POLITECNICO DI MILANO

V FACOLTÀ DI INGEGNERIA

Corso di laurea in Ingegneria Informatica

Progetto Algebre e Logica 2

Analisi di un theorem proover

Relatore: professoressa Alessandra Cherubini
Progetto di: Massimiliano Carrara
Matricola 667110

Anno accademico 2004-2005

Indice

Capitolo 1

Introduzione

MSPASS[1] è nato come miglioramento di SPASS, un noto theorem prover per logiche del prim'ordine, per l'interpretazione (attraverso un processo di traduzione) di formule modali, di formule per logiche descrittive (DL) e di formule per il calcolo relazionale. Per questo motivo il linguaggio di input di SPASS è stato esteso per permettergli di accettare formule modali, descrittive e relazionali. Un'analisi dettagliata di questo linguaggio e della sua sintassi è proposta nel Capitolo 2.

I costruttori messi a disposizione del linguaggio includono:

- gli operatori booleani standard: true, false, not, and, or, implies (subsumed by), implied (subsumes),equiv
- gli operatori multi-modali: dia e box
- gli operatori relazionali: comp (composizione), sum (somma relativa), conv (converso), id (identità), div (diversità), test, domrestr (restrizione del dominio) e ranrestr (restizzione del range)

MSPASS supporta gli assiomi modali, che devono essere veri in ogni possibile mondo, e la possibilità di specificare propietà valide per i singoli frame, al fine di applicare restrizioni con una granularità più fine.

1.1 Archiettura del sistema

L'attività di reasoning in MSPASS [1] viene effettuata in tre passaggi:

a Traduzione delle formule modali e relazionali in un insieme di formule del prim'ordine.

b Trasposizione nella forma a clausole.

c Risuluzione col metodo basato col "Metodo della Saturazione"[1].

Ovviamente un ruolo cruciale è svolto dal punto A. Perciò analizziamo più in dettaglio i metodi di traduzione disponibili , nella versione attuale, che sono:

- il metodo di traduzione relazionale[2]

- il metodo di traduzione funzionale[3].

- il metodo di traduzione funzionale ottimizzato[4].

- il metodo di traduzione semi-funzionale

A quento presentato fin'ora, tuttavia, bisogna applicare alcune restrizioni. Infatti i metodi di traduzione funzionale, funzionale ottimizzato e semi-funzionale sono disponibili solo per logiche multi-modali $K_{(n)}$ e per le loro estensioni.

1.1.1 Il metodo di inferenza

Le funzionalità di inferenza di MSPASS sono implementate attraverso una resuluzione basata sulla saturazione ed attraverso il calcolo superposizionale. In particolare:

- utilizza la risoluzione ordinata ed il calcolo superposizionale con ordinamento per la selezione

- supporta metodi di semplificazione dell'albero dei cammini: "tagli e condensa dei salti" (cioè taglia i sotto-alberi che non contengono nodi finali di interesse)

- ha un insieme considerevole di regole per la semplificazione e la riduzione

- supporta gli ordinamenti dinamici attraverso l'inferenza addizionale e la riduzione delle regole

[1] Il metodo risolutivo basato sulla saturazione (*saturation-based resolution*) è un raffinamento della *Refutazione con risoluzione* che si applica solo in presenza di teorie con identità.

[2] Questo è il metodo di default

1.2 L'algoritmo di deduzione

Il ciclo di inferenza di MSPASS[1] è controllato da due insiemi di clausole:

a l'insieme delle clausule utilizzabili.

b l'insieme delle clausole già utilizzate.

All'inizio tutte le clausole di input sono utilizzabili. Il theorem prover comincia scegliendo, seconodo alcune euristiche, una clausola dall'insime delle utilizzabili (insieme a) e spostandola in quello delle già utilizzate (insieme b). A questo punto sono applicate tutte le regole di inferenza alla clausola scelta ed all'insieme delle clausole già utilizzate (insime b). Le clausole che ne derivano saranno ovviamente figlie delle regole di riduzione, ad esempio della cancellazione per sussunzione, e le clausole non ridondanti sono unite all'insieme delle clausule usabili (insime a).

Se viene derivato il falso (cioè una clausola nulla) o se l'insieme delle clausole usabili è vuoto, il theorem prover si ferma avendo dedotto o una contraddizione o un insieme soddisfacibile di clausole.

Altrimenti, si riprocede con il calcolo appena effettuato e cioè si sceglie un'altra clausola dall'insieme a, la si sposta nell'insieme delle clausole non utilizzabili (insime b) e si ripete il ciclo.

L'euristica utilizzata da MSPASS sceglie la clausola con il numero minimo di simboli.

Tra le regole di inferenza lo "splitting" ha la priorità. Se quest'ultima è applicabile ad una clausola, una euristica è utilizzata per determinare quali sottoclassi sono generate e in quale ordine esse vanno considerate. L'attuale implementzaione dello "splitting" non separa le clausole di Horn e l'euristica salterà semplicemnete al primo (nell'ordine di come sono memorizzate internamente) letterale positivo della clauola scelta.

1.3 Alcune considerazioni finali

Come visto fin'ora MSPASS[1] è un reasoner per una larga classe di logiche modali e descrittive. Tuttavia esso non permette procedure di decisione per tutte le possibili logiche alle quali potremmo essere interessati (ad esempio PDL non è supportata).

Questo apparentemente può sembrare uno svanatggio, ma cosi non è. Infatti MSPASS può anche essere utilizzato come strumento per l'analisi e lo studio di logiche non-classiche che non sono state anticipate degli sviluppatori. Per questo utilizzo è una fortuna che MSPASS non produca una risposta

del tipo SI/NO: nel output sono disponibili l'insieme della clausole utilizzate sia se il problema in input risulti soddisfacibile, sia che risulti insoddisfacibile. Infatti un insieme finito di clausole saturate fornisce una descrizione della classe di modelli ricevuta in input.

Capitolo 2

Specifiche del linguaggio

Passiamo ora a dare un'analisi più precisa e formale del linguaggio di script con il quale andremo a specificare la logica che vogliamo analizzare tramite MSPASS [1].

Si premette che qui verranno solo definiti i concetti principali e la semantica dei vari comandi, mentre per una specifica completa della sintassi del linguaggio, cosí come compare nella specifica ufficiale disponibile in [2], si rimanda all'Appendice A.

Il linguaggio, con il quale possibile specificare una logica, un semplice linguaggio di scripting che utilizza i soli caratteri ASCI. Per questo sono state introdotte una serie di key-words che ci permettono di utilizzare sia i simboli della logica classica sia quello delle logiche modali.

Gli script sono sempre composti da una serie di unita informative che sono caratterizzate dall'avere un tipo, definito tramite la sintassi:

```
tipoUnita := begin_tipoUnita(eventuali parametri)
             ....
             end_tipoUnita.
```

ed al loro interno possono avere una serie annidata di altre unità informative.

Facciamo notare come tutte le assegnazioni termino sempre tramite un punto e che i commenti sono introdotti dal simbolo % il quale commenta l'intera riga che lo segue.

L'unità base che descrive ciò che vogliamo risolvere il problema (*problem*). Un problema non può contenere formule o clausole, ma solo informazioni e parametri per le varie unità in esso contenuti.

La definizione formale del problema :

```
problem ::= begin_problem(identifier).
            description
```

```
logical_part
{settings}*
end_problem.
```

Passeremo ora all'analisi più dettagliata delle singole parti. Introdurremo spesso nuove parole chiave che ovviamente devono avere una loro specifica sintassi. Per le definizioni che non vengono riportate in questo capitolo, in quanto abbastanza intuitive, si rimanda all'Appendice A dove presente la definizione del linguaggio in versione completa.

2.1 Il campo logical part

Questo il campo principale nel quale, in pratica, si scrive l'intera logica da analizzare. Qui andremo a scrivere le nostre variabili, i simboli e tutte le formule che costituiscono la nostra conoscenza. Anchequesto campo a sua volta suddiviso in una serie di parti che assolvono un preciso ruolo in termini funzionali: un qualsiasi simbolo non predefinito usato nel problema deve prima essere definito nella parte riservata alle dichiarazioni. In seguito in questa parte si può procedere con la formulazione del problema utilizzando formule e clausole in forma normale. Inoltre possono essere aggiunte formule speciali ed una lista di valori per le prove. La suddivisione la seguente:

```
logical_part  ::=   {symbol_list}
                    {declaration_list}
                    {formula_list}
                    {special_formula_list}
                    {clause_list}
                    {proof_list}*
```

Analizziamo in dettaglio le singoile parti.

2.1.1 La lista dei simboli

Come detto poc'anzi, un simbolo non predefinito deve essere dichiarato all'inizio del problema. Si intende con simbolo una qualsivoglia espressione letterale che non sia supportata dal linguaggio: in pratica tutte le funzioni e predicati che vorremo utilizzare vanno qui dichiarate.

La dichiarazione di nuovi simboli si può effettuare in questa sezione. Tutti i nuovi simboli introdotti devono essere differenti sia gli uni dagli altri e sia dai simboli predefiniti. E' supportato un ricco linguaggio per la definizione dei nuovi simboli: non sono ammesse variabili libere nella dichiarazione dei termini, ma ammesso l'ordinamento polimorfico.

Facciamo notare (come si vede nell'esempio visibile alla tabella 3.1 a pagina 13) che la definizioni delle variabili può avvenire come dichiarazione di una funzione a zero argomenti.

La sintassi per la definizione di nuovi simboli e di la seguente:

```
symbol_list ::= list_of_symbols.
         {functions[fun_sym | (fun_sim , arity)
            {, fun_sym | (fun_sim , arity)}* ].}
         {predicates[pred_sym | (pred_sim , arity)
            {, pred_sym | (pred_sim , arity)}* ].}
         {sorts[sort_sym | {, sort_sim }* ].}
         {operators[op_sym | (op_sim , arity)
            {, op_sym | (op_sim , arity)}* ].}
         {quantifiers[quant_sym | (quant_sim , arity)
            {, quant_sym | (quant_sim , arity)}* ].}
         {translpairs[ (pred_sym , pred_sym)
            { (pred_sym , pred_sym)}* ].}
         end_of_list
```

A volte per le logiche modali e le logiche descrittive è necessario creare una corrispondenza tra il simbolo proposizionale ed il corrispettivo simbolo della logica del prim'ordine. Questo può essere fatto con una dichiarazione di *translpairs*. Il primo simbolo in una coppia deve essere un simbolo di predicato con una arità nulla, mentre il secondo elemento della coppia è solitamente un predicato di cardinalità unaria o binaria.

2.1.2 La lista delle dichiarazioni

Tramite questo costrutto è possibile comunicare al reasoner la sematica nel nuovo simbolo introdotto. Qui si deve specificare il ruolo che il nuovo termine funzionale assume all'interno della notrsa logica.
Il linguaggio per definire questi ruoli è molto ricco. L'unica limitazione è che non possono essere definite variabili libere, ma in companso è ammesso l'ordinamento polimorfico.

Per le logiche più semplici si può omettere questa parte, ma quando se ne fanno di più complesse diventa necessaria. Nell'esempio 3.1 infatti non era presente per via dell'enorme semplicità. Per vedere invece un utilizzo di questo costrutto si guardi all'esempio 3.2.

```
declaration_list ::= list_of_declarations.
                     { declaration }*
                     end of list.
```

```
declaration ::= subsort_decl | term_decl | pred_decl | gen_decl
gen_decl    ::= sort sort_sym {freely} generated by func_list.
func list   ::= [ fun sym {,fun sym}*]
subsort_decl::= subsort(sort_sym , sort_sym).
term_decl   ::= forall ( term_list, term ). | term.
pred_decl   ::= predicate(pred_sym {, sort_sym}+ ).
sort_sym    ::= identifier
pred_sym    ::= identifier
fun_sym     ::= identifier
```

2.1.3 La lista delle formule

Ci sono due differenti tipi di formule: assiomi (axiom) e congiunture (conjucture). Se lo stato del problema, definito nel campo description (vedi paragrafo 2.2) è "unsatifsable" esso riferirà alla forma normale a clausole risultante dalla congiunzione di tutti gli assiomi con la negazione delle congiunzioni delle formule di congettura. Ovviamente lo stato di "satisfiable" significa che tutte le formule avranno un modello.

```
formula_list  ::= list_of_formulae(origin_type).
                  {formula({term}{, label}). }*
                  end_of_list.
origin_type   ::= axioms | conjectures
```

Si assume che tutte le formule siano chiuse: per questo non sono permesse variabili libere all'interno di un espressione.

I quantificatori hanno sempre due argomenti: una lista di termini e la sottoformula. La lista di termini si assume che sia una lista di variabili (o una lista di variabili contrassegnate da un ordinamento) per gli usuali quantificatori della logica classica. Si possono creare anche dei quantificatori "non classici" nel seguente modo:

```
term          ::= quant_sym(term_list , term) | symbol |
                  symbol(term {, term}*)
term_list     ::= [term {, term}*]
quant_ sym    ::= forall | exists | identifier
symbol        ::= equal | true | false | or | and | not
                  implies | implied | equiv | identifier
```

Le formule inserite possono essere sia nella forma disgiuntiva normale sia che nella forma a clausole. Ogni clausola deve essere scritta nella sua corrispondente formula ed in particolare ogni variabile deve essere limitata

dal proprio quantificatore. Sono disponibili diversi costrutti per la creazione di clausole nelle varie forme, per la sintassi di questi si rimanda all'Appendice A.

2.1.4 Le formule speciali

I problemi di logica modale sono specificati utilizzando un tipo speciale di formule le quali includono le formule del prim'ordine, le formule proposizionali (o Boolenae) e le formule relazionali. La definizione di questo tipo di formula è la seguente:

```
special_formula_list ::=
        list_of_special_formulae(origin_type , special_type).
        {labelled_formula}*
        end_of_list.
labelled_formula ::= formula({term}{, label}) |
        prop_formula_name({prop_term}{, label}) |
        rel_formula_name({rel_term}{, label})
prop_formula_name ::= prop_formula | concept_formula
rel_formula_name  ::= rel_formula | role_formula
special_type      ::= eml | dl
```

Gli operatori e quantificatori predefiniti sono:

- Gli operatori standard Booleani della logica proposizionale: true, false, not or, imples (è sussunto da), implied (sussume), equiv.
- Gli operatori della logica multi-modale: dia e box (con i sinonimi di some e all)
- Alcuni operatori relazionali addizionali: comp (composizione), sum (somma relativa), conv (converso), id (la relazione identità), div (la relazione diversità), test (per il test), domrestr (restrizioni sul dominio), ranrestr (restrizioni sul coodominio o range).

E questa è la loro sintassi:

```
prop_term       ::= prop_symbol |
                    prop_symbol(prop_term{, prop_term}*) |
                    prop_quant_sym(rel_term , prop_term) |
                    prop_quant_sym_unary(rel_term)
prop_symbol     ::= true | false | or | and | not |
                    implies | implied | equiv |
```

```
                    identifier
prop_quant_sym  ::= box | dia | all | some
prop_quant_sym_unary ::= domain | range
rel_term        ::= rel_symbol |
                    rel_symbol(rel_term {, rel_term}*) |
                    rel_prop_sym(rel_term , prop_term) |
                    rel_prop_sym_unary(prop_term)
rel_symbol      ::= true | false | id | div | or | and | not |
                    implies | implied | equiv | comp | sum |
                    conv | identifier
rel_prop_sym    ::= domrestr | ranrestr
rel_prop_sym_unary ::= test
```

2.1.5 Le prove

Infine è possibile definire un smplice formato per le prove da effettuare. Fondamentalmente una prova consiste in una sequenza di "semplici" passaggi nella fase di risoluzione.

Fin'ora abbiamo scritto degli esempi che utilizzavano automaticamente le prove di risoluzione (cioè utilizzavano quelle di default senza specificarne alcuna). Ora invece con questo campo si possono abilitare controlli complicati e lunghe prove automaticamente utilizzando un differente prover.

Per le definizioni delle possibili opzioni si rimanda all'Appendice A.

2.2 Il campo description

Questo campo serve per fornire informazioni aggiuntive circa l'autore. Inoltre è possibile inserire anche una breve descrizione del problema.

Poichè le informazioni in esso contenuto sono decisamente intuitive, per la definizione delle possibili opzioni si rimanda all'Appendice A.

2.3 Il campo settings

Questo campo è stato introdotto per permettere di inserire nella specifica del problema i valori per riprodurre specifiche prove che dipendono dal particolare tipo di reasoner che si vuole adottare.

Per le possibili opzioni si rimanda all'Appendice A.

Capitolo 3

Esempi ed analisi degli output

In questo capitolo ci addentreremo nel profondo dell'analisi di MSPASS [1] andando a vedere i risultati del suo funzionamento.

Per questo utilizzeremo dei semplici esempi che ci aiuteranno a capirne i complicati e lunghi messaggi di risposta ad un problema. Questi messaggi spesso non vengono riportati per intero in questo paragrafo (per ovvi motivi di leggibilità), ma sono comunque tutti disponibili in forma completa nell'Appendice B.

3.1 Esempio 1 - Una semplicissima logica

Passeremo ora ad illustrare un esempio molto semplice dell'utilizzo di quanto visto fin'ora. Proveremo ad eseguire l'analisi di una logica molto semplice studiando i risultati che si ottengono in due distinti casi:

a) In questo primo caso passeremo al reasoner una serie di assiomi e una formula da verificare: dovrà dunque eseguire una prova di correttezza.

b) Nel secondo caso gli passiamo i soli degli assiomi: il reasoner dovrà trovare tutte le formule che riusltano essere compatibili (cioè devono essere vere) con i dati forniti.

Fondamentalmente questi due casi sono le due grandi tipologie di problemi che possiamo risolvere con MSPASS [1].

La logica che si vuole analizzare è decisamnte modesta, ma l'esempio ci sarà utile per studiare il formato dei messaggi di output. Ricordiamo che questi messaggi non sono riportati per intero, ma solo nelle loro parti principali. Sono comunque disponibili tutti questi messaggi nell'appendice B.

Prima di procedere con l'analisi dei messaggi vediamo la logica che analizzeremo. La "conoscenza" che forniremo all'applicazione è costituita dalle seguenti formule:

$mioPredicato(miaFormula(a, b), miaFormula(b, c))$

$mioPredicato(miaFormula(b, c), miaFormula(a, c))$

$\forall x, y, z(mioPredicato(x, y) \wedge mioPredicato(y, z)) \Rightarrow mioPredicato(x, z)$

dove a,b e c sono delle costanti. La terza formula è l'unica proprietà che deve essere verificata, metre le prime due costituiscono una sorta di conscenza sul mondo. (Come vedremo più avanti nelle logiche DL questi prendono i nomi di A-Box e T-Box)

Ed infine ecco la formula da verificare:

$mioPredicato(miaFormula(a, b), miaFormula(a, c))$

Ovviamente così come sono scritte non sono input compatibili per il nostro reasoner. Le traduciamo con le regole viste nel capitolo 2.

3.1.1 Caso A - verifica

Il primo caso che analizzaremo è quello in cui si chiede a MSPASS [1] di verificare la correttezza di una formula rispetto ad un altro insieme di formule.

Il testo del problema, riportato nella sintassi richiesta, è visibile nella tabella 3.1. Come si vede la sintassi è piuttosto intuitiva. Segnaliamo solo che il fatto che caratterizza che stiamo eseguendo una verifica di una formula è dato dalla presenza di entrabe le voci:

```
list_of_formulae(axioms)
....
list_of_formulae(conjectures)
....
```

Ora possiamo eseguire il nostro semplice esempio e passare ad analizzare i risultati ottenuti.

Analisi output

L'output del programma è costituito da una pagina web in cui vengono riportate tutte le conclusione tratte ed i passi effettuati per raggiungerle.

Effettueremo ora un'analisi molto dettagliata di questi messaggi che nella struttura rimangono pressochè immutati indipendentemente dalle conlcusioni tratte. Dunque le considerazioni fatte sul segunete esempio saranno valide anche per tutti gli altri casi. La risposta completa a questo esempio è visibile nell'Appendice B.1 alla pagina 37.

Come prima cosa vengono visualizzate le informazioni del problema. A seguito della riga:

```
Input Problem:
```

si trovano le informazioni riassunte di quanto fornito. In primis vengono riassunte le formule della logica da analizzare secondo il seguente schema:

- le formule di sole costanti in input sono evidenziate da una → posta in testa

s

```
begin_problem(SempliceEsempio).
  list_of_descriptions.
    name({* Semplice Esempio *}).
    author({* Massimiliano Carrara *}).
    status(unsatisfiable).
    description({* A description logic example. *}).
  end_of_list.

  list_of_symbols.
    functions[(miaFunzione,2), (a,0), (b,0), (c,0)].
    predicates[(mioPredicato,2)].
  end_of_list.

  list_of_formulae(axioms).
    formula(mioPredicato(miaFunzione(a,b),miaFunzione(b,c))).
    formula(mioPredicato(miaFunzione(b,c),miaFunzione(a,c))).
    formula(forall([U,V,W],implies(and(mioPredicato(U,V),
                    mioPredicato(V,W)),mioPredicato(U,W)))).
  end_of_list.

  list_of_formulae(conjectures).
    formula(mioPredicato(miaFunzione(a,b),miaFunzione(a,c))).
  end_of_list.
end_problem.
```

Tabella 3.1: Esempio 1.A - verifica

- le formule di sole costanti da verificare sono evidenziate da una → posta in coda
- le formule di variabili (libere o vincolate) sono riscritte cos come sono

Seguono alune informazioni circa le opzioni del reasoner, le precedenze adottate ed altri parametri come qui riporato:

```
Input Problem:
1[0:Inp]  ||  -> mioPredicato(miaFunzione(b,c),miaFunzione(a,c))*.
2[0:Inp]  ||  -> mioPredicato(miaFunzione(a,b),miaFunzione(b,c))*.
3[0:Inp]  || mioPredicato(miaFunzione(a,b),miaFunzione(a,c))* -> .
4[0:Inp]  || mioPredicato(U,V)* mioPredicato(W,U)* ->
                                                  mioPredicato(W,V)*.
 This is a first-order Horn problem without equality.
 The conjecture is ground.
 Axiom clauses: 3 Conjecture clauses: 1
 Inferences: IORe
 Reductions: RFClR RBClR RObv RUnC RTaut RFSub RBSub RCon
 Extras    : Input Saturation, Dynamic Selection, No Splitting,
         Full Reduction,  Ratio: 5, FuncWeight: 1, VarWeight: 1
 Precedence: miaFunzione > div > id > mioPredicato > c > b > a
 Ordering  : KBO
 .......
```

Dopo di questi vengono visualizzate una lunga serie di formule che costituiscono tutte le clausole che il programma è stato in grado di generare con gli input forniti.

A seguire queste si trova l'istruzione:

```
 SPASS beiseite: Proof found.
```

la quale indica ovviamente che la prova è stata effettivamente trovata. Seguono una serie di statistiche sui tempi e sulle risore di calcolo utilizzate per 'operazione appena compiuta.

Infine vengono riportati i soli passaggi utilizzati per la dimostrazione della formula richiesta (metre prima venivano riportate TUTTE le clausole generabili) dove il prefisso $[0 : Imp : ..]$ segnala che la formula seguente era tra quelle fornite dal problema, metre il prefisso $[0 : Res : ..]$ segnala che si tratta di una formula ricavata dal reasoner.
Per maggiore chiarezza riportiamo anche questa parte di codice:

```
Here is a proof with depth 2, length 7 :
1[0:Inp]  ||  -> mioPredicato(miaFunzione(b,c),miaFunzione(a,c))*.
2[0:Inp]  ||  -> mioPredicato(miaFunzione(a,b),miaFunzione(b,c))*.
3[0:Inp]  || mioPredicato(miaFunzione(a,b),miaFunzione(a,c))* -> .
4[0:Inp]  || mioPredicato(U,V)*+ mioPredicato(W,U)* ->
                                                  mioPredicato(W,V)*.
7[0:Res:1.0,4.0]  || mioPredicato(U,miaFunzione(b,c))* ->
                                 mioPredicato(U,miaFunzione(a,c)).
9[0:Res:2.0,7.0]  ||  -> mioPredicato(miaFunzione(a,b),
                                                  miaFunzione(a,c))*.
11[0:ClR:9.0,3.0]  ||  -> .
Formulae used in the proof : axiom1 axiom0 conjecture0 axiom2
```

3.1.2 Caso B - ricerca

In questo secondo caso chiederemo a MSPASS[1] di ricercare tutte le formule compatibili con quelle date. Questo poichè definiamo solo una serie di formule con l'etichetta "list_of_formulae(axiom)" e nessuna in "list_of formulae(conjectures)" (ma discorso analogo sarebbe se definissimo solo congiunture e nessun assioma..).

Notiamo dalla lettura del testo del problema, visibile nella tabella 3.2, che in pratica i problemi sono gli stessi, solo che in questo caso manca l'ultima serie di formule.

Analisi output

Le considerazioni fatte nel caso precedente sono ancora valide. Vengono riportare alcune considerazioni sui dati forniti e l'elenco di tutte le clasuole generate. Anche per questo caso, la risposta completa a questo esempio è visibile nell'Appendice B.2 alla pagina 39.

In seguito, se si è trovata una conclusione al processo di generazione delle clausole, è riportata la segunete dicitura:

```
SPASS beiseite: Completion found.
```

e le solite statistiche.

Infine viene visualizzata l'elenco di tutte e clausole trovate: la sintassi dei prefissi è quella vista per il caso precedente. Anche in questo caso riportiamo parte di questo messaggio a titolo di esempio:

```
The saturated set of worked-off clauses is :
10[0:Res:8.1,3.0]  || mioPredicato(U,miaFunzione(a,b))* +
```

```
        mioPredicato(V,U)* -> mioPredicato(V,miaFunzione(a,c))*.
6[0:Res:4.1,3.0]  || mioPredicato(U,miaFunzione(a,b))*+
         mioPredicato(V,U)* -> mioPredicato(V,miaFunzione(b,c))*.
8[0:Res:4.1,5.0]  || mioPredicato(U,miaFunzione(a,b)) ->
                              mioPredicato(U,miaFunzione(a,c))*.
7[0:Res:2.0,5.0]  ||  -> mioPredicato(miaFunzione(a,b),
                                      miaFunzione(a,c))*.
5[0:Res:1.0,3.0]  || mioPredicato(U,miaFunzione(b,c))* ->
                               mioPredicato(U,miaFunzione(a,c)).
4[0:Res:2.0,3.0]  || mioPredicato(U,miaFunzione(a,b)) ->
                              mioPredicato(U,miaFunzione(b,c))*.
3[0:Inp]  || mioPredicato(U,V)*+ mioPredicato(W,U)* ->
                                      mioPredicato(W,V)*.
1[0:Inp]  ||  -> mioPredicato(miaFunzione(b,c),miaFunzione(a,c))*.
2[0:Inp]  ||  -> mioPredicato(miaFunzione(a,b),miaFunzione(b,c))*.
```

Infine facciamo notare come tra le formula trovate ci sia anche quella che nel

```
begin_problem(SempliceEsempio).
  list_of_descriptions.
    name({* Semplice Esempio *}).
    author({* Massimiliano Carrara *}).
    status(unsatisfiable).
    description({* A description logic example. *}).
  end_of_list.
  list_of_symbols.
    functions[(miaFunzione,2), (a,0), (b,0), (c,0)].
    predicates[(mioPredicato,2)].
  end_of_list.
  list_of_formulae(axioms).
    formula(mioPredicato(miaFunzione(a,b),miaFunzione(b,c))).
    formula(mioPredicato(misaFunzione(b,c),miaFunzione(a,c))).
    formula(forall([U,V,W],implies(and(mioPredicato(U,V),
                    mioPredicato(V,W)),mioPredicato(U,W)))).
  end_of_list.
end_problem.
```

Tabella 3.2: Esempio 1.B - ricerca

caso precedente avevamo utilizzato come prova.

3.1.3 Alcune note finali

Oltre alle prove qui illustrate sono state eseguite altre prove, sempre con la medesima logica da analizzare, ma tese a valutare il comportamento in termini di tempi e di efficenza del reasoner.

In primo luogo abbiamo provveduto ad eseguire alcune prove per verificare l'utilità della voce "status(satisfiable)" del campo "list_of_description". In seguito alle prove effettuate si è verificato che in pratica non cambia niente. Nell' esempio effettuto, che chiameremo Esempio 1.C (non riportiamo il codice in quanto identico all'Esempio 1.B salvo la voce discussa poc'anzi) si vede che anche in termini prestazionali non vi è la minima differenza.

Siamo poi passati ad analizzare un caso un pò più interessante: cosa succede se si aggiungono delle clausole "inutili" al problema? Di quanto di complica? Per provare abbiamo eseguito una variante dell'Esempio 1.B con la seguente aggiunta:

```
list_of_formulae(axioms).
  formula(mioPredicato(miaFunzione(a,b),miaFunzione(b,c))).
  formula(mioPredicato(miaFunzione(b,c),miaFunzione(a,c))).
  formula(forall([U,V,W],implies(and(mioPredicato(U,V),
                  mioPredicato(V,W)),mioPredicato(U,W)))).
  % formule inutili,
  formula(pippo(miaFunzione(a,b))).
  formula(pippo(pluto(a,b))).
  formula(pippo(pluto(a,c))).
end_of_list.
```

In cui si vede chiaramente che il predicato "pippo" e la fonzione "pluto" non servono a niente salvo a complicare il processo di ragionamento. Dall'analisi dell'output, coma ci si aspettaava, è risultato un tempo leggermente maggiore per la definizione del problema (in quanto andavano processate più clausole, ma le conclusioni si sono rilevate corrette e non utilizzanti le fomrule "in più").

Infine abbiamo voluto provare il comportamento del reasoner nel caso si trovi di fronte ad una prova irrisolvibile. Per questo abbiamo eseguito il nostro solito programma con la seguente modifica:

```
list_of_formulae(conjectures).
  formula(pippo(pluto(b,c))).
end_of_list.
```

In questo caso, non potendo trovare la prova richiesta, il reasoner si limita a fornire l'elenco di tutte le clausole trovate come nel caso dell'Esempio 1.B.

3.2 Esempio 2 - una logica modale

Ora illustreremo un semplice esempio che utilizza una logica modale. Come detto precedentemente MPASS è nato fondamentalemnte per la gestione delle logiche modali: ci aspetteremo dunque che in questo campo ottenga le migliori performance.

La logica analizzata è decisampente semplice, si tratta solo di tre assimi:
L'assimoma K:
$\Box(A \Rightarrow B) \Rightarrow (\Box A \Rightarrow \Box B)$
La proprietà transitiva:
$\Box A \Rightarrow \Box\Box A$
La proprietà riflessiva:
$\Box A \Rightarrow A$

Abbiamo utilizzato solo questi poichè un'inutile complessità non avrebbe comunque giovato alle considerazioni che si possono fare sugli output.

3.2.1 Definizione del problema

La traduzione nel linguaggio di MPASS delle pochisime regole viste prima si può facilmente ottenere ed è presentata nella Tabella 3.3.

Come si può facilmente vedere dal testo la traduzione delle fomrmule modali è piuttosto banale. L'unica nota doverosa riguarda l'uso dell'operatore "box"[1]: infatti, come si nota dal testo del problema, "box" ha arità binaria. Questo ci scandalizza un poco in quanto sappiamo che invece nella logica modale box è un operatore unario. Tuttavia questa cardinalità binaria di box è solo appparente. Infatti il primo termine indica solo la relazione sulla quale deve essere verificata.

Notiamo infine come siano presenti delle coppie di *"translpairs"*. Queste ci servono in quanto dobbiamo creare una corrispondenza tra gli assiomi che enunciano le propietà volute e le formule con le variabili (o le costanti) con le quali si testa la logica. Si ricorda che la formulazione utilizzata è solo una delle possibili e si sarebbe anche potuto utilizzare la sola etichetta *"formula"* anche per la definizione delle propietà. In quanto caso si poteva fare a meno dei predicati A e B e dunque anche delle coppie definite in *"translpairs"*. Tuttavia quest'ultima formulazione, pur essendo identica nella semantica, ha una scrittura che appare meno chiara rispetto a quella fornita.

[1] Identico dispcorso anche per l'uso di "dia"

```
begin_problem(semplice_prova_logica_modale).
list_of_descriptions.
  name({* Semplcie prova di una logica modale*}).
  author({* Massimiliano Carrara *}).
  status(satisfiable).
  description({* Semplice logica modale*}).
end_of_list.
list_of_symbols.
functions [ (x,0),  (y,0),  (z,0) ].
predicates[ (r,0),(A,0),(B,0), (a,2),(b,2) ].
translpairs[ (a,A),(b,B) ].
end_of_list.

list_of_special_formulae(conjectures, eml).
prop_formula(            %schema K
   implies(
      box(r, implies(A , B)),
      implies(box(r, A) , box(r, B))
   )
).
prop_formula(            %propiet\‘{a} transitiva
   implies( box(r, A), box(r, box(r, A) ))
).
prop_formula(            %propiet\‘{a} riflessiva
   implies( box(r, A), A)
).
 formula( a(x,y) ).
 formula( a(y,z) ).
end_of_list.

list_of_special_formulae(axioms, EML).
 formula( a(x,z) ).
end_of_list.
end_problem.
```

Tabella 3.3: Esempio 2 - Codice

3.2.2 Analisi output

Passiamo ora ad analizzare i risultati forniti da quest'esempio. Come nel caso precedente non verranno riportati tutti i messaggi, ma solo el parti più significative. Per una Visione completa dei messaggi si rimanda all'Appendice B.3

A differenza del caso precedente notiamo che c'è una novità. Infatti come prima cosa ci vine presentata una sezione che identifica il tipo di traduzione effettuata.

```
----------------------TRANSLATION-START-------------------------
 Translation method: relational
 Translation flags: EML EMLAuto EMLFuncNdeQ
 Precedence:  equal > true > false > div > id > x > y > z > r >
                                                  A > B > a > b
Axioms:
Label : axiom0
 [FOL] (a (x) (z))
 [Simpl] (a (x) (z))
 [FOL] (a (x) (z))
Conjecture:
 [EML] (not (implies (box (r) (implies (A) (B)))
                            (implies (box (r) (A)) (box (r) (B)))))
 [Simpl] (not (implies (box (r) (implies (A) (B)))
                            (implies (box (r) (A)) (box (r) (B)))))
 [RelTr]  (not (forall ( U) (implies (forall ( V) (implies (R_r U V)
               (implies (P_A V) (P_B V)))) (implies (forall ( W)
               (implies (R_r U W) (P_A W))) (forall ( X)
               (implies (R_r U X) (P_B X)))))))
 ...........
```

Come discusso nelle pagine precedenti la traduzione delle formule fornite in clausole FOL ha un ruolo decisivo nel processo di reasoning. Poichè non sono state specificate delle opzioni particolari la traduzipone avviene con il meccanismo di default (che ricordiamo è il *"metodo di traduzione relazionale"*).

Successivamente si passa all'elenco delle formule fornite secondo le modalità discusse nell'esempio precedente.
Si arriva cosi alla parte finale:

```
SPASS beiseite: Proof found.
Problem: Interactive Web Input, Sun Jan  9 16:11:37 2005
```

```
SPASS derived 3 clauses, backtracked 0 clauses and kept 16 clauses.
SPASS allocated 1269 KBytes.
SPASS spent 0:00:00.04 on the problem.
0:00:00.01 for the input.
0:00:00.01 for the FLOTTER CNF translation, of which
0:00:00.00 for the translation from EML to FOL.
0:00:00.00 for inferences.
0:00:00.00 for the backtracking.
0:00:00.00 for the reduction.


Here is a proof with depth 1, length 7 :
6[0:Inp]  ||  -> R_r(skc6,skc7)*.
7[0:Inp]  || P_B(skc7)*+ -> .
12[0:Inp]  || R_r(skc6,U)*+ -> P_A(U).
13[0:Inp] P_A(U) || R_r(skc6,U)* -> P_B(U).
14[0:ClR:13.0,12.1]  || R_r(skc6,U)*+ -> P_B(U).
21[0:Res:6.0,14.0]  ||  -> P_B(skc7)*.
22[0:ClR:21.0,7.0]  ||  -> .
Formulae used in the proof : conjecture0
```

Come si vede la prova voluta è stata trovata. Qusto poichè avevamo utilizzato sia le etichette "axiom" che "conjecture".

3.3 Esempio 3 - una logica DL

Eccoci infine all'analisi di una logica DL. Certo abbiamo visto, al praragrafo precedente, che MSPASS [1] ottiene ottimi risultati con le logiche modali. Dunque sembrerebbe inutile testarlo anche con le DL, visto che dopotutto si tratta solo di un tipo "speciale" di questa categoria di logiche[2].

Tuttavia si è ritenuto utile eseguire un test anche con questo tipo di logiche in quanto costituiscono una famiglia ampliamente utilizzata in ambito applicativo (si pensi alle ontologie ed al sematic web) e supportata da numerosi tool per il reasoning (che tuttavia utilizzano quasi sempre il metodo dei tableuax).

3.3.1 Una breve introduzione alle logiche descrittive

Utilizzando FOL si possono esprimere conoscenze molto articolate e, almeno in linea di principio, eseguire in modo automatico ragionamenti complessi. Ci sono però due problemi.

Il primo è che in FOL la procedura di deduzione (detta anche *procedura di prova o calcolo*) non è una procedura di decisione, ma soltanto di semidecisione. Ciò significa che:

- quando la conclusione è deducibile delle premesse, la procedura termina in un numero finito di passi producendo una prova;
- quando invece la conclusione non è deducibile delle premesse, la procedura può non terminare.

Ciò significa che un sistema informatico può "andare in ciclo" se tenta di dedurre da un insieme di premesse una conclusione che in effetti non è deducibile delle premesse.

Il secondo problema è che la procedura, anche nei casi in cui termina, è spesso molto costosa in termini di risorse di calcolo.
Entrambi i problemi dipendono, almeno in parte, dalla notevole espressività di FOL: come è naturale, più un linguaggio di rappresentazione è espressivo, più sono complesse le procedure di ragionamento basate su di esso. Molte ricerche nel campo dei linguaggi di *Knowledge Rappresentation* (KR) hanno l'obiettivo di identificare un sottolinguaggio di FOL tale che:

- il linguaggio sia comunque abbastanza espressivo per le applicazioni;

[2]Infatti si può vedere la *logica ALC* come una variante sintattica di $K_{(n)}$: per esempio le relazioni verrebero tradotte in diversi operatori box e dia, uno per ciascuna relazione appartenente alla logica DL

- la deduzione si basi su una procedura di decisione (che quindi termina in ogni caso dopo un numero finito di passi, sia quando la conclusione è deducibile dalle premesse, sia quando non lo è);
- la procedura di deduzione abbia complessità computazionale accettabile (ovvero richieda una quantità accettabile di risorse di calcolo).

I sistemi di questo tipo hanno preso il nome di logiche descrittive (DL, *Description Logic*). Le DL utilizzano una sintassi[3] semplificata rispetto a FOL.

Infine ricordiamo che fondamentalmente un'ontologia è formata da due parti principali: la **T-Box** e l' **A-Box**. Nella prima vi stanno le definizioni terminologiche ed i ruoli, cioè le conoscenze generali, mentre nella seconda si trovano le conoscenze fattuali ovvero le "istanze" dei termini.

3.3.2 Una semplice ontologia

La logica che si è voluto testare è visibile nella Tabella 3.4. Si è deciso di riportare per intero la struttura di questa semplice DL anche in uno pseudo-linguaggio molto simile al quello della logica classica affinchè si possano notare le conversioni effettuate nel linguaggio del reasoner.
La logica in questione è una semplicissima ontologia di una festa di compleanno al quale sono presenti delle persone vegetariane. Vogliamo fornire al reasoner la conoscenza sufficente affinchè trovi tutte le relazioni possibili, ovvero vogliamo conoscere se ci sono dei vegetariani presenti alla festa.

Tralasciamo tutti i commenti sulla sintassi per la stesura della logica: sono stati ampliamente discussi nell'Esempio 3.1(pagina 11) e sono sostanzialmente invariati.

Vorremmo solo far notare l'utilizzo della etichetta di "translpairs". Infatti, come accennato nel Capitolo 2, a volte per le logiche modali e le logiche descrittive è necessario creare una corrispondenza tra il simbolo proposizionale ed il corrispettivo simbolo della logica del primordine. Questo può essere fatto con una dichiarazione di translpairs. Il primo simbolo in una coppia deve essere un simbolo di predicato con una arità nulla, mentre il secondo elemento della coppia solitamente un predicato di cardinalità unaria o binaria.

Ricapitolando, ecco cosa abbiamo dichiarato nella parte inziale dedicata alle dichiarazione dei simboli dell'utente:

1. Si definiscono come formule di arità zero tutti i termini che vogliamo che costituiscano la nostra conoscenza (come definivamo negli esempi

[3]In questa breve relazione utilizziamo per i simboli il font standard Symbol. Nella letteratura delle logiche descrittive vengono utilizzati simboli analoghi ma "a spigoli".

T-Box
$PERSONA \equiv UOMO \cup DONNA$
$CIBO \equiv VEGETALE \cup ANIMALE$
$VEGETARIANO \equiv PERSONA \cap \neg Mangia(ANIMALE)$
Ruolo "Mangia":
$\bot \equiv \forall Mangia^{-}.PERSONA$
$\bot \equiv \forall Mangia.CIBO$
A-Box
$DONNA(anna)$
$UOMO(bob)$

$Mangia(anna, maiale)$
$Mangia(bob, insalata)$
$\neg Mangia(bob, maiale)$
$\neg Mangia(bob, mucca)$

$ANIMALI(maiale)$
$ANIMALI(mucca)$
$ANIMALI(pollo)$
$VEGETALI(insalata)$
$VEGETALI(pomodoro)$

Tabella 3.4: Esempio 3 - DL

precedenti le costanti). In altre parole le istanze dei termini che definiamo nella T-Box.
Inoltre, come nei casi precedenti, definimo sempre sotto questa etichetta i simboli funzionali (che in questo caso sono detti ruoli) e la loro arità.

2. Definiamo due tipi di predicati. Il primo ad arità zero (contrassegnato dal nome scritto in caratteri maiuscoli) che utilizzo nelle definizioni terminologiche "classiche", ovvero quelle nelle quali non interviene alcun ruolo, e che pertanto vanno definite dalle formule contrassegnate da *concept_formula*. Il secondo tipo di predicati (contrassegnato del nome tutto in minuscolo)avrà invece arità unitaria e utilizzerò quest'ultimo nelle definizioni nelle quali si debbono utilizzare i quantificatori (e perciò definite dalla laber *formula*).

3. Ovviamente si dovrà avere una corrispondenza tra queste die diverse is-

tanze degli stessi predicati. Per ottenere questo utilizziamo le definizioni di *transpairs*.

Successivamente nella sezione dedicata alle formule abbiamo introdotto la nostra ontologia. Le definizioni terminologiche che non richiedevano l'utilizzo dei ruoli sono state tradotte utilizzando l'etichetta *concept_formula* (definita solo per le logiche modali): in pratica si possono praticamente riscivere le formule cosi come definite nella logica di partenza. Invece per le formule che necessitavano dei ruoli abbiamo dovuto utilizzate la sintassi per la definizione delle formule FOL classiche ed i predicati ad arità unitaria.

La definizione completa del testo del problema tradotto nel linguaggio di MSPASS è visibile nella tabella 3.5.

L'esempio appena illustrato non illustra nulla di nuovo, l'unica nota meritevole riguarda le due righe che compaiono commentate e che definiscono dominio e codominio della relazione Mangia. Le specifiche del linguaggio permettono una siffatta definizione, ma effettivamente non si possono utilizzare. Infatti, dopo numerose prove, si è scoperto che se si utilizzano le definizioni in questione l'esecutore si blocca. Questo è probabilmente dovuto a qualche errore nel reasoner visto che l'errore si presenta anche con logiche semplissime.

3.3.3 Analisi output

Le considerazioni sugli output forniti dal programma non differiscono molto da quelle dei casi precedenti. Infatti, una volta che il testo non presenti errori, il processo di reasoning può essere avviato e nel giro di pochi secondi fornisce le sue considerazioni.

Come prima cosa presenta la traduzione effettuata delle formule in input[4]. Successivamente elenca le clausole fornite e procede con l'analisi ed il processo di reasoning ed infine fornisce l'elenco di tutte la clasuole trovate.

Proprio quest'ultima considerazione è interessante. Infatti sono state eseguite due prove della DL illustrata precedentemente:

- Una prima formata dai soli assiomi: ci aspetteremo dunque che il reasoner illustri tutte le formule valide che può ricavare.
- Una seconda esecuzione che aveva lo scopo di verificare la validità di una formula fornita: poichè quest'ultima era derivabile dall'insieme delle conoscenze fornite il reasoner è riuscito giustamente a trovarla.

[4]Anche in questo caso sono stati utilizzati i metodi di default come nell'Esempio 2

Analizziamo dunque più in dettaglio queste due esempi. Ricordiamo che verranno riportati, e di conseguna commentati, solo alcuni piccoli spezzoni dell'output fornito. Per una visuone completa del codice si rimanda all'appendice B.

Esempio 3.A - Ricerca

In questo caso abbiamo fornito solo una lista di assiomi, dunque il reasoner cercherà tutte le formule compatibili con le informazioni fornite. Questo caso è utile se si vogliono trovare tutte le conoscenze derivabili dalle informazioni fornite.

La parte più interessante del messaggio di output è la seguente:

```
SPASS V 1.0.0t
SPASS beiseite: Completion found.
Problem: Interactive Web Input, Sun Jan  9 11:25:04 2005
SPASS derived 3 clauses, backtracked 0 clauses and kept 21 clauses.
SPASS allocated 1277 KBytes.
SPASS spent 0:00:00.05 on the problem.
0:00:00.01 for the input.
0:00:00.01 for the FLOTTER CNF translation, of which
0:00:00.00 for the translation from EML to FOL.
0:00:00.00 for inferences.
0:00:00.00 for the backtracking.
0:00:00.00 for the reduction.

The saturated set of worked-off clauses is :
39[0:SSi:36.0,4.0]  ||  -> vegetariano(bob)*.
20[0:Inp] animale(U) ||  -> vegetariano(V) Mangia(V,U)*.
19[0:Inp] P_PERSONA(U) ||  -> P_DONNA(U)* P_UOMO(U).
14[0:Inp] P_VEGETARIANO(U) ||  -> P_PERSONA(U)*.
15[0:Inp] P_DONNA(U) ||  -> P_PERSONA(U)*.
16[0:Inp] P_UOMO(U) ||  -> P_PERSONA(U)*.
17[0:Inp] P_VEGETALE(U) ||  -> P_CIBO(U)*.
18[0:Inp] P_ANIMALE(U) ||  -> P_CIBO(U)*.
11[0:Inp]  || Mangia(bob,pollo)*+ -> .
12[0:Inp]  || Mangia(bob,mucca)*+ -> .
13[0:Inp]  || Mangia(bob,maiale)*+ -> .
8[0:Inp]  ||  -> Mangia(bob,pomodoro)*.
9[0:Inp]  ||  -> Mangia(bob,insalata)*.
10[0:Inp]  ||  -> Mangia(anna,maiale)*.
```

```
1[0:Inp]  ||  -> vegetale(pomodoro)*.
2[0:Inp]  ||  -> vegetale(insalata)*.
3[0:Inp]  ||  -> animale(pollo)*.
4[0:Inp]  ||  -> animale(maiale)*.
5[0:Inp]  ||  -> animale(mucca)*.
6[0:Inp]  ||  -> uomo(bob)*.
7[0:Inp]  ||  -> donna(anna)*.
```

Dal messaggio si capisce chiaramente che non si trattava di una verifica (dalla riga *"SPASS beiseite: Completion found."*) e che le conoscenze derivate dalle informazioni di input sono che bob è un vegetariano.
Questa è effettivamente l'unica conoscenza derivata dal ragionamento e lo si capisce dal fatto che non presenta il prefisso *[0:Inp]*, tipico delle formule fonrite in input, ma invece presenta il prefisso *[0:SSi:..]*.

Esempio 3.B - Verifica

In questo caso, come visto nell'Esempio 1.B, il reasoner fornisce una conferma del successo della prova (*SPASS beiseite: Proof found.*) e una dimostrazione schematica della prova effettuata.

```
SPASS V 1.0.0t
SPASS beiseite: Proof found.
Problem: Interactive Web Input, Sun Jan  9 11:27:37 2005
SPASS derived 4 clauses, backtracked 0 clauses and kept 25 clauses.
SPASS allocated 1277 KBytes.
SPASS spent 0:00:00.05 on the problem.
0:00:00.01 for the input.
0:00:00.01 for the FLOTTER CNF translation, of which
0:00:00.00 for the translation from EML to FOL.
0:00:00.00 for inferences.
0:00:00.00 for the backtracking.
0:00:00.01 for the reduction.

Here is a proof with depth 2, length 7 :
4[0:Inp]  ||  -> animale(maiale)*.
11[0:Inp]  || vegetariano(bob)*+ -> .
14[0:Inp]  || Mangia(bob,maiale)*+ -> .
21[0:Inp] animale(U) ||  -> vegetariano(V) Mangia(V,U)*.
22[0:Res:21.1,11.0] animale(U) ||  -> Mangia(bob,U)*.
38[0:Res:22.1,14.0] animale(maiale) ||  -> .
41[0:SSi:38.0,4.0]  ||  -> .
```

```
Formulae used in the proof : axiom9 conjecture0 axiom16 axiom1
```

```
begin_problem(esempio_festa).
list_of_descriptions.
  name({* Esempio DL - Vegetariani*}).
  author({* Massimiliano Carrara *}).
  status(unsatisfiable).
  description({* Esempio di logica descrittiva. *}).
end_of_list.
list_of_symbols.
functions[
    (anna,0),(bob,0),
    (maiale,0),(mucca,0),(pollo,0),
    (pomodoro,0),(insalata,0)
].
predicates[
    (PERSONA, 0),(UOMO, 0),(DONNA, 0),
    (CIBO, 0),(VEGETALE, 0),  (ANIMALE, 0),(VEGETARIANO, 0),
    (persona, 1),(uomo, 1),(donna, 1),
    (cibo, 1),(vegetale, 1),  (animale, 1),(vegetariano, 1),
    (Mangia, 2)
].
translpairs[
(persona,PERSONA),(uomo, UOMO),  (donna, DONNA),
(cibo, CIBO),(vegetale, VEGETALE),
(animale, ANIMALE),(vegetariano, VEGETARIANO)
].
end_of_list.
list_of_special_formulae(axioms, DL).
%T-Box
concept_formula(
    implies( or( ANIMALE, VEGETALE ), CIBO ) ).
%concept_formula( domain(PERSONA), Mangia).
%concept_formula( range(CIBO),  Mangia).
formula(
    forall([x,y], implies( and( animale(y) ,
    not(Mangia(x,y))) , vegetariano(x)))
).
concept_formula( implies( PERSONA, or(UOMO,DONNA))).

concept_formula( implies( UOMO,  PERSONA) ).

concept_formula( implies( DONNA, PERSONA) ).

concept_formula( implies( VEGETARIANO , PERSONA ) ).
```

```
% ABox
formula( donna( anna ) ).
formula( uomo( bob ) ).
formula( animale( mucca ) ).
formula( animale( maiale ) ).
formula( animale( pollo ) ).
formula( vegetale( insalata ) ).
formula( vegetale( pomodoro ) ).
formula( Mangia(anna, maiale ) ).
formula( Mangia(bob, insalata ) ).
formula( Mangia(bob, pomodoro ) ).
formula( not (Mangia(bob, maiale ) )).
formula( not (Mangia(bob, mucca ) )).
formula( not (Mangia(bob, pollo ) )).
end_of_list.
end_problem.
```

Tabella 3.5: Esempio 3 - Codice

Appendice A

Definizione sintassi

Presentiamo qui di seguito la definizione completa del linguaggio con cui è possibile specificare la logica da testare.

Cominciamo col definire alcuni simboli che verranno utilizzati per le successive definizioni.

A.1 Definizione alfabeto e simboli

Come prima cosa definiamo il significato di:

```
{ } = opazionale
{ }* = arbitraio
{ }+ = almeno uno
```

E dei simboli base dell'alfabeto:

```
identifier      ::= { letter | digit | special symbol }+
letter          ::= a-z | A-Z
arity           ::= -1 | number
number          ::= { digit }+
digit           ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
special_symbol  ::= _
```

A.2 Definizione problema

```
problem ::= begin_problem(identifier).
            description
            logical_part
            {settings}*
            end_problem.
```

A.2.1 Definizione campo "logical_part"

```
logical_part  ::=   {symbol_list}
                    {declaration_list}
                    {formula_list}
                    {special_formula_list}
                    {clause_list}
                    {proof_list}*
```

Simboli funzionali

```
symbol_list ::= list_of_symbols.
         {functions[fun_sym | (fun_sim , arity)
            {, fun_sym | (fun_sim , arity)}* ].}
         {predicates[pred_sym | (pred_sim , arity)
            {, pred_sym | (pred_sim , arity)}* ].}
         {sorts[sort_sym | {, sort_sim }* ].}
         {operators[op_sym | (op_sim , arity)
            {, op_sym | (op_sim , arity)}* ].}
         {quantifiers[quant_sym | (quant_sim , arity)
            {, quant_sym | (quant_sim , arity)}* ].}
         {translpairs[ (pred_sym , pred_sym)
            { (pred_sym , pred_sym)}* ].}
         end_of_list
```

Dichiarazioni

```
declaration_list ::= list_of_declarations.
                     { declaration }*
                     end of list.
declaration ::= subsort_decl | term_decl | pred_decl | gen_decl
gen_decl    ::= sort sort_sym {freely} generated by func_list.
func list   ::= [ fun sym {,fun sym}*]
subsort_decl::= subsort(sort_sym , sort_sym).
term_decl   ::= forall ( term_list, term ). | term.
pred_decl   ::= predicate(pred_sym {, sort_sym}+ ).
sort_sym    ::= identifier
pred_sym    ::= identifier
fun_sym     ::= identifier
```

Formule

```
formula_list  ::= list_of_formulae(origin_type).
                  {formula({term}{, label}). }*
                  end_of_list.
origin_type   ::= axioms | conjectures
label         ::= identifier
term          ::= quant_sym(term_list , term) | symbol |
                  symbol(term {, term}*)
term_list     ::= [term {, term}*]
quant_ sym    ::= forall | exists | identifier
symbol        ::= equal | true | false | or | and | not
                  implies | implied | equiv | identifier

clause list   ::= list_of_clauses(origin_type , clause_type).
                  {clause({cnf_clause | dnf clause}{, label}).}*
                  end_of_list.
clause_type   ::= cnf | dnf
cnf_clause    ::= forall(term_list , cnf_clause_body) |
                  cnf_clause_body
dnf_clause    ::= exists(term_list , dnf_clause_body) |
                  dnf_clause_body
cnf_clause_body ::= or(term {, term}*)
dnf_clause_body ::= and(term {, term}*)
```

Formule speciali

```
special_formula_list ::=
        list_of_special_formulae(origin_type , special_type).
        {labelled_formula}*
        end_of_list.
labelled_formula ::= formula({term}{, label}) |
        prop_formula_name({prop_term}{, label}) |
        rel_formula_name({rel_term}{, label})
prop_formula_name ::= prop_formula | concept_formula
rel_formula_name  ::= rel_formula | role_formula
special_type      ::= eml | dl

prop_term       ::= prop_symbol |
                    prop_symbol(prop_term{, prop_term}*) |
                    prop_quant_sym(rel_term , prop_term) |
```

```
                    prop_quant_sym_unary(rel_term)
prop_symbol     ::= true | false | or | and | not |
                    implies | implied | equiv |
                    identifier
prop_quant_sym  ::= box | dia | all | some
prop_quant_sym_unary ::= domain | range
rel_term        ::= rel_symbol |
                    rel_symbol(rel_term {, rel_term}*) |
                    rel_prop_sym(rel_term , prop_term) |
                    rel_prop_sym_unary(prop_term)
rel_symbol      ::= true | false | id | div | or | and | not |
                    implies | implied | equiv | comp | sum |
                    conv | identifier
rel_prop_sym    ::= domrestr | ranrestr
rel_prop_sym_unary ::= test
```

Prove

```
proof list  ::= list_of_proof{(proof_type{, assoc_list})}.
                {step(reference , result , rule_appl ,
                parent _list{, assoc_list}).}*
                end_of_list.
reference   ::= term | identifier | user_reference
result      ::= term | user result
rule_appl   ::= term | identifier | user_rule_appl
parent_list ::= [parent{, parent}*]
parent      ::= term | identifier | user_parent
assoc_list  ::= [key:value{,key:value}*]
key         ::= term | identifier | user_key
value       ::= term | identifier | user_value
proof_type  ::= identifier | user_proof_type
```

Prove predefinite di SPASS

```
user_reference  ::= number
user_result     ::= cnf_clause
user_rule_appl  ::= GeR | SpL | SpR | EqF | Rew | Obv |
                    EmS | SoR | EqR | MPm | SPm | OPm |
                    SHy | OHy | URR | Fac | Spt | Inp |
                    Con | RRE | SSi | ClR | UnC | Ter
user_parent     ::= number
```

```
user_proof_type ::= SPASS
user_key        ::= splitlevel
user_value      ::= number
```

A.2.2 Definizione campo "description"

```
description ::=  list_of_descriptions.
                 name( {* text *}).
                 author({* text *}).
                 {version({* text *}).}
                 {logic({* text *}).}
                 status(log_state).
                 description({* text *}).
                 {date({* text *}).}
                 end of list.
log state ::= satisfiable | unsatisfiable | unknown
```

A.2.3 Definizione campo "settings"

```
settings ::= list_of_general_settings
             {setting_entry}+
             end_of_list.
             |
             list_of_settings(setting_label).
             {* text *}
             end_of_list.
setting_entry  ::= hypothesis[label, {, label}*].
setting_label  ::= KIV | LEM | OTHER | PROTEIN |
                   SATURATE | 3TAP | SETHEO | SPASS
```

Appendice B

Risposte dei problemi

Riportiamo ora l'output dei principali esempi discussi nei capitoli precedenti. Si tratta di lunghi campi testuali che non sono stati modificati, salvo nella formattazione per permetterne l'impaginazione.

B.1 Esempio 1.A - Risposte

```
-------------------------SPASS-START-----------------------------
Input Problem:
1[0:Inp]  ||  -> mioPredicato(miaFunzione(b,c),miaFunzione(a,c))*.
2[0:Inp]  ||  -> mioPredicato(miaFunzione(a,b),miaFunzione(b,c))*.
3[0:Inp]  || mioPredicato(miaFunzione(a,b),miaFunzione(a,c))* -> .
4[0:Inp]  || mioPredicato(U,V)* mioPredicato(W,U)* -> mioPredicato(W,V)*.
 This is a first-order Horn problem without equality.
 The conjecture is ground.
 Axiom clauses: 3 Conjecture clauses: 1
 Inferences: IORe
 Reductions: RFClR RBClR RObv RUnC RTaut RFSub RBSub RCon
 Extras    : Input Saturation, Dynamic Selection, No Splitting,
 Full Reduction,  Ratio: 5, FuncWeight: 1, VarWeight: 1
 Precedence: miaFunzione > div > id > mioPredicato > c > b > a
 Ordering  : KBO
Processed Problem:

Worked Off Clauses:

Usable Clauses:
2[0:Inp]  ||  -> mioPredicato(miaFunzione(a,b),miaFunzione(b,c))*.
```

```
1[0:Inp]  ||  -> mioPredicato(miaFunzione(b,c),miaFunzione(a,c))*.
3[0:Inp]  || mioPredicato(miaFunzione(a,b),miaFunzione(a,c))* -> .
4[0:Inp]  || mioPredicato(U,V)* mioPredicato(W,U)* -> mioPredicato(W,V)*.
5[0:Res:4.2,3.0]  || mioPredicato(U,miaFunzione(a,c))*
                        mioPredicato(miaFunzione(a,b),U)* -> .
Given clause: 2[0:Inp]  ||  -> mioPredicato(miaFunzione(a,b),
                                    miaFunzione(b,c))*.
Given clause: 1[0:Inp]  ||  -> mioPredicato(miaFunzione(b,c),
                                    miaFunzione(a,c))*.
Given clause: 3[0:Inp]  || mioPredicato(miaFunzione(a,b),
                                    miaFunzione(a,c))* -> .
Given clause: 4[0:Inp]  || mioPredicato(U,V)* mioPredicato(W,U)*
                                    -> mioPredicato(W,V)*.
Given clause: 6[0:Res:2.0,4.0]  || mioPredicato(U,miaFunzione(a,b)) ->
                                     mioPredicato(U,miaFunzione(b,c))*.
Given clause: 7[0:Res:1.0,4.0]  || mioPredicato(U,miaFunzione(b,c))* ->
                                      mioPredicato(U,miaFunzione(a,c)).
SPASS V 1.0.0t
SPASS beiseite: Proof found.
Problem: Interactive Web Input, Tue Dec 14 17:49:30 2004
SPASS derived 6 clauses, backtracked 0 clauses and kept 10 clauses.
SPASS allocated 1143 KBytes.
SPASS spent 0:00:00.06 on the problem.
0:00:00.03 for the input.
0:00:00.02 for the FLOTTER CNF translation.
0:00:00.00 for inferences.
0:00:00.00 for the backtracking.
0:00:00.00 for the reduction.


Here is a proof with depth 2, length 7 :
1[0:Inp]  ||  -> mioPredicato(miaFunzione(b,c),miaFunzione(a,c))*.
2[0:Inp]  ||  -> mioPredicato(miaFunzione(a,b),miaFunzione(b,c))*.
3[0:Inp]  || mioPredicato(miaFunzione(a,b),miaFunzione(a,c))* -> .
4[0:Inp]  || mioPredicato(U,V)*+ mioPredicato(W,U)* -> mioPredicato(W,V)*.
7[0:Res:1.0,4.0]  || mioPredicato(U,miaFunzione(b,c))*
                                -> mioPredicato(U,miaFunzione(a,c)).
9[0:Res:2.0,7.0]  ||  -> mioPredicato(miaFunzione(a,b),miaFunzione(a,c))*.
11[0:ClR:9.0,3.0]  ||  -> .
Formulae used in the proof : axiom1 axiom0 conjecture0 axiom2
```

```
-------------------------SPASS-STOP-----------------------------
```

B.2 Esempio 1.B - Risposte

```
-------------------------SPASS-START----------------------------
Input Problem:
1[0:Inp]  ||  -> mioPredicato(miaFunzione(b,c),miaFunzione(a,c))*.
2[0:Inp]  ||  -> mioPredicato(miaFunzione(a,b),miaFunzione(b,c))*.
3[0:Inp]  || mioPredicato(U,V)* mioPredicato(W,U)* -> mioPredicato(W,V)*.
 This is a first-order Horn problem without equality.
 Axiom clauses: 3 Conjecture clauses: 0
 Inferences: IORe
 Reductions: RFClR RBClR RObv RUnC RTaut RFSub RBSub RCon
 Extras    : Input Saturation, Dynamic Selection, No Splitting,
 Full Reduction,  Ratio: 5, FuncWeight: 1, VarWeight: 1
 Precedence: miaFunzione > div > id > mioPredicato > c > b > a
 Ordering  : KBO
Processed Problem:

Worked Off Clauses:

Usable Clauses:
2[0:Inp]  ||  -> mioPredicato(miaFunzione(a,b),miaFunzione(b,c))*.
1[0:Inp]  ||  -> mioPredicato(miaFunzione(b,c),miaFunzione(a,c))*.
3[0:Inp]  || mioPredicato(U,V)* mioPredicato(W,U)* -> mioPredicato(W,V)*.
Given clause: 2[0:Inp]  ||  -> mioPredicato(miaFunzione(a,b),
                                        miaFunzione(b,c))*.
Given clause: 1[0:Inp]  ||  -> mioPredicato(miaFunzione(b,c),
                                        miaFunzione(a,c))*.
Given clause: 3[0:Inp]  || mioPredicato(U,V)* mioPredicato(W,U)*
                                        -> mioPredicato(W,V)*.
Given clause: 4[0:Res:2.0,3.0]  || mioPredicato(U,miaFunzione(a,b)) ->
                                mioPredicato(U,miaFunzione(b,c))*.
Given clause: 5[0:Res:1.0,3.0]  || mioPredicato(U,miaFunzione(b,c))* ->
                                mioPredicato(U,miaFunzione(a,c)).
Given clause: 7[0:Res:2.0,5.0]  ||  -> mioPredicato(miaFunzione(a,b),
                                miaFunzione(a,c))*.
Given clause: 8[0:Res:4.1,5.0]  || mioPredicato(U,miaFunzione(a,b)) ->
                                mioPredicato(U,miaFunzione(a,c))*.
Given clause: 6[0:Res:4.1,3.0]  || mioPredicato(U,miaFunzione(a,b))*
```

```
            mioPredicato(V,U)* -> mioPredicato(V,miaFunzione(b,c))*.
Given clause: 10[0:Res:8.1,3.0]  || mioPredicato(U,miaFunzione(a,b))*
            mioPredicato(V,U)* -> mioPredicato(V,miaFunzione(a,c))*.
SPASS V 1.0.0t
SPASS beiseite: Completion found.
Problem: Interactive Web Input, Tue Dec 14 17:50:48 2004
SPASS derived 7 clauses, backtracked 0 clauses and kept 9 clauses.
SPASS allocated 1141 KBytes.
SPASS spent 0:00:00.03 on the problem.
0:00:00.01 for the input.
0:00:00.00 for the FLOTTER CNF translation.
0:00:00.00 for inferences.
0:00:00.00 for the backtracking.
0:00:00.00 for the reduction.


 The saturated set of worked-off clauses is :
10[0:Res:8.1,3.0]  || mioPredicato(U,miaFunzione(a,b))*+
                 mioPredicato(V,U)* -> mioPredicato(V,miaFunzione(a,c))*.
6[0:Res:4.1,3.0]  || mioPredicato(U,miaFunzione(a,b))*+
                  mioPredicato(V,U)* -> mioPredicato(V,miaFunzione(b,c))*.
8[0:Res:4.1,5.0]  || mioPredicato(U,miaFunzione(a,b)) ->
                  mioPredicato(U,miaFunzione(a,c))*.
7[0:Res:2.0,5.0]  ||  -> mioPredicato(miaFunzione(a,b),miaFunzione(a,c))*.
5[0:Res:1.0,3.0]  || mioPredicato(U,miaFunzione(b,c))* ->
                     mioPredicato(U,miaFunzione(a,c)).
4[0:Res:2.0,3.0]  || mioPredicato(U,miaFunzione(a,b)) ->
                      mioPredicato(U,miaFunzione(b,c))*.
3[0:Inp]  || mioPredicato(U,V)*+ mioPredicato(W,U)* -> mioPredicato(W,V)*.
1[0:Inp]  ||  -> mioPredicato(miaFunzione(b,c),miaFunzione(a,c))*.
2[0:Inp]  ||  -> mioPredicato(miaFunzione(a,b),miaFunzione(b,c))*.
-------------------------SPASS-STOP-----------------------------
```

B.3 Esempio 2 - Risposte

```
----------------------TRANSLATION-START-------------------------
 Translation method: relational
 Translation flags: EML EMLAuto EMLFuncNdeQ
 Precedence:  equal > true > false > div > id > x > y > z > r >
                                                 A > B > a > b
```

```
Axioms:
Label : axiom0
 [FOL] (a (x) (z))
 [Simpl] (a (x) (z))
 [FOL] (a (x) (z))
Conjecture:
 [EML] (not (implies (box (r) (implies (A) (B)))
                          (implies (box (r) (A)) (box (r) (B)))))
 [Simpl] (not (implies (box (r) (implies (A) (B)))
                          (implies (box (r) (A)) (box (r) (B)))))
 [RelTr]  (not (forall ( U) (implies (forall ( V) (implies (R_r U V)
               (implies (P_A V) (P_B V)))) (implies (forall ( W)
               (implies (R_r U W) (P_A W))) (forall ( X)
               (implies (R_r U X) (P_B X)))))))
 [EML] (not (implies (box (r) (A)) (box (r) (box (r) (A)))))
 [Simpl] (not (implies (box (r) (A)) (box (r) (box (r) (A)))))
 [RelTr]  (not (forall ( U) (implies (forall ( V) (implies (R_r U V)
               (P_A V))) (forall ( W) (implies (R_r U W) (forall ( X)
               (implies (R_r W X) (P_A X)))))))
 [EML] (not (implies (box (r) (A)) (A)))
 [Simpl] (not (implies (box (r) (A)) (A)))
 [RelTr]  (not (forall ( U) (implies (forall ( V) (implies (R_r U V)
                      (P_A V))) (P_A U))))
 [FOL] (not (a (x) (y)))
 [Simpl] (not (a (x) (y)))
 [FOL] (not (a (x) (y)))
 [FOL] (not (a (y) (z)))
 [Simpl] (not (a (y) (z)))
 [FOL] (not (a (y) (z)))
-------------------------SPASS-START----------------------------
Input Problem:
1[0:Inp]  ||  -> a(x,z)*.
2[0:Inp]  || P_A(skc11)* -> .
3[0:Inp]  || P_A(skc10)* -> .
4[0:Inp]  ||  -> R_r(skc8,skc9)*.
5[0:Inp]  ||  -> R_r(skc9,skc10)*.
6[0:Inp]  ||  -> R_r(skc6,skc7)*.
7[0:Inp]  || P_B(skc7)* -> .
8[0:Inp]  || a(y,z)* -> .
9[0:Inp]  || a(x,y)* -> .
10[0:Inp]  || R_r(skc11,U)* -> P_A(U).
```

```
11[0:Inp]  || R_r(skc8,U)* -> P_A(U).
12[0:Inp]  || R_r(skc6,U)* -> P_A(U).
13[0:Inp]  || P_A(U) R_r(skc6,U)* -> P_B(U).
 This is a first-order Horn problem without equality.
 This is a problem that has, if any, a finite domain model.
 There are no function symbols.
 This is a problem that contains sort information.
 Axiom clauses: 1 Conjecture clauses: 12
 Inferences: IEmS ISoR IORe
 Reductions: RFClR RBClR RObv RUnC RTaut RSST RSSi RFSub RBSub RCon
 Extras    : No Input Saturation, Always Selection, No Splitting,
                 Full Reduction,  Ratio: 5, FuncWeight: 1, VarWeight: 1
 Precedence: div > id > r > A > B > a > b > P_B > P_A > R_r > skc11
                     > skc10 > skc9 > skc8 > skc7 > skc6 > skc5 > skc4
                     > skc3 > skc2 > skc1 > skc0 > z > y > x
 Ordering  : KBO
Processed Problem:

Worked Off Clauses:

Usable Clauses:
7[0:Inp]  || P_B(skc7)* -> .
6[0:Inp]  ||  -> R_r(skc6,skc7)*.
5[0:Inp]  ||  -> R_r(skc9,skc10)*.
4[0:Inp]  ||  -> R_r(skc8,skc9)*.
3[0:Inp]  || P_A(skc10)* -> .
2[0:Inp]  || P_A(skc11)* -> .
1[0:Inp]  ||  -> a(x,z)*.
9[0:Inp]  || a(x,y)* -> .
8[0:Inp]  || a(y,z)* -> .
12[0:Inp]  || R_r(skc6,U)* -> P_A(U).
11[0:Inp]  || R_r(skc8,U)* -> P_A(U).
10[0:Inp]  || R_r(skc11,U)* -> P_A(U).
14[0:ClR:13.0,12.1]  || R_r(skc6,U)* -> P_B(U).
Given clause: 7[0:Inp]  || P_B(skc7)* -> .
Given clause: 6[0:Inp]  ||  -> R_r(skc6,skc7)*.
Given clause: 5[0:Inp]  ||  -> R_r(skc9,skc10)*.
Given clause: 4[0:Inp]  ||  -> R_r(skc8,skc9)*.
Given clause: 3[0:Inp]  || P_A(skc10)* -> .
Given clause: 2[0:Inp]  || P_A(skc11)* -> .
Given clause: 1[0:Inp]  ||  -> a(x,z)*.
```

```
Given clause: 9[0:Inp]  || a(x,y)* -> .
Given clause: 8[0:Inp]  || a(y,z)* -> .
Given clause: 12[0:Inp]  || R_r(skc6,U)* -> P_A(U).
Given clause: 19[0:Res:6.0,12.0]  ||  -> P_A(skc7)*.
Given clause: 11[0:Inp]  || R_r(skc8,U)* -> P_A(U).
Given clause: 20[0:Res:4.0,11.0]  ||  -> P_A(skc9)*.
Given clause: 10[0:Inp]  || R_r(skc11,U)* -> P_A(U).
Given clause: 14[0:ClR:13.0,12.1]  || R_r(skc6,U)* -> P_B(U).
SPASS V 1.0.0t
SPASS beiseite: Proof found.
Problem: Interactive Web Input, Sun Jan  9 16:11:37 2005
SPASS derived 3 clauses, backtracked 0 clauses and kept 16 clauses.
SPASS allocated 1269 KBytes.
SPASS spent 0:00:00.04 on the problem.
0:00:00.01 for the input.
0:00:00.01 for the FLOTTER CNF translation, of which
0:00:00.00 for the translation from EML to FOL.
0:00:00.00 for inferences.
0:00:00.00 for the backtracking.
0:00:00.00 for the reduction.


Here is a proof with depth 1, length 7 :
6[0:Inp]  ||  -> R_r(skc6,skc7)*.
7[0:Inp]  || P_B(skc7)*+ -> .
12[0:Inp]  || R_r(skc6,U)*+ -> P_A(U).
13[0:Inp] P_A(U) || R_r(skc6,U)* -> P_B(U).
14[0:ClR:13.0,12.1]  || R_r(skc6,U)*+ -> P_B(U).
21[0:Res:6.0,14.0]  ||  -> P_B(skc7)*.
22[0:ClR:21.0,7.0]  ||  -> .
Formulae used in the proof : conjecture0

--------------------------SPASS-STOP------------------------------
```

B.4 Esempio 3 - Risposte

```
----------------------TRANSLATION-START-------------------------
 Translation method: relational
 Translation flags: EML EMLAuto EMLFuncNdeQ
 Precedence:  equal > true > false > div > id > anna > bob > maiale
```

```
              > mucca > pollo > pomodoro > insalata > PERSONA > UOMO
              > DONNA > CIBO > VEGETALE > ANIMALE > VEGETARIANO
              > persona > uomo > donna > cibo > vegetale > animale
              > vegetariano > Mangia
Axioms:
Label : axiom0
 [EML] (implies (or (ANIMALE) (VEGETALE)) (CIBO))
 [Simpl] (implies (or (ANIMALE) (VEGETALE)) (CIBO))
 [RelTr]  (forall ( U) (implies (or (P_ANIMALE U) (P_VEGETALE U)) (P_CIBO U)))
Label : axiom1
 [FOL] (forall ( U V) (implies (and (animale V)
        (not (Mangia U V))) (vegetariano U)))
 [Simpl] (forall ( U V) (implies (and (animale V)
          (not (Mangia U V))) (vegetariano U)))
 [FOL] (forall ( U V) (implies (and (animale V)
        (not (Mangia U V))) (vegetariano U)))
Label : axiom2
 [EML] (implies (PERSONA) (or (UOMO) (DONNA)))
 [Simpl] (implies (PERSONA) (or (UOMO) (DONNA)))
 [RelTr]  (forall ( U) (implies (P_PERSONA U) (or (P_UOMO U) (P_DONNA U))))
Label : axiom3
 [EML] (implies (UOMO) (PERSONA))
 [Simpl] (implies (UOMO) (PERSONA))
 [RelTr]  (forall ( U) (implies (P_UOMO U) (P_PERSONA U)))
Label : axiom4
 [EML] (implies (DONNA) (PERSONA))
 [Simpl] (implies (DONNA) (PERSONA))
 [RelTr]  (forall ( U) (implies (P_DONNA U) (P_PERSONA U)))
Label : axiom5
 [EML] (implies (VEGETARIANO) (PERSONA))
 [Simpl] (implies (VEGETARIANO) (PERSONA))
 [RelTr]  (forall ( U) (implies (P_VEGETARIANO U) (P_PERSONA U)))
Label : axiom6
 [FOL] (donna (anna))
 [Simpl] (donna (anna))
 [FOL] (donna (anna))
Label : axiom7
 [FOL] (uomo (bob))
 [Simpl] (uomo (bob))
 [FOL] (uomo (bob))
Label : axiom8
```

```
 [FOL] (animale (mucca))
 [Simpl] (animale (mucca))
 [FOL] (animale (mucca))
Label : axiom9
 [FOL] (animale (maiale))
 [Simpl] (animale (maiale))
 [FOL] (animale (maiale))
Label : axiom10
 [FOL] (animale (pollo))
 [Simpl] (animale (pollo))
 [FOL] (animale (pollo))
Label : axiom11
 [FOL] (vegetale (insalata))
 [Simpl] (vegetale (insalata))
 [FOL] (vegetale (insalata))
Label : axiom12
 [FOL] (vegetale (pomodoro))
 [Simpl] (vegetale (pomodoro))
 [FOL] (vegetale (pomodoro))
Label : axiom13
 [FOL] (Mangia (anna) (maiale))
 [Simpl] (Mangia (anna) (maiale))
 [FOL] (Mangia (anna) (maiale))
Label : axiom14
 [FOL] (Mangia (bob) (insalata))
 [Simpl] (Mangia (bob) (insalata))
 [FOL] (Mangia (bob) (insalata))
Label : axiom15
 [FOL] (Mangia (bob) (pomodoro))
 [Simpl] (Mangia (bob) (pomodoro))
 [FOL] (Mangia (bob) (pomodoro))
Label : axiom16
 [FOL] (not (Mangia (bob) (maiale)))
 [Simpl] (not (Mangia (bob) (maiale)))
 [FOL] (not (Mangia (bob) (maiale)))
Label : axiom17
 [FOL] (not (Mangia (bob) (mucca)))
 [Simpl] (not (Mangia (bob) (mucca)))
 [FOL] (not (Mangia (bob) (mucca)))
Label : axiom18
 [FOL] (not (Mangia (bob) (pollo)))
```

```
 [Simpl] (not (Mangia (bob) (pollo)))
 [FOL] (not (Mangia (bob) (pollo)))
Conjecture:
-------------------------SPASS-START-----------------------------
Input Problem:
1[0:Inp]  ||  -> vegetale(pomodoro)*.
2[0:Inp]  ||  -> vegetale(insalata)*.
3[0:Inp]  ||  -> animale(pollo)*.
4[0:Inp]  ||  -> animale(maiale)*.
5[0:Inp]  ||  -> animale(mucca)*.
6[0:Inp]  ||  -> uomo(bob)*.
7[0:Inp]  ||  -> donna(anna)*.
8[0:Inp]  ||  -> Mangia(bob,pomodoro)*.
9[0:Inp]  ||  -> Mangia(bob,insalata)*.
10[0:Inp]  ||  -> Mangia(anna,maiale)*.
11[0:Inp]  || Mangia(bob,pollo)* -> .
12[0:Inp]  || Mangia(bob,mucca)* -> .
13[0:Inp]  || Mangia(bob,maiale)* -> .
14[0:Inp]  || P_VEGETARIANO(U) -> P_PERSONA(U)*.
15[0:Inp]  || P_DONNA(U)* -> P_PERSONA(U).
16[0:Inp]  || P_UOMO(U)* -> P_PERSONA(U).
17[0:Inp]  || P_VEGETALE(U) -> P_CIBO(U)*.
18[0:Inp]  || P_ANIMALE(U) -> P_CIBO(U)*.
19[0:Inp]  || P_PERSONA(U) -> P_DONNA(U)* P_UOMO(U).
20[0:Inp]  || animale(U) -> vegetariano(V) Mangia(V,U)*.
 This is a first-order Non-Horn problem without equality.
 This is a problem that has, if any, a finite domain model.
 There are no function symbols.
 This is a problem that contains sort information.
 The following monadic predicates have finite extensions: donna, uomo,
              animale, vegetale.
 Axiom clauses: 20 Conjecture clauses: 0
 Inferences: IEmS ISoR IORe IOFc
 Reductions: RFClR RBClR RObv RUnC RTaut RSST RSSi RFSub RBSub RCon
 Extras    : Input Saturation, Always Selection, Full Splitting,
 Full Reduction,  Ratio: 5, FuncWeight: 1, VarWeight: 1
 Precedence: div > id > PERSONA > UOMO > DONNA > CIBO > VEGETALE
    > ANIMALE > VEGETARIANO > persona > uomo > donna > cibo
    > vegetale > Mangia > vegetariano > animale > P_CIBO > P_ANIMALE
    > P_VEGETALE > P_DONNA > P_UOMO > P_PERSONA > P_VEGETARIANO >
    insalata > pomodoro > pollo > mucca > maiale > bob > anna
```

```
 Ordering  : KBO
Processed Problem:

Worked Off Clauses:

Usable Clauses:
7[0:Inp]  ||  -> donna(anna)*.
6[0:Inp]  ||  -> uomo(bob)*.
5[0:Inp]  ||  -> animale(mucca)*.
4[0:Inp]  ||  -> animale(maiale)*.
3[0:Inp]  ||  -> animale(pollo)*.
2[0:Inp]  ||  -> vegetale(insalata)*.
1[0:Inp]  ||  -> vegetale(pomodoro)*.
10[0:Inp]  ||  -> Mangia(anna,maiale)*.
9[0:Inp]  ||  -> Mangia(bob,insalata)*.
8[0:Inp]  ||  -> Mangia(bob,pomodoro)*.
13[0:Inp]  || Mangia(bob,maiale)* -> .
12[0:Inp]  || Mangia(bob,mucca)* -> .
11[0:Inp]  || Mangia(bob,pollo)* -> .
18[0:Inp] P_ANIMALE(U) ||  -> P_CIBO(U)*.
17[0:Inp] P_VEGETALE(U) ||  -> P_CIBO(U)*.
16[0:Inp] P_UOMO(U) ||  -> P_PERSONA(U)*.
15[0:Inp] P_DONNA(U) ||  -> P_PERSONA(U)*.
14[0:Inp] P_VEGETARIANO(U) ||  -> P_PERSONA(U)*.
19[0:Inp] P_PERSONA(U) ||  -> P_DONNA(U)* P_UOMO(U).
20[0:Inp] animale(U) ||  -> vegetariano(V) Mangia(V,U)*.
Given clause: 7[0:Inp]  ||  -> donna(anna)*.
Given clause: 6[0:Inp]  ||  -> uomo(bob)*.
Given clause: 5[0:Inp]  ||  -> animale(mucca)*.
Given clause: 4[0:Inp]  ||  -> animale(maiale)*.
Given clause: 3[0:Inp]  ||  -> animale(pollo)*.
Given clause: 2[0:Inp]  ||  -> vegetale(insalata)*.
Given clause: 1[0:Inp]  ||  -> vegetale(pomodoro)*.
Given clause: 10[0:Inp]  ||  -> Mangia(anna,maiale)*.
Given clause: 9[0:Inp]  ||  -> Mangia(bob,insalata)*.
Given clause: 8[0:Inp]  ||  -> Mangia(bob,pomodoro)*.
Given clause: 13[0:Inp]  || Mangia(bob,maiale)* -> .
Given clause: 12[0:Inp]  || Mangia(bob,mucca)* -> .
Given clause: 11[0:Inp]  || Mangia(bob,pollo)* -> .
Given clause: 18[0:Inp] P_ANIMALE(U) ||  -> P_CIBO(U)*.
Given clause: 17[0:Inp] P_VEGETALE(U) ||  -> P_CIBO(U)*.
```

```
Given clause: 16[0:Inp] P_UOMO(U) ||  -> P_PERSONA(U)*.
Given clause: 15[0:Inp] P_DONNA(U) ||  -> P_PERSONA(U)*.
Given clause: 14[0:Inp] P_VEGETARIANO(U) ||  -> P_PERSONA(U)*.
Given clause: 19[0:Inp] P_PERSONA(U) ||  -> P_DONNA(U)* P_UOMO(U).
Given clause: 20[0:Inp] animale(U) ||  -> vegetariano(V) Mangia(V,U)*.
Given clause: 39[0:SSi:36.0,4.0]  ||  -> vegetariano(bob)*.
SPASS V 1.0.0t
SPASS beiseite: Completion found.
Problem: Interactive Web Input, Sun Jan  9 11:25:04 2005
SPASS derived 3 clauses, backtracked 0 clauses and kept 21 clauses.
SPASS allocated 1277 KBytes.
SPASS spent 0:00:00.05 on the problem.
0:00:00.01 for the input.
0:00:00.01 for the FLOTTER CNF translation, of which
0:00:00.00 for the translation from EML to FOL.
0:00:00.00 for inferences.
0:00:00.00 for the backtracking.
0:00:00.00 for the reduction.


 The saturated set of worked-off clauses is :
39[0:SSi:36.0,4.0]  ||  -> vegetariano(bob)*.
20[0:Inp] animale(U) ||  -> vegetariano(V) Mangia(V,U)*.
19[0:Inp] P_PERSONA(U) ||  -> P_DONNA(U)* P_UOMO(U).
14[0:Inp] P_VEGETARIANO(U) ||  -> P_PERSONA(U)*.
15[0:Inp] P_DONNA(U) ||  -> P_PERSONA(U)*.
16[0:Inp] P_UOMO(U) ||  -> P_PERSONA(U)*.
17[0:Inp] P_VEGETALE(U) ||  -> P_CIBO(U)*.
18[0:Inp] P_ANIMALE(U) ||  -> P_CIBO(U)*.
11[0:Inp]  || Mangia(bob,pollo)*+ -> .
12[0:Inp]  || Mangia(bob,mucca)*+ -> .
13[0:Inp]  || Mangia(bob,maiale)*+ -> .
8[0:Inp]  ||  -> Mangia(bob,pomodoro)*.
9[0:Inp]  ||  -> Mangia(bob,insalata)*.
10[0:Inp]  ||  -> Mangia(anna,maiale)*.
1[0:Inp]  ||  -> vegetale(pomodoro)*.
2[0:Inp]  ||  -> vegetale(insalata)*.
3[0:Inp]  ||  -> animale(pollo)*.
4[0:Inp]  ||  -> animale(maiale)*.
5[0:Inp]  ||  -> animale(mucca)*.
6[0:Inp]  ||  -> uomo(bob)*.
```

```
7[0:Inp]  ||  -> donna(anna)*.
--------------------------SPASS-STOP-----------------------------
```

Bibliografia

[1] Sito ufficiale di MPSASS: *http://www.cs.man.ac.uk/*

$\sim$

schmidt/mspass/

[2] Reiner Hahnle, Manfred Kerber, Christoph Weidenbach, Renate A. Schmidt *Common Syntax of the DFG-Schwerpunktprogramm Deduktion*

[3] H. J. Ohlbach *Semantics based translation methods for modal logics* Ed. J. Logic Computat. , 1991.

[4] H. J. Ohlbach and R. A. Schmidt *Functional translation and second-order frame properties of modal logics* Ed. J. Logic Computat. , 1997