Model Checking Software Specifications

Tevfik Bultan

Department of Computer Science
University of California, Santa Barbara

Office: 2123 Engineering I
bultan@cs.ucsb.edu
http://www.cs.ucsb.edu/~bultan/

January 25, 1999

Model Checking

- What is a model checker?

A model checker is a program that checks if a system satisfies a property.

- What type of system?

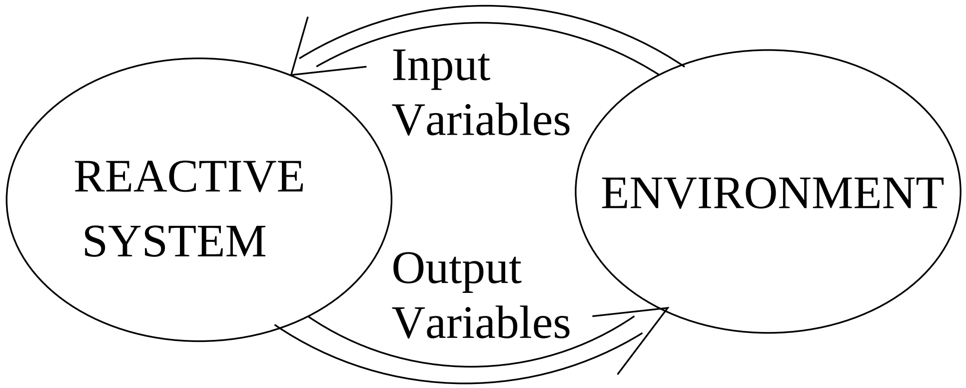
Reactive Systems: Systems which continuously interact with their environment.

- What type of property?

Temporal properties: Such as "A condition holds throughout all possible executions of a system" or "Eventually a condition becomes true".

- You can learn a lot about model checking if you take my course next quarter (290I: Computer aided verification).

Reactive Systems

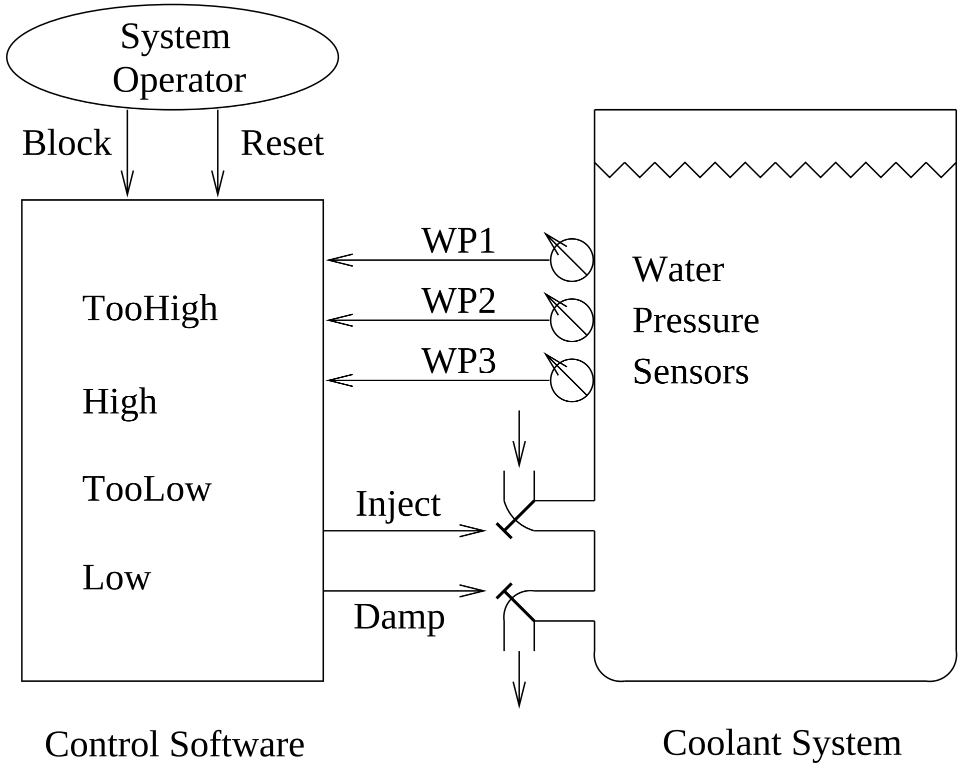


- Systems which continuously interact with their environment.
- Examples: Protocols, hardware systems, software specifications . . .
- Properties of reactive systems involve *invariance* and *eventuality*.

Outline

- Two Specifications
- Model Checking Technique
  - Symbolic Representations
  - Composite Models
- Checking Software Specifications
- Future Directions

Safety Injection System



- Original SCR specification [Courtois and Parnas 93].
- Finite state abstraction verified [Bharadwaj and Heitmeyer 97].
- Full specification verified [Bultan, Gerber and League 98].

Safety Injection System: SCR Specification

Mode Class Pressure

Old Mode	Event	New Mode
–	@T(CTLow)	TooLow
–	@T(CLow)	Low
–	@T(CHigh)	High
–	@T(CTHigh)	TooHigh

Term Overridden

Mode	Events	
High	False	@T(InMode)
TooLow, Low, TooHigh	@T(Block=On) WHEN Reset=Off	@T(InMode) OR @T(Reset=On)
Overridden	True	False

Condition Inject

Mode	Conditions	
Low, High, TooHigh	True	False
TooLow	Overridden	NOT Overridden
Inject	Off	On

Condition Damp

Mode	Conditions	
TooLow, Low, High	True	False
TooHigh	Overridden	NOT Overridden
Damp	Off	On

$$\text{CTLow} \stackrel{\text{def}}{=} (\texttt{WP1},\ \texttt{WP2} < \texttt{low})\ \text{OR}\ (\texttt{WP1},\ \texttt{WP3} < \texttt{low})\ \text{OR}\ (\texttt{WP2},\ \texttt{WP3} < \texttt{low})$$

$$\text{CLow} \stackrel{\text{def}}{=} (\texttt{low} \leq \texttt{WP1},\ \texttt{WP2} < \texttt{high})\ \text{OR}\ (\texttt{low} \leq \texttt{WP1},\ \texttt{WP3} < \texttt{high})\ \text{OR}\ (\texttt{low} \leq \texttt{WP2},\ \texttt{WP3} < \texttt{high})$$

$$\text{CHigh} \stackrel{\text{def}}{=} (\texttt{high} \leq \texttt{WP1},\ \texttt{WP2} < \texttt{toohigh})\ \text{OR}\ (\texttt{high} \leq \texttt{WP1},\ \texttt{WP3} < \texttt{toohigh})\ \text{OR}\ (\texttt{high} \leq \texttt{WP2},\ \texttt{WP3} < \texttt{toohigh})$$

$$\text{CTHigh} \stackrel{\text{def}}{=} (\texttt{toohigh} \leq \texttt{WP1},\ \texttt{WP2})\ \text{OR}\ (\texttt{toohigh} \leq \texttt{WP1},\ \texttt{WP3})\ \text{OR}\ (\texttt{toohigh} \leq \texttt{WP2},\ \texttt{WP3})$$

$$\texttt{min} \leq \texttt{WP1},\ \texttt{WP2},\ \texttt{WP3} \leq \texttt{max}\ \ \text{AND}\ \ \texttt{min} < \texttt{low} < \texttt{high} < \texttt{toohigh} < \texttt{max}$$

Example Properties

If the reset switch is on and the water pressure is too low, then safety injection should be active:

$$\textsc{Invariant}(\ Reset\ \wedge\ TLow\ \implies\ Inject\ )$$

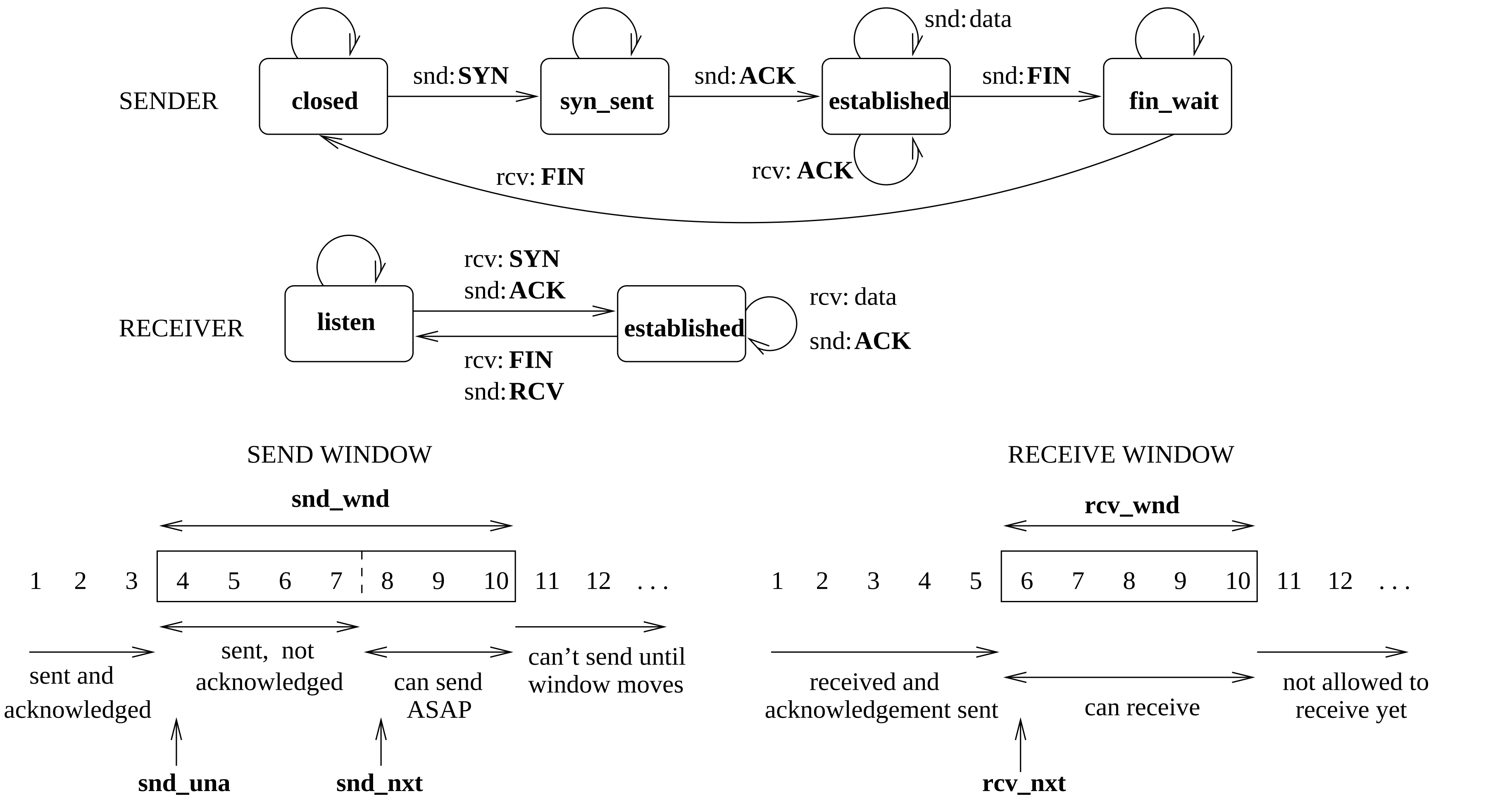
If two of the water pressure sensors show that water pressure is high, then the system is in either `High` or `TooHigh` mode:

$$\textsc{Invariant}(\ high \leq wp1, wp2\ \implies\ (High\ \vee\ TooHigh)\ )$$

If the safety injection stays active, then eventually the water pressure will be OK:

$$\textsc{Invariant}(\ Inject\ \textsc{Until}\ low \leq wp1\ )$$

A Transport Protocol



Example Properties

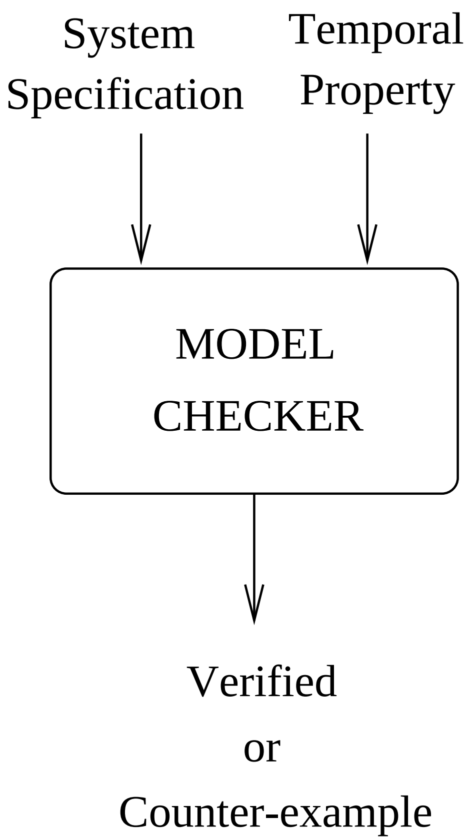
During data transfer, the acknowledged messages at the sender can not exceed the received messages at the receiver:

$$\textsc{Invariant}(sender = receiver = \texttt{established} \implies snd\_una \leq rcv\_nxt)$$

During data transfer, the received messages at the receiver can not exceed the sent messages at the receiver:

$$\textsc{Invariant}(sender = receiver = \texttt{established} \implies rcv\_nxt \leq snd\_nxt)$$

Verification vs. Falsification



- Using a model checker one can either *verify* or *falsify* a property.

Model Checking: Goal

Given a system and a property p :

- Either show that all initial states satisfy the temporal property p :

$$\text{set of initial states} \subseteq \text{truth set of } p$$

- Or find an initial state which does not satisfy the property p :

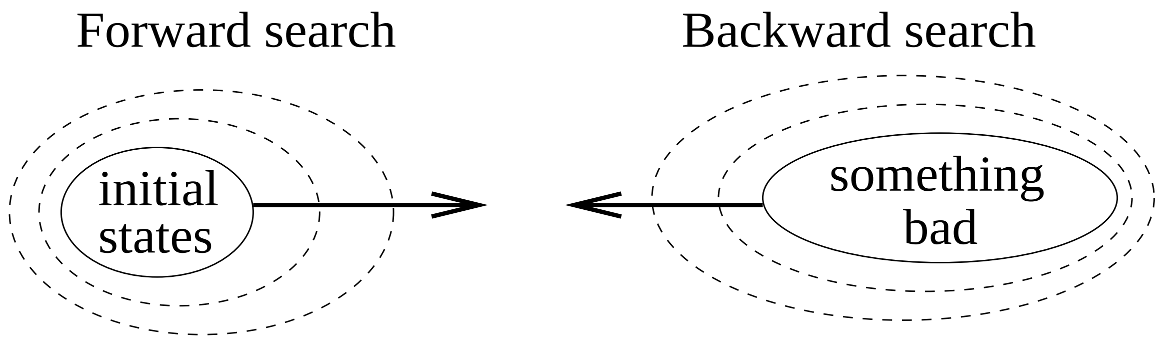
$$\text{a program state} \in \text{set of initial states} \cap \text{truth set of } \neg p$$

GOAL: To find a representation for the truth set of p which will enable us to test these conditions.

Basic Technique: State Exploration

- Given a set of states find the states which are reachable from (*forward* search) or which can reach to (*backward* search) those states.

Example: Invariant(Not(something bad))



Symbolic Representations

Question: How do we represent sets of states during state exploration?

Answer 1) Explicitly, just list them one by one.
Problems: Not feasible for systems with large state spaces.
Remedies: State minimization techniques, partial order techniques, etc.

Answer 2) Use symbolic representations (Symbolic model checking).
Problems: Finding efficient symbolic representations.
Remedies: Binary Decision Diagrams (BDDs), arithmetic constraints, etc.

Symbolic Representations: Required Functionality

- Basic set operations for:
  $p \ \wedge \ q, \quad p \ \vee \ q, \quad \neg p$

```
Symbolic And(Symbolic,Symbolic)
Symbolic Or(Symbolic,Symbolic)
Symbolic Not(Symbolic)
```

- Equivalence checking: $p \equiv q$?

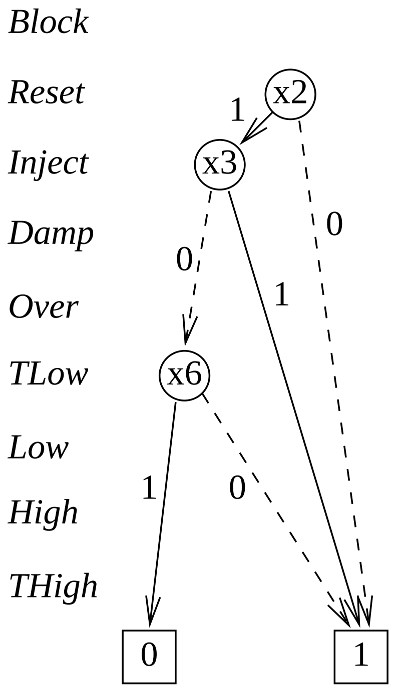
```
Boolean IsSatisfiable(Symbolic)
```

- Precondition function: $\textsc{Pre}(p)$

```
Symbolic Pre(Symbolic)
```

Representing Boolean States: BDDs

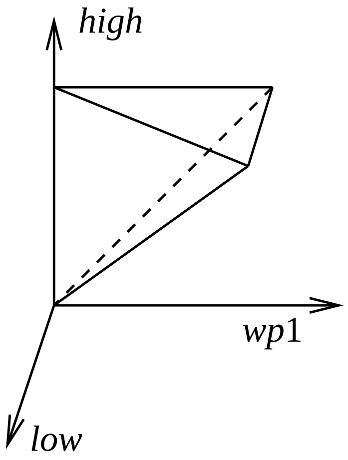
- How can we represent a Boolean set q?
- Variables: $Block, Reset, Inject, Damp, Over, TLow, Low, High, THigh$

Explicit State Enumeration	Boolean Logic	BDD
$\{FFFFFFFFF, FFFFFFFFT,$ $FFFFFFFTF, FFFFFFFTT,$ $FFFFFFTFF, FFFFFFTFT,$ $FFFFFFTTF, \ldots\}$	$Reset \wedge TLow \implies Inject$ $\equiv \neg Reset \vee \neg TLow \vee Inject$	

Representing Integer States: Arithmetic Constraints

- How can we represent an integer set q?
- Variables: $wp1, wp2, wp3, min, max, low, high, toohigh$

Explicit State Enumeration ?	Arithmetic Constraints
$\{(0,0,0,0,0,0,0,0,0), (0,0,0,0,0,0,0,1,0),$ $(1,0,0,0,0,0,0,1,0), (0,0,0,0,0,0,0,2,0),$ $(1,0,0,0,0,0,1,1,0), \ldots$	$0 \leq low \leq wp1 \leq high$



Symbolic Model Checking

In a BDD based model checker:

- Input system is represented by a set of boolean logic formulas.
- State exploration is performed by manipulating boolean logic formulas that represent sets of states.

In an arithmetic constraint based model checker:

- Input system is represented by a set of arithmetic constraints.
- State exploration is performed by manipulating arithmetic constraints that represent sets of states.

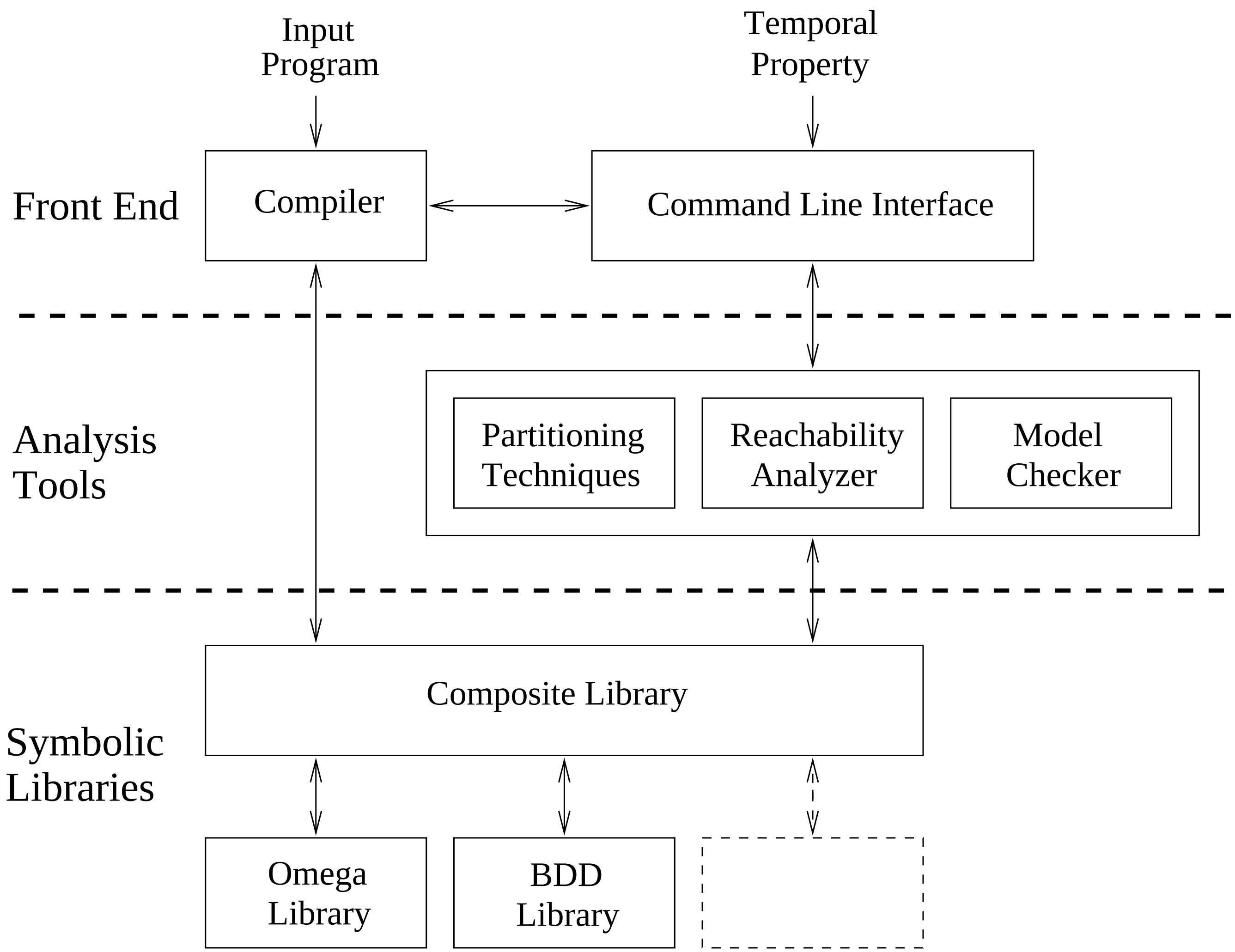
Is One Symbolic Representation Enough?

• BDDs are efficient for representing boolean variables, but they are not very efficient in representing integers. Moreover they cannot represent unbounded variables.

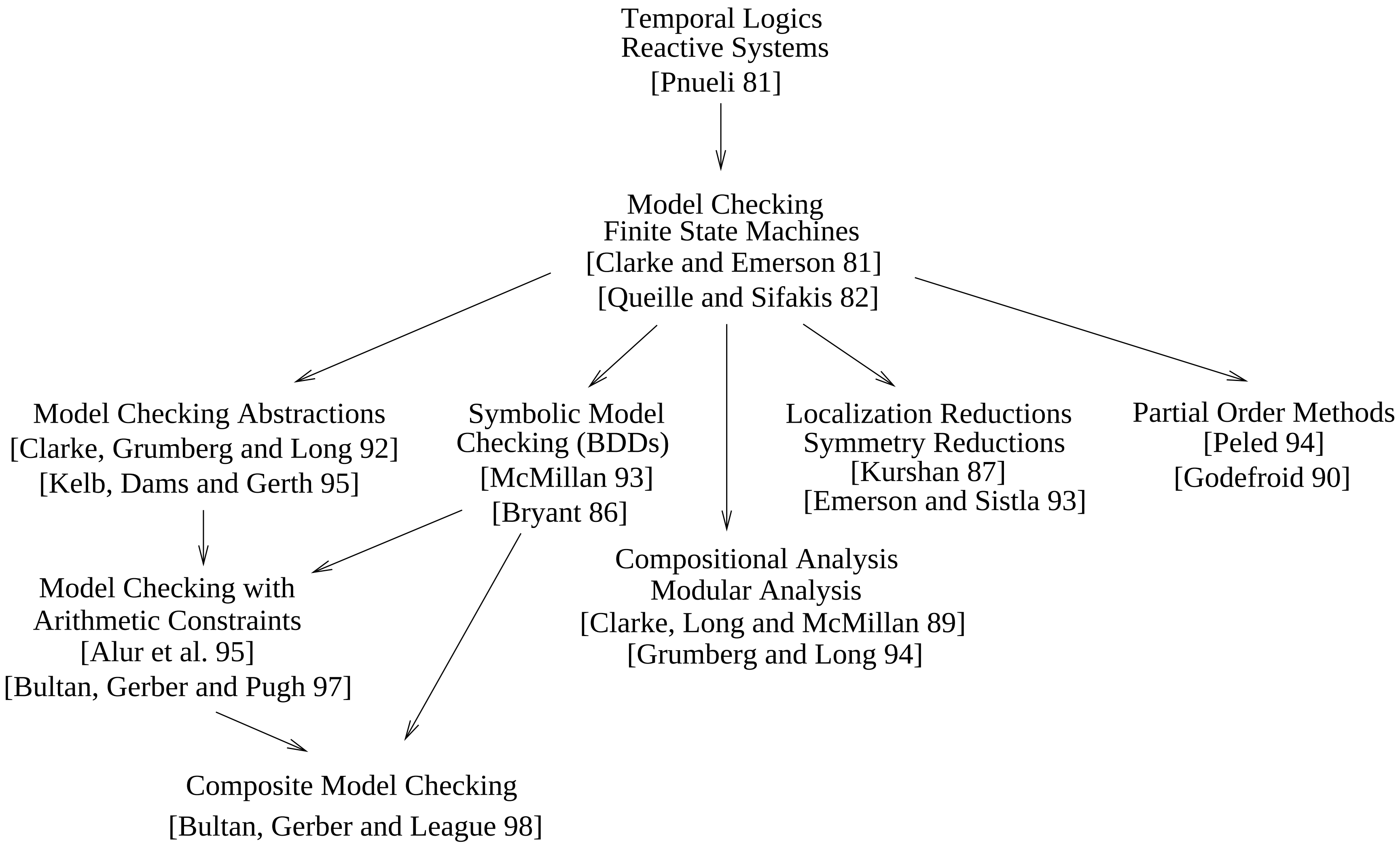
• Arithmetic constraint representation does not scale to large numbers of variables. Representing boolean variables using arithmetic constraints is not feasible.

Solution: Use a composite model. Develop a composite representation which uses BDDs for booleans (and enumerated types) and arithmetic constraints for integers.

Composite Model Checker: Software Architecture



Model Checking Research



Model Checking Software Specifications: Recent Results

• Requirements for the control software of PWR Nuclear Power Plant's reactor cooling system.
Specification language: SCR (Software Cost Reduction model).
Model checker: SPIN [Bharadwaj et al. 97].
Model checker: My yet to be named model checker [Bultan et al. 98].

• Requirements specification of Traffic Alert and Collision Avoidance System II (TCAS II).
Specification language: RSML (Requirements State Machine Language).
Model checker: SMV [Chan et al. 98].

• Software requirements specifications of A-7E aircraft.
Specification language: SCR.
Model checker: SMV [Sreemani et al. 98].

Challenges in Model Checking Software Specifications

- Translating specifications to the input language of the model checker.
- Developing efficient state exploration techniques for various specification languages.
- Stating the properties of the system to be analyzed.
- State space explosion problem: Too many components, too many variables.

Incremental Analysis

- Use abstraction and compositional analysis to analyze the system incrementally.
- Abstract away unrelated variables.
- Analyze parts of the system. Do not compose components until it is necessary.
- Allow user to guide the abstractions and the compositional analysis. Semi-automated instead of fully-automated analysis.

Other Successful Applications of Model Checking

- Cache Coherence Protocols: Verification of Futurebus+ cache coherence protocol using SMV [Clarke et al. 92]. First automatically found bug in a published IEEE standard.

- Protocol specifications: Found an inconsistency in Mobile IPv6 specification using Nitpick model checker [Jackson 98].

- Hardware verification: First wave of startup companies are trying to cash in.

My Longterm Goal

Establish a relationship between model checkers (in general verifiers) and specification languages similar to the relationship between compilers and programming languages.

Future Directions

- Extending the composite model checking technique to other symbolic representations (such as automata which could be very useful in protocol analysis).

- Developing a model checker for a specific specification language (Statecharts) which exploits its structure.

- Developing automated abstraction and compositional analysis techniques to improve scalability of model checking.

- Static analysis and verification of specification languages used for electronic commerce: Workflow languages. (595I Wednesdays 1:30–3pm, CS conference room).

- Using model checking to guarantee security of untrusted code. Static analysis and verification of executable code.