Towards Constraint Logic Programming over Strings for Test Data Generation

Sebastian Krings, J. Schmidt, P. Skowronek, J. Dunkelau, D. Ehmke

Hochschule Niederrhein
University of Applied Sciences

Test Data vs. Privacy

Software testing needs appropriate test data

- Covers desired scenarios
- Realistic structure
- Realistic amount
- Available

Personal data ~~should~~ has to be protected

- GDPR / DSGVO
- ISO 27k
- …
- Best Practices

Just Anonymize?

First Name	Last Name	Birthday	Customer No.
Egon	Maier	10.10.1963	EM63-005
Harald	Müller	08.04.1973	HM73-001
Hannah	Michels	06.09.1973	HM73-002
…	…	…	…

Just Anonymize? It‘s complicated ….

First Name	Last Name	Birthday	Customer No.
Egon	Maier	10.10.1963	EM63-005
Harald	Müller	08.04.1973	HM73-001
Hannah	Michels	06.09.1973	HM73-002
…	…	…	…

First Name	Last Name	Birthday	Customer No.
Stephan	Kaiser	08.08.2007	XY68-005
Stephanie	Michels	08.02.1976	HM73-001
…	…	…	…
…	…	…	…

Data Generators

- Database-based generators using schemata or creating copies
  - Rely on production data
  - => ☹
- Interface-based generators analyze API
  - Blackbox only
  - Lacking intellectual redundancy (four-eyes principle)
  - => ☹
- Code-based generators take source code into account
  - Unable to work with source code that is not available
  - Lacking intellectual redundancy (four-eyes principle)
  - => ☹
- Specification-based generators using specifications in a formal notation
  - => Needs formal notation
  - => Needs an appropriate backend, i.e., a constraint solver over all used data types
  - => ☺ ?

Requirements Towards Solvers

- Idea: Follow Oracle SQL
  - designed for the description of data flows
  - It is widely used by developers, test data specialists and technical testers
  - SQL statements can easily be extracted from source
  - SQL is declarative and offers a “good” level of abstraction
- Unbounded unicode strings
- Integers, fixed point numbers, reals, booleans and dates.
- 54 functions on strings
  - Concatenation, length, regex, substring, conversion to int, …
- Constraint handlers for all types must interwork
- Expect correctness, cannot expect (refutation) completeness

Current Approach: autogen / CLPQS

- Proprietary solution implemented and used by periplus instruments
- Specification-based
- SQL as input language
- Mostly focussed on model-based testing
- Goal: experiment with different representations and propagation rules

Check alternative approaches

- Avoid not-invented-here-syndrome!
- MiniZinc / FlatZinc / Zinc Solvers
- SMT Solvers
  - Z3 and derivates
  - CVC4
  - Trau, G-Strings, Geocode, …
- Hampi
- Sushi
- Solvers translating to bit vectors, etc.

Alternative Approaches

Solver	Strings			Combined Solver		
	Unbounded	Unicode	SQL Operations	Integer	Boolean	Real
CLPQS	✓	✓	✓	✓	✓	(✓)
MiniZinc	✗	✗	(✓)	✗	✗	✗
CVC4	✓	✗	✓	✓	✗	✗
Z3-str3	✓	✗	(✓)	✓	✗	✗
S3	✓	✗	✓	✓	✓	✗
Hampi	✗	✗	(✓)	✗	✗	✗
Sushi	✓	✓	✓	✗	✗	✗
G-Strings	✗	✗	(✓)	✗	✗	✗
Trau	✓	✗	✓	✓	✗	✗

New Prolog / CHR-based Solver

- Aim for proof-of-concept first
- Domain definition:
  - Unbounded Strings
  - Over extended ASCII by default, unicode by request
  - Regex-based domain literals
- Domain representation as finite automata:
  - `automaton_dom([…states…],[(0,a,1),…],[…initial…],[…final…])`

CHR Rules by Example

Listing 1. CHR rules for the membership constraint `str_in/2`.

```
1  str_in(S1, S2) <=>
2      string(S2) | gen_dom(S2, D),str_in(S1, D).
3  str_in(_, D) ==> is_empty(D) | fail.
4  str_in(Var,D) ==> D = string_dom(Cst) | Var = Cst.
5  str_in(S, D1), str_in(S, D2) <=>
6      D1 \= D2 | intersection(D1, D2, D3), str_in(S, D3).
7  str_in(S, D1)\ str_in(S, D2) <=>  D1 == D2 | true.
```

Listing 2. CHR rules for the concatenation constraint `str_concat/3`.

```
1  str_in(S1, D1), str_in(S2, D2), str_concat(S1, S2, S3) ==>
2      concat(D1, D2, D3), str_in(S3, D3).
3  str_in(S1, D1), str_concat(S1, S1, S3) ==>
4      concat(D1, D1, D3), str_in(S3, D3).
```

New Prolog / CHR-based Solver

Efficiency?

Case Studies

- Two case studies performed
  - IBAN numbers
  - Dates
- Simple studies with well-understood test data
- Check proof-of-concept before proceeding further
- No sub-solvers for now => no complicated constraints for now

Case Study 2: IBAN Numbers

Listing 5. Constraint system to compute all valid german IBANs.

```
1  iban(IBAN) :-
2      SigmaC in 0..96,
3      BBAN in 10000000000000000..99999999999999999,
4      SigmaB #= BBAN * 1000000 + 131400,
5      SigmaB mod 97 #= SigmaC,
6      str_label([SigmaB, SigmaC]),
7      str_to_int(BBANStr, BBAN),
8      CheckSum #= 98 - SigmaC,
9      str_to_intl(CheckSumStr, CheckSum),
10     str_size(CheckSumStr, 2),
11     str_in(DE, "DE"),
12     str_concatenation(DE, CheckSumStr, IBANPrefix),
13     str_concatenation(IBANPrefix, BBANStr, IBAN),
14     str_label([IBAN]).
```

Table 2. Benchmarks for generating IBANs. Walltime in seconds.

Amount	1	10	100	1,000	10,000	100,000	250,000
CLPQS	0.006	0.024	0.240	2.029	32.163	1525.457	9261.204
ConString	0.007	0.038	0.105	1.066	26.573	1342.597	9841.225

Case Study 2: Dates

Listing 6. Constraint system to compute diverse calendar date expressions

```
1  date(Date) :-
2    WeekDay str_in "Monday|Tuesday|...|Sunday",
3    Month str_in "January|February|...|December",
4    Day str_in "[1-9]|[1-2][0-9]|3[0-1]",
5    Year str_in "[1-9][0-9]{0,3}",
6    MonthDay match Month + "_" + Day,
7    MonthDayYear match MonthDay + ",_" + Year \/ MonthDay,
8    FullDate match WeekDay + ",_" + MonthDayYear,
9    Date match MonthDayYear \/ FullDate \/ WeekDay,
10   str_label([Date]).
```

Table 3. Benchmarks for generating date expressions. Walltime in seconds.

Amount	1	10	100	1,000	10,000	100,000
CLPQS	0.000	0.000	0.000	0.000	0.000	0.010
ConString	0.010	0.010	0.010	0.011	0.080	0.965

Future Work

- An efficient backend
  - Better data structures in Prolog?
  - Native data structures in C?
  - Port dk.brics.automaton to Prolog
- Combining solvers
  - Add SMT solvers and others as sub-solvers
  - Need to figure out communication / integration / shared state
- More thorough case studies

Conclusions

- Solvers for string constraints have made considerable progress recently
- However, hurdles remain and test data generation remains complicated
- (Simple) prototypical generator for synthetic test data implemented
- Combination of constraint logic programming / classical domain propagation resonable
- No single solver will be able to handle all requirements sufficiently
  - Reimplementing features commonly found in other solvers might not worthwhile
  - Integration of solvers very promising

Last …

Thank you for your attention!

Any questions?