Logic in Automatic Verification

Javier Esparza

Sofware Reliability and Security Group

Institute for Formal Methods in Computer Science

University of Stuttgart

Many thanks to Abdelwaheb Ayari, David Basin,
Armin Biere, Paul Gastin and Denis Oddoux

Automatic verification

The dream:

feed a machine with a system and a specification

push a button

get ‘yes’ or ‘no, because . . . ’

In this talk: three small examples of application of decision procedures for logics to this problem

SAT / QBF

Temporal logics

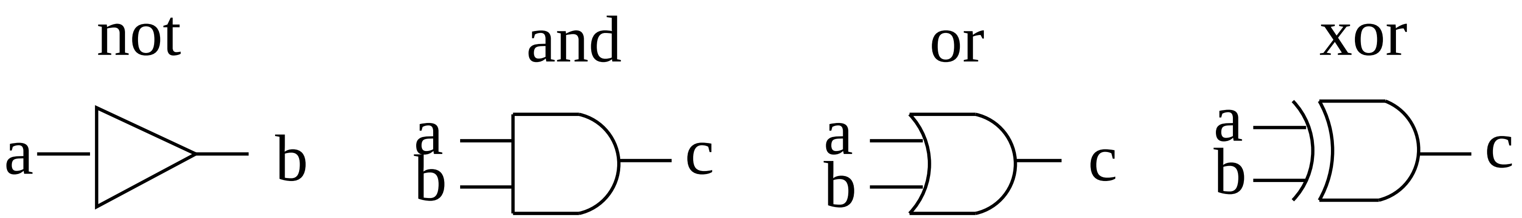
Monadic second order logics

Verifying adders with boolean logic

Modelling circuits with QBL

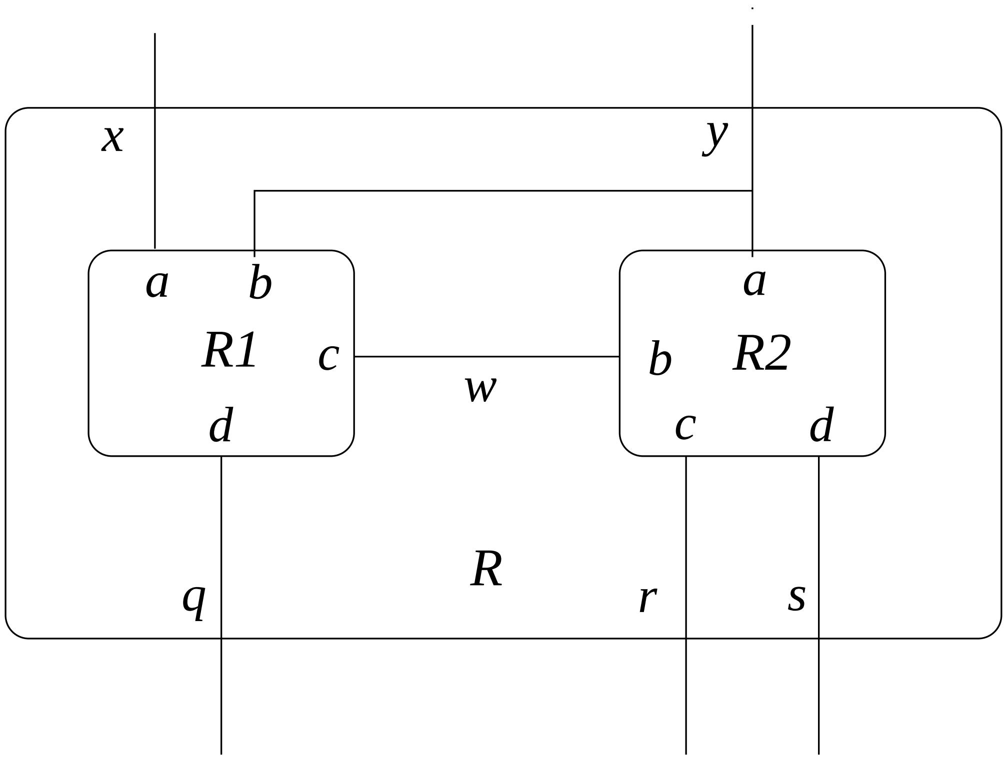
Gates as boolean formulas

Stable states as satisfying truth assignments



$$
\begin{array}{llll}
\textbf{not}(a,b) \equiv \neg a \leftrightarrow b & & \textbf{and}(a,b,c) \equiv (a \wedge b) \leftrightarrow c \\
\textbf{or}(a,b,c) \equiv (a \vee b) \leftrightarrow c & & \textbf{xor}(a,b,c) \equiv ((\neg a \wedge b) \vee (a \wedge \neg b)) \leftrightarrow c
\end{array}
$$

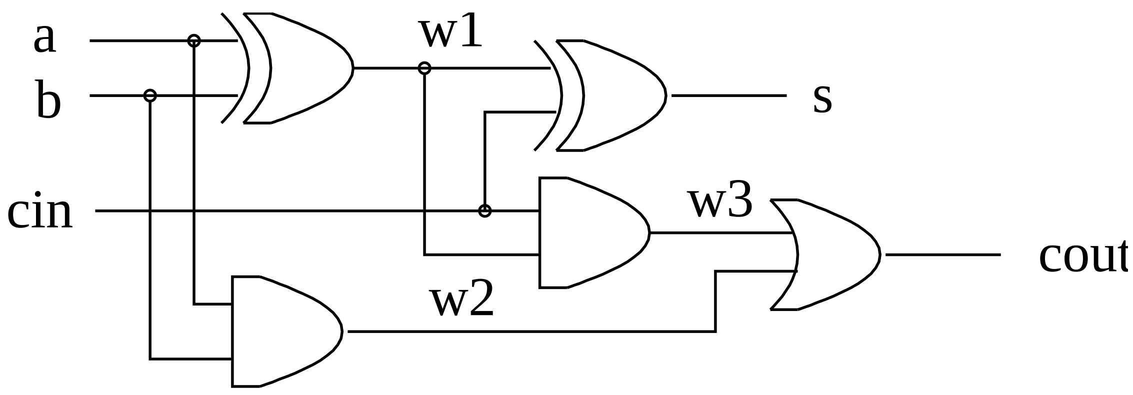
Combine gates with $\wedge$, $\exists$ (and renaming of variables)



$$R(x, y, q, r, s) = \exists w.\, R_1(x, y, w, q) \wedge R_2(y, w, r, s)$$

A full adder

	cout	*cin*	...
	...	*a*	...
+	...	*b*	...
	...	*s*	...

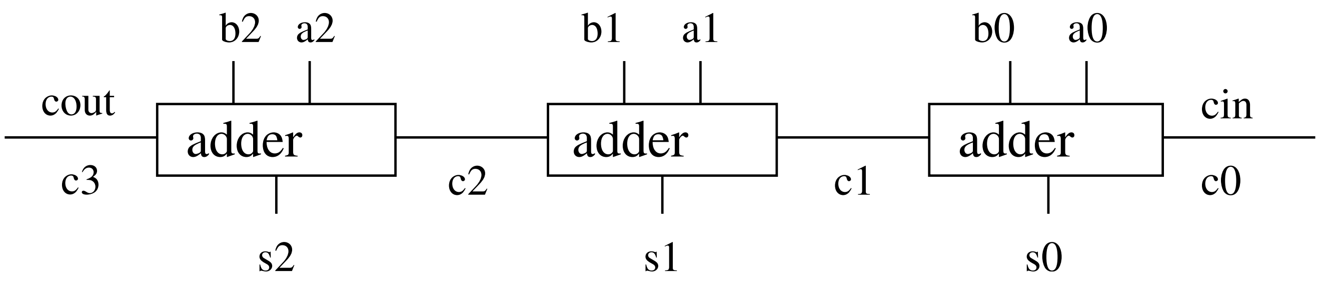


$$\textbf{full\_adder}(a, b, s, cin, cout) \equiv$$
$$\exists w_1, w_2, w_3.\ \textbf{xor}(a, b, w_1) \wedge \textbf{xor}(w_1, cin, s) \wedge \textbf{and}(a, b, w_2) \wedge$$
$$\textbf{and}(cin, w_1, w_3) \wedge \textbf{or}(w_3, w_2, cout)$$

An n-bit ripple-carry adder

$$
\begin{array}{lllll}
 & \color{red}{c_2} & \color{red}{c_1} & \color{red}{cin} & (= 0) \\
 & a_2 & a_1 & a_0 & \\
+ & b_2 & b_1 & b_0 & \\
\hline
cout & s_2 & s_1 & s_0 &
\end{array}
$$

Wire together n 1-bit adders where ith carry-out is i+1st carry-in, first carry is the carry-in and last is the carry-out.



We obtain the formula

$$\mathbf{adder}_n(a_0,\ldots,a_{n-1},b_0,\ldots,b_{n-1},s_0,\ldots,s_{n-1},\ cin,cout) \equiv$$
$$\exists c_0,\ldots,c_n.\,(c_0 \leftrightarrow cin) \wedge (c_n \leftrightarrow cout) \wedge \bigwedge_{i=1}^{n-1} \mathbf{full\_adder}(a_i,b_i,s_i,c_i,c_{i+1}))$$

Problem: too slow!!

Each c_i can only be computed after all of $c_{i-1},\ldots,c_0$ have been computed

Delay: $2n+2$ time units for n-bit numbers

A carry-look-ahead n-adder

Compute all of $c_{n-1}, \ldots, c_0$ (and *cout*) concurrently

First step: given $a_{n-1} \ldots a_0$ and $b_{n-1} \ldots b_0$, identify the $i \in [0, n-1]$ that are

- Generating: $c_{i+1} \equiv 1$ independently of c_i.
  These are the positions such that $1 = g_i \equiv \mathbf{and}(a_i, b_i)$.
- Propagating: $c_{i+1} \equiv c_i$, i.e., c_i is 'propagated' to c_{i+1}.
  These are the positions such that $1 = p_i \equiv \mathbf{xor}(a_i, b_i)$

Observe: all g_i, p_i can be computed simultaneously

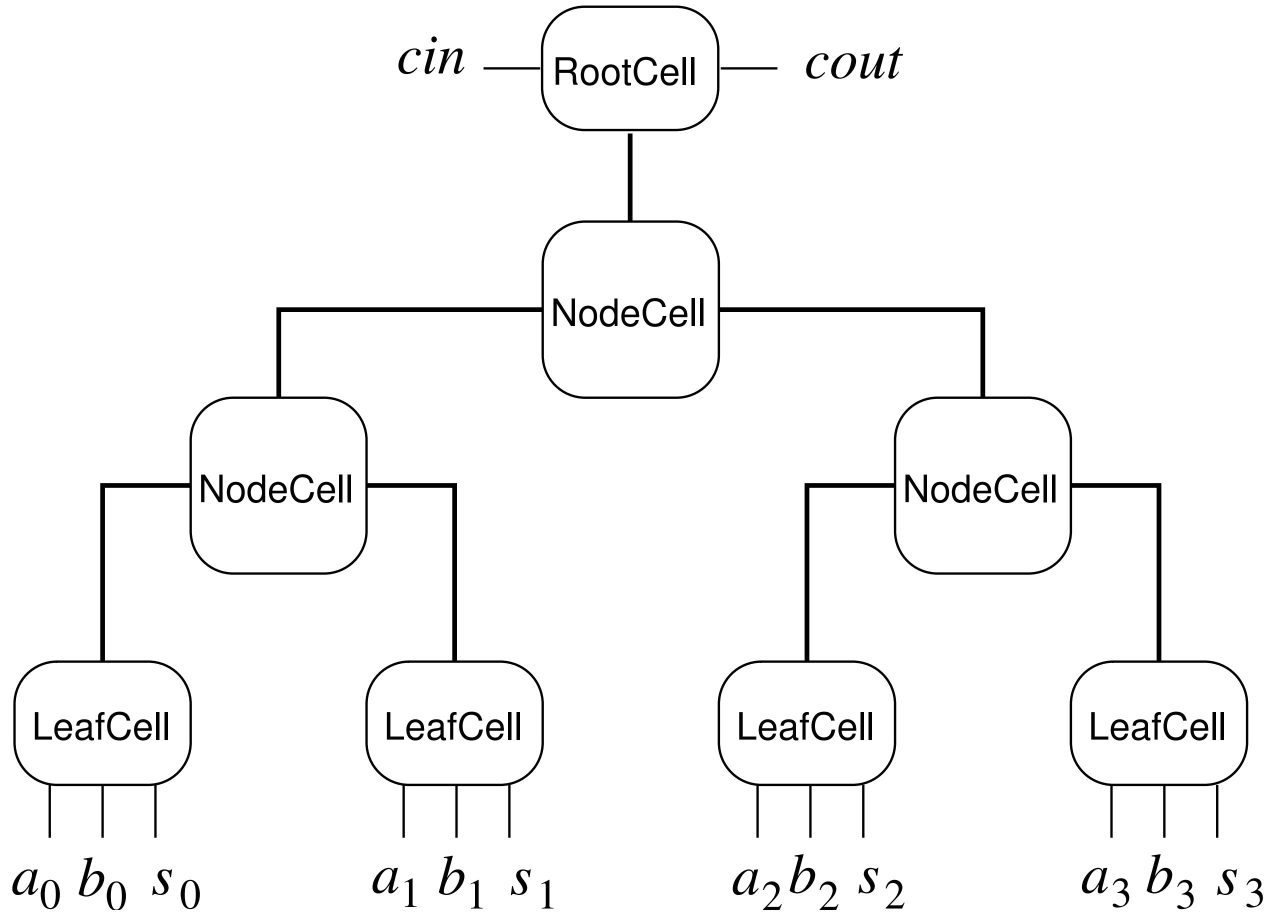
Second step: compute the c_i's by

$$c_i \equiv g_i \vee (p_i \wedge g_{i-1}) \vee (p_i \wedge p_{i-1} \wedge g_{i-2}) \vee \ldots \vee (p_i \wedge p_{i-1} \wedge \ldots \wedge g_0)$$

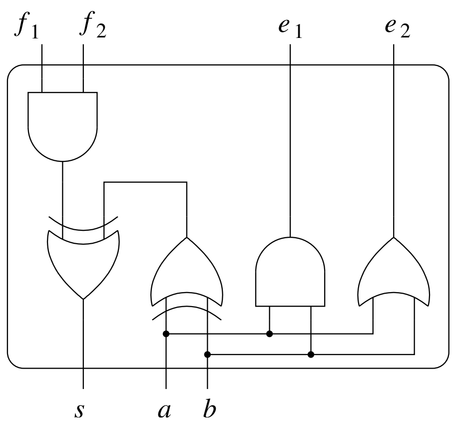
Logarithmic delay in n using balanced $\vee$-trees and $\wedge$-trees

Delay depends on tree structure. For 64 bits: 23-56 units (instead of 130)

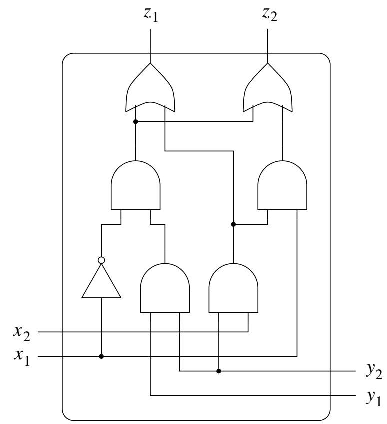
Description of the circuit (for 4 bits)



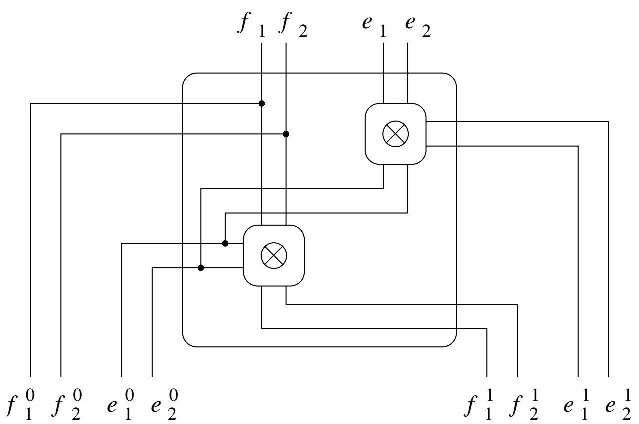
Description of the circuit II



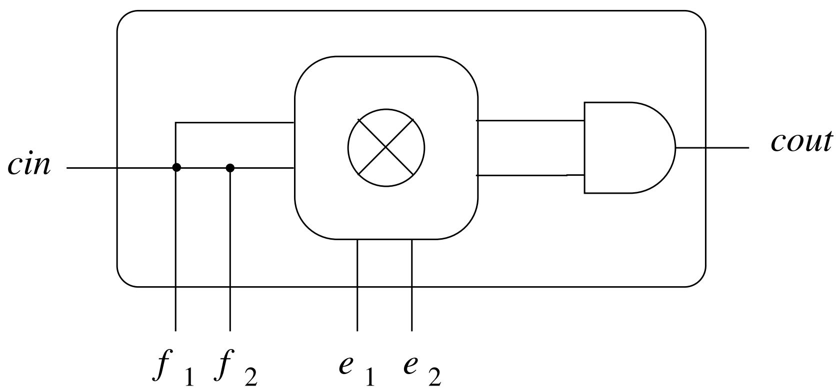
LeafCell circuit



$\otimes$ circuit



NodeCell circuit



RootCell circuit

Verification of the carry-look-ahead *n*-adder

Check if

$$\mathbf{adder}_n(a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}, s_0, \ldots, s_{n-1},\ cin, cout)$$

$$\Leftrightarrow$$

$$\mathbf{cla}_n(a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}, s_0, \ldots, s_{n-1},\ cin, cout)$$

Use SAT solvers or QBF solvers

Results of the SAT 2002 competition on a variant of this problem:

- Task was to compare 2, 4, 8, . . . , 256 bits adders (8 problems)
- From 26 variables and 70 3CNF clauses to 4584 variables and 13226 clauses
- Fastest solver (Zchaff) checked all 8 problems in 14 seconds
- More info at www.satlive.org/SATCompetition/2002/index.jsp

Rule-of-thumb: circuits with some hundreds of gates are routinely solved

Verifying mutual exclusion algorithms

with propositional temporal logics

The mutual exclusion problem

The setting:

- Two computers connected to a database (e.g., of plane bookings)
- Can communicate with each other through shared variables (i.e., variables that both can read and write)
- Both computers run a program having a critical section, from which the program can update the database records

The problem: design the program run by the computers so that

- At any time, at most one computer can be in the critical section
- If a computer wishes to enter the critical section, it eventually will
- These properties still hold if any of the computers breaks down in the non-critical section

Observe: not an input/output system!

A solution due to Peterson

var flag[0], flag[1] : $\{$**true**, **false**$\}$ **init** **false**;
var turn : $\{0,1\}$;

while **true** **do**	**while** **true** **do**
s_0 non-critical section	r_0 non-critical section
s_1 flag[0] := **true**;	r_1 flag[1] := **true**;
s_2 turn := 1;	r_2 turn := 0;
s_3 **while** (flag[1] **and** turn=1) **skip** ;	r_3 **while** (flag[0] **and** turn=0) **skip** ;
s_4 critical section	r_4 critical section
s_5 flag[0] := **false**;	r_5 flag[1] := **false**;
od	**od**

Linear-time temporal logic (LTL)

Built on top of a set AP of atomic propositions

World: valuation of the atomic propositions over $\{\mathbf{true}, \mathbf{false}\}$

Formulas of LTL interpreted over runs: infinite sequences of worlds

Notation:

$$\begin{aligned} \text{run } \rho &= \rho_0\rho_1\rho_2\cdots \\ \text{suffix } \rho|_i &= \rho_i\rho_{i+1}\rho_{i+2}\cdots \end{aligned}$$

Type	Formula	$\rho \models \varphi$ iff ...	Intuition
atomic	p	p is true at ρ_0	p holds now
boolean	$\neg\varphi$	$\rho \not\models \varphi$	
	$\varphi \vee \psi$	$\rho \models \varphi$ or $\rho \models \psi$	
temporal	$\mathbf{X}\,\varphi$	$\rho\|_1 \models \varphi$	φ holds next
	$\mathbf{F}\,\varphi$	$\rho\|_i \models \varphi$ for some $i \in \mathbb{N}$	eventually φ
	$\mathbf{G}\,\varphi$	$\rho\|_i \models \varphi$ for all $i \in \mathbb{N}$	always φ
	$\varphi\ \mathbf{U}\ \psi$	there is $i \in \mathbb{N}$ such that $\rho\|_i \models \psi$ and $\rho\|_j \models \varphi$ for all $0 \leq j < i$	φ until ψ

Application to the mutex algorithm

Atomic propositions: flag[0] = **true**, at_s_4 , . . .

The program satisfies a property if all its runs (executions) satisfy it

The mutual exclusion property:

$$\mathbf{G}(\neg \text{at\_}s_4 \vee \neg \text{at\_}r_4)$$

If computer 0 wants to enter the critical section, it eventually will:

$$\mathbf{G}(\text{flag[0]} = \textbf{true} \Rightarrow \mathbf{F}\,\text{at\_}s_4)$$

But this property does not take breakdowns out of the non-critical section into account . . .

Dealing with breakdowns

Introduce propositions last_0, last_1 saying which computer did the last step

No breakdowns for computer 0:

$$\mathbf{G}\,\mathbf{F}\,\text{last\_0}$$

No breakdowns for computer 0 but possibly in the non-critical section:

$$\mathbf{G}\,\mathbf{F}\,\text{last\_0} \;\vee\; \mathbf{F}\,\mathbf{G}\,\text{at\_}s_0$$

The final property to be checked:

$$(\mathbf{G}\,\mathbf{F}\,\text{last\_0} \vee \mathbf{F}\,\mathbf{G}\,\text{at\_}s_0) \wedge (\mathbf{G}\,\mathbf{F}\,\text{last\_1} \vee \mathbf{F}\,\mathbf{G}\,\text{at\_}r_0)$$

$$\Longrightarrow$$

$$\mathbf{G}(\text{flag[0]} = \textbf{true} \Rightarrow \mathbf{F}\,\text{at\_}r_4) \wedge \mathbf{G}(\text{flag[1]} = \textbf{true} \Rightarrow \mathbf{F}\,\text{at\_}s_4)$$

Automatic verification

The model-checking problem: whether all runs of the algorithm satisfy a given LTL formula

Can be algorithmically solved in three steps (Vardi, Wolper 85):

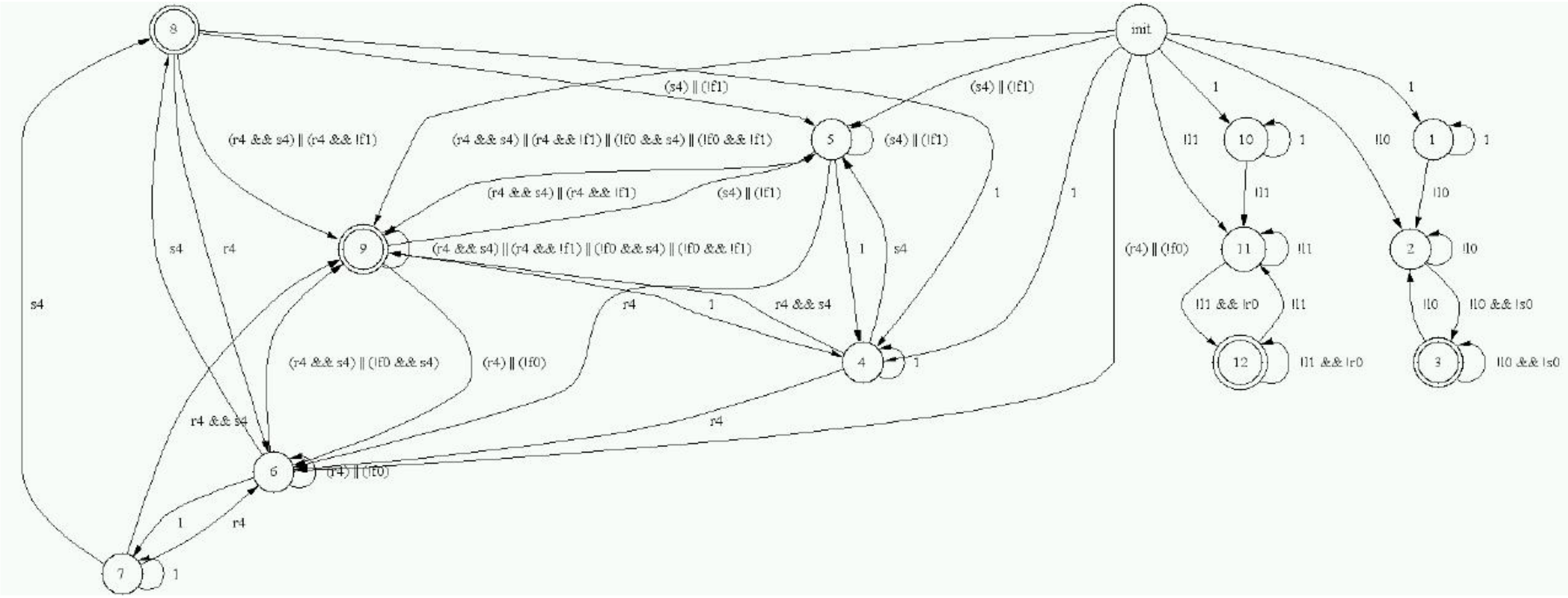
- Construct a Büchi automaton for the negation of the formula (decision procedure for satisfiability)
- Construct the product of this automaton and the state space of the system
- Check emptyness of the product

Linear complexity in the number of states of the program, exponential complexity in the size of the formula

Formula verified in less than one second with Holzmann’s SPIN checker (http://spinroot.com/)

Automaton for the formula

LTL2BA by Gastin and Oddoux (www.liafa.jussieu.fr/ oddoux/ltl2ba/)



Quite sophisticated: formula → alternating Büchi → generalized Büchi → Büchi, with simplification heuristics at each step

The automaton for the negation of the formula has 36 states

Verifying parameterized adders
with monadic second order logics

WS1S : weak MSO logic of one successor

First order variables $p, q, \ldots$ interpreted over $\mathbb{N}$

Second-order variables $X, Y, \ldots$ interpreted over finite subsets of $\mathbb{N}$

$$\phi \; ::= \; \mathbf{s}(q) \mid X(p) \mid \neg\phi \mid \phi \vee \phi \mid \exists p.\, \phi \mid \exists X.\, \phi$$

Definitions (Sample)

$$
\begin{array}{lcl}
\phi_1 \wedge \phi_2 & \equiv & \neg(\neg\phi_1 \vee \neg\phi_2) \\
\forall p.\, \phi & \equiv & \neg\exists p.\, \neg\phi \\
X(0) & \equiv & \exists p.\ (\forall q.\, p \neq \mathbf{s}(q)) \wedge X(p) \\
x = y & \equiv & \forall X.\, X(x) \leftrightarrow X(y) \\
X(p+n) & \equiv & \exists p_1, \ldots, p_n.\, p_1 = \mathbf{s}(p) \wedge \ldots \wedge p_n = \mathbf{s}(p_{n-1}) \wedge X(p_n) \\
x \leq y & \equiv & \forall X.\, (X(y) \wedge \forall z, w.\, (X(z) \wedge s(w) = z \rightarrow X(w)) \rightarrow X(x)) \\
x < y & \equiv & x \leq y \wedge \neg(x = y)
\end{array}
$$

WS1S as a logic of binary strings

Second-order variables interpreted as strings over $\{0, 1\}$

First-order variables interpreted as positions in the string

'$X(p)$ holds iff string X has a 1 at position p'

Formula ϕ with free variables determines a language $\mathcal{L}(\phi)$

$$1101 \in \mathcal{L}(X(1) \wedge X(3)) \qquad 1011 \notin \mathcal{L}(X(1) \wedge X(3))$$

n free variables in ϕ determine language over $\{0, 1\}^n$

$$\forall p.\, p < 4 \rightarrow (X(p) \leftrightarrow \neg Y(p))$$

$$\begin{array}{c|c|c|c|c|}
\hline
X & 0 & 1 & 1 & 0 \\
\hline
Y & 1 & 0 & 0 & 1 \\
\hline
\end{array} \in L(\phi) \quad \text{and} \quad \begin{array}{c|c|c|c|}
\hline
X & 0 & 1 & 1 \\
\hline
Y & 0 & 0 & 0 \\
\hline
\end{array} \notin L(\phi)$$

Examples

$\exists p, q.\, p \neq q \wedge X(p) \wedge X(q)$

– X is a string with a 1 in at least 2 positions, e.g., 010100

$\exists p.\, (\forall q.\, p \neq \mathbf{s}(q)) \wedge X(p)$

– X is a string whose initial letter is 1

$\forall p.\, X(p) \leftrightarrow Y(\mathbf{s}(p))$

– Y is X 'right-shifted' 1 position, e.g.,

0	1	1	0
0	0	1	1

Well-known results

Satisfiability of WS1S is decidable in non-elementary time
(each quantifier alternation adds one exponential)

The language $\mathcal{L}(\phi)$ is regular

A finite automaton accepting $\mathcal{L}(\phi)$ can be computed directly from ϕ

Modelling the family of ALL ripple-carry adders

Recall the formula for a ripple carry n-adder

$$\begin{aligned}
&\mathbf{adder}_n(a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}, s_0, \ldots, s_{n-1}, cin, cout) \equiv \\
&\quad \exists c_0, \ldots, c_n.\ (c_0 \leftrightarrow cin) \wedge (c_n \leftrightarrow cout) \wedge \\
&\quad\quad \bigwedge_{i=0}^{n-1} \mathbf{full_adder}(a_i, b_i, s_i, c_i, c_{i+1})
\end{aligned}$$

We construct the WS1S formula

$$\begin{aligned}
&\mathbf{adder}(n, A, B, S, cin, cout) \equiv \\
&\quad \exists C.\ (C(\mathbf{0}) \leftrightarrow cin) \wedge (C(n) \leftrightarrow cout) \wedge \\
&\quad\quad \forall p.\ p < n \rightarrow \mathbf{full_adder}(A(p), B(p), S(p), C(p), C(p + \mathbf{1})) \wedge \\
&\quad\quad \forall p.\ p \geq n \rightarrow (\neg A(p) \wedge \neg B(p) \wedge \neg S(p))
\end{aligned}$$

A model of **adder**$(A, B, S, cin, cout)$

	0	1	2	3	4	5		
A	1	1	1	0	0	0	...	7
B	1	0	0	1	0	0	...	9
S	0	0	0	0	1	0	...	16

$$\begin{aligned} \mathbf{n} &= 4 \\ \mathbf{cin} &= 0 \\ \mathbf{cout} &= 0 \end{aligned}$$

The set of models of **adder** is 'the union' of all the sets of models of **adder**$_n$

WS2S : weak MSO logic of two successors

Seen as a logic over binary trees

Second-order variables interpreted as trees over $\{0, 1\}$

First-order variables interpreted as positions (nodes) in the tree

Example:

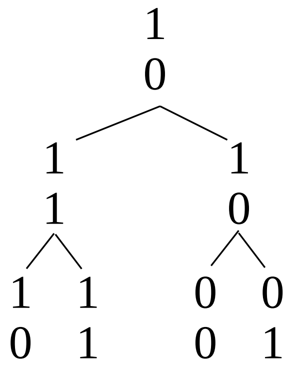
$$X(\epsilon) \;\wedge\; (\forall p.\, X(\mathbf{s}_0(p)) \leftrightarrow X(\mathbf{s}_1(p))) \;\wedge\; \forall p.\, \neg Y(\mathbf{s}_0(p)) \vee \neg Y(\mathbf{s}_1(p))$$

'X contains the root node ϵ, and
a node p is in X iff its brother is also in X, and
for any node p, Y contains at most one of p's successors'

Models

A model of a formula with n free variables is a ‘superposition’ of trees over $\mathcal{B}$, i.e., a tree whose nodes are labelled with elements of $\{0, 1\}^n$

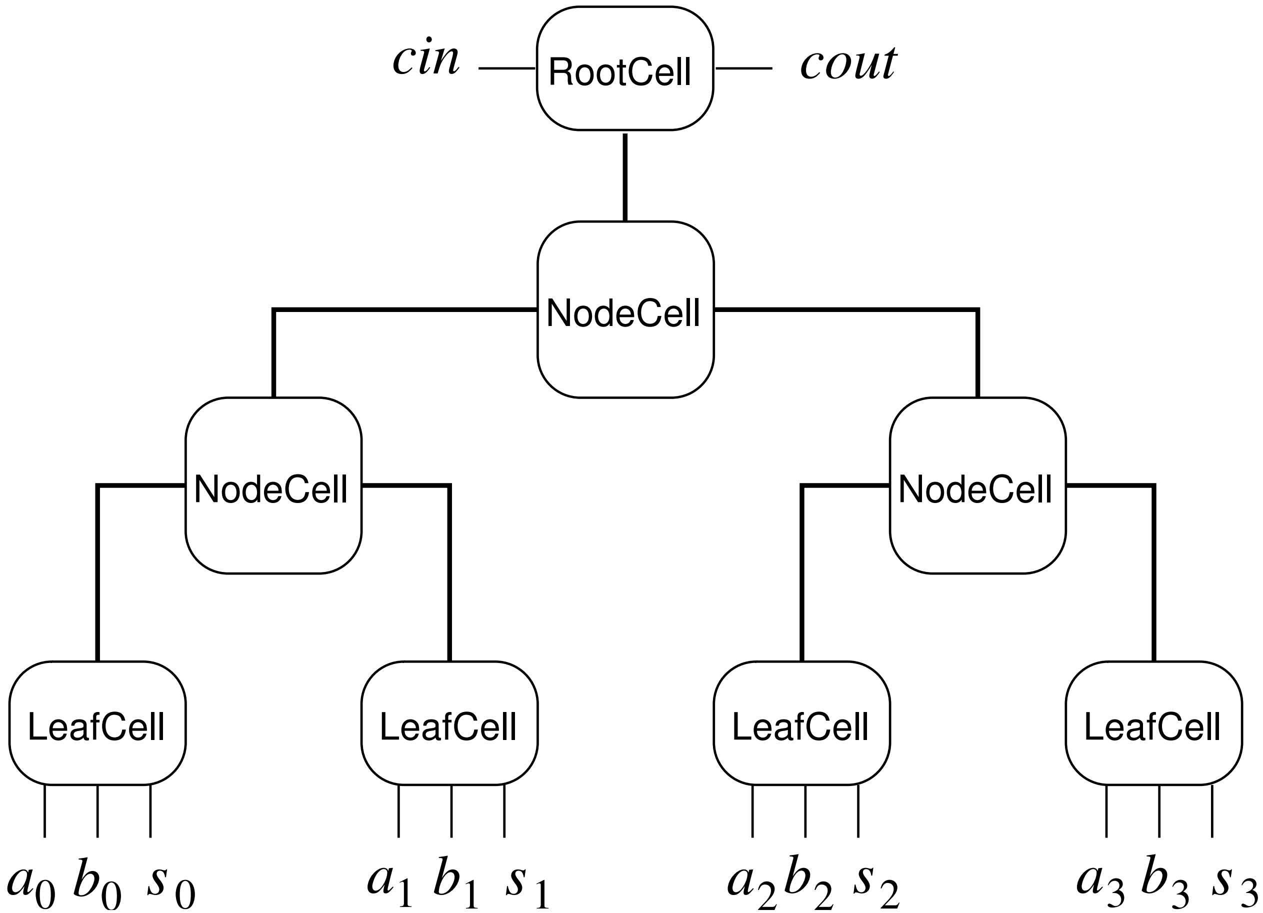
The tree



is a model of

$$X(\epsilon) \ \wedge \ (\forall p. X(\mathbf{s}_0(p)) \leftrightarrow X(\mathbf{s}_1(p))) \ \wedge \ \forall p. \neg Y(\mathbf{s}_0(p)) \vee \neg Y(\mathbf{s}_1(p))$$

Modelling the family of ALL carry-look-ahead adders



The family can be modelled by the formula

$$
\begin{aligned}
&\textbf{cla}(A, B, S, cin, cout) \equiv \exists T, E_1, E_2, F_1, F_2 \\
&\quad \mathbf{RootCell}(F_1, F_2, E_1, E_2, cin, cout) \land \\
&\quad (\forall p.(\mathbf{leaf}(p, T) \rightarrow \mathbf{LeafCell}(A, B, S, F_1, F_2, E_1, E_2, p)) \land \\
&\qquad (\mathbf{node}(p, T) \rightarrow \mathbf{NodeCell}(F_1, F_2, E_1, E_2, p))) \land \\
&\quad \mathbf{shape\_cond}(A, B, S, T)
\end{aligned}
$$

Verification of a parameterized cla-adder

Check validity of

$$\forall A, B, S, cin, cout.\ \mathbf{adder}(A, B, S, cin, cout) \Leftrightarrow \mathbf{cla}(A, B, S, cin, cout)$$

(Requires to embed WS1S into WS2S)

Checked in 1 second by MONA (Mona at www.brics.dk/ mona)

Restrictions:

– only array or tree structures

– only one parameter (two parameters $\rightarrow$ quantification on binary relations)

Conclusions

Conclusions

No conclusions, just examples!