Systèmes de transitions - Modélisation TLA$^+$

Durée 1h45 - Documents autorisés

6 mars 2018

Les réponses indiquées sont des éléments de correction, parfois sommaires. D'autres réponses correctes peuvent (parfois) exister. Il n'est pas exclu qu'une erreur se soit glissée dans cette correction.

1 Questions de cours (4 points)

Soit trois variables x, S, T. x est un entier, S une séquence finie d'entiers naturels, et T un ensemble fini d'entiers naturels. Répondre aux questions suivantes en respectant la syntaxe TLA$^+$. On rappelle que, outre les opérateurs définies sur les séquences (`Head`, `Tail`...), une séquence d'entiers est aussi une fonction de $[1..Len(S) \to Nat]$.

1. Donner un prédicat qui exprime que tous les éléments de S sont dans T.
   $\{S[i] : i \in 1..Len(S)\} \subseteq T$
   $\forall i \in 1..Len(S) : S[i] \in T$
2. Donner une action qui enlève de T les éléments plus petits ou égaux à x.
   $T' = T \setminus 0..x$
   $T' = \{i \in T : i > x\}$
3. Donner une propriété temporelle qui dit que S contient toujours au moins deux éléments.
   $\Box(Len(S) >= 2)$
4. Donner une propriété temporelle qui dit que x n'est jamais à la fois dans S et T.
   $\Box(\neg(x \in T \land x \in Range(S)))$
   $\Box(\neg(x \in T \land \exists i \in 1..Len(S) : S[i] = x))$

2 Exercice (4 points)

Soit le module TLA$^+$ ci-dessous définissant le système de transitions `Spec`.

MODULE *examen17_test*

EXTENDS *Naturals*
VARIABLES x, y

$TypeInvariant \triangleq x \in \text{SUBSET } Nat \land y \in Nat$

$GetY \triangleq y' \in x \land y' \neq y \land \text{UNCHANGED } x$
$ChangeX \triangleq y \leq 2 \land y + 1 \notin x \land x' = x \cup \{y + 1\} \land \text{ UNCHANGED } y$
$ResetXY \triangleq x' = \{0\} \land y' = 0$

$Fairness \triangleq \mathrm{SF}_{\langle x, y\rangle}(GetY) \wedge \mathrm{WF}_{\langle x, y\rangle}(ChangeX)$
$Init \triangleq x = \{0\} \wedge y = 0$
$Spec \triangleq Init \wedge \Box[GetY \vee ChangeX \vee ResetXY]_{\langle x, y\rangle} \wedge Fairness$

Indiquer si les propriétés suivantes, exprimées en logique LTL ou CTL, sont vérifiées. Justifier les réponses (argumentaire ou contre-exemple).

1. $\Box(Cardinality(x) \leq 4)$
2. $\Diamond(2 \in x)$
3. $\Box\Diamond(y \neq 0)$
4. $y = 1 \rightsquigarrow y = 2$
5. $\exists\Diamond(y = 2)$
6. $\exists\Box(y = 0)$
7. $\forall\Box(y \in x)$
8. $\forall\Box\exists\Diamond(x = \{0\} \wedge y = 0)$

Le graphe de transition est figure 1 (il n'est pas nécessaire de le dessiner pour répondre aux questions).

1. *OK : ChangeX ne permet de mettre que les nombre de 0 à 3 dans x*
2. *KO : en faisant $(ChangeX; GetY; ResetXY)^\omega$, on a : $\{0\}, 0 \to \{0,1\}, 0 \to \{0,1\}, 1 \to$ back to 1*
3. *OK : équité ChangeX puis GetY.*
4. *KO : en faisant $(ChangeX; GetY; ResetXY)^\omega : \Box(y \in \{0,1\})$.*
5. *OK : $y = 2$ accessible, cf graphe ou exécution $\{0\}, 0 \to \{0,1\}, 0 \to \{0,1\}, 1 \to \{0,1,2\}, 1 \to \{0,1,2\}, 2 \to \cdots$.*
6. *KO : l'équité force à passer par $y = 1$*
7. *OK : par induction : GetY garantit $y' \in x'$; ChangeX fait augmenter x (donc la propriété $y \in x$ est conservée) ; ResetXY garantit $y' \in x'$. Donc $y \in x \wedge Next \Rightarrow y' \in x'$. Comme $Init \Rightarrow y \in x$, on a donc $\Box(y \in x)$ et $\forall\Box(y \in x)$.*
8. *OK : réinitialisation toujours possible via ResetXY.*

3 Problème (12 points [1])

On souhaite modéliser et étudier le problème de l'élection. Initialement, un certain nombre de personnes sont candidats. À chaque tour, chaque personne vote pour un candidat (arbitrairement). Quand tous ont voté, on élimine un candidat parmi ceux ayant eu le moins de votes. Quand il ne reste plus qu'un candidat, il est élu.

Un squelette de module TLA$^+$ `election.tla` est fourni à la fin du sujet. Noter que les variables `votants` et `votes` contiennent les valeurs pour le tour courant, alors que la variable `candidats` est une séquence qui garde trace de l'évolution de l'ensemble des candidats. `Head(candidats)` est l'ensemble des candidats initiaux et `Last(candidats)` est l'ensemble des candidats au tour courant. Se souvenir qu'une séquence s peut aussi être manipulée comme une fonction $[1..Len(s) \to \ldots]$.

1. Toutes les questions valent autant sauf la 13 qui vaut double.

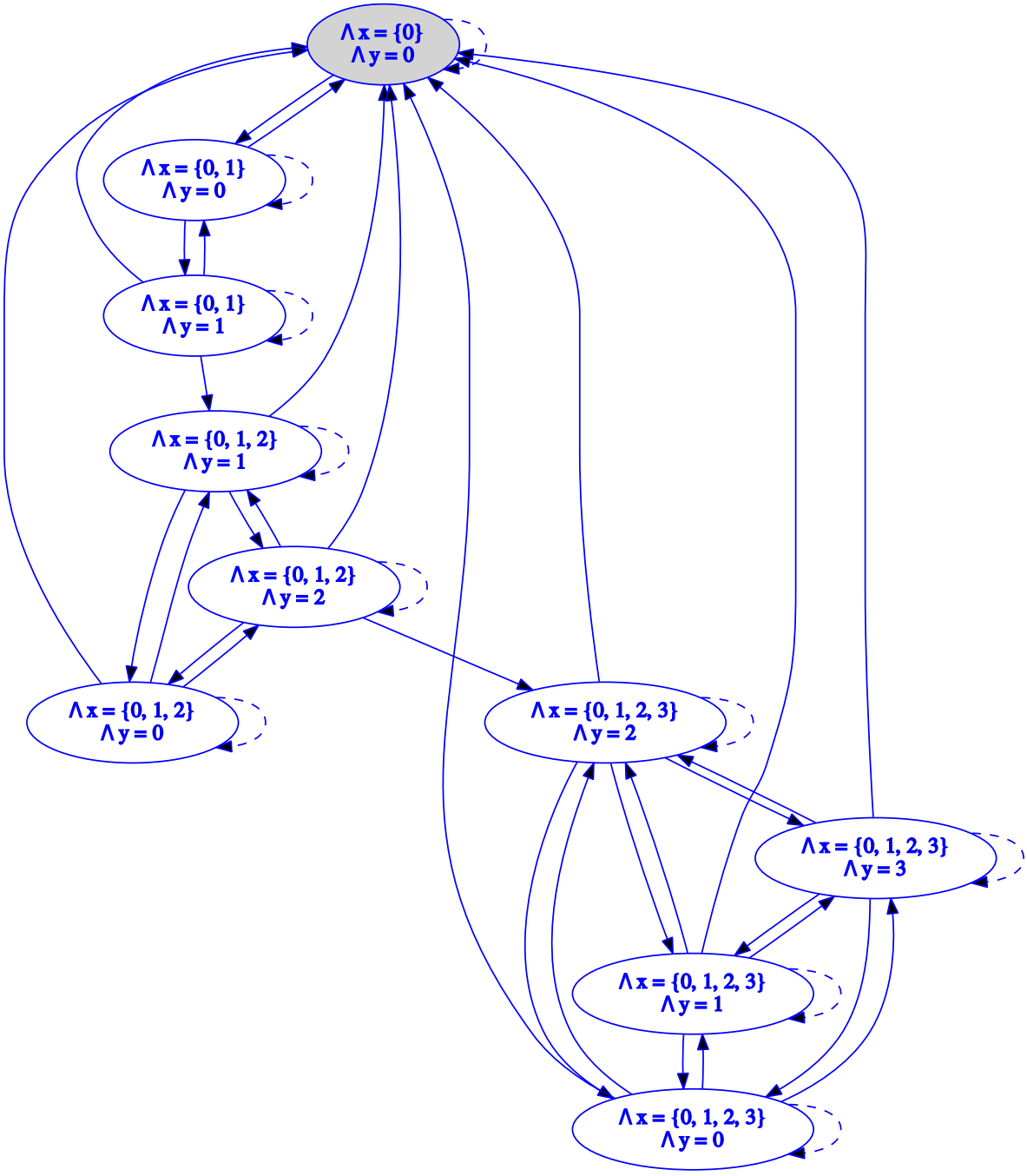

FIGURE 1 – Graphe de transition de `test`

3.1 Module complet

1. Écrire l'action `finelection`, faisable quand il ne reste qu'un seul candidat et qui positionne alors `elu` avec celui-ci.
   $$\begin{aligned} &\mathit{finelection} \triangleq \\ &\quad \land \mathit{Cardinality}(\mathit{Last}(\mathit{candidats})) = 1 \\ &\quad \land \mathit{elu}' \in \mathit{Last}(\mathit{candidats}) \\ &\quad \land \text{UNCHANGED } \langle \mathit{votes}, \mathit{votants}, \mathit{candidats}, \mathit{tour} \rangle \end{aligned}$$
2. Définir le prédicat de transitions `Next` qui représente toutes les transitions possibles.
   $$\begin{aligned} \mathit{Next} \triangleq \; & \lor \exists\, i \in \mathit{Population} : \mathit{vote}(i) \\ & \lor \mathit{fintour} \\ & \lor \mathit{finelection} \end{aligned}$$
3. Définir la propriété `Spec` qui décrit le système de transitions (sans équité).
   $\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{Next}]_{\mathit{vars}} \land \mathit{Fairness}$

3.2 Spécification

Exprimer en TLA$^+$ les propriétés suivantes (qui ne sont pas nécessairement vérifiées par le modèle) :

4. `NombreToursMax` : Le nombre de tours nécessaires est plus petit que la taille de la population (N).
   $\mathit{NombreToursMax} \triangleq \Box(\mathit{tour} \leq N)$
5. `EluParmiCandidats` : s'il y a un élu, il fait parti des candidats présents au premier tour.
   $\mathit{EluParmiCandidats} \triangleq \Box(\mathit{elu} \neq \mathit{NoChoice} \Rightarrow \mathit{elu} \in \mathit{Head}(\mathit{candidats}))$
6. `CandidatsDecroit` : l'ensemble des candidats décroît à chaque tour.
   $\Box(\forall\, i, j \in \text{DOMAIN } \mathit{candidats} : i < j \Rightarrow (\mathit{candidats}[j] \subseteq \mathit{candidats}[i] \land \mathit{candidats}[j] \neq \mathit{candidats}[i]))$
7. `FinalementÉlu` : il y a finalement un élu.
   $\mathit{FinalementElu} \triangleq \Diamond(\mathit{elu} \neq \mathit{NoChoice})$
8. `PasDAbstention` : un votant ne peut pas s'abstenir, c'est-à-dire que la somme des votes courants est égale au nombre de votants (voir l'opérateur `Somme` défini dans le module).
   $\mathit{PasDAbstention} \triangleq \Box(\mathit{Cardinality}(\mathit{votants}) = \mathit{Somme}(\mathit{votes}))$

3.3 Équité

9. Énoncer l'équité minimale qui vous semble nécessaire pour que la propriété `FinalementÉlu` soit vérifiée.
   Il faut de l'équité sur chaque action : pour qu'un électeur qui peut voter finisse par le faire, pour que quand tous ont voté le tour et l'élection finissent. Sinon, il y a possibilité de bégaiement infini. L'équité faible suffit car chaque action faisable reste continûment faisable tant qu'elle n'est pas faite. Donc :
   $\forall i \in \mathit{Population} : \mathit{WF}_{\mathit{vars}}(\mathit{vote}(i)) \land \mathit{WF}_{\mathit{vars}}(\mathit{fintour}) \land \mathit{WF}_{\mathit{vars}}(\mathit{finelection})$
   (en fait $\mathit{WF}_{\mathit{vars}}(\mathit{Next})$ *fonctionne aussi !)*

3.4 Évolution : `election2.tla`

On souhaite maintenant qu'un électeur vote pour lui-même s'il est candidat.

10. Modifier l'ancienne action `vote` pour qu'elle ne soit pas faisable si l'électeur est candidat.
    Ajouter : $i \notin Last(candidats)$.
11. Écrire une nouvelle action `vote2(i)` où un électeur vote pour lui-même s'il est candidat.
    $vote2(i) \triangleq$
    $\land\ Cardinality(Last(candidats)) \geq 2$
    $\land\ i \notin votants$
    $\land\ i \in Last(candidats)$
    $\land\ votes' = [votes \text{ EXCEPT } ![i] = votes[i] + 1]$
    $\land\ votants' = votants \cup \{i\}$
    $\land\ \text{UNCHANGED } \langle candidats, elu, tour \rangle$
12. Cette action `vote2` est ajoutée à `Next`. Quelle relation existe-t-il entre les spécifications `election1!Spec` et `election2!Spec` ?
    C'est un raffinage : $election2!Spec \Rightarrow election1!Spec$.

3.5 Vérification de `election2.tla`

13. Dessiner le graphe de transitions pour le cas particulier suivant : $N = 3$, candidats initiaux $= \{1, 2\}$ (16 états).
    Voir figure 2.
14. Comment vérifie-t-on la propriété `CandidatsDecroit` en examinant le graphe ? La propriété est-elle vérifiée ?
    En regardant chaque transition. Oui, vérifiée.
15. Comment vérifie-t-on la propriété `FinalementElu` en examinant le graphe ? La propriété est-elle vérifiée ?
    Pas de cycle infini hors ceux du bégaiement, éliminés avec l'équité faible $\Rightarrow$ propriété vérifiée.

3.6 Évolution : `election3.tla`

Cette élection est injuste : si tous les candidats obtiennent le même nombre de voix, on en élimine pourtant un quelconque avant de passer au tour suivant. On souhaite maintenant que si tous les candidats ont le même nombre de voix, on repart pour un nouveau tour (en remettant les votes à 0).

16. Ajouter une action `recommencer` qui remet les votes et les votants à zéro si tous les candidats ont autant de voix.
    On peut au choix incrémenter tour ou pas. Dans le premier cas, on obtient un système ayant un nombre infini d'états et candidats ne décroît plus strictement.
    recommencer $\triangleq$
    $\land\ votants = Population$
    $\land\ Cardinality(Last(candidats)) \geq 2$
    $\land\ \forall\, i, j \in Last(candidats) : votes[i] = votes[j]$

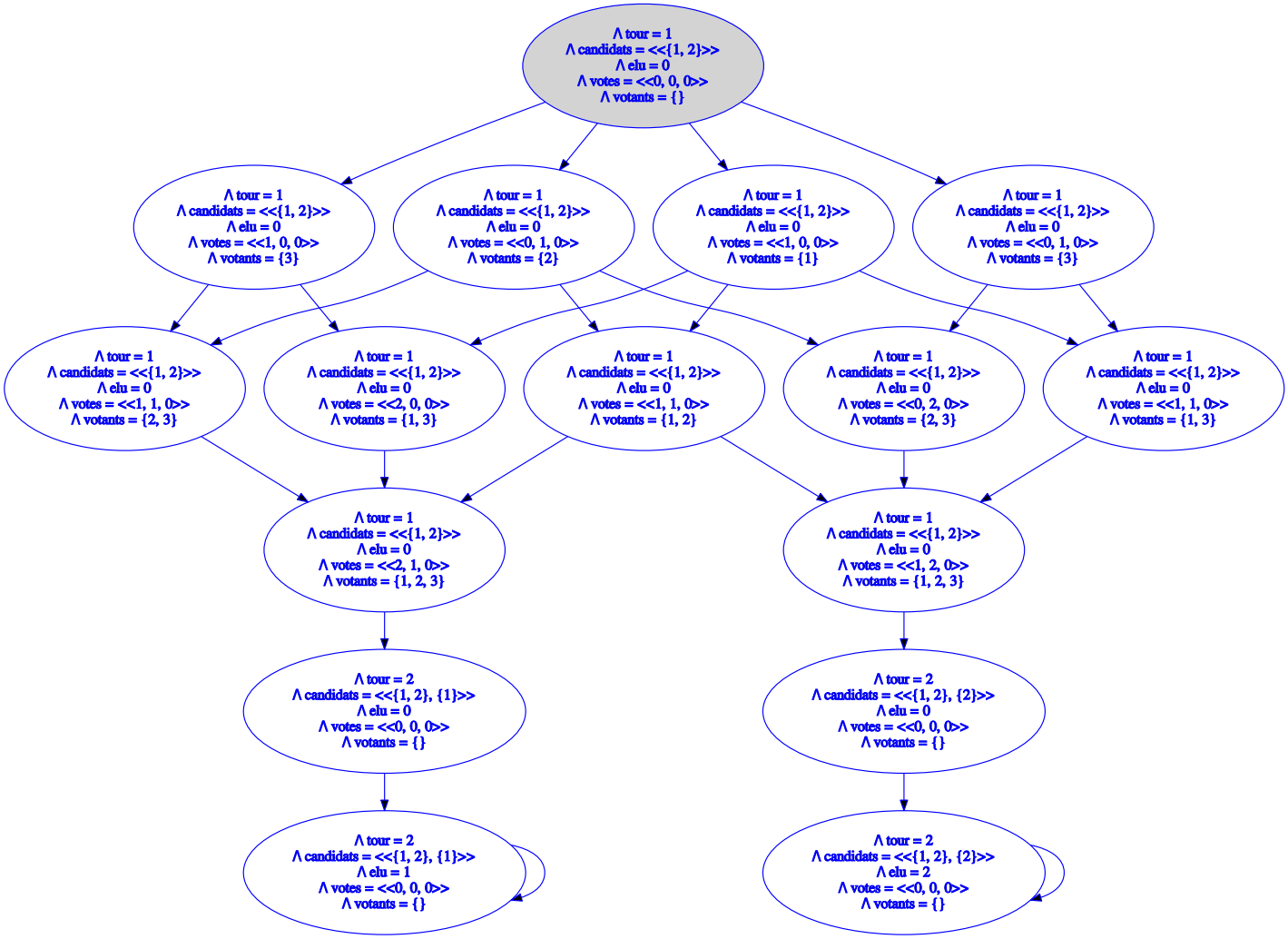

FIGURE 2 – Graphe de transition de `election2`

$\land\ votes' = [i \in Population \mapsto 0]$
$\land\ votants' = \{\}$
$\land$ UNCHANGED $\langle elu,\ candidats,\ tour\rangle$

17. Modifier l'ancienne action `fintour` pour qu'elle ne soit pas faisable si tous les candidats ont autant de voix.
Ajouter : $\neg(\forall i, j \in Last(candidats) : votes[i] = votes[j])$

18. Parmi les propriétés `NombreToursMax`, `EluParmiCandidats`, `CandidatsDecroit`, `FinalementÉlu` et `PasDAbstention`, quelles sont celles qui ne sont plus vérifiées et pourquoi ?
NombreToursMax et FinalementÉlu (et selon sa forme, CandidatsDecroit) : 2 électeurs, tous deux candidats ⇒ chacun une voix et on recommence.

19. Proposer une adaptation (éventuellement injuste) pour que `FinalementÉlu` soit vérifiée.
   — *Quand on atteint N tours, on prend un candidat au hasard.*
   — *Quand il n'y pas eu de changement deux tours consécutifs, on élimine un candidat au hasard*
   — ...

3.7 Module fourni : `election1.tla`

MODULE *election1*

EXTENDS *Naturals*, *FiniteSets*, *Sequences*

CONSTANT N

$Population \triangleq 1\,..\,N$
$NoChoice \triangleq 0$
ASSUME $NoChoice \notin Population$

VARIABLES

tour,	tour courant
candidats,	séquence avec l'ensemble des *candidats* de chaque tour
votes,	les votes du tour courant
votants,	les *votants* du tour courant
elu	l'élu final

$vars \triangleq \langle candidats,\ votes,\ votants,\ elu,\ tour\rangle$

Renvoie le dernier élément de la séquence *seq*.
Last(*candidats*) est l'ensemble des *candidats* du tour courant.
$Last(seq) \triangleq seq[Len(seq)]$

$TypeInvariant \triangleq$
$\land\ tour \in Nat$
$\land\ candidats \in Seq(\text{SUBSET } Population)$
$\land\ votes \in [Population \to Nat]$
$\land\ votants \in \text{SUBSET } Population$
$\land\ elu \in Population \cup \{NoChoice\}$

```
  Calcul la somme des éléments du tableau T
  Ainsi Somme(votes) = la somme des votes.
Somme(T) ≜
  LET sommetab[S ∈ SUBSET DOMAIN T] ≜
        IF S = {} THEN 0
         ELSE LET x ≜ CHOOSE x ∈ S : TRUEIN
                T[x] + sommetab[S \ {x}]
  IN    sommetab[DOMAIN T]
```

```
Init ≜
  ∧ tour = 1
  ∧ ∃ S  ∈ SUBSET Population :
      ∧ S ≠ {}
      ∧ candidats = ⟨S⟩
  ∧ votants = {}
  ∧ votes = [i ∈ Population ↦ 0]
  ∧ elu = NoChoice

  L'électeur i vote pour l'un des candidats
vote(i) ≜
  ∧ Cardinality(Last(candidats)) ≥ 2
  ∧ i ∉ votants
  ∧ ∃ c ∈ Last(candidats) : votes' = [votes EXCEPT ![c] = votes[c] + 1]
  ∧ votants' = votants ∪ {i}
  ∧ UNCHANGED ⟨candidats, elu, tour⟩

  Quand tout le monde a voté, fin du tour : on élimine l'un des candidats parmi ceux ayant le moins de voix
fintour ≜
  ∧ votants = Population
  ∧ tour' = tour + 1
  ∧ ∃ elimine ∈ Last(candidats) :
      ∧ ∀ cc ∈ Last(candidats) : votes[elimine] ≤ votes[cc]
      ∧ candidats' = Append(candidats, Last(candidats) \ {elimine})
  ∧ votes' = [i ∈ Population ↦ 0]
  ∧ votants' = {}
  ∧ UNCHANGED ⟨elu⟩

  Fini quand il ne reste plus qu'un candidat.
finelection ≜
    XXXX TODO XXXX

Next ≜ TODO
Spec ≜ TODO
```