

Correction de l'examen : Partiel DFS

2023-2024 Session 1-2

Exercice 1 : Modèle initial - Messages

1.1. Ajout de l'invariant

- $\text{inv4} : \text{Messages} = \text{Stored} \cup \text{InTransit}$
- $\text{inv5} : \text{Stored} \cap \text{InTransit} = \emptyset$

Type d'invariant :

- inv4 est une propriété de couverture (tout message est soit stocké, soit en transit).
- inv5 est une propriété de consistance (les deux ensembles ne peuvent pas avoir de recouvrement).

1.2. Initialisation

```
INITIALISATION
THEN
  Messages :=
  Stored :=
  InTransit :=
END
```

Exercice 2 : Événements

2.1. Compléter les gardes

- Pour Send : $\text{grd2} : m \in \text{Stored}$

- Pour Receive : $\text{grd2} : m \in \text{InTransit}$
- Pour Create : $\text{grd2} : m \notin \text{Messages}$

2.2. Compléter les actions

- Pour Send :

```

THEN
  Stored := Stored \ {m}
  InTransit := InTransit \ {m}
END

```

- Pour Receive :

```

THEN
  InTransit := InTransit \ {m}
  Stored := Stored \ {m}
END

```

- Pour Create :

```

THEN
  Messages := Messages \ {m}
  Stored := Stored \ {m}
END

```

2.3. Obligation de preuve : Démontrer que Send préserve inv5

- **Hypothèse initiale** : $\text{Stored} \cap \text{InTransit} = \emptyset$
- **Actions de Send** : modifient Stored et InTransit de manière disjointe.
- **Conclusion** : Stored et InTransit restent disjoints après exécution de Send.

Exercice 3 : Modélisation du réseau

3.1. Compléter axm1

axm1: Network NOEUDS NOEUDS

Justification : Le réseau est représenté comme une relation binaire (un graphe orienté) entre les noeuds.

3.2. Compléter axm2

axm2: $n \cdot (n \text{ NOEUDS } n' \cdot (n \text{ } n') \text{ Network } (n' \text{ } n) \text{ Network})$

Justification : Chaque noeud appartient au graphe décrit par Network.

Exercice 4 : En-tête avec stockage par noeuds

4.1. L'opérateur $7 \rightarrow$

Représente une relation fonctionnelle. Ici, chaque message est associé à un unique noeud.

4.2. Compléter les invariants

inv3: Stored = dom(StoredAt)

inv4: InTransit = dom(InTransitTo)

4.3. Type des invariants

Ce sont des invariants de raffinement, car ils relient les variables concrètes aux variables abstraites.

4.4. Initialisation

```
INITIALISATION
  THEN
    Messages :=
    StoredAt :=
    InTransitTo :=
  END
```

Exercice 5 : Raffinement de Send

5.1. Compléter les gardes

```
grd3: StoredAt(m)  n  Network
grd4: m  dom(StoredAt)
```

5.2. Actions

```
THEN
  StoredAt := StoredAt \ {m}
  InTransitTo(m) := n
END
```

5.3. Obligations de preuve

- Le raffinement doit préserver l'invariant abstrait (par exemple, relation entre Stored et StoredAt).
- Les gardes abstraites doivent être implicites dans les gardes concrètes.

Exercice 6 : Événements Receive et Create

6.1. Actions pour Receive

```
THEN
  InTransitTo := InTransitTo \ {m}
  StoredAt(m) := InTransitTo(m)
END
```

6.2. Gardes et actions pour Create

```
grd3: n  NOEUDES
```

```
THEN
  Messages := Messages  {m}
  StoredAt(m) := n
END
```

Exercice 7 : Ajout des variables

7.1. Types des variables

```
inv1: Origin  MESSAGES → NOEUDS  
inv2: Destination MESSAGES → NOEUDS  
inv3: Done    MESSAGES
```

7.2. Compléter l'invariant

```
inv4: m · (m Messages m dom(Origin) m dom(Destination))
```

7.3. Ajouter inv5

```
inv5: m · (m Done StoredAt(m) = Destination(m))
```

7.4. Ajouter inv6

```
inv6: m · (m Messages Origin(m) Destination(m))
```

Exercice 8 : Intégration des nouvelles variables

8.1. Raffinement de Create

```
THEN  
  Messages := Messages {m}  
  StoredAt(m) := n  
  Origin(m) := n  
  Destination(m) := d  
END
```

8.2. Raffinement de Receive

Si $\text{StoredAt}(m) = \text{Destination}(m)$, ajouter m à Done .

8.3. Contrainte pour Send

```
grd5: m Done
```