# Design

## Data Structure

Define in state.hh:

value\_v = the value of variable in Boolean programs

sbool = symbolic value of a variable used in Boolean program: F, T, N

state\_v = vector<bool>

size\_pc = the domain of pc

size\_tc = the domain of thread counts;

shared\_state: shared state, a valuation for shared variables

local\_state : local state, a valuation for local variables

thread\_state : a combination of shared and local states

global\_state : a global configuration stored in counter abstraction form

cab\_locals = map<local\_state, size\_tc>

Define in cfg.hh:

type\_stmt: the type of statement involving in our current Boolean programs

expr : define the expression used in Boolean programs

stmt : it wraps the type and condition (if applicable) of a statement

edge : the edge of

## Statement Analysis

1. Statement: parallel assignments with constrain

<id>+ := <expr>+ constrain <expr>

Note: the expression following constrain could involve both v and 'v: The valuation before and after executing that statement.

2. Statement if...else...

Note: the if statements in our benchmarks have the uniformed form:

if <expr> then goto pc; fi;

The expr is very limited into one of the following formats:

(1) 0

(2) \*

(3) !(\*)

(4) v

(5) !v

At this stage, we could assume that all of the if...else... statements follow the above form, and then extend to more general cases in the future.

3. Statement assert(<expr>)

Note: This is a dead end statement;

4. atomic section:

form:

1: atomic\_begin;

2: assume(<expr>);

3: <id>+ := <expr>+ constrain <expr>; || skip;

4: atomic\_end;

Note: except atomic\_begin and atomic\_end, the atomic section only involves three kinds of statements: assume, skip and parallel assignments.

Note that: a goto statement never appears in an atomic section.

5. The only two reasons we could have multiple successors are **goto** and **parallel assignment**.

Goto: However, goto statement is much easier, just update the pc. As we build CFG, we thus already deal with the nondeterministic goto in the CFG via different outgoing edges.

Parallel Assignment: this is the most complicate case because:

1. parallel assignments: multiple variables and expressions;
2. nondeterministic values;
3. constrains following assignments: it probably involves the valuation of variables before and after assignment execution.

Tomorrow, I will compare the concrete version and symbolic version step by step and see which one is better. And this must be done by tomorrow!!! This is pretty important.

For now, I just prefer use symbolic version in the expression evaluations (the **eval** function in expr): I think this is the best thing we can do now, which means:

1. the eval returns sool;
2. if eval returns a “\*”, then split it on demand;

This only needs in parallel assignment …

1. all eval calls in a if statement has to be changed to

if(eval(s,l) != sool::F) {

// …

}