Unification in Description Logics

Part III: Unification in the DL $\mathcal{FL}_0$

Oliver Fernández Gil

Chair of Automata Theory

TECHNISCHE UNIVERSITÄT DRESDEN

ESSLLI'19

Riga, August 2019

The DL $\mathcal{FL}_0$

- Fragment of $\mathcal{ALC}$:

$$C ::= \top \mid A \mid C \sqcap C \mid \forall r.C$$

The DL $\mathcal{FL}_0$

- Fragment of $\mathcal{ALC}$:

$$C ::= \top \mid A \mid C \sqcap C \mid \forall r.C$$

- Received much attention in the early days of DL research, but

The DL $\mathcal{FL}_0$

- Fragment of $\mathcal{ALC}$:

$$C ::= \top \mid A \mid C \sqcap C \mid \forall r.C$$

- Received much attention in the early days of DL research, but
  - it was later shown that subsumption w.r.t. acyclic TBoxes is not tractable,

The DL $\mathcal{FL}_0$

- Fragment of $\mathcal{ALC}$:

$$C ::= \top \mid A \mid C \sqcap C \mid \forall r.C$$

- Received much attention in the early days of DL research, but
  - it was later shown that subsumption w.r.t. acyclic TBoxes is not tractable,
  - and in the presence of GCIs ExpTime-complete. Same complexity as the more expressive DL $\mathcal{ALC}$.

The DL $\mathcal{FL}_0$

- Fragment of $\mathcal{ALC}$:

$$C ::= \top \mid A \mid C \sqcap C \mid \forall r.C$$

- Received much attention in the early days of DL research, but
  - it was later shown that subsumption w.r.t. acyclic TBoxes is not tractable,
  - and in the presence of GCIs ExpTime-complete. Same complexity as the more expressive DL $\mathcal{ALC}$.

- However,

The DL $\mathcal{FL}_0$

- Fragment of $\mathcal{ALC}$:

$$C ::= \top \mid A \mid C \sqcap C \mid \forall r.C$$

- Received much attention in the early days of DL research, but
  - it was later shown that subsumption w.r.t. acyclic TBoxes is not tractable,
  - and in the presence of GCIs ExpTime-complete. Same complexity as the more expressive DL $\mathcal{ALC}$.
- However,
  - Reasoning in $\mathcal{FL}_0$ has an interesting connection to formal language problems.

The DL $\mathcal{FL}_0$

- Fragment of $\mathcal{ALC}$:

$$C ::= \top \mid A \mid C \sqcap C \mid \forall r.C$$

- Received much attention in the early days of DL research, but
  - it was later shown that subsumption w.r.t. acyclic TBoxes is not tractable,
  - and in the presence of GCIs ExpTime-complete. Same complexity as the more expressive DL $\mathcal{ALC}$.

- However,
  - Reasoning in $\mathcal{FL}_0$ has an interesting connection to formal language problems.
  - The unification problem corresponds to unification in ACUIh.

Connection to formal languages - Subsumption

Connection to formal languages - Subsumption

Normal form

- Apply $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ as rewrite rule (from left to right):

$$\forall r.(\forall s.A \sqcap \forall r.B) \sqcap \forall r.A \sqcap B$$

Connection to formal languages - Subsumption

Normal form

- Apply $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ as rewrite rule (from left to right):

$$\forall r.(\forall s.A \sqcap \forall r.B) \sqcap \forall r.A \sqcap B \longrightarrow \forall r.\forall s.A \sqcap \forall r.\forall r.B \sqcap \forall r.A \sqcap B$$

Connection to formal languages - Subsumption

Normal form

- Apply $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ as rewrite rule (from left to right):

$$\forall r.(\forall s.A \sqcap \forall r.B) \sqcap \forall r.A \sqcap B \longrightarrow \forall r.\forall s.A \sqcap \forall r.\forall r.B \sqcap \forall r.A \sqcap B$$

- Abbreviate using languages over $\mathsf{N_R}$:

Connection to formal languages - Subsumption

Normal form

- Apply $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ as rewrite rule (from left to right):

$$\forall r.(\forall s.A \sqcap \forall r.B) \sqcap \forall r.A \sqcap B \longrightarrow \forall r.\forall s.A \sqcap \forall r.\forall r.B \sqcap \forall r.A \sqcap B$$

- Abbreviate using languages over $\mathsf{N_R}$:

$$\forall rs.A \sqcap \forall rr.B \sqcap \forall r.A \sqcap B$$

Connection to formal languages - Subsumption

Normal form

- Apply $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ as rewrite rule (from left to right):

$$\forall r.(\forall s.A \sqcap \forall r.B) \sqcap \forall r.A \sqcap B \longrightarrow \forall r.\forall s.A \sqcap \forall r.\forall r.B \sqcap \forall r.A \sqcap B$$

- Abbreviate using languages over $\mathsf{N_R}$:

$$\forall rs.A \sqcap \forall rr.B \sqcap \forall r.A \sqcap B \longrightarrow \forall \{rs, r\}.A \sqcap \forall \{rr, \varepsilon\}.B$$

Connection to formal languages - Subsumption

Normal form

- Apply $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ as rewrite rule (from left to right):

$$\forall r.(\forall s.A \sqcap \forall r.B) \sqcap \forall r.A \sqcap B \longrightarrow \forall r.\forall s.A \sqcap \forall r.\forall r.B \sqcap \forall r.A \sqcap B$$

- Abbreviate using languages over $\mathsf{N_R}$:

$$\forall rs.A \sqcap \forall rr.B \sqcap \forall r.A \sqcap B \longrightarrow \forall\{rs, r\}.A \sqcap \forall\{rr, \varepsilon\}.B$$

- Let $\mathsf{N_C} = \{A_1, \ldots, A_k\}$. Then, every pair of concepts C, D can be represented as:

$$C \equiv \forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k,$$
$$D \equiv \forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k,$$

  where L_i, R_i are finite languages over $\mathsf{N_R}$.

Connection to formal languages - Subsumption

Normal form

- Apply $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ as rewrite rule (from left to right):

$$\forall r.(\forall s.A \sqcap \forall r.B) \sqcap \forall r.A \sqcap B \longrightarrow \forall r.\forall s.A \sqcap \forall r.\forall r.B \sqcap \forall r.A \sqcap B$$

- Abbreviate using languages over $\mathsf{N_R}$:

$$\forall rs.A \sqcap \forall rr.B \sqcap \forall r.A \sqcap B \longrightarrow \forall\{rs, r\}.A \sqcap \forall\{rr, \varepsilon\}.B$$

- Let $\mathsf{N_C} = \{A_1, \ldots, A_k\}$. Then, every pair of concepts C, D can be represented as:

$$C \equiv \forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k,$$
$$D \equiv \forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k,$$

where L_i, R_i are finite languages over $\mathsf{N_R}$.

Characterization of subsumption [BN01]

$C \sqsubseteq D$ iff $R_i \subseteq L_i$ for all $i, 1 \leq i \leq k$.

Connection to formal languages - Subsumption

Normal form

- Apply $\forall r.(C \sqcap D) \equiv \forall r.C \sqcap \forall r.D$ as rewrite rule (from left to right):

$$\forall r.(\forall s.A \sqcap \forall r.B) \sqcap \forall r.A \sqcap B \longrightarrow \forall r.\forall s.A \sqcap \forall r.\forall r.B \sqcap \forall r.A \sqcap B$$

- Abbreviate using languages over $\mathsf{N_R}$:

$$\forall rs.A \sqcap \forall rr.B \sqcap \forall r.A \sqcap B \longrightarrow \forall \{rs, r\}.A \sqcap \forall \{rr, \varepsilon\}.B$$

- Let $\mathsf{N_C} = \{A_1, \ldots, A_k\}$. Then, every pair of concepts C, D can be represented as:

$$C \equiv \forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k,$$
$$D \equiv \forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k,$$

where L_i, R_i are finite languages over $\mathsf{N_R}$.

Characterization of subsumption [BN01]

$C \sqsubseteq D$ iff $R_i \subseteq L_i$ for all $i, 1 \leq i \leq k$. $\rightarrow$ Subsumption is polynomial w.r.t. $\mathcal{T} = \emptyset$.

Connection to formal languages - Unification

Let $\Gamma = \{C \equiv^? D\}$.

Connection to formal languages - Unification

Let $\Gamma = \{C \equiv^? D\}$.

Considering variables as concept names, Γ can be seen as:

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^?$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k \sqcap \forall R_1^*.X_1 \sqcap \ldots \sqcap \forall R_m^*.X_m$$

Connection to formal languages - Unification

Let $\Gamma = \{C \equiv^? D\}$.

Considering variables as concept names, Γ can be seen as:

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^?$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k \sqcap \forall R_1^*.X_1 \sqcap \ldots \sqcap \forall R_m^*.X_m$$

Example

$\forall r.(A_1 \sqcap \forall r.A_2) \sqcap \forall r.\forall s.X_1 \equiv^? \forall r.\forall s.(\forall s.A_1 \sqcap \forall r.A_2) \sqcap \forall r.X_1 \sqcap \forall r.\forall r.A_2$

Connection to formal languages - Unification

Let $\Gamma = \{C \equiv^{?} D\}$.

Considering variables as concept names, Γ can be seen as:

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^{?}$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k \sqcap \forall R_1^*.X_1 \sqcap \ldots \sqcap \forall R_m^*.X_m$$

Example

$$\forall r.(A_1 \sqcap \forall r.A_2) \sqcap \forall r.\forall s.X_1 \equiv^{?} \forall r.\forall s.(\forall s.A_1 \sqcap \forall r.A_2) \sqcap \forall r.X_1 \sqcap \forall r.\forall r.A_2$$

$$\downarrow$$

$$\forall\{r\}.A_1 \sqcap \forall\{rr\}.A_2 \sqcap \forall\{rs\}.X_1 \equiv^{?} \forall\{rss\}.A_1 \sqcap \forall\{rsr, rr\}.A_2 \sqcap \forall\{r\}.X_1$$

Connection to formal languages - Unification

Let $\Gamma = \{C \equiv^? D\}$.

Considering variables as concept names, Γ can be seen as:

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^?$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k \sqcap \forall R_1^*.X_1 \sqcap \ldots \sqcap \forall R_m^*.X_m$$

Example

$$\forall r.(A_1 \sqcap \forall r.A_2) \sqcap \forall r.\forall s.X_1 \equiv^? \forall r.\forall s.(\forall s.A_1 \sqcap \forall r.A_2) \sqcap \forall r.X_1 \sqcap \forall r.\forall r.A_2$$

$$\downarrow$$

$$\forall\{r\}.A_1 \sqcap \forall\{rr\}.A_2 \sqcap \forall\{rs\}.X_1 \equiv^? \forall\{rss\}.A_1 \sqcap \forall\{rsr, rr\}.A_2 \sqcap \forall\{r\}.X_1$$

What to replace X_1 for to make the resulting forms "equal"?

Connection to formal languages - Unification

Let $\Gamma = \{C \equiv^? D\}$.

Considering variables as concept names, Γ can be seen as:

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^?$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k \sqcap \forall R_1^*.X_1 \sqcap \ldots \sqcap \forall R_m^*.X_m$$

Example

$$\forall r.(A_1 \sqcap \forall r.A_2) \sqcap \forall r.\forall s.X_1 \equiv^? \forall r.\forall s.(\forall s.A_1 \sqcap \forall r.A_2) \sqcap \forall r.X_1 \sqcap \forall r.\forall r.A_2$$

$$\downarrow$$

$$\forall\{r\}.A_1 \sqcap \forall\{rr\}.A_2 \sqcap \forall\{rs\}.X_1 \equiv^? \forall\{rss\}.A_1 \sqcap \forall\{rsr, rr\}.A_2 \sqcap \forall\{r\}.X_1$$

What to replace X_1 for to make the resulting forms “equal”?

$X_1 \mapsto A_1 \sqcap \forall s.A_1 \sqcap \forall r.A_2$

Connection to formal languages - Unification

Let $\Gamma = \{C \equiv^? D\}$.

Considering variables as concept names, Γ can be seen as:

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^?$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k \sqcap \forall R_1^*.X_1 \sqcap \ldots \sqcap \forall R_m^*.X_m$$

Example

$$\forall r.(A_1 \sqcap \forall r.A_2) \sqcap \forall r.\forall s.X_1 \equiv^? \forall r.\forall s.(\forall s.A_1 \sqcap \forall r.A_2) \sqcap \forall r.X_1 \sqcap \forall r.\forall r.A_2$$

$$\downarrow$$

$$\forall\{r\}.A_1 \sqcap \forall\{rr\}.A_2 \sqcap \forall\{rs\}.X_1 \equiv^? \forall\{rss\}.A_1 \sqcap \forall\{rsr, rr\}.A_2 \sqcap \forall\{r\}.X_1$$

What to replace X_1 for to make the resulting forms "equal"?

$X_1 \mapsto A_1 \sqcap \forall s.A_1 \sqcap \forall r.A_2$

$$A_1\text{: } \{r\} \cup \{rs.\varepsilon, rs.s\} = \{rss\} \cup \{r.\epsilon, r.s\}$$

Connection to formal languages - Unification

Let $\Gamma = \{C \equiv^? D\}$.

Considering variables as concept names, Γ can be seen as:

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^?$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k \sqcap \forall R_1^*.X_1 \sqcap \ldots \sqcap \forall R_m^*.X_m$$

Example

$$\forall r.(A_1 \sqcap \forall r.A_2) \sqcap \forall r.\forall s.X_1 \equiv^? \forall r.\forall s.(\forall s.A_1 \sqcap \forall r.A_2) \sqcap \forall r.X_1 \sqcap \forall r.\forall r.A_2$$
$$\downarrow$$
$$\forall\{r\}.A_1 \sqcap \forall\{rr\}.A_2 \sqcap \forall\{rs\}.X_1 \equiv^? \forall\{rss\}.A_1 \sqcap \forall\{rsr, rr\}.A_2 \sqcap \forall\{r\}.X_1$$

What to replace X_1 for to make the resulting forms "equal"?

$X_1 \mapsto A_1 \sqcap \forall s.A_1 \sqcap \forall r.A_2$

$$A_1\colon\ \{r\} \cup \{rs.\varepsilon, rs.s\} = \{rss\} \cup \{r.\epsilon, r.s\}$$

$$A_2\colon\ \{rr\} \cup \{rs.r\} = \{rsr, rr\} \cup \{r.r\}$$

Connection to formal languages - Unification

Let $\Gamma = \{C \equiv^? D\}$.

Considering variables as concept names, Γ can be seen as:

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^?$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k \sqcap \forall R_1^*.X_1 \sqcap \ldots \sqcap \forall R_m^*.X_m$$

Example

$$\forall r.(A_1 \sqcap \forall r.A_2) \sqcap \forall r.\forall s.X_1 \equiv^? \forall r.\forall s.(\forall s.A_1 \sqcap \forall r.A_2) \sqcap \forall r.X_1 \sqcap \forall r.\forall r.A_2$$

$$\downarrow$$

$$\forall\{r\}.A_1 \sqcap \forall\{rr\}.A_2 \sqcap \forall\{rs\}.X_1 \equiv^? \forall\{rss\}.A_1 \sqcap \forall\{rsr, rr\}.A_2 \sqcap \forall\{r\}.X_1$$

What to replace X_1 for to make the resulting forms "equal"?

$X_1 \mapsto A_1 \sqcap \forall s.A_1 \sqcap \forall r.A_2$

A_1: $\{r\} \cup \{rs.\varepsilon, rs.s\} = \{rss\} \cup \{r.\epsilon, r.s\}$

A_2: $\{rr\} \cup \{rs.r\} = \{rsr, rr\} \cup \{r.r\}$

Fixing A_1 is idependent of fixing A_2, and vice versa!

Unification in $\mathcal{FL}_0 \rightarrow$ Solving linear equations

Unification in $\mathcal{FL}_0 \rightarrow$ Solving linear equations

Example (continuation)

Unification in $\mathcal{FL}_0 \to$ Solving linear equations

Example (continuation)

$$\{r\} \cup \{rs\}.X_{1,A_1} =^? \{rss\} \cup \{r\}.X_{1,A_1}$$

Unification in $\mathcal{FL}_0 \to$ Solving linear equations

Example (continuation)

$$\{r\} \cup \{rs\}.X_{1,A_1} =^? \{rss\} \cup \{r\}.X_{1,A_1}$$

$$\{rr\} \cup \{rs\}.X_{1,A_2} =^? \{rsr, rr\} \cup \{r\}.X_{1,A_2}$$

Unification in $\mathcal{FL}_0 \rightarrow$ Solving linear equations

Example (continuation)

$$\{r\} \cup \{rs\}.X_{1,A_1} =^? \{rss\} \cup \{r\}.X_{1,A_1}$$

$$\{rr\} \cup \{rs\}.X_{1,A_2} =^? \{rsr, rr\} \cup \{r\}.X_{1,A_2}$$

Formally,

Unification in $\mathcal{FL}_0 \rightarrow$ Solving linear equations

Example (continuation)

$$\{r\} \cup \{rs\}.X_{1,A_1} =^? \{rss\} \cup \{r\}.X_{1,A_1}$$

$$\{rr\} \cup \{rs\}.X_{1,A_2} =^? \{rsr, rr\} \cup \{r\}.X_{1,A_2}$$

Formally,

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^?$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k \sqcap \forall R_1^*.X_1 \sqcap \ldots \sqcap \forall R_m^*.X_m$$

has a solution **iff** for all $1 \leq i \leq k$ the equation:

$$L_i \cup L_1^*.X_{1,i} \cup \ldots \cup L_m^*.X_{1,m}$$
$$=^?$$
$$R_i \cup R_1^*.X_{1,i} \cup \ldots \cup R_m^*.X_{1,m}$$

has a finite solution.

Unification in $\mathcal{FL}_0$ → Solving linear equations

Example (continuation)

$$\{r\} \cup \{rs\}.X_{1,A_1} =^? \{rss\} \cup \{r\}.X_{1,A_1}$$

$$\{rr\} \cup \{rs\}.X_{1,A_2} =^? \{rsr, rr\} \cup \{r\}.X_{1,A_2}$$

Formally,

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m \\ \equiv^? \\ \forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k \sqcap \forall R_1^*.X_1 \sqcap \ldots \sqcap \forall R_m^*.X_m$$

has a solution **iff** for all $1 \leq i \leq k$ the equation:

$$L_i \cup L_1^*.X_{1,i} \cup \ldots \cup L_m^*.X_{1,m} \\ =^? \\ R_i \cup R_1^*.X_{1,i} \cup \ldots \cup R_m^*.X_{1,m}$$

has a finite solution.

Why finite?

How to solve these linear equations?

How to solve these linear equations?

- A solution of a single equation can be seen as a finite language:

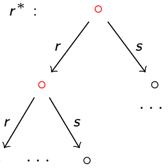
  in our example, $\mathcal{L}_1 = \{r, rs, rss\}$ and $\mathcal{L}_2 = \{rr, rsr\}$ for A_1 and A_2, resp.

How to solve these linear equations?

- A solution of a single equation can be seen as a finite language:

  in our example, $\mathcal{L}_1 = \{r, rs, rss\}$ and $\mathcal{L}_2 = \{rr, rsr\}$ for A_1 and A_2, resp.

- A language over a finite alphabet can be represented as a tree:

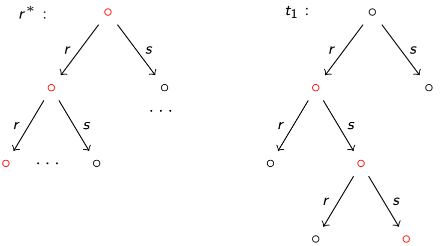


How to solve these linear equations?

- A solution of a single equation can be seen as a finite language:

  in our example, $\mathcal{L}_1 = \{r, rs, rss\}$ and $\mathcal{L}_2 = \{rr, rsr\}$ for A_1 and A_2, resp.

- A language over a finite alphabet can be represented as a tree:

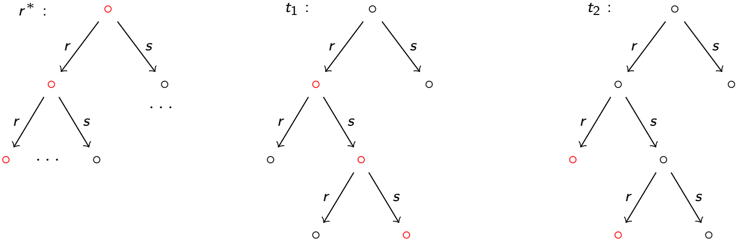


How to solve these linear equations?

- A solution of a single equation can be seen as a finite language:

  in our example, $\mathcal{L}_1 = \{r, rs, rss\}$ and $\mathcal{L}_2 = \{rr, rsr\}$ for A_1 and A_2, resp.

- A language over a finite alphabet can be represented as a tree:

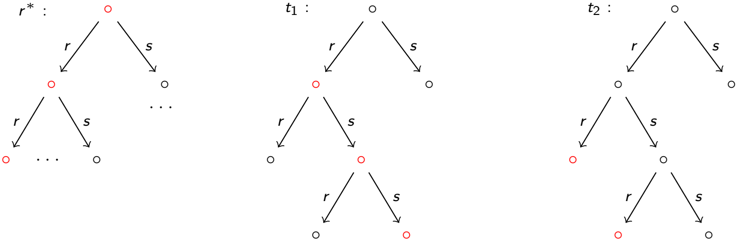


How to solve these linear equations?

- A solution of a single equation can be seen as a finite language:

  in our example, $\mathcal{L}_1 = \{r, rs, rss\}$ and $\mathcal{L}_2 = \{rr, rsr\}$ for A_1 and A_2, resp.

- A language over a finite alphabet can be represented as a tree:



- Construct a finite tree automata $\mathcal{A}_\mathcal{E}$ such that

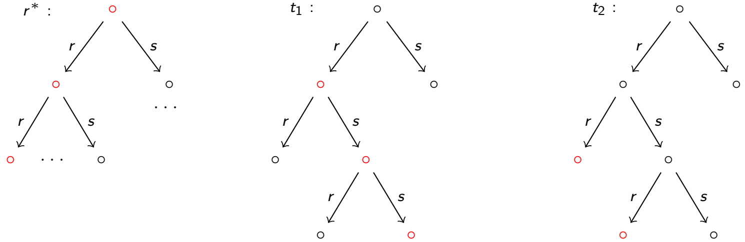
$$\mathcal{L}(\mathcal{A}_\mathcal{E}) \neq \emptyset \textbf{ iff } \mathcal{E} \text{ has a solution.}$$

How to solve these linear equations?

- A solution of a single equation can be seen as a finite language:

  in our example, $\mathcal{L}_1 = \{r, rs, rss\}$ and $\mathcal{L}_2 = \{rr, rsr\}$ for A_1 and A_2, resp.

- A language over a finite alphabet can be represented as a tree:



- Construct a finite tree automata $\mathcal{A}_\mathcal{E}$ such that

$$\mathcal{L}(\mathcal{A}_\mathcal{E}) \neq \emptyset \textbf{ iff } \mathcal{E} \text{ has a solution.}$$

- $\mathcal{A}_\mathcal{E}$ accepts exactly the trees representing solutions of an equation.

Detour to finite tree automata

Detour to finite tree automata

Definition 11 (Σ-tree)

Let Σ be a finite alphabet such that each $f \in \Sigma$ has a rank $\mathrm{rk}(f) \geq 0$. A finite Σ-tree is a mapping $t : \mathrm{dom}(t) \to \Sigma$ such that:

Detour to finite tree automata

Definition 11 (Σ-tree)

Let Σ be a finite alphabet such that each $f \in \Sigma$ has a rank $\mathrm{rk}(f) \geq 0$. A finite Σ-tree is a mapping $t : \mathrm{dom}(t) \to \Sigma$ such that:

- $\mathrm{dom}(t)$ is a finite subset of $\{1, \ldots, \max(\mathrm{rk}(f))\}^*$,

Detour to finite tree automata

Definition 11 (Σ-tree)

Let Σ be a finite alphabet such that each $f \in \Sigma$ has a rank $\mathrm{rk}(f) \geq 0$. A finite Σ-tree is a mapping $t : \mathrm{dom}(t) \to \Sigma$ such that:

- $\mathrm{dom}(t)$ is a finite subset of $\{1, \ldots, \max(\mathrm{rk}(f))\}^*$,
- $\varepsilon \in \mathrm{dom}(t)$,

Detour to finite tree automata

Definition 11 (Σ-tree)

Let Σ be a finite alphabet such that each $f \in \Sigma$ has a rank $\mathrm{rk}(f) \geq 0$. A finite Σ-tree is a mapping $t : \mathrm{dom}(t) \to \Sigma$ such that:

- $\mathrm{dom}(t)$ is a finite subset of $\{1, \ldots, \max(\mathrm{rk}(f))\}^*$,
- $\varepsilon \in \mathrm{dom}(t)$,
- $u.i \in \mathrm{dom}(t)$ iff $u \in \mathrm{dom}(t)$ and $i \leq \mathrm{rk}(t(u))$.

Detour to finite tree automata

Definition 11 (Σ-tree)

Let Σ be a finite alphabet such that each $f \in \Sigma$ has a rank $\mathsf{rk}(f) \geq 0$. A finite Σ-tree is a mapping $t : \mathsf{dom}(t) \to \Sigma$ such that:

- $\mathsf{dom}(t)$ is a finite subset of $\{1, \ldots, \max(\mathsf{rk}(f))\}^*$,
- $\varepsilon \in \mathsf{dom}(t)$,
- $u.i \in \mathsf{dom}(t)$ iff $u \in \mathsf{dom}(t)$ and $i \leq \mathsf{rk}(t(u))$.

Leaves of t must be mapped to constants, i.e., arity 0.

Detour to finite tree automata

Definition 11 (Σ-tree)

Let Σ be a finite alphabet such that each $f \in \Sigma$ has a rank $\mathrm{rk}(f) \geq 0$. A finite Σ-tree is a mapping $t : \mathrm{dom}(t) \to \Sigma$ such that:

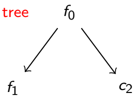
- $\mathrm{dom}(t)$ is a finite subset of $\{1, \dots, \max(\mathrm{rk}(f))\}^*$,
- $\varepsilon \in \mathrm{dom}(t)$,
- $u.i \in \mathrm{dom}(t)$ iff $u \in \mathrm{dom}(t)$ and $i \leq \mathrm{rk}(t(u))$.

Leaves of t must be mapped to constants, i.e., arity 0.

Example

Let $\Sigma = \{f_0^{(2)}, f_1^{(2)}, c_0^{(0)}, c_1^{(0)}\}$

not a tree



Detour to finite tree automata

Definition 11 (Σ-tree)

Let Σ be a finite alphabet such that each $f \in \Sigma$ has a rank $\mathrm{rk}(f) \geq 0$. A finite Σ-tree is a mapping $t : \mathrm{dom}(t) \to \Sigma$ such that:

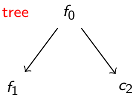
- $\mathrm{dom}(t)$ is a finite subset of $\{1, \ldots, \max(\mathrm{rk}(f))\}^*$,
- $\varepsilon \in \mathrm{dom}(t)$,
- $u.i \in \mathrm{dom}(t)$ iff $u \in \mathrm{dom}(t)$ and $i \leq \mathrm{rk}(t(u))$.

Leaves of t must be mapped to constants, i.e., arity 0.

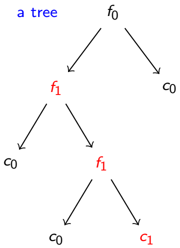
Example

Let $\Sigma = \{f_0^{(2)}, f_1^{(2)}, c_0^{(0)}, c_1^{(0)}\}$

not a tree



a tree



Detour to finite tree automata

Definition 11 (Σ-tree)

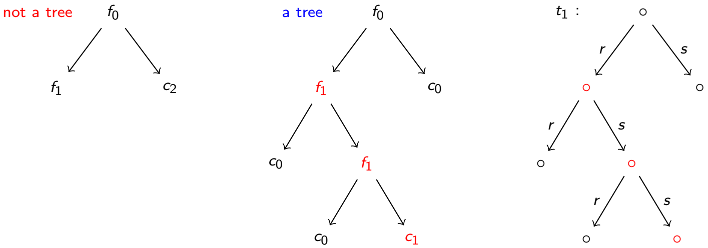
Let Σ be a finite alphabet such that each $f \in \Sigma$ has a rank $\mathrm{rk}(f) \geq 0$. A finite Σ-tree is a mapping $t : \mathrm{dom}(t) \to \Sigma$ such that:

- $\mathrm{dom}(t)$ is a finite subset of $\{1, \ldots, \max(\mathrm{rk}(f))\}^*$,
- $\varepsilon \in \mathrm{dom}(t)$,
- $u.i \in \mathrm{dom}(t)$ iff $u \in \mathrm{dom}(t)$ and $i \leq \mathrm{rk}(t(u))$.

Leaves of t must be mapped to constants, i.e., arity 0.

Example

Let $\Sigma = \{f_0^{(2)}, f_1^{(2)}, c_0^{(0)}, c_1^{(0)}\}$



Detour to finite tree automata

Detour to finite tree automata

Definition 12 (Root-to-frontier tree automata)

A root-to-frontier tree automaton (RFA) on Σ-trees is a tuple $\mathcal{A} = (\Sigma, Q, I, \Delta, F)$ where:

- Q is a finite set of states.
- $I \subseteq Q$ is the set of initial states.
- Δ is a transition function s.t.:
$$\forall f \in \Sigma,\ \mathrm{rk}(f) = n > 0\colon\ \Delta(f) \subseteq Q \times Q^n.$$
- $F : \Sigma_0 \to 2^Q$ (the acceptance condition).

Detour to finite tree automata

Definition 12 (Root-to-frontier tree automata)

A root-to-frontier tree automaton (RFA) on Σ-trees is a tuple $\mathcal{A} = (\Sigma, Q, I, \Delta, F)$ where:

- Q is a finite set of states.
- $I \subseteq Q$ is the set of initial states.
- Δ is a transition function s.t.:

$$\forall f \in \Sigma,\ \mathrm{rk}(f) = n > 0\colon\ \Delta(f) \subseteq Q \times Q^n.$$

- $F : \Sigma_0 \to 2^Q$ (the acceptance condition).

A run ρ of $\mathcal{A}$ on a Σ-tree t is a mapping $\rho : \mathrm{dom}(t) \to Q$ such that:

$$(\rho(u), \rho(u.1), \ldots, \rho(u.n)) \in \Delta, \text{ for all } u \text{ with } \mathrm{rk}(u) = n > 0.$$

Detour to finite tree automata

Definition 12 (Root-to-frontier tree automata)

A root-to-frontier tree automaton (RFA) on Σ-trees is a tuple $\mathcal{A} = (\Sigma, Q, I, \Delta, F)$ where:

- Q is a finite set of states.
- $I \subseteq Q$ is the set of initial states.
- Δ is a transition function s.t.:

$$\forall f \in \Sigma,\ \mathrm{rk}(f) = n > 0\text{:}\ \ \Delta(f) \subseteq Q \times Q^n.$$

- $F : \Sigma_0 \to 2^Q$ (the acceptance condition).

A run ρ of $\mathcal{A}$ on a Σ-tree t is a mapping $\rho : \mathrm{dom}(t) \to Q$ such that:

$$(\rho(u), \rho(u.1), \ldots, \rho(u.n)) \in \Delta, \text{ for all } u \text{ with } \mathrm{rk}(u) = n > 0.$$

This run is successful if,

- $\rho(\varepsilon) \in I$
- $\rho(u) \in F(t(u))$, for all leaves u of t.

Detour to finite tree automata

Definition 12 (Root-to-frontier tree automata)

A root-to-frontier tree automaton (RFA) on Σ-trees is a tuple $\mathcal{A} = (\Sigma, Q, I, \Delta, F)$ where:

- Q is a finite set of states.
- $I \subseteq Q$ is the set of initial states.
- Δ is a transition function s.t.:
$$\forall f \in \Sigma,\ \mathrm{rk}(f) = n > 0\text{: } \Delta(f) \subseteq Q \times Q^n.$$
- $F : \Sigma_0 \to 2^Q$ (the acceptance condition).

A run ρ of $\mathcal{A}$ on a Σ-tree t is a mapping $\rho : \mathrm{dom}(t) \to Q$ such that:

$$(\rho(u), \rho(u.1), \ldots, \rho(u.n)) \in \Delta, \text{ for all } u \text{ with } \mathrm{rk}(u) = n > 0.$$

This run is successful if,

- $\rho(\varepsilon) \in I$
- $\rho(u) \in F(t(u))$, for all leaves u of t.

Tree language accepted by $\mathcal{A}$:

$$\mathcal{L}(\mathcal{A}) = \{t \mid \mathcal{A} \text{ has a successful run on } t\}.$$

Detour to finite tree automata - Example

Let $\Sigma = \{f_0^{(2)}, f_1^{(2)}, c_0^{(0)}, c_1^{(0)}\}$. Construct an automaton $\mathcal{A}$ that accepts

$$\mathcal{L} = \{t \mid \exists u \in \mathrm{dom}(t) \text{ s.t. } u.rs \text{ is a } c_1 \text{ leaf}\}.$$

Detour to finite tree automata - Example

Let $\Sigma = \{f_0^{(2)}, f_1^{(2)}, c_0^{(0)}, c_1^{(0)}\}$. Construct an automaton $\mathcal{A}$ that accepts

$$\mathcal{L} = \{t \mid \exists u \in \mathrm{dom}(t) \text{ s.t. } u.rs \text{ is a } c_1 \text{ leaf}\}.$$

Idea.

Detour to finite tree automata - Example

Let $\Sigma = \{f_0^{(2)}, f_1^{(2)}, c_0^{(0)}, c_1^{(0)}\}$. Construct an automaton $\mathcal{A}$ that accepts

$$\mathcal{L} = \{t \mid \exists u \in \mathrm{dom}(t) \text{ s.t. } u.rs \text{ is a } c_1 \text{ leaf}\}.$$

Idea.

- At every moment, $\mathcal{A}$ will guess whether u is in the current subtree:

Detour to finite tree automata - Example

Let $\Sigma = \{f_0^{(2)}, f_1^{(2)}, c_0^{(0)}, c_1^{(0)}\}$. Construct an automaton $\mathcal{A}$ that accepts

$$\mathcal{L} = \{t \mid \exists u \in \mathrm{dom}(t) \text{ s.t. } u.rs \text{ is a } c_1 \text{ leaf}\}.$$

Idea.

- At every moment, $\mathcal{A}$ will guess whether u is in the current subtree:
  - state $q_{\mathsf{N}} \to$ "not in this subtree".

Detour to finite tree automata - Example

Let $\Sigma = \{f_0^{(2)}, f_1^{(2)}, c_0^{(0)}, c_1^{(0)}\}$. Construct an automaton $\mathcal{A}$ that accepts

$$\mathcal{L} = \{t \mid \exists u \in \mathrm{dom}(t) \text{ s.t. } u.rs \text{ is a } c_1 \text{ leaf}\}.$$

Idea.

- At every moment, $\mathcal{A}$ will guess whether u is in the current subtree:
  - state $q_{\mathrm{N}} \rightarrow$ "not in this subtree".
  - states $q_{s0}, q_{s1} \rightarrow$ "it is in the subtree", "it is actually the root of the subtree".

Detour to finite tree automata - Example

Let $\Sigma = \{f_0^{(2)}, f_1^{(2)}, c_0^{(0)}, c_1^{(0)}\}$. Construct an automaton $\mathcal{A}$ that accepts

$$\mathcal{L} = \{t \mid \exists u \in \mathrm{dom}(t) \text{ s.t. } u.rs \text{ is a } c_1 \text{ leaf}\}.$$

Idea.

- At every moment, $\mathcal{A}$ will guess whether u is in the current subtree:
  - state $q_{\mathrm{N}} \rightarrow$ "not in this subtree".
  - states $q_{s0}, q_{s1} \rightarrow$ "it is in the subtree", "it is actually the root of the subtree".
- Guessing (use non-determinism):

Detour to finite tree automata - Example

Let $\Sigma = \{f_0^{(2)}, f_1^{(2)}, c_0^{(0)}, c_1^{(0)}\}$. Construct an automaton $\mathcal{A}$ that accepts

$$\mathcal{L} = \{t \mid \exists u \in \mathrm{dom}(t) \text{ s.t. } u.rs \text{ is a } c_1 \text{ leaf}\}.$$

Idea.

- At every moment, $\mathcal{A}$ will guess whether u is in the current subtree:
    - state $q_{\mathrm{N}} \rightarrow$ "not in this subtree".
    - states $q_{s0}, q_{s1} \rightarrow$ "it is in the subtree", "it is actually the root of the subtree".
- Guessing (use non-determinism):
    - $(q_{s0}, f_0/f_1, q_{\mathrm{N}}, q_{s0}/q_{s1})$ and $(q_{s0}, f_0/f_1, q_{s0}/q_{s1}, q_{\mathrm{N}})$.

Detour to finite tree automata - Example

Let $\Sigma = \{f_0^{(2)}, f_1^{(2)}, c_0^{(0)}, c_1^{(0)}\}$. Construct an automaton $\mathcal{A}$ that accepts

$$\mathcal{L} = \{t \mid \exists u \in \mathrm{dom}(t) \text{ s.t. } u.rs \text{ is a } c_1 \text{ leaf}\}.$$

Idea.

- At every moment, $\mathcal{A}$ will guess whether u is in the current subtree:
  - state $q_\mathrm{N} \to$ "not in this subtree".
  - states $q_{s0}, q_{s1} \to$ "it is in the subtree", "it is actually the root of the subtree".
- Guessing (use non-determinism):
  - $(q_{s0}, f_0/f_1, q_\mathrm{N}, q_{s0}/q_{s1})$ and $(q_{s0}, f_0/f_1, q_{s0}/q_{s1}, q_\mathrm{N})$.
  - $(q_\mathrm{N}, f_0/f_1, q_\mathrm{N}, q_\mathrm{N})$

Detour to finite tree automata - Example

Let $\Sigma = \{f_0^{(2)}, f_1^{(2)}, c_0^{(0)}, c_1^{(0)}\}$. Construct an automaton $\mathcal{A}$ that accepts

$$\mathcal{L} = \{t \mid \exists u \in \mathrm{dom}(t) \text{ s.t. } u.rs \text{ is a } c_1 \text{ leaf}\}.$$

Idea.

- At every moment, $\mathcal{A}$ will guess whether u is in the current subtree:
  - state $q_\mathrm{N} \to$ "not in this subtree".
  - states $q_{s0}, q_{s1} \to$ "it is in the subtree", "it is actually the root of the subtree".
- Guessing (use non-determinism):
  - $(q_{s0}, f_0/f_1, q_\mathrm{N}, q_{s0}/q_{s1})$ and $(q_{s0}, f_0/f_1, q_{s0}/q_{s1}, q_\mathrm{N})$.
  - $(q_\mathrm{N}, f_0/f_1, q_\mathrm{N}, q_\mathrm{N})$
- Checking the suffix once in q_{s1} (use two additional states q_s, q_ε):
  - $(q_{s1}, f_0/f_1, q_s, q_\mathrm{N})$ and $(q_s, f_0/f_1, q_\mathrm{N}, q_\varepsilon)$.

Detour to finite tree automata - Example

Let $\Sigma = \{f_0^{(2)}, f_1^{(2)}, c_0^{(0)}, c_1^{(0)}\}$. Construct an automaton $\mathcal{A}$ that accepts

$$\mathcal{L} = \{t \mid \exists u \in \mathrm{dom}(t) \text{ s.t. } u.rs \text{ is a } c_1 \text{ leaf}\}.$$

Idea.

- At every moment, $\mathcal{A}$ will guess whether u is in the current subtree:
  - state $q_{\mathrm{N}} \rightarrow$ "not in this subtree".
  - states $q_{s0}, q_{s1} \rightarrow$ "it is in the subtree", "it is actually the root of the subtree".
- Guessing (use non-determinism):
  - $(q_{s0}, f_0/f_1, q_{\mathrm{N}}, q_{s0}/q_{s1})$ and $(q_{s0}, f_0/f_1, q_{s0}/q_{s1}, q_{\mathrm{N}})$.
  - $(q_{\mathrm{N}}, f_0/f_1, q_{\mathrm{N}}, q_{\mathrm{N}})$
- Checking the suffix once in q_{s1} (use two additional states q_s, q_ε):
  - $(q_{s1}, f_0/f_1, q_s, q_{\mathrm{N}})$ and $(q_s, f_0/f_1, q_{\mathrm{N}}, q_\varepsilon)$.
- Acceptance condition:
  - $F(c_0) = \{q_{\mathrm{N}}\}$ and $F(c_1) = \{q_{\mathrm{N}}, q_\varepsilon\}$.

Solving linear equations

Recall: we want to construct an RFA $\mathcal{A}_{\mathcal{E}}$ accepting exactly the "trees" solving

$$L_i \cup L_1^*.X_{1,i} \cup \ldots \cup L_m^*.X_{1,m}$$
$$=^?$$
$$R_i \cup R_1^*.X_{1,i} \cup \ldots \cup R_m^*.X_{1,m}.$$

Solving linear equations

Recall: we want to construct an RFA $\mathcal{A}_{\mathcal{E}}$ accepting exactly the "trees" solving

$$L_i \cup L_1^*.X_{1,i} \cup \ldots \cup L_m^*.X_{1,m}$$
$$=^?$$
$$R_i \cup R_1^*.X_{1,i} \cup \ldots \cup R_m^*.X_{1,m}.$$

Actually, we will instead build one accepting the solutions of

$$\overline{L_i} \cup Y_{1,i}.\overline{L_1^*} \cup \ldots \cup Y_{1,m}.\overline{L_m^*}$$
$$=^?$$
$$\overline{R_i} \cup Y_{1,i}.\overline{R_1^*} \cup \ldots \cup Y_{1,m}.\overline{R_m^*},$$

Solving linear equations

Recall: we want to construct an RFA $\mathcal{A}_\mathcal{E}$ accepting exactly the "trees" solving

$$L_i \cup L_1^*.X_{1,i} \cup \ldots \cup L_m^*.X_{1,m}$$
$$=^?$$
$$R_i \cup R_1^*.X_{1,i} \cup \ldots \cup R_m^*.X_{1,m}.$$

Actually, we will instead build one accepting the solutions of

$$\overline{L_i} \cup Y_{1,i}.\overline{L_1^*} \cup \ldots \cup Y_{1,m}.\overline{L_m^*}$$
$$=^?$$
$$\overline{R_i} \cup Y_{1,i}.\overline{R_1^*} \cup \ldots \cup Y_{1,m}.\overline{R_m^*},$$

where $\overline{w} = u_m \ldots u_1$, for $w = u_1 \ldots u_m \in \mathsf{N_R}^*$; and $\overline{\mathcal{L}} = \{\overline{w} \mid w \in \mathcal{L}\}$.

Solving linear equations

Recall: we want to construct an RFA $\mathcal{A}_{\mathcal{E}}$ accepting exactly the "trees" solving

$$L_i \cup L_1^*.X_{1,i} \cup \ldots \cup L_m^*.X_{1,m}$$
$$=^?$$
$$R_i \cup R_1^*.X_{1,i} \cup \ldots \cup R_m^*.X_{1,m}.$$

Actually, we will instead build one accepting the solutions of

$$\overline{L_i} \cup Y_{1,i}.\overline{L_1^*} \cup \ldots \cup Y_{1,m}.\overline{L_m^*}$$
$$=^?$$
$$\overline{R_i} \cup Y_{1,i}.\overline{R_1^*} \cup \ldots \cup Y_{1,m}.\overline{R_m^*},$$

where $\overline{w} = u_m \ldots u_1$, for $w = u_1 \ldots u_m \in \mathsf{N_R}^*$; and $\overline{\mathcal{L}} = \{\overline{w} \mid w \in \mathcal{L}\}$.

Property

$$X_{1,i}, \ldots, X_{1,m} \text{ solves } \mathcal{E}$$
iff
$$Y_{1,i} = \overline{X_{1,i}}, \ldots, Y_{1,m} = \overline{X_{1,m}} \text{ solves } \overline{\mathcal{E}}.$$

Construction of the automaton - Example

Construction of the automaton - Example

Original equation and solution:

Construction of the automaton - Example

Original equation and solution:

$$\mathcal{E}: \quad \{r\} \cup \{rs\}.X_{1,A_1} =^? \{rss\} \cup \{r\}.X_{1,A_1}$$

Construction of the automaton - Example

Original equation and solution:

$$\mathcal{E}: \quad \{r\} \cup \{rs\}.X_{1,A_1} =^? \{rss\} \cup \{r\}.X_{1,A_1}$$

$$X_{1,A_1} \mapsto \{\varepsilon, s\} \text{ and } \mathcal{L}_1 = \{r, rs, rss\}$$

Construction of the automaton - Example

Original equation and solution:

$$\mathcal{E}: \quad \{r\} \cup \{rs\}.X_{1,A_1} =^{?} \{rss\} \cup \{r\}.X_{1,A_1}$$

$$X_{1,A_1} \mapsto \{\varepsilon, s\} \text{ and } \mathcal{L}_1 = \{r, rs, rss\}$$

Reverse equation and solution:

Construction of the automaton - Example

Original equation and solution:

$$\mathcal{E}: \quad \{r\} \cup \{rs\}.X_{1,A_1} =^{?} \{rss\} \cup \{r\}.X_{1,A_1}$$
$$X_{1,A_1} \mapsto \{\varepsilon, s\} \text{ and } \mathcal{L}_1 = \{r, rs, rss\}$$

Reverse equation and solution:

$$\overline{\mathcal{E}}: \quad \{r\} \cup Y_{1,A_1}.\{sr\} =^{?} \{ssr\} \cup Y_{1,A_1}.\{r\}$$

Construction of the automaton - Example

Original equation and solution:

$$\mathcal{E}: \quad \{r\} \cup \{rs\}.X_{1,A_1} \stackrel{?}{=} \{rss\} \cup \{r\}.X_{1,A_1}$$

$$X_{1,A_1} \mapsto \{\varepsilon, s\} \text{ and } \mathcal{L}_1 = \{r, rs, rss\}$$

Reverse equation and solution:

$$\overline{\mathcal{E}}: \quad \{r\} \cup Y_{1,A_1}.\{sr\} \stackrel{?}{=} \{ssr\} \cup Y_{1,A_1}.\{r\}$$

$$Y_{1,A_1} \mapsto \{\varepsilon, s\} \text{ and } \mathcal{L}_1 = \{r, sr, ssr\}$$

Construction of the automaton - Example

Original equation and solution:

$$\mathcal{E} : \quad \{r\} \cup \{rs\}.X_{1,A_1} \stackrel{?}{=} \{rss\} \cup \{r\}.X_{1,A_1}$$

$$X_{1,A_1} \mapsto \{\varepsilon, s\} \text{ and } \mathcal{L}_1 = \{r, rs, rss\}$$

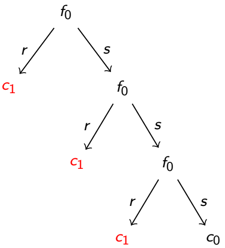
Reverse equation and solution:

$$\overline{\mathcal{E}} : \quad \{r\} \cup Y_{1,A_1}.\{sr\} \stackrel{?}{=} \{ssr\} \cup Y_{1,A_1}.\{r\}$$

$$Y_{1,A_1} \mapsto \{\varepsilon, s\} \text{ and } \mathcal{L}_1 = \{r, sr, ssr\}$$

The automaton must accept the tree:

t :



Construction of the automaton - Example

Reverse equation and solution:

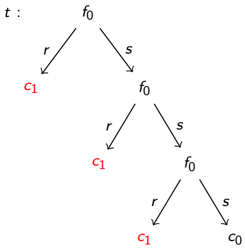
$$\overline{\mathcal{E}}: \quad \{r\} \cup Y_{1,A_1}.\{sr\} =^? \{ssr\} \cup Y_{1,A_1}.\{r\}$$
$$Y_{1,A_1} \mapsto \{\varepsilon, s\} \text{ and } \mathcal{L}_1 = \{r, sr, ssr\}$$

Construction of the automaton - Example

Reverse equation and solution:

$$\overline{\mathcal{E}}: \quad \{r\} \cup Y_{1,A_1}.\{sr\} =^{?} \{ssr\} \cup Y_{1,A_1}.\{r\}$$
$$Y_{1,A_1} \mapsto \{\varepsilon, s\} \text{ and } \mathcal{L}_1 = \{r, sr, ssr\}$$

Let us decorate the tree as follows:



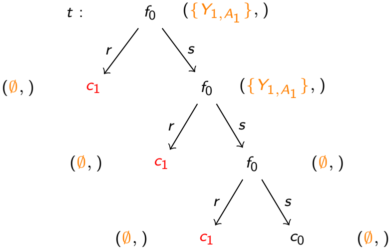
Construction of the automaton - Example

Reverse equation and solution:

$$\overline{\mathcal{E}}: \quad \{r\} \cup Y_{1,A_1}.\{sr\} =^? \{ssr\} \cup Y_{1,A_1}.\{r\}$$

$$Y_{1,A_1} \mapsto \{\varepsilon, s\} \text{ and } \mathcal{L}_1 = \{r, sr, ssr\}$$

Let us decorate the tree as follows:



- variables

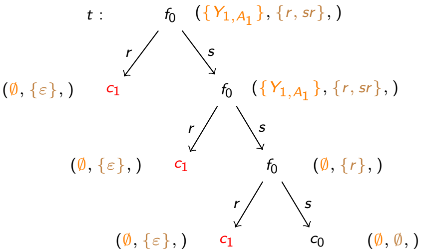
Construction of the automaton - Example

Reverse equation and solution:

$$\overline{\mathcal{E}}: \quad \{r\} \cup Y_{1,A_1}.\{sr\} =^? \{ssr\} \cup Y_{1,A_1}.\{r\}$$

$$Y_{1,A_1} \mapsto \{\varepsilon, s\} \text{ and } \mathcal{L}_1 = \{r, sr, ssr\}$$

Let us decorate the tree as follows:



- variables
- suffixes left-hand side

Construction of the automaton - Example

Reverse equation and solution:

$$\overline{\mathcal{E}}: \quad \{r\} \cup Y_{1,A_1}.\{sr\} =^? \{ssr\} \cup Y_{1,A_1}.\{r\}$$
$$Y_{1,A_1} \mapsto \{\varepsilon, s\} \text{ and } \mathcal{L}_1 = \{r, sr, ssr\}$$

Let us decorate the tree as follows:

$$t: \quad \begin{tikzcd}
 & f_0\ (\{Y_{1,A_1}\},\{r,sr\},\{ssr,r\}) \arrow[dl,"r"'] \arrow[dr,"s"] & & & \\
(\emptyset,\{\varepsilon\},\{\varepsilon\})\ c_1 & & f_0\ (\{Y_{1,A_1}\},\{r,sr\},\{sr,r\}) \arrow[dl,"r"'] \arrow[dr,"s"] & & \\
 & (\emptyset,\{\varepsilon\},\{\varepsilon\})\ c_1 & & f_0\ (\emptyset,\{r\},\{r\}) \arrow[dl,"r"'] \arrow[dr,"s"] & \\
 & & (\emptyset,\{\varepsilon\},\{\varepsilon\})\ c_1 & & c_0\ (\emptyset,\emptyset,\emptyset)
\end{tikzcd}$$

- variables
- suffixes left-hand side
- suffixes right-hand side

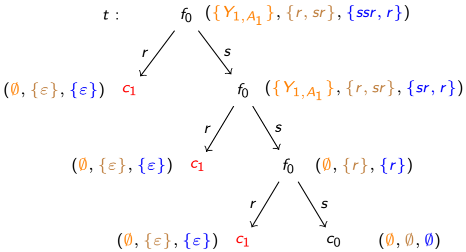
Construction of the automaton - Example

Reverse equation and solution:

$$\overline{\mathcal{E}}: \quad \{r\} \cup Y_{1,A_1}.\{sr\} =^? \{ssr\} \cup Y_{1,A_1}.\{r\}$$

$$Y_{1,A_1} \mapsto \{\varepsilon, s\} \text{ and } \mathcal{L}_1 = \{r, sr, ssr\}$$

Let us decorate the tree as follows:



- variables
- suffixes left-hand side
- suffixes right-hand side

Third and second sets are sets of suffixes occurring in $\overline{\mathcal{E}}$.

The automaton

The automaton

Idea: when traversing a tree, $\mathcal{A}_{\mathcal{E}}$ guesses the words assigned to variables and keeps track of the suffixes that show up.

The automaton

Idea: when traversing a tree, $\mathcal{A}_{\mathcal{E}}$ guesses the words assigned to variables and keeps track of the suffixes that show up.

- Set of states: $Q := 2^{M} \times 2^{SL} \times 2^{SR}$.

The automaton

Idea: when traversing a tree, $\mathcal{A}_{\mathcal{E}}$ guesses the words assigned to variables and keeps track of the suffixes that show up.

- Set of states: $Q := 2^M \times 2^{SL} \times 2^{SR}$.
  - $M = \{1, \ldots, m\}$, where m is the number of variables.

The automaton

Idea: when traversing a tree, $\mathcal{A}_{\mathcal{E}}$ guesses the words assigned to variables and keeps track of the suffixes that show up.

- Set of states: $Q := 2^{M} \times 2^{SL} \times 2^{SR}$.
  - $M = \{1, \ldots, m\}$, where m is the number of variables.
  - SL suffixes occurring in the left hand-side of $\overline{\mathcal{E}}$ (symmetrically for SR)

The automaton

Idea: when traversing a tree, $\mathcal{A}_{\mathcal{E}}$ guesses the words assigned to variables and keeps track of the suffixes that show up.

- Set of states: $Q := 2^{M} \times 2^{SL} \times 2^{SR}$.
  - $M = \{1, \ldots, m\}$, where m is the number of variables.
  - SL suffixes occurring in the left hand-side of $\overline{\mathcal{E}}$ (symmetrically for SR)
- Initial states:

$$I := \{(V, L, R) \mid G \subseteq M, L = \overline{L_i} \cup \bigcup_{j \in V} \overline{L_j^*}, R = \ldots\}$$

The automaton

Idea: when traversing a tree, $\mathcal{A}_{\mathcal{E}}$ guesses the words assigned to variables and keeps track of the suffixes that show up.

- Set of states: $Q := 2^M \times 2^{SL} \times 2^{SR}$.
  - $M = \{1, \ldots, m\}$, where m is the number of variables.
  - SL suffixes occurring in the left hand-side of $\overline{\mathcal{E}}$ (symmetrically for SR)
- Initial states:

$$I := \{(V, L, R) \mid G \subseteq M, L = \overline{L_i} \cup \bigcup_{j \in V} \overline{L_j^*}, R = \ldots\}$$

- Transition function:

$$((G_0, L_0, R_0), f_\ell, (G_1, L_1, R_1), \ldots, (G_k, L_k, R_k)) \in Q \times Q^k \textbf{ iff}$$

The automaton

Idea: when traversing a tree, $\mathcal{A}_{\mathcal{E}}$ guesses the words assigned to variables and keeps track of the suffixes that show up.

- Set of states: $Q := 2^M \times 2^{SL} \times 2^{SR}$.
  - $M = \{1, \ldots, m\}$, where m is the number of variables.
  - SL suffixes occurring in the left hand-side of $\overline{\mathcal{E}}$ (symmetrically for SR)
- Initial states:

$$I := \{(V, L, R) \mid G \subseteq M, L = \overline{L_i} \cup \bigcup_{j \in V} \overline{L_j^*}, R = \ldots\}$$

- Transition function:

$$((G_0, L_0, R_0), f_\ell, (G_1, L_1, R_1), \ldots, (G_k, L_k, R_k)) \in Q \times Q^k \textbf{ iff}$$

  - Suffixes are properly propagated.

The automaton

Idea: when traversing a tree, $\mathcal{A}_{\mathcal{E}}$ guesses the words assigned to variables and keeps track of the suffixes that show up.

- Set of states: $Q := 2^{M} \times 2^{SL} \times 2^{SR}$.
  - $M = \{1, \ldots, m\}$, where m is the number of variables.
  - SL suffixes occurring in the left hand-side of $\overline{\mathcal{E}}$ (symmetrically for SR)
- Initial states:

$$I := \{(V, L, R) \mid G \subseteq M, L = \overline{L_i} \cup \bigcup_{j \in V} \overline{L_j^*}, R = \ldots\}$$

- Transition function:

$$((G_0, L_0, R_0), f_\ell, (G_1, L_1, R_1), \ldots, (G_k, L_k, R_k)) \in Q \times Q^k \textbf{ iff}$$

  - Suffixes are properly propagated.
  - The resulting left and right languages are the same:

$$\varepsilon \in L_0 \text{ iff } \varepsilon \in R_0 \text{ iff } \ell = f_1,$$

The automaton

Idea: when traversing a tree, $\mathcal{A}_{\mathcal{E}}$ guesses the words assigned to variables and keeps track of the suffixes that show up.

- Set of states: $Q := 2^M \times 2^{SL} \times 2^{SR}$.
  - $M = \{1, \dots, m\}$, where m is the number of variables.
  - SL suffixes occurring in the left hand-side of $\overline{\mathcal{E}}$ (symmetrically for SR)
- Initial states:

$$I := \{(V, L, R) \mid G \subseteq M, L = \overline{L_i} \cup \bigcup_{j \in V} \overline{L_j^*}, R = \dots\}$$

- Transition function:

$$((G_0, L_0, R_0), f_\ell, (G_1, L_1, R_1), \dots, (G_k, L_k, R_k)) \in Q \times Q^k \textbf{ iff}$$

  - Suffixes are properly propagated.
  - The resulting left and right languages are the same:

$$\varepsilon \in L_0 \text{ iff } \varepsilon \in R_0 \text{ iff } \ell = f_1,$$

- Final states:

$$F(c_0) = \{(G, L, R) \mid L = R = \emptyset\} \text{ and } F(c_1) = \{(G, L, R) \mid L = R = \{\varepsilon\}\}.$$

Complexity of solving linear equations

Lemma 13

The following are equivalent

1. $\mathcal{L}(A_{\mathcal{E}}) \neq \emptyset$.
2. there are finite sets $Y_{1,i}, \ldots, Y_{1,m}$ such that:

$$\overline{L_i} \cup Y_{1,i}.\overline{L_1^*} \cup \ldots \cup Y_{1,m}.\overline{L_m^*}$$
$$=$$
$$\overline{R_i} \cup Y_{1,i}.\overline{R_1^*} \cup \ldots \cup Y_{1,m}.\overline{R_m^*},$$

Complexity of solving linear equations

Lemma 13

The following are equivalent

1. $\mathcal{L}(A_{\mathcal{E}}) \neq \emptyset$.
2. there are finite sets $Y_{1,i}, \ldots, Y_{1,m}$ such that:

$$\overline{L}_i \cup Y_{1,i}.\overline{L_1^*} \cup \ldots \cup Y_{1,m}.\overline{L_m^*} = \overline{R_i} \cup Y_{1,i}.\overline{R_1^*} \cup \ldots \cup Y_{1,m}.\overline{R_m^*},$$

Algorithm

1. Γ is translated into k linear equations of size polynomial.
2. Build an automaton $\mathcal{A}_{\mathcal{E}}$ for each equation. The size is exponential w.r.t. Γ!
3. Check emptiness of each $\mathcal{A}_{\mathcal{E}}$. In polynomial time in the size of $\mathcal{A}_{\mathcal{E}}$.

Complexity of solving linear equations

Lemma 13

The following are equivalent

1. $\mathcal{L}(A_{\mathcal{E}}) \neq \emptyset$.
2. there are finite sets $Y_{1,i}, \ldots, Y_{1,m}$ such that:

$$\overline{L_i} \cup Y_{1,i}.\overline{L_1^*} \cup \ldots \cup Y_{1,m}.\overline{L_m^*} = \overline{R_i} \cup Y_{1,i}.\overline{R_1^*} \cup \ldots \cup Y_{1,m}.\overline{R_m^*},$$

Algorithm

1. Γ is translated into k linear equations of size polynomial.
2. Build an automaton $\mathcal{A}_{\mathcal{E}}$ for each equation. The size is exponential w.r.t. Γ!
3. Check emptiness of each $\mathcal{A}_{\mathcal{E}}$. In polynomial time in the size of $\mathcal{A}_{\mathcal{E}}$.

Lemma 14 [BN01]

Unification in $\mathcal{FL}_0$, ACUIh and solving linear equations are in ExpTime.

Complexity of solving linear equations - hardness

Complexity of solving linear equations - hardness

A very nice, but complicated reduction. Only very general details:

Complexity of solving linear equations - hardness

A very nice, but complicated reduction. Only very general details:

- Reduction from the intersection emptiness problem for deterministic RTF automata.

Complexity of solving linear equations - hardness

A very nice, but complicated reduction. Only very general details:

- Reduction from the intersection emptiness problem for deterministic RTF automata.

Instance:	A sequence $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of DRTF over a ranked alphabet Σ.
Question:	Is there a tree $t \in \mathcal{L}(\mathcal{A}_1) \cap \ldots \cap \mathcal{L}(\mathcal{A}_n)$?

Complexity of solving linear equations - hardness

A very nice, but complicated reduction. Only very general details:

- Reduction from the intersection emptiness problem for deterministic RTF automata.

Instance:	A sequence $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of DRTF over a ranked alphabet Σ.
Question:	Is there a tree $t \in \mathcal{L}(\mathcal{A}_1) \cap \ldots \cap \mathcal{L}(\mathcal{A}_n)$?

The problem is ExpTime-complete [Sei94].

Complexity of solving linear equations - hardness

A very nice, but complicated reduction. Only very general details:

- Reduction from the intersection emptiness problem for deterministic RTF automata.

Instance:	A sequence $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of DRTF over a ranked alphabet Σ.
Question:	Is there a tree $t \in \mathcal{L}(\mathcal{A}_1) \cap \ldots \cap \mathcal{L}(\mathcal{A}_n)$?

The problem is ExpTime-complete [Sei94].

- Given $\mathcal{A}_1, \ldots, \mathcal{A}_n$, construct a system of n linear equations $\{\mathcal{E}_1, \ldots, \mathcal{E}_n\}$ such that:

$$\mathcal{L}(\mathcal{A}_1) \cap \ldots \cap \mathcal{L}(\mathcal{A}_n) \neq \emptyset$$

iff

$\{\mathcal{E}_1, \ldots, \mathcal{E}_n\}$ has a solution.

Complexity of solving linear equations - hardness

A very nice, but complicated reduction. Only very general details:

- Reduction from the intersection emptiness problem for deterministic RTF automata.

Instance:	A sequence $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of DRTF over a ranked alphabet Σ.
Question:	Is there a tree $t \in \mathcal{L}(\mathcal{A}_1) \cap \ldots \cap \mathcal{L}(\mathcal{A}_n)$?

The problem is ExpTime-complete [Sei94].

- Given $\mathcal{A}_1, \ldots, \mathcal{A}_n$, construct a system of n linear equations $\{\mathcal{E}_1, \ldots, \mathcal{E}_n\}$ such that:

$$\mathcal{L}(\mathcal{A}_1) \cap \ldots \cap \mathcal{L}(\mathcal{A}_n) \neq \emptyset$$

iff

$\{\mathcal{E}_1, \ldots, \mathcal{E}_n\}$ has a solution.

Theorem 15 [BN01]

Unification in $\mathcal{FL}_0$, ACUIh and solving linear equations are ExpTime-complete.

Unification in $\mathcal{FL}_{reg}$

Unification in $\mathcal{FL}_{reg}$

$\mathcal{FL}_{reg}$ extends $\mathcal{FL}_0$ with complex roles:

$$\forall\varepsilon,\ \forall\emptyset,\ \forall(R \cup S),\ \forall(R \circ S) \text{ and } \forall R^*.$$

Unification in $\mathcal{FL}_{reg}$

$\mathcal{FL}_{reg}$ extends $\mathcal{FL}_0$ with complex roles:

$$\forall\varepsilon,\ \forall\emptyset,\ \forall(R \cup S),\ \forall(R \circ S) \text{ and } \forall R^*.$$

Semantics of complex roles:

$$\begin{aligned}
\varepsilon^{\mathcal{I}} &:= \{(d,d) \mid d \in \Delta^{\mathcal{I}}\}\\
\emptyset^{\mathcal{I}} &:= \emptyset\\
(R \cup S)^{\mathcal{I}} &:= R^{\mathcal{I}} \cup S^{\mathcal{I}}\\
(R \circ S)^{\mathcal{I}} &:= \{(d,e) \mid \exists f : (d,f) \in R^{\mathcal{I}} \wedge (f,e) \in R^{\mathcal{I}}\}\\
(R^*)^{\mathcal{I}} &:= \bigcup_{n \geq 0} (R^{\mathcal{I}})^n.
\end{aligned}$$

Unification in $\mathcal{FL}_{reg}$

$\mathcal{FL}_{reg}$ extends $\mathcal{FL}_0$ with complex roles:

$$\forall\varepsilon,\ \forall\emptyset,\ \forall(R \cup S),\ \forall(R \circ S) \text{ and } \forall R^*.$$

Semantics of complex roles:

$$\begin{aligned}
\varepsilon^{\mathcal{I}} &:= \{(d,d) \mid d \in \Delta^{\mathcal{I}}\}\\
\emptyset^{\mathcal{I}} &:= \emptyset\\
(R \cup S)^{\mathcal{I}} &:= R^{\mathcal{I}} \cup S^{\mathcal{I}}\\
(R \circ S)^{\mathcal{I}} &:= \{(d,e) \mid \exists f : (d,f) \in R^{\mathcal{I}} \wedge (f,e) \in R^{\mathcal{I}}\}\\
(R^*)^{\mathcal{I}} &:= \bigcup_{n \geq 0} (R^{\mathcal{I}})^n.
\end{aligned}$$

Example

$\{r\} \circ (\{s\} \cup \{r\}) \circ \{s\}^* \rightarrow$ pairs (d,e) such that d reaches e through a word in $r.(s|r).s^*$.

Unification in $\mathcal{FL}_{reg}$

$\mathcal{FL}_{reg}$ extends $\mathcal{FL}_0$ with complex roles:

$$\forall\varepsilon,\ \forall\emptyset,\ \forall(R \cup S),\ \forall(R \circ S) \text{ and } \forall R^*.$$

Semantics of complex roles:

$$\begin{aligned}
\varepsilon^{\mathcal{I}} &:= \{(d,d) \mid d \in \Delta^{\mathcal{I}}\} \\
\emptyset^{\mathcal{I}} &:= \emptyset \\
(R \cup S)^{\mathcal{I}} &:= R^{\mathcal{I}} \cup S^{\mathcal{I}} \\
(R \circ S)^{\mathcal{I}} &:= \{(d,e) \mid \exists f : (d,f) \in R^{\mathcal{I}} \wedge (f,e) \in R^{\mathcal{I}}\} \\
(R^*)^{\mathcal{I}} &:= \bigcup_{n \geq 0} (R^{\mathcal{I}})^n.
\end{aligned}$$

Example

$\{r\} \circ (\{s\} \cup \{r\}) \circ \{s\}^* \rightarrow$ pairs (d,e) such that d reaches e through a word in $r.(s|r).s^*$.

A complex role can be seen as a regular expression/language!

Unification in $\mathcal{FL}_{reg}$ - Complexity

Unification in $\mathcal{FL}_{reg}$ - Complexity

- in ExpTime: similar as for $\mathcal{FL}_0$, but

Unification in $\mathcal{FL}_{reg}$ - Complexity

- in ExpTime: similar as for $\mathcal{FL}_0$, but
  - We need to deal with linear equations that can have infinite languages as coefficients.

Unification in $\mathcal{FL}_{reg}$ - Complexity

- in ExpTime: similar as for $\mathcal{FL}_0$, but
  - We need to deal with linear equations that can have infinite languages as coefficients.
  - Use automata on infinite trees: looping tree automata.

Unification in $\mathcal{FL}_{reg}$ - Complexity

- in ExpTime: similar as for $\mathcal{FL}_0$, but
  - We need to deal with linear equations that can have infinite languages as coefficients.
  - Use automata on infinite trees: looping tree automata.
- Exptime-hard?

Unification in $\mathcal{FL}_{reg}$ - Complexity

- in ExpTime: similar as for $\mathcal{FL}_0$, but
  - We need to deal with linear equations that can have infinite languages as coefficients.
  - Use automata on infinite trees: looping tree automata.
- Exptime-hard?
  - Exptime-hardness from $\mathcal{FL}_0$ cannot be directly inherited!

Unification in $\mathcal{FL}_{reg}$ - Complexity

- in ExpTime: similar as for $\mathcal{FL}_0$, but
  - We need to deal with linear equations that can have infinite languages as coefficients.
  - Use automata on infinite trees: looping tree automata.

- Exptime-hard?
  - Exptime-hardness from $\mathcal{FL}_0$ cannot be directly inherited!
  - Adapt the same reduction, but using a sequence of looping tree automata.

Unification in $\mathcal{FL}_{reg}$ - Complexity

- in ExpTime: similar as for $\mathcal{FL}_0$, but
  - We need to deal with linear equations that can have infinite languages as coefficients.
  - Use automata on infinite trees: looping tree automata.
- Exptime-hard?
  - Exptime-hardness from $\mathcal{FL}_0$ cannot be directly inherited!
  - Adapt the same reduction, but using a sequence of looping tree automata.

Theorem 16 [BK01]

Unification in $\mathcal{FL}_{reg}$ is ExpTime-complete.

Matching with respect to general $\mathcal{FL}_0$ TBoxes

Deciding matching w.r.t. the empty TBox

Deciding matching w.r.t. the empty TBox

Matching is a unification problem of the form:

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^?$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k.$$

Deciding matching w.r.t. the empty TBox

Matching is a unification problem of the form:

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^?$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k.$$

Example

matching problem: $\forall\{r, s\}.A \sqcap \forall\{s\}.B \equiv^? \forall\{rr\}.A \sqcap \forall\{r, s\}.X_1 \sqcap \forall\{s\}.X_2$

Deciding matching w.r.t. the empty TBox

Matching is a unification problem of the form:

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^?$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k.$$

Example

matching problem: $\forall\{r,s\}.A \sqcap \forall\{s\}.B \equiv^? \forall\{rr\}.A \sqcap \forall\{r,s\}.X_1 \sqcap \forall\{s\}.X_2$
equation for A: $\{r,s\} =^? \{rr\} \cup \{r,s\}.X_1 \cup \{s\}.X_2$

Deciding matching w.r.t. the empty TBox

Matching is a unification problem of the form:

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^?$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k.$$

Example

matching problem: $\forall\{r,s\}.A \sqcap \forall\{s\}.B \equiv^? \forall\{rr\}.A \sqcap \forall\{r,s\}.X_1 \sqcap \forall\{s\}.X_2$

equation for A: $\{r,s\} =^? \{rr\} \cup \{r,s\}.X_1 \cup \{s\}.X_2$

has no solution: a) $rr \notin \{r,s\}$ b) $s \in X_1$ yields a suffix $ss \notin \{r,s\}$

Deciding matching w.r.t. the empty TBox

Matching is a unification problem of the form:

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^?$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k.$$

Example

matching problem: $\forall\{r,s\}.A \sqcap \forall\{s\}.B \equiv^? \forall\{rr\}.A \sqcap \forall\{r,s\}.X_1 \sqcap \forall\{s\}.X_2$

equation for A: $\{r,s\} =^? \{rr\} \cup \{r,s\}.X_1 \cup \{s\}.X_2$

has no solution: a) $rr \notin \{r,s\}$ b) $s \in X_1$ yields a suffix $ss \notin \{r,s\}$

How to decide matching?

Deciding matching w.r.t. the empty TBox

Matching is a unification problem of the form:

$$\forall L_1.A_1 \sqcap \ldots \sqcap \forall L_k.A_k \sqcap \forall L_1^*.X_1 \sqcap \ldots \sqcap \forall L_m^*.X_m$$
$$\equiv^?$$
$$\forall R_1.A_1 \sqcap \ldots \sqcap \forall R_k.A_k.$$

Example

matching problem: $\forall\{r,s\}.A \sqcap \forall\{s\}.B \equiv^? \forall\{rr\}.A \sqcap \forall\{r,s\}.X_1 \sqcap \forall\{s\}.X_2$

equation for A: $\{r,s\} =^? \{rr\} \cup \{r,s\}.X_1 \cup \{s\}.X_2$

has no solution: a) $rr \notin \{r,s\}$ b) $s \in X_1$ yields a suffix $ss \notin \{r,s\}$

How to decide matching?

Lemma 17 [BN01]

An $\mathcal{FL}_0$-matching problem has a solution iff the following is a solution:

$$\sigma(X_i) = \bigcap_{u \in L_i^*} u^{-1}.R_0, \text{ where } u^{-1}.R_0 := \{v \mid uv \in R_0\}.$$

Decidable in PTime.

Characterization of subsumption in $\mathcal{FL}_0$ - TBox

In the presence of a non-empty TBox, a new characterization is needed:

$$\mathcal{L}_{\mathcal{T}}(C) := \{(w, A) \in \mathsf{N_R}^* \times \mathsf{N_C} \mid C \sqsubseteq_{\mathcal{T}} \forall w.A\}$$

Characterization of subsumption in $\mathcal{FL}_0$ - TBox

In the presence of a non-empty TBox, a new characterization is needed:

$$\mathcal{L}_{\mathcal{T}}(C) := \{(w, A) \in \mathsf{N_R}^* \times \mathsf{N_C} \mid C \sqsubseteq_{\mathcal{T}} \forall w.A\}$$

Characterization of subsumption w.r.t. a TBox

Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox and C, D $\mathcal{FL}_0$ concepts. Then, $C \sqsubseteq_{\mathcal{T}} D$ iff $\mathcal{L}_{\mathcal{T}}(D) \subseteq \mathcal{L}_{\mathcal{T}}(C)$.

Characterization of subsumption in $\mathcal{FL}_0$ - TBox

In the presence of a non-empty TBox, a new characterization is needed:

$$\mathcal{L}_\mathcal{T}(C) := \{(w, A) \in \mathsf{N_R}^* \times \mathsf{N_C} \mid C \sqsubseteq_\mathcal{T} \forall w.A\}$$

Characterization of subsumption w.r.t. a TBox

Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox and C, D $\mathcal{FL}_0$ concepts. Then, $C \sqsubseteq_\mathcal{T} D$ iff $\mathcal{L}_\mathcal{T}(D) \subseteq \mathcal{L}_\mathcal{T}(C)$.

Back to the example

$$\begin{array}{c} C := \forall\{r, s\}.A \sqcap \forall\{s\}.B \\ \equiv^? \\ D := \forall\{rr\}.A \sqcap \forall\{r, s\}.X_1 \sqcap \forall\{s\}.X_2 \end{array} \qquad \mathcal{T} = \{A \sqsubseteq \forall r.A, \forall s.B \sqsubseteq A\}$$

Characterization of subsumption in $\mathcal{FL}_0$ - TBox

In the presence of a non-empty TBox, a new characterization is needed:

$$\mathcal{L}_\mathcal{T}(C) := \{(w, A) \in {\mathsf{N_R}}^* \times \mathsf{N_C} \mid C \sqsubseteq_\mathcal{T} \forall w.A\}$$

Characterization of subsumption w.r.t. a TBox

Let $\mathcal{T}$ be an $\mathcal{FL}_0$ TBox and C, D $\mathcal{FL}_0$ concepts. Then, $C \sqsubseteq_\mathcal{T} D$ iff $\mathcal{L}_\mathcal{T}(D) \subseteq \mathcal{L}_\mathcal{T}(C)$.

Back to the example

$$C := \forall\{r, s\}.A \sqcap \forall\{s\}.B$$
$$\equiv^?$$
$$D := \forall\{rr\}.A \sqcap \forall\{r, s\}.X_1 \sqcap \forall\{s\}.X_2$$

$$\mathcal{T} = \{A \sqsubseteq \forall r.A, \forall s.B \sqsubseteq A\}$$

solution: $\sigma(X_1) := A$ and $\sigma(X_2) := B$

$$\mathcal{L}_\mathcal{T}(C, A) = \{s\} \cup r^* = \mathcal{L}_\mathcal{T}(\sigma(D), A)$$

Matching w.r.t. general TBoxes

Matching w.r.t. general TBoxes

Matching problem (after applying normalization)

Matching w.r.t. general TBoxes

Matching problem (after applying normalization)

$$C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_n.X_n.$$

Matching w.r.t. general TBoxes

Matching problem (after applying normalization)

$$C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_n.X_n.$$

Can we extend the idea for the empty TBox to the sets $\mathcal{L}_{\mathcal{T}}(C, A)$?

Matching w.r.t. general TBoxes

Matching problem (after applying normalization)

$$C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_n.X_n.$$

Can we extend the idea for the empty TBox to the sets $\mathcal{L}_{\mathcal{T}}(C, A)$?

- for each variable X_i and concept name A_j:

$$\widehat{L}_{i,j} := \bigcap_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j).$$

Matching w.r.t. general TBoxes

Matching problem (after applying normalization)

$$C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_n.X_n.$$

Can we extend the idea for the empty TBox to the sets $\mathcal{L}_{\mathcal{T}}(C, A)$?

- for each variable X_i and concept name A_j:

$$\widehat{L}_{i,j} := \bigcap_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j).$$

- define $\sigma(X_i)$ as follows?

$$\sigma(X_i) := \widehat{L}_{i,1}.A_1 \sqcap \ldots \sqcap \widehat{L}_{i,k}.A_k.$$

Matching w.r.t. general TBoxes

Matching problem (after applying normalization)

$$C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_n.X_n.$$

Can we extend the idea for the empty TBox to the sets $\mathcal{L}_{\mathcal{T}}(C, A)$?

- for each variable X_i and concept name A_j:

$$\widehat{L}_{i,j} := \bigcap_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j).$$

- define $\sigma(X_i)$ as follows?

$$\sigma(X_i) := \widehat{L}_{i,1}.A_1 \sqcap \ldots \sqcap \widehat{L}_{i,k}.A_k.$$

$\widehat{L}_{i,j}$ can be infinite $\rightarrow \sigma(X_i)$ is not an $\mathcal{FL}_0$ concept!

Matching w.r.t. general TBoxes

Matching problem (after applying normalization)

$$C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_n.X_n.$$

Can we extend the idea for the empty TBox to the sets $\mathcal{L}_{\mathcal{T}}(C, A)$?

- for each variable X_i and concept name A_j:

$$\widehat{L}_{i,j} := \bigcap_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j).$$

- define $\sigma(X_i)$ as follows?

$$\sigma(X_i) := \widehat{L}_{i,1}.A_1 \sqcap \ldots \sqcap \widehat{L}_{i,k}.A_k.$$

$\widehat{L}_{i,j}$ can be infinite $\rightarrow \sigma(X_i)$ is not an $\mathcal{FL}_0$ concept!

Workaround: consider $\mathcal{FL}_{reg}$ concept descriptions.

Deciding the existence of an $\mathcal{FL}_{reg}$-matcher

Deciding the existence of an $\mathcal{FL}_{reg}$-matcher

What do we know about the sets $\widehat{L}_{i,j} := \bigcap_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j)$

Deciding the existence of an $\mathcal{FL}_{reg}$-matcher

What do we know about the sets $\widehat{L}_{i,j} := \bigcap_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j)$

- They can be infinite, but they are always regular languages.

Deciding the existence of an $\mathcal{FL}_{reg}$-matcher

What do we know about the sets $\widehat{L}_{i,j} := \bigcap_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j)$

- They can be infinite, but they are always regular languages.
- Then, $\widehat{\sigma}(X_i) := \widehat{L}_{i,1}.A_1 \sqcap \ldots \sqcap \widehat{L}_{i,k}.A_k$ is an $\mathcal{FL}_{reg}$ concept.

Deciding the existence of an $\mathcal{FL}_{reg}$-matcher

What do we know about the sets $\widehat{L}_{i,j} := \bigcap_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j)$

- They can be infinite, but they are always regular languages.
- Then, $\widehat{\sigma}(X_i) := \widehat{L}_{i,1}.A_1 \sqcap \ldots \sqcap \widehat{L}_{i,k}.A_k$ is an $\mathcal{FL}_{reg}$ concept.

The following is true

$C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_n.X_n$ has an $\mathcal{FL}_{reg}$ matcher

iff

$\widehat{\sigma}(X_i) := \widehat{L}_{i,1}.A_1 \sqcap \ldots \sqcap \widehat{L}_{i,k}.A_k$ is a matcher.

Deciding the existence of an $\mathcal{FL}_{reg}$-matcher

What do we know about the sets $\widehat{L}_{i,j} := \bigcap_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j)$

- They can be infinite, but they are always regular languages.
- Then, $\widehat{\sigma}(X_i) := \widehat{L}_{i,1}.A_1 \sqcap \ldots \sqcap \widehat{L}_{i,k}.A_k$ is an $\mathcal{FL}_{reg}$ concept.

The following is true

$C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_n.X_n$ has an $\mathcal{FL}_{reg}$ matcher

iff

$\widehat{\sigma}(X_i) := \widehat{L}_{i,1}.A_1 \sqcap \ldots \sqcap \widehat{L}_{i,k}.A_k$ is a matcher.

Checking that $\widehat{\sigma}$ is a matcher

Deciding the existence of an $\mathcal{FL}_{reg}$-matcher

What do we know about the sets $\widehat{L}_{i,j} := \bigcap_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j)$

- They can be infinite, but they are always regular languages.
- Then, $\widehat{\sigma}(X_i) := \widehat{L}_{i,1}.A_1 \sqcap \ldots \sqcap \widehat{L}_{i,k}.A_k$ is an $\mathcal{FL}_{reg}$ concept.

The following is true

$C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_n.X_n$ has an $\mathcal{FL}_{reg}$ matcher

iff

$\widehat{\sigma}(X_i) := \widehat{L}_{i,1}.A_1 \sqcap \ldots \sqcap \widehat{L}_{i,k}.A_k$ is a matcher.

Checking that $\widehat{\sigma}$ is a matcher

1. $C \sqsubseteq_{\mathcal{T}} E$
2. $E \sqcap \forall L_1.\widehat{\sigma}(X_1) \sqcap \ldots \sqcap \forall L_n.\widehat{\sigma}(X_n) \sqsubseteq_{\mathcal{T}} C$.

Deciding the existence of an $\mathcal{FL}_{reg}$-matcher

What do we know about the sets $\widehat{L}_{i,j} := \bigcap\limits_{u \in L_i} u^{-1}\mathcal{L}_{\mathcal{T}}(C, A_j)$

- They can be infinite, but they are always regular languages.
- Then, $\widehat{\sigma}(X_i) := \widehat{L}_{i,1}.A_1 \sqcap \ldots \sqcap \widehat{L}_{i,k}.A_k$ is an $\mathcal{FL}_{reg}$ concept.

The following is true

$$C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_n.X_n \text{ has an } \mathcal{FL}_{reg} \text{ matcher}$$

iff

$$\widehat{\sigma}(X_i) := \widehat{L}_{i,1}.A_1 \sqcap \ldots \sqcap \widehat{L}_{i,k}.A_k \text{ is a matcher.}$$

Checking that $\widehat{\sigma}$ is a matcher

1. $C \sqsubseteq_{\mathcal{T}} E$ 2. $E \sqcap \forall L_1.\widehat{\sigma}(X_1) \sqcap \ldots \sqcap \forall L_n.\widehat{\sigma}(X_n) \sqsubseteq_{\mathcal{T}} C$.

This can be done in exponential time using automata on infinite trees [BGM18].

Deciding the existence of an $\mathcal{FL}_0$-matcher

By construction of $\widehat{L}_{i,j}$

$$C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_n.X_n \text{ has an } \mathcal{FL}_0 \text{ matcher}$$

iff

1. $C \sqsubseteq E$
2. There are finite languages $L_{i,j} \subseteq \widehat{L}_{i,j}$ such that:

$$E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_n.\sigma(X_n) \sqsubseteq_{\mathcal{T}} C$$

Deciding the existence of an $\mathcal{FL}_0$-matcher

By construction of $\widehat{L}_{i,j}$

$$C \equiv^?_\mathcal{T} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_n.X_n \text{ has an } \mathcal{FL}_0 \text{ matcher}$$

iff

1. $C \sqsubseteq E$
2. There are finite languages $L_{i,j} \subseteq \widehat{L}_{i,j}$ such that:

$$E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_n.\sigma(X_n) \sqsubseteq_\mathcal{T} C$$

Applying the Compactness Theorem

$$E \sqcap \forall L_1.\widehat{\sigma}(X_1) \sqcap \ldots \sqcap \forall L_n.\widehat{\sigma}(X_n) \sqsubseteq_\mathcal{T} C$$

iff

there are finite languages $L_{i,j} \subseteq \widehat{L}_{i,j}$ s.t. $E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_n.\sigma(X_n) \sqsubseteq_\mathcal{T} C$.

Deciding the existence of an $\mathcal{FL}_0$-matcher

By construction of $\widehat{L}_{i,j}$

$$C \equiv^?_{\mathcal{T}} E \sqcap \forall L_1.X_1 \sqcap \ldots \sqcap \forall L_n.X_n \text{ has an } \mathcal{FL}_0 \text{ matcher}$$

iff

1. $C \sqsubseteq E$
2. There are finite languages $L_{i,j} \subseteq \widehat{L}_{i,j}$ such that:

$$E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_n.\sigma(X_n) \sqsubseteq_{\mathcal{T}} C$$

Applying the Compactness Theorem

$$E \sqcap \forall L_1.\widehat{\sigma}(X_1) \sqcap \ldots \sqcap \forall L_n.\widehat{\sigma}(X_n) \sqsubseteq_{\mathcal{T}} C$$

iff

there are finite languages $L_{i,j} \subseteq \widehat{L}_{i,j}$ s.t. $E \sqcap \forall L_1.\sigma(X_1) \sqcap \ldots \sqcap \forall L_n.\sigma(X_n) \sqsubseteq_{\mathcal{T}} C$.

Consequences

- There exists an $\mathcal{FL}_0$-matcher **iff** there is an $\mathcal{FL}_{reg}$-matcher.
- Deciding the existing of an $\mathcal{FL}_0$-matcher is ExpTime-complete.

Summary - Unification in the DL $\mathcal{FL}_0$

Summary - Unification in the DL $\mathcal{FL}_0$

- The problem is ExpTime-complete w.r.t. the empty TBox.

Summary - Unification in the DL $\mathcal{FL}_0$

- The problem is ExpTime-complete w.r.t. the empty TBox.

- This result carries over to $\mathcal{FL}_{reg}$.

Summary - Unification in the DL $\mathcal{FL}_0$

- The problem is ExpTime-complete w.r.t. the empty TBox.

- This result carries over to $\mathcal{FL}_{reg}$.

- Unification in $\mathcal{FL}_0$ corresponds to unification in the equational theory ACUIh and to solving linear equations over finite languages.

Summary - Unification in the DL $\mathcal{FL}_0$

- The problem is ExpTime-complete w.r.t. the empty TBox.

- This result carries over to $\mathcal{FL}_{reg}$.

- Unification in $\mathcal{FL}_0$ corresponds to unification in the equational theory ACUIh and to solving linear equations over finite languages.

- In the presence of a general TBox.
  - Decidability is an open problem.
  - It is only known that it is ExpTime-complete for the special case of matching (non-constructive proof).

References I

Franz Baader, Oliver Fernandez Gil, and Pavlos Marantidis.
Matching in the description logic FL0 with respect to general tboxes.
In *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Awassa, Ethiopia, 16-21 November 2018*, volume 57 of *EPiC Series in Computing*, pages 76–94. EasyChair, 2018.

Franz Baader and Ralf Küsters.
Unification in a description logic with transitive closure of roles.
In *Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conference, LPAR 2001, Havana, Cuba, December 3-7, 2001, Proceedings*, volume 2250 of *Lecture Notes in Computer Science*, pages 217–232. Springer, 2001.

Franz Baader and Paliath Narendran.
Unification of Concept Terms in Description Logics.
J. Symb. Comput., 31(3):277–305, 2001.

Helmut Seidl.
Haskell overloading is dexptime-complete.
Inf. Process. Lett., 52(2):57–60, 1994.