1 Files d'attente et équivalences

1.1 $B2err \gtrsim B1$

$B2err$ simule $B1$ car :
- $B2err \gtrsim B1$:
  - $B1 \xrightarrow{inp} outp; B1$, et il existe $B2err \xrightarrow{inp} outp; B2err$.
  - Il faut donc que $outp; B2err \gtrsim outp; B1$.
- $outp; B2err \gtrsim outp; B1$:
  - $outp; B1 \xrightarrow{outp} B1$, et il existe $outp; B2err \xrightarrow{outp} B2err$.
  - Il faut donc que $B2err \gtrsim B1$, ce qui est notre premier point.

1.2 $\neg$ $B2err$ **conf** $B1$

$B2err$ n'est pas conforme à $B1$ car $Ref(B2err, inp) = \{\{inp\}, \{outp\}\}$, alors que $Ref(B1, inp) = \{\{outp\}\}$, et donc on n'a pas $Ref(B2err, inp) \subseteq Ref(B1, inp)$.

1.3 $B2par$ **respecte-t-il la spécification** B_20 **?**

$B2par$ respecte la spécification B_20 :

i) Un message émis a toujours été reçu : pour qu'un message soit émis par la porte *outp* du $B1$ de droite, il faut qu'il ait été reçu par la porte *mid* du $B1$ de droite, synchronisé sur la porte *mid* du $B1$ de gauche. Pour que le message soit émis sur la porte *mid* du $B1$ de gauche, il faut qu'il ait au préalable été reçu sur la porte *inp* du $B1$ de gauche.

ii) Lorsqu'un $B1$ reçoit un message, il refuse les *inp* jusqu'à ce qu'il ait effectué un *outp*. Comme la porte *outp* du $B1$ de gauche est synchronisée sur la porte *inp* du $B1$ de droite, cela signifie que si les deux $B1$ «contiennent» un message, celui de droite n'accepera pas d'*inp* tant qu'il n'aura pas effectué d'*outp*, et en cascade, celui de gauche ne pourra pas faire d'*outp* tant que celui de droite n'aura pas été «vidé», et donc celui de gauche refusera les *inp*. On ne pourra donc pas effectuer un *inp* sur le $B1$ de gauche tant que les deux $B1$ «contiendront» un message, et il ne pourra donc pas y avoir plus de deux messages dans la file.

1.4 $B2seq \approx B2par$

$B2seq$ et $B2par$ sont observationnellement équivalents car :
- $B2s \approx B2par$:
  - $B2s \xrightarrow{inp} B21$, et il existe $B2par \xRightarrow{\hat{inp}} (i; B1[inp, i] \;||[i]||\; B1[i, outp])$ et
    $B2par \xRightarrow{\hat{inp}} (B1[inp, i] \;||[i]||\; outp; B1[i, outp])$.
  - Dans l'autre sens, $B2par \xrightarrow{inp} (i; B1[inp, i] \;||[i]||\; B1[i, outp])$ et il existe $B2s \xrightarrow{inp} B21$.
  - Il faut donc que $B21 \approx (i; B1[inp, i] \;||[i]||\; B1[i, outp])$.
- $B21 \approx (i; B1[inp, i] \;||[i]||\; B1[i, outp])$:
  - $B21 \xrightarrow{inp} outp; B21$, et il existe
    $(i; B1[inp, i] \;||[i]||\; B1[i, outp]) \xRightarrow{\hat{inp}} (i; B1[inp, i] \;||[i]||\; outp; B1[i, outp])$
  - $B21 \xrightarrow{outp} B2s$, et et il existe
    $(i; B1[inp, i] \;||[i]||\; B1[i, outp]) \xRightarrow{\hat{outp}} (B1[inp, i] \;||[i]||\; B1[i, outp])$, autrement dit
    $(i; B1[inp, i] \;||[i]||\; B1[i, outp]) \xRightarrow{\hat{outp}} B2par$

- Dans l'autre sens, $(i; B1[inp, i] \;||[i]||\; B1[i, outp]) \xrightarrow{i} (B1[inp, i] \;||[i]||\; outp; B1[i, outp])$, mais comme on $\hat{i}$ est le chemin vide, on n'aura rien à vérifier sur $B21$.
- Il faut donc que $outp; B21 \approx (i; B1[inp, i] \;||[i]||\; outp; B1[i, outp])$, et que $B2s \approx B2par$. Cette deuxième condition est en fait notre premier point.

– $outp; B21 \approx (i; B1[inp, i] \;||[i]||\; outp; B1[i, outp])$:
- $outp; B21 \xrightarrow{outp} B21$, et il existe $(i; B1[inp, i] \;||[i]||\; outp; B1[i, outp]) \stackrel{\hat{outp}}{\Longrightarrow} (i; B1[inp, i] \;||[i]||\; B1[i, outp])$.
- Il faut donc que $B21 \approx (i; B1[inp, i] \;||[i]||\; B1[i, outp])$, ce qui est notre deuxième point.

1.5 $B2ent = B2par$

Étant donné que $B2ent$ et $B2par$ ne commencent pas par l'action interne, tester leur égalité au sens de la congruence observationnelle revient à tester s'ils sont observationnellement équivalents.

On applique donc la même méthode que dans la section précédente, avec les équivalences suivantes :

– Pour que $B2ent = B2par$, il faut qu'après inp (la seule action que les systèmes peuvent exécuter dans leurs états respectifs), $(outp; B1[inp; outp]|||B1[inp; outp]) \approx (i; B1[inp; i] \;||[i]||\; B1[i, outp])$.

– Pour cela, il faut qu'après $outp$, $(B1[inp; outp]|||B1[inp; outp]) \approx (B1[inp; i] \;||[i]||\; B1[i, outp])$, autrement dit $B2ent = B2par$, ce qui est une condition plus faible que le premier point.
Il faut aussi qu'après inp, $(outp; B1[inp; outp]|||outp; B1[inp; outp]) \approx (i; B1[inp; i] \;||[i]||\; outp; B1[i, outp])$.
Dans l'autre sens, on peut ignorer ce qui se passe après le i de $(i; B1[inp; i] \;||[i]||\; B1[i, outp])$, car $\hat{i}$ est le chemin vide, donc pas de conditions sur $(outp; B1[inp; outp]|||B1[inp; outp])$.

– Occupons-nous de $(outp; B1[inp; outp]|||outp; B1[inp; outp]) \approx (i; B1[inp; i] \;||[i]||\; outp; B1[i, outp])$. Dans ces états, les systèmes ne peuvent faire qu'$outp$. Il faut donc qu'après $outp$, $(outp; B1[inp; outp]|||B1[inp; outp]) \approx (i; B1[inp; i] \;||[i]||\; B1[i, outp])$, ce qui correspond au deuxième point.

1.6 $B2par \gtrapprox B1$

$B2par$ simule observationnellement $B1$ car :

- $B1 \xrightarrow{inp} outp; B1$, et il existe $B2par \stackrel{\hat{inp}}{\Longrightarrow} (i; B1[inp, i] \;||[i]|B1[i; outp])$ et $B2par \stackrel{\hat{inp}}{\Longrightarrow} (B1[inp, i] \;||[i]|outp; B1[i; outp])$. Il suffit donc que soit $(i; B1[inp, i] \;||[i]|B1[i; outp]) \gtrapprox outp; B1$, soit $(B1[inp, i] \;||[i]|outp; B1[i; outp]) \gtrapprox outp; B1$. On prend
- On prend cette deuxième possibilité. $outp; B1 \xrightarrow{outp} B1$, et il existe $(B1[inp, i] \;||[i]|outp; B1[i; outp]) \stackrel{ou}{=}$ $(B1[inp, i] \;||[i]|B1[i; outp])$, autrement dit $(B1[inp, i] \;||[i]|outp; B1[i; outp]) \stackrel{\hat{outp}}{\Longrightarrow} B2par$, notre premier point.

1.7 $\not B2par$ **conf** $B1$

Pour que $B2par$ soit conforme à $B1$, car pour toute trace t de $B1$, $\mathrm{Acc}(B2par) \subset\subset \mathrm{Ref}(B1)$.

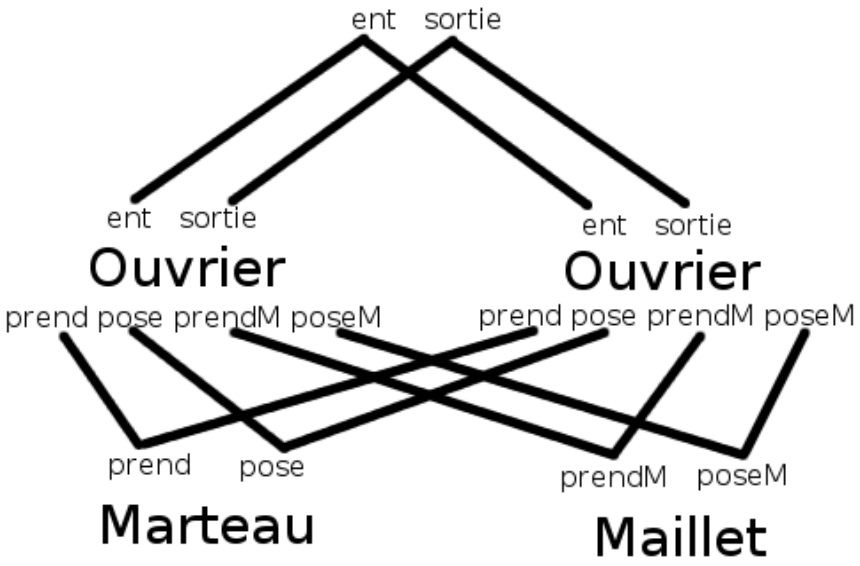
- Pour la trace $\emptyset$, $\mathrm{Acc}(B2par) = \mathrm{Acc}(B1) = \{\{inp\}\}$.
- Pour la trace inp, $\mathrm{Acc}(B2par) = \{\{\}, \{inp, outp\}\}$, mais $\mathrm{Acc}(B1) = \{\{outp\}\}$.

– Pour la trace $inp; outp$, on est revenu aux mêmes états que la trace $\emptyset$.

Pour le point 2 $\forall X \in \{\{\}, \{inp, outp\}\}, \quad \{outp\} \not\subseteq X$, et donc $\neg(B2par \text{ conf } B1)$

2 Exercice de l'atelier

2.1 Question 1



2.2 Question 2

```
Maillet[prendM,poseM] := prendM;MailletOcc[prendM,poseM]
MailletOcc[prendM,poseM] := poseM;Maillet[prendM,poseM]
```

2.3 Question 3

```
UtiliseMarteau[ent,sortie,prend,pose,prendM,poseM](travail:TRAVAIL) :=
  prend;sortie!fait(travail);pose;
  Ouvrier[ent,sortie,prend,pose,prendM,poseM]

UtiliseMaillet[ent,sortie,prend,pose,prendM,poseM](travail:TRAVAIL) :=
  prendM;sortie!fait(travail);poseM;
  Ouvrier[ent,sortie,prend,pose,prendM,poseM]

Commence[ent,sortie,prend,pose,prendM,poseM](travail:TRAVAIL) :=
  [facile(travail)] ->
    sortie!fait(travail);
    Ouvrier[ent,sortie,prend,pose,prendM,poseM]
  []
  [difficile(travail)] ->
    UtiliseMarteau[ent,sortie,prend,pose,prendM,poseM](travail)
```

```
  []
  [(not(facile(travail)) and not(difficile(travail)))] ->
    UtiliseOutil[ent,sortie,prend,pose,prendM,poseM](travail)

Ouvrier[ent,sortie,prend,pose,prendM,poseM] :=
  ent?travail:TRAVAIL;
  Commence[ent,sortie,prend,pose,prendM,poseM](travail)

Atelier[ent,sortie] :=
  hide prend,pose,prendM,poseM in
    (
      (
        Marteau[prend,pose]
        |||
        Maillet[prendM,poseM]
      )
      |[prend,pose,prendM,poseM]|
      (
        Ouvrier[ent,sortie,prend,pose,prendM,poseM]
        |||
        Ouvrier[ent,sortie,prend,pose,prendM,poseM]
      )
    )
```

2.4 Question 4

Nécessite l'outil `casear`.

2.5 Question 5

```
AtelierV[ent,sortie] :=
  hide e,s in
    Verificateur[ent,sortie,e,s]
    |[e,s]|
    Atelier[e,s]
```

La composition parallèle générale synchronisant les ouvriers et le vérificateur est la synchronisation sur l'ensemble de portes $\{ent, sortie\}$ de l'atelier sans vérificateur.

On place le vérificateur «entre» les ouvriers et l'environnement : Si un travail est mal fait, le vérificateur le défait et le repasse à l'entrée des ouvriers, sinon, il l'envoie vers la sortie de l'environnement. À tout moment, il peut récupérer du travail depuis l'environnement, et le passer aux ouvriers. Nous avons choisi de ne pas limiter le nombre de fois que le vérificateur peut laisser les ouvriers obtenir du travail depuis l'environnement, puisque nous ne savons pas combien de travaux peuvent être sur les tapis roulants *ent* et *sortie* sans que le vérificateur y ait encore eu accès.

```
Verificateur[ent,sortie,e,s] :=
  (
    s?produit:PRODUIT;
    [not(correct(produit))] ->
      e!defait(produit);Verificateur[ent,sortie,e,s]
```

```
    []
    [correct(produit)] ->
      sortie!produit;Verificateur[ent,sortie,e,s]
  )
  []
  (
    ent?travail:TRAVAIL;
    e!travail;
    Verificateur[ent,sortie,e,s]
  )
```

2.6 Question 6

Assemble n'est pas observationnellement équivalent à *Atelier*, car après avoir exécuté l'action *ent*, *Atelier* peut encore exécuter *ent* (puisqu'il y a deux ouvriers, qui peuvent accepter deux travaux), tandis que *Assemble* ne pourra exécuter que *sortie*.

- $Atelier[\dots] \xrightarrow{ent} (\dots) \,|[\dots]|\, (Commence[\dots]|||Ouvrier[\dots])$, et il existe $Assemble[\dots] \xrightarrow{ent} Assembler[\dots]$. Il faudrait que $(\dots) \,|[\dots]|\, (Commence[\dots]|||Ouvrier[\dots]) \approx Assembler[\dots]$.
- Mais $(\dots) \,|[\dots]|\, (Commence[\dots]|||Ouvrier[\dots]) \not\approx Assembler[\dots]$ car $(\dots) \,|[\dots]|\, (Commence[\dots]|||Ouvrier[\dots]) \xrightarrow{ent}$ $(\dots) \,|[\dots]|\, (Commence[\dots]|||Commence[\dots])$ tandis que $Assembler[\dots] \overset{ent}{\not\longrightarrow}$.

2.7 Question 7

Nécessite l'outil `aldebaran`.