Estruturas de Dados Incompletas

- Diferença de Listas
- Diferença de Estruturas
- Exemplos.

Diferença de Listas

Motivação: Manipulação de listas é bastante ineficiente e.g. *append* de listas.

Ver listas como acumuladores. Isto é as listas são construidas por adição ao fim do resto da lista.

Noção de *diferença de listas*.

Uma lista pode ser vista como a diferença entre duas listas. Por exemplo a lista

$$[1, 2]$$

pode ser vista como a diferença entre a lista

$$[1, 2, 3] \qquad \text{e} \qquad [3]$$

Ou seja, igual a $[1, 2, 3] - [3]$.

Pode também ser a diferença de listas $[1, 2] - []$.

Em geral podemos pensar na diferença entre duas listas *incompletas* i.e. $[1, 2 \mid Xs] - Xs$.

Se instanciarmos a variável Xs com uma lista a juntar obtemos junção de listas de uma forma muito eficiente (em tempo constante contre tempo linear do predicado *append*).

Isto é, $Xs/[4, 5]$ obtemos $[1, 2, 4, 5] - [4, 5]$.

Completar listas incompletas pela forma de *stamped addressed envelope*. i.e. instanciar uma lista incompleta com outra lista incompleta.

Se tivermos

$$[1, 2, 3 \mid Xs] - Xs$$

fazemos

$$Xs/[4, 5 \mid Ys].$$

Sempre que quisermos expandir a lista instanciamos a variável Ys com uma nova lista incompleta.

Definir *append* para diferença de listas:

```
append_dl(Xs-Ys,Ys-Zs,Xs-Zs).
```

Exemplo:

$$?\mathsf{append\_dl}([\mathsf{a}, \mathsf{b}, \mathsf{c} \mid \mathsf{Xs}] - \mathsf{Xs}, [\mathsf{d}, \mathsf{e}] - [], \mathsf{Ys})$$

obtêm-se:

$$\mathsf{Xs} = [\mathsf{d}, \mathsf{e}], \mathsf{Ys} = [\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}] - [].$$

Generalização:

Ao resolvermos a interrogação:

$$?p([1, 2, 3 \mid Xs] - Xs)$$

com a cláusula

$$p(H - T) \leftarrow \ldots \;\&\; literal_1 \;\&\; literal_2 \;\&\; \ldots$$

obtém-se a unificação:

$$H = [1, 2, 3 \mid Xs] \qquad \text{e} \qquad T = Xs$$

Se fizermos, na cláusula:

$$T = [4, 5 \mid T1]$$

obtemos H instanciado com a lista $[1, 2, 3, 4, 5 \mid T1]$.

Podemos agora expandir a lista H instanciando a variável $T1$ com uma lista completa/incompleta.

Exemplos

O predicado flatten transforma uma lista de listas numa lista de elementos atómicos. e.g. ?flatten$([1, [2, 3], [4], [[5], [6]]], X)$ retorna $X = [1, 2, 3, 4, 5, 6]$.

```
flatten([X|Xs],Ys) :-
        flatten(X,Ys1),
        flatten(Xs,Ys2),
        append(Ys1,Ys2,Ys).

flatten(X,[X]) :-
        atom(X),
        X \== [].

flatten([],[]).
```

Versão com diferença de listas

```
flatten(Xs,Ys) :-
        flatten_dl(Xs,Ys-[]).

flatten_dl([X|Xs],Ys-Zs) :-
        flatten_dl(X,Ys-Ys1),
        flatten_dl(Xs,Ys1-Zs).

flatten_dl(X,[X|Xs]-Xs) :-
        atom(X), X \== [].

flatten_dl([],Xs-Xs).
```

Outro exemplo

O predicado reverse já conhecido recorrendo do predicado append.

```
reverse([],[]).

reverse([X|Xs],Zs) :-
        reverse(Xs,Ys), append(Ys,[X],Zs).
```

Versão com diferença de listas

```
reverse([X|Xs],Ys-Zs) :-
        reverse(Xs,Ys-[X|Zs]).

reverse([],Xs-Xs).
```

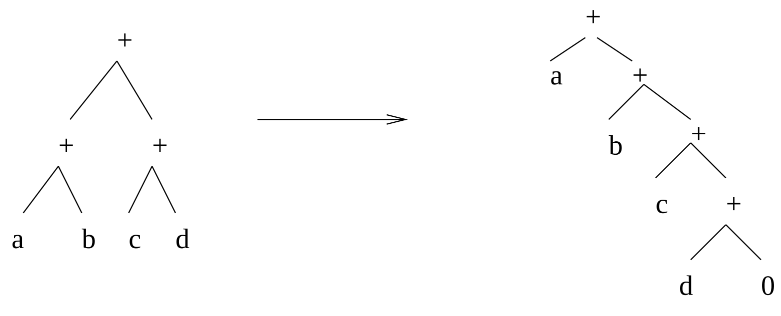
A junção de listas é diluida na construção da diferença de listas. Isto é, a junção é feita incrementalmente.

Diferença de Estruturas

Podemos generalizar esta noção de estruturas incompletas para representar resultados parciais de uma computação.

Exemplo: normalização de somas. A soma $(a + b) + (c + d)$ pode ser reescrita na soma com parêntesis à direita $(a + (b + (c + d)))$. Este processso é útil na simplificação algébrica.

Podemos implementar este processo recorrendo ao uso de uma estrutura de dados incompleta: a diferença de somas. Este diferença é representada pela estrutura $E1 + +E2$ onde $E1$ e $E2$ são somas normalizadas incompletas. ++ é um operador binário definido e 0 é usado para indicar soma vazia.



Normalização de Somas

```
normaliza(Exp,Norm):-
            normaliza_ds(Exp,Norm++0).

normaliza_ds(A+B,Norm++Espaco):-
            normaliza_ds(A,Norm++NormB),
            normaliza_ds(B,NormB++Espaco).

normaliza_ds(A,(A+Espaco)++Espaco):-
            constant(A).
```

Notar que o programa constrói a soma normalizada de uma forma *top-down*. Facilmente podiamos transformar o programa obtendo a construção da mesma estrutura *bottom-up*.