A spatial model checker in GPU (extended version)*

Laura Bussi[1], Vincenzo Ciancia[2], and Fabio Gadducci[1]

[1] Dipartimento di Informatica, Università di Pisa
[2] Istituto di Scienza e Tecnologie dell'Informazione, CNR

Abstract. The tool `VoxLogicA` merges the state-of-the-art library of computational imaging algorithms `ITK` with the combination of declarative specification and optimised execution provided by spatial logic model checking. The analysis of an existing benchmark for segmentation of brain tumours via a simple logical specification reached state-of-the-art accuracy. We present a new, GPU-based version of `VoxLogicA` and discuss its implementation, scalability, and applications.

Keywords: Spatial logics · Model Checking · GPU computation

1 Introduction

Spatial and Spatio-temporal model checking have gained an increasing interest in recent years in various application domains, including collective adaptive [13,18] and networked systems [5], runtime monitoring [26,4], modelling of cyber-physical systems [31] and medical imaging images [24,3]. Current research in this field has its origin in the *topological* approach to spatial logics, whose early theoretical foundations have been extended to encompass reasoning about *discrete* spatial structures, such as graphs and images [16], and reasoning over space-time [19,31,26], with related model checking algorithms and tools [14,10,6].

Introduced in [10], continuing the research line of [8,3,7], `VoxLogicA` (*Voxel-based Logical Analyser*)[3] caters for a novel,declarative approach to (medical) image segmentation, supported by spatial model checking. Therein, the main case study was brain tumour segmentation for radiotherapy (see e.g., [23,34,21,30,22]), using the BraTS 2017 dataset (a publicly available set of benchmark MRI images for brain tumour segmentation including high-quality *ground truth*: see [2]). A simple high-level specification for glioblastoma segmentation was proposed and tested using `VoxLogicA`. The procedure competes in accuracy with state-of-the-art techniques, most of which based on machine learning. This work presents a

* Research partially supported by the MIUR Project PRIN 2017FTXR7S "IT- MaT-TerS". The authors are thankful to Raffaele Perego, Franco Maria Nardini and the High Performance Computing Laboratory at ISTI-CNR for providing access to the powerful machine used in our tests in Section 4. The authors also acknowledge fruitful discussions with Gina Belmonte, Diego Latella, and Mieke Massink.

[3] `VoxLogicA`: see `https://github.com/vincenzoml/VoxLogicA`

variant of `VoxLogicA`, named `VoxLogicA-GPU`, that implements the core logical primitives in GPU. The motivation is shared with a recent trend on theory and implementation of formal methods in GPU (see [11,32,33,25,28,27]): to take advantage of the availability of high-performance, massively parallel computational devices, in order to increase the size of tractable problems. We describe the tool implementation, architecture, and issues (in particular, connected component labelling in GPU). We compared the CPU and GPU implementations, both on artificial test cases consisting of very large formulas, and on the brain tumour segmentation case study. Results aimed at checking scalability on large formulas are very encouraging, as the command pipeline of the GPU is fully exploited, and there is a considerable speed-up with respect to using the CPU. On the case study, the current limitations of `VoxLogicA-GPU` (namely, the restriction to 2D images and the smaller number of primitive operations that has been implemented) forbid a direct comparison with [10], but the GPU version was still able to outperform the CPU on a simplified experiment, which is particularly relevant, as the CPU algorithms rely on a state-of-the-art imaging library.

2 The Spatial Logic SLCS

In this section, we briefly review the syntax of the spatial logic SLCS, defined in [15,16], and its interpretation, restricted to the case of two-dimensional images which is currently handled by VoxLogicA-GPU. For the general definition on so-called *closure spaces*, and the link between (multi-dimensional) images, graphs, closure spaces and topological spatial logics we refer the reader to [16,3,10]. The syntax of the logic we use in this paper is its most up-to-date rendition, where the *surrounded* connective from [16] is a derived one, whereas reachability is primitive, as in [4,17,20]. Given set P of *atomic predicates*, with $p \in P$, the syntax of the logic is described by the following grammar:

$$\phi ::= p \mid \neg\,\phi \mid \phi_1 \wedge \phi_2 \mid \mathcal{N}\phi \mid \rho\,\phi_1[\phi_2] \tag{1}$$

The logic is interpreted on the pixels of an image $\mathcal{M}$ of fixed dimensions. The truth values of a formula ϕ on *all* the pixels can be rendered as a binary image of the same dimensions of $\mathcal{M}$. Therefore, in particular, **atomic propositions** correspond to binary images. To get an intuition, consider that typical atomic propositions are numeric constraints (thresholds) on imaging features, such as intensity, or red, green, blue colour components. **Boolean operators** are defined pixel-wise: $\neg\phi$ is the complement of the binary image representing ϕ, and $\phi_1 \wedge \phi_2$ is binary image intersection. The **modal** formula $\mathcal{N}\phi$ is interpreted as the set of pixels that share a vertex or an edge with any of the pixels that satisfy ϕ (adopting the so-called *Moore neighbourhood*); in imaging terminology, this is the *dilation* of the binary image corresponding to ϕ. The **reachability** formula $\rho\,\phi_1[\phi_2]$ is interpreted as the set of pixels x such that there is a pixel y, a path π and an index ℓ such that $\pi_0 = x$, $\pi_\ell = y$, y satisfies ϕ_1, and all the intermediate pixels $\pi_1, \ldots, \pi_{\ell-1}$ (if any) satisfy ϕ_2.

From the basic operators, several interesting notions can be derived, such as *interior* (corresponding to the imaging primitive of *erosion*), *surroundedness*, *contact* between regions (see also [17], encoding the *discrete* Region Calculus RCC8D of [29] in a variant of SLCS).

3 The tool VoxLogicA-GPU

VoxLogicA-GPU is a *global*, *explicit state* model checker for the logic SLCS defined in [16] , aiming at high efficiency and maximum portability VoxLogicA-GPU is implemented in FSharp, using the NET Core infrastructure, and the *General-Purpose GPU computing* library OpenCL.[4] Efficiency is one of the major challenges, as outperforming VoxLogicA inherently means designing *in-GPU* imaging primitives faster than the state-of-the-art library ITK. The focus of the first release of VoxLogicA-GPU is on the implementation and early dissemination of a free and open source infrastructure to experiment with GPU-based spatial model checking, and demonstrate its scalability. Thus, development has been narrowed to a core implementation that is powerful enough to reach the stated objectives, although not as feature-complete as VoxLogicA. IN particular, the implemented primitives are those of SLCS, plus basic arithmetic. More complex operations (normalization, distance transform, statistical texture analysis) are yet to be implemented. Furthermore, in the first release, computation is restricted to 2D images and 16 bit unsigned integers[5]; this eased development, as a separate kernel is needed for each dimensionality and each numeric type.

3.1 Syntax

VoxLogicA-GPU is a command-line tool. It takes only one parameter, a text file (usually, with `.imgql` extension) containing the specification to be executed. In the following, `f, x1,..., xN, x` are identifiers, `"s"` is a string, and `e1, ..., eN, e` are expressions (to be detailed later). A specification consists of a text file containing a sequence of **commands** (see Section 4.2 for an example). Five commands are currently implemented: `let`, `load`, `save`, `print`, `import`. For the scope of this work, it suffices to describe the first three. The command `let f(x1,...,xN) = e` is *function declaration* (`let f = e` declares a constant) with a special syntax for *infix* operators; the command `load x = "s"` loads an image from file `"s"` and binds it to `x` for subsequent usage; the command `save "s" e` stores the image resulting from evaluation of expression `e` to file `"s"`. VoxLogicA-GPU comes equipped with built-in arithmetic operators and logic primitives. Furthermore, a "standard library" is provided containing short-hands for commonly used functions, and for derived operators. An **expression** may be a number (with no distinction between floating point and integer constants), an identifier (e.g. `x`), a function application (e.g. `f(x1,x2)`), an infix operator application (e.g. `x1 + x2`), or a parenthesized expression (e.g. `(x1 + x2)`).

[4] FSharp: see https://fsharp.org. NET Core: see https://dotnet.microsoft.com. OpenCL: see https://www.khronos.org/opencl. ITK: see https://itk.org.

[5] The precision of 16 bits is needed for the BraTS dataset, see Section 4.

3.2 Implementation details

The core model-checking algorithm of `VoxLogicA-GPU` is in common with `VoxLogicA`. After parsing, imported libraries are resolved, load instructions are executed, and parametric macros are expanded, avoiding to duplicate sub-expressions (see [10] for more details). The result is a *directed acyclic graph*, where each node contains a task to execute, after the tasks from which it depends (denoted by edges) are completed. A task can be either an operator of the language, or an output instruction. The semantics of operators is delegated to the specific implementation of the VoxLogicA API, which must define the type `Value`, on which tasks operate. Operators are implemented in a module running on CPU that launches one or more *kernels* (i.e. functions running on GPU), executed in an asynchronous way. As the execution order is not known a priori, the desired kernel is retrieved at runtime and passed to an auxiliary module that performs the actual call to the GPU code. This module is also in charge of setting the actual parameters (i.e. image buffers or numeric values) of the kernel. The type Value is used to describe information about the type of data stored in the GPU memory (a binary or numeric image, or a number). More precisely, it contains a pointer to an OpenCL buffer stored in the GPU memory, the pixel type of the image and the number of components per pixel. These information are needed in order to have fully composable operators, as the image type may vary during computation (e.g. when computing connected components labelling). Providing a pointer to the OpenCL buffers minimises data transferring between host and device, as data is retrieved from the CPU only when it is strictly necessary to continue execution. The correct execution order is preserved using OpenCL *events*, i.e. objects that communicate the status of a command. Due to asynchronicity, kernels are pushed in a *command queue* that may also contain global barriers or read/write to the GPU memory. Each time a command is added to the queue, a new event is added to the event list and will be eventually fired at the command completion. Events can be chained, and the CPU can wait for an event to complete and read the result of an associated computation from the GPU. Wait instructions are only used in the implementation of reachability-based logical primitives, in order to check if the GPU reported termination of an iterative algorithm, and when saving results, achieving very high GPU usage. Just like in `VoxLogicA`, the reachability operator $\rho\ \phi_1[\phi_2]$ is implemented using connected components labelling, which makes the implementation such operation in GPU very relevant for this paper.In order to get an intuition about this, consider a binary image I whose non-zero points are the points satisfying ϕ_2. Consider any pixel z in a connected component of I that has a point in common with the dilation of ϕ_1. By construction there is a path π and an index ℓ, with $\pi_0 = z$, π_ℓ satisfying ϕ_1, and all the pixels $\pi_0, \ldots, \pi_{\ell-1}$ satisfying ϕ_2. Therefore the points in the dilation of the set of all z in such situation satisfies $\rho\ \phi_1[\phi_2]$, as well as the pixels satisfying $\mathcal{N}\phi_1$; there are no other pixels satisfying $\rho\ \phi_1[\phi_2]$.

3.3 Connected components labelling in `VoxLogicA-GPU`

The current, iterative algorithm for connected component labelling is based on the *pointer jumping* technique, and it has been designed to balance implementation simplicity with performance. A more detailed investigation of this and other algorithms is left for future research. Algorithm 1 presents the pseudo-code of the kernels (termination checking is omitted); see Figure 1 for an example.

Algorithm 1: Pseudocode for connected components labelling

```
1  initialization(start: image of bool, output: image of int × int )
2      for (i, j) ∈ Coords do
3          if start(i,j) then
4              output(i, j) = (i, j) // null otherwise
5  mainIteration(start: image of bool, input, output: image of int × int)
6      for (i, j) ∈ Coords do
7          if start(i,j) then
8              (i′, j′) = input(i, j)
9              output(i, j) = maxNeighbour(input, i′, j′)
10 reconnect(start: image of bool, input, output: image of int × int)
11     for (i, j) ∈ Coords do
12         if start(i,j) then
13             (i′, j′) = input(i, j)
14             (a, b) = maxNeighbour(input, i, j)
15             (c, d) = input(i′, j′)
16             if (a, b) > (c, d) then
17                 output(i′, j′) = (a, b) // Requires atomic write
```

The algorithm uses coordinates (x, y) as labels. Starting from a binary image I, the algorithm initialises the output image (a two-dimensional array of pairs) by labelling each non-zero pixel of I with its coordinates. At each iteration, in parallel on the GPU, for each p at (x, y) which is non-zero in I, containing (x', y'), the Moore neighbourhood of (x', y') is inspected to find a maximum value (x'', y''); the lexicographic order is used (possibly, $(x'', y'') = (x', y')$). The value (x'', y'') is written at (x, y). See Figure 1 for an example.

Such algorithm converges in a logarithmic number of iterations with respect to the number of pixels due to pointer jumping. The algorithm may fail to label uniquely those connected components that contain a particular type of "concave corner", namely a pixel at coordinates (a, b), together with the pixels at $(a+1, b)$, $(a, b + 1)$, but not the pixel at $(a + 1, b + 1)$.

To overcome this, each k main iterations (with k very small, $k = 16$ in the current implementation) the algorithm employs a *reconnect* step, that inspects the neighbourhood of each pixel p, containing (x, y), looking for a pair (x', y') greater than the pair stored at (x, y). In that case, (x', y') is written at (x, y), and the main iterations are restarted, which immediately propagates (x', y') to p, and all other pixels that contain (x', y'). At this point termination is checked, by invoking a separate kernel that checks if any pixel which is true in the original

binary image has a neighbour with a different label (if no pixel is in this situation, then the connected components have been properly labelled)

By construction, the algorithm exits *only if* all connected components have been correctly labelled. An invariant is that after each main iteration, each pixel p containing (x, y) lays in the same connected component as (x, y). For termination, call k the number of distinct labels in the image along computation. Clearly, k does not increase after a main iteration (in most iterations it decreases). After each reconnect step, there are three possibilities: 1) k decreases; 2) k does not change, but in the next step, the algorithm terminates; 3) k does not change, but in the next main iteration, k decreases. Therefore, k always decreases after a reconnect step and a main iteration, unless the algorithm terminates. Thus, by the invariant, k converges to a minimum which is the number of connected components. In practice, convergence is quite fast (usually, logarithmic in the number of pixels) and suffices to establish the results in Section 4.

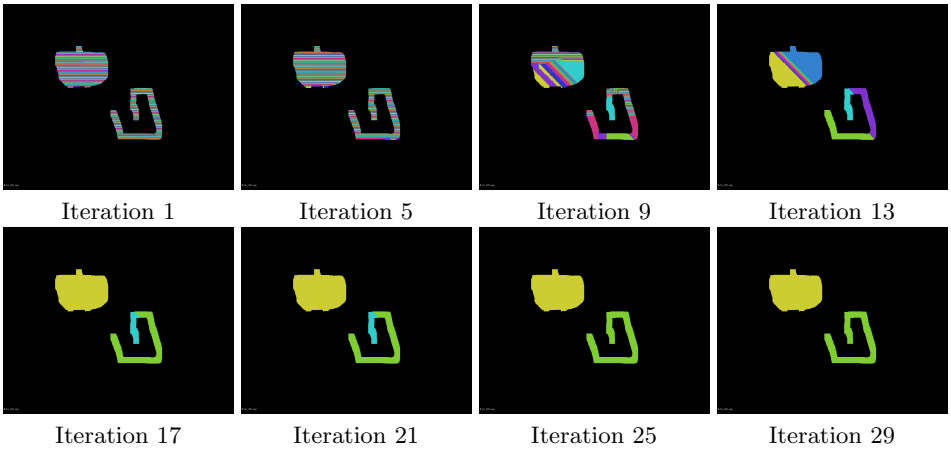

Fig. 1. CC-labelling algorithm on a test image. Different colours represent different labels. Reconnect is called every 8 main iterations. At iteration 13, the main iterations have converged, therefore the image stays unchanged until iteration 16 (reconnect); iteration 17 shows label propagation after reconnect. Then until iteration 24 (last reconnect), images do not change again.

4 Tests and Brain Tumour Segmentation Case Study

4.1 Performance on large formulas

We built two kinds of large formulas in order to stress the infrastructure: sequential forumlas (i.e. formulas of the form f(g(...(x)))), and "random" ones, were all the operators are composed in various ways and applied over different images. In

both these cases, `VoxLogicA-GPU` scales better than the CPU version by a linear factor as the size of the formula grows. It is worth noting that the CPU version has better performances on very small formulas in the random test, due to the overhead needed to set up GPU computation. Figure 4.1 shows how the CPU and the GPU versions scale differently on growing formulas. As in Section 4.2, execution times are small. We foresee more important speed-ups on real-world experiments, in the near future.

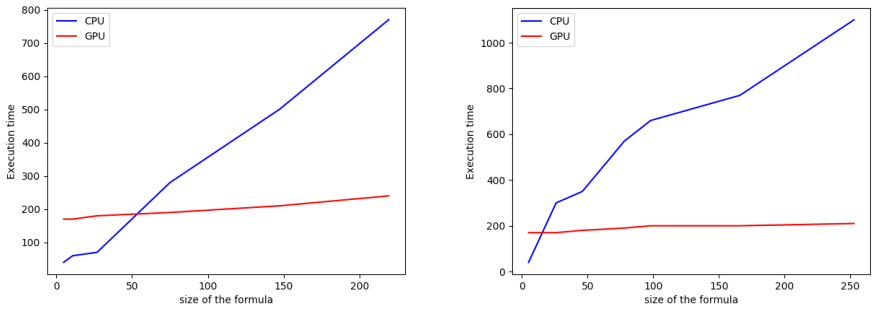

Fig. 2. Performance on growing sequential (left) and random (right) formulas

4.2 Brain Tumour Segmentation

The main case study of [10] was brain tumour segmentation for radiotherapy, achieving accuracy results on par with the state of the art tools. Validation was performed on the BraTS 2017 dataset[6], which provides *MRI-flair* 3D images composed by circa 9 millions of *voxels* (three-dimensional pixels) and manual segmentations for accuracy scores. Given the current restrictions of `VoxLogicA-GPU` to 2D images, core logical primitives, and 16-bit unsigned integers (see Section 3), we have extrapolated a simplified dataset and the specification in Figure 3.

The specification uses the operator $grow(\phi_1, \phi_2)$, derived from ρ, to identify pixels that either satisfy ϕ_1, or belong to a region which is in contact with ϕ_1, and only contains pixels satisfying ϕ_2. The particular simplicity of such procedure should not raise doubts about its effectiveness, as "semantic noise removal" is the core of the specification in [10], and contributes to its accuracy (and computational cost) for the main part; but note that the restriction to 2D images makes the procedure too fast to be measured in smaller images, requiring artificial up-scaling in order to compare the performance of `VoxLogicA` and `VoxLogicA-GPU`.

To extract a 2D slice out of each 3D MRI-flair scan of the BraTS dataset, each 3D image is first normalised using the `percentiles` operator of VoxLogicA,

[6] See `http://braintumorsegmentation.org`

```
// Load data (16 bit png, normalised)
load img = "normalised-MRI-flair.png"

// 1. Thresholds
let hI = intensity(img) >. 62258 // (62258 = 0.95 * 65535; hyperintense)
let vI = intensity(img) >. 56360 // (56360 = 0.86 * 65535; very intense)

// 2. Semantic noise removal via region growing
let gtv = grow(hI,vI)

// Save the results
save "segmentation.png" gtv
```

Fig. 3. Brain Tumour Segmentation in `VoxLogicA-GPU`

in order to remain faithful to the kind of processing done in [10]. The normalised image has floating point intensity values between 0 and 1; these have been multiplied by $2^{16} - 1$ (the maximum 16-bit unsigned integer) in order to maximise precision (therefore the thresholds in the specification were also multiplied by the same number). Finally, the "most significant" slice has been selected as the one where the manual segmentation has a larger volume, resulting in an image which contains enough information to test the performance of `VoxLogicA-GPU`. Finally, the slice has been up-scaled (with no filtering) in order to obtain a very large image, (size is 7680×7680 pixels, that is circa 60 megapixels) in order to challenge the efficiency of VoxLogicA, where the implementation of most logical primitives is delegated to the state-of-the-art imaging library ITK[7].

We have tested the specification on 4 devices, averaging the measurements on several runs. The GPU algorithm has been tested with `VoxLogicA-GPU` on two different GPUs, namely an *Intel HD Graphics 630* (integrated GPU with very low performance), taking about 3 seconds to complete; and a *NVIDIA TITAN Xp GP102*, which is a high-end desktop GPU, taking about 600ms to complete. The CPU algorithm has been tested with `VoxLogicA` on the same machine used in [10], equipped with a *Intel Core i7-7700* CPU, taking about 1100 milliseconds, and on the machine hosting the *NVIDIA TITAN Xp*, which also has a *Intel Xeon E5-2630* CPU, taking 750 milliseconds. Runtime was measured from the moment in which the tools print "starting computation" in the log (signifying that all preparation, parsing, memoization, etc. has been finalised) to the moment in which the tools print "saving file ...", meaning that the results have been fully computed, and transferred from the GPU to the CPU (for `VoxLogicA-GPU`). The measured times are quite low, hence not very significant, due to the simplicity of the procedure and the restriction to 2D images. Doubling the image size is not possible at the moment as it would not be possible to allocate the needed memory buffers on the devices available at the moment. Obviously, larger and

[7] The Insight Toolkit, see `https://itk.org/`.

more complex specifications, such as the gray and white matter segmentation in [9], which takes minutes to complete in CPU, will benefit more of the GPU implementation (as discussed in Section 4.1). More relevant tests will be possible when the tool will support 3D images (which are more challenging for the CPU) and after the implementation of further, heavy operations of `VoxLogicA` such as local histogram analysis. It is noteworthy that the GPU implementation already outperforms the CPU in this case study (on the available devices), and that both implementations achieve impressive practical results, by segmenting a brain tumour on such a large image in less than one second. In CPU, this is due to the state-of-the art implementation of primitives in the ITK library. In GPU, the connected components algorithm that we designed yields very good results.

5 Conclusions

Our preliminary evaluation of spatial model checking in GPU is encouraging: large formulas benefit most of the GPU, with significant speed-ups. Connected components labelling should be a focus for future work; indeed, the topic is very active, and our simple algorithm, that we used as a proof-of-concept, may as well be entirely replaced by state-of-the-art, more complex procedures (see e.g. the recent work [1]). Making `VoxLogicA-GPU` feature-complete with respect to `VoxLogicA` is also a stated goal for future development. In this respect, we remark that although in this work we decided to go through the "GPU-only" route, future development will also consider a *hybrid* execution mode with some operations executed on the CPU, so that existing primitives in `VoxLogicA` can be run in parallel with those that already have a GPU implementation. Usability of `VoxLogicA-GPU` would be greatly enhanced by an user interface; however, understanding modal logical formulas is generally considered a difficult task, and the novel combination of declarative programming and domain-specific medical image analysis concepts may hinder applicability of the methodology, as cognitive/human aspects may become predominant with respect to technological concerns. In this respect, we plan to investigate the application of formal methodologies (see e.g. [12]).

References

1. Allegretti, S., Bolelli, F., Grana, C.: Optimized block-based algorithms to label connected components on gpus. IEEE Trans. Parallel Distributed Syst. **31**(2), 423–438 (2020). https://doi.org/10.1109/TPDS.2019.2934683, `https://doi.org/10.1109/TPDS.2019.2934683`
2. Bakas, S., Akbari, H., Sotiras, A., Bilello, M., Rozycki, M., Kirby, J.S., Freymann, J.B., Farahani, K., Davatzikos, C.: Advancing the cancer genome atlas glioma MRI collections with expert segmentation labels and radiomic features. Scientific Data **4** (2017). https://doi.org/10.1038/sdata.2017.117, `https://doi.org/10.1038/sdata.2017.117`, online publication date: 2017/09/05

3. Banci Buonamici, F., Belmonte, G., Ciancia, V., Latella, D., Massink, M.: Spatial logics and model checking for medical imaging. Int. J. Softw. Tools Technol. Transf. **22**(2), 195–217 (2020). https://doi.org/10.1007/s10009-019-00511-9, https://doi.org/10.1007/s10009-019-00511-9
4. Bartocci, E., Bortolussi, L., Loreti, M., Nenzi, L.: Monitoring mobile and spatially distributed cyber-physical systems. In: Talpin, J., Derler, P., Schneider, K. (eds.) Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2017, Vienna, Austria, September 29 - October 02, 2017. pp. 146–155. ACM (2017). https://doi.org/10.1145/3127041.3127050, https://doi.org/10.1145/3127041.3127050
5. Bartocci, E., Gol, E., Haghighi, I., Belta, C.: A formal methods approach to pattern recognition and synthesis in reaction diffusion networks. IEEE Transactions on Control of Network Systems pp. 1–1 (2016). https://doi.org/10.1109/tcns.2016.2609138
6. Bartocci, E., Bortolussi, L., Loreti, M., Nenzi, L., Silvetti, S.: Moonlight: A lightweight tool for monitoring spatio-temporal properties. In: Deshmukh, J., Nickovic, D. (eds.) Runtime Verification - 20th International Conference, RV 2020, Los Angeles, CA, USA, October 6-9, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12399, pp. 417–428. Springer (2020). https://doi.org/10.1007/978-3-030-60508-7_23, https://doi.org/10.1007/978-3-030-60508-7_23
7. Belmonte, G., Ciancia, V., Latella, D., Massink, M., Biondi, M., De Otto, G., Nardone, V., Rubino, G., Vanzi, E., Banci Buonamici, F.: A topological method for automatic segmentation of glioblastoma in MR FLAIR for radiotherapy - ESMRMB 2017, 34th annual scientific meeting. Magnetic Resonance Materials in Physics, Biology and Medicine **30**(S1), 437 (oct 2017). https://doi.org/10.1007/s10334-017-0634-z, https://doi.org/10.1007/s10334-017-0634-z
8. Belmonte, G., Ciancia, V., Latella, D., Massink, M.: From collective adaptive systems to human centric computation and back: Spatial model checking for medical imaging. In: ter Beek, M.H., Loreti, M. (eds.) Proceedings of the Workshop on FORmal methods for the quantitative Evaluation of Collective Adaptive SysTems, FORECAST@STAF 2016, Vienna, Austria, 8 July 2016. EPTCS, vol. 217, pp. 81–92 (2016). https://doi.org/10.4204/EPTCS.217.10, https://doi.org/10.4204/EPTCS.217.10
9. Belmonte, G., Ciancia, V., Latella, D., Massink, M.: Innovating medical image analysis via spatial logics. In: ter Beek, M.H., Fantechi, A., Semini, L. (eds.) From Software Engineering to Formal Methods and Tools, and Back - Essays Dedicated to Stefania Gnesi on the Occasion of Her 65th Birthday. Lecture Notes in Computer Science, vol. 11865, pp. 85–109. Springer (2019). https://doi.org/10.1007/978-3-030-30985-5_7, https://doi.org/10.1007/978-3-030-30985-5_7
10. Belmonte, G., Ciancia, V., Latella, D., Massink, M.: Voxlogica: A spatial model checker for declarative image analysis. In: Vojnar, T., Zhang, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11427, pp. 281–298. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_16, https://doi.org/10.1007/978-3-030-17462-0_16

11. Berkovich, S., Bonakdarpour, B., Fischmeister, S.: Gpu-based runtime verification. In: 2013 IEEE 27th International Symposium on Parallel and Distributed Processing. pp. 1025–1036 (2013)
12. Broccia, G., Milazzo, P., Ölveczky, P.C.: Formal modeling and analysis of safety-critical human multitasking. Innov. Syst. Softw. Eng. **15**(3-4), 169–190 (2019). https://doi.org/10.1007/s11334-019-00333-7, https://doi.org/10.1007/s11334-019-00333-7
13. Ciancia, V., Gilmore, S., Grilletti, G., Latella, D., Loreti, M., Massink, M.: Spatio-temporal model checking of vehicular movement in public transport systems. International Journal on Software Tools for Technology Transfer (Jan 2018). https://doi.org/10.1007/s10009-018-0483-8, https://doi.org/10.1007/s10009-018-0483-8
14. Ciancia, V., Grilletti, G., Latella, D., Loreti, M., Massink, M.: An experimental spatio-temporal model checker. In: Software Engineering and Formal Methods - SEFM 2015 Collocated Workshops. Lecture Notes in Computer Science, vol. 9509, pp. 297–311. Springer (2015). https://doi.org/10.1007/978-3-662-49224-6_24
15. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Specifying and verifying properties of space. In: Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8705, pp. 222–235. Springer (2014). https://doi.org/10.1007/978-3-662-44602-7_18
16. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Model Checking Spatial Logics for Closure Spaces. Logical Methods in Computer Science **Volume 12, Issue 4** (Oct 2016). https://doi.org/10.2168/LMCS-12(4:2)2016, http://lmcs.episciences.org/2067
17. Ciancia, V., Latella, D., Massink, M.: Embedding RCC8D in the Collective Spatial Logic CSLCS. In: Boreale, M., Corradini, F., Loreti, M., Rosario, P. (eds.) To be defined, pp. 251–266. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2019), accepted for publication
18. Ciancia, V., Latella, D., Massink, M., Paškauskas, R., Vandin, A.: A tool-chain for statistical spatio-temporal model checking of bike sharing systems. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9952, pp. 657–673 (2016). https://doi.org/10.1007/978-3-319-47166-2_46, http://dx.doi.org/10.1007/978-3-319-47166-2_46
19. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Spatial logic and spatial model checking for closure spaces. In: Bernardo, M., Nicola, R.D., Hillston, J. (eds.) Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems - 16th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2016, Bertinoro, Italy, June 20-24, 2016, Advanced Lectures. Lecture Notes in Computer Science, vol. 9700, pp. 156–201. Springer (2016). https://doi.org/10.1007/978-3-319-34096-8_6, https://doi.org/10.1007/978-3-319-34096-8_6
20. Ciancia, V., Latella, D., Massink, M., de Vink, E.P.: Towards spatial bisimilarity for closure models: Logical and coalgebraic characterisations. CoRR **abs/2005.05578** (2020), https://arxiv.org/abs/2005.05578
21. Despotović, I., Goossens, B., Philips, W.: MRI segmentation of the human brain: Challenges, methods, and applications. Computational and Mathematical Methods

in Medicine **2015**, 1–23 (2015). https://doi.org/10.1155/2015/450341, http://dx.doi.org/10.1155/2015/450341
22. Dupont, C., Betrouni, N., Reyns, Vermandel, M.: On image segmentation methods applied to glioblastoma: State of art and new trends. IRBM **37**(3), 131–143 (jun 2016). https://doi.org/10.1016/j.irbm.2015.12.004
23. Fyllingen, E., Stensjøen, A., Berntsen, E., Solheim, O., Reinertsen, I.: Glioblastoma segmentation: Comparison of three different software packages. PLOS ONE **11**(10), e0164891 (oct 2016). https://doi.org/10.1371/journal.pone.0164891
24. Grosu, R., Smolka, S., Corradini, F., Wasilewska, A., Entcheva, E., Bartocci, E.: Learning and detecting emergent behavior in networks of cardiac myocytes. Commun. ACM **52**(3), 97–105 (2009)
25. Neele, T., Wijs, A., Bošnački, D., van de Pol, J.: Partial-order reduction for gpu model checking. In: Artho, C., Legay, A., Peled, D. (eds.) Automated Technology for Verification and Analysis. pp. 357–374. Springer International Publishing, Cham (2016)
26. Nenzi, L., Bortolussi, L., Ciancia, V., Loreti, M., Massink, M.: Qualitative and quantitative monitoring of spatio-temporal properties. In: Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9333, pp. 21–37. Springer (2015). https://doi.org/10.1007/978-3-319-23820-3_2
27. Osama, M., Wijs, A.: Parallel SAT simplification on GPU architectures. In: Vojnar, T., Zhang, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11427, pp. 21–40. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_2, https://doi.org/10.1007/978-3-030-17462-0_2
28. Osama, M., Wijs, A.: Sigma: Gpu accelerated simplification of sat formulas. In: Ahrendt, W., Tapia Tarifa, S.L. (eds.) Integrated Formal Methods. pp. 514–522. Springer International Publishing, Cham (2019)
29. Randell, D.A., Landini, G., Galton, A.: Discrete mereotopology for spatial reasoning in automated histological image analysis. IEEE Trans. Pattern Anal. Mach. Intell. **35**(3), 568–581 (2013). https://doi.org/10.1109/TPAMI.2012.128, https://doi.org/10.1109/TPAMI.2012.128
30. Simi, V., Joseph, J.: Segmentation of glioblastoma multiforme from MR images – a comprehensive review. The Egyptian Journal of Radiology and Nuclear Medicine **46**(4), 1105–1110 (dec 2015). https://doi.org/10.1016/j.ejrnm.2015.08.001
31. Tsigkanos, C., Kehrer, T., Ghezzi, C.: Modeling and verification of evolving cyber-physical spaces. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. pp. 38–48. ESEC/FSE 2017, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3106237.3106299, http://doi.acm.org/10.1145/3106237.3106299
32. Wijs, A., Bosnacki, D.: Many-core on-the-fly model checking of safety properties using gpus. Int. J. Softw. Tools Technol. Transf. **18**(2), 169–185 (2016). https://doi.org/10.1007/s10009-015-0379-9, https://doi.org/10.1007/s10009-015-0379-9
33. Wijs, A., Neele, T., Bošnački, D.: Gpuexplore 2.0: Unleashing gpu explicit-state model checking. In: Fitzgerald, J., Heitmeyer, C., Gnesi, S., Philippou, A. (eds.) FM 2016: Formal Methods. pp. 694–701. Springer International Publishing, Cham (2016)

34. Zhu, Y., Young, G., Xue, Z., Huang, R., You, H., Setayesh, K., Hatabu, H., Cao, F., Wong, S.: Semi-automatic segmentation software for quantitative clinical brain glioblastoma evaluation. Academic Radiology **19**(8), 977–985 (aug 2012). https://doi.org/10.1016/j.acra.2012.03.026