Às vezes uso um nome/conceito antes de defini-lo. Isso é porque acho que deixar a leitora pensar sobre o significado de algo antes de esse lhe ser dado tende a facilitar a memorização posterior. Além disso, no lugar de definir tudo o que vou usar, prefiro definir apenas algumas coisas e esperar que as demais fiquem claras pelo contexto ou pelos exemplos (ou, eventualmente, pode ser porque eu mesmo não sei bem a definição e só trabalho pelos exemplos.. nestes casos, quando eu chegar a uma definição satisfatória, é possível que atualize este documento).
Se não for o caso de que algo ficou claro o bastante só pela exposição aqui presente, for favor me dê um toque.

Para evitar ambiguidade será utilizada a convenção de "*The Art of Prolog" para diferenciação de fatos e goals:* fatos serão escritos como usual e goals com um ponto de interrogação:

```
p(a).       % É um fato
p(a)?       % É um goal
```

Predicados de inspeção de estrutura

São predicados importantes aos quais, creio eu, já demos uma rápida olhada antes. Será útil vê-los com um pouco mais de detalhe.

Entre esses predicados estão os predicados de tipo

Um tipo, em prolog, é algo como:

```
% X é uma árvore binária se
%     X = arvore_b(R, E, D) e (leia "X é unificável com ..")
%     e E e D são uma árvore binária

arvore_b(void).
arvore_b(arvore_b(R, E, D) :-
        arvore_b(E),
        arvore_b(D).
```

Esse tipo de tipo é importante e útil, mas não é a isso que me refiro, e sim aos tipos primitivos do prolog. Alguns predicados de tipo são:

integer/1, real/1, atom/1, compound/1

Eles podem ser interpretados como uma lista infinita de fatos:

integer(0). integer(1). ...

E são o que se espera: verdadeiros quando seu argumento é do tipo buscado, falsos caso contrário. Segue uma breve explicação dos outros dois predicados:

> *atom/1* : um termo é um átomo se ele é uma constante não numérica
> *compound/1* : um termo é composto se ele é composto (isto é, algo como *p(a, b), c(v, B), etc.)*

A partir desses predicados, podemos formar outros:

> *number(X) :- integer(X); real(X).* % real aqui significa ponto flutuante
>
> *constant(X) :- number(X); atom(X).*
> *% lembre que ";", lê-se, intuitivamente, como "ou"*

Obs$_1$.: Apesar de predicados de tipo serem importantes para a correção do programa, são geralmente omitidos por questões de eficiência, exceto nos predicados a seguir.
Obs$_2$.: *real/1* não está presente no swi prolog. No lugar, usa-se o *float/1.*

Acesso de termos compostos

Um aspecto de inspeção de estruturas é inspeção de tipos, lidada acima. Outro é o acesso de termos compostos.

Um dos predicados para acesso de termos compostos é o *functor/3:*

> *functor(Termo, F, Aridade)? tem sucesso se F é o funtor principal do termo Termo de aridade Aridade*
>
> *isto é, functor(f(x_1, ...,x_N), f, N)? tem sucesso.*

Esse predicado é usado principalmente para decomposição e criação de termos:

> *functor(tio(a,b), X, Y)? tem a solução {X = tio, Y = 2}*
> *functor(F, tio(a,b), 2)? tem a solução F = tio*

Outro predicado desse tipo é o arg/3:

> *arg(N, Termo, Arg)? tem sucesso se Arg é o N-ésimo argumento do termo Termo*
>
> *arg(1, tio(A, B), X)?, por exemplo, tem sucesso*

Obs.: Um termo constante é considerado, em Prolog padrão, um funtor de aridade 0.

Outro predicado para inspeção de estruturas é o, assim chamado, *univ, escrito como "=..". Termo =.. Lista?* tem sucesso se Lista é uma lista cujo primeiro elemento é o nome do funtor do termo Termo e cuja calda (o resto da lista) são os argumentos para Termo:

tio(a,b) =.. [tio, a, b]? tem sucesso

Univ tem dois usos: construir um termo dada uma lista, ou uma lista dado um termo.

Exemplos de programas com aplicações dos conceitos vistos acima estão em arquivo companheiro.

Obs.: Predicados para acesso e construção de termos têm origem na família Prolog de Edinburgo (que tem se tornado o padrão *de facto*). A forma "=.." para univ vem do Prolog-10, onde era usado ",.." no lugar de "|" em listas ([a, b,.. Xs] no lugar de [a, b|Xs]).

Predicados para meta-programação

Predicados de meta-programação nos permite superar duas dificuldades principais no use de variáveis:

1º) No uso de predicados tais como os predicados aritméticos, não é permitido o uso de variáveis (não instanciadas): Z is X + Y resulta em erro.
2º) Durante a inspeção de estrutura, variáveis podem ser instanciadas acidentalmente.

É esperado que essas dificuldades sejam melhor compreendidas com o restante do texto.

O predicado de meta-programação básico é o *var/1*. Seu comportamento é o intuitivamente esperado: *var(Termo)?* tem sucesso se Termo é uma variável e falha caso contrário:

var(X)? tem sucesso, var(a)? e var([X[Xs]) falham

O predicado *nonvar/1* é análogo ao *var/1* e tem o comportamento intuitivamente esperado.

Predicados meta-logicos (como os acima) podem ser usados para dar maior flexibilidade a programas. Exemplos estão no arquivo companheiro.

Convenções a serem usadas adiante

Algumas das convenções a serem usadas estão listadas abaixo para referência (algumas delas já estão em uso, como deve ter notado, e outras serão melhor explicitadas durante a leitura do programa, sendo, portanto, omitidas):

.Usamos *p/n* para expressar que o predicado p tem aridade n;
.Sempre que uma variável não for usada, será usado a variável anônima '_'. Em vários ambientes Prolog, uma mensagem é emitida quando isso não é feito. O objetivo é não tirar o foco da informação importante;

Os programas-exemplo até agora foram suficientemente simples para não exigir muita documentação ou muitos comentários no código. À medida que fizermos programas mais complicados/complexos, isso não será mais uma realidade.

O desenvolvimento de programas não tipados pode render uma prototipagem mais rápida e desenvolvimento mais interativo, mas, para projetos mais envolventes, a presença de tipos tem se mostrado importante, e devem ser especificados. Ademais, enquanto que, a princípio, um programa lógico tem vários usos, na prática, isso raramente é o caso. Os possíveis usos de um programa devem ser especificados.

Para indicar quais entradas, para um dado uso, de um predicado devem ser argumentos instanciados, quais não são, e quais se tornam, será utilizada a seguinte convenção:

.Quando um argumento é instanciado, usa-se +
.Quando um argumento não é instanciado, usa-se -;
.Quando tanto faz se um argumento é instanciado, usa-se ?;

Ex.:
p(+, +, -) → *p(+, +, +)* indica que, ao se chamar *p/3*, os dois primeiros argumentos são instanciados e o terceiro não e, no final, todos os três argumentos são instanciados

Uma especificação de programa, quando utilizada, deve seguir o seguinte esqueleto:

$p(T_1, \ldots, T_N)$

Tipos:

T_1: tipo

.

.

.

T_N: tipo

Esquema de relação:
Modos de uso:
Multiplicidades de solução.