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 $\land$ limited time $\implies$ high level presentation
The automata/logic connection

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

<table>
<thead>
<tr>
<th>Logic</th>
<th>Languages</th>
<th>Automata</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2L-STR, WS1S</td>
<td>regular languages</td>
<td>automata</td>
</tr>
<tr>
<td>S1S</td>
<td>$\omega$-regular languages</td>
<td>Büchi-automata</td>
</tr>
<tr>
<td>(W)S2S</td>
<td>regular tree languages</td>
<td>tree-automata</td>
</tr>
</tbody>
</table>

\[ X(0) \land \forall p < \$. \ X(p) \leftrightarrow Y(s(p)) \iff \left\{ \begin{array}{c|c} 1 & 1 \\ \hline 0 & 0 \end{array}, \begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & 1 & 0 \end{array}, \begin{array}{c|c|c} 1 & 1 & 0 \\ \hline 0 & 1 & 1 \end{array}, \ldots \right\} \]
Syntax for Monadic Logics on Strings

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

• Syntax:

$$
t ::= 0 \mid $ \mid p \mid s(t) \quad p \in V_1 \\
\phi ::= X(t) \mid t = t \mid t < t \mid X = Y \mid \neg \phi \mid \phi \lor \phi \mid \phi \land \phi \mid \exists p. \phi \mid \forall p. \phi \mid \exists X. \phi \mid \ldots \quad p \in V_1 \text{ and } X, Y \in V_2
$$

• An example

$$X(0) \land \forall p < $. (X(p) \leftrightarrow Y(s(p))))$$
Word-Model Semantics \( (M2L_{-\text{STR}}, \text{MSO}[S]) \)

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

\[
X \begin{array}{ccc}
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{array} \models X(0) \land \forall p < \$. (X(p) \leftrightarrow Y(s(p)))
\]

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

\[
p \begin{array}{ccc}
1 & 0 & 0 \\
0 & 0 & 1 \\
\end{array} \models \exists q. p < q \land q < r \land X(q)
\]

- 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 \( B^{\|fV(\phi)\|} \))

\[
\models \forall X. \exists Y. \forall p < \$. (X(p) \leftrightarrow Y(s(p)))
\]
Decidability

Büchi/Elgot/Trahtkenbrot For every M2L-STR formula $\phi$, with free variables $X_1, \ldots X_k$, the language $L(\phi) \subseteq \{0, 1\}^k$ 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, $w \not\models \phi$

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.
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
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
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 succinct than as automata or temporal logic formulae.
  – This expressiveness is a mixed blessing (as is standard).
Complexity Problem I: \( \Sigma \)

- \( \phi \) with \( n \)-free variables determines a language over \( \Sigma = B^n \)

- Transition relation \( \delta : \Sigma \times Q \times Q \) is exponential in \( n \)

- Solution (Klarlund et. al.): use OBDDs to represent \( \delta \).

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!

\[ \forall X. \exists Y. \phi \leadsto \neg \exists X. \neg \exists Y. \phi \]

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

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

**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 detection/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 $\phi$ and $k \in \mathbb{N}$

**Question:** Is there $w$ such that $|w| = k$ and $w$ satisfies $\phi$?
BMC for M2L-Str

• Use (equivalent) minimal syntax:

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

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

• Translation: \([.\]_k : \text{MSO} \rightarrow \text{QBF}\)

\[
\begin{align*}
[X \subseteq Y]_k &= \bigwedge_{0 \leq i \leq k-1} (x_i \rightarrow y_i) \\
[succ(X, Y)]_k &= \text{singleton}(x_0, \cdots, x_{k-1}) \land \text{singleton}(y_0, \cdots, y_{k-1}) \land \\
& \lor_{0 \leq i < k-1} (x_i \rightarrow y_{i+1}) \\
[\phi_1 \land \phi_2]_k &= [\phi_1]_k \land [\phi_2]_k \\
[\neg \phi]_k &= \neg [\phi]_k \\
[\exists X. \phi]_k &= \exists x_0 \cdots x_{k-1}. [\phi]_k
\end{align*}
\]

• Complexity: \([.\]_k translation requires polynomial time
Results for **BMC for M2L-STR**

**Theorem** (Correctness)
For $\phi$ a MSO formula.

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

**Theorem** (Complexity)
BMC for M2L-STR is PSPACE-complete.

Proof:

- membership: $\left\lceil . \right\rceil_k$ 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

\[
\begin{align*}
M2L-\text{Str} & \xrightarrow{\forall/\exists-\text{Elim}} \text{QBL} & \xrightarrow{\text{Zchaff}} \text{BL} & \{\text{yes, } \sigma(y_i) = \ldots, \text{no, }\}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Examples</th>
<th>\textbf{MONA} (k)</th>
<th>\textbf{BMC} (\text{sec})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ripple-carry adder</td>
<td>0.11</td>
<td>5</td>
</tr>
<tr>
<td>Mutual exclusion</td>
<td>0.58</td>
<td>9</td>
</tr>
<tr>
<td>FlipFlop</td>
<td>0.20</td>
<td>7</td>
</tr>
<tr>
<td>Barrel Shifter (8)</td>
<td>1.03</td>
<td>6</td>
</tr>
<tr>
<td>Barrel Shifter (32)</td>
<td>\text{abort}</td>
<td>6</td>
</tr>
<tr>
<td>Barrel Shifter (64)</td>
<td>\text{abort}</td>
<td>6</td>
</tr>
<tr>
<td>Counter (15)</td>
<td>242.17</td>
<td>8</td>
</tr>
<tr>
<td>Counter (32)</td>
<td>\text{abort}</td>
<td>8</td>
</tr>
</tbody>
</table>
## Byte code verification (buggy code)

<table>
<thead>
<tr>
<th>Examples</th>
<th>States #</th>
<th>Spin sec</th>
<th>Smv sec</th>
<th>BMC sec</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>StringBuffer.substringLjava.lang.String</td>
<td>$10^4$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Object.toStringLjava.lang.String</td>
<td>$10^5$</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>Integer.toStringLjava.lang.String</td>
<td>$10^{10}$</td>
<td>⊥</td>
<td>0</td>
<td>134</td>
<td>18</td>
</tr>
<tr>
<td>StringBuffer.appendLjava.lang.StringBuffer</td>
<td>$10^8$</td>
<td>⊥</td>
<td>1</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>Compiler_S_clinitLjava.lang.StringBuffer</td>
<td>$10^8$</td>
<td>⊥</td>
<td>4</td>
<td>76</td>
<td>22</td>
</tr>
<tr>
<td>FDBigInt.longValueJ</td>
<td>$10^{18}$</td>
<td>⊥</td>
<td>5</td>
<td>115</td>
<td>17</td>
</tr>
<tr>
<td>Object.waitLjava.lang.Int</td>
<td>$10^{10}$</td>
<td>⊥</td>
<td>26</td>
<td>21</td>
<td>11</td>
</tr>
<tr>
<td>StringBuffer.insertLjava.lang.StringBuffer</td>
<td>$10^{11}$</td>
<td>⊥</td>
<td>34</td>
<td>617</td>
<td>36</td>
</tr>
<tr>
<td>FDBigInt.multLjava.lang_FDBigInt</td>
<td>$10^{18}$</td>
<td>⊥</td>
<td>⊥</td>
<td>43</td>
<td>5</td>
</tr>
<tr>
<td>FDBigInt.multaddMeLjava.lang.Int</td>
<td>$10^{19}$</td>
<td>⊥</td>
<td>⊥</td>
<td>170</td>
<td>13</td>
</tr>
<tr>
<td>Character_S_clinitV</td>
<td>$10^{16}$</td>
<td>⊥</td>
<td>⊥</td>
<td>268</td>
<td>19</td>
</tr>
<tr>
<td>Long.parseLongLjava.lang.StringI_J</td>
<td>$10^{19}$</td>
<td>⊥</td>
<td>⊥</td>
<td>415</td>
<td>10</td>
</tr>
<tr>
<td>Short.decodeLjava.lang_StringLjava.lang.Short</td>
<td>$10^{12}$</td>
<td>⊥</td>
<td>⊥</td>
<td>727</td>
<td>30</td>
</tr>
<tr>
<td>FDBigInt.subLjava.lang_FDBigIntLjava.lang_FDBigInt</td>
<td>$10^{19}$</td>
<td>⊥</td>
<td>⊥</td>
<td>343</td>
<td>16</td>
</tr>
<tr>
<td>String.toLowerCaseLjava.util.LocaleLjava.lang.String</td>
<td>$10^{14}$</td>
<td>⊥</td>
<td>⊥</td>
<td>200</td>
<td>26</td>
</tr>
<tr>
<td>Integer.decodeLjava.lang_StringLjava.lang_Integer</td>
<td>$10^{12}$</td>
<td>⊥</td>
<td>⊥</td>
<td>303</td>
<td>303</td>
</tr>
</tbody>
</table>
BMC for WS1S

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

**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)
• **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**

- **Parameterized Families and Induction Principles**

- **Specification Languages and Complexity Issues**

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

- **Inductive Boolean Functions and Tree Automata**