Génération de spécification formelle pour l’algorithme Egalitarian Paxos

Projet ASR - Alexandre SIRET

Tuteur - Pierre SUTRA

Plan

- Rappel du contexte du projet.
- Egalitarian Paxos, qu’est ce que ça fait de plus par rapport à Paxos?
- Comment j’ai simuler l’algorithme en TLA+?
- Résultats et utilisation de GCP.
- À quel point l’IA à été utile?

Contexte

EPaxos = Algo de Consensus

Contexte

EPaxos = Algo de Consensus

Making Democracy Work: Fixing and Simplifying Egalitarian Paxos

Fedor Ryabinin - Alexey Gotsman - Pierre Sutra

Contexte

EPaxos = Algo de Consensus

Making Democracy Work: Fixing and Simplifying Egalitarian Paxos
Fedor Ryabinin - Alexey Gotsman - Pierre Sutra

L'objectif est de rajouter une couche de certitude sur la correction de l'algorithme grâce à la vérification formelle.

EPaxos > Paxos?

- On n’a pas besoin d’élire un leader

EPaxos > Paxos?

- On n’a pas besoin d’élire un leader
- On prend en compte le fait que des commandes sont commutatives:
  - Ordre partiel par rapport à ordre total
  - Possibilité de prendre un fast path

EPaxos > Paxos?

- On n’a pas besoin d’élire un leader

- On prend en compte le fait que des commandes sont commutatives:
    - Ordre partiel par rapport à ordre total
    - Possibilité de prendre un fast path

- Mais l’algorithme est plus compliqué à implémenter

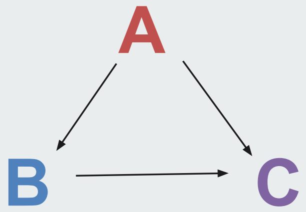
EPaxos : Partie Execution

$A = x \leftarrow 42$ $B = y \leftarrow x + 7$ $C = z \leftarrow x + y$

$dep(A) = \varnothing$

$dep(B) = \{A\}$

$dep(C) = \{A, B\}$



Éxecute les commandes localement en suivant l'ordre partiel du graphe des dépendances

while true
- **let** $G \subseteq \mathsf{dep}$ be the largest subgraph such that $\forall id \in G.\, \mathsf{phase}[id] = \text{COMMITTED} \wedge \mathsf{dep}[id] \subseteq G$
- **for** $C \in \mathsf{SCC}(G)$ in topological order **do**
  - **for** $id \in C$ in the order of command identifiers **do**
    - **if** $id \notin \mathsf{executed} \wedge \mathsf{cmd}[id] \neq \mathsf{Nop}$ **then**
      - $\mathtt{execute}(\mathsf{cmd}[id])$
      - $\mathsf{executed} \leftarrow \mathsf{executed} \cup \{id\}$

EPaxos : Partie Exécution

while true
  let $G \subseteq \mathsf{dep}$ be the largest subgraph such that $\forall id \in G.\, \mathsf{phase}[id] = \textsc{committed} \wedge \mathsf{dep}[id] \subseteq G$
  for $C \in \mathsf{SCC}(G)$ in topological order **do**
    for $id \in C$ in the order of command identifiers **do**
      if $id \notin \mathsf{executed} \wedge \mathsf{cmd}[id] \neq \mathsf{Nop}$ **then**
        $\mathtt{execute}(\mathsf{cmd}[id])$
        $\mathsf{executed} \leftarrow \mathsf{executed} \cup \{id\}$

Simple car…

- L’algorithme s’exécute localement, pas besoin de gérer la communication entre les processus.

Compliqué car…

- **for** $C \in \mathsf{SCC}(G)$ in topological order
- $G \subseteq \mathsf{dep}$ be the largest subgraph such that

EPaxos : Partie Commit

Construit les graphes de dépendances pour la partie exécution.

```
submit(c):
    let id = new_id()
    send PreAccept(id, c, {id' | cmd[id'] ⋈ c})
      to all

when received PreAccept(id, c, D) from q
    pre: bal[id] = 0 ∧ phase[id] = INITIAL
    cmd[id] ← c
    initCmd[id] ← c
    initDep[id] ← D
    dep[id] ← D ∪ {id' | cmd[id'] ⋈ cmd[id]}
    phase[id] ← PREACCEPTED
    send PreAcceptOK(id, dep[id]) to q

when received PreAcceptOK(id, D_q)
  from all q ∈ Q
    pre: bal[id] = 0 ∧ phase[id] = PREACCEPTED ∧
           |Q| ≥ n − f
    let D = ⋃_{q∈Q} D_q
    if |Q| ≥ n − e ∧ ∀q ∈ Q. D_q = initDep[id] then
        send Commit(0, id, cmd[id], D) to all
    else
        send Accept(0, id, cmd[id], D) to all

when received Accept(b, id, c, D) from q
    pre: bal[id] ≤ b ∧
           (bal[id] = b ⟹ phase[id] ≠ COMMITTED)
    bal[id] ← b
    abal[id] ← b
    cmd[id] ← c
    dep[id] ← D
    phase[id] ← ACCEPTED
    send AcceptOK(b, id) to q

when received AcceptOK(b, id) from Q
    pre: bal[id] = b ∧ phase[id] = ACCEPTED ∧
           |Q| ≥ n − f
    send Commit(b, id, cmd[id], dep[id]) to all

when received Commit(b, id, c, D) from q
    pre: bal[id] = b
    abal[id] ← b
    cmd[id] ← c
    dep[id] ← D
    phase[id] ← COMMITTED
```

Simple car...

- Chaque ligne du pseudo code est simple à retranscrire individuellement.

Compliqué car...

- Il faut simuler le fait qu'on ait plusieures processus qui communiquent entre eux.

EPaxos : Partie Recovery

```
43 recover(id):
44   let b = (a ballot owned by p such that b > bal[id])
45   send Recover(b, id) to all

46 when received Recover(b, id) from q
47   pre: bal[id] < b
48   bal[id] ← b
49   send RecoverOK(b, id, abal[id], cmd[id], dep[id], initDep[id], phase[id]) to q
```

```
81 when received Validate(b, id, c, D) from q
82   pre: bal[id] = b
83   cmd[id] ← c
84   initCmd[id] ← c
85   initDep[id] ← D
86   let I = {(id′, phase[id′]) | id′ ≠ id ∧ id′ ∉ D ∧
              (phase[id′] = COMMITTED ⟹ cmd[id′] ≠ Nop ∧ cmd[id′] ⋈ c ∧ id ∉ dep[id′]) ∧
              (phase[id′] ≠ COMMITTED ⟹ initCmd[id′] ≠ ⊥ ∧ initCmd[id′] ⋈ c ∧ id ∉ initDep[id′])}
87   send ValidateOK(b, id, I) to all
```

```
50 when received RecoverOK(b, id, abal_q, c_q, dep_q, initDep_q, phase_q) from all q ∈ Q
51   pre: bal[id] = b ∧ |Q| ≥ n − f
52   let b_max = max{abal_q | q ∈ Q}
53   let U = {q ∈ Q | abal_q = b_max}
54   if ∃q ∈ U. phase_q = COMMITTED then send Commit(b, id, c_q, dep_q) to all
55   else if ∃q ∈ U. phase_q = ACCEPTED then send Accept(b, id, c_q, dep_q) to all
56   else if initCoord(id) ∈ Q then send Accept(b, id, Nop, ∅) to all
57   else if ∃R ⊆ Q. |R| ≥ |Q| − e ∧ ∀q ∈ R. (phase_q = PREACCEPTED ∧ dep_q = initDep_q) then
58     let R_max be the largest set R that satisfies the condition at line 57
59     let (c, D) = (c_q, dep_q) for any q ∈ R
60     send Validate(b, id, c, D) to all processes in Q
61     wait until received ValidateOK(b, id, I_q) from all q ∈ Q
62     let I = ⋃_{q∈Q} I_q
63     if I = ∅ then
64       send Accept(b, id, c, D) to all
65     else if (∃(id′, COMMITTED) ∈ I) ∨ (|R_max| = |Q| − e ∧ ∃(id′, _) ∈ I. initCoord(id′) ∉ Q) then
66       send Accept(b, id, Nop, ∅) to all
67     else
68       send Waiting(id, |R_max|) to all
69       wait until
70         case ∃(id′, _) ∈ I. phase[id′] = COMMITTED ∧ (cmd[id′] ≠ Nop ∧ id ∉ dep[id′]) do
71           send Accept(b, id, Nop, ∅) to all
72         case ∀(id′, _) ∈ I. phase[id′] = COMMITTED ∧ (cmd[id′] = Nop ∨ id ∈ dep[id′]) do
73           send Accept(b, id, c, D) to all
74         case ∃(id′, _) ∈ I. (p received Waiting(id′, k′)) ∧ k′ > n − f − e do
75           send Accept(b, id, Nop, ∅) to all
76         case p received RecoverOK(b, id, _, cmd, dep, _, phase) from q ∉ Q with
           phase = COMMITTED ∨ phase = ACCEPTED ∨ q = initCoord(id) do
77           if phase = COMMITTED then send Commit(b, id, cmd, dep) to all
78           else if phase = ACCEPTED then send Accept(b, id, cmd, dep) to all
79           else send Accept(b, id, Nop, ∅) to all
80   else send Accept(b, id, Nop, ∅) to all
```

On a les difficultés des deux parties précédentes!

EPaxos : Commit + Recovery

```
 8 submit(c):
 9   let id = new_id()
10   send PreAccept(id, c, {id′ | cmd[id′] ⋈ c})
       to all
11 when received PreAccept(id, c, D) from q
12   pre: bal[id] = 0 ∧ phase[id] = INITIAL
13   cmd[id] ← c
14   initCmd[id] ← c
15   initDep[id] ← D
16   dep[id] ← D ∪ {id′ | cmd[id′] ⋈ cmd[id]}
17   phase[id] ← PREACCEPTED
18   send PreAcceptOK(id, dep[id]) to q
19 when received PreAcceptOK(id, D_q)
     from all q ∈ Q
20   pre: bal[id] = 0 ∧ phase[id] = PREACCEPTED ∧
          |Q| ≥ n − f
21   let D = ⋃_{q∈Q} D_q
22   if |Q| ≥ n − e ∧ ∀q ∈ Q. D_q = initDep[id] then
23     send Commit(0, id, cmd[id], D) to all
24   else
25     send Accept(0, id, cmd[id], D) to all
26 when received Accept(b, id, c, D) from q
27   pre: bal[id] ≤ b ∧
          (bal[id] = b ⟹ phase[id] ≠ COMMITTED)
28   bal[id] ← b
29   abal[id] ← b
30   cmd[id] ← c
31   dep[id] ← D
32   phase[id] ← ACCEPTED
33   send AcceptOK(b, id) to q
34 when received AcceptOK(b, id) from Q
35   pre: bal[id] = b ∧ phase[id] = ACCEPTED ∧
          |Q| ≥ n − f
36   send Commit(b, id, cmd[id], dep[id]) to all
37 when received Commit(b, id, c, D) from q
38   pre: bal[id] = b
39   abal[id] ← b
40   cmd[id] ← c
41   dep[id] ← D
42   phase[id] ← COMMITTED
```

```
43 recover(id):
44   let b = (a ballot owned by p such that b > bal[id])
45   send Recover(b, id) to all
46 when received Recover(b, id) from q
47   pre: bal[id] < b
48   bal[id] ← b
49   send RecoverOK(b, id, abal[id], cmd[id], dep[id], initDep[id], phase[id]) to q
```

```
50 when received RecoverOK(b, id, abal_q, c_q, dep_q, initDep_q, phase_q) from all q ∈ Q
51   pre: bal[id] = b ∧ |Q| ≥ n − f
52   let b_max = max{abal_q | q ∈ Q}
53   let U = {q ∈ Q | abal_q = b_max}
54   if ∃q ∈ U. phase_q = COMMITTED then send Commit(b, id, c_q, dep_q) to all
55   else if ∃q ∈ U. phase_q = ACCEPTED then send Accept(b, id, c_q, dep_q) to all
56   else if initCoord(id) ∈ Q then send Accept(b, id, Nop, ∅) to all
57   else if ∃R ⊆ Q. |R| ≥ |Q| − e ∧ ∀q ∈ R. (phase_q = PREACCEPTED ∧ dep_q = initDep_q) then
58     let R_max be the largest set R that satisfies the condition at line 57
59     let (c, D) = (c_q, dep_q) for any q ∈ R
60     send Validate(b, id, c, D) to all processes in Q
61     wait until received ValidateOK(b, id, I_q) from all q ∈ Q
62     let I = ⋃_{q∈Q} I_q
63     if I = ∅ then
64       send Accept(b, id, c, D) to all
65     else if (∃(id′, COMMITTED) ∈ I) ∨ (|R_max| = |Q| − e ∧ ∃(id′, _) ∈ I. initCoord(id′) ∉ Q) then
66       send Accept(b, id, Nop, ∅) to all
67     else
68       send Waiting(id, |R_max|) to all
69       wait until
70         case ∃(id′, _) ∈ I. phase[id′] = COMMITTED ∧ (cmd[id′] ≠ Nop ∧ id ∉ dep[id′]) do
71           send Accept(b, id, Nop, ∅) to all
72         case ∀(id′, _) ∈ I. phase[id′] = COMMITTED ∧ (cmd[id′] = Nop ∨ id ∈ dep[id′]) do
73           send Accept(b, id, c, D) to all
74         case ∃(id′, _) ∈ I. (p received Waiting(id′, k′)) ∧ k′ > n − f − e do
75           send Accept(b, id, Nop, ∅) to all
76         case p received RecoverOK(b, id, _, cmd, dep, _, phase) from q ∉ Q with
           phase = COMMITTED ∨ phase = ACCEPTED ∨ q = initCoord(id) do
77           if phase = COMMITTED then send Commit(b, id, cmd, dep) to all
78           else if phase = ACCEPTED then send Accept(b, id, cmd, dep) to all
79           else send Accept(b, id, Nop, ∅) to all
80   else send Accept(b, id, Nop, ∅) to all
```

```
81 when received Validate(b, id, c, D) from q
82   pre: bal[id] = b
83   cmd[id] ← c
84   initCmd[id] ← c
85   initDep[id] ← D
86   let I = {(id′, phase[id′]) | id′ ≠ id ∧ id′ ∉ D ∧
             (phase[id′] = COMMITTED ⟹ cmd[id′] ≠ Nop ∧ cmd[id′] ⋈ c ∧ id ∉ dep[id′]) ∧
             (phase[id′] ≠ COMMITTED ⟹ initCmd[id′] ≠ ⊥ ∧ initCmd[id′] ⋈ c ∧ id ∉ initDep[id′])}
87   send ValidateOK(b, id, I) to all
```

MAIS J'AI VAINCU

Execution Protocol

Commit Protocol

Recovery Protocol

Commit + Recovery Protocol

Simulation de plusieurs processus en TLA+

Comment simuler l’espace local de chaque processus?

when received $\texttt{PreAccept}(id, c, D)$ **from** q

  pre: $\mathsf{bal}[id] = 0 \land \mathsf{phase}[id] = \text{INITIAL}$

  $\mathsf{cmd}[id] \leftarrow c$

On met à jour cmd[id] dans la **mémoire locale du processus** qui reçoit le PreAccept

Simulation de l’algo en TLA+

cmd[id]

cmd

$$\begin{array}{cc} 1 & \\ 2 & \\ 3 & \end{array}\begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

(Dans chaque processus)

$$\Longrightarrow$$

cmd[p][id]

cmd

$$\begin{array}{c|c} & \begin{array}{ccc} p1 & p2 & p3 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \begin{pmatrix} x & \varnothing & x \\ y & \varnothing & \varnothing \\ \varnothing & z & z \end{pmatrix} \end{array}$$

Simulation de l’algo en TLA+

Comment simuler la communication entre mes processus?

submit(c):

- **let** $id = \texttt{new\_id}()$
- **send** $\texttt{PreAccept}(id, c, \{id' \mid \mathsf{cmd}[id'] \bowtie c\})$ **to all**

when received $\texttt{PreAccept}(id, c, D)$ **from** q

- **pre:** $\mathsf{bal}[id] = 0 \wedge \mathsf{phase}[id] = \text{INITIAL}$
- $\mathsf{cmd}[id] \leftarrow c$

Simulation de l'algo en TLA+

Comment simuler la communication entre mes processus?

$$\textbf{send}\ \mathtt{PreAccept}(id, c, \{id' \mid \mathsf{cmd}[id'] \bowtie c\})$$

```
/\ msgs' = msgs \cup
      { PreAcceptMsg(p, q, id, c, D0, 0) : q \in Proc }
```

PreAcceptMsg a pour paramètres les paramètres du pseudo-code + le sender p et le destinataire q

Simulation de l'algo en TLA+

Comment simuler la communication entre mes processus?

when received PreAccept(id, c, D) **from** q

```
\/ \E m \in msgs :
      \/ HandlePreAccept(m)
```

```
HandlePreAccept(m) ==
    /\ m.type = TypePreAccept
```

```
/\ msgs' = msgs \ {m}
```

Je peux lancer l'opération when received PreAccept() correctement en récupérant p et q du message m.

Invariants/Propriétées

Agreement : Si un id de commande est comité dans deux processus différents, alors c'est la même commande

Visibilité : Si deux id de commandes conflictuelles id et id' sont comité, alors id est dans les dépendances de id', ou id' est dans les dépendances de id.

```
Agreement ==
  \A id \in Id :
    \A p, q \in Proc :
      /\ phase[p][id] = "committed"
      /\ phase[q][id] = "committed"
      => /\ dep[p][id] = dep[q][id]
         /\ cmd[p][id] = cmd[q][id]
```

```
Visibility ==
  \A id, id2 \in Id : \E p, q \in Proc :
    /\ id # id2
    /\ phase[p][id] = "committed"
    /\ phase[q][id2] = "committed"
    /\ Conflicts(cmd[p][id], cmd[q][id2])
    => \/ id \in dep[q][id2]
       \/ id2 \in dep[p][id]
```

Liveness

Une commande soumise sera toujours exécuté par tous les processus corrects au bout d’un certain temps.

```
Liveness ==
    \A id \in Id :
      id \in submitted
      => \E p \in Proc :
           phase[p][id] = "committed"
```

C’est quoi le model checker?

D’abord, un état :

Un état est défini par la description des valeurs de toutes les variables.

```
/\ phase = <<<<"Initial", "Initial">>, <<"Initial", "Initial">>>>
/\ abal = <<<<0, 0>>, <<0, 0>>>>
/\ bal = <<<<1, 0>>, <<1, 0>>>>
/\ dep = <<<<{}, {}>>, <<{}, {}>>>>
/\ submitted = {}
/\ cmd = <<<<"Nop", "Nop">>, <<"Nop", "Nop">>>>
/\ initDep = <<<<{}, {}>>, <<{}, {}>>>>
/\ msgs = { [type |-> "Recover", from |-> 1, to |-> 1, body |-> [id |-> 1, b |-> 1]],
  [ type |-> "RecoverOK",
    from |-> 2,
    to |-> 1,
    body |->
        [ id |-> 1,
          b |-> 1,
          abalq |-> 0,
          cq |-> "Nop",
          depq |-> {},
          initDepq |-> {},
          phaseq |-> "Initial" ] ] }
/\ initCmd = <<<<"Nop", "Nop">>, <<"Nop", "Nop">>>>
/\ recovered = <<<<1, 0>>, <<0, 0>>>>
/\ initCoord = <<"NoProc", "NoProc">>
```

C’est quoi le model checker?

Le model checker change d’état grâce au opérations que j’ai défini en suivant le pseudo code:

Une opération change les valeurs d’une ou plusieures variables ie change l’état.

```
Next ==
    \/ \E m \in msgs :
          \/ HandlePreAccept(m)
          \/ HandlePreAcceptOK(m)
          \/ HandleAccept(m)
          \/ HandleAcceptOK(m)
          \/ HandleCommit(m)

          \/ HandleRecover(m)
          \/ HandleRecoverOK(m)
          \/ HandleValidate(m)
          \/ HandleValidateOK(m)
          \/ HandlePostWaitingMsg(m)


    \/ \E q \in Proc, id \in Id, c \in Cmd :
          Submit(q, id, c)

    \/ \E p \in Proc, id \in Id :
          StartRecover(p, id)
```

C’est quoi le model checker?

Le model checker prend une configuration initiale et parcourt toute les possibilités, et vérifie les invariants sur chacun des états que le système peut prendre.

```
CONSTANTS
    Proc = {1, 2}                          \* Set of processes
    F = 0                                 \* Max # crash failures
    E = 0                                 \* e-fast parameter (E <= F)
    Cmd = {A, B}                             \* Command payloads
    Id = {1, 2}                              \* Command identifiers
    NoCmd = "NoCmd"                       \* Special value representing no command
    NoProc = "NoProc"                     \* Special value representing no process


INVARIANT Agreement
INVARIANT Visibility
PROPERTY Liveness
```

Démo

Résultats et Utilisation de GCP

2 processus 2 commandes :

```
Progress(28) at 2026-01-20 16:25:57: 121,844,076 states generated (2,350,417 s/min), 39,840,192 distinct states found (924,467 ds/min), 997,806 states left on queue.
Progress(30) at 2026-01-20 16:26:57: 124,000,006 states generated (2,155,930 s/min), 40,734,978 distinct states found (894,786 ds/min), 341,715 states left on queue.
Model checking completed. No error has been found.
```

Prochain objectif : 3 processus 2 commandes :

Ca ne tourne pas sur mon ordi :

=> Tentative d’utilisation de GCP pour une machine plus puissante.

Résultats et Utilisation de GCP

Type de machine	n2-standard-32 (32 vCPU, 128 Go de mémoire)

Quotas par région : Limite du nombre de CPUs, Il faut faire une demande pour l’augmenter.

Résultats et Utilisation de GCP

Type de machine	n2-standard-32 (32 vCPU, 128 Go de mémoire)

Quotas par région : Limite du nombre de CPUs, Il faut faire une demande pour l’augmenter.

```
Computing initial states...
Finished computing initial states: 1 distinct state generated at 2026-01-24 13:39:34.
Progress(11) at 2026-01-24 13:39:37: 1,060,129 states generated (1,060,129 s/min), 344,944 distinct states found (344,944 ds/min), 184,742 states left on queue.
Progress(18) at 2026-01-24 13:40:37: 32,868,880 states generated (31,808,751 s/min), 10,134,026 distinct states found (9,789,082 ds/min), 2,897,521 states left on queue.
Progress(21) at 2026-01-24 13:41:37: 61,780,948 states generated (28,912,068 s/min), 19,183,641 distinct states found (9,049,615 ds/min), 3,877,130 states left on queue.
Progress(23) at 2026-01-24 13:42:37: 86,685,041 states generated (24,904,093 s/min), 27,305,580 distinct states found (8,121,939 ds/min), 3,821,564 states left on queue.
Progress(25) at 2026-01-24 13:43:37: 110,712,805 states generated (24,027,764 s/min), 35,639,572 distinct states found (8,333,992 ds/min), 2,594,690 states left on queue.
Model checking completed. No error has been found.
```

x10!

Résultats et Utilisation de GCP

```
Error: when writing the disk (StatePoolWriter.run):
No space left on device
```

Le model checker stocke chaque état distinct qu'il trouve.
Initialement, le disk de la machine est que 10GB!

Résultats et Utilisation de GCP

```
Error: when writing the disk (StatePoolWriter.run):
No space left on device
```

Le model checker stocke chaque état distinct qu'il trouve.
Initialement, le disk de la machine est que 10GB!

GCP permet de rajouter un disque : encore une fois, quota de 500GB

Il faut partitionner, formater, et monter le disque.

Résultats et Utilisation de GCP

```
Progress(16) at 2026-01-28 10:58:26: 5,723,764,681 states generated (40,918,393 s/min), 1,474,354,894 distinct states found (10,180,267 ds/min), 942,939,560 states left on queue.
Progress(16) at 2026-01-28 10:59:26: 5,768,779,945 states generated (45,015,264 s/min), 1,485,478,641 distinct states found (11,123,747 ds/min), 950,085,635 states left on queue.
Progress(16) at 2026-01-28 11:00:26: 5,812,889,435 states generated (44,109,490 s/min), 1,496,184,271 distinct states found (10,705,630 ds/min), 956,955,608 states left on queue.
Error: when writing the disk (StatePoolWriter.run):
No space left on device
```

Après seulement 2h20 et 1 milliard et demi d’états distincts, on rempli le disque de 500GB

Apports et limites de l’IA

Points positifs

- Gain de temps pour démarrer.
- Aide à la structuration initiale.

Limites

- L’IA a du mal avec un langages de niche comme TLA+.
- Elle invente beaucoup de choses.
- Elle fait des erreurs.

Quoi faire pour utiliser l’IA sur un sujet niche comme TLA+

- Utiliser plutôt moins souvent que plus souvent.
- Partir du principe que tout ce qui est généré aura des erreurs.
- Agir comme si l’IA essaye de te berner en te donnant le code qui paraît le plus safe possible en ayant quand même des erreurs.
- Ne pas s’acharner à essayer de faire régler un problème à l’IA où elle a déjà échoué une ou deux fois.