

**Atelier B**

# **Animateur**

**Manuel Utilisateur**

**version 3.6**



ATELIER B  
Animateur Manuel Utilisateur  
version 3.6

Document établi par CLEARSY.

Ce document est la propriété de CLEARSY et ne doit pas être copié, reproduit, dupliqué  
totalement ou partiellement sans autorisation écrite.

Tous les noms des produits cités sont des marques déposées par leurs auteurs respectifs.

CLEARSY  
Maintenance ATELIER B  
Parc de la Duranne - 320 avenue Archimède  
Les Pléiades III - Bat A  
13857 AIX EN PROVENCE CEDEX 3  
France

Tél 33 (0)4 42 37 12 70

Fax 33 (0)4 42 37 12 71

email : [contact@atelierb.eu](mailto:contact@atelierb.eu)

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Lancement de l'Animateur</b>	<b>3</b>
<b>3</b>	<b>Initiation par l'exemple</b>	<b>5</b>
<b>4</b>	<b>Description des commandes</b>	<b>15</b>



# Chapitre 1

## Introduction

L'Animateur de l'Atelier B est un outil de validation de machines abstraites B. En animant une opération, il est possible de voir évoluer les variables et les propriétés de la spécification, et de vérifier qu'elles prennent bien les valeurs attendues. Les prédicats peuvent être prouvés grâce au Prouveur Automatique de l'Atelier B. Certaines fonctionnalités permettent d'organiser des campagnes de tests, en jouant de manière automatique des animations précédemment enregistrées.

Ce document est construit de la manière suivante :

- Le chapitre 2 décrit la manière de lancer l'Animateur.
- Le chapitre 3 présente, au moyen d'un exemple concret, ce qu'est une animation.
- Le chapitre 4 décrit les différentes commandes de l'Animateur, leurs conditions d'utilisation et leurs effets.



## Chapitre 2

# Lancement de l'Animateur

L'Animateur se lance uniquement à partir de l'Atelier B en mode commandes.

Après avoir ouvert un projet il faut taper : `ani nom_machine`.

Le paramètre `nom_machine` est le nom de l'une des machines abstraites que l'on souhaite animer.

L'Atelier B vérifie que la machine a bien été passée au Contrôleur de types et fait appel à l'Animateur. Celui-ci charge la machine et affiche le prompt `ANI>`. L'animation peut commencer.



## Chapitre 3

# Initiation par l'exemple

Ce chapitre présente un exemple de modélisation d'un train. Une première machine nommée `basic_train` modélise le comportement de ce train. Une seconde machine nommée `train` modélise une gestion possible de sa vitesse.

Un scénario est proposé afin de mettre en évidence les différentes étapes d'une animation de ces spécifications. Il présente de manière pratique les différentes fonctionnalités de l'Animateur. Le but de ce scénario est de mettre le train dans une situation critique : le train qui le précède s'arrête, et la distance entre eux deux devient nulle. Il permet de vérifier que dans ce cas la collision est correctement évitée.

### 3.1 Spécification d'un train

MACHINE

```
/* Modélisation du comportement d'un train */  
basic_train
```

CONSTANTS

```
freinage_urgence,  
accel_max
```

PROPERTIES

```
freinage_urgence : NAT1 &  
accel_max : NAT &  
accel_max <= freinage_urgence &  
freinage_urgence + accel_max <= MAXINT
```

VARIABLES

```
vitesse,  
/* vitesse courante du train */  
distance  
/* distance avec le train de devant */
```

INVARIANT

```
vitesse : NATURAL &  
distance : INTEGER
```

INITIALISATION

vitesse := 0 || distance : : NATURAL

OPERATIONS

pilote(acceleration) =

/\* Cette opération permet de piloter le train en lui imposant une accélération de la valeur du paramètre. Cette opération est supposée être appelée toutes les secondes.\*/

PRE

acceleration : INT &  
acceleration <= accel\_max

THEN

ANY nouv\_vit\_dev WHERE nouv\_vit\_dev : NATURAL THEN

vitesse, distance := max(0,vitesse + acceleration), distance + nouv\_vit\_dev - max(0,vitesse+acceleration)

END

END;

vit\_cour, dist\_cour < -- mesure =

/\* Cette opération permet de connaître la vitesse courante du train et sa distance par rapport au train devant.\*/

vit\_cour, dist\_cour := vitesse, min(distance,MAXINT)

END

## 3.2 Spécification de la gestion de la vitesse du train

MACHINE

/\* Modélisation de la gestion de la vitesse du train \*/

train

INCLUDES

basic\_train

INVARIANT

distance >= 0

OPERATIONS

cycle =

/\* Cette opération est appelée toutes les secondes. Elle gère la vitesse du train. \*/

ANY nouv\_accel WHERE

nouv\_accel : INT &  
nouv\_accel <= accel\_max &

!nouv\_vit\_dev.(nouv\_vit\_dev : NATURAL =>

distance + nouv\_vit\_dev - max(0,vitesse + nouv\_accel) >= 0)

THEN

```

    pilote(nouv_accel)
END
END

```

### 3.3 Initialiser le contexte de l'animation

L'Animateur a été lancé avec le paramètre `train`. La machine `train` a donc déjà été chargée, ainsi que celle dont elle dépend, i.e. `basic_train` (cf chapitre 2).

L'animation se déroule de la manière suivante :

- Précisons immédiatement dans quel fichier enregistrer les résultats, afin de ne pas les perdre.

```
ANI> svr("/usr2/asd/CG/TESTANI/train/res_train")
```

- Les données importantes doivent être sélectionnées. Elles seront affichées systématiquement, ce qui permettra de suivre l'évolution du projet au cours de l'animation.

```
ANI> dyv(basic_train, vitesse)
    vitesse == (vitesse$0)
ANI> dyv(basic_train, distance)
    distance == (distance$0)
```

Ces commandes effectuent un premier affichage des variables.

Remarquons ici `vitesse$0` et `distance$0`. Le `$0` indique la valeur initiale de la variable. Ainsi, si `vitesse` n'est pas initialisée, l'Animateur travaillera en considérant `vitesse$0` comme sa valeur initiale.

- L'étape suivante consiste à initialiser les constantes du projet.

```
ANI> stc(basic_train, freinage_urgence, 200)
Display for machine : basic_train
    vitesse == (vitesse$0)
    distance == (distance$0)
ANI> stc(basic_train, accel_max, 100)
Display for machine : basic_train
    vitesse == (vitesse$0)
    distance == (distance$0)
```

- On vérifie rapidement - au moyen de la commande `shp` - que la partie critique des propriétés n'est pas trivialement fausse. Puis la preuve est lancée sur toutes les propriétés et contraintes du projet.

```
ANI> shp(basic_train, accel_max <= freinage_urgence)
    (accel_max<=freinage_urgence) == (btrue)
ANI> prc
Do you want to prove properties of machine basic_train ?
ANI> pr
Properties of machine basic_train are true
```

- Les machines `train` et `basic_train` sont ensuite initialisées.

```
ANI> ini(train)
Display for machine : basic_train
    vitesse == (vitesse$0)
```

```

vitesse will be (0)
distance == (distance$0)
Animation is stopped in machine basic_train initialisation.
Set variable distance such as distance: NATURAL

```

L'initialisation de `basic_train` s'arrête sur la substitution indéterministe : `distance :: NATURAL`. L'Animateur affiche les valeurs courantes des affichages permanents. L'affichage de `vitesse` est particulier : deux valeurs sont présentées. En effet, on se trouve dans une substitution parallèle. La nouvelle valeur de `vitesse` a déjà été calculée : 0. Toutefois dans la branche actuelle, on voit encore l'ancienne valeur : `vitesse$0`.

- L'animation continue avec l'affectation d'une valeur à `distance`.

```

ANI> stv(basic_train, distance, 90)
(distance: NATURAL) == (btrue)

```

L'affichage du prédicat nous confirme qu'il est trivialement vrai, l'animation continue.

```

ANI> cont
Display for machine : basic_train
vitesse == (0)
distance == (90)

```

- L'initialisation des variables est terminée. L'affichage nous permet de contrôler leur valeurs. Il est possible de demander à l'Animateur de prouver que l'invariant est bien établi.

```

ANI> pri
Do you want to prove invariant of machine basic_train ?
ANI> pr
Invariant of machine basic_train is true
Do you want to prove invariant of machine train ?
ANI> pr
Invariant of machine train is true

```

### 3.4 Animer l'opération cycle

L'animation de l'opération cycle permet de valider la bonne utilisation de basic\_train par train.

- L'animation de l'opération cycle débute.

```
ANI> ani(train, cycle)
Display for machine : basic_train
  vitesse == (0)
  distance == (90)
Animation is stopped in machine train, operation cycle.
Set variable nouv_accel such as
  nouv_accel<=2147483647
& 0<=2147483647+nouv_accel
& nouv_accel<=100
& !nouv_vit_dev.(0 <= nouv_vit_dev =>
0<=90+nouv_vit_dev-max({0,nouv_accel}))
```

- L'animation stoppe sur la substitution ANY de l'opération cycle. Cette substitution définit la variable locale nouv\_accel qui doit vérifier le prédicat suivant :

```
(nouv_accel : INT & nouv_accel <= accel_max
& nouv_accel<=100 & !nouv_vit_dev.(0 <= nouv_vit_dev =>
distance + nouv_vit_dev - max({0,vitesse + nouv_accel}) >= 0))
Une étude hâtive de ce prédicat retient seulement la partie nouv_accel<=100. Il est
alors tentant de valuer la variable locale avec 100.
```

```
ANI> stv(train, nouv_accel, 100)
  nouv_accel<=2147483647
  & 0<=2147483647+nouv_accel
  & nouv_accel<=100
  & !nouv_vit_dev.(0 <= nouv_vit_dev =>
0<=90+nouv_vit_dev-max({0,nouv_accel}))
==
  (!nouv_vit_dev.(0 <= nouv_vit_dev => 10<=nouv_vit_dev))
Do you want to prove this predicate ?
```

L'Animateur affiche la simplification du prédicat après valuation de la variable locale. Il est évident que notre valuation a été trop rapide et qu'elle ne vérifie pas la fin du prédicat.

- La commande cont signifie à l'Animateur qu'il ne doit pas tenir compte du prédicat. Il est ensuite possible de valuer nouv\_accel avec une valeur correcte, par exemple 90.

```
ANI> cont
ANI> stv(train, nouv_accel, 90)
  nouv_accel<=2147483647
  & 0<=2147483647+nouv_accel
  & nouv_accel<=100
  & !nouv_vit_dev.(0 <= nouv_vit_dev =>
0<=90+nouv_vit_dev-max({0,nouv_accel}))
==
  (!nouv_vit_dev.(0 <= nouv_vit_dev => 0<=nouv_vit_dev))
Do you want to prove this predicate ?
```

- Cette fois le prédicat semble juste. Après l'avoir vérifié en utilisant le prouveur, la com-

mande `cont` signifie à l'Animateur que l'instanciation des variables locales est terminée. Il continue donc l'animation du corps de l'opération.

```
ANI> pr
This predicate is true
ANI> cont
Operation pilote precondition :
  (acceleration: INT & acceleration<=accel_max) == (btrue)
Display for machine : basic_train
  vitesse == (0)
  distance == (90)
Animation is stopped in machine basic_train, operation pilote.
Set variable nouv_vit_dev such as 0 <= nouv_vit_dev
```

L'opération `pilote` de la machine `basic_train` est appelée. L'Animateur affiche sa précondition. La simplification de celle-ci aboutit à `btrue`. L'animation continue donc, et s'arrête sur la substitution `ANY`.

- La nouvelle variable locale `nouv_vit_dev` doit vérifier `nouv_vit_dev: NATURAL`. Sa valeur peut donc par exemple être 0.

```
ANI> stv(basic_train, nouv_vit_dev, 0)
  (0 <= nouv_vit_dev) == (btrue)
ANI> cont
End of operation cycle animation
Display for machine : basic_train
  vitesse == (90)
  distance == (0)
Do you want to prove invariant of machine train ?
```

- L'animation du corps de l'opération est terminée. L'Animateur propose de prouver la préservation de l'invariant. Après avoir vérifié que celui-ci est vrai, on anime à nouveau l'opération `cycle`. L'affichage permanent indique en effet que la distance entre les deux trains est nulle, alors que celui de derrière a une vitesse de 90. Il est donc intéressant de vérifier que même dans un cas aussi extrême, la collision n'a pas lieu.

```
ANI> pr
Invariant of machine train is true
ANI> ani(train, cycle)
Display for machine : basic_train
  vitesse == (90)
  distance == (0)
Animation is stopped in machine train, operation cycle.
Set variable nouv_accel such as
  nouv_accel<=2147483647 &
0<=2147483647+nouv_accel & nouv_accel<=100 &
!nouv_vit_dev.(nouv_vit_dev: NATURAL =>
max({0,90+nouv_accel})<=nouv_vit_dev)
```

- L'animation de l'opération `cycle` est à nouveau stoppée par la substitution `ANY`. Pour ne pas entrer en collision avec le train précédent, il est nécessaire de ne pas accélérer. Une vitesse nulle est, par exemple, possible.

```
ANI> stv(train, nouv_accel, 0)
  nouv_accel<=2147483647
  & 0<=2147483647+nouv_accel
```

```

& nouv_accel<=100 &
!nouv_vit_dev.(0 <= nouv_vit_dev =>
max({0,90+nouv_accel})<=nouv_vit_dev))
==
(!nouv_vit_dev.(0 <= nouv_vit_dev => 90<=nouv_vit_dev))
Do you want to prove this predicate ?
ANI> pr
Cannot prove this predicate

```

- La preuve échoue, et effectivement si `nouv_vit_dev` est inférieur strictement à 90, la propriété n'est pas vérifiée. Il est en fait nécessaire que `nouv_accel` soit inférieure ou égale à -90.

```

ANI> stv(train, nouv_accel, -90)
nouv_accel<=2147483647
& 0<=2147483647+nouv_accel
& nouv_accel<=100 &
!nouv_vit_dev.(0 <= nouv_vit_dev =>
max({0,90+nouv_accel})<=nouv_vit_dev))
==
(!nouv_vit_dev.(0 <= nouv_vit_dev => 0<=nouv_vit_dev))
Do you want to prove this predicate ?
ANI> pr
This predicate is true
ANI> cont
Operation pilote precondition :
(acceleration: INT & acceleration<=accel_max) == (btrue)
Display for machine : basic_train
vitesse == (90)
distance == (0)
Animation is stopped in machine basic_train, operation pilote.
Set variable nouv_vit_dev such as 0 <= nouv_vit_dev

```

- L'animation est à nouveau stoppée sur la substitution ANY de l'opération appelée `pilote`. La variable locale `nouv_vit_dev` doit être évaluée dans NATURAL. Dans le pire des cas, le train de devant reste arrêté, c'est-à-dire `nouv_vit_dev` vaut 0. Cette variable est instanciée, et l'animation de l'opération se termine.

```

ANI> stv(basic_train, nouv_vit_dev, 0)
(0 <= nouv_vit_dev) == (btrue)
ANI> cont
End of operation cycle animation
Display for machine basic_train :
vitesse == (0)
distance == (0)
Do you want to prove invariant of machine train ?
ANI> pr
Invariant of machine train is true

```

L'affichage permanent nous permet de contrôler que la distance entre les deux trains n'est pas devenue négative : la collision n'a pas eu lieu. Cette animation nous a confirmé que la machine train utilise correctement `basic_train` : le prédicat de la substitution ANY impose à la nouvelle accélération de tenir compte de toutes les nouvelles vitesses possibles pour le train de devant. Il est donc impossible d'avoir une collision.

### 3.5 Animer l'opération mesure

Une dernière animation possible est celle de l'opération `mesure` de la machine `basic_train`. Il est en effet possible d'animer les opérations de toutes les machines chargées, et non pas seulement celles de la machine source.

```
ANI> ani(basic_train, mesure)
List of output parameters :
  vit_cour == (0)
  dist_cour == (0)
End of operation mesure animation
Display for machine basic_train :
  vitesse == (0)
  distance == (0)
Do you want to prove invariant of machine basic_train ?
ANI> as
```

Après l'animation du corps de l'opération, les valeurs des paramètres de sortie sont affichées. L'invariant est admis : cette opération ne modifie pas les valeurs des variables.

### 3.6 Enregistrer et terminer une animation

Avant de terminer l'animation, il est intéressant d'enregistrer la liste des commandes utilisées afin de pouvoir la rejouer de manière automatique.

```
ANI> sva("/usr2/asd/CG/TESTANI/train/ani_train")
ANI> qu
```

### 3.7 Fichier d'animation

Le contenu du fichier d'enregistrement de la liste des commandes du scénario présenté est le suivant :

```
THEORY Commands_Anim IS
ldm(train);
svr("/usr2/asd/CG/TESTANI/train/res_train");
dyv(basic_train, vitesse);
dyv(basic_train, distance);
stc(basic_train, freinage_urgence, 200);
stc(basic_train, accel_max, 100);
shp(basic_train, accel_max <= freinage_urgence);
prc;
pr;
ini(train);
stv(basic_train, distance, 90);
cont;
pri;
pr;
pr;
ani(train, cycle);
stv(train, nouv_accel, 100);
cont;
stv(train, nouv_accel,90);
pr;
cont;
stv(basic_train, nouv_vit_dev, 0);
cont;
pr;
ani(train, cycle);
stv(train, nouv_accel, 0);
pr;
stv(train, nouv_accel, -90);
pr;
cont;
stv(basic_train, nouv_vit_dev, 0);
cont;
pr;
```

```
ani(basic_train, mesure);  
as  
END
```

Ce fichier permet de rejouer de manière automatique le scénario décrit. Il peut être modifié à la main.

### 3.8 Fichier d'enregistrement des résultats

Un extrait du fichier d'enregistrement des résultats du scénario est présenté ci-dessous :

```
STOP IN MACHINE basic_train, OPERATION pilote AT INDETERMINISTIC SUBSTITUTION :  
nouv_vit_dev:: (nouv_vit_dev: NATURAL)  
Display for machine basic_train :  
  vitesse == (0)  
  distance == (90)  
  
OPERATION cycle RESULT  
Display for machine basic_train :  
  vitesse == (90)  
  distance == (0)
```

## Chapitre 4

# Description des commandes

Ce chapitre décrit les différentes commandes de l'Animateur. Ces commandes sont classées par fonctionnalité :

- Initialiser l'Animateur
- Animer une opération
- Animer une substitution indéterministe
- Prouver un prédicat
- Afficher les résultats
- Enregistrer des animations

Chaque commande possède une forme complète et une forme réduite. L'une ou l'autre peut être indifféremment utilisée.

Dans la suite, les termes "mode preuve" sont utilisés. L'Animateur est en mode preuve lorsque l'une des commandes `AssumeCtxt`, `AssumeInv`, `ProveCtxt` ou `ProveInv` a été utilisée. Dans ce cas toutes les simplifications et preuves ont pour hypothèse les invariants, contraintes et propriétés prouvés ou admis.

## 4.1 Initialiser l'Animateur

### 4.1.1 Liste des commandes nécessaires

<i>Commande complète</i>	<i>Réduction</i>	<i>Description</i>
<code>AssumeContext</code>	<code>asc</code>	Admettre les propriétés et les contraintes du projet
<code>AssumeInvariant</code>	<code>asi</code>	Admettre les invariants du projet
<code>Init(M)</code>	<code>ini(M)</code>	Initialiser la machine M
<code>LoadMachines(M)</code>	<code>ldm(M)</code>	Charger une machine et toutes celles dont elle dépend
<code>ProveContext</code>	<code>prc</code>	Prouver les propriétés et les contraintes du projet
<code>ProveInvariant</code>	<code>pri</code>	Prouver les invariants du projet
<code>SetCompParameter(M,p,V)</code>	<code>stp(M,p,E)</code>	Affecter la valeur de l'expression E au paramètre p de la machine M
<code>SetConstants(M,c,V)</code>	<code>stc(M,c,E)</code>	Affecter la valeur de l'expression E à la constante ou à l'ensemble c de la machine M
<code>SetVariable(M,v,V)</code>	<code>stv(M,v,E)</code>	Affecter la valeur de l'expression E à la variable v de la machine M

### 4.1.2 Charger les machines à animer

La commande `LoadMachines(M)` charge la machine M et toutes les machines dont dépend M. Il suffit donc de charger la racine du projet que l'on souhaite animer.

Il est inutile de charger la machine passée en paramètre de la commande lançant l'Animateur. Cette commande est automatiquement effectuée.

### 4.1.3 Initialiser les constantes, les ensembles, les paramètres et les variables du projet

Les commandes suivantes initialisent les différents éléments du projet :

`SetConstant(M,c,E)`, `SetCompParameter(M,p,E)`, `SetVariable(M,v,E)` et `Init(M)`.

L'expression E passée en paramètre de ces commandes peut être complexe. Les variables non libres sont éventuellement évaluées, puis l'expression est simplifiée avant d'être affectée à l'identificateur cible.

La commande `SetConstant` permet de valuer aussi bien les constantes que les ensembles abstraits des machines chargées.

La commande `Init(M)` initialise les machines dont dépend M, puis M.

La commande `SetVariable(M,v,E)` permet de changer l'initialisation des variables prévue dans les sources.

L'initialisation n'est pas obligatoire ; il est possible de travailler de manière symbolique. Dans ce cas la valeur initiale d'une variable, ou la valeur d'une constante est représentée en concaténant son nom avec \$0.

### 4.1.4 Initialiser les hypothèses de l'animation

Lorsque l'on souhaite privilégier la preuve dans son animation, il est intéressant d'avoir en hypothèse les propriétés, contraintes et invariants de son projet. Ceux-ci seront mis

à jour au fur et à mesure des évolutions des variables. Ainsi, une simplification ou une preuve aura toujours comme hypothèse l'invariant instancié avec les dernières valeurs des variables pour lesquelles il a été prouvé.

Les commandes `ProveContext` et `AssumeContext` permettent de prouver ou d'admettre que la valuation des constantes, ensembles et paramètres vérifie les propriétés et contraintes associées.

Les commandes `ProveInvariant` et `AssumeInvariant` permettent de prouver ou d'admettre que l'initialisation des variables établit les invariants associés.

Pour les commandes `ProveContext` et `ProveInvariant`, l'Animateur gère l'ordre des preuves. Il assure ainsi que chacune d'entre elles a les hypothèses nécessaires. Dans cette optique, il est obligatoire de traiter les propriétés et les contraintes avant les invariants.

Ces quatre commandes font passer l'Animateur en mode preuve : les prédicats prouvés ou admis seront utilisés en hypothèse des simplifications et preuves qui suivront.

Notons que les commandes `ProveContext` et `ProveInvariant` s'arrêtent sur chaque machine et propose à l'utilisateur de la prouver ou non. Il est ainsi possible de ne prouver que les machines intéressantes.

## 4.2 Animer une opération

### 4.2.1 Liste des commandes nécessaires

<i>Commande complète</i>	<i>Réduction</i>	<i>Description</i>
<code>Anim(M, o)</code>	<code>ani(M, o)</code>	Animer l'opération sans paramètre <code>o</code> de la machine <code>M</code>
<code>Anim(M, o, (L))</code>	<code>ani(M,o,(L))</code>	Animer l'opération <code>o</code> de la machine <code>M</code> avec la liste de paramètres effectifs <code>L</code>

### 4.2.2 Traiter les préconditions

La précondition de l'opération est évaluée et simplifiée. Le résultat est affiché à l'écran. Si la simplification ne se résume pas à `btrue`, l'Animateur propose de la prouver. Ce dialogue et les réponses possibles sont décrits dans le paragraphe 4.4.

Si la preuve réussit ou est admise et si l'Animateur est en mode preuve, la précondition est utilisée en hypothèse de toutes les simplifications et preuves effectuées pendant l'animation du corps de l'opération.

La commande `Continue` permet d'animer une opération en brisant les préconditions. En effet le corps de l'opération est animé sans la précondition en hypothèse : cela évite d'avoir une hypothèse fausse qui rendrait toutes les preuves triviales.

Lorsque la preuve échoue, ou lorsque la commande `Cancel` est utilisée, le corps de l'opération n'est pas animé.

### 4.2.3 Animer le corps de l'opération

Dans le cas de structures de contrôle - IF, SELECT, CASE - l'Animateur évalue les différentes conditions à l'aide du Prouveur. Lorsque ces simplifications ne suffisent pas

pour savoir si une condition est vraie ou non, l'utilisateur doit en indiquer la valeur.

Dans le cas des substitutions indéterministes, un dialogue avec l'utilisateur permet à l'Animateur de décider quelle substitution effectuer. Ce dialogue est décrit au paragraphe 4.3. En fin d'animation, les valeurs des paramètres de sortie sont affichées.

#### 4.2.4 Prouver la préservation de l'invariant

Après avoir animé une opération, et éventuellement affiché les valeurs des paramètres de sortie, l'Animateur propose à l'utilisateur de prouver la préservation de l'invariant. Ce dialogue et les réponses possibles sont décrits dans le paragraphe 4.4.

Si l'invariant est prouvé ou admis et si l'Animateur est en mode preuve, sa nouvelle valeur - c'est-à-dire l'invariant instancié avec les nouvelles valeurs des variables - sera utilisée en hypothèse de la suite de l'animation. Si l'Animateur n'est pas en mode preuve, cette preuve est seulement faite à titre d'information et n'a aucune influence sur la suite de l'animation.

La commande `Continue`, ou une preuve évaluant l'invariant à faux, fait continuer l'animation avec les mêmes hypothèses. Les variables gardent la valeur que l'opération leur a éventuellement affectée.

La commande `Cancel` annule les effets de l'opération : les variables ont à nouveau les valeurs qu'elles avaient avant l'animation de l'opération.

### 4.3 Animer une substitution indéterministe

#### 4.3.1 Liste des commandes nécessaires

<i>Commande complète</i>	<i>Réduction</i>	<i>Description</i>
<code>Continue</code>	<code>cont</code>	Continuer l'animation
<code>SetSubstitution(S)</code>	<code>sts(S)</code>	Animer la substitution <code>S</code> à la place de la substitution indéterministe affichée
<code>SetVariable(M, v, V)</code>	<code>stv(M, v, V)</code>	Affecter la valeur <code>V</code> à la variable <code>v</code> de la machine <code>M</code>

#### 4.3.2 Substitutions CHOICE et SELECT

Lorsque l'Animateur stoppe sur une substitution indéterministe, il l'affiche et demande à l'utilisateur une substitution à animer à la place. Dans le cas de la substitution `SELECT`, seules les branches dont les conditions ont été évaluées à `true` sont affichées.

La réponse attendue est la commande `SetSubstitution(S)`. La substitution `S` est alors animée.

#### 4.3.3 Substitutions ANY, devient tel que, devient appartient

L'Animateur affiche les variables à instancier et le prédicat qu'elles doivent vérifier.

La commande `SetVariable(M, v, V)`, permet d'instancier les variables de son choix. Le prédicat simplifié est ensuite affiché et l'Animateur propose de le prouver - ce dialogue est

décrit au paragraphe 4.4. Une même variable peut être instanciée plusieurs fois, afin de visualiser le prédicat dans les différents cas. Il est également possible de ne pas instancier toutes les variables, l'animation continue alors avec des variables symboliques.

La commande `Continue` précise à l'Animateur que l'instanciation des variables est terminée. L'animation peut continuer.

## 4.4 Prouver un prédicat

### 4.4.1 Liste des commandes nécessaires

<i>Commande complète</i>	<i>Réduction</i>	<i>Description</i>
<b>Assume</b>	<b>as</b>	Admettre que le prédicat est vrai
<b>Cancel</b>	<b>cl</b>	Annuler la commande en cours
<b>Continue</b>	<b>cont</b>	Continuer l'animation sans tenir compte du prédicat
<b>Prove</b>	<b>pr</b>	Appeler le prouveur en force 0

### 4.4.2 Influence des différentes réponses sur la suite de l'animation

La commande **Prove** lance la preuve en force 0. Si le Prouveur échoue, l'Animateur demande à l'utilisateur de lui préciser la valeur à choisir : **true** ou **false**.

Si le prédicat est prouvé ou admis - commande **Assume** - et si l'Animateur est en mode preuve, ce prédicat est utilisé comme hypothèse dans la suite de l'animation.

La commande **Continue** indique à l'Animateur de continuer sans tenir compte du prédicat : il est supposé vrai mais n'est pas utilisé comme hypothèse dans la suite de l'animation.

La commande **Cancel** ou un échec de la preuve fait continuer l'animation avec les mêmes hypothèses. Selon le contexte, l'animation en cours peut être interrompue.

## 4.5 Afficher les résultats

### 4.5.1 Liste des commandes nécessaires

<i>Commande complète</i>	<i>Réduction</i>	<i>Description</i>
DisplayConst(M)	dyc(M)	Afficher en permanence la valeur simplifiée des contraintes de la machine M
DisplayExp(M, e)	dye(M, e)	Afficher en permanence la valeur simplifiée de l'expression e dont les éléments sont définis dans la machine M
DisplayInv(M)	dyi(M)	Afficher en permanence la valeur simplifiée de l'invariant de la machine M
DisplayPred(M, p)	dyp(M, p)	Afficher en permanence la valeur simplifiée du prédicat p dont les éléments sont définis dans la machine M
DisplayProp(M)	dypp(M)	Afficher en permanence la valeur simplifiée des propriétés de la machine M
DisplayVar(M, v)	dyv(M, v)	Afficher en permanence la valeur simplifiée de l'entité v définie dans la machine M
ShowConst(M)	shc(M)	Afficher la valeur simplifiée des contraintes de la machine M
ShowExp(M, e)	she(M, e)	Afficher la valeur simplifiée de l'expression e dont les éléments sont définis dans la machine M
ShowInv(M)	shi(M)	Afficher la valeur simplifiée de l'invariant de la machine M
ShowPred(M, p)	shp(M, p)	Afficher la valeur simplifiée du prédicat p dont les éléments sont définis dans la machine M
ShowProp(M)	shpp(M)	Afficher la valeur simplifiée des propriétés de la machine M
ShowVar(M, v)	shv(M, v)	Afficher la valeur simplifiée de l'entité v définie dans la machine M
UnDisplayConst(M)	udc(M)	Annuler l'affichage permanent de la valeur des contraintes de la machine M
UnDisplayExp(M, e)	ude(M, e)	Annuler l'affichage permanent de la valeur de l'expression e dont les éléments sont définis dans la machine M
UnDisplayInv(M)	udi(M)	Annuler l'affichage permanent de la valeur de l'invariant de la machine M
UnDisplayPred(M, p)	udp(M, p)	Annuler l'affichage permanent de la valeur du prédicat p dont les éléments sont définis dans la machine M
UnDisplayProp(M)	udpp(M)	Annuler l'affichage permanent de la valeur des propriétés de la machine M
UnDisplayVar(M, v)	udv(M, v)	Annuler l'affichage permanent de la valeur de l'entité v définie dans la machine M

### 4.5.2 Utilisation des commandes "Show"

L'utilisation des commandes **Show** permet une vérification ponctuelle de l'état de l'Animateur. Elle aide également à savoir si une preuve est nécessaire : il suffit d'afficher la partie critique de l'invariant pour savoir s'il est utile de lancer la preuve.

### 4.5.3 Utilisation des commandes "Display"

Ces commandes permettent l'affichage permanent de variables, prédicats ou expressions significatifs de l'état du projet. Cet affichage est effectué à chaque fois qu'un changement des valeurs s'est éventuellement produit. Ainsi, il n'est pas effectué après la commande `SaveAnim(f)`, par contre il l'est systématiquement après une commande `Anim(M, o, L)`. Pour connaître la valeur courante d'une variable dont l'affichage est permanent, il suffit donc de se reporter au dernier affichage.

L'utilisation de ces commandes permet de voir évoluer l'état du projet au cours de l'animation, et de juger rapidement des effets des opérations.

## 4.6 Enregistrer des animations

### 4.6.1 Liste des commandes nécessaires

<i>Commande complète</i>	<i>Réduction</i>	<i>Description</i>
<code>LoadAnim(f)</code>	<code>lda(f)</code>	Charger et exécuter les commandes contenues dans le fichier <code>f</code>
<code>SaveAnim(f)</code>	<code>sva(f)</code>	Enregistrer la liste des commandes utilisées précédemment dans le fichier <code>f</code>
<code>SaveResults(f)</code>	<code>svr(f)</code>	Enregistrer les futurs résultats de l'affichage permanent dans le fichier <code>f</code>

### 4.6.2 Enregistrer les résultats d'une animation

La commande `SaveResults(f)` sauve les affichages permanents dans le fichier `f`. Seuls les affichages après utilisation de cette commande sont conservés. Ceux qui la précèdent sont perdus.

Le paramètre de cette commande est le chemin d'accès au fichier entre guillemets.

### 4.6.3 Enregistrer une animation

La commande `SaveAnim(f)` sauve la liste des commandes utilisées dans le fichier `f`. Seules les commandes effectuées avant l'utilisation de cette commande sont sauvegardées.

La commande `LoadAnim(f)` réalise l'animation contenue dans le fichier `f`. Attention, si la liste de commandes contenait `SaveResults(g)`, l'enregistrement des résultats s'effectuera également dans `g`, écrasant les précédents. Il est donc nécessaire de recopier le contenu de `g` dans un autre fichier, ou de modifier directement `f` en changeant le paramètre de `SaveResults`.

Le paramètre de ces commandes est le chemin d'accès au fichier entre guillemets.

#### 4.6.4 Format du fichier d'enregistrement d'un animation

La première ligne de ce fichier est :

```
THEORY Commands_Anim IS
```

La dernière ligne de ce fichier est :

```
END
```

Chacune des lignes intermédiaires contient une commande. Toute commande, sauf la dernière, est suivie d'un point-virgule.

Si la liste des commandes est vide, ce fichier contient une unique ligne :

```
THEORY Commands_Anim END
```