Библиотека теории групп в системе автоматического доказательства Why3

Описывается библиотека теории групп, представленная в файле Group1.mlw в виде набора теорий на языке WhyML. Библиотека с доказательствами некоторых лемм в системе Why3 доступна по адресу: https://persons.iis.nsk.su/files/persons/pages/group1.zip . Библиотека построена на базе стандартной библиотеки теории групп космического агенства NASA для системы автоматического доказательства PVS и специальной библиотеки теории групп для системы доказательства Coq. Данную библиотеку предполагается использовать для обучения студентов методам автоматического доказательства на компьютере теорем из области математики в системе автоматического доказательства Why3: http://why3.lri.fr.

Система доказательства Why3 намного проще известных систем интерактивного доказательства Coq и PVS. Она универсальна и потенциально применима для доказательства любых математических теорем, хотя изначально ориентирована на дедуктивную верификацию программ. Математические теоремы формулируются на языке формул функционального языка программирования WhyML. Студенты будут строить доказательства теорем с помощью команд системы Why3 в интерактивном режиме на компьютере. Огромное преимущество Why3 перед другими подобными системами в том, что в любой ветви процесса доказательства из Why3 можно запустить около 20 других систем доказательства, автоматических и интерактивных. Почти все системы автоматического доказательства работают в логике первого порядка. Чтобы запуск автоматических систем из Why3 был возможен, язык WhyML базируется на непараметрической логике первого порядка. Создатели системы Why3 много сделали для достижения ее универсальности. В частности, простые функции могут быть параметрами формул.

Студентам необходимо установить систему Why3 на свой компьютер. Дистрибутив для Windows доступен из https://persons.iis.nsk.su/files/persons/fmcfiles/why3cdecl.pdf. Там же видеолекции для самостоятельного освоения системы Why3. В дистрибутиве подключены четыре инструмента автоматического доказательства: Z3 4.8.6, CVC4 1.7, CVC3 2.4.1 и Vampire 4.2.2, которые можно запустить в процессе доказательства на Why3.

Далее описывается библиотека теории групп для системы Why3. В конце сформулированы леммы Бернсайда конечности групп <a, b>, порожденных двумя элементами a и b, при условии, что порядок всех элементов группы равен 2 или 3. Леммы Бернсайда не входят в состав библиотеки. Они приведены в качестве потенциальных более сложных задач. Для них необходимо расширение библиотеки промежуточными теориями.

Библиотека теории групп организована в виде дерева теорий. Каждая теория – модуль на языке WhyML с набором определений типов, констант, аксиом, функций, предикатов и лемм (теорем). С помощью декларации use теория может импортировать другую теорию, и тогда все объекты импортируемой теории становятся доступными в данной теории. Другой способ связи между теориями с помощью декларации clone, предполагающей копирование (клонирование) указанной теории внутрь данной с возможной заменой неинтерпретированных объектов. В руководстве по Why3 описание clone неполное; детальное описание в статье: https://hal.inria.fr/hal-02696246/document .

Головная теория библиотеки определяется модулем Group_def, клонирующим теорию Group из стандартной библиотеки Why3: http://why3.lri.fr/stdlib/algebra.html с введением других обозначений: e для единицы группы и инфиксной операции * для бинарной операции группы. Также копируются все аксиомы цепочки клонируемых модулей библиотеки algebra. Предельная лаконичность и компактность библиотеки algebra вызывает восхищение. Однако создатели Why3 построили библиотеку algebra не из любви к алгебре, а потому что определение целых чисел в виде коммутативного кольца повышает эффективность SMT-решателей, запускаемых из Why3.

```
module Group_def
  use int.Int

  type t
  constant e : t
  function (*) t t : t
  function inv t : t

  clone export algebra.Group   with type t = t, constant unit = e,
                     function op = (*), function inv = inv,  axiom .
end (*Group_def*)
```

Некоторые из многочисленных свойств бинарной операции * и унарной операции инверсии inv, т.е. взятия обратного элемента, приведены в модуле Group_lems. Подавляющее большинство лемм заимствованы из стандартной библиотеки NASA для PVS и лишь некоторые из специальной библиотеки для системы Coq.

Зеленый маркер в начале леммы указывает, что лемма доказана автоматически запуском одного из четырех автоматических инструментов доказательства: Z3 4.8.6, CVC4 1.7, CVC3 2.4.1 и Vampire 4.2.2. Синий маркер в начале леммы означает, что лемма не доказана, причем она не доказывается автоматически указанными решателями. Леммы с синим маркером предназначены для доказательства студентами применением команд системы Why3. Красный маркер указывает, что лемма доказана применением команд и решателей. Доказательство леммы с красным маркером – это образец техники доказательства для студентов.

Леммы модуля Group_lems в принципе можно было поместить в составе головного модуля Group_def. Однако это не улучшит, а наоборот, затормозит работу автоматических решателей. Обратите внимание, что в библиотеке algebra нет лемм. Почти нет. На основной магистрали работы решателей, то есть в области, доступной по импорту, не должно быть посторонних малозначимых, бесполезных для решателей лемм.

```
module Group_lems
  use int.Int
  use Group_def

• lemma Cancel_right:     forall x y z: t. x*z = y*z <-> x = y
• lemma Cancel_left:      forall x y z: t. z*x = z*y <-> x = y
• lemma Inv_inv:          forall x: t. inv (inv x) = x
• lemma Cancel_right_inv: forall x y z: t. x*(inv z) = y*(inv z) <-> x = y
• lemma Cancel_left_inv:  forall x y z: t. z*(inv x) = z*(inv y) <-> x = y

• lemma Inv_one:          inv(e) = e
• lemma Inv_star:         forall x y: t. inv (x*y) = (inv y)*(inv x)
• lemma Inv_triv1:        forall x y: t. (inv x) * (x * y) = y
• lemma Inv_triv2:        forall x y: t. (y * x) * (inv x) = y
• lemma Inv_resolve:      forall x y: t. y * x = e -> y = inv x
• lemma Cancel_triv:      forall x y: t. x * y = x -> y = e
• lemma Unique_inv:       forall x y: t. x*y = e /\ y*x = e <-> y = inv x

• lemma Divby:            forall x y z: t. x = y * z <-> (inv y) * x = z

• lemma Unique_left_identity:   forall x y: t. y*x = x <-> y = e
• lemma Unique_right_identity:  forall x y: t. x*y = x <-> y = e
end (*Group_lems*)
```

Теория модуля Exponential с помощью трех аксиом вводит инфиксную операцию ^ возведения элемента группы в степень, в том числе и отрицательную. Данный модуль – калька соответствующего модуля библиотеки int в системе Why3 для целых чисел. Леммы модуля Exponential присутствовали в исходном модуле. Возможно, некоторые из этих лемм следует переместить в следующий модуль Expt_lems. Определение отрицательной степени заимствовано из библиотеки NASA для PVS.

Известно, что решатели Z3 и CVC4 способны проводить доказательство по индукции. Однако редко достигают успеха. Для последних трех лемм технику доказательства по индукции предстоит продемонстрировать студентам. Образцом такой техники является доказательство леммы Power_sum.

Порядком элемента группы a является минимальное неотрицательное число n, при возведении в которое получается единица группы e, т.е. a^n = e. Однако порядок не всегда существует. Например, его нет для элементов группы целых чисел. Предикат finite_order в конце теории Exponential является истинным, если порядок элемента группы существует и равен некоторому конечному числу.

```
module Exponential
  use int.Int

  clone export Group_def  with axiom .

  function (^) t int : t
  axiom Power_0 : forall x: t. x ^ 0 = e
  axiom Power_s : forall x: t, n: int. n >= 0 ->  x^(n + 1) = x * x^n
  axiom Power_inv : forall x: t, n: int. n < 0 ->  x^n = (inv x)^(- n)

• lemma Power_s_alt: forall x: t, n: int. n > 0 -> x^n = x * (x^(n-1))
• lemma Power_1 : forall x : t. x ^ 1 = x

• lemma Power_sum : forall x: t, n m: int. 0 <= n -> 0 <= m ->
        x ^ (n+m) = (x ^ n) * (x ^ m)
• lemma Power_mult : forall x:t, n m : int. 0 <= n -> 0 <= m ->
        x ^ (Int.(*) n m) = (x ^ n) ^ m

• lemma Power_comm1 : forall x y: t. x * y = y * x ->
       forall n: int. 0 <= n ->  (x ^ n) * y = y * (x ^ n)
• lemma Power_comm2 : forall x y: t. x * y = y * x ->
    forall n: int. 0 <= n ->   (x * y) ^ n = (x ^ n) * (y ^ n)

  predicate finite_order (a: t) = exists n: int. n>=0 /\ a ^ n = e

end (*Exponential*)
```

Почти все леммы модуля Expt_lems заимствованы из библиотеки NASA для PVS. Последние две леммы взяты из библиотеки int в системе Why3 для целых чисел.

Показательно, что все леммы модулей Group_lems и Expt_lems, кроме одной, доказаны автоматически, одним щелчком мыши. Доказательство этих простых лемм в системах PVS и Coq для модулей Group_lems, Exponential и Expt_lems потребует от трех до пяти дней работы. Это реально подтверждает существенное преимущество Why3 перед другими системами.

Доказательство набора лемм (или группы подцелей после применения команды split) начинается запуском автоматических решателей. Сначала выпускаем Vampir'a. Он быстрее остальных и если не доказывает, то обычно быстро заканчивает работу. Потом всех остальных, которые работают только по недоказанным целям. Сильнее всех CVC4. Однако нередко лучшим оказывается Z3, а иногда и CVC3. Можно запустить все решатели на группу целей. Они работают в параллельном режиме, используя все процессоры вашего компьютера. Если ни один из решателей не смог доказать некоторую цель, дальнейшее доказательство цели возможно лишь применением студентом команд Why3 в интерактивном режиме.

```
module Expt_lems
   use int.Int
   use Exponential
   use Group_lems

•  lemma Inv_power:       forall a: t, m: int. inv (a^m) = (inv a)^m
•  lemma Power_inv_right: forall a: t, m: int. a^m * (inv a)^m = e
•  lemma Power_inv_left:  forall a: t, m: int. (inv a)^m * a^m = e

•  lemma Expt_0:        forall a: t. a^0     = e
•  lemma Expt_1:        forall a: t. a^1     = a
•  lemma Expt_m1:        forall a: t. a^(-1)  = inv a
•  lemma One_expt:       forall i: int. e^i   = e
•  lemma Expt_neg:       forall a: t, i: int. a^(-i)           = (inv a)^i
•  lemma Inv_expt:       forall a: t, i: int. inv (a^i)        = (inv a)^i
•  lemma Expt_def1:      forall a: t, i: int. a^(i+1)           = (a^i)*a
•  lemma Expt_def2:      forall a: t, i: int. a^(i+1)           = a*(a^i)
•  lemma Expt_mult:      forall a: t, i j: int. (a^i)*(a^j)      = a^(i+j)
•  lemma Expt_div :      forall l a: t, i j: int. (a^i)*(inv a)^j = a^(i-j)
•  lemma Expt_expt:      forall a: t, i j: int. (a^i)^j   = a ^ (Int.(*) i j)
•  lemma Expt_commutes:  forall a: t, i j: int. (a^i)*(a^j)      = (a^j)*(a^i)
•  lemma Expt_inv_right: forall a: t, i: int. a^i * (inv a)^i   = e
•  lemma Expt_inv_left:  forall a: t, i: int. (inv a)^i * a^i   = e

  use int.ComputerDivision
•  lemma Power_even : forall x:t, n: int. n >= 0 -> mod n 2 = 0 ->
       x ^ n = (x * x) ^ (div n 2)
•  lemma Power_odd : forall x:t, n: int. n >= 0 -> mod n 2 <> 0 ->
     x ^ n = x * ((x * x) ^ (div n 2))
end (*Expt_lems*)
```

Принципиально то, что даже в системе Coq нет средств определения конечности типа. По этой причине в системах PVS и Coq группа определяется сразу на множествах. Конечность множества там задается предикатом is_finite. В системе Why3 пока не требовалось определения конечности групп. Поэтому в Why3 множество элементов группы представлено неинтерпретированным типом t. Однако нам необходимо доказывать конечность групп. Поэтому понятие группы далее доопределяется для подмножеств исходного типа t.

Множества и операции с ними определены в библиотеке set системы Why3. Тип множества подмножеств в языке WhyML определяется конструкцией: set u, где u – некоторый базисный тип. Модуль FiniteSet содержит необходимые нам расширения библиотеки set. Функция order s определяет число элементов множества при условии, что множество s конечно. Множество из двух элементов определяется функцией couple. В библиотеке set функция singleton определяет множество из одного элемента, константа empty – пустое множество, константа all – супермножество (полное множество), предикат mem a t – принадлежность элемента a множеству t. Лемма

Finite_surj является обобщением аналогичной леммы из библиотеки set. Сюръективная функция имеет тип map u.

```
module FiniteSet
  use int.Int, set.Set, set.Cardinal, map.Map

  type t

  let ghost function order (s: set t) : int
      requires { is_finite s}
   =   cardinal s

  function couple (x y: t): set t = add y (singleton x)
• lemma Couple_def: forall x y z: t. mem z (couple x y) <-> z = x \/ z = y

  predicate surjective (s: set t) (a: map int t) (n: int) = n > 0 /\
        forall x: t. mem x s -> exists j: int. (0 <= j < n /\ a[j] = x)

• lemma Finite_surj: forall s: set t.
         (exists n: int, f: map int t. n>0 /\ surjective s f n) -> is_finite s
end (*FiniteSet*)
```

Предикат group g истинен, если подмножество g является группой. Необходимым условием этого является замкнутость относительно операций * и inv.

```
module GroupSub
  use int.Int, set.Set, set.Cardinal
  clone export Exponential with axiom .

  predicate g_star (g : set t) = forall x y: t. mem x g /\ mem y g -> mem (x*y) g
  predicate g_inv (g : set t) = forall x: t. mem x g  -> mem (inv x) g
  predicate group (g : set t) = mem e g /\ g_star g /\ g_inv g

• lemma Gr_pow: forall g : set t. group g ->
                forall x: t, n: int. mem x g  -> mem (x ^ n) g
end (*GroupSub*)
```

Модуль GroupSub_lem представляет набор очевидных лемм из библиотеки NASA для PVS, вынесенных из основной магистрали.

```
module GroupSub_lem
  use int.Int, set.Set, set.Cardinal
  use export GroupSub
  clone FiniteSet with type t=t
```

- lemma GrAll: group(all)
- lemma GrE: group(singleton e)
- lemma GrFinE: is_finite (singleton e)

- lemma Inv_exists: forall g : set t. group g ->
    forall x: t. mem x g -> exists y: t. mem y g /\ x*y = e /\ y*x = e
- lemma Inv_member: forall g : set t. group g ->
                          forall x: t. mem (inv x) g <-> mem x g
- lemma Order_is_1: forall s: set t.
          group s /\ is_finite s /\ order s = 1 -> s = singleton e

```
end (*GroupSub_lem*)
```

Функция genSet s модуля GroupGen определяет множество элементов, порожденных многократным последовательным применением операций * и inv к элементам исходного подмножества s. Этому модулю нет аналогов. В библиотеке NASA для PVS определяется лишь замыкание одноэлементного множества относительно операции ^. Функция gen0 определяет один шаг замыкания исходного множества s относительно операций * и inv. Предикат genN s n w истинен, если множество w получается последовательным применением n шагов замыкания.

Определение функций gen0 и genSet дается через постусловие – формулу, связывающую аргументы функции и ее результирующее значение, обозначаемое в формуле стандартной переменной result. Рекурсивный предикат genN задается здесь определением inductive, заимствованным Why3 из Coq, к сожалению, без описания в руководстве. Предикат genN определяется как дизъюнкция одной из ветвей G0 и GN. Предикат genN s 0 s утверждает, что на нулевом шаге результатом является исходное множество s. Предикат genN s p s2 истинен, если истинны посылки p>0 /\ genN s (p-1) s1 /\ s2 = gen0 s1, т.е. s2 получается применением gen0 к множеству, порожденному за p-1 шагов.

```
module GroupGen
  use int.Int, set.Set, set.Cardinal
  clone export GroupSub with axiom .

  val function gen0 (s: set t): set t
     ensures { forall y: t. mem y result <->  mem y s \/
                      exists a: t. mem a s /\ y = inv a \/
                      exists a b: t. mem a s /\ mem b s /\ y = a*b }

  inductive genN (set t) int (set t) =
     | G0:   forall s: set t. genN s 0 s
     | GN:   forall s s1 s2: set t, p : int. p>0 /\
                      genN s (p-1) s1 /\ s2 = gen0 s1 -> genN s p s2

  val function genSet (s: set t): set t
     ensures { forall y: t. mem y result <->
                   exists sn: set t, n: int. genN s n sn /\ mem y sn }

end (*GroupGen*)

module GroupGen_lems
  use int.Int, set.Set, set.Cardinal
  use GroupGen

• lemma Gen_emp:  genSet empty = empty
• lemma Gen_set:  forall g s: set t.
            group g /\ subset s g /\ not is_empty s ->   group (genSet s)
• lemma Gen_single:  forall g: set t, a: t. group g /\ mem a g ->
             group (genSet (singleton a))
• lemma Gen_sub:  forall g s: set t. group g /\ subset s g ->
                              subset (genSet s) g
• lemma Gen_is_finite: forall a: t. finite_order a ->
                                is_finite (genSet (singleton a))
end (*GroupGen_lems*)
```

Теория модуля GroupCenter для функции center, экстрагирующей коммутативную часть группы, построена как аналог соответствующей функции в библиотеке NASA для PVS.

```
module GroupCenter
  use int.Int, set.Set, set.Cardinal
  use GroupSub

  val function center (g: set t): set t
     requires {group g}
     ensures { forall y: t. mem y result <-> (forall x: t. mem x g -> x*y = y*x)}
```

- ```
  lemma Center_def: forall g: set t. group g ->
     forall x: t. mem x (center g) <-> mem x g /\ forall y: t. mem y g -> y*x = x*y
  ```

- ```
  lemma Center_subgroup:  forall g: set t. group g -> subset (center g) g
  ```

```
end (*GroupCenter*)
```

Наконец, леммы Бернсайда. Множество b2 порождается двумя элементами x и y. В посылках лемм B22, B23 и B24 порядки всех элементов равны, соответственно, 2, 3 и 4. В леммах Bm2 и Bm3 группа bm порождается из исходного множества s размера m.

```
module BurnsideLems
  use int.Int, set.Set, set.Cardinal
  use GroupGen
  clone FiniteSet with type t=t

  constant x: t
  constant y: t
  constant b2: set t = genSet ( couple x y)
```

- ```
  lemma B22:  (forall a: t. mem a b2 -> a^2 = e) -> is_finite b2
  ```

- ```
  lemma B23:  (forall a: t. mem a b2 -> a^3 = e) -> is_finite b2
  ```

- ```
  lemma B24:  (forall a: t. mem a b2 -> a^4 = e) -> is_finite b2
  ```

```
  constant  s: set t
  constant m: int
  axiom SetM: is_finite s /\ m>0 /\ order s = m
  constant bm: set t = genSet s
```

- ```
  lemma Bm2:  (forall a: t. mem a bm -> a^2 = e) -> is_finite bm
  ```

- ```
  lemma Bm3:  (forall a: t. mem a bm -> a^3 = e) -> is_finite bm
  ```

```
end (*BurnsideLems*)
```
```