Progress and sheaves in concurrent refinement algebra

School of Information Technology and Electrical Engineering
Advisors: Prof Ian Hayes, Dr Larissa Meinicke

Nasos Evangelou-Oost

THE UNIVERSITY OF QUEENSLAND
AUSTRALIA

Table of contents

Introduction

Concurrency

- *Concurrency* is the execution of multiple programs in overlapping time periods.
- Individual programs may interact/interfere with each other.
- Not the same as *parallelism*: concurrency is possible on a single processor!
- Very hard to reason about due to state space explosion:

Example

$$a = \langle a_1, a_2, a_3 \rangle, \qquad b = \langle b_1, b_2, b_3, b_4 \rangle$$

$$a \parallel b = \{ \langle a_1, a_2, a_3, b_1, b_2, b_3, b_4 \rangle, \langle a_1, b_1, a_2, a_3, b_2, b_3 \rangle, \langle b_1, a_1, a_2, a_3, b_2, b_3 \rangle, \langle a_1, b_1, a_2, b_2, a_3, b_3 \rangle, \ldots \}$$

(number of shuffles is $\binom{m+n}{m}$: if a and b each have 140 ‘steps’ there are more shuffles in $a \parallel b$ than atoms in the universe!)

Concurrency in practice

- Most programming languages, operating systems, and CPUs support concurrency and have basic constructs to control it.
- Most common primitive is the *lock* that restricts access to a resource. But these
  1. don't solve the state explosion problem
  2. don't help much to reason about concurrency
  3. create new problems: e.g. deadlock

Formal methods

- Formal methods is a discipline that helps to ensure correctness of software by formally refining from mathematical specifications.
- Example: *Hoare logic* for sequential programs is a set of rules for reasoning about the correctness of sequential programs.
- Program $\mathtt{a}$ is augmented to a triple

$$\{\mathtt{pre}\}\mathtt{a}\{\mathtt{post}\}$$

  - $\mathtt{pre}$ a predicate on the initial states of *a*.
  - $\mathtt{post}$ a predicate on the final states of *a*.

Property (Rule of sequential composition)

$$\frac{\{\mathtt{p}\}\mathtt{a}\{\mathtt{q}\} \wedge \{\mathtt{q}\}\mathtt{b}\{\mathtt{r}\}}{\{\mathtt{p}\}\mathtt{a}\,;\mathtt{b}\{\mathtt{r}\}}$$

Example of Hoare logic

Example

Let

- $\{p\}a\{q\} := \{x+1=57\}(y := x+1)\{y=57\}$
- $\{q\}b\{r\} := \{y=57\}(z := y)\{z=57\}$

From the compositional law

$$\frac{\{\mathtt{p}\}\mathtt{a}\{\mathtt{q}\} \wedge \{\mathtt{q}\}\mathtt{b}\{\mathtt{r}\}}{\{\mathtt{p}\}\mathtt{a}\,;\mathtt{b}\{\mathtt{r}\}}$$

we deduce

$$\{x+1=57\}(y := x+1);(z := y)\{z=57\}$$

Previous work

Rely-guarantee

- *Rely-guarantee* (RG) (Jones [1]) generalises Hoare logic to concurrent programs.
- The data of a program includes not only its own behaviour but the possible behaviour of the environment.
- To formulate a compositional rule we must know the interference a program can...
  1. ...tolerate from the environment: rely `r`.
  2. ...inflict on the environment: guarantee `g`.

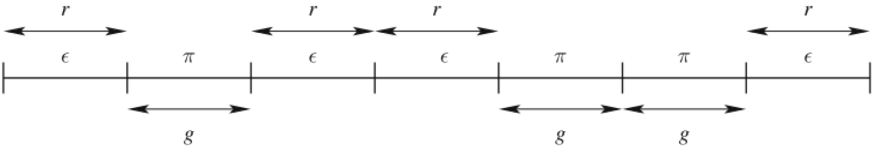

Figure: Execution of a program. Program steps π, environment steps ϵ, rely r, guarantee g.

Compositional rule of RG logic

- Program $\mathtt{a}$ is augmented to a quintuple

$$\{\mathtt{pre}, \mathtt{rely}\}\mathtt{a}\{\mathtt{guar}, \mathtt{post}\}$$

  - $\mathtt{rely}$ is a predicate on the environment steps.
  - $\mathtt{guar}$ is a predicate on the program steps.

Property (Rule of parallel composition)

$$\frac{\{\mathtt{p}_1, \mathtt{r}_1\}\mathtt{a}\{\mathtt{g}_1, \mathtt{q}_1\} \wedge \{\mathtt{p}_2, \mathtt{r}_2\}\mathtt{b}\{\mathtt{g}_2, \mathtt{q}_2\} \wedge \mathtt{g}_2 \implies \mathtt{r}_1 \wedge \mathtt{g}_1 \implies \mathtt{r}_2}{\{\mathtt{p}_1 \wedge \mathtt{p}_2, \mathtt{r}_1 \wedge \mathtt{r}_2\}\mathtt{a} \parallel \mathtt{b}\{\mathtt{g}_1 \vee \mathtt{g}_2, \mathtt{q}_1 \wedge \mathtt{q}_2\}}$$

Concurrent refinement algebra

- *Concurrent refinement algebra* (CRA) (Hayes et al [2]) is a model to facilitate algebraically deriving concurrent programs from formal specifications.
- Programs and specifications are partially ordered:

$$p \leq q$$

  means p implements/refines q.
- RG logic encodes in CRA through algebraic constructs.
- Program executions are classified into four types:
  1. terminating
  2. infinite
  3. aborting
  4. infeasible/incomplete
- A general program may have executions of all types.

Mathematical structure of CRA

- CRA is parameterised by a set of states and specified by axioms.
  - Atomic steps (sequentially indivisible programs) and tests (conditional statements) form boolean algebras.
  - Parallel $\parallel$ and weak conjunction $\Cap$ on atomic steps and tests form commutative quantales.
  - Along with sequential $;$, are monoids on the set of all commands.
  - All operators preserve nonempty suprema.
  - All commands form a complete distributive lattice.
  - Axioms define interactions between these structures, e.g. the interchange laws

$$(a \parallel b) \, ; (a' \parallel b') \leq (a \, ; a') \parallel (b \, ; b')$$
$$(a \Cap b) \, ; (a' \Cap b') \leq (a \, ; a') \Cap (b \, ; b')$$
$$(a \Cap b) \parallel (a' \Cap b') \leq (a \parallel a') \Cap (b \parallel b')$$

  - Iteration is defined by least/greatest fixed points.
- A trace model has been given for CRA, proving its consistency [3].

Main issues

Progress

- Progress of a program is the property that it is not inhibited from advancing.
- Derived properties are:
  - liveness: something good will eventually happen, and dually—
  - safety: something bad will never happen

Examples

- Liveness: the program will eventually release the lock.
- Safety: the program will not launch the missiles.

- Reasoning about liveness implicates infinite executions.

Problem!

The RG rules as encoded within CRA do not apply to infinite executions.

Approach

Category theory

1. A branch of mathematics that studies compositionality and represents structures by directed graphs.
2. Emphasises the relationships between things: a thing is fully determined by its relationships to other things.

Importantly

Category theory is a common language for the variety of algebraic structures that CRA is composed of, and provides a powerful tool set to work with them uniformly.

Example

A set, a lattice, or a monoid are categories. Also the collection of all sets, lattices, or monoids are categories.

Presheaves

- A mapping between categories is a functor.

$$\mathcal{F} : \mathsf{C} \to \mathsf{D}$$

- It gives a picture of C inside D.
- If $\mathsf{D} = \mathsf{Set}$ we call $\mathcal{F}$ a presheaf.
- A presheaf is a parameterised set.
- Presheaves on C form a category $\widehat{\mathsf{C}}$.

Example (Programs are presheaves)

Let $\mathsf{N} = \{0, 1, 2, \ldots\}$ and $F : \widehat{\mathsf{N}}$.
$\mathcal{F}n$ is the set of execution traces of length n.

Sheaves

- A category can be given a topology that gives it a concept of nearness.
- Presheaves that are “continuous" in the topology are sheaves.
- The global data of sheaves are determined locally.
- Presheaves may fail to be sheaves in two ways: global extensions of local data…
  1. …may not exist (*“overdetermined"*);
  2. …may not be unique (*“underdetermined"*).

Our research hypothesis

1. Category theory provides a common language and tool set to handle the variety of algebraic structures of CRA
2. Sheaves provide a precise formalism for describing program behaviours, conjoining them consistently with respect to a specification, or analysing the obstructions to conjunction
3. The logic of sheaves is natively temporal: when the base category is a model of time (or spacetime, i.e. time and memory), a proposition in the sheaf logic is not either true or false, but true *over a certain region of time* (or of time and memory), and the logical operators are temporal ones
4. Sheaves comes equipped with an internal language that acts like a *domain-specific language* for reasoning about and constructing programs and specifications

Timeline

Projected timeline

- (2021 RQ1, Apr 1) Milestone 1: confirmation.
- (2021 RQ1–RQ2) Develop a sheaf semantic model for CRA and prove its soundness.
- (2021 RQ3–RQ4) Formulate a compositional RG type rule capable of reasoning about progress aspects using the internal logic of sheaves.
- (2022 RQ1, Apr 1) Milestone 2: mid-candidature review.
- (2022 RQ1–RQ2) Formalise the sheaf semantic model in *Isabelle/HOL*.
- (2022 RQ2–RQ3) Implement a proof-of-concept domain specific language for the model using the internal language of the sheaf topos in *Isabelle/HOL*.
- (2022 RQ4–2023 RQ1) Thesis writeup.
- (2023 RQ1, Apr 1) Milestone 3: thesis review.

References I

C. B. Jones.
'Specification and design of (parallel) programs'.
Proceedings of the IFIP 9th World Computer Congress, Paris, France, pp. 321–324, 1983.

I. J. Hayes, L. A. Meinicke, K. Winter, and R. J. Colvin.
'A synchronous program algebra: a basis for reasoning about shared-memory and event-based concurrency'.
Formal Aspects Comput., vol. 31, no. 2, pp. 133–163, 2019.

R. J. Colvin, I. J. Hayes, and L. A. Meinicke.
'Designing a semantic model for a wide-spectrum language with concurrency'.
Formal Aspects Comput., vol. 29, no. 5, pp. 853–875, 2017.

References II

J. A. Goguen.
'Sheaf semantics for concurrent interacting objects'.
Math. Struct. Comput. Sci., vol. 2, no. 2, pp. 159–191, 1992.