Regole di inferenza (1)

- Eliminazione dell'implicazione (modus ponens): $\frac{a \rightarrow b,\ a}{b}$;
- Eliminazione degli and: $\frac{a_1 \wedge a_2 \wedge \ldots \wedge a_i \wedge \ldots \wedge a_n}{a_i}$;
- Introduzione di and: $\frac{a_1, a_2, \ldots, a_n}{a_1 \wedge a_2 \wedge \ldots \wedge a_n}$;
- Introduzione di or: $\frac{a_i}{a_1 \vee a_2 \vee \ldots \vee a_i \vee \ldots \vee a_n}$;
- Risoluzione unitaria: $\frac{a \vee b,\ \neg b}{a}$;
- Risoluzione: $\frac{a \vee b,\ \neg b \vee c}{a \vee c}$;

Regole di inferenza (2)

- Eliminazione di $\forall$: $\frac{\forall v \;\; a}{Sost(\{v/g\},a)}$ dove g é un termine ground;
- Eliminazione di $\exists$: $\frac{\exists v \;\; a}{Sost(\{v/g\},a)}$ dove g é un termine ground nuovo;
- Introduzione di $\exists$: $\frac{a}{\exists v \;\; Sost(\{g/v\},a)}$ dove g é un termine ground che compare in a;
- Modus Ponens generalizzato: $\frac{P'_1,\ldots,P'_n,\ (P_1,\ldots,P_n)\rightarrow Q}{Sost(\theta,Q)}$
  se $\exists\theta$ tale che $Sost(\theta,P'_i) = Sost(\theta,P_i) \;\; \forall i \in \{1,\ldots,n\}$;
- Risol. generaliz.: $\frac{P_1\vee\ldots\vee P_j\vee\ldots\vee P_m, Q_1\vee\ldots\vee Q_k\vee\ldots\vee Q_n}{Sost(\theta, P_1\vee\ldots\vee P_{j-1}\vee P_{j+1}\vee\ldots\vee P_m\vee Q_1\vee\ldots\vee Q_{k-1}\vee Q_{k+1}\vee Q_n)}$
  dove $\theta = Unify(P_j, \neg Q_k)$.

Refutazione

Refutazione: $((KB \wedge \neg P) \models Falso) \Leftrightarrow (KB \models P)$.

Per dimostrare P si assume P falso. Se la dimostrazione porta a una contraddizione la base di conoscenza implica P.

La refutazione (o dimostrazione per assurdo) mediante risoluzione è una procedura di inferenza completa.

Dimostratori di Teoremi

Strategie di risoluzione:

- **Preferenza dell'unità**: Si applica la risoluzione, dove una delle formule è costituita da un singolo letterale (clausola unitaria);
- **Risoluzione di Input**: Si applica la risoluzione, dove una delle formule è una formula di input (KB+query).
- **Insieme di Supporto**: Si applica la risoluzione tra una delle formule di sos e un'altra formula; il risultato dell'inferenza è aggiunto a sos.

NOTA: Se si sceglie il sos in modo tale che le restanti formule siano contemporaneamente soddisfacibili la risoluzione basata su sos è completa.

Applicazioni dei Dimostratori di Teoremi

- Dimostrazione di teoremi matematici

  ES: Il dimostratore di teoremi di Boyer-Moore è stato il primo a dimostrare il teorema di incompletezza di Godel.

- Aiuto nelle dimostrazioni
- Verifica di dimostrazioni
- Verifica del Software

  ES: L'algoritmo di criptazione a chiave pubblica RSA.

- Verifica di Hardware
- Sintesi di software
- Sintesi di hardware

Otter

```
otter < input > output
```

Indirizzi Internet utili:

Lucidi dellla prima lezione: http://prometeo.ing.unibs.it/ia/lucidi-fse.html

Introduzione a Otter: http://prometeo/ia/introtter.htm

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

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

Otter - Regole di inferenza (1)

- Binary resolution (flag `bin_res`)

  Date due clausole in contraddizione Otter deriva la clausola vuota.

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

- Unit resulting resolution (flag `ur_res`)

  Il risultato deve essere una clausola unitaria.

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

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

- Hyper-Resolution (flag `hyper_res`)

  Una clausola deve avere almeno un letterale negativo, mentre le altre clausole non devono averne.

  Il risultato non deve contenere 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 (2)

- Paramodulation

  Uno degli argomenti del predicato di uguaglianza deve unificare con un sottotermine dell'altra clausola.

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

  - 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

- Le asserzioni devono essere specificate o come clausole o come formule del primo ordine (clausole = disgiunzione di predicati le cui variabili sono implicitamente quantificate universalmente; formule = necessitano una quantificazione esplicita);
- 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

Liste di Clausole

- usable: clausole usate per l'inferenza;
- sos: clausole che partecipano alla ricerca;
- passive: clausole usate per sussunzione in avanti e unit conflict;
  NOTA: Usata per controllare il processo in dimostrazioni difficili.
- demodulators: uguaglianze utilizzate per la riscrittura di clausole inferite.

OSSERVAZIONE: La strategia del sos richiede di dividere le clausole in due insiemi, quelle con supporto (sos) e quelle senza (usable). All'inizio le sos sono le clausole supportate e le usable sono quelle non supportate.

Esempio di Liste di Clausole

```
list(usable).
-citta(x) | -citta(y) | -aereo(z) | -at(z,x) | at(z,y).
citta(Brescia).
citta(Milano).
aereo(a1).
-at(a1,Milano).
end_of_list.

list(sos).
at(a1,Brescia).
end_of_list.
```

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_clause=inferisci(given_clause, clause_usable)
  if retention_test(new_clause) then
     sos:=sos U new_clause;
endwhile.
```

Esempio di Hyuper Resolution

```
given clause #1: (wt=3) 6 [] at(a1,Brescia).
** KEPT (pick-wt=3): 7 [hyper,6,1,2,3,4] at(a1,Milano).
----> UNIT CONFLICT at   0.00 sec ----> 8 [binary,7.1,5.1] $F.
```

In conclusione, la dimostrazione è la seguente:

```
 1 [] -citta(x)| -citta(y)| -aereo(z)| -at(z,x)|at(z,y).
 2 [] citta(Brescia).
 3 [] citta(Milano).
 4 [] aereo(a1).
 5 [] -at(a1,Milano).
 6 [] at(a1,Brescia).
 7 [hyper,6,1,2,3,4] at(a1,Milano).
 8 [binary,7.1,5.1] $F.
```

Sintassi

Otter distingue tre tipi di stringhe:

- nomi ordinari: stringa composta da alfanumerici, \$, _.
- nomi speciali: stringa composta da $* + - / \backslash^{<} >='\sim:?@\&!;\#|$.
- nome quotato: stringa racchiusa tra “ ” o ’ ’.

Altri caratteri:

- %: commenti
- . : fine espressione
- ,()[]|: punteggiatura e simboli di raggruppamento.

Terminologia

I termini sono valutati come oggetti.

Costanti, variabili, simbolo di funzione n-aria applicato a n termini

ES: A, Michele, x, Nasodi(Michele), ecc...

Formule atomiche sono valutati come valori di verità.

Simbolo proposizionale, simboli di predicato n-ario

ES: Pippo, Fratello(Giovanni,Piero)

Termini

Il contesto determina il tipo (variabile, costante):

- **CLAUSOLE**: Il simbolo determina il tipo.
  Regola di default: le variabili iniziano con `u, v, w, x, y, z`.
  Se il flag prolog_style_variables é ON: le variabili iniziano con _ oppure con una lettera maiuscola (nome ordinario).
- **FORMULE**: Il simbolo é una variabile se e solo se è quantificato.

I nomi che iniziano con $ sono riservati.
`=, eq, Eq, EQ` sono predicati di uguaglianza.

Predicati/funzioni: simbolo predicato/funzione (argomenti separati da ,)
NO white-space tra simbolo di predicato/funzione e (.
x+y=1 non può essere il nome di una variabile

Clausole

Disgiunzione di letterali (atomo o la sua negazione).

Table 1: Lista di simboli logici per la costruzione di clausole.

Operazione	Simbolo	Priorità
Negazione	`-`	1
Disgiunzione	`|`	3

Le clausole possono essere scritte in forma:

prefissa `|( -(a), |( -(b), c) )`

infissa `-a | -b | c`

WHITE-SPACE prima di "-".

Formule (1)

Una formula é un insieme di termini e atomi combinati usando i simboli sotto riportati

Table 2: Lista di simboli logici per la costruzione di formule.

Operazione	Simbolo	Priorità
Negazione	`-`	1
Congiunzione	`&`	2
Disgiunzione	`|`	3
Implicazione	`->`	4
Equivalenza	`<->`	4
Quantificatore esistenziale	`exists`	-
Quantificatore universale	`all`	-

Formule (2)

Le formule possono essere scritte in forma:

infissa `$Quantified(all, x, y, exists, z, |(P(x,y), -(Q(z))))`

prefissa `all x all y exists z (P(x,y) | -Q(z)).`

Semplicazione della quantificazione:

`all x y exists z (P(x,y) | -Q(z)).`

WHITE-SPACE prima e dopo “all” ed “exists” e prima di “-”.

Comandi e File di Input (1)

Otter riconosce due tipi di opzioni:

- flag: opzioni booleane assegnate con `set` e `clear`
  ES: set(binary_res);
- parametri: valori interi asseganti con `assign`
  ES: assign(max_seconds,100).

La lista inizia con `list(list_name).`, `formula_list(list_name).`, oppure `weight_list(weight_list_name).` La lista termina con `end_of_list.`

- usable (clausole e formule)
- sos (clausole e formule)
- demodulators (clausole)
- passive (clausole e formule)

Comandi e File di Input (2)

Comando	Descrizione
include(file_name)	include file_name nel file corrente
op(precedenza,tipo,nome)	dichiara l'operatore nome
make_evaluable(simbolo,simbolo_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
skolem(symbol_list)	identifica le funzioni di skolem
lrpo_multiset_status(symbol_list)	stato di lrpo

Modalita' automatica

La modalitá automatica é impostata tramite il flag auto: `set(auto)`.

NOTE:
set(auto): primo comando

clausole in `list(usable)`

Otter decide:

- regole di inferenza;
- strategia di ricerca;
- lista sos.

Il Problema dei ladruncoli

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; Bill si professa innocente e John afferma di non aver compiuto il furto.
Sapendo che solo uno dei tre dice la verità, chi è il ladro.

Soluzione del problema dei ladruncoli

```
set(ur_res).

list(usable).
% le testimonianze dei tre ceffi
-furto(Bill) | verita(Jack).
furto(Bill) | verita(Bill).
furto(John) | verita(John).

% solo uno dei tre ceffi dice la verita
-verita(Jack) | -verita(Bill).
-verita(Jack) | -verita(John).
-verita(John) | -verita(Bill).
end_of_list.

list(sos).
-furto(John).
end_of_list.
```

Il Problema dell'insegnante e dell'infermiera

Roberta e Michele hanno entrambi un lavoro.
I loro lavori sono insegnante e infermiere.
Sapendo che l'infermiere viene svolto da un uomo, che lavoro svolge Roberta?

Soluzione del Problema dell'insegnante e dell'infermiere

```
set(ur_res).

list(usable).

HASAJOB(x, Nurse) | HASAJOB(x, Teacher).
HASAJOB(Roberta, x) | HASAJOB(Steve, x).

-HASAJOB(x, Nurse) | MALE(x).

FEMALE(x) | MALE(x).
-FEMALE(x) | - MALE(x).
FEMALE(Roberta).
MALE(Steve).
end_of_list.

list(sos).
-HASAJOB(Roberta, Teacher). % contraddizione
end_of_list.
```

Il Problema del Ristorante

Gianni, Mario e Diego sono gestore, cuoco e cameriere di un ristorante, non nell'ordine considerato. Gianni, amico del fratello di Mario, è più giovane di Mario e ha gli occhi neri. Il cameriere ha gli occhi azzurri, mentre il gestore è figlio unico ed è il più anziano dei tre.
Qual'è il lavoro di ciascuno?

Suggerimento

Informazione esplicite del problema:
Il gestore non ha fratelli. Mario ha fratelli.
Il cameriere ha gli occhi azzurri. Gianni non ha gli occhi azzurri.
Il gestore è il più vecchio. Gianni non è il più vecchio.

Informazioni implicite del problema:
Due persone non possono fare lo stesso lavoro.
ES: Gianni e Mario non possono fare entrambi il cameriere
Ogni persona deve fare almeno un lavoro.
ES: Gianni fa il cameriere o il cuoco o il gestore

Unica soluzione possibile:
Mario fa il cameriere, Gianni il cuoco e Diego il gestore.

Soluzione del problema del ristorante (1)

```
set(auto).

formula_list(usable).
% Due persone non possono fare lo stesso lavoro.
-cameriere(Gianni) | -cameriere(Mario).
-cameriere(Gianni) | -cameriere(Diego).
-cameriere(Mario)  | -cameriere(Diego).
-cuoco(Gianni) | -cuoco(Mario).
-cuoco(Gianni) | -cuoco(Diego).
-cuoco(Mario)  | -cuoco(Diego).
-gestore(Gianni) | -gestore(Mario).
-gestore(Gianni) | -gestore(Diego).
-gestore(Mario)  | -gestore(Diego).

% Ogni persona deve fare almeno un lavoro
cameriere(Gianni) | cameriere(Diego) | cameriere(Mario).
cuoco(Gianni) | cuoco(Diego) | cuoco(Mario).
gestore(Gianni) | gestore(Diego) | gestore(Mario).
```

Soluzione del problema del ristorante (2)

```
  % dati del problema
  all x y (gestore(x) -> -fratello(x,y)).
  exists y (fratello(Mario,y)).
  all x (cameriere(x) -> occhiazzurri(x)).
  -occhiazzurri(Gianni).
  all x (gestore(x) -> piuvecchio(x)).
  -piuvecchio(Gianni).

  % contraddizione dell'unica soluzione possibile
  -cameriere(Mario) | -cuoco(Gianni) | -gestore(Diego).

end_of_list.
```