PennState

CSE 597: Security of Emerging Technologies

Module: Formal Verification (Part 2)

Prof. Syed Rafiul Hussain

Systems and Network Security (SyNSec) Research Group

Department of Computer Science and Engineering

The Pennsylvania State University

Verification vs. Falsification

- An automated verification tool
  - can report that the system is verified (with a proof);
  - or that the system was not verified.
- When the system was not verified, it would be helpful to explain why
  - Model checkers can output an error counterexample: a concrete execution scenario that demonstrates the error.
- Can view a model checker as a falsification tool –
  - The main goal is to find bugs
- So what can we verify or falsify?

Temporal Properties

- Temporal Property
  - A property with time-related operators such as "invariant" or "eventually"

- Invariant(p)
  - is true in a state if property p is true in every state on all execution paths starting at that state
  - G, AG, ("globally" or "box" or "forall")

- Eventually(p)
  - is true in a state if property p is true at some state on every execution path starting from that state F, AF, ◊ ("future" or "diamond" or "exists")

An Example Concurrent Program

PennState

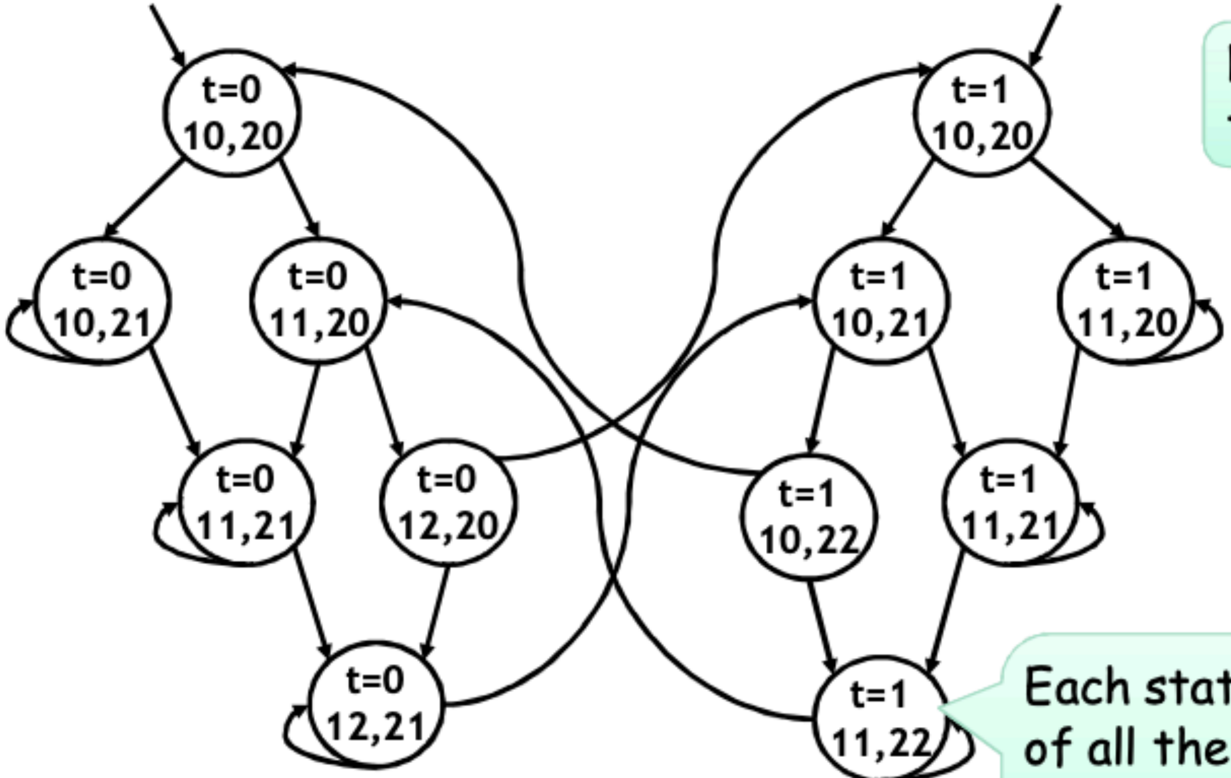
- A simple concurrent mutual exclusion program
- Two processes execute asynchronously
- There is a shared variable turn
- Two processes use the shared variable to ensure that they are not in the critical section at the same time
- Can be viewed as a “fundamental” program: any bigger concurrent one would include this one

```
10: while (true){
11:     wait(turn == 0);
        // critical section
12:     work(); turn = 1;
13: }

// concurrently with

20: while (true) {
21:     wait(turn == 1);
        // critical section
22: work(); turn = 0;
23: }
```

Reachable States of the Example Program



Next: formalize this intuition …

Each state is a valuation of all the variables: `turn` and the two program counters for two processes

Analyzed System is a Transition System

PennState

- Labeled transition system

  $T = (S, I, R, L)$ –

  S = Set of states // standard FSM

  $I \subseteq S$ = Set of initial states // standard FSM

  $R \subseteq S \times S$ = Transition relation // standard FSM

  $L: S \rightarrow 2^{AP}$ = Labeling function // this is new!

- AP: Set of atomic propositions (e.g., "x=5"∈AP)
  - Atomic propositions capture basic properties
  - For software, atomic props depend on variable values
  - The labeling function labels each state with the set of propositions true in that state

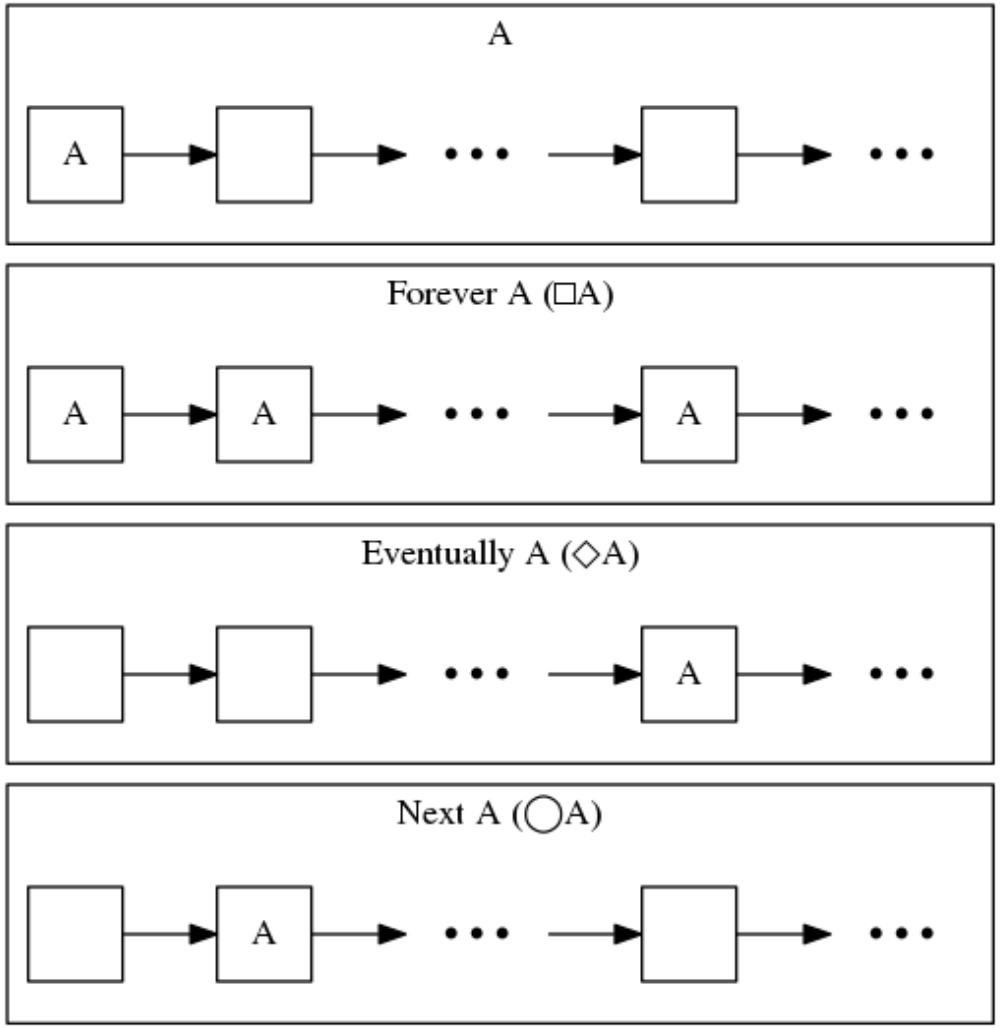
Example Properties of the Program

- "In all the reachable states (configurations) of the system, the two processes are never in the critical section at the same time"
  - "pc1=12", "pc2=22" are atomic properties for being in the critical section
  - Invariant ($\neg$ (PC1=12 $\wedge$ PC2 = 22)

- "Eventually the first process enters the critical section
  - Eventually (PC1 = 12)

Temporal Logics

- There are four basic temporal operators:
- X p Next p, p holds in the next state
- G p: Globally p, p holds in every state, p is an invariant
- F p: Future p, p will hold in a future state, p holds eventually
- p U q: p Until q, assertion p will hold until q holds
- Precise meaning of these temporal operators is defined on execution paths



Execution Paths

- A path in a transition system is an infinite sequence of states
  - (s0 , s1 , s2 , ...), such that ∀i≥0. (si , si+1) ∈ R
- A path (s0 ,s1 ,s2 ,...) is an execution path if s0 ∈ I
- Given a path x = (s0 , s1 , s2 , ...)
  - h_i denotes the ith state: s_i
  - h^i denotes the i-th suffix: (s_i , s_{i+1}, s_{i+2}, ...)
  - In some temporal logics one can quantify paths starting from a state using path quantifiers
    - A : for all paths
    - E : there exists a path

Paths and Predicates

- We write

$$h \models p$$

"the path x makes the predicate p true"

  - h is a path in a transition system
  - p is a temporal logic predicate •

- Example: A h. $h \models G\ (\neg(pc1=12 \land pc2=22))$

Linear Temporal Logic (LTL)

- LTL properties are constructed from atomic propositions in AP; logical operators $\wedge$, $\vee$, $\neg$ and temporal operators X, G, F, U.
- The semantics of LTL is defined on **paths**
- Given a path h: $h \models p$

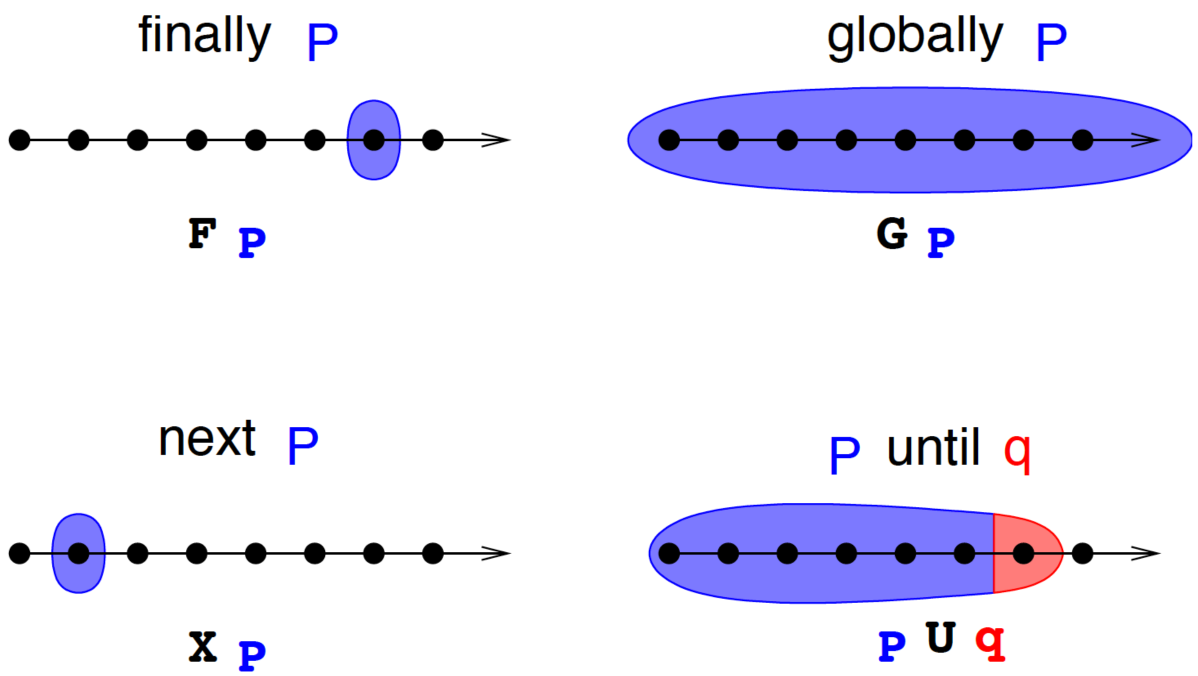
$h \models p$ iff $L(h^0, ap)$ atomic prop

$h \models X\, p$ iff $h^1\, p$ next

$h \models F\, p$ iff $\exists i \geq 0.\ h^i\, p$ future

$h \models G\, p$ iff $\forall i \geq 0.\ h^i\, p$ globally

$h \models p\, U\, q$ iff $\exists i \geq 0.\ h^i\, q$ and $\forall j < i.\ h^j\, p$ until

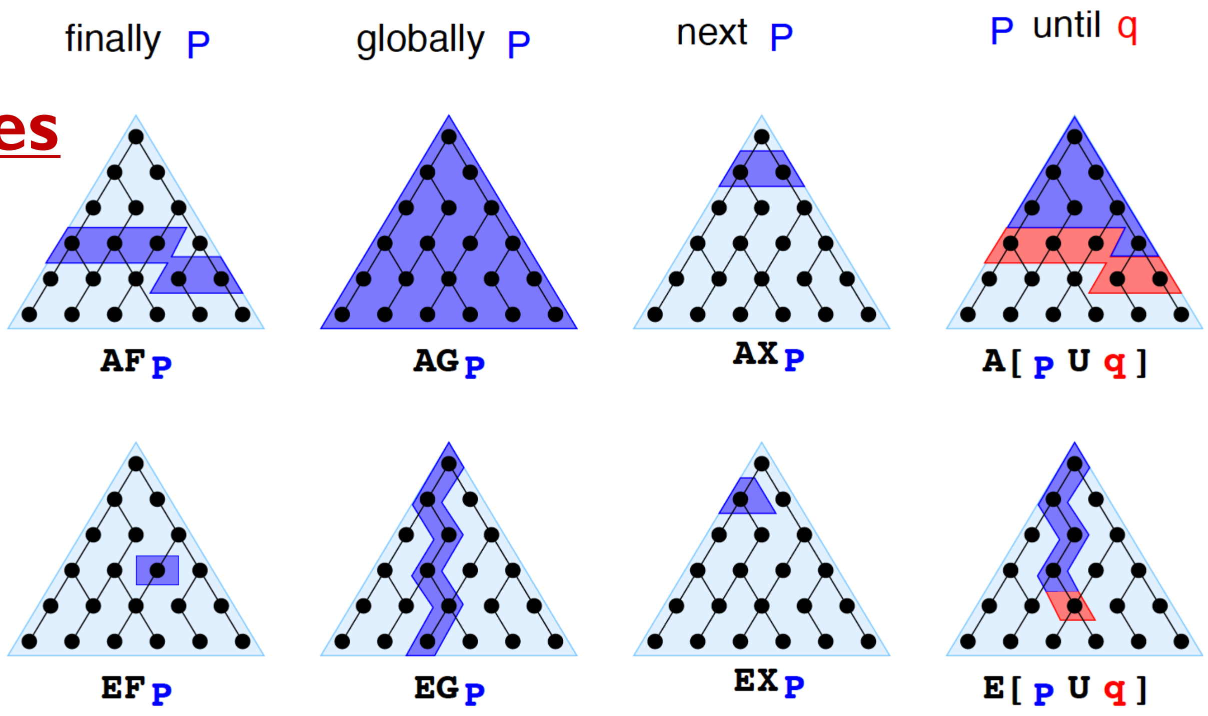


Satisfying Linear Time Logic

- Given a transition system T = (S, I, R, L) and an LTL property p, T satisfies p if all paths starting from all initial states I satisfy p

Computation Tree Logic

- In CTL, temporal properties use path quantifiers:
  - A : for all paths, E : there exists a path
- The semantics of CTL is defined on **states**
- Given a state s
- s $\models$ ap iff L(s, ap)
- $s_0 \models$ EX p iff $\exists$ a path $(s_0, s_1, s_2, \ldots)$. $s_1 \models p$
- $s_0 \models$ AX p iff $\forall$ paths $(s_0, s_1, s_2, \ldots)$. $s_1 \models p$
- $s_0 \models$ EG p iff $\exists$ a path $(s_0, s_1, s_2, \ldots)$. $\forall i \geq 0.\ s_i \models p$
- $s_0 \models$ AG p iff $\forall$ paths $(s_0, s_1, s_2, \ldots)$. $\forall i \geq 0.\ s_i \models p$



Examples of CTL formulas

PennState

- EF φ
  - It is possible to get to a state where φ is true
- AG AF enabled
  - A certain process is enabled infinitely often on every computation path
- AG (requested → AF acknowledged)
  - for any state, if a request ocurs, then it will eventually be acknowledged
- AG (φ → E[φ U ψ])
  - for any state, if φ holds, then there is a future where ψ eventually holds, and φ holds for all points in between
- AG (φ → EG ψ)
  - for any state, if φ holds then there is a future where ψ always holds

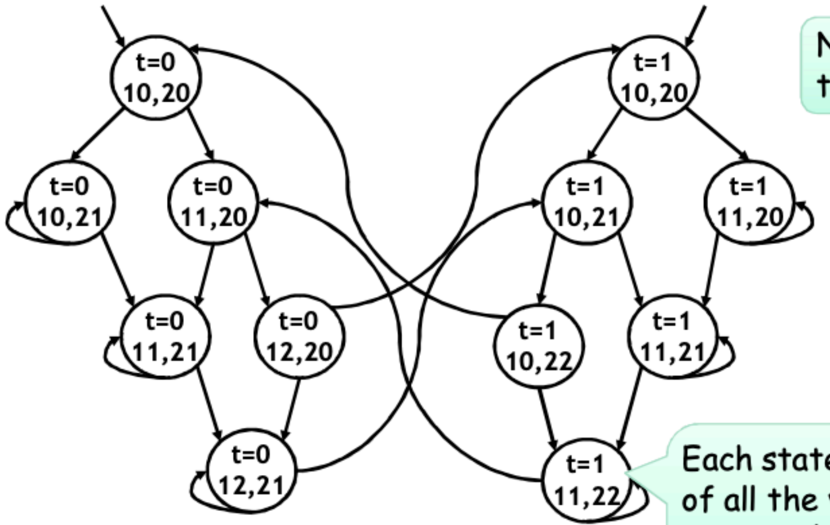
Linear vs. Branching Time

- LTL is a linear time logic
  - When determining if a path satisfies an LTL formula, we are only concerned with a single path
- CTL is a branching time logic
  - When determining if a state satisfies a CTL, formula we are concerned with multiple paths
  - In CTL the computation is instead viewed as a computation tree which contains all the paths
- The expressive powers of CTL and LTL are incomparable incomparable
  - LTL $\subseteq$ CTL*, CTL $\subseteq$ CTL*
  - Basic temporal properties can be expressed in both logics
  - Not in this lecture, sorry! (Take a class on Modal Logics)

LTL vs. CTL

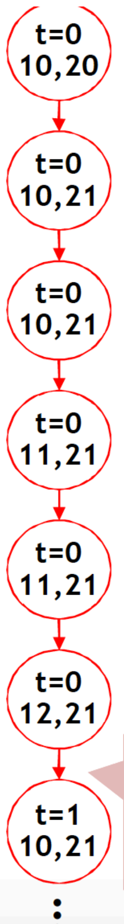
- Some LTL formulae cannot be translated into CTL formaulae.
  - FG s **-** This formula denotes the **property of stability** : in each execution of the program, s will finally be true until the end of the program (or forever if the program never stops).
  - CTL can only provide a formula that is too strict (AF AG s) or too permissive (AF EG s).
  - (AF EG s) is clearly wrong. It is not so straightforward for the first.
  - But AF AG s is erroneous. Consider a system that loops on A1, can go from A1 to B and then will go to A2 on the next move. Then the system will stay in A2 state forever. Then "the system will finally stay in a A state" is a property of the type FG*s*. It is obvious that this property holds on the system. However, AF AG s cannot capture this property since the opposite is true.

Linear vs. Branching Time



Next: formalize this intuition …

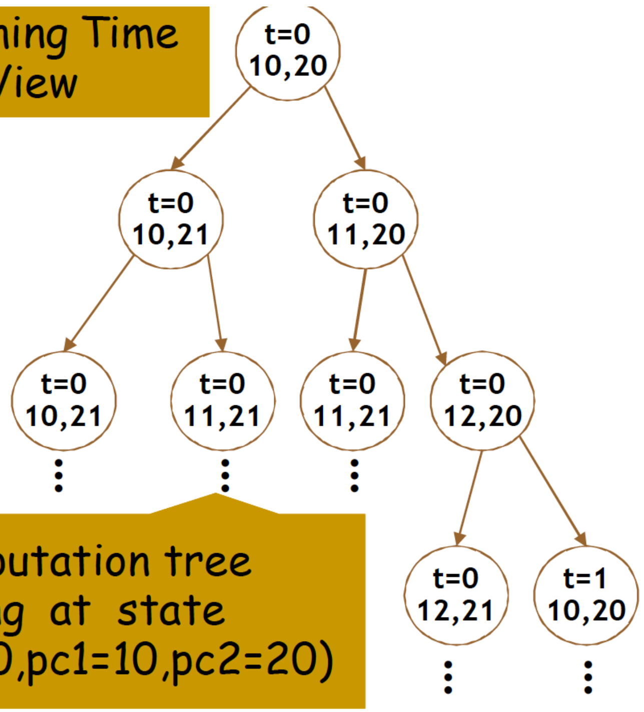
Each state is a valuation of all the variables: `turn` and the two program counters for two processes



Linear Time View

One path starting at state (turn=0,pc1=10,pc2=20)

Branching Time View



A computation tree starting at state (turn=0,pc1=10,pc2=20)

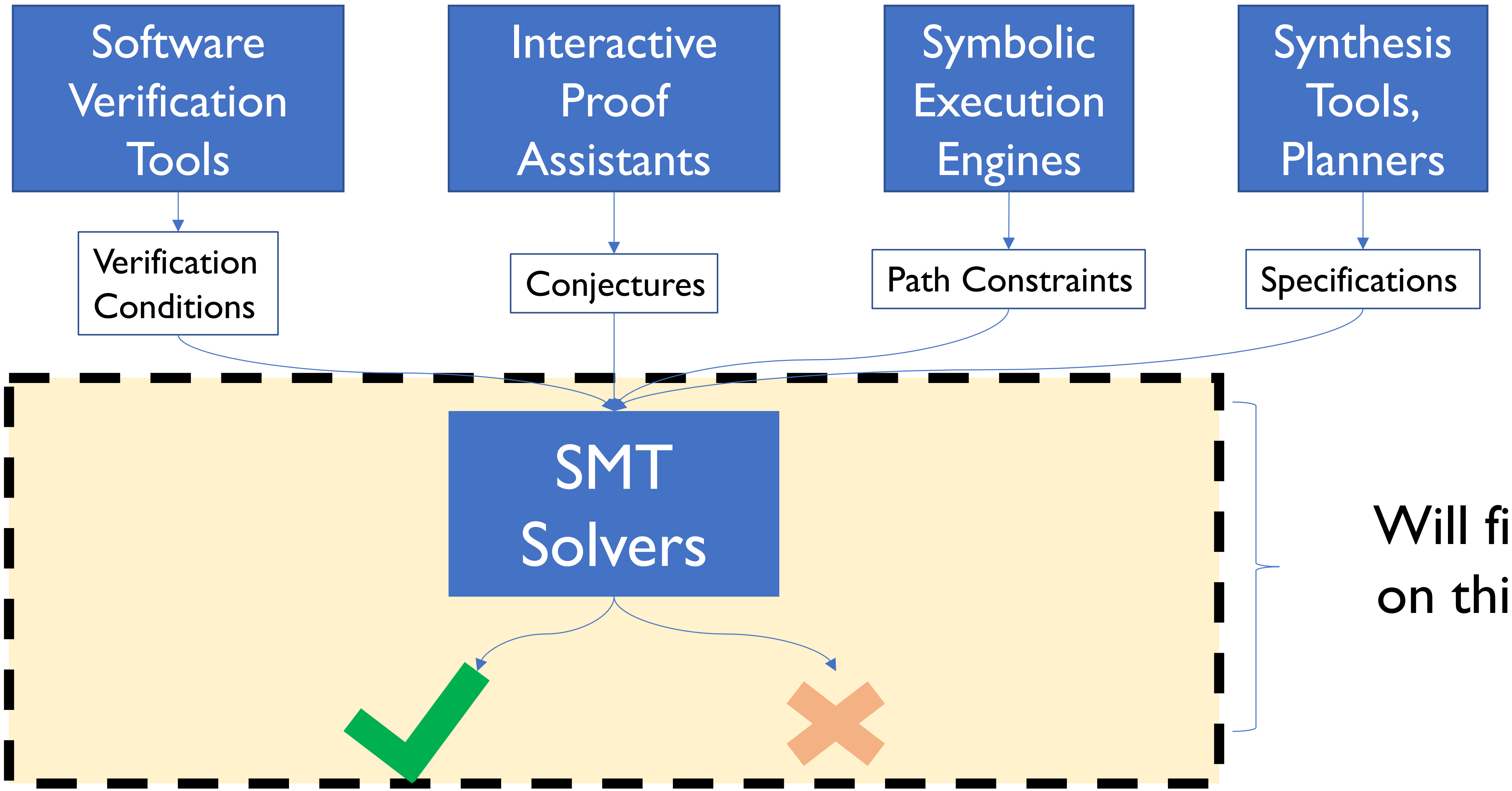
State Space Explosion

- The complexity of model checking increases linearly with respect to the size of the transition system (|S| + |R|)
- However, the size of the transition system (|S| + |R|) is exponential in the
- number of variables and number of concurrent processes
- This exponential increase in the state space is called the state space explosion
  - Dealing with it is one of the major challenges in model checking research

Symbolic Model Checking

- Symbolic model checking represents state sets and the transition relation as Boolean logic formulas
  - Fixed point computations manipulate sets of states rather than individual states
- Use an efficient data structure for manipulation of Boolean logic formulas
  - Binary Decision Diagrams (BDDs)
- SMV (Symbolic Model Verifier) was the first CTL model checker to use BDDs

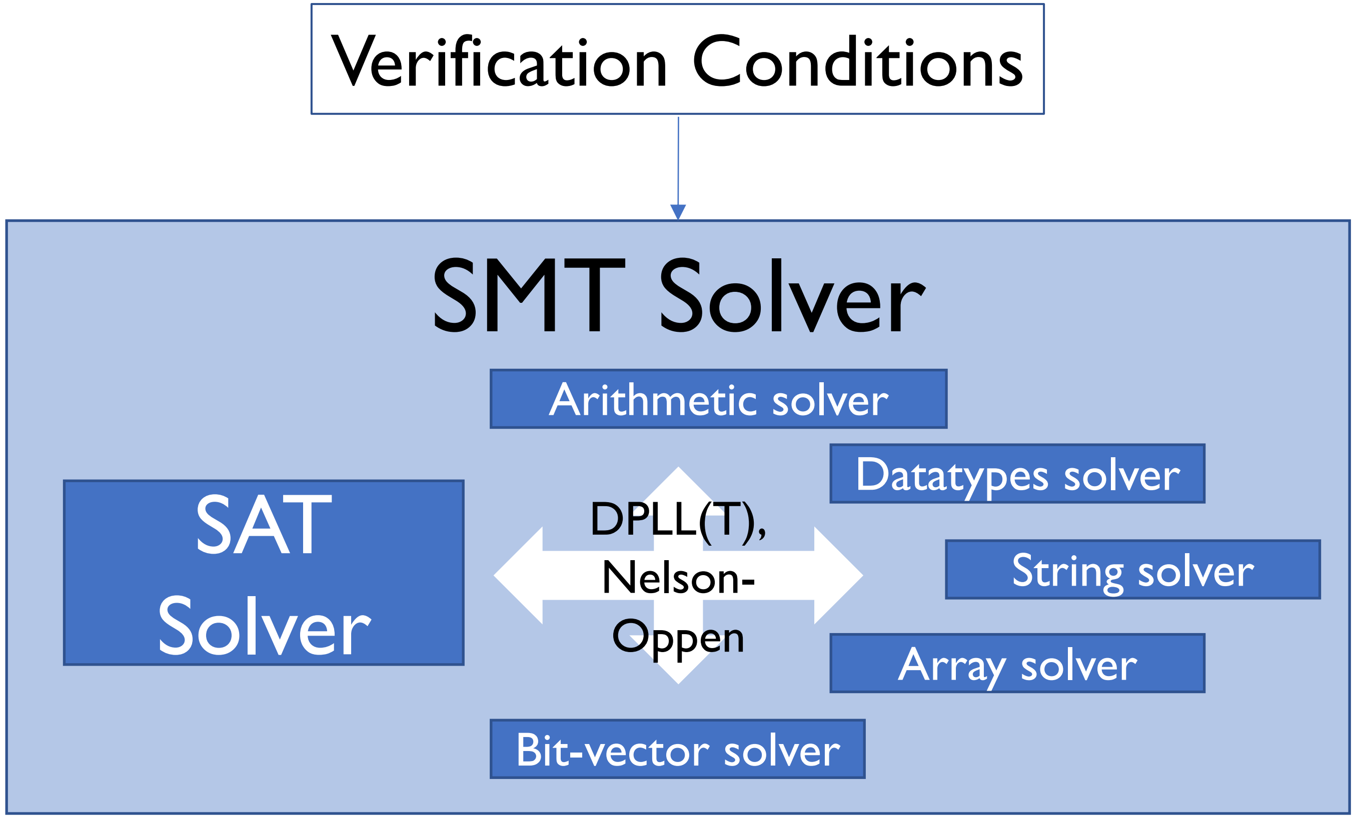
Satisfiability Modulo Theories (SMT) Solvers



Will first focus on this portion

SMT solvers

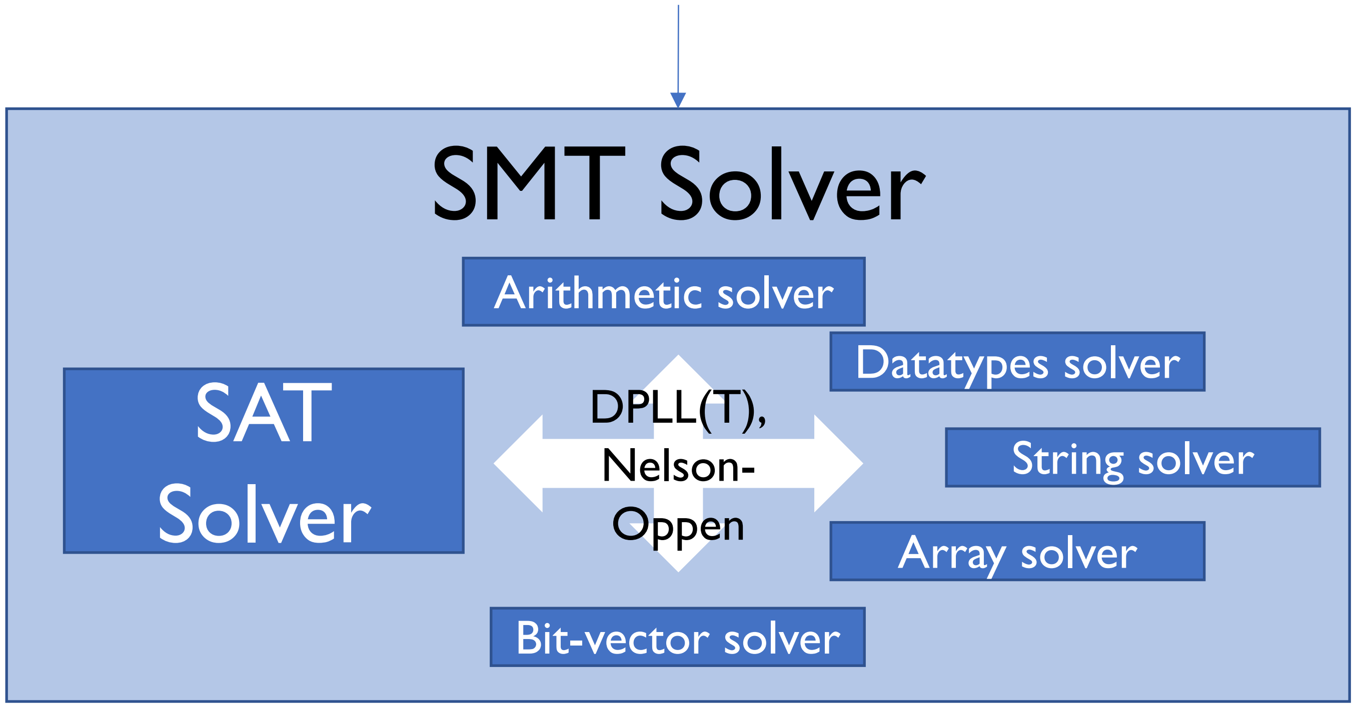
- Efficient tools for satisfiability and satisfiability *modulo theories*



SMT solvers

- Efficient tools for satisfiability and satisfiability *modulo theories*

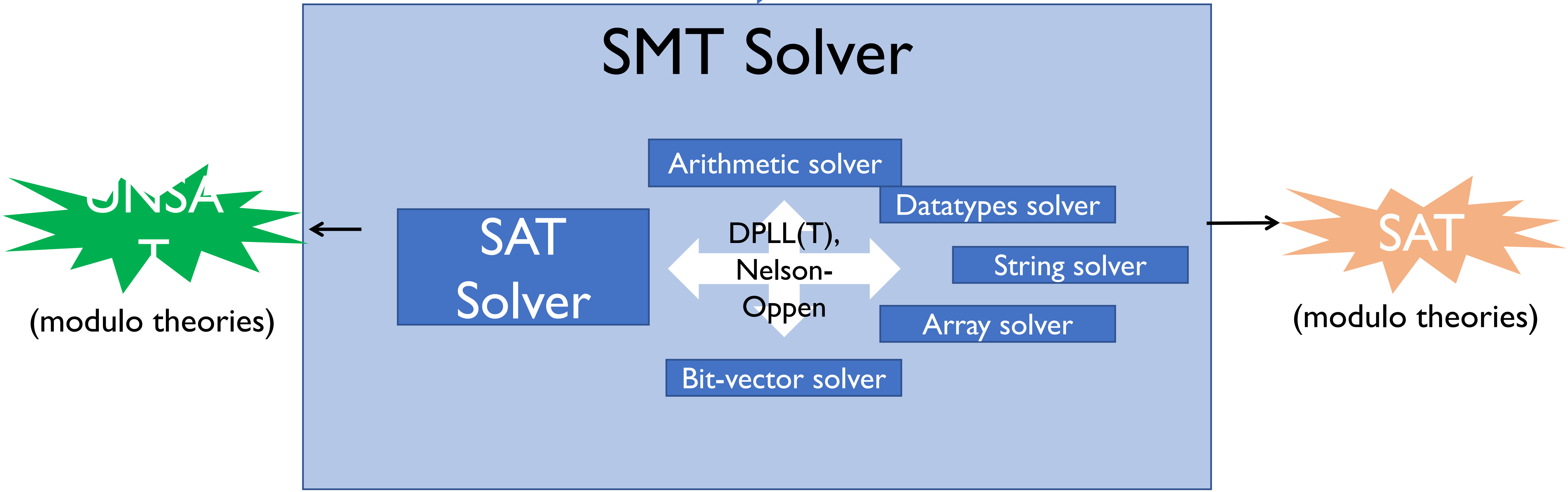
$$(A[x]+B[x]>0 \vee x+y>0) \wedge (cons(\text{“abc”},d_1)\neq d_2 \vee x<0)$$



SMT solvers

- Efficient tools for satisfiability and satisfiability *modulo theories*

$$(A[x]+B[x]>0 \vee x+y>0) \wedge (\text{cons}(\text{“abc”},d_1)\neq d_2 \vee x<0)$$

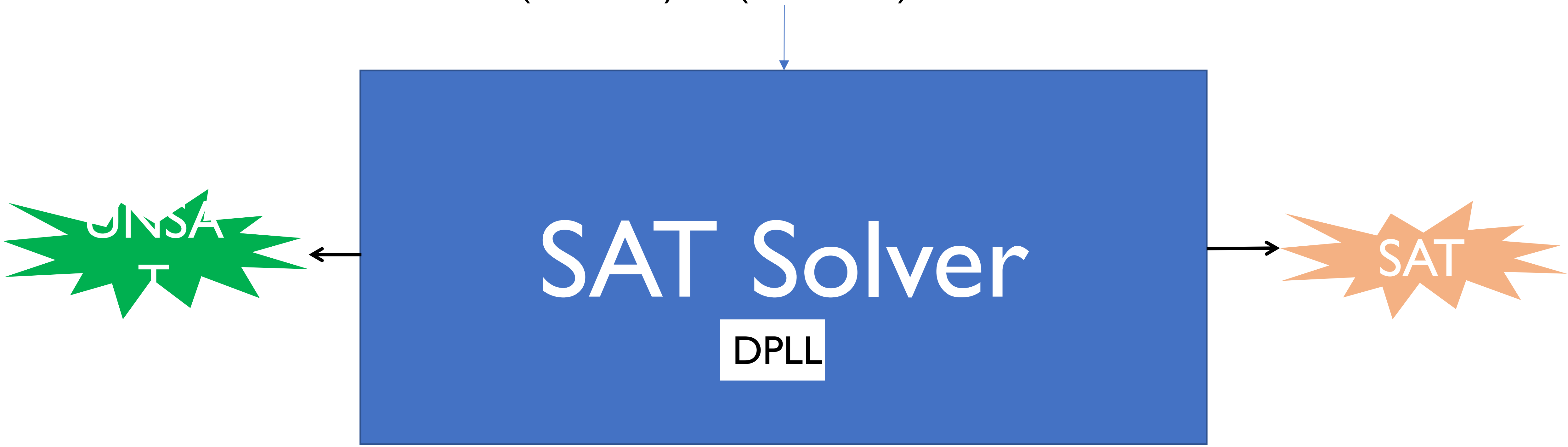


…but first : SAT solvers

PennState

- Efficient tools for *satisfiability*

$$(A \lor B) \land (C \lor D) \land \neg B$$

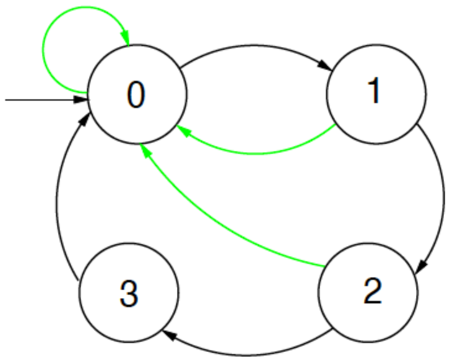


NuXmv Example: Modulo 4 counter with reset

```
MODULE main
VAR  b0     : boolean; b1    : boolean;
     reset : boolean;
ASSIGN
  init(b0) := FALSE;
  next(b0) := case  reset  : FALSE;
                    !reset : !b0;
              esac;
  init(b1) := FALSE;
  next(b1) := case  reset : FALSE;
                    TRUE  : ((!b0 & b1) |
                            (b0 & !b1));
              esac;
DEFINE out := toint(b0) + 2*toint(b1);

INVARSPEC out < 2
```

- recall:



- The invariant is false

```
nuXmv > read_model -i counter4reset.smv;
nuXmv > go; check_invar
-- invariant out < 2  is false
...
  -> State: 1.1 <-
    b0 = FALSE
    b1 = FALSE
    reset = FALSE
    out = 0
  -> State: 1.2 <-
    b0 = TRUE
    out = 1
  -> State: 1.3 <-
    b0 = FALSE
    b1 = TRUE
    out = 2
```

LTL Specifications

- Specications Examples:
  - A state in which out = 3 is eventually reached
  - LTLSPEC F out = 3

- Condition out = 0 holds until reset becomes false
  - LTLSPEC (out = 0) U (!reset)

- Every time a state with out = 2 is reached, a state with out = 3 is reached afterward
  - LTLSPEC G (out = 2 -> F out = 3)

LTL Specifications

All the previous specifications are false:

```
NuSMV > check_ltlspec
-- specification  F out = 3 is false ...
-- loop starts here --
-> State 1.1 <-
    b0 = FALSE
    b1 = FALSE
    reset = TRUE
    out = 0
-> State 1.2 <-
-- specification (out = 0 U (!reset)) is false ...
-- loop starts here --
-> State 2.1 <-
    b0 = FALSE
    b1 = FALSE
    reset = TRUE
    out = 0
-> State 2.2 <-
-- specification  G (out = 2 ->  F out = 3) is false ...
```

Q: why?

Model Programs in NuXmv

```
    void main() {
        ... // initialization of a and b
l1:     while (a!=b) {
l2:         if (a>b)
l3:             a=a-b;
            else
l4:             b=b-a;
        }
l5:     ... // GCD=a=b
    }
```

```
MODULE main()
VAR  a: 0..100;  b: 0..100;
  pc: {l1,l2,l3,l4,l5};
ASSIGN
  init(pc):=l1;
  next(pc):=
    case
      pc=l1 & a!=b    : l2;
      pc=l1 & a=b     : l5;
      pc=l2 & a>b     : l3;
      pc=l2 & a<=b    : l4;
      pc=l3 | pc=l4   : l1;
      pc=l5           : l5;
    esac;
```

```
next(a):=
  case
    pc=l3 & a > b: a - b;
    TRUE: a;
  esac;

next(b):=
  case
    pc=l4 & b >= a: b-a;
    TRUE: b;
  esac;
```

Takeaways

PennState

- A system can be modeled as a Labeled Transition System (LTS).
- Based on the expressiveness of the property, we use LTL or CTL property.
- Need to take care of state explosion problem with different types abstractions.
- Model checking is useful for testing many safety critical systems.

Thanks

Thanks to Bor-Yuh Evan Chang, Andrew Reynolds, and Patrick Trentin for some slides.