Decidability and Complexity of Tree Share Formulas

Xuan Bach Le\textsuperscript{1}  Aquinas Hobor\textsuperscript{1}  Anthony W. Lin\textsuperscript{2}

\textsuperscript{1} National University of Singapore  \textsuperscript{2} University of Oxford

December 14, 2016
Decidability and Complexity of Tree Share Formulas

Introduction

\[ \text{tree}(x, \tau) \land \text{WRITE}(\tau) \]

\[ \text{tree}(x, \tau_1) \land \text{READ}(\tau_1) \quad \parallel \quad \text{tree}(x, \tau_2) \land \text{READ}(\tau_2) \]

\[ \text{tree}(x, \tau) \land \text{WRITE}(\tau) \]
Shares

Shares are embedded into separation logic to reason about resource accounting:

\[
\begin{align*}
\text{addr}^{\tau_1 \oplus \tau_2} \text{val} & \iff \text{addr}^{\tau_1} \text{val} \ast \text{addr}^{\tau_2} \text{val}
\end{align*}
\]
Shares

Shares are embedded into separation logic to reason about resource accounting:

\[ \text{addr} \overset{\tau_1 \oplus \tau_2}{\mapsto} \text{val} \iff \text{addr} \overset{\tau_1}{\mapsto} \text{val} \ast \text{addr} \overset{\tau_2}{\mapsto} \text{val} \]

- Allow resources to be split and shared in large scale:

\[
\text{tree}(\ell, \tau) \overset{\text{def}}{=} (\ell = \text{null} \land \text{emp}) \lor \exists \ell_l, \ell_r. (\ell \overset{\tau}{\mapsto} (\ell_l, \ell_r) \ast \text{tree}(\ell_l, \tau) \ast \text{tree}(\ell_r, \tau))
\]

\[
\text{tree}(\ell, \tau_1 \oplus \tau_2) \iff \text{tree}(\ell, \tau_1) \ast \text{tree}(\ell, \tau_2)
\]
Shares

Shares are embedded into separation logic to reason about resource accounting:

\[
\text{addr } \tau_1 \oplus \tau_2 \text{ val } \iff \text{addr } \rightarrow \text{val } * \text{addr } \rightarrow \text{val}
\]

- Allow resources to be split and shared in large scale:

\[
\text{tree}(\ell, \tau) \overset{\text{def}}{=} (\ell = \text{null } \land \text{emp}) \lor \exists \ell_l, \ell_r. (\ell \mapsto (\ell_l, \ell_r) * \text{tree}(\ell_l, \tau) * \text{tree}(\ell_r, \tau))
\]

\[
\text{tree}(\ell, \tau_1 \oplus \tau_2) \iff \text{tree}(\ell, \tau_1) * \text{tree}(\ell, \tau_2)
\]

- Share policies to reason about permissions for single writer and multiple readers:

\[
\text{WRITE}(\tau) \quad \text{WRITE-READ} \quad \text{READ}(\tau) \quad \text{SPLIT-READ}
\]

\[
\exists \tau_1, \tau_2. \quad \tau_1 \oplus \tau_2 = \tau \land \text{READ}(\tau_1) \land \text{READ}(\tau_2)
\]
Shares

Shares enable resource reasoning in concurrent programming
Shares

Shares enable resource reasoning in concurrent programming

- Rational numbers [Boyland (2003)]: disjointness problem makes tree split equivalence false:
  \( \neg \left( \text{tree}(\ell, \tau_1 \oplus \tau_2) \iff \text{tree}(\ell, \tau_1) \ast \text{tree}(\ell, \tau_2) \right) \)

Subsets of natural numbers [Parkinson (2005)]
- Finite sets: recursion depth is finite
- Infinite sets: intersections may not be in the model
Shares

Shares enable resource reasoning in concurrent programming

- Rational numbers [Boyland (2003)]: disjointness problem makes tree split equivalence false:
  \[ \neg (\text{tree}(\ell, \tau_1 \oplus \tau_2) \iff \text{tree}(\ell, \tau_1) \ast \text{tree}(\ell, \tau_2)) \]

- Subsets of natural numbers [Parkinson (2005)]
  - Finite sets: recursion depth is finite
  - Infinite sets: intersections may not be in the model
A tree share $\tau \in T$ is a boolean binary tree equipped with the reduction rules $R_1$ and $R_2$ (their inverses are $E_1, E_2$ resp.):

$$\tau \overset{\text{def}}{=} \circ | \bullet | \tau \quad \tau \overset{R_1}{=} \bullet \quad \tau \overset{R_2}{=} \circ$$

The tree domain $T$ contains canonical trees which are irreducible with respect to the reduction rules.
A tree share $\tau \in T$ is a boolean binary tree equipped with the reduction rules $R_1$ and $R_2$ (their inverses are $E_1$, $E_2$ resp.):

$$\tau \overset{\text{def}}{=} \circ | \bullet | \tau \quad R_1: \bullet \quad \rightarrow \bullet \quad R_2: \circ \quad \rightarrow \circ$$

The tree domain $T$ contains canonical trees which are irreducible with respect to the reduction rules.
A tree share $\tau \in T$ is a boolean binary tree equipped with the reduction rules $R_1$ and $R_2$ (their inverses are $E_1, E_2$ resp.):

$$
\tau \overset{\text{def}}{=} \circ \mid \bullet \mid \begin{array}{c} \tau \\
\tau \end{array} \\
R_1 : \begin{array}{c} \bullet \\
\bullet \\
\tau \end{array} \rightarrow \bullet \\
R_2 : \begin{array}{c} \circ \\
\circ \\
\circ \\
\circ \\
\end{array} \rightarrow \circ
$$

The tree domain $T$ contains canonical trees which are irreducible with respect to the reduction rules.

- $\circ$ is the empty tree, and $\bullet$ the full tree.
A tree share $\tau \in \mathbb{T}$ is a boolean binary tree equipped with the reduction rules $R_1$ and $R_2$ (their inverses are $E_1, E_2$ resp.):

$$
\begin{align*}
\tau & \overset{\text{def}}{=} \circ | \bullet | \tau - \tau \\
R_1 : \overset{\text{def}}{=} & \bullet \leftrightarrow \bullet \\
R_2 : \overset{\text{def}}{=} & \circ \leftrightarrow \circ
\end{align*}
$$

The tree domain $\mathbb{T}$ contains canonical trees which are irreducible with respect to the reduction rules.

- $\circ$ is the empty tree, and $\bullet$ the full tree.
- $\text{READ}(\tau) \overset{\text{def}}{=} \tau \neq \circ$  
- $\text{WRITE}(\tau) \overset{\text{def}}{=} \tau = \bullet$
Tree Share Operators

- The complement □:

\[
\neg E_i \lor R_i \land E_i
\]

\[
\neg E_i \land R_i \lor E_i
\]
Tree Share Operators

- The complement □:

```
  □
 / \   □
1   2  3
  □     □
  1     2
```

```
  □
 / \   □
1   2  3
  □  □
  1  2  3
```

The Boolean function union ∨ and intersection ∧ operator:
Tree Share Operators

- The complement $\neg$:

- The Boolean function union $\lor$ and intersection $\land$ operator:
Tree Share Operators

- **The complement** \( \square \): 

- **The Boolean function** union \( \sqcup \) and intersection \( \sqcap \) operator:
Tree Share Operators

- **The complement □:**

- **The Boolean function union □ and intersection □ operator:**
Properties of $\cup$, $\cap$ and $\bar{\cdot}$

$M = (\cup, \cap, \bar{\cdot}, \bullet, \circ)$ forms a Boolean Algebra [Dockins et al. (2009)]:

$B1a. \ (\tau_1 \cap \tau_2) \cap \tau_3 = \tau_1 \cap (\tau_2 \cap \tau_3)$

$B2a. \ \tau_1 \cap \tau_2 = \tau_2 \cap \tau_1$

$B3a. \ \tau_1 \cap (\tau_2 \cup \tau_3) = (\tau_1 \cap \tau_2) \cup (\tau_1 \cap \tau_3)$

$B4a. \ \tau_1 \cap (\tau_1 \cup \tau_2) = \tau_1$

$B5a. \ \tau \cap \bullet = \tau$

$B6a. \ \tau \cap \bar{\tau} = \circ$

$B1b. \ (\tau_1 \cup \tau_2) \cup \tau_3 = \tau_1 \cup (\tau_2 \cup \tau_3)$

$B2b. \ \tau_1 \cup \tau_2 = \tau_2 \cup \tau_1$

$B3b. \ \tau_1 \cup (\tau_2 \cap \tau_3) = (\tau_1 \cup \tau_2) \cap (\tau_1 \cup \tau_3)$

$B4b. \ \tau_1 \cup (\tau_1 \cap \tau_2) = \tau_1$

$B5b. \ \tau \cup \circ = \tau$

$B6b. \ \tau \cup \bar{\tau} = \bullet$

(associativity) (commutativity) (distributivity) (absorption) (identity) (complement)
Tree Share Operators (cont.)

The partial \textit{join} function $\oplus$: 
Tree Share Operators (cont.)

The partial join function $\oplus$:

$$\tau_1 \oplus \tau_2 = \tau_3 \quad \text{def} \quad \tau_1 \cup \tau_2 = \tau_3 \land \tau_1 \cap \tau_2 = \circ$$
The partial join function $\oplus$:

$$\tau_1 \oplus \tau_2 = \tau_3 \overset{\text{def}}{=} \tau_1 \lor \tau_2 = \tau_3 \land \tau_1 \land \tau_2 = \circ$$
Properties of $\oplus$

$\mathcal{O} = (\mathbb{T}, \oplus)$ for fractional permission in Separation Logic [Dockins et al. (2009)]:

$J_1. \quad \tau_1 \oplus \tau_2 = \tau_3 \Rightarrow \tau_1 \oplus \tau_2 = \tau_3' \Rightarrow \tau_3 = \tau_3'$  
(functionality)

$J_2. \quad \tau_1 \oplus \tau_2 = \tau_2 \oplus \tau_1$  
(commutativity)

$J_3. \quad \tau_1 \oplus (\tau_2 \oplus \tau_3) = (\tau_1 \oplus \tau_2) \oplus \tau_3$  
(associativity)

$J_4. \quad \tau_1 \oplus \tau_2 = \tau_3 \Rightarrow \tau_1' \oplus \tau_2 = \tau_3 \Rightarrow \tau_1 = \tau_1'$  
(cancellation)

$J_5. \quad \exists u. \forall \tau. \tau \oplus u = \tau$  
(unit)

$J_6. \quad \tau_1 \oplus \tau_2 = \tau_2 \Rightarrow \tau_1 = \tau_2$  
(disjointness)

$J_7. \quad a \oplus b = z \land c \oplus d = z \Rightarrow \exists ac, ad, bc, bd.$  
(cross split)

$a c \oplus a d = a \land b c \oplus b d = b \land a c \oplus b c = c \land a d \oplus b d = d$  
(infinite split)

$J_8. \quad \tau \neq \circ \Rightarrow \exists \tau_1, \tau_2. \tau_1 \neq \circ \land \tau_2 \neq \circ \land \tau_1 \oplus \tau_2 = \tau$
Tree Share Operators (cont.)

The injection bowtie function \( \bowtie \) replaces \( \bullet \) with tree:
Tree Share Operators (cont.)

The injection bowtie function \( \bowtie \) replaces \( \bullet \) with tree:

\[
\begin{align*}
\text{● ○ ○ ●} & \bowtie \text{○ ● ○ ●} = \text{○ ● ○ ●} \\
\text{○ ● ○ ●} & \bowtie \text{○ ○ ● ○} = \text{○ ○ ● ○}
\end{align*}
\]
Tree Share Operators (cont.)

The injection bowtie function △ replaces ● with tree:

```
โธ่ กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิน โดย กศิ

Allow resources to be split uniformly:

\[ \tau_1 \cdot \text{tree}(\ell, \tau_2) \overset{\text{def}}{=} \text{tree}(\ell, \tau_2 \triangleleft \tau_1) \]

\[ (\tau_1 \oplus \tau_2) \cdot \text{tree}(\ell, \tau) \iff \tau_1 \cdot \text{tree}(\ell, \tau) \star \tau_2 \cdot \text{tree}(\ell, \tau) \]

\[ \tau_1 \cdot \text{tree}(\ell, \tau_2 \triangleright \tau_3) \iff (\tau_3 \triangleright \tau_1) \cdot \text{tree}(\ell, \tau_2) \]
Tree Share Operators (cont.)

The injection bowtie function $\bowtie$ replaces $\bullet$ with tree:

$\text{tree}(\ell, \tau_1 \bowtie \tau_2) \begin{equation} \overset{\text{def}}{=} \text{tree}(\ell, \tau_2 \bowtie \tau_1) \end{equation}$

$(\tau_1 \oplus \tau_2) \cdot \text{tree}(\ell, \tau) \leftrightarrow \tau_1 \cdot \text{tree}(\ell, \tau) \bowtie \tau_2 \cdot \text{tree}(\ell, \tau)$

$
\tau_1 \cdot \text{tree}(\ell, \tau_2 \bowtie \tau_3) \leftrightarrow (\tau_3 \bowtie \tau_1) \cdot \text{tree}(\ell, \tau_2)
$

$\bowtie$ can be hard to think about. Is this equation satisfiable?
Properties of $\bowtie$

$S = (\bowtie, \bullet)$ forms an Monoid with additional properties [Dockins et al. (2009)]:

M1. $(\tau_1 \bowtie \tau_2) \bowtie \tau_3 = \tau_1 \bowtie (\tau_2 \bowtie \tau_3)$ \hspace{1cm} \text{(associativity)}

M2. $\tau \bowtie \bullet = \bullet \bowtie \tau = \tau$ \hspace{1cm} \text{(identity)}

M3. $\tau \bowtie \circ = \circ \bowtie \tau = \circ$ \hspace{1cm} \text{(collapse point)}

M4. $\tau_1 \bowtie (\tau_2 \diamond \tau_3) = (\tau_1 \bowtie \tau_2) \bowtie (\tau_1 \bowtie \tau_3), \; \diamond \in \{\cap, \cup, \oplus\}$ \hspace{1cm} \text{(distributivity)}

M5. $\tau \bowtie \tau_1 = \tau \bowtie \tau_2 \Rightarrow \tau \neq \circ \Rightarrow \tau_1 = \tau_2$ \hspace{1cm} \text{(left cancellation)}

M6. $\tau_1 \bowtie \tau = \tau_2 \bowtie \tau \Rightarrow \tau \neq \circ \Rightarrow \tau_1 = \tau_2$ \hspace{1cm} \text{(right cancellation)}
Decidability and Complexity of Tree Share Formulas

Introduction

\[
\text{tree}(x, \tau) \\
\text{tree}(x, \tau) \cdot (\dot\boxplus \circ \dot\circ \cdot \cdot) \\
\text{tree}(x, \tau) \cdot \cdot \circ \cdot \circ \cdot \\
\text{tree}(x, \tau) \\
\text{tree}(x, \tau) \cdot (\dot\boxplus \circ \dot\circ \cdot \cdot) \\
\]
Outline

1 Introduction

2 Decidability and Complexity results
   - Model for Countable Atomless Boolean Algebra
   - From $\bowtie$ to string concatenation
   - Tree Automatic Structures

3 Conclusion
Tree Shares as Countable Atomless Boolean Algebra

\[ \mathcal{M} = (\sqcup, \sqcap, \overline{\_}, \bullet, \circ) \] is Countable Boolean Algebra because the domain \( T \) is countable.
Tree Shares as Countable Atomless Boolean Algebra

- $\mathcal{M} = (\sqcup, \sqcap, \equiv, \bullet, \circ)$ is Countable Boolean Algebra because the domain $\mathbb{T}$ is countable.
- Atomless properties of $\mathcal{M}$:
  - Let $\tau_1 \neq \tau_2$, we denote $\tau_1 \sqsubset \tau_2$ iff $\tau_1 \cup \tau_2 = \tau_2$. 

Tree Shares as Countable Atomless Boolean Algebra

- $\mathcal{M} = (\cup, \cap, \varnothing, \bullet, \circ)$ is Countable Boolean Algebra because the domain $\mathbb{T}$ is countable.

- Atomless properties of $\mathcal{M}$:
  - Let $\tau_1 \neq \tau_2$, we denote $\tau_1 \sqsubseteq \tau_2$ iff $\tau_1 \cup \tau_2 = \tau_2$.
  - $\mathcal{M}$ is atomless if for $\tau_1 \sqsubseteq \tau_3$, there exists $\tau_2$ such that $\tau_1 \sqsubseteq \tau_2 \sqsubseteq \tau_3$. 
Tree Shares as Countable Atomless Boolean Algebra

- $\mathcal{M} = (\cup, \cap, \Box, \bullet, \circ)$ is Countable Boolean Algebra because the domain $\mathbb{T}$ is countable.
- Atomless properties of $\mathcal{M}$:
  - Let $\tau_1 \neq \tau_2$, we denote $\tau_1 \sqsubset \tau_2$ iff $\tau_1 \cup \tau_2 = \tau_2$.
  - $\mathcal{M}$ is atomless if for $\tau_1 \sqsubset \tau_3$, there exists $\tau_2$ such that $\tau_1 \sqsubset \tau_2 \sqsubset \tau_3$.
  - Let $\tau_1 = \circ \bullet \circ$ and $\tau_3 = \bullet \circ$ then $\tau_1 \sqsubset \tau_3$. We extend $\tau_3$ to the shape of $\tau_1$: $\tau_3 \mapsto E_i$.
Tree Shares as Countable Atomless Boolean Algebra

- $\mathcal{M} = (\sqcup, \sqcap, \boxdot, \bullet, \circ)$ is Countable Boolean Algebra because the domain $\mathbb{T}$ is countable.
- Atomless properties of $\mathcal{M}$:
  - Let $\tau_1 \neq \tau_2$, we denote $\tau_1 \sqsubset \tau_2$ iff $\tau_1 \sqcup \tau_2 = \tau_2$.
  - $\mathcal{M}$ is atomless if for $\tau_1 \sqsubset \tau_3$, there exists $\tau_2$ such that $\tau_1 \sqsubset \tau_2 \sqsubset \tau_3$.
  - Let $\tau_1 = \overline{\circ} \overline{\bullet} \circ$ and $\tau_3 = \overline{\bullet} \circ$ then $\tau_1 \sqsubset \tau_3$. We extend $\tau_3$ to the shape of $\tau_1$:

$$
\tau_3 \overset{E_i}{\mapsto}
\begin{array}{c}
\bullet \\
\bullet
\end{array} \overline{\circ}
$$

then replace one of the $\bullet$ leaves of $\tau_3$ that is not in $\tau_1$ with $\overline{\circ}$:

$$
\begin{array}{c}
\bullet \\
\bullet
\end{array} \circ
$$
Decidability of $\mathcal{M}$

The first-order theory of $\mathcal{M}$ is decidable. The lower bound for its complexity is $\bigcup_{c<\omega} \text{STA}(\star, 2^{cn}, n)$ [Kozen (1980)].
Decidability and Complexity of Tree Share Formulas

Decidability results

From $\bowtie$ to string concatenation

## Decidability of $\bowtie$

<table>
<thead>
<tr>
<th>Decidability of $S = (T, \bowtie)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let $S = (T, \bowtie)$ then:</td>
</tr>
<tr>
<td>- The existential theory of $S$ is decidable in PSPACE.</td>
</tr>
<tr>
<td>- The existential theory of $S$ is NP-hard.</td>
</tr>
<tr>
<td>- The general first-order theory over $S$ is undecidable.</td>
</tr>
</tbody>
</table>
Decidability of $\bowtie$

**Decidability of $S = (\mathbb{T}, \bowtie)$**

Let $S = (\mathbb{T}, \bowtie)$ then:

- The existential theory of $S$ is decidable in PSPACE.
- The existential theory of $S$ is NP-hard.
- The general first-order theory over $S$ is undecidable.

**Decidability of $S^+ = (\mathbb{T}\setminus\{\circ\}, \bowtie)$**

Let $S^+ = (\mathbb{T}\setminus\{\circ\}, \bowtie)$ then:

- The existential theory of $S^+$ is decidable in PSPACE.
- The existential theory of $S^+$ is NP-hard.
- The general first-order theory over $S^+$ is undecidable.
Isomorphism between $\bowtie$ and $\cdot$.

To prove these results on $S^+ = (T\setminus\{\circ\}, \bowtie)$, we will construct an isomorphism between $S^+$ equations and word equations.
Isomorphism between $\bowtie$ and $\cdot$

To prove these results on $S^+ = (\mathbb{Z}\backslash\{\circ\}, \bowtie)$, we will construct an isomorphism between $S^+$ equations and word equations.

In particular, we will transform $\bowtie$ into string concatenation. The trick is that we must find an “alphabet” for $S^+$ equations.
Review of Word Equations

- Let $\mathcal{A} = \{a, b, \ldots\}$ be the finite set of alphabet and $\mathcal{V} = \{v_1, v_2, \ldots\}$ the set of variables.
Review of Word Equations

- Let $\mathcal{A} = \{a, b, \ldots\}$ be the finite set of alphabet and $\mathcal{V} = \{v_1, v_2, \ldots\}$ the set of variables.
- A word $w$ is a string in $(\mathcal{A} \cup \mathcal{V})^*$. A word equation $E$ is a pair of words $w_1 = w_2$. 
Review of Word Equations

- Let $A = \{a, b, \ldots\}$ be the finite set of alphabet and $V = \{v_1, v_2, \ldots\}$ the set of variables.
- A word $w$ is a string in $(A \cup V)^*$. A word equation $E$ is a pair of words $w_1 = w_2$.
- $E$ has a solution if there exists a homomorphism $f : A \cup V \mapsto A^*$ that maps each letter in $A$ to itself.
Review of Word Equations

- Let \( \mathcal{A} = \{a, b, \ldots\} \) be the finite set of alphabet and \( \mathcal{V} = \{v_1, v_2, \ldots\} \) the set of variables.
- A word \( w \) is a string in \((\mathcal{A} \cup \mathcal{V})^*\). A word equation \( E \) is a pair of words \( w_1 = w_2 \).
- \( E \) has a solution if there exists a homomorphism \( f : \mathcal{A} \cup \mathcal{V} \rightarrow \mathcal{A}^* \) that maps each letter in \( \mathcal{A} \) to itself.
- For example, the equation \( v_1 v_2 ab = b a v_2 v_1 \) has a solution:

\[
    f(v_1) = b, f(v_2) = a
\]
Review of Word Equations

- Let $\mathcal{A} = \{a, b, \ldots\}$ be the finite set of alphabet and $\mathcal{V} = \{v_1, v_2, \ldots\}$ the set of variables.
- A word $w$ is a string in $(\mathcal{A} \cup \mathcal{V})^*$. A word equation $E$ is a pair of words $w_1 = w_2$.
- $E$ has a solution if there exists a homomorphism $f : \mathcal{A} \cup \mathcal{V} \mapsto \mathcal{A}^*$ that maps each letter in $\mathcal{A}$ to itself.
- For example, the equation $v_1 v_2 ab = bav_2 v_1$ has a solution:
  \[ f(v_1) = b, f(v_2) = a \]
- The satisfiability problem of word equation: checking whether a word equation $E$ has a solution.
Word Equation Results

Decidability and Complexity of Word Equation

- The satisfiability problem of word equation is decidable. The lower bound is NP-complete while the upper bound is PSPACE [Plandowski (1999)].
Word Equation Results

Decidability and Complexity of Word Equation

- The satisfiability problem of word equation is decidable. The lower bound is NP-complete while the upper bound is PSPACE [Plandowski (1999)].

- The satisfiability of a system of word equations with regular constraints $v_i \in \text{REG}_i$ can be reduced to the satisfiability of a single word equation [Schulz (1990)].
Word Equation Results

Decidability and Complexity of Word Equation

- The satisfiability problem of word equation is decidable. The lower bound is NP-complete while the upper bound is PSPACE [Plandowski (1999)].

- The satisfiability of a system of word equations with regular constraints $v_i \in \text{REG}_i$ can be reduced to the satisfiability of a single word equation [Schulz (1990)].

- The existential theory of string concatenation is decidable with lower bound NP-complete and upper bound PSPACE. The first-order theory of string concatenation is undecidable (folklore).
Tree factorization

Prime trees

A tree $\tau \in T \setminus \{\bullet, \circ\}$ is prime iff $\tau = \tau_1 \bowtie \tau_2$ then either $\tau_1 = \bullet$ or $\tau_2 = \bullet$. 
Tree factorization

Prime trees

A tree \( \tau \in \mathbb{T}\setminus\{\bullet, \circ\} \) is prime iff \( \tau = \tau_1 \bowtie \tau_2 \) then either \( \tau_1 = \bullet \) or \( \tau_2 = \bullet \).

A tree share \( \tau \) can be factorized into prime trees using \( \bowtie \):
A tree $\tau \in T\setminus\{\bullet, \circ\}$ is prime iff $\tau = \tau_1 \bowtie \tau_2$ then either $\tau_1 = \bullet$ or $\tau_2 = \bullet$.

A tree share $\tau$ can be factorized into prime trees using $\bowtie$:
Tree factorization

Prime trees

A tree \( \tau \in T \setminus \{\bullet, \circ\} \) is prime iff \( \tau = \tau_1 \bowtie \tau_2 \) then either \( \tau_1 = \bullet \) or \( \tau_2 = \bullet \).

A tree share \( \tau \) can be factorized into prime trees using \( \bowtie \):
Tree factorization

Prime trees

A tree $\tau \in T \setminus \{\bullet, \circ\}$ is prime iff $\tau = \tau_1 \bowtie \tau_2$ then either $\tau_1 = \bullet$ or $\tau_2 = \bullet$.

A tree share $\tau$ can be factorized into prime trees using $\bowtie$:
Unique factorization

Let $\tau \in \mathbb{T}\setminus\{\circ, \bullet\}$ then there exists a unique sequence of prime trees $\tau_1, \ldots, \tau_n$ such that:

$$
\tau = \tau_1 \Join \ldots \Join \tau_n
$$

Furthermore, the factorization problem is in PTIME.

Proof sketch: By induction on the structure of the tree.
Let $\mathbb{T}_p \subset \mathbb{T}$ be the set of prime trees then $\mathbb{T}_p$ is countably infinite.
Infinite alphabet

- Let $T_p \subset T$ be the set of prime trees then $T_p$ is countably infinite.
- $T_p$ is our *alphabet* for the word equation but we need to reduce it to finite alphabet.
Decidability and Complexity of Tree Share Formulas

Decidability and Complexity results

From \( \bowtie \) to string concatenation

Infinite alphabet (cont.)

From infinity to finite

Let \( \Sigma \) be the set of word equations and inequations over infinite alphabet \( A \) then \( \Sigma \) has a solution iff it has a solution over some finite alphabet \( B \subset A \) such that:

1. \( \mathcal{A}(\Sigma) \subset B \)
2. \( |B| = |\mathcal{A}(\Sigma)| + n \) where \( n \) is the number of inequations in \( \Sigma \).

The choice of the extra letters in \( B \) is not important.
Example
Example

\[ V_1 \bowtie V_2 \bowtie \begin{array}{c} \circ \\ \bullet \end{array} \bowtie \circ = \begin{array}{c} \circ \\ \bullet \end{array} \bowtie V_2 \bowtie V_1 \]

\[ V_1 \bowtie V_2 \bowtie \begin{array}{c} \circ \\ \bullet \end{array} \bowtie \begin{array}{c} \circ \\ \bullet \end{array} \bowtie \begin{array}{c} \circ \\ \bullet \end{array} = \begin{array}{c} \circ \\ \bullet \end{array} \bowtie \begin{array}{c} \circ \\ \bullet \end{array} \bowtie \begin{array}{c} \circ \\ \bullet \end{array} \bowtie V_2 \bowtie V_1 \]
Example

\[ v_1 \bowtie v_2 \bowtie \begin{array}{c} \circ \bullet \circ \\ \end{array} = \begin{array}{c} \circ \bullet \circ \\ \end{array} \bowtie v_2 \bowtie v_1 \]

\[ v_1 \bowtie v_2 \bowtie \begin{array}{c} \circ \bullet \circ \circ \bullet \\ \end{array} = \begin{array}{c} \circ \bullet \circ \bullet \circ \\ \end{array} \bowtie v_2 \bowtie v_1 \]

\[ v_1 v_2 ab = bav_2 v_1 \]
Example

\[ v_1 \bowtie v_2 \bowtie \begin{array}{c} \circ \\ \bullet \\ \circ \end{array} = \begin{array}{c} \circ \\ \bullet \\ \circ \end{array} \bowtie v_2 \bowtie v_1 \]

\[ v_1 \bowtie v_2 \bowtie \begin{array}{c} \bullet \\ \circ \\ \circ \\ \bullet \end{array} = \begin{array}{c} \bullet \\ \circ \\ \bullet \end{array} \bowtie v_2 \bowtie v_1 \]

\[ v_1 v_2 ab = bav_2 v_1 \quad \text{solution: } v_1 = b, \ v_2 = a \]
Decidability and Complexity of Tree Share Formulas

Decidability and Complexity results

From \( \otimes \) to string concatenation

Example

\[
V_1 \otimes V_2 \otimes \begin{array}{c} \circ \\ \bullet \\ \circ \end{array} = \begin{array}{c} \circ \\ \bullet \\ \circ \end{array} \otimes V_2 \otimes V_1
\]

\[
V_1 \otimes V_2 \otimes \begin{array}{c} \circ \\ \bullet \\ \circ \\ \bullet \end{array} = \begin{array}{c} \circ \\ \bullet \\ \circ \\ \bullet \end{array} \otimes V_2 \otimes V_1
\]

\[
v_1 v_2 ab = bav_2 v_1 \]  

solution: \( v_1 = b, v_2 = a \)

\[
\begin{array}{c} \circ \\ \bullet \\ \circ \\ \bullet \\ \circ \\ \bullet \\ \circ \\ \bullet \end{array} \otimes \begin{array}{c} \circ \\ \bullet \\ \circ \\ \bullet \end{array} = \begin{array}{c} \circ \\ \bullet \\ \circ \\ \bullet \\ \circ \\ \bullet \\ \circ \\ \bullet \end{array} \otimes \begin{array}{c} \circ \\ \bullet \\ \circ \\ \bullet \end{array} \otimes \begin{array}{c} \circ \\ \bullet \\ \circ \\ \bullet \end{array}
\]
Find a decidable fragment for \( \bowtie \)

Since the first-order theory of \( S = (T, \bowtie) \) is undecidable, we want to find a decidable fragment of \( \bowtie \) together with \( (\sqcup, \sqcap, \square) \).
Connection to Tree Automatic Structures

Let $\bowtie_\tau$ be the unary function over trees such that

$$
\bowtie_\tau(\tau') = \tau' \bowtie \tau
$$
Connection to Tree Automatic Structures

- Let $\bowtie_\tau$ be the unary function over trees such that

$$\bowtie_\tau(\tau') = \tau' \bowtie \tau$$

- Example:

$\bowtie_\tau\left(\begin{array}{c}
\circ \\
\bullet \\
\bullet \\
\bullet \\
\circ
\end{array}\right) = \begin{array}{c}
\bullet \\
\bullet \\
\circ \\
\bullet
\end{array} \bowtie \begin{array}{c}
\circ \\
\bullet \\
\bullet \\
\circ
\end{array} = \begin{array}{c}
\circ \\
\bullet \\
\circ \\
\bullet
\end{array}$
Let $\bowtie_\tau$ be the unary function over trees such that

$$\bowtie_\tau(\tau') = \tau' \bowtie \tau$$

Example:

Let $\mathcal{T} = (T, \cup, \cap, \overline{\phantom{T}}, \bowtie_\tau)$ then $\mathcal{T}$ is tree automatic, i.e., its domain and relations are recognized by tree automata. Consequently, the first-order theory of $\mathcal{T}$ is decidable [Blumensath (1999); Blumensath and Gradel (2004)].
Decidability and Complexity of Tree Share Formulas

Outline

1. Introduction

2. Decidability and Complexity results
   - Model for Countable Atomless Boolean Algebra
   - From $\bowtie$ to string concatenation
   - Tree Automatic Structures

3. Conclusion
Contributions

- We show that $\mathcal{M} = (\cup, \cap, \square, \cdot, \circ)$ forms a Countably Atomless Boolean Algebra.
- We reduce $\bowtie$ to string concatenation.
- We show $\mathcal{T} = (\mathbb{T}, \cup, \cap, \square, \bowtie, \triangleright)$ is tree-automatic.
Future Work

- Complexity of \((\mathbb{T}, \cap, \cup)\) (\(\exists\)-theory and first-order theory).
- Decidability of \((\mathbb{T}, \cap, \cup, \bowtie)\) (\(\exists\)-theory).
- Complexity of \(\mathcal{T} = (\mathbb{T}, \cup, \cap, \bar{owtie}, \bowtie_{\tau})\) (\(\exists\)-theory and first-order theory).
- Extension of word equation to tree equation.

Thank you! ☺️
Proof sketch:

- Let $f$ be a solution of $\Sigma$. For each inequation $w_1 \neq w_2$ to hold, it suffices to have a single position where they differs.
Proof sketch:

- Let $f$ be a solution of $\Sigma$. For each inequation $w_1 \neq w_2$ to hold, it suffices to have a single position where they differ.
- Therefore, there is at most one letter $a_i \notin A(\Sigma)$ in each inequation that we need to preserve.
Proof sketch:

- Let $f$ be a solution of $\Sigma$. For each inequation $w_1 \neq w_2$ to hold, it suffices to have a single position where they differ.
- Therefore, there is at most one letter $a_i \notin \mathcal{A}(\Sigma)$ in each inequation that we need to preserve.
- For other letters $b_i \notin \mathcal{A}(\Sigma)$, we simply replace them with a letter in $\mathcal{A}(\Sigma)$. 
Proof sketch:

- Let \( f \) be a solution of \( \Sigma \). For each inequation \( w_1 \neq w_2 \) to hold, it suffices to have a single position where they differs.

- Therefore, there is at most one letter \( a_i \notin \mathcal{A}(\Sigma) \) in each inequation that we need to preserve.

- For other letters \( b_i \notin \mathcal{A}(\Sigma) \), we simply replace them with a letter in \( \mathcal{A}(\Sigma) \).

- As a result, the new solution satisfies the alphabet constraint.


