Logica Proposizionale
– ESERCITAZIONE –

Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Brescia
Alessandro Saetti
saetti@ing.unibs.it

Materiale per il Corso di Intelligenza Artificiale
(Prof. Alfonso Gerevini)

Esercizio 1

Se l'unicorno e' un animale mitico e' immortale
$\texttt{mitico} \rightarrow \texttt{immortale}$

Se non e' mitico allora e' un mammifero mortale
$\neg\texttt{mitico} \rightarrow \neg\texttt{immortale} \wedge \texttt{mammifero}$

Se e' immortale o e' un mammifero allora ha un corno
$(\texttt{immortale} \vee \texttt{mammifero}) \rightarrow \texttt{cornuto}$

L'unico e' magico se ha un corno
$\texttt{cornuto} \rightarrow \texttt{magico}$

Dimostrare per refutazione con la strategia basata su insieme di supporto quanto segue.

- *L'unicorno è mitico?*
- *L'unicorno è magico?*
- *L'unicorno ha un corno?*

Applicazioni Dimostratori di Teoremi

- Dimostrazione, supervisione e verifica di teoremi
  - Teorema di incompletezza di Godel
- Verifica software, hardware, protocolli di Sicurezza
  - Sistema di controllo dello space-shuttle
  - Divisione Floating-Point per il processore AMD5k86
  - Protocollo RSA
- Generazione di software
  - Programmi per comandare satelliti

Otter

- Otter = Organized Techniques for Theorem-proving and Effective Research
- È basato su logica del primo ordine
- Sistema semi-automatico
- Funziona anche in batch mode: `otter < input > output`
- **Indirizzi Internet utili**:

  Home Page di Otter: `http://www.mcs.anl.gov/AR/otter/`

  Manuale: `http://www-unix.mcs.anl.gov/AR/otter/otter3_manual.pdf`

Sintassi

- Nomi ordinari: stringa composta da alfanumerici, _
- Caratteri speciali:
  - %: commenti
  - . : fine espressione
  - ,() : punteggiatura e simboli di raggruppamento
- Nomi che iniziano con $ sono riservati (Es. $AND)
- Lettere minuscole differenti da lettere maiuscole

Sintassi - Clausole

Operatori	Simbolo	Priorità
Negazione	-	1
Disgiunzione	|	3

- Forma prefissa: `|( -(a), |( -(b), c) ).`
- Forma infissa: `-a | -b | c.`
- Spaziatura obbligatoria prima di "–"
- Tutte le clausole terminano con "."

Sintassi - Formule

Operazione	Simbolo	Priorità
Negazione	-	1
Congiunzione	&	2
Disgiunzione	|	3
Implicazione	->	4
Equivalenza	<->	4

- Tutte le formule terminano con "."
- Otter trasforma le formule in clausole

Liste di Clausole/Formule

Otter utilizza la strategia basata su insieme di supporto

- usable: clausole/formule usate per l'inferenza
- sos: clausole/formule che partecipano alla ricerca
- passive: clausole/formule usate per controllare il processo in dimostrazioni difficili (sussunzioni/unit-conflict)
- demodulators: uguaglianze utilizzate per la riscrittura di clausole/formule inferite

Comandi e File di Input

- Otter riconosce due tipi di opzioni:
  - flag: opzioni booleane assegnate con `set` e `clear`
    ES: set(binary_res);
  - parametri: valori interi assegnati con `assign`
    ES: assign(max_seconds,100).

- Le liste iniziano con `list(list_name).`, `formula_list(list_name).`

  list_name è `usable`, `sos`, `demodulators`, oppure `passive`

- Le liste terminano con `end_of_list.`

Comandi e File di Input

Comando	Descrizione
include(file_name)	include file_name nel file corrente
op(precedenza,tipo,nome)	dichiara un operatore
make_evaluable(simb,simb_valutato)	dichiara un alias di simbolo_valutato
set(flag_name)	setta il flag flag_name
clear(flag_name)	resetta il flag flag_name
assign(parameter_name,integer)	parameter_name=integer
list(list_name)	lista delle clausole
formula_list(list_name)	lista delle formule
weight_list(weight_list_name)	lista dei pesi
lex(symbol_list)	assegna un ordinamento ai simboli

Modalità Automatica

- Impostata tramite il flag auto: `set(auto)`
- set(auto) DEVE essere il primo comando
- La KB è interamente in `list(usable)`
- Otter decide:
  - regole di inferenza;
  - strategia di ricerca;
  - lista sos.

Esempio

C'era una volta due talpe che vivevano in una buca. Nessuna delle due condivideva la tana con l'altra.

```
set(auto).
list(usable).
% Ogni talpa vive in un buco.
ViveT1B1.
ViveT2B1.
% Nella buca vive al piu' una talpa.
-ViveT1B1 | -ViveT2B1.
end_of_list.
```

Regole di inferenza

- **Binary resolution** (flag `binary_res`)
  Risoluzione con due clausole.

  $$\frac{infermiera(Roberta), \neg infermiera(Roberta)}{False}$$

- **Unit resulting resolution** (flag `ur_res`)
  Il risultato è una clausola unitaria.

  Data una clausola con n termini e $n-1$ clausole unitarie le risolve in un colpo solo.

  $$\frac{\neg P(x,y) \vee Q(y,z) \vee \neg R(u,v) \vee S(x,z),\ \ P(John,Mary),\ \neg Q(Mary,Cats), R(Birds,Mice)}{S(John,Cats)}$$

- **Hyper-Resolution** (flag `hyper_res`)
  Una clausola ha almeno un letterale negativo, mentre le altre non ne hanno. Il risultato non contiene letterali negativi.

  $$\frac{\neg P(x,y) \vee Q(x) \vee \neg R(y)\ \ \ P(Spock,Data)\ \ \ R(z) \vee S(z)}{Q(Spock) \vee S(Data)}$$

Regole di inferenza

- **Paramodulation**
  Uno dei sottotermini del predicato di uguaglianza deve unificare con un sottotermine dell'altra clausola.

  $$\frac{difference(two\_squared, b)\ \ \ equal(two\_squared, four)}{difference(four, b)}$$

  - **Paramodulation from the given clause** (flag `para_from`): la given clause contiene un predicato di uguaglianza
  - **Paramodulation into the given clause** (flag `para_into`): il predicato di uguaglianza non è contenuto nella given clause, ma il risultato dell'inferenza è la sostituzione di un parametro della given_clause.

Caratteristiche

- La KB è specificata tramite clausole oppure formule del primo ordine
- Forward demodulation: semplifica le clausole inferite
- Backward Demodulation: semplifica le clausole esistenti
- Forward Sussumption: cancella le clausole inferite
- Backward Sussumption: cancella le clausole esistenti
- pesi e ordinamento lessicale

Algoritmo di Otter

```
while (sos non vuota & nessuna dimostrazione e' stata trovata)
begin
  given_clause := migliore_clausola(sos);
  usable := usable U given_clause;
  sos := sos - given_clause;
  new_clauses := inferisci(given_clause, clause_usables)
  if retention_test(new_clauses) then
     sos:=sos U new_clauses;
endwhile.
```

Esempio (cont.)

C'era una volta due talpe che vivevano in una buca. Nessuna delle due condivideva la tana con l'altra.

Utilizzare OTTER per dimostrare che quanto racconta questa storia non è possibile.

```
given clause #1: (wt=3) 3 [] ViveT2B1.
** KEPT (pick-wt=0): 4 [hyper,3,2,1].
-----> EMPTY CLAUSE at   0.00 sec ----> 4 [hyper,3,2,1] $F.
```

In conclusione, la dimostrazione è la seguente:

```
1 [] ViveT1B1.
2 [] -ViveT1B1| -ViveT2B1.
3 [] ViveT2B1.
4 [hyper,3,2,1] $F.
```

Esercizio 1

John lo sfregiato, Bill lo squartatore e Jack il duro vengono interrogati con l'accusa di aver rubato la cassetta delle elemosina della parrocchia.

Jack dice che il furto e' stato commesso da Bill.
$\texttt{FurtoBill} \leftrightarrow \texttt{VeritaJack}$

Bill si professa innocente.
$\neg\texttt{FurtoBill} \leftrightarrow \texttt{VeritaBill}$

John afferma di non aver compiuto il furto.
$\neg\texttt{FurtoJohn} \leftrightarrow \texttt{VeritaJohn}$

Solo uno dei tre dice la verità.
$\texttt{VeritaJack} \leftrightarrow \neg\texttt{VeritaBill} \wedge \neg\texttt{VeritaJohn}$
$\texttt{VeritaBill} \leftrightarrow \neg\texttt{VeritaJack} \wedge \neg\texttt{VeritaJohn}$
$\texttt{VeritaJohn} \leftrightarrow \neg\texttt{VeritaBill} \wedge \neg\texttt{VeritaJack}$

Chi è il ladro?

Esercizio 2

Sapendo che gli enigmi sono dei misteri
$\texttt{Enigma} \rightarrow \texttt{Mistero}$

...e che non ci sono indovinelli che non sono enigmi.
$\neg(\texttt{Indovinello} \wedge \neg\texttt{Enigma})$

...utilizzare il dimostratore di teoremi per determinare quali tra le seguenti affermazioni sono vere:

1. *I misteri sono indovinelli.*
2. *Non ci sono misteri che non sono enigmi.*
3. *Gli indovinelli sono misteri.*
4. *Non ci sono enigmi che non sono indovinelli.*
5. *Gli enigmi sono indovinelli.*

Esercizio 3

C'era una volta 3 talpe a 2 buche:

Ognuna delle tre talpe viveva in (almeno) una delle due buca.
$\texttt{ViveT1B1} \vee \texttt{ViveT1B2}$
$\texttt{ViveT2B1} \vee \texttt{ViveT2B2}$
$\texttt{ViveT3B1} \vee \texttt{ViveT3B2}$

Dimostrare che in una buca vivono più talpe.