Spatial Model Checking with mCRL2

Zeven, Floris J.

Award date:
2022

Spatial Model Checking with mCRL2

Master Thesis

Floris Zeven
f.j.zeven@student.tue.nl

Supervisor: dr. B. Luttik

Department of Mathematics and Computer Science
Cluster Formal System Analysis
Eindhoven Technical University
The Netherlands
June 27, 2022

Abstract

Image analysis using spatial model checking is a relatively recent approach that has promising applications in the medical field. We investigated whether it is possible to verify spatial properties using the mCRL2 toolset, which was built to verify concurrent systems and protocols. This was achieved by translating Spatial Logic for Closure Spaces (SLCS) formulae to μ-calculus formulae, proving this translation is correct, and by creating a script that associates an mCRL2 specification with an image. As mCRL2 only verifies properties on an initial state, which would correspond to a single pixel, a proof-of-concept implementation was subsequently created that utilizes the mCRL2 toolset to mark every pixel that satisfies a spatial property.

Contents

Chapter 1

Introduction

Developing software that always behaves according to its requirements is notoriously hard. Especially for systems with many components, their complexity grows quickly and verifying behaviour by hand becomes an infeasible process. A manually constructed testing suite to check system properties can contain mistakes, or miss rare circumstances that cause systems to behave unexpectedly because there are simply too many possibilities to inspect. Also, the defects that arise from a lack of verification can have disastrous consequences in critical applications.

Model checking aims to solve many of these challenges by automatically verifying properties on systems, based on formal and mathematically sound languages. This process provides guarantees that other approaches may not give, and is able to uncover faults in an earlier design stage.

Generally, applying model checking to a system requires a number of components. First, a *model* $\mathcal{M}$ contains an abstraction of a system that captures all possible behaviour. To help describe models accurately, they are often described using a formal specification language. We then wish to check whether this model satisfies a certain *property* Φ, which is formulated in a logic language. An associated model checker algorithmically attempts to validate the satisfaction of Φ in the model $\mathcal{M}$, either for some initial state x, or the entirety of $\mathcal{M}$. The former is often denoted as $\mathcal{M}, x \models \Phi$.

Model checking classically focussed on concurrent systems and protocols, and this field has seen a lot of development over the years [2]. Recent work includes the application of model checkers to industrial multi-agent systems [9], and the scheduling of wireless sensor and actuator networks [16]. However, investigating the *spatial* aspects of systems or objects with similar methodologies is a relatively recent development in Model Checking. Using models that have their basis in topology, properties such as nearness and reachability can be constructed and verified.

The medical field provides ample relevant applications to spatial analysis. Evaluating magnetic resonance images (MRI) is one of them. Approaching image segmentation from the model-checking field has lately been researched by Buonamici et al [5] and was also the main motivation for this thesis. Currently, machine learning applications are one of the most popular techniques to automate the detection and segmentation of tumours in images. While these applications have been shown to be comparable or better than experts [7], one of the challenges for these kinds of methods is the large variability between the images that different MRI scanners produce.

Also, the segmentation of MRI images relies on the relative intensity of greyscale belonging to tumours and surrounding tissue, which can be more difficult to process due to noise, and the low contrast between tumour core and surrounding fluid buildup [17]. While model checking does not inherently solve these challenges, it may help medical specialists with another challenge that machine learning introduced, namely the *explainability* of these automated methods. As Artificial Intelligence algorithms are often viewed as a black box, a specialist may wonder how an application came to the conclusion that certain regions of an MRI image are marked. An approach that builds upon a logic language potentially makes more sense to a specialist; their thought process during manual marking can be written as human-readable logic formulae and subsequently executed by a model checker.

The fragment of spatial logic proposed in [5] to perform medical analysis is the *Spatial Logic for Closure Spaces* (SLCS). This logic includes two spatial operators: a *near* operator $\mathcal{N}$ representing a single step modality, and a *surrounded* operator $\mathcal{S}$, which in practice shows some similarity to the 'until' operator in temporal logics. A point satisfies the formula $\Phi_1 \ \mathcal{S} \ \Phi_2$, if it satisfies some property Φ_1 and cannot reach a point where neither Φ_1 nor Φ_2 hold without passing a point satisfying Φ_2 first. This implies the point is surrounded by Φ_2.

As development of model checking on concurrent systems has been continuing for multiple decades, verifying complex properties has become more and more efficient. One of the toolsets that accomplishes this is mCRL2 [4], which has its own formal specification language to model concurrent systems, and contains multiple tools to verify their properties. We therefore wondered whether it is possible to verify spatial properties with mCRL2 as well, and whether this is a viable approach. This leads directly to the main research question of this thesis: can we do spatial model checking using the mCRL2 toolset?

We identified a number of steps that are needed to address this problem. Namely, both the specification and logic language that the mCRL2 toolset utilizes are different from an image specification and spatial logic SLCS. Therefore, this thesis presents a method to transform an image to a mCRL2 specification, and provides a correctness proof for a translation between SLCS- and μ-calculus formula.

The type of solution the mCRL2 toolset provides introduces the final problem that this thesis aims to solve. Namely, mCRL2 will return a Boolean stating whether a property holds or not for the initial state of the system, while we wish to find *all* pixels satisfying this property. We found that, by adding a prefix to the μ-calculus formulae, the mCRL2 toolset latently maintains the satisfiability of every pixel. The complete solution is combined in a proof of concept implementation that, given an image and SLCS formula, returns a copy of the image where every pixel that satisfies the formula is marked. This implementation was written in Python, and can be found in the `spatial_mcrl2` GitHub repository[1].

The underlying logic to write down properties in mCRL2 is based on the modal μ-calculus, which itself is an extension of Hennessy-Milner Logic with fixed point operators. While the recursive nature of fixed points introduces a new layer of complexity, it allows for the verification of much more interesting properties involving fairness, termination and reaching states 'infinitely often'. Fixed points seem to be suitable for dealing with spatial properties, as the evaluation of fixed points has much in common with flood filling algorithms that are used in spatial model checkers. For a comprehensive tutorial of the capabilities of mCRL2, see [13].

The outline of this thesis is as follows: Chapter 2 contains the necessary preliminaries of both

[1]https://github.com/FlorisZeven/spatial_mcrl2/

SLCS and the modal μ-calculus, introducing their syntaxes and semantics. We continue by presenting a translation between SLCS and the modal μ-calculus, and proving its correctness, in Chapter 3. This is achieved by first creating an Labelled Transition System (LTS) associated with the model of an image, and then, using this transformation, show there exists a μ-calculus formula that correctly represents each SLCS operator. In Chapter 4 we show how this translation leads to an implementation that verifies spatial properties on an image using the mCRL2 toolset. Finally, we conclude our work and suggest how this proof of concept could be improved and expanded upon in Chapter 5.

Chapter 2

Preliminaries

This chapter will cover the background material and definitions that will be used throughout this thesis. Some examples will be given to aid the understanding of these concepts. Definitions and examples are both closed by the $\triangleleft$ symbol.

We start out with the introduction of spatial logics in Section 2.1, delving into the syntax and semantics of a specific logic that deals with closure spaces. Subsequently, we describe the syntax and semantics for the modal μ-calculus in Section 2.2.

2.1 Spatial logic

A mathematical basis to reason about spaces, such as images and the pixels they consist of, is very useful. Approaching images in this way will aid the translation from spatial logic formulae to the modal μ-calculus, used in mCRL2, as we can formally capture logical operations. We begin with a structure that defines a space. Closure spaces generalize the more familiar concept of topological spaces. The ideas introduced here are explored and defined in more detail by Galton [12]. This section will provide the relevant ideas of closure spaces from Galton by mostly following the approach and notation by Buonamici et al. [5].

Definition 2.1. A *closure space* is a pair $(X, \mathcal{C})$, where X represents a non-empty set of points, and $\mathcal{C} : 2^X \to 2^X$ is a function denoting the closure of this point set, which for all subsets $Y, Y_1, Y_2 \subseteq X$ satisfies:

1. $\mathcal{C}(\emptyset) = \emptyset$
2. $Y \subseteq \mathcal{C}(Y)$
3. $\mathcal{C}(Y_1 \cup Y_2) = \mathcal{C}(Y_1) \cup \mathcal{C}(Y_2)$ $\triangleleft$

In contrast to topological spaces, the idempotence axiom $\mathcal{C}(\mathcal{C}(Y)) = \mathcal{C}(Y)$ is not included. Topological spaces are in fact a subclass of closure spaces as defined in Definition 2.1. We now restrict the closure space to obtain another subclass. Namely, we consider closure spaces generated by binary relations on the set of points to be *quasi-discrete* closure spaces. In particular, let $R \subseteq X \times X$ be a binary relation on X, then this gives rise to a new closure function $\mathcal{C}_R : 2^X \to 2^X$, where $\mathcal{C}_R(Y) = Y \cup \{x \mid \exists y \in Y.\ yRx\}$. This function satisfies the axioms in Definition 2.1, meaning the pair $(X, \mathcal{C}_R)$ is also a closure space. The notion that a closure space is quasi-discrete iff the above relation exists was proven by Galton [12], located in Theorem 1.

A graph is a prime example of a quasi-discrete closure space. Namely, the relation R is the set of edges, and every vertex is a point $x \in X$. This can directly be applied to the idea of an image; every pixel of an image is a vertex in a graph, and a connection between pixels is an edge. We say xRy holds if there exists an edge between pixels x and y. This means the closure of x includes itself and all *adjacent* pixels that share an edge with x. What adjacency means here may differ depending on what we want to model, but for now we assume the Von Neumann or 'orthogonal' adjacency relation applies. In other words, pixels of an image sharing a vertical or horizontal edge with pixel x are an element of $\mathcal{C}_R(x)$.

We now define the notion of paths. To simplify the proofs further on in this paper, we define a path as a sequence of points that are related under R, as opposed to a topological definition that incorporates a continuous function equipped with the closure operator.

Definition 2.2. A *path* π in $(X, \mathcal{C}_R)$ is a sequence $x_1 \dots x_n$ such that for all indices $0 \leq i \leq n-1$, it holds that $x_i \; R \; x_{i+1}$. From now on we write $\pi(i)$ to denote the point at index i on path π. ◁

Pixels are not merely objects, they often contain information. In practice, pixels either consist of three parameters that contain the intensity of each RGB colour component, or a single intensity parameter in the case of greyscale images. Therefore, we extend the quasi-discrete closure space with a valuation function that returns the value of a specific *atomic predicate* of a pixel, taken from a set P of atomic predicates. This leads to the complete definition of a closure model:

Definition 2.3. A *closure model* $\mathcal{M}$ is a tuple $((X, \mathcal{C}_R), \mathcal{V})$, where $(X, \mathcal{C}_R)$ is a quasi-discrete closure space, and valuation $\mathcal{V} : P \to 2^X$ denotes the set of points where some atomic predicate p holds. ◁

This closure model will interpret the operators of a fragment of the *Spatial Logic for Closure Spaces* (SLCS). These operators include the common negation and conjunction, but also two new operators specifically designed to tell meaningful properties of a space, namely $\mathcal{N}$ for *near* and $\mathcal{S}$ for *surrounded*. We proceed by giving the syntax of this spatial logic:

Definition 2.4. The set of SLCS formulae is defined, given a set of *atomic predicates* $p, q, r... \in P$, by the following grammar:

$$\Phi := p \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \; \mathcal{N} \; \Phi \mid \Phi_1 \; \mathcal{S} \; \Phi_2$$

We also define the logical operator $\vee$ for disjunction as:

$$\Phi_1 \vee \Phi_2 \equiv \neg(\neg\Phi_1 \wedge \neg\Phi_2)$$ ◁

As is conventional in the model checking field, the notion of *satisfaction* is used to show that some formula Φ is satisfied in a particular state x in model $\mathcal{M}$, written as $\mathcal{M}, x \models \Phi$. Let us now give meaning to the defined operators by providing their semantics:

Definition 2.5. *Satisfaction* $\mathcal{M}, x \models \Phi$ of point $x \in X$ in some model $\mathcal{M} = ((X, \mathcal{C}_R), \mathcal{V})$ with predicates $p \in P$ is defined by induction on the structure of formulae, as follows:

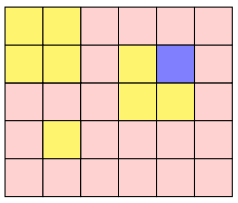
$$
\begin{aligned}
\mathcal{M}, x &\models p && \Leftrightarrow\; x \in \mathcal{V}(p)\\
\mathcal{M}, x &\models \neg\Phi && \Leftrightarrow\; \mathcal{M}, x \not\models \Phi\\
\mathcal{M}, x &\models \Phi_1 \wedge \Phi_2 && \Leftrightarrow\; \mathcal{M}, x \models \Phi_1 \text{ and } \mathcal{M}, x \models \Phi_2\\
\mathcal{M}, x &\models \mathcal{N}\,\Phi && \Leftrightarrow\; x \in \mathcal{C}_R(\{y \mid \mathcal{M}, y \models \Phi\})\\
\mathcal{M}, x &\models \Phi_1\, \mathcal{S}\, \Phi_2 && \Leftrightarrow\; \mathcal{M}, x \models \Phi_1 \text{ and } \forall \pi, i :\\
& && \qquad \pi(0) = x \text{ and } \mathcal{M}, \pi(i) \models \neg\Phi_1\\
& && \qquad \text{implies } \exists j \text{ such that } 0 < j \leq i\\
& && \qquad\quad \text{and } \mathcal{M}, \pi(j) \models \Phi_2
\end{aligned}
$$

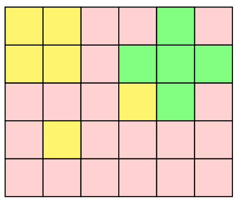
◁

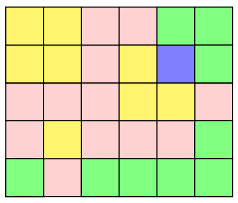
As an atomic predicate contains the information of a point x, a model satisfies some atomic predicate if it exists in the set of points that has this predicate. The negation and conjunction interpretations are straightforward, but what exactly constitutes near or surrounded is less trivial. Formula $\mathcal{N}\,\Phi$ is satisfied in a point x when there exists at least one point y satisfying Φ whose closure includes x. This can therefore be x itself, or the set of points that are R-related to x. Intuitively one can consider the near operator to mean 'a single step' away from Φ. The interpretation of surrounded, written as $\Phi_1\, \mathcal{S}\, \Phi_2$, can be viewed as there being 'no way out' of an area satisfying Φ_1 without first passing a point where Φ_2 holds. Formally, this property holds for some point x if it satisfies Φ_1, and for every path from x to some other point y where Φ_1 does not hold, we must have seen a point that satisfies Φ_2 somewhere on the path from x to y.

The following example aims to highlight the key aspects of the definitions introduced so far:

Example 2.6. Let $\mathcal{M}_1 = ((X, \mathcal{C}_R), \mathcal{V})$ be a closure model. Figure 1a shows a visual representation of $\mathcal{M}_1$, where each pixel is a point $x \in X$ that has a single atomic predicate representing their colour $c \in \{yellow, pink, blue\}$. For example, $\mathcal{V}(blue)$ would return the only blue pixel. The closure of this blue pixel is highlighted in Figure 1b with green pixels; it contains itself and its neighbours. The grid in Figure 1c highlights the pixels for which the property $(\neg\, \mathcal{N}\; yellow)$ holds. All these pixels cannot reach a yellow pixel in a single step. Finally, figure 1d verifies the property $(yellow\ \mathcal{S}\ pink)$. This does not hold for the group of three yellow pixels on the right, as starting from the blue pixel, we can reach one of these yellow pixels without passing a pink pixel first. Note that the border of our image does not 'break' the surrounding property, as paths outside the border simply do not exist. The group of yellow pixels in the top left are therefore still surrounded by pink.


(a) input grid


(b) $\mathcal{C}_R(blue)$


(c) $\neg\, \mathcal{N}\ yellow$

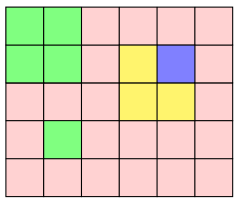

(d) $yellow\ \mathcal{S}\ pink$

Figure 1: Example 2D grid

◁

2.2 The modal-μ calculus

This section will introduce the syntax and semantics of the modal μ-calculus used throughout this paper. When referring to modal μ-calculus formulae, we often abbreviate to *μ-calculus formulae*. For more information, the reader is directed to work by Bradfield [3] and Demri et al [8], which explains these definitions in more detail.

The underlying model of the *modal μ-calculus* are Labelled Transition Systems (LTS), hence we begin this section by giving its definition:

Definition 2.7. A Labelled Transition System is a tuple $\mathcal{L} = (S, Act, \rightarrow)$, where S denotes the set of states, Act a finite set of actions, and $\rightarrow \subseteq S \times Act \times S$ the labelled transition relation. $\triangleleft$

Symbols s, t, u are used to denote states, while we use symbols a, b, c to denote actions. A transition $(s, a, t) \in \rightarrow$ is typically written as $s \xrightarrow{a} t$, namely the transition from state s to t under action a. We then move to the syntax of the modal μ-calculus:

Definition 2.8. The set of μ-calculus formulae is defined, given a set of *variables* $X, Y, Z \ldots \in Var$ and a set of actions Act, by the following grammar:

$$\Phi := X \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid [A]\Phi \mid \nu X.(\Phi)$$

Here, A is some subset of Act, and variable X is an element of Var. An occurrence of variable X is *bound* when it is in the scope of a fixed point νX, an occurrence of X is *free* if it is not bound. Fixed point $\nu X.(\Phi)$ is restricted such that any free occurrence of X in Φ must be within the scope of an even number of negations. The scope of $\nu X.(\Phi)$ is often delimited by parentheses for the sake of clarity, and parentheses are also used to resolve any ambiguity in the precedence of operations. We also define the logical operator $\vee$ for disjunction, and the duals of $[A]\Phi$ and $\nu X.(\Phi)$ as:

$$\begin{aligned}
\Phi_1 \vee \Phi_2 &\equiv \neg(\neg\Phi_1 \wedge \neg\Phi_2) \\
\langle A\rangle\Phi &\equiv \neg[A]\neg\Phi \\
\mu X.(\Phi) &\equiv \neg\nu X.(\neg\Phi)[\neg X/X]
\end{aligned}$$

Note that $[\Psi/X]$ represents the substitution of free variables X with Ψ, thus only substituting when a variable is not bound to any fixed point variable other than X. We also assume that in a formula, there exist no occurrences of operators μ and ν that bind a variable with the same name. Finally, we define the Boolean constant $true = \nu X.(X)$ and $false = \neg true$. $\triangleleft$

This syntax allows us to verify an enormous amount of surprisingly complex properties of an LTS. The Boolean constants are used to represent all states and no state respectively. Formulae can be negated and put in conjunction using their corresponding logical operators.

Some clarification for the other operators is then appropriate. Essentially, the box modality $[\{a\}]\Phi$ states that formula Φ *must* hold after taking an a-action. The diamond modality is different in the sense that it captures possibility; $\langle\{a\}\rangle\Phi$ tells us that there *exists* an a-action after which Φ holds. In practice these modalities are often combined with the Boolean constants. For example, formula $[a]false$ expresses that an a-action is not possible; after any a we reach the property $false$, which holds in no state, and thus implies that any a action is impossible to perform. Conversely, the formula $\langle a\rangle true$ verifies whether an a-action is possible, since this would lead to property $true$, which holds in all states. Finally, least fixed points $(\mu X.(\Phi))$ are often constructed to model an *eventuality*, while greatest fixed points $(\nu X.(\Phi))$ are useful when defining *guarantees*. Particularly common properties involving fixed points are 'on some path consisting of only a-actions, Φ must hold at some point', depicted by $\mu X.(\Phi \vee \langle a \rangle X)$; and similarly 'on all a-paths, Φ must hold', depicted by $\nu X.(\Phi \wedge [a]X)$.

To give a formal meaning to these properties in the context of an LTS, we now move to the semantics of the modal μ-calculus:

Definition 2.9. Let $\mathcal{L} = (S, Act, \rightarrow)$ be an LTS, then a *variable assignment* for S' is a valuation $\mathcal{V} : Var \rightarrow 2^S$ assigning subsets $S' \in S$ to variables $X \in Var$ as follows:

$$\mathcal{V}[X := S'](Y) := \begin{cases} S' & \text{if } X = Y \\ \mathcal{V}(Y) & \text{otherwise} \end{cases}$$

$\triangleleft$

Definition 2.10. Let $\mathcal{L} = (S, Act, \rightarrow)$ be an LTS, and $\mathcal{V}$ be a variable assignment with regards to subsets S'. Denoting the set of states in $\mathcal{L}$ satisfying formula Φ relative to $\mathcal{V}$ is written as $\|\Phi\|^{\mathcal{L}}_{\mathcal{V}}$, where we usually omit $\mathcal{L}$ if it is clear from the context. This is inductively defined as:

$$\begin{aligned}
\|X\|_{\mathcal{V}} &= \mathcal{V}(X) \\
\|\neg\Phi\|_{\mathcal{V}} &= S - \|\Phi\|_{\mathcal{V}} \\
\|\Phi_1 \wedge \Phi_2\|_{\mathcal{V}} &= \|\Phi_1\|_{\mathcal{V}} \cap \|\Phi_2\|_{\mathcal{V}} \\
\|\Phi_1 \vee \Phi_2\|_{\mathcal{V}} &= \|\Phi_1\|_{\mathcal{V}} \cup \|\Phi_2\|_{\mathcal{V}} \\
\|[a]\Phi\|_{\mathcal{V}} &= \{s \mid \forall t : s \xrightarrow{a} t \Rightarrow t \in \|\Phi\|_{\mathcal{V}}\} \\
\|\langle a \rangle\Phi\|_{\mathcal{V}} &= \{s \mid \exists t : s \xrightarrow{a} t \wedge t \in \|\Phi\|_{\mathcal{V}}\} \\
\|\nu X.(\Phi)\|_{\mathcal{V}} &= \bigcup\{S' \subseteq S \mid S' \subseteq \|\Phi\|_{\mathcal{V}[X:=S']}\} \\
\|\mu X.(\Phi)\|_{\mathcal{V}} &= \bigcap\{S' \subseteq S \mid \|\Phi\|_{\mathcal{V}[X:=S']} \subseteq S'\}
\end{aligned}$$

From this we also derive the meaning of Boolean constants $true$ and $false$:

$$\begin{aligned}
\|true\|_{\mathcal{V}} &= \|\nu X.(X)\|_{\mathcal{V}} = S \\
\|false\|_{\mathcal{V}} &= S - \|true\|_{\mathcal{V}} = S - S = \emptyset
\end{aligned}$$

$\triangleleft$

As an addition to this, we define the satisfiability of Φ for a single point, and some equivalences that follow from this definition. This will be helpful when comparing the satisfaction relations of both closure spaces in Definition 2.5 and the modal μ-calculus.

Definition 2.11. Given a modal μ-calculus-model, consisting of a labelled transition system $\mathcal{L}$ and a formula Φ. If x is a state in $\mathcal{L}$ and an element of $\|\Phi\|_{\mathcal{V}}$, then we write $\mathcal{L}, x \models \Phi$, or in other words, '$x$ satisfies Φ'. $\triangleleft$

Proposition 2.12. The following equivalences hold for the binary relation $\mathcal{L}, x \models \Phi$:

$$\begin{aligned}
\mathcal{L},x &\models \neg\Phi && \Leftrightarrow \mathcal{L}, x \not\models \Phi \\
\mathcal{L},x &\models \Phi_1 \wedge \Phi_2 && \Leftrightarrow \mathcal{L}, x \models \Phi_1 \text{ and } \mathcal{L}, x \models \Phi_2 \\
\mathcal{L},x &\models \Phi_1 \vee \Phi_2 && \Leftrightarrow \mathcal{L}, x \models \Phi_1 \text{ or } \mathcal{L}, x \models \Phi_2 \\
\mathcal{L},x &\models [a]\Phi && \Leftrightarrow \forall y : x \xrightarrow{a} y \Rightarrow \mathcal{L}, y \models \Phi \\
\mathcal{L},x &\models \langle a \rangle\Phi && \Leftrightarrow \exists y : x \xrightarrow{a} y \wedge \mathcal{L}, y \models \Phi
\end{aligned}$$

Proof. All bi-equivalences directly follow from applying Definition 2.11 to the semantics of Definition 2.10. □

By far the most complex semantics are introduced by the fixed point operators. Let us therefore go into some more detail about their evaluation. It is useful to think of fixed points as sets of states; the smallest and largest subsets $S' \subseteq S$ that satisfy the equation $S' = \|\Phi\|_{\mathcal{V}[X:=S']}$ are the solutions for least and greatest fixed points respectively. To find these sets of states in the case of least fixed points, we start with substituting all free occurrences of X within the body of the fixed point for the empty set. Then, using the variable assignment, we keep substituting X for new subsets until this set no longer grows, meaning it satisfies the equation $X = X$. In the case of greatest fixed points we do the opposite. We begin its evaluation by substituting X for the full set S, and continuously remove states from S until it stabilizes.

The notion of fixed point *approximants* describes this evaluation process. The annotation of a fixed point with an ordinal number α with some limit ω will provide us with a finite amount of new formulae. As with our intuition, the zeroth approximant of a least fixed point is the empty set $\emptyset$, and full set S for greatest fixed points. Furthermore, we define a successor ordinal $\alpha + 1$ that substitutes each occurrence of X with the previous approximation.

Definition 2.13. Let $\sigma \in \{\mu, \nu\}$, and α an ordinal with limit ω. The meaning of $\|\sigma^\alpha X.(\Phi)\|_{\mathcal{V}}$, is given by:

$$\begin{aligned}
\|\mu^0 X.(\Phi)\|_{\mathcal{V}} &= \emptyset \\
\|\nu^0 X.(\Phi)\|_{\mathcal{V}} &= S \\
\|\sigma^{\alpha+1} X.(\Phi)\|_{\mathcal{V}} &= \|\Phi\|_{\mathcal{V}[X:=T]}, \qquad \text{where } T = \|\sigma^\alpha X.(\Phi)\|_{\mathcal{V}} \\
\|\mu^\omega X.(\Phi)\|_{\mathcal{V}} &= \bigcup_{\alpha<\omega} \|\mu^\alpha X.(\Phi)\| \\
\|\nu^\omega X.(\Phi)\|_{\mathcal{V}} &= \bigcap_{\alpha<\omega} \|\mu^\alpha X.(\Phi)\| \qquad \lhd
\end{aligned}$$

To give an example, take formula $\mu X.(\langle a \rangle X \vee \langle b \rangle true)$, which in natural language states that 'there exists a path of zero or more a actions towards a point where we can perform a b action'. In the first approximation of this fixed point, all occurrences of variable X will be substituted for the empty set, meaning we evaluate $(\langle a \rangle \emptyset \vee \langle b \rangle true)$. This gives us all states that have some a-action towards the empty set (implying all states with an outgoing a-action), and take the union with states having an outgoing b-action. We then build the next approximation by substituting X for this set, hence $\langle a \rangle X$ returns all states that can perform an a-action towards X. The second approximation will therefore consist of states that can perform an a-action to a state that can do a b-action. One can see that, as we keep iterating, this gives rise to a set of states that can perform zero or more a-actions until a b-action is taken. Eventually this set of states no longer grows, and we return the set of the final approximation as the solution to our fixed point.

Similarly, formula $\nu X.([a]X \wedge \langle b \rangle true)$ expresses 'on all paths intermediate state'. The first approximation substitutes all occurrences of X for the full set S, and we end up with the states that are able to perform an a-action (since all states satisfy being in S), intersected with states that are able to perform a b-action. This facilitates the restriction on S; at some point X does not become smaller, and we have found our solution.

In Definition 2.8 we mentioned that any free occurrence of X in Φ must be within the scope of an even number of negations. This is important, because blindly constructing formulae with fixed points can lead to properties that have no meaning. More specifically, there only exists a solution to a formula if any free occurrence of X in the body of fixed point $\mu X.(\Phi)$ or $\nu X.(\Phi)$ is in the scope of an even number of negations. The easiest way to understand why is by taking the simple formula $\mu X.(\neg X)$, where the occurrence of X is preceded by a single negation. The zeroth approximation of X is the empty set, meaning its first approximation will be its complement: the full set S. The next approximation will then be the empty set again, and this repeats infinitely often. Clearly we will never obtain a solution, as it is not possible that X equals its complement. Let us therefore state the definition of a 'correct' modal μ-calculus formula:

Definition 2.14. A modal μ-calculus formula Φ is *correct* if and only if, for all subformulae of Φ of the form $\mu X.(\Phi')$ or $\nu X.(\Phi')$, all free occurrences of X in the body of Φ' are *positive*, or in other words, occur within the scope of an even number of negations. $\triangleleft$

From here on, we assume all modal μ-calculus-formulae mentioned are correct. One could argue that negation can then only be applied to variables. However, we will still see formulae where negation is applied to other operators. This is possible as long as the original formula is correct. Using the dualities in Definition 2.8, we can always rewrite the formula by pushing negations inwards to the point where they only apply to free variables, and the formula remains correct.

Take formula $\neg\nu X.(P \wedge \neg\langle a\rangle\neg X)$ as an example of this, where P is a variable representing some arbitrary property. This formula is correct as negation is applied twice to the occurrence of X in the body of its corresponding fixed point. We can rewrite this formula as follows:

$$\begin{aligned}\neg\nu X.(P \wedge \neg\langle a\rangle\neg X) &= \mu X.(\neg(P \wedge \neg\langle a\rangle X))\\ &= \mu X.(\neg P \vee \neg\neg\langle a\rangle X)\\ &= \mu X.(\neg P \vee \langle a\rangle X)\end{aligned}$$

Variable X remains in the scope of an even number of negations, because negation is only applied to variable P.

It has been shown that monotonicity property is satisfied for all correct μ-calculus formulae, which is in fact a requirement to have solutions to fixed point formulae in the first place. We do not repeat this proof but refer the reader to Lemma 8.1.10 in the book by Demri et al [8].

Lemma 2.15. Let Φ be a correct modal-μ formula and let $\mathcal{F}_\Phi : 2^S \to 2^S$ be a function mapping subsets to subsets, then for any LTS $\mathcal{L}$ and valuation $\mathcal{V}$, the map $S' \mapsto \|\Phi\|_{\mathcal{V}[X:=S']}$ is monotone with respect to $\subseteq$. $\square$

We finish this section with an example of a labelled transition system and some modal μ-calculus-formulae that apply to it.

Example 2.16. Consider the simple LTS $\mathcal{L}_1$ below:

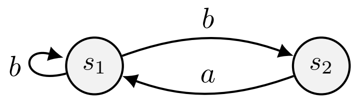

$$\begin{tikzcd} s_1 \arrow[loop left, "b"] \arrow[r, bend left, "b"] & s_2 \arrow[l, bend left, "a"] \end{tikzcd}$$

Figure 2: Example LTS $\mathcal{L}_1$

Let us verify which states in $\mathcal{L}_1$ satisfy certain μ-calculus formulae. Formula $\langle a\rangle true$ is satisfied in state s_2, since it has an outgoing a-transition. Similarly, $[b]false$ is also only satisfied in state s_2 as it has no outgoing b-transition.

Fixed point formula $\mu X.(\langle b \rangle X \vee [b]false)$ tells us that there is always some path consisting of only b-transitions that lead to a state where we can no longer perform a b-transition. Clearly s_2 has no outgoing b-transition, and from s_1 we can always take a b-transition that leads to s_2. Hence this formula is true in all states. $\triangleleft$

Chapter 3

A translation from SLCS- to modal-μ calculus formulae

This chapter works towards a formal translation from SLCS- to μ-calculus formulae. As the mCRL2 toolset verifies specifications using the modal μ-calculus, a proof showing that each operator of SLCS has a μ-calculus formula that represents the exact same property is imperative.

We approach this problem in a few steps. In Section 3.1, it is shown how a Labelled Transition System (LTS) can be constructed from the closure model of an image. Then, we describe the translation in Section 3.2, and prove that all resulting μ-calculus formulae conform to its syntax. To prove the correctness of the translation, we first characterize the μ-calculus formula for surrounded operator in Section 3.3, and conclude the chapter by combining these findings to form the correctness proof in Section 3.4.

3.1 An LTS associated with a quasi-discrete closure space

When we observe the closure model of SLCS and the modal μ-calculus model, it is clear they are not directly comparable. Namely, SLCS is defined on a quasi-discrete closure space where points are related using closure operation R, while the modal μ-calculus model is defined in the context of an LTS. Also, the valuation function $\mathcal{V}$ in SLCS is not directly defined in such a transition system. This subsection deals with the creation of an LTS that is associated with a quasi-discrete closure space. This will aid the translation that we set out to make.

Definition 3.1. Given a closure model $\mathcal{M} = ((X, \mathcal{C}_R), \mathcal{V})$, we define an LTS associated with $\mathcal{M}$ as $\mathcal{L} = (S, \mathit{Act}, \rightarrow)$ such that $S = X$, and $\mathit{Act} = P \cup \{R\}$. Finally, $\rightarrow$ is the least relation such that $x \xrightarrow{R} y$ if and only if $x \in \mathcal{C}_R(y)$, and $x \xrightarrow{p} x$ if and only if $x \in \mathcal{V}(p)$. $\triangleleft$

Intuitively, the idea behind this transformation is as follows: every point in $\mathcal{M}$ becomes a state in $\mathcal{L}$, and atomic predicates of $\mathcal{M}$ become actions of $\mathcal{L}$. We represent the binary relation R as transitions between two points in $\mathcal{L}$ under action R. By creating self-loops for each atomic predicate $p \in P$ that holds in some state x, we effectively replace the valuation function $\mathcal{V}(p)$ with the possibility of a p-transition.

This approach also ensures we can also define paths in Definition 2.2 as sequences of R-transitions.

Lemma 3.2. There exists a sequence of transitions $x_0 \xrightarrow{R} x_1 \xrightarrow{R} \ldots \xrightarrow{R} x_n$ in the LTS $\mathcal{L}$ associated with quasi-discrete closure space $\mathcal{M}$ iff there exists a path π from point x_0 to x_n in $\mathcal{M}$.

Proof. ($\Rightarrow$) From Definition 2.2 it follows there exist relations $x_i\ R\ x_{i+1}$ for all indices $0 \leq i \leq n-1$ and $n \geq 1$. Hence, point $x_i \in \mathcal{C}_R(x_{i+1})$ for all i. By Definition 3.1 we then have transitions $x_i \xrightarrow{R} x_{i+1}$ for all i, which gives rise to the sequence of transitions.
($\Leftarrow$) By Definition 3.1, each transition $x_i \xrightarrow{R} x_{i+1}$ in the sequence only exists in $\mathcal{L}$ if $x_i \in \mathcal{C}_R(x_{i+1})$. Hence, since the sequence ranges from point x_0 to x_n, by Definition 2.2 the path π from x_0 to x_n exists in $\mathcal{M}$. $\square$

Note that index $\pi(i)$ of a path in $\mathcal{M}$ corresponds to the point x_i of a transition sequence in $\mathcal{L}$. Furthermore, as R is symmetric, for every transition sequence from x_0 to x_n, there is also a sequence from x_n to x_0.

3.2 Translating SLCS formulae to modal μ-calculus formulae

With the transformation of Section 3.1 in place, we define a translation $T_{SLCS\rightarrow\mu}$, which we shorten to T from now on. We observe that this translation is not minimal; by this we mean that rewriting the surrounded formula to one where negation is only applied to variable symbols is possible, but it would be less intuitive to see how it constitutes the properties of surrounded.

Definition 3.3. We define the translation T of SLCS formulae to modal μ-calculus formulae as follows:

$$\begin{array}{rlcl}
\text{ATOMIC PREDICATE:} & T(p) & = & \langle p\rangle true \\
\text{NEGATION:} & T(\neg\Phi) & = & \neg T(\Phi) \\
\text{CONJUNCTION:} & T(\Phi_1 \wedge \Phi_2) & = & T(\Phi_1) \wedge T(\Phi_2) \\
\text{NEAR:} & T(\ \mathcal{N}\ \Phi) & = & \langle R\rangle T(\Phi) \\
\text{SURROUNDED:} & T(\Phi_1\ \mathcal{S}\ \Phi_2) & = & T(\Phi_1) \wedge \neg\mu X.(\\
 & & & \quad \neg(T(\Phi_1) \vee T(\Phi_2)) \vee \\
 & & & \quad (T(\Phi_1) \wedge \langle R\rangle X) \\
 & & &)
\end{array} \quad \triangleleft$$

Informally, the translation can be explained as follows:

If point x has some atomic proposition p, this is represented as the possibility of performing action p in the associated LTS. If p can be done in x, the atomic proposition holds. Negation and conjunction connectives behave as expected.

For a point to be 'near' Φ in a closure model, its modal-μ equivalent is seen as reaching a state satisfying the translation of Φ in a single step, represented by $\langle R\rangle$. Because of how R is defined in Definition 3.1, it is possible to transition to itself, or adjacent states. After taking action R, formula $T(\Phi)$ can be checked appropriately.

The surrounded operator is more involved. Focusing on the least fixed point in the formula first, we aim to find all states where neither $T(\Phi_1)$ nor $T(\Phi_2)$ hold that can be reached using only R transitions to states where $T(\Phi_1)$ holds. We take the complement of these states by negating the fixed point, and then intersecting with all states where $T(\Phi_1)$ hold, leaving us with the states where $T(\Phi_1)$ holds and that cannot reach any state satisfying neither $T(\Phi_1)$ nor $T(\Phi_2)$ without first passing a state satisfying $T(\Phi_2)$. Therefore, these states are surrounded by property $T(\Phi_2)$. The approach for this translation was inspired by Buonamici et al [5], whose algorithm in their tooling verifies the same surrounded property.

It is crucial that our translation T only produces modal μ-calculus formulae that are correct.

Lemma 3.4. If Φ is a spatial logic formula, then $T(\Phi)$ is a correct modal μ-calculus formula.

Proof. We prove this property by induction on the structure of Φ.

A formula $T(\Phi)$ is correct iff, for all subformulae of the form $\mu X.(\Phi')$ or $\nu X.(\Phi')$, all free occurrences of X in the body of Φ' occur within the scope of an even number of negations.

Our only base case is given by atomic propositions. Let $\Psi = p$, then the resulting modal-μ formula $T(\Psi)$ is $T(p)$, which equals $\langle p\rangle \mathit{true}$. This formula contains the fixed point $\mu X.(X)$ by the definition of *true*. However, no negation is applied to X in the body of μ, showing that $T(p)$ is correct.

Take two arbitrary spatial logic formulae Φ_1 and Φ_2, then we assume $T(\Phi_1)$ and $T(\Phi_2)$ are correct modal-μ formulae. (IH)

Let $\Psi = \neg\Phi_1$. By Definition 3.3 the translation of $T(\neg\Phi_1)$ equals $\neg T(\Phi_1)$. Since this negation is not applied to free variables X in the body of fixed points in our translation, meaning $T(\Phi_1)$ is never X, and we assumed by IH that $T(\Phi_1)$ is a correct formula, formula $\neg T(\Phi_1)$ is also correct.

Let $\Psi = \Phi_1 \wedge \Phi_2$. Then, $T(\Psi) = T(\Phi_1 \wedge \Phi_2)$, which by Definition 3.3 equals $T(\Phi_1) \wedge T(\Phi_2)$. We assumed $T(\Phi_1)$ and $T(\Phi_2)$ to be correct formulae, hence their conjunction is also correct as this operator does not introduce negations nor fixed points.

Let $\Psi = \mathcal{N}\ \Phi_1$. Then, $T(\Psi) = T(\mathcal{N}\ \Phi_1)$, which by Definition 3.3 equals $\langle R\rangle T(\Phi_1)$. We assumed $T(\Phi_1)$ to be a correct formula. The diamond modality does not introduce a fixed point, thus $\langle R\rangle T(\Phi_1)$ is also a correct formula.

Finally, let $\Psi = \Phi_1\ \mathcal{S}\ \Phi_2$. Then $T(\Psi) = T(\Phi_1\ \mathcal{S}\ \Phi_2)$, which by Definition 3.3 equals:

$$\begin{aligned} T(\Phi_1\ \mathcal{S}\ \Phi_2) \ = \ & T(\Phi_1) \wedge \neg\mu X.(\\ & \quad \neg(T(\Phi_1) \vee T(\Phi_2)) \vee \\ & \quad (T(\Phi_1) \wedge \langle R\rangle X) \\ &) \end{aligned}$$

We assumed $T(\Phi_1)$ is a correct formula, thus the only relevant subformula in $T(\Phi_1) \wedge \neg\mu X.(\vartheta)$ which remains to be proven is $\mu X.(\vartheta)$, where ϑ represents the body of the fixed point above.

This implies we must show that all free occurrences of X in ϑ are in the scope of an even number of negations, or, in other words, are positive.

Let us first look at the instances of $T(\Phi_1)$ and $T(\Phi_2)$ in ϑ. By our translation in Definition 3.3, there clearly is only one possibility where $T(\Phi_1)$ or $T(\Phi_2)$ in ϑ contains an occurrence of X, namely when the corresponding spatial logics formulae Φ_1 or Φ_2 contained a $\mathcal{S}$ operator. However, these occurrences of X will then be bound to the fixed point with the same name induced by the translation of this $\mathcal{S}$ operator. Hence, there are no free occurrences of X in all subformulae $T(\Phi_1)$ or $T(\Phi_2)$ in ϑ. From this we can also conclude that the entire subformula $\neg(T(\Phi_1) \vee T(\Phi_2))$ contains no free occurrences of X.

The only occurrence of X that then remains in ϑ is free and positive. The diamond modality, conjunction and disjunction operators applied will give rise to zero (in the case of diamond and conjunction) or two (in the case of disjunction) negations. It is therefore clear that all free occurrences of X in $\mu X.(\vartheta)$ will remain positive. Since this is what we set out to prove, we conclude the entire formula given by $T(\Phi_1 \; \mathcal{S} \; \Phi_2)$ is correct.

As we have handled all cases of the structure of Φ, this finishes the proof. $\square$

3.3 Correct characterization of a fixed point formula related to surrounded

In Subsection 3.4 the correctness proof of translation T will be given. However, this proof will be easier to follow if the fixed point formula in the surrounded case is characterized as a set of states that satisfies certain properties. This subsection is therefore about the subformula that is exactly the least fixed point in the translation of $T(\Phi_1 \; \mathcal{S} \; \Phi_2)$. The approach used to show correctness is heavily inspired by the book 'Reactive Systems' by Aceto, Ingolfsdottir, Larsen and Srba [1], with one difference being that we use approximants to simplify our proofs.

To achieve our characterization we first denote the following shorthand notation for the operators of the box- and diamond modalities, where $[\cdot a \cdot], \langle \cdot a \cdot \rangle : 2^S \rightarrow 2^S$, as:

$$\begin{aligned}
\|[a]\Phi\|_{\mathcal{V}} &= [\cdot a \cdot]\|\Phi\|_{\mathcal{V}} = \{s \mid \forall t : s \xrightarrow{a} t \Rightarrow t \in \|\Phi\|_{\mathcal{V}}\} \\
\|\langle a \rangle \Phi\|_{\mathcal{V}} &= \langle \cdot a \cdot \rangle \|\Phi\|_{\mathcal{V}} = \{s \mid \exists t : s \xrightarrow{a} t \wedge t \in \|\Phi\|_{\mathcal{V}}\}
\end{aligned}$$

Informally, the property that we wish our fixed point denotes is described as the set of states that:

1. Neither satisfies $T(\Phi_1)$ nor $T(\Phi_2)$; or
2. there exists a sequence of states satisfying $T(\Phi_1)$ with R-labelled transitions reaching a state satisfying 1.

Let us call this set $\mathcal{R}$, and define it formally as:

$$\begin{aligned}
\mathcal{R} = \{s \mid \exists n : s = s_n \xrightarrow{R} s_{n-1} \xrightarrow{R} \ldots \xrightarrow{R} s_0 \wedge \forall i.\, 0 < i \leq n : s_i \in \|T(\Phi_1)\| \\
\wedge\, s_0 \in (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2)\|))\}
\end{aligned}$$

of which we note that n can be zero; $\mathcal{R}$ would then only consist of states satisfying neither $T(\Phi_1)$ nor $T(\Phi_2)$.

We wish to show that this set is indeed characterized by the subformula that is the body of the fixed point in the translation for surrounded, depicted in Definition 3.3. Let us give this subformula the name $\mu\mathcal{R}$ and define it as:

$$\mu\mathcal{R} = \mu X.(\neg(T(\Phi_1) \vee T(\Phi_2)) \vee (T(\Phi_1) \wedge \langle R \rangle X))$$

In Lemma 3.4 we have shown this fixed point is a correct formula. We then obtain the semantics of $\mu\mathcal{R}$, written as $\|\mu\mathcal{R}\|$, using Definition 2.10:

$$\|\mu\mathcal{R}\| = \|\mu X.(\Psi)\|, \text{ where } \|\Psi\| = (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2)\|) \cup (\|T(\Phi_1)\| \cap \langle \cdot R \cdot \rangle X)$$

From this we also derive:

$$\|\mu X.(\Psi)\| = \bigcap\{S' \subseteq S \mid \|\Psi\|_{\mathcal{V}[X:=S']} \subseteq S'\}, \text{ where}$$
$$\|\Psi\|_{\mathcal{V}[X:=S']} = (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2)\|)) \cup (\|T(\Phi_1)\| \cap \langle \cdot R \cdot \rangle S')$$

To prove that $\|\mu\mathcal{R}\|$ then correctly characterizes $\mathcal{R}$, we wish to show that in any LTS associated with a quasi-discrete closure space, they capture exactly the same states.

Lemma 3.5. In any LTS $\mathcal{L} = (S, Act, \rightarrow)$, $\|\mu\mathcal{R}\| = \mathcal{R}$

Proof. We prove both subset inclusions individually:

($\|\mu\mathcal{R}\| \subseteq \mathcal{R}$)
It is then sufficient to show that $\|\Psi\|_{\mathcal{V}[X:=\mathcal{R}]} \subseteq \mathcal{R}$, as, if this is true for one set of which the intersection is taken, the solution of this intersection given by $\|\mu X.(\Psi)\|$ must also be a subset of $\mathcal{R}$, which would imply $\|\mu\mathcal{R}\| \subseteq \mathcal{R}$.

We therefore continue this inclusion by proving $\|\Psi\|_{\mathcal{V}[X:=\mathcal{R}]} \subseteq \mathcal{R}$, needing to show that for all states $t \in \|\Psi\|_{\mathcal{V}[X:=\mathcal{R}]}$, it holds that $t \in \mathcal{R}$.

If $t \in \|\Psi\|_{\mathcal{V}[X:=\mathcal{R}]}$, then by its definition, two cases arise. Either $t \in (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2)\|))$, or both $t \in \|T(\Phi_1)\|$ and $t \in \langle \cdot R \cdot \rangle\mathcal{R}$. We show that for both cases, $t \in \mathcal{R}$

In the case that $t \in (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2)\|))$, it immediately follows that $t \in \mathcal{R}$ by the definition of $\mathcal{R}$.

If $t \in \|T(\Phi_1)\|$ and $t \in \langle \cdot R \cdot \rangle\mathcal{R}$, the latter being equivalent to $t \in \{s \mid \exists u : s \xrightarrow{R} u \wedge u \in \mathcal{R}\}$, then, from the definition of $\mathcal{R}$ we can assume u is an element such that there exist $t_n, \ldots, t_0$ where $u = t_n \xrightarrow{R} t_{n-1} \xrightarrow{R} \ldots \xrightarrow{R} t_0$ and where $\forall 0 < i \leq n$, $t_i \in \|T(\Phi_1)\|$. But by $t \in \langle \cdot R \cdot \rangle\mathcal{R}$, the transition $t \xrightarrow{R} u$ must then exist, and since we know $t \in \|T(\Phi_1)\|$, there exist $t_{n+1}, \ldots, t_0$ such that $t = t_{n+1} \xrightarrow{R} t_n \xrightarrow{R} t_{n-1} \xrightarrow{R} \ldots \xrightarrow{R} t_0$ and $\forall 0 < i \leq n+1$, $t_i \in \|T(\Phi_1)\|$. From this it follows that state $t \in \mathcal{R}$, which is what we set out to prove.

($\mathcal{R} \subseteq \|\mu\mathcal{R}\|$)
We wish to show that for all states $t \in \mathcal{R}$, it holds that $t \in \|\mu\mathcal{R}\|$. As $\|\mu\mathcal{R}\|$ is equivalent to $\|\mu X.(\Psi)\|$, it is then sufficient to show that $t \in \|\mu^\alpha X.(\Psi)\|$ for some fixed point approximation α.

First note that by $t \in \mathcal{R}$, there exist $t_n, \ldots, t_0$ such that $t = t_n \xrightarrow{R} t_{n-1} \xrightarrow{R} \ldots \xrightarrow{R} t_0$ where $\forall 0 < i \leq n$, $t_i \in \|T(\Phi_1)\|$ and $t_0 \in (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2)\|))$.

We show that $t_n \in \|\mu^{n+1}X.(\Psi)\|$ by induction on n, the length of the transition sequence between t_n and t_0.

Base: $(i = 0)$ If the length of our transition sequences is 0, all states satisfy $t_0 \in (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2)\|))$. But the approximation $\|\mu^{0+1}X.(\Psi)\|$ covers precisely that, as:

$$\begin{aligned} t \in \|\mu^1 X.(\Psi)\| &\Leftrightarrow t \in \|\Psi\|_{\mathcal{V}[X=\mu^0 X.(\Psi)]} \\ &\Leftrightarrow t \in (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2)\|)) \cup (\|T(\Phi_1)\| \cap \langle \cdot R \cdot \rangle \emptyset) \\ &\Leftrightarrow t \in (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2)\|)) \end{aligned}$$

Step: $(i > 0)$ We assume $t_{i-1} \in \|\mu^i X.(\Psi)\|$, and set out to prove that $t_i \in \|\mu^{i+1}X.(\Psi)\|$.

We first expand $t_i \in \|\mu^{i+1}X.(\Psi)\|$:

$$\begin{aligned} t_i \in \|\mu^{i+1} X.(\Psi)\| &\Leftrightarrow t_i \in \|\Psi\|_{\mathcal{V}[X:=\|\mu^i X.(\Psi)\|]} \\ &\Leftrightarrow t_i \in (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2)\|)) \cup (\|T(\Phi_1)\| \cap \langle \cdot R \cdot \rangle \|\mu^i X.(\Psi)\|) \end{aligned}$$

Suppose $t_i \notin (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2)\|))$, it remains to show that $t_i \in \|T(\Phi_1)\|$ and $t_i \in \langle \cdot R \cdot \rangle \|\mu^i X.(\Psi)\|$. The former already holds as we assumed $t_i \in \mathcal{R}$. By the same assumption there exist $t_i, \ldots, t_0$ such that $t_i \xrightarrow{R} t_{i-1} \xrightarrow{R} \ldots \xrightarrow{R} t_0$. But then, there is a transition from t_i to t_{i-1}, and since we assumed by our induction hypothesis that $t_{i-1} \in \|\mu^i X.(\Psi)\|$, it must hold that $t_i \in \langle \cdot R \cdot \rangle \|\mu^i X.(\Psi)\|$, which finishes the step case.

By this induction proof we conclude all states $t \in \mathcal{R}$ are added in some approximation of $\|\mu X.(\Psi)\|$, hence all states $t \in \|\mu\mathcal{R}\|$.

This proves the second inclusion and finishes the proof. □

With Lemma 3.5 we conclude the section regarding the correct characterization of the fixed point formula in the surrounded operator. We now know that the above fixed point formula characterizes a set with a number of properties, which will be used in the correctness of our translation.

3.4 Correctness of translation

The most important result of this chapter is the proof that our translation T from Definition 3.3 is correct. More specifically, if formulae Φ is satisfied in a point x in model $\mathcal{M}$, then x satisfies the translated modal μ-calculus formulae $T(\Phi)$ in the LTS $\mathcal{L}$ associated with $\mathcal{M}$, and vice-versa. Without further ado:

Theorem 3.6. Let $\mathcal{M}$ be a closure space, let $\mathcal{L}$ be its corresponding LTS

Then, $\mathcal{M}, x \models \Phi \Leftrightarrow \mathcal{L}, x \models T(\Phi)$ for all points x and for every SLCS formulae Φ.

Proof. We prove this using induction on the structure of Φ, considering Definition 3.3:

ATOMIC PROPOSITION:

We derive $\mathcal{M}, x \models p \Leftrightarrow \mathcal{L}, x \models T(p)$ as follows:

($\Rightarrow$) Assume $\mathcal{M}, x \models p$ holds. Then, by Definition 2.5, $x \in \mathcal{V}(p)$. This means that by Definition 3.1, there exists a transition $x \xrightarrow{p} x$ in the LTS $\mathcal{L}$ corresponding to $\mathcal{M}$. Since the property *true* holds in all states, we have $\exists y.x \xrightarrow{p} y \wedge \mathcal{L}, y \models \mathit{true}$. From Definition 2.12 we then get $\mathcal{L}, x \models \langle p \rangle \mathit{true} = T(p)$, finishing the implication.
($\Leftarrow$) Assume $\mathcal{L}, x \models T(p) = \langle p \rangle \mathit{true}$ by Definition 2.12. Then, we know there exists a p-labelled transition from x. By Definition 3.1, the only p-transition in $\mathcal{L}$ is the transition $x \xrightarrow{p} x$ and by the same definition this implies $x \in \mathcal{V}(p)$. If this holds, Definition 2.5 shows us that $\mathcal{M}, x \models p$, proving the implication.

Take spatial logic formulae Φ_1 and Φ_2 such that $\mathcal{M}, x \models \Phi_1$ and $\mathcal{M}, x \models \Phi_2$. Then, we assume $\mathcal{L}, x \models T(\Phi_1)$ and $\mathcal{L}, x \models T(\Phi_2)$ hold as our induction hypothesis (IH).

NEGATION:

We derive $\mathcal{M}, x \models \neg\Phi_1 \Leftrightarrow \mathcal{L}, x \models T(\neg\Phi_1)$ as follows:

$$\begin{aligned}
\mathcal{M}, x \models \neg\Phi_1 &\Leftrightarrow \mathcal{M}, x \not\models \Phi_1 && \text{(by def: 2.5)}\\
&\Leftrightarrow \mathcal{L}, x \not\models T(\Phi_1) && \text{(by IH)}\\
&\Leftrightarrow \mathcal{L}, x \models \neg T(\Phi_1) && \text{(by def: 2.12)}\\
&\Leftrightarrow \mathcal{L}, x \models T(\neg\Phi_1) && \text{(by def: 3.3)}
\end{aligned}$$

CONJUNCTION:

We derive $\mathcal{M}, x \models \Phi_1 \wedge \Phi_2 \Leftrightarrow L, x \models T(\Phi_1 \wedge \Phi_2)$ as follows:

$$\begin{aligned}
\mathcal{M}, x \models \Phi_1 \wedge \Phi_2 &\Leftrightarrow \mathcal{M}, x \models \Phi_1 \text{ and } \mathcal{M}, x \models \Phi_2 && \text{(by def: 2.5)}\\
&\Leftrightarrow \mathcal{L}, x \models T(\Phi_1) \text{ and } \mathcal{L}, x \models T(\Phi_2) && \text{(by IH (2x))}\\
&\Leftrightarrow \mathcal{L}, x \models T(\Phi_1) \wedge T(\Phi_2) && \text{(by def: 2.12)}\\
&\Leftrightarrow \mathcal{L}, x \models T(\Phi_1 \wedge \Phi_2) && \text{(by def: 3.3)}
\end{aligned}$$

NEAR:

We derive $\mathcal{M}, x \models \mathcal{N}\ \Phi_1 \Leftrightarrow \mathcal{L}, x \models T(\mathcal{N}\ \Phi_1)$ as follows:

$$\begin{aligned}
\mathcal{M}, x \models \mathcal{N}\ \Phi_1 &\Leftrightarrow x \in \mathcal{C}_R(\{y \mid \mathcal{M}, y \models \Phi_1\}) && \text{(by def 2.5)}\\
&\Leftrightarrow \exists y : x \xrightarrow{R} y \wedge \mathcal{L}, y \models T(\Phi_1) && \text{(by def 3.1 and IH)}\\
&\Leftrightarrow \mathcal{L}, x \models \langle R \rangle T(\Phi_1) && \text{(by def: 2.12)}\\
&\Leftrightarrow \mathcal{L}, x \models T(\mathcal{N}\ \Phi_1) && \text{(by def: 3.3))}
\end{aligned}$$

SURROUNDED:

We derive $\mathcal{M}, x \models \Phi_1\ \mathcal{S}\ \Phi_2 \Leftrightarrow \mathcal{L}, x \models T(\Phi_1\ \mathcal{S}\ \Phi_2)$, proving both implications individually, as follows:

($\Rightarrow$) Assume $\Phi_1 \ \mathcal{S} \ \Phi_2$ is satisfied in a point x for a quasi-discrete closure space $\mathcal{M}$, then we aim to show its translation $T(\Phi_1 \ \mathcal{S} \ \Phi_2)$ is satisfied in x for the labelled transition system $\mathcal{L}$ that corresponds to $\mathcal{M}$.

We take from Definition 2.5 that, if $\mathcal{M}, x \models \Phi_1 \ \mathcal{S} \ \Phi_2$, then:

$$\begin{aligned}
&\mathcal{M}, x \models \Phi_1 \text{ and } \forall \pi, i : \\
&\qquad \pi(0) = x \text{ and } \mathcal{M}, \pi(i) \models \neg\Phi_1 \\
&\qquad \text{implies } \exists j \text{ such that } 0 < j \leq i \\
&\qquad\qquad \text{and } \mathcal{M}, \pi(j) \models \Phi_2
\end{aligned} \tag{1}$$

and show, following translation T in Definition 3.3, that then x must satisfy modal μ-formula:

$$\begin{aligned}
\mathcal{L}, x \models \ & T(\Phi_1) \wedge \neg\mu X.(\\
&\quad \neg(T(\Phi_1) \vee T(\Phi_2)) \ \vee \\
&\quad (T(\Phi_1) \wedge \langle R \rangle X) \\
&\quad)
\end{aligned} \tag{2}$$

which would imply that $\mathcal{L}, x \models T(\Phi_1 \ \mathcal{S} \ \Phi_2)$.

The modal μ-formula consists of a conjunction. The left hand side simply states that $\mathcal{L}, x \models T(\Phi_1)$, which follows immediately from (1) after applying the IH to $\mathcal{M}, x \models \Phi_1$. Hence, after applying Definition 2.12 to the negation sign before the fixed point, what remains to be shown is:

$$\begin{aligned}
\mathcal{L}, x \not\models \ & \mu X.(\\
&\quad \neg(T(\Phi_1) \vee T(\Phi_2)) \ \vee \\
&\quad (T(\Phi_1) \wedge \langle R \rangle X) \\
&\quad)
\end{aligned} \tag{3}$$

This is exactly the modal μ-formula defined by $\mu\mathcal{R}$ in Subsection 3.3. By Definition 2.11, it then follows that $x \notin \|\mu\mathcal{R}\|$. In Lemma 3.5, we have shown that $\|\mu\mathcal{R}\|$ is characterized by the set $\mathcal{R}$. Therefore, what remains to be shown is $x \notin \mathcal{R}$. We will use a proof by contradiction to achieve this; first assuming $x \in \mathcal{R}$, and then showing that the paths from x do not adhere to the surrounded property given in (1).

First, consider the paths from $x = \pi(0)$ to $\pi(i)$ defined in (1). Applying the IH to $\pi(0)$, and all points $\pi(i)$ and $\pi(j)$, we get $\mathcal{L}, \pi(0) \models T(\Phi_1)$; $\mathcal{L}, \pi(i) \models \neg T(\Phi_1)$ and $\mathcal{L}, \pi(j) \models T(\Phi_2)$. Then, Lemma 3.2 can be applied to the paths in our closure model from x_0 to x_j to x_i, meaning path indices $\pi(0), \pi(i), \pi(j)$ in $\mathcal{M}$ correspond to points x_0, x_i, x_j in $\mathcal{L}$. This implies the following transition sequences originating from x_0 must exist in $\mathcal{L}$:

$$\begin{aligned}
\forall i : x = x_0 \xrightarrow{R} \ldots \xrightarrow{R} x_i \text{ and } \mathcal{L}, x_i \models \neg T(\Phi_1) \\
\text{implies } \exists j \text{ such that } 0 < j \leq i \\
\text{and } \mathcal{L}, x_j \models T(\Phi_2)
\end{aligned} \tag{4}$$

In other words, for all sequences starting in x, before reaching a point satisfying $\neg T(\Phi_1)$, we must pass a point satisfying $T(\Phi_2)$ first.

Recalling the definition of $\mathcal{R}$ from Section 3.3, we know:

$$\begin{aligned}
x \in \{x \mid \exists i : x = x_0 \xrightarrow{R} x_1 \xrightarrow{R} \ldots \xrightarrow{R} x_{i-1} \xrightarrow{R} x_i \wedge \forall j.\, 0 \leq j < i : x_j \in \|T(\Phi_1)\| \\
\wedge\, x_i \in (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2)\|))\}
\end{aligned}$$

Note that some indices of this definition have been renamed for the sake of clarity.

We now prove that, if $x \in \mathcal{R}$, that this contradicts with the sequences in 4, no matter what value i takes. Two cases are to be considered:

If $i = 0$, then $x_0 \in (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2\|))$, meaning $\mathcal{L}, x_0 \models \neg(T(\Phi_1) \vee T(\Phi_2))$ by Definition 2.11 and Definition 2.12. This contradicts with the transition sequences in (4), as $\mathcal{L}, x_0 \models T(\Phi_1)$ must hold.

If $i > 0$, then there exist $x_0, \ldots, x_i$ such that $x = x_0 \xrightarrow{R} x_1 \xrightarrow{R} \ldots \xrightarrow{R} x_{i-1} \xrightarrow{R} x_i$ and $\forall j.0 \leq j < i : x_j \in \|T(\Phi_1)\|$ and $x_i \in (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2\|))$. Without loss of generality, we can choose the smallest value of i such that $\forall j < i : \mathcal{L}, x_j \models T(\Phi_1)$. As x_i satisfies neither $T(\Phi_1)$ nor $T(\Phi_2)$, we know $\mathcal{L}, x_i \models \neg T(\Phi_1)$. If this is the case, there must exist some x_j on the sequence from x_0 to x_i such that $\mathcal{L}, x_j \models T(\Phi_2)$. However, from the definition of $\mathcal{R}$, we know $\forall j.\, 0 \leq j < i : \mathcal{L}, x_j \models T(\Phi_1)$, and $\mathcal{L}, x_i \models \neg T(\Phi_2)$. Hence, there exists no x_j such that $\mathcal{L}, x_j \models T(\Phi_2)$, which shows the contradiction with the transition sequences in (4).

This finishes the proof by contradiction, showing that $x \notin \|\mu\mathcal{R}\|$. Since this is what we set out to prove, this finishes the implication.

($\Leftarrow$) The implication from right to left will follow some parts of the proof above. We aim to show that, assuming $\mathcal{L}, x \models T(\Phi_1 \ \mathcal{S} \ \Phi_2)$, that then $\mathcal{M}, x \models \Phi_1 \ \mathcal{S} \ \Phi_2$.

Following translation T in Definition 3.3, if $\mathcal{L}, x \models T(\Phi_1 \ \mathcal{S} \ \Phi_2)$ holds, this is equivalent to:

$$\begin{aligned} \mathcal{L}, x \models \ & T(\Phi_1) \wedge \neg\mu X.(\\ & \quad \neg(T(\Phi_1) \vee T(\Phi_2)) \ \vee \\ & \quad (T(\Phi_1) \wedge \langle R \rangle X) \\ &) \end{aligned} \tag{5}$$

and we aim to show that by Definition 2.5, this satisfies:

$$\begin{aligned} & \mathcal{M}, x \models \Phi_1 \text{ and } \forall \pi, i : \\ & \quad \pi(0) = x \text{ and } \mathcal{M}, \pi(i) \models \neg\Phi_1 \\ & \quad \text{implies } \exists j \text{ such that } 0 < j \leq i \\ & \quad\quad \text{and } \mathcal{M}, \pi(j) \models \Phi_2 \end{aligned} \tag{6}$$

The left-hand side of the conjunction in (6) immediately follows from applying the IH to $\mathcal{L}, x \models T(\Phi_1)$, after which we achieve $\mathcal{M}, x \models \Phi_1$. What then remains to be shown is the following:

$$\begin{aligned} & \forall \pi, i : \pi(0) = x \text{ and } \mathcal{M}, \pi(i) \models \neg\Phi_1 \\ & \quad \text{implies } \exists j \text{ such that } 0 < j \leq i \\ & \quad\quad \text{and } \mathcal{M}, \pi(j) \models \Phi_2 \end{aligned} \tag{7}$$

As shown in the right implication proof at (4), by applying the IH and Lemma 3.2 to (7) we achieve:

$$\begin{aligned} & \forall i : x = x_0 \xrightarrow{R} \ldots \xrightarrow{R} x_i \text{ and } \mathcal{L}, x_i \models \neg T(\Phi_1) \\ & \quad \text{implies } \exists j \text{ such that } 0 < j \leq i \\ & \quad\quad \text{and } \mathcal{L}, x_j \models T(\Phi_2) \end{aligned} \tag{8}$$

Furthermore, as the left-hand side of the conjunction formula in (5) has been shown, what remains is the negation of the fixed point in the right-hand side. From the definition of $\mu\mathcal{R}$ in

Section 3.3 this is equivalent to $\mathcal{L}, x \not\models \mu\mathcal{R}$ after applying the negation implication in Definition 2.12. It follows by Definition 2.11 that $x \notin \|\mu\mathcal{R}\|$. By Lemma 3.5, this implies $x \notin \mathcal{R}$.

We thus wish to show that (8) follows from $x \notin \mathcal{R}$. This is achieved by proving the contrapositive, we assume some arbitrary sequence starting in x does not satisfy (8), and show that this implies x must be an element of $\mathcal{R}$.

Assume $\forall i : x = x_0 \xrightarrow{R} ... \xrightarrow{R} x_i$ and $\mathcal{L}, x_i \models \neg T(\Phi_1)$, but there exists no j such that $0 < j \leq i$ and $\mathcal{L}, x_j \models T(\Phi_2)$. Then, $\mathcal{L}, x_i \models \neg T(\Phi_2)$. From this and our assumption $\mathcal{L}, x_i \models \neg T(\Phi_1)$, we know $\mathcal{L}, x_i \models \neg(T(\Phi_1) \vee T(\Phi_2))$ by applying Definition 2.12.

By definition we know $\mathcal{L}, x_0 \models T(\Phi_1)$. Without loss of generality, we may then choose i such that $\forall j. 0 \leq j < i : \mathcal{L}, x_j \models T(\Phi_1)$.

If $x \in \mathcal{R}$, the following must hold for x by the definition of $\mathcal{R}$ in Section 3.3:

$$\begin{aligned} x \in \{x \mid \exists i : x = x_0 \xrightarrow{R} x_1 \xrightarrow{R} ... \xrightarrow{R} x_{i-1} \xrightarrow{R} x_i \wedge \forall j.\, 0 \leq j < i : x_j \in \|T(\Phi_1)\| \\ \wedge\, x_i \in (S - (\|T(\Phi_1)\| \cup \|T(\Phi_2)\|))\} \end{aligned} \tag{9}$$

In other words, there is some i that neither satisfies $T(\Phi_1)$ nor $T(\Phi_2)$ that can be reached from x using only transitions to states satisfying $T(\Phi_1)$. We can rewrite this to $\mathcal{L}, x_i \models \neg(T(\Phi_1) \vee T(\Phi_2))$ and $\forall j.\, 0 \leq j < i : \mathcal{L}, x_j \models T(\Phi_1)$ by Definition 2.12. However, the transition sequence that arose from our assumption exactly satisfies these conditions, and therefore $x \in \mathcal{R}$. Since this proves the contrapositive, we know (8) follows from $x \notin \mathcal{R}$, which is what remained to be proven, and hence proves the implication.

As we have covered all cases of our translation, this shows $\mathcal{M}, x \models \Phi \Leftrightarrow \mathcal{L}, x \models T(\Phi)$ for all points x and every SLCS formula Φ. □

Chapter 4

Verifying spatial properties using mCRL2

The mCRL2 toolset has been in development for several decades [15]. It implements even more decades of theory, in order to verify behaviour of concurrent and distributed systems. These systems consist of processes that are written down in the specification language of mCRL2, and are verified using the modal μ-calculus. Typical questions that mCRL2 can answer are: 'is my system deadlock-free?' or: 'will every message that my buffer reads eventually be sent?'. These properties are verified on the initial state of the system, as this allows for optimizations during the verification process.

This chapter will describe the implementation that allows us to verify spatial properties using mCRL2. To aid the understanding of this chapter, the full workflow is given in Figure 3, with the data structures or concepts in blue, and the operations on them in orange. The scope of mCRL2 with its relevant tools is shown. A model-checking problem in mCRL2 is tackled by first creating a Linear Process Specification (LPS) from an mCRL2 specification using the tool `mcrl22lps`. Then, a Parameterised Boolean Equation System (PBES) [14] is constructed from the LPS and a μ-calculus formula encoding some property, by executing the tool `lps2pbes`. Running `pbessolve` on this PBES returns whether or not the property is satisfied in the initial state of the system.

Formalizing the goal of this chapter, we wish to use the mCRL2 toolset to verify all pixels in an image I that satisfy a spatial property Φ. Because the types of systems that mCRL2 classically verifies are quite different from images, we identified three main components outside the scope of mCRL2 that are needed to perform spatial model checking:

First, we require an mCRL2 specification S that represents the image I. In Definition 3.1 we have seen that we can define a Labelled Transition System (LTS) from a closure model of an image, which helps facilitate this transformation. As every mCRL2 specification has a corresponding LTS, it is sufficient to create a specification whose implementation satisfies all aspects of Lemma 3.1.

Furthermore, a spatial property Φ must be encoded as a μ-calculus formula $T(\Phi)$, such that mCRL2 is able to verify it on a specification. We have shown in Chapter 3 that this is possible; Theorem 3.6 proves every SLCS formula can be translated to a correct μ-calculus formula using translation T.

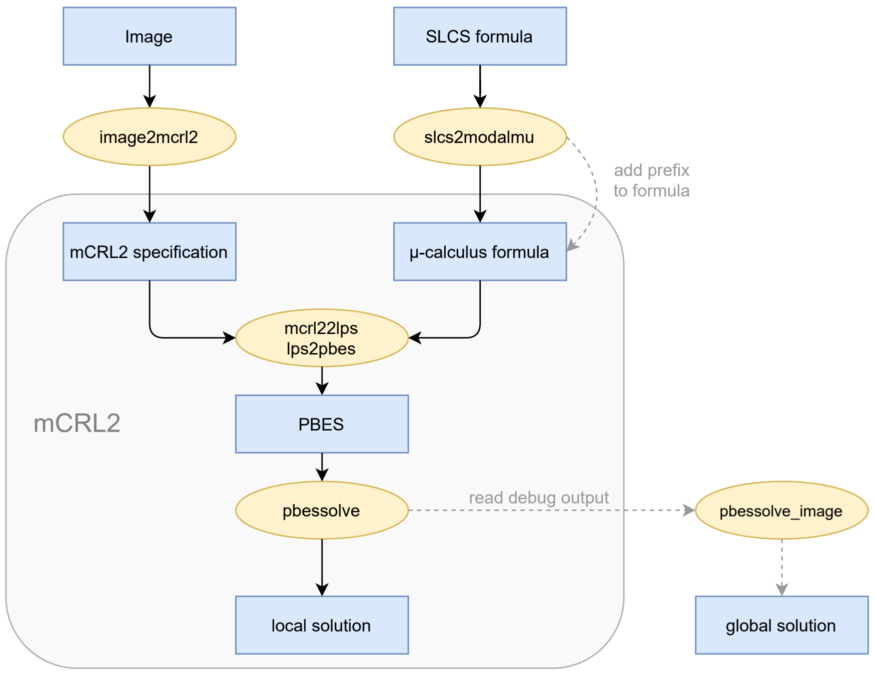


Figure 3: Diagram showing the process of spatial model checking in mCRL2

Finally, there is a discrepancy between the type of solution mCRL2 provides, and the one we wish to find. We mentioned that mCRL2 only verifies properties on the initial state of the system, and this must be specifically encoded in the mCRL2 specification. Therefore, if we were to directly verify $T(\Phi)$ on our specification S, we only know whether the initial state satisfies spatial property Φ. In other words, it solves the *local* model checking problem of a system. This is somewhat problematic for our application. Conventionally, spatial model checking answers the *global* model checking problem; we wish to return every pixel of an image that satisfies a given property, not just a single one. The dotted grey arrows in Figure 3 specify the additions that allow us to get a global solution using the mCRL2 toolset. We found that the μ-calculus formula $T(\Phi)$ can be adapted in such a way that, when the toolset is invoked, a solution to the *global* model checking problem can be extracted from the output stream of `pbessolve`.

This workflow also contains a number of new scripts outside the mCRL2 scope, that together form the implementation. These scripts will be explained throughout this chapter and can be found in the `spatial_mcrl2` GitHub repository[1].

This chapter is therefore outlined as follows: we first focus on solving the local model checking problem in Section 4.1. This is done by describing the methods that are needed to utilize the mCRL2 toolset to verify formulae on a single pixel of an image. Then, we explain the additions needed such that the toolset can be used to obtain a global solution in Section 4.2. The chapter continues by showing some extensions to the obtained mCRL2 specification to allow for more flexible analysis in Section 4.3, and finishes with some experiments and their results in Section 4.4 to evaluate the speed of this approach.

4.1 The local model checking problem

The mCRL2 toolset was built and optimized to verify properties on a single, user-specified state of the model. In an image model, this would be equivalent to verifying spatial properties on a single pixel. While this is not what spatial model checking aims to do, it is helpful to understand the process of obtaining a local solution, before moving to the additions that lead to a global solution.

This section describes two steps that are required to solve the local model checking problem. We cover how to express an image as an mCRL2 process specification, and how our translation from the previous chapter can be used to verify spatial properties on a single pixel. We shall mention the implementation of corresponding scripts `image2mcrl2` and `slcs2modalmu` later, as they will include the specification extensions covered in 4.3.

Let us start with the mCRL2 process specification that represents an image. We first denote a fragment of the syntax of an mCRL2 process, stating only the parts that are relevant to this thesis. For the full process specification, the reader is referred to [15].

A process P is syntactically defined by the following grammar:

$$P = a \mid P + P \mid P \cdot P \mid exp \rightarrow P \mid (P)$$

Here, a is an action that can take some data type, and exp is some Boolean expression. The $+$ operator expresses choice, while the $\cdot$ operator denotes sequential composition. The $\rightarrow$ operator signifies a conditional statement; only if Boolean expression exp is true, process P can be executed. Parentheses are useful to avoid ambiguity in the precedence of operators.

Our process specification should, when converted to a LTS, correspond to the closure model of an image. Recall that the closure model of an image is defined by $\mathcal{M} = ((X, \mathcal{C}_R), \mathcal{V})$. From Definition 3.1, we conclude our LTS must have a state for each pixel $x \in X$, appropriate R-labelled transitions between neighbouring pixels, and, for each pixel, have a self loop containing its atomic predicates. In this section we shall sometimes use the terms 'image data' or 'attributes' in favour of atomic predicates.

An image should, in this context, be seen as a 2D grid of pixels where each pixel contains data. We can specify a grid `g` in mCRL2 by defining a list of lists, where the outer list and inner lists represent the column and row index of the grid respectively. Note that this structure only captures the image data and position on the grid, much like a lookup table.

Now consider closure operator $\mathcal{C}_R$. According to Definition 3.1, in an LTS corresponding to the closure model, each pixel should have an R-labelled transition to itself and the pixels it shares an horizontal or vertical edge with. We can create these pixel states and their transitions with the process `Grid` :

```
proc
    Grid(x:Int, y: Int) =
        report(g.y.x) . Grid(x,y)
        + R . Grid(x,y)                          % reflexive step
        + (x != 0)        -> R . Grid(x-1, y)    % left step
        + (x != size_x)   -> R . Grid(x+1, y)    % right step
        + (y != 0)        -> R . Grid(x, y-1)    % down step
        + (y != size_y)   -> R . Grid(x, y+1)    % up step
    ;
```

```
init
    allow ({R, report},
        comm({}, Grid(start_x, start_y))
    );
```

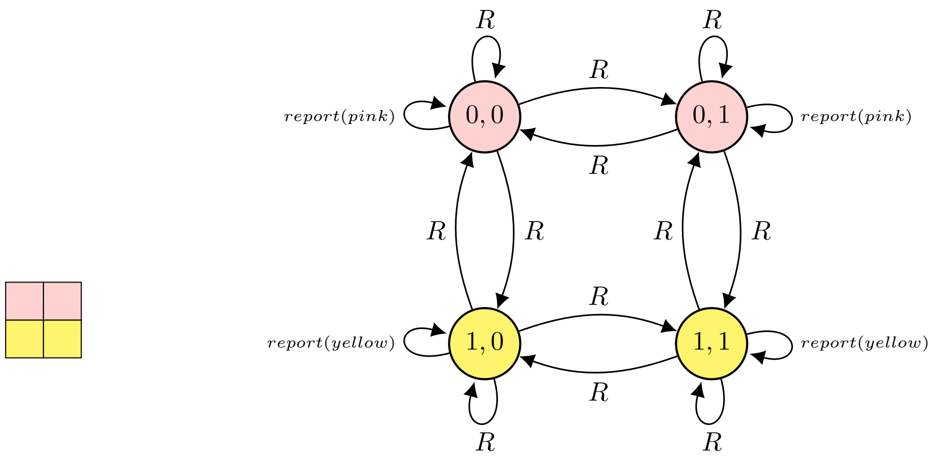
From the snippet above, we see that process **Grid** takes two arguments. Integers x and y represent the horizontal and vertical position on the grid. This process has, at all times, a choice between 6 options, and then recurses, possibly with a new grid position. We briefly ignore the first option for now. The other options all take an R-action representing the closure operation, and either end up in the same, or an adjacent position on the grid. For each R-transition to a neighbouring pixel, conditional checks should be applied that ensure the movement is valid, as we may not end up outside the boundaries of our image. We can prevent invalid moves from happening by, for example, not allowing a left step if we are at the left border, hence the conditional `(x != 0)`. The other boundary checks then follow. This leaves us with exactly one state for each valid x and y position, and all the required R actions for a pixel x.

Let us now return to the first option of process **Grid**. For each pixel, the process generates a **report** action that looks up the atomic predicates of pixel, which can be found by accessing the structure that contains its data. The atomic predicates at position (x, y) are accessed using '**g.y.x**', where **g** is the data structure of the grid. This satisfies our final property, because each pixel x now has a self-loop containing its data.

This specification now satisfies all properties of Definition 3.1. The snippet above also shows how to initialize the grid process. We only allow **R** and **report** actions to occur, there is no communication with other components, and the grid is initialized with starting x- and y positions. The following example will aim to explain what the mCRL2 specification of a small image looks like

Example 4.1. Consider the image **small** consisting of 4 pixels in a 2 by 2 grid. Each pixel of the image takes either the value yellow or pink, modelled by the enumerative data type **Colour**. We then map **small** to a grid and define its data structure. The image in question is depicted in Figure 4a, and its generated state space representation that follows from the **Grid** process is shown in Figure 4b. Notice that R-actions leading outside the grid boundaries are not generated by this model. Actions representing atomic predicates follow from the outcome of **small.y.x** in its respective function **report(small.y.x)**. States are filled with their assigned colour and coordinates for the sake of clarity.

```
sort
    Grid = List(List(Pixel))
    Pixel = Colour
    Colour = struct pink | yellow
map small : Grid
eqn small = [
        [pink, pink],
        [yellow, yellow]
    ]
```



(a) Visualisation of input grid

(b) State space representation

Figure 4: Example of a small grid.

◁

With the mCRL2 specification in place, we finish this section by showing how to implement the translation from Chapter 3 in mCRL2. There are some notational differences; conjunction and disjunction become their programming language counterparts `&&` and `||` respectively, while fixed points are written as `mu` and `nu`.

The translation can otherwise directly be implemented in mCRL2, bar one change specific to this implementation. We saw in Definition 3.3 that the satisfaction of an atomic predicate p in a state x should be translated to $\langle p \rangle \mathit{true}$, or in other words, there exists a transition p from state x. In the mCRL2 specification we created, however, there are neither separate actions nor transitions for each possible atomic predicate. Instead we have a single `report` action whose contents represent the atomic predicates of x. Therefore, to verify whether a pixel satisfies p, the lookup table access '`g.y.x`' in the report function must return p. In other words, every instance of $\langle p \rangle \mathit{true}$ must be replaced with $\langle \texttt{report(}p\texttt{)} \rangle \mathit{true}$.

Example 4.2. Consider again the image `small` from Example 4.1. Two examples of that we may verify on this image are:

A pixel is yellow:

SLCS	yellow
μ-calculus	$\langle \texttt{report(}\mathit{yellow}\texttt{)} \rangle \mathit{true}$

A pixel is near a yellow pixel, and is pink:

SLCS	$\mathcal{N}\ \mathit{yellow} \wedge \mathit{pink}$
μ-calculus	$(\langle R \rangle \langle \texttt{report(}\mathit{yellow}\texttt{)} \rangle \mathit{true})$ && $\langle \texttt{report(}\mathit{pink}\texttt{)} \rangle \mathit{true}$

◁

This concludes the section that covered the process of verifying spatial properties locally in mCRL2. Notice that the structure defining a single pixel is currently an enumerative data type, which clearly has a limited amount of use and flexibility. Section 4.3 will extend this model such that multiple and more complex attributes, like numerical values, can be asserted.

4.2 The global model checking problem

So far, we were able to correctly verify spatial properties on single pixels. In image analysis, however, it is much more useful to provide a solution to the image as a whole. Ideally we 'mark' every pixel in an image that satisfies a property.

The modal μ-calculus is suitable to answer these kind of questions. In particular, the denotational semantics from Definition 2.10 already considered sets of states, but the implementation of mCRL2 (rightfully) reduces the problem complexity by only verifying the μ-calculus formula for the initial state. There still is a naïve way of obtaining a global solution. We could call upon a new instance of the mCRL2 specification in Section 4.1 for each individual pixel, where only the coordinate parameters denoting the initial position is changed. Clearly, this is not a scalable solution; any real-world image would require thousands, if not millions of iterations.

This section therefore will present an alternative way to find a global solution. Due to the manner in which mCRL2 toolset calculates solutions to fixed points, a few tricks can be applied to ensure the solution to every pixel is latently available during the verification of a μ-calculus formula on a single pixel. We will first go into more depth on how mCRL2 verifies model checking problems in Section 4.2.1. Then, Section 4.2.2 elaborates what changes to the μ-calculus formulae need to be made in order to extract the solution of each individual pixel.

4.2.1 How mCRL2 verifies the image model

Previously, we mentioned what mCRL2 tools are often used to solve model checking problems. Figure 3 shows that the tools `mcrl22lps` and `lps2pbes` are used to construct a *Parameterized Boolean Equation System (PBES)* from a mCRL2 specification and a μ-calculus formula. In this section we show why it is not possible to extract a global solution in mCRL2, when we give it a μ-calculus formula constructed from our translation in Chapter 3. We do this by taking a look at `pbessolve`, which, as the name suggests, partially solves a PBES to verify the initial state of the mCRL2 specification.

A PBES is a sequence of equations taking the shape of $\sigma X(d_1 : D_1, \ldots, d_n : D_n) = \Phi$, with $\sigma = \{\mu, \nu\}$ being either fixed point symbol, predicate variable X, data variables d_i of type D_i, and Φ an action formula.

Given an mCRL2 specification and a μ-calculus formula, the mCRL2 toolset converts the specification to a Linear Process Specification (LPS) and then creates a PBES from the LPS and the formula. Its solution is an *environment*, containing the satisfiability of the μ-calculus formula for each possible combination of data variable values. As these values could theoretically be numerals going to infinity, obtaining the complete environment is often very slow. However, since mCRL2 knows the initial process parameters, the tool `pbessolve` can find a partial environment by instantiating a PBES to a *Boolean Equation System* (BES) and inserting the reported BES equations in a so-called structure graph.

A BES is a sequence of equations $(\sigma_1 X_1 = \Phi_1) \ldots (\sigma_1 X_n = \Phi_n)$. Each equation $(\sigma_i X_i = \Phi_i)$ represents the solution to one fixed point in the corresponding PBES for specific data variables. Instantiating is done by generating fresh equations whenever the solution to one BES equation requires the evaluation of a (possibly new) fixed point with new data variables. This is repeated until every right hand side consists of either existing equations, or ones that evaluate to true or false.

In our case, the only relevant variables in the mCRL2 image specification are the integer pixel coordinates defined by the process `Grid(x:Int, y:Int)`. The mCRL2 specification requires an initial state to be given. We therefore know the starting coordinates and thus the first BES equation that is generated. If this initial state is the coordinate $(0, 0)$, the evaluation of BES equation $(\sigma X(0, 0) = \dots)$ is the solution to this coordinate for fixed point X.

In the case where μ-calculus formulae do not depend on other coordinates, like atomic predicates, solving the PBES is trivial; its solution can immediately be deduced. The evaluation of properties like 'surrounded', however, may depend on many other coordinates due to the fixed point it introduces. Clearly, since our image has a fixed size, at some point instantiation will halt as there are no new coordinates to be discovered.

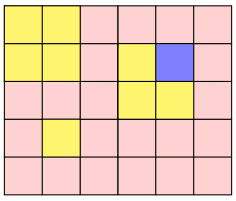
A structure graph is then built from the generated BES equations. Vertices are created for each BES equation, and edges between two vertices X and Y exist if Y appears in the BES equation associated to X. The solution of a BES equation is captured in a decoration $d \in \{true, false, conjunction, disjunction\}$ and assigned to its corresponding vertex. In the case this equation does not depend on other positions or nested fixed points and is therefore solvable, the decoration will be its Boolean outcome. Otherwise, depending on the fixed point formula, its decoration must be conjunction or disjunction. If the solution of X only depends on a single BES equation, a decoration is not assigned.

This structure graph is a representation of a parity game. These two-player games are the most common reduction from a μ-calculus formula and specification, as parity games are expressive enough to handle formulae with (nested) fixed points. However, we do not consider all vertices of the structure graph, as we already know the answer to BES equations decorated with true and false. Furthermore, an important observation is made. Many equations can be solved without the need for a parity game solver by using so-called attractor sets. Namely, all vertices in the structure graph decorated with *conjunction* that have an edge to a vertex *false* are immediately also false, and likewise for equations decorated with *disjunction* and *true* respectively.

If at this point there are still unsolved BES equations, a parity game is constructed from the remaining vertices, and is solved using Zielonka's algorithm [18]. These two-player games are the most common reduction from a μ-calculus formula and specification, as parity games are expressive enough to handle formulae with (nested) fixed points. The algorithm returns a partition of the vertices into two sets: they contain the BES equations evaluating to *true* and *false* respectively. As these sets contain all the remaining equations, this finishes the solving process; the solution to the very first generated equation is reported.

To find a solution for the initial state of the specification, this approach is ideal. By only using a minimal amount of information to verify a property a lot of computation time is saved. However, these optimizations get in the way of a global solution. We finish this section with Example 4.3, which aims to highlight the key steps in verifying an SLCS property on an image and why the entire image is often not considered.

Example 4.3. Consider the image from Example 2.6:



We wish to verify SLCS formula `yellow S pink` on the entire picture. This has the associated μ-calculus formula:

$$\begin{aligned}(yellow\ \mathcal{S}\ pink) = \; & yellow \wedge \neg\mu Y.(\\ & \quad \neg(yellow \vee pink) \vee \\ & \quad (yellow \wedge \langle R\rangle Y) \\ &)\end{aligned}$$

Because the formula does not start with a fixed point and therefore does not adhere to the PBES format, a dummy fixed point νX is generated around this formula. The R-action is expanded to all possible pixels a current position can move to, as followed from the `Grid` process in Section 4.1. The resulting PBES is given below, where the `report` actions for atomic predicates are left out for readability purposes:

$$\begin{aligned}(\nu X(x : Int, y : Int) = \; & yellow \wedge \neg Y(x, y)) \\ (\mu Y(x : Int, y : Int) = \; & \neg(yellow \vee pink) \ \vee (\\ & \quad (yellow \wedge (Y(x, y) \vee \\ & \quad (x \neq 0) \Rightarrow Y(x-1, y) \vee \\ & \quad (x \neq 6) \Rightarrow Y(x+1, y) \vee \\ & \quad (y \neq 0) \Rightarrow Y(x, y-1) \vee \\ & \quad (y \neq 5) \Rightarrow Y(x, y+1)) \\ &)\end{aligned}$$

The BES from this PBES is then instantiated. Suppose our starting position from the mCRL2 initialization parameters is the top left coordinate $(0, 0)$, then the first considered instance of the PBES is $\nu X(0, 0)$. Its evaluation depends on whether this coordinate is yellow, which it is, and the negation of fixed point Y with the given coordinates. Hence the first generated BES equation is $\nu X(0, 0) = \neg\mu Y(0, 0)$, and we continue exploration by finding the solution to $Y(0, 0)$.

As the top left pixel is yellow, the first part of the equation is satisfied. What remains is the possible movements from this coordinates, which arose from the grid specification and is specified in this PBES. As both x and y coordinates are 0, the generated BES equation becomes $\mu Y(0, 0) = \mu Y(0, 0) \wedge \mu Y(0, 1) \wedge \mu Y(1, 0)$. This exploration then continues; as $\mu Y(0, 0)$ is already explored, we process $Y(0, 1)$ and subsequently $\mu Y(1, 0)$. Important to observe is that exploration can be halted early; the pixel at position $(0, 2)$ is pink, hence $\mu Y(0, 2)$ is *false* and no further instantiating past this equation is necessary. This eventually leads to the following (shortened) BES:

$$
\begin{aligned}
&(\nu X(0,0) = \neg \mu Y(0,0))\\
&(\mu Y(0,0) = \mu Y(0,0) \vee \mu Y(0,1) \vee \mu Y(1,0))\\
&(\mu Y(0,1) = \mu Y(0,1) \vee \mu Y(0,0) \vee \mu Y(0,2) \vee \mu Y(1,1))\\
&\qquad\qquad \dots\\
&(\mu Y(1,1) = \mu Y(0,1) \vee \mu Y(1,0) \vee \mu Y(1,1) \vee \mu Y(1,2) \vee \mu Y(2,1))\\
&(\mu Y(1,2) = false)\\
&(\mu Y(2,1) = false)
\end{aligned}
$$

During exploration a structure graph is filled accordingly. As every pink pixel causes the $\mu Y(x, y)$ formula to be false, instantiating past these pixels is not needed. Apparently, information of only a small set of pixels was sufficient to determine that a solution can be found. We see the solutions to BES equations for the group of yellow pixels all depend on each other and the attractor sets of pink pixels do not include yellow ones. A parity game is therefore constructed for the remaining equation, and the solution to $Y(0,0)$ then appears to be $false$. This implies $X(0,0)$ is $true$, which is what we expected and set out to verify. $\triangleleft$

4.2.2 A prefix to the μ-calculus formulae

In the previous section it was shown that due to optimizations in the instantiation of a PBES, verifying translated SLCS properties `pbessolve` will not provide a global solution. Often, the evaluation of an initial state can be found without generating solutions for other coordinates, and we cannot pass multiple initial states to mCRL2. To work around this restriction, we extend μ-calculus with a prefix. This section explains what changes need to be made to the μ-calculus formulae to ensure `pbessolve` does consider every pixel.

Our first attempt involved stating that 'for all reachable paths formula Φ holds'; this would presumably lead to Φ being verified for each pixel during execution. This property can be encoded as:

$$\nu X.(\Phi \wedge [true]X)$$

In mCRL2, a shorthand for this is given in the shape of regular formula:

$$[true^*]\Phi$$

Resulting in PBES:

$$\nu X(x : Int, y : Int) = (\Phi \wedge (X(x, y) \wedge X(x + 1, y) \wedge ...)$$

Unfortunately, this will not yield a global solution yet. Due to the way BES equations are generated, if the first pixel we analyse contradicts with Φ, we already know that 'for all reachable paths' Φ is not satisfied, and no other pixels are evaluated. Similarly, an alternative regular formula $\langle true^*\rangle\Phi$ denoting 'there exists a reachable path to a point where Φ holds' is already satisfied the moment we find any pixel that satisfies Φ. Therefore, our next attempt involved a second addition to the μ-calculus formula. If we introduce a fresh fixed point variable between $[true^*]$ and Φ, BES equations will first be written for this fresh fixed point before Φ is evaluated. We therefore end up with our final prefix:

$$[true^*]\nu X.\Phi$$

A PBES for these formulae would look like:

$$\nu X(x : Int, y : Int) = (Y(x, y) \wedge (X(x, y) \wedge X(x + 1, y) \wedge ...)$$
$$\nu Y(x : Int, y : Int) = \Phi$$

This subtle change to the PBES avoids any optimizations, because `pbessolve` cannot immediately deduce the answer to $Y(x, y)$, representing Φ, even though there otherwise are no differences with the previous PBES. This results in `pbessolve` solving $X(x, y)$ for all reachable x and y. From our specification it follows that all coordinates are always reachable. This means the solution to $Y(x, y)$ will also be deduced for every coordinate. Also, with this addition, it does in fact not matter whether we write $[true^*]$ or $\langle true^*\rangle$; the fresh fixed point variable will have the same effect.

We now have all the ingredients to obtain a global solution, but `pbessolve` itself will still not give us one, because we gave it a different question to answer. Namely, only if Φ holds for all reachable pixels, the property holds in its entirety. Fortunately, by setting the `--debug` flag when running `pbessolve`, all BES equations are reported back to the user. Since we can easily deduce from our final PBES what fixed point represents Φ, the instantiated BES equations corresponding to this fixed point can be extracted from the debug output.

This is the essence of what our Python script `pbessolve_image` does. While the mCRL2 toolset solves the PBES of an edited μ-calculus formulae that contains the added prefix, this program keeps track of a separate structure for BES Equations, containing its identifier, the coordinates, structure graph decoration, and a Boolean stating whether this is a BES equation that encodes the solution to Φ. The decoration of these equations is used to store the solution. This is determined in one of three ways:

1. During execution, `pbessolve` immediately determines Φ is satisfied at a position x, y or not and gives it the decoration $true$ or $false$. No further action then needs to be taken.
2. When determining the attractor sets of BES equations with decoration $true$ or $false$, `pbessolve` reports a mapping that contains the strategy for BES equations whose solution can be deduced from attractor sets. This mapping is processed by changing the decorations accordingly.
3. For the remaining vertices in the structure graph, the parity game solver is needed, `pbessolve` reports the partitioning found by Zielonka's algorithm. We simply set its decoration according to the partitions.

Once `pbessolve` is finished, `pbessolve_image` simply reports the coordinates of BES equations that encode the solution to Φ, and whose decoration is $true$. We can now combine this method with the mCRL2 specification of an image and translated SLCS formula from Section 4.1, completing the workflow from Figure 3.

4.3 Extensions of the spatial verification process

In the previous sections of this chapter, we have described the complete process of verifying spatial properties on an image using the mCRL2 toolset. One may have noticed that the implementation is fairly inflexible. Most images do not have a single colour attribute for each pixel, but instead hold RGB data. This consists of three numerical values for the intensity of colours red, green and blue. While it is possible to assign colours to pixels as a preprocessing step, it is much more precise if we could directly process this type of image data. Buonamici et al [5] extended their image model in a similar way to facilitate this.

Section 4.3.1 will describe how the specification from Section 4.1 can be extended to support more complex data structures, the ability to make assertions on them, and what effect this has on the μ-calculus formula for atomic predicates. These extensions are subsequently covered in the description of `slcs2modalmu`, a simple lexer and parser implementing both the translation from Chapter 3 and the implementation-specific additions from this chapter, such that the resulting μ-calculus formulae are immediately verifiable using the mCRL2 toolset. This description can be found in Section 4.3.2.

4.3.1 Extending the mCRL2 image model with data

The mCRL2 toolset extends the modal μ-calculus described in Definition 2.8 with data to facilitate data-dependent processes. So far, our mCRL2 specifications have only handled a single, enumerative data type that consisted of a select number of colour options. This section will show an extension of this specification, to allow verifying pixels that have an arbitrary amount of atomic predicates with arbitrarily many different data types. We first explain how more complex data structures for pixels should be created, and then show how we can make assertions on these structures using the modal μ-calculus with data.

While mCRL2 provides numerous ways to create complex data types, a suitable approach for images is to use the mCRL2 construct 'Structured Sorts'. In the case of RGB values, we need to define three parameters for each pixel, that contain a numerical value between 0 and 255 specifying its intensity. This can be done as follows:

```
    sort
        Pixel = struct RGB(
            red:Intensity,
            green:Intensity,
            blue:Intensity,
        );
        Intensity = Int
```

A pixel now consists of a tuple named RGB with three parameters. Functions are created for each parameter by mCRL2, that when called upon return the colour intensity. They take the name of their respective parameter, meaning the red intensity value can be accessed by stating **`red(Pixel)`**. What remains is correctly represent pixel data in the image grid. If we take Example 4.1, and take RGB values for the colours pink and yellow, we can do this as follows:

```
    eqn small = [
            [RGB(255,255,0)   , RGB(255,255,0)   ],
            [RGB(255,192,203), RGB(255,192,203)]
        ]
```

Clearly, this approach allows for a lot more flexibility. As larger images have an enormous amount of different colours, individually specifying them is infeasible, and the verification process would become very cumbersome to write down.

In previous examples, we verified the existence of a **`report`** action whose function returned the pixel attribute. Now such an action consists of three different parameters which we want to individually access. It is possible to assert this more complex structure in μ-calculus formulae, for which it is helpful to understand the notion of an *action formula*, introduced to facilitate data-dependent processes in mCRL2. We refer the reader to work by Groote and Mousavi [15] for a full comprehensive description, or alternatively the documentation of mCRL2 [10]. We will only state a fragment containing the relevant parts to our specification.

An *Action Formula* is syntactically defined by the following grammar:

$$\alpha ::= t \mid true \mid false \mid \neg\alpha \mid \alpha_1 \wedge \alpha_2 \mid \alpha_1 \vee \alpha_2 \mid \exists d{:}D\ \alpha \mid \forall d{:}D\ \alpha$$

Where, t is a Boolean expression, and d a data identifier of D. Negation, con- and disjunction, and existential quantifiers behave as expected. Data types D can, for example, be some set of numbers, like $\mathbb{N}, \mathbb{Z}$. The mCRL2 toolset also allows D to be a predefined structure, like the RGB tuple we created before. The quantifications represent a set of actions. In the case of an existential quantifier, there is some data identifier d for which action formula α holds, while α must hold for all d in the case of an universal quantifier.

The existential quantifier in the μ-calculus formulae is helpful to make assertions on the data attributed to a pixel. We can write $\exists d : D.\langle \texttt{report}(d)\rangle true$ to state there exists some data identifier d of data type D for which the process can perform a $\texttt{report}(d)$ transition. This formula alone will naturally always be true as long as there exists a report action of type D, so we must restrict what report actions we allow using Boolean expressions on d.

We have already seen a simple application of this with the `Colour` structure in Example 4.1, where we could also write $\langle \texttt{report}(yellow)\rangle true$ as:

$$\exists\ c : Colour\ .\ \texttt{val}(c == yellow) \wedge \langle \texttt{report}(c)\rangle true$$

Note that in mCRL2, the keyword `val()` must be written around Boolean expressions in μ-calculus formulae for parsing reasons. A more generalized expression to evaluate pixel data with assertions $a_1 \dots a_n$ would now be:

$$\exists\ px : Pixel\ .\ \texttt{val}(a_1 \wedge a_2 \wedge \cdots \wedge a_n) \wedge \langle \texttt{report}(px)\rangle true$$

We essentially verify the existence of a report action that satisfies all assertions defined within the `val()` block. With this implementation verifying *ranges* of colours is now possible by letting an RGB value fall within certain intensities. As an example, we may check whether a pixel is 'approximately' pink by writing the μ-calculus formula:

```
exists px:Pixel.val(
    250 <= red(px)    && red(px)    <= 255 &&
    190 <= green(px) && green(px) <= 195 &&
    200 <= blue(px)   && blue(px)   <= 205
) && <report(px)>true
```

Since the medical field often deals with greyscale images, we also provide a second example. In this case, the three parameters that define an RGB value always take the same value. This allows for the following simplification of our Pixel structure and corresponding μ-calculus formula:

```
sort
    Pixel = Int
eqn small = [
    [125, 125],
    [190, 190]
]
```

```
    exists px:Pixel.val(
        120 <= px && px <= 130
    ) && <report(px)>true
```

This concludes the section about extending the mCRL2 specification of an image with more complex data types to make analysis more flexible. Python script `image2mcrl2` was created to automatically generate such a specification, given an image with numerical RGB data. This program is fairly straightforward, as the structure containing data of individual pixels is the only variable element. Other parts of the specification are static. An optional argument `--greyscale` can be passed for greyscale images, to create the simplified structure.

4.3.2 slcs2modalmu, a lexer and parser for translating SLCS formulae

This section briefly describes the implementation of a simple lexer and parser `slcs2modalmu` following the translation in Definiton 3.3, with optional arguments to facilitate the verification in mCRL2. The script was created in Python. It takes a `.slcs` file, verifies its syntax, and transforms the input to a modal μ-formula.

The required syntax of SLCS formulae in `slcs2modalmu` is stated below:

```
<FORM> ::=
<ATOM_PRED>              (atomic predciates, reserved names are below)
| (<FORM>)               (Subformula)
| ! <FORM>               (NOT operator)
| N <FORM>               (Near operator)
| <FORM> && <FORM>       (AND operator)
| <FORM> S <FORM>        (Surround operator)
```

The precedence of operations depends on their amount of arguments; operators taking one subformula have precedence over operators that take two. For example, the SLCS formula `N a S b` is parsed in the same manner as `(N a) S b`. Bracket usage is still encouraged to avoid unwanted behaviour. The usage of comments is possible; they should be preceded by a `%` character. Any further tokens after `%` are ignored by the parser until the next line of the input file. Comments will not reappear in the output file.

The SLCS formula is lexed using the python module `re` [11] to match regular expressions and subsequently tokenize the input. An atomic predicate must adhere to the regular expression `[a-ZA-Z0-9_]+`, meaning only groups of (capital) letters, numbers and underscores are matched. Capitals N and S are reserved for the near and surround operator in SLCS; they should be avoided as identifiers for atomic predicates, and spaces around these operators are therefore necessary.

If the optional argument `--mcrl2` is added, all instances of atomic predicates are changed to be compatible with the mCRL2 image model, as described in the previous section. Atomic predicates should be written down in `.slcs` as three comma-separated pairs of RGB intensity ranges surrounded by square brackets: `[`R_{min}`-`R_{max}`,`G_{min}`-`G_{max}`,`B_{min}`-`B_{max}`]`. In Section 4.3.1, we approximated pink, which would be written down here as
`[250-255,190-195,200-205]`. If the optional argument `--greyscale` is also set, only one pair

is needed; the intensity range would be represented by `[100-120]`. Spaces are allowed inside the square brackets, but hyphens and commas are necessary.

Parsing is done by creating an Abstract Syntax Tree from the tokens, which is then processed recursively to produce a modal μ-formula. Type checking is not implemented, but syntax errors will be raised if operators do not have the correct number of arguments or if atomic predicates have no binding operator. The result is saved to a `.mcf` file that is in accordance with the formatting of modal μ-formulae in mCRL2 [10].

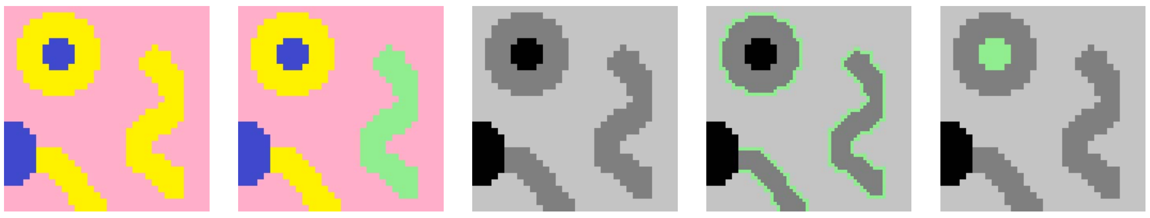
4.4 Experiments and Results

This section contains a small number of experiments that were performed to evaluate the speed of this spatial model checking approach. Experiments were run on an Intel i7-4710MQ CPU processor running at 2.50GHz.

To combine the findings of previous sections in this chapter, Python script `verify_image` was created. It takes a SLCS formula Φ and an image file, and performs the following steps.

1. Create a mCRL2 specification from an image file using script `image2mcrl2`.
2. Execute `slcs2modalmu` on Φ to obtain a μ-calculus formula verifiable by mCRL2.
3. Execute mCRL2 tools `mcrl2lps` and `lps2pbes`
4. Execute `pbessolve_image` to extract the coordinates of the pixels satisfying Φ.
5. Mark the image according to these coordinates.

Figure 5 shows the sample image on which three properties were verified, the greyscale alternative, and the respective images returned by `verify_image`. The property in Figure 5e should be considered a 'layered surrounded', in which we mark all blue pixels surrounded by yellow that themselves are surrounded by pink or blue pixels. This final disjunction is necessary to capture the correct pixels for the nested surround operator, as the yellow pixels cannot be exclusively surrounded by two separate colours.



(a) (b) (c) (d) (e)

Figure 5: Experiments ran with `verify_image`.
5a: the 64x64 sample image. **5b:** image 5a, with $y \ \mathcal{S} \ p$ marked in green,
5c: greyscale version of image 5a, **5d:** image 5c, with $y \wedge \ \mathcal{N} \ p$ marked
5e: image 5c, with $b \ \mathcal{S} \ (y \ \mathcal{S} \ p \vee b)$ marked

The experiments were timed for different resolutions of the same image. The results are shown in Table 4.1. Compared to the benchmarks by [5], the image size this approach can verify spatial properties on within 10 minutes is significantly lower. The simplifications a greyscale image introduces in both model and μ-calculus formula causes verification to be faster, yet grows equally quick for larger images. The colour intensity range only has have a marginal effect on speed; verifying $y \ \mathcal{S} \ p$ for exact RGB values versus a range of 10 only sees a 10% decrease in computation time.

The formula itself does have a noticeable impact. The near operator is especially costly, as every pixel could potentially be near a property and all its outgoing transitions must always be checked. Surrounded operators are more efficient if there are not many pixels satisfying the left-hand side in the first place; even the more complex layered surrounded is evaluated faster than the near operator. This does mean that the algorithms in mCRL2 work as intended. When solutions to many pixels can be found early in outer fixed points, verification speed increases dramatically.

	$y \wedge \mathcal{N} \ p$		$y \ \mathcal{S} \ p$		$b \ \mathcal{S} \ (y \ \mathcal{S} \ p \vee b)$	
Resolution	RGB	Grey	RGB	Grey	RGB	Grey
32x32	18.80	2.75	3.99	1.74	5.16	1.77
64x64	304.64	34.39	23.78	14.03	25.84	11.34
128x128	>600	545.40	201.76	146.29	184.64	128.48
256x256	>600	>600	>600	>600	>600	>600

Table 4.1: Experiment results (in seconds)

The overhead that `verify_image` introduces is significant. The computation time of script `image2mcrl2` scales linearly with the amount of pixels, as the only variable element in the mCRL2 specification is the table containing RGB data of each pixel. Script `slcs2modalmu` has a negligible execution time. Most extra work comes from parsing the debug output of `pbessolve` in script `pbessolve_image`. Its overhead grows with complexity $O(n * f)$, with n being the amount of pixels, and f the amount of fixed points. This is because `pbessolve` generates at most one BES equation per pixel for every fixed point, and every equation only needs to be parsed once using regular expressions. Additionally, the decoration of an equation is updated from either the mapping or the parity game solver at most once. The total overhead does not seem to grow faster than the work done by mCRL2, as simply running `pbessolve` without the `--debug` flag and with no parsing, cuts the computation time roughly in half, independent of image size and formula complexity. We therefore expect that, while an implementation directly built in the mCRL2 toolset will save time, evaluating larger images will remain costly.

Chapter 5

Future work and conclusion

This thesis investigated a new approach to spatial model checking, in which we used the mCRL2 toolset for analysing concurrent systems to verify spatial properties on images. Spatial model checking is a promising and relatively new field in Computer Science has applications in the medical field, and one of its main advantages is the ability to make the process of medical imaging, often performed by machine learning applications, more explainable and intelligible for specialists.

By creating both a transformation between the underlying models of SLCS and the modal μ-calculus, and a correct translation between their logics, we were able to create generalized mCRL2 specifications and μ-calculus formulae that verify spatial properties. By adding a prefix to the μ-calculus formulae, the solution for every pixel could be extracted from the intermediate computations by mCRL2. Also, a proof of concept implementation was created that reads image data, creates an mCRL2 specification from, translates SLCS- to μ-calculus formulae, runs some tools in the mCRL2 toolset and finally extracts the relevant data to mark the original image accordingly.

While our implementation was successful, it does not outperform existing toolsets. We did not expect this approach to be faster, since dedicated toolsets [6] were specifically built for topological applications. A direct implementation in mCRL2 remains as future work, which would remove significant overhead from the computation time. To speed up this process for applications requiring successive SLCS formulae, one could reduce the state space considered after each processed formula. This would most likely require the construction of a new model from the marked coordinates, on which more complex and time-consuming formula could be run.

The SLCS logic fragment we discussed is not sufficient to do precise image segmentation in medical images, leading [5] to present the language ImgQL, which extends SLCS with a distance operator and region similarity to perform texture analysis. Their spatio-temporal model checker `topochecker` [6] also implements these techniques. These extensions, however, fell outside the scope of this thesis. Future work would involve the possibility of their implementation in mCRL2.

Bibliography

[1] Luca Aceto, Anna Ingolfsdottir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems.* Cambridge University Press, 2007. DOI: 10.1017/cbo9780511814105. URL: http://dx.doi.org/10.1017/CBO9780511814105.

[2] Christal Baier and Joost P. Katoen. *Principles of Model Checking.* English. United States: MIT Press, May 2008. ISBN: 978-0-262-02649-9.

[3] Julian Charles Bradfield. "Program Logics and the Mu-Calculus". In: *Verifying Temporal Properties of Systems* (1992), pp. 14–29. DOI: 10.1007/978-1-4684-6819-9_2.

[4] Olav Bunte, Jan Friso Groote, Jeroen J.A. Keiren, Maurice Laveaux, Thomas Neele, Erik P. de Vink, Wieger Wesselink, Anton Wijs, and Tim A.C. Willemse. "The mCRL2 toolset for analysing concurrent systems: improvements in expressivity and usability". In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer, Jan. 2019, pp. 21–39. DOI: 10.1007/978-3-030-17465-1_2.

[5] Fabrizio Banci Buonamici, Gina Belmonte, Vincenzo Ciancia, Diego Latella, and Mieke Massink. "Spatial logics and model checking for medical imaging". In: *International Journal on Software Tools for Technology Transfer* 22 (2020), pp. 195–217. DOI: 10.1007/s10009-019-00511-9.

[6] Vincenzo Ciancia and Gina Belmonte. *Topochecker: a topological model checker.* URL: https://github.com/vincenzoml/topochecker (visited on 05/20/2022).

[7] Noel Codella, Quoc-Bao Nguyen, S. Pankanti, David Gutman, Brian Helba, Allan Halpern, and John Smith. "Deep Learning Ensembles for Melanoma Recognition in Dermoscopy Images". In: *Ibm Journal of Research and Development* 61 (June 2017). DOI: 10.1147/JRD.2017.2708299.

[8] Stéphane Demri, Valentin Goranko, and Martin Lange. "The Modal Mu-Calculus". In: *Temporal Logics in Computer Science: Finite-State Systems.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016, pp. 271–328. DOI: 10.1017/CBO9781139236119.008.

[9] Stefan Edelkamp and Christoph Greulich. "A case study of planning for smart factories: Model checking and Monte Carlo search for the rescue". In: *International Journal on Software Tools for Technology Transfer* 20 (Oct. 2018), pp. 515–528. DOI: 10.1007/s10009-018-0498-1.

[10] Technische Universiteit Eindhoven. *mCRL2 - mu-calculus.* URL: https://www.mcrl2.org/web/user_manual/language_reference/mucalc.html#state-formulas (visited on 04/12/2022).

[11] Python Software Foundation. *re - Regular expression operations - Python 3.10.4 documentation.* URL: https://docs.python.org/3/library/re.html (visited on 04/12/2022).

[12] Antony Galton. “A generalized topological view of motion in discrete space”. In: *Theoretical Computer Science* 305.1-3 (2003), pp. 111–134. DOI: 10.1016/s0304-3975(02)00701-6.

[13] J.F. Groote, Jeroen J.A. Keiren, Bas Luttik, Erik P. de Vink, and Tim A.C. Willemse. *Modelling and analysing software in mCRL2.* Computer Science Reports. Technische Universiteit Eindhoven, Dec. 2019.

[14] J.F. Groote and T.A.C. Willemse. “Parametrised Boolean equation systems”. English. In: *Theoretical Computer Science* 343.3 (2005), pp. 332–369. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2005.06.016.

[15] Jan Friso Groote and Mohammad Reza Mousavi. *Modelling and analysis of communicating systems.* 2014. DOI: 10.7551/mitpress/9946.001.0001.

[16] Ehsan Khamespanah, Marjan Sirjani, Kirill Mechitov, and Gul Agha. “Modeling and analyzing real-time wireless sensor and actuator networks using actors and model checking”. In: *International Journal on Software Tools for Technology Transfer* 20 (Oct. 2018). DOI: 10.1007/s10009-017-0480-3.

[17] Bjoern H. Menze et al. “The Multimodal Brain Tumor Image Segmentation Benchmark (BRATS)”. In: *IEEE Transactions on Medical Imaging* 34.10 (2015), pp. 1993–2024. DOI: 10.1109/TMI.2014.2377694.

[18] Wieger Wesselink and Tim Willemse. *PBES Instantiating and Solving - mCRL2 documentation.* URL: https://www.mcrl2.org/web/_downloads/772324bb392ab1334dada8d65c8192ca/pbes-instantiation-solving.pdf (visited on 06/11/2022).