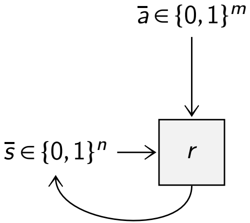
Encoding Finite Transition Systems with Bits: Sequential Circuit

Consider a deterministic finite-state transition system: $M = (S, I, r, A)$
If we pick $n \geq \log_2 |S|$ and $m \geq \log_2 |A|$, we can represent the finite-state transition system using boolean functions:

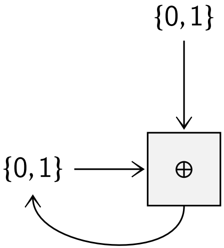
- each element of S as $\bar{s} \in \{0,1\}^n$, so $S = \{0,1\}^n$
- each element of A as $\bar{a} \in \{0,1\}^m$, so $A = \{0,1\}^m$
- initial states $I \subseteq S$ by the characteristic function $\{0,1\}^n \to \{0,1\}$
- deterministic transition relation $r \subseteq S \times A \times S$ as function $(S \times A) \to S$, that is, $\{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$



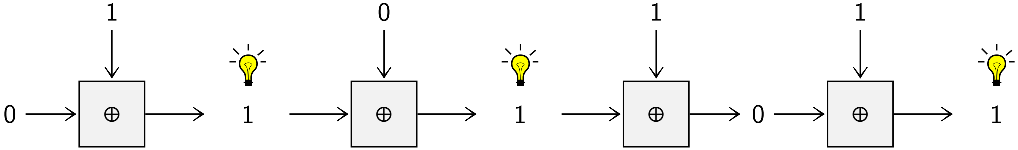
(For non-deterministic systems, we represent r as $(S \times A \times S) \to \{0,1\}$)

Example: Blinking Lights

- $S = \{0,1\}$ ($1 =$ "light on")
- $A = \{0,1\}$ ($1 =$ "toggle light")
- $I(s) = (s = 0)$
- $r(s,a) = s \oplus a$

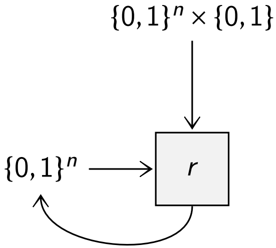


Example trace:

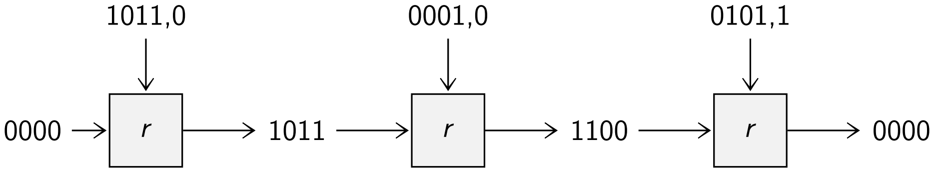


Example: Accumulator with Add and Clear Commands

- $S = \{0,1\}^n$ (value of accumulator)
- $A = \{0,1\}^n \times \{0,1\}$ (number to add, clear signal)
- $I(s) = (s = 0^n)$
- $r(s,(i,c)) =$ if (c) then 0 else $s +_n i$
  ($+_n$ is addition modulo 2^n)



Example trace:

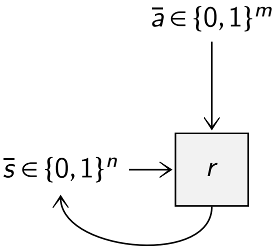


Encoding Finite Transition Systems with Bits: Sequential Circuit

Consider a deterministic finite-state transition system: $M = (S, I, r, A)$

If we pick $n \geq \log_2 |S|$ and $m \geq \log_2 |A|$, we can represent the finite-state transition system using boolean functions:

- each element of S as $\bar{s} \in \{0,1\}^n$, so $S = \{0,1\}^n$
- each element of A as $\bar{a} \in \{0,1\}^m$, so $A = \{0,1\}^m$
- initial states $I \subseteq S$ by the characteristic function $\{0,1\}^n \rightarrow \{0,1\}$
- deterministic transition relation $r \subseteq S \times A \times S$ as function $(S \times A \times S) \rightarrow S$, that is, $\{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n$



How to represent boolean functions, like r, efficiently?

Boolean Function Representation: Circuits

Formulas correspond to *trees*: variables are leaves, operations internal nodes.
More efficient representation that exploits sharing: directed acyclic graphs (DAGs).
We can view DAGs as formulas with *auxiliary variable* definitions.
Example for simple (ripple-carry) n-bit adder:

- input numbers: $s_1 \dots s_n$ and $a_1 \dots a_n$
- output: $s'_1 \dots s'_n$

The formula with auxiliary variables $c_1, \dots, c_{n+1}$:

$$c_1 = 0 \wedge \bigwedge_{i=1}^{n} (s'_i = s_i \oplus a_i \oplus c_i) \wedge (c_{i+1} = (s_i \wedge a_i) \vee (s_i \wedge c_i) \vee (a_i \wedge c_i))$$

We can implement such definitions in hardware: route an output of one gate to multiple other gates.
To get back a tree: substitute all auxiliary variables c_i, but we get much bigger formula. Or, existentially quantify all auxiliary variables.

Observation about Eliminating Variables

Let F, G be propositional formulas and c a propositional variable
Let $F[c := G]$ denote the result of replacing in F *each* occurrence of c by G:

$$\begin{array}{rcl} c[c := G] & = & G \\ (F_1 \wedge F_2)[c := G] & = & F_1[c := G] \wedge F_2[c := G] \\ (F_1 \vee F_2)[c := G] & = & F_1[c := G] \vee F_2[c := G] \\ (\neg F_1)[c := G] & = & \neg(F_1[c := G]) \end{array}$$

We also generalize to simultaneous replacement of many variables, $F[\bar{c} := \bar{G}]$
Then following formulas are equivalent (have same truth for all free variables):

- $F[c := G]$
- $\exists c.((c = G) \wedge F)$
- $\forall c.((c = G) \rightarrow F)$

Note: free variables are the variables occurring in the formula minus quantified ones (c)

Recap: Free Variables for Quantified Boolean Formulas

Quantified boolan formulas (QBF) are build from propositional variables and constants $0, 1$ using $\wedge, \vee, \neg, \rightarrow, \leftrightarrow, \exists, \forall$

(We also write $=$ for $\leftrightarrow$.) A boolean formula is a QBF without quantifiers $\forall, \exists$.

Definition of free variables of a formula:

$$
\begin{array}{rcl}
FV(v) & = & \{v\} \text{ when } v \text{ is a propositional variable} \\
FV(F_1 \wedge F_2) & = & FV(F_1) \cup FV(F_2) \\
FV(F_1 \vee F_2) & = & FV(F_1) \cup FV(F_2) \\
FV(F_1 \rightarrow F_2) & = & FV(F_1) \cup FV(F_2) \\
FV(\neg F_1) & = & FV(F_1) \\
FV(\exists v. F_1) & = & FV(F_1) \setminus \{v\} \\
FV(\forall v. F_1) & = & FV(F_1) \setminus \{v\}
\end{array}
$$

An environment e maps propositional variables to $\{0, 1\}$ (sometimes written $\{\bot, \top\}$)

For vector of n boolean variables $\bar{p} = (p_1, \ldots, p_n)$ and $\bar{v} = (v_1, \ldots, v_n) \in \{0, 1\}^n$, we denote $[\bar{p} \mapsto \bar{v}]$ the environment e given by $e(p_i) = v_i$ for $1 \le i \le n$.

We write $e \models F$ to denote that F is true in environment e.

Recap: Validity, Satisfiability, Equivalence

Definition: Formula F is satisfiable, iff there exists e such that $e \models F$. Otherwise it is called unsatisfiable.
A SAT solver is a program that, given boolean formula F, either gives one satisfying assignment e such that $e \models F$ (if such e exists), or else returns **unsat** (implying that no satisfying assignment exists).

Definition: Formula F is valid, iff for all e, $e \models F$.

Observation: F is valid iff $\neg F$ is unsatisfiable.

Definition: Formulas F and G are equivalent iff for every e that defines all variables in $FV(F) \cup FV(G)$, we have: $e \models F$ iff $e \models G$.

Observation: F and G are equivalent iff $F \longleftrightarrow G$ is valid.

$\exists p.F$ is equivalent to $P[p := 0] \vee P[p := 1]$ whereas $\forall p.F$ to $P[p := 0] \wedge P[p := 1]$

Formula Representation of Sequential Circuits

We represent sequential circuit as $C = (\bar{s}, Init, R, \bar{x}, \bar{a})$ where:

- $\bar{s} = (s_1, \ldots, s_n)$ is the vector of state variables
- $Init$ is a boolean formula with $FV(Init) \subseteq \{s_1, \ldots, s_n\}$
- $\bar{a} = (a_1, \ldots, s_m)$ is the vector of input variables
- $\bar{x} = (x_1, \ldots, x_k)$ is the vector of auxiliary variables (for R)
- R is a boolean formula called transition formula, for which

$$FV(R) \subseteq \{s_1, \ldots, s_n, a_1, \ldots, a_m, x_1, \ldots, x_k, s'_1, \ldots, s'_n\}$$

Transition system for C is (S, I, r, A) where $S = \{0,1\}^n$, $A = \{0,1\}^m$,

- $I = \{\bar{v} \in \{0,1\}^n \mid [\bar{s} \mapsto \bar{v}] \models Init\}$
- $r = \{(\bar{v}, \bar{u}, \bar{v}') \in \{0,1\}^{n+m+n} \mid [(\bar{s}, \bar{a}, \bar{s}') \mapsto (\bar{v}, \bar{u}, \bar{v}')] \models \exists \bar{x}.R\}$

Auxiliary variables $\bar{x}$ are treated as existentially quantified, can use conjucts $x_i = E(\bar{s}, \bar{a}, \bar{x})$ to express intermediate values.

Checking Inductive Invariant using SAT Queries

Given sequential circuit representation $C = (\bar{s}, Init, R, \bar{x}, \bar{a})$ and a formala Inv with $FV(Inv) \subseteq \{s_1, \ldots, s_n\}$, how do we check that Inv is an inductive invariant?
Let us write negations of "$Init \subseteq Inv$" and "$Inv \bullet r \subseteq Inv$"

- An initial state is not included in invariant:

$$Init \wedge \neg Inv$$

- There is a state satisfying invariant, leading to a state that breaks invariant:

$$\underbrace{Inv}_{\bar{s}} \wedge \underbrace{R}_{\bar{s},\bar{a},\bar{x},\bar{s}'} \wedge \underbrace{\neg Inv[\bar{s} := \bar{s}']}_{\bar{s}'}$$

  Note that $\bar{a}, \bar{x}$ variables are also existentially quantified, as they should be.

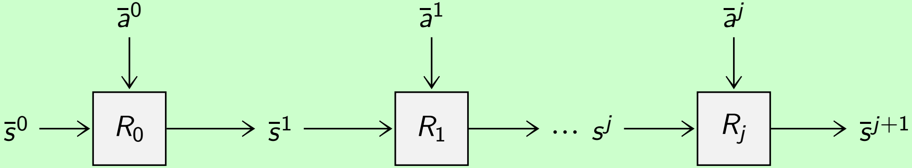
We can check if a formula is an inductive invariant using two queries to a SAT solver and making sure that they both return **unsat**.

Bounded Model Checking for Reachability

We construct a propositional formula T_j such that formula is satisfiable if and only if there exist a trace of length j starting from initial state that satisfies error formula E where $FV(E) \subseteq \{s_1, \ldots, s_n\}$.

$\bar{s}^i$ denotes state variables in step i.

$\bar{a}^i$ denotes inputs in step i.



$$T_j \;\equiv\; Init[\bar{s} := \bar{s}^0] \wedge \left(\bigwedge_{i=0}^{j-1} R_i \right) \wedge E[\bar{s} := \bar{s}^j]$$

where R_i is our transition formula, with variables renamed:

$$R_i \equiv R[(\bar{s}, \bar{a}, \bar{x}, \bar{s}') := (\bar{s}^i, \bar{a}^i, \bar{x}^i, \bar{s}^{i+1})]$$

Write These Conditions Using (Quantified) Boolean Formulas (1/2)

1. Does a property P hold in all states reachable in at most k steps?
   Solution: Define $\bar{R}_i \equiv R_i \vee \bar{s}^i = \bar{s}^{i+1}$. The following formula is valid if and only if the property P holds in all states reachable in at most k steps:
   $$\forall \bar{s}^0,\ldots,\bar{s}^k,\bar{a}^0,\ldots,\bar{a}^{k-1},\bar{x}^0,\ldots,\bar{x}^{k-1}.\ \left(Init[\bar{s} := \bar{s}^0] \wedge \bigwedge_{i=0}^{k-1} \bar{R}_i \right) \rightarrow P[\bar{s} := \bar{s}^k]$$
2. Is there a simple path (no repeated states) of length j from state satisfying F_1 to a state satisfying F_2?
   Solution: First, define the predicate Same over states $\bar{s}$ and $\bar{s}'$ which holds when $\bar{s}$ and $\bar{s}'$ are equal: $\text{Same} \equiv \bigwedge_{k=0}^{n} \bar{s}_k \longleftrightarrow \bar{s}'_k$
   Then, we say there exists a path from a state satisfying F_1 to a state satisfying F_2, such that no two states are the same:
   $$\exists \bar{s}^0,\ldots,\bar{s}^j,\bar{a}^0,\ldots,\bar{a}^{j-1},\bar{x}^0,\ldots,\bar{x}^{j-1}.\ F_1[\bar{s} := \bar{s}^0] \wedge \left(\bigwedge_{i=0}^{j-1} R_i \right) \wedge F_2[\bar{s} := \bar{s}^j] \wedge$$
   $$\bigwedge_{0 \le i_1 < i_2 \le j} \neg Same[\bar{s} := \bar{s}^{i_1}, \bar{s}' := \bar{s}^{i_2}]$$

Write These Conditions Using (Quantified) Boolean Formulas (2/2)

3. Is the system *input enabled* in every state: no matter what the input is, there exists a possible next state?
   Solution:
   $$\forall \bar{s}.\ \mathrm{IE} \qquad \text{where} \qquad \mathrm{IE} \equiv \forall \bar{a}.\ \exists \bar{x}, \bar{s}'.\ R$$
4. Can the system reach in j steps a state where, for some inputs, it cannot make a step?
   Solution: First, define a formula over $\bar{s}$ that holds for states reachable in k steps.
   $$\mathrm{Reach}_k \equiv \exists \bar{s}^0, \ldots, \bar{s}^k, \bar{a}^0, \ldots, \bar{a}^{k-1}, \bar{x}^0, \ldots, \bar{x}^{k-1}.\ \mathit{Init}[\bar{s} := \bar{s}^0] \wedge \left(\bigwedge_{i=0}^{k-1} R_i \right) \wedge \mathrm{Same}[\bar{s}' := \bar{s}^k]$$
   Then, the formula we want is:
   $$\exists \bar{s}.\ \mathrm{Reach}_j \wedge \neg \mathrm{IE}$$