HAWatcher复现工作计划表

- HAWatcher的复现工作大部分都需要自己完成（除了实验数据集的来源和correlation mining中的基于nlp计算哪些属性相关联），作者并未将工作开源，仅仅在github上传了testbed的一些细节和一个excel。
- 我认为这次的工作并不是出于要给谁看的目的，而是体会项目的工作流程，锻炼自己的项目实操能力，因此最后生成的结果有多难看，甚至最后能不能生成出来，我都不在意，关键是问问自己真正有没有收获。
- 具体每个工作阶段分配的时间合不合适（比如太多了还是太少了），目前只是根据工作量有个大概，后续在面对具体问题时会有所调整。
- 不求把每个细节做到完美（当然也做不到），而是尽力把整个流程顺下来，如果中间有实在解决不了的问题就先放在一边。

1.准备工作--1周

第一周打算进行一些准备工作，具体如下：

- HAWatcher原文并未提及语义提取工作的实现方法，具体需要参见作者前文`DSN20-Cross App Interference Threats in Smart Homes Categorization, Detection and Handling`,通过gpt我了解了语义提取的大概流程如下：
    1. **静态分析**：
        - 符号执行是一种静态分析技术，它系统地探索程序的所有可能执行路径，以发现潜在的错误或威胁。
        - 对于SmartApps，符号执行器会分析应用的源代码，以识别定义的自动化规则。
    2. **路径搜索策略**：
        - 由于SmartApps通常较小且执行路径有限，所以可以不遇到路径爆炸问题的情况下进行分析。
        - 使用简单的深度优先搜索策略来遍历SmartApp中的所有路径。
    3. **符号输入**：
        - 将数据值不依赖于其他数据的输入视为符号输入或源。
        - 在SmartApps中，这些源包括设备引用、设备属性值、设备事件、用户输入、HTTP响应以及API的常量值和返回值。
    4. **分析入口点和汇点（Sinks）**：
        - 入口点包括SmartApp的生命周期方法，如`installed`、`updated`和`uninstalled`。
        - 汇点包括受保护的设备命令和SmartThings的安全敏感API，如`setLocationMode()`。
    5. **生成控制流图（CFG）**：
        - 通过AST转换生成CFG，以模拟规则的触发器-条件-动作结构。
        - 每个订阅方法（`subscribe`）代表一个触发器，追踪处理程序以识别执行路径上的汇点。
    6. **约束生成**：
        - 从每个值赋值语句建立数据约束，从条件语句中建立谓词约束。

    - 处理三元表达式，将每个表达式分解为两个分支。
7. **API建模**：
    - 处理SmartThings提供的闭源API，通过手动审查开发者文档来建模对象、方法和对象属性访问。
    - 为API方法和属性访问编写静态建模函数，根据其参数和返回值进行建模。
8. **编译器定制**：
    - 通过Groovy编译器配置添加编译定制器实例，以便在语义分析阶段修改编译过程。

因此我认为第一步是阅读这篇文章的语义提取部分（第一周最重要的工作），熟悉其方法和细节，并将其作为复现工作的第一步，因为语义提取是HAWatcher实现的基础。

- 回顾python相关知识

  所有代码复现工作是使用python实现的，python相关知识需要回顾一下，这部分会很快，不应花太长时间，后面遇到什么具体困难再现学是最好的。

- 学会git

  目前有个想法是将每周的工作定期上传到github上，一方面可以起到监督的作用，另一方面是可以学习到git方面的知识（之前只知道在上面找资料，但是别人工作是如何上传是不了解的。

2.语义提取、分析工作--2-3周

这部分的工作主要是实现`DSN20-Cross App Interference Threats in Smart Homes Categorization, Detection and Handling`里的语义提取部分，以及如何从一个rule（trigger(T)-condition(C)-action(A)）生成假设相关性，另外要熟悉数据集。

我认为这部分工作是最艰巨的，原因是：

- 万事开头难。这里头不仅是心理的，更是实际的，有许多许多问题会在刚开始暴露（我猜的），可能会凭空产生很多之前没有想到的工作，现在能想到的有：怎么定义数据结构、可能的环境搭建。
- 目前还不清楚语义提取这块的工作量有多大，可能会很麻烦。
- 这部分工作完成了，后面的工作（`Semantic Analysis`、`Correlation Mining`、`Correlation Refining`、`Anomaly Detection`）都是基于从smart app提取的rule进行的操作，再难应该也不会难到哪里去。

具体工作内容：

- 熟悉学长给的数据。具体需要的数据有:
    - 日志（event logs）
    - 所用smart app的源代码
    - 数据中用到了哪些device，每个device具有什么属性等等。。
    - 日志中哪些是异常/故障（也有可能异常并不在日志中体现），异常有很多创造方法，有插入事件日志的，有修改事件日志的，有移除事件日志的，也有通过物理手段的等等

这部分尚不清楚具体的工作量，目测不会小。

这里也要打一个预防针，因为数据不可能是和原文作者自己在testbed生成的数据一模一样的，可能生成的异常和文章中的不一样，可能很多方面都不一样，也可能数据很乱，但这块影响不大，重在体验整个流程的实现上。

- 从smart app的源代码（groovy语言编写）提取语义，即**trigger(T)-condition(C)-action(A)**。
- 将rule转化成correlation(e2e correlation、e2s correlation)。

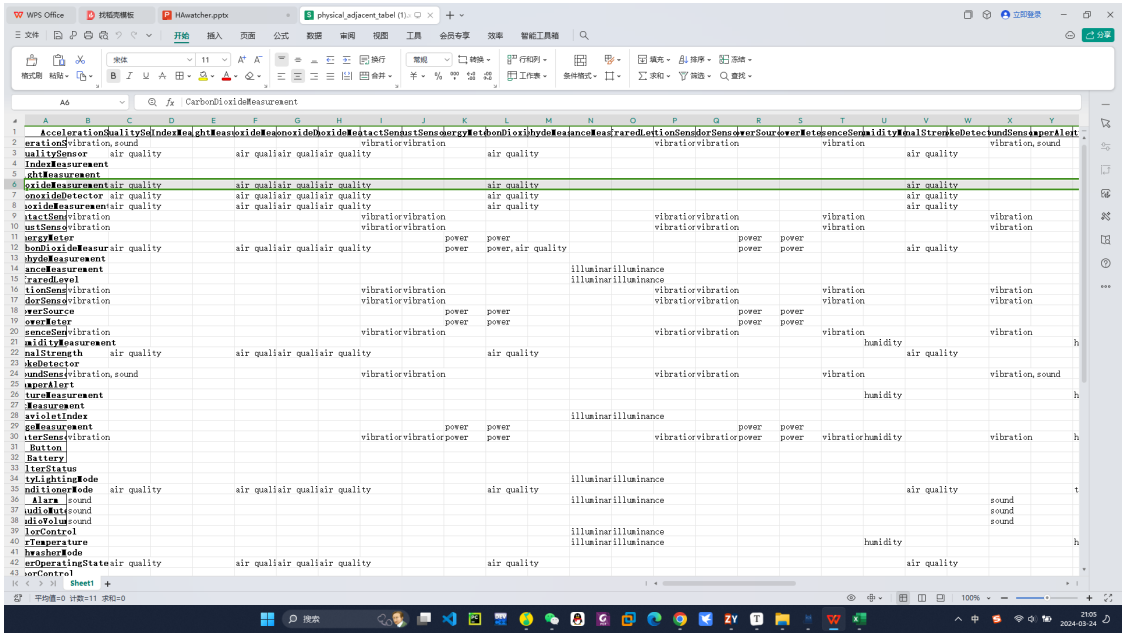
3.语义挖掘、细化工作--2-3周

这块的工作流程较多，但算不上困难，主要是Correlation Mining和Correlation Refining两块

Correlation Mining部分

主要分为三个工作：

- **Prepossessing Event Logs**，即日志预处理。主要工作是：
  - 3 sigma、
  - 聚类、
  - 移除重复值。
- **Hypothetical Correlation Generation**，也就是physical channel和 user activity channel的语义提取，具体工作有：
  - 基于nlp生成属性表，也就是哪些属性可以通过某个物理属性相关联。其实这部分工作作者已经做好了并把excel放到github上了，所以目前尚不确定自己到时候会不会重新做一遍，如果前面很顺利/时间很充裕/觉得这部分很有实现的意义，就自己做一遍。



  - 根据属性表生成所有可能的相关性。
  - 将智能应用程序的语义与邻接表的语义结合起来

    注：这部分virtual devices如何加入到邻接表中没说清楚。
- Hypothesis Testing，即遍历事件日志，利用假设检验看看生成的假设相关性哪些是符合真实世界的。这部分说的比较清楚，没什么疑问。

Correlation Refining部分工作

我认为这部分工作论文一点都没有说清楚，并且存在疏漏，作者之所以进行这步，其中一个重要原因是**当应用程序发生变化时，公认的假设相关性可能会过时**，因此需要替换，但我仔细分析了原理，认为这块完全起不到这个作用，**甚至作者后面自己给的应用程序发生变化的例子，根据这部分的原理，都解决不了。**

另外有的地方是真没说清楚，比如

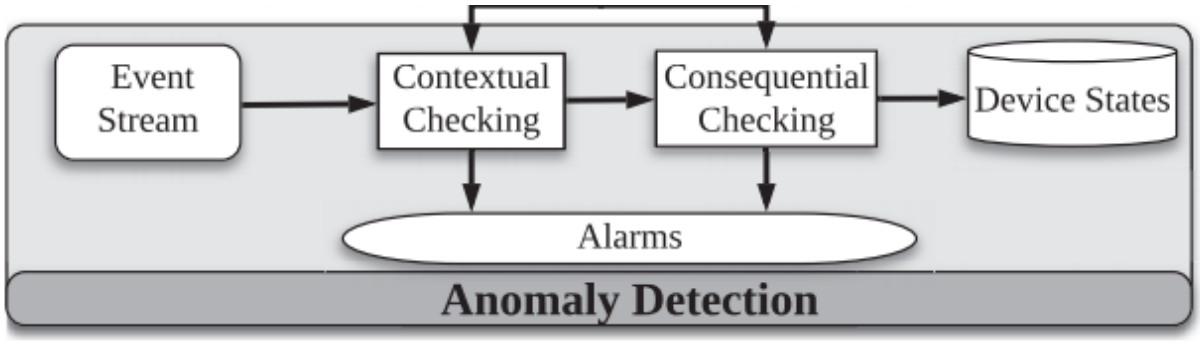
they have the same posterior event (i.e., $\mathcal{E}_b^{\beta(B)} = \mathcal{E}_d^{\delta(D)}$), and

2) $\mathcal{E}_a^{\alpha(A)}$ (logically) implies $\mathcal{E}_c^{\gamma(C)}$ (i.e., $\mathcal{E}_a^{\alpha(A)} \Rightarrow \mathcal{E}_c^{\gamma(C)}$).

满足什么条件前者可以imply后者压根没说，只有作者给的那一个例子是显而易见前者可以推出后者的，其他的情况全靠读者猜。

所以这部分未来可能会选择性跳过，因为refining这部分只是一个锦上添花的东西，所以不影响整个流程顺下来。

4.Anomaly Detection、evaluation--1-2周



这块的工作本质上是一个遍历的过程。之前的所有部分属于训练，这部分属于测试，比如作者用了前三周的数据集生成假设相关性，第四周的数据集用于anomaly detection，因此最初使用数据集时就要把数据集拆分开。

后面的评估工作，可以参照原文进行适当评估。