Parsing: Episode I

Matthew Might
University of Utah
matt.might.net
ucombinator.org

Administrivia

- Project 1: Use the source!

Agenda

- What is parsing?
- Context-free languages
- Context-free grammars
- Recursive descent parsing
- Properties of grammars

What is parsing?

A **parser** converts a token stream
from the lexer into a parse tree.

Example

```
f x = x
```

Example

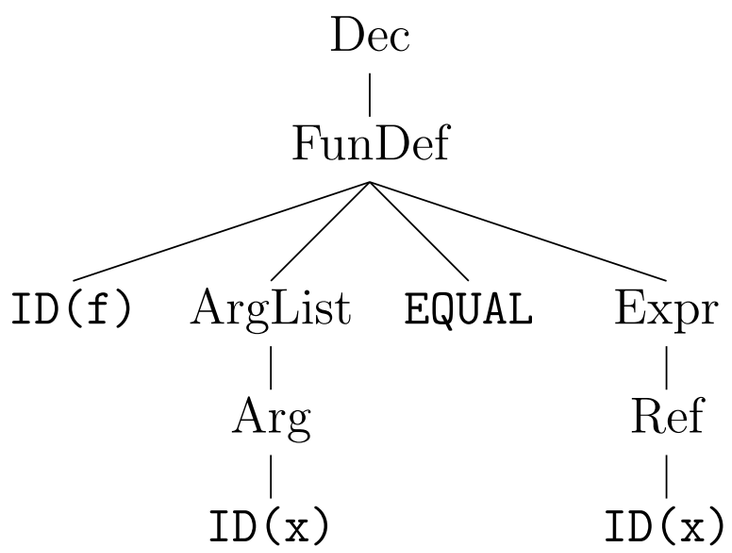
```
f x = x
```

```
ID(f) ID(x) EQUAL ID(x)
```

Example

```
f x = x
```

```
ID(f) ID(x) EQUAL ID(x)
```



Parsing methods

- LALR(*k*)
- LR(*k*)
- SLR(*k*)
- LL(*k*)
- Back-tracking search
- Nondet. rec. descent
- Predictive rec. descent
- PEG/Packrat
- Combinators
- Earley

Context-free languages

Context-free languages

- Natural choice for describing syntax
- Like regular expressions plus recursion

Example

- Language of balanced parentheses
- Language is context-free language
- But language is not regular language

As formal language

- Context-free languages are formal languages
- Two operations allowed: catenation, union
- Recursive equations are allowed as well

Example

$$L_B = \{\epsilon\} \cup (\{(\} \cdot L_B \cdot \{)\} \cdot L_B) \, .$$

Problem: Recursion!

How do we assign meaning to recursive definitions?

Fixed points!

Fixed points

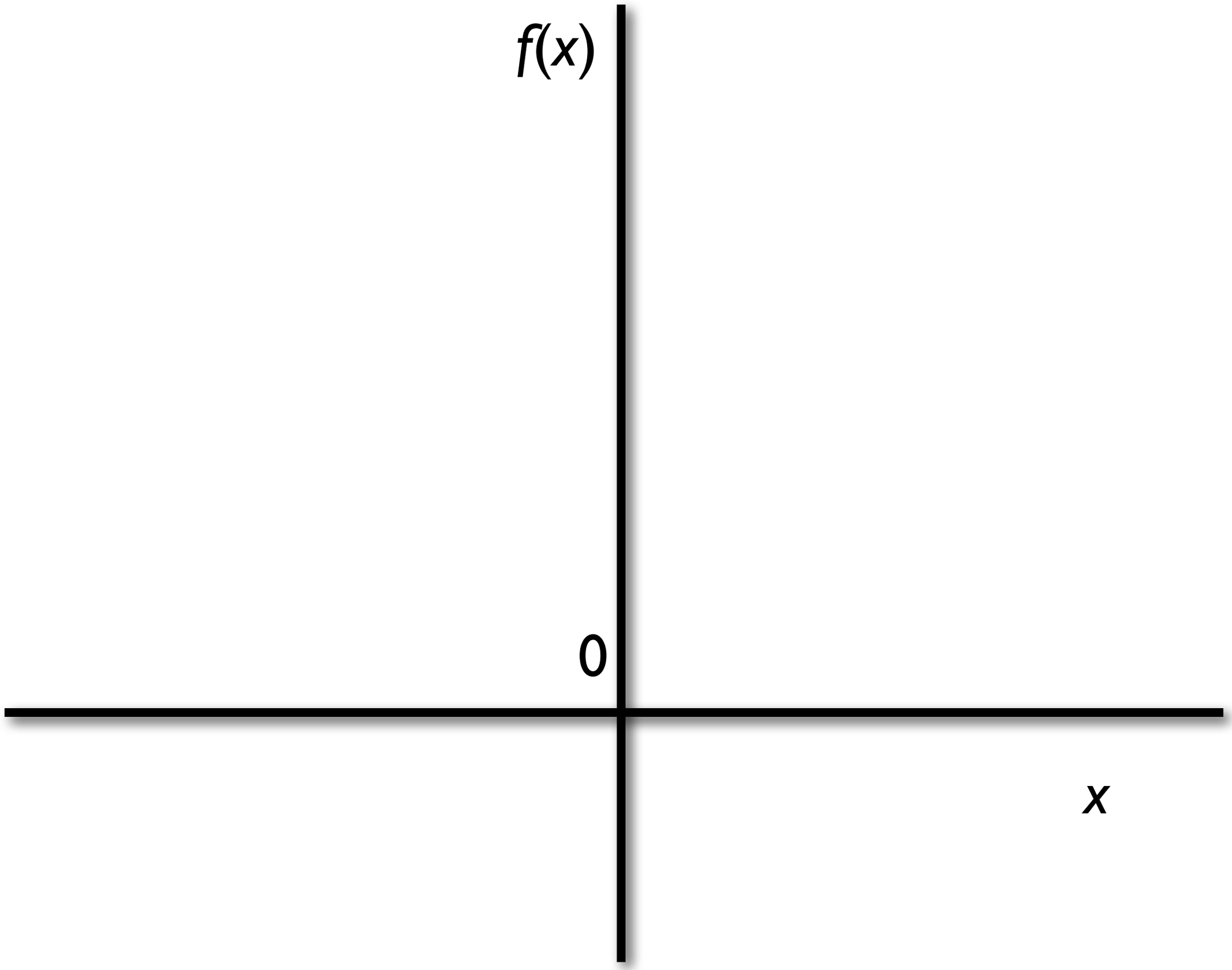
If $x = f(x)$, then the point x is
a **fixed point** of the function f.

Fixed points

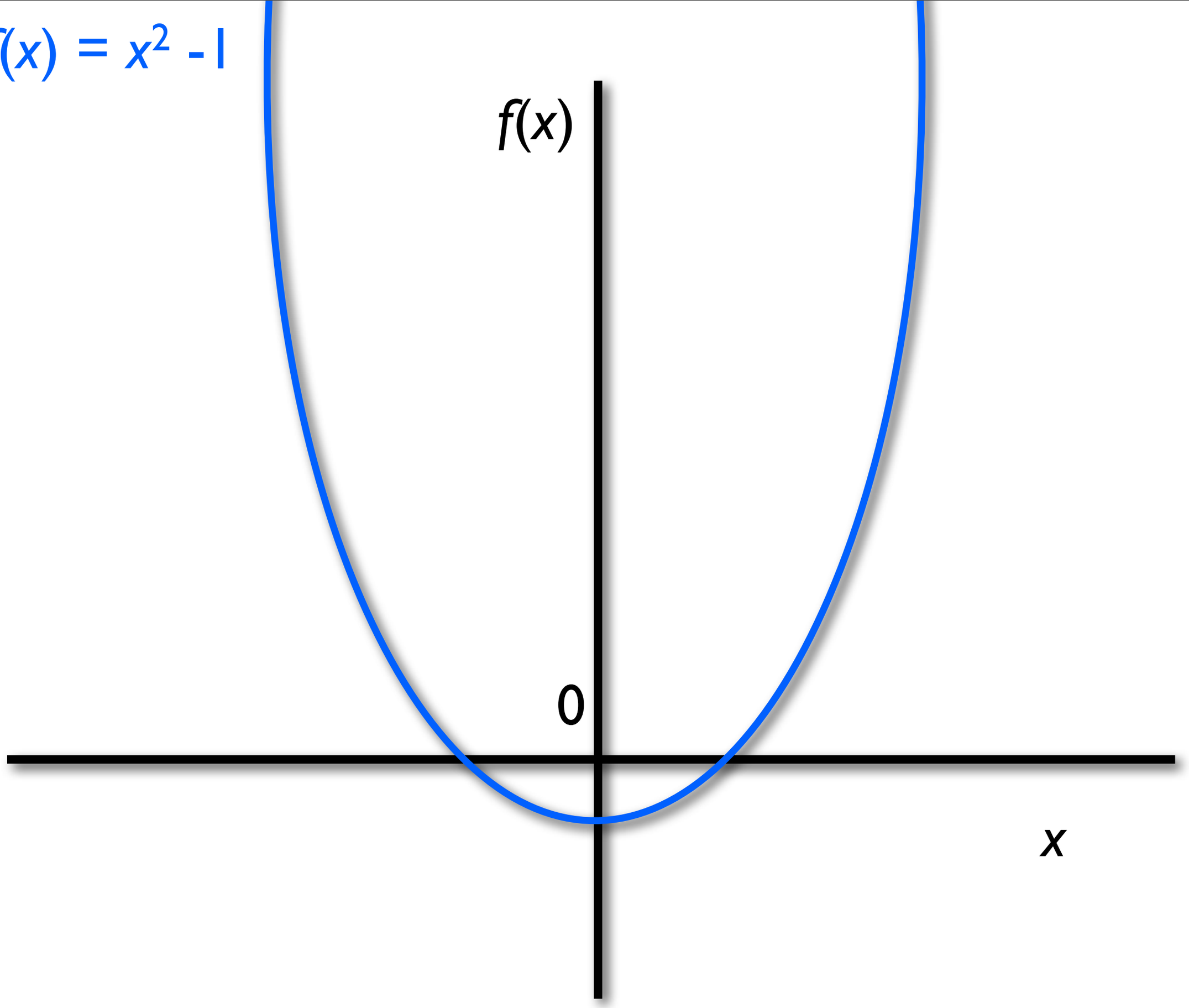
$$Fix(f) = \{L : L = f(L)\}\,.$$

Algebra

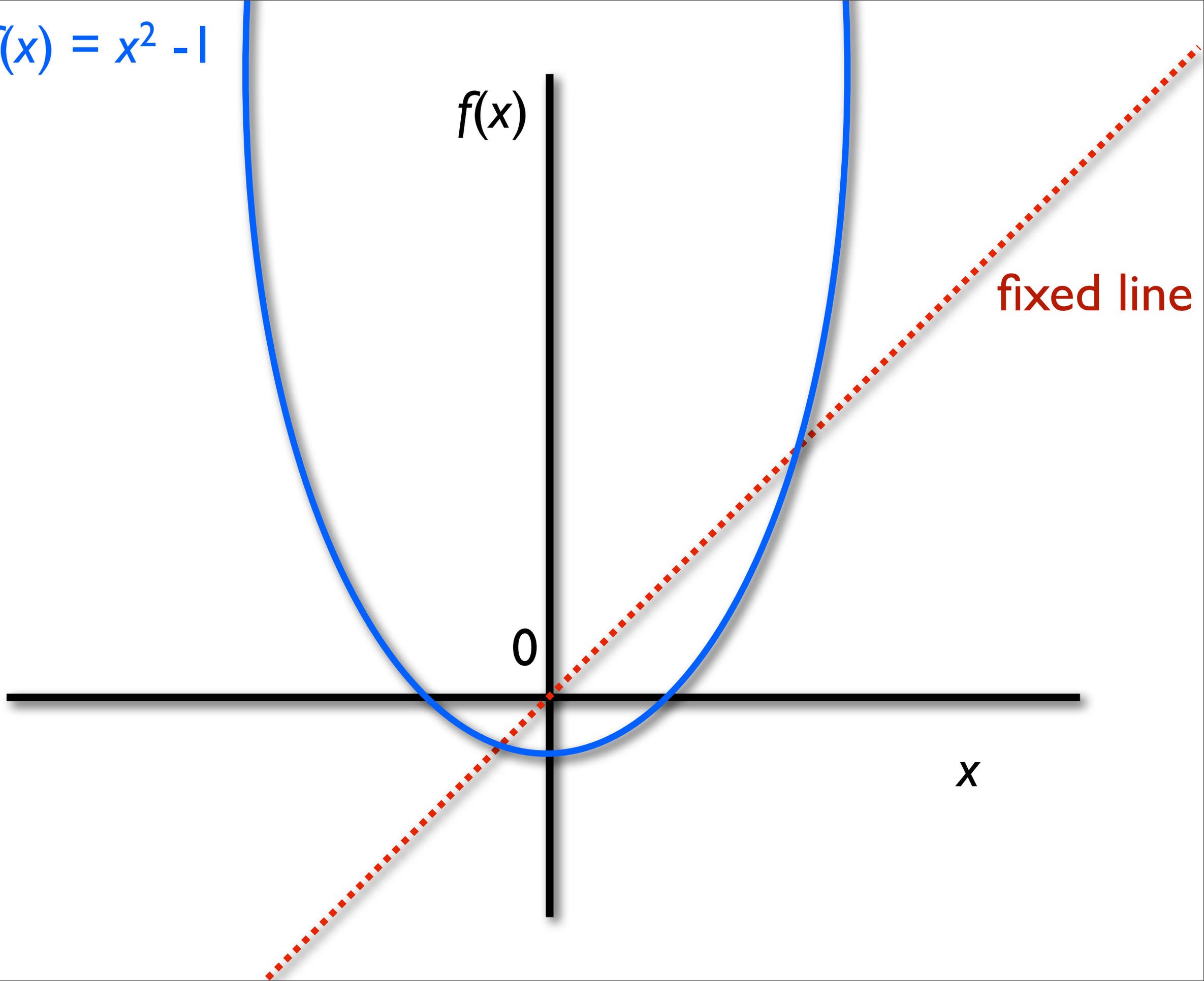
- $x = x^2 - 1$ is a recursive definition of x
- If $f(v) = v^2 - 1$, then $x = f(x)$.
- Solutions are the fixed points of f.


f(x)
0
x

$f(x) = x^2 - 1$



$f(x) = x^2 - 1$



Refactoring

$$L_B = f(L_B)$$

$$f(L) = \{\epsilon\} \cup (\{\,(\,\} \cdot L \cdot \{\,)\,\} \cdot L)$$

Candidates

$$L_B \in Fix(f)$$

Sensible choices

$$\mathrm{lfp}(f) = \bigcap_{L \in Fix(f)} L$$

$$\mathrm{gfp}(f) = \bigcup_{L \in Fix(f)} L$$

Greatest fixed point

- Includes infinitely long strings!
- Example: ()()()()()()()()...

Kleene's theorem (specialized)

If a function f is *continuous*, then:

$$\mathrm{lfp}(f) = \bigcup_{n \geq 1}^{\infty} f^n(\emptyset)$$

Continuous

The function f is **continuous** only if:

$$f\left(\bigcup_i x_i\right) = \bigcup_i f(x_i)$$

Constructive observation

$$\emptyset \subseteq f(\emptyset) \subseteq f^2(\emptyset) \subseteq f^3(\emptyset) \subseteq \cdots$$

Excursion

In general

In general, for a set of recursive equations over the languages $L_1, \ldots, L_n$, if

$$\begin{aligned} L_1 &= f_1(L_1, \ldots, L_n) \\ L_2 &= f_2(L_1, \ldots, L_n) \\ \vdots &= \qquad\qquad \vdots \\ L_n &= f_n(L_1, \ldots, L_n), \end{aligned}$$

then these languages are a fixed point of the function $F : \mathcal{P}\left(A^*\right)^n \to \mathcal{P}\left(A^*\right)^n$:

$$\begin{aligned} F(L_1, \ldots, L_n) = (&f_1(L_1, \ldots, L_n), \\ &f_2(L_1, \ldots, L_n), \\ &\vdots \\ &f_n(L_1, \ldots, L_n)), \end{aligned}$$

and by default, the least fixed point of this function:

$$(L_1, \ldots, L_n) = \mathrm{lfp}(F).$$

Context-free grammars

Context-free grammars

A **context-free grammar** is a quadruple (A, N, R, n_0), where:

- the set A contains the terminal symbols of the language—its alphabet; and
- the set N contains the non-terminal symbols of the language; and
- the set $R \subseteq N \times (A \times N)^*$ contains non-terminal-to-terminal substitution rules; and
- the symbol $n_0 \in N$ is the top-level "start" symbol.

Example

$$A = \{(,)\}$$

$$N = \{B\}$$

$$R \ni B \to (B)\,B$$

$$R \ni B \to \epsilon$$

$$n_0 = B.$$

Recognizing strings

$$\frac{(n \to s_1 \ldots s_n) \in R \qquad wnw' \in \mathcal{L}(A, N, R, n_0)}{ws_1 \ldots s_n w' \in \mathcal{L}(A, N, R, n_0).}$$

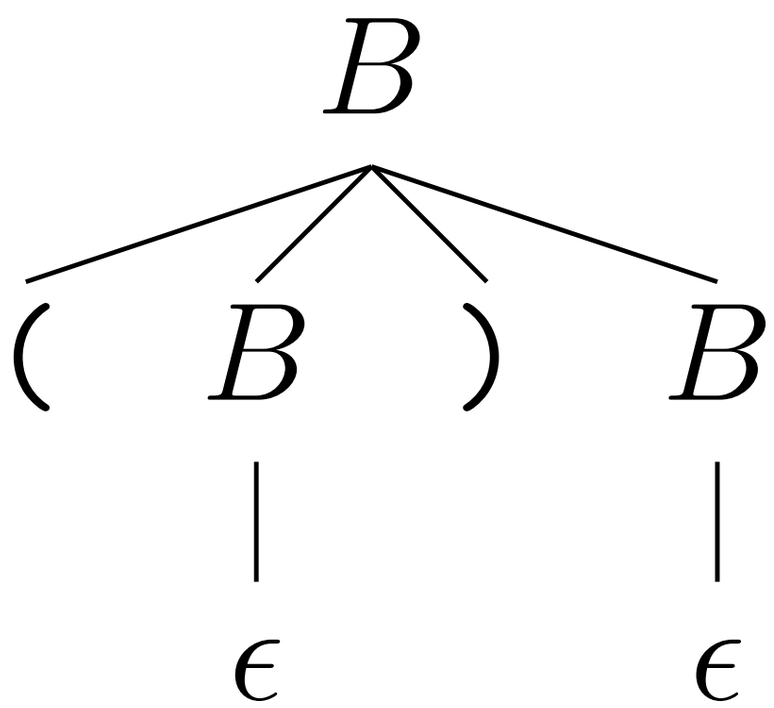
Example

$$\dfrac{\dfrac{(B \to \epsilon) \in R \qquad \dfrac{(B \to (B)\,B) \in R \qquad \dfrac{B = n_0}{B \in \mathcal{L}(G_B)}}{(B)\,B \in \mathcal{L}(G_B)}}{() \in \mathcal{L}(G_B).}}$$

Parse trees

- Convenient diagrammatic notation
- Demonstrates membership in language
- Simultaneously shows structure of string

Example



Example: Regexes

$$
\begin{aligned}
A &= \{\texttt{(}, \texttt{)}, \texttt{a}, \ldots, \texttt{z}, \texttt{|}, \texttt{*}\} \\
N &= \{E, T, F, K\} \\
R &\ni E \to T \mid E \\
R &\ni E \to T \\
R &\ni T \to F\ T \\
R &\ni T \to F \\
R &\ni F \to K\texttt{*} \\
R &\ni F \to K \\
R &\ni K \to \texttt{(}E\texttt{)} \\
R &\ni K \to a, \text{ for every } a \in \{\texttt{a}, \ldots, \texttt{z}\} \\
n_0 &= E.
\end{aligned}
$$

Parse tree: (a|b)*

$$A = \{(,), \mathtt{a}, \ldots, \mathtt{z}, |, *\}$$

$$N = \{E, T, F, K\}$$

$$R \ni E \to T \mid E$$

$$R \ni E \to T$$

$$R \ni T \to F\ T$$

$$R \ni T \to F$$

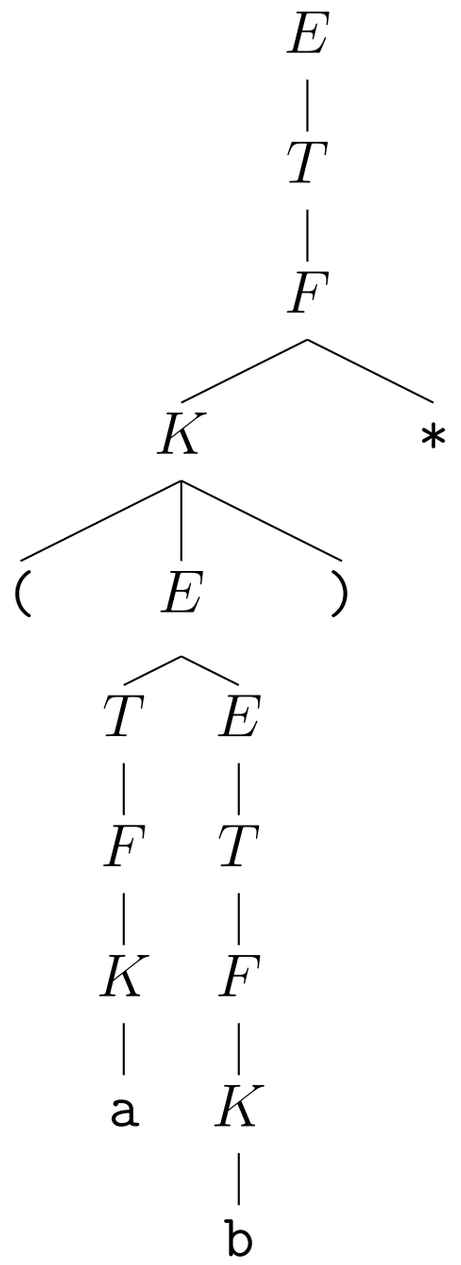
$$R \ni F \to K*$$

$$R \ni F \to K$$

$$R \ni K \to (E)$$

$$R \ni K \to a, \text{ for every } a \in \{\mathtt{a}, \ldots, \mathtt{z}\}$$

$$n_0 = E.$$



Ambiguous grammars

A grammar is **ambiguous**
if there is at least one string that
has one or more parse trees.

Example: Ambiguity

$$A = \{(,), +, *\} \cup \mathbb{Z}$$

$$N = \{E\}$$

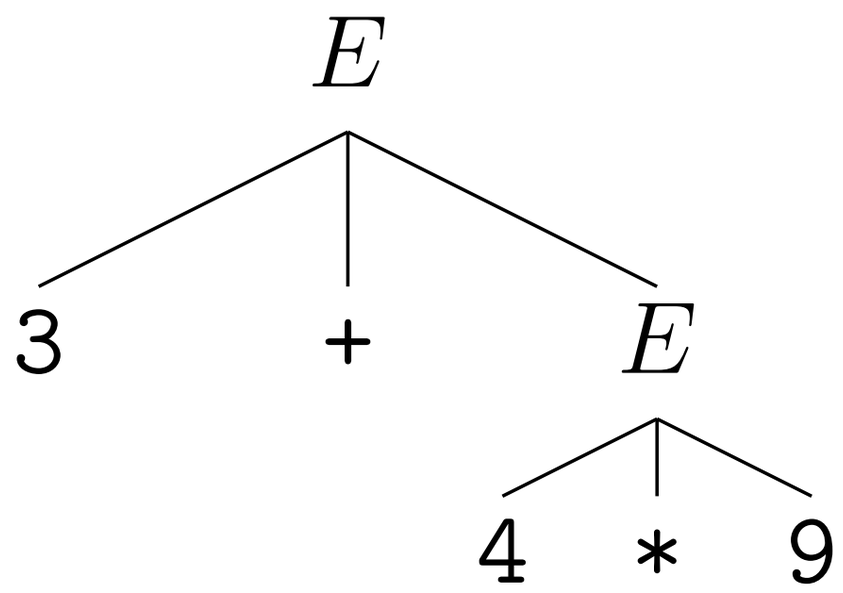
$$R \ni E \to E + E$$

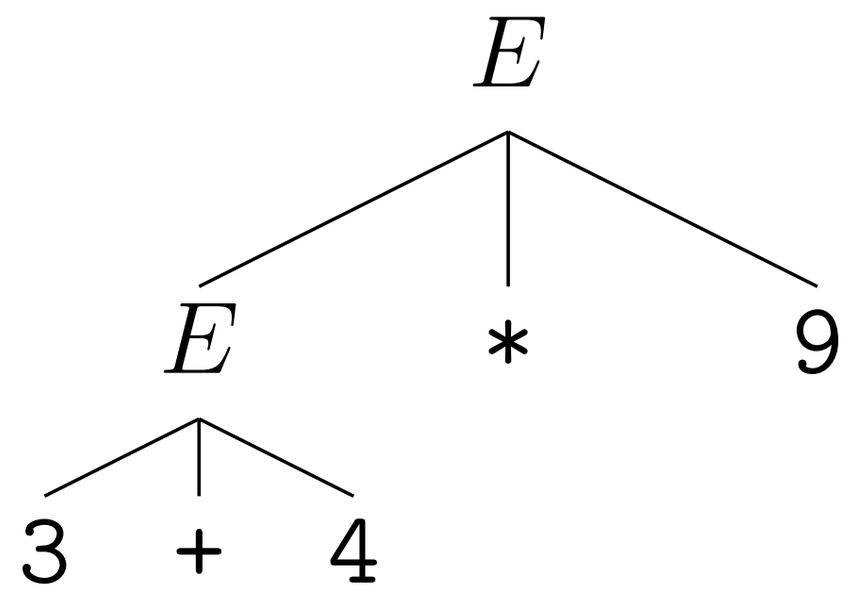
$$R \ni E \to E * E$$

$$R \ni E \to z, \text{ for every } z \in \mathbb{Z}$$

$$n_0 = E.$$

Example: 3 + 4 * 9





Left-recursion

A grammar is **left-recursive** if a non-terminal symbol can derive a new string with itself in leftmost position.

Example: Left-recursion

$$S \longrightarrow S \ \mathbf{,} \ x$$

$$S \longrightarrow x$$

Example: Factoring

$$S \longrightarrow x \ , \ S$$

$$S \longrightarrow x$$

Exercise:
Nondeterministic recursive descent

Grammar

$$X \rightarrow (X^*)$$

$$X \rightarrow num$$

$$X \rightarrow sym$$

$$X^* \rightarrow X\ X^*$$

$$X^* \rightarrow \epsilon.$$

Exercise: Predictive recursive descent

Lexer API

- `next() : Token`
- `eat(t : TokenType)`
- `peek(k : Int) : TokenType`

CFG properties

Nullability

The **nullability function**, $\delta : (A \cup N) \rightarrow \{\{\epsilon\}, \emptyset\}$, returns the set $\{\epsilon\}$ if the provided symbol can derive the empty string, and $\emptyset$ otherwise:

$$\delta(a) =$$
$$\delta(n) \supseteq$$
$$\delta(n) \supseteq$$

Nullability

The **nullability function**, $\delta : (A \cup N) \to \{\{\epsilon\}, \emptyset\}$, returns the set $\{\epsilon\}$ if the provided symbol can derive the empty string, and $\emptyset$ otherwise:

$$\delta(a) = \emptyset$$
$$\delta(n) \supseteq \delta(s_1) \cdot \ldots \cdot \delta(s_n) \text{ if } (n \to s_1 \ldots s_n) \in R$$
$$\delta(n) \supseteq \{\epsilon\} \text{ if } (n \to \epsilon) \in R.$$

Inclusion constraints

$$X_1 \supseteq f_1(X_1, \ldots, X_n)$$

$$\vdots \qquad\qquad \vdots$$

$$X_n \supseteq f_n(X_1, \ldots, X_n)$$

Inclusion constraints

$$X_1 \supseteq f_1(X_1, \ldots, X_n)$$

$$\vdots \qquad\qquad \vdots$$

$$X_n \supseteq f_n(X_1, \ldots, X_n)$$

Solving inclusions

$X_i \leftarrow \emptyset$ for all i

$changed \leftarrow \mathbf{true}$

while $(changed)$

$changed \leftarrow \mathbf{false}$

$X_i' \leftarrow f_i(X_1, \ldots, X_n)$

if $(X_i \neq X_i')$

$X_i \leftarrow X_i'$

$changed \leftarrow \mathbf{true}$.

First sets

In context-free grammars, first sets are easily computed with subset-inclusion constraints; for every rule $(n \longrightarrow s_1 \ldots s_m) \in R$:

$$\mathit{first}(n) \supseteq$$

First sets

In context-free grammars, first sets are easily computed with subset-inclusion constraints; for every rule $(n \longrightarrow s_1 \ldots s_m) \in R$:

$$\mathit{first}(n) \supseteq \bigcup_{i \geq 1}^{m} \delta(s_1 \ldots s_{i-1}) \cdot \mathit{first}(s_i).$$

Follow sets

$follow : (A \cup N) \rightarrow A$; for every rule $n \rightarrow s_1 \ldots s_n$

$$follow(s_i) \supseteq$$

$$\cup$$

Follow sets

$$follow : (A \cup N) \to A\text{; for every rule } n \to s_1 \ldots s_n$$

$$follow(s_i) \supseteq \bigcup_{j \geq i}^{n-1} \delta(s_{i+1} \ldots s_j) \cdot first(s_{j+1})$$

$$\cup \ \delta(s_{i+1} \ldots s_n) \cdot follow(n).$$

CFL trivia

- Are regular languages context-free?
- Are CFLs closed under complement?
- Is the intersection of CFLs context-free?
- Does a CFG accept no strings?
- Does a CFG accept a finite set?
- Does a CFG accept every string?
- Is one CFL a subset of another CFL?