Parsing with the Logic FC

Owen M. Bell Sam M. Thompson Dominik D. Freydenberger

Loughborough University, UK

LangSec 2025

Background: A Brief History of FC

Logic and Databases

- SQL is 'syntactic sugar' for First Order Logic (FO).
- FC is to querying strings what FO is to querying relational structures.

FC vs Other String Logics

- Treats strings as strings, not sequences of positions.
- Can test the equality of two strings.
- Makes writing queries "user friendly".

Original Motivation

Information Extraction.

Work on FC

- Studied in database theory.
- 'Lenses' for efficient evaluation (using techniques from database theory).
- Various tractability criteria.
- Various extensions to increase expressivity.

FC for LangSec

A framework for declarative input-handling.

What Is FC?

FC

- The **F**inite model version of the theory of **C**oncatenation.
- Finite model semantics: our "universe" is all substrings of an input string $\mathfrak{s}$.

Defining FC

- String Equations $x \doteq \alpha$
  - x: variable,
  - α: string of constants and variables.
- Combine with: $\wedge$, $\vee$, $\neg$, $\exists$, $\forall$
  - and, or, not, exists, for all.

Example

$\exists x \colon (x \doteq \texttt{SPW} \vee x \doteq \texttt{LangSec})$.

The Model Checking Problem

- Does a formula hold for an input string?
- In context of parsing: recognition.

The Finite Model is Crucial

- Infinite: Model checking is undecidable.
- Finite: Model checking is decidable.

Defining Formal Languages in FC

Language of a formula

All strings for which the formula is true.

Example 1

$\varphi_1 \coloneqq \neg\exists x\colon x \doteq \mathtt{bab}.$

- $\mathcal{L}(\varphi_1)$: all strings that do not contain bab.

Example 2

$\varphi_2 \coloneqq \exists x, y\colon x \doteq yy.$

- $\mathcal{L}(\varphi_2)$: all stings that contain a square

Example 3

$\varphi_3 \coloneqq \exists x\colon \mathfrak{s} \doteq \mathtt{a}x\mathtt{a}x \wedge \forall y, z\colon \neg(x \doteq y\mathtt{a}z).$

- $\mathcal{L}(\varphi_3) \coloneqq \mathtt{ab}^n\mathtt{ab}^n$ (for $\Sigma = \{\mathtt{a}, \mathtt{b}\}$).

Returning a Query Result Instead

By having free (unbound) variables.

Modifying Example 2

$\varphi_{2'}(x) \coloneqq \exists y\colon x \doteq yy.$

- Return all substrings that are squares.

Lenses for Model Checking

Theorem

Model Checking for FC is PSPACE-complete.

Top-Down

- Recursive algorithm.
- Every quantifier is a loop.
- Until we reach atomic formulas.

Top-Down Lens

- Quantifier Rank (QR)
  - Nesting depth of quantifiers.
- Bounded QR: PTIME model checking.

Bottom-Up

- Starting from atomic formulas.
- Build a relation for each subformula.

Bottom-Up Lens

- Formula Width
  - Max number of free variables in a subformula.
- Bounded treewidth $\rightarrow$ Bounded width.
- Bounded width: PTIME model checking.

Another Fragment and Lens

Conjunctive Queries

- A central topic of database theory.
- Use only $\wedge$ and $\exists$.

FC-CQ

Conjunctive Queries in FC.

Theorem

Model checking for FC-CQ is NP-complete.

Another Lens

- Acyclicity
  - Does the FC-CQ have a join tree (a tree representation where each variable appears only in one connected subtree)?
- Acyclic FC-CQ: PTIME model checking.

Extending Expressive Power: Constraints

FC on its own may not have enough expressive power in certain use cases.

Constraints

- Concise ways of increasing power.
- As we treat string as strings, we can easily add arbitrary constraints.
- Additional atomic formulas $x \mathrel{\dot{\in}} R$ for a language representation R.

Complexity

As long as the constraints can be evaluated efficiently, adding constraints does not affect any complexity results.

Regular Constraints

- $x \mathrel{\dot{\in}} \gamma$
  - x: variable,
  - γ: regular expression.
- x represents a member of $\mathcal{L}(\gamma)$.

FC[REG]

- Strictly more expressive than FC.
- Captures generalized core spanners, a popular information extraction framework.

More on Constraints

Length Constraints

- $len(x, y)$.
- The images of x and y have the same length.

Constrained Quantifiers

- $\exists x \dot{\in} R$ and $\forall x \dot{\in} R$
  - x: variable,
  - R: relation.
- "Engineering" optimization.
- If the constraint holds for only a small number of substrings, then we can evaluate the rest of the formula on only these.
- A mechanism for 'filtering out' substrings.

Even More Expressive Power: Recursion

FC-Datalog

- Extends FC-CQ with recursion.
- FC analog of Datalog (a relational query language).

Motivation

Implementing Context Free Grammars for NLP.

Parse Trees

- We can define FC-Datalog using proof trees.
- And therefore obtain parse trees for FC-Datalog programs.

FC-Datalog program Q

$$\begin{aligned} \mathsf{Ans}() &\leftarrow \mathfrak{s} \doteq yz,\ \ E(y,z); \\ E(x,y) &\leftarrow x \doteq \varepsilon,\ \ y \doteq \varepsilon; \\ E(x,y) &\leftarrow x \doteq \mathtt{a}u,\ \ y \doteq \mathtt{b}v,\ \ E(u,v). \end{aligned}$$

$\mathcal{L}(Q) := \{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbb{N}\}$.

Complexity

- Captures PTIME.
- Model checking: EXPTIME-complete.

Efficient Recursion - Fragments of FC-Datalog

Deterministic One Letter Lookahead+ (DOLLA+)

- Captures LOGSPACE.

Strictly Decreasing (SD)

- Model checking: linear time.

DOLLA+ FC-Datalog as Generalized Automata

- Relation: State,
- Rule: Transition,
- String variable: Head,
- Heads read words instead of letters,
- Nonregular string computations in transitions.

Benefits

- Not bound by left-to-right parsing.
- Can deterministically express context-free languages languages not accepted by a deterministic PDA.

SD FC-Datalog Program Q'

$$\begin{aligned} \mathsf{Ans}() &\leftarrow R(\mathfrak{s}); \\ R(x) &\leftarrow x \doteq \varepsilon; \\ R(x) &\leftarrow x \doteq \mathsf{a}; \quad \text{for all } \mathsf{a} \in \Sigma, \\ R(x) &\leftarrow x \doteq \mathsf{a}y\mathsf{a}, \quad \text{for all } \mathsf{a} \in \Sigma. \end{aligned}$$

$\mathcal{L}(Q')$ is the palindrome language.

Simulating a Class of Regex

Regex

- Regular expressions with back-references.
- $\langle x\colon \gamma\rangle$ saves the string matched by γ in the memory x.
- x recalls the saved string.

Deterministic Regex (DRX)

- Regex whose extended Glushkov automaton are deterministic.
- Can define nonregular languages.

Example Deterministic Regex

$\gamma := \langle x\colon (\mathtt{a} \vee \mathtt{b})^{+}\rangle \cdot \mathtt{d} \cdot x.$

- Matches all words $u\mathtt{d}u$ where $u \in \{\mathtt{a}, \mathtt{b}\}^{+}$.

Example Noneterministic Regex

$\gamma' := \langle x\colon (\mathtt{a} \vee \mathtt{b})^{+}\rangle \cdot x.$

- Matches all words uu where $u \in \{\mathtt{a}, \mathtt{b}\}^{+}$.

Simulating DRX in FC-Datalog

We can simulate deterministic regex in SD FC-Datalog (linear time model checking).

FC and LangSec

LangSec Core Principles

1. *Valid input defined as formal language.*
2. *Full recognition before processing.*
3. *Principle of least expressiveness.*
4. *Principle of parser equivalence.*

FC and These Principles

1. FC (and its extensions) define/accept formal languages.
2. FC has optimizations for model checking: deciding input validity (recognition).
3. The natural restrictions and extensions to FC give us a framework with an expressivity hierarchy.
4. More difficult. Equivalence is undecidable, even for for FC-CQ.

Area for Future Work

Fragments with decidable equivalence.

A Declarative Model for LangSec

FC vs Computational Machines

- Simpler writing process.
- Can mitigate against unintended consequences arising when defining a specific implementation.
- Provide clearer error messaging by presenting unsatisfied subformulas to users.
- More concise. The size blow up from an FC formula to an equivalent regular expression is not bounded by any recursive function.
- Easier to visually compare in cases where complex data formats are required (so equivalence cannot be checked).

Example Grammar G

- $\mathcal{L}(G)$: All strings that do not contain $\mathtt{bab}$.
- For binary alphabet $\{\mathtt{a}, \mathtt{b}\}$:

$$
\begin{array}{ll}
S \to \mathtt{a}S; & A \to \mathtt{a}B; \\
S \to \mathtt{b}A; & A \to \mathtt{b}A; \\
S \to \varepsilon; & B \to \mathtt{a}S; \\
A \to \varepsilon; & B \to \varepsilon.
\end{array}
$$

- As the alphabet grows, so does the grammar.

Equivalent FC formua

$\neg\exists x : x \doteq \mathtt{bab}.$

FC vs Regex

Regex Issues

- Difficult to write and interpret.
- Portability.
- Security (ReDoS - Regular expression Denial of Service).

FC as a Replacement for Regex

- Declarative formulas easier to write and interpret.
- Explicit reuse and compositionality of FC formulas makes portability much simpler.
- Compositionality of components mitigates against ReDoS issues.

A Framework for Combining Parsers

FC Compositionality

- Every formula defines a relation.
- These relations can be used in other formulas.

We can also use relations from other sources.

Some Example External Relation Sources

- Parsers from other models.
- An existing relational database.
- a by-hand implementation (for a relation that is efficient to verify).

A Unifiying Framework

- For combining multiple parsers into a single text verifier.
- Modular components - much more natural in FC than for computational models.

Weak Equivalence

- Need only to decide equivalence of our subrelations.
- Further aligns FC with the LangSec principle of parser equivalence.

Conclusions and Next Steps

Conclusions

- FC: a declarative logic for recognition.
- Efficient model checking using lenses (that tell us which algorithm to use).
- Natural fragments and extensions with desirable properties.
- Aligns with LangSec core principles.
- A framework for combining parsers.

Next Steps: Three Main Directions

- Investigating for further fragments of FC and its extensions where static analysis problems such as equivalence become tractable.
- Optimizing evaluation algorithms - "engineering".
- Building a fully-fledged implementation for FC and its extensions, including such an optimizer.

From both theoretical and practical perspectives.

Thank You!

Any Questions?