1 Lista di pesi

Una lista di pesi puó essere dei seguenti tipi:

- pick_given: usata per scegliere la given_clause dalla lista sos;
- purge_gen: in combinazione con il parametro max_weight serve a scartare le clausole generate che superano la complessitá massima.
- pick_and_purge: é l'unione delle due precedenti. Se presente non devono essere presenti le due precedenti.
- terms: usata per definire il peso dei termini quando si utilizza weight_list_terms.

Ad esempio:

```
weight_list(pick_and_purge).
  weight(a, 0).                   % il peso della costante a e' 0
  weight(g($(2)), -50).           % due volte ip peso dell'argomento di g - 50
  weight(P($(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.
```

2 Dichiarazione di nuovi operatori

In Otter si possono dichiarare nuovi funtori da aggiungere a quelli giá predefiniti.
La sintassi per dichiarare un nuovo operatore é:
op(precedenza,tipo,funtore).
Precedenza é un intero compreso tra 0 e 1000. piú é piccolo e piú la precedenza é alta. Il funtore é una stringa del tipo

- forma infissa: `xfx`, `xfy` e `yfx`;
- forma prefissa: `fx` e `fy`;
- forma postfissa: `xf` e `yf`.

`y` indica la possibilitá di richiamare ricorsivamente l'operatore. Ad esempio "+" é di tipo xfy quindi é possibile scrivere la formula a + b + c che verrá tradotta in a+(b+c).

3 Demodulazione

La demodulazione é il processo che gestisce le uguaglianze. Le uguaglianze sono elencate nella lista `demodulators`. Se il flag `demod_inf` é settato allora il processo di demodulazione é utilizzato come una regola di inferenza e quindi é applicato

alla given_clause. Se il flag `process_input` é settato le clausole in usable e sos vengono analizzate alla ricerca di uguaglianze da utilizzare come demodulatori.

Otter ha vari flag per raffinare la tecnica basilare di demodulazione. Ad esempio si puó scegliere se applicare i demodulatori partendo dal predicato piú esterno scendendo poi ricorsivamente ad analizzare i termini di cui é composto. La terminazione é garantita dalla presenza di un limite sul numero di demodulazioni effettuabili. Oppure si puó scegliere il comportamento opposto che prima di demodulare un termine demodula tutti i sottotermini di cui é composto.

Un'altra caratteristica importante é l'ordinamento dei demodulatori che permette di scegliere in che verso verrá applicato un demodulatore (cioé se l'uguaglianza sará applicata da destra verso sinistra o viceversa). Inoltre l'ordinamento é utilizzato per decidere se un'uguaglianza inferita sará inserita nella lista dei demodulatori.

4 Ordinamento

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.

L'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.

L'ordinamento LRPO ha come pregio il fatto di garantire che il processo di demodulazione abbia termine (con il precedente invece potrebbe non terminare). 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`.

5 Valutazione di Funzioni e Predicati

Otter puó usare una modalitá di funzionamento detta "programmata". In questa modalitá spesso si utilizzano i simboli riservati che iniziano con $. Ad esempio: $SUM, $LT.
Questi simboli valutabili operano su quattro tipi di termini:

- costanti intere;
- stringhe di bit;
- costanti booleane;

Tabella 1: Funzioni e Predicati valutabili.

$int \times int \rightarrow int$	`$SUM`, `$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`

- termini arbitrari.

La valutazione é una parte del processo di demodulazione: se la demodulazione incontra ad esempio il termine `$SUM(2,3)` cerca di convertire gli argomenti nei tipi necessari al predicato. Se gli argomenti sono del tipo corretto sostituisce al termine il risultato della della valutazione come se ci fosse il demodulatore `$SUM(2,3)=5`.

La valutazione puó avvenire anche durante l'hyper-resolution. Il meccanismo é il seguente: se l'hyper-risoluzione incontra un predicato negativo allora esso demodula l'atomo (il letterale senza il segno di negazione) se il risultato della demodulazione é `$T` *allora il letterale é considerato risolto.*

Scrivere espressioni utilizzando i simboli che iniziano con `$` puó essere tedioso. Quindi é stato introdotto il comando `make_evaluable` che crea un alias al simbolo. Ad esempio `make_evaluable(_=_,$SUM(_,_))` associa il simbolo `$SUM` con `=`.

5.1 Operazioni in Floating Point

In tabella 2 lista i predicati che accettano come parametri delle costanti float.

Tabella 2: Floating-Point Operations

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