Classifying groupoids and semirings of finite order using logic programming

Stefan Guettel, 2005 November 10

1 Introduction

Given a non-empty finite set $S = \{e_1, e_2, \ldots, e_m\}$ and a *binary operation* $\boldsymbol{B}$ on S, i.e. $\boldsymbol{B} : S \times S \to S$. For arbitrary elements $a, b, c \in S$ we write $a\boldsymbol{B}b = c$ instead of $\boldsymbol{B} : (a, b) \mapsto c$. The pair $(S, \boldsymbol{B})$ is called a *groupoid*, $m = \#S$ is the *order of* $(S, \boldsymbol{B})$.

Definitions.

- $\boldsymbol{B}$ is *associative* if $(a\boldsymbol{B}b)\boldsymbol{B}c = a\boldsymbol{B}(b\boldsymbol{B}c)$ for all $a, b, c \in S$.
- $\boldsymbol{B}$ is *commutative* if $a\boldsymbol{B}b = b\boldsymbol{B}a$ for all $a, b \in S$.
- $n \in S$ is an *identity element for* $\boldsymbol{B}$ if $n\boldsymbol{B}a = a\boldsymbol{B}n = a$ for all $a \in S$.
- $a' \in S$ is an *inverse (element) of* $a \in S$ *for* $\boldsymbol{B}$ if $a'\boldsymbol{B}a = a\boldsymbol{B}a' = n$, where $n \in S$ is the identity element for $\boldsymbol{B}$.
- $\boldsymbol{B}$ is *invertible* if for each element $a \in S$ there exists an inverse a'.

Note. The identity element n is uniquely determined. For associative $\boldsymbol{B}$ the inverse a' of an element a for $\boldsymbol{B}$ is also uniquely determined.

We are ready to define some algebraic structures using the above definitions.

Definitions.

- $(S, \boldsymbol{B})$ is a *semigroup* if $\boldsymbol{B}$ is associative.
- $(S, \boldsymbol{B})$ is a *group* if it is a semigroup and $\boldsymbol{B}$ is invertible.
- $(S, \boldsymbol{B})$ is an *abelian group* if it is a group and $\boldsymbol{B}$ is commutative.

Given another binary operation $\boldsymbol{B'}$ on S we may define more complicated algebraic structures.

Definition.

- $\boldsymbol{B}$ is *left-distributive over* $\boldsymbol{B'}$ if $a\boldsymbol{B}(b\boldsymbol{B'}c) = (a\boldsymbol{B}b)\boldsymbol{B'}(a\boldsymbol{B}c)$.
- $\boldsymbol{B}$ is *right-distributive over* $\boldsymbol{B'}$ if $(a\boldsymbol{B'}b)\boldsymbol{B}c = (a\boldsymbol{B}c)\boldsymbol{B'}(b\boldsymbol{B}c)$.
- $\boldsymbol{B}$ is *distributive over* $\boldsymbol{B'}$ if it is left- and right-distributive over $\boldsymbol{B'}$.

Definitions.

- $(S, \boldsymbol{B}, \boldsymbol{B'})$ is a *semiring* if $(S, \boldsymbol{B})$ and $(S, \boldsymbol{B'})$ are semigroups and $\boldsymbol{B}$ is distributive over $\boldsymbol{B'}$.
- $(S, \boldsymbol{B}, \boldsymbol{B'})$ is a *ring* if it is a semiring and $(S, \boldsymbol{B'})$ is an abelian group.
- $(S, \boldsymbol{B}, \boldsymbol{B'})$ is a *commutative ring* if it is a ring and $\boldsymbol{B}$ is commutative.
- $(S, \boldsymbol{B}, \boldsymbol{B'})$ is a *ring with identity* if it is a ring and $n \in S$ is the identity element for $\boldsymbol{B}$.
- $(S, \boldsymbol{B}, \boldsymbol{B'})$ is a *commutative ring with identity* if it is a commutative ring and $n \in S$ is the identity element for $\boldsymbol{B}$.
- Let $n' \in S$ be the identity element for $\boldsymbol{B'}$. Then $z \in S$, $z \neq n'$ is called a *zero divisor* if for some $a \neq n'$ there holds $z\boldsymbol{B}a = a\boldsymbol{B}z = n'$.
- $(S, \boldsymbol{B}, \boldsymbol{B'})$ is an *integrity domain* if it is a commutative ring with identity and without any zero divisors.
- $(S, \boldsymbol{B}, \boldsymbol{B'})$ is a *skew field* if it is a ring and $(S, \boldsymbol{B} \setminus \{n'\})$ is a group, where n' is the identity element for $\boldsymbol{B'}$.
- $(S, \boldsymbol{B}, \boldsymbol{B'})$ is a *field* if it is a skew field and $(S, \boldsymbol{B} \setminus \{n'\})$ is an abelian group.

Note. In case of finite order there holds: $(S, \boldsymbol{B}, \boldsymbol{B'})$ is an integrity domain $\Leftrightarrow$ $(S, \boldsymbol{B}, \boldsymbol{B'})$ is a skew field $\Leftrightarrow$ $(S, \boldsymbol{B}, \boldsymbol{B'})$ is a field. The last equivalence is known as *Wedderburns' theorem.*

2 Implementation in Prolog

The set S is defined as a collection of facts about its elements:

```
element(0).
element(1).
...
element(m).
```

Furthermore we have to declare and define the binary operations on S, in this example `sum` and `prod`.

```
declare_bin_op(sum).       // declaration
declare_bin_op(prod).

bin_op(sum,0,0,0).         // definition
bin_op(sum,0,1,1).
...
bin_op(sum,m,m,*).

bin_op(prod,0,0,0).
bin_op(prod,0,1,0).
...
bin_op(prod,m,m,*).
```

For each declared operation `op` the relation `bin_op` has to define a closed mapping on $S \times S$, i.e. for each `X` and `Y` that fulfill `element(X)`, `element(Y)` there must exist one and only one element `Z` such that `bin_op(op,X,Y,Z)`. We can check this using the following code:

```
bin_op_is_not_correct(Op) :- declare_bin_op(Op), element(X),
                             element(Y), \+ bin_op(Op,X,Y,_).
bin_op_is_not_correct(Op) :- bin_op(Op,_,_,Z), \+ element(Z).
bin_op_is_not_correct(Op) :- bin_op(Op,X,Y,Z1), bin_op(Op,X,Y,Z2),
                             Z1\=Z2.
bin_op_is_correct(Op) :- declare_bin_op(Op),
                         \+ bin_op_is_not_correct(Op).
```

Here we cannot avoid to make extensive use of negation `\+`. As another example we could check a binary operation for being commutative:

```
bin_op_is_not_commutative(Op,X,Y) :- bin_op(Op,X,Y,Z1),
                                     bin_op(Op,Y,X,Z2), Z1\=Z2.
bin_op_is_commutative(Op) :- declare_bin_op(Op),
                             \+ bin_op_is_not_commutative(Op,_,_).
```

Unfortunately, there is no other way for testing a binary operation to be commutative than comparing the results of $a\boldsymbol{B}b$ and $b\boldsymbol{B}a$ for all $a, b \in S$. Existence $\exists$ is much easier proven in logic programming than assertions with $\forall$-quantifiers (therefore it might be easier to prove that an operation is not commutative). At the end we cannot avoid to make use of logic negation, exploiting the following logic equivalence:

There holds $a\boldsymbol{B}b = b\boldsymbol{B}a$ **for all** $a, b \in S$
if and only if
there exist no $a, b \in S$ such that $a\boldsymbol{B}b \neq b\boldsymbol{B}a$.

The relations `bin_op_is_associative(Op)`, `bin_ops_are_distributive(Op1,Op2)`, etc. are similarly implemented.

To check if a binary relation defines a groupoid or a semigroup we state

```
groupoid(Op)  :- bin_op_is_correct(Op).
semigroup(Op) :- groupoid(Op), bin_op_is_associative(Op).
```

The complete listing of the Prolog-program is given in section 4.

3 Examples

A commutative groupoid

Definition.

```
element(a).
element(b).

declare_bin_op(sum).

bin_op(sum,a,a,b). bin_op(sum,a,b,b).
bin_op(sum,b,a,b). bin_op(sum,b,b,a).
```

Queries.

```
?- bin_op_is_associative(X).
   No.
?- bin_op_is_commutative(X).
   X = sum ;
   No.
```

$\mathbb{Z}$ mod 3

Definition.

```
element(0). element(1). element(2).

declare_bin_op(sum).
declare_bin_op(prod).
```

```
bin_op(sum,0,0,0). bin_op(sum,0,1,1). bin_op(sum,0,2,2).
bin_op(sum,1,0,1). bin_op(sum,1,1,2). bin_op(sum,1,2,0).
bin_op(sum,2,0,2). bin_op(sum,2,1,0). bin_op(sum,2,2,1).
bin_op(prod,0,0,0). bin_op(prod,0,1,0). bin_op(prod,0,2,0).
bin_op(prod,1,0,0). bin_op(prod,1,1,1). bin_op(prod,1,2,2).
bin_op(prod,2,0,0). bin_op(prod,2,1,2). bin_op(prod,2,2,1).
```

Queries.

```
?- comm_ring_with_identity(Op1,Op2).
   Op1 = prod
   Op2 = sum ;
   No.
?- el_is_identity(prod,N).
   N = 1 ;
   No.
?- field(prod,sum).
   Yes.
```

$\mathbb{Z}$ mod 4

Definition.

```
element(0). element(1). element(2). element(3).

declare_bin_op(sum).
declare_bin_op(prod).

bin_op(sum,X,Y,Z)  :- element(X), element(Y),
                      element(Z), (X+Y) mod 4 =:= Z.
bin_op(prod,X,Y,Z) :- element(X), element(Y),
                      element(Z), (X*Y) mod 4 =:= Z.
```

Queries.

```
?- comm_ring_with_identity(Op1,Op2).
   Op1 = prod
   Op2 = sum ;
   No.
?- el_is_identity(prod,N).
   N = 1 ;
   No.
?- field(prod,sum).
   No.
```

This might be a good place to give the following

Theorem. $\mathbb{Z}$ mod n is a field if and only if n is prime.

A field of non-prime order

Definition.

sum	0	1	a	a+1
0	0	1	a	a+1
1	1	0	a+1	a
a	a	a+1	0	1
a+1	a+1	a	1	0

prod	0	1	a	a+1
0	0	0	0	0
1	0	1	a	a+1
a	0	a	a+1	1
a+1	0	a+1	1	a

Queries.

```
?- field(prod,sum).
   Yes.
?- inverse(sum,a,N).
   N = a ;
   No.
?- inverse(prod,a+1,N).
   N = a ;
   No.
```

Boolean semigroups

Definition.

```
element(f).
element(t).

declare_bin_op(and). declare_bin_op(or). declare_bin_op(xor).

bin_op(and,f,f,f). bin_op(and,f,t,f).
bin_op(and,t,f,f). bin_op(and,t,t,t).
bin_op(or,f,f,f).  bin_op(or,f,t,t).
bin_op(or,t,f,t).  bin_op(or,t,t,t).
bin_op(xor,f,f,f). bin_op(xor,f,t,t).
bin_op(xor,t,f,t). bin_op(xor,t,t,f).
```

Queries.

```
?- el_is_identity(Op,N).
   Op = and
   N = t ;
   Op = or
   N = f ;
   Op = xor ;
   N = f ;
   No.
?- group(Op).
   Op = xor ;
   No.
?- ring(Op1,Op2).
   Op1 = and
   Op2 = xor ;
   No.
```

4 Listing

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Define Algebraic Structure (Elements, Operations)
%%% here: Z mod 7
%%%
element(X) :- member(X,[0,1,2,3,4,5,6]).

declare_bin_op(sum). declare_bin_op(prod).

bin_op(sum,X,Y,Z)  :- element(X), element(Y),
                      element(Z), (X+Y) mod 7 =:= Z.
bin_op(prod,X,Y,Z) :- element(X), element(Y),
                      element(Z), (X*Y) mod 7 =:= Z.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Some useful relations
%%%
member(X,[X|_]).
member(X,[_|L2]) :- member(X,L2).

pair(X,Y) :- element(X), element(Y).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Check binary operation if being properly defined
%%%
bin_op_is_not_correct(Op) :- declare_bin_op(Op), pair(X,Y),
                             \+ bin_op(Op,X,Y,_).
bin_op_is_not_correct(Op) :- bin_op(Op,_,_,Z), \+ element(Z).
bin_op_is_not_correct(Op) :- bin_op(Op,X,Y,Z1), bin_op(Op,X,Y,Z2),
                             Z1\=Z2.

bin_op_is_correct(Op) :- declare_bin_op(Op),
                         \+ bin_op_is_not_correct(Op).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Check for associativity, i.e. (A Op B) Op C = A Op (B Op C)
%%%
bin_op_is_not_associative(Op,A,B,C) :-
                             bin_op(Op,A,B,D1), bin_op(Op,D1,C,E1),
                             bin_op(Op,B,C,D2), bin_op(Op,A,D2,E2),
                             E1\=E2.

bin_op_is_associative(Op) :- declare_bin_op(Op),
                             \+ bin_op_is_not_associative(Op,_,_,_).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Check for left/right distributivity
%%% Left distributivity: A Op1 (B Op2 C) = (A Op1 B) Op2 (A Op1 C)
%%% Right distribtivity: (A Op2 B) Op1 C = (A Op1 C) Op2 (B Op1 C)
%%%
bin_ops_are_not_left_distributive(Op1,Op2,A,B,C) :-
                bin_op(Op2,B,C,D1), bin_op(Op1,A,D1,E1),
                bin_op(Op1,A,B,F1), bin_op(Op1,A,C,F2),
                bin_op(Op2,F1,F2,E2), E1\=E2.

bin_ops_are_not_right_distributive(Op1,Op2,A,B,C) :-
                bin_op(Op2,A,B,D1), bin_op(Op1,D1,C,E1),
                bin_op(Op1,A,C,F1), bin_op(Op1,B,C,F2),
                bin_op(Op2,F1,F2,E2), E1\=E2.

bin_ops_are_left_distributive(Op1,Op2)  :-
                declare_bin_op(Op1), declare_bin_op(Op2),
                \+ bin_ops_are_not_left_distributive(Op1,Op2,_,_,_).

bin_ops_are_right_distributive(Op1,Op2) :-
                declare_bin_op(Op1), declare_bin_op(Op2),
                \+ bin_ops_are_not_right_distributive(Op1,Op2,_,_,_).

bin_ops_are_distributive(Op1,Op2) :-
                bin_ops_are_left_distributive(Op1,Op2),
                bin_ops_are_right_distributive(Op1,Op2).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Check binary operation for commutativity, i.e. X Op Y = Y Op X.
%%%
bin_op_is_not_commutative(Op,X,Y) :- bin_op(Op,X,Y,Z1),
                                     bin_op(Op,Y,X,Z2), Z1\=Z2.
bin_op_is_commutative(Op) :- declare_bin_op(Op),
                             \+ bin_op_is_not_commutative(Op,_,_).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Check binary operation for an identity element
%%% and define the inverse for an element
%%%
el_is_identity(Op,N) :- declare_bin_op(Op), element(N),
                        \+ (bin_op(Op,X,N,Z), X\=Z),
                        \+ (bin_op(Op,N,Y,Z), Y\=Z).

inverse(Op,X,Y) :- declare_bin_op(Op), pair(X,Y),
                   el_is_identity(Op,N), bin_op(Op,X,Y,N).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Check binary operation if being invertible,
%%% one element can be exluded, otherwise Exc=undef
%%%
bin_op_is_not_invertible(Op,Exc) :- declare_bin_op(Op),
                                    element(X), X\=Exc,
                                    \+ inverse(Op,X,_).
bin_op_is_invertible(Op,undef) :- declare_bin_op(Op),
                                  \+ bin_op_is_not_invertible(Op,undef).
bin_op_is_invertible(Op,Exc):- declare_bin_op(Op), element(Exc),
                                  \+ bin_op_is_not_invertible(Op,Exc).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% ALGEBRAIC CLASSIFICATION
%%%
groupoid(Op) :- bin_op_is_correct(Op).

semigroup(Op) :- groupoid(Op), bin_op_is_associative(Op).

group(Op) :- bin_op_is_invertible(Op,undef),semigroup(Op).

abelgroup(Op) :- group(Op), bin_op_is_commutative(Op).


semiring(Op1,Op2) :- semigroup(Op1), semigroup(Op2),
                     bin_ops_are_distributive(Op1,Op2).

ring(Op1,Op2) :- abelgroup(Op2), semigroup(Op1),
                 bin_ops_are_distributive(Op1,Op2).

comm_ring(Op1,Op2) :- ring(Op1,Op2),
                      bin_op_is_commutative(Op1).

ring_with_identity(Op1,Op2) :- ring(Op1,Op2),
                               el_is_identity(Op1,_).

comm_ring_with_identity(Op1,Op2) :- comm_ring(Op1,Op2),
                                    el_is_identity(Op1,_).

field(Op1,Op2) :- semigroup(Op1), abelgroup(Op2),
                  bin_op_is_commutative(Op1),
                  bin_ops_are_left_distributive(Op1,Op2),
                  bin_op_is_invertible(Op1,_).


%%% EOF
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

References

[1] E. Z. Goren. *Group theory and introduction to rings.* Course notes, McGill University 2005.

[2] U. Hebisch. *Universelle Algebra.* Course notes, TU Bergakademie Freiberg 2002.

[3] E. Sterling, L. Shapiro. *The Art of Prolog.* MIT Press, Massachusetts 1994.