Monadic Logics, Automata, BDDs, SAT Procedures, and Enormous Search Spaces

David Basin
University of Freiburg

Overview

Goal: Motivate monadic logics as a specification framework that supports different representation and search techniques.

Thesis: Monadic logics are simple, very expressive, and many problems can be quickly analyzed.

Road map:

- The automata/logic connection
- An example
- Bounded model construction

N.B.: heterogeneous audience $\wedge$ limited time $\Longrightarrow$ high level presentation

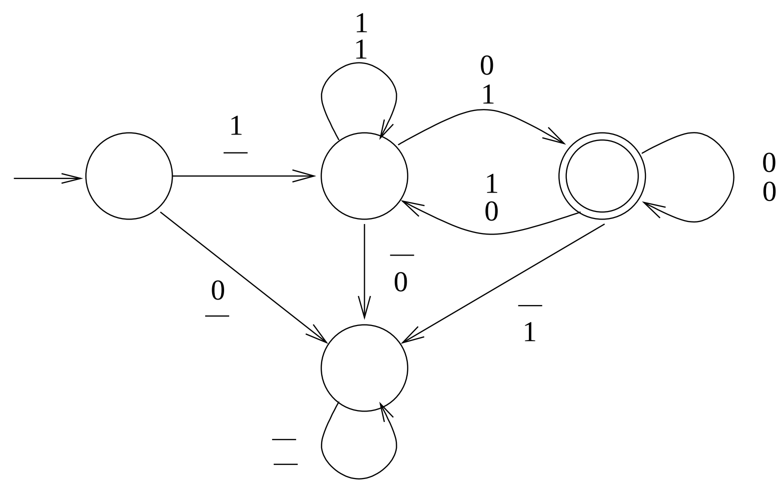
The automata/logic connection

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

Logic	Languages	Automata
M2L-Str, WS1S	regular languages	automata
S1S	ω-regular languages	Büchi-automata
(W)S2S	regular tree languages	tree-automata

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

$\Leftrightarrow$



Syntax for Monadic Logics on Strings

- Let V_1 and V_2 be disjoint sets of first and second-order variables.

- Syntax:

$$
\begin{array}{lcll}
t & ::= & 0 \mid \$ \mid p \mid \mathsf{s}(t) & p \in V \\
\phi & ::= & X(t) \mid \; t = t \mid t < t \mid X = Y & \\
 & & \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \exists p.\, \phi \mid \forall p.\, \phi \mid \; \mid \exists X.\, \phi \mid \; \ldots & p \in \mathcal{V}_1 \text{ and } X, Y \in \mathcal{V}
\end{array}
$$

- An example

$$X(\mathsf{0}) \wedge \forall p < \$.\, (X(p) \leftrightarrow Y(\mathsf{s}(p)))$$

Word-Model Semantics (M2L-STR, MSO[S])

- A formula with n free variables is interpreted over a word in $(\mathbb{B}^n)^*$

$$\begin{array}{c} X \\ Y \end{array} \quad \begin{array}{|c|c|c|} \hline 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline \end{array} \models \quad X(0) \wedge \forall p < \$.\, (X(p) \leftrightarrow Y(\mathsf{s}(p)))$$

- First-order variables interpreted by "tracks" with a single bit set

$$\begin{array}{c} p \\ r \\ X \end{array} \quad \begin{array}{|c|c|c|} \hline 1 & 0 & 0 \\ \hline 0 & 0 & 1 \\ \hline 1 & 1 & 0 \\ \hline \end{array} \models \quad \exists q.\, p < q \wedge q < r \wedge X(q)$$

- Semantics of functions/predicates/quantifiers as expected

  Successor requires convention to handle "last position"

- Validity: $\models \phi$ iff $s \models \phi$ for all strings (over alphabet $B^{|fv(\phi)|}$)

$$\models \forall X.\, \exists Y.\, \forall p < \$.\, (X(p) \leftrightarrow Y(\mathsf{s}(p)))$$

Decidability

Büchi/Elgot/Trahktenbrot For every $\mathrm{M2L}\text{-}\mathrm{S}\textsc{tr}$ formula ϕ, with free variables $X_1, \ldots X_k$, the language $\mathcal{L}(\phi) \subseteq (\{0,1\}^k)^*$ is regular.

Decision Procedure: Given formula ϕ

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

There are tools (e.g., MONA) that implement this procedure efficiently.

An 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.

Model in M2L-Str

```
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

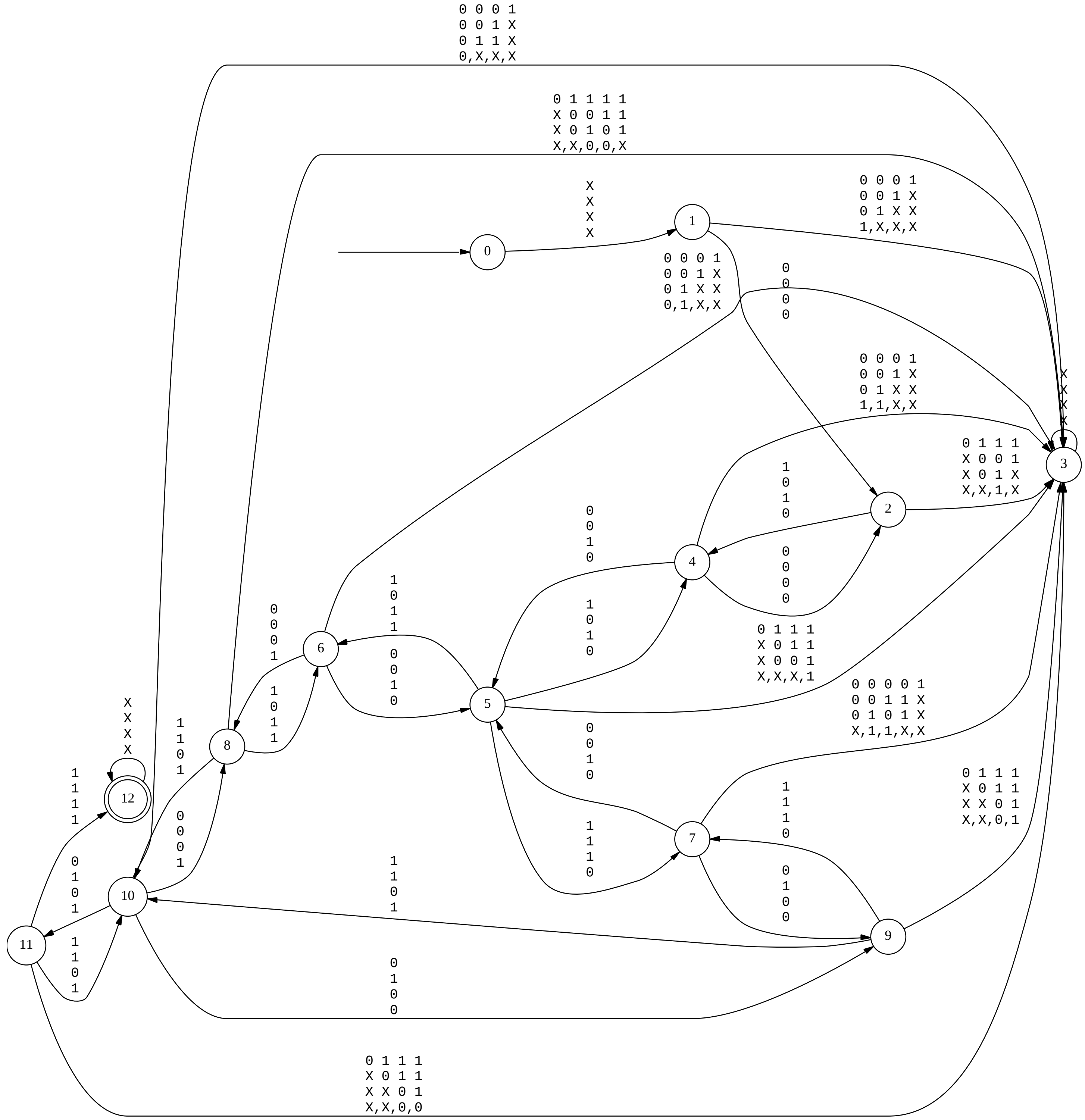
```
MONA v1.3 for WS1S/WS2S

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
```

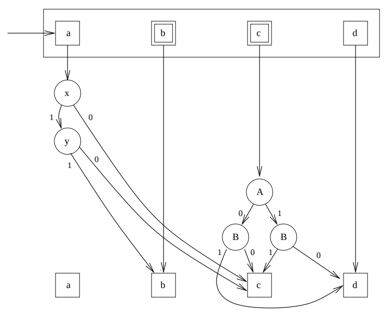

0 0 0 1
0 0 1 X
0 1 1 X
0,X,X,X
0 1 1 1 1
X 0 0 1 1
X 0 1 0 1
X,X,0,0,X
X
X
X
X
0 0 0 1
0 0 1 X
0 1 X X
1,X,X,X
0
1
0 0 0 1
0 0 1 X
0 1 X X
0,1,X,X
0
0
0
0
0 0 0 1
0 0 1 X
0 1 X X
1,1,X,X
X
X
X
X
0 1 1 1
X 0 0 1
X 0 1 X
X,X,1,X
3
1
0
1
0
2
0
0
1
0
4
0
0
0
0
1
0
1
1
1
0
1
0
0
0
0
1
6
0 1 1 1
X 0 1 1
X 0 0 1
X,X,X,1
0
0
1
0
5
1
0
1
1
0 0 0 0 1
0 0 1 1 X
0 1 0 1 X
X,1,1,X,X
X
X
X
X
1
1
0
1
8
0
0
1
0
1
1
1
1
12
1
1
1
0
0 1 1 1
X 0 1 1
X X 0 1
X,X,0,1
0
0
0
1
7
0
1
0
1
1
1
1
0
1
1
0
1
0
1
0
0
10
11
1
1
0
1
9
0
1
0
0
0 1 1 1
X 0 1 1
X X 0 1
X,X,0,0

Example summary

- What this example shows. . .
  - Simple declarative formalism for formalizing automata (state spaces)
  - Automata based procedures manipulate representation of entire state space
  - Same formalism for specifying system and properties, e.g., safety properties, planning goals, etc.
- What this example does not show
  - Logic excellent for reasoning about parameterized systems.
    E.g. systems parameterized over time or size (bit-width)
  - Pointed reasoning very useful: e.g., trivial to encode (pointed, interval, . . .) temporal logics, etc.
  - Logic is very expressive: properties can be expressed non-elementary more succint than as automata or temporal logic formulae.
  - This expressiveness is a mixed blessing (as is standard).

Complexity Problem I: Σ

- ϕ with n-free variables determines a language over $\Sigma = \mathbb{B}^n$
- Transition relation $\delta : \Sigma \times Q \times Q$ is exponential in n
- Solution (Klarlund et. al.): use OBDDs to represent δ.



This generalizes OBDDs to (regular) relations over strings

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

- Can lead to exponential compression. Empirically often the case!

Complexity Problem II: Q

- 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

- Basin and Klarlund have proven (Formal Methods in System Design, 1998) that for certain (useful) formula classes such blow-ups do not occur.

Some Statistics

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)

Alternatives for error dection/planning/. . .

- Good complexity guarantees are possible
- But sometimes blowups cannot be avoided. Nonelementary is nonelementary.
- Can we still analyze problems and benefit from the simplicity/conciseness/... of monadic logics?

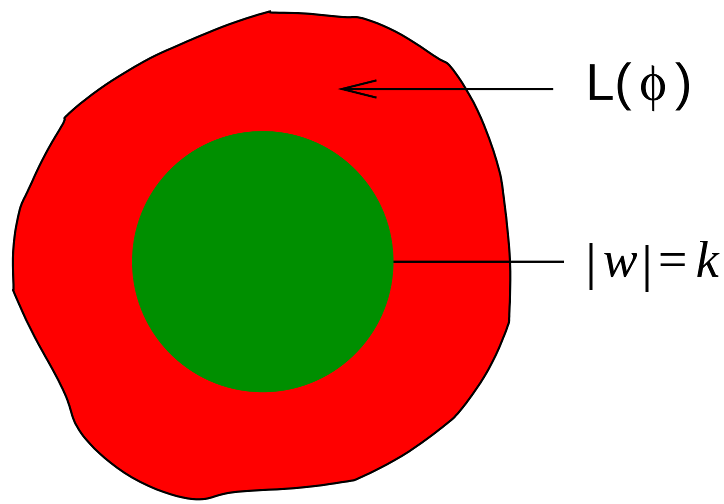
  Input: Problem (e.g., circuit + expected behavior)
  Output: a counter example (or plan, . . .), when it exists

- Questions:
  - Is this possible?
  - For which logics?
  - How would this work?
  - Is it feasible?

Bounded model construction

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

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



BMC **for** M2L-Str

- **Use (equivalent) minimal syntax:**

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

- **Idea:** $M \subseteq \{0, \ldots, k-1\}$ encoded by the Booleans b_0, ..., b_{k-1}: $i \in M$ iff b_i is true

- **Translation:** $\lceil . \rceil_k : \mathrm{MSO} \longrightarrow \mathrm{QBF}$

$$\begin{array}{lcl}
\lceil X \subseteq Y \rceil_k & = & \bigwedge_{0 \le i \le k-1} (x_i \rightarrow y_i) \\
\lceil succ(X, Y) \rceil_k & = & \mathsf{singleton}(x_0, \cdots, x_{k-1}) \wedge \mathsf{singleton}(y_0, \cdots, y_{k-1}) \wedge \\
 & & \bigvee_{0 \le i < k-1} (x_i \rightarrow y_{i+1}) \\
\lceil \phi_1 \wedge \phi_2 \rceil_k & = & \lceil \phi_1 \rceil_k \wedge \lceil \phi_2 \rceil_k \\
\lceil \neg\phi \rceil_k & = & \neg \lceil \phi \rceil_k \\
\lceil \exists X.\, \phi \rceil_k & = & \exists x_0 \cdots x_{k-1}.\, \lceil \phi \rceil_k
\end{array}$$

- **Complexity:** $\lceil . \rceil_k$ translation requires polynomial time

Results for BMC for $\mathrm{M2L\text{-}S{\scriptsize TR}}$

Theorem (Correctness)
For ϕ a MSO formula.

$$\models_{\text{M2L}} \phi \qquad \text{iff} \qquad \models_{\text{QBL}} \lceil \phi \rceil_k, \text{ for all } k \geq 0$$

Theorem (Complexity)
BMC for $\mathrm{M2L\text{-}S{\scriptsize TR}}$ is PSPACE-complete.

Proof:

– membership: $\lceil . \rceil_k$ can be tested in polynomial space.
– hardness:
  * QBL can be encoded in $\mathrm{M2L\text{-}S{\scriptsize TR}}$ and
  * QBL-satisfiability can be reduced to finding a model of length 1

Implementation of BMC for M2L-STR

$$\boxed{\text{M2L-STR}} \xRightarrow{\lceil\cdot\rceil_k} \boxed{\text{QBL}} \xRightarrow{\forall/\exists\text{-Elim}} \boxed{\text{BL}} \xRightarrow{Z\,chaff} \begin{cases} \text{yes}, & \sigma(y_i) = \ldots, \\ \text{no}, \end{cases}$$

Examples	MONA sec	BMC k	BMC 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)

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

BMC for WS1S

- Alternative semantics: standard weak second-order interpretation over $\mathcal{N}$.

  Theorem BMC for $\mathrm{WS1S}$ is nonelementary

  Proof:

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

  Corollary BMC is nonelementary for

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

Conclusion

- Monadic Logics are expressive and decidable modeling languages.

  Natural extension of (Q)BL with quantifiers. Good for practitioners.

- BDD-represented automata provide a powerful way of building and search a state-space.

- SAT procedures provide a fast alternative for finding small (counter)models.

- The right logic/tools makes it easy to switch between approaches/technologies when modeling and reasoning about enormous state spaces.

Some papers

Following available at `www.informatik.uni-freiburg.de/~basin/pubs/pubs.html`

- **Monadic Logics, Automata, and Circuits**

  D.B./Klarlund, Hardware Verification using Monadic Second-Order Logics, CAV 1995.

  D.B./Klarlund, Automata Based Symbolic Reasoning in Hardware Verification, Formal Methods in System Design, 1998.

  Ayari/D.B./Friedrich, Structural and Behavioral Modeling using Monadic Logics, ISMVL 1999.

- **Parameterized Families and Induction Principles**

  D.B./Friedrich, Combining WS1S and HOL, Frocos 2000.

- **Specification Languages and Complexity Issues**

  Ayari/D.B./Podelski, LISA: A Specification Language based on WS2S, CSL 1997.

- **Bounded Model Checking/Construction for Monadic Logics**

  Ayari/D.B., Bounded Model Construction for Monadic Second-Order Logics, CAV 2000.

- **Inductive Boolean Functions and Tree Automata**

  Ayari/D.B./Klaedtke, Decision Procedures for Inductive Boolean Functions based on Alternating Automata, CAV 2000.