

# Abstract State Machines Model Checking



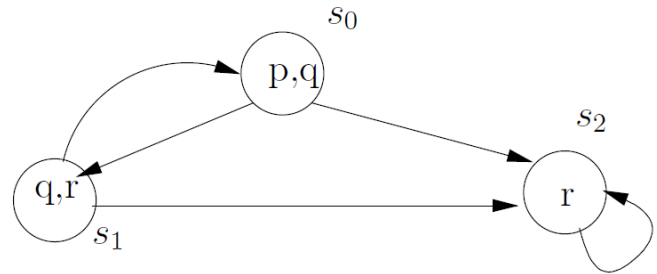
AsmetaSMV

# Model checking

---

- Formal verification technique of properties defined in a temporal logic.
- A model checker works in three steps:
  1. definition of a model  $\mathcal{M}$  using the *Kripke structures*;
  2. definition of a temporal formula  $\phi$ , that describes a property that we want to verify;
  3. the model checker verifies that  $\mathcal{M} \models \phi$ .
- Exhaustive verification of all the state space.
- Finite domains.

# Kripke structure



A Kripke structure is defined by the 4-tuple

$$\mathcal{M} = (S, \Delta, S_0, L)$$

where:

- $S$  is a finite set of states;
- $\Delta$  (or  $\rightarrow$ ) is a transition total relation, that is

$$\forall s \in S \exists s' \in S \text{ such that } s \rightarrow s'$$

- $S_0 \subseteq S$  is the set of initial states;
- $L : S \rightarrow 2^{AP}$  is a labelling function that links each state with a label; the label lists the atomic propositions that are true in that state.  $AP$  is a set of atomic propositions.

# Temporal logics

---

Temporal logics are divided into:

- *Linear Time Logics* (LTL) represent time as infinite sequences of instant; you can declare properties that must be true over all sequences;
- *Branching Time Logics* (BTL) represent time as a tree, where the root is the initial instant and its children the possible evolutions of the system; you can declare properties concerning all the paths or just some of them.

Temporal logics, moreover, can be classified in continuous time logics and discrete time logics.

# Computation Tree Logic (CTL)

---

- *Computation Tree Logic* (CTL), is a discrete time BTL.
- CTL permits to express logic formulas concerning paths, that is sequences of state transitions.
- Each CTL formula has a path quantifier that says if the formula must be true in all paths (*A, along All paths*) or if must be true in at least one path (*E, Exists at least one path*). Moreover can be used the temporal operators:
  - $X p$ : the property  $p$  must be verified in the next state;
  - $F p$ : the property  $p$  must be verified in a future state;
  - $G p$ : the property  $p$  must be verified in all the states;
  - $p U q$ : the property  $p$  must be true until the  $q$  property becomes true.

# Computation Tree Logic (CTL) - Allowed formulas

---

## Allowed formulas

It's  $AP\{p, q, r, \dots\}$  a set of atomic formulas; CTL formulas can be expressed in the following way:

$$\phi ::= \top \mid \perp \mid p \in AP \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \mid AX\phi \mid EX\phi \mid A[\phi U \phi] \mid E[\phi U \phi] \mid AG\phi \mid EG\phi \mid AF\phi \mid EF\phi$$

where  $\top$ ,  $\perp$ ,  $\neg$ ,  $\wedge$ ,  $\vee$  and  $\rightarrow$  are the connectives of propositional logic and  $AX$ ,  $EX$ ,  $AG$ ,  $EG$ ,  $AU$ ,  $EU$ ,  $AF$  and  $EF$  are temporal connectives .

## Operators priority

The unary operators have the highest priority; then there are the binary operators  $\vee$  and  $\wedge$  and, at last, the binary operators  $\rightarrow$ ,  $AU$  and  $EU$ .

# NuSMV

---

- NuSMV<sup>1</sup> is a symbolic model checker derived from CMU SMV;
- permits to verify properties written both in *Computation Tree Logic* (CTL) and in *Linear Temporal Logic* (LTL);
- the internal representation of the model uses the *Binary Decision Diagrams* (BDDs);
- states are determined by the variables values;
- transitions between states are determined by the updates of the variables.

# AsmetaSMV

## Goals

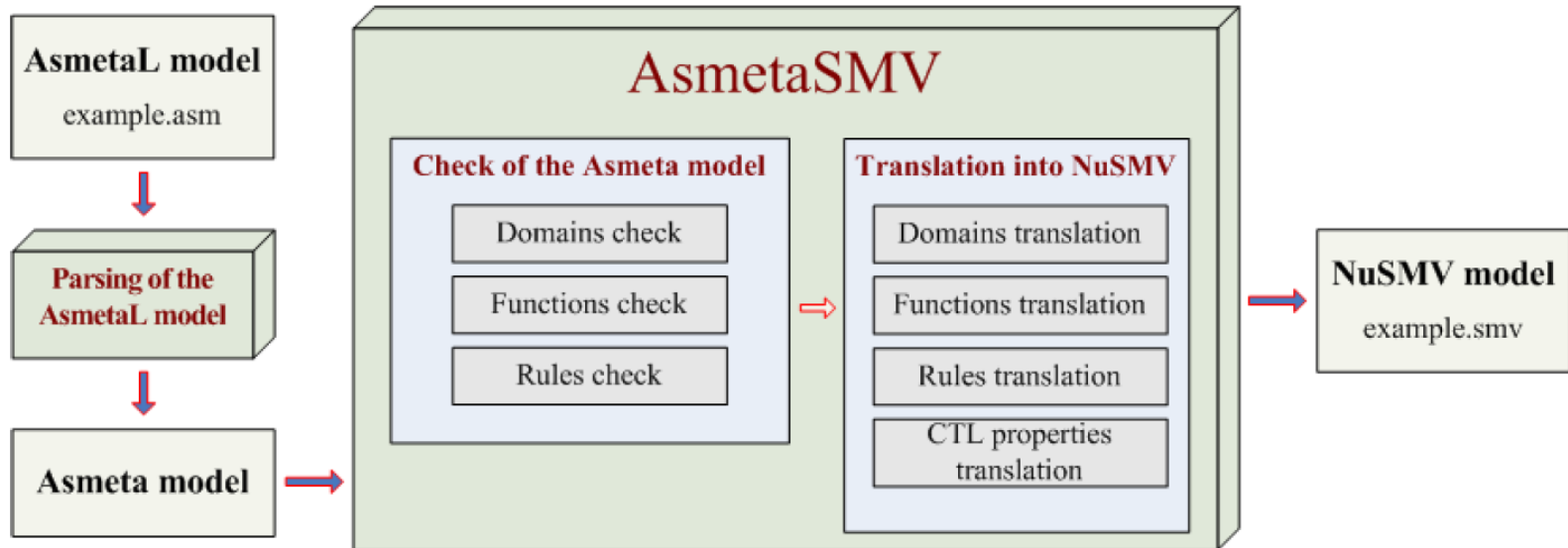
- Exploit the capabilities of NuSMV in ASMETA;
- to be able to define CTL and LTL properties directly in the AsmetaL model;
- the user could ignore the NuSMV syntax, he must only know the temporal operators.

## Functioning of the tool

- Writing of an AsmetaL code;
- translation of the AsmetaL code into a NuSMV code;
- execution of the NuSMV code with the model checker.



# AsmetaSMV architecture



# Mapping overview

---

- AsmetaSMV **cannot translate any AsmetaL element** into NuSMV; we say that an AsmetaL element is supported if it can be translated into NuSMV, otherwise that it's not supported;
- an AsmetaL element is not supported because:
  - it requires conditions that cannot be satisfied in NuSMV. For example, type domains Real, Integer, Char, don't have a corresponding type in NuSMV and so they cannot be used as domains or codomains of functions in AsmetaL models that must be translated;
  - the translation would be too complicated; it's the case of many turbo rules.

# Domains

---

## Classification of AsmetaL domains

- type domains:
  - basic type domains: *Complex*, *Real*, *Integer*, *Natural*, *String*, *Char*, *Boolean*, *Rule* and *Undef*;
  - structured type domains: *ProductDomain*, *SequenceDomain*, *PowersetDomain*, *BagDomain* and *MapDomain*;
  - abstract type-domains: are generic domains;
  - enum domains.
- concrete domains: subset of concrete domains.

## Mapping of AsmetaL domains

Are supported **only finite domains** and whose type is available in NuSMV:

- *Boolean*;
- *enum domain*;
- concrete domains of *Integer* and *Natural*.

# Mapping functions

---

## AsmetaL Model

```
asm arity2and3
import ./StandardLibrary

signature:
  domain SubDom subsetof Integer
  enum domain EnumDom = {AA | BB}
  dynamic controlled foo2:
    Prod( Boolean, EnumDom) -> SubDom
  dynamic controlled foo3:
    Prod( SubDom, EnumDom, SubDom) -> Boolean

definitions:
  domain SubDom = {1..2}

  main rule r_Main =
    skip
```

## NuSMV Model

```
MODULE main
  VAR
    foo2_FALSE_AA: { -2147483647, 1, 2 };
    foo2_FALSE_BB: { -2147483647, 1, 2 };
    foo2_TRUE_AA: { -2147483647, 1, 2 };
    foo2_TRUE_BB: { -2147483647, 1, 2 };
    foo3_1_AA_1: boolean;
    foo3_1_AA_2: boolean;
    foo3_1_BB_1: boolean;
    foo3_1_BB_2: boolean;
    foo3_2_AA_1: boolean;
    foo3_2_AA_2: boolean;
    foo3_2_BB_1: boolean;
    foo3_2_BB_2: boolean;
```

# Limitations

---

- Limited domains
- No equivalence between ASM and NuSVM model due to the Boolean undef
- All functions mapped to variables
  - Too many variables may compromise the NuSMV performance

# CTL/LTL library

---

## CTL properties in NuSMV

**SPEC**  $p_{CTL}$

where  $p_{CTL}$  is a CTL formula.

## LTL properties in NuSMV

**LTLSPEC**  $p_{LTL}$

where  $p_{LTL}$  is a LTL formula.

## CTL and LTL operators in AsmetaL

In order to write CTL and LTL formulas in AsmetaL, we have created the libraries *CTLlibrary.asm* and *LTLlibrary.asm* where, for each CTL and LTL operator, an equivalent function is declared.

# CTLlibrary.asm

---

## Mapping of CTL operators into CTL functions

<b>NuSMV CTL operator</b>	<b>AsmetaL CTL function</b>
EG p	static eg: Boolean $\rightarrow$ Boolean
EX p	static ex: Boolean $\rightarrow$ Boolean
EF p	static ef: Boolean $\rightarrow$ Boolean
AG p	static ag: Boolean $\rightarrow$ Boolean
AX p	static ax: Boolean $\rightarrow$ Boolean
AF p	static af: Boolean $\rightarrow$ Boolean
E[p U q]	static e: Prod(Boolean, Boolean) $\rightarrow$ Boolean
A[p U q]	static a: Prod(Boolean, Boolean) $\rightarrow$ Boolean

# LTLlibrary.asm

## Mapping of LTL operators into LTL functions

NuSMV LTL operator	AsmetaL LTL function
X p	static x: Boolean $\rightarrow$ Boolean
G p	static g: Boolean $\rightarrow$ Boolean
F p	static f: Boolean $\rightarrow$ Boolean
p U q	static u: Prod(Boolean, Boolean) $\rightarrow$ Boolean
p V q	static v: Prod(Boolean, Boolean) $\rightarrow$ Boolean
Y p	static y: Boolean $\rightarrow$ Boolean
Z p	static z: Boolean $\rightarrow$ Boolean
H p	static h: Boolean $\rightarrow$ Boolean
O p	static o: Boolean $\rightarrow$ Boolean
p S q	static s: Prod(Boolean, Boolean) $\rightarrow$ Boolean
p T q	static t: Prod(Boolean, Boolean) $\rightarrow$ Boolean



# Mapping example

```
asm ctIExample
import ./StandardLibrary
import ./CTLLibrary

signature:
  dynamic controlled fooA: Boolean
  dynamic controlled fooB: Boolean
  dynamic monitored mon: Boolean

definitions:
  CTLSPEC ag(fooA iff ax(not(fooA))) //true
  CTLSPEC ag(not(fooA) iff ax(fooA)) //true
  //false. Gives counterexample.
  CTLSPEC not(ef(fooA != fooB))

  main rule r_Main =
    par
      fooA := not(fooA)
      if(mon) then
        fooB := not(fooB)
      endif
    endpar

default init s0:
  function fooA = true
  function fooB = true
```

```
MODULE main
VAR
  fooA: boolean; —controlled
  fooB: boolean; —controlled
  mon: boolean; —monitored
ASSIGN
  init(fooA) := TRUE;
  init(fooB) := TRUE;
  next(fooA) := !(fooA);
  next(fooB) :=
    case
      (mon): !(fooB);
      TRUE: fooB;
    esac;

—CTL properties
CTLSPEC AG(fooA <=> AX(!(fooA)));
CTLSPEC AG(!(fooA) <=> AX(fooA));
CTLSPEC !(EF(fooA != fooB));
```

# Execution

```
[user@localhost asmetasmv]$ NuSMV ctlExample.smv
*** This is NuSMV 2.5.2 (compiled on Sat Oct 30 12:18:33 UTC 2010)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <nusmv-users@fbk.eu>

-- specification AG (fooA <-> AX !fooA) is true
-- specification AG (!fooA <-> AX fooA) is true
-- specification !(EF fooA != fooB) is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  fooA = TRUE
  fooB = TRUE
  mon = FALSE
-> State: 1.2 <-
  fooA = FALSE
```

# Mapping invariants

## AsmetaL Model

```
asm ag
import ./StandardLibrary
import ./CTLlibrary

signature:
  dynamic controlled fooA: Boolean
  dynamic controlled fooB: Boolean

definitions:

  //invariant for simulation with AsmetaS
  invariant over fooA, fooB: fooA != fooB

  //property for NuSMV
  CTLSPEC ag(fooA != fooB)

  main rule r_Main =
    par
      fooA := not(fooA)
      fooB := not(fooB)
    endpar

  default init s0:
    function fooA = true
    function fooB = false
```

## NuSMV Model

```
MODULE main
  VAR
    fooA: boolean; —controlled
    fooB: boolean; —controlled
  ASSIGN
    init(fooA) := TRUE;
    init(fooB) := FALSE;
    next(fooA) := !(fooA);
    next(fooB) := !(fooB);
  —CTL properties
  CTLSPEC AG(fooA != fooB);
```

## Execution of the NuSMV Model

```
[user@localhost code]$ NuSMV ag.smv
*** This is NuSMV 2.4.1 (compiled on Sat Jun 13 10:57:42 UTC 2009)
*** For more information on NuSMV see <http://nusmv.irst.itc.it>
*** or email to <nusmv-users@irst.itc.it>.
*** Please report bugs to <nusmv@irst.itc.it>.

-- specification AG fooA != fooB is true
```

# Verification properties

---

- Reachability property
  - Safety property
  - Liveness property
  - Absence of deadlock
  - Other property to guarantee correctness of specification
- 
- They have precise specification patterns

# Reachability Property

---

- Reachability: “exists a future state satisfying a property  $\varphi$ ”
  - $\varphi$  is called “present tense formula” (no temporal operators inside)
  - For instance, “A process will enter its critical section”
- In CTL  $EF\varphi$ 
  - $EF$  critical1

# Safety property

---

- Safety: “Nothing bad will happen”.
  - For instance, “Only one process is in its critical section at any time”.
- In CTL  $AG\phi$ 
  - (with 2 processes only):
  - $AG(\neg(\text{critical1} \wedge \text{critical2}))$

# Liveness property

---

- Liveness: “Something good will eventually happen”.
  - For instance: “Whenever any process requests to enter its critical section, it will eventually be permitted to do so”.
- In CTL  $AG + AF$  or  $AG + EF$ 
  - $AG(\text{request} \rightarrow AF(\text{critical}))$

# Deadlock absence

---

- In CTL  $AG\ EX\ true$ 
  - whatever the status reached ( $AG$ ), there is a status immediately successor ( $EX\ true$ )
  - No deadlock



# Examples

---

- Ferryman
  - Counter example for the well-known path
- Tic tac toe
- Children
- SwapBoard