Lista dei Pesi

Distinguiamo le seguenti liste di pesi:

- `pick_given`: per scegliere la given_clause dalla lista sos;
- `purge_gen`: per scartare le clausole generate che superano la complessitá massima (parametro `max_weight`);
- `pick_and_purge`: é l'unione delle due precedenti.
- `terms`: per definire il peso dei termini quando si utilizza `weight_list_terms`.

Lista dei Pesi (Esempio)

```
weight_list(pick_and_purge).
  weight(a, 0).                      % il peso della costante a e' 0
  weight(g($(2)), -50).              % due volte il peso dell'argomento
                                     % di g - 50
  weight(P($(1),$(1)), 100).         % la somma dei pesi di P + 100
  weight(x, 5).                      % tutte le variabili hanno peso 5
  weight(f(g($(3)),$(4)), -300). % 3 volte il peso dell'argomento di
                                     % g + 4 volte il peso del secondo
                                     % argomento di f - 300
end_of_list.
```

Operatori

op(precedenza, tipo, funtore).

- Funtore: nome del nuovo operatore;
- Precedenza [0-1000]. Piú é piccolo e piú la precedenza é alta;
- Tipo [xfx, xfy, yfx, fx, fy, xf, yf]: y per richiamare ricorsivamente l'operatore;

  Es: "+" é di tipo xfy: `op(500, xfy, +)`

  a+b+c equivale: a+(b+c).

Ordinamento (1)

Otter esegue due tipi di ordinamento:

- Ad hoc: Collezione di metodi empirici per l'ordinamento dei termini;
- LRPO (lexicographic recursively path ordering): è abilitato settando il flag lrpo.

Ordinamento ad hoc: é composto da due tecniche che Otter sceglie automaticamente a seconda del contesto.

- La prima é il classico ordinamento alfabetico.
- La seconda tecnica invece dell'ordinamento alfabetico utilizza la funzione di pesatura (`weight_list_term`) per determinare l'ordine fra due termini.

Ordinamento (2)

Ordinamento LRPO: é garantita la terminazione del processo di demodulazione.

L'ordinamento é totale e l'utente puó fissarlo tramite il comando `lex`.

Ad esempio il comando `lex([a, b, c, d, or(_,_)])` imposta il seguente ordine: `a` $\prec$ `b` $\prec$ `c` $\prec$ `d` $\prec$ `or(_,_)`.

Demodulazione (1)

Processo che gestisce le uguaglianze.

Le uguaglianze sono elencate nella lista `demodulators`.

Si applicano le uguaglianze nella lista dei demodulatori alle clausole generate dalle regole di inferenza.

Se `demod_inf` ON: demodulazione usata come una regola di inferenza.

Se `process_input` ON: cerca demodulatori in *usable* e *sos*.

Se `back_demod` ON: ogni nuovo demodulatore viene utilizzato per demodulare le clausole in `demodulators`, `usable` e `sos`.

`LRPO` garantisce la terminazione della demodulazione. In ogni caso si puó utilizzare il parametro `demod_limit` per limitare il numero di demodulazioni.

Demodulazione (2)

Puó essere usata per eliminare clausole che soddisfano particolari criteri. Ad esempio per eliminare tutte le clausole che contengono il termine `f(x)` basta aggiungere il demod. `f(x)=delete` e associare al simbolo `delete` un peso maggiore del parametro `max_weight`.

Se `demod_out_in` ON: la modulazione é applicata dal termine piú esterno fino ai piú interni.

L'ordinamento é utilizzato anche per

- scegliere in che verso verrá applicato un demodulatore (cioé se l'uguaglianza sará applicata da destra verso sinistra o viceversa);
- decidere se un'uguaglianza inferita sará inserita nella lista dei demodulatori.

Valutazione di funzioni e predicati (1)

Modalitá **programmata**. Uso di simboli `$`.

Es: `$SUM, $LT`.

Scrivere espressioni utilizzando `$SUM` puó essere tedioso.

Quindi si usa `make_evaluable` per creare un alias del simbolo.

Es: `make_evaluable(_+_,$SUM(_,_))` associa il simbolo `$SUM` con `+`.

Questi simboli valutabili operano su quattro tipi di termini:

- costanti intere;
- stringhe di bit;
- costanti booleane;
- termini arbitrari.

Valutazione di funzioni e predicati (2)

$int \times int \rightarrow int$	`$SUM`, `$PROD`, `$DIFF`, `$DIV`, `$MOD`
$int \times int \rightarrow bool$	`$EQ`, `$NE`, `$LT`, `$LE`, `$GT`, `$GE`
$bits \times bits \rightarrow bits$	`$BIT_AND`, `$BIT_OR`, `$BIT_XOR`
$bits \times int \rightarrow bits$	`$SHIFT_LEFT`, `$SHIFT_RIGHT`
$bits \rightarrow bits$	`$BIT_NOT`
$int \rightarrow bits$	`$INT_TO_BITS`
$bits \rightarrow int$	`$BITS_TO_INT`
$term \times term \rightarrow bool$ (lexical)	`$ID`, `$LNE`, `$LLT`, `$LLE`, `$LGT`, `$LGE`
$\rightarrow bool$	`$T`, `$F`
$bool \times bool \rightarrow bool$	`$AND`, `$OR`
$bool \rightarrow bool$	`$TRUE`, `$NOT`
$term \rightarrow bool$	`$ATOMIC`, `$INT`, `$BITS`, `$VAR`, `$GROUND`
$\rightarrow int$	`$NEXT_CL_NUM`
$bool \times term \times term \rightarrow term$	`$IF`

Valutazione di funzioni e predicati (3)

La valutazione é una parte del processo di **demodulazione**:
Es: se la demodulazione incontra il termine `$SUM(2,3)` sostituisce al termine il risultato della valutazione come se ci fosse il demodulatore `$SUM(2,3)=5`.

La valutazione puó avvenire anche durante la **hyper-resolution**.
Se il predicato negativo della hyper-resolution è valutabile allora si demodula il termine. Se il risultato della demodulazione è `$T` allora è considerato risolto.
Es: `... | -(x+y<2) | ...` x+y viene valutato come se ci fosse il demodulatore `$SUM(x,y) = z`. Se z è minore di 2 il risultato e' `$T` e il letterale e' risolto.

Valutazione di funzioni e predicati (4)

In tabella si mostra i predicati che accettano come parametri delle costanti float.

$float \times float \rightarrow float$	`$FSUM`, `$FPROD`, `$FDIFF`, `$FDIV`, `$FMOD`
$float \times float \rightarrow bool$	`$FEQ`, `$FNE`, `$FLT`, `$FLE`, `$FGT`, `$FGE`

Il Problema delle parentele

Dimostrare che il genero della moglie del padre del fratello di mia madre è mio padre.

Soluzione del Problema delle Parentele

```
set(auto).

formula_list(usable).

all x y z (fratello(x,y) & madre(y,z) -> zio(x,z)).
all x y z (papa(x,y) & zio(y,z) -> nonno(x,z)).
all x y z (moglie(x,y) & nonno(y,z) -> nonna(x,z)).
all x y z (genero(x,y) & nonna(y,z) -> padre(x,z)|zio(x,z)).
all x y z (nuora(x,y) & nonna(y,z) -> madre(x,z)|zia(x,z)).

madre(Daniela,Alessandro).
fratello(Flavio,Daniela).
papa(Piero,Flavio).
moglie(Lina,Piero).
genero(Riccardo,Lina).
-padre(Riccardo,Alessandro).
-zio(Riccardo,Alessandro).
end_of_list.
```

Il Problema delle Talpe Pigre

C'erano una volta 3 talpe. Le 3 talpe abitavano tutte nella stessa buca con mamma talpa finche' un bel giorno la mamma talpa stanca dei suoi figlioli nullafacenti li cacciò di buca. Dato che le 3 talpe erano molto pigre, cercarono dei buchi gia' fatti e disabitati. Dopo numerose peripezie giunsero in una bella e lussureggiante vallata. Decisero così di comune accordo di stabilirvisi, ma a malincuore scoprirono che nella vallata c'erano solo 2 buche disabitate. Dato che nessuna delle 3 talpe volle condividere la tana con una delle altre talpe, dimostrare che non fù possibile per tutte le 3 talpe pigre vivere nella vallata dei loro sogni.

Suggerimento

Risolvere il problema per 2 talpe e 1 buca (generalizzare).

Ogni talpa vive in al piu' un buco.

In ciascun buco vive al piu' una talpa.

Soluzione per 2 talpe e 1 buco

```
set(auto).

list(usable).
% Ogni talpa vive in al piu' un buco.
tb11.
tb21.

% In ciascun buco vive al piu' una talpa.
-tb11 | -tb21.

end_of_list.
```

Soluzione per 3 talpe e 2 buchi

```
set(auto).

list(usable).
% Ogni talpa vive in al piu' un buco.
tb11 | tb12.
tb21 | tb22.
tb31 | tb32.

% In ciascun buco vive al piu' una talpa.
-tb11 | -tb21.
-tb11 | -tb31.
-tb21 | -tb31.

-tb12 | -tb22.
-tb12 | -tb32.
-tb22 | -tb32.

end_of_list.
```

Il Problema dei Fusti

Rosalie, la moglie di Jack il contadino sta facendo una torta di mele. A un certo punto si accorge di aver finito il latte e chiede a Jack di andargli a mungere due litri esatti di latte. Jack esce fischiettando, entra nella stalla, mette lo sgabello davanti alla mucca e si appresta a iniziare la mungitura, quando improvvisamente un terribile dubbio assale la sua mente.
Nella stalla Jack ha solo un fusto da 3 litri e uno da 4.
Come può fare Jack a portare esattamente due litri di latte a Rosalie?

Suggerimento per il Problema dei Fusti

Rappresentare lo stato di riempimento dei due fusti con una funzione del tipo: `ff(x,y)`

Cosa è necessario rappresentare?
Operazione di riempimento/svuotamento del fusto di 3 litri.
Operazione di riempimento/svuotamento del fusto di 4 litri.
Svuotare il fusto da 3 litri nel fusto da 4 litri.
Svuotare il fusto da 4 litri nel fusto da 3 litri.
Svuotare il fusto da 3 litri nel fusto da 4 litri fino all'orlo.
Svuotare il fusto da 4 litri nel fusto da 3 litri fino all'orlo.
Lo stato iniziale: fusti vuoti.
Lo stato goal: uno dei due fusti è pieno.

Soluzione del Problema dei Fusti

```
make_evaluable(_+_, $SUM(_,_)).
make_evaluable(_-_, $DIFF(_,_)).
make_evaluable(_<=_, $LE(_,_)).
make_evaluable(_>_, $GT(_,_)).
set(auto).

list(usable).
-ff(x, y) | ff(3, y).  % riempi il f3
-ff(x, y) | ff(0, y).  % svuota il f3
-ff(x, y) | ff(x, 4).  % riempi il f4
-ff(x, y) | ff(x, 0).  % svuota il f4
-ff(x, y) | -(x+y <= 4) | ff(0, y+x).   % f3 -> f4
-ff(x, y) | -(x+y <= 3) | ff(x+y, 0).  % f4 -> f3
-ff(x, y) | -(x+y >  4) | ff(x - (4-y), 4).   % f3 -> f4 finche' f4 pieno
-ff(x, y) | -(x+y >  3) | ff(3, y - (3-x)).  % f4 -> f3 finche' f3 pieno
ff(0, 0).   % stato iniziale: fusti vuoti
-ff(x, 2).  % stato goal: in uno dei due fusti ci sono 2 litri
end_of_list.
```

Il Problema dei missionari

Sulla riva ovest di un fiume ci sono 3 missionari, 3 cannibali e 1 barca.
Tutti vogliono attraversare il fiume, ma la barca contiene solo 2 persone.
Come se non bastasse se i cannibali si trovassero in sovrannumero su una delle due sponde del fiume o sulla barca non ci penserebbero due volte a sbranare i missionari.
Come è possibile per i sei attraversare il fiume senza che i missionari facciano una brutta fine?

Suggerimento per il Problema dei Missionari (1)

Rappresentare lo stato del problema con un predicato del tipo:
`state(MBS,CBS,BS)`, dove:

- MBS: missionari sul lato del fiume dove si trova la barca;
- CBS: cannibali sul lato del fiume dove si trova la barca;
- BS: lato del fiume dove si trova la barca.

Utilizzare un predicato `pick(M)` e un predicato `pick(C)` per rappresentare rispettivamente i missionari e i cannibali che salgono sulla barca.
`pick(0)`, `pick(1)` e `pick(2)` indica che sulla barca salgono 0, 1 o 2 cannibali/missionari.

Come si può rappresentare l'azione di attraversamento del fiume?
Utilizzare un predicato del tipo `otherside(BS)` nella lista dei demodulatori.

Suggerimento per il Problema dei Missionari (2)

Cosa e' necessario rappresentare?
E' sufficiente una implicazione che modellizza l'attraversamneto della barca.
A sinistra dell'implicazione:

- Dato lo stato attuale
- I cannibali che salgono sulla barca devono essere in numero inferiore ai cannibali che sono presenti sulla riva del fiume (Es: `C<=CBS`). Idem per i missionari.
- Sulla barca ci deve essere qualcuno.
- Sulla barca non ci devono essere più di 2 persone.
- Nessuna carneficina sulla barca (a bordo della barca i missionari devono essere più dei cannibali oppure non ci devono essere affatto).
- Nessuna carneficina dopo che la barca è partita
- Nessuna carneficina dopo che la barca è arrivata

A destra dell'implicazione:

- Stato dopo l'attraversamento della barca.

Soluzione del Problema dei Missionari (1)

```
set(hyper_res).
set(prolog_style_variables).

op(900, xfx, ||).

make_evaluable(_-_, $DIFF(_,_)).
make_evaluable(_+_, $SUM(_,_)).
make_evaluable(_<_, $LT(_,_)).
make_evaluable(_<=_, $LE(_,_)).
make_evaluable(_>_, $GT(_,_)).
make_evaluable(_>=_, $GE(_,_)).
make_evaluable(_==_, $EQ(_,_)).
make_evaluable(_||_, $OR(_,_)).
```

Soluzione del Problema dei Missionari (2)

```
list(usable).
-state(MBS,CBS,BS)  % rappresentazione dello stato
| -pick(M)          % missionari che attraversano
| -pick(C)          % cannibali che attraversano
| -(M<=MBS)
| -(C<=CBS)
| -(M+C>0)          % nella barca ci deve essere qualcuno
| -(M+C<=2)         % nella barca non ci sono piu' di 2 persone
| -(M>=C || M==0)     % I missionari sulla barca devono essere piu'
                    % dei cannibali oppure non ci devono essere
% if no conversions after the boat leaves current side,
| -(MBS-M>=CBS-C || (MBS-M==0))
% if no conversions when the boat arrives at the other side,
| -((3-MBS)+M >= (3-CBS)+C || (3-MBS)+M==0)
% then a crossing can occur
| state((3-CBS)+C, (3-MBS)+M, otherside(BS)).
```

Soluzione del Problema dei Missionari (3)

```
% The following give the number of each that can cross.
pick(0).
pick(1).
pick(2).

-state(3,3,east).   % goal state
end_of_list.

list(sos).
state(3,3,west).    % initial state
end_of_list.

list(demodulators).
otherside(west) = east.
otherside(east) = west.
end_of_list.
```