Implementing Rho-Calculus in TOM

with the help of slides from C. Kirchner, P.-E. Moreau, A. Reilles, E. Balland

Germain Faure
LORIA-UHP

Horatiu Cirstea
LORIA-Nancy2

Claude Kirchner
LORIA-INRIA

Pierre-Etienne Moreau
LORIA-INRIA

The title is not: A full presentation of TOM

Once upon a time [...]

and the end of the story was:

Nice: how can I use it?

elan provides a robust and efficient implementation of these concepts [...]

But...

I have 2 000 000 lines of C code, should I rewrite everything in ELAN ?

Is formal methods (especially rewriting) arriving too late?

Is formal methods (especially rewriting) arriving too late?

YES! Read newspapers: $40 + 40 = 100$

Is formal methods (especially rewriting) arriving too late?

YES! Read newspapers: $40 + 40 = 100$

Ariane et David se sont rendus sur le site Internet de la SNCF pour y reserver leurs billets. Deux allers et retours pour deux adultes. [...] Les deux allers coûtent 40 euros, idem pour les deux retours. Ariane et David s'attendent, comme 40 et 40 font 80, s'attendent à s'acquitter de cette somme. Sauf que le total calculé par la SNCF donne 100 euros... [...] Renseignements pris il s'agit d'un bug informatique. [...]

"Le Canard Enchainé [07/07/2004]"

Is formal methods (especially rewriting) arriving too late?

YES! Read newspapers: $40 + 40 = 100$

Ariane et David se sont rendus sur le site Internet de la SNCF pour y reserver leurs billets. Deux allers et retours pour deux adultes. [...] Les deux allers coûtent 40 euros, idem pour les deux retours. Ariane et David s'attendent, comme 40 et 40 font 80, s'attendent à s'acquitter de cette somme. Sauf que le total calculé par la SNCF donne 100 euros... [...] Renseignements pris il s'agit d'un bug informatique. [...]

"Le Canard Enchainé [07/07/2004]"

YES! The current software corpus is huge and we still develop C code

Is formal methods (especially rewriting) arriving too late?

YES! Read newspapers: $40 + 40 = 100$

Ariane et David se sont rendus sur le site Internet de la SNCF pour y reserver leurs billets. Deux allers et retours pour deux adultes. [...] Les deux allers coûtent 40 euros, idem pour les deux retours. Ariane et David s'attendent, comme 40 et 40 font 80, s'attendent à s'acquitter de cette somme. Sauf que le total calculé par la SNCF donne 100 euros... [...] Renseignements pris il s'agit d'un bug informatique. [...]

"Le Canard Enchainé [07/07/2004]"

YES! The current software corpus is huge and we still develop C code

APSIT, l'Agence des Prestataires de Services en Informatiques et Télécoms recherche pour l'un de ses clients Grand Compte 2 Ingénieurs de développement C. Vous possédez une solide expérience de développement en langage de programmation C [...]

Is formal methods (especially rewriting) arriving too late?

YES! Read newspapers: $40 + 40 = 100$

Ariane et David se sont rendus sur le site Internet de la SNCF pour y reserver leurs billets. Deux allers et retours pour deux adultes. [...] Les deux allers coûtent 40 euros, idem pour les deux retours. Ariane et David s'attendent, comme 40 et 40 font 80, s'attendent à s'acquitter de cette somme. Sauf que le total calculé par la SNCF donne 100 euros... [...] Renseignements pris il s'agit d'un bug informatique. [...]

"Le Canard Enchainé [07/07/2004]"

YES! The current software corpus is huge and we still develop C code

APSIT, l'Agence des Prestataires de Services en Informatiques et Télécoms recherche pour l'un de ses clients Grand Compte 2 Ingénieurs de développement C. Vous possédez une solide expérience de développement en langage de programmation C [...]

But

Is formal methods (especially rewriting) arriving too late?

YES! Read newspapers: $40 + 40 = 100$

Ariane et David se sont rendus sur le site Internet de la SNCF pour y reserver leurs billets. Deux allers et retours pour deux adultes. [...] Les deux allers coûtent 40 euros, idem pour les deux retours. Ariane et David s'attendent, comme 40 et 40 font 80, s'attendent à s'acquitter de cette somme. Sauf que le total calculé par la SNCF donne 100 euros... [...] Renseignements pris il s'agit d'un bug informatique. [...]

"Le Canard Enchainé [07/07/2004]"

YES! The current software corpus is huge and we still develop C code

APSIT, l'Agence des Prestataires de Services en Informatiques et Télécoms recherche pour l'un de ses clients Grand Compte 2 Ingénieurs de développement C. Vous possédez une solide expérience de développement en langage de programmation C [...]

But At significant cost this is usable

Is formal methods (especially rewriting) arriving too late?

YES! Read newspapers: $40 + 40 = 100$

Ariane et David se sont rendus sur le site Internet de la SNCF pour y reserver leurs billets. Deux allers et retours pour deux adultes. [...] Les deux allers coûtent 40 euros, idem pour les deux retours. Ariane et David s'attendent, comme 40 et 40 font 80, s'attendent à s'acquitter de cette somme. Sauf que le total calculé par la SNCF donne 100 euros... [...] Renseignements pris il s'agit d'un bug informatique. [...]

"Le Canard Enchainé [07/07/2004]"

YES! The current software corpus is huge and we still develop C code

APSIT, l'Agence des Prestataires de Services en Informatiques et Télécoms recherche pour l'un de ses clients Grand Compte 2 Ingénieurs de développement C. Vous possédez une solide expérience de développement en langage de programmation C [...]

But At significant cost this is usable

It is fundamentally useful

How to make rewriting usable, to share and reuse our technology, our software?

How to make rewriting usable, to share and reuse our technology, our software?

INTEGRATE REWRITING INTO EXISTING PROGRAMMING LANGUAGES

How to make rewriting usable, to share and reuse our technology, our software?

INTEGRATE REWRITING INTO EXISTING PROGRAMMING LANGUAGES

- Usable at code design or maintenance time.

How to make rewriting usable, to share and reuse our technology, our software?

INTEGRATE REWRITING INTO EXISTING PROGRAMMING LANGUAGES

- Usable at code design or maintenance time.
- Allows one to use proof techniques to validate properties of the formal part of the code.

How to make rewriting usable, to share and reuse our technology, our software?

INTEGRATE REWRITING INTO EXISTING PROGRAMMING LANGUAGES

- Usable at code design or maintenance time.
- Allows one to use proof techniques to validate properties of the formal part of the code.
- An intensive use induces absolutely no dependence since the formal part is compiled into the target language.

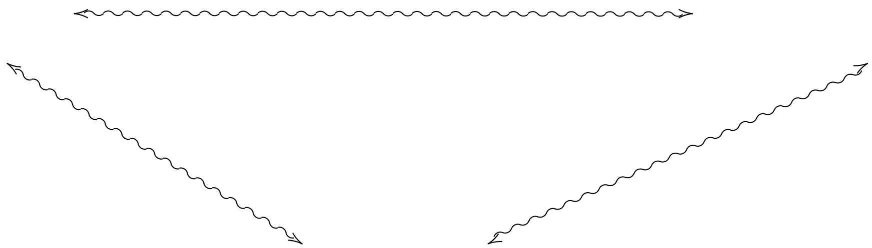
How to make rewriting usable, to share and reuse our technology, our software?

INTEGRATE REWRITING INTO EXISTING PROGRAMMING LANGUAGES

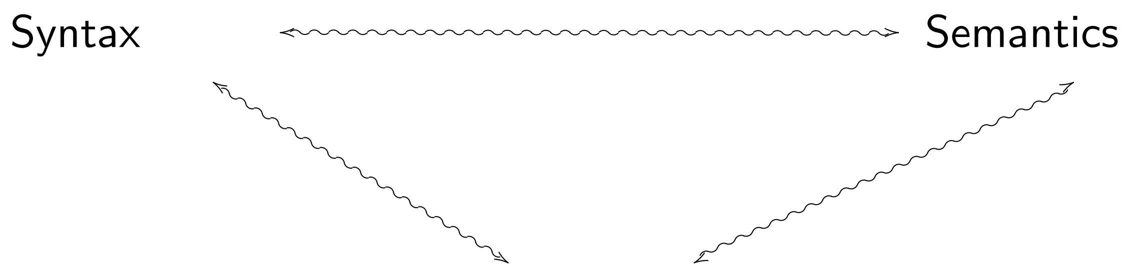
- Usable at code design or maintenance time.
- Allows one to use proof techniques to validate properties of the formal part of the code.
- An intensive use induces absolutely no dependence since the formal part is compiled into the target language.
- The generated code is safer.

The TOM-project

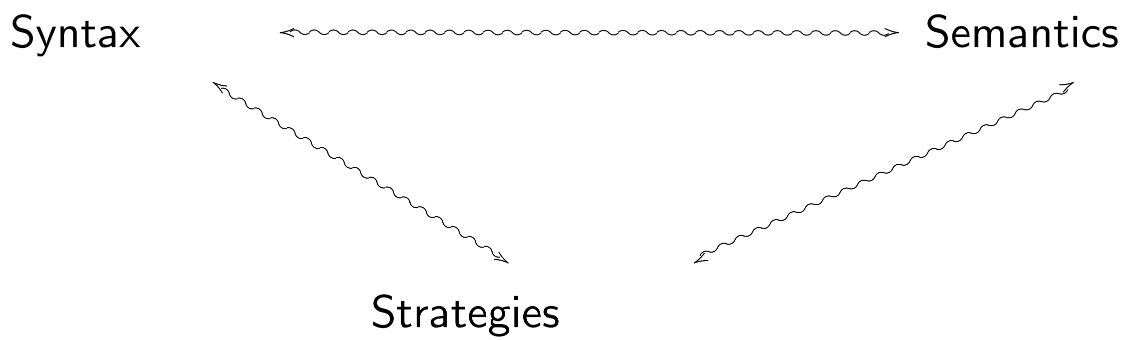
Syntax



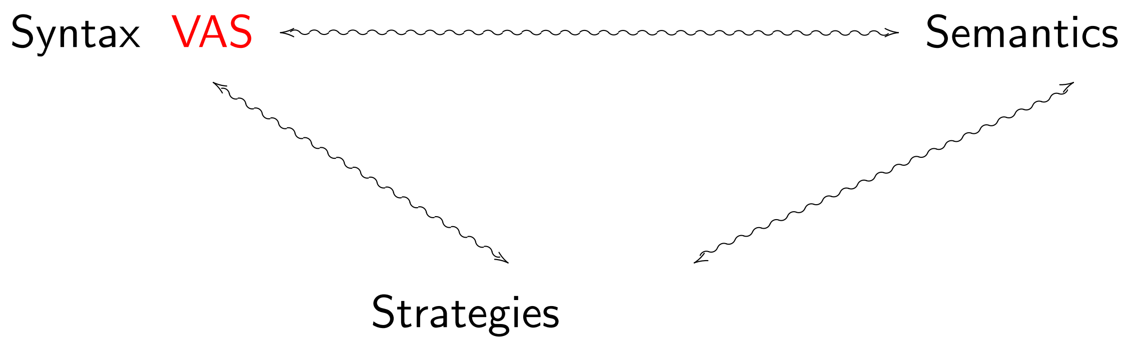
The TOM-project



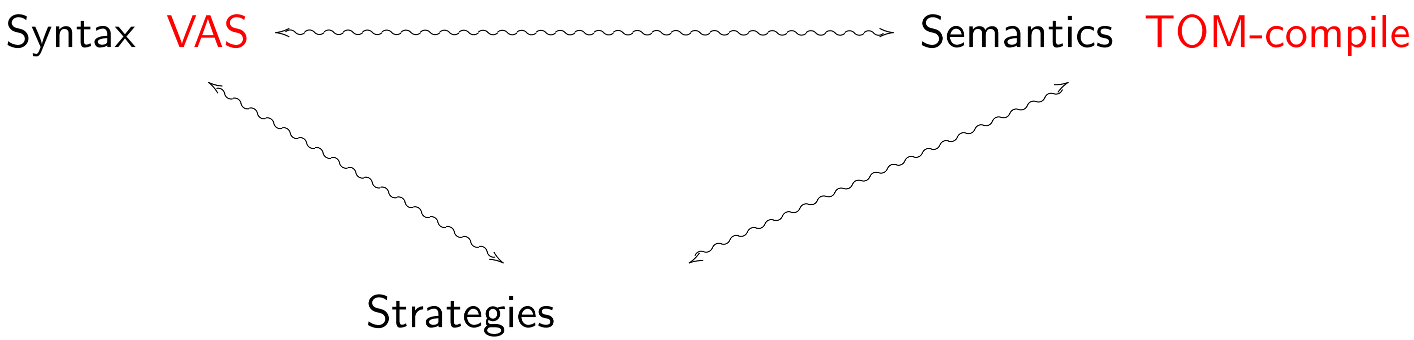
The TOM-project



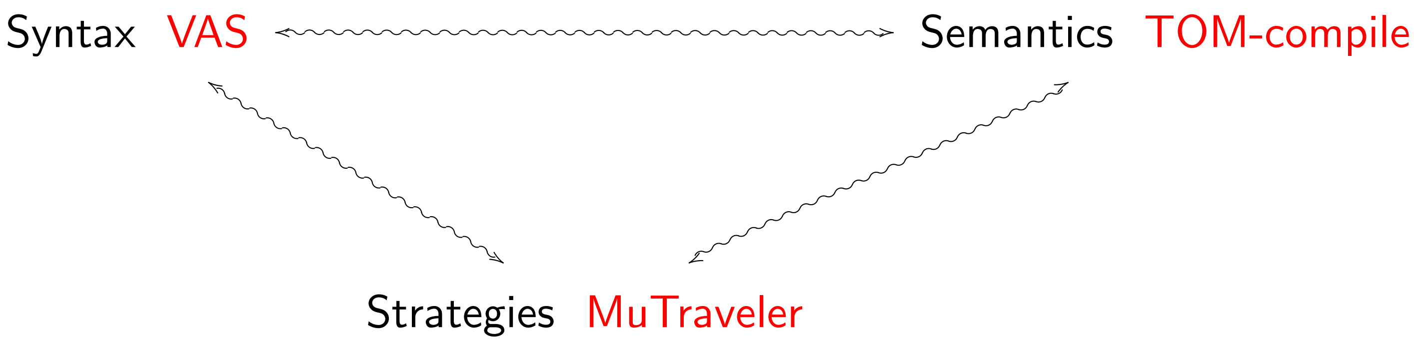
The TOM-project



The TOM-project



The TOM-project



Syntax of the $\rho_{\mathtt{x}}^{\circ}$

Terms	M, N, P	$::=$	X	(Variables)
		$\mid$	c	(Constants)
		$\mid$	$P \twoheadrightarrow M$	(Abstraction)
		$\mid$	$M \bullet N$	(Functional application)
		$\mid$	$M \wr N$	(Structure)
		$\mid$	$[\phi]\, N$	(Substitution application)
		$\mid$	$[\mathcal{C}]\, N$	(Constraint application)

Syntax of the $\rho_{\mathtt{x}}^{\circ}$

Terms	M, N, P	$::=$	X	(Variables)
		$\mid$	c	(Constants)
		$\mid$	$P \twoheadrightarrow M$	(Abstraction)
		$\mid$	$M \bullet N$	(Functional application)
		$\mid$	$M \wr N$	(Structure)
		$\mid$	$[\phi]\, N$	(Substitution application)
		$\mid$	$[\mathcal{C}]\, N$	(Constraint application)
Substitutions	ϕ, ψ	$::=$	id_s	(Identity)
		$\mid$	$X = M$	(Equation)
		$\mid$	$\phi \wedge \psi$	(Conjunction of equations)

Syntax of the $\rho_{\mathtt{x}}^{\circ}$

Terms	M, N, P	$::=$	X	(Variables)
		$\mid$	c	(Constants)
		$\mid$	$P \twoheadrightarrow M$	(Abstraction)
		$\mid$	$M^\bullet N$	(Functional application)
		$\mid$	$M \wr N$	(Structure)
		$\mid$	$[\phi]\ N$	(Substitution application)
		$\mid$	$[\mathcal{C}]\ N$	(Constraint application)
Substitutions	ϕ, ψ	$::=$	id_s	(Identity)
		$\mid$	$X = M$	(Equation)
		$\mid$	$\phi \wedge \psi$	(Conjunction of equations)
Constraints	$\mathcal{C}, \mathcal{D}$	$::=$	id_m	(Identity)
		$\mid$	$P \ll M$	(Match-equation)
		$\mid$	$\mathcal{C} \wedge \mathcal{D}$	(Conjunction of constraints)

Syntax of the $\rho_{\mathrm{x}}^{\circ}$

Terms	M, N, P	$::=$	X	(Variables)
		$\mid$	c	(Constants)
		$\mid$	$P \twoheadrightarrow M$	(Abstraction)
		$\mid$	$M^{\bullet} N$	(Functional application)
		$\mid$	$M \wr N$	(Structure)
		$\mid$	$[\phi]\, N$	(Substitution application)
		$\mid$	$[\mathcal{C}]\, N$	(Constraint application)
Substitutions	ϕ, ψ	$::=$	id_s	(Identity)
		$\mid$	$X = M$	(Equation)
		$\mid$	$\phi \wedge \psi$	(Conjunction of equations)
Constraints	$\mathcal{C}, \mathcal{D}$	$::=$	id_m	(Identity)
		$\mid$	$P \ll M$	(Match-equation)
		$\mid$	$\mathcal{C} \wedge \mathcal{D}$	(Conjunction of constraints)

where $\wedge$ is associative and id_s and id_m are neutral elements

From rewrite rules to constraints

(ρ)	$(P \rightarrow M)^{\bullet} N$	$\longrightarrow$	$[P \ll N]M$
(δ)	$(M_1 \wr M_2)^{\bullet} N$	$\longrightarrow$	$M_1{}^{\bullet} N \wr M_2{}^{\bullet} N$

From rewrite rules to constraints

(ρ)	$(P \twoheadrightarrow M)^{\bullet} N$	$\longrightarrow$	$[P \ll N]M$
(δ)	$(M_1 \wr M_2)^{\bullet} N$	$\longrightarrow$	$M_1{}^{\bullet} N \wr M_2{}^{\bullet} N$

From constraints to substitutions

Decomposition

$(Decompose_{\wr})$	$M_1 \wr M_2 \ll N_1 \wr N_2$	$\longrightarrow$	$M_1 \ll N_1 \wedge M_2 \ll N_2$
$(Decompose_{\mathcal{F}})$	$f(M_1, \ldots, M_n) \ll f(N_1, \ldots, N_n)$	$\longrightarrow$	$\bigwedge_{i=1}^{n} (M_i \ll N_i)$

From rewrite rules to constraints

(ρ)	$(P \twoheadrightarrow M)^{\bullet} N$	$\to$	$[P \ll N] M$
(δ)	$(M_1 \wr M_2)^{\bullet} N$	$\to$	$M_1{}^{\bullet} N \wr M_2{}^{\bullet} N$

From constraints to substitutions

Decomposition

$(Decompose_{\wr})$	$M_1 \wr M_2 \ll N_1 \wr N_2$	$\to$	$M_1 \ll N_1 \wedge M_2 \ll N_2$
$(Decompose_{\mathcal{F}})$	$f(M_1, \ldots, M_n) \ll f(N_1, \ldots, N_n)$	$\to$	$\bigwedge_{i=1}^{n} (M_i \ll N_i)$

From constraints to substitutions

$(Idem)$	$\mathcal{C} \wedge (X \ll M) \wedge \mathcal{D} \wedge (X \ll M) \wedge \mathcal{E}$	$\to$	$\mathcal{C} \wedge (X \ll M) \wedge \mathcal{D} \wedge \mathcal{E}$
$(ToSubst)$	$[\mathcal{C} \wedge (X \ll M) \wedge \mathcal{D}]\, N$	$\to$	$[\mathcal{C} \wedge \mathcal{D}]\, ([X = M]\, N)$ if $X \notin \mathcal{D}om(\mathcal{C} \wedge \mathcal{D})$

From rewrite rules to constraints

(ρ)	$(P \rightarrow M)^{\bullet} N$	$\rightarrow$	$[P \ll N] M$	
(δ)	$(M_1 \wr M_2)^{\bullet} N$	$\rightarrow$	$M_1{}^{\bullet} N \wr M_2{}^{\bullet} N$	

From constraints to substitutions

Decomposition

$(Decompose_{\wr})$	$M_1 \wr M_2 \ll N_1 \wr N_2$	$\rightarrow$	$M_1 \ll N_1 \wedge M_2 \ll N_2$	
$(Decompose_{\mathcal{F}})$	$f(M_1, \ldots, M_n) \ll f(N_1, \ldots, N_n)$	$\rightarrow$	$\bigwedge_{i=1}^{n} (M_i \ll N_i)$	

From constraints to substitutions

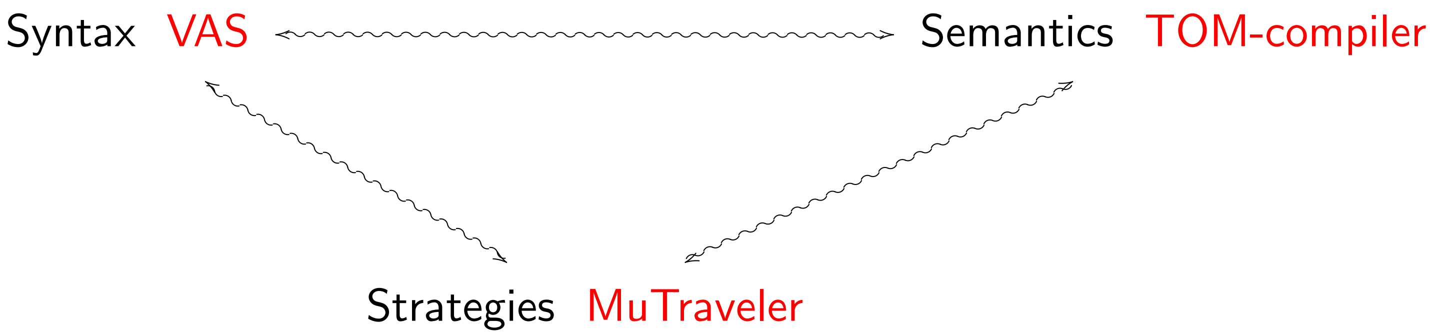
$(Idem)$	$\mathcal{C} \wedge (X \ll M) \wedge \mathcal{D} \wedge (X \ll M) \wedge \mathcal{E}$	$\rightarrow$	$\mathcal{C} \wedge (X \ll M) \wedge \mathcal{D} \wedge \mathcal{E}$	
$(ToSubst)$	$[\mathcal{C} \wedge (X \ll M) \wedge \mathcal{D}]\, N$	$\rightarrow$	$[\mathcal{C} \wedge \mathcal{D}]\, ([X = M]\, N)$	if $X \notin \mathcal{D}om(\mathcal{C} \wedge \mathcal{D})$

Substitution applications

$(Identity)$	$[\mathtt{id}]\, M$	$\rightarrow$	M	
$(Replace)$	$[\phi \wedge X = M \wedge \psi]\, X$	$\rightarrow$	M	
(Var)	$[\phi]\, Y$	$\rightarrow$	Y	$Y \notin \mathcal{D}om(\phi)$
$(Const)$	$[\phi]\, c$	$\rightarrow$	c	
(Abs)	$[\phi]\, (P \rightarrow M)$	$\rightarrow$	$P \rightarrow [\phi]\, M$	
(App)	$[\phi]\, (M^{\bullet} N)$	$\rightarrow$	$[\phi]\, M^{\bullet}\, [\phi]\, N$	
$(Struct)$	$[\phi]\, (M \wr N)$	$\rightarrow$	$[\phi]\, M \wr [\phi]\, N$	
$(Constraint)$	$[\phi]\, ([P_i \ll N_i]_{i=1}^{n}(M))$	$\rightarrow$	$[P_i \ll [\phi]\, N_i]_{i=1}^{n}([\phi]\, M)$	$n > 0$
$(Compose)$	$[\phi]\, ([X_i = M_i]_{i=1}^{n}(N))$	$\rightarrow$	$[\phi \wedge X_i = [\phi]\, M_i]_{i=1}^{n}(N)$	$n > 0$

Examples

The TOM-project



Syntax VAS

- Terms could be user defined.
- Mapping between algebraic and concrete sorts are automatically generated by TOM.

Syntax VAS

- Terms could be user defined.
- Mapping between algebraic and concrete sorts are automatically generated by TOM.

$$
\begin{array}{rcr}
M, N & ::= & X \\
 & | & c \\
 & | & M \bullet N \\
 & | & \ldots
\end{array}
$$

```
%vas{    [...]
   var(na:String)                  -> RTerm
   const(na:String)                -> RTerm
   app(lhs:RTerm, rhs:RTerm)       -> RTerm
   [...]
}
```

Back to The TOM-project

Syntax VAS

- List Matching (and thus Associative and Unitary matching [Reilles05])

Syntax VAS

- List Matching (and thus Associative and Unitary matching [Reilles05])

- $i : Elem \to List$

	in TOM	Usually
Neutral Element	$conc()$	e
Injection	$conc(a)$	$i(a)$
List Element	$conc(X*)$	X

Syntax VAS

- List Matching (and thus Associative and Unitary matching [Reilles05])

- $i : Elem \rightarrow List$

	in TOM	Usually
Neutral Element	$conc()$	e
Injection	$conc(a)$	$i(a)$
List Element	$conc(X*)$	X

$$
\begin{array}{lcr}
M, N & ::= & X \\
 & \mid & c \\
 & \mid & M \bullet N \\
 & \mid & [\mathcal{C}]\ N \\
 & \mid & \ldots \\
\mathcal{C}, \mathcal{D} & ::= & id_m \\
 & \mid & P \ll M \\
 & \mid & \mathcal{C} \wedge \mathcal{D}
\end{array}
$$

```
%vas{    [...]
   variable(na:String)                      -> RTerm
   constant(na:String)                      -> RTerm
   app(lhs:RTerm, rhs:RTerm)                -> RTerm
   appC(co:ListConst , term:RTerm)          -> RTerm
   [...]

   match(lhs:RTerm, rhs:RTerm)              -> Const
   andC( Const* )                           -> ListConst
```

Back to The TOM-project

TOM-compiler

$$(P \multimap M)^{\bullet} N \qquad \mapsto \quad [P \ll N]\, M$$

```
%match(RTerm arg)

 app(abs(P,M),N) ->
    {return `appC(andC(match(P,N)),M);}
```

TOM-compiler

$$\begin{array}{lcl} (P \rightharpoondown M)^{\bullet} N & \mapsto & [P \ll N]\, M \\ [\mathcal{C} \wedge (X \ll M) \wedge \mathcal{D}]\, N & \mapsto & [\mathcal{C} \wedge \mathcal{D}]\, ([X = M]\, N) \end{array}$$

```
%match(RTerm arg)

 app(abs(P,M),N) ->
    {return `appC(andC(match(P,N)),M);}

 appC(andC(C*,match(X@var[],M),D*),N) ->
    {return `appC(andC(C*,D*),appS(andS(eq(X,M)),N));}
```

Back to The TOM-project

Strategies

- Controlling rewriting by rewriting [BorovanskyKirchnerKirchner WRLA'96]

Strategies

- Controlling rewriting by rewriting [BorovanskyKirchnerKirchner WRLA'96]

- Controlling rewriting: study and implementation of a strategy formalism. [Borovansky WRLA'98]

Strategies

- Controlling rewriting by rewriting [BorovanskyKirchnerKirchner WRLA'96]
- Controlling rewriting: study and implementation of a strategy formalism. [Borovansky WRLA'98]
- A core language for rewriting. [VisserBenaissa WRLA'98 and ICFP'98]

Strategies

- Controlling rewriting by rewriting [BorovanskyKirchnerKirchner WRLA'96]

- Controlling rewriting: study and implementation of a strategy formalism. [Borovansky WRLA'98]

- A core language for rewriting. [VisserBenaissa WRLA'98 and ICFP'98]

- Visitor combination and traversal control. [Visser OOPSLA'01]
  Strategies à la Stratego in an object point of view (Visitor Design Pattern): JJTraveler

Strategies

Definition. *A strategy is defined using:*

1. *Recursion operator:* μ

Strategies

Definition. *A strategy is defined using:*

1. *Recursion operator:* μ
2. *Sequential composition:* $s_1; s_2$

Strategies

Definition. *A strategy is defined using:*

1. *Recursion operator:* μ
2. *Sequential composition:* $s_1; s_2$
3. *Choice:* $choice(s_1, s_2)$ *(if* s_1 *fails then* s_2 *else* s_1*)*

Strategies

Definition. *A strategy is defined using:*

1. *Recursion operator:* μ
2. *Sequential composition:* $s_1; s_2$
3. *Choice:* $choice(s_1, s_2)$ *(if* s_1 *fails then* s_2 *else* s_1*)*
4. *Identity:* id

Strategies

Definition. *A strategy is defined using:*

1. *Recursion operator:* μ
2. *Sequential composition:* $s_1; s_2$
3. *Choice:* $choice(s_1, s_2)$ *(if* s_1 *fails then* s_2 *else* s_1*)*
4. *Identity:* id
5. *Failure:* $fail$

Strategies

Definition. *A strategy is defined using:*

1. *Recursion operator:* μ
2. *Sequential composition:* $s_1; s_2$
3. *Choice:* $choice(s_1, s_2)$ *(if* s_1 *fails then* s_2 *else* s_1*)*
4. *Identity:* id
5. *Failure:* $fail$
6. *Congruent operators*
   (a) $All(s)$*: succeeds if* s *succeeds on all children of the current node.*
   (b) $One(s)$*: succeeds if* s *succeeds on at least on child of the current node.*

Strategies

Definition. *A strategy is defined using:*

1. *Recursion operator:* μ
2. *Sequential composition:* $s_1; s_2$
3. *Choice:* $choice(s_1, s_2)$ *(if* s_1 *fails then* s_2 *else* s_1*)*
4. *Identity:* id
5. *Failure:* $fail$
6. *Congruent operators*
   (a) $All(s)$*: succeeds if* s *succeeds on all children of the current node.*
   (b) $One(s)$*: succeeds if* s *succeeds on at least on child of the current node.*

$$try(s) \quad = \quad choice(s, id)$$

Strategies

Definition. *A strategy is defined using:*

1. *Recursion operator:* μ
2. *Sequential composition:* $s_1; s_2$
3. *Choice:* $choice(s_1, s_2)$ *(if s_1 fails then s_2 else s_1)*
4. *Identity:* id
5. *Failure:* $fail$
6. *Congruent operators*
   (a) $All(s)$*: succeeds if s succeeds on all children of the current node.*
   (b) $One(s)$*: succeeds if s succeeds on at least on child of the current node.*

$$\begin{array}{lcl} try(s) & = & choice(s, id) \\ repeat(s) & = & \mu x.try(s; x) \end{array}$$

Strategies

Definition. *A strategy is defined using:*

1. *Recursion operator:* μ
2. *Sequential composition:* $s_1; s_2$
3. *Choice:* $choice(s_1, s_2)$ *(if* s_1 *fails then* s_2 *else* s_1*)*
4. *Identity:* id
5. *Failure:* $fail$
6. *Congruent operators*
   (a) $All(s)$*: succeeds if* s *succeeds on all children of the current node.*
   (b) $One(s)$*: succeeds if* s *succeeds on at least on child of the current node.*

$$
\begin{array}{lcl}
try(s) & = & choice(s, id) \\
repeat(s) & = & \mu x.try(s; x) \\
oncebu(s) & = & \mu x.choice(One(x), s)
\end{array}
$$

Strategies

Definition. *A strategy is defined using:*

1. *Recursion operator:* μ
2. *Sequential composition:* $s_1; s_2$
3. *Choice:* $choice(s_1, s_2)$ *(if* s_1 *fails then* s_2 *else* s_1*)*
4. *Identity:* id
5. *Failure:* $fail$
6. *Congruent operators*
   (a) $All(s)$*: succeeds if* s *succeeds on all children of the current node.*
   (b) $One(s)$*: succeeds if* s *succeeds on at least on child of the current node.*

$$
\begin{array}{lcl}
try(s) & = & choice(s, id) \\
repeat(s) & = & \mu x.try(s; x) \\
oncebu(s) & = & \mu x.choice(One(x), s) \\
innermost(s) & = & repeat(oncebu(s))
\end{array}
$$

Strategies

Definition. *A strategy is defined using:*

1. *Recursion operator:* μ
2. *Sequential composition:* $s_1; s_2$
3. *Choice:* $choice(s_1, s_2)$ *(if* s_1 *fails then* s_2 *else* s_1*)*
4. *Identity:* id
5. *Failure:* $fail$
6. *Congruent operators*
   (a) $All(s)$*: succeeds if* s *succeeds on all children of the current node.*
   (b) $One(s)$*: succeeds if* s *succeeds on at least on child of the current node.*

$$
\begin{array}{lcl}
try(s) & = & choice(s, id) \\
repeat(s) & = & \mu x.try(s; x) \\
oncebu(s) & = & \mu x.choice(One(x), s) \\
innermost(s) & = & repeat(oncebu(s)) \\
fast-innermost(s) & = & \mu x.All(x); try(s; x)
\end{array}
$$

Exercise: try to define the weak normalization strategy

- In ELAN you need to define congruence rules for each symbol of the signature.

Exercise: try to define the weak normalization strategy

- In ELAN you need to define congruence rules for each symbol of the signature.
- Using MuTraveler library:

$$
\begin{array}{lcl}
oneStepWeakNormalisation & = & \mu x.Choice(rules, One_a bs(x)) \\
myStrategy & = & repeat(oneStepWeakNormalisation)
\end{array}
$$

Exercise: try to define the weak normalization strategy

And print each intermediate step

```
VisitableVisitor myStrategy =
        `Repeat(Seq(oneStepWeakNormalisation,print));
```

OVERALL OF THE IMPLEMENTATION

Weak normalisation strategy for linear patterns

No α-conversion

```
class Rho {
    %vas{
        module rhoterm
            var(na:String) -> RTerm
            const(na:String) -> RTerm
            app(lhs:RTerm,rhs:RTerm) -> RTerm
            appC(co:ListConstraint,term:RTerm) -> RTerm
            [...]
            }

    %op VisitableVisitor One_abs(strat:VisitableVisitor) {
        make(v) {`Sequence(Not_abs(),One(v)) }
    }
    VisitableVisitor rules = new ReductionRules();
    VisitableVisitor print = new Print();
    VisitableVisitor oneStepWeakNormalisation = `mu(MuVar("x"),Choice(rules,One_abs(MuVar("x"))));
    VisitableVisitor myStrategy = `Repeat(oneStepWeakNormalisation);

    RTerm subject = factory.RTermFromString(s);
    System.out.println(myStrategy.visit(subject));
```

```
class Print extends rhotermVisitableFwd {
        Print() {
                super(`Fail());
        }
        RTerm visit_RTerm(RTerm arg) throws  VisitFailure {
                System.out.println("|-->>" + arg);
                return arg;
        }
}
```

```
class Not_abs extends rhotermVisitableFwd {
        Not_abs() {
                super(`Identity());
        }

        RTerm visit_RTerm_Abs(Abs arg) throws jjtraveler.VisitFailure {
                throw new VisitFailure();
        }

}
```

```
class ReductionRules extends rhotermVisitableFwd {
        ReductionRules() {
              super(`Fail());
        }
        RTerm visit_RTerm(RTerm arg) throws  VisitFailure {
              %match(RTerm arg){
                     /*Compose */
                     appS(phi@andS(l*),appS(andS(L*),N)) -> {
                            ListSubst result = `mapS(((ListSubst)(L.reverse())),phi,andS());
                            return `appS(andS(l*,result*),N);}
                     appS(phi,struct(M,N)) -> {return `struct(appS(phi,M),appS(phi,N));}
                     /*Rho*/
                     app(abs(P,M),N) -> {return `appC(andC(match(P,N)),M);}
                     /*Delta*/
                     app(struct(M1,M2),N) -> {return `struct(app(M1,N),app(M2,N));}
               }
               throw new VisitFailure();
        }
        ListConstraint visit_ListConstraint(ListConstraint l) throws VisitFailure {
              %match(ListConstraint l){
                     /*Decompose et Decompose_ng min(n,m) > 0 */
                     l:(X*,m@match(app[],app[]),Y*) -> {
                            ListConstraint head_is_constant = `headIsConstant(m);
                            %match(ListConstraint head_is_constant) {
                                   (match[]) -> { break l; }
                                   (matchKO()) -> { return `andC(X*,matchKO(),Y*); }
                            }
                            ListConstraint result = `computeMatch(andC(m));
                            return `andC(X*,result*,Y*);
                     }
              }
              throw new VisitFailure();}}
```

Conclusion

- I will not give a flash demo (toy – see conjunction and idempotence) but...

Conclusion

- I will not give a flash demo (toy – see conjunction and idempotence) but...
- use TOM to make easy, trustable and modular implementation of calculi.

Conclusion

- I will not give a flash demo (toy – see conjunction and idempotence) but...

- use TOM to make easy, trustable and modular implementation of calculi.
- The three components of the TOM-project make a bridge between the rewriting community and real-life programmers.

Conclusion

- I will not give a flash demo (toy – see conjunction and idempotence) but...

- use TOM to make easy, trustable and modular implementation of calculi.
- The three components of the TOM-project make a bridge between the rewriting community and real-life programmers.
- How to provide a nice way to trace intermediate computation?