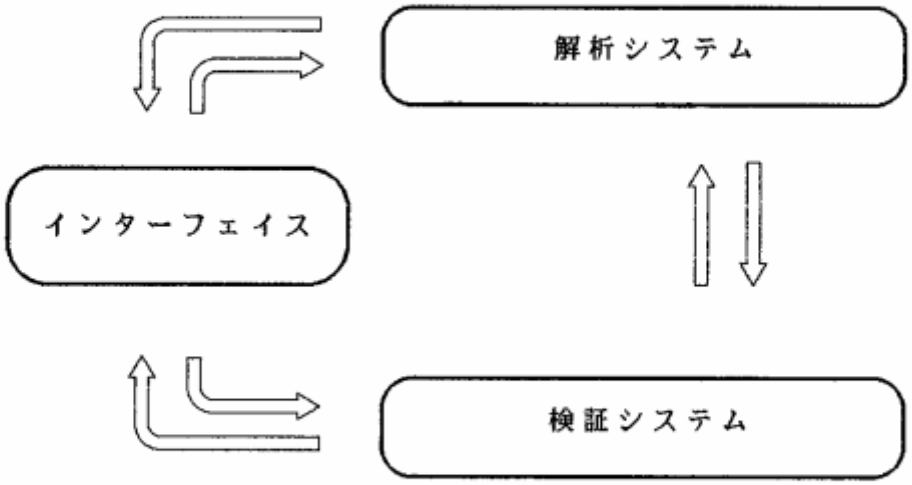
題　名	プログラム解析検証実験システム（Ａｒｇｕｓ）
目　的	プログラムの解析・検証を統一的に行うことにより、ユーザのプログラミング活動を支援するシステムを開発する。
概要及び特徴	Argusは、論理プログラムについて、以下の解析・検証を行います。 ・ゴールが成功する時、各変数がどんなタイプ、形の項に具体化されるかを推定したり、答えは一つか、有限で止まるかを検査します。 ・プログラムが正しく書かれているなら持つはずの論理的性質を持つかどうかを、種々の推論を自動的に行うことによって証明します。 このような解析・検証を行うにあたって、次の特徴を持っています。 ・解析したい性質に応じて抽象化した領域上でプログラムを近似的に実行するという統一的な手法（抽象解釈と呼びます）で様々な解析を行います。その抽象解釈の枠組みとして、トップダウンとボトムアップとの混合型解釈を用いています。 ・Prologプログラムの実行メカニズムを拡張した単純で効率の良い一階推論やProlog向きに特別に工夫された帰納法を用いて推論を行います。また、いろいろなヒューリスティクスを用いて推論をどのようなときにどのような順で行うかを制御して自動的に証明します。
構　成	

デ　　モ　　内　　容　（1／3）

Argusは、次のようにしてプログラムの解析・検証を行います。

(1) 入力となるPrologプログラムです。

```
プログラム ：
   reverse ( [] , [] ).
   reverse ( [A|B] , C)  :-
      reverse (B, D),
      append (D, [A] , C).
   append ( [] , A, A).
   append ( [A|B] , C, [A|D] )  :-
      append (B, C, D).
```

(2) まず、プログラム解析システムArgus/Aによってこのプログラムのタイプ推定を行います。

```
質問 ：
   reverse (A, B)
      <>
```

(3) 解析されるゴールのパターンです。この例では、reverseの引数として任意の項が与えられた場合の解析を行います。

(4) タイプの領域で抽象解釈が行われます。

(5) 解析結果が順次表示されます。

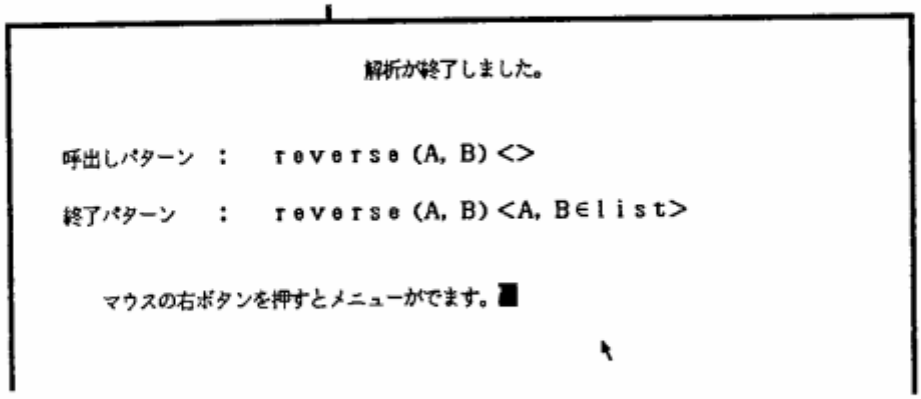
```
■ノード ：
ノード ： 0
      reverse (A, B),
      <>

子ノード ：
ノード ： 1
      [] ,
      <A, B∈list>
ノード ： 2
      reverse (A, B),
      append (B, [C] , D),
      <>
```

```
reverse (A, B) <>:
      [<A, B∈list>]
```

デ　　モ　　内　　容　（2／3）

(6) タイプ推定が終了しました。ゴールreverse(A,B)の実行が成功したときは、変数A,Bがリスト(list)になることがわかります。



Argus/Aは、この他、様々なプログラムの性質を解析します。

(7) 次に、プログラム検証システムArgus/Vによって(1)のプログラムが次のような性質（ゴール）を満たしているかどうかを検証します。

次の仕様　reverse_reverse　の検証を始めます。

g0　：　reverse (A, B)　->　reverse (B, A)

デ　　モ　　内　　容　（3／3）

(8) 帰納法を用いてゴールを簡単にします。（計算帰納法）

```
|
|  reverse(B, A)  に対して
|     reverse(A, B)  について帰納法を適用します。
|
|   用いられるプログラムは
|         reverse([], []).
|         reverse([A|B], C)  :-  reverse(B, D), append(D, [A], C).
|   です。
V

g1 & g2    (帰納レベル 1 )

g1 : reverse(A, B)  &  append(A, [C], D)  ->  reverse(D, [C|B])

g2 : reverse([], [])
```

(9) 拡張実行を行います。

① ゴールの中にプログラムの節頭に代入を施した形が現れたとき、それをそのプログラムの本体に代入を施したものに置き換えます。

```
g3 :  reverse(A, [B|C])  &  append(C, [D], E)  ->
              reverse([D|A], [B|E])
|
|  reverse([D|A], [B|E])  に対してDCIを適用します。
V

g5 :  exist A
      reverse(B, [C|D])  &  append(D, [E], F)  ->
              reverse(B, A)  &  append(A, [E], [C|F])
|
```

② ゴールの中に同じ形のものが二つ以上現れる時、それらが真の場合と偽の場合の単純なゴールにします。

```
g6 :  exist A
      reverse(B, [C|D])  &  append(D, [E], F)  ->
              reverse(B, [C|A])  &  append(A, [E], F)
|
|  reverse(B, [C|D])とreverse(B, [C|A])  についての単純化を行います。
V

g7 :  reverse(A, [B|C])  &  append(C, [D], E)  ->  append(C, [D], E)
```

Argus/Vは、このようにしてプログラムの検証を行います。