Manual del Parser del CAT
(Conversational Agent Toolkit)

Introducción

La herramienta del Parser fue diseñada para el desarrollo de interfaces robustas de Lenguaje Natural para aplicaciones, especialmente para aquellas de lenguaje hablado. Debido a que el habla espontánea esta a menudo mal formada, y porque el reconocedor puede generar errores, es necesario que el parser sea robusto a los errores de reconocimiento, en su gramática y a la fluidez del habla. Este parser esta diseñado para permitir un análisis confiable de este tipo de entradas.

Configuración de la versión

Esta versión de Phoenix contiene los directorios:

- **lib/Phoenix/ParserLib –** Este directorio contiene todas las librerías de las funciones del parser, las cuales son compiladas en el archivo librería libparse.a
- **lib/Phoenix/DMLib –** Este directorio contiene todas las librerías de las funciones del manejador de diálogo las cuales son compiladas en la librería libdm.a
- **lib/Phoenix/Grammar –** Este directorio contiene todas las gramáticas semánticas independientes de la tarea las cuales pueden ser compiladas con las gramáticas dependientes de la tarea. Así mismo contiene el programa compilador de gramáticas el cual compila estas gramáticas en Redes de Transición Recursiva. Además contiene el programa analizador_de_texto el cual provee a interfaz para una entrada escrita (stdin) al parser, con una salida escrita también (stdout).
- **lib/Phoenix/Widgets -** Estas son funciones asociadas a un subconjunto de gramáticas independientes de la tarea. Estas son usadas para poner "tokens" de forma canónica, por ejemplo, fechas, horas, números etc.
- **servers/phoenix –** Este directorio contiene al "wrapper" para compilar el parser en el servidor "Hub de Darpa"**.**
- **app/communicator/kb/Grammar –** Este directorio contiene la gramática especifica de la tarea y el archivo de frames para la tarea de viajes.
- **app/communicator/scripts/testing –** Este directorio contiene a los scripts para la prueba interactiva de las gramáticas y los archivos de prueba para el diálogo.

Teoría de la operación del Parser

El parser Phoenix mapea una entrada a una secuencia de Frames semánticas. Un frame de Phoenix es un conjunto de slots que tienen nombre, donde los slots representan piezas de información relacionadas. La figura 1 muestra un ejemplo de un Frame para la solicitud de información de vuelos. Cada slots tiene asociada una gramática libre de contexto que especifica los patrones de orden de las cadenas de palabras que corresponden a dicho slot. Las gramáticas son compiladas en Redes de Transición Recursiva (RTNs). Un ejemplo de un Frame ya analizado se muestra en la figura 2. Cuando un análisis esta completo, cada slot contiene un árbol semántico para cada cadena de palabras que son las que lo hacen crecer. La raíz de dicho árbol es el nombre del slot.

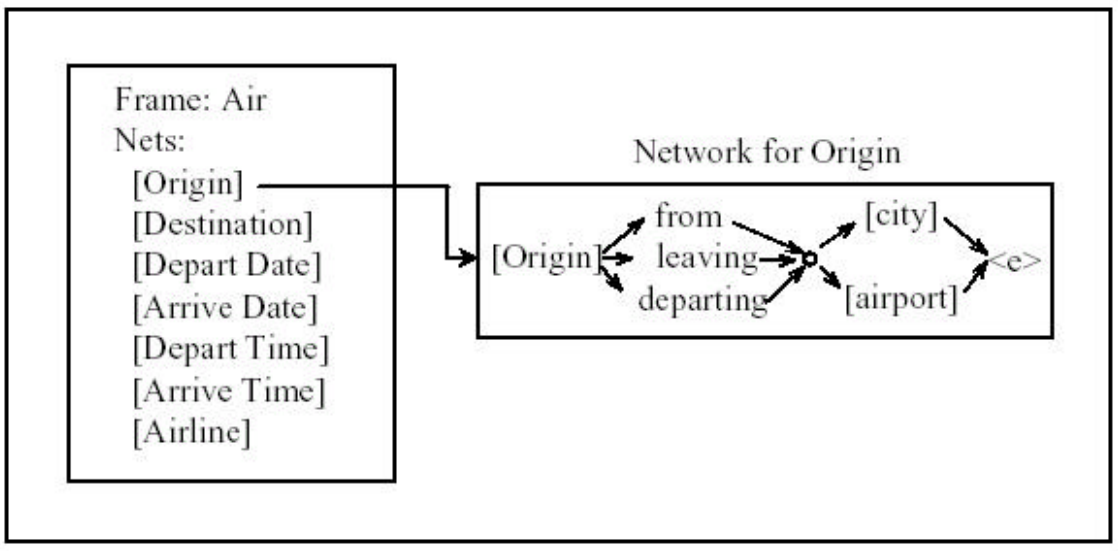


Figura 1 Ejemplo de un frame de Phoenix

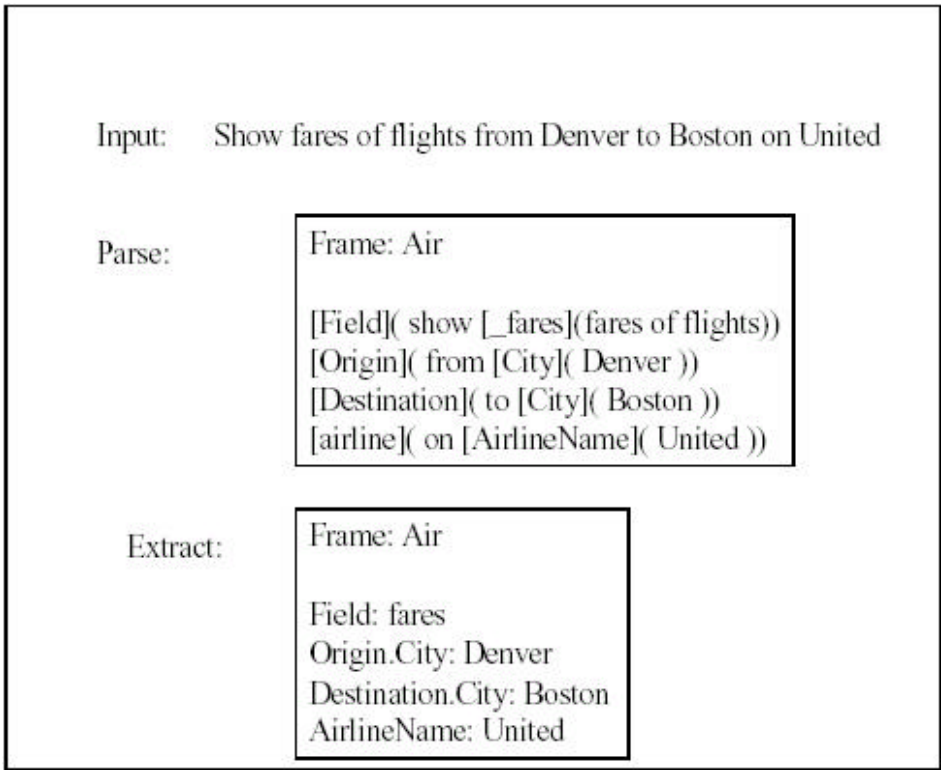


Figura 2 Ejemplo de una salida de analisis

El proceso de parsing se muestra en la figura 3. El algoritmo de búsqueda es muy similar al proceso de correspondencia para la producción de grafos de palabras. Las gramáticas para los slot, son relacionadas de nuevo a una cadena de palabras para producir un grafo de palabras. El conjunto de frames activos define un conjunto de slots activos. Cada slot apunta a la raíz de una Red de Transición Recursiva saciada a él. Estas redes son relacionadas de nuevo a la secuencia de palabras de entrada por un algoritmo de análisis semántico de características Mapa, top-down, y Redes de Transición Recursiva.

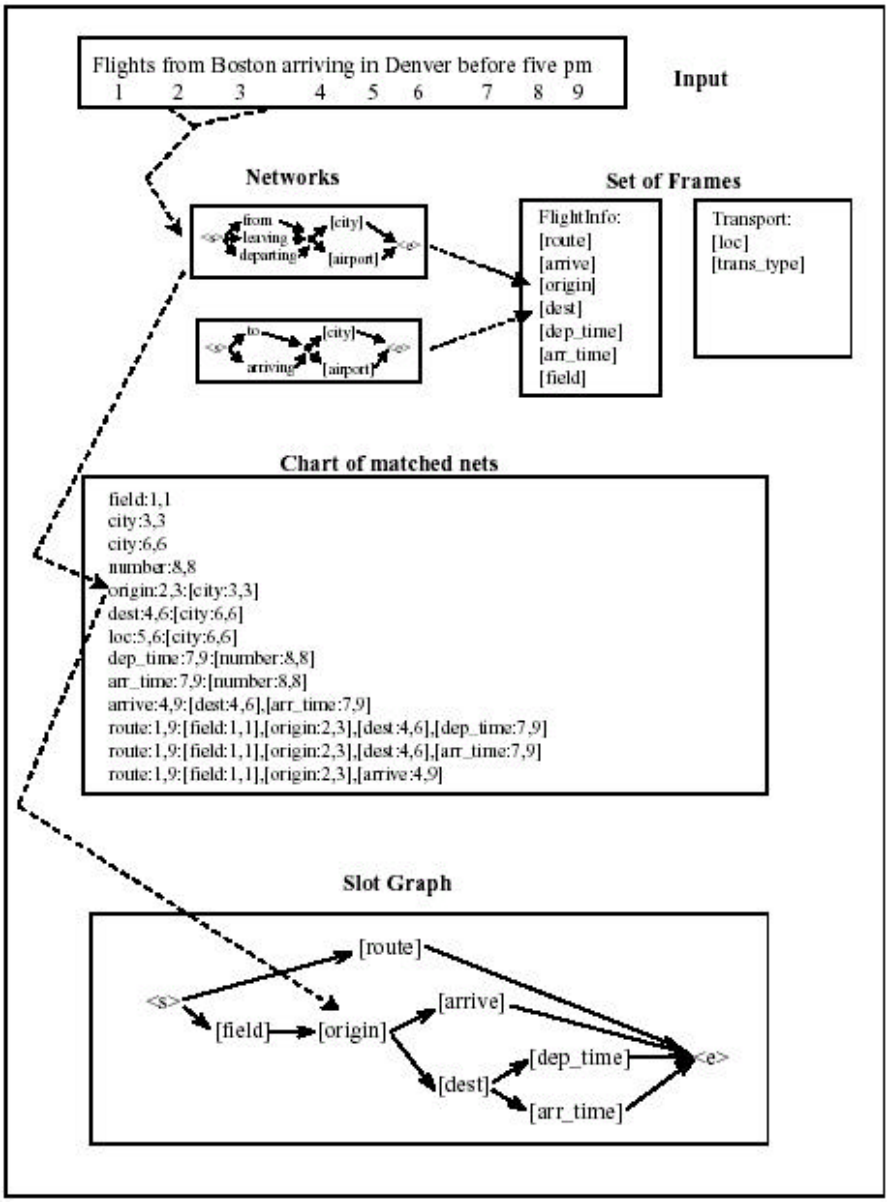


Figura 3 el proceso de parsing

El parser procede de izquierda a derecha intentando relacionar cada slot con el inicio de una red, con cada palabra de la entrada como esta aquí:

```
For (cada palabra de la entrada)
    For (cada slot activo)
        Match_red (slot, palabra)
```

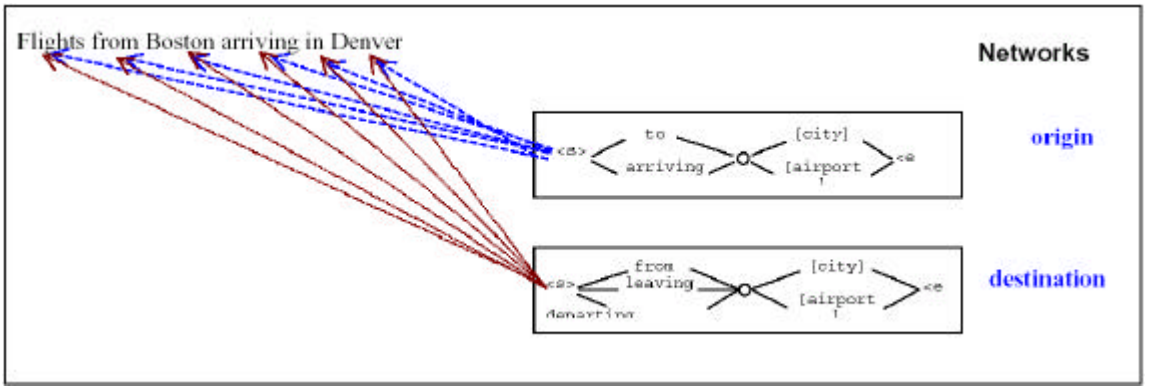


Figura 4 El proceso match

La función match_red, es una función recursiva que relaciona una RTN con una cadena de palabras, comenzando en la posición de una palabra en específico. La función produce todas las relaciones para la red, comenzando en la posición de una palabra y puede tener diferentes puntos finales. Las redes no están diseñadas para analizar oraciones completas. Solo secuencias de palabras cortas. Las Redes de Transición Recursiva para los slots pueden llamar a otras redes durante el proceso de matching. Cada vez que se intenta la relación con una red (todas las redes no solo slots), esta es anotada en el Mapa. Todas las relaciones de red son agregadas al Mapa como se fueron encontrando. Cada vez que se intenta una relación con una red, se checa primero el Mapa de relaciones, para ver si este intento ya fue hecho antes. Cuando la relación con un slot es encontrada, esta es agregada al grafo del slot. Cada secuencia de slots en el grafo de slots es un camino. El puntaje para el camino, es el numero de palabras requeridas para la secuencia. Algunas palabras no pueden ser ignoradas durante el proceso de matching de un slot, pero otras palabras pueden ser omitidas entre slots ya relacionados. El proceso aumenta el grafo, y además poda los caminos que resulta con un puntaje pobre, tanto como la búsqueda acústica se efectúa. Los criterios de poda, el numero de palabras requeridas son primero, segundo el grado de fragmentación de la secuencia. Si dos caminos cubren el mismo tanto de la entrada y si uno considera un mayor numero de palabras que el otro, el menos completo es podado. Si dos caminos consideran el mismo numero de palabras, y si uno ocupa menos slots que el otro, el que tiene mas slots es podado.

El grafo resultante representa todas aquellas secuencias encontradas que tienen un puntaje igual al mejor. La secuencia de slots representadas por el grafo son agrupadas en frames. Esto es hecho por una simple asignación de etiquetas de frame a los slots. De nuevo en este agrupamiento. Los análisis menos fragmentados son escogidos. Esto significa que si dos análisis, cada uno con cinco slots, y uno de ellos ocupa dos frames y el otro ocupa tres, entonces el análisis que usa dos frames es escogido. El resultado es un grafo de slots, cada uno etiquetado con uno o más nombres de frames. Cada camino a través del grafo, todos puntuados igualmente, es un análisis. Este mecanismo naturalmente produce análisis parciales o fragmentados. La búsqueda en programación dinámica produce el análisis mas completo y menos fragmentado posible, dada la gramática y la entrada.

El parser no requiere limites de oraciones, y es capaz de procesar entradas muy largas con pocos inicios nuevos de línea. Este segmenta la entrada en puntos naturales de interrupción, y desarrolla colección de basura en sus buffers. Este parser permite analizar reportes enteros como intervenciones simples si es necesario. La velocidad del proceso es generalmente lineal de acuerdo con la longitud de la entrada. Si un sistema es robusto o restrictivo, éste depende de cómo los frames y las gramáticas están estructurados. El uso de un frame con un solo campo puede producir un parser CFG estándar. Esto es eficiente pero no muy robusto para una entrada inesperada. En caso contrario, tomando un slot para cada palabra producirá un parser de tipo key_word (palabra clave). En algún punto entre estos dos enfoques se obtiene la mejor combinación de la precisión y lo robusto.

Escribiendo gramáticas y frames semánticos

El parser requiere de dos archivos de entrada un archivo de frames y otro de gramáticas.

El archivo de frames

Este archivo especifica todos los frames que serán usados por el parser. Un frame representa algún tipo básico de acción u objeto para la aplicación. Los slots en un frame representan información que es relevante para la acción u objeto. La sintaxis para la definición de un frame es:

comment

FRAME: nombre del frame
NETS: las redesa nivel raiz que se utilizaran en el

[nombre red]
[nombre red]
[nombre red]
;

Ejemplo de un frame para una consulta de reservación de hotel podría ser:

reserve hotel room
FRAME: Hotel
NETS:

[Busqueda_Hotel]
[Nombrehotel]
[Duracion_hotel]
[Localización]
[Tipo_habitación]
[Fecha:llegada]
[Quiero]

;

La definición de un frame debe terminar con ';' en el comienzo de una linea.
¡IMPORTANTE! NO OLVIDE TERMINAR LA DEFINICIÓN DEL FRAME CON PUNTO Y COMA, DE LO CONTRARIO NO SERÁ RECONOCIDO.

El archivo de gramáticas

Los nombres de los archivos de gramáticas tienen la extensión ".gra". Estos archivos tienen la definición de las gramáticas para cada red. Las definiciones son reglas libres de contexto que especifican los patrones de ordenamiento de las palabras. La sintaxis de una gramática para un token es:

```
# comentario opcional
[nombre_token]
 (patron a)
 (patron b)
NT
 (regla de reescritura para NT)
;
```

El nombre del token está encerrado en corchetes. Después de este sigue un conjunto de patrones de reescritura, uno por línea, cada uno encerrado por paréntesis, con un espacio en blanco al inicio de cada línea.
Después de los patrones de reescritura básicos siguen las reglas no-terminales de reescritura, las cuales tienen el mismo formato.
La notación usada para la especificación de los patrones es:

- Cadenas de letras minúsculas son terminales.
- Cadenas de letras mayúsculas son no_terminales.
- Los nombres encerrados entre [] son llamadas a otras redes.
- Expresiones regulares

  * *dato* indica 0 o 1 repeticiones de ese dato

+ indica 1 o mas repeticiones

+* indica 0 o mas repeticiones (equivalente a la estrella Kleene)

?? #include <nombrearchivo> lee el archivo que contiene las reglas de reescritura no-terminales.

Compilando la gramática

La gramática es compilada por la invocación del script **compile**, en el directorio **Grammar**, el cual a su vez llama a **compile_grammar**. La gramática puede ser un simple archivo o puede estar distribuida en varios archivos. Si la gramática se encuentra en varios archivos esto se indica a través de la variable **SingleFile** que obtiene el valor de 0, la compilación del script concatena estos archivos a un solo archivo antes de pasarlo al script **compile_grammar.** Este a su vez abre cualquier gramática encontrada en el directorio /home/CU/lib/Phoenix/Grammars para el archivo antes de invocar a **compile_grammar.** La variable LIBS se coloca junto con el conjunto de gramáticas que serán cargadas desde la librería común. Lo siguiente es el script para compilar la gramática para la aplicación de viajes. Esta usa dos gramáticas a su vez en el directorio local, **Travel.gra** y **Query.gra**, y además carga algunas gramáticas de la librería común. Las gramáticas individuales son concatenadas dentro del archivo **TA.gra** el cual entonces es compilado en el archivo **TA.net**. El compilador también crea otros dos archivos en el directorio local **base.dic**, el diccionario y **nets,** el conjunto de redes que fueron compiladas.

```
# compile grammars in .
set DIR=`pwd`
# put compiled nets in file TA.net
set TASK="TA"
# set PHOENIX to point to root of Phoenix system
set PHOENIX="../../../../lib/Phoenix"
# set LIBS to .gra files to load from $PHOENIX/Grammars
set LIBS="anaph.gra cost.gra date_time.gra existential.gra number.gra \
places.gra pre_term.gra query.gra repair.gra response.gra social.gra"
```

```
# if SingleFile == 1, then all grammar rules are in file $TASK.gra
# if SingleFile == 0, then compiles all files in dir with extension .gra
@ SingleFile = 0
# if separate files, pack into single file, TA.gra
if( $SingleFile == 0 ) then
ls *.gra | $PHOENIX/Scripts/pack_gra.perl > $TASK.gra
endif
# append lib grammars to file
cd $PHOENIX/Grammars
cat $LIBS >> $DIR/$TASK.gra
cd $DIR
# create list of nets to be compiled
cat $TASK.gra | $PHOENIX/Scripts/mk_nets.perl > nets
# compile grammar output messages to file "log"
echo "compiling grammar"
$PHOENIX/ParserLib/compile_grammar -SymBufSize 200000 \
-f $TASK > log
grep WARN log
# flag leaf nodes for extracts function
echo "flagging leaf nodes"
$PHOENIX/ParserLib/concept_leaf -grammar $TASK.net
```

El compilador lee los archivos de frames y de gramáticas (.gra) y produce los siguientes archivos:

.net archivo - archivo de la gramática

Estos son archivos ascii que representan las Redes de Transición Recursiva y tienen el siguiente formato:

La primera línea da el numero de redes que fueron compiladas

Number of Nets= 378

Después vienen las redes compiladas. Para cada red:

La primera línea da: el nombre de la red, numero de red, numero de nodos en la red, bandera de concepto legal.

En tal caso los nodos son listados, cada uno seguido por los arcos que contiene.
Una entrada de datos por nodo al archivo tiene el siguiente formato:

Numero de nodo, numero de arcos del nodo, bandera de final: 0 = no es un nodo final, 1 = es un nodo final.

Para cada red, los nodos son numerados secuencialmente, con el nodo de inicio como 0.

La entrada de datos para los arcos de un nodo siguen el formato del nodo. Estas tienen el siguiente formato:

Numero_palabra numero_red al_nodo

Un arco puede ser un arco_de_palabra, con palabra numero_palabra y numero_red=0 es un arco nulo. Indicado por numero_palabra = 0 y numero_red = 0 un arco invocado, invocado por net_num>0.

[Airline] 27 5 0
0 5 0
0 0 2
20294 0 3

372 372 4
344 344 2
30 30 2
1 0 1
2 1 0
72 72 1
3 3 0
18753 0 1
18762 0 1
16707 0 1
4 3 0
18753 0 1
18762 0 1
16707 0 1

Nets

Este archivo lista los nombres de las redes en la gramática . Este es usado por el script **compile_grammar** para crear el archivo NETS el cual asigna números a las redes.

Base.dic – El diccionario

Este es un archivo de texto que contiene las cadenas de palabras y los números asignados a ellas. Cada línea en el archivo contiene una palabra y el numero asignado a ella. El numero de palabra no es secuencial, son asignados por código hash.

Estructura de Datos

La estructura de la gramática:

Phoenix es capaz de usar múltiples gramáticas. La función read_grammar lee una gramática compilada desde el archivo a.net y regresa un apuntador para una estructura de gramática asociada.

```
struct gram *read_grammar(dir, dict_file, grammar_file, frames_file);

typedef struct gram
{
FrameDef *frame_def; /* apuntadores a la estructura de definición del frame
*/
char **frame_name; /* apuntadores a los nombres de frame */
int num_frames; /*numeros de frames leidos  */
char **labels; /* apuntadores a nombres de red */
Gnode **Nets; /* apuntadores a inicios de red */
int num_nets; /* numero de redes leidas */
char **wrds; /* apuntadores a cadenzas de palabras */
int num_words; /* numero ode palabras en el vocabulario */
char *sym_buf; /* scadenas de palabras y nombres */
} Gram;
```

Tantas gramáticas como se desee pueden ser leídas, y los apuntadores correspondientes a las estructuras de las gramáticas son almacenados. Cuando la función de parsing es invocada, esta pasa la cadena de palabras para analizar y un apuntador de la estructura de la gramática a usarse para el análisis:

```
Parse(cadena_palabras, gramatica).
```

Conjunto de slots activos:

Para agregar un mayor control de cual gramática se puede usar en el análisis, el programador tiene un control dinámico de el conjunto de slots que usa cuando realiza cada análisis. La variable cur_nets(int *cur_nets) es usada para proporcionar esta capacidad. cur_net apunta al conjunto de slots (los numeros de red) que seran usados durante el análisis. Si un subconjunto de todos los slots sera usado durante el análisis, el conjunto de los numeros de red es generado como cur_nets y es apuntado a la lista. Este puede ser apuntado de nuevo al conjunto enterocuando lo desee.

El ciclo principal de match del análisis se muestra como:

```
for( word_position=1; word_position < script_len; word_position++ ) {
for( slot_number= 0; slot_number < num_active; slot_number++ ) {
match_net( cur_nets[slot_number], word_position, gram)
```

El Mapa

El Mapa es una estructura de datos que guarda todas las relaciones de las redes que se han encontrado. Esto se hace para que no se repita el proceso de matching si este es llamado en otro punto de la gramática. En algunos casos, esto salva un gasto computacional considerable. El mapa es una matriz triangular superior que esta indexada por la palabra de inicio y la palabra final del proceso de match de la secuencia de palabras. (Figura 5).

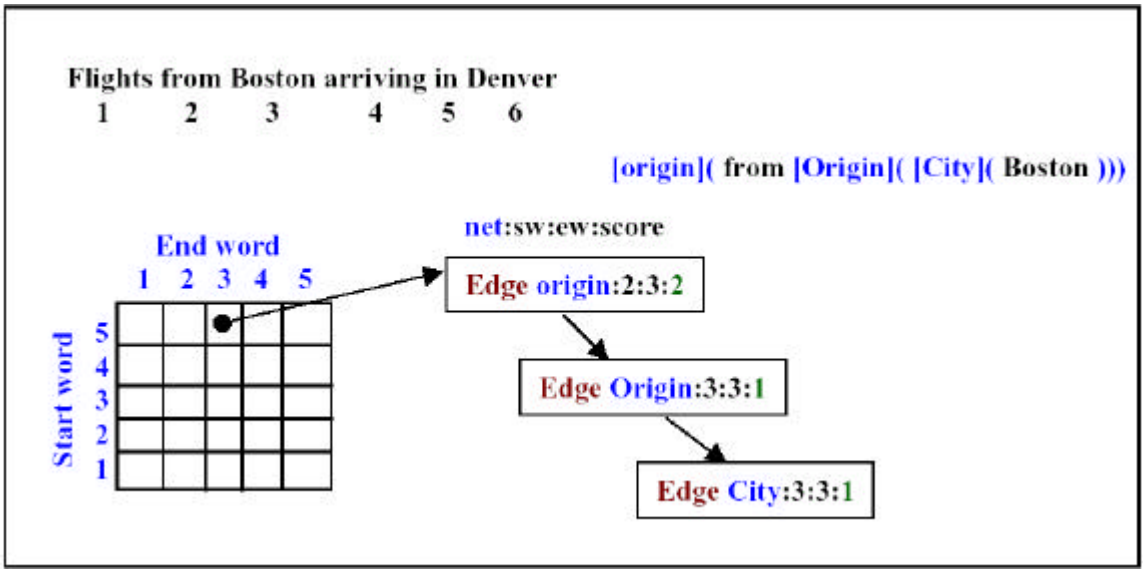


Figura 5 Ejemplo de un mapa

Puesto que una sola matriz no seria suficiente, el Mapa actualmente no esta implementado como tal, pero si como una lista ligada con la palabra de inicio como la de mayor clave y la final como la de menor clave.

```
EdgeLink **chart;

/* estructura para ligar lados en el mapa */
typedef struct edge_link {
int sw;
struct edge_link *link;

Edge *edge;
}
typedef struct edge {
short netid, sw, ew, score;
char nchld; /* number of child edges */
struct edge **chld, /* edge called by this edge */
*link; / link to next edge in list */
} Edge;
```

Estructura de datos del resultado

Cuando una structura de análisis ha sido creado. Este es puesto en un buffer apuntado por la variable *parses*. Un análisis es una secuancia de SeqNodes, los cuales representan bordes cada una con un id de frame agregado. La estructura SeqNode contiene un id de frame y un apuntador a una borde del Mapa. Una secuencia de bordes con el mismo id de frame se encuentran en el mismo frame. Los nodos están registrados en un buffer en orden, con el primer slot de un nuevo análisis anexado después del análisis anterior. Todos los análisis para una intervención pueden tener el mismo numero de slots llenos, debido a que algunos estan mas fragmentados, estos son cortados. El numero de slots en cada análisis esta en la variable global "n_slots". El numero de análisis alternativos esta en "num_parses".

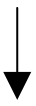
```
SeqNode **parses;

typedef struct seq_node {
Edge *edge;
Fid *frame_id;
short n_frames; /* frame count for path */
} SeqNode;
```

Extraccción de información

El parser provee un mecanismo para imprimir la salida del analisis mas directamente, tal como lo requiere el manejador de diálogo. Este puede imprimir solo las cadenas mas "importantes". Por ejemplo la información que se extraerá. En el orden de uso de este mecanismo el desarrollador debe observar otra cuestion cuando escribe sus gramáticas. Las redes comienzan con mayúscula cuando aparecen en la extracción de información, y las salidas de esas redes pueden ser los valores extraidos. Las redes que comienzan con minúsculas no aparecen en la extracción de salida. Las cadenas de palabras que solo tiene minúsculas en su analisis no aparecen en la salida.

```
movie_info:[movie_info] (
[Display_Movie] ( [_theatre__name] ( where ) ) [is] (is)
[Movie] ( AMERICAN BEAUTY ) [showing] ( playing ) )
```



movie_info:[Display_Movie].theatre_name

movie_info:[Movie].AMERICAN BEAUTY

Una segunda característica de la extracción de salida son los pre_terminales. Estas redes son las que su nombre comienza con el carácter guión bajo (_). Para estas redes el nombre mas que la cadena de palabras es usada como el valor de extracción de salida. Observe *theatre_name* en el ejemplo. Esto es una forma conveniente para poner los valores en forma canónica. Por ejemplo allí pueden haber varias formas de decir *si* pero usted puede necesitar pasar el valor de la cadena exacto a la base de datos. Si usted define a una red [_si] que especifique todas las formas de decir *si* entonces la cadena *si*, mas que la cadena de palabras original, aparecerá en la salida. Esto también funciona para mapear cadenas como *Philly* para *Philadelphia,* etc.

Imprimiendo el análisis.

```
void print_parse(int parse_num, char *out_str, int extract, Gram *gram)
{
int frame;
SeqNode *fptr;
char *out_ptr;
fptr= parses[parse_num];
for(j=0; j < n_slots; j++, fptr++) {
if( fptr->n_act != frame ) frame= fptr->n_act; /* if starting a new frame */
sprintf(out_ptr, "%s:", gram->frame_name[frame]); /* print frame name and
colon */
if( extract )
out_ptr= print_extracts( fptr->edge, gram, out_ptr, 0, gram-
>frame_name[frame]);
else
out_ptr= print_edge( fptr->edge, gram, out_ptr); /* print slot edge */
}
}
```

Corriendo el Parser

Parametros de commandos en linea

Parámetros requeridos:

-dir <directorio de la gramática>

Especifica el directorio que contiene todos los archivos de gramáticas.

-grammar <nombre del archivo>

Nombre del archivo .net que contiene la gramatica que se usará

Banderas:

-verbose <numero>

Numero que puede tener valores de 0 – 6. Tanto como se incremente este numero, progresivamente se incrementará la cantidad de información que se imprimirá de salida. 0 no imprime nada. El valor por default es 3.

-extract 0/1

0 es por default e imprime un análisis completo encerrado en corchetes

1 imprime la parte extraida para ser enviada al DM

-PROFILE 0/1

1 Activa a la obtención de perfiles. Esta da información de el tiempo de analisis y otros recurso usados. Esta puede ser usada para optimizar las medidas de estructuras internas alojadas. El valor por default es 0.

-IGNORE_OOV 0/1

0 causa que las palabras fuera de vocabulario prevengan un match de regla

1 causa que una palabra fuera de vocabulario sean ignoradas y no prevengan matches. El valor por default es 1.

-ALL_PARSES 0/1

0 es valor por default y causa que el sistema imprima un solo análisis.

1 imprime todos los análisis después del proceso total

-BIGRAM_PRUNE 0/1

1 causa un podado más agresivo en la búsqueda. Valor por default 0.

- MAX_PATHS 0/1

1 limita el maximo de caminos que pueden ampliarse durante el proceso de match. Esta es para depurar en caso de una "escapada" de regla, esta causa un gran cantidad de información en la salida. Valor por default 0.

Archivos opcionales:

-function_wrd_file <nombre de archivo>

El archivo contiene la lista de palabras de alto. Éstas son las palabras de función que no deben ser contadas al anotar la calidad de un análisis.

-config_file <nombre de archivo>

Lee una linea de parametros con comandos Rdesde el archivo especificado.

Cambiando los símbolos por default:

-start_sym <cadena ascii>

Pone el inicio del simbolo de incio de una intervención a cadena. El valor por default es <s>.

-end_sym <cadena ascii>

Pone el final del simbolo de una intervencion a cadena. El valor por default es </s>

Medidas de Buffer:

El parser reserva localidades de memoria para los buffers. Cada uno de ellos tiene un valor por default si durante la ejecución un buffer corre fuera del espacio permitido, el parser imprime un mensaje de error diciendo cual buffer esta fuera del espacio permitido y su configuración actual. Desde la lista de parámetros para comandos en línea puede ser usada para incrementar la medida del buffer. Esta tiene todos los valores por default. El parámetro PROFILE puede ser usada para reportar que tanto espacio ha sido usado durante el análisis, por si se desea optimizar las medidad de los buffers.

EdgeBufSize= 1000, /* buffer para los bordes de las estructuras */
ChartBufSize= 40000, /* mapa de estructuras */
PeBufSize= 2000, /* numero de Valores de slot para los arboles */
InputBufSize= 1000, /* maximo de palabras en linea para entrada */
StringBufSize= 50000, /* maximo de palabras en linea de entrada */
SlotSeqLen= 200, /* maximo de numero de slots en una secuencia*/
FrameBufSize= 500, /* buffer para nodos de frame */
SymBufSize= 50000, /* mediad de buffer para mantener cadenzas de palabras */
ParseBufSize= 200, /* buffer para analisis */
SeqBufSize= 500, /* buffer para secuencia de nodos*/
PriBufSize= 2000, /* buffer para secuencia de nodos */
FidBufSize= 1000; /* buffer para ids de frames */