An Experimental Spatio-Temporal Model Checker*

Vincenzo Ciancia[1], Gianluca Grilletti[2], Diego Latella[1], Michele Loreti[3,4], and Mieke Massink[1]

[1] Istituto di Scienza e Tecnologie dell'Informazione 'A. Faedo', CNR, Pisa, Italy
[2] Scuola Normale Superiore, Pisa, Italy
[3] Università di Firenze, Italy
[4] IMT Alti Studi, Lucca, Italy

Abstract. In this work we present a spatial extension of the global model checking algorithm of the temporal logic CTL. This classical verification framework is augmented with ideas coming from the tradition of topological spatial logics. More precisely, we add to CTL the operators of the *Spatial Logic of Closure Spaces*, including the *surrounded* operator, with its intended meaning of a point being surrounded by entities satisfying a specific property. The interplay of space and time permits one to define complex spatio-temporal properties. The model checking algorithm that we propose features no particular efficiency optimisations, as it is meant to be a reference specification of a family of more efficient algorithms that are planned for future work. Its complexity depends on the product of temporal states and points of the space. Nevertheless, a prototype model checker has been implemented, made available, and used for experimentation of the application of spatio-temporal verification in the field of *collective adaptive systems.*

1 Introduction

A *collective system* consists of a large set of interacting individuals. The temporal evolution of the system is not only determined by the decisions taken by the individuals at the *local* level, but also by their interactions, that are observable at the *global* level. By their own nature, such systems feature a "spatial" distribution of the individuals (e.g., locations in physical space, or nodes of some digital or social network), affecting interaction possibilities and patterns. Verification of collective systems and of their adaptation mechanisms requires one to take such spatial constraints into account.

In this work, we provide a preliminary study on the feasibility of model checking as a fully automated analysis of spatio-temporal models. Our work is grounded on the so-called *snapshot models* (see [12] for an introduction). Spatial information is encoded by some topological structure, in the tradition of *topological spatial logics* [1], whereas temporal information is described by a *Kripke*

* Research partially funded by EU project QUANTICOL (nr. 600708) and IT MIUR project CINA

frame. The valuation of atomic propositions is a function of temporal states, and spatial locations. We employ *Čech closure spaces* for the spatial part of the modelling, following the research line initiated in [6] with the definition of the *Spatial Logic of Closure Spaces* (SLCS). Čech closure spaces are a generalisation of topological spaces also encompassing directed graphs.

Starting from a spatial and a temporal formalism, *spatio-temporal* logics may be defined, by introducing some mutually recursive nesting of spatial and temporal operators. Several combinations can be obtained, depending on the chosen spatial and temporal fragments, and the permitted forms of nesting of the two. A great deal of possibilities are explored in [12], for spatial logics based on topological spaces. We investigate one such structure, in the setting of closure spaces, namely the combination of the temporal logic *Computation Tree Logic* (CTL) and of SLCS, resulting in the *Spatio-Temporal Logic of Closure Spaces* (STLCS). STLCS permits arbitrary mutual nesting of its spatial and temporal fragments. As a proof of concept, we define a simple model checking algorithm, which is a variant of the classical CTL labelling algorithm [8,2], augmented with the algorithm in [6] for the spatial fragment. The algorithm, which operates on finite spaces, has been implemented as a prototype [10], that we discuss in the current paper. The same algorithm is also currently implemented in the tool `topochecker` [4], which is meant to be further developed and maintained, and was used in [7] in order to check spatio-temporal properties of bike sharing systems.

Related work. The literature on topological spatial logics is rich (see [1]). However, model checking is typically not taken into account; this is discussed in detail in [6]. Relevant exceptions are the recent works [11], where statistical model checking is used on a (linear) spatio-temporal logics of signals, and [13], also developing a model checker for a linear spatio-temporal logic of signals, augmented with some metric information, and inheriting the approach of closure spaces from [6]. Our work diverts from these research lines in that the underlying temporal model is branching, thus permitting thorough evaluation of system properties in the presence of information about nondeterministic choice in the model. In computer science, the term *spatial logics* has also been used for logics that predicate about the internal structure of processes in process calculi. A model checker for such kind of logics was developed in [3]. Indeed, the theory and tool we present are linked to topological spatial logics rather than the area of process calculi, thus the developed algorithms are very different in nature.

2 Motivating Example: Adaptive Smart Transport Network

This work is part of a larger research effort aimed at formal verification of spatio-temporal requirements of *collective adaptive systems*, in the scope of the EU FP7 QUANTICOL project[5]. In order to motivate the proposed tool in the theory of

[5] See the web site `http://www.quanticol.eu`

verification of adaptive systems, we briefly report on a recent case study, detailed in [5], where the STLCS model checker has been used in the context of adaptive systems, and in particular of *smart transport networks*. The context is the bus network of a city. The model checker is primarily used to identify occurrences of *clumping* of buses, that is, buses of the same line that are "too close in space-time" to each other, resulting in several buses of the same line passing by the same stops within a short amount of time, and longer intervals without any buses at certain stops. More precisely, a bus is part of a *clump* if *it is close to a point where another bus of the same line will be very soon*. This statement is inherently spatio-temporal, and classical temporal logics do not have the ability to directly express it. It turns out that there is some ambiguity in the formalisation of this sentence, resulting in different possible STLCS formulas characterising it. Once established these formulas, the bus coordination system is equipped with an adaptation layer, enabling buses to wait for some time at a stop, in order to avoid the emergence of clumps at the expenses of some additional delay on the line. The underlying hypothesis is that clumping happens when some buses are forced to delay (e.g. because of traffic conditions) but the system evolves immediately afterwards, in such a way that subsequent buses of the same line do not delay. The STLCS model checker is used to define an analysis methodology that estimates the impact of adaptation, before deployment, starting from existing traces (logs) of the system. Each trace, in the form of a series of GPS coordinates for each bus, is considered as a deterministic system. For traces featuring clumping (checked using the model checker), the expected non-deterministic behaviour of the system under the effect of the adaptation layer is then computed as a spatio-temporal model, by augmenting the existing trace with the possible "wait" steps of each bus. The counterexample-generation capabilities of the model checker are finally used on such Kripke frame to analyse the impact of the adaptation, by identifying new traces containing wait instructions that correct the problem. By doing this, one is able to check if, and under what conditions, the adaptation strategy succeeds in mitigating or eliminating the clumping problem, and confirm or disprove (depending on the actual situation) the hypothesis underlying the choice of the adaptation strategy. For more details on the specific case study, we refer the reader to [5]; in the remainder of the paper, we shall focus on the formal definition of the STLCS logic, and its model checking algorithm, as both were not presented in [5].

3 Closure Spaces

In this work, we use *closure spaces* to define basic concepts of *space*. Below, we recall several definitions, most of which are explained in [9]. See also [6] for a thorough description of SLCS, the spatial logic of closure spaces, and its model-checking algorithm. A closure space is a set equipped with a *closure operator* obeying to certain laws. In the finite case, closure spaces are graphs, but also (infinite) topological spaces are an instance of the more general constructions.

Definition 1. *A* closure space *is a pair* $(X, \mathcal{C})$ *where* X *is a set, and the* closure operator $\mathcal{C} : 2^X \to 2^X$ *assigns to each subset of* X *its* closure*, obeying to the following laws, for all* $A, B \subseteq X$*:*

1. $\mathcal{C}(\emptyset) = \emptyset$*;*
2. $A \subseteq \mathcal{C}(A)$*;*
3. $\mathcal{C}(A \cup B) = \mathcal{C}(A) \cup \mathcal{C}(B)$*.*

The notion of *interior*, dual to closure, is defined as $\mathcal{I}(A) = X \setminus \mathcal{C}(X \setminus A)$. Closure spaces are a generalisation of *topological spaces.* The axioms defining a closure space are also part of the definition of a *Kuratowski closure space*, which is one of the possible alternative definitions of a topological space. More precisely, a topological space is a closure space where the axiom $\mathcal{C}(\mathcal{C}(A)) = \mathcal{C}(A)$ (idempotency) holds. We refer the reader to, e.g., [9] for more information.

Various notions of *boundary* can be defined. The *closure boundary* (often called *frontier*) is used for the *surrounded* operator in STLCS.

Definition 2. *In a closure space* $(X, \mathcal{C})$*, the* boundary *of* $A \subseteq X$ *is defined as* $\mathcal{B}(A) = \mathcal{C}(A) \setminus \mathcal{I}(A)$*. The* interior boundary *is* $\mathcal{B}^-(A) = A \setminus \mathcal{I}(A)$*, and the* closure boundary *is* $\mathcal{B}^+(A) = \mathcal{C}(A) \setminus A$*.*

A closure space may be derived starting from a *binary relation*, that is, a *graph.* In particular all finite spaces are in this form. This is easily seen by the equivalent characterization of *quasi-discrete* closure spaces.

Definition 3. *Consider a set* X *and a relation* $R \subseteq X \times X$*. A closure operator is obtained from* R *as* $\mathcal{C}_R(A) = A \cup \{x \in X \mid \exists a \in A.(a, x) \in R\}$*.*

Closure spaces derived from a relation can be characterised as *quasi-discrete* spaces (see also Lemma 9 of [9] and the subsequent statements).

Definition 4. *A closure space is* quasi-discrete *if and only if one of the following equivalent conditions holds: i) each* $x \in X$ *has a* minimal neighbourhood[6] N_x*; ii) for each* $A \subseteq X$*,* $\mathcal{C}(A) = \bigcup_{a \in A} \mathcal{C}(\{a\})$*.*

Proposition 1. *A closure space* $(X, \mathcal{C})$ *is quasi-discrete if and only if there is a relation* $R \subseteq X \times X$ *such that* $\mathcal{C} = \mathcal{C}_R$*.*

Summing up, a closure space enjoys minimal neighbourhoods, and the closure of A is determined by the closure of the singletons composing A, if and only if the space is derived from a relation using Definition 3.

[6] A *minimal neighbourhood* of x is a set A that is a *neighbourhood* of x, namely, $x \in \mathcal{I}(A)$, and is included in all other neighbourhoods of x.

4 The Spatio-Temporal Logic of Closure Spaces

We define a logic interpreted on a variant of Kripke models, where valuations are interpreted at points of a closure space. Fix a set P of proposition letters.

Definition 5. *STLCS formulas are defined by the following grammar, where p ranges over P:*

$$
\begin{array}{rll}
\Phi ::= & \top & [\textsc{True}] \\
\mid & p & [\textsc{Atomic predicate}] \\
\mid & \neg \Phi & [\textsc{Not}] \\
\mid & \Phi \vee \Phi & [\textsc{Or}] \\
\mid & \mathcal{N} \Phi & [\textsc{Close}] \\
\mid & \Phi \mathcal{S} \Phi & [\textsc{Surrounded}] \\
\mid & \mathtt{A}\, \varphi & [\textsc{All Futures}] \\
\mid & \mathtt{E}\, \varphi & [\textsc{Some Future}]
\end{array}
$$

$$
\begin{array}{rll}
\varphi ::= & \mathcal{X} \Phi & [\textsc{Next}] \\
\mid & \Phi \,\mathcal{U} \Phi & [\textsc{Until}]
\end{array}
$$

The logic STLCS features the CTL path quantifiers `A` ("for all paths"), and `E` ("there exists a path"). As in CTL, such quantifiers must necessarily be followed by one of the path-specific temporal operators, such as[7] $\mathcal{X}\Phi$ ("next"), $\mathtt{F}\Phi$ ("eventually"), $\mathtt{G}\Phi$ ("globally"), $\Phi_1 \mathcal{U} \Phi_2$ ("until"), but unlike CTL, in this case Φ, Φ_1 and Φ_2 are STLCS formulas that may make use of spatial operators. Further operators of the logic are the boolean connectives, and the spatial operators $\mathcal{N}\Phi$, denoting closeness to points satisfying Φ, and $\Phi_1 \mathcal{S} \Phi_2$, denoting that a specific point satisfying Φ_1 is surrounded, via points satisfying Φ_1, by points satisfying Φ_2. The mutual nesting of such operators permits one to express spatial properties in which the involved points are constrained to certain temporal behaviours. Let us proceed with a few examples. Consider the STLCS formula `EG (green S blue)`. This formula is satisfied in a point x in the graph, associated to the initial state s_0, if there exists a (possible) evolution of the system, starting from s_0, in which point x, in every state in the path, satisfies *green* and is surrounded by blue. A further, nested, example is the STLCS formula `EF (green S (AX blue))`. This formula is satisfied by a point x in the graph, in the initial state s_0, if there is a (possible) evolution of the system, starting from s_0, in which point x is eventually green and surrounded by points y that, for every possible evolution of the system from then on, will be blue in the next time step.

A model $\mathcal{M}$ is composed of a Kripke structure $(S, \mathcal{T})$, where S is a non-empty set of *states*, and $\mathcal{T}$ is a non-empty *accessibility relation* on states, and a closure space $(X, \mathcal{C})$, where X is a set of points and $\mathcal{C}$ the closure operator. Every state s has an associated valuation $\mathcal{V}_s$, making $((X, \mathcal{C}), \mathcal{V}_s)$ a *closure model* according to

[7] Some operators may be derived from others; for this reason, e.g., in Definition 5, and Section 5, we use a minimal set of connectives. As usual in logics, there are several different choices for such a set.

Definition 6 of [6]. Equivalently, valuations have type $S \times X \to 2^P$, where P is the set of atomic propositions, thus, the valuation of atomic propositions depends both on states and points of the space. Intuitively, there is a set of possible worlds, i.e. the states in S, and a spatial structure represented by a closure space. In each possible world there is a different valuation of atomic propositions, inducing a different "snapshot" of the spatial situation which "evolves" over time. In this paper we assume that the spatial structure $(X, \mathcal{C})$ does not change over time. Other options are indeed possible. For instance, when space depends on S, one may consider an S-indexed family $(X_s, C_s)_{s \in S}$ of closure spaces.

Definition 6. *A model is a structure $\mathcal{M} = ((X, \mathcal{C}), (S, \mathcal{T}), \mathcal{V}_{s \in S})$ where $(X, \mathcal{C})$ is a closure space, $(S, \mathcal{T})$ is a Kripke frame, and $\mathcal{V}$ is a family of* valuations*, indexed by states. For each $s \in S$, we have $\mathcal{V}_s : P \to \mathcal{P}(X)$.*

A path in the Kripke structure is a sequence of *spatial models* (in the sense of [6]) indexed by instants of time; see Fig. 1, where space is a two-dimensional structure, and valuations at each state are depicted by different colours.

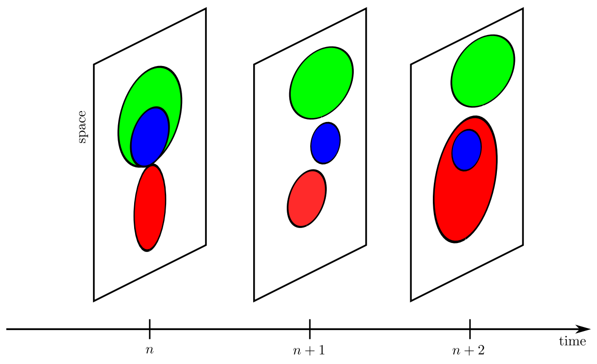

Fig. 1: In spatio-temporal logics, a temporal path represents a sequence of *snapshots* induced by the time-dependent valuations of the atomic propositions.

Definition 7. *Given Kripke frame $\mathcal{K} = (S, \mathcal{T})$, a* path *$\sigma$ is a function from $\mathbb{N}$ to S such that for all $n \in \mathbb{N}$ we have $(\sigma(i), \sigma(i+1)) \in \mathcal{T}$. Call $\mathcal{P}_s$ the set of infinite paths in $\mathcal{K}$* rooted *at s, that is, the set of paths σ with $\sigma(0) = s$.*

The evaluation contexts are of the form $\mathcal{M}, x, s \models \Phi$, where Φ is a STLCS formula, s is a state of a Kripke structure, and x is a point in space X.

Definition 8. *Satisfaction is defined in a model $\mathcal{M} = ((X, \mathcal{C}), (S, \mathcal{T}), \mathcal{V}_{s \in S})$ at point $x \in X$ and state $s \in S$ as follows:*

$$\begin{array}{lcl}
\mathcal{M}, x, s \models \top & & \\
\mathcal{M}, x, s \models p & \iff & x \in \mathcal{V}_s(p) \\
\mathcal{M}, x, s \models \neg \Phi & \iff & \mathcal{M}, x, s \not\models \Phi \\
\mathcal{M}, x, s \models \Phi \vee \Psi & \iff & \mathcal{M}, x, s \models \Phi \text{ or } \mathcal{M}, x, s \models \Psi \\
\mathcal{M}, x, s \models \mathcal{N} \Phi & \iff & x \in \mathcal{C}(\{y \in X | \mathcal{M}, y, s \models \Phi\}) \\
\mathcal{M}, x, s \models \Phi \mathcal{S} \Psi & \iff & \exists A \subseteq X . x \in A \wedge \forall y \in A . \mathcal{M}, y, s \models \Phi \wedge \\
 & & \quad \wedge \forall z \in \mathcal{B}^+(A) . \mathcal{M}, z, s \models \Psi \\
\mathcal{M}, x, s \models \mathtt{A}\, \varphi & \iff & \forall \sigma \in \mathcal{P}_s . \mathcal{M}, x, \sigma \models \varphi \\
\mathcal{M}, x, s \models \mathtt{E}\, \varphi & \iff & \exists \sigma \in \mathcal{P}_s . \mathcal{M}, x, \sigma \models \varphi \\
 & & \\
\mathcal{M}, x, \sigma \models \mathcal{X} \Phi & \iff & \mathcal{M}, x, \sigma(1) \models \Phi \\
\mathcal{M}, x, \sigma \models \Phi \mathcal{U} \Psi & \iff & \exists n . \mathcal{M}, x, \sigma(n) \models \Psi \text{ and } \forall n' \in [0, n) . \mathcal{M}, x, \sigma(n') \models \Phi
\end{array}$$

The syntax we provide is rather essential. Further operators can be derived from the basic ones; e.g., one can define conjunction and implication using negation and disjunction; spatial *interior* is defined as the dual of $\mathcal{N}$; several derived path operators are well-known for the temporal fragment, by the theory of CTL. We do not attempt to make an exhaustive list; for the classical temporal connectives, see e.g., [2]; for spatial operators, [6] provides some interesting examples. In Section 6 we show some simple spatial and spatio-temporal formulas. More complex formulas can be found in [5].

5 Model checking

In this section we describe the model checking algorithm, which is a variant of the well-known CTL labelling algorithm. For more information on CTL and its model checking techniques, see e.g., [2] or [8]. This algorithm operates in the case of *finite, quasi-discrete closure spaces*, represented as finite graphs. Assume the type `Set` implementing a finite set-like data structure[8], with elements of type `El` and operations `union`, `inter`, `diff`, `times`, `emptyset`, with the obvious types.

We represent a *finite directed graph* as the triple

$$(\mathtt{G} : \mathtt{Set}, \mathtt{Pred\_G} : \mathtt{El} \to \mathtt{Set}, \mathtt{Cl\_G} : \mathtt{Set} \to \mathtt{Set})$$

where the argument and result of the operators implementing closure `Cl_G`, and predecessor `Pred_G`, are constrained to belong to `G`. We describe a model by a pair of graphs $\mathcal{M} = (\mathcal{X}, \mathcal{T})$ where the spatial component is $\mathcal{X} = (\mathtt{X}, \mathtt{Pred\_X}, \mathtt{Cl\_X})$, and the temporal component (which can be thought of as a Kripke frame) is $\mathcal{T} = (\mathtt{T}, \mathtt{Pred\_T}, \mathtt{Cl\_T})$.

[8] We remark that the complexity of operations on such type affect the complexity of the algorithm; however, since the algorithm is global, the `Set` type may be implemented using an explicit lookup table, that is, an array of boolean values indexed by states, as usual in model checking, obtaining the complexity that we discussed.

Consider the finite set `S = X times T` of *points in space-time*; given a subset `A` $\subseteq$ `S`, and a state `t` $\in$ `T`, we let `space_sec(A,t)` be the subset of `X` containing the points x such that $(x,t) \in$ `A`; we define `time_sec` in a similar way. With `choose` we indicate the operation of choosing an element from a non-empty set (without making explicit how to pick it). For Φ an STLCS formula, and $\mathcal{M}$ a model, we let $[\![\Phi]\!]^{\mathcal{M}} = \{(x,t) \subseteq S \mid \mathcal{M}, x, t \models \Phi\}$.
Given a formula Φ and a model $\mathcal{M}$, the algorithm proceeds by induction on the structure of Φ; the output of the algorithm is the set $[\![\Phi]\!]^{\mathcal{M}}$. In the following, we present the relevant code portions addressing each case of the syntax; we omit the cases for the boolean connectives, and use a minimal set of connectives, apt to efficient verification, for the temporal part, namely $\mathtt{E}\mathcal{X}$, `AF`, $\mathtt{E}\ \mathcal{U}$. The cases for $\Phi = \mathtt{E}\mathcal{X}\Phi'$ and $\mathtt{E}(\Phi_1 \mathcal{U} \Phi_2)$ make use of the auxiliary function `pred_time`:

```
function pred_time(A)
  F := emptyset;
  foreach ((x,t) in A)
      U := Pred_T(t);
      F := F union ({x} times U);
  return F;
```

Case $\Phi = \mathcal{N}\Phi'$*:* The result is computed as the set $\bigcup_{(x,t)\in[\![\Phi']\!]^{\mathcal{M}}} \{(y,t) \mid y \in \mathcal{C}_X(x)\}$, which is correct in a quasi-discrete closure space $(X, \mathcal{C})$, as, for all sets A, we have $\mathcal{C}(A) = \bigcup_{x\in A} \mathcal{C}(\{x\})$.

```
let A = $[\![\Phi']\!]^{\mathcal{M}}$;
P := emptyset;
foreach ((x,t) in A)
    P := P union (Cl_X({x}) times {t});
return P;
```

Case $\Phi = \Phi_1 \mathcal{S} \Phi_2$*:* For every state t, we compute the spatial components of $[\![\Phi_1]\!]^{\mathcal{M}}$ and $[\![\Phi_2]\!]^{\mathcal{M}}$ at state `t` (called `R` and `Bs` in the pseudo-code). Then we apply the algorithm described in [6].

```
let A = $[\![\Phi_1]\!]^{\mathcal{M}}$;
let B = $[\![\Phi_2]\!]^{\mathcal{M}}$;
F := emptyset;
foreach (t in T)
    R := space_sec(A,t);
    Bs := space_sec(B,t);
    U := R union Bs;
    D := Cl_X(U) diff U;
    while (D != emptyset)
        s := choose(D);
        N := ( Cl_X({s}) inter R ) diff Bs;
        R := R diff N;
        D := ( D union N ) diff {s};
    F := F union (R times {t})
return F;
```

Case $\Phi = \mathtt{E}\mathcal{X}\Phi'$: The set of predecessors (in time) of the points in space-time belonging to the semantics of Φ' are computed and returned.

```
let A = $[\![\Phi']\!]^{\mathcal{M}}$;
return pred_time(A);
```

Case $\Phi = \mathtt{AF}\Phi'$: The case for `AF` is essentially the efficient algorithm for `EG` presented in [2], except that it is presented in "dual" form, using the fact that $[\![\mathtt{EG}\Phi']\!]^{\mathcal{M}} = [\![\neg\mathtt{AF}(\neg\Phi')]\!]^{\mathcal{M}}$. The algorithm is iterated for each point of the space. More precisely, for each $\mathtt{x} \in \mathtt{X}$, vector `count`, whose indices are states in `T`, is used to maintain the following invariant property along the `while` loop: *whenever* `count[t]` *is* 0*, we have* $\mathcal{M}, x, t \models \mathtt{AF}\Phi'$. In order to establish such invariant property, before the `while` loop, `count[t]` is initialised to 0 for each point in `F`, which is the set of points `t` such that there is some `x`, with $\mathcal{M}, \mathtt{x}, \mathtt{t} \models \Phi'$ (therefore, also $\mathcal{M}, \mathtt{x}, \mathtt{t} \models \mathtt{AF}\Phi'$ by definition). For each remaining state `t`, the value of `count[t]` is set to the number of its successors. Along the `while` loop, the set `U` is the set of states t that, at the previous iteration (or at initialisation), have been shown to satisfy $\mathcal{M}, \mathtt{x}, \mathtt{t} \models AF\Phi'$. At each iteration, for each `t` in `U`, function `sem_af_aux` is used to inspect each predecessor `y` of `t` and decrease the value of `count[y]`. When `count[y]` becomes 0, `y` is added to `U`, as it is proved that all the successors of `y` satisfy $\mathtt{AF}\Phi'$; no state is added twice to `U` (which is guaranteed by the check `if count[y] > 0` in function `sem_af_aux`).

```
let A = $[\![\Phi']\!]^{\mathcal{M}}$;
M := emptyset;
foreach (x in X)
    F := time_sec(A,x);
    U := F;
    foreach (t in (T minus F))
        count[t] := cardinality (Cl_T({t}));
    foreach (t in F)
        count[t] := 0;
    while(U != emptyset)
        U' := U;
        U := emptyset;
        foreach (t in U')
            sem_af_aux(F,U,count,t);
    M := M union ({x} times F);
return M;

function sem_af_aux(F,U,count,t)
  foreach (y in Pred_T(t))
    if count[y] > 0 then
      count[y] := count[y] - 1;
      if (count[y] = 0)
      then
          U := U union {y};
          F := F union {y};
```

Case $\mathtt{E}(\Phi_1 \mathcal{U} \Phi_2)$*:* In this case, the algorithm computes the set of points that either satisfy Φ_2, or satisfy Φ_1 and can reach points satisfying Φ_2 in a finite number of temporal steps. This is accomplished by maintaining, along the `while` loop, the set `F` of points that have already been shown to be in this situation (initialised to the points satisfying Φ_2), and the set `L` of points that satisfy Φ_1, are not in `F`, and can reach `F` in one (temporal) step. At each iteration, `F` is augmented by the points in `L`, and `L` is recomputed. When `L` is empty, `F` contains all the required points. The set `P`, initialised to the points satisfying Φ_1, is used to guarantee termination, or more precisely, that no node is added twice to `L`.

```
let A = $[\![\Phi_1]\!]^{\mathcal{M}}$;
let B = $[\![\Phi_2]\!]^{\mathcal{M}}$;
F := B;
P := A diff B;
L := pred_time(F) inter P;
while(L <> emptyset)
    F := F union L;
    P := P diff L;
    L := pred_time( L ) inter P;
return F
```

In the implementation, available at [10], the definition of the Kripke structure is given by a file containing a graph, in the plain text graph description language[9] `dot`. Quasi-discrete closure models are provided either in the form of a graph, or in the form of a set of images, one for each state in the Kripke structure, having the same size. The colours of the pixels in the image are the valuation function, and atomic propositions actually are colour ranges for the red, green, and blue components of the colour of each pixel. In this case, the model checker verifies a special kind of closure spaces, namely finite regular grids.

The model checker interactively displays the image corresponding to a "current" state. The most important command of the tool is `sem` *`colour formula`*, that changes the colour of points satisfying the given formula, to the specified colour, in the current state. The tool has the ability to define parametrised names for formulas (no recursion is allowed). Formulas are automatically saved and restored from a text file. The implementation is that of a so-called "global" model checker, that is, all points in space-time satisfying the given formulas are coloured/returned at once. More information on the tool, as well as the complete source code, is available at [10].

The complexity of the currently implemented algorithm is linear in the product of number of states plus arcs of the temporal model, subformulas, and points plus arcs of the space, which is a consequence of the algorithm described in [6] being linear in the number of points, and the classical algorithm for CTL being linear in the number of states (in both cases, for each specific formula). Such efficiency is sufficient for experimenting with the logic (see [5]), but if both the space and the Kripke structures are large, model checking may become impractical.

[9] Further information on the dot notation can, for example, be found at http://www.graphviz.org/Documentation.php.

Remark 1. Even though we consider a thorough performance analysis of the basic algorithm beyond the scope of this preliminary investigation, and possibly redundant, we can provide some hints about the feasibility and the efficiency problems of spatio-temporal model checking. Our prototype has been implemented in OCaml, trying to make use of the declarative features of the language. For example, we use the `Set` module of OCaml, implementing a purely functional data type for sets, in order to make use of Definition 8 directly, rather than attempting to use bit arrays to improve performance, as it is typical in global model checking. In the example of Section 2, we considered rather small Kripke frames, in the order of one hundred states. However, the images associated to each state contain around one million points. Therefore, even though the state space seems rather small, the number of examined points in space-time is in the order of 50-100 millions of states. The model checker is able to perform the required analyses in a time that roughly varies between some seconds and 30 minutes, depending on the formula, on a quite standard laptop computer. On the one hand, this proves that non-trivial examples may be analysed using the simple algorithm we proposed, but on the other hand, the same data strongly suggests that effective optimisations need to be found to make large-scale spatio-temporal model checking feasible (more on this in the conclusions).

6 Examples

Finally, we show some simple examples to illustrate operation of the tool. Consider the Kripke frame in Fig. 2. To each state, an image is associated, that the model checker considers an undirected graph whose nodes are pixels, and whose arcs go from each pixel to the neighbouring ones, in the four main directions *north*, *south*, *east*, *west*. The image associated to each state are shown in Fig. 3.

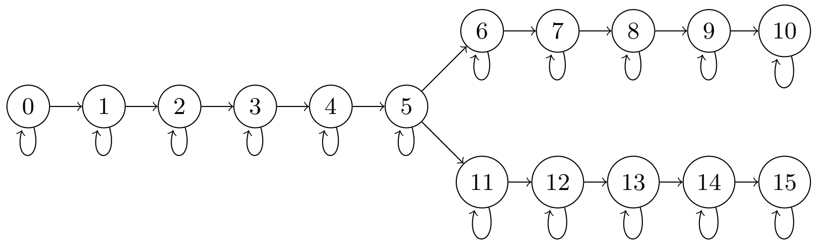

Fig. 2: The Kripke frame of our example.

Let us consider the green circle with red boundary, in the first image of Fig. 3. The centre of the circle in the figures moves along time towards the right.

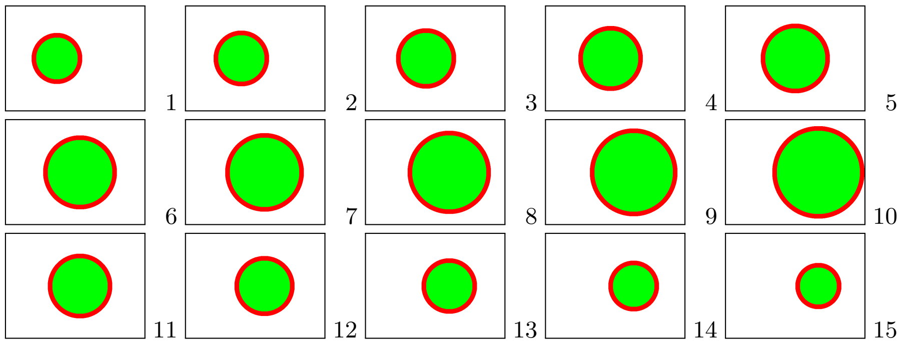

Fig. 3: The images providing valuations for the atomic propositions. Each valuation depicts a green, filled circle with a red border.

Its radius grows at constant speed in turn. Then, in state 5, there is a non-deterministic choice point. In the first possible future (states $6 - 10$), the radius keeps growing, whereas in the second future (states $11 - 15$) the radius shrinks. In the following, we shall use atomic propositions g, r, evaluating to the green and red points (boundary of the green area) in the figures.

Let us first consider the spatial formula $g\mathcal{S}r$ (green points surrounded by a red boundary). Such formula is evaluated, colouring in blue the points satisfying it, by executing the command below. Its output is displayed in Fig. 4, for each point in space and state of the Kripke structure:

```
sem blue S[<g>,<r>]
```

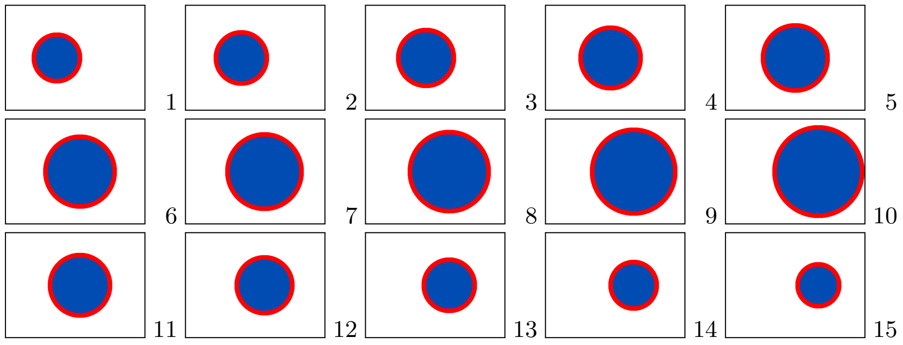

Fig. 4: The points satisfying $g\mathcal{S}r$ are displayed in blue. These are the filled circles; their borders remain red.

A second example is the spatio-temporal formula $\texttt{EF}(g\mathcal{S}r)$, computed by:

```
sem blue EF (S[<g>,<r>])
```

See output in Fig. 5. For each point in space and temporal state, the points that will eventually satisfy green and be surrounded by red, are coloured in blue.

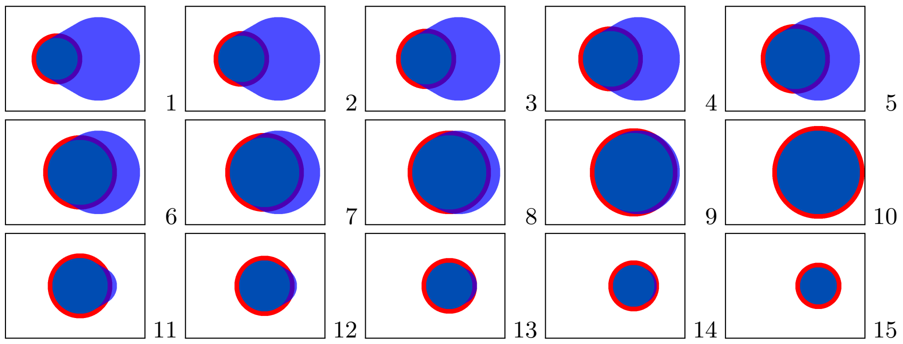

Fig. 5: In blue, the semantics of $\mathtt{EF}(g\mathcal{S}r)$.

Finally, we show the semantics of the spatio-temporal formula $\mathtt{AG}g$, characterising points that will be green forever in all futures. In Fig. 6 we show the output of

```
sem blue AG <g>
```

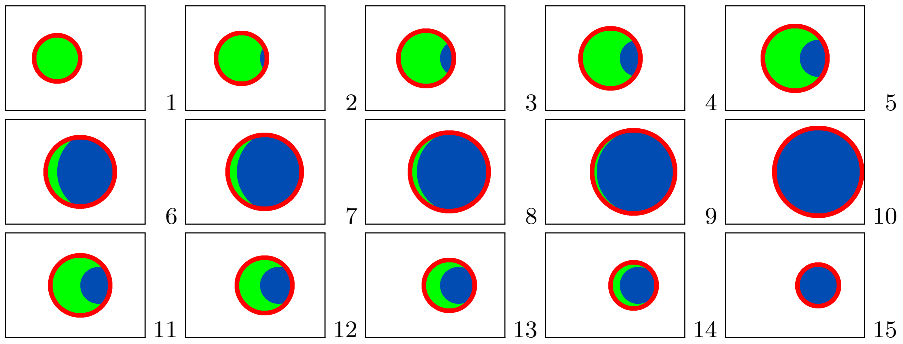

Fig. 6: In blue, the semantics of $\mathtt{AG}g$.

In states $1 - 5$ of Fig. 6, the valuation of the formula in each state is the intersection of two circular areas, namely the intersection of the green area in the chosen state and the green area in state 15. By this, in particular, the valuation of the formula is the empty set in state 1. In states $11 - 15$, the situation is similar, since state 15 is a possible future. On the other hand, in states $6 - 10$, state 15 is not reachable, thus the area which will be forever green is larger.

7 Conclusions and future work

In this paper we studied an extension of the Spatial Logic of Closure Spaces of [6] with classical CTL temporal logic operators. A simple, proof-of-concept, temporal extension of the spatial model checker for SLCS has also been presented together with a simple example. The spatio-temporal model checker has been used for a urban transportation case study, as described in detail in [5], and for analysing properties of bike sharing systems [7].

The use of a spatial model checker provides us with a sophisticated tool for checking properties of systems where location plays an important role, as it does in many collective adaptive systems. By enhancing this standpoint with a temporal perspective, the interplay of space and time allows one to define complex spatio-temporal formulas, predicating over the relation between points of a spatial model that varies over branching time.

Current work is focused on defining *collective* variants of spatial and spatio-temporal properties; that is, the satisfaction value of a formula is defined on a set of points, rather than on a single point, so that the satisfaction value of a formula with respect to a set of points (a collective property) is not necessarily determined by the satisfaction values over the points composing the set (an individual property). Such interpretation of spatio-temporal logics is particularly motivated by the setting of collective adaptive systems and *emergent properties.*

High priority in future work will be given to the investigation of various kinds of optimisations for spatio-temporal model checking, including partition refinement of models, symbolic methods, and on-line algorithms taking advantage of differential descriptions of the change between system states. An orthogonal, but nevertheless interesting, aspect of spatio-temporal computation is the introduction of probability and stochastic aspects, as well as the introduction of metrics, yielding bounded versions of the introduced spatio-temporal connectives. Such features will be studied in the context of STLCS. Investigating efficient model checking algorithms in this setting is important for practical applications, which are very often quantitative rather than boolean.

Another ongoing work is the development of qualitative and quantitative spatio-temporal analysis of the behaviour of complex systems, which was started in [13], and features an extension of *Signal Temporal Logic* to accommodate spatial information. In that case, models are deterministic (thus non-branching) and *monitoring* plays a central role. Single, infinite traces (intended to be the outcome of some approximation of a complex sytem, described by a system of differential equations) are analysed to check whether specific spatio-temporal properties are satisfied, such as, the formation of specific patterns.

References

1. Marco Aiello, Ian Pratt-Hartmann, and Johan van Benthem, editors. *Handbook of Spatial Logics.* Springer, 2007.
2. C. Baier and J. P. Katoen. *Principles of model checking.* MIT Press, 2008.

3. Luís Caires and Hugo Torres Vieira. SLMC: A tool for model checking concurrent systems against dynamical spatial logic specifications. In *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS*, volume 7214 of *LNCS*, pages 485–491. Springer, 2012.
4. V. Ciancia. topochecker - a topological model checker, 2015. `http://fmt.isti.cnr.it/topochecker` – `https://github.com/vincenzoml/topochecker`.
5. V. Ciancia, S. Gilmore, G. Grilletti, D. Latella, M. Loreti, and M. Massink. Spatio-temporal model-checking of vehicular movement in transport systems. *submitted for journal publication*, available from the authors.
6. V. Ciancia, D. Latella, M. Loreti, and M. Massink. Specifying and Verifying Properties of Space. In *8th IFIP-TCS conference, Track B*, volume 8705 of *LNCS*, pages 222–235. Springer, 2014.
7. V. Ciancia, D. Latella, M. Massink, and R. Paskauskas. Exploring spatio-temporal properties of bike-sharing systems. In *9th International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, volume to appear. IEEE, 2015.
8. E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 2001.
9. A. Galton. A generalized topological view of motion in discrete space. *Theoretical Computer Science*, 305(1–3):111 – 134, Elsevier, 2003.
10. G. Grilletti and V. Ciancia. STLCS model checker, 2014. `https://github.com/cherosene/ctl_logic`.
11. Iman Haghighi, Austin Jones, Zhaodan Kong, Ezio Bartocci, Radu Grosu, and Calin Belta. Spatel: a novel spatial-temporal logic and its applications to networked systems. In Antoine Girard and Sriram Sankaranarayanan, editors, *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC'15, Seattle, WA, USA, April 14-16, 2015*, pages 189–198. ACM, 2015.
12. R. Kontchakov, A. Kurucz, F. Wolter, and M. Zakharyaschev. Spatial logic + temporal logic = ? In M. Aiello, I. Pratt-Hartmann, and J. van Benthem, editors, *Handbook of Spatial Logics*, pages 497–564. Springer, 2007.
13. Laura Nenzi, Luca Bortolussi, Vincenzo Ciancia, Michele Loreti, and Mieke Massink. Qualitative and quantitative monitoring of spatio-temporal properties. In *Runtime Verification - 6th International Conference, RV*, volume 9333 of *Lecture Notes in Computer Science*, pages 21–37. Springer, 2015.