I. Introduction

While developing mathematical theories and proofs it is important to maintain logical soundness. Even if the proof correctness may be obvious to its author, the peer researchers are often unable (because the proof is not detailed enough) or not willing (because the proof is too involved) to verify it rigorously. This way, the errors in formal proofs are sometimes found long after the publication. [?]

This paper uses the **Agda** programming language and proof environment [?] for formalization of earlier developed Parametrised Graphs (PG) theory [?]. The paper additionally implements the algorithm for conversion of PG formulae to canonical form and verifies its correctness.

II. Graph Algebra

A. Definition

We define a graph algebra as an algebraic structure over a set G supporting the following operations:

- An empty graph, denoting no actions.
  $\varepsilon : G$
- Graph overlay, denoting the parallel composition of actions from both graphs. $\_ + \_ : G \to G \to G$
- Graph sequencing, denoting the causal dependency between actions in the first graph and in the second graph. $\_ \Rightarrow \_ : G \to G \to G$

Additionally, the operations must conform to the following rules:

- Overlay is commutative and associative. 1. a + b = b + a 2. (a + b) + c = a + (b + c)
- Empty graph is a no-op in relation to sequencing. 3. 1 * a = a 4. a * 1 = a
- Sequencing is associative. 5. (a * b) * c = a * (b * c)
- Sequencing distributes over addition. 6. a * (b + c) = a * b + a * c 7. (a + b) * c = a * c + b * c
- Decomposition. 8. a * b * c = a * b + a * c + b * c
  This axiom allows one to decompose any causality graph into smaller components.

** Derived theorems

* Parametrised Graphs

The graph algebra introduced in the previous section can only describe static action dependencies. To describe complex systems one has to consider the conditional behaviour as well. To do this, we've extended the graph algebra by annotating the graphs with conditions. Given a set G of the parametrised graphs and a set B of all the possible boolean conditions, together with the following operations:

$\_ \wedge \_ : B \to B \to B$
$\_ \vee \_ : B \to B \to B$
$\top : B$

we introduce a new operation called *condition*:

$[\_]\_ : B \to G \to G$

The condition operation must have the following properties:

$[]x = x$
$[bc]x = [b][c]x$
$[b]x + [c]x = [bc]x$
$[b](x + y) = [b]x + [b]y$
$[b](x * y) = [b]x * [b]y$

We say that there is a *parametrised graph algebra* on a set G with a condition set B if there is a graph algebra on G, a boolean algebra on B and a condition operator satisfying the requirements above.

** Theorems

* Parametrised Graph Formulae

To do automated manipulations with PG algebra formulae, we describe the formulae as a data structure in the following way.

$zogo$
$bogo$
$kojo\ bojo : mojo$
zo
mo
do
$dataPGFormula : Setwhere$
$\_ + \_ : (x\ y : PGFormula) \to PGFormula$
$\_ \Rightarrow \_ : (x\ y : PGFormula) \to PGFormula$
$\varepsilon : PGFormula$
$var : (v : V) \to PGFormula$
$cond : (cond : B) \to PGFormula \to PGFormula$

Here A is a set of graph variables and B is a set of condition variables. Next, we have a constructor for each of the algebra operations with an additional constructor to reference the variables. This way we can construct the formulae in a straightforward way (var "x" + var "y" var "z"). Formula evaluation then is catamorphism of PGFormula, replacing constructor applications with the corresponding algebra operations and 'var' constructors with the actual variable values.

$pg - eval : \{ABG : Set\} \to (\_ + \_\_ \Rightarrow \_ : G \to G \to G) \to (\varepsilon : G) \to (var : A \to G) \to (cond : B \to G \to G) \to PGFormulaAB \to G$

* Formula equivalence

One can notice that it is possible to write the same mathematical function in many different ways, creating a formulae may be equivalent

* Canonical Form

* Conclusion

As a first step, we construct a model of a single system configuration as a partially ordered. We model the graph mo