CSV Exporter Documentation

A component of the NIST, SJU, NMSU CPS System

Written By Matt Bundas (bundasma@nmsu.edu)

Table of Contents

Summary

The CSV Exporter provides the connection between two components of the CPS Ontology System, the ASP Solver and Tableau visualization. It does this by processing output from the ASP Solver and constructing a CSV file which contains all of the information in the correct format for the Tableau visualization to do its thing. The CSV Exporter is called inside the ASP Solver through the use of a button “Export CSV” which can be used after a query has been executed. The CSV Exporter opens a file containing the ASP output, parses the line containing relevant information using the Parse python package, constructs a pseudo ontology tree behind the scenes, then processes the tree to write the correct information to the CSV file. The new CSV file can be used in Tableau by connecting the necessary Tableau workbook to the CSV as a data source, or refreshing the data source if it is already connected.

Requirements

This component ultimately only serves a purpose within the CPS Ontology System, as its purpose is to bridge the gap between the ASP Solver and Tableau visualization. The CSV Exporter shares requirements with both the ASP Solver and Tableau visualization, so please see the documentation for these components. Requirements just specific to the ASP Solver are a working python version and the following packages:

parse - https://pypi.org/project/parse/

argparse - https://docs.python.org/3/library/argparse.html
csv (comes with most python installations) - https://docs.python.org/3/library/csv.html
os (comes with most python installations) - https://docs.python.org/3/library/os.html

Currently all files/scripts related to the system can be found here:
https://github.com/thanhnh-infinity/Research_CPS

Files to run the CSV Exporter standalone are in CSVExport/product. To run the exporter standalone, you will need just exportASP.py, and an ASP Output file to use as input for the CSV exporter, such as the ones found in /product/SR01_inputs/.

Usage

Although the CSV Exporter is contained in its own python script, it is embedded in the ASP Solver, so to use the CSV Exporter to its full extent, the ASP Solver must be used, and to make use of the CSV Exporter output, Tableau must be used. However, the CSV Exporter can be used on its own to convert an existing ASP output file into a CSV file as well.

Use Within CPS Ontology System

See the ASP Solver and Tableau documentation for a more comprehensive usage of these components. In a simplified way, the ASP Solver is used after a query is executed in the ASP Solver, and is called by pressing the “Export CSV” button on the interface. This will call the CSV Exporter python script, and export the ASP Solver output to a csv file which contains all of the information the Tableau visualization needs to do its thing. The csv file is exported to the Research_CPS/Integration/CSV folder, and is saved as out.csv, although this will likely be changed in the future to allow for more customization. To use the csv file you just outputted, launch the desired Tableau workbook containing an ontology visualization, and assign the data source to be the new out.csv file, or if it’s already the data source, refresh it to use the newly updated out.csv file. To refresh navigate to the Data Source tab, and in the top left of the window there is a refresh button. To see the visualization using the new out.csv file, navigate to the desired graph using the tabs at the bottom.

Use As Standalone Component

At the moment, the CSV Exporter is used as a standalone component by running exportASP.py in terminal/command prompt with two arguments, the path to the desired input file, and the path to the desired output file (including file extensions).

For example, running:

“python exportASP.py ./SR01_inputs/use_case_1_elevator_after_cyberattack.txt ./SR01_outputs/test.csv/”, would input use_case_1_elevator_after_cyberattack.txt as the ASP output, and export it to test.csv in CSV format.

Design

The CSV Exporter is written in python and makes use of a package called “parse” to make it easier to parse the ASP output. The driving class is the ASPExporter class, which contains the functions and data members to make the CSV exportation possible. This class makes use of another class, csvNode where each individual of csvNode is an individual in the ontology that we care about, like its name, type, lineage, level, sum etc. ASPExporter uses “parse” to parse the correct line in the ASP file and extract all of the information it needs to create a pseudo-tree to represent the ontology. It then processes this tree and writes to a csv file the necessary information needed for the Tableau visualization, which is largely stored in the tree as a whole and individual csvNodes.

End CSV File Content

The CSV file content is all of the information required to create the Tableau visualization. Its requirements are specified by SJU and the NMSU team tries to provide a CSVExporter which fits their needs. The Tableau visualization is quite complex behind the scenes, so it needs several data columns, and is in a format such that Tableau can create a sunburst plot. The information of the entire tree is written first with Pad and To Pad of “1”, then written again below it with Pad and To Pad value of “203”.

Level - Horizontal level the individual exists in the tree, where the uppermost aspects begin at level 1, and its children are at increasing levels.

Sum - The node’s footprint in the sunburst plot. Essentially the number of descendants, including itself, which are leaf nodes in the tree.

Satisfied - “Satisfied” if concern is satisfied “Unsatisfied” if concern is not satisfied.

Aspect Tree - Aspect tree which the individual is a member of.

Pad - Used by Tableau as a flag to process the line correctly. One of either “1”, or “203.

To Pad - Used by Tableau as a flag to process the line correctly. One of either “1”, or “203.

Other csvNode Data Members

Each csvNode represents an individual in the ontology tree, and stores all relevant information about the individual as it exists in the tree, including its type, parents, children, level etc.

Name - Name of individual
Type - Type of individual, aspect, concern, property etc.
Parents - Before splitting nodes with multiple parents into separate nodes, is a list of csvNodes containing all of the node's parents, afterwards just one of them.
Children - List of csvNodes containing all children of the node as it exists in the tree.
Lineage - List of csvNodes containing all nodes in its family tree starting with itself upwards. Used to create csv in desired format for Tableau
Leaf - True/False, whether the node is a leaf or not as it exists in the tree. If it has no children, the node is a leaf node.

Parsing ASP Output

The first step in exporting the CSV file, is to open and parse the ASP output file to create the pseudo-tree. All of the information needed is provided on a single line (fifth) in the ASP txt file, which is hardcoded into the class. All individuals, relations, and satisfactions of concerns are encoded into this line, in the form of concern("cpsf:ConcernName"), subconcern("cpsf:ParentName", "cpsf:ChildName"), h(sat("cpsf:ConcernName"),Step#) etc. Each encoding has a specific pattern, so the ASPExporter makes use of the "parse" package to look for all occurrences of these patterns, and then process them accordingly. For example, to find concerns it looks for the pattern " concern(\"cpsf:{}\")", and returns what inside the {}, which we know to be the name of a concern, so then we can declare a csvNode with the specified name, and of type concern, and add it to a list to be processed more later. We do this for all patterns including aspects, subconcern relations, and satisfaction. By the end of parsing the line, all individuals will be instantiated as csvNodes, with satisfaction specified, and stored in their necessary lists. Relations will be stored just by name (not as csvNode) in lists to be processed later.

Assigning Attributes to Individuals

At this point, all individuals (concerns, aspects) have their own csvNode, with name, type and satisfaction specified. Relations (subconcern) are stored by name in list format (not csvNode). Now the ASPExporter can look at all of the individual csvNodes and all of the relations, create the structure of the tree by looking at the relations (setting up parent/child relationship), then eventually parse this tree to assign data members to store information in the csvNodes specific to what the Tableau visualization needs in its csv file, such as level, sum etc.

Assigning Parent/Child Relations

Assigning the parent/child relationship is done by the function addAllChildrenParents. This function looks at all of the relations stored ion allSubConcernRelations_str_str, which stores each relationship as a single tuple in the form (parent, child), so the function can look at each tuple, determine the parent/child by indexing, find the csvNode corresponding to the parent/child, and then assign the parent/child data member accordingly.

Splitting Up Nodes With Multiple Parents

After all parent/child relationships have been set up, nodes with multiple parents are split into separate nodes with only a single parent. This is to ensure that the node in the Tableau visualization shows as a child of each parent, not just a single parent. All nodes are looked at, if it finds a node with multiple parents, then for each extra parent, a new csvNode is created, with the just the additional parent as a parent, but identical attributes as the original csvNode otherwise.

Assigning Level

The level of a given node is found using a recursive function, called assignLevel. This function is called on each node, where before the function is called the node's level is assigned to 0. The function starts at the passed node, if it has a parent it moves to the parent node and adds 1 to the original passed node's sum, and does so until the node has no parent.

Assigning AspectTree

AspectTree is assigned using a recursive function called assignAspectTree. This function is called on each node, where it navigates up the tree until it finds a node with no parents, assumes it is an aspect, and assigns the aspectTree for the original node to be this found node.

Assigning Leaf

Leaf is assigned using a non-recursive function called assignLeaf, which is called on each node. If the node has 0 children, it sets Leaf to True, otherwise sets it to False.

Assigning Sum

The sum of a node is assigned using a recursive function called assignLeaf, which is called on each node. It is recursively called on each child, if it finds that the child is a leaf it adds one to the original calling node's sum, and halts.

Assigning Lineage

The lineage of a node is assigned using a recursive function, which is called on each node. It starts at the calling node, inserts itself into the lineage, then navigates up the tree, inserting each node, and stops when it finds a node with no parents (the top of the tree).

Writing Tree to CSV

At the point of writing the tree to a CSV file, all individuals have a csvNode with data members completely filled out, so the ASPExporter is ready to loop through all of these nodes and write their information to a CSV. What is written is determined by SJU, and is everything that the Tableau visualization needs to do its thing. Each row of the CSV is written one at a time, and makes use of a masterLine, which is an empty list with length according to how long each row in the CSV should be. When writing a line, a copy of the masterLine is created, filled with the necessary information, then added to a list of lists storing all rows to be outputted to the CSV.

maxLevel

To make Tableau's life easier, a maxLevel is set to be 10, ensuring that whatever tree Tableau is dealing with, it can't be more than 10 levels. In this way, the csv will have 10 columns holding the aspects/concern names no matter the actual height of the tree.

lineLength

lineLength is the length of each row in the csv. It is found by adding maxLevel and the number of fields to the right of it in the csv, like level, sum, etc. This will be used to help put information about the tree in the correct location in the csv no matter how big the actual ontology tree is.

masterLine

masterLine is used as a template for each line written to the csv. It is a list of length lineLength, where each element is an empty string "", so each element can be assigned to a new value, and if it doesn't then it just shows up as an empty cell in the csv.

Setting Fields (Column Headers)

To set the fields of the csv, a masterline is copied and then filled with the correct fields. The first element is set to "Aspect", since the highest level of the tree is an Aspect. After that, elements 2-maxLevel are set to be CPS_Concern_1, CPS_Concern_2… etc. These columns will hold the names of the aspects/concerns. After this, the headers for informational columns are inserted, based on lineLength.

Creating Data Rows

The format of the csv required for Tableau is such that each leaf node and each node in its lineage get their own row in the csv. In each row is the node's lineage in the leftmost columns, and its data/information like sum, level, etc in the rightmost columns. So to write this information to the csv, each leaf node is looped through, then each node in its lineage is looped through, and each of those nodes gets a masterLine, the masterLine is filled with the node's information, then appended to outputRows, which holds all of the rows to be written to the csv. This has to be done twice for the entire tree, first the entire tree where the “Pad” and “To Pad” columns are 1, and then another time where they are “203”.

The creation of the data rows is done by the function setOutputRows.

Writing to CSV

At this point, all rows have been created and are stored in outputRows, and column headers have been created, stored in outputFields. Writing this information to the CSV is done using the csv package, and handled by the function writeRows, which has an output file name as a parameter. A file with the given output name is opened, a csvwriter is created, the headers are written using the csvwriter, then the data rows are written.

Driver Function - exportCSV(input_str,output_str)

A function exportCSV exists to drive the entire process of writing the ASP output to a csv. This function takes two parameters, a file path to an existing ASP output txt file (including .txt), and a file path to the desired new csv file (including .csv). It instantiates an ASPExporter object instance, whose constructor handles reading the ASP output and creating and configuring the csvNodes. It then calls .doOutput on this object instance which writes the constructed tree to a csv as described above.

Main Function - Processing arguments from command line

The exportASP.py file supports use from the command line by including code necessary to facilitate a “main” function. It is called from the command line using “python exportASP.py input_filepath output_filepath”. A package, argparse is used to process these arguments as strings, and then call exportCSV, which handles exporting to CSV.

Connection to System

The CSV Exporter’s role in the system is to process output from the ASP Solver after a query, and export it as a CSV so it can be displayed in Tableau. It contains all information to draw the tree structure and satisfaction of individuals. The CSV Exporter is called in the ASP Solver after a query using a button. The csv output is used by Tableau as either a new Data source, or as a refreshed data source to show modifications in the ontology from the last visualization. The CSV Exporter plays a crucial role in connecting two larger components of the system without the user having to do much manual work, as they had to before the introduction of the CSV

Exporter. Without it, a user would have to manually edit the data source used by Tableau, instead of having it automated.