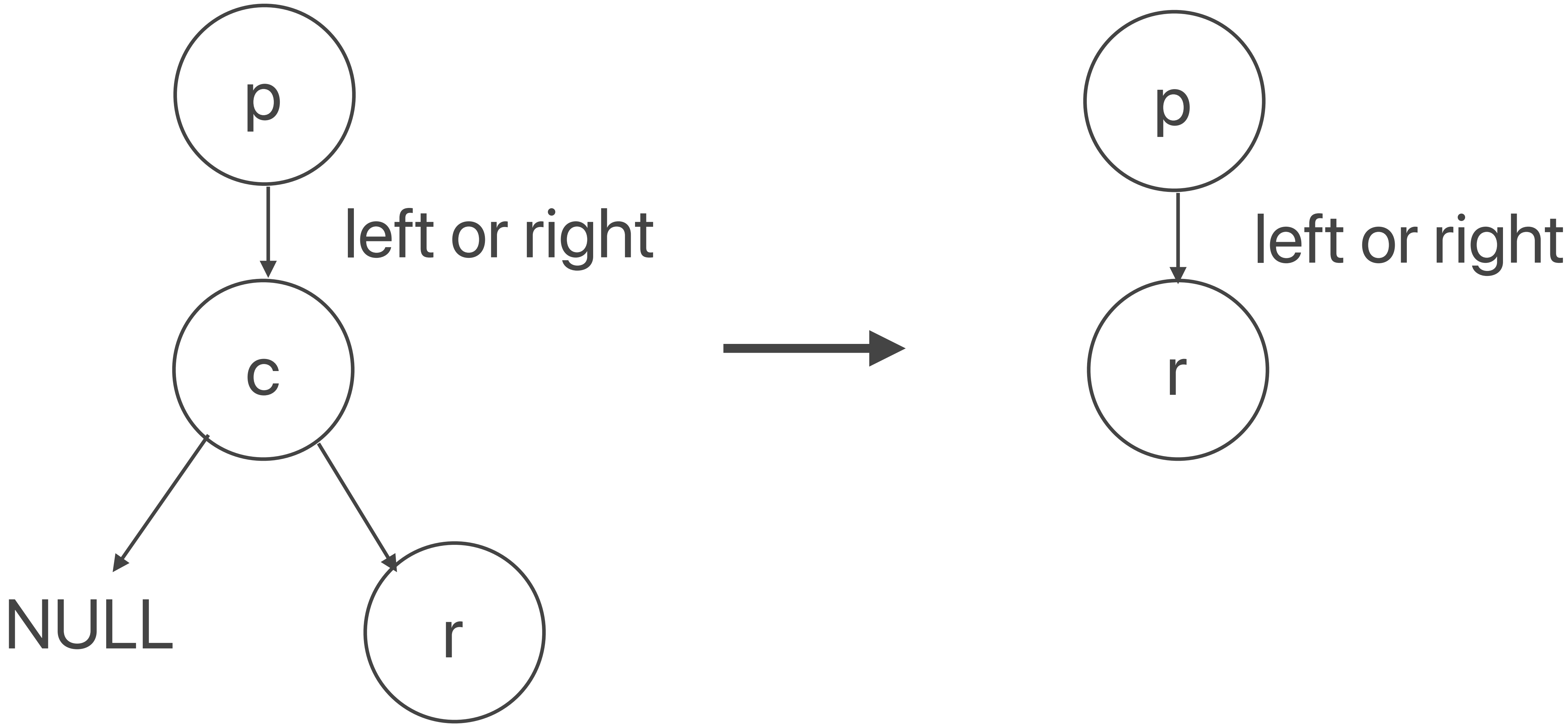
Binary Search Trees: Deleting a Node

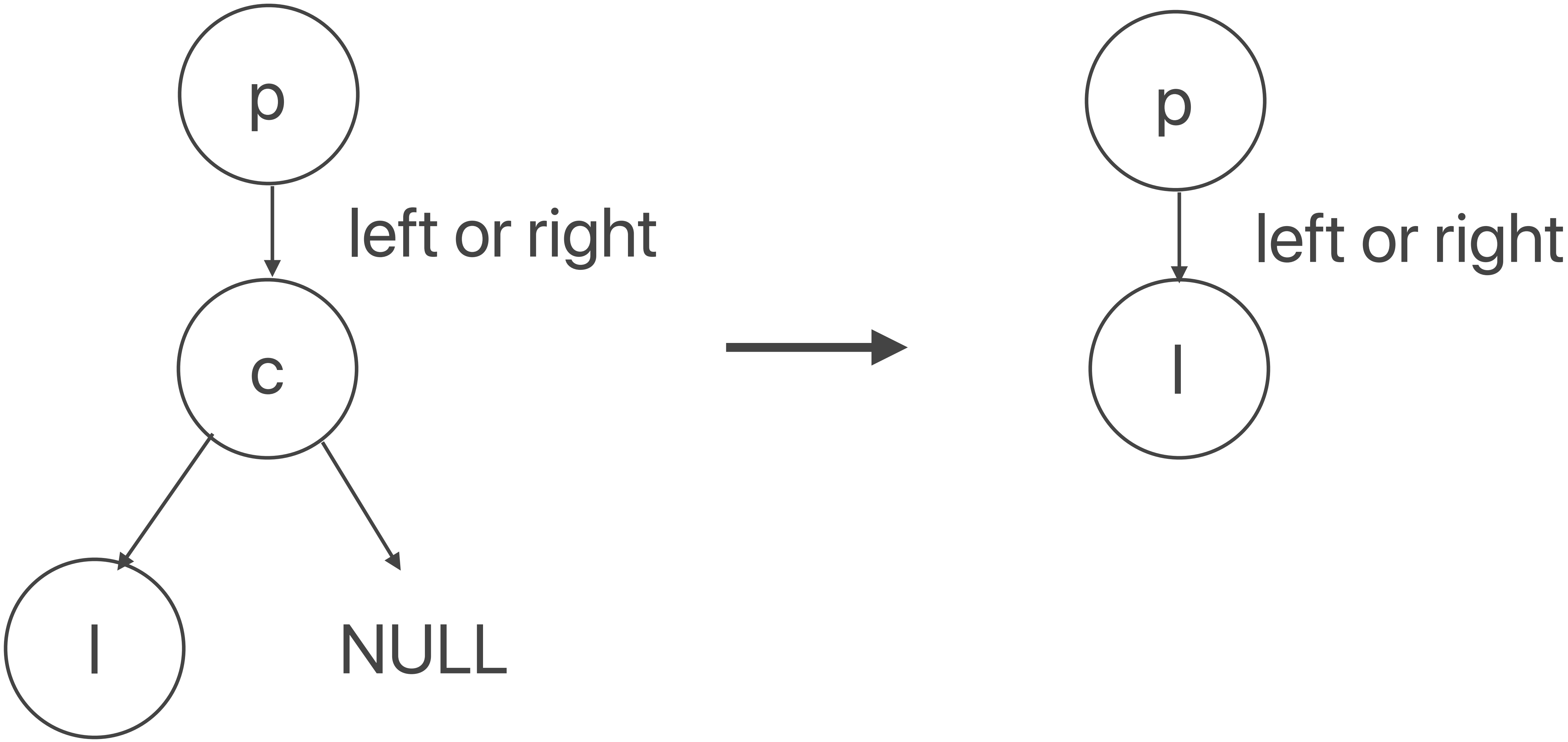
Baochun Li
Department of Electrical and Computer Engineering
University of Toronto

Deleting a node from a binary search tree

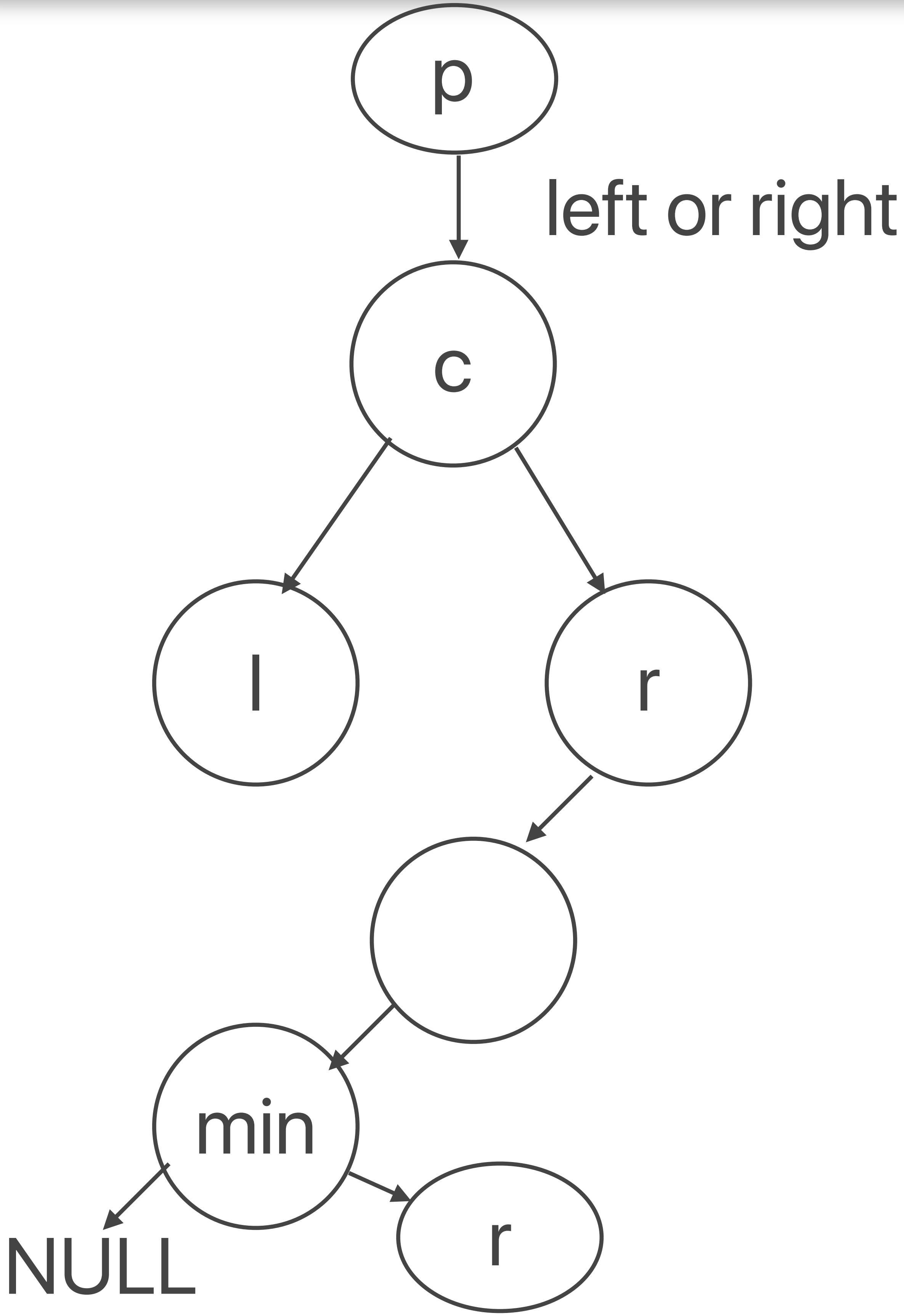
Case #1: Deleting a node with right child only



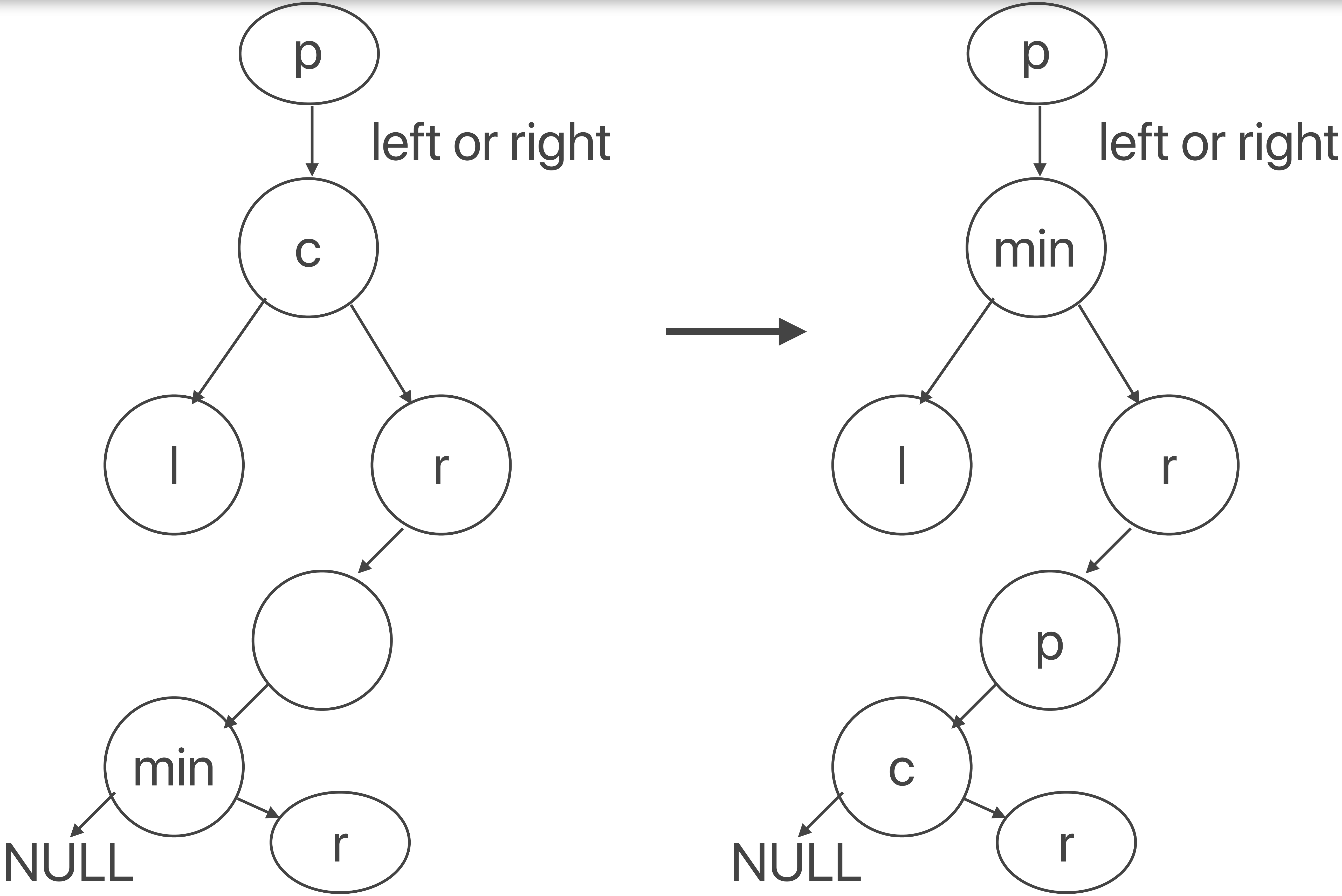
Case #2: Deleting a node with left child only



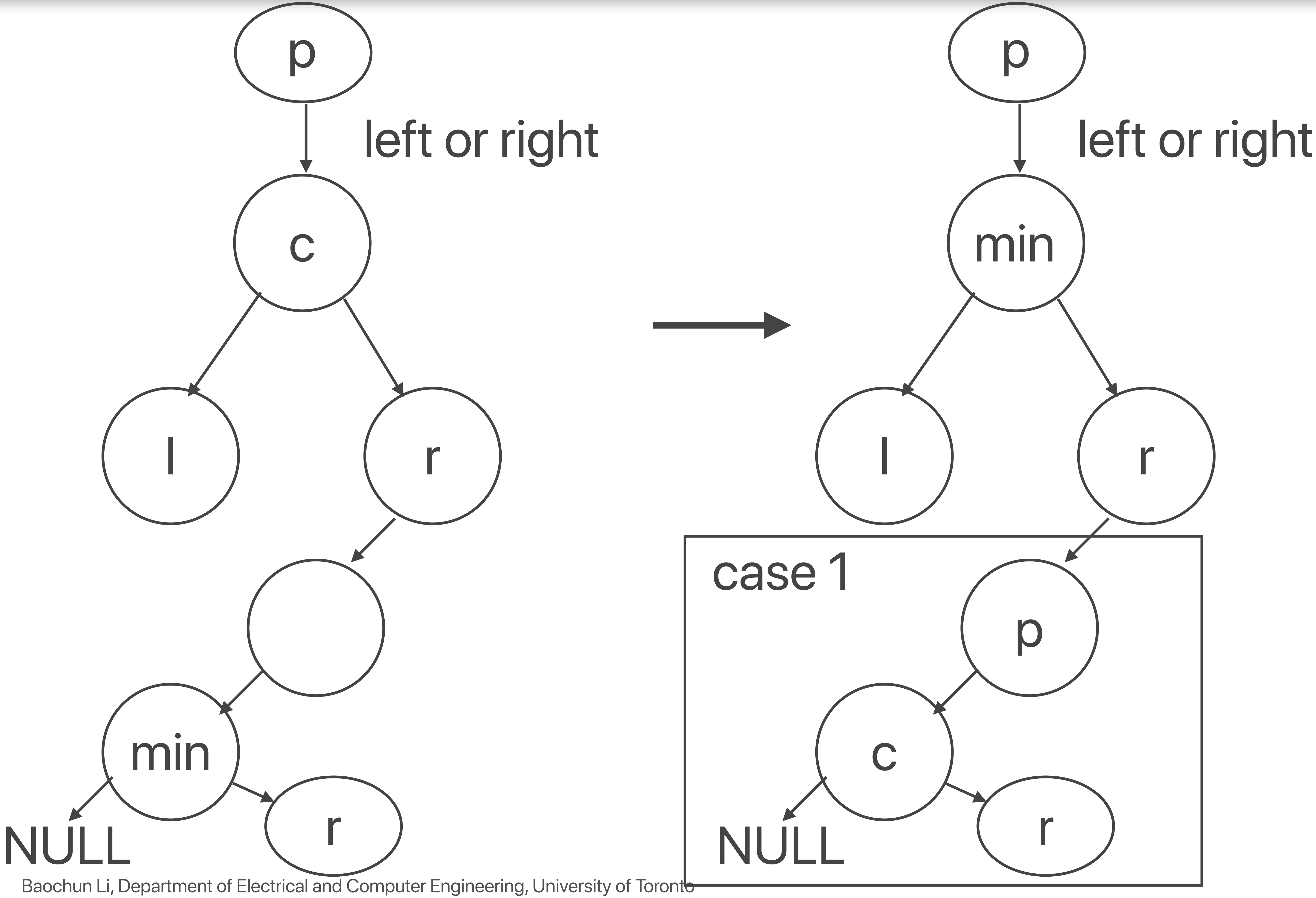
Case #3: Deleting a node with both children



Case #3: Deleting a node with both children



Case #3: Deleting a node with both children



Search for the node first

```
bool deleteNode(BSTree *tree, int value) {
  Node *parent = NULL;
  Node *current = tree -> root;

  // Search for the item to be deleted,
     remembering its parent
  while (current != NULL && current -> data !=
         value) {
    parent = current;
    if (value < current -> data)
      current = current -> left;
    else
      current = current -> right;
  }
```

If we cannot find the item or root is NULL

```
// If we cannot find the item, return
if (current == NULL)
  return false;
```

If the node has two children

```
// If the node to be deleted has two children
if (current -> left != NULL && current ->
    right != NULL) {
  parent = current;
  Node *successor = current -> right;

  // Look for the node with the minimum value
     in the right sub-tree
  while (successor -> left != NULL) {
    parent = successor;
    successor = successor -> left;
  }

  current -> data = successor -> data;
  current = successor;
}
```

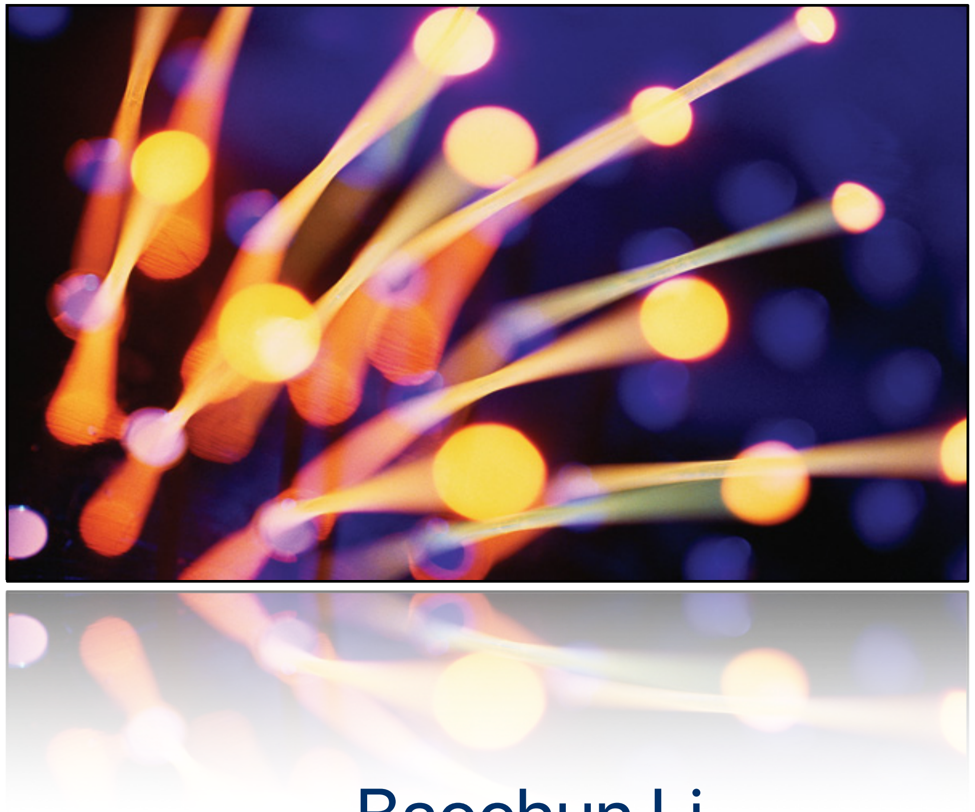
If the node has a right child only

```
// If the node to be deleted has a right child only
if (current -> left == NULL) {
  // If the root is to be deleted
  if (parent == NULL) {
    Node *rightSubtree = current -> right;
    free(current);
    tree -> root = rightSubtree;
    return true;
  }

  if (parent -> left == current)
    parent -> left = current -> right;
  else
    parent -> right = current -> right;
  free(current);
  return true;
}
```

If the node has a left child only

```
// If the node to be deleted has a left child only
if (current -> right == NULL) {
  // If the root is to be deleted
  if (parent == NULL) {
    Node *leftSubtree = current -> left;
    free(current);
    tree -> root = leftSubtree;
     return true;
  }

  if (parent -> left == current)
    parent -> left = current -> left;
  else
    parent -> right = current -> left;
  free(current);
  return true;
}
return true;
```

APS 105S: Review and Outlook



Baochun Li
Department of Electrical and Computer Engineering
University of Toronto

Online teaching evaluations — to be closed on April 12 20% now! :(

APS 105: Final Exam Coverage

All the material covered in our lectures

- Material not covered in our lectures will not be in the exam

Types of questions that may appear in the exam

- Short answer questions
- Finding compile-time errors in C programs
- Tracing the output of C programs
- Programming questions

Function prototypes are given to you in all the programming questions — but you can define your own as you wish

Programming Questions: What can they be?

- Binary search trees
- Linked lists
- Strings
- Searching and sorting
- Basic structures before strings, including arrays, loop structures, if statement, and primitive types
- Any of the above may involve recursion

Strategies preparing for the exam

Come to Exam Jam and ask questions

Friday, April 12, 10 AM – 1 AM, room to be announced

Try out programming questions in the textbook

Type your solution into the computer to see if it is correct

Timing is important

try coming up with a solution within timing constraints

Strategies preparing for the exam

Take advantage of the 2018 final exam

- Try to take the exam within 150 minutes with pen and paper
- Compare your solutions with the sample solutions
- Find out your weak areas to reinforce
- Do not read the solutions the first time you see the exam

Read the textbook *and* lecture notes carefully

- Pay full attention to the key concepts

Think about strategies to come up with an idea in a programming question

How to come up with an idea quickly?

Regular cases —

How a node can be inserted into a sorted, non-empty list?

How do we handle the **recursive case** to make the problem smaller?

Work through an example

The example may even be provided to you in the exam question

Use the back of the previous page as scratch paper

Special cases —

Example: Empty list, list with only one node, insertion point at the end

Try out your existing solution to see if it works already in special cases

If not, add a few lines of code to handle the special cases

Base case(s) in a recursion question: once you know what they are, not hard to handle

Strategies while taking the exam

Read the exam questions (including page 1) carefully

Each word is carefully chosen, refined and hand-crafted over many drafts with efforts from all the instructors

Do not need to fill all the empty space

Write legibly

Make sure that you do not make a mistake in easier questions

They are there to help you

But don’t under-estimate: some questions may be harder than you think

Plan the use of your time well — bring a (non-smart) watch

Next Steps

(or the journey to $200/hr contracts)

Programming Paradigms

Fundamental styles of computer programming

- Ways to think about how complex functionality can be broken into smaller pieces and implemented

APS 105 is all about procedural programming

- Divide functionality into *functions* (or *procedures*)
- Each contains a series of computational steps to be carried out *consecutively*
- Best example: the C programming language
- Example scenarios used —
  - The baseband of a mobile phone
  - The kernel of an operating system
  - Compilers for programming languages

New Programming Paradigm: Object-Oriented

Divide the world of functionality into *objects*

Each object include

- some *instance variables:* just like a "struct" in C
- some *methods* to operate on these instance variables
- Analogy: A *recipe* in a cookbook needs *ingredients* and *methods* to mix them together

Important programming languages

- Swift: Mac and iOS development
- Java: Android development
- Python: Machine learning

We have mentioned it before!

A **linear list** includes nodes (as instance variables) and a few methods as “interface” to the outside world

Interface can be completely detached from the implementation

We can change the implementation from an array to a linked list, but still keep the same interface: “insert”, “delete”, “create”

Great way to have a team working together, or for a company to release a software “framework” for other developers to use

A good way to represent a complex world

Objects represent a natural way to look at complexity in the world of software

A Graphical User Interface (GUI) can consist of many *objects*

Buttons, dialog boxes, tool bars, menu items: all can be objects

When you left-click, right-click or double-click a button: the functions handling them can be part of the button object itself

A "class" is a "template" to define an *instance object*

For example, a class can be defined for all button *instance objects*

A "method" is what an object instance *can do*

A button object: *display*

How do we ask an object to do something?

In Swift —

```
var pattern =
searchTerm.replacingOccurrences(of:
" ", with: "\n")
```

That’s it for this course

Thank you