Internship Report

Solal Rapaport

June 2022

1 Introduction

There are two goals here, the first one is to build formulas that will allow robots, spread on a ring, to gather. We have k robots and we will use view vectors to build those formulas. The formulas will be an interpretation of the pseudo-code given in the research report [1].

The formulas we are building, will be used with formulas given in an other research report [2], and then will be tested in the acceleration algorithm using an interpolant [2]. Which leads us to the second goal, we want to implement and, if possible, improve this algorithm.

2 Logical Formulas

In this section, we will translate the algorithms given in the research report [1]. Some changes will have to be made because we can't literally translate an algorithm into a first-order logic formula.

Before each formula we will describe briefly their scope : when will they be true (or false). We won't present to you the implementation of those formulas in this report. There will be an annex available with the *Python* implementation that we use in order to test those formulas and to put them in the algorithm [2].

We have three strategies. Each of them allows a robot to move in a given direction based on its environment. They all have the same definition, they take one argument : the view vector (distance vector).

2.1 Configurations with single multiplicity

The strategy ϕ_{SM} is *true* if the given configuration has a single multiplicity and that the robot calling the strategy should move toward the robot at distance d_0 :

$$\begin{gathered}\phi_{SM}(d_0,\dots,d_{k-1}) := \\ (\bigvee_{i=0}^{k-1}(d_i = 0 \bigwedge_{j=0\ \ j\neq i}^{k-1}(d_j > 0 \vee (d_j = 0 \wedge d_{j-1} = 0))))\wedge \\ (d_{k-1} \neq 0)\wedge \\ ((d_1 = 0 \wedge d_{k-2} = 0 \wedge d_0 \leq d_{k-1}) \vee (d_1 = 0 \wedge d_{k-2} \neq 0))\end{gathered}$$

In order to test our strategy we need a function that will initialize our first configuration and make it one with a single multiplicity without being already a winning one. Here is the formula $InitSM$ which is *true* if p, s and t form a configuration with a single multiplicity, the configuration is not a winning one, all p are initialized in the right scope, all t are initialized at 0 and all s are initialized at RLC (-1) :

$$\begin{gathered}InitSM(p_0,\dots,p_{k-1},s_0,\dots,s_{k-1},t_0,\dots,t_{k-1},size_{ring}) := \\ \bigvee_{i=0}^{k-1}(p_i \neq p_{i+1 \mod k-1})\wedge \\ (\bigwedge_{i=0}^{k-1}(p_i \geq 0 \wedge p_i < size_{ring} \wedge s_i = -1 \wedge t_i = 0))\wedge \\ (\bigvee_{i=0}^{k-1}(\bigvee_{j=0,j\neq i}^{k-1}(p_j = p_i \wedge \bigwedge_{h=0}^{k-1}(\bigwedge_{l=0,l\neq h}^{k-1}(p_h \neq p_l \vee p_h = p_i)))))\end{gathered}$$

2.2 Gathering rigid configurations

Let d_{ij} be the value j of the view vector of the robot i, and ds_{ij} the value j of the symmetrical view of the robot i. The robot is calling the strategy ϕ_R.

Here are all the logic formulas used in order to build ϕ_R:

AllView is *true* if $d_{00}, \dots, d_{k-1k-1}$ are all the views you can obtain from a single view vector $dist_0, \dots, dist_{k-1}$:

$$
\begin{gathered}
AllView(dist_0, \dots, dist_{k-1}, d_{00}, \dots, d_{k-1k-1}) := \\
(\bigwedge_{i=0}^{k-1} (\bigwedge_{j=0}^{k-1} (d_{ij} = dist_{(j+i) \mod k})))
\end{gathered}
$$

IsRigid is *true* if the given configuration is a rigid configuration. Meaning, all views are distinct, there is no multiplicity, and the configuration isn't symmetric nor periodic.

$$
\begin{gathered}
IsRigid(d_{00}, \dots, d_{k-1k-1}, ds_{00}, \dots, ds_{k-1k-1}) := \\
\bigwedge_{i=0}^{k-1} (\bigwedge_{j=0}^{k-1} d_{ij} \neq 0) \wedge \\
\bigwedge_{i=0}^{k-1} (\bigwedge_{l=0\ l \neq i}^{k-1} ((\bigvee_{j=0}^{k-1} d_{ij} \neq d_{lj}) \wedge (\bigvee_{j=0}^{k-1} d_{ij} \neq ds_{lj}) \\
\wedge (\bigvee_{j=0}^{k-1} ds_{ij} \neq d_{lj}) \wedge (\bigvee_{j=0}^{k-1} ds_{ij} \neq ds_{lj})))
\end{gathered}
$$

AllCode is *true* if (α'_r, β'_r) is the set of two natural numbers of the robot r such as α'_r and β'_r are codes of r's views, with $\alpha'_r < \beta'_r$. The process which leads us to obtain all view codes is defined in the research report [1].

$$
\begin{gathered}
AllCode(d_{00}, \dots, d_{k-1k-1}, ds_{00}, \dots, ds_{k-1k-1}, \alpha_0, \dots, \alpha_{k-1}, \beta_0, \dots, \beta_{k-1}, \\
\alpha'_0, \dots, \alpha'_{k-1}, \beta'_0, \dots, \beta'_{k-1}) := \\
\bigwedge_{i=0}^{k-1} (\alpha'_i < \beta'_i \wedge (\alpha'_i = \alpha_i \vee \alpha'_i = \beta_i) \wedge (\beta'_i = \alpha_i \vee \beta'_i = \beta_i)) \wedge \\
(\ (\alpha_0 < \alpha_1 < \dots < \alpha_{k-1} < \beta_0 < \dots < \beta_{k-1}) \wedge \\
(\bigvee_{p=0}^{k-1} (\bigwedge_{q=0}^{p-1} (d_{0q} = d_{1q}) \wedge d_{0p} > d_{1p})) \wedge \dots \wedge \\
(\bigvee_{p=0}^{k-1} (\bigwedge_{q=0}^{p-1} (ds_{(k-2)q} = ds_{(k-1)q}) \wedge ds_{(k-2)p} > ds_{(k-1)p})) \\
\vee \\
((\alpha_0 < \alpha_2 < \alpha_1 < \dots < \alpha_{k-1} < \beta_0 < \dots < \beta_{k-1}) \wedge \dots) \vee \dots\ )
\end{gathered}
$$

CodeMaker is *true* if the configuration is rigid and if $(a_0, \dots, a_{k-1}, as_0, \dots, as_{k-1})$ are each code of each view passed as a parameter :

$$
\begin{gathered}
CodeMaker(d_{00}, \dots, d_{k-1k-1}, ds_{00}, \dots, ds_{k-1k-1}, a_0, \dots, a_{k-1}) := \\
IsRigid(d_{00}, \dots, d_{k-1k-1}, ds_{00}, \dots, ds_{k-1k-1}) \wedge \\
\exists \alpha_0, \dots, \alpha_{k-1}, \beta_0, \dots, \beta_{k-1},\ \alpha'_0, \dots, \alpha'_{k-1}, \beta'_0, \dots, \beta'_{k-1}, \\
AllCode(d_{00}, \dots, d_{k-1k-1}, ds_{00}, \dots, ds_{k-1k-1}, \alpha_0, \dots, \alpha_{k-1}, \beta_0, \dots, \beta_{k-1}, \\
\alpha'_0, \dots, \alpha'_{k-1}, \beta'_0, \dots, \beta'_{k-1}) \\
(\bigwedge_{i=0}^{k-1} (\bigwedge_{j=0, j \neq i}^{k-1} ((a_i > a_j \wedge \alpha'_j > \alpha'_i) \vee (a_i < a_j \wedge \alpha'_j < \alpha'_i)))) \\
\bigwedge_{i=0}^{k-1} (\bigwedge_{j=0, j \neq i}^{k-1} a_i \neq a_j)
\end{gathered}
$$

FindMax is *true* if *Max* is the highest value of the view vector passed as a parameter :

$$
\begin{gathered}
FindMax(dist_0, \dots, dist_{k-1}, Max) := \\
(\bigwedge_{i=0}^{k-1} (Max \geq dist_i) \wedge (\bigvee_{i=0}^{k-1} (Max = dist_i)))
\end{gathered}
$$

FindM is *true* if M is the index of the robot (index in the view vector) which has the largest code of view and a neighboring robot at distance *Max* :

$$
\begin{gathered}
FindM(d_{00}, \dots, d_{k-1k-1}, a_0, \dots, a_{k-1}, Max, dM_0, \dots, dM_{k-1}) := \\
\bigvee_{m=0}^{k-1} ((\bigwedge_{i=0}^{k-1} ((a_m \geq a_i \wedge (d_{i0} = Max \vee d_{ik-1} = Max)) \\
\vee (d_{i0} < Max \wedge d_{ik-1} < Max))) \wedge M = m)
\end{gathered}
$$

FindN is *true* if N is the index of the robot (index in the view vector) with the largest code of view and M as a neighboring robot at distance *Max* :

$$\begin{gathered}
FindN(d_{00},\dots,d_{k-1k-1},a_0,\dots,a_{k-1},Max,M,N) := \\
(d_{M0} = Max \wedge d_{Mk-1} = Max \wedge \\
((N = ((M+1) \mod k) \wedge a_{(M+1) \mod k} > a_{(M-1) \mod k}) \vee \\
(N = ((M-1) \mod k) \wedge a_{(M-1) \mod k} > a_{(M+1) \mod k}))) \\
\vee \\
(d_{M0} = Max \wedge d_{Mk-1} \neq Max \wedge N = ((M+1) \mod k)) \\
\vee \\
(d_{M0} \neq Max \wedge d_{Mk-1} = Max \wedge N = ((M-1) \mod k))
\end{gathered}$$

Since those formulas can't be implemented in *Python* because it is impossible to work around a variable index, we choose to build a new formula, $FindMN$ that will be *true* if both vectors dM and dN are the view vector of, respectively, M and N.

$$\begin{gathered}
FindMN(d_{00},\dots,d_{k-1k-1},a_0,\dots,a_{k-1},Max,M,N, \\
dM_0,\dots,dM_{k-1},dN_0,\dots,dN_{k-1}) := \\
\bigvee_{m=0}^{k-1}(\quad (\bigwedge_{i=0}^{k-1}((a_m \geq a_i \wedge (d_{i0} = Max \vee d_{ik-1} = Max)) \\
\vee(d_{i0} < Max \wedge d_{ik-1} < Max))) \wedge M = m \wedge \\
(\quad (d_{m0} = Max \wedge d_{mk-1} = Max \wedge \\
((N = M+1 \mod k \wedge a_{(m+1) \mod k} > a_{(m-1) \mod k}) \vee \\
(N = M-1 \mod k \wedge a_{(m-1) \mod k} > a_{(m+1) \mod k}))) \vee \\
(d_{m0} = Max \wedge d_{mk-1} \neq Max \wedge N = M+1 \mod k) \vee \\
(d_{m0} \neq Max \wedge d_{mk-1} = Max \wedge N = M-1 \mod k) \quad) \wedge \\
((N = M-1 \mod k \wedge (\bigwedge_{l=0}^{k-1}(dN_l = d_{(m-1 \mod k)((k-1)-l)} \wedge dM_l = d_{ml}))) \vee \\
(N = M+1 \mod k \wedge (\bigwedge_{l=0}^{k-1}(dN_l = d_{(m+1 \mod k)l} \wedge dM_l = d_{m((k-1)-l)})))) \quad)
\end{gathered}$$

ϕ_R is *true* if the configuration is rigid, and if the robot is M and has a closest neighbor than N, or if the robot is N and has a closest neighbor than M.

$$\begin{gathered}
\phi_R(dist_0,\dots,dist_{k-1}) := \\
\exists d_{00},\dots,d_{k-1k-1}, \ AllView(dist_0,\dots,dist_{k-1},d_{00},\dots,d_{k-1k-1}) \wedge \\
\exists ds_{00},\dots,ds_{k-1k-1}, \bigwedge_{i=0}^{k-1}(ViewSym(d_{i0},\dots,d_{ik-1},ds_{i0},\dots,ds_{ik-1})) \wedge \\
\exists Max, a_0,\dots,a_{k-1}, dM_0,\dots,dM_{k-1}, dN_0,\dots,dN_{k-1}, \\
CodeMaker(d_{00},\dots,d_{k-1k-1},ds_{00},\dots,ds_{k-1k-1},a_0,\dots,a_{k-1}) \wedge \\
FindMax(dist_0,\dots,dist_{k-1},Max) \wedge \\
FindMN(d_{00},\dots,d_{k-1k-1},a_0,\dots,a_{k-1},Max,dM_0,\dots,dM_{k-1},dN_0,\dots,dN_{k-1}) \wedge \\
\exists dM2_0,\dots,dM2_{k-1},dN2_0,\dots,dN2_{k-1}, \\
((\bigwedge_{i=0}^{k-1}(dM2_i = dM_{i+1 \mod k})) \vee (\bigwedge_{i=0}^{k-1}(dM2_i = dM_{i-1 \mod k}))) \wedge \\
(\bigvee_{i=0}^{k-1}(dM2_i \neq dN_i)) \wedge \\
((\bigwedge_{i=0}^{k-1}(dN2_i = dN_{i+1 \mod k})) \vee (\bigwedge_{i=0}^{k-1}(dN2_i = dN_{i-1 \mod k}))) \wedge \\
(\bigvee_{i=0}^{k-1}(dN2_i \neq dM_i)) \wedge \\
\exists distM_0,\dots,distM_{k-1},distN_0,\dots,distN_{k-1}, \\
\bigwedge_{i=0}^{k-1}(distM_i = (\textstyle\sum_{l=0}^{i} dM_l) \wedge distN_i = (\sum_{l=0}^{i} dN_l)) \wedge \\
(\bigvee_{i=0}^{k-1} \quad (\quad (distM_i < distN_i \bigwedge_{q=0}^{i}(distM_q = distN_q) \bigwedge_{j=0}^{k-1}(dM_j = dist_j)) \vee \\
(distM_i > distN_i \bigwedge_{q=0}^{i}(distM_q = distN_q) \bigwedge_{j=0}^{k-1}(dN_j = dist_j)) \quad) \quad)
\end{gathered}$$

2.3 Gathering an odd number of robots

We are now building a strategy, ϕ_{ON}, that will gather an odd number of robots on a non-periodic configuration. It is the strategy with the lowest priority, meaning that the configuration won't be rigid and won't have any multiplicity.

First we build the formula, $IsPeriodic$, that will return *true* if the configuration is periodic with an odd number of robots :

$$\begin{gathered}
IsPeriodic(dist_0,\dots,dist_{k-1}) := \\
\exists p \in [1; \lfloor \tfrac{k}{3} \rfloor], (p+1) \mod 2 = 0 \wedge \\
\exists d'_0,\dots,d'_{p-1}, \bigwedge_{i=0}^{k-1}(d'_{i \mod p} = dist_i)
\end{gathered}$$

Now, we build build ϕ_{OD}, the strategy returns *true* if the configuration is non-rigid, non-periodic, has no multiplicity and has an odd number of robots. If the robot is axial then it moves in order to create a multiplicity or a rigid configuration.

$$
\begin{gathered}
\phi_{ON}(dist_0, \ldots, dist_{k-1}) := \\
\exists d_{00}, \ldots, d_{k-1k-1},\ AllView(dist_0, \ldots, dist_{k-1}, d_{00}, \ldots, d_{k-1k-1}) \wedge \\
\exists ds_{00}, \ldots, ds_{k-1k-1}, \bigwedge_{i=0}^{k-1} (ViewSym(d_{i0}, \ldots, d_{ik-1}, ds_{i0}, \ldots, ds_{ik-1})) \wedge \\
\neg IsRigid(d_{00}, \ldots, d_{k-1k-1}, ds_{00}, \ldots, ds_{k-1k-1}) \wedge \\
((k+1) \mod 2 = 0) \wedge \\
\neg IsPeriodic(dist_0, \ldots, dist_{k-1}) \wedge \\
(\bigwedge_{i=0}^{k-1} dist_i \neq 0) \wedge \\
(\bigwedge_{i=0}^{k-1} dist_i = ds_{0i})
\end{gathered}
$$

3 Algorithms

Now that we have done all of our logical formulas, we need to test those in the acceleration algorithm using an interpolant [2] and in an alternate version of that same algorithm. //TODO

4 Tests

In order to test the algorithm [2] we will use the python code we show you at the beginning : *InitSM* and *phiSM*.

We will use the SAT-solver to test different configurations. We will change the number of robots and the size of the ring from a test to an other.

4.1 Test *InitSM*

First, we test the function *InitSM* alone : can we have an initial configuration with a single multiplicity with those parameters ?

nb-robot \size-ring	2	3	4	5	6
2	Unsat	Unsat	Unsat	Unsat	Unsat
3	Sat	Sat	Sat	Sat	Sat
4	Sat	Sat	Sat	Sat	Sat
5	Sat	Sat	Sat	Sat	Sat
6	Sat	Sat	Sat	Sat	Sat

The results make sense : we can't create a multiplicity with 2 robots which is not a winning configuration. Else, even on a ring size of 2 we can have a multiplicity on one spot and only one robot on the other spot.

4.2 Test ϕ_{SM}

Now we test ϕ_{SM} through the algorithm [2], we also use the function *InitSM* that makes sure we have a single multiplicity at the beginning.

nb-robot \size-ring	2	3	4	5	6
3	Timeout	Timeout	Timeout	...	...
4	Timeout	...	...	...	...
5	...	...	...	...	...
6	...	...	...	...	...

Same test but with the function *Init* instead.

nb-robot \size-ring	2	3	4	5	6
3	Timeout	Loose	Loose	Loose	...
4		...	...	...	...
5	...	...	...	...	...
6	...	...	...	...	...

For $nb_{robot} = 3$ and $size_{ring} = 2$ we face this problem :

```
Traceback (most recent call last):
  File "algov5.py", line 56, in <module>
    Ip = tree_interpolant(And(Interpolant(And(tmpAndInterpolant)),
    And(tmpAndContext)))
  File "/usr/lib/python3.8/site-packages/z3/z3.py", line 8297,
  in tree_interpolant
    res = Z3_compute_interpolant(ctx.ref(),f.as_ast(),p.params,ptr,mptr)
  File "/usr/lib/python3.8/site-packages/z3/z3core.py", line 4074,
  in Z3_compute_interpolant
    _elems.Check(a0)
  File "/usr/lib/python3.8/site-packages/z3/z3core.py", line 1336, in Check
    raise self.Exception(self.get_error_message(ctx, err))
z3.z3types.Z3Exception: b'theory not supported by interpolation or bad proof'
```

```
foreach synchronous winning strategy f do
    k = 1;
    while true do
        I(c) = Init(c);
        continue = true;
        while continue do
            if MaybeThisSize ≠ null then
                NotThisSizeBis =[i for i in range(k) and i ∉ elem];
                if Init(c) ∧ Post(c, c1), Post(c1, c2) ∧ ··· ∧ Post(c_{k-1}, c_k) ∧
                  BouclePerdante(c_k, NotThisSizeBis)   SAT then
                    exit;                                  /* Loosing Strategy */
                end
            end
            if I(c) ∧ Post(c, c1), Post(c1, c2) ∧ ··· ∧ Post(c_{k-1}, c_k) ∧
              BouclePerdante(c_k, NotThisSize)   SAT then
                if I = Init then
                    exit;                                  /* Loosing Strategy */
                else
                    MaybeThisSize.append(k);
                    k = k + 1;
                    continue = false;
                end
            else
                I' = Interpolant(I(c) ∧ Post(c, c1), Post(c1, c2) ∧ ··· ∧
                  Post(c_{k-1}, c_k) ∧ BouclePerdante(c_k, NotThisSize));
                if I' ⟹ I then
                    if k = size_max then
                        exit;                              /* Winning Strategy */
                    else
                        NotThisSize.append(k);
                        k = k + 1;
                        continue = false;
                    end
                else
                    I = I ∨ I';
                end
            end
        end
    end
end
```

References

[1] Ralf Klasing, Euripides Markou, and Andrzej Pelc. *Gathering asynchronous oblivious mobile robots in a ring*. Tech. rep. RR-1422-07. UMR 5800 - Université Bordeaux 1, 351, cours de la Libération, 33405 Talence CEDEX, France: Laboratoire Bordelais de Recherche en Informatique, Jan. 2007.

[2] Nathalie Sznajder and Souheib Baarir. *Algorithme d'accélération par interpolants. (French) [Acceleration Algorithm using an interpolant]*. Tech. rep. Laboratoire Informatique de Paris 6 (LIP6), Feb. 2022.