

语义分析部分

先前的HAWatcher部分，语义分析部分在另一篇文章 `DSN20-Cross App Interference Threats in Smart Homes Categorization, Detection and Handling` 中由于语义分析部分较为复杂，且只针对Groovy语言的smartapp提取自动化规则，我并没有实现。

现如今在 `CP-IoT: A Cross-Platform Monitoring System for Smart Home` 中，实现了跨平台的自动化规则提取功能，下面是介绍：

介绍

- 先前的异常检测系统具有较低的跨平台特性：
- 大部分方法为SmartThings设计并开发基于代码的方法，用于规则提取/数据收集和威胁发现。然而，不同的平台具有很强的异质性。

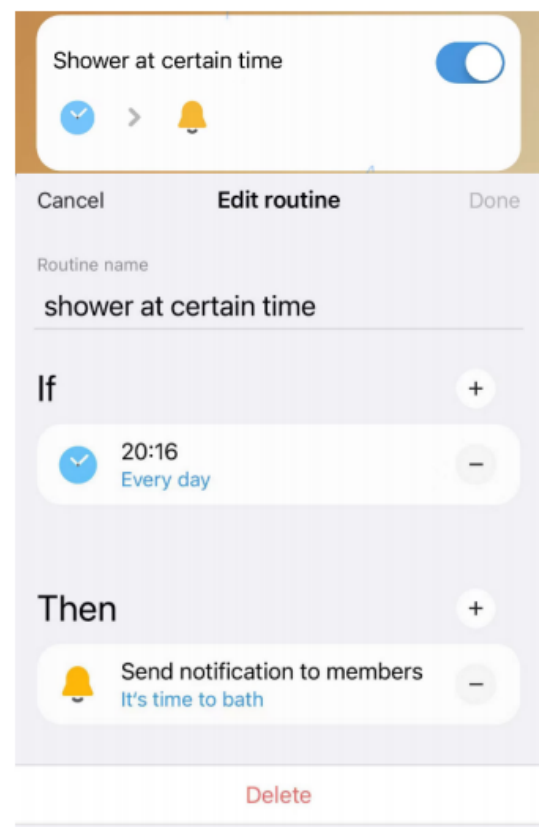
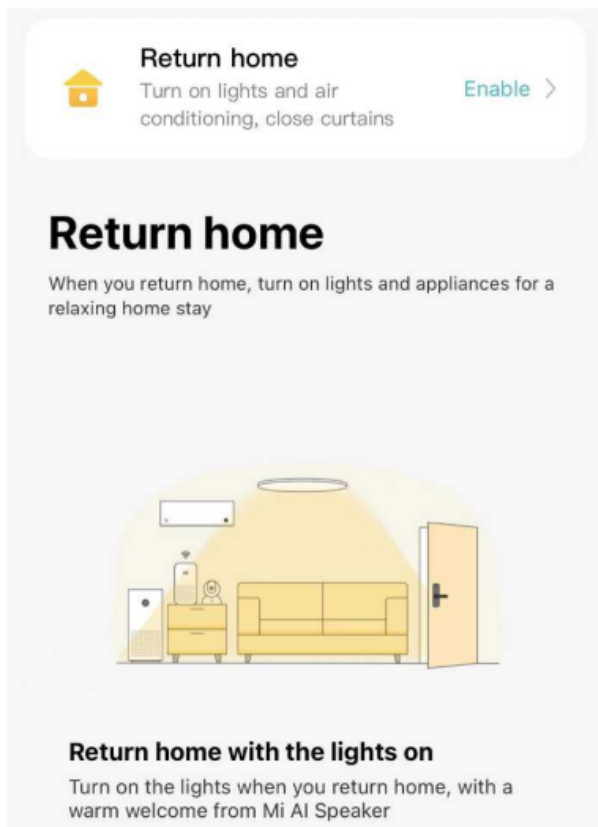
SmartThings使用Groovy作为编程语言，Google Home使用Kotlin和Java, Homekit使用Swift和Objective c。此外，一些平台的代码框架不是开源的，比如苹果的Homekit，对开发者的访问权限是有限的。传统的代码工具不是通用的。此外，两个不同平台之间的日志格式和流量模式存在一些差异，这增加了将特定于平台的框架移植到其他平台的难度。

- 如何从异构智能家居平台中提取自动化规则？常用的方法是分析源代码[14]、[15]、[24]、[29]、[30]。然而，不同的物联网平台之间存在很强的异构性，编程语言不同或代码框架不是开源的，因此基于代码的分析方法无法处理架构差异。另一种方法是使用UI页面的规则描述[15]，[31]，[32]。然而，不同平台的应用界面有着不同的描述粒度。例如，Homekit对自动化规则有模糊的描述。此外，大多数描述不包含用户策略，例如确定温度的阈值是否高。更糟糕的是，它们是为特定平台的应用设计的，无法处理不同平台之间的界面差异。

我们的想法是添加基于描述分析的配置页面的语义分析，这些分析在不同的平台之间共享。此外，它包含细粒度的规则定义和用户策略，两个阶段的结果可以相互补充，以达到较高的准确性。

- paper中语义分析部分的具体原理可参见 `V. SYSTEM MODEL CONSTRUCTION` 的 `A. Automation Rules Extraction` 部分，下面进行简单介绍：

原理



(a) The description page of automation rule “Return Home” in Xiaomi Home. (b) The configuration page of automation rule “Shower at certain time” in SmartThings.

自动化规则提取的来源参见上图，即各种智能家居平台中用户创建自动化规则的描述页面和配置页面，它们包含丰富的语义信息，等待我们提取。

整个提取过程如下：

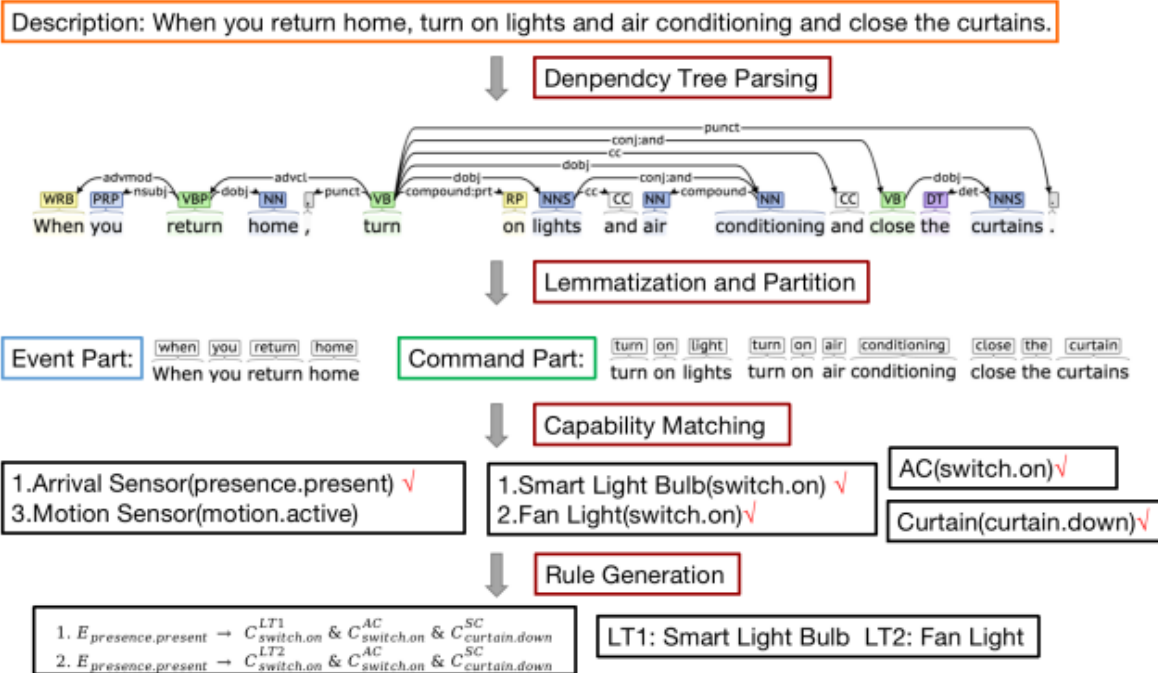


Fig. 5: The workflow of app descriptions analysis.

针对描述页面

针对描述页面，首先将描述自动化规则的句子进行依赖树解析，我们得到了每个单词的词性和单词之间的依赖关系。

然后，我们根据wh-副词(WRB)和标点符号(punct)确定句子的因果关系，并将其分为Event部分和Command部分。考虑到复杂的语法仍然存在，我们对两个部分都采用了词序化。

具体来说，我们把每个名词复合词和动词复合词作为一个整体来考虑，如“turn”和“on”，“air”和“conditioning”。基于动词(VB/VBP)和名词(NN/NNS)之间的直接对象(dobj)依赖关系，我们对它们进行成对组合，以获得每个事件和命令的最小表示。例如，动词复合词“turn on”和名词复合词“air conditioning”组合成一个命令从句。

接下来的部分是把Event部分和Command部分匹配成对应的capability-value pair，例如：

turning on/off the light can be abstracted into “switch.on”/“switch.off”，

我们将SmartThings开发人员文档[37]中的每个功能及其值结合到功能值对中，作为基础事实，这提供了跨不同平台的设备事件的良好抽象。

然后我们使用预训练的BERT[38]模型来获得每个字符串的嵌入，并使用余弦相似度来计算子句与每个能力值对之间的相关性。最后，将相似度得分最高的能力值对作为匹配结果。

针对配置页面

配置页面是描述页面的辅助，之所以要单独拎出来配置页面分析，是因为考虑到温度、湿度等带有具体值的规则，在描述页面是体现不出来的。

具体来说，我们将IF/Condition/Trigger/WHEN块的内容与规则的Event部分相匹配，将THEN/Action/Adjust块的内容与规则的Command部分相匹配。如图4(b)所示，IF块中的“20:16”与“time.startTime”有关。同样，我们通过BERT模型获得每个块中内容的嵌入，并计算与每个能力值对的相似度。

工具

StanfordCoreNLP

Stanford CoreNLP是由斯坦福大学自然语言处理组开发的一套自然语言处理工具集合，旨在帮助用户进行文本分析和理解。它集成了一系列强大的自然语言处理工具，包括分词、词性标注、命名实体识别、句法分析、情感分析和依存关系分析等功能。Stanford CoreNLP能够处理英语等多种语言的文本数据，并提供丰富的API接口，方便用户进行自然语言处理相关应用的开发和调用。

Stanford CoreNLP的功能和特点

- **多语言支持**：Stanford CoreNLP支持处理多种语言的文本数据，包括英语、中文等，具有较好的通用性和灵活性。
- **多模块集成**：包括分词、词性标注、命名实体识别、句法分析、情感分析、依存关系分析等模块，覆盖了自然语言处理的多个方面。
- **准确性高**：基于斯坦福大学在自然语言处理领域的研究成果，Stanford CoreNLP在文本分析的准确性和效率上表现优秀。
- **开源免费**：Stanford CoreNLP是开源项目，用户可以免费使用和修改源代码，符合很多开发者的需求。

直接给出一个例子：

```
# Simple usage
from stanfordcorenlp import StanfordCoreNLP

nlp = StanfordCoreNLP(r'G:\JavaLibraries\stanford-corenlp-full-2018-02-27')

sentence = 'Guangdong University of Foreign Studies is located in Guangzhou.'
print 'Tokenize:', nlp.word_tokenize(sentence)
print 'Part of Speech:', nlp.pos_tag(sentence)
print 'Named Entities:', nlp.ner(sentence)
print 'Constituency Parsing:', nlp.parse(sentence)
print 'Dependency Parsing:', nlp.dependency_parse(sentence)

nlp.close() # Do not forget to close! The backend server will consume a lot memory.
```

对应结果：

```
# Tokenize
['Guangdong', 'University', 'of', 'Foreign', 'Studies', 'is', 'located',
'u'in', 'Guangzhou', '.']

# Part of Speech
[(u'Guangdong', u'NNP'), (u'University', u'NNP'), (u'of', u'IN'), (u'Foreign',
u'NNP'), (u'Studies', u'NNPS'), (u'is', u'VBZ'), (u'located', u'JJ'), (u'in',
u'IN'), (u'Guangzhou', u'NNP'), (u'.', u'.')]

# Named Entities
```

```
[('Guangdong', 'ORGANIZATION'), ('University', 'ORGANIZATION'), ('of', 'ORGANIZATION'), ('Foreign', 'ORGANIZATION'), ('Studies', 'ORGANIZATION'), ('is', 'O'), ('located', 'O'), ('in', 'O'), ('Guangzhou', 'LOCATION'), ('.', 'O')]
```

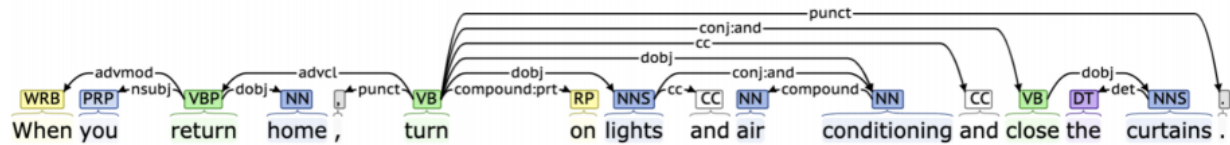
Constituency Parsing

```
(ROOT
  (S
    (NP
      (NP (NNP Guangdong) (NNP University))
      (PP (IN of)
        (NP (NNP Foreign) (NNPS Studies))))
    (VP (VBZ is)
      (ADJP (JJ located)
        (PP (IN in)
          (NP (NNP Guangzhou))))))
    (. .)))
```

Dependency Parsing

```
[('ROOT', 0, 7), ('compound', 2, 1), ('nsubjpass', 7, 2), ('case', 5, 3), ('compound', 5, 4), ('nmod', 2, 5), ('auxpass', 7, 6), ('case', 9, 8), ('nmod', 7, 9), ('punct', 7, 10)]
```

这里我们用的主要是 Constituency Parsing 和 Dependency Parsing 这部分，也就是想要得到词性和依赖关系



下面是 when you return home, turn on lights and air conditioning and close the curtains. 这句话的分析结果：

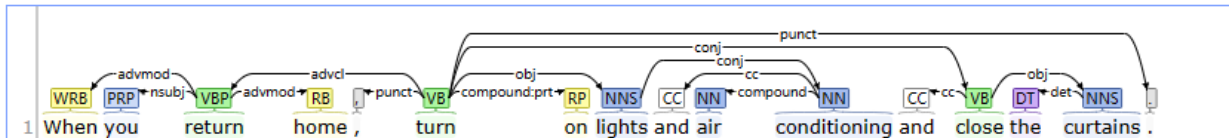
```
(ROOT
  (S
    (SBAR
      (WHADVP (WRB when)))
      (S
        (NP (PRP you))
        (VP (VBP return)
          (ADVP (RB home)))))
    (, ,)
    (VP
      (VP (VB turn)
        (PRT (RP on))
        (NP
          (NP (NNS lights))
          (CC and)
          (NP (NN air) (NN conditioning)))))
      (VP (VB close)
        (NP (DT the) (NNS curtains)))
      (. .)))
```

```

(CC and)
(VP (VB close)
  (NP (DT the) (NNS curtains))))
(. .)))
[('ROOT', 0, 6), ('advmod', 3, 1), ('nsubj', 3, 2), ('advcl', 6, 3), ('advmod', 3, 4), ('punct', 6, 5), ('compound:prt', 6, 7), ('obj', 6, 8), ('cc', 11, 9), ('compound', 11, 10), ('conj', 8, 11), ('cc', 13, 12), ('conj', 6, 13), ('det', 15, 14), ('obj', 13, 15), ('punct', 6, 16)]

```

Process finished with exit code 0



可以看到和预想的结果大差不差，但有一点是需要注意的，这里模型把 return 和 home 的关系理解为副词修饰的关系，paper得出的结果是动词和名词的关系，两者都对，但确实不一样。

但是，参照corenlp可视化网站 (<https://corenlp.run/>) 可知，基础版的依赖树解析并不能达到我们想要的效果，比如 turn 和 conditioning 只有在 Enhanced++ 版本才认为是obj关系，这就很麻烦，如果用基础版的还得自己推导，所以我们应该尽量使用增强版的。

version 4.5.5

— Text to annotate —

When you return home, turn on lights and air conditioning and close the curtains.

— Annotations —

parts-of-speech X named entities X dependency parse X

— Language —

English

Submit

Part-of-Speech:

1 When you return home, turn on lights and air conditioning and close the curtains .

Named Entity Recognition:

1 When you return home, turn on lights and air conditioning and close the curtains .

Basic Dependencies:

1 When you return home, turn on lights and air conditioning and close the curtains .

Enhanced++ Dependencies:

1 When you return home, turn on lights and air conditioning and close the curtains .

CoreNLP Tools:

TokensRegex Semgrex Tregex

增强版参照链接：[分词、词性标注、命名实体识别、句法分析？三行Python代码调用斯坦福自然语言处理工具~ - 知乎 \(zhihu.com\)](https://corenlp.run/)

作者提供的方法：

- 1.调用网站的结果。这个会受延迟影响。
- 2.本地调用。这个比较好，我采用这个。

代码略，结果跑出来很不错，`enhanced++ dependencies` 满足了我想要的效果.下面分析一下（在这里我把与语义分析无关的结果省略）：

```
[
  {
    "dep": "advmod",
    "governor": 3,
    "governorGloss": "return",
    "dependent": 4,
    "dependentGloss": "home"
  },
  {
    "dep": "obl:on",
    "governor": 6,
    "governorGloss": "turn",
    "dependent": 8,
    "dependentGloss": "lights"
  },
  {
    "dep": "compound",
    "governor": 11,
    "governorGloss": "conditioning",
    "dependent": 10,
    "dependentGloss": "air"
  },
  {
    "dep": "obl:on",
    "governor": 6,
    "governorGloss": "turn",
    "dependent": 11,
    "dependentGloss": "conditioning"
  },
  {
    "dep": "obj",
    "governor": 13,
    "governorGloss": "close",
    "dependent": 15,
    "dependentGloss": "curtains"
  },
]
```

上面的结果很好的描述了所有的自动化规则。

至于如何区分Event和Command，就非常简单了，只需要根据词性标注pos找出 `WRB` 和 `,`，即可划分

```
[
  {
    "index": 1,
    "word": "when",
    "originalText": "when",
    "characterOffsetBegin": 0,
    "characterOffsetEnd": 4,
    "pos": "WRB",
    "before": "",
    "after": " "
  },

  {
    "index": 5,
    "word": ",",
    "originalText": ",",
    "characterOffsetBegin": 20,
    "characterOffsetEnd": 21,
    "pos": ",",
    "before": "",
    "after": " "
  },
]
```

注：结果解释：

- 'dep': 依存关系的类型，通常包含有关两个词之间关系的信息。
- 'governor': 主导词的索引。
- 'governorGloss': 主导词的内容。
- 'dependent': 从属词的索引。
- 'dependentGloss': 从属词的内容。

BERT

BERT (Bidirectional Encoder Representations from Transformers) 是一种基于Transformer架构的预训练语言模型，由Google在2018年提出。它通过大规模无标签文本数据进行预训练，然后通过微调或者迁移学习的方式应用到下游自然语言处理任务中。

BERT的核心思想是通过掩码语言模型 (Masked Language Model, MLM) 和下一句预测 (Next Sentence Prediction, NSP) 任务来学习句子的上下文语境，从而获得文本的深层表示。在预训练过程中，BERT会利用大量的文本数据，将输入的文本序列中的一些词随机地用特殊的掩码符号替换，然后让模型预测被掩码的词。这样的方式可以使得模型在训练过程中同时考虑到句子中的前后文信息，从而更好地理解句子的语义。

BERT模型的主要特点包括：

1. 双向性：BERT模型通过使用Transformer编码器中的多头自注意力机制来处理文本序列，从而实现了双向编码，使得模型能够同时考虑到输入句子中的前后文信息。
2. 预训练与微调：BERT模型首先在大规模文本数据上进行预训练，然后通过微调的方式应用到各种下游自然语言处理任务中，如文本分类、命名实体识别、问答系统等。

- 3. 多层架构：BERT模型采用了多层的Transformer编码器作为基础架构，在预训练阶段可以选择不同层数的编码器进行训练，从而灵活地平衡模型的性能和计算资源之间的关系。
- 4. 多任务学习：BERT模型的预训练过程包括两个任务，即掩码语言模型和下一句预测，这样的设计可以使得模型学习到更加丰富和深层次的语言表示。

实现

新增python文件：

name	usage
get_rules_from_GUI.py	the main workflow of getting rules from description page
causal_division.py	Lemmatization and Partition
calculate_similarity_BERT.py	Capability Matching

Dependency Tree Parsing

这部分不多说了，直接调用函数即可。

Lemmatization and Partition

在后续操作之前，需要知道token和dependency_tree和constituency_tree的数据结构，方便后续处理：

```
corenlp_dir = (str) C:\semantic_analysis\stanford-corenlp-4.3.0
token = (list: 1) [(index: 1, 'word': 'When', 'originalText': 'When', 'characterOffsetBegin': 0, 'characterOffsetEnd': 4, 'pos': 'WRB', 'before': '', 'after': ' '), (index: 2, 'word': 'you', 'originalText': 'you', 'characterOffsetBegin': 5, 'characterOffsetEnd': 8, 'pos': 'PRP', 'before': ' ', 'after': ' '), (index: 3, 'word': 'return', 'originalText': 'return', 'characterOffsetBegin': 9, 'characterOffsetEnd': 15, 'pos': 'VBP', 'before': ' ', 'after': ' '), (index: 4, 'word': 'the', 'originalText': 'the', 'characterOffsetBegin': 16, 'characterOffsetEnd': 19, 'pos': 'DT', 'before': ' ', 'after': ' '), (index: 5, 'word': 'home', 'originalText': 'home', 'characterOffsetBegin': 20, 'characterOffsetEnd': 24, 'pos': 'NN', 'before': ' ', 'after': ' '), (index: 6, 'word': '.', 'originalText': '.', 'characterOffsetBegin': 24, 'characterOffsetEnd': 25, 'pos': '.', 'before': ' ', 'after': ' ')]

dependency_tree = (list: 1) [(dep: 'ROOT', 'governor': 0, 'governorGloss': 'ROOT', 'dependent': 7, 'dependentGloss': 'turn'), (dep: 'advmod', 'governor': 7, 'governorGloss': 'turn', 'dependent': 1, 'dependentGloss': 'When'), (dep: 'nsubj', 'governor': 3, 'governorGloss': 'return', 'dependent': 2, 'dependentGloss': 'you'), (dep: 'advcl', 'governor': 7, 'governorGloss': 'turn', 'dependent': 3, 'dependentGloss': 'return'), (dep: 'det', 'governor': 5, 'governorGloss': 'home', 'dependent': 4, 'dependentGloss': 'the'), (dep: 'obj', 'governor': 3, 'governorGloss': 'return', 'dependent': 5, 'dependentGloss': 'home')]
```

能够看出，它们都是list嵌套一个list嵌套n个dict。

constituency_tree结构：字符串

(ROOT (S (SBAR (WHADVP (WRB When)) (S (NP (PRP you)) (VP (VBP return) (NP (DT the) (NN home)))))) (, .) (VP (VP (VB turn) (PRT (RP on)) (NP (NP (NNS lights)) (CC and) (NP (NN air) (NN conditioning)))) (CC and) (VP (VB close) (NP (DT the) (NNS curtains)))) (. .)))

接下来首先进行分区。

分区

事实上，这部分 cp-IoT 并没有讲清楚是如何分区的，只简单提了一句，通过查阅其他文献 IOTGAZE: IoT Security Enforcement via Wireless Context Analysis 可得，具体的做法是：

构建 constituency parse tree 即选区解析树，并将句子按标签S(简单陈述句)或SBAR(由从属连词引入的从句)进行拆分。如图所示，句子 Turn your lights on when a open/close sensor opens and the space is dark. 的选取解析树是：

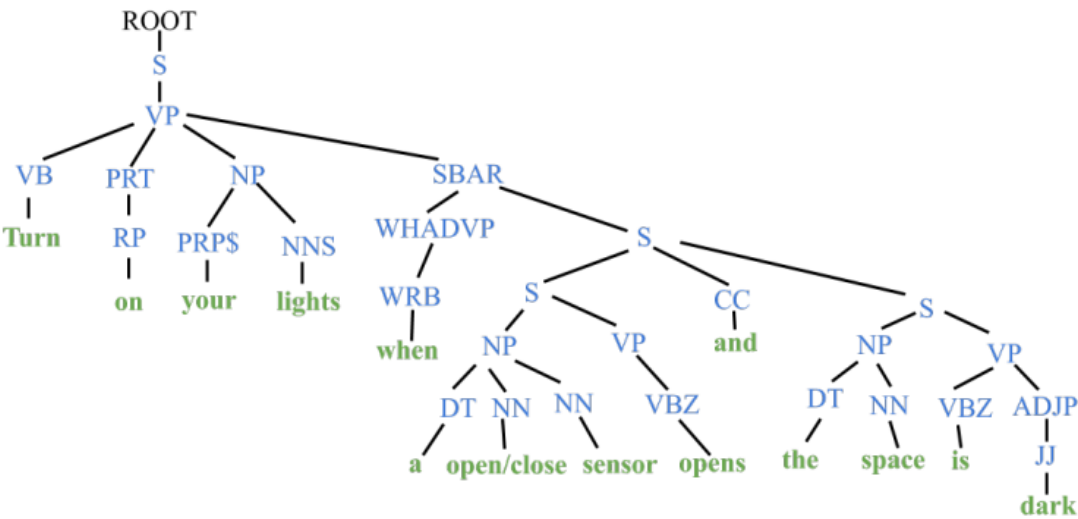
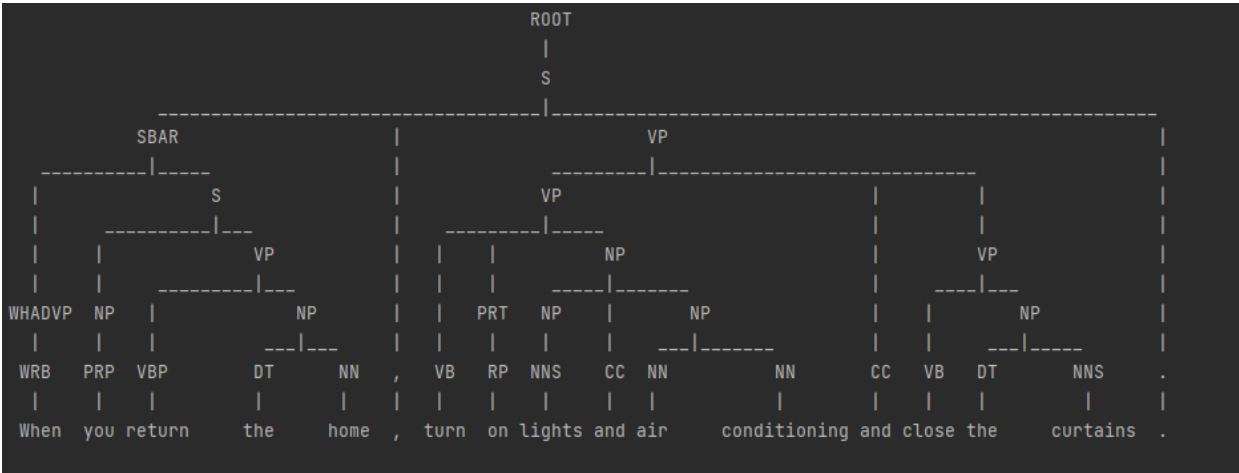
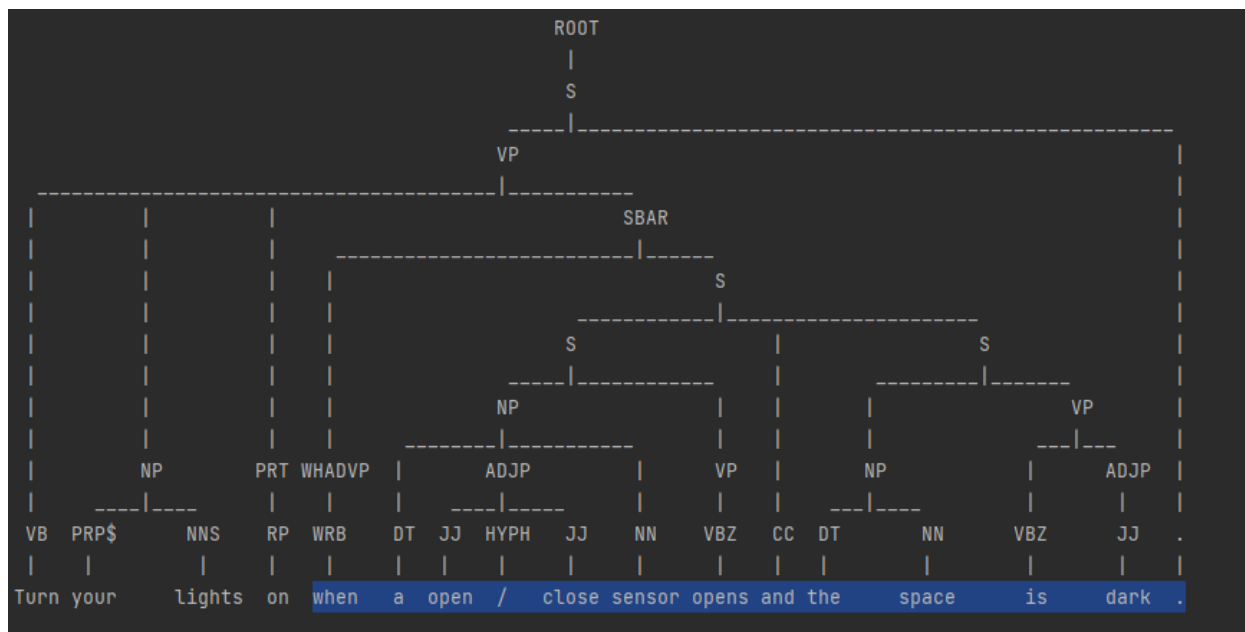


Fig. 4. Stanford constituency tree representation of the description from the Brighten-Dark-Places SmartApp.

程序跑出来的结果是：



(ROOT (S (SBAR (WHADVP (WRB When)) (S (NP (PRP you)) (VP (VBP return) (NP (DT the) (NN home)))))) (, .) (VP (VP (VB turn) (PRT (RP on)) (NP (NP (NNS lights)) (CC and) (NP (NN air) (NN conditioning)))) (CC and) (VP (VB close) (NP (DT the) (NNS curtains)))) (. .)))



(ROOT (S (VP (VB Turn) (NP (PRP\$ your) (NNS lights)) (PRT (RP on)) (SBAR (WHADVP (WRB when)) (S (S (NP (DT a) (ADJP (JJ open) (HYPH /) (JJ close)) (NN sensor)) (VP (VBZ opens)))) (CC and) (S (NP (DT the) (NN space)) (VP (VBZ is) (ADJP (JJ dark))))))))) (. .)))

程序结果很逆天，结果给的是字符串，涉及到括号匹配、移除所有词性标记的模式（即移除例如(NP, VP等），这部分通过chatgpt生成了对应的代码。具体运用的知识有括号匹配和正则表达式。

下面是结果展示：

```

causal division (1) x
C:\Users\Administrator\AppData\Local\Programs\Python\Python312\python.exe "G:/学习
SBAR part: (SBAR (WHADVP (WRB when)) (S (S (NP (DT a) (ADJP (JJ open) (HYPH /) (JJ
Rest of the sentence: (S (VP (VB Turn) (NP (PRP$ your) (NNS lights)) (PRT (RP on))

Event part: when a open / close sensor opens and the space is dark
Command part: Turn your lights on

Process finished with exit code 0

Event part: When you return the home
Command part: , turn on lights and air conditioning and close the curtains
  
```

相关代码见 `causal_division.py`

词序化

分区之后需要进行词序化，首先看一下两部分的依赖解析树：

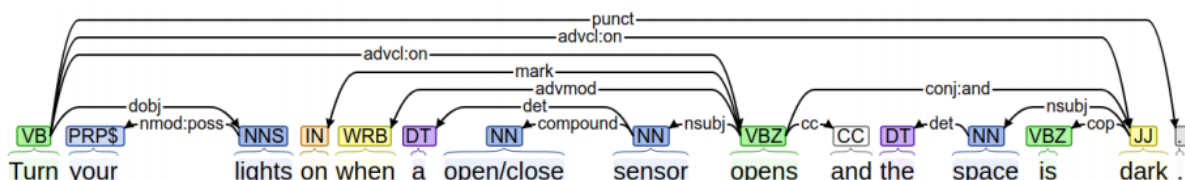
```
{'dep': 'ROOT', 'governor': 0, 'governorGloss': 'ROOT', 'dependent': 3,
'dependentGloss': 'return'}
{'dep': 'advmod', 'governor': 3, 'governorGloss': 'return', 'dependent': 1,
'dependentGloss': 'when'}
{'dep': 'nsubj', 'governor': 3, 'governorGloss': 'return', 'dependent': 2,
'dependentGloss': 'you'}
{'dep': 'det', 'governor': 5, 'governorGloss': 'home', 'dependent': 4,
'dependentGloss': 'the'}
{'dep': 'obj', 'governor': 3, 'governorGloss': 'return', 'dependent': 5,
'dependentGloss': 'home'}
```

```
{'dep': 'ROOT', 'governor': 0, 'governorGloss': 'ROOT', 'dependent': 2,
'dependentGloss': 'turn'}
{'dep': 'punct', 'governor': 2, 'governorGloss': 'turn', 'dependent': 1,
'dependentGloss': ','}
{'dep': 'compound:prt', 'governor': 2, 'governorGloss': 'turn', 'dependent': 3,
'dependentGloss': 'on'}
{'dep': 'obj', 'governor': 2, 'governorGloss': 'turn', 'dependent': 4,
'dependentGloss': 'lights'}
{'dep': 'cc', 'governor': 7, 'governorGloss': 'conditioning', 'dependent': 5,
'dependentGloss': 'and'}
{'dep': 'compound', 'governor': 7, 'governorGloss': 'conditioning', 'dependent': 6,
'dependentGloss': 'air'}
{'dep': 'obj', 'governor': 2, 'governorGloss': 'turn', 'dependent': 7,
'dependentGloss': 'conditioning'}
{'dep': 'conj:and', 'governor': 4, 'governorGloss': 'lights', 'dependent': 7,
'dependentGloss': 'conditioning'}
{'dep': 'cc', 'governor': 9, 'governorGloss': 'close', 'dependent': 8,
'dependentGloss': 'and'}
{'dep': 'conj:and', 'governor': 2, 'governorGloss': 'turn', 'dependent': 9,
'dependentGloss': 'close'}
{'dep': 'det', 'governor': 11, 'governorGloss': 'curtains', 'dependent': 10,
'dependentGloss': 'the'}
{'dep': 'obj', 'governor': 9, 'governorGloss': 'close', 'dependent': 11,
'dependentGloss': 'curtains'}
```

按照 CP-IOT 的说法，主要工作分为两步：

- 1.把compound部分结合起来
- 2.obj关系的词语就是主语和宾语

事实上这并不严谨，很多句子其实是识别不出来的，比如



这个例句只有Command部分是dobj，其他都是主谓/主系表结构，很显然是识别不出来的。

但其实 IOT-GAZE 这部分也并不严谨，都没有考虑完全。

在这里我就直接按照cp-iot这篇文章的方法来了。

跑出来的结果如图：

```
['return home']
['turn on lights', 'turn on conditioning air', 'close curtains']
```

conditioning air 这部分是错误的，之所以有这种错误，是因为依赖解析树中compound部分的顺序有时候是反的，比如

```
{'dep': 'compound', 'governor': 7, 'governorGloss': 'conditioning', 'dependent': 6,
'dependentGloss': 'air'}
```

所以需要加一层限制：每次遇到compound时两个word按原文的顺序组合。

结果正确：

```
['return home']
['turn on lights', 'turn on air conditioning', 'close curtains']
```

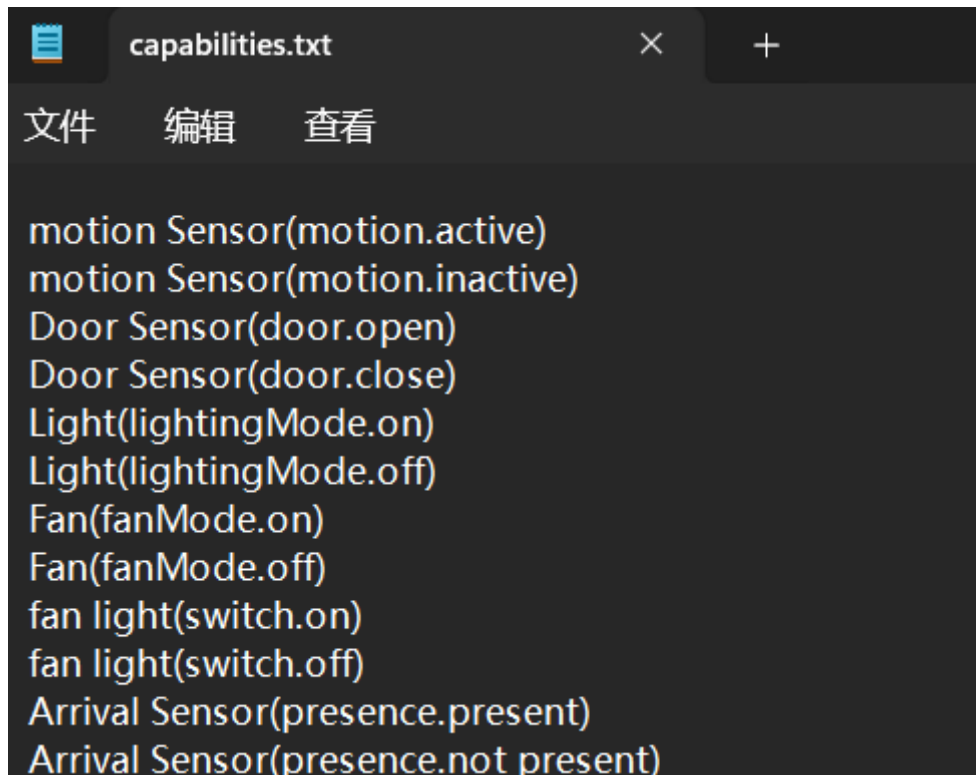
Capability Matching

首先我们需要熟悉smartThings中 Capability model：

TABLE I
EXAMPLES OF CAPABILITIES IN THE SMARTTHINGS FRAMEWORK

Capability	Commands	Attributes
capability.lock	lock(), unlock()	lock (lock status)
capability.battery	N/A	battery (battery status)
capability.switch	on(), off()	switch (switch status)
capability.alarm	off(), strobe(), siren(), both()	alarm (alarm status)
capability.refresh	refresh()	N/A

在开发者网站收集了一些device(capability.command):



```
capabilities.txt
文件 编辑 查看
motion Sensor(motion.active)
motion Sensor(motion.inactive)
Door Sensor(door.open)
Door Sensor(door.close)
Light(lightningMode.on)
Light(lightningMode.off)
Fan(fanMode.on)
Fan(fanMode.off)
fan light(switch.on)
fan light(switch.off)
Arrival Sensor(presence.present)
Arrival Sensor(presence.not present)
```

原文方法：

Then we use the pre-trained BERT [38] model to obtain the embedding of each string and use cosine similarity to compute the correlation between the clause and each capability-value pair

也就是要使用预训练的BERT模型获取文本串的嵌入（“embedding”指的是将每个字符串转换为密集的向量表示。），并使用余弦相似度来计算句子（如“turn on light”）与各种功能-值（如“Light(lightningMode.on)”）之间的相关性。

相关代码位于 `calculate_similarity_BERT.py`

```
return home:
motion Sensor(motion.active):0.8092541694641113
motion Sensor(motion.inactive):0.7884817123413086
Door Sensor(door.open):0.8126003742218018
Door Sensor(door.close):0.8153672218322754
Light(lightningMode.on):0.874354362487793
Light(lightningMode.off):0.8746254444122314
Fan(fanMode.on):0.8685153126716614
Fan(fanMode.off):0.8446669578552246
fan light(switch.on):0.8445630073547363
fan light(switch.off):0.8269367814064026
Arrival Sensor(presence.present):0.8103473782539368
Arrival Sensor(presence.not present):0.7479217648506165
Air conditioning(switch.on):0.8259332180023193
Air conditioning(switch.off):0.8214419484138489
Curtain(curtain.up):0.8894107937812805
Curtain(curtain.down):0.8885064721107483
audio(audioMute.mute):0.7921611070632935
audio(audioMute.unmute):0.8193264603614807
fan(fanSpeed.setFanSpeed):0.829630434513092
```

TV(tv.channelDown):0.8807995319366455
TV(tv.channelUp):0.8446865677833557
TV(tv.volumeDown):0.9059712886810303
TV(tv.volumeUp):0.8985701203346252
TV(tv.channelDown):0.8807995319366455
value(valve.close):0.832897961139679
value(valve.open):0.8196173310279846

turn on lights:

motion Sensor(motion.active):0.7882766127586365
motion Sensor(motion.inactive):0.7705143690109253
Door Sensor(door.open):0.8144199848175049
Door Sensor(door.close):0.809032678604126
Light(lightningMode.on):0.821605384349823
Light(lightningMode.off):0.8335978984832764
Fan(fanMode.on):0.7790526151657104
Fan(fanMode.off):0.7793368101119995
fan light(switch.on):0.8376429677009583
fan light(switch.off):0.8358701467514038
Arrival Sensor(presence.present):0.7773969769477844
Arrival Sensor(presence.not present):0.7508180737495422
Air conditioning(switch.on):0.8172221183776855
Air conditioning(switch.off):0.8261152505874634
Curtain(curtain.up):0.8064098358154297
Curtain(curtain.down):0.808698832988739
audio(audioMute.mute):0.7780826687812805
audio(audioMute.unmute):0.7831870317459106
fan(fanSpeed.setFanSpeed):0.7767094969749451
TV(tv.channelDown):0.8137410879135132
TV(tv.channelUp):0.7821347713470459
TV(tv.volumeDown):0.8095131516456604
TV(tv.volumeUp):0.7974914908409119
TV(tv.channelDown):0.8137410879135132
value(valve.close):0.8079686760902405
value(valve.open):0.7910963892936707

turn on air conditioning:

motion Sensor(motion.active):0.8124589920043945
motion Sensor(motion.inactive):0.7982076406478882
Door Sensor(door.open):0.8160245418548584
Door Sensor(door.close):0.8249605298042297
Light(lightningMode.on):0.8208644986152649
Light(lightningMode.off):0.8283361792564392
Fan(fanMode.on):0.8019250631332397
Fan(fanMode.off):0.7940729856491089
fan light(switch.on):0.8650178909301758
fan light(switch.off):0.8644896745681763
Arrival Sensor(presence.present):0.8017614483833313
Arrival Sensor(presence.not present):0.7734987139701843
Air conditioning(switch.on):0.8796746134757996
Air conditioning(switch.off):0.8882384300231934
Curtain(curtain.up):0.8155074715614319

```
Curtain(curtain.down):0.8158771395683289
audio(audioMute.mute):0.7863947153091431
audio(audioMute.unmute):0.7961013913154602
fan(fanSpeed.setFanSpeed):0.7734627723693848
TV(tv.channelDown):0.8312050104141235
TV(tv.channelUp):0.8081678152084351
TV(tv.volumeDown):0.8367226719856262
TV(tv.volumeUp):0.8274871110916138
TV(tv.channelDown):0.8312050104141235
value(valve.close):0.8124629259109497
value(valve.open):0.7926195859909058

close curtains:
motion sensor(motion.active):0.7790619730949402
motion sensor(motion.inactive):0.7545693516731262
Door sensor(door.open):0.8127344846725464
Door sensor(door.close):0.8166368007659912
Light(lightningMode.on):0.8597076535224915
Light(lightningMode.off):0.8705711364746094
Fan(fanMode.on):0.8187989592552185
Fan(fanMode.off):0.8112107515335083
fan light(switch.on):0.8425136208534241
fan light(switch.off):0.8314974308013916
Arrival sensor(presence.present):0.7879775762557983
Arrival sensor(presence.not present):0.7381658554077148
Air conditioning(switch.on):0.8202495574951172
Air conditioning(switch.off):0.8261350989341736
Curtain(curtain.up):0.865357518196106
Curtain(curtain.down):0.8694771528244019
audio(audioMute.mute):0.7924776673316956
audio(audioMute.unmute):0.806419849395752
fan(fanSpeed.setFanSpeed):0.8106038570404053
TV(tv.channelDown):0.8458131551742554
TV(tv.channelUp):0.8105347156524658
TV(tv.volumeDown):0.859366774559021
TV(tv.volumeUp):0.852849006652832
TV(tv.channelDown):0.8458131551742554
value(valve.close):0.8419775366783142
value(valve.open):0.8109685182571411

['TV(tv.volumeDown)']
['fan light(switch.on)', 'Air conditioning(switch.off)', 'Light(lightningMode.off)']
```

结果差强人意。只有部分规则能够生成。我认为是BERT模型的锅，并不能很好的表达出两个字符串之间语义信息的相似度。

在 IOTGAZE 中的方法是

Because the Word2Vec only gives embedding for words, we split every phrase into a tuple of individual words. This operation is also performed for capability names. We take the highest score of all the possible word pairs between a phrase tuple and a capability tuple as the similarity of these two tuples.

这个方法与HAWatcher中运用Word2vec的方法很类似，很明显都非常不严谨，无法保证正确率，如果在生成规则这一块都生成不了正确的，后面的异常检测就无从谈起。

Rule Generation

略

参考文献

官方文档

[Overview - CoreNLP \(stanfordnlp.github.io\)](https://stanfordnlp.github.io/Overview)

由于stanfordnlp是java语言编写的，python包是其他开源作者贡献的：

[Lynten/stanford-corenlp: Python wrapper for Stanford CoreNLP. \(github.com\)](https://github.com/Lynten/stanford-corenlp)

教程：

[Stanford CoreNLP超简单安装及简单使用，句法分析及依存句法分析 stanfordcorenlp-CSDN博客](#)

[Stanford CoreNLP简介及安装指南 - CSDN文库](#)

[Stanford CoreNLP 入门指南 - 知乎 \(zhihu.com\)](#)

[FileNotFoundError: [WinError 2](#)] 系统找不到指定的文件 winerror2系统找不到指定文件python-CSDN博客

[分词、词性标注、命名实体识别、句法分析？三行Python代码调用斯坦福自然语言处理工具~- 知乎 \(zhihu.com\)](#)

词性文档：

De Marneffe M C, Manning C D. Stanford typed dependencies manual[R]. Technical report, Stanford University, 2008.

可视化网站：

corenlp.run