Exploiting Galois connections for 'Rely/Guarantee Thinking'

DALI/TRUST Kick-off Workshop

University of Minho, September 19-20

J.N. Oliveira

INESC TEC & University of Minho

Ackns

I wish to thank Cliff Jones for inviting me to visit Newcastle last May, where the approach presented in this talk was discussed.

I also thank Ian Hayes, David Naumann, Bernhard Möller and Martin Eric Müller for exchange of ideas about the same topic.

Motivation

Programs don't run in isolation.

Interference, noise...

... but also **cooperation**.

Challenges:

- **Reliability** in presence of interference.
- **Generic** notion of interference? (e.g. CPS)

Context

- For **functional** and **concurrency-free** imperative programs we seem to have stable program development methodologies.
- We haven't got the same level of stability in **concurrent programming**.
- Shared-state concurrency abstractions — **separation** logic, "**rely/guarantee** thinking", ...
- Can we make such abstractions **algebraic** and **calculational**?

Functional programming

A while ago, Shin-Cheng Mu and I tried to show how good **Galois connections** (GCs) are as specification devices, to which greedy / dynamic calculational programming techniques can be applied.

Simple example:

$$x \times y \leqslant z \quad \Leftrightarrow \quad x \leqslant z \div y \tag{1}$$

specifies a "difficult" operator $(\div)$ which can be calculated using the "easy" algebra of $(\times)$

$$(\div y) = R \upharpoonright \geqslant \qquad \textbf{where } x\ R\ z = x \times y \leqslant z$$

where the relational combinator $R \upharpoonright S$ expresses the **optimization** of R by S. Calculations lead to the expected

$$x \div y = \textbf{if } x < y \textbf{ then } 0 \textbf{ else } 1 + (x - y) \div y$$

Imperative programming

Hoare logic — Hoare triple

$$\{p\}P\{q\} \tag{2}$$

means (in relation algebra)

$$[\![P]\!] \cdot p? \;\subseteq\; q? \cdot [\![P]\!] \tag{3}$$

where $[\![P]\!]$ is the meaning of P and $p?$ is the sub-identity (coreflexive) which represents p.

Again this can be expressed by a Galois connection

$$\rho\,(P \cdot p?) \subseteq q? \quad\Leftrightarrow\quad p? \subseteq \;\mathbf{wp}\,(P, q?) \tag{4}$$

expressing weakest preconditions (**wp**) in terms of range ($\rho\,_$) etc.

Shared state concurrent programming

Need for a similar approach to reasoning not about

$$
\begin{array}{l}
r := n; i := 0; \\
\mathit{while}\ (i < r)\ \{ \\
\quad \textbf{if}\ p\ i\ \textbf{then}\ r := i \\
\quad \textbf{else}\ i := 1 + 1\}
\end{array}
$$

but rather about:

$$
\begin{array}{l}
l := n; \\
r := n; \\
\left(
\begin{array}{l}
i := 0; \\
\mathit{while}\ (i < l \sqcap r)\ \{ \\
\quad \textbf{if}\ p\ i\ \textbf{then}\ l := i \\
\quad \textbf{else}\ i := i + 2\}
\end{array}
\ \middle\|\
\begin{array}{l}
j := 1; \\
\mathit{while}\ (j < l \sqcap r)\ \{ \\
\quad \textbf{if}\ p\ j\ \textbf{then}\ r := j \\
\quad \textbf{else}\ j := j + 2\}
\end{array}
\right)
\end{array}
$$

Rely / Guarantee

What is new is the $P \parallel Q$ combinator.

What is the equivalent to $\{p\}\ P\ \{q\}$ for such a combinator?

$\{p\}\ P\ \{q\}$ can be written $p \xrightarrow{P} q$, providing a "type system" for **sequential** composition: $\{p\}\ P; Q\ \{q\}$ provided for some r

$$p \xrightarrow{P} r \xrightarrow{Q} q$$

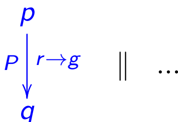
In presence of $\parallel$, triples $\{p\}\ P\ \{q\}$ become **quintuples** $\{p\}\ \{r\}\ P\ \{g\}\ \{q\}$ where

regime	**asset**	**onus**
sequential (;)	*pre* (p)	*post* (q)
concurrent ($\parallel$)	*rely* (r)	*guarantee* (g)

pioneered by Cliff Jones (1981, 1983).

Rely / Guarantee

I will denote **R/G-quintuples** by arrows of the form $p \xrightarrow[r \to g]{P} q$ as linearization of



$\parallel \quad \ldots$

picturing **sequential** composition downwards and **concurrent** composition horizontally.

Informally, $p \xrightarrow[r \to g]{P} q$ means: if

P executes in a p-state;
every ***environment*** *step satisfies r;*
every ***step*** *of P satisfies g;*
P terminates,

then P will end in a q-state.

R/G quintuples

Following Staden (2015),

$$p \xrightarrow[R \rightarrow G]{P} q \quad \Leftrightarrow \quad \{p\}\,(P \parallel \mathit{traces}\ R)\,\{q\} \land \mathit{steps}\ P \ \subseteq\ G \qquad (5)$$

where upper-case R, G stress that they are **binary** relations, and GC

$$\mathit{steps}\ P \ \subseteq\ R \quad \Leftrightarrow \quad P \subseteq \mathit{traces}\ R \qquad (6)$$

is assumed relating program **traces** with program **steps** — a GC which is **perfect** on the steps side:

$$\mathit{steps}\ (\mathit{traces}\ R) = R \qquad (7)$$

Intuition: $\mathit{traces}\ R$ is the largest program whose steps are at most R. (More details later.)

RG quintuple "refactoring"

Next we show how to convert the **logic** statement $p \xrightarrow[R \to G]{P} q$ — that is,

$$steps\ P \subseteq G \ \wedge\ \{p\}\,(P \parallel traces\ R)\,\{q\}$$

— into a more **algebraic** style.

We will make use of the **specification statement** (Morgan, 1988) construct

$$\{p\}\,P\,\{q\} \ \Leftrightarrow\ P \subseteq [p, q] \tag{8}$$

assuming that such a **largest** program $[p, q]$ exists in our setting (discussion later).

RG quintuple "refactoring"

$$\mathit{steps}\ P \subseteq G \ \wedge\ \{p\}\ P \parallel \mathit{traces}\ R\ \{q\}$$

$$\Leftrightarrow \qquad \{\ \text{specification statement (8)}\ \}$$

$$\mathit{steps}\ P \subseteq G \ \wedge\ P \parallel \mathit{traces}\ R \subseteq [p, q]$$

$$\Leftrightarrow \qquad \{\ \text{assume } (\parallel \mathit{traces}\ R) \dashv (\mathbf{rely}\ R)\text{, see below}\ \}$$

$$\mathit{steps}\ P \subseteq G \wedge P \subseteq \mathbf{rely}\ R\ [p, q]$$

$$\Leftrightarrow \qquad \{\ \text{GC (5) ; } \cdot \cap \cdot\text{-universal property}\ \}$$

$$P \subseteq \mathit{traces}\ G \cap \mathbf{rely}\ R\ [p, q]$$

$$\Leftrightarrow \qquad \{\ \text{introduce } \mathbf{guar}\ G\ Q = \mathit{traces}\ G \cap Q\ \}$$

$$P \subseteq \mathbf{guar}\ G\ (\mathbf{rely}\ R\ [p, q])$$

Note the synthesis of two algebraic combinators: **rely** and **guar**.

Definitions of **rely**/**guar**

In summary,

$$p \xrightarrow[R \to G]{P} q \Leftrightarrow P \subseteq \textbf{guar}\ G\ (\textbf{rely}\ R\ [p, q]) \tag{9}$$

where **rely**

$$Q \parallel \mathit{traces}\ R \subseteq P \Leftrightarrow Q \subseteq \textbf{rely}\ R\ P \tag{10}$$

and **guar**

$$\mathit{steps}\ Q \subseteq G \wedge Q \subseteq P \Leftrightarrow Q \subseteq \textbf{guar}\ G\ P \tag{11}$$

are adjoints of suitable GCs.

Informal meaning of **rely**/**guar**

GC

$$Q \parallel traces\ R \subseteq P \Leftrightarrow Q \subseteq \mathbf{rely}\ R\ P$$

tells that **rely** R P is the **largest** program Q which under interference bound by R still approximates P.

GC

$$steps\ Q \subseteq G \wedge Q \subseteq P \Leftrightarrow Q \subseteq \mathbf{guar}\ G\ P$$

tells that **guar** G P is the **largest** approximation Q of P whose steps ensure guarantee condition G.

Algebra of **rely**/**guar**

GCs (6), (10) and (11) — together with indirect equality and basic properties of $\parallel$ etc — yield an **algebra** for the **rely**/**guar** combinators, with properties such as

$$\mathbf{guar}\ \top\ P = P \tag{12}$$

$$\mathbf{rely}\ \bot\ P = P \tag{13}$$

$$\mathbf{guar}\ G\ (\mathbf{guar}\ G'\ P) = \mathbf{guar}\ (G \cap G')\ P \tag{14}$$

$$\mathbf{rely}\ R\ (\mathbf{guar}\ G\ P) = \mathbf{guar}\ G\ (\mathbf{rely}\ R\ P) \ \Leftarrow\ R \subseteq G \tag{15}$$

and such as

$$\mathbf{rely}\ (R_1 \cup R_2)\ P \subseteq \mathbf{rely}\ R_1\ P \cap \mathbf{rely}\ R_2\ P \tag{16}$$

$$(\mathbf{guar}\ G\ P_1 \parallel \mathbf{guar}\ G\ P_2) \subseteq \mathbf{guar}\ G\ (P_1 \parallel P_2) \tag{17}$$

etc

Algebra of **rely**/**guar**

One can also derive standard laws of R/G quintuples in an easy way, e.g. the obvious

$$p \xrightarrow[\bot \to \top]{P} q \;= p \; \{ \, P \, \} \; q \tag{18}$$

and the **parallel-compose** rule:

$$p_1 \wedge p_2 \xrightarrow[R_1 \cap R_2 \to G_1 \cup G_2]{P_1 \| P_2} p_1' \wedge p_2'$$

holds provided

$$\left\{ \begin{array}{l} p_1 \xrightarrow[R_1 \to G_1]{P_1} p_1' \\ p_2 \xrightarrow[R_2 \to G_2]{P_2} p_2' \end{array} \right. \wedge \left\{ \begin{array}{l} G_1 \subseteq R_2 \\ G_2 \subseteq R_1 \end{array} \right.$$

and so on.

Algebra of **rely**/**guar**

All these properties and others are proved by standard GC calculation.

A number of properties of the "easy" operators — notably $\|$ — need to hold, namely

$$(P \parallel \mathit{traces}\ R) = \mathit{traces}\ R \ \Leftarrow\ 1 \ \subseteq\ P \ \subseteq\ \mathit{traces}\ R \tag{19}$$

$$\mathit{traces}\ (R \cup S) = \mathit{traces}\ R \parallel \mathit{traces}\ S \tag{20}$$

$$\mathit{steps}\ (P \parallel Q) = \mathit{steps}\ P \cup \mathit{steps}\ Q \tag{21}$$

$$\mathit{traces}\ \bot = \mathit{skip} \tag{22}$$

and so on.

This leads into CKAs (Hoare et al., 2014).

CKAs of programs

Standard KA (or **quantale**):

Idempotent semiring $(S, +, ;, 0, 1)$ *in which* $a \leqslant b \Leftrightarrow a + b = b$ *is a complete lattice and* $;$ *distributes over arbitrary suprema.*

Concurrent KA (or “double quantale):

$(S, +, ;, \|, 0, 1)$ *where the two multiplications* $(;, \|)$ *are linked by a so-called* **exchange** *axiom:*

$$(a \parallel b) ; (c \parallel d) \leqslant (b ; c) \parallel (a ; d) \tag{23}$$

Example: take S the set of all (non-deterministic) programs under sequential and parallel composition.

CKAs of programs

Quite rich a structure — Galois connections

$$a \,;\, b \leqslant c \Leftrightarrow a \leqslant c \,/\, b$$
$$a \parallel b \leqslant c \Leftrightarrow a \leqslant c \,/\!\!/\, b$$

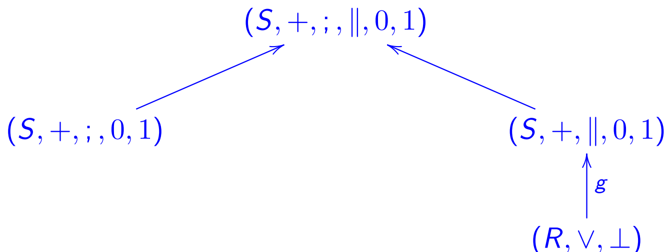
which, together with the **exchange** law (23), yield a number of useful properties, eg.

$$a \parallel b = b \parallel a$$
$$a \,;\, b \leqslant a \parallel b$$
$$(a \parallel b) \,;\, c \leqslant a \parallel (b \,;\, c)$$
$$a \,;\, (b \parallel c) \leqslant (a \,;\, b) \parallel c$$

etc.

CKAs + interference

Add an "interference **monoid**" R to the overall scheme:



$g : R \to S$ is an injective monoid homomorphism telling how interference impacts on programs.

As $\vee$ must be idempotent and commutative, monoid $(R,\vee,\perp)$ becomes a **bounded semilatice**.

CKAs + interference

So, by construction

$$g\ (r \vee s) = g\ r \parallel g\ s$$
$$g\ \bot = 1$$

Simple way of defining g:

Think of it as upper adjoint of a GC

$$f\ p\ \subseteq\ r \Leftrightarrow p \leqslant g\ r \tag{24}$$

perfect on the interference-semilatice side: $f \cdot g = id$.

In our example before, $f = steps$ and $g = traces$.

CKAs + interference

Overall picture, adapting a well-known diagram by Roland Backhouse (2004) — in case *R* is also a lattice:

CKAs + interference

'Unity of opposites' — Theorem 6.42 in (Backhouse, 2004):

$$f\ 1 = \bot$$
$$f\ (p \parallel q) = (f\ p) \vee (f\ q)$$

— *"inverse functions have inverse properties"* **:)** — and so on.

Thus we have all we need to define the two **generic** combinators **rely** and **guar**,

$$g\ r \parallel q \leqslant p \quad \Leftrightarrow \quad q \leqslant \textbf{rely}\ r\ p$$
$$f\ q \subseteq r \wedge q \leqslant p \quad \Leftrightarrow \quad q \leqslant \textbf{guar}\ r\ p$$

granting the R/G algebra briefly presented before.

Comments

Ok, a rich framework, we seem to have a strategy.

However, how do we choose the CKA and the interfering semilatice?

- P in (Staden, 2015) is the powerset of finite **traces** of program steps.
- Hayes et al. (2014) rely on the program refinement ordering and allow for **infinite** traces (buying some complexity).

Question — should we allow for **infinitely long** interference? **interleaved** interference?

By the way — I've briefly experimented with defining the CKA **monadically**, bearing a **probabilistic approach** to interference in mind (McIver et al., 2014).

Epilogue

After an exchange of ideas with David Naumann — cf. his *Calculational Design of Information Flow Monitors* — my current view about R/G is the following:

The key idea of R/G is, after all, to work with an **abstract interpretation** *(AI) of the environment*

In our setting the AI is captured by GC (24).

Think of an environment p. Then $p \leqslant g\ (f\ p)$ by (24). It is **the larger** $g\ (f\ p)$ that we are taking as "allowed" interference.

Is the difference $g\ (f\ p) - p$ an acceptable cost for the abstract interpretation? Or is it **too big a gap**?

Future work

Generic enough for

- probabilistic **memory sharing** (e.g. over faulty memory)
- R/G thinking about **cyber-physical** systems (CPS)

?

Need to check the literature, etc etc.

References

R.C. Backhouse. *Mathematics of Program Construction*. Univ. of Nottingham, 2004. Draft of book in preparation. 608 pages.

I.J. Hayes, C.B. Jones, and R.J. Colvin. Laws and semantics for rely-guarantee refinement. Technical Report CS-TR-1425, Newcastle University, 2014.

C.A. Hoare, S. van Staden, B. Möller, G. Struth, J. Villard, H. Zhu, and P. OHearn. Developments in concurrent Kleene algebra. In *RAMiCS*, volume 8428 of *LNCS*, pages 1–18. 2014.

C.B. Jones. *Development Methods for Computer Programs including a Notion of Interference*. PhD thesis, Oxford University, June 1981. Printed as: Programming Research Group, Technical Monograph 25.

C.B. Jones. Specification and design of (parallel) programs. In *Proceedings of IFIP'83*, pages 321–332. North-Holland, 1983.

A. McIver, T.M. Rabehaja, and G. Struth. Probabilistic rely-guarantee calculus. *CoRR*, abs/1409.0582, 2014. URL `http://arxiv.org/abs/1409.0582`.

C. Morgan. The specification statement. *ACM Trans. Program. Lang. Syst.*, 10 (3):403–419, 1988. doi: 10.1145/44501.44503.

S. Staden. On rely-guarantee reasoning. In *MPC 2015*, pages 30–49, 2015. doi: 10.1007/978-3-319-19797-5_2.