1 Encoding

Given a type system with n qualifiers, and a slot whose ID is $slotId$. Then we can use an integer value range from $((slotId-1)*n+1)$ to $((slotId-1)*n+n)$ represent each type qualifier of that slot.

For example, if we consider about the universe type system, the representation is showing in following table:

$Qualifier$ / $SlotId$	peer	rep	lost	any
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
...				
9	33	34	35	36
...				
12	45	46	47	48

In the above table, every number in the middle represents the corresponding slot is annotated corresponding qualifier. We call those numbers *variables*. For example 6 means slot 2 is annotated qualifier rep.

Now, we will combine this table with the constraints generated from *Checker Framework Inference*. We noticed that every constraint could be encoded as an "implies relation". For example, if we get a *Equality Constraint:* $1 = 2$, where 1 and 2 are both slot ID. Then this constraint can be encoded into $(1 \implies 5) \wedge (2 \implies 6) \wedge (3 \implies 7) \wedge (4 \implies 8)$.

Based on the above observation, we abstracted all the type constraints into the a bunch of implies logic. Then encode the those implies logic as MAX-SAT problem and LogiQL language, and solve them by sat solver and LogicBlox.

1.1 SAT encoding

1.2 LogiQL encoding

LogiQL is a declarative logic programming language ran by LogicBlox database system. Given a set of constraints from a particular type system, we can encode those constraints into LogiQL form, and solve them by LogicBlox.

1.2.1 Basic Encoding

Basic encoding consists some predicates that won't depend on type systems.

```
variable(v), hasvariableName(v:i)->int(i).
```

This predicate stores all the variables, which will be used for all other tables.

```
isAnnotated[v] = i -> variable(v), boolean(i).
```

This predicate is a functional predicate mapping variable entities to boolean values. Boolean value equals to true means the corresponding slot is annotated by corresponding annotation.

```
hasToBeTrue[v] = h -> variable(v), boolean(h).
```

This predicate is for the situation that a variable must be true, this predicate is only used for the case that there is an equality constraint between a constant slot and a variable slot, a variable slot is supertype of the top type, or a variable slot is subtype of the bottom type.

```
set_1_variable[v] = s -> variable(v), boolean(s).
```

The predicate's name has the pattern "set_x_variable" means a group of variable, x is the number of variables in this set, the whole predicate would be true if one of variables in this set is true.

```
set_1_variable[v] = true <- hasToBeTrue[v] = true.
```

hasToBeTrue[v] = true is equivalent to *set_1_variable[v] = true*, because both of them mean variable v is true. And in other steps, the information in *set_1_variable[v]* will be used, so that we make this rule ensuring textitset_1_variable[v] contains all necessary information.

```
implies1[v1,v2] = e -> variable(v1), variable(v2),
    boolean(e).
```

The predicate that has the pattern "impliesx" represents the "implies" logic in our encoding. "x" indicates how many variables in the right hand side of implies logic. For example, implies1[1,3] = true, means "$1 \implies 3$".

```
rightSide[v] = r -> variable(v), boolean(r).
```

rightSide predicate indicates whether the right side of logic implies operated by logic negation. For example, implies1[1, 2] = true, rightSide[2] = false means $1 \implies \neg 3$.

```
set_1_variable[v2] = true <- implies[v1,v2] = true,
    set_1_variable[v1] = true, rightSide[v2] = true.
set_1_variable[v2] = false <- implies[v1,v2] = true,
    set_1_variable[v1] = true, rightSide[v2] = false.
```

These two rules indicate that the implies relationship between two variables, the value of v2 in set_1_variable depends on the value of v1 , and whether there is a negation in right side of implies. The first one could be read as "If v1 implies v2, v1 is true, and there is no negation for the right side of implies logic, then v2 is true."

These two rules are basic encoding because every type systems has at least one type qualifier.

```
isAnnotated[v] = true <- set_1_variable[v] = true.
isAnnotated[v] = false <- set_1_variable[v] = false.
```

Predicate *isAnnotated[v]* is the place that we store all the results. *isAnnotated[v] = true* means the slot that represented by v is annotated by corresponding qualifier, and *isAnnotated[v] = false* is the opposed case.

```
implies2[v1,v2,v3] = e -> variable(v1), variable(v2),
    variable(v3),boolean(e).
set_2_variable[v1,v2] = e ->variable(v1),variable(v2),
    boolean(e).
```

The idea of predicate *implies2[v1,v2,v3]* is same as the *implies1[v1,v2]*, but it represents the implies relation among three variables.

Similarly the predicate *set_2_variable[v1,v2]* is used for 2 variables.

```
set_2_variable[v2,v3] = true <- implies2[v1,v2,v3] =
    true, set_1_variable[v1] = true;implies2[v1,v2,v3]
    = true,isAnnotated[v1] = true.
```

The above rule can be read as "If $v1 \implies (v2 \vee v3)$, and $v1$ is true, then one of $v2$ and $v3$ has to be true. We put union two conditions here because both *isAnnotated* and *set_1_variable* can indicate the truth value of one variable."

```
isAnnotated[v1] = true <- set_2_variable[v1,_] = true,
     !set_1_variable[v1] = false.
isAnnotated[v2] = true <- set_2_variable[v1,v2] = true
    , isAnnotated[v1] = false ,!set_1_variable[v2] =
    false.
```

If we know the fact that one of v1 and v2 is true, then we will check v1 and v2 one by one, if there is no constraint says v1 cannot be false, we set *isAnnotated[v1] = true*. If v1 mustn't be true, then we will check whether v2 could be true.

All predicates listed above are general encoding because every type system needs them, but if we only have the above predicates, we can only solve the constraint generated for the type systems containing only 2 qualifiers. In order to handle the type system has more qualifiers, some predicate like

```
implies3[v1,v2,v3,v4] = e -> variable(v1), variable(v2
    ), variable(v3), variable(v4),boolean(e).
set_3_variable[v1,v2,v3] = e ->variable(v1),variable(
    v2),variable(v3),boolean(e).
set_3_variable[v2,v3,v4] = true <- implies3[v1,v2,v3,
    v4] = true, set_1_variable[v1] = true;implies3[v1,
    v2,v3,v4] = true,isAnnotated[v1] = true.
```

need to be introduced. And we need to following rules as well:

```
implies3[v1,v2,v3,v4] = e -> variable(v1), variable(v2
    ), variable(v3), variable(v4),boolean(e).
isAnnotated[v1] = true <- set_3_variable[v1,_,_] =
    true, !set_1_variable[v1] = false.
isAnnotated[v2] = true <- set_3_variable[v1,v2,_] =
    true, isAnnotated[v1] = false ,!set_1_variable[v2] =
     false.
isAnnotated[v3] = true <- set_3_variable[v1,v2,v3] =
    true, isAnnotated[v1] = false ,isAnnotated[v2] =
    false ,!set_1_variable[v3] = false.
```