Electronic Notes in Theoretical Computer Science 59 No. 3 (2002)
URL: http://www.elsevier.nl/locate/entcs/volume59.html 31 pages

Linear Embedding for a Quantitative Comparison of Language Expressiveness

Antonio Brogi, Alessandra Di Pierro

Department of Computer Science, University of Pisa, Italy

Herbert Wiklicky

Department of Computing, Imperial College London, UK

Abstract

We introduce the notion of *linear embedding* which refines Shapiro's notion of embedding by recasting it in a linear-space based semantics setting. We use this notion to compare the expressiveness of a class of languages that employ asynchronous communication primitives á la Linda. The adoption of a linear semantics in which the observables of a language are linear operators (matrices) representing the programs transition graphs allows us to give *quantitative* estimates of the different expressive power of languages, thus improving previous results in the field.

1 Introduction

The evolution of Computer Science has been accompanied by the creation of a huge number of different programming languages. Quoting [9]:

> All reasonable programming languages are equivalent, since they are Turing-complete. However, if the differences between them were not material, we would not have invented so many of them.

On the other hand, in the practice of programming, certain languages are considered to be more "powerful" than others for their capability to express control and data structures.

Comparing the expressiveness of languages gets more complicated when considering concurrent languages. Nondeterminism and non-successful computations play a crucial role there, and there is no correspondent for the notion of Turing-completeness to measure the absolute expressive power of a language.

The question of formally comparing the expressiveness of languages has been the object of a quite large body of research (see [2] for an introduction). A natural way to compare the expressive power of two languages is to verify

$$
\begin{tikzcd}
L_1 \arrow[r, "Obs_1"] \arrow[d, "\mathcal{C}"'] & \mathcal{O}_1 \\
L_2 \arrow[r, "Obs_2"'] & \mathcal{O}_2 \arrow[u, "\mathcal{D}"']
\end{tikzcd}
$$

Fig. 1. Embedding

whether all programs written in one language can be "easily" and "equivalently" translated into the other one. This idea is formalised by the notion of *embedding* introduced in [9]:

> Let L_1, L_2 be two languages. Assume given the observation criteria $Obs_1 : L_1 \mapsto \mathcal{O}_1$ and $Obs_2 : L_2 \mapsto \mathcal{O}_2$ where $\mathcal{O}_1$ and $\mathcal{O}_2$ are some suitable domains. Then L_2 *embeds* L_1 if there exists a mapping $\mathcal{C}$ (compiler) from L_1 to L_2 and a mapping $\mathcal{D}$ (decoder) from $\mathcal{O}_2$ to $\mathcal{O}_1$ such that the diagram in Figure 1 commutes, that is, for every program $P \in L_1$:
>
> $$Obs_1(P) = \mathcal{D}(Obs_2(\mathcal{C}(P))).$$

The notion of embedding, and the refined notion of *modular embedding* [2], have been employed to compare the relative expressive power of a number of different languages by establishing *qualitative* separation results ($L_1 \leq L_2$ and $L_2 \not\leq L_1$) as well as equivalence results ($L_1 \leq L_2$ and $L_2 \leq L_1$).

Our aim is to introduce *quantitative* aspects in the comparison of the expressiveness of languages. We will present a methodology to assign quantitative estimates to separation results, so as to estimate "how much" a language is more expressive than another one.

Our approach can be summarised as follows:

- We introduce a notion of *linear embedding* as a base to quantitatively compare the expressive power of languages. We start from the standard notion of embedding as introduced by Shapiro [9] and recast it in a linear setting by taking linear spaces as semantic domains. Additionally, we require the "compositionality" of $\mathcal{C}$ and $\mathcal{D}$ (as in [2]), that is we require $\mathcal{C}$ to be defined compositionally on the program structure, and $\mathcal{D}$ to be a linear map.
- The observation criteria are defined in terms of a linear semantics which associates to each program a linear operator in a suitably defined linear algebra.
- The notion of linear embedding induces a partial order over the languages and allows us to establish separation ($L_1 \prec L_2$) and equivalence ($L_1 = L_2$) results between languages. We then annotate each separation result with a quantity representing the difference in expressive power. This quantity depends on the dimensions of the algebras associated with the two languages.

Intuitively, the dimension of an algebra gives an estimate of the number of different possible behaviours expressible with that language.

- We apply our notion of linear embedding and quantitative comparisons to a family of Linda-like languages and we compare the results with the ones established in [1].

In the next section we introduce a family of simple Linda-like languages which we will use to exemplify the linear semantics, the notion of linear embedding and the quantitative comparison based on the latter. The general definition of linear embedding is then introduced in Section 3, while quantitative estimates are discussed in Section 4.

2 The Family of Linda-like Languages

2.1 The Syntax of $\mathcal{L}(X)$

Following [1], we consider a family of languages $\mathcal{L}(X)$ which differ from one another for the set X of communication primitives used. These primitives correspond to the basic Linda primitives for adding a token to a shared data-space, getting it from the data-space, and checking for its presence or absence in the data-space. The languages $\mathcal{L}(X)$ also include standard prefix and choice operators.

The syntax of $\mathcal{L}(X)$ is formally defined by the following grammar:

$$P ::= \mathbf{stop} \;\Big|\; C.P \;\Big|\; P + P$$
$$C ::= \mathbf{ask}(t) \;\Big|\; \mathbf{nask}(t) \;\Big|\; \mathbf{tell}(t) \;\Big|\; \mathbf{get}(t)$$

where t is a generic element called *token* in a denumerable set $\mathcal{T}$, P is a process and C a communication action (or prefix). The parameter X defining our Linda-like languages is a subset of the primitives defined by C.

A program in $\mathcal{L}(X)$ is therefore either an inactive, trivial program **stop**, or a sequential composition $C.P$ or a choice $P + P$. As usual we omit a trailing **stop** if it is prefixed by a non-empty sequence of basic actions C. Note also that for the current treatment we do not consider a parallel construct.

2.2 A Linear Semantics for $\mathcal{L}(X)$

A computation in $\mathcal{L}(X)$ consists of a set of processes which share a common store represented as a multi-set of tokens, and perform communication actions on it. We model such a computation as a transformation of a state (initial store) into another state (final store). We represent states as vectors in the free vector space $\mathcal{V}(\mathcal{S})$ on the set $\mathcal{S}$ of all possible stores (viz multi-sets of tokens). The free vector space $\mathcal{V}(\mathcal{S})$ on $\mathcal{S}$ is defined as the set of all formal linear combinations of elements of $\mathcal{S}$:

$$\mathcal{V}(\mathcal{S}) = \left\{ \sum_{s \in \mathcal{S}} x_s \vec{s} \;\middle|\; x_s \in \mathbb{R} \right\}.$$

The idea is to encode the effect of executing the program P in a state x which is represented by a vector $\vec{x}$ in terms of a linear operator on $\mathcal{V}(\mathcal{S})$. That is, if $\mathbf{M}(P)$ represents a program P and $\vec{x}$ the state where it is executed in, then the resulting state is represented by $\vec{x} \cdot \mathbf{M} = \vec{y}$, i.e. the vector obtained by multiplying $\vec{x}$ with the matrix $\mathbf{M}$.

If $\mathcal{S}$ is a finite set of cardinality n, then these linear operators are finite-dimensional or, more precisely they are $n \times n$ matrices. For denumerable sets of stores, we have instead to consider infinite-dimensional linear operators. Although we assume our languages $\mathcal{L}(X)$ to be defined in general on a countable set of stores, we will always refer to finite sets in our reasoning and in the examples shown throughout the paper.

In particular, we will denote by $\mathcal{L}_t^s(X)$ the finite approximations of the language $\mathcal{L}(X)$ defined on t tokens and on stores of size at most s. The number of all possible stores of size s which can be constructed with t tokens is given by the following basic combinatorial formula (e.g. [6, Sect. 1.4]):

$$n(s,t) = \sum_{i=0}^{s} \binom{i+t-1}{i} = \binom{s+t}{s}.$$

For every number i between 0 and the maximal store size s we count the number of possible multisets (i.e. combinations with repetitions) containing t different types of tokens, and sum them up.

Example 2.1 Consider a language $\mathcal{L}_3^2(X)$. The set $\mathcal{S}$ of stores for $\mathcal{L}_3^2(X)$ can be enumerated:

$$\{\!\!\{\}\!\!\}, \{\!\!\{t_1\}\!\!\}, \{\!\!\{t_2\}\!\!\}, \{\!\!\{t_3\}\!\!\}, \{\!\!\{t_1, t_1\}\!\!\}, \{\!\!\{t_1, t_2\}\!\!\}, \{\!\!\{t_1, t_3\}\!\!\}, \{\!\!\{t_2, t_2\}\!\!\}, \{\!\!\{t_2, t_3\}\!\!\}, \{\!\!\{t_3, t_3\}\!\!\}.$$

The free vector space $\mathcal{V}(\mathcal{S})$ is then 10-dimensional and therefore isomorphic to $\mathbb{R}^{10}$. In this vector space, the set of stores

$$\{\{\!\!\{\}\!\!\}, \{\!\!\{t_2\}\!\!\}, \{\!\!\{t_1, t_2\}\!\!\}, \{\!\!\{t_1, t_3\}\!\!\}\}$$

is represented by the vector

$$(1, 0, 1, 0, 0, 1, 1, 0, 0, 0).$$

In order to correctly model the behaviour of a program when considering finite approximations on n stores, we have to introduce an additional *overflow state* ω. This allows us to deal with the case in which the addition of a further token to the store make its size bigger than n. Such a state has the property that once it is reached no further transitions which leave that state are possible.

The linear operators associated to the basic communication actions are defined as follows (we denote by $\mathbf{M}_{ij}$ the element at row i and column j of the linear operator $\mathbf{M}$). We denote the operators representing a program P by $[\![P]\!]$ or $\mathbf{M}(P)$. For all stores $s_1 \neq \omega$ the entries in the operators representing the basic actions or commands are defined as:

$$(\llbracket \mathbf{tell}(t) \rrbracket)_{s_1,s_2} = \begin{cases} 1 \text{ if } s_2 = s_1 \cup \{\!|t|\!\} \text{ and } \|s_2\| \leq n \\ 1 \text{ if } s_2 = \omega \text{ and } \|s_1 \cup \{\!|t|\!\}\| > n \\ 0 \text{ otherwise} \end{cases}$$

$$(\llbracket \mathbf{ask}(t) \rrbracket)_{s_1,s_2} = \begin{cases} 1 \text{ if } t \in s_1 \text{ and } s_2 = s_1 \\ 0 \text{ otherwise} \end{cases}$$

$$(\llbracket \mathbf{get}(t) \rrbracket)_{s_1,s_2} = \begin{cases} 1 \text{ if } t \in s_1 \text{ and } s_2 = s_1 \setminus \{\!|t|\!\} \\ 0 \text{ otherwise} \end{cases}$$

$$(\llbracket \mathbf{nask}(t) \rrbracket)_{s_1,s_2} = \begin{cases} 1 \text{ if } t \notin s_1 \text{ and } s_2 = s_1 \\ 0 \text{ otherwise} \end{cases}$$

where $\cup$ and $\setminus$ denote multi-set union and difference, respectively. For $s_1 = \omega$, the entries for all of the four basic commands $\mathbf{tell}(t)$, $\mathbf{ask}(t)$, $\mathbf{get}(t)$, and $\mathbf{nask}(t)$ are defined in the same way:

$$(\llbracket C(t) \rrbracket)_{s_1,s_2} = \begin{cases} 1 \text{ if } s_2 = \omega \\ 0 \text{ otherwise.} \end{cases}$$

Prefix and nondeterministic choice are modelled by multiplication and sum over the algebra of linear operators:

$$\llbracket C.P \rrbracket = \llbracket C \rrbracket \cdot \llbracket P \rrbracket \text{ and } \llbracket P + Q \rrbracket = \llbracket P \rrbracket + \llbracket Q \rrbracket.$$

Finally, $\mathbf{stop}$ is represented by the identity operator on $\mathcal{C}(\mathcal{S})$, i.e.

$$(\llbracket \mathbf{stop} \rrbracket)_{s_1,s_2} = \begin{cases} 1 \text{ if } s_1 = s_2 \\ 0 \text{ otherwise.} \end{cases}$$

This *linear semantics* for Linda-like languages $\mathcal{L}(X)$ can be seen as a vector space encoding of the standard operational semantics for $\mathcal{L}(X)$ as presented for example in [1].

The linear operator $\mathbf{M} = \mathbf{M}(P)$ in effect encodes the transition relation $\langle P|s_0\rangle \longrightarrow \langle Q_i|s_i\rangle$ on *configurations*, i.e. pairs $\langle P|s\rangle$ consisting of a program P and stores s: If $\langle P|s_0\rangle$ can make transitions to a number of $\langle Q_i|s_i\rangle$, for $i = 1, \ldots, k$, then one can show that $\mathbf{M} \cdot \vec{s}_0 = \sum_{i=0}^{n} \vec{s}_i$.

In other words, each row in $\mathbf{M}(P)$ encodes the observables of P when executed in the store corresponding to this row. If a program P is deadlocked in a certain store, then the corresponding row is "empty", i.e. contains only zeros.

In this sense we can distinguish between *successful termination* of $\langle P|s_0\rangle$ — if row s_0 in $\mathbf{M}(P)$ contains a non-zero entry — and *failure* or *deadlock* — if row s_0 in $\mathbf{M}(P)$ contains only zeros. In the following we will call two programs P and Q *equivalent* if they have the same semantics, that is if $\llbracket P \rrbracket = \llbracket Q \rrbracket$.

This means, of course, that for all initial stores the observables (for successful computations) are the same.

Example 2.2 Consider the language $\mathcal{L}_2^2(X)$, where X is the set of all communication actions and the number of possible stores is $n = 6$. Therefore, we can represent programs in $\mathcal{L}_2^2(X)$ by means of 7×7 matrices — allowing also for the overflow store ω.

As examples, we present in the following the linear operators for $\mathbf{tell}(t_2)$, $\mathbf{get}(t_1)$, $\mathbf{ask}(t_1)$ and $\mathbf{ask}(t_1).\mathbf{tell}(t_2)$. We assume the following enumeration of stores:

$$\{\!|\,|\!\}, \{\!| t_1 |\!\}, \{\!| t_2 |\!\}, \{\!| t_1, t_1 |\!\}, \{\!| t_1, t_2 |\!\}, \{\!| t_2, t_2 |\!\}, \omega$$

Then the operators representing some of the basic actions are given as:

$$\begin{array}{ccc} [\![\mathbf{tell}(t_2)]\!] & [\![\mathbf{get}(t_1)]\!] & [\![\mathbf{ask}(t_1)]\!] \\ = & = & = \\ \begin{pmatrix} 0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1 \end{pmatrix} & \begin{pmatrix} 0&0&0&0&0&0&0 \\ 1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1 \end{pmatrix} & \begin{pmatrix} 0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1 \end{pmatrix} \end{array}$$

The semantics of, for example, $\mathbf{ask}(t_1).\mathbf{tell}(t_2)$ is the product:

$$\begin{array}{ccccc} [\![\mathbf{ask}(t_1)]\!] & & [\![\mathbf{tell}(t_2)]\!] & & [\![\mathbf{ask}(t_1).\mathbf{tell}(t_2)]\!] \\ = & & = & & = \\ \begin{pmatrix} 0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1 \end{pmatrix} & \cdot & \begin{pmatrix} 0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1 \end{pmatrix} & = & \begin{pmatrix} 0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1 \end{pmatrix} \end{array}$$

3 Linear Embedding

Given two languages L_1 and L_2, we consider their *linear semantics*, namely the mappings $S_1 : L_1 \mapsto \mathcal{A}_1$ and $S_2 : L_2 \mapsto \mathcal{A}_2$, where $\mathcal{A}_1$ and $\mathcal{A}_2$ are two

$$\begin{tikzcd} L_1 \arrow[r, "S_1"] \arrow[d, "\mathcal{C}"'] & \mathcal{A}_1 \\ L_2 \arrow[r, "S_2"'] & \mathcal{A}_2 \arrow[u, "\mathcal{D}"'] \end{tikzcd}$$

Fig. 2. Linear Embedding

linear algebras. To simplify the presentation, we assume that both languages are defined over the same set of *stores* $\mathcal{S}$. The $\mathcal{A}_1$ and $\mathcal{A}_2$ are linear algebras over $\mathcal{V}(\mathcal{S})$, i.e. algebras of linear operators on the free vector space over the states $\mathcal{S}$.

We then define the notion of *linear embedding* in terms of the commutativity of the diagram depicted in Figure 2. This is essentially the notion of embedding introduced by Shapiro [9] translated into a linear (i.e. vector space based) setting. Similarly to [2,1], we put additional requirements on the compiler $\mathcal{C}$ and the decoder $\mathcal{D}$, requesting a *modular* translation between L_1 and L_2. We recall first the three conditions for modular embedding (cf. Figure 1), e.g. [2] or [1]:

Definition 3.1 We say that language L_2 *modularly embeds* L_1 ($L_1 \leq L_2$) if there exists a map $\mathcal{C} : L_1 \mapsto L_2$, a *compiler*, and a map $\mathcal{D} : \mathcal{O}_2 \mapsto \mathcal{O}_1$, a *decoder*, such that for all statements $A \in L_1$ we have:

$$Obs_1(A) = \mathcal{D}(Obs_2(\mathcal{C}(A)))$$

where:

(i) $\mathcal{C}$ is *compositional*, i.e. for all P_1, P_2 in L_1:

$$\mathcal{C}(P_1.P_2) = \mathcal{C}(P_1) \cdot \mathcal{C}(P_2) \quad \text{and} \quad \mathcal{C}(P_1 + P_2) = \mathcal{C}(P_1) + \mathcal{C}(P_2);$$

(ii) $\mathcal{D}$ is defined *element-wise* on $\mathcal{O}_2$, i.e. there exists a $\mathcal{D}_{el}$ such that:

$$\forall X \in \mathcal{O}_2 : \quad \mathcal{D}(X) = \{\mathcal{D}_{el}(x) \mid x \in X\}\,;$$

(iii) $\mathcal{D}$ preserves the termination mode.

As in [1], the termination mode expresses whether a program computation starting in a given initial store s reaches the final configuration $(\textbf{stop}, u)$ for some store u (successful termination) or not (deadlock). In the linear semantics the first case is expressed by a 1 in the entry $\mathbf{M}_{su}$ of the matrix associated to the program, while the second case corresponds to the row indexed by s in the matrix being the zero vector.

Definition 3.2 We say that language L_2 *linearly embeds* L_1 ($L_1 \preceq L_2$) if there exists a map $\mathcal{C} : L_1 \mapsto L_2$, a *compiler*, and a map $\mathcal{D} : \mathcal{A}_2 \mapsto \mathcal{A}_1$, a *decoder*, such that for all statements $A \in L_1$ we have:

$$S_1(A) = \mathcal{D}(S_2(\mathcal{C}(A)))$$

$$
\begin{tikzcd}
L_1 \arrow[r, "\mathrm{Obs}_1"'] \arrow[rr, bend left, "S_1"] \arrow[d, "\mathcal{C}"'] & \mathcal{O}_1 \arrow[r, "\mathrm{lift}"'] & \mathcal{A}_1 \\
L_2 \arrow[r, "\mathrm{Obs}_2"] \arrow[rr, bend right, "S_2"'] & \mathcal{O}_2 \arrow[u, "\mathcal{D}"'] \arrow[r, "\mathrm{lift}"] & \mathcal{A}_2 \arrow[u, "\mathcal{D}_\ell"']
\end{tikzcd}
$$

Fig. 3. Lifting

where:

(i) $\mathcal{C}$ is defined *compositionally*, and

(ii) $\mathcal{D}$ is a *linear* map.

The last two conditions in Definition 3.1 are reflected in the linearity of the decoder in the notion of linear embedding, as will be made clear in the proof of Proposition 3.3.

The previously introduced notion of a modular embedding defines a partial order on languages and can be used to establish separation results ($L' \leq L$ and $L \not\leq L'$, denoted by $L' < L$) and equivalence results ($L' \leq L$ and $L \leq L'$, denoted by $L' = L$). A similar order $\preceq$ on languages is introduced by linear embedding. The following proposition shows that this ordering is qualitatively the same as the one induced by the modular embedding.

Proposition 3.3 *Let L_1 and L_2 be two languages. Then L_2 modularly embeds L_1 iff L_2 linearly embeds L_1.*

Proof.

($\Rightarrow$) Suppose that $(\mathcal{C}, \mathcal{D})$ is a modular embedding of L_1 in L_2, that is the diagram in Figure 1 commutes for the languages L_1 and L_2 and that the conditions in Definition 3.1 hold. We show how to lift the semantics Obs_1 and Obs_2 to linear semantics S_1 and S_2 (see Figure 3).

To this purpose, consider the following "input dependent" formulation of the observation criteria Obs_1 and Obs_2:

$$Obs_1(P_1) : X_1 \mapsto \mathcal{P}(X_1) \text{ and } Obs_2(P_2) : X_2 \mapsto \mathcal{P}(X_2)$$

where X_1 (or X_2) represents the input domain for the programs in L_1 (or L_2), and $\mathcal{P}(X_1)$ (or $\mathcal{P}(X_2)$) is a suitable domain contained in the powerset of X_1 (or X_2), whose elements represent the possible outcomes of a computation of a program in L_1 (or L_2). Thus, the map Obs_1 (or Obs_2) can be seen as a map associating to each program in L_1 (or L_2) and each input the corresponding results.

We can then define the linear semantics S_1 and S_2 by *lifting* the observation criteria Obs_1 and Obs_2 as follows. For each program $P \in \mathcal{L}_1$ we define

$$\begin{tikzcd}
L_1 \arrow[r, "S_1"'] \arrow[rr, bend left, "\mathrm{Obs}_1"] \arrow[d, "\mathcal{C}"'] & \mathcal{A}_1 \arrow[r, "\mathrm{supp}"'] & \mathcal{O}_1 \\
L_2 \arrow[r, "S_2"] \arrow[rr, bend right, "\mathrm{Obs}_2"'] & \mathcal{A}_2 \arrow[r, "\mathrm{supp}"] \arrow[u, "\mathcal{D}"'] & \mathcal{O}_2 \arrow[u, "\mathcal{D}_m"']
\end{tikzcd}$$

Fig. 4. Support

the operator

$$(S_1(P))_{xy} = \begin{cases} 1 \text{ if } y \in Obs_1(P)(x) \\ 0 \text{ otherwise.} \end{cases}$$

The semantics S_2 for the language L_2 is defined similarly. The domain $\mathcal{A}_1$ ($\mathcal{A}_2$) is the minimal algebra containing the operators $S_1(P)$ ($S_2(P)$) for all $P \in L_1$ ($P \in L_2$).

Starting from the modular embedding $(\mathcal{C}, \mathcal{D})$ we can define a linear embedding $(\mathcal{C}_\ell, \mathcal{D}_\ell)$ with $\mathcal{C}_\ell = \mathcal{C}$, and $\mathcal{D}_\ell$ defined as the linear extension on $\mathcal{A}_2$ of the map $\mathcal{D}'_\ell$ defined on all elements $S_2(\mathcal{C}(P_1))$ by:

$$(\mathcal{D}'_\ell(S_2(\mathcal{C}(P_1)))_{xy} = \begin{cases} 1 \text{ if } y \in \mathcal{D}_{el}(z) \text{ and } z \in Obs_2(\mathcal{C}(P_1))(\mathcal{C}(x)) \\ 0 \text{ otherwise} \end{cases}$$

where $\mathcal{D}_{el}$ is the element-wise definition of $\mathcal{D}$, $x, y \in X_1$, $z \in X_2$, and $\mathcal{C}(x)$ denotes the compiled input for $\mathcal{C}(P_1)$. If we assume that the stores space for both L_1 and L_2 we can take $\mathcal{C}(x) = x$.

As $\mathcal{D}'_\ell$ is defined via $\mathcal{D}_{el}$ it is linear. We know that $S_2(\mathcal{C}(P_1)) + S_2(\mathcal{C}(P_2)) = S_2(\mathcal{C}(P_1) + \mathcal{C}(P_1))$ from the definition of the linear semantics. Therefore:

$$(\mathcal{D}'_\ell(S_2(\mathcal{C}(P_1)) + S_2(\mathcal{C}(P_2))))_{xy} = (\mathcal{D}'_\ell(S_2(\mathcal{C}(P_1) + \mathcal{C}(P_1)))_{xy} = 1$$

iff $y \in \mathcal{D}_{el}(z)$ for z in $Obs_2(\mathcal{C}(P_1) + \mathcal{C}(P_2))(\mathcal{C}(x))$. But as $Obs_2(\mathcal{C}(P_1) + \mathcal{C}(P_2))(\mathcal{C}(x)) = Obs_2(\mathcal{C}(P_1)) + Obs_2(\mathcal{C}(P_2)$ z must be in $Obs_2(\mathcal{C}(P_1))$ or in $Obs_2(\mathcal{C}(P_2)$. This means that $(\mathcal{D}'_\ell(S_2(\mathcal{C}(P_1)))_{xy} = 1$ or $(\mathcal{D}'_\ell(S_2(\mathcal{C}(P_2)))_{xy} = 1$. In other words for all x and y we have:

$$(\mathcal{D}'_\ell(S_2(\mathcal{C}(P_1)) + S_2(\mathcal{C}(P_2))))_{xy} = (\mathcal{D}'_\ell(S_2(\mathcal{C}(P_1)))_{xy} + \mathcal{D}'_\ell(S_2(\mathcal{C}(P_2)))_{xy}$$

($\Leftarrow$) Suppose that $(\mathcal{C}, \mathcal{D})$ is a linear embedding of L_1 in L_2 according to Definition 3.2. Starting from the linear semantics S_1 and S_2 we construct two observation criteria Obs_1 and Obs_2 for L_1 and L_2 respectively, as the *support* of S_1 and S_2 (see Figure 4).

For the language L_1, this is defined as follows: For all $P_1 \in L_1$ and $x \in X_1$,

$$Obs_1(P_1)(x) = \{y \in X_1 \mid y \in \mathrm{supp}(S_1(P_1)(x))\},$$

where for a vector $\vec{z} = (z_1, \ldots, z_n)$, $\mathrm{supp}(\vec{z}) = \{z_i \mid z_i \neq 0\}$. We then define the domain $\mathcal{O}_1$ as

$$\mathcal{O}_1 = \mathrm{supp}(\mathcal{A}_1) = \bigcup_{T \in \mathcal{A}_1} \mathrm{supp}(T),$$

where $\mathrm{supp}(T) = \{O \mid O = \mathrm{supp}(T(x)), x \in X_1\}$. Clearly, $\mathcal{O}_1$ is contained in the power-set of X_1 and contains $Obs_1(P_1)(x)$ for all $P_1 \in L_1$ and $x \in X_1$. Analogously we define Obs_2 and $\mathcal{O}_2$.

We now construct the embedding $(\mathcal{C}_m, \mathcal{D}_m)$, by taking $\mathcal{C}_m = \mathcal{C}$ and $\mathcal{D}_m : \mathcal{O}_2 \mapsto \mathcal{O}_1$ the map defined by

$$\mathcal{D}_m(O) = \mathcal{D}(T),$$

for all $O \in \mathcal{O}_2$, with $O = \mathrm{supp}(T)$ for some $T \in \mathcal{A}_2$.

The first condition for a modular embedding is satisfied trivially: $\mathcal{C}$ is compositional. Furthermore, it is easy to see that $\mathcal{D}_m$ is defined element-wise by

$$\mathcal{D}_m(O) = \{\mathcal{D}_{el}(x) \mid x \in O\},$$

where $\mathcal{D}_{el}(x) = \mathrm{supp}(\mathcal{D}(T)(x)$. The linearity of $\mathcal{D}$ also ensures that the third condition of Definition 3.1 holds: empty rows are mapped to empty rows, it preserves deadlock, and by definition it also preserves successful termination.

It remains to show that $(\mathcal{C}_m, \mathcal{D}_m)$ is an embedding, that is the corresponding diagram commutes. Since by definition $Obs(\mathcal{C}(P_1)) = \mathrm{supp}(S_2(\mathcal{C}(P_1)))$, we have that $\mathcal{D}_m(Obs_2(\mathcal{C}(P_1))) = \mathrm{supp}(\mathcal{D}(S_2(\mathcal{C}(P_1))))$. Moreover, by definition $Obs_1(P_1) = \mathrm{supp}(S_1(P_1))$ holds too. Thus, we have

$$\mathcal{D}_m(Obs_2(\mathcal{C}(P_1))) = Obs_1(P_1),$$

for all $P_1 \in L_1$.

□

4 Measuring the Expressive Power

Our ultimate aim is to quantify the difference between the expressive power of Linda-like languages. To this end, we annotate the separation results devised by modular/linear embedding with a quantity which measures "how much" a language is more or less expressive than another. This quantity is defined in terms of the "dimension" of the operator algebras defining the linear semantics of the languages we are comparing.

4.1 Algebras and Dimensions

A standard way of constructing an algebra is to generate all possible linear combinations starting from a set of basic operators (generators). We adopt this method to associate to each language its algebra of operators.

Definition 4.1 [5] Given a set $\mathcal{M} = \{\mathbf{M}_i\}_{i \in I} \subseteq Lin(\mathcal{V})$ of linear operators on a vector space $\mathcal{V}$. A *word* over $\mathcal{M}$ is a linear operator on $\mathcal{V}$ of the form:

$$\mathbf{W} = \prod_{j \in J} \mathbf{M}_j$$

with J a multi-set of indices in I. We denote the set of all words $\mathbf{W}$ over $\mathcal{M}$ by $\mathcal{W} = \mathcal{W}(\mathcal{M})$.

The *algebra* $\mathcal{A}(\mathcal{M})$ *generated by* $\mathcal{M}$ is given by the set of all linear combinations of words $\mathbf{W}_k$ over $\mathcal{M}$, i.e. operators of the form (with $x_k \in \mathbb{R}$):

$$\mathbf{A} = \sum_k x_k \cdot \mathbf{W}_k.$$

The algebra $\mathcal{A}(\mathcal{M})$ generated by a set of linear operators $\mathcal{M}$ is thus the *linear span* $\langle \mathcal{W}(\mathcal{M}) \rangle$ of words $\mathbf{W}_k$ over $\mathcal{M}$.

Definition 4.2 We define the algebra $\mathcal{A}(X)$ associated to a language $\mathcal{L}(X)$ as the algebra generated by the basic actions $\mathcal{B} = \{[\![C]\!] \mid C \in X\}$.

It is easy to show that for the class of Linda-like languages we consider, the algebra generated by the basic actions is the smallest algebra which contains the semantics of all programs in the language.

Proposition 4.3 *Let $\mathcal{L}(X)$ be a Linda-like language with basic actions $\mathcal{B}$. Then*

$$\langle S(\mathcal{L}(X)) \rangle = \mathcal{A}(X)$$

where $S(\mathcal{L}(X)) \subseteq \mathcal{L}in(\mathcal{V})$ denotes the semantical image of $\mathcal{L}(X)$, i.e.

$$S(\mathcal{L}(X)) = \{[\![P]\!] \mid P \in \mathcal{L}(X)\}$$

Proof.

$\langle S(\mathcal{L}(X)) \rangle \subseteq \mathcal{A}(X)$**:**

By structural induction:

- The **stop** agent is represented by the identity operator $\mathbf{I}$. It is in $\mathcal{B}$ by definition, and therefore in $\mathcal{A}(X)$.
- Each of the basic operations **ask**(t), **nask**(t), **tell**(t), and **get**(t) is represented by an operator in $\mathcal{A}(X)$ (cf. Section 2), and therefore belongs to $\mathcal{A}(X)$.
- Suppose we have program of the form $C.P$ where $[\![P]\!] \in \mathcal{A}(X)$ and $[\![C]\!] \in \mathcal{B}$. Then $[\![P]\!]$ is some linear combination of words in $\mathbf{W}_k \in \mathcal{W}(\mathcal{B})$:

  $$[\![P]\!] = \sum_k x_k \cdot \mathbf{W}_k.$$

  The semantics of $C.P$ is therefore:

  $$[\![C.P]\!] = [\![C]\!] \cdot [\![P]\!] = [\![C]\!] \sum_k x_k \cdot \mathbf{W}_k = \sum_k x_k \cdot [\![C]\!] \cdot \mathbf{W}_k$$

  As the product of an operator representing a basic action $[\![C]\!]$ and any word W_k gives another word we can conclude that $[\![C.P]\!]$ is also a linear combination of words, i.e.

  $$[\![C.P]\!] \in \langle \mathcal{W}(\mathcal{B}) \rangle = \mathcal{A}(X).$$

- Suppose we have program of the form $P + Q$ where $[\![P]\!] \in \mathcal{A}(X)$ and $[\![Q]\!] \in \mathcal{A}(X)$. Then $[\![P]\!]$ and $[\![Q]\!]$ are some linear combinations of words $\mathbf{W}_k \in \mathcal{W}(\mathcal{B})$ and $\mathbf{V}_l \in \mathcal{W}(\mathcal{B})$:
$$[\![P]\!] = \sum_k x_k \cdot \mathbf{W}_k \text{ and } [\![Q]\!] = \sum_l x_l \cdot \mathbf{V}_l$$
The semantics of $P + Q$ is therefore:
$$[\![P + Q]\!] = [\![P]\!] + [\![Q]\!] = \sum_k x_k \cdot \mathbf{W}_k + \sum_l x_l \cdot \mathbf{V}_l$$
i.e. a linear combination of words over $\mathcal{B}$, and so we have:
$$[\![P + Q]\!] \in \langle \mathcal{W}(\mathcal{B}) \rangle = \mathcal{A}(X).$$

$\mathcal{A}(X) \subseteq \langle S(\mathcal{L}(X)) \rangle$**:**

- We show that for all $\mathbf{W} \in \mathcal{W}(\mathcal{B})$ there exists $P \in \mathcal{L}(X)$ such that $[\![P]\!] = \mathbf{W}$.
  Given a word $\mathbf{W} = \mathbf{M}_1 \cdot \mathbf{M}_2 \ldots \mathbf{M}_n$ of length n, then each of the $\mathbf{M}_i \in \mathcal{B}$ corresponds to a basic action B_i. Therefore, there exists at least one program [1] P such that $[\![P]\!] = \mathbf{W}$. In fact,
$$[\![P]\!] = [\![B_1.B_2.\ldots.B_n]\!] = \mathbf{M}_1 \cdot \mathbf{M}_2 \ldots \mathbf{M}_n = \mathbf{W}$$
- The set of words formed by basic actions is thus a subset of the semantics of programs in $\mathcal{L}(X)$, i.e.
$$\mathcal{W}(\mathcal{B}) \subseteq S(\mathcal{L}(X)).$$
The same relation holds thus also for the linear span $\langle \mathcal{W} \rangle$ — the algebra $\mathcal{A}(X)$ — and the linear span $\langle S(\mathcal{L}(X)) \rangle$, i.e.
$$\mathcal{A}(X) = \langle \mathcal{W} \rangle \subseteq \langle S(\mathcal{L}(X)) \rangle .$$
□

We now recall the definition of the dimension of a linear space, i.e. a vector space or a linear algebra like $\mathcal{A}(X)$.

Definition 4.4 [7] Let $\mathcal{Z}$ be a linear space over a field $\mathbb{K}$. A sequence of elements $a_1, \ldots a_n$ in $\mathcal{Z}$ is called *linearly dependent* if there exists a sequence $x_1, \ldots, x_n$ in $\mathbb{K}$ such that not all the x_i are equal to 0 and $x_1 a_1 + \ldots + x_n a_n = 0$. A sequence in $\mathcal{Z}$ is called *linearly independent* if it is not linearly dependent.

The maximal number of linearly independent objects in a vector space or an algebra defines its *dimension*.

Definition 4.5 [7] Let S be a linear space over a field $\mathbb{K}$. The dimension of $\mathcal{Z}$, $\dim(\mathcal{Z})$, is the number n such that there exists a linearly independent sequence of n elements in $\mathcal{Z}$, and no sequence of $n+1$ elements in $\mathcal{Z}$ is linearly independent.

[1] Note that the same $\mathbf{W}$ might be the product of different combinations of basic actions.

For a language defined on a finite number of stores we will denote by $\dim(\mathcal{L})$ the dimension of the algebra associated to the language $\mathcal{L}$. In practical terms, we only need to determine the number of linearly independent words generated by the basic actions in order to calculate the dimension of the algebra associated to $\mathcal{L}$.

Proposition 4.6 *Given a set of generators $\mathcal{M}$, the dimension of the $\mathcal{A}(\mathcal{M})$ is the maximal number of linearly independent words W over $\mathcal{M}$.*

Proof. Let w be the maximal number of linearly independent words over $\mathcal{M}$.

$\dim(\mathcal{A}(\mathcal{M})) \geq w$**:** Obviously there are already w linearly independent objects in $\mathcal{A}(\mathcal{M})$, namely the linearly independent words $\mathbf{W} \in \mathcal{W}$. The dimension of $\mathcal{A}(\mathcal{M})$ must thus be at least as large as w.

$\dim(\mathcal{A}(\mathcal{M})) \leq w$**:** Suppose there is an element $\mathbf{A}$ in $\mathcal{A}(\mathcal{M})$ which is linearly independent of the words over $\mathcal{M}$, i.e.

$$\mathbf{A} \neq \sum_{\mathbf{W}_i \in \mathcal{W}} x_i \mathbf{W}_i$$

for any $x_i \neq 0$. But this contradicts the assumption that $\mathbf{A}$ is in the algebra generated by $\mathcal{M}$, i.e. that $\mathbf{A}$ *is* a linear combination of words over $\mathcal{M}$.

□

We will use the notion of dimension to quantify the expressive power of a language. Since for any non-trivial language on an infinite set of stores the corresponding algebra is an infinite-dimensional vector space, we will consider for a given language $\mathcal{L}(X)$ its finite approximations $\mathcal{L}^s_t(X)$, that is the language restricted to sets of $n(s,t) < \infty$ stores. For these approximations we can calculate the dimension of the associated algebras. This dimension is clearly a function of the number of stores $n(s,t)$, and we can use its *growth rate* as a measure for the expressiveness of $\mathcal{L}(X)$.

Example 4.7 Consider the language $\mathcal{L}^2_1(\mathbf{tell})$. For this language we have two generators. Using the enumeration of stores

$$\{\!\!|\,|\!\!\}, \{\!\!|t|\!\!\}, \{\!\!|t,t|\!\!\}, \omega,$$

the program **stop** is represented by the identity matrix (indicating zero entries simply by .):

$$\mathbf{I} = [\![\mathbf{stop}]\!] = \begin{pmatrix} 1 & . & . & . \\ . & 1 & . & . \\ . & . & 1 & . \\ . & . & . & 1 \end{pmatrix}$$

and the basic action $\textbf{tell}(t)$ is represented by:

$$\mathbf{M} = [\![\textbf{tell}(t)]\!] = \begin{pmatrix} . & 1 & . & . \\ . & . & 1 & . \\ . & . & . & 1 \\ . & . & . & 1 \end{pmatrix}$$

We can use $\mathbf{M}$ to generate two additional linearly independent operators, namely

$$\mathbf{M}^2 = \mathbf{M} \cdot \mathbf{M} = [\![\textbf{tell}(t).\textbf{tell}(t)]\!] = \begin{pmatrix} . & . & 1 & . \\ . & . & . & 1 \\ . & . & . & 1 \\ . & . & . & 1 \end{pmatrix}$$

and

$$\mathbf{M}^3 = \mathbf{M} \cdot \mathbf{M} \cdot \mathbf{M} = [\![\textbf{tell}(t).\textbf{tell}(t).\textbf{tell}(t)]\!] = \begin{pmatrix} . & . & . & 1 \\ . & . & . & 1 \\ . & . & . & 1 \\ . & . & . & 1 \end{pmatrix}$$

For all $i > 3$ we have $\mathbf{M}^i = \mathbf{M}^{i-1}$. Hence the dimension of the algebra $\mathcal{A}(\textbf{tell}(t))$ is 4. More precisely the algebra generated by $\textbf{tell}(t)$ is of the form:

$$\mathcal{A}(\textbf{tell}(t)) = \left\{ \begin{pmatrix} a & b & c & d \\ . & a & b & c+d \\ . & . & a & b+c+d \\ . & . & . & a+b+b+c \end{pmatrix} \;\middle|\; a, b, c, d \in \mathbb{R} \right\}.$$

In order to determine the dimension of Linda-like languages it will be sufficient to consider — as in the example above — only the words generated by the basic actions (cf. Proposition 4.6).

It is clear that every program in $\mathcal{L}(\textbf{tell})$, $\mathcal{L}(\textbf{ask}, \textbf{tell})$, etc, is a choice between sequential compositions of **ask**, **nask**, **get** and **tell**. Using the distributivity in the algebra of matrices representing the semantics of programs we assume that there are only "top level" choices, e.g.:

$$[\![P_1.(P_2 + P_3)]\!] = [\![P_1]\!] \cdot ([\![P_2]\!] + [\![P_3]\!]) = [\![P_1]\!] \cdot [\![P_2]\!] + [\![P_1]\!] \cdot [\![P_3]\!]$$

As the sum, i.e. linear combination, does not contribute to the dimension of algebra generated we can restrict ourself to determine how many (purely) sequential programs there are in a particular language $\mathcal{L}(X)$ which have linearly independent matrices associated by the semantics. These sequential programs

$$
\begin{tikzcd}
\mathcal{L}(\text{tell}) \arrow[d] \arrow[dr] & & \\
\mathcal{L}(\text{nask, tell}) \arrow[d] & \mathcal{L}(\text{ask, tell}) \arrow[dl] \arrow[d] & \\
\mathcal{L}(\text{ask, nask, tell}) \arrow[d] & \mathcal{L}(\text{get, tell}) \arrow[r, leftrightarrow] \arrow[d] & \mathcal{L}(\text{ask, get, tell}) \\
\mathcal{L}(\text{ask, nask, get, tell}) \arrow[r, leftrightarrow] & \mathcal{L}(\text{nask, get, tell}) &
\end{tikzcd}
$$

Fig. 5. The hierarchy of languages.

are obviously each represented by a word generated by the basic actions, i.e. generators in $\mathcal{M}$.

4.2 Measuring the Expressiveness of Linda-like Languages

By Proposition 3.3, the same hierarchy on the Linda-like languages established in [1] with respect to the *modular embedding* also holds with respect to the *linear embedding.*

The whole set of separation and equivalence results are summarised in Figure 5, where an arrow from a language $\mathcal{L}_1$ to a language $\mathcal{L}_2$ means that $\mathcal{L}_2$ embeds $\mathcal{L}_1$, that is $\mathcal{L}_1 \leq \mathcal{L}_2$. Note that, thanks to the transitivity of embedding, the figure contains only a minimal amount of arrows. However, apart from these induced relations, no other relation holds. In particular, when there is one arrow from $\mathcal{L}_1$ to $\mathcal{L}_2$ but there is no arrow from $\mathcal{L}_2$ to $\mathcal{L}_1$, then $\mathcal{L}_1$ is strictly less expressive than $\mathcal{L}_2$, that is $\mathcal{L}_1 < \mathcal{L}_2$.

We now apply the technique described in Section 4.2 to annotate this hierarchy with quantities describing the difference in expressiveness of two languages which are qualitatively separated in the hierarchy. As explained in Section 4.2 such quantities are given in terms of the rate of growth of the dimensions of the algebras associated to the languages when increasing the number of stores $n(s,t)$.

We will show that this quantitative notion of expressiveness induces an equivalence relation on the set of languages $\mathcal{L}(X)$ which is coarser than the one represented in Figure 5: it identifies languages which are separated by modular embedding. In particular, we will show that the set of languages $\mathcal{L}(X)$ can be partitioned in three classes by the equivalence relation $\approx$ defined in the following. This relation identifies two languages whose dimensions have the same rate of growth. By using the notation in [4, (9.8)], this can be defined for two generic functions $f(n)$ and $g(n)$ as follows:

$$f(n) \asymp g(n) \text{ iff } |f(n)| \leq k|g(n)| \text{ and } |g(n)| \leq k|f(n)|,$$

for some constant k and for all sufficiently large n.

Definition 4.8 Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two Linda-like languages, and let $(\mathcal{L}_1)_n$ and $(\mathcal{L}_2)_n$ be their approximations on $n < \infty$ stores. Then

$$\mathcal{L}_1 \approx \mathcal{L}_2 \text{ iff } dim((\mathcal{L}_1)_n) \asymp dim((\mathcal{L}_2)_n).$$

Strictly speaking, the dimension of the truncated languages we consider $\mathcal{L}(X)_n = \mathcal{L}(X)_{n(s,t)} = \mathcal{L}(X)^s_t$ depends on two parameters, namely the store size s and the number of distinct tuples t. In order to simplify our treatment of the languages comparison we will concentrate on the "diagonal growth", i.e. the case $s = t$. This corresponds to assuming that when n tends to infinity, s and t tend to infinity with the same speed, that is the ratio s/t is 1.

Proposition 4.9 *The quotient set $\mathcal{L}(X)/_{\approx}$ consists of the following three classes:*

$$\begin{aligned}
[\mathcal{L}(\mathbf{tell})]/_{\approx} &= \{\mathcal{L}(\mathbf{tell})\},\\
[\mathcal{L}(\mathbf{ask},\mathbf{tell})]/_{\approx} &= \{\mathcal{L}(\mathbf{ask},\mathbf{tell}),\mathcal{L}(\mathbf{nask},\mathbf{tell}),\mathcal{L}(\mathbf{ask},\mathbf{nask},\mathbf{tell})\},\\
[\mathcal{L}(\mathbf{get},\mathbf{tell})]/_{\approx} &= \{\mathcal{L}(\mathbf{get},\mathbf{tell}),\mathcal{L}(\mathbf{ask},\mathbf{get},\mathbf{tell}),\mathcal{L}(\mathbf{nask},\mathbf{get},\mathbf{tell}),\\
&\qquad \mathcal{L}(\mathbf{ask},\mathbf{nask},\mathbf{get},\mathbf{tell})\}.
\end{aligned}$$

The proof of this proposition will be given by calculating the dimension of each language in $\mathcal{L}(X)$. In general, it is easy to see that there is an upper limit for the dimension for all $(\mathcal{L}(X))_n$ given by $(n+1)^2$, i.e. the dimension of the algebra of all $n+1 \times n+1$-matrices modelling the programs in $(\mathcal{L}(X))_n$. The particular nature of the overflow ω allows us to tighten this upper limit:

Proposition 4.10 *For all languages $(\mathcal{L}(X))_n$ we have:*

$$\dim(\mathcal{L}(X))_n \leq n(n+1)+1.$$

Proof. The general form of a matrix representing a program in $(\mathcal{L}(X))_n$ for $n = n(s,t)$ is:

$$\mathbf{M} = \begin{pmatrix} m_{11} & m_{12} & \dots & m_{1n} & m_{1,n+1} \\ m_{21} & m_{22} & \dots & m_{2n} & m_{2,n+1} \\ \dots & \dots & \dots & \dots & \dots \\ m_{n1} & m_{n2} & \dots & m_{nn} & m_{n,n+1} \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}$$

The values of entries m_{ij} depend on the program represented by $\mathbf{M}$. However the last row, corresponding to the overflow state ω, is the same for all matrices representing any program in $(\mathcal{L}(X))_n$, and expresses the fact that once it is reached it is not possible to make a transition to any other state. $\square$

4.2.1 The Language $\mathcal{L}(\mathbf{tell})$

In this section we calculate the dimension of the languages $\mathcal{L}^s_t(\mathbf{tell})$ approximating $\mathcal{L}(\mathbf{tell})$.

For the product $[\![P]\!] \cdot [\![Q]\!]$ of two operators $[\![P]\!]$ and $[\![Q]\!]$ representing two programs in $\mathcal{L}_t^s(\mathbf{tell})$ the following observation is often helpful.

Lemma 4.11 *Let $[\![P]\!] = (\mathbf{P})_{ij}$ and $[\![Q]\!] = (\mathbf{Q})_{ij}$ be two operators representing two programs P and Q in $\mathcal{L}_t^s(\mathbf{tell})$, and let $\mathcal{S}$ be the set of stores. Then the entry $(\mathbf{PQ})_{lk}$ in their product matrix $\mathbf{PQ}$ is given by:*

$$(\mathbf{PQ})_{lk} = \begin{cases} 1 \text{ if there exist } R \in \mathcal{L}(X) \text{ and } s_m \in \mathcal{S}: \\ \qquad \langle P, s_l \rangle \longrightarrow \langle R, s_m \rangle \wedge \langle R, s_m \rangle \longrightarrow \langle P, s_k \rangle \\ 0 \text{ otherwise,} \end{cases}$$

where $\longrightarrow$ is a transition relation defining a small step operational semantics for $\mathcal{L}_t^s(\mathbf{tell})$.

Proof. The multiplication of linear operators is defined by:

$$(\mathbf{PQ})_{lk} = \sum_m \mathbf{P}_{lm} \cdot \mathbf{Q}_{mk}$$

As $[\![P]\!] = (\mathbf{P})_{ij}$ and $[\![Q]\!] = (\mathbf{Q})_{ij}$ have 0/1 entries the entry $(\mathbf{PQ})_{lk}$ is non-zero if and only if there exists at least one summand $\mathbf{P}_{lm} \cdot \mathbf{Q}_{mk} \neq 0$. This means that there must be a store $s_m \in \mathcal{S}$ such that $\mathbf{P}_{lm} \neq 0$ and $\mathbf{Q}_{mk} \neq 0$, that is the transitions $\langle P, s_l \rangle \longrightarrow \langle R, s_m \rangle$ and $\langle R, s_m \rangle \longrightarrow \langle P, s_k \rangle$ must take place. $\square$

In $\mathcal{L}_t^s(\mathbf{tell})$ we refer to a program of the form $P \equiv \mathbf{tell}(t_1).\mathbf{tell}(t_2)\ldots\mathbf{tell}(t_i)$ as a *prefix of length i*. For each pair of values (s, t), the language $\mathcal{L}_t^s(\mathbf{tell})$ satisfies the following properties

Proposition 4.12 *For the language $\mathcal{L}_t^s(\mathbf{tell})$ we have that*

(i) *All prefixes P of length $i > s$ are equivalent.*

(ii) *All permutations of a prefix P of length i, with $1 \leq i \leq s$ are equivalent to P.*

Proof.

(i) Consider a prefix P of length s. Because of the limited size of the store any number of $\mathbf{tell}(t_0)$, $t_0 \in \mathcal{T}$ following P can only produce an overflow. Therefore, by Lemma 4.11 all matrices for $P.Q$ with Q a prefix of length ≥ 1 will be of the form:

$$M = [\![P.Q]\!] = \begin{pmatrix} 0 & 0 & \ldots & 0 & 1 \\ 0 & 0 & \ldots & 0 & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 0 & 1 \\ 0 & 0 & \ldots & 0 & 1 \end{pmatrix}.$$

(ii) Let $P \equiv \mathbf{tell}(t_1).\mathbf{tell}(t_2)\ldots\mathbf{tell}(t_i)$, let π be any permutation of $\{1, \ldots, i\}$, and let $Q \equiv \mathbf{tell}(t_{\pi(1)}).\mathbf{tell}(t_{\pi(2)})\ldots\mathbf{tell}(t_{\pi(i)})$.

Independently of the order in which the $\mathbf{tell}(t_j)$, $j \in \{1, \dots, i\}$, are executed we always end up with the same final store $\{\!|t_1, t_2, \dots, t_i|\!\}$. Therefore, the matrices associated to the programs P and Q are the same (by Lemma 4.11).

$\square$

Proposition 4.13 *For the languages* $\mathcal{L}_t^s(\mathbf{tell})$ *we have:*

$$\dim(\mathcal{L}_t^s(\mathbf{tell})) = n(s,t) + 1.$$

Proof.

$\dim(\mathcal{L}_t^s(\mathbf{tell})) \leq n + 1$**:** By Proposition 4.12 (*ii*) we note that for each length $0 \leq i \leq s$ of the store there are exactly $\binom{i+t-1}{i}$ non-equivalent prefixes. Moreover, by Proposition 4.12 (*i*) there is a single class for all programs generating an overflow.

The number of all non-equivalent prefixes of $\mathcal{L}_t^s(\mathbf{tell})$ is therefore exactly $\sum_{i=0}^{s} \binom{i+t-1}{i} + 1 = n(s,t) + 1$.

Since all other matrices representing programs in $\mathcal{L}_t^s(\mathbf{tell})$ are linear combinations of the matrices associated to the prefixes we can conclude that $\dim(\mathcal{L}_t^s(\mathbf{tell})) \leq n + 1$.

$\dim(\mathcal{L}_t^s(\mathbf{tell})) \geq n + 1$**:** For $\mathbf{M}_j = [\![\mathbf{tell}(t_{j_1}).\mathbf{tell}(t_{j_2}) \dots \mathbf{tell}(t_{j_i})]\!]$ associated to the prefix $P_j \equiv \mathbf{tell}(t_{j_1}).\mathbf{tell}(t_{j_2}) \dots \mathbf{tell}(t_{j_i})$ we have a single non-zero entry in the first row, in position k corresponding to the store $\{\!|j_1, j_2, \dots, j_i|\!\}$:

$$(\mathbf{M}_{j_1, j_2, \dots, j_i})_{1,k} \begin{cases} 0 & \text{for } k \text{ corresponding to } \{\!|j_1, j_2, \dots, j_i|\!\} \\ 1 & \text{otherwise.} \end{cases}$$

The matrices associate to the following $n(s,t) + 1$ non-equivalent prefixes $\mathbf{stop}, \mathbf{tell}(t_1), \mathbf{tell}(t_2) \dots, \mathbf{tell}(t_t), \mathbf{tell}(t_1).\mathbf{tell}(t_1), \dots, \mathbf{tell}^{s+1}(t_1)$ are therefore linearly independent. This shows that $\dim(\mathcal{L}_t^s(\mathbf{tell})) \geq n + 1$.

$\square$

4.2.2 The Languages in $[\mathcal{L}(\mathbf{get}, \mathbf{tell})]/_{\approx}$

We compute explicitely the dimension of $\mathcal{L}_t^s(\mathbf{get}, \mathbf{tell})$ and we show that this is exactly the same as the dimension of $\mathcal{L}_t^s(\mathbf{ask}, \mathbf{get}, \mathbf{tell})$ and $\mathcal{L}_t^s(\mathbf{nask}, \mathbf{get}, \mathbf{tell})$.

We will need the following lemma which guarantees the existence of an appropriate enumeration of stores.

Lemma 4.14 *There exists an enumeration* $\sigma_1 = \{\!|\,|\!\}, \sigma_2, \dots, \sigma_n, \sigma_{n+1} = \omega$ *of stores for* $\mathcal{L}(X)_n$ *which is compatible with the inclusion order, i.e.*

$$\sigma_i \subseteq \sigma_j \;\Rightarrow\; i \leq j.$$

Proof. Consider an ordering where $\sigma_0 = \{\!|\,|\!\}$, then (in any order) all singleton stores, i.e. stores with $|\sigma| = 1$, then all stores (in any order) with $|\sigma| = 2$, etc. $\square$

We now show that the dimension of $\mathcal{L}_t^s(\mathbf{get}, \mathbf{tell})$ is equal to the maximal dimension for any language $\mathcal{L}_t^s$ as established in Proposition 4.10.

Proposition 4.15 *For the the languages* $\mathcal{L}_t^s(\mathbf{tell}, \mathbf{get})$ *we have:*

$$\dim(\mathcal{L}_t^s(\mathbf{tell}, \mathbf{get})) = n(n+1) + 1.$$

Proof. The proof is based on the fact that for an enumeration of stores such that

$$\sigma_i \subseteq \sigma_j \;\Rightarrow\; i \leq j,$$

(cf Lemma 4.14) one can construct for each pair of stores σ_i, σ_j a program

$$P_{\sigma_i,\sigma_j} \in \mathcal{L}_t^s(\mathbf{tell}, \mathbf{get})$$

such that its semantics $[\![P_{\sigma_i,\sigma_j}]\!] = \mathbf{M}_{ij}$ is given by a matrix of the form:

$$\mathbf{M}_{ij} = \begin{pmatrix} 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & \dots & 0 & 1 & ? & \dots & ? & ? \\ ? & \dots & ? & ? & ? & \dots & ? & ? \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ ? & \dots & ? & ? & ? & \dots & ? & ? \\ 0 & \dots & 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix}$$

where ? denotes any matrix entry. The definition of $\mathbf{M}_{ij}$ is therefore:

$$(\mathbf{M}_{ij})_{kl} = \begin{cases} 0 & \text{if } k < i \\ 0 & \text{if } k = i \text{ and } l < j \\ 1 & \text{if } k = i \text{ and } l = j \\ 0 & \text{if } k = n+1 \text{ and } l < n+1 \\ 1 & \text{if } k = l = n+1 \\ ? & \text{otherwise.} \end{cases}$$

For example, the program P_{σ_i,σ_j} for $\sigma_i = \{\!|t_{i_1}, t_{i_2}, \dots, t_{i_{s_i}}|\!\}$ and $\sigma_j = \{\!|t_{j_1}, t_{j_2}, \dots, t_{i_{s_j}}|\!\}$ is given by $P_{\sigma_i,\sigma_j} = \mathbf{get}(\sigma_i).\mathbf{tell}(\sigma_j)$, or more precisely by

$$P_{\sigma_i,\sigma_j} = \mathbf{get}(t_{i_1}).\mathbf{get}(t_{i_2}) \dots \mathbf{get}(t_{i_{s_i}}).\mathbf{tell}(t_{i_1}).\mathbf{tell}(t_{i_2}) \dots \mathbf{tell}(t_{i_{s_i}}).$$

For $i = j = 1$, $P_{\sigma_i,\sigma_j} = \mathbf{stop}$.

For a store σ_k with $k \not\leq i$ we have $\sigma_k \not\subseteq \sigma_i$. Therefore there is at least one (multiple occurrence of) $t \in \{\!|t_{i_1}, t_{i_2}, \dots, t_{i_{s_i}}|\!\}$ for which the corresponding $\mathbf{get}(t)$ in P_{σ_i,σ_j} fails. That means that there is no transition for P_{σ_i,σ_j} if executed in σ_k and thus the corresponding row in $\mathbf{M}_{ij}$ contains only zeros.

As P_{σ_i,σ_j} is deterministic we know that the row corresponding to any store — in particular for store σ_i — contains at most one non-zero entry. Clearly this non-zero entry in row σ_i is in the column corresponding to σ_j as P_{σ_i,σ_j} first removes all tuples in σ_i to obtain the empty store $\{\!|\,|\!\}$ and then puts all tuples of σ_j back into the store.

The remaining entries in $\mathbf{M}_{ij}$ can take any value except for the last row which corresponds to the overflow state ω.

Clearly the matrices $\mathbf{M}_{ij}$ are linearly independent as for each pair of stores we get a different leading 1.

There are exactly $n(n+1)+1$ such matrices. Since the maximal number of linearly independent matrices for any language $\mathcal{L}_t^s(X)$ is also $n(n+1)+1$, this is exactly the dimension of $\mathcal{L}_t^s(\mathbf{tell}, \mathbf{get})$. □

Example 4.16 Consider the language $\mathcal{L}_1^1(\mathbf{tell}, \mathbf{get})$. The number of possible stores, including the over-full state, is $\binom{1+1}{1} + 1 = \binom{2}{1} + 1 = 3$, namely:

$$\{\!|\,|\!\}, \{\!|t|\!\}, \omega.$$

The dimension of $\mathcal{L}_1^1(\mathbf{tell}, \mathbf{get})$ is therfore:

$$\dim(\mathcal{L}_1^1(\mathbf{tell}, \mathbf{get})) = 2(2+1)+1 = 7.$$

The seven linearly independent matrices and their corresponding programs are given as follows:

$$\begin{pmatrix} 1 & ? & ? \\ ? & ? & ? \\ 0 & 0 & 1 \end{pmatrix} = [\![\mathbf{stop}]\!]$$

$$\begin{pmatrix} 0 & 1 & ? \\ ? & ? & ? \\ 0 & 0 & 1 \end{pmatrix} = [\![\mathbf{tell}(t)]\!] \qquad \begin{pmatrix} 0 & 0 & 1 \\ ? & ? & ? \\ 0 & 0 & 1 \end{pmatrix} = [\![\mathbf{tell}(t).\mathbf{tell}(t)]\!]$$

$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & ? & ? \\ 0 & 0 & 1 \end{pmatrix} = [\![\mathbf{get}(t)]\!] \qquad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & ? \\ 0 & 0 & 1 \end{pmatrix} = [\![\mathbf{get}(t).\mathbf{tell}(t)]\!]$$

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} = [\![\mathbf{get}(t).\mathbf{tell}(t).\mathbf{tell}(t)]\!] \qquad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = [\![\mathbf{get}(t).\mathbf{get}(t)]\!]$$

It is now easy to see that the languages $\mathcal{L}(\mathbf{nask}, \mathbf{get}, \mathbf{tell})$, $\mathcal{L}(\mathbf{ask}, \mathbf{get}, \mathbf{tell})$, and $\mathcal{L}(\mathbf{ask}, \mathbf{nask}, \mathbf{get}, \mathbf{tell})$ are all equivalent to $\mathcal{L}(\mathbf{get}, \mathbf{tell})$.

Corollary 4.17 *For the languages* $\mathcal{L}(\mathbf{nask}, \mathbf{get}, \mathbf{tell})$*,* $\mathcal{L}(\mathbf{ask}, \mathbf{nask}, \mathbf{get}, \mathbf{tell})$*, and* $\mathcal{L}(\mathbf{ask}, \mathbf{get}, \mathbf{tell})$ *we have:*

$$\dim(\mathcal{L}^s_t(\mathbf{ask}, \mathbf{get}, \mathbf{tell})) = n(n+1) + 1$$
$$\dim(\mathcal{L}^s_t(\mathbf{nask}, \mathbf{get}, \mathbf{tell})) = n(n+1) + 1$$
$$\dim(\mathcal{L}^s_t(\mathbf{ask}, \mathbf{nask}, \mathbf{get}, \mathbf{tell})) = n(n+1) + 1$$

and therefore we get:

$$\begin{aligned}\mathcal{L}(\mathbf{get}, \mathbf{tell}) &\approx \mathcal{L}(\mathbf{nask}, \mathbf{get}, \mathbf{tell})\\ &\approx \mathcal{L}(\mathbf{ask}, \mathbf{nask}, \mathbf{get}, \mathbf{tell})\\ &\approx \mathcal{L}(\mathbf{ask}, \mathbf{get}, \mathbf{tell}).\end{aligned}$$

Proof. For any subset of communication actions X which contains $\{\mathbf{get}, \mathbf{tell}\}$ the algebras associated to the languages $\mathcal{L}^s_t(X)$ contain $\mathcal{A}(\{\mathbf{get}, \mathbf{tell}\})$, so their dimension cannot be smaller than the dimension of $\mathcal{L}^s_t(\mathbf{get}, \mathbf{tell})$. Therefore, by Proposition 4.10 and Proposition 4.15 we have:

$$n(n+1) + 1 = \dim(\mathcal{L}^s_t(\mathbf{get}, \mathbf{tell})) \leq \dim(\mathcal{L}^s_t(X)) \leq n(n+1) + 1.$$

This shows that for any set X_1 and X_2 of communication primitives which contain $\{\mathbf{get}, \mathbf{tell}\}$ we have: $\mathcal{L}^s_t(X_1) \asymp \mathcal{L}^s_t(X_2)$. □

4.2.3 The Languages in $[\mathcal{L}(\mathbf{ask}, \mathbf{tell})]/_{\approx}$

Analogously to the case of $\mathcal{L}(\mathbf{tell})$, we construct standard representations of sequential programs containing all the primitives of the languages we are considering, namely **tell**s and the "passive" guards **ask** and **nask**.

Lemma 4.18 *Every program* P *which is a sequential composition of* **ask**, **nask** *and* **tell** *respectively in* $\mathcal{L}(\mathbf{ask}, \mathbf{tell})$*,* $\mathcal{L}(\mathbf{nask}, \mathbf{tell})$ *and* $\mathcal{L}(\mathbf{ask}, \mathbf{nask}, \mathbf{tell})$ *is equivalent to a program of the form:*

$$\mathbf{ask}(t_{a_1}) \ldots \mathbf{ask}(t_{a_k}).\mathbf{nask}(t_{b_1}) \ldots \mathbf{nask}(t_{b_l}).\mathbf{tell}(t_{c_1}) \ldots \mathbf{tell}(t_{c_m})$$

with $t_{a_1}, \ldots, t_{a_k}, t_{b_1}, \ldots, t_{b_l}$ *all different, or a blocked program represented by the zero matrix* $\mathbf{O}$.

Proof. The proof is based on a number of obvious structural equivalences, which allow either to remove a **ask** or **nask** which is not at the beginning of P, to move them to the beginning of P, or to identify P as being equivalent to the zero matrix $\mathbf{O}$.

$$\mathbf{ask}(t_i).\mathbf{ask}(t_i) \equiv \mathbf{ask}(t_i)$$
$$\mathbf{ask}(t_i).\mathbf{ask}(t_j) \equiv \mathbf{ask}(t_j).\mathbf{ask}(t_i)$$

$$\mathbf{nask}(t_i).\mathbf{nask}(t_i) \equiv \mathbf{nask}(t_i)$$
$$\mathbf{nask}(t_i).\mathbf{nask}(t_j) \equiv \mathbf{nask}(t_j).\mathbf{nask}(t_i)$$

$$\mathbf{tell}(t_i).\mathbf{tell}(t_j) \equiv \mathbf{tell}(t_j).\mathbf{tell}(t_i)$$

$$\begin{aligned}
\mathbf{ask}(t_j).\mathbf{nask}(t_i) &\equiv \mathbf{nask}(t_j).\mathbf{ask}(t_i) \\
\mathbf{ask}(t_i).\mathbf{nask}(t_i) &\equiv \mathbf{O} \\
\mathbf{nask}(t_i).\mathbf{ask}(t_i) &\equiv \mathbf{O} \\
\mathbf{tell}(t_i).\mathbf{ask}(t_i) &\equiv \mathbf{tell}(t_i) \\
\mathbf{tell}(t_i).\mathbf{ask}(t_j) &\equiv \mathbf{ask}(t_j).\mathbf{tell}(t_i) \\
\mathbf{tell}(t_i).\mathbf{nask}(t_i) &\equiv \mathbf{O} \\
\mathbf{tell}(t_i).\mathbf{nask}(t_j) &\equiv \mathbf{nask}(t_j).\mathbf{tell}(t_i)
\end{aligned}$$

□

This canonical representation of sequential programs allows us to enumerate all basic programs.

Lemma 4.19 *The number of standard or canonical programs in* $\mathcal{L}(\mathbf{ask}, \mathbf{tell})$ *and* $\mathcal{L}(\mathbf{nask}, \mathbf{tell})$ *of the forms:*

$$\mathbf{ask}(t_{a_1}) \ldots \mathbf{ask}(t_{a_k}).\mathbf{tell}(t_{c_1}) \ldots \mathbf{tell}(t_{c_m})$$

$$\mathbf{nask}(t_{b_1}) \ldots \mathbf{nask}(t_{b_l}).\mathbf{tell}(t_{c_1}) \ldots \mathbf{tell}(t_{c_m})$$

is given by

$$\left(\sum_{i=0}^{t} \binom{t}{i}\right) \left(\sum_{j=0}^{s} \binom{j+t-1}{i}\right) = \left(\sum_{i=0}^{t} \binom{t}{i}\right) \binom{s+t}{s}.$$

The number of standard programs in $\mathcal{L}(\mathbf{ask}, \mathbf{nask}, \mathbf{tell})$ *of the form:*

$$\mathbf{ask}(t_{a_1}) \ldots \mathbf{ask}(t_{a_k}).\mathbf{nask}(t_{b_1}) \ldots \mathbf{nask}(t_{b_l}).\mathbf{tell}(t_{c_1}) \ldots \mathbf{tell}(t_{c_m})$$

is

$$\sum_{i=0}^{t} \binom{t}{i} \sum_{k=0}^{i} \binom{i}{k} \left(\sum_{j=0}^{s} \binom{j+t-1}{i}\right) = \left(\sum_{i=0}^{t} 2^i \cdot \binom{t}{i}\right) \binom{s+t}{s}.$$

Proof.

(i) Guards: We observe that the order of **ask**s and **nask**s at the beginning of a normalised programs is irrelevant. Furthermore it makes only sense to consider one **ask**/**nask** per different token, multiple **ask**s/**nask** for the same token can be replaced by a single **ask**/**nask**.

We can therefore conclude that the number of different **ask**/**nask**-prefixes for programs in $\mathcal{L}(\mathbf{ask}, \mathbf{tell})$ and $\mathcal{L}(\mathbf{nask}, \mathbf{tell})$ is:

$$\sum_{i=0}^{t} \binom{t}{i}.$$

For programs in $\mathcal{L}(\mathbf{ask}, \mathbf{nask}, \mathbf{tell})$ we have to partition the tokens in two disjoint sets corresponding to the **ask** and **nask** guards respectively:

$$\sum_{i=0}^{t} \binom{t}{i} \sum_{k=0}^{i} \binom{i}{k} = \sum_{i=0}^{t} 2^i \cdot \binom{t}{i}$$

(ii) Tells: As in the case of the $\mathcal{L}(\mathbf{tell})$ language any number up to s **tell**s with repetition can be part of a normalised program:

$$\sum_{j=0}^{s} \binom{j+t-1}{i} = \binom{s+t}{s}.$$

□

However not all of these basic programs are linearly independent. Basically, whenever we **ask** or **nask** for a particular token t a number of initial stores will result in a blocked program — those which contain or do not contain t. This means that certain rows in the matrix representing $\mathbf{tell}(t_{c_1}) \ldots \mathbf{tell}(t_{c_m})$ are "deleted", i.e. set to be zero.

The result of this is that all those **tell**-sequences for which the matrices are different on only those "deleted" rows result in linearly independent matrices when combined with certain **ask**s or **nask**s.

Example 4.20 Consider the language $\mathcal{L}^2_2(\mathbf{tell}, \mathbf{ask})$. The number of possible stores, including the over-full state, is $\binom{2+2}{2} + 1 = \binom{4}{2} + 1 = 6 + 1 = 7$ namely:

$$\{\!\!\{\}\!\!\}, \{\!\!\{t_1\}\!\!\}, \{\!\!\{t_2\}\!\!\}, \{\!\!\{t_1, t_1\}\!\!\}, \{\!\!\{t_1, t_2\}\!\!\}, \{\!\!\{t_2, t_2\}\!\!\}, \omega$$

Take as an example the guard sequence $\mathbf{ask}(t_1).\mathbf{ask}(t_2)$, its matrix is given by:

$$[\![\mathbf{ask}(t_1).\mathbf{ask}(t_2)]\!] = \begin{pmatrix} . & . & . & . & . & . & . \\ . & 1 & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & 1 & . & . & . \\ . & . & . & . & 1 & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & 1 \end{pmatrix} \cdot \begin{pmatrix} . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & 1 & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & 1 & . & . \\ . & . & . & . & . & 1 & . \\ . & . & . & . & . & . & 1 \end{pmatrix} = \begin{pmatrix} . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & 1 & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & 1 \end{pmatrix}.$$

If we combine the guard sequence $\mathbf{ask}(t_1).\mathbf{ask}(t_2)$ with, for example, $\mathbf{tell}(t_1)$ and $\mathbf{tell}(t_1)$ with

$$[\![\mathbf{tell}(t_1)]\!] = \begin{pmatrix} . & 1 & . & . & . & . & . \\ . & . & . & 1 & . & . & . \\ . & . & . & . & 1 & . & . \\ . & . & . & . & . & . & 1 \\ . & . & . & . & . & . & 1 \\ . & . & . & . & . & . & 1 \\ . & . & . & . & . & . & 1 \end{pmatrix} \qquad [\![\mathbf{tell}(t_2)]\!] = \begin{pmatrix} . & . & 1 & . & . & . & . \\ . & . & . & . & 1 & . & . \\ . & . & . & . & . & 1 & . \\ . & . & . & . & . & . & 1 \\ . & . & . & . & . & . & 1 \\ . & . & . & . & . & . & 1 \\ . & . & . & . & . & . & 1 \end{pmatrix}$$

we get the same matrix:

$$\begin{pmatrix} \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1 \end{pmatrix} \cdot \begin{pmatrix} \cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1 \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1 \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1 \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1 \end{pmatrix} = \begin{pmatrix} \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1 \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1 \end{pmatrix}$$

$$\begin{pmatrix} \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1 \end{pmatrix} \cdot \begin{pmatrix} \cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1 \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1 \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1 \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1 \end{pmatrix} = \begin{pmatrix} \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1 \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot \\ \cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1 \end{pmatrix}$$

representing two different canonical programs, namely $\mathbf{ask}(t_1).\mathbf{ask}(t_2).\mathbf{tell}(t_1)$ and $\mathbf{ask}(t_1).\mathbf{ask}(t_2).\mathbf{tell}(t_2)$.

Indeed we get the same result for all **tell**-sequences: Combining the guard sequence $\mathbf{ask}(t_1).\mathbf{ask}(t_2)$ and any **tell**-sequence results either in a deadlock (for all those stores which do not contain t_1 and t_2) or in an overflow (for the stores $\{\!\!\{t_1, t_2\}\!\!\}$ and ω). All these canonical programs are is represented by the same matrix.

As a result, the dimension of the algebra generated by the canonical programs — i.e. the dimension of the algebra associated to the truncated languages $\mathcal{L}_t^s(\mathbf{ask}, \mathbf{tell})$, $\mathcal{L}_t^s(\mathbf{nask}, \mathbf{tell})$ and $\mathcal{L}_t^s(\mathbf{ask}, \mathbf{nask}, \mathbf{tell})$ — is smaller than the number of canonical programs.

Proposition 4.21 *For the finite languages* $\mathcal{L}_t^s(\mathbf{ask}, \mathbf{tell})$*,* $\mathcal{L}_t^s(\mathbf{nask}, \mathbf{tell})$ *and* $\mathcal{L}_t^s(\mathbf{ask}, \mathbf{nask}, \mathbf{tell})$ *we have:*

$$\left.\begin{array}{l} \dim(\mathcal{L}_t^s(\mathbf{tell}, \mathbf{ask})) \\ \dim(\mathcal{L}_t^s(\mathbf{tell}, \mathbf{nask})) \\ \dim(\mathcal{L}_t^s(\mathbf{tell}, \mathbf{ask}, \mathbf{nask})) \end{array}\right\} = \left(\sum_{i=0}^{t} \binom{t}{i} \sum_{j=0}^{s-i} \binom{j+t-1}{j} + 1 \right) + 1.$$

Proof. The number of linearly independent matrices we get by considering the canonical programs in $\mathcal{L}_t^s(\mathbf{ask}, \mathbf{tell})$, $\mathcal{L}_t^s(\mathbf{nask}, \mathbf{tell})$ and $\mathcal{L}_t^s(\mathbf{ask}, \mathbf{nask}, \mathbf{tell})$

— and thus their dimensions — is given for all three languages by:

$$\#\{\mathbf{ask/nask}\} \cdot \#\{\text{“valid” } \mathbf{tell}\} + 1 = \sum_{i=1}^{t} \binom{t}{i} \cdot \#\{\text{“valid” } \mathbf{tell}\} + 1$$

where “valid” **tell**-sequences are those **tell**-sequences which result in linearly independent matrices. Additionally we also have to consider again the “identity program”, i.e. **stop**. The number of “valid” **tell**-sequences is the same for all three languages. However the reason for this is different for **ask**s and **nask**s.

(i) $\mathcal{L}_t^s(\mathbf{tell}, \mathbf{ask})$: If we have i different **ask**s in the prefix we know that they will block any program which is executed in a store with less than i tokens. This program might also be blocked in stores with more tokens, but we are sure that if there are less tokens in the store at least one **ask** will fail. On the other hand for every set of i different **ask**s in the prefix there is one store where the program continues past the **ask**-prefix (namely the store containing exactly the tokens we **ask** for).

This implies that after i **ask**s only at most $s - i$ **tell**s will not result in an overflow ω of the store. This therefore limits the number of different **tell** sequences which lead to non-equivalent, i.e. linearly independent, programs. We get for i **ask**s:

$$\#\{\text{“valid” } \mathbf{tell}\} = \left(\sum_{j=0}^{s-i} \binom{j+t-1}{j} + 1 \right).$$

That is: After i **ask**s we can (i) **tell** up to $s-1$ tokens and obtain a store with up to s tokens in it, or (ii) or we end up with and overflow store ω.

(ii) $\mathcal{L}_t^s(\mathbf{tell}, \mathbf{nask})$: If we have i different **nask**s in the prefix, then we know that the matrices representing the **tell**-sequences which follow will get those rows “deleted” which correspond to the stores containing these i tokens.

In other words, the **tell**-sequences which differ only on stores containing these i tokens will result in the same matrix. This means, for every i we have to subtract the number of “possible” **tell**-sequences by the number of those stores which contain the i tokens in order to obtain the number of “valid” **tell**-sequences. Therefore we get again after i **nask**s:

$$\#\{\text{“valid” } \mathbf{tell}\} = \left(\sum_{j=0}^{s-i} \binom{j+t-1}{j} + 1 \right).$$

(iii) $\mathcal{L}_t^s(\mathbf{tell}, \mathbf{ask}, \mathbf{nask})$: For this we only have to combine the arguments for the **ask**s and **nask**s in order to obtain the same expression.

□

Example 4.22 Consider the languages $\mathcal{L}_2^2(X, \mathbf{tell})$ for $X \subseteq \{\mathbf{ask}, \mathbf{nask}\}$. The possible stores are:

$$\{\!\!\{\}\!\!\}, \{\!\!\{t_1\}\!\!\}, \{\!\!\{t_2\}\!\!\}, \{\!\!\{t_1, t_1\}\!\!\}, \{\!\!\{t_1, t_2\}\!\!\}, \{\!\!\{t_2, t_2\}\!\!\}, \omega$$

Programs are therefore represented by 7×7 matrices.

The possible guard combinations are:

$$\llbracket \mathbf{stop} \rrbracket = \begin{pmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{pmatrix} \quad \llbracket \mathbf{ask}(t_1) \rrbracket = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{pmatrix} \quad \llbracket \mathbf{ask}(t_2) \rrbracket = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{pmatrix}$$

$$\llbracket \mathbf{nask}(t_1) \rrbracket = \begin{pmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \quad \llbracket \mathbf{nask}(t_1) \rrbracket = \begin{pmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \quad \llbracket \mathbf{ask}(t_1).\mathbf{ask}(t_2) \rrbracket = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{pmatrix}$$

$$\llbracket \mathbf{nask}(t_1).\mathbf{nask}(t_2) \rrbracket = \begin{pmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \quad \llbracket \mathbf{ask}(t_1).\mathbf{nask}(t_2) \rrbracket = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \quad \llbracket \mathbf{nask}(t_1).\mathbf{ask}(t_2) \rrbracket = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

The possible **tell** combinations are:

$$[\![\mathbf{tell}(t_1)]\!] = \begin{pmatrix} \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{pmatrix} \quad [\![\mathbf{tell}(t_2)]\!] = \begin{pmatrix} \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{pmatrix} \quad \mathbf{tell}(t_1).\mathbf{tell}(t_1) = \begin{pmatrix} \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{pmatrix}$$

$$[\![\mathbf{tell}(t_1).\mathbf{tell}(t_2)]\!] = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{pmatrix} \quad [\![\mathbf{tell}(t_2).\mathbf{tell}(t_2)]\!] = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{pmatrix} \quad [\![\mathbf{tell}(t_1).\mathbf{tell}(t_1).\mathbf{tell}(t_1)]\!] = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{pmatrix}$$

If we now look at combinations of guards and **tell**s. First, we observe that all six matrices for the **tell** sequences are linearly independent. If we therefore combine **stop**, i.e. the identity, with either of them we get six independent matrices. The same happens, perhaps surprisingly, if we combine the **tell** sequences with $\mathbf{nask}(t_1).\mathbf{nask}(t_2)$, as they all differ in the first row, the only one which "survives" the multiplication with $[\![\mathbf{nask}(t_1).\mathbf{nask}(t_2)]\!]$.

If we continue this way, constructing all linearly independent combinations of **ask/nask**-guards and **tell**-sequences we get as the dimension of these languages:

$$\dim(\mathcal{L}_2^2(X, \mathbf{tell})) = 18.$$

4.2.4 Comparison

We have now a method to associate to each language $\mathcal{L}(X)$ a measure of its expressiveness given in terms of the dimension

$$g_X(s,t) = \dim(\mathcal{L}_t^s(X)),$$

of its finite approximation $\mathcal{L}_t^s(X)$ on stores of finite length s containing a finite number t of tokens. We can therefore compare two languages by comparing

s	t	n	$\dim(\mathcal{L}_s^t(\mathbf{tell}))$	$\dim(\mathcal{L}_s^t(\mathbf{ask}, \mathbf{tell}))$	$\dim(\mathcal{L}_s^t(\mathbf{get}, \mathbf{tell}))$
1	1	2	3	6	7
1	2	3	4	10	13
1	3	4	5	16	21
1	4	5	6	26	31
2	1	3	4	8	13
2	2	6	7	18	43
2	3	10	11	34	111
2	4	15	16	58	241
3	1	4	5	10	21
3	2	10	11	30	111
3	3	20	21	72	421
3	4	35	36	146	1261
4	1	5	6	12	31
4	2	15	16	46	241
4	3	35	36	138	1261
4	4	70	71	338	4971

Table 1
Growth rate of $\dim(\mathcal{L}_s^t(X))$

the growth rate of the dimension of the algebras associated to their finite approximations.

The dimensions we calculated for the languages $\mathcal{L}(X)$ are functions in the two variables s and t and allow us to quantitatively compare the various languages for any fixed choice of s and t (cf. Tables 1 and 2)

In general, the analysis of the functions $g_X(s,t)$ allows us to conclude consistently with the results in [1], that the expressiveness of languages grows when the number of stores increases and this growth is much faster in those languages where the primitive **get** is present than it is in languages where it is not present. Moreover, among the latter, the languages which have nevertheless some synchronisation primitives, such as **ask**, grow in expressiveness faster than the language where only communication but no form of synchronisation can occur, namely $\mathcal{L}(\mathbf{tell})$.

By assuming as mentioned in Section 4.2, that the asymptotic growth of the two variables s and t is the same, we can reduce $g_X(s,t)$ to a function in

s	t	n	$\dim(\mathcal{L}_s^t(\mathbf{tell}))$	$\dim(\mathcal{L}_s^t(\mathbf{ask},\mathbf{tell}))$	$\dim(\mathcal{L}_s^t(\mathbf{get},\mathbf{tell}))$
1	1	2	3	6	7
2	2	6	7	18	43
3	3	20	21	72	421
4	4	70	71	338	4971
5	5	252	253	1716	63757
6	6	924	925	9054	854701
7	7	3432	3433	48768	11782057
8	8	12870	12871	265986	165649771
9	9	48620	48621	1463076	2363953021
10	10	184756	184757	8098478	34134964293

Table 2
Diagonal growth of $\dim(\mathcal{L}_s^t(X))$

one variable $t = s$, thus simplifying the analysis.

We observe that some languages which are separated by the linear embedding (as well as by the modular embedding) have nevertheless the same growth rate associated, e.g. $\mathcal{L}(\mathbf{get}, \mathbf{tell})$ and $\mathcal{L}(\mathbf{ask}, \mathbf{get}, \mathbf{tell})$. This is due to the fact that these languages generate the same algebra. In fact, by Proposition 4.3 the algebra generated by a language is "much bigger" than the actual set of operators corresponding to the programs in the language. This makes it impossible to obtain a separation as fine as the one given by the linear/modular embedding.

4.2.5 Some Numerical Results

In Table 1 we report some results showing the dimension of the algebra of $\mathcal{L}_t^s(\mathbf{tell})$,$\mathcal{L}_t^s(\mathbf{tell}, \mathbf{ask})$ and $\mathcal{L}_t^s(\mathbf{tell}, \mathbf{get})$ for some values of s and t. We also report the corresponding number $n(s,t)$ of all possible stores which gives the matrix dimension.

The growth-rate "in the diagonal" i.e. for the truncated languages with $s = t$, clearly illustrates how for the three different classes of languages the dimension of their associated algebras increases as n increases, see Table 2.

5 Concluding Remarks

The major contributions of this work are the following:

The notion of modular embedding, originally formulated for languages with set-based observables, has been reformulated for languages with a linear semantics in the style of [3]. The correspondence between the classical notion of modular embedding and the newly introduced notion of linear embedding has been established (Proposition 3.3), which guarantees that the qualitative notion can be recovered from the quantitative one.

We introduced a measure which describes the expressiveness of languages in terms of the dimensions of their corresponding algebras. This allows for a quantitative comparison of the "richness" of languages. The dimension of the algebra associated with a language can be used as a kind of "index" (in the sense of algebraic topology) for a (partial) classification of languages. Indeed, if $\dim(\mathcal{L}_1) > \dim(\mathcal{L}_2)$ then it is impossible to embed (neither linearly nor modularly) $\mathcal{L}_1$ in $\mathcal{L}_2$. Hence dimensions provide an alternative way to establish non-embeddability results.

We applied this dimension-based analysis of expressiveness to a class of Linda-like languages. As a result we obtained a quantitative comparison of those languages reflecting the hierarchy in [1].

The results presented are intended to be a first step towards a more ambitious program which will address a number of further issues, including:

- The development of a better intuitive understanding of the growth rate of the dimensions of the (approximated) languages.
- The investigation of how to associate algebraic objects which are more refined than dimensions to describe the expressiveness of languages, such as dimension groups [8].

The ultimate goal would be a complete classification of languages with respect to their expressiveness based on such generalised indices.

Finally, we would like to apply our method to other classes of languages such as concurrent constraint languages and, in particular to probabilistic languages, where the use of a linear semantics seems to be particularly appropriate.

References

[1] Brogi, A. and J. Jacquet, *On the expressiveness of Linda-like concurrent languages*, Electronic Notes in Theoretical Computer Science **16** (1998), p. 22.

[2] de Boer, F. S. and C. Palamidessi, *Embedding as a tool for language comparison*, Information and Computation **108** (1994), pp. 128–157.

[3] Di Pierro, A. and H. Wiklicky, *Linear structures for concurrency in probabilistic programming languages*, in: *Proceedings of MFCSIT00– First Irish Conference on the Mathematical Foundations of Computer Science and Information Technology, Cork, Ireland*, Electronic Notes in Theoretical Computer Science **40** (2001).

[4] Graham, R., D. Knuth and O. Patashnik, "Concrete Mathematics: A Foundation for Computer Science," Addison–Wesley, Reading, Massachusetts, 1989.

[5] Greub, W. H., "Linear Algebra," Grundlehren der mathematischen Wissenschaften **97**, Springer Verlag, New York, 1967, third edition.

[6] Grimaldi, R. P., "Discrete and Combinatorial Mathematics — An Applied Introduction," Addison-Wesley, Reading, Massachusetts, 1999, forth edition.

[7] Ito, K., editor, "Encyclopedic Dictonary of Mathematics," MIT Press, Cambridge, Massachusetts – London, England, 1987, second english edition.

[8] Lind, D. and B. Marcus, "An Introduction to Symbolic Dynamics and Coding," Cambridge University Press, Cambridge – New York – Melbourne, 1995.

[9] Shapiro, E., *Embeddings among concurrent programming languages*, in: W. R. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, Lecture Notes in Computer Science **630** (1992), pp. 486–503.