Algorithme d'accélération par interpolants

Nathalie Sznajder, Souheib Baarir

21 février 2022

1 Formules utiles

Sur un anneau de taille n sur lequel évoluent k robots, la vue d'un robot est donné par un k-uplet de distances $\langle d_1, \ldots, d_k \rangle$ tel que $\Sigma_{i=1}^{k} d_i = n$. De plus on impose que $d_1 \neq 0$ (une distance entre deux robots est 0 si les deux robots partagent la même position. Si $d_1 = 0$, cela signifie que le robot dont on définit la vue est sur une tour de robots. Dans ce cas, on peut définir sa vue en mettant les distances à 0 en tout fin du k-uplet). Pour une vue $\mathbf{V}$, on note la vue $\overleftarrow{\mathbf{V}} = \langle d_j, \ldots, d_1, 0, \ldots, 0 \rangle$ la vue correspondant au cas où le robot regarde l'anneau dans la direction opposée, avec j le plus grand indice tel que $d_j \neq 0$.

Une stratégie (ou un algorithme) pour les robots peut être vu comme une fonction sur l'ensemble des vues tel que $\phi(\mathbf{V})$ vaut `true` si le robot doit bouger dans la direction définie par cette vue. Ainsi, si $\mathbf{V} \neq \overleftarrow{\mathbf{V}}$, on ne peut pas avoir à la fois $\phi(\mathbf{V})$ et $\phi(\overleftarrow{\mathbf{V}})$ qui sont vrais. Si $\mathbf{V} = \overleftarrow{\mathbf{V}}$, ce qui correspond à un robot désorienté, alors on aura $\phi(\mathbf{V}) = \phi(\overleftarrow{\mathbf{V}})$, c'est-à-dire que soit la formule est fausse dans tous les cas (si le robot est désorienté, il ne bouge pas), soit la formule est vraie dans tous les cas, et donc le robot va bouger, mais on ne sait pas dans quelle direction.

$$
\begin{gathered}
\texttt{ConfigView}_i(y, p_1, \ldots, p_k, d_1, \ldots, d_k) := \\
\exists d'_1, \ldots, d'_{k-1} \cdot \textstyle\bigwedge_{j=1}^{k-2} d'_j \leq d'_{j+1} \wedge \\
\textstyle\bigwedge_{j=1, j\neq i}^{k} (\bigvee_{\ell=1}^{k-1} p_j = (p_i + d'_\ell) \mod y) \wedge \\
\textstyle\bigwedge_{\ell=1}^{k} (\bigvee_{j=1, j\neq i}^{k} p_j = (p_i + d'_\ell) \mod y)) \wedge \\
0 < d'_1 \wedge \textstyle\bigwedge_{j=1}^{k-1} d'_j \leq y \wedge \\
d_1 = d'_1 \wedge \textstyle\bigwedge_{j=2}^{k-1} d_j = d'_j - d'_{j-1} \wedge d_k = y - d'_{k-1},
\end{gathered}
$$

avec y taille de l'anneau, $p_1, \ldots, p_k$ positions des robots, $d_1, \ldots, d_k$ vue du robot i en regardant dans le sens des aiguilles d'une montre.

$$
\begin{gathered}
\texttt{ViewSym}(d_1, \ldots, d_k, d'_1, \ldots, d'_k) := \\
\textstyle\bigvee_{j=1}^{k} (\bigwedge_{\ell=j+1}^{k} (d_\ell = 0 \wedge d'_\ell = 0) \wedge \bigwedge_{\ell=1}^{j} d'_\ell = d_{j-\ell+1})
\end{gathered}
$$

vrai si et seulement si $d_1, \ldots, d_k$ est la vue symétrique de $d'_1, \ldots, d'_k$.

$$
\begin{array}{c}
\texttt{Move}_i^{\phi}(y, p_1, \ldots, p_k, p') := \\
\exists d_1, \ldots, d_k, d'_1, \cdots, d'_k. \\
\texttt{ConfigView}_i(y, p_1, \ldots, p_k, d_1, \ldots, d_k) \wedge \\
\texttt{ViewSym}(d_1, \ldots, d_k, d'_1, \ldots, d'k) \wedge \\
\Big[\Big(\phi(d_1, \ldots, d_k) \wedge ((p_i < y - 1 \wedge p' = p_i + 1) \\
\vee (p_i = y - 1 \wedge p' = 0))\Big) \vee \\
\Big(\phi(d'_1, \ldots, d'_k) \wedge ((p_i > 0 \wedge p' = p_i - 1) \\
\vee (p_i = 0 \wedge p' = y - 1))\Big) \\
\vee \Big(\neg\phi(d_1, \ldots, d_k) \wedge \neg\phi(d'_1, \ldots, d'_k) \wedge (p' = p_i)\Big)\Big]
\end{array}
$$

vrai si et seulement si robot i se déplace à la position p' d'après l'algo ϕ et depuis la position $p_1, \ldots, p_k$.

$$
\begin{array}{c}
\texttt{AsyncPost}_{\phi}(y, p_1, \ldots, p_k, s_1, \ldots, s_k, t_1 \ldots, t_k, p'_1, \ldots, p'_k, s'_1, \ldots, s'_k, t'_1, \ldots, t'_k) := \\
\bigvee_{i=1}^{k}\Big(\bigwedge_{j \neq i}(p'_j = p_j \wedge s'_j = s_j \wedge t'_j = t_j) \wedge \\
((s_i = \texttt{RLC} \wedge \texttt{Move}_i^{\phi}(n, p_1, \ldots, p_k, s'_i) \wedge p'_i = p_i) \vee \\
(s_i \neq \texttt{RLC} \wedge p'_i = s_i \wedge s'_i = \texttt{RLC}) \\
\wedge(((\bigwedge_{j \neq i} t_j = 1) \wedge (\bigwedge_{j=1}^{k} t'_j = 0)) \vee ((\bigvee_{j \neq i} t_j = 0 \wedge t'_i = 1 \wedge \bigwedge_{j \neq i} t'_j = t_j)))\Big)
\end{array}
$$

Une configuration $c = (\mathbf{p}, \mathbf{s}, \mathbf{t})$ est perdante si elle appartient à une boucle perdante : une boucle équitable ne visitant jamais de configuration gagnante.

`BouclePerdante(p,s,t):=`

```
1  foreach stratégie gagnante en synchrone f do
2      k :=1
3      while true do
4          I(c) = Init(c)
5          continue = true
6          while continue do
7              if I(c) ∧ Post(c,c1) ∧ ... ∧ Post(c_{k-1}, c_k) ∧ BouclePerdante(c_k) SAT then
8                  if I=Init then
9                      exit;                    /* stratégie perdante en asynchrone */
10                 end
11                 else
12                     k :=k+1; /* inconclusive - recommencer en augmentant k */
13                     continue=false
14                 end
15             end
16             else
17                 I' := Interpolant(I(c) ∧ Post(c, c_1), Post(c_1, c_2) ∧ ... ∧
                    Post(c_{k-1}, c_k) ∧ BouclePerdante(c_k))
18                 if I'[c/c_1] ⇒ I then
19                     exit ;                              /* stratégie gagnante */
20                 end
21                 else
22                     I = I ∨ I'[c/c_1]
23                 end
24             end
25         end
26     end
27 end
```