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 $(\cdot, +, \star, 1, 0)$
  - Residuated semi-lattices $(\cdot, +, /, \backslash, 1, 0)$
  - Action algebras (Pratt) $(\cdot, +, /, \backslash, \star, 1, 0)$
  - Allegories (Freyd & Scedrov) $(\cdot, +, \wedge, /, \backslash, \bar{\cdot}, 1, 0)$
- 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 $(\cdot, +, \star, 1, 0)$
  - Residuated semi-lattices $(\cdot, +, /, \backslash, 1, 0)$
  - Action algebras (Pratt) $(\cdot, +, /, \backslash, \star, 1, 0)$
  - Allegories (Freyd & Scedrov) $(\cdot, +, \wedge, /, \backslash, \bar{\cdot}, 1, 0)$
- In each case, different decidability / complexity properties
- Tools and theorems rather than the algebraic hierarchy itself

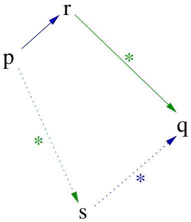
Motivations

- 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 $(\cdot, +, \star, 1, 0)$
  - Residuated semi-lattices $(\cdot, +, /, \backslash, 1, 0)$
  - Action algebras (Pratt) $(\cdot, +, /, \backslash, \star, 1, 0)$
  - Allegories (Freyd & Scedrov) $(\cdot, +, \wedge, /, \backslash, \bar{\cdot}, 1, 0)$
- In each case, different decidability / complexity properties
- Tools and theorems rather than the algebraic hierarchy itself

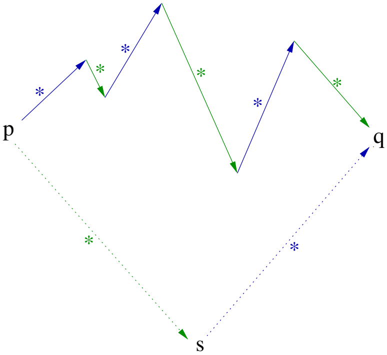
Kleene algebras

- 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

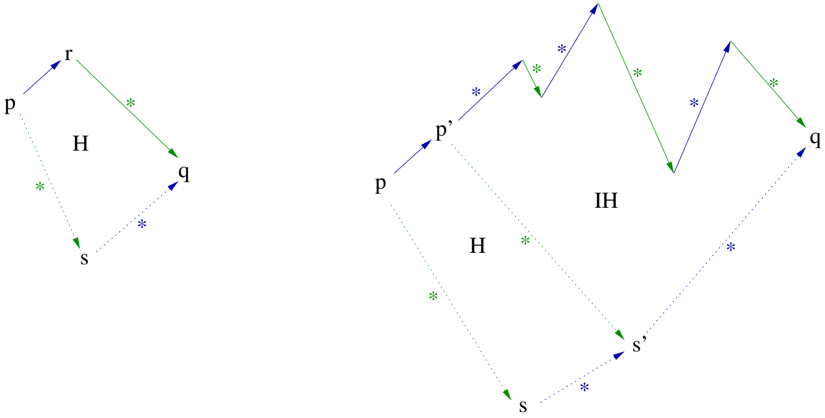
Church-Rosser



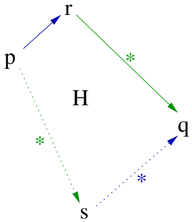
implies

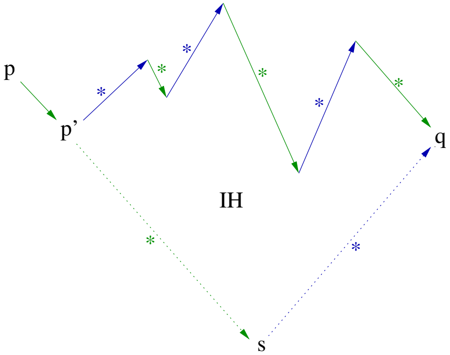


Church-Rosser (Diagrammatic proof)

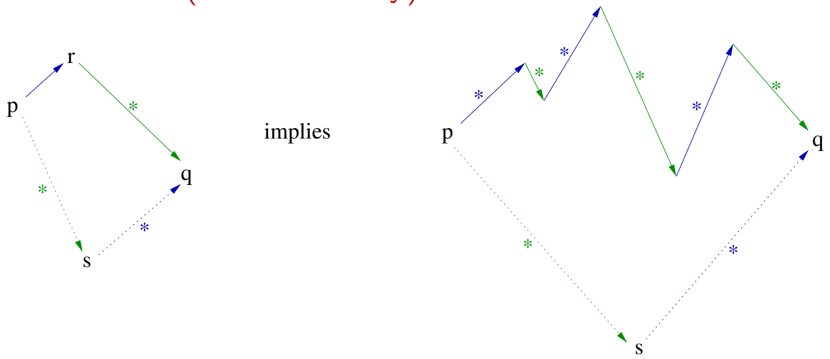


Church-Rosser (Diagrammatic proof)



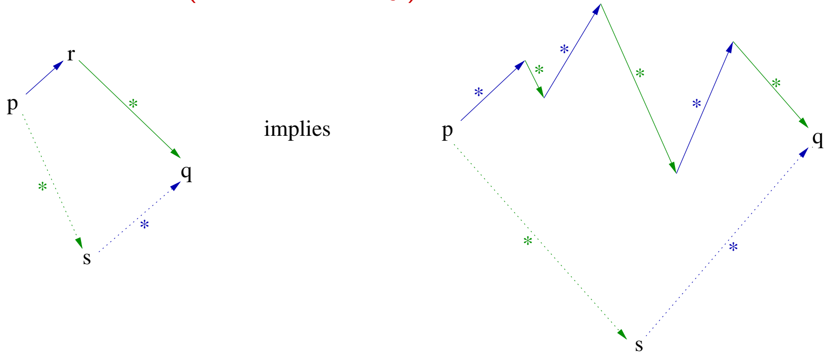


Church-Rosser (more formally)



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

Church-Rosser (more formally)



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

$$R \cdot S^\star \subseteq S^\star \cdot R^\star \Rightarrow (R \cup S)^\star \subseteq S^\star \cdot R^\star$$

Church-Rosser, with points

```
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 :=
  ∀ p r q, R p r → S⋆ r q → ∃s, S⋆ p s ∧ R⋆ s q.

Definition ChurchRosser :=
  ∀ p q, (R+S)⋆ p q → ∃s, S⋆ p s ∧ R⋆ s q.
```

Church-Rosser, with points

Do not read this slide!

```
(** naive proof **)
Theorem WeakConfluence_is_ChurchRosser0:
  WeakConfluence → ChurchRosser.
Proof.
intros H p q Hpq.
induction Hpq as [p | p q q' Hpq Hqq' IH].

  ∃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'].
  ∃s. assumption.
  apply trans_rt1n.
  apply rt_trans with s';
  apply rt1n_trans;
  assumption.

  destruct IH as [s Hqs Hsq']. ■
  ∃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 · 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_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.
```

```
■
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 · S ⋆⊆ S⋆ · R⋆
============================

R · (S⋆ · R⋆) ⊆ S⋆ · R⋆
```

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_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.
```

```
■
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 · S ⋆⊆ S⋆ · R⋆
============================

(R · S⋆ ) · R⋆ ⊆ S⋆ · R⋆
```

Church-Rosser, with tools

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

```
Theorem WeakConfluence_is_ChurchRosser2:
  R · S⋆ ⊆ S⋆ · R⋆ → (R+S)⋆ ⊆ S⋆ · 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.
```

```
■
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 · S ⋆ ⊆ S ⋆ · R ⋆
============================

1 + (R + S) · (S ⋆ · R ⋆) ⊆ S ⋆ · R ⋆
```

Church-Rosser, with tools

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

```
Theorem WeakConfluence_is_ChurchRosser2:
  R · S⋆ ⊆ S⋆ · R⋆ → (R+S)⋆ ⊆ S⋆ · 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.
```

```
■
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 · S ⋆⊆ S ⋆· R ⋆
============================

1 + R · S⋆ · R⋆ + S · S⋆ · R⋆ ⊆ S⋆ · R⋆
```

Church-Rosser, with tools

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

```
Theorem WeakConfluence_is_ChurchRosser2:
  R · S⋆ ⊆ S⋆ · R⋆ → (R+S)⋆ ⊆ S⋆ · 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.
```

```
■
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 · S⋆ ⊆ S⋆ · R⋆
============================

1 ⊆ 1 · 1
```

Church-Rosser, with tools

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

```
Theorem WeakConfluence_is_ChurchRosser2:
  R · S⋆ ⊆ S⋆ · R⋆ → (R+S)⋆ ⊆ S⋆ · 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.
```

```
■
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 · S ⋆⊆ S ⋆· R ⋆
star_trans : ∀ R, R ⋆· R ⋆== R ⋆
============================

(S⋆ · R⋆) · R ⋆⊆ S⋆ · R⋆
```

Church-Rosser, with better tools

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

```
Theorem WeakConfluence_is_ChurchRosser3:
   R · S⋆ ⊆ S⋆ · R⋆ → (R+S)⋆ ⊆ S⋆ · 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 · S⋆ ⊆ S⋆ · R⋆
============================
 1 + S⋆ · R⋆ · R⋆ + S · S⋆ · R⋆ ⊆ S⋆ · R⋆
```

Church-Rosser, with better tools

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

```
Theorem WeakConfluence_is_ChurchRosser3:
   R · S⋆ ⊆ S⋆ · R⋆ → (R+S)⋆ ⊆ S⋆ · R⋆.
Proof.
intro H.
star_left_induction.
semiring_normalize.
rewrite H. ■
kleene_reflexivity.
Qed.
```

```
■
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 · S⋆ ⊆ S⋆ · R⋆
============================

 1 + S⋆ · R⋆ · R⋆ + S · S⋆ · R⋆ ⊆ S⋆ · 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

Scott vs. Kozen

Let α and β be two regular expressions $(+, \cdot, 0, 1, ^\star)$.

Scott '50 α and β represent the same language iff the corresponding minimal automata are isomorphic.

Scott vs. Kozen

Let α and β be two regular expressions $(+, \cdot, 0, 1, ^\star)$.

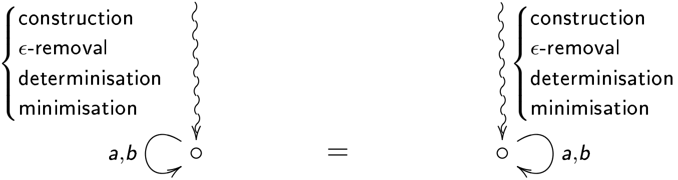
Scott '50 α and β represent the same language iff the corresponding minimal automata are isomorphic.

Kozen '94 Initiality of this model for Kleene algebras:
If α and β 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

$(a+b)^*$ $\qquad\qquad$ $a^* \cdot (b \cdot a^*)^*$



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:

```
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:

```
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.
```

($\equiv$ is the equality generated by the axioms of Kleene Algebras)

- Then we wrap this in a tactic.

Kozen's Proof

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

$$\begin{pmatrix}\cdots & u & \cdots\end{pmatrix} \cdot \begin{pmatrix}\cdots & \cdots & \cdots \\ \cdots & M & \cdots \\ \cdots & \cdots & \cdots\end{pmatrix} \cdot \begin{pmatrix}\vdots \\ v \\ \vdots\end{pmatrix}$$

Kozen's Proof

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

$$\begin{pmatrix} \cdots & u & \cdots \end{pmatrix} \cdot \begin{pmatrix} \cdots & \cdots & \cdots \\ \cdots & M & \cdots \\ \cdots & \cdots & \cdots \end{pmatrix}^{\star} \cdot \begin{pmatrix} \vdots \\ v \\ \vdots \end{pmatrix}$$

- Matrices over a Kleene algebra form a Kleene algebra.

Kozen's Proof

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

$$\begin{pmatrix} \cdots & u & \cdots \end{pmatrix} \cdot \begin{pmatrix} \cdots & \cdots & \cdots \\ \cdots & M & \cdots \\ \cdots & \cdots & \cdots \end{pmatrix}^{\star} \cdot \begin{pmatrix} \vdots \\ v \\ \vdots \end{pmatrix} \quad =_{\mathcal{A}} \quad \alpha$$

- Matrices over a Kleene algebra form a Kleene algebra.

Kozen's Proof

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

$$\begin{pmatrix} \cdots & u & \cdots \end{pmatrix} \cdot \begin{pmatrix} \cdots & \cdots & \cdots \\ \cdots & M & \cdots \\ \cdots & \cdots & \cdots \end{pmatrix}^{\star} \cdot \begin{pmatrix} \vdots \\ v \\ \vdots \end{pmatrix} \quad =_{\mathcal{A}} \quad \alpha$$

- 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:

$$\begin{pmatrix} \cdots & u & \cdots \end{pmatrix} \cdot \begin{pmatrix} \cdots & \cdots & \cdots \\ \cdots & M & \cdots \\ \cdots & \cdots & \cdots \end{pmatrix}^{\star} \cdot \begin{pmatrix} \vdots \\ v \\ \vdots \end{pmatrix} \quad =_{\mathcal{A}} \quad \alpha$$

- 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

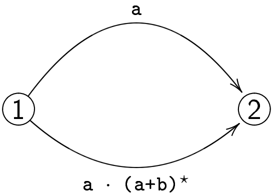
$$a + a \cdot (a{+}b)^\star$$

	1	2
1		
2		

Construction

A variant of Illie and Yu's

$$\mathtt{a} + \mathtt{a} \cdot (\mathtt{a}+\mathtt{b})^\star$$

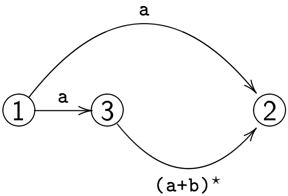


	1	2
1		a
2		

Construction

A variant of Illie and Yu's

a + a · (a+b)*

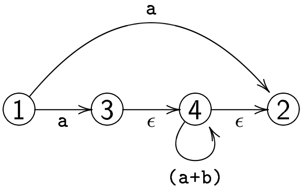


	1	2	3
1		a	a
2			
3			

Construction

A variant of Illie and Yu's

$$\texttt{a} + \texttt{a} \cdot (\texttt{a+b})^\star$$

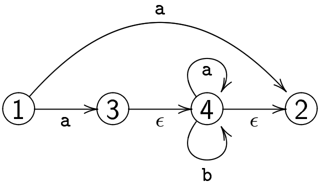


	1	2	3	4
1		a	a	
2				
3				ϵ
4		ϵ		

Construction

A variant of Illie and Yu's

a + a · (a+b)*



	1	2	3	4
1		a	a	
2				
3				ϵ
4		ϵ		a,b

About the construction

- We prove that the construction is correct algebraically

$$\begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & \mathtt{a} & \mathtt{a} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \epsilon \\ 0 & \epsilon & 0 & \mathtt{a+b} \end{pmatrix}^{\star} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} =_{\mathcal{A}} \mathtt{a} + \mathtt{a} \cdot (\mathtt{a+b})^{\star}$$

About the construction

- We prove that the construction is correct algebraically

$$\begin{pmatrix}1 & 0 & 0 & 0\end{pmatrix} \cdot \begin{pmatrix}0 & \mathtt{a} & \mathtt{a} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \epsilon \\ 0 & \epsilon & 0 & \mathtt{a+b}\end{pmatrix}^{\star} \cdot \begin{pmatrix}0 \\ 1 \\ 0 \\ 0\end{pmatrix} =_{\mathcal{A}} \mathtt{a} + \mathtt{a} \cdot (\mathtt{a+b})^{\star}$$

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

	1	2	3	4
1		a	a	
2				
3				ϵ
4		ϵ		a,b

$1 \xrightarrow{\mathtt{a}} \{2, 3\}$
$4 \xrightarrow{\mathtt{a}} \{4\}$
$4 \xrightarrow{\mathtt{b}} \{4\}$

$3 \xrightarrow{\epsilon} \{4\}$
$4 \xrightarrow{\epsilon} \{2\}$

About the construction

- We prove that the construction is correct algebraically

$$\begin{pmatrix}1 & 0 & 0 & 0\end{pmatrix} \cdot \begin{pmatrix}0 & \mathtt{a} & \mathtt{a} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \epsilon \\ 0 & \epsilon & 0 & \mathtt{a+b}\end{pmatrix}^{\star} \cdot \begin{pmatrix}0 \\ 1 \\ 0 \\ 0\end{pmatrix} =_{\mathcal{A}} \mathtt{a} + \mathtt{a} \cdot (\mathtt{a+b})^{\star}$$

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

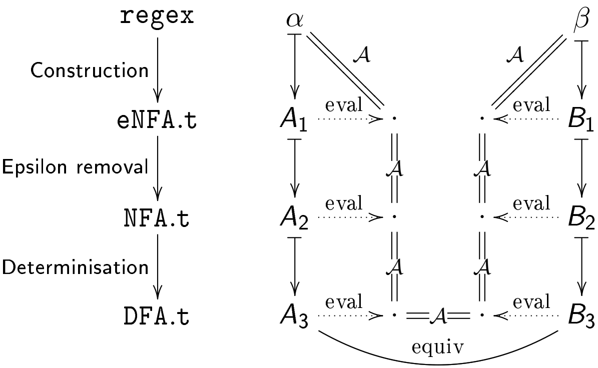
	1	2	3	4
1		a	a	
2				
3				ϵ
4		ϵ		a,b

$$1 \xrightarrow{\mathtt{a}} \{2,3\} \quad 4 \xrightarrow{\mathtt{a}} \{4\} \quad 4 \xrightarrow{\mathtt{b}} \{4\}$$

$$3 \xrightarrow{\epsilon} \{4\} \quad 4 \xrightarrow{\epsilon} \{2\}$$

- We prove that the constructions in the algebraic setting and the efficient setting are equivalent

The big picture
and the datastructures



No minimisation (too costly)

Outline

Algebraic hierarchy
another one

- We follow the mathematical algebraic hierarchy using Typeclasses:

  SemiLattice

  $<:$ SemiRing $<:$ KleeneAlg $<: \ldots$

  Monoid

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

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

Algebraic hierarchy

another one

- We follow the mathematical algebraic hierarchy using Typeclasses:

$$\begin{matrix}\text{SemiLattice} \\ \\ \text{Monoid}\end{matrix} <: \text{SemiRing} <: \text{KleeneAlg} <: \ldots$$

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

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

`What about matrices ?`

Matrices

- Infinite fonctions, with a constrained pointwise equality:

```
Definition MX n m := nat → nat → X.

Definition equal n m (M N : MX n m) :=
    ∀ i j, i<n → j<m → M i j  ≡  N i j.
```

- No bound proofs required for the access

Matrices

- Infinite fonctions, with a constrained pointwise equality:

```
Definition MX n m := nat → nat → X.

Definition equal n m (M N : MX n m) :=
    ∀ i j, i<n → j<m → M i j  ≡  N i j.
```

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

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

```
Fixpoint sum k (f: nat → X) :=
    match k with 0 ⇒0 | S k ⇒f k + sum k f end.

Definition dot n m p (M: MX n m) (N: MX m p) :=
    fun i j ⇒sum m (fun k ⇒M i k ∗ N k j).
```

Matrices

- Infinite fonctions, with a constrained pointwise equality:

```
Definition MX n m := nat → nat → X.

Definition equal n m (M N : MX n m) :=
    ∀ i j, i<n → j<m → M i j  ≡  N i j.
```

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

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

```
Fixpoint sum k (f: nat → X) :=
    match k with 0 ⇒0 | S k ⇒f k + sum k f end.

Definition dot n m p (M: MX n m) (N: MX m p) :=
    fun i j ⇒sum m (fun k ⇒M i k * N k j).
```

bounds proofs are easy to cope with, in proof mode

no hidden boilerplate

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.

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 $(\cdot, +, \dots)$ are partial (dimensions have to agree)

```
X: Type.

dot: X → X → X.
one: X.

plus: X → X → X.
zero: X.

star: X → X.

dot_neutral_left:
 ∀ x, 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 $(\cdot, +, \dots)$ are partial (dimensions have to agree)

```
                             T: Type.
X: Type.                     X: T → T → Type.

dot: X → X → X.              dot: ∀ n m p, X n m → X m p → X n p.
one: X.                      one: ∀ n, X n n.

plus: X → X → X.             plus: ∀ n m, X n m → X n m → X n m.
zero: X.                     zero: ∀ n m, X n m.

star: X → X.                 star: ∀ n, X n n → X n n.

dot_neutral_left:            dot_neutral_left:
 ∀ x, dot one x = x.          ∀ n m (x: X n m), 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 $(\cdot, +, \ldots)$ are partial (dimensions have to agree)

```
X: Type.

dot: X → X → X.
one: X.

plus: X → X → X.
zero: X.

star: X → X.

dot_neutral_left:
 ∀ x, dot one x = x.

...
```

```
T: Type.
X: T → T → Type.

dot: ∀ n m p, X n m → X m p → X n p.
one: ∀ n, X n n.

plus: ∀ n m, X n m → X n m → X n m.
zero: ∀ n m, X n m.

star: ∀ n, X n n → X n n.

dot_neutral_left:
 ∀ n m (x: X n m), dot one x = x.

...
```

- Introduce typed structures from the beginning

Typed structures

We handle heterogeneous relations (`X A B := A → B → Prop`), as well as matrices:

`MxSemiLattice` : `SemiLattice` → `SemiLattice`.
`MxSemiRing` : `SemiRing` → `SemiRing`.
`MxKleeneAlgebra` : `KleeneAlgebra` → `KleeneAlgebra`.

Here, we deal with typed structures

- All theorems are inherited at the matricial level.

Typed structures

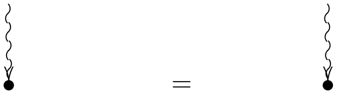
We handle heterogeneous relations (`X A B := A → B → Prop`), as well as matrices:

$$\texttt{MxSemiLattice} : \texttt{SemiLattice} \to \texttt{SemiLattice}.$$
$$\texttt{MxSemiRing} : \texttt{SemiRing} \to \texttt{SemiRing}.$$
$$\texttt{MxKleeneAlgebra} : \texttt{KleeneAlgebra} \to \texttt{KleeneAlgebra}.$$

Here, we deal with typed structures

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

$$a \cdot (b \cdot a)^\star \quad \stackrel{?}{=} \quad (a \cdot b)^\star \cdot a$$



Typed structures

We handle heterogeneous relations (`X A B := A → B → Prop`), as well as matrices:

```
MxSemiLattice : SemiLattice → SemiLattice.
MxSemiRing : SemiRing → SemiRing.
MxKleeneAlgebra : KleeneAlgebra → KleeneAlgebra.
```

Here, we deal with typed structures

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

$$a \cdot (b \cdot a)^{\star} \stackrel{?}{=} (a \cdot b)^{\star} \cdot a \quad : \quad A \to B$$

$$\bullet = \bullet$$

$$a : A \to B, b : B \to A$$

Typed structures

We handle heterogeneous relations (`X A B := A → B → Prop`), as well as matrices:

`MxSemiLattice : SemiLattice → SemiLattice.`
`MxSemiRing : SemiRing → SemiRing.`
`MxKleeneAlgebra : KleeneAlgebra → KleeneAlgebra.`

Here, we deal with typed structures

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

$$a \cdot (b \cdot a)^\star \quad \stackrel{?}{=} \quad (a \cdot b)^\star \cdot a \quad : \quad A \rightarrow B$$

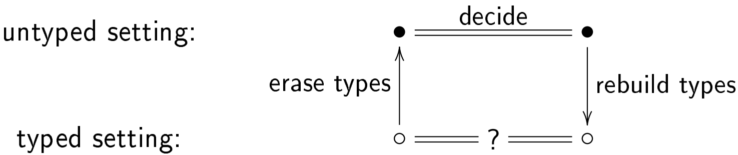
$$\bullet \quad = \quad \bullet$$

$a : A \rightarrow B, b : B \rightarrow A$

tackle the problem differently... let's erase types!

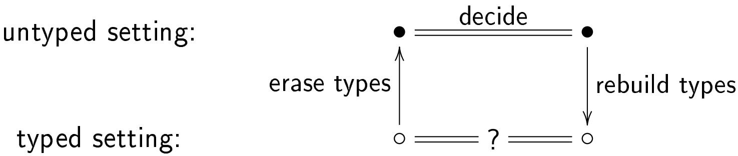
Untyping

The general scheme



Untyping

The general scheme



- Depending on the algebraic structure:

$\mathcal{A}$	
semi-lattices	trivial
monoids	rather easy
semirings	tricky
Kleene algebras	same as for semirings
residuated lattices	with constraints
action algebras/lattices	?

So, everything is fine. . .

Outline

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:

$$\overline{M}^{\star} \cdot X = X \cdot M^{\star} \qquad \overline{u} \cdot X = u$$

- We deduce

$$\begin{aligned} \overline{u} \cdot \overline{M}^{\star} \cdot X \cdot v &= \overline{u} \cdot X \cdot M^{\star} \cdot v \\ &= u \cdot M^{\star} \cdot v \end{aligned}$$