arXiv:2303.15984v1 [cs.SE] 28 Mar 2023

基于规范的 VDM CSV 支持

Leo Freitas[1] 和 Aaron John Buhagiar[2]

[1] 计算机学院，纽卡斯尔大学，
leo.freitas@newcastle.ac.uk[2] 转化与临床研究所，纽卡斯尔大学，a.j.buhagiar2@newcastle.ac.uk

摘要。 CSV 是一种广泛使用的格式，用于表示系统控制、信息交换与处理、日志记录等方面的数据。然而，这种格式充斥着棘手的边界情况和不一致性，可能导致输入数据不可靠，从而使建模或仿真实验无法使用或不安全。我们通过提供一个**SAFE**-CSV VDM 库来解决这一问题。该库具备以下特点：**S**imple（简单）、**A**ccurate（准确）、**F**ast（快速）和**E**ffective（有效）。它扩展了其他 VDM 数学工具包扩展组成的生态系统，其中还包括用于 VDM 的 Isabelle 翻译与证明环境。

关键词： VSCode、VDM、CSV、文件格式、库

1 引言

逗号分隔值（CSV）格式广泛应用于各种领域：从数据科学，例如器官移植分配[1], , 到嵌入式系统中的表示，例如医疗器械的状态机 [3, 12], 、制药应用 [10],，以及各种系统上的日志文件、数据库、支付、政府系统等。尽管已有标准（RFC 4180）[2], , 但仍存在许多版本和变体[3]。它悠久的使用历史[4] 启发了各种其他用于数据管理和交换的“简单”格式，例如 JSON [5],、XML 和各种电子表格格式。

CSV 处理是涉及多个应用领域的计算工作中的重要组成部分。依赖 CSV 输入的领域饱受细微错误和不一致问题的困扰，这些问题最终可能导致与 CSV 数据相关的系统响应失效。当此类数据本身表示系统架构（例如拨号器状态机表示 [3]），而非系统数据时，后果可能是灾难性的。

需要明确的是，我们将 CSV 解析（即 对 CSV 语法/格式本身的处理）交由外部工具完成。这里我们介绍一个 VDM CSV 库，用于捕获正在导入 VDM 的 CSV 数据的各个方面，使用户能够像对待 VDM 数据类型一样，为 CSV 数据引入规约。对于系统架构 CSV 文件，VDMJ 的

[1] https://unos.org/data/[2] https://www.rfc-editor.org/rfc/rfc4180[3] https://commons.apache.org/proper/commons-csv/[4] https://bytescout.co m/blog/csv-format-history-advantages.html

CSVReader 或 [4,9,12] 中的此类定制工具更为合适；而对于涉及多种 CSV 方言的一般数据处理以及对 IO 速度的要求（例如 日志文件、模拟运行、等等），我们提出的 CSVLib 库则更为适合。

因此，我们认为，对 CSV 输入所表示的数据结构进行形式化刻画十分重要，同时还应具备足够的灵活性和多样性，以满足实际应用的需求。为此，我们选择了 VDM。至于 CSV 文件格式本身的形式化刻画，则交由形式文法[5] 和解析工具负责。

维也纳开发方法（VDM）已广泛应用于工业和学术领域，涵盖安全 [6, 7], 容错 [11], 医疗器械 [8], 等多个领域。我们通过一套工具和数学库[6]扩展了 VDM 规范的支持。该工作还集成在 VDM Visual Studio Code（VS Code）集成开发环境中。

我们决定不再使用标准的 VDM-CSV 库，因为它速度较慢，无法处理各种 CSV 方言，依赖 CSV 定位管理（例如需要知道自己在文件中的位置；一次读取一行 CSV），并且不会检查 CSV 数据本身。

本文介绍了 VDM 工具包近期扩展的功能，使其支持 CSV 格式解析、验证和打印。我们通过多种在实际使用 CSV 时常见的场景，展示了该功能的用法。

2 背景

CSV 中存在各种棘手的情况，容易导致意外错误 [1,10, 12]。对于依赖此格式的关键应用而言，这尤其成问题。因此，我们创建了一个形式化环境，以清晰、简洁地描述 CSV 问题，并根据不同类型的用户定义不变量验证结果，例如 CSV 单元格上的数据类型不变量，以及行数据或列数据上的一致性不变量，如列顺序或行数据冗余一致性（例如 体重、身高和身体质量指数信息必须彼此关联）。此外，还需要定义整个文件的不变量。CSV 文件中还经常包含隐式默认值或不一致对应关系。因此，我们希望借助 VDM 库（CSVLib）以形式化方式描述这些约束。我们将该库集成到了 VDM VSCode 扩展[7],和 VDMJ [2]中。

关于正式处理 CSV（及数据源），我们曾研究如何从描述 EMV 支付系统数据字典的 XSD 文件生成 VDM 文件 [4], ，也曾将类似工作应用于功能模拟单元标准 [8]。另一个例子是，将透析机 [9] 和脑起搏器 [12] 的控制系统有限状态机定义以 CSV 格式载入。这些应用会读取相应格式（CSV 或 XML）的数据，并生成定制的 VDM 模型，以体现隐含理解的设计决策。

[5] 有关形式化 CSV 语法的示例，请参阅 https://github.com/antlr/grammars-v4/tree/master/csv [6] https://github.com/leouk/VDM_Toolkit/ [7] https://github.com/overturetool/vdm-vscode [8] https://github.com/INTO-CPS-Association/FMI-VDM-Model

这些将 CSV 应用于 VDM 的做法与我们在此采用的方法不同。我们的方法侧重于在 VDM 数据类型中载入原始 CSV 数据，随后可通过动态不变量进一步检查各种结构一致性。与我们的 CSV 库（CSVLib.vdmsl）类似，VDMJ 标准库（CSV.vdmsl）中也定义了一个 CSV 库：它通过原生调用为用户提供一个简单的 CSV 解析器，并返回一系列通配符（?）类型。

3 设计原则

我们的 VDM CSV 库遵循四项核心设计原则：

1. **S**简洁：CSV 处理是一项普遍存在的任务，因此易于使用至关重要；2. **A**准确：CSV 输入错误会造成相当多的建模过程不准确；因此，我们希望提供一种解决方案，使多种预期验证方式都能实现；3. **F**快速：CSV 输入通常可能很大；因此，提供多样且计算效率高的 IO 解析方案十分重要 [9]；4. **E**有效：CSV 格式有多种变体 [10]；因此，能够容忍格式差异十分重要。

这些 **SAFE** 设计原则构成了整个库设计目标的基础。其架构分为三个部分：i）IO 原生调用；ii）多种 CSV 不变量检查机制，分别针对单元格、行间、列间以及整个文件；iii）一系列支持函数和操作，以便轻松使用 CSV 功能。

此外，我们承认，实际的 CSV 应用中充斥着错误和不准确之处。因此，我们还提供详尽的错误报告，并支持处理违反预期不变量的 CSV，同时向用户提供准确的信息。完成验证后，用户便可放心地访问 CSV 数据以用于目标用途，也可以将其（处理后或未经处理）打印到文件中。

4 库架构

接下来，我们将介绍实现设计原则的库架构（见图 1）。用户模型必须导入 CSVLib.vdmsl，该文件通过 VDM 原生函数调用提供 CSV-IO（例如，解析和打印）功能。VDM 原生函数是一种将 VDM 规范与底层 Java 实现连接起来的机制，用于实现所定义的每个 VDM 原生调用。例如，VDMJ 为部分三角函数、随机值生成等功能提供了原生调用。
CSVLib.vdmsl 提供多种功能，包括：

1. 设置：行注释、跳过空行，等等；2. 解析器：为提升速度而设计的实现、支持多种 CSV 格式，等等；3. 表头：强类型，并带有默认值；4. 不变式：检查单元格、跨行和列以及整个文件；5. 错误处理：定位到单元格的错误，并说明失败原因。

[9] https://github.com/uniVocity/csv-parsers-comparison [10] https://commons.apache.org/proper/commons-csv/

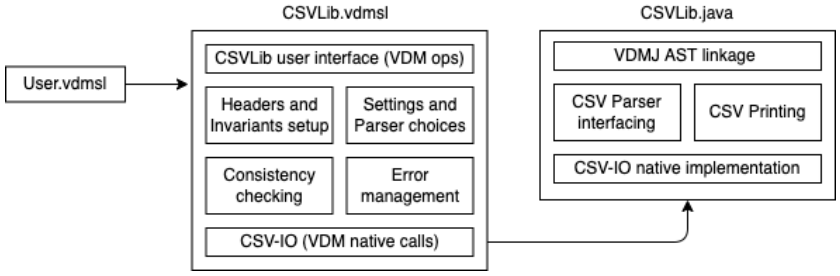

图 1：CSVLib.vdmsl 架构。

CSVLib 的使用

CSVLib 有两个入口：可以直接调用 CSV-IO 原生调用，也可以通过基于状态的规范中定义的操作来调用，这种规范描述了预期的库使用方式。前者能够最灵活地访问 CSV 功能，后者则提供最便捷的访问方式。例如，使用 CSV 库最简单的方法是调用以下操作：

```
loadSimpleHeadersCSV(路径, <Native>, [<Str>,<Int>,<Bool>],false);
```

这会使用给定的解析器接口实现来加载指定路径上的 CSV 文件。目前我们支持两种接口：我们自己的原生实现，以及我们能找到的速度最快的实现之一 uniVocity CSV[11]。其他实现也有可能，或已计划开发。

引号值列表包含预期 CSV 列的类型，而最后一个参数则表明处理过程是否采用严格模式。也就是说，任何未通过的不变式检查都会被容忍，并记录在状态中，以供用户进一步检查。反之，如果 strict 为 true，则不会容忍不变式检查失败，结果将是一个空 CSV。

CSV 表头

许多 CSV 文件都不包含表头。尽管如此，对于读取的每个值，它们仍应有类型正确性的概念（即 隐式类型）。在 CSVLib 中，表头必须定义类型和默认值。默认值对于行中出现逗号连续排列等常见情况十分重要。类型对于实现类型一致性和不变量检查很重要，而默认值则有助于推断空字段背后隐藏的意图，并确保默认值满足类型不变量。例如，下面的表头表示年龄是一个整数，默认值为 18：

```
mk_表头("年龄", <整数>, 18, 单元格_inv, col_i                nv)
```

CSVLib表头也可以包含描述和两个不变量：一个针对表头下方的每个单元格，另一个针对整个列。

[11] https://github.com/uniVocity/csv-parsers-comparison

CSV 不变量

我们有四种 CSV 不变量，由以下 VDM 函数类型定义：

```
CSV 单元格不变量  =(CSV 类型 * CSV 值->原因); CSV 行不变量     =
(表头 * 行->原因); CSV 列不变量     =(表头 * 转置行->原因); CSV 文
件不变量  =(表头 * 矩阵->原因);
```

不变量检查会返回一个原因：如果不变量得到满足，则返回 **nil**；否则返回一个非空字符串，说明失败原因。这使规范编写者能够对 CSV 文件中发生的错误进行精细控制。单元格不变量接收声明的表头类型和待检查的读取的单元格值。行不变量接收所有读取器和待检查的当前单元格行。列不变量接收列标题和待检查的当前单元格列，并将其视为转置后的行。最后，文件不变量定义了适用于整个 CSV 矩阵的属性。

某些不变量类型具有隐式检查，无论是否提供用户定义的不变量都会执行。对于单元格，这会根据相应的声明表头类型检查值；对于行，我们会隐式检查表头大小是否与行大小相符（例如 不允许行过短）；列或文件不变量则没有隐式检查。

对于上面的年龄表头，用户可以定义 cell_inv（L1–3）来强制执行年龄限制，而 col_inv（L5）则可以强制确保年龄具有唯一性，具体如下：

```
（lambda -: CSV 类型, v: CSV 值 &if v < 18 then "低于最小年龄" elseif v > 65 then "高
于最大年龄" elsenil）；（lambda h: 表头, c: 转置行 &if card elems c  <> len cthen "
不允许存在重复的年龄" else nil）；列表 1.1：单元格和列不变量定义
```

行不变量可用于检查行内单元格之间的依赖关系和冗余关系。如果 CSV 文件中有三个额外的 <浮点数> 表头，分别表示体重、身高和身体质量指数（BMI），则可以进行一些有趣的检查。例如，给定年龄类型范围（{18...65}），可以推断出预期的最小身高（即 行依赖不变量）；而给定 BMI 的公式（ $\frac{height}{weight^2}$ ），则可以计算其单元格值（即 行冗余不变量）。

文件不变量作用于整个数据集。当需要在多行和多列之间关联某些属性时，这可能会很有用。例如，导入年末财务记录时，可以针对每一行计算个人税款，并将其总和与某个先前确定的值进行关联。

CSV VDM 原生调用

CSVLib 有四个原生调用：三个函数用于实现文件状态、CSV 读写；另一个操作用于实现底层 IO（非 CSV）错误。

```
文件_状态 : 路径 ->  文件状态; csv_读取_数据 : 路径 * CSV 解析器 *CSV 设置
*表头 ->布尔值 * 错误 *数据; csv_写入_数据: 路径 * 数据 ->  布尔值; l
astError : ()  ==>原因;
```

csv_read_data 使用 Java CSV 解析器实现了该链接。它需要有效的路径、已知的解析器、设置和有效的表头；并返回一个三元组。在结果中，如果成功标志为 **false**，则表示发生了某种意外的 Java 错误，可以通过调用 lastError() 检查；三元组的另外两部分（分别表示短行错误和数据）为空。否则，如果成功标志为 **true** 且错误列表不为空，则表示 CSV 解析发现了短行（例如 行的长度小于表头的长度），其中包含此类错误在 CSV 文件中发生位置的详细信息。如果成功标志为 **true**，则结果数据会载入 CSV 数据，但不会进行不变式检查。这样便可以载入无效数据，随后对其进行处理，并向用户说明错误原因。最后，如果成功标志为 **true** 且错误列表不为空，则数据中不会包含已发现的短行。

CSV Java 原生实现

VDM 原生调用的实现在 Java 中完成。CSVLib.java 文件实现了 VDM 链接，以及各种专门用于处理 VDMJ 值 AST 的方法。对于需要在 Java 中处理 VDM 原生方法调用实现里的 VDMJ 值的其他库而言，这些方法可能会很有用。例如，我们为 Java 中的值 AST 添加了 VDM 记录更新所对应的 VDM mu 表达式。这使得在 Java 中进行 VDM 记录构造和更新的过程与直接在 VDM 中操作非常相似。例如，上文的 loadSimpleHeadersCSV 调用没有提供表头名称。该名称将从 CSV 本身获取，并且必须以类似 mu(header, name ↦ "年龄") 的方式更新表头名称字段。在 Java 中，这通过以下方式实现：

```
命名表头于 I  =  ValueFactoryHelper.muRecord(headerAtI,
ValueFactoryHelper.mkFieldMap( Arrays.asList(表头_字段_名称),
ValueFactoryHelper.mkValueList(
ValueFactoryHelper.mkString(nameStr)), Arrays.asList(true), ctx ),
ctx);
```

ValueFactoryHelper 类提供了许多实用的 VDM 值 AST 支持方法。上面的方法会使用给定的值更新指定字段名的记录。这说明，其他需要执行复杂 VDM 值 AST 操作的库也可以使用 CSVLib 库。

最后，CSVLib.java 委托给一个独立的 CSVParser Java 接口，该接口提供本地调用实现所需的必要服务。这对于将 VDMJ 值 AST 处理与底层 CSV-IO 解析器实现分离开来非常重要。这样一来，使用其他 CSV 解析器扩展该库就变得相对简单：用户可以使用自己偏好的 CSV 解析器实现该接口，并将相应的 JAR 文件添加到 VDM-VSCode 类路径中。

```
公共接口 CSV 解析器 {公共 迭代器<字符串[]> 解析CSV(最终读取器 reader)
抛出 输入输出异常;公共 字符串 lastError();公共 void 清除();公共 CSV 设置
getSettings(); }
```

任何底层实现只需返回输入流的 Java 迭代器视图，CSVLib.java 即可按照此处所述的设计原则工作。其他方法的含义不言自明。我们优先使用 Java Reader 作为输入类型，以便将文件编码问题纳入考虑。

将 VDM 原生调用的 Java 实现与其他 Java 代码分离非常重要，因为在 VDM 规范旁调试 VDM 原生调用的设置相对棘手。如今，这些考量和难题已经得到解决，其他需要实现 VDM 原生调用的 VDM 库开发者也可以复用相关方案。实际上，该设置使库开发者能够使用 VDMJ 的命令行调试器处理 VDM 库规范调试，同时使用 VSCode Java 调试器处理 Java 对 VDM 原生库调用的实现。

CSV 数据和错误

CSV 数据类型包含 CSV 设置、表头，以及由值序列组成的序列形式的数据矩阵。在 CSVLib 中，当某个不变式检查失败时，就会发生错误。CSV 错误记录单元格位置（即行和列），并附带一段非空说明，解释错误发生的原因。

```
数据 :: 设置: CSV 设置 表头: 表头 矩阵: 矩阵; 错误 :: 行号: nat1列号: nat1 原因: 原因;
```

成功调用 csv_read_data 后，用户可以通过调用函数检查哪些不变量遭到了违反

```
csv_invariants_failed: 数据 -> 一组 错误
```

它会返回错误集合，其中包含位置（行/列）以及不变量失效的原因。

CSV 状态和操作

CSV VDM 原生调用和错误处理，通过上述函数和类型即可灵活、便捷地使用。不过，其表现力也可能导致规范变得复杂。为避免给最终用户带来这种情况，我们提供了一个基于状态的接口，其中的操作可用于访问 CSV 数据矩阵、发现的任何错误，以及其他（IO）错误或可用性错误。

```
状态CSV 的
    file : [路径] 解析器: CSV 解析器 ferr: 原因
```

```
    strict: 布尔值 pos : 错误集合 data: 数据 不变式mk_CSV(file, -, -,
strict, pos, data)  ==(file  <>  空集  =>file_status(file)  =  <有效>) 且
(strict  =>csv_不变式_失败(data)  =pos  ={})...
```

CSV 状态包含 CSV 文件路径和解析器类型、底层错误原因（输入输出错误及其他非 CSV 错误）、严格的不变式验证、不变式错误的位置以及 CSV 数据矩阵。我们展示了部分状态不变式，它们规定非空文件状态必须有效（例如，文件存在且不是目录等），并且严格模式下的 CSV 不容许任何错误（包括短行输入输出错误）。这些状态和操作有效地展示了如何使用 VDM 原生调用。

最终用户有三个操作。上文的 loadSimpleHeadersCSV 操作示例调用 loadCSV，并传入适当创建的表头。loadCSV 操作会确保文件路径有效，（重新）设置所有相关状态参数，并加载 CSV 数据。数据加载遵循针对 VDM 原生调用 csv_read_data 所说明的逻辑，在各种情况下会向用户提供描述性的错误消息和/或信息消息。在此阶段，CSV 数据和不变式检查错误都可作为状态的一部分供检查和进一步处理。

```
loadSimpleHeadersCSV: 路径 * CSV 解析器 * seq1 of  CSV 类型 * bool   ==>  (); l
oadCSV : 路径 * CSV 解析器 * CSV 设置 * 表头 * bool   ==>  (); printCSV : 路径  ==>();
```

最后，操作 printCSV 会将已加载的 CSV 输出到指定的文件路径。这要求加载已成功完成，并且指定的路径与 CSV 文件路径本身不同，以避免数据覆盖；同时，相应的 CSV 文件将会被创建（或覆盖）。

5 评估

高性能是我们的 SAFE-CSV 库的核心设计原则。为证明这一点，我们使用了多个 CSV 文件，其行数从 100 到 35,000 不等，并将我们的库与 VDM 中的标准 CSV 库进行了性能比较。

标准 CSV 库在可接受的 CSV 输入格式种类方面功能有限，其 IO 机制较为底层（即 逐行读取），并且不具备数据验证能力。性能测试的范围仅限于标准 CSV 库所支持的最大功能子集。SAFE-CSV 提供了比 VDM CSV 更丰富的数据验证方式；不过，为了在性能测试中保持两个库之间的可比性，本节未使用这些功能。

由于 SAFE-CSV 还能够更改库所使用的底层解析器，因此在本项工作中，使用 Native 解析器和速度更快的 Univocity 解析器对 SAFE-CSV 的性能进行了测试[12]。

图 2 展示了使用相同输入数据时各库的性能。SAFE-CSV 解析器的表现显著优于标准库，加速比为

[12] https://github.com/uniVocity/csv-parsers-comparison

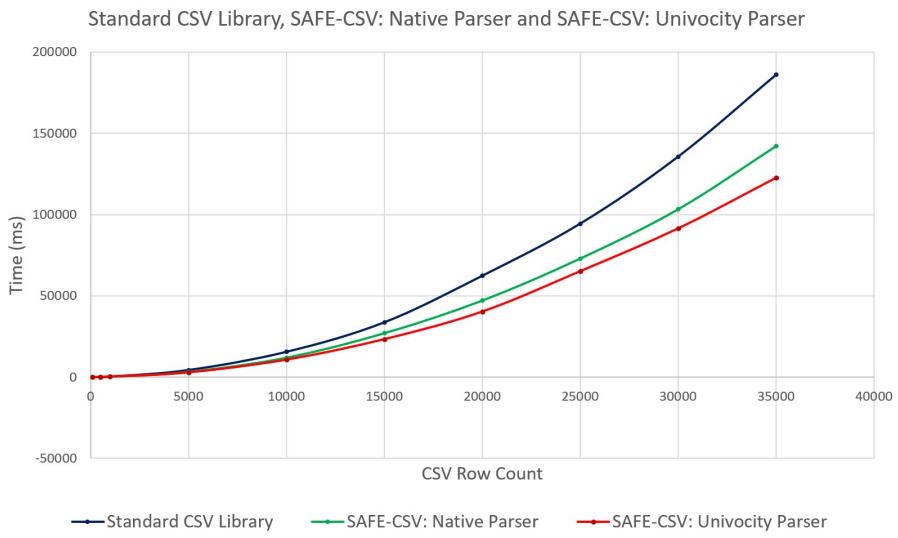

图 2：标准 CSV 库和 SAFE-CSV 库（使用 Native 和 Univocity 解析器）的性能比较

～ 1.5 倍。SAFE-CSV 库所采用的底层解析器会影响性能，因为 Univocity 解析器的表现甚至优于 Native 解析器。对于更大的数据集，Univocity 解析器的平均加速比也高于 Native 解析器。

6 个示例

接下来，我们通过几个示例说明如何使用我们的库。首先，用户必须提供有关 CSV 表头的信息。在我们的示例中，我们有一个姓名列表，以及每个人对应的年龄、体重（以 kg 为单位）、身高（以 cm 为单位）和身体质量指数。然后，我们创建相应的 VDM 表头：

```
示例 _HEADERS : 表头 =
[mk_表头("Name" , <字符串> , “名称”, nil, COL_INV_UNIQUE_NAME),
 mk_表头("年龄" , <整数>, MIN_AGE, CELL_INV_年龄,空),
 mk_表头("体重（千克）", <浮点数> , 最小_体重_千克, 单元格_逆_体重,nil),
 mk_表头("身高（厘米）", <浮点数> , 最小_身高_厘米, 单元格_逆_身高,nil),
 mk_表头("身体质量指数" , <浮点数> , 最小值_身体质量指数, 单元格_反转_身体质量指数, 空值) ];
```

提供了默认值，用于测试 CSV 逗号序列（例如，没有任何值的 CSV 单元格），并展示单元格不变量和列不变量示例。这些默认值用于确保隐式的行不变量检查：行大小必须与表头大小一致。年龄、体重、身高的单元格不变量以及唯一性的列不变量，与上文所示的类似（参见列表 1.1）。

接下来，我们要定义一个行不变量，用于检查冗余 BMI 字段是否与身高和体重一致。可以说，CSV 中不应包含 BMI 字段。尽管如此，许多 CSV 文件确实包含这类冗余信息，而且这些信息与预期值往往并不准确。BMI 检查行不变量定义如下：

```
(lambda h: 表头0, r: 行 &iflen h < 5 then "无效的 BMI 表头"else(let
bmi: real = calculated_bmi(r) inif approx_eq(r(5), bmi, PRECISION)
thennilelse "给定 CSV 体重和身高对应的无效 BMI"));
```

对于给定的行，用户定义函数 calculate_bmi 使用该行中的身高和体重值计算 BMI，而 r(5) 获取 CSV BMI 值，该值需要与计算结果近似相等。这里的近似比较会考虑特定的精度位数，以比较单元格值和计算值。

CSVLib.vdmsl 和 CSVExample.vdmsl[13] 中提供了其他示例。这些示例展示了使用不同 IO 解析器进行 CSV 解析、CSV 打印、字符串单元格中的转义引号（例如 跨越多行的字符串）、默认值（例如 CSV 行中的逗号序列）、短行（例如 小于预期表头的行）、各种类型的不变量违反（例如 隐式不变量违反和用户定义的不变量违反）等。

CSV（IO）错误：忽略来自“CSVExample.csv”的 1 行短行（1，5）：“CSV 行 1 的长度不足以匹配表头：预期 5 列，实际发现 4 列” “CSVExample.csv”的 CSV 不变量检查失败：14 个单元格存在 CSV 不变量失败：.....（3，2）：“单元格不变量无效：低于最小年龄”（4，1）：“列不变量无效：姓名重复”（4，2）：“单元格不变量无效：超过最大年龄”（4，3）：“单元格不变量无效：超过最大体重”（4，5）：“单元格不变量无效：超过最大身体质量指数”.....

由于导入问题而生成的输出会以行/列元组的形式显示违反规则的单元格，以帮助用户识别并修正问题。由自定义不变式导致的错误输出会提供用户定义的消息，以及不变式的类型（单元格、行、列、文件）。对于短行等隐式失败，则会提供标准化输出。

7 结果与讨论

在本文中，我们介绍了一个在 VDM 中正式定义的 CSV 库，该库遵循我们的**SAFE** 设计原则（见第 3 节）。该库架构（第 4 节）Simple,

[13] https://github.com/leouk/VDM_Toolkit/tree/main/plugins/vdmlib

因为它以分层方式提供功能访问，并最终提供近乎简单的用户界面访问点。鉴于存在多种**A**ccurate 用户定义的不变量，以及其他结构和数据验证检查（例如检测短行，以及检查单元格值类型相对于已声明的表头是否具有一致性），它同样**A**ccurate。由于其架构布局支持即插即用不同的 CSV-IO 解析方式，因此它**F**ast。最后，鉴于它结合了速度、易用性、围绕 CSV 格式处理和 CSV 设置的多种功能，以及错误处理等能力，它是**E**ffective。

与标准 VDM CSV 库相比，我们的 SAFE-CSV 性能更佳，速度提升超过 50%。虽然在性能分析中没有使用 SAFE-CSV 库更健壮的不变量系统，但其执行基础设施和内置类型检查仍贯穿整个库。进行不变量检查对性能的影响微乎其微。这与用户在标准 VDM CSV 库的 IO 功能之上自行编写此类 VDM 检查（check inv）的性能相当。

这项工作的一个关键动机，是需要更健壮的数据导入功能，以分析器官保存医疗设备生成的数据。标准 VDM CSV 库能够处理的 CSV 格式过于有限。此外，它不具备数据验证功能，而这对于安全关键应用是必不可少的。

未来工作。 未来，我们希望集成 VDM 原生库调用的调试环境，以便使用 VDM VSCode DAP 协议 [13]。此外，在 CSV 格式方面，我们还希望支持嵌套 CSV 格式、更多 CSV 类型以及每个文件包含多个 CSV 表头等。这样，VDM 库开发者便能通过 VSCode 调试 VDM 库规范及其示例，同时进行 Java 调试。

致谢 我们感谢 NCSC 对更安全支付协议开发的资助。我们非常感谢与 Nick Battle 就 VDMJ 原生调用设置展开的富有成效的讨论。

参考文献

1. Abba，A.H.，Hassan，M.：CSV 验证系统的设计与实现。载于：第3届信息技术应用国际会议论文集。第111–116页。ICAIT'2018，计算机协会，纽约，纽约州，美国（2018），https://doi.org/10.1145/3274856.3274879 2. Battle，N.：VDMJ 用户指南。技术报告，富士通服务有限公司，英国（2009） 3. D.Harrison，M.，Freitas，L.，Drinnan，M.，Campos，J.C.，Masci，P.，di Maria，C.，Whitaker，M.：新型透析机软件组件安全分析中的形式化技术。《计算机编程科学》175，17–34（2019年2月） 4. Freitas，L.：大规模 VDM：EMV 第二代内核的建模。载于：第 21 届巴西形式化方法研讨会，SBMF。第26–30页（2018年） 5. ISO：ISO/IEC 21778 信息技术——JSON 数据交换语法。技术报告，ISO（2017年） 6. Kulik，T.，Macedo，H.D.，Talasila，P.，Larsen，P.G.：VDM 中 HUBCAP 沙箱架构的建模：安全性研究。载于：Fitzgerald，J.S.，Oda，T.，Macedo，H.D.（编）：第18届国际 Overture 研讨会论文集。第20–34页。Overture（2020年12月）

7. Kulik，T.，Talasila，P.，Greco，P.，Veneziano，G.，Marguglio，A.，Sutton，L.F.，Larsen，P.G.，Macedo，H.D.：扩展 hubcap 沙箱的形式安全分析。载于：Macedo，H.D.，Thule，C.，Pierce，K.（编）：第19届国际 Overture 研讨会论文集。Overture（2021年10月） 8. Macedo，H.D.，Larsen，P.G.，Fitzgerald，J.：使用 VDM 对心脏起搏系统的分布式实时模型进行增量开发。载于：Cuellar，J.，Maibaum，T.，Sere，K.（编）：FM 2008：形式化方法，第15届国际形式化方法研讨会。《计算机科学讲义》，第5014卷，第181–197页。Springer-Verlag（2008年） 9. Martinkute，E.：新型婴儿透析器的形式化规约。硕士论文，纽卡斯尔大学计算科学学院，英国（2016年8月） 10. Nazario，A.C.：制药行业数据完整性实施策略中的计算机系统验证 CSV。硕士论文，工业与系统工程系（2018年） 11. Nilsson，R.，Lausdahl，K.，Macedo，H.D.，Larsen，P.G.：将工业案例研究从 VDM++ 转换为 VDM-SL。载于：Pierce，K.，Verhoef，M.（编）：第16届 Overture 研讨会。第107 1524 12 122 2018 – 页。纽卡斯尔大学，计算机学院，牛津（7月），TR- 。Pollitt，A.：在 eCv 中验证 CANDO 项目的 Optrode 命令接口。硕士论文，纽卡斯尔大学计算科学学院，英国（2018年8月） 13. Rask，J.K.，Madsen，F.P.，Battle，N.，Freitas，L.，Macedo，H.D.，Larsen，P.G.：Visual Studio Code 中的高级 VDM 支持。载于：第20届国际 Overture 研讨会论文集。第35–50页（2022年）