Machine-assisted Formalisation of Parametrised Graph Algebra

Arseniy Alekseyev, Andrey Mokhov,
Alex Yakovlev, Alex Bystrov

Microelectronic System Design Group
Department of EECE
Newcastle University

January 26, 2012

Describing hardware microcontrollers

Low level options:

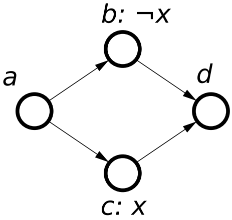
- Logic gate circuits
- State machines
- ...

High level options:

- Petri nets
- Process algebra
- High-level languages
- ...

Conditional Partial Order Graphs

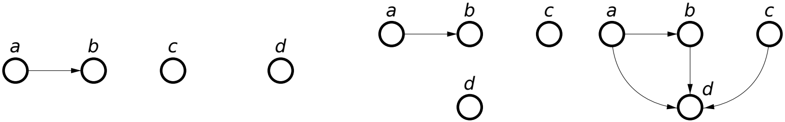
- Vertices represent events
- Edges represent causal dependencies
- Annotated with conditions



Parametrised Graph Algebra

PG Algebra is a generalisation of CPOGs

- Arbitrary set together with algebraic operations on it
- Equivalence relation satisfying certain laws



$G_1 = a \gg b + c$ $\quad$ $G_2 = d$ $\quad$ $G_1 + G_2$ $\quad$ $G_1 \gg G_2$

Desired PG software support

- Formula manipulations
- Conversions to/from different formalisms
- Hardware synthesis

Formal methods

- How do we know the theory is sound?
- How do we know the tools are correct?
- Need a way to statically ensure this.

Agda

Why Agda?

- A total functional programming language
- A proof environment based on Curry-Howard isomorphism
- Easy to learn when you know Haskell
- Newbie-friendly community

Graph Algebra

```
record GraphOps G : Set where
field
  ε : G
  _+_ : G → G → G
  _≫_ : G → G → G

record IsGraphAlgebra : Set where
field
  +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)
  ≫ identityˡ : ∀ {p}     → ε ≫ p ≈ p
  ≫ identityʳ : ∀ {p}     → p ≫ ε ≈ p
  distribˡ    : ∀ {p q r} → p ≫ (q + r) ≈ p ≫ q + p ≫ r
  distribʳ    : ∀ {p q r} → (p + q) ≫ r ≈ p ≫ r + q ≫ r
  decomposition : ∀ {p q r} → p ≫ q ≫ r ≈ p ≫ q + p ≫ r + q
```

Introducing conditions

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

$$
\begin{array}{ll}
\text{boolean-algebra} & : \ \mathsf{BooleanAlgebra\ B} \\
\text{true-condition} & : \ \forall\, x \to [\top]\ x \approx x \\
\text{false-condition} & : \ \forall\, x \to [\bot]\ x \approx \varepsilon \\
\text{and-condition} & : \ \forall\, f\ g\ x \to [f \wedge g]\ x \approx [f]\ [g]\ x \\
\text{or-condition} & : \ \forall\, f\ g\ x \to [f \vee g]\ x \approx [f]\ x + [g]\ x \\
\text{conditional+} & : \ \forall\, f\ x\ y \to [f]\ (x + y) \approx [f]\ x + [f]\ y \\
\text{conditional}\gg & : \ \forall\, f\ x\ y \to [f]\ (x \gg y) \approx [f]\ x \gg [f]\ y
\end{array}
$$

PG Algebra theorems

The following theorems has been derived from the axioms:

$$+\text{identity} : \forall\ p \to p + \varepsilon \approx p$$

$$+\text{idempotence} : \forall\ p \to p + p \approx p$$

$$\text{absorption}^{\text{l}} : \forall\ p\ q \to p \gg q + p \approx p \gg q$$

$$\text{absorption}^{\text{r}} : \forall\ p\ q \to p \gg q + q \approx p \gg q$$

$$\text{choice-propagation}_1 : \forall\ b\ p\ q\ r \to$$
$$[b]\ (p \gg q) + [\neg\ b]\ (p \gg r) \approx p \gg ([b]\ q + [\neg\ b]\ r)$$

$$\text{choice-propagation}_2 : \forall\ b\ p\ q\ r \to$$
$$[b]\ (p \gg r) + [\neg\ b]\ (q \gg r) \approx ([b]\ p + [\neg\ b]\ q) \gg r$$

$$\text{condition-regularisation} : \forall\ f\ g\ p\ q \to$$
$$[f]\ p \gg [g]\ q \approx [f]\ p + [g]\ q + [f \wedge g]\ (p \gg q)$$

Formula data structure

Formula data structure mimics the algebra operations:

$$
\begin{array}{l}
\textbf{data}\ \mathsf{PGFormula}\ \mathsf{B}\ \mathsf{V}\ :\ \mathsf{Set}\ \textbf{where} \\
\quad \_+\_\ :\ (x\ y\ :\ \mathsf{PGFormula}) \to \mathsf{PGFormula} \\
\quad \_\gg\_\ :\ (x\ y\ :\ \mathsf{PGFormula}) \to \mathsf{PGFormula} \\
\quad \varepsilon\ :\ \mathsf{PGFormula} \\
\quad \mathsf{var}\ :\ (a\ :\ \mathsf{V}) \to \mathsf{PGFormula} \\
\quad [\_]\ \_\ :\ (c\ :\ \mathsf{B}) \to \mathsf{PGFormula} \to \mathsf{PGFormula}
\end{array}
$$

Making sense of the formulae

We need to give the formula semantics in terms of algebra.

$$
\begin{aligned}
&\mathsf{pg\text{-}eval} \;:\; \mathsf{PGFormula\ B\ V} \\
&\quad \to (\mathsf{V} \to \mathsf{G}) \\
&\quad \to \mathit{PGAlgebra}\ \mathsf{G\ V} \\
&\quad \to \mathsf{G}
\end{aligned}
$$

$$
\begin{aligned}
&\_{\approx_{\mathsf{f}}}\_ \;:\; \mathsf{PGFormula\ B\ V} \to \mathsf{PGFormula\ B\ V} \to \mathsf{Set} \\
&a \approx_{\mathsf{f}} b \;=\; \forall\ \mathit{assign}\ \mathit{alg} \\
&\quad \to \mathsf{pg\text{-}eval}\ a\ \mathit{assign}\ \mathit{alg} \approx \mathsf{pg\text{-}eval}\ b\ \mathit{assing}\ \mathit{alg}
\end{aligned}
$$

PG Formula normal form

$$
\begin{aligned}
\mathsf{BF} &= \mathsf{BoolFormula}\ \mathsf{B} \\
\mathsf{PG} &= \mathsf{PGFormula}\ \mathsf{BF}\ \mathsf{V} \\
\mathsf{Node} &= \mathsf{V} \uplus (\mathsf{V} \times \mathsf{V}) \\
\mathsf{Lit} &= \mathsf{Node} \times \mathsf{BF} \\
\mathsf{NF} &= \mathsf{List}\ \mathsf{Lit}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{fromNode} : \mathsf{Node} \to \mathsf{PG} \\
&\mathsf{fromNode}\ (\mathsf{inj}_1\ x) = \mathsf{var}\ x \\
&\mathsf{fromNode}\ (\mathsf{inj}_2\ (x, y)) = \mathsf{var}\ x \gg \mathsf{var}\ y \\
&\mathsf{fromLit} : \mathsf{Lit} \to \mathsf{PG} \\
&\mathsf{fromLit}\ (node, cond) = [cond]\ \mathsf{fromNode}\ node \\
&\mathsf{fromNF} : \mathsf{NF} \to \mathsf{PG} \\
&\mathsf{fromNF} = \mathsf{foldr}\ \_{+}\_\ \varepsilon \circ \mathsf{map}\ \mathsf{fromLit}
\end{aligned}
$$

PG Formula normalisation

fromNF : NF → PG

normalise : PG → NF

... (*35 lines of implementation*)

normalise-correct : $\forall\, f \to f \approx$ fromNF (normalise f)

... (*100 lines of proof*)

Conclusions

- We have successfuly formalised the PG Algebra
- We have developed a simple verified program for converting formulae to normal forms
- We thank you for your attention