Adding recursion to Dpi

Samuel Hym and Matthew Hennessy

Abstract. Dpi is a distributed version of the pi-calculus, in which processes are explicitly located, and a migration construct may be used for moving between locations. We argue that adding a recursion operator to the language increases significantly its

Figure 1 Syntax of recDpi

M, N	::=	*Systems*
$\|JP\|$		Located Process
M		

current site, migrates there and launches a recursive call at this new site. ■

We refrain from burdening the reader with a formal reduction semantics for recDpi, as it is a minor extension of that of Dpi. However in Section 5 we give a typed labelled transition system for the language, the τ-moves of which provides our reduction semantics, see Figure 14. For the current discussion we can focus on the following rules:

(Its-here)
kJhere [x

Figure 2 Recursive pre-types

Base Types:	$B ::= \mathbf{int} \mid \mathbf{bool} \mid \mathbf{unit} \ldots$
Local Channel Types:	$A ::= r\ U \mid w\ T \mid rw\ U, T$
Capability Types:	$C ::= u : A$
Location Types:	$K ::= loc[C_1, \ldots, C_n], n \geq 0 \mid \mu Y.K \mid Y$
Value Types:	$V ::= B \mid A \mid (\tilde{A})_{@}K$
Transmission Types:	$T, U ::= (V_1, \ldots, V_n), n \geq 0$

If *k*'s neighbour is *l*, this further reduces to (up to some reorganisation)

$$(\text{new } ans)\ k\langle ans?(news)\ldots\rangle \mid l\langle Q\rangle$$
$$\mid (\text{new } ans)\ l\langle neigh?(y : R)\ (\text{goto } y.req!\ \text{data}, ans_{@}l\ \ Quest$$
$$\mid ans?(news)\ldots)\rangle$$

with *Q* some code running at *l* to answer the request brought by Quest.

The here construct can also be used to write a process initialising a doubly linked list starting from a simply linked one. We assume for this that the cells are locations containing two specific channels: *n* to get the name of the next cell in the list, *p*

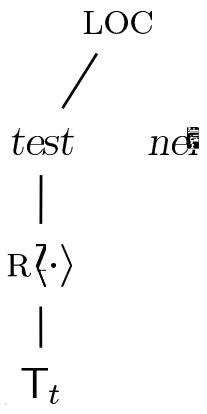
indicating that the local channels u_i may be used at the corresponding type A_i.

However with recursive processes it turns out that we need to consider *infinite* location types. To see this consider again the searching process Search from Example 2.1. Any site, such as *k*, which can support this process needs to have a local channel called *neigh* from which values can be read. These values must be locations, and let us consider their type, that is the object type of *neigh*. These locations must have a local channel called *test*, of an appropriate type, and a local channel called *neigh*; the object type of this local channel must be in turn the same as the type we are trying to describe. Using a recursion operator μ, this type can be described as

$$\mu Y.\mathrm{loc}[\mathit{test} : \mathrm{r}\ T_t\ ,\ \mathit{neigh} : \mathrm{r}\ Y\ ]$$

which will be used as the type S in the definition of Search; it describes precisely the requirements on any site wishing to host this process.

The set of recursive pre-types is given in Figure 2, and is obtained by adding the operator $\mu Y.K$ as a constructor to the type formation rules for Dpi. Following [SW01] we can associate with each recursive pre-type T a co-inductive pre-type denoted Tree(T), which takes the form of a finite-branching, but possibly infinite, tree whose nodes are labelled by the type constructors. For example Tree(S) is the infinite tree



Definition 3.1 (Contractive and Tree pre-type). *We call a recursive pre-type S* contractive *if for every $\mu Y.S$ it contains,* Y *can only appear in S under an occurrence of* loc. *In the paper we will only consider contractive pre-types.*

For every contractive S we can define Tree(S), the unique tree satisfying the following equations:

- *unwinding recursive pre-types* $Tree(\mu Y.S) = Tree(S\{^{\mu Y.S}/_{Y}\})$

Figure 3 Dpi subtyping rules

(sub-base)

$$\frac{}{\textbf{base} <: \textbf{base}}$$

(sub-cap)

$$\frac{A <: B}{u : A <: u : B}$$

(sub-tuple)

$$\frac{C_i <: C_i}{(\overline{C}) <: (\overline{C})}$$

3.2 Theory of tree types

Now that we have defined a notion of tree types out of recursive pre-types, we want to prove some properties of subtyping over these types. For this, the co-inductive definition of subtyping gives rise to a natural co-inductive proof method, the dual of the usual inductive proof method used for sub-typing in Dpi.

This proof method works as follows. To show that some element, say a, is in the greatest fixpoint of any function f it is su□cient to give a set S such that

- a is in S;
- S is a postfixpoint of f, that is $S \quad f(S)$.

From this it follows that S is a subset of the greatest fixpoint of f, which therefore contains the element a.

We will apply this technique to the function Sub, and this case showing that a given S is a postfixpoint is facilitated by the fact that it is invertible, in the meaning of [GLP03, Pie02]. The *inverse* is the partial function defined as follows. Since there is always at most one conclusion in a rule, we can consider this partial function only on pairs:

support$_{\text{Sub}}((\mathbf{T}_1, \mathbf{T}_2))$ =
if $\mathbf{T}_1 = \mathbf{T}_2 =$ **base**
$\{(\mathbf{A}, \mathbf{B})\}$ if $\mathbf{T}_1 = u : \mathbf{A}$ and $\mathbf{T}_2 = u : \mathbf{B}$
$\{(\mathbf{C}_i, \mathbf{C}_i)\}$ if $\mathbf{T}_1 = (\tilde{\mathbf{C}})$ and $\mathbf{T}_2 = (\tilde{\mathbf{C}})$, with same arity
$\{(\mathbf{T}_2, \mathbf{T}_1)\}$ if $\mathbf{T}_1 = \text{w}\ \mathbf{T}_1$ and $\mathbf{T}_2 = \text{w}\ \mathbf{T}_2$
$\{(\mathbf{U}_1, \mathbf{U}_2)\}$ if $\mathbf{T}_1 = \text{r}\ \mathbf{U}_1$ and $\mathbf{T}_2 = \text{r}\ \mathbf{U}_2$
$\{(\mathbf{T}_1, \mathbf{U}_1), (\mathbf{U}_1, \mathbf{U}_1$ if,,, $\mathbf{T}$ 1 U$_1$ \ $\mathbf{T}_2 =$ e same arityi
f $(\mathbf{T} =$

$\text{support}_{\text{Sub}}(R)$ being undefined as soon as $\text{support}_{\text{Sub}}$ is undefined for some tuple in R. Note that this definition implies that $\text{support}_{\text{Sub}}$, as a function from relations to relations, is monotonic on its definition domain.

Intuitively, $\text{support}_{\text{Sub}}$ computes the set of hypotheses needed to reach a given conclusion by Sub. In this sense it can be considered to be an inverse of Sub. Formally the relationship between these two functions is given in the following lemma.

Lemma 3.3. *(1) for a pair t if there exists R such that t* $\text{Sub}(R)$ *then* $\text{support}_{\text{Sub}}(t)$ *is defined;*

(2) $\text{support}_{\text{Sub}}(\text{Sub}(R))$ *R for any relation R.*

Proof. First we prove (1). Suppose $(\mathbf{T}_1, \mathbf{T}_2)$ is in $\text{Sub}(R)$ for some R. It must be by one of the cases in the definition of Sub, and each case

Lemma 3.6 (Transitivity). *Let us suppose that for some tree types* $\mathbf{T}_1$, $\mathbf{T}_2$ *and* $\mathbf{T}_3$, $\mathbf{T}_1 <: \mathbf{T}_2$ *and* $\mathbf{T}_2 <: \mathbf{T}_3$. *Then* $\mathbf{T}_1 <: \mathbf{T}_3$.

Proof. Let us write Tr for the function $\mathrm{Tr}(R) = R \quad R \circ R$. Then, what we want to prove can be formulated as

$$\mathrm{Tr}(\nu\mathrm{Sub}) \quad \nu\mathrm{Sub}$$

for which we can use the coinduction proof principle. It is su□cient to prove that

$$\mathrm{Tr}(\nu\mathrm{Sub}) \quad \mathrm{Sub}(\mathrm{Tr}(\nu\mathrm{Sub})) \tag{1}$$

i.e. that $\mathrm{Tr}(\nu\mathrm{Sub})$ is a postfixpoint of Sub.

For this, let us consider a pair $(\mathbf{T}_1, \mathbf{T}_3)$ in $\mathrm{Tr}(\nu\mathrm{Sub})$. By definition of Tr this implies that either $(\mathbf{T}_1, \mathbf{T}_3)$ is in $\nu\mathrm{Sub}$, in which case it is easy to establish that it is also in $\mathrm{Sub}(\mathrm{Tr}(\nu\mathrm{Sub}))$ because $\nu\mathrm{Sub} \quad \mathrm{Tr}(\nu\mathrm{Sub})$ implies that $\nu\mathrm{Sub} = \mathrm{Sub}(\nu\mathrm{Sub}) \quad \mathrm{Sub}(\mathrm{Tr}(\nu\mathrm{Sub}))$, or else there exists some type $\mathbf{T}_2$ such that $(\mathbf{T}_1, \mathbf{T}_2)$ and $(\mathbf{T}_2, \mathbf{T}_3)$ are in $\nu\mathrm{Sub}$. Therefore $\mathrm{support}_{\mathrm{Sub}}((\mathbf{T}_1, \mathbf{T}_2))$ and $\mathrm{support}_{\mathrm{Sub}}((\mathbf{T}_2, \mathbf{T}_3))$ are defined.

Now we prove that

$$(\mathbf{T}_1, \mathbf{T}_3) \quad \mathrm{Sub}(\mathrm{Tr}(\mathrm{support}_{\mathrm{Sub}}((\mathbf{T}_1, \mathbf{T}_2)) \quad \mathrm{support}_{\mathrm{Sub}}((\mathbf{T}_2, \mathbf{T}_3)))) \tag{2}$$

by case analysis on the fact that $(\mathbf{T}_1, \mathbf{T}_2)$ is in $\mathrm{Sub}(\nu\mathrm{Sub})$. In all there are ten possibilities, of which we examine two typical ones.

- $\mathbf{T}_i = \mathrm{rw}\ \mathbf{U}_i, \mathbf{T}_i$ with $\mathbf{T}_2 <: \mathbf{T}_1$, $\mathbf{T}_1 <: \mathbf{U}_1$ and $\mathbf{U}_1 <: \mathbf{U}_2$. Then $\mathbf{T}_3$ can be any one of the forms $\mathrm{r}\ \mathbf{U}_3$, $\mathrm{w}\ \mathbf{T}_3$, or $\mathrm{rw}\ \mathbf{U}_3, \mathbf{T}_3$, with the relevant constraints among $\mathbf{T}_3 <: \mathbf{T}_2$, $\mathbf{T}_2 <: \mathbf{U}_2$ and $\mathbf{U}_2 <: \mathbf{U}_3$. In the case where $\mathbf{T}_3 = \mathrm{rw}\ \mathbf{U}_3, \mathbf{T}_3$, this implies:

$$\mathrm{support}_{\mathrm{Sub}}((\mathbf{T}_1, \mathbf{T}_2)) \quad \mathrm{support}_{\mathrm{Sub}}((\mathbf{T}_2, \mathbf{T}_3)) = \{(\mathbf{T}_2, \mathbf{T}_1), (\mathbf{T}_1, \mathbf{U}_1), (\mathbf{U}_1, \mathbf{U}_2), (\mathbf{T}_3, \mathbf{T}_2), (\mathbf{T}_2, \mathbf{U}_2), (\mathbf{U}_2, \mathbf{U}_3 \quad \mathrm{Tr}($$

Figure 5 Dpi join rules

(join-base)

$$\frac{}{\mathbf{base}_1 \quad \mathbf{base}_2 = \mathbf{base}_3} \; \mathbf{base}_1 = \mathbf{base}_2 = \mathbf{base}_3$$

(join-cap)

$$\frac{A_1 \quad A_2 = A_3}{u : A_1 \quad u : A_2 = u : A_3}$$

(join-tuple)

$$\frac{C_i \quad C_i = C_i}{(e) \quad (C^e) = (C^f)}$$

(join-chan)

$$\frac{\begin{array}{l} U_1 \quad U_2 = U_3 \\ T_1 \quad T_2 = T_3 \end{array}}{\mathrm{rw}\ U_1, T_1 \quad \mathrm{rw}\ U_2, T_2 \quad = \mathrm{rw}\ U_3, T_3} \; \begin{array}{l} T_1 <: U_1 \\ T_2 <: U_2 \end{array}$$

(join-hom)

$$\frac{\begin{array}{l} A_1 \quad A_2 = A_3 \\ K_1 \quad K_2 = K_3 \end{array}}{A_1 @ K_1 \quad A_2 @ K_2 = A_3 @ K_3}$$

(join-loc)

$$\frac{U_i \quad U_i = U_i}{\mathrm{loc}[(u_i : U_i)_i; (v_j : V_j)_j] \quad \mathrm{loc}[(u_i : U_i)_i; (w_k : W_k)_k] = \mathrm{loc}[(u_i : U_i)_i]}$$

- $\mathrm{loc}[u_1 : \mathbf{A}_1, \ldots, u_n : \mathbf{A}_n$

Figure 6 MeetJoin definition

$$
\begin{aligned}
&\mathsf{MeetJoin}(\mathcal{R}) = \\
&\quad \{\sqcap(\mathbf{base}, \mathbf{base}, \mathbf{base})\} \\
&\cup \{\sqcap(u : \mathbf{A}_1, u : \mathbf{A}_2, u : \mathbf{A}_3) \text{ if } \sqcap(\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3) \text{ is in } \mathcal{R}\} \\
&\cup \{\sqcap((\widetilde{\mathbf{C}}), (\widetilde{\mathbf{C}'}), (\widetilde{\mathbf{C}''})) \text{ if } \sqcap(\mathbf{C}_i, \mathbf{C}'_i, \mathbf{C}''_i) \text{ is in } \mathcal{R} \text{ for all } i\} \\
&\cup \{\sqcap(\mathsf{r}\langle \mathbf{U}_1\rangle, \mathsf{r}\langle \mathbf{U}_2\rangle, \mathsf{r}\langle \mathbf{U}_3\rangle) \text{ if } \sqcap(\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3) \text{ is in } \mathcal{R}\} \\
&\cup \{\sqcap(\mathsf{w}\langle \mathbf{T}_1\rangle, \mathsf{w}\langle \mathbf{T}_2\rangle, \mathsf{w}\langle \mathbf{T}_3\rangle) \text{ if } \sqcup(\mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3) \text{ is in } \mathcal{R}\} \\
&\cup \{\sqcap(\mathsf{r}\langle \mathbf{U}_1\rangle, \mathsf{w}\langle \mathbf{T}_2\rangle, \mathsf{rw}\langle \mathbf{U}_1, \mathbf{T}_2\rangle) \text{ if } \mathbf{T}_2 <: \mathbf{U}_1\} \\
&\cup \{\sqcap(\mathsf{w}\langle \mathbf{T}_1\rangle, \mathsf{r}\langle \mathbf{U}_2\rangle, \mathsf{rw}\langle \mathbf{U}_2, \mathbf{T}_1\rangle) \text{ if } \mathbf{T}_1 <: \mathbf{U}_2\} \\
&\cup \{\sqcap(\mathsf{rw}\langle \mathbf{U}_1, \mathbf{T}_1\rangle, \mathsf{r}\langle \mathbf{U}_2\rangle, \mathsf{rw}\langle \mathbf{U}_3, \mathbf{T}_1\rangle) \text{ if } \sqcap(\mathbf{U}_1, \mathbf{U}_2, \mathbf{U}_3) \text{ is in } \mathcal{R} \text{ and } \mathbf{T}_1 <: \mathbf{U}_3\} \\
&\cup \{\sqcap(\mathsf{rw}\langle \mathbf{U}_1, \mathbf{T}_1\rangle, \mathsf{w}\langle \mathbf{T}_2\rangle \mathsf{w} \qquad \text{and } \mathbf{T}_1
\end{aligned}
$$

Lemma 3.8 (Symmetry on arguments). *For any types* $\mathbf{T}_1$, $\mathbf{T}_2$ *and* $\mathbf{T}_3$, $\mathbf{T}_1 \quad \mathbf{T}_2 = \mathbf{T}_3$ *if and only if* $\mathbf{T}_2 \quad \mathbf{T}_1 = \mathbf{T}_3$.

Proof. Since the definition of MeetJoin is completely symmetric on its first two components, it is enough to consider the relation

$$R = \{ (\mathbf{T}_2, \mathbf{T}_1, \mathbf{T}_3) \mid \mathbf{T}_1 \quad \mathbf{T}_2 = \mathbf{T}_3 \} \quad \{ (\mathbf{T}_2, \mathbf{T}_1, \mathbf{T}_3) \mid \mathbf{T}_1 \quad \mathbf{T}_2 = \mathbf{T}_3 \}$$

which is easily shown to be a postfixpoint of the operator MeetJoin. □

Lemma 3.9 (Reflexivity of and). *For any type* $\mathbf{T}$, *we have* $\mathbf{T} \quad \mathbf{T} = \mathbf{T}$ *and* $\mathbf{T} \quad \mathbf{T} = \mathbf{T}$.

Proof. We simply consider the relation

$$R = \{ (\mathbf{T}, \mathbf{T}, \mathbf{T}), \ (\mathbf{T}, \mathbf{T}, \mathbf{T}) \mid \mathbf{T} \text{ is any type} \}$$

and prove that it is a postfixpoint of MeetJoin.

For this, let us consider a triple in R. We reason on the form of that triple, namely on the head construct for the type it is based on, and on the operator, or . All the cases are very similar, we give only a few of them.

- (**base, base, base**). Obviously this triple is also in MeetJoin(R).
- (rw $\mathbf{U}, \mathbf{T}$, rw $\mathbf{U}, \mathbf{T}$, rw $\mathbf{U}, \mathbf{T}$). As we know that both ($\mathbf{U}, \mathbf{U}, \mathbf{U}$) and ($\mathbf{T}, \mathbf{T}, \mathbf{T}$) are also in R, we can conclude that our triple is in MeetJoin(R).
- (loc[u_i : $\mathbf{A}_i$], loc[u_i : $\mathbf{A}_i$], loc[u_i : $\mathbf{A}_i$]) is in MeetJoin(R) because of the triples ($\mathbf{A}_i o$ ψΤD A TJ F ψ ΤfψψΤD i TJ F ψ ΤfψψΤD

- $\mathbf{T}^i = \mathbf{C}^{f_i}$, with $\mathbf{C}^2_j \quad \mathbf{C}^3_j = \mathbf{C}^1_j$ for all j. Then, for all j, $\mathbf{C}^1_j \; R \; \mathbf{C}^2_j$, which implies that $(\mathbf{T}^1, \mathbf{T}^2)$ is in R.
- If the triple is $(\mathrm{r}\ \mathbf{U}^2_0\ , \mathrm{w}\ \mathbf{T}^3_0\ , \mathrm{rw}\ \mathbf{U}^2_0, \mathbf{T}^3_0\ )$ we know that $\mathbf{T}^3_0 <: \mathbf{U}^2_0$ so $(\mathbf{T}^3_0, \mathbf{U}^2_0)$ is in particular in R. Moreover, by lemma 3.4, we know that $\mathbf{U}^2_0 <: \mathbf{U}^2_0$. These two hypotheses allow us to conclude that $(\mathrm{rw}\ \mathbf{U}^2_0, \mathbf{T}^3_0\ ), \mathrm{r}\ \mathbf{U}^2_0\ )$ is in Sub(R).
- If the triple is $(\mathrm{rw}\ \mathbf{U}^2_0, \mathbf{T}^2_0\ , \mathrm{r}\ \mathbf{U}^3_0\ , \mathrm{rw}\ \mathbf{U}^1_0, \mathbf{T}^2_0\ )$ we know that $\mathbf{U}^2_0 \quad \mathbf{U}^3_0 = \mathbf{U}^1_0$ so $(\mathbf{U}^1_0, \mathbf{U}^2_0)$ is in R and that $\mathbf{T}^2_0 <: \mathbf{U}^1_0$ so $(\mathbf{T}^2_0, \mathbf{U}^1_0)$ is in particular in R. Moreover, by lemma 3.4, we know that $\mathbf{T}^2_0 <: \mathbf{T}^2_0$. These three hypotheses allow us to conclude that $(\mathrm{rw}\ \mathbf{U}^1_0, \mathbf{T}^2_0\ , \mathrm{rw}\ \mathbf{U}^2_0, \mathbf{T}^2_0\ )$ is in Sub(R).
- If the triple is $(\mathrm{loc}[u_i : \mathbf{A}^2_i, v_j : \mathbf{B}^2_j], \mathrm{loc}[u_i : \mathbf{A}$

the form rw $\mathbf{U}_0, \mathbf{T}_0$ with $\mathbf{U}_0 <: \mathbf{U}_0^1$ and $\mathbf{T}_0^2 <: \mathbf{T}_0$. By well-formedness of $\mathbf{T}$ we also know that $\mathbf{T}_0 <: \mathbf{U}_0$. Which means that $(\mathbf{T}, \mathbf{T}^3)$ is in Sub(νSub) Sub(R).

- r $\mathbf{U}_0^1$ rw $\mathbf{U}_0^2, \mathbf{T}_0^2$ = rw $\mathbf{U}_0^3, \mathbf{T}_0^2$, then $\mathbf{T} <: \mathbf{T}^2$ implies that $\mathbf{T}$ is of the form rw $\mathbf{U}_0, \mathbf{T}_0$ with $\mathbf{U}_0 <: \mathbf{U}_0^2$ and $\mathbf{T}_0^2 <: \mathbf{T}_0$. We also have that $\mathbf{U}_0 <: \mathbf{U}_0^1$. Of course, we have that $\mathbf{U}_0^1$ $\mathbf{U}_0^2 = \mathbf{U}_0^3$, so $(\mathbf{U}_0, \mathbf{U}_0^3)$ is in R. As so is $(\mathbf{T}_0^2, \mathbf{T}_0)$ and $(\mathbf{T}_0, \mathbf{U}_0)$ by well-formedness of $\mathbf{T}$, $(\mathbf{T}, \mathbf{T}^3)$ is in Sub(R).
- loc$[u_i : A1_i, v_j : B1_j]$ loc$[u_i : A2_i, w_k : B2_k]$ = loc$[u_i : A3_i, v_j : B1_j, w_k : B2_k]$, which implies that $\mathbf{A}_i^1$ $\mathbf{A}_i^2 = \mathbf{A}_i^3$ for all i. The fact $\mathbf{T}$ is a common subtype of $\mathbf{T}^1$ and $\mathbf{T}^2$ implies that it must be of the form loc$[u_i : AA_i, v_j : B4_j, w_k : B5_k, x_l : B6_l]$ with $\mathbf{A}_i <: \mathbf{A}_i^1$ and $\mathbf{A}_i <: \mathbf{A}_i^2$ for all i, and with $\mathbf{B}_j^4 <: \mathbf{B}_j^1$ and $\mathbf{B}_k^5 <: \mathbf{B}_k^2$. This implies that $(\mathbf{A}_i, \mathbf{A}_i^3)$, $(\mathbf{B}_j^4, \mathbf{B}_j^1)$ and $(\mathbf{B}_k^5, \mathbf{B}_k^2)$ are in R. So $(\mathbf{T}, \mathbf{T}^3)$ is in Sub(R).

For the second case, let us now suppose that the pair is of the form $(\mathbf{T}^3, \mathbf{T})$, with the corresponding types $\mathbf{T}^1$ and $\mathbf{T}^2$. We reason on $\mathbf{T}^1$ $\mathbf{T}^2 = \mathbf{T}^3$.

- rw $\mathbf{U}_0^1, \mathbf{T}_0^1$ rw $\mathbf{U}_0^2, \mathbf{T}_0^2$ = r $\mathbf{U}_0^3$ which implies that $\mathbf{U}_0^1$ $\mathbf{U}_0^2 = \mathbf{U}_0^3$ and $\mathbf{T}_0^1 \mathbf{T} \downarrow_0^2$. If $\mathbf{T}$ was of the form w $\mathbf{T}_0$ or rw $\mathbf{U}_0, \mathbf{T}_0$, $\mathbf{T}^1 <: \mathbf{T}$ and $\mathbf{T}^2 <: \mathbf{T}$ would imply $\mathbf{T}_0 <: \mathbf{T}_0^1$ and $\mathbf{T}_0 <: \mathbf{T}_0^2$, which would contradict the fact that those two types are incompatible. So $\mathbf{T}$ must be of the form r $\mathbf{U}_0$, with $\mathbf{U}_0^1 <: \mathbf{U}_0$ and $\mathbf{U}_0^2 <: \mathbf{U}_0$ which means that $(\mathbf{U}_0, \mathbf{U}_0^3)$ is in R and $(\mathbf{T}, \mathbf{T}^3)$ in Sub(R).

The last case is that the pair is in νSub, which means that it is obviously in Sub(R).

So we have proved that R is a subset of νSub, which finishes our proof. □

The major property of the meet operator is given by the following lemma. This lemma is again proved by using that approach of coinductive proofs. To state it, let us write $\mathbf{T}_1 \downarrow \mathbf{T}_2$ to mean that there is some $\mathbf{T}$ such that $\mathbf{T} <: \mathbf{T}_1$ and $\mathbf{T} <: \mathbf{T}_2$, that is $\mathbf{T}_1$ and $\mathbf{T}_2$ are compatible.

Theorem 3.12 (Partial meets). *The set of tree types, ordered by* <:*, has partial meets. That is* $\mathbf{T}_1 \downarrow \mathbf{T}_2$ *implies* $\mathbf{T}_1$ *and* $\mathbf{T}_2$ *have a meet.*

Proof. Let us consider two types $\mathbf{T}_1$ and $\mathbf{T}_2$ that are compatible. Their compatibility implies that the set $\{\mathbf{T} \mid \mathbf{T} <: \mathbf{T}_1, \mathbf{T} <: \mathbf{T}_2\}$ is not empty. So we can consider its set of maximal elements, which are elements $\mathbf{T}$ such that for any element $\mathbf{T}$, $\mathbf{T} <: \mathbf{T}$ implies $\mathbf{T} = \mathbf{T}$. Notice that this equality

is the one we previously defined by Eq. We write $M(\mathbf{M}(\mathbf{T}$

If t is of the form $(\mathbf{T}_1, \mathbf{T}_2, \mathbf{T}_3)$, we reason on the fact that $\mathbf{T}_3$ is a supertype of $\mathbf{T}_1$. Let us just see the most interesting case, loc[...].

- $\mathbf{T}_1 = \text{loc}[u_1 : \mathbf{A}_1, \ldots, u_n : \mathbf{A}_n, u_{n+1} : \mathbf{A}_{n+1}, \ldots, v_1 : \mathbf{B}_1, \ldots]$ and $\mathbf{T}_3 = \text{loc}[u_1 : \mathbf{A}_1, \ldots, u_n : \mathbf{A}_n]$. Then $\mathbf{T}_2$ must be of the form $\text{loc}[u_1 : \mathbf{A}_1, \ldots, u_n : \mathbf{A}_n, u_{n+1} : \mathbf{A}_{n+1}, \ldots, w_1 : \mathbf{B}_1, \ldots]$. Since $\mathbf{T}_3$ is minimal among the common supertypes of $\mathbf{T}_1$ and $\mathbf{T}_2$, we know that, for every j, $\mathbf{A}_{n+j} \uparrow \mathbf{A}_{n+j}$: otherwise, if $\mathbf{A}_{n+1} \uparrow \mathbf{A}_{n+1}$, we can define the type $\mathbf{A}_{n+1}$ as a supertype of $\mathbf{A}_{n+1}$ and $\mathbf{A}_{n+1}$, and consider $\text{loc}[u_1 : \mathbf{A}_1, \ldots, u_n : \mathbf{A}_n, u_{n+1} : \mathbf{A}_{n+1}, \mathbf{A}$

Figure 7 Subtyping rules

(sr-ax)

$$\frac{}{\Sigma, T_1 <: T_2 \quad T_1 <: T_2}$$

(sr-cap)

$$\frac{\Sigma \quad A <: B}{\Sigma \quad u : A <: u : B}$$

(sr-chan)

$$\frac{\Sigma \quad T_1 <: T_2 <: U_1 <: U_2}{\begin{array}{l}\Sigma \quad w\ T_2 <: w\ T_1 \\ \Sigma \quad r\ U_1 <: r\ U_2 \\ \Sigma \quad rw\ U_1, T_2 <: r\ U_2 \\ \Sigma \quad rw\ U_1, T_2 <: w\ T_1 \\ \Sigma \quad rw\ U_1, T_2 <: rw\ U_2, T_1\end{array}}$$

(sr-loc)

$$\Sigma \quad U_i <: U$$

(sr-base)

$$\frac{}{\Sigma \quad \mathbf{base} <: \mathbf{base}}$$

(sr-tuple)

$$\frac{\Sigma \quad C_i <: C_i}{\Sigma \quad (\tilde{e}) <: (\tilde{e})}$$

(sr-hom)

$$\frac{\Sigma \quad A_1 <: A_2 \qquad \Sigma \quad K_1 <: K_2}{\Sigma \quad A_1 @ K_1 <: A_2 @ K_2}$$

Figure 8 Meet inference rules

(meet-ax)

$$\frac{}{\Sigma, \mathsf{T}_1 \sqcap \mathsf{T}_2 = \mathsf{T}_3 \vdash \mathsf{T}_1 \sqcap \mathsf{T}_2 = \mathsf{T}_3}$$

(meet-tuple)

$$\frac{\Sigma \vdash \mathsf{C}_i \sqcap \mathsf{C}'_i = \mathsf{C}''_i}{\Sigma \vdash (\widetilde{\mathsf{C}}) \sqcap (\widetilde{\mathsf{C}'}) = (\widetilde{\mathsf{C}''})}$$

(meet-base)

$$\frac{}{\Sigma \vdash \mathbf{base}_1 \sqcap \mathbf{base}_2 = \mathbf{base}_3} \; \mathbf{base}_1 = \mathbf{base}_2 = \mathbf{base}_3$$

(meet-cap)

$$\frac{\Sigma \vdash \mathsf{A}_1 \sqcap \mathsf{A}_2 = \mathsf{A}_3}{\Sigma \vdash u : \mathsf{A}_1 \sqcap u : \mathsf{A}_2 = u : \mathsf{A}_3}$$

(meet-chan)

$$\frac{\Sigma \vdash \mathsf{U}_1 \sqcap \mathsf{U}_2 = \mathsf{U}_3 \quad \Sigma \vdash \mathsf{T}_1 \sqcup \mathsf{T}_2 = \mathsf{T}_3}{\Sigma \vdash \mathsf{rw}\langle \mathsf{U}_1, \mathsf{T}_1 \rangle \sqcap \mathsf{rw}\langle \mathsf{U}_2, \mathsf{T}_2 \rangle = \mathsf{rw}\langle \mathsf{U}_3, \mathsf{T}_3 \rangle} \; \mathsf{T}_3 <: \mathsf{U}_3$$

(meet-hom)

$$\frac{\Sigma \vdash \mathsf{A}_1 \sqcap \mathsf{A}_2 = \mathsf{A}_3 \quad \Sigma \vdash \mathsf{K}_1 \sqcap \mathsf{K}_2 = \mathsf{K}_3}{\Sigma \vdash \mathsf{A}_1@}$$

Figure 9 Join inference rules

(join-ax)

$$\frac{}{\Sigma, T_1 \sqcup T_2 = T_3 \vdash T_1 \sqcup T_2 = T_3}$$

(join-tuple)

$$\frac{\Sigma \vdash C_i \sqcup C'_i = C''_i}{\Sigma \vdash (\widetilde{C}) \sqcup (\widetilde{C'}) = (\widetilde{C''})}$$

(join-base)

$$\frac{}{\Sigma \vdash \mathbf{base}_1 \sqcup \mathbf{base}_2 = \mathbf{base}_3} \quad \mathbf{base}_1 = \mathbf{base}_2 = \mathbf{base}_3$$

(join-cap)

$$\frac{\Sigma \vdash A_1 \sqcup A_2 = A_3}{\Sigma \vdash u : A_1 \sqcup u : A_2 = u : A_3}$$

(join-chan-rw-rw-r)

$$\frac{\Sigma \vdash U_1 \sqcup U_2 = U_3}{\Sigma \vdash \mathsf{rw}\langle U_1, T_1\rangle \sqcup \mathsf{rw}\langle U_2, T_2\rangle = \mathsf{r}\langle U_3\rangle} \quad \begin{array}{l} T_1 <: U_1 \quad T_2 <: U_2 \\ U_1 \uparrow U_2 \quad T_1 \downarrow T_2 \end{array}$$

(join-chan-rw-rw-w)

$$\frac{\Sigma \vdash T_1 \sqcap T_2 = T_3}{\Sigma \vdash \mathsf{rw}\langle U_1, T_1\rangle \sqcup \mathsf{rw}\langle U_2, T_2\rangle = \mathsf{w}\langle T_3\rangle} \quad \begin{array}{l} T_1 <: U_1 \quad T_2 <: U_2 \\ U_1 \uparrow U_2 \quad T_1 \downarrow T_2 \end{array}$$

(join-chan-rw-rw-rw)

$$\frac{\Sigma \vdash U_1 \sqcup U_2 = U_3 \quad \Sigma \vdash T_1 \sqcap T_2 = T_3}{\Sigma \vdash \mathsf{rw}\langle U_1, T_1\rangle \sqcup \mathsf{rw}\langle U_2, T_2\rangle = \mathsf{rw}\langle U_3, T_3\rangle} \quad \begin{array}{l} T_1 <: U_1 \quad T_2 <: U_2 \\ U_1 \uparrow U_2 \quad T_1 \downarrow T_2 \end{array}$$

(join-hom)

$$\frac{\Sigma \vdash A_1 \sqcup A_2 = A_3 \quad \Sigma \vdash K_1 \sqcup K_2 = K_3}{\Sigma \vdash A_1@K_1 \sqcup A_2@K_2 = A_3@K_3}$$

(join-loc)

$$\frac{\Sigma \vdash U_i \sqcup U'_i = U''_i}{\begin{array}{l}\Sigma \vdash \mathsf{loc}[(u_i : U_i)_i; (u_{n+i} : U_{n+i})_i; (v_j : V_j)_j] \\ \quad \sqcup \mathsf{loc}[(u_i : U'_i)_i; (u_{n+i} : U'_{n+i})_i; (w_k : W_k)_k] = \mathsf{loc}[(u_i : U''_i)_i]\end{array}} \quad \begin{array}{l} U_i \uparrow U'_i \\ U_{n+i} \uparrow U'_{n+i} \end{array}$$

(join-rec-1)

$$\frac{\Sigma, \mu\mathbf{Y}.T'_1 \sqcup T_2 = T_3 \vdash T'_1\{^{\mu\mathbf{Y}.T_1}/_{\mathbf{Y}}\} \sqcup T_2 = T_3}{\Sigma \vdash \mu\mathbf{Y}.T'_1 \sqcup T_2 = T_3}$$

(join-rec-2)

$$\frac{\Sigma, T_1 \sqcup \mu\mathbf{Y}.T'_2 = T_3 \vdash T_1 \sqcup T'_2\{^{\mu\mathbf{Y}.T_2}/_{\mathbf{Y}}\} = T_3}{\Sigma \vdash T_1 \sqcup \mu\mathbf{Y}.T'_2 = T_3}$$

(join-rec-3)

$$\frac{\Sigma, T_1 \sqcup T_2 = \mu\mathbf{Y}.T'_3 \vdash T_1 \sqcup T_2 = T'_3\{^{\mu\mathbf{Y}.T_3}/_{\mathbf{Y}}\}}{\Sigma \vdash T_1 \sqcup T_2 = \mu\mathbf{Y}.T'_3}$$

seen. The rules we obtain look like the ones in [AC93] in the Dpi setting.

Conversely, let us consider some proof of $\quad T_1 \quad T_2 = T_3$. We define the relation

$$\begin{aligned} R = \{ & \ (\mathrm{Tree}(T_1), \mathrm{Tree}(T_2), \mathrm{Tree}(T_3)) \mid \ \Sigma \text{ such that} \\ & \quad \Sigma \quad T_1 \quad T_2 = T_3 \text{ appears in the proof of } \quad T_1 \quad T_2 = T_3 \} \\ \{ & \ (\mathrm{Tree}(T_1), \mathrm{Tree}(T_2), \mathrm{Tree}(T_3)) \mid \ \Sigma \text{ such that} \\ & \quad \Sigma \quad T_1 \quad T_2 = T_3 \text{ appears in the proof of } \quad T_1 \quad T_2 = T_3 \} \end{aligned}$$

Let us prove that this relation R is a postfixpoint of MeetJoin. We consider a triple in R and we reason on the last rule used to reach the corresponding statement in the proof of $\quad T_1 \quad T_2 = T_3$.

- (meet-ax). Then we know that $T_1 \quad T_2 = T_3$ can have been introduced in Σ only by a rule (meet-rec-i) higher in that branch of the proof. Since our types are contractive, we then know that T_i must be of the form

Figure 10 Well-formed environments

$$\frac{}{\Gamma \quad \textbf{env}}\ \text{(e-empty)}$$

$$\frac{\Gamma \quad \textbf{env}}{\Gamma, u : \textbf{base} \quad \textbf{env}}\ \Gamma(u) \downarrow \textbf{base}\ \text{(e-base)}$$

$$\frac{\Gamma \quad \textbf{env} \quad \Gamma \quad w : \mathrm{loc} \quad \Gamma(u@w) = \{A_i\}}{\Gamma, u@w : A \quad \textbf{env}}\ \{A_i\} \downarrow A\ \text{(e-new-lchan)}$$

$$\frac{\Gamma \quad \textbf{env}}{\Gamma, v : \mathrm{loc} \quad \textbf{env}}\ \Gamma(v) \downarrow \mathrm{loc}\ \text{(e-loc)}$$

$$\frac{\Gamma \quad \textbf{env}}{\Gamma, Z : \mathrm{loc}[(u_i : A_i)] \quad \textbf{env}}\ Z \quad \Gamma\ \text{(e-rec)}$$

$$\frac{\Gamma \quad \textbf{env} \quad \Gamma(Z) = \mathrm{loc}[\ldots, u : A, \ldots]}{\Gamma, u@Z : A \quad \textbf{env}}\ \text{(e-dec-at-rec)}$$

Figure 11 Typing values

$$\frac{\Gamma, u : T, \Gamma \quad \textbf{env}}{\Gamma, u : T, \Gamma \quad u : T}\ T <: T'\ \text{(v-name)}$$

$$\frac{\Gamma \quad u : T \quad \Gamma \quad w : \mathrm{loc}}{\Gamma \quad {}_w u : T}\ \text{(v-located)}$$

$$\frac{\Gamma, u@w : A, \Gamma \quad w : \mathrm{loc}}{\Gamma, u@w : A, \Gamma \quad {}_w u : A}\ A <: A'\ \text{(v-channel)}$$

$$\frac{\Gamma \quad {}_w u : T_1 \quad \Gamma \quad {}_w u : T_2}{\Gamma \quad {}_w u : T_1 \quad T_2}\ \text{(v-meet)}$$

$$\frac{\Gamma \quad {}_w u_i : T_i}{\Gamma \quad {}_w (\tilde{u}) : (\tilde{T})}\ \text{(v-tuple)}$$

$$\frac{\Gamma \quad w : \mathrm{loc}}{\Gamma \quad {}_w u : \textbf{base}}\ u \quad \textbf{base}\ \text{(v-base)}$$

$$\Gamma \quad {}_v u_i : A_i \ ; \Gamma\ \text{(v-located-channel)}$$

Figure 12 Typing Systems

$$\frac{\Gamma, c@k : \mathsf{C} \quad M}{\Gamma \quad (\mathsf{new}\, c@k : \mathsf{C})\, M}\ (\text{t-cnew})$$

$$\frac{}{\Gamma \quad \mathbf{env}}\ (\text{t-nil}) \qquad \frac{}{\Gamma \quad \mathbf{0}}$$

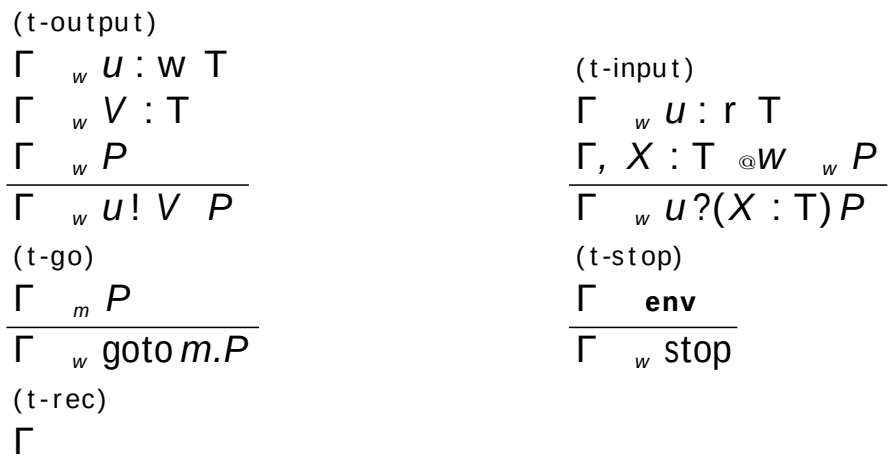
$$\frac{\Gamma \quad M \qquad \Gamma \quad N}{\Gamma \quad M \mid N}\ (\text{t-par})$$

$$\frac{\Gamma \quad {}_{k} P}{\Gamma \quad k[\![P]\!]}\ (\text{t-proc}) \qquad \frac{\Gamma, \{k : \mathsf{K}\} \quad M}{\Gamma \quad (\mathsf{new}\, k : \mathsf{K})\, M}\ (\text{t-lnew})$$

ing a recursion variable) and T its type or of the form $u@w : A$ with u a name or a variable standing for a channel and A its type. A given u can appear more than once in that list as long as the types at which it is known in a given location are compatible. This is useful for the names received during communications: if you get some name at two different types (through communication on two different channels), you can simply consider the environment in which that name is given those two types. Of course, the typing rules will ensure that this situation will arise only when the channel types are indeed compatible. Note that the formation rules for

$\mathsf{K}\} \quad \dot{M}k \; \Psi f \psi \qquad F^{\psi}\psi \quad \Psi f \psi \qquad \Psi f \psi \quad dom F \; \psi \quad \Psi f \psi \qquad F \; \psi \Psi f \psi \qquad \psi$

Figure 13 recDpi processes typing system



and the here construct. The latter is straightforward:

$$\frac{\Gamma \vdash_w P[w/x]}{\Gamma \vdash_w \mathsf{here}[x].P}\ (\text{t-here})$$

However in order to derive judgements about recursive processes, such as

$$\Gamma \vdash_k \mathsf{rec}\,(Z : \mathrm{R}).\ P \tag{3}$$

we will need the entries for recursion variables. Recall that here the type R is a location type, such as $\mathsf{loc}[u_1 : \mathrm{A}_1, \ldots u_n : \mathrm{A}_n]$, indicating the minimal requirements on any location wishing to host a call to the recursive procedure. So in some way we want to consider recursion variables in the same manner as locations. But we must be careful as we need to know

typing proceeds. Z will never be a value in real terms, this being syntactically prohibited.

Example 4.1. Referring back to Example 2.1 let us see how these rules can be used to infer $\Gamma \vdash_k$ Search, assuming that Γ knows about locations home, k, etc. and their channels. So, by (t-rec), this will amount to:

$$\Gamma, Z : S \vdash_z \mathit{test}?(x)\text{if } p(x) \text{ then goto home}.\mathit{report}!\, x \text{ else } \mathit{neigh}?(y) \text{ goto}$$

The main new technical property of the type inference system is given by:

Lemma 4.1 (Recursion Variable Substitution). *Suppose that* $\Gamma \vdash_w \mathbf{rec}\, Z : R.\, P$. *Then* $\Gamma \vdash_w P\{^{\mathbf{rec}\, Z:R.\, P}/_Z\}$.

Proof. This is done by induction on the proof that P is well-typed. So we generalise the property we prove into: for any location or recursion variable v and for any environment Γ if we have $\Gamma \vdash_v P$ and if for any Γ' and w such that $\Gamma' <: \Gamma$ and $\Gamma' \vdash w : \Gamma(Z)$ we have $\Gamma' \vdash_w Q$ then $\Gamma' \vdash_v P\{^Q/_Z\}$.

- (t-recvar) so $P = Z$ and we know $\Gamma \vdash v : \Gamma(Z)$. By hypothesis that implies that $\Gamma \vdash_v Q = P\{^Q/_Z\}$.
- (t-output) so $P = u!\langle V \rangle.P$. This implies that $\Gamma \vdash_v P$, on which we can apply the induction hypothesis. Therefore we have

  $$\Gamma \vdash_v u!\langle V \rangle.(P\{^Q/_Z\})$$

  which is exactly $\Gamma \vdash_v P$.
- (t-input) so $P = u?(X : \mathrm{T}).P$ and $\Gamma, X : \mathrm{T} \otimes v \vdash_v P$. By weakening we know that, for any w such that $\Gamma, X : \mathrm{T} \otimes v \vdash w : (\Gamma, X : \mathrm{T} \otimes v)(Z)$, $\Gamma, X : \mathrm{T} \otimes v \vdash_w Q$.
- (t-match) which implies that $P = \mathbf{if}\ u = u\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2$ and that $\Gamma \vdash u : \mathrm{U}, u : \mathrm{U}$, $\Gamma \vdash_v P_2$ and, if $\Gamma, u : \mathrm{U} \otimes v, u : \mathrm{U} \otimes v \vdash \mathbf{env}$, $\Gamma, u : \mathrm{U} \otimes v, u : \mathrm{U} \otimes v \vdash_v P_1$. Then, by our induction hypothesis, we know that $\Gamma \vdash_v P_2\{^Q/_Z\}$. And since $\Gamma, u : \mathrm{U} \otimes v, u : \mathrm{U} \otimes v <: \Gamma$ then $\Gamma, u : \mathrm{U} \otimes v, u : \mathrm{U} \otimes v \vdash_v P_1\{^Q/_Z\}$.
- (t-here) so $P = \mathbf{here}[x].P$ and $\Gamma \vdash_v P[^v/_x]$. By our induction hypothesis we have $\Gamma \vdash_v P[^v/_x]\{^Q/_Z\}$ and $P\{^Q/_Z\}[^v/_x] = P\{^Q/_Z\}[^v/_x]$ since the two substitutions do not deal with the same objects (recursion variables as terms and location variables). So applying (t-here) again gives $\Gamma \vdash_v (\mathbf{here}[x].P)\{^Q/_Z\}$.
- (t-rec) so $P = \mathbf{rec}\, Z : \mathrm{R}.\, P$ with $\Gamma, Z : \mathrm{R} \vdash_z P$. Since $\Gamma, Z : \mathrm{R}$ is a subtype-environment of Γ we can apply our induction hypothesis on it to get $\Gamma, Z : \mathrm{R} \vdash_z P\{^Q/_Z\}$ which implies that $\Gamma \vdash_v P\{^Q/_Z\}$.

Now we must prove that what we just proved indeed applies to processes of the form $\mathbf{rec}\, Z : \mathrm{R}.\, P$. We know that $\Gamma \vdash_w \mathbf{rec}\, Z : \mathrm{R}.\, P$. This implies that $\Gamma, Z : \mathrm{R} \vdash_z P$. By weakening, we obtain that, for any Γ'

such that $\Gamma <: \Gamma, \Gamma, Z : \mathrm{R} \vdash_z P$. So, for any location v such that $\Gamma \vdash v : (\Gamma, Z : \mathrm{R})(Z) = \mathrm{R}$, we have $\Gamma \vdash_v \mathrm{rec}\ Z : \mathrm{R}.\ P$.

So we can use $\mathrm{rec}\ Z : \mathrm{R}.\ P$ as a "Q" in the previous proof and then conclude. □

This in turn leads to:

Theorem 4.2 (Subject Reduction). *$\Gamma \vdash M$ and $M \xrightarrow{\tau} M'$ implies that $\Gamma \vdash M'$.*

Proof. This proof heavily relies on the preexisting proof of subject reduction in Dpi. We simply added two derivation rules (lts-here) and (lts-rec) so we just have to deal with those two.

- $M = k[\mathrm{here}\,[x]\,P]$ and $M' = k[P[k/x]]$. The result is direct since the only rule to prove that $\Gamma \vdash_k \mathrm{here}\,[x]\,P$ assumes that $\Gamma \vdash_k P[k/x]$.
- $M = k[\mathrm{rec}\ Z : \mathrm{R}.\ P]$ and $M' = k[P\{^{\mathrm{rec}\ Z:\mathrm{R}.\ P}/_Z\}]$. By the previous lemma $\Gamma \vdash_k \mathrm{rec}\ Z : \mathrm{R}.\ P$ implies that $\Gamma \vdash_k P\{^{\mathrm{rec}\ Z:\mathrm{R}.\ P}/_Z\}$. That proves that $\Gamma \vdash M'$.

□

5 Implementing recursion using iteration

The problem of implementing recursion using iteration in Dpi, contrary to the pi-calculus, is that any code of the form $k[\ P]$ will force every instance of P to be launched at the originating site k; this is in contrast to $k[\mathrm{rec}\ (Z : \mathrm{R}).\ P]$ where the initial instance of the body P is launched at k but subsequent instances may be launched at arbitrary sites, provided they are appropriately typed.

Nevertheless, at the expense of repeated migrations, we can mimic the behaviour of a recursive process using iteration by designating a *home base* to which the process must return before a new instance is launched. For example if *home* is deemed to be the home base then we can implement our example $k[\mathrm{Search}]$ using

$$home[\ \mathrm{IterSearch}] \mid k[\mathrm{FireOne}]$$

where

$$\mathrm{IterSearch} \triangleq ping?(l)\ \mathrm{goto}\ l.test?(x)$$

call with a few reductions. FireOne is the "translation" for the recursive calls, which means going to the home base and firing a new instance. This shows why the construct here is necessary: the translation for recursive calls needs to detect its current location to indeed trigger the new instance in the "proper" context. Then the replicated IterSearch starts o by migrating to the actual location where it will run.

This approach underlies our general translation of recursive processes into iterative processes, which we now explain.

As we want to ensure that our translation will be compositional, we will have to dynamically generate the home bases for iterative processes where, in the example IterSearch, the home base and the replicated process were already set up. We will also dynamically generate the registered channel *ping* used to provide to a new instance of the process the name of the location where the recursive call took place. The last thing to do when the recursion is unwound for the first time is to start the iterative process, which means two things: move the code that will be replicated to its home base and fire the first instance. As we explained with the example, the replicated code will just have to wait for the name of a location when the recursion is unwound, go there and behave as the recursive process.

- $\mathrm{unrec}(\mathrm{rec}\ Z : \mathrm{R}.\ P) = (\mathrm{newloc}\ home_Z : \mathrm{loc}[ping_Z : \mathrm{rw}\ \mathrm{R}\ ])$
  $(\mathrm{unrec}(Z) \mid$
  $\mathrm{goto}\ home_Z.$
  $ping_Z\ ?(l : \mathrm{R})\ \mathrm{goto}\ l.\mathrm{unrec}(P))$
- $\mathrm{unrec}(Z) = \mathrm{here}[x]\,\mathrm{goto}\ home_Z.ping_Z\ !\ x$
- $\mathrm{unrec}(u\ !\ V\ \ P) = u\ !\ V\ \ \mathrm{unrec}(P)$; all the other cases are similar.

We stress the fact that this translation heavily relies on migration to mimic the original process. We conjecture that in a Dpi setting where locations or links can fail, like in [FH05], it would not be possible to get a reasonable encoding of recursion into iteration.

We could also give another translation, which would be closer to the one proposed for the pi-calculus in [SW01] by:

- closing the free names of recursive processes, and then communicating their actual values through the channel *ping*, at the same time as the location;
- creating all the home bases at the top-level of the process, once and for all.

So the translation of a system would start by identifying the set of recursion variables: let us write this set $\{Z_i\}$, and their corresponding processes

$\{P_i\}$ when "rec $Z_i : \mathrm{R}_i.\ P_i$" appear in the system. For any process P_i among those we will note $\tilde{n}_i$ its set of free names. Then the components of the system are simply translated the following way:

- $\text{nc-unrec}(Z_i) = \text{here}[x].\text{goto}\, \mathit{home}_{Z_i}.\mathit{ping}_{Z_i}\,!\,x, \tilde{n}_i$
- $\text{nc-unrec}(\text{rec}\, Z_i : \mathrm{R}_i.\ P_i) = \text{nc-unrec}(Z_i)$
- $\text{nc-unrec}(u\,!\,V\ P) = u\,!\,V\ \text{nc-unrec}(P)$; all the other cases are similar.

A system M is then translated, as a whole, into the following process:

$$(\text{new}\, \mathit{ping}_{Z_1})\,(\text{new}\, \mathit{home}_{Z_1})\,(\text{new}\, \mathit{ping}_{Z_2})\,(\text{new}\, \mathit{home}_{Z_2})\ldots$$
$$\mathit{home}_{Z_1}\mathrm{J}\ \ \mathit{ping}_{Z_1}\,?(l : \mathrm{R}_1, \tilde{n}_1).\text{goto}\, l.\text{nc-unrec}(P_1)\mathbb{K}\,|$$
$$\mathit{home}_{Z_2}\mathrm{J}\ \ \mathit{ping}_{Z_2}\,?(l : \mathrm{R}_2, \tilde{n}_2).\text{goto}\, l.\text{nc-unrec}(P_2)\mathbb{K}\,|\ldots|$$
$$\text{nc-unrec}(M)$$

But, of course, such an approach would not be compositional, as the name $\text{nc-unrec}(\cdot)$ suggests.

Now that we have described our translation, we want to prove that the translation and the original process are "equivalent", in some sense. Since we are in a typed setting, the first property we need to check is the following.

Lemma 5.1. $\Gamma \quad M$ *if and only if* $\Gamma \quad \text{unrec}(M)$

Proof. We define the function ϕ over environments:

$$\phi(\Gamma,\ Z_i : \mathrm{R}_i\ \ , u_{i_j} @ Z_i : \mathrm{A}_{i_j}) =$$
$$\Gamma, \mathit{home}_{Z_i} : \text{loc}, \mathit{ping}_{Z_i} @ \mathit{home}_{Z_i} : \text{rw}\ \mathrm{R}_i\ ,\ l_i : \mathrm{R}_i\ \ , u_{i_j} @ l_i : \mathrm{A}_{i_j}$$

ϕ^{-1} is defined as expected.

We now prove the following generalised statement:

- $\Gamma \quad M$ implies $\phi(\Gamma) \quad \text{unrec}(M)$;
- $\Gamma \quad_v P$ implies that $\phi(\Gamma) \quad_v \text{unrec}(i$ [illegible]

The reasoning would be identical for $\Gamma \;{}_{z_j}$ rec $Z_i : \mathrm{R}_i.\ P$ but w would have to be replaced by Z_j when typing in Γ and by I_j when in $\phi(\Gamma)$.

- (t-recvar): $\Gamma \;{}_{w}\; Z$ implies that $\Gamma \quad w : \Gamma(Z)$. Then $\phi(\Gamma) \quad w : \Gamma(Z$

Figure 14 Labelled transition semantics. Internal actions.

(lts-go)
$\Omega \rtimes k[\text{goto } l.P] \xrightarrow{\tau}_\beta \Omega \rtimes l[P]$

(lts-split)
$\Omega \rtimes k[P \mid Q] \xrightarrow{\tau}_\beta \Omega \rtimes k[P] \mid k[Q]$

(lts-iter)
$\Omega \rtimes k[\ \ P] \xrightarrow{\tau}_\beta \Omega \rtimes k[\ \ P] \mid k[P]$

(lts-here)
$\Omega \rtimes k[\text{here}\,[x]\,P] \xrightarrow{\tau}_\beta \Omega \rtimes k[P\,[{}^{k}/_{x}]]$

(lts-rec)
$\Omega \rtimes k[\text{rec } (Z : \mathrm{R}).\ P] \xrightarrow{\tau}_\beta \Omega \rtimes k[P\,\{^{\text{rec } (Z:\mathrm{R}).\ P}/_{Z}\}]$

(lts-l-create)
$\Omega \rtimes k[(\text{newloc } l : \mathrm{L})\,P] \xrightarrow{\tau}_\beta \Omega \rtimes (\text{new } l : \mathrm{L})\,k[P]$

(lts-c-create)
$\Omega \rtimes k[(\text{newc } c : \mathrm{C})\,P] \xrightarrow{\tau}_\beta \Omega \rtimes (\text{new } c@k : \mathrm{C})\,k[P]$

(lts-eq)
$\Omega \rtimes k[\text{if } u = u \text{ then } P \text{ else } Q] \xrightarrow{\tau}_\beta \Omega \rtimes k[P]$

(lts-neq)
$\Omega \rtimes k[\text{if } u = v \text{ then } P \text{ else } Q] \xrightarrow{\tau}_\beta \Omega \rtimes k[Q] \qquad \text{when } u = v$

(lts-comm)
$$\frac{\Omega_M \rtimes M \xrightarrow{(\tilde{n}:\tilde{\mathrm{T}})k.a!V} \Omega_M \rtimes M \qquad \Omega_N \rtimes N \xrightarrow{(\tilde{n}:\tilde{\mathrm{U}})k.a?V} \Omega_N \rtimes N}{\Omega \rtimes M \mid N \xrightarrow{\tau} \Omega \rtimes (\text{new } \tilde{n} : \tilde{\mathrm{T}})\,M \mid N \quad \Omega \rtimes N \mid M \xrightarrow{\tau} \Omega \rtimes (\text{new } \tilde{n} : \tilde{\mathrm{T}})\,N \mid M} \quad \tilde{n} \cap \mathrm{fn}(N) =$$

- *for every u but recursion variables in* $\mathrm{dom}(\Gamma)$ *we have* $\Gamma(u) <: \Gamma(u)$*;*
- *for every recursion variable Z in* $\mathrm{dom}(\Gamma)$ *we have* $\Gamma(Z) = \Gamma(Z)$. ■

Definition 5.3 (Configurations). *We call* configuration *a tuple of an environment Ω and a system M, written $\Omega \rtimes M$, such that there exists an environment Γ, with $\Gamma <: \Omega$ and $\Gamma \vdash M$.* ■

The reader is referred to [HMR03] for the formal details.

Theorem 5.4. *Suppose $\Gamma \vdash M$. Then $\Gamma \models M =_{rbc} \mathrm{unrec}(M)$.*

The proof uses a characterisation of this relation as a bisimulation equivalence in a labelled transition system in which:

- the states are configurations;
- the actions take the form $\Omega \rtimes M \xrightarrow{\mu} \Omega \rtimes M$; these are based on the labelled transitions system given in Figure 14 and 15.

Figure 15

6 Proof of recursion implementability

Let us hint the problems encountered in trying to prove the equation (4) on an example. For this, let us consider a parameterised server version of our Search process that would be exploring a binary tree instead of a list:

PSearch $\triangleq$ *search*_

lines of that in [SW01], would give rise to the following state, corresponding to (5) above:

$$(\text{new}\ \mathit{ping})\ (\text{new}\ \mathit{base})\ \text{Server}\langle\ \mathit{search\_req}?(x, \mathit{client})\ \text{goto}\ k_0.\mathrm{F}(x, \mathit{client})\rangle$$
$$\mid \mathit{base}\langle\ \mathit{ping}?(k, x, \mathit{client})\ \text{goto}\ k.\mathit{test}?(y)\ldots\rangle$$
$$\mid k_1^1\langle\ldots\mathrm{F}(x_1, \mathit{client}_1)\rangle \mid k_1^2\langle\ldots\mathrm{F}$$

Definition 6.2 (Residual). *We call* residual *of an occurrence o in M after a reduction $\Omega \rhd M \xrightarrow{\mu} \Omega \rhd M$ the occurrence defined by the following function:*

- $\mathrm{Res}(\varepsilon, \Omega \rhd M \xrightarrow{\mu} \Omega \rhd M) = \varepsilon$
- $\mathrm{Res}(1o, \Omega \rhd M \mid N \xrightarrow{\mu} \Omega \rhd M \mid N) = \mathrm{Res}(o, \Omega \rhd M \xrightarrow{\mu} \Omega \rhd M)$
- $\mathrm{Res}(2o, \Omega \rhd M \mid N \xrightarrow{\mu} \Omega \rhd M \mid N) = 2o$
- $\mathrm{Res}(0, \Omega \rhd k\langle a!V\, P\rangle \xrightarrow{k.a!V} \Omega \rhd k\langle O\rangle) =$
- $\mathrm{Res}(00o, \Omega \rhd k\langle a!V\, P\rangle \xrightarrow{k.a!V} \Omega \rhd k\langle O\rangle) = 0o$
- $\mathrm{Res}(0o, \Omega \rhd ($

This will therefore heavily rely on implicit α-conversions.

- if $o0$ is in P, then it must be in some O in P;

$$\begin{array}{l}\mathrm{unrec}^{o}_{P}(k\mathrm{Jrec}\ Z : \mathrm{R}.\ P\mathrm{K}) = \\ \quad home_{Z_O}\mathrm{J}ping_{Z_O}\ ?(l : \mathrm{R})\,\mathrm{goto}\ l.\mathrm{unrec}^{o00}_{P}(P)\mathrm{K} \\ \quad\quad |\, home_{Z_O}\mathrm{J}ping_{Z_O}\ !\ k\ \mathrm{K}\end{array}$$

- if o is in P, then it must be in some O in P, when the previous case cannot apply;

$$\mathrm{unrec}^{o}_{P}(\mathrm{rec}\ Z : \mathrm{R}.\ P) = \mathrm{here}[x]\,\mathrm{goto}\ home_{Z_O}.ping_{Z_O}\ !\ x$$

- we write o for the occurrence of the binder of the occurrence o of Z; if o is in P, then it must be in some O in P and Z must be "Z_O";

$$\mathrm{unrec}^{o}_{P}(Z) = \mathrm{here}[x]\,\mathrm{goto}\ home_{Z_O}.ping_{Z_O}\ !\ x$$

- we write o for the occurrence of the binder of the occurrence o of Z; if o is not in P:

$$\mathrm{unrec}^{o}_{P}(Z) = \mathrm{here}[x]\,\mathrm{goto}\ home_{Z}.ping_{Z}\ !\ x$$

- $\mathrm{unrec}^{o}_{P}(u\ !\ V\ \ P) = u\ !\ V\ \ \mathrm{unrec}^{o0}_{P}(P)$; all the other cases for processes are similar;

- if o is the longest system-prefix of the occurrences in $(O_i)\quad P$, we translate the system this way, with n_i the annotation of O_i in P and o_i one occurrence in O_i:

$$\begin{array}{l}\mathrm{unrec}^{o}_{P}((\mathrm{new}\ e : \mathrm{E})\ M) = \\ \quad (\mathrm{new}\ e : \mathrm{E})\,(\mathrm{new}\ home_{Z_{O_1}} : \mathrm{loc}[ping_{Z_{O_1}} : \mathrm{rw}\ \mathrm{R}_{O_1}\ ]) \\ \quad\quad home_{Z_{O_1}}\mathrm{J}\ \ ping_{Z_{O_1}}\ ?(l : \mathrm{R}_{O_1})\,\mathrm{goto}\ l.\mathrm{unrec}^{o_i 0}_{P}(M|_{o_i 0})\mathrm{K} \\ \quad\quad\quad \vdots \times n_1 \\ \quad\quad |\, home_{Z_{O_1}}\mathrm{J}\ \ ping_{Z_{O_1}}\ ?(l : \mathrm{R}_{O_1})\,\mathrm{goto}\ l.\mathrm{unrec}^{o_i 0}_{P}(M|_{o_i 0})\mathrm{K} \\ \quad\quad |\, home_{Z_{O_2}}\mathrm{J}...\mathrm{K} \\ \quad\quad\quad \vdots \\ \quad\quad |\, \mathrm{unrec}^{o0}_{P}(M)\end{array}$$

All other cases for system are similar, with the "generation" of all the home-bases that are required at that occurrence before the inductive case.

Notice that, up-to congruence for the order between the different locations $home_Z$ introduced by the last case of the definition, $\mathrm{unrec}^{o}_{P}(k\mathrm{Jrec}\ Z : \mathrm{R}.\ P\mathrm{K})$ when $o0$ is not in P is equal to $\mathrm{unrec}^{o}_{P\ \{\{o0\}^1\}}(k\mathrm{Jrec}\ Z : \mathrm{R}.\ P\mathrm{K})$.

Of course, we extend the notion of residual of an occurrence to the one of residual of a set P.

We write $\mathsf{unrec}_P(M)$ for $\mathsf{unrec}^{\varepsilon}_P(M)$. Note that we do not need a special case for the translation of $k\llbracket Z \rrbracket$ since we know that this is an impossible situation.

To deal with the extra steps introduced by the translation, we will resort to a proof technique given in [JR04], namely bisimulation up-to-β. This is based on the remark that, among the reductions added by the translation, only the communication on the channel *ping* is "dangerous", because it could fail if one of the two agents involved in the communication were absent. Every other step is a so-called β-move, written $\xrightarrow{\tau}_{\beta}$ in the LTS, in Figure 14. Thanks to bisimulations up-to-β we can focus only on the communication moves. Then we can consider that the *ping*-communication (which is a τ-move) in the translation corresponds to the recursion unwinding in recDpi.

Lemma 6.3 (unrec() is an bisimulation). *Suppose an environment* Γ *and a system* M*. Then* $\Gamma \vdash M$ *implies* $(\Gamma \natural M) \approx_{bis} (\Gamma \natural \mathsf{unrec}(M))$

(Its-go). The term it reaches is

$$(\text{new}\, home_{Z_{o_1}})\,(\text{new}\, home_{Z_{o_2}})\ldots \\ home_{Z_{o_1}}\langle\!\langle \ldots \rangle\!\rangle \mid \ldots \mid \langle\!\langle \text{unrec}_P^{o00}(P) \rangle\!\rangle$$

which might need some extra β-reductions to become the translation of $M\mid_o = \langle\!\langle P \rangle\!\rangle$ because there are different possible cases for the form of P. If P is of the form $\text{rec}\, Z : \text{R}.\, P$:

if $o\,0$, the occurrence for the recursion operator, is in P then it must be in some set O in P and $\text{unrec}_P^o\,(M\mid_o)$ is

$$(\text{new}\, home_{Z_{o_1}}) \ldots home_{Z_o}\langle\!\langle ping_{Z_o} ! \langle\!\langle \rangle\!\rangle \rangle\!\rangle \\ \mid home_{Z_o}\langle\!\langle ping_{Z_o} ?(l : \text{R})\,\text{goto}\, l.\text{unrec}_P^{o\,00}(P) \rangle\!\rangle$$

but, we will take P to be the residual of P after the move so that $o\,0$ is in P exactly when $o00$ was in a set O in P. This implies that $\langle\!\langle \text{unrec}_P^{o00}(P) \rangle\!\rangle$ is of the form

$$\langle\!\langle \text{here}\,[x]\,\text{goto}\, home_{Z_o}.ping_{Z_o} ! x \rangle\!\rangle$$

which reduces by β-moves to $home_{Z_o}\langle\!\langle ping_{Z_o} ! x \rangle\!\rangle$. We also know by definition of the translation $\text{unrec}_P(M)$ that at the longest common system-prefix among occurrences in O is generated the server in the home-base:

$$(\text{new}\, home_{Z_o})\, home_{Z_o}\langle\!\langle \; ping_{Z_o} ?(l : \text{R})\,\text{goto}\, l.\text{unrec}_P^{o00}(P) \rangle\!\rangle$$

so one β-move generates a new instance of the replicated process

$$home_{Z_o}\langle\!\langle \; ping_{Z_o} ?(l : \text{R})\,\text{goto}\, l.\text{unrec}_P^{o00}(P) \rangle\!\rangle$$

which is exactly the system we need. And we can put this new instance by o by congruence.

if $o\,0$ is not in P , we know that the translation we will give will be of the form

$$(\text{new}\, home_{Z_{\{o\,0\}}} : \text{loc}[ping_{Z_{\{o\,0\}}} : \text{rw R}_{\{o\,0\}}\,]) \\ home_{Z_{\{o\,0\}}}\langle\!\langle \; ping_{Z_{\{o\,0\}}} ?(l : \text{R})\,\text{goto}\, l.\text{unrec}_P^{o\,00}{}_{\{\{o\,0\}\}}(P) \rangle\!\rangle \\ \mid home_{Z_{\{o\,0\}}}\langle\!\langle ping_{Z_{\{o\,0\}}} ?(l : \text{R})\,\text{goto}\, l.\text{unrec}_P^{o\,00}{}_{\{\{o\,0\}\}}(P) \rangle\!\rangle \\ \mid home_{Z_{\{o\,0\}}}\langle\!\langle ping_{Z_{\{o\,0\}}} ! k \rangle\!\rangle$$

but in that case, we will have $o00$ not in P so $\langle\!\langle \text{unrec}_P^{o00}(P) \rangle\!\rangle$ will be of the form

$$\langle\!\langle (\text{newloc}\, home_Z : \text{loc}[ping_Z : \text{rw R}\,]) \\ (\text{unrec}(Z) \mid \\ \text{goto}\, home_Z. \quad ping_Z ?(l : \text{R})\,\text{goto}\, l.\text{unrec}(P)) \rangle\!\rangle$$

so by (lts-l-create), (lts-split), (lts-here), (lts-go) and (lts-iter) this reduces by β-moves into the translation of $M\ |_o$.

Otherwise, if P is not of the form rec Z : R. P , we know that it cannot be of the simple form Z, since Z would in that case be a free recursion variable in the system. So it must be one of the various possible cases for processes. If we take the example of a

of P for P , we get

$$\mathrm{unrec}^{o}_{P}\ (k\llbracket P\{^{\mathrm{rec}\ (Z:\mathrm{R}).\ P}/_{Z}\}\rrbracket) = \mathrm{unrec}^{o}_{P}\ (k\llbracket P\rrbracket)$$

As in the case for rule (lts-go), showing the adequation between this translation and $k\llbracket \mathrm{unrec}^{o00}_{P}(P)\rrbracket$ turns out to be a simple case analysis on the form of P.

– (lts-comm): $M|_{o} = M_1 \mid M_2$ and there exists some Ω_1 and Ω_2 such that $\Omega_1 \,\natural\, M|_{o1} \xrightarrow{(\tilde{n}:\Theta)k.a!V} \Omega_1 \,\natural\, M|_{o\ 1}$ and $\Omega_2 \,\natural\, M|_{o2} \xrightarrow{(\tilde{n}:\Theta)k.a?V} \Omega_2 \,\natural\, M|_{o\ 2}$. By our induction hypothesis, we can conclude that, writing P for the residual of P after the communication move

$$\Omega_1 \,\natural\, \mathrm{unrec}^{o1}_{P}(M|_{o1}) \xrightarrow{(\tilde{n}:\Theta)k.a!V} \twoheadrightarrow_{\beta} \equiv \Omega_1 \,\natural\, \mathrm{unrec}^{o\ 1}_{P}(M|_{o\ 1})$$

and

$$\Omega_2 \,\natural\, \mathrm{unrec}^{o2}_{P}(M|_{o2}) \xrightarrow{(\tilde{n}:\Theta)k.a?V} \twoheadrightarrow_{\beta} \equiv \Omega_2 \,\natural\, \mathrm{unrec}^{o\ 2}_{P}(M|_{o\ 2})$$

which implies

$$\begin{aligned}
&\Omega \,\natural\, \mathrm{unrec}^{o}_{P}(M|_{o}) \twoheadrightarrow \twoheadrightarrow_{\beta} \\
&\quad \equiv \Omega \,\natural\, (\mathrm{new}\,\tilde{n}:\Theta)\,\mathrm{unrec}^{o\ 1}_{P}(M|_{o\ 1}) \mid \mathrm{unrec}^{o\ 2}_{P}(M|_{o\ 2}) \\
&\quad = \Omega \,\natural\, (\mathrm{new}\,\tilde{n}:\Theta)\,\mathrm{unrec}^{o}_{P}(M|_{o}) \\
&\quad = \Omega \,\natural\, \mathrm{unrec}^{o}_{P}(M|_{o})
\end{aligned}$$

in M

(ICFP), 2000. Also appears as Chapter 21 of *Types and Programming Languages* by Benjamin C. Pierce (MIT Press, 2002).

[HMR03] Matthew Hennessy, Massimo Merro, and Julian Rathke. Towards a behavioural theory of access and mobility control in distributed systems. *Theoretical Computer Science*, 322:615–669, 2003.

[HR02] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.

[JR04] Alan Je□rey and Julian Rathke. A theory of bisimulation for a fragment of concurrent ml with local names.