Reasoning about Higher-Order Processes

by

Roberto M. Amadio and Mads Dam
Reasoning about Higher-Order Processes

Roberto M. Amadio  Mads Dam
CRIN-INRIA, Nancy 1  SICS, Stockholm 2

October 20, 1994

SICS Research Report RR:94–18

1CRIN-INRIA, BP 239, F-54506, Vandœuvre, FRANCE, e-mail: amadio@loria.fr. The first author is partially supported by ESPRIT BRA 6454 CONFER, part of this work was done while visiting SICS.

2SICS, Box 1263, S-164 28, Kista, SWEDEN, e-mail: mfd@sics.se. The second author is partially supported by ESPRIT BRA 8130 LOMAPS.
Abstract

We address the specification and verification problem for process calculi such as Chocs, CML and Facile where processes or functions are transmissible values. Our work takes place in the context of a static treatment of restriction and of a bisimulation-based semantics. As a paradigmatic and simple case we concentrate on (Plain) Chocs. We show that Chocs bisimulation can be characterized by an extension of Hennessy-Milner logic including a constructive implication, or function space constructor. This result is a non-trivial extension of the classical characterization result for labelled transition systems. In the second part of the paper we address the problem of developing a proof system for the verification of process specifications. Building on previous work for CCS we present an infinitary sound and complete proof system for the fragment of the calculus not handling restriction.

Keywords: Higher-order process calculi; Bisimulation; Modal logics; Program specification; Program verification.
1 Introduction

In the last years there has been a rising interest in calculi and programming languages where complex data such as processes and functions are transmissible values [4, 6, 13, 16, 20]. At least two main motivations for these studies can be identified:

(i) to generalize the functional model of computation to a parallel and/or concurrent framework, and

(ii) to model the notion of code transmission which is relevant to the programming of distributed systems.

A key issue in these languages is the interaction between process transmission and the static scoping discipline for communication channels. Here we consider Thomsen’s Plain Chocs. This is an extension of CCS where processes are transmissible values and the restriction operator is subject to a static scoping discipline.

A considerable effort has been put into the development of a bisimulation based semantics for this calculus (c.f. [20, 2, 17]). The specification of Plain Chocs processes (and processes in related calculi) is a much less developed topic. Two notable attempts in this direction are described in [19, 7]. These works are based on logics extracted from a domain theoretic interpretation of the calculus, following general ideas described in, e.g., [1]. This approach has been rather successful in the case of dynamic scoping. On the other hand it is not clear how to obtain a fully abstract denotational semantics of restriction in the case of static scoping (c.f. [15] for some typical problems). This motivates our shift towards an operational approach to the problem, along the lines of Hennessy and Milner [9].

What to specify? First, let us fix some notation for a process calculus of higher-order processes: $c!P.P$ is the process which sends $P$ along the channel $c$ and then becomes $P'$, $c?x.P$ is the process which performs an input along the channel $c$, and upon reception of some process $Q$ becomes $[Q/x]P$. In $\nu c.P$ the restriction operator $\nu$ creates a new channel which will be local to the process $P$. Finally $+$ is the non-deterministic choice, $|$ is the parallel composition, and $0$ is the nil process, with the usual CCS semantics [11]. We briefly refer to this calculus as Chocs, after [19].

Second, we should determine some requirements for our candidate specification logic. Roughly, we expect it to be an extension of Hennessy-Milner logic which characterizes some standard Chocs bisimulation.

Previous work on extending Hennessy-Milner logic to calculi including value and channel transmission (c.f. [5, 8, 12]) relies on the recurrent idea of introducing modalities that state properties of the transmitted values. For instance, one can specify that a process $P$ can output the value 3 on channel $c$ and then satisfy property $\phi$ by writing: $P : (c!3).\phi$. 

1
This approach does not seem to scale up to process transmission. The naive idea of writing:

\[ P : \langle c! \phi' \rangle. \phi \text{ if } P \xrightarrow{\xi_{P'}} Q \text{ and } P' : \phi' \text{ and } Q : \phi \]

does not take into account the fact that \( P' \) and \( Q \) might share local channels. For instance, consider the process \( P = \nu a. c! (a? x.x).a! 0.Q \). In this example the actions of the process transmitted on channel \( c \) and of the relative continuation are clearly inter-dependent. We did not find any satisfying way to express this dependency. An alternative is to express properties of processes like \( P \) above in terms of the effect the output has when \( P \) is put in a receiving environment. Since receiving environments are just abstractions this suggests a simple extension of Hennessy-Milner logic by means of a constructive implication, say \( \Rightarrow \). We can now write specifications such as:

\[ P : [c?](\phi \Rightarrow \psi) \quad Q : \langle c! \rangle((\phi \Rightarrow \psi) \Rightarrow \gamma) \]

For inputs the interpretation is the expected one: \( P \) satisfies \([c?](\phi \Rightarrow \psi)\) if whenever \( P \) makes an input action on channel \( c \) and receives an input satisfying \( \phi \) then the continuation will satisfy \( \psi \). In first approximation, the intuition for output is the following: \( Q \) satisfies \( \langle c! \rangle((\phi \Rightarrow \psi) \Rightarrow \gamma) \) if it is possible for \( Q \) to output a process, say \( Q_1 \), along \( c \) such that in any receiving context, say \( \lambda x. Q_2 \), if \( \lambda x. Q_2 \) satisfies \( \phi \Rightarrow \psi \), and if \( Q_3 \) is the continuation of \( Q \) after performing its output, then \( ([Q_1/x]Q_2) \mid Q_3 \) satisfies \( \gamma \). In general \( Q_1 \) and \( Q_3 \) may share channels local to \( Q \), say \( c_1, \ldots, c_n \), whose scope is \textit{extruded} by the output communication. Note that in the specification we never need to speak about these extruded channels.

**Operational Semantics.** Reflecting this intuition, a labelled transition system is given to Plain Chocs that maps (closed) processes to (closed) process functionals, depending on the action performed. More precisely, a process \( Q \) rewrites by an input action to a process function \( \lambda x. Q_2 \) and by an output action to a process functional \( \lambda f. \nu c_1 \ldots c_n. (fQ_1 \mid Q_3) \), where as in the previous explication \( Q_1 \) is the transmitted process, \( Q_3 \) is the continuation, and \( c_1, \ldots, c_n \) are shared local channels. The result of a communication is simply computed by applying the process functional to the process function:

\[
(\lambda f. \nu c_1 \ldots c_n. (fQ_1 \mid Q_3))(\lambda x. Q_2) \equiv \nu c_1 \ldots c_n. ([Q_1/x]Q_2 \mid Q_3)
\]

The standard rules for substitution avoid clashes between local channels. Also note that in this formulation Plain Chocs actions coincide with standard CCS actions. Of course one has to pay a price for this, namely one has to lift the notion of bisimulation higher-order by introducing a suitable notion of exponentiation. Section 2 will show that this can be achieved in an elegant and simple way. It should be remarked that the resulting bisimulation coincides with the one considered in [2, 17], which in turn has been shown to be compatible with the \( \pi \)-calculus semantics.
Logical Characterization. Having found a suitable way to specify properties of Chocs processes we pursue our programme of relating logical equivalence to bisimulation equivalence. In the CCS case, this is achieved by means of a coinductive view of bisimulation. Roughly, the bisimulation relation, say \( \sim \), can be seen as the limit of a descending sequence of equivalence relations \( \sim^k \). Equivalence in \( \sim^k \) is then related to logical equivalence w.r.t. formulas having modal depth bound by \( k \). In the higher-order case the task is complicated by the contravariance of the constructive implication in its first argument. This is discussed in more detail in section 2 once some notation has been introduced. We obtain a logical characterization of Chocs bisimulation modulo a technical lemma that relates the approximation \( \sim^k \) to a sharpened approximation \( \sim^k_4 \).

Towards a Proof System. As a second contribution, we address the problem of developing a sound and complete proof system to verify that a process meets (or realizes) a specification. We found a simple and clean solution for the restriction-free fragment of the calculus. The basic judgment \( \Gamma \vdash P : \psi \) states that the process \( P \) realizes the specification \( \psi \) under the hypothesis \( \Gamma \). Hypotheses state assumptions on the parameters of \( P \). The system thus allows for reasoning about open processes. Completeness is achieved by the introduction of an \( \omega \)-rule and by the hypothesis that there is only a finite number of channels. It appears that a treatment of restriction would require a considerable complication of the proof system as one has to represent the dependencies among functional variables and dynamically generated channels. We leave this problem for further investigation.

Paper Organization. Section 2 presents the Chocs calculus and the relative notion of bisimulation. Section 3 introduces a Hennessy-Milner logic to specify properties of Chocs processes and shows that it captures exactly Chocs bisimulation. Section 4 describes a sound and complete formal system for proving properties of processes.

2 The Calculus and its Bisimulation Based Semantics

Language. The expressions of the language are classified in two kinds: channels and processes. Channels are variables and ranged over by \( c, d, \ldots \). Actions have one of the forms \( \tau, c? \) or \( c! \) and they are ranged over by \( \alpha, \alpha', \ldots \). To each process is associated a unique order among the orders: 0 (processes), 1 (process functions), and 2 (process functionals). We use \( x, x', \ldots, f, f', \ldots, \) and \( F, F', \ldots \), for variables of order 0, 1, 2, respectively. We use \( z, z' \ldots \) as generic variables. Open processes of order 0 are then generated by the following grammar:

\[
P ::= 0 \mid X \mid \tau P \mid (P + P) \mid (P \mid P) \mid (\nu c.P) \mid (c!P.P) \mid (c?x.P)
\]
Whenever we write \( P[z] \) we intend that \( z \) is the only variable that can be free in \( P \) and moreover we identify \( P \) with the function \( \lambda z. P \). Thus alpha-conversion applies to identify \( P[z] \) with \( ([z'/z]P)[z'] \) whenever \( z' \) does not occur freely in \( \lambda z. P \), and to identify, e.g., \( \nu c.(P[z]) \) with \( (\nu c.P)[z] \). We also write \( P(z) \) for an open process in which \( z \) is the only variable that can occur free. If \( z \) is free then \( P(z) \) is identified with \( P[z] \). If \( z \) is not free then \( P(z) \) can ambiguously represent either a closed process or the constant function \( \lambda z. P \). The context will allow us to disambiguate this situation.

**Operational semantics.** The labelled transition system is based on three kinds of judgments: \( P \rightarrow Q, P \rightarrow^* Q'[x], \) and \( P \rightarrow^* Q''[f], \) where \( P, Q \) are closed processes. We assume that sum and parallel composition are associative and commutative operators, and that restriction commutes with parallel composition according to the standard law \( \nu c.(P | P') \rightarrow \nu c.(P | P') \) whenever \( c \) is not free in \( P' \). Then it can be showed that whenever \( P \rightarrow^* Q''[f] \) in the transition system specified below then \( Q''[f] \) has the form \( \nu c_1 \ldots \nu c_n(f \parallel P' | P'') \). Finally, note that in the rule (!?) a second-order substitution is employed. That is, one replaces first \( Q'[x] \) for \( f \), and then the argument of \( f \) for \( x \).

\[
(!) \ c!P'.P \rightarrow (fP' | P)[f] \\
(?!) c?x.P \rightarrow^* P[x] \\
(!?) if P \rightarrow (fP' | P)[f] and Q \rightarrow Q'[x] then P | Q \rightarrow [Q'[x]/f]P'[f] \\
(+) if P \rightarrow P' then P + Q \rightarrow P' \\
(\nu) if P \rightarrow P' then P | Q \rightarrow P' | Q \\
(\nu) if P \rightarrow P' and \alpha \neq c!, c? then \nu c.P \rightarrow \nu c.P'
\]

**Bisimulation.** Let \( Pr_0 \) be the collection of closed processes, \( Pr_1 \) be the collection of \( P[z] \) processes, and \( Pr_2 \) be the collection of \( P[f] \) processes. Because of the input-output actions a notion of bisimulation over \( Pr_0 \) needs to be lifted to \( Pr_1 \) and \( Pr_2 \). For this purpose the following general notion of exponentiation is introduced:

\[
P[z] [S \rightarrow S'] P'[z] if Q(w) S Q'(w) implies [Q(w)/z]P S' [Q'(w)/z]P'
\]

Given a relation \( S \) over \( Pr_0^2 \) and an action \( \alpha \) we define the relations \( S[\alpha] \) as follows, where \( Id_0 = \{(P, P) \mid P \in Pr_0\} \), and \( Id_1 = \{(P[z], P[z]) \mid P[z] \in Pr_1\} \):

\[
S[\tau] = S \quad S[c?] = [Id_0 \Rightarrow S] \quad S[c!] = [Id_1 \Rightarrow S]
\]
Definition 2.1 (Bisimulation) A bisimulation \( S \) is a relation over \( Pr_0 \) such that whenever \( PSQ \) and \( P \xrightarrow{\alpha} P'(z) \) then for some \( Q'(z) \), \( Q \xrightarrow{\alpha} Q'(z) \) and \( P'(z)S[\alpha]Q'(z) \); and symmetrically. We denote with \( \sim \) the largest bisimulation.

Up to some notational conventions \( \sim \) is the bisimulation studied in [2, 17]. The relation \( \sim \) is extended to process functionals by considering their equivalence on all closed instances, e.g. \( P[f] \sim Q[f] \) if any \( R[x], \ [R[x]/f]P \sim [R[x]/f]Q \). Define now the function \( F : Pr_0^2 \to Pr_0^2 \) by \( P \sim F(S)Q \) if whenever \( P \xrightarrow{\alpha} P'(z) \) then \( Q \xrightarrow{\alpha} Q'(z) \) for some \( Q'(z) \) and \( P'(z)S[\alpha]Q'(z) \); and symmetrically. Also, let \( \sim^0 = Pr_0 \), \( \sim^{s+1} = F(\sim^s) \), and \( \sim^\omega = \bigcap_{n<\omega} \sim^n \). The relations \( \sim^n \) are extended to functionals following the convention for \( \sim \).

Proposition 2.2 (Properties of \( F \)) The set \( 2^{Pr_0^2} \) is a complete lattice when ordered by set inclusion. Then:

1. \( F \) is monotone.
2. \( S \) is a bisimulation iff \( S \subseteq F(S) \).
3. If \( \{X_i\}_{i \in I} \) is a directed set, then \( F(\bigcap_{i \in I} X_i) = \bigcap_{i \in I} F(X_i) \).
4. The greatest bisimulation \( \sim \) exists and coincides with \( \sim^\omega \).

Proof. (1) Observe that \( S \subseteq S' \) implies \( S[\alpha] \subseteq S'[\alpha] \). (2) Immediate.

(3) Since \( F \) is monotone it remains to show: \( F(\bigcap_{i \in I} X_i) = \bigcap_{i \in I} F(X_i) \).

Suppose: \( P \bigcap_{i \in I} F(X_i)Q \) and \( P \xrightarrow{\alpha} P'(z) \).

By hypothesis: \( \forall i \in I. \exists Q'(z). (Q \xrightarrow{\alpha} Q'(z) \land P'(z)X_i[\alpha]Q'(z)) \).

Since the labelled transition system is image finite and \( \{X_i[\alpha]\}_{i \in I} \) is a directed set, one can switch the two quantifications and derive:

\( \exists Q'(z). \forall i \in I. (Q \xrightarrow{\alpha} Q'(z) \land P'(z)X_i[\alpha]Q'(z)) \).

This concludes the proof in the case \( \alpha = \tau \). In the other cases observe:

\( \forall i \in I. (P'[z]X_i[\alpha]Q'[z]) \) is equivalent to \( P'[z](\bigcap_{i \in I} X_i[\alpha])Q'[z] \) which is what we need to prove.

(4) Corollary of (3). \( \square \)

Proposition 2.3 (Congruence) The relations \( \sim^k \), for \( k \leq \omega \), are congruences with respect to all the calculus operators. That is,

\[
\begin{align*}
P_i \sim^k Q_i, i = 1, 2 & \Rightarrow P_1 + P_2 \sim^k Q_1 + Q_2, P_1 \mid P_2 \sim^k Q_1 \mid Q_2, \ c!P_1.P_2 \sim^k c!Q_1.Q_2 \\
P \sim^k Q & \Rightarrow \nu c.P \sim^k \nu c.Q \\
P[x] \sim^k Q[z] & \Rightarrow c!x.P \sim^k c!z.Q
\end{align*}
\]

Proof. The only difficulty arises with parallel composition. For instance, in the case \( k = \omega \) one shows that \( \{\nu c_1.\cdots\nu c_n.(P \mid Q), \nu c_1.\cdots\nu c_n.(P' \mid Q) \mid P \sim P'\} \) is a bisimulation. \( \square \)

We give an alternative characterisation of the \( \sim^k \) relations in terms of "sharpened" approximations, \( \sim^k_\sharp \). These will be important when it comes to relating the logical and bisimulation based equivalences. These sharpened relations \( \sim^k_\sharp \) are defined as follows:
\[ P \sim^0 Q \quad \text{always} \]
\[ P \sim^{k+1}_q Q \quad \text{if } P \overset{\omega}{\rightarrow} P'(z) \text{ implies } Q \overset{\omega}{\rightarrow} Q'(z) \text{ for some } Q'(z) \]
\[ \text{such that } P'(z) \sim^k Q'(z); \text{ and symmetrically} \]
\[ P[x] \sim^{k}_q Q[x] \quad \text{if } P[x] \overset{[\sim^k_q \Rightarrow \sim^k_q]}{\rightarrow} Q[x] \]
\[ P[f] \sim^{k}_q Q[f] \quad \text{if } P[f] \overset{[\sim^k_q \Rightarrow \sim^k_q]}{\rightarrow} Q[f] \]

We can now show that the sharpened approximation relations coincide with the approximations \( \sim^k \). This result relies on the congruence properties of \( \sim^k \).

**Proposition 2.4** For any \( k < \omega \), \( \sim^k \) coincides with \( \sim^k_q \).

**Proof.** By induction on \( k \) and the order. We present the order 1 case.

Suppose \( P[x] \sim^{k+1}_q Q[x] \). For any \( R \) we know: \( R \sim^{k+1}_q R \).

Hence \( [R/x]P \sim^{k+1}_q [R/x]Q \).

This implies by ind. hyp. \( [R/x]P \sim^{k+1}_q [R/x]Q \). Conclude: \( P[x] \sim^{k+1}_q Q[x] \).

Vice versa, suppose \( P[x] \sim^{k+1}_q Q[x] \) and \( R \sim^{k+1}_q R' \).

By hyp. \( R \sim^{k+1}_q R' \).

By congruence properties: \( [R/x]P \sim^{k+1}_q [R'/x]P \) and \( [R/x]Q \sim^{k+1}_q [R'/x]Q \).

By hypothesis: \( [R/x]P \sim^{k+1}_q [R/x]Q \) and \( [R'/x]P \sim^{k+1}_q [R'/x]Q \).

By transitivity: \( [R'/x]P \sim^{k+1}_q [R'/x]Q \). By ind. hyp. \( [R/x]P \sim^{k+1}_q [R'/x]Q \).

Conclude: \( P[x] \sim^{k+1}_q Q[x] \).

\[ \square \]

### 3 Logical Characterization

**Modal Formulas.** Process properties are specified by the modal formulas which are generated by the following grammar, where \( X \) is a countable set. As in the case of processes, specifications also have an order. A specification of a certain order can only be predicated of a process of the same order. Conjunction and disjunction apply to formulas of the same order.

\[
\phi ::= \bigwedge_{x \in X} \phi_x \mid \bigvee_{x \in X} \phi_x \mid \langle \alpha \rangle \phi \mid [\alpha] \phi \mid \phi \Rightarrow \phi
\]

The truth- and falsehood constants \( \top \) and \( \bot \) are defined as usual: \( \top = \bigwedge \emptyset \) and \( \bot = \bigvee \emptyset \). These formulas are overloaded as they may have order 0, 1, and 2. We sometimes use \( (\cdot) \) as a meta-connective ranging over \( \{\cdot, [\cdot]\} \).

**Realizability.** We specify when a process \( P(z) \) realizes a formula \( \phi \), written as \( \models P(z) : \phi \), by induction on the structure of \( \phi \). Note that a realizer of a formula \( \phi \Rightarrow \psi \) is always a function, and a realizer of a modality is always a ground process.

\[
\models P(z) : \bigwedge_{x \in X} \phi_x \quad \text{if for all } x \in X \models P(z) : \phi_x
\]
\[
\models P(z) : \bigvee_{x \in X} \phi_x \quad \text{if for some } x \in X \models P(z) : \phi_x
\]
\[
\models P[x] : \phi \Rightarrow \psi \quad \text{if for all } Q(z'), \models Q(z') : \phi \text{ implies } \models [Q(z')/z]P : \psi
\]
\[
\models P : \langle \alpha \rangle \phi \quad \text{if for some } P'(z), P \overset{\omega}{\rightarrow} P'(z) \text{ and } \models P'(z) : \phi
\]
\[
\models P : [\alpha] \phi \quad \text{if whenever } P \overset{\omega}{\rightarrow} P'(z), \models P'(z) : \phi
\]

6
The modal depth $|\phi|$ of a formula $\phi$ is defined as follows:

$$|\Lambda x \in X \phi x| = |\forall x \in X \phi x| = \sup_{x \in X} |\phi x|$$

$$|\phi \Rightarrow \psi| = |\psi|$$

$$|\langle \alpha \rangle \phi| = |[\alpha] \phi| = 1 + |\phi|$$

**Definition 3.1 (Logical equivalences)** We define the family of equivalence relations on process (functionals) $\sim^*_\kappa$ by $P(z) \sim^*_\kappa Q(z')$ if for all $\phi$ such that $|\phi| \leq \kappa$, $P(z) : \phi$ if and only if $Q(z') : \phi$. Also set: $P(z) \sim_L Q(z')$ if $P(z) \sim^*_\kappa Q(z')$, for any $\kappa$.

**Proposition 3.2** (1) For any $\kappa$, if $P(z) \sim^\kappa Q(z')$ then $P(z) \sim^*_\kappa Q(z')$. (2) If $P(z) \sim Q(z')$ then $P(z) \sim_L Q(z')$.

**Proof.** (1) We show that if $P(z) \sim^\kappa Q(z')$, $|\phi| \leq \kappa$, and $\models P(z) : \phi$ then $\models Q(z') : \phi$ by induction on the structure of $\phi$. The only non-standard case is when $\phi$ has the form $\phi_1 \Rightarrow \phi_2$. Suppose $\models P(z_1) : \phi_1$. Then $\models [P(z_1)/z]P : \phi_2$. By congruence of $\sim^\kappa$, $[P(z_1)/z]P \sim^\kappa [P(z_1)/z']Q$. By the induction hypothesis, $\models [P(z_1)/z']Q : \phi_2$ as desired.

(2) Immediate from (1). \qed

**Definition 3.3 (Characteristic formula)** For any process functional $P(z)$, and ordinal $k \leq \omega$ we inductively define a formula $C^k(P(z))$:

$$C^0(P(z)) = T$$

$$C^{k+1}(P) = \bigwedge_{P^a \models P(z)} ((\langle \alpha \rangle, C^k(P'(z)))) \land \bigwedge_{\alpha \in \text{Act}} \bigvee_{P^a \models P(z)} C^k(P'(z))$$

$$C^{k+1}(P[z]) = \bigwedge_{R(z')} C^{k+1}([R'(z')/z]P)$$

$$C^{\omega}(P(z)) = \bigwedge_{k<\omega} C^k(P(z))$$

Observe that for any $k \leq \omega$, $|C^k(P(z))| \leq k$.

**Proposition 3.4** For any $k < \omega$,

(1) For all $P(z)$, $\models P(z) : C^k(P(z))$.

(2) For all $P(z), Q(z')$, $\models P(z) : C^k(Q(z'))$ iff $P(z) \sim^k Q(z')$.

**Proof.** One proves (1) and (2) at the same time, by induction on $k$ and the order. We present the function case.

(1) We have to show: $\models P(x) : \Lambda_{R} C^{k+1}(R) \Rightarrow C^{k+1}([R/x]P)$.

Suppose: $\models R' : C^{k+1}(R)$. By ind. hyp. and (2): $R' \sim_{k+1} R$.

By congruence: $[R'/x]P \sim_{k+1} [R/x]P$.

By ind. hyp. and (2): $\models [R'/x]P : C^{k+1}([R/x]P)$.
(2) Suppose \( \vdash P[x] : C^{k+1}(Q[x]) \). That is any \( R \), 
\( \vdash P[x] : C^{k+1}(R) \Rightarrow C^{k+1}([R/x]Q) \). By ind. hyp and (1): \( \vdash R : C^{k+1}(R) \).
Hence: \( \vdash [R/x]P : C^{k+1}([R/x]Q) \). That is: \( [R/x]P \sim_{k+1} [R/x]Q \).
Vice versa, suppose \( P[x] \sim_{k+1} Q[x] \). Given any \( R \), let \( \vdash R' : C^{k+1}(R) \).
Then \( R' \sim_{k+1} R \), which implies: \( [R'/x]P \sim_{k+1} [R/x]P \sim_{k+1} [R/x]Q \).

We thus obtain

**Corollary 3.5** For any \( k < \omega \), \( P \sim_{L}^{k} Q \) implies \( P \sim^{k} Q \).

**Theorem 3.6 (Logical characterization)** For any processes \( P, Q \),

\[
P \sim Q \iff P \sim_{L} Q.
\]

**Proof.** Follows immediately from previous results.

**Remark.** We can now explain a technical problem that would arise in a direct proof of the logical characterization theorem. We want to show by contradiction that \( P \sim_{L} P' \) implies \( P \sim P' \). We consider the function case. Suppose \( P[x][Id \Rightarrow \sim \mapsto P'[x] \) does not hold. Then neither does \( P[x][Id \Rightarrow \sim \mapsto [Q/x]P' \) for some \( k \), i.e. there is a \( Q \) such that \( [Q/x]P \sim_{k} [Q/x]P' \) fails. Inductively one can find a formula \( \psi \) that separates \( [Q/x]P \) from \( [Q/x]P' \), and such that \( |\psi| \leq k \). The problem is that (appearingly) we have no control on \( Q \) which can be an arbitrarily complex process. However, our approach shows that it is possible to give a sufficiently good description of \( Q \) by means of a formula of modal depth bounded by \( k \).

**A Characterization of Chocs Bisimulation.** There is an alternative and natural definition of bisimulation, which resembles the definition of sharpened approximation. Given a relation \( S \) over \( Pr_{2} \) and an action \( \alpha \) we define the relations \( S\{\alpha\} \) as follows:

\[
S\{\tau\} = S \quad S\{c?\} = [S \Rightarrow S] \quad S\{cl\} = [[S \Rightarrow S] \Rightarrow S]
\]

**Definition 3.7 (Modified bisimulation)** A modified bisimulation \( S \) is a relation over \( Pr_{0} \) such that whenever \( PSQ \) and \( P \Rightarrow P'(z) \) then for some \( Q'(z) \), \( Q \Rightarrow Q'(z) \) and \( P'(z)S\{\alpha\}Q'(z) \); and symmetrically.

**Proposition 3.8** Among the modified bisimulations \( S \) such that \( Id_{0} \subseteq S \) and \( [S \Rightarrow S] \subseteq Id_{1} \) there is largest one and it coincides with the largest bisimulation.

**Proof.** Let \( S \) be a modified bisimulation with the required properties. Then for any action \( \alpha \), \( S\{\alpha\} \subseteq S\{\alpha\} \). Hence \( S \) is a bisimulation. Next observe that \( \sim \) is a modified bisimulation by its congruence properties. So it is also the largest modified bisimulation with the required properties.
Intervals. The notion of modified bisimulation is rather hard to handle as it does not induce a monotone function on the space $2^{Pr^2}$, because of the contravariance of the constructive implication. There is a well-known technique to overcome this problem which consists in lifting the problem to a space of intervals. In our case an interval is a pair $[I, J]$, where: $I \subseteq J \subseteq Pr^2_0$. Intervals are an example of three-valued interpretation in which one speaks about what is true, what is possibly true (or false) and what is false. Intervals can be ordered by $[I, J] \leq [I', J']$ if and only if $I \subseteq I'$ and $J' \subseteq J$. Over this space the following interpretation of the constructive implication is monotone:

$[I, J] \Rightarrow [I', J'] = [J \Rightarrow I', I \Rightarrow J']$

Following these ideas it would be possible to base our work on the notion of modified bisimulation.

4 Towards a Proof System

In this section we develop a proof system to prove properties of processes stated in the finitary fragment of the Hennessy-Milner logic previously introduced. These results are of a preliminary nature as they are obtained under the following strong assumptions:

1. We drop restriction.
2. We assume an infinitary $\omega$-rule.
3. We suppose that the calculus has a finite number of channels.

The $\omega$-rule reduces the provability of open terms to the provability of their closed instances. Whether some weak completeness result can be obtained while dropping this rule remains to be seen. The restriction to finite label alphabet has two important corollaries. First, the following rule PAR-BOX-$\tau$ is finitary. Without this hypothesis a finitary statement of the PAR-BOX-$\tau$ rule could be obtained, for instance, by introducing a channel quantifier in the specification language. Indeed it appears that this would be a natural extension of the language. For example, note that the property that a process can perform no actions could be stated as: $\forall c. ([c] \bot \land [\tau] \bot \land [c?] \bot)$. Second, the k-th characteristic formula of any process is finite, for $k < \omega$. This is quite useful in arguing about the completeness of the system.

We regard (1) as the main limitation of our system as in practice it is possible to prove interesting facts without applying the $\omega$-rule and assuming a finite number of channels.
4.1 Syntactic Conventions

Judgments. A context $\Gamma$ is a set $z_1 : \psi_1, \ldots, z_n : \psi_n$ where all $z_i$ are pairwise distinct. The basic judgments are sequents of the following shape:

$$\Gamma \vdash P : \psi$$

The process $P$ and the context $\Gamma$ might contain variables of order 0, 1, 2. There can be at most one variable which is free in $P$ and does not occur in $\Gamma$. Following our conventions this variable should be intended as $\lambda$-abstracted (note that in our specific case this variable can be of order 0, 1). The grammars of processes $P_0, P_1, P_2$ of orders 0, 1, 2, respectively, in a context $\Gamma$ can be given as follows:

$$P_0 ::= 0 \mid x \mid P_0 + P_0 \mid c! P_0 . P_0 \mid c? x . P_0 \mid f P_0 \mid F P_1$$

$$P_1 ::= P_0[x] \quad P_2 ::= P_0[f]$$

In the following $P$ will denote a generic process and $\psi$ a generic formula.

Eta-expansions. By convention we eta-expand functional variables so that: $f = f x[x], F = F f[f]$. This allows to fit functional variables in the grammar for $P_1$ and $P_2$. In the following we will write $z_1 : P_0$. If $z_1 \equiv f_1$ then $z_1 : P_0 \equiv (f_1 x : P_0)[x]$, and, similarly, if $z_1 \equiv F_1$ then $z_1 : P_0 \equiv (F_1 f : P_0)[f]$ ($x, f$ fresh variables).

Interpretation. We write $z_1 : \psi_1, \ldots, z_n : \psi_n \models P : \psi$ if for all closed $P_1$ such that $\models P_1 : \psi_i$ (i = 1, ..., n) we have $\models [P_1/z_1, ..., P_n/z_n]P : \psi$.

4.2 Proof System

We divide the rules of the proof system in three groups: general rules for the manipulation of the sequents, sequent calculus rules which allow for the (right and left) introduction of logical operators, and finally rules which exploit the process structure (see fig. 1).

While the first two groups are quite stable (coming straight from proof theory) the last group is still open to debate. Here we follow quite closely previous work by Colin Stirling [18] concerning proof systems for CCS (note however the different formulation of the rules for parallel composition). The rules reflect very closely the operational semantics. The most involved rules are those for parallel composition. To prove a property $\phi$ of a parallel composition, say, $P \mid Q$, one needs in general to:

1. guess properties of the parallel constituents $P$ and $Q$,

2. show that they hold, and
3. show that the holding of these properties for the constituents entails the holding of \( \phi \) for their parallel composition.

We regard this as quite natural and reflecting closely the *compositional* nature of the proof system. We do not expect that any of these three tasks can be automated in general though they can be for certain special cases. For instance for finite state CCS processes it is possible to efficiently transform the guess for one parallel component to a property required to hold for the other (c.f. [10, 3]). In general, however, we can as yet provide no assistance, and in practice during a proof one is often forced to backtrack in order to modify the hypotheses. This is clearly a potential problem, and it remains to be seen if a more efficient proof system can be developed.

We have omitted the rules symmetric to AND-L, OR-R, SUM-DIA, PAR-DIA, and PAR-DIA-\( \tau \). Really, \( \land, \lor, +, \) and \( | \) should be understood as commutative operators.

Most rules should be self-explanatory. The essential idea is that in general the holding of \( \Gamma \vdash P : \phi \) depends on the structure of both \( P \) and \( \phi \). In all cases, but for the modal operators, \( P \) can be dealt with uniformly — these are the logical rules. For the modal operators, however, the structure of \( P \) is essential, and its transition behaviour is exposed by the operational semantics from which the rules for the modal operators are derived in a quite systematic fashion.

### 4.3 Soundness and Completeness

In this section we establish the soundness and completeness of the proof system w.r.t. the proposed interpretation.

**Proposition 4.1 (Soundness)** If \( \Gamma \vdash P : \psi \) then \( \Gamma \vdash P : \psi \).

**Proof.** Proofs are well-founded, countably branching trees. To every proof one can associate a (transfinite) ordinal which measures its depth. Proceed by transfinite induction on the proof depth. \( \square \)

We next turn to the issue of completeness. First notice that without the \( \omega \)-rule the proof system is incomplete, even if we restrict attention to closed processes. For instance one cannot prove the following valid judgment: \( \vdash c?x.x : \langle c? \rangle (\top \Rightarrow (\langle \tau \rangle \top \lor [\tau] \bot)) \). However, even without using the \( \omega \)-rule it is possible to prove simple, but non-trivial, properties of processes. In fig. 2 we give an example proof of the judgment \( \vdash a!b!0.b?y.c!0 \mid a?x.x : \langle \tau \rangle \langle \tau \rangle \langle c! \rangle \top \) where we have adopted the abbreviations \( \phi_0 = a_1 \Rightarrow \phi_1 \), \( \phi_1 = \langle b! \rangle (\phi_2 \Rightarrow \langle c! \rangle \top) \), and \( \phi_2 = \top \Rightarrow \langle c! \rangle \top \). Typically, proofs are constructed bottom up. It is useful to consider successive refinements of the formulas involved in the PAR-DIA-\( \tau \) rule in fig. 2. In practice one introduces formula variables which are incrementally resolved as the proof goes on. For instance the instantiations of \( \phi_0, \phi_1 \) and \( \phi_2 \) in fig. 2 have been arrived at in this way.

11
Sequent Structure Rules

HYP \[ \Gamma, z : \psi \vdash z : \psi \]

OUT \[ \Gamma \vdash P' : \psi', \Gamma, z : \psi' \vdash P : \psi \]
\[ \Gamma \vdash [P'/z]P : \psi \]

OMEGA \[ \text{For all } P' \text{ such that } \models P' : \psi', \Gamma \vdash [P'/z]P : \psi \]

Logical Rules

BOT-L \[ \Gamma, z : \bot \vdash P : \psi \]

TOP-R \[ \Gamma \vdash P : \top \]

AND-L \[ \Gamma, z : \psi_1 \vdash P : \psi \]
\[ \Gamma, z : \psi_1 \land \psi_2 \vdash P : \psi \]

AND-R \[ \Gamma \vdash P : \psi_1 \]
\[ \Gamma \vdash P : \psi_2 \]
\[ \Gamma \vdash P : \psi_1 \land \psi_2 \]

OR-L \[ \Gamma, z : \psi_2 \vdash P : \psi \]
\[ \Gamma, z : \psi_1 \lor \psi_2 \vdash P : \psi \]

OR-R \[ \Gamma \vdash P : \psi_1 \]
\[ \Gamma \vdash P : \psi_2 \]
\[ \Gamma \vdash P : \psi_1 \lor \psi_2 \]

⇒-L \[ \Gamma \vdash P' : \psi_1 \]
\[ \Gamma, z : \psi_2 \vdash P : \psi \]
\[ \Gamma, z : \psi_1 \Rightarrow \psi_2 \vdash [zP'/z]P : \psi \]

⇒-R \[ \Gamma, z : \psi' \vdash P : \psi \]
\[ \Gamma \vdash P[z] : \psi' \Rightarrow \psi \]

Process Structure Rules

OUT-1 \[ \Gamma \vdash (fP' \mid P)[f] : \psi \]
\[ \Gamma \vdash c!P'.P : [c]\psi \]

OUT-2 \[ \Gamma \vdash c!P'.P : [c]\psi \]

IN-1 \[ \Gamma \vdash P[z] : \psi \]
\[ \Gamma \vdash c?x.P : (c?)\psi \]

IN-2 \[ \Gamma \vdash c?x.P : (c?)\psi \]

SUM-DIA \[ \Gamma \vdash P : [\alpha]\psi \]
\[ \Gamma \vdash P + P' : [\alpha]\psi \]

SUM-BOX \[ \Gamma \vdash P : [\alpha]\psi \]
\[ \Gamma \vdash P + P' : [\alpha]\psi \]

NIL \[ \Gamma \vdash 0 : [\alpha]\psi \]

PAR-DIA \[ \Gamma \vdash P_1 : (\langle \alpha \rangle)\psi_1 \]
\[ \Gamma, z_1 : \psi_1 + z_1 \vdash P_2 : \psi \]
\[ \Gamma \vdash P_1 + P_2 : (\langle \alpha \rangle)\psi \]

PAR-DIA-τ \[ \Gamma \vdash P_1 : (d\langle \tau \rangle)\psi_1 \]
\[ \Gamma \vdash P_2 : (d\langle \tau \rangle)\psi_2 \]
\[ z_1 : \psi_1, z_2 : \psi_2 \vdash z_1 z_2 : \psi \]
\[ \Gamma \vdash P_1 + P_2 : (\langle \tau \rangle)\psi \]

PAR-BOX \[ \Gamma \vdash P_1 : [\alpha]\psi_1 \]
\[ \Gamma, z_1 : \psi_1 + z_1 \vdash P_2 : \psi \]
\[ \Gamma, z_2 : \psi_2 \vdash P_1 + z_2 : \psi \]
\[ (\alpha \neq \tau) \]

PAR-BOX-τ \[ \Gamma \vdash P_1 : [\tau]\psi_1 \]
\[ \Gamma, z_1 : \psi_1 + z_1 \vdash P_2 : \psi \]
\[ \Gamma, z_2 : \psi_2 \vdash P_1 + z_2 : \psi \]
\[ \Gamma \vdash P_1 + P_2 : [\tau]\psi_2 \]

\[ \Gamma \vdash P_1 : [d\langle \tau \rangle]\psi_1 \]
\[ \Gamma \vdash P_2 : [d\langle \tau \rangle]\psi_2 \]
\[ z_1 : \psi_1, z_2 : \psi_2 \vdash z_1 z_2 : \psi \]
\[ z_1 \vdash P_3 : [\langle \tau \rangle]\psi_3 (all \ d) \]
\[ z_2 \vdash P_4 : [\langle \tau \rangle]\psi_4 (all \ d) \]

PAR-BOX-τ \[ \Gamma \vdash P_3 + P_4 : [\tau]\psi \]

Figure 1: Proof System
\[ \vdash 0 : \top \quad \vdash x : (\epsilon) \top \vdash x : (\epsilon) \top \]
\[ \vdash \psi_0 : \phi_0 \vdash \psi_0 : (\epsilon) \top \]
\[ \vdash \psi_0[\psi] : \top \]
\[ \vdash \psi_0[\psi] : \phi_0 \]
\[ \vdash \theta[\psi_0] : \top \]
\[ \vdash \theta[\psi_0] : \phi_0 \]
\[ \vdash x : \phi_0 \vdash x : \phi_0 \]
\[ \vdash b ? y, c : (\epsilon) \phi_0 \]
\[ \vdash a x, x : (\epsilon) \phi_0 \]

Figure 2: Proof example

**Proposition 4.2** Suppose that there are a finite number of channels. Then, for any process \( P \) and number \( k < \omega \): (1) \( C^k(P) \) is a finite formula, (2) \( \{ C^k(P) \mid P \text{ process} \} \) is a finite set (up to identification of \( \psi \) with \( \psi \land \psi \)).

**Proof.** Prove 1 and 2 together by induction on \( k \) and \( P \) order. \( \qed \)

**Theorem 4.3 (Completeness for closed processes)** If \( \models P : \psi \) then \( \vdash P : \psi \).

**Proof.** By induction on the following lexicographic order:

\[ (\|\psi\|, \text{order}(\psi), \text{struct}(P), \text{struct}(\psi)) \]

One proceeds by case analysis on the structure of \( P \) and \( \psi \).

1. \( \psi ::= \top \mid \bot \): Direct.
2. \( \psi ::= \psi_1 \land \psi_2 \mid \psi_1 \lor \psi_2 \): \( \text{struct}(\psi) \) decreases.
3. \( \psi ::= \psi_1 \Rightarrow \psi_2 \): \( \text{order}(\psi) \) decreases, use \( \omega \)-rule.
4. \( \psi ::= (\alpha)\psi \): We analyse the structure of the process \( P \) (which has order 0).

   (a) \( P ::= 0 \): Direct.
   (b) \( P ::= c!P.P \mid c?x.P : \|\psi\| \) decreases.
   (c) \( P ::= P + P : \text{struct}(P) \) decreases.
   (d) \( P ::= P \mid P \): There are two subcases.

   i. \( \psi ::= (\alpha)\psi \): We give this case in some detail.

   - Suppose \( \models P_1 \mid P_2 : (\alpha)\psi \) because \( P_1 \xrightarrow{\alpha} P'_1 \) and \( \models P'_1 \mid P_2 : \psi \).

   Let \( k = \|\psi\| \) and \( \psi_1 = C^k(P'_1) \). We know: \( \models P'_1 : \psi_1 \). Hence:

   \( \models P_1 : (\alpha)\psi_1 \). We can conclude \( \vdash P_1 : (\alpha)\psi_1 \), by ind. hyp. on \( P \).

   Next we show: \( z_1 : \psi_1 \models z_1 : P'_1 : \psi \). Suppose \( \models P''_1 : \psi_1 \), then \( P''_1 \approx_k P'_1 \), which implies \( P''_1 \mid P_2 \approx_k P'_1 \mid P_2 \). Conclude:
$\models P''_1 \mid P_2 : \psi$. By induction on $|\psi|$ we have: $\vdash P''_1 \mid P_2 : \psi$. By the $\omega$-rule and PAR-DIA we prove: $\vdash P_1 \mid P_2 : (\alpha)\psi$.

- Otherwise suppose $\alpha \equiv \tau$ and $\models P_1 \mid P_2 : (\tau)\psi$ because $P_1 \xrightarrow{d!} P''_1[f], \ P_2 \xrightarrow{d?} P'_2[x]$, and $\models [P''_1[x]/f][P'_2] : \psi$. Let $k = |\psi|, \ \psi_1 = C^k(P''_1[f]), \ \text{and} \ \psi_2 = C^k(P'_2[x])$. Clearly: $\models P_1 : (d!)\psi_1$ and $\models P_2 : (d?)\psi_2$. Conclude: $\vdash P_1 : (d!)\psi_1$ and $\vdash P_2 : (d?)\psi_2$, by ind. hyp. on $P$.

It remains to show $z_1 : \psi_1, z_2 : \psi_2 \vdash z_1 z_2 : \psi$. Apply again the logical characterization of the $\sim_k$ relation. Then apply twice the $\omega$ rule to get: $z_1 : \psi_1, z_2 : \psi_2 \vdash z_1 z_2 : \psi$. Conclude $\vdash P_1 \mid P_2 : (\tau)\psi$, by PAR-DIA-$\tau$.

ii. $\psi ::= [\alpha]\psi$: This behaves as the previous case w.r.t. the induction hypothesis. We only consider the case: $\models P_1 \mid P_2 : [\tau]\psi$. Let: $|\psi| = k, \psi_i = \land P_{1,i} C^k(P_i'), \psi_{i,d} = \land P_{2,i} C^k(P_i')$, and $\psi_{i,d} = \land P_{2,i} C^k(P_i')$ (for $i = 1, 2$, and any channel $d$). Clearly: $\models P_i : \psi_i$.

By ind. hyp. on $|\psi|$ follows: $\vdash P_i : \psi_i$, ($i = 1, 2$).

Next show: $x_1 : \psi_1 \vdash x_1 \mid P_2 : \psi$, using the logical characterization of $\sim_k$. By induction on $|\psi|$ and $\omega$-rule follows: $x_1 : \psi_1 \vdash x_1 \mid P_2 : \psi$.

A symmetric argument gives: $x_2 : \psi_2 \vdash P_1 \mid x_2 : \psi$.

Finally show, e.g., $z_1 : \psi_{1,d}, z_2 : \psi_{2,d} \vdash z_2 z_1 : \psi$, by logical characterization of $\sim_k$. By induction on $|\psi|$ and $\omega$-rule follows: $z_1 : \psi_{1,d}, z_2 : \psi_{2,d} \vdash z_2 z_1 : \psi$. \qed

Note that this proof never uses the left introduction rules as hypotheses on the left of the sequent are eliminated by means of the $\omega$-rule. Of course a finitary system would make essential use of the left introduction rules, as in fig. 2.

**Corollary 4.4 (Completeness)** If $\Gamma \models P : \psi$ then $\Gamma \vdash P : \psi$.

**Proof.** Use completeness of the system for closed processes and apply $n$ times the $\omega$-rule, where $n$ is the length of the context $\Gamma$. \qed

## 5 Conclusions and Research Directions

We have given a characterization of Choc bisimulation by means of a Hennessy-Milner logic and developed a related sound and complete proof system for a fragment of the calculus.

We believe that our approach is sufficiently general to be applied to more complex higher-order calculi such as CML and Facile. However in these cases the modal operators for process/function transmission need to be combined with modal operators which specify, e.g., value and channel transmission. The formalization of this system is a non-trivial task.
Probably the most interesting problem which remains to be settled is that of developing a proof system which can handle restriction. In order to appreciate the difficulties, one may try to develop rules to prove the following valid fact, where \( \text{Nil} \) is a formula stating that a process can do no action:

\[
\nu a. (b!(a!x.0).a?x.0) : \langle a! \rangle (\text{Nil} \Rightarrow \text{Nil}) \Rightarrow \text{Nil}
\]

On a more speculative level we would like to understand the relationships between specifications and types for higher-order process calculi. For instance, type systems for concurrent applicative languages such as CML and Facile have been recently proposed [14, 21] which partially describe the input-output behaviour of a program. Is there a connection between these types and specifications as presented here?

Acknowledgments. We are indebted to Lone Leth, Sanjiva Prasad, and Bent Thomsen for several discussions on the topics presented here.

References


