CxSOM

Hervé Frezza-Buet
Herve.Frezza-Buet@centralesupelec.fr

Contents

1 Introduction

The `CxSOM` software suite enables to model consensus based multi-SOMs, explored in the BISCUIT team of teh Loria lab.

The current documentation is at work, and mainly technical stuff are reported here. See the README at the package root for an introduction, as well as examples.

2 The model

The computation enabled by `CxSOM` consists of successively updating data, keeping trace of the data values in history files.

2.1 Data instances (DI)

The elementary piece of data handled in `CxSOM` is called a *data instance* (*DI*). Each of the computed DIs ends as a record in a file, once it has been computed.

DIs are denoted by a triplet $[T,\ X,\ t]$, that reads as "the instance of variable X hosted by the timeline T at the time instant t of that specific timeline". Only elementary $[T,\ X,\ t]$ DIs matter, the concepts of *variable*, *timeline* and *time instant* only serve for describing the computation.

Each DIs is a value that is stored in a file once definitively computed. A value has a type, that can be:

- `Scalar`: A floating point value, usually in $[0, 1]$.
- `Pos1D`: A value in $[0, 1]$ corresponding to a position in a 1D map.
- `Pos2D`: A value in $[0, 1]^2$ corresponding to a position in a 2D map.
- `Array=`n: A value in $\mathbb{R}^n$, usually $[0,\ 1]^n$, $n \in \mathbb{N}^\star$.
- `Map1D<`$\mathcal{X}$`>=`n: A 1D map of type $\mathcal{X}$, i.e. a value in $\mathcal{X}^n$, $n \in \mathbb{N}^\star$.
- `Map2D<`$\mathcal{X}$`>=`n: A 2D squared map of type $\mathcal{X}$, i.e. a value in $(\mathcal{X}^n)^n$, $n \in \mathbb{N}^\star$.

where type $\mathcal{X} \in \{$`Scalar`, `Pos1D`, `Pos2D`, `Array=`$k\}$.

In `CxSOM`, the type is associated to a variable, i.e. all the $[T,\ X,\ \bullet]$ are DIs with the same type.

Every DI is also doted with a status variable $\langle [T,\ X,\ t] \rangle \in \{$`busy`, `ready`$\}$:

- `ready`: The computation of the value of the DI is definitively done. The DI value will not change anymore.
- `busy`: The definitive value of the DI is still to be determined.

More generally, in the following, $\langle x \rangle$ reads as "the status of x".

2.2 Updates

An *update* of some DI is the (re)computation by the `CxSOM` computer of the value of that DI. There are at most two updates for a DI, one for the first update (initialization) that is optional, and one, mandatory, for other updates.

2.2.1 An update is a function taking arguments and computing a result

An update u for a DI $[T,\ X,\ t]$ is made of:

- The decription of some computation, i.e. the function called to realize an update.
- The result res_u, i.e. $\text{res}_u \stackrel{\text{def}}{=} [T,\ X,\ t]$ itself, that is computed by the function of the update.
- Other DIs, thats serve as arguments to the function (their value is read when the function is called). Among arguments, a distinction is made between *in-arguments* and *out-arguments*:
  - The in-arguments in_u of the update u such as $\text{res}_u = [T,\ X,\ t]$ is the set of DIs used as arguments of u of the form $[T,\ \bullet,\ t]$. All the DIs handled in the simulation of the form $[T,\ \bullet,\ t]$ define a *time step* $\mathcal{S}_T^t$ of the simulation, so the in-arguments are the DIs used for the computation of an update which belong to the same timestep as the result.
  - The out-arguments out_u of the update u are the other DIs used as arguments for the computation of the update.

2.2.2 Datation and relaxation

Before it has a definitive value (i.e. before being in the `ready` status), the value of a DI may change several times. Indeed, all DIs in a timestep $\mathcal{S}_T^t = [T,\ \bullet,\ t]$ are updated until all of them get `ready`, and during that process, the values of the DIs may be recomputed several times. In order

to handle the dependancies of the DIs within a timestep, we need to timestamp the values of the DIs, in order to determine those of them that have to be updated. Timestamp play the role of file dates in makefile, when the date of a target is compared to the date of its dependancies. Here, in order to know if an update u needs to be performed, we have to consider two things:

- Do we have $\forall [T,\ X,\ t] \in \text{out}_u,\ \langle [T,\ X,\ t] \rangle = \texttt{ready}$? If not, the update cannot be done, it is considered as *impossible*.
- Have the in-arguments been updated since the last computation of the result? If the answer is yes, the result needs to be recomputed.

To compute the second conditon, every DI comes with a *datation* denoted by $\text{d}_{[T,\ X,\ t]} \in \mathbb{N}$.

An update then stores, for each of its in-arguments, the datation it had at the last computation of the result. When the simuator considers an update, it compares the current datation of the in-arguments to the ones stored by the update. If some are newer (or if the result has never been computed so far) the result is recomputed, and

$$\text{d}_{\text{res}_u} = 1 + \max_{[T,\ X,\ t] \in \text{in}_u} \text{d}_{[T,\ X,\ t]} \tag{1}$$

Considering an update for computation can thus lead to the following status of the update u, denoted by $\langle u \rangle$:

- $\langle u \rangle = \texttt{impossible}$: $\langle \text{res}_u \rangle = \texttt{busy}$, computation is not feasible yet, since some out-arguments are `busy`.
- $\langle u \rangle = \texttt{uptodate}$: $\langle \text{res}_u \rangle = \texttt{busy}$, nothing changed in the in-arguments input dates from last update, or the new value was not a significant modification. The datation d_{res_u} has not been modified.
- $\langle u \rangle = \texttt{updated}$: $\langle \text{res}_u \rangle = \texttt{busy}$, the computation has modified the value of res_u significantly.
- $\langle u \rangle = \texttt{done}$: $\langle \text{res}_u \rangle = \texttt{ready}$, the computation has modified the value res_u definitively.
- $\langle u \rangle = \texttt{none}$: Update status is not determined yet (used for initialization only).

The datation mechanism enables to control the update of all the DIs in a given timestep, i.e. all the DIs like $[T,\ \bullet,\ t]$ for the time instant t of the timeline T. When no more significant writes of the results can be done, the whole timestep is stable. So the relevance of the datation mechanism is to enable a *relaxation* of the DIs inside a timestep until stabilization of all of them is reached.