldsymbol{\nu}\, x)(P_2|Q)^{u:\tau_3}$. ∎

The example in the proof above suggests that, for regaining compositionality in $\approx_s$, we need to restrict the set of processes to those which do not transfer information at some high-level to lower levels. In other words, we require information flow in processes to be *secure* [8]. Below we say l^A is *receiving at* s if l^A is a linear branching and moreover $\mathsf{sec}(A(\mathsf{sbj}(l))) = s$.

Definition 8. (behavioural secrecy) A set of typed processes $\mathcal{S}$ is a *secrecy witness* if the following holds: whenever $P^A \in \mathcal{S}$ and $P^A \xrightarrow{l} Q^B$, we have (1) $Q^B \in \mathcal{S}$ and (2) if l^A is receiving at s then $P^A \approx_{s'} Q^B$ for each s' such that $s \not\sqsubseteq s'$. P^A is *behaviourally secure* iff P^A is in some secrecy witness.

Only linear branching counts as "receiving", which is an exact dual of $\approx_s$ (where we consider abstraction by secrecy levels only for linear selection). Intuitively, a process is behaviourally secure iff, whenever it receives non-trivial information at some level, it behaves, to a lower-level observer, as if the action had not taken place. Some examples of (non-)secure processes follow.

Example 6. $\mathbf{0}^\emptyset$ is secure. If P^A is secure and $(\boldsymbol{\nu}\, x)P^{A/x}$ is well-typed, the latter is secure. If $P^{\vec{y}:\vec{\tau}\otimes ?A}$ is secure and $!x(\vec{y}).P^{x:(\vec{\tau})^! \to A}$ is well-typed, the latter is secure. Finally, given $A \stackrel{\text{def}}{=} x : \mathbb{B}_\top \to y : \overline{\mathbb{B}}_\bot$, $x[.\overline{y}\mathtt{in}_1\&.\overline{y}\mathtt{in}_2]^A$ is not secure but $x[.\overline{y}\mathtt{in}_1\&.\overline{y}\mathtt{in}_1]^A$ is secure (the latter is because $x[.\overline{y}\mathtt{in}_1\&.\overline{y}\mathtt{in}_1]^A \approx_\bot^y \overline{y}\mathtt{in}_1^{A/x}$).

The following is proved precisely as Theorem 1 except that the use of s-invisibility is compensated by behavioural secrecy.

Proposition 4. *The centre of $\approx_s$ over behaviourally secure processes is compatible with all operators except linear branching.*

Without using a syntactic type discipline for secrecy, Proposition 4 offers a framework for fully compositional reasoning for secure processes (we can further close $\approx_s$ under linear branching using the following condition: $x[\&_i(\vec{y}_i).P_i]$ is secure with x given level s if $P_i \approx_{s'} P_j$ for any s' such that $s \not\sqsubseteq s'$).

To investigate the relationship with the present theory of secrecy and the type-based approach in [23], we introduce $\mathsf{tamp}(A)$ (the lowest possible effect level of A), a type discipline for secrecy $\vdash_{\mathsf{sec}} P \triangleright A$ (which we read: P is securely typed by A) and $\cong_s$ (a secrecy-sensitive contextual congruence), all from [23].

Definition 9. (tamper level and secure typing [23]) The *tamper level* of τ, denoted $\mathsf{tamp}(\tau)$, is defined as follows. (1) $\mathsf{tamp}(\tau) = \top$ if τ is not affecting; (2) $\mathsf{tamp}((\vec{\tau})_s^\uparrow) = \sqcap_i \mathsf{tamp}(\tau_i)$, $\mathsf{tamp}([\oplus_i \vec{\tau}_i]_s^\uparrow) \stackrel{\text{def}}{=} s$, $\mathsf{tamp}((\vec{\tau})_s^!) \stackrel{\text{def}}{=} \sqcap_i \mathsf{tamp}(\tau_i)$, $\mathsf{tamp}([\&_i \vec{\tau}_i]_s^!) \stackrel{\text{def}}{=} \sqcap_{ij} \mathsf{tamp}(\tau_{ij})$. Then $\mathsf{tamp}(A) \stackrel{\text{def}}{=} \sqcap\{\mathsf{tamp}(A(x)) \mid x \in \mathsf{fn}(A)\}$. Sequents of the security typing have the form $\vdash_{\mathsf{sec}} P \triangleright A$, whose rules are left to Appendix A. If $\vdash_{\mathsf{sec}} P \triangleright A$, we say P *is securely typable with* A.

Note that $\mathsf{tamp}(\tau) = \top$ whenever τ is not affecting.

Definition 10. (secrecy-sensitive contextual equality) For each s, *s-contextual congruence* $\cong_s$ is defined as the maximum typed congruence satisfying the following condition: if $P \Downarrow_x^i$ and $P \cong_\pi^{x:\mathbb{B}_{s'}} Q$ with $s' \sqsubseteq s$, then $Q \Downarrow_x^i$ (i=1,2).

Proposition 5. *1. If $\vdash_{\mathsf{sec}} P_{1,2} \triangleright A$ and $\mathsf{tamp}(A) \not\sqsubseteq s$ then $P_1^A \approx_s P_2^A$.*
2. If $\vdash_{\mathsf{sec}} P \triangleright A$, then P is behaviourally secure.
3. $\approx_s$ is congruent over securely typed processes, i.e. it is an equivalence closed under typed contexts given by the secure typing $\vdash_{\mathsf{sec}}$ in Definition 9.

Note that, in (2), the other direction does not hold (for the proof, we can use the last process in Example 6). This proposition allows us to consistently integrate the secrecy typing of [20, 23] with the present behavioural theory, for the purpose of secrecy analysis in processes and, via embeddings, in programs. For example, given a $\lambda_{()\times+}$-term MN, we can check the secrecy of $[\![M]\!]_m$ by typing, $[\![N]\!]_n$ by behavioural secrecy, and finally verify their combination using typing. Another consequence of Proposition 5 is a simple proof of the following noninterference result, first given in [23].

Corollary 2. (noninterference) *Let $\vdash_{\mathsf{sec}} P_{1,2} \triangleright A$ and $\mathsf{tamp}(A) \not\sqsubseteq s$. Then we have $\vdash P_1 \cong_s P_2 \triangleright A$.*

We conclude our technical development with a simple example of reasoning about secrecy, guaranteeing the λ-term mentioned in the introduction is indeed secure.

Example 7. (secrecy via encoding) Let $M \stackrel{\text{def}}{=} \mathtt{case}\, y^\top\ \mathtt{of}\ \{\mathtt{in}_i() : \mathtt{in}_1(\star)\}_{i\in\{1,2\}}$. We show $[\![M : \mathtt{bool}_\top]\!]_u \stackrel{\text{def}}{=} !u(c).\overline{y}(e)e[.\overline{c}\mathtt{in}_1 \& .\overline{c}\mathtt{in}_1] \triangleright u : (\mathbb{B}_\bot)^! \rightarrow y : (\overline{\mathbb{B}_\top})^?$ is secure. By Proposition 4, it suffices to show $e[.\overline{c}\mathtt{in}_1 \& .\overline{c}\mathtt{in}_1] \triangleright e : \overline{\mathbb{B}}_\top \rightarrow c : \mathbb{B}_\bot$ is secure. But this has already been shown in Example 6, hence done.

Extensions to Other Type Structures. We have presented a theory of behavioural secrecy focussing on the pure linear π-calculus. First, the same results can be obtained for the free name passing linear calculus via a fully abstract encoding [17]. The framework is also systematically extendible to other type structures which integrate linearity with affinity (nontermination) [6], statefulness (references) [23] and nondeterminism [20]. In each case, the only necessary extensions are (1) the incorporation of a new s-affecting action into s-bisimilarity and its dual receiving action into behavioural secrecy, and (2) when affinity is in the type structure, we change Definition 1 as follows: $\mathbb{B}$ becomes $()^{\uparrow_A}$ ($\uparrow_A$ indicates possibly diverging, or affine, output), and the condition "$C[P_1] \Downarrow_u^1$ and $C[P_2] \Downarrow_u^2$" becomes "$C[P_i] \Downarrow_x$ and $C[P_j] \Uparrow_x$ with $i \neq j$" (here $\Downarrow_x$ iff $P \longrightarrow^* \overline{x}|P'$ for some P', and $\Uparrow_x$ iff not $\Downarrow_x$). Except for these two changes, Definitions 1–8 can be used as they are. Because we have the same liveness property for call sequences (Lemma 1 (2)) in each extension, the same proof methods can be used to obtain the corresponding results such as Theorem 1 and Proposition 4. Together with full abstraction, the framework offers a uniform basis for behavioural analysis of secrecy in programming languages.

References

1. Abadi, M., Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.
2. Abadi, M., Secrecy in programming-language semantics, *MFPS XV*, ENTCS, 20 (April 1999).
3. Abadi, M., Banerjee, A., Heintze, N. and Riecke, J., A core calculus of dependency, *POPL'99*, ACM, 1999.
4. Abramsky, S., Jagadeesan, R. and Malacaria, P., Full Abstraction for PCF. *Info. & Comp.* 163 (2000), 409–470.
5. Boreale, M. and Sangiorgi, D. Bisimulation in name-passing calculi without matching. *LICS'98*, IEEE, 1998.
6. Berger, M., Honda, K. and Yoshida, N., Sequentiality and the π-Calculus, *TLCA'01*, LNCS 2044, 29–45, Springer, 2001. The full version available at www.dcs.qmw.ac.uk/~kohei.
7. Boudol, G. and Castellani, I., Noninterference for Concurrent Programs, *ICALP'01*, LNCS, Springer, 2001.
8. Denning, D. and Denning, P., Certification of programs for secure information flow. *Communication of ACM*, ACM, 20:504–513, 1997.
9. Focardi, R. and Gorrieri, R., The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering,* 23(9), 1997.
10. Focardi, R., Gorrieri, R. and Martinelli, F., Non-interference for the analysis of cryptographic protocols. *ICALP'00*, LNCS 1853, Springer, 2000.
11. Gay, S. and Hole, M., Types and Subtypes for Client-Server Interactions, *ESOP'99*, LNCS 1576, 74–90, Springer, 1999.
12. Girard, J.-Y., Linear Logic, *TCS*, 50, 1–102, 1987.
13. Hennessy, M. and Rathke, J., Typed behavioural equivalences for processes in the presence of subtyping, To appear in Proc. *CATS 2002*.
14. Hennessy, M. and Riely, J., Information flow vs resource access in the asynchronous pi-calculus, *ICALP'00*, LNCS 1853, 415–427, Springer, 2000.
15. Honda, K., Types for Dyadic Interaction. *CONCUR'93*, LNCS 715, 509–523, 1993.
16. Honda, K., Composing Processes, *POPL'96*, 344–357, ACM, 1996.
17. Honda, K., Notes on Linear Typing for Free Outputs, May, 2001.
18. Honda, K., Kubo, M. and Vasconcelos, V., Language Primitives and Type Discipline for Structured Communication-Based Programming. *ESOP'98*, LNCS 1381, 122–138. Springer-Verlag, 1998.
19. Honda, K. and Tokoro, M. An object calculus for asynchronous communication. *ECOOP'91*, LNCS 512, 133–147, 1991.
20. Honda, K., Vasconcelos, V. and Yoshida, N., Secure Information Flow as Typed Process Behaviour, *ESOP'00*, LNCS 1782, 180–199, 2000.
21. Honda, K. and Yoshida, N. On Reduction-Based Process Semantics. *TCS*, 151, 437–486, 1995.
22. Honda, K. and Yoshida, N., Game-theoretic analysis of call-by-value computation. *TCS*, 221 (1999), 393–456, 1999.
23. Honda, K. and Yoshida, N., A uniform type structure for secure information flow, To appear in *POPL'02*, ACM, 2002.
24. Hyland, M. and Ong, L., "On Full Abstraction for PCF": I, II and III. *Info. & Comp.* 163 (2000), 285–408, 2000.

25. Kobayashi, N., Pierce, B., and Turner, D., Linearity and the π-calculus, *POPL'96*, 358–371, 1996.
26. Milner, R., Functions as Processes. *MSCS*, 2(2), 119–146, CUP, 1992.
27. Milner, R. and Sangiorgi, D., Barbed Bisimulation, *ICALP'92*, LNCS 623, 685–695, Springer, 1992.
28. Philippou, A. and Walker, D., On confluence in the π-Calculus, *ICALP'97*, LNCS 1256, 314–324, Springer, 1997.
29. Sangiorgi, D. π-calculus, internal mobility, and agent-passing calculi. *TCS*, 167(2):235–271, 1996.
30. Sangiorgi, D., The name discipline of uniform receptiveness, *TCS*, 221, 457–493, 1999.
31. Sabelfield, A. and Sands, D. A per model of secure information flow in sequential programs. *ESOP'99*, LNCS 1576, Springer, 1999.
32. Smith, G. and Volpano, D., Secure information flow in a multi-threaded imperative language, 355–364, *POPL'98*, ACM, 1998.
33. Vasconcelos, V., Typed concurrent objects. *ECOOP'94*, LNCS 821, 100–117. Springer, 1994.
34. Yoshida, N. Graph Types for Mobile Processes. *FST/TCS'16*, LNCS 1180, 371–386, Springer, 1996.
35. Yoshida, N., Berger, M. and Honda, K., Strong Normalisation in the π-Calculus, *LICS'01*, IEEE, 2001. The full version as MCS technical report, 2001-09, University of Leicester, 2001. Available at www.mcs.le.ac.uk/˜nyoshida/paper.html.
36. A full version of this paper. MCS technical report, 2001-48, University of Leicester, 2001. Available at www.mcs.le.ac.uk/˜nyoshida/paper.html.

A Typing Rules

In the following rules, $A\langle\vec{y}:\vec{\tau}\rangle$ indicates each $y_i : \tau_i$ occurs in A. $\otimes$ is disjoint union. $x:\tau \to A$ adds new edges from $x:\tau$ to the non-suppressed nodes in A. $A^{\text{-}x}$ indicates $x \notin \mathsf{fn}(A)$. pA indicates $p \in \mathsf{md}(A)$. We also assume $\uparrow A$ in $(\mathsf{In}^{\downarrow})$ and $(\mathsf{Bra}^{\downarrow})$ is either a singleton or empty ("unique-answer-per-thread").

$$(\mathsf{Zero})\ \frac{-}{\vdash \mathbf{0} \triangleright \_} \qquad (\mathsf{Par})\ \frac{\vdash P_i \triangleright A_i \quad (i=1,2) \qquad A_1 \asymp A_2}{\vdash P_1|P_2 \triangleright A_1 \odot A_2} \qquad (\mathsf{Res})\ \frac{\vdash P \triangleright A\langle x:\tau\rangle \qquad \mathsf{md}(\tau) \in \{*, \mathbf{!}\}}{\vdash (\boldsymbol{\nu}\, x)P \triangleright A/x} \qquad (\mathsf{Weak}\text{-}*,\mathbf{?})\ \frac{\vdash P \triangleright A^{\text{-}x} \qquad \mathsf{md}(\tau) \in \{*, \mathbf{?}\}}{\vdash P \triangleright A \otimes x:\tau}$$

$$(\mathsf{In}^{\downarrow})\ \frac{\vdash P \triangleright \vec{y}:\vec{\tau} \otimes \uparrow A^{\text{-}x} \otimes \mathbf{?} B^{\text{-}x}}{\vdash x(\vec{y}).P \triangleright (x:(\vec{\tau})^{\downarrow} \to A) \otimes B} \qquad (\mathsf{In}^{\mathbf{!}})\ \frac{\vdash P \triangleright \vec{y}:\vec{\tau} \otimes \mathbf{?} A^{\text{-}x}}{\vdash !\, x(\vec{y}).P \triangleright x:(\vec{\tau})^{\mathbf{!}} \to A} \qquad (\mathsf{Out})\ (p \in \{\uparrow, \mathbf{?}\})\ \frac{\vdash P \triangleright C\langle\vec{y}:\vec{\tau}\rangle \qquad C/\vec{y} = A \asymp x:(\vec{\tau})^p}{\vdash \overline{x}(\vec{y})P \triangleright A \odot x:(\vec{\tau})^p}$$

$$(\mathsf{Bra}^{\downarrow})\ \frac{\vdash P_i \triangleright \vec{y}_i:\vec{\tau}_i \otimes \uparrow A^{\text{-}x} \otimes \mathbf{?} B^{\text{-}x}}{\vdash x[\&_i(\vec{y}_i).P_i] \triangleright (x:[\&_i\vec{\tau}_i]^{\downarrow} \to A) \otimes B} \qquad (\mathsf{Bra}^{\mathbf{!}})\ \frac{\vdash P_i \triangleright \vec{y}_i:\vec{\tau}_i \otimes \mathbf{?} A^{\text{-}x}}{\vdash !\, x[\&(\vec{y}_i).P_i] \triangleright x:[\&_i\vec{\tau}_i]^{\mathbf{!}} \to A} \qquad (\mathsf{Sel})\ (p \in \{\uparrow, \mathbf{?}\})\ \frac{\vdash P \triangleright C\langle\vec{y}:\vec{\tau}_j\rangle \qquad C/\vec{y} = A \asymp x:[\oplus_i\vec{\tau}_i]^p}{\vdash \overline{x}\mathtt{in}_j(\vec{y})P \triangleright A \odot x:[\oplus_i\vec{\tau}_i]^p}$$

In Section 5, we consider a secrecy enhancement of typed terms. This is done in two stages. First we simply annotate channels with secrecy levels, leaving the rules themselves precisely as above except types are now annotated by secrecy levels (which the rules simply ignore). Second we consider *securely typed processes*, which are the set of sequents of form $\vdash_{\mathsf{sec}} P \triangleright A$ which are derived from the above rules except the rule $(\mathsf{Bra}^{\downarrow})$ is replaced by the following one, using the channel types annotated by secrecy levels and using the sequent $\vdash_{\mathsf{sec}} P \triangleright A$ instead of $\vdash P \triangleright A$.

$$(\mathsf{Bra}^{\downarrow}) \quad \frac{\vdash_{\mathsf{sec}} P_i \triangleright \vec{y}_i : \vec{\tau}_i \otimes \uparrow A^{\text{-}x} \otimes \mathbf{?} B^{\text{-}x} \qquad s \sqsubseteq \mathsf{tamp}(A)}{\vdash_{\mathsf{sec}} x[\&_i(\vec{y}_i).P_i] \triangleright (x : [\&_i \vec{\tau}_i]^{\downarrow}_{s} {\rightarrow} A) \otimes B}$$

B Transition Rules

We assume all l.h.s. processes are well-typed. A allows l unless: (1) $A(\mathsf{sbj}(l)) = *$ or (2) l is output and $\mathsf{md}(A(\mathsf{sbj}(l))) = \mathbf{!}$, cf. [6]. $\mathsf{n}(l)$ is the set of names in l.

$$\begin{array}{lll}
x[\&_i \vec{y}_i.P_i]^{A} \xrightarrow{x\mathtt{in}_i(\vec{y}_i)} P_i^{\vec{y}_i:\vec{\tau}_i \otimes A/x} & & (x : [\& \vec{\tau}_i]^{\downarrow} \in A) \\
!x[\&_i \vec{y}_i.P_i]^{A} \xrightarrow{x\mathtt{in}_i(\vec{y}_i)} !x[\&_i \vec{y}_i.P_i] | P_i^{\vec{y}_i:\vec{\tau}_i \otimes A} & & (x : [\& \vec{\tau}_i]^{!} \in A) \\
\overline{x}\mathtt{in}_i(\vec{y})P^{A} \xrightarrow{\overline{x}\mathtt{in}_i(\vec{y})} P^{\vec{y}:\vec{\tau}_i \otimes A/x} & & (x : [\oplus_i \vec{\tau}_i]^{\uparrow} \in A) \\
\overline{x}\mathtt{in}_i(\vec{y})P^{A} \xrightarrow{\overline{x}\mathtt{in}_i(\vec{y})} P^{\vec{y}:\vec{\tau}_i \otimes A} & & (x : [\oplus_i \vec{\tau}_i]^{\mathbf{?}} \in A)
\end{array}$$

$$\frac{P_1' \equiv_\alpha P_1 \;\; P_1^{A_1} \xrightarrow{l} P_2^{A_2} \;\; P_2 \equiv_\alpha P_2'}{P_1'^{\,A_1} \xrightarrow{l} P_2'^{\,A_2}} \qquad \frac{P_1^{A_1} \xrightarrow{l} P_2^{A_2} \quad x \notin \mathsf{n}(l)}{(\boldsymbol{\nu}\, x)P_1^{A_1/x} \xrightarrow{l} (\boldsymbol{\nu}\, x)P_2^{A_2/x}}$$

$$\frac{P_1^{A_1} \xrightarrow{l} P_2^{A_2} \quad A_1 \odot B \text{ allows } l}{P_1|Q^{\,A_1 \odot B} \xrightarrow{l} P_2|Q^{\,A_2 \odot B}} \qquad \frac{P_1^{A_1} \xrightarrow{l} P_2^{A_2} \quad Q_1^{B_1} \xrightarrow{\bar{l}} Q_2^{B_2}}{P_1|Q_1^{A_1 \odot B_1} \xrightarrow{\tau} (\boldsymbol{\nu}\, \mathsf{bn}(l))(P_2|Q_2)^{\,A_2 \odot B_2/\mathsf{bn}(l)}}$$

$$\frac{P_1^{A_1} \xrightarrow{l} P_2^{A_2} \quad \mathsf{n}(l) \cap \{\vec{y}\} = \emptyset}{\overline{x}\mathtt{in}_i(\vec{y})P_1^{\,A_1/\vec{y} \odot x:[\oplus_i \vec{\tau}_i]^p} \xrightarrow{l} \overline{x}\mathtt{in}_i(\vec{y})P_2^{\,A_2/\vec{y} \odot x:[\oplus_i \vec{\tau}_i]^p_i}}$$

$$\frac{P_1^{A_1} \xrightarrow{x\mathtt{in}_i(\vec{z})} P_2^{A_2}}{\overline{x}\mathtt{in}_i(\vec{y})P_1^{\,A_1/\vec{y} \odot x:[\oplus_i \vec{\tau}_i]^p} \xrightarrow{\tau} (\boldsymbol{\nu}\, \vec{y})P_2\{\vec{y}/\vec{z}\}^{\,A_2/\vec{z}}}$$

We omit rules for unary actions and symmetric case of $|$. The rules are well-typed in the sense that if $P_1^{A_1}$ is well-typed and $P_1^{A_1} \xrightarrow{l} P_2^{A_2}$ then $P_2^{A_2}$ is well-typed.