Machine-assisted Formalisation of Parametrised Graph Algebra

Arseniy Alekseyev, Andrey Mokhov, Alex Yakovlev
School of EECE, Newcastle University

***Abstract*—The paper shows a machine-assisted approach to formal modelling by considering a graph-based model used to describe parametrised systems of causally related events, such as microcontrol circuits. The paper gives a formal description of the model encoded with the type system of Agda programming language, defining it as an arbitrary set with specific operations on its members, forming an algebra. The data structure for formulae of this algebra is then introduced to be used as a representation type for the model. An example of an algorithm manipulating the formulae is shown. The algorithm correctness and termination are proven with each proof being checked by the compiler. The result is both a machine-verifiable formal proof of the theorems and a runnable tool for formula manipulation.**

I. Introduction

Describing complex systems in a natural and compositional way is an important challenge with wide area of application, in particular in hardware design. To address this challenge, the Conditional Partial Order Graph (CPOG) formalism has been previously proposed [2] [4], where the systems with complex behaviour are described as sets of partial orders of events with each partial order corresponding to an individual system operation mode. The partial orders are further annotated with mutually exclusive conditions, serving to select the current mode of operation of the system. CPOGs are then proposed as a condensed description of such systems. Events in the system are associated with the graph vertices and ordering relationships between events are associated with the graph edges. To describe multiple partial orders both vertices and edges can be annotated with conditions determining the set of modes where the graph node should be present. This allows to exploit the similarities between modes of system operation. However, this approach lacks in compositionality and the ability to transform the system specifications while preserving the important properties.

The new Parametrised Graphs formalism [3] builds upon the CPOG formalism by introducing the following features:

- it lifts the assumption of graph acyclicity, allowing general graphs instead of partial orders;
- it adds algebraic operations for combining existing specifications, thus achieving compositionality;
- it axiomatically defines the equivalence relation on specifications, allowing for equivalence preserving transformations.

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. To avoid such problems we have decided to encode the theory in a formal system so that only definitions would require careful inspection, with proofs being checked automatically.

This paper uses Agda [5] – a programming language and proof assistant based on the Martin-Löf type theory – for formalization of Parametrised Graphs theory. The paper additionally describes the algorithm for conversion of PG formulae to normal form and shows that the correctness of the algorithm has been verified.

The paper extensively uses the syntax of Agda and references several definitions from the Agda standard library [1].

II. Graph Algebra

We start with defining an algebra of non-parametrised graphs, to extend them with conditions later.

We define graph algebra as an algebraic structure over a set G with an equivalence relation $\approx$ supporting the following operations:

- An empty graph, denoting no actions.

$$\varepsilon \;:\; \mathsf{G}$$

- Graph overlay, denoting the parallel composition of actions from both graphs.

$$\_+\_ \;:\; \mathsf{G} \to \mathsf{G} \to \mathsf{G}$$

- Graph sequencing, denoting the causal dependency between actions in the first graph and in the second graph.

$$\_\gg\_ \;:\; \mathsf{G} \to \mathsf{G} \to \mathsf{G}$$

Additionally, the operations must satisfy the following properties:

- Overlay is commutative and associative.

$$+\mathsf{assoc} \;:\; \forall\; p\; q\; r \to (p + q) + r \approx p + (q + r)$$
$$+\mathsf{comm} \;:\; \forall\; p\; q \to p + q \approx q + p$$

- Sequencing is associative.

$$\gg \mathsf{assoc} \;:\; \forall\; p\; q\; r \to (p \gg q) \gg r \approx p \gg (q \gg r)$$

- Empty graph is a no-op in relation to sequencing.

$$\gg \mathsf{identity}^{\mathsf{l}} \;:\; \forall\; p \to \varepsilon \gg p \approx p$$
$$\gg \mathsf{identity}^{\mathsf{r}} \;:\; \forall\; p \to p \gg \varepsilon \approx p$$

- Sequencing distributes over overlay.

$$\begin{aligned}&\mathsf{distrib}^{\mathsf{l}} : \forall\ p\ q\ r \rightarrow \\ &\quad p \gg (q + r) \approx p \gg q + p \gg r \\ &\mathsf{distrib}^{\mathsf{r}} : \forall\ p\ q\ r \rightarrow \\ &\quad (p + q) \gg r \approx p \gg r + q \gg r\end{aligned}$$

- Sequence of more than two actions may be decomposed into shorter sequences, forming the original sequence with overlay.

$$\begin{aligned}&\mathsf{decomposition} : \forall\ p\ q\ r \rightarrow \\ &\quad p \gg q \gg r \approx p \gg q + p \gg r + q \gg r\end{aligned}$$

A. Derived theorems

The following theorems has been derived from the axioms:

- Empty graph is a no-op in relation to overlay.

$$+\mathsf{identity} : \forall\ p \rightarrow p + \varepsilon \approx p$$

- Overlay is idempotent.

$$+\mathsf{idempotence} : \forall\ p \rightarrow p + p \approx p$$

- Absorption.

$$\begin{aligned}&\mathsf{absorption}^{\mathsf{l}} : \forall\ p\ q \rightarrow p \gg q + p \approx p \gg q \\ &\mathsf{absorption}^{\mathsf{r}} : \forall\ p\ q \rightarrow p \gg q + q \approx p \gg q\end{aligned}$$

III. Parametrised Graphs

The graph algebra introduced in the previous section can only describe static event dependencies. To describe complex dynamic systems one has to consider the conditional behaviour as well. To do this, we have 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:

$$\begin{aligned}&\_\vee\_ : \mathsf{B} \rightarrow \mathsf{B} \rightarrow \mathsf{B} \\ &\_\wedge\_ : \mathsf{B} \rightarrow \mathsf{B} \rightarrow \mathsf{B} \\ &\neg : \mathsf{B} \rightarrow \mathsf{B} \\ &\top : \mathsf{B} \\ &\bot : \mathsf{B}\end{aligned}$$

we require a new operation called *condition*:

$$[\_]\_ : \mathsf{B} \rightarrow \mathsf{G} \rightarrow \mathsf{G}$$

The condition operation must have the following properties:

$$\begin{aligned}&\mathsf{true\text{-}condition} : \forall\ x \rightarrow [\top]\ x \approx x \\ &\mathsf{false\text{-}condition} : \forall\ x \rightarrow [\bot]\ x \approx \varepsilon \\ &\mathsf{and\text{-}condition} : \forall\ f\ g\ x \rightarrow [f \wedge g]\ x \approx [f]\ [g]\ x \\ &\mathsf{or\text{-}condition} : \forall\ f\ g\ x \rightarrow [f \vee g]\ x \approx [f]\ x + [g]\ x \\ &\mathsf{conditional}+ : \forall\ f\ x\ y \rightarrow [f]\ (x + y) \approx [f]\ x + [f]\ y \\ &\mathsf{conditional}\gg : \forall\ f\ x\ y \rightarrow [f]\ (x \gg y) \approx [f]\ x \gg [f]\ y\end{aligned}$$

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.

A. Derived Theorems

The following theorems has been derived for the Parameterised Graph algebra.

Choice propagation. If we have a choice between similar subgraphs, we can factor out the similarity and propagate choice onto the differing parts.

$$\begin{aligned}&\mathsf{choice\text{-}propagation}_1 : \forall\ b\ p\ q\ r \rightarrow \\ &\quad [b]\ (p \gg q) + [\neg\ b]\ (p \gg r) \approx p \gg ([b]\ q + [\neg\ b]\ r) \\ &\mathsf{choice\text{-}propagation}_2 : \forall\ b\ p\ q\ r \rightarrow \\ &\quad [b]\ (p \gg r) + [\neg\ b]\ (q \gg r) \approx ([b]\ p + [\neg\ b]\ q) \gg r\end{aligned}$$

Condition regularisation. A sequence of conditional events can be rewritten as an overlay of simpler terms.

$$\begin{aligned}&\mathsf{condition\text{-}regularisation} : \forall\ f\ g\ p\ q \rightarrow \\ &\quad [f]\ p \gg [g]\ q \approx [f]\ p + [g]\ q + [f \wedge g]\ (p \gg q)\end{aligned}$$

Strengthened condition regularisation. This generalizes the regularisation theorem by allowing any z containing all the edges between p and q to be used instead of $p \gg q$.

$$\begin{aligned}&\mathsf{condition\text{-}regularisation}_{\mathsf{s}} : \forall\ f\ g\ p\ q\ z \\ &\quad \rightarrow p \gg q \approx p + q + z \\ &\quad \rightarrow [f]\ p \gg [g]\ q \approx [f]\ p + [g]\ q + [f \wedge g]\ z\end{aligned}$$

IV. Parametrised Graph Formulae

To perform automated manipulations of PG algebra formulae, we describe the formulae as an algebraic data type in the following way.

$$\begin{aligned}&\textbf{data}\ \mathsf{PGFormula} : \mathsf{Set}\ \textbf{where} \\ &\quad \_+\_ : (x\ y : \mathsf{PGFormula}) \rightarrow \mathsf{PGFormula} \\ &\quad \_\gg\_ : (x\ y : \mathsf{PGFormula}) \rightarrow \mathsf{PGFormula} \\ &\quad \varepsilon : \mathsf{PGFormula} \\ &\quad \mathsf{var} : (a : \mathsf{A}) \rightarrow \mathsf{PGFormula} \\ &\quad [\_]\_ : (c : \mathsf{B}) \rightarrow \mathsf{PGFormula} \rightarrow \mathsf{PGFormula}\end{aligned}$$

Here A is a set of graph variables and B is a set of condition variables. We also have a constructor of PGFormula corresponding to each of the algebra operations and an additional constructor to reference the free variables. This way we can construct the formulae in a straightforward way:

$$\mathsf{var}\ \texttt{"x"} + \mathsf{var}\ \texttt{"y"} \gg \mathsf{var}\ \texttt{"z"}$$

Formula evaluation then is catamorphism of PGFormula, replacing constructor applications with the corresponding algebra operations and var constructors with the actual variable values.

$$\begin{aligned}&\mathsf{pg\text{-}eval} : \{\mathsf{A}\ \mathsf{B}\ \mathsf{G} : \mathsf{Set}\} \\ &\quad \rightarrow (\_+_{\mathsf{s}}\_\ \_\gg_{\mathsf{s}}\_ : \mathsf{G} \rightarrow \mathsf{G} \rightarrow \mathsf{G}) \\ &\quad \rightarrow (\varepsilon_{\mathsf{s}} : \mathsf{G}) \\ &\quad \rightarrow ([\_]_{\mathsf{s}}\_ : \mathsf{B} \rightarrow \mathsf{G} \rightarrow \mathsf{G}) \\ &\quad \rightarrow (\mathsf{var}_{\mathsf{s}} : \mathsf{A} \rightarrow \mathsf{G}) \\ &\quad \rightarrow \mathsf{PGFormula}\ \mathsf{A}\ \mathsf{B} \\ &\quad \rightarrow \mathsf{G}\end{aligned}$$

We use the same technique to define the BoolFormula data structure, with constructors $\_\wedge\_$, $\_\vee\_$, $\neg\_$, $\top$, $\bot$ and var.

V. Formula equivalence

Naturally, it is possible to write the same mathematical function in many structurally different, but logically equivalent ways. Here we define a notion of PG formula equivalence. We say that two formula are equivalent iff they can be structurally transformed one into the other by the set of rules corresponding to the equality rules of PG algebra. We express this with an indexed inductive data family by explicitly enumerating all the important constructors.

```
data _≈_ : PGFormula (BoolFormula B) V
  → PGFormula (BoolFormula B) V → Set where
  +assoc : ∀ p q r → (p + q) + r ≈ p + (q + r)
  +comm  : ∀ p q → p + q ≈ q + p
  ≫assoc : ∀ p q r → (p ≫ q) ≫ r ≈ p ≫ (q ≫ r)
  ...
```

This definition allows for convenient formula manipulation, without mentioning its semantics. However, the meaning of this definition is dubious because it was constructed manually without any mention of PG Algebra. To connect the formulae equivalence with an algebra object equivalence, we have defined the proper equivalence relation on formulae, in terms of their semantics. We say that equivalent formulae must give equivalent results for any algebra they are evaluated in.

```
f1 ≈ˢ f2 =
  ∀ G → (algebra : PGAlgebra G) → (f : V → G) →
  eval algebra f f1 ≈ eval algebra f f2
```

Here we assume that eval *algebra* applies pg-eval to all of the *algebra* operations. Now we can show that our easier to use equivalence relation is equivalent to the semantics-based definition:

```
≈→≈ˢ : ∀ f g → f ≈ g → f ≈ˢ g
≈ˢ→≈ : ∀ f g → f ≈ˢ g → f ≈ g
```

VI. Normal Form

We say that a normal form (NF) of PG formula (PG) is an overlay of literals (Lit) where each literal is a Node annotated with a condition and each node is either a variable (V) or two variables connected with a sequence operator. We encode these definitions assuming boolean formulae (BF) as conditions.

```
Node = V ⊎ V × V
Lit  = Node × BF
NF   = List Lit
```

So far we have defined the structure of those types without formally saying anything about their semantics. We define the semantics for them by providing a corresponding Parametrised Graph Formulae (PG).

A Node, depending on its constructor, corresponds to either a single variable or two variables connected via the sequence operator.

```
fromNode : Node → PG
fromNode (inj₁ x)   = var x
fromNode (inj₂ (x,y)) = var x ≫ var y
```

A Lit of the form $(node,condition)$ corresponds to the formula $[condition]\ node$.

```
fromLit : Lit → PG
fromLit (node,cond) = [cond] fromNode node
```

NF corresponds to the overlay of all of its literals.

```
fromNF : NF → PG
fromNF = foldr _+_ ε ∘ map fromLit
```

VII. Normalisation algorithm

To automate the translation of formulae to normal form we have developed the algorithm presented in this section.

The top-level normalisation function traverses the PG formula recursively, normalising all of the subformulae and combining them with the appropriate functions ($\_+_{\mathsf{NF}}\_$ for $+$, $\_\gg_{\mathsf{NF}}\_$ for $\gg$, etc.).

```
normalise : PG → NF
normalise = pg-eval
  _+NF_
  _≫NF_
  []
  addCondition
  fromVar
```

The individual functions manipulating normal forms are implemented in the following way.

- The normal form of ε is empty list.
- The normal form of a variable literal x is a singleton list containing $[\top]\ x$.

```
fromVar : V → NF
fromVar x = (inj₁ x,⊤) :: []
```

- Overlay of two normal forms is concatenation of their literals.

```
_+NF_ : NF → NF → NF
a +NF b = a ++ b
```

- Sequence of two normal forms can be defined by applying the distributivity rules as a sum of pairwise sequencing of their literals.

```
_≫r_ : Lit → NF → NF
lit ≫r []       = lit :: []
lit ≫r (x :: xs) = (lit ≫1 x) + (lit ≫r xs)

_≫NF_ : NF → NF → NF
[] ≫NF b       = b
(h :: t) ≫NF b = (h ≫r b) + (t ≫NF b)
```

- Sequence of two literals $[f]\ p \gg [g]\ q$ then can be defined as $[f]\ p + [g]\ q + [f \wedge g]\ r$ where r = newArrows $p\ q$ is the set of new arc nodes formed by sequencing the nodes p and q.

```
vertices : Node → List V
vertices (inj₁ x)   = x :: []
vertices (inj₂ (x,y)) = x :: y :: []

newArrows : Node → Node → List Node
newArrows p q =
  map inj₂ (vertices p ⊗ vertices q)

_≫₁_ : Lit → Lit → List Lit
(p,f) ≫₁ (q,g) = (p,f) :: (q,g)
  :: (map (flip _,_ (f ∧ g)) (newArrows p q))
```

Here vertices n is the list of graph vertices contained in node n – one vertex when n is a vertex node and two vertices when n is an arc node.

newArrows a b then is a set of arc nodes connecting each of the vertices in a to each of the vertices in b.

A. Algorithm Correctness

We define the correctness of normalisation by saying that the semantics of the resulting normal form must be equivalent to the original formula.

```
normalise-correct : ∀ f → f ≈ fromNF (normalise f)
```

To prove this theorem we had to prove several simpler statements.

Normal form overlay is correct. That is, the semantics of concatenated normal forms is the overlay of their individual semantics.

```
+correct : ∀ x y →
  fromNF x + fromNF y ≈ fromNF (x +NF y)
```

This follows from the monoid structure of overlay.

The normal form sequencing functions are correct.

```
≫ correct : ∀ x y →
  fromNF x ≫ fromNF y ≈ fromNF (x ≫NF y)
```

This relies on the right distributivity and the correctness of $\gg_r$.

```
≫r correct : ∀ x y →
  fromLit x ≫ fromNF y ≈ fromNF (x ≫ r y)
```

This relies on the left distributivity and the correctness of $\gg_1$.

```
≫1 correct : ∀ x y →
  fromLit x ≫ fromLit y ≈ fromNF (x ≫ 1 y)
```

The correctness of $\gg_1$ is proven by the following chain of reasoning.

```
fromLit (x,f) ≫ fromLit (y,g)
  ≈ ⟨ condition-regularisation_s; newArrows-correct ⟩
fromLit (x,f) + fromLit (y,g)
  + [f ∧ g] sumNodes (newArrows x y)
  ≈ ⟨ propagating the condition to the literals ⟩
fromLit (x,f) + fromLit (y,g)
  + fromNF (map (flip _,_ (f ∧ g)) (newArrows x y))
  ≈ ⟨ by +assoc and definitions ⟩
fromNF ((x,f) ≫ 1 (y,g))
```

The desired properties of the newArrows function are not as obvious as the properties of the other functions. We have formulated them as follows.

```
newArrows-correct : ∀ x y →
  fromNode x ≫ fromNode y ≈
  fromNode x + fromNode y
    + sumNodes (newArrows x y)
```

where sumNodes = foldr _+_ ε ∘ map fromNode. Our proof of this property is less than elegant. We manually enumerate all four cases (vertex and vertex, vertex and arc, arc and vertex, arc and arc) and prove four theorems individually, using the decomposition, commutativity and associativity axioms. It's likely possible to simplify the proof by treating the nodes as lists of sequenced vertices and prove by induction on those lists, instead of enumerating all the possible cases.

VIII. Conclusion

We have formalised the definitions of algebra of parametrised graph in Agda, and developed the machine-checked proofs of several properties of that algebra.

The formula representation data structure was designed together with the custom structural equivalence relation on formula representations for convenient formula manipulations. The equivalence relation has been proven equivalent to the one defined using formula semantics, thus showing its adequacy.

The normal form representation data structure for PG formulae was designed with its semantics defined as translation to the corresponding general formula representations.

The algorithm of finding the normal form of general formulae was developed and was proven to be correct.

The immediate future work includes formalizing the proof of CPOG being a model of PG Algebra and modification of algorithm to compute the canonical form (where each graph node is mentioned no more than once) instead of just a normal form. Canonical form is much more useful because its size is at most quadratic while the size of a normal form is exponential in general.

The code used in this paper can be obtained by contacting the authors.

References

[1] Nils Anders Danielsson. Agda standard library. http://wiki.portal.chalmers.se/agda/agda.php?n=Libraries.StandardLibrary, 2011.

[2] Andrey Mokhov. *Conditional Partial Order Graphs*. PhD thesis, Newcastle University, 2009.

[3] Andrey Mokhov, Victor Khomenko, Arseniy Alekseyev, and Alex Yakovlev. Algebra of parametrised graphs. Technical report, Newcastle University, 2011 (pending).

[4] Andrey Mokhov and Alex Yakovlev. Conditional Partial Order Graphs: Model, Synthesis and Application. *IEEE Transactions on Computers*, 59(11):1480–1493, 2010.

[5] Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, September 2007.