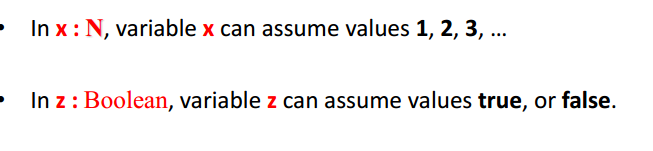
Z specification languag- Prelude

我们怎么声明变量的type，

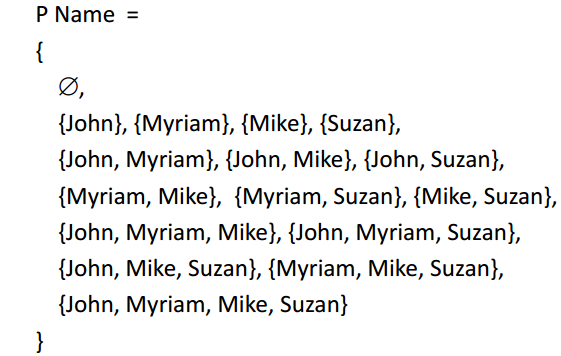
变量V可以被指定成任意值，只要这个值是他的type



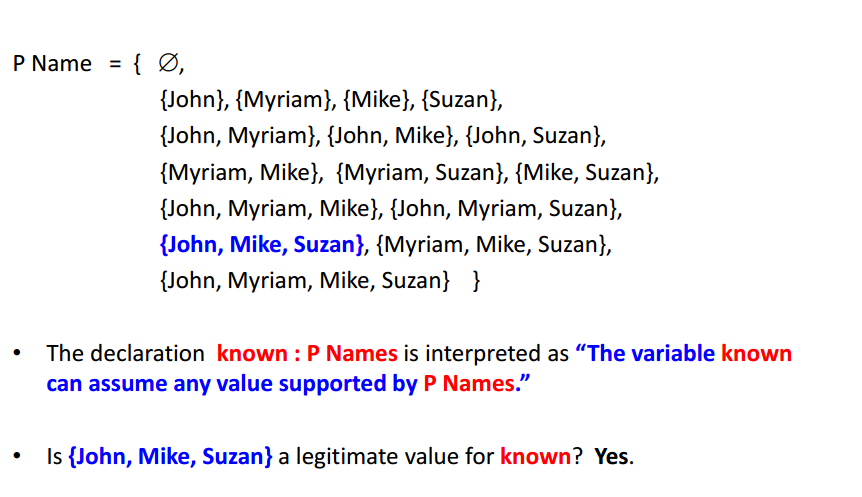
power set



power set就是由name元素组成的所有可能set的set

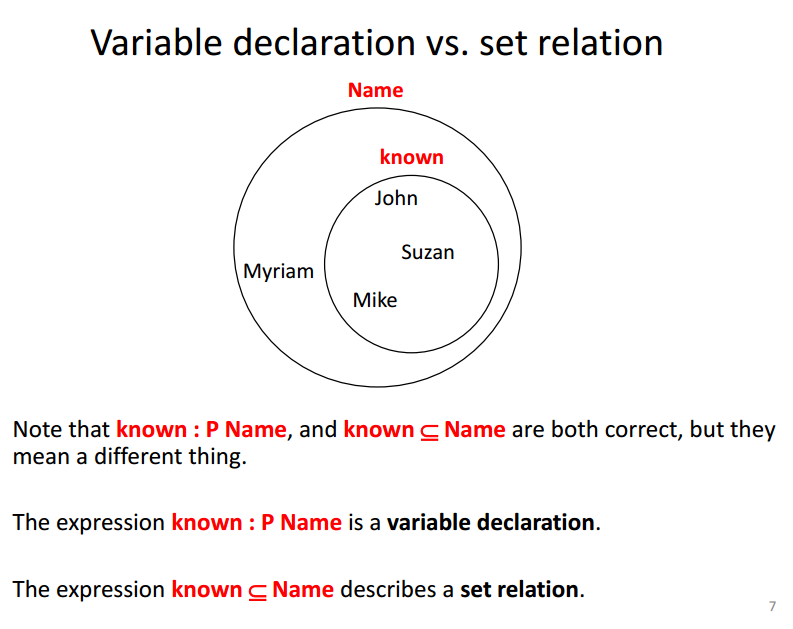


关于变量的合法值的推理



假如我们声明了一个变量known，类型是P NAMES，(这个Known的值可以是P NAME中任意一个元素)

那么NAME中的一个值JOHN,MIKE,SUZAN就是可以被支持的合理值



注意known: P name与known∈name的区别：前者是声明了一个变量，known，他可以的值是NAME

P NAME: name的power set

后者是set之间的关系

Z specification language

Z 是一个specification language 规范语言，基于predicate logic 与set theory

Z中的所有formal specification 标准规范,目的都是描述这个系统干什么，而不是怎么干

formal specification都是独立实现的implementation independent

基本的构建过程

1,Declaration:声明，介绍这个变量

2.expressions:表示，描述了variable可以等于的值

3.predicate:描述，给这些变量可以等于的值加上一个限制constraint

Z一共有两种type：simple type与composite type

simple type一共有两种，

1.primitivve type：Z里已经有的type

2.Basic type:用户定义的type

例如：银行里面，客户被定义成basic type：users

许多basic type可以在一行里面被定义[users,staff,clients]

composite type:

composite type利用type constructor 在simple type上构建

1.Power set type:

2.Cartesian product types:笛卡尔积type



这样的一个type的变量是一个tuple元祖

3，schema type式类型

abbreviation缩写

1.所写的方式就是X==Y，Y被缩写成X



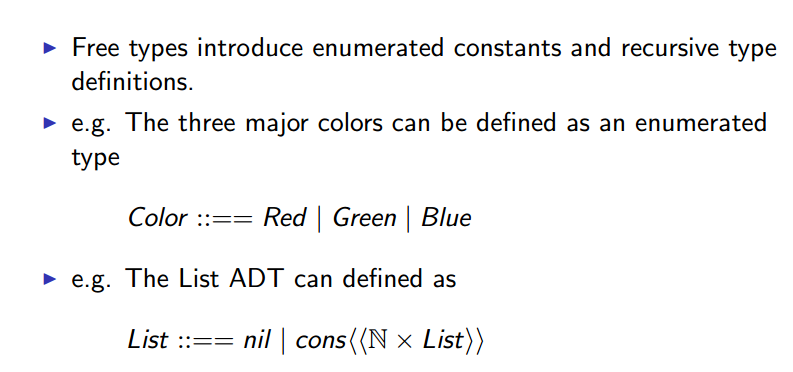
2.。

，缩写的时候定义了一种新type，room number可以的值是右边type的子集之一//注意左边可以并不完全等于右边，可以等于右边的子集

3.我们可以明确指定这个newtype是哪部分子集，利用点



free type，：：==

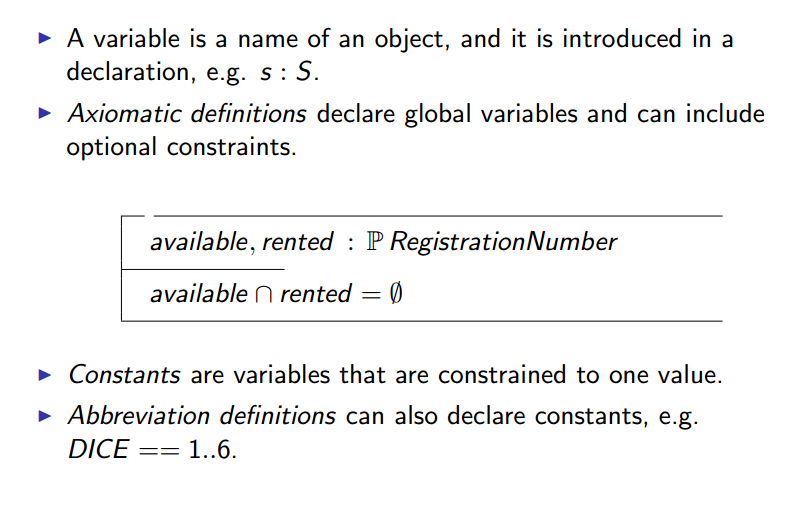


free type, 自由类型引入枚举常量或递归类型

第一个就时枚举enum

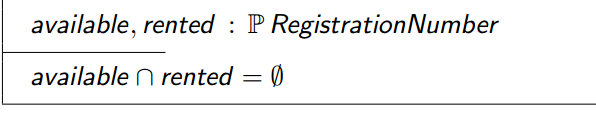
第二个就是递归类型，要么是nil，要么是不停的在原list的基础上加上N

Variable



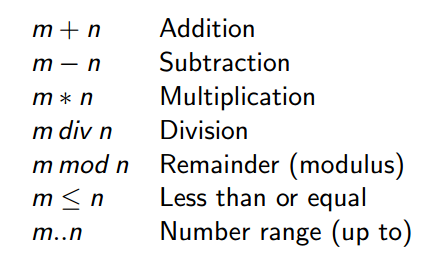
一个variable就是object的名字，在declaration部分中写出来

axiomatic definition，公理化定义描述了一个global variable,并且可以加限制（也可以不加）



第一行，declaration,定义来了两类object,available,rented，他们都是 P registrationNumber的一部分

没有交集：限制

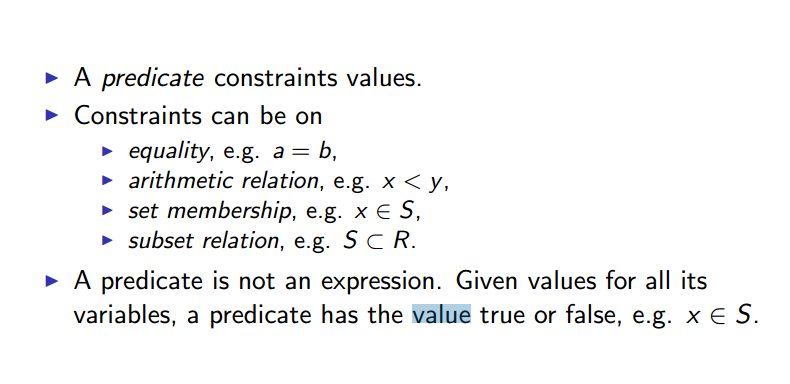


Z里面除法是div，区间是..,包括左右两边

predicate 谓语

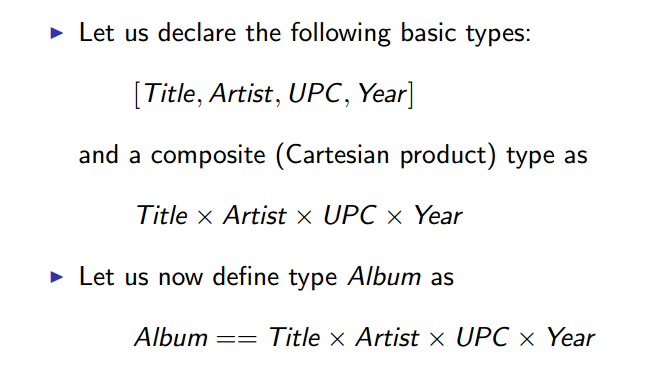
一个谓语给值加上了限制

限制可以有，等于，数学关系（小于大于），属于，子集关系的属于，predicate不是表达，而是一种判断，



声明type

声明简单type的方法，中括号一括，复合的就是乘起来，然后缩写

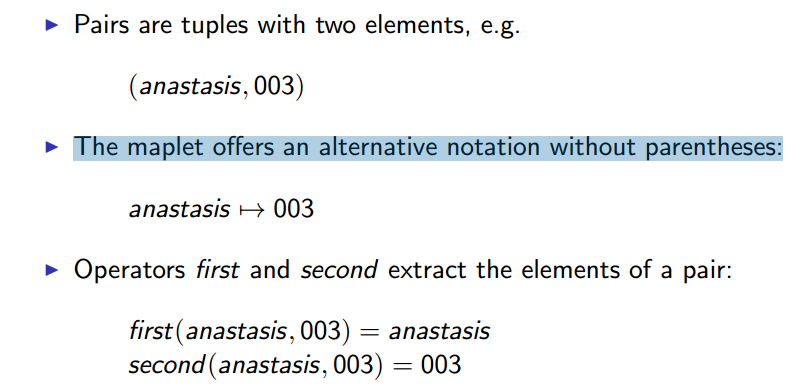
 ，

pair

pair就是有两个元素的tuple向量

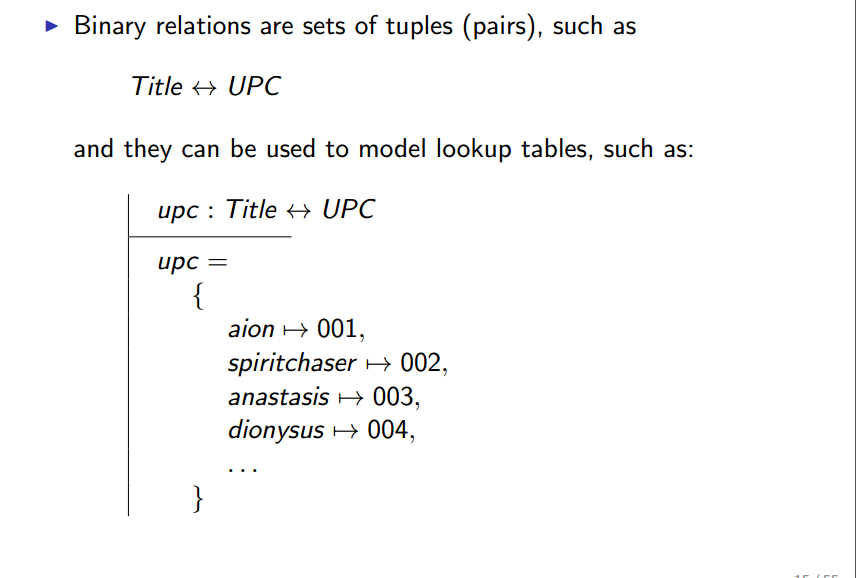
maplet提供了一种不用括号的替代符号

operator first与second会提取对应元素



Binary relation

是由pair组成的set



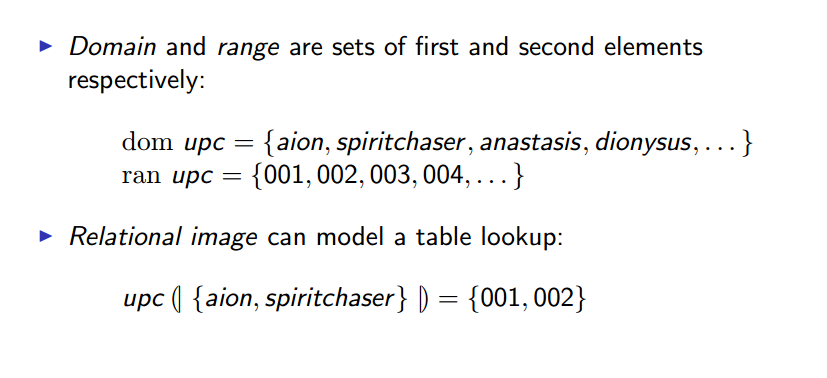
上面一行是declaration，下面的是限制（也是解释）

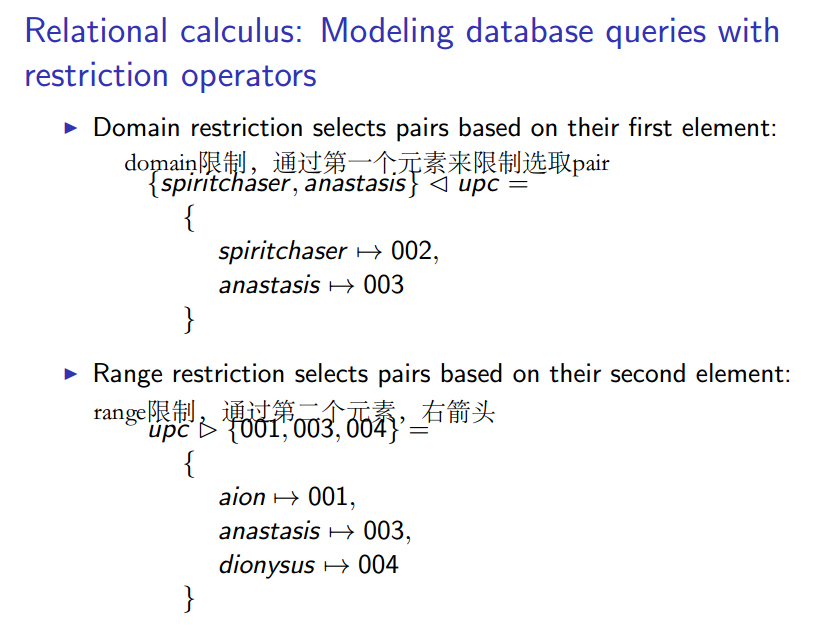
relational calculus关系演算,binary relation中的定义域与范围

dom，定义域，左边

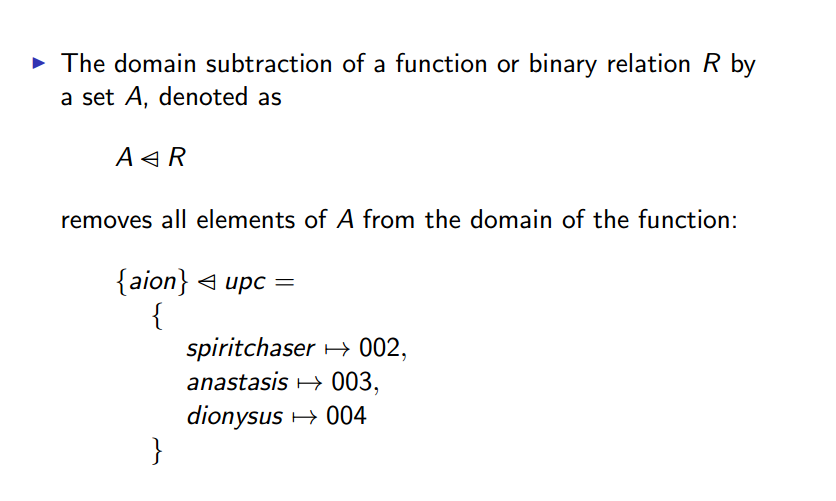
ran,右边

relational image:可以模仿上图的table lookup

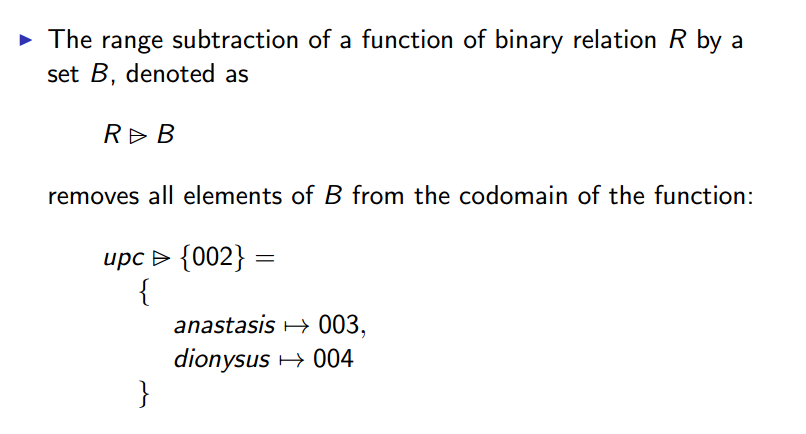




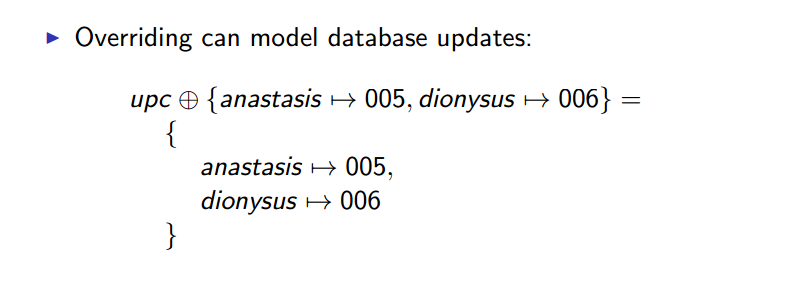
关系演算，域减法·



就是一个binary relation R(被减的集合)，除掉A中的元素，左箭头减去第一个，右箭头减去第二个

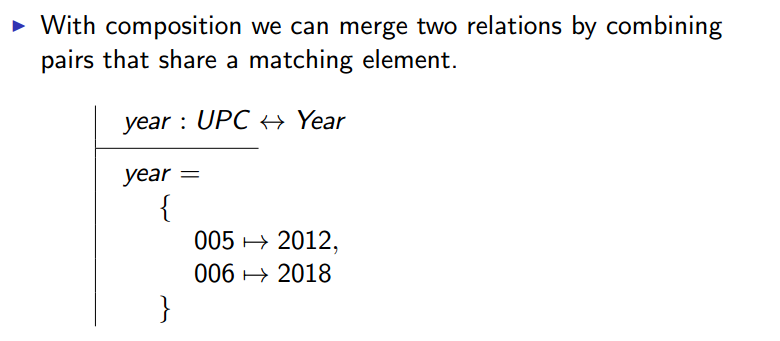


override，

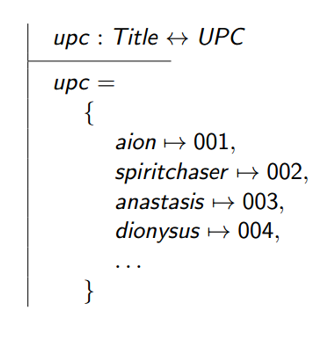


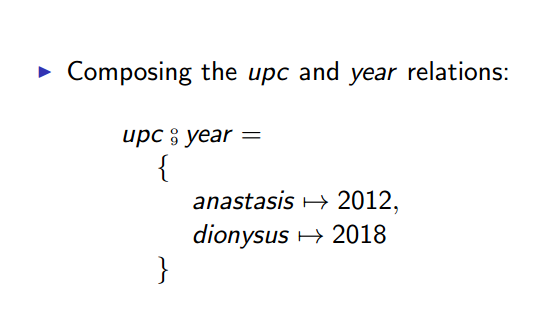
用异或符号，upc整个都被改写，而不仅是anastasis dionysus两个

使用composition来merge



我们看可以把两个关系合并起来，冒号就是最基本的表示法，注意这里UPC是大写，只是单纯的编号

和这个是一样的，composition就是创造一个新的元素类型，



而compose是新符号，用竖着写的09，他们会找到共用的元素

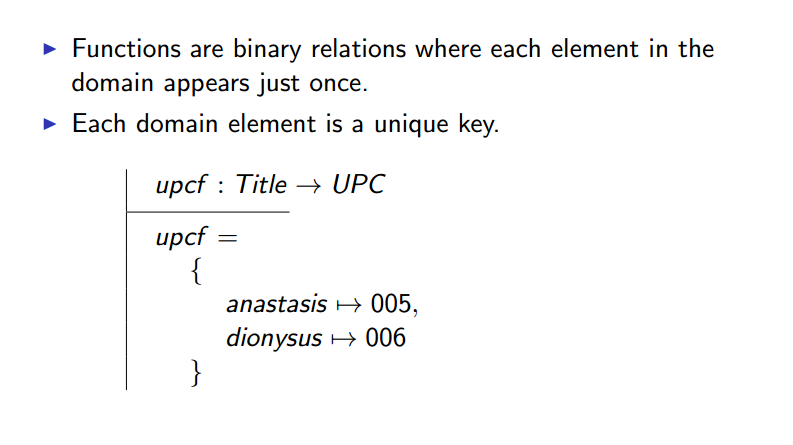
upc是 title<->upc

而year是Upc<->year

共用元素就是upc

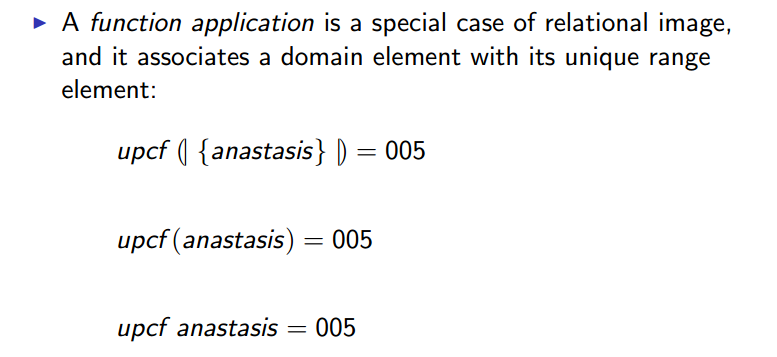
舍去

然后就变成新的composition



Function,也是binary relation，然而每个domain的元素是独立且不同的

function是relational image的一种特殊情况，也就是说左边的domain element只能出现一次，只能指向一个值



Relational calculus:Sequence

枚举本身虽然能定义一种type，但是没有顺序

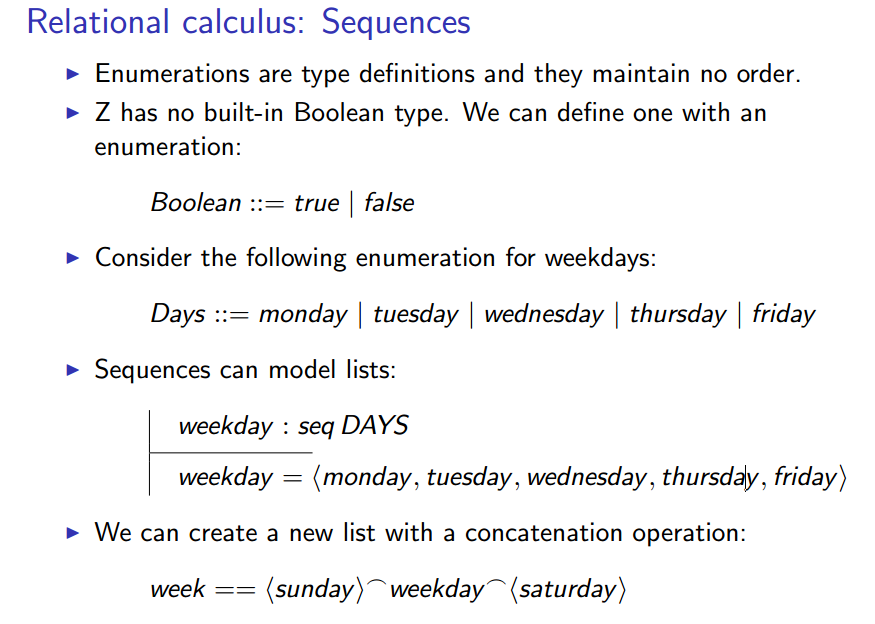
例如boolean（Z里没有boolean）

或者weekdays

但是他们是没有顺序的，

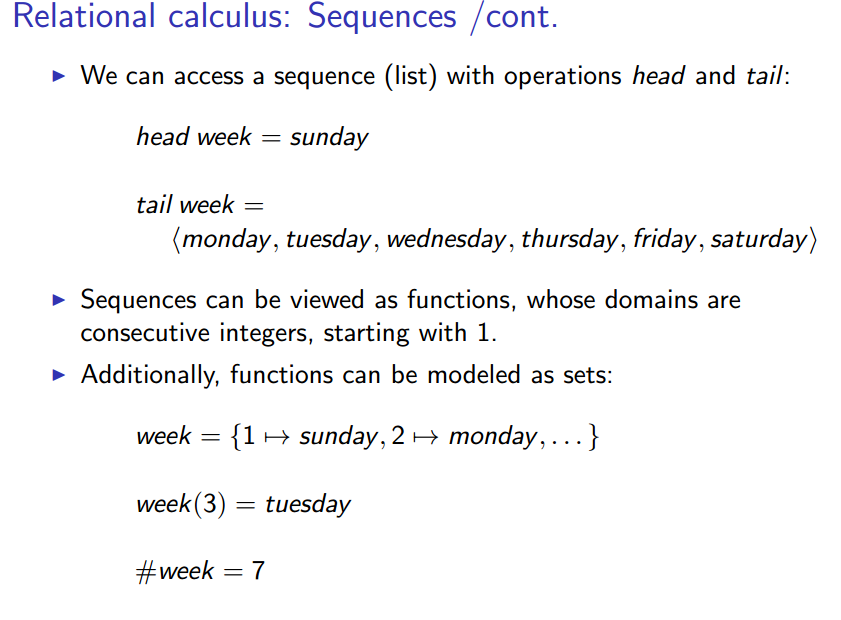
而sequence却可以构建一个list，符号是seq

我们也可以用连接符来构建新List(list是有顺序的)



进入sequence(list)方法就是head tail

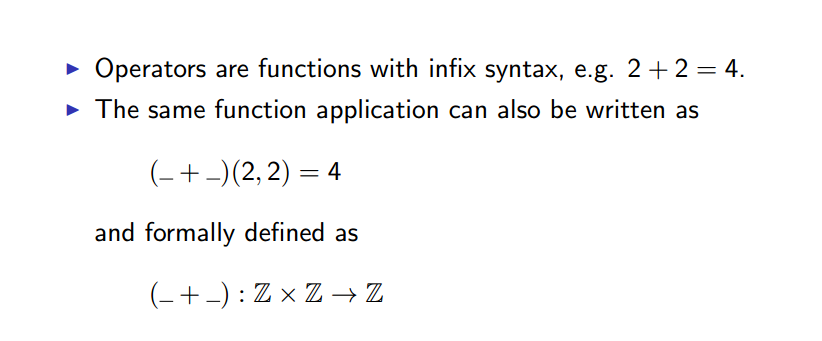
sequence也可以看作是function，domain是连续自动生成的正整数（可以看做php的array）



井号代表这个array里总共的数量

operator:加减符号一般都写在中间，infix syntax

但也可以写在前面作为一个function



例子

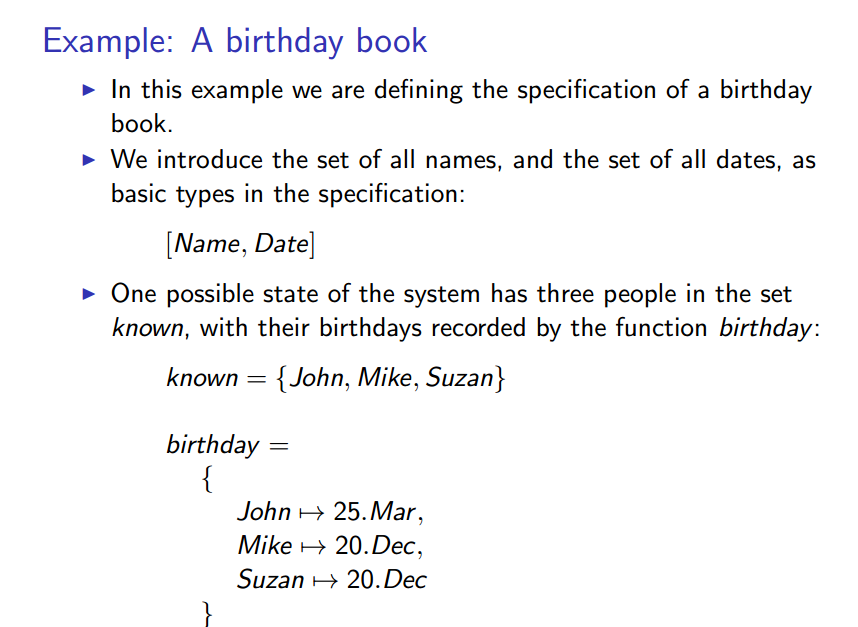
我们给birthday book描述一个规范

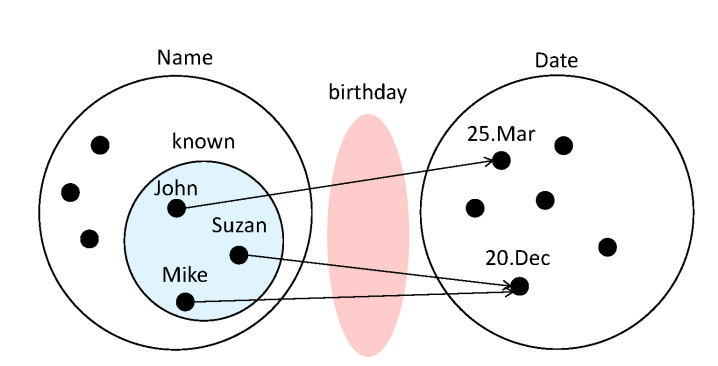
第一步

首先我们declare所有的基本type， [NAME,DATE]

然后这个系统可能的一个状态就是known，我们已知他们的生日

birthday是一个function





state schema状态模式

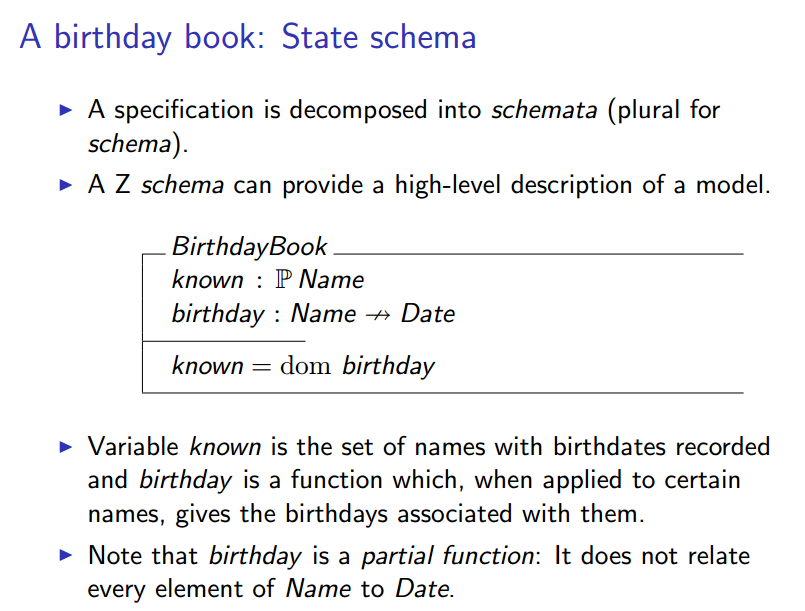
一个specification被分解成schemata,(多个schema)

一个Z schema可以高度描述一个model

变量known 是一组已经有生日的名字

birthday是一个function，当你给他一个具体名字，就会有对应的生日

注意birthday是一个partial function，并不是每个日期都有对应的人



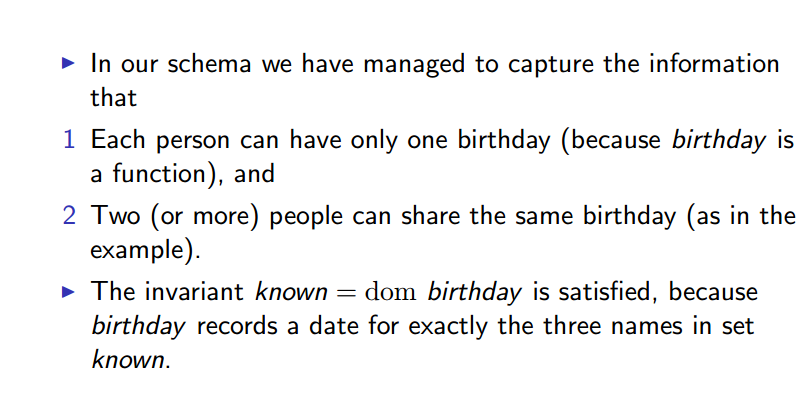
在我们的schema中，我们可以得到以下信息

1.每个人只有一个生日·，因为birthday是function

2.多个人可以share一个birthday

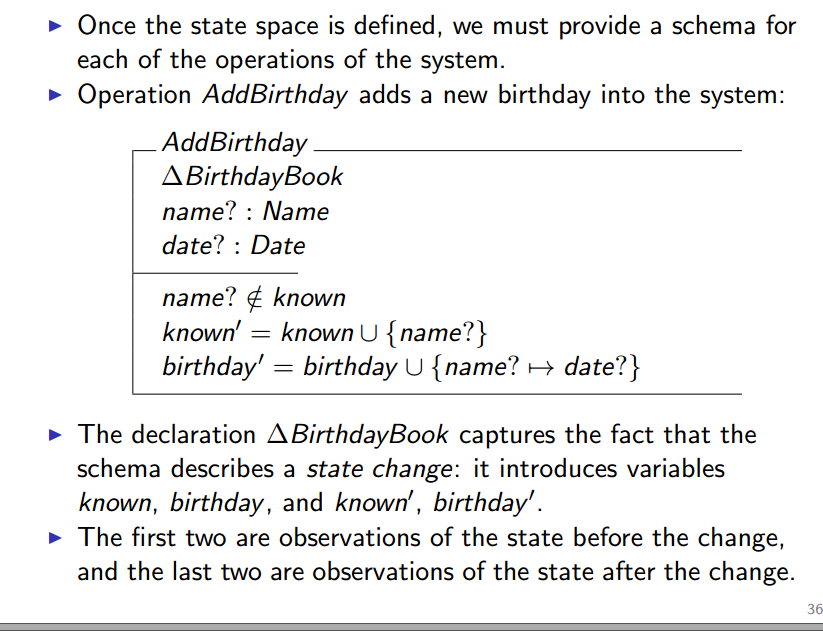
3，满足等式，Known是已知生日的name，birthday是name->date

那么两个相等就能找到对应的人名

.

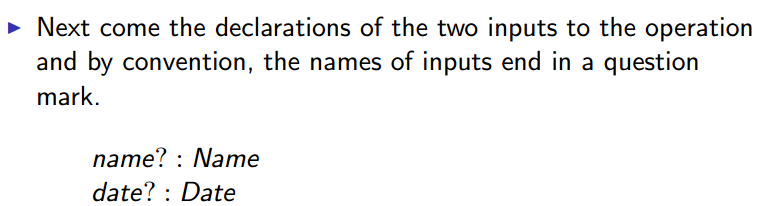
当我们声明了一个state，我们会用一个操作schema来描述这个system的不同动作

add birthday

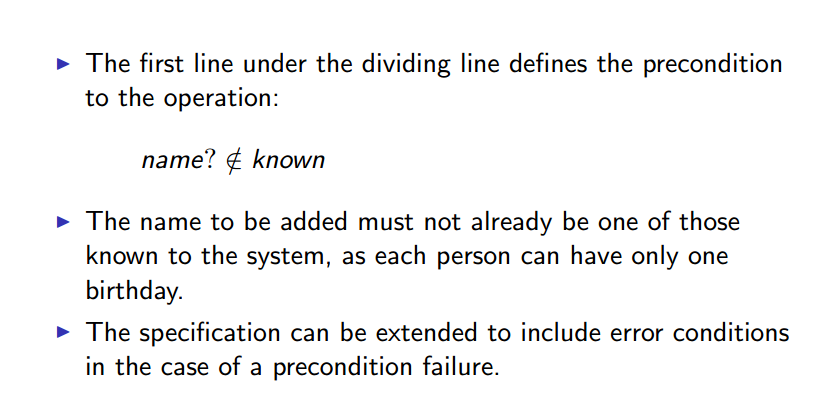


会catch到他所要改变的fact(state schema),意思是我要加上操作了，

前两行问号是我们要输入的变量（input）,命名为name，date，后门的大写是变量类型限制

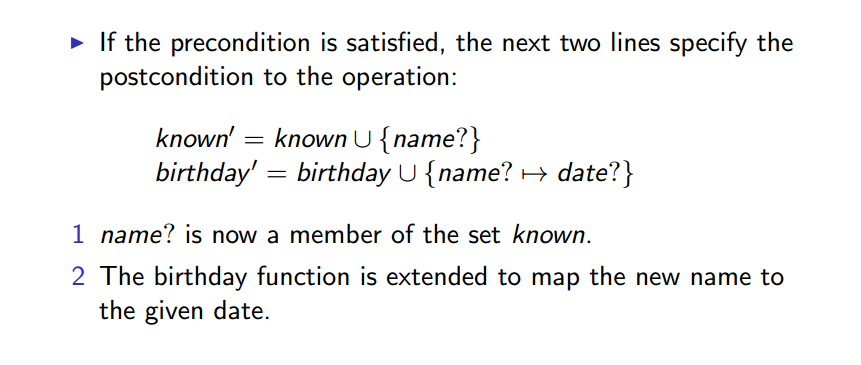


是先决条件，所加的名字必须是未知的，因为每个人只能有一个生日

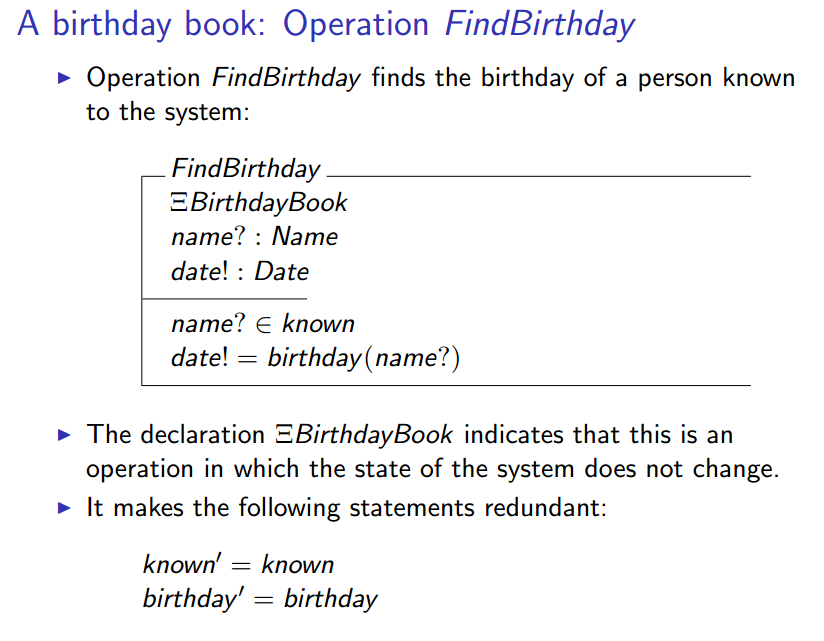


如果满足了先决条件，后面两行就告诉你了这个操作的后果

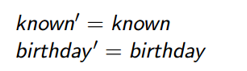
后面两行是要改变的新变量known与birthday,known加上新的东西，birthday加上新的东西(这个function现在可以map新name到given date)

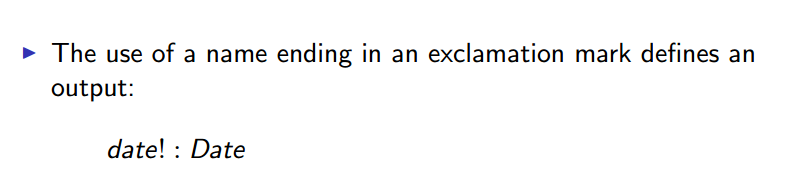


Find birthday



这个符号是代表着这个操作本身不会改变system的state

他对变量做出多余操作

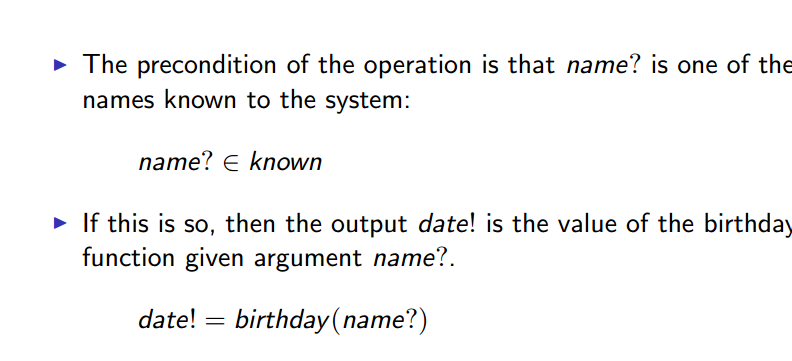


感叹号代表一个输出

先决条件，name?这个变量是不是known中的一部分

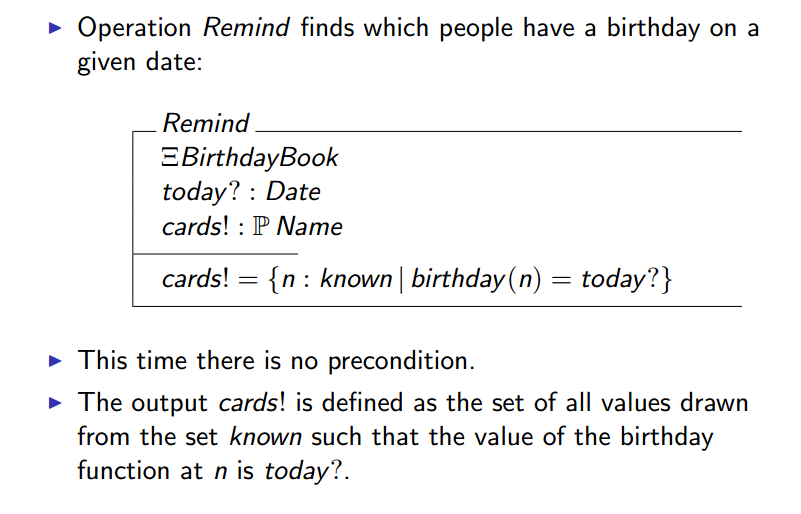


如果是，那么就让输出date!的值等于所输入name对应的birthday



\

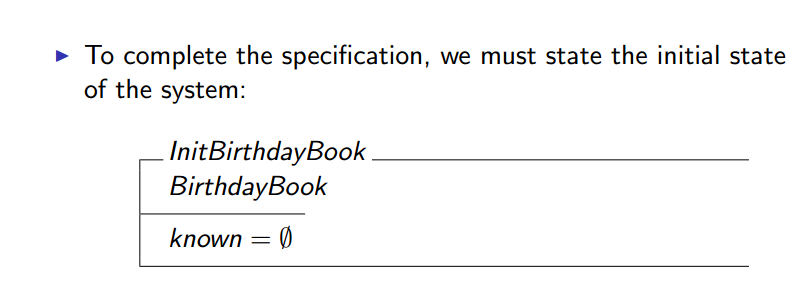
remind，



没有先决条件，

输出card! ，等于n，n是一个集合，代表着所有birthday(n)=today?所对应的人的名字的集合，因此类型是P NAME

为了完成这个z specification,我们需要给他一个初始状态

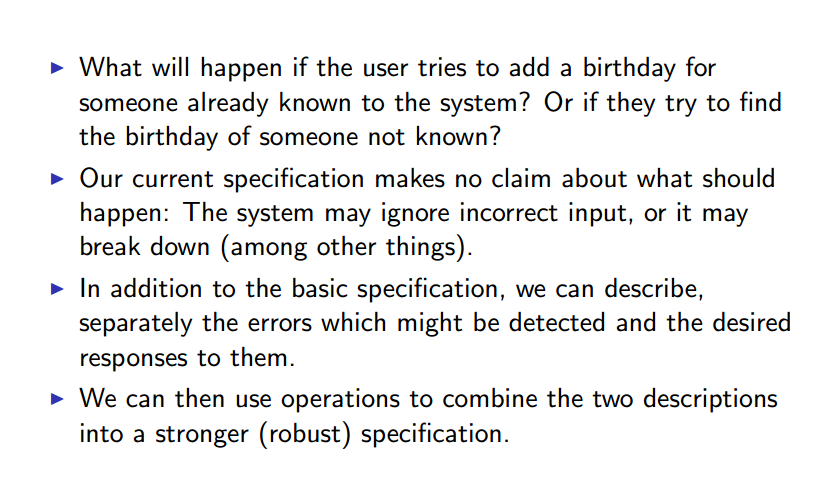


Handling errors  
如果人们试图加入一个生日，而这个人已经在system里？

如果不描写error ，会忽视,或彻底崩溃

除了基本的specification规则，我们还可以分别描述不同的error和对应回复

我们可以用操作来把两个描述连接成一个更强大的specification

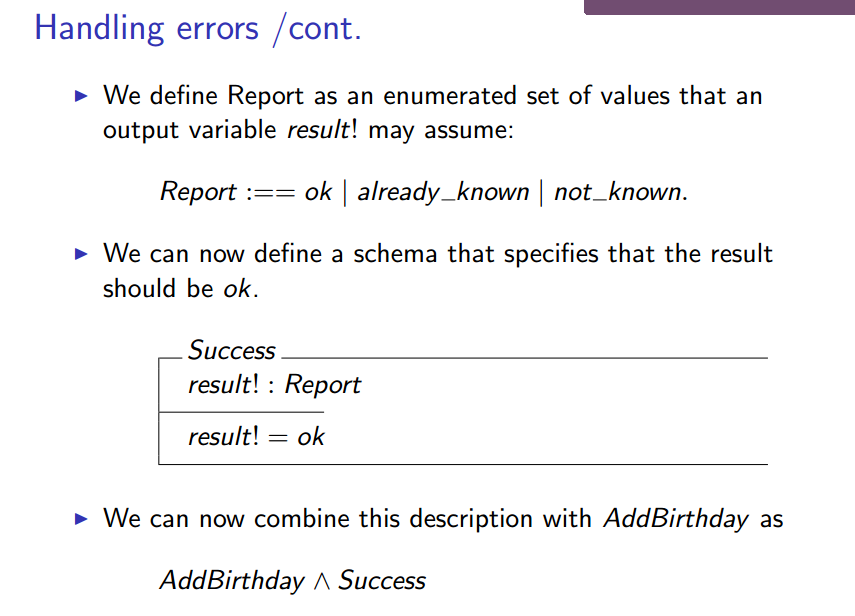


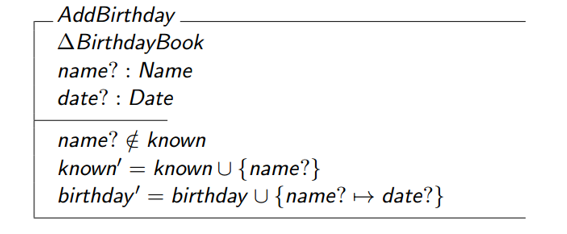
我们描述一个新Type Report:枚举描述，是result!的类型

可以使，ok, alredy\_known,not\_known

我们可以定义一个schema来指定result应该OK，现在我们combine两者

success的唯一作用就是输出ok

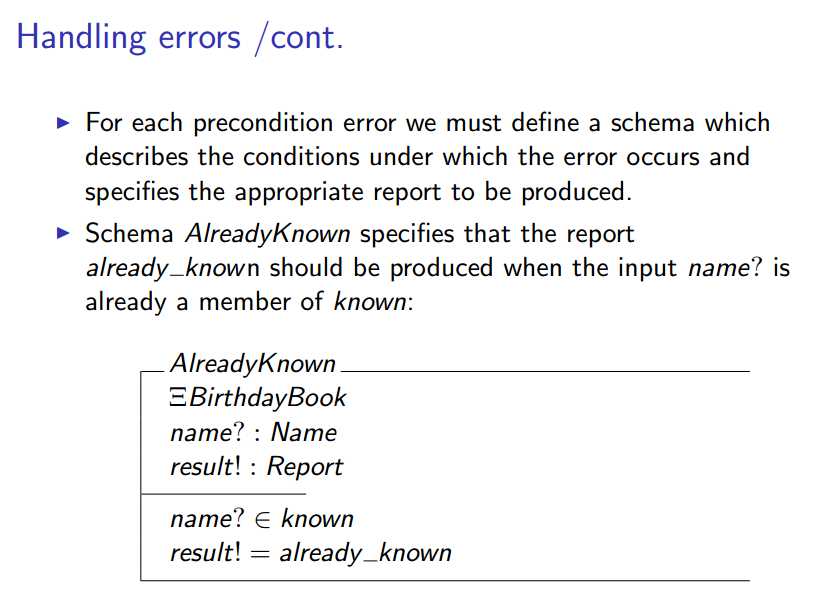


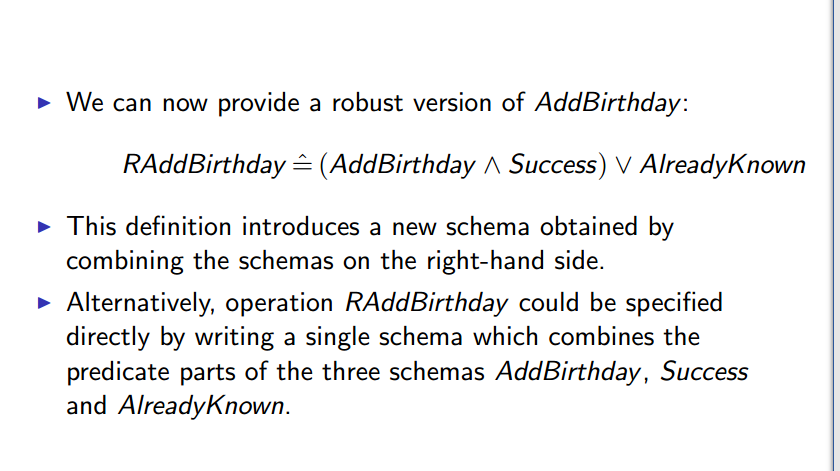


add birthday成功了以后就会输出OK!

对于先决条件产生的error，我们必须要用schema来控制

ALREADY KNOWN规定了name必须是known的一部分，这样result就会被设置成already\_known

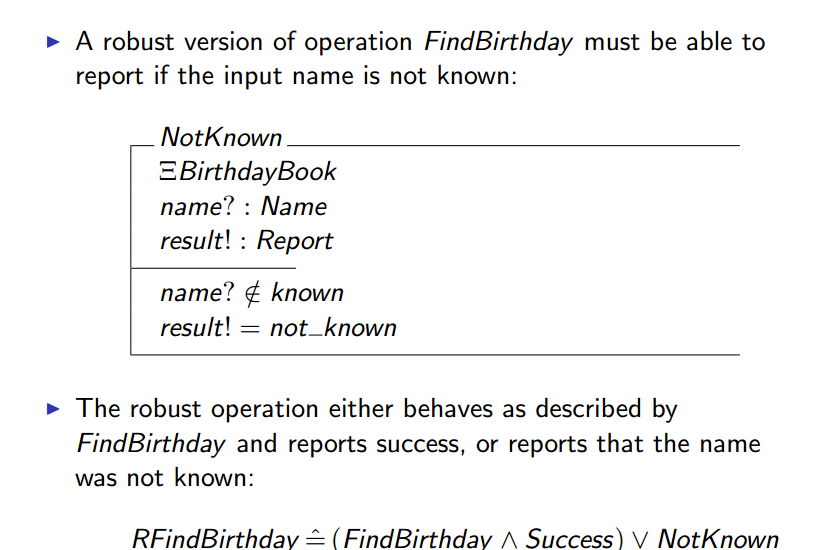




再次链接，ADDBITRHDAY成功输出OK(SUCCESS),失败already known

这个描述了一个新schema通过combine

当然你也可以直接写一个Raddbirthday



同理·