CS 601 Algorithmic Complexity

Instructor Sandeep Bhatt
sandeep.bhatt@stevens.edu

Course Assistant: Ramana Nagasamudram
rnagasam@stevens.edu

Office hours and zoom links will be posted on Canvas

Course Organization

Lectures

TuTh 2-3:15, Babbio 203

On zoom in case of inclement weather (link is in Canvas modules)

Textbook

Introduction to the Theory of Computation, 3rd Edition, 2012, by Michael Sipser.

Course Work and Grading

10%	Participation
35%	Problem Sets
20%	Midterm
35%	Final

Final letter grades will be curved

Any grade dispute must be brought to Ramana’s attention within one week of the grade being issued.

Class Participation

Be actively engaged in class: ask and answer questions.

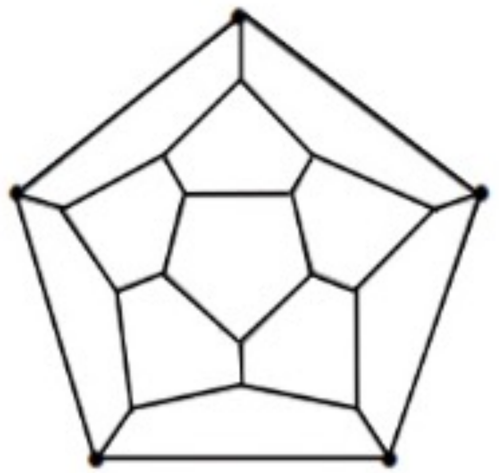
Seek help during office hours.

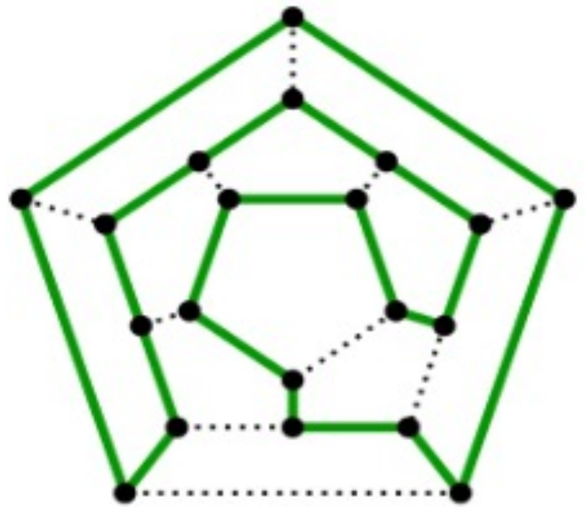
- Think through a problem on your own before seeking help.
- It is acceptable to discuss ideas in groups, but the final submission must be your own.

A Seemingly Complex Problem

Does this graph have a cycle that contains every vertex exactly once?

Hamiltonian cycle

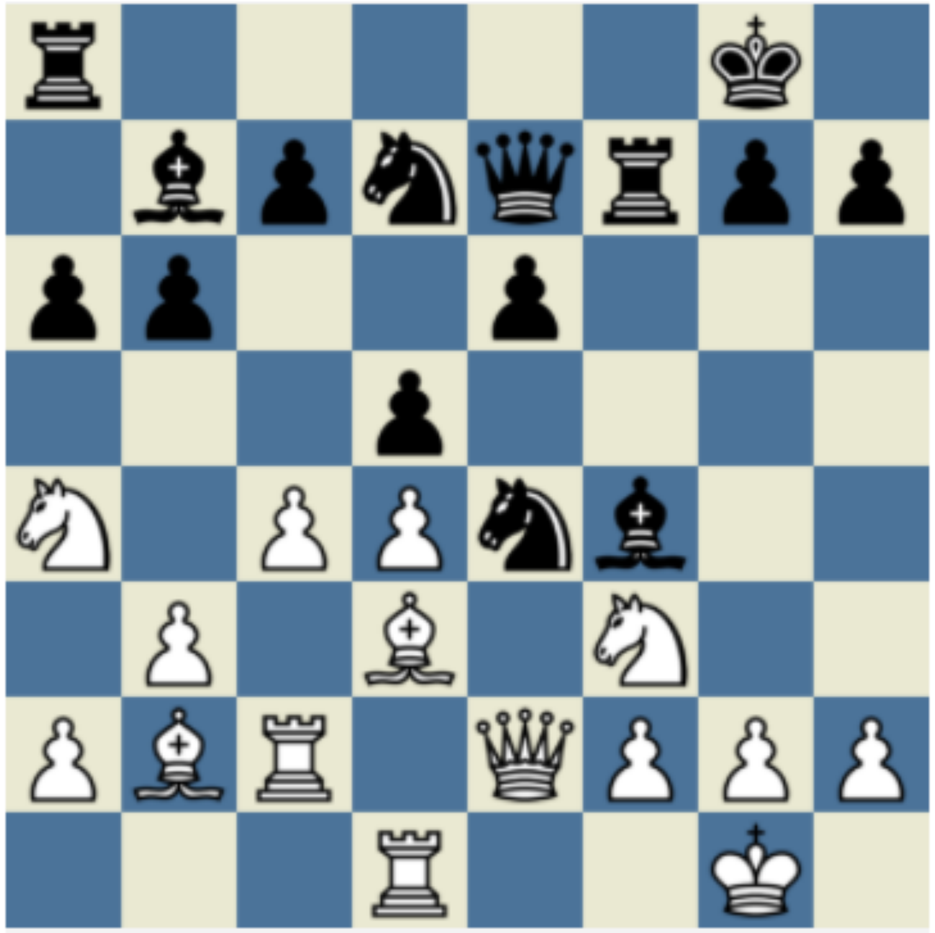




But it's easy to verify a solution that someone gives you.

you can guess!

An Even Harder Problem



How do I verify that White has a guaranteed win in this position?

Course Themes

Automata Theory

What is a machine? What is a computation?

Computability Theory

What can be computed?

What cannot be computed by any machine?

How do I design a virus?

Complexity Theory

Why are some things harder to compute than others?

Why is multiplication easy, but factoring numbers seems hard?

Why is it sometimes hard to solve a problem, but easy to verify a solution?

Why is it hard to verify solutions to other harder problems?

Develop mathematical formalisms to address these questions.

Course Outline

Models of Computation

- Finite State Machines and Regular Languages (Chapter 1)
- Pushdown Automata and Context-Free Languages (Chapter 2)
- Turing Machines: universal computation (Chapter 3)

Computability

- Decidability and Undecidability (Chapters 4,5,6)

Complexity

- Time and Space Complexity, NP-completeness (Chapters 7,8)
- Hierarchy Theorems (Chapter 9)
- Randomness, Interactive Proof Systems, Approximation Algorithms (Chapter 10)

Assignment

Read Chapters 0 and 1 of the textbook.

Solve every exercise and problem in Chapter 0.

Work on your own. Seek my help as needed.

You do not need to submit your solutions.

Scoring a Game of Tennis

Scoring rules:

Each player starts with a score of 0 (“love”).

Winning points advance score from 0 to 15 to 30 to 40.

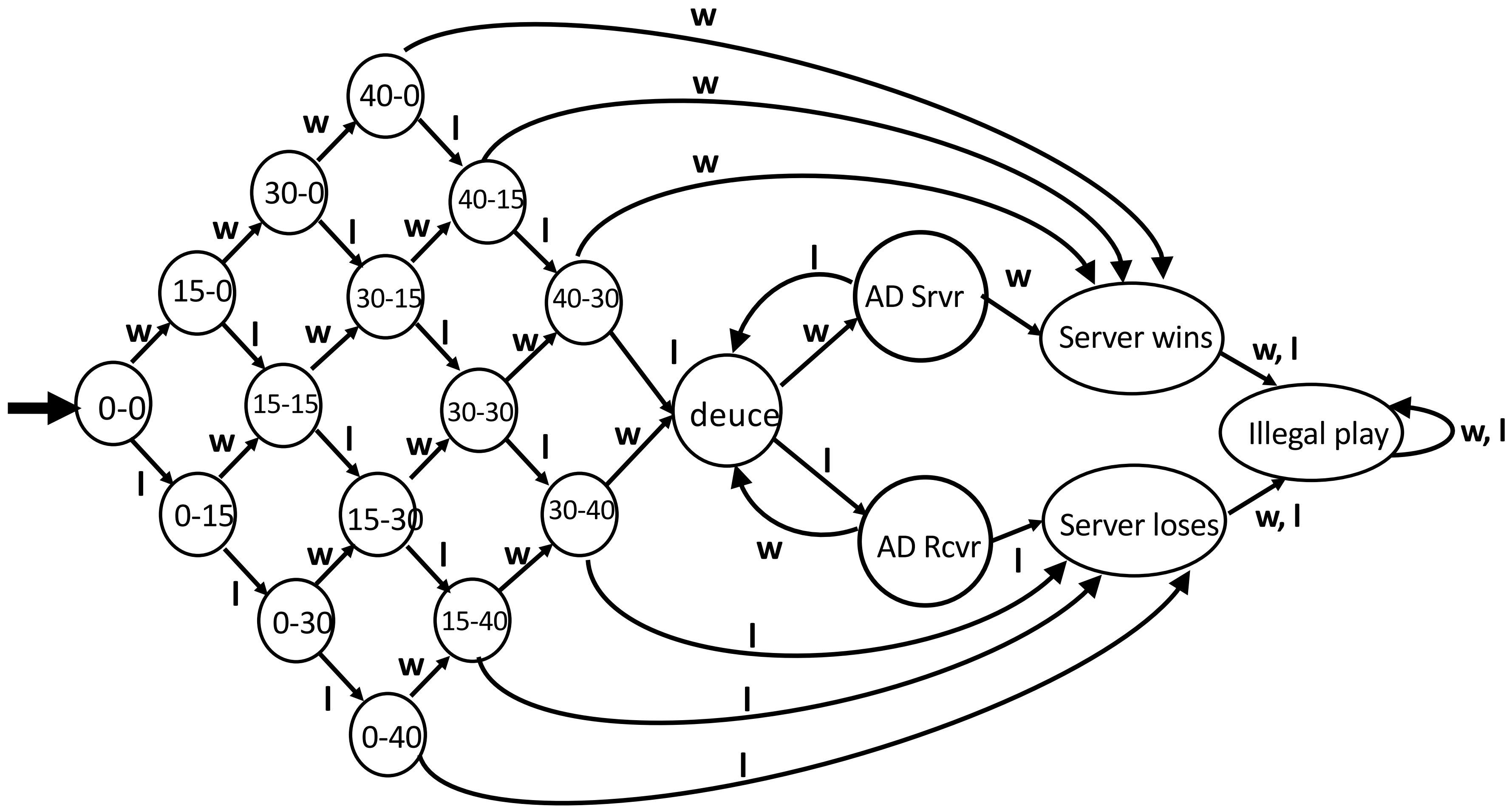
If both players reach a score of 40 the game is tied at deuce.

The player who wins the point following deuce gets an “advantage.”

The player with an advantage loses a point, the game is back at deuce.

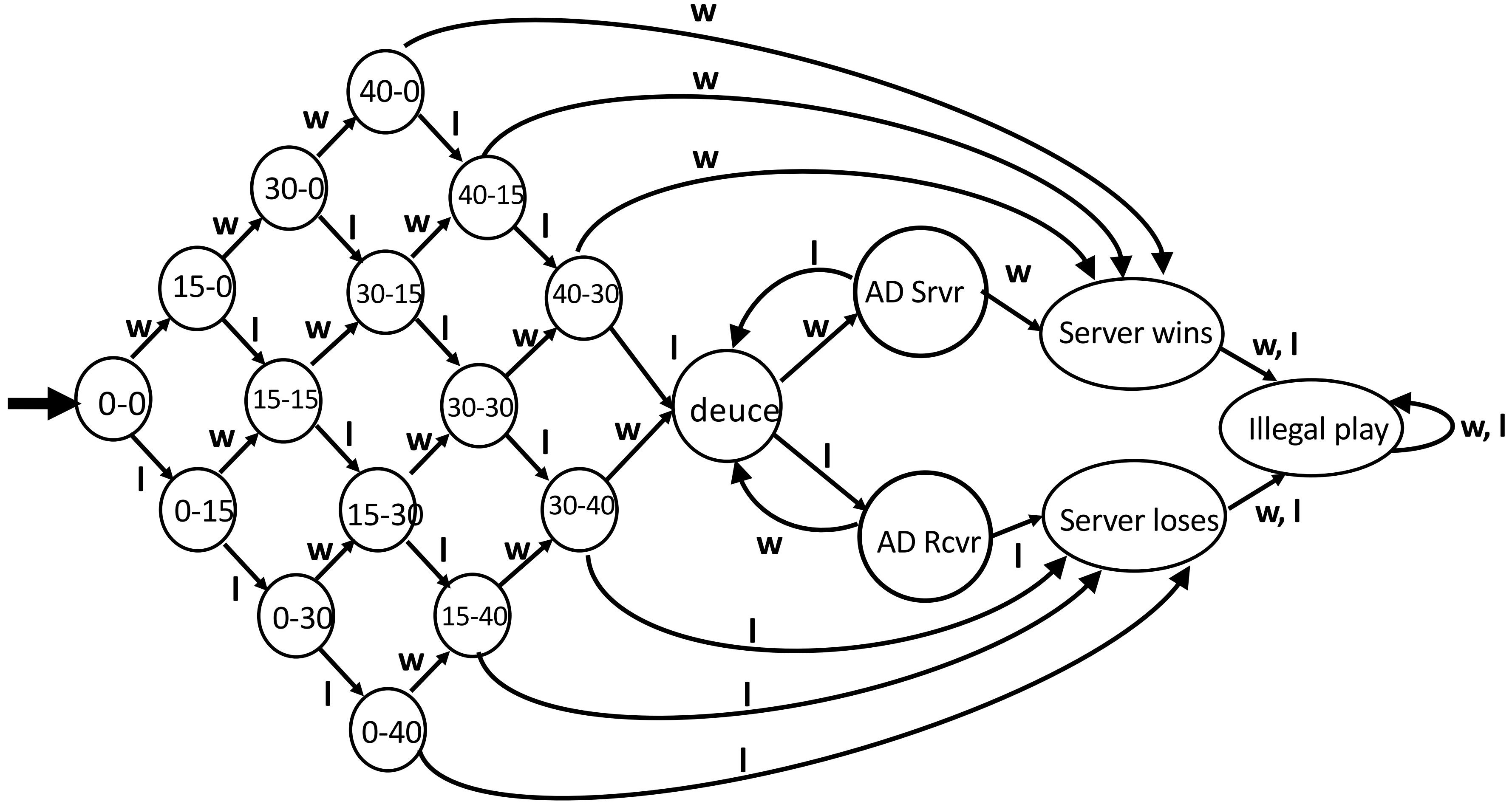
The player with an advantage who wins a point wins the game.

A Finite State Model for a Game of Tennis



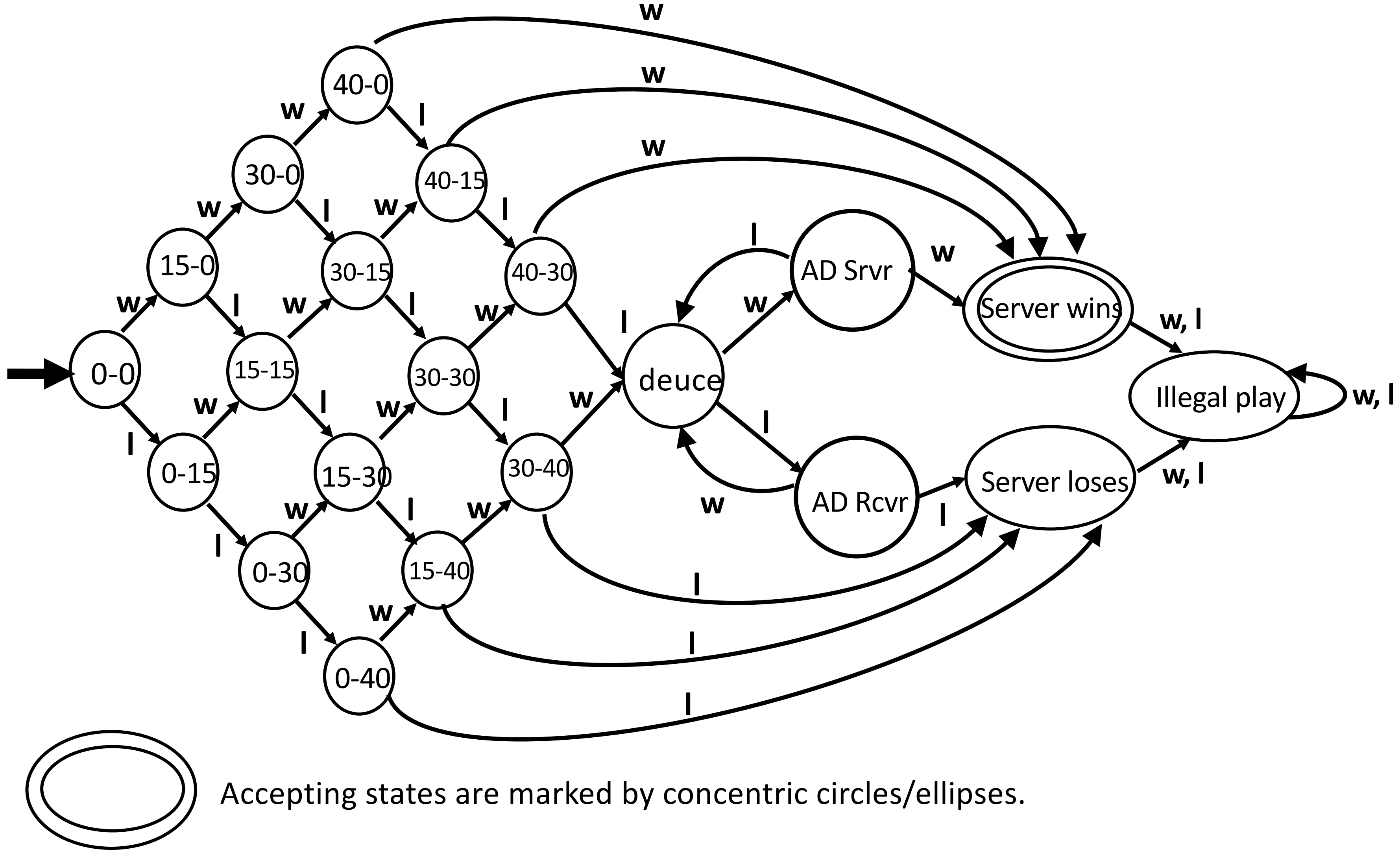
: starting state **w:** server wins point **l:** server loses point

Winning Sequences for the Server



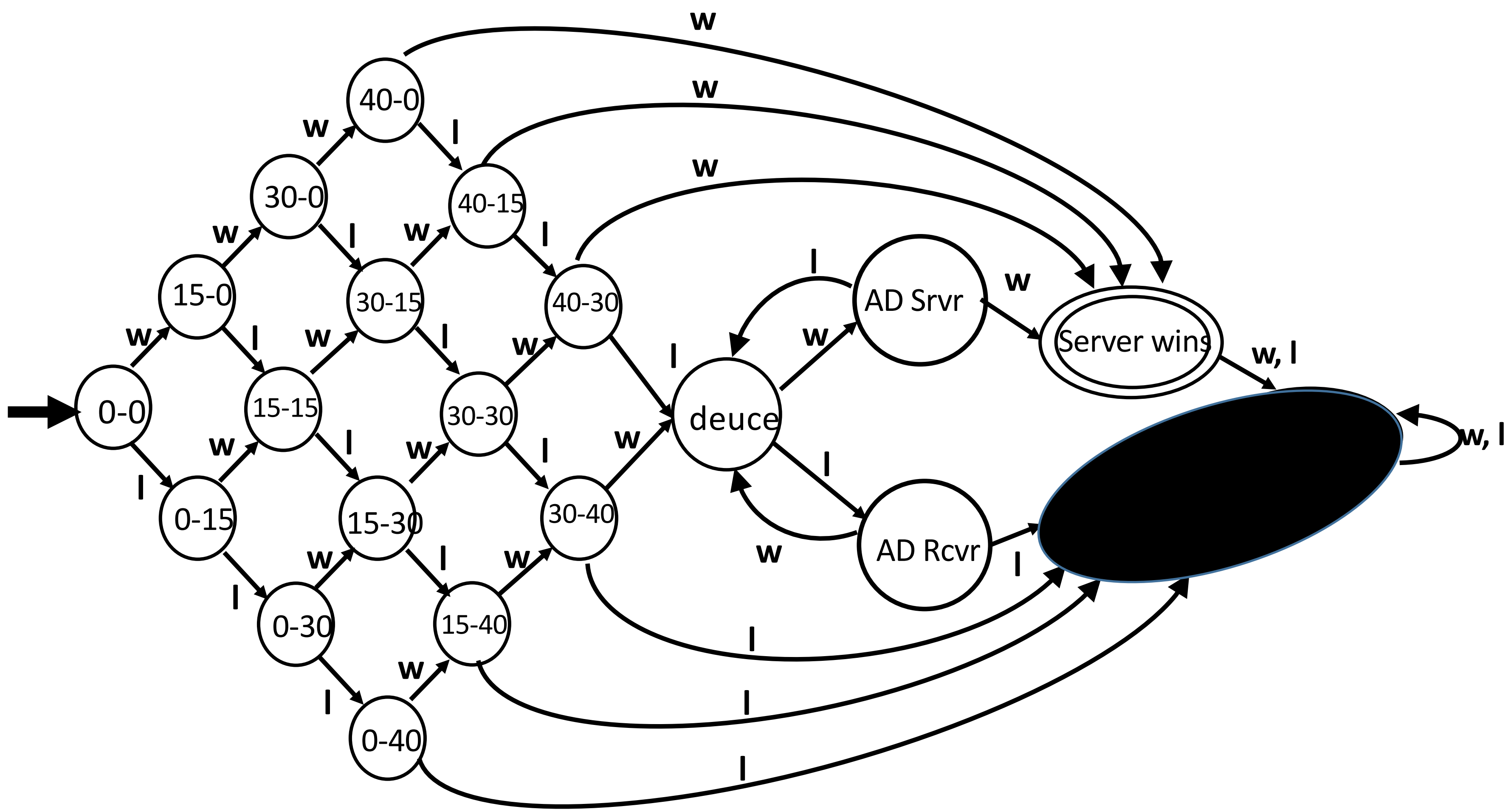
Is **wwwllllw** a winning sequence for the server? What about **lllwwl**? **llwlwwwllwwllwllw**?

Recognizing only Server Wins



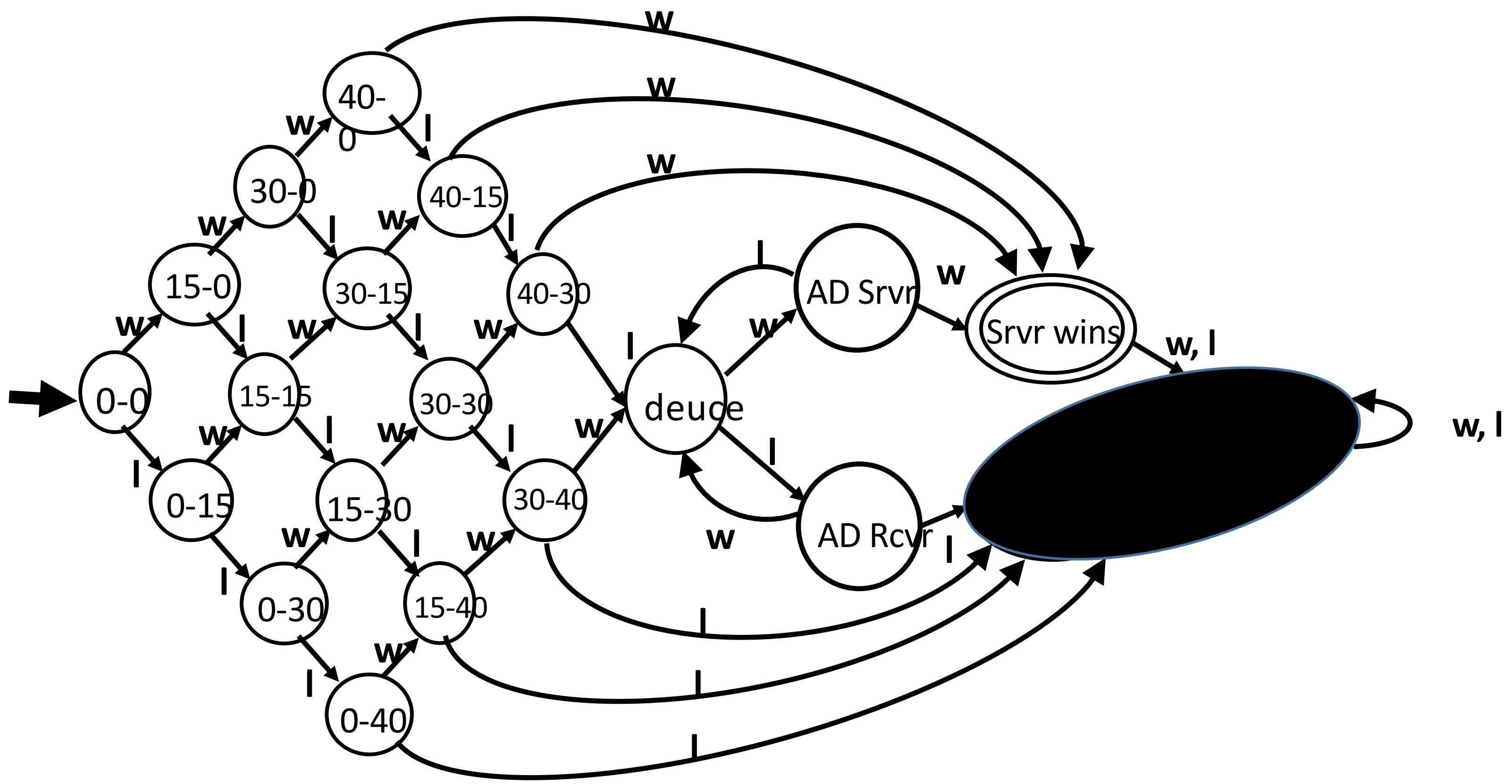
Accepting states are marked by concentric circles/ellipses.

Reducing the Number of States



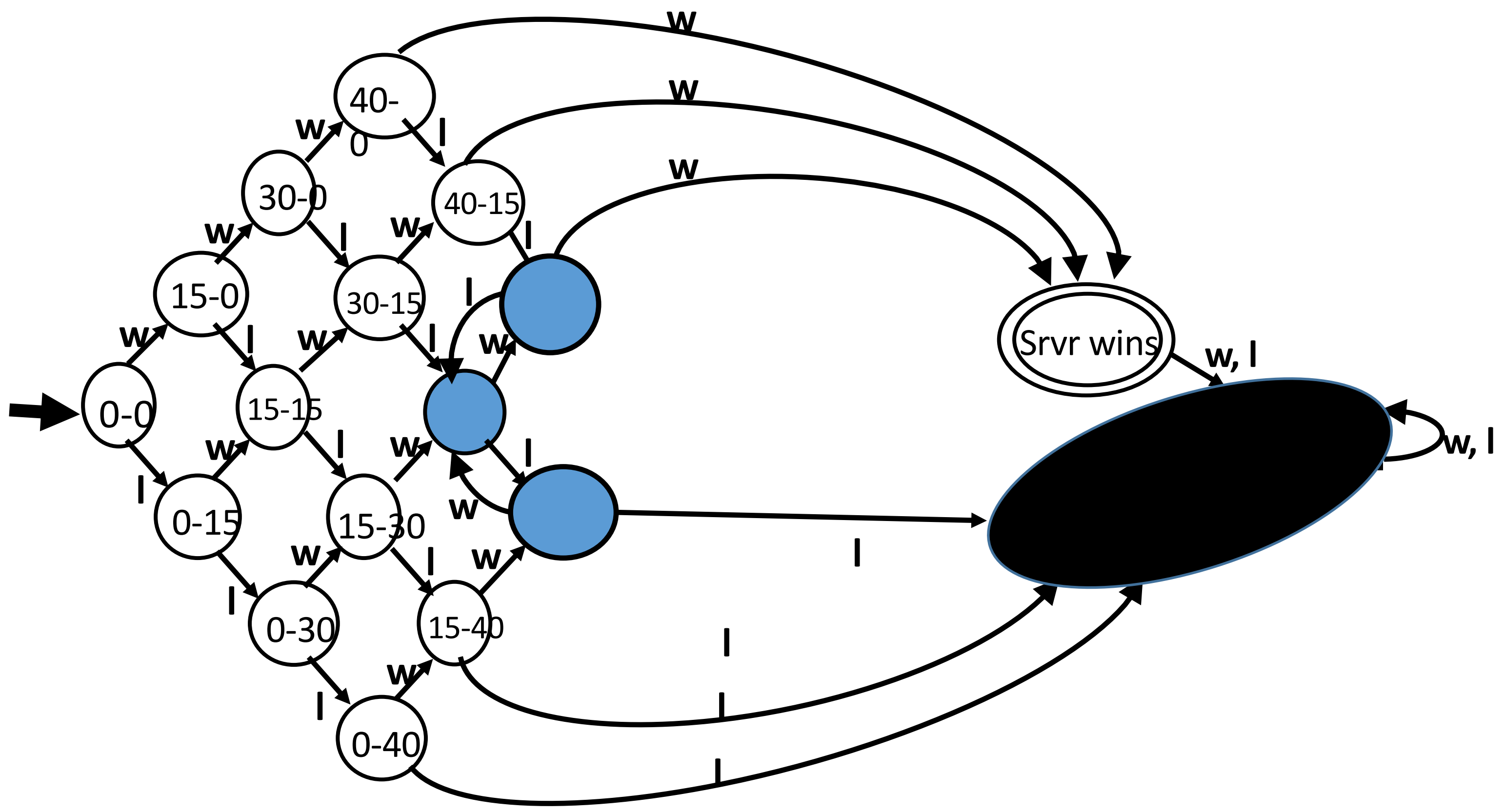
Can we make further reductions?

Reducing the Number of States



A sequence from 40-30 wins *iff* it wins from AD Srvr.
A sequence from 30-40 wins *iff* it wins from AD Rcvr.
A sequence from 30-30 wins *iff* it wins from deuce.
We can merge equivalent states!

The Reduced Finite State Model



Every winning sequence for the server in a real tennis game is a winning sequence above.
Every non-winning sequence for the server in a real game is a non-winning sequence above.
The finite state model captures *exactly* the set of winning sequences for the server.
It does not mimic the real game, nor does it track scores faithfully.

Where are we headed?

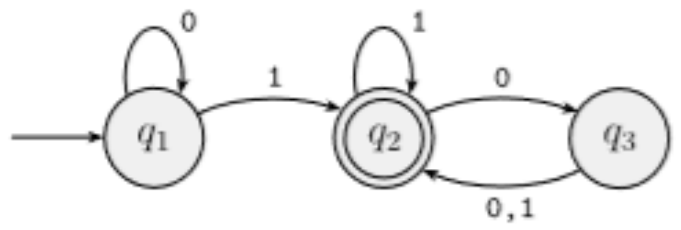
Given a finite state machine diagram, what language does it accept?

Given a language, can it be recognized by a finite state machine?

If yes: Design an FSA for the language

Minimize the number of states of the FSA

Finite State Machines: Example


$$\begin{tikzcd}
\arrow[r] & q_1 \arrow[loop above, "0"] \arrow[r, bend left, "1"] & q_2 \arrow[loop above, "1"] \arrow[r, bend left, "0"] & q_3 \arrow[l, bend left, "{0,1}"]
\end{tikzcd}$$

The set of ***states*** is $\{q_1, q_2, q_3\}$
q_1 is the ***start state***
q_2 is an ***accept state***
The ***alphabet*** of symbols is $\{0,1\}$

From each state there is a state ***transition*** for every symbol in the alphabet.
The machine ***accepts*** a string if, beginning with the start state and following the transitions corresponding to successive symbols in the string, the machine ends in an accept state. If not, the machine rejects the string.

000	Reject
001	Accept
100	Accept
1001	Accept
10010	Reject
100100	Accept

What is the language – the set of strings – accepted by this machine?

Getting Formal

A finite automaton M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

Q is the finite set of states

Σ is the finite alphabet of symbols

$\delta: Q \times \Sigma \to Q$ is the state transition ***function***

q_0 is the start state

F is the set of accept states

Formal Definition 2

A finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ ***accepts*** a string $w = w_1 w_2 \cdots w_n$ if there is a sequence $r_0, r_1, \ldots, r_n$ of states in Q such that:

1. $r_0 = q_0$, (start in the start state)
2. $r_{i+1} = \delta(r_i, w_{i+1})$ for $0 \leq i < n$, and (every transition is legal)
3. $r_n \in F$. (the final state is an accept state)

A Convenient Definition

Suppose we are given a DFA $M = (Q, \Sigma, \delta, q_0, F)$

In state q, if the next input symbol is a, the next state is $\delta(q, a)$.

What is the final state when we start in q and process string w?

$$\Delta(q, w)$$

Define: $\Delta\colon \mathrm{Q} \times \Sigma^* \rightarrow Q$ recursively as follows:

$$\Delta(q, \epsilon) = q$$

$$\Delta(q, ax) = \Delta(\delta(q, a), x), \qquad a \in \Sigma, x \in \Sigma^*$$

$$M\ accepts\ w\ iff\ \Delta(q_0, w) \in F$$

Terminology

The ***language*** of finite automaton M is the set of strings accepted by M.

$$L(M) = \{w\colon M\ accepts\ w\}$$

We say that M ***recognizes*** A if $L(M) = A$.

A language A is ***regular*** if it is recognized by a finite automaton.