
NTIF

A General Symbolic Model for Communicating
Sequential Processes with Data

Hubert Garavel, Frédéric Lang

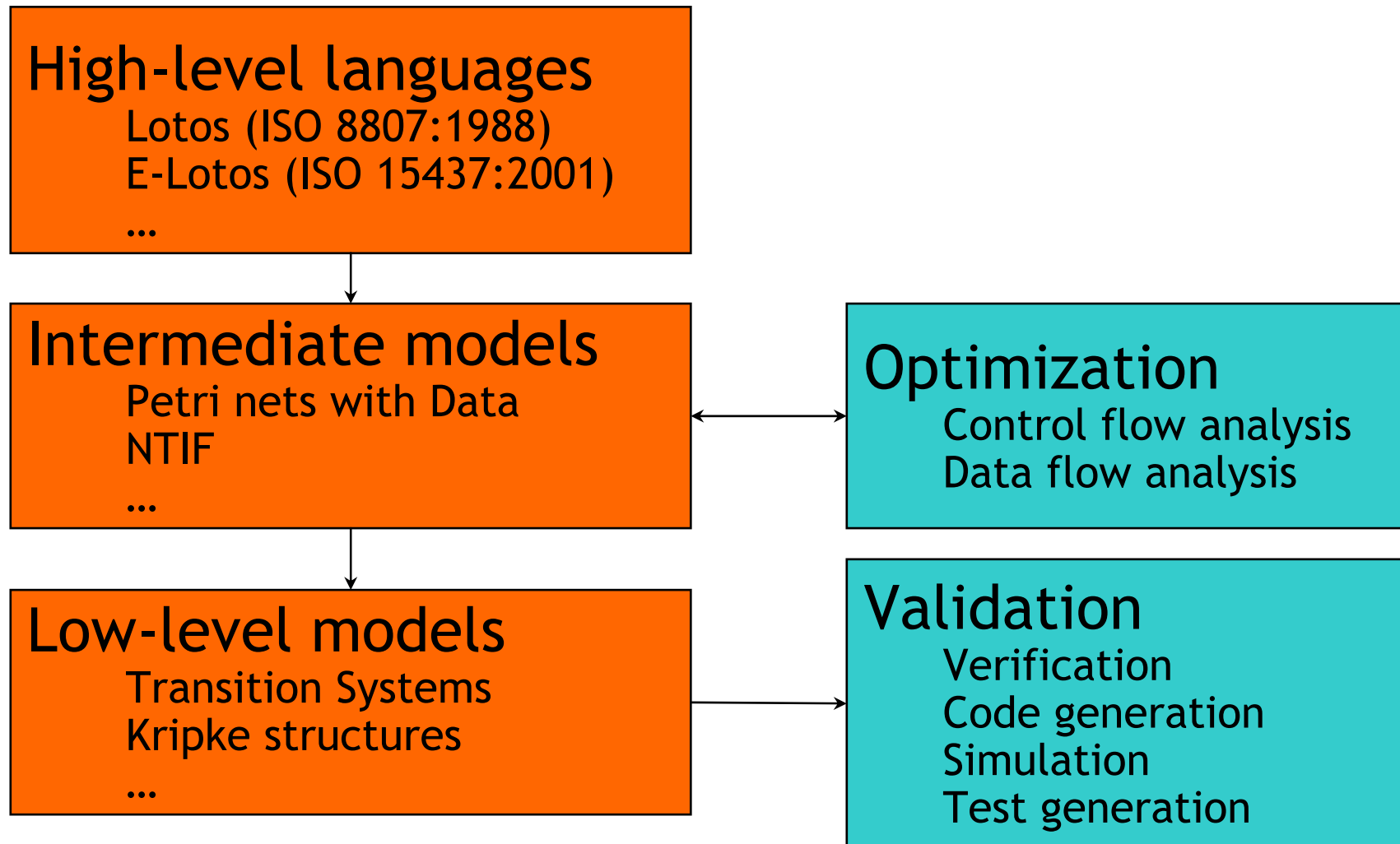
INRIA Rhône-Alpes / VASY
655, avenue de l'Europe
F-38330 Montbonnot Saint Martin



Introduction



Intermediate Models



An Early Example of Intermediate Model

- Interpreted Petri nets with data (Garavel & Sifakis, 1990)
 - Global state variables
 - Net transitions labeled with communication events
 - Net transitions labeled with actions (guards, variable assignments, variable resets, etc.)
- Benefits
 - Separate language-dependent and independent aspects
 - Improve the efficiency of verification algorithms by operating on a simplified semantic model
 - Can be used for several high-level languages



Choosing an Intermediate Model

Two conflicting requirements:

- **Generality and expressiveness**
 - The model should be useable for several languages
 - The model should preserve language semantics
- **Simplicity**
 - The model should not contain unnecessary details
 - The model should not complicate analysis



Existing Formalisms Based on Condition/Action



Condition/Action Based Models

- Many models based on condition/action

Input/Output Automata, Linear Process Operators (muCRL), Symbolic Transition Systems (CCS), IF (SDL), Communicating State Machines (Basic LOTOS), etc.

- Condition/action: Transitions $s \xrightarrow{E \Rightarrow A / C} s'$ where

- s, s' are states
- E is a boolean condition for firing the transition
- A (action) is a sequence of variable assignments
- C is a communication event
 - Input: $G ?V$ (gate G , variable V)
 - Output: $G !E$ (gate G , expression E)
 - Silent event: τ



Limitations of Condition/Action Models

4 main limitations (developed in next slides)

- Transition firing is determined by a unique condition
- Conditions and actions can not be intertwined
- The language of actions is too simple to preserve *big-step semantics*
- Boolean conditions present in the high-level language are duplicated



1. Unique Firing Condition (1/2)

- Depending on the formalism, the condition E is evaluated:

- Either "*before*" communication

Example $V > 2 \Rightarrow \text{null} / G ?V$

means "if $V > 2$ then read on gate G a new value for V "

- Or "*during*" communication

Same example means "fire transition only if it is possible to read on gate G a value for V greater than 2"

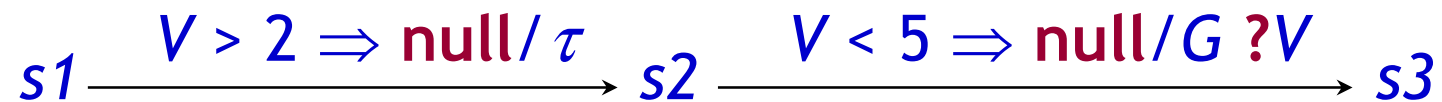


Unique Firing Condition (2/2)

- Both cases are possible at language level

Example **if** $V > 2$ **then** $G ?V$ **where** $V < 5$

- Not implementable if evaluation "*before*"
- Two transitions necessary if evaluation "*during*"



- Both types of conditions must be provided to avoid additional transitions



2. No Intertwining of Condition and Action

- In all models condition is evaluated before action
- In practice actions (assignments) preceding condition would be useful

Example **if** $F(F'(V))$ **then** $G !V; V := F'(V)$

should be writeable as

$W := F'(V);$ **if** $F(W)$ **then** $G !V; V := W$

without adding extra transitions in the model

- More generally: Intertwining conditions and actions is necessary



3. Semantics (1/3)

- Two kinds of concurrent language semantics exist
 - *Small-step*: one transition per assignment

Example $V_1 := 0; V_2 := 0; V_3 := 0$

- 3 transitions by default in PROMELA
- possible to aggregate explicitly: `dstep`
- *Big-step*: transitions induced by communication

Example $V_1 := 0; V_2 := 0; V_3 := 0; G !V_1$

- 1 transition in E-LOTOS
- Variables are not shared between processes
- No transitions associated to pure sequences of assignments



Semantics (2/3)

- Semantics of condition/action models
 - If action contains at most one assignment then the semantics is purely **small-step**
 - If action enables sequential composition of assignments then the semantics is a combination of **small-step** and **big-step**
- In both cases the language of actions is **inappropriate** for real big-step semantics



Semantics (3/3)


- Example Loops and big step semantics

- $V[1] := 0; V[2] := 0; V[3] := 0$ can be translated into

$$s_0 \xrightarrow{V[1] := 0; V[2] := 0; V[3] := 0 / \tau} s_1$$

- But **for** i **in** $1..3$ **do** $V[i] := 0$ must be translated into

$$s_0 \xrightarrow{i := 1 / \tau} s_1 \xrightarrow{i > 3 \Rightarrow \text{null} / \tau} s_2$$


 $i \leq 3 \Rightarrow i := i+1; V[i] := 0 / \tau$

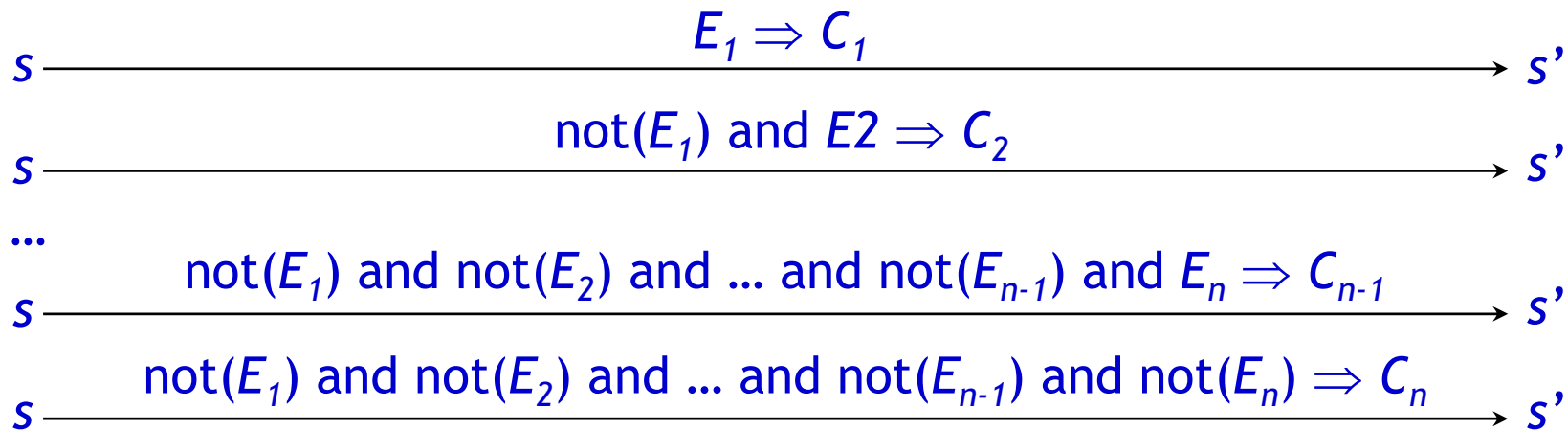
- The language of actions must be extended to preserve big-step semantics



4. Condition Duplications (1/2)

- Translation of conditionals (if-else, case) leads to condition duplications

Example if E_1 then C_1 elsif E_2 then C_2 ... else C_n
translates into



i.e., $n(n+1) / 2$ conditions to evaluate instead of n



Condition Duplications (2/2)

- Condition duplications penalize user-friendliness
 - Models are laborious to write by hand
 - Models are hard to read and debug
 - Condition duplications penalize analysis efficiency
 - Both condition and its negation must be evaluated during model checking or simulation
 - Properties that are obvious from the high-level standing point become hard to prove at the intermediate level
- Example Checking that at most one (or exactly one) transition can be fired from a given state



Conclusion on Condition/Action

- Although often used in the literature condition/action models are not good:
 - Neither for hand writing
 - Neither for reading and debugging
 - Nor for automatic processing
- A better formalism is needed: **NTIF** (created in April 2001)



NTIF: The New Technology Intermediate Form



NTIF Program

- An NTIF program is a collection of communicating sequential processes with data
 - Parallelism is left for further work
- An NTIF process is made of
 - States s, s', \dots
 - Typed parameters with condition of validity
 - Typed (local) state variables
 - *Multi-branching* transitions between states of the form "**from** s A " where A is an action containing control structures and jumps to next state



NTIF Actions

- Actions are built upon the following syntactic elements

- Types written T

- Variables V

- Gates G

- Expressions E

- Patterns P

- Offers $O ::= !E \mid ?P \text{ where } E$



Syntax of Actions

- $A ::= \text{null}$
 - | $V_0, \dots, V_n := E_0, \dots, E_n$ *Inaction*
 - | $V_0, \dots, V_n := \text{any } T_0, \dots, T_n$ *Assignment*
 - | **reset** V_0, \dots, V_n *Nondeterministic assignment*
 - | $G O_1 \dots O_n$ *Variable deactivation*
 - | **to** s *Communication (rendezvous)*
 - | $A_1; A_2$ *Jump to state*
 - | **select** $A_1 [] \dots [] A_n$ **end select** *Sequential composition*
 - | **case** E **is**
 - $P_1 \rightarrow A_1 \mid \dots \mid P_n \rightarrow A_n$
 - end case** *Nondeterministic choice*
 - | **while** E **do** A_0 **end while** *Deterministic choice*
While loop



Derived NTIF Constructs

- **for** derived from **while**
- **if-then** and **if-then-else** derived from **case**

if E then A_1 else A_2 end if =

case E is true $\rightarrow A_1$ | false $\rightarrow A_2$ end case

if E then A_1 end if = if E then A_1 else null end if

- **stop** derived from **select**
stop = **select end select**



NTIF Static Semantics

- Ensures program well-formedness
- Several checks
 - Patterns and assignments bind variables without ambiguity
 - Variables are defined before used
 - At most one communication occurs on each transition execution path (e.g., no comm. in **while** loops)
 - No blocking between a communication and jump to next state
 - Some "**case**" statements cover all possible values of a given type



NTIF Dynamic Semantics

- Formal and intuitive semantics
- Expressed in SOS form (Structured Operational Semantics)
 - Associates a (timed) LTS to each instance of a process
 - $[A], \rho \Rightarrow^l s', \rho'$ means that in store ρ :
 - A runs without deadlock
 - Processes action l
 - Jumps to state s' with store ρ'
 - "**from** s A " and $\ll [A], \rho \Rightarrow^l s', \rho' \gg$
implies a transition $\langle s, \rho \rangle \rightarrow^l \langle s', \rho' \rangle$



NTIF Solves the Limitations of Condition/Action

- Conditions can occur before and during communications thanks to **if** actions and conditional patterns

Example **if** $V > 2$ **then** $G ?V$ **where** $V < 5$ **end if**

- Conditions and actions can be freely intertwined
- Big-step semantics are preserved thanks to **while** loops
- Conditions are not duplicated thanks to the multi-branching structure of transitions



NTIF Example: if-else-elsif

```
from Cep_Test_NT
  if Deactivated then
    Init_Load_Resp.Status := x9106;
    Cep_Reply !Init_Load_Resp;
    to Cep_Init
  elsif Locked then
    Init_Load_Resp.Status := x9110;
    Cep_Reply !Init_Load_Resp;
    to Cep_Init
```

```
  elsif NT >= NT_Limit then
    Init_Load_Resp.Status := x9102;
    Cep_Reply !Init_Load_Resp;
    to Cep_Init
  else
    Load_Amount := Inquiry.Load_Amt;
    Slot_Index := 0;
    Currency_Sought := Inquiry.Currency;
    Slots_Available := 0;
    Last_Avail_Slot := SlotCount;
    to Cep_IFL_Locate_Slot
  end if
```



Same Example in IF (Condition/Action)

```
from Cep_Test_NT
  if Deactivated
  sync tau
  do {Init_Load_Resp.Status := x9106}
to S_00017 ;

from S_00017
  if Reply_Type_Value_0 = Init_Load_Resp
  sync Cep_Reply!(Reply_Type_Value_0)
to Cep_Init ;

from Cep_Test_NT
  if not Deactivated and Locked
  sync tau
  do {Init_Load_Resp.Status := x9110}
to S_00018 ;

from S_00018
  if Reply_Type_Value_0 = Init_Load_Resp
  sync Cep_Reply!(Reply_Type_Value_0)
to Cep_Init ;
```

```
from Cep_Test_NT
  if not Deactivated and
  not Locked and (NT >= NT_Limit)
  sync tau
  do { Init_Load_Resp.Status := x9102 }
to S_00019 ;

from S_00019
  if Reply_Type_Value_0 = Init_Load_Resp
  sync Cep_Reply!(ReplyType_Value_0)
to Cep_Init ;

from Cep_Test_NT
  if not Deactivated and not Locked and
  not (NT >= NT_Limit)
  sync tau
  do {
    Load_Amount := Inquiry.Load_Amt,
    Slot_Index := 0,
    Currency_Sought := Inquiry.Currency,
    Slots_Available := 0 ,
    Last_Avail_Slot := Slot_Count
  }
to Cep_IFL_Locate_Slot ;
```



NTIF Example: case

```
from Cep_Command_Case
  Cep_Command ?Inquiry;
  case Inquiry.Command is
    ALLSLOTS00 ->
      Slots_Reported := 0;
      Slot_Index := 0;
      to Cep_Slot_Inquiry_Sequence
| ALLSLOTS01 ->
      to Cep_SIQ_Reply
| any ->
      to Cep_Command_Out_Of_Sequence
  end case
```



Same Example in IF

```
from Cep_Command_Case
  sync Cep_Command ?Command_Type_Value_0
  do {Inquiry := Command_Type_Value_0}
to S_00023
```

```
from S_00023
  if Inquiry.Command = ALLSLOTS00
  sync tau
  do { Slots_Reported := 0, Slot_Index := 0 }
to Cep_Slot_Inquiry_Sequence ;
```

```
from S_00023
  if (Inquiry.Command <> ALLSLOTS00) and (Inquiry.Command = ALLSLOTS01)
  sync tau
to Cep_SIQ_Reply ;
```

```
from S_00023
  if (Inquiry.Command <> ALLSLOTS00) and (Inquiry.Command <> ALLSLOTS01)
  sync tau
to Cep_Command_Out_Of_Sequence ;
```



NTIF Example: while

```
from Cep_Slot_Inquiry_Sequence
while (Slot_Index < Slot_Count) do
  if (Slots[Slot_Index].In_Use) then
    Slots[Slot_Index].Reported := false;
    Slot_Index := Slot_Index + 1
  else
    Slots[Slot_Index].Reported := true;
    Slot_Index := Slot_Index + 1;
    Slots_Reported := Slots_Reported + 1
  end if
end while;
Cep_Reply !Slot_Info;
to Cep_SIQ_Reply
```



Same Example in IF

```
from Cep_Slot_Inquiry_Sequence
  sync tau
to S_00009 ;

from S_00009
  if (Slot_Index < Slot_Count) and
     Slots[Slot_Index].In_Use
  sync tau
  do {
    Slots[Slot_Index].Reported := false,
    Slot_Index := Slot_Index + 1
  }
to S_00009 ;
```

```
from S_00009
  if (Slot_Index < Slot_Count) and
     not Slots[Slot_Index].In_Use
  sync tau
  do {
    Slots[Slot_Index].Reported := true,
    Slot_Index := Slot_Index + 1,
    Slots_Reported := Slots_Reported + 1
  }
to S_00009 ;

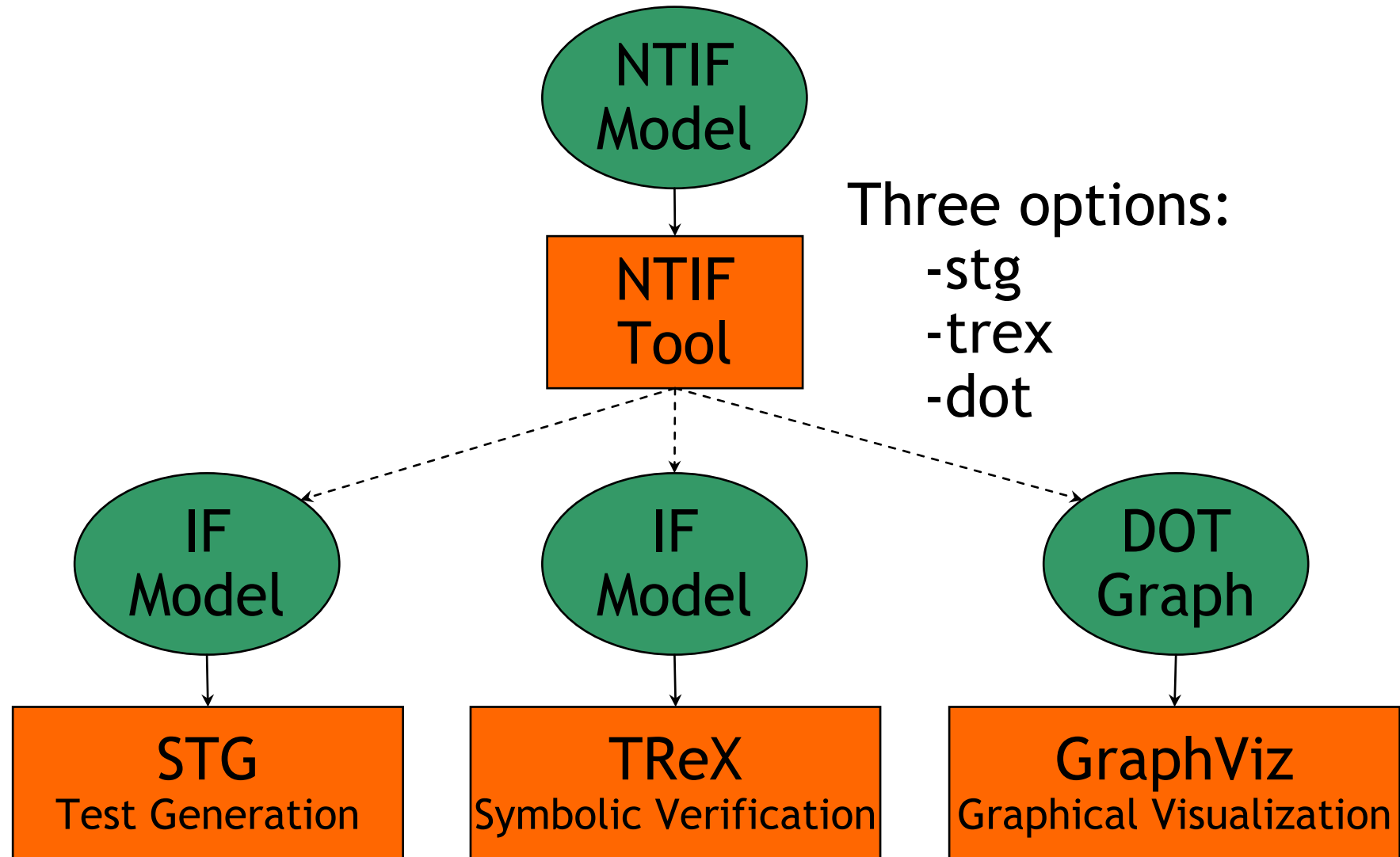
from S_00009
  if not (SlotIndex < SlotCount) and
     (Reply_Type_Value_0 = Slot_Info)
  sync Cep_Reply !Reply_Type_Value_0
to Cep_SIQ_Reply ;
```



Tools for NTIF



The NTIF Tool



The NTIF Tool

- Symbolic unfolding of **NTIF** transitions into two dialects of IF 1.0
 - IF for STG (INRIA, Rennes)
used for symbolic test generation
 - IF for TReX (LIAFA, Paris)
used for symbolic reachability analysis
- Graphical visualization of NTIF descriptions using the DOT format of the GraphViz Package (AT&T)



Development of the NTIF Tool

- Started in April 2001
- Use of the SYNTAX + TRAIAN compiler construction technology
<http://www.inrialpes.fr/vasy/Publications/Garavel-Lang-Mateescu-02.html>
- 12 000 lines of code
 - 2 200 lines of SYNTAX code
 - 8 300 lines of LOTOS NT code
 - 1 500 lines of C code
- Versions for Solaris, Linux, and Windows



Case Studies



Electronic Purse

- Specification of a multi-currency electronic purse (CEPS standard) starting from an existing IF 1.0 description (Feb. 2001)
- Numerous bugs found in the IF code due to:
 - Condition duplications: non-exclusive conditions, non-covered cases
 - Use of undefined variables
- Translation into IF using NTIF and symbolic test generation with STG
- Currently : symbolic verification with TReX



Operating System for Smart Card

- Administrative commands of an OS for smart card dedicated to 3GPP mobile telephony (F.-X. Ponscarne, INRIA, Rennes, July 2001)
- Case study performed in industrial context (provided by Schlumberger)
- Translation into IF with NTIF and symbolic test generation with STG



Statistics NTIF vs IF

- NTIF leads to more concise descriptions, containing less states and transitions than IF

	CEPS			OS 3GPP		
	IF	NTIF	% ↓	IF	NTIF	% ↓
# lines	598	418	30 %	697	498	28 %
# transitions	63	23	63 %	78	22	71 %
# states	31	21	32 %	34	20	41 %
Branching	1	2.21		1	2.77	



Graphical Visualization

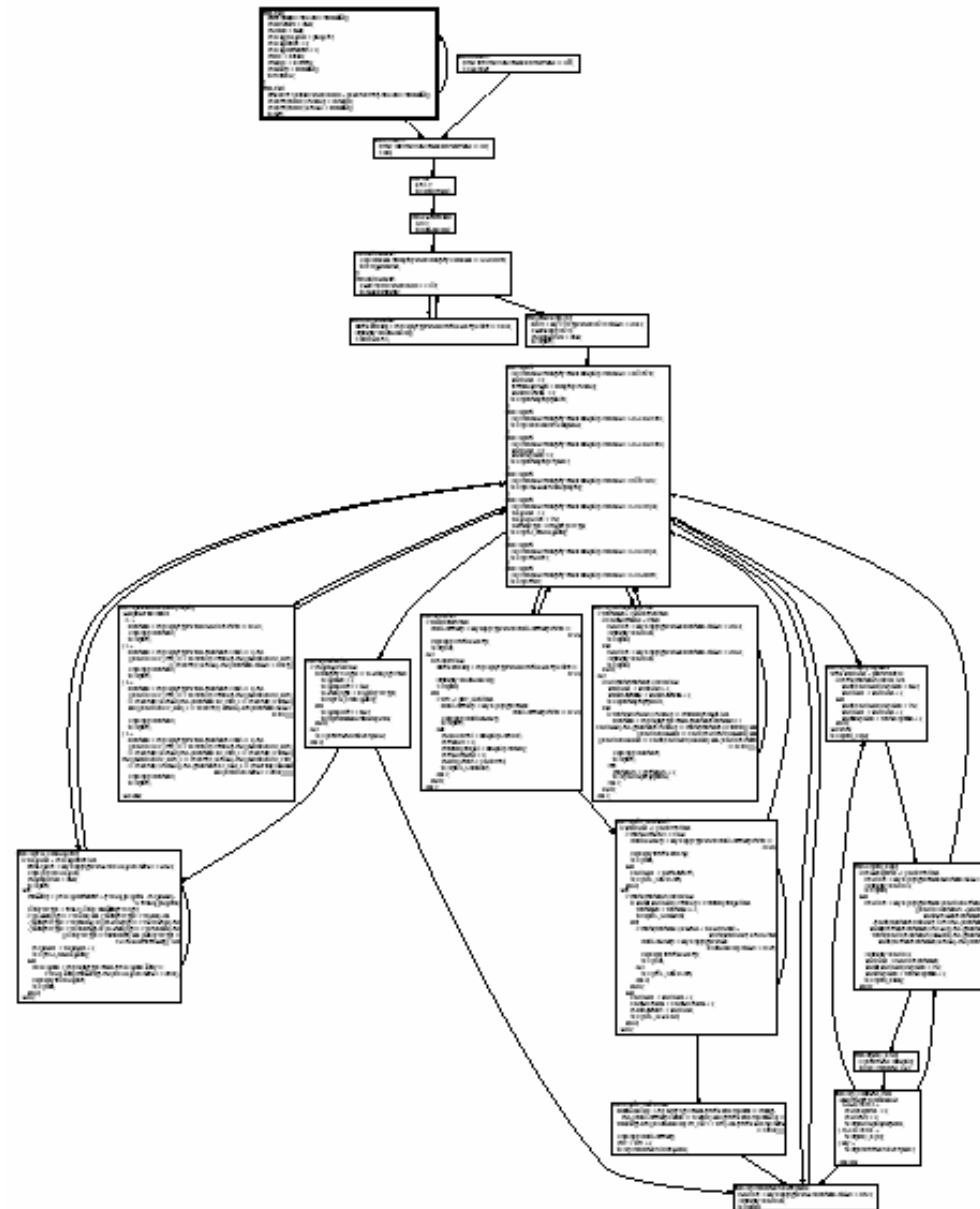
- NTIF leads to better structured descriptions as can be seen using DOT

Graphical visualization of the CEPS encoded in **IF**
(Produced with STG)



Graphical Visualization

Graphical visualization of the CEPS encoded in NTIF (NTIF tool)



Zoom on a State

CEPS state
encoded in NTIF

```
from CepIFL_LocateSlot
if vSlotIndex >= pSlotCount then
  if vSlotsAvailable == 0 then
    mInitLoadResp := any ReplyType where mInitLoadResp.Status ==
                                                                x9401;

    CepReply !mInitLoadResp;
    to CepInit;
  else
    vSlotIndex := vLastAvailSlot;
    to CepIFL_InitForLoad;
  end if;
else
  if vSlots[vSlotIndex].InUse then
    if vSlots[vSlotIndex].Currency != vCurrencySought then
      vSlotIndex := vSlotIndex + 1;
      to CepIFL_LocateSlot;
    else
      if vSlots[vSlotIndex].Balance + vLoadAmount >
          vSlots[vSlotIndex].BalMax then
        mInitLoadResp := any ReplyType where
            mInitLoadResp.Status == x9402;

        CepReply !mInitLoadResp;
        to CepInit;
      else
        to CepIFL_InitForLoad;
      end if;
    end if;
  else
    vSlotIndex := vSlotIndex + 1;
    vSlotsAvailable := vSlotsAvailable + 1;
    vLastAvailSlot := vSlotIndex;
    to CepIFL_LocateSlot;
  end if;
end if;
```



Conclusion



Conclusion

- Condition/action models are ill-adapted for system description and analysis (symbolic or exhaustive)
- Created in April 2001, **NTIF** is a structured intermediate model that solves the problems
- Tools exist and have been applied to several case-studies



Ongoing Work

- Implementation of the full language of data (records, arrays, lists, trees, constructor-based types)
- NTIF extension with time constructs (delays, urgency, etc.)
- Verification of time constraints
 - Extension of the connection to TReX
 - Connection to UppAal



Future Work

- Compiling LOTOS and E-LOTOS via NTIF, which requires extensions to support
 - Parallelism
 - Exceptions
- Connection to CADP for enumerative verification, simulation, and test



More Information..

Conference paper published at FORTE 2002

<http://www.inrialpes.fr/vasy/Publications/Garavel-Lang-02.html>

