TeachingSpace Documentation

Release 0.1

Zhiqiang Ren

June 24, 2015

Contents

Courses:

CHAPTER 1

CS455/655 Introduction to Computer Networks

1.1 General Information

This is an auxiliary webpage for the class. The main webpage is https://sites.google.com/site/alex2ren/cs455-655-introduction-to-computer-networks-fall-2013-tf.

1.2 How to use WireShark in the Undergraduate Lab.

Due to security issue, you cannot run WireShark directly on the machines in the Undergraduate Lab (EMA 304). Though you are recommended to install WireShark on your own machines, we still provide a way for you to try WireShark in the Undergraduate Lab. Each Linux machine in the Undergraduate Lab has a virtual machine installed. Login into the Linux machine and turn on the virtual machine, and then you can run WireShark inside the virtual machine. Please refer to the following link for information about how to use the virtual machine.

> http://www.cs.bu.edu/tiki/tiki-index.php?page=CS455+and+CS655+-+VMPlayer+HowTo

You may see some warning during the process of booting the virtual machine. Simply choosing *OK* or *Remind Me Later* is fine. The virtual machine is configured in such a way that it is in a private network consisting of only the virtual machine and the Linux machine hosting it. You can still get access to the internet inside the virtual machine via NAT. You will get more familiar with these concepts as the course goes on.

1.3 Labs

WireShark has been installed on all the Windows machines in the Instructional Lab (EMA 302). It is only usable during the period of the lab session.

1.3.1 Lab 03:

Keywords

- TCP / APP
- Blocking / Nonblocking
- Multi-threading / Select
- Reliable / Guarantee

- Succeed / Fail
- Error / Exception

Code Example

- `echoclient.py`

```
#!/usr/bin/env python


import socket
import sys
import struct

def connect(addr, port):
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((addr,port))
        return (0, s)
    except socket.error, (value,message):
        if s:
            s.close()
        return (-1, message)


def sendMsg(conn, msg):
    if msg=="":
        return (-1, "Cannot send empty string.")
    size = 1024
    try:
        conn.sendall(msg)
        return (0, None)
    except socket.error, (value,message):
        conn.close()
        return (-1, message)

def sendNum(conn, num):
    msg = struct.pack("<i", num)
    size = 1024
    try:
        conn.sendall(msg)
        return (0, None)
    except socket.error, (value,message):
        conn.close()
        return (-1, message)

def recvMsg(conn):
    size = 1024
    try:
        data = conn.recv(size)
        return (0, data)
    except socket.error, (value,message):
        conn.close()
        return (-1, message)

def close(conn):
    try:
```

```
        conn.close()
        return (0, None)
    except socket.error, (value,message):
        return (-1, message)
```

- echoserver.py

```
#!/usr/bin/env python


import socket
import sys

host = ''
backlog = 5

from echoserver_worker import ServerWorker

class Server:
    def main(self):
        port = 0
        try:
            port = int(sys.argv[1])
        except:
            print "Usage: Server.py Server_port"
            port = 50001
            print "Choose default port (", port, ")"


        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
        s.bind((host,port))
        s.listen(backlog)

        while True:
            clientInfo = {}
            (newSocket, clientAddress) = s.accept()
            clientInfo['socket'] = (newSocket, clientAddress)
            ServerWorker(clientInfo).run()

if __name__ == "__main__":
    (Server()).main()
```

- echoserver_worker.py

```
import sys, traceback, threading, socket
import struct
import random

size = 1024

class ServerWorker:

    def __init__(self, clientInfo):
        self.clientInfo = clientInfo
        self.connSocket = self.clientInfo['socket'][0]
        self.clientAddr = self.clientInfo['socket'][1]

    def run(self):
```

```
        print "running worker"
        threading.Thread(target=self.work0).start()

    def work0(self):
        print self.clientAddr, "has connected."
        try:
            while 1:
                data = self.connSocket.recv(size)
                print self.clientAddr, "sends:", data
                if data == "":
                    break
                self.connSocket.sendall(data)
        except socket.error, (code, message):
                    print "error processing client", self.clientAddr
        finally:
            print "work is done"
            if self.connSocket:
                self.connSocket.close()
```

- echoclient_gui.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
Author: Zhiqiang Ren
last modified: Sep. 2013
"""

import Tkinter as tk

from Tkinter import Frame, Button, Entry, Text, Label, Message

import echoclient as ec
import struct

class Client(Frame):

    def __init__(self, parent):
        Frame.__init__(self, parent, relief=tk.RAISED, borderwidth=10)

        self.parent = parent

        self.initUI()

    def initUI(self):
        self.parent.title("Client")
        frame = Frame(self, relief=tk.RAISED, borderwidth=1)

        # The width of the first column gets almost no change.
        frame.columnconfigure(0, pad=10, weight=1)
        frame.columnconfigure(1, pad=10, weight=1000)

        lbl_addr = Label(frame, text="Address")
        lbl_addr.grid(row=0,column=0, sticky=tk.W+tk.S)

        lbl_port = Label(frame, text="Port")
        lbl_port.grid(row=0,column=1, sticky=tk.W+tk.S)
```

```
        ent_addr = Entry(frame, width=15)
        ent_addr.grid(row=1,column=0, sticky=tk.W+tk.N)

        ent_port = Entry(frame, width=6)
        ent_port.grid(row=1,column=1, sticky=tk.W+tk.N)

        lbl_msg = Label(frame, text="Input Message", anchor=tk.E)
        lbl_msg.grid(row=2,column=0, sticky=tk.W+tk.S)

        ent_msg = Text(frame, width=30, height=4)
        ent_msg.grid(row=3,column=0, columnspan=2, sticky=tk.W+tk.E+tk.N) # sticky can be used to exp

        lbl_num = Label(frame, text="Input Number")
        lbl_num.grid(row=4,column=0, sticky=tk.W+tk.S)

        ent_num = Entry(frame, width=6)
        ent_num.grid(row=5,column=0, sticky=tk.W+tk.N)
        # ======================

        ret_indicator = tk.StringVar()
        ret_indicator.set("Result")
        lab_res = Label(frame, textvariable=ret_indicator)
        lab_res.grid(row=6,column=0, sticky=tk.W+tk.S)

        var_res = tk.StringVar()
        msg_res = Message(frame,textvariable=var_res, width=500)
        msg_res.grid(row=7,column=0, columnspan=2,sticky=tk.W+tk.E+tk.N)

        frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=1)

        def connect():
            self.addr = ent_addr.get()
            portStr = ent_port.get()

            if self.addr == "":
                self.addr = "localhost"

            if portStr == "":
                self.port = 50000
            else:
                self.port = int(portStr)

            (ret, info) = ec.connect(self.addr, self.port)
            if ret == 0:
                ret_indicator.set("Connection Succeeded")
                self.conn = info
                var_res.set("")
            else:
                ret_indicator.set("Connection Failed")
                var_res.set(info)

        def sendMsg():
            msg = ent_msg.get("0.0", tk.END)[0:-1]
            print "msg to be sent is: " + repr(msg)
            (ret, info) = ec.sendMsg(self.conn, msg.encode('utf-8'))
            if ret == 0:
                ret_indicator.set("Send Succeeded")
```

```
                var_res.set("")
            else:
                ret_indicator.set("Send Failed")
                var_res.set(info)

        def sendNum():
            msg = ent_num.get()
            print "msg to be sent is: " + repr(msg)
            (ret, info) = ec.sendNum(self.conn, int(msg))
            if ret == 0:
                ret_indicator.set("Send Succeeded")
                var_res.set("")
            else:
                ret_indicator.set("Send Failed")
                var_res.set(info)

        def recvMsg():
            (ret, info) = ec.recvMsg(self.conn)
            if ret == 0:
                ret_indicator.set("Receive Succeeded")
            else:
                ret_indicator.set("Receive Failed")
            var_res.set(info)

        def close():
            (ret, info) = ec.close(self.conn)
            if ret == 0:
                ret_indicator.set("Close Succeeded")
                var_res.set("")
            else:
                ret_indicator.set("Close Failed")
                var_res.set(info)

        frame2 = Frame(self, relief=tk.RAISED, borderwidth=1)

        """Buttoms are always in the middle."""
        frame2.columnconfigure(0, pad=10, weight=1)
        frame2.rowconfigure(0, weight=1000)
        frame2.rowconfigure(1, weight=1)
        frame2.rowconfigure(2, weight=1)
        frame2.rowconfigure(3, weight=1)
        frame2.rowconfigure(4, weight=1)
        frame2.rowconfigure(5, weight=1)
        frame2.rowconfigure(6, weight=1000)

        but_conn = Button(frame2, text="Connect", command=connect)
        but_conn.grid(row=1,column=0, sticky=tk.W+tk.E)

        but_send_msg = Button(frame2, text="Send Message", command=sendMsg)
        but_send_msg.grid(row=2,column=0, sticky=tk.W+tk.E)

        but_send_num = Button(frame2, text="Send Number", command=sendNum)
        but_send_num.grid(row=3,column=0, sticky=tk.W+tk.E)

        but_recv = Button(frame2, text="Receive", command=recvMsg)
        but_recv.grid(row=4,column=0, sticky=tk.W+tk.E)

        but_close = Button(frame2, text="Close", command=close)
```

```
        but_close.grid(row=5,column=0, sticky=tk.W+tk.E)

        frame2.pack(side=tk.LEFT, fill=tk.BOTH,expand=1)

        # expand=1 cannot be omitted
        self.pack(fill=tk.BOTH, expand=1)

def main():

    root = tk.Tk()
    # root.geometry("300x200+300+300")
    root.geometry("+500+500")
    app = Client(root)
    root.mainloop()


if __name__ == '__main__':
    main()
```

1.3.2 Lab 04: Wireshark Lab of DNS and HTTP

Part 1: DNS Protocol

Usage of *ipconfig*:

```
ipconfig /all
ipconfig /displaydns
ipconfig /flushdns
```

Resource Records (RRs) in DNS distributed database:

(Name, Value, Type, TTL)

Usage of *nslookup*

```
nslookup [-option] name [server]

nslookup www.eecs.mit.edu
nslookup -type=A www.eecs.mit.edu
nslookup -type=NS mit.edu
nslookup -type=CNAME www.eecs.mit.edu name_of_server
```

Part 2: HTTP Protocol

Web Caching

If-modified-since / Etag

Wireshark Lab: HTTP

Please download `Wireshark_HTTP_v6.1.pdf` for the instruction of this lab session.

Part 3: Miscellaneous

Filters for Wireshark

Find the address of the webserver:

```
http.host=="gaia.cs.umass.edu"
```

Locate specific http connection:

```
ip.addr==xxx.xxx.xxx.xxx && http
```

CHAPTER 2

CS4440/640 Introduction to Artificial Intelligence

2.1 General Information

This is an auxiliary webpage for the class. The main webpage is https://sites.google.com/site/alex2ren/cs440-640-introduction-to-artificial-intelligence-spring-2014-tf.

2.2 Discussion Session

2.2.1 Lab 01: Python and Unit Test

Unit Test in Python

Module: unittest.py

Code Example

- calc.py

```
"""
NAME: Zhiqiang Ren
Discussion Session 1
CS440 / CS640 Artificial Intelligence
"""

class Calculator(object):
    """A simple calculator for integers."""

    def __init__(self, x):
        self.x = x

    def add(self, x):
        ret = self.x + x
        self.x = x
        return ret

    def sub(self, x):
        ret = self.x - x
        self.x = x
```

```
        return ret

    def mul(self, x):
        ret = self.x * x
        self.x = x
        return ret

    def div(self, x):
        ret = self.x / x
        self.x = x
        return ret
```

- testcalc1.py

```
"""
NAME: Zhiqiang Ren
Discussion Session 1
CS440 / CS640 Artificial Intelligence
"""

from calc import Calculator

import unittest

class Calc0TestCaseAdd(unittest.TestCase):
    def runTest(self):
        calc = Calculator(0)
        assert calc.add(1) == 1, "addition is wrong"


class Calc0TestCaseSub(unittest.TestCase):
    def runTest(self):
        calc = Calculator(0)
        assert calc.sub(1) == -1, "substraction is wrong"


def getTestSuite():
    suite = unittest.TestSuite()
    suite.addTest(Calc0TestCaseAdd())
    suite.addTest(Calc0TestCaseSub())
    return suite


def main():
    runner = unittest.TextTestRunner()
    runner.run(getTestSuite())


if __name__ == '__main__':
    main()
```

- testcalc2.py

```
"""
NAME: Zhiqiang Ren
Discussion Session 1
CS440 / CS640 Artificial Intelligence
"""

```

```
from calc import Calculator

import unittest

class Calc0TestCase(unittest.TestCase):
    def setUp(self):
        self.calc = Calculator(0)
    def tearDown(self):
        self.calc = None

    def testAdd1(self):
        assert self.calc.add(1) == 1, "addition is wrong"

    def testAdd2(self):
        assert self.calc.add(2) == 2, "addition is wrong"

    def testSub(self):
        assert self.calc.sub(1) == -1, "substraction is wrong"

def getTestSuite():
    suite = unittest.TestSuite()
    suite.addTest(Calc0TestCase("testAdd1"))
    suite.addTest(Calc0TestCase("testAdd2"))
    suite.addTest(Calc0TestCase("testSub"))
    return suite


def main():
    runner = unittest.TextTestRunner()
    runner.run(getTestSuite())


if __name__ == '__main__':
    main()
```

- testcalc3.py

```
"""
NAME: Zhiqiang Ren
Discussion Session 1
CS440 / CS640 Artificial Intelligence
"""

from calc import Calculator

import unittest

class Calc0TestCase(unittest.TestCase):
    def setUp(self):
        self.calc = Calculator(0)
    def tearDown(self):
        self.calc = None

    def testAdd1(self):
        assert self.calc.add(1) == 1, "addition is wrong"

    def testAdd2(self):
        assert self.calc.add(2) == 2, "addition is wrong"

```

```
    def testSub(self):
        assert self.calc.sub(1) == -1, "substraction is wrong"

class Calc1TestCase(unittest.TestCase):
    def setUp(self):
        self.calc = Calculator(1)
    def tearDown(self):
        self.calc = None

    def testAdd1(self):
        assert self.calc.add(1) == 2, "addition is wrong"

    def testAdd2(self):
        assert self.calc.add(2) == 3, "addition is wrong"

    def testSub(self):
        assert self.calc.sub(1) == 0, "substraction is wrong"


def getTestSuite():
    suite1 = unittest.makeSuite(Calc0TestCase, "test")
    suite2 = unittest.makeSuite(Calc1TestCase, "test")

    suite = unittest.TestSuite()
    suite.addTest(suite1)
    suite.addTest(suite2)
    return suite


def main():
    runner = unittest.TextTestRunner()
    runner.run(getTestSuite())


if __name__ == '__main__':
    main()
```

- testcalc4.py

```
"""
NAME: Zhiqiang Ren
Discussion Session 1
CS440 / CS640 Artificial Intelligence
"""

from calc import Calculator

import unittest

class Calc0TestCase(unittest.TestCase):
    def setUp(self):
        self.calc = Calculator(0)
    def tearDown(self):
        self.calc = None

    def testAdd1(self):
        assert self.calc.add(1) == 1, "addition is wrong"

    def testAdd2(self):
```

```
        assert self.calc.add(2) == 2, "addition is wrong"

    def testSub(self):
        assert self.calc.sub(1) == -1, "substraction is wrong"

class Calc1TestCase(unittest.TestCase):
    def setUp(self):
        self.calc = Calculator(1)
    def tearDown(self):
        self.calc = None

    def testAdd1(self):
        assert self.calc.add(1) == 2, "addition is wrong"

    def testAdd2(self):
        assert self.calc.add(2) == 3, "addition is wrong"

    def testSub(self):
        assert self.calc.sub(1) == 0, "substraction is wrong"


def getTestSuite():
    suite1 = unittest.makeSuite(Calc0TestCase, "test")
    suite2 = unittest.makeSuite(Calc1TestCase, "test")

    suite = unittest.TestSuite()
    suite.addTest(suite1)
    suite.addTest(suite2)
    return suite

if __name__ == '__main__':
    unittest.main()
# python testcalc4.py
# python testcalc4.py Calc0TestCase
# python testcalc4.py Calc1TestCase
# python testcalc4.py getTestSuite
# python testcalc4.py Calc0TestCase.testSub
```

CHAPTER 3

CS320 Concepts of Programming Languages (Spring 2015)

Welcome!

This is the teaching fellow's page for course CS 320 Concepts of Programming Languages. I will post related material here from time to time so that both students and professor can keep track of the updated info about the lab session. This course will use Piazza as the main source of communication among the class. I shall keep the information between these two sites synchronized. Piazza has the higher priority than this website however. It would be a superset of this site, at least it would contain the notice that something new has been added here. Simply check Piazza first before visiting here and no need to worry about missing a thing.

3.1 General Information

- Official course page: http://cs-people.bu.edu/lapets/320/
- Piazza: https://piazza.com/bu/spring2015/cs320
- Instructor: http://cs-people.bu.edu/lapets/
- Teaching Assistant: Zhiqiang Ren (Alex)
  - Email: aren AT cs DOT bu DOT edu
  - Office hour: Mon 4:00 PM - 5:30 PM, Wed 3:00 PM - 4:30 PM, Undergraduate Lab
  - Lab Session
    - 10:00 - 11:00 Wed (MCS B23)
    - 11:00 - 12:00 Wed (MCS B23)
    - 12:00 - 13:00 Wed (MCS B19)

3.2 Working With CS Computing Resources

See the Getting Started (thanks to Likai) guide for tips on working from home and transferring files over, and for a primer on using Linux. There is no need to follow these instructions if you are familiar with Linux, they are for your reference only. PuTTy is a free SSH and telnet client. If you are a BU student, you can get X-Win32 here.

3.3 Contents

The contents here will be updated as the course goes on.

3.3.1 Discussion Session 1

Submissions

In this course, we are using `gsubmit` to submit homework. The note describes how to install the softwares necessary to use `gsubmit`. You can find more details about `gsubmit` here. And please **strictly** follow the instructions on that page to submit your homework, **or otherwise your homework will not be graded at all**.

Some important thing to know *(extracted from the documentation)*:

- Before submission, make sure all the files are under a **single folder** named `hwXX`, or otherwise we can't see it in the right place. *e.g.* `hw01`, `hw02`, ...
- Submit that folder as a whole into the right course. *e.g.* `gsubmit cs320 hw01`

> **Warning:** The command is **case-sensitive**, `gsubmit CS320 hw01` will submit your work to another planet. Please use lower-case whenever possible.

- If you forget how to use `gsubmit`, try `gsubmit --help` for help.

Quick Usage Example

Suppose I have three files to submit: `hello.h`, `hello.c`, and `main.c`, which are already on the csa2.bu.edu server.

```
ssh username@csa2.bu.edu            #login to csa2.bu.edu using your BU CS login name
mkdir hw01                          #create a folder using a correct name for this homework
cp hello.h hello.c main.c hw01      #copy everything into this folder
gsubmit cs320 hw01                  #submit
gsubmit cs320 -ls                   #double check that everything is submitted
```

Python

The note contains a simple introduction of Python programming language. We will use Python 3 for grading. Make sure you are using the correct version when working on your homework.

Background

1. Programming language is a set of programs.
2. We use grammars to describe programming languages.
3. Writing a program using a programming language means keying a sequence of characters/tokens compabile with the grammar of the programming language.
4. We use notations to describe grammers.

Formal Language

1. It has an **alphabet**, which is a finite set of symbols, and is usually denoted as Σ.
2. **String** is a finite sequence of symbols from the alphabet, including empty string ϵ.
3. A **formal language** is *a set of strings* defined over an alphabet, including the empty set $\emptyset$.
4. We use notations to describe grammars.
5. We implement grammars as automata.
6. We use automata to recognize programming languages.

Formal Grammar

Formal grammar is a set of production rules which generate all the strings of a corresponding formal language.

Types of Grammars

Different grammars have different abilities of describing languages. According to Chomsky *[wikich]*, there are four types of grammars in descending order w.r.t. their abilities.

Type 0 Unrestricted grammars. This type of grammars generate recursively enumerable languages.

Type 1 Context-sensitive grammars. These grammars generate context-sensitive languages.

Type 2 Context-free grammars. These grammars generate context-free languages.

Type 3 Regular grammars. These grammars generate regular languages.

Note: Note that actually, people can add restrictions onto these four types of grammars, and use those subset grammars to generate subset languages. For example, there are some important subsets of context-free grammars, like *LL(k)* and *LR(k)* grammars. You don't need to learn it for now. Just get some sense of those terminologies and their relationship.

Regular Language and Regular Expression

Regular language is a formal language, regular expression (in formal language theory) is a way (notation) to describe regular grammar.

Regular Language

Recall that a language is essentially a set of strings.

- The empty set is a regular language.
- Every symbol of $\Sigma \cup \{\epsilon\}$ is a regular language.
- If L_1, L_2 are regular languages, then
  - $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$ is a regular language. It is formed by concatenate strings in both languages. Sometimes it is written as $L_1 L_2$.
  - $L_1 \cup L_2$ is a regular language. It is simply the union of both languages.

  - L^* is a regular language. This is called the Kleene-Star, or Kleene closure. It is formed by concatenating any strings in L any times (including zero times). *e.g.* $\{a, b\}^* = \{\epsilon, a, b, ab, aa, bb, abb, aab, aaa, baa, bba, bbb, ...\}$.

- And there is no other regular languages.

Examples

Assume $\Sigma = \{a, b\}$. $\{\epsilon\}, \emptyset, \{a\}, \{a, a\}, \{abaab, babba\}$ are regular languages. $\{a^n b^n \mid n \in N\}$ is not.

Regular Expression

A regular expression describes a regular language. It is actually a compact notation for regular grammars. A regular expression itself is a character string of special form. The set of all valid regular expressions is itself a language. An informal description (grammar) of such language is given in the note.

Question

Can this language be described by a regular expression?

Let's play with regular expression a little bit. http://www.regexr.com/

- Match number between 0 and 255.

  ---

  text

  .11.

  .0.

  .249.

  .253.

- Match phone number of US formats.

  ---

  text

  1-234-567-8901

  1-234-567-8901

  1-234-567-8901

  1 (234) 567-8901

  1.234.567.8901

  1/234/567/8901

  12345678901

BNF: Backus Naur Form

BNF stands for **Backus Naur Form** (*Backus Normal Form* is not suggested *[bnf]*), which is a notation technique to describe **context-free grammars** *[wikibnf] [wikicfg]*.

As mentioned, those grammars correspond to different type of languages, and they use different notations to describe themselves. **BNF is one of the notations that can describe context-free grammars.**

BNF in Action

```
number  ::=   digit | digit number
digit   ::=   0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

This can be explained line by line in English as follows:

- A number consists of a digit, **or alternatively**, a digit **followed by** a number **recursively**.
- And a digit consists of any single digit from 0 to 9.

This may not be a perfect definition for numbers, but you can get a sense.

BNF Syntax

- Each group containing a `::=` is a rule, where the LHS will be further expanded into RHS.
- Those names on the LHS of `::=` are rule names.

  In the above example, there are two rules, `number` and `digit`.
- The vertical bar `|` can be read as "or alternatively" as used in the above explanation. It seperates different expansion alternatives of a single rule.
- Those names that only appear in the RHS are **terminals**. And those names appear on LHS, or on both sides, are **non-terminals**.

  `digit`, `number` are non-terminals, while `0 .. 9` are terminals.

Variations

Different versions of BNF exists, and one of those core problems is to differ terminals from non-terminals. Someone may be familiar with this:

```
<number> ::= <digit> | <digit> <number>
<digit>  ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
```

where terminals are in `''`, and non-terminals are in `<>`. Other syntaxs exist, but they are pretty much similar.

Extensions

BNF has some extentions, and they are generally for the sake of simplicity and succinctness. Please google EBNF and ABNF for ideas.

Here I want to present some commonly used notions.

- `+` means repeating one or more times. *e.g.* `number ::= digit+`
- `*` means repeating zero or more times. *e.g.* `number ::= digit digit*`
- `[]` means repeating zero or one time, namely an optional part. *e.g.* `function_call ::= function_name '(' [params] ')'`
- `{}` means repeating zero or more times, just as `*`. *e.g.* `id ::= letter {letter | digit}`
- `()` means a group. *e.g.* `id ::= letter (letter | digit)*`

Warning: The same symbols may have different meanings in different context. Here we are using them in the scope of formal language theory. Later you will use them in Python and Haskell, where they have different meanings.

BNF can replace regular expression

As mentioned, regular expression is a compact notation of regular grammars. And grammar is actually a set of production rules. So we can actually rewrite regular expressions using BNF notation.

Say we have a regular expression `00[0-9]*` (*this is a coder's way of regexp, a math people would write* $00(0|1|2|3|4|5|6|7|8|9)^*$ *instead*), it can be written as

```
Start  ::=   0 A1
A1     ::=   0
A1     ::=   0 A2
A2     ::=   0 | 1 | ...  | 9
A2     ::=   (0 | 1 | ...  | 9) A2
```

Describe the language of regular expression using BNF

```
RE  ::=   char
RE  ::=   RE RE
RE  ::=   RE | RE
RE  ::=   RE+
RE  ::=   RE*
RE  ::=   (RE)
```

Bibliography

3.3.2 Discussion Session 2

Grammar

- Nonterminal
- Terminal
- Production Rule (Left Hand Side, Right Hand Side)

Example

```
start  ::=   exp // Start
exp    ::=   exp + term // Add
exp    ::=   exp - term // Minus
```

```
exp     ::=   term // Term
term    ::=   ID // Id
term    ::=   NUM // Num
term    ::=   (exp) // Group
```

Parsing

Parsing is the procedure of transforming a list of tokens into a tree with tokens as leaves.

Note: This process is also called derivation.

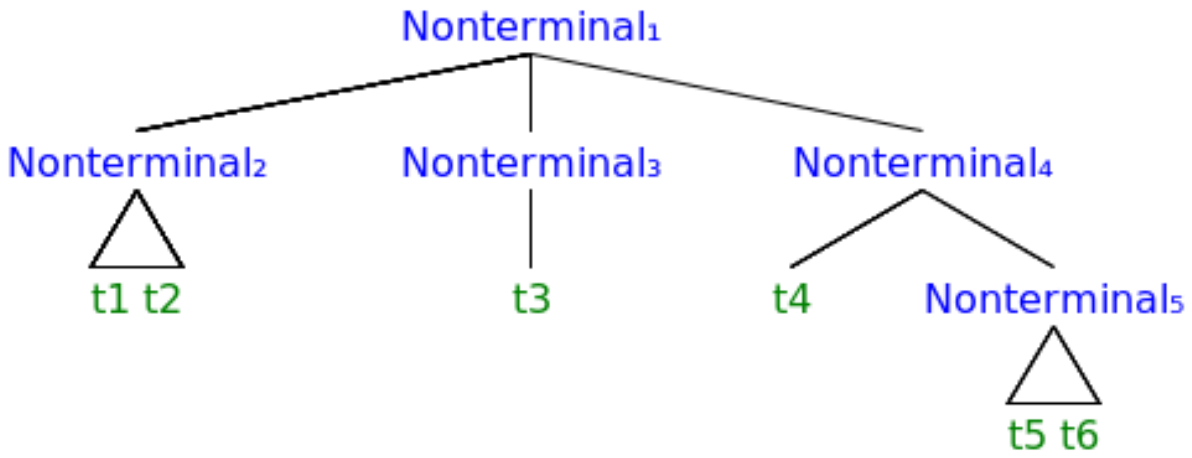
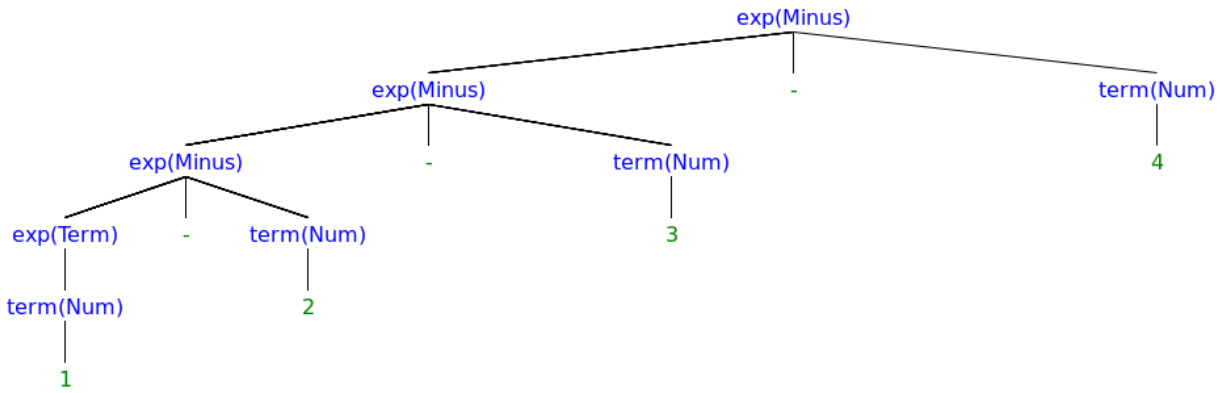


Fig. 3.1: Token stream `t1, t2, t3, t4, t5, t6, ......` and Parsing Tree (Abstract Syntax)

Color	Meaning
Green	Terminal
Blue	Nonterminal

Example

The abstract syntax for `1 - 2 - 3 - 4` goes as follows:



- A good grammar has no ambiguity. Only one tree can be constructed from the token stream.
- A good grammar should match human expectation.

Example

Try the following grammar on `1 - 2 - 3 - 4`.

```
start  ::=   exp // Start
exp    ::=   term + exp // Add
exp    ::=   term - exp // Minus
exp    ::=   term // Term
term   ::=   ID // Id
term   ::=   NUM // Num
term   ::=   (exp) // Group
```

Anatomy of LL(k)

L: Left-to-right

Examine the input (token stream) left-to-right in one parse.

L: Leftmost-derivation

Create / expand the left most sub-tree first.

Note: **R**: Rightmost-derivation

---

demo

Try parsing "1 + 2 - 3 * 4 + 5" with the following grammar.

```
start  ::=   exp // Start
exp    ::=   exp + term // Add
exp    ::=   exp - term // Minus
exp    ::=   term // Term
term   ::=   term * NUM // Mul
term   ::=   term / NUM // Div
term   ::=   NUM // Num
term   ::=   (exp) // Group
```

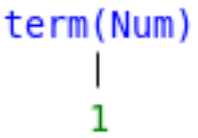


Fig. 3.2: Step 1

---

k: Lookahead

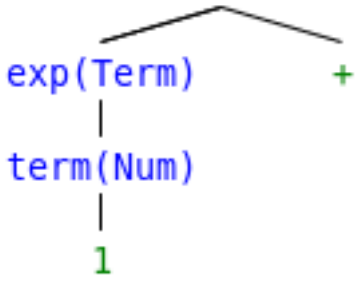


Fig. 3.3: Step 2

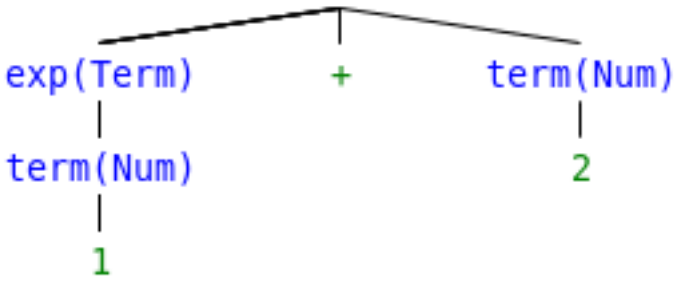


Fig. 3.4: Step 3

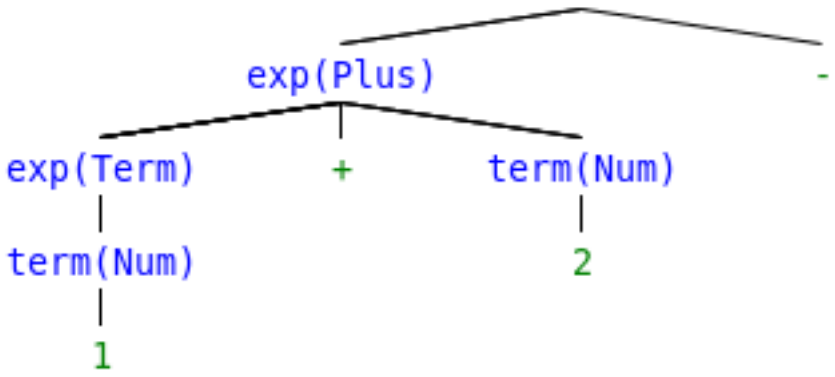


Fig. 3.5: Step 4

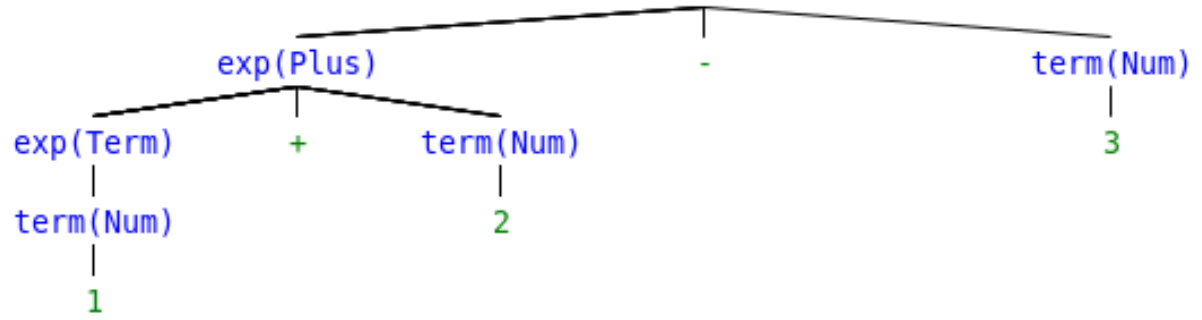


Fig. 3.6: Step 5

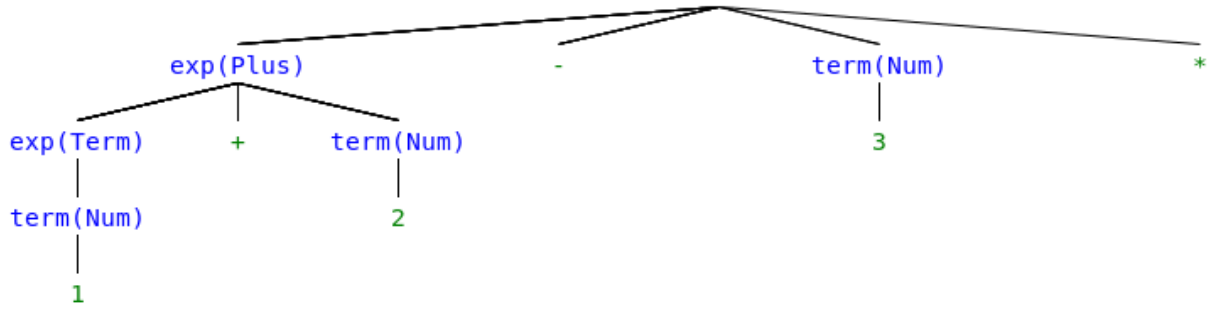


Fig. 3.7: Step 6

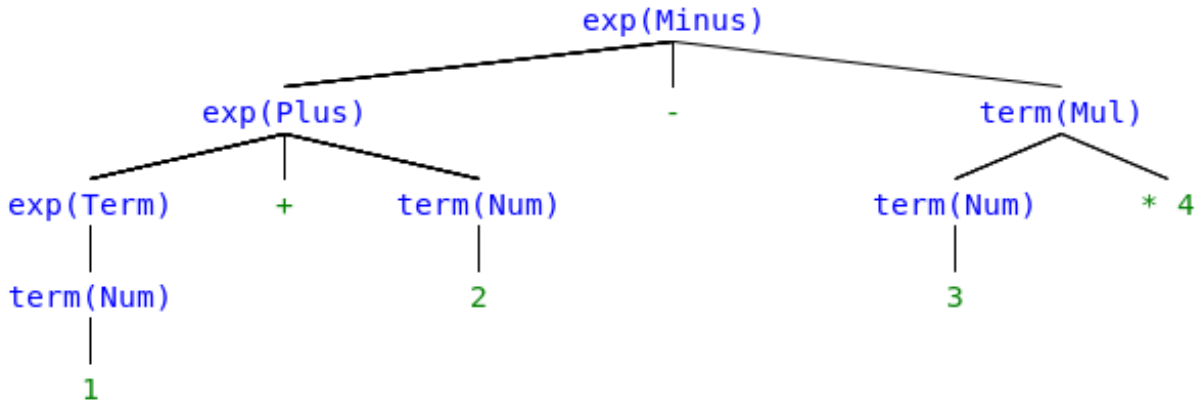


Fig. 3.8: Step 7

Inspect the first **k** tokens before making decision.

Eliminating Left Recursion

For productions like this:

$$P \to P\alpha_1 \mid P\alpha_2 \mid \cdots \mid P\alpha_m \mid \beta_1 \mid \cdots \mid \beta_n$$
$$\text{where } \alpha \neq \varepsilon, \beta \text{ don't start with } P$$

It will be turned into

$$P \to \beta_1 P' \mid \beta_2 P' \mid \cdots \mid \beta_n P'$$
$$P' \to \alpha_1 P' \mid \alpha_2 P' \mid \cdots \mid \alpha_m P' \mid \varepsilon$$

And you can verify that the resulting language is the same.

Warning: This is actually eliminating direct left recursions, and turning them into right recursions. There are methods to eliminate all recursions, direct or indirect, but it is more complicated, and needs some restrictions on the input grammar.

Example

```
start  ::=  exp // Start
exp    ::=  exp + term // Add
exp    ::=  exp - term // Minus
exp    ::=  term // Term
term   ::=  ID // Id
term   ::=  NUM // Num
term   ::=  (exp) // Group
```

is turned into

```
start  ::=  exp
exp    ::=  term exp1
exp1   ::=  + term exp1
exp1   ::=  - term exp1
exp1   ::=  epsilon
```

Coding Demo

Left-factoring

For productions like this:

$$A \to \delta\beta_1 \mid \delta\beta_2 \mid \cdots \mid \delta\beta_n \mid \gamma_1 \mid \cdots \mid \gamma_m$$

We turn them into

$$A \to \delta A' \mid \gamma_1 \mid \cdots \mid \gamma_m$$
$$A' \to \beta_1 \mid \cdots \mid \beta_n$$

Example

```
start  ::=   exp // Start
exp    ::=   exp + term // Add
exp    ::=   exp - term // Minus
exp    ::=   term // Term
term   ::=   ID // Id
term   ::=   NUM // Num
term   ::=   (exp) // Group
```

is turned into

```
start  ::=   exp
exp    ::=   exp term1
term1  ::=   + term
term1  ::=   - term
exp    ::=   term
```

Do left recursion elimination.

```
start  ::=   exp
exp    ::=   term exp1
exp1   ::=   term1 exp1
exp1   ::=   epsilon
term1  ::=   + term
term1  ::=   - term
```

3.3.3 Discussion Session 3

Ambiguity X Left Recursion X Associativity

Grammar with ambiguity:

```
term  ::=   term + term
term  ::=   term - term
term  ::=   NUM
```

After doing left-recursion elimination mechanically:

```
term   ::=   NUM term2
term2  ::=   + term term2
term2  ::=   - term term2
term2  ::=   // Empty
```

Left-recursion elimination won't be able to remove ambiguity. The grammar still has ambiguity, but we can use

recursive descent parsing technique now. Ambiguity is ressolved by the parsing process. (Semantics of the language is now influenced by the parsing process.)

Grammar without ambiguity but with left recursion (left associativity):

```
term  ::=   term + NUM
term  ::=   term - NUM
term  ::=   NUM
```

After doing left-recursion elimination mechanically:

```
term   ::=   NUM term2
term2  ::=   + NUM term2
term2  ::=   - NUM term2
term2  ::=   // Empty
```

A smarter way to express such grammar (right associativity):

```
term  ::=   NUM + term
term  ::=   NUM - term
term  ::=   NUM
```

Operator Precedence and Associativity

```
start   ::=   term2 // lowest precedence
term2   ::=   term2 Opr2a term1 // left associative
term2   ::=   term2 Opr2b term1 // left associative
term2   ::=   term1 // next precedence
term1   ::=   term Opr1 term1 // right associative
term1   ::=   term // next precedence
term    ::=   factor + term // right associative
term    ::=   factor // next precedence
factor  ::=   factor * base // left associative
factor  ::=   base // next precedence
base    ::=   - base
base    ::=   log (term2)
base    ::=   (term2)
base    ::=   Variable
base    ::=   Number
```

1. Operator Precedence: The lower in the grammar, the higher the precedence.
2. Operator Associativity:

- Tie breaker for precedence
- Left recursion in the grammar means
  - left associativity of the operator
  - left branch in the tree

- Right recursion in the grammar means
  - Right associativity of the operator
  - Right branch in the tree

Limitation of our implementation of Recursive Descendent Parser

```
program     ::=   print expression ; program
program     ::=   // Empty
expression  ::=   formula
expression  ::=   term
```

```
formula  ::=   prop
formula  ::=   prop and formula
formula  ::=   prop or formula
prop     ::=   true
prop     ::=   false
prop     ::=   VAR
```

```
term    ::=   factor + factor
term    ::=   factor - factor
term    ::=   factor
factor  ::=   NUM
factor  ::=   VAR
```

Question

Does this grammar have ambiguity?

Assume we do not care. Let parsing process dissolve the ambiguity for concrete syntax like `print x`.

Question

Can our recursive descendent parser handle the concrete syntax `print true and false`?

Ordering of rules matters.

Question

Can our recursive descendent parser handle the concrete syntax `print x + 1`?

Any remedy?

```
program  ::=   print formula ; program
program  ::=   print term ; program
program  ::=   // Empty
```

```
program     ::=   print expression program
program     ::=   // Empty
expression  ::=   formula;
expression  ::=   term;
```

Question

What is perfect backtracking?

```
a1  ::=   b1 b2 b3
b1  ::=   c1
b1  ::=   c2
```

Search all possible sequences of choices: record the state at each choice and backtrack if necessary.

3.3.4 Discussion Session 4

Language of Regular Expressions

The language of all regular expressions can be described by the following grammar (BNF)

```
reg  ::=   CHAR
reg  ::=   .
reg  ::=   reg reg
reg  ::=   reg | reg
reg  ::=   reg *
reg  ::=   reg ?
reg  ::=   ( reg )
```

The following grammar doesn't have ambiguity and sets different precedences for operators `|`, space (invisible operator), and `*`.

```
reg    ::=   seq | reg // Alt
reg    ::=   seq
seq    ::=   block seq // Cat
seq    ::=   block
block  ::=   atom * // Star
block  ::=   atom ?  // Opt
block  ::=   atom
atom   ::=   CHAR // Char
atom   ::=   .  // Any
atom   ::=   ( reg ) // Paren
```

Question

Can the language of regular expressions be described by a regular expression?

Example of Abstract Syntax Tree

```
{"Char": ["a"]}     # a

{"Alt": [           # ab|c
    {"Cat": [
        {"Char": ["a"]},
        {"Char": ["b"]}]},
    {"Char": ["c"]}]}

{"Star": [{"Char": ["a"]}]}   # a*

"Any"         # .
```

Match Regular Expression against String

Simple implementation based on Search: `regmatch.py`

```
# Author: Zhiqiang Ren
# Date: 02/18/2015

def reg_match(reg, exp):
    node = (exp, reg, [])
    return search([node])

def search(nodes):
    while (not nodes_is_empty(nodes)):
        node = nodes_get_node(nodes)
        # print "node is", node

        if (node_is_sol(node)):
            return True

        new_nodes = node_get_next(node)
        nodes = nodes_add(nodes[1:], new_nodes)

    return False


# implement nodes as a stack
def nodes_is_empty(nodes):
    return len(nodes) == 0

def nodes_get_node(nodes):
    return nodes[0]

def nodes_add(nodes, new_nodes):
    return nodes + new_nodes


# node: (exp, cur_reg, rest_regs)
def node_is_sol(node):
    (exp, cur_reg, rest_regs) = node
    if (exp == "" and cur_reg == None and rest_regs == []):
        return True
    else:
        return False
```

```
40
41  def node_get_next(node):
42      (exp, cur_reg, rest_regs) = node
43      if (cur_reg == None):
44          return []
45
46      if (type(cur_reg) == str):  # Terminal
47          if ("Any" == cur_reg):
48              if (len(exp) <= 0):
49                  return []
50              else:
51                  if (len(rest_regs) > 0):
52                      return [(exp[1:], rest_regs[0], rest_regs[1:])]
53                  else:
54                      return [(exp[1:], None, [])]
55          else:
56              raise NotImplementedError(cur_reg + " is not supported")
57
58      elif (type(cur_reg) == dict):
59          [(label, children)] = cur_reg.items()
60          if ("Char" == label):
61              ch = children[0]
62              if (len(exp) <= 0):
63                  return []
64              else:
65                  if (exp[0] == ch):
66                      if (len(rest_regs) > 0):
67                          return [(exp[1:], rest_regs[0], rest_regs[1:])]
68                      else:
69                          return [(exp[1:], None, [])]
70                  else:
71                      return []
72
73          elif ("Cat" == label):
74              reg1 = children[0]
75              reg2 = children[1]
76              return [(exp, reg1, [reg2] + rest_regs)]
77
78          elif ("Alt" == label):
79              reg1 = children[0]
80              reg2 = children[1]
81
82              return [(exp, reg1, rest_regs), (exp, reg2, rest_regs)]
83
84          elif ("Opt" == label):
85              reg = children[0]
86
87              if (len(rest_regs) > 0):
88                  new_node = (exp, rest_regs[0], rest_regs[1:])
89              else:
90                  new_node = (exp, None, rest_regs[1:])
91
92              return [(exp, reg, rest_regs), new_node]
93
94          elif ("Star" == label):
95              reg = children[0]
96              num = children[1]
97
```

```
            if (num == len(exp)):
                return []

            # empty
            if (len(rest_regs) > 0):
                node1 = (exp, rest_regs[0], rest_regs[1:])
            else:
                node1 = (exp, None, rest_regs[1:])

            new_star = {"Star": [reg, len(exp)]}
            node2 = (exp, reg, [new_star] + rest_regs)

            return [node1, node2]
        else:
            raise NotImplementedError(label + " is not supported")

    else:
        raise NotImplementedError(str(type(cur_reg)) + " is not supported")


if __name__ == "__main__":
    reg_a = {"Char": ["a"]}
    reg_b = {"Char": ["b"]}
    reg_c = {"Char": ["c"]}
    reg_ab = {"Cat": [reg_a, reg_b]}
    reg_a_b = {"Alt": [reg_a, reg_b]}
    reg_ao = {"Opt": [reg_a]}
    reg_any = "Any"

    reg_as = {"Star": [reg_a, -1]}
    reg_ass = {"Star": [reg_as, -1]}

    # ret = reg_match(reg_a, "a")
    # print "=== ret is ", ret
    # ret = reg_match(reg_b, "b")
    # print "=== ret is ", ret
    # ret = reg_match(reg_ab, "ab")
    # print "=== is ", ret
    # ret = reg_match(reg_a_b, "a")
    # print "=== is ", ret
    # ret = reg_match(reg_a_b, "b")
    # print "=== is ", ret
    # ret = reg_match(reg_any, "ba")
    # print "=== is ", ret
    # ret = reg_match(reg_ao, "a")
    # print "=== is ", ret
    # ret = reg_match(reg_as, "")
    # print "=== is ", ret
    # ret = reg_match(reg_ass, "aaaaa")
    # print "=== is ", ret

    # (a|b)*
    reg1 = {"Star": [reg_a_b, -1]}
    # (a|b)*c
    reg2 = {"Cat": [reg1, reg_c]}
    # ((a|b)*c)*
    reg3 = {"Star": [reg2, -1]}
    ret = reg_match(reg3, "aabaabaccccaaaaaaaabbccc")
```

```
    print "=== is ", ret
```

3.3.5 Discussion Session 5

Computer Architecture

In computer engineering, computer architecture is a set of disciplines that describes the functionality, the organization and the implementation of computer systems; that is, it defines the capabilities of a computer and its programming model in an abstract way, and how the internal organization of the system is designed and implemented to meet the specified capabilities. *[wikiarch]*

The following emulator describes the architecture in an abstract way.

```
######################################################################
#
# CAS CS 320, Spring 2015
# Assignment 3 (skeleton code)
# machine.py
#

def simulate(s):
    instructions = s if type(s) == list else s.split("\n")
    instructions = [l.strip().split(" ") for l in instructions]
    mem = {0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: -1, 6: 0}
    control = 0
    outputs = []
    while control < len(instructions):
        # Update the memory address for control.
        mem[6] = control

        # Retrieve the current instruction.
        inst = instructions[control]
        # print control
        # print("memory: "+str(mem))
        # print "ins is", inst

        # Handle the instruction.
        if inst[0] == 'label':
            pass
        if inst[0] == 'goto':
            control = instructions.index(['label', inst[1]])
            continue
        if inst[0] == 'branch' and mem[int(inst[2])]:
            control = instructions.index(['label', inst[1]])
            continue
        if inst[0] == 'jump':
            control = mem[int(inst[1])]
            continue
        if inst[0] == 'set':
            mem[int(inst[1])] = int(inst[2])
        if inst[0] == 'copy':
            mem[mem[4]] = mem[mem[3]]
        if inst[0] == 'add':
            mem[0] = mem[1] + mem[2]

        # Push the output address's content to the output.
        if mem[5] > -1:
```

```
            outputs.append(mem[5])
            mem[5] = -1

        # Move control to the next instruction.
        control = control + 1

    print("memory: "+str(mem))
    return outputs

# Examples of useful helper functions from lecture.
def copy(frm, to):
   return [\
      'set 3 ' + str(frm),\
      'set 4 ' + str(to),\
      'copy'\
   ]

# eof
```

Instruction Set

- goto *label*
- branch *label* [addr]
- jump [addr]
- set [addr] **val**
- copy [addr_from] [addr_to]
- add

Memory Model

The range of memory address spans from negative infinity to positive infinity. (We have infinite amount of memory, which is not possible in real case.)

Memory addresses with special functionalites: 0, 1, 2, 3, 4, 5, 6:

- 0, 1, 2 are used by *add* instruction.
- 3, 4 are used by *copy* instruction.
- 5 is used for output.
- 6 contains the current value of the program counter (instruction pointer).

Manual division of memory area:

- Stack: <= -1
- 7 is used to store the address of the top of the stack.
- Heap: > 8

Program Examples

1. Please write the machine code, the execution of which would increase the value in address 8 by 1.

   Ans:

   machine.py

```
from machine import *

init1 = ["set 8 100"]

ans1 = ["set 3 8", \
        "set 4 1", \
        "copy", \
        "set 2 1", \
        "add", \
        "set 3 0", \
        "set 4 8", \
        "copy"]

def copy(src, dst):
    return ["set 3 " + str(src), \
            "set 4 " + str(dst), \
            "copy"]

ans1a = copy(8, 1) + \
        ["set 2 1", \
         "add" ] + \
         copy(0, 8)

print simulate(init1 + ans1)
print simulate(init1 + ans1a)
```

2. Please write the machine code, the execution of which have the following property:

   Assume in the initial state, memory 8 stores a natural number n, after the execution memory 9 should store the summation of all the natural numbers less than or equal to n. For instance, if memory 8 stores 10 initially, then memory 9 should store 55 after the execution.

   Ans:

   machine.py

```
from machine import *

def copy(src, dst):
    return ["set 3 " + str(src), \
            "set 4 " + str(dst), \
            "copy"]

def decrement(addr):
    return copy(addr, 1) + \
        ["set 2 -1", \
         "add" ] + \
         copy(0, addr)

def addto(src, dst):
    return copy(src, 1) + \
           copy(dst, 2) + \
           ["add"] + \
```

```
            copy(0, dst)

ans2 = ["set 9 0"] + \
        copy(8, 10) + \
        ["label start", \
        "branch loop 10", \
        "goto end", \
        "label loop"] + \
        addto(10, 9) + \
        decrement(10) + \
        ["goto start", \
        "label end"]

init2 = ["set 8 100"]
print simulate(init2 + ans2)
```

3. Please write the machine code for the defintion of a recursive function, which can output 100 several times according to the value stored in memory 8. Please also write the code for invoking such function.

   Ans:

   machine.py

```
from machine import *

init1 = ["set 8 100"]

ans1 = ["set 3 8", \
        "set 4 1", \
        "copy", \
        "set 2 1", \
        "add", \
        "set 3 0", \
        "set 4 8", \
        "copy"]

def copy(src, dst):
    return ["set 3 " + str(src), \
            "set 4 " + str(dst), \
            "copy"]

ans1a = copy(8, 1) + \
        ["set 2 1", \
         "add" ] + \
         copy(0, 8)

def increment(addr):
    return copy(addr, 1) + \
        ["set 2 1", \
         "add" ] + \
         copy(0, addr)

def decrement(addr):
    return copy(addr, 1) + \
        ["set 2 -1", \
         "add" ] + \
         copy(0, addr)

def addto(src, dst):
```

```
    return copy(src, 1) + \
           copy(dst, 2) + \
           ["add"] + \
           copy(0, dst)

print simulate(init1 + ans1)
print simulate(init1 + ans1a)
print simulate(init1 + decrement(8))

ans2 = ["set 9 0"] + \
       copy(8, 10) + \
       ["label start", \
       "branch loop 10", \
       "goto end", \
       "label loop"] + \
       addto(10, 9) + \
       decrement(10) + \
       ["goto start", \
       "label end"]

init2 = ["set 8 100"]
print simulate(init2 + ans2)

init3 = ["set 7 -1", \
         "set 8 5"]

ans3 = [ "goto printx_end", \
         "label printx_start", \
         "branch printx_skip 8", \
         "goto printx_return", \
         "label printx_skip",
         "set 5 100"] + \
         decrement(8) + \
         decrement(7) + \
         ["set 3 6", \
          "set 4 1", \
          "copy", \
          "set 2 9", \
          "add"] + \
          copy(7, 4) + \
          ["set 3 0", \
           "copy"] + \
          ["goto printx_start"] + \
          increment(7) + \
          ["label printx_return"] + \
           copy(7, 3) + \
          ["set 4 4", \
           "copy", \
           "jump 4", \
           "label printx_end"] + \
          \
         decrement(7) + \
         ["set 3 6", \
          "set 4 1", \
          "copy", \
          "set 2 9", \
          "add"] + \
          copy(7, 4) + \
```

```
            ["set 3 0", \
             "copy"] + \
            ["goto printx_start"] + \
            increment(7)

print simulate(init3 + ans3)

def procedure(name, body):
    return [ "goto " + name + "_end", \
          "label " + name + "_start"] + \
          body + \
          copy(7, 3) + \
          ["set 4 4", \
           "copy", \
           "jump 4", \
           "label " + name + "_end"]

def call(name):
    return decrement(7) + \
          ["set 3 6", \
           "set 4 1", \
           "copy", \
           "set 2 9", \
           "add"] + \
           copy(7, 4) + \
           ["set 3 0", \
            "copy"] + \
           ["goto " + name + "_start"] + \
           increment(7)

body =  ["branch printx_skip 8", \
          "goto printx_return", \
          "label printx_skip",
          "set 5 100"] + \
          decrement(8) + \
          call("printx") + \
          ["label printx_return"]

init4 = ["set 7 -1", \
          "set 8 5"]
ans4 = procedure("printx", body) + call("printx")

print simulate(init4 + ans4)
```

Tail Call Optimization

While loop and recursive tail call are equivalent.

Discussion

What's the benefit?

Bibliography

3.3.6 Discussion Session 6: Program Verification

Intepretation

```
formula  ::=  true | false
formula  ::=  not formula
formula  ::=  formula and formula
formula  ::=  formula or formula
```

Python code:

```python
def formula_true():
    return "True"

def formula_false():
    return "False"

def formula_not(formula):
    return {"Not": [formula]}

def formula_and(formula1, formula2):
    return {"And": [formula1, formula2]}

def formula_or(formula1, formula2):
    return {"Or": [formula1, formula2]}

def evaluateFormula(formula):
    if is_true(formula):
        return True
    if is_false(formula):
        return False
    if is_not(formula):
        return not evaluateFormula(formula["Not"][0])
    if is_and(formula):
        formula1 = formula["And"][0]
        formula2 = formula["And"][1]
        return evaluateFormula(formula1) and evaluateFormula(formula2)
    if is_or(formula):
        formula1 = formula["Or"][0]
        formula2 = formula["Or"][1]
        return evaluateFormula(formula1) or evaluateFormula(formula2)
    return None
```

Discussion

- What's the type for formula?
- How to prove that evaluateFormula is implemented correctly?
- What is correct? (Hint: Evaluation Rule)

Bounded Exhaustive Testing

Set of all possible inputs is defined inductively. We can enumerate them exhaustively. See notes. The introduction of metric guarantees that we do enumerate all the possibilities.

- If formula is of format “true” or “false”, then its height is 1.
- If formula is of format “not formula0”, then its height is 1 + height of “formula0”.
- If formula is of format “formula1 and formula2”, then its height is 1 + max(height of “formula1”, height of “formula2”.
- If formula is of format “formula1 and formula2”, then its height is 1 + max(height of “formula1”, height of “formula2”.

Code:

```
def metric(f):
    if is_true(f) or is_false(f):
        return 1
    if is_not(f):
        return 1 + metric(f["Not"][0])
    if is_and(f):
        f1 = f["And"][0]
        f2 = f["And"][1]
        return 1 + max(metric(f1), metric(f2))
    if is_or(f):
        f1 = f["Or"][0]
        f2 = f["Or"][1]
        return 1 + max(metric(f1), metric(f2))

def formulas(n):
    if n <= 0:
        []
    elif n == 1:
        return [formula_true(), formula_false()]
    else:
        fs = formulas(n-1)
        fsN = []
        fsN += [formula_not(f) for f in fs]
        fsN += [formula_and(f1, f2) for f1 in fs for f2 in fs]
        fsN += [formula_or(f1, f2)  for f1 in fs for f2 in fs]
        return fs + fsN
```

Proof by Induction

- Base Case: evaluateFormula is correct for formula whose height is 1.
- Inductive Step: The input formula has height n+1.
- Induction Hypothesis: evaluateFormula is correct for formula whose height is <= n.

Example of fibonacci function

Definition of fibonacci function

fib(n) = 0 if n = 0

1 if n = 1

fib(n-1) + f(n-2) if n > 1

Implementation of fibonacci function

```
def Fib(n):
  def Fib0(n, x, y):
    if n = 0:
      return y
    if n > 0:
      return Fib0(n - 1, x + y, x)

  return Fib0(n, 1, 0)
```

Verification Task

For any n >= 0, fib(n) == Fib(n).

Proof By Induction

We prove the following instead.

For any n >= 0, for any a >= 0, Fib0(n, fib(a+1), fib(a)) == fib(a+n).

Base Case:

When n = 0, we have
for any a >= 0, Fib0(0, fib(a+1), fib(a)) = fib(a) <== (By def of Fib0)

Inductive Step: n = m > 0

Inductive Hypothesis: For any m0 < m, for any a >= 0, Fib0(m0, fib(a+1), fib(a)) == fib(a+m0).

For any a >= 0, we have the following

Fib0(m, fib(a+1), fib(a))

= Fib0(m-1, fib(a+1) + fib(a), fib(a+1)) <== (By def of Fib0)

= Fib0(m-1, fib(a+2), fib(a+1)) <== (By def of fib)

= fib(a+1 + m-1) <== (By Induction Hypothesis)

= fib(a+m) <== (Done)

Programming with Theorem Proving

Advanced programming languages provide us with abilities to write proof along with code so that the compiler can help check the correctness of our implementation. One example of such language is ATS.

```
dataprop fib (int, int) =
| fibzero (0, 0) of ()
| fibone (1, 1) of ()
| {n:nat} {r1,r2:int}
  fibcons (n+2, r1 + r2) of (fib (n+1, r1), fib (n, r2))

fun Fib0 {n:nat} {a:nat} {r1,r0:int} .<n>.(
  pf1: fib (a+1, r1), pf0: fib (a, r0)
  | x: int n, y1: int r1, y0: int r0):<fun>
  [r: int] (fib (a+n, r) | int r) =
```

```
if x = 0 then (pf0 | y0)
else let
  prval pf2 = fibcons (pf1, pf0) // fib (a+2, r1 + r0)
in
  Fib0 (pf2, pf1 | x - 1, y1 + y0, y1)
end

fun Fib {n:nat} .<>.(x: int n):<fun> [r:int] (fib (n, r) | int r) =
  Fib0 (fibone, fibzero | x, 1, 0)
```

You can try type checking the code at http://www.ats-lang.org/SERVER/MYCODE/Patsoptaas_serve.php?mycode=hello.

3.3.7 Discussion Session 7: Type System

Compiler must terminate!

Whether interpreter terminates depends on the content of the program.

Discussion

- Why does your compilation program terminate?

Type Theory

Background

Type theory was invented by Bertrand Russell as a solution to his own well-known paradox, which cast doubt on the foundations of mathematics upon its discovery. The problem posed by the paradox is essentially

> Given a set of all sets that do not contain themselves, does that set contain itself?

Type theory resolves this issue by classifying the members of sets according to some type, and in such a system a set definition like this one is not permissible.

Nowadays, type systems are at the heart of the development of many modern programming languages. What is the benefit of a type system in a programming language? In the words of Benjamin Pierce [1]: A type system is a syntactic method for automatically checking the absence of certain erroneous behaviors by classifying program phrases according to the kinds of values they compute.

Discussion

- What are erroneous behaviors? (Hint: For our machine language, what program can cause our "machine" to crash?)

Robin Milner [2] provided the following slogan to describe type safety:

> Well-typed programs cannot "go wrong".

[1] Pierce, Benjamin C. (2002). Types and Programming Languages. MIT Press. ISBN 978-0-262-16209-8.
[2] Milner, Robin (1978), A Theory of Type Polymorphism in Programming, Jcss 17: 348–375.

Type Inference / Judgement / Derivation

All these names are referring to the same concept: How to assign types to each part of the abstract syntax. The corresponding rules are normally closely related to the syntax of the language.

Example

Let's add one more production rule to the language.

```
expression ::= if expression then expression else expression
```

What type inference rule shall we add?

Type Checking Algorithm

Let's implement the type checking algorithm for our language with indexed string type.

Sample Program:

```
# function foo(string[6] x) {
#   return x + x;
# }
# x = "abc"
# y = "def"
# z = foo(x + y)

program = {"Function": [ {"Variable": ["foo"]} \
                       , {"String": [{"Number": [6]}]} \
                       , {"Variable": ["x"]} \
                       , {"Add": [ {"Variable": ["x"]} \
                                 , {"Variable": ["x"]}]} \
                       , {"Assign": [ {"Variable": ["x"]} \
                                    , {"String": ["abc"]} \
                                    , {"Assign": [ {"Variable": ["y"]} \
                                                 , {"String": ["def"]} \
                                                 , {"Assign": [ {"Variable": ["z"]} \
                                                              , {"Apply": [ {"Variable": ["foo"]} \
                                                                          , {"Add": [ {"Variable": ["
                                                                                    , {"Variable": ["
                                                              , "End" ]} ]} ]} ]}
```

Solution: `check_sol.py`

```
Node = dict
Leaf = str

def isStringType(ty):
    if ty == None:
        return False
    if ty == "Void":
        return False
    for label in ty:
        if label == "String":
            return True
        else: # Arrow
            return False
```

```
14
15 def tyExpr(env, e):
16     if type(e) == Node:
17         for label in e:
18             children = e[label]
19
20             if label == "String":
21                 return {"String": [{"Number": [len(children[0])]}]}
22
23             elif label == 'Variable':
24                 x = children[0]
25                 return env[x]
26
27             elif label == "Concat":
28                 (e1, e2) = children
29                 tyE1 = tyExpr(env, e1)
30                 tyE2 = tyExpr(env, e2)
31                 if isStringType(tyE1) and isStringType(tyE2):
32                     index1 = tyE1["String"][0]
33                     index2 = tyE2["String"][0]
34                     index = {"Add": [index1, index2]}
35                     return {"String": [index]}
36
37             elif label == 'Apply':
38                  [f, eArg] = children
39                  f = f['Variable'][0] # Unpack.
40                  tyArg = tyExpr(env, eArg)
41                  tyPara = env[f]['Arrow'][0]
42
43                  if tyArg == tyPara:
44                      return env[f]['Arrow'][1]
45                  else:
46                      print("not match")
47                      print("tyArg is", tyArg)
48                      print("tyPara is", tyPara)
49
50                  if isStringType(tyArg) == False:
51                      return None
52
53                  tv0 = evalIndex(tyArg["String"][0])
54                  tv1 = evalIndex(tyPara["String"][0])
55
56                  if tv0 == tv1:
57                      print("tv0 == tv1 ==", tv0)
58                      return env[f]['Arrow'][1]
59                  else:
60                      print("not match")
61                      print("tyArg evaluates to", tv0)
62                      print("tyPara evaluates to", tv1)
63
64
65 def tyProg(env, s):
66     if type(s) == Leaf:
67         if s == 'End':
68             return 'Void'
69     elif type(s) == Node:
70         for label in s:
71             if label == 'Assign':
```

```
                [x,e,p] = s[label]
                x = x['Variable'][0] # Unpack.
                tExpr = tyExpr(env, e)
                env[x] = tExpr  # add to environment
                tProg = tyProg(env, p)
                if isStringType(tExpr) and tProg == 'Void': # checking
                    return 'Void'
            if label == 'Function':
                [f, tyArg, x, e, p] = s[label]
                if not isStringType(tyArg): # checking
                    print("type error, tyAry is", tyAry)
                    return None
                name = f['Variable'][0]
                x = x['Variable'][0]
                envF = env.copy()
                envF[x] = tyArg
                tBody = tyExpr(envF, e)
                if not isStringType(tBody): # checking
                    print("type error, tBody is", tBody)
                    return None
                env[name] = {'Arrow':[tyArg,tBody]}
                tProg = tyProg(env, p)
                if tProg == 'Void':
                    return tProg

def evalIndex(ti):
    for label in ti:
        if label == "Number":
            return ti[label][0]
        elif label == "Add":
            ti0 = ti[label][0]
            ti1 = ti[label][1]
            return evalIndex(ti0) + evalIndex(ti1)

def main():
    # function foo(string[6] x) {
    #   return x + x;
    # }
    # x = "abc"
    # y = "def"
    # z = foo(x + y)

    program = {"Function": [ {"Variable": ["foo"]} \
                           , {"String": [{"Number": [6]}]} \
                           , {"Variable": ["x"]} \
                           , {"Concat": [ {"Variable": ["x"]} \
                                        , {"Variable": ["x"]}]} \
                           , {"Assign": [ {"Variable": ["x"]} \
                                        , {"String": ["abc"]} \
                                        , {"Assign": [ {"Variable": ["y"]} \
                                                     , {"String": ["def"]} \
                                                     , {"Assign": [ {"Variable": ["z"]} \
                                                                  , {"Apply": [ {"Variable": ["foo"]}
                                                                              , {"Concat": [ {"Variab
                                                                                           , {"Variab
                                                                  , "End" ]} ]} ]} ]}
    print(program)
    print()
```

```
130     tyProg({}, program)
131
132
133 if __name__ == "__main__":
134     main()
```

Bibliography

3.3.8 Discussion Session 8: Unification

Statement of Problem

Unification, in computer science and logic, is an algorithmic process of solving equations between *symbolic* expressions. [3]

```
term     ::=   variable
term     ::=   id (termlst)
termlst  ::=   term termlst
termlst  ::=
```

Substitution is a mapping from *id* to *term*.

The essential task of unification is to find a substitution σ that *unifies* two given terms (i.e., makes them equal). Let's write $\sigma(t)$ for the result of applying the substitution σ to the term t. Thus, given t_1 and t_2, we want to find σ such that $\sigma(t_1) = \sigma(t_2)$. Such a substitution σ is called a unifier for t_1 and t_2. For example, given the two terms:

```
f(x, g(y))       f(g(z), w)
```

where `x`, `y`, `z`, and `w` are variables, the substitution:

```
sigma = {x: g(z), w: g(y)}
```

would be a unifier, since:

```
sigma( f(x, g(y)) ) = sigma( f(g(z), w) ) = f(g(z), g(y))
```

Unifiers do not necessary exist. However, when a unifier exists, there is a *most general unifier* (mgu) that is unique up to renaming. A unifier σ for t_1 and t_2 is an mgu for t_1 and t_2 if

- σ is a unifier for t_1 and t_2; and
- any other unifier σ' for t_1 and t_2 is a refinement of σ; that is, σ' can be obtained from σ by doing further substitutions.

Application of Unification

- Pattern Matching (A simplified version of Unification)

  Algorithm and example (notes)

- Type Inference

  Let's set up some typing rules for Python similar to those of Java or C. Then we can use unification to infer the types of the following Python programs:

[3] http://en.wikipedia.org/wiki/Unification_%28computer_science%29

```
    def foo(x):
        y = foo(3)
        return y

    def foo(x):
        y = foo(3)
        z = foo(y)
        return z

    def foo(x):
        y = foo(3)
        z = foo(4)
        r = foo(y, z)
        return r
```

- Logic Programming (Prolog)

A More General Unification Algorithm

Some examples that simplified algorithm cannot handle:

```
x     f(x)
f(x, g(x))     f(h(y), y)
```

Instead of unifying a pair of terms, we work on a list of pairs of terms:

```
# We use a list of pairs to represent unifier. The unifier has a property
# no variable on a lhs occurs in any term earlier in the list
# [(x3: x4), (x1: f(x3, x4)), (x2: f(g(x1, x3), x3))]
# Another way to view this.
# Let's rename these variables by ordering them.
# x3 -> y1
# x4 -> y0
# x1 -> y2
# x2 -> y3
# [(y1: y0), (y2: f(y1, y0)), (y3: f(g(y2, y1), y1))]

def unify_one(t1, t2): # return a list of pairs for unifier
    if t1 is variable x and t2 is variable y:
        if x == y:
            return []
        else:
            return [(x, t2)]
    elif t1 is f(ts1) and t2 is g(ts2): # ts1 and ts2 are lists of terms.
        if f == g and len(ts1) == len(ts2):
            return unify(ts1, ts2)
        else:
            return None # Not unifiable: id conflict
    elif t1 is variable x and t2 is f(ts):
        if x occurrs in t2:
            return None # Not unifiable: circularity
        else:
            return [(x, t2)]
    else: # t1 is f(ts) and t2 is variable x
        if x occurrs in t1:
            return None # Not unifiable: circularity
        else:
            return [(x, t1)]
```

```
def unify(ts1, ts2): # return a list of pairs for unifier
    if len(ts1) == 0:
        return []
    else:
        ts1_header = ts1[0]
        ts1_tail = ts1[1:]
        ts2_header = ts2[0]
        ts1_tail = ts2[1:]

        s2 = unify(ts1_tail, ts2_tail)
        t1 = apply(s2, ts1_header)
        t2 = apply(s2, ts2_header)
        s1 = unify_one(t1, t2)
        return s1 + s2

def apply(s, t):
    // substitute one by one backward
    n = len(s) - 1
    while n >= 0:
        p = s[n]
        t = subs(p, t)
        n = n - 1
    return t
```

Example:

```
f(x1, g(x2, x1), x2)    f(a, x3, f(x1, b))
```

Bibliography

3.3.9 Discussion Session 9: Play with Haskell

Problem Set

- List Operation (Merge Sort)
- Tree Operation (Binary Search Tree)
- Conversion between List and Tree
- Higher Order Function
- Lazy Evaluation and Stream

Code Example

```haskell
module Quiz where

data IList = Nil
           | Cons Int IList
           deriving (Eq, Show)

ilength :: IList -> Int
ilength xs = case xs of
             Nil -> 0
             Cons x xs -> 1 + ilength xs
```

```
11
12 itake :: IList -> Int -> IList
13 itake xs 0 = Nil
14 itake (Cons x xs) n = Cons x (itake xs (n - 1))
15
16
17 iremove :: IList -> Int -> IList
18 iremove xs 0 = xs
19 iremove (Cons x xs) n = iremove xs (n - 1)
20
21 data Option a = None
22             | Some a
23
24 iremove_opt :: IList -> Int -> Option IList
25 iremove_opt xs 0 = Some xs
26 iremove_opt (Cons x xs) n = iremove_opt xs (n - 1)
27 iremove_opt xs n = None
28
29 -- Please implement the function to append two IList's.
30 -- iappend :: IList -> IList -> IList
31
32 -- Please implement the merge sort algorithm on IList
33 -- merge_sort :: IList -> IList
34
35 -- imap
36 -- ifold
37
38 data ITree = Leaf
39             | Node Int ITree ITree
40            deriving (Eq, Show)
41
42 from_ilist :: IList -> ITree
43 from_ilist Nil = Leaf
44 from_ilist xs = let
45     n = ilength xs
46   in
47     from_ilist_num xs n
48
49
50 from_ilist_num :: IList -> Int -> ITree
51 from_ilist_num xs 0 = Leaf
52 from_ilist_num (Cons x xs) 1 = Node x Leaf Leaf
53 from_ilist_num xs n = let
54     n1 = n `div` 2
55     xs1 = itake xs n1
56     Some (Cons x xs2) = iremove_opt xs n1
57     t1 = from_ilist_num xs1 n1
58     t2 = from_ilist_num xs2 (n - 1 - n1)
59   in
60     Node x t1 t2
61
62 nats :: IList
63 nats = genNat 0
64
65 genNat :: Int -> IList
66 genNat n = Cons n (genNat (n + 1))
67
68 nat3 = itake nats 3
```

```
69
70  ifilter :: IList -> (Int -> Bool) -> IList
71  ifilter Nil f = Nil
72  ifilter (Cons x xs) f =
73    if (f x) then Cons x (ifilter xs f)
74    else ifilter xs f
75
76  isEven x = if x `mod` 2 == 0 then True else False
77
78  evens = ifilter nats isEven
79
80  even3 = itake evens 3
81
82  genPrime0 (Cons x xs) f = if (f x) then genPrime0 xs f
83                            else let
84                              newf y = if (f y) then True
85                                       else if (y `mod` x == 0) then True
86                                            else False
87                              xs2 = genPrime0 xs newf
88                            in
89                              Cons x xs2
90
91  f0 n = if n < 2 then True else False
92
93  genPrime xs = genPrime0 xs f0
94
95  primes = genPrime nats
96
97  prime5 = itake primes 5
98  prime10 = itake primes 10
99
100
101
102
103
104 main :: IO ()
105 main = do putStrLn (show prime10)
```

quiz.hs

3.3.10 Discussion Session 10: Search

Haskell Syntax

Complicated Haskell programs can be built upon the following syntax.

```
1  -- Elementary expression
2  a = 1
3  b = "abc"
4  c = (show a) ++ b
5
6  -- "let" expression
7  x = let
8    x = 1  -- binding
9    y = 2  -- binding
10   sum :: Int -> Int -> Int -- function declaration
11   sum a b = a + b  -- function definition
```

```
  in
  sum x y

-- "if" expression
y = if x > 0 then 'a'
else
  if x == 0 then 'b'
  else 'c'

data IList =
  Cons Int IList
 | Nil

xs = Cons 1 (Cons 2 Nil)

-- "case" expression
z = case xs of
  Cons _ _ -> 1  -- all clauses must be of the same indentation
  Nil -> 2

-- "case" expression
z' = case y of
  'a' -> "a"
  'b' -> "b"

fz 'a' = "a"
fz 'b' = "b"
z'' = fz y


-- type of main is IO (), "IO" is a type constructor
main :: IO ()
main = do
  let
    a = 1 -- Indentation is important
    b = let
      c = 1
      in
      a + c
  putStrLn (show z)  -- The following two lines are of the same indentation.
  putStrLn (show z)
```

Concept

- State
- Search Tree / Graph
- Strategy
- Solution / Best Solution

Code Example

8-Queen problem

```haskell
module EightQueen where

data Cell = E | Q deriving (Show)
type Position = (Integer, Integer)
data Board = Board [Position]

data Graph =
   Branch Board [Graph]
 | Finish Board

instance Show Board where
  show (Board xs) = let
    show_line_loop cur pos len accu =
      if cur >= len then accu
      else if cur == pos then show_line_loop (cur + 1) pos len (accu ++ "Q ")
      else show_line_loop (cur + 1) pos len (accu ++ ". ")

    show_loop ((_, pos) : tail) accu = let
        new_line = show_line_loop 0 pos 8 ""
      in
        show_loop tail (accu ++ new_line ++ "\n")

    show_loop [] accu = accu

    in
      show_loop xs ""

nextMoves :: Board -> [Board]
nextMoves (Board xs) = let
    row = toInteger (length xs)
    cols = [0 .. 7]
    candidates = [(row, y) | y <- cols]

    conflict :: Position -> Position -> Bool
    conflict (x1,y1) (x2,y2) = if y1 == y2 then True
                               else if abs(x1 - x2) == abs(y1 - y2) then True
                               else False

    conflict_list :: Position -> [Position] -> Bool
    conflict_list x (head : tail) = (conflict x head) || (conflict_list x tail)
    conflict_list x [] = False

    new_moves = [c | c <- candidates, (conflict_list c xs) == False]

    new_boards = [Board (xs ++ [pos]) | pos <- new_moves]
  in
    new_boards

graph :: Board -> Graph
graph (Board xs) = if (length xs) >= 8 then Finish (Board xs)
                   else let
                       new_boards = nextMoves (Board xs)
                     in
                       if length(new_boards) == 0 then Finish (Board xs)
```

```
                        else let
                            new_graphs = [graph b | b <- new_boards]
                          in
                            Branch (Board xs) new_graphs


is_sol :: Board -> Bool
is_sol (Board xs) = (length xs) == 8

find_sol :: Graph -> [Board]
find_sol (Branch b gs) = let
  bss = [find_sol g | g <- gs]
  bs = foldl (\x -> \y -> x ++ y) [] bss
  in
  if is_sol b then b : bs
  else bs

find_sol (Finish b) =
  if is_sol b then [b]
  else []

search_space = graph (Board [])
sol = find_sol (search_space)

show_boards (b : bs) = (show b) ++ "\n\n\n" ++ (show_boards bs)
show_boards [] = "\n"

main :: IO ()
main = do putStrLn (show_boards sol)
          putStrLn $ "Total solutions found: " ++ show (length sol)
```

Solution: `eightqueens.hs` `Makefile`

Regular Expression Match

```
module RegExp where

data Regexp =
   REnil
  | REemp
  | REany
  | REchar Char
  | REalt Regexp Regexp
  | REcat Regexp Regexp
  | REstar Regexp

string_regexp_match str reg = let
  str_isempty :: String -> Bool
  str_isempty [] = True
  str_isempty _ = False
  in
  accept str reg str_isempty

accept str reg k =
  case reg of
  REnil -> False
  REemp -> k str
```

```
  REany -> (case str of
             _ : cs -> k cs
             [] -> False
           )
  REchar c -> (case str of
             c' : cs -> if c == c' then k cs else False
             [] -> False
             )
  REalt reg1 reg2 -> if accept str reg1 k then True
                     else accept str reg2 k
  REcat reg1 reg2 -> let
    k' xs = accept xs reg2 k
    in
    accept str reg1 k'
  REstar reg0 -> if k str then True
                 else let
                   k' xs = if (length xs) == (length str) then False
                           else accept xs reg k
                   in
                   accept str reg0 k'


main = do
  let
    reg_a = REchar 'a'
    reg_b = REchar 'b'
    reg_ab = REalt reg_a reg_b
    reg_abs = REcat reg_ab (REstar reg_ab)
    reg_as = REstar reg_a
    reg_ass = REstar reg_as
  putStrLn $ show $ string_regexp_match "abaaab" reg_abs
  putStrLn (show (string_regexp_match "ac" reg_abs))
  putStrLn (show (string_regexp_match "aa" reg_ass))
```

Solution: `regexp.hs` `Makefile`

Misc

Some syntax sugar first.

The followings are equivalent:

```
putStrLn (show (1 + 1))
putStrLn $ show $ 1 + 1
putStrLn . show $ 1 + 1
```

The `$` sign is used to avoid parenthesis. Whatever on the right of it takes precedence. `.` sign is used to chain functions. The output of RHS will be the input of LHS.

And here is a very good place to lookup useful functions in the Prelude module of Haskell. http://hackage.haskell.org/package/base-4.6.0.1/docs/Prelude.html

CHAPTER 4

CS320 Concepts of Programming Languages (Summer 2015)

Welcome!

This is the teaching fellow's page for course CS 320 Concepts of Programming Languages. I will post related material here from time to time so that both students and professor can keep track of the updated info about the lab session. This course will use Piazza as the main source of communication among the class. I shall keep the information between these two sites synchronized. Piazza has the higher priority than this website however. It would be a superset of this site, at least it would contain the notice that something new has been added here. Simply check Piazza first before visiting here and no need to worry about missing a thing.

4.1 General Information

- Official course page: http://www.cs.bu.edu/~hwxi/academic/courses/CS320/Summer15/classpage.html
- Piazza: https://piazza.com/class/i9enorcjabc233
- Instructor: http://www.cs.bu.edu/~hwxi/
- Teaching Assistant: Zhiqiang Ren (Alex)
  - Email: aren AT cs DOT bu DOT edu
  - Office hour: Mon 5:00 PM - 06:30 PM, Thu 01:00 PM - 02:30 PM, Undergraduate Lab
  - Lab Session
    - 14:00 - 15:00 Wed (EPC 201)

4.2 Working With CS Computing Resources

See the Getting Started (thanks to Likai) guide for tips on working from home and transferring files over, and for a primer on using Linux. There is no need to follow these instructions if you are familiar with Linux, they are for your reference only. PuTTy is a free SSH and telnet client. If you are a BU student, you can get X-Win32 here.

4.3 Contents

The contents here will be updated as the course goes on.

4.3.1 Lab Session 1

Usage of Git

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. In this class we use Git to distribute course materials, assignments, projects. Grading is also based on the usage of Git. You can find various information (documentation, tutorial, and etc) about Git from its website http://git-scm.com/ as well as many other sources available online. However, we only need a very small set of Git commands, which will be illustrated later in this page.

Bitbucket

Bitbucket https://bitbucket.org provides online Git repository as well as related management facilities. After getting a free account on Bitbucket, you can create private repository holding your work for this class.

Create repository for this class

Use the webpage of Bitbucket to create a new repository. Make sure that the new repository has the name CS320-Summer-2015 and keep all the other settings by default.

Record the name of your new repository. For example, mine is `https://alex2ren@bitbucket.org/alex2ren/cs320-sum`

Cloud9

Cloud9 provides a full Ubuntu workspace in the cloud as well as online code editors. Simply put, it provides a virtual machine which can be accessed via web browser. Just imagine that you can access csa2.bu.edu via a web browser instead of a Telnet/SSH client (e.g. PuTTy). What's even better is that you are provided with text editors to edit files.

Create a new workspace on Cloud9 for this class. Open a terminal in the newly created workspace and execute the following commands to install ATS2 in this workspace.

```
# download the script for installing ATS2 into your workspace
wget https://gist.githubusercontent.com/githwxi/7e31f4fd4df92125b73c/raw/ATS2-C9-install.sh
# execute the script
sh ./ATS2-C9-install.sh
```

Execute the following commands to set up the Git repository for tracking your code.

```
mkdir cs320-summer-2015
cd cs320-summer-2015
git init # initialize the repository
# add your own repository on Bitbucket with name mybitbucket
git remote add mybitbucket https://alex2ren@bitbucket.org/alex2ren/cs320-summer-2015.git
# add the repository for this course with name origin
git remote add origin http://bithwxi@bitbucket.org/bithwxi/cs320-summer-2015.git
# get all the resources released so far
git fetch origin
git merge origin/master
# push everything to your own repository on Bitbucket.
git push -u mybitbucket --all # pushes up the repo and its refs for the first time
```

Now you can work on assignment00. You can share your workspace with others by inviting them into your workspace by clicking *share*. My C9 id is *alex2ren*.

After finish your coding, try the following command:

```
cd cs320-summer-2015
git status
```

Git would remind you which files have been modified. The following is an example:

> On branch master Your branch is up-to-date with 'mybitbucket/master'.
>
> **Changes not staged for commit:** (use "git add <file>..." to update what will be committed) (use "git checkout – <file>..." to discard changes in working directory)
>
> modified: assignments/00/assign00_sol.dats
>
> **Untracked files:** (use "git add <file>..." to include in what will be committed)
>
> assignments/00/assign00_sol assignments/00/assign00_sol_dats.c
>
> no changes added to commit (use "git add" and/or "git commit -a")

Use the following command to stage your modification.

```
git add assignments/00/assign00_sol.dats
```

Use the following command to commit your modification.

```
git commit -m "This is my first try of ATS."
```

Try the following command to check the history of your commit.

```
git log
```

Use the following command to push your commit onto your own repository on Bitbucket.

```
git push mybitbucket master
```

Now you can go to Bitbucket and share your repository with us for grading.

In the future, you can use the following commands on Cloud9 to get the newest materials of the class.

```
cd cs320-summer-2015 # get into the directory for the repository
git fetch origin
git merge origin/master
```

Warning: For each assignment, some files contain incomplete code left for you to finish. You can edit these files and creating new files. Do not edit other files. The following command can help undo your modification to an existing file.

```
git checkout -- <file>  # replace <file> with the path of the file
```

csa2.bu.edu

ATS has been installed on csa2.bu.edu. To use it, you need to import the required environment. If you use *bash*, you can use the following command.

```
source /cs/coursedata/cs320/environment
```

The Git related command used on Cloud9 can also be used on csa2. Assume you wrote some code on Cloud9, and succeeded in pushing it onto your repository on Bitbucket, you can use the following command to pull such update into the repository created on csa2.

```
cd cs320-summer-2015 # assume you already created such repository on csa2.bu.edu
git fetch mybitbucket
git merge mybitbucket/master
```

4.3.2 Lab Session 2

Usage of Git: Merge and Resovle Conflict

You can find useful information here on Git's website. We shall also discuss about this in the session.

Use the following commands to create a conflict in Git repository.

```
$ mkdir repoA
$ mkdir repoB
$ cd repoA
$ git init
Initialized empty Git repository in /home/grad2/aren/teach/website/temprepos/repoA/.git/
$ cat > a.txt
Hello World AAA!
$ git add a.txt
$ git commit -m "first commit of A"
[master (root-commit) 0a7b425] first commit of A
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 a.txt
$ git status
# On branch master
nothing to commit (working directory clean)
$ cd ../repoB
$ git init
Initialized empty Git repository in /home/grad2/aren/teach/website/temprepos/repoB/.git/
$ git remote add repoA ../repoA
$ git fetch repoA
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects:  33% (1/3)
Unpacking objects:  66% (2/3)
Unpacking objects: 100% (3/3)
Unpacking objects: 100% (3/3), done.
From ../repoA
 * [new branch]      master     -> repoA/master
$ git merge repoA/master
$ ls
a.txt
$ cat > a.txt
Hello World BBB!
$ git add a.txt
$ git commit -m "update of B"
[master bec83e3] update of B
 1 files changed, 1 insertions(+), 1 deletions(-)
$ cd ../repoA
$ cat >> a.txt
Hello World CCC!
$ git add a.txt
$ git commit -m "second update of A"
[master 21e0ea4] second update of A
 1 files changed, 1 insertions(+), 0 deletions(-)
$ cd ../repoB
```

```
$ git fetch repoA
remote: Counting objects: 5, done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects:  33% (1/3)
Unpacking objects:  66% (2/3)
Unpacking objects: 100% (3/3)
Unpacking objects: 100% (3/3), done.
From ../repoA
   0a7b425..21e0ea4  master     -> repoA/master
$ git merge repoA/master
Auto-merging a.txt
CONFLICT (content): Merge conflict in a.txt
Automatic merge failed; fix conflicts and then commit the result.
$ git status
# On branch master
# Unmerged paths:
#   (use "git add/rm <file>..." as appropriate to mark resolution)
#
#   both modified:      a.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
$ cat a.txt
<<<<<<< HEAD
Hello World BBB!
=======
Hello World AAA!
Hello World CCC!
>>>>>>> repoA/master
$ cat > a.txt
Hello World !!!
$ git add a.txt
$ git commit
```

Slides of the session

`lecture.pdf`.

Review of Assignment 01

```
(*
// Please implement [show_triangle] as follows:
// show_triangle (3) outputs
//    *
//   ***
//  *****
//
// show_triangle (5) outputs
//      *
//     ***
//    *****
//   *******
//  *********
*)

(*
** HX: 10 points
*)
```

```
extern
fun show_triangle (level: int): void

implement
show_triangle (level) = let
  fun printn (c: char, n: int): void =
    if n > 0 then let
      val () = print c
    in
      printn (c, n - 1)
    end
    else ()

  fun print_lines (cur: int, total: int): void =
    if cur >= total then ()
    else let
      val n_blank = 1 + total - cur
      val n_star = 2 * cur + 1
      val () = printn (' ', n_blank)
      val () = printn ('*', n_star)
      val () = print ("\n")
    in
      print_lines (cur + 1, total)
    end
in
  print_lines (0, level)
end
```

4.3.3 Lab Session 4

Unification Problem

Unification, in computer science and logic, is an algorithmic process of solving equations between symbolic expressions. In the following text, we shall give out its formal definiton in maths and programming language ATS.

Expression and Substitution

Definition of language of expressions:

```
exp     ::=   VAR // variable (leaf)
exp     ::=   INT // integer (leaf)
exp     ::=   ID (explst) // constructor (branch)
explst  ::=
explst  ::=   exp explst
VAR     ::=   x, y, z, ...  // strings of characters
INT     ::=   ..., -1, 0, 1, ...  // integers
ID      ::=   x, y, z, ...  // strings of characters
```

Encode the language of expressions in ATS

```
datatype exp =
| VAR of (string)
| INT of (int)
```

```
| CONS of (string (*constructor id*), explst)
where
explst = list0 (exp)
```

Algorithm (equality =): The following algorithm can determine whether two expressions are equivalent.

equal(a, b): two expressions *a* and *b*

if both *a* and *b* are leaf nodes and are equivalent return true

if both *a* and *b* have the same ID and the same number of children

in order from left to right, check each pair of corresponding children of *a* and *b* for equality
return true if all the subexpressions are equivalent
return false otherwise

ATS code

```
extern fun equal (a: exp, b: exp): bool

extern fun print_exp (e: exp): void

implement equal (a, b) =
case+ (a, b) of
| (VAR (na), VAR (nb)) => na = nb
| (INT (ia), INT (ib)) => ia = ib
| (CONS (ida, expsa), CONS (idb, expsb)) =>
  (
  if (ida <> idb) then false else let
    fun cmp2 (xs: list0 exp, ys: list0 exp): bool =
      case+ (xs, ys) of
      | (nil0 (), nil0 ()) => true
      | (cons0 (x, xs1), cons0 (y, ys1)) =>
          if equal (x, y) = false then false
          else cmp2 (xs1, ys1)
      | (_, _) => false
  in
    cmp2 (expsa, expsb)
  end
  )
| (_, _) => false
```

Definition of substitution: A mapping from variabes to expressions, e.g. {“v”: 3, “xs”: cons0 (1, nil0), ...}.

ATS code

```
abstype substitution = ptr
typedef subs = substitution

exception conflict of ()
exception notfound of ()

extern fun subs_create (): subs
extern fun subs_add (s: subs, name: string, v: exp): subs  // may raise exception
extern fun subs_merge (s1: subs, s2: subs): subs  // may raise exception
extern fun subs_get (s: subs, n: string): exp // may raise exception

extern fun print_subs (s: subs): void

assume substitution = '(string, exp)  // one possible implementation
```

Definition of substitute expresion *a* with substitution σ: Replace the variables in an expression with corresponding expression designated in the substitution.

ATS code

```
extern fun subs_substitute (e: exp, s: subs): exp
```

Unification

Definition of Unification (not very strict): Given two expressions a and b, find σ such that $\sigma(a) = \sigma(b)$

Algorithm (pattern matching unification): Whether unification can be computed efficiently, or at all, depends on what restrictio

unify(a, b): two expressions *a* and *b*

if both *a* and *b* are leaf nodes and are equivalent return the empty substitution $\sigma 0$

if *a* is a variable node representing a variable *x* return the substitution $\{x: b\}$

if *b* is a variable node representing a variable *x* return the substitution $\{x: a\}$

if both *a* and *b* have the same ID and the same number of children

in order from left to right, unify each pair of corresponding children of a and b
as long as they do not overlap on any variables, combine the substitutions obtained above
return the combined substitution

ATS code

```
fun unify (a: exp, b: exp): subs  // may raise exception
```

Skeleton Code

```
#define
ATS_PACKNAME "LAB_UNIFICATION"

#include "share/atspre_define.hats"
#include "share/atspre_staload.hats"

#include "share/HATS/atspre_staload_libats_ML.hats"

datatype exp =
| VAR of (string)
| INT of (int)
| CONS of (string (*constructor id*), explst)
where
explst = list0 (exp)


extern fun equal (a: exp, b: exp): bool

implement equal (a, b) =
case+ (a, b) of
| (VAR (na), VAR (nb)) => na = nb
| (INT (ia), INT (ib)) => ia = ib
| (CONS (ida, expsa), CONS (idb, expsb)) =>
  (
  if (ida <> idb) then false else let
    fun cmp2 (xs: list0 exp, ys: list0 exp): bool =
```

```
      case+ (xs, ys) of
      | (nil0 (), nil0 ()) => true
      | (cons0 (x, xs1), cons0 (y, ys1)) =>
          if equal (a, b) = false then false
          else cmp2 (xs1, ys1)
      | (_, _) => false
  in
    cmp2 (expsa, expsb)
  end
  )
| (_, _) => false

fun print_exp (e: exp): void =
case+ e of
| VAR (x) => print x
| INT (x) => print x
| CONS (id, xs) => let
  val () = print (id)
  val () = print (" (")
  val () = print_explst (xs)
  val () = print (")")
in
end
and
print_explst (es: list0 exp): void = let
  fun print_explst_tail (es: list0 exp): void =
    case+ es of
    | cons0 (e, es1) => let
      val () = print ", "
      val () = print_exp (e)
    in
      print_explst_tail (es1)
    end
    | nil0 () => ()
in
  case+ es of
  | cons0 (x, es) => let
    val () = print_exp (x)
    val () = print_explst_tail es
  in end
  | nil0 () => ()
end

overload print with print_exp

(* ************** *************** *)

abstype substitution = ptr
typedef subs = substitution

exception conflict of ()
exception notfound of ()

extern fun subs_create (): subs

// may raise exception
extern fun subs_add (xs: subs, name: string, v: exp): subs
```

```
// may raise exception
extern fun subs_merge (s1: subs, s2: subs): subs

// may raise exception
extern fun subs_get (s: subs, n: string): exp

extern fun subs_substitute (e: exp, s: subs): exp

extern fun print_subs (s: subs): void

local
  assume substitution = list0 '(string, exp)
in
  implement subs_create () = nil0 ()

  implement subs_add (xs, name, v) = let
    fun cmp (res: '(string, exp), x: '(string, exp)):<cloref1> '(string, exp) =
      if (res.0 = x.0) then $raise conflict
      else res

    val _ = list0_foldleft<'(string, exp)><'(string, exp)> (xs, '(name, v), cmp)
  in
    cons0 {'(string, exp)} ('(name, v), xs)
  end

  // implement subs_merge (s1, s2) = todo

  // implement subs_get (s: subs, n: string) = todo

  // implement subs_substitute (e, s) = todo

  implement print_subs (s) = let
    fun print_subs_tail (s: subs): void =
      case+ s of
      | cons0 (p, s) => let
        val () = print! (", ", p.0, ": ", p.1)
      in end
      | nil0 () => ()
    val () = print "{"
    val () = case+ s of
      | cons0 (m, s1) => let
        val () = print! (m.0, ": ", m.1)
      in
        print_subs_tail (s1)
      end
      | nil0 () => ()
    val () = print "}"
  in end

end  // end of [local]

// may raise exception
extern fun unify (a: exp, b: exp): subs

// implement unify (a, b) = todo

implement main0 () = let
  // cons1 (y, nil1 ())
```

```
  val e1 = CONS ("cons1", cons0 (VAR ("y"),
                          cons0 (CONS ("nil1", nil0 ()),
                          nil0))
           )
  // cons1 (x, cons1 (y, nil1 ()))
  val e2 = CONS ("cons1", cons0 (VAR ("x"),
                          cons0 (e1,
                          nil0))
           )

  // cons1 (2, nil1 ())
  val e3 = CONS ("cons1", cons0 (INT (2),
                          cons0 (CONS ("nil1", nil0 ()),
                          nil0))
           )
  // cons1 (1, cons1 (2, nil1 ()))
  val e4 = CONS ("cons1", cons0 (INT (1),
                          cons0 (e3,
                          nil0))
           )
  val e5 = VAR ("x")

  val () = println! ("e2 is ", e2)
  val () = println! ("e4 is ", e4)
  val s = unify (e2, e4)
  val () = print "s = "
  val () = print_subs (s)
  val () = println! ()

  val e2' = subs_substitute (e2, s)
  val e4' = subs_substitute (e4, s)
  val () = println! ("s(e2) = ", e2')
  val () = println! ("s(e4) = ", e4')

  val () = println! ()
  val () = println! ()

  val () = println! ("e5 is ", e5)
  val () = println! ("e3 is ", e3)
  val s = unify (e5, e3)
  val () = print "s = "
  val () = print_subs (s)
  val () = println! ()

  val e5' = subs_substitute (e5, s)
  val e3' = subs_substitute (e3, s)
  val () = println! ("s(e5) = ", e5')
  val () = println! ("s(e3) = ", e3')
in
end
```

unification_skeleton.dats

Solution

unification.dats

Makefile

Bibliography

4.3.4 Lecture 06/11/2015

Reference and Matrix

Reference Type: ref (a)

You can find description of the usage of reference here. Don't forget to add the following in the beginning of your *.dats* file.

```
#include "share/atspre_staload.hats"
```

Interface of *ref (a)*:

```
// quoted from $PATSHOME/prelude/SATS/reference.sats

// Create a reference with initial value
fun{a:vt0p} ref (x: a):<!wrt> ref a

// Get the value stored in the reference
fun{a:t0p} ref_get_elt (r: ref a):<!ref> a

// Store a value into the reference
fun{a:t0p} ref_set_elt (r: ref a, x: a):<!refwrt> void
```

Code Example:

```
val refa = ref<int>(0)  // Apply type argument to template
val x = ref_get_elt (refa)  // O.K. to omit the type argument
val () = ref_set_elt (refa, x + 1)  // O.K. to omit the type argument

val y = !refa  // Simplified form of ref_get_elt
val () = !refa := y + 1  // Simplified form of ref_set_elt
```

Matrix Type: mtrxszref (a, m, n) and matrix0 (a)

mtrxszref (a, m, n) You can find description of the usage of mtrxszref (a) here. Don't forget to add the following in the beginning of your ATS file.

```
#include "share/atspre_staload.hats"
```

Interface of *mtrxszref (a, m, n)*

```
// quoted from $PATSHOME/prelude/SATS/matrixref.sats

fun{
a:t0p
} matrixref_make_elt
  {m,n:int}
  (m: size_t m, n: size_t n, x0: a):<!wrt> matrixref (a, m, n)
// end of [matrixref_make_elt]

fun{a:t0p}
```

```
matrixref_get_at_int
  {m,n:int}
(
  M: matrixref (a, m, n), i: natLt(m), n: int(n), j: natLt(n)
) :<!ref> (a) // end of [matrixref_get_at_int]

fun{a:t0p}
matrixref_get_at_size
  {m,n:int}
(
  M: matrixref (a, m, n), i: sizeLt(m), n: size_t(n), j: sizeLt(n)
) :<!ref> (a) // end of [matrixref_get_at_size]

symintr matrixref_get_at
overload matrixref_get_at with matrixref_get_at_int of 0
overload matrixref_get_at with matrixref_get_at_size of 0

fun{a:t0p}
matrixref_set_at_int
  {m,n:int}
(
  M: matrixref (a, m, n), i: natLt (m), n: int n, j: natLt (n), x: a
) :<!refwrt> void // end of [matrixref_set_at_int]

fun{a:t0p}
matrixref_set_at_size
  {m,n:int}
(
  M: matrixref (a, m, n), i: sizeLt (m), n: size_t n, j: sizeLt (n), x: a
) :<!refwrt> void // end of [matrixref_set_at_size]

symintr matrixref_set_at
overload matrixref_set_at with matrixref_set_at_int of 0
overload matrixref_set_at with matrixref_set_at_size of 0
```

matrixref (a, m, n) uses the feature of dependent type in ATS, which makes it a little bit difficult to use for beginners. Therefore for this course we prefer use a simpler interface matrix0 *(a)*.

matrix0 (a) To use *matrix0 (a)*, please add the following in the beginning of your ATS file.

```
#include "share/HATS/atspre_staload_libats_ML.hats"
```

Interface of *matrix0 (a)*

```
// quoted from $PATSHOME/libats/ML/SATS/matrix0.sats

fun{a:t0p}
matrix0_make_elt
  (nrow: size_t, ncol: size_t, init: a):<!wrt> matrix0 (a)
// end of [matrix0_make_elt]

fun{a:t0p}
matrix0_get_at_int
  (M: matrix0(a), i: int, j: int):<!exnref> a
//
fun{a:t0p}
matrix0_get_at_size
  (M: matrix0 (a), i: size_t, j: size_t):<!exnref> a
```

```
//
symintr matrix0_get_at
overload matrix0_get_at with matrix0_get_at_int
overload matrix0_get_at with matrix0_get_at_size
//
//
fun{a:t0p}
matrix0_set_at_int
  (M: matrix0(a), i: int, j: int, x: a):<!exnrefwrt> void
//
fun{a:t0p}
matrix0_set_at_size
  (M: matrix0 (a), i: size_t, j: size_t, x: a):<!exnrefwrt> void
//
symintr matrix0_set_at
overload matrix0_set_at with matrix0_set_at_int
overload matrix0_set_at with matrix0_set_at_size

overload [] with matrix0_get_at_int of 0
overload [] with matrix0_get_at_size of 0
overload [] with matrix0_set_at_int of 0
overload [] with matrix0_set_at_size of 0

//

fun{}
matrix0_get_nrow{a:vt0p} (M: matrix0 a):<> size_t
fun{}
matrix0_get_ncol{a:vt0p} (M: matrix0 a):<> size_t

//

overload .nrow with matrix0_get_nrow
overload .ncol with matrix0_get_ncol
```

Code Example:

```
val m = matrix0_make_elt<int> (i2sz(3), i2sz(2), 0)  // Apply type argument to the template.
val nrow = matrix0_get_nrow (m)  // type of nrow is size_t
val nrow2 = m.nrow () // Simplified form of matrix0_get_nrow

val ncol = matrix0_get_ncol (m)  // type of nrol is size_t
val ncol2 = m.ncol ()  // Simplified form of matrix0_get_ncol

val x = matrix0_get_at (m, 1 (*row*), 1 (*column*))  // O.K. to omit type argument.
val x2 = m[1, 1]  // Simplified form of matrix0_get_at

val () = matrix0_set_at_int (m, 1, 1, x + 1)
val () = m[1,1] := x + 1  // Simplified form of matrix0_set_at
```

Trouble of size_t and int

size_t and *int* are two different types in ATS. Literal numbers like *1*, *43* are of type *int*. Also ATS compiler doesn't know how to do arithmetic operations on both *size_t* and *int*. Therefore sometimes we may need to use cast function *i2sz* and *sz2i* to convert between these two types:

```
val m = matrix0_make_elt<int> (i2sz(3), i2sz(2), 0)
val sz = m.ncol ()
val x = sz2i (sz) - 1
```

Practice

```
extern fun add (m1: matrix0 int, m2: matrix0 int): matrix0 int
extern fun sub (m1: matrix0 int, m2: matrix0 int): matrix0 int
extern fun mul (m1: matrix0 int, m2: matrix0 int): matrix0 int

extern fun transpose (m: matrix0 int): void

extern fun determinant (m: matrix0 int): int
```

Game of Tetris

You can find the discussion about the game here. To play with it, simply go to http://ats-lang.sourceforge.net/COMPILED/doc/PROJECT/SMALL/JSmygame/Tetris/tetris.html.

```
// Size of the board
#define ROW = 30
#define COL = 14

// Size of the box holding the block.
#define SIZE = 4

// m1 is the current board and m2 is the block to be added on.
// offset is the horizontal difference between
// the board and the box from left to right.
// Modify m1 in place. Keep m2 unchanged. The updated m1 should
// reflect the configuration after dropping the block onto the
// board vertically.
extern fun merge (m1: matrix0 bool,
                  m2: matrix0 bool,
                  offset: size_t
                  ): void

// Find out a way to meature the situation of the current board.
// Feel free to use any heuristics.
extern fun metric (m: matrix0 bool): int
```

4.3.5 Lab Session 5

Quick Sort

Algorithm

Quicksort is an efficient sorting algorithm, serving as a systematic method for placing the elements of an array in order.

In efficient implementations it is not a stable sort, meaning that the relative order of equal sort items is not preserved. Quicksort can operate in-place on an array, requiring small additional amounts of memory to perform the sorting.

Mathematical analysis of quicksort shows that, on average, the algorithm takes $O(nlogn)$ comparisons to sort n items. In the worst case, it makes $O(n^2)$ comparisons, though this behavior is rare.

Bullet Points

- Form up a metric which gets decreased each time invoking a function recursively.

- Assignment formal meaning to function parameters.
- Form up invariant that is valid in the beginning and end of a recursive function.

Code

`Makefile`

The file `quicksort.dats` is an implementation for array of integers.

```
#define
ATS_PACKNAME "LAB_QUICKSORT"

#include "share/atspre_define.hats"
#include "share/atspre_staload.hats"

#include "share/HATS/atspre_staload_libats_ML.hats"


// return value ret: new position for the pivot
// ret < enda
extern fun array0_partition (
  arr: array0 int, beg: size_t, enda: size_t): size_t

extern fun array0_quicksort (arr: array0 int): void

implement array0_quicksort (arr) = let
  val sz = arr.size ()

  fun array0_quicksort_size (
    arr: array0 int,
    beg: size_t,
    enda: size_t
    ): void =
    if enda <= beg then () else let
      // val () = fprint! (stdout_ref, "arr = ")
      // val () = fprint_array0 (stdout_ref,arr)
      // val () = fprintln! (stdout_ref)
      val mid = array0_partition (arr, beg, enda)
      // val () = println! ("array0_quicksort_size, mid is ", mid)
    in
      let
        val () = array0_quicksort_size (arr, beg, mid)
        val () = array0_quicksort_size (arr, succ mid, enda)
      in end
    end
in
  array0_quicksort_size (arr, i2sz 0, sz)
end

implement array0_partition (arr, beg, enda) = let
  // val () = println! ("array0_partition start ",
  //                    " beg is ", beg,
  //                    " enda is ", enda)

  val pivot = arr[enda - i2sz(1)]

  // return value ret: new position for the pivot
```

```
  // ret < enda
  // elements in [beg, div) are less than p
  // elements in [div, cur) are no less than p
  fun loop (arr: array0 int,
            beg: size_t,
            div: size_t,
            cur: size_t,
            enda: size_t,
            p: int
            ): size_t =
  let
    // val () = println! ("loop: beg is ", beg,
    //                    " div is ", div,
    //                    " cur is ", cur,
    //                    " enda is ", enda,
    //                    " pivot is ", p)
  in
    // put pivot at the appropriate position
    if cur = pred (enda) then let
      val v = arr[cur]
      val () = arr[cur] := arr[div]
      val () = arr[div] := v
    in
      div
    end
    else let
      val v = arr[cur]
    in
      if v < p then // swap elements arr[div] and arr[cur]
        if cur = div then loop (arr, beg, succ (div), succ (cur), enda, p)
        else let
          val () = arr[cur] := arr[div]
          val () = arr[div] := v
        in
          loop (arr, beg, succ (div), succ (cur), enda, p)
        end
      else  // div is unchanged
        loop (arr, beg, div, succ (cur), enda, p)
    end
  end  // end of [fun loop]
  val ret = loop (arr, beg, beg, beg, enda, pivot)

  // val () = println! ("array0_partition end, div is ", ret)
in
  ret  // end of array0_partition
end

implement main0 () = let

  // create an array0 in an easy way
  val xs = $arrpsz{int} (2, 9, 8, 4, 5, 3, 1, 7, 6, 0): arrpsz(int, 10) (* end of [val] *)
  val xs = array0 (xs): array0 int

  val () = fprint! (stdout_ref, "xs(*input*)  = ")
  val () = fprint_array0 (stdout_ref, xs)
  val () = fprintln! (stdout_ref)

  val () = array0_quicksort (xs)
```

```
  val () = fprint! (stdout_ref, "xs(*output*)  = ")
  val () = fprint_array0 (stdout_ref, xs)
  val () = fprintln! (stdout_ref)

in
end
```

The file `quicksort_generic.dats` is a generic implementation based on the template system of ATS. This implementation is derived on the previous implementation with tiny modification (e.g. using *gcompare_val_val* instead of < for comparison.

```
#define
ATS_PACKNAME "LAB_QUICKSORT"

#include "share/atspre_define.hats"
#include "share/atspre_staload.hats"

#include "share/HATS/atspre_staload_libats_ML.hats"


// return value ret: new position for the pivot
// ret < enda
extern fun {a:t@ype} array0_partition (
  arr: array0 a, beg: size_t, enda: size_t): size_t

extern fun {a:t@ype} array0_quicksort (arr: array0 a): void

implement {a} array0_quicksort (arr) = let
  val sz = arr.size ()

  fun {a:t@ype} array0_quicksort_size (
    arr: array0 a,
    beg: size_t,
    enda: size_t
    ): void =
    if enda <= beg then () else let
      val mid = array0_partition (arr, beg, enda)
    in
      let
        val () = array0_quicksort_size (arr, beg, mid)
        val () = array0_quicksort_size (arr, succ mid, enda)
      in end
    end
in
  array0_quicksort_size (arr, i2sz 0, sz)
end

implement {a} array0_partition (arr, beg, enda) = let
  val pivot = arr[enda - i2sz(1)]

  // return value ret: new position for the pivot
  // ret < enda
  // elements in [beg, div) are less than p
  // elements in [div, cur) are no less than p
  fun loop (arr: array0 a,
            beg: size_t,
            div: size_t,
            cur: size_t,
```

```
            enda: size_t,
            p: a
            ): size_t =
  if cur = pred (enda) then let
    val v = arr[cur]
    val () = arr[cur] := arr[div]
    val () = arr[div] := v
  in
    div
  end
  else let
    val v = arr[cur]
    val sgn = gcompare_val_val (v, p)  // generic comparison function
  in
    if sgn < 0 then // swap elements arr[div] and arr[cur]
      if cur = div then loop (arr, beg, succ (div), succ (cur), enda, p)
      else let
        val () = arr[cur] := arr[div]
        val () = arr[div] := v
      in
        loop (arr, beg, succ (div), succ (cur), enda, p)
      end
    else  // div is unchanged
      loop (arr, beg, div, succ (cur), enda, p)
  end  // end of [fun loop]
  val ret = loop (arr, beg, beg, beg, enda, pivot)

in
  ret  // end of array0_partition
end

implement main0 () = let

  typedef T = int
  // create an array0 in an easy way
  val xs = $arrpsz{T} (2, 9, 8, 4, 5, 3, 1, 7, 6, 0) (* end of [val] *)
  val xs = array0 (xs)

  val () = fprint! (stdout_ref, "xs(*input*)  = ")
  val () = fprint_array0 (stdout_ref, xs)
  val () = fprintln! (stdout_ref)

  // override the default implementation for gcompare_val_val for int
  implement gcompare_val_val<int> (x1, x2) = ~(x1 - x2)

  val () = array0_quicksort<int> (xs)

  val () = fprint! (stdout_ref, "xs(*output*)  = ")
  val () = fprint_array0 (stdout_ref, xs)
  val () = fprintln! (stdout_ref)

in
end
```

Bibliography

4.3.6 Lab Session 6

Operations on Braun Tree

Algorithm

- size
- get_at

Complexity

size: $O(\log^2 n)$.

Code

`Makefile`

`brauntree.dats`

```
#include "share/atspre_staload.hats"

#include
"share/HATS/atspre_staload_libats_ML.hats"


(* ************** ************** *)

datatype
tree0 (a:t@ype) =
  | tree0_nil of ()
  | tree0_cons of (tree0 a, a, tree0 a)

fun {a:t@ype} brauntree_size (t: tree0 a): int =
case+ t of
| tree0_nil () => 0
| tree0_cons (t1, _, t2) => let
  val sz1 = brauntree_size (t1)
  val sz2 = brauntree_size (t2)
in
  sz1 + sz2 + 1
end


(* ************** ************** *)

// Decide whether the size of t is sz or sz - 1,
// assuming the size of the input tree is either sz or sz - 1
// true: sz
// false: sz - 1
fun {a:t@ype} brauntree_size_is (t: tree0 a, sz: int): bool =
case+ t of
| tree0_nil () => if (sz = 0) then true else false
```

```
| tree0_cons (t1, _, t2) =>
  if sz mod 2 = 1 then brauntree_size_is (t2, sz / 2)
  else brauntree_size_is (t1, sz / 2)

(* ************** ************** *)

fun {a:t@ype} brauntree_size2 (t: tree0 a): int =
case+ t of
| tree0_nil () => 0
| tree0_cons (t1, _, t2) => let
  val sz1 = brauntree_size2 (t1)
in
  if brauntree_size_is (t2, sz1) then 2 * sz1 + 1
  else 2 * sz1
end

(* ************** ************** *)

exception notfound of ()

fun{a:t@ype}
brauntree_get_at(t: tree0(a), i: int): a = let
  val sz = brauntree_size2 (t)

  fun aux (t: tree0 a, i: int, sz: int): a = let
    val sz1 = sz / 2
  in
    case- t of
    | tree0_cons (t1, x, t2) => let
      val sz1 = sz / 2
    in
      if i < sz1 then aux (t1, i, sz1)
      else if i = sz1 then x
      else aux (t2, i - sz1 - 1, sz - sz1 - 1)
    end
  end
in
  if i < 0 || i >= sz then $raise notfound () else aux (t, i, sz)
end

(* ************** ************** *)


extern fun {a:t@ype} mylist0_to_brauntree (xs: list0 a): tree0 a

implement
{a}(*tmp*)
mylist0_to_brauntree(xs) = let
  fun aux (xs: list0 a, len: int): (tree0 a, list0 a) =
  if len = 0 then (tree0_nil (), xs)
  else let
    val len1 = len / 2
    val (t1, xs1) = aux (xs, len1)
    val- cons0 (x, xs2) = xs1
    val (t2, xs3) = aux (xs2, len - len1 - 1)
  in
    (tree0_cons (t1, x, t2), xs3)
  end
```

```
  val len = list0_length (xs)
  val (t, xs1) = aux (xs, len)
in
  t
end

(* ************** ************** *)

implement main0 () = let
  val xs = g0ofg1 ($list (1, 2, 3, 4, 5))
  val t = mylist0_to_brauntree<int> (xs)
  val r = brauntree_size_is (t, 5)
  val () = assertloc (r)

  val xs = g0ofg1 ($list (1, 2, 3, 4))
  val t = mylist0_to_brauntree<int> (xs)
  val r = brauntree_size_is (t, 5)
  val () = assertloc (~r)

  val xs = g0ofg1 ($list (1, 2, 3, 4))
  val t = mylist0_to_brauntree<int> (xs)
  val r = brauntree_size_is (t, 4)
  val () = assertloc (r)

  (* ******** ******** *)

  val xs = g0ofg1 ($list (1, 2, 3, 4, 5))
  val t = mylist0_to_brauntree<int> (xs)
  val r = brauntree_size2 (t)
  val () = assertloc (r = 5)

  (* ******** ******** *)

  val x0 = brauntree_get_at (t, 0)
  val x1 = brauntree_get_at (t, 1)
  val x2 = brauntree_get_at (t, 2)
  val x3 = brauntree_get_at (t, 3)
  val x4 = brauntree_get_at (t, 4)

  val () = assertloc (x0 = 1)
  val () = assertloc (x1 = 2)
  val () = assertloc (x2 = 3)
  val () = assertloc (x3 = 4)
  val () = assertloc (x4 = 5)

  val () = (try let
              val _ = brauntree_get_at (t, 6)
            in
              assertloc (false)
            end
            with ~notfound () => ()
            ): void
in
end
```

[wikibnf] https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_Form

[wikicfg] http://en.wikipedia.org/wiki/Context-free_grammar

[wikich] http://en.wikipedia.org/wiki/Chomsky_hierarchy

[bnf] Knuth, D. E. (1964). Backus normal form vs. backus naur form. Communications of the ACM, 7(12), 735-736.

[wikiarch] http://en.wikipedia.org/wiki/Computer_architecture

[wikiunification] http://en.wikipedia.org/wiki/Unification_%28computer_science%29

[wikiqsort] https://en.wikipedia.org/?title=Quicksort