Computation Theory with Atoms

Bartek Klin, University of Warsaw

Warsaw, 27-29 April, 2016

V

Programming with Atoms

Warsaw, 27/04/2017

Programming with finite sets

Haskell syntax used

```
type Set a = [a]

empty :: Set a

insert :: a -> Set a -> Set a

map :: (a -> b) -> Set a -> Set b

filter :: (a -> Bool) -> Set a -> Set a

sum :: Set (Set a) -> Set a

...
```

Example: transitive closure

```
comp :: Set (a,b) -> Set (b,c) -> Set (a,c)
comp s r = ...

transCl :: Set (a,a) -> Set (a,a)
transCl r =
    let r1 = comp r r in
    if isSubsetOf r1 r
    then r
    else transCl (union r1 r)
```

```
> transCl [(1,2),(2,3)]
[(1,2),(2,3),(1,3)]
```

Other examples

1. Graph 2-colorability

```
twoColorable :: Set (a,a) -> Bool
```

- look for cycles of odd length

2. Graph 3-colorability

```
threeColorable :: Set (a,a) -> Bool
```

- generate all 3-partitions of vertices
- for each of them, check legality

NLambda: a Haskell library

```
type Atom

type Set a = ...

empty :: Set a

atoms :: Set Atom

insert :: a -> Set a -> Set a

map :: (a -> b) -> Set a -> Set b

sum :: Set (Set a) -> Set a

isEmpty :: Set a -> Formula

...
```

Example

```
> atoms
{a : for a in A}
> map (\a -> map (\b -> (a,b)) atoms) atoms
{{(a,b) : for b in A} : for a in A}
> sum it
{(a,b) : for a,b in A}
> filter (\(a,b) -> eq a b) it
{(a,a) : for a in A}
> forAll (\a -> member a atoms) atoms
True
```

NLambda: a Haskell library

```
type Atom

type Set a = [(a,[Var],Formula)]

empty :: Set a

atoms :: Set Atom

insert :: a -> Set a -> Set a

map :: (a -> b) -> Set a -> Set b

sum :: Set (Set a) -> Set a

isEmpty :: Set a -> Formula

...
```

Semantics

- Orbit-finite sets internally represented by FO formulas and set-builder expressions

- Condition evaluation delayed when possible:

```
> ite (eq a b) (singleton c) atoms
{c : a=b, d : a!=b for d in A}
```

- Formulas evaluated by calling an SMT solver

```
> isEmpty atoms
False
```

Example: transitive closure

```
comp :: Set (a,b) -> Set (b,c) -> Set (a,c)
comp s r = ...

transCl :: Set (a,a) -> Set (a,a)
transCl r =
    let r1 = comp r r in
    ite (isSubsetOf r1 r)
        r
        (transCl (union r1 r))
```

The same code!*

*essentially

Other examples

- Graph 2-colorability

```
twoColorable :: Set (a,a) -> Bool
```

- Angluin algorithm for automata learning
  - interact with a teacher to learn an automaton
  - Moerman, Sammartino, Silva, K., Szynwelski: *Learning nominal automata*, POPL'17

Also the same code*

*essentially

3-colorability

```
threeColorable :: Set (a,a) -> Bool
```

- generate all 3-partitions of vertices ...

Cannot be done!

Different code:

- if coloring exists then an equivariant one exists
- generate 3-partitions of orbits ...

```
supports :: NType a => [Atom] -> a -> Bool
orbits   :: NType a => Set a -> Set (Set a)
```

...

Ordered atoms needed

The easy, the hard & the impossible

Easy: code copied verbatim*

*essentially

```
transCl :: Set (a,a) -> Set (a,a)
twoColorable :: Set (a,a) -> Bool
learnAngluin :: ...
```

Hard: supports, orbits etc. required

```
threeColorable :: Set (a,a) -> Bool
```

Impossible: atom enumeration

```
toList :: Set a -> [a]
foldl :: (b -> a -> b) -> b -> Set a -> b
```

NLambda

Available at:

`http://www.mimuw.edu.pl/~szynwelski/nlambda/`

Ongoing work:

- closer integration with Haskell
- LOIS: the same thing, but for C++

  (Kopczyński, Toruńczyk: POPL'17)
- understand the hard crowd