Using Relation Algebraic Methods in the Coq Proof Assistant

Damien Pous, CNRS

RAMiCS, 19.9.2o12

Relation algebra / point-free reasonning

- A wonderful tool to work with binary relations:
  - concise and expressive
  - allows calculational proofs
  - decidable fragments (KA, CKA, KAT, residuated lattices, ...)
- Can be exploited in other models
  - min-max algebras
  - languages, traces
  - matrices
- Very well suited to mechanised reasoning

Objectives

Exploit relation algebra tools and methodology

in the Coq proof assistant

Outline

Basic introduction to Coq

Two case-studies

- Commuting diagrams
- Compiler optimisations

Behind the scene

- How do `ka`/`kat` work?
- Types, untyping
- Algebraic hierarchy

Challenges for future work

What is Coq?

- A purely functional programming language
- A specification language
- An interactive proof assistant

What is it useful for?

- Build certified software

  Compcert (Leroy et al. '05-'10)

- Certify algorithms, prove mathematical theorems

  4 colours theorem (Gonthier '04)

  Feit-Thompson theorem (Gonthier et al. '08-)

What it's not?

- A Turing-complete programming language

Every program terminates

What it's not?

- A Turing-complete programming language

  Every program terminates

- An automatic theorem prover (ATP)

  The user writes the proofs

What it's not?

- A Turing-complete programming language

Every program terminates

- An automatic theorem prover (ATP)

The user writes the proofs

- An oracle

The user writes the programs

What it's not?

- A Turing-complete programming language

  Every program terminates

- An automatic theorem prover (ATP)

  The user writes the proofs

- An oracle

  The user writes the programs

- Something always easy to work with

  The user writes the programs and the proofs

Curry-Howard-de Bruijn correspondence

- A proof of A is a lambda-term of type A
- Checking a proof amounts to type-checking a lambda-term

easy

Curry-Howard-de Bruijn correspondence

- A proof of A is a lambda-term of type A
- Checking a proof amounts to type-checking a lambda-term

  easy
- Writing a proof amounts to writing a lambda-term

  terrible

Curry-Howard-de Bruijn correspondence

- A proof of A is a lambda-term of type A
- Checking a proof amounts to type-checking a lambda-term

easy

- Writing a proof amounts to writing a lambda-term

terrible

- Use tactics to produce lambda-terms (proofs)
- At "Qed", Coq type-checks the lambda-term

Small demo

Outline

Basic introduction to Coq

Two case-studies

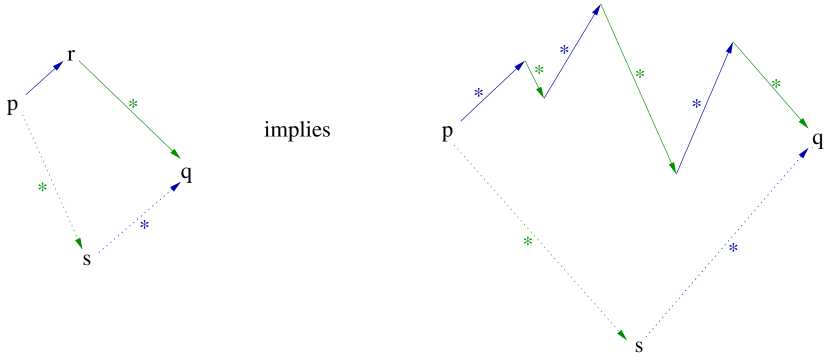
- Commuting diagrams
- Compiler optimisations

Behind the scene

- How do `ka`/`kat` work?
- Types, untyping
- Algebraic hierarchy

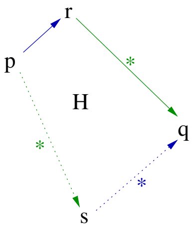
Challenges for future work

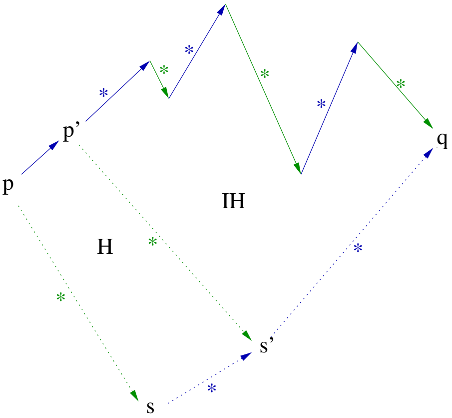
A Church-Rosser property



(Usually presented with $\textcolor{blue}{\rightarrow} \ = \ \textcolor{green}{\leftarrow}$)

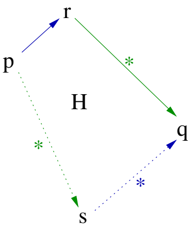
Diagrammatic proof

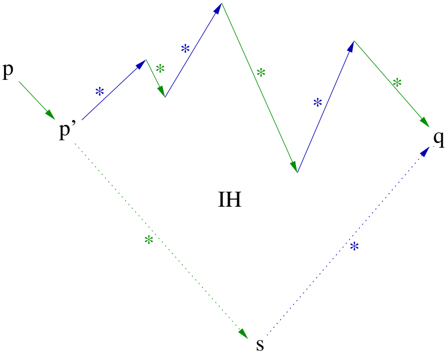




(Usually presented with $\rightarrow\ =\ \leftarrow$)

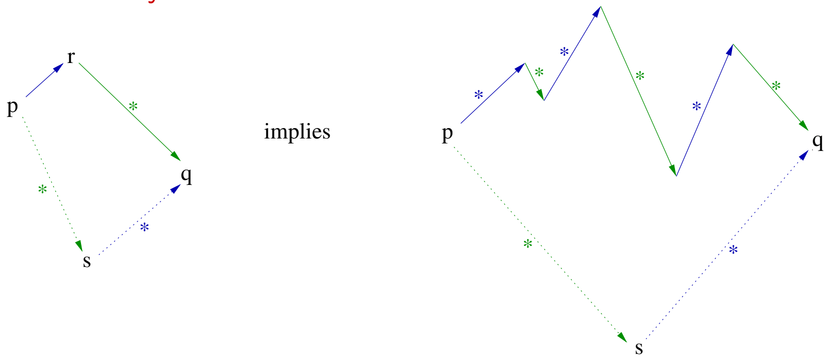
Diagrammatic proof





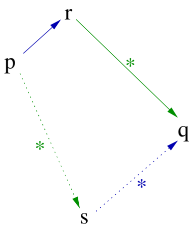
(Usually presented with $\rightarrow$ $=$ $\leftarrow$)

More formally

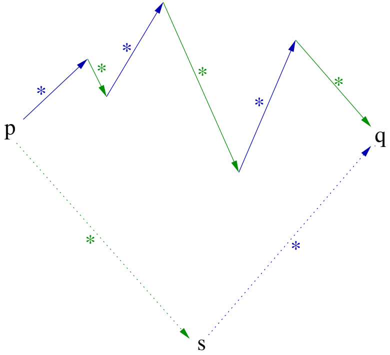


$$(\forall p, r, q, pRr, rS^\star q \Rightarrow \exists s, pS^\star s \wedge sR^\star q)$$
$$\Rightarrow \qquad (\forall p, q, p(R+S)^\star q \Rightarrow \exists s, pS^\star s \wedge sR^\star q)$$

More formally



implies



$$(\forall p, r, q, pRr, rS^\star q \Rightarrow \exists s, pS^\star s \wedge sR^\star q)$$
$$\Rightarrow \qquad (\forall p, q, p(R+S)^\star q \Rightarrow \exists s, pS^\star s \wedge sR^\star q)$$

$$R; S^\star \subseteq S^\star; R^\star \Rightarrow (R+S)^\star \subseteq S^\star; R^\star$$

Point-free reasoning

- The point-free statement is much nicer than the expanded one
- The same holds for the corresponding proofs
- both with pen and pencil ...
  ... and with Coq

demo

Newman's Lemma

If R terminates, then

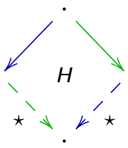
$$R^{\circ}; R \subseteq R^{\star}; R^{\circ\star} \quad \Rightarrow \quad R^{\circ\star}; R^{\star} \subseteq R^{\star}; R^{\circ\star}$$

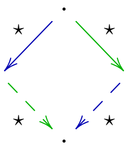
$$\begin{tikzcd}
& \cdot \arrow[dl] \arrow[dr] & \\
{} \arrow[dr, dashed, "\star"'] & H & {} \arrow[dl, dashed, "\star"] \\
& \cdot &
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
& \cdot \arrow[dl, "\star"'] \arrow[dr, "\star"] & \\
{} \arrow[dr, dashed, "\star"'] & & {} \arrow[dl, dashed, "\star"] \\
& \cdot &
\end{tikzcd}$$

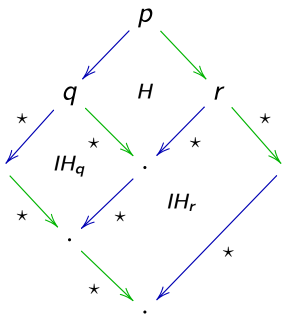
Newman's Lemma

If R terminates, then

$$R^{\circ}; R \subseteq R^{\star}; R^{\circ\star} \quad \Rightarrow \quad R^{\circ\star}; R^{\star} \subseteq R^{\star}; R^{\circ\star}$$







Abstract termination

How to express the termination hypothesis in an abstract setting?

Abstract termination

How to express the termination hypothesis in an abstract setting?

"A Calculational Approach to Mathematical Induction"
H. Doornbos, R. Backhouse, and J. van der Woude '97

(Alternative: ω-algebras)

Factors and monotype factors in division allegories

- Residuated semiring:

$$y \leq x \backslash z \qquad \Leftrightarrow \qquad x ; y \leq z \qquad \Leftrightarrow \qquad x \leq y / z$$

left and right factors

Factors and monotype factors in division allegories

- Residuated semiring:

$$y \le x \backslash z \qquad \Leftrightarrow \qquad x ; y \le z \qquad \Leftrightarrow \qquad x \le y / z$$

left and right factors

- Monotypes: elements below 1

denoted by A, B

Factors and monotype factors in division allegories

- Residuated semiring:

$$y \leq x \backslash z \qquad \Leftrightarrow \qquad x ; y \leq z \qquad \Leftrightarrow \qquad x \leq y / z$$

left and right factors

- Monotypes: elements below 1

denoted by A, B

- Left monotype factor:

$$B \leq x \mathbin{\backslash\!\!\!\cdot} A \qquad \Leftrightarrow \qquad x ; B \leq A ; x$$

With relations: $x \mathbin{\backslash\!\!\!\cdot} A$ is the set of points whose all predecessors by x belong to A

weakest precondition

Properties of monotype factors

- Cancellation: $x; (x \mathbin{\underline{\backslash}} A) \;\leq\; A; x$
- Duplication: $(x \mathbin{\underline{\backslash}} A); (x \mathbin{\underline{\backslash}} A) \;=\; x \mathbin{\underline{\backslash}} A$
- Reversal: $x \mathbin{\underline{\backslash}} A \;=\; A \mathbin{\underline{/}} x^\circ$

$A \mathbin{\underline{/}} x$ being defined symmetrically

Abstract well-foundedness [DBvdW'97]

Definition: Call t well-founded
if for all monotypes $A \leq 1$, $t \mathbin{\setminus\!\!\cdot} A \leq A$ entails $1 \leq A$.

- This pointfree notion coincides with the usual notion of well-foundedness on binary relations.
- In particular, t is well-founded iff t° terminates.

Newman's Lemma using abstract well-founded induction

Setting $y = x\circ$, the proof reduces to showing that forall $A \leq 1$,

$$y^*; A; x^* \leq x^*; y^* \text{ (IH)} \quad \text{entails} \quad y^*; y; (y \backslash A); x; x^* \leq x^*; y^*$$

$$\begin{aligned}
& y^*; y; (y \backslash A); x; x^* & \\
= \; & y^*; y; (y \backslash A); (y \backslash A); x; x^* & \text{(duplication)} \\
= \; & y^*; y; (y \backslash A); (A / x); x; x^* & \text{(converse)} \\
\leq \; & y^*; A; y; (A / x); x; x^* & \text{(left cancellation)} \\
\leq \; & y^*; A; y; x; A; x^* & \text{(right cancellation)} \\
\leq \; & y^*; A; x^*; y^*; A; x^* & \text{(local confluence)} \\
\leq \; & x^*; y^*; y^*; A; x^* & \text{(IH)} \\
= \; & x^*; y^*; A; x^* & \text{(KA)} \\
\leq \; & x^*; x^*; y^* & \text{(IH)} \\
= \; & x^*; y^* & \text{(KA)}
\end{aligned}$$

demo

Compiler optimisations in KAT

"Certification of Compiler Optimizations using Kleene Algebra with Tests"

D. Kozen and M. C. Patron '00

demo

Summary

- fairly short point-free proofs
- thanks to several tactics:
  - `ka` for deciding Kleene algebra equations [Braibant, P. '09]
  - `kat/hkat` for deciding Kleene algebra with tests equations, under Hoare assumptions [P., to be released]
  - `mrewrite` for rewriting modulo associativity [Braibant, P. '11]
- proofs can even be searched this way, by exploiting the provided counter-examples

Outline

Basic introduction to Coq

Two case-studies

- Commuting diagrams
- Compiler optimisations

Behind the scene

- How do `ka`/`kat` work?
- Types, untyping
- Algebraic hierarchy

Challenges for future work

Tactics by reflection

- Take a decidable property

  chose a decision procedure for it

- Coq is a programming language

  program the decision procedure in Coq

- Coq is a proof assistant

  prove the correctness of your decision procedure

- Coq knows how to compute

  let it go

An example

How do `ka/kat` work?

1. Implement an algorithm to check language equivalence of regular expressions with tests
2. Prove it correct

rather easy, using the coalgebraic presentation of KAT

How do `ka`/`kat` work?

1. Implement an algorithm to check language equivalence of regular expressions with tests
2. Prove it correct

   rather easy, using the coalgebraic presentation of KAT

3. Formalise Kozen's completeness theorem for KA

   harder, need matrices

4. Formalise Kozen's reduction of KAT to KA
   ($KAT \vdash \widehat{e} = e$, $G(\widehat{e}) = L(\widehat{e})$)

   non-trivial too

How do `ka`/`kat` work?

1. Implement an algorithm to check language equivalence of regular expressions with tests
2. Prove it correct

   rather easy, using the coalgebraic presentation of KAT

3. Formalise Kozen's completeness theorem for KA

   harder, need matrices

4. Formalise Kozen's reduction of KAT to KA
   $(KAT \vdash \widehat{e} = e,\ G(\widehat{e}) = L(\widehat{e}))$

   non-trivial too

5. Pack everything into a tactic, using reflection

Types

- The completeness proof for KA requires matrices

  Rectangular matrices

- The corresponding algebra is "typed"

  Operations are partial

Types

- The completeness proof for KA requires matrices

Rectangular matrices

- The corresponding algebra is "typed"

Operations are partial

- Natural models are also "typed"

Heterogeneous relations

$$(x;y)^*;x \;=\; x;(y;x)^* \qquad \begin{cases} x : n \to m \\ y : m \to n \end{cases}$$

$\rightarrow$ Work in a categorical setting !

- really easy with Coq dependent types
- except for decision procedures

Untyping theorem

Theorem:

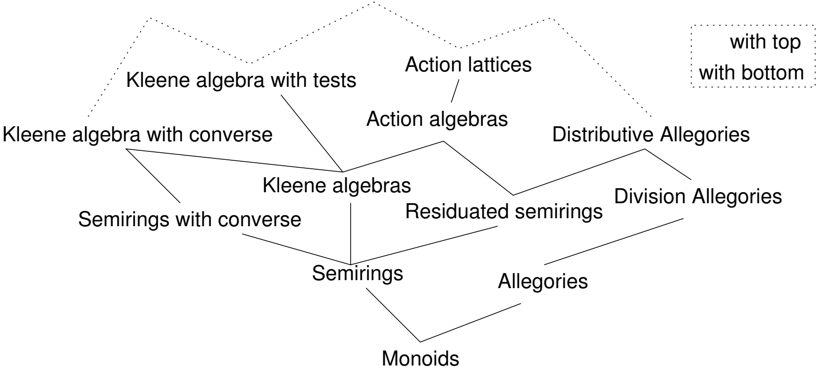
Let $e, f : n \to m$ be typed regular expressions.
Let $u(e), u(f)$ denote their untyped counterparts.
If $KA \vdash u(e) = u(f)$ then $KA \vdash e = f : n \to m$.

Untyping theorem

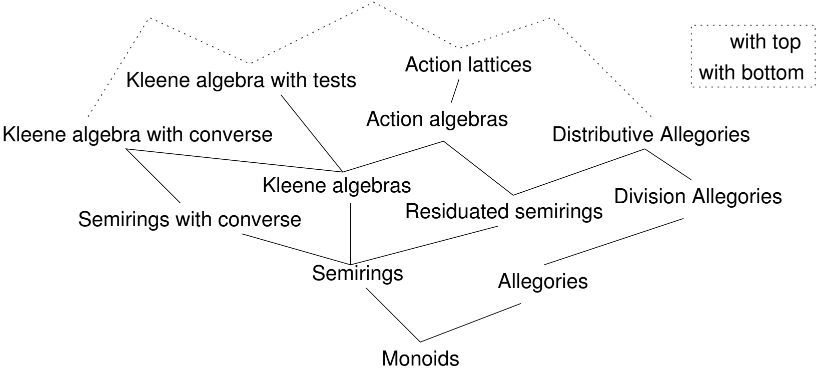
Theorem:
Let $e, f : n \to m$ be typed regular expressions.
Let $u(e), u(f)$ denote their untyped counterparts.
If $KA \vdash u(e) = u(f)$ then $KA \vdash e = f : n \to m$.

- Holds for various other fragments of relation algebra [P. '10]
  residuated semirings, allegories, cyclic linear logic
- Not all of them
  $\top \leq \top;\top$, but $\top_{A,B} \not\leq \top_{A,\emptyset};\top_{\emptyset,B}$

The cloud of relation algebra fragments



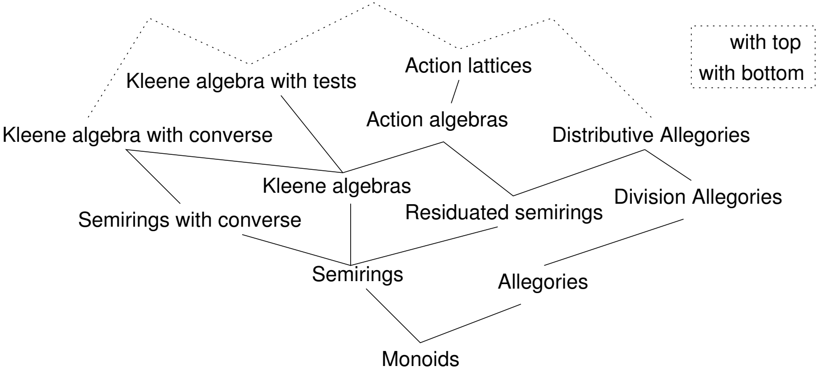
The cloud of relation algebra fragments



We need a modular presentation of the algebraic hierarchy:

- to capture the largest possible range of models
- to benefit from tools from lower structures when working in higher ones

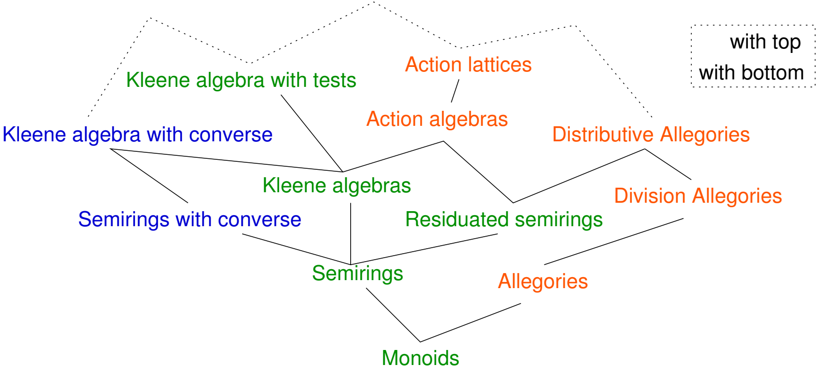
Modular algebraic hierarchy



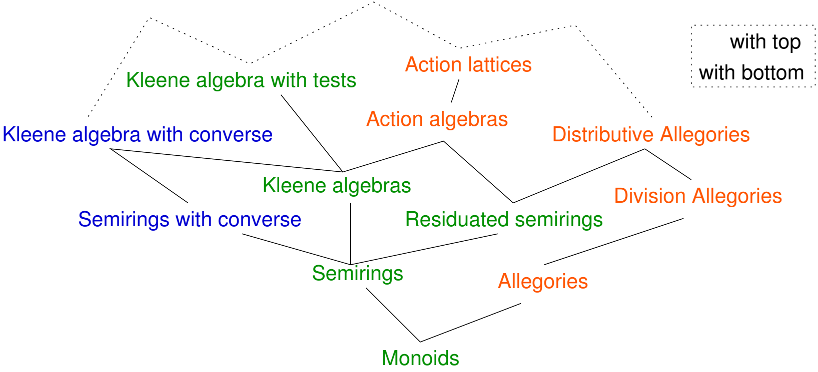
- we failed using modules
- typeclasses do not scale
- current solution: exploit Coq's dependent types to make the relationships first-class

Outline

Algorithmics of relation algebra



Algorithmics of relation algebra



- Decidability:
  - tractable algorithm for Kleene algebra with converse?
  - decidability of action algebra? of allegories? ...
- Other properties:
  - elimination of hypotheses
  - matching / word problem
  - untyping theorem for action algebra?