

Troisième partie

Raffinement de programme



Plan

- 1 Introduction
- 2 Implantation des modules en TLA
- 3 Sémantique d'exécution en TLA
- 4 Exemple : le jeu de Marienbad
 - Exemple d'exécution du module Marienbad
- 5 Définitions et concepts préliminaires
- 6 Raffinement de modules
 - Définition
- 7 Implantation du raffinement en TLA
- 8 Exemple : stratégie gagnante au jeu de Marienbad
 - Exemples de vérification du raffinement de Marienbad
- 9 Patrons de raffinement de module



Raffinement et simulation

- On s'intéresse au raffinement de modules = variables + procédures \simeq Classe
- Le raffinement que nous considérons est lié à la simulation dans les systèmes de transitions.
- Les propriétés temporelles de sûreté du module \mathcal{M} seront donc “préservées” par le module \mathcal{M}' .
- Le module raffiné \mathcal{M} sera appelé module abstrait.
- Le module raffinant \mathcal{M}' sera appelé module concret.



Module versus S.T.E.

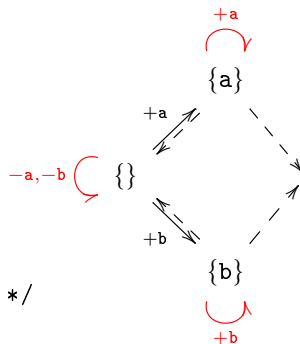
Module = Interface + Action

- **Interface** : décrit les procédures exécutables, à chaque instant
- dépend des paramètres et des variables internes du module
- décrit la **potentialité** des actions
- également appelé “**précondition**”
- **Action** : décrit l'exécution des procédures
- valeur de retour et changement d'état des variables internes
- décrit les **transitions**
- également appelé “**postcondition**”



Exemple : l'ensemble

Un ensemble où l'on peut avoir 2 éléments parmi a et b



```

interface Ensemble
  /* précondition: e not in this */
  void add(a_ou_b e);

  /* précondition: e in this */
  void rem(a_ou_b e);
  
```



Interface et potentialité

- la correspondance procédure \leftrightarrow action n'est pas unique :
 - version statique : une action = une procédure sans paramètres
 - version dynamique : une seule procédure, l'action est en paramètre
- compromis en général, par ex. l'ensemble
- question : l'interface exprime-t-elle ce que le module peut ou doit pouvoir faire ?
- raffinement \neq simulation



Interface et potentialité

Interface

- ce que l'utilisateur **peut** demander au module
- le module **doit** toujours pouvoir le faire

Définition pour un S.T.E.

$$\mathcal{I}(s) \triangleq \{l \in L \mid \forall s'. (s \Rightarrow s') \implies \exists s''. s' \xRightarrow{l} s''\}$$

Une étiquette l appartient à l'interface d'un état s ssi quels que soient les états atteints à partir de s par τ -transitions (spontanées), une transition par l est toujours possible.



Exemples d'interfaces

- $P \triangleq a + b$, a pour interface $\mathcal{I}(P) = \{a, b\}$
- $Q \triangleq \tau.a + \tau.b$, a pour interface $\mathcal{I}(Q) = \{\}$
- ils ne sont pas bisimilaires (P est plus “fiable” que Q)
- mais néanmoins $P \leq Q$ et $Q \leq P!!$
- lequel raffine l'autre ?



Raffinement

Raffinement = simulation + prise en compte de l'interface

Définition 1 (Relation de Raffinement)

On dit qu'une relation $Raff \subseteq S \times S'$ est une relation de raffinement de S' par S ssi on a les 2 propriétés suivantes :

$$\forall s_1, s_2 \in S, s'_1 \in S', I \in \mathcal{I}(s'_1).$$

$$(\langle s_1, s'_1 \rangle \in Raff \wedge s_1 \xrightarrow{I} s_2$$

$$\Rightarrow$$

$$\exists s'_2. \langle s_2, s'_2 \rangle \in Raff \wedge s'_1 \xrightarrow{I} s'_2)$$

$$\wedge$$

$$(\langle s_1, s'_1 \rangle \in Raff \wedge s_1 \xrightarrow{\tau} s_2$$

$$\Rightarrow$$

$$\exists s'_2. \langle s_2, s'_2 \rangle \in Raff \wedge s'_1 \Rightarrow s'_2)$$

$$\forall s_1 \in S, s'_1 \in S'.$$

$$\langle s_1, s'_1 \rangle \in Raff$$

$$\Rightarrow$$

$$\mathcal{I}(s_1) \supseteq \mathcal{I}(s'_1)$$

Raffinement

- la formule induit une contrainte supplémentaire, mais régulière (inclusion) → toujours des petits carrés, mais moins!
 - on retrouvera les 2 parties dans la définition du raffinement appliquée aux modules
 - les relations de raffinement ont les mêmes propriétés que les relations de simulation
- i.e. transitivité, ordre, union, plus grande relation, etc
- la plus grande relation de raffinement est notée $S' \sqsubseteq S$
 - en pratique, les données manipulées par un module sont trop grandes pour calculer le plus grand raffinement
- il faut fournir une relation manuellement ou utiliser des **patrons de raffinement**



Développement par raffinement

Pas de distinction nette entre spécification et implantation car :

- spécification et implantation s'expriment dans le même langage.
- un module \mathcal{M}' peut jouer un rôle :
 - de spécification vis-à-vis d'un module \mathcal{M}''
 - d'implantation (raffinement) vis-à-vis d'un module \mathcal{M} .



Développement par raffinement

Les étapes du développement par raffinement sont :

- 1 De partir d'un **module initial** faisant office de **spécification**. La description doit être :
 - assez précise pour capturer les besoins.
 - assez abstraite pour pouvoir prouver facilement la correction (invariants).
 - ne pas oblitérer des choix possibles d'implantation.



Développement par raffinement

Les étapes du développement par raffinement sont :

- ② De produire successivement des modules intermédiaires qui se raffinent, à partir de la spécification.
 - On utilise en g^{al} un ens. de **transformations de programmes**.
 - Ces transformations sont choisies et appliquées par le développeur.
 - Elles doivent minimiser ou éviter la preuve de raffinement.



Développement par raffinement

Les étapes du développement par raffinement sont :

- ③ D'obtenir enfin un module **implantable**, i.e. :
 - **déterministe**.
 - respectant la spécification.
 - Pas de deadlock supplémentaire (% à la spécification).
 - dont les actions et les variables peuvent s'implanter directement par un simple changement de syntaxe dans un langage de programmation.



Contrat développeur/client

La définition du raffinement de module fait intervenir la notion de contrat logiciel :

- Si le **client** respecte la spécification du module (utiliser les opérations avec leurs **préconditions** vérifiées).
- Alors le **développeur** lui fournit une implantation de ces procédures respectant les propriétés attendues par le client (par ex., l'**invariant** du module et les **postconditions**).



Sémantique de type jeu

On peut distinguer deux types d'actions :

- Les actions du client, qui consistent à choisir une procédure à exécuter, en respectant ses préconditions (i.e. son interface).
- Les actions du système, qui répondent au choix précédent en exécutant la procédure (en choisissant une image particulière), tout en respectant les propriétés attendues.

Ainsi, la sémantique d'un module correspond à un jeu à tour de rôle entre :

- Le développeur qui essaye de préserver le bon fonctionnement de son module.
- Le client qui peut l'utiliser de toutes les façons possibles pour le mettre en défaut (\simeq test exhaustif).



Sémantique de type jeu

De ce point de vue :

- Un développement par raffinement consiste à déterminer une **stratégie gagnante** pour le développeur.
- Une stratégie “gagnante”, car elle garantit que le module final sera correct % à sa spécification.
- Cette stratégie est donnée sous forme algorithmique (le “code” du module final).

Le développeur du module gagne ssi :

- il s'agit d'une partie finie, c'est au tour du client de jouer et il ne peut plus jouer.
- il s'agit d'une partie infinie.



Plan

- 1 Introduction
- 2 **Implantation des modules en TLA**
- 3 Sémantique d'exécution en TLA
- 4 Exemple : le jeu de Marienbad
 - Exemple d'exécution du module Marienbad
- 5 Définitions et concepts préliminaires
- 6 Raffinement de modules
 - Définition
- 7 Implantation du raffinement en TLA
- 8 Exemple : stratégie gagnante au jeu de Marienbad
 - Exemples de vérification du raffinement de Marienbad
- 9 Patrons de raffinement de module



Représentation des procédures

- Une (spécification de) procédure, à la JML, est sous la forme :

```
//@ requires pre(param,var)
//@ ensures post(param,\old(var),var,\result)
T_result procedure(T_param param);
```

- Elle sera représentée par les prédicats TLA suivants :

```
Pre_procedure(param, var) ==
/\ param \in T_param
/\ pre(param, var)
```

```
Act_procedure(param, var, var_prime, result) ==
/\ result \in T_result
/\ post(param, var, var_prime, result)
```



Représentation des modules

On assimile module/classe et procédure/méthode :

```
class Mod {  
  // variable d'état = attribut  
  T_var var;  
  //@ invariant inv(var)  
  // initialisation: constructeur 'abstrait'  
  //@ ensures init(var)  
  Mod();  
  // procédures 'abstraites'  
  // procédure 1:  
  //@ requires pre1(param,var)  
  //@ ensures post1(param,\old(var),var,\result)  
  T_result procedure1(T_param param);  
  // procédure 2:  
  ...  
}
```



Module TLA équivalent

On trouve :

- Le prédicat d'initialisation, l'invariant, les pré/post-conditions.
- aucune variable de module, uniquement des définitions.

```
---- MODULE mod ----  
\* type des états du module  
ETAT == T_var  
\* invariant du module  
Invariant(var) ==  
/\ var \in ETAT  
/\ ...  
\* état initial  
Init(var) == ...  
\* traduction TLA des procédures  
...
```



Module TLA équivalent (suite)

- On trouve enfin les prédicats/contrats qui définissent les actions permises du client et du module.
- Ces contrats sont utilisés pour exécuter/prouver :
 - la faisabilité et la correction d'un module.
 - le raffinement entre modules.
- Engendrés automatiquement.

```
\* CONTRAT CLIENT
```

```
ContratClient(choix, param, var) ==
```

```
\/ (choix="proc1" /\ Pre_procedure1(param,var))
```

```
\/ (choix="proc2" /\ Pre_procedure2(param,var))
```

```
\/ ...
```

```
\* CONTRAT MODULE
```

```
ContratModule(choix, param, var, var_prime, result) ==
```

```
\ (choix="proc1" => Act_proc1(param,var,var_prime,result))
```

```
\ (choix="proc2" => Act_proc2(param,var,var_prime,result))
```

```
\ ...
```



Plan

- 1 Introduction
- 2 Implantation des modules en TLA
- 3 Sémantique d'exécution en TLA
- 4 Exemple : le jeu de Marienbad
 - Exemple d'exécution du module Marienbad
- 5 Définitions et concepts préliminaires
- 6 Raffinement de modules
 - Définition
- 7 Implantation du raffinement en TLA
- 8 Exemple : stratégie gagnante au jeu de Marienbad
 - Exemples de vérification du raffinement de Marienbad
- 9 Patrons de raffinement de module



Exécution d'un module

- Interprète générique (i.e. paramétré par le module à exécuter).
- Permet d'engendrer toutes les exécutions/parties possibles.
- Module TLA `run_module.tla`.

```
---- MODULE run_module ----
```

```
VARIABLES
```

```
    Param, Etat, Result, \* variables utilisateur
```

```
    Tour, Choix           \* variables internes de simulation
```

```
\* paramètres symboliques instanciés
```

```
\* par le module à exécuter
```

```
CONSTANTS
```

```
    Init(_),
```

```
    ContratClient(_,_,_),
```

```
    ContratModule(_,_,_,_,_)
```



Exécution d'un module (suite)

- Les états initiaux.
- Le client fera le premier mouvement (le client et le module jouant à tour de rôle).

```
Initial ==  
  /\ Tour    = "client"  
  /\ Choix   = "__NO_DATA"  
  /\ Param   = "__NO_DATA"  
  /\ Result  = "__NO_DATA"  
  /\ Init(Etat)
```



Exécution d'un module (suite)

- Les actions du client instanciées : définissant le choix de la procédure à exécuter et ses paramètres.
- Les actions du module instanciées : définissant le résultat de la procédure et le changement d'état des variables du module.

```
ActionClient ==
```

```
  /\ Tour      = "client"
```

```
  /\ Tour'     = "module"
```

```
  /\ Etat'     = Etat
```

```
  /\ Result'   = "__NO_DATA"
```

```
  /\ ContratClient(Choix', Param', Etat)
```

```
ActionModule ==
```

```
  /\ Tour      = "module"
```

```
  /\ Tour'     = "client"
```

```
  /\ Choix'    = "__NO_DATA"
```

```
  /\ Param'    = "__NO_DATA"
```

```
  /\ ContratModule(Choix, Param, Etat, Etat', Result')
```

Exécution d'un module (fin)

- Enfin, le système de transitions.

Next ==

 \ / ActionClient

 \ / ActionModule

Spec ==

 /\ Initial

 /\ [] [Next] __ <<Tour, Choix, Param, Etat, Result>>

Pour exécuter, il reste à définir :

- Un module principal TLA : interprète + module.
- Son fichier de configuration : S.T. + invariants et autres propriétés.



Plan

- 1 Introduction
- 2 Implantation des modules en TLA
- 3 Sémantique d'exécution en TLA
- 4 Exemple : le jeu de Marienbad
 - Exemple d'exécution du module Marienbad
- 5 Définitions et concepts préliminaires
- 6 Raffinement de modules
 - Définition
- 7 Implantation du raffinement en TLA
- 8 Exemple : stratégie gagnante au jeu de Marienbad
 - Exemples de vérification du raffinement de Marienbad
- 9 Patrons de raffinement de module



Le jeu de Marienbad

- Le jeu de Marienbad ou jeu des allumettes se joue à 2 joueurs.
 - Chaque joueur prend à tour de rôle entre 1 et 3 allumettes dans un tas.
 - Initialement, le tas contient 21 allumettes.
 - Le perdant est celui qui tire la dernière allumette.
 - Il existe une stratégie gagnante pour le second joueur.
-
- Le client joue en premier.
 - Une seule action jouer.
 - La précondition définit les coups valides du client.
 - La postcondition définit les réponses valides du module (et exécute les 2 demi-coups).



Module sous forme TLA

```
----- MODULE marienbad -----  
EXTENDS Naturals  
-----  
\* INVARIANT  
ETAT == 1..21  
Inv(tas) ==  
  /\ tas \in ETAT  
-----  
\* ETAT INITIAL  
Init(tas) ==  
  /\ tas = 21  
-----
```



Module sous forme TLA

```
\* PROCEDURE jouer
Pre_jouer(param, tas) ==
  \E a \in 1..3 : tas - a > 0 /\ param = a

Act_jouer(param, tas, tas_prime, result) ==
  \E a \in 1..3 :
    /\ tas - param - a > 0
    /\ tas_prime = tas - param - a
    /\ result      = "__NO_DATA"

\* PROCEDURE perdu
Pre_perdu(param, etat) == param = 0 /\ etat = 1

Act_perdu(param, etat, etat_p, result) ==
  /\ etat_p = etat
  /\ result = "__NO_DATA"
```



Module sous forme TLA

```
\* CONTRAT CLIENT
```

```
ContratClient(choix, param, tas) ==
```

```
  /\ (choix="jouer" /\ Pre_jouer(param,tas))
```

```
  /\ (choix="perdu" /\ Pre_perdu(param, etat))
```

```
-----  
\* CONTRAT MODULE
```

```
ContratModule(choix, param, tas, tas_prime, result) ==
```

```
  /\ (choix="jouer" => Act_jouer(param,tas,tas_prime,result))
```

```
  /\ (choix="perdu" => Act_perdu(param,etat,etat_p,result))
```

```
=====
```



Exécution du module

- On vérifie les propriétés du module.
- Module principal `run_marienbad.tla`.

```

----- MODULE run_marienbad -----
EXTENDS var_module, contrats_marienbad
INSTANCE run_module

Invariant == [] Inv(Etat)
=====

```

- Fichier de configuration associé `run_marienbad.cfg`.

```

/* On vérifie les propriétés du module, i.e :
/* - l'invariant est respecté (Invariant)
/* - il n'y pas de deadlock
SPECIFICATION Spec
PROPERTIES Invariant

```

Vérification

On lance le model checker

```
% java -jar tla2tools.jar run_marienbad  
[...]
```

Error: Deadlock reached.

Error: The behavior up to this point is:

```
[...]
```

State 8: <Action line 3, col 1 to line 3, col 19 of module 1

```
/\ Choix = 'jouer'
```

```
/\ Result = '<NO_DATA>'
```

```
/\ Etat = 4
```

```
/\ Param = 3
```

```
/\ Tour = 'module'
```



Plan

- 1 Introduction
- 2 Implantation des modules en TLA
- 3 Sémantique d'exécution en TLA
- 4 Exemple : le jeu de Marienbad
 - Exemple d'exécution du module Marienbad
- 5 Définitions et concepts préliminaires**
- 6 Raffinement de modules
 - Définition
- 7 Implantation du raffinement en TLA
- 8 Exemple : stratégie gagnante au jeu de Marienbad
 - Exemples de vérification du raffinement de Marienbad
- 9 Patrons de raffinement de module



Faisabilité / implantabilité d'un module

Définition 2 (Faisabilité)

- ❶ Une procédure $\langle Pre, Act \rangle$ est faisable % à un invariant Inv ssi :

$$\forall p, v. Pre(p, v) \wedge Inv(v) \Rightarrow \exists v', r'. Act(p, v, v', r') \wedge Inv(v')$$

- ❷ Un module est faisable % à un invariant Inv ssi :

- $\forall i, \langle Pre_i, Act_i \rangle$ est faisable % à Inv .
- $Init \wedge Inv \neq False$, i.e. $\exists v. Init(v) \wedge Inv(v)$

Note : L'absence de deadlock en TLA garantit :

- La faisabilité du module.
- Que le client aura toujours la possibilité d'effectuer une action.



Correction d'un module

Définition 3 (Correction)

- ① Une procédure $\langle Pre, Act \rangle$ est correcte % à un invariant Inv ssi :

$$\forall p, v. Pre(p, v) \wedge Inv(v) \Rightarrow \forall v', r'. Act(p, v, v', r') \Rightarrow Inv(v')$$

- ② Un module est correct % à un invariant Inv ssi :
- $\forall i, \langle Pre_i, Act_i \rangle$ est correcte % à Inv .
 - $\forall v. Init(v) \Rightarrow Inv(v)$



Déterminisme d'un module

Définition 4 (Déterminisme)

- 1 Une procédure $\langle Pre, Act \rangle$ est déterministe % à un invariant Inv ssi :

$$\forall p, v. Pre(p, v) \wedge Inv(v) \Rightarrow \exists !v', !r'. Act(p, v, v', r') \wedge Inv(v')$$

- 2 Un module est déterministe % à un invariant Inv ssi :
 - $\langle Pre_i, Act_i \rangle$ est déterministe % à $Inv, \forall i$.
 - $\exists !v. Init(v) \wedge Inv(v)$



Généralités

- Une procédure (un module) est dit respectivement faisable ou déterministe ssi elle (il) l'est par rapport à *True*.
- Une procédure (un module) est toujours correct(e) par rapport à *True*.
- Seules les procédures faisables peuvent être raffinées en code.

Dans un développement par raffinement :

- Les modules successifs doivent toujours rester faisables et corrects. C'est ce qui garantit qu'on puisse obtenir par raffinement du code qui respecte la spécification.
- Le module ultime doit être déterministe.



Plan

- 1 Introduction
- 2 Implantation des modules en TLA
- 3 Sémantique d'exécution en TLA
- 4 Exemple : le jeu de Marienbad
 - Exemple d'exécution du module Marienbad
- 5 Définitions et concepts préliminaires
- 6 Raffinement de modules**
 - Définition
- 7 Implantation du raffinement en TLA
- 8 Exemple : stratégie gagnante au jeu de Marienbad
 - Exemples de vérification du raffinement de Marienbad
- 9 Patrons de raffinement de module



Hypothèses

Soient les modules \mathcal{M}^A et \mathcal{M}^C suivants :

- Tous deux faisables et corrects.
- Le nombre de procédures n'étant pas nécessairement le même.
- Les propriétés attendues pouvant être \neq (invariants, etc).

MODULE \mathcal{M}^A
 $Init^A(etat) \triangleq \dots$

⋮

$Pre_i^A(param, etat) \triangleq \dots$

$Act_i^A(param, etat, etat_p, result) \triangleq \dots$

⋮

MODULE \mathcal{M}^C
 $Init^C(etat) \triangleq \dots$

⋮

$Pre_i^C(param, etat) \triangleq \dots$

$Act_i^C(param, etat, etat_p, result) \triangleq \dots$

⋮



Raffinement

La relation de raffinement de \mathcal{M}^A par \mathcal{M}^C est donné par une fonction *Liaison*(*étatC*) qui à tout état concret *étatC* fait correspondre l'ensemble des états abstraits *étatA* associés.

- *Liaison* doit être “établie” par les initialisations des 2 modules.
 - *Liaison* doit être “préservée” par les actions (du client ou du module) des 2 modules.
- Donc, 3 conditions doivent être respectées.
- Il s'agit toujours d'une histoire de petits carrés...

Note : *Liaison* est appelée (abusivement) invariant de liaison, ou de couplage.



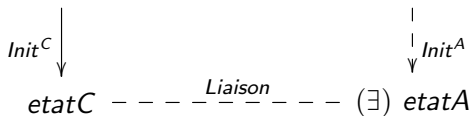
Initialisation

Définition 5 (Condition de raffinement d'état initial)

\mathcal{M}^C satisfait la condition de raffinement d'état initial
% à \mathcal{M}^A , ssi :

$$\forall \text{etat}C. \text{Init}^C(\text{etat}C) \Rightarrow \exists \text{etat}A. \text{Init}^A(\text{etat}A) \wedge \text{etat}A \in \text{Liaison}(\text{etat}C)$$

- C'est la même contrainte que pour les états initiaux dans la définition d'une simulation.
- Graphiquement :



Actions du module

Définition 6 (Condition de raffinement d'actions du module)

\mathcal{M}^C satisfait la condition de raffinement des actions du module % à \mathcal{M}^A , ssi :

$$\begin{aligned}
 & \forall \text{choix}, \text{param}, \text{etatA}, \text{etatC}, \text{etatC}', \text{result}' . \\
 & \quad \text{Inv}^A(\text{etatA}) \wedge \text{Inv}^C(\text{etatC}) \\
 & \quad \wedge \text{ContratClient}^A(\text{choix}, \text{param}, \text{etatA}) \\
 & \quad \wedge \text{ContratClient}^C(\text{choix}, \text{param}, \text{etatC}) \\
 & \quad \wedge \text{ContratModule}^C(\text{choix}, \text{param}, \text{etatC}, \text{etatC}', \text{result}') \\
 & \quad \wedge \text{etatA} \in \text{Liaison}(\text{etatC}) \\
 & \Rightarrow \exists \text{etatA}' . \text{ContratModule}^A(\text{choix}, \text{param}, \text{etatA}, \text{etatA}', \text{result}') \\
 & \quad \wedge \text{etatA}' \in \text{Liaison}(\text{etatC}')
 \end{aligned}$$

Actions du module

- Là encore, c'est la contrainte de simulation des actions (transitions) du module \mathcal{M}^C par les actions (transitions) du module \mathcal{M}^A .
- Graphiquement :

$$\begin{array}{ccc}
 \text{choix}, \text{param}, \text{etatC} & \xrightarrow{\text{Inv}^C \wedge \text{Liaison} \wedge \text{Inv}^A} & \text{choix}, \text{param}, \text{etatA} \\
 \text{ContratModule}^C \downarrow & & \downarrow \text{ContratModule}^A \\
 \text{etatC}', \text{result}' & \text{-----} \text{Liaison} \text{-----} & (\exists) \text{etatA}', \text{result}'
 \end{array}$$

Actions du client

Définition 7 (Condition de raffinement d'actions du client)

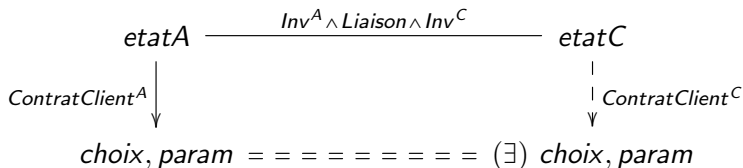
\mathcal{M}^C satisfait la condition de raffinement des actions du client
% à \mathcal{M}^A , ssi :

$$\begin{aligned} & \forall \text{choix}, \text{param}, \text{etat}^A, \text{etat}^C. \\ & \quad \text{Inv}^A(\text{etat}^A) \wedge \text{Inv}^C(\text{etat}^C) \\ & \quad \wedge \text{etat}^A \in \text{Liaison}(\text{etat}^C) \\ & \quad \wedge \text{ContratClient}^A(\text{choix}, \text{param}, \text{etat}^A) \\ & \Rightarrow \text{ContratClient}^C(\text{choix}, \text{param}, \text{etat}^C) \end{aligned}$$



Actions du client

- Ici, au contraire, les possibilités de choix du client ne doivent pas être contraintes lorsqu'on raffine, mais au contraire élargies. Ainsi, les actions du client \mathcal{M}^C simulent les actions du client \mathcal{M}^A .
- Graphiquement :



Définition du raffinement

Définition 8 (Raffinement de modules)

Le module \mathcal{M}^C raffine le module \mathcal{M}^A , noté $\mathcal{M}^C \sqsubseteq \mathcal{M}^A$, ssi Le module \mathcal{M}^C , par rapport au module \mathcal{M}^A , satisfait les conditions suivantes :

- 1 la condition de raffinement d'état initial.
- 2 la condition de raffinement d'actions du module.
- 3 la condition de raffinement d'actions du client.



Propriétés du raffinement

Le raffinement étant monotone pour les actions du module et anti-monotone pour les actions du client, on a les propriétés suivantes :

- À *ContratClient* constant, i.e.
 $\text{ContratClient}^C \Leftrightarrow \text{ContratClient}^A$, le S.T.E. $\mathcal{S}_{\mathcal{M}^A}$ du module \mathcal{M}^A simule $\mathcal{S}_{\mathcal{M}^C}$.
- Inversement, à *ContratModule* constant, i.e.
 $\text{ContratModule}^C \Leftrightarrow \text{ContratModule}^A$, le S.T.E. $\mathcal{S}_{\mathcal{M}^C}$ du module \mathcal{M}^C simule $\mathcal{S}_{\mathcal{M}^A}$.
- la relation de raffinement est une relation de pré-ordre partiel (réflexive et transitive).
- un module raffinant \mathcal{M}^C peut remplacer un module raffiné \mathcal{M}^A dans toute application, sans provoquer plus d'erreurs à l'exécution.



Plan

- 1 Introduction
- 2 Implantation des modules en TLA
- 3 Sémantique d'exécution en TLA
- 4 Exemple : le jeu de Marienbad
 - Exemple d'exécution du module Marienbad
- 5 Définitions et concepts préliminaires
- 6 Raffinement de modules
 - Définition
- 7 **Implantation du raffinement en TLA**
- 8 Exemple : stratégie gagnante au jeu de Marienbad
 - Exemples de vérification du raffinement de Marienbad
- 9 Patrons de raffinement de module



Vérification du raffinement en TLA

- Vérification générique (i.e. paramétrée par les 2 modules abstraits et concrets à comparer).
- Permet de vérifier l'existence de tous les “petits carrés” possibles.
- Pour cela, on exécute “en parallèle” les modules abstraits et concrets.
- Module TLA `run_raffinement.tla`.



Paramètres et variables du raffinement

```
---- MODULE run_raffinement ----  
VARIABLES  
    Param, EtatA, EtatC, Result, /* variables utilisateur  
    Tour, Choix, RaffOk          /* variables internes  
  
/* paramètres symboliques instanciés  
/* par les modules abstraits et concrets à exécuter  
CONSTANTS  
    InitA(_)                , InitC(_),  
    ContratClientA(_,_,_),  ContratClientC(_,_,_),  
    ContratModuleA(_,_,_,_), ContratModuleC(_,_,_,_,_),  
    Liaison(_)
```



Vérification du raffinement d'état initial

```
Initial ==  
  /\ Tour    = "client"  
  /\ Choix   = "__NO_DATA"  
  /\ Param   = "__NO_DATA"  
  /\ Result  = "__NO_DATA"  
  /\ InitC(EtatC)  
  /\ LET InitialA ==  
      { etata \in Liaison(EtatC) :  
        InitA(etata) }  
  IN IF (InitialA # {})  
    THEN /\ RaffOk = TRUE  
          /\ EtatA \in InitialA  
    ELSE /\ RaffOk = FALSE  
          /\ EtatA = "__NO_DATA"
```



Vérification du raffinement d'actions du client

```
ActionClient ==  
  /\ Tour      = "client"  
  /\ Tour'     = "module"  
  /\ Etata'    = Etata  
  /\ Result'   = "__NO_DATA"  
  /\ ContratClientA(Choix',Param',EtatA)  
  /\ UNCHANGED EtatC  
  /\ RaffOk'   = ContratClientC(Choix',Param',EtatC)
```



Vérification du raffinement d'actions du module

```

ActionModule ==
  /\ Tour    = "module"
  /\ Tour'   = "client"
  /\ Choix'  = "__NO_DATA"
  /\ Param'  = "__NO_DATA"
  /\ ContratModuleC(Choix,Param,EtatC,EtatC',Result')
  /\ LET EtatA_prime ==
      { etatA_p \in Liaison(EtatC') :
        ContratModuleA(Choix,Param,EtatA,etatA_p,Result') }
  IN IF (EtatA_prime # {})
      THEN /\ RaffOk' = TRUE
           /\ EtatA' \in EtatA_prime
      ELSE /\ RaffOk' = FALSE
           /\ EtatA' = "__NO_DATA"

```

Système de transitions et propriétés

```
Next == ActionClient \/ ActionModule
```

```
Vars == <<Tour,Choix,Param,EtatA,EtatC,Result,RaffOk>>
```

```
Spec ==
```

```
 /\ Initial
```

```
 /\ [] [ Next ]__Vars
```

```
\* spécification du raffinement:
```

```
\* l'invariant de liaison est maintenu
```

```
RaffinementOk == [] RaffOk
```

Pour vérifier le raffinement, il reste à définir :

- Un module principal TLA : vérificateur + module abstrait + module concret.
- Son fichier de configuration : S.T. + propriété de raffinement.

Plan

- 1 Introduction
- 2 Implantation des modules en TLA
- 3 Sémantique d'exécution en TLA
- 4 Exemple : le jeu de Marienbad
 - Exemple d'exécution du module Marienbad
- 5 Définitions et concepts préliminaires
- 6 Raffinement de modules
 - Définition
- 7 Implantation du raffinement en TLA
- 8 Exemple : stratégie gagnante au jeu de Marienbad
 - Exemples de vérification du raffinement de Marienbad
- 9 Patrons de raffinement de module



Stratégie gagnante

- La stratégie est de conserver un nombre d'allumettes tel que $tas \% 4 = 1$ à chaque coup (= 2 demi-coups).
- On assouplit les règles du client (il peut passer son tour).
- On restreint les règles du module (un seul coup possible).
- Dans l'absolu, le module raffiné peut réaliser un coup impossible auparavant.
- Néanmoins, si le client se plie aux anciennes règles, le module y sera conforme également.



Stratégie gagnante sous forme TLA

```

----- MODULE strategie -----
ETAT == 1..21
Init(etat) == etat = 21
-----

Pre_jouer(param, etat) ==
  \E a \in 0..3 : etat - a > 0 /\ param = a

Act_jouer(param, etat, etat_p, result) ==
  LET a == IF param = 0 THEN 0 ELSE 4 - param IN
  /\ etat - param - a > 0
  /\ etat_p = etat - param - a
  /\ result = "__NO_DATA"

Pre_perdu(param, etat) == param = 0 /\ etat = 1
Act_perdu(param, etat, etat_p, result) ==
  /\ etat_p = etat /\ result = "__NO_DATA"

```

Stratégie gagnante sous forme TLA

- Les contrats clients et modules sont génériques.
- Même structure dans tous les modules.

```
\* CONTRAT CLIENT
```

```
ContratClient(choix, param, tas) ==
```

```
  /\ (choix="jouer" /\ Pre_jouer(param,tas))
```

```
  /\ (choix="perdu" /\ Pre_perdu(param, etat))
```

```
-----
```

```
\* CONTRAT MODULE
```

```
ContratModule(choix, param, tas, tas_prime, result) ==
```

```
  /\ (choix="jouer" => Act_jouer(param,tas,tas_prime,result))
```

```
  /\ (choix="perdu" => Act_perdu(param,etat,etat_p,result))
```

```
=====
```



Exécution du module

- Tout d'abord, on vérifie les propriétés du module seul.
- Module principal `run_strategie.tla`.

```

----- MODULE run_stratégie -----
EXTENDS var_module, contrats_strategie
INSTANCE run_module
Gagnant ==
  ([] (Tour = "module" => Param /= 0))
=> [] (Tour = "module" => Etat - Param /= 1)
=====

```

- Fichier de configuration associé `run_strategie.cfg`.

```

/* On vérifie les propriétés du module, i.e :
/* - la stratégie est gagnante
/* - il n'y pas de deadlock
SPECIFICATION Spec
PROPERTIES Gagnant

```

Exécution du raffinement

- Module principal `run_marienbad_strategie.tla`.

```


----- MODULE run_marienbad_strategie -----
EXTENDS var_raffinement

A == INSTANCE marienbad
C == INSTANCE strategie

Liaison(etatC) == { etatC }

INSTANCE run_raffinement
WITH InitA <- A!Init, ContratClientA <- A!ContratClient,
     ContratModuleA <- A!ContratModule,
     InitC <- C!Init, ContratClientC <- C!ContratClient,
     ContratModuleC <- C!ContratModule
=====

```



Exécution du raffinement

- Fichier de configuration `run_marienbad_strategie.cfg`.
- On ne vérifie qu'une propriété : le raffinement.

```
\* On vérifie le raffinement du module marienbad  
\* par le module stratégie, i.e :  
\* - RaffinementOk
```

```
SPECIFICATION Spec  
PROPERTIES RaffinementOk
```

On lance le model checker

```
% java -jar tla2tools.jar run_marienbad_strategie  
[...]  
Model checking completed. No error has been found.  
[...]
```



Plan

- 1 Introduction
- 2 Implantation des modules en TLA
- 3 Sémantique d'exécution en TLA
- 4 Exemple : le jeu de Marienbad
 - Exemple d'exécution du module Marienbad
- 5 Définitions et concepts préliminaires
- 6 Raffinement de modules
 - Définition
- 7 Implantation du raffinement en TLA
- 8 Exemple : stratégie gagnante au jeu de Marienbad
 - Exemples de vérification du raffinement de Marienbad
- 9 Patrons de raffinement de module



Raffinement de module

Quelques raffinements classiques de module, i.e. :

- Ajout de variable.
 - Retrait de variable auxiliaire.
 - À variables constantes :
 - Renforcement des états initiaux.
 - Renforcement des actions du module.
 - Affaiblissement des actions du client.
 - Ajout de procédure.
 - Cas particulier : raffinement de procédure / de code.
 - Il existe d'autres raffinements plus généraux.
- Raffinement de données (par ex. ensemble par liste).
Ne s'exprime pas par des règles simples.

