

# Monadic Logics and their Applications

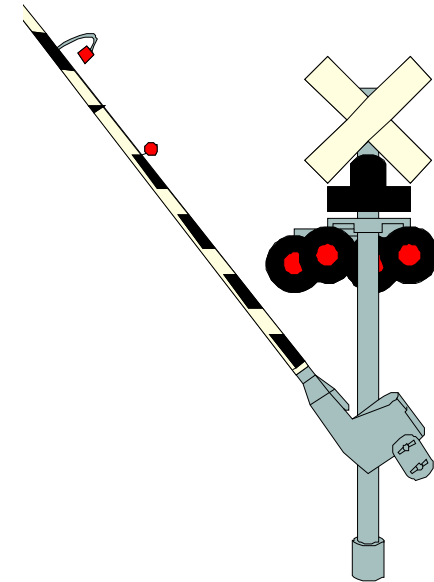
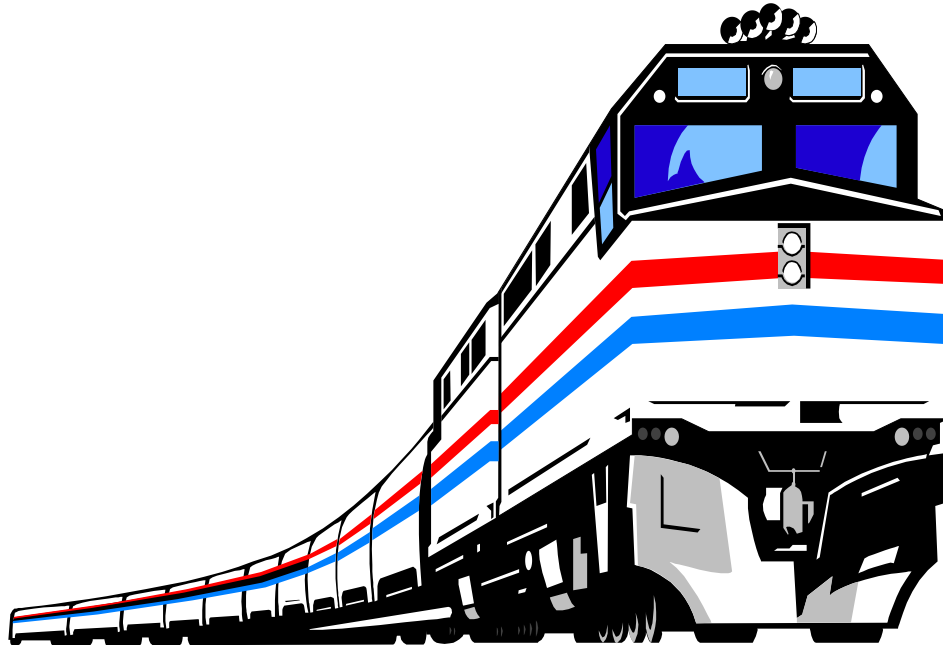
David Basin  
University of Freiburg

# What this tutorial is (and isn't) about

- **Goal:** introduction to modeling with monadic logics (mostly on strings).
- Emphasis is on the basics: how and why.
- Theory will be treated only lightly.
- Combine breadth and depth. Supplement with pointers to literature.

Let's begin with why.

# Correctness matters!

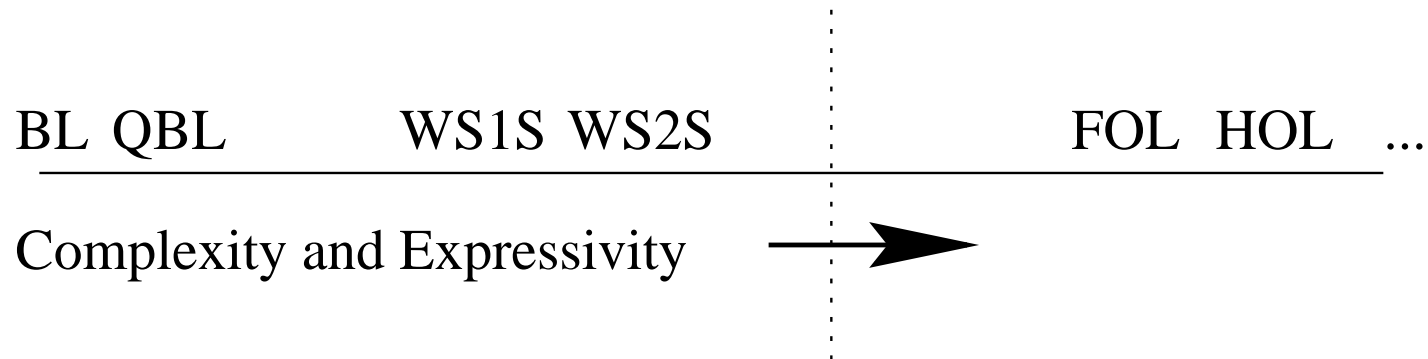


How do we model and reason about the structure and behavior of systems?

## What logic should we use for modeling?

Logic	Modeling (Expressibility)	Complexity
Boolean Logic $\neg(A_0 \wedge B_0) \wedge \neg(A_1 \wedge B_1)$	Bad	Decidable (often quickly)
Temporal Logics $\Box \neg(A \wedge B)$	Good	Decidable (sometimes quickly)
Monadic Logics $\forall p. \neg(A(p) \wedge B(p))$	Very Good	Decidable
<b>Dream-Logic</b> ???	<b>Great</b>	<b>Decidable</b>
FOL/HOL $\forall p. \neg(A(p, f(p)) \wedge B(g(p)))$	Great	Undecidable

# Thesis: monadic second-order logics approach the dream



- They are simple and natural to use

They can be viewed as extensions of (Q)BL with quantifiers over Boolean strings and trees. This is important for practitioners.

- They have a wide range of applications, and
- Complexity can be acceptable (depending on the application).

# Road map

- Motivation/background
  - Practical
  - Technical
- Syntax, semantics, decidability.
- Applications
- Complexity/alternatives

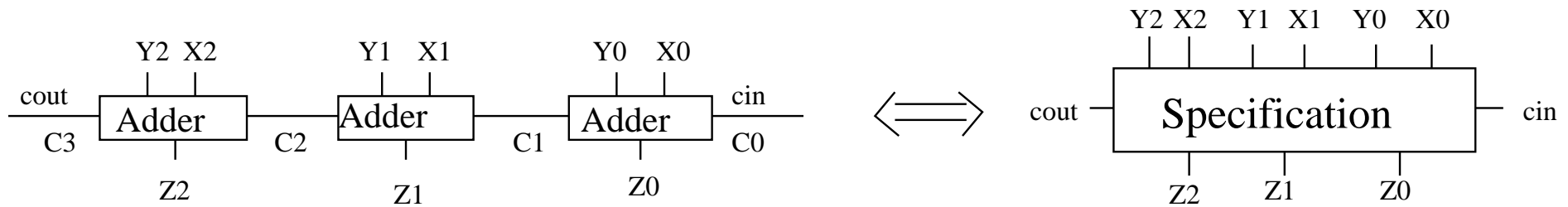
## Motivation (technical)



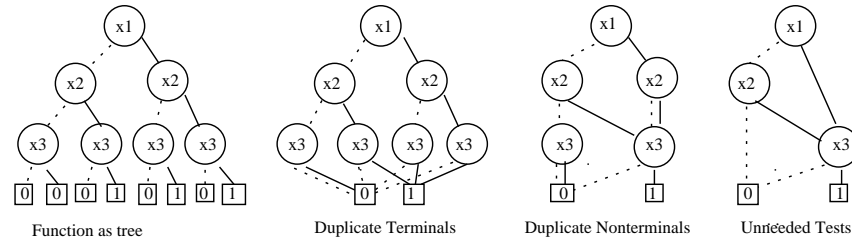
- How can we increase automation/application?
- How can we combine the expressive and automatic paradigms?

# Motivation (cont.): extend BL/BDDs to regular designs

- Finite function equivalence by state-space enumeration

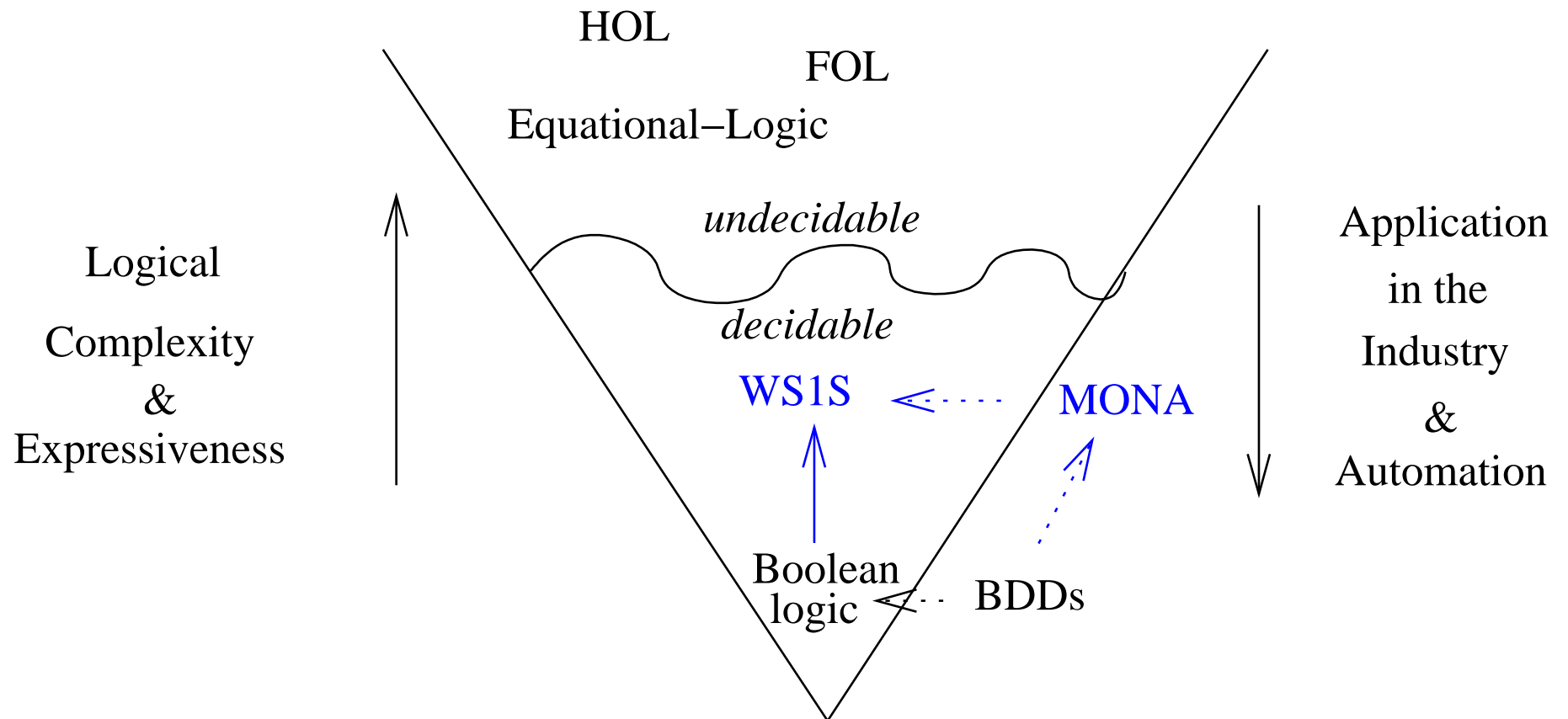


- Efficient using Ordered BDDs



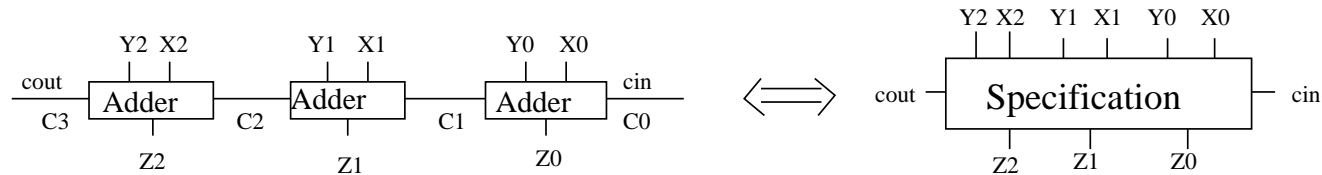
- Infinite state-spaces  $\Rightarrow$  undecidable logics ???
- Solution: Quantified Boolean Logic with string variables  
Generalizes extensions of BDDs to parameterized designs

## Motivation (cont.)



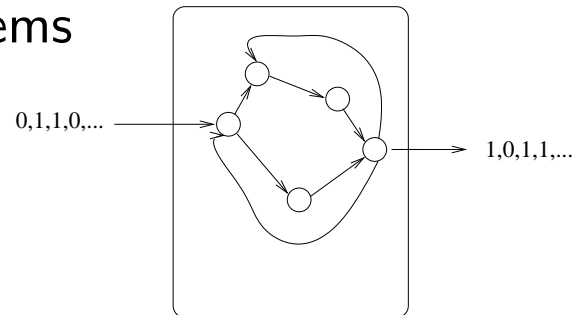
## Approach: exploit regularity

- Regular families



$$\forall X, Y, Z, cin, cout. \text{adder}(X, Y, Z, cin, cout) \leftrightarrow \text{spec}(X, Y, Z, cin, cout)$$

- Finite state transition systems



$$\forall T. \text{sys}(T) \rightarrow \text{props}(T)$$

- Declaratively specify and reduce automatically to automata

# Road map

- Motivation/background

⇒ Syntax, semantics, decidability.

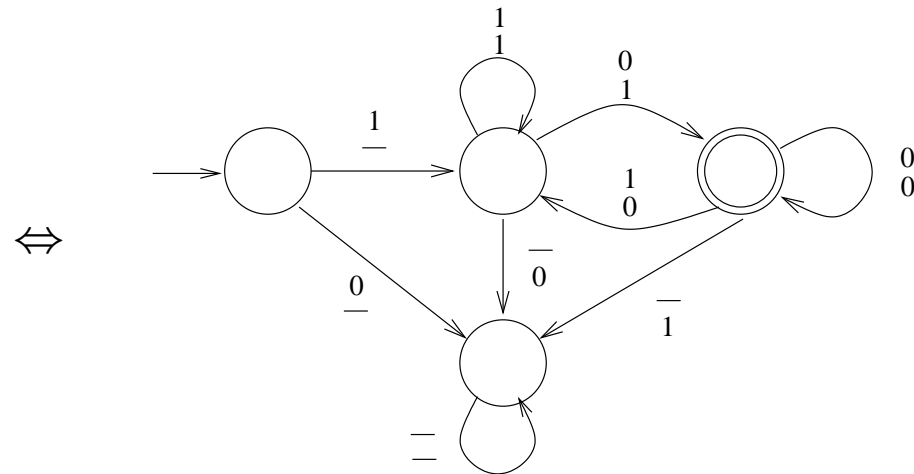
- Applications
- Complexity/alternatives

## The automata/logic connection

Correspondence discovered in the 1950s/60s (Rabin, Büchi, ...):

Logic	Languages	Automata
<b>WS1S</b>	<b>regular languages</b>	<b>automata</b>
S1S	$\omega$ -regular languages	Büchi-automata
(W)S2S	regular tree languages	tree-automata

$$X(0) \wedge \forall p. (X(p) \leftrightarrow Y(s(p))) \Leftrightarrow \left\{ \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array}, \begin{array}{|c|c|c|} \hline 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline \end{array}, \begin{array}{|c|c|c|} \hline 1 & 1 & 0 \\ \hline 0 & 1 & 1 \\ \hline \end{array}, \dots \right\}$$



# Syntax

- Same syntax for several **monadic logics of one successor**
  - WS1S
  - M2L-STR (also known as MSO[S], , ...)

- Syntax:

$\mathcal{V}_1$  and  $\mathcal{V}_2$  sets of first-order and second-order variables.

$$\phi ::= p = s(q) \mid p \in X \mid \neg\phi \mid \phi \vee \phi \mid \exists p. \phi \mid \exists X. \phi, \quad p, q \in \mathcal{V}_1 \text{ and } X \in \mathcal{V}_2$$

- Alternative syntax (as definitional extension)

$$\begin{aligned}
 t & ::= 0 \mid p \mid t + n && p \in \mathcal{V}_1 \text{ and } n \in \mathbb{N} \\
 \phi & ::= t \in X \mid t = t \mid t < t \mid X = Y \\
 & \quad \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \exists p. \phi \mid \forall p. \phi \mid \dots && p \in \mathcal{V}_1 \text{ and } X, Y \in \mathcal{V}_2
 \end{aligned}$$

## Definitions (Sample)

$$\phi_1 \wedge \phi_2 \quad \equiv \quad \neg(\neg\phi_1 \vee \neg\phi_2)$$

$$\forall p. \phi \quad \equiv \quad \neg\exists p. \neg\phi$$

$$X(0) \quad \equiv \quad \exists p. (\forall q. p \neq s(q)) \wedge X(p)$$

$$X(p) \quad \equiv \quad p \in X$$

$$X(p + n) \quad \equiv \quad \exists p_1, \dots, p_n. p_1 = s(p) \wedge \dots \wedge p_n = s(p_{n-1}) \wedge X(p_n)$$

$$x = y \quad \equiv \quad \forall X. X(x) \leftrightarrow X(y)$$

$$x \leq y \quad \equiv \quad \forall X. (X(y) \wedge \forall z, w. (X(z) \wedge s(w) = z \rightarrow X(w)) \rightarrow X(x))$$

$$x < y \quad \equiv \quad x \leq y \wedge \neg(x = y)$$

# Semantics WS1S

- **Interpretation:** interpreted over  $\mathbb{N}$ :  
 $s$  denotes the successor function  
 $\in$  denotes the set membership relation
- **Substitutions:**  $\sigma: x \mapsto i \in \mathbb{N}$  and  $X \mapsto M \in \mathcal{F}(\mathbb{N})$
- **Satisfiability:**

$\sigma \models_{\text{WS1S}} p = s(q)$	iff	$\sigma(p) = \sigma(q) + 1$
$\sigma \models_{\text{WS1S}} p \in X$	iff	$\sigma(p) \in \sigma(X)$
$\sigma \models_{\text{WS1S}} \neg\phi$	iff	$\sigma \not\models_{\text{WS1S}} \phi$
$\sigma \models_{\text{WS1S}} \phi_1 \vee \phi_2$	iff	$\sigma \models_{\text{WS1S}} \phi_1$ or $\sigma \models_{\text{WS1S}} \phi_2$
$\sigma \models_{\text{WS1S}} \exists p. \phi$	iff	$\sigma[i/p] \models_{\text{WS1S}} \phi$ , for some $i \in \mathbb{N}$
$\sigma \models_{\text{WS1S}} \exists X. \phi$	iff	$\sigma[M/X] \models_{\text{WS1S}} \phi$ , for some $M \in \mathcal{F}(\mathbb{N})$

- **Validity:**  $\models_{\text{WS1S}} \phi \equiv \sigma \models_{\text{WS1S}} \phi$ , for all substitutions  $\sigma$

## WS1S as a logic of strings

- A valuation can be encoded as a string

$$\begin{aligned}\sigma(x) &= 2 \rightsquigarrow 001 \\ \sigma(X) &= \{1, 3\} \rightsquigarrow 0101\end{aligned}$$

- Abuse notation:  $0101 \models_{\text{WS1S}} X(1) \wedge X(3)$
- Formula  $\phi$  with free variables determines a language  $\mathcal{L}(\phi)$

$$0101 \in \mathcal{L}(X(1) \wedge X(3)) \quad 1011 \notin \mathcal{L}(X(1) \wedge X(3))$$

$n$  free variables in  $\phi$  determine language over  $\underbrace{\mathcal{B} \times \dots \times \mathcal{B}}_n$

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

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

## Examples

- $\exists p, q. p \neq q \wedge X(p) \wedge X(q)$ 
  - $X$  is a set containing at least two elements, e.g.,  $\{1, 3\}$
  - $X$  is a string with a 1 in at least 2 positions, e.g., 010100
- $\exists p. (\forall q. p \neq s(q)) \wedge X(p)$ 
  - $X$  is a set containing the element 0
  - $X$  is a string whose initial letter is 1
- $\forall X. \exists Y. \forall p. X(p) \leftrightarrow Y(s(p))$ 
  - For every set  $X$  there is a  $Y$  with precisely the successors of elements in  $X$
  - $Y$  is  $X$  'right-shifted' 1 position, e.g.,
 

0	1	1	0
0	0	1	1

## Semantics M2L-STR

- **Interpretation:** interpreted over  $[k] = \{0, \dots, k-1\}$ 
  - $s$  denotes the **partial** function for successor over  $[k]$
  - Sets can be encoded in strings of length  $k$
- **Satisfiability:**  $\sigma$  and  $k \in \mathbb{N}$

$$\begin{array}{ll}
 \sigma^k \models_{\text{M2L}} p = s(q) & \text{iff } \sigma(p) = \sigma(q) + 1 \text{ and } \sigma(p) \in [k] \\
 \sigma^k \models_{\text{M2L}} p \in X & \text{iff } \sigma(p) \in \sigma(X), \sigma(p) \in [k] \text{ and } \sigma(X) \subseteq [k] \\
 \sigma^k \models_{\text{M2L}} \neg\phi & \text{iff } \sigma^k \not\models_{\text{M2L}} \phi \\
 \sigma^k \models_{\text{M2L}} \phi_1 \vee \phi_2 & \text{iff } \sigma^k \models_{\text{M2L}} \phi_1 \text{ or } \sigma^k \models_{\text{M2L}} \phi_2 \\
 \sigma^k \models_{\text{M2L}} \exists p. \phi & \text{iff } (\sigma[i/p])^k \models_{\text{M2L}} \phi, \text{ for some } i \in [k] \\
 \sigma^k \models_{\text{M2L}} \exists X. \phi & \text{iff } (\sigma[M/X])^k \models_{\text{M2L}} \phi, \text{ for some } M \subseteq [k]
 \end{array}$$

- **Validity:**  $\models_{\text{M2L}} \phi \stackrel{\text{def}}{=} \sigma^k \models_{\text{M2L}} \phi$ , for all  $\sigma$  and  $k \in \mathbb{N}$

## Some satisfiability examples

	WS1S	M2L-STR
$X \subseteq Y$	$X \mapsto \{2\}, Y \mapsto \{2, 3\}$	$k = 4, X \mapsto \{2\}, Y \mapsto \{2, 3\}$ $k = 2, X \mapsto \{2\}, Y \mapsto \{2, 3\}$ ☹️
$\exists X. \forall p. p \in X$	unsatisfiable	valid for every $k \in \mathbb{N}$ , $X \mapsto \{0, \dots, k - 1\}$
$\exists X. \exists p. p \in X$	valid	satisfiable for $k > 0$ unsatisfiable for $k = 0$

N.B.: Sets in WS1S correspond to multiple strings

0	0	1	0	0	0	1	0	0	...
0	0	1	1	0	0	1	1	0	

so an automaton accepting the first must accept all other “null paddings”, which is not the case in M2L-STR.

## WS1S & M2L-STR decidable

**Büchi/Elgot/Trahktenbrot** For every WS1S (respectively M2L-STR) formula  $\phi$ , with free variables  $X_1, \dots, X_k$ , the language  $\mathcal{L}(\phi) \subseteq (\{0, 1\}^k)^*$  is regular.

**Decision Procedure:** Given formula  $\phi$

- Translate  $\phi$  to automaton  $A_\phi = \langle Q, \Sigma, \delta, q_0, F \rangle$ , accepting  $w$  iff  $w \models \phi$
- **Output:**
  - “Valid” when  $A_\phi$  accepts all strings, **or**
  - a **minimal countermodel**, if string  $w$  exists,  $w \not\models \phi$

## Translation: work on equivalent “minimal” syntax

- Syntax: Only second-order variables:  $X, Y, \dots$

$$\phi ::= X \subseteq Y \mid \text{succ}(X, Y) \mid \exists X. \phi \mid \neg \phi \mid \phi_1 \wedge \phi_2$$

$\text{succ}(X, Y)$  says  $X$  and  $Y$  are singletons  $\{p\}$  and  $\{s(p)\}$

- Reduction:

– translate out atomic formulae  $y = s(x)$  and  $y \in Z$ , e.g.,

$$\begin{aligned} & \forall x \exists y. (y = s(x) \wedge y \in Z) \\ & \sim \forall X. \text{sing}(X) \rightarrow \exists Y. (\text{sing}(Y) \wedge \text{succ}(X, Y) \wedge Y \subseteq Z) \end{aligned}$$

– using following definitions (set with exactly one proper subset is a singleton)

$$\begin{aligned} \text{sing}(X) & \equiv \exists Y. Y \subseteq X \wedge Y \neq X \wedge \neg \exists Z. (Z \subseteq X \wedge Z \neq X \wedge Z \neq Y) \\ X = Y & \equiv X \subseteq Y \wedge Y \subseteq X \end{aligned}$$

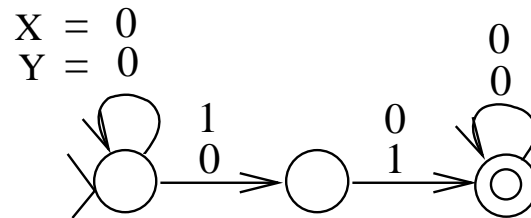
## Translation: inductive argument

- Construct  $A_\phi$  where  $\mathcal{L}(\phi) = \mathcal{L}(A_\phi)$

- Base Case 1:  $\phi \equiv succ(X, Y)$ :
 

0	*
0	

1	0	0
0	1	0



To Prove:  $\mathcal{L}(succ(X, Y)) = \mathcal{L}(A_{succ(X, Y)})$

- Base Case 2:  $\phi \equiv X \subseteq Y$ :
 

1
1

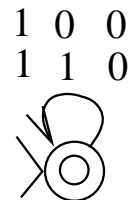
 +
 

0
1

 +
 

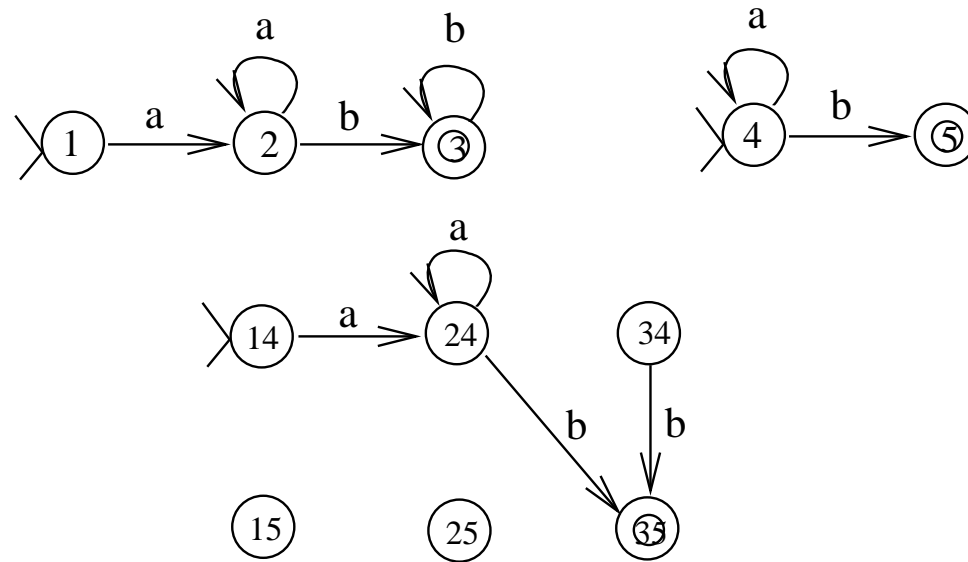
0
0

 )<sup>\*</sup>



## Translation: step case (conjunction)

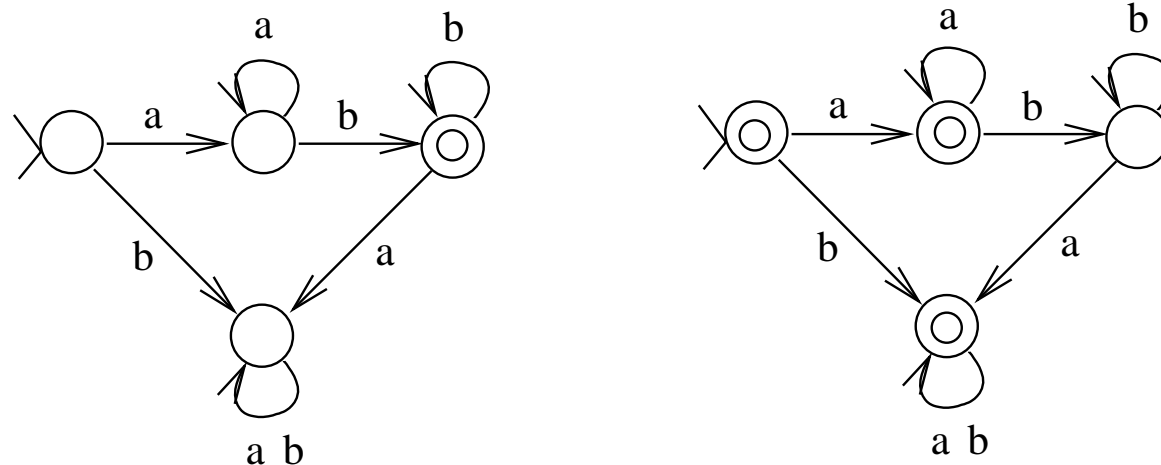
- $\phi_1 \wedge \phi_2$ : product construction  
 Consider  $\mathcal{L}(\phi_1) = aa^*b^*b$  and  $\mathcal{L}(\phi_2) = a^*b$



- Correctness:  $\mathcal{L}(\phi_1 \wedge \phi_2) = \mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2) = \mathcal{L}(A_{\phi_1}) \cap \mathcal{L}(A_{\phi_2}) = \mathcal{L}(A_{\phi_1} \times A_{\phi_2})$
- Complexity:  $O(|A_{\phi_1}| * |A_{\phi_2}|)$

## Translation: step case (negation)

- $\neg\phi$ : complement final states
- Example  $aa^*bb^*$

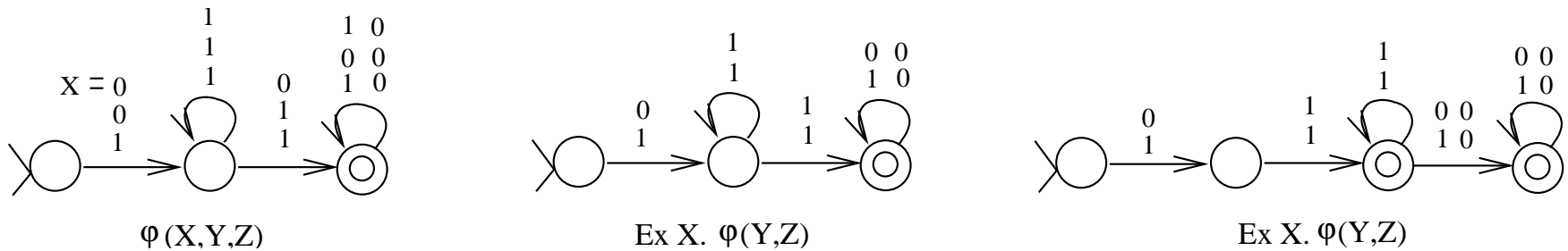


Complement:  $\epsilon + aa^* + b(a + b)^* + aa^*bb^*a(a + b)^*$

- Correctness:  $\mathcal{L}(\neg\phi) = \overline{\mathcal{L}(\phi)} = \overline{\mathcal{L}(A_\phi)} = \mathcal{L}(\mathbf{Comp} A_\phi) = \mathcal{L}(A_{\neg\phi})$
- Complexity: Linear Time, but requires deterministic  $A_\phi$

## Translation: step case (existential quantification)

- $\exists X. \phi$ . Solution for M2L-STR: project  $X$ -‘track’

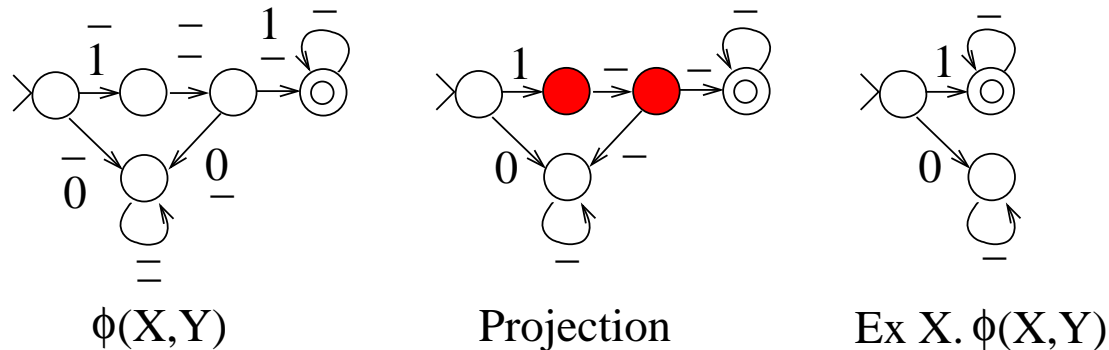


Automaton accepts word if exists (guess!) valuation for  $X$

- Correctness (for M2L-STR): Let  $\Pi_X$  mean ‘delete  $X$ -track’  
 $\mathcal{L}(\exists X. \phi) = \Pi_X \mathcal{L}(\phi) = \Pi_X \mathcal{L}(A_\phi) = \mathcal{L}(A_{\exists X. \phi})$
- Complexity: Linear time but result is non-deterministic!

## Translation: existential quantification/WS1S

- Construction a bit more complicated. Consider  $\phi(X, Y) \equiv Y(0) \wedge X(2)$



- After projection, middle automata would accept  $1000^*$  but not 10 or 1.
  - Result OK for M2L-STR. A solution for  $\exists X. \phi(X, Y)$  requires  $n \geq 3$ .
  - Incorrect for WS1S: these strings all encode same set.
  - WS1S Solution: add red states as end-states (using backward search)
- Correctness:  $\mathcal{L}(\exists X_i. \phi) = \Pi_{X_i}(\mathcal{L}(\phi) / \mathcal{L}^{X_i})$   
 where
 
$$L/L' \equiv \{w \mid \exists u \in L'. wu \in L\}$$

$$\mathcal{L}^{X_i} \equiv \{w \in \mathcal{B}^k \mid j\text{th track is all 0 for } j \neq i\}$$
 and  $k$  is number of free variables in  $\phi$ .

## Translation — Summary

- Establishes that M2L-STR/WS1S formula define regular languages
  - Converse is also true. (Proof based on simple encoding)
- Constructive proof yields basis of decision procedure
  - A valid formula yields a trivial automata that accepts all strings



- An invalid formulae doesn't accept at least one string (a counter model)

Is this practical?

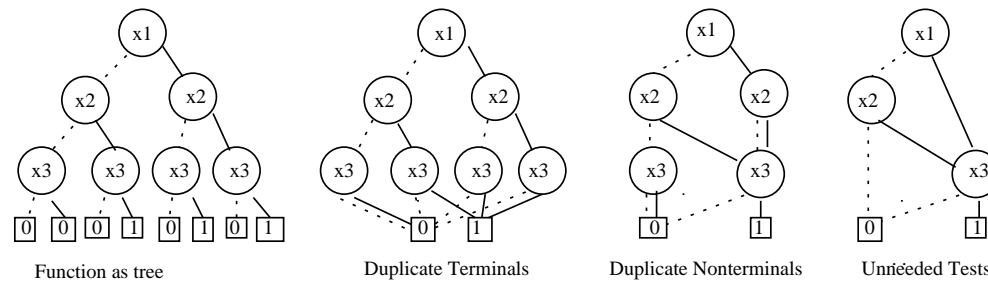
## Problem (Part I): representing $\delta$

- $\phi$  with  $n$ -free variables determines a language over  $\Sigma = \underbrace{B \times \dots \times B}_n$
- Transition function  $\delta$  is a relation  $\Sigma \times Q \times Q$
- Size of relation exponential in number of free variables

$$2^n \times |Q| \times |Q|$$

- Naive (table) encoding  $\Rightarrow$  exponential algorithm independent of state-space size

# OBDDs: partially solve problem I



- Idea (Klarlund et. al.): use OBDDs to represent transition relation  $\delta : \Sigma \times Q \times Q$
- Generalizes OBDDs to (regular) relations over strings

$$R(X, Y) \equiv \forall p. X(p) \leftrightarrow Y(s(p))$$

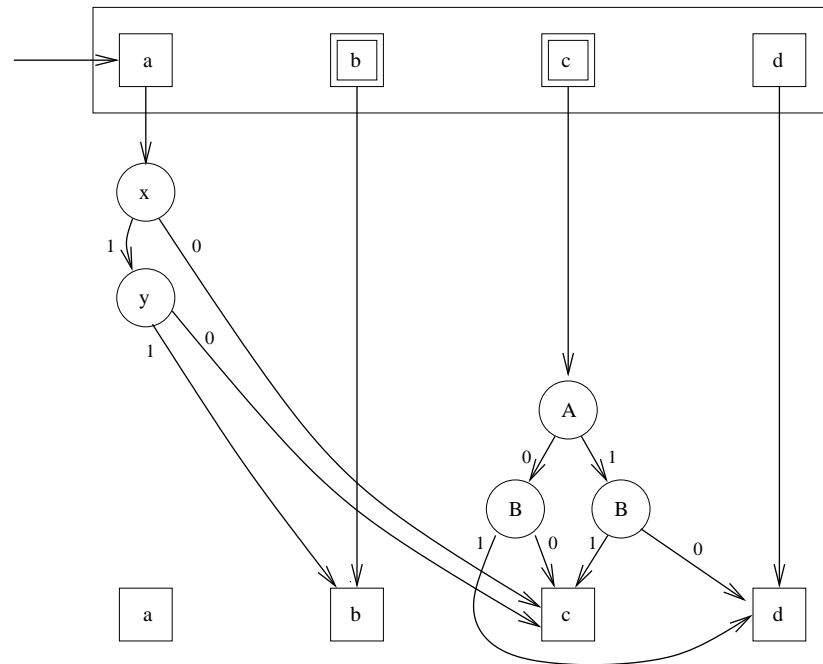
- **Can** lead to exponential compression

## OBDDs (cont.)

$$\phi \equiv x \wedge y \vee A = B$$

Interpretation:  $x = 1, y = 0, n = 3, A = \{0, 2\}, B = \{0, 2\}$

	-1	0	1	2
$x$	1	X	X	X
$y$	0	X	X	X
$A$	X	1	0	1
$B$	X	1	0	1



Note: -1st position is a trick to separate out propositional BDDs.

## OBDDs (cont.)

- Canonical in 2 ways:
  1. Transition function is canonical (reduced)
  2. State-space that of canonical (minimal) automaton
- Subsumes standard BDDs
  - $\phi$  contains only Boolean variables  $\Rightarrow$  3 states:  
1) initial, 2) looping accepting, and 3) looping rejecting
  - Ignoring loops, transition identical to OBDD for  $\phi$
  - Automata operations degenerate to BDD operations
- Subsumes LIFs (Gupta/Fisher) in an “appropriate sense”  
(see Ayari/D.B./Klaedtke CAV 2000)

## Problem (Part II): state explosion

- Negation requires determinization
- Existential quantification introduces non-determinism
- Quantifier alternation yields exponential blow-ups!

$$\forall X. \exists Y. \phi \rightsquigarrow \neg \exists X. \neg \exists Y. \phi$$

If  $|A_\phi| = n$ , then  $|A_{\neg \exists Y. \phi}| = O(2^{|n|})$   
 and  $|A_{\neg \exists X. \neg \exists Y. \phi}| = O(2^{2^{|n|}})$

- Is this bad?

**Pro:** Some systems have many states. Allows nonelementary compression.

**Con:** Not enough memory for some applications

We will later show that for certain formula classes such blow-ups do not occur.

# Road map

- Motivation/background
- Syntax, semantics, decidability.

## ⇒ Applications

- A warm up example
- Parameterized hardware
- Other examples
- Complexity/alternatives

## Note on tool support

- All examples shown here used the Mona system

**<http://www.brics.dk/mona/>**

- Optimized implementation of WS1S and WS2S.
- Special support for M2L-STR **Klarlund CAV'99**
- Other implementations exist: Mosel, MOSEL, implementation in STeP, ...
- Several front-ends exist providing friendly high-level input languages

**Fido (Strings/Trees):** **Klarlund/Schwartzbach IEEE TOS 1999**

**Lisa (Strings/Trees):** based on feature-logic **Ayari/D.B./Podelski, CSL 1997**

**FMona (Strings):** **Bodeveix/Filali, TACAS 2000**

## A warm up example

**Problem:** Model a system in which a man must cross a river with a wolf, goat, and a cabbage. His boat carries at most himself and one other plant or animal. If he leaves the goat and wolf by themselves, or the goat and the cabbage then something bad will happen. The **goal** is for the man to bring all objects safely over to the other side.



## Representation (in Mona)

```
ws1s;  
var2 M, W, G, C;  
  
pred manL(var1 t) = t notin M;  
pred wolfL(var1 t) = t notin W;  
pred goatL(var1 t) = t notin G;  
pred cabL(var1 t) = t notin C;  
  
pred manR(var1 t) = ~manL(t);  
pred wolfR(var1 t) = ~wolfL(t);  
pred goatR(var1 t) = ~goatL(t);  
pred cabR(var1 t) = ~cabL(t);
```

First we encode the domain using second-order variables.  
(Such uninteresting work should be supported by a front-end.)

## Representation (cont.)

```
pred stable(var1 t, var2 A) = (t in A <=> t+1 in A);
pred stable2(var1 t, var2 A, B) = stable(t,A) & stable(t,B);
pred stable3(var1 t, var2 A, B, C) = stable2(t,A,B) & stable(t,C);

pred mLR(var1 t) = manL(t) & manR(t+1);
pred mRL(var1 t) = manR(t) & manL(t+1);
pred manLR(var1 t) = mLR(t) & stable3(t,W,G,C);
pred manRL(var1 t) = mRL(t) & stable3(t,W,G,C);
pred wolfLR(var1 t) = mLR(t) & wolfL(t) & wolfR(t+1) & stable2(t,G,C);
pred wolfRL(var1 t) = mRL(t) & wolfR(t) & wolfL(t+1) & stable2(t,G,C);
pred goatLR(var1 t) = mLR(t) & goatL(t) & goatR(t+1) & stable2(t,W,C);
pred goatRL(var1 t) = mRL(t) & goatR(t) & goatL(t+1) & stable2(t,W,C);
pred cabLR(var1 t) = mLR(t) & cabL(t) & cabR(t+1) & stable2(t,G,W);
pred cabRL(var1 t) = mRL(t) & cabR(t) & cabL(t+1) & stable2(t,G,W);
```

More encoding: movement across the river

## Representation (cont.)

```

pred init = manL(0) & wolfL(0) & goatL(0) & cabL(0);
pred final(var1 t) = manR(t) & wolfR(t) & goatR(t) & cabR(t);

pred badState(var1 t) =
  (wolfL(t)& goatL(t) & manR(t)) | (wolfR(t)& goatR(t) & manL(t)) |
  (cabL(t)& goatL(t) & manR(t)) | (cabR(t)& goatR(t) & manL(t));

pred trans(var1 l) =
  all1 t: t < l =>
    (manLR(t) | manRL(t) | wolfLR(t) | wolfRL(t) |
     goatLR(t) | goatRL(t) | cabLR(t) | cabRL(t))
    & ~badState(t);

ex1 last: init & trans(last) & final(last);

```

Next we encode the initial state, final state and legal transitions.

# Output

MONA v1.3 for WS1S/WS2S

Conjunctive automaton has 13 states, 57 BDD-nodes and 33 transitions

## ANALYSIS

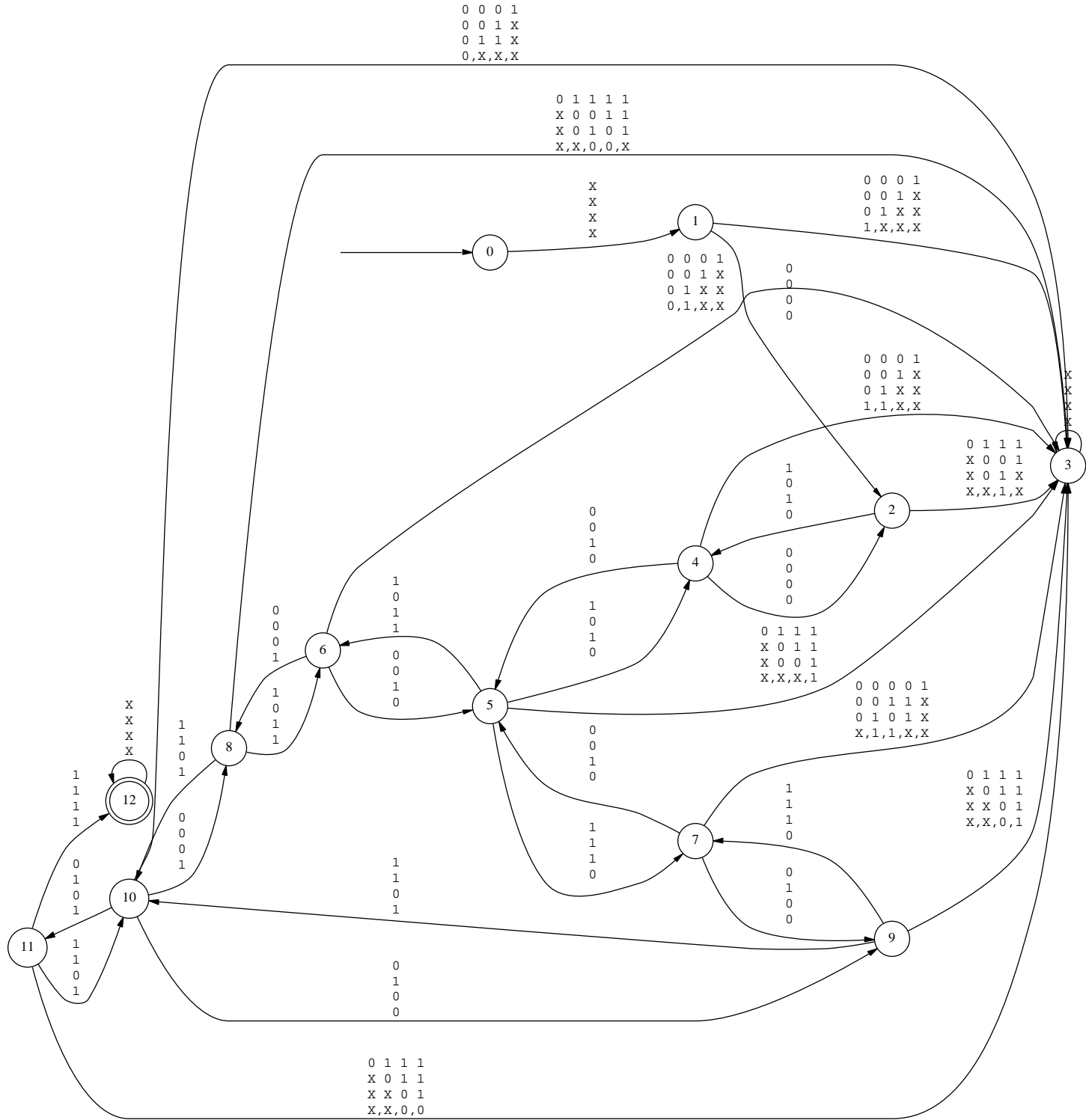
A counter-example (for assertion => main) of least length (0) is:

M	X
W	X
G	X
C	X

A satisfying example (for assertion & main) of least length (8) is:

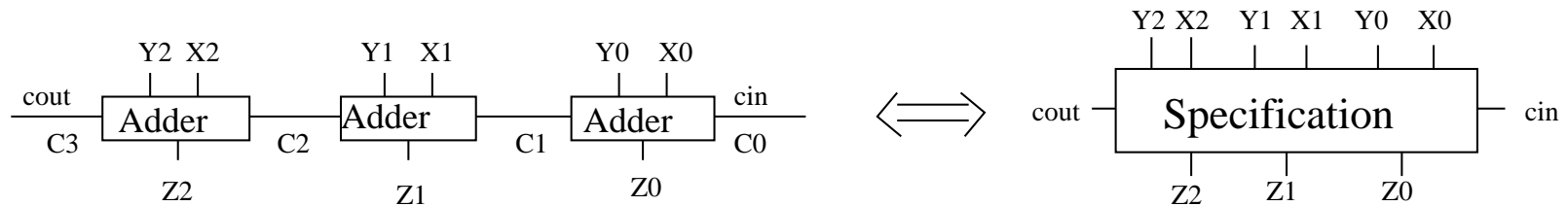
M	X 01010101
W	X 00000111
G	X 01110001
C	X 00011111

Total time: 00:00:00.17

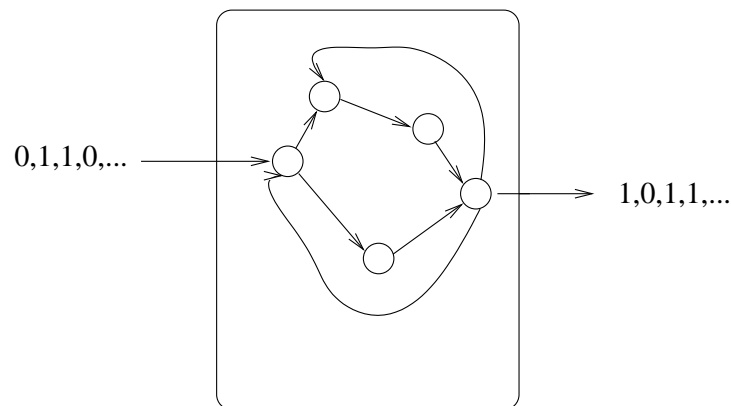


# Parameterized hardware

- Modeling control and data in regular systems
- Systems are parameterized in space or time
  - Infinite families of systems:  $n$ -bit adders, ALUs, ...



- Systems with 'evolving' (but regular) behavior (FSMs, sequential systems)

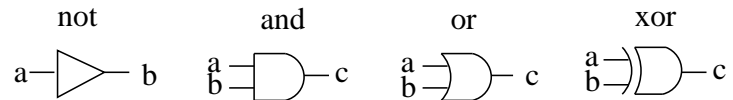


# Circuit specification in WS1S

- Encode Quantified Boolean Logic in WS1S (Mona built-in)

$$\forall x. \exists y. x \leftrightarrow y \quad \sim \quad \forall X. \exists Y. X(0) \leftrightarrow Y(0)$$

- Gates as Boolean relations (QBL in WS1S)



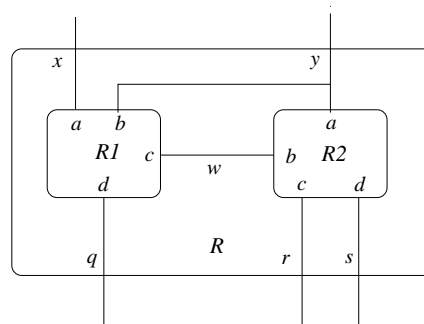
$$\text{not}(a, b) \equiv \neg a \leftrightarrow b$$

$$\text{and}(a, b, c) \equiv (a \wedge b) \leftrightarrow c$$

$$\text{or}(a, b, c) \equiv (a \vee b) \leftrightarrow c$$

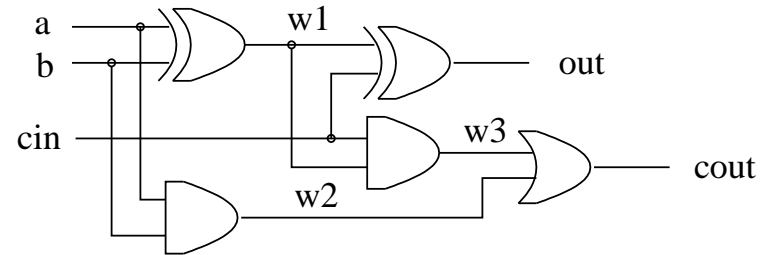
$$\text{xor}(a, b, c) \equiv ((\neg a \wedge b) \vee (a \wedge \neg b)) \leftrightarrow c$$

- Combine relations with  $\wedge/\exists$



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

## Modeling with boolean logic — full adder



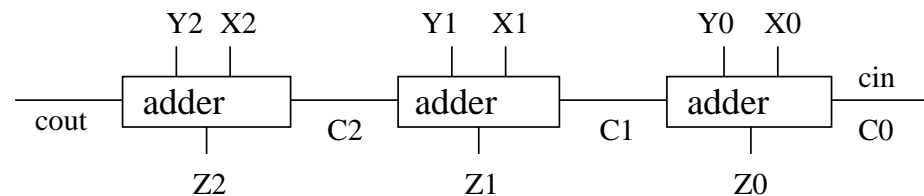
$$\begin{aligned} \text{full\_adder}(a, b, \text{out}, \text{cin}, \text{cout}) &\equiv \\ &\exists w_1, w_2, w_3. \text{xor}(a, b, w_1) \wedge \text{xor}(w_1, \text{cin}, \text{out}) \wedge \text{and}(a, b, w_2) \wedge \\ &\text{and}(\text{cin}, w_1, w_3) \wedge \text{or}(w_3, w_2, \text{cout}) \end{aligned}$$

Can prove theorem (0.01 seconds):

$$\begin{aligned} &\forall a, b, \text{cin}. \exists \text{out}, \text{cout}. \text{full\_adder}(a, b, \text{out}, \text{cin}, \text{cout}) \\ &\wedge \forall o, co. (\text{full\_adder}(a, b, o, \text{cin}, co) \rightarrow ((o \leftrightarrow \text{out}) \wedge (co \leftrightarrow \text{cout}))) \end{aligned}$$

Encoding in QBL. No essential difference to using OBDDs

## Family of adders — structural model



General case:

1. wire together  $n$  1-bit adders where
2.  $i$ th carry-out is  $i+1$ st carry-in
3. first carry is the carry-in and last is the carry-out.

$$\begin{aligned} \text{adder}(n, A, B, S, \text{cin}, \text{cout}) \equiv \\ \exists^2 C. (C(0) \leftrightarrow \text{cin}) \wedge (C(n) \leftrightarrow \text{cout}) \wedge \\ \forall p. p < n \rightarrow \text{full\_adder}(A(p), B(p), S(p), C(p), C(p + 1)) \end{aligned}$$

## Example: behavioral modeling

- Must code up arithmetic over strings

	0	1	2	3	4	
<b>n</b>	0	0	0	0	1	4
<b>A</b>	1	1	0	0	0	3
<b>B</b>	1	0	0	1	0	9
<b>S</b>	1	0	1	1	0	13
<b>cin</b>	1	0	0	0	0	1
<b>cout</b>	0	0	0	0	0	0

- Encodes relation:  $\text{val}(n, S) + 2^n \text{vl}(\text{cout}) = \text{val}(n, A) + \text{val}(n, B) + \text{vl}(\text{cin})$

$$13 + 2^4 * 0 = 3 + 9 + 1$$

## Adder behavior (cont.)

$$\text{val}(n, S) + 2^n \text{vl}(\text{cout}) = \text{val}(n, A) + \text{val}(n, B) + \text{vl}(\text{cin})$$

- Encode arithmetic, encoding functions as relation

$$\begin{aligned} \text{mod\_two}(a, b, c, d) &\equiv a \leftrightarrow b \leftrightarrow c \leftrightarrow d \quad \% d = a + b + c \pmod 2 \\ \text{at\_least\_two}(a, b, c, d) &\equiv d \leftrightarrow (a \wedge b) \vee (b \wedge c) \vee (a \wedge c) \\ \text{add}(A, B, S) &\equiv \exists C. \neg C(0) \wedge (\forall p. \text{mod\_two}(A(p), B(p), C(p), S(p)) \\ &\quad \wedge \text{at\_least\_two}(A(p), B(p), C(p), C(p+1))) \\ \text{val}(n, X, Y) &\equiv \forall p. Y(p) \leftrightarrow (p < n \wedge X(p)) \\ \text{powof2}(n, b, X) &\equiv \forall p. X(p) \leftrightarrow (p = n \wedge b) \quad \% X = 2^n \times b \\ \text{vl}(b, N) &\equiv \forall p. N(p) \leftrightarrow (p = 0 \wedge b) \end{aligned}$$

- Encode behavioral specification

$$\begin{aligned} \text{adder\_beh}(n, A, B, S, \text{cin}, \text{cout}) &\equiv \\ &\exists A', B', S', \text{CIN}, \text{COUT}, U, V, W. \\ &\quad \text{val}(n, S, S') \wedge \text{powof2}(n, \text{cout}, \text{COUT}) \wedge \text{add}(S', \text{COUT}, U) \\ &\quad \wedge \text{val}(n, A, A') \wedge \text{val}(n, B, B') \wedge \text{vl}(\text{cin}, \text{CIN}) \wedge \text{add}(A', B', V) \wedge \text{add}(V, \text{CIN}, W) \\ &\quad \wedge U = W \end{aligned}$$

# Adder — properties

- Correctness:

$$\forall n, A, B, S, cin, cout. \text{adder}(n, A, B, S, cin, cout) \leftrightarrow \text{adder\_beh}(n, A, B, S, cin, cout)$$

- Functional behavior

$$\begin{aligned} &\forall n, A, B, cin. \exists S, cout. \text{adder}(n, A, B, S, cin, cout) \\ &\quad \wedge \forall O, co. (\text{adder}(n, A, B, O, cin, co) \rightarrow (S = O \wedge (cout \leftrightarrow co))) \end{aligned}$$

- Algebraic properties

$$\forall n, A, B, S, cin, cout. \text{adder}(n, A, B, S, cin, cout) \leftrightarrow \text{adder}(n, B, A, S, cin, cout)$$

- Not possible with OBDDs. Typically induction in FOL/HOL.  
All a fraction of a second in Mona.

# Mona script

```

pred XOR(var0 a, b, c) = c <=> (~ a<=> b);  pred OR(var0 a, b, c) = c <=> (a|b);
pred AND(var0 a, b, c) = c <=> (a&b);

pred FA(var0 a, b, s, ci, co) =
  ex0 w1, w2, w3: XOR(a, b, w1) & XOR(w1, ci, s) & AND(a, b, w2) & AND(w1, ci, w3) & OR(w2, w3, co);

pred ADDER(var1 n, var2 A, var2 B, var2 S, var0 ci, var0 co) =
  ex2 C: (0 in C <=> ci) & (n in C <=> co) & (all1 p: p < n => FA(p in A, p in B, p in S, p in C, p + 1 in C));

pred mod_two(var0 a, var0 b, var0 c, var0 d) = a <=> b <=> c <=> d;
pred at_least_two(var0 a, var0 b, var0 c, var0 d) = d <=> a & b | (b & c | a & c);

pred ADD(var2 A, var2 B, var2 S) = ex2 C: 0 notin C &
  (all1 p: mod_two(p in A, p in B, p in C, p in S) & at_least_two(p in A, p in B, p in C, p + 1 in C));

pred VAL(var1 n, var2 X, var2 Y) = all1 p: p in Y <=> (p < n & p in X);
pred POWOF2(var1 n, var0 b, var2 X) = all1 p: p in X <=> (p = n & b);
pred VL(var0 b, var2 N) = all1 p: p in N <=> (p = 0 & b);

pred ADDERSPEC (var1 n, var2 A, var2 B, var2 S, var0 ci, co) =
  ex2 A1, B1, S1, CI, CO, U, V, W: VL(ci, CI) & POWOF2(n, co, CO) & VAL(n, A, A1) & VAL(n, B, B1)
    & VAL(n, S, S1) & ADD(S1, CO, U) & ADD(A1, B1, V) & ADD(V, CI, W) & U = W;

var1 n;  var2 A, B, S;  var0 ci, co;
all1 n: all2 A,B,S: all0 ci,co: ADDER(n, A,B,S,ci,co) <=> ADDERSPEC(n, A,B,S,ci,co) ;

```

The last line is the theorem to be proven. Mona responds:

**ANALYSIS: Formula is valid** Total time: 00:00:00.10

## Intermezzo — arithmetic expressiveness

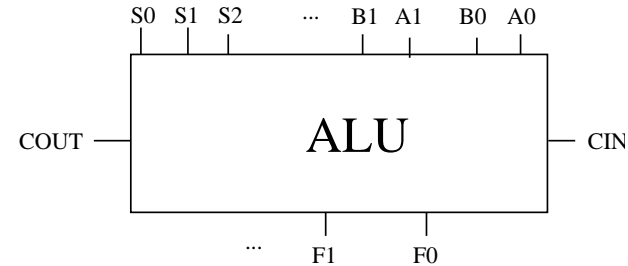
- Addition expressible in both M2L-STR and WS1S. Differ on carry encoding.
- WS1S: direct decision procedure for Presburger arithmetic  
First-order theory of  $\mathcal{N}$  with 0, 1, and +

$$\forall x. \exists y, z. x + y = z \wedge \dots \rightsquigarrow \forall X. \exists Y, Z. plus(X, Y, Z) \wedge \dots$$

- Can augment with inequalities, multiplication by constants, ...
- Questions:
  - Is *plus*( $x, y, z$ ) over **positions** expressible?
  - Is *times*( $X, Y, Z$ ) over **strings** expressible?
  - Is WS1S effective in practice?

See Shiple/Kukula/Ranjan, *A comparison of Presburger Engines for EFSM Reachability* in CAV-98, for a comparison of automata-based procedures (including Mona) with a state-of-the-art polyhedral-based package

# ALU case study



Signals  $S_0$ ,  $S_1$ ,  $S_2$ ,  $cin$  select one of 12 functions

Selection				Output	Function
$s_2$	$s_1$	$s_0$	$cin$		
0	0	0	0	$F = A$	Transfer $A$
0	0	0	1	$F = A + 1$	Increment $A$
0	0	1	0	$F = A + B$	Addition No Carry
0	0	1	1	$F = A + B + 1$	Addition With Carry
0	1	0	0	$F = A - B - 1$	Subtract With Borrow
0	1	0	1	$F = A - B$	Subtract
0	1	1	0	$F = A - 1$	Decrement $A$
0	1	1	1	$F = A$	Transfer $A$
1	0	0	X	$F = A \vee B$	OR
1	0	1	X	$F = A \oplus B$	XOR
1	1	0	X	$F = A \wedge B$	AND
1	1	1	X	$F = \overline{A}$	Complement $A$

## ALU (cont.)

- For a change of pace, let's use M2L-STR!
- Logical part defined directly (length  $n$  implicit)

$$\text{trans}(To, From) \equiv To = From$$

$$\text{OR}(A, B, F) \equiv \forall x : \text{or}(A(x), B(x), F(x))$$

$$\vdots$$

- Addition must be defined differently
  - Let  $\$$  denote last position a string:  $X(\$) \equiv \exists p. \neg \exists q. q = s(p) \wedge X(p)$
  - Define addition using standard “kindergarten algorithm”

$$\begin{aligned} \text{plus}(A, B, C, cin, cout) \equiv & \\ & \exists C. (\forall p. \text{mod\_two}(A(p), B(p), C(p), Out(p)) \\ & \wedge (cout \leftrightarrow \text{at\_least\_two}(A(\$), B(\$), C(\$))) \wedge C(0) \leftrightarrow cin \\ & \wedge ((p < \$) \rightarrow (C(p + 1) \leftrightarrow \text{at\_least\_two}(A(p), B(p), C(p)))))) \end{aligned}$$

## ALU (cont.)

- Arithmetic part defined with *plus*

$$\text{zero}(B) \equiv \forall p : \neg B(p)$$

$$\text{one}(B) \equiv B(0) \wedge \forall p : (p > 0 \rightarrow \neg B(p))$$

$$\text{anc}(A, B, F, \text{cout}) \equiv \exists \text{cin} : \text{zero}(\text{cin}) \wedge \text{plus}(A, B, F, \text{cin}, \text{cout})$$

$$\text{awc}(A, B, F, \text{cout}) \equiv \exists \text{cin} : \text{one}(\text{cin}) \wedge \text{plus}(A, B, F, \text{cin}, \text{cout})$$

⋮

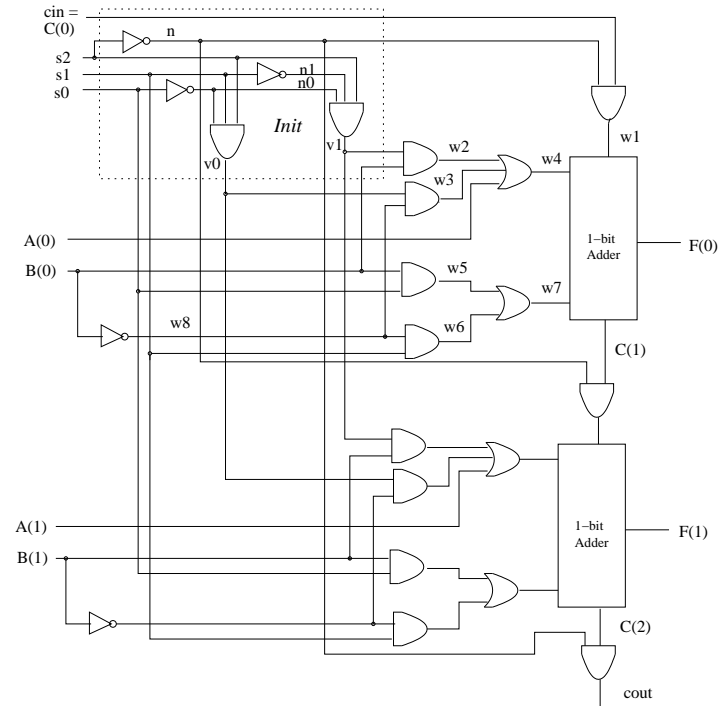
- Entire ALU specified as a “table”:  $\text{if}_4(a, b, c, d, e) \equiv (a \wedge b \wedge c \wedge d) \rightarrow e$

$$\text{n\_alu}(s_0, s_1, s_2, A, B, F, \text{cin}, \text{cout}) \equiv \text{if}_4(\neg s_2, \neg s_1, \neg s_0, \neg \text{cin}, \text{trans}(A, F)) \wedge$$

$$\text{if}_4(\neg s_2, \neg s_1, \neg s_0, \text{cin}, \text{inc}(A, F, \text{cout}))$$

⋮

# ALU implementation



$$\begin{aligned}
 \text{n\_alu}(s_0, s_1, s_2, A, B, F, \text{cin}, \text{cout}) = & \\
 & \exists^2 C, D. \exists v_0, v_1, n. \text{init}(s_0, s_1, s_2, v_0, v_1, n) \\
 & \wedge (\forall p. \text{one\_alu}(A(p), B(p), F(p), C(p), D(p))) \\
 & \wedge (\forall p. (p < \$) \rightarrow (D(p) \leftrightarrow C(p + 1))) \\
 & \wedge (C(0) \leftrightarrow \text{cin}) \wedge (D(\$) \leftrightarrow \text{cout})
 \end{aligned}$$

where `init` and ALU-slice `one_alu` combinational

# Verification

- Correctness theorem (2 seconds):

$$\forall A, B, F. \forall s_0, s_1, s_2, cin, cout. n\_alu(s_0, s_1, s_2, A, B, F, cin, cout) \rightarrow alu\_spec(s_0, s_1, s_2, A, B, F, cin, cout)$$

- Converse is false. Free variables reveal counter-model

$$alu\_spec(s_0, s_1, s_2, A, B, F, cin, cout) \rightarrow n\_alu(s_0, s_1, s_2, A, B, F, cin, cout),$$

- Mona responds:

A counter-example of least length (1) is:

`cout = 1, s2 = 1, s1 = 1, s0 = 1`

Second-order:

A	0
B	X
F	1

Spec says  $F$  is the complement of  $A$  for any value of  $B$  and  $cout$ .  
But  $cout = 1$  not consistent with the implementation.

## Comparison with other methodologies

<i>Tool</i>	<i>Methodology</i>	<i>Proof time</i>
MONA	M2L(Str)	2 seconds
BDDs	Not possible	
PVS (SRI)	Tactic + BDDs	2 minutes
CLAM (Edinburgh)	Induction theorem prover	6 minutes

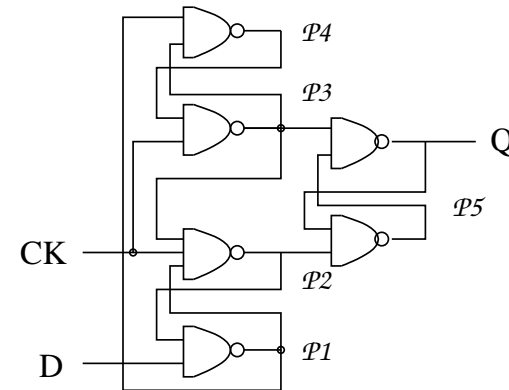
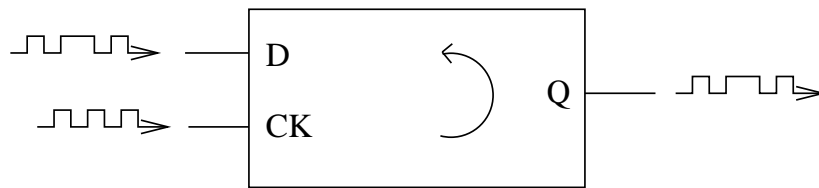
- Numbers should **NOT** be over-emphasized.
- Not bad for a non-elementary time decision procedure!

Is this a coincidence ?!?

# Sequential systems

- Formalize time dependent behavior
  - $X(0) \dots X(n - 1)$ : evolution of  $X$  over  $n$  time-units
- Can formulate safety properties
  - WS1S: properties holding over all finite time intervals
  - S1S: properties of finite and infinite time intervals
- Case study: **edge-triggered D-type flipflop**
  - Shows embedding of sequential behavior in M2L(Str)
  - Clarifies counter-example generation
  - Suggests how Mona can be used as a development tool.

# D-Flipflop



- Synchronous system with feedback
- Structurally trivial — 6 nand gates!

$\text{dtype\_imp}(D, CK, Q) \equiv$

$\exists P_1, P_2, P_3, P_4, P_5. \text{nand}(P_2, D, P_1) \wedge \text{nand}_3(P_3, CK, P_1, P_2)$

$\wedge \text{nand}(P_4, CK, P_3) \wedge \text{nand}(P_1, P_3, P_4) \wedge \text{nand}(P_3, P_5, Q) \wedge \text{nand}(Q, P_2, P_5)$

- Circuit model: unit gate delay

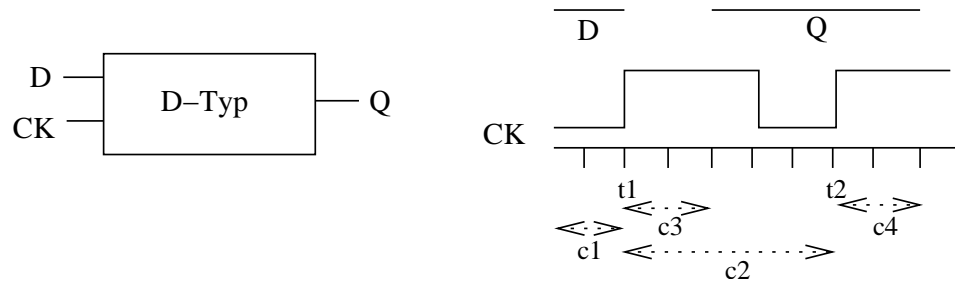
$$\text{nand}(I_1, I_2, O) \equiv \forall t. O(t+1) \leftrightarrow \neg(I_1(t) \wedge I_2(t))$$

## Behavior is difficult to understand

It turns out, on analysis, that the *modus operandi* of this circuit is far from simple: in fact, it is unusually complex, and (so the authors found) difficult to understand intuitively. If, like most people, you find this remark difficult to accept at face value, read the rest of this account, then set it aside, and attempt, within (say) one working day, to come up with a carefully justified account of 'how' the proposed implementation is intended to function.

— Hanna and Daeche 1986

## Behavior: informal



Specification: Mike Gordon "Formal Aspects of VLSI Design" 1986

### If

1. Clock  $CK$  has rising edge at time  $t_1$ , and
2. next rising edge is at time  $t_2$ , and
3.  $D$  is stable  $c_1$  time-units before  $t_1$  (Setup time), and
4. at least  $c_2$  time-units between  $t_1$  and  $t_2$

### Then

1.  $Q$  is stable between times  $t_1 + c_3$  and  $t_2 + c_4$  and,
2. between  $t_1 + c_3$  and  $t_2 + c_4$ ,  $Q$  has the same value as  $D$  during the setup-time

## Behavior: formal

- First we encode a bit of interval temporal logic

**the value of  $F$  is stable in  $[t_1, t_2]$ :**

$$\text{stable}(t_1, t_2, F) \equiv \forall t. t_1 \leq t \leq t_2 \rightarrow (F(t) \leftrightarrow F(t_1))$$

**$t_2$  is the first instant after  $t_1$  when  $F$  becomes high:**

$$\text{next}(t_1, t_2, F) \equiv t_1 < t_2 \wedge F(t_2) \wedge (\forall t. t_1 < t < t_2 \rightarrow \neg F(t))$$

**$F$  rises/falls at  $t$ :**

$$\text{rise}(t, F) \equiv t > 0 \wedge (\neg F(t-1) \wedge F(t))$$

$$\text{fall}(t, F) \equiv t > 0 \wedge (F(t-1) \wedge \neg F(t))$$

- **Behavioral Specification** (schematic w.r.t.  $(c_1, c_2, c_3)$ )

$$\text{dtype}(c_1, c_2, c_3, c_3)(D, CK, Q) \equiv$$

$$\forall t_1, t_2. (\text{rise}(t_1, CK) \wedge \text{next}(t_1, t_2, CK) \wedge \text{stable}(t_1 - c_1, t_1, D) \wedge (t_2 > c_2 + t_1) \\ \rightarrow (\text{stable}(t_1 + c_3, t_2 + c_4, Q) \wedge Q(t_2) = D(t_1)))$$

## Verification

- Mike Gordon writes (“Formal Aspects ...”, 1986):  
“It can be proved that

$$\text{dtype\_imp}(D, CK, Q) \rightarrow \text{dtype}(2, 3, 4, 1)(D, CK, Q)$$

The formal proof of this is fairly complicated. It has been done by hand by ...”

- MONA says: There is a minimal counter-example of length 8 with free variables for  $D, CK, Q, P_1, \dots, P_5$

$D$	$CK$	$Q$	$P_1$	$P_2$	$P_3$	$P_4$	$P_5$
1	1	1	0	1	1	1	1
1	0	0	0	1	0	1	0
0	1	1	0	1	1	1	1
1	1	0	1	1	0	1	0
0	0	1	0	1	0	1	1
1	0	1	1	1	1	1	0
0	1	1	0	1	1	0	0
0	0	1	1	1	1	1	0

## The problem

- Unstable signals (e.g.,  $CK$ ) cause oscillation
- Analysis of counterexamples yielded sufficient conditions for stability
- With new 'environment' verification in 2 second

$$\text{dtype\_imp} \wedge \text{input\_requirements} \rightarrow \text{dtype}$$

- Problems also found by Wilk and Pnueli using linear-tense logic

They gave an alternative formalization of Gordon's requirements as Direct formulation/translation could involve a non-elementary blowup.

(see Wilk/Pnueli, IEEE/ACM Conference on Computer Aided Design, 1989)

## Parametric iterative circuits

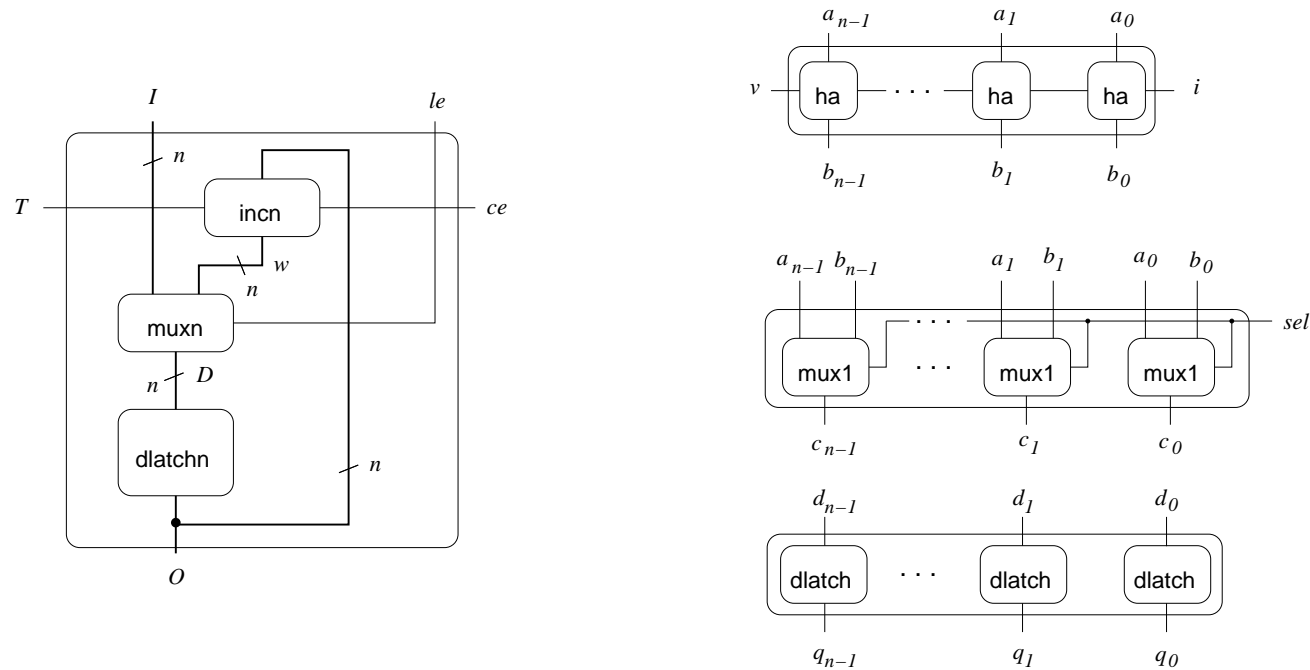
- Parameter allows one unbounded dimension
  - **Size** of function/relation
  - **Time** upon which behavior depends
- Both together?
  - Not in a **monadic** logic:  $X(p)$  not  $X(p_1, p_2)$
  - Most extension to binary relations undecidable...

0100101	⟨ TM head at time $i$ ⟩	01001
	↓	
01001011	⟨ TM head at time $i + 1$ ⟩	1001

... because TM computations can be encoded on grids

- Solution: use meta-reasoning to eliminate the time dimension

# Case study: ripple-carry counter



Behavior: (CE is count enable and LE is load enable)

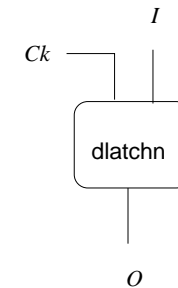
CE	LE	Action (Effect)
X	H	Load
H	L	Increment (mod $n$ )
L	L	Hold (previous value)

and  $T$  set on incrementer overflow

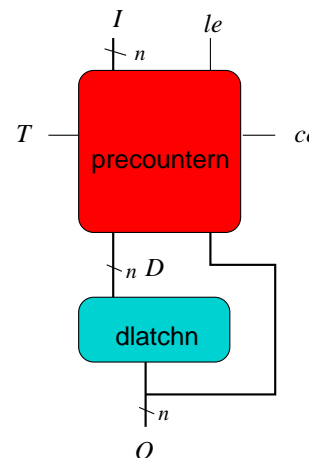
## Parameterized counter

- Time dependent: increment and hold depend on previous output

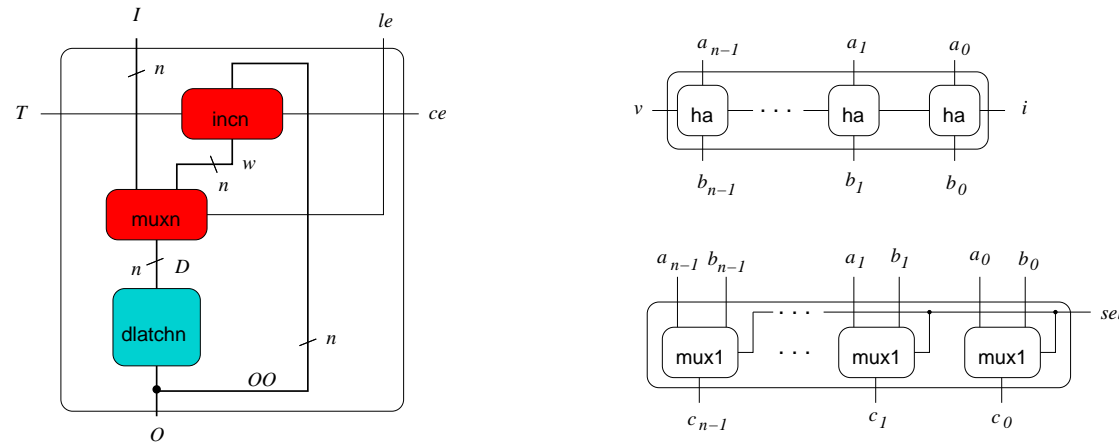
$I(t)$	$O(t+1)$	
0	0	Clear to 0
1	1	Set to 1



- Time dependency is simpler than data dependencies
- Idea/Metareasoning:** Eliminate clock/latch. Sufficient to require correct behavior over any two consecutive time units.



# Implementation



```

pred xor(var0 a,b) = ((a | b) & ~(a & b));
pred mux1(var0 a,b,o,sel) = (sel => (o <=> a)) & (~sel => (o <=> b));
pred ha(var0 a,b,i,v) = (b <=> xor(a,i)) & (v <=> (a & i));

```

```

pred incn(var2 A,B, var0 ce,t) = ex2 C, D:
  (all1 p: ha(p in A, p in B, p in C, p in D))
  & (all1 p: (p < $) => (p in D <=> (p +o 1 in C)))
  & (0 in C <=> ce) & ($ in D <=> t);

```

```

pred muxn(var2 A,B,0, var0 sel) = all1 p: mux1(p in A, p in B, p in 0, sel);

```

```

pred precount(var0 le, ce, t, var2 I, D, OO) =
  ex2 W: incn(OO, W, ce, t) & muxn(I,W,D,le);

```

## Specification and verification (.27 seconds)

CE	LE	Action (Effect)
X	H	Load
H	L	Increment (mod $n$ )
L	L	Hold (previous value)

```
pred if(var0 cond, then, else) = (cond => then) & (~cond => else);
```

```
pred inc(var2 Old, New, var0 t) =
  if(all1 p: p in Old,
    t & New = empty,
    ~t & ex1 j : j notin Old & (all1 k : k < j => k in Old) &
      all1 l : (l < j => l notin New) & (l = j => l in New)
      & (l > j => (l in Old <=> l in New)));
```

```
pred spec(var0 le, ce, t, var2 I, D, 00) =
  if(le, D = I, if(ce, inc(00,D,t), D = 00));
```

```
precount(le, ce, t, I, D, 00) => spec(le, ce, t, I, D, 00);
```

## Multiple parameters — conclusions

- Reasoning about multiple parameters is possible, but not entirely in WS1S
- Can use a stronger logic + reduction
  - Example was carried out Isabelle/HOL
  - WS1S was embedded in HOL and Mona used as an oracle.
  - Details in [Combining WS1S and HOL, D.B./Friedrich, FRODOS 2000](#)
- No general reduction method possible but induction can suffice to eliminate a parameter

## Other kinds of examples

⇒ Regular Model Checking and Iterative Procedures in WS1S

- Modeling/Verifying Tree Structured Circuits in WS2S

See [Ayari/D.B./Friedrich, 1999](#) for examples, e.g., CLA.

⇒ Formal Design Constraints for CORBA

See [Klarlund/Koistinen/Schwartzbach, OOPSLA '96](#)

- Parsing with Logical Side Constraints

See [Damgaard/Klarlund/Schwartzbach, DLT'99](#)

- Analysis of Natural Language Grammars

See, e.g., [The Logic-Automaton Connection in Linguistics](#) by [Cornell/Morawietz, LNAI 1582](#).

- Reasoning about C Programs with Pointers

See [MONA web page](#) for various publications

## Regular model checking — Idea

Kesten/Maler/Marcus/Pnueli/Shahar, CAV'97

Model checking based on iteration from final (or starting states) of a system  $P$  to establish a property  $\Box g$ .

Procedure Symb-MC( $g$ : assertion);

  assertion:  $\phi_0, \phi_1, \dots$

  Let  $\phi_0 := \neg g$ ;

  For  $i = 0, 1, \dots$  repeat

    Let  $\phi_{i+1} := \phi_{-i} \vee \text{pred}_P(\phi_i)$ ;

  until  $\phi_{i+1} = \phi_i$ ;

  Check that  $\phi_i \wedge \text{init}_P = F$ ;

Requires language  $\mathcal{L}$ :

- where properties  $g$  and  $\text{init}_P$  are expressible in  $\mathcal{L}$
- that is effectively closed under negation and disjunction
- where the predicate transformer is definable and equivalence is decidable

## Regular model checking (cont.)

- Regular languages have these properties

KMMPS used regular expressions and built a system using regular transducers

- One can use WS1S directly

$$\begin{aligned} \mathit{init}_P(S) &\equiv \dots \\ \mathit{trans}(S_1, S_2) &\equiv \dots \end{aligned}$$

- But iteration is problematic

$$\begin{aligned} \mathit{reachable}_{\mathit{trans}}(S_1, S_2) &\equiv S_1 = S_2 \vee \mathit{trans}(S_1, S_2) \vee \\ &(\exists T. \mathit{trans}(S_1, T) \wedge \mathit{trans}(T, S_2)) \vee \dots \end{aligned}$$

Reachable not formalizable in WS1S (why?). Must approximate.

## Regular model checking (cont.)

FMona formalization [Bodeveix/Filali, TACAS 2000](#).

```
pred iterate(type State, var nat N, pred(var State s,s') tr,
             pred(var State s) init, var State s) =
  if N = 0 then init(s)
  else ex State s': iterate(N-1,tr,init,s') & (s'=s | tr(s',s)) endif;
```

```
pred check_bwd(type State, pred(var State s,s') tr,
               pred(var State s) bad, pred(var State s) init) =
  stable(bad,inverse(tr)) & all State s: init(s) => ~bad(s);
```

```
pred backward(type State, var nat N, pred(var State s,s') tr
              pred(var State s) init, pred(var State s) g) =
  check_bwd(tr, iterate(N, inverse(tr), Not(g)), init);
```

Can apply to problems of unbounded size/length ([infinite state model checking](#)).  
 Moreover, can declaratively formalize and exploit [acceleration techniques](#).

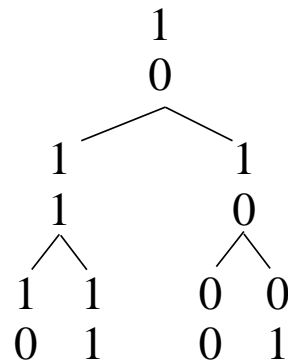
## Applications based on WS2S

- Generalization to two successors unproblematic
- Example:

$$X(\epsilon) \wedge (\forall p. X(p.0) \leftrightarrow X(p.1)) \wedge \forall p. \neg Y(p.0) \vee \neg Y(p.1)$$

$X$  contains the root position  $\epsilon$  and that a position  $p$  is in  $X$  iff its brother is also in  $X$ . Moreover, for any  $p$ ,  $Y$  contains at most one of  $p$ 's successors.

- Semantics based on trees (and tree automata)



- Trees can represent behavior of hardware, parse trees, etc.

## Example: formal design constraints

Klarlund/Koistinen/Schwartzbach, OOPSLA'96

**Problem:** OO-Design criteria include **Design Constraints**

E.g. OMG/CORBA compliant designs require things like

*Classes with persistent instances should inherit only from other classes with persistent instances and should not provide asynchronous operations.*

*Classes abstracting hardware resources should provide asynchronous operations and respond through an event channel.*

**Idea:** provide a **constraint language** for **formalizing** and **checking constraints**

- Formalization done in sugared version of WS2S
- Resulting tree automata used to check constraints on program parse trees

## Design constraints (cont.)

- Example: Operation arguments of interfaces cannot denote object values

*If  $x$  is an operation argument node in the syntax tree of a **CORBA** object type, then the node  $y$  below denoting its type cannot represent an object type.*

Formalized as (for full details see paper):

**Constraint corba for**

ObjectTypeSpecification **IS**

$\forall x: \text{Argument}. \exists y: \text{Type}. x \triangleleft y \wedge \neg \text{OT}(y);$

where `ObjectTypeSpecification`, `Argument`, and `Type` are production names of the syntax, and `OT(y)` is true when `y` is a node denoting an object type. The expression `x  $\triangleleft$  y` holds when node `x` is the parent of node `y`.

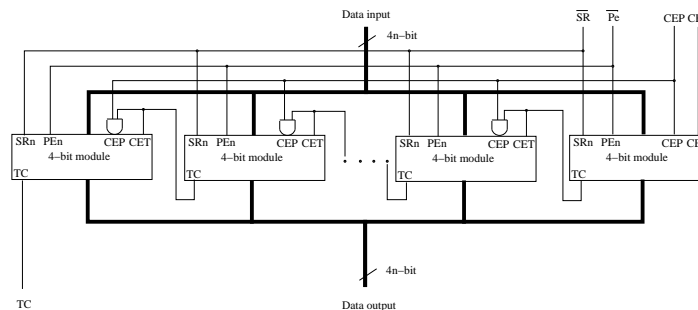
- Constraint translated to WS2S and compiled to a tree automaton
- Idea has general applicability, e.g., parsing with logical constraints.

# Road map

- Motivation/background
  - Syntax, semantics, decidability.
  - Applications
- ⇒ Complexity/alternatives

## Some statistics

- Tiny example: ripple-carry adder
  - 109 product and projection operations
  - average: 5 states, 12 BDD nodes
  - largest: 21 states, 71 BDD nodes
- Small example: 74LS163 4-bit counter (ripple-carry)



- Requires 4 seconds and computes 827 automata
- Average: 47 states, 189 BDD nodes
- Largest: 3,037 states, 12,865 BDD nodes.

## Statistics (cont.)

From *MONA Implementation Secrets*, Klarlund/Møller/Schwartzbach, CIAA'00

Name	Size	Logic	Time	Space
dfilopflop	2 KB	WS1S (M2L-Str)	0.4 sec	3 MB
Euclid	6 KB	WS1S (Presburger)	33	217 MB
fisher_mutex	43 KB	WS1s	15	13
lift_controller	36 KB	WS1S	8 sec	15 MB
szymanski_acc	144 KB	WS1S	20	9 MB
von_neumann_adder	5 KB	WS1S	139 sec	116 MB
search_tree	19 KB	WS2S	30 sec	5 MB
html3_grammar	39 KB	WS2S	137 sec	208 MB
xbar_theory	14 KB	WS2S	136 sec	518 MB

**Euclid** Encoding of reachability on a machine implementing Euclid's GCD algorithm (Shiple)

**Fisher\_mutex and lift\_controller** Translated Duration Calculus encodings (Pandya)

**Szymanski\_acc** iterated analysis of Szymanski Problem (Filali)

**von\_neumann** Equivalence of 8-bit von Neumann adder with carry-chain adder (Mödersheim)

**search\_tree** verifies a C program that deletes a search treenode (Klarlund)

**html3\_grammar** WS2S encoding of HTML3.0 Grammar (Damgaard)

**xbar** WS2S encoding of part of a theory of natural languages (Morawietz)

## Why does it work? — state space explosion

- Non-elementary blowup associated with subset construction
  - Quantifier alternations potentially deadly!
  - But running times often very reasonable
- An example

$$at\_least\_two(a, b, c) \equiv (a \wedge b) \vee (a \wedge c) \vee (b \wedge c)$$

$$mod\_two(a, b, c, d) \equiv a \leftrightarrow b \leftrightarrow c \leftrightarrow d$$

$$add(A, B, Out, cin, cout) \equiv$$

$$\exists C. \ \$ \geq 0 \Rightarrow$$

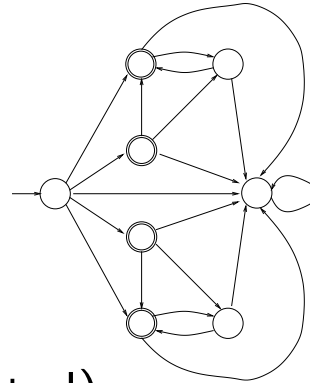
$$\left( \forall p. mod\_two(A(p), B(p), C(p), Out(p)) \right)$$

$$\wedge \left( (p < \$) \rightarrow (C(p+1) \leftrightarrow at\_least\_two(A(p), B(p), C(p))) \right)$$

$$\wedge (cout \leftrightarrow at\_least\_two(A(\$), B(\$), C(\$))) \wedge C(0) \leftrightarrow cin$$

## Why does it work? — 2nd-order quantification

- Focus on computation related to projection in  $\exists C. \phi$



8 states (& 32 BDD nodes, omitted)

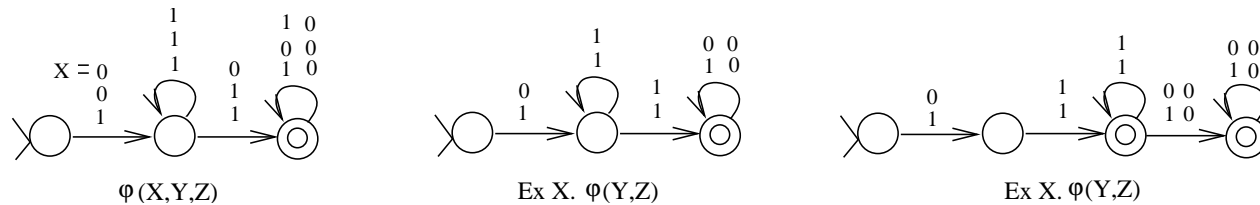
- Intuition:
  - 4 final states correspond to initial transition (after reading 2 booleans)
  - mod 2 counters. E.g., for  $cin = cout = 0$

$$\begin{array}{|c|} \hline 1 \\ \hline 1 \\ \hline 0 \\ \hline \end{array} \notin L \text{ but } \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \in L$$

- 2 pairs since it must remember carry-out
- right-most state is dead-state

## 2nd-order quantification (cont.)

- $A_{\exists C}. \phi$  requires project & subset construction  
Recall how projection works (here for  $X$ )

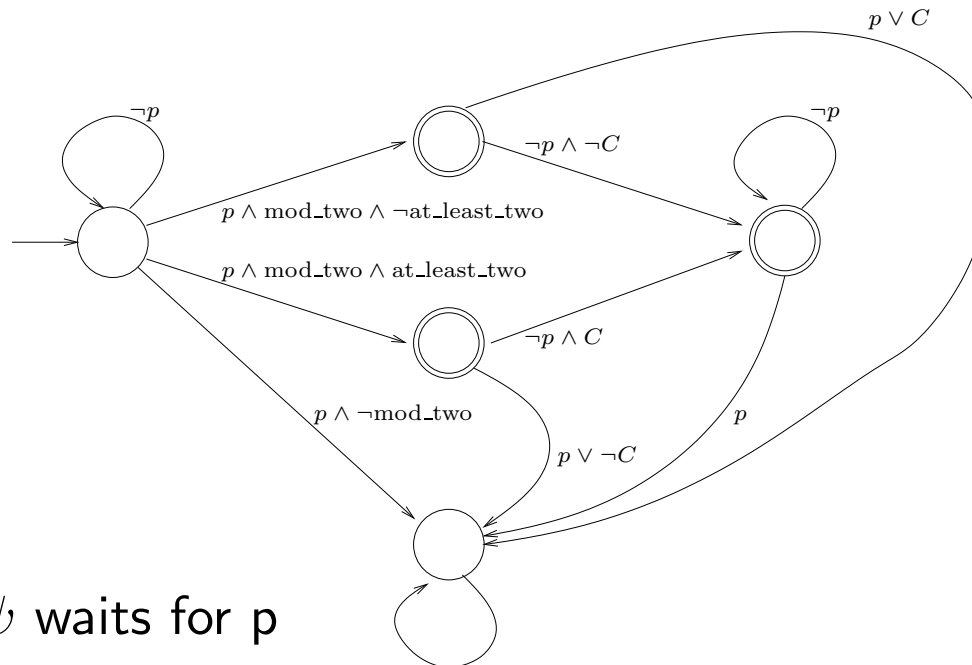


- Carry-string functionally determined from others  $\Rightarrow$  no blowup!
- In example: 8 states increase to 10 (initial transition complication)
- **Proposition:** Let  $\phi = \exists X. \psi(P)$ , where  $P$  is functionally determined (for any interpretation of other free variables in  $\psi$ , there is exactly one interpretation of  $X$  making  $\psi$  true). Then, the subset automaton for  $\phi$  is linear in  $|A_\psi|$ .
- Useful semantic characterization: e.g., iterative architectures with functional information flow

## Limited 1st-order quantification, $\phi \equiv \forall p. \psi$

1st-order variables are singleton set-variables

$$\psi \equiv \text{mod\_two}(A(p), B(p), C(p), \text{Out}(p)) \wedge ((p < \$) \rightarrow (C(p + 1) \leftrightarrow \text{at\_least\_two}(A(p), B(p), C(p))))$$

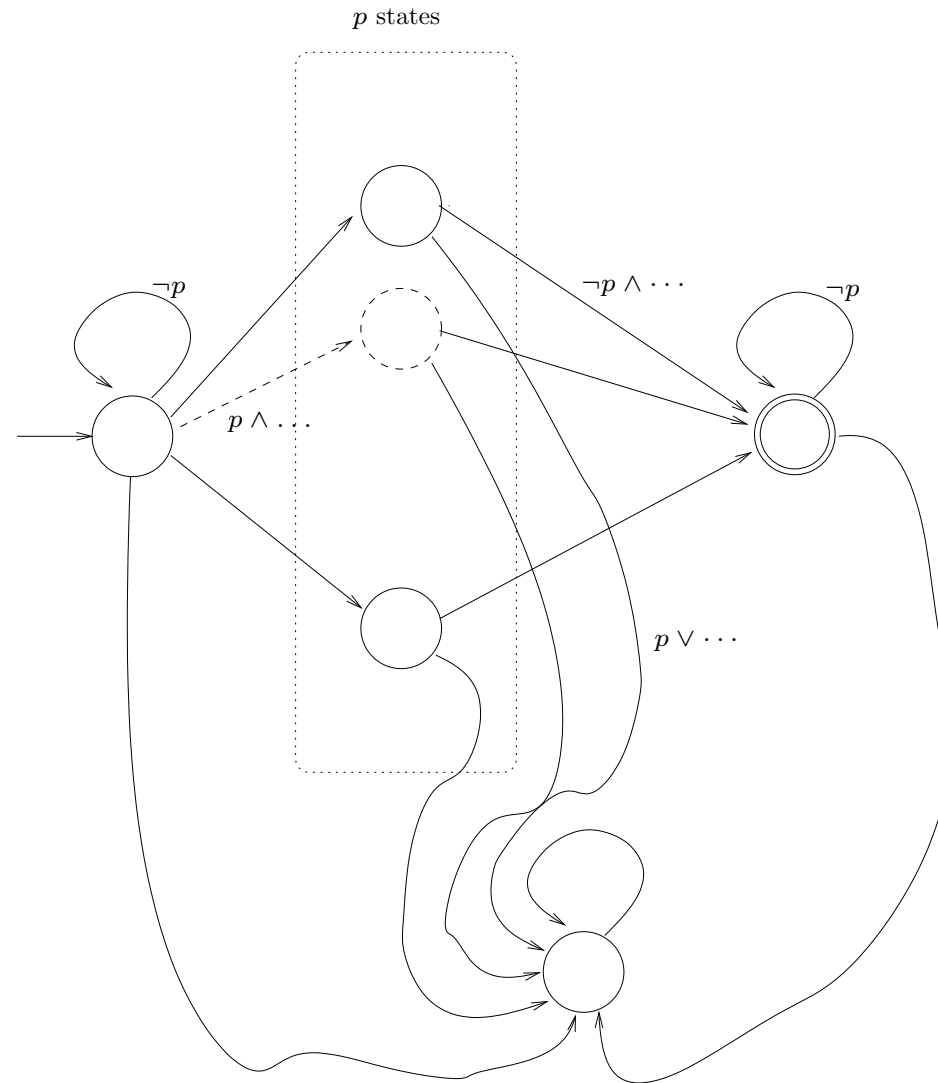


1. Automaton for  $\psi$  waits for  $p$
2. Rejects if the *mod\_two* predicate false at position  $p$ , or
3. branches if the *mod\_two* predicate holds to a state remembering the value *at\_least\_two* at  $p$  and checks  $p + 1$  value.

## 1st-order quantification (cont.)

- Subset automaton after  $p$ -project is small (approx. same size)
- Consider more general case:  
 $\psi$  equivalent to a Boolean combination of formulas of the form  $p \in X_i$  or  $p \leq \$ \Rightarrow p + 1 \in X_i$ .
- Automaton  $A$  must:
  1. Before  $p$  occurs,  $\psi$  says nothing about any other variable.
  2. At  $p$  a new state (in dotted box) is reached determined by  $X_i$ s at  $p$ .
  3. if  $p$  is not the last position, truth of  $\psi$  is determined  $X_i$ s at  $p + 1$ .

# 1st-order quantification



## 1st-order quantification

Reachable states of  $A$  in the subset construction are those:

$\{s \mid \text{for some } \alpha, s \text{ is the state reached when some } p\text{-track is added to } \alpha\}$ ,

where  $\alpha$  determines an interpretation of the  $X_i$ .

Any such set contains at most one state from the box in the figure above. Hence, there is only a linear expansion.

**Proposition:** For  $\phi \equiv \exists p. \psi(p, \{X_i\})$ , where  $\psi$  only mentions  $p$  in terms that are of the form  $p$  or the form  $p + 1$  (where in the latter case, the occurrence is under the provision  $p \leq \$$ ), the calculation of the subset automaton for  $\phi$  is linear in the size of the automaton for  $\psi$ .

## Above proposition generally applicable

- Transition System Encodings:  $\forall p < \$ : St(p) \wedge In(p) \Rightarrow St'(p + 1) \wedge \dots$

- Linear Inductive Functions (Gupta/Fisher)

$$\text{for } i = 1 \quad sum_1 = a_1 \oplus b_1 \oplus cin$$

$$carry_1 = (a_1 \wedge b_1) \vee ((a_1 \vee b_1) \wedge cin)$$

$$\text{for } i > 1 \quad sum_i = a_i \oplus b_i \oplus carry_{i-1}$$

$$carry_i = (a_i \wedge b_i) \vee ((a_i \vee b_i) \wedge carry_{i-1})$$

Expressed in WS1S as:

$$add(A, B, Sum, Carry, cin) \equiv \forall i :$$

$$i = 0 \rightarrow$$

$$Sum(0) \leftrightarrow xor(A(0), xor(B(0), cin)) \wedge$$

$$Carry(0) \leftrightarrow (A(0) \wedge B(0)) \vee ((A(0) \vee B(0)) \wedge cin)$$

$$0 < i \rightarrow$$

$$Sum(i) \leftrightarrow xor(A(i), xor(B(i), Carry(i - 1))) \wedge$$

$$Carry(i) \leftrightarrow (A(i) \wedge B(i)) \vee ((A(i) \vee B(i)) \wedge Carry(i - 1))$$

**Result:** Same complexity as LIF algorithms

## Complexity (summary, so far)

- Good complexity guarantees are possible

You get what you pay for

and Mona can subsume specialized algorithms.

- But sometimes blowups cannot be avoided. Nonelementary is nonelementary.
- But not all problems are well-behaved!
- Question: Can we still analyze problems and benefit from the simplicity/conciseness/... of monadic logics?

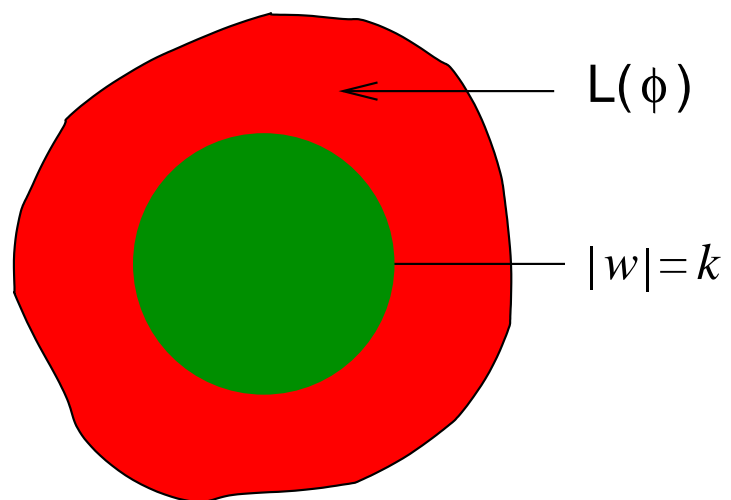
## Goal: quick error detection

- Input: specification (e.g., circuit + expected behavior)
- Output: a counter example (when it exists)
- Questions:
  - Is this possible?
  - For which logic?
  - How would this work?
  - Is it feasible?

## Bounded model construction

INSTANCE: A formula  $\phi$  and  $k \in \mathbb{N}$

QUESTION: Is there  $w$  such that  $|w| = k$  and  $w$  satisfies  $\phi$ ?



## BMC for M2L-STR

- **Recall minimal syntax:**

$$\phi ::= X \subseteq Y \mid \text{succ}(X, Y) \mid \exists X. \phi \mid \neg \phi \mid \phi_1 \wedge \phi_2$$

- **Idea:**  $M \subseteq [k]$  encoded by the Booleans  $b_0, \dots, b_{k-1}$ :  $i \in M$  iff  $b_i$  is true
- **Translation:**  $[\cdot]_k : \text{MSO} \longrightarrow \text{QBF}$

$$\begin{aligned} [X \subseteq Y]_k &= \bigwedge_{0 \leq i \leq k-1} (x_i \rightarrow y_i) \\ [\text{succ}(X, Y)]_k &= \text{singleton}(x_0, \dots, x_{k-1}) \wedge \text{singleton}(y_0, \dots, y_{k-1}) \wedge \\ &\quad \bigvee_{0 \leq i < k-1} (x_i \rightarrow y_{i+1}) \\ [\phi_1 \wedge \phi_2]_k &= [\phi_1]_k \wedge [\phi_2]_k \\ [\neg \phi]_k &= \neg [\phi]_k \\ [\exists X. \phi]_k &= \exists x_0 \dots x_{k-1}. [\phi]_k \end{aligned}$$

- **Complexity:**  $[\cdot]_k$  translation requires polynomial time

## Results for BMC for M2L-STR

### Theorem (Correctness)

For  $\phi$  a MSO formula.

$$\models_{\text{M2L}} \phi \quad \text{iff} \quad \models_{\text{QBL}} [\phi]_k, \text{ for all } k \geq 0$$

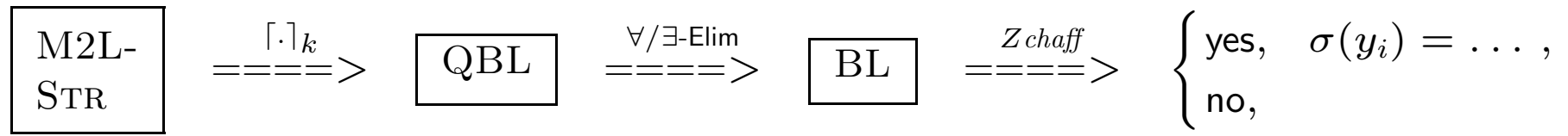
### Theorem (Complexity)

BMC for M2L-STR is PSPACE-complete.

Proof:

- membership:  $[\cdot]_k$  can be tested in polynomial space.
- hardness:
  - \* QBL can be encoded in M2L-STR and
  - \* QBL-satisfiability can be reduced to finding a model of length 1

# Implementation of BMC for M2L-STR



Examples	MONA	BMC	
	sec	$k$	sec
Ripple-carry adder	0.11	5	0.21
Mutual exclusion	0.58	9	1.07
FlipFlop	0.20	7	0.20
Barrel Shifter (8)	1.03	6	0.27
Barrel Shifter (32)	abort	6	0.90
Barrel Shifter (64)	abort	6	4.49
Counter (15)	242.17	8	0.70
Counter (32)	abort	8	3.42

## Byte code verification (buggy code)

<i>Examples</i>	<i>States</i> #	<i>Spin</i> sec	<i>Smv</i> sec	<i>BMC</i>	
				sec	<i>k</i>
StringBuffer_substring_I_Ljava_lang_String	$10^4$	0	0	0	3
Object_toString__Ljava_lang_String	$10^5$	0	0	3	10
Integer_toString_II_Ljava_lang_String	$10^{10}$	⊥	0	134	18
StringBuffer_append_C_Ljava_lang_StringBuffer	$10^8$	⊥	1	22	22
Compiler_S_clinit___V	$10^8$	⊥	4	76	22
FDBigInt_longValue__J	$10^{18}$	⊥	5	115	17
Object_wait_JI_V	$10^{10}$	⊥	26	21	11
StringBuffer_insert_I[C_Ljava_lang_StringBuffer	$10^{11}$	⊥	34	617	36
FDBigInt_mult_I_Ljava_lang_FDBigInt	$10^{18}$	⊥	⊥	43	5
FDBigInt_multaddMe_II_V	$10^{19}$	⊥	⊥	170	13
Character_S_clinit___V	$10^{16}$	⊥	⊥	268	19
Long_parseLong_Ljava_lang_StringI_J	$10^{19}$	⊥	⊥	415	10
Short_decode_Ljava_lang_String_Ljava_lang_Short	$10^{12}$	⊥	⊥	727	30
FDBigInt_sub_Ljava_lang_FDBigInt_Ljava_lang_FDBigInt	$10^{19}$	⊥	⊥	343	16
String_toLowerCase_Ljava_util_Locale_Ljava_lang_String	$10^{14}$	⊥	⊥	200	26
Integer_decode_Ljava_lang_String_Ljava_lang_Integer	$10^{12}$	⊥	⊥	303	303

# BMC for WS1S

**Theorem** BMC for WS1S is nonelementary

Proof:

- closed formulae are either valid or unsatisfiable
- closed formula  $\phi$  has a model of length  $k$  iff  $\phi$  is valid
- validity in WS1S is nonelementary

**Corollary** BMC is nonelementary for

- WFO[<] (first-order fragment of WS1S)
- S1S (like WS1S but second-order variables range over infinite subsets of  $\mathbb{N}$ )
- FO[<] (first-order fragment of S1S)

# Conclusions

- Regular systems have wide application
  - Infinite families of systems
  - ‘Evolving’ (but regular) behavior
  - Tree (and even some graph) structures.
- Monadic logics approach the dream
  - Simplicity: natural extension of (Q)BL with quantifiers. Good for practitioners
  - Wide range of applications and application domains
  - Basis for prototyping ideas before building specialized tools
  - Complexity can be acceptable and error detection is fast

# Thanks

Abdelwaheb Ayari

Felix Klaedtke

Nils Klarlund

Stefan Friedrich

Sebastian Mödersheim

## Literature — pointers

- **Background Theory, History, etc.**

W. Thomas, *Automata on Infinite Objects*, Handbook of Theoretical Computer Science, 1990.

W.Thomas, *Languages, Automata, and Logic*, Handbook of Formal Language Theory, 1997.

- **Mona/Documentation/Implementation**

See Mona web page at [www.brics.dk/mona](http://www.brics.dk/mona)

- **Monadic Logics, Circuits, BMC**

See publications at [www.informatik.uni-freiburg.de/~basin/pubs/pubs.html](http://www.informatik.uni-freiburg.de/~basin/pubs/pubs.html)

- **Regular Model Checking**

Y. Kesten, O. Maler, M. Marcus, A. Pnueli and E. Shohar, *Symbolic Model Checking with Rich Assertional Languages*, CAV'97.

A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. *Regular model checking*, CAV'00.

J.P. Bodeveix, M. Filali. *FMona: a tool for expressing validation techniques over infinite state systems*, TACAS 2000.