Tema AA-6: Introducción a Aleph

José A. Alonso Jiménez
Miguel A. Gutiérrez Naranjo

Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Aleph

- **A Learning Engine for Proposing Hypotheses (Aleph)**
- **Aleph está escrito para Yap Prolog por Ashwin Srinivasan en la Universidad de Oxford**
- **Aleph puede encontrarse en**
  - **http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.pl**
- **Laboratorio de Aprendizaje Automático y Computación de la Universidad de Oxford**
  - **http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/**

Aleph

- **Yap Prolog**
  - **http://www.ncc.up.pt/ vsc/Yap/**
  - **Aleph requiere Yap 4.1.15 o posterior, compilado con la `DEPTH_LIMIT` flag fijada en 1 (esto es, incluye `-DDEPTH_LIMIT=1` en las opciones de compilación).**
- **Manual**
- **PS:**
  - **http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/misc/aleph.ps**
- **HTML:**
  - **http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/**

Algoritmo básico

- **El algoritmo básico de Aleph se puede describir en cuatro pasos**
- **Selección de un ejemplo:**
  - **Cuando ya no queden ejemplos para, en otro caso sigue con el paso siguiente**
- **Construcción de la cláusula más específica:**
  - **Construye la cláusula más específica que cubra el ejemplo seleccionado y pertenezca al conjunto de cláusulas determinado por las restricciones. Usualmente es una cláusula definida con muchos literales llamada la** *bottom clause***. Este paso se suele llamar** *saturación*

Algoritmo básico

- **Búsqueda**
  - **Busca una cláusula más general que la *bottom clause*. Esto se hace buscando un subconjunto de literales en la bottom clause que, en cierto sentido, se clasifiquen como mejores. Este paso se suele llamar *reducción*. La búsqueda se realiza mediante el algoritmo de tipo "ramifica y acota" (branch–and–bound).**
- **Eliminación de redundancias**
  - **La cláusula seleccionada en el paso anterior se añade a la teoría construida en ese momento. En este paso, los ejemplos redundantes (que puedan ser probados a partir del resto de las cláusulas) son eliminados.**

Uso de Aleph

- **Uso simple de Aleph**
- **Tres ficheros**
  - **El fichero del conocimiento básico** `fichero.b`
  - **El fichero de los ejemplos positivos** `fichero.f`
  - **El fichero de los ejemplos negativos** `fichero.n`
- **Consultar los datos con** `read_all(fichero)`

```
torcal:~> yap
[ Restoring file /usr/local/bin/yap ]
[ YAP version Yap4.2.0 ]

yes
   ?- [aleph].
         ...
[ aleph consulted 727172 bytes in 1.62 seconds ]
   ?- read_all(fichero).
[ reconsulting fichero.b... ]
[ train.b reconsulted 43928 bytes in 0.11 seconds ]
[consulting] [pos examples]
[consulting] [neg examples]
[settings]
         caching=false
         ...
         version=2
yes
   ?-
```

- **Construir una teoría con el comando** `induce`**.**

Conocimiento base

- El conocimiento básico que vamos a usar está en el fichero con extensión ".b". Esta es la información que no son instancias del concepto que queremos a prender pero que consideramos útil para el proceso de aprendizaje.
- En este fichero también incluimos las restricciones sobre el espacio de hipótesis y el sistema de búsqueda .Esto lo haremos mediante:
  - Declaraciones de modo
  - Restricciones de tipo
  - Declaraciones de determinación
  - Parámetros

Declaraciones de modo

- **Estas declaraciones determinan el modo de llamada que pueden tener los predicados que aparecen en las cláusulas que conforman la teoría de salida.**
- **Tienen la forma:**
  - `mode(Numero_llamadas,Modo)`
- `Numero_llamadas` **determina el número de éxitos esperado para una llamada del predicado. Puede ser un entero positivo o "*", que representa una cantidad indeterminada.**
- `Modo` **es un patrón del uso que vamos a hacer del predicado. Tiene la forma:**
  - **pred(Tipo_arg_1,Tipo_arg_2,... Tipo_arg_n)**
- **Ejemplos:**

```
:- mode(1,mem(+number,+list)).
:- mode(1,dec(+integer,-integer)).
:- mode(1,mult(+integer,+integer,-integer)).
:- mode(1,plus(+integer,+integer,-integer)).
:- mode(1,(+integer)=(#integer)).
:- mode(*,has_car(+train,-car)).
```

Declaraciones de modo

- Cada `Tipo_arg` es simple o compuesto.
- Un `Tipo_arg` simple puede ser de la forma:
  - `+T` indica que cuando un literal con el símbolo de predicad `pred` aparezca en la cláusula construida, el correspondiente argumento debe ser una variable de entrada de tipo `T`.
  - `-T` indica que el correspondiente argumento debe ser una variable de salida de tipo `T`.
  - `#T` indica que el correspondiente argumento debe ser una constante de tipo T
- Todos los ejemplos anteriores tienen `Tipo_arg` simples.
- Un `Tipo_arg` compuesto es de la forma `f(...)` donde `f` es un símbolo de función y sus argumentos son `Tipo_arg` simples o compuestos. Por ejemplo:

```
:- mode(1,elto(+numero,[+numero|+lista])).
```

Declaraciones de modo

- **Con estas directivas aseguramos que para toda cláusula $A \leftarrow B_1, \ldots B_n$ aprendida verifique:**
- **Variables de entrada:**
  - **Cualquier variable de entrada de tipo `T` en el literal B_i del cuerpo aparece como variable de salida de tipo `T` en un literal del cuerpo que aparezca antes de B_i o aparece como variable de entrada de tipo `T` en la cabeza `H`.**
- **Variables de salida:**
  - **Cualquier variable de salida de tipo de tipo `T` en la cabeza `H` aparece como variable de salida de tipo de tipo `T` en algún literal B_i del cuerpo.**
- **Constantes:**
  - **Cualquier argumento denotado por `#T` sólo puede ser ocupado por términos cerrados de tipo `T`.**

Tipos

- **El tipo tiene que ser especificado para cada argumento de todos los predicados usados en la construcción de las hipótesis. Para Aleph los tipos son sólo nombres y la declaración de tipos no es más que un conjunto de hechos. Por ejemplo:**

```
shape(elipse).  shape(hexagon).  shape(rectangle).
shape(triangle). shape(circle).
```

- **.., es la declaración de que las constantes** `elipse`, `hexagon`, `rectangle`, `triangle` **y** `circle` **son de tipo** `shape`

Determinaciones

- **Las declaraciones de determinación fijan qué predicados pueden usarse para construir una hipótesis. Tienen la forma:**

```
determination(Pred_objetivo/Aridad,Pred_basico/Aridad)
```

- **El primer argumento es el nombre y la aridad del predicado del cual queremos obtener una definición, esto es, el predicado que aparecerá en la cabeza de las cláusulas aprendidas. El segundo argumento es el nombre de un predicado (y su aridad) que puede aparecer en el cuerpo de las cláusulas aprendidas,**

- **Usualmente hay muchas determinaciones para un mismo predicado objetivo. Una para cada predicado que pensamos que puede ser importante para obtener la definición. Si no escribimos la declaración correspondiente a un predicado del conocimiento base, este no aparecerá en las cláusulas aprendidas.**

- **Ejemplos:**

```
:- determination(eastbound/1,has_car/2).
:- determination(mult/3,mult/3).
:- determination(p/1,'='/2).
```

Parámetros

- **Aleph permite hacer restricciones en el espaciode hipótesis y la búsqueda mediante el uso de parámetros.**
- **El predicado `set/2` permite fijar valores a los distintos parámetros**
  - `set(Parametro, Valor)`
- **Podemos obtener el valor actual del paráemtro mediante el predicado**
  - `setting(Parametro, Valor)`
- **Podemos volver al valor por defecto de un parámetro mediante el predicado**
  - `setting(Parametro, Valor)`

Parámetros

- **Ejemplos:**
  - `set(i,+V) V` **es un entero (por defecto 2). Da una cota superior para la profundidad de las nuevas variables.**
  - `set(clauselength,+V) V` **es un entero (por defecto 4). Da una cota superior para el número de literales en una cláusula aceptable.**
  - `set(minpos,+V) V` **es un entero (por defecto 1). Da una cota inferior para el número de ejemplos positivos cubiertos por una cláusula aceptable.**
  - `set(searchtime,+V) V` **es un entero (o** `inf`**). Da una cota superior en segundos para el tiempo de búsqueda.**
  - `set(verbosity,+V) V` **es un entero (por defecto 1). Fija el nivel de comentarios**
- **Otros muchos parámetros en el manual**

Ejemplos

- **Suministramos los ejemplos positivos y negativos como conjuntos de hechos en dos ficheros. Los positivos con la extensión ".f"y los negativos con la extensión ".n".**
- **Ejemplos positivos (train.f)**

```
eastbound(east1).
eastbound(east2).
eastbound(east3).
eastbound(east4).
eastbound(east5).
```

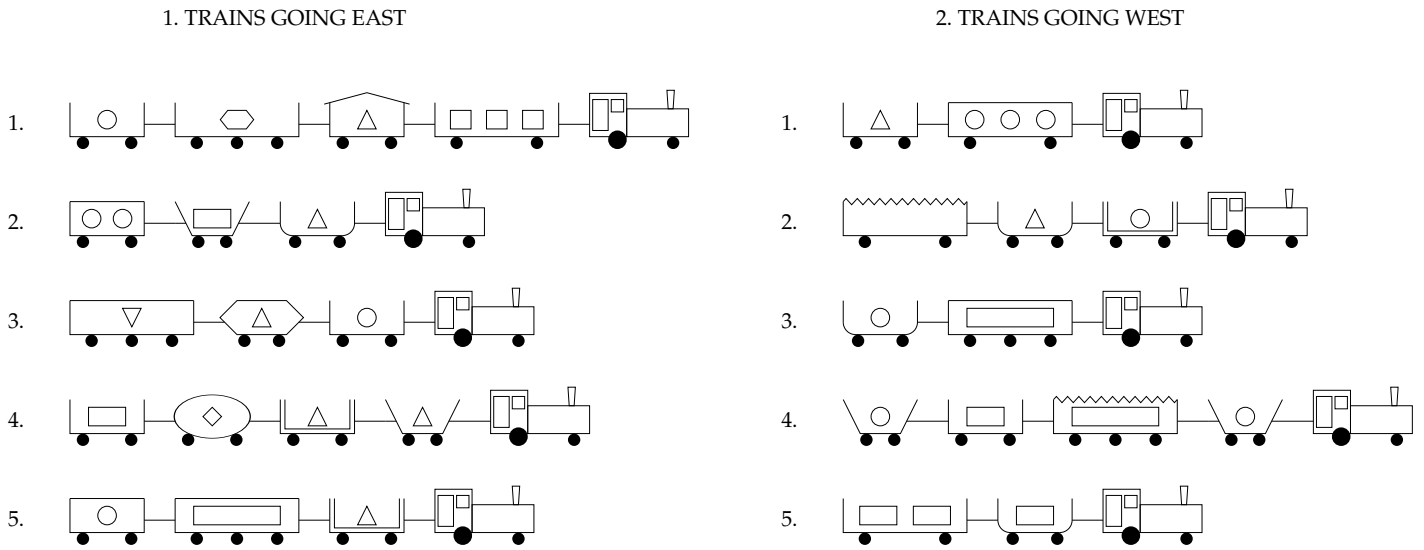
- **Ejemplos negativos (train.n)**

```
eastbound(west6).
eastbound(west7).
eastbound(west8).
eastbound(west9).
eastbound(west10).
```

- **Aleph permite representar el conocimiento negativo de manera más compacta con el uso de restricciones y además puede aprender considerando únicamente ejemplos positivos**

Un ejemplo: Los trenes de Ryszard Michalski

- **Buscamos una explicación que nos permita distinguir entre los trenes que van al este y los trenes que van al oeste.**



Ejemplos

- **Ejemplos positivos** `train.f`

```
eastbound(east1).
eastbound(east2).
eastbound(east3).
eastbound(east4).
eastbound(east5).
```

- **Ejemplos negativos** `train.f`

```
eastbound(west6).
eastbound(west7).
eastbound(west8).
eastbound(west9).
eastbound(west10).
```

Conocimiento básico `train.b`

- **Parámetros**

```
:- set(i,2).
:- set(verbose,1).
```

- **Declaraciones de modo**

```
:- mode(1,eastbound(+train)).
:- mode(1,short(+car)).
:- mode(1,closed(+car)).
:- mode(1,long(+car)).
:- mode(1,open_car(+car)).
:- mode(1,double(+car)).
:- mode(1,jagged(+car)).
:- mode(1,shape(+car,#shape)).
:- mode(1,load(+car,#shape,#int)).
:- mode(1,wheels(+car,#int)).
:- mode(*,has_car(+train,-car)).
```

Conocimiento básico `train.b`

- **Tipos**

```
car(car_11).  car(car_12).  car(car_13).  car(car_14).
car(car_21).  car(car_22).  car(car_23).
car(car_31).  car(car_32).  car(car_33).
car(car_41).  car(car_42).  car(car_43).  car(car_44).
car(car_51).  car(car_52).  car(car_53).
car(car_61).  car(car_62).
car(car_71).  car(car_72).  car(car_73).
car(car_81).  car(car_82).
car(car_91).  car(car_92).  car(car_93).  car(car_94).
car(car_101).  car(car_102).

shape(elipse).  shape(hexagon).  shape(rectangle).  shape(u_shaped).
shape(triangle). shape(circle). shape(nil).

train(east1).  train(east2).  train(east3).  train(east4).  train(east5).
train(west6).  train(west7).  train(west8).  train(west9).  train(west10).
```

Conocimiento básico `train.b`

- **Conocimiento básico:**
  - **Consta de la descripción de cada uno de los 10 trenes.**
  - **Tren 1**

```
short(car_12).
closed(car_12).
long(car_11).
long(car_13).
short(car_14).
open_car(car_11).
open_car(car_13).
open_car(car_14).
shape(car_11,rectangle).
shape(car_12,rectangle).
shape(car_13,rectangle).
shape(car_14,rectangle).
load(car_11,rectangle,3).
load(car_12,triangle,1).
load(car_13,hexagon,1).
load(car_14,circle,1).
wheels(car_11,2).
wheels(car_12,2).
wheels(car_13,3).
wheels(car_14,2).
has_car(east1,car_11).
has_car(east1,car_12).
has_car(east1,car_13).
has_car(east1,car_14).
```

  - **Tren 2 ...**

Sesión

```
torcal:~> yap
[ Restoring file /usr/local/bin/yap ]
[ YAP version Yap4.2.0 ]

yes
   ?- [aleph].
[ consulting aleph... ]


A L E P H
Version 2

Manual: http://www.comlab.ox.ac.uk/oucl/groups/
            machlearn/Aleph/aleph_toc.html

[ aleph consulted 727172 bytes in 1.61 seconds ]
   ?- read_all(train).
[ reconsulting train.b... ]
[ train.b reconsulted 43708 bytes in 0.1 seconds ]
[consulting] [pos examples]
[consulting] [neg examples]
[settings]
        caching=false
        check_redundant=false

        ...

        verbosity=1
        version=2

yes
```

Sesión

```
   ?- induce.
[select example] [5]
[sat] [5]
[eastbound(east5)]
[bottom clause]
eastbound(A) :-
   has_car(A,B), has_car(A,C), has_car(A,D), short(B),
   short(C), short(D), closed(B), closed(C),
   open_car(D), double(D), shape(B,rectangle),
   shape(C,rectangle), shape(D,rectangle), wheels(B,2),
   wheels(C,3), wheels(D,2), load(B,circle,1),
   load(C,rectangle,1), load(D,triangle,1).
[literals] [20]
[saturation time] [0.03]
[reduce]
...
[-----------------------------------]
[clauses constructed] [38]
[search time] [0.08]
[best clause]
eastbound(A) :-
   has_car(A,B), short(B), closed(B).
[pos-neg] [5]
[atoms left] [0]

...
```

Sesión

```
[theory]

[Rule 1] [Pos cover = 5 Neg cover = 0]
eastbound(A) :-
   has_car(A,B), short(B), closed(B).
[pos-neg] [5]

[Training set performance]
          Actual
       +         -
     + 5         0         5
Pred
     - 0         5         5

       5         5         10

Accuracy = 1.0
[Training set summary] [[5,0,0,5]]
[time taken] [0.13]

yes
   ?-
```