An efficient Coq Tactic for Deciding Kleene Algebras

Thomas Braibant et Damien Pous (Grenoble)

ITP 2010
Motivations

- Ease the formalisation of proofs dealing with binary relations in Coq (bisimulations ... )
Motivations

- Ease the formalisation of proofs dealing with binary relations in Coq (bisimulations ...)
- [Tarski et al.]: no finite axiomatisation
- A lot of partial axiomatisations
  - non-commutative monoids $(\cdot, 1)$
  - semi-lattices $(+, 0)$
  - non-commutative idempotent semirings $(\cdot, +, 1, 0)$
  - Kleene algebras $(\cdot, +, \ast, 1, 0)$
  - Residuated semi-lattices $(\cdot, +, /, \backslash, 1, 0)$
  - Action algebras (Pratt) $(\cdot, +, /, \backslash, \ast, 1, 0)$
  - Allegories (Freyd & Scedrov) $(\cdot, +, \wedge, /, \backslash, \ast, 1, 0)$
- In each case, different decidability / complexity properties
Motivations

- Ease the formalisation of proofs dealing with **binary relations** in Coq (bisimulations . . .)
- [Tarski et al.]: no finite axiomatisation
- A lot of partial axiomatisations
  - non-commutative monoids \((\cdot, 1)\)
  - semi-lattices \((+, 0)\)
  - non-commutative idempotent semirings \((\cdot, +, 1, 0)\)
  - Kleene algebras \((\cdot, +, *, 1, 0)\)
  - Residuated semi-lattices \((\cdot, +, /, \backslash, 1, 0)\)
  - Action algebras (Pratt) \((\cdot, +, /, \backslash, *, 1, 0)\)
  - Allegories (Freyd & Scedrov) \((\cdot, +, \wedge, /, \backslash, \vee, 1, 0)\)
- In each case, different decidability / complexity properties
- Tools and theorems rather than the algebraic hierarchy itself
Motivations

- Ease the formalisation of proofs dealing with binary relations in Coq (bisimulations . . .)
- [Tarski et al.]: no finite axiomatisation
- A lot of partial axiomatisations
  - non-commutative monoids $(\cdot, 1)$
  - semi-lattices $(+, 0)$
  - non-commutative idempotent semirings $(\cdot, +, 1, 0)$
  - Kleene algebras $(\cdot, +, \ast, 1, 0)$
  - Residuated semi-lattices $(\cdot, +, /, \backslash, 1, 0)$
  - Action algebras (Pratt) $(\cdot, +, /, \backslash, \ast, 1, 0)$
  - Allegories (Freyd & Scedrov) $(\cdot, +, \wedge, /, \backslash, \ast, 1, 0)$
- In each case, different decidability / complexity properties
- Tools and theorems rather than the algebraic hierarchy itself
Kleene algebras

- Models of Kleene algebras: regular languages, binary relations, ...
- Example: “Weak confluence implies the Church-Rosser property”
  - Standard (hand-waving) proof
  - Naive formalisation
  - Algebraic formalisation
  - Algebraic formalisation with tools
Church-Rosser

\[ \text{implies} \]

\[ \begin{align*}
  p & \rightarrow r \\
  r & \rightarrow q \\
  s & \rightarrow s
\end{align*} \]
Church-Rosser (Diagrammatic proof)
Church-Rosser (Diagrammatic proof)
Church-Rosser (more formally)

\((\forall p, r, q, pR r, rS^* q \Rightarrow \exists s, pS^* s \land sR^* q)\)

\(\Rightarrow \quad (\forall p, q, p(R \cup S)^* q \Rightarrow \exists s, pS^* s \land sR^* q)\)
Church-Rosser (more formally)

\[(\forall p, r, q, pRr, rS^*q \Rightarrow \exists s, pS^*s \land sR^*q) \Rightarrow \]
\[ (\forall p, q, p(R \cup S)^*q \Rightarrow \exists s, pS^*s \land sR^*q) \]

\[ R \cdot S^* \subseteq S^* \cdot R^* \Rightarrow (R \cup S)^* \subseteq S^* \cdot R^* \]
Variable $P$: Set.
Variables $R$ $S$: relation $P$.

(** notations for reflexive and transitive closure, and for union of relations **)
Notation "$R^*$" := (clos_refl_trans_1n _ $R$).
Notation "$R + S$" := (union _ $R$ $S$).

Definition WeakConfluence :=
\[ \forall p r q, R p r \rightarrow S^* r q \rightarrow \exists s, S^* p s \land R^* s q. \]

Definition ChurchRosser :=
\[ \forall p q, (R+S)^* p q \rightarrow \exists s, S^* p s \land R^* s q. \]
(** naive proof **)  

**Theorem** WeakConfluence_is_ChurchRosser0:  
WeakConfluence → ChurchRosser.

**Proof.**

```plaintext
intros H p q Hpq.
induction Hpq as [p | p q q' Hpq H qq' IH].

exists p. constructor. constructor.
destruct Hpq as [Hpq Hpq].
destruct IH as [s' Hqs' Hs'q'].
destruct (H p q s' Hpq Hqs') as [s Hps Hss'].
exists s. assumption.
apply trans_rt1n.
apply rt_trans with s';
apply rt1n_trans;
assumption.

destruct IH as [s Hqs Hsq']. ■
exists s.
apply rt1n_trans with q;
assumption.
assumption.
Qed.
```

---

**P:** 
- **Set**
- **R:** relation P
- **S:** relation P
- **H:** WeakConfluence
- **p:** P
- **q:** P
- **q':** P
- **Hpq:** S p q
- **Hqq':** (R + S)* q q'
- **s:** P
- **Hqs:** S* q s
- **Hsq':** R* s q'

---

∃s0 : P, S* p s0 ∧ R* s0 q'
Church-Rosser, no points, no tools
Not yet a short proof, but readable context

Context ‘{KA: KleeneAlgebra}.

Variable A: T.
Variables R S: X A A.

(**
    ⊆ is the inclusion of relations
    ⋆ is the reflexive and transitive closure
    · is the composition
    + is the union
**)
Theorem WeakConfluence_is_ChurchRosser1:
    \( R \cdot S^* \subseteq S^* \cdot R^* \rightarrow (R+S)^* \subseteq S^* \cdot R^* \).
Proof.
    intro H.
    star_left_induction.
    rewrite dot_distr_left.
    repeat apply plus_destruct_leq.
    do 2 rewrite ← one_leq_star_a.
    rewrite dot_neutral_left. reflexivity.
    ■ rewrite dot_assoc. rewrite H.
    rewrite ← dot_assoc.
    rewrite (star_trans R).
    reflexivity.
    rewrite dot_assoc.
    rewrite a_star_a_leq_star_a.
    reflexivity.
Qed.
Church-Rosser, no points, no tools
Not yet a short proof, but readable context

Context '{KA: KleeneAlgebra}.

Variable A: T.
Variables R S: X A A.

(**
  ⊆ is the inclusion of relations
  * is the reflexive and transitive closure
  · is the composition
  + is the union
)**

Theorem WeakConfluence_is_ChurchRosser1:
R · S* ⊆ S* · R* → (R+S)* ⊆ S* · R*.

Proof.
intro H.
star_left_induction.
rewrite dot_distr_left.
repeat apply plus_destruct_leq.
do 2 rewrite ← one_leq_star_a.
rewrite dot_distr_left. reflexivity.
rewrite dot_assoc. rewrite H.
rewrite ← dot_assoc.
 rewrite (star_trans R).
reflexivity.
rewrite dot_assoc.
 rewrite a_star_a_leq_star_a.
reflexivity.
Qed.
Church-Rosser, with tools
With high-level tactics, we can skip the administrative steps

**Theorem** WeakConfluence_is_ChurchRosser2:

\[ R \cdot S^* \subseteq S^* \cdot R^* \rightarrow (R+S)^* \subseteq S^* \cdot R^*. \]

**Proof.**

`intro H.`

`star_left_induction.`

`semiring_normalize.`

`repeat apply plus_destruct_leq.`

`do 2 rewrite ← one_leq_star_a.`

`monoid_reflexivity.`

`rewrite H. monoid_rewrite (star_trans R).`  
`reflexivity.`

`rewrite a_star_a_leq_star_a. reflexivity.`

Qed.

\[ 1 + (R + S) \cdot (S^* \cdot R^*) \subseteq S^* \cdot R^* \]
Church-Rosser, with tools

With high-level tactics, we can skip the administrative steps

**Theorem** WeakConfluence_is_ChurchRosser2:

\[ R \cdot S^* \subseteq S^* \cdot R^* \rightarrow (R+S)^* \subseteq S^* \cdot R^*. \]

**Proof.**

intro H.

star_left_induction.

semiring_normalize. \[\blacksquare\]

repeat apply plus_destruct_leq.

do 2 rewrite ← one_leq_star_a.

monoid_reflexivity.

rewrite H. monoid_rewrite (star_trans R).

reflexivity.

rewrite a_star_a_leq_star_a. reflexivity.

Qed.
Church-Rosser, with tools
With high-level tactics, we can skip the administrative steps

**Theorem** WeakConfluence_is_ChurchRosser2:
\[ R \cdot S^* \subseteq S^* \cdot R^* \rightarrow (R+S)^* \subseteq S^* \cdot R^*. \]

**Proof.**
intro H.
star_left_induction.
semiring_normalize.
repeat apply plus_destruct_leq.
do 2 rewrite \( \leftarrow \) one_leq_star_a.
  \( \square \) monoid_reflexivity.
rewrite H. monoid_rewrite (star_trans R).
  reflexivity.
rewrite a_star_a_leq_star_a. reflexivity.
Qed.
Church-Rosser, with tools

With high-level tactics, we can skip the administrative steps

\textbf{Theorem} WeakConfluence_is_ChurchRosser2:
\[ R \cdot S^* \subseteq S^* \cdot R^* \implies (R+S)^* \subseteq S^* \cdot R^*. \]

\textbf{Proof}.
\begin{align*}
\text{intro } & H. \\
\text{star_left_induction.} \\
\text{semiring_normalize.} \\
\text{repeat apply plus_destruct_leq.} \\
\text{do 2 rewrite } & \leftarrow \text{one_leq_star_a.} \\
\text{monoid_reflexivity.} \\
\text{rewrite } & H. \quad \text{monoid_rewrite (star_trans R).} \\
\text{rewrite } & \text{reflexivity.} \\
\text{rewrite } & a_{\text{star_a_leq_star_a}} \text{. reflexivity.} \\
\text{Qed.}
\end{align*}
Church-Rosser, with better tools

We can do better: equationnal theory of Kleene Algebras is decidable

Theorem WeakConfluence_is_ChurchRosser3:
\[ R \cdot S^* \subseteq S^* \cdot R^* \rightarrow (R+S)^* \subseteq S^* \cdot R^*. \]

Proof.
intro H.
star_left_induction.
semiring_normalize.
rewrite H. ■

G : Graph
Mo : Monoid_Ops
SLo : SemiLattice_Ops
Ko : Star_Op
KA : KleeneAlgebra
A : T
R : X A A
S : X A A
H : R \cdot S^* \subseteq S^* \cdot R^*

1 + S^* \cdot R^* \cdot R^* + S \cdot S^* \cdot R^* \subseteq S^* \cdot R^*
Church-Rosser, with better tools
We can do better: equationnal theory of Kleene Algebras is decidable

Theorem WeakConfluence_is_ChurchRosser3:
\[ R \cdot S^* \subseteq S^* \cdot R^* \rightarrow (R+S)^* \subseteq S^* \cdot R^*. \]

Proof.
intro H.
star_left_induction.
semiring_normalize.
rewrite H.
kleene_reflexivity.
Qed.

\[
1 + S^* \cdot R^* \cdot R^* + S \cdot S^* \cdot R^* \subseteq S^* \cdot R^*
\]
Objectives

- The algebraic view improves:
  - goals readability;
- but we saw the need for:
  - decision tactics (à la ring, omega):
    - kleene_reflexivity, monoid_reflexivity, semiring_reflexivity...
  - simplification tactics (ring_simplify):
    - semiring_normalize, aci_normalize...
  - rewriting tactics (modulo A, modulo AC):
    - monoid_rewrite

btw, we now have a dedicated plugin for rewriting modulo AC
Outline

Motivations

Deciding Kleene Algebras in Coq

Underlying parts of the development

Conclusions and perspectives
Scott vs. Kozen

Let \( \alpha \) and \( \beta \) be two regular expressions \((+,\cdot,0,1,\ast)\).

Scott ’50 \( \alpha \) and \( \beta \) represent the same language iff the corresponding minimal automata are isomorphic.
Let $\alpha$ and $\beta$ be two regular expressions ($+, \cdot, 0, 1, \star$).

**Scott ’50** $\alpha$ and $\beta$ represent the same language iff the corresponding minimal automata are isomorphic.

**Kozen ’94** Initiality of this model for Kleene algebras:
If $\alpha$ and $\beta$ lead to the same automata, then $\mathcal{A} \vdash \alpha = \beta$, for any Kleene algebra $\mathcal{A}$. 
Scott vs. Kozen (again)

Initiality of the model of regular languages

Scott ’50 : We deduce $L((a + b)^*) = L(a^* \cdot (b \cdot a^*)^*)$.

Kozen ’94 : We go further, we deduce $\mathcal{A} \vdash (a + b)^* = a^* \cdot (b \cdot a^*)^*$. 
Making a reflexive tactic

- Theoretical complexity is PSPACE-complete...
  - however, tractable in practice...
  - as long as we take some care in the implementation
Making a reflexive tactic

- Theoretical complexity is PSPACE-complete...
  - however, tractable in practice...
  - as long as we take some care in the implementation

- Coq is a programming language, we code the algorithm:

  ```coq
  Definition decide_Kleene: regexp → regexp → bool := ...
  ```
Making a reflexive tactic

- Theoretical complexity is PSPACE-complete...
  - however, tractable in practice...
  - as long as we take some care in the implementation

- Coq is a programming language, we code the algorithm:
  
  ```coq
  Definition decide_Kleene: regexp → regexp → bool := ...
  
  We formalize Kozen’s proof in Coq:
  
  Theorem Kozen: ∀ a b: regexp, decide_Kleene a b = true ↔ a ≡ b.

  (≡ is the equality generated by the axioms of Kleene Algebras)

  Then we wrap this in a tactic.
The main idea is to represent automata algebraically, with matrices:

\[
(\cdots u \cdots) \cdot \begin{pmatrix} \cdots & M & \cdots \\ \cdots & \cdots & \cdots \end{pmatrix} \cdot \begin{pmatrix} \vdots \\ \vdots \end{pmatrix}
\]
Kozen’s Proof

- The main idea is to represent automata algebraically, with matrices:

\[
\left( \begin{array} {ccc} 
\cdots & u & \cdots \\
\end{array} \right) \cdot \left( \begin{array} {ccc} 
\cdots & M & \cdots \\
\cdots & \cdots & \cdots \\
\end{array} \right)^* \cdot \left( \begin{array} {c} 
\vdots \\
\vdots \\
\end{array} \right)
\]

- Matrices over a Kleene algebra form a Kleene algebra.
The main idea is to represent automata algebraically, with matrices:

\[
\begin{pmatrix}
\cdots & u & \cdots \\
\vdots & M & \vdots \\
\vdots & \vdots & \vdots \\
\end{pmatrix}^* \cdot 
\begin{pmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{pmatrix} = A \cdot \alpha
\]

Matrices over a Kleene algebra form a Kleene algebra.
Kozen’s Proof

- The main idea is to represent automata algebraically, with matrices:

\[
\begin{pmatrix}
\vdots & u & \vdots
\end{pmatrix} \cdot \begin{pmatrix}
\vdots & \vdots & M & \vdots
\end{pmatrix}^* \cdot \begin{pmatrix}
\vdots \\
\vdots
\end{pmatrix} = A \alpha
\]

- Matrices over a Kleene algebra form a Kleene algebra.
- Transcribe and validate the algorithms in this algebraic setting.
Kozen’s Proof

- The main idea is to represent automata algebraically, with matrices:

\[
\left( \cdots \ u \ \cdots \right) \cdot \left( \begin{array}{ccc} \cdots & \cdots & \cdots \\ \cdots & M & \cdots \\ \cdots & \cdots & \cdots \end{array} \right)^* \cdot \left( \begin{array}{c} \vdots \\ \vdots \\ \vdots \end{array} \right) = A \alpha
\]

- Matrices over a Kleene algebra form a Kleene algebra.

- Transcribe and validate the algorithms in this algebraic setting.

In this talk, only a glimpse of these
Construction
A variant of Illie and Yu’s

\[ a + a \cdot (a+b)^* \]

\[ \begin{array}{cc}
\text{1} & \text{2} \\
1 & 2 \\
\end{array} \]
Construction

A variant of Illie and Yu’s

\[ a + a \cdot (a+b)^* \]
Construction
A variant of Illie and Yu’s

\[ a + a \cdot (a+b)^* \]

\[ \begin{array}{c|ccc}
   & 1 & 2 & 3 \\
\hline
1 & 1 & a & a \\
2 & a & a & \\
3 & & & \\
\end{array} \]
Construction
A variant of Illie and Yu’s

\[ a + a \cdot (a+b)^* \]
Construction
A variant of Illie and Yu’s

\[ a + a \cdot (a+b)^* \]
About the construction

- We prove that the construction is **correct algebraically**

\[
(1 \ 0 \ 0 \ 0) \cdot \begin{pmatrix} 0 & a & a & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \epsilon \\ 0 & \epsilon & 0 & a+b \end{pmatrix}^* \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = A 
\]

\[
a + a \cdot (a+b)^* 
\]
About the construction

- We prove that the construction is **correct algebraically**

\[
(1 \ 0 \ 0 \ 0) \cdot \begin{pmatrix} 0 & a & a & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \epsilon \\ 0 & \epsilon & 0 & a+b \end{pmatrix}^* \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \mathcal{A} \cdot a + a \cdot (a+b)^*
\]

- We use **efficient** data-structures to represent the automata (Patricia trees for maps and sets vs matrices)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>\epsilon</td>
<td>\epsilon</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>\epsilon</td>
<td></td>
<td>\epsilon</td>
<td>a,b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>\rightarrow</td>
<td>{2, 3}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>\rightarrow</td>
<td>{4}</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>\rightarrow</td>
<td>{4}</td>
</tr>
<tr>
<td>4</td>
<td>\rightarrow</td>
<td>{4}</td>
<td></td>
<td>\rightarrow</td>
</tr>
</tbody>
</table>
About the construction

- We prove that the construction is correct algebraically

\[
(1 \ 0 \ 0 \ 0) \cdot \begin{pmatrix} 0 & a & a & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \epsilon \\ 0 & \epsilon & 0 & a+b \end{pmatrix}^* \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = A \cdot a + a \cdot (a+b)^*
\]

- We use efficient data-structures to represent the automata (Patricia trees for maps and sets vs matrices)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>a</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td></td>
<td>\epsilon</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>\epsilon</td>
<td></td>
<td>a,b</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>\epsilon</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>a,b</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- We prove that the constructions in the algebraic setting and the efficient setting are equivalent
The big picture
and the datastructures

- **regex**
  - Construction → eNFA.t
  - Epsilon removal → NFA.t
  - Determinisation → DFA.t

- **\(\alpha\)**:
  - \(A\) → \(A_1\) (eval)
  - \(A_2\) (eval) → \(A\) → \(B_1\) (eval)

- **\(\beta\)**:
  - \(A\) → \(A_2\) (eval) → \(A_3\) (eval) → \(A\) → \(B_2\) (eval) → \(B_3\) (eval)

\[equiv\]

**No minimisation (too costly)**
Outline

Motivations

Deciding Kleene Algebras in Coq

Underlying parts of the development

Conclusions and perspectives
We follow the mathematical algebraic hierarchy using Typeclasses:

- **SemiLattice**
  - `<: SemiRing `<: KleeneAlg `<: ...
- **Monoid**

We inherit the tools we developed for monoids, lattices, semi-rings, etc.

(e.g., semiring_reflexivity in the context of a Kleene algebra.)
We follow the mathematical algebraic hierarchy using Typeclasses:

- SemiLattice

  \(<: \text{SemiRing} <: \text{KleeneAlg} <: \ldots\)

- Monoid

We inherit the tools we developed for monoids, lattices, semi-rings, etc.

(e.g., semiring_reflexivity in the context of a Kleene algebra.)

What about matrices?
Matrices

- Infinite functions, with a constrained pointwise equality:
  
  **Definition** \( \text{MX} \ n \ m := \text{nat} \to \text{nat} \to X. \)

  **Definition** \( \text{equal} \ n \ m (M \ N : \text{MX} \ n \ m) := \)
  
  \[ \forall \ i \ j, \ i < n \to j < m \to M_{ij} \equiv N_{ij}. \]

- No bound proofs required for the access
Matrices

- Infinite functions, with a constrained pointwise equality:

\[ \text{Definition } MX \ n \ m := \text{nat} \rightarrow \text{nat} \rightarrow X. \]

\[ \text{Definition equal } n \ m (M \ N : MX \ n \ m) := \]
\[ \forall \ i \ j, i < n \rightarrow j < m \rightarrow M \ i \ j \equiv N \ i \ j. \]

- No bound proofs required for the access

- Easy to manipulate (proof/programs separation)

\[ (M \ast N)_{i,j} = \sum_{k=0}^{m} M_{i,k} \ast N_{k,j} \]

\[ \text{Fixpoint sum} \ k \ (f : \text{nat} \rightarrow X) := \]
\[ \text{match } k \text{ with } 0 \Rightarrow 0 \mid S \ k \Rightarrow f \ k + \text{sum } k \ f \text{ end.} \]

\[ \text{Definition dot } n \ m \ p (M : MX \ n \ m) (N : MX \ m \ p) := \]
\[ \text{fun } i \ j \Rightarrow \text{sum } m \ (\text{fun } k \Rightarrow M \ i \ k \ast N \ k \ j). \]
Matrices

- Infinite functions, with a constrained pointwise equality:

  \[ \text{Definition } MX \; n \; m := \; \text{nat} \rightarrow \text{nat} \rightarrow X. \]

  \[ \text{Definition } \text{equal} \; n \; m \; (M \; N : MX \; n \; m) := \]
  \[ \forall \; i \; j, \; i < n \rightarrow j < m \rightarrow M \; i \; j \equiv N \; i \; j. \]

- No bound proofs required for the access
- Easy to manipulate (proof/programs separation)

\[
(M * N)_{i,j} = \sum_{k=0}^{m} M_{i,k} * N_{k,j}
\]

\[ \text{Fixpoint } \text{sum} \; k \; (f : \text{nat} \rightarrow X) := \]
\[ \text{match } k \text{ with } \begin{array}{ll} O \Rightarrow & 0 \mid S \; k \Rightarrow f \; k + \text{sum} \; k \; f \end{array} \text{ end.} \]

\[ \text{Definition } \text{dot} \; n \; m \; p \; (M : MX \; n \; m) \; (N : MX \; m \; p) := \]
\[ \text{fun } i \; j \Rightarrow \text{sum} \; m \; (\text{fun } k \Rightarrow M \; i \; k * N \; k \; j). \]

bounds proofs are easy to cope with, in proof mode

no hidden boilerplate
Thanks to typeclasses, we inherit tools and theorems for matrices:

- square matrices built over a semi-ring form a semi-ring;
- square matrices built over a Kleene algebra form a Kleene algebra.
Matrices cont.

Thanks to typeclasses, we inherit tools and theorems for matrices:

- square matrices built over a semi-ring form a semi-ring;
- square matrices built over a Kleene algebra form a Kleene algebra.

At several places, we need rectangular matrices!
How to deal with rectangular matrices?

- Without extra stuff, we cannot re-use tools for them: rectangular matrices do not form a semiring
  - operations (\(\cdot, +, \ldots\)) are partial (dimensions have to agree)

\[
X : \text{Type}.
\]

\[
dot : X \to X \to X.
\]

\[
one : X.
\]

\[
\text{plus} : X \to X \to X.
\]

\[
\text{zero} : X.
\]

\[
\text{star} : X \to X.
\]

\[
dot_{\text{neutral left}}:
\forall x, \text{dot one } x = x.
\]

...
How to deal with rectangular matrices?

- Without extra stuff, we cannot re-use tools for them: rectangular matrices do not form a semiring
  - operations (⋅, +, ...) are partial (dimensions have to agree)

\[
\begin{align*}
X &: \text{Type.} \\
T &: \text{Type.} \\
\text{dot}: X &\to X \to X. \\
\text{dot}: \forall n \text{ m p}, X \text n m &\to X \text m p \to X \text n p. \\
\text{one}: X. \\
\text{one}: \forall n, X \text n n. \\
\text{plus}: X &\to X \to X. \\
\text{plus}: \forall \text n \text m, X \text n m &\to X \text n m \to X \text n m. \\
\text{zero}: X. \\
\text{zero}: \forall n \text m, X \text n m. \\
\text{star}: X &\to X. \\
\text{star}: \forall n, X \text n n \to X \text n n. \\
\text{dot_neutral_left}: \\
\forall x, \text{dot one x }= x. \\
\text{dot_neutral_left}: \\
\forall n \text m (x: X \text n m), \text{dot one x }= x.
\end{align*}
\]
How to deal with rectangular matrices?

- Without extra stuff, we cannot re-use tools for them: rectangular matrices do not form a semiring
  - operations (⋅, +, ...) are partial (dimensions have to agree)

\[ \forall \ x, \text{dot one } x = x. \]

\[ \forall \ n \ m \ (x: X n m), \text{dot one } x = x. \]

- Introduce typed structures from the beginning
Typed structures

We handle heterogeneous relations ($X \ A \ B := A \rightarrow B \rightarrow \text{Prop}$), as well as matrices:

- $M \times \text{SemiLattice} : \text{SemiLattice} \rightarrow \text{SemiLattice}$.
- $M \times \text{SemiRing} : \text{SemiRing} \rightarrow \text{SemiRing}$.
- $M \times \text{KleeneAlgebra} : \text{KleeneAlgebra} \rightarrow \text{KleeneAlgebra}$.

Here, we deal with typed structures

- All theorems are inherited at the matricial level.
Typed structures

We handle heterogeneous relations \((X \ A \ B := A \rightarrow B \rightarrow \text{Prop})\), as well as matrices:

\[
\begin{align*}
\text{MxSemiLattice} & : \text{SemiLattice} \rightarrow \text{SemiLattice}. \\
\text{MxSemiRing} & : \text{SemiRing} \rightarrow \text{SemiRing}. \\
\text{MxKleeneAlgebra} & : \text{KleeneAlgebra} \rightarrow \text{KleeneAlgebra}.
\end{align*}
\]

Here, we deal with typed structures

- All theorems are inherited at the matricial level.
- What about extending decision procedures to deal with typed structures?

\[
\begin{array}{c}
a \cdot (b \cdot a)^* \\
\Downarrow \\
\bullet
\end{array} 
\begin{array}{c}
\Downarrow \\
\bullet = (a \cdot b)^* \cdot a \\
\Downarrow \\
\bullet
\end{array}
\]
Typed structures

We handle heterogeneous relations \((X \ A \ B := A \to B \to \text{Prop})\), as well as matrices:

\[
\begin{align*}
\text{MxSemiLattice} &: \text{SemiLattice} \to \text{SemiLattice}. \\
\text{MxSemiRing} &: \text{SemiRing} \to \text{SemiRing}. \\
\text{MxKleeneAlgebra} &: \text{KleeneAlgebra} \to \text{KleeneAlgebra}.
\end{align*}
\]

Here, we deal with typed structures

- All theorems are inherited at the matricial level.
- What about extending decision procedures to deal with typed structures?

\[
\begin{align*}
\bullet \cdot (b \cdot a)^* & \equiv (a \cdot b)^* \cdot \bullet : A \to B \\
\bullet & \equiv \bullet
\end{align*}
\]

\[
a : A \to B, b : B \to A
\]
Typed structures

We handle heterogeneous relations \((X A B := A \rightarrow B \rightarrow \text{Prop})\), as well as matrices:

\[
\begin{align*}
\text{MxSemiLattice} & : \text{SemiLattice} \rightarrow \text{SemiLattice.} \\
\text{MxSemiRing} & : \text{SemiRing} \rightarrow \text{SemiRing.} \\
\text{MxKleeneAlgebra} & : \text{KleeneAlgebra} \rightarrow \text{KleeneAlgebra.}
\end{align*}
\]

Here, we deal with typed structures

- All theorems are inherited at the matricial level.
- What about extending decision procedures to deal with typed structures?

\[
\begin{align*}
a \cdot (b \cdot a)^* & \overset{?}{=} (a \cdot b)^* \cdot a : A \rightarrow B \\
\bullet & = \bullet
\end{align*}
\]

\[
a : A \rightarrow B, b : B \rightarrow A
\]

tackle the problem differently... let’s erase types!
Untyping

The general scheme

untyped setting:

typed setting:

\[ \bullet \xrightarrow{\text{decide}} \bullet \]

\[ \bigtriangleup \]

\[ \text{erase types} \]

\[ \bigtriangleup \]

\[ \text{rebuild types} \]

\[ \bigtriangleup \]

Depending on the algebraic structure:

- Semi-lattices: trivial
- Monoids: rather easy
- Semirings: tricky
- Kleene algebras: same as for semirings
- Residuated lattices: with constraints
- Action algebras/lattices: ?

So, everything is...
Untyping
The general scheme

untyped setting:

• decide

↓

• rebuild types

erase types

yped setting:

•

? =

Depending on the algebraic structure:

\( \mathcal{A} \)

<table>
<thead>
<tr>
<th>semi-lattices</th>
<th>trivial</th>
</tr>
</thead>
<tbody>
<tr>
<td>monoids</td>
<td>rather easy</td>
</tr>
<tr>
<td>semirings</td>
<td>tricky</td>
</tr>
<tr>
<td>Kleene algebras</td>
<td>same as for semirings</td>
</tr>
<tr>
<td>residuated lattices</td>
<td>with constraints</td>
</tr>
<tr>
<td>action algebras/lattices</td>
<td>?</td>
</tr>
</tbody>
</table>

So, everything is fine...
Outline

Motivations

Deciding Kleene Algebras in Coq

Underlying parts of the development

Conclusions and perspectives
Conclusions

▶ A decision tactic for Kleene algebras (available on the web):
  ▶ reflexive
  ▶ efficient (first version: 40 symbols, now: 1000)
  ▶ correct and complete
  ▶ \( \sim \) 7000 lines of spec (definitions, functions)
  ▶ \( \sim \) 7000 lines of proofs
  ▶ 182 Kb of compressed .v files using gzip (current trunk)
Conclusions

▶ A decision tactic for Kleene algebras (available on the web):
  ▶ reflexive
  ▶ efficient (first version: 40 symbols, now: 1000)
  ▶ correct and complete
  ▶ \(\sim\) 7000 lines of spec (definitions, functions)
  ▶ \(\sim\) 7000 lines of proofs
  ▶ 182 Kb of compressed .v files using gzip (current trunk)

▶ Other tools for the underlying structures:
  ▶ various algebraic structures,
  ▶ matrices
Learnings

- **Finite sets/Finite maps:**
  - used a lot in our development
  - Patricia trees rule for binary positive numbers
  - mixing proofs and programs hinders performances (slightly)

- **Typeclasses:**
  - much more supple to use than modules
  - overhead due to the inference of implicit arguments

- **Interfaces:**
  - in order to compute, cannot hide a module behind a signature (coercions)
  - break proofs when changing the implementation
  - example: going from AVL based FSets to Patricia trees
What’s coming next?

- Kleene algebras with tests (automation for Hoare logic)
- Merging the equivalence check and the determinisation
- Back-end for simulation proof obligations
Thanks you for your attention

Any Questions?
http://sardes.inrialpes.fr/~braibant/atbr/
Determinisation

- Construct the powerset automata
- Let $X$ be the decoding matrix of the accessible subsets of the automata $(u, M, v)$:

$$X_{sj} \triangleq j \in s$$

- We can define $\overline{M}$ and $\overline{u}$ such that:

$$\overline{M}^* \cdot X = X \cdot M^* \quad \overline{u} \cdot X = u$$

- We deduce

$$\overline{u} \cdot \overline{M}^* \cdot X \cdot v = \overline{u} \cdot X \cdot M^* \cdot v$$

$$= u \cdot M^* \cdot v$$