SEQUENTIAL AND PARALLEL COMPUTATION STRATEGIES ON COHERENCE SPACES

Ruben Gerardo Schneider Sellanes\(^1\)
Instituto de Informática - UFRGS
gerardo@inf.ufrgs.br

Antônio Carlos da Rocha Costa\(^1\)
Instituto de Informática - UFRGS
rocha@inf.ufrgs.br

Abstract

Curien defined the sequential algorithms as mathematical objects. For all Distributive Concrete Data Structure (dcds) \(M, M'\), he defines a dcds \(M \Rightarrow M'\), the states of which are the sequential algorithms from \(M\) to \(M'\). For all cdss's \(M\) and \(M'\) that are, in particular, webs of coherence spaces, we define the linear algorithm as a state of a cdss denoted by \(!M \Rightarrow M'\). Given a stable function \(f: M \rightarrow M'\), we can obtain a linear algorithm that contains all the strategies of computation for computing \(f\). In fact, this linear algorithm can be considered as a "meta-algorithm". We define a strategy of computation so as one can use such notion to give compositional operational semantics to programs segments.

1 Introduction

Girard ([GIR 89]) showed that, given a stable function \(f: M \rightarrow M'\), we can obtain a linear function \(lin(f)\):

\(!M \rightarrow M'\), where "\(\Rightarrow\)" is the "of course" operator of Linear Logic. This gives us the idea that from the

\(^1\) Author’s address: Instituto de Informática - UFRGS, Caixa Postal 15064. CEP: 91501-970, Porto Alegre, RS. Brasil. F: (+55-51) 336-8399, Fax: (+55-51) 336-5576
definition of sequential algorithms (that are obtained from the sequential functions, which are stable functions) we can obtain a similar concept: the linear algorithms\(^2\).

In [ZHA 89] Zhang shows that a coherence space can be considered as a particular case of dl-domains and of event structures. On the other hand, in [SCH 95] the author studies the relationships among coherence spaces, event domain and concrete domain, and among their “concrete” counterparts: webs of coherence spaces, event structures and concrete data structures (cds). From those results, we know that a cds \(M = (C, V, E, \models)\) is a web of a coherence space iff for all cell \(c \in C\), \(\emptyset \models c\). The fact above implies that the set of states of a cds such that for all cell \(c \in C\), \(\emptyset \models c\), is a coherence space.

In this paper we define, for all cds \(M \in M'\), the linear algorithms as states of a cds denoted by \(!M \rightarrow M'\). We consider cds that are, in particular, webs of coherence spaces, but we claim that this notion can be extended to any cds. We obtain, constructively, a linear algorithm \(A\) (a state of the cds \(!M \rightarrow M'\)) from any given stable function \(f : M \rightarrow M'\). This linear algorithm “contains” all the sequential and parallel strategies of computation that compute \(f\). We define a strategy of computation so that one can use it to give the semantics of segments of programs, and this gives a sort of compositional operational semantics.

### 2 Concrete Data Structures

In this section we define the concrete data structure (cds) and their states, following [CUR 86].

#### 2.1 Definition

A concrete data structure or cds \((C, V, E, \models)\) is given by three sets \(C, V, E\) of cells, of values and of events such that

\[
E \subseteq C \times V \text{ and } \forall c \in C, \exists v \in V \text{ s.t. } (c, v) \in E \text{ ("any cell may be filled")},
\]

\(^2\)The name is due to the fact that the algorithms thus obtained compute linear functions.
and a relation $\vdash$, called accessibility relation between finite parts of $E$ and elements of $C$. We say that
\{e_1, \ldots, e_n\} is an enabling of $c$ if $\{e_1, \ldots, e_n\} \vdash c$. A cell $c$ such that $\vdash c$ is called initial. $C$ and $V$ are supposed countable.

A state is a subset $x$ of $E$ such that

1. $(c, v_1), (c, v_2) \in x \Rightarrow v_1 = v_2$

2. if $(c, v) \in x$, then there exists a sequence of events $e_0, \ldots, e_n = (c, v)$ such that $e_i = (c_i, v_i) \in x$ and $\{e_i \mid j < i\}$ contains an enabling of $c_i$ for all $i \leq n$.

The conditions 1. and 2. are called consistence and safety respectively. A sequence as described in condition 2. is a deduction.

The sets of states of a cds $M$, ordered by inclusion, is a partial order denoted by $(D(M), \leq)$ (or $(D(M), \subseteq)$).

If $D$ is isomorphic to $D(M)$, we say that $M$ generates or represents $D$. $\blacksquare$

The following definitions contain useful notation.

2.2 Definition

Let $x$ be a set of events of a cds. A cell $c$ is

- filled (with $v$) in $x$ iff $(c, v) \in x$

- enabled in $x$ iff $x$ contains an enabling of $c$

- accessible from $x$ iff it is enabled, but not filled in $x$.

We denote by $F(x)$, $E(x)$ and $A(x)$ the sets of cells filled, enabled and accessible in or from $x$. $\blacksquare$

2.3 Definition

Let $x$ and $y$ be states of a cds $M$. Then we say that $y$ covers $x$, we write $x \prec y$, if $x \prec y$ and $(\forall z, x \prec z \leq y \Rightarrow z = y)$.

We write $x \prec y$ (or $x \prec y$) iff $c \in A(x)$, $c \in F(y)$ and $x \prec y$ (or $x \prec y$). $\blacksquare$
3 Linear Algorithms

Following in an analogous way Curien's definition of sequential algorithms, we define here the linear algorithms, which are more general than the sequential ones. The "of course" operator ("!!") has a fundamental role here. If we want to have parallel strategies, we must allow, on the algorithms, the possibility of doing as many operations as necessary. This implies that the "valof" and the "output" commands should be modified, as well as the notion of state: if \( M \) is a cds (in particular a web of a coherence space) whose set of states is a concrete domain (coherence space) \( D(M) \), then \( !D(M) \) is a domain (coherence space) such that its objects are sets of sets of states. So, the actual state in \( !D(M) \) is not a single state but a set of states whose maximal states can be seen as a states of different concurrent processors.

We define now the notion of accessibility in \( !D(M) \) and the precedence (cover) relation.

3.1 Definition

Let \( M \) be a cds. If \( X \in !D(M) \), then we say that a cell \( c \) is accessible from \( X \) iff exists a state \( x \in \text{maximal}(X) \) such that \( c \) is accessible from \( x \). ☐

3.2 Definition

Let \( M \) be a cds. If \( X, Y \in !D(M) \), then we say that \( X \) precedes (cover) \( Y \) excepts \( c_1, \ldots, c_n \), we write \( Y < c_1, \ldots, c_n X \), iff for all cell \( c_i \), with \( 1 \leq i \leq n \), exists \( x \in \text{maximal}(X) \) and \( y \in \text{maximal}(Y) \) such that \( c_i \in A(x) \), \( c_i \in F(y) \) and \( X < Y (X < Y) \). ☐

Now we have the elements to define the cds \( !M \rightarrow !M' \) and the linear algorithms.

3.3 Definition

If \( M, M' \) are two cds, the cds \( !M \rightarrow !M' \) is defined by:

1. if \( x \) is finite state of \( M \), \( c_1', \ldots, c_k' \) are cells of \( M' \), then the cells of \( !M \rightarrow !M' \) are of type \( X(c_1', \ldots, c_k') \), where \( X \in \mathcal{G}(x) \) (or equivalently, \( X \subseteq lx) \);

2. the values and the events are of two types:
a) type "valof": if \( c_1, \ldots, c_n \) are cells of \( M, X, \ldots, X_n \subseteq \text{maximal}(X) \) are sets of states, then \( \text{valof} \{ c_1^m, \ldots, c_n^m \} \) is a value of \( !M \rightarrow_o M' \), and \( (X(c_1', \ldots, c_k'), \text{valof} \{ c_1^m, \ldots, c_n^m \}) \) is an event of \( !M \rightarrow_o M' \) iff \( c_1, \ldots, c_n \) are accessible from \( X_1, \ldots, X_n \) respectively;

b) type "output": if \( v_1', \ldots, v_k' \) are values of \( M' \), then \( \text{output} \{ v_1', \ldots, v_k' \} \) is a value of \( !M \rightarrow_o M' \), and \((X(c_1', \ldots, c_k'), \text{output} \{ v_1', \ldots, v_k' \}) \) is an event of \( !M \rightarrow_o M' \) iff \( (c_1', v_1'), \ldots, (c_k', v_k') \) are events of \( M' \);

3. the enabling relations are of two types:

a) \((Y(c_1', \ldots, c_k'), \text{valof} \{ c_1^m, \ldots, c_n^m \}) \vdash X(c_1', \ldots, c_k') \) iff \( Y \prec c_1, \ldots, c_n X \) and \( X \) is finite (type "valof");

b) \((X^1(c_1^1', \ldots, c_k^1'), \text{output} \{ v_1^{1'}, \ldots, v_k^{1'} \}), \ldots, (X^h(c_1^h', \ldots, c_k^h'), \text{output} \{ v_1^{h'}, \ldots, v_k^{h'} \}) \vdash X(c_1', \ldots, c_k') \)

iff \( X = \cup \{ X^1 \mid 1 \leq h \} \), \( X \) is finite and \((c_1^1', v_1^{1'}), \ldots, (c_k^1', v_k^{1'}), (c_1'^+', v_1'^+), \ldots, (c_k'^+', v_k'^+) \vdash c_1', \) or \( \emptyset \vdash c_1', \)

with \( 1 \leq i \leq k \) (type "output").

3.4 Definition

A state of the cds \( !M \rightarrow_o M' \) is called linear algorithm.

Having the notion of algorithm, we need the definition of application of the algorithm:

3.5 Definition

If \( A \) is a state of \( !M \rightarrow_o M' \) and \( X \) is an object of \( !D(M) \), then

\[
AX = \{ (c', v') \mid \exists Y \subseteq X \text{ such that } c' \in (c_1', \ldots, c_k') \land v' \in (v_1', \ldots, v_k') \land (Y(c_1', \ldots, c_k'), \text{output} \{ v_1', \ldots, v_k' \}) \in A \}
\]

is the application of \( A \) to \( X \). The function \( X \mapsto AX \) is called the input-output function computed by \( A \).

The operations fork and merge of concurrent languages can be simulated by the "valof" command as we'll see in the following definitions.
3.6 Definition

Let \( !M \rightarrow M' \) be a cds, \( X \) an object of \( !D(M) \) and \( x_1, \ldots, x_n \in \text{maximal}(X) \), then \( \text{merge} \{ x_1, \ldots, x_n \} = \text{valof}(c_1^{m_1}, \ldots, c_n^{m_n}) \), with \( Y_1, \ldots, Y_t \subseteq \{ x_1, \ldots, x_n \} \), is a value of \( !M \rightarrow M' \) and \( (X \{ c_1', \ldots, c_t' \}, \text{merge} \{ x_1, \ldots, x_n \}) \) is an event of \( !M \rightarrow M' \), iff

1. not exists the supremum of \( X \) in \( X \) \((n>1)\),
2. \( x_1, \ldots, x_n \) are consistent,
3. \( m_1 = \ldots = m_t \),
4. \( c_i \in \bigcup_{j=1}^{n} x_j - \bigcup_{y \in Y_t} y \). \( \blacksquare \)

The first condition is needed because if exists the supremum of \( X \) in \( X \) the operation \( \text{merge} \) doesn’t make sense because \( \text{maximal}(X) = \{ x \} \) for some \( x \), and \( \text{merge}(x) = x \). The condition of consistency is for avoid the cases in which two different maximal states have a cell \( c \) with two different values, for example \( \text{merge}((c_1, 0), \ldots, (c_1, 1), \ldots) = ((c_1, 0), (c_1, 1), \ldots) \) that is a inconsistent state. The third condition becomes obvious from the intuitive \( \text{merge} \) definition, i.e., “merge” some maximal states with one processor.

The last condition is to guarantee that the values of the cells not filled in all the involved states in the \( \text{merge} \) be computed.

3.7 Definition

Let \( !M \rightarrow M' \) be a cds, \( X \) an object of \( !D(M) \) and \( y \in \text{maximal}(X) \). Then \( \text{fork}_y \{ c_1^{m_1}, \ldots, c_n^{m_n} \} = \text{valof}(c_1^{m_1}, \ldots, c_n^{m_n}) \) is a value of \( !M \rightarrow M' \) and \( (X \{ c_1', \ldots, c_t' \}, \text{fork}_y \{ c_1^{m_1}, \ldots, c_n^{m_n} \}) \) is an event of \( !M \rightarrow M' \) iff \( c_1, \ldots, c_n \) not belong to \( y \). \( \blacksquare \)

If a cell \( c \) has a value in the state \( y \), then it must remain with the same value in every state that include \( y \), this is the reason of the condition that the cells to be computed not belong to \( y \).
4 How to obtain a linear algorithm for a stable function

We introduce now a constructive way for obtain the linear algorithm (a state of the cds \(\mathcal{M} \rightarrow \mathcal{M}'\)) from a stable function \(f: \mathcal{M} \rightarrow \mathcal{M}'\). Let’s introduce first the notion of compatible operation.

4.1 Definition

An operation (value) \(\nu\) of a cds \(\mathcal{M} \rightarrow \mathcal{M}'\), is compatible with a set of states \(X\) of \(\Delta(D(M))\), iff

1. \(\nu = valof\{c_{1}^{m_{1}}, ..., c_{l}^{m_{l}}\}\) and the execution of \(\nu\) on the set of states \(X\) generate a set of states \(X'\) that belong to \(\Delta(D(M))\);

2. \(\nu = output\{v_{1}', ..., v_{k}'\}\) and exists \(x \in \text{maximal}(X)\) such that the subjacent function \(f(x)\) be defined.

Let \(M \in \mathcal{M}\) be two cds (webs of coherence spaces) and \(f: \mathcal{M} \rightarrow \mathcal{M}'\) a stable function. We can obtain a cds \(\mathcal{M} \rightarrow \mathcal{M}'\) in the following way:

1. Obtain \(\Delta(D(M))\) from \(D(M)\)

2. Starting with the object \(\{\}\) of \(\Delta(D(M))\) write all the events \((\{\} \{c_{1}', ..., c_{k}'\}, valof\ C)\), where \(C\) is the set of cells to be filled. For the atom \(\emptyset\), write \((\{\} \{c_{1}', ..., c_{k}'\}, start)\), where \(start\) is the virtual operation that initiate the system

3. For all the operations of the events of the step 2., generate all the variable events of the form \((A \{c_{1}', ..., c_{k}'\}, valof\ c_{1}^{m_{1}}, ..., c_{n}^{m_{n}})\) (where \(A \in B_{1}, ..., B_{n}\) are variable).\(^3\)

4. For all \(X \in \Delta(D(M))\), instanciate the events generated in step 3., instanciate \(A\) with \(X\), and \(B_{1}, ..., B_{n}\) with subsets \(X_{1}, ..., X_{n} \subseteq \text{maximal}(X)\), iff the operator is compatible with \(X\).

5. For all \(X\) such that exist \(\text{sup}(X)\) in \(X\) and \(f(\text{sup}(X)) \neq \emptyset\) (i.e. \(\text{sup}(X)\) has a defined value), write the event \((X\{c_{1}', ..., c_{k}'\}, output\{v_{1}', ..., v_{k}'\})\), where \(v_{1}', ..., v_{k}' \in \mathcal{M}'\).

\(^3\)The event that has \(start\) as value is not included because this operation it’s only possible when the object is \(\{\}\).
Obviously, the stable functions considered must be computable as establish Asperti in [ASP 90], otherwise we would be giving a way of compute a non computable function, which is an absurd.

5 Trace of a linear algorithm

We define in this section the trace of the linear algorithm, that characterize the algorithm.

5.1 Definition

Let \( \mathcal{M} \rightarrow \mathcal{M}' \) be a cds and \( A \in \mathcal{M} \rightarrow \mathcal{M}' \) a linear algorithm. Then we define the trace of the linear algorithm, we write \( tr_{alg} \), by

\[
tr_{alg}(A) = \{ (X, \{(c_i', v_i'),..., (c_k', v_k')\}) \mid 1 \leq i \leq k \ (X[c_i',..., c_k'], output [v_i',..., v_k']) \in A \}. \]

6 Strategies of computation

We define now the notion of strategies of computation but before we need some auxiliary notation and definitions. We call an event of a cds \( \mathcal{M} \rightarrow \mathcal{M}' \) a (partial) computation.

Notation. Eventually we call \( C \) the set of output cells \( \{c_1',..., c_k'\} \) when this not make confusion with the set of all the cell of a cds.

6.1 Definition

Let \( \mathcal{M} \rightarrow \mathcal{M}' \) be a cds. Let \( A \) be a linear algorithm of \( \mathcal{M} \rightarrow \mathcal{M}' \) and \( (X_iC_i', v_i) \) and \( (X'C', v') \) be two computations of \( A \). Then we say that \( (X_iC_i', v_i) \) precedes computationally \( (X'C', v') \), we write \( (X_iC_i', v_i) \leq_{comp} (X'C', v') \), iff exists a sequence of events of \( A \) \( (X_iC_i', v_i), ..., (X_nC_n', v_n) = (X'C', v') \) such that \( (X_iC_i', v_i) \mid X_{i+1}C_{i+1}', \forall i, 1 \leq i \leq n \). If \( n = 1 \) then we sat that \( (X_iC_i', v_i) \) cover computationally \( (X'C', v') \) and is denoted by \( (X_iC_i', v_i) \prec_{comp} (X'C', v') \). \( \blacksquare \)
6.2 Definition

Let \( !(M \rightarrow M') = (C, V, E, \rightarrow) \) be a cds, and \( A \subseteq E \) be a linear algorithm. Then we say that a subset \( a \subseteq A \) is a **strategy of computation** (or simply **strategy**) iff given a set of states \( X_0 \) and an operation \( v_0 \),

1. \( \exists !(X_0C_0', v_0) \in a \) such that \( \neg \exists (X'C', v) \in a, (XC', v) \rightarrow (X_0C_0', v_0) \)

2. \( \forall (X'C', v') \in a \in (X'C', v'') \in a \), then \( v' = v'' \)

3. \( \forall (X'C', v') \in a, (X_0C_0', v_0) \) precedes computationally \( (X'C', v') \). \[ \square \]

The above definition allow us to give semantic \( c_\circ \) segments of programs, which is useful for obtaining the semantic of the main program as a composition \( c_\circ \) the semantics of the segments.

6.3 Definition

Let \( !(M \rightarrow M') = (C, V, E, \rightarrow) \) be a cds, \( A \subseteq E \) be a linear algorithm and \( a \) be a strategy of computation of \( A \).

We say that \( a \) is a **total strategy of computation** iff

1. \( (\varnothing C_0', v') \in a; \)

2. \( \forall (X'C', v') \in a, \exists (X'C'', v) \in A \) such that \( (X'C', v') \rightarrow X'C'', (XC'', v) \in a. \)

In other case, we say that the strategy is **partial**. \[ \square \]

There are some strategies of computation that in: each computation fill only one cell, meanwhile others that apply the operation to more than one cell. This is leads to the definition of sequential and parallel strategies.

6.4 Definition

Let \( !(M \rightarrow M') = (C, V, E, \rightarrow) \) be a cds, \( A \subseteq E \) be a linear algorithm and \( a \) be a strategy of computation of \( A \).

We say that \( a \) is a **sequential strategy of computation** iff

1. \( \forall (X[c_1',..., c_k'], output \{v_1',..., v_k'\}) \in a, k = 1 \) and

2. \( \forall (X[c_1',..., c_k']) \), \( valof \{c_1^{m_1},...,c_k^{m_k}\}) \in a, n = 1. \)
Otherwise we say that the strategy is parallel. ■

Intuitively, the strategies of computation are the programs that compute the subjacent function. The following diagram shows how the linear algorithms, strategies of computations, linear functions, stable functions and their traces are related.

\[ \begin{align*}
\text{(stable function)} & \quad \text{tr}(f) \\
\text{lin}(f) & \quad \text{trlin}(f) \\
A & \quad \text{a \subseteq A (strategy)} \\
\text{tr}_{\text{alg}}(A) & \quad \text{many-to-one} \\
\text{one-to-one} &
\end{align*} \]

7 Composition of strategies

Suppose that we have a program \( P \) that call two subprograms \( P_1 \) and \( P_2 \). \( P \) is represented in the following diagram:

\[
\begin{array}{c}
P \\
/ \\
P_1 \\
/ \\
P_2
\end{array}
\]

\( P \) can be considered as the composition of five "segments": \( S_1, P_1, S_2, P_2, S_3 \). Clearly, with the given strategy of computation definition, we have that for each segment there exists a strategy. Intuitively, the total strategy for \( P \) would be the composition of the strategies of the five segments. In the context of \( P \), each of the segment strategies is partial. If \( P_1 \) and \( P_2 \) are independent of the environment then their strategies will
be total. But, what is the relation between, for example, $P_1$ and $S_2$? Clearly $P_1$ must be computed before $S_2$ and the final state of $P_1$ must be the initial state of $S_2$. Formally:

### 7.1 Definition

Let $\mathcal{M} = (C, V, E, \vdash)$ be a cds, $A \subseteq E$ be a linear algorithm and $a_1$, $a_2$ be strategies of computation of $A$. We define the composition of $a_1$ and $a_2$, we write $a_1 \circ a_2$, iff

1. $a_1 \cup a_2$ is a strategy of computation, and
2. there no exists a computation of $a_2$ that precedes computationally a computation of $a_1$.

Thus, a compositional operational semantics for program segments would be a function mapping program segments to computation strategies, which preserves program composition.

### 8 Conclusion

The definition of the linear algorithms can be considered as an extension of the sequential algorithms of Curien ([CUR 86]). One advantage of the former is that its includes not only the sequential strategies but the parallel ones too. We had proved (see [SCH 95]) that for all sequential algorithm there exists a linear algorithm that contains one sequential strategy equivalent to the Curien's algorithm. We have the intuition that this relation is valid for the Brookes & Geva's parallel algorithms too (see [BRO 90]). Those parallel strategies would take advantage of the availability of resources in parallel machines. In [SCH 95] we present the sketch of an abstract machine that computes linear algorithms, choosing strategies depending on the machine capacity.

The strategies of computation were defined to give semantics of programs segments. We define the composition of strategies which allows us to give a compositional semantics.

The linear algorithms can be used to give computation strategies for all kind of functions, not only the stables ones. This is trivial, we only need to consider as a subjacent function $f$ any function, without
Clearly the effect of this is that the application of the operator to \( f \) doesn't obtain a linear function. The reader can see an example (the sort function) in [SCH 95].

Clearly, also, we don't claim that all computation strategies are effective, in the sense of the classical theory of computability.

9 References


