INTRO TO THE FIRST-ORDER QUERYING LANGUAGE IN FLAGCALC

PETER GLENN

1. Queries

Flagcalc answers "queries" as in the follow general patterns that operate on one hundred graphs of eight vertices and an edge incidence likelihood of $14/\binom{8}{2} = 0.5$:

```
-r 8 14 100 -a s=<bool-valued query> all -v crit allcrit
-r 8 14 100 -a z=<discrete-valued query> all -v crit alltall
-r 8 14 100 -a a=<continuous-valued query> all -v crit allmeas
-r 8 14 100 -a e=<set-valued query> all -v crit allsets
-r 8 14 100 -a p=<tuple-valued query> all -v crit allsets
```

Each of these are passed on the command line to `./flagcalc`, e.g.

```
./flagcalc -r 8 14 100 -a s="cr1" all -v crit allcrit
```

computes the property "triangle-free" (aka `cr1` [1]) for each of the graphs (`allcrit`) or just the likelihood of being triangle-free (replace `-v crit allcrit` with `-v crit min` to see just the respective "`true`" and "`false`" totals out of these hundred graphs). Also, always use verbosity level "`rt`" to output runtime in seconds.

2. The idea behind quantifiers with three inputs

Flagcalc has "first order" quantifiers, taking two or three inputs: first input is a list of variables and sets to draw their values from; next input is optional: a boolean-evaluated "criteria"; and third input is a "value" to use when the criteria evaluates as `True`, for example in the case of quantifier `FORALL`:

`FORALL (`v_1 `IN` S_1, v_2 `IN` S_2, ..., `<boolean criteria>`, `<value>)`

where the middle ("criteria") is optional and is evaluated according to the weak-typing of flagcalc into a boolean. Some examples

```
FORALL (v IN V, v > 0, ac(v-1,v))
```

is a basic example that checks each vertex is adjacent to its immediate successor.

Now, matching parentheses is difficult in nested queries; a syntax-highlighting IDE such as JetBrains' can help. For an example of a nested query:

```
FORALL (u IN V, FORALL (v IN V, u == v OR ac(u,v)))
```

or equivalently

Date: December 2, 2025.

[1] Other measures have more meaningful names, this was a first one implemented, before the overall picture emerged

`FORALL (u IN V, FORALL (v IN V, u != v, ac(u,v)))`

These have just tested the property of a graph being "complete" (having all possible edges). Check this with

`cliquem == dimm IFF FORALL (u IN V, FORALL (v IN V, u != v, ac(u,v)))`

on a larger vertex incidence, e.g. `-r 5 8 100`.

3. Flagcalc is weakly-typed

FlagCalc is weakly-typed, meaning it tries to fit a typecast variable into whatever type is expected in the context. Some of the types in flagcalc are integer ("discrete"), real ("continuous"), and boolean ("bool"). Others include set, tuple, string, and graph. Sets and tuples can be nested (and here it is very interesting to study algorithmic complexity classes).

For example, treating the set `{0,2,3}` as a discrete (integer) outputs its cardinality, 3. Treating a continuous value like 3.5 as a discrete value outputs 3 as well. Treating a discrete or continuous value as a bool outputs 1 if non-zero, and 0 otherwise. Treating a tuple as a set simply "forgets" the ordering and removes duplicates. Treating a set as a tuple simply orders it randomly (or to be precise, in whatever order the set was put together).

4. A list of first-order quantifiers

We can now dream big around this list of implemented "quantifiers": each considers the value only when the criteria passes.

- `FORALL` and `EXISTS`: these are familiar as $\forall$ and $\exists$ in standard first-order logic (as in, e.g., H. Enderton's "A Mathematical Introduction to Logic" (2001, 1972) textbook).
- `SUM` and `TALLY`: the continuous (resp. discrete) values are added up.
- `PRODUCT`: the product of all values
- `COUNT`: same as `TALLY` with value constant 1: `COUNT` omits the "value" input
- `AVERAGE`: the mean, that is, the average of all values (shorthand for `SUM...` divided by `COUNT ...`)
- `MIN` and `MAX`: what is the minimum/maximum of the associated value (including infinity or negative infinity)
- `RANGE`: shorthand (and algorithmic improvement) for `MAX...` minus `MIN...`
- `MEDIAN` *Not implemented*
- `MODE` *Not implemented*
- `SET` and `SETD`: form a set consisting of each of the values, and in case of `SET` remove duplicates; do not waste time removing duplicates when `SETD` is used
- `TUPLE`: form a tuple (an ordered set) consisting of each of the values in the order they arise
- `BIGCUP`, `BIGCUPD`, and `BIGCAP`: borrowing from LaTeXnotation, when values are sets, find their union $\bigcup$ (resp. intersection $\bigcap$); and for `BIGCUPD` like in `SETD`, do not waste time removing duplicates

5. Fun with adjacency

Now it connects to the built-in "Ns", which takes a set as an input and returns the vertices that neighbor this set (excluding those within the set):

```
-r 12 33 100000 -a s="Ns({0}) == SET (v IN V, ac(0,v), v)" all -v crit min rt
```

```
/home/peterglenn/CLionProjects/flagcalc/cmake-build-debug/flagcalc -r 12 33 100000 -a "s=Ns({0}) == SET (v IN V, ac(0,v), v)" all -v crit min rt
TIMEDRUN TIMEDRUN100000:
0.496462
APPLYBOOLEANCRITERION APPLYBOOLEANCRITERION100001:
Criterion Sentence Ns({0}) == SET (v IN V, ac(0,v), v) results of graphs:
result == 1: 100000 out of 100000, 1
TIMEDRUN TIMEDRUN100002:
0.592452

Process finished with exit code 0
```

See that all 100000 sample random graphs return "1" aka True.

Additionally a set can be dereferenced: the elements of "E" are sets of size two.

```
-r 12 33 100000 -a s="FORALL (e IN E, ac(e[0],e[1]))" all -v crit min rt
```

```
/home/peterglenn/CLionProjects/flagcalc/cmake-build-debug/flagcalc -r 12 33 100000 -a "s=FORALL (e IN E, ac(e[0],e[1]))" all -v crit min rt
TIMEDRUN TIMEDRUN100000:
0.486843
APPLYBOOLEANCRITERION APPLYBOOLEANCRITERION100001:
Criterion Sentence FORALL (e IN E, ac(e[0],e[1])) results of graphs:
result == 1: 100000 out of 100000, 1
TIMEDRUN TIMEDRUN100002:
0.638336

Process finished with exit code 0
```

And measures like "Nt" ("Neighbor Tally") are occasion to experiment with BIGCUP:

```
-r 5 5 10000 -a s="BIGCUP (e IN E, e) == V IFF FORALL (v IN V, Nt({v}) > 0)"
all -v crit min rt
```

```
/home/peterglenn/CLionProjects/flagcalc/cmake-build-debug/flagcalc -r 5 5 10000 -a "s=BIGCUP (e IN E, e) == V IFF FORALL (v IN V, Nt({v}) > 0)" all -v crit min rt
TIMEDRUN TIMEDRUN10000:
0.031552
APPLYBOOLEANCRITERION APPLYBOOLEANCRITERION10001:
Criterion Sentence BIGCUP (e IN E, e) == V IFF FORALL (v IN V, Nt({v}) > 0) results of graphs:
result == 1: 10000 out of 10000, 1
TIMEDRUN TIMEDRUN10002:
0.067922

Process finished with exit code 0
```

or something unnatural, that reveals the underlying ordered nature of sets:

```
-r 12 33 100000 -a s="FORALL (e1 IN E, e2 IN E, NOT (e1[0]==e2[1] AND e1[1]==e2[0]))"
all -v crit min rt
```

```
/home/peterglenn/CLionProjects/flagcalc/cmake-build-debug/flagcalc -r 12 33 100000 -a "s=FORALL (e1 IN E, e2 IN E, NOT (e1[0]==e2[1] AND e1[1]==e2[0]))" all -v crit min rt
TIMEDRUN TIMEDRUN100000:
0.481745
APPLYBOOLEANCRITERION APPLYBOOLEANCRITERION100001:
Criterion Sentence FORALL (e1 IN E, e2 IN E, NOT (e1[0]==e2[1] AND e1[1]==e2[0])) results of graphs:
result == 1: 100000 out of 100000, 1
TIMEDRUN TIMEDRUN100002:
2.01385

Process finished with exit code 0
```

Indeed, the following example returns all true:

```
-r 12 33 100000 -a s="FORALL (e IN E, e[0]<e[1])" all -v crit min rt
```

All this makes perfect sense to a logician, but is horribly confusing to someone new to first order logic. Therefore, the rest of this paper will focus on concrete examples of getting things done. But first, some extensions for the expert.

6. Extensions

6.1. **Naming.** In C++ one can use "`using`" to give a shorthand for an expression. Here, without the A.I. or automatic code speedups, we manually tell flagcalc that we want to compute "expression" once, then refer to it by a given "name".

The word `NAMING` takes two comma-delimited inputs: a name and an expression.

`NAMING (`s_1 `AS` E_1`,` s_2 `AS` E_2`, , ..., <expression>)`

For example,

```
FORALL (v IN V, NAMING (ns AS SETD (w IN V, ac(v,w), w), st(ns) < dimm AND
st(ns) % 2 == 0)
```

This computes the set "`ns`" of immediate neighbors to a given `v`: is the size of `ns` less than the graph's dimension ("`dimm`") and an even number (zero mod two)?

Immediately this led to a dilemma, the fact that quantifiers have both a criteria and a value section, so `NAMING` cannot bridge that comma delimiter. The solution is to allow the user to omit the word `NAMING` and build in namings into the quantifier itself using just `AS`:

```
FORALL (v IN V, ns AS SETD (w IN V, ac(v,w), w), st(ns) > 0, dimm/st(ns) <
0.5)
```

6.2. **Concurrency.** Very briefly, any flagcalc quantifier can be preceded by `THREADED`, and a few by now are implemented to be preceded by `GPU`:

```
THREADED FORALL (s IN Ps(V), st(s) > 0, EXISTS (p IN Cyclesvs(s[0]), p <=
s)
```

Because the powerset "`Ps(V)`" of the set of vertices grows quickly in size, here the CPU's threads are split amongst the subsets of `V`, and the question is then posed as to whether there is a cycle originating in the first element in the subset `s` that is contained entirely within `s`.

6.3. **Partitions and Sortings.** The "relational" operators `PARTITION` and `SORT` have the same syntax as the quantifiers above, except only take the first two out of three inputs.

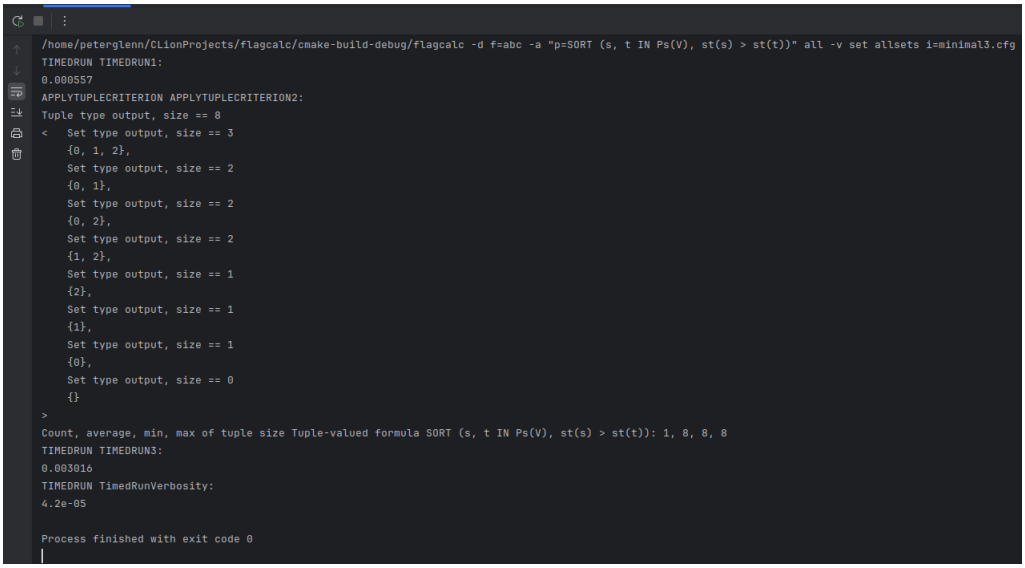
- `PARTITION`: In the variables section (first section) use two instead of one variable. In the middle "criteria" section give some boolean criteria involving those two variables (that is an equivalence relation: symmetric, transitive, and reflexive). The result is the set of sets of values from the set quantified over, i.e. partitioned by "criteria":

  ```
  -r 120 70 1 -a s="st(THREADED PARTITION (u,v IN V, connvc(u,v))) ==
  connm" all -v set allsets i=minimal3.cfg
  ```

  checks that the size of the set partitioning by the notion of "connectedness" is the same as the measure "`connm`".
- `SORT`: In this case the criteria is an ordering. The result is like in `PARTITION`, but instead of a set a tuple.

`-d f="abc" -a p="SORT (s, t IN Ps(V), st(s) > st(t))" all -v set allsets i=minimal3.cfg`

sorts the powerset on three elements by set size.



7. Examples

Please see the examples in the `scripts` *folder of all shell scripts found there...*
This paper is under construction