QUANTICOL

A Quantitative Approach to Management and Design of Collective and Adaptive Behaviours

quantiCOL

http://www.quanticol.eu

TR-QC-10-2014

A spatio-temporal model-checker

Revision: 1.4; May, 25, 2015

Author(s): Vincenzo Ciancia (CNR), Gianluca Grilletti (Scuola Normale Superiore, Pisa), Diego Latella (CNR), Michele Loreti (IMT), Mieke Massink (CNR)

Publication date: December, 15, 2014

Funding Scheme: Small or medium scale focused research project (STREP)
Topic: ICT-2011 9.10: FET-Proactive 'Fundamentals of Collective Adaptive Systems' (FOCAS)
Project number: 600708

Coordinator: Jane Hillston (UEDIN)
e-mail: Jane.Hillston@ed.ac.uk
Fax: +44 131 651 1426

Part. no.	Participant organisation name	Acronym	Country
1 (Coord.)	University of Edinburgh	UEDIN	UK
2	Consiglio Nazionale delle Ricerche – Istituto di Scienza e Tecnologie della Informazione "A. Faedo"	CNR	Italy
3	Ludwig-Maximilians-Universität München	LMU	Germany
4	Ecole Polytechnique Fédérale de Lausanne	EPFL	Switzerland
5	IMT Lucca	IMT	Italy
6	University of Southampton	SOTON	UK
7	Institut National de Recherche en Informatique et en Automatique	INRIA	France

COOPERATION

Contents

Abstract

We define a spatio-temporal logic, the *Spatio-temporal Logic of Closure Spaces* (STLCS). The logic is interpreted on *Kripke models*, in which each state has an associated *closure space*. The logic extends CTL with spatial operators, in the spirit of topological modal logic. More specifically, we add to CTL the operators of the *Spatial Logic of Closure Spaces*, including the *spatial until* 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. We present a simple model-checking algorithm for STLCS, which has been implemented in a prototype tool.

1 Introduction

This technical report is meant to provide the definitions and references for a recently developed (prototypical) *spatio-temporal* model-checker. In particular, we start from *spatial* verification of the *Spatial Logic of Closure Spaces* (SLCS), as described in [2, 3]. For an introduction to spatial logics, we refer the reader to [11], and the `QUANTICOL` technical report [4].

Spatio-temporal logics may be defined by permitting mutually recursive nesting of spatial and temporal operators. Several combinations are possible, depending on the chosen spatial and temporal fragments, and the permitted combinations of the two. A great deal of possibilities are explored in [10], for spatial logics based on topological spaces. We investigated one such structure in this work, 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). In this work, we assume knowledge of CTL; for a further introduction to and more details on CTL and its model checking, the reader may consult [6, 1].

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[1] $\mathcal{X}\Phi$ ("next"), `F`Φ ("eventually"), `G`Φ ("globally"), $\Phi_1 \mathcal{U} \Phi_2$ ("until"), but unlike CTL, in this case Φ, Φ_1 and Φ_2 are STLCS formulas. 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 is always, i.e. in every state in the path, green and surrounded by blue. A further, nested, example is the STLCS formula `EF (green S (AX blue))`. This formula is satisfied in a point x in the graph associated to 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.

Spatio-temporal model checking is performed using a variant of the classical CTL labelling algorithm [5, 1], augmented with the algorithm in [2] for the spatial fragment. The algorithm has been implemented in a prototype tool, which is publicly available at [9].

2 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 [8]. See also [2] for a thorough description of SLCS, the spatial logic of closure spaces, and its model-checking algorithm.

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

Definition 2.1. 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 definition of a topological space. Below, we spell out the needed additional axiom. The link between topological and closure spaces is deep, and also involves the so-called *Alexandrov* spaces; we refer the reader to, e.g., [8] for more information.

Definition 2.2. A *topological space* is a closure space where the closure operator is *idempotent*, that is, for all $A \subseteq X$, $\mathcal{C}(\mathcal{C}(A)) = \mathcal{C}(A)$.

In [7], a discrete variant of the topological definition of the boundary of a set A is given, for the case where a closure operator is derived from a reflexive and symmetric relation (see Definition 3.1 in the next section). Therein, in Lemma 5, it is proved that the definition coincides with the one we provide below. The latter is only given in terms of closure and interior, and coincides with the definition of boundary in a topological space. We also need to define two variants on this notion, namely the *interior* and *closure* boundary (the latter is sometimes called *frontier*).

Definition 2.3. 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 $\mathcal{B}^+(A) = \mathcal{C}(A) \setminus A$.

3 Quasi-discrete closure spaces

In this section we see how a closure space may be derived starting from a *binary relation*, that is, a *graph*. The following comes from [8].

Definition 3.1. Consider a set X and a relation $R \subseteq X \times X$. A closure operator is obtained from R as $\mathcal{C}_R(A) = A \cup \{x \in X \mid \exists a \in A.(a, x) \in R\}$.

Proposition 3.2. The pair $(X, \mathcal{C}_R)$ is a closure space.

Closure spaces derived from a relation can be characterised as *quasi-discrete* spaces (see also Lemma 9 of [8] and the subsequent statements).

Definition 3.3. 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*[2] N_x; ii) for each $A \subseteq X$, $\mathcal{C}(A) = \bigcup_{a \in A} \mathcal{C}(\{a\})$.

The following is proved as Theorem 1 in [8].

Theorem 3.4. *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, whenever one starts from an arbitrary relation $R \subseteq X \times X$, the obtained closure space $(X, \mathcal{C}_R)$ enjoys minimal neighbourhoods, and the closure of a set A is the union of the closure of the singletons composing A. Furthermore, such nice properties are only true in a closure space when there is some R such that the closure operator of the space is derived from R.

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

4 The Spatio-Temporal Logic of Closure Spaces

We define a logic which is interpreted on a variant of Kripke models, where the valuation function is interpreted at a point in a closure space, or, equivalently, closure models whose valuations are parametrised over the states of a Kripke frame. In the following, we fix a set P of proposition letters.

Definition 4.1. STLCS formulas are defined by the following grammar, where p ranges over P:

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

A model $\mathcal{M}$ of the STLCS logic 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 Definition 6 of [2]. An equivalent interpretation is that the valuation function has 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. The intuition is that there is a set of possible worlds, i.e. the states in S, and a spatial structure represented by a closure space. At 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 4.2. 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 denotes a sequence of spatial models indexed by instants of time; see Fig. 1 for a pictorial illustration, in which the space is a two-dimensional structure, and valuations at each state are depicted by different colours.

Definition 4.3. Given Kripke frame $\mathcal{K} = (S, \mathcal{T})$, a *path* σ 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 4.4. 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

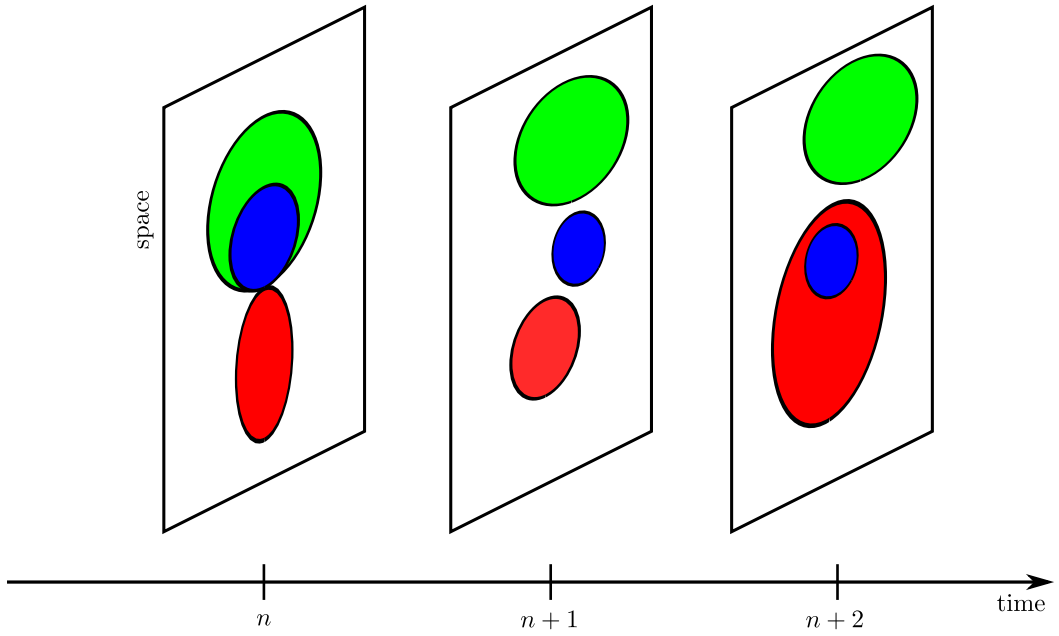

Figure 1: In spatio-temporal logics, a temporal path represents a sequence of *snapshots* that are induced by the time-dependent valuations of the atomic propositions.

state $s \in S$ as follows:

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

Indeed, the syntax we provided is rather essential. Several more operators can be derived from the basic ones; e.g., by the well known theory of Boolean logics 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., [1]; for spatial operators, [3] provides some interesting examples.

5 Model-checking

In this section we describe the model-checking algorithm, which is a variant of the well-known CTL labelling algorithm. An exhaustive reference for CTL and its model-checking can be found in [1]. We only remark that complexity of the currently implemented algorithm is linear in the product of number of states, subformulas, and points of the space, which is a consequence of the algorithm described in [3]

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).

Our algorithm operates only in the special case of quasi-discrete closure spaces, therefore we can safely assume that the closure space of the model is a graph. We describe the model-checking algorithm in terms of pseudo-code, which ought to be self-explanatory. Assume the type `Set` implementing a finite set-like data structure[3], with elements of type `El` and operations `union`, `inter`, `diff`, `times`, `emptyset`, with the obvious types.

We represent a *finite graph* as the triple

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

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 $\mathcal{M} = (\mathcal{X}, \mathcal{T})$, comprising a graph representing space

$$\mathcal{X} = (\texttt{X}, \texttt{Pred\_X}, \texttt{Cl\_X})$$

and a graph representing a Kripke frame

$$\mathcal{T} = (\texttt{T}, \texttt{Pred\_T}, \texttt{Cl\_T}).$$

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 for the temporal part, namely `E`$\mathcal{X}$, `AF`, `E` $\mathcal{U}$.

- case $\Phi = \mathcal{N}\Phi'$:

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

  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 $\mathcal{C}(A) = \bigcup_{x\in A}\mathcal{C}(\{x\})$.

- case $\Phi = \Phi_1 \mathcal{S} \Phi_2$:

```
let A = [[Φ1]]^M;
let B = [[Φ2]]^M;
F := emptyset;
foreach (t in T) {
    R := space_sec(A,t);
    Bs := space_sec(B,t);
```

[3]We remark that the complexity of operations on such type affect the complexity of the algorithm; however, since the algorithm is global, it may be implemented using an explicit lookup table, as usual in model checking, obtaining the linear complexity that we mentioned.

```
    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;
```

For every state t, we consider the spatial components of A and B (namely R and Bs). Then we apply the algorithm described in [2].

- Case $\Phi = \mathtt{E}\mathcal{X}\Phi'$:

```
let A = [[Φ']]^M;
return pred_time(A);

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

The set of predecessors (in time) of the points in space-time belonging to the semantics of Φ' are computed and returned. The auxiliary function `pred_time` is also used in the semantics of the until operator.

- Case $\Phi = \mathtt{AF}\Phi'$:

```
let A = [[Φ']]^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};
      }
    }
  }
}
```

The case for AF is essentially the efficient algorithm for EG presented in [1], 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 x ∈ 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 $\mathtt{AF}\Phi'$; this requires one to show that no state is added twice to U (which is guaranteed by the line if count[y] <> 0).

- Case $\mathtt{E}(\Phi_1 \mathcal{U} \Phi_2)$:

```
let A = [[Φ1]]^M;
let B = [[Φ2]]^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
```

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`.

6 The model-checker

In the implementation, available at [9], the definition of the Kripke structure is given by a file containing a graph, in the plain text graph description language `dot` for graph representation[4]. Quasi-discrete closure models are provided 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. Thus, the model-checker actually verifies a special kind of closure spaces, namely finite regular grids[5].

The model-checker interactively displays the image corresponding to a "current" state. The most important command of the tool is `sem` *`colour formula`*, that will change 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. However, for some auxiliary functions of the model-checker, e.g. counterexample inspection/visualization, it makes sense to consider also a current point in time, and a current formula. The complete list of commands is reported in Fig. 2.

7 A simple example

We shall now provide an example that illustrates basic usage of the STLCS model-checker. Consider the Kripke frame in Fig. 3. The images associated to each state are shown in Fig. 4.

We consider a green circle having a red boundary. Intuitively, the centre of the circle in the figures moves along time towards the right, at constant speed. Its radius grows at constant speed in turn. Then, in state 5, there is a non-deterministic choice point. In the first future (states $6-10$), the radius keeps growing, whereas in the second future (states $11-15$) the radius shrinks at constant speed again. 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.

Our first test is the 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

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

whose output is displayed in Fig. 5.

Our second example is the formula $(\neg(g \vee r))\mathcal{S}r$, computed by the command

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

whose output is displayed in Fig 6. This colours the white area (not green and not red) whose boundary is the red area.

Finally, we test the formula `AG`g, describing those points that will be green forever in all futures. The output from the command

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

is shown in Fig. 7.

In states $1-5$, 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. In states

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

[5]We note in passing that the internal implementation (available at [9]) actually represents images using a graph-like data structure; thus a loader for graphs in the `dot` format is planned.

show store; Shows currently defined formulas.

show status; Shows the current status of the tool, comprising current state and point in space, current formula, and (local) truth value of the current formula on the current time and point.

show future; Shows the possible next states, and truth value of the current formula at the current point, in the next states.

show space; Colours on the map the current point.

show time; Shows the name of the current state (an integer value coming from the .dot file representing the Kripke structure).

show formula; Shows the current formula.

set space ⟨int⟩ ⟨int⟩; Sets the current point.

set time ⟨int⟩; Sets the current state.

let ⟨ide⟩ = ⟨fsyntax⟩; Defines a new formula. Use `@x, @y, @z ...` for formal parameters (implicitly assumed in the function declaration at the moment, will be changed in the future).

sem ⟨color⟩ ⟨ide⟩;

sem ⟨color⟩ ⟨ide⟩ ⟨fsyntax⟩ ... ⟨fsyntax⟩;

sem ⟨color⟩ ⟨fsyntax⟩; Colours the points corresponding to the semantics of a formula.

save store; Saves the currently defined formulas in the file `formula.fr`.

load store; Loads formulas from `formula.fr`

save image ⟨filename⟩; Saves a set of images corresponding to the current status of the tool (including coloured points). Files are named `filenameNN.bmp` where `NN` is replaced by the name of each state of the Kripke frame.

reset; Resets the system status.

refresh; Reloads the images.

exit; Exists.

Figure 2: Commands of the model-checker.

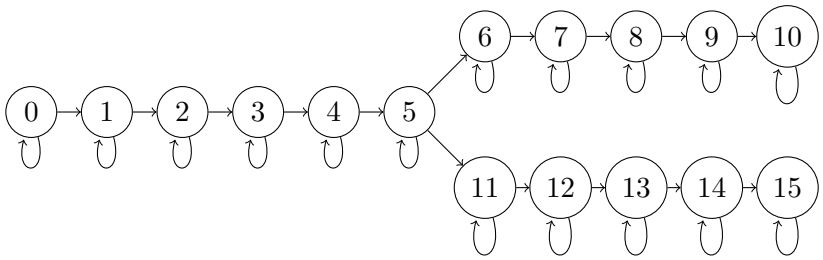

Figure 3: The Kripke frame of our example.

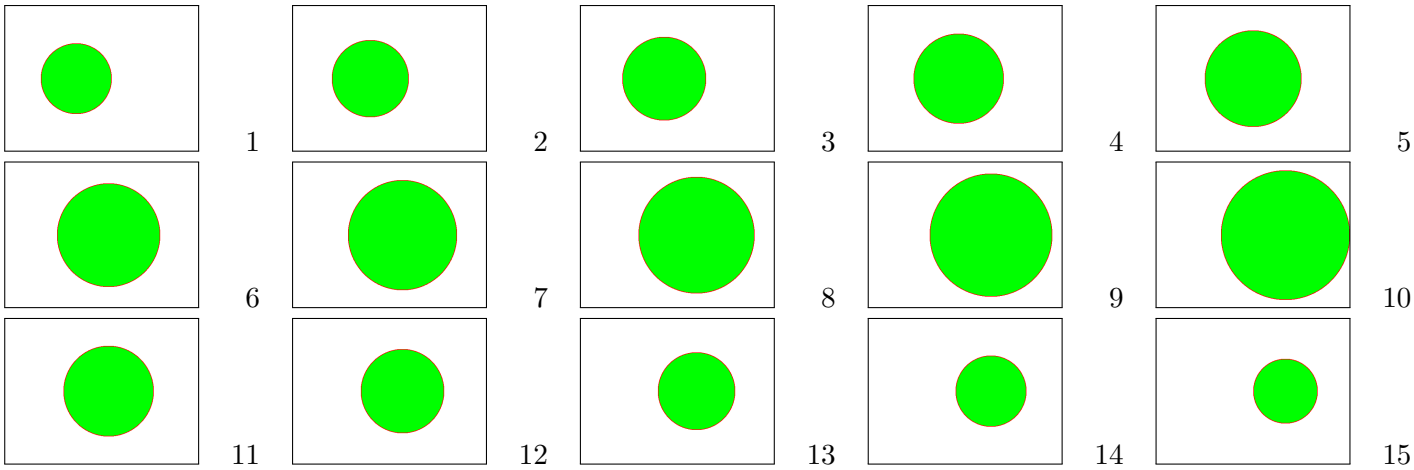

Figure 4: The images providing valuations for the atomic propositions.

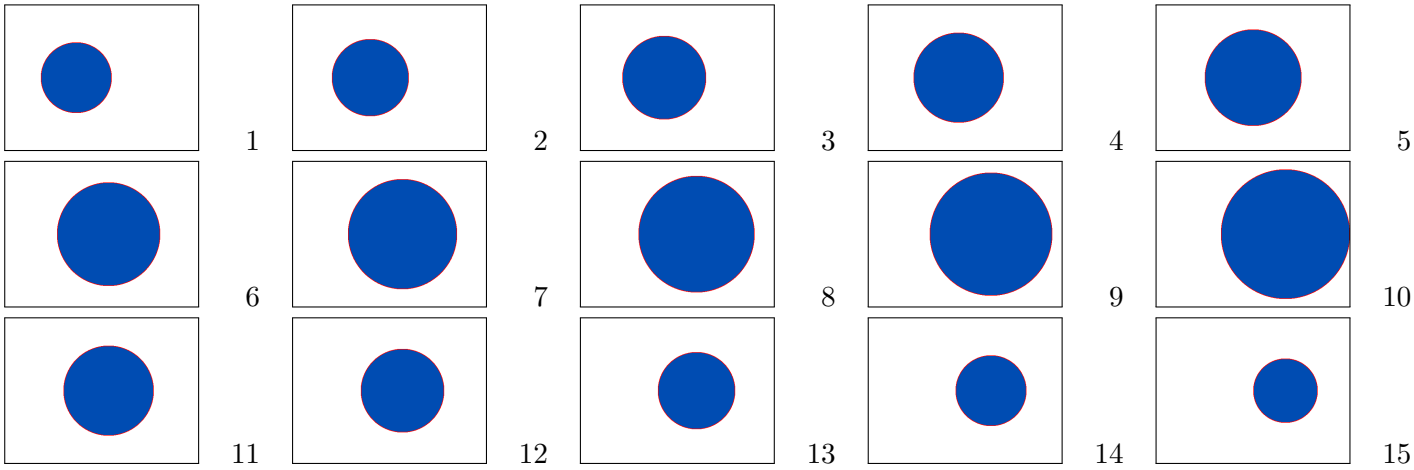

Figure 5: The points satisfying $g\mathcal{S}r$ are displayed in blue.

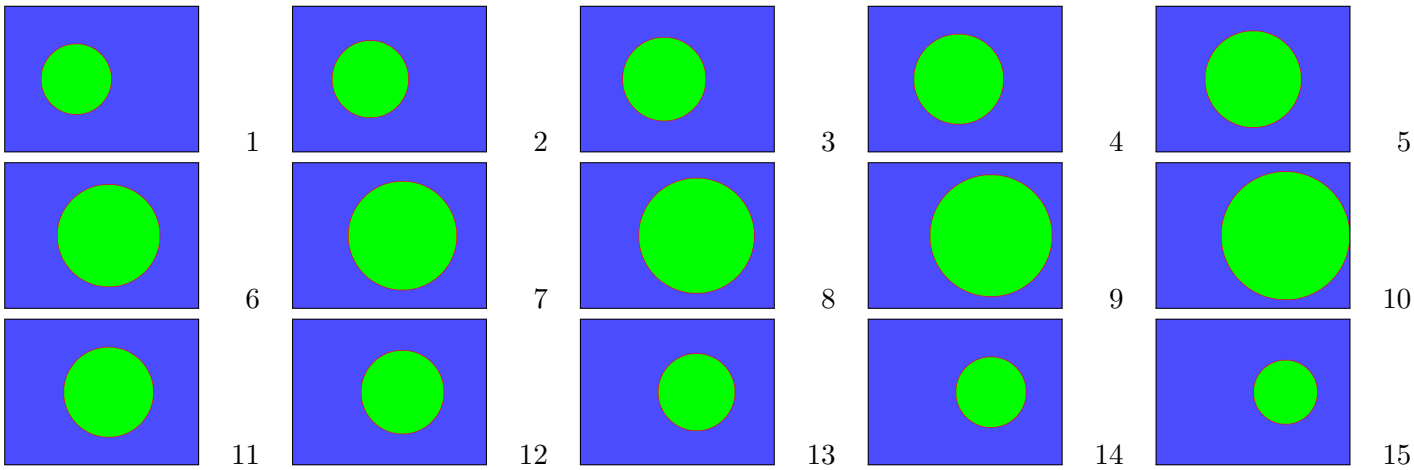

Figure 6: In blue, the semantics of $(\neg(g \vee r))\mathcal{S}r$.

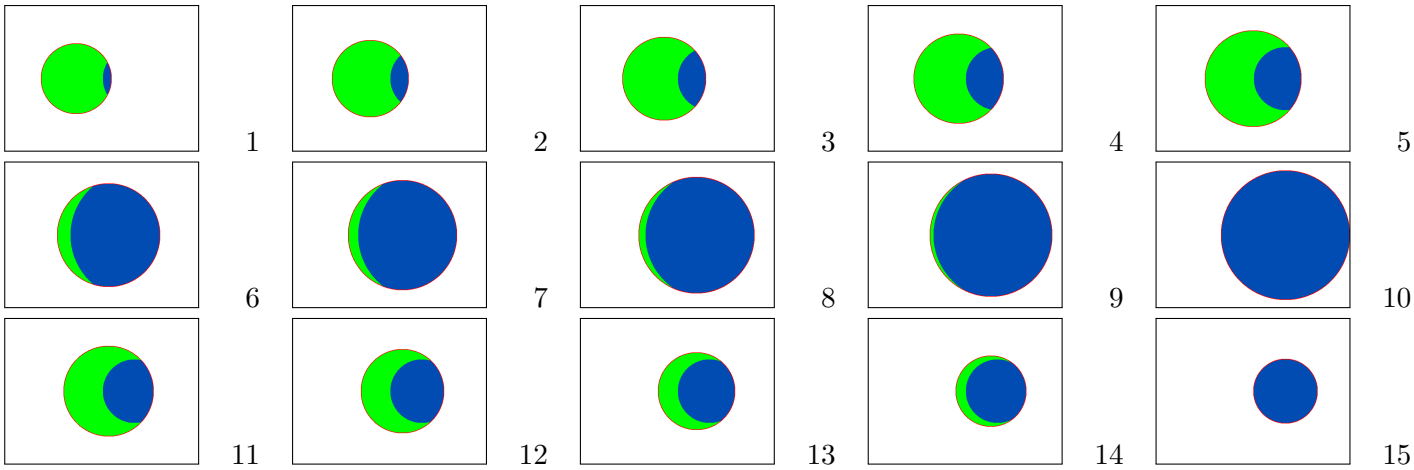

Figure 7: In blue, the semantics of $\texttt{AG}g$.

$11-15$, 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.

8 Future work

The use of a spatial model checker provides us with a sophisticated tool for checking complex properties over systems where location plays an important role, as it does in many collective adaptive systems. By enhancing the existing spatial logic and model checker 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 an image that varies over the states of a branching temporal model.

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.

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 changes between system states.

An orthogonal, but nevertheless interesting, aspect of computation is the introduction of probability and of stochastic features. Such features can also be added to our spatio-temporal logic, and investigating efficient model checking algorithms in this setting is important for practical applications, which are very often quantitative rather than boolean. The applications in the evaluation of correction strategies that we presented would be very much enhanced by forms of quantitative analysis. Such methodologies would be able to address questions like "how likely it is that a given strategy fixes the problem?" or "how frequently does the problem manifest itself, before and after applying a given strategy?".

References

[1] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[2] V. Ciancia, D. Latella, M. Loreti, and M. Massink. Specifying and Verifying Properties of Space. In Springer, editor, *The 8th IFIP International Conference on Theoretical Computer Science, TCS 2014, Track B*, volume 8705 of *Lecture Notes in Computer Science*, pages 222–235, 2014.

[3] V. Ciancia, D. Latella, M. Loreti, and M. Massink. Specifying and Verifying Properties of Space - Extendend version. Technical Report TR-QC-06-2014 also available as arXiv:1406.6393, QUANTICOL, 2014.

[4] V. Ciancia, D. Latella, and M. Massink. Logics of Space and Time. Technical Report TR-QC-01-2014, QUANTICOL, 2014.

[5] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 53–71. Springer, 1986.

[6] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT Press, 2001.

[7] Antony Galton. The mereotopology of discrete space. In Christian Freksa and DavidM. Mark, editors, *Spatial Information Theory. Cognitive and Computational Foundations of Geographic Information Science*, volume 1661 of *Lecture Notes in Computer Science*, pages 251–266. Springer Berlin Heidelberg, 1999.

[8] Antony Galton. A generalized topological view of motion in discrete space. *Theoretical Computer Science*, 305(1–3):111 – 134, 2003.

[9] Gianluca Grilletti and Vincenzo Ciancia. STLCS model checker, 2014. `https://github.com/cherosene/ctl_logic`.

[10] Roman Kontchakov, Agi Kurucz, Frank Wolter, and Michael Zakharyaschev. Spatial logic + temporal logic = ? In Marco Aiello, Ian Pratt-Hartmann, and Johan van Benthem, editors, *Handbook of Spatial Logics*, pages 497–564. Springer, 2007.

[11] Johan van Benthem and Guram Bezhanishvili. Modal logics of space. In Marco Aiello, Ian Pratt-Hartmann, and Johan van Benthem, editors, *Handbook of Spatial Logics*, pages 217–298. Springer, 2007.