

Model-Checking

- Idea of model-checking: establish that the system is a model of a formula (doing a search).
- CTL Model Checking
- SMV input language and its semantics
- SMV examples
- Notion of fairness
- Binary Decision Diagrams.
- Symbolic model-checking and fixpoints.
- Abstractions

18

CTL Model checking

Assumptions:

1. finite number of processes, each having a finite number of finite-valued variables.
2. finite length of CTL formula

Problem:

Determine whether formula f_0 is true in the finite structure M .

Algorithm overview:

1. $f_0 = \text{TRANSLATE}(f_0)$ (in terms of AF, EU, EX, \wedge , \vee , \perp)
2. Label the states of M with the subformulas of f_0 that are satisfied there and working outwards towards f_0 .

Ex: $\text{AF}(a \wedge \text{E}(b \cup c))$

3. If starting state s_0 is in the final set, then f_0 is holds on M , i.e.

$$(s_0 \in \{s \mid M, s \models f_0\}) \Rightarrow (M \models f_0)$$

19

Labeling Algorithm

Suppose ψ is a subformula of f and states satisfying all the immediate subformulas of ψ have already been labeled. We want to determine which states to label with ψ If ψ is:

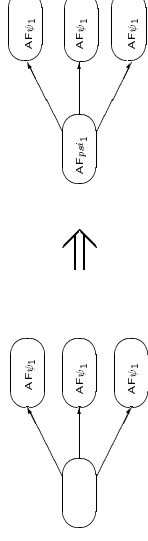
- \perp : then no states are labeled with \perp .
- p (prop. formula): label s with p if $p \in I(s)$.
- $\psi_1 \wedge \psi_2$: label s with $\psi_1 \wedge \psi_2$ if s is already labeled both with ψ_1 and with ψ_2 .
- $\neg\psi_1$: label s with $\neg\psi_1$ if s is not already labeled with ψ_1 .
- EX ψ_1 : label any state with EX ψ_1 if one of its successors is labeled with ψ_1 .

20

Labeling Algorithm (Cont'd)

- AF ψ_1 :
 - If any state s is labeled with ψ_1 , label it with AF ψ_1 .
 - Repeat: label any state with AF ψ_1 if all successor states are labeled with AF ψ_1 , until there is no change.

Ex:

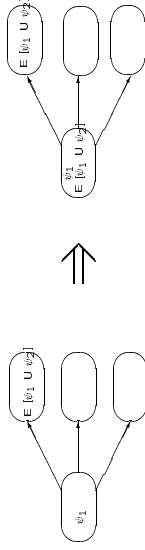


21

Labeling Algorithm (Cont'd)

- $E[\psi_1 \cup \psi_2]$:
 - If any state s is labeled with ψ_2 , label it with $E[\psi_1 \cup \psi_2]$.
 - Repeat: label any state with $E[\psi_1 \cup \psi_2]$ if it is labeled with ψ_1 and at least one of its successors is labeled with $E[\psi_1 \cup \psi_2]$, until there is no change.

Ex:



Output states labeled with f .

Complexity: $O(|f| \times S \times (S + |R|))$ (linear in the size of the formula and quadratic in the size of the model).

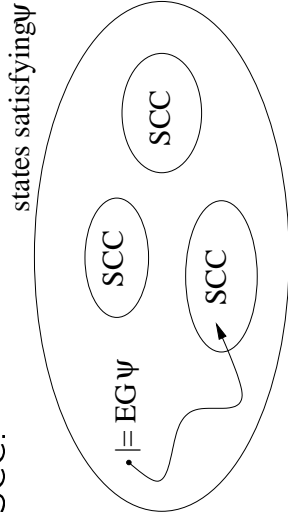
Handling $EG\psi_1$ directly

- $EG\psi_1$:
 - Label *all* the states with $EG\psi_1$.
 - If any state s is *not* labeled with ψ_1 , *delete* the label $EG\psi_1$.
 - Repeat: *delete* the label $EG\psi_1$ from any state if *none* of its successors is labeled with $EG\psi_1$; until there is no change.

This is a *backward* analysis.

Even Better Handling of EG

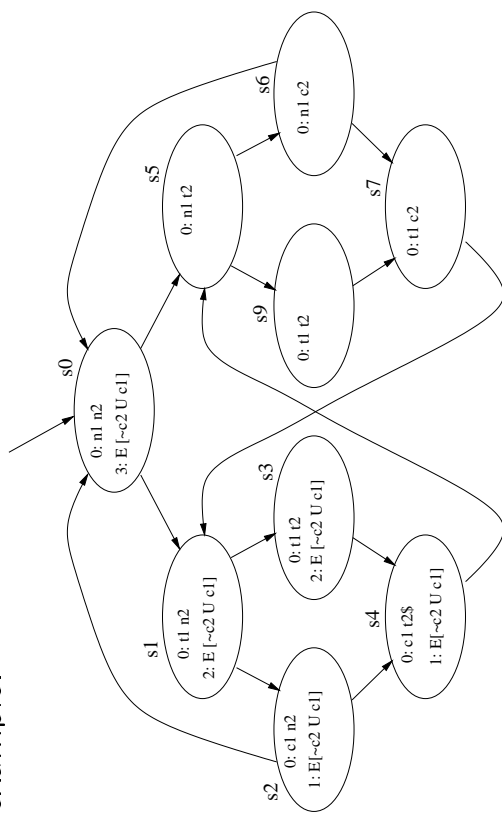
- restrict the graph to states satisfying ψ_1 , i.e., delete all other states and their transitions;
- find the maximal *strongly connected components* (SCCs); these are maximal regions of the state space in which every state is linked with every other one in that region.
- use breadth-first searching on the restricted graph to find any state that can reach an SCC.



Complexity: $O(|f| \times (S + |R|))$ (linear in size of model and size of formula).

Example

Verifying $E[\neg c_2 \cup c_1]$ on the mutual exclusion example.



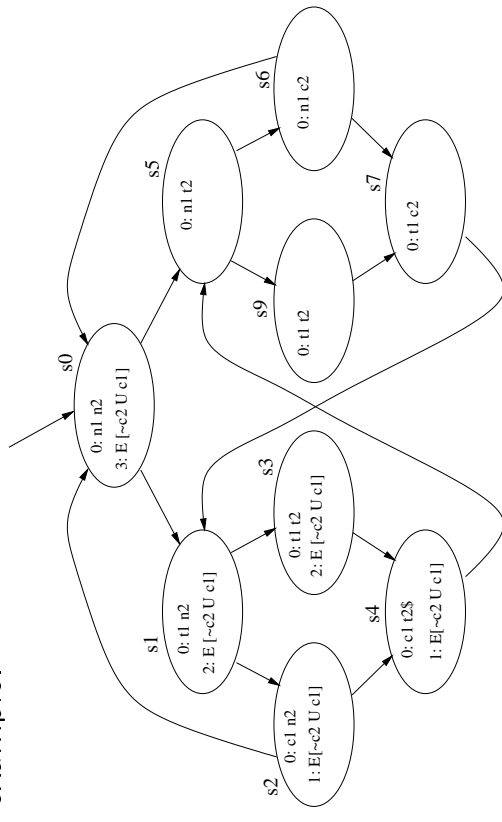
CTL Model-Checking

- Michael Browne, CMU, 1989.
- Usually for verifying concurrent *synchronous* systems (hardware, SCR specs...)
- Specify correctness criteria: safety, liveness...
- Instead of keeping track of labels for each state, keep track of a set of states in which a certain formula holds.

26

Example

Verifying $E[\neg c_2 \cup c_1]$ on the mutual exclusion example.



State Explosion

Imagine that you a Kripke structure of size n . What happens if we add another boolean variable to our model?

How to deal with this problem?

- Symbolic model checking with efficient data structures (BDDs). Don't need to represent and manipulate the entire model. See Ch. 6 and later in the course. Model-checker SMV [McMillan, 1993].
- Abstraction: we abstract away variables in the model which are not relevant to the formula being checked (see later).
- Partial order reduction: for asynchronous systems, several interleavings of component traces may be equivalent as far as satisfaction of the formula to be checked is concerned.
- Composition: break the verification problem down into several simpler verification problems.

SMV

Symbolic model verifier – a program that uses symbolic model checking algorithm. The language for describing the model is a simple parallel assignment.

- Can have synchronous or asynchronous parallelism.
- Model environment non-deterministically.
- Also use non-determinism for systems which are not fully implemented or are abstract models of complex systems.

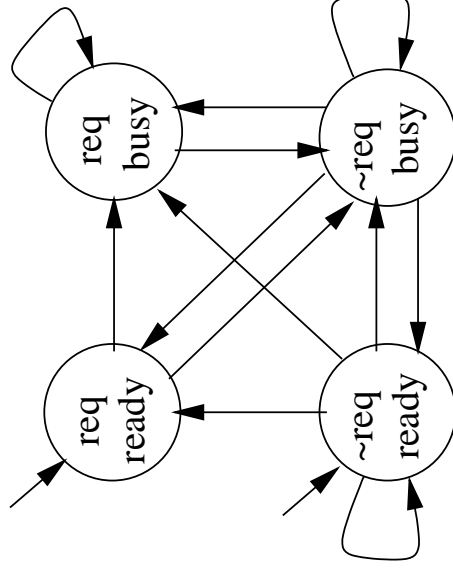
First SMV Example

```
MODULE main
VAR
  request : boolean;
  state : {ready, busy};
ASSIGN
  init(state) := ready;
  next(state) := case
    request : busy;
    1: {ready, busy}
  esac;
SPEC
  AG(request -> AF state = busy)
```

Note that request never receives an assignment – this models input.

30

Model for First SMV Example



31

More About the Language

- Program may consist of several modules, but one has to be called `main`.
- Each variable is a state machine, described by `init` and `next`.
- Variables are passed into modules by reference.
- Comment – anything starting with `--` and ending with a newline.
- No loops.
- Datatypes: boolean, enumerated types, user-defined modules, arrays, integer subranges.

```
VAR
state : {on, off};
state1 : array 2..5 of {on, off};
state2 : computeState(1);
state3 : compute;
state4 : array 2..5 of state; <- error
state5 : array on..off of boolean; <- error
```

32

Another Example

```
MODULE main
VAR
  bit0 : counter_cell(1);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);
SPEC
  AG AF bit2.carry_out
SPEC AG(!bit2.carry_out)
MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := 0;
  next(value) := (value + carry_in) mod 2;
DEFINE
  carry_out := value & carry_in;
```

- *a.b* – component *b* of module *a*.
- **DEFINE** – same as **ASSIGN** but
 - cannot be given values non-deterministically
 - are dynamically typed
 - do not increase the size of state space.

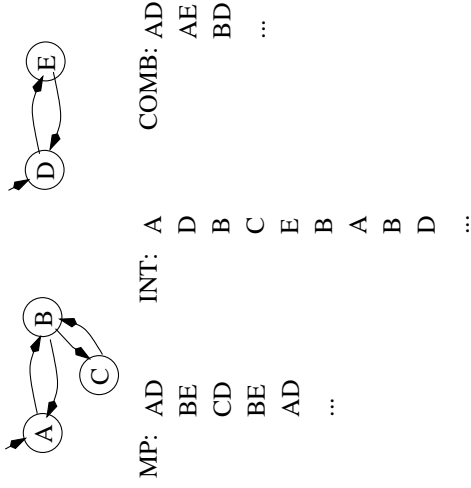
33

Models of Concurrency

Maximum parallelism – "simultaneous execution of atomic actions in all system modules capable of performing an operation."

Interleaving – "concurrent execution of modules is represented by interleaving of their atomic actions".

Example:



Modeling Interleaving

Keyword process for modeling interleaving. The program executes a step by non-deterministically choosing a process, then executing all of its assignment statements in parallel.

```

MODULE main
VAR
    gate1 : process inverter(gate3.output);
    gate2 : process inverter(gate1.output);
    gate3 : process inverter(gate2.output);
SPEC
    (AG AF gate1.output) & (AG AF !gate1.output)

MODULE inverter(input)
VAR
    output : boolean;
ASSIGN
    init(output) := 0;
    next(output) := !input;

```

Output of Running SMV

```
-- specification AG AF gate1.output & ... is false
-- as demonstrated by the following execution sequence
-- loop starts here --
state 1.1:
gate1.output = 0
gate2.output = 0
gate3.output = 0
[stuttering]

state 1.2:
[stuttering]

resources used:
user time: 0.11 s, system time: 0.16 s
BDD nodes allocated: 303
Bytes allocated: 1245184
BDD nodes representing transition relation: 32 + 1
```

What went wrong? We never specified that each process has to execute infinitely often – a *fairness* constraint.

Fixing the Example

```
MODULE main
VAR
  gate1 : process inverter(gate3.output);
  gate2 : process inverter(gate1.output);
  gate3 : process inverter(gate2.output);
SPEC
  (AG AF gate1.output) & (AG AF !gate1.output)

MODULE inverter(input)
VAR
  output : boolean;
ASSIGN
  init(output) := 0;
  next(output) := !input;
FAIRNESS
  running
  -- specification AG AF gate1.output .. is true

resources used:
user time: 0.03 s, system time: 0.23 s
BDD nodes allocated: 288
Bytes allocated: 1245184
BDD nodes representing transition relation: 32 + 1
```

Advantages of Interleaving Model

- Allows for a particularly efficient representation of the transition relation:

The set of states reachable by one step of the program is the union of the sets reachable by each individual process. So, do not need reachability graph.

food for slide eater

- But sometimes have increased complexity in representing the set of states reachable in n steps (can have up to s^n possibilities).