MIKE

Sommario:

MIKE: Micro Interpreter for Knowledge Engineering

- Cos'è
- Dove trovarlo
- Componenti
- Come eseguire un programma MIKE
  - backward chaining
  - forward chaining
- Frames
  - sintassi
  - accesso
  - modifica
  - estensioni (demoni, tipi, cardinalità)
- Regole
  - sintassi
- Uilitites
  - queries per l'interazione con l'utente
  - Spiegazioni e giustificazioni
  - Tracing

Esempio di classificazione euristica:

- diagnosi di malattie dell'apparato respiratorio con MIKE

MIKE

MIKE è un ambiente per costruire sistemi basati sulla conoscenza che è stato sviluppato alla Open University in UK come ausilio all'insegnamento del corso "Knowledge Engineering"

Caratteristiche:

- regole backward e forward
    - di cui l'utente ha la possibilità di definire la strategia di risoluzione dei conflitti

- linguaggio di rappresentazione dei frame con
    - ereditarietà, di cui l'utente puo' definire la strategia
    - 'demoni' (codice attivato dall'accesso o cambiamento dei frame)

- spiegazioni 'how' automatiche (proof histories)

- spiegazioni 'why' definite dall'utente

- possibilità di rule tracing a diversi livelli di granularità
    - stampa di un 'albero di dimostrazione' che mostra come è stata ottenuta una certa conclusione
    - stampa di una 'tabella delle regole' che mostra la storia dell'esecuzione delle regole

Dove trovare MIKE

I file di MIKE si trovano all'indirizzo
http://www-lia.deis.unibo.it/Courses/AI/Software.html
MIKE è free, quindi potete copiarlo e usarlo a casa su qualsiasi PC con il DOS.

Noi useremo la versione 2.50:

- è la più recente
- è costituita da un programma DOS con interfaccia a carattere e menu a tendina
- non è disponibile il sorgente

E' disponibile anche la versione 2.03 perché contiene il sorgente Prolog

Importante: i **manuali** più aggiornati sono in Mike25.zip nei files: Mikeref.doc (di base), Mike2ref.doc (caratteristiche avanzate) e 00readme.txt (specifico per MIKE 2.50)

Architettura e filosofia di MIKE

MIKE e' stato implementato in Prolog e quindi ne condivide la sintassi:

- occorre terminare gli input al prompt con un punto,
- le variabili hanno la stessa sintassi di quelle Prolog,
- gli operatori MIKE sono parole chiave che agiscono come comandi e che hanno la sintassi degli operatori Prolog infissi (come '+' e '-'), dato che sono appunto implementati così.

Ha 4 componenti principali:

- top level
- working memory
- frame memory
- rule memory

Top level: è il prompt in cui l'utente inserisce i comandi. In MIKE 2.50, oltre al prompt, l'utente può usare i menu a tendina.
Working memory (wm): può contenere:
atomi alfanumerici, numeri interi o floating point, stringhe, liste o formule atomiche Prolog generali (predicato + argomenti) come ama(giovanni,maria).
Gli elementi della wm sono aggiunti o tolti durante la computazione usando i predicati MIKE **`add`** e **`remove`**. Tipicamente, prima di ogni esecuzione, la wm è ri-inizializzata: tutti gli elementi sono rimossi ed è aggiunto l'atomo `start`. Il comando da menu 'Forward chaining' inizializza la wm e fa partire la nuova computazione.

Frame memory

Frame memory (fm): contiene la rappresentazione degli oggetti strutturati. E' permanente, anche se può essere modificata. Gli oggetti possono essere di due tipi: **istanze** o **classi**. Ogni oggetto è caratterizzato da una sequenza di **slot** e relativo **filler**, che possono essere considerati coppie attributo/valore del tipo:

```
fred_smith instance_of person with
    age: 49,
    birthday: [29,november],
    weight: 160,
    occupations: [teacher,lifeguard,parent].
```

L'accesso alla frame memory può avvenire direttamente dalle regole, usando la sintassi <frame-acces> che può avere la forma

the <slot> of <object> is <variable-or-filler>

oppure alternativamente la forma

all <slot> of <object> are <variable-or-list-of-fillers>

Gli oggetti frame sono normalmente creati usando un text editor e poi caricati in MIKE. Però possono essere modificati direttamente dalle regole oppure dall'utente usando l'operatore **`note`**. Esempi di modifica e accesso ad un frame dal prompt dei comandi:

```
MIKE ?- note the age of fred_smith is 50.
Yes
MIKE ?- the age of fred_smith is What.
What=50
```

Rule memory

Rule memory: contiene la rappresentazione delle regole. E' permanente, anche se può essere modificata.
Le regole sono di due tipi: **backward chaining** o **forward chaining**.

Regole forward chaining: hanno
- una serie di condizioni, che accedono alla working memory corrente o alla frame memory permanente
- una serie di azioni, che usualmente aggiungono o rimuovono elementi dalla wm (usando `add` o `remove`) oppure alterano la frame memory (usando `note`).

Normalmente, è l'alterazione della wm che fa scattare un'altra regola forward, finché l'azione speciale `halt` non è invocata da una delle regole oppure nessuna regola è applicabile.

Regole backward chaining: hanno
- una serie di condizioni come quelle delle regole forward,
- una singola conclusione, che è un pattern da dedurre. Le deduzioni non sono memorizzate (nelle wm o fm) ma semplicemente falliscono o hanno successo.

Il backward chaining termina quando una conclusione è stata dedotta con successo oppure è fallita.

Esecuzione di un programma MIKE

Una volta scritti i frames e le regole in un file di testo (chiamato **knowledge base**) e caricatolo in MIKE, l'utente può far partire l'esecuzione in due modi invocando:

- il forward chaining (da menu oppure da prompt con **`fc`**)
- il backward chaining (da prompt con l'operatore **`deduce`**)

Durante il forward chaining, se ci sono più regole applicabili nello stesso momento, per scegliere quella da applicare si usa una **strategia di risoluzione del conflitto**. MIKE ne ha 3 differenti:

1. **refractoriness**: le regole hanno un periodo 'refractory' che significa che una regola, dopo essere stata applicata una volta (con una specifica instanziazione delle sue variabili), non potrà essere applicata di nuovo (con le stesse instanziazioni) per un certo tempo

2. **recency**: le regole che si applicano agli elementi della wm aggiunti più recentemente vengono preferite alle altre

3. **specificity**: le regole con più condizioni nella parte sinistra vengono preferite alle altre, perché si pensa che siano più precise.

L'ordine in cui sono applicate è 1,2,3 ma può essere cambiato.

Esempio di esecuzione

```
rule demo forward
   if
      start &
      deduce 'it is going to rain today'
   then
      announce ['I am not going out today'] &
      halt.

rule conclude_rain backward
   if
      'the barometric pressure is rising' &
      'the western sky is cloudy'
   then
      'it is going to rain today'.

-------------------------------

/* esempio di 'seeding' della wm */
MIKE ?- add 'the barometric pressure is
rising'.
yes
MIKE ?- add 'the western sky is cloudy'.
yes
MIKE ?- add start.
yes
MIKE ?- go.
/* comando alternativo a fc: non cancella */
/* la wm, né aggiunge start */
I am not going out today.

Production system halted.
yes
```

Lo schermo di MIKE

E’ diviso in 3 parti:

- la linea di stato (la prima linea in alto)
- la finestra di dialogo (in basso)
  è la finestra in cui compare il prompt di MIKE e in cui comandi e query possono essere inseriti in formato testo.
- l'area di menu (il resto dello schermo)

```
MIKE 2.50   Enter=Select option   Esc=Leave menu and enter MIKE dialogue
+ Main Menu ----------------------------------------+
¦ Forward chain                                     ¦
¦ Load new knowledge base [currently: DEMO5.PL]     ¦
¦ Edit-then-load knowledge base [alt-x to return]   ¦
¦ Working memory display [0 elements]               ¦
¦ Memory changes (wm and frames)                    ¦
¦ Browse frame hierarchy [19 frames]                ¦
¦ Rule display [13 rules]                           ¦
¦ History display [0 cycles]                        ¦
¦ Tracing options                                   ¦
¦ Status of options and user preferences            ¦
¦ switch to DOS [type exit to return]               ¦
¦ Quit MIKE                                         ¦
+---------------------------------------------------+
+ MIKE dialogue   Current kb: DEMO5.PL ------------------------------------------+
¦                                                                              ¦
¦MIKE ?-                                                                       ¦
¦                                                                              ¦
¦                                                                              ¦
¦                                                                              ¦
¦Clearing up...OK.                                                             ¦
¦                                                                              ¦
¦MIKE ?-                                                                       ¦
+------------------------------------------------------------------------------+
```

Lo schermo di MIKE (2)

La linea di stato mostra la lista di tasti utili a seconda del contesto

La finestra di dialogo è la finestra in cui compare il prompt di MIKE e in cui comandi e query possono essere inseriti in formato testo. Tipicamente si usa per invocare il backward chaining (con deduce) e per manipolare la wm e i frames.

Dal prompt, premendo F2, si apre il menu principale nell'area di menu, interrompendo l'input da prompt. Comunque, durante alcune attività guidate da menu, come il forward chaining, la finestra di dialogo può essere usata per mostrare informazioni (dal tracer per esempio) all'utente.

Da qualsiasi menu premendo Esc si torna al menu precedente.

Frames

I frames sono le strutture dati usate in MIKE per descrivere degli oggetti. Gli oggetti (classi o istanze) sono entità qualsiasi del mondo che hanno certe proprietà.

La sintassi (versione semplice) per i frames è

```
<object> <instance-or-subclass-of> <class> with
    <slot1>:<filler-or-list-of-fillers1>,
    <slot2>:<filler-or-list-of-fillers2>,
    ......
    <slotn>:<filler-or-list-of-fillersn>.
```

- Un frame deve avere almeno una coppia <slot>:<filler>
- <object>, <class> and <slot> devono essere un atomo alfanumerico (il primo carattere deve essere una lettera minuscola).
- <instance-or-subclass-of> è uno dei due operatori infissi predefiniti **`instance_of`** oppure **`subclass_of`**.

Esempi:

```
man subclass_of person with
    sex: male

tom instance_of man with
    age 34,
    hobbies: [skiing, photography].
```

Nota che le classi alla radice della gerarchia (come person) non richiedono una definizione!

L'ereditarietà multipla è vietata: un oggetto non può essere sottoclasse o istanza di più classi. In tal caso il comportamento è indeterminato.

Accesso ai Frames

5 modi diversi, che possono apparire nel lato sinistro di una regola oppure al top level:

the <slot> of <object> is <filler>
all <slot> of <object> are <list-of-fillers>
the <slot> of <object> <math-operator> <number>
<object> instance_of <class>
<class> subclass_of <class>

<math-operator> è '<' o '>' ed è usato per confrontare il contenuto di <slot> con <number>

Variabili possono essere usate al posto di <filler> o <list-of-fillers> (a anche dentro <list-of-fillers>) nei primi due casi, oppure al posto di <object> o <class> negli ultimi due (ma non di entrambi).

Per i primi 3 casi, <slot> e <object> non possono essere variabili non istanziate.

In realtà MIKE non da errore di sintassi in questo caso, ma il comportamento dipende dall'ordine delle clausole nel KB e quindi il risultato non è garantito.

```
MIKE ? - the age of X is 64.
MIKE ? - the X of tom is 64.
```

Non sono ammessi a meno che X non sia già istanziato.

Esempi di accesso ai frames

```
MIKE ? - the age of tom is X.
X=34   /* semplice accesso diretto */

MIKE ? - the age of tom > 15.
Yes.   /* operatore matematico */

MIKE ? - the sex of tom is What.
What=male /* ereditato dalla classe man */

MIKE ? - X instance_of man.
X = tom   /* reperimento semplice */

MIKE ?- man subclass_of C.
C = person   /* reperimento semplice */
```

Esempi di accesso ai frames(2)

```
MIKE ?- all hobbies of tom are What.
What = [skiing, photography]
   /* accesso diretto a tutte le soluzioni */

MIKE ?- the hobbies of tom is X.
X = skiing
More Solutions (y/n) :y

X = photography
More Solutions (y/n) :y
no
   /* in bactracking si reperiscono tutti i
   valori di una lista */

MIKE ?- the hobbies of tom is skiing.
   /* verifica di una */
yes /* soluzione */

MIKE ?- all hobbies of tom are [X,
photography].
X = skiing   /* accesso ad una soluzione */

MIKE ?- all hobbies of tom are [photography,
X].
No /* l'ordine è sbagliato */
```

Modifica di frame

2 modi diversi, che possono apparire nel lato destro di una regola forward oppure al top level:

note the <slot> of <object> is <filler-or-list-of-fillers>.

note (<object> <instance-or-subclass-of> <class> with
    <slot1>: <filler-or-list-of-fillers1>,
    ...
    <slotn>: <filler-or-list-of-fillersn>).

La prima forma è usata per modificare o creare slot di un oggetto esistente.

La seconda crea un nuovo oggetto.

Esempi di modifica

```
MIKE ?- note joseph instance_of person with
   age: 38,
   hobbies: [swimming, tennis].
yes

MIKE ?- the age of joseph is X.
X = 38

MIKE ?- note the age of joseph is 49.
Yes    /*cambiamento distruttivo */

MIKE ?- the age of joseph is X.
X = 49

MIKE ?- all hobbies of joseph are What.
What = [swimming, tennis]

MIKE ?- note the hobbies of joseph is [music,
badminton]./* cambiamento distruttivo */
yes

MIKE ?- all hobbies of joseph are What.
What = [music, badminton].
/* i vecchi valori sono andati persi*/

MIKE ?- note the heigth of joseph is 6.
    /* aggiunge uno slot e relativo filler */

MIKE ?- the heigth of tom is K.
K = 6
```

Frames avanzati usando gli slot facets

Con i facets possiano specificare meglio le caratteristiche di uno slot come il tipo, la cardinalità, il tipo di ereditarietà e gli eventuali demoni ad esso associati

Tipi di facet:

- Value serve a contenere il valore dello slot: equivale allo slot standard, senza facet.

- Inheritance definisce come vengono ereditati i valori dello slot. Ci sono due modi:
    - supersede: quello standard, in cui i valori di un oggetto sovrascrivono quelli ereditati
    - merge: i valori di un oggetto si aggiungono a quelli ereditati.

- Type specifica il tipo che il contenuto di uno slot dovrebbe avere. Può essere un tipo Prolog (intero, atomo, lista), una classe o una lista di valori ammessi (produce solo un warning).

- Cardinality indica il numero (o il range) di valori che uno slot dovrebbe avere (anche questo produce solo un warning).

- Change_rule e acces_rule sono demoni: specificano il codice che deve essere eseguito nel caso di modifica di uno slot oppure di solo accesso.

Sintassi dei facet

```
<object> <instance-or-subclass-of> <class> with
    <slot1>: [<facet-filler-pair1a>,
              <facet-filler-pair1b>,
              ...
              <facet-filler-pair1n>],
    <slot2>: ...,
    ...
    <slotn>: [<facet-filler-pairna>,
              <facet-filler-pairnb>,
              ...
              <facet-filler-pairnn>].
```

<facet-filler-pair> può essere uno dei seguenti

- value: <filler-or-list-of-fillers>
- inheritance: <inheritance-type>
- type: <class-or-list-of-specific-choices>
- cardinality: <integer-or-range>
- change_rule: <change_rule-code>
- access_rule: <access_rule-code>

Vedi i manuali per i dettagli sulla sintassi dei singoli facet.

Esempio di acces_rule

```
tank subclass_of vessel with
   volume :
      [value : unknown,
       access_rule :
         (if
            the height of ?self is Height &
            the width of ?self is Width &
            the depth of ?self is Depth &
            prolog(Volume is
                 Height*Weight*Depth)
         then
            make_value Volume)].

small_tank instance_of tank with
   height : 10,
   width  : 10,
   depth  : 10.


-----------------------------


MIKE ?- the volume of small_tank is What.
What = 1000
yes.

MIKE ?- describe small_tank.
small_tank instance_of tank with
   height : 10,
   width  : 10,
   depth  : 10,
   volume : 1000.
yes.
```

Sintassi delle regole forward

```
rule <rule-name> forward
    if
        <condition1> &
        <condition2> &
        ...
        <conditionn>
    then
        <action1> &
        <action2> &
        ...
        <actionn>.
```

Possono contenere anche disgiunzioni (con 'or') nella parte sinistra.

Esempi

```
rule example forward
   if              /* premises are */
      a & b        /* a and b */
   or              /* OR */
      c & d        /* c and d */
   then
      halt.        /* then halt interpreter */
```

Condizioni ammissibili per le regole forward

- <wm-pattern> verifica se il pattern è presente nella wm. Il pattern può essere:
    - un atomo, come hi_there
    - una stringa, come 'it is raining'
    - una lista, come [once, upon, a, time]
    - una formula atomica Prolog, come goal(refine)

  Gli ultimi due possono contenere variabili.

- <frame-access>può essere uno dei modi visti in precedenza

- deduce <wm-pattern> fanno partire il backward chaining
- deduce <frame-access> per provare l'argomento

- -- <wm-pattern> verifica che il pattern sia assente dalla wm. ('--' è il simbolo di negazione in MIKE)
- -- <frame-access> verifica che il frame pattern specificato non può essere trovato nella frame memory

- <var-or-number> <math-operator> <var-or-number>

- forall(<wm-pattern1>, <wm-pattern2>) verifica che tutte le variabili che fanno match con la wm in <wm-pattern1>, facciano match con la wm anche in <wm-pattern2>.

Condizioni ammissibili per le regole forward

- prolog(<goal>)
- prolog((<goal1>,<goal2>,...,<goaln>))

i programmi MIKE possono contenere anche normali clausole Prolog.

- <query-template> receives_answer <answer>
  verifica se è già stata effettuata una certa query all'utente. Se <answer> è una variabile la istanzia con la risposta.

Azioni ammissibili per le regole forward

- add <wm-pattern>
- remove <wm-pattern>

- halt

- note the <slot> of <object> is <filler-or-list-of-fillers>
- note (<object> <instance-or-subclass-of> <class> with
  <slot1>: <filler-or-list-of-fillers1>,
  ...
  <slotn>: <filler-or-list-of-fillersn>).

- query <query-template> receives_answer<answer-template>
permette di fare una domanda all'utente durante la computazione

- announce <announcement-list>
il contenuto della lista è stampato a schermo

- ask_menu(<object>, <relation>, <list>)
mostra all'utente un menu da cui scegliere una o più opzioni (vedi i manuali per i dettagli)

- <var> := <math-expression> esegue un calcolo assegna il risultato a <var>

- prolog(<goal>)
- prolog((<goal1>,<goal2>,...,<goaln>))

Sintassi delle regole backward

```
rule <rule-name> backward
      if
            <condition1> &
            <condition2> &
            ...
            <conditionn>
      then
            <conclusion>.
```

Le condizioni ammissibili per le regole backward sono le stesse di quelle forward a cui va aggiunta

query <query-template> receives_answer <answer-template>

che esegue una query all'utente.

Le conclusioni ammissibili per le regole backward sono:
- un <wm-pattern>
- un <frame-access>

Se le condizioni sono tutte verificate, allora la conclusione (singola) è 'dedotta' ma la conclusione stessa non è aggiunta alla wm o fm.

Una certa regola viene invocata da deduce(<conclusion>) nel caso in cui <conclusion> unifichi con la conclusione della regola in questione.
Se nelle condizioni vi sono dei deduce, la catena backward prosegue, altrimenti si ferma.

Interazione con l'utente attraverso query

Le query possono comparire nelle parti sinistre delle regole backward e nelle parti destre delle regole forward.

Esempi:

- `query  the age of fred receives_answer A`
  aggiorna il frame di fred
- `query  the age of fred is 49 receives_answer yes`
  la condizione è verificata se l'utente risponde sì e viene messo 49 nello slot age di fred
- `query  'What ailments does fred show?' receives_answer N`
  stampa la stringa e mette la risposta in N
- `query  [please, enter, the, outcome, of, the, X, test] receives_answer OUTCOME`
  permette di variare run-time la domanda
- `query  'Are you happy today?' receives_answer no.`
  la condizione è verificata se l'utente risponde no

Sintassi:
query<question-template>receives-answer<answer-template>

<question-template> può essere uno dei seguenti:

- una stringa fra apici 'What is your name?'
- una lista eventualmente con variabili, come [please, perform, test, T, on, the, patient]
- un "frame-access pattern", come the age of fred is 49
- uno "short-frame-access specifier", come the age of fred

Query (2)

<answer-template> può essere uno dei seguenti:

- yes
- no
- una variabile Prolog, come X o WHAT
- un termine Prolog privo di variabili (è trattato come una costante arbitraria)

Quando la query è nella parte destra di una regola forward l'<answer-template> deve essere una variabile non istanziata, in quanto non è una condizione da verificare ma una azione da compiere il cui risultato è l'istanziazione della variabile.

Le risposte, oltre a dare luogo ad aggiornamenti della frame memory, vengono registrate nella wm come:

<question-template> receives-answer <answer-template>

e possono essere usate come premesse sia per le regole backward che forward.

Al prompt della query l'utente può anche rispondere con how o why (in MIKE 2.50 con how solo nelle query con risposta sì/no, per le quali viene presentato un menu con le alternative: yes, no, how, why).

Spiegazioni how e why

Rispondendo how ad una domanda oppure scegliendo dal menu principale ’Working memory display’, MIKE mostra l’elenco di elementi nella wm. Scegliendone uno, viene visualizzato un ’proof tree’ che mostra ricorsivamente (fino ad una certa profondità) le condizioni che hanno portato a quella conclusione.

```
+ Conclusion from rule eliminate -----------------
¦ likely(laryngitis)
¦ +-goal(discriminate)
¦ +-possible_instance(laryngitis)
¦ ¦ +-goal(refine)
¦ ¦ +-possible(upper_rtd)..
¦ ¦ +-laryngitis instance_of upper_rtd
¦ ¦ +---possible(laryngitis)
¦ ¦ +-deduce(passes_phys_sign_test(laryngitis))
¦ +-deduce(passes_discriminating_test(laryngitis))
+-------------------------------------------------
```

Rispondendo why ad una domanda, viene presentato un testo esplicativo relativo alla domanda. Queste risposte preconfezionate vengono specificate usando l’operatore explained by con la sintassi:

<question-template> explained_by <text-list>.

<text-list> è una lista composta di atomi, stringhe ed eventualmente variabili che viene mostrata all'utente quando risponde con why alla domanda <question-template>.
Questo comportamento è diverso da quello tradizionale di why, che permette di risalire nell'albero della dimostrazione.

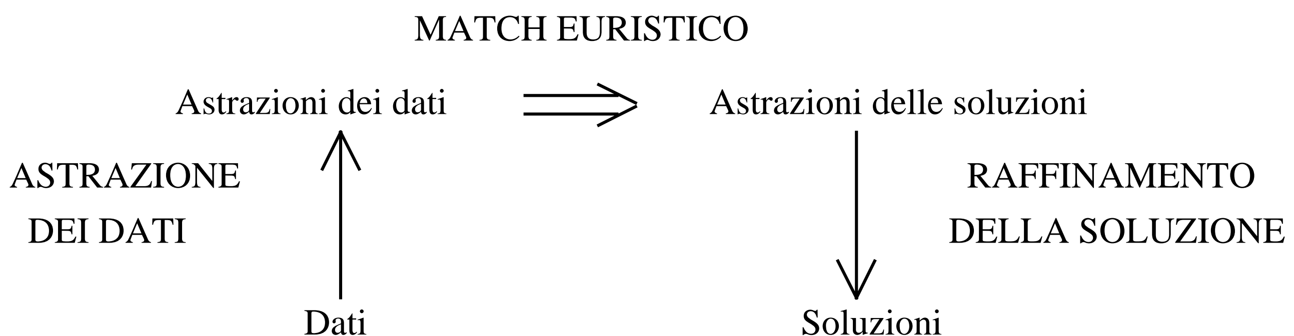
Tracing

MIKE offre una serie di opzioni per il tracing, che sono selezionabili dal menu principale:

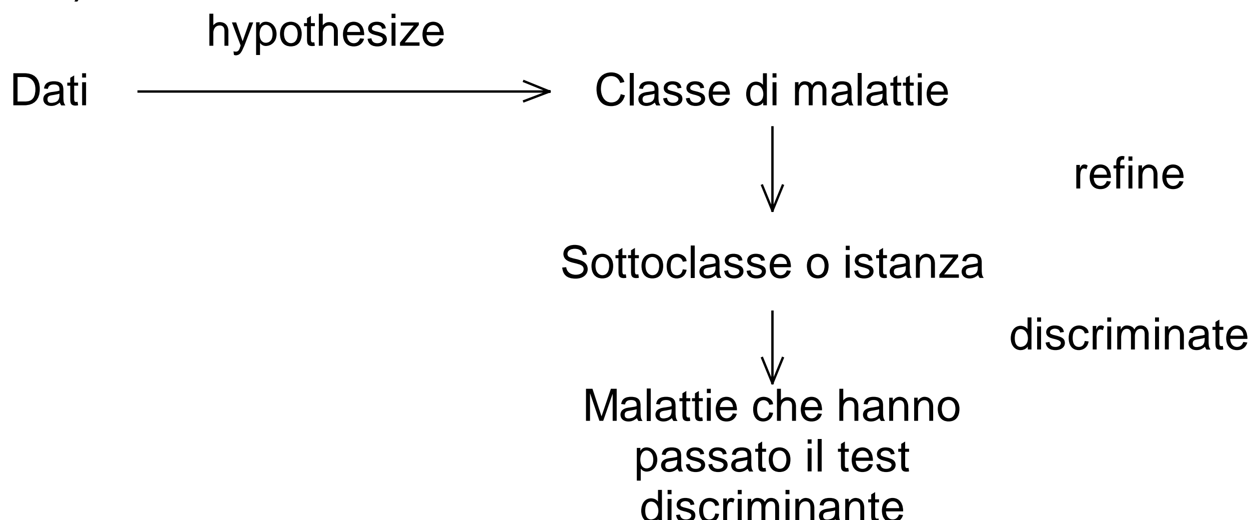
- show conflict set: mostra tutte le regole che sono applicabili ad ogni ciclo dell’interprete
- show refractoriness: mostra tutte le regole applicabili dopo che la strategia di risoluzione del conflitto ’refractoriness' è stata applicata
- show recency: mostra tutte le regole applicabili dopo l'applicazione della 'recency'
- show specificity: mostra tutte le regole applicabili dopo l'applicazione della 'specificity'
- show new working memory elements or frame changes: mostra gli elementi che sono stati appena aggiunti alla wm o i cambiamenti dei frame
- show chosen rule: mostra la regola scelta per l'applicazione
- show backward chaining: mostra una regola backward nel momento in cui è invocata
- show outcome of backward chaining: mostra se le condizioni della regola hanno avuto tutte successo o meno
- show single stepping: mostra l'esecuzione passo per passo (al prompt premere 'h' o '?' per maggiori dettagli)
- show history on request: memorizza la storia dell'esecuzione. Al termine, l'utente può richiederla da menu principale e vederla sotto forma di tabella in cui sono indicate, per ogni ciclo, le regole applicabili (indicate con '+') e quella scelta (indicata con '*').

Esempio di classificazione euristica: diagnosi medica

La classificazione euristica è un metodo per la soluzione di problemi in cui lo spazio delle soluzioni può essere preenumerato.



Nel seguito vedremo un sistema per la diagnosi di malattie dell'apparato respiratorio scritto con MIKE. Analizzando Il metodo di soluzione al '**livello della conoscenza**' ('knowledge level'), si vede che ricade nel modello classificazione euristica.



Analisi dei passi

In questo sistema è assente il passo di astrazione dei dati, c'è invece il passo di match euristico (costituito da hypothesize) e di raffinamento delle soluzioni (costituito da refine e da discriminate).

Durante il passo **hypothesize** si cerca una classe di malattie tale che almeno uno dei sintomi del paziente sia fra gli indicatori della classe.

Nel seguente passo **refine** si parte dalla classe identificata in precedenza e si scende nella tassonomia di malattie fino ad arrivare ad un insieme di istanze.

- Si considera una sottoclasse se almeno uno dei suoi indicatori è fra i sintomi del paziente.
- Si considera una istanza se supera il test dei 'physical signs', ovvero si verifica se almeno uno dei segni fisici della malattia è fra i segni che mostra il paziente

Nel passo **discriminate** si considerano tutte le istanze trovate al passo precedente e per ciascuna si chiede all'utente il risultato di un test discriminante specifico.

Implementazione in MIKE

Il programma si chiama DIAGNOSI.PL ed è nella directory di MIKE. E' costituito da 6 parti:

- Un database dei pazienti, rappresentati con i frames, contenente 5 pazienti.

- Un database delle malattie, anch'esse rappresentate con i frames.

- Un gruppo di regole forward che controllano il passaggio da un passo all'altro.

- Un gruppo di regole forward che effettuano la diagnosi vera e propria (una regola per passo tranne refine che ne ha due, una per le sottoclassi e una per le istanze).

- Un gruppo di regole backward che sono usati come filtri, ovvero per scartare le malattie che non soddisfano certi requisiti.

- Un gruppo di spiegazioni why.

Codice

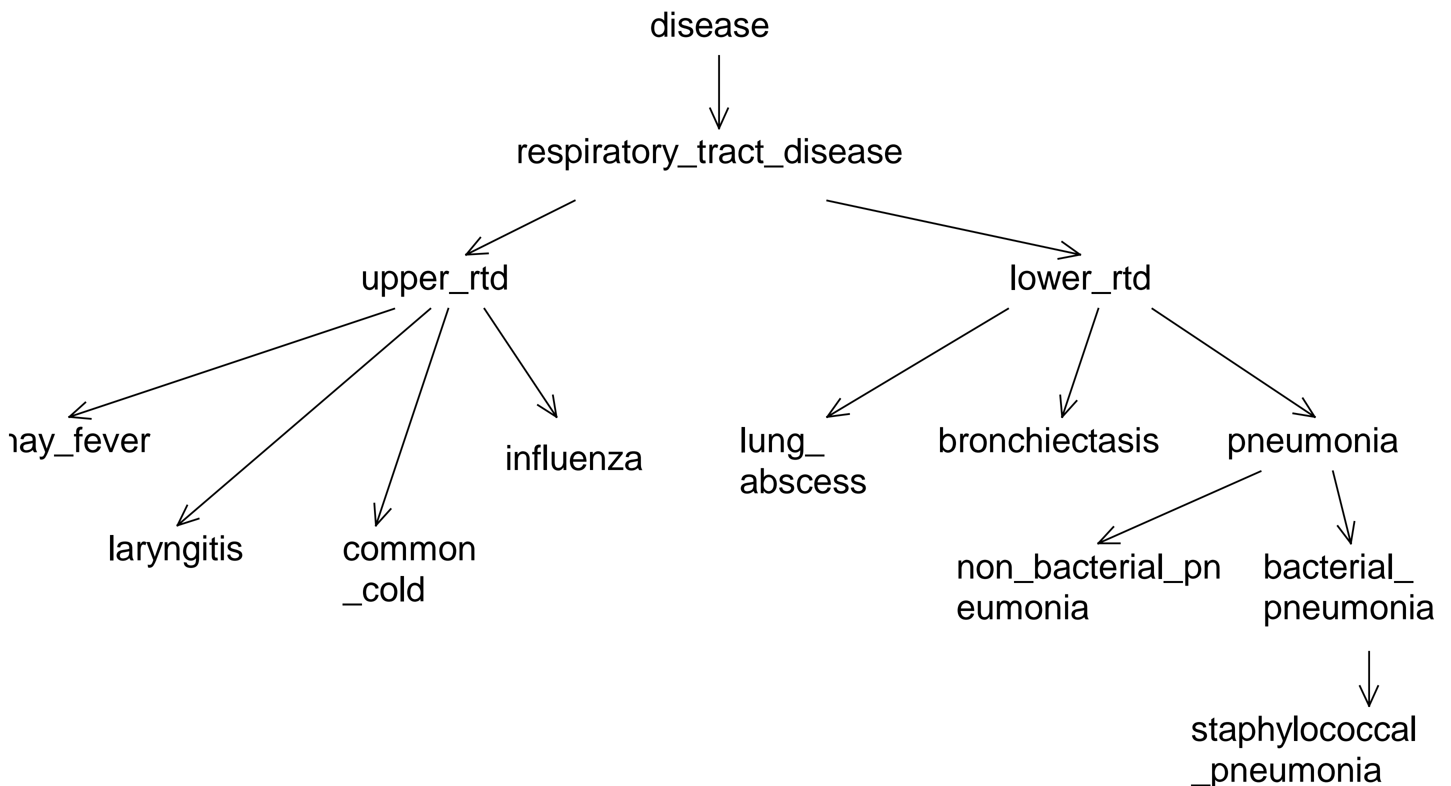
Pazienti:

```
bob instance_of patient with
    symptoms: [sneezing, runny_nose],
    history: [],
    signs: [red_itchy_watery_eyes].
```

Malattie:

```
hay_fever instance_of upper_rtd with
    indicators: [sneezing, runny_nose],
    physical_signs: [red_itchy_watery_eyes],
    typical_histories_or_contexts: [],
    discriminators:
[positive_reaction_to_allergens].
```

Tassonomia di malattie:



Esempio di esecuzione

Facciamo partire il forward chaining da menu scegliendo ’Fresh start’ dal relativo sottomenu. Questo cancella la wm e aggiunge l’atomo start.

I ciclo

Nel Conflict Set c'è l'unica regola:

```
rule init forward
  if
    start
  then
    remove start &
    query
      the name of patient
    receives_answer
      _15630 &
    add goal(hypothesize).
```

La regola fa partire il passo di hypothesize e chiede il nome del paziente all'utente.

Nuovi elementi della wm o cambiamenti dei frames::

```
the name of patient receives_answer bob
the name of patient is bob
goal(hypothesize)
```

Il ciclo

Il Conflict Set è:

```
rule switch_strategies_1 forward
  if
    goal(hypothesize)
  then
    remove goal(hypothesize) &
    add goal(refine).

rule ordinary_diagnosis forward
  if
    goal(hypothesize) &
    the name of patient is bob &
    the symptoms of bob is sneezing &
    respiratory_tract_disease subclass_of disease &
    --possible(respiratory_tract_disease) &
    the indicators of respiratory_tract_disease is
sneezing
  then
    add possible(respiratory_tract_disease).

rule ordinary_diagnosis forward
```

uguale alla precedente ma il sintomo che viene considerato è runny_nose

Viene scelta la seconda.
Nuovi elementi della wm o cambiamenti dei frames:
`possible(respiratory_tract_disease)`

III ciclo

Il Conflict Set è:

```
rule switch_strategies_1 forward
  if
    goal(hypothesize)
  then
    remove goal(hypothesize) &
    add goal(refine).
```

Cambio di strategia: passo di refine, si scende nella gerarchia di malattie

Nuovi elementi della wm o cambiamenti dei frames:

```
goal(refine)
```

IV ciclo

Vengono fatte partire le regole backward presenti negli antecedenti delle regole forward che hanno le altre condizioni verificate

```
<- ? allowable(upper_rtd)
<-  ? the name of patient is _15770 & the
indicators of upper_rtd is _15798 & the symptoms of
_15770 is _15798
<-  ? the name of patient is _15770   yes,_15770=bob
<-  ? the indicators of upper_rtd is _15798
       yes, _1798=fever, dry_cough, sneezing
<-  ? the symptoms of bob is fever       no
<-  ? the symptoms of bob is dry_cough   no
<-  ? the symptoms of bob is sneezing    yes
```

```
<- ? allowable(lower_rtd)
<-  ? the name of patient is _15770 & the
indicators of lower_rtd is _15798 & the symptoms of
_15770 is _15798
<-  ? the name of patient is _15770   yes,_15770=bob
<-  ? the indicators of lower_rtd is _15798
       yes, _15798=productive_cough,
       breathlessness, fever
<-  ? the symptoms of bob is productive_cough    no
<-  ? the symptoms of bob is breathlessness      no
<-  ? the symptoms of bob is fever               no
```

Il Conflict Set è:

```
rule switch_strategies_2 forward
  if
    goal(refine)
  then
    remove goal(refine) &
    add goal(discriminate).

rule refinement_to_subclass forward
  if
    goal(refine) &
    possible(respiratory_tract_disease) &
    upper_rtd subclass_of respiratory_tract_disease
&
    --possible(upper_rtd) &
    deduce allowable(upper_rtd)
  then
    announce
      [just refined down to subclass ,upper_rtd] &
    add possible(upper_rtd).
```

Il filtro recency non elimina nessuna regola mentro quello specificity elimina la prima regola perché meno specifica quindi viene applicata la seconda.

Nuovi elementi della wm o cambiamenti dei frames:

```
possible(upper_rtd)
```

V ciclo

Vengono fatte partire le regole backward presenti negli antecedenti delle regole forward che hanno le altre condizioni verificate

Viene verificato di nuovo lower_rtd che torna a fallie:

```
<- ? allowable(lower_rtd)
```

Inoltre viene tentata la verifica dei physical_sign per le istanze di upper_rtd, e solo quella per hay_fever ha successo

```
<- ? passes_phys_sign_test(hay_fever)    yes
<- ? passes_phys_sign_test(laryngitis)   no
<- ? passes_phys_sign_test(common_cold)  no
<- ? passes_phys_sign_test(influenza)    no
```

Il Conflict Set è:

```
rule switch_strategies_2 forward

rule refinement_to_instance forward
  if
    goal(refine) &
    possible(upper_rtd) &
    hay_fever instance_of upper_rtd &
    --possible(hay_fever) &
    deduce passes_phys_sign_test(hay_fever)
  then
    announce
      [just passed physical sign for disease
instance ,hay_fever] &
    add possible_instance(hay_fever).
```

La regola di recency preferisce la seconda perché possible(upper_rtd) è stato aggiunto dopo.
Nuovi elementi della wm o cambiamenti dei frames:

```
possible_instance(hay_fever)
```

VI ciclo

Vengono ritentate alcune verifiche backward

```
<- ? allowable(lower_rtd)                yes
<- ? passes_phys_sign_test(hay_fever)    yes
<- ? passes_phys_sign_test(laryngitis)   no
<- ? passes_phys_sign_test(common_cold)  no
<- ? passes_phys_sign_test(influenza)    no
```

Il Conflict Set è:

```
rule switch_strategies_2 forward

rule refinement_to_instance forward
....
    hay_fever instance_of upper_rtd &
```

La refractoriness elimina la II regola perché è stata usata prima, quindi
Nuovi elementi della wm o cambiamenti dei frames:

```
goal(discriminate)
```

VII ciclo

Adesso siamo nel passo discriminate in cui si chiede all'utente il risultato di un test discriminante riguardo le malattie possibili

```
<- ? passes_discriminating_test(hay_fever)
<-  ? the discriminators of hay_fever is _15738 &
query[Upon further investigation, is there solid
evidence of ,_15738]receives_answer yes
<-  ? the discriminators of hay_fever is _15738
<-  ? query[Upon further investigation, is there
solid evidence of ,positive_reaction_to_allergens]
receives_answer yes
```

Rispondendo why alla domanda si ottiene:
```
Having worked our way down through the hierarchy of
disease classes and subclasses all the way down to
individual instances, we are at last in a position
to make a discrimination among all of the final
contenders by performing a critical test.  This
particular test, namely evidence of
positive_reaction_to_allergens
will help us home in on a final choice
```

La domanda viene riproposta e si risponde sì.

Il Conflict Set è:
```
rule switch_strategies_3 forward
  if
    goal(discriminate)
  then
    remove goal(discriminate) &
    halt.

rule eliminate forward
  if
    goal(discriminate) &
    possible_instance(hay_fever) &
    deduce passes_discriminating_test(hay_fever)
  then
    announce
      [a highly likely candidate after
discriminating test is ,hay_fever] &
    add likely(hay_fever).
```

Viene preferita la seconda per la specificity.
Nuovi elementi della wm o cambiamenti dei frames:
```
likely(hay_fever)
```

VIII ciclo

Se non ci sono altre malattie (oltre ad hay_fever), si finisce.

```
<- ? passes_discriminating_test(hay_fever)   yes
```

Il Conflict Set è:

```
rule switch_strategies_3 forward
rule eliminate forward

Refractoriness filter threw out the following rule:
    eliminate
```

Nuovi elementi della wm o cambiamenti dei frames:

```
halt
```

Il Forward chaining termina.

Malattie

```
respiratory_tract_disease subclass_of disease with
    indicators: [productive_cough, dry_cough,
breathlessness, sneezing,
        runny_nose, fever],
    physical_signs: [],
    typical_histories_or_contexts: [],
    discriminators: [].

upper_rtd subclass_of respiratory_tract_disease
with
    indicators: [fever, dry_cough, sneezing,
runny_nose],
    physical_signs: [],
    typical_histories_or_contexts: [],
    discriminators: [].

lower_rtd subclass_of respiratory_tract_disease
with
    indicators: [productive_cough, breathlessness,
fever],
    physical_signs: [],
    typical_histories_or_contexts: [],
    discriminators: [].

pneumonia subclass_of lower_rtd with
    indicators: [productive_cough, fever,
systemic_upset,
    purulent_sputum, mucoid_sputum],
    physical_signs: [],
    typical_histories_or_contexts: [],
    discriminators: [].
```

```
bacterial_pneumonia subclass_of pneumonia with
    indicators: [purulent_sputum],
    physical_signs: [],
    typical_histories_or_contexts: [],

    discriminators: [].
non_bacterial_pneumonia subclass_of pneumonia with
    indicators: [mucoid_sputum],
    physical_signs: [],
    typical_histories_or_contexts: [],
    discriminators: [].

staphylococcal_pneumonia instance_of
bacterial_pneumonia with
    indicators: [fever, purulent_sputum, malaise ],
    physical_signs: [crepitations ],
    typical_histories_or_contexts:
[previous_respiratory_disease],
    discriminators: [gram_positive_cocci].

bronchiectasis instance_of lower_rtd with
    indicators: [fever, systemic_upset,
purulent_sputum],
    physical_signs: [finger_clubbing, halitosis,
breathlessness, cyanosis],
    typical_histories_or_contexts:
[previous_lung_disease],
    discriminators: ['bronchiectasis seen on
bronchogram'].

lung_abscess  instance_of lower_rtd with
    indicators: [fever, systemic_upset,
purulent_sputum],
    physical_signs: [consolidation, halitosis,
finger_clubbing],
    typical_histories_or_contexts:
[unresolving_pneumonia],
    discriminators:
[pulmonary_cavitation_with_fluid_levels].
```

```
hay_fever instance_of upper_rtd with
    indicators: [sneezing, runny_nose],
    physical_signs: [red_itchy_watery_eyes],
    typical_histories_or_contexts: [],
    discriminators:
[positive_reaction_to_allergens].

laryngitis instance_of upper_rtd with
    indicators: [fever, dry_cough, malaise],
    physical_signs: [hoarse_voice],
    typical_histories_or_contexts: [],
    discriminators: [inflamed_larynx ].

common_cold instance_of upper_rtd with
    indicators: [runny_nose, sneezing, fever],
    physical_signs: [headache],
    typical_histories_or_contexts: [],
    discriminators:
[negative_reaction_to_allergens].

influenza instance_of upper_rtd with
    indicators: [fever, dry_cough, malaise],
    physical_signs: [discomfort],
    typical_histories_or_contexts: [],
    discriminators:
[sore_throat_and_persistent_dry_cough].
```

Regole di controllo

```
/* control rules */
rule init forward
    if
        start
    then
        remove start &
        query the name of patient receives_answer X
&
        add goal(hypothesize).
/* see next comment for explanation of 'goals' */

rule switch_strategies_1 forward
    if
        goal(hypothesize)
    then
        remove goal(hypothesize) &
        add goal(refine).

rule switch_strategies_2 forward
    if
        goal(refine)
    then
        remove goal(refine) &
        add goal(discriminate).

rule switch_strategies_3 forward
    if
        goal(discriminate)
    then
        remove goal(discriminate) &
        halt.
```

Regole di diagnosi

```
/* forward chaining diagnosis rules */
rule ordinary_diagnosis forward
    if
        goal(hypothesize) &
                        /* in 'hypothesis' mode? */
        the name of patient is N &
                        /* retrieve name */
        the symptoms of N is Symp &
            /* now find ANY symptom Symp */
        D subclass_of disease &
            /* and any category of disease...  */
        --possible(D) &
            /*that we haven't suggested already...*/
        the indicators of D is Symp
            /* which might be indicated by Symp */
    then
        add possible(D).
            /* place it in working memory */

rule refinement_to_subclass forward
    if
        goal(refine) &
        possible(DiseaseClass) &
            /* given this candidate */
        Subclass subclass_of DiseaseClass &
            /* find a subclass of it...  */
        --possible(Subclass) &
            /* which we haven't dealt with yet */
        deduce allowable(Subclass)
    then
        announce ['just refined down to subclass'
                ,Subclass] &
        add possible(Subclass).
            /* if so, add to set of 'possibles' */
```

```
rule refinement_to_instance forward
/* as above, but only for instances */
    if
        goal(refine) &
        possible(DiseaseClass) &
            /* given this candidate */
        Disease instance_of DiseaseClass &
            /* find an instance of it...  */
        --possible(Disease) &
            /* which we haven't dealt with yet */
        deduce passes_phys_sign_test(Disease)
            /* see if it passes further tests */
    then
        announce ['just passed physical sign for
            disease instance ',Disease] &
        add possible_instance(Disease).
            /* if so, add to set of 'possibles' */

rule eliminate forward
    if
        goal(discriminate) &
        possible_instance(X) &
        deduce passes_discriminating_test(X)
    then
        announce ['a highly likely candidate after
            discriminating test is ',X] &
        add likely(X).
```

Regole filtro backward

```
/* ------ backward chaining 'filters'------------*/

rule allowable_1 backward
    if
        the name of patient is N &
        the indicators of Disease is Ind &
        the symptoms of N is Ind
        /*one symptom in common with indicators*/
    then
        allowable(Disease).


rule necessary_sign_test backward
    if
        the physical_signs of Disease is Sign &
                            /* get any sign */
        the name of patient is N &
        the signs of N is Sign
                    /* see if patient has got it */
    then
        passes_phys_sign_test(Disease).

rule discriminatory_diagnosis backward
    if
        the discriminators of X is D &
        query ['Upon further investigation, is
            there solid evidence of ', D]
            receives_answer yes
    then
        passes_discriminating_test(X).
```