

Automated Recognition of Axiomatic Theories

FroCoS 2013

Guillaume Burel, Simon Cruanes

ÉNSIIE/Cédric, 1 square de la résistance, 91025 Évry cedex, France
guillaume.burel@ensiie.fr
<http://www.ensiie.fr/~guillaume.burel/>

École polytechnique and INRIA, 23 Avenue d'Italie, 75013 Paris, France
simon.cruanes@inria.fr
<https://who.rocq.inria.fr/Simon.Cruanes/>

September 19th, 2013

Summary

- 1 Introduction
- 2 Detection of axioms
- 3 Detecting theories with Datalog
- 4 Application

- Goal: automated theorem proving within theories
- Recognize theories \rightarrow use specific knowledge
- Already done ad-hoc (e.g., for AC)
- We seek at more general method.

Context:

- First-order theorem proving with equality
- CNF formulas, saturation process (superposition)
- **Axiomatic theories** (finite sets of axioms)
- The theory is specified independently from signature
- Discovery also occurs during proof search
- Implementation in **Zipperposition**¹

¹see <https://www.rocq.inria.fr/deducteam/Zipperposition/>

Context:

- First-order theorem proving with equality
- CNF formulas, saturation process (superposition)
- **Axiomatic theories** (finite sets of axioms)
- The theory is specified independently from signature
- Discovery also occurs during proof search
- Implementation in **Zipperposition**¹

Examples:

- Group theory ✓
- Ring theory ✓
- Interpreted arithmetic ✗
- Peano arithmetic without induction ✓
- Theory of lattices ✓

¹see <https://www.rocq.inria.fr/deducteam/Zipperposition/>

What if we know about groups?

- Rewrite system:

$$\text{add}(X, 0) \rightarrow X$$

$$\text{add}(0, X) \rightarrow X$$

$$\text{add}(X, \text{minus}(X)) \rightarrow 0$$

$$\text{add}(\text{minus}(X), X) \rightarrow 0$$

$$\text{minus}(0) \rightarrow 0$$

$$\text{minus}(\text{minus}(X)) \rightarrow X$$

$$\text{minus}(\text{add}(X, Y)) \rightarrow \text{add}(\text{minus}(Y), \text{minus}(X))$$

$$\text{add}(\text{add}(X, Y), Z) \rightarrow \text{add}(X, \text{add}(Y, Z))$$

$$\text{add}(X, \text{add}(\text{minus}(X), Y)) \rightarrow Y$$

$$\text{add}(\text{minus}(X), \text{add}(X, Y)) \rightarrow Y$$

- Specific decision procedure
- Heuristics
- ...

What we would like to do:

- **abstract** a rewrite system from a specific signature
- detect instances of the group theory
 - $\langle \text{add}, \text{minus}, 0 \rangle$
 - $\langle \text{product}, \text{inverse}, 1 \rangle$
 - ...
- specialize rewrite systems with corresponding signatures
- solve (with the help of rewrite system)

→ This way, able to use more knowledge about problem!

Axiomatization of groups:

Axioms

$$\text{add}(X, 0) = X$$

$$\text{add}(\text{add}(X, Y), Z) = \text{add}(X, \text{add}(Y, Z))$$

$$\text{add}(X, \text{minus}(X)) = 0$$

Consider:

Axioms

$$s(X, Y, Z) \wedge s(X, Y, Z') \rightarrow Z = Z'$$

$$s(X, Y, a(X, Y))$$

$$s(z, X, X)$$

$$s(X, z, X)$$

$$s(X, m(X), z)$$

$$s(m(X), X, z)$$

$$s(X, Y, U) \wedge s(Y, Z, V) \wedge s(U, Z, W) \rightarrow s(X, V, W)$$

$$s(X, Y, U) \wedge s(Y, Z, V) \wedge s(X, V, W) \rightarrow s(U, Z, W)$$

Consider:

Axioms

$$s(X, Y, Z) \wedge s(X, Y, Z') \rightarrow Z = Z'$$

$$s(X, Y, a(X, Y))$$

$$s(z, X, X)$$

$$s(X, z, X)$$

$$s(X, m(X), z)$$

$$s(m(X), X, z)$$

$$s(X, Y, U) \wedge s(Y, Z, V) \wedge s(U, Z, W) \rightarrow s(X, V, W)$$

$$s(X, Y, U) \wedge s(Y, Z, V) \wedge s(X, V, W) \rightarrow s(U, Z, W)$$

First was GRP004+0.ax, second is GRP003-0.ax from TPTP.

Same theory modulo naming, **different axiomatizations**.

- First axiomatization is **easier** for superposition. . .
- But many problems use the second one.
- Can we **reduce** the second to the first?

- First axiomatization is **easier** for superposition. . .
- But many problems use the second one.
- Can we **reduce** the second to the first?

We can go from second to first by introducing definition

$$\text{sum}(X, Y, Z) \Leftrightarrow Z = \text{add}(X, Y)$$

then expansion+ simplification.

Abstracting this definition: $\text{sum}(X, Y, Z) \Leftrightarrow Z = \text{add}(X, Y)$

Theory of **total functions as relations**

- 1 detect **instances** of axioms
 - $P(X, Y, Z) \wedge P(X, Y, Z') \rightarrow Z = Z'$
 - $P(X, Y, F(X, Y))$
- 2 here, instance is $P \mapsto \text{sum}, F \mapsto \text{add}$
- 3 introduce definition of P : $P(X, Y, Z) \Leftrightarrow Z = F(X, Y)$
- 4 expand + simplify (eliminating P)

Our system has **two levels of discourse**:

- 1 The **prover** level
 - First order formulas/clauses
 - Try to find refutation of problem
- 2 The **meta-prover** level
 - Proves **properties** about problem
 - No contradiction, only facts about symbols
 - Humans also do that

Both can proceed “in parallel” and interact.

Proof process (eagle view)

prover

meta-prover

read problem clauses

⋮

add $x + y = y + x$

⋮

add $(x + y) + z = x + (y + z)$

⋮

enable redundancy criterion

→

read theory descriptions

add commutative(+)

add associative(+)

↓

deduce ac(+)

←

Summary

- 1 Introduction
- 2 Detection of axioms
- 3 Detecting theories with Datalog
- 4 Application

Theory parametrized by symbols

Ad-hoc format to describe theories:

- TPTP for formulas ($|$, $\&$, \sim , $!$, $?$, $=$ etc.)
- Declare axioms, theories, lemmas
- Function/predicate symbols are **bound**

```
% AC symbols theory
```

```
associative(f) is f(X, f(Y, Z)) = f(f(X, Y), Z).
```

```
commutative(f) is f(X, Y) = f(Y, X).
```

```
theory ac(f) is
```

```
    associative(f) and
```

```
    commutative(f).
```

→ f is a bound variable.

Axiom definitions can be reused:

```
% Monoid structure
```

```
leftIdentity(mult, e) is mult(e, X) = X.
```

```
rightIdentity(mult, e) is mult(X, e) = X.
```

```
theory monoid(mult, e) is  
  leftIdentity(mult, e) and  
  rightIdentity(mult, e) and  
  associative(mult).
```

Theories definitions can be reused too:

```
% Group structure  
  
leftInverse(mult, e, inverse) is  
  mult(inverse(X), X) = e.  
rightInverse(mult, e, inverse) is  
  mult(X, inverse(X)) = e.  
  
theory group(mult, e, inverse) is  
  monoid(mult, e) and  
  leftInverse(mult, e, inverse) and  
  rightInverse(mult, e, inverse).
```

Patterns

- Used to represent an axiom in **any** signature
- Pattern: **curried term** with 2 kinds of variables
 - **Symbol variables** : abstracted functions/predicates
 - **Proper variables** : variables of the clause
- Currying: helps replacing functions by variables

Patterns

- Used to represent an axiom in **any** signature
- Pattern: **curried term** with 2 kinds of variables
 - **Symbol variables** : abstracted functions/predicates
 - **Proper variables** : variables of the clause
- Currying: helps replacing functions by variables

Example:

- $\text{sum}(X, Y, Z) \wedge \text{sum}(X, Y, Z') \rightarrow Z = Z'$
- $\text{sum}(X, Y, \text{add}(X, Y))$

becomes:

- $((P\ X\ Y\ Z \wedge P\ X\ Y\ Z') \dot{\rightarrow} Z \doteq Z')$
- $(P\ X\ Y\ (F\ X\ Y))$

Matching

We match a pattern against a (beforehand curried) clause

Algorithm (sketch)

- 1 rename variables if needed
- 2 match terms (modulo AC for $\dot{\vee}$, $\dot{=}$, $\dot{\wedge}$...)
 - **Symbol variables** bind to any non-red term
 - **Proper variables** bind only to **proper variables**
- 3 Keep bindings for **Symbol variables**

We match a pattern against a (beforehand curried) clause

Algorithm (sketch)

- 1 rename variables if needed
- 2 match terms (modulo AC for $\dot{\vee}$, $\dot{=}$, $\dot{\wedge}$...)
 - **Symbol variables** bind to any non-red term
 - **Proper variables** bind only to **proper variables**
- 3 Keep bindings for **Symbol variables**

Example:

- Pattern $F (F X) \dot{=} X$ (involutivity)
- Clause $Y = \text{div}(1, \text{div}(1, Y))$
- Curried clause $Y \dot{=} \text{div } 1 (\text{div } 1 Y)$
- Yield: $F \mapsto (\text{div } 1)$

Summary

- 1 Introduction
- 2 Detection of axioms
- 3 Detecting theories with Datalog
- 4 Application

Recognize theories

A theory gathers several patterns, with **consistent binding** of symbols.

- Consider the theory:

```
associative(f) is f(X, f(Y, Z)) = f(f(X, Y), Z).
```

```
commutative(f) is f(X, Y) = f(Y, X).
```

```
theory ac(f) is associative(f) and commutative(f).
```

- If, in a problem, we detect instances
 - associative(add).
 - associative(mult).
 - commutative(add).
- Then we can deduce ac(add), but not ac(mult)
- Use **Datalog** to combine facts.

Datalog in a nutshell

- Fragment of First-Order logic
- Horn clauses, no function symbols
- $A \leftarrow B_1 \wedge B_2 \wedge \dots \wedge B_n$
- Restriction: $\forall v \in \text{vars}(A), \exists i, v \in \text{vars}(B_i)$
- Always a single minimal model (fixpoint semantics)
- **Efficient Computation** of fixpoint

Patterns do not belong to the Datalog fragment \rightarrow use **boxing**

Boxing

- boxing a term t is $\lceil t \rceil$
- unboxing $\cdot \mapsto \lfloor \cdot \rfloor$ is the inverse operation
- $\lceil t \rceil$ is a Datalog **constant** (mod. renaming)

Patterns do not belong to the Datalog fragment \rightarrow use **boxing**

Boxing

- boxing a term t is $\lceil t \rceil$
- unboxing $\cdot \mapsto \lfloor \cdot \rfloor$ is the inverse operation
- $\lceil t \rceil$ is a Datalog **constant** (mod. renaming)

Example (continued):

- Pattern $F (F X) \doteq X$
- Instance $F \mapsto (\text{div } 1)$
- Fact $\text{involutive}(\lceil \text{div } 1 \rceil)$.

Theories as Datalog clauses

```
associative(f) is f(f(X,Y),Z) = f(X,f(Y,Z)).
theory monoid(mult, e) is leftIdentity(mult, e)
    and rightIdentity(mult, e) and associative(mult).
theory group(mult, e, inverse) is
    monoid(mult, e) and
    mult(X, inverse(X)) = e.
```

transformed into one Datalog clause per definition:

```
associative(F) :- pattern([F (F X Y) Z ≐ F X (F Y Z)], F).
monoid(M,E) :- leftIdentity(M,E),
               rightIdentity(M,E), associative(M).
group(M,E,I) :-
    monoid(M,E),
    pattern([M X (I X) ≐ X], M, I).
```

Summary

- 1 Introduction
- 2 Detection of axioms
- 3 Detecting theories with Datalog
- 4 Application**

- **Lemmas:**

- A theorem that is used to prove another theorem.
- A lemma is also a Datalog clause!

We use this lemma in Zipperposition:

```
functional(p) is ~ p(X,Y,Z) | ~p(X,Y,Z2) | Z=Z2.  
total(p, f) is p(X,Y,f(X,Y)).  
totalFunction(p, f) is p(X,Y,Z) <=> Z = f(X,Y).  
lemma totalFunction(p, f)  
  if functional(p) and total(p, f).
```

- Ground convergent RW systems for redundancy [1]
- Choice of heuristics or term ordering.

[1] Avenhaus, Hillebrand and Löchner 2003

Example Run (RNG005-1.p)

```
$ zipperposition RNG005-1.p
% *** process file RNG005-1.p ***
% parsed 20 clauses
% meta-prover: axiom functional2(product)
% meta-prover: axiom functional2(sum)
% meta-prover: axiom total2(sum, add)
% meta-prover: lemma [(sum(X0, X1, X2) <=> (X2 = add(X0, X1)))]
% meta-prover: axiom total2(product, multiply)
% meta-prover: lemma [(product(X0, X1, X2) <=> (X2 = multiply(X0, X1)))]
% precedence: c > d > b > a > sum > add > product > multiply > additive_inverse > additive_identity > $false > $true
% selection function: SelectComplex
% meta-prover: axiom left_identity(add, additive_identity)
% meta-prover: axiom right_identity(add, additive_identity)
% meta-prover: axiom commutative(add)
% meta-prover: axiom left_inverse(add, additive_identity, additive_inverse)
% meta-prover: axiom right_inverse(add, additive_identity, additive_inverse)
% meta-prover: axiom left_distributive(multiply, add)
% meta-prover: axiom associative(multiply)
% meta-prover: axiom associative(add)
% meta-prover: theory monoid(add, additive_identity)
% meta-prover: theory ac(add)
% meta-prover: new gnd_convergent : ac(5 equations, ord rpo6(add))
% meta-prover: new gnd_convergent : monoid(7 equations, ord rpo6(add))
% meta-prover: theory group(add, additive_identity, additive_inverse)
% meta-prover: theory abelian_group(add, additive_identity, additive_inverse)
% meta-prover: new gnd_convergent : abelian_group(12 equations, ord rpo6(add>additive_inverse>additive_identity))
% =====
% done 134 iterations
% datalog contains 101 clauses
# SZ5 status Theorem
```


Prover	Proved (over 1047)
SPASS	863
zipperposition	531
zipperposition-no-theories	504

Figure : Number of Solved Problems

- 3 problems (e.g., GRP392-1.p) solved with meta-prover but not E nor SPASS
- still room for improvement (prototype)

Related work

- Waldmeister (unit eq): ordering+heuristics
- Discount (unit eq): heuristics
- Many provers, for AC

Future work

- Some higher-order matching: $f(X, Y, a) = f(Y, X, a)$ should yield $\text{commutative}(\lambda x.\lambda y.f(x, y, a))$.^a
- Interactions between distinct provers, through meta-prover

^aAlready the case for $f(a, X, Y) = f(a, Y, X)$

Questions?