GENERATING SAMPLE INSTANCES SATISFYING TAXONOMY ASSERTIONS

Eugeniusz Tomaszewski
Business Analyst
FUJITSU

Frankfurt, 18 June

AGENDA

1. Project background
2. Assertion solving process - simplified
3. Challenges: variable dependency, multiple executions, more supported expressions (... and many other)
4. Assertion solving process - enhanced
5. Roadmap

Project background

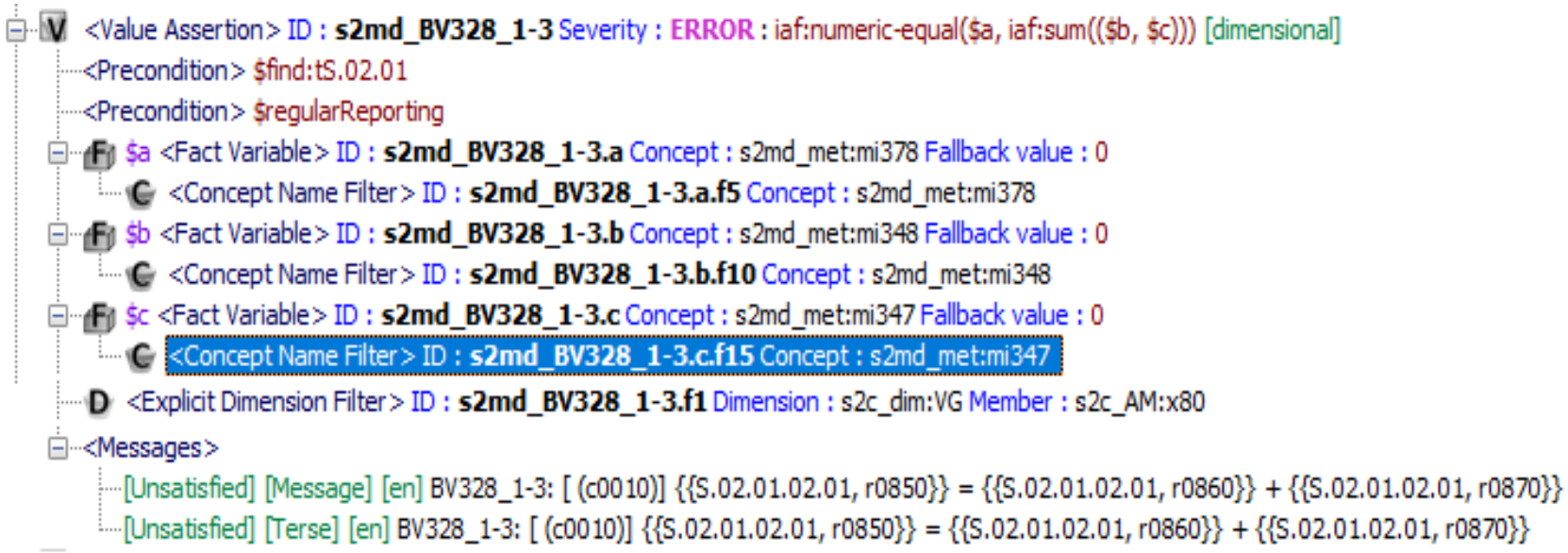
PROJECT BACKGROUND - PROBLEM DEFINITION

When a taxonomy contains a formula linkbase (i.e. a validation layer), then immediately several questions arise (of interest to a taxonomy developer and an end user alike):

- are the assertions correct from syntactical and semantical viewpoint?
- how to document the intended meaning of an assertion e.g. which reports or tables are impacted?
- how to figure out which facts are evaluated by an assertion?

PROJECT BACKGROUND - PROBLEM DEFINITION

Looking at an assertion's XBRL definition, it is hard to fully understand its meaning. And there can be hundreds of assertions defined in one entrypoint.



Still we can glean some basic understanding:

- there are 3 fact variables $a, $b and $c which impose constraints on which facts will take part in assertion's evaluation
- facts are constrained by their concept (and common dimension)
- expression being evaluated is (in effect) $a = sum($b, $c)

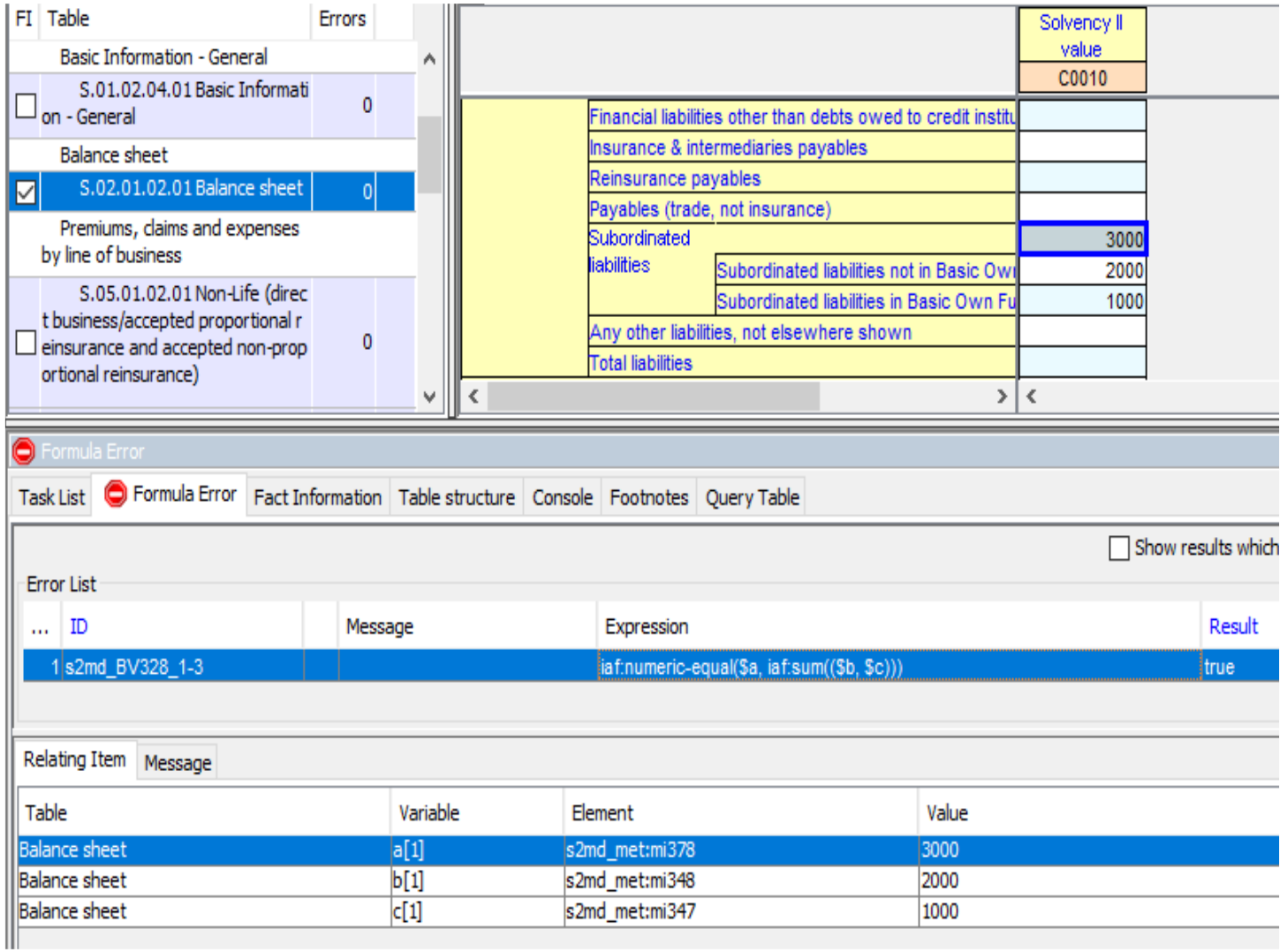
PROJECT BACKGROUND - PROPOSED SOLUTION

A solution is to provide sample correct data (XBRL facts), for each assertion or a group of assertions associated with an entrypoint. Then, a user can visualize the facts on a report.

Once data has been presented on a table (here: Balance sheet) it is immediately visible that BV328_1-3 just checks the expression:

Subordinated liabilites (SL) =

SL not in Basic Own Funds + SL in Basic Own Funds



BENEFITS OF XBRL ASSERTION SOLVER

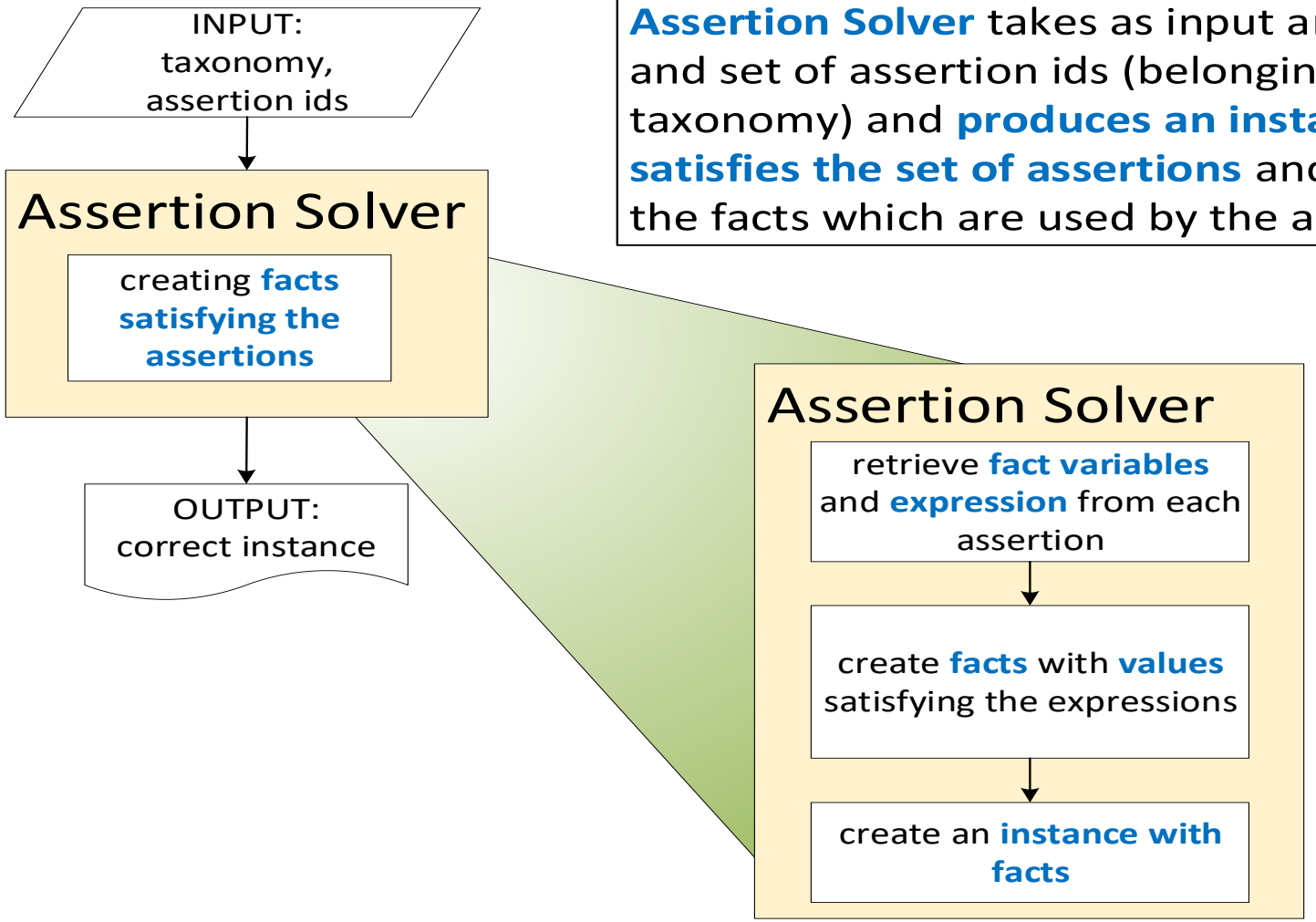
The idea of providing a tool (XBRL Assertion Solver) which can generate correct set of facts has been a recurring topic which we encountered at various conferences, projects or less formal conversations.

XBRL Assertion Solver generating instances based on assertions from a taxonomy can be used for a variety of purposes:

- providing illustrative examples of correct and incorrect instances
- formula linkbase quality assurance e.g.:
  - verifying whether for each assertion there exists a set of satisfying facts
  - verifying whether a set of assertions can be satisfied by a set of facts
- generating realistic test data for performance benchmarks of Formula processors

Assertion solving process - simplified

ASSERTION SOLVING PROCESS - OVERVIEW



Assertion Solver takes as input any taxonomy and set of assertion ids (belonging to the taxonomy) and **produces an instance which satisfies the set of assertions** and contains just the facts which are used by the assertions.

XBRL FACTS AND ASSERTIONS - A QUICK OVERVIEW

Before we will decompose the assertion solving process any further let's recall the basic characteristics of XBRL facts and assertions.

A simple numeric XBRL fact is a unit of reported information, composed of:
1/ value (and its accuracy)
2/ datapoint aspects: concept and dimensions
3/ other aspects*: period, entity and unit.

Aspect	Aspect Value
period	2018-01-31
entity	815600A60E71CFC3A230 .../iso/17442
unit	EUR

FACT = (EXTENDED) DATAPOINT + VALUE

Thus, the Assertion Solver, when constructing facts, needs to take care both of datapoints (which must satisfy aspect conditions on variables $a, $b and $c in case of BV328_1-3) and values (which must satisfy the expression, $a = sum($b, $c)).

$a => FACT_1

value	3000

Aspect	Aspect Value
concept	s2md_met:mi378
s2c_dim:VG	s2c_AM:x80

$b => FACT_2

value	1000

Aspect	Aspect Value
concept	s2md_met:mi348
s2c_dim:VG	s2c_AM:x80

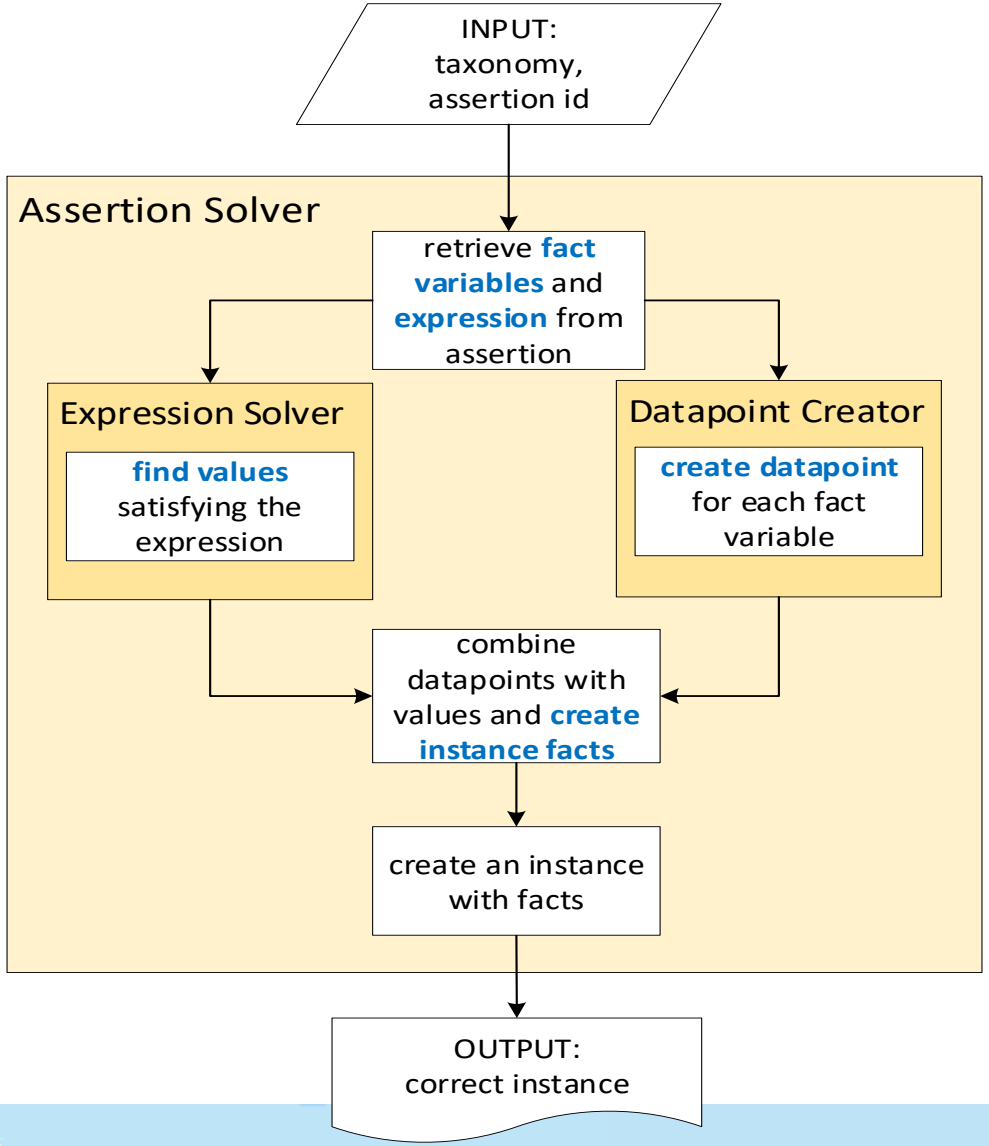
$c => FACT_3

value	2000

Aspect	Aspect Value
concept	s2md_met:mi347
s2c_dim:VG	s2c_AM:x80

* For simplification we assume that period, entity and unit aspects are identical to all facts and fixed part of any (extended) datapoint

ASSERTION SOLVING PROCESS - OVERVIEW

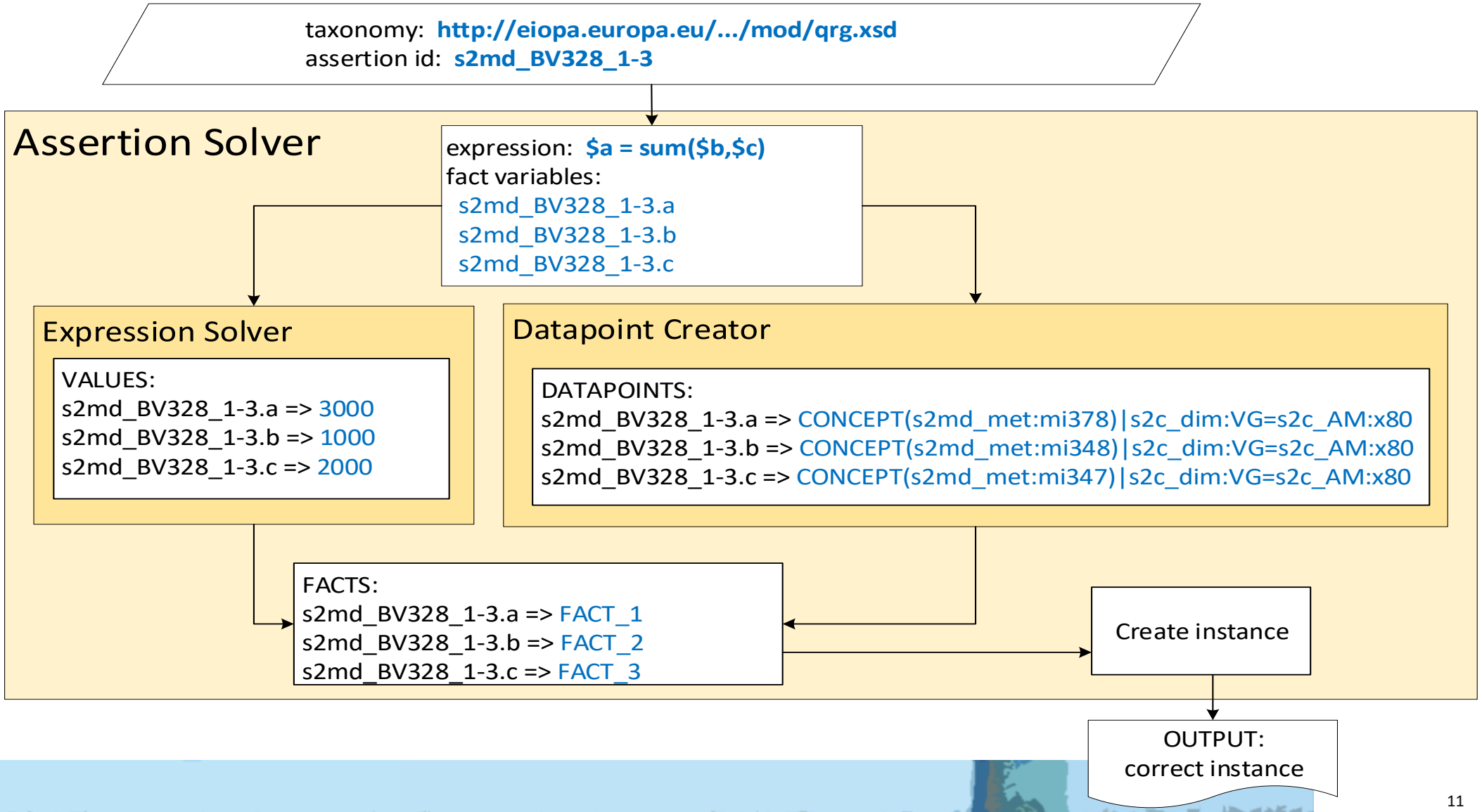


The solver must perform two operations when solving an assertion:
1/ **create datapoints** (set of aspects) satisfying the conditions associated with fact variable's filters
2/ **find values** which satisfy the expression associated with the assertion
In effect we can identify two major modules of the Assertion Solver: **Datapoint Creator** and **Expression Solver**.

In the final step, a datapoint and value are combined together as an **XBRL fact**.

The first task (creating a datapoint) is XBRL-specific. The second (finding a numeric solution) is more common and an existing constraint solver library can be used e.g. **Choco** (http://www.choco-solver.org/) or **JaCoP** (https://github.com/radsz/jacop)

SOLVING A SINGLE ASSERTION - EXAMPLE



WHAT EXPRESSIONS CAN BE SOLVED?

The numeric expression solver has been implemented with the use of JaCoP - an open-source constraint solver*.

The solver supports the following (and many more) constraints (or operators):

- arithmetic: +,−,∗, /, =,≠, <,≤, >,≥, X mod Y, X^Y

- logical: or, and

- conditional: if ... then ... (else ...)

Examples of assertions from **qrg** entrypoint (Solvency II taxonomy), with percentage occurrence, which can be easily solved using JaCoP.

#, count, percent, simplified expression, sample assertion id
1, **28.70%**, $a =sum($b, $c, ...), [s2md_BV313_1-3]
2, 9.57%, $a = sum($b, $c, ..., -1*($d)), [s2md_BV208-2]
3, 7.83%, $a = sum($b, -1*($c)), [s2md_BV330_1-3]
4, 6.96%, $a = $b, [s2md_BV139-4]

*see: http://jacopguide.osolpro.com/guideJaCoP.pdf

Challenges:

- variable dependency
- multiple executions of an assertion
- more supported expressions
- ... and many other

CHALLENGE 1 - ASSERTION DEPENDENCY

Fact variables in various assertions may reference the same fact.

The solver must identify whether any two variables are equivalent or not before finding values satisfying expressions in question. Equivalent variables occur e.g. in assertions BV95-1 and BV102-1 belonging to Solvency II qrg entrypoint.

			Total	Tier 1 - unrestricted	Tier 1 - restricted	Tier 2	Tier 3
			C0010	C0020	C0030	C0040	C0050
Ancillary own funds			-	-	-	-	-
	Unpaid and uncalled ordinary share capital callable on demand	R0300	-86	-	-		-
	Unpaid and uncalled initial funds, members' contributions or the equivalent basic own fund item for mutual and mutual - type undertakings, callable on demand	R0310	25	-	-		-
	Unpaid and uncalled preference shares callable on demand	R0320	48	-	-		
	A legally binding commitment to subscribe and pay for subordinated liabilities on demand	R0330	$b 11	-	-		
	Letters of credit and guarantees under Article 96(2) of the Directive 2009/138/EC	R0340	-100	-	-		-
	Letters of credit and guarantees other than under Article 96(2) of the Directive 2009/138/EC	R0350	42	-	-		
	Supplementary members calls under first subparagraph of Article 96(3) of the Directive 2009/138/EC	R0360	-25	-	-		-
	Supplementary members calls - other than under first subparagraph of Article 96(3) of the Directive 2009/138/EC	R0370	91	-	-		
	Non available ancillary own funds at group level	R0380	$c $a 78	-	-	$b 8	$c 70
	Other ancillary own funds	R0390	$d 71	-	-		
Total ancillary own funds		R0400	$a -1	-	-		

BV95-1: $a = sum(sum($b),-1*$c, $d)

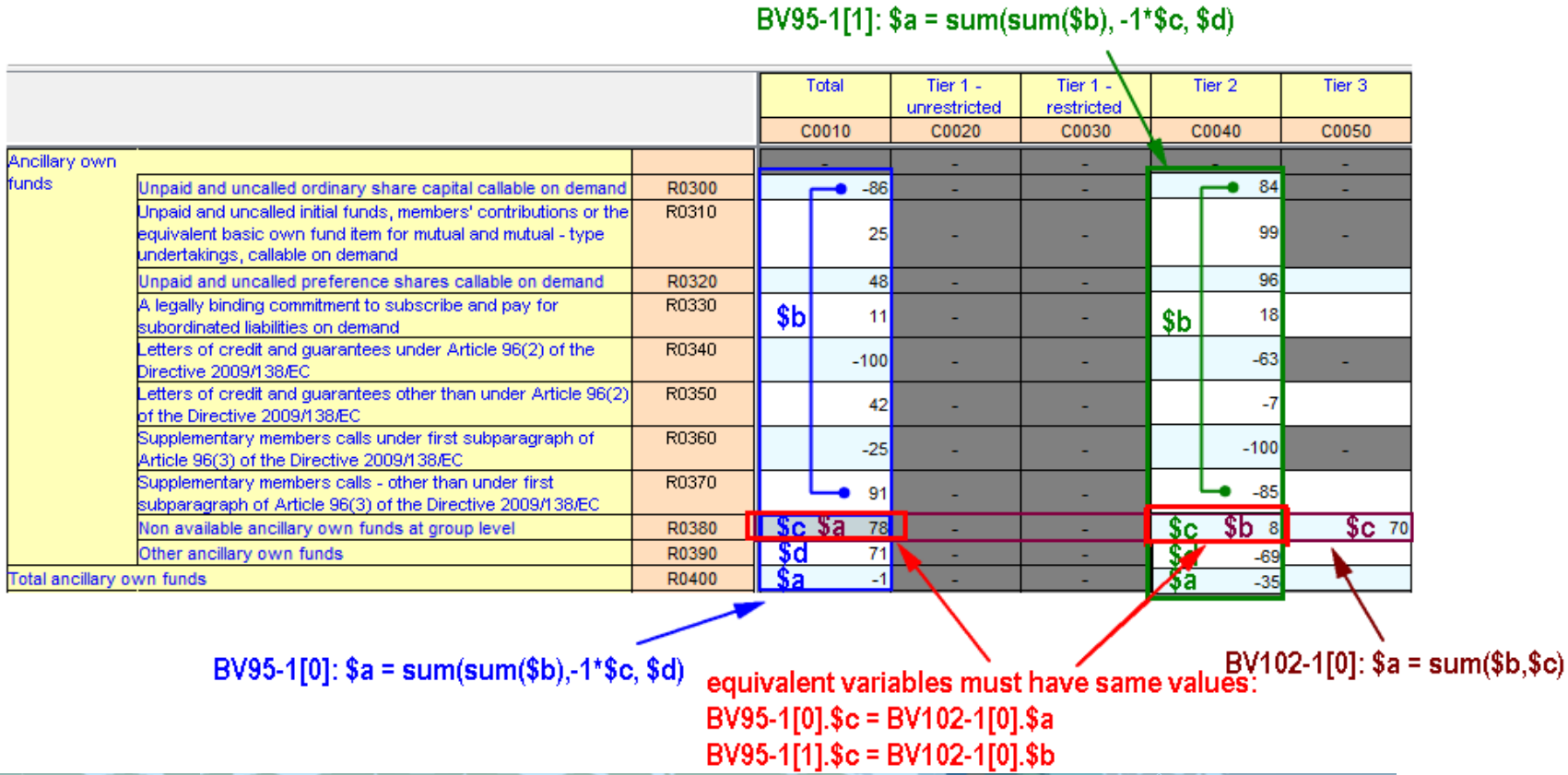
BV102-1: $a = sum($b,$c)

BV95-1.$c = BV102-1.$a variables reference the same fact and this must be taken into account by the solver!

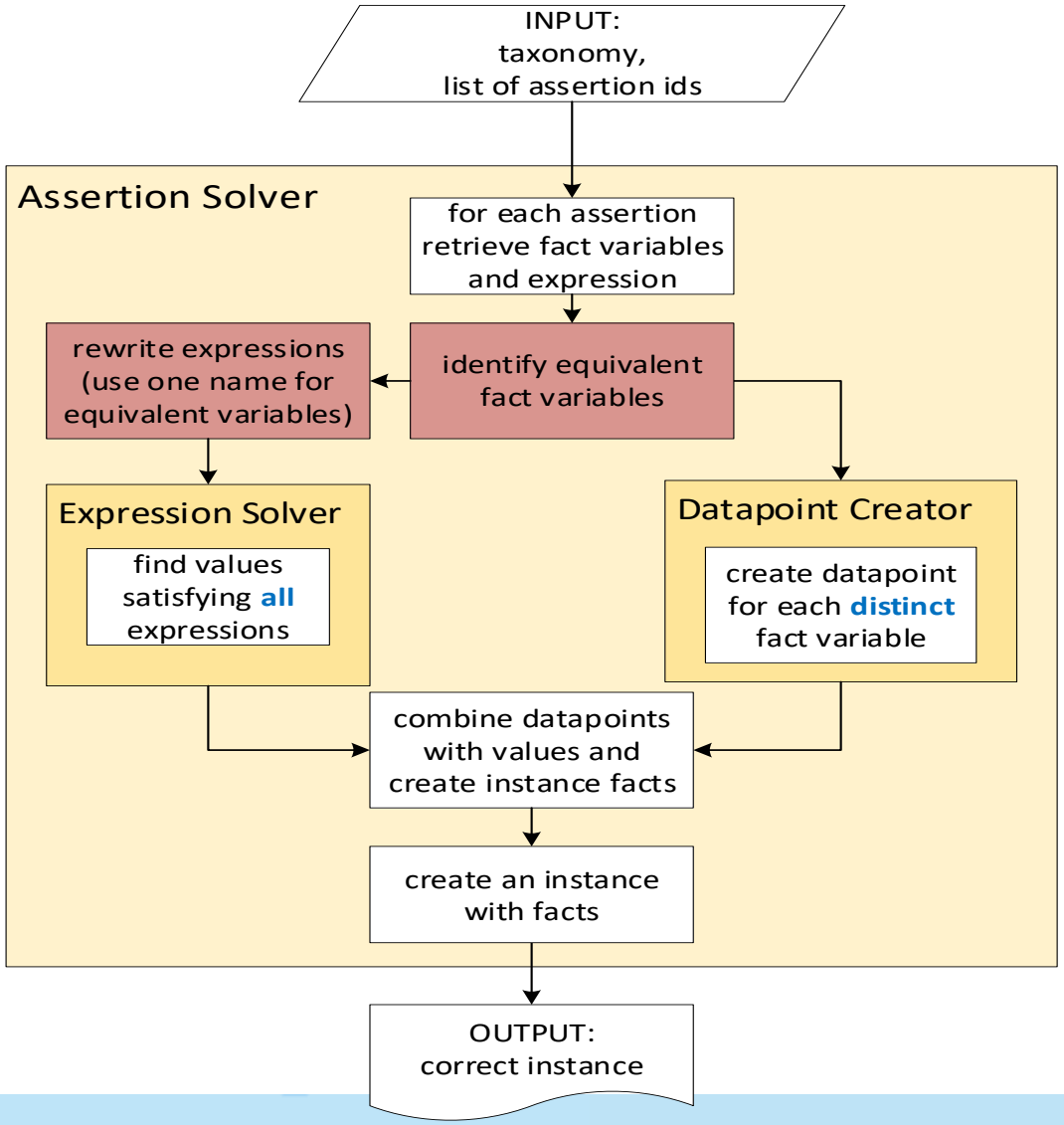
CHALLENGE 2 - MULTIPLE EXECUTIONS OF AN ASSERTION

Dependency between assertions can get even trickier when an assertion is executed multiple times (on different sets of facts from the same instance).

In the example below, the fact associated with $b variable of BV102-1, causes second execution of the BV95-1 assertion! So we need to index each individual assertion occurrence in order to properly identify variables and then find out whether they are equivalent. In the example, there are 3 occurences of assertions: BV102-1[0], BV95-1[0] and BV95-1[1].



ASSERTION SOLVING PROCESS - MULTIPLE ASSERTIONS

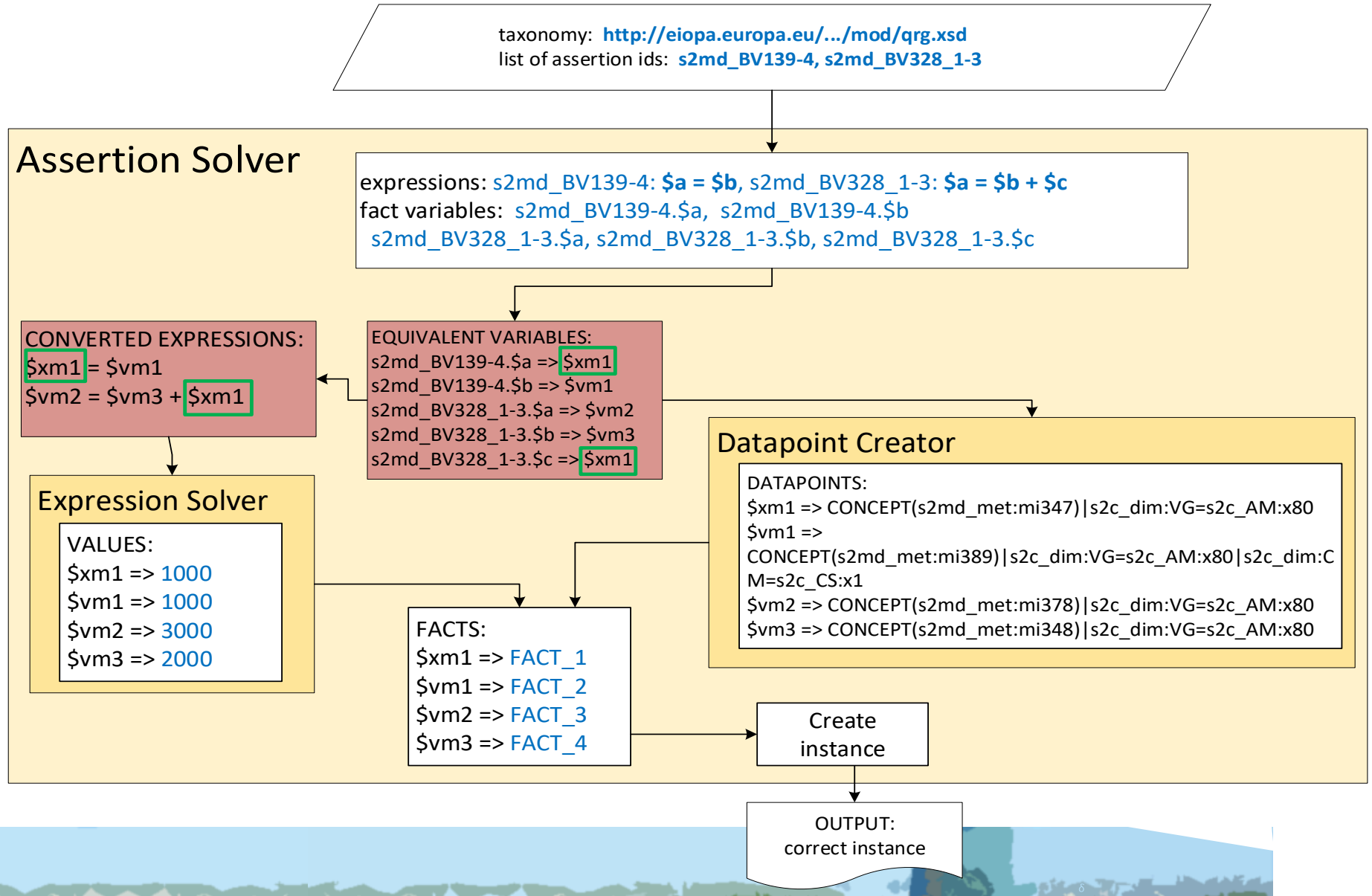


When finding a solution for multiple assertions **additional steps** must be performed before creating datapoints and before numeric expression solving.

The solver must identify whether fact variables used in various assertions are **equivalent** i.e. the filters associated with a fact variable ($a) define the same filtering conditions as the filters associated with another fact variable ($b). If so, both variables are replaced with a new variable ($x1) being a representative of the entire equivalence set.

Datapoints are created only for the representatives of the variable equivalence sets and **expressions to be solved are rewritten** before submitting to the expression solver.

SOLVING MULTIPLE ASSERTIONS - EXAMPLE



CHALLENGE 3 - MORE SUPPORTED EXPRESSION TYPES 1/3

Looking at top 10 most frequently occurring expression types (accounting for approx. 50% of all expressions) in Solvency II, we can find out that we can categorize them in two groups:

1. simple numeric expression:

a comparison between results of arithmetic operations on fact variables (e.g. $a = sum($b, $c))

2. QName implication numeric:

implication of the form: if (QName($a) = *literal*) then *simple_numeric_expression*

Easily we can add two more categories:

3. simple text expression:

expressions with form: matches($a, *regular_expression*)

4. QName implication text:

implication of the form: if (QName($a) = *literal*) then *simple_text_expression*

CHALLENGE 5 - MORE SUPPORTED EXPRESSION TYPES 2/3

Examples:

simple numeric:

```
iaf:numeric-equal($a, $b)
iaf:numeric-equal($a, iaf:max(($b, $c)))
iaf:numeric-equal($a, iaf:max((0, (iaf:sum(($b, $c, $d))))))
```

QName implication numeric:

```
if ($a = xs:QName('s2c_CN:x1')) then (iaf:numeric-equal($b, $c)) else (true())
if ($a = xs:QName('s2c_CN:x1')) then (iaf:numeric-equal($b, iaf:sum(($c, $d, $e)))) else (true())
if ($a = xs:QName('s2c_CN:x59')) then ($b ge abs($c)) else (true())
```

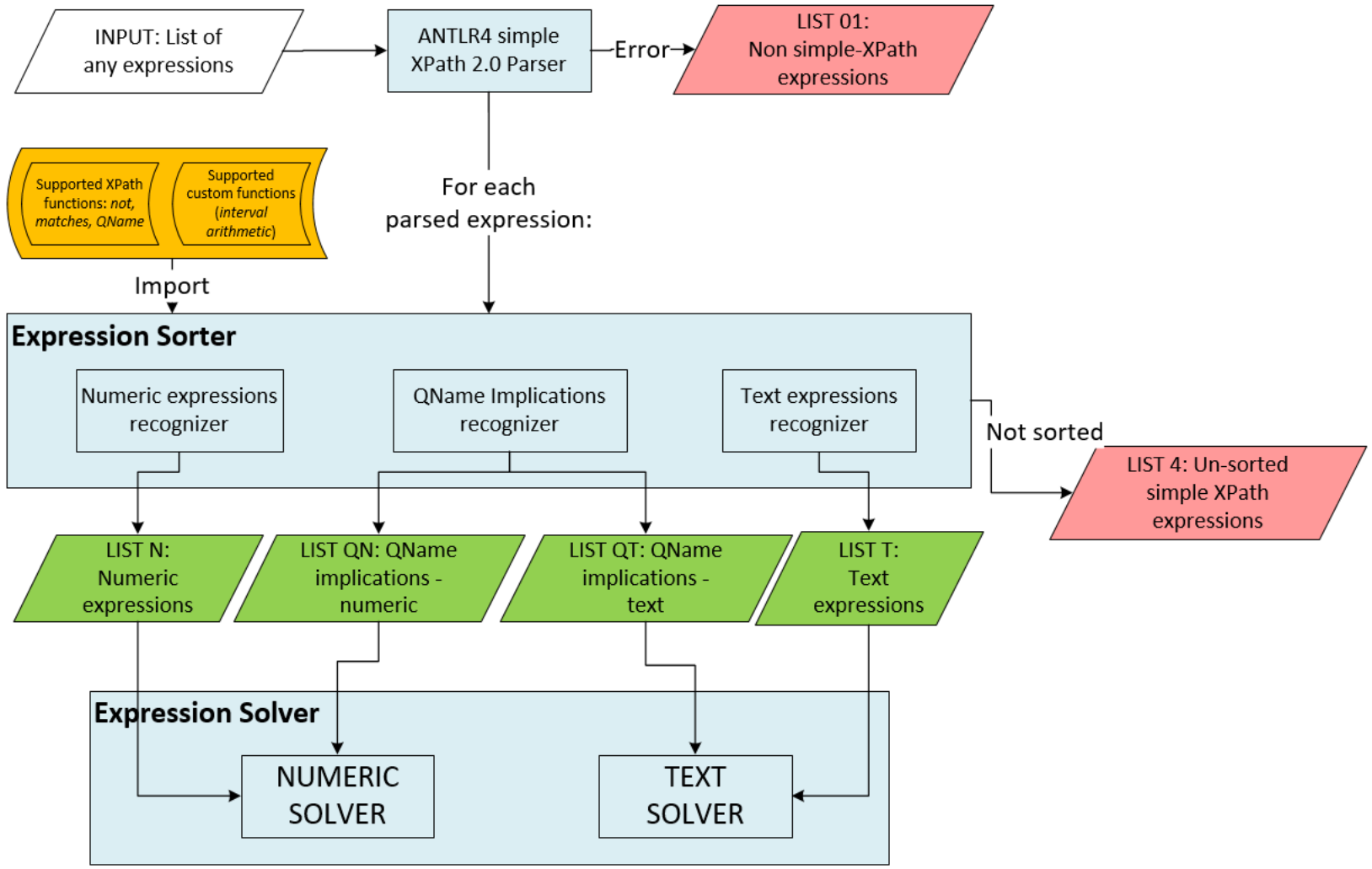
simple text:

```
matches($a, '^((1$)|(9$)){1}$')
```

CHALLENGE 5 - MORE SUPPORTED EXPRESSION TYPES 3/3

In order to handle the 4 categories of expressions, the solver needs to:

1. correctly recognize an expression's category
2. check whether an implication's predecessor holds (in case of QName implications)
3. route the expression to appropriate solver: numeric solver (based on JaCoP) or text solver (Generex library)



ROADMAP

Version: 0.1
November 2018
DataAmplified, Dubai

Version: 0.2
June 2019
Eurofiling Conf. Frankfurt

Version: 0.3
2019
???

goal:

- **generate correct facts for a demo taxonomy**

restrictions:

- single execution of an assertion
- only concept and dimensional formula filters
- single fact per sequence variable
- only numeric expressions

goal:

- **generate correct facts for Solvency II taxonomy**

enhancements:

- all formula filters supported
- multiple executions of an assertion
- any number of facts per sequence variable
- preconditions: filing indicators and enums
- formal grammar (ANTLR) of supported expressions
- POC: text expressions, QName implications

requirements:

- increase coverage to 90% for ars, qrg, qrb
- finalize text and QName implication solvers' implementation
- command line interface
- EBA taxonomy tests

SOLVING SOLVENCY II ENTRYPOINTS

Statistics from numeric solver (using Formula Solver v.0.2) for Solvency II entrypoints: qrg, qrb and ars:

****

ENTRYPOINT:
....solvency2/2017-07-15/mod/qrg.xsd

****

CATEGORIZED EXPRESSIONS:
All: 107*
Numeric: 70; pct of total: 65.42%
Uncategorized: 32; pct of total: 29.91%

****

VALIDATION RESULTS:
Number of assertion occurrences: 226
Number of success evaluations: 226
Number of failed evaluations: 0

****

ENTRYPOINT:
....solvency2/2017-07-15/mod/qrb.xsd

****

CATEGORIZED EXPRESSIONS:
All: 167*
Numeric: 127; pct of total: 76.05%
Uncategorized: 36; pct of total: 21.56%

****

VALIDATION RESULTS:
Number of assertion occurrences: 361
Number of success evaluations: 347
Number of failed evaluations: 14

****

ENTRYPOINT:
....solvency2/2017-07-15/mod/ars.xsd

****

CATEGORIZED EXPRESSIONS:
All: 1286*
Numeric: 408; pct of total: 31.73%
QName implications numeric: 474; pct of total: 36.86%
Uncategorized: 392; pct of total: 30.48%

****

VALIDATION RESULTS:
Number of assertion occurrences: 1318
Number of success evaluations: 1315
Number of failed evaluations: 3

*some expressions (e.g. existence checking or duplicated) have been filtered-out before sending to the solver

CONTACT

Eugeniusz Tomaszewski
Business Analyst

FUJITSU

FQS POLAND LIMITED

E-Mail: xbrl@fqs.pl