DAG Automata and DAG Transducers

Dagstuhl Seminar 17142 – Formal Methods of Transformations

F. Drewes
(joint work with J. Blum, D. Chiang, D. Gildea, A. Lopez, G. Satta)

5 April 2017
Overview

1. Why DAGs?
2. DAG automata
3. What a difference a root makes
4. DAG transducers – a proposal
DAG Automata
“John desperately wants Mary to believe him, and she claims she does.”

[inspired by Abstract Meaning Representation]
Various earlier notions of DAG automata exist:

• Kamimura & Slutzki 1981
• Charatonik 1999 and Anantharaman et al. 2005
• Priese 2007
• Fujiyoshi 2010
• Quernheim & Knight 2012
• ... and a few others.

None of them fits very well, and most are too powerful. We want a simple type of DAG automaton.
• **Nodes have bounded rank**  
  [use symmetric version of first-child-next-sibling otherwise]

• **Incoming/outgoing edges are totally ordered**  
  [no disadvantage, but makes determinism reasonable]

• **Edges are unlabelled**  
  [can be encoded by auxiliary nodes]
Obvious idea: Computations assign states to edges.

A DAG Automaton consists of states, node labels, and rules of the form

\[
\begin{align*}
\text{incoming states} & \quad \downarrow \\
\{p_1, \ldots, p_m\} & \quad \text{node label} \\
\sigma & \quad \downarrow \\
\text{outgoing states} & \quad \{q_1, \ldots, q_n\}
\end{align*}
\]

A run is an assignment of states to edges. It is accepting if it locally coincides with a rule at each node:
• The model is symmetric with respect to “up” and “down”.
• No initial/final states; use rules $\varepsilon \xrightarrow{\sigma} q_1 \cdots q_n$ and $p_1 \cdots p_m \xrightarrow{\sigma} \varepsilon$.
• The automata as such do not ensure/require acyclicity, but we do.
• We consider only nonempty connected DAGs because a disjoint union of DAGs is accepted iff each component is.
• In particular, this makes it meaningful to talk about emptiness and finiteness of accepted languages.
• Swapping targets of edges with the same state preserves acceptance.
Example

\[ \text{paths}(L(A)) \cap a^*b^* = \{a^n b^n \mid n > 0\} \]
(likewise for \(a^n b^n c^n\) etc)
Example

\[ \text{paths}(L(A)) \cap a^*b^* = \{ a^n b^n \mid n > 0 \} \]
(likewise for \(a^n b^n c^n\) etc)
All (?) the previously proposed notions of DAG automata can restrict the number of roots.

<table>
<thead>
<tr>
<th></th>
<th>our model</th>
<th>restricted to single root</th>
</tr>
</thead>
<tbody>
<tr>
<td>emptiness</td>
<td>polynomial</td>
<td>non-elementary</td>
</tr>
<tr>
<td>finiteness</td>
<td>polynomial</td>
<td>?</td>
</tr>
<tr>
<td>path languages</td>
<td>regular</td>
<td>? (but not context-free)</td>
</tr>
<tr>
<td>unfolding</td>
<td>regular tree language</td>
<td>? (but non-regular)</td>
</tr>
<tr>
<td>membership</td>
<td>NP-complete</td>
<td>NP-complete</td>
</tr>
<tr>
<td>Parikh image</td>
<td>?semi-linear?</td>
<td></td>
</tr>
</tbody>
</table>
DAG Transducers – a Proposal
Desiderata

Possible uses:

- Turn syntax tree into semantic DAG (tree-to-DAG)
- Manipulate semantic DAG (DAG-to-DAG)
- Generate syntax tree from DAG (DAG-to-tree)

Ideally:

- “Natural” extension of the linear bottom-up tree transducer.
- Simple “local” rules to allow for training/learning.
- Extended rules should be covered.
- They should have some (un-)folding ability.
- Top-down should be the dual case.
Proposed Rule Type, Intuition

\[ p_1 \rightarrow \sigma \rightarrow \tau \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \]

... some DAG ...
Example: Folding

(+ many more rules that don’t fold)
Example: Folding
Concluding Remarks

• **Origin information** could be useful (comes naturally from training corpora).
• **Symbolic versions** can be of interest.
• I did not mention **weights**, but they are needed for NLP.
• Learning/training will be crucial.
• **It’s fun, please join!**
Thank you!