

---

**Institut de Recherche  
en Informatique de Toulouse**

**CNRS (UMR 5055) - INPT - UPS**

**Lotrec**

**Démonstrateur générique de  
théorèmes**

**Manuel utilisateur et de  
référence**

Version 1

Date d'application : 15 avril 2001



---

# Table des matières

---

<b>Préface .....</b>	<b>v</b>
Objectifs.....	vi
Public cible .....	vi
Version.....	vi
Historique .....	vi
Conventions d'écriture.....	vi
Contacts .....	vii
L'IRIT .....	vii
Bibliographie .....	viii

## Chapitre 1

<b>Présentation et installation de Lotrec .....</b>	<b>1-1</b>
1.1 Généralités .....	1-2
Objectif.....	1-2
Domaine d'application.....	1-2
Exemples de réalisation .....	1-2
1.2 Installation .....	1-3
Introduction.....	1-3
Téléchargement.....	1-3
Installation sur la machine locale.....	1-3

## Chapitre 2

<b>Utilisation de Lotrec .....</b>	<b>2-1</b>
2.1 Architecture .....	2-2
2.2 Logique .....	2-3
Introduction.....	2-3
Définition des Connecteurs.....	2-3
Définition des règles de tableau .....	2-4

2.3	Stratégie .....	2-7
	Introduction .....	2-7
	Construction de la stratégie .....	2-7
	Exemple de stratégie .....	2-8
2.4	Formule à vérifier .....	2-9
	Principe .....	2-9
	Exemple .....	2-9
2.5	Déroulement d'une session Lotrec .....	2-10
	Contexte de l'exemple .....	2-10
	Construction des fichiers d'entrée.....	2-10
	Activation de Lotrec ; test de la première formule .....	2-10
	Test de la deuxième formule .....	2-12

### Chapitre 3

## ***Section de référence ..... 3-1***

3.1	Connector .....	3-2
	Syntaxe.....	3-2
	Exemple .....	3-2
3.2	Expression .....	3-3
	Introduction.....	3-3
	Variable.....	3-3
	Constante.....	3-3
	Formule avec connecteur .....	3-3
	Fonction .....	3-3
3.3	Rule .....	3-4
	Présentation.....	3-4
	Syntaxe.....	3-4
3.4	Descriptor .....	3-5
3.5	Action .....	3-6
3.6	Strategy .....	3-7
	Syntaxe dans la version 2.0.....	3-7
	Syntaxe de la version 1.0 .....	3-7
	Options.....	3-7
3.7	// (Marques de commentaire) .....	3-9

## ***Annexes ..... A-1***

A.1	Fichier K4.thy .....	A-2
A.2	Fichier K4.str .....	A-3

## ***Index ..... I-1***

---

# *Préface*

Dans ce chapitre

- 
- **Objectifs**
  - **Public cible**
  - **Version**
  - **Conventions d'écriture**
  - **Contacts**
  - **L'IRIT**
  - **Bibliographie**

---

## Objectifs

Lotrec est un démonstrateur générique de théorèmes.

---

## Public cible

Ce manuel s'adresse aux utilisateurs de l'outil Lotrec, à savoir :

- les chercheurs spécialisés dans la logique modale;
- les étudiants concernés par l'apprentissage de la logique modale.

---

## Version

La version actuelle du logiciel Lotrec est : 1.5. (13/01/2001)

Une version est en cours de développement sous le numéro : 2.0

---

## Historique

Numéro de version	Date	Statut
1.5	13/01/2001	Création
2.0	?	Modification

---

## Conventions d'écriture

La représentation des lignes de code respecte les conventions suivantes :

Représentation	Désignation
courrier	Toute ligne de code, instruction, commande,...
courrier italique	Variabes dans une ligne de code
<>	Encadrement d'une variable constituée de plusieurs mots
()	Encadrement d'un paramètre optionnel
[ ]*	Encadrement d'un paramètre pouvant se répéter plusieurs fois

---

## Contacts

Contact scientifique

Andreas.Herzig@irit.fr, Olivier.Gasquet@irit.fr

Transfert industriel

Daniel.Ventre@irit.fr - 33 (0)5 61 55 63 04

---

## L'IRIT

L'Institut de Recherche en Informatique de Toulouse (IRIT) est une unité associée au Centre National de la Recherche Scientifique (CNRS), à l'Institut National Polytechnique de Toulouse (INPT) et à l'Université Paul Sabatier (UPS).

Environ 335 personnes travaillent à l'IRIT dont 265 chercheurs et enseignants chercheurs (parmi lesquels 115 doctorants) et 78 ingénieurs, techniciens et administratifs.

Les recherches de l'IRIT couvrent l'ensemble des domaines où l'informatique se développe aujourd'hui, que ce soit dans son axe propre, de l'architecture des machines au génie logiciel et aux réseaux, comme dans son extension les plus contemporaines : intelligence artificielle et systèmes cognitifs, interaction multi-media homme-système, analyse et synthèses d'images.

L'IRIT à l'UPS

118 route de Narbonne - 31062 Toulouse Cedex 4

Tél. 05 61 55 67 65 / fax 05 61 55 62 58

Web : [www.irit.fr](http://www.irit.fr)

L'IRIT à ENSEEIHT

2 rue Camichel - 31071 Toulouse Cedex

Tél. 05 61 62 78 62 / fax 05 61 58 82 09

Web : [www.enseeiht.fr](http://www.enseeiht.fr)

---

## **Bibliographie**

- M. Castilho, L. Fariñas del Cerro, O. Gasquet & A. Herzig. *"Modal Tableaux with propagation rules and structural rules"*. **Fund Inf.** 32(3):281-297,1997.
- M. Castilho, L. Fariñas del Cerro, O. Gasquet & A. Herzig. *"Lotrec: System Description"*. Proc. IJCAR'01. Springer Verlag, LNCS,2001.
- D. Fauthoux. *Lotrec, un outil javanais de traitement formel sur les graphes* (rapport DEA). IRIT,2000.
- S.Suwanmanee. *Expérimentation et Manuel de Lotrec* (rapport DEA). IRIT;2001.

# *Présentation et installation de Lotrec*

---

Dans ce chapitre

- Généralités
- Installation

---

## 1.1 Généralités

### Objectif

Lotrec est un démonstrateur générique de théorèmes ouvert à toute logique de tendance modale. Il permet aux utilisateurs d'expérimenter et de modéliser différentes logiques et stratégies de démonstration. Comme il est développé en Java, ce logiciel fonctionne sur diverses plateformes telles que Unix ou Windows. Par rapport aux démonstrateurs classiques, Lotrec offre plusieurs avantages grâce à son aspect générique:

- une implémentation flexible et portable ;
- un langage évolué pour les règles et les stratégies de programmation ;
- une interface conviviale.

Lotrec met à disposition une bibliothèque de logiques standards mais permet également à l'utilisateur de constituer ses propres logiques.

### Domaine d'application

Lotrec est ouvert aux domaines d'application suivants :

- comme outil pédagogique pour l'enseignement de l'intelligence artificielle.
- comme prototypage rapide de langage développé pour la recherche.

### Exemples de réalisation

Lotrec a été utilisé dans les contextes suivants :

- Vérificateur de langage d'objets concurrents développés par des étudiants ;
- Prototypage rapide de démonstrateur utilisé pour des travaux pratiques.

---

## 1.2 Installation

### Introduction

L'installation se déroule en deux étapes :

- téléchargement de la machine virtuelle et des fichiers Lotrec;
- extraction des fichiers sur la machine locale.

### Téléchargement

Téléchargement de la machine virtuelle Java

La plus utilisée des machines virtuelles Java est celle de Sun Microsystem, à l'adresse suivante :

<http://java.sun.com/products/jdk/1.2/jre/index.html>

Téléchargement des fichiers Lotrec

Lotrec se charge à partir d'Internet, à l'adresse suivante :

- <http://www.irit.fr/ACTIVITES/LILaC/Lotrec/>

Les membres de LILaC peuvent exécuter directement la version courante de Lotrec dans le répertoire :

`/net/irit1/ftp+www/WWW/ACTIVITES/LILaC/Lotrec/Run`

### Installation sur la machine locale

#### ***Pour installer Lotrec***

1. Installer la machine virtuelle Java
2. Décompresser le fichier téléchargé dans le répertoire `lotrec`



---

Dans ce chapitre

- **Architecture**
- **Logique**
- **Stratégie**
- **Formule à vérifier**
- **Déroulement d'une session Lotrec**

## 2.1 Architecture

Lotrec prend en entrée trois éléments. Ce sont des fichiers décrivant la logique, la stratégie et la formule à démontrer. Pour vérifier la validité ou la satisfaisabilité de la formule, Lotrec génère le graphe associé à la démonstration par la méthode des tableaux sémantiques.

Le fonctionnement de Lotrec s'inscrit dans le schéma suivant :

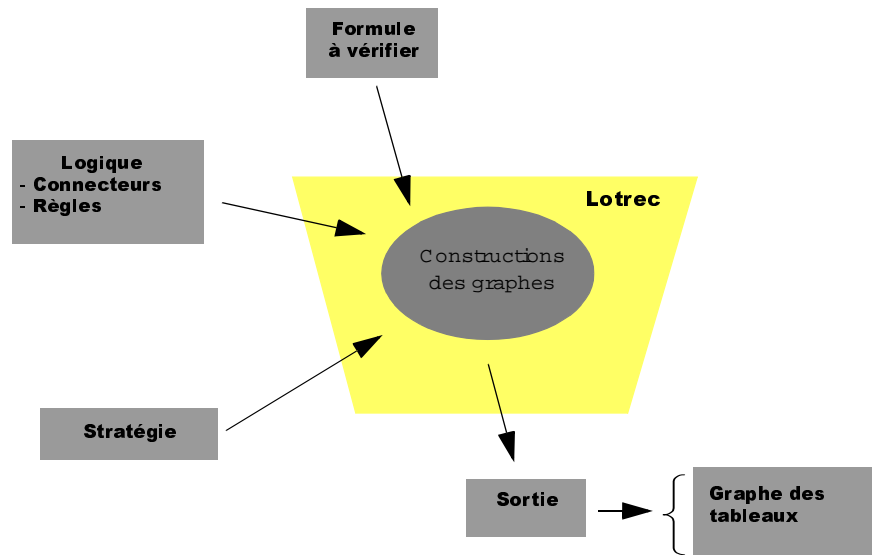


Figure 2.1 Schéma du fonctionnement de Lotrec

Les fichiers d'entrée sont des fichiers textuels édité par n'importe quel éditeur de texte en respectant un minimum de contraintes syntaxiques de Lotrec.

Le fichier avec l'extension...	contient...
.thy	la description de la logique comprenant la définition des connecteurs (le langage) et des règles de tableau
.str	la définition de la stratégie
.wff	la description de la formule à vérifier

Au niveau de la sortie, les tableaux résultant sont présentés sous forme de graphe interactif qui permet aux utilisateurs de reorganiser l'apparence du graphe.

## 2.2 Logique

### Introduction

Dans Lotrec, la description d'une logique est constituée de deux parties: des connecteurs et des règles de tableau. Ces deux sont définies dans le langage spécifique de Lotrec dans un fichier textuel (\*.thy).

### Définition des Connecteurs

En général, le langage d'une logique comprend trois éléments de base : constantes, variables et connecteurs (ou opérateurs). Dans Lotrec, on a systématiquement à disposition les constants et les variables. Pour déterminer le langage d'une logique, il suffit donc de définir des connecteurs de cette logique.

Les connecteurs sont:

- soit des opérateurs classiques tels que `and` (et, conjonction), `not` (non, négation), `or` (ou, disjonction), etc;
- soit des connecteurs non-classiques comme les opérateurs de la logique modale: `[]` (nécessaire) et `<>` (possible), les connecteurs de la logique temporelle DUX : `o` (next), `[]` (always), `U` (until), etc.

Les paramètres du connecteur

On définit les deux type de connecteurs de la même façon. Un connecteur est caractérisé par cinq paramètres:

- son nom interne c'est-à-dire le nom de connecteur que l'on utilise dans les formules ou les expressions au niveau des descriptions des règles de tableau et la formule à vérifier
- son nombre d'arguments ( 0,1,2,...)
- sa propriété d'associativité (vrai ou faux)
- sa représentation externe c'est-à-dire la façon dont on affiche le connecteur et son(ses) argument(s)
- sa priorité (0,1,2,..). Plus le chiffre est élevé, plus la priorité est importante.

Construction d'un connecteur

Un connecteur se construit de la manière suivante :

```
connector <NOM-INT> <A> <ASSOC> "EXT" <P>
```

où

- `<NOM-INT>` est le nom interne du connecteur
- `<A>` est le nombre d'arguments du connecteur
- `<ASSOC>` prend la valeur `true` si le connecteur est associatif et la valeur `false` sinon.
- `"EXT"` est la présentation de l'expression construite avec le connecteur dans le graphe de tableau. On remplace chaque argument par `_`, dans l'ordre.
- `<P>` est la priorité du connecteur par rapport aux autres. Plus le nombre est grand, plus la priorité est importante.

## 2.2 Logique, suite

Exemple

Pour mieux comprendre le fonctionnement de la définition, prenons les exemples suivants :

```
// the connectives of logic of action
connector   falsum      0   true   "FALSUM"  4
connector   and         2   true   "_ & _"   3
connector   not         1   true   "~_"      5
connector   feasible    2   true   "<_>_"    4
connector   after       2   true   "[_]_"    4
```

D'une manière générale, les connecteurs fonctionnent comme des opérateurs préfixes c'est-à-dire que le connecteur précède toujours son (ses) argument(s). Les parenthèses sont optionnelles. L'expression de la forme (and constant P not constant Q), s'affichera dans le tableau sorti comme Q & ~P. De la même façon, l'expression interne ;

(after (constant hit) (feasible(constant smash)(constant broken))  
correspond à [hit]<smash>broken à la sortie.

Comme la priorité de and est moins importante que celle de after, la formule [a]p&q correspond à ([a]p)&q (et non à [a](p&q)).

A noter que le commentaire est précédé par //. Ceci est applicable dans tous les fichiers Lotrec.

Définition des règles de tableau

Les règles de tableau fonctionnent comme les règles d'inférence. Comme dans la méthode des tableaux sémantiques, les règles de tableau décrivent comment le tableau évolue. Une règle de tableau est constituée de deux parties: « description » et « action ». La partie « description » (matérialisée par la notion « descriptor ») contient les conditions nécessaires pour que la règle soit applicable et la partie « action » décrit les opérations qui devront s'effectuer si la règle est applicable.

## 2.2 Logique, suite

Syntaxe

Au niveau de la syntaxe, une règle de tableau est définie de la façon suivante:

```
rule "<nom-règle>"
  descriptor <desc-1>
    ...
  descriptor <desc-m>

  action <act-1>
    ...
  action <act-n>
end
```

où

- <desc-1>...<desc-m> sont les conditions qui doivent être vérifiées avant d'appliquer la règle. Il y a plusieurs descriptors disponibles (liste de descriptors voir Section Référence), par exemple :
  - `hasElement <noeud0> <expression>` qui vérifie si le noeud <noeud0> contient une expression <expression> ;
  - `contains <noeud0> <noeud1>` qui vérifie si le noeud <noeud1> est inclu dans le noeud <noeud0>, etc.
- <act-1>...<act-n> sont les actions qu'il faut réaliser si toutes les conditions précédentes sont satisfaites. (liste d'actions voir Section Référence) Quelques exemples des actions :
  - `add <noeud0> <expression>` qui ajoute <expression> dans le noeud <noeud0> ;
  - `newNode <noeud0> <noeud1>` qui crée un nouveau noeud <noeud1> dans le tableau où le noeud <noeud0> est présent, etc.

Règles pour les connecteurs classiques

A titre d'exemple, supposons que dans notre logique, on a les connecteurs classiques `not` (négation) et `and` (conjonction). On peut définir des règles de tableau pour les deux opérateurs classiques de la manière suivante:

```
// double negation
rule "not not"
  descriptor hasElement node (not (not variable A))
  action add node0 (variable A)
end

// alpha rule
rule "and"
  descriptor hasElement node0 (and(variable A)(variableB))
  action add node0 (variable A)
  action add node0 (variable B)
end
```

Si dans un noeud se trouve une formule qui commence par `not not`, par la règle "not not" la double négation sera éliminée. Quant à la règle "and", elle ajoute les deux éléments de la conjonction dans le noeud qui la contient.

---

## 2.2 Logique, suite

Règles pour les connecteurs `nec` et `pos`

Dans la logique modale standard  $K$  avec les connecteurs `nec` (nécessaire) et `pos` (possible), on peut décrire deux règles de tableau correspondant à leur propriété dans l'exemple suivant.

```
// possible
rule "diamond"
  descriptor hasElement node0 (pos (variable A))
  action newNode node0 node1
  action link node0 node1 (R)
  action add node1 (variable A)
end

//nécessaire
rule "K"
  descriptor links node0 node1(R)
  descriptor hasElement node0 (nec variable A)
  action add node1 (variable A)
end
```

La règle "diamond" crée un nouveau noeud (un monde possible) à partir d'un noeud qui contient une formule commençant par `pos` (ici, `pos A`). Les deux mondes sont relié par un arc orienté étiqué `R` et puis la formule `A` sera ajouté dans le nouveau noeud. C'est une règle de structure.

La règle "K" vérifie d'abord s'il y a deux noeuds qui sont reliés par la relation `R` si une formule de la forme `nec A` est présente dans le noeud parent. Si c'est le cas, la formule `A` sera mis dans le noeud fils. Ce genre de règle fait partie des règles de propagation.

---

## 2.3 Stratégie

### Introduction

Après avoir défini le langage et les règles de tableau de la logique, la question qui se pose est comment les combiner et les appliquer. C'est la stratégie qui fait ce genre de travail. Elle met en correspondance des tableaux avec des tableaux (ou pose cela comme une règle de réunion logique) en appliquant répétitivement les règles de manière appropriée. Dans ce cas, appliquer une règle signifie "appliquer les règles simultanément pour tous les modèles possibles dans le tableau".

### Construction de la stratégie

La construction de la stratégie<sup>1</sup> est la suivante :

```
strategy ::= rule |
           repeat strategy end |
           allRules strategy1; strategy2; ... ; strategyN end |
           firstRule strategy1; strategy2; ... ; strategyN end
```

Nous utilisons `firstRule rule1; rule2; ... end` lorsque nous souhaitons appliquer la première règle applicable. Nous utilisons `allRule rule1; rule2; ...end` lorsque nous souhaitons appliquer toutes les règles applicables parmi `rule1; rule2;` et ainsi de suite. Par exemple, si `rule1` et `rule3` sont les règles applicables alors `firstRule` n'appliquera que `rule1` alors que `allRules` appliquera d'abord `rule1` et ensuite `rule3` au résultat de la première règle.

Si un utilisateur dispose d'un jeu de règles  $\{rule1, rule2, \dots, ruleN\}$  qui ont été démontrées complètes pour la logique concernée, alors il peut rapidement implémenter le démonstrateur de théorème complet pour cette logique par une stratégie équitable qui répète séquentiellement l'application de toutes les règles. Une telle stratégie s'écrit :

```
repeat allRules rule1; rule2; ...; ruleN end end
```

Cette implémentation naïve peut engendrer une boucle dans certaines logiques. En conséquence, une stratégie plus sophistiquée devrait être envisagée. Par exemple, la règle `InclusionTest` qui vérifie si un noeud est déjà inclu dans un de ses noeuds ancêtres sera utile pour éviter le bouclage. En effet, avant de passer à une règle comme `diamond` qui crée un nouveau noeud, il faut appliquer la règle `InclusionTest` qui va marquer le noeud qui ne doit pas se développer. Ainsi, la création infinie de noeuds ne se produit pas.

---

1. La structure de stratégie de Lotrec de version 1.5 n'est pas tout à fait la même que celle décrite ici cf. Section de référence : 3.6 Strategy

---

## 2.3 Stratégie, suite

Exemple de stratégie

Voici un exemple de stratégie de la logique K4:

```
// Logique K4
repeat firstRule
  "stop"
  // propositional rules
  repeat allRules
    "not not"; "and";
    "not and"
  end end;
  //generate and check successors
  allRules
    "diamond"; "K"; "4"; "inclusionTest"
  end
end
end
```

## 2.4 Formule à vérifier

### Principe

Lorsqu'on a bien défini une logique et une stratégie associée, le démonstrateur est « prêt à tourner ». Il n'attend qu'une formule de départ, à savoir la formule dont on souhaite vérifier la validité ou la satisfaisabilité dans la logique concernée.

La méthode des tableaux fonctionne par réfutation : si la formule P est valide, il faut prouver que  $\sim P$  n'est pas satisfaisable. Ainsi, si nous voulons démontrer la validation d'une formule, la formule d'entrée est la négation d'une telle formule. Le démonstrateur affichera le résultat constitué du tableau ou des tableaux générés par l'application des règles d'inférence (règles de tableau) à la formule. La formule en question est satisfaisable si et seulement si le tableau est ouvert. En revanche; si le tableau est fermé (toute les branches contiennent la contradiction), cela signifie que la formule d'entrée n'a pas de modèle, elle est donc insatisfaisable. En conséquence, sa négation est valide.

Une formule d'entrée est une expression qui fait partie du langage de la logique sur laquelle on travaille. On la décrit en utilisant des connecteurs de la logique, des constantes et des variables.

### Exemple

Pour illustrer la définition des formules, supposons que dans notre logique, on a les connecteurs suivants: `and(&)`, `not(~)`, `nec([])`, `pos(<>)`.

Formule	Forme interne
$A \ \& \ [](B\&C)$	<code>and (constant A)(nec and constant B constant C)</code>
$P \vee Q$	<code>not (and (not constant P) (not constant Q))</code>
$P \rightarrow Q$	<code>not (and (constant P) (not (constant Q) ) )</code>
$[](P\&\sim Q) \ \& \ \langle Q$	<code>(and (nec (and (constant P)(not constant Q))) (pos constant Q))</code>

A noter que les parenthèses servent à rendre lisible l'expression mais on peut les supprimer en conservant l'équivalence de l'expression.

## 2.5 Déroutement d'une session Lotrec

Contexte de l'exemple Il s'agit de réaliser la logique K4 et de tester 2 formules dans cette logique.

Construction des fichiers d'entrée Une exécution d'une session Lotrec prend comme entrée trois éléments :

- une description de logique (\*.thy),
- une stratégie (\*.str)
- une formule à vérifier (\*.wff).

### Fichier Logique et fichier Stratégie

A titre d'exemple, on tente d'implémenter la logique K4.

- D'abord on construit le fichier `K4.thy` (voir le contenu en Annexes : Fichier `K4.thy`) contenant les définitions des connecteurs et des règles de tableau pour cette logique. Pour faciliter la présentation, on ne dispose que de deux connecteurs classiques (`and` et `not`), un connecteur modal (`nec`) et une constante (`falsum`) servant à signaler la contradiction. D'autres connecteurs peuvent se dériver de ces opérateurs de base de façon habituelle (`or A B = not (and not A not B)`, `pos A = not nec not A`, etc).
- Ensuite, on décrit une stratégie correspondant dans le fichier `K4.str` (voir le contenu en Annexes : Fichier `K4.str`).

### Fichier Formules

Supposons qu'on a deux formules dont on veut démontrer la validité :

- $([] (P \& Q) \rightarrow [] Q)$   
et
- $\sim ([] \langle \rangle P \ \& \ \langle \rangle P)$

On met respectivement leur négation dans les fichiers `K4fm1.wff` et `K4fm2.wff`. Le contenu des deux fichiers de formule est le suivant :

```
// K4fm1.wff
// formule ~ ([](P&Q) -> []Q)
// equivalente a ([](P&Q) & ~[]<Q)
(and (nec and constant P constant Q) (not nec constant Q))

// K4fm2.wff
// formule []<>P & <>P
// equivalente a ([]~[]~P & <>P)
(and (nec not nec not constant P) (not nec not constant P))
```

Activation de Lotrec ;  
test de la première  
formule

### Procédure

On lance une exécution au moyen d'une commande `run.sh` (l'interface développé par F. Massacci) en mettant comme paramètre les trois fichiers Lotrec précédents ;

```
run.sh K4 K4 K4fm1
```

## 2.5 Déroulement d'une session Lotrec, suite

Résultat de la procédure

Pour la première formule, on obtient le tableau de la Figure 2.2 . On voit que le tableau est fermé car il comprend une seule branche et la contradiction (FALSE) est présente dans cette branche. La formule de départ est donc insatisfaisable. On en conclut que  $(\Box(P \& Q) \rightarrow \Box Q)$  est valide dans cette logique.

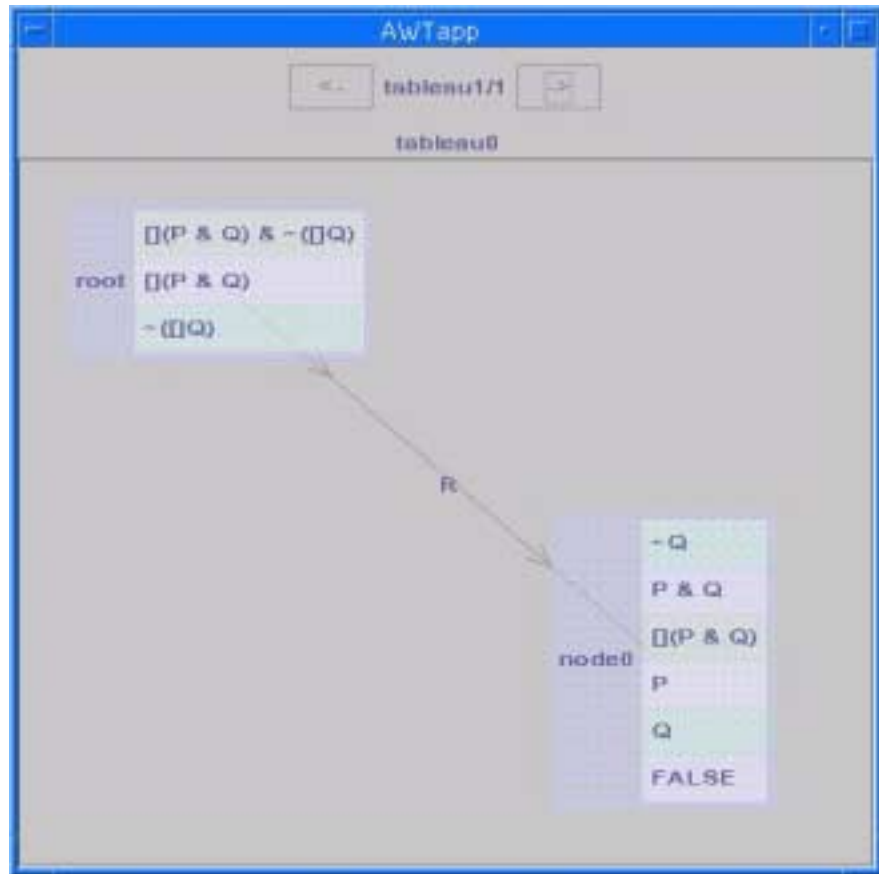


Figure 2.2 Tableau pour la formule  $\Box(P \& Q) \rightarrow \Box Q$

## 2.5 Déroulement d'une session Lotrec, suite

Test de la deuxième formule

Procédure

On intègre la seconde formule en exécutant de nouveau Lotrec ;

```
run.sh K4 K4 K4fm2
```

Résultat de la procédure

Pour la deuxième formule, le tableau de la Figure 2.3 nous montre que la formule d'entrée est satisfaisable. Par conséquent, la formule  $[ ] \langle \rightarrow P \rangle \& \langle \rightarrow P \rangle$  n'est pas valide. On voit que le noeud `node1` a été marqué `#contained#` à cause de la règle `InclusionTest`. Si on n'avait pas fait ce test, le tableau aurait pu développer indéfiniment des nouveaux noeuds.

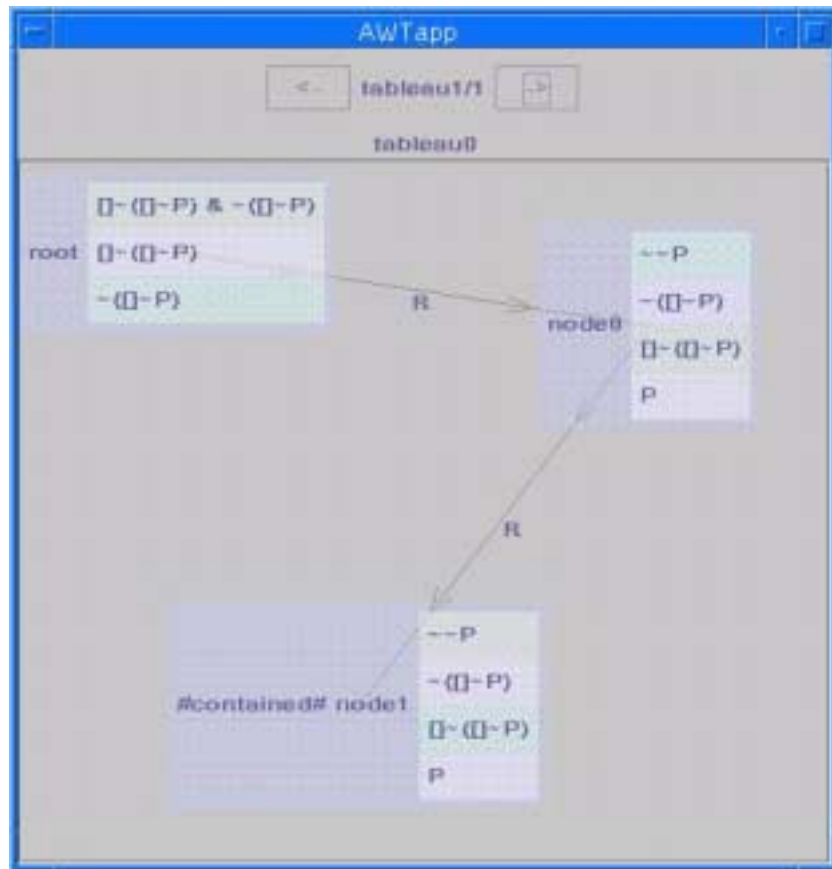


Figure 2.3 Tableau pour la formule  $K4fm2.wff$

---

Dans ce chapitre

- **Connector**
- **Expression**
- **Rule**
- **Descriptor**
- **Action**
- **Strategy**
- **// (Marques de commentaire)**

---

### 3.1 Connector

Syntaxe                    connector    <NOM-INT>    <A>    <ASSOC>    "EXT" <P>

Exem ple                    connector    and       2       true    "\_ and \_"    3

où

- <NOM-INT> est le nom interne du connecteur
- <A> est le nombre d'arguments du connecteur
- <ASSOC> prend la valeur `true` si le connecteur est associatif et la valeur `false` sinon.
- "EXT" est la manière d'afficher le connecteur; ses arguments est représenté ici par `_`.

Par exemple `and A (and C B)` s'affichera `A & B & C` (car <ASSOC> est `true` )

- <P> est la priorité (0,1,2,...) du connecteur par rapport aux autres.

Plus le nombre est grand, plus la priorité est importante.

---

## 3.2 Expression

### Introduction

Il existe quatre types d'expression :

- variable
- constante
- formule avec connecteur
- fonction

### Variable

**syntaxe** : variable <string>

**exemple** : variable A, variable formule2

Une expression variable pourra être remplacée par n'importe quelle autre expression.

### Constante

**syntaxe** : constant <string>

**exemple** : constant P

Contrairement à la variable, une expression de type constant ne peut pas être remplacée par une autre expression.

### Formule avec connecteur

**syntaxe** : <nom-connecteur> <exp-1> ... <exp-n>

**exemple** : and (variable A) (and (constant P) (variable B))

Le connecteur <nom-connecteur> doit être défini(cf. 3.1). Des arguments sont des expressions quelconques.

### Fonction

**syntaxe** : <nom-foncteur> <exp-1> ... <exp-n>

**exemple** : (variable Ki) (O (variable P))

C'est une suite d'expressions. La première expression fonctionne comme un foncteur.

---

### **3.3 Rule**

Présentation                      Une règle de tableau (Rule) est constitué d'une suite de descripteurs suivie par d'une suite d'actions.

Syntaxe

```
rule <nom-règle>
    <suite-descripteurs>
    <suite-actions>
end

<suite-descripteurs> ::= [<descriptor>]*

<suite-actions> ::= [<action>]*
```

### 3.4 Descriptor

Types de descriptor	Syntaxe et description
areIdentical	<i>descriptor areIdentical</i> <noeud1> <noeud2> vérifie que les noeuds <noeud1> et <noeud2> ont le même le nom (indépendamment de leur contenu).
areNotIdentical	<i>descriptor areNotIdentical</i> <noeud1> <noeud2> vérifie que les noeuds <noeud1> et <noeud2> ont des noms différents (indépendamment de leur contenu).
contains	<i>descriptor contains</i> <noeud1> <noeud2> vérifie que le noeud <noeud1> inclut le noeud <noeud2> .
hasElement	<i>descriptor hasElement</i> <noeud> <expression> vérifie que le noeud <noeud> contient l'expression <expression> .
hasNotElement	<i>descriptor hasNotElement</i> <noeud> <exp> vérifie que le noeud <noeud> ne contient pas l'expression <exp>
isAncestor	<i>descriptor isAncestor</i> <noeud-ancêtre> <noeud-descendant> vérifie qu'il existe une succession d'arcs orientés (quelle que soit leur étiquette) entre les noeuds <noeud-ancêtre> et <noeud-descendant>.
isMarked	<i>descriptor isMarked</i> <noeud> <marque> vérifie que le noeud <noeud> est marqué par la marque <marque> (niveau métalogue).
isNotMarked	<i>descriptor isNotMarked</i> <noeud> <marque> vérifie que le noeud <noeud> n'est pas marqué par la marque <marque>
links	<i>descriptor links</i> <noeud-source> <noeud-dest> [<expression>] vérifie que les deux noeuds <noeud-source> et <noeud-dest> sont reliés par un arc orienté de <noeud-source> vers <noeud-dest> et paramétré par l'expression optionnelle <expression> .

### 3.5 Action

Types d'action	Syntaxe et description
add	<i>action add</i> <noeud> <expression> a pour effet d'ajouter l'expression <expression> au noeud <noeud>.
callOracle	<i>action callOracle</i> <noeud> <ligne de commande> créé un fichier <code>exchangeOracle</code> à partir du contenu du noeud <noeud>, appelle l'oracle par la commande <ligne de commande> qui doit alors lire le contenu du fichier <code>exchangeOracle</code> , puis le remplace par le résultat de son calcul.
createNewConstant	<i>action createNewConstant</i> variable <variable-associée> définit un nouveau symbole de variable <variable-associée> (l'utilisation de variables non instanciées provoque une erreur d'exécution).
duplicate	<i>action duplicate</i> <noeud-référence> begin <noeud-source 1> <noeud-dupliqué 1> ... <noeud-source n> <noeud-dupliqué n> end a pour effet de dupliquer le graphe auquel appartient le noeud de référence <noeud-référence> . La séquence entre begin et end permet d'associer, dans les deux graphes obtenus, des noeuds deux par deux. (voir l'utilisation dans la règle "not and" dans le Fichier K4.thy en Annexes)
hide	<i>action hide</i> <noeud> <expression> a pour effet de masquer (et non de supprimer) l'expression <expression> dans le noeud <noeud> (utile pour dans les règles de réécriture, pour ne pas surcharger l'affichage).
link	<i>action link</i> <noeud-source> <noeud-dest> [<expression>] a pour effet d'ajouter un arc orienté entre les noeuds <noeud-source> et <noeud-dest> dont le label (optionnel) est <expression> .
mark	<i>action mark</i> <noeud> <marque> a pour effet de marquer (niveau métalogique) le noeud <noeud> avec la marque<marque> (à utiliser avec les descripteurs <code>isMarked</code> et <code>isNotMarked</code> ).
markExpressions	<i>action markExpressions</i> <noeud> <marque> <formule> a pour effet de marquer (niveau métalogique) la formule <formule> contenue dans le noeud <noeud> avec la marque <marque> .
newNode	<i>action newNode</i> <noeud-référence> <noeud-créé> a pour effet de créer un noeud <noeud-créé> dans le tableau auquel le noeud <noeud-référence> appartient.
stop	<i>action stop</i> <noeud-référence> a pour effet d'arrêter l'application de règles (quelles qu'elles soient) dans tout le graphe auquel le noeud <noeud-référence> appartient.
unMark	<i>action unMark</i> <noeud> <marque> a pour effet de supprimer le marquage d'un noeud <noeud> (à utiliser avec les descripteurs <code>isMarked</code> et <code>isNotMarked</code> ).
unMarkExpressions	<i>action unMarkExpressions</i> <noeud> <marque> <formule> a pour effet de supprimer le marquage d'une formule <formule> du noeud <noeud> par la marque <marque>.
kill	<i>action kill</i> <noeud> a pour effet de supprimer le tableau auquel le noeud <noeud> appartient ( à utiliser par exemple quand on veut supprimer des tableaux fermés).

### 3.6 Strategy

Syntaxe dans la version 2.0

```
strategy ::= rule |
           repeat strategy end |
           allRules strategy1; strategy2; ... ; strategyN end |
           firstRule strategy1; strategy2; ... ; strategyN end
```

où

- `rule` est une règle de tableau (cf. Rule)
- `firstRule rule1; rule2; ... ruleN end` : appliquer la première règle applicable
- `allRule rule1; rule2;... ruleN end` : appliquer toutes les règles applicable parmi `rule1; rule2;` et ainsi de suite.

Par exemple, si `rule1` et `rule3` sont les règles applicables alors `firstRule` n'appliquera que `rule1` alors que `allRules` appliquera d'abord `rule1` et ensuite `rule3` au résultat de la première règle.

Syntaxe de la version 1.0

*Remarque:* On envisage de faire évoluer la structure de stratégie de Lotrec pour obtenir comme la structure ci-dessus. Dans la version courante, une stratégie se construit de la manière suivante:

```
strategy <strategy-name> <keep-exp>
```

où

<strategy-name> est le nom d'une stratégie (identique au nom de fichier « .str » qui la contient)

```
<keep-exp> ::= keep fair <rule-sequence> end
<rule-sequence> ::= rule | <keep-exp> |
                  rule <rule-sequence> |
                  <rule-sequence> rule
```

En fait, `keep fair R1 R2 ... Rn end` fonctionne exactement comme :  
`repeat allRules R1; R2; ... Rn end end`

Options

Dans le fichier contenant une stratégie (\*.thy) avant ou après la définition de la stratégie, on peut insérer certaines options destinées à caractériser une exécution.

Option	Syntaxe et description
step	<code>step "&lt;n&gt;"</code> a pour effet de limiter le nombre de pas du calcul. <n> est un nombre entier correspondant le nombre maximum de pas. Cette option est utile dans le cas où l'exécution risque de boucler mais on souhaite de voir quelques résultats.
checkMarkActivationValidity	<code>checkMarkActivationValidity &lt;true_or_false&gt;</code> Lorsqu'une règle s'active sur un marquage (action mark), il est possible que ce marquage soit enlevé par une autre règle action unmark au moment où la stratégie lui demande de s'exécuter. Dans le cas où <true_or_false> prend la valeur true, Lotrec doit vérifier que le marquage est toujours présent. En revanche; si <true_or_false> prend la valeur false, il fait confiance à l'activation qu'il a reçu (voir l'exemple suivant).

---

### 3.6 Strategy, suite

Pour illustrer le fonctionnement de l'option `checkMarkActivationValidity`, prenons cet exemple :

```
// testMark.thy
rule mark1
  descriptor isNewNode n
  action mark MARK1
end
rule antimark1
  descriptor isMark n MARK1
  action unmark MARK1
end
rule test_mark1
  descriptor isMarked n MARK1
  action add n constant MARK1_FOUND
end
rule test_antimark1
  descriptor isNotMarked n MARK1
  action add n constant MARK1_NOT_FOUND
end

// testMark.str
checkMarkActivationValidity false
strategy testMark
keep fair
  mark1
  antimark1
  test_mark1
  test_antimark1
end

// testMark.wff
constant BEGIN
```

Après l'exécution de Lotrec avec les trois fichiers d'entrée ci-dessus, on obtient un tableau contenant un noeud dans lequel il y a trois formules :

- BEGIN
- MARK1\_FOUND
- MARK1\_NOT\_FOUND

Par contre, si on met `checkMarkActivationValidity true` au lieu de `false` comme avant, le tableau résultat ne contient que deux formules :

- BEGIN
- MARK1\_NOT\_FOUND.

---

### 3.7 // (Marques de commentaire)

Tout ce qui suit le double slash (//) est considéré comme un commentaire qui ne sera pas pris en compte lors de la compilation.

Exemple:

```
// connecteurs classiques -----
connector falsum 0 true "FALSE"      4
connector and    2 true "_ & _"      3
connector not    1 true "~_"         5

// connecteurs modals -----
connector futr   1 true "[F] _"      4
connector past  1 true "[P] _"      4
```

---

# *Annexes*

Dans ce chapitre

- **Fichier K4.thy**
- **Fichier K4.str**

## A.1 Fichier K4.thy

```

// RULES FOR K4 =====
// A. Herzig

// LANGUAGE -----

connector falsum 0 true "FALSE"      4
connector and    2 true "_ & _"      3
connector not    1 true "~_"         5

// modal:
connector nec    1 true "[ ]_"       4

// TABLEAU RULES -----

// stop:
rule stop
  descriptor hasElement node0 (not (variable A))
  descriptor hasElement node0 (variable A)
  action add node0 falsum
  action stop node0
end

// classical:
rule and
  descriptor hasElement node0 (and (variable A) (variable B))
  action add node0 (variable A)
  action add node0 (variable B)
end

rule "not not"
  descriptor hasElement node0 (not (not (variable A)))
  action add node0 (variable A)
end

rule "not and"
  descriptor hasElement node0 (not and (variable A) (variable B))
  action duplicate node0 begin node0 node1 end
  action add node0 (not variable A)
  action add node1 (not variable B)
end

// modal:
rule diamond
  descriptor hasElement node0 (not nec (variable A))
  descriptor isNotMarked node0 CONTAINED
  action newNode node0 node1
  action link node0 node1 (R)
  action add node1 not (variable A)
end

rule K
  descriptor links node0 node1 (R)
  descriptor hasElement node0 (nec (variable A))
  action add node1 (variable A)
end

rule "4"
  descriptor links node0 node1 (R)
  descriptor hasElement node0 (nec (variable A))
  action add node1 (nec (variable A))
end

rule InclusionTest
  descriptor isNewNode node1
  descriptor isAncestor node0 node1
  descriptor contains node0 node1
  descriptor isNotMarked node1 CONTAINED
  action mark node1 CONTAINED
end

```

## **A.2 Fichier K4.str**

```
// STRATEGIE -----
strategy K4
keep fair
  stop;
  keep fair
  // ... local saturation
  keep fair
  "not not"
  and
  "not and"
  end
  // ... node creation
  diamond;
  // ... propagation
  keep fair
  K
  "4"
  end
  // loop testing
  InclusionTest
end
end
```

---

---

# *Index*

---

## **A**

Action 3-6

Architecture 2-2

## **C**

Commentaire 3-9

Connecteurs

  construction 2-3

  définition 2-3

  exemples 2-4

  paramètres 2-3

Connector 3-2

## **D**

Descriptor 3-5

Domaine d'application 1-2

## **E**

Exécution 2-10

Expression 3-3

## **F**

Fichiers d'entrée 2-2

Formule

  définition 2-9

  formule à vérifier 2-9

## **I**

Installation

  fichiers Lotrec 1-3

  machines virtuelles 1-3

  sur machine locale 1-3

## **L**

Lotrec

  présentation 1-2

  principe de fonctionnement 2-2

## **R**

Règles de tableau

  définition 2-4

  syntaxe 2-5

Rule 3-4

## **S**

Session Lotrec. déroulement 2-10

Stratégie

  construction 2-7

  exemple 2-8

Strategy 3-7