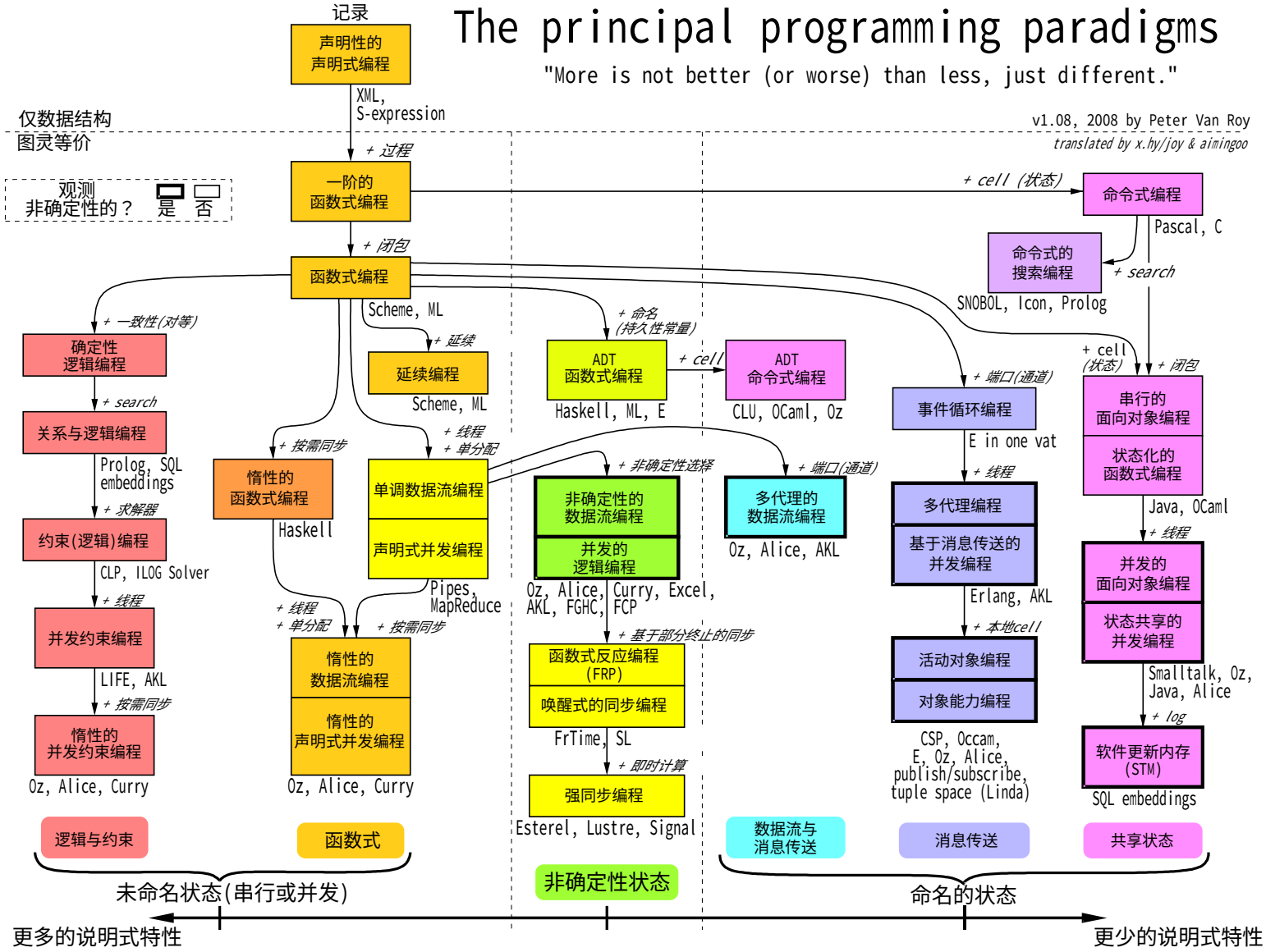
The principal programming paradigms

"More is not better (or worse) than less, just different."



说明(Explanations)

参见《Concepts, Techniques, and Models of Computer Programming》

本图表根据编程范型的核心语言（在所有该范型的抽象中都能被定义的语言核心）进行了分类。核心语言依据其被创建的基本原理进行排列：当仅仅通过局部转换，无法将一个新的概念编码实现时，它就会被添加进来。对程序员来说，实现了相同范型的两种语言之间，也可能具有极大差异的“风味”，因为它们在编程技术与风格上做出了不同的选择。

当一门语言在某个范型中被提及，它的意思是语言的某部分（被它的设计者）实现来支持该范型，以避免来自于其它范型的干扰。这并不是说该语言与范型非常适配。仅仅在语言中实现一个支持该范型的库是远远不够的。这个语言的核心（语言部分）应该支持这种范型。（在本图表中，）相关语言家族中通常只有一种被提及，以避免不必要的混乱。缺少某种语言，并不暗示对它的价值的评判。

状态是记忆信息的一种能力，更精确地说，是及时存贮值序列的能力。它在包含它的范型的影响下体现出强烈的表现力。我们对此划分出四种表现等级，它们的不同在于：状态是未命名或命名的、确定的或非确定的，以及串行的或并发的。(在“状态”方面)表现最弱的是函数式编程范型（线程状态，例如DCGs和monads：未命名的，确定的和串行的）。然后，增加并发性，则得到声明式并发编程范型（例如：synchrocells：未命名的，确定的和并发的）。如果增加非确定性选择，则得到并发的非确定性数据流编程范型（使用流合并：未命名的，非确定的和并发的）。再则，分别增加端口和cells，就各自产生了消息传递或者共享状态范型（两种都是命名的,非确定的和并发的）。非确定性对真实世界交互很重要（例如在“客户/服务器”编程模型中）。命名状态，则对模块化有着相当重要的意义。

该图是在类型、方向和领域特性上正交的。类型并非完全正交：在表现上有一些影响。方向则可能是完全正交的，因为它们是程序规格的一部分。一种特定领域语言(DSL)则可以在任何范型中定义（除非该领域需要一个特定的——语言范型的——概念）。

元程序设计是提高语言表达能力的又一种方式。该术语还涉及许多不同的途径，从高阶程序设计、句法扩展（例如：宏），到结合句法支持的高阶编程（例如：meta.object协议与泛型），再到对核心语言的完全、胡乱地修补（自省和反射）。语法扩充和核心语言修补在本图中尤其正交。一些语言，例如Scheme，在实现方言时过于灵活。这种灵活性在图表中未有展示。