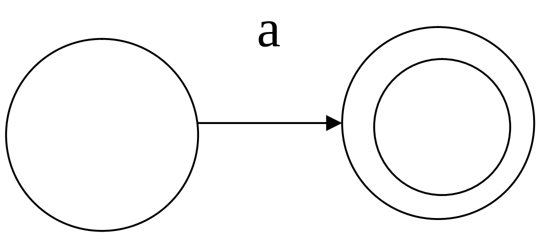
Regular Language to NFA

1. Regular Expression → NFA

a. Derivatives are backward edges of NFA

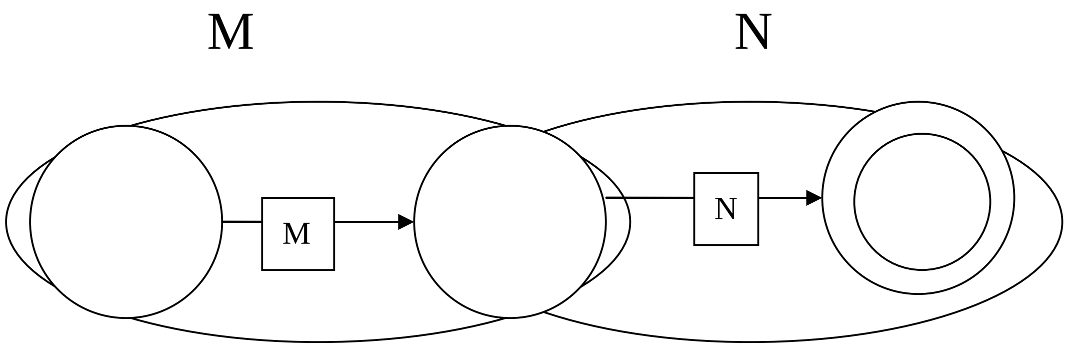
b. Thompson construction:

a



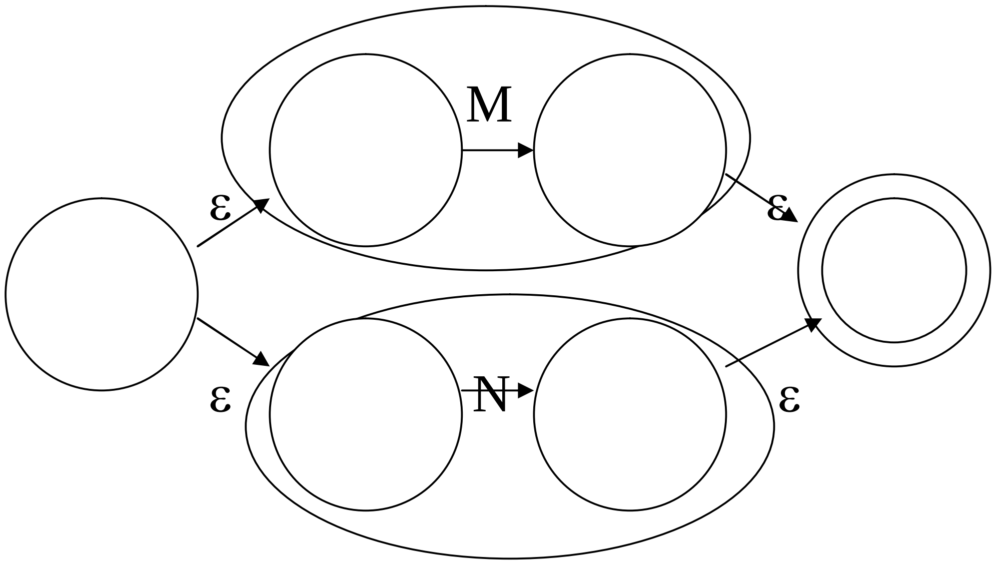
1. concatenation of machines

MN

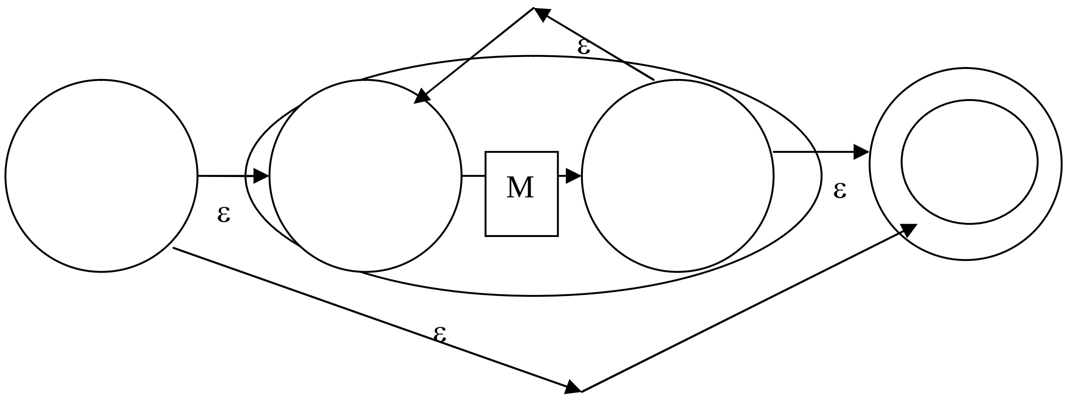


2. union – four epsilon edges

M | N



3. Kleene star M*



NFA to a DFA

Subset Construction Algorithm

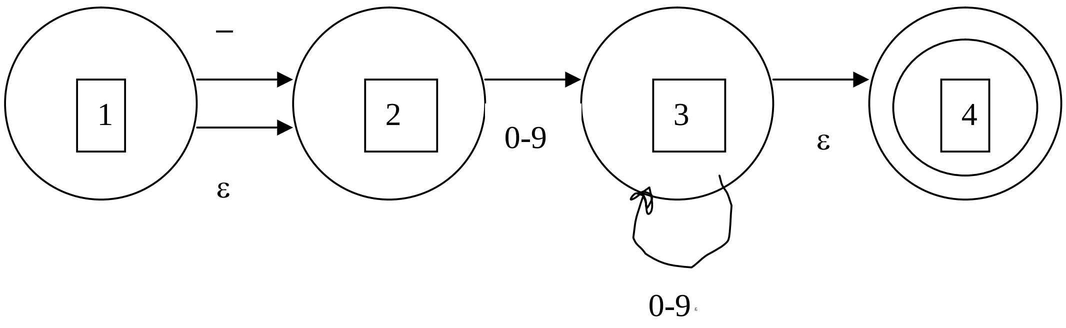
• An NFA is inefficient to implement directly. Therefore convert to a DFA that recognizes the same strings.

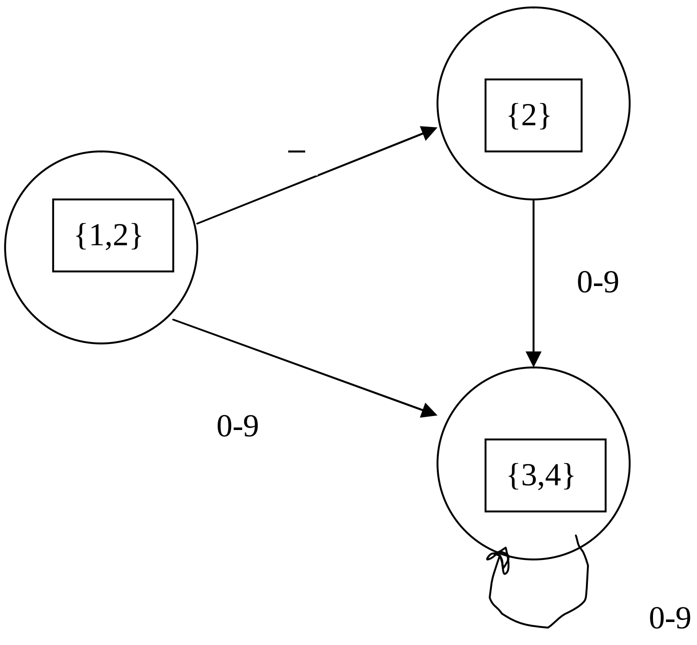
• Subset Construction Algorithm:

1. NFA can be in multiple states simultaneously.
2. Each DFA state corresponds to a distinct set of NFA states.
3. n-state NFA may be 2^n state DFA in worst case.

NFA to DFA Example

Subset Construction Algorithm

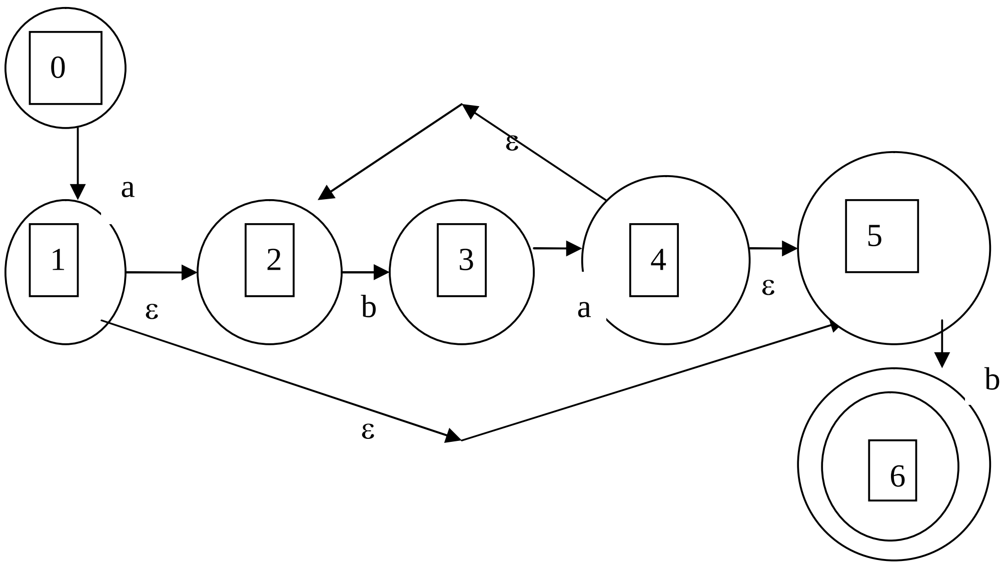




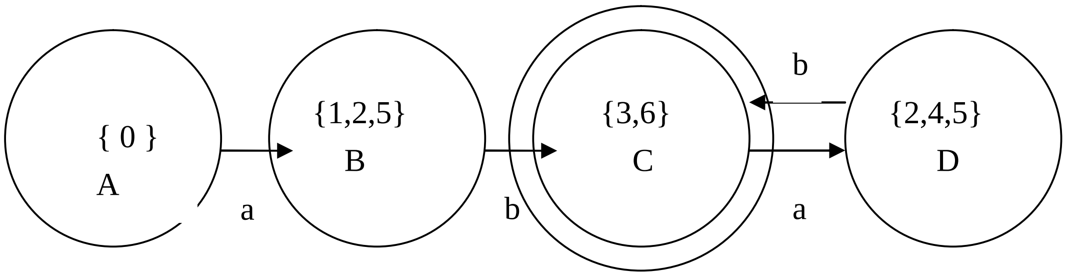
Regular Expression to NFA

Using Thompson Construction

a (ba) * b

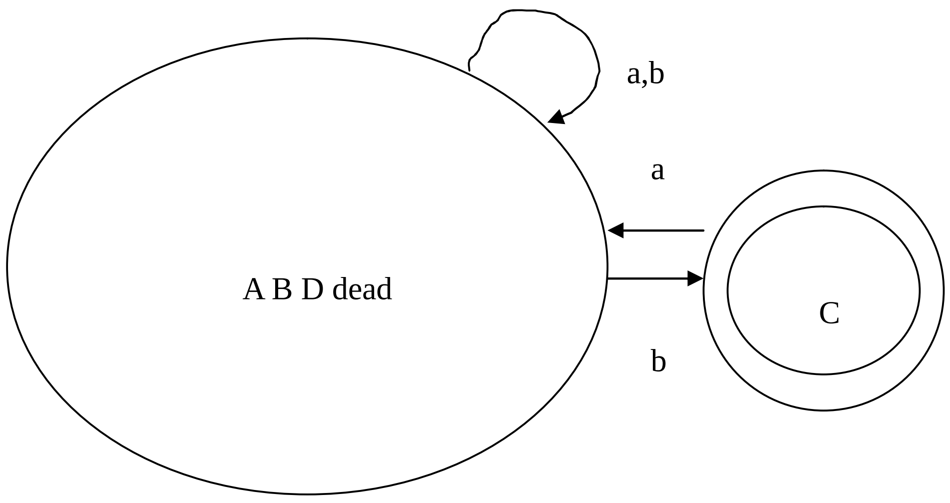


NFA to DFA

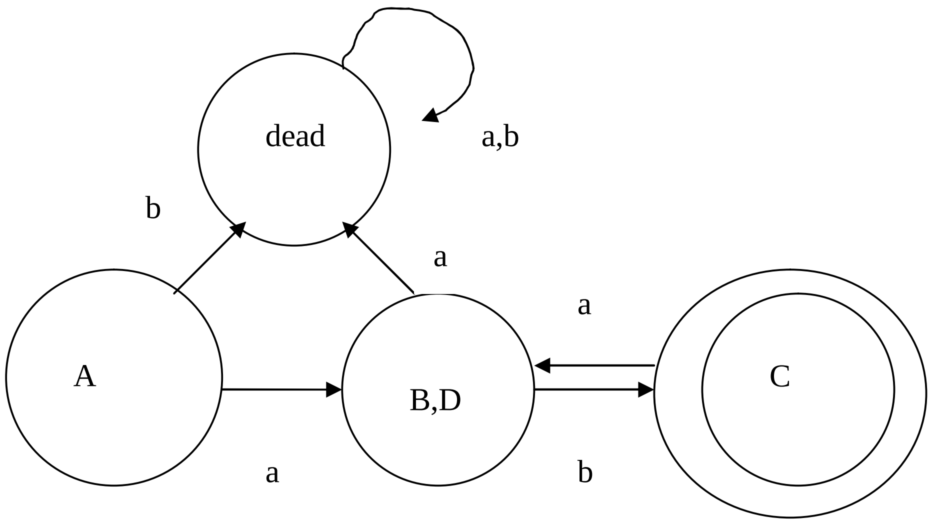


Minimized DFA

Step 1: Final vs. non-final states



Step k – Separate (partition) those states which go to different partitions on a given input



(e.g. A and dead go to different partitions on “a”)

Programming Homework #1 Scanner

Due March 20, 2008

• Create a lexical analyzer (scanner) for the MiniJava language (in the Appendix). Print the lexemes for the sample program on page 486.

You will be building a compiler for MiniJava using one of these tools:

1. SableCC – an LALR(1) tool (builds AST – abstract syntax tree for visitor design pattern)
2. JavaCC – an LL(1) tool with lookaheads (uses JJTree to build the parse tree)
3. JLex and Cup LALR(1) (build your own parse tree) (You can also choose JFLEX)

Available from Appel's website.

http://www.cs.princeton.edu/~appel/modern/

HW #1 (continued)

• Your scanner may not assume any limits on the lengths of identifiers, strings, integers, comments, etc. Additionally, care must be taken to ensure that the values of integers are numerically accurate and that errors such as numeric overflow are detected.

Encountering an error, your scanner must print an informative error message and then exit immediately.

• Your scanner must be able to detect erroneous double-quoted strings which fail to have a terminating double quote prior end-of-line. Similarly, your scanner must be able to detect erroneous comments which fail to have terminating */ prior to end-of-file.

MiniJava Language Lexical Specification

During lexical analysis, characters in MiniJava source text are reduced to a series of tokens. The MiniJava compiler recognizes five kinds of tokens: reserved words, identifiers, integer literals, operators, and separators. Comments and white spaces such as blanks (spaces), tabs, and line feeds are not tokens and will be discarded.

•**Comments**

Comments start with /* and end with */ and may be nested.

/* this /* is /* a */ comment */ line */

// This is also a comment