Linear Uniform Receptiveness in a Pi-Calculus with Location Failures

Peter Finderup  Hans Hüttel  Jakob Svane Knudsen  Johannes Garm Nielsen

Department of Computer Science, Aalborg University, Denmark
Email: {finde, hans, jak, johsgn}@cs.aau.dk

Abstract

The notion of receptiveness arises in the π-calculus as a guarantee of determinacy in the behaviour of callable entities and was first investigated by Sangiorgi.

The DrF process calculus, introduced by Francalanza and Hennessy, extends the π-calculus with located processes and location and link failures. In this paper we extend the notion of receptiveness to DrF and give sound characterizations of the property of linear uniform receptiveness in DrF in the form of two type systems.

Our first type system ensures receptiveness, the property that no pending output will ever be left unattended. We achieve this by ensuring linearity and by ensuring that the input and output remain at the same location, such that location failure will effectively remove either both or none. Our second type system allows for migration but ensures that input capabilities remain within locations which are hidden from the context and thus not subject to failures.

Keywords: π-calculus, type systems, location failures.

1 Introduction

The π-calculus has been used to describe and reason about many different computational phenomena. In particular, it has been used to describe callable entities such as functions (Milner 1990), objects (Kleist & Sangiorgi 2002) and higher-order communications (Sangiorgi 1996). In these settings it is important that the behaviour of a call is determinate. This can be ensured if a name n which refers to a callable entity is uniform, so an input on channel n always leads to the same continuation and receptive, so no calls will be left unattended.

In (Sangiorgi 1999) Sangiorgi defines a number of type systems that give sound characterizations of versions of this property of uniform receptiveness. Further, he shows that uniform receptiveness is important from the point of view of process calculus theory since this assumption makes it easier to reason about behavioural equivalences: for instance, proofs of transformations that introduce parallelism can be simplified and the proof of the correctness of an optimized translation of higher-order process calculi into the π-calculus can be simplified.

The notion of uniform receptiveness is also important from the point of view of implementation, since uniform receptive names can be implemented more efficiently than arbitrary names. An example application of this is found in the Pict language (Pierce & Turner 2000) whose compiler is able to recognize receptive names and use this knowledge to perform code optimizations based on Sangiorgi’s results.

The problem of uniform receptiveness also arises in the setting of distributed systems with localities. In (Hennessy & Riely 2002), the π-calculus is extended to deal with such features and the result is the distributive π-calculus Dr. Later, Hennessy and Francalanza have studied a further extension, DrF, which also allows one to extend describe link and location failures (Francalanza & Hennessy 2005).

In this paper we examine how and to which extent results concerning uniform receptiveness carry over to the rich setting of the DrF-calculus, where failures can occur.

Consider as a first example the scenario where a number of user processes located at location l wish to make a remote procedure call to a server process, located at location k. To obtain transparency, an interface process, which has knowledge of the server location, is used to relay the request. In DrF this example could be described as follows.

\[ U(n, D) \equiv \lambda (n) \cdot b!?(m : val).D(m) \]
\[ H \equiv \ast a?(p : val).go \ k \ e!\langle p \rangle.D(q : val).go \ l \ b\langle q \rangle \]
\[ S \equiv \ast c?(r : val).d!\langle f(r) \rangle \]
\[ N \equiv \lambda [U(i, P)] | [U(j, Q)] | [l[H]] | k[S] \]

The system is defined as N, where P is some process that needs the name f(i) and Q is some process that needs f(j).

A user process, U, sends a value n over the channel a and waits to receive another value over channel b. The received value is then used as an argument in the subsequent process D. The interface process, H, is a replicated input process. Once a replication has been unfolded, the replicated section can receive a value over the channel a and then migrate to the location k. Once there, it can emit the received value on the channel c. Then it will wait for a response in the form of a value sent over the channel d. When the value has arrived, the process migrates back to D and emits the received value on channel b. The server process S is also replicated. A replication of S will result in a process which awaits a value sent over the channel c and, once having received a value over c, sends the result of a function f(r) back over d, where r is the value it received over c.

Unfortunately, since the two user processes use the same channels for communication with H, the return values may get mixed up. That is, the process that has sent i may end up receiving f(j) instead of f(i). This is due to nondeterminism; the following

Copyright ©2010, Australian Computer Society, Inc. This paper appeared at the 16th CATS Symposium (Computing: the Australasian Theory Symposium), Brisbane, Australia. Conference in Research and Practice in Information Technology (CRPIT), Vol. 109, A. Potamian and T. Viglas, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.
is valid in the DπF transition system, where the network representation D describes the link structure and available locations:

\[ \Delta \vdash N \xrightarrow{a} \Delta P \]

\[ \begin{align*}
\Delta \vdash N & \quad \Rightarrow \\
& \quad l[x](m : \text{val}), P(m) \\
& \quad l[x'](m : \text{val}), Q(m) \\
& \quad \text{l}[]H \mid k[S]
\end{align*} \]

The resulting configuration can now reduce in a multitude of ways by input on the channel b. In other words, b is not uniform. A uniform channel is one such that input on the channel will always result in the same continuation.

We can easily ensure that P receives \( f(i) \) and Q receives \( f(j) \) by introducing private channels denoted \( x, y, z \) and \( w \) in the following.

\[ \begin{align*}
U(n, D) & \triangleq (x \vdash \text{ch(val)})(a \vdash (n, x), x'i(m : \text{val}, D(m))) \\
H & \triangleq *a?(p \vdash \text{val}, y \vdash \text{ch(val)}); \text{go } k. \\
S & \triangleq \text{*c?(r \vdash \text{val}, w \vdash \text{ch(val)}); w!(f(r))} \\
N & \triangleq \text{l[U(i, P)] | l[U(j, Q)] | l[H] | k[S]}
\end{align*} \] (1)

The latter example takes advantage of the fact that a channel which is only used once for input is always uniform. Channels which are only used once are called linear.

It would be desirable to ensure that once a user process sends a request for a value, it would be guaranteed to wait for the value on the new private channel \( x \), regardless of any actions made by the surrounding context. That is, it would be convenient if the private channels \( x \) and \( y \) were receptive at \( l \). A channel is receptive on a location if output can be delivered there, as long as there is a possibility for output on the channel at that location.

Now consider the problems introduced by location failures. The following transitions are possible under the labelled transition system of DπF.

\[ \begin{align*}
\Delta \vdash N & \xrightarrow{*} \\
\Delta & \vdash \\
& \quad l[x](m : \text{val}), P(m) \\
& \quad l[x'](m : \text{val}), Q(m) \\
& \quad k[\text{go } l.x!(f(i))] \\
& \quad k[\text{go } l.x'!(f(j))] \\
& \quad [l[H] | k[S]]
\end{align*} \]

Because the location \( l \) has been killed, no further reductions can take place on \( l \), so the process in \( l[x](m : \text{val}), P(m) \) will never be able to receive the value \( f(i) \). Additionally the context is capable of performing a break action which means that the link between two locations will disappear; this has similar consequences for receptiveness.

These problems form the focus of our paper. We show how to adapt the type system of Sangiorgi (Sangiorgi 1999) to ensure linear receptiveness in the DπF-calculus. In (Amadio et al. 2003), Amadio, Lhoussaine and Bondol proposed a method for obtaining receptiveness in a distributed π-calculus. However, their process calculus pre-dates the work on DπF and consequently does not address the notion of failures.

The remainder of our paper is organized as follows: Section 2 introduces the syntax and semantics of the DπF language. In Section 3 we present linear uniform receptiveness based on (Sangiorgi 1999). Then, in Section 4 we adapt the method and introduce our first type system which we prove is sound, i.e. that in a well typed DπF system all channels of a certain type are linear uniform receptive. Finally, in Section 5 we revise the type system to accommodate another interpretation of the soundness result.

2 A distributed pi-calculus with failures

Our point of departure is the DπF-calculus introduced by Francalanza and Hennessy in (Francalanza & Hennessy 2005) which we present in this section.

To simplify our presentation of the type system, we use a monadic version of the DπF-calculus. However, the examples that follow employ a polyadic version of the calculus. It is straightforward to transfer the results of our paper to this setting.

We add to the location types of DπF the channel types presented in Section 2.2 and omit channel names from the network representation, since we can keep track of these names in our type environments. This leaves the network representation to represent the state of the network solely.

2.1 Syntax

We assume a countably infinite set of basic names \( N = \{a, b, \ldots, i, j, k, \ldots, u, v, \ldots, x, y, z, \ldots\} \). Processes \( P, Q \) in DπF are located at named locations, and a system of located processes can be described using the system constructs. We show the abstract syntax of processes and systems below; \( T \) ranges over the set of types, defined in Section 2.2.

\[ \begin{align*}
\pi, \nu \quad &::= \quad 0 & \text{Process termination} \\
\text{go } u.P & \quad \text{Migration} \\
\text{u!}(v : T).P & \quad \text{Input} \\
\text{u?(v).P} & \quad \text{Output} \\
\text{kill } P & \quad \text{Location termination} \\
\text{break } u & \quad \text{Link termination} \\
\text{P | Q} & \quad \text{Parallel composition} \\
\nu(v : T).P & \quad \text{Name restriction} \\
\text{*u?}(v : T).P & \quad \text{Replicated input} \\
\text{u = v}.P, Q & \quad \text{Match} \\
\text{u P, Q} & \quad \text{Ping} \\
\text{M, N} & \quad \text{Nil system} \\
\text{M} | \text{N} & \quad \text{Parallel composition} \\
\text{(u v : T) N} & \quad \text{Name restriction} \\
\text{k[P]} & \quad \text{Located process}
\end{align*} \]

We shall always omit trailing occurrences of 0. Names used in a process or system are either bound or free. For any process or system \( P \), we denote the sets of bound and names in \( P \) by \( \text{bn}(P) \) and \( \text{fn}(P) \), respectively. These sets are defined in the usual way, see e.g. (Hennessy & Riely 2002).

We will use the standard notation of capture-avoiding name substitution and let \( \text{P}[x/y] \) denote the process \( P \) where all occurrences of the free name \( y \) have been replaced with the name \( x \).

2.2 Types

Our language of types extends that of Milner’s sorts for the π-calculus (Milner 1999).

A normal channel name used for communicating names of type \( T \) has the channel type \( T = \text{ch}(T) \). In later parts of our paper we shall make a further distinction between regular channel types and linear
channel types. A name of linear channel type is used to communicate names of type $T$ at the location $l$ and has the type $\text{ch}(T)$. For a channel type $T_c = \text{ch}(T)$, we write $T_c \leq \text{ch} = \text{ch}(T)$ if the object type $T$ is not important and for simplicity write $T_c \leq \text{ch} = \text{ch}(T)$ if neither the location nor the object type is important.

A location name $k$ has location type $L = \text{loc}(C)$ if it is used as a location, where $S$ denotes the status of the location as either a (alive) or d (dead), and $C$ is the set of locations which are connected to $k$ in our network model. We write $L \leq \text{loc}$ if the status and connections of the location are not important.

We also use the basic type val for names which are neither used as channels or locations and use a set of types $T$ containing all name types. The syntax of the types used in this paper is

$$T ::= \text{ch}(T) \quad \text{Basic channel type}$$
$$\text{ch}_l(T) \quad \text{Linear channel type}$$
$$\text{val} \quad \text{Basic type}$$
$$\text{loc}(C) \quad \text{Location type}$$

We define two operations on types.

$$T - l = \begin{cases} \text{loc}(C) \setminus \{l\} & \text{if } T = \text{loc}(C) \\ T & \text{otherwise} \end{cases}$$

and

$$T + l = \begin{cases} \text{loc}(C) \cup \{l\} & \text{if } T = \text{loc}(C) \\ T & \text{otherwise} \end{cases}$$

A type environment $\Gamma \subseteq N \times T$ is a finite set of pairs $(n,T)$. We denote a type annotation in $\Gamma$ as $n : T$. We demand that $\Gamma$ behaves as a function with finite type environment where $A$ represents a network and information about links between them.

2.3 Semantics

Our labelled transition semantics is adapted from (Francalanza & Hennessy 2005). It describes internal reductions of a system as well as the effect a context can have on the system, including the ability of an observer to kill a location or break a link.

The networks that DrF systems describe consist of locations connected by links. A network representation $\Delta$ contains the name of each location in the network and information about links between them. A link between two locations $l,k$ is denoted $l \leftrightarrow k$. Connections are bi-directional, so the link relation is symmetric.

Given a location $k$ and a set of locations $L = \{l_1, \ldots, l_n\}$, we will sometimes write $k \leftrightarrow L$ for the set of links $\{l \leftrightarrow k \mid \text{set} \in L\}$. We let $\text{dom}(l \leftrightarrow k)$ denote the set of names $(l,k)$, and expand this notation to sets of links in the natural way such that $\text{dom}(l_1 \leftrightarrow k_1, \ldots, l_n \leftrightarrow k_n) = \bigcup_{i=1}^n \text{dom}(l_i \leftrightarrow k_i)$.

Definition 1 (Network representation) A network representation $\Delta$ is a triple $(\Delta_N, \Delta_O, \Delta_H)$ where

- $\Delta_N \subseteq N$ is the set of names of locations in the network.
- $\Delta_O$ is the set of live locations on the network that are observable by a context.
- $\Delta_H$ is the set of live links that are hidden from the context.

such that $\text{dom}(\Delta_O \cup \Delta_H) \subseteq \Delta_N$ and $\Delta_O \cap \Delta_H = \emptyset$.

Note that the status of links and locations on the network is captured by this representation simply by listing the live links, since $l \leftrightarrow k \in \Delta_O \cup \Delta_H$ indicates that $l$ and $k$ are alive. An isolated live location $l$ will be represented by a loop $l \leftrightarrow l$, which also indicates that migration from a location to itself is always possible.

Definition 2 (Network predicates) We use the following predicates to describe the state of the network:

$$\Delta \vdash l \quad \text{if } l \in \text{dom}(\Delta_O \cup \Delta_H)$$

$$\Delta \vdash \text{obs } l \quad \text{if } l \in \text{dom}(\Delta_O)$$

$$\Delta \vdash \text{loc}(C) \quad \text{if } S = a \text{ and } C \subseteq \text{dom}(\Delta_O \cup \Delta_H)$$

$$\Delta \vdash \text{obs } \text{loc}(C) \quad \text{if } S = a \text{ and } C \subseteq \text{dom}(\Delta_O)$$

$$\Delta \vdash l \leftrightarrow k \quad \text{if } l \leftrightarrow k \in \Delta_O \cup \Delta_H$$

$$\Delta \vdash \text{obs } l \leftrightarrow k \quad \text{if } l \leftrightarrow k \in \Delta_O$$

$$\Delta \vdash l \leftrightarrow k \quad \text{if } l_1, \ldots, l_n \text{ exist such that }$$

$$\Delta \vdash l \leftrightarrow l_1, \Delta \vdash l \leftrightarrow l_{i+1} \quad \text{for } i = 1 \ldots n - 1 \text{ and }$$

$$\Delta \vdash l \leftrightarrow n \quad \text{if } \text{for any } l \in L, \Delta \vdash l \leftrightarrow k$$

Definition 3 (Network updates) We use the following notation to describe updates to a network state.

$$\Delta + n : T = \begin{cases} \Delta & \text{if } T \leq \text{loc} \\ \langle \Delta_N \cup \{n\}, \Delta_O, \Delta_H \rangle & \text{if } T = \text{loc}(C) \text{ and } C \cap \text{dom}(\Delta_O) = \emptyset \\ \langle \Delta_N \cup \{n\}, \Delta_O \cup (n \leftrightarrow C) \cup D, \Delta_H \setminus D \rangle & \text{if } T = \text{loc}(C) \text{ and } C \cap \text{dom}(\Delta_O) \neq \emptyset \text{ where } D = \{l \leftrightarrow k \in \Delta_H \mid \Delta \vdash C \leftrightarrow l\} \\ \Delta - l = \langle \Delta_N \setminus \{l\}, \Delta_O \setminus l \leftrightarrow \Delta_N, \Delta_H \setminus l \leftrightarrow \Delta_N \rangle \\ \Delta - l \leftrightarrow k = \langle \Delta_N, \Delta_O \setminus \{l \leftrightarrow k\}, \Delta_H \setminus \{l \leftrightarrow k\} \rangle \\ \top \Delta = \langle \Delta_N, \Delta_O \cup \Delta_H, \emptyset \rangle \end{cases}$$

In the labelled semantics, a configuration $C$ is a pair $\Delta \triangleright N$.

Definition 4 (Labelled semantics) The labelled transition system of DrF has transitions of the form

$$C \xrightarrow{\mu \#} C'$$

where $C$ and $C'$ are configurations and $\mu$ is a label of one of the following forms.

- $\tau$ - An internal (unobservable) action.
- $(a : U) a @ l ? (a : T)$ - An observable input on channel $a$ at location $l$.
- $(a : U) a @ l ! (a : T)$ - An observable output on channel $a$ at location $l$.
- $\text{kill } l$ - The context kills the location $l$.
- $\text{break } l \leftrightarrow k$ - The context breaks the link between the locations $l$ and $k$.
In the input case, \( n : U \) represents the new name that the context introduces and in the output case it represents the name that is made known to the context.

The labelled transition system is the least relation up to structural equivalences (Table 1) closed under the rules in Tables 2, 3 and 4.

We will write \( C \xrightarrow{\tau} C' \) to represent the reflexive and transitive closure of the labelled transition relation. We write \( C \Rightarrow C' \) if \( C \xrightarrow{\tau} * C' \) by a sequence of \( \tau \)-transitions. Finally, we let \( C \xrightarrow{\mu} C \) if \( C \Rightarrow \mu C' \).

3 Linear Uniform Receptiveness

In a D\( \pi \)F system a free name is receptive if the network can receive input over that name as long as output over the name can occur. A pleasant consequence of receptiveness is that no output on a receptive channel will ever be left unattended. Uniform receptiveness captures a notion of determinacy: A channel name is uniform, if input on this channel is always handled in the same way.

Sangiorgi (Sangiorgi 1999) defines two forms of receptiveness which guarantee uniformity in the \( \pi \)-calculus: \( \omega \)-receptiveness, where receptive names can only occur in input-replicated form and linear receptiveness, where receptive names must be linear. In this paper we focus on the latter.

Since communication in D\( \pi \)F happens within a location, receptiveness is also location-specific, and a name may very well be receptive at one location but not at another. As an example, consider

\[
P = a?((x : \text{val}), b?(y : \text{val})) \quad Q = b?(y : \text{val})\]

and the system \( l[P] \mid k[Q] \) : \( a \) is receptive at the location \( l \) and \( b \) is receptive at the location \( k \), but \( b \) is not receptive at the location \( l \). To formalize the localized capabilities of D\( \pi \)F systems we use the notion of barbs.

Definition 5 (Barbs) Given D\( \pi \)F system \( N \), channel name \( a \), network representation \( \Delta \), and location name \( l \),

- \( N \) has a strong output barb on \( a@l \), written \( N \downarrow a@l \), if \( \Delta \triangleright N \xrightarrow{\Delta'(x:T)} N \) or \( \Delta \triangleright N \).
- \( N \) has a weak output barb on \( a@l \), written \( N \downarrow a@l \), if \( \Delta \triangleright N \xrightarrow{\Delta'(x:T)} N' \) such that \( N' \downarrow a@l \) for some network representation \( \Delta' \).
- \( N \) has a strong input barb on \( a@l \), written \( N \uparrow a?@l \), if \( \Delta \triangleright N \xrightarrow{\Delta'(x:T)} \) for some name \( x \) and type \( T \).
- \( N \) has a weak input barb on \( a@l \), written \( N \uparrow a?@l \), if \( \Delta \triangleright N \xrightarrow{\Delta'(x:T)} N' \) such that \( N' \uparrow a?@l \) for some network representation \( \Delta' \), name \( x \) and type \( T \).

With the definition of barbs we can now define the notion of receptiveness.

Definition 6 (Receptive names) Given a network representation \( \Delta \), a channel name \( a \) is receptive on a location \( l \) in a system \( N \), if whenever \( N \downarrow a@l \), then also \( N \downarrow a?@l \).

In the example (1) in Section 1, we can convince ourselves that all the channel names are receptive in the system \( N \), if we disregard the possibility of the context interfering. We can even convince ourselves that the names \( x \), \( y \) and \( z \) are linear receptive, and therefore also uniform receptive.

The property of receptiveness has different consequences in D\( \pi \), which does not model arbitrary failures, and D\( \pi \)F which does.

Definition 7 (Communication over a) Let \( \Delta \) be a network representation \( \Delta \). Then \( N \) has a communication over channel \( a \) if \( \Delta \triangleright N \xrightarrow{\Delta'} N' \) for some \( \Delta' \triangleright N' \), where \( a \) is free in \( N \) and the proof of the transition involves an application of (L-COMM-1) or (L-COMM-2) with \( x \) instantiated to \( a \). We then write \( \Delta \triangleright N \xrightarrow{\pi \text{ with } a} \).

Lemma 1 (Receptiveness in D\( \pi \)) Let a be a linear name in a D\( \pi \) system \( N \) and let \( \Delta \) be a network representation \( \Delta \). If \( a \) is receptive, and \( N \downarrow a@l \) then we have \( \Delta \triangleright N \xrightarrow{\Delta' \triangleright N'} \text{ with } a \) for some \( N' \).

The proof of Lemma 1 is a straightforward induction on the number of transitions it takes for the weak output barb to become a strong output barb. However, this result does not hold for the D\( \pi \)F-calculus due to the presence of location and link failures.

4 Settled Obligations

As we saw in the introduction, since processes can migrate but locations may fail, the input on a receptive name \( n \) can be caught at a dead site while the output lives on. This is possible, since observable locations can be killed at any time, and it implies that \( n \) cannot be receptive.

An obvious way to circumvent this problem is to ensure that any process which can perform an input or output on a receptive name cannot migrate. This means that in order to be receptive, such processes must always remain at the same location and, if this location were to be killed, both the input- and output-barb would disappear and the statement in Definition 6 would be satisfied.

We will use a notion of obligations on linear names to ensure that processes keep their promise of either taking an input or giving an output on these names. An obligation is a located name of the form \( a@l \). Like Sangiorgi (Sangiorgi 1999), we consider two sets of these: a set of input obligations and a set of output obligations.

Our type system has two typing judgments:

- \( \Gamma, I, O \vdash T \), defined in Tables 5 and 6, states that process \( P \) is well-typed when placed at location \( l \), given type environment \( \Gamma \), input obligations \( I \) and output obligations \( O \).
- \( \Gamma, I, O \vdash N \), defined in Table 7, states that network \( N \) is well-typed, given type environment \( \Gamma \), input obligations \( I \) and output obligations \( O \).

The intended interpretation of a judgment \( \Gamma, I + a@l, O + b@k \vdash N \) is that the system \( N \) can immediately receive input on channel \( a \) at location \( l \) exactly once, may give output on the channel \( b \) at location \( k \) at most once and will never use names of type ch1 in \( \Gamma \) in a nonlinear or nonreceptive way.

The rules in Tables 5, 6 and 7 are for the most part adaptations of Sangiorgi’s type rules from (Sangiorgi 1999) with additional rules for the constructs specific to D\( \pi \)F. (wp-go), or rather the lack of a rule which allows processes with input- or output obligations to
\[
\begin{align*}
(s\text{-comm}) & \quad N \mid M \equiv M \mid N \\
(s\text{-assoc}) & \quad (N \mid M) \mid M' \equiv N \mid (M \mid M') \\
(s\text{-extr}) & \quad M \mid (vn \mid T)N \equiv (vn \mid T)(M \mid N) \quad \text{if } n \notin \text{fn}(M) \\
(s\text{-flip}) & \quad (vn \mid T)(vn \mid U)N \equiv (vn \mid U)(vn \mid T)N \quad \text{if } n \notin \text{fn}(U) \\
(s\text{-flip-2}) & \quad (vn \mid T)(vn \mid U)N \equiv (vn \mid U-n)(vn \mid T+m)N \quad \text{if } n \in \text{fn}(U) \text{ and } U, T \leq 10c \\
(s\text{-inact}) & \quad (vn \mid T)N \equiv N \quad \text{if } n \notin \text{fn}(N)
\end{align*}
\]

Table 1: The structural rules of DrF networks

\[
\begin{align*}
(l\text{-str}) & \quad N \equiv M \quad \Delta \triangleright M \xrightarrow{\mu} \Delta' \triangleright M' \quad M' \equiv N' \\
(l\text{-rest}) & \quad \Delta + n : T \triangleright N \xrightarrow{\mu} \Delta' + n : U \triangleright M \\
 & \quad \Delta \triangleright (vn \mid T)N \xrightarrow{\mu} \Delta' \triangleright (vn \mid U)M \\
(l\text{-par-1}) & \quad \Delta \triangleright N \xrightarrow{\mu} \Delta' \triangleright N' \\
 & \quad \Delta \triangleright N \mid M \xrightarrow{\mu} \Delta' \triangleright N' \mid M \\
(l\text{-par-2}) & \quad \Delta \triangleright M \xrightarrow{\mu} \Delta' \triangleright M' \\
 & \quad \Delta \triangleright N \mid M \xrightarrow{\mu} \Delta' \triangleright N \mid M'
\end{align*}
\]

Table 2: Labelled transitions for arbitrary actions.

Assuming \( \Delta \vdash_{\text{obs}} l \)

\[
\begin{align*}
(l\text{-obs-kill}) & \quad \Delta \triangleright N \xrightarrow{\text{kill } l} \Delta - l \triangleright N \\
(l\text{-obs-break}) & \quad \Delta \triangleright N \xrightarrow{\text{break } l \leftarrow k} \Delta - l \leftarrow k \triangleright N \\
(l\text{-obs-open}) & \quad \Delta + n : T \triangleright N \xrightarrow{\text{kill } l \mid T} \Delta' \triangleright N' \\
 & \quad \Delta + n : T \triangleright \text{dom}(\Delta_{obs} \cup \Delta_{H}) \\
 & \quad \Delta \triangleright (vn \mid T)N \xrightarrow{vn \mid U} \Delta' \triangleright N' \\
(l\text{-obs-weak}) & \quad \Delta + n : T \triangleright N \xrightarrow{\text{kill } l \mid T} \Delta' \triangleright N' \\
 & \quad \Delta + n : T \vdash_{\text{obs}} T \\
(l\text{-obs-typ}) & \quad \Delta + k : T \triangleright N \xrightarrow{(vn' \mid U) \text{kill } l \mid T} \Delta + n : V + k : T' \triangleright N' \\
 & \quad k \in \text{fn}(V') \\
 & \quad \Delta \triangleright (vk \mid T)N \xrightarrow{vn' \mid U} \Delta + n : V \triangleright (vk \mid T')N' \\
(l\text{-obs-in}) & \quad \Delta \triangleright l[a?x \mid T].P \xrightarrow{a?x \mid T} \Delta \triangleright l[P[y/x]] \\
(l\text{-obs-out}) & \quad \Delta \triangleright l[a!x \mid T].P \xrightarrow{a!x \mid T} \Delta \triangleright l[P]
\end{align*}
\]

where \( T \sqcup \text{dom}(\Delta_{obs} \cup \Delta_{H}) \equiv \begin{cases} 10c_S[D] & \text{where } D = C \cap \text{dom}(\Delta_{obs} \cup \Delta_{H}) \\
T & \text{if } T = 10c_S[C] \\
otherwise \end{cases} \)

Table 3: The labelled transitions for observer actions on DrF configurations.
Assuming $\Delta \vdash l$

(L-GO)  
$$\Delta \vdash l \leftrightarrow k$$
$$\Delta \triangleright l[go \ k.P] \xrightarrow{\tau} \Delta \triangleright k[P]$$

(L-NGO)  
$$\Delta \nvdash l \leftrightarrow k$$
$$\Delta \triangleright l[go \ k.P] \xrightarrow{\tau} \Delta \triangleright \lbrack 0 \rbrack$$

(L-COMM-1)  
$$\uparrow \Delta \triangleright N \xrightarrow{(n : T) \eta \setminus T(n : T)} \Delta' \triangleright N'$$
$$\uparrow \Delta \triangleright M \xrightarrow{(n : T) \eta \setminus T(n : T)} \Delta'' \triangleright M'$$

(L-COMM-2)  
$$\uparrow \Delta \triangleright N \xrightarrow{(n : T) \eta \setminus T(n : T)} \Delta' \triangleright N'$$
$$\Delta \triangleright N' | M \xrightarrow{\tau} \Delta \triangleright N' | M'$$

(L-REP)  
$$\Delta \triangleright l[\ast a?(x : T).P] \xrightarrow{\tau} \Delta \triangleright l[\ast a?(x : T).P | \ast a?(x : T).P]$$

(L-FORK)  
$$\Delta \triangleright l[P | Q] \xrightarrow{\tau} \Delta \triangleright l[P] | l[Q]$$

(L-MATCH)  
$$\Delta \triangleright l[u = u]P, Q \xrightarrow{\tau} \Delta \triangleright l[P]$$

(L-MISM)  
$$u \neq v$$
$$\Delta \triangleright l[u = v]P, Q \xrightarrow{\tau} \Delta \triangleright l[Q]$$

(L-PING)  
$$\Delta \triangleright l[\circ k]P, Q \xrightarrow{\tau} \Delta \triangleright l[P]$$

(L-MISP)  
$$\Delta \nvdash l[\circ k]P, Q \xrightarrow{\tau} \Delta \triangleright l[Q]$$

(L-KILL)  
$$\Delta \triangleright l[k111] \xrightarrow{\tau} \Delta \triangleright \lbrack 0 \rbrack$$

(L-BREAK)  
$$\Delta \triangleright l \leftrightarrow k$$
$$\Delta \triangleright l[\text{break} k] \xrightarrow{\tau} \Delta \triangleright l \leftrightarrow k \triangleright \lbrack 0 \rbrack$$

(L-NEW)  
$$\Delta \triangleright l[(\nu u : T)P] \xrightarrow{\tau} \Delta \triangleright (\nu u : T)l[P]$$

(L-NEWL)  
$$\Delta \triangleright l[(\nu k : \text{loc}_S[C])P] \xrightarrow{\tau} \Delta \triangleright (\nu k : \text{loc}_S[D])l[P]$$

Table 4: Labelled transitions for internal actions of DπF configurations.
migrate, is notable. Additionally, it is worth noting that the type of a linear channel carries a location name that denotes the location where input can be delivered on that channel. This is necessary in order to give processes the correct obligation upon receiving linear channels, as shown with the rule \[(\text{WN-INPUT-L})\].

We return to the example from Section 1 to see the effect of these rules. We modify the example slightly, giving the names \(x, y\) and \(z\) the new linear channel type, as it is our intention to enforce receiverness on these channels.

\[
\begin{align*}
U(n, D) &\equiv (\nu x. \chi ![(val)] !a(n, x) !x? (m : val)) \cdot D(m) \\
H &\equiv a^? (p : val, y : \chi ![val]) . \text{go} \ k. \ (\nu z. \chi ![(val)] (I(p, z)) . \text{go} \ l(y(q))) \\
S &\equiv \text{loc} !r : val, w : \chi ![val]) . \text{go} \ l(f(r)) \\
N &\equiv \text{loc} !U(i, j) | \text{loc} !U(j, i) | \text{loc} !H | k[S]
\end{align*}
\]

The system \(N\) defined above is not well typed for any \(\Gamma, I, O\). For instance, the system creates linear names but cannot immediately afterwards receive on them.

Consider instead the altered example shown below. The entirety of the changes lies in \(U\) and \(H\), which now creates linear names and then in parallel listen on the name while another process will relay it. Using this definition, and the type environment

\[
\Gamma = \{a : \text{ch}(val, \chi ![val]), c : \text{ch}(val, \chi ![val]), l : \text{loc}, k : \text{loc}, i : \text{val}, j : \text{val}\}
\]

we can verify that \(\Gamma, \emptyset, \emptyset \vdash N\).

\[
\begin{align*}
U(n, D) &\equiv (\nu x. \chi ![(val)] !a(n, x) !x? (m : val)) \cdot D(m) \\
H &\equiv a^? (p : val, y : \chi ![val]) . \text{go} \ k. \ (\nu z. \chi ![(val)] (I(p, z)) . \text{go} \ l(y(q))) \\
S &\equiv \text{loc} !r : val, w : \chi ![val]) . \text{go} \ l(f(r)) \\
N &\equiv \text{loc} !U(i, j) | \text{loc} !U(j, i) | \text{loc} !H | k[S]
\end{align*}
\]

We can now state the main result about our first type environment. This soundness theorem states that for any labelled network transition, a transition involving a name preserves typability. In case 4, note that the type environment must be extended with the extruded name.

Also note that we need to assume that the context is also well typed, such that the receiverness of our linear channels will not be lost.

**Theorem 1 (Soundness)** Given a \(\text{DrF} \) system \(N\), sets of obligations \(I, O\), a type environment \(\Gamma\) and a network representation \(\Delta\), if \(\Gamma, I, O \vdash N\) then

1. If \(\Gamma(x) = \chi ![T]\) and \(x \notin I\) then for all \(T\) such that \(\Delta \vdash N\) \(\Delta \vdash \text{loc} ![T]\) \(\Delta \vdash N\).
2. If \(\Gamma(a) = \text{ch}(T)\) where \(T \leq \chi k\), \(x \notin O\) and \(\Delta \vdash \text{loc} ![T]\) \(\Delta \vdash N\) then \(\Gamma + t : T, I, O + x \notin k \vdash N\).
3. If \(\Gamma(x) = \chi ![T], T \leq \chi\) and \(\Delta \vdash N\) \(\Delta \vdash N\) then \(\Gamma + a : T, I, O \vdash N\).
4. If \(\Gamma(a) \not\subseteq \chi\) and \(\Delta \vdash N\) \(\Delta \vdash \text{loc} ![T]\) \(\Delta \vdash N\).
5. If \(\Delta \vdash N \not\vdash \Delta \vdash N\) then either \(\Gamma, I, O \vdash N\) or there is an \(x \notin I\) such that \(\Gamma, I, O \vdash x \notin I \vdash N\).
6. If \(\Gamma(x) \not\subseteq \chi\) and \(\Gamma(a) \not\subseteq \chi\) and \(\Delta \vdash N\) \(\Delta \vdash \text{loc} ![T]\) \(\Delta \vdash N\) \(\Delta \vdash N\) then \(\Delta \vdash N\).
7. If \(\Delta \vdash N \not\vdash \Delta \vdash N\) then \(\Gamma, I, O \vdash N\).

The theorem is proved by induction on the transition rules in Tables 2, 3 and 4.

Let us relate Theorem 1 to the desired property of linear uniform receiverness. Cases 2, 3, 4 and 6 ensure linear receiverness by ensuring that if \(\Gamma, I, O \vdash N\), then for any name \(x\) where \(\Gamma(x) \leq \chi\) we have

if \(x \notin I\) then \(N \not\vdash x \notin I\)

if \(N \not\vdash x \notin I\) then \(x \notin I\)

Case 1 makes this receiverness uniform and cases 5 and 7 are akin to a standard subject reduction theorem in that they ensure that obligations only disappear because of location failures, this simple interpretation does not hold. Rather, every channel \(a\) of type \(\chi\) is receivable in the sense of Definition 6 and the uniformity expressed by case 1 holds.

**5 Migrating Obligations**

The type system of the previous section will rule out any system with migrating processes with barbs on linear channels, even if the channel is still receivable. This is unnecessarily restrictive in the setting of a distributed calculus. In our second type system we allow migration but control which locations can be killed.

To do this, we introduce a new location type and enforce that input on linear channels must reside on a location of this new type while processes without input on linear names are free to migrate as they please. Since input and output on linear names can thus be separated on the network, we need to ensure that these new locations which house the input cannot be killed.

We call the new location type \(\text{loc} 1\) such that names that are used as locations can have the type \(\text{loc} 1 s[C]\) where \(S\) and \(C\) have the same meaning as in the plain location type \(\text{loc} s[C]\). Further, we redefine the notation \(T \leq \text{loc} 1\) to mean that either \(T = \text{loc} 1 s[C]\) or \(T = \text{loc} 1 s[C]\) for some \(S\) and \(C\).

Finally, we introduce the changes to \(\text{wp-go}\), \(\text{wp-kill}\) and \(\text{wn-input-L}\) shown in Figure 8. These modifications ensure that only locations of type \(\text{loc} 1\) allow for input on linear channels and that they cannot be killed.

Note that we do not require that the composition of the context and a network type gives us a receptive network. Also note that an ill-typed context is always able to kill any location which it can observe. In \(\text{DrF}\) this is captured by the transition rule (\(\text{L-OBS-KILL}\)) of Table 3. To ensure that the context does not kill
Table 5: Type rules for prefixed DπF processes.

Table 6: Type rules for DπF processes that are not prefix processes.
The new output rules take the new location type into account and ensure that outputs of type `loc` cannot happen on locations of type `loc`. Because of this condition locations of type `loc` cannot reveal themselves to the context and thus unintentionally expose a branch of hidden locations. It should be noted that we do not impose any constraints on output if the location is of type `loc` since a location of this type is always hidden. We will now use the new type system on the example (1) of Section 1.

If `l` and `k` are assumed to be of the type `loc` this example will not be well-typed under the new rules, as linear channels are used on locations of type `loc`. In fact the only requirement for this example to be well-typed is that both locations are of the type `loc`. However, the entire system should not be hidden from the context. Instead we modify the example so that values are exchanged at a hidden location `j`.

<table>
<thead>
<tr>
<th>Table 7: Type rules for DrF systems.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \Gamma, I, O \vdash \mu _ \Gamma, \emptyset, O \vdash P ]</td>
</tr>
<tr>
<td>[ \Gamma, \emptyset, O \vdash \mu _ \Gamma \vdash k ]</td>
</tr>
<tr>
<td>[ \Gamma, \emptyset, O \vdash \mu _ \Gamma + (x, T), I, O \vdash N ]</td>
</tr>
<tr>
<td>[ \Gamma, I, O \vdash \mu _ \Gamma + (\mu x : T)N ]</td>
</tr>
<tr>
<td>[ \Gamma, I, O \vdash \mu _ \Gamma \vdash k ]</td>
</tr>
<tr>
<td>[ \Gamma, I, O \vdash \mu _ \Gamma \vdash k ]</td>
</tr>
<tr>
<td>[ \Gamma, I, O \vdash \mu _ \Gamma \vdash k ]</td>
</tr>
<tr>
<td>[ \Gamma, I, O \vdash \mu _ \Gamma \vdash k ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 8: Modified type rules for migration and location termination.</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \Gamma, I, O \vdash \mu _ \Gamma + (\mu x : T)N ]</td>
</tr>
<tr>
<td>[ \Gamma, I, O \vdash \mu _ \Gamma + (\mu x : T)N ]</td>
</tr>
<tr>
<td>[ \Gamma, I, O \vdash \mu _ \Gamma + (\mu x : T)N ]</td>
</tr>
<tr>
<td>[ \Gamma, I, O \vdash \mu _ \Gamma + (\mu x : T)N ]</td>
</tr>
</tbody>
</table>

any location of type `loc`, we must keep such locations hidden. A bound location can become known to the context if it is extruded on an observable channel. However, it is not enough that no output of a name with type `loc` occurs, because of the way in which locations with observable links are added to the network representation as the following example shows.

Assume we have a system `N` and a network representation `Δ` that are defined as follows:

\[ N = (vk : \text{loc}_c\{l\})[l]a[k] \] and

\[ \Delta = \langle \{l\}, \{l \mapsto l\}, \{l \mapsto j\} \rangle \]

By (l-obs-open) and (l-obs-out) the system `N` can make an observable output

\[ \Delta \vdash (vk : \text{loc}_c\{l\})[l]a[k] \]

\[ \Delta + k : \text{loc}_c\{k\} \vdash l[0] \]

Note that using Definition 3 `Δ’ = Δ + k : loc_c\{l\} = \langle \{l\}, \{l \mapsto l, l \mapsto k, l \mapsto j\}, \{\} \rangle`.

Now the previously hidden location `j` is known to the context and thus by (l-obs-kill)

\[ \Delta’ \vdash l[0] \xrightarrow{k|l|j} \Delta’ - j \vdash l[0] \]

Therefore, we need to ensure that output on channels which can carry locations cannot occur on observable locations. This is enforced by the new output rules in Table 9.

The new output rules take the new location type into account and ensure that outputs of type `loc` cannot happen on locations of type `loc`. Because of this condition locations of type `loc` cannot reveal themselves to the context and thus unintentionally expose a branch of hidden locations. It should be noted that we do not impose any constraints on output if the location is of type `loc` since a location of this type is always hidden. We will now use the new type system on the example (1) of Section 1.

\[ (\text{wn-loc}) \]

\[ I \in O \vdash t[\Gamma] \]

\[ I \in O \vdash \text{loc}_c\{l\} \]

\[ I \in O \vdash \text{loc}_c\{l\} \]

\[ I \in O \vdash \text{loc}_c\{l\} \]

\[ I \in O \vdash \text{loc}_c\{l\} \]
longer result in the loss of input barbs on names of type chl, which implies that the interpretation discussed in the previous section actually holds. That is, for any name a of type chl apart from adhering to the statement in Definition 6, it also holds that

\[
x @l \in I \Rightarrow N \upharpoonright x?@l \quad N \downarrow x!@l \Rightarrow x@l \in O
\]

To sum up, the new type system ensures that all linear channel names reside on a hidden location of type locl, thus it can not be killed by the context. Since the hidden locations stay hidden in our type system, the context does not have to obey the type rules and thus can behave without restraints which result in a network with dynamic link and location failures.

6 Conclusions and further work

We have presented two type systems for enforcing linear uniform receptiveness in the DrF-calculus. The first type system enforces uniform receptiveness by imposing constraints on the migration of processes with obligations to communicate over linear channels. This approach, however, requires the context to be subject to the type system. The second type system instead imposes constraints on where processes may communicate over linear channels and does not require the context to be subject to the type system, but instead forbids communication of location names at observable locations.

As far as we know, these type systems are the first to guarantee linear uniform receptiveness in a process calculus with locations.

An obvious future extension of our work would be to expand our type system to a polyadic setting.

Another possible future line of the work would be a type system enforcing the notion of \( \omega \)-receptiveness presented in (Sangiorgi 1999). Here, all inputs over linear channels are replicated and all outputs over linear channels are only present once per channel. It would also be interesting to investigate the possibilities for achieving uniform receptiveness in DrF using location types only.

Finally, it would be interesting to apply the type inference strategy introduced in (Lhoussaine 2004) to our setting; this would provide us with a sound strategy for determining if a given DrF process is uniform receptive.

References


