Property-Based Testing for Spark SQL

Databricks Hackathon, Dec, 15
Cheng Lian & Wenchen Fan

Hey bricks,

Beijing just issued red alarm over air pollution for the first time. well... the air outside, which smells like smoke, makes me feel like Santa Claus climbing through a chimney... So, here's your X'mas present! Hope you like it :-)

databricks™

I  ENUMERATING TEST CASES

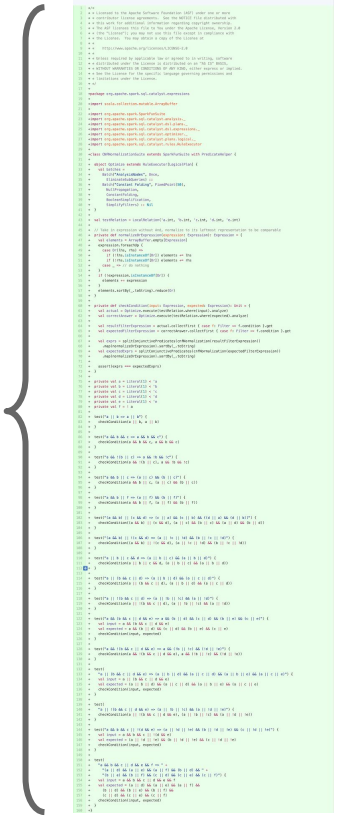
I  ENUMERATING TEST CASES

Don’t you?

PR #8200: CNF Conversion

25 lines of functional changes

160 lines of test cases, desperately enumerating all kinds of predicates…

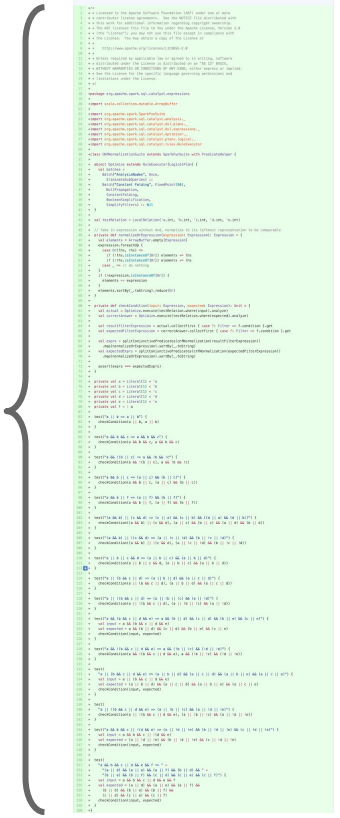


databricks™

PR #8200: CNF Conversion

25 lines of functional changes

160 lines of test cases, desperately enumerating all kinds of predicates…
(Well, Apache licence takes 10%)



databricks™

ScalaCheck & Property-Based Testing to the Rescue!

ScalaCheck helps you...

Converting:

```
def add(x: Int, y: Int) = x + y

test("add") {
  assert(add(0, 0) === 0)
  assert(add(1, 0) === 1)
  assert(add(-1, 0) === -1)
  assert(add(42, -1) === 41)
  assert(add(Int.MaxValue, 1) === Int.MinValue)
  assert(add(Int.MinValue, -1) === Int.MaxValue)
  // ...
}
```

ScalaCheck helps you...

Into:

```
def add(x: Int, y: Int) = x + y

test("add") {
  check { (a: Int, b: Int) =>
   add(a, b) == a + b
  }
}
```

Property-Based Testing

- Makes declarative statements about the output of your code based on the input

Property-Based Testing

- Makes declarative statements about the output of your code based on the input
- These statements are verified for many different random inputs

Property-Based Testing

- Makes declarative statements about the output of your code based on the input
- These statements are verified for many different random inputs
- Special "interesting" inputs are also covered
  - `Int.MaxValue`
  - `Double.NaN`
  - `...`

Input Minimisation

When a test case fails, you probably don't want to look at a deeply nested, randomly generated spaghetti input schema / expression / logical plan.

Input Minimisation

ScalaCheck provides a Shrink API for shrinking large, complex random input data into smaller, simpler ones to help developers identifying corner cases.

What We Did...

This Hackathon Project

Is based on a minimized version of Catalyst, named Scraper. It was originally written by Cheng as an optimizing compiler of the Brainf**k language in less than 300 loc for illustrating the power of Catalyst in a local conference held in China early this year.

Now it's used as a Catalyst playground for polishing and prototyping all kinds of ideas.

This Hackathon Project

- Implements a prototype of property-based testing facilities for Spark SQL using ScalaCheck
  - Provides ScalaCheck generators for
    - random SQL data types
    - random SQL values
    - random type checked SQL expressions
    - random resolved logical query plans

This Hackathon Project

- Implements ScalaCheck Shrink API for expressions
  - Shrink support for logical plans is on the way!

This Hackathon Project

- Brings a bonus
  - A set of concise, flexible, and declarative API for query plan and expression type checking

CNFConversion Test Revisited

```
testRule(CNFConversion, FixedPoint.Unlimited) { optimizer =>
  implicit val arbPredicate =
    Arbitrary(genPredicate(TupleType.empty.toAttributes))

  check { predicate: Expression =>
    val optimizedPlan = optimizer(SingleRowRelation filter predicate)
    val conditions = optimizedPlan.collect {
      case _ Filter condition => splitConjunction(condition)
    }.flatten

    conditions.forall {
      _.collect { case _ And _ => () }.isEmpty
    }
  }
}
```

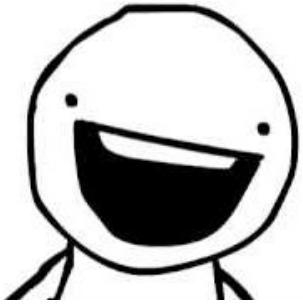
CNFConversion Test Revisited

```
testRule(CNFConversion, FixedPoint.Unlimited) { optimizer =>
  implicit val arbPredicate =
    Arbitrary(genPredicate(TupleType.empty.toAttributes))

  check { predicate: Expression =>
    val optimizedPlan = optimizer(SingleRowRelation filter predicate)
    val conditions = optimizedPlan.collect {
      case _ Filter condition => splitConjunction(condition)
    }.flatten

    conditions.forall {
      _.collect { case _ And _ => () }.isEmpty
    }
  }
}
```

Look ma!
No enumerating!



What If Things Go Wrong?

Let's break rule `CNFConversion` by removing a case branch, so that "Or"s with an "And" as its right branch fail the test:

```
object CNFConversion extends Rule[LogicalPlan] {
  override def apply(tree: LogicalPlan): LogicalPlan = tree transformDown {
    case plan: Filter =>
      plan transformExpressionsDown {
        case Not(lhs Or rhs)                => !lhs && !rhs
        case Not(lhs And rhs)               => !lhs || !rhs
        case (innerLhs And innerRhs) Or rhs => (innerLhs || rhs) && (innerRhs || rhs)
     // case lhs Or (innerLhs And innerRhs) => (innerLhs || lhs) && (innerRhs || lhs)
      }
  }
}
```

What If Things Go Wrong?

Expression minimisation works, and gives exactly the simplest corner case after 13 shrinks!

```
[info] - BadCNFConversion *** FAILED ***
[info]   GeneratorDrivenPropertyCheckFailedException was thrown during property evaluation.
[info]    (OptimizerSuite.scala:78)
[info]     Falsified after 1 successful property evaluations.
[info]     Location: (OptimizerSuite.scala:78)
[info]     Occurred when passed generated values (
[info]       arg0 =
[info]   Or
[info]   ├─ false: boolean
[info]   └─ And
[info]     ├─ false: boolean
[info]     └─ false: boolean // 13 shrinks
[info]     )
```

Potentials

- Property-based testing
  - Easier, finer grained testing for
    - expressions
    - analyzer
    - query optimizer
    - query planner
  - Potentially faster test speed since we can test in finer grain without issuing Spark jobs all the time
  - Better test coverage (well, if we ever measure it)

Potentials

- Random query generator
  - The one Impala brings is great, but it has too many moving parts, and doesn't shrink inputs when things go wrong
  - Generators for logical plans and SQL values can be building blocks for an embedded random query generator for Spark SQL

Potentials

- The new type check API
  - Should be able to significantly simplify existing type casting and type checking rules in Spark SQL analyzer
  - SQL linting
    - (Hey, there are strings implicitly casted into doubles, which can be dangerous!)

Follow-ups

- Porting to Catalyst
- Logical plan to SQL translation for building random query generator (and can be useful for fully implementing native view)
- Logical plan minimisation
- Enrich generators and shrinkers for
  - Aggregations
  - Window functions
  - ……

References

- Scraper
- ScalaCheck together with user guide
- Our poor example CNFConversion pull request
- Just in case you are not familiar with CNF
- Initial discussion about the type check API

Thanks!

No Q & A this time