环境上下文一致性错误处理平台GUI版

1. 项目简介

1.1. 项目背景

上下文感知程序(context-aware application)已经在我们的生活中得到越来越广泛的应用，这类程序可以利用各种从环境中收集的信息，如GPS数据、环境温度、氧气浓度等来帮助程序提供更智能的服务，这部分程序感兴趣的环境信息也被称为程序的上下文(context)。上下信息通常是通过程序关联的各类传感器进行感知，但是由于传感器感知过程容易受到噪声等不可控因素的影响，最终感知到的上下文信息很有可能不准确、不完整甚至相互冲突，造成上下文不一致(inconsistency)问题，不一致的上下文信息会影响程序的正常功能，甚至会造成一些不可预期的严重后果。因此，如何对于上下文信息的一致性进一个预先的检测判断至关重要。

例如，一个上下文感知程序为监测城市X和Y的人员流动的程序P，如图1.1所示，它所关心的环境上下文为城市X和Y的人员流动情况，有人员离开或进入某个城市会使得上下文信息发生改变，在时刻3，程序P得到“Sanji enters city Y”的动作发生的信息后，程序P可能会发生错误，因为Sanji早 在0时刻已经进入了城市X，并且在时刻3之前程序P并未得到“Sanji leaves city X”的动作信息，程序P会认为Sanji在时刻3既城市X又在城市Y，即它认为Sanji在同一时刻出现在了不同的物理空间，这显然是不可能发生的，由此影响了程序P的正常功能。我们的项目适用于对这样的上下文不一致问题做出预先的检测和报告，为后续上下文信息的修复或直接丢弃提供指导作用。

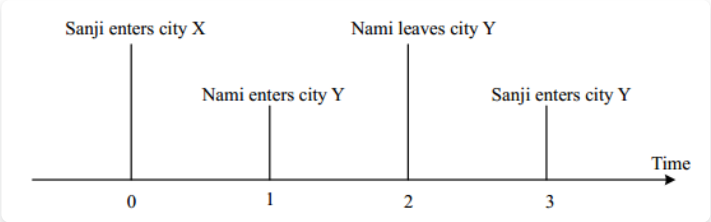

图1.1 城市X和Y的人员流动情况

1.2. 解决方案

由于直接判定上下文信息的一致性十分困难，所以我们项目的解决方式是通过根据上下文感知程序的应用场景预定义一系列不可违反的一致性规则(consistency constraint)，并根据这些规则来帮助检测上下文信息的一致性，一旦发现任何违反预定义规则的情况就认为此时的上下文信息存在不一致。

还是以图1.1为例，我们定义一致性规则为“同一个人不能同时出现在不同的地方”，因此，在时刻3，“Sanji enters city Y”的动作发生后，由于一致性规则的存在，这个不一致问题会被系统报告，从而避免程序P发生不可预测的错误。

2. 环境依赖

2.1. 硬件依赖

- 主存：至少8GB
- 显卡 (可选)：项目运行 GAIN 技术需要一块支持 CUDA 编程的 NVIDIA 系列显卡，这是一种基于 GPU 的加速技术，若无需运行 GAIN 则可忽略本条

2.2. 软件依赖

- 操作系统：Linux 或 Windows，Linux 推荐使用 Ubuntu 16.04 LTS版本，Windows 推荐使用 Windows 10版本
- Java 运行环境：Oracle Java 8 或以上的版本
- 显卡驱动与 CUDA Tool Kit (可选)：根据本机显卡的型号到驱动官网和NVIDIA开发者官网下载安装

3. 平台框架

如图3.1所示，平台GUI界面分为**数据展示区**和**功能选择区**两个部分：

图3.1 总体框架

3.1. 数据展示区

总体界面图的上半部分为**数据展示区**，以图表和文字结合的方式主要展示了以下内容：

- **已检测的数据量**：以柱状图的形式展示
- **数据间隔**：包括瞬时间隔和一段时间内的平均间隔，其中平均间隔分别取了0.5秒内和1秒内的平均值
- **检测进度**：以进度条的方式展示
- 检测所用到的**规则**和**模式**的个数
- **一致性错误数**：以柱状图的方式展示包括当前汇报的一致性错误数，以及误报数和漏报数（误报数和漏报数仅在动态检测下并提供了对比文件才会有展示，对比文件可以由相应的静态检测生成）
- **检测时间**：到目前为止的一致性检测耗时

除此之外，系统提供**导出日志**功能，可以将上述展示数据以日志的形式导出到文件中。

3.2. 功能选择区

总体界面的下办部分为**功能选择区**，可以选择和配置的功能如下：

- **模式文件**：选择一个描述上下文集合的模式文件
- **规则文件**：选择一个描述一致性规则的规则文件
- **检测技术**：通过组合框选择检测技术，可选项为ECC、PCC、Con-C、GAIN、CPCC
- **调度策略**：通过组合框选择调度策略，组合框中的可选项会根据**运行方式**的不同而变化
  - 在time-based下仅可选Immed
  - 在change-based下可选Immed、GEAS-ori、GEAS-opt
- **运行方式**：通过组合框选择运行方式，可选项为static-change-based、static-time-based、dynamic-change-based、dynamic-time-based
- **数据文件**：仅在静态检测可选，动态检测下数据由客户端发送，数据选择按键无效

- **对比文件**：仅在动态检测可选，默认静态检测所产生的log为对应动态检测的对比文件，若在动态检测时选择相应的对比文件会在动态检测过程中展示当前汇报的一致性错误的误报和漏报数
- **并发线程**：仅在**检测技术**为Con-C或CPCC的时候可选有效
- **启动**：在上述选项配置完成的情况下点击启动运行一致性检测，若由配置错误会有弹窗提示，关于错误处理会在后面给出例子
- **终止**：终止当前的一致性检测进程
- **退出**：退出当前程序
- **帮助**：打开帮助文档，获取更为详细的信息

4. 功能展示

4.1. 静态检测

以PCC + GEAS-ori为例，展示静态检测，效果如图4.1所示：

图4.1 静态检测示例

- 在检测过程中禁止修改配置
- 可以暂停、终止或退出程序，以及阅读帮助文档
- 可以点击选择的文件链接查看相应的模式、规则和数据文件
- 数据展示区相应地展示运行时数据

4.2. 动态检测

以ECC + GEAS-ori为例，展示动态检测服务器端，效果如图4.2所示：

图4.2 动态检测-服务器示例

- 在检测过程中禁止修改配置
- 可以暂停、终止或退出程序，以及阅读帮助文档
- 可以点击选择的文件链接查看相应的模式、规则和对比文件
- 数据展示区相应地展示运行时数据

客户端如图4.3所示：

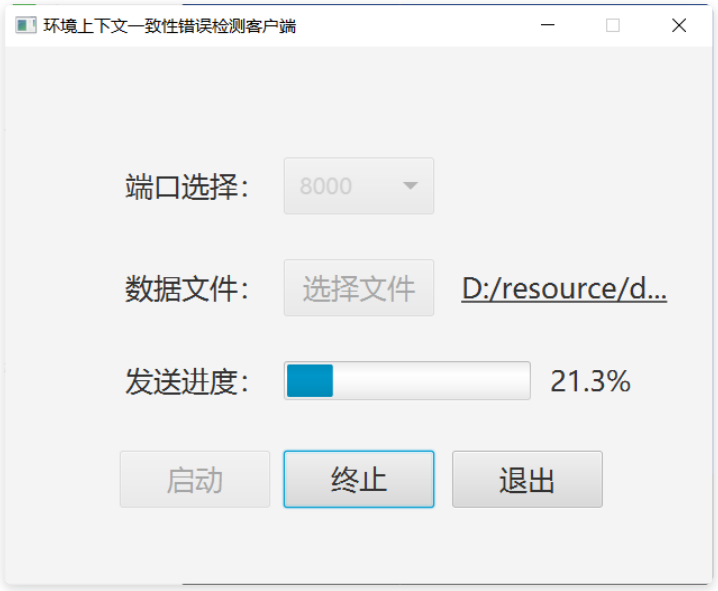

图4.3 动态检测-客户端示例

- **端口选择**：服务器端随机给出端口，客户端根据相应的端口值进行选择
- **数据文件**：选择要发送的上下文数据
- **启动**：点击后开始发送，并展示发送进度
- **终止**：点击后停止发送
- **退出**：点击后结束程序

4.3. 错误处理

- 某个配置项无配置，以模式文件项无配置为例，如图4.4所示，会给出以下错误提示：

图4.4 模式文件无配置

- 规则、模式或数据文件编写错误，以规则文件错误为例，如图4.5所示：

图4.5 规则文件错误

给出错误信息，并提示如何做出修改这个错误，以及指导用户查看规则文件的书写格式。

4.4. 配置差异

- 性能排序如下表所示，数字越小表示性能越高，“—”表示无该配置

	ECC	Con-C	GAIN	PCC	CPCC
Immed	14	13	12	8	—
GEAS-ori	11	10	6	5	3
GEAS-opt	9	7	4	2	1

- 资源开销排序如下表所示，数字越小开销越大，“—”表示无该配置

	ECC	Con-C	GAIN	PCC	CPCC
Immed	14	11	10	9	—
GEAS-ori	13	8	6	4	2
GEAS-opt	12	7	5	3	1

5. 附录

5.1. 规则文件格式

系统使用的规则文件是用一阶逻辑语言描述的，并用XML格式描述，列如，一致性规则为“所有出租车必须处于纬 [20, 30]之间” 则可以用一阶逻辑语言写成如下形式

$$\forall v_1 \in pat\_000(sz\_loc\_range(v_1))$$

将其写成XML格式如图5.1所示， 每个<rule>元素共有 2 个子元素：

```xml
<rule>
    <id>rule_00</id>
    <formula>
        <forall var = "v1" in = "pat_000">
            <bfunction name = "sz_loc_range">
                <param pos = "1" var = "v1" field = "object" />
            </bfunction>
        </forall>
    </formula>
</rule>
```

图5.1 规则的XML表示

- <id>：其文本内容表示该 rule 的名称，可任意修改，但不能出现重复名称
- <formula>：其包含的是用一阶逻辑描述的一致性规则的 XML 形式，它的一般描述方式如下：
    - 对于全称命题"$\forall v \in pattern(P(v))$"，其XML格式如图5.2所示：

```xml
<forall var=”v” in=”pattern”>
    <!-- pattern是由patterns.xml描述的特定数据集合 -->
    P(v)的XML形式
</forall>
```

图5.2 全称命题的XML表示

    - 对于存在命题“$\exists v \in pattern(P(v))$”，其XML格式如图5.3所示：

```xml
<exists var=”v” in=”pattern”>
    <!-- pattern是由patterns.xml描述的特定数据集合 -->
    P(v)的XML形式
</exists>
```

图5.3 存在命题的XML表示

    - 对于一阶公式“$\neg P(v)$”，其XML格式如图5.4所示：

```
<not>
    P(v)的XML形式
</not>
```

图5.4 非的XML表示

- 对于一阶公式"$P(x) \wedge Q(y)$"，其XML格式如图5.5所示：

```
<and>
    P(x)的XML形式
    Q(y)的XML形式
</and>
```

图5.5 与的XML表示

- 对于一阶公式"$P(x) \vee Q(y)$"，其XML格式如图5.6所示：

```
<or>
    P(x)的XML形式
    Q(y)的XML形式
</or>
```

图5.6 或的XML表示

- 对于一阶公式"$P(x) \rightarrow Q(y)$"，其XML格式如图5.7所示：

```
<implies>
    P(x)的XML形式
    Q(y)的XML形式
</implies >
```

图5.7 蕴含的XML表示

- 对于布尔函数"$sz\_loc\_range(v_1)$"，其XML格式如图5.8所示：

```
<bfunction id=" sz_loc_range">
    <!-- id表示函数名 -->
    <param pos="1" var="v1" field = "object">
</bfunction>
```

图5.8 布尔函数的XML表示

5.2. 模式文件格式

模式文件用于描述数据集合的特征，它所描述的集合供规则文件中的全程命题和存在命题使用，如图5.9所示， 每个<pattern> 元素共有 7 个子元素：

```xml
<pattern>
    <id>pat_000</id>
    <freshness>2000</freshness>  <!-- ms -->
    <category>location</category>
    <subject>any</subject>
    <predicate>any</predicate>
    <object>any</object>
    <site>any</site>
</pattern>

```

图5.9 模式的XML表示

- <id>：其文本内容是该模式的名称，可任意修改，但不能出现重复名称
- <freshness>：其文本内容表示数据加入模式表示的数据集合的时间限制，如取值为 2000 表示集合中数据的有效时间为 2000 毫秒
- <category>：元素表示模式的类型，可以任意修改，只作为注释用
- <subject>：在本例中元素文本内容默认为 any，表示对模式描述的数据集合中对数据主体不作限制，可根据具体情况修改
- <predicate>：在本例中元素文本内容可取值为 run_with_service、run_without_service 和 any，分别表示有载客、无载客和任意（有载客和无载客皆可），在特定场景中可以被赋予其它意义，表示数据主体的行动
- <object>：在本例中元素文本内容默认为 any，示对模式描述的数据集合中对数据客体不作限制，可根据具体情况修改
- <site>：在本例中元素文本内容可取值为 sutpc_x（x 为 0~9 的数字）和 any，分别表示尾号为 x 的车辆和任意车辆