

Ré-identification sans coordination dans les types de données répliquées sans conflits

Matthieu Nicolas (matthieu.nicolas@loria.fr)

20 décembre 2022

<i>Rapporteurs :</i>	Hanifa Boucheneb Davide Frey	Professeure, Polytechnique Montréal Chargé de recherche, HdR, Inria Rennes Bretagne-Atlantique
<i>Examineurs :</i>	Hala Skaf-Molli Stephan Merz	Maîtresse de conférences, HdR, Nantes Université, LS2N Directeur de Recherche, Inria Nancy - Grand Est
<i>Encadrants :</i>	Olivier Perrin Gérald Oster	Professeur des Universités, Université de Lorraine, LORIA Maître de conférences, Université de Lorraine, LORIA

Conflict-free Replicated Data Types (CRDTs)^[1]

- Nouvelles spécifications des types de données, e.g. *Ensemble* ou *Séquence*
- Conçues pour données répliquées

[1]. SHAPIRO et al., « Conflict-Free Replicated Data Types ».

Conflict-free Replicated Data Types (CRDTs)^[1]

- Nouvelles spécifications des types de données, e.g. *Ensemble* ou *Séquence*
- Conçues pour données répliquées

Propriétés des CRDTs

- Permettent modifications **sans coordination**
- Garantissent la **convergence forte**

[1]. SHAPIRO et al., « Conflict-Free Replicated Data Types ».

Conflict-free Replicated Data Types (CRDTs)^[1]

- Nouvelles spécifications des types de données, e.g. *Ensemble* ou *Séquence*
- Conçues pour données répliquées

Propriétés des CRDTs

- Permettent modifications **sans coordination**
- Garantissent la **convergence forte**

Convergence forte

Ensemble des noeuds ayant intégrés le même ensemble de modifications obtient des états équivalents, sans nécessiter d'actions ou messages supplémentaires

[1]. SHAPIRO et al., « Conflict-Free Replicated Data Types ».

LogootSplit^[3], un CRDT pour le type Séquence

- Assigne **identifiant de position** à chaque élément de la séquence

[2]. PREGUICA et al., « A Commutative Replicated Data Type for Cooperative Editing ».

[3]. ANDRÉ et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

LogootSplit^[3], un CRDT pour le type Séquence

- Assigne **identifiant de position** à chaque élément de la séquence

Propriétés des identifiants de position^[2]

1. Unique
2. Immuable
3. Ordonnable par une relation d'ordre strict total $<_{id}$
4. Appartenant à un espace dense

[2]. PREGUICA et al., « A Commutative Replicated Data Type for Cooperative Editing ».

[3]. ANDRÉ et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

LogootSplit^[3], un CRDT pour le type Séquence

- Assigne **identifiant de position** à chaque élément de la séquence

Propriétés des identifiants de position^[2]

1. Unique
2. Immuable
3. Ordonnable par une relation d'ordre strict total $<_{id}$
4. Appartenant à un espace dense

- **Ordonne les éléments** entre eux **en utilisant** leurs **identifiants**

[2]. PREGUICA et al., « A Commutative Replicated Data Type for Cooperative Editing ».

[3]. ANDRÉ et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

Identifiant

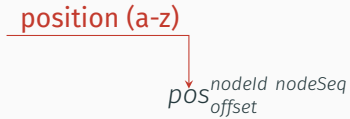
- Composé d'un ou plusieurs tuples suivants

$pos_{offset}^{nodeId \ nodeSeq}$

Identifiant LogootSplit

Identifiant

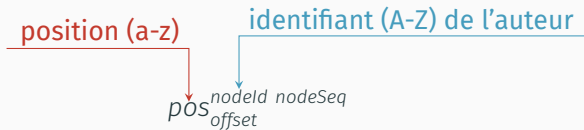
- Composé d'un ou plusieurs tuples suivants



Identifiant LogootSplit

Identifiant

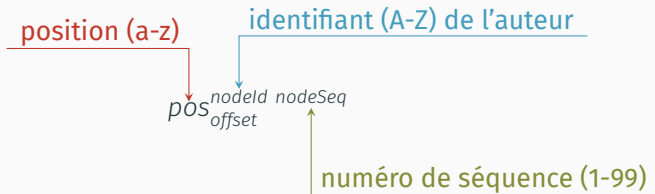
- Composé d'un ou plusieurs tuples suivants



Identifiant LogootSplit

Identifiant

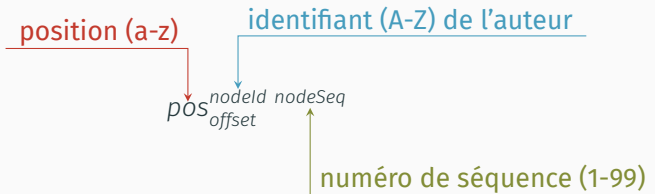
- Composé d'un ou plusieurs tuples suivants



Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples suivants



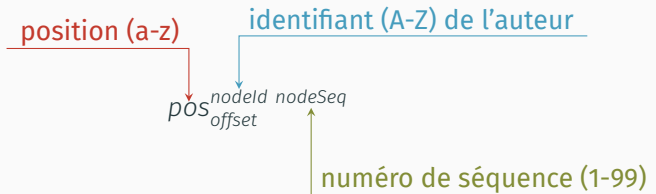
Exemples

d_0^{F5}

Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples suivants



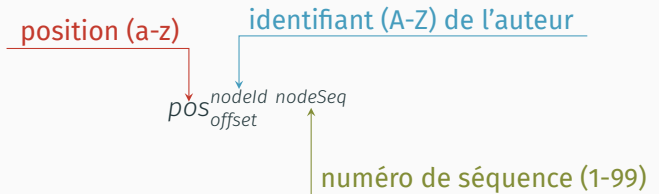
Exemples

$$d_0^{F5} <_{id} m_0^{C1}$$

Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples suivants



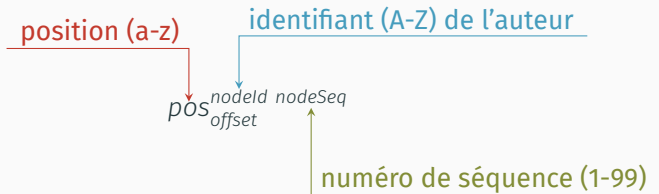
Exemples

$$d_0^{F5} <_{id} m_0^{C1} <_{id} m_0^{C1} f_0^{E1}$$

Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples suivants



Exemples

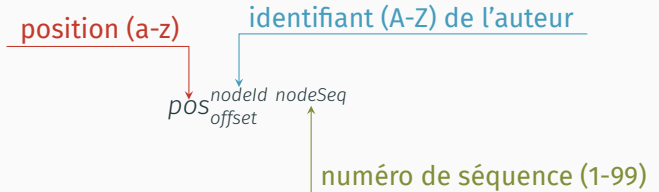
$$d_0^{F5} <_{id} m_0^{C1} <_{id} m_0^{C1} f_0^{E1}$$

$$i_0^{B1} <_{id} ? <_{id} i_1^{B1}$$

Identifiant LogootSplit

Identifiant

- Composé d'un ou plusieurs tuples suivants



Exemples

$$d_0^{F5} <_{id} m_0^{C1} <_{id} m_0^{C1} f_0^{E1}$$

$$i_0^{B1} <_{id} i_0^{B1} f_0^{A1} <_{id} i_1^{B1}$$

- Coûteux de stocker les identifiants de chaque élément

B	A	N	J	O
m_0^{C1}	m_1^{C1}	m_2^{C1}	m_3^{C1}	m_4^{C1}

Bloc LogootSplit

- Coûteux de stocker les identifiants de chaque élément

B	A	N	J	O
m_0^{C1}	m_1^{C1}	m_2^{C1}	m_3^{C1}	m_4^{C1}

- Aggrège en un **bloc** éléments ayant **identifiants contigus**

Identifiants contigus

Deux identifiants sont contigus si et seulement si les deux identifiants sont identiques à l'exception de leur dernier offset et que leur derniers offsets sont consécutifs.

Bloc LogootSplit

- Coûteux de stocker les identifiants de chaque élément

B	A	N	J	O
m_0^{C1}	m_1^{C1}	m_2^{C1}	m_3^{C1}	m_4^{C1}

- Aggrège en un **bloc** éléments ayant **identifiants contigus**

Identifiants contigus

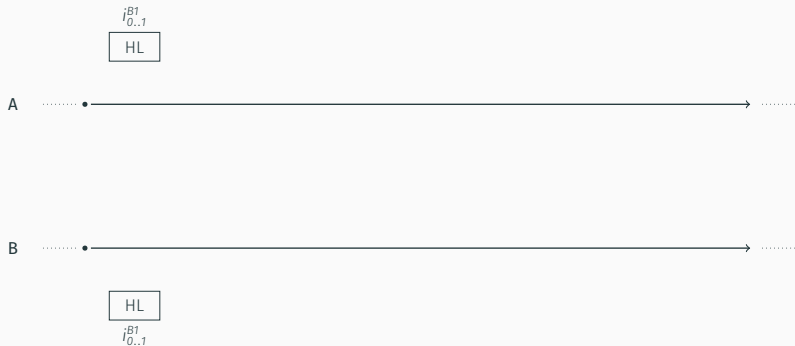
Deux identifiants sont contigus si et seulement si les deux identifiants sont identiques à l'exception de leur dernier offset et que leur derniers offsets sont consécutifs.

- Note l'intervalle d'identifiants d'un bloc : $pos_{begin..end}^{nodeId nodeSeq}$

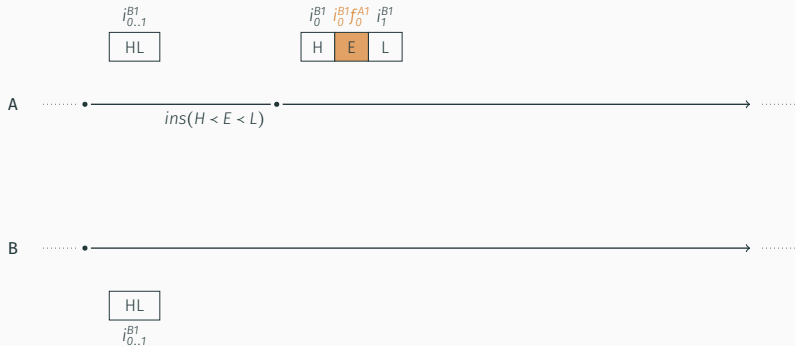
BANJO

$m_{0..4}^{C1}$

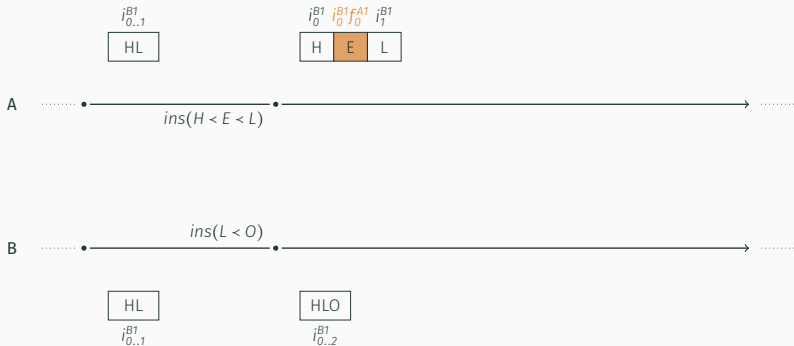
Exemple insertions concurrentes



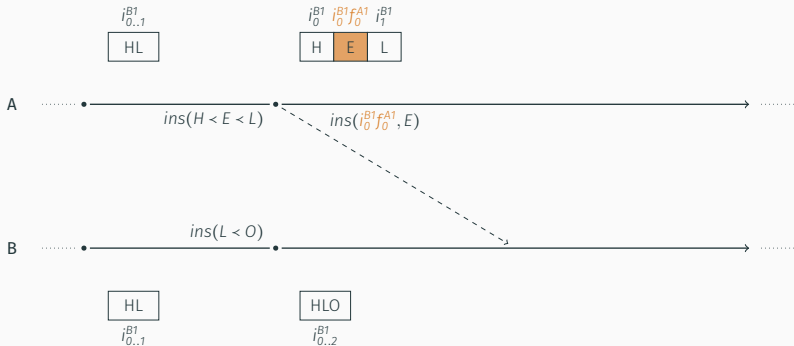
Exemple insertions concurrentes



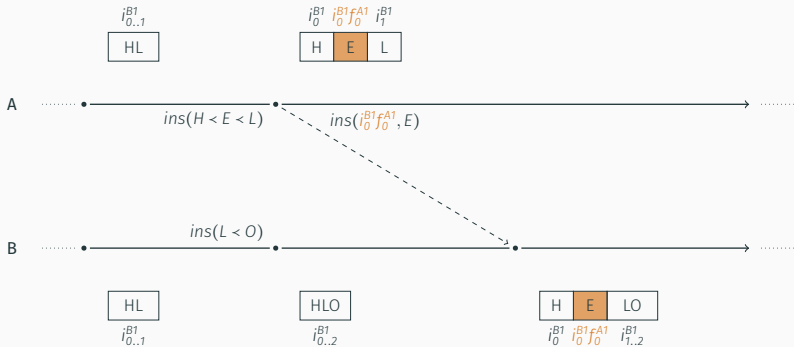
Exemple insertions concurrentes



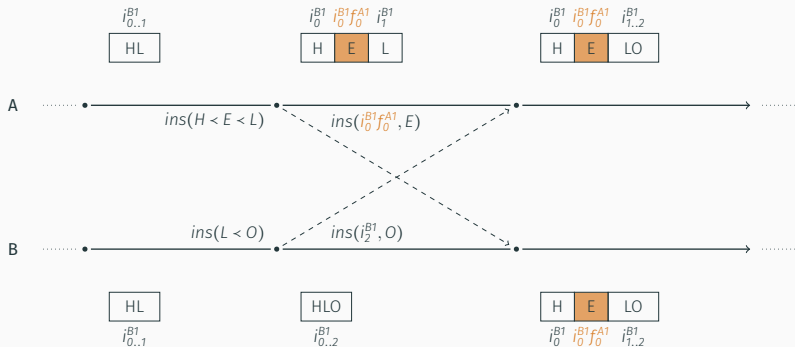
Exemple insertions concurrentes



Exemple insertions concurrentes



Exemple insertions concurrentes



Limites de LogootSplit

Sources croissance métadonnées

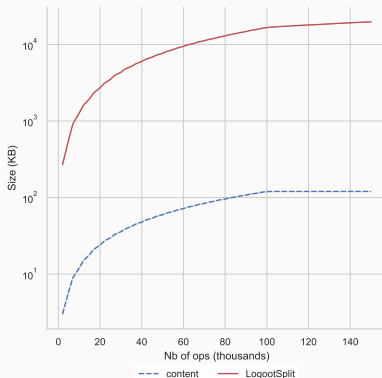
- Croissance non-bornée de la taille des identifiants
- Fragmentation en blocs courts

Limites de LogootSplit

Sources croissance métadonnées

- Croissance non-bornée de la taille des identifiants
- Fragmentation en blocs courts

Taille du contenu comparé à la taille de la séquence LogootSplit



L'approche core-nebula^[4]

- Ré-assigne des identifiants courts aux éléments, c.-à-d. les *renomme*
- Transforme les opérations *insert* et *remove* concurrentes...

[4]. ZAWIRSKI et al., « Asynchronous rebalancing of a replicated tree ».

L'approche core-nebula^[4]

- Ré-assigne des identifiants courts aux éléments, c.-à-d. les *renomme*
- Transforme les opérations *insert* et *remove* concurrentes...
- ...mais ne supportent pas opérations *rename* concurrentes

[4]. ZAWIRSKI et al., « Asynchronous rebalancing of a replicated tree ».

Mitigation du surcoût des CRDTs pour le type Séquence

L'approche core-nebula^[4]

- Ré-assigne des identifiants courts aux éléments, c.-à-d. les *renomme*
- Transforme les opérations *insert* et *remove* concurrentes...
- ...mais ne supportent pas opérations *rename* concurrentes

Inadaptée aux applications pair-à-pair

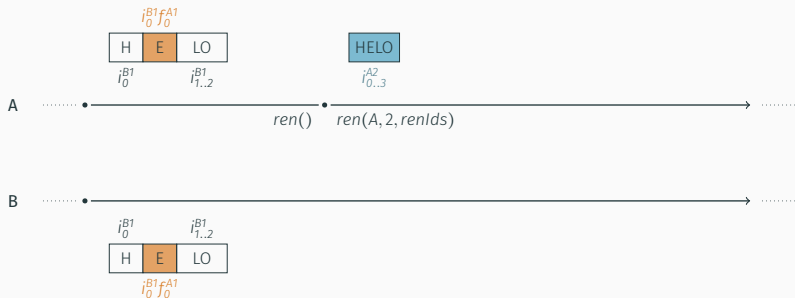
[4]. ZAWIRSKI et al., « Asynchronous rebalancing of a replicated tree ».

Proposition

Mécanisme de renommage supportant les
renommages concurrents

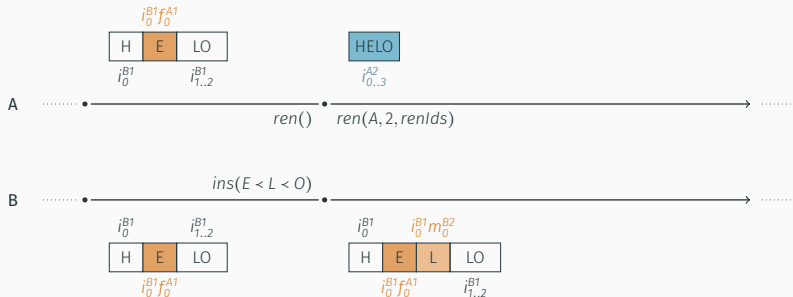
RenamableLogootSplit

Interactions avec opérations *insert* et *remove* concurrentes



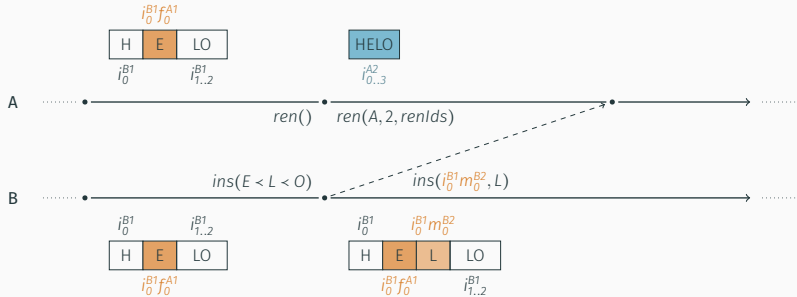
- Peuvent générer opérations concurrentes aux opérations *rename*

Interactions avec opérations *insert* et *remove* concurrentes



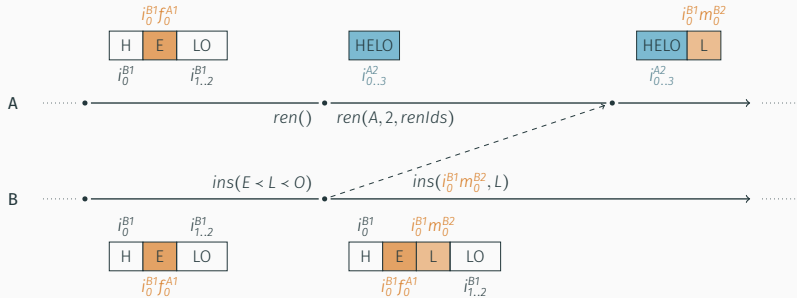
- Peuvent générer opérations concurrentes aux opérations *rename*

Interactions avec opérations *insert* et *remove* concurrentes



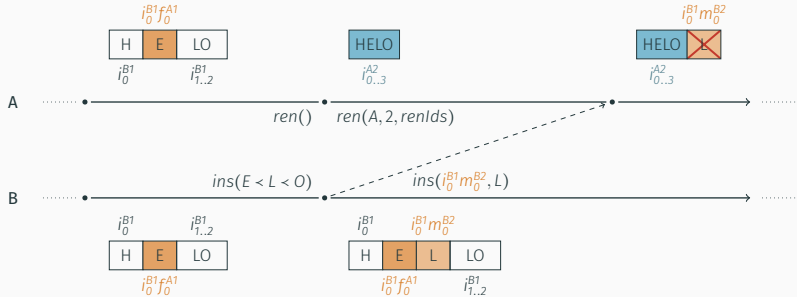
- Peuvent générer opérations concurrentes aux opérations *rename*

Interactions avec opérations *insert* et *remove* concurrentes



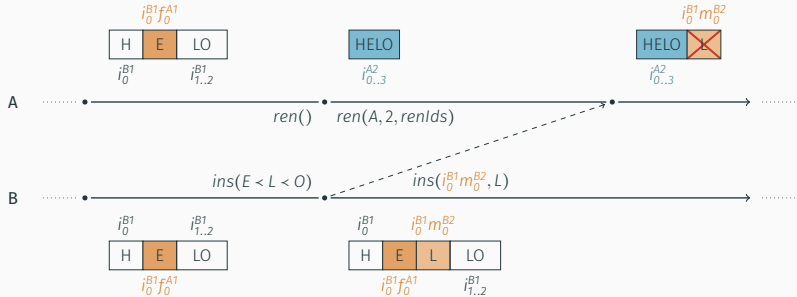
- Peuvent générer opérations concurrentes aux opérations *rename*

Interactions avec opérations *insert* et *remove* concurrentes



- Peuvent générer opérations concurrentes aux opérations *rename*
- Produisent anomalies si intégrées naïvement

Interactions avec opérations *insert* et *remove* concurrentes



- Peuvent générer opérations concurrentes aux opérations *rename*
- Produisent anomalies si intégrées naïvement

Nécessité d'un mécanisme dédié

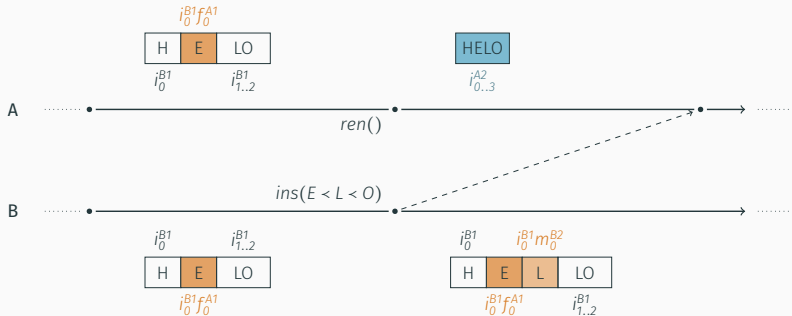
Besoins

1. Détecter les opérations concurrentes aux opérations *rename*
2. Prendre en compte effet des opérations *rename* lors de l'intégration des opérations concurrentes

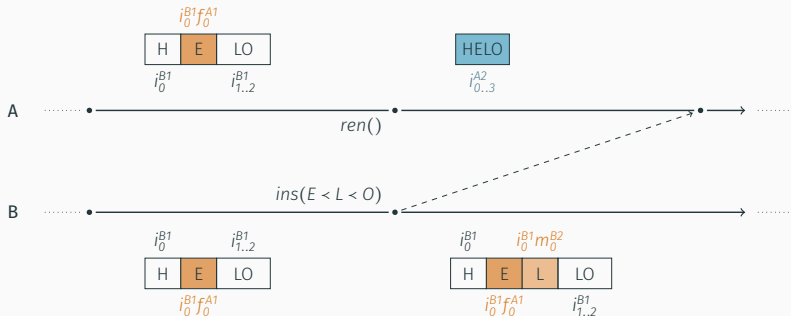
Besoins

1. Détecter les opérations concurrentes aux opérations *rename*
2. Prendre en compte effet des opérations *rename* lors de l'intégration des opérations concurrentes

Détection des opérations concurrentes à opération *rename*

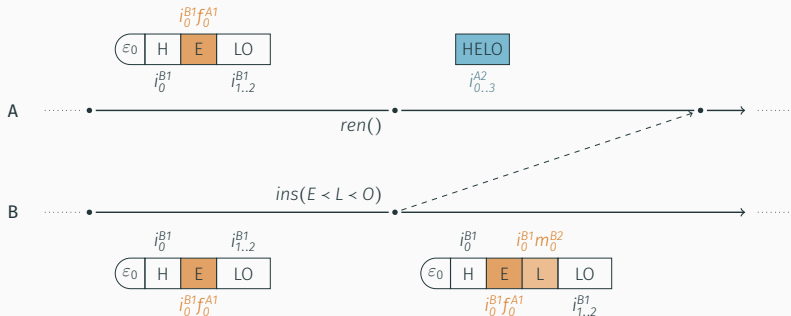


Détection des opérations concurrentes à opération *rename*



Ajout mécanisme d'époques

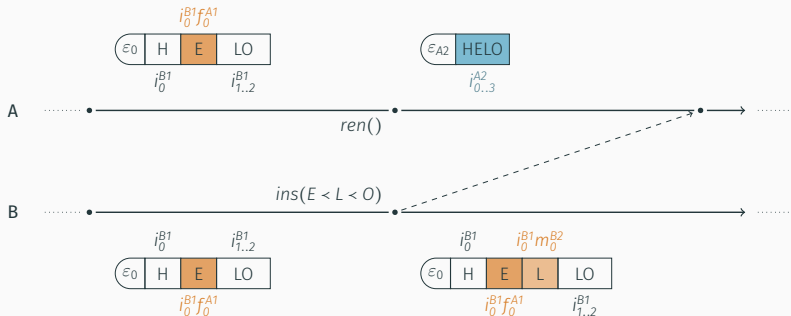
Détection des opérations concurrentes à opération *rename*



Ajout mécanisme d'époques

- Séquence commence à époque d'origine, notée ϵ_0

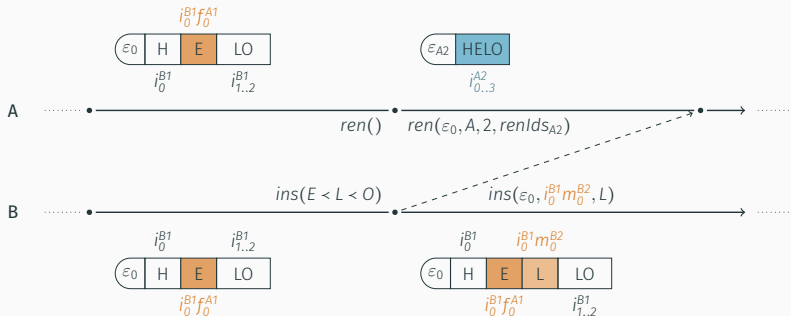
Détection des opérations concurrentes à opération *rename*



Ajout mécanisme d'époques

- Séquence commence à époque d'origine, notée ϵ_0
- *rename* font progresser à nouvelle époque, $\epsilon_{nodeId \ nodeSeq}$

Détection des opérations concurrentes à opération *rename*



Ajout mécanisme d'époques

- Séquence commence à époque d'origine, notée ϵ_0
- *rename* font progresser à nouvelle époque, $\epsilon_{nodeId \ nodeSeq}$
- Opérations labellisées avec époque de génération

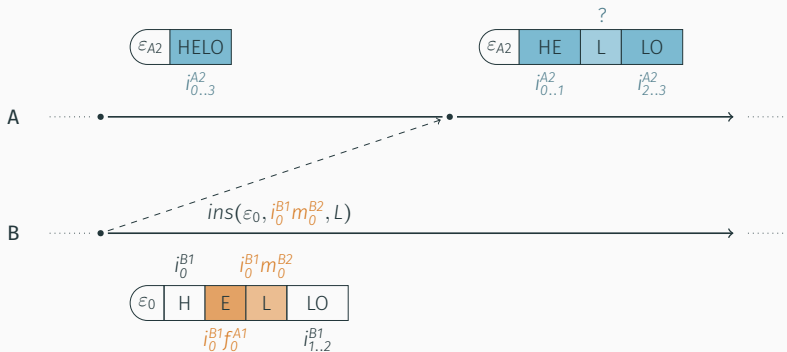
Besoins

1. Détecter les opérations concurrentes aux opérations *rename*
2. Prendre en compte effet des opérations *rename* lors de l'intégration des opérations concurrentes

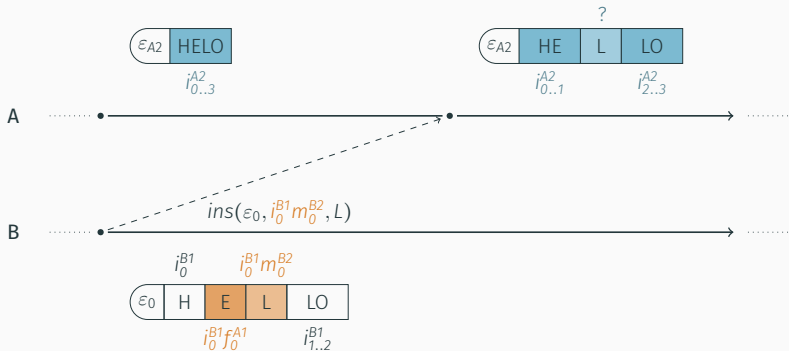
Besoins

1. Détecter les opérations concurrentes aux opérations *rename*
2. Prendre en compte effet des opérations *rename* lors de l'intégration des opérations concurrentes

Intégration des opérations *insert* et *remove* concurrentes

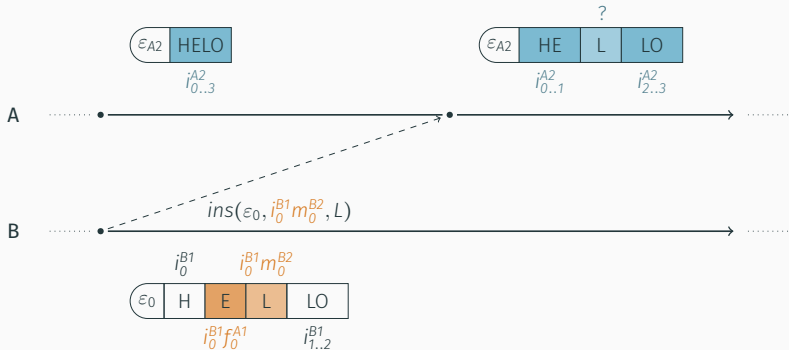


Intégration des opérations *insert* et *remove* concurrentes



Ajout d'un mécanisme de transformation des opérations *insert* et *remove* concurrentes

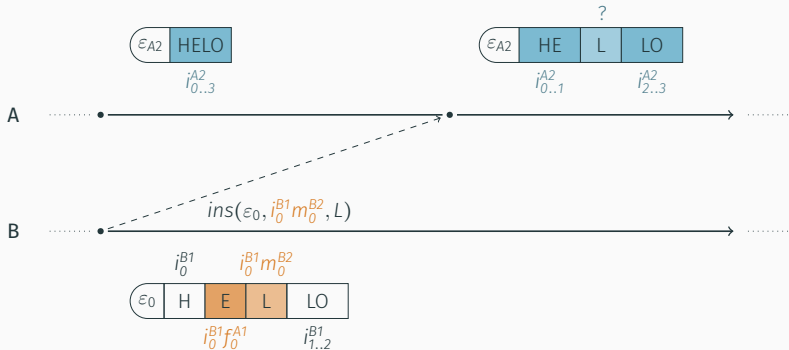
Intégration des opérations *insert* et *remove* concurrentes



Ajout d'un mécanisme de transformation des opérations *insert* et *remove* concurrentes

- Prend la forme de l'algorithme `renameId`

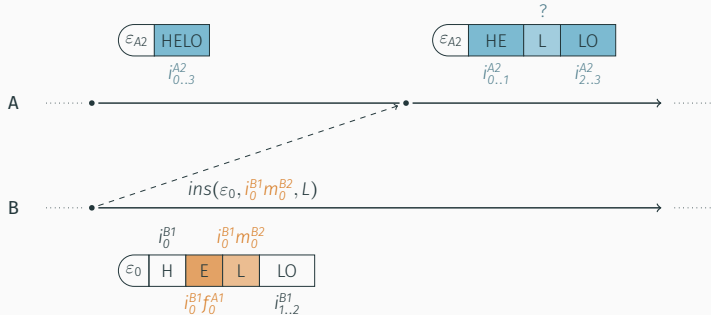
Intégration des opérations *insert* et *remove* concurrentes



Ajout d'un mécanisme de transformation des opérations *insert* et *remove* concurrentes

- Prend la forme de l'algorithme `renameId`
- Inclure l'effet de l'opération *rename* dans l'opération transformée

Exemple de renameId

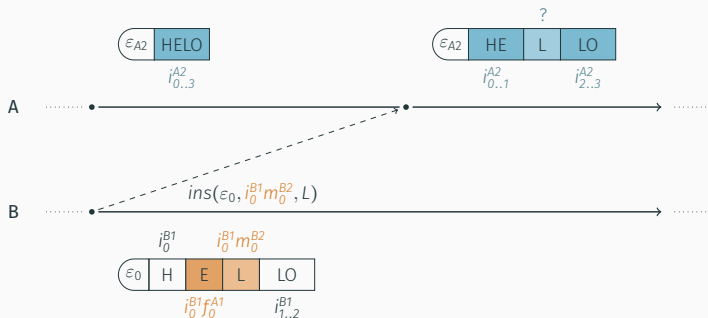


Rappel :

$$renIds_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_0^{B1} m_0^{B2}$

Exemple de renameId



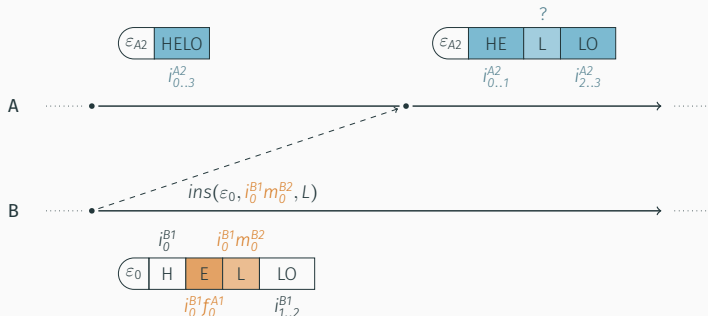
Rappel :

$$renIds_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_0^{B1} m_0^{B2}$

- Trouver son prédécesseur dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$

Exemple de renameId



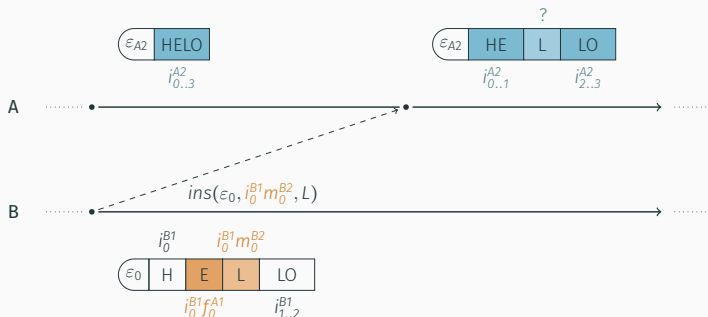
Rappel :

$$renIds_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_0^{B1} m_0^{B2}$

- Trouver son prédécesseur dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$
- Trouver son équivalent à l'époque cible ϵ_{A2} : i_1^{A2}

Exemple de renameId



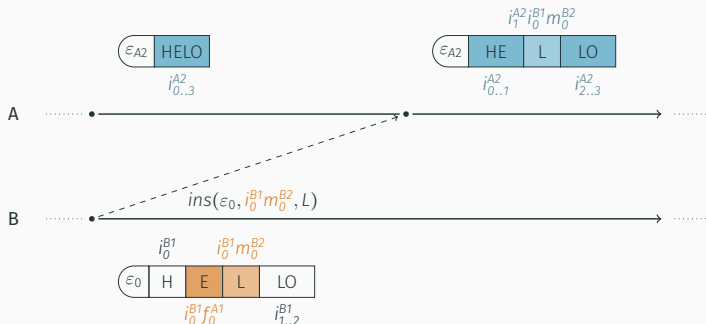
Rappel :

$$renIds_{A2} = [i_0^{B1}, i_0^{B1}f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_0^{B1}m_0^{B2}$

- Trouver son prédécesseur dans $renIds_{A2}$: $i_0^{B1}f_0^{A1}$
- Trouver son équivalent à l'époque cible ϵ_{A2} : i_1^{A2}
- Préfixer $i_0^{B1}m_0^{B2}$ par ce dernier : $i_1^{A2}i_0^{B1}m_0^{B2}$

Exemple de renameId



Rappel :

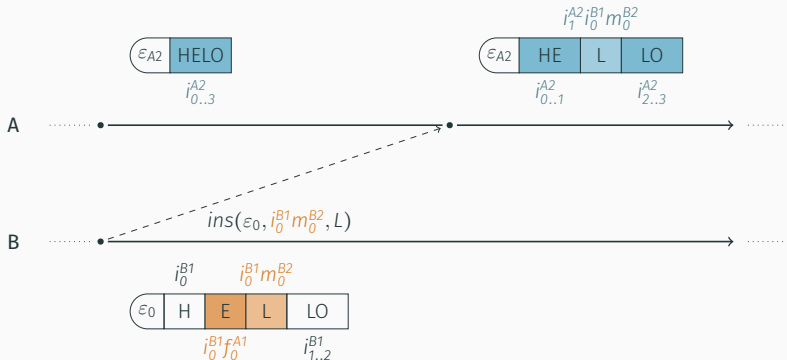
$$renIds_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_0^{B1} m_0^{B2}$

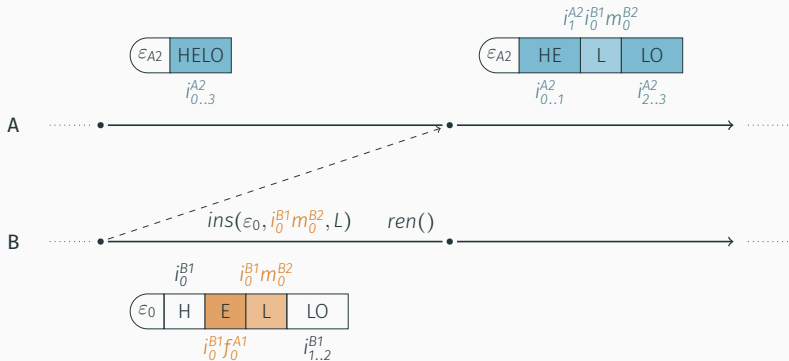
- Trouver son prédécesseur dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$
- Trouver son équivalent à l'époque cible ϵ_{A2} : i_1^{A2}
- Préfixer $i_0^{B1} m_0^{B2}$ par ce dernier : $i_1^{A2} i_0^{B1} m_0^{B2}$

Et en cas d'opérations *rename* concurrentes?

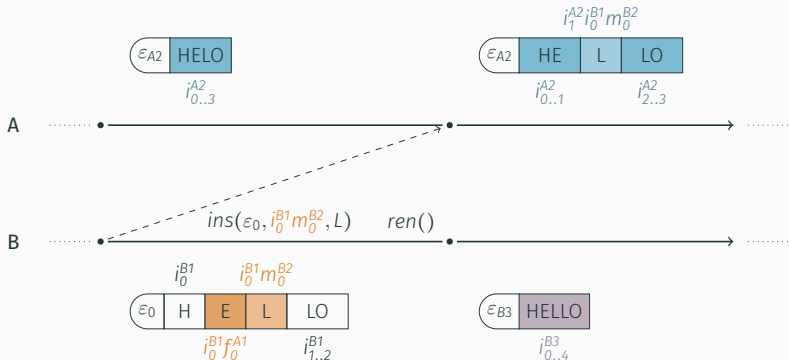
Opérations *rename* concurrentes



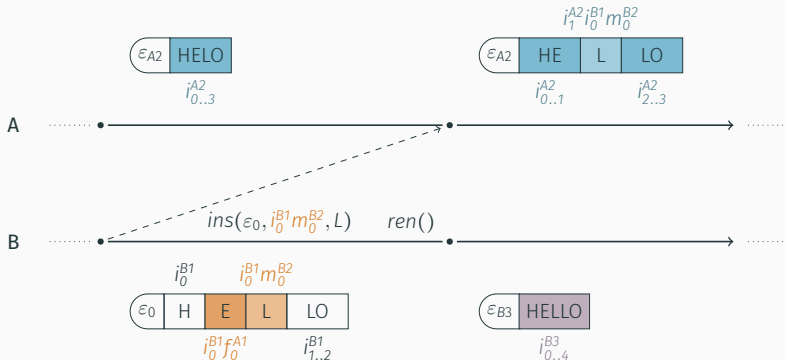
Opérations *rename* concurrentes



Opérations *rename* concurrentes

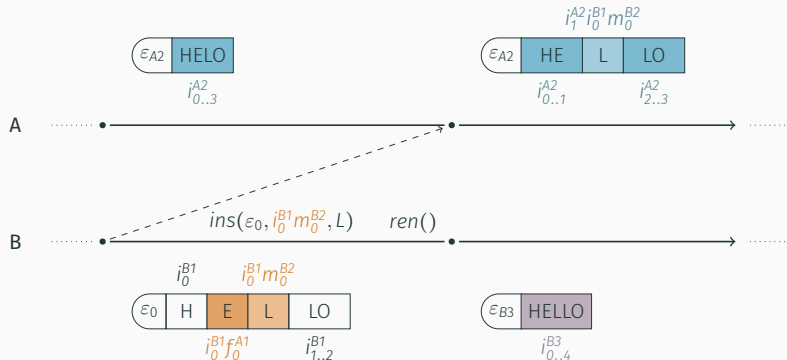


Opérations *rename* concurrentes



Comment faire converger les noeuds ?

Opérations *rename* concurrentes



Comment faire converger les noeuds ?

Besoin d'un mécanisme additionnel de résolution de conflits

Observation

- Opérations *rename* sont des opérations systèmes...
- ...pas des opérations utilisateur-rices

Observation

- Opérations *rename* sont des opérations systèmes...
- ...pas des opérations utilisateur-rices

Proposition

- Considérer une opération *rename* comme prioritaire...
- ...et ignorer les opérations *rename* en conflit avec elle

Algorithme d'intégration d'une opération *rename*

Intuition

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque **l'époque cible**

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque **l'époque cible**
3. Si changement d'époque cible

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque **l'époque cible**
3. Si changement d'époque cible
 - 3.1 Calculer chemin entre époque courante et époque cible, et notamment leur Plus Proche Ancêtre Commun (PPAC)

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque **l'époque cible**
3. Si changement d'époque cible
 - 3.1 Calculer chemin entre époque courante et époque cible, et notamment leur Plus Proche Ancêtre Commun (PPAC)
 - 3.2 Annuler l'effet des opérations *rename* de l'époque courante au PPAC

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque l'époque cible
3. Si changement d'époque cible
 - 3.1 Calculer chemin entre époque courante et époque cible, et notamment leur Plus Proche Ancêtre Commun (PPAC)
 - 3.2 Annuler l'effet des opérations *rename* de l'époque courante au PPAC
 - 3.3 Appliquer l'effet des opérations *rename* du PPAC à l'époque cible

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque **l'époque cible**
3. Si changement d'époque cible
 - 3.1 Calculer chemin entre époque courante et époque cible, et notamment leur Plus Proche Ancêtre Commun (PPAC)
 - 3.2 Annuler l'effet des opérations *rename* de l'époque courante au PPAC
 - 3.3 Appliquer l'effet des opérations *rename* du PPAC à l'époque cible

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque l'époque cible
3. Si changement d'époque cible
 - 3.1 Calculer chemin entre époque courante et époque cible, et notamment leur Plus Proche Ancêtre Commun (PPAC)
 - 3.2 Annuler l'effet des opérations *rename* de l'époque courante au PPAC
 - 3.3 Appliquer l'effet des opérations *rename* du PPAC à l'époque cible

Choisir une époque comme époque cible

A →

B →

C →

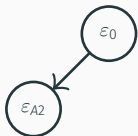
Arbre des époques



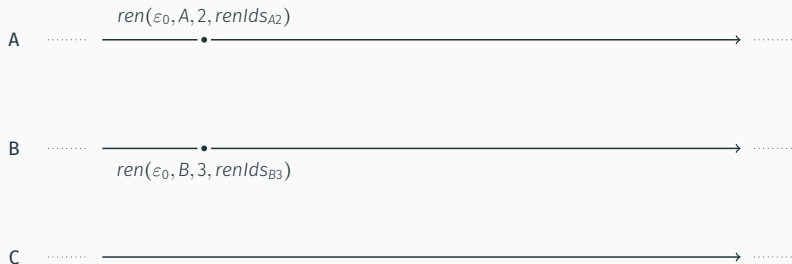
Choisir une époque comme époque cible



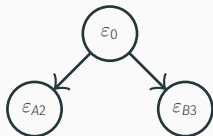
Arbre des époques



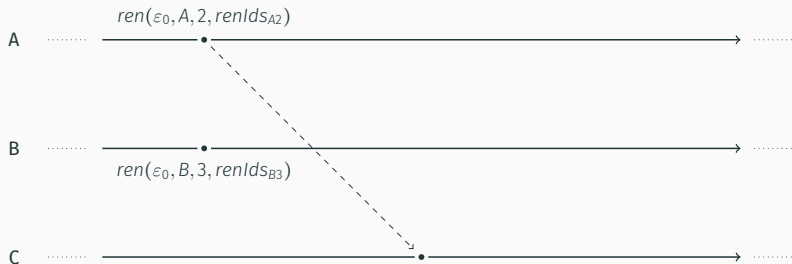
Choisir une époque comme époque cible



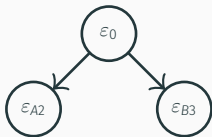
Arbre des époques



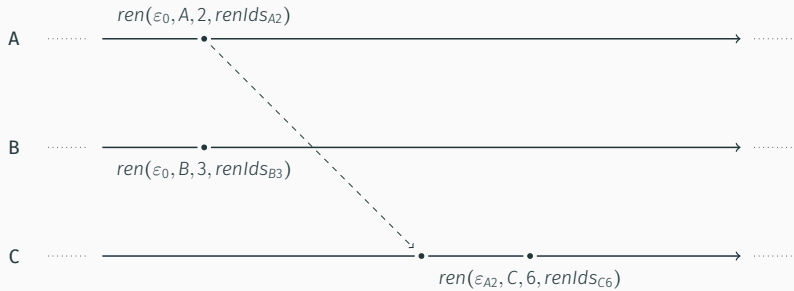
Choisir une époque comme époque cible



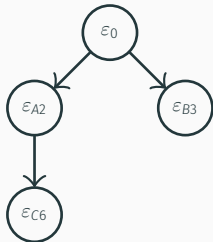
Arbre des époques



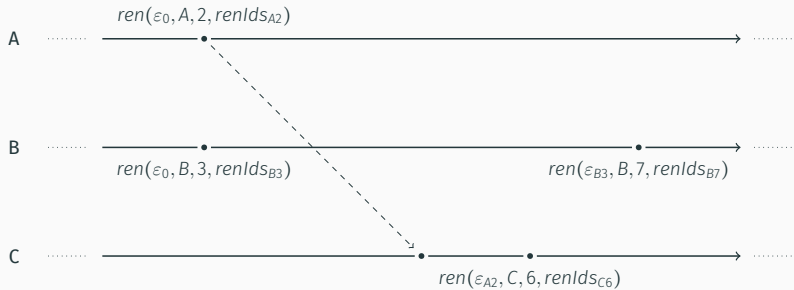
Choisir une époque comme époque cible



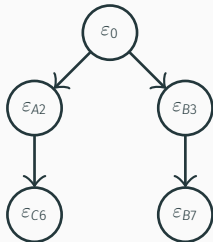
Arbre des époques



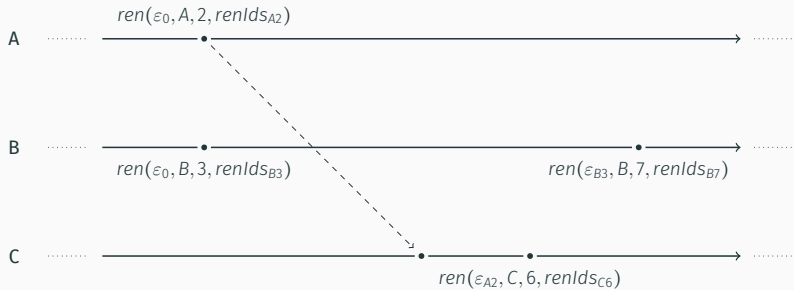
Choisir une époque comme époque cible



Arbre des époques

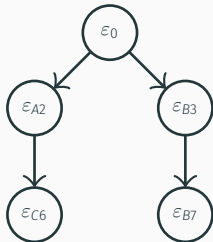


Choisir une époque comme époque cible

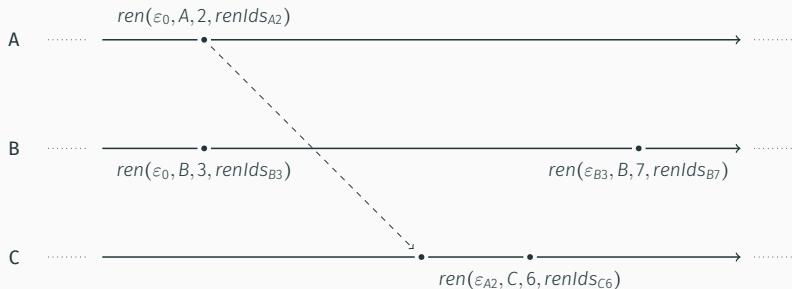


Arbre des époques

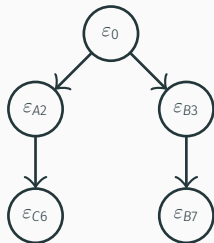
Comment choisir?



Choisir une époque comme époque cible



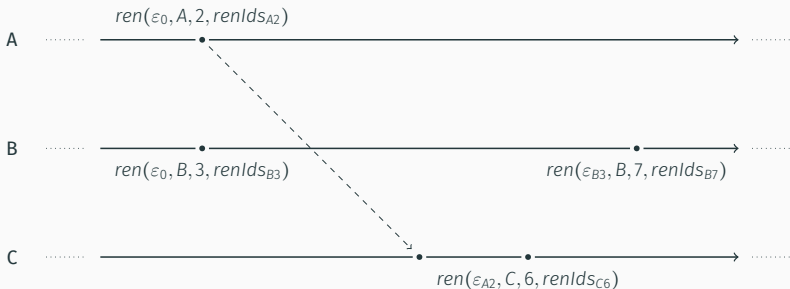
Arbre des époques



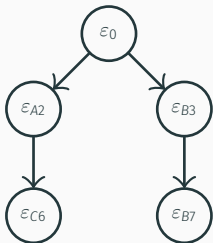
Comment choisir?

- Définit relation *priority*, notée $<_{\varepsilon}$, ordre strict total sur les époques

Choisir une époque comme époque cible



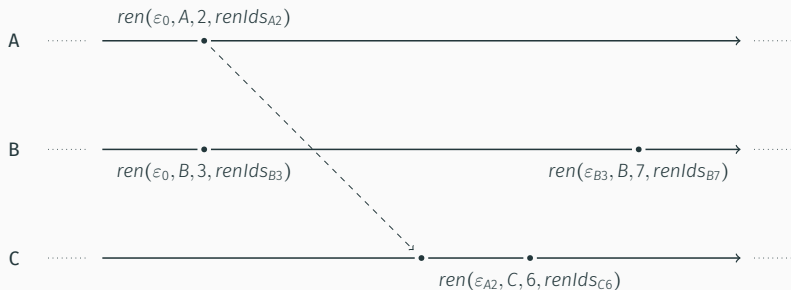
Arbre des époques



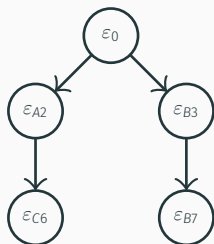
Comment choisir?

- Définit relation *priority*, notée $<_{\varepsilon}$, ordre strict total sur les époques
- Utilise **ordre lexicographique sur chemins** des époques dans l'arbre

Choisir une époque comme époque cible



Arbre des époques



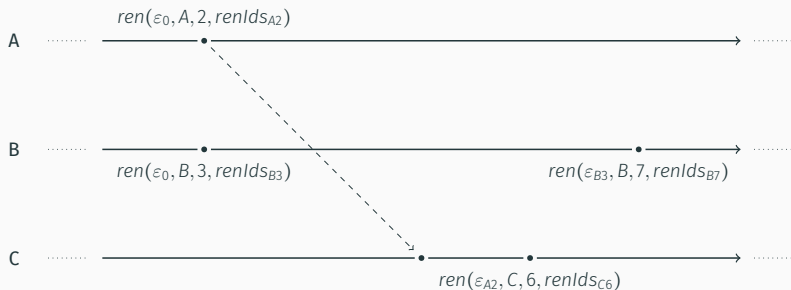
Comment choisir?

- Définit relation *priority*, notée $<_{\epsilon}$, ordre strict total sur les époques
- Utilise **ordre lexicographique sur chemins** des époques dans l'arbre

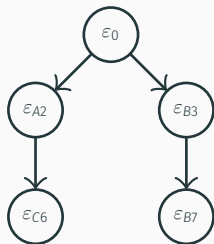
Exemple

$$\epsilon_0 < \epsilon_0 \epsilon_{A2}$$

Choisir une époque comme époque cible



Arbre des époques



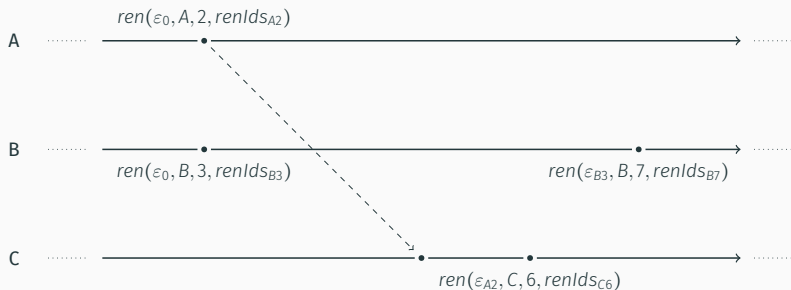
Comment choisir?

- Définit relation *priority*, notée $<_{\epsilon}$, ordre strict total sur les époques
- Utilise **ordre lexicographique sur chemins** des époques dans l'arbre

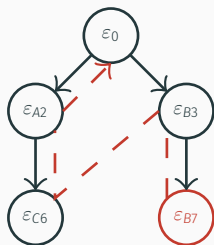
Exemple

$$\epsilon_0 < \epsilon_0 \epsilon_{A2} < \epsilon_0 \epsilon_{A2} \epsilon_{C6}$$

Choisir une époque comme époque cible



Arbre des époques



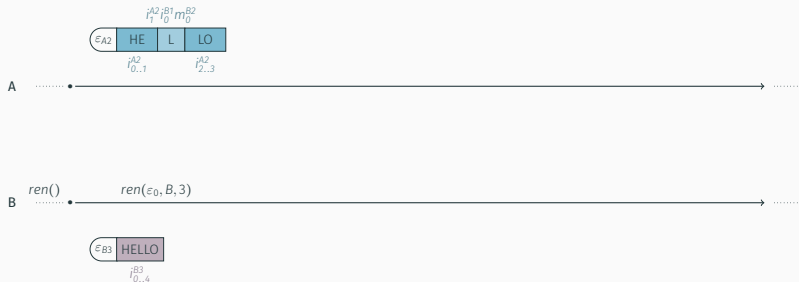
Comment choisir?

- Définit relation *priority*, notée $<_{\varepsilon}$, ordre strict total sur les époques
- Utilise **ordre lexicographique sur chemins** des époques dans l'arbre

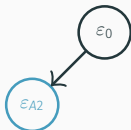
Exemple

$$\varepsilon_0 < \varepsilon_0 \varepsilon_{A2} < \varepsilon_0 \varepsilon_{A2} \varepsilon_{C6} < \varepsilon_0 \varepsilon_{B3} \varepsilon_{B7}$$

Exemple - Calculs des transformations à effectuer



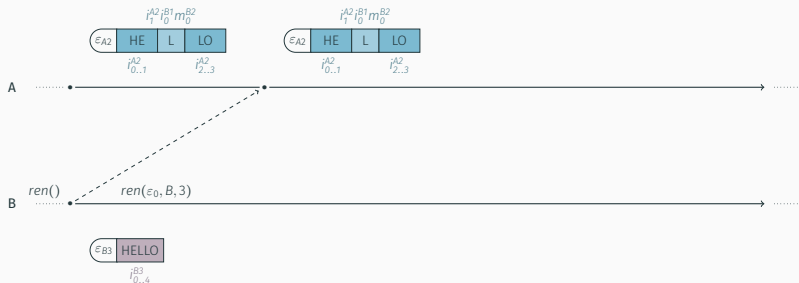
Arbre des époques de A



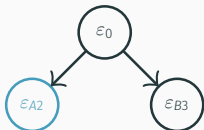
Étapes

- Époque courante : ϵ_{A2}

Exemple - Calculs des transformations à effectuer



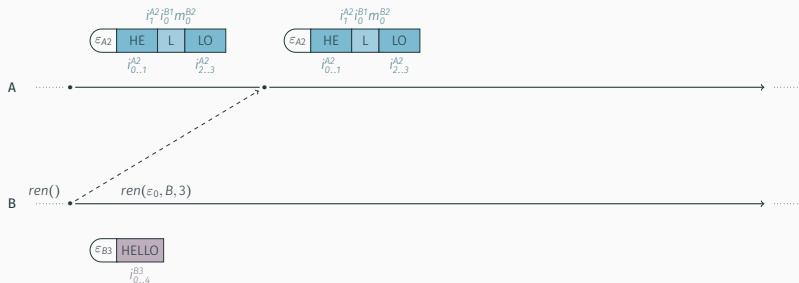
Arbre des époques de A



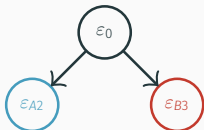
Étapes

- Époque courante : ϵ_{A2}

Exemple - Calculs des transformations à effectuer



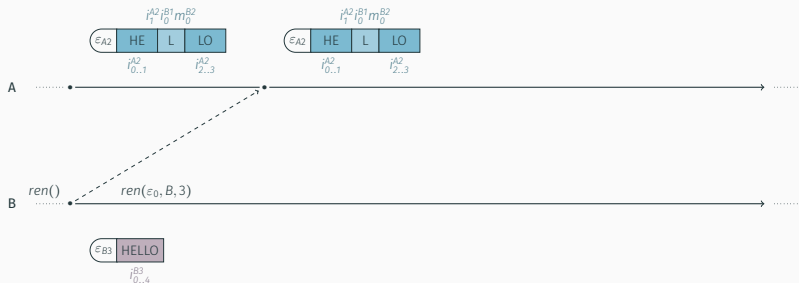
Arbre des époques de A



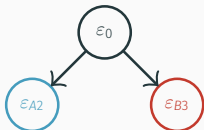
Étapes

- Époque courante : ϵ_{A2}
- Époque cible : ϵ_{B3}

Exemple - Calculs des transformations à effectuer



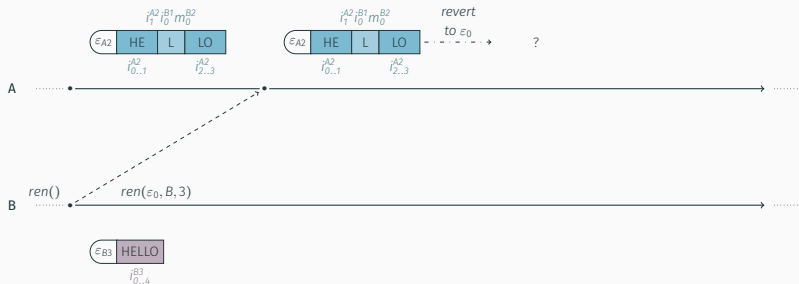
Arbre des époques de A



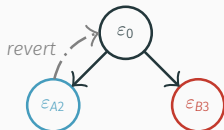
Étapes

- Époque courante : ε_{A2}
- Époque cible : ε_{B3}
- Plus Proche Ancêtre Commun : ε_0

Exemple - Calculs des transformations à effectuer



Arbre des époques de A

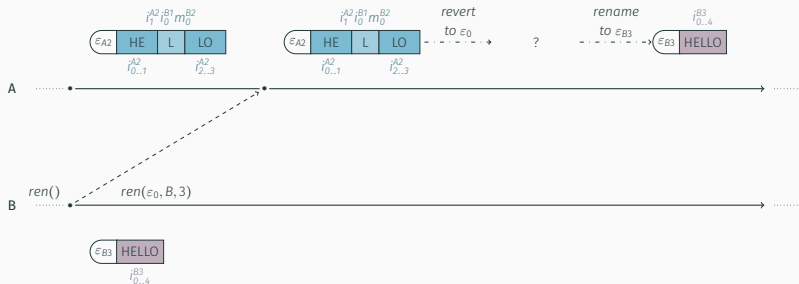


Étapes

- Époque courante : ε_{A2}
- Époque cible : ε_{B3}
- Plus Proche Ancêtre Commun : ε_0

Doit annuler ε_{A2}

Exemple - Calculs des transformations à effectuer



Arbre des époques de A



Étapes

- Époque courante : ϵ_{A2}
- Époque cible : ϵ_{B3}
- Plus Proche Ancêtre Commun : ϵ_0

Doit annuler ϵ_{A2} puis appliquer ϵ_{B3}

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque l'époque cible
3. Si changement d'époque cible
 - 3.1 Calculer chemin entre époque courante et époque cible, et notamment leur Plus Proche Ancêtre Commun (PPAC)
 - 3.2 Annuler l'effet des opérations *rename* de l'époque courante au PPAC
 - 3.3 Appliquer l'effet des opérations *rename* du PPAC à l'époque cible

Algorithme d'intégration d'une opération *rename*

Intuition

1. Ajouter l'époque créée par l'opération *rename* à l'ensemble des époques connues
2. Choisir entre époque courante et nouvelle époque l'époque cible
3. Si changement d'époque cible
 - 3.1 Calculer chemin entre époque courante et époque cible, et notamment leur Plus Proche Ancêtre Commun (PPAC)
 - 3.2 Annuler l'effet des opérations *rename* de l'époque courante au PPAC
 - 3.3 Appliquer l'effet des opérations *rename* du PPAC à l'époque cible

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenameId*

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenameId*
- Exclure l'effet de l'opération *rename*

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenamed*
- Exclure l'effet de l'opération *rename*

Intuition

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenameld*
- Exclure l'effet de l'opération *rename*

Intuition

1. *id* fait partie des identifiants renommés : doit retourner son ancienne valeur

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenameId*
- Exclure l'effet de l'opération *rename*

Intuition

1. *id* fait partie des identifiants renommés : doit retourner son ancienne valeur
2. *id* a (potentiellement) été inséré en concurrence : doit restaurer sa (potentielle) ancienne valeur

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenamedId*
- Exclure l'effet de l'opération *rename*

Intuition

1. *id* fait partie des identifiants renommés : doit retourner son ancienne valeur
2. *id* a (potentiellement) été inséré en concurrence : doit restaurer sa (potentielle) ancienne valeur
3. *id* a été inséré après le renommage : doit retourner une valeur qui préserve l'ordre

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

- Prend la forme de l'algorithme *revertRenamed*
- Exclure l'effet de l'opération *rename*

Intuition

1. *id* fait partie des identifiants renommés : doit retourner son ancienne valeur
2. *id* a (potentiellement) été inséré en concurrence : doit restaurer sa (potentielle) ancienne valeur
3. *id* a été inséré après le renommage : doit retourner une valeur qui préserve l'ordre

Distingue cas par filtrage par motif

Annuler l'effet d'une opération *rename*

Ajout d'un nouveau mécanisme de transformation

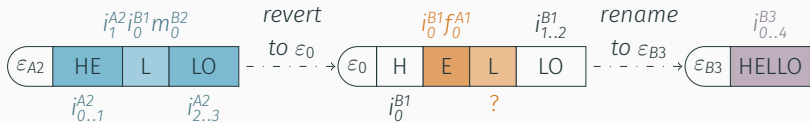
- Prend la forme de l'algorithme *revertRenamed*
- Exclure l'effet de l'opération *rename*

Intuition

1. *id* fait partie des identifiants renommés : doit retourner son ancienne valeur
2. *id* a (potentiellement) été inséré en concurrence : doit restaurer sa (potentielle) ancienne valeur
3. *id* a été inséré après le renommage : doit retourner une valeur qui préserve l'ordre

Distingue cas par filtrage par motif

Exemple de `revertRenameId`

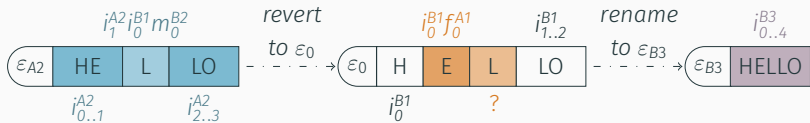


Rappel :

$$\text{renIds}_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_1^{A2} i_0^{B1} m_0^{B2}$

Exemple de `revertRenameId`



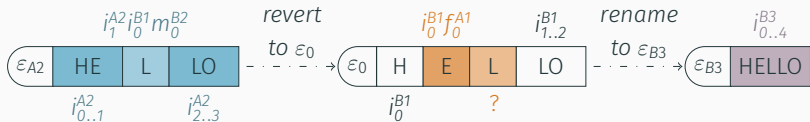
Rappel :

$$\text{renIds}_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_1^{A2} i_0^{B1} m_0^{B2}$

- Est de la forme $i_1^{A2} \oplus i_0^{B1} m_0^{B2}$: cas 2 ou 3

Exemple de `revertRenameId`



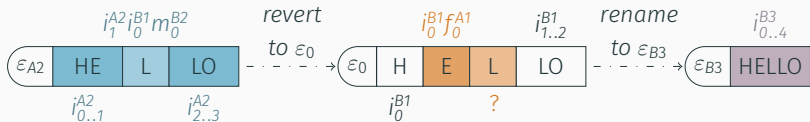
Rappel :

$$renIds_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_1^{A2} i_0^{B1} m_0^{B2}$

- Est de la forme $i_1^{A2} \oplus i_0^{B1} m_0^{B2}$: cas 2 ou 3
- Trouver l'équivalent de i_1^{A2} dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$

Exemple de `revertRenameId`



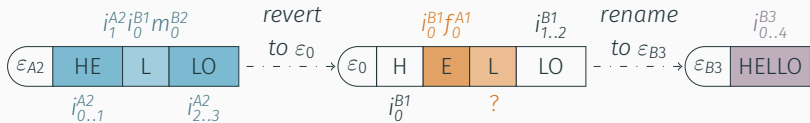
Rappel :

$$renIds_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_1^{A2} i_0^{B1} m_0^{B2}$

- Est de la forme $i_1^{A2} \oplus i_0^{B1} m_0^{B2}$: cas 2 ou 3
- Trouver l'équivalent de i_1^{A2} dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$
- Trouver l'équivalent de i_2^{A2} dans $renIds_{A2}$: i_1^{B1}

Exemple de revertRenameId



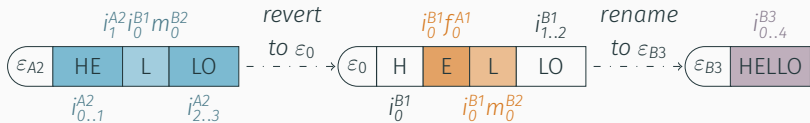
Rappel :

$$\text{renIds}_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_1^{A2} i_0^{B1} m_0^{B2}$

- Est de la forme $i_1^{A2} \oplus i_0^{B1} m_0^{B2}$: cas 2 ou 3
- Trouver l'équivalent de i_1^{A2} dans renIds_{A2} : $i_0^{B1} f_0^{A1}$
- Trouver l'équivalent de i_2^{A2} dans renIds_{A2} : i_1^{B1}
- Comparer $i_0^{B1} m_0^{B2}$ avec ces derniers : $i_0^{B1} f_0^{A1} <_{id} i_0^{B1} m_0^{B2} <_{id} i_1^{B1}$

Exemple de revertRenameId



Rappel :

$$renIds_{A2} = [i_0^{B1}, i_0^{B1} f_0^{A1}, i_1^{B1}, i_2^{B1}]$$

Exemple avec $i_1^{A2} i_0^{B1} m_0^{B2}$

- Est de la forme $i_1^{A2} \oplus i_0^{B1} m_0^{B2}$: cas 2 ou 3
- Trouver l'équivalent de i_1^{A2} dans $renIds_{A2}$: $i_0^{B1} f_0^{A1}$
- Trouver l'équivalent de i_2^{A2} dans $renIds_{A2}$: i_1^{B1}
- Comparer $i_0^{B1} m_0^{B2}$ avec ces derniers : $i_0^{B1} f_0^{A1} <_{id} i_0^{B1} m_0^{B2} <_{id} i_1^{B1}$
- Retourner $i_0^{B1} m_0^{B2}$