Monadic Second-Order Logic and Pomset Languages

Tobias Kappé
Cornell University

CPP 2021 — Lightning Talks
\[ P \models \phi \]

\[ \forall x. \lambda(x) = b \implies \exists y. x \leq y \land \lambda(y) = c \]

\[ \llbracket P \rrbracket = \left\{ \begin{array}{c}
    a \\
    \quad \quad \quad b \\
    \quad \quad \quad c, \ldots \\
\end{array} \right\} \]
\[
[P] = \left\{ \begin{array}{c}
    a \\
    \overset{b}{\longrightarrow}
    c, \ldots
  \end{array} \right\}
\]

\[
P \models \phi
\]

\[
\forall x. \lambda(x) = b \implies \exists y. x \leq y \land \lambda(y) = c
\]
\[ [P] = \begin{cases} \{ & \begin{array}{c} a \quad \rightarrow \quad b \\ b \quad \rightarrow \quad c, \ldots \end{array} \end{cases} \]

\[
\begin{aligned}
P \models \phi \\
\forall x. \lambda(x) = b \implies \\
\exists y. x \leq y \land \lambda(y) = c
\end{aligned}
\]
Theorem (Büchi, Elgot, Trakhtenbrot)

Let $\mathcal{L}$ be an MSO-definable language of words.
We can construct a finite automaton for $\mathcal{L}$.

Theorem (Kuske)

Let $\mathcal{L}$ be an MSO-definable language of pomsets.
There exists a finite bimonoid that recognises $\mathcal{L}$. 
Theorem (Büchi, Elgot, Trakhtenbrot)

Let $\mathcal{L}$ be an MSO-definable language of words. 
We can construct a finite automaton for $\mathcal{L}$.

Theorem (Kuske)

Let $\mathcal{L}$ be an MSO-definable language of pomsets. 
There exists* a finite bimonoid that recognises $\mathcal{L}$. 

*Existence is non-constructive.
Objectives

- Constructive translation of formulas to bimonoids.
- Proof that computed bimonoid accepts the same pomsets.
- Extraction of bimonoid code for use in model checking.
Objectives

- Constructive translation of formulas to bimonoids.
- Proof that computed bimonoid accepts the same pomsets.
- Extraction of bimonoid code for use in model checking.
Objectives

- Constructive translation of formulas to bimonoids.
- Proof that computed bimonoid accepts the same pomsets.
- Extraction of bimonoid code for use in model checking.
Record pomset (A: Type) := MkPomset {  
pomset_carrier: Type;  
pomset_order: pomset_carrier -> pomset_carrier -> Prop;  
pomset_labeling: pomset_carrier -> A;  

(* + pomset laws *)
}.
**Inductive** formula (A TP TS: Type) :=

<table>
<thead>
<tr>
<th>Before (x y: TP)</th>
<th>(* x ≤ y *)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member (x: TP) (X: TS)</td>
<td>(* x ∈ X *)</td>
</tr>
<tr>
<td>Label (x: TP) (a: A)</td>
<td>(* λ(x) = a *)</td>
</tr>
<tr>
<td>Disjunction (l r: formula)</td>
<td>(* l ∨ r *)</td>
</tr>
<tr>
<td>Negation (inner: formula)</td>
<td>(* ¬inner *)</td>
</tr>
<tr>
<td>ExPos (inner: formula A (TP + unit) TS)</td>
<td>(* ∃x. inner *)</td>
</tr>
<tr>
<td>ExSet (inner: formula A TP (TS + unit))</td>
<td>(* ∃X. inner *)</td>
</tr>
</tbody>
</table>
Fixpoint satisfies

{A TP TS: Type}
(f: formula A TP TS)
(u: pomset A)
: Prop
:=

(* snip *)
Fixpoint implementation
{A TP TS: Type}
(f: formula A TP TS)
: bimonoid
:=
(* work in progress *)
.
Lemma correctness

\{A: Type\}
\( (f: \text{formula } A \text{ Empty_set Empty_set}) \)
:=
\[\forall u: \text{pomset } A, \]
\( \text{bimonoid_eval (implementation } f) u = \text{true} \)
\( <-> \text{satisfies } f u \)

.  
Proof.

(* work in progress *)
Admitted.
Thoughts

- Constructive nature is nice for this proof.
- Still learning; currently grokking finite sets…
- Let's talk! tobias@kap.pe
Thoughts

- Constructive nature is nice for this proof.
- Still learning; currently grokking finite sets...
- Let's talk! tobias@kap.pe
Thoughts

- Constructive nature is nice for this proof.
- Still learning; currently grokking finite sets...
- Let's talk! tobias@kap.pe