高级建模

|  |
| --- |
| **1.参考文献**  条件可能比状态变量和固定值之间的简单检查更复杂。例如，可能需要指定两个变量A和B必须具有相同的值（无论该值可能是多少）。这可以通过<if .. \。>标记的value属性内的*变量引用*进行编码。用大括号{}来表示对状态变量的引用，以将它们与通常的字符串区分开来。  例如，假设两个状态变量firstvar和secondvar，条件firstvar == secondvar`可以写为：  <if var="firstvar" relation="=" value="{secondvar}"/>  因此，如果firstvar = foo和secondvar = foo，条件将成立。大括号非常重要：如果没有它们，则条件表明firstvar必须将值secondvar（而不是由变量表示的值）作为值。  可以构建更复杂的条件：  <if var="firstvar" relation="=" value="something({secondvar},{thirdvar})"/>  在这种情况下，如果secondvar = foo，thirdvar = bar，firstvar = something（foo，bar），则情况将为true 。  变量引用可以类似地用于效果：  <set var="firstvar" value="{secondvar}"/>  在这种情况下，firstvar的值将被分配给secondvar的值。  以这种方式引用的每个变量被自动添加为从概率规则构建的规则节点的输入节点。  **2.量化**  通过在规则规范中引入普遍量化的变量，可以大大提高概率规则的表达能力。[[1]](http://www.opendial-toolkit.net/user-manual/advanced-modelling-1#%5B1%5D) 例如，人们可以编写一个规则来确定对象*X*的可抓性如何依赖于其形状：  <rule id="r3">   <case>     <condition>       <if var="shape({X})" value="sphere"/>     </condition>     <effect prob="0.9">       <set var="graspable({X})" value="true"/>     </effect>     <effect prob="0.1">       <set var="graspable({X})" value="false"/>     </effect>   </case>   <case>     <condition>       <if var="shape({X})" value="cone"/>     </condition>     <effect prob="0.2">       <set var="graspable({X})" value="true"/>     </effect>     <effect prob="0.8">       <set var="graspable({X})" value="false"/>     </effect>   </case> </rule>  然后该规则将应用对话状态中存在状态变量形状（*X*）的任意值*X.*为了区分自由变量*X*与正常字符串，自由变量必须用大括号{}包围。  自由变量也可以用于实用规则中：  <rule id="r4">   <case>     <condition>       <if var="task" value="grasp({X})"/>       <if var="graspable({X})" value="true"/>     </condition>     <effect util="2">       <set var="a\_m" value="grasp({X})"/>     </effect>   </case>   <case>     <effect util="-2">       <set var="a\_m" value="grasp({X})"/>     </effect>   </case> </rule>  您可能会注意到，对状态变量（参见上文）和自由变量的引用都使用相同的符号（大括号）。OpenDial自动判定是否符号表示由假设给定的参考或定量变量{X} 表示参考如果变量X在对话状态存在，并且将假设X表示自由变量否则。  **3.预测变量**  概率规则可以用于两个不同的目的。第一个目的是更新给定的一些新的信息的特定的状态变量的值（例如，推断用户对话充当*一个ù*给定所观察到的用户话语*Ú Ù*）。但是概率规则也可以用来 为将来的状态变量提供一个*预先*分配，这个变量预计会在下一个时间步骤中被观察到。举例来说，在最后一个用户的意图*我ü* 可以用来预测来自用户的下一个对话行为。这个预测本身并不会产生一个新的对话行为（这只是一个预测），但它可以作为这个观察的一个有用的先验。[[2]](http://www.opendial-toolkit.net/user-manual/advanced-modelling-1#%5B2%5D)  为了将这些先验值与实际观察值区分开来，OpenDial依赖于预测变量用上标^ p表示的约定。变量X ^ p因此表示对将来要观察的变量X的预测。  在运行时，OpenDial会自动将预测分布和观察分布连接起来，并推导出后验分布。  以下[逐步示例的规则](http://www.opendial-toolkit.net/user-manual/step-by-step-example) 说明，例如，如果系统要求用户重复，则用户需要遵从（并重复相同的对话行为），概率为0.95。  <rule id="repeatpredict">   <case>     <condition>       <if var="a\_m" value="AskRepeat" />     </condition>     <effect prob="0.95">       <set var="a\_u^p" value="{a\_u}" />     </effect>   </case> </rule>  请注意，对现有对话的引用会 影响a\_u。与其他假设相比，上述规则允许系统启动重复对话行为的可能性。  **4.字符串匹配**  对话系统的大多数变量都将值编码为字符串。OpenDial包含特殊功能，可以轻松地按照概率规则处理字符串。规则条件尤其适用于执行字符串匹配，如下例所示：  <rule id="r14">   <case>     <condition>       <if var="u\_u" value="take the {OBJ}" relation="contains"/>     </condition>     <effect>       <set var="a\_u" value="Request({OBJ})"/>     </effect>   </case> </rule>  上面的规则表明，如果变量u\_u包含匹配模式的子字符串，则使用{OBJ}（其中{OBJ}是占位符），变量a\_u将设置为请求（{OBJ}）。再次注意使用大括号来表示占位符。  如果插槽的实际内容可以忽略，则可以用通配符替换占位符：  <rule>   <case>     <condition>       <if var="u\_u" value="turn \* left" relation="contains" />     </condition>     <effect>       <set var="a\_u" value="Request(Left)" />     </effect>   </case> </rule>  上述规则将针对用户话语（例如“ 左转 ”，“ 左转 ”，甚至“ 现在机器人请转左 ”）（因为匹配可以是部分）而被解雇。  OpenDial也支持带有可选或替代子字符串的模式。可选的子字符串可以通过（the\_substring）来表示？建设，如在：  <if var="u\_u" value="turn (a bit)? to the left" relation="contains" />  上述条件将匹配“ 向左转 ”和“ 向左转 ”。  最后，替代元素可以通过构造（alternative\_1 | alternative\_2 | ...）表达，例如在以下情况下：  <if var="u\_u" value="turn (a bit|a little bit)? to the left" relation="contains"/>  这最后一个条件匹配字符串“ 向左转 ”，“ 向左转 ”和“ 向左转 ”。  **5.操纵元素集**  除了字符串之外，许多状态变量还与表示为元素*集*的值相关联（例如， 最近的*n个*对话行为或当前场景中感知的对象集）。概率规则带有内置语法来操纵这些集合。  *将值指定为集*  为了指定一个特定的值是一个集合，只需使用用方括号括起来的以逗号分隔的列表，例如：  <variable id="some\_set">    <value prob="0.6">[value1, value2, value3]</value>    <value prob="0.2">[value2,value4]</value>    <value prob="0.2">[]</value>  </variable>  *关系包含，！包含，in和！in*  规则条件可以检查变量中特定项目的存在/不存在，例如：  <if var="some\_set" value="value1" relation="contains"/>  上述条件将简单地检查变量some\_set的值是否包含元素值1。要检查元素是否*不是*给定集合的一部分，可以使用关系！contains（不包含）。  也可以通过in和in中的两个关系来检查某个单个元素是否包含在集合中，例如：  <if var="a\_u" value="[Request(Left),Request(Right)]" relation="in"/>  *允许多个值的效果*  默认情况下，规则的输出变量只能关联到单个值。考虑以下两条规则：  <rule id="r1">   <case>     <effect prob="0.9">       <set var="A" value="val1"/>     </effect>   </case> </rule>  <rule id="r2">   <case>     <effect prob="0.9>       <set var="A" value="val2"/>     </effect>   </case> </rule>  这两个规则r1和r2是冲突的，这意味着输出变量A的分布将是P（A = val1）= 0.495，P（A = val\_2）= 0.495和P（A =无）= 0.01。有时候，人们可能想要解除每个变量单个值的约束。例如，用户的最后一次对话行为可能包含几个可以相互组合的基本元素。在这种情况下，最后的对话行为可以表示为一组元素。  要指定输出变量的值不是互斥的，只需要插入属性exclusive =“false”。 例如，可以重写两条规则r1和r2：  <rule id="r1">   <case>     <effect prob="0.9">       <set var="A" value="val1" exclusive="false"/>     </effect>   </case> </rule>  <rule id="r2">   <case>     <effect prob="0.9">       <set var="A" value="val2" exclusive="false"/>     </effect>   </case> </rule>  输出变量的分布阿将在这种情况下是P（A = [VAL1，val2次]）= 0.81，P（A = [VAL1]）= 0.09，P（A = [val2次]）= 0.09和P（A =无）= 0.01 **。**请注意，具有多个值的这种影响仅适用于概率规则（而非实用规则）。  *将元素添加到现有列表*  如果您有一个变量some\_set，其值被定义为列表，并且您希望规则为其添加新的元素val，则可以将更新的变量定义为现有列表和新元素的联合：  <set var="some\_set" value="{some\_set}+{val}" />  **6.嵌套条件**  我们在[对话领域](http://www.opendial-toolkit.net/user-manual/dialogue-domains)的章节中解释说 ，规则条件基本上被定义为基本条件的连接或分离。如果需要，还可以定义更高级的嵌套条件，例如A = a1 **而** **不是**（B = b1 **或** C！= c1）：  <condition>   <if var="A" relation="=" value="a1"/>   <not>     <or>       <if var="B" relation="=" value="b1"/>       <if var="C" relation="!=" value="c1"/>     </or>   </not> </condition>  三个XML标记可用于这种嵌套条件：or，and和not，分别表示析取，连词和否定。  **7.概率和效用函数**  给定效果的概率或效用通常由固定值（如in <effect prob="0.8">）或参数（如in <effect util="theta1">）编码。但是，也可以定义通过数学函数定义的更复杂的值。  作为一个例子，假设你想编写一个规则，该规则基于一组感知的可视对象找到包含大小属性（“大”，“小”）的引用表达式的潜在匹配。当然，您希望与包含单词“big”的表达式匹配的对象的概率与对象的感知（数值）大小成正比，而包含单词“small”的表达式的匹配概率应该成反比到对象大小。  对此进行编码的一种方法是将概率表示为数学函数：  <rule>     <case>         <condition>             <if var="size({Object})" value="{Size}" />                                 <if var="refer\_expression" relation="contains" value="small"/>         </condition>         <effect prob="exp(-{Size})">             <set var="refer\_matches value="{Object}" exclusive="false"/>         </effect>     </case>     <case>         <condition>             <if var="size({Object})" value="{Size}" />                                 <if var="refer\_expression" relation="contains" value="big"/>         </condition>         <effect prob="exp(-1/{Size})">             <set var="refer\_matches value="{Object}" exclusive="false"/>         </effect>     </case> </rule>  正如我们所看到的，第一个效应的概率与对象的大小成反比（即如果大小接近0，函数的值将接近1）。第二种效应恰恰相反。这些数学函数可以是任意复杂的，可以使用通常的算术运算符（+，\*， - ，/）和以下预定义函数：[abs，acos，asin，atan，cbrt，ceil，cos，cosh，exp， floor，log，log10，log2，sin，sinh，sqrt，tan，tanh]。  **8.自定义功能**  如果我们想在规则中使用不在上面列出的一组预定义数学函数中的特定函数，该怎么办？例如，在自然语言理解任务中，可能想要定义两个语义表示之间的某种“相似性”度量，但是没有简单的方法直接在规则中编码这种度量。  OpenDial的最后一个版本允许您在Java中定义自己的函数，并在概率规则中使用它们。首先，在XML域文件中，您应该按照以下方式声明您的函数：  <function name="your\_function\_name">class.path.to.your.function</function>  然后，您可以在概率规则中的任何位置使用此自定义函数。例如，假设你已经创建了一个函数相似度，它接受两个参数并返回两者之间的相似性度量，你可以这样写：  <rule>     <case>         <effect prob="similarity({u\_u},(move|go) \* forward)">             <set var="a\_u" value="Move(Forward)"/>         </effect>         <effect prob="similarity({u\_u},(go|turn) \* left)">             <set var="a\_u" value="Move(Left)"/>         </effect>     </case> </rule>  最后，最重要的是，您需要为您的自定义函数实际编写Java类class.path.to.your.function。该类**必须**实现接口java.util.function <List <String>，Value>。如果方法apply（List <String> args），它接受函数的两个参数，并返回结果（在这种情况下，结果必须是double值，因为它代表概率），该类的核心部分  。下面是相似度函数代码的一个非常简单的例子：  package opendial;  import java.util.List; import java.util.function.Function;  import opendial.bn.values.Value; import opendial.bn.values.ValueFactory; import opendial.templates.Template;  public class Similarity implements Function<List<String>,Value> {      public Value apply(List<String> args) {         String utterance = args.get(0);         Template t = Template.create(args.get(1));         if (t.partialmatch(utterance).isMatching()) {             return ValueFactory.create(1.0);         }         return ValueFactory.create(0.0);         } }  **9.关系结构**  到目前为止，我们使用的状态值仅限于简单的实体（字符串，数字等）。换句话说，基础状态表示基本上保持命题（槽 - 值对）。在很多情况下，人们希望以更先进的关系结构进行操作。考虑一个话语的语义内容，复杂的用户意图或分层任务（参见[D. Ramachandran和A. Ratnaparkhi。“用叠加关系树进行信念跟踪”（SIGDIAL 2015）](https://aclweb.org/anthology/W/W15/W15-4609.pdf)进行讨论）。   最后一个版本的OpenDial允许你创建这样的关系结构并使用概率规则来“操纵”它们。关系结构的语法基本上与Stanford Core NLP包中的语法相同。表示标有R的关系在两个元素A和B之间，简单地将其写为A R> B，并使用括号来划分子图。下面是一个简单的例子： [look subject> hearer object> [cylinder attribute> red location> right]]。上面的例子创建了一个有5个节点和4个关系的关系结构。   您还可以将属性关联到每个节点，例如在图形中的元素上提供一些附加标签。这通过添加|来完成bar后面跟着一个key：value对，紧跟在节点内容之后。例如，loves | pos：VB表示该节点的pos属性的值为VB。  然后，您可以创建利用这种关系结构的规则。例如，可以创建条件来检查特定关系模式的发生情况，例如[look subject> {X}]（任何具有节点外观并且后跟主题关系的子图）。而且您可以创建操纵图形的效果，以便合并或中继关系结构的某些部分。  [1]见[Lison（2014年）](http://folk.uio.no/plison/pdfs/thesis/thesis-plison2014.pdf)，第9页。这些量化机制的理论基础是67-68和74-76。  [2]见[Lison（2014年）](http://folk.uio.no/plison/pdfs/thesis/thesis-plison2014.pdf)，第9页。详情请参阅78-79。 |