Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 180 (2007) 3–19

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

A Preliminary Comparative Study on the Expressive Power of Reo and Linda

Silvia Amaro [1,4]

Dpto. de C. de la Computación, National University of Comahue
Neuquén, Argentina

Ernesto Pimentel[2,5]

Dpto. de Lenguajes y Ciencias de la Computación, University of Málaga
Málaga, Spain

Ana M. Roldan[3,5]

Dpto. de Ing. Electrónica y Sist. Informáticos, University of Huelva
Huelva, Spain

Abstract

Component-based Software Development is an emerging discipline in the field of Software Engineering. In this context, coordination languages may be used to specify the interactive behavior of software components, and most of the proposals presented in the literature are based on shared data-space models, as Linda. On the other hand, a new model for coordination based on communication channels (Reo) is also emerging, and we argue it also can be used to describe component protocols in a very elegant way. Making a comparative analysis on the expressiveness of this channel based model and Linda is the main objective of the present work, which presents a first step to make an exhaustive formal study. Thus, in this paper we provide a couple of modular embeddings for the synchronous case and the asynchronous case by defining a common formalism in order to allow the comparison of both models at an homogeneous level of abstraction. We hope these results will help us to develop a complete study about the Reo's expressiveness, and to define an interaction description language based on Reo for component coordination, as it has already made in the context of Linda.

Keywords: Coordination, process algebra, modular embedding, expressivity.

[1] Email: samaro@uncoma.edu.ar

[2] Email: ernesto@lcc.uma.es

[3] Email: amroldan@diesia.uhu.es

[4] The work of Silvia Amaro has been partially supported by CYTED, proyect VII-J-RITOS2

[5] The work of Ana M. Roldán and E. Pimentel has been partially supported by the Spanish project TIC2001-2705-C03-02

doi:10.1016/j.entcs.2006.10.043

1 Introduction

A number of works on coordination models and languages are centered in software interoperability. Such models and languages have evolved towards data oriented models and control-event oriented models [1]. Recently a new model, called Reo, based on the combination of connectors (constructed as a set of communicating channels) has been defined [2].

Linda [10] is one of the most representative coordination languages, originally presented as a set of inter-agent communication primitives which can be added to virtually any programming language. Linda's communication primitives allow processes to add, delete and test for the presence/absence of tuples in a shared *tuple space*. The tuple space is a multiset of data (tuples), shared by concurrently running processes. Delete and test operations are blocking and follow an associative naming scheme that operates like *select* in relational databases. Reo is a recently introduced channel-based coordination model which enforces the use of connectors for the coordination of concurrent processes or component instances in a component-based system. Channels are the basic connectors from which more complex connectors can be constructed through composition. The channel composition mechanism in addition to the great diversity of channel types (with a well defined behavior) allows the construction of many different connectors, where each connector imposes a specific coordination pattern.

The expressive power of both models (Linda and Reo) has been independently studied by using different approaches. A very complete study on the expressive power of Linda was made by Brogi and Jacquet in [6,7] where different Linda-like concurrent languages were compared by using the notion of modular embedding [5]. On the other hand, when Reo was introduced, Arbab [2] provided a number of examples to show the expressive power of his proposal, simulating in a simple and elegant way different communication mechanisms. The objective of this paper is starting a comparison between both models, making a comparative analysis between Reo and Linda, considering both the asynchronous behavior (inherent to Linda), and the synchronous one. Both models present very different abstraction levels: Linda is based on a set of communication primitives accessing to a shared tuple space, whereas Reo is also defined in terms of communications primitives, but acting on connectors which are constructed as a combination of different kinds of channels. In order to have an homogeneous level of abstraction which allows a fair comparison, we will define two process calculi for each formalism, following the work by [9], and we will compare them.

In the case of Linda, we will define a process calculus $\mathcal{L}$ encoding the primitives and including usual parallel and non-deterministic choice operators, such as it was made in [8]. A similar process calculus $\mathcal{R}$ will be defined for Reo, but in this case, the operational semantics will be parameterized with respect to the connector on which primitives are accessing. Thus, the actual comparison will be made between Linda and several "instantiations" of Reo, and it will be given by the definition of a couple of modular embeddings [5], one for the asynchronous case (considering the *bag* channel in Reo), and the other one for the synchronous case (where a synchronous channel will be used in Reo).

The rest of the paper is organized as follows. In Section 2 we present both interaction models, their semantics, the two process calculi previously mentioned, and the properties defining them. The comparing method and its application to compare both models is presented in Section 3. Finally, we give some conclusions and future work.

2 Shared tuples and channel-based connectors

In the context of coordination models and languages we can identify data-driven oriented languages and control-driven oriented ones. Nowadays, a new channel-based coordination language based on composition of communication channels (introduced by Arbab and others [4]) is appearing. The main characteristic of data-driven coordination models and languages is that the state of the computation at any time is defined in terms of both values of the data received or sent and the actual configuration of the coordinated components. In the case of channel-based coordination models the framework evolves by means of performing communication actions over input or output ends of channels to which the coordinated components are connected.

Linda [10] belongs to the first family (data-driven). It was the first coordination language, originally presented as a set of inter-agent communication primitives which can be added to virtually any programming language. Linda's communication primitives allow processes to add, delete and test for the presence/absence of tuples in a shared *tuple space*. The tuple space is a multiset of data (tuples), shared by concurrently running processes. Reo [2] is a recently introduced channel-based coordination model which enforces the use of connectors for the coordination of concurrent processes or component instances in a component-based system. Channels are the basic connectors from which more complex connectors can be constructed through composition. The channel composition mechanism in addition to the great diversity of channel types (with a well defined behavior given by ordering schemes, buffers, synchronism, end types) allow the construction of many different connectors. Each connector imposes a specific coordination pattern.In this context communication among component instances takes place by means of input and output actions over connector ends, acting as connection points. Connectors are modelled as *abstract behavior types* (ABT)[3]. An ABT defines an abstract behavior as a relation among the observable input/output that occur through a set of connection points. Each connection point is identified as an input or output portal. The expression $R(I_1, I_2, ..., I_k; O_1, O_2, ..., O_h)$ defines an ABT with k input portals, h output portals and the relation R over them. For example the behavior of a synchronous channel (which is one of the basic connectors proposed by Reo) is captured by the following ABT:

$$Sync(\langle \alpha, a \rangle; \langle \beta, b \rangle) \equiv \langle \alpha, a \rangle = \langle \beta, b \rangle$$

This ABT represents the behavior of any entity that produces an output identical to its input and at the same time.

The variety of channel types that can be used in Reo makes possible to construct other coordination models. Consider the *bag* channel type, it has one input end, one output end and an unbounded buffer in which all data items matching the filter of the channel are accepted and kept, as in a multiset. So the write operation is always possible, and a read or take operation over the output end selects in a non-deterministic way one of the items that match the pattern specified. This behavior permits us to simulate the actions of Linda over the store. In a forward section we will prove this relation.

In this work, we present a comparative study on the expressive power of these two models. As Linda and Reo describe coordination issues at two different levels of abstraction, we propose two process calculi $\mathcal{L}$ and $\mathcal{R}$ to define a common formalism including the communication primitives of both models.

2.1 The Linda calculus

Following [9], we shall consider a process algebra $\mathcal{L}$ containing the communication primitives of Linda. These primitives permit to add a tuple (out), to remove a tuple (in), and to test the presence of a tuple (rd) in the shared dataspace. The language $\mathcal{L}$ also includes the standard prefix, choice and parallel composition operators in the style of CCS.

The syntax of $\mathcal{L}$ is formally defined as follows:

$$\begin{aligned} P &::= 0_{\mathcal{L}} \;\mid\; A.P \;\mid\; P+P \;\mid\; P \parallel P \mid recX.P \\ A &::= rd(t) \;\mid\; in(t) \;\mid\; out(t) \end{aligned}$$

where $0_{\mathcal{L}}$ denotes the empty process and t denotes a tuple.

The operational semantics of $\mathcal{L}$ can be modelled by a labelled transition system defined by the rules of Table 1. Notice that the configurations of the transition system extend the syntax of agents by allowing parallel composition of tuples. Formally, the transition system of Table 1 refers to the extended language $\mathcal{L}'$ defined as:

$$P' ::= P \;\mid\; P' \,||_{\mathcal{L}}\, \langle t \rangle$$

Rule $(1)_{\mathcal{L}}$ states that the output operation consists of an internal move (τ_{out}) which creates the tuple $\langle t \rangle$. Rule $(2)_{\mathcal{L}}$ shows that a tuple $\langle t \rangle$ is ready to offer itself to the environment by performing an action labelled $\overline{t}$. Rules $(3)_{\mathcal{L}}$ and $(4)_{\mathcal{L}}$ describe the behavior of the prefixes $in(t)$ and $rd(t)$ whose labels are t, and $\underline{t}$, respectively. Rule $(5)_{\mathcal{L}}$ is the standard rule for choice composition. Rule $(6)_{\mathcal{L}}$ is the standard rule for the synchronization between the complementary actions t and $\overline{t}$: It models the effective execution of an $in(t)$ operation. Rule $(7)_{\mathcal{L}}$ defines the synchronization between two processes performing a transition labelled $\underline{t}$ and $\overline{t}$, respectively. Notice that the process performing $\overline{t}$ is left unchanged, since the read operation $rd(t)$ does not modify the dataspace. The usual rule $(8)_{\mathcal{L}}$ for the parallel operator can be applied with different labels. Following [9], there are no rules for recursion since its semantics is defined by structural axiom $recX.P \equiv P[recX.P/X]$ which applies an unfolding step to a recursively defined process. We also consider the transition

$(1)_{\mathcal{L}}$	$out(t).P \xrightarrow{\tau} \langle t\rangle \parallel_{\mathcal{L}} P$	$(5)_{\mathcal{L}}$	$\dfrac{P \xrightarrow{\alpha} P'}{P +_{\mathcal{L}} Q \xrightarrow{\alpha} P'}$
$(2)_{\mathcal{L}}$	$\langle t\rangle \xrightarrow{\bar{t}} 0$	$(6)_{\mathcal{L}}$	$\dfrac{P \xrightarrow{t} P' \; Q \xrightarrow{\bar{t}} Q'}{P \parallel_{\mathcal{L}} Q \xrightarrow{\tau} P' \parallel_{\mathcal{L}} Q'}$
$(3)_{\mathcal{L}}$	$in(t).P \xrightarrow{t} P$	$(7)_{\mathcal{L}}$	$\dfrac{P \xrightarrow{t} P' \; Q \xrightarrow{\bar{t}}}{P \parallel_{\mathcal{L}} Q \xrightarrow{\tau} P' \parallel_{\mathcal{L}} Q}$
$(4)_{\mathcal{L}}$	$rd(t).P \xrightarrow{t} P$	$(8)_{\mathcal{L}}$	$\dfrac{P \xrightarrow{\alpha} P'}{P \parallel_{\mathcal{L}} Q \xrightarrow{\alpha} P' \parallel_{\mathcal{L}} Q}$

Table 1
Transition system for $\mathcal{L}$.

system closed under the usual structural axioms for parallel or choice operators.

The rules of Table 1 are used to define the set of derivations for a Linda system. Following [9] we consider the output action ($\tau_{\mathcal{L}}$) as a observable transitions. Notice that the above operational characterization of $\mathcal{L}$ employs the so-called *ordered* semantics of the output operation. Namely, when a sequence of outputs is executed, the tuples are rendered in the same order as they are emitted. It is also worth noting that also the store can be seen as a process which is the parallel composition of a number of tuples.

2.2 The Reo calculus

We also propose a process algebra $\mathcal{R}$ based on the communication primitives of Reo. We consider the basic actions to insert an item in a connector (*write*), to remove an item from the connector (*take*) and to capture an item without removing it (*read*). Agents in $\mathcal{R}$ are constructed by means of the prefix operator, the nondeterministic choice and the parallel composition. Formally, the syntax of $\mathcal{R}$ is defined as follows:

$$P ::= 0_{\mathcal{R}} \mid A.P \mid P + P \mid P \parallel P \mid recX.P$$
$$A ::= wr(c, v) \mid tk(c, v) \mid rd(c, v)$$

where $0_{\mathcal{R}}$ denotes the empty process and c denotes an input or output end of a connector. The prefixes *wr*, *tk* and *rd* are shorthand for the basic operations *write*, *take* and *read* respectively. As in Reo communication is possibly only in presence of a connector, in order to define the operational semantics of $\mathcal{R}$ we must consider the semantics of the connector on which actions are acting on.

Following the notion of ABT, we consider a connector C defined as a tuple

$$\mathrm{C} = (I_1, I_2, ..., I_k; O_1, O_2, ..., O_h; \Sigma; \longmapsto_{\mathrm{C}})$$

where I_i, O_j represents input and output ends respectively, Σ is the set of possible states of the connector, and $\longmapsto_{\mathrm{C}}$ is a set of transition rules giving its behavior.

The transitions are of the form:

$$\langle \mathrm{C}, \overline{act} \rangle \overset{\overline{act}_1}{\longmapsto}_{\mathrm{C}} \langle \mathrm{C}', \overline{act}_2 \rangle$$

where C represents the connector C in a state σ and C$'$ represent it in a state σ', –$\sigma, \sigma' \in \Sigma$–, $\overline{act}$ denotes the set of actions which applied in parallel over the ends of C in a state σ may produce a progress on the connector, producing eventually a state change. The set $\overline{act}_1$ denotes the actions actually applied, and $\overline{act}_2$ represents pending actions. These multisets must respond to the relation $\overline{act} = \overline{act}_1 \biguplus \overline{act}_2$. A connector progresses when at least one successful input or output operation is performed on one of its ends.

An example of a simple connector in Reo is given below. It has one input end I_1 and one output end O_1, and presents the same behavior as a channel of type *bag* [2], though it may be in two different states: σ when the buffer is empty, and σ' when the buffer contains data items. We will denote this connector by $LR = (I_1; O_1; \{\sigma, \sigma'\}; \longmapsto_{\mathrm{LR}})$, and consider the following transitions:

(1) $\langle \mathrm{LR}, \{wr(I_1, d)\} \rangle \overset{\{wr(I_1,d)\}}{\longmapsto}_{\mathrm{LR}} \langle \mathrm{LR}', \emptyset \rangle$

(2) $\langle \mathrm{LR}', \{wr(I_1, d)\} \rangle \overset{\{wr(I_1,d)\}}{\longmapsto}_{\mathrm{LR}} \langle \mathrm{LR}', \emptyset \rangle$

(3) $\langle \mathrm{LR}', \{tk(O_1, d)\} \rangle \overset{\{tk(O_1,d)\}}{\longmapsto}_{\mathrm{LR}} \langle \mathrm{LR}', \emptyset \rangle$

(4) $\langle \mathrm{LR}', \{tk(O_1, d)\} \rangle \overset{\{tk(O_1,d)\}}{\longmapsto}_{\mathrm{LR}} \langle \mathrm{LR}, \emptyset \rangle$

(5) $\langle \mathrm{LR}', \{rd(O_1, d)\} \rangle \overset{\{rd(O_1,d)\}}{\longmapsto}_{\mathrm{LR}} \langle \mathrm{LR}', \emptyset \rangle$

The transition system presented in Table 2 describes the computational model of $\mathcal{R}$. Rule $(2)_{\mathcal{R}}$ describe the behavior of the choice operator. In rule $(3)_{\mathcal{R}}$ the operational meaning of the parallel operator is defined in the standard way. The synchronization between two agents intending to perform complementary actions is modelled by rule $(4)_{\mathcal{R}}$. The observable progression of agents will be made following rules $(5)_{\mathcal{R}}$ and $(6)_{\mathcal{R}}$, where the corresponding connector (on which actions are being applied) can proceed by consuming some of the pending actions. Transition $\overset{\overline{act}}{\rightarrowtail}_{\mathrm{C}}$ represents any derivation $\overset{\overline{act}_1}{\longmapsto}_{\mathrm{C}} \overset{\overline{act}_2}{\longmapsto}_{\mathrm{C}} \ldots \overset{\overline{act}_n}{\longmapsto}_{\mathrm{C}}$, such that $\overline{act} = \biguplus_i \overline{act}_i$. As in $\mathcal{L}$, there are no rules for recursion, its semantics is defined by the structural axiom $recX.P \equiv P[recX.P/X]$ which applies an unfolding step to a recursively defined process. Finally, we consider the transition system closed under the structural axioms for parallel and choice operators.

3 Comparing Linda and Reo

In this section, we will compare the languages previously defined, $\mathcal{L}$ and $\mathcal{R}$, taking into account that we will get different versions of $\mathcal{R}$ depending on the connector to be considered. To do this, we first formalize the notion of modular embedding used for the comparison. Then, we analyze the expressiveness of both languages $\mathcal{L}$ and $\mathcal{R}$, in both directions, by defining compilers for each case, and proving the existence of modular embedding between $\mathcal{L}$ and $\mathcal{R}$.

$(1)_{\mathcal{R}}$	$\langle act \cdot P, \mathrm{C}\rangle \xrightarrow{act} \langle P, \mathrm{C}\rangle$
$(2)_{\mathcal{R}}$	$\dfrac{\langle P_1, \mathrm{C}\rangle \xrightarrow{\overline{act}} \langle P_1', \mathrm{C}\rangle}{\langle P_1 + P_2, \mathrm{C}\rangle \xrightarrow{\overline{act}} \langle P_1', \mathrm{C}\rangle}$
$(3)_{\mathcal{R}}$	$\dfrac{\langle P_1, \mathrm{C}\rangle \xrightarrow{\overline{act}} \langle P_1', \mathrm{C}\rangle}{\langle P_1 \parallel P_2, \mathrm{C}\rangle \xrightarrow{\overline{act}} \langle P_1' \parallel P_2, \mathrm{C}\rangle}$
$(4)_{\mathcal{R}}$	$\dfrac{\langle P_1, \mathrm{C}\rangle \xrightarrow{\overline{act_1}} \langle P_1', \mathrm{C}\rangle \; \langle P_2, \mathrm{C}\rangle \xrightarrow{\overline{act_2}} \langle P_2', \mathrm{C}\rangle}{\langle P_1 \parallel P_2, \mathrm{C}\rangle \xrightarrow{\overline{act_1} \uplus \overline{act_2}} \langle P_1' \parallel P_2', \mathrm{C}\rangle}$
$(5)_{\mathcal{R}}$	$\dfrac{\langle P, \mathrm{C}\rangle \xrightarrow{\overline{act}} \langle P', \mathrm{C}\rangle \; \langle \mathrm{C}, \overline{act}\rangle \overset{\overline{act}}{\rightarrowtail}_{\mathrm{C}} \langle \mathrm{C}', \emptyset\rangle}{\langle P, \mathrm{C}\rangle \longrightarrow \langle P', \mathrm{C}'\rangle}$
$(6)_{\mathcal{R}}$	$\dfrac{\langle P_1, \mathrm{C}\rangle \xrightarrow{\overline{act_1}} \langle P_1', \mathrm{C}\rangle \; \langle P_2, \mathrm{C}\rangle \xrightarrow{\overline{act_2}} \; \langle \mathrm{C}, \overline{act}\rangle \overset{\overline{act_1}}{\rightarrowtail}_{\mathrm{C}} \langle \mathrm{C}', \overline{act_2}\rangle}{\langle P_1 \parallel P_2, \mathrm{C}\rangle \longrightarrow \langle P_1' \parallel P_2, \mathrm{C}'\rangle}$

Table 2
Transition system for $\mathcal{R}$

3.1 Comparison method: modular embedding

To compare the expressive power of two languages we use the notion of modular embedding propose by De Boer and Palamidessi in [5], which is an extension of the basic idea of embedding considered by Shapiro in [12], suited for a concurrent framework. Now we summarize the method.

Consider two languages L and L', and assume the semantics mappings (observables) $\mathcal{O} : L \to Obs_L$ and $\mathcal{O}' : L' \to Obs_{L'}$, where Obs_L and $Obs_{L'}$ are some domains. Then we say that L *is more expressive than* L' (or L' *can be embedded into* L), if exists a mapping $\mathcal{C}$ (compiler) from the statements of L' to the statements of L ($\mathcal{C} : L' \to L$) and a mapping $\mathcal{D}$ (decoder) from Obs_L to $Obs_{L'}$ such that for every statement A in $L, \mathcal{D}(\mathcal{O}(\mathcal{C}(A))) = \mathcal{O}'(A)$', i.e. the following diagram commutes:

$$\begin{tikzcd}
L' \arrow[r, "\mathcal{O}'"] \arrow[d, "\mathcal{C}"] & Obs_{L'} \\
L \arrow[r, "\mathcal{O}"] & Obs_L \arrow[u, "\mathcal{D}"']
\end{tikzcd}$$

For the embedding to be modular the coder $\mathcal{C}$ and the decoder $\mathcal{D}$ must satisfy the following three properties:

(i) $\mathcal{D}$ should be defined in an element-wise way with respect to Obs_L , that is

$$\forall X \in Obs_L : \mathcal{D}(X) = \{\mathcal{D}_{el}(x) | x \in X\}$$

for some convenient mapping $\mathcal{D}_{el}$.

(ii) the compiler $\mathcal{C}$ should be defined in a compositional way with respect to every n-ary operator op in the language $\mathcal{R}$:

$$\mathcal{C}(op(A_1, \ldots, A_n)) = \tilde{op}(\mathcal{C}(A_1), \ldots, \mathcal{C}(A_n))$$

(iii) the embedding must preserve the behavior of the original processes with respect to failure and success, this is:

$$\forall X \in Obs_L, \forall x \in X : tm'(\mathcal{D}_{el}(x)) = tm(x)$$

where tm and tm' give information about the termination mode of the observables of L and L', respectively.

3.2 Asynchronous case

In this particular case, we use the notion of modular embedding described in the previous section to study the embedding between $\mathcal{L}$ and $\mathcal{R}$. We will consider a very simple notion of observability for both languages, encoding successful and failure computations. That is, $Obs_{\mathcal{L}} = Obs_{\mathcal{R}} = \wp(\{ss, f\!f\})$.

Definition 3.1 Observables in $\mathcal{L}$ are given by the mapping $O_{\mathcal{L}} : \mathcal{L} \to Obs_{\mathcal{L}}$ which is defined as follows: (1) $ss \in O_{\mathcal{L}}(P)$ if P is a $\mathcal{L}$-process presenting a successful trace (i.e. $P \to^* 0$, or there exist n tuples, $t_1, \ldots, t_n$ such that $P \to^* < t_1 > \| \, .. \, \| < t_n >$), and (2) $f\!f \in O_{\mathcal{L}}(P)$ if P presents a failure trace (i.e. there exits a non-empty process Q, which is not a parallel composition of tuples, such that $P \to^* Q \not\to$.

Similarly, the notion of observables is defined for $\mathcal{R}$.

Definition 3.2 Given a connector C, observables in $\mathcal{R}$ are given by the mapping $O_{\mathcal{R}} : \mathcal{R} \to Obs_{\mathcal{R}}$ which is defined as follows: (1) $ss \in O_{\mathcal{R}}(P)$ if P is a $\mathcal{R}$-process presenting a successful trace (i.e. $\langle P, \mathrm{C} \rangle \to^* \langle 0_{\mathcal{R}}, \mathrm{C}' \rangle$, and (2) $f\!f \in O_{\mathcal{R}}(P)$ if P presents a failure trace (i.e. there exists a non-empty $\mathcal{R}$-process Q such that $\langle P, \mathrm{C} \rangle \to^* \langle Q, \mathrm{C}' \rangle \not\to$).

In this section we work over the asynchronous case. We compare the expressive power of $\mathcal{L}$ and $\mathcal{R}$ in presence of the connector LR introduced in section 2.2. As the connector LR has only one input end (I_1) and one output end (O_1), to simplify the presentation of the compiler we will omit them when $\mathcal{R}$-actions are used. Thus, input actions like $tk(t, O_1)$ and $rd(t, O_1)$ will be denoted by $tk(t)$ and $rd(t)$, respectively, and the output action $wr(t, I_1)$ will be denoted by $wr(t)$.

Definition 3.3 We define the compiler $\mathcal{C} : \mathcal{L} \longrightarrow \mathcal{R}$ by:

$$\begin{aligned}
\mathcal{C}(in(t).P) &=^{def} tk(t).\mathcal{C}(P) \\
\mathcal{C}(out(t).P) &=^{def} wr(t).\mathcal{C}(P) \\
\mathcal{C}(rd(t).P) &=^{def} rd(t).\mathcal{C}(P) \\
\mathcal{C}(P_1 +_{\mathcal{L}} P_2) &=^{def} \mathcal{C}(P_1) +_{\mathcal{R}} \mathcal{C}(P_2)
\end{aligned}$$

$$\mathcal{C}(P_1 \parallel_{\mathcal{L}} P_2) =^{def} \mathcal{C}(P_1) \parallel_{\mathcal{R}} \mathcal{C}(P_2)$$
$$\mathcal{C}(0_{\mathcal{L}}) =^{def} \mathcal{C}(0_{\mathcal{R}})$$

As we mentioned previously the buffer of the bag type in Reo is a multiset as the store in Linda. Each time a tuple is present in the store, ie. $St = \langle t \rangle$, also a data item t is present in the buffer of the connector LR, and if the store is empty, ie. $St = \langle \rangle$, there is no data item in the connector. In both cases we can consider that the connector LR is in a state equivalent to the store, and we indicate this by the expression LR_{st}.

Proposition 3.4 *Given two processes $P, P' \in \mathcal{L}$, if $P \parallel_{\mathcal{L}} St \xrightarrow{\alpha} P' \parallel_{\mathcal{L}} St'$ $(\alpha \neq \overline{t})$ then $\langle \mathcal{C}(P), LR_{St} \rangle \xrightarrow{\alpha_{\mathcal{R}}} \langle \mathcal{C}(P'), LR_{St'} \rangle$, where*

$$\alpha_{\mathcal{R}} = \begin{cases} \tau & \textit{if } \alpha = \tau \\ tk(t) & \textit{if } \alpha = t \\ rd(t) & \textit{if } \alpha = \underline{t} \end{cases}$$

Proof. See appendix 5.1.

The previous result shows the relation among the transition systems of both calculi. Now considering this relation, we can establish the following corollary.

Corollary 3.5 *If P and P' are two processes in $\mathcal{R}$ such that $\langle P, LR_{St} \rangle \xrightarrow{\tau_{\mathcal{R}}} \langle P', LR_{St'} \rangle$, and there exists Q,Q' in $\mathcal{L}$ verifying $\mathcal{C}(Q) = P$, $\mathcal{C}(Q') = P'$, then*

$$Q \parallel St \xrightarrow{\tau_{\mathcal{L}}} Q' \parallel St'$$

Proof. We proof this corollary reasoning as in the proof of the proposition 3.4, by applying structural induction over the transition system of $\mathcal{R}$.

Considering the definition of the compiler $\mathcal{C}$, the observability notion, and the decoder $\mathcal{D}$ that we defined as the identity, we can prove the following theorem.

Theorem 3.6 *The compiler $\mathcal{C}$ is a modular embedding.*

Proof. See appendix 5.1.

Notice that we can define a modular embedding in the opposite direction, considering the same connector. Both compilers allows to claim that considered models based on Linda and Reo are equally expressive when considering the asynchronous case and the connector LR.

3.3 Synchronous case

In order to compare Reo and Linda for the synchronous case, we introduce the connector $RLS = (I_1; O_1; \{\sigma\}; \longmapsto_{\text{RLS}})$. Because of its nature this connector may be only in one state, so RLS $= \mathcal{RLS}'$. Its behavior is given by the following transitions:

(6) $\langle \text{RLS}, \{wr(I_1, d), tk(O_1, d)\}\rangle \overset{\{wr(I_1,d),tk(O_1,d)\}}{\longmapsto}_{\text{RLS}} \langle \text{RLS}, \emptyset\rangle$

(7) $\langle \text{RLS}, \{wr(I_1, d), rd(O_1, d)\}\rangle \overset{\{rd(O_1,d)\}}{\longmapsto}_{\text{RLS}} \langle \text{RLS}, \{wr(I_1, d)\}\rangle$

In this case we restringe the comparison to a subset of $\mathcal{R}$, considering only the communication primitives *wr* and *tk*. We will consider the same notion of observable defined in previous section. To simulate the operations in Reo as operations in Linda, we need to represent the synchronous behavior of Reo in presence of the connector *RLS*. To do this we propose that processes communicate the actions they can offer and then each process accept one of the actions in the store in case it can on his side perform the complementary action. The compilation of parallel and alternative operators in $\mathcal{R}$ is given in terms of the parallel operator in $\mathcal{L}$. In order to get compositionality, we associate to each agent an identifier u which ranges in an arbitrary countable set Id of process identifiers. Again we are in presence of a very simple connector with only one input end and one output end. Thus, we will proceed as in Definition 3.3, missing the name of channels.

Definition 3.7 We define the compiler $\mathcal{C}$ from $\mathcal{R}$ into $\mathcal{L}$ as:

$$\begin{aligned}
\mathcal{C}_u(P_1 \parallel_{\mathcal{R}} P_2) &=^{def} \mathcal{C}_{u1}(P_1) \parallel_{\mathcal{L}} \mathcal{C}_{u2}(P_2)\\
\mathcal{C}_u(P_1 +_{\mathcal{R}} P_2) &=^{def} \mathcal{C}_{u}(P_1) \parallel_{\mathcal{L}} \mathcal{C}_{u}(P_2)\\
\mathcal{C}_u(tk(t).P) &=^{def} out(tk, t, u).(rd(wr, t, v).out(u, v).\\
&\qquad (in(v, u).in(wr, t, v).out(u).\mathcal{C}_{u1}(P) + rd(v).in(u, v).0)\\
&\qquad +rd(u).0)\\
\mathcal{C}_u(wr(t).P) &=^{def} out(wr, t, u).(rd(tk, t, v).out(u, v).\\
&\qquad (in(v, u).in(tk, t, v).out(u).\mathcal{C}_{u1}(P) + rd(v).in(u, v).0)\\
&\qquad +rd(u).0)\\
\mathcal{C}_u(0_{\mathcal{R}}) &=^{def} \mathcal{C}_u(0_{\mathcal{L}})
\end{aligned}$$

In this context, we consider that the parallel composition of two processes in Reo is translated into the parallel composition of two independent processes in Linda, that are identified by different labels. On the other hand, in the case of the nondeterministic choice, the application of the compiler must preserve the identifier of the original process to avoid the synchronization between two branches of the same process. Now we show some useful results to prove the modularity of the embedding.

Lemma 3.8 *Let P, Q be processes in $\mathcal{R}$, if $\langle P, RLS\rangle \xrightarrow{\{wr(t)\}}_{\mathcal{R}} \langle P', RLS\rangle$ and $\langle Q, RLS\rangle \xrightarrow{\{tk(t)\}}_{\mathcal{R}} \langle Q', RLS\rangle$ then $\forall u \in Id_1, v \in Id_2, \exists u' \in (Id_1 - Id_1^*), v' \in (Id_2 - Id_2^*)$ such that*

$$\mathcal{C}_u(P) \parallel_{\mathcal{L}} \mathcal{C}_v(Q) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P') \parallel_{\mathcal{L}} \mathcal{C}_{v'}(Q') \parallel_{\mathcal{L}} \langle u_i\rangle \parallel_{\mathcal{L}} \langle v_j\rangle \parallel_{\mathcal{L}} St'$$

where Id_1^* and Id_2^* represents the subsets of identifiers of processes derived from u and v respectively, $Id_1 \cup Id_2 = Id$, $Id_1 \cap Id_2 = \emptyset$, $u_i \in Id_1^*$ and $v_j \in Id_2^*$

Proof. See appendix 5.2.

Lemma 3.9 *Let P be a process in $\mathcal{R}$, if $\langle P, RLS\rangle \xrightarrow{\{wr(t),tk(t)\}}_{\mathcal{R}} \langle P', RLS\rangle$ then $\forall u \in Id, \exists u' \in Id^*$ such that*

$$\mathcal{C}_u(P) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P') \parallel_{\mathcal{L}} St'$$

Proof. The proof is by induction over the complexity of processes in $\mathcal{R}$. See appendix 5.2.

Proposition 3.10 *Given two processes P, P' in $\mathcal{R}$ and the connector RLS, if $\langle P, RLS\rangle \longrightarrow_{\mathcal{R}} \langle P', RLS'\rangle$ then $\mathcal{C}(P) \rightarrow^*_{\mathcal{L}} \mathcal{C}(P') \parallel St'$*

Proof. We will analyze rule $5_{\mathcal{R}}$. See appendix 5.2.

Corollary 3.11 *If P and P' are two processes in $\mathcal{L}$ such that $P \parallel St \xrightarrow{\tau_{\mathcal{L}}} P' \parallel St'$, and $\exists$ Q and Q' in $\mathcal{R}$ that verifies $\mathcal{C}(Q) = P$, $\mathcal{C}(Q') = P'$ then*

$$\langle Q, RLS_{St}\rangle \longrightarrow \langle Q', RLS_{St'}\rangle$$

Considering the definition of the compiler $\mathcal{C}$, the sets of observables and the decoder $\mathcal{D}$ that we defined as the identity, we can prove the following theorem.

Theorem 3.12 *The compiler $\mathcal{C}$ is a modular embedding*

Proof. See appendix 5.2.

4 Conclusions

In this paper, we have presented some preliminary results on the comparison of the expressive power of Linda and Reo. To do this, firstly we defined a common formalism to work at the same level of abstraction, and then we study the asynchronous and synchronous cases using specific connectors of Reo, showing that both languages are equivalent for these connectors. Thus, the complexity embedded in the compiler of the synchronous case suggests that Reo is more expressive than Linda in presence of connectors imposing special communication patterns, because of the possible merging of synchronous and asynchronous behaviors. Thus, we can find some connectors for which Reo is strictly more expressive than Linda.

Our future work will be devoted to make a more detailed study of the expressive power of Reo, providing an exhaustive analysis of more complex connectors and different variations of Linda-like languages, and also considering more sophisticated notions of observables, like finite sequences of actions corresponding to successful terminating computations or finite sequences that represent failure situations.

References

[1] F. Arbab. The IWIM model for coordination of concurrent activities. In *First International Conference on Coordination Models, Languajes and Applications (Coordination'96)*,1061 Lecture Notes

in Computer Science, pages 34–56, 1996.

[2] F. Arbab. A Channel-based Coordination Model for Component Composition. Electronic Notes in Theorethical Computer Science, 68(3), 2003.

[3] F. Arbab. Abstract behavior types: a foundation model for components and their composition. *CWI Report SEN- R0305 ISSN 1386-3711* , 2003.

[4] F. Arbab and J.J.M.M. Rutten. A coinductive calculus of component connectors. *CWI Report SEN-R0216 ISSN 1386-369X* , 2002.

[5] F.S. de Boer and C. Palamidesi. Embedding as a tool for language comparision. *In Information and Computation*, 108(1):128-157, 1991.

[6] A. Brogi, N. Busi, M. Gabbrielli and G. Zavattaro Comparative analysis of the expressiveness of shared dataspace coordination. Electronic Notes in Theorethical Computer Science, 62,2001.

[7] A. Brogi and J. M. Jacquet. On the Expressiveness of Linda-like Concurrent Languages. Electronic Notes in Theoretical Computer Science, 16, 1998.

[8] A. Brogi, E. Pimentel, and A. Roldán. Compatibility of Linda-based Component Interfaces. Electronic Notes in Theoretical Computer Science, 66(4), 2002.

[9] N. Busi, R. Gorrieri, and G. Zavattaro. Comparing three semantics for Linda-like languages. Theoretical Computer Science, 240(1): 49-90, 2000.

[10] N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, 32(4):444-458, 1989.

[11] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7:1 (1985), pp.80-112.

[12] E.Y. Shapiro. Embedding among Concurrent Programming Languages. In *Proceedings of CONCUR'92*, pages 486-503, Springer Verlag, 1992.

5 Appendix

In this Appendix we show the proofs of the results presented in the paper.

5.1 *Asynchronous case*

Proposition 3.4 Given two processes $P, P' \in \mathcal{L}$, if $P \parallel_{\mathcal{L}} St \xrightarrow{\alpha} P' \parallel_{\mathcal{L}} St'$ $(\alpha \neq \overline{t})$ then $\langle \mathcal{C}(P), LR_{St}\rangle \xrightarrow{\alpha_{\mathcal{R}}} \langle \mathcal{C}(P'), LR_{St'}\rangle$, where

$$\alpha_{\mathcal{R}} = \begin{cases} \tau & \text{if } \alpha = \tau \\ tk(t) & \text{if } \alpha = t \\ rd(t) & \text{if } \alpha = \underline{t} \end{cases}$$

Proof. The proof is by structural induction. First we analyze the base cases that are identified with the single rules of the transitions system of $\mathcal{L}$:

(i) *Base Cases*

  (a) Consider the rule $(1)_{\mathcal{L}}$. Let $P_1 \parallel St \xrightarrow{\tau} P_1' \parallel St'$, with $P_1 = out(t).P$ and $St = \emptyset$. So, we have that $out(t).P \xrightarrow{\tau} \langle t\rangle \parallel P$, where $P = P_1'$ and $St' = \langle t\rangle$. We want to show that $\langle wr(t).\mathcal{C}(P), LR_\emptyset\rangle \xrightarrow{\tau} \langle \mathcal{C}(P), LR_{\langle t\rangle}\rangle$. By definition of the compiler $\mathcal{C}$ and the rule $(1)_{\mathcal{R}}$, we infer $\langle wr(t).\mathcal{C}(P), LR_\emptyset\rangle \xrightarrow{wr(t)} \langle \mathcal{C}(P), LR_\emptyset\rangle$, and considering the rule (1) of the connector LR and the rule $(5)_{\mathcal{R}}$, we obtain
  $\langle wr(t).\mathcal{C}(P), LR_\emptyset\rangle \xrightarrow{\tau} \langle \mathcal{C}(P), LR_{\langle t\rangle}\rangle$.

(b) Let $P_1 \parallel St \xrightarrow{t} P_1' \parallel St$ with $P_1 = in(t).P$, and $St = \emptyset = St'$, so, $in(t).P \xrightarrow{t} P$ and $LR_{St} = LR_{St'}$. Now by definition of $\mathcal{C}$ and the rule $(1)_{\mathcal{R}}$, we obtain $\langle tk(t).\mathcal{C}(P), LR_\emptyset\rangle \xrightarrow{tk(t)} \langle \mathcal{C}(P), LR_\emptyset\rangle$.

(c) Consider $P_1 \parallel St \xrightarrow{t} P_1' \parallel St'$, being $P_1 = rd(t).P$. Then, $rd(t).P \xrightarrow{t} P$ and $St = \emptyset = St'$ and $LR_{St} = LR_{St'}$. Finally, following with a similar reasoning to the previous case, we obtain $\langle rd(t).\mathcal{C}(P), LR_\emptyset\rangle \xrightarrow{rd(t)} \langle \mathcal{C}(P), LR_\emptyset\rangle$.

(ii) Now, we analyze the rest of the rules that describe the transition system in $\mathcal{L}$. The inductive hypothesis supposes that the proposition is true for every antecedent of the considered rules.

(a) Analyze the rule $(5)_{\mathcal{L}}$. Then, $(P +_{\mathcal{L}} Q) \parallel St \xrightarrow{\alpha} P' \parallel St'$. We want to prove that $\langle \mathcal{C}(P +_{\mathcal{L}} Q), LR_{St}\rangle \xrightarrow{\alpha_R} \langle \mathcal{C}(P'), LR_{St'}\rangle$. Considering the inductive hypothesis we have $\langle \mathcal{C}(P), LR_{St}\rangle \xrightarrow{\alpha_R} \langle \mathcal{C}(P'), LR_{St'}\rangle$.

If $\alpha = \tau$ then $\alpha_{\mathcal{R}} = \tau$, and this mean that $\exists \overline{act} \neq \emptyset$ such that $\langle \mathcal{C}(P), LR_{St}\rangle \xrightarrow{\overline{act}} \langle \mathcal{C}(P'), LR_{St}\rangle$, and $\langle LR_{St}, \overline{act}\rangle \overset{\overline{act}}{\longmapsto}_{\mathrm{LR}} \langle LR_{St'}, \emptyset\rangle$. So applying rule $(2)_{\mathcal{R}}$, the previous one over the connector and rule $(5)_{\mathcal{R}}$ we obtain that $\langle \mathcal{C}(P +_{\mathcal{R}} Q), LR_{St}\rangle \xrightarrow{\alpha} \langle \mathcal{C}(P' +_{\mathcal{R}} Q), LR_{St'}\rangle$.

If $\alpha \neq \tau$ then $St = St'$, so by applying rule $(2)_{\mathcal{R}}$ and the compiler definition we obtain $\langle \mathcal{C}(P +_{\mathcal{R}} Q), LR_{St}\rangle \xrightarrow{\alpha_{\mathcal{R}}} \langle \mathcal{C}(P' +_{\mathcal{R}} Q), LR_{St'}\rangle$.

(b) Consider the rule $(6)_{\mathcal{L}}$. Then $(P \parallel Q) \parallel St \xrightarrow{\tau} (P' \parallel Q') \parallel St'$ and we want to prove $\langle \mathcal{C}(P \parallel Q), LR_{St}\rangle \xrightarrow{\tau} \langle \mathcal{C}(P' \parallel Q'), LR_{St'}\rangle$. Suppose $St = St_p \parallel St_q$ and the rule $(6)_{\mathcal{L}}$ we have: $P \parallel St_p \xrightarrow{t} P' \parallel St_p'$ and $Q \parallel St_q \xrightarrow{\bar{t}} Q' \parallel St_q'$, where $St_p' = St_p$, $Q = Q'$ and $St_q = St_q' \parallel \langle t\rangle$.

By inductive hypothesis $\langle \mathcal{C}(P), LR_{St_p}\rangle \xrightarrow{tk(t)} \langle \mathcal{C}(P'), LR_{St_p}\rangle$. Thus, by applying rule $(3)_{\mathcal{R}}$ we can affirm that $\langle \mathcal{C}(P) \parallel \mathcal{C}(Q), LR_{St}\rangle \xrightarrow{tk(t)} \langle \mathcal{C}(P') \parallel \mathcal{C}(Q), LR_{St}\rangle$. On the other hand the connector LR is in a state equivalent to $St_p \parallel St_q' \parallel \langle t\rangle$, so it is possible to apply transition (3) or (4) of LR and then, using the rule $(5)_{\mathcal{R}}$ we obtain $\langle \mathcal{C}(P) \parallel \mathcal{C}(Q), LR_{St}\rangle \xrightarrow{tk(t)} \langle \mathcal{C}(P') \parallel \mathcal{C}(Q), LR_{St'}\rangle$. Finally, by definition of the compiler and the fact that $Q = Q'$, results that $\langle \mathcal{C}(P \parallel Q), LR_{St}\rangle \xrightarrow{\tau} \langle \mathcal{C}(P' \parallel Q'), LR_{St'}\rangle$.

(c) Suppose the rule $(7)_{\mathcal{L}}$, then $(P \parallel Q) \parallel St \xrightarrow{\tau} (P' \parallel Q) \parallel St'$. As in the previous case, we consider $St = St_p \parallel St_q$ and so, $P \parallel St_p \xrightarrow{t} P' \parallel St_p'$ y $Q \parallel St_q \xrightarrow{\bar{t}} Q' \parallel St_q'$, where $St_p' = St_p$, $Q = Q'$ and $St_q' = St_{q'} \parallel \langle t\rangle$. By inductive hypothesis we have $\langle \mathcal{C}(P), LR_{St_p}\rangle \xrightarrow{tk(t)} \langle \mathcal{C}(P'), LR_{St_p}\rangle$, and the connector LR is in a state in which transition 3 can be applied. Following the same reasoning, considering rules $(3)_{\mathcal{R}}$, (5) of LR and $(5)_{\mathcal{R}}$ and the definition of $\mathcal{C}$ we obtain $\langle \mathcal{C}(P \parallel Q), LR_{St}\rangle \xrightarrow{\tau} \langle \mathcal{C}(P' \parallel Q), LR_{St'}\rangle$.

(d) Finally we analyze the rule $(8)_{\mathcal{L}}$. If $(P \parallel Q) \parallel_{\mathcal{L}} St \xrightarrow{\alpha} (P' \parallel Q) \parallel St'$ and $\alpha \neq \bar{t}$, we want to prove that $\langle \mathcal{C}(P \parallel Q), LR_{St}\rangle \xrightarrow{\alpha_R} \langle \mathcal{C}(P' \parallel Q), LR_{St'}\rangle$. We prove this reasoning as in cases (b) and (c), and by the application of the rule $(3)_{\mathcal{R}}$. If $\alpha = \overline{t}$, we reason as in case (a).

Theorem 3.6 The compiler $\mathcal{C}$ is a modular embedding.

Proof. The properties (1) y (3) of 3.1 are satisfied directly, considering the notion of observables and that $\mathcal{D}$ is the identity . (2) is evident observing that the compiler preserves the operators. In order to prove (4), we need to verify that $\mathcal{O}(\mathcal{C}(A))) = \mathcal{O}'(A)$. We will proceed proving the double inclusion considering the possible elements in the sets of observables: *ss* y *ff*.

a. $\mathcal{O}'(A) \subseteq \mathcal{O}(\mathcal{C}(A))$

i. If $ss \in O'(A)$, then $A \rightarrow^*_{\mathcal{L}} 0_{\mathcal{L}}$. In this case, we have a sequence similar to $A \rightarrow_{\mathcal{L}} A_1 \rightarrow_{\mathcal{L}} A_2 \rightarrow_{\mathcal{L}} ....A_n \rightarrow_{\mathcal{L}} 0_{\mathcal{L}}$, where $A_i = B_i \parallel St_i$. We want to prove $ss \in \mathcal{O}(\mathcal{C}(A))$, i.e. $\langle \mathcal{C}(A), \mathrm{LR}\rangle \rightarrow^*_{\mathcal{R}} \langle 0_{\mathcal{R}}, LR_{n+1}\rangle$. In this case, we have the sequence $\langle \mathcal{C}(A), \mathrm{LR}\rangle \rightarrow_{\mathcal{R}} \langle \mathcal{C}(B_1), \mathrm{LR}_1\rangle \rightarrow_{\mathcal{R}} \langle \mathcal{C}(B_2), \mathrm{LR}_2\rangle \rightarrow_{\mathcal{R}} ..... \rightarrow_{\mathcal{R}} \langle 0_{\mathcal{R}}, \mathrm{LR_{n+1}}\rangle$. We only need to prove that for each transition $A_j \rightarrow_{\mathcal{L}} A_k$ in $\mathcal{L}$ there is one transition $\langle \mathcal{C}(B_j), \mathrm{LR_j}\rangle \rightarrow_{\mathcal{R}} \langle \mathcal{C}(B_k), \mathrm{LR_k}\rangle$ in $\mathcal{R}$. Immediate by proposition 3.4.

ii. If $ff \in \mathcal{O}'(A)$ then $\exists Q \neq 0$. $A \rightarrow^*_{\mathcal{L}} Q \not\longmapsto$, . We assume $Q = Q' \parallel St$. In this situation, we want to obtain $\langle \mathcal{C}(A), \mathrm{LR}_{\emptyset}\rangle \rightarrow^*_{\mathcal{R}} \langle \mathcal{C}(Q'), \mathrm{LR}'\rangle \not\rightarrow_{\mathcal{R}}$. We prove it by reduction to the absurd. We suppose that $\exists S$. $\langle \mathcal{C}(Q'), \mathrm{LR}'\rangle \rightarrow_{\mathcal{R}} \langle S, LR''\rangle$, then there are a set of actions $\overline{act}$ which respond to $\langle \mathcal{C}(Q'), LR'\rangle \xrightarrow{\overline{act}} \langle S, LR'\rangle$ and $\langle LR', \overline{act}\rangle \overset{\overline{act}}{\longmapsto}_{\mathrm{LR}} \langle LR'', \emptyset\rangle$. By corollary 3.5 $\exists P$ in $\mathcal{L}$, $P = P' \parallel St$. $\mathcal{C}(P') = S$ y $Q \rightarrow_{\mathcal{L}} P$. Contradiction. So, $\langle \mathcal{C}(Q'), \mathrm{LR}'\rangle \not\rightarrow_{\mathcal{R}}$ and $ff \in \mathcal{O}(\mathcal{C}(A))$

By (i) and (ii), we obtain $\mathcal{O}'(A) \subseteq \mathcal{O}(\mathcal{C}(A)))$

b. $\mathcal{O}(\mathcal{C}(A)) \subseteq \mathcal{O}'(A)$

i. If $ss \in O(\mathcal{C}(A))$, then $\langle \mathcal{C}(A), \mathrm{LR}_{\emptyset}\rangle \rightarrow^*_{\mathcal{R}} \langle 0_{\mathcal{R}}, LR_{n+1}\rangle$.This indicates a sequence similar to $\langle \mathcal{C}(A), \mathrm{LR}\rangle \rightarrow_{\mathcal{R}} \langle B_1, \mathrm{LR}_1\rangle \rightarrow_{\mathcal{R}} \langle B_2, \mathrm{LR}_2\rangle \rightarrow_{\mathcal{R}} ..... \rightarrow_{\mathcal{R}} \langle 0_{\mathcal{R}}, \mathrm{LR_n}\rangle$. We want to prove $ss \in \mathcal{O}'(A)$, i.e., $A \rightarrow^*_{\mathcal{L}} 0_{\mathcal{L}}$. In this case, we have a sequence like $A \rightarrow_{\mathcal{L}} A_1 \rightarrow_{\mathcal{L}} A_2 \rightarrow_{\mathcal{L}} ...A_n \rightarrow_{\mathcal{L}} 0_{\mathcal{L}}$, where $A_i = D_i \parallel St_i$ and $B_i = \mathcal{C}(D_i)$. We prove it following the same reasoning that we consider in a.)i.), by the application of the corollary 3.5.

ii. If $ff \in O(\mathcal{C}(A))$, then $\exists S \neq 0$. $\langle \mathcal{C}(A), \mathrm{LR}_{\emptyset}\rangle \rightarrow^*_{\mathcal{R}} \langle S, \mathrm{LR}'\rangle \not\rightarrow_{\mathcal{R}}$. We want to verify $ff \in O'(A)$, i.e., $\exists Q \neq 0$. $A \rightarrow^*_{\mathcal{L}} Q \not\longmapsto$. We prove it following the same reasoning that we considered in a.)ii.), by the application of the proposition 3.4.

By (i) and (ii), we obtain $\mathcal{O}(\mathcal{C}(A)) \subseteq \mathcal{O}'(A)$.

5.2 Synchronous case

Lemma 3.8 Let P, Q be processes in $\mathcal{R}$, if $\langle P, RLS\rangle \xrightarrow{\{wr(t)\}}_{\mathcal{R}} \langle P', RLS\rangle$ and $\langle Q, RLS\rangle \xrightarrow{\{tk(t)\}}_{\mathcal{R}} \langle Q', RLS\rangle$ then $\forall u \in Id_1, v \in Id_2, \exists u' \in (Id_1 - Id_1^*), v' \in$

$(Id_2 - Id_2^*)$ such that

$$\mathcal{C}_u(P) \parallel_{\mathcal{L}} \mathcal{C}_v(Q) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P') \parallel_{\mathcal{L}} \mathcal{C}_{v'}(Q') \parallel_{\mathcal{L}} \langle u_i \rangle \parallel_{\mathcal{L}} \langle v_j \rangle \parallel_{\mathcal{L}} St'$$

where Id_1^* and Id_2^* represents the subsets of identifiers of processes derived from u and v respectively, $Id_1 \cup Id_2 = Id$, $Id_1 \cap Id_2 = \emptyset$, $u_i \in Id_1^*$ and $v_j \in Id_2^*$

Proof. The proof is by induction over the complexity of processes in $\mathcal{R}$.

(i) Base case: Let $P = wr(t).P'$ and $Q = tk(t).Q'$. We will work over $\mathcal{C}_u(P) \parallel_{\mathcal{L}} \mathcal{C}_v(Q)$.

$\mathcal{C}_u(P) \parallel_{\mathcal{L}} \mathcal{C}_v(Q) \longrightarrow 1_{\mathcal{L}}, 7_{\mathcal{L}}, 1_{\mathcal{L}} \longrightarrow (A \parallel_{\mathcal{L}} \langle wr, t, u\rangle \parallel_{\mathcal{L}} \langle u, v\rangle) \parallel_{\mathcal{L}} (B \parallel_{\mathcal{L}} \langle tk, t, v\rangle \parallel_{\mathcal{L}} \langle v, u\rangle)$ where

$A = in(v,u).in(tk,t,v).out(u).\mathcal{C}_{u'}(P') + rd(u)..in(u,v).0$,

$B = in(u,v).in(wr,t,u).out(v).\mathcal{C}_{v'}(Q') + rd(v).in(v,u).0$.

Now, applying rule $6_{\mathcal{L}}$ and rule $1_{\mathcal{L}}$ results that $\mathcal{C}_u(P) \parallel_{\mathcal{L}} \mathcal{C}_v(Q) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P') \parallel_{\mathcal{L}} \mathcal{C}_{v'}(Q') \parallel_{\mathcal{L}} \langle u\rangle \parallel_{\mathcal{L}} \langle v\rangle \parallel_{\mathcal{L}} St'$

(ii) Inductive hypothesis: the lemma is true for every pair of processes P_i, Q_j in $\mathcal{R}$.

(a) Let $P = P_1 \parallel_{\mathcal{R}} P_2$ and $Q = tk(t).Q'$.

If the rule applied is $3_{\mathcal{R}}$ and P_1 is the process which proceed then $\langle P_1, RLS\rangle \xrightarrow{\{wr(t)\}}_{\mathcal{R}} \langle P_1', RLS\rangle$ and $P' = P_1' \parallel_{\mathcal{R}} P_2$.

Applying the hypothesis over P_1 and Q, $\mathcal{C}_{u_1}(P_1) \parallel_{\mathcal{L}} \mathcal{C}_v(Q) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u_1'}(P_1') \parallel_{\mathcal{L}} \mathcal{C}_{v'}(Q') \parallel_{\mathcal{L}} \langle u_1\rangle \parallel_{\mathcal{L}} \langle v\rangle \parallel_{\mathcal{L}} St'$, and finally by rule $8_{\mathcal{L}}$ and the compiler definition

$\mathcal{C}_u(P_1 \parallel_{\mathcal{R}} P_2) \parallel_{\mathcal{L}} \mathcal{C}_v(Q) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P_1' \parallel_{\mathcal{R}} P_2) \parallel_{\mathcal{L}} \mathcal{C}_{v'}(Q') \parallel_{\mathcal{L}} St'$. Then $\mathcal{C}_u(P) \parallel_{\mathcal{L}} \mathcal{C}_v(Q) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P') \parallel_{\mathcal{L}} \mathcal{C}_{v'}(Q') \parallel_{\mathcal{L}} St'$

(b) Let $P = P_1 +_{\mathcal{R}} P_2$ and $Q = tk(t).Q'$

If the rule applied is $2_{\mathcal{R}}$ and P_1 is the process which proceed then $\langle P_1, RLS\rangle \xrightarrow{\{wr(t)\}}_{\mathcal{R}} \langle P_1', RLS\rangle$ and $P' = P_1'$.

On the other hand $\mathcal{C}_u(P) \parallel_{\mathcal{L}} \mathcal{C}_v(Q) = \mathcal{C}_u(P_1 + P_2) \parallel_{\mathcal{L}} \mathcal{C}_v(Q) = \mathcal{C}_u(P_1) \parallel_{\mathcal{L}} \mathcal{C}_u(P_2) \parallel_{\mathcal{L}} \mathcal{C}_v(Q)$.

Applying the hypothesis over P_1 and Q, $\mathcal{C}_u(P_1) \parallel_{\mathcal{L}} \mathcal{C}_v(Q) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P_1') \parallel_{\mathcal{L}} \mathcal{C}_{v'}(Q') \parallel_{\mathcal{L}} St'$, $(St' = \langle u\rangle \parallel \langle v\rangle)$,

and finally by rule $8_{\mathcal{L}}$

$\mathcal{C}_u(P_1) \parallel_{\mathcal{L}} \mathcal{C}_u(P_2) \parallel_{\mathcal{L}} \mathcal{C}_v(Q) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P_1') \parallel_{\mathcal{L}} \mathcal{C}_u(P_2) \parallel_{\mathcal{L}} \mathcal{C}_{v'}(Q') \parallel_{\mathcal{L}} St'$. By definition of the compiler $\mathcal{C}_u(P_2)$ can only proceed choosing the branch $rd(u).0$, then applying rule $7_{\mathcal{L}}$ we conclude that $\mathcal{C}(P) \parallel_{\mathcal{L}} \mathcal{C}(Q) \rightarrow^*_{\mathcal{L}} \mathcal{C}(P') \parallel_{\mathcal{L}} \mathcal{C}(Q') \parallel_{\mathcal{L}} St'$

(c) Let $P = P_1 \parallel_{\mathcal{R}} P_2$ and $Q = Q_1 +_{\mathcal{R}} Q_2$.

By hypothesis $\langle P_1 \parallel_{\mathcal{R}} P_2, RLS\rangle \xrightarrow{\{wr(t)\}} \langle P', RLS\rangle$

and $\langle Q_1 +_{\mathcal{R}} Q_2, RLS\rangle \xrightarrow{\{tk(t)\}} \langle Q', RLS\rangle$.

If rules applied are $3_{\mathcal{R}}$ and $2_{\mathcal{R}}$, and the processes which proceed are P_1 and Q_2 then

$\langle P_1, RLS\rangle \xrightarrow{\{wr(t)\}} \langle P_1', RLS\rangle$, $\langle Q_1, RLS\rangle \xrightarrow{\{tk(t)\}} \langle Q_1', RLS\rangle$, and $P' = P_1' \parallel_{\mathcal{R}} P_2$, $Q' = Q_1'$. So by inductive hypothesis $\mathcal{C}_{u_1}(P_1) \parallel_{\mathcal{L}} \mathcal{C}_{v_1}(Q_1) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u_1'}(P_1') \parallel_{\mathcal{L}} \mathcal{C}_{v_1'}(Q_1') \parallel_{\mathcal{L}} \langle u_1\rangle \parallel_{\mathcal{L}} \langle v_1\rangle \parallel_{\mathcal{L}} St'$.

Now by $8_{\mathcal{L}}$ and the compiler definition

$\mathcal{C}_u(P) \parallel_{\mathcal{L}} \mathcal{C}_{v_1}(Q_1) \parallel_{\mathcal{L}} \mathcal{C}_{v_1}(Q_2) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P') \parallel_{\mathcal{L}} \mathcal{C}_{v_1'}(Q_1') \parallel_{\mathcal{L}} \langle u_1\rangle \parallel_{\mathcal{L}} \langle v_1\rangle \parallel_{\mathcal{L}} St \parallel_{\mathcal{L}} \mathcal{C}_{v_1}(Q_2)$.

But as $\mathcal{C}_{v_1}(Q_2)$ can only proceed by the branch $rd(v_1).0$, considering the store and applying $7_{\mathcal{L}}$ results $\mathcal{C}_{v_1}(Q_2) \longrightarrow 0$. Therefore

$\mathcal{C}_u(P) \parallel_{\mathcal{L}} \mathcal{C}_{v_1}(Q) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P') \parallel_{\mathcal{L}} \mathcal{C}_{v_1'}(Q') \parallel_{\mathcal{L}} \langle u_1\rangle \parallel_{\mathcal{L}} \langle v_1\rangle \parallel_{\mathcal{L}} St \parallel$.

For the rest of the cases the proof is analogous to the previous ones.

Lemma 3.9 Let P be a process in $\mathcal{R}$, if $\langle P, RLS\rangle \xrightarrow{\{wr(t),tk(t)\}}_{\mathcal{R}} \langle P', RLS\rangle$ then $\forall u \in Id, \exists u' \in Id^*$ such that

$$\mathcal{C}_u(P) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P') \parallel_{\mathcal{L}} St'$$

Proof. The proof is by induction over the complexity of processes in $\mathcal{R}$.

(i) Base case: P is a single process, so the hypothesis does not verify.

(ii) Inductive hypothesis: the lemma is true for every process P_i in $\mathcal{R}$ present in the expression of P.

(a) Let $P = P_1 \parallel_{\mathcal{R}} P_2$.

If rule applied is $4_{\mathcal{R}}$, then

$\langle P_1, RLS\rangle \xrightarrow{\{wr(t)\}}_{\mathcal{R}} \langle P_1', RLS\rangle$,

$\langle P_2, RLS\rangle \xrightarrow{\{tk(t)\}}_{\mathcal{R}} \langle P_2', RLS\rangle$ and $P' = P_1' \parallel_{\mathcal{R}} P_2'$.

By applying lemma 3.8 we obtain

$\mathcal{C}_{u_1}(P_1) \parallel_{\mathcal{L}} \mathcal{C}_{u_2}(P_2) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u_1'}(P_1') \parallel_{\mathcal{L}} \mathcal{C}_{u_2'}(P_2') \parallel_{\mathcal{L}} \langle u_1\rangle \parallel_{\mathcal{L}} \langle u_2\rangle \parallel_{\mathcal{L}} St$.

Therefore by definition of compiler $\mathcal{C}_u(P) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P') \parallel_{\mathcal{L}} St'$, $(St' = \langle u_1\rangle \parallel_{\mathcal{L}} \langle u_2\rangle \parallel_{\mathcal{L}} St)$.

If the rule applied is $3_{\mathcal{R}}$ and P_1 is the process which proceed then $\langle P_1, RLS\rangle \xrightarrow{\{wr(t),tk(t)\}}_{\mathcal{R}} \langle P_1', RLS\rangle$ and $P' = P_1' \parallel_{\mathcal{R}} P_2$. By inductive hypothesis $\mathcal{C}_{u_1}(P_1) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u_1'}(P_1') \parallel_{\mathcal{L}} St'$. Applying rule $8_{\mathcal{L}}$ and compiler definition results $\mathcal{C}_u(P) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P') \parallel_{\mathcal{L}} St'$.

(b) Let $P = P_1 +_{\mathcal{R}} P_2$, so $\langle P_1 +_{\mathcal{R}} P_2, RLS\rangle \xrightarrow{\{wr(t),tk(t)\}}_{\mathcal{R}} \langle P', RLS\rangle$. Applying rule $2_{\mathcal{R}}$ and reasoning as in the previous cases, we obtain that $\mathcal{C}_u(P) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P') \parallel_{\mathcal{L}} St'$.

Proposition 3.10 Given two processes P, P' in $\mathcal{R}$ and the connector RLS, if $\langle P, RLS\rangle \longrightarrow_{\mathcal{R}} \langle P', RLS\rangle$ then $\mathcal{C}(P) \rightarrow^*_{\mathcal{L}} \mathcal{C}(P') \parallel St'$.

Proof. We will analyze rule $5_{\mathcal{R}}$.

(i) Let $P = P_1 \parallel_{\mathcal{R}} P_2$, so we have $\langle P_1 \parallel_{\mathcal{R}} P_2, RLS\rangle \rightarrow_{\mathcal{R}} \langle P', RLS\rangle$. By applying rule $5_{\mathcal{R}}$ we have that $\langle P_1 \parallel_{\mathcal{R}} P_2, RLS\rangle \xrightarrow{\overline{act}}_{\mathcal{R}} \langle P', RLS\rangle$ and $\langle RLS, \overline{act}\rangle \overset{act}{\rightarrowtail}_{\mathrm{C}}$

$\langle RLS, \emptyset\rangle$. By definition of RLS transitions, $\overline{act}$ must be $\{wr(t), tk(t)\}$

(a) if we consider rule $4_{\mathcal{R}}$, $P' = P_1' \parallel_{\mathcal{R}} P_2'$ and
$\langle P_1, RLS\rangle \overset{\{wr(t)\}}{\longrightarrow}_{\mathcal{R}} \langle P_1', RLS\rangle$,
$\langle P_2, RLS\rangle \overset{\{tk(t)\}}{\longrightarrow}_{\mathcal{R}} \langle P_2', RLS\rangle$.
Applying lemma 3.8 we obtain
$\mathcal{C}_{u_1}(P_1) \parallel_{\mathcal{L}} \mathcal{C}_{u_2}(P_2) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u_1'}(P_1') \parallel_{\mathcal{L}} \mathcal{C}_{u_2'}(P_2') \parallel St'$. Now by definition of compiler $\mathcal{C}$, results that
$\mathcal{C}_u(P_1 \parallel_{\mathcal{R}} P_2) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P_1' \parallel_{\mathcal{R}} P_2') \parallel St'$, that is $\mathcal{C}_u(P) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P') \parallel St'$.

(b) if we consider rule $3_{\mathcal{R}}$, assuming P_1 proceeds,
$P' = P_1' \parallel_{\mathcal{R}} P_2$ and $\langle P_1, RLS\rangle \overset{\{wr(t),tk(t)\}}{\longrightarrow}_{\mathcal{R}} \langle P_1', RLS\rangle$. By lemma 3.9, $\mathcal{C}_u(P_1) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P_1') \parallel_{\mathcal{L}} St'$. Now applying rule $8_{\mathcal{L}}$ and the compiler $\mathcal{C}_u(P_1 \parallel_{\mathcal{R}} P_2) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P_1' \parallel_{\mathcal{R}} P_2) \parallel St'$. Therefore $\mathcal{C}_u(P) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P') \parallel St'$

(ii) Let $P = P_1 +_{\mathcal{R}} P_2$. Reasoning as in previous cases, and applying rules $5_{\mathcal{R}}$, $2_{\mathcal{R}}$ and lemma 3.9 we obtain $\mathcal{C}_u(P_1) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P_1') \parallel St'$. By rule $8_{\mathcal{L}}$, $\mathcal{C}_u(P_1) \parallel_{\mathcal{L}} \mathcal{C}_u(P_2) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P_1') \parallel_{\mathcal{L}} \mathcal{C}_u(P_2) \parallel_{\mathcal{L}} St'$. Now as $\mathcal{C}_u(P_2)$ can only proceed by branch $rd(u)..0$ and $\langle u\rangle \subseteqq St'$, by rule $7_{\mathcal{L}}$, and the compiler definition we obtain $\mathcal{C}_u(P) \rightarrow^*_{\mathcal{L}} \mathcal{C}_{u'}(P') \parallel St'$.

Theorem 3.12 The compiler $\mathcal{C}$ is a modular embedding.

Proof. The properties (1) y (3) of 3.1 are satisfied directly, considering the notion of observables and that $\mathcal{D}$ is the identity . (2) is evident noting that $\mathcal{C}(A \bigotimes B) = \mathcal{C}(A) \parallel \mathcal{C}(B)$ where $\bigotimes \in \{\parallel, +\}$.

In order to prove (4), we need to verify that $\mathcal{O}'(\mathcal{C}(A))) = \mathcal{O}(A)$. We will proceed proving the double inclusion considering the possible elements in the sets of observables: *ss* and *ff*.

a. $\mathcal{O}(A) \subseteq \mathcal{O}'(\mathcal{C}(A))$

i. If $ss \in O(A)$, then $\langle A, RLS\rangle \rightarrow^*_{\mathcal{R}} \langle 0_{\mathcal{R}}, RLS\rangle$. In this case, we have a sequence similar to $\langle A, \mathrm{RLS}\rangle \rightarrow_{\mathcal{R}} \langle A_1, \mathrm{RLS}\rangle \rightarrow_{\mathcal{R}} \langle A_2, \mathrm{RLS}\rangle \rightarrow_{\mathcal{R}} ..... \rightarrow_{\mathcal{R}} \langle 0_{\mathcal{R}}, \mathrm{RLS}\rangle$. By applying previous properties we obtain the following sequence of τ-transitions in $\mathcal{L}$: $\mathcal{C}(A) \rightarrow \mathcal{C}(A_1) \parallel St_1 \rightarrow \mathcal{C}(A_2) \parallel St_2 \rightarrow ....\mathcal{C}(A_{n-1}) \parallel St_{n-1} \rightarrow \mathcal{C}(0) \parallel St_n$, where $St_{i-1} \subseteqq St_i$. Then $\mathcal{C}(A) \rightarrow^*_{\mathcal{L}} 0_{\mathcal{L}} \parallel_{\mathcal{L}} St_n$. Therefore $ss \in \mathcal{O}'(\mathcal{C}(A))$.

ii. If $ff \in \mathcal{O}(A)$ then $\exists Q \neq 0_{\mathcal{R}}$ and a state of the connector RLS' such that $\langle A, RLS\rangle \rightarrow^*_{\mathcal{R}} \langle Q, RLS\rangle \not\rightarrow_{\mathcal{R}}$. By applying previous properties we obtain $\mathcal{C}(A) \rightarrow^*_{\mathcal{L}} \mathcal{C}(Q) \parallel St_q$. Now reasoning by reduction to the absurd, suppose the existence of a process S, $S = S' \parallel St_s$ such that $\mathcal{C}(Q) \parallel_{\mathcal{L}} St_q \rightarrow_{\mathcal{L}} S' \parallel_{\mathcal{L}} St_s$. But by applying corollary 3.11 exists a state of the connector RLS_{St_s} and a process P in $\mathcal{R}$ that verifies $\mathcal{C}(P) = S'$ and $\langle Q, \mathrm{RLS}\rangle \rightarrow_{\mathcal{R}} \langle P, \mathrm{RLS}_{St_s}\rangle$. Contradiction. Therefore $ff \in \mathcal{O}'(\mathcal{C}(A))$.

By (i) and (ii), we obtain $\mathcal{O}(A) \subseteq \mathcal{O}'(\mathcal{C}(A)))$.

b. $\mathcal{O}'(\mathcal{C}(A)) \subseteq \mathcal{O}(A)$. The proof is analogous to the one given for case *a*.