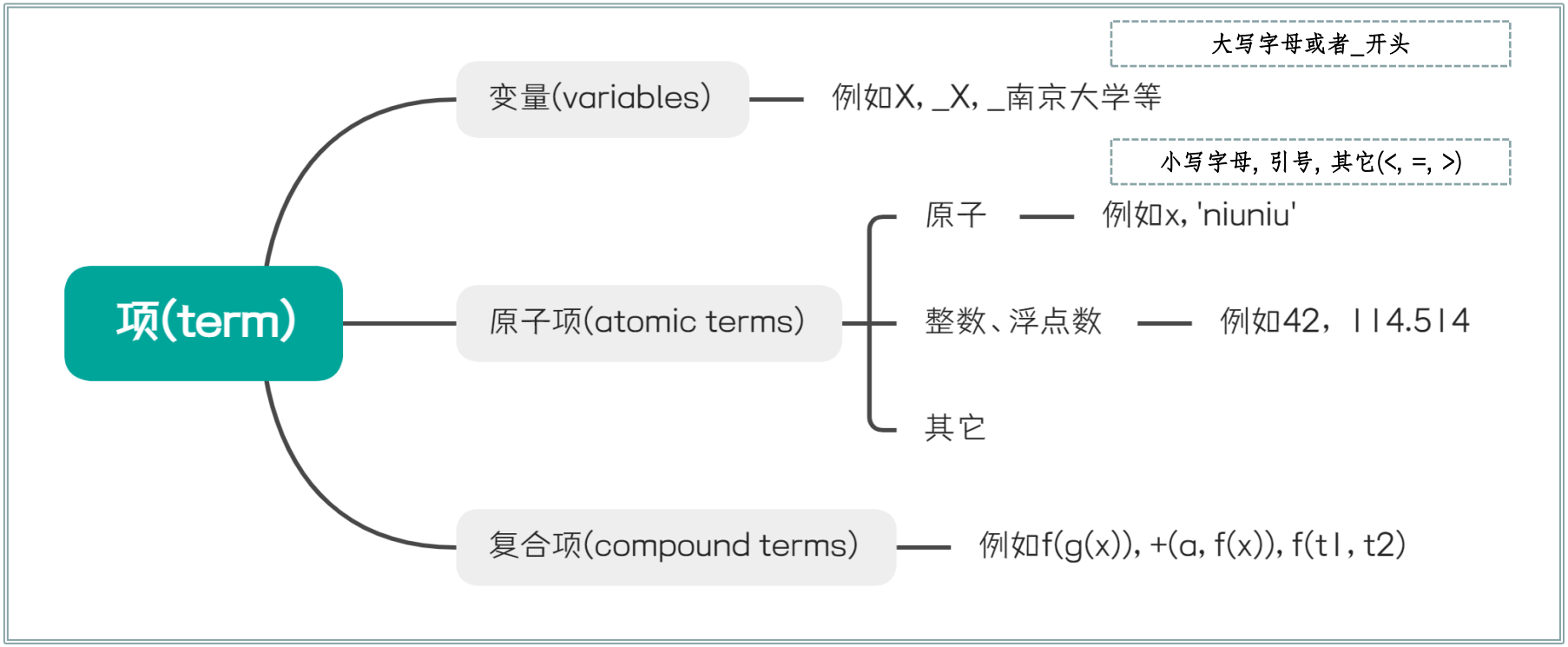
Prolog的数据结构




```
2 ?- 5 + 2 == 7
.
false.
```

复合项：如果$T_1, T_2, ..., T_n$ 是项，则F ($T_1, T_2, ..., T_n$)也是一个项，其中F称为含子名(functor name),遵循与原子相同的语法规则。F /N称为复合项的主函子

Prolog的数据结构

一些需要明确的东西

X	x	‘x’	“x”	[x]
变量	原子	原子	列表	列表

```
1 ?- X == x.
false.

2 ?- x == 'x'.
true.

3 ?- 'x' == "x".
false.

4 ?- "x" == [x].
true.
```

```
10 ?- _ == x.
false.

11 ?- ['南','京','大','学'] == "南京大学".
true.

12 ?- 南京大学 == '南京大学'.
true.
```

一些需要明确的东西

predicate（谓词）vs functor（函子）vs function（函数）

- predicate（谓词）
  - 由一组子句定义，其中子句是事实/规则
  - 任意子句为真，则整个谓词为真
- functor（函子）
  - 只是用来构造复合项的标签，但并不会执行什么操作
- function（函数）
  - Prolog 核心无专门概念，算术运算的“函数”由可求值函子（evaluable functor）＋ is/2 实现，其它“函数”只能伪装成返回值的谓词

Prolog的数据结构

- 基本项(Ground): 不含变量的项
- 部分实例化项(partially instantiated): 子项中包含变量的复合项

Prolog的动态性

1. 用原子zero代表0
2. 用复合项s(X)表示X的后继
3. 可以用s(s(s(zero)))表示数字3

Prolog的数据结构

列表

- 列表是项的特例
- 定义
  1. 原子[]是一个列表，表示空列表。
  2. 如果LS是一个列表，那么项'.'(L, LS)也是一个列表

- 简写
  1. 列表'.'(a, '.'(b, '.'(c, [])))也可以写为[a,b,c]。
  2. 项'.'(L,LS)也可以写成[L|LS]
  3. 这些符号可以以任何方式组合。例如，项[a,b|LS]是列表当且仅当LS是列表

```
4 ?- .(a, .(b, [])) == [a, b].
ERROR: Type error: `dict' expected, found `b' (an atom)ERROR: In:
ERROR:   [15] throw(error(type_error(dict,b),_1578))
ERROR:   [12] '<meta-call>'(user:user: ...) <foreign>
ERROR:   [11] toplevel_call(user:user: ...) at f:/program files/swipl/boot/toplevel.pl:1317
ERROR:
ERROR: Note: some frames are missing due to last-call optimization.
ERROR: Re-run your program in debug mode (:- debug.) to get more detail.
```

Prolog的数据结构

思考

1. [a,b|[]] == [a, b].
2. [a|b] == [a, b].
3. [a|[b]] == [a, b].

```
5 ?- [a,b|[]] == [a, b].
true.

6 ?- [a|b] == [a, b].
false.

7 ?- [a|[b]]== [a, b].
true.
```

Prolog的数据结构

什么时候适合使用列表？

1. 任意多个元素
2. 可以有0元素
3. 元素顺序重要
4. 同一类元素

如果元素数目为定值，用什么比较好？

谓词！
因为谓词推理能力更强，并且更节省内存空间！

Prolog的数据结构

对子(pairs)

- 定义：对是具有主函子(-)/2 的项。例如，项-(A, B)表示元素A和B的对。在 Prolog 中，(-)/2定义为中缀运算符。因此，该项可以等效地写为 A-B

常见用途

- 排序！，可以使用标准谓词keysort进行实现！

```
11 ?- keysort([3-x, 1-y, 2-z], Sorted).
Sorted = [1-y, 2-z, 3-x].
```

Prolog的数据结构

关联列表(Association lists)

- 定义方式
  - 空树用原子t表示
  - 内部节点用复合项t(Key, Value, Balance, Left, Right)表示，其中：
    - Key和Value表示键值关联
    - Balance是表示平衡条件的原子，如<, =, >表示左右子树节点数关系
    - Left和Right是AVL子树

- 功能
  - 许多 Prolog 系统提供关联列表，支持比线性时间更快的元素访问，通常基于平衡树（如 AVL 树）实现。

Prolog的数据结构

Swilog使用关联列表的示例

代码

输出

```
1 ?- demo_assoc.
After inserts: t(banana,5,-,t(apple,3,-,t,t),t(orange,2,-,t,t))
banana = 5
deleted apple, value was 3
current pairs: [banana-5,orange-2]
built from list: t(b,2,-,t(a,1,-,t,t),t(c,3,-,t,t))
merged back: t(banana,5,>,t,t(orange,2,-,t,t))
true.
current pairs: [banana-5,orange-2]
built from list: t(b,2,-,t(a,1,-,t,t),t(c,3,-,t,t))
merged back: t(banana,5,>,t,t(orange,2,-,t,t))
true.
built from list: t(b,2,-,t(a,1,-,t,t),t(c,3,-,t,t))
merged back: t(banana,5,>,t,t(orange,2,-,t,t))
true.
true.
```

```
% demo_assoc.pl
:- use_module(library(assoc)).

% demo_assoc/0 演示常用操作
demo_assoc :-
    % 1. 创建一个空的 AVL 树
    empty_assoc(T0),

    % 2. 插入/更新键-值对
    %    put_assoc(+Key, +OldTree, +Value, -NewTree)
    put_assoc(apple,  T0, 3,  T1),
    put_assoc(banana, T1, 5,  T2),
    put_assoc(orange, T2, 2,  T3),
    format('After inserts: ~w~n', [T3]),

    % 3. 查找
    %    get_assoc(+Key, +Tree, -Value)
    ( get_assoc(banana, T3, BV) ->
        format('banana = ~w~n', [BV])
    ; format('banana not found~n') ),

    % 4. 删除
    %    del_assoc(+Key, +TreeIn, -Value, -TreeOut)
    ( del_assoc(apple, T3, AV, T4) ->
        format('deleted apple, value was ~w~n', [AV])
    ; format('apple not found~n') ),

    % 5. 转回键-值列表（按 Key 升序）
    %    assoc_to_list(+Tree, -Pairs)
    assoc_to_list(T4, Pairs),
    format('current pairs: ~w~n', [Pairs]),

    % 6. 从列表快速构造
    %    list_to_assoc/2 要求列表 Key 升序或者使用 ord_list_to_assoc/2
    list_to_assoc([a-1, b-2, c-3], A5),
    format('built from list: ~w~n', [A5]),

    % 7. 合并：可以把一个 assoc 拆成 list，再 foldl put_assoc
    foldl(
      {T4}/[K-V, Tin, Tout]>>put_assoc(K, Tin, V, Tout),
      Pairs,        % 从上面得到的 Pairs
      T4,           % 初始树
      MergedTree
    ),
    format('merged back: ~w~n', [MergedTree]).
```

Prolog的数据结构

Prolog没有哪些东西？

没有字符串！ 但是有字符列表！

```
3 ?- "hello world!" == [h,e,l,l,o,' ',w,o,r,l,d,!].
true.

3 ?- 'hello world!' == [h,e,l,l,o,' ',w,o,r,l,d,!].
false.
```

1. Richard O'Keefe 在《Prolog 库提案》中已有精辟论述：字符串是错误的选择，几乎所有涉及字符串处理的场景，最合理的做法是尽早将字符串转换为树结构（Prolog 擅长处理树）。
2. 在 Prolog 中，字符串的自然表示是字符列表（单字符原子的列表）。若将 Prolog 标志double_quotes设为chars，双引号字符串会自动解释为字符列表。

Prolog的数据结构

Prolog没有哪些东西？

没有数组！ 但是有列表！

逻辑编程不支持破坏性修改。在 Prolog 中，数据结构的变化通过谓词描述前后状态的关系，因此纯修改通常需要复制数据，至少带来对数级开销（如复制平衡树的子树）

思考：数组和列表到底有什么区别？

- 访问复杂度，数组O(1), 列表O(k)
- 存储结构，数组连续，列表不一定
- 长度动态性，列表优于数组（一般情况下）

Prolog的数据结构

类型测试

- 标准测试类型谓词
  - atom/1、integrate/1、compound/1等
  - 如果一个谓词对于某个更“通用”（less instantiated）的参数失败，而对更“具体”（more instantiated）的参数成功，就破坏了单调性。
- 新型测试类型谓词
  - ..._si/1、 must_be/2、can_be/2等
  - 实现了具有所需逻辑属性的类型测试

```
5 ?- atom(X).
false.

6 ?- X = a, atom(X).
X = a.
```

```
1 ?- atom_si(X).
ERROR: Unknown procedure: atom_si/1 (DWIM could not correct goal)
2 ?- can_be(atom, X).
ERROR: Unknown procedure: can_be/2 (DWIM could not correct goal)
^  Exception: (4) setup_call_cleanup('$toplevel':notrace(call_repl_loop_hook(begin, 0)), '$toplevel':'$query_loop'(0), '$toplevel':notrace(call_repl_loop_hook(end, 0))) ?
creep
3 ?- must_be(atom, X).
ERROR: Arguments are not sufficiently instantiated
ERROR: In:
ERROR:   [15] throw(error(instantiation_error,_162))
ERROR:   [11] toplevel_call(user:user: ...) at f:/program files/swipl/boot/toplevel.pl:1317
ERROR:
ERROR: Note: some frames are missing due to last-call optimization.
ERROR: Re-run your program in debug mode (:- debug.) to get more detail.
^  Exception: (4) setup_call_cleanup('$toplevel':notrace(call_repl_loop_hook(begin, 0)), '$toplevel':'$query_loop'(0), '$toplevel':notrace(call_repl_loop_hook(end, 0))) ?
creep
4 ?- X = x, must_be(atom, X).
X = x.
```

Prolog的数据结构

清晰的表述（clean representation）

- 好的Prolog代码：可以区分每个成分的种类与主函子
- 坏的Prolog代码：无法通过主函子区分成分的种类

例子

任务：在prolog中表示一个满二叉树

方案一：leaf(L)表示叶子L， node(Left, Right)示一个节点及其两个子节点

方案二：只使用node(_, _)

假设想要表达一个只有1个根节点，2个叶子节点的二叉树，那么node(leaf(_), leaf(_))足够精确，而node(_, _)不再精确，

Prolog的阅读

声明式阅读(Declarative reading)

- Prolog 程序以声明的方式陈述了什么成立。Prolog 程序由子句组成，每个子句要么是事实，要么是规则。事实陈述了什么 始终为真。规则陈述了在特定条件下什么为真。
- 这种阅读 Prolog 程序的方式也称为 推导式阅读（concluding reading）。
- 优点：描述清晰、易于理解和使用——你只需陈述在何种条件下什么成立，Prolog 引擎会自动为你搜索解。
- 缺点：无法解释为何逻辑等价的程序变体在性能或可终止性方面会有差异。

示例

```
list_list_together([], Bs, Bs).
list_list_together([A|As], Bs, [A|Cs]) :-
        list_list_together(As, Bs, Cs).
```

1. 空列表 [] 与任意列表 Bs 的拼接就是 Bs 本身。
2. 如果列表 As 与 Bs 的拼接是 Cs，那么对于任意元素 A，列表 [A|As] 与 Bs 的拼接就是 [A|Cs]

Prolog的阅读

过程式阅读（Procedural reading）

----考虑 Prolog 引擎的实际计算策略。

调用Prolog谓词和其他语言的区别

1. Prolog 变量是真正的变量，可能未绑定或只部分实例化——大多数其他语言的变量不允许这样。
2. Prolog 内置回溯（backtracking）机制，会穷举地尝试所有备选分支。

- 优点：能够解释不同程序变体之间的性能差异与终止性质。
- 缺点：从过程角度理解 Prolog 程序要比理解其他语言的控制流困难得多，需要考虑变量的实例化状态、变量之间的别名（aliasing）、回溯时出现的可选分支。

示例

```
list_list_together([], Bs, Bs).
list_list_together([A|As], Bs, [A|Cs]) :-
        list_list_together(As, Bs, Cs).

?- list_list_together([x,y], [z], Cs).
```

Prolog的阅读

程序切片（Program Slicing）

程序切片是一种简单而强大的技术，它利用纯 Prolog 的一些非常通用的性质来研究对程序进行泛化（generalization）和特化（specialization）时所产生的影响。

性质

- 删除某个目标（goal）最多只能使程序变得更通用（more general），绝不会变得更具体（more specific）；
- 删除某个子句（clause）最多只能使程序变得更具体，绝不会变得更通用；
- 在规则中任意两目标之间插入 false/0，可以让我们忽略该点之后所有目标的过程效应。

示例

```
list_length([_|Ls], N) :-
    list_length(Ls, N0),
    N #> 0,
    N #= N0 + 1.
list_length([], 0).
```

- 当列表已充分实例化时，该谓词表现正常
- 但对于最通用的查询，它不会产生任何解，也不会终止。

Prolog的全局变量？

没错！Prolog没有全局变量！，一般使用谓词进行信息的传递。

原因

设想如果存在一个可被谓词修改的全局状态，它将与我们对“关系”（relation）应有的预期产生根本冲突。例如，当以关系来编程时，我们期望：

- 能够反向使用同一个关系；
- 能够单独推理某个关系，支持声明式测试；
- 同一个关系被多次调用时，其含义完全一致；
- 回溯时所有改动都能自动撤销；
- 在并行计算中天然线程安全；

当然，Prolog 也提供了修改全局数据库的手段，例如 assertz/1 可以动态地断言新子句。但这种方式不被提倡，视为额外手段

Prolog的全局变量？

示例

累加器的C++代码

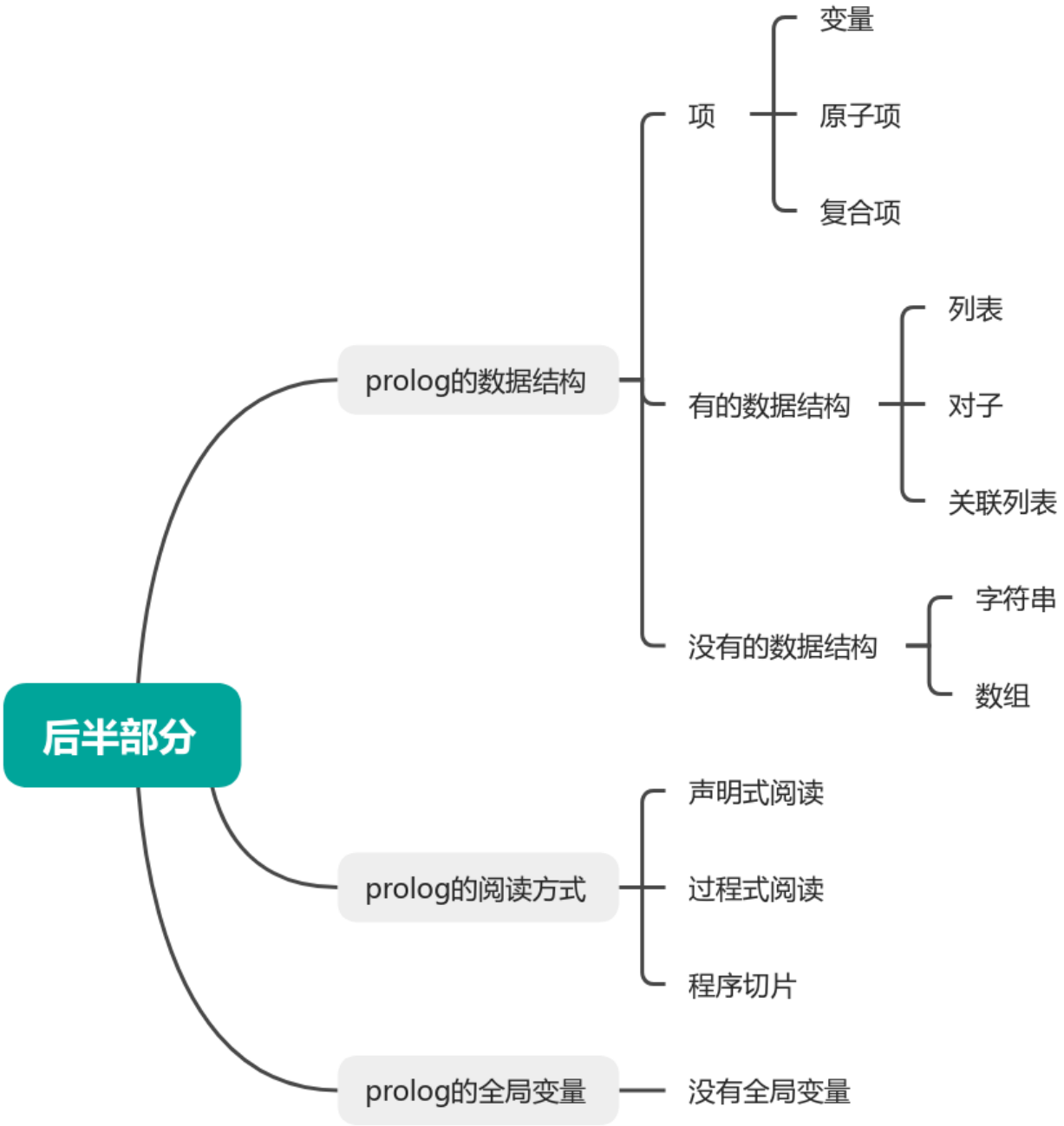
```
int sum = 0;
for (i = 0; i < n; i++) sum += a[i];
```

```
% 对外接口，只需两个参数
sum_list(List, Sum) :-
    sum_list(List, 0, Sum).

% 递归终止
sum_list([], Acc, Acc).

% 递归推进：Acc0→Acc1
sum_list([X|Xs], Acc0, Sum) :-
    Acc1 is Acc0 + X,
    sum_list(Xs, Acc1, Sum).
```

总结


后半部分
prolog的数据结构
项
变量
原子项
复合项
有的数据结构
列表
对子
关联列表
没有的数据结构
字符串
数组
prolog的阅读方式
声明式阅读
过程式阅读
程序切片
prolog的全局变量
没有全局变量

感谢聆听