

# 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  $14/{8 \choose 2} = 0.5$ :

```
-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 alltally
-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<sup>1</sup>) 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 (v1 IN S1, v2 IN S2, ..., <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; allowing the use of measure dimm (the graph’s dimension), this becomes

```
FORALL (v IN V, v+1 < dimm, ac(v,v+1))
```

---

*Date:* December 13, 2025.

<sup>1</sup>Other measures have more meaningful names, this was a first one implemented, before the overall picture emerged

Now, matching parentheses is difficult in nested queries; a syntax-highlighting IDE such as the default Ubuntu/your flavor of Linux text editors or 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

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

or equivalently as well,

```
FORALL (u IN V, 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)))
```

for fun using a larger vertex incidence, e.g. `-r 5 8 100`. Looking ahead to pseudo-second order queries, we have the same test:

```
FORALL (s IN Ps(V), st(s) == 2 IMPLIES s ELT E) IFF cliquem == dimm
```

i.e. “the two statements are equivalent (IFF):

- “for all sets  $s$  in the powerset of the set of vertices, if the size of  $s$  is two then  $s$  is an element of the set of edges, and
- “the graph’s largest clique (complete subgraph) has dimension equal to the graph’s dimension”.

If that’s not rewarding enough, note some logically equivalent queries, and along the way see `Knc` which takes a clique size and a minimal number of embeddings (returning `true` if the graph has at least that minimal number of complete subgraphs on the given vertex size):

```
FORALL (s IN Ps(V), st(s) == 2 IMPLIES s ELT E) IFF Knc(dimm,1)
```

or,

```
FORALL (s IN Ps(V), st(s) == 2, s ELT E) IFF cliquem == dimm
```

or,

```
FORALL (s IN Ps(V), st(s) == 2, s ELT E) IFF st(E) == nchoosek(dimm,2).
```

For a little peek, the two parameters passed to `Knc` are  $n$  and  $mincnt$ :

```
155 }
156 knpcrit( mrecordsx recin ) : crit( recin , shortnamein => "Knc", name => "Parameterized K_n criterion (parameters are complete set size and min count)" ) {
157     populatekns();
158     vals p1 {};
159     p1.t = measuretype::mtdiscrete;
160     nps.push_back( & std::pair< const string, [ & ](p1) {
161         nps.push_back( & std::pair< const string, [ & ](p1) {
162             bindnamedparams();
163         } );
164     } );
165 }
```

and there are analogous “tallies” (integer-valued measures) `Knt`, taking one argument.

### 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).

Please get accustomed to seeing 0 for `false` and 1 for `true`; we are hard-core, after all. Also, please note the association:

“C++”-style type	flagcalc type	flagcalc measure name	flagcalc measure suffix
bool	mtbool	criterion	c
integer	mtdiscrete	tally	t
float	mtcontinuous	measure	m
<various>	mtset	set	s
<various>	mttuple	tuple	p
<unimplemented>	mtgraph	graph	g
std::string	mtstring	string	r

### 4. A LIST OF FIRST-ORDER QUANTIFIERS

We can now create many features 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 LATEX notation, 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
TIME RUN TIME RUN 100000:
0.496462
APPLY BOOLEAN CRITERION APPLY BOOLEAN CRITERION 100001:
Criterion Sentence Ns({0}) == SET (v IN V, ac(0,v), v) results of graphs:
result == 1: 100000 out of 100000, 1
TIME RUN TIME RUN 100002:
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
TIME RUN TIME RUN 100000:
0.480843
APPLY BOOLEAN CRITERION APPLY BOOLEAN CRITERION 100001:
Criterion Sentence FORALL (e IN E, ac(e[0],e[1])) results of graphs:
result == 1: 100000 out of 100000, 1
TIME RUN TIME RUN 100002:
0.638336

Process finished with exit code 0
```

Please note a feature/bug: to dereference several layers deep, one needs parentheses: for example, if dereferencing two layers deep, then instead of **s[1][2]** use **(s[1])[2]**.

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
TIME RUN TIME RUN 10000:
0.031592
APPLY BOOLEAN CRITERION APPLY BOOLEAN CRITERION 100001:
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
TIME RUN TIME RUN 100002:
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 (s1 AS E1, s2 AS E2, , ..., <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.

```
/home/peterglenn/ClionProjects/Flagcalc/cmake-build-debug/flagcalc -d f=abc -a "p=SORT (s, t IN Ps(V), st(s) > st(t))" all -v set allsets i=minimal3.cfg
TIMEORUN TIMEORUN:
0.000557
APPLYTUPLECRITERION APPLYTUPLECRITERION2:
 Tuple type output, size == 8
 < Set type output, size == 3
   {0, 1, 2},
   Set type output, size == 2
   {0, 1},
   Set type output, size == 2
   {0, 2},
   Set type output, size == 2
   {1, 2},
   Set type output, size == 1
   {2},
   Set type output, size == 1
   {1},
   Set type output, size == 1
   {0},
   Set type output, size == 0
   {}
>
Count, average, min, max of tuple size Tuple-valued formula SORT (s, t IN Ps(V), st(s) > st(t)): 1, 8, 8, 8
TIMEORUN TIMEORUNS:
0.003016
TIMEORUN TimedRunVerbosity:
4.2e-05

Process finished with exit code 0
```

**6.4. Stored Procedures.** One of the most powerful features of Flagcalc as far as code maintenance and readability, is the ability to “code” in the query language: we would like to invoke, say,

```
-a s="NOT cr1 IMPLIES NOT Bipartiteviacycles"
```

or more naturally

```
-a s="cr1 IF Bipartiteviacycles"
```

that is, “triangle-free if bipartite”. So how, if `Bipartiteviacycles` is not a built-in measure (aka not found in the output of `./flagcalc -h`)? See if the following excerpt from a file named `storedprocedures.dat` makes sense

```
1 mtbool Bipartiteviacycles
2
3 FORALL (v1 IN V, FORALL (c IN Cyclesys(v1), st(c) % 2 == 0))
4
5 END|
```

It is using modular arithmetic to make sure all cycles have an even length.  
We use it all together as in

```
-d f="abc=defg" -a isp="..../scripts/storedprocedures.dat" s="cr1 IF
    Bipartiteviacycles" all -v crit.
```

Please note that stored procedures can be recursive or call each other even if declared in the wrong order; that a stored procedures .dat file can “include” other .dat files, and that stored procedures (a term borrowed from SQL) can take any number of arguments, as in

```
12      mtbool Cycleoflength ( mtdiscrete n )
13
14      EXISTS (c IN Cycless, st(c) == n)
15
16
17      END
18
```

To add comments, bracket them with /\* and \*/; to “include” another stored procedure file, use #include "<filename.dat>", and please be inspired by glancing at the various examples in flagcalc’s default `scripts` folder (such as, `storedprocedures.dat` and `planarity.dat`).

## 7. EXAMPLES

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