A Guarded Action Language to express system semantics

Yann Thierry-Mieg

Joint work with B. Berard, M.Colange, F. Kordon, D. Poitrenaud, S. Baarir

Sept. 2013 – Univ. Mohamed V, Rabat

Journée Vérification et réseaux de capteurs Sans Fil

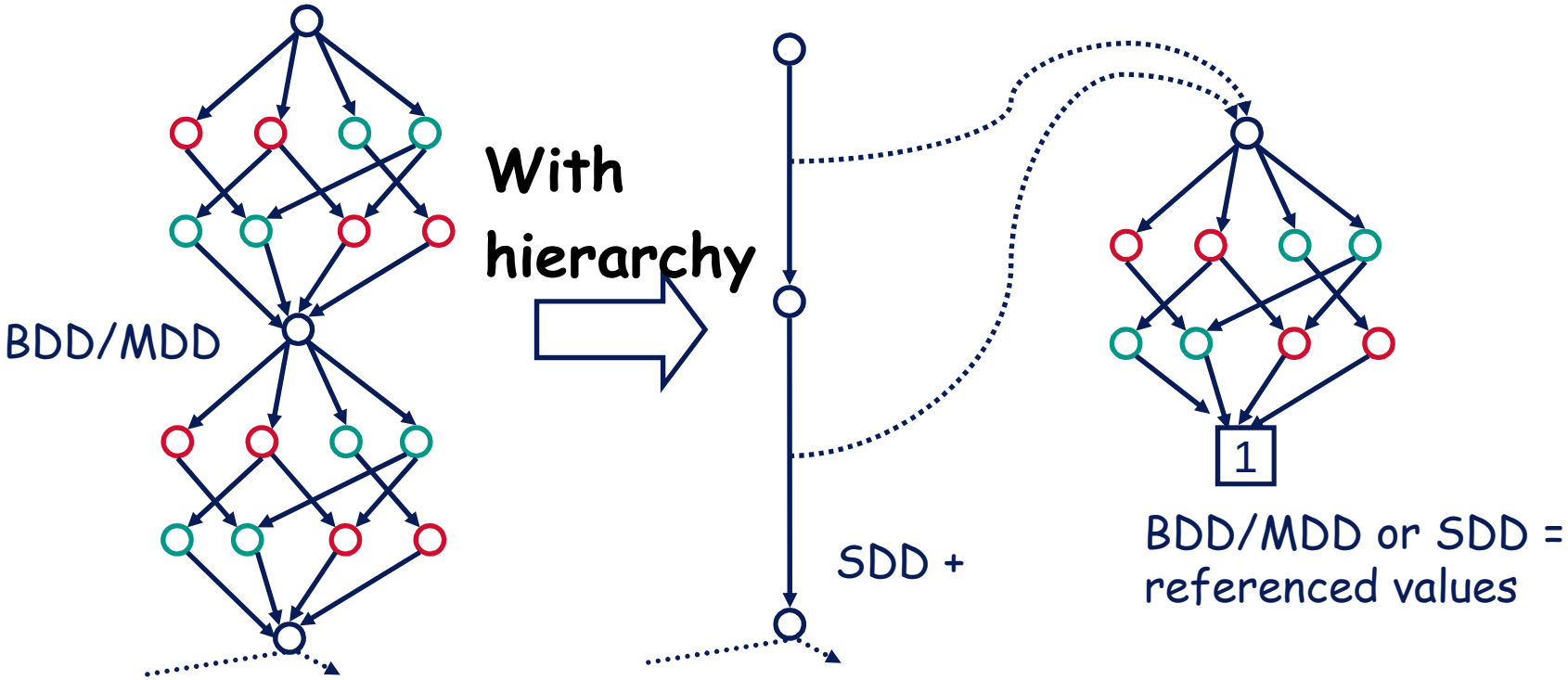
Symbolic Model-checking

- An effective technology for Model-Checking
- Binary Decision diagrams (BDD Bryant'85)
  - Based on Boolean logic
  - Originally designed for hardware systems
- Many BDD extensions proposed:
  - Integer valued : MDD, DDD
  - Multi-Terminal : ADD, MTBDD
- Hierarchical Set Decision Diagrams (SDD)
  - Direct encoding of a structured specification
  - High-level transition encoding with inductive homomorphisms

Hierarchical Set Decision Diagram (SDD) [FORTE'05]

- BDD : booleans, MDD : integers…
- <u>Idea : hierarchy</u>
  - Label the arcs with a SET = Set Decision Diagram

- Increases sharing
  - Memory gain
  - Time gain
    - *cache*
    - *traversals*



SDD offer Automatic Saturation [ICATPN'08, TACAS'09]

- Symbolic model-checking based on transitive closures over the transition relation
  - Dominates overall complexity
- [BCMDH'92] based on <u>BFS iterations</u>
- [Roig'95] <u>Chaining</u> may converge faster, not strict BFS
- [CLS'01-CMS'03] <u>Saturation</u> is empirically 1 to 3 orders of magnitude better, adapt firing order to DD structure

[SDD and Automatic Saturation – ICATPN'08]
Defines rewriting rules to <u>automatically and transparently</u> enable saturation in SDD

Symbolic analysis of expressions

- Symbolic data structures: BDD, MDD
  - k boolean variables => 2^k
- Encode transitions with sets : $2^k \times 2^k$
- Use the support (only k' vars) of transitions
  - Build clusters of transitions
- Reorder evaluations in fixpoint computation (chaining, saturation)
- But :
  - Exponential worst case complexity when k' grows
  - In general, necessary to invoke an explicit solver for each new state in $2^{k'}$

DDD support high level transition relations (NEW CAV'2013)

- Computing the support
  - A[x + y] => pessimistic assumptions
  - x=x+1; y=y+1 => Atomic sequence of updates produce artificially large support
- What if we could compute this on the fly ?
  - Carry the expression in a dedicated operation
  - Traverse a state -> path
  - Resolve variables as they are encountered
  - Drop pessimistic assumptions ASAP
- But we must still reason with sets !

An equivalence relation ?

- Partial expression evaluation
  - f = a + b
  - States : s1:(a=1, b=0) s2:(a=0,b=1)
  - If both a and b are known : f(s1)=f(s2), s1 and s2 are equivalent
  - If only a is known, f(s1)=1+b f(s2)=0+b s1 and s2 are NOT equivalent
- Algorithm discovers variable values and builds equivalence classes on the fly

Evaluating expressions over DD

Algorithm 1: EquivSplit(ϕ,V,i)

Input: ϕ an expression that does not depend on $x_1,\ldots,x_{i-1}$
Input: V a finite set of valuations
Input: i an integer between 1 and $|X|$
Output: a set of pairs $\{(\phi_1,c_1),\ldots,($
classes of $\sim_{\phi}^{\{x_i,\ldots x_n\}}$ over V,
any $\mu \in c_j$.

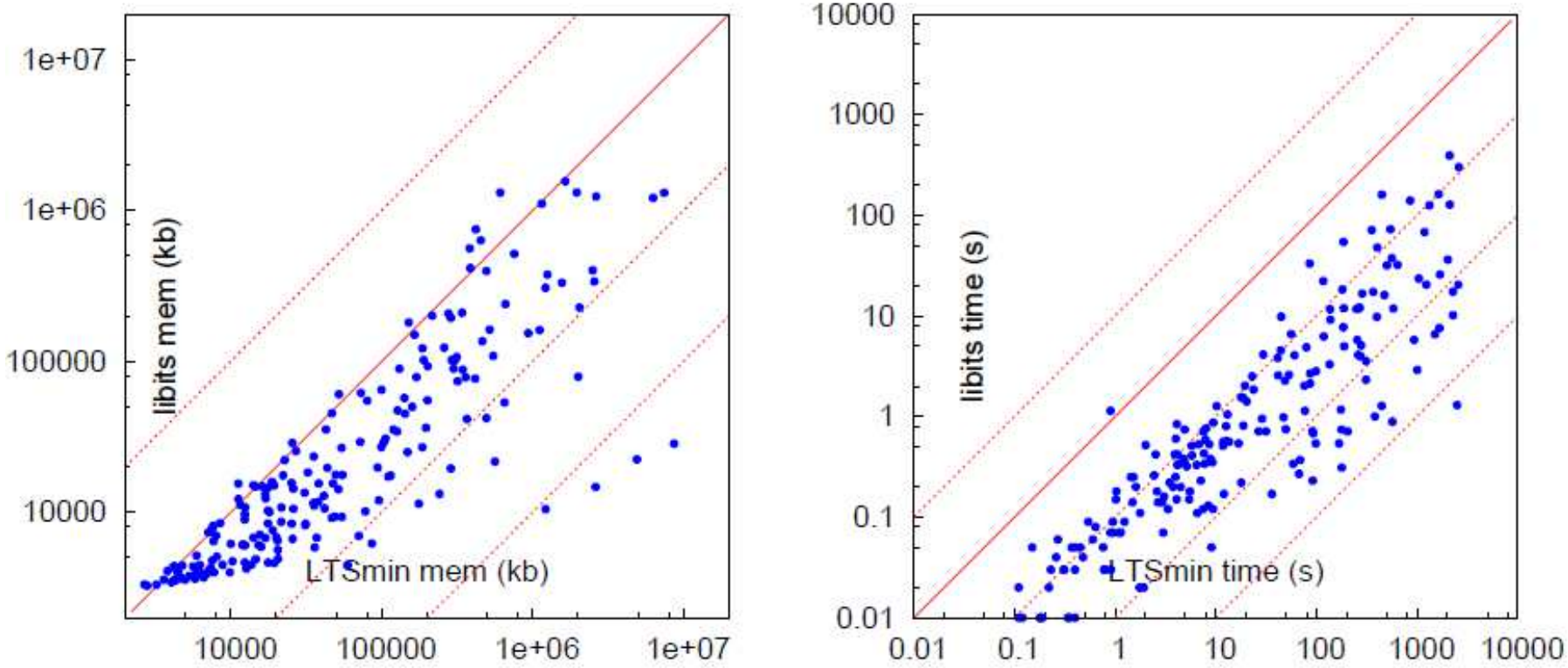
if ϕ *is constant* **then**
  return $\{(V,\phi)\}$
else
  $map < Expr, 2^V > res$
  let $\alpha_d = \{\mu \in V | \mu(x_i) = d\}$ for d
  foreach $\alpha_d \neq \emptyset$ **do**
    // Substitution
    $\theta = \phi[\delta(x_i) \leftarrow d]$
    // to remove nested δ
    for $(\psi,c) \in SolveSub(\theta,\alpha_d,i$
      // Refinement
      for $(\psi',c') \in EquivSplit($
        // Merge
        $res[\psi'] = res[\psi'] \cup c'$
return res

Algorithm 2: SolveSub(ϕ,V,i)

Input: ϕ an expression that does not depend on $x_1,\ldots,x_{i-1}$
Input: V a set of valuations that all agree on the value d of x_i
Input: i an integer between 1 and $|X|$
Output: a set of pairs $\{(\phi_1,c_1),\ldots,(\phi_n,c_n)\}$ such that $c_1,\ldots,c_n$ is a partition of V, and for each $1 \leq j \leq n$, ϕ_j is a reduced expression obtained by removing all dependencies on x_i from ϕ, and all valuations in c_j agree on this reduction ϕ_j

$map < Expr, 2^V > res$
$map < Expr, 2^V > tmp$
$tmp[\phi] = V$
while *tmp is not empty* **do**
  $(\psi,c) = tmp.pop()$
  if ψ *has an* x_i*-expression* **then**
    $\theta =$ an x_i-expression of ψ
    for $(\theta',c') \in EquivSplit(\theta,c,i)$ **do**
      $\psi' = \psi[\theta \leftarrow \theta']$
      $\psi' = \psi'[\delta(x_i) \leftarrow d]$
      $tmp[\psi'] = tmp[\psi'] \cup c'$
  else
    // ψ does not depend on x_i
    $res[\psi] = res[\psi] \cup c$
return res

Performance evaluation (BEEM)



(a) comparison on peak memory

(b) comparison on time

DDD/SDD Symbolic engine

- Superior performances
- Historically (2002+) : hard coded support for Petri nets
- Defining transition relation symbolically is difficult => high expertise
- How to define a system to take advantage of symbolic engine ?

Model Driven Development and Model-checking

- In MDD approaches
  - Build a Domain Specific Language
  - Use model transformation for specific targets
- Choosing a target formalism
  - Expressive enough to capture your semantics
  - Efficient solution engine
- We propose ITS/GAL formalism
  - Allows to express discrete state semantics
  - Symbolic model-checking

Guarded Action Language

- GAL : a « Universal Semantic Assembly Language »
  - Simple to use, easy C-like syntax
  - Straightforward Petri net style concurrent semantics
  - Integer variables and arrays + arbitrarily nested array expressions
  - Efficient symbolic solution engine
  - Subsumed by Instantiable Transition Systems (ITS), allowing hierarchical composition of GAL modules
- Meant to be a back-end target in a transformation process.
- Define your semantics in GAL.

GAL example

```
GAL system {
    // Variable declarations
    int variable = 5 ;
    array [2] tab = (1, 2) ;

    transition t1 [variable > 9] {
        tab [0] = tab [1] * tab [0] ;
        variable = variable * 5 ;
    }

    transition t2 [variable == 23] label "a" {
        tab [1] = 0 ;
    }
}
```

GAL Variables

- All variables are 32 bit integers or arrays of fixed size.
- Any variable must be initialized
  - int a = 0;
  - array [3] tab = (0,0,0);

Arithmetic and Boolean

- Terminal expressions are signed constants, parameters, variables, array access with arbitrary index expression
  - 3,-2,$MAX,x,tab[x+1],tab[tab[$MAX-x]]
- All C operators supported
  - Bitwise : &, |, ^, <<, >>, ~
  - Integer : +, -, *, /, %, **
- Boolean expressions
  - Basics : true, false, &&, ||, !
  - Comparisons of integers: ==,!=, <,<=, >,>=
- x = (y == 255) * 100; // x is 0 or 100

GAL Statements

- <lhs=rhs> assign integer expression rhs to variable designated by lhs.
- <s1;..;sn> sequence of statements, <nop> the empty sequence
- <ite(c,t,f)> an if-then-else statement
- <abort> return the empty set (!)
- <for(min, max, b)> a limited form of iteration
- <fixpoint(b)> fixpoint statement
- <call(a)> call a label (i.e. an arbitrary transition with label a) of « self »

GAL Transitions

- Tuple : <label, guard, body>
- Fire : In any state where guard is enabled, process body statement(s) atomically
- Tau (empty) label for local transitions
- Labeled transitions are not fireable by Locals outside of call or synchronization

System Parameters

- Allow easier configuration of a model

```
GAL paramSystem ($N = 2, $K = 1) {
    int variable = $N ;
    array [2] tab = ($N + $K, $N - 1) ;
    transition t1 [variable > $N] {
        tab [$K] = tab [1] * tab [0] ;
        variable = variable * 5 ;
    }
    transition t2 [variable == $N] label "a" {
        tab [1] = 0 ;
    }
}
```

Range Type definitions & Transition parameters


```
GAL paramDef ($N=2) {
    typedef paramType = 0..$N;
    typedef paramType2 = 0..1;
    int variable = 0;

    // a transition compactly modeling ($N+1)*2
    // basic transitions
    transition trans (paramType $p1, paramType2 $p2)
            [$p1 != $p2] {
        variable = $p1 + $p2;
    }
}
```

If then else

```
GAL iteExample {
    int variable = 0 ;

    transition invert [variable == 0 || variable == 1] {
        if (variable == 0) {
            variable = 1;
        } else {
            variable = 0;
        }
    }
}
```

Equivalent to xor : variable = variable ^ 1

For loop

- **Limited iteration**

```
GAL forLoop  {
    typedef Dom = 0..2;

    array [3] tab  = (0,0,0);

    transition forExample [true] {
        for ($i : Dom) {
            tab[$i] = $i;
        }
    }
}
```

```
GAL forLoop_inst {
    array [3] tab = (0, 0, 0) ;

    transition forExample [true] {
        tab [0] = 0 ;
        tab [1] = 1 ;
        tab [2] = 2 ;
    }
}
```

Call

```
GAL callExample {
    int variable = 0 ;

    transition NDassignX [variable == 0 || variable == 1] {
        self."setX" ;
    }

    transition callee1 [true] label "setX" {
        variable = 1 ;
    }

    transition callee2 [true] label "setX" {
        variable = 0 ;
    }
}
```

Abort

```
GAL abortExample ($EFT = 1, $LFT = 3) {
    int a = 1 ;
    int b = 0 ;
    int t.clock = 0 ;

    transition t [a >= 1 && t.clock >= $EFT] {
        a = a - 1 ;
        b = b + 1 ;
        t.clock = 0 ;
    }
```

Abort (2)

```
transition elapse [true] label "elapse" {
    // is t enabled ?
    if (a >= 1) {
        // is t's clock strictly less than
        // its latest firing time ?
        if (t.clock < $LFT) {
            // if yes increment t clock
            t.clock = t.clock + 1 ;
        } else {
            // otherwise, time cannot elapse,
            //  kill exploration
            abort ;
        }
```

Fixpoint : sort example


```
GAL sortEx {
  typedef index = 0..3;                    // 0 to  n-2
  array [5] tab = (3,1,2,4,5);
  int tmp = 0;

  transition swap (index $i) [ tab[$i] > tab[$i+1] ]  label "sort"
  { tmp = tab[$i];  tab[$i] = tab[$i+1]; tab[$i+1] = tmp; tmp = 0;   }

  transition sorted  label "sort" [true] {
        for ($i : index) {
                if (tab[$i] > tab[$i+1]) { abort; } } }

  transition sort [true] {
        fixpoint { self."sort"; }
  }
}
```

Transient

```
GAL loopTransient {
    int i = 0 ;
    array [4] tab = (0, 0, 0, 0) ;
    transition t1 [i < 4] {
        tab [i] = i ;
        if (i < 3) {
            i = i + 1 ;
        } else {
            i = 0 ;
        }
    }
    TRANSIENT = (i != 0) ;
}
```

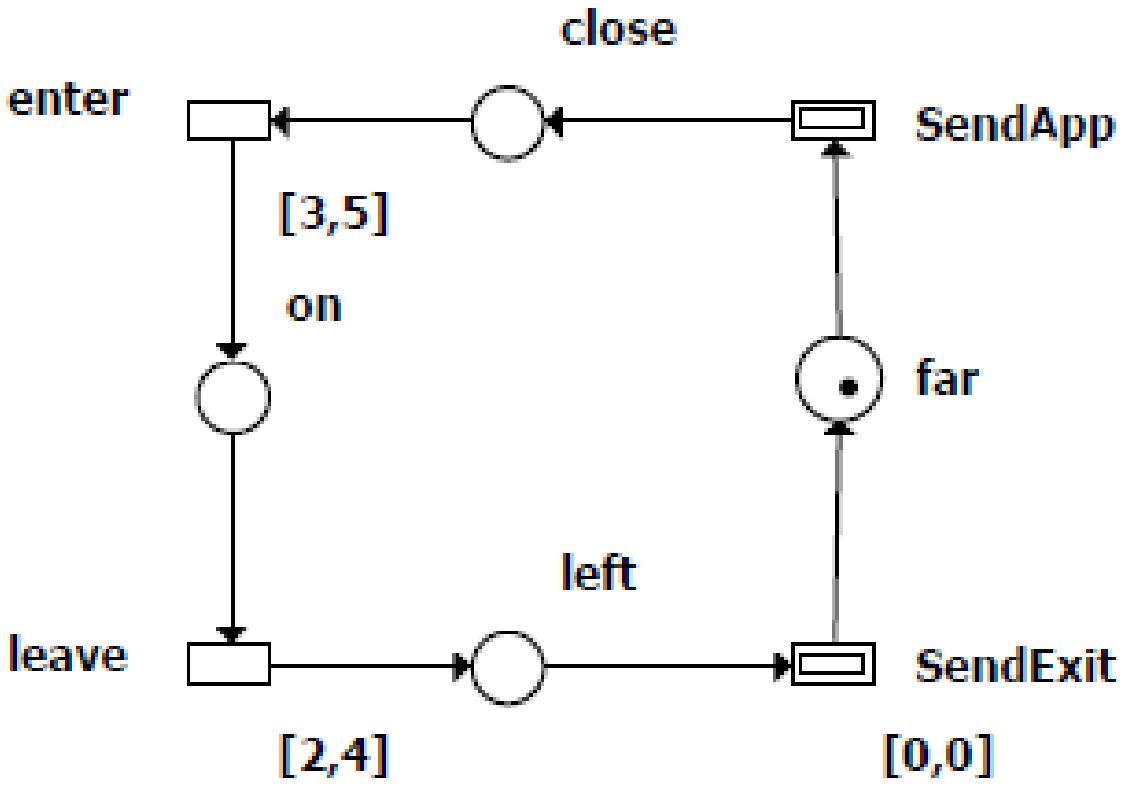
Some applications of GAL

- Petri nets
- Discrete Time Petri nets
- Colored Petri Nets
- Divine (Promela-like) models
- CCSL clock logic
- …

Encoding Petri nets

- Each place => a variable
- Each transition => a transition
  - Guard tests enabling conditions
  - Actions update state variables
- Easy to support many extensions of PN
  - Test arcs
  - Reset arcs
  - Inhibitor
  - Capacity places
  - …

Labeled Discrete TPN model of a train



Discrete Time Petri Net Semantics

- Place -> integer variable,
  - initial value=initial marking
- Transitions -> define variable t.clock
  - unless [0,0] or [0,inf[
- Time elapse -> additional transition labeled « elapse »
  - Sequence, for each transition t

```
If (enabled(t)) {
      If ( t.clock < lft(t) ) {
            t.clock=t.clock+1;
      } else {
            abort;
      }
}
```

General case

```
If (enabled(t)) {
      abort;
}
```

[0,0] urgent case

```
If (enabled(t)) {
      If ( t.clock < eft(t) ) {
            t.clock=t.clock+1;
      }
}
```

[a,inf[infinite lft case

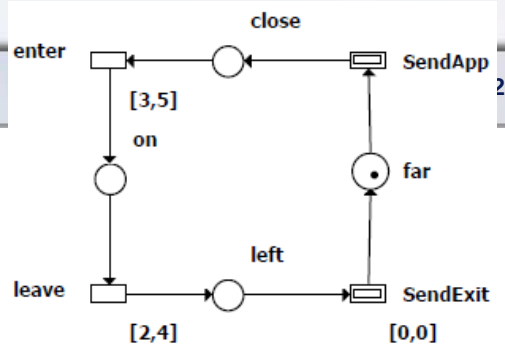
DTPN : Transitions

- Transition t -> transition <l,g,b>
  - l: Label is copied from input transition,
  - g: enabled(t) and t.clock >= eft(t)
  - b:
    - *update place markings according to arc types and inscriptions,*
    - *then reset current clock,*
    - *then reset any disabled transition clocks (call(reset))*
- **Reset disabled transitions:** <reset,private,true,b> for each transition t :

```
If (! enabled(t)) {
        t.clock=0;
}
 General case
```

```
If no clock do nothing
```

Train Example, GAL




```
GAL train {
      int far = 1 ;
      int close = 0 ;
      int on = 0 ;
      int left = 0 ;
      int enter.clock = 0 ;
      int leave.clock = 0 ;

      transition SendApp [ far >= 1 ] label "SendApp" {
            far = far - 1 ;
            clo
            sel
      }

      transitio
            clo
            on
            ent
            sel
      }

      transitio
            on
            lef
            lea
            sel
      }

      transition SendExit [ left >= 1 ] label "SendExit" {
            left = left - 1 ;
            far = far + 1 ;
            self.reset ;
      }
```

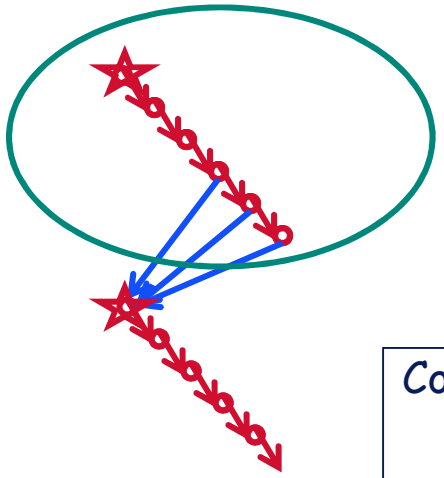
```
transition elapse [ True ] label "elapse" {
      if (close >= 1) {
            if (enter.clock < 5) {
                  enter.clock = enter.clock + 1 ;
            } else {
                  abort ;
            }
      }
```

```
transition enter [ close >= 1 && enter.clock >= 3 ] {
      close = close - 1 ;
      on = on + 1 ;
      enter.clock = 0 ;
      self.reset ;
}
```

```
eave.clock + 1 ;

eset" {

      if (! on >= 1) {
            leave.clock = 0 ;
      }
}

}
```

Using a fixpoint

- Essential states construction [Popova]
  - Letting time elapse cannot disable transitions
  - Let time progress, only consider states that immediately follow a discrete transition



Time Elapse

Discrete Transition

Essential State

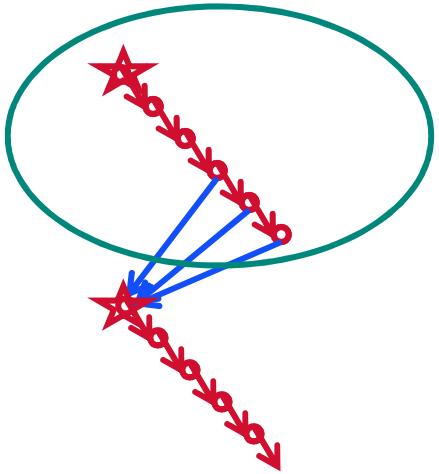
Compute green set :
let time elapse if it can
cumulate states
Fire transitions (succ) from resulting set

Fixpoint for « Essential States » of TPN

```
GAL tpnModel ($EFT = 3, $LFT = 5) {
    int a = 1 ;
    int b = 0 ;
    int t.clock = 0 ;

    transition t [a >= 1 && t.clock >= $EFT] label "succ"
    {
        a = a - 1 ;
        b = b + 1 ;
        t.clock = 0 ;
    }
```

Essential States (2)

```
  transition elapseIfpossible
     [ a >= 1 && t.clock < $LFT] label "elapseIP" {
      t.clock = t.clock + 1 ;
  }
   transition id [true] label "elapseIP" {
  }
 transition nextState [true] {
   fixpoint {
     self."elapseIP" ;
   }
  self."succ" ; // Any discrete transition
 }
}
```

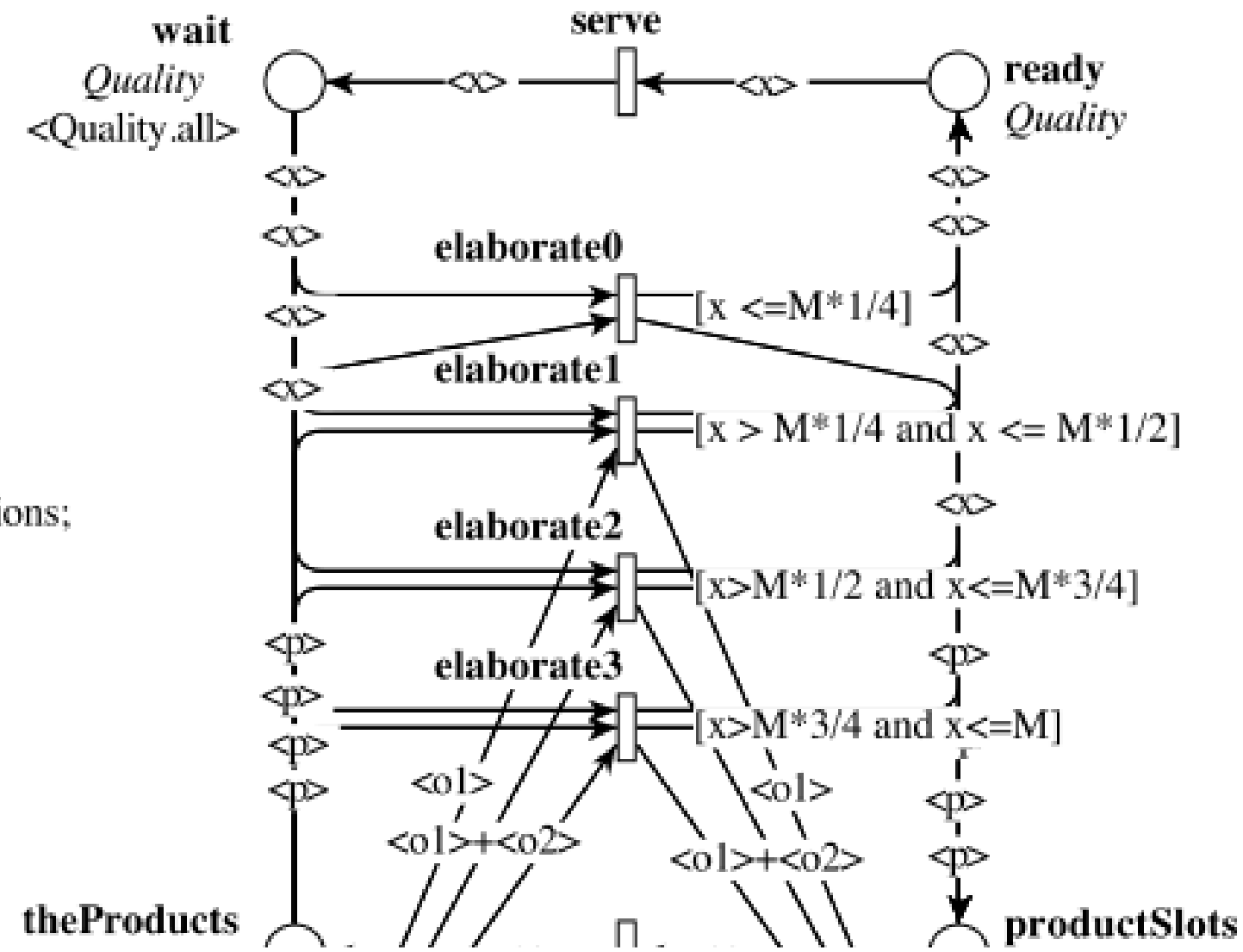


Colored Petri nets

- Each place => an array of dimension proportional to domain
  - Uncolored places => size 1
- Formal parameters => typedef of a range
- Each transition => transition with parameters

An example

Class
  Quality is 1..M;
  Products is 1..N;
  Options is 1..N;
Var
  p in Products;
  o1, o2, o3 in Options;
  x in Quality;



CPN example

```
Class
   Quality is 1..M;
   Products is 1..N;
   Options is 1..N;
Var
   p in Products;
   o1, o2, o3 in Options;
   x in Quality;
```

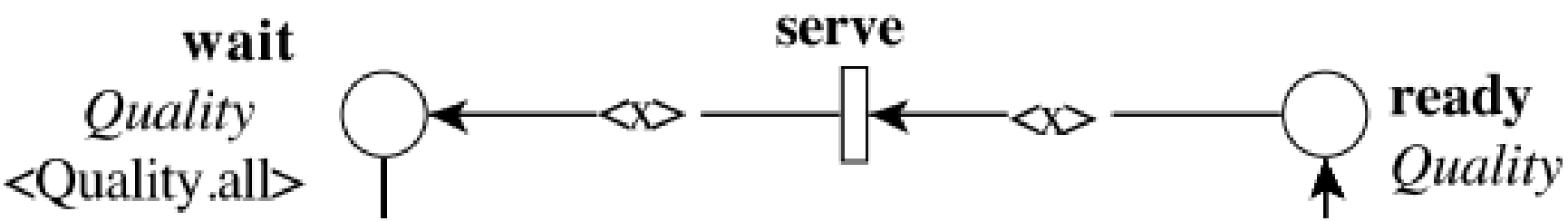
```
GAL DrinkVending2 {
typedef Options = 0 .. 1 ;
typedef Products = 0 .. 1 ;
typedef Quality = 0 .. 7 ;
array [8] ready = (0, 0, 0, 0, 0, 0, 0, 0) ;
array [8] wait = (1, 1, 1, 1, 1, 1, 1, 1) ;
array [2] theProducts = (1, 1) ;
array [2] productSlots = (0, 0) ;
array [2] theOptions = (1, 1) ;
array [2] optionSlots = (0, 0) ;
```

Color Domains
M=8, N=2

Places

CPN example (2)

```
transition serve (Quality $x) [ready [$x] >= 1] {
        ready [$x] = ready [$x] - 1 ;
        wait [$x] = wait [$x] + 1 ;
}
```



CPN example (3)

```
transition elaborate2(Options $o1,Products $p,Quality $x,Options $o2)
        [$x > 3 && $x <= 5 && theProducts [$p] >= 1
        && theOptions [$o1] >= 1 && theOptions [$o2] >= 1
        && wait [$x] >= 1] {
theProducts [$p] = theProducts [$p] - 1 ;
theOptions [$o1] = theOptions [$o1] - 1 ;
theOptions [$o2] = theOptions [$o2] - 1 ;
wait [$x] = wait [$x] - 1 ;
optionSlots [$o2] = optionSlots [$o2] + 1 ;
optionSlots [$o1] = optionSlots [$o1] + 1 ;
productSlots [$p] = productSlots [$p] + 1 ;
ready [$x] = ready [$x] + 1 ;
}
```

Divine : a language for concurrent process

Global Variables

```
byte id;
byte t[3] = { 255 ,255 ,255  };
process P_0 {
        state NCS, try, wait, CS;
        init NCS;
trans
        NCS -> try
{ guard id == 0; effect t[0] = 2;},
        try -> wait
{ effect t[0] = 3, id =0 +1; },
        wait -> wait
{ guard t[0] == 0; effect t[0] = 255;}, …
```

Process + Channels

Analyzing Divine models

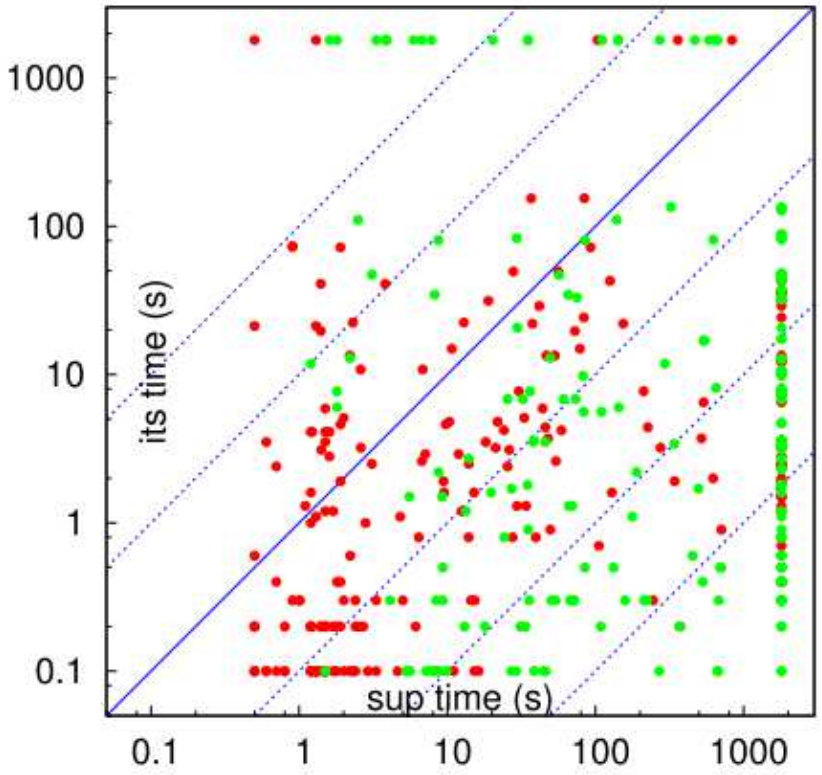
- Translate Divine concepts to GAL
  - Process state => variable
  - Divine variables and arrays => GAL equivalent
  - Guards, Instructions => GAL equivalent
  - Synchronizations => use GAL call semantics
  - Channels => GAL arrays + variable for size

Comparison with super_prove

- reachability properties: 4 cores, 900s wall-clock, 1Gb (HWMCC)
- there are difficult instances for both tools

UNSAT
SAT

instances	456
libits	376
super_prove	282
both	258
none	56



Composite Model

- A composite contains a set of ITS instances
- State is cartesian product of subcomponent states
- Synchronizations : <label,body>
  - <s1;..;sn> sequence of statements
  - <i.call(a)> call a label of subcomponent i.
  - <self.call(a)> call a label (i.e. an arbitrary synchronization with label a) of « this »

Composite example

```
composite traingate {
 train t ;
 gate g ;
 synchronization approach
  {t.App ; g.Close ;}
 synchronization leave
  {t.Exit ; g.Open ;}
 synchronization elapse label "elapse "
  {t.elapse ; g.elapse ;}
}
```

A GAL counter

```
GAL counter {
  int cpt = 0;
  transition inc label "inc"
   {cpt = cpt + 1;}
  transition dec [cpt > 0] label "dec"
   {cpt = cpt - 1;}
  transition z [cpt==0] label "iszero"
   {}
  transition nz [cpt!=0] label "notzero"
   {}
}
```

A controlled gate


```
composite controlledgate {
 counter c ;
 gate g ;
 synchronization enterfirst label "Close "
  {c.iszero ; c.inc ; g.close ;}
 synchronization enterother label "Close "
  {c.notzero ; c.inc ;}
 synchronization leavelast label "Open "
  {c.dec ; c.iszero ; g.open ;}
 synchronization leaveother label "Open "
  {c.dec ; c.notzero ;}
 synchronization elapse label "elapse "
  {g.elapse ;}
}
```

Performance: Reachability

Fischer (N is the number of processes)

	Roméo			RED		UPPAAL/sym			Roméo/SDD		
N	*tm*	*mm*	*sm*	*tm*	*mm*	*tm*	*mm*	*sm*	*tm*	*mm*	*sm*
8	1 051	282 108	740 633	11	278 028	0.01	160	137	0.1	2 020	$1.17\ 10^{6}$
9	73 071	$1.77\ 10^{6}$	$3.72\ 10^{6}$	67	785 108	0.03	160	172	0.1	2 156	$6.20\ 10^{6}$
10	DNF	OOM	OOM	652	$2.35\ 10^{6}$	0.1	160	211	0.1	2 332	$3.26\ 10^{7}$
170	-	-	-	-	OOM	7 783	47 956	57 971	23	101 896	$2.27\ 10^{120}$
700	-	-	-	-	-	DNF	-	-	1391	$1.82\ 10^{6}$	$2.66\ 10^{491}$
730	-	-	-	-	-	-	-	-	1803	$2.33\ 10^{6}$	$2.58\ 10^{512}$

Train (N is the number of trains)

	Roméo			RED		UPPAAL/sym			Roméo/SDD		
N	*tm*	*mm*	*sm*	*tm*	*mm*	*tm*	*mm*	*sm*	*tm*	*mm*	*sm*
6	43.1	36 948	29 640	7	202 412	0.14	908	432	1.5	7 360	$4.83\ 10^{6}$
7	6 115	377 452	131 517	66	723 428	0.23	3 200	957	2.5	10 304	$6.28\ 10^{7}$
8	DNF	-	-	-	OOM	1	3 336	2 078	4	14 188	$8.16\ 10^{8}$
13	-	-	-	-	-	2 634	13 188	79 598	26	56 660	$3.02\ 10^{14}$
15	-	-	-	-	-	60 860	61 256		42	86 360	$5.11\ 10^{16}$
16	-	-	-	-	-	DNF	-	-	52	104 848	$6.65\ 10^{17}$
44	-	-	-	-	-	-	-	-	1143	$2.13\ 10^{6}$	$1.03\ 10^{49}$

Table 1. Performances measured for the *Fischer* and *train* models.

Conclusion

- GAL modeling
  - A natural model for many discrete semantics
  - Efficient symbolic solution
- ITS Composite for compositional modeling
  - Modular and hierarchical specifications
  - Efficient support for symmetric models
- Model checking engine
  - Reachability (shortest traces)
  - CTL (Forward algorithms, traces)
  - LTL with Spot (Fully symbolic or hybrid)

GAL Eclipse plugin (Thanks Xtext !)

Java - Models/loop.gal - Eclipse SDK

File Edit Navigate Search Project Run Window Help

Java | Coloane Modeler | Debug | Team Synchronizing | SVN Repository Exploring | CVS Repository Exploring | Plug-in Developme

Package E... | Type Hier... | Navigator

- ImportPNML
- incubation
- MCC4PNMLLIP6Fr-2012
- MCJa-0.1
- Models
  - loop.gal
  - loop.inst.gal
  - loop.sep.flat.gal
  - loop.sep.flat.inst.gal
  - small_loop.gal
  - sort.gal
  - traceUnitaire.rtf
- org.xtext.example.mydsl
- org.xtext.example.mydsl.sdk
- org.xtext.example.mydsl.tests

*loop.gal | loop.sep.fl... | fr.lip6.mov... | GalFormatte... | XtypeGramma...

```
1  GAL model {
2    typedef A = 0..2;
3    typedef B = 0..2;
4
5    int dummy = 0;
6
7    transition tr [true] {
8      dum = 0;
9
10     for ($i : A) {
11       dummy = 1;
12     }
13
14     for ($i : B) {
15       dummy = 1;
16     }
17
18     for ($i : B) {
19       dummy = 1;
20     }
```

- abort
- fixpoint
- for
- if
- pop
- push
- self

Ctrl+Space to show shortest proposals

Outline

- loop

Probl... | Java... | Decla...

Thank you for your attention !

SDD and ITS-tools are distributed as an open-source LGPL/GPL C++ source and pre-compiled tools :

http://ddd.lip6.fr

Eclipse plugin for GAL/ITS manipulation and CTL model-checking

http://coloane.lip6.fr/night-updates