Bug Squashing with SQLsmith

andreas.seltenreich@credativ.de

October 25, 2018

Outline

Motivation

Testing Methodology

Analysis of Bugs Uncovered

Design

Future Work

Motivation: My Story

credativ

- Inspired by Csmith, a random C program generator
- While working on a C compiler, I learned that one can never have enough testing
  - Regression tests, unit tests and testbenches were all green
  - Csmith made assertions fail in my optimization phase
- Me in 2015: We need an SQLsmith!
- In total, it found:
  - 71 Bugs in PostgreSQL
  - 3 in SQLite
  - 50 in MonetDB
  - 6 in various libraries (even glibc!)

Motivation: Who's it for?

credativ

- Developers of SQL speaking databases
- Extension writers
- Reviewers of submitted patches
- Security auditers
- Indirectly, all users profit from additional quality assurance

On Fuzz Testing: Bit-Level

credativ

Bit-Level fuzzers (e.g. AFL, libFuzzer)

- Only applicable when information density is very high
- Do not grasp high-level concepts such as syntax, schema, catalog, identifiers or scope
- Works ok for fuzzing Postgres' regexp parser
- Need ages to find the first trivial syntactically correct query
- Need eons to find a hit in the catalog/schema

On Fuzz Testing: Domain-Aware

Domain-aware fuzzers (Csmith, SQLsmith)

- Generate syntactically and semantically valid input at high speed
- Still cannot interpret the result semantically
- Semantics may be verified indirectly

Prior work

- CSmith by `utah.edu` (since 2011)
  - Found over 400 bugs in various compilers
  - Finds bugs in optimizations, code generators, register allocators, etc. despite fuzzing the parser
  - BSD-style license
- RAGS by Microsoft (conference paper from 1998)
  - They implemented differential testing
  - Queries look similar to SQLsmith's, albeit smaller
  - No code available

Running SQLsmith

- Just tell it the target database

```
$ sqlsmith --target="host=/tmp port=65432 dbname=regression"
$ sqlsmith --sqlite="file:~/.firefox/places.sqlite?mode=ro"
$ sqlsmith --monetdb="mapi:monetdb://localhost:50000/smith"
```

- Using `--verbose`, it prints a character for each query

symbol	meaning
.	ok
S	syntax error
t	timeout
C	broken connection
e	other error

Advanced Options

`--log-to=connstr`	log errors to postgres database
`--seed=int`	seed RNG with specified int instead of PID
`--dump-all-graphs`	dump generated ASTs
`--dump-all-queries`	print queries as they are generated
`--dry-run`	print queries instead of executing them
`--exclude-catalog`	don't generate queries using catalog relations
`--max-queries=long`	terminate after generating this many queries
`--rng-state=string`	deserialize dumped rng state

How to Hunt Bugs

credativ

Watch out for symptoms like:

- Core dumping due to failed assertions, PANICs
- Outlandish error messages or warnings
  - Log them into a database to allow filtering
  - Analysis of historical data may also give insights
- Processes bloating, hogging CPU
  - Need to monitor system load to find these bugs

Nature of Bugs Found: Crashes

```
postgres=# select bit '1' >> (-2^31)::int;

LOG:  server process (PID 15838) was terminated by signal 11: Segmentatio
LOG:  terminating any other active server processes
LOG:  database system was not properly shut down; automatic recovery in p
LOG:  redo is not required
LOG:  database system is ready to accept connections
```

Nature of Bugs Found: Crashes (cont.)

credativ

```
Datum bitshiftleft(PG_FUNCTION_ARGS)
{
    VarBit  *arg = PG_GETARG_VARBIT_P(0);
    int32   shft = PG_GETARG_INT32(1);

    /* Negative shift is a shift to the right */
    if (shft < 0)
        PG_RETURN_DATUM(DirectFunctionCall2(
                            bitshiftright,
                            VarBitPGetDatum(arg),
                            Int32GetDatum(-shft)));
    /* do bitshift left for positive arguments */
```

Nature of Bugs Found: PANICs

credativ

```
postgres=# update brintest
  set oidcol = coalesce(brintest.oidcol, pg_my_temp_schema()
    timestamptzcol = clock_timestamp(), uuidcol = null
  returning brintest.byteacol;
WARNING: specified item offset is too large
PANIC: failed to add BRIN tuple
server closed the connection unexpectedly
```

Nature of Bugs Found: Failed Assertions


```
From:    Andreas Seltenreich <seltenreich(at)gmx(dot)de>
To:      pgsql-hackers(at)postgresql(dot)org
Subject: [sqlsmith] Failed assertion in postgres_fdw/deparse.c:1116

Creating some foreign tables via postgres_fdw in the regression db of
master as of de33af8, sqlsmith triggers the following assertion:

    TRAP: FailedAssertion("!(((((const Node*)(var))->type) == T_Var))",
            File: "deparse.c", Line: 1116)

gdb says var is holding a T_PlaceHolderVar instead.
```

Nature of Bugs Found: Internal ERRORs

credativ

```
ERROR: failed to build any 8-way joins
ERROR: could not devise a query plan for the given query
ERROR: plan should not reference subplan's variable
ERROR: failed to assign all NestLoopParams to plan nodes
ERROR: could not find pathkey item to sort
ERROR: too late to create a new PlaceHolderInfo
```

Nature of Bugs Found: Other ERRORs

credativ

```
From:    Andreas Seltenreich <seltenreich(at)gmx(dot)de>
To:      pgsql-hackers(at)postgresql(dot)org
Subject: [sqlsmith] Missing CHECK_FOR_INTERRUPTS in tsquery_rewrite

[...]
testing with sqlsmith yielded an uncancellable backend hogging CPU time.
[...]
select ts_rewrite(
(select string_agg(i::text, '&')::tsquery from generate_series(1,32) g(i)
(select string_agg(i::text, '&')::tsquery from generate_series(1,19) g(i)
'foo');
```

How to Hunt Bugs (cont.)

credativ

- Complicate DUT configuration (replication, non-default settings)
- Make interesting objects or values available to sqlsmith
  - Use a regression DB as a starting point
  - Add Foreign Tables
  - Have infinity, NaN, 2^{31}-1, etc around in your database
- Use additional tools
  - low-memory/libfailmalloc
  - ASAN
  - valgrind
  - trap on division by zero

My Testing Rig

credativ

- Cluster of cheap surplus Sandy Bridge quad-cores in my apartment
- Ansible to put testing arrangements on machines
- gdb scripts to harvest backtraces from appearing coredumps
- `sinfod`
  - Broadcasts system load on the network
  - Yields a real-time view on the entire cluster load
  - Many failure modes are readily identifyable

BUGs by Nature over Modules

credativ

	Plan	Exec	Access	TX	Oper	Contrib	ADT	$\sum$
Segfault	2	6	1	3	8	1		21
PANIC			1		1			2
TRAP	11	4	4	1	4	1		25
ERROR	10				4	1		15
$\div 0$	3						2	5
other		1			2			3
$\sum$	26	11	6	4	19	3	2	71

Regarding SQLite3, all three bugs were failed assertions in the planner and executor

Test Coverage

```
src/postgres$ ./configure --enable-coverage
```

test load	overall	parser
`sqlsmith`	39.8	30.3
`make check`	62	80.2
`sqlsmith+make check`	65.1	80.4

Numbers generated using sqlsmith commit 7ffac2d, running 4 instances w/25000 queries each. Postgres code for the analysis was from master branch at around the same time.

Project Goals for SQLsmith

Inspired by Csmith, the following goals were set

- Be product-agnostic
- No requirement for templates/user-provided grammar/etc

  $\rightsquigarrow$ The language is the limit
- Deterministic generation for reproducability/benchmarking
- Speed: The bottleneck should always be the database under test (DUT)

Language Choice: C++11

credativ

- OO design well-suited for AST construction
- Absolute type safety
- Implicit memory management
- Standardized multi-threading
- Exceptions, also employed for backtracking in AST generation
- Speed

Product Abstraction

Two front-end classes provide product-agnostic access to the DUT

- Schema class
- DUT class

Implemented for:

- PostgreSQL 9.1 or later
- SQLite 3
- MonetDB (contributed by cwi.nl)
- Various forks on github

Auxiliary Modules

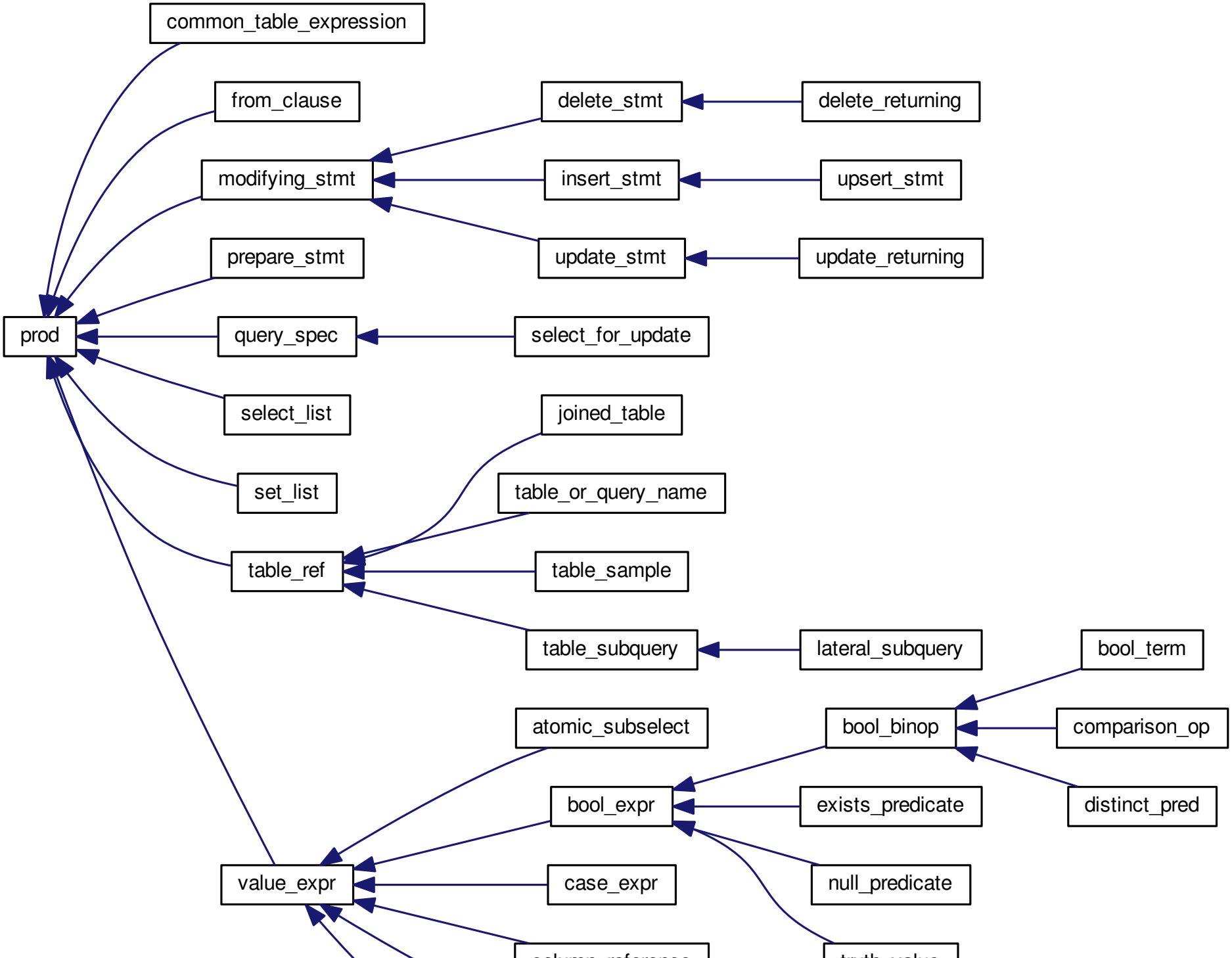
- Class `logger`
  - Invoked for generation and result
  - Implementations for
    - logging to stderr (with primitive analysis)
    - logging into a database (allows filtering)
    - collecting statistics
- Impedance matching module
  - Allows to adapt grammar to the DUT
  - Productions consistently leading to errors are blacklisted

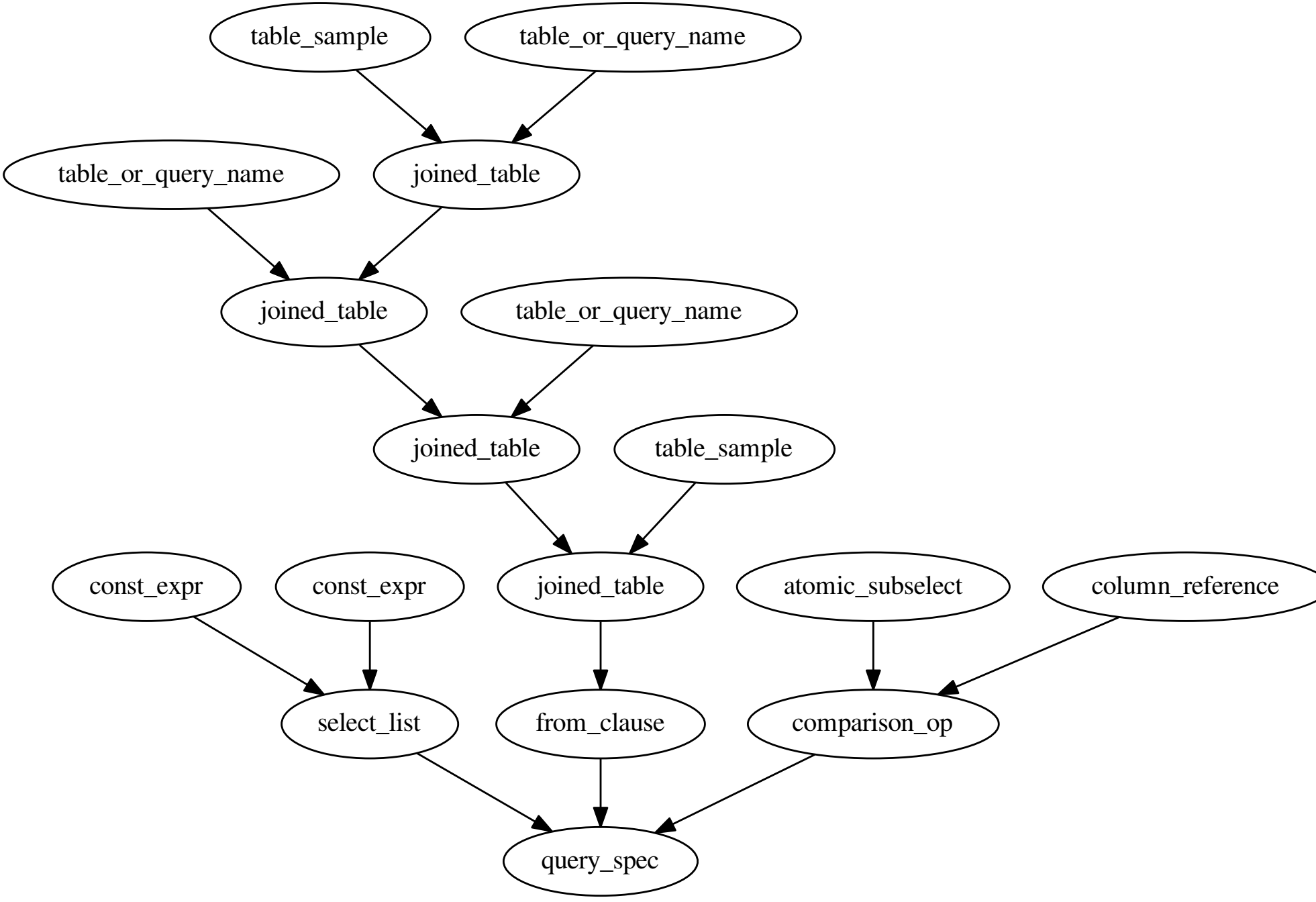
Grammar Production Class Hierarchy

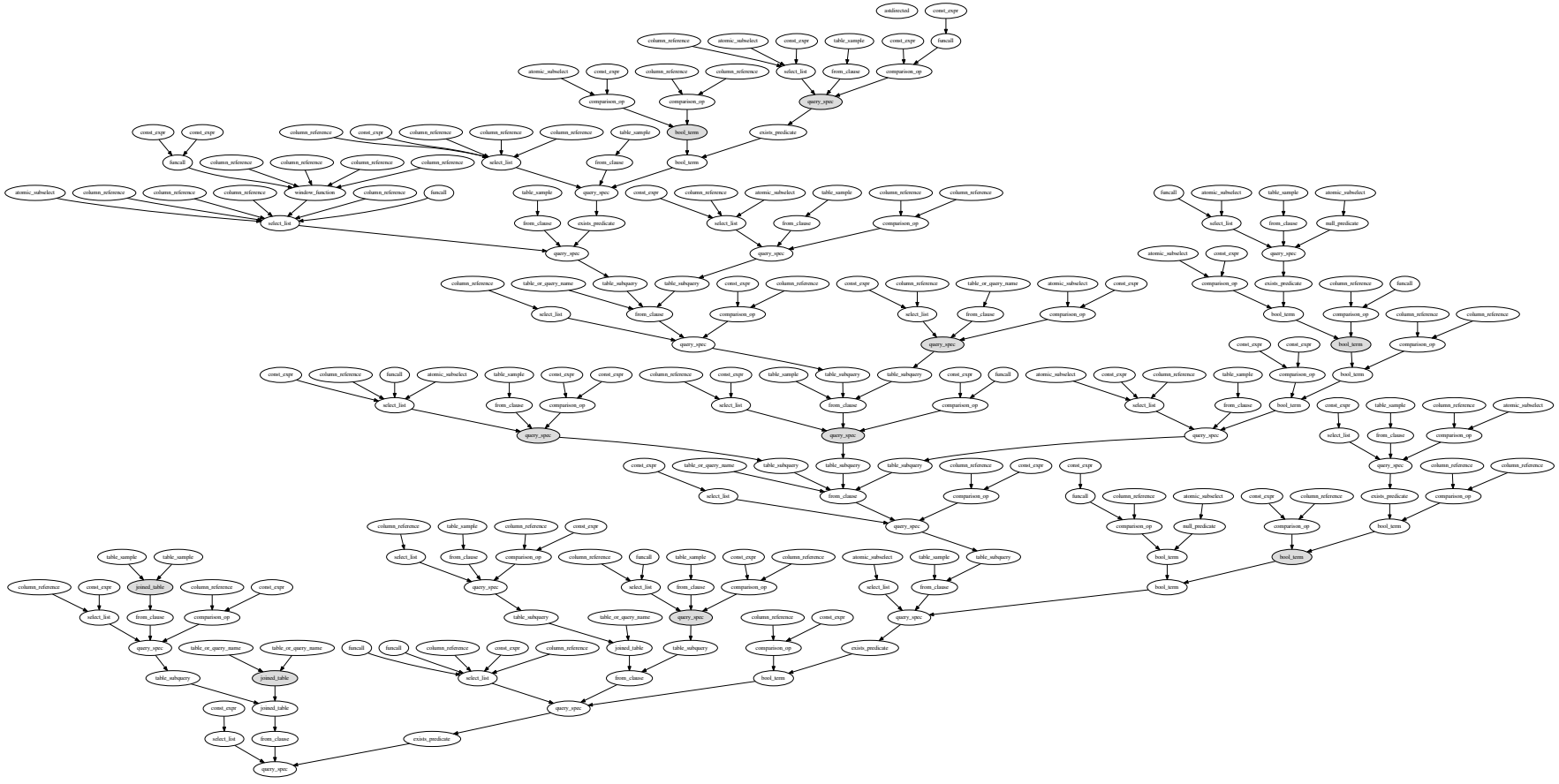
credativ

- Base class `prod` for all grammar productions
- Instantiation yields a random object of the respective production
- Some members of `prod` subclasses are derived from `prod` as well, forming the AST
- Visitor design pattern allows walking this AST
- `operator«` emits SQL for a production
- Productions are instantiated speculatively
  - Constructors throw exception when there is no way to create a valid instance in the context

  $\rightsquigarrow$ Backtracking to higher AST levels to get out of dead ends


common_table_expression
from_clause
delete_stmt
delete_returning
modifying_stmt
insert_stmt
upsert_stmt
prepare_stmt
update_stmt
update_returning
prod
query_spec
select_for_update
select_list
joined_table
set_list
table_or_query_name
table_ref
table_sample
table_subquery
lateral_subquery
bool_term
atomic_subselect
bool_binop
comparison_op
bool_expr
exists_predicate
distinct_pred
value_expr
case_expr
null_predicate


table_sample
table_or_query_name
table_or_query_name
joined_table
joined_table
table_or_query_name
joined_table
table_sample
const_expr
const_expr
joined_table
atomic_subselect
column_reference
select_list
from_clause
comparison_op
query_spec


unlimited
const_expr
columns_reference
atomic_subselect
const_expr
table_sample
const_expr
funcall
atomic_subselect
const_expr
column_reference
column_reference
select_list
from_clause
comparison_op
comparison_op
comparison_op
query_spec
const_expr
const_expr
column_reference
column_reference
column_reference
column_reference
table_sample
bool_term
exists_predicate
funcall
column_reference
column_reference
column_reference
column_reference
select_list
from_clause
bool_term
atomic_subselect
column_reference
column_reference
column_reference
window_function
column_reference
funcall
table_sample
query_spec
const_expr
column_reference
atomic_subselect
table_sample
column_reference
column_reference
funcall
atomic_subselect
table_sample
atomic_subselect
select_list
from_clause
exists_predicate
select_list
from_clause
comparison_op
select_list
from_clause
null_predicate
query_spec
query_spec
query_spec
atomic_subselect
const_expr
comparison_op
exists_predicate
column_reference
funcall
column_reference
table_or_query_name
table_subquery
table_subquery
const_expr
column_reference
const_expr
column_reference
table_or_query_name
atomic_subselect
const_expr
comparison_op
bool_term
comparison_op
column_reference
column_reference
select_list
from_clause
comparison_op
select_list
from_clause
comparison_op
bool_term
comparison_op
query_spec
query_spec
const_expr
const_expr
bool_term
const_expr
column_reference
funcall
atomic_subselect
table_sample
const_expr
const_expr
column_reference
const_expr
table_sample
table_subquery
table_subquery
const_expr
funcall
atomic_subselect
const_expr
column_reference
table_sample
comparison_op
bool_term
select_list
from_clause
comparison_op
select_list
from_clause
comparison_op
select_list
from_clause
bool_term
const_expr
table_sample
column_reference
atomic_subselect
query_spec
query_spec
query_spec
select_list
from_clause
comparison_op
const_expr
table_or_query_name
table_subquery
table_subquery
table_subquery
column_reference
const_expr
const_expr
query_spec
column_reference
column_reference
select_list
from_clause
comparison_op
funcall
column_reference
atomic_subselect
const_expr
column_reference
exists_predicate
comparison_op
comparison_op
null_predicate
comparison_op
bool_term
bool_term
bool_term
column_reference
table_sample
column_reference
const_expr
table_sample
table_sample
select_list
from_clause
comparison_op
column_reference
funcall
table_sample
const_expr
column_reference
atomic_subselect
table_sample
table_subquery
select_list
from_clause
comparison_op
select_list
from_clause
bool_term
column_reference
const_expr
joined_table
column_reference
const_expr
query_spec
query_spec
table_subquery
table_or_query_name
query_spec
column_reference
const_expr
select_list
from_clause
comparison_op
table_or_query_name
table_or_query_name
funcall
funcall
column_reference
const_expr
column_reference
joined_table
table_subquery
comparison_op
exists_predicate
table_subquery
joined_table
select_list
from_clause
bool_term
const_expr
joined_table
query_spec
select_list
from_clause
exists_predicate
query_spec

`scope` Class

- So far we can generate syntactically correct queries
- Modelling of scope needed to generate semantically correct ones
- `scope` class models column and relation visibility
- Production constructors take a `scope` object where they pick their references from
- Productions may create a different scope to pass to their members

`sqltype` Class

credativ

- Type matching among columns, operators, functions
- Originally, SQLsmith did only consider type equality

  $\rightsquigarrow$ turned out too primitive
- Now there's a method: `sqltype::consistent(sqltype*)`
- Schema classes may fill the scope with a derived `sqltype` to adapt the grammar to a product-specific type model

On Randomness

- Csmith uses a sophisticated stochastical model
- SQLsmith uses
  - dice throws when the grammar allows alternatives

  $\rightsquigarrow$ d6(), d12(), d20() calls in factories
  - `random_pick<>()` from container when the schema/scope allows alternatives
  - All of them use an instance of C++11's `std::mt19937_64`

Extending SQLsmith

- Add support for a new RDBMs
  - Implement a Schema and DUT class for your product
- Extend the grammar
  - Derive something from the `prod` class
  - Extend a factory to return instances
- Doxygen-documenation available to visualize the class hierarchy and their collaboration diagrams

Future Work: Differential Testing

Product-differential testing:

- Microsoft did it with RAGS. Summary:
  - pro: "output validation proved to be extremely useful"
  - con: "the common SQL subset is relatively small"
  - con: "protability issues are problematic"
- Further: Deterministic queries are not enough, also need deterministic results. E.g. `...join pg_stat_activity ... where t > CURRENT_TIME ...`

Setting-differential testing:

- Repeat queries with idempotent GUC settings

Version-differential testing:

- Allow spotting regressions wrt. semantics

Future Work: Statement Simplification

- Generated statements are largish and it takes effort to reduce them to an often simple testcase
- This can be automated by cutting things from the AST while maintaining the failure mode
- Creduce is the solution for Csmith, implementing a SQLreduce is a natural step
- Postgres' parser has been factored out for stand-alone use, that's a good starting point
- Microsoft also did it for RAGS (no code available)

Future Work: Miscellaneous

- Multithreading
- Support more products
- Add more grammar productions
- Improve SQLsmith's primitve type system
  - About 25% of the queries currently result in type errors
- Extend Postgres with a compiled regexp type to improve filtering performance
- Log SQLSTATE
  - Need to fix libpqxx or use libpq instead

Thank You!

credativ

Selected ERRORs of the Day:

- value for domain things violates check constraint "meow"
- link of phone to hub does not make sense
- time zone "Bruce Momjian" not recognized
- return type mismatch in function declared to return things
- dimension mismatch in "+" operation: "6 Gy", "173.505 kmol"
- Lost connection to MySQL server during query

Bibliography

AFL: http://lcamtuf.coredump.cx/afl/

Creduce: https://embed.cs.utah.edu/creduce/

Csmith: https://embed.cs.utah.edu/csmith/

MSR-TR-98-21: https://www.microsoft.com/en-us/research/publication/massive-stochastic-testing-of-sql/

SQLsmith: https://github.com/anse1/sqlsmith

SQLsmith score list maintained by users: https://github.com/anse1/sqlsmith/wiki

libFuzzer: http://llvm.org/docs/LibFuzzer.html

sinfod: http://www.ant.uni-bremen.de/whomes/rinas/sinfo/man_sinfod.html

TODO: github-link to factored-out parser