Aalborg University, department of Computer Science

Visualization of Zones in Real-Time Models

Studerende:
Nicolai Brobak: nbroba13@student.aau.dk

Vejleder:
Ulrik Mathias Nyman: ulrik@cs.aau.dk

June 15, 2018

1. Summary

This project deals with visualization of zones of real-time models. In section 3 the concepts of real-time models and zones are introduced along with some modeling and verification tools based on these concepts. In order to show a zone it is necessary to find it's vertices. Therefore section 4 presents the criss-cross algorithm for enumeration of the vertices of a polytope. Since the criss-cross algorithm is a rather broad approach, considering the specific case of real-time zone, two subproblems are presented in section 5. One regarding a vertex enumeration algorithm and one regarding how visualization of zones can be used and what can be gained from it.

Section 6 describes a new algorithm based on the criss-cross algorithm. The algorithm has also been implemented to examine how visualization can help in tasks concerning real-time models. The details of how this solution was implemented are described in section 7. Then the features and the user interface of the solution are presented in section 8.

The solution has been used to evaluate how such a tool might help in solving tasks related to real-time models and their zones. 6 test subjects solved 12 tasks in an evaluation which is described in section 9. In section 10 the results of the evaluation are presented.

Finally the findings are discussed in section 11, where it is concluded that visualization of zones is a powerful tool to add to a real-time model checking tools symbolic simulator, although it is unclear whether the algorithm developed is the best way to enumerate vertices in this case. This and more avenues of further work is laid out in section 12.

2. Introduction

This master thesis was written at the research unit Distributed, Embedded and Intelligent Systems (DEIS) at the Department of Computer Science of Aalborg University during the spring semester of 2018. The overall theme is visualization, specifically in model checking tools. With that a tool has been developed and evaluated after exploring the following problem.

2.1. Initiating Problem

A feasibility zone or region is the set of all set of values that satisfy a system of constraints. In the field of modelling and verification of real-time systems, feasibility zones are defined by the guards and invariants on the system. The feasibility zones can then be expanded through symbolic evaluation in order to verify the properties of the system. As such feasibility zones are very important, and visualizing them is often used as a tool when understanding how symbolic evaluation is used to verify properties. Despite all this no existing tools for modelling and verification of real-time systems can visualize feasibility zones.

2.2. Acknowledgements

Thank you to my supervisor Ulrik Mathias Nyman for advice and suggestions throughout the project and for introducing me to the problem. Thank you to the group I shared a group room with: Michael Kusk Christensen, Søren Nørgreen Gustafsson and Lasse Würtz for discussions. Thank you to the test subjects who participated in the evaluation. Lastly thank you to Casper Møller Bartholomæussen and Rene Mejer Lauritsen for cooperation as they worked on an ECDAR simulator.

3. Timed Models

This section will present concepts used in modeling and formal verification of real-time systems, as well as some tools that use them.

3.1. Models of Real Time Systems

A real time system is a system where correctness not only depends on the logical order of events but also on their timing[1]. In order to model such systems Labeled Transition Systems (LTS)[2] are insufficient. Therefore Timed Labeled Transition Systems (TLTS) are used. TLTS extend the semantics of LTS with a number of real valued clocks. These clocks are perfectly synchronized and the system can let time elapse instead of taking a transition, which increases the values of all clocks by the amount of the delay. These clocks are used when taking a transition. Besides a label a transition can now have guards and resets. A guard is a constraint on a clock value. A reset is where a set of clocks are set to 0. There can also be constraints on states called invariants. It is only possible to delay by an amount that does not cause a clock to violate an invariant on the current state.[3][4]

Formally a TLTS is a tuple (L, Σ, X, I, E) where

- L is a finite set of locations
- Σ is a finite set of labels
- X is a finite set of clocks
- I is a mapping where each location s is mapped to some clock constraint in $\Phi(X)$
- $E \subseteq L \times \Sigma \times 2^X \times \Phi(X) \times L$ is the set of transitions. A transition $(s, a, \lambda, \phi, s')$ is an edge from location s to location s' on symbol a, $\lambda \subseteq X$ is the set of clocks to be reset when taking the transition, and ϕ is a clock constraint over X that specifies when the transition can be taken.

The set of clock constraints $\Phi(X)$ over clocks X is either $\phi = \phi_1 \wedge \phi_2$ or $\phi = x \otimes c$ where $x \in X$, $c \in \mathbb{Q}$ and the inequality relation is $\otimes \in \{<, >, \leq, \geq\}$.

The semantics of a TLTS is defined by associating a transition system S_A with it. A state of S_A is a pair (s, v) where $s \in L$ and v is a valuation on X such that for each $x \in X$, $v(x) = \mathbb{R}_{>0}$ and v satisfies the invariant $I(s)$.

There are two types of transitions in S_A. The first is an elapse of time where some state (s, v) and some delay $\delta \in \mathbb{R}_{>0}$ gives the transition $(s, v) \xrightarrow{\delta} (s, v + \delta)$ given that for all $0 \geq \delta' \geq \delta$, $v + \delta'$ satisfies the invariant $I(s)$. The other type of transition is a location change where some state (s, v) and some $t \in E$ gives the transition $(s, v) \xrightarrow{a} (s', v')$ given that v satisfies ϕ. v' is a clock valuation such that $v'(x) = 0$ for all $x \in \lambda$ and $v'(x) = v(x)$ for the rest of the clocks.[3]

3.2. Zones

When trying to verify the properties of a real time system we explore the state space. However unlike a normal system which can be modeled with finite states, a model of a timed system will have infinite states. This can be seen by looking at the transition system S_A where time elapse transitions have delays $\delta \in \mathbb{R}_{>0}$. Zones, also known as symbolic states, are a way to handle this, that results in finite states.[4]

A zone is the constraint system, which is used to create a transition system where each state is a pair (s, z) where s is a location and z is a zone. (s, z) represents all states (s, v) of S_A where v satisfies the constraints of z. That means this new transition system can represent the same system, but with a finite state space.

3.2.1. Calculation of a Zone

A zone is calculated from the initial state where all clocks will start with the value 0. Unless an invariant prevents it, the system can delay some amount of time δ. δ must be smaller than the value of the most restrictive invariant on the initial state, let's call that value k. This means all clocks now have values between 0 and k, but they all have the same value. This is the zone of the initial state.

A transition is active as long as some valuation v that satisfies the constraints of the zone, also satisfies the guards on the transition. When taking a transition a new zone is calculated from the old one. All clock values are also constrained by the guards of the transition, and the clocks that are reset get the value of zero. That means all clocks does not necessarily have the same value, but instead we can say something about the difference between clocks. If a clock x has been reset and another clock y has not, then $y - x$ will be in the same range as the value of y. This difference stays the same as time elapses. When the transition has been taken, the system can once again delay some amount of time allowed by the invariants on the new location, which extends the range of possible values of the clocks. The resulting zone is the zone of the new state.

By continuously calculating the zone of each state in this way, the state space can be explored.

3.2.2. Properties of Zones

All feasibility zones are convex polytopes in n dimensions, due to the fact that they are created by constraints bounding the space. Therefore the zone of a timed model is also a convex polytope and n is the number of clocks in the model.

Another property of zones of timed models is that they are guaranteed to have an optimum and a minimum vertex. A minimum vertex is one which has a valuation v such that for all clocks $x \in X$ there exists no other valuation v' in the zone such that $v'(x) < v(x)$. In other words the minimum vertex has the lowest value of the zone in all dimensions, and the opposite is true for the maximum vertex. This is not true for all feasibility zones. It is true for zones of timed models because a clock can't go backwards.

3.3. Existing Tools

There already exist a number of tools that use the theory of real-time modeling and zones for formal verification of real time systems. One such tool in UPPAAL.

3.3.1. UPPAAL

UPPAAL is a toolbox for modeling, simulation and verification of real-time systems created in collaboration by Aalborg University and Uppsala University.[4] Real-time systems can be modeled as a network of timed automata communicating through a number of synchronization channels and shared variables. UPPAAL allows for modeling with both a graphical and a textual description language. Communication on a channel happens by two processes synchronizing on the channel. That means they both take a transition at the same time with complementary labels. For example two timed automata can synchronize on a if one can take a transition with the label a and the other can take one with the label $\overline{a}$. We say that the transition with the label a receives an a and the other one sends an a. In UPPAAL the syntax is $a?$ and $a!$ for receiving and sending respectively. Communication with shared variables happens simply by reading and writing to the integer and clock variables which are global.

UPPAAL has two more modeling concepts. One is urgent channels. An urgent channel is one which forces networks to synchronize on it as soon as it is possible. In other words time can not elapse if there are two automata that can synchronize on an urgent channel. The other concept is committed locations. A committed location is one which must be left immediately. This is used to ensure a sequence of transitions are taken atomically.

Model checking in UPPAAL happens by using the symbolic method with zones to reduce reachability problems to manipulation and solving of constraint systems. In combination with this technique *on-the-fly* searching techniques are used to improve efficiency. Once it has been proven whether a property is satisfied or not, the model checker can generate a diagnostic trace explaining the result.

A model can also be explored in the simulator. The simulator allows a user to take transitions and see how the zone evolves, and which transitions are allowed when. This can also be used to visualize the diagnostic trace generated by the model checker. However the zone is only ever described with the range of values the clocks and their differences are in. In other words by the constraints that make up the zone.[4]

3.3.2. ECDAR

ECDAR is another toolbox for modeling, simulation and verification of real time systems.[5] It uses the backend of a variant of UPPAAL called UPPAAL-Tiga which specializes in timed game automata[6]. ECDAR itself is designed to check for refinement, specification and consistency properties of a system modeled as components that are related with conjunctions, compositions and quotients.

ECDAR 2.0 is a new version with a new front end inspired by modern IDEs. It is built on the codebase of another UPPAAL tool called H-UPPAAL[7]. The result is an Integrated Modeling and Verification Environment (IMVE) for compositional real time systems. It focuses on simplifying the workflow of modeling and verifying. For example with project management and automation of common tasks. It also prevents constructing invalid models when possible and immediately gives a warning otherwise instead of waiting until validation of the model.[5]

Like UPPAAL ECDAR has three main components. The first is used to construct components and models of real time systems. The second one can be used to verify properties of the modeled system by creating queries and running them on the models. Lastly ECDAR also has a simulator to explore traces in the model. As is the case in UPPAAL, the zone is presented as it evolves during the simulation, but only with constraints as text.

4. Vertex Enumeration

As mentioned the tools presented in section 3.3 cannot show a zone graphically. It could however be a powerful tool during simulation. In order to do such a visualization it is necessary to create a geometric representation.

As mentioned a zone is an n-dimensional convex polytope, however only three dimensions can be shown at once. A zone can be projected to three dimensions and then that can be shown as a 3D shape, which will be a convex polyhedron.

In order to render the shape it is necessary to find its vertices. This is a problem that several algorithms have already been developed to solve.

4.1. Pivoting

Pivoting refers to the the way a new vertex can be found from a known one. Suppose we have a vertex specified by d hyperplanes that intersect at this point, where d is the number of dimensions. Note that this corresponds to d constraints with a single solution in a constraint system. To pivot is to interchange one of the hyperplanes of the vertex with another hyperplane from the system that is not currently used.

In the non simple case there can be more than d hyperplanes meeting in the same point. We call this a degenerate vertex. The principle when pivoting is similar, however we may need to remove more than one hyperplane. Likewise we may need to add more than one hyperplane. When removing one or more hyperplanes the idea is that the result is conceptually an edge of the polytope. That means the resulting set after removing some hyperplanes should only contain hyperplanes that touch each other outside the pivot. In more than three dimensions it is possible to have a degenerate edge, meaning the hyperplane that was removed had nothing to do with the degeneracy. When the hyperplanes of an edge has been found, add another hyperplane whose normal is linearly independent from the ones that make up the edge and which wasn't part of the old vertex. This produces a new vertex. Note that not all vertices found this way will be feasible, meaning they are not actually part of the polytope.[8]

4.2. Criss-cross Algorithm

David Avis and Komei Fukuda have created a criss-cross algorithm, which among other things can be used to enumerate all vertices of a convex polytope.[8] The algorithm is based on inverting finite pivot algorithms for linear programming.

By using pivoting a path can be found from any vertex to an optimum vertex that maximizes some objective function. The chosen path depends on which pivot rule is used to decide which vertex to pivot to. Avis and Fukuda uses the rule known as Bland's rule or the least subscript rule[9]. This rule guarantees a unique path from any vertex to the optimum vertex. The combination of all such paths form a spanning tree rooted in the optimum vertex.

The algorithm traverses this tree depth first by starting at an optimum vertex and reversing Bland's rule to follow the paths that lead away from that vertex.

4.2.1. Dictionaries

The algorithm uses the concept of dictionaries. A dictionary is a matrix-vector product. A system of equations, or conversely a system of hyperplanes, can be transformed into a dictionary. Pivoting can now be done as matrix transformation. Checking for feasibility still requires actually doing the reverse pivot, but then feasibility can be decided by checking signs of certain values in the dictionary.

Using dictionaries also means that only the vertices themselves are found, or specifically their coordinates. That means information about the which constraint gives a given face is lost, and so is information about strictness, since the algorithm only considers intersection of hyperplanes.

4.2.2. Algorithm Properties

The algorithm has a number of interesting properties.[8]

a) Virtually no additional space is required beyond that used to store the input

b) Each vertex is output exactly once

c) All degenerate cases are handled

d) No advanced data structures are needed

e) The running time is output sensitive for non-degenerate inputs

f) The algorithm is easy to efficiently parallelize

5. Problem Formulation

Visualizing zones from real-time models raises the question of what can be gained from such a visualization. However, in order to solve that problem, the vertices of a 3D representation of the zone must be found. This proved to be a challenge in and of itself and therefore there are two subproblems to consider.

- In order to visualize a zone, it is necessary to enumerate the vertices of the zone. Pivoting has been used for this task in the general case of inequality systems. How can the method of pivoting be specialized to the case of guards and invariants as known from models of real-time systems?

- When trying to get an overview of what can happen in a given timed model, one has to rely on the textual representation of the constraints. This makes it hard to know when an action will be possible, especially when evaluating a model. What can be gained in regards to correctness and speed by using visualization of zones when doing these tasks?

6. Design

This sections describes the abstract design of the solution and the things that had to be taken into consideration before making it. There are two main points. First there are the things that have to be considered in regards to the mathematical concepts used for the solution. Secondly there is the algorithm used for enumerating vertices of the zone.

6.1. High Dimensionality for Calculations

The representation created by this system is at most in three dimensions. Therefore it seems intuitive to only consider constraints of these three dimensions. However any constraint that can constrain one of these dimensions must be considered, and that means our intuition turns out to be wrong, and actually all constraints must be considered.

To understand why, consider only showing dimensions x and y, with a zone constrained by $y - z < 2$ and $z < 5$. There are no constraints on only x and y, however the first constraint means that y can't be greater than 7 unless z is greater than 5. Therefore y is constrained to being less than 7. Visually this is shown on figure 1, where y is bounded despite there not being a y less than bound, and when projecting this polytope to 2D it will become a rectangle where y is still constrained.

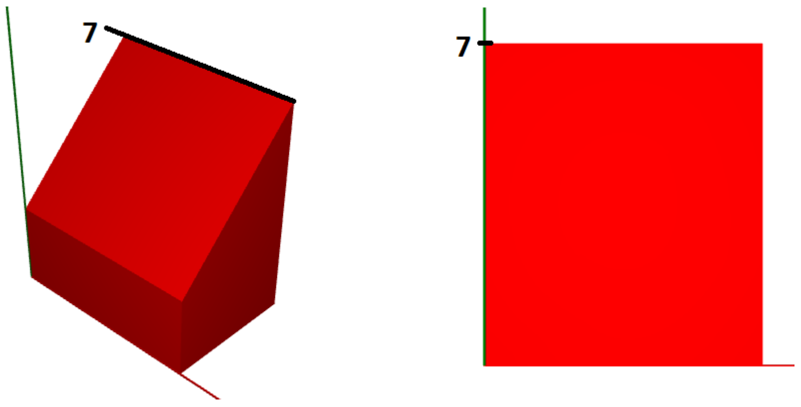


Figure 1: Example of 3D zone projected to 2D

This insight can be extended to further dimensions. In the example y is constrained by $y - z < 2$ and a less than constraint on z, however z can be similarly bounded by constraints on another dimension, thus meaning that how y is constrained depends on constraints in which the y dimension doesn't explicitly appear. This chain of dependency may be extended to all dimensions of the system in some cases, and therefore all constraints must be considered when calculating the zone. Afterwards the zone can be projected to the two or three dimensions that are shown in the system.

Evidently constraints can act as less than constraints on other dimensions (or reversely as greater than constraints). However they can also act similar to constraints on two dimensions, like $y - x < 5$. Consider the constraints $y - z < 5$ and $z - x < 5$. This is visualized in figure 2. This creates a slanted face effectively acting like $y - x < 5$ when projected to x and y.

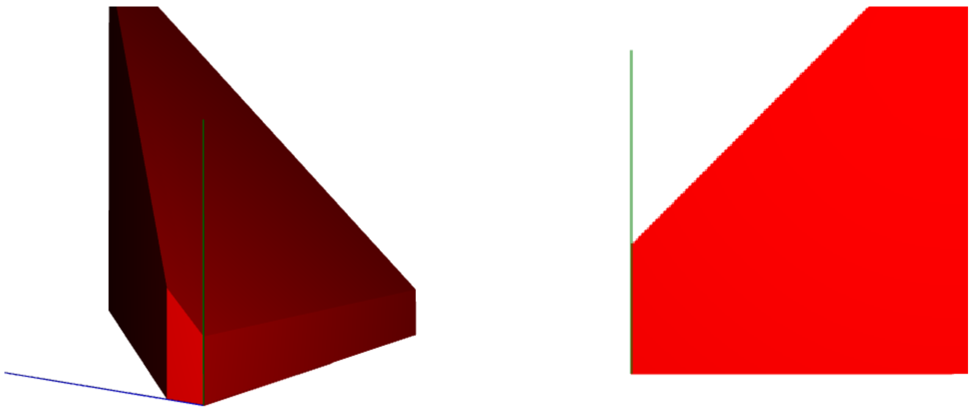


Figure 2: Example of 3D zone (shown rotated 90°) projected to 2D with slanted face

6.2. Pivoting Algorithm

The algorithm used for vertex enumeration is a pivoting algorithm. Pivoting can be seen as starting from one vertex and finding all edges going out from that vertex. Unlike Avis and Fukuda's algorithm[8] this algorithm doesn't transform everything into dictionaries, but works with the constraints themselves.

It should be noted that the algorithm assumes that there are no unnecessary constraints. In other words constraints that are less restrictive than others in every way and therefore doesn't intersect the zone. Given the underlying data structure in the model checking tools this is not a problem.

For the remainder of this subsection a constraint with $<$ can also refer to $\leq$, and $>$ can also refer to $\geq$ as they represent the same hyperplane, and therefore are equivalent when finding vertices.

6.2.1. Pivoting with Constraints

Consider each constraint as a hyperplane in n-dimensional space where n is the number of clocks of the system. A vertex is then the point where n hyperplanes intersect. Although a vertex can be seen simply as a set of n constraints with a single solution, we will consider that each constraint decides the value for one dimension.

A constraint on only one clock naturally decides the value of the corresponding dimension, whereas a constraint where one clock is subtracted from another can decide the value of either one of the corresponding dimensions, depending on which other constraints are in the vertex. In other words it will always decide the value of the dimension not decided by any other constraints.

6.2.2. Initial Vertex

In order to start pivoting, an initial vertex is needed. Additionally we would prefer to not visit vertices more than necessary. This can be done by starting from the vertex with the lowest value for all dimensions and only following edges that maximize one or more dimensions. For example removing an $x < 10$ constraint, can never maximize the x dimension as any vertex found must have a value for x that is lower than 10. In general upper bounds on a dimension should not be removed. This means that if a dimension x is decided by $x < k$ or by $x - y < k$ for some constant k and some clock y then it cannot be used for pivoting on that vertex.

In order to find the initial vertex we need the lowest value for all dimensions, that is still in the zone. If there exists an $x > k$ with some constant k for the dimension, then it is part of the initial vertex, as any other constraint acting as lower bound of that dimension would either make $x > k$ obsolete or give a lower value than k. A lower value than k is not feasible, since $x > k$ is too restrictive for that. If there isn't a constraint on that form then the x dimension is decided either by a constraint on the form $y - x < k$ for some clock y, or by the implicit $x \geq 0$.

In order to find the constraints of the initial vertex we use pivoting. We start with the vertex where each dimension is decided either by $x > k$ or by the implicit $x \geq 0$. This vertex might not be feasible, so we start pivoting. For each constraint c on the form $y - x < k$ we calculate the x value if we use c instead of what is used currently. If that value is greater than the old value, we pivot to the vertex where x is bounded by c. Since this gives us a new value for x we need to recalculate the value of all constraints on the form $x - y < k$ to see if they now give us a greater value for y, and so on.

Once values have been calculated for all constraints and there are no more with a greater value, pivoting is finished. That means the reached vertex is feasible, and will be the initial vertex used in the rest of the algorithm.

6.2.3. Finding Edges

Once an initial vertex has been found, we can pivot on that. An edge of the zone is where $n - 1$ hyperplanes intersect, in other words an edge consists of $n - 1$ constraints. An edge going from a vertex can be found by removing one of the constraints of the vertex. In other words for each dimension, remove the constraint deciding that dimension. The resulting sets of constraints are then the edges resulting from all possible pivotations on that vertex.

Note that some edges might go the wrong way, minimizing one or more dimensions. Therefore if the constraint to remove for some dimension is an upper bound on that dimension, then the resulting edge should be ignored.

6.2.4. Finding Potential Vertices

Once we have an edge we follow it to find potential vertices. A potential vertex is comprised of the constraints of the edge and some constraint that intersects with it. There will be one potential vertex for each constraint intersecting the edge, and exactly one of them is feasible, not counting the pivot. In order to find the constraints intersecting with the edge we first find the dimensions to look for upper bounds on. In other words which dimensions the edge is maximizing. We pivoted by removing the constraint deciding some dimension. Naturally that dimension is one which the edge is maximizing. Additionally the edge will maximize all dimensions for which a corresponding clock is part of a constraint on the form $x - y < k$ where the other clock x or y corresponds to a dimension the edge is maximizing. Note that this can add dimensions recursively, potentially until all dimensions are being maximized.

Once we know which dimensions are being maximized we find constraints that are upper bounds on those dimensions. Naturally any constraint on the form $x < k$ for some constant k, will be an upper bound for x. Additionally any constraint on the form $x - y < k$ is also an upper bound for x, but not all of the upper bounds containing two clocks intersect the edge. If the normal of a constraint is linearly dependent on the normals of the constraints making up the edge, then it does not intersect, but runs in parallel to the edge. This can also be found by checking if both x and y are being maximized by the edge. If that is the case then the constraint does not intersect the edge.

6.2.5. Finding a Feasible Vertex

When all of the potential vertices have been found we need to decide which one is feasible. The most restrictive constraints will give the feasible vertex, however several dimensions were potentially being maximized. Since all dimensions change at the same rate, the most restrictive in one dimension will be the most restrictive in all dimensions. Therefore any dimension can be chosen to decide which vertex is feasible, as long as it was being maximized by the edge. One dimension that we know was being maximized, is the one for which we removed the deciding constraint when pivoting. Therefore the feasible vertex is the one with the lowest value for that dimension.

6.2.6. Completing the Algortihm

The found vertex is feasible and can therefore be added to the zone. We keep doing this for the other edges found when pivoting. Then we start pivoting on the next vertex, which we just found, and so on, finding more and more vertices. At some point we will start finding vertices we have already found since there can be multiple paths from the minimum vertex to another vertex. If a vertex has already been found then it should not be added to the zone again, and it should not be used as pivot again if it has already been used once. This means the queue of vertices to pivot on will eventually be empty, as all vertices have been found and used as pivot. Then the algorithm is done.

6.2.7. Degenerate Vertices

A problem that has not yet been addressed is the issue of degenerate vertices. As mentioned in 4.1 a degenerate vertex is one where the number of dimensions is lower than the number of constraints for which the vertex is a solution.

Because of the chosen representation for a vertex, there will be a at least one dimension which is decided by multiple constraints. Since a constraint with two clocks may be a bound on either one of the corresponding dimensions, such a constraint may be deciding either dimension in a degenerate vertex. Indeed which dimension it is assigned as a bound on depends on which vertex was used as pivot to find it. Care must be taken that such a difference doesn't lead to it being recognized as a new vertex.

Actually finding a degenerate vertex is a fairly simple adjustment when following the described algorithm. When following an edge, if there are several potential vertices which result in the same value, then merge them into one vertex which includes all of the constraints intersecting the edge. It is important to not only do this when following the edges after pivoting but also when finding the initial vertex, if several constraints give the same value.

The other problem with degenerate vertices appears when they are are used as pivots. First of all there may be multiple constraints deciding a dimension, so they all need to be removed when removing the constraint deciding that dimensions. However if any of them is an upper bound then ignore the pivot. As described in section 4.1 actually using a degenerate vertex as pivot is also slightly more complicated. For each dimension d when removing the deciding constraint, also remove all constraints for which there is another constraint deciding the same dimension, if it can also be a bound on d, or if a dimension decided by multiple constraints, is decided by a constraint on the form $x < k$ or $x > k$ then remove all other constraints on that dimension, unless the resulting edge is degenerate, which it is if none of the constraints on degenerate dimensions are removed.

Some extra edges have to be found since the constraints that all decide the same dimension also touch each other. For each dimension that is decided by more than one constraint, create an edge containing those constraints as well as any constraints from the pivot that do not share any clocks with them.

After finding all the edges the procedure is exactly the same as described in section 6.2.3.

6.2.8. Calculating Coordinates

When the constraints of a vertex are known, calculating the coordinates becomes trivial. To calculate the coordinate of a given dimension x, we find the value from a constraint c deciding that dimension. If c is $x < k$ or $x > k$ then the coordinate of x is k. If c is $x - y < k$ then the coordinate of x is $k + y$ and finally if c is $y - x < k$ then the coordinate of x is $y - k$. This may require calculating y in order to calculate x, which may in turn require calculating the coordinate of another dimension, and so on recursively. However each vertex has at least one constraint of the form $x < k$ or $x > k$, which means calculating the coordinate of a given dimension may require calculating up to all coordinates for that vertex, but it is always possible.

7. Implementation

A solution has been implemented in Java. This section will describe the choices made when implementing the solution. It will also explain how the problems encountered during implementation were solved.

7.1. JavaFX

The solution was written with Java Development Kit (JDK) 1.8 update 172. The GUI for the solution was created using the JavaFX platform. This was done mainly because the new front end for ECDAR 2.0 is using JavaFX, and so it can easily be integrated as part of the toolbox. Using JavaFX also led to using Java 8 rather than Java 9 because certain packages did not behave as expected in Java 9.

JavaFX is a platform for creating applications in Java that work across a variety of platforms. It is integrated in the JDK and contains a set of packages for creating graphics. Among other things it can be used to create 3D-objects and show them in the application.

7.1.1. 3D Controls

Some challenges did arise from the use of JavaFX. The graphics packages are primarily developed for 2D applications, and many 3D features are still somewhat lacking. For example the ability to move the camera of the 3D scene does not behave as expected, when moving up, down or to the sides. That means only zoom could be easily implemented. The effect of moving the camera around an object was created by adding everything as children of a single parent node in the 3D-space and then applying a rotation transformation to that node, while the camera was looking directly at it.

7.1.2. Showing Text in 3D-space

JavaFX has labels and similar constructs for showing text in 2D. These can be transformed into a 3D-space, however it is not well supported and the behaviour becomes unpredictable. There are no constructs with the main purpose of showing text in 3D. The result is that there are no labels in the 3D-scene of the solution, because there was not enough time to find a workaround or find a working third party package to solve the problem.

7.1.3. Depth Buffering

Another challenge was that depth buffering does not work with transparent 3D-objects. This is a problem since it would prevent making the zone a little transparent to enable users to see if something is inside it. However the problem can be ignored by manually ensuring that the zone is added to the scene after everything else. This way the zone is drawn last and therefore on top of what is already there, as it should be.

7.1.4. Backface Rendering

A further complication happens because backfaces of a 3D-object are not rendered. A 3D object is made up of triangles called the faces of the object. Each face has a normal specifying what is the front of the face. The front is the only side that can be rendered with a material, which means the back is either not shown (culled), or shown as a pitch black triangle. In order to handle this, each face of the zone is actually two objects with opposite normals. However that causes problems with the depth buffering described in subsection 7.1.3. Therefore the backfaces also have to be added to the scene before the front faces, but after other objects in the scene.

7.2. Pivoting Algorithm

There were some challenges encountered during the implementation of the pivoting algorithm described in section 6.2.

7.2.1. Data Structure for Vertices

One challenge was how to efficiently check whether a vertex had been found before. An AVL-tree had been used for a similar task before[8], however for the sake of simplicity a hash table was used in the current implementation. A more advanced data structure may be preferable in the future, however the efficiency was not a problem in regards to evaluating the system.

The hashes of vertices is calculated as the hash of the set of constraints. This can be done because the constraints are unique. There are not two different objects representing the same constraint.

7.2.2. Collections of Constraints

During pivoting the same sets of constraints will often be needed over and over again. For example whenever an edge is maximizing x, all upper bounds on x are needed. Therefore a number of collections were kept, such that once the constraints had been found once, a mapping was added from the dimension being maximized to the collection of constraints that were upper bounds on the dimension. Note that this was only necessary for the upper bounds, because lower bounds only needed to be found once, when the initial pivot was found.

7.2.3. Early Implementation

During development of the algorithm, both the algorithm and therefore the solution went through many iterations, sometimes with many changes. This included changing the implementation so that the different steps of the algorithm were carried out where they made sense.

At one point the implementation had to be redone almost from scratch, because many classes had ended up doing things that no longer made sense to do in them. What had started out as simple tasks had in multiple places become things that should be their own classes, and even worse the information flow through the solution had become cluttered as information was passed back and forth.

Specifically the task of finding potential vertices, and then deciding which one was feasible, was split over many classes. Almost all the information had to be sent along to all classes, and it was rarely obvious what the purpose of the class was and what should be returned after a step.

Therefore a new class structure was created and implemented, replacing most of the previous implementation.

7.3. Rendering of a Polytope

JavaFX cannot readily use the result of the pivoting algorithm to render the polytope.

7.3.1. Zone Representation

The result of the pivoting algorithm is a zone consisting of vertices and faces. A vertex of the zone is a mapping where each dimension is mapped to a value. There is no particular ordering. A face is a convex polygon represented by an unordered set of vertices outlining the convex hull of the face. Additionally a face also knows the constraint that it represents, from which the normal of the face can be found. The normal is also a mapping where each dimension is mapped to a value.

7.3.2. JavaFX 3D-object Representation

JavaFX represents a 3D-object as a mesh consisting of vertices, faces and normals. A vertex is three numbers and the order decides which dimension each value is associated with. A face is three vertices and their order should be counterclockwise in relation to the normal of the face. A normal is represented as three numbers. Again the ordering of the numbers decide which dimension each value is associated with. The ordering of the normals also decide which face each normal is associated with, so the first normal is for the first vertex and so on.

7.3.3. Projecting a Zone

The first step in enabling JavaFX to render the zone is projecting it to three dimensions. Projecting a vertex to 3D is simple. A vector in 3D for the vertex with valuation v can be found as $(v(x1), v(x2), v(x3))$ where $x1$, $x2$ and $x3$ are the ordered set X_P of dimensions to project to. The problem then becomes finding which vertices to project.

It doesn't make sense to project the vertices of a face which isn't bounding one of the dimensions to project to. In other words if the normal of the face is $\mathbf{0}$ when projected to X_P, then it won't make sense to draw it.

If a face represents a constraint where all the clocks bounded by it are in X_P, then it should simply be projected by projecting all vertices in the face. If a face represents a constraint with two clocks, and only one of the clocks is in X_P then extra care must be taken. I should be projected to a single plane in the dimension which is in X_P, and there may be two faces taking up the same space, in which case only one of them should be drawn.

7.3.4. Ordering Vertex Values of Faces

Once the vertices of a face have been projected to the desired dimensions, the face needs to be triangulated. The triangulation of a convex polygon is trivial, however before getting that far an ordering of the vertices must be found. JavaFX requires the vertices of the triangles to be ordered counterclockwise so that ordering will be used for the whole set. For the sake of simplicity all vertices are assumed to be defined locally in relation to the center of the face while calculating the ordering.

The normal $\vec{n}$ of the face is known. It can be projected the same way a vertex was. A reference $\vec{r}$ in the face is needed to be used as 0 degrees. The vector to any vertex of the face can be used as reference. The chosen vertex will be the first in the ordering. To define a direction another reference is needed to differentiate partitions. This can be found as $\vec{p} = \vec{n} \times \vec{r}$, since this will give a vector in the right plane, in a 90° angle from $\vec{r}$.

For a vertex $\vec{v}$ we can calculate $\vec{v} \cdot \vec{p}$ to show if the vertex is in the first or second 180° from $\vec{r}$. Special care must be taken if the scalar product is 0, since it can mean the vertex is at 0° or 180°. Once we know know which half of the space a vertex $\vec{v1}$ belongs to, we can compare it to another vertex $\vec{v2}$ in the same space by calculating $(\vec{v1} \times \vec{v2}) \cdot \vec{n}$. If this value is lower than 0 then $\vec{v1}$ should come first in the ordering.

7.3.5. Triangulation

Once the vertices are ordered, the polygon can be triangulated by choosing the first three vertices as a face, and then removing the second vertex from the set of vertices. By repeating this until there are less than 3 vertices left, all faces are found, and the ordering of vertices in each face is counterclockwise.

8. Examples from the Solution

This section will show examples from the implemented solution and explain how the main features are used.

Figure 3 shows how a zone is visualized in the solution.

8.1. Content Area

The primary panel in the window is the right panel, where a zone is shown in 3D-space. Inspiration for how to manipulate the view has been taken from other applications showing 3D-objects, like the game engine Unity®[10]. Users can zoom using the mouse wheel and pivot the camera around the center of the zone by dragging the mouse while pressing a mouse button.

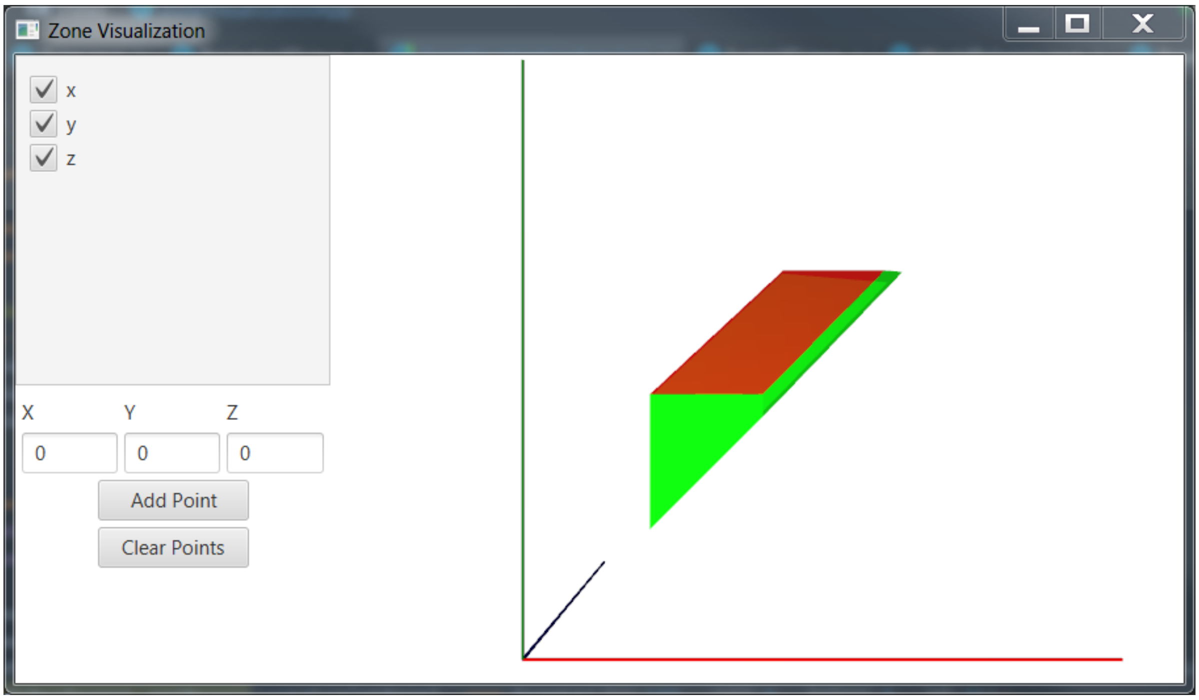


Figure 3: Showing a zone in the solution

As seen in figure 3 the zone has red and green faces. A red face represents a strict constraint, and a green face represents a non strict constraint. Meaning the green faces themselves are part of the zone, while the red ones are not. This also allows the user to see whether an edge or a vertex is in the zone. If even a single face intersecting there is strict then they are not included in the zone.

As can also be seen in figure 3, the axes of the coordinate system shown have neither values nor labels to indicate which clocks they represent. As mentioned in subsection 7.1.2 this was not easily accomplished, and it is a known problem with the solution. The clock corresponding to an axis can however be shown by hovering over the axis.

8.2. List of Clocks

The left side of the window shown in figure 3 shows a menu with panels for further affecting what is shown in the content area. In the upper left corner is a list of the clocks in the model. Here the user can choose which dimensions to project to. In the example there are three clocks: x, y and z that are all selected.

By selecting only two clocks it is possible to have the zone projected to only two dimensions as shown in figure 4. However the 2D projection does not indicate whether an edge is strict or not, and the color of the area doesn't mean anything. It is always red.

Selecting less than two clocks has no effect and it is not possible to select more than

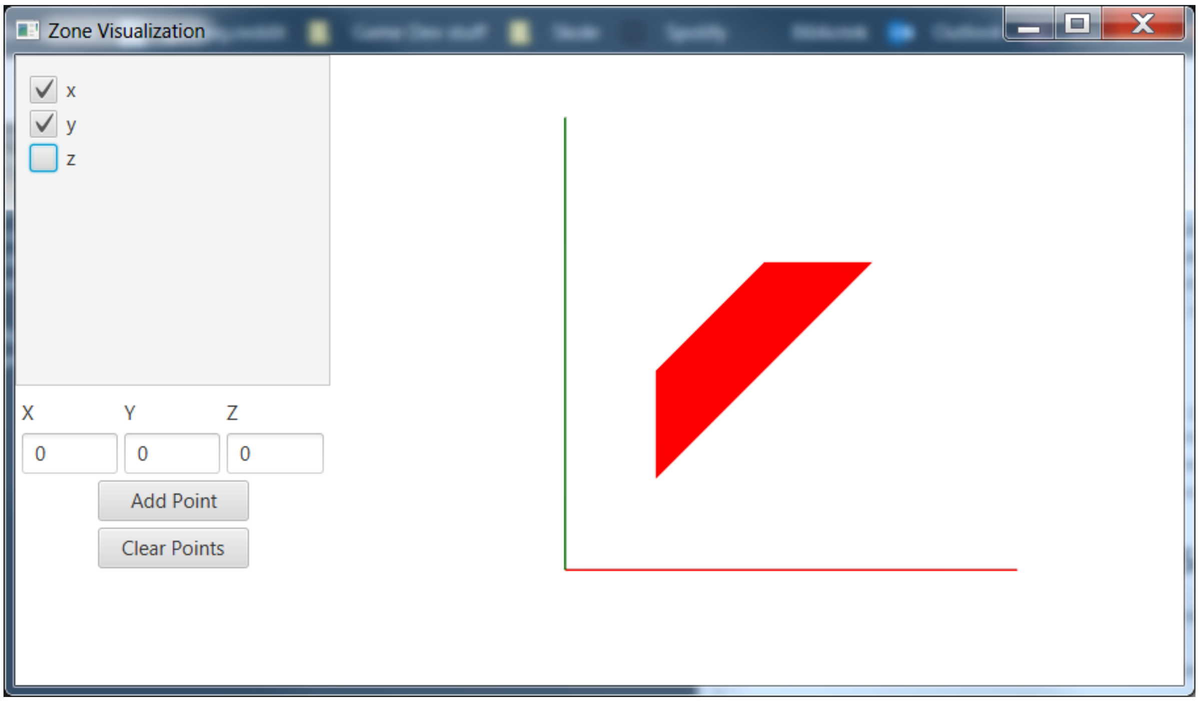


Figure 4: Showing a zone that has been projected to 2D

three.

The order in which dimensions are selected matters. The first clock will become the x-axis (red), the second one the y-axis (green) and the third one the z-axis (blue) when shown in the content area. This also means removing for example the clock y in the example, and then readding it will change the order, and therefore the projected zone, because y would become depth, rather than height.

8.3. Adding Points

It is also possible to insert points into the 3D-space. This is done through the panel below the list of clocks. You simply write the coordinates of a point and press the "Add Point" button. This can be done repeatedly to insert more points, and the result can be seen in figure 5.

Adding points can be used to realize whether a specific valuation is inside the zone or not. Additionally it is a much needed tool while there are no values on axes, since it is the only way to see specific values in the solution.

There is no limit on how many points can be inserted by the user, other than how many their hardware allows them to render. However at any point all the inserted points can be removed again by pressing the "Clear Points" button.

Inserting points does not work well with changing the order of the dimensions to project to. The points will remain in the same position in the shown coordinate system,

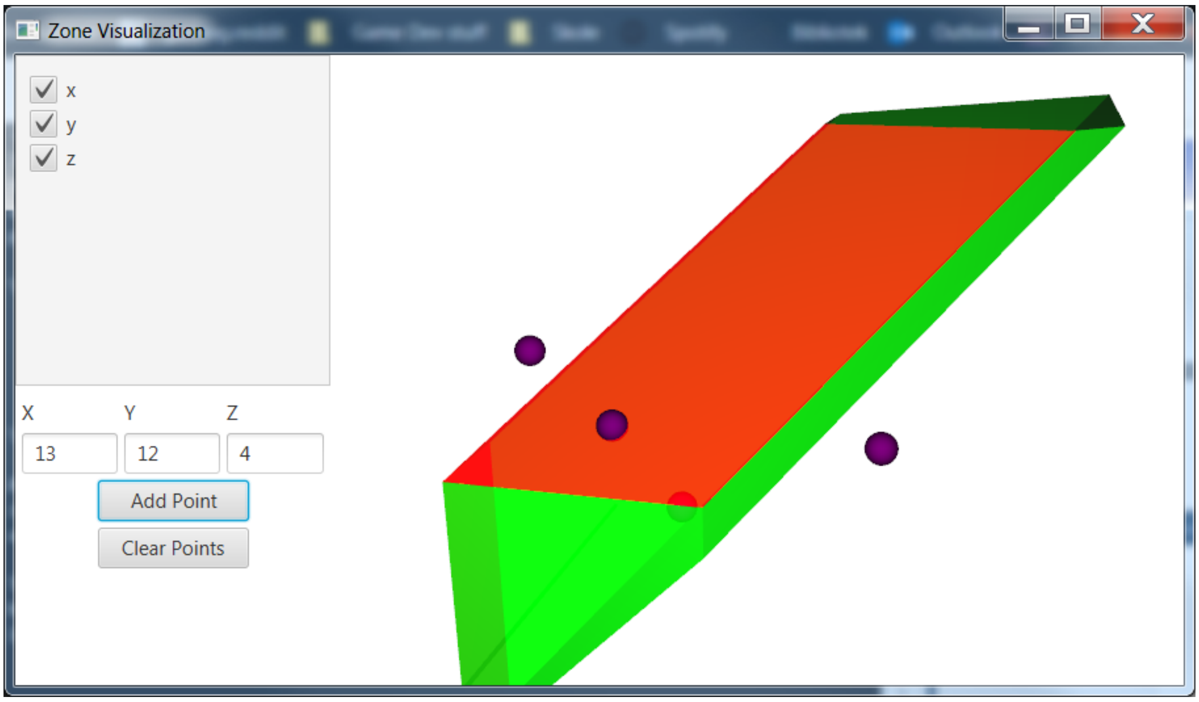


Figure 5: Shows 4 points. 2 outside the zone, 1 inside and 1 in a face

and not be affected by the new ordering. Combining these two features in a meaningful way is difficult since not just the ordering can be changed, but which dimensions being projected to can be changed as well. Instead the points should be cleared when changing dimensions.

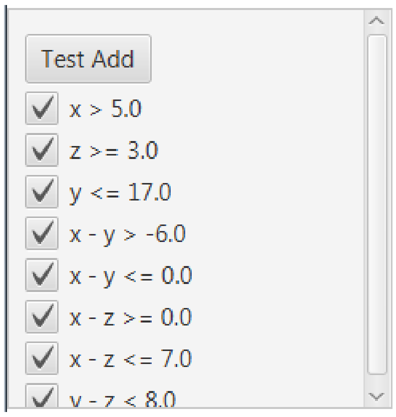
8.4. Input Panel

Because the tool cannot currently receive its input data directly from a modeling and verification tool, there is actually another panel, which is not intended as part of the finished tool. In figure 3 this panel is hidden under the edge of the window in the bottom left corner. The panel is used to show a zone made up of a subset of the constraints of a hard-coded zone, which has been used for testing. The panel is shown in figure 6.

9. Evaluation Method

The evaluation of the system will focus on what can be gained from visualizing zones of a timed model. The evaluation will compare how test subjects solve tasks with insights gained only from a textual representation of constraints compared to insights gained from a visual representation of a zone.

The evaluation method used for this project is inspired by the one described by Kjeldskov et. al.[11]. This method is well suited for getting quick results with only 4 to


Test Add

- ✓ x > 5.0
- ✓ z >= 3.0
- ✓ y <= 17.0
- ✓ x - y > -6.0
- ✓ x - y <= 0.0
- ✓ x - z >= 0.0
- ✓ x - z <= 7.0
- ✓ y - z < 8.0

Figure 6: A panel that can set a hard-coded zone to be shown in the content area

6 test subjects. They use the think-aloud protocol along with their own Instant Data Analysis (IDA) technique to evaluate a software system. However that technique is used for evaluating the usability of a system, whereas this project focuses on how insights gained from a system may help in solving tasks. Therefore the focus during evaluation will not be on how test subjects interact with the system, but rather how fast they can solve the tasks, as well as how correct their results are.

The think-aloud protocol can still be used to understand what insights users gain from the system, as well as what insights they are still missing, in regards to the task they are solving.

One researcher participated in the evaluation. Unlike IDA where there are three different roles, only one was used here, resembling mostly the role of test monitor. The researcher also took notes on what the test subjects remarked during the evaluation. These notes were used to understand what users gained from the system in the following brainstorm.

There will be two data sets gained from this method of evaluation. The time and correctness for each task make up the first data set, and the second data set consists of the findings from the following brainstorm session.

9.1. Participants

The participants of the evaluation were 6 test subjects working to solve the given tasks with one researcher facilitating the evaluation as test monitor.

The 6 test subjects were all male computer scientist and software engineering students from AAU with knowledge of timed models ranging from basic knowledge from courses, to having worked with it for one year.

Since the visual tool makes use of red and green colors it should be noted that one of the test subjects was red-green color blind. He was however able to distinguish the two colors in the tool, sometimes it just took a little longer.

9.2. Setting

The evaluation sessions were conducted with the test subject and the test monitor sitting next to each other. The test subject was sitting in front of a table with a standard laptop and a conventional keyboard and mouse setup. The system had been preinstalled, and before each task a model was loaded into the system. Only the relevant representation was shown on the screen.

Additionally a voice recording device was placed between the test subject and test monitor to record the conversation.

9.3. Tasks

The test subjects were given 12 tasks. The goal in the tasks was to decide whether a given transition in a system of timed automata is possible. The transition had guards that either only allowed a specific valuation of all clocks or a zone of valuations. When only a specific valuation was allowed the test subjects had to figure out if the corresponding point was inside the shown zone. When a zone was allowed the test subjects had to figure out if the zone intersected with the shown zone. 6 of the tasks had only one timed automaton, while the other 6 had two timed automata. For each of these 6 tasks, 3 had guards that allowed only a specific valuation, and 3 had guards that restricted to a zone. Each of the set of 3 tasks had one where the transition was not possible, as the values needed to take the transition was outside the zone. There was one task where it was possible, in other words the valuation was inside the zone. The last task had the valuation on the edge of the zone so whether it was possible to take the transition depended on whether the constraints were strict or not. The given tasks can be seen in appendix A.

In order to reduce bias, test subjects alternated between solving a task with only the textual representation of constraints and solving a task with the visual representation. This was done so one method wouldn't seem better because the test subjects were getting better at understanding the models in general. Additionally half the test subjects started by solving the first task using only the textual representation, while the other half started with the visual representation. This way all tasks were solved an equal number of times with each method.

The tasks were solved in the same order by all test subjects, meaning they might have had a greater understanding of how to use the models in later tasks. However the tasks varied in difficulty so comparing tasks to each other makes little sense regardless. Rather the two methods should be compared for each task. Therefore randomizing the order of the tasks should be unnecessary and would only complicate the process of ensuring all tasks were solved an equal number of times with each method.

9.4. Procedure

The evaluation was conducted using the following procedure.

Briefing and introduction. First the test subjects had the evaluation process and the system explained to them. Then they answered a few questions about their demographic profile.

Task solving. The test subjects were given the tasks one at a time and worked on them until they had solved them or until four and a half minutes had passed, in which case the test monitor asked them to move on. The test monitor logged whether the task was completed, and if so whether it was solved correctly. Additionally the test monitor noted the time the task took.

Thinking-aloud The test subjects were asked to think aloud and say how they were solving the tasks, especially what they were learning from the information shown on the screen. The test monitor did not help or provide extra information but could ask for clarification.

Debriefing At the end of each evaluation session the test subject was debriefed about their experience with the system and the evaluation.

9.5. Data Analysis

After the evaluation sessions there was a brainstorm session similar to the one used in IDA[11]. However there was only one researcher and the focus was not on usability, but still rather on what users had gained from using the system. The task descriptions, voice recordings and notes taken during evaluation were used to assist the brainstorm. The result was a list of insights test subjects were able to gain with each of the two methods, ranked by how important they were to solving the task, and another list with information the test subjects still found lacking at some point in their task solving process, ranked by how much more difficult they found the task due to the lacking information.

After the brainstorm session the average time and completion of each task was compiled into a table.

10. Evaluation Results

The evaluation resulted in two data sets. The first data set is quantitatively showing how well and how fast test subjects were able to solve each task. The other data set is qualitative and meant to show what was found to be useful, or missing in the tool during task solving.

10.1. Time and Correctness of Task Solving

The full data set of time and correctness for each task can be seen in appendix B.

Table 1 shows how many times each task was solved correctly with each method, and the average time of the cases when the task was solved correctly. The time limit for task solving was only reached once, which was when solving task 4 with the visual representation.

The times for solving tasks incorrectly are not considered because errors when using the textual representation often come from jumping to conclusions too quickly, either because of a wrong method or because only one constraint needs to be unsatisfied for the transition to be impossible to take, and so test subjects could stop after finding just one with the textual representation. However when they were wrong they actually should have made more calculations. This was not an issue with the visual representation because the point was shown in relation to all constraints at once.

There were a number of cases where the right conclusion was reached despite errors or the use of a wrong method. This happened with both representations, however the test monitor estimates that it happened 2 to 3 times with the visual representation and at least 5 times with the text representation. Specifically one test subject used a wrong method when using the text representation and yet he got the correct answer 4 out of 6 times. Getting the right answer despite errors is a problem with the textual representation because as mentioned a conclusion was often reached prematurely. Therefore some of the times for the textual representations are shorter than they should be. The test monitor estimates this happened at least 3 times.

	Textual Representation		Visual Representation	
	Correct	Avg. Time of Correct	Correct	Avg. Time of Correct
Task 1	0 / 3	-	2 / 3	0:57.42
Task 2	3 / 3	1:20.37	3 / 3	1:19.09
Task 3	2 / 3	0:55.31	3 / 3	1:04.92
Task 4	2 / 3	1:20.92	1 / 3	1:56.31
Task 5	1 / 3	1:12.90	3 / 3	1:19.44
Task 6	3 / 3	1:29.59	3 / 3	0:47.28
Task 7	3 / 3	1:35.70	3 / 3	1:04.00
Task 8	2 / 3	0:44.14	2 / 3	0:33.99
Task 9	3 / 3	0:58.35	3 / 3	0:20.41
Task 10	3 / 3	0:45.23	3 / 3	0:30.16
Task 11	3 / 3	2:11.97	2 / 3	1:04.95
Task 12	2 / 3	0:56.56	3 / 3	1:05.01
All	27 / 36	1:13.73	31 / 36	1:00.25

Table 1: The correctness and average time aggregated for each task

The average times for each task shown in table 1 are on average lower for the visual tool. When using the T-test[12] on the average times for each task (except task 1) we get $t = 1.16909$ and $p = 0.128057$ meaning the average times are *not* significantly lower for the visual tool at $p < 0.05$.

One thing to note is that generally more time was spent on understanding the task, and considering which method to use when solving the first tasks, while more time was spent on actually solving the tasks when solving the later tasks. Therefore the results of the T-test on the average times of just the last 6 tasks could be interesting as well.

Here $t = 1.97315$ and $p = 0.03598$ meaning the average times are significantly lower for the visual tool.

10.2. Brainstorm Results

During the brainstorm session the observations and findings from the evaluation were considered. The notes from this session can be found in appendix C. The points found during the session were compiled into two lists. The first list contains the benefits that were gained from using the visual tool for the tasks. The second list contains things that the test subjects found lacking or that might have helped them while solving the tasks.

Benefits of the visual tool

- Easier to see if a specific point is in the zone
- Overview of what is possible
- Strict and non strict constraints are intuitively understood at a glance
- It is easy to realise which values need to change and in which direction to get a point in the zone after inserting a point
- Multiple constraints with two clocks affecting each other does not make the problem more complicated
- Enables an explorative approach
- Users were more confident in their answers
- Less mental strain

Shortcomings of the visual tool

- Values on axes
- Names of clocks on axes
- Clarity on whether a point is inside the zone or not
- The ability to draw custom zones and not just points
- Show coordinates of vertices

Causes of Errors

The causes of errors using the two methods were also considered. The most common causes of errors in order of how common they were:

Textual Representation

- Used wrong method or formula when calculating constraint satisfaction
- Arithmetic errors
- Misread the assignment and used > instead of <

Visual Representation

- Thought delays could only be integers
- Misread the assignment and used > instead of <
- Thought a point was inside the zone when it was actually behind it

11. Discussion & Conclusion

This section will be used to discuss the proposed pivoting algorithm for enumerating vertices of a real-time systems zone, and how the visualization of a zone can help in getting an overview of a real-time system.

11.1. Discussion of the Proposed Algorithm

The algorithm described in section 6.2 proves that the zone of a real-time system has some properties that can be used when designing a pivoting algorithm. However the question is what can be gained from using the new algorithm over the one described by Avis and Fukuda[8] which was presented in section 4.2.

There has not been any formal comparison or benchmarking of the two algorithms to see which one performs best. However many of the nice properties of the original algorithm will have been lost in the new algorithm. The new algorithm will not be as space efficient since it saves the reported vertices to be able to check whether a vertex has been found before. Additionally it also saves all edges from a given pivot, before enumerating them which essentially means the new algorithm is a breadth first search of the spanning tree. All in all the space complexity of the new algorithm will inevitably be worse.

The property of only outputting each vertex once also holds for the new algorithm, although as mentioned this is at the cost of checking whether a vertex has already been reported when it is found.

The new algorithm is able to handle degenerate cases as well, although not in the most elegant way. One could argue the original algorithm has problems with this as well, at least in the case where the initial vertex is degenerate. However the original algorithm has used formal proofs to guarantee that all degenerate cases are handled, which has not been done for the new algorithm, so there is still a risk that there may be some edge case that is not handled.

The new algorithm does need to use advanced data structures. Although all concepts pertaining to the polytope can be expressed as collections of constraints, the collection of found vertices needs a data structure so it can efficiently be determined whether a vertex has been found before.

The running time of the new algorithm is still output sensitive, since all output vertices are also used as pivots. However the running time is hard to compare to that of the original algorithm since a complexity analysis has not been performed on the new algorithm.

As to parallelizing the algorithm it is possible and some performance can be gained, especially when working in many dimensions, as this leads to a broader spanning tree of the polytope vertices. However it is not trivial to do so, whereas it is easy with the original algorithm.

However some things have been gained by using the new algorithm. One thing that is quite important for the case of visualization is that the algorithm can easily output the faces of the polytope as well. Another thing which proved very important is that the constraints are used throughout the system, which meant a face could be colored according to whether its corresponding constraint was strict or not.

Obviously losing most of the nice properties of the original algorithm is bad. Since both the space complexity, and likely the time complexity as well, are worse for the new algorithm, and they both grow with the number of clocks, the original algorithm will almost invariably be better when a model has many clocks.

However in most cases this matters little, since even fairly big models in UPPAAL doesn't have that many clocks, and so correctness and actually having more information available for visualization becomes the main factors. The original algorithm is still better when it comes to correctness, since it has proven that all cases are handled, even all degenerate ones. However the new algorithm has more information available.

In the end the benefits of the original algorithm means that it might be easier to find a way of adapting that algorithm to also carry some extra information, and make it available after finding the vertices, but it could also still be possible to find a reversible pivot rule and do a depth first search in the new algorithm, which would give it most of the nice properties again.

11.2. Discussion of the Zone Visualization & Evaluation

The results presented in section 10.1 show that the test subjects were able to get more correct answers with the visual representation than with the textual representation currently used by UPPAAL. Additionally the tasks were on average solved faster using the visual tool, although the difference was not significant.

Looking closer at the results we find that the majority of errors were in the first tasks, and actually both methods caused the same amount of errors in the last 6 tasks. This indicates that the visual representation is more intuitive, whereas the second or third time the test subjects met a type of task they had often found a good method for solving it, regardless of which representation they were using. However the average times for solving the tasks were significantly better when using the visual tool for the last 6 tasks.

The results speak even more in favor of the visual representation when considering that more errors leading to the correct result, were observed when using the textual representation. It is hard to know to which extent this happened but it does indicate that the difference of errors is even higher than 4.

Similarly as explained in section 10.1 some of the times were actually lower than they should be for the textual representation. It is also hard to know how much of a difference this makes, but it does make the difference greater.

Another factor to consider is the fact that only considering the times of correct answers might also introduce bias. It was done because wrong answers could be reached faster, however it may also result in only getting the times of either the most thorough or of the ones who are best at solving the tasks, and therefore do it faster. For example the average time of solving task 1 with the textual representation is over 3 minutes with three wrong answers and around 1 minute with the visual representation with only one wrong answer, which was also the slowest, although not by much. However while this does seem to have influenced the average times, it doesn't change the fact that the visual representation was faster. On the contrary it would have had the greatest effect on the average times of the textual representation and made them greater. The average times including wrong answers were 1:33.09 for the textual representation and 1:04.14 for the visual representation.

Considering the causes of errors found during the brainstorm we also find that the most common errors for the textual representation was because of the method needed when using that representation. The visual representation on the other hand mainly had errors from misconceptions about the zone and from misreading the assignment. Only one error for the visual tool was actually because of an error in using the tool, and that error might indicate that some work is needed on the usability of the tool. Some of the errors might even be helped if more information is sent from the model checking tool and some of the missing features were added. For example being able to draw a zone rather than a point would make it easy to see that $x > 5$ was accidentally used instead of $x < 5$.

This leads us to the findings of the brainstorm. There was some information lacking from the tool when it was used to solve tasks during the evaluation. Some of it was known beforehand, like the fact that axes were missing values and clock names. Some were features that test subjects suggested to solve some problem they had encountered. Finally there were some features where it became apparent that they would be useful as several of the test subjects were struggling with the same part of some task.

The benefits of the tool were also quite apparent. Once the test subjects had tried the visual tool they became very fast and very confident in their answers when inserting a point and checking if it was inside the zone or not. They were also able to realize whether a zone was intersecting in tasks 4-6 and 10-12 by inserting a point where the guards intersected and checking greater or lower values.

For the test subjects that were unsure of how to proceed, the visual tool also got them started very quickly as they started inserting points seeing how different values were placed in relation to the zone and eventually reaching a conclusion. This gave them a better understanding of the zone they were working with, eventually also leading to them finding more efficient methods for the later tasks.

Finally there is the issue of strict and non strict constraints, which users seemed to grasp quite easily within the visual tool. This is important as test subjects expressed concerns that it was hard to keep track of strictness while doing calculations, and therefore they felt they might forget them. Therefore having no problems with them in the visual tool, and simply checking the color when it becomes relevant, is another important benefit.

11.3. Conclusion

The solution implemented for this report showed how a zone could be visualized, and how such a visualization might help in understanding a model and its zone. Through evaluation the tool proved to be better regarding both time and correctness than using the textual representation that is currently the only option. Despite some features that the tool is still lacking, it also let the test subjects easily get started and have more confidence in their answers.

The algorithm implemented in the solution however did not actually have many benefits over the one that inspired it, as it was likely to perform worse in most cases, and had worse space complexity. It did however provide all the needed information for the features that were implemented in the tool.

The conclusion is that zone visualization would definitely be a powerful tool to add to the symbolic simulator of a real-time model checker. However more work is needed to determine how best to enumerate the vertices of the zone.

12. Future Work

In the future the findings of this report can be expanded upon in several areas. One area to work on is the practical application of implementing the visualization tool in a model checking tool like UPPAAL or ECDAR.

In the same vein it would also be useful to work on the usability of the tool. Usability has not been a focus of this project, however during the evaluation several usability problems came up, and more can surely be found in a proper usability evaluation. If the tool was to be integrated into one of the model checking tools, then solving these usability problems would be an important contribution.

Another way to improve the tool and the way it can be used to understand a model is by adding the features that were already identified as lacking in section 10.2. By adding names and values to the axes a user would be able to gain far more information about the zone at a glance. If it was possible to draw a user defined zone or the plane of a single constraint, it would also be easier to understand how a guard interacted with the zone. A feature that has not been discussed before is animation of the zone. It might help in understanding a transition if the zone isn't simply shown before and after but an animation shows how it transitions from one to the other.

A completely different avenue is to research algorithmic improvements. There are multiple ways to to try and improve the algorithm described in section 6.2. Inspiration can be taken from Avis and Fukuda[8] to try and find a reversible pivot rule, and so avoid checking if a vertex has been found before. Additionally the algorithm can be changed to a depth first algorithm to improve space complexity. Another thing to examine is whether the fact that all vertices have positive integer coordinates can be utilized. Finally it would be possible to adapt Avis and Fukudas algorithm[8] using dictionaries, in such a way that it too can provide the information needed during visualization. This might also include examining how dictionaries are affected by the limitations of real-time model zones, and how that can be utilized.

12.1. Known Issues

A number of issues have also been identified in the current solution.

12.1.1. 2D Projection

When only two dimensions are chosen, the zone is projected to two dimensions. However if more than one dimension is going to infinity, in other words if (∞, ∞) is in the zone, then the infinity is not visualized correctly. This is strictly a projection bug.

12.1.2. Degenerate Vertices

There is a specific case where a degenerate vertex can lead to a bug. The exact cause has not been found, but it causes extra vertices to appear. All the right vertices are also there, but some pivot seems to have gone wrong leading to an upper bound being ignored so the zone goes to infinity instead.

References

[1] Kim Guldstrand Larsen. Formal methods for real time systems, automatic verification & validation, nov 1998.

[2] Roberto Gorrieri. *Labeled Transition Systems*, pages 15–34. Springer International Publishing, Cham, 2017.

[3] Rajeev Alur. Timed automata. In Nicolas Halbwachs and Doron Peled, editors, *Computer Aided Verification*, pages 8–22, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[4] Kim Guldstrand Larsen, Paul Pettersson, and Wang yi. Uppaal in a nutshell. 1:134–152, 12 1997.

[5] Casper Møller Bartholomæussen, Tobias Rosenkrantz Gundersen, Rene Mejer Lauritsen, and Christian Ovesen. Ecdar 2.0 a new integrated modelling and verification environment for compositional real-time systems. Technical report, Aalborg University, jan 2018.

[6] Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Uppaal-tiga: Time for playing games! In Werner Damm

and Holger Hermanns, editors, *Computer Aided Verification*, pages 121–125, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[7] Niklas Kirk Mouritzsen and Rasmus Holm Jensen. H-uppaal. 2017.

[8] David Avis and Komei Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra. *Discrete & Computational Geometry*, 8(3):295–313, Sep 1992.

[9] Robert G. Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2(2):103–107, 1977.

[10] Unity Technologies. Unity®, 2018.

[11] Jesper Kjeldskov, Mikael B. Skov, and Jan Stage. Instant data analysis: Conducting usability evaluations in a day. In *Proceedings of the Third Nordic Conference on Human-computer Interaction*, NordiCHI '04, pages 233–240, New York, NY, USA, 2004. ACM.

[12] Jeremy Stangroom. Social science statistics, 2018.

A. Evaluation Tasks

A.1. Task 1

The shown zone has been calculated from the model shown on figure 7. A number of transitions has been taken and the active state is L1. Is it possible to take the b! transition to reach L2 with the given zone?

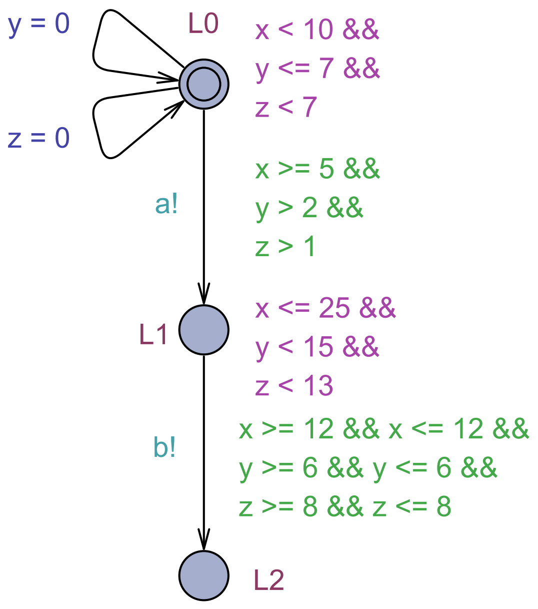
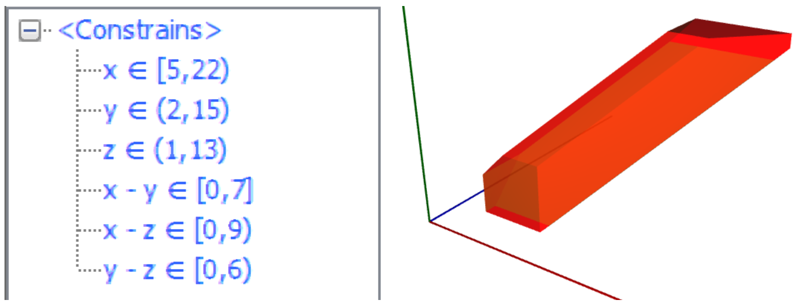


Figure 7: Automaton for task 1

Zone presentation

The test subjects were presented with one of the two representations of the zone shown in figure 8.



<Constrains>
- $x \in [5,22)$
- $y \in (2,15)$
- $z \in (1,13)$
- $x - y \in [0,7]$
- $x - z \in [0,9)$
- $y - z \in [0,6)$

Figure 8: Zone for task 1

Answer:

The transition can not be taken. The point is outside the zone.

A.2. Task 2

The shown zone has been calculated from the model shown on figure 9. A number of transitions has been taken and the active state is L1. Is it possible to take the b! transition to reach L2 with the given zone?

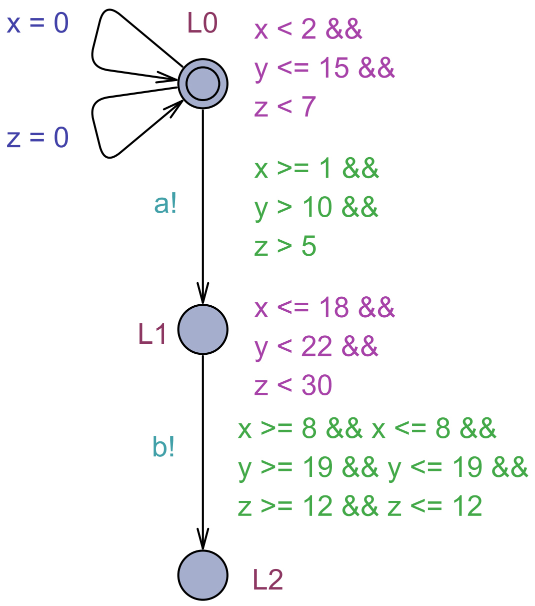
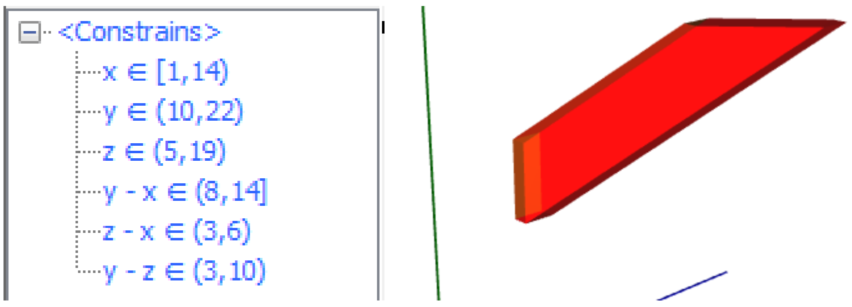


Figure 9: Automaton for task 2

Zone presentation

The test subjects were presented with one of the two representations of the zone shown in figure 10.



```
x ∈ [1,14)
y ∈ (10,22)
z ∈ (5,19)
y - x ∈ (8,14]
z - x ∈ (3,6)
y - z ∈ (3,10)
```

Figure 10: Zone for task 2

Answer:

The transition can be taken. The point is well inside the zone.

A.3. Task 3

The shown zone has been calculated from the model shown on figure 11. A number of transitions has been taken and the active state is L1. Is it possible to take the b! transition to reach L2 with the given zone?

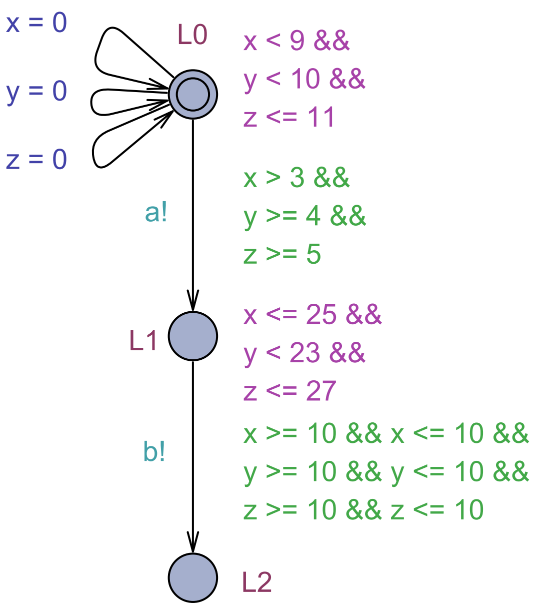
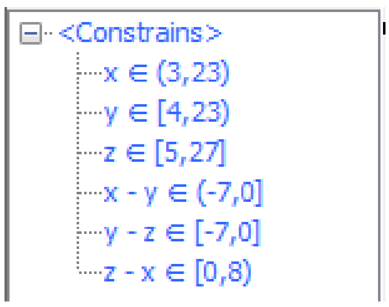


Figure 11: Automaton for task 3

Zone presentation

The test subjects were presented with one of the two representations of the zone shown in figure 12.



<Constrains>
- x ∈ (3,23)
- y ∈ [4,23)
- z ∈ [5,27]
- x - y ∈ (-7,0]
- y - z ∈ [-7,0]
- z - x ∈ [0,8)

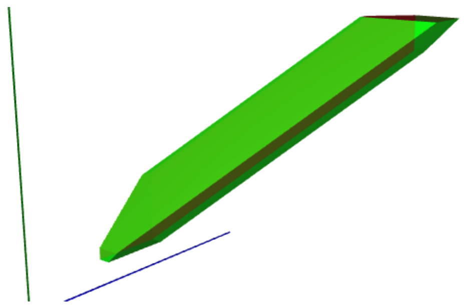


Figure 12: Zone for task 3

Answer:

The transition can be taken. The point is on a non strict constraint.

A.4. Task 4

The shown zone has been calculated from the model shown on figure 13. A number of transitions has been taken and the active state is L1. Is it possible to take the b! transition to reach L2 with the given zone?

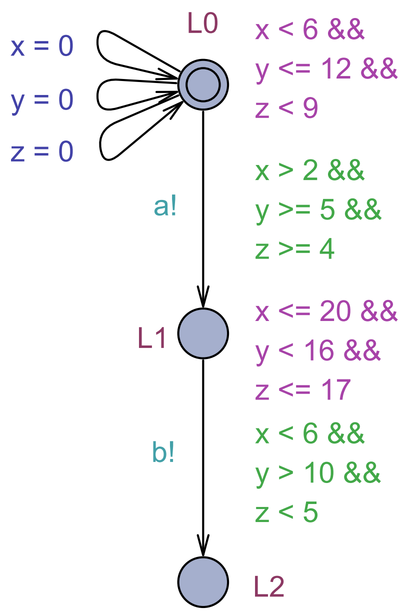


Figure 13: Automaton for task 4

Zone presentation

The test subjects were presented with one of the two representations of the zone shown in figure 14.

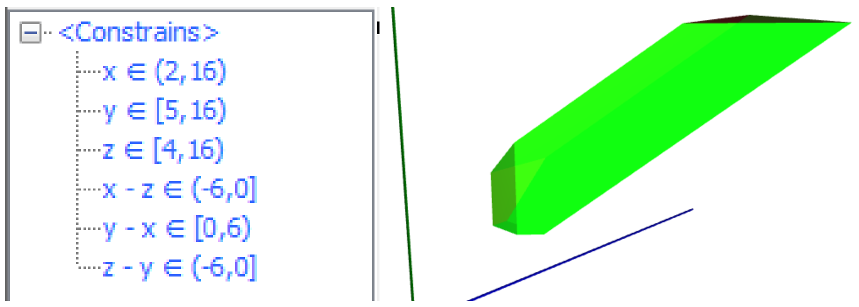

<Constrains>
- $x \in (2,16)$
- $y \in [5,16)$
- $z \in [4,16)$
- $x - z \in (-6,0]$
- $y - x \in [0,6)$
- $z - y \in (-6,0]$

Figure 14: Zone for task 4

Answer:

The transition can be taken. The zones overlap.

A.5. Task 5

The shown zone has been calculated from the model shown on figure 15. A number of transitions has been taken and the active state is L1. Is it possible to take the b! transition to reach L2 with the given zone?

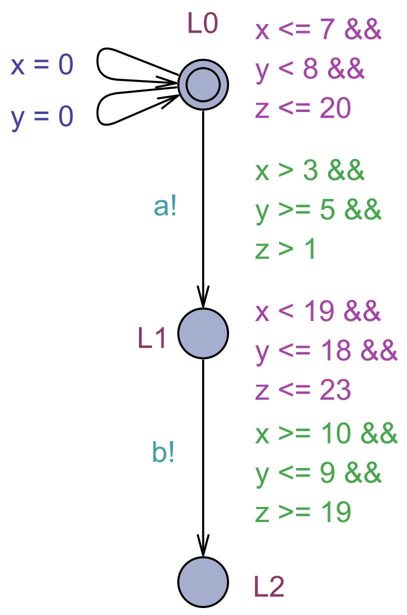
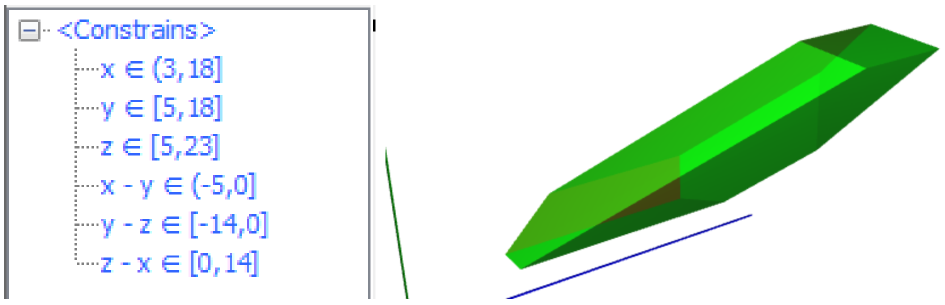


Figure 15: Automaton for task 5

Zone presentation

The test subjects were presented with one of the two representations of the zone shown in figure 16.



<Constrains>
- x ∈ (3,18]
- y ∈ [5,18]
- z ∈ [5,23]
- x - y ∈ (-5,0]
- y - z ∈ [-14,0]
- z - x ∈ [0,14]

Figure 16: Zone for task 5

Answer:

The transition can not be taken. The zones don't overlap.

A.6. Task 6

The shown zone has been calculated from the model shown on figure 17. A number of transitions has been taken and the active state is L1. Is it possible to take the b! transition to reach L2 with the given zone?

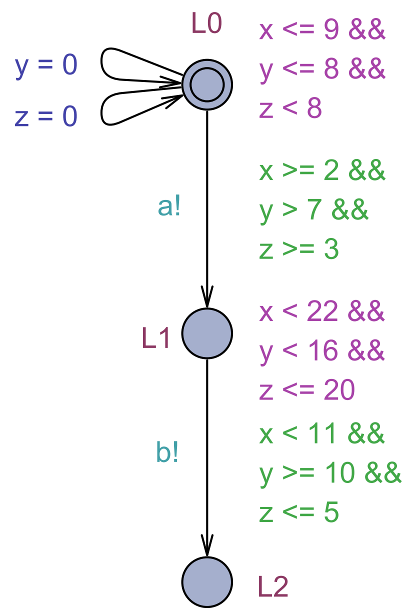
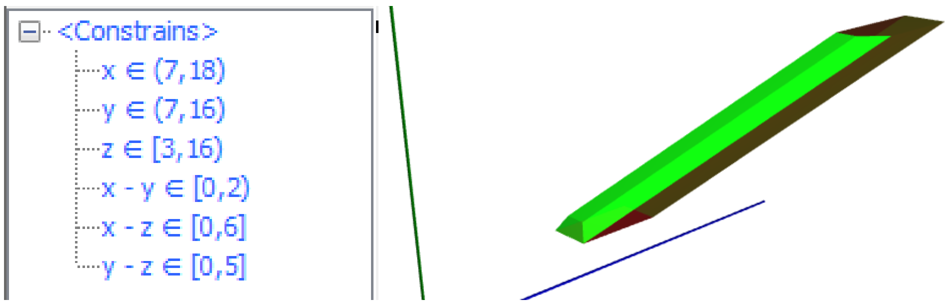


Figure 17: Automaton for task 6

Zone presentation

The test subjects were presented with one of the two representations of the zone shown in figure 18.



<Constrains>

- x ∈ (7,18)
- y ∈ (7,16)
- z ∈ [3,16)
- x - y ∈ [0,2)
- x - z ∈ [0,6]
- y - z ∈ [0,5]

Figure 18: Zone for task 6

Answer:

The transition can be taken. The zones touch and the constraints at that point are not strict.

A.7. Task 7

The shown zone has been calculated from the model shown on figure 19. A number of transitions has been taken and the active state is (L1, L1). Is it possible to handshake on b and transition to reach (L2, L2) with the given zone?

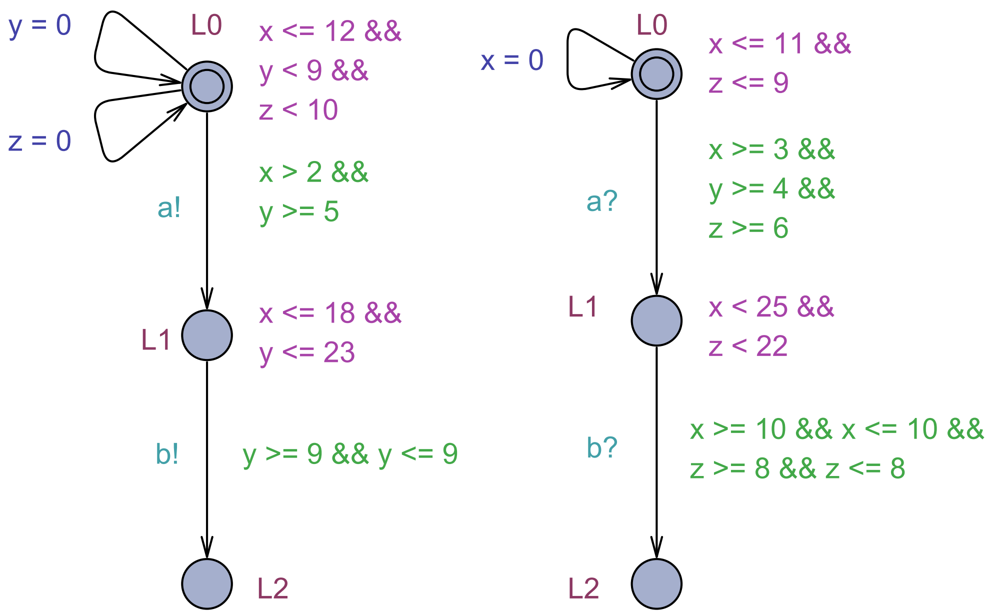
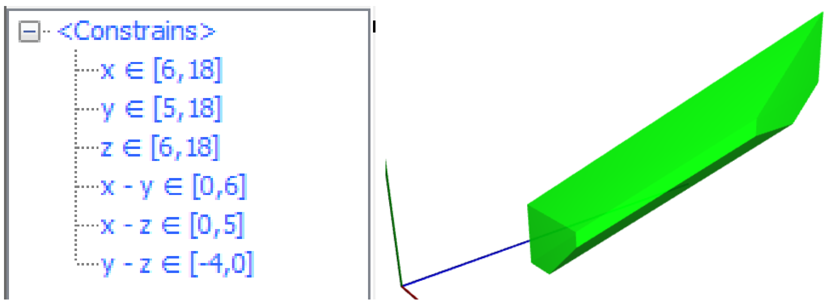


Figure 19: Automata for task 7

Zone presentation

The test subjects were presented with one of the two representations of the zone shown in figure 20.



<Constrains>
- x ∈ [6,18]
- y ∈ [5,18]
- z ∈ [6,18]
- x - y ∈ [0,6]
- x - z ∈ [0,5]
- y - z ∈ [-4,0]

Figure 20: Zone for task 7

Answer:

The transitions can not be taken. The point is outside the zone.

A.8. Task 8

The shown zone has been calculated from the model shown on figure 21. A number of transitions has been taken and the active state is (L1, L1). Is it possible to handshake on b and transition to reach (L2, L2) with the given zone?

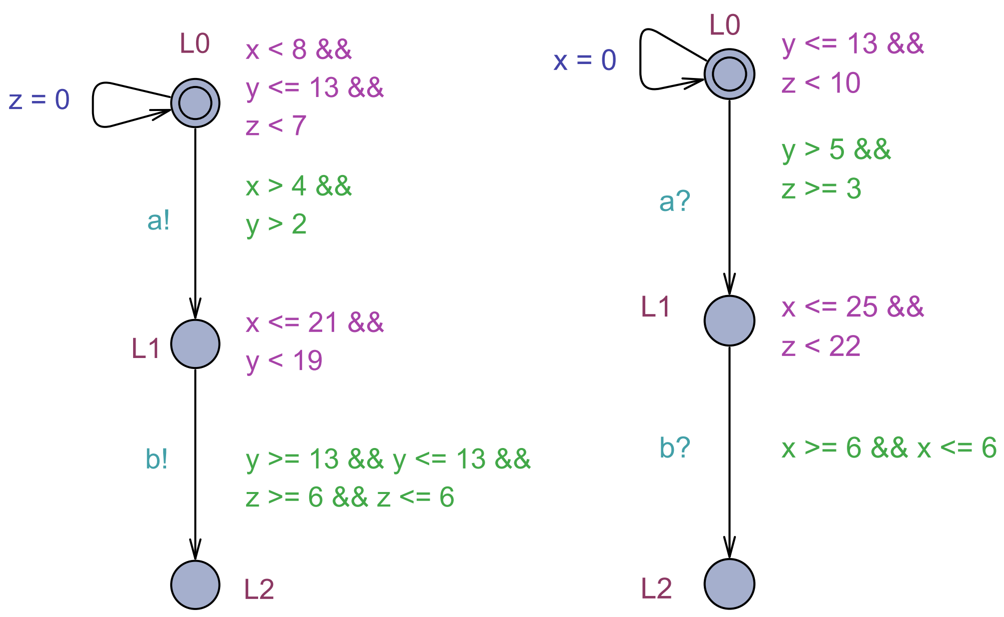
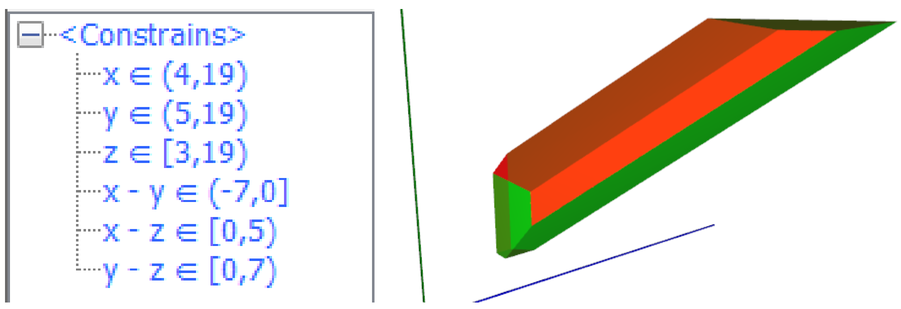


Figure 21: Automata for task 8

Zone presentation

The test subjects were presented with one of the two representations of the zone shown in figure 22.



```
<Constrains>
  x ∈ (4,19)
  y ∈ (5,19)
  z ∈ [3,19)
  x - y ∈ (-7,0]
  x - z ∈ [0,5)
  y - z ∈ [0,7)
```

Figure 22: Zone for task 8

Answer:

The transitions can not be taken. The point is on a strict constraint.

A.9. Task 9

The shown zone has been calculated from the model shown on figure 23. A number of transitions has been taken and the active state is (L1, L1). Is it possible to handshake on b and transition to reach (L2, L2) with the given zone?

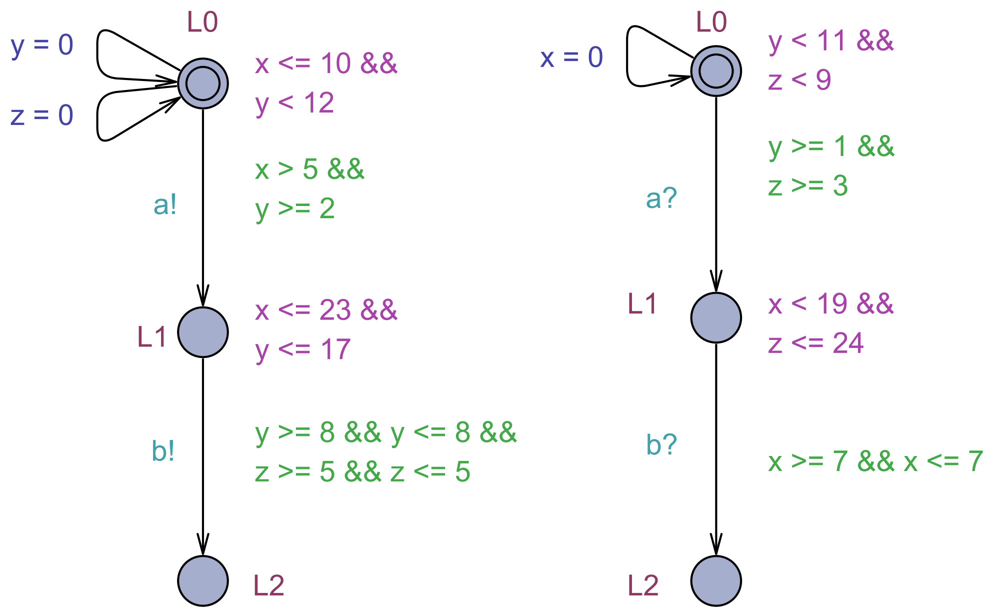
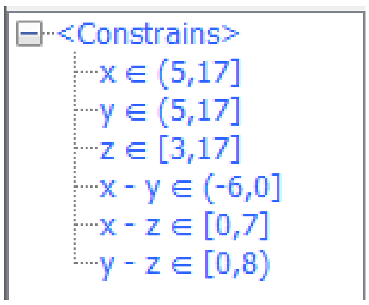


Figure 23: Automata for task 9

Zone presentation

The test subjects were presented with one of the two representations of the zone shown in figure 24.



```
<Constrains>
  x ∈ (5,17]
  y ∈ (5,17]
  z ∈ [3,17]
  x - y ∈ (-6,0]
  x - z ∈ [0,7]
  y - z ∈ [0,8)
```

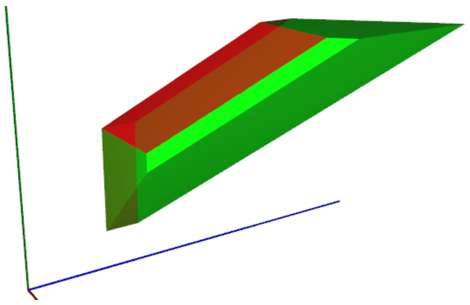


Figure 24: Zone for task 9

Answer:

The transitions can be taken. The point is inside the zone.

A.10. Task 10

The shown zone has been calculated from the model shown on figure 25. A number of transitions has been taken and the active state is (L1, L1). Is it possible to handshake on b and transition to reach (L2, L2) with the given zone?

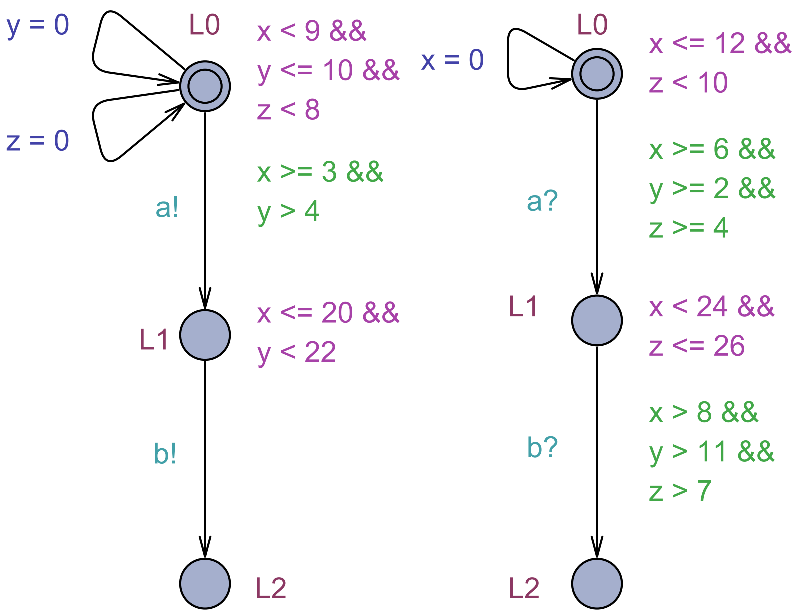
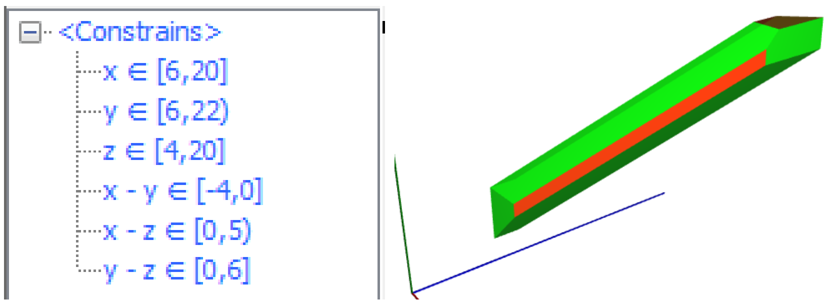


Figure 25: Automata for task 10

Zone presentation

The test subjects were presented with one of the two representations of the zone shown in figure 26.



<Constrains>
- x ∈ [6,20]
- y ∈ [6,22)
- z ∈ [4,20]
- x - y ∈ [-4,0]
- x - z ∈ [0,5)
- y - z ∈ [0,6]

Figure 26: Zone for task 10

Answer:

The transitions can be taken. The zones overlap.

A.11. Task 11

The shown zone has been calculated from the model shown on figure 27. A number of transitions has been taken and the active state is (L1, L1). Is it possible to handshake on b and transition to reach (L2, L2) with the given zone?

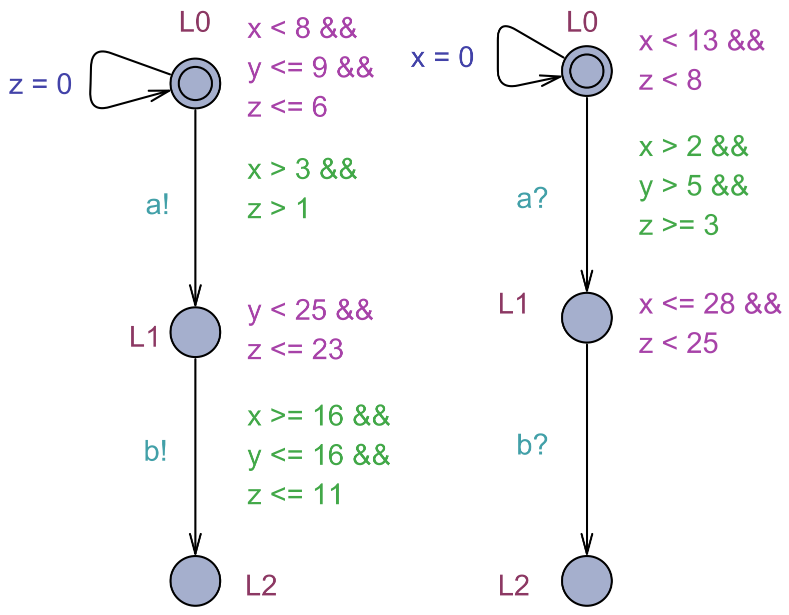
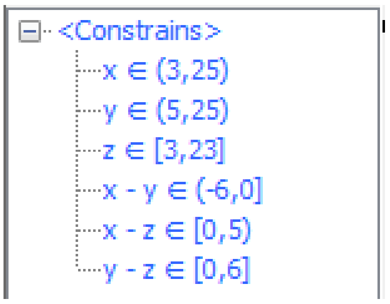


Figure 27: Automata for task 11

Zone presentation

The test subjects were presented with one of the two representations of the zone shown in figure 28.



<Constrains>
- x ∈ (3,25)
- y ∈ (5,25)
- z ∈ [3,23]
- x - y ∈ (-6,0]
- x - z ∈ [0,5)
- y - z ∈ [0,6]

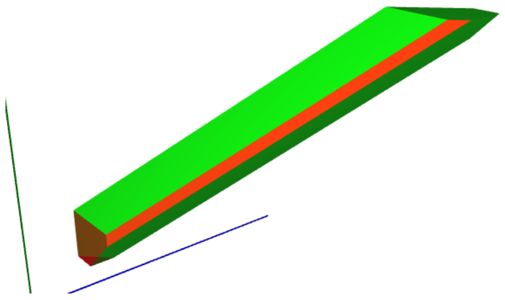


Figure 28: Zone for task 11

Answer:

The transition can not be taken. The zones touch each other, but the constraints at that point are strict.

A.12. Task 12

The shown zone has been calculated from the model shown on figure 29. A number of transitions has been taken and the active state is (L1, L1). Is it possible to handshake on b and transition to reach (L2, L2) with the given zone?

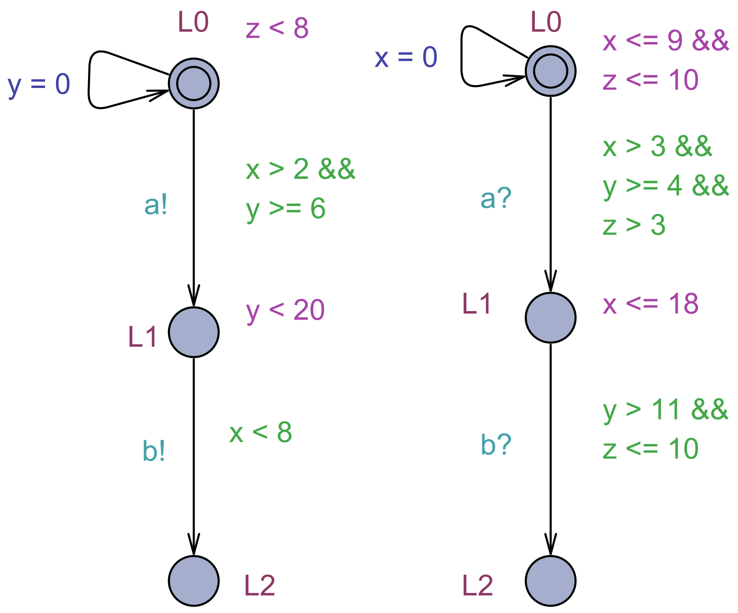


Figure 29: Automata for task 12

Zone presentation

The test subjects were presented with one of the two representations of the zone shown in figure 30.

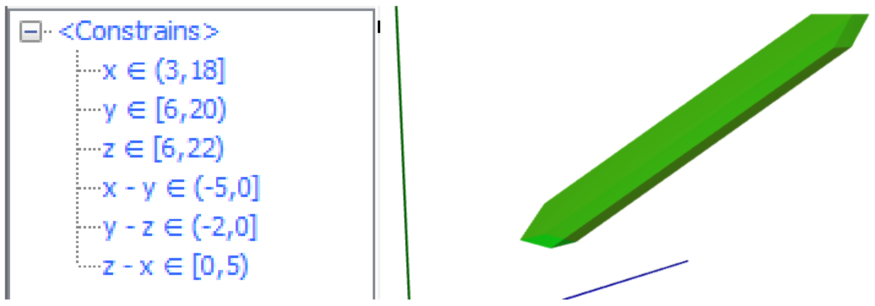


Figure 30: Zone for task 12

Answer:

The transition can not be taken. The zones don't overlap.

B. Evaluation Results: Time and Correctness

Table 2 shows how each test subject did on each task and how long it took. The grey cells show results solving the tasks using the visual representation of the zone, and the white cells show results solving the tasks using a textual representation of the zone.

Test Subject	1	2	3	4	5	6
Task 1	Incorrect 1:13.56	Incorrect 4:15.91	Correct 0:42.62	Incorrect 2:08.81	Correct 1:12.22	Incorrect 2:51.54
Task 2	Correct 0:55.63	Correct 1:27.12	Correct 1:08.63	Correct 1:02.03	Correct 1:56.85	Correct 1:28.13
Task 3	Correct 1:26.82	Incorrect 1:08.62	Correct 1:08.78	Correct 0:24.88	Correct 0:39.17	Correct 1:25.73
Task 4	Correct 1:30.91	Did not finish	Correct 1:10.93	Incorrect 1:03.82	Incorrect 2:04.05	Correct 1:56.31
Task 5	Correct 0:44.68	Incorrect 4:30.03	Correct 1:53.65	Correct 1:12.90	Correct 1:19.98	Incorrect 3:27.29
Task 6	Correct 0:34.10	Correct 1:30.91	Correct 2:41.38	Correct 0:37.31	Correct 1:13.30	Correct 1:10.44
Task 7	Correct 1:39.91	Correct 1:59.15	Correct 0:33.31	Correct 0:57.82	Correct 0:58.79	Correct 1:50.13
Task 8	Incorrect 0:57.16	Incorrect 0:44.53	Correct 0:38.00	Correct 0:29.10	Correct 0:50.28	Correct 0:38.87
Task 9	Correct 0:14.09	Correct 1:12.13	Correct 0:23.10	Correct 0:28.34	Correct 0:24.03	Correct 1:14.58
Task 10	Correct 0:29.28	Correct 0:39.57	Correct 0:52.00	Correct 0:21.94	Correct 0:54.41	Correct 0:28.98
Task 11	Correct 0:33.69	Correct 3:30.22	Correct 1:36.21	Correct 0:47.09	Incorrect 1:17.31	Correct 2:18.61
Task 12	Incorrect 0:17.53	Correct 2:00.35	Correct 1:08.75	Correct 0:42.63	Correct 0:44.36	Correct 0:32.05

Table 2: Time and Correctness of Task Solving

C. Brainstorm Notes

A test subject failed to notice that a point he added was actually behind the zone in the visual tool. Therefore he thought it was in the zone and that the transition therefore was possible.

Almost all users would like to be able to insert their own zone, so they can see if and how it overlaps with the feasibility zone. Many users when having less restrictive guards, in other words a second zone, ended up drawing points to outline the zone. It would be easier and give more information to be able to just draw the zone from constraints. This also fits well with how the simulator shows it as it shows the zone (with text) as it would look after taking a transition. Showing the zone before and the zone created by the guards and how they overlap can give a better understanding of what happens between the two steps.

When handling less restrictive guards with textual representation everyone resorted to trying a single point and seeing if it was within the zone by calculating the constraints as equations one by one. This is not certain to give the correct answer in all cases.

Something important lacking was numbers on the axes to get an idea of the scale of things and give more information at a glance. This could also include a grid to make it easier to quickly see how far points are from each other. In the same vein showing the coordinates of vertices could also be helpful, or at least of some vertices like the minimum and maximum vertices.

All subjects found it was easier to find out whether a point was in the zone using the visual tool. Especially the constraints with two clocks were troublesome and slow to calculate and often caused errors, whereas it made no difference when just inserting a point in the tool. Some would like if it was indicated whether the inserted point was in the zone or not.

Some users made mistakes with both methods because of a mistaken understanding that delays could only be whole numbers, thus believing x ¿ 10 was the same as x ¿= 11.

Users had no problems understanding that a point was in the zone if it was in the middle of a green face, or on an edge with all green faces. There was some confusion as to whether a point was actually in the face or just close to it. There was a usability problem with color blindness because the used colors were red and green.

Most test subjects were more confident in their answers when using the visual tool. There was no arithmetics that could have gone wrong and they were sure all the constraints with two clocks were satisfied at the same time (something hard to convince oneself of by looking only at the constraints).

Some test subjects "went the wrong way" in a dimension when checking for a less restrictive zone in the visual tool. For example they started adding points further into the space despite having a z ¡ k constraint.

The visual tool allowed for a more explorative approach were the test subjects tried adding a point, and then adding more points with new values in one of the dimensions simply to see what happened. One remarked this fit well with how they used the simulator as that was also simply seeing what happened with one transition and with the other, to understand the model. This approach was also easier to get started with as the text required math to even get started which is hard if you don't know which equations you start with.

Names are lacking on axes. That means it takes a little longer to familiarize yourself with the tool.

When text representation forced calculation with constraints with two clocks there was a lot to keep in your head, with values, which to substract from which (one did mix them up), negative bounding values and whether a constraint is strict or not. One person used paper for calculations.

A suggestion to show the coordinates of the point you are hovering over on the zone.

Once a point has been added that isn't in the zone it is quite easy to see what needs to be done to find one that is, for example trying a lower value for y if the point is above. That is hard to realise with text, and trying a new valuation is also quite difficult as all the equations pretty much needs to be recalculated.

It was sometimes hard to see a point inside the zone because of the reflections on the zone from the 3D lighting.

Suggestion that points and other zones could also come from simulation tool.