Specification-based CSV Support in VDM

The 21st Overture Workshop
10th March 2023

Leo Freitas
Aaron John Buhagiar

1. School of Computing, Newcastle University
2. Translational and Clinical Research Institute, Newcastle University
Introduction

• Standard (RFC4180) data exchange format
• Ubiquitous use in a myriad of domains
  • Data science applications
  • Medical data and embedded
  • Payment systems
• Many variations and versions are used
• VDM standard CSV has limitations
• We created an improved CSV library with:
  • Data validation formal specification
  • Ease-of-use
  • High performance
• Distributed with the VDM Toolkit [1]

VDM standard CSV Library

- Slow and error-prone (e.g. low-level IO)
- Line-by-line parsing in VDM
- Limited support for various CSV formats
- Only hard-coded IO native calls available
  - Limits CSV performance and formats variety
- Imported data is of a wildcard (?) type
  - Users have to further introspect meaning
- Different from CSVReader
  - CSV as user data not structural information
Design Principles

Simple
Accurate
Fast
Effective
Types and Parsers

- CSV Lib offers basic types for column typing
  
  ```
  CSVType = <Integer> | <Float> | <String> | <Boolean>
  ```

  ```
  CSVValue = int | real | String | bool
  ```

- Multiple parses are available
  
  ```
  CSVParser = <Native> | <Univocity> | <Apache> | <OpenCSV> | <QuirkCSV>
  ```
Simple: Ease of Use

• Accessible entry points that abstract from IO native calls
  1) Out-of-the-box setup
  2) Configurable (e.g. CSV settings)
• Allows direct native calls for better extensibility and control (3)
• Data validation can be formally specified
Simple: Configurable Setup

- CSV entries strong typing defined through semantic headers as:
  - Column Name
  - Datatype
  - Default Value
  - Cell invariant
  - Column Invariant

- CSV settings define expected file properties as:
  - Presence of blank lines
  - Existence of a header row
  - Comment string

```csharp
EXAMPLE_DATA_HEADERS: Headers =
  [mk_Header0("Name", <String>, "Joe", nil, COL_INVARIANT_UNIQUE_NAME),
   mk_Header0("Age", <Integer>, MIN_AGE, CELL_INVARIANT_AGE, nil),
   mk_Header0("Weight(Kg)", <Float>, MIN_WEIGHT_KG, CELL_INVARIANT_WEIGHT, nil),
   mk_Header0("Height(cm)", <Float>, MIN_HEIGHT_CM, CELL_INVARIANT_HEIGHT, nil),
   mk_Header0("BMI", <Float>, MIN_BMI, CELL_INVARIANT_BMI, nil)]
```
Simple: Reporting

- Simple and descriptive reporting
  - Short rows (i.e. not enough columns)
  - Declared type violations (e.g. string for nat)
  - User defined invariant violations
- Provides cell locations for correction
- Striving to have a strongly-typed CSV
Accurate

- Checks on imported data
  - Short rows
  - Declared header type validation

- User defined invariants
  1. Cell invariants
  2. Column invariants
  3. Row invariants
  4. File Invariants

- Lambda abstractions capture invariants as record fields

- Invariants return a **Reason**

```plaintext
1. CSVCellInv = (CSVType * CSVValue -> Reason);
2. CSVColInv = (Header0 * TransposedRow -> Reason);
3. CSVRowInv = (Headers0 * Row -> Reason);
4. CSVFileInv = (Headers0 * Matrix -> Reason);
```
Accurate: Cell Invariant

• Invariants that act upon cells directly
• Allows for cell validation
• Examples
  • Upper/lower bounds
  • Cell text validation
  • Specific value enforcement
  • etc…

```plaintext
CELL_INVARIANT_AGE: CSVCellInv =
(lambda :: CSVType, v :: CSVValue &
  if v < MIN_AGE then
    "below minimal age"
  else if v > MAX_AGE then
    "above maximal age"
  else
    nil);

CELL_INVARIANT_WEIGHT: CSVCellInv =
(lambda :: CSVType, v :: CSVValue &
  if v < MIN_WEIGHT_KG then
    "below minimal weight"
  else if v > MAX_WEIGHT_KG then
    "above maximal weight"
  else
    nil);
```
Accurate: Column Invariant

- Column-wide invariant
- Allows for validation per header across rows
- Example
  - Uniqueness
  - Dependence
Accurate: Row Invariant

• Row-wide invariant
• Allows for validation per row across all headers

• Examples
  • Consistency
  • Dependence
  • Redundance

```
ROW_INVARIANT_BMI_CONSISTENCY: CSVRowInv =
  (lambda h: Headers0, r: Row &
   if BMI_ROW_INDEX > len h then
     "invalid BMI header"
   else
     (let bmi: real = calculated_bmi(r) in
       if floor(bmi) = floor(r(BMI_ROW_INDEX)) then
         nil
       else
         "invalid BMI for given CSV weight and height: expected " ^
         val2seq_of_char[real](bmi) ^"; found " ^ val2seq_of_char[real](r(BMI_ROW_INDEX)))))
```
Accurate: File Invariant

- Invariant across all CSV cells
- Example
  - Dependence
  - Redundancy

```haskell
FILE_INVARIANT_CONSISTENCY: CSVFileInv =
(lambda h: Headers0, m: Matrix &
  if NAME_ROW_INDEX > len h then
    "invalid name header"
  else if AGE_ROW_INDEX > len h then
    "invalid age header"
  else
    (let
      dups : set of nat = { j | i, j in set inds m.cells & i <> j
        and
        check_name_age_uniqueness(m.cells(i), m.cells(j))
      } in
        if (dups <> {}) then
          "duplicate persons found in row(s) " ^ val2seq_of_char[set of nat](dups)
        else
          nil
    )
```

13
Fast

Standard CSV Library, SAFE-Csv: Native Parser and SAFE-Csv: Univocity Parser

Time (ms)

0 5000 10000 15000 20000 25000 30000 35000 40000
CSV Row Count

-50000 0 50000 100000 150000 200000

- Standard CSV Library
- SAFE-Csv: Native Parser
- SAFE-Csv: Univocity Parser
Effective

- Multiple CSV format variants
  - Better tolerance to CSV format variability (e.g. CSV UTF8, MS-DOS, Mac, etc.)
  - Delegate CSV format type to parsers (e.g. formalisation of CSV format itself)
- Ease of use: use of VDM state and operations as entry points
- Improved validation, error handling and reporting
- Faster performance through multiple CSV parsers
Effective

• Tested on multiple CSV format variants
  • CSV UTF8, MS-DOS and Mac

• Applied to multiple domains
  • ScubaTx organ preservation device medical device logs
  • UNOS (United Network for Organ Sharing) lung transplant history logs
  • EMV payment system transaction logs
  • Personalised medicine DSL “certificate of treatment”
  • Neonatal dialyser finite state machine control system definitions
  • Etc.
Library Architecture

User.vdmsl

CSVLib.vdmsl
- CSVLib user interface (VDM ops)
- Headers and Invariants setup
- Consistency checking
- CSV-IO (VDM native calls)
- Settings and Parser choices
- Error management

CSVLib.java
- VDMJ AST linkage
- CSV Parser interfacing
- CSV Printing
- CSV-IO native implementation
Future Work

• Implementation of debugging environment
• Improved variety of CSV formats
  • Nested CSVs
  • CSV Settings
  • More CSV data types
  • Multiple CSV headers per file
Thanks for Listening