# 結合則・交換則
# ツリーオートマトンシミュレータ

# ユーザーマニュアル

2005 年 4 月 1 日作成

# 目　次

# 1　ACTAS とは

ACTAS とは、結合則・交換則ツリーオートマトンのシミュレータで、Associative Commutative Tree Automata Simulator の略です。オートマトンの基本的な演算である、共通集合や和集合、空判定などができます。また、結合則・交換則ツリーオートマトン A と結合則・交換則項書換え系 R を与えて、A の受理言語の R による閉包を受理する結合則・交換則ツリーオートマトンを求めることもできます。使い方によっては、暗号通信プロトコルの安全性の検証にも使えます。

## 1.1　用語の説明

ここでは直観的な説明しかしません。詳しくは参考文献などを見て下さい。

**シグネチャ**　関数記号と定数の集合です。関数記号は引数の数が決まっています。

**項**　項は、関数記号と定数と変数から構成されるデータです。シグネチャ $F$ と変数の集合 $V$ から構成される項全体の集合を $T(F,V)$ と書きます。次のように定義されます。

(1) 定数と変数は項である。
(2) $t_1,\ldots,t_n$ が項で、関数記号 $f$ の引数が $n$ のとき、 $f(t_1,\ldots,t_n)$ も項である。

変数には代入ができます。例えば、変数を $x,y$ 、代入 $\sigma$ を $\sigma=\{x\mapsto a, y\mapsto g(b)\}$ とすると、項 $f(x,y)$ に代入 $\sigma$ を適用すると、項 $f(a,g(b))$ が得られます。適用した結果を $t\sigma$ と書きます。

**結合則・交換則**　あるシグネチャに対し、次のような公理を与えたとき、関数記号 $f$ は結合則を持つといいます。

$$f(x,f(y,z))=f(f(x,y),z)$$

次のような公理を与えたとき、関数記号 $f$ は交換則を持つといいます。

$$f(x,y))=f(y,z)$$

結合則と交換則の両方をもつ関数記号を AC 関数記号と呼びます。AC は、Associative（結合的）and Commutative（交換的）の略です。

**項書換え系**　項書換え系は書換え規則の有限集合で、書換え規則は、項の順序対です。書換え規則は、$l\to r$ と書き、$l$ を左辺、$r$ を右辺と呼びます。AC 関数記号を含む項書換え系を結合則・交換則項書換え系と呼びます。以下では、結合則・交換則項書換え系を単に項書換え系と呼こともあります。項 $t$ と書換え規則 $l\to r$ が与えられたとき、$t$ の中に $l$ のパターンにマッチする部分項が存在したとき、その部分項は $r$ に置き換えられ、この関係を書換え関係と呼びます。項 $t$ から項 $t'$ に項書換え系 $R$ に含まれる規則で書換えられるとき、$t\to_R t'$ と書きます。また、$t$ から 0 回以上の書換えで $t'$ に到達するとき、$t\to_R^* t$ と書きます。

**書換え閉包**　項の集合をツリー言語と呼びます。ツリー言語 $L$ と項書換え系 $R$ に対し、$L$ の $R$ による書換え閉包というのは、$L$ に含まれる項から $R$ によって書換えて得られる項全て

からなるツリー言語で、$(\rightarrow_R^*)[L]$。と書きます。つまり、$(\rightarrow_R^*)[L]=\{t \mid s\rightarrow_R^* t, s\in L\}$ となります。

**ツリーオートマトン**　ツリーオートマトンは、$(F,Q,\Delta,Q_f)$ の組で、ここで、$F$ はシグネチャ、$Q$ は有限集合で、その要素は状態と呼ばれます。$\Delta$ は特殊な形をした項書換え系で、$Q_f$ は $Q$ の部分集合で終了状態と呼ばれます。AC 関数記号を含むツリーオートマトンを結合則・交換則ツリーオートマトンと呼びます。

## 1.2　計算モード

ACTAS にある 6 つの計算モードを説明します。

### 1.2.1　Descendant

結合則・交換則ツリーオートマトン $A$ と結合則・交換則項書換え系 $R$ を与えて、 $A$ の受理言語の $R$ による書換え閉包を受理する結合則・交換則ツリーオートマトンを返します。入力によっては計算できない場合があります。

### 1.2.2　Union

結合則・交換則ツリーオートマトン二つ $A_1, A_2$ を与えて、それらの受理言語の和を受理する結合則・交換則ツリーオートマトン $A$ を返します。つまり、返値 $A$ は $L(A)=L(A_1)\cup L(A_2)$ を満たします。常に計算できます。

### 1.2.3　Intersection

結合則・交換則ツリーオートマトン二つ $A_1, A_2$ を与えて、それらの受理言語の共通を受理する結合則・交換則ツリーオートマトン $A$ を返します。つまり、返値 $A$ は $L(A)=L(A_1)\cap L(A_2)$ を満たします。常に計算できます。

### 1.2.4　Complement

現在は実装されていません。結合則・交換則ツリーオートマトン $A$ の受理言語の補集合を受理する結合則・交換則ツリーオートマトン $A'$ を返すモードです。つまり、返値 $A'$ は $L(A')=T(F)\setminus L(A)$ を満たしま一般には、結合則・交換則ツリーオートマトンは補集合について閉じていません。ただし、結合則・交換則ツリーオートマトンの部分クラスである正則結合則・交換則ツリーオートマトンにおいては閉じています。

### 1.2.5　Emptiness

結合則・交換則ツリーオートマトン $A$ を与えて、その受理言語が空であるか否かを返します。一般に、結合則・交換則ツリーオートマトンの常に計算できますが、計算時間が

#### 1.2.6　Membership

結合則・交換則ツリーオートマトン $A$ と、基礎項 $t$ を与えて、$t$ が $A$ によって受理されるか否かを判定します。常に計算できます。

## 1.3　ルール

各ルールの説明をします。

#### 1.3.1　T-Rule

結合則・交換則ツリーオートマトンを一つ定義します。

#### 1.3.2　R-Rule

結合則・交換則項書換え系を一つ定義します。

#### 1.3.3　Term

項を一つ定義します。

#### 1.3.4　Terms

一つ以上の項を定義します。

#### 1.3.5　Signature

シグネチャを定義します。ただし、ここでは、関数記号を書く必要はありません。関数記号は、T-Rule, R-Rule で使われているものとします。

#### 1.3.6　Constant

定数を定義します。

#### 1.3.7　Var

変数を定義します。

#### 1.3.8　State

結合則・交換則ツリーオートマトンで使う状態を定義します。

## 1.4　GUI の用語説明

GUI に関連する用語の説明をします。

### 1.4.1　データモジュール

T-Rule や R-Rule のことです。データモジュールはデータモジュール名を持つことができます。

### 1.4.2　データモジュールファイル

Signature や T-Rule、R-Rule 等が記述されているファイルのことです。ファイル名がデータモジュールファイル名になります。拡張子には.mod を使用してください。

### 1.4.3　モード選択部

計算モードや計算対象のデータモジュールを選択するウインドウです。詳細は 5.1 節を参照してください。

### 1.4.4　入力部

データモジュールを編集するためのウインドウです。詳細は 5.2 節を参照してください。

### 1.4.5　出力部

計算結果やエラーメッセージを表示するウインドウです。詳細は 5.3 節を参照してください。

### 1.4.6　ログ出力部

計算ログを表示するウインドウです。詳細は 5.5 節を参照してください。

# 2　ACTAS への入力フォーマット

ACTAS への入力フォーマットで記述するファイルを「データモジュールファイル」と呼びます。データモジュールファイルは、データモジュールを記述した、拡張子が.mod のファイルです。

## 2.1　データモジュールファイルのブロックと行

データモジュールファイルは以下のブロックと行から構成されます。

- Signature ブロック　識別子の宣言を行うブロックです。
- R-Rule ブロック　書換規則を記述するブロックです。
- T-Rule ブロック　遷移規則を記述するブロックです。
- Term ブロック　項を記述するブロックです。
- $include 行　他のデータモジュールファイルを読込みます。
- コメント行　任意のコメントを記述します。

各ブロックは任意の順番で記述できます。

**ブロック**　「ブロック」は、ブロック宣言部と定義部で構成されます。

**ブロック宣言部**　ブロックの開始行です。行全体を’[]’で囲みます。
ブロック宣言部は 1 行に記述します。同じ行にブロック宣言部以外の記述をすることはできません。また、複数行に跨がって記述することもできません。
ブロック宣言部には、ブロック名を記述します。ブロックの種類によっては、ブロック名以外の情報を記述する場合があります。
ブロック名としてあらかじめ定義されている名称には、以下のものがあります。これらのブロック名は、大文字、小文字を区別しません。

| | |
|---|---|
| Signature | Signature ブロック。入出力で使用します。 |
| R-Rule | R-Rule ブロック。入力で使用します。 |
| T-Rule | T-Rule ブロック。入出力で使用します。 |
| Term | Term ブロック。入力で使用します。 |
| Message | Message ブロック。出力で使用します。 |
| Comment | Comment ブロック。出力で使用します。 |

上記以外のブロック名が記述されたブロックは、ブロック全体を無視します。

**定義部**　ブロック宣言部の次の行から、次のブロック宣言部の直前の行までが定義部です。
一番最後のブロックは、ファイルの最後までを定義部とします。

## 2.2　使用可能な文字

### 2.2.1　識別子

- 以下の文字は、任意の記号の識別子として使用可能です。単独で使用しても、複数の文字を組合せても構いません。

  **英数字**　英大文字(A-Z)、英小文字(a-z)、数字(0-9)
  識別子は、大文字と小文字を区別します。”Abc”と“abc”は異なる識別子として扱います。

  **特殊文字**　`! @ $ % ~ & . ' _ ? ; { } -`
  ただしこのうち、”-”は、現在のバージョンでは使用できません。

- 以下の文字は、AC 関数記号として、単独で使用できます。その他の記号として使用したり、他の文字と組合せたりすることはできません。

  **特殊文字**　`+ * /`

- 以下の文字は、関数記号として、単独で使用できます。その他の記号として使用したり、他の文字と組合せたりすることはできません。

  **特殊文字**　`^`

### 2.2.2　その他の文字

識別子以外の用途で、以下に示す文字を使用します。
’[’ ’]’ ブロック宣言部を囲む記号
’:’ ブロック宣言部で使用する記号
’(’ ’)’ 関数記号の引数を囲む文字列例: g(y)
‘,’ 関数記号が複数宣言できる場合に関数記号の区切りとして使用します。例: f(x,y)
‘->’ 記号の左辺と右辺をわける記号例: f(x,y) -> g(y)
’#’ コメント行を示す記号

## 2.3　各ブロックと行の記述形式

### 2.3.1　Signature ブロック

A 関数記号、AC 関数記号、C 関数記号、定数記号、変数記号を定義するブロックです。ここに定義した情報をもとに、R-Rule ブロック、T-Rule ブロック、Term ブロックを構文解析します。

Signature ブロックは、ブロック宣言部`[Signature]`で開始します。

定義部には、「属性名 : 識別子」の形式で、A 関数記号、AC 関数記号、C 関数記号、定数記号、変数記号を定義します。

**a :** A 関数記号の識別子を定義します。

**ac：**AC 関数記号の識別子を定義します。

**c：**C 関数記号の識別子を定義します。

**const：** 定数記号の識別子を定義します。

**var：** 変数記号の識別子を定義します。

属性名(a, ac, c, const, var)の大文字、小文字は区別しません。
識別子の定義は、‘,’で区切って複数記述することができます。
記述例）

```
[Signature]
ac: a,b,c          #AC 関数記号
a: d               #A 関数記号
c: f,g             #C 関数記号
var: x, y, z       #変数記号
const: id, bb, cc  #定数記号
```

後述の R-Rule ブロックや T-Rule ブロックで、Signature ブロックに定義していない識別子が使われている場合は、次のように扱います。

- 引数を持つ記号関数記号とみなします。
- 引数を持たない記号状態記号とみなします。
- 計算エンジンにわたすデータモジュール内において、A 関数記号、AC 関数記号、C 関数記号、定数記号、関数記号、状態記号、変数記号として使用する記号が重複する場合は、シンタックスエラーとなります。
- 識別子の宣言が不要な場合は、Signature ブロック全体を省略できます。また、ブロック宣言部だけを記述することもできます。
- A 関数記号と C 関数記号は、計算エンジンでは未実装です。[Signature]ブロックでの宣言は可能ですが、遷移規則や書換規則で使用することはできません。使用した場合、計算エンジンでエラーが発生します。
- AC 関数記号は、常に引数を 2 個持ちます。また、定数記号は、引数が 0 個の関数記号です。

### 2.3.2　R-Rule ブロック

書換系を定義するブロックです。
R-Rule ブロックは、ブロック宣言部[R-Rule:データモジュール名]で開始します。

**データモジュール名**　計算対象とする R-Rule ブロックを指定するために使用します。同じファイルに同じデータモジュール名の R-Rule ブロックが複数記述されている場合は、最初に見つかった R-Rule ブロックを使用します。

定義部には、書換規則を記述します。書換規則が複数ある場合は、改行を区切文字として、

続けて記述できます。

書換規則の構文は以下のとおりです。

左辺->右辺

左辺および右辺には、関数記号、変数記号、定数記号を記述できます。

‘->’　　#右辺と左辺を区切る記号
定数記号　# const 宣言した識別子
変数記号　# var 宣言した識別子
関数記号　#遷移規則中の()をもつ記号

記述例）

```
[Signature]
const: id # 定数記号の宣言
var: x, y # 変数記号の宣言
[R-Rule: rrule1]
e(y)->f(x,y) # 関数記号(変数記号) -> 関数記号(変数記号, 変数記号)
f(x,x)->id # 関数記号(変数記号, 変数記号) -> 定数記号
f(id,x)->x # 関数記号(定数記号, 変数記号) -> 変数記号
g(e(id,y))->z # 関数記号(関数記号(定数記号, 変数記号)) -> 変数記号
```

- 同じ記号の関数記号は同じ数の引数をもつ必要があります。同じ関数記号で引数の数が異なる記述はシンタックスエラーとなります。

  エラーの例）

  ```
  [R-Rule:r]
  f(id,x)->x # OK. 関数記号 f の引数は 2 つ
  f(a,b,c)->y # NG. 関数記号 f の引数が 3 つ
  ```

- 書換規則で使用する関数記号は、その引数に関数記号を持つことができます（ネストすることができます）。

  例）

  ```
  [R-Rule:r]
  f(g(a))-> g(x(y)) # ネストできる。
  ```

- シンタックスチェックは、指定されたデータモジュール名についてのみ行います。

### 2.3.3　T-Rule ブロック

T-Rule ブロックは、ブロック宣言部[T-Rule(受理状態):データモジュール名]で開始します。

**受理状態**　0 個以上の状態記号を記述します。受理状態が複数ある場合は、‘,’（カンマ）で区切って記述します。定義部で使用していない状態記号も記述することができます。

**データモジュール名**　計算対象とする T-Rule ブロックを指定するために使用します。同じフ

ァイルに同じデータモジュール名の T-Rule ブロックが複数記述されている場合は、最初に見つかった T-Rule ブロックを使用します。

定義部には、遷移規則を記述します。遷移規則が複数ある場合は、改行を区切文字として、続けて記述できます。

遷移規則の構文は以下のとおりです。

左辺-> 右辺

左辺および右辺には、関数記号、AC 関数記号、定数記号、状態記号を記述できます。

| | |
|---|---|
| '->' | #右辺と左辺を区切る記号。 |
| 定数記号 | #const 宣言した識別子 |
| 状態記号 | #Signature で宣言していない、引数を持たない記号。 |
| 関数記号 | #遷移規則中の（）をもつ記号。 |
| AC 関数記号 | #ac 宣言した関数記号 |

記述例）

```
[Signature]
const: a, b, c
ac: f
[T-Rule(p,q): trule1]
a->q_a # 定数記号->状態記号
b->q_b # 定数記号->状態記号
c->q_c # 定数記号->状態記号

k(q_a)->q_ka # 関数記号(状態記号)->状態記号
k(q_b)->q_kb # 関数記号(状態記号)->状態記号
k(q_c)->q_kc # 関数記号(状態記号)->状態記号

f(q_ka,q_kb)->p # AC 関数記号(状態記号)->状態記号
```

- 遷移規則に記述する関数記号および AC 関数記号は、その引数に関数記号および AC 関数記号を持つことはできません（ネストすることは出来ません）。

  例）

  ```
  [T-Rule(rurl):trule]
  # OK. y(a)->y(b)
  x(a)->x(x(a)) # NG. ネストできない
  ```

- 左辺と右辺の両方に関数記号または AC 関数記号を記述する場合は、同じ識別子の記号でなければなりません。

  例）

  ```
  [T-Rule(p): tr1]
  f(x) -> f(y) #OK. 左辺と右辺の識別子がともに“f”
  ```

```
f(x) -> g(y) #NG. 左辺と右辺の識別子がそれぞれ“f”と“g”とで異なる
```

- 同じ記号の関数記号は同じ数の引数をもつ必要があります。同じ関数記号で引数の数が異なる記述はシンタックスエラーとなります。

  例）

```
[T-Rule(p): tr1]
f(x) -> f(y) # OK. f の引数の数は 1。
f(x,y) -> p  # NG. f の引数の数は 1 でなければならない。
```

- シンタックスチェックは、指定されたデータモジュール名についてのみ行います。

### 2.3.4 Term ブロック

Term ブロックは、ブロック宣言部[Term:データモジュール名]で開始します。

**データモジュール名** 計算対象とする Term ブロックを指定するために使用します。同じファイルに同じデータモジュール名の Term ブロックが複数記述されている場合は、最初に見つかった Term ブロックを使用します。

定義部には、項を 1 行だけ記述します。
項には以下の記号を記述できます。

| | |
|---|---|
| 定数記号 | #const 宣言した識別子 |
| 関数記号 | #項の中の（）をもつ記号 |
| AC 関数記号 | #ac 宣言した関数記号 |

例）

```
[Signature]
const: a
[Term: term1]
a # 定数記号
[Term: term2]
f(a) # 関数記号(定数記号)
```

- 項に記述する関数記号および AC 関数記号は、その引数に関数記号および AC 関数記号を持つ（ネストする）ことができます。

  例）

```
[Signature]
const: a, b, t3
ac: f
[Term:term1]
t1(t2(t3))
[Term:term2]
f(a,f(a,g(a,b)))
```

- シンタックスチェックは、指定されたデータモジュール名についてのみ行います。

### 2.3.5　$include 行

ファイル中に$include(ファイル名)を宣言することより、データモジュールファイルを読み込むことができます。ファイルは、環境変数 ACTAS INCLUDE に指定されたディレクトリのファイルを読み込みます。ACTAS INCLUDE が設定されていない場合は、カレントディレクトリのファイルを読み込みます。ファイル名にパスは指定できません。ファイル名に‘/’が記述されている場合、‘/’を含んだファイルとして扱われます。

データモジュールファイルは、.mod 拡張子ですが、$include に指定できるファイル名は.mod 以外の任意のファイル名を指定することができます。

$include 行は、データモジュールファイルの最初のブロック宣言部の前にだけ記述することができます。これ以外の場所に記述した場合はシンタックスエラーになります。

例) actas.mod モジュールファイルを指定した場合

```
$include(actas.mod)
```

すでに読み込んだファイルを指定されている場合は、その$include は無視されます。

$include で指定したデータモジュールファイル内に記述されている$include は無視されます。

### 2.3.6　コメント行

#から行末までは構文解析の対象とはなりませんので任意の文字列を記述できます。コメント行は、任意の行に記述できます。シンタックスチェックを行う前に削除されます。

## 2.4　計算エンジンにわたすブロック

計算エンジンにわたすブロックはモードにより異なります。また、モードにより計算エンジンに param1, param2, param3 を渡す必要があります。

※モード、データモジュール名、param1, param2, param3 は GUI によりユーザが指定します。

### 2.4.1　Descendant モード

Descendant モードの処理対象は以下になります。

- Signature ブロック
- データモジュール名で指定された R-Rule ブロック
- データモジュール名で指定された T-Rule ブロック

計算エンジンに param1, param2, param3 を渡す必要があります。

### 2.4.2　Union モード

Union モードの処理対象は以下になります。

- Signature ブロック
- データモジュール名で指定された T-Rule ブロック
- データモジュール名で指定された T-Rule ブロック

※T-Rule ブロックの指定は 2 つ必要です。

### 2.4.3　Intersection モード

Intersection モードの処理対象は以下になります。

- Signature ブロック
- データモジュール名で指定された T-Rule ブロック
- データモジュール名で指定された T-Rule ブロック

※T-Rule ブロックの指定は 2 つ必要です。

### 2.4.4　Complement モード

Complement モードの処理対象は以下になります。

- Signature ブロック
- データモジュール名で指定された T-Rule ブロック

計算エンジンに param1, param2 を渡す必要があります。

### 2.4.5　Emptiness モード

Emptiness モードの処理対象は以下になります。

- Signature ブロック
- データモジュール名で指定された T-Rule ブロック

計算エンジンに param2, param3 を渡す必要があります。

### 2.4.6　Membership モード

Membership モードの処理対象は以下になります。

- Signature ブロック
- データモジュール名で指定された T-Rule ブロック

- Term ブロック

# 3　関連ソフトのインストール方法

本システムを利用するために必要な環境について説明します。

## 3.1　必要なソフトウェア

本システムを利用するためには、以下のソフトウェアが必要です。

| | |
|---|---|
| 推奨 OS | Turbolinux 10 desktop, Red Hat Linux 9.0, Fedora Core 1 |
| 動作確認済み OS | Turbolinux 8 Workstation, Red Hat Linux 8.0 |
| Java | Sun J2SE1.4.2 03 |
| ERLANG | Erlang/OTP R9B-0 |

詳細なインストール方法は各ソフトウェアのマニュアルを参照してください。

### 3.1.1　Java

本システムの GUI 部分および Java で実装した計算エンジンを動作させるために必要です。インストール方法は以下の通りです。

また、インストールの詳細については Sun(http://jp.sun.com)にあるインストールマニュアルを参照してください。

1. Sun の web サイトより J2SEv1.4.2 03 をダウンロードします。
2. アーカイブを展開しようとすると「ライセンスに従いますか?」とインストーラが聞いてきますので、ライセンスに了解した上で、「yes」と入力してください。
3. スーパーユーザ(root)になり、展開された J2SE を/usr/にコピーします。

```
Java インストール例：
$ ./j2sdk-1_4_2_03-linux-i586.bin
Do you agree to the above license terms? [yes or no]
yes
$ ls
j2sdk-1_4_2_03-linux-i586.bin j2sdk1.4.2_03
$ su
パスワードを入力
$ cp -r j2sdk1.4.2_03 /usr/
```

### 3.1.2　ERLANG

Erlang で実装した計算エンジン部分を動作させるために必要です。インストール方法は以下の通りです。

また、インストールの詳細については ERLANG.org(http://www.erlang.org/download.html) にあるインストールマニュアルを参照してください。

1. ERLANG.org より Erlang/OTP R9B-0 の rpm パッケージをダウンロードします。
2. スーパーユーザ(root)になり、ERLANG のインストールします。

   ERLANG インストール例：

```
$ su
```

   パスワードを入力

```
$ rpm -ivh erlang-9.0-0beta2mdk.i586.rpm
```

# 4　ACTAS のインストール、起動方法

## 4.1　ACTAS のインストール方法

1. actas$VERSION.tar.gz を AIST(http://)よりダウンロードします。
2. actas$VERSION.tar.gz 展開します。
3. インストールしたいディレクトリにコピーします。

インストール例:

```
$ tar -zxvf actas$VERSION.tar.gz
$ ls
actas.version actas.version.tar.gz
$ cp -r actas$VERSION /home/hoge/bin/
```

## 4.2　環境変数設定

本システム利用のために、以下の環境変数の設定を行うことができます。

### 4.2.1　PATH

ACTAS を使用するためには java コマンド,erl コマンドにパスを通す必要があります。以下の例では java は/usr/java/bin に、erl は/usr/local/bin にインストールされている場合の例です。

csh, tcsh での例:

```
setenv PATH /usr/java/bin/:/usr/local/bin/:$PATH
```

sh, bash での例:

```
PATH=$PATH:/usr/java/bin/:/usr/local/bin:
export PATH
```

### 4.2.2　ACTAS ENGINE

ACTAS ENGINE は計算エンジンがインストールされているディレクトリを指定するための環境変数です。設定がされていない場合には、lib ディレクトリが計算エンジンのインストールディレクトリになります。

### 4.2.3　ACTAS INCLUDE

ACTAS INCLUDE はデータモジュールファイルを include するディレクトリを指定するための環境変数です。設定がされていない場合には、カレントディレクトリがインクルードディレクトリになります。

#### 4.2.4　ACTAS TMP

ACTAS TMP は ACTAS が利用する一時ファイルを作成するディレクトリを指定するための環
境変数です。ACTAS TMP の設定がされていない場合には/tmp/が一時ファイル作成ディレクトリになります。また ACTAS TMP で指定したディレクトリに一時ファイルが作成できない場合には/tmp/に作成します。

### 4.3　起動方法

actas コマンドがインストールされているディレクトリに移動し actas コマンドを実行します。また、ACTAS ENGINE が指定されている場合にはディレクトリの移動は必要ありません。

```
$ cd $WORK/actas/bin/
($WORK は ACTAS がインストールされているディレクトリ)
$ ./actas
```

起動に成功しますと図 1 の様な画面が表示されます。

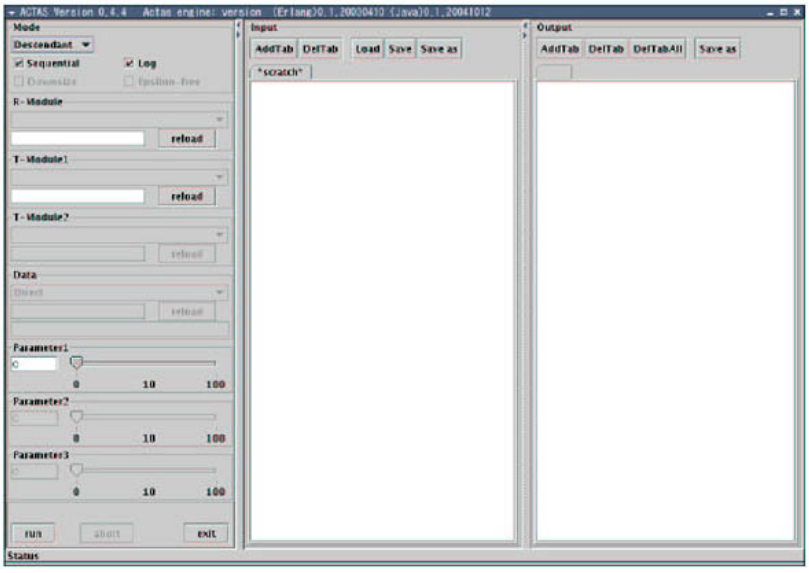


図 1: ACTAS

### 4.4　起動オプション

- -v

  計算エンジンとの入出力、デバッグ用出力を表示します。

- -h

  起動時のヘルプを表示します。

# 5　各部の操作方法

## 5.1　操作部

図 2 が ACTAS の操作部です。操作部は、計算モード選択部、書換系選択部、ツリーオートマトン選択部、項選択部、パラメータ設定部ならびに処理選択部からなります。

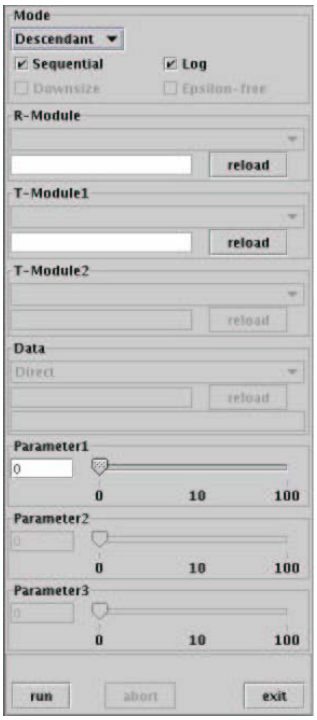


図 2: 操作部

### 5.1.1　計算モード選択部

コンボボックスから計算モードを選択します。

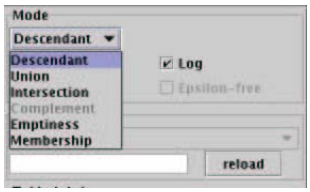


図 3: 計算モード選択部

選択した計算モードにより、扱える GUI 部品が異なります。

**Descendant**　Descendant モードでは、R-Module, T-Module1 が処理対象となります。また、Parameter1, 2, 3 を設定する必要があります。Sequential, Log チェックボックスをチ

ェックすることができます。

**Union**　Union モードでは、T-Module1, T-Module2 が処理対象となります。Log チェックボックスをチェックすることができます。

**Intersection**　Intersection モードでは、T-Module1, T-Module2 が処理対象となります。Log チェックボックスをチェックすることができます。

**Complement**　Complement モードでは、T-Module1 が処理対象となります。また、Parameter1, 2 を設定する必要があります。なお、現在 Complement モードは実装されていないので、選択することはできません。

**Emptiness**　Emptiness モードでは、T-Module1 が処理対象となります。また、Parameter2,3 を設定する必要があります。

**Membership**　Membership モードでは、T-Module1, Data が処理対象となります。

### 5.1.2　計算条件選択部

各チェックボックスを用いて計算条件を選択できます。


図 4: 計算条件選択部

- Sequential チェックボックス

  計算エンジンをシーケンシャルモードで動作させる場合にチェックします。

- Log チェックボックス

  計算結果のログを表示させる場合にチェックします。

- Downsize チェックボックス

  計算エンジンを Downsize モードで動作させる場合にチェックします。現在は計算エンジンが Downsize モード未実装のため選択できません。

- Epsilon-free チェックボックス

  計算エンジンを Epsilon-free モードで動作させる場合にチェックします。現在は計算エンジンが Epsilon-free モード未実装のため選択できません。

### 5.1.3　書換系選択部（R-Module）

書換系選択部では、計算エンジンに処理を行わせる書換系データモジュール(R-Rule ブロック)を選択します。


図 5: 書換系選択部

- reload ボタン

  入力部に表示されているデータモジュールの中から、R-Rule ブロックを検索します。

- 書換系データモジュール名表示部

  reload ボタンを押した際に、R-Rule ブロックのデータモジュール名をコンボボックスに表示します。R-Rule ブロックが検索されていない場合には書換系データモジュール名表示部を使用することはできません。

- データモジュールファイル名表示部

  reload ボタンを押した際に選択されていたデータモジュールファイル名を表示します。また、データモジュールファイル名表示部でマウスの右ボタンを押すと、ファイルの絶対パス名を表示します。

### 5.1.4　ツリーオートマトン選択部(T-Module1, T-Module2)

ツリーオートマトン選択部では、計算エンジンに処理を行わせるツリーオートマトンデータモジュール(T-Rule ブロック)を選択します。

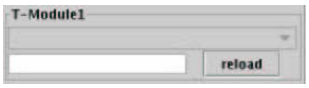


図 6: ツリーオートマトン選択部

- reload ボタン

  入力部に表示されているデータモジュールの中から、T-Rule ブロックを検索します。

- ツリーオートマトンデータモジュール名表示部

  reload ボタンを押した際に、T-Rule ブロックのデータモジュール名をコンボボックスに表示します。T-Rule ブロックが検索されていない場合にはツリーオートマトンデータモジュール名表示部を使用することはできません。

- データモジュールファイル名表示部

  reload ボタンを押した際に選択されていたデータモジュールファイル名を表示します。また、データモジュールファイル名表示部でマウスの右ボタンを押すと、ファイルの絶対パス名を表示します。

### 5.1.5　項選択部(Data)

項選択部では、計算エンジンに処理を行わせる項データモジュール(Term ブロック)を選択します。また、項を直接記述することができます。


図 7: 項選択部

- reload ボタン

  入力部に表示されているデータモジュールの中から、Term ブロックを検索します。

- 項データモジュール名表示部

  reload ボタンを押した際に、Term ブロックのデータモジュール名をコンボボックスに表示します。Term ブロックが検索されていない場合には項データモジュール名表示部を使用することはできません。

- データモジュールファイル名表示部

  reload ボタンを押した際に選択されていたデータモジュールファイル名を表示します。また、データモジュールファイル名表示部でマウスの右ボタンを押すと、ファイルの絶対パス名を表示します。

- 簡易入力部

  項を直接入力することができます。入力した項を使って計算エンジンに処理を行わせるには、項データモジュール名表示部で“Direct”を選択します。

### 5.1.6　パラメータ設定部(Parameter1,Parameter2,Parameter3

スライドバーにより 0～100 までパラメータ変更することができます。また、テキストフィールドに直接値を入力することもできます。テキストフィールドには 100 以上の値を入力することもできます。


図 8: パラメータ設定部

### 5.1.7　処理選択部

- run ボタン

  選択されたデータモジュール、パラメータを用いて計算を実行します。計算結果、構

文解析エラーは出力部（5.3 節）に表示します。

- abort ボタン

  計算を中止するボタンです。

- exit ボタン

  ACTAS を終了します。

## 5.2　入力部

データモジュールファイルを編集するためのウインドウです。

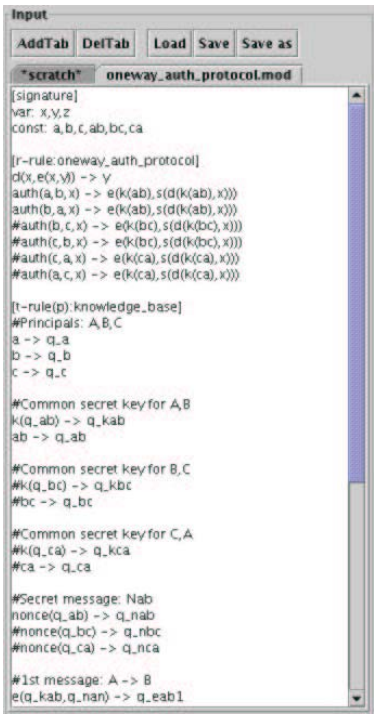


図 9: 入力部

### 5.2.1　ボタン

- AddTab

  新しく編集用のタブを作成します。作成時のタブ名は“scratch”です。

- DelTab

  アクティブなタブを削除します。

- Load

ファイルチューザ（図 10 参照）を開きます。ファイルチューザは、インクルードディレクトリ（4.2.3 参照）に存在するデータモジュールファイル（拡張子が.mod のファイル）およびディレクトリの一覧を表示します。ファイルチューザで選択したファイルは、新しいタブに開かれます。すでに開かれているファイルを開く事はできません。

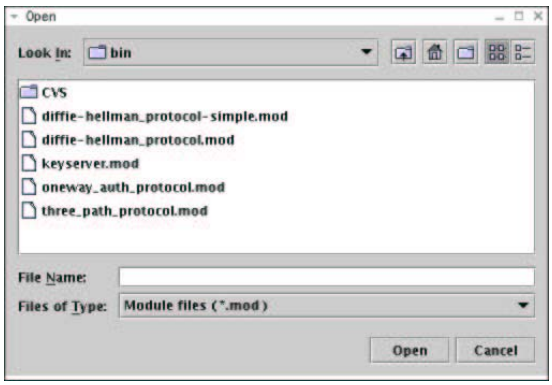


図 10: ファイルチューザ

- Save

  アクティブなタブに表示されている内容をファイルに上書き保存します。保存するファイルが選択されていない場合(タブ名が”scratch”の場合)にはファイルチューザが開きファイル名を指定して新規保存します。

- Save As

  アクティブなタブの内容を新規ファイルに保存します。ファイルチューザが開きファイル名を指定して新規保存します。

### 5.2.2 タブ

アクティブなタブのテキストを直接編集することができます。

## 5.3 出力部

計算エンジンの計算結果を出力する画面です。

### 5.3.1 ボタン

- AddTab

  新しく出力用のタブを作成します。

- DelTab

  アクティブなタブを削除します。

- DelTabAll

  全てのタブを削除し、新しいタブをひとつ作成します。

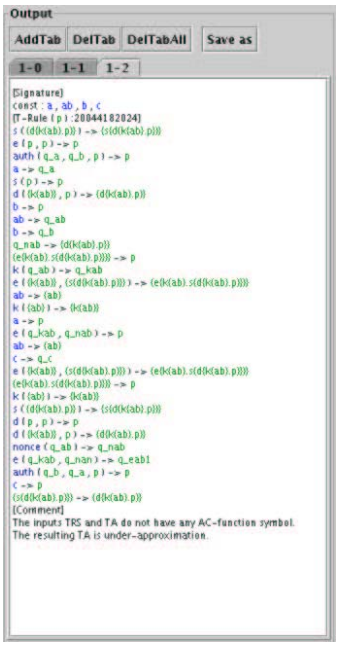


図 11: 出力部

- Save As

  アクティブなタブの内容を新規ファイルに保存します。ファイルチューザが開きファイル名を指定して新規保存します。

### 5.3.2　タブ

計算結果は新規タブに表示します。計算条件によって、下記のように出力方法が異なります。

- Decendant モード

  Sequential チェックボックスをチェックした場合は、処理回数分のタブを表示します。タブ名には処理回数を表示します。
  Sequential チェックボックスをチェックしない場合は、ひとつのタブを更新して表示します。タブ名は空欄になります。

- Union モード

  ひとつのタブに表示します。タブ名は空欄になります。

- Intersection モード

  ひとつのタブに表示します。タブ名は空欄になります。

- Emptiness モード

  結果はダイアログボックスで表示します。タブには出力しません。

- Membersip モード

  結果はダイアログボックスで表示します。タブには出力しません。

- 構文エラーや計算エラー

  ひとつのタブに表示します。タブ名は空欄になります。

### 5.3.3　色分け

計算結果は、各記号を色分けして表示します。色分けのルールは以下の様になっています。

色分けルール:
状態記号深緑色
関数記号藍色
定数記号青色
変数記号赤色

## 5.4　その他の操作

### 5.4.1　画面分割バー

モードセレクト画面、入出力画面の境界にある画面分割バーを左右に移動させる事により自由に 3 つの画面の割合を変更することができます。また、画面分割バーにある「◀」ボタンを押すと、瞬時に 3 つ画面の開閉を行うことができます。

### 5.4.2　ステータスバー

アプリケーションの状態を表示します。

計算中の状態:

| | |
|---|---|
| Status | 初期状態 |
| Parse Error | 構文解析失敗 |
| Finished | 処理終了 |
| Computing... | 計算エンジン計算中 |
| Processing... | 字句解析、構文解析中 |
| Abort | 処理中断 |

データモジュールファイル操作時の状態:

| | |
|---|---|
| Saved | 保存終了 |
| Reloaded | リロード終了 |

### 5.4.3　キーボードショートカット

本システムでは以下の機能を持つキーボードショートカットが割り当てられています。

キーボードショートカット一覧(C-はコントロールを押しながらという意味です。)

ACTAS 全体:

C-r ファイルチューザで指定された全てのデータモジュールファイルをリロードします。

C-x 計算を実行します。

C-c 計算を強制終了します。

C-q ACTAS を終了します。

入力画面のみ:

C-s アクティブなタブの内容を保存します。(入力部のみ)。

C-a 行頭にカーソルを移動する。

C-e 行末にカーソルを移動する。

C-k カーソルの位置から行末までの文字列をカットする。

C-y C-k でカットした文字列をペーストする。

C-d 1 文字削除する(Delete)。

C-h 1 文字削除する(BS)。

## 5.5　ログ画面

モード選択部で Log チェックボックスにチェックを入れた場合に計算ログが出力されます。ログが出力できるのは Decendant, Union, Intersection モードです（計算エンジンから T-Rule の出力があるモードのみ）。

### 5.5.1　Status

計算条件を表示します。計算条件は以下の通りです。また、計算モードにあわせて表示される項目が異なります。

計算状況項目:

| | |
|---|---|
| Mode | 計算モード名 |
| R-Module | R-Module 名 |
| T-Module | T-Module 名 |
| Parameter1 | パラメータ 1 |
| Parameter2 | パラメータ 2 |
| Parameter3 | パラメータ 3 |

### 5.5.2　Date

処理時間を表示します。

処理時間項目:

| | |
|---|---|
| Start | 計算開始時間 |
| Finish | 計算終了時間 |

### 5.5.3　Log

計算毎のログを表示します。ログは処理毎の T-Rule の数、状態記号の数、処理時間を表示します。

計算ログ項目:

| | |
|---|---|
| No | 通し番号 |
| T-Rule | T-Rule の数 |
| State | 状態記号の数 |
| Time | 処理時間（計算エンジンの動作時間のみ） |

### 5.5.4　Graphs

計算毎のログを元にグラフを表示します。グラフが描画されるのは Decendant モードかつ Parameter1 が 2 以上の場合のみです。

- グラフの種類

グラフの種類は以下の 3 種類があります。

グラフの種類:

| グラフ名 | X 軸 | Y 軸 |
|---|---|---|
| T-Rule per Time | 処理時間 | T-Rule の数 |
| State per Time | 処理時間 | 状態記号の数 |
| Time per Loop number | 処理回数 | 処理時間 |

- グラフのスケールの変更

  グラフの下部にある X, Y テキストボックスに任意の数値を入れることによりグラフの X, Y 軸の最大値を変更することができます。図 15 は図 13 のグラフを X を 10,Y を 100 に変更した例です。また最適な描画をするため、指定した値が内部的に若干変更される場合があります。

- 近似曲線グラフ

  beziercurve チェックボックスにチェックをいれるとグラフが近似曲線（図 15）で描かれます。

### 5.5.5　Comment

コメント欄にコメントを入力することにより、ログを保存する際にコメントを残すことができます。

### 5.5.6　Save ボタン

ログを保存できます。ログファイルは HTML ファイルとグラフの画像ファイル（png 形式）で保存します。ログファイルは任意のファイル名を付けることができます。保存形式の詳細は以下の様になります。

ログファイル名:

| | |
|---|---|
| HTML ファイル名 | 任意のファイル名 |
| 画像ファイルディレクトリ | 任意のファイル名_files |
| T-Rule per Time グラフ | 画像ファイルディレクトリ/trule graph.png |
| State per Time グラフ | 画像ファイルディレクトリ/state graph.png |
| Time per Loop number グラフ | 画像ファイルディレクトリ/count graph.png |

### 5.5.7　Close ボタン

ログ画面を閉じます。

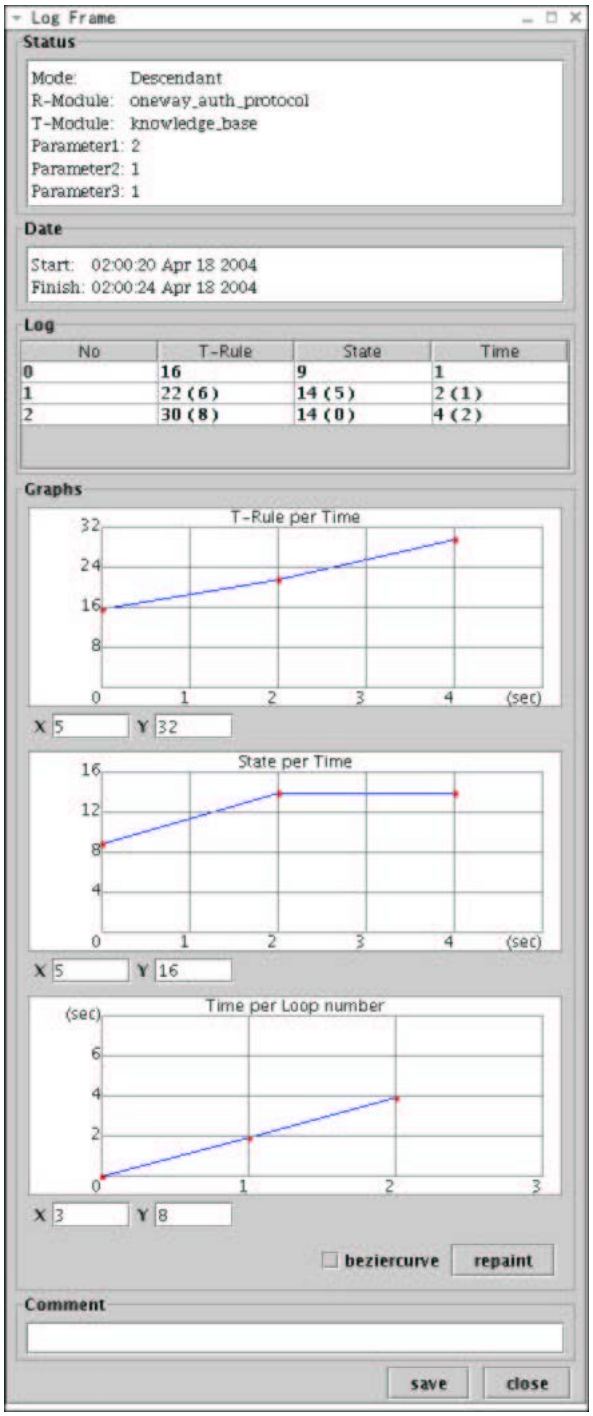
Log Frame
Status
Mode: Descendant
R-Module: oneway_auth_protocol
T-Module: knowledge_base
Parameter1: 2
Parameter2: 1
Parameter3: 1
Date
Start: 02:00:20 Apr 18 2004
Finish: 02:00:24 Apr 18 2004
Log
No
T-Rule
State
Time
0
16
9
1
1
22 (6)
14 (5)
2 (1)
2
30 (8)
14 (0)
4 (2)
Graphs
T-Rule per Time
32
24
16
8
0
1
2
3
4
(sec)
X 5
Y 32
State per Time
16
12
8
4
0
1
2
3
4
(sec)
X 5
Y 16
Time per Loop number
(sec)
6
4
2
0
1
2
3
X 3
Y 8
beziercurve
repaint
Comment
save
close

図 12: ログ画面

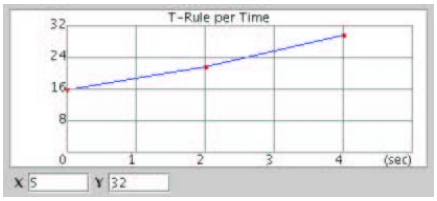


図 13: 通常のグラフ

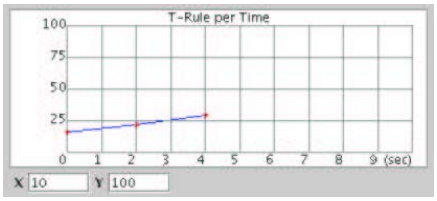


図 14: グラフのスケール変更


図 15: 近似曲線グラフ

## 5.6　ACTAS 操作例

ACTAS の操作例です。Descendant モードを例に説明しています。各部の詳細な説明については 5.1 節を参照してください。

1. モードの選択

   操作部の計算モード選択部を用いて Descendant モードを選択します。

2. 処理対象データモジュールファイルを選択

   入力部の Load ボタンを押してファイルチューザを開き、処理対象のデータモジュールファイルを開きます。

3. 処理対象データモジュールを選択

   reload ボタンを押し、データモジュールファイルから処理できるデータモジュールを表示します。表示されたデータモジュールから計算エンジンに渡すデータモジュールを選択します。

4. パラメータ 1,2,3 の設定

   計算するパラメータを変更します。

5. 実行

   run ボタンを押し実行し、入力ファイルの構文解析が成功し、計算エンジンの処理が終了すると、出力部に計算結果が出力されます。

# 6　エラーメッセージ

出力部に出力されるエラーメッセージには以下のものがあります。

- Module files must be loaded to define R- and T-rules.
  現在のモードで，必要な R-モジュールまたは T-モジュールが選択されていない場合に出力されます。
- Module name needs to be selected from the list of loadable modules.
  現在のモードで，必要な R-モジュールまたは T-モジュールのモジュール名が選択されていない場合に出力されます。
- Null message was returned as the computed result.
  計算エンジンから出力がない場合に出力されます。
- Null T-rule module was returned as the computed result.
  計算エンジンから T-Rule が出力されない場合に出力されます。
- Null T-rule module and signature were returned as the computed result.
  計算エンジンから T-Rule,Signature が出力されない場合に出力されます。
- The arrow symbol “->”+ is missing in R- and/or T-rules.
  モジュール定義中に，矢印記号“->”が抜けている箇所がある場合に出力されます。
- The root function symbols in the left- and right-hand sides must be the same in the same (monotone) T-rule.
  ある（monotone な）T-rule で，左辺および右辺の関数記号が異なる場合に出力されます。
- Flexible argument is forbidden to the standard function symbols (e.g. f(p) and f(p,q) are not permitted to appear in the same module).
  同じ関数記号が，2 通り以上の異なる引数の数を取っている場合に出力されます。
- A special symbol “(”, “)”+ or “->” is detected in an unexpected location
  特殊記号（カッコまたは矢印の記号）が予期しない場所に出現している場合に出力されます。
- A missing and/or extra parenthesis is detected.
  閉括弧が不足している，もしくは余分にある場合に出力されます。
- A variable occurs in [T-rule], [Term], or [Terms] module.
  変数が T-モジュールまたは Term-モジュールに出現する場合に出力されます。
- Without declaration of variables and/or constant symbols, unknown 0-arity symbol (except
- state symbols in [T-rule] modules) appears in [R-rule] or [Term] module.
  変数または定数記号が，宣言なしに使用されている場合に出力されます。（引数がない関数記号が R-モジュールもしくは T-モジュールのなかで使用されている）

- Multiple terms are defined in a [Term] block, instead of using [Terms] declaration.
  [Term]宣言内で，複数の項を定義している場合に出力されます。
- Some symbol is redundantly used in the same input modules, i.e. AC-symbols, constant symbols, and variable names must be pairwise disjoin.
  使用する記号の宣言が，重複しておこなわれている場合に出力されます。
- Multiple [Terms] declarations are forbidden.
  複数の[Terms]宣言がおこなわれている場合に出力されます。

以上。

ACTAS　ユーザーマニュアル

2005 年 4 月 1 日

編集・発行：大崎人士

連　絡　先：独立行政法人産業総合研究所　システム検証研究センター
〒661-0974　兵庫県尼崎市若王寺 3-11-46
e-mail:ohsaki@ni.aist.go.jp