



**HAL**  
open science

## ELAN ou la programmation par réécriture

Claude Kirchner

► **To cite this version:**

Claude Kirchner. ELAN ou la programmation par réécriture. FAC 2000, May 2000, Toulouse, France, 96 p. inria-00107865

**HAL Id: inria-00107865**

**<https://inria.hal.science/inria-00107865>**

Submitted on 19 Oct 2006

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# ELAN ou la programmation par réécriture

**Claude Kirchner**

LORIA & INRIA

Nancy

France

Toulouse, Mai 2000

# Rules are ubiquitous in Mathematics, Logic and Computer Science

Production rules

Program transformation rules

Grammar rules

Transition rules

Logic programming rules

Constraint manipulation rules

Inference rules

Type checking rules

Derivative rules

and much more ...

## Some examples

- the Unix mail system: look at the `/etc/sendmail.cf` file!
- tree automaton
- the so called *natural semantics*
- Mathematica
- Reve, Spass, Otter, Saturate
- ...

## Some success

- re-engineering of Cobol programs [ASF+SDF, CWI Amsterdam]
- proof of the Robin's conjecture [New York Times, December 1996]
- ... ... and ELAN

## fibonacci

$$fib(0) \rightarrow 1$$

$$fib(1) \rightarrow 1$$

$$fib(n) \rightarrow fib(n-1) + fib(n-2)$$

$$fib(3) \rightarrow fib(2) + fib(1)$$

$$fib(2) + fib(1) \rightarrow fib(2) + 1$$

$$fib(2) + 1 \rightarrow fib(1) + fib(0) + 1$$

$$fib(1) + fib(0) + 1 \rightarrow 1 + fib(0) + 1$$

...

# /etc/sendmail.cf

##### @(#)nullrelay.m4 8.19 (Berkeley) 5/19/1998 #####

#  
# This configuration applies only to relay-only hosts. They send  
# all mail to a hub without consideration of the address syntax  
# or semantics, except for adding the hub qualification to the  
# addresses.

# This is based on a prototype done by Bryan Costales of ICSI.  
#

#####

#####

#####

##### REWRITING RULES

#####

#####

#####

```
#####
```

```
### Ruleset 3 -- Name Canonicalization ###
```

```
#####
```

```
S3
```

```
# handle null input
```

```
R$@ $@ <@>
```

```
# strip group: syntax (not inside angle brackets!) and trailing semicolon
```

```
R$* $: $1 <@>mark addresses
```

```
R$* < $* > $* <@>$: $1 < $2 > $3 unmark <addr>
```

```
R$* :: $* <@>$: $1 :: $2 unmark node::addr
```

```
R:include: $* <@>$: :include: $1 unmark :include:...
```

```
R$* : $* <@>$: $2 strip colon if marked
```

```
#####  
### Ruleset 4 -- Final Output Post-rewriting ###  
#####
```

S4

```
R$* <@>$@ handle <> and list;;
```

```
# strip trailing dot off before passing to nullclient relay
```

```
R$* @ $+ . $1 @ $2
```



## A simple game

The rules of the game:

●● → ○

○○ → ○

●○ → ●

○● → ●

A starting point:

● ○ ● ○ ● ○ ● ● ● ● ○ ○ ● ○ ○ ● ● ○

Who wins? (i.e. put the last white)

●○ ● ○ ● ○ ● ● ○ ○ ● ○ ○ ● ● ○

●● ○ ● ○ ● ● ○ ○ ● ○ ○ ● ● ○

○ ○ ● ○ ● ● ○ ○ ● ○ ○ ● ●○

○ ○ ● ○ ● ● ○ ○ ● ○ ○○ ● ●

○ ○ ●○ ● ● ○ ○ ● ○ ● ●

○○ ● ● ● ○ ○ ● ○ ● ●

○● ● ● ○ ○ ● ○ ● ●

● ● ●○ ○ ● ○ ● ●

● ● ● ○ ● ○ ●●

● ● ● ○ ●○○

● ● ● ○ ●○

● ● ● ○○ ●

● ● ●●

●●○

●●

○

## Rewrite description of a sorting algorithm

```
sorts NeList List ;    subsorts Nat < NeList < List ;
```

```
operators
```

```
nil : List ;
```

```
@ @ : (List List) List      [associative id: nil] ;
```

```
@ @ : (NeList List) NeList  [associative] ;
```

```
hd @ : (NeList) Nat ;
```

```
tl @ : (NeList) List ;
```

```
sort @ : (List) List ;
```

```
end
```

```
rules for List
```

```
X, Y : Nat ; L L' L'' : List;
```

```
hd (X L) => X ;                tl (X L) => L ;
```

```
sort nil => nil .
```

```
sort (L X L' Y L'') => sort (L Y L' X L'') if Y < X .
```

```
end
```

```
sort (6 5 4 3 2 1) => ...
```

## Alternative Rings

$$0 + x = x$$

$$0 * x = 0$$

$$x * 0 = 0$$

$$i(x) + x = 0$$

$$i(x + y) = i(x) + i(y)$$

$$i(i(x)) = x$$

$$x * (y + z) = (x * y) + (x * z) \quad (x + y) * z = (x * z) + (y * z)$$

$$(x * y) * y = x * (y * y)$$

$$(x * x) * y = x * (x * y)$$

$$i(x) * y = i(x * y)$$

$$x * i(y) = i(x * y)$$

$$i(0) = 0$$

$$(x + y) + z = x + (y + z)$$

$$x + y = y + x$$

## Can we prove by rewriting the Moufang Identities?

$$\begin{aligned}(x * y) * x &= x * (y * x) \\ x * ((y * z) * x) &= (x * (y * z)) * x \\ x * (y * (x * z)) &= ((x * y) * x) * z \\ ((z * x) * y) * x &= z * (x * (y * x)) \\ (x * y) * (z * x) &= (x * (y * z)) * x\end{aligned}$$

(R.Moufang, 1933)

(S.Anantharaman and J.Hsiang, JAR 6, 1990)

## Needham-Schroeder public-key protocol

- proposed in '78 by Needham and Schroeder
- proved insecure in '95 by G. Lowe
- Needham-Schroeder public-key protocol has been already analyzed using several methodologies
  - model-checkers - FDR, Mur $\varphi$
  - theorem proving - NRL
  - rewriting - Maude

## The protocol

The Needham-Schroeder public-key protocol aims to establish a mutual authentication between an initiator and a responder that communicate via an insecure network.

	Initiator	Responder	Net
1. $A \rightarrow B: \{N_A, A\}_{K(B)}$	$A^{SLEEP}$	$B^{SLEEP}$	$\emptyset$
	$A^{WAIT}$	$B^{SLEEP}$	$\{N_A, A\}_{K(B)}$
2. $B \rightarrow A: \{N_A, N_B\}_{K(A)}$	$A^{WAIT}$	$B^{\{N_A, A\}_{K(B)}}$	$\emptyset$
	$A^{WAIT}$	$B^{WAIT}$	$\{N_A, N_B\}_{K(A)}$
3. $A \rightarrow B: \{N_B\}_{K(B)}$	$A^{\{N_A, N_B\}_{K(A)}}$	$B^{WAIT}$	$\emptyset$
	$A^{COMMIT}$	$B^{WAIT}$	$\{N_B\}_{K(B)}$
4. $N_B$ is the session key	$A^{COMMIT}$	$B^{COMMIT}$	$\emptyset$

## Methodology

Detect possible attacks of the protocol by the symbolic execution of all the possible scenarios.

Undeterministic strategies are needed to ease the description of the search space.

More details after introducing ELAN.



## Two main kinds of rules

### Computation rules

$2 + 3$

`fib(33)`

`sort(data-base)`

Compute as fast as possible the unique result

### Deduction rules

Sequent calculus

Resolution method

$\text{Solve}(2x + 4y - 3z - u = 0 \text{ in } \mathbf{N})$

The deduction has to be guided

## Deduction rule application needs to be controlled

Lazy evaluation

Innermost/Outermost reduction

Search plans

Action plans

Tactics

User interaction

► This requires to **search** for a particular derivation corresponding to the desired control.

**rewrite** **rewrite** **rewrite** **rewrite** **rewrite** **rewrite** **rewrite** **rewrite**

Logic Programming, Theorem Proving, Constraint Solving, Expert Systems are instances of the same deduction schema:

Apply rewrite rules on formulas with some strategy,  
until getting specific forms

- ▶ Rewrite blindly: implements computations
- ▶ Rewrite wisely: implements deduction

Poincaré principle [BarendregtBarendsen97]

Deduction modulo [DowekHardinKirchner98]

## What this talk is about

- Give a short presentation of ELAN with an emphasis on control of rewriting
- Present some examples on the use of ELAN
- The rewriting calculus as a semantical foundation for controlled rule based systems

## Presentation of ELAN

ELAN= computation rules + (deduction rules + strategies)

## ELAN's General features

- ELAN has been designed to prototype, experiment and study deduction systems, in particular for constraint solving and theorem proving.
- ELAN is a declarative rewriting language.
- ELAN has two main originalities w.r.t. other algebraic languages:
  - non-deterministic computations
  - a user defined strategy language to control rewriting

## Deductions and computations

- Rules for computations:
  - ▶ unique normal form required
  - ▶ leftmost innermost deterministic strategy fixed
- Rules for deductions:
  - ▶ no confluence nor termination required
  - ▶ application strategy required.
    - non-determinism handled in practice by backtracking
    - strategies are used to express choices
    - strategy may fail

## Example 1: Very simple ...

```
module fib_builtin
import global builtinInt;
end

operators global
    fib(@) : (builtinInt) builtinInt ;
end

rules for builtinInt
    n : builtinInt ;
global
    [] fib(0) => 1 end
    [] fib(1) => 1 end
    [] fib(n) => fib(n - 1) + fib(n - 2) if greater_builtinInt(n,1) end
end
end
```

fib(33) = 5702887      11405773 rewrite steps in 0.695 s      16.411.184 rewrite/s



## Example 2: Polynome derivative in ELAN

```
module poly1      import global int;      sort poly;
operators global
  X              : poly;
  @              : ( int ) poly;
  @ + @          : ( poly poly ) poly  assocRight;
  deriv(@)       : ( poly ) poly;
end
rules for poly
  p1, p2 : poly;  n : int;
global
  [] deriv(X)      => 1                end
  [] deriv(n)      => 0                end
  [] deriv(p1+p2)  => deriv(p1)+deriv(p2)  end
end
```

```
module poly3[Vars]
import global int Vars identifier list[identifier];
sort variable poly;
operators global
  FOR EACH Id:identifier SUCH THAT
    Id:=(listExtract) elem(Vars) : { Id : variable; }
  @      : ( variable ) poly;
  @      : ( int )      poly;
  @ + @   : ( poly poly ) poly  assocRight pri 1 (AC);
  (@ + @) : ( poly poly ) poly  alias @ + @:;
  @ * @   : ( poly poly ) poly  assocRight pri 2 (AC);
  (@ * @) : ( poly poly ) poly  alias @ * @:;
  deriv(@,@) : ( poly variable ) poly;
```

```
end
```

```
... ..
```

## A simple strategy

```

[] P+0          => P          end
[] P*0          => 0          end
[] deriv(x,x)  => 1          end
[] deriv(n,x)  => 0          end
[] deriv(p1+p2,x) => deriv(p1,x) + deriv(p2,x) end

```

```

[factorize] P + n1*p1 + n2*p1      => P + n3*p1
                                     where n3:=()n1+n2 end

```

```

[factorize] P + (p1*p2) + (p1*p3) => P + p1*(p2+p3) end

```

```

[expand] P + (p1+p2)*p3          => P + p2*p3 + p1*p3 end

```

end

strategies for poly

```

[] simplify =>

```

```

    normalize( one(expand) ) ; normalize( one(factorize) )

```

end

## Elementary strategies

Built from a few built-in constructors:

· Sequential composition

$S_1; S_2$

· Iteration

$\text{repeat}^*(S)$  or  $\text{iterate}^*(S)$

· Choice points

$\text{dk}(S_1, \dots, S_n)$  : all results of all  $S_i$

$\text{dc}(S_1, \dots, S_n)$  : all results of one  $S_i$

$\text{first}(S_1, \dots, S_n)$  : all results of the first  $S_i$

· Cut points

$\text{dc one}(S_1, \dots, S_n)$  : one result of one  $S_i$

$\text{first one}(S_1, \dots, S_n)$  : one result of the first  $S_i$

· Failure **fail** and Identity **id**

## Another example: propositional sequent calculus

$$\frac{H, P \vdash Q}{H \vdash \neg P, Q} \text{neg-r}$$

rules for Seq

P, Q, R : Pred; H : Pred; S1, S2 : Seq;

global

[neg-r] H |- ^P : Q => S1

where S1 := (dedstrat) H : P |- Q

end

...

[axio] P : H |- P : R => \$

if neq\_Pred(P, EmptyP)



## Strategies

```
strategies for Seq  
implicit
```

```
  [] SetRules => first one(  
    axio  
    ,negd ,disjd  
    ,impd ,negg ,conjg  
    ,disjg ,conjd ,impg)  
end
```

```
end
```

```
strategies for Seq  
implicit
```

```
  [] dedstrat => first one( Start );  
    repeat*( SetRules )  
end
```

```
end
```

## The resulting proof term

`[dedstrat] (A  $\Rightarrow$  B  $\vdash$   $\neg$ (B)  $\Rightarrow$   $\neg$ (A))`

evaluates to:

```
#infer [#impd] <(A#to B)#vdash(#neg(B)#to#neg(A))>
<#infer [#negd] <(A#to B),#neg(B)#vdash#neg(A)>
<#infer [#negg] <A,(A#to B),#neg(B)#vdash EmptyP>
<#infer [#impg] <A,(A#to B)#vdash B>
<#infer [#axiom] <A,B#vdash B><#mbox<>>&
#infer [#axiom] <A#vdash A,B><#mbox<>>>>>>
end
```



$$\frac{\frac{\frac{}{A, B \vdash B} \textit{Axiom} \quad \frac{}{A \vdash A, B} \textit{Axiom}}{\frac{}{A, (A \rightarrow B) \vdash B} \rightarrow_G}}{\frac{}{A, (A \rightarrow B), \neg(B) \vdash} \neg_G} \neg_D}{\frac{}{(A \rightarrow B), \neg(B) \vdash \neg(A)} \rightarrow_D} \rightarrow_D}{(A \rightarrow B) \vdash (\neg(B) \rightarrow \neg(A))}$$

## To summarize: Rules in ELAN

Rules are possibly labeled conditional rewrite rules with local variable affectations

$[\text{lab}] : l(x) \Rightarrow r(y) \quad \text{if } v \quad \text{where } y := [S]u(x)$

- $\text{lab}$  is the rule label,
- $l$  and  $r$  the respective left and right-hand sides,
- $v$  the condition and
- $y := [S]u$  a local affectation, assigning to the local variable  $y$  one of the results of the strategy  $S$  applied to the term  $u$ .

## Computation and Deduction in ELAN

**Computations** are described with unlabeled rules

**Deductions** are described by labeled rules

**Strategies** for deductions are themselves described using labeled or unlabeled rewrite rules.

## The Computational System Tower

...

...

Meta Meta Strategies

Meta Strategies

Strategies

Terms

## Rewriting with strategies IS now a realistic programming paradigm

On typical applications, ELAN applies per second:

- 10 millions of non-labeled rules
- 1 million of labeled rules
- 100 000 of AC rules

On real size applications:

- Jobs shop 10x10: 2 billions rewrites applied in 2 hours

## Some Applications developed in ELAN

- Theorem provers (KBs, Spike, B)
- Automata computations
- CP on finite domains
- Unifications
- Combination of deduction systems
- ...

## N-queens

$queens(n, size) \rightarrow x.ql$  if  $n > 0$

where  $ql := (queens\_strat)queens(n - 1, size)$

where  $x := (iterate * (dc(range\_rule)))size$

if  $noattack(1, x, ql)$

...

	Interpreter	Compiler	Compiler
Query	queens(8)	queens(8)	queens(11)
Applied rules	128,949	128,949	19,286,638
Speed (inf/sec)	6,570	2,387,000	2,743,000

- ▶ **Note:** local affectations and non-deterministic strategies
- ▶ **Note:** efficient management of non-deterministic strategies
- ▶ **Speedup:** 360

## Completion à la Knuth Bendix

- 225 nonamed rules
- 105 named rules
- 43 strategies (8 dont know, 58 dont care, 6 iterate, 14 repeat)

	Interpreter	Compiler	Compiler
Query	group	group	p8
Applied rules	181,752	181,752	21,607,279
Speed (inf/sec)	1,452	826,145	919,458

- **Note:** conditional rules, local affectations and non-deterministic strategies,...
- **Speedup:** 135



## Nat10

Arithmetic rules introduced by Marché et al

- 56 rules rooted by the AC symbol +
- 11 rules rooted by the AC symbol \*
- 82 syntactic rules

	Interpreter	Compiler	Compiler
Query	Fib(16)	Fib(16)	Fib(25)
Applied rules	15,384	15,384	1,171,043
Speed (inf/sec)	44	49,625	53,717

# Encoding and analysing the Needham-Schroeder public-key protocol in ELAN

Thanks to Horatiu Cirstea

- user-definable number of initiators and responders
- user-definable size of the network
- the strategies guiding the rewrite rules describe a form of model-checking in which all the possible behaviors are explored

## Data structures

- The agent

```
@ + @ + @ : ( AgentId SWC Nonce ) Agent;  
@ : ( Agent ) listAgent;  
@ || @ : ( listAgent listAgent ) listAgent (AC);
```

- The messages

```
@-->@:@[,@,@] : ( AgentId AgentId Key Nonce Nonce Address ) message;  
@ : ( message ) network;  
@ & @ : ( network network ) network (AC);
```

- The intruders

```
@ # @ # @ : ( AgentId listNonce network ) intruder;
```

- The global state

```
@ <> @ <> @ <> @ : ( listAgent listAgent intruder network ) state;
```

## Rewrite rules for the agents

- initiator starts the communication with a responder

```
[initiator-1]
  x+SLEEP+resp  || E <> D <> w#1#11 <> 1s          =>
  x+WAIT+N(x,y) || E <> D <> w#1#11 <> x-->y:K(y) [N(x,y),DN,A(x)] & 1s
                where (Agent)y+std+init :=(extAgent) elemIA(D || w+SLEEP+DN)
end
```

- responder reads the message and sends the acknowledgement

```
[responder-1]
  E <> y+SLEEP+init  || D <> I <> w-->y:K(y) [N(n1,n3),N(n2,n4),A(z)] & 1s =>
  E <> y+WAIT+N(y,z) || D <> I <> y-->z:K(z) [N(n1,n3),N(y,z),A(y)] & 1s
end
```

## Rewrite rules for the agents

- initiator receives the acknowledgement and checks its validity

```
[initiator-2]
x+WAIT+N(x,v) || E <> D <> I <> w-->x:K(x) [N(n1,n3),N(n2,n4),A(z)] & ls =>
S
  choose
    try
      if x==n1 and v==n3
        where S:=() x+COMMIT+N(x,v) || E <> D <> I <> x-->v:K(v) [N(n2,n4),DN,DA] & ls
      try
        if x!=n1 or v!=n3
          where S:=() ERROR
      end
    end
  end
```

## Rewrite rules for the intruder

- the intruder intercepts all the messages in the network but the messages generated by himself and stores or decrypts them.

```
[intruder-1]
```

```
  E <> D <> w#l#ll <> z-->x:K(w)[N(n1,n3),N(n2,n4),A(v)] & ls =>
```

```
  E <> D <> w#N(n1,n3) | N(n2,n4) | l#ll <> ls
```

```
      if w!=z
```

```
end
```

- the nonces obtained previously by the intruder are used in order to generate fake messages that are sent to all the agents.

```
[intruder-4]
```

```
  E <> D <> w # resp | l # ll <> ls                                     =>
```

```
  E <> D <> w # l # ll <> w-->y:K(y)[resp,DN,A(xadd)] & ls
```

```
      where (Agent)y+std+dn :=(extAgent) elemIA(D || E)
```

```
      where (Agent)xadd+std1+dn1 :=(extAgent) elemIA(D || E)
```

```
end
```

## The invariants

- authenticity of the responder: if an initiator  $x$  committed with a responder  $y$ , then  $y$  has really been involved in the protocol.

```
[attack-1] x+COMMIT+N(x,y) || E <> D <> I <> ls =>  
  ATTACK  
    if y!=i  
    if not(existAgent(y+WAIT+N(y,x),D)) and  
      not(existAgent(y+COMMIT+N(y,x),D))  
end
```

- authenticity of the initiator: if a responder  $y$  committed with an initiator  $x$  then the initiator have committed as well with  $y$ .

```
[attack-2] E <> y+COMMIT+N(y,x) || D <> I <> ls =>  
  ATTACK  
    if x!=i  
    if not(existAgent(x+COMMIT+N(x,y),E))  
end
```

## The strategy

We apply repeatedly all the rewrite rules in any order and in all the possible ways until one of the attack rules can be applied.

```
[attStrat => repeat*(
    dk(
        attack-1, attack-2,
        intruder-1, intruder-2, intruder-3, intruder-4,
        initiator-1, initiator-2, responder-1, responder-2)
    );
  attackFound
end
```

where

```
[attackFound] ATTACK => ATTACK end
```



## The attack

- I.1.  $A \rightarrow I$  :  $\{N_A, A\}_{K(I)}$
- II.1.  $I(A) \rightarrow B$  :  $\{N_A, A\}_{K(B)}$
- II.2.  $B \rightarrow I(A)$  :  $\{N_A, N_B\}_{K(A)}$
- I.2.  $I \rightarrow A$  :  $\{N_A, N_B\}_{K(A)}$
- I.3.  $A \rightarrow I$  :  $\{N_B\}_{K(I)}$
- II.3.  $I(A) \rightarrow B$  :  $\{N_B\}_{K(B)}$

## A second strategy

```

[]attStrat => repeat*(
    dk(
        attack-1, attack-2,
        initiator-1, initiator-2, responder-1, responder-2,
        intruder-1, intruder-2, intruder-3, intruder-4)
    );
attackFound
end

```

- I.1.  $A \rightarrow I$  :  $\{N_A, A\}_{K(I)}$
- II.1.  $I(A) \rightarrow B$  :  $\{N_A, A\}_{K(B)}$
- I+II.2.  $B \rightarrow A$  :  $\{N_A, N_B\}_{K(A)}$
- I.3.  $A \rightarrow I$  :  $\{N_B\}_{K(I)}$
- II.3.  $I(A) \rightarrow B$  :  $\{N_B\}_{K(B)}$

## The corrected protocol

1.  $A \rightarrow B: \{N_A, A\}_{K(B)}$
2.  $B \rightarrow A: \{N_A, N_B, B\}_{K(A)}$
3.  $A \rightarrow B: \{N_B\}_{K(B)}$

*Modified rule: initiator-2*

[initiator-2] x+WAIT+N(x,v) || E <> D <> I <> w-->x:K(x)[N(n1,n3),N(n2,n4),A(z)] & ls =>

```

S
choose
  try
    if v==z          // expected responder
    if x==n1 and v==n3
    where S:=() x+COMMIT+N(x,v) || E <> D <> I <> x-->v:K(v)[N(n2,n4),DN,DA] & ls
  try
    if v!=z          // not expected responder
    if x!=n1 or v!=n3
    where S:=() ERROR
end
end

```

## Optimizations of the current implementation

- each message is given a type according to its relevant fields (nonce and address),
- initiator and responder listen only to messages coming from the intruder,
- the intruder sends fake messages only to those agents that are able to handle it

## Comparison with Mur $\varphi$

ini	res	net	Mur $\varphi$	ELAN simple	ELAN(Mur $\varphi$ like)	ELAN optimized
1	1	1	2578 rules 0.56s	7442 rules 0.167s	4308 rules 0.125s	711 rules 0.064s
1	1	2	136273 rules 18.40s	453514 rules 7.007s	333552 rules 4.980s	6700 rules 0.162s
2	1	1	25701 rules 6.81s	575101 rules 8.571s	257087 rules 3.946s	26011 rules 0.483s
2	2	1	557430 rules 303.46s	118214389 rules 1658.296s	22985807 rules 333.096s	753785 rules 12.392s

## Concluding remarks on this example

- computational systems can be used as a logical framework for representing the Needham-Schroeder public-key protocol,
- this approach can be easily extended to other authentication protocols (TMN),
- the strategy used for guiding the application of the rewrite rules is important when an attack on the protocol exists,
- the behavior of intruders can be easily modified by changing the corresponding rules,
- the state space is finite and thus, some other methods should be used in order to show that properties proved for the finite model can be lifted to an unbounded model,

## How does ELAN compilation work?

- leftmost-innermost normalization
  - many-to-one matching
  - re-using left-hand-side parts
  - shared terms allowed
  - particular choice point management
- What about the semantics?

## The rewriting calculus as a semantics of ELAN



## Towards a new calculus

That gives a first class status to:

- ▶ rewrite rules

and

- ▶ strategies

## The main ideas (1)

apply a rule at the top level of a term

$$[l \rightarrow r](t)$$

## The main ideas (2)

The application operator  $[ ]( )$  returns results that should be handled explicitly.

For example, if  $+$  is commutative, what is the result of the application of the rule

$$x + y \rightarrow x$$

on

$$a + b$$

should it be  $a$ ?

or  $b$ ?

or  $a \diamond b$ ?

## The main ideas (3)

— rule application

— results object

should be **explicit** objects of the calculus

## The calculus ingredients

Five components:

- 1** The syntax of terms and substitutions,
- 2** The description of the substitution application on terms,
- 3** The matching algorithm used to bind variables to their actual arguments,
- 4** The evaluation rules describing the way the calculus operates locally.
- 5** The strategy describing how the evaluation rules operate globally.

# 1 $\rho$ -Calculus Syntax

- elements in  $\mathcal{X}$  (variables) and in  $\mathcal{T}(\mathcal{F}, \mathcal{X})$  are  $\rho$ -terms,

If  $t, u, t_1, \dots, t_m$  are  $\rho$ -terms and  $f \in \mathcal{F}_m$  then the following are  $\rho$ -terms:

- $f(t_1, \dots, t_m)$
- $\{t_1, \dots, t_m\}$  (if  $m = 0$  we have the  $\rho$ -term  $\emptyset$ ),
- $[t](u)$  (application of the  $\rho$ -term  $t$  to the  $\rho$ -term  $u$ ),
- $t \rightarrow u$  (rewrite rule).

$$t ::= x \mid \{t, \dots, t\} \mid f(t, \dots, t) \mid [t](t) \mid t \rightarrow t \quad (x \in \mathcal{X}, f \in \mathcal{F})$$

## Examples of $\rho$ -terms

- $f(x, y)$  a first order term
- $f(x, x) \rightarrow x$  a “standard” rewrite rule
- $[a \rightarrow b](a)$  the application of the rule  $a \rightarrow b$  to the term  $a$
- $[f(x, y) \rightarrow g(x, y)](f(a, b))$  a classical rule application
- $[x \rightarrow x](a)$  the result is  $\{a\}$
- $[x \rightarrow y](a)$  the result is  $\{y\}$
- $[x \rightarrow (x \rightarrow x)]([x \rightarrow y](a))$  similar to the  $\lambda$ -term  $((\lambda x.\lambda x.x) ((\lambda x.y) a))$
- $[x \rightarrow [x](x)](x \rightarrow [x](x))$  the well-known  $(\Delta \Delta)$   $\lambda$ -term
- $[(a \rightarrow b) \rightarrow c](a)$  a more complicated  $\rho$ -term
- $[\{a \rightarrow b, a \rightarrow c\}](a)$  apply several rules

## Handling results

Handling results via the set datastructure:

- has consequences on the calculus evaluation rules and their properties
- could be done in a different ways using e.g. list, multisets, ...



## 2 Substitution application on $\rho$ -terms

Takes care of variable capture.

Grafting	Substitution
$x \mapsto u$	$x/u$
$\{x \mapsto a + y\}(y \rightarrow y + x) = y \rightarrow y + (a + y)$	$\{x/a + y\}(y \rightarrow y + x) = y' \rightarrow y' + (a + y)$

## 4 $\rho$ -Calculus: Evaluation Rules

The main rule in general:

$$\mathbf{Fire} \quad [l \rightarrow r](t) \quad \mapsto \quad r \langle\langle \mathit{Solution}(l \ll_T^? t) \rangle\rangle$$

For example

$$\mathbf{Fire} \quad [f(x) \rightarrow x](f(a)) \quad \mapsto \quad x \langle\langle \mathit{Solution}(f(x) \ll_T^? f(a)) \rangle\rangle$$

In case of syntactic matching:

$$\mathbf{Fire} \quad [l \rightarrow r](\sigma(l)) \quad \mapsto \quad \{\sigma(r)\}$$

$$\mathbf{Fire} \quad [l \rightarrow r](t) \quad \mapsto \quad \emptyset$$

if  $l$  does not match  $t$

## Congruence rules on operators

<b>Congruence</b>	$[f(u_1, \dots, u_n)](f(v_1, \dots, v_n))$	$\mapsto$	$\{f([u_1](v_1), \dots, [u_n](v_n))\}$
<b>Congruence_fail</b>	$[f(u_1, \dots, u_n)](g(v_1, \dots, v_m))$	$\mapsto$	$\emptyset$

Equivalent to the reduction of

$$f(x_1, \dots, x_n) \rightarrow f([u_1](x_1), \dots, [u_n](x_n))(t)$$

## Congruence rules on sets

<b>Distrib</b>	$[\{u_1, \dots, u_n\}](v)$	$\mapsto$	$\{[u_1](v), \dots, [u_n](v)\}$
<b>Batch</b>	$[v](\{u_1, \dots, u_n\})$	$\mapsto$	$\{[v](u_1), \dots, [v](u_n)\}$
<b>SwitchL</b>	$\{u_1, \dots, u_n\} \rightarrow v$	$\mapsto$	$\{u_1 \rightarrow v, \dots, u_n \rightarrow v\}$
<b>SwitchR</b>	$u \rightarrow \{v_1, \dots, v_n\}$	$\mapsto$	$\{u \rightarrow v_1, \dots, u \rightarrow v_n\}$
<b>OpOnSet</b>	$f(v_1, \dots, \{u_1, \dots, u_m\}, \dots, v_n)$	$\mapsto$	$\{f(v_1, \dots, u_1, \dots, v_n),$ $\dots,$ $f(v_1, \dots, u_m, \dots, v_m)\}$

## Handling sets of sets

**Flat**  $\{u_1, \dots, \{v_1, \dots, v_n\}, \dots, u_m\} \rightsquigarrow \{u_1, \dots, v_1, \dots, v_n, \dots, u_m\}$

## Applying substitutions

Meta-rules in this version

$$\begin{array}{ll} \mathbf{Propagate} & r\langle\langle\{s_1, \dots, s_n\}\rangle\rangle \rightsquigarrow \{s_1(r), \dots, s_n(r)\} \\ \mathbf{Clash} & r\langle\langle\emptyset\rangle\rangle \rightsquigarrow \emptyset \end{array}$$

Could be made explicit:  $\rho\sigma$ -calculus

## Remarks

$\rho$ -calculus describes rewrite rules application **at the top level of a term**

$$[f(x) \rightarrow g(x)](f(b)) \xrightarrow[\text{Fire}]{} \{g(b)\}$$

out

$$[x + 0 \rightarrow x](f(3 + 0)) \xrightarrow[\text{Fire}]{} \emptyset$$

and

$$[f(x + 0 \rightarrow x)](f(3 + 0)) \xrightarrow[\text{Congruence}]{} f([x + 0 \rightarrow x](3 + 0)) \xrightarrow[\text{Fire}]{} f(\{3\})$$

The **evaluation rules** expressing the behavior of  $\rho$ -calculus are applied **\*everywhere\*** in the term (e.g. like  $\beta$ -reduction).

### 3 Matching

In general

For a theory  $T$  a  $T$ -*match-equation* is a formula of the form  $t \ll_T^? t'$ , where  $t$  and  $t'$  are  $\rho$ -terms.

A substitution  $\sigma$  is a  $T$ -solution of  $t \ll_T^? t'$  when

$$\sigma(t) =_T t'$$

$T$ -matching is in general **undecidable**.

**Decidable** cases:

- higher-order pattern matching [Miller-89]
- higher-order matching (up to the fourth order [PadovaniThese-96])
- many first-order equational theories and their combinations



## Solutions set

We define the function *Solution* on a  $T$ -matching system  $\mathcal{S}$  as returning the set of:

- all  *$T$ -matches* of  $\mathcal{S}$  when  $\mathcal{S}$  is not trivial
- $\{Id\}$  (the identity substitution) when  $\mathcal{S}$  is trivial
- $\emptyset$  when the matching algorithm fails.

## Specificities of the $\rho_T$ -calculus

- calculus of explicit rule application
- get the “matching power”
- explicit handling of result sets,
- one of the main parameter of the calculus is the matching algorithm,
- allows the full control of the rewrite rule application
- allows a uniform combination of higher-order and first-order features,
- allows expressing strategies (e.g. search strategies)
- the  $\rho$ -calculus is quite different from the rewrite relation generated by a trs.

## The $\rho_\emptyset$ -calculus

An instance of  $\rho_T$ -calculus where:

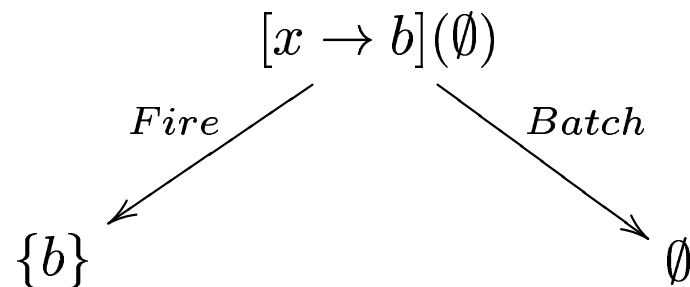
- $T = \emptyset$
- the left members of matching equations are composed only of first-order terms (i.e. not containing arrows or applications).

From now on we only consider  $\rho_\emptyset$ -calculus.

No difficulty to extend to the case of first-order equational matching (e.g. AC-matching)

## Technical specificities

- A rule application always fires ... but the result may be the empty set (when the rule does not match an under-evaluated term),
- Thus the wild  $\rho_\emptyset$ -calculus is trivially non-confluent:



## ConfStratStrict

The term  $[l \rightarrow r](t)$  is reduced using the evaluation rule **Fire** only if  $t \in \mathcal{T}(\mathcal{F})$ .

## ConfStrat

The term  $[l \rightarrow r](t)$  is reduced using the evaluation rule **Fire** only if:

- the term  $l$  is linear,
- the term  $l$  weakly subsumes the term  $t$  ( $\mathcal{FPos}(l) \subseteq \mathcal{FPos}(t)$ ),
- the term  $t$  contains no empty set and no set with more than one element,
- the term  $t$  contains no sub-term of the form  $[u](v)$  where  $u$  is not a rewriting rule,
- for all sub-term  $[u \rightarrow w](v)$  of  $t$ ,  $u$  subsumes  $v$ ,

**or**

- $t \in \mathcal{T}(\mathcal{F})$  is a closed first order term.

## ConfStratOper

The term  $[l \rightarrow r](t)$  is reduced using the evaluation rule **Fire** only if  $t \in \mathcal{T}(\mathcal{F})$  or:

- the term  $l$  is linear and
- the term  $l$  weakly subsumes the term  $t$

**and**

- the rewrite rule  $l \rightarrow r$  is conservative ( $FV(l) \subseteq FV(r)$ )

**or**

- the term  $t$  contains no empty set
- the term  $t$  contains no sub-term of the form  $[u](v)$  where  $u$  is not a rewriting rule,
- for all sub-term  $[u \rightarrow w](v)$  of  $t$ ,  $u$  subsumes  $v$ ,

**and**

- the rewrite rule  $l \rightarrow r$  is right linear

**or**

- the term  $t$  contains no set with more than one element.

When using one of the strategies *ConfStratStrict*, *ConfStrat* or *ConfStratOper*, the  $\rho_\emptyset$ -calculus is confluent.

## About expressiveness of the $\rho$ -calculus



## Encoding the $\lambda$ -calculus

$t ::= x \mid \{t\} \mid f(t, \dots, t) \mid [t](t) \mid x \rightarrow t$

**FirePropagate**  $[x \rightarrow r](t) \rightsquigarrow \{\{x/t\}r\}$

$\rho(\lambda x.t) = x \rightarrow t$

$\rho((s t)) = [s](t)$

Given  $t$  and  $t'$  two  $\lambda$ -terms. Then,  $t \longrightarrow_{\beta} t'$  iff  $\varphi(t) \longrightarrow_{\rho} \{\varphi(t')\}$ .

**Proof:** For any  $\lambda$ -term  $(\lambda x.t u)$  that reduces to  $\{x/u\}t$  the corresponding  $\rho$ -reduction is  $[x \rightarrow t](u) \longrightarrow_{\text{FirePropagate}^*} \{\{x/u\}t\}$ .  $\square$

## Encoding rewriting

$$t ::= x \mid \{t\} \mid f(t, \dots, t) \mid [t](l) \mid l \rightarrow l$$

Given  $t$  and  $t'$  two terms in  $\mathcal{T}(\mathcal{F}, \mathcal{X})$  and  $\mathcal{R}$  a first order term rewrite system.

If  $t \xrightarrow{*}_{\mathcal{R}} t'$  then there exists a  $\rho$ -term  $u$  constructed using the rules in  $\mathcal{R}$  such that  $[u](t) \xrightarrow{*}_{\rho} \{t'\}$ .

**Proof:** Let us consider a rewrite rule  $(l \rightarrow r)$  that applies on the term  $t$  at position  $p$  and thus,  $t[\theta l]_p \xrightarrow{\mathcal{R}} t[\theta r]_p = t'$ .

The  $\rho$ -term  $u$  to be applied in the  $\rho$ -calculus is  $t[l \rightarrow r]_p$  and we get:

$$[t[l \rightarrow r]_p](t[\theta l]_p) \xrightarrow{*}_{\rho} \{t[\theta r]_p\}$$

□

## (Normalization) Strategies

Given a rewrite theory  $\mathcal{R}$  does it exist a  $\rho$ -term  $\xi_{\mathcal{R}}$  such that for any term  $u$ , if  $u$  reduces to the term  $v$  in the rewrite theory  $\mathcal{R}$  then  $[\xi_{\mathcal{R}}](u)$   $\rho$ -reduces to  $\{\dots, v, \dots\}$ ?

Given a rewrite theory  $\mathcal{R}$  does it exist a  $\rho$ -term  $\xi_{\mathcal{R}}$  such that for any term  $u$ , if  $u$  normalizes to the term  $v$  in the rewrite theory  $\mathcal{R}$  then  $[\xi_{\mathcal{R}}](u)$   $\rho$ -normalizes to  $\{\dots, v, \dots\}$ ?

## Extending the calculus: with syntactic sugar

$$t ::= id \mid fail \mid t; t \mid dk(t, t)$$

Evaluation rules

**Identity**      $id$       $\mapsto$       $x \rightarrow x$

**Fail**      $fail$       $\mapsto$       $x \rightarrow \emptyset$

**Compose**      $[s_1; s_2](t)$       $\mapsto$       $[s_2]([s_1](t))$

**DK**      $[dk(s_1, s_2)](t)$       $\mapsto$       $\{[s_1](t), [s_2](t)\}$

## The *first* operator

**First**             $[first(s_1, \dots, s_n)](t) \quad \mapsto \quad \langle [s_1](t), \dots, [s_n](t) \rangle$

**First\_fail**      $\langle \emptyset, t_1, \dots, t_n \rangle \quad \mapsto \quad \langle t_1, \dots, t_n \rangle$

**First\_success**  $\langle t, t_1, \dots, t_n \rangle \quad \mapsto \quad \{t\}$

*if  $t \neq \emptyset$  is closed and contains no redex*

**First\_single**    $\langle \rangle \quad \mapsto \quad \{\}$

## Term traversal

<b>Tseq</b>	$[\Phi(r)](f(u_1, \dots, u_n))$	$\mapsto$	$\langle \{f([r](u_1), \dots, u_n)\}, \dots, \{f(u_1, \dots, [r](u_n))\} \rangle$
<b>Tseq_ct</b>	$[\Phi(r)](c)$	$\mapsto$	$\emptyset$
<b>Tpar</b>	$[\Psi(r)](f(u_1, \dots, u_n))$	$\mapsto$	$\{f([r](u_1), \dots, [r](u_n))\}$
<b>Tpar_ct</b>	$[\Psi(r)](c)$	$\mapsto$	$\{c\}$

## Fix point operators

$$\Theta = [A](A) \quad \text{with} \quad A = x \rightarrow (y \rightarrow [y]([x](x))(y)))$$

Thus

$$[\Theta](G) \xrightarrow{*}_{\rho} \{[G]([\Theta](G))\}$$

## Bottom-up application

$$Once_{bu}(r) \equiv [\Theta](H_{bu}(r))$$

where

$$H_{bu}(r) \equiv f \rightarrow (x \rightarrow [first(\Phi(f), r)](x))$$

**Ex:**  $[Once_{bu}(a \rightarrow b)](f(a, g(a))) \Longrightarrow_{\rho} \{f(b, g(a))\}$

## Repeat operator

$$\mathit{repeat}^*(r) \equiv [\Theta](J(r))$$

where

$$J(r) \equiv f \rightarrow (x \rightarrow [\mathit{first}(r; f, \mathit{id})](x))$$

## Normalization strategies

$$\mathit{im}(r) \equiv \mathit{repeat}^*(\mathit{Once}_{bu}(r))$$

$$\mathit{om}(r) \equiv \mathit{repeat}^*(\mathit{Once}_{td}(r)).$$



## Encoding Conditional Rewriting

The conditional rewrite rule:

$$l \rightarrow r \text{ if } c$$

is simply encoded by

$$l \rightarrow [\{\mathbf{T} \rightarrow r, \mathbf{F} \rightarrow \emptyset\}]([lmim](c))$$

Or even simpler

$$l \rightarrow [\mathbf{T} \rightarrow r]([lmim](c))$$

## ELAN's semantics

Using rewriting logic [Meseguer 92]

Using  $\rho$ -calculus

## The ELAN basic object: rewrite rules with where

Rules are labelled conditional rewrite rules with local variable affectations

```
[lab]     $l \Rightarrow r(x, y)$   
         where  $x := [S_1]u_1$   
         if  $c_1(x)$   
         where  $y := [S_2]u_2$   
         if  $c_2(x, y)$ 
```

- `lab` is the rule label,
- $l$  and  $r$  are the respective left and right-hand sides,
- $c_1, c_2$  are the conditions and
- $z := [S]u$  are local affectations, assigning to the local variable  $z$  one of the result of the strategy  $S$  applied to the term  $u$ .

## Example of ELAN's rule

$$\begin{aligned}
 [\text{deriveSum}] \quad p_1 + p_2 &\Rightarrow p'_1 + p'_2 \\
 &\text{where } p'_1 := (\text{derive})p_1 \\
 &\text{where } p'_2 := (\text{derive})p_2
 \end{aligned}$$

can be represented by the following two  $\rho$ -terms

$$p_1 + p_2 \rightarrow [\text{derive}](p_1) + [\text{derive}](p_2)$$

$$p_1 + p_2 \rightarrow [p'_1 \rightarrow [p'_2 \rightarrow p'_1 + p'_2]]([\text{derive}](p_2))([\text{derive}](p_1))$$

## An expression of ELAN's rules in $\rho$ -calculus

**[lab]**      $l \Rightarrow r(x)$   
              **where**  $x := [S_1]u_1$   
              **if**  $c_1(x)$

The rule is expressed as the  $\rho$ -term:

$$l \rightarrow [x \rightarrow [True \rightarrow r(x)]([lmim](c_1(x)))]([S_1](u_1))$$

## With two where if

```

[lab]   l ⇒ r(x)
        where x := [S1]u1
        if c1(x)
        where y := [S2]u2
        if c2(x, y)
  
```

The rule is expressed as the  $\rho$ -term:

$$\begin{aligned}
 l \rightarrow & [x \rightarrow [ \textit{True} \rightarrow [ \ y \rightarrow [\textit{True} \rightarrow r(x, y)]([\textit{lmim}](c_2(x, y))) \\
 & \qquad \qquad \qquad ]([S_2](u_2)) \\
 & \qquad \qquad \qquad ]([\textit{lmim}](c_1(x))) \\
 & ]([S_1](u_1))
 \end{aligned}$$

## To sum-up

- ▶ Rule based programming a la ELAN allows to integrate the functional, transitional and nondeterministic programming paradigms
- ▶ ELAN makes exactly the *deduction modulo* concept at work
- ▶ It is now practical to use thanks to the compilation techniques developped (Moreau, Vittek)
- ▶ A relatively large number of programs have been developped in ELAN (Needham-Schroeder, CSP, TP) but also in other rule based programming environments (ASF+SDF, Maude, OBJ, Stratego, ...)
- The programming methodology is still to be better understood
- Connection with other systems at work (COQ-ELAN, CoFI, ASF+SDF parsing and editing advanced technology)

- ▶ The works on ELAN's semantics give rise to a new appealing calculus
- ▶  $\rho_T$ -calculus is a simple, elegant and powerful new calculus
  - explicit rule application (and thus “matching power”)
  - explicit handling of result sets
  - parameterized by a matching theory  $T$
- ▶ It makes a clear distinction between the rewrite *relation* and the rewrite *calculus*
- ▶ It allows uniform combination of first and higher-order computations
- ▶  $\rho$ -calculus gives a simple semantics to ELAN programs
- ▶ The simply typed  $\rho$ -calculus
  - Models of the  $\rho$ -calculus
  - The  $\rho$ -typed  $\rho$ -calculus



More on The logo for ELAN, featuring the word 'elan' in a stylized, lowercase, blue font. The letter 'e' is green and has a white arrow pointing to the right, passing through the center of the 'e' and extending towards the right edge of the image.

<http://www.loria.fr/ELAN>

Current release: V3.4

The ELAN team:

Peter Borovanský, Horatiu Cirstea, Hubert Dubois,  
Claude Kirchner, H el ene Kirchner, Pierre-Etienne Moreau,  
Huy Nguyen, Christophe Ringeissen, Marian Vittek.