Artificial Languages, Part 1 (Syntax)

Dave Dubin

March 27, 2017

Key concepts for language topics

1. Syntax governs the *form* expressions of a language may take.

Key concepts for language topics

1. Syntax governs the *form* expressions of a language may take.
2. Semantics governs the *relationship* between language expressions and the *domain* we're modeling.

Key concepts for language topics

1. Syntax governs the *form* expressions of a language may take.
2. Semantics governs the *relationship* between language expressions and the *domain* we're modeling.
3. We have the usual formalist agenda of reducing our accounts of language to simple mathematical objects.

Key concepts for language topics

1. Syntax governs the *form* expressions of a language may take.
2. Semantics governs the *relationship* between language expressions and the *domain* we're modeling.
3. We have the usual formalist agenda of reducing our accounts of language to simple mathematical objects.
4. That agenda is partly in the service of understanding, but also our aim to process language expressions using software.

Key concepts for language topics

1. Syntax governs the *form* expressions of a language may take.
2. Semantics governs the *relationship* between language expressions and the *domain* we're modeling.
3. We have the usual formalist agenda of reducing our accounts of language to simple mathematical objects.
4. That agenda is partly in the service of understanding, but also our aim to process language expressions using software.
5. The cost of processing language is measured in processing time and memory.

Key concepts for language topics

1. Syntax governs the *form* expressions of a language may take.
2. Semantics governs the *relationship* between language expressions and the *domain* we're modeling.
3. We have the usual formalist agenda of reducing our accounts of language to simple mathematical objects.
4. That agenda is partly in the service of understanding, but also our aim to process language expressions using software.
5. The cost of processing language is measured in processing time and memory.
6. The more expressive our language, the more expensive it will be to run our software over it.

Key concepts for language topics

1. Syntax governs the *form* expressions of a language may take.
2. Semantics governs the *relationship* between language expressions and the *domain* we're modeling.
3. We have the usual formalist agenda of reducing our accounts of language to simple mathematical objects.
4. That agenda is partly in the service of understanding, but also our aim to process language expressions using software.
5. The cost of processing language is measured in processing time and memory.
6. The more expressive our language, the more expensive it will be to run our software over it.
7. Classifications of languages (like the Chomsky hierarchy) help us recognize what kind of software we need to accomplish our processing goals.

Grammars for artificial languages

- We've seen grammars in earlier presentations.

Grammars for artificial languages

- We've seen grammars in earlier presentations.
- Propositional logic:

Grammars for artificial languages

- We've seen grammars in earlier presentations.
- Propositional logic:
    - Let P be a set of proposition letters and let $p \in P$.

Grammars for artificial languages

- We've seen grammars in earlier presentations.
- Propositional logic:
  - Let P be a set of proposition letters and let $p \in P$.
  - $\varphi ::= p | \neg\varphi | (\varphi \wedge \varphi) | (\varphi \vee \varphi) | (\varphi \rightarrow \varphi) | (\varphi \leftrightarrow \varphi)$

Grammars for artificial languages

- We've seen grammars in earlier presentations.
- Propositional logic:
  - Let P be a set of proposition letters and let $p \in P$.
  - $\varphi ::= p | \neg\varphi | (\varphi \wedge \varphi) | (\varphi \vee \varphi) | (\varphi \rightarrow \varphi) | (\varphi \leftrightarrow \varphi)$
- Predicate logic:

Grammars for artificial languages

- We've seen grammars in earlier presentations.
- Propositional logic:
  - Let P be a set of proposition letters and let $p \in P$.
  - $\varphi ::= p|\neg\varphi|(\varphi \wedge \varphi)|(\varphi \vee \varphi)|(\varphi \rightarrow \varphi)|(\varphi \leftrightarrow \varphi)$
- Predicate logic:
  - $\mathbf{v} ::= x|y|z|\ldots$

Grammars for artificial languages

- We've seen grammars in earlier presentations.
- Propositional logic:
    - Let P be a set of proposition letters and let $p \in P$.
    - $\varphi ::= p | \neg\varphi | (\varphi \wedge \varphi) | (\varphi \vee \varphi) | (\varphi \rightarrow \varphi) | (\varphi \leftrightarrow \varphi)$
- Predicate logic:
    - $\mathbf{v} ::= x | y | z | \ldots$
    - $\mathbf{c} ::= a | b | c | \ldots$

Grammars for artificial languages

- We've seen grammars in earlier presentations.
- Propositional logic:
  - Let P be a set of proposition letters and let $p \in P$.
  - $\varphi ::= p|\neg\varphi|(\varphi \wedge \varphi)|(\varphi \vee \varphi)|(\varphi \rightarrow \varphi)|(\varphi \leftrightarrow \varphi)$
- Predicate logic:
  - $\mathbf{v} ::= x|y|z|\ldots$
  - $\mathbf{c} ::= a|b|c|\ldots$
  - $\mathbf{t} ::= \mathbf{v}|\mathbf{c}$

Grammars for artificial languages

- We've seen grammars in earlier presentations.
- Propositional logic:
  - Let P be a set of proposition letters and let $p \in P$.
  - $\varphi ::= p|\neg\varphi|(\varphi \wedge \varphi)|(\varphi \vee \varphi)|(\varphi \rightarrow \varphi)|(\varphi \leftrightarrow \varphi)$
- Predicate logic:
  - $\mathbf{v} ::= x|y|z|\ldots$
  - $\mathbf{c} ::= a|b|c|\ldots$
  - $\mathbf{t} ::= \mathbf{v}|\mathbf{c}$
  - $\mathbf{P} ::= P|Q|R|\ldots$

Grammars for artificial languages

- We've seen grammars in earlier presentations.
- Propositional logic:
  - Let P be a set of proposition letters and let $p \in P$.
  - $\varphi ::= p|\neg\varphi|(\varphi \wedge \varphi)|(\varphi \vee \varphi)|(\varphi \rightarrow \varphi)|(\varphi \leftrightarrow \varphi)$
- Predicate logic:
  - $\mathbf{v} ::= x|y|z|\ldots$
  - $\mathbf{c} ::= a|b|c|\ldots$
  - $\mathbf{t} ::= \mathbf{v}|\mathbf{c}$
  - $\mathbf{P} ::= P|Q|R|\ldots$
  - $\mathbf{Atom} ::= \mathbf{Pt}_1 \ldots \mathbf{t}_n$ where n is the arity of $\mathbf{P}$

Grammars for artificial languages

- We've seen grammars in earlier presentations.
- Propositional logic:
  - Let P be a set of proposition letters and let $p \in P$.
  - $\varphi ::= p|\neg\varphi|(\varphi \wedge \varphi)|(\varphi \vee \varphi)|(\varphi \rightarrow \varphi)|(\varphi \leftrightarrow \varphi)$
- Predicate logic:
  - $\mathbf{v} ::= x|y|z|\ldots$
  - $\mathbf{c} ::= a|b|c|\ldots$
  - $\mathbf{t} ::= \mathbf{v}|\mathbf{c}$
  - $\mathbf{P} ::= P|Q|R|\ldots$
  - $\mathbf{Atom} ::= \mathbf{Pt}_1 \ldots \mathbf{t}_n$ where n is the arity of $\mathbf{P}$
  - $\varphi ::= \mathbf{Atom}|\neg\varphi|(\varphi \wedge \varphi)|(\varphi \vee \varphi)|(\varphi \rightarrow \varphi)|(\varphi \leftrightarrow \varphi)|\forall\mathbf{v}\varphi|\exists\mathbf{v}\varphi$

Bottom-up parse

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$

Bottom-up parse

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$

Bottom-up parse

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$

Bottom-up parse

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$

Bottom-up parse

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$
- $\forall \mathbf{vv}((\mathbf{Atom} \wedge \mathbf{Atom}) \rightarrow \exists \mathbf{vv}((\mathbf{Atom} \wedge \mathbf{Atom}) \wedge (\mathbf{Atom} \wedge \mathbf{Atom})))$

Bottom-up parse

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$
- $\forall \mathbf{vv}((\mathbf{Atom} \wedge \mathbf{Atom}) \rightarrow \exists \mathbf{vv}((\mathbf{Atom} \wedge \mathbf{Atom}) \wedge (\mathbf{Atom} \wedge \mathbf{Atom})))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}((\varphi \wedge \varphi) \wedge (\varphi \wedge \varphi)))$

Bottom-up parse

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$
- $\forall \mathbf{vv}((\mathbf{Atom} \wedge \mathbf{Atom}) \rightarrow \exists \mathbf{vv}((\mathbf{Atom} \wedge \mathbf{Atom}) \wedge (\mathbf{Atom} \wedge \mathbf{Atom})))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}((\varphi \wedge \varphi) \wedge (\varphi \wedge \varphi)))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}((\varphi \wedge \varphi) \wedge \varphi))$

Bottom-up parse

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$
- $\forall \mathbf{vv}((\mathbf{Atom} \wedge \mathbf{Atom}) \rightarrow \exists \mathbf{vv}((\mathbf{Atom} \wedge \mathbf{Atom}) \wedge (\mathbf{Atom} \wedge \mathbf{Atom})))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}((\varphi \wedge \varphi) \wedge (\varphi \wedge \varphi)))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}((\varphi \wedge \varphi) \wedge \varphi))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}(\varphi \wedge \varphi))$

Bottom-up parse

- $\forall wx((Fxw \land Aw) \to \exists yz((Pxy \land Ay) \land (Fxz \land Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \land A\mathbf{v}) \to \exists \mathbf{vv}((P\mathbf{vv} \land A\mathbf{v}) \land (F\mathbf{vv} \land D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \land \mathbf{Pv}) \to \exists \mathbf{vv}((\mathbf{Pvv} \land \mathbf{Pv}) \land (\mathbf{Pvv} \land \mathbf{Pv})))$
- $\forall \mathbf{vv}((\mathbf{Ptt} \land \mathbf{Pt}) \to \exists \mathbf{vv}((\mathbf{Ptt} \land \mathbf{Pt}) \land (\mathbf{Ptt} \land \mathbf{Pt})))$
- $\forall \mathbf{vv}((\mathbf{Atom} \land \mathbf{Atom}) \to \exists \mathbf{vv}((\mathbf{Atom} \land \mathbf{Atom}) \land (\mathbf{Atom} \land \mathbf{Atom})))$
- $\forall \mathbf{vv}((\varphi \land \varphi) \to \exists \mathbf{vv}((\varphi \land \varphi) \land (\varphi \land \varphi)))$
- $\forall \mathbf{vv}((\varphi \land \varphi) \to \exists \mathbf{vv}((\varphi \land \varphi) \land \varphi))$
- $\forall \mathbf{vv}((\varphi \land \varphi) \to \exists \mathbf{vv}(\varphi \land \varphi))$
- $\forall \mathbf{vv}(\varphi \to \exists \mathbf{vv}(\varphi \land \varphi))$

Bottom-up parse

- $\forall wx((Fxw \land Aw) \to \exists yz((Pxy \land Ay) \land (Fxz \land Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \land A\mathbf{v}) \to \exists \mathbf{vv}((P\mathbf{vv} \land A\mathbf{v}) \land (F\mathbf{vv} \land D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \land \mathbf{Pv}) \to \exists \mathbf{vv}((\mathbf{Pvv} \land \mathbf{Pv}) \land (\mathbf{Pvv} \land \mathbf{Pv})))$
- $\forall \mathbf{vv}((\mathbf{Ptt} \land \mathbf{Pt}) \to \exists \mathbf{vv}((\mathbf{Ptt} \land \mathbf{Pt}) \land (\mathbf{Ptt} \land \mathbf{Pt})))$
- $\forall \mathbf{vv}((\mathbf{Atom} \land \mathbf{Atom}) \to \exists \mathbf{vv}((\mathbf{Atom} \land \mathbf{Atom}) \land (\mathbf{Atom} \land \mathbf{Atom})))$
- $\forall \mathbf{vv}((\varphi \land \varphi) \to \exists \mathbf{vv}((\varphi \land \varphi) \land (\varphi \land \varphi)))$
- $\forall \mathbf{vv}((\varphi \land \varphi) \to \exists \mathbf{vv}((\varphi \land \varphi) \land \varphi))$
- $\forall \mathbf{vv}((\varphi \land \varphi) \to \exists \mathbf{vv}(\varphi \land \varphi))$
- $\forall \mathbf{vv}(\varphi \to \exists \mathbf{vv}(\varphi \land \varphi))$
- $\forall \mathbf{vv}(\varphi \to \exists \mathbf{vv}\varphi)$

Bottom-up parse

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$
- $\forall \mathbf{vv}((\mathbf{Atom} \wedge \mathbf{Atom}) \rightarrow \exists \mathbf{vv}((\mathbf{Atom} \wedge \mathbf{Atom}) \wedge (\mathbf{Atom} \wedge \mathbf{Atom})))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}((\varphi \wedge \varphi) \wedge (\varphi \wedge \varphi)))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}((\varphi \wedge \varphi) \wedge \varphi))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}(\varphi \wedge \varphi))$
- $\forall \mathbf{vv}(\varphi \rightarrow \exists \mathbf{vv}(\varphi \wedge \varphi))$
- $\forall \mathbf{vv}(\varphi \rightarrow \exists \mathbf{vv}\varphi)$
- $\forall \mathbf{vv}(\varphi \rightarrow \varphi)$

Bottom-up parse

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$
- $\forall \mathbf{vv}((\mathbf{Atom} \wedge \mathbf{Atom}) \rightarrow \exists \mathbf{vv}((\mathbf{Atom} \wedge \mathbf{Atom}) \wedge (\mathbf{Atom} \wedge \mathbf{Atom})))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}((\varphi \wedge \varphi) \wedge (\varphi \wedge \varphi)))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}((\varphi \wedge \varphi) \wedge \varphi))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}(\varphi \wedge \varphi))$
- $\forall \mathbf{vv}(\varphi \rightarrow \exists \mathbf{vv}(\varphi \wedge \varphi))$
- $\forall \mathbf{vv}(\varphi \rightarrow \exists \mathbf{vv}\varphi)$
- $\forall \mathbf{vv}(\varphi \rightarrow \varphi)$
- $\forall \mathbf{vv}\varphi$

Bottom-up parse

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{vv}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$
- $\forall \mathbf{vv}((\mathbf{Atom} \wedge \mathbf{Atom}) \rightarrow \exists \mathbf{vv}((\mathbf{Atom} \wedge \mathbf{Atom}) \wedge (\mathbf{Atom} \wedge \mathbf{Atom})))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}((\varphi \wedge \varphi) \wedge (\varphi \wedge \varphi)))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}((\varphi \wedge \varphi) \wedge \varphi))$
- $\forall \mathbf{vv}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{vv}(\varphi \wedge \varphi))$
- $\forall \mathbf{vv}(\varphi \rightarrow \exists \mathbf{vv}(\varphi \wedge \varphi))$
- $\forall \mathbf{vv}(\varphi \rightarrow \exists \mathbf{vv}\varphi)$
- $\forall \mathbf{vv}(\varphi \rightarrow \varphi)$
- $\forall \mathbf{vv}\varphi$
- φ

Parsing is a search through a space of possible solutions

We can go wrong!

- $\forall wx((Fxw \land Aw) \to \exists yz((Pxy \land Ay) \land (Fxz \land Dz)))$

Parsing is a search through a space of possible solutions

We can go wrong!

- $\forall wx((Fxw \land Aw) \rightarrow \exists yz((Pxy \land Ay) \land (Fxz \land Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \land A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \land A\mathbf{v}) \land (F\mathbf{vv} \land D\mathbf{v})))$

Parsing is a search through a space of possible solutions

We can go wrong!

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$

Parsing is a search through a space of possible solutions

We can go wrong!

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{tt}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{tt}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$

Parsing is a search through a space of possible solutions

We can go wrong!

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{tt}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{tt}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$
- $\forall \mathbf{tt}((\mathbf{Atom} \wedge \mathbf{Atom}) \rightarrow \exists \mathbf{tt}((\mathbf{Atom} \wedge \mathbf{Atom}) \wedge (\mathbf{Atom} \wedge \mathbf{Atom})))$

Parsing is a search through a space of possible solutions

We can go wrong!

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{tt}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{tt}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$
- $\forall \mathbf{tt}((\mathbf{Atom} \wedge \mathbf{Atom}) \rightarrow \exists \mathbf{tt}((\mathbf{Atom} \wedge \mathbf{Atom}) \wedge (\mathbf{Atom} \wedge \mathbf{Atom})))$
- $\forall \mathbf{tt}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{tt}((\varphi \wedge \varphi) \wedge (\varphi \wedge \varphi)))$

Parsing is a search through a space of possible solutions

We can go wrong!

- $\forall wx((Fxw \land Aw) \rightarrow \exists yz((Pxy \land Ay) \land (Fxz \land Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \land A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \land A\mathbf{v}) \land (F\mathbf{vv} \land D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \land \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \land \mathbf{Pv}) \land (\mathbf{Pvv} \land \mathbf{Pv})))$
- $\forall \mathbf{tt}((\mathbf{Ptt} \land \mathbf{Pt}) \rightarrow \exists \mathbf{tt}((\mathbf{Ptt} \land \mathbf{Pt}) \land (\mathbf{Ptt} \land \mathbf{Pt})))$
- $\forall \mathbf{tt}((\mathbf{Atom} \land \mathbf{Atom}) \rightarrow \exists \mathbf{tt}((\mathbf{Atom} \land \mathbf{Atom}) \land (\mathbf{Atom} \land \mathbf{Atom})))$
- $\forall \mathbf{tt}((\varphi \land \varphi) \rightarrow \exists \mathbf{tt}((\varphi \land \varphi) \land (\varphi \land \varphi)))$
- $\forall \mathbf{tt}((\varphi \land \varphi) \rightarrow \exists \mathbf{tt}((\varphi \land \varphi) \land \varphi))$

Parsing is a search through a space of possible solutions

We can go wrong!

- $\forall wx((Fxw \land Aw) \rightarrow \exists yz((Pxy \land Ay) \land (Fxz \land Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \land A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \land A\mathbf{v}) \land (F\mathbf{vv} \land D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \land \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \land \mathbf{Pv}) \land (\mathbf{Pvv} \land \mathbf{Pv})))$
- $\forall \mathbf{tt}((\mathbf{Ptt} \land \mathbf{Pt}) \rightarrow \exists \mathbf{tt}((\mathbf{Ptt} \land \mathbf{Pt}) \land (\mathbf{Ptt} \land \mathbf{Pt})))$
- $\forall \mathbf{tt}((\mathbf{Atom} \land \mathbf{Atom}) \rightarrow \exists \mathbf{tt}((\mathbf{Atom} \land \mathbf{Atom}) \land (\mathbf{Atom} \land \mathbf{Atom})))$
- $\forall \mathbf{tt}((\varphi \land \varphi) \rightarrow \exists \mathbf{tt}((\varphi \land \varphi) \land (\varphi \land \varphi)))$
- $\forall \mathbf{tt}((\varphi \land \varphi) \rightarrow \exists \mathbf{tt}((\varphi \land \varphi) \land \varphi))$
- $\forall \mathbf{tt}((\varphi \land \varphi) \rightarrow \exists \mathbf{tt}(\varphi \land \varphi))$

Parsing is a search through a space of possible solutions

We can go wrong!

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{tt}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{tt}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$
- $\forall \mathbf{tt}((\mathbf{Atom} \wedge \mathbf{Atom}) \rightarrow \exists \mathbf{tt}((\mathbf{Atom} \wedge \mathbf{Atom}) \wedge (\mathbf{Atom} \wedge \mathbf{Atom})))$
- $\forall \mathbf{tt}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{tt}((\varphi \wedge \varphi) \wedge (\varphi \wedge \varphi)))$
- $\forall \mathbf{tt}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{tt}((\varphi \wedge \varphi) \wedge \varphi))$
- $\forall \mathbf{tt}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{tt}(\varphi \wedge \varphi))$
- $\forall \mathbf{tt}(\varphi \rightarrow \exists \mathbf{tt}(\varphi \wedge \varphi))$

Parsing is a search through a space of possible solutions

We can go wrong!

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{tt}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{tt}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$
- $\forall \mathbf{tt}((\mathbf{Atom} \wedge \mathbf{Atom}) \rightarrow \exists \mathbf{tt}((\mathbf{Atom} \wedge \mathbf{Atom}) \wedge (\mathbf{Atom} \wedge \mathbf{Atom})))$
- $\forall \mathbf{tt}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{tt}((\varphi \wedge \varphi) \wedge (\varphi \wedge \varphi)))$
- $\forall \mathbf{tt}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{tt}((\varphi \wedge \varphi) \wedge \varphi))$
- $\forall \mathbf{tt}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{tt}(\varphi \wedge \varphi))$
- $\forall \mathbf{tt}(\varphi \rightarrow \exists \mathbf{tt}(\varphi \wedge \varphi))$
- $\forall \mathbf{tt}(\varphi \rightarrow \exists \mathbf{tt}\varphi)$

Parsing is a search through a space of possible solutions

We can go wrong!

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{tt}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{tt}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$
- $\forall \mathbf{tt}((\mathbf{Atom} \wedge \mathbf{Atom}) \rightarrow \exists \mathbf{tt}((\mathbf{Atom} \wedge \mathbf{Atom}) \wedge (\mathbf{Atom} \wedge \mathbf{Atom})))$
- $\forall \mathbf{tt}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{tt}((\varphi \wedge \varphi) \wedge (\varphi \wedge \varphi)))$
- $\forall \mathbf{tt}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{tt}((\varphi \wedge \varphi) \wedge \varphi))$
- $\forall \mathbf{tt}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{tt}(\varphi \wedge \varphi))$
- $\forall \mathbf{tt}(\varphi \rightarrow \exists \mathbf{tt}(\varphi \wedge \varphi))$
- $\forall \mathbf{tt}(\varphi \rightarrow \exists \mathbf{tt}\varphi)$
- Stuck! No rule applies, so we must backtrack.

Parsing is a search through a space of possible solutions

We can go wrong!

- $\forall wx((Fxw \wedge Aw) \rightarrow \exists yz((Pxy \wedge Ay) \wedge (Fxz \wedge Dz)))$
- $\forall \mathbf{vv}((F\mathbf{vv} \wedge A\mathbf{v}) \rightarrow \exists \mathbf{vv}((P\mathbf{vv} \wedge A\mathbf{v}) \wedge (F\mathbf{vv} \wedge D\mathbf{v})))$
- $\forall \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \rightarrow \exists \mathbf{vv}((\mathbf{Pvv} \wedge \mathbf{Pv}) \wedge (\mathbf{Pvv} \wedge \mathbf{Pv})))$
- $\forall \mathbf{tt}((\mathbf{Ptt} \wedge \mathbf{Pt}) \rightarrow \exists \mathbf{tt}((\mathbf{Ptt} \wedge \mathbf{Pt}) \wedge (\mathbf{Ptt} \wedge \mathbf{Pt})))$
- $\forall \mathbf{tt}((\mathbf{Atom} \wedge \mathbf{Atom}) \rightarrow \exists \mathbf{tt}((\mathbf{Atom} \wedge \mathbf{Atom}) \wedge (\mathbf{Atom} \wedge \mathbf{Atom})))$
- $\forall \mathbf{tt}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{tt}((\varphi \wedge \varphi) \wedge (\varphi \wedge \varphi)))$
- $\forall \mathbf{tt}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{tt}((\varphi \wedge \varphi) \wedge \varphi))$
- $\forall \mathbf{tt}((\varphi \wedge \varphi) \rightarrow \exists \mathbf{tt}(\varphi \wedge \varphi))$
- $\forall \mathbf{tt}(\varphi \rightarrow \exists \mathbf{tt}(\varphi \wedge \varphi))$
- $\forall \mathbf{tt}(\varphi \rightarrow \exists \mathbf{tt}\varphi)$
- Stuck! No rule applies, so we must backtrack.
- A conforming expression should have *some* path to our start symbol, but how do we program software to make the right choices?

Parsing as search

- Grammars like the ones we've seen typically have the parsing problem of which rules to apply in which order.

Parsing as search

- Grammars like the ones we've seen typically have the parsing problem of which rules to apply in which order.
- Nondeterministic parsing software explores one path of choices, and if no rule applies will back up and try a different path.

Parsing as search

- Grammars like the ones we've seen typically have the parsing problem of which rules to apply in which order.
- Nondeterministic parsing software explores one path of choices, and if no rule applies will back up and try a different path.
- If all possible paths are exhausted for an expression, then the parse fails because the expression doesn't conform to the grammar.

Parsing as search

- Grammars like the ones we've seen typically have the parsing problem of which rules to apply in which order.
- Nondeterministic parsing software explores one path of choices, and if no rule applies will back up and try a different path.
- If all possible paths are exhausted for an expression, then the parse fails because the expression doesn't conform to the grammar.
- The time required to explore all those possibilities is expensive, even for very fast computers.

Parsing as search

- Grammars like the ones we've seen typically have the parsing problem of which rules to apply in which order.
- Nondeterministic parsing software explores one path of choices, and if no rule applies will back up and try a different path.
- If all possible paths are exhausted for an expression, then the parse fails because the expression doesn't conform to the grammar.
- The time required to explore all those possibilities is expensive, even for very fast computers.
- Rules of thumb (heuristics) for ordering the productions can save time, but only on average.

Parsing as search

- Grammars like the ones we've seen typically have the parsing problem of which rules to apply in which order.
- Nondeterministic parsing software explores one path of choices, and if no rule applies will back up and try a different path.
- If all possible paths are exhausted for an expression, then the parse fails because the expression doesn't conform to the grammar.
- The time required to explore all those possibilities is expensive, even for very fast computers.
- Rules of thumb (heuristics) for ordering the productions can save time, but only on average.
- Ideally we'd like an efficient and deterministic path through the search space that allows us to quit early if the expression doesn't conform.

Parsing as search

- Grammars like the ones we've seen typically have the parsing problem of which rules to apply in which order.
- Nondeterministic parsing software explores one path of choices, and if no rule applies will back up and try a different path.
- If all possible paths are exhausted for an expression, then the parse fails because the expression doesn't conform to the grammar.
- The time required to explore all those possibilities is expensive, even for very fast computers.
- Rules of thumb (heuristics) for ordering the productions can save time, but only on average.
- Ideally we'd like an efficient and deterministic path through the search space that allows us to quit early if the expression doesn't conform.
- It turns out that some languages are easy to parse: consider, for example, the set of strings consisting of the letter 'b'.

An easy URL subset grammar

Consider this simple syntax for a subset of URL web addresses:

- Conforming expressions will all begin with `http://`

An easy URL subset grammar

Consider this simple syntax for a subset of URL web addresses:

- Conforming expressions will all begin with `http://`
- followed by strings of one or more lower case letters that are separated by periods,

An easy URL subset grammar

Consider this simple syntax for a subset of URL web addresses:

- Conforming expressions will all begin with `http://`
- followed by strings of one or more lower case letters that are separated by periods,
- the last such string will be one of `com`, `org`, or `edu`,

An easy URL subset grammar

Consider this simple syntax for a subset of URL web addresses:

- Conforming expressions will all begin with `http://`
- followed by strings of one or more lower case letters that are separated by periods,
- the last such string will be one of `com`, `org`, or `edu`,
- then there's a slash,

An easy URL subset grammar

Consider this simple syntax for a subset of URL web addresses:

- Conforming expressions will all begin with `http://`
- followed by strings of one or more lower case letters that are separated by periods,
- the last such string will be one of `com`, `org`, or `edu`,
- then there's a slash,
- then zero or more strings of lower case letters that are separated by slashes

An easy URL subset grammar

Consider this simple syntax for a subset of URL web addresses:

- Conforming expressions will all begin with `http://`
- followed by strings of one or more lower case letters that are separated by periods,
- the last such string will be one of `com`, `org`, or `edu`,
- then there's a slash,
- then zero or more strings of lower case letters that are separated by slashes
- with one of those slashes being the rightmost character.

An easy URL subset grammar

Consider this simple syntax for a subset of URL web addresses:

- Conforming expressions will all begin with `http://`
- followed by strings of one or more lower case letters that are separated by periods,
- the last such string will be one of `com`, `org`, or `edu`,
- then there's a slash,
- then zero or more strings of lower case letters that are separated by slashes
- with one of those slashes being the rightmost character.
- We can easily parse this from left to right, and quit right away if one of the rules is broken.

Parsing a URL

- `http://www.whatever.something.com/abc/cba/wxy/qrs/`

Parsing a URL

- `http://www.whatever.something.com/abc/cba/wxy/qrs/`
- **http:**`//www.whatever.something.com/abc/cba/wxy/qrs/`

Parsing a URL

- `http://www.whatever.something.com/abc/cba/wxy/qrs/`
- **http:**`//www.whatever.something.com/abc/cba/wxy/qrs/`
- **http:**`//`**www.**`whatever.something.com/abc/cba/wxy/qrs/`

Parsing a URL

- `http://www.whatever.something.com/abc/cba/wxy/qrs/`
- **http://**`www.whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.**`whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.**`something.com/abc/cba/wxy/qrs/`

Parsing a URL

- `http://www.whatever.something.com/abc/cba/wxy/qrs/`
- **http://**`www.whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.**`whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.**`something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.something.**`com/abc/cba/wxy/qrs/`

Parsing a URL

- `http://www.whatever.something.com/abc/cba/wxy/qrs/`
- **http:**`//www.whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.**`whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.**`something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.something.**`com/abc/cba/wxy/qrs/`
- **http://www.whatever.something.com/**`abc/cba/wxy/qrs/`

Parsing a URL

- `http://www.whatever.something.com/abc/cba/wxy/qrs/`
- **http:/**`/www.whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.**`whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.**`something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.something.**`com/abc/cba/wxy/qrs/`
- **http://www.whatever.something.com/**`abc/cba/wxy/qrs/`
- **http://www.whatever.something.com/abc/**`cba/wxy/qrs/`

Parsing a URL

- `http://www.whatever.something.com/abc/cba/wxy/qrs/`
- **http:**`//www.whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.**`whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.**`something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.something.**`com/abc/cba/wxy/qrs/`
- **http://www.whatever.something.com/**`abc/cba/wxy/qrs/`
- **http://www.whatever.something.com/abc/**`cba/wxy/qrs/`
- **http://www.whatever.something.com/abc/cba/**`wxy/qrs/`

Parsing a URL

- `http://www.whatever.something.com/abc/cba/wxy/qrs/`
- **http:**`//www.whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.**`whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.**`something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.something.**`com/abc/cba/wxy/qrs/`
- **http://www.whatever.something.com**`/abc/cba/wxy/qrs/`
- **http://www.whatever.something.com/abc/**`cba/wxy/qrs/`
- **http://www.whatever.something.com/abc/cba/**`wxy/qrs/`
- **http://www.whatever.something.com/abc/cba/wxy/**`qrs/`

Parsing a URL

- `http://www.whatever.something.com/abc/cba/wxy/qrs/`
- **http:**`//www.whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.**`whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.**`something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.something.**`com/abc/cba/wxy/qrs/`
- **http://www.whatever.something.com/**`abc/cba/wxy/qrs/`
- **http://www.whatever.something.com/abc/**`cba/wxy/qrs/`
- **http://www.whatever.something.com/abc/cba/**`wxy/qrs/`
- **http://www.whatever.something.com/abc/cba/wxy/**`qrs/`
- **http://www.whatever.something.com/abc/cba/wxy/qrs/**

Parsing a URL

- `http://www.whatever.something.com/abc/cba/wxy/qrs/`
- **http://**`www.whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.**`whatever.something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.**`something.com/abc/cba/wxy/qrs/`
- **http://www.whatever.something.**`com/abc/cba/wxy/qrs/`
- **http://www.whatever.something.com/**`abc/cba/wxy/qrs/`
- **http://www.whatever.something.com/abc/**`cba/wxy/qrs/`
- **http://www.whatever.something.com/abc/cba/**`wxy/qrs/`
- **http://www.whatever.something.com/abc/cba/wxy/**`qrs/`
- **http://www.whatever.something.com/abc/cba/wxy/qrs/**
- **Success!**

Regular expressions

- We can summarize the URL subset grammar using *regular expression* notation.

Regular expressions

- We can summarize the URL subset grammar using *regular expression* notation.
- Not all grammars can be encoded this way, but those that can will admit the kind of efficient, left-to-right parsing shown on the previous slide.

Regular expressions

- We can summarize the URL subset grammar using *regular expression* notation.
- Not all grammars can be encoded this way, but those that can will admit the kind of efficient, left-to-right parsing shown on the previous slide.
- Most popular programming languages, and many text editors include support for regular expressions.

Regular expressions

- We can summarize the URL subset grammar using *regular expression* notation.
- Not all grammars can be encoded this way, but those that can will admit the kind of efficient, left-to-right parsing shown on the previous slide.
- Most popular programming languages, and many text editors include support for regular expressions.
- The full grammar is expressed as `(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*`

Regular expressions

- We can summarize the URL subset grammar using *regular expression* notation.
- Not all grammars can be encoded this way, but those that can will admit the kind of efficient, left-to-right parsing shown on the previous slide.
- Most popular programming languages, and many text editors include support for regular expressions.
- The full grammar is expressed as `(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*`
- `[a-z]` means any single lower case letter.

Regular expressions

- We can summarize the URL subset grammar using *regular expression* notation.
- Not all grammars can be encoded this way, but those that can will admit the kind of efficient, left-to-right parsing shown on the previous slide.
- Most popular programming languages, and many text editors include support for regular expressions.
- The full grammar is expressed as `(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*`
- `[a-z]` means any single lower case letter.
- `[a-z]+` means a string of one or more lower case letters.

Regular expressions

- We can summarize the URL subset grammar using *regular expression* notation.
- Not all grammars can be encoded this way, but those that can will admit the kind of efficient, left-to-right parsing shown on the previous slide.
- Most popular programming languages, and many text editors include support for regular expressions.
- The full grammar is expressed as `(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*`
- `[a-z]` means any single lower case letter.
- `[a-z]+` means a string of one or more lower case letters.
- `\.` means a literal period.

Regular expressions

- We can summarize the URL subset grammar using *regular expression* notation.
- Not all grammars can be encoded this way, but those that can will admit the kind of efficient, left-to-right parsing shown on the previous slide.
- Most popular programming languages, and many text editors include support for regular expressions.
- The full grammar is expressed as `(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*`
- `[a-z]` means any single lower case letter.
- `[a-z]+` means a string of one or more lower case letters.
- `\.` means a literal period.
- `(([a-z]+)\.)+` means one or more sequences of lower case letter strings separated by periods.

Regular expressions

```
(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*
```

- `(com|org|edu)` means one of either `com`, `org`, or `edu`.

Regular expressions

```
(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*
```

- `(com|org|edu)` means one of either `com`, `org`, or `edu`.
- `(([a-z]+)/)*` means zero or more lower case letter strings separated by slashes.

Regular expressions

`(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*`

- `(com|org|edu)` means one of either `com`, `org`, or `edu`.
- `(([a-z]+)/)*` means zero or more lower case letter strings separated by slashes.
- So `(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*` means:

Regular expressions

`(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*`

- `(com|org|edu)` means one of either `com`, `org`, or `edu`.
- `(([a-z]+)/)*` means zero or more lower case letter strings separated by slashes.
- So `(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*` means:
    - `http://` followed by

Regular expressions

```
(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*
```

- `(com|org|edu)` means one of either `com`, `org`, or `edu`.
- `(([a-z]+)/)*` means zero or more lower case letter strings separated by slashes.
- So `(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*` means:
  - `http://` followed by
  - one or more strings of lower case letters, separated by periods, followed by

Regular expressions

```
(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*
```

- `(com|org|edu)` means one of either `com`, `org`, or `edu`.
- `(([a-z]+)/)*` means zero or more lower case letter strings separated by slashes.
- So `(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*` means:
  - `http://` followed by
  - one or more strings of lower case letters, separated by periods, followed by
  - one of either `com`, `org`, or `edu`,

Regular expressions

```
(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*
```

- `(com|org|edu)` means one of either `com`, `org`, or `edu`.
- `(([a-z]+)/)*` means zero or more lower case letter strings separated by slashes.
- So `(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*` means:
  - `http://` followed by
  - one or more strings of lower case letters, separated by periods, followed by
  - one of either `com`, `org`, or `edu`,
  - followed by a slash, followed by

Regular expressions

`(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*`

- `(com|org|edu)` means one of either `com`, `org`, or `edu`.
- `(([a-z]+)/)*` means zero or more lower case letter strings separated by slashes.
- So `(http://)(([a-z]+)\.)+(com|org|edu)/(([a-z]+)/)*` means:
  - `http://` followed by
  - one or more strings of lower case letters, separated by periods, followed by
  - one of either `com`, `org`, or `edu`,
  - followed by a slash, followed by
  - zero or more lower case letter strings separated by slashes.

Abstract state machines

- We can diagram the rules for the URL subset grammar as a state transition diagram.

Abstract state machines

- We can diagram the rules for the URL subset grammar as a state transition diagram.
- States (circles) are situations or configurations of the parser.

Abstract state machines

- We can diagram the rules for the URL subset grammar as a state transition diagram.
- States (circles) are situations or configurations of the parser.
- As we parse the string from left to right, we try to move from the start state (circle number 1) to the goal state (circle 7).

Abstract state machines

- We can diagram the rules for the URL subset grammar as a state transition diagram.
- States (circles) are situations or configurations of the parser.
- As we parse the string from left to right, we try to move from the start state (circle number 1) to the goal state (circle 7).
- If our input matches the label on an arc, we can follow that arrow.

Abstract state machines

- We can diagram the rules for the URL subset grammar as a state transition diagram.
- States (circles) are situations or configurations of the parser.
- As we parse the string from left to right, we try to move from the start state (circle number 1) to the goal state (circle 7).
- If our input matches the label on an arc, we can follow that arrow.
- Otherwise we look for a default arc labeled with an asterisk.

Abstract state machines

- We can diagram the rules for the URL subset grammar as a state transition diagram.
- States (circles) are situations or configurations of the parser.
- As we parse the string from left to right, we try to move from the start state (circle number 1) to the goal state (circle 7).
- If our input matches the label on an arc, we can follow that arrow.
- Otherwise we look for a default arc labeled with an asterisk.
- Our first diagram is almost deterministic, but may still require some backtracking.

Abstract state machines

- We can diagram the rules for the URL subset grammar as a state transition diagram.
- States (circles) are situations or configurations of the parser.
- As we parse the string from left to right, we try to move from the start state (circle number 1) to the goal state (circle 7).
- If our input matches the label on an arc, we can follow that arrow.
- Otherwise we look for a default arc labeled with an asterisk.
- Our first diagram is almost deterministic, but may still require some backtracking.
- The second diagram adds some additional states and arcs, but is completely deterministic.

Nondeterministic Finite State Automaton

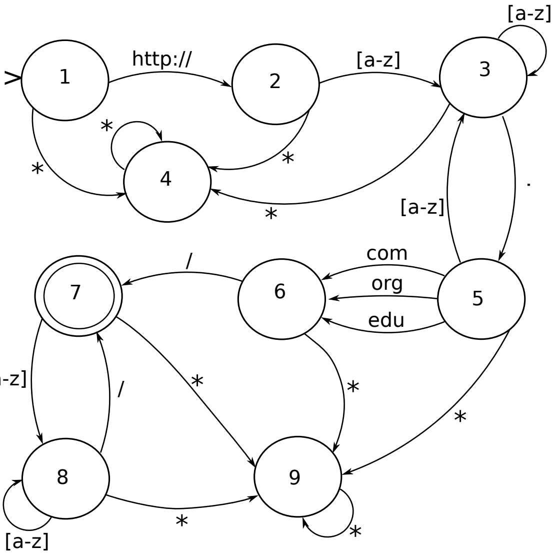

Figure 1: NFA Parser for URL grammar

Deterministic Finite State Automaton

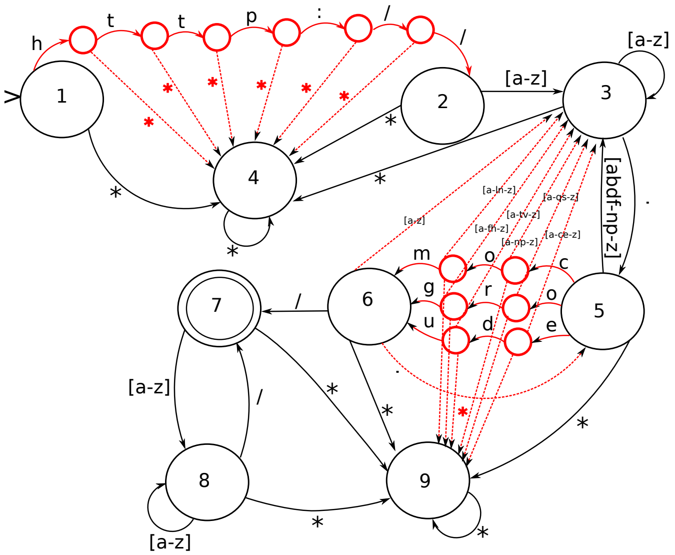

Figure 2: DFA Parser for URL grammar

Top-down derivation

Derivation (a term you read in Rosen) is parsing in reverse.

- φ

Top-down derivation

Derivation (a term you read in Rosen) is parsing in reverse.

- φ
- $\forall \mathbf{v} \varphi$

Top-down derivation

Derivation (a term you read in Rosen) is parsing in reverse.

- φ
- $\forall \mathbf{v} \varphi$
- $\forall \mathbf{v} (\varphi \leftrightarrow \varphi)$

Top-down derivation

Derivation (a term you read in Rosen) is parsing in reverse.

- φ
- $\forall \mathbf{v} \varphi$
- $\forall \mathbf{v}(\varphi \leftrightarrow \varphi)$
- $\forall \mathbf{v}(\varphi \leftrightarrow \exists \mathbf{v} \varphi)$

Top-down derivation

Derivation (a term you read in Rosen) is parsing in reverse.

- φ
- $\forall \mathbf{v} \varphi$
- $\forall \mathbf{v}(\varphi \leftrightarrow \varphi)$
- $\forall \mathbf{v}(\varphi \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v}(\mathbf{Atom} \leftrightarrow \exists \mathbf{v} \varphi)$

Top-down derivation

Derivation (a term you read in Rosen) is parsing in reverse.

- φ
- $\forall \mathbf{v} \varphi$
- $\forall \mathbf{v}(\varphi \leftrightarrow \varphi)$
- $\forall \mathbf{v}(\varphi \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v}(\mathbf{Atom} \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v}(\mathbf{Atom} \leftrightarrow \exists \mathbf{v}\mathbf{Atom})$

Top-down derivation

Derivation (a term you read in Rosen) is parsing in reverse.

- φ
- $\forall \mathbf{v} \varphi$
- $\forall \mathbf{v}(\varphi \leftrightarrow \varphi)$
- $\forall \mathbf{v}(\varphi \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v}(\mathbf{Atom} \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v}(\mathbf{Atom} \leftrightarrow \exists \mathbf{vAtom})$
- $\forall \mathbf{v}(\mathbf{Pt} \leftrightarrow \exists \mathbf{vAtom})$

Top-down derivation

Derivation (a term you read in Rosen) is parsing in reverse.

- φ
- $\forall \mathbf{v} \varphi$
- $\forall \mathbf{v}(\varphi \leftrightarrow \varphi)$
- $\forall \mathbf{v}(\varphi \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v}(\mathbf{Atom} \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v}(\mathbf{Atom} \leftrightarrow \exists \mathbf{vAtom})$
- $\forall \mathbf{v}(\mathbf{Pt} \leftrightarrow \exists \mathbf{vAtom})$
- $\forall \mathbf{v}(\mathbf{Pt} \leftrightarrow \exists \mathbf{vPtt})$

Top-down derivation

Derivation (a term you read in Rosen) is parsing in reverse.

- φ
- $\forall \mathbf{v} \varphi$
- $\forall \mathbf{v}(\varphi \leftrightarrow \varphi)$
- $\forall \mathbf{v}(\varphi \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v}(\mathbf{Atom} \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v}(\mathbf{Atom} \leftrightarrow \exists \mathbf{vAtom})$
- $\forall \mathbf{v}(\mathbf{Pt} \leftrightarrow \exists \mathbf{vAtom})$
- $\forall \mathbf{v}(\mathbf{Pt} \leftrightarrow \exists \mathbf{vPtt})$
- $\forall \mathbf{v}(\mathbf{Pv} \leftrightarrow \exists \mathbf{vPvv})$

Top-down derivation

Derivation (a term you read in Rosen) is parsing in reverse.

- φ
- $\forall \mathbf{v} \varphi$
- $\forall \mathbf{v} (\varphi \leftrightarrow \varphi)$
- $\forall \mathbf{v} (\varphi \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v} (\mathbf{Atom} \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v} (\mathbf{Atom} \leftrightarrow \exists \mathbf{vAtom})$
- $\forall \mathbf{v} (\mathbf{Pt} \leftrightarrow \exists \mathbf{vAtom})$
- $\forall \mathbf{v} (\mathbf{Pt} \leftrightarrow \exists \mathbf{vPtt})$
- $\forall \mathbf{v} (\mathbf{Pv} \leftrightarrow \exists \mathbf{vPvv})$
- $\forall \mathbf{v} (\mathbf{Pv} \leftrightarrow \exists \mathbf{v} L \mathbf{vv})$

Top-down derivation

Derivation (a term you read in Rosen) is parsing in reverse.

- φ
- $\forall \mathbf{v} \varphi$
- $\forall \mathbf{v}(\varphi \leftrightarrow \varphi)$
- $\forall \mathbf{v}(\varphi \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v}(\mathbf{Atom} \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v}(\mathbf{Atom} \leftrightarrow \exists \mathbf{vAtom})$
- $\forall \mathbf{v}(\mathbf{Pt} \leftrightarrow \exists \mathbf{vAtom})$
- $\forall \mathbf{v}(\mathbf{Pt} \leftrightarrow \exists \mathbf{vPtt})$
- $\forall \mathbf{v}(\mathbf{Pv} \leftrightarrow \exists \mathbf{vPvv})$
- $\forall \mathbf{v}(\mathbf{Pv} \leftrightarrow \exists \mathbf{v} L \mathbf{vv})$
- $\forall \mathbf{v}(V \mathbf{v} \leftrightarrow \exists \mathbf{v} L \mathbf{vv})$

Top-down derivation

Derivation (a term you read in Rosen) is parsing in reverse.

- φ
- $\forall \mathbf{v} \varphi$
- $\forall \mathbf{v} (\varphi \leftrightarrow \varphi)$
- $\forall \mathbf{v} (\varphi \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v} (\mathbf{Atom} \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v} (\mathbf{Atom} \leftrightarrow \exists \mathbf{v} \mathbf{Atom})$
- $\forall \mathbf{v} (\mathbf{Pt} \leftrightarrow \exists \mathbf{v} \mathbf{Atom})$
- $\forall \mathbf{v} (\mathbf{Pt} \leftrightarrow \exists \mathbf{v} \mathbf{Ptt})$
- $\forall \mathbf{v} (\mathbf{Pv} \leftrightarrow \exists \mathbf{v} \mathbf{Pvv})$
- $\forall \mathbf{v} (\mathbf{Pv} \leftrightarrow \exists \mathbf{v} L\mathbf{vv})$
- $\forall \mathbf{v} (V\mathbf{v} \leftrightarrow \exists \mathbf{v} L\mathbf{vv})$
- $\forall \mathbf{v} (V\mathbf{v} \leftrightarrow \exists z L\mathbf{v}z)$

Top-down derivation

Derivation (a term you read in Rosen) is parsing in reverse.

- φ
- $\forall \mathbf{v} \varphi$
- $\forall \mathbf{v}(\varphi \leftrightarrow \varphi)$
- $\forall \mathbf{v}(\varphi \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v}(\mathbf{Atom} \leftrightarrow \exists \mathbf{v} \varphi)$
- $\forall \mathbf{v}(\mathbf{Atom} \leftrightarrow \exists \mathbf{v} \mathbf{Atom})$
- $\forall \mathbf{v}(\mathbf{Pt} \leftrightarrow \exists \mathbf{v} \mathbf{Atom})$
- $\forall \mathbf{v}(\mathbf{Pt} \leftrightarrow \exists \mathbf{v} \mathbf{Ptt})$
- $\forall \mathbf{v}(\mathbf{Pv} \leftrightarrow \exists \mathbf{v} \mathbf{Pvv})$
- $\forall \mathbf{v}(\mathbf{Pv} \leftrightarrow \exists \mathbf{v} L\mathbf{vv})$
- $\forall \mathbf{v}(V\mathbf{v} \leftrightarrow \exists \mathbf{v} L\mathbf{vv})$
- $\forall \mathbf{v}(V\mathbf{v} \leftrightarrow \exists z L\mathbf{v}z)$
- $\forall x(Vx \leftrightarrow \exists z Lxz)$

Formalization: Rosen on vocabularies

A vocabulary (or alphabet) V is a finite nonempty set of elements, called symbols. A word (or sentence) over V is a string of finite length of elements of V. The empty string or null string, denoted by λ, is the string containing no symbols. The set of all words over V is denoted by V^*. A language over V is a subset of V^*.

Rosen on grammars

A phrase-structure grammar $G = \langle V, T, S, P \rangle$ *consists of a vocabulary* V*, a subset* T *of* V *consisting of terminal elements, a start symbol* S *from* V*, and a set of productions* P*. The set* $V - T$ *is denoted by* N*. Elements of* N *are called nonterminal symbols. Every production in* P *must contain at least one nonterminal on its left side.*

Phrase-structure grammar

```
<protocol> ::= http://
<letter>   ::= a|b|c|d|e|f|g|h|i|j|k|l|m
<letter>   ::= n|o|p|q|r|s|t|u|v|w|x|y|z
<slash>    ::= /
<dot>      ::= .
<string>   ::= <letter><string>|<letter>
<host>     ::= <string><dot><host>|<string><dot>
<domain>   ::= com|org|edu
<site>     ::= <host><domain><slash>
<dir>      ::= <string><slash>
<body>     ::= <dir><body>|<dir>
<url>      ::= <protocol><site><body>|<protocol><site>
```

Rosen on languages and derivability

Let $G = \langle V, T, S, P \rangle$ be a phrase-structure grammar. Let $w_0 = lz_0r$ (that is, the concatenation of l, z_0, and r) and $w_1 = lz_1r$ be strings over V. If $z_0 \rightarrow z_1$ is a production of G, we say that w_1 is directly derivable from w_0 and we write $w_0 \Rightarrow w_1$. If $w_0, w_1, \ldots w_n, n \geq 0$, are strings over V such that $w_0 \Rightarrow w_1, w_1 \Rightarrow w_2, \ldots w_{n-1} \Rightarrow w_n$, then we say that w_n is derivable from w_0, and we write $w_0 \overset{*}{\Rightarrow} w_n$. The sequence of steps used to obtain w_n from w_0 is called a derivation.

Rosen on languages and derivability

Let $G = \langle V, T, S, P \rangle$ be a phrase-structure grammar. Let $w_0 = lz_0r$ (that is, the concatenation of l, z_0, and r) and $w_1 = lz_1r$ be strings over V. If $z_0 \rightarrow z_1$ is a production of G, we say that w_1 is directly derivable from w_0 and we write $w_0 \Rightarrow w_1$. If $w_0, w_1, \ldots w_n, n \geq 0$, are strings over V such that $w_0 \Rightarrow w_1, w_1 \Rightarrow w_2, \ldots w_{n-1} \Rightarrow w_n$, then we say that w_n is derivable from w_0, and we write $w_0 \overset{*}{\Rightarrow} w_n$. The sequence of steps used to obtain w_n from w_0 is called a derivation.

Let $G = \langle V, T, S, P \rangle$ be a phrase-structure grammar. The language generated by G (or the language of G), denoted by $L(G)$, is the set of all strings of terminals that are derivable from the starting state S. In other words, $L(G) = \{w \in T^* | S \overset{*}{\Rightarrow} w\}$.

Production type determines grammar classification

Per Rosen, section 10.1:

- A *type 0* grammar has no restriction on its productions.

Production type determines grammar classification

Per Rosen, section 10.1:

- A *type 0* grammar has no restriction on its productions.
- A *type 1* grammar can have productions only of the form $w_1 \rightarrow w_2$, where the length of w_2 is greater than or equal to the length of w_1, or of the form $w_1 \rightarrow \lambda$.

Production type determines grammar classification

Per Rosen, section 10.1:

- A *type 0* grammar has no restriction on its productions.
- A *type 1* grammar can have productions only of the form $w_1 \rightarrow w_2$, where the length of w_2 is greater than or equal to the length of w_1, or of the form $w_1 \rightarrow \lambda$.
- A *type 2* (context free) grammar can have productions only of the form $w_1 \rightarrow w_2$, where w_1 is a single symbol that is not a terminal symbol.

Production type determines grammar classification

Per Rosen, section 10.1:

- A *type 0* grammar has no restriction on its productions.
- A *type 1* grammar can have productions only of the form $w_1 \rightarrow w_2$, where the length of w_2 is greater than or equal to the length of w_1, or of the form $w_1 \rightarrow \lambda$.
- A *type 2* (context free) grammar can have productions only of the form $w_1 \rightarrow w_2$, where w_1 is a single symbol that is not a terminal symbol.
- A *type 3* (regular) grammar can have productions only of the form $w_1 \rightarrow w_2$ with $w_1 = A$, and either $w_2 = aB$ or $w_2 = a$, where A and B are nonterminal symbols and a is a terminal symbol, or with $w_1 = S$ and $w_2 = \lambda$.

Chomsky hierarchy implications

- Regular languages (like our URL subset) can be summarized using DFAs, NFAs, and regular expressions.

Chomsky hierarchy implications

- Regular languages (like our URL subset) can be summarized using DFAs, NFAs, and regular expressions.
- Therefore, their expressions can be parsed in very little time, using little memory.

Chomsky hierarchy implications

- Regular languages (like our URL subset) can be summarized using DFAs, NFAs, and regular expressions.
- Therefore, their expressions can be parsed in very little time, using little memory.
- Our grammars for propositional and predicate logic are *not* regular, but they are context free.

Chomsky hierarchy implications

- Regular languages (like our URL subset) can be summarized using DFAs, NFAs, and regular expressions.
- Therefore, their expressions can be parsed in very little time, using little memory.
- Our grammars for propositional and predicate logic are *not* regular, but they are context free.
- No regular expression is powerful enough to recognize any arbitrary logic expression, because logics admit arbitrarily deep levels of nesting: we can always open up a new set of parentheses, just as with Boolean search languages.

Chomsky hierarchy implications

- Regular languages (like our URL subset) can be summarized using DFAs, NFAs, and regular expressions.
- Therefore, their expressions can be parsed in very little time, using little memory.
- Our grammars for propositional and predicate logic are *not* regular, but they are context free.
- No regular expression is powerful enough to recognize any arbitrary logic expression, because logics admit arbitrarily deep levels of nesting: we can always open up a new set of parentheses, just as with Boolean search languages.
- Parsing non-regular, context free languages like our logic grammar always requires a stack memory, and often requires backtracking.

Chomsky hierarchy implications

- Regular languages (like our URL subset) can be summarized using DFAs, NFAs, and regular expressions.
- Therefore, their expressions can be parsed in very little time, using little memory.
- Our grammars for propositional and predicate logic are *not* regular, but they are context free.
- No regular expression is powerful enough to recognize any arbitrary logic expression, because logics admit arbitrarily deep levels of nesting: we can always open up a new set of parentheses, just as with Boolean search languages.
- Parsing non-regular, context free languages like our logic grammar always requires a stack memory, and often requires backtracking.
- We can usually tell that a language is not regular by seeing whether its productions meet Rosen's constraints.

Context free, non-regular grammars

- $\mathbf{v} ::= x|y|z| \ldots$

Context free, non-regular grammars

- **v** $::= x|y|z|\ldots$
- **c** $::= a|b|c|\ldots$

Context free, non-regular grammars

- $\mathbf{v} ::= x|y|z| \ldots$
- $\mathbf{c} ::= a|b|c| \ldots$
- $\mathbf{t} ::= \mathbf{v}|\mathbf{c}$

Context free, non-regular grammars

- $\mathbf{v} ::= x|y|z| \ldots$
- $\mathbf{c} ::= a|b|c| \ldots$
- $\mathbf{t} ::= \mathbf{v}|\mathbf{c}$
- $\mathbf{P} ::= P|Q|R| \ldots$

Context free, non-regular grammars

- $\mathbf{v} ::= x|y|z|\ldots$
- $\mathbf{c} ::= a|b|c|\ldots$
- $\mathbf{t} ::= \mathbf{v}|\mathbf{c}$
- $\mathbf{P} ::= P|Q|R|\ldots$
- $\mathbf{Atom} ::= \mathbf{Pt}_1 \ldots \mathbf{t}_n$ where n is the arity of $\mathbf{P}$

Context free, non-regular grammars

- $\mathbf{v} ::= x|y|z|\ldots$
- $\mathbf{c} ::= a|b|c|\ldots$
- $\mathbf{t} ::= \mathbf{v}|\mathbf{c}$
- $\mathbf{P} ::= P|Q|R|\ldots$
- $\mathbf{Atom} ::= \mathbf{Pt}_1\ldots\mathbf{t}_n$ where n is the arity of $\mathbf{P}$
- $\varphi ::= \mathbf{Atom}|\neg\varphi|(\varphi\wedge\varphi)|(\varphi\vee\varphi)|(\varphi\rightarrow\varphi)|(\varphi\leftrightarrow\varphi)|\forall\mathbf{v}\varphi|\exists\mathbf{v}\varphi$

Caveats for the Rosen definitions

- Being regular and being context-free are really properties of languages, not grammars.

Caveats for the Rosen definitions

- Being regular and being context-free are really properties of languages, not grammars.
- Some grammars that don't satisfy Rosen's type 3 definition still summarize regular languages.

Caveats for the Rosen definitions

- Being regular and being context-free are really properties of languages, not grammars.
- Some grammars that don't satisfy Rosen's type 3 definition still summarize regular languages.
- It can be challenging to verify whether a language is regular just by looking at the productions.

The URL subset is a regular language

```
<protocol> ::= http://
<letter>   ::= a|b|c|d|e|f|g|h|i|j|k|l|m
<letter>   ::= n|o|p|q|r|s|t|u|v|w|x|y|z
<slash>    ::= /
<dot>      ::= .
<string>   ::= <letter><string>|<letter>
<host>     ::= <string><dot><host>|<string><dot>
<domain>   ::= com|org|edu
<site>     ::= <host><domain><slash>
<dir>      ::= <string><slash>
<body>     ::= <dir><body>|<dir>
<url>      ::= <protocol><site><body>|<protocol><site>
```

Context free, non-regular grammars

- $\langle const \rangle ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p$

Context free, non-regular grammars

- $\langle const \rangle ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p$
- $\langle var \rangle ::= q|r|s|t|u|v|w|x|y|z$

Context free, non-regular grammars

- $\langle const \rangle ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p$
- $\langle var \rangle ::= q|r|s|t|u|v|w|x|y|z$
- $\langle pred \rangle ::= A|B|C|D|E|F|G|H|I|J|K|L|M$

Context free, non-regular grammars

- $\langle const \rangle ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p$
- $\langle var \rangle ::= q|r|s|t|u|v|w|x|y|z$
- $\langle pred \rangle ::= A|B|C|D|E|F|G|H|I|J|K|L|M$
- $\langle pred \rangle ::= N|O|P|Q|R|S|T|U|V|W|X|Y|Z$

Context free, non-regular grammars

- $\langle const \rangle ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p$
- $\langle var \rangle ::= q|r|s|t|u|v|w|x|y|z$
- $\langle pred \rangle ::= A|B|C|D|E|F|G|H|I|J|K|L|M$
- $\langle pred \rangle ::= N|O|P|Q|R|S|T|U|V|W|X|Y|Z$
- $\langle lp \rangle ::= ($

Context free, non-regular grammars

- $\langle const \rangle ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p$
- $\langle var \rangle ::= q|r|s|t|u|v|w|x|y|z$
- $\langle pred \rangle ::= A|B|C|D|E|F|G|H|I|J|K|L|M$
- $\langle pred \rangle ::= N|O|P|Q|R|S|T|U|V|W|X|Y|Z$
- $\langle lp \rangle ::= ($
- $\langle rp \rangle ::=)$

Context free, non-regular grammars

- $\langle const \rangle$ ::= $a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p$
- $\langle var \rangle$ ::= $q|r|s|t|u|v|w|x|y|z$
- $\langle pred \rangle$ ::= $A|B|C|D|E|F|G|H|I|J|K|L|M$
- $\langle pred \rangle$ ::= $N|O|P|Q|R|S|T|U|V|W|X|Y|Z$
- $\langle lp \rangle$ ::= (
- $\langle rp \rangle$ ::=)
- $\langle quant \rangle$ ::= $\forall|\exists$

Context free, non-regular grammars

- $\langle const \rangle$ ::= $a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p$
- $\langle var \rangle$ ::= $q|r|s|t|u|v|w|x|y|z$
- $\langle pred \rangle$ ::= $A|B|C|D|E|F|G|H|I|J|K|L|M$
- $\langle pred \rangle$ ::= $N|O|P|Q|R|S|T|U|V|W|X|Y|Z$
- $\langle lp \rangle$ ::= (
- $\langle rp \rangle$ ::=)
- $\langle quant \rangle$ ::= $\forall|\exists$
- $\langle not \rangle$ ::= $\neg$

Context free, non-regular grammars

- $\langle const \rangle$ ::= $a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p$
- $\langle var \rangle$ ::= $q|r|s|t|u|v|w|x|y|z$
- $\langle pred \rangle$ ::= $A|B|C|D|E|F|G|H|I|J|K|L|M$
- $\langle pred \rangle$ ::= $N|O|P|Q|R|S|T|U|V|W|X|Y|Z$
- $\langle lp \rangle$ ::= (
- $\langle rp \rangle$ ::=)
- $\langle quant \rangle$ ::= $\forall|\exists$
- $\langle not \rangle$ ::= $\neg$
- $\langle binop \rangle$ ::= $\wedge|\vee|\rightarrow|\leftrightarrow$

Context free, non-regular grammars

- $\langle const \rangle ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p$
- $\langle var \rangle ::= q|r|s|t|u|v|w|x|y|z$
- $\langle pred \rangle ::= A|B|C|D|E|F|G|H|I|J|K|L|M$
- $\langle pred \rangle ::= N|O|P|Q|R|S|T|U|V|W|X|Y|Z$
- $\langle lp \rangle ::= ($
- $\langle rp \rangle ::=)$
- $\langle quant \rangle ::= \forall|\exists$
- $\langle not \rangle ::= \neg$
- $\langle binop \rangle ::= \wedge|\vee|\rightarrow|\leftrightarrow$
- $\langle term \rangle ::= \langle const \rangle|\langle var \rangle$

Context free, non-regular grammars

- $\langle const \rangle$ ::= $a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p$
- $\langle var \rangle$ ::= $q|r|s|t|u|v|w|x|y|z$
- $\langle pred \rangle$ ::= $A|B|C|D|E|F|G|H|I|J|K|L|M$
- $\langle pred \rangle$ ::= $N|O|P|Q|R|S|T|U|V|W|X|Y|Z$
- $\langle lp \rangle$ ::= (
- $\langle rp \rangle$ ::=)
- $\langle quant \rangle$ ::= $\forall|\exists$
- $\langle not \rangle$ ::= $\neg$
- $\langle binop \rangle$ ::= $\wedge|\vee|\rightarrow|\leftrightarrow$
- $\langle term \rangle$ ::= $\langle const \rangle|\langle var \rangle$
- $\langle atom \rangle$ ::= $\langle pred \rangle\langle term \rangle|\langle atom \rangle\langle term \rangle$

Context free, non-regular grammars

- $\langle const \rangle$ ::= $a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p$
- $\langle var \rangle$ ::= $q|r|s|t|u|v|w|x|y|z$
- $\langle pred \rangle$ ::= $A|B|C|D|E|F|G|H|I|J|K|L|M$
- $\langle pred \rangle$ ::= $N|O|P|Q|R|S|T|U|V|W|X|Y|Z$
- $\langle lp \rangle$ ::= (
- $\langle rp \rangle$ ::=)
- $\langle quant \rangle$ ::= $\forall|\exists$
- $\langle not \rangle$ ::= $\neg$
- $\langle binop \rangle$ ::= $\wedge|\vee|\rightarrow|\leftrightarrow$
- $\langle term \rangle$ ::= $\langle const \rangle|\langle var \rangle$
- $\langle atom \rangle$ ::= $\langle pred \rangle\langle term \rangle|\langle atom \rangle\langle term \rangle$
- $\langle phi \rangle$ ::= $\langle atom \rangle|\langle not \rangle\langle phi \rangle|\langle quant \rangle\langle var \rangle\langle phi \rangle$

Context free, non-regular grammars

- $\langle const\rangle$ ::= $a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p$
- $\langle var\rangle$ ::= $q|r|s|t|u|v|w|x|y|z$
- $\langle pred\rangle$ ::= $A|B|C|D|E|F|G|H|I|J|K|L|M$
- $\langle pred\rangle$ ::= $N|O|P|Q|R|S|T|U|V|W|X|Y|Z$
- $\langle lp\rangle$ ::= (
- $\langle rp\rangle$ ::=)
- $\langle quant\rangle$ ::= $\forall|\exists$
- $\langle not\rangle$ ::= $\neg$
- $\langle binop\rangle$ ::= $\wedge|\vee|\rightarrow|\leftrightarrow$
- $\langle term\rangle$ ::= $\langle const\rangle|\langle var\rangle$
- $\langle atom\rangle$ ::= $\langle pred\rangle\langle term\rangle|\langle atom\rangle\langle term\rangle$
- $\langle phi\rangle$ ::= $\langle atom\rangle|\langle not\rangle\langle phi\rangle|\langle quant\rangle\langle var\rangle\langle phi\rangle$
- $\langle phi\rangle$ ::= $\langle lp\rangle\langle phi\rangle\langle binop\rangle\langle phi\rangle\langle rp\rangle$

Parsing context free languages

- Adding a stack memory to our NFA gives us a *push down automaton* (PDA).

Parsing context free languages

- Adding a stack memory to our NFA gives us a *push down automaton* (PDA).
- We only access a stack at one end: pushing a symbol on top, or popping one off and discarding it.

Parsing context free languages

- Adding a stack memory to our NFA gives us a *push down automaton* (PDA).
- We only access a stack at one end: pushing a symbol on top, or popping one off and discarding it.
- We can't search for a symbol in the middle of a stack, but that means that stack access is always independent of how much material is stored there.

Parsing context free languages

- Adding a stack memory to our NFA gives us a *push down automaton* (PDA).
- We only access a stack at one end: pushing a symbol on top, or popping one off and discarding it.
- We can't search for a symbol in the middle of a stack, but that means that stack access is always independent of how much material is stored there.
- Transitions on the next diagram include requirements for the next input symbol (just like the NFA), operations on the stack, or both.

Parsing context free languages

- Adding a stack memory to our NFA gives us a *push down automaton* (PDA).
- We only access a stack at one end: pushing a symbol on top, or popping one off and discarding it.
- We can't search for a symbol in the middle of a stack, but that means that stack access is always independent of how much material is stored there.
- Transitions on the next diagram include requirements for the next input symbol (just like the NFA), operations on the stack, or both.
- −/− means leave the stack unchanged;

Parsing context free languages

- Adding a stack memory to our NFA gives us a *push down automaton* (PDA).
- We only access a stack at one end: pushing a symbol on top, or popping one off and discarding it.
- We can't search for a symbol in the middle of a stack, but that means that stack access is always independent of how much material is stored there.
- Transitions on the next diagram include requirements for the next input symbol (just like the NFA), operations on the stack, or both.
- −/− means leave the stack unchanged;
- /\$ means push \$ on the top of the stack;

Parsing context free languages

- Adding a stack memory to our NFA gives us a *push down automaton* (PDA).
- We only access a stack at one end: pushing a symbol on top, or popping one off and discarding it.
- We can't search for a symbol in the middle of a stack, but that means that stack access is always independent of how much material is stored there.
- Transitions on the next diagram include requirements for the next input symbol (just like the NFA), operations on the stack, or both.
- −/− means leave the stack unchanged;
- /\$ means push \$ on the top of the stack;
- \$/ means pop \$ off the top of the stack;

Pushdown Automaton

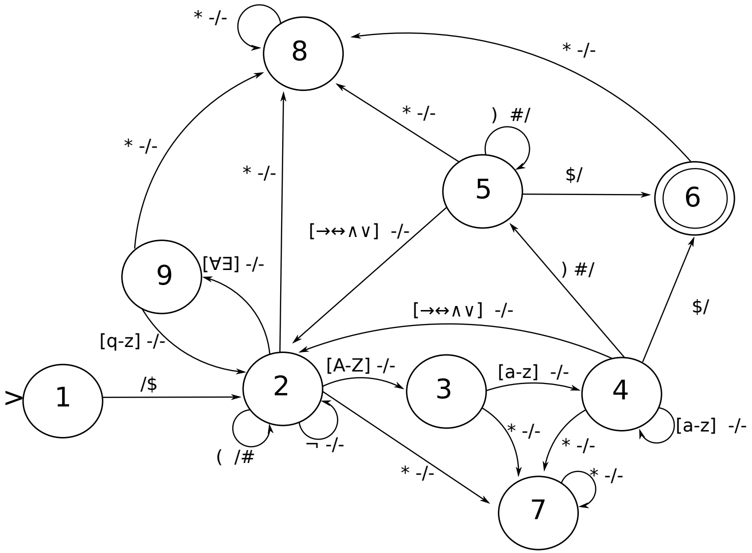

Figure 3: PDA Parser for the predicate logic grammar