POLITECNICO DI MILANO
MILANO 1863

Formal Methods For Concurrent and Real-Time Systems

FORMAL ANALYSIS OF SEARCH-AND-RESCUE SCENARIOS

Homework Project

Authors
Giordanelli Gabriele, Saccone Alessandro, Tognoli Tommaso

Instructors
Prof. Pierluigi San Pietro
Dr. Livia Lestingi
A.Y. 2023-2024

Abstract

Rescue services increasingly exploit autonomous systems as a support tool during mass emergencies. This project adopts a framework that is currently under research, envisaging civilian survivors helping those in need or asking the first-responders to intervene. In this scenario drones are used to scan the area and detect attention-worthy situations to maximise the percentage of saved survivors.

1 High-Level Model Description

The considered model includes 3 types of actors:

- **First Responder:** they are trained personnel that can intervene to help survivors.
- **Civilian:** they are the survivors and their goal is to get to safety. They can also help other survivors by either becoming a zero responder or by contacting a first responder.
- **Drone:** they fly over the area looking for endangered civilians with the intent of getting them to safety.

2 Entities Description

To model a scenario 4 types of automaton are available: one for each entity and one for synchronizing the whole system.

2.1 Sync

It is the automaton responsible of for the initialization of some of the global variables to their default values (usually -1) that will then be used for communication between entities. Also, after this initial phase it is responsible for the coordination of all the entities. The goal is to make sure that all entities perform the same amount of moves, so no entity can stay still for potentially unlimited periods of time.

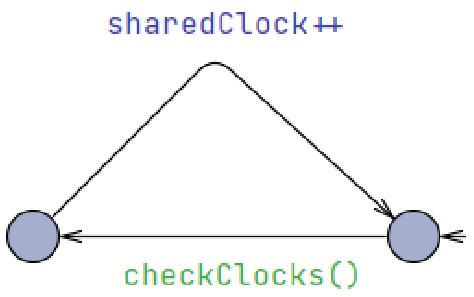


Figure 1: Clock Generator

The mechanism works like a move counter that makes sure that all entities perform one move for each time unit (time is represented by the "shared-Clock" global variable), without anyone starving. A possible alternative would have been using a global clock and state invariant to force movement, but ours seemed like an easier to implement solution for our purpose.

2.2 Civilian

The civilian TA represents the survivors and has the following behavior:

- **moving:** at the beginning the civilian moves according to the decided moving policy. After each move (so after his position is updated) he checks whether he is now close to a fire or a exit, and in that case, moves to the "danger" or "safe" state respectively. Otherwise, he will simply go back to the "moving" state. We assumed that no survivor is initially placed close to a fire, meaning that at the beginning of a simulation there will be no civilian in the "danger" state;
- **danger:** when a civilian moves to a cell adjacent to a fire (diagonally-adjacent counts too) he is considered in danger and it stops moving while he waits for help. From this state he can either become a casualty (if help does not arrive before "Tv" time units) or get helped by a FR or a ZR and transition to the "beingHelpedFR" or "beingHelpedZR" state respectively.
- **beingHelpedFR and beingHelpedZR:** when a FR or a ZR starts helping, the civilian will wait in these two states. After this, two possible things can happen:
  - the civilian still becomes a casualty: we assumed that the civilian will remain endangered until the helper is done, so the "Tv" count-down continues even while a FR or ZR is trying to save him. For this reason, if the Tv count-down expires before the civilian is saved, he will still become a casualty;
  - the civilian is saved: if the helper got to the endangered civilian early enough, he will manage to save him and the civilian will move to the "safe" state successfully;
- **safe:** this is a final state where saved civilians end up. Civilians here exit the scenario and are no longer on the grid. When moving to the "safe" state all civilians increment a global variable called "count_safe";
- **casualty:** this is the second possible final state for a civilian. Civilian will get here if after they get to the "danger" state they are not made safe within "Tv" time units. When getting to this state, the civilian leaves the grid and exits the scenario;

- **zeroResponder:** a civilian who is not in danger can be contacted by a drone to be asked to help an endangered civilian directly. The contacted civilian will become a ZR and this is the first state where he will go. Here, the ZR will stop moving on the grid and a timer called clockZR will start. The ZR will then contact the endangered civilian through the "zeroResponderHelps" channel and then wait for the needed time to pass. If the civilian becomes a casualty while the ZR responder is helping, the ZR responder will go back to the "moving" state thanks to the "tooLate" channel, otherwise, when the clockZR gets to the target value (overheadZR + Tzr) both the endangered civilian and the ZR will successfully move to the "safe" state;

- **firstResponder:** this is the state where a free (not endangered) civilian will move when instructed to contact a FR by the drone. Similarly to what was happening in the "zeroResponder" case, the needed time periods will be checked (thanks to the clockFR variable) before the needed FR is actually contacted. After this moment, the control of the saving procedure is passed to the FR himself. At this point, the civilian will wait in the "contactedFR" state for the end of the procedure. When the FR is done, the civilian will receive a signal on the "firstResponderDone" channel and move to the "safe" state. We assumed this is going to happen even in the case in which the helped civilian becomes a casualty while being helped.

2.3 First Responder

First responder (FR) are in charge of assisting as many endangered civilians as possible. They can do so by being contacted by a survivor or by simply being in the 1-cell range of a survivor currently in danger. The states of the respective TA are as follows:

- **idle**: this is the state from which the FR decides the action to perform next. The possibilities are:
  - helping a near endangered civilian
  - if contacted by a survivor, assist an endangered civilian located further than the 1-cell range
  - move non-deterministically to one of the 8 adjacent cells

- **moving**: if no civilian can be helped, FRs move according to the chosen policy. In our case we opted for a purely random movement. After a non-deterministic move, the automaton goes back to the idle location. This is the action with the lowest possible precedence as FRs will always prioritize helping.

- **helpingNear**: when a FR is located within a 1-cell range of an endangered civilian, it gives assistance directly for Tfr time units. By doing so, it will be known via the variable updates that the specific FR is busy (i.e. he cannot be called for other tasks) and that the specific civilian is already being assisted and does not need help. The FR has two ways of going back to the idle state:
  - the civilian was saved in time and once Tfr time units have passed it returns to idle and goes back to being available;
  - the civilian becomes a casualty as the rescue did not come in time. If this happens, it returns to idle before Tfr time units have passed;
- **calledForHelp**:

2.4 Drone

The drone is in charge of scanning the scenario looking for endangered civilians and possible helpers. The whole process will be performed in one single action, so the drone could potentially help one civilian per time unit (for each "sharedClock" increment). The possible states of the respective TA are:

- **moving:** The drone will start in this state at the beginning of all actions. The chosen movement policy is as follows: the drone will move along the perimeter of a square with a given diameter. Each time a drone is at the corner of the square it will take one time unit more in order to change the direction (the total cost moving on the corner is of two time units);
- **step1, step2, step3:** the first part of all actions for the drone will consist in trying to go through the state sequence <"step1", "step2", "step3">while checking if the situation in the detected area requires attention. This will be done by looking at the cells surrounding the drone according to the N_v variable given as a parameter, that indicates the action radius of the drone itself expressed as Manhattan distance. The drone will check for:
  - an endangered civilian;
  - a free civilian that can help the endangered one either by becoming a ZR or by contacting a FR;

  - a FR within a radius of ”tresh” cells. The ”tresh” parameter indicates the maximum distance between the drone and the FR, for the FR to be contacted instead of relying on the ZR’s help. This parameter is used to decide whether the helper (the contacted civilian) will help directly or simply contact a FR.

  If we actually get to the ”step3” location, it means that the drone detected a situation where it could potentially help;

- **contacting:** from ”step3”, if the drone also detected an available FR within its radius, the automaton will move to the ”contacting” location indicating that the helping civilian only needs to contact the FR. From this location we have two possible developments:

  - the detected helping civilian is actually available: the drone will instruct him and then it will notify the selected FR to avoid him being selected for other tasks;
  - the selected civilian is already busy helping someone: the drone’s helping procedure will simply end and nobody will be contacted;

  In both cases, the drone will then go back to the ”moving” state, ready to repeat the whole process in the following clock cycle.

- **helping:** if we get to ”step3” without having detected any FR, the drone will move to the ”helping” location and behave in a similar way to what it does in the ”contacting” location. If the helping civilian is not already busy he will be contacted and told to become a ZR, otherwise the helping procedure will end. Again, the drone will always go back to the ”moving” location at the end of the action, no matter what.

In all entities we used ”Committed” locations to make complex actions atomic and make sure that no action would be left half finished.

3 Design Choices

3.1 Environment Design

Our scenario consists of a grid which we modelled using matrices in our global variables. The layout of the fires and of the available exits are managed using two boolean matrices. These two matrices are never modified during a simulation. We then have two other matrices to keep track of which cells are already taken by either a civilian or a first responder. The default value for these matrices is -1, indicating an empty cell. These matrices will be updated by the entities as they move around during a simulation by replacing -1 with their "id" as they move into a cell (they do the opposite when leaving it). The size of the grid can be customized thanks to the "n" and "m" parameters and the position of fires and exits can be changed by flipping bits in the respective matrices.

4 Studied Deterministic Scenarios

To check properties on our model, we wrote queries using the "sharedClock" variable as time constraint and the global variable "count_safe" to evaluate the portion of saved survivors. We could not express the constraint on the saved civilians percentage in other ways as the Verifier does not support doubles.

4.1 Scenario 1

For the first scenario we decided for a 10x10 grid with the following initial setup:

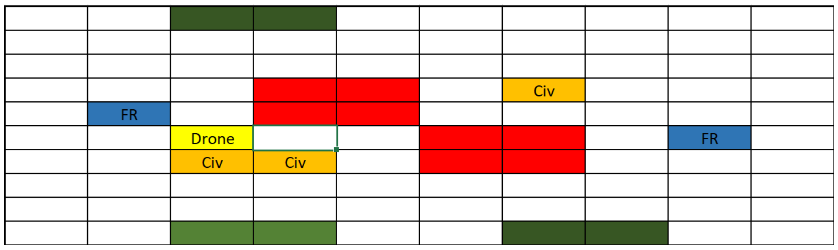
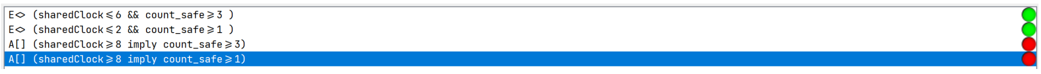


Figure 2: First Setup

The moving policy for the civilians and the first responders is random (they can move in any of their free surrounding cells) while the drone moves clockwise following a square pattern. The selected queries and their results are reported below:



```
E<> (sharedClock ≤ 6 && count_safe ≥ 3 )
E<> (sharedClock ≤ 2 && count_safe ≥ 1 )
A[] (sharedClock ≥ 8 imply count_safe ≥ 3)
A[] (sharedClock ≥ 8 imply count_safe ≥ 1)
```

Figure 3: First Scenario's Queries

- Both queries indicating the possibility of a certain number of civilian to be saved (in this case 1 civilian means 33% and 3 civilians means 100%) are satisfied as it is in fact possible for all of them to get to an exit within the sharedClock constraint.
- Both queries verifying whether it is possible for a certain number of civilians to always be saved are not satisfied. This is because the moving policy is random, so the verifier can for sure find a worst case scenario where all civilians make the wrong decisions and all become casualties, denying the wanted property. This is true no matter how many civilians we want to ensure to be saved.

4.2 Scenario 2

For the second scenario we used a bigger grid (25x15) with more entities. We focused the entities around the fires in order to increase the probability for interactions among them. Also, to make it easier for the civilians to be saved, we increased the radius of the drone route (light purple area), to make it control a larger area and we also increased its threshold to make it contact a FR more often. This could be helpful because we parameterized the FR so it is way more efficient compared to a ZR when helping a civilian (it takes them less time to bring the civilian to safety).

The grid is structured as follows:

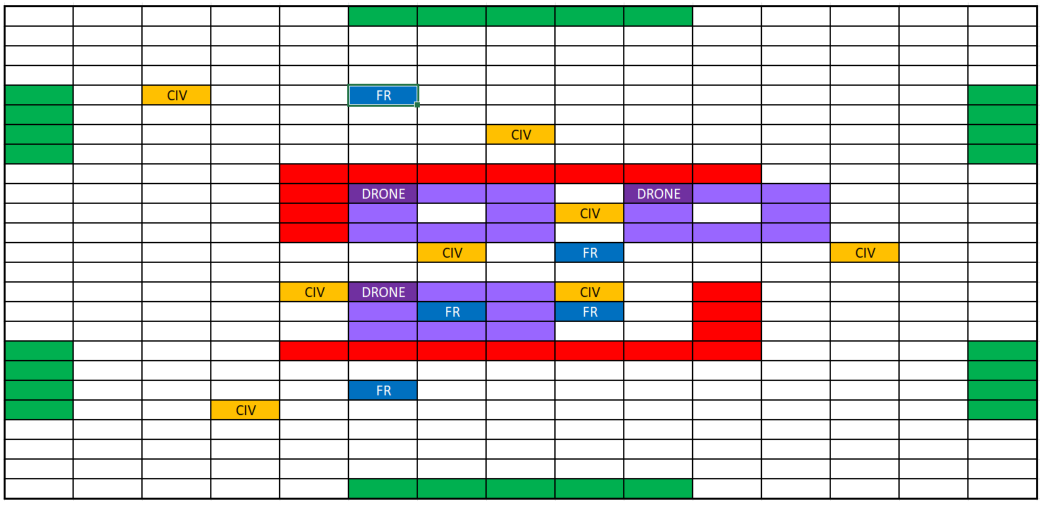
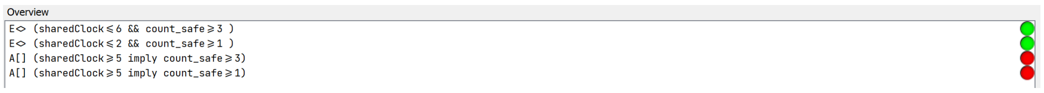


Figure 4: Grid for the Second and Third Scenario

Although we parameterized this scenario to make it easier for the civilians to get to safety and we even included some civilians that are way closer to an exit than they are to a fire, the queries produced results similar to the ones obtained in the first scenario:



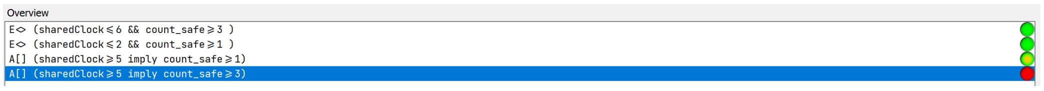
```
E<> (sharedClock ≤ 6 && count_safe ≥ 3 )
E<> (sharedClock ≤ 2 && count_safe ≥ 1 )
A[] (sharedClock ≥ 5 imply count_safe ≥ 3)
A[] (sharedClock ≥ 5 imply count_safe ≥ 1)
```

Figure 5: Second Scenario's Queries

In particular, the queries about guaranteeing the safety of a certain number of civilians failed again. This happened because no matter how easy we make it for a civilian to get to safety, if there is a possibility for him to become a casualty, the wanted property will not be formally verified.

4.3 Scenario 3

For the third scenario we used the same grid as scenario 2 (see Figure 4) but changed the civilian moving policy. In order to be able to guarantee that some civilians will be saved, we made them smarter and made them move towards the border of the grid when they detect an exit within their view range. This avoids civilians to go back to the middle of grid (so further from safety and potentially towards a fire) when they are already close to an exit. The results for the queries we ran on this model are the following:



```
E<> (sharedClock ≤ 6 && count_safe ≥ 3 )
E<> (sharedClock ≤ 2 && count_safe ≥ 1 )
A[] (sharedClock ≥ 5 imply count_safe ≥ 1)
A[] (sharedClock ≥ 5 imply count_safe ≥ 3)
```

Figure 6: Third Scenario's Queries

In this case, we manage to obtain a positive result for one of the queries on always ensuring that a certain number of civilians get to safety. In particular, it may be true that one out of seven civilians (around 14%) will always get to safety. This is not 100% sure as the verifier cannot check all possible development of the scenario to prove correctness. On the other hand, when we increased the number of civilians we wanted to save (last query, askin for around 40% of civilians to get to safety) we got a negative result, meaning that our changes were for sure not enough to achieve that level of success in our rescuing mission.

5 Conclusions

The analyzed scenarios show that when entities behave randomly, it is impossible to ensure any degree of success of the rescuing operation, no matter the number of responders and drones we deploy. On the other hand, entities can be made smarter and adapted to specific grid settings in order to ensure better performances. Because the model can be customized, it can be used to emulate real world scenarios and study them in order to evaluate what possible changes to the entities can result in the best overall result in particular conditions.