

Foundations of Computer Science, Spring 2007

School of Computing, University of Utah

Course Numbers: CPSC 5100 and 6100

FAQs

- FAQs on tools are described in Section 2.
- FAQs on Concepts and Assignments is in Section 1.

1 FAQs on Concepts, Assignments

Asg 1: 2.13 “23=24” is to be simplified to “false,” of course. I don’t care whether you view numbers as sets or not. (Numbers as sets was really mentioned in the passing; we will regard numbers as numbers for most of our work.)

2.19 The “X” is a third value modeling “don’t cares” as in digital circuits. For your purposes, take “X” as a third value (besides true / false), as if in a ternary logic system.

2.19 Please take *Int* to be as defined in the book (0, -1, 1, -2, 2, etc.).

2.37, 2.38 Take 2-input gates. What is the AND-gate table?

```
00 0
01 0
10 0
11 1
```

~

```
|-- the output column decides it is an AND-gate. For an OR-gate, the output
column is 0111. So there are 16 ways to fill the output column (each place
2 ways). So there are ‘‘2 raised to (2 raised to 2)’’ 2-input functions.
Do it for a K-input
function. How many places in the output column? How many ways to fill them?
```

2.39 For a partial function over two inputs, here is what’s all possible:

This is an XOR gate. It is *partial* because that is a special case of *total*

```
00 0
01 1
10 1
11 0
```

This is an XOR gate that does not any well-defined behavior for 00 (say, the designer did not design the circuit right for that case):

```
00 undefined
01 1
10 1
11 0
```

This is an XOR gate that does not any well-defined behavior for all inputs (it was a fake):

```
00 undefined
01 undefined
10 undefined
11 undefined
```

So you see, we are using the third value in the range just to denote *that case of the input where the function mapping does not exist*. It is not a real “third value.” The use of \perp or some “fake values” like that to denote undefined behavior is standard in programming semantics.

Asg 2: S-B theorem proof of Hurd: The function f in the lemma operates from A to one of its subsets B . In illustration of the main theorem, A is *Odd* and B is $g(\textit{Even})$.

Asg 3: Starred vs Non-starred: My online description describes my policies accurately (I have *not* changed this description recently - but in case you never read it, please do so.).

5.3: And what about 5.3?

This is an interesting problem, and I recommend that you do it.

As we discussed in class,

$$((a \Rightarrow (b \Rightarrow c)) \Rightarrow (a \Rightarrow b)) \Rightarrow (a \Rightarrow c)$$

is neither *true* nor *false* (does not emerge equal to *true* for all settings of a, b, c – nor does it emerge equal to *false* for all a, b, c).

However, the parenthesization was not what was intended. The following is the correctly parenthesized formula (also mentioned on Page 325 as Rule of Inference A2, written in terms of s, p, q :)

$$(a \Rightarrow (b \Rightarrow c)) \Rightarrow ((a \Rightarrow b) \Rightarrow (a \Rightarrow c))$$

It would be great practice to see which of the above formulae emerges to be a tautology – i.e. which one is always *true* no matter what a, b, c you choose.

5.5: This question should be reworded.

Here is what you should answer:

In each case, determine the mappings defined by f (f may be undefined everywhere, partially defined, or totally defined; identify which case applies, and write a description of the mappings whenever f is defined).

5.27: > The problem states that "everyone at the party was mingling and shaking hands."

> Does this mean we can assume that the minimum number of handshakes any 1

> person made is ≥ 1 ?

You can - but you can also view it as the number of handshakes in which each person participated.

This problem has so many renderings and the only constraints really are:

1- a person does not shake their own hands (does not count)

2- handshake is a symmetric relation (A can't shake B without B having shaken A)

3- for every A and B, A shakes B no more than once (i.e. the "AB" handshake occurred no more than once)

4a- I think you can do the problem either assuming there is a loner
(does not shake hands at all) or

4b- that there is no loner as you say

The assumption of a loner makes the problem even easier

So state your assumptions and then try to do.

Then have fun by changing the assumptions slightly to see if the situation changes

So, don't break #1 or #2 above.

Try breaking #3 above - do things change?

Try breaking #4a - do things change?

Try breaking #4b - do things change?

6.8: Here are the steps

The proposed solution is $f2 = x \geq y \rightarrow x+1, y-1$. Check that.

We have: $F(x,y) = (x=y) \rightarrow y+1, F(x, F(x-1, y+1))$

Let us define $y0 = (x-1) \geq (y+1) \rightarrow x, y$

Then we are asking whether

$(x \geq y) \rightarrow x+1, y-1 \stackrel{?}{=} (x=y) \rightarrow y+1, ((x \geq y0) \rightarrow x+1, y0-1)$

Case $x=y$:

=====

LHS reduces to $x+1$ and RHS reduces to $y+1$ which is equal to $x+1$

Case $x > y$:

=====

In this case, LHS reduces to $x+1$

RHS becomes

$(x \geq y0) \rightarrow x+1, y0-1$

Now we have to instantiate $y0$; thus, we get

$(x \geq$
 $(x-1) \geq (y+1) \rightarrow x, y$
 $)$

$\rightarrow x+1,$

In other words, `App(L1,L2)` just builds a tree of height 1, with the root node labeled "App" and the leaves being the lambda expressions L1 and L2.

Functions such as `App` are called uninterpreted functions. They have no further elaboration.

It is the same idea as representing (as some people do), number 0 as 0, number 1 as `succ(0)`, number 2 as `succ(succ(0))`, etc. Just put that many "shells" around 0 – and lo, you can describe any number (by counting the shells!)

So `App` applied to `foo` and `bar` becomes the term "`App(foo,bar)`".

What can you do with such a term?

All you can do is pull out "foo" and "bar" by, say, pattern matching "`App(foo,bar)`" with say "`App(x,y)`".

So try this: type to the Ocaml prompt

```
# let App(x,y) = App(<some lambda exp>, <another lambda exp>);;
```

You will then find it pulls out `x` and `y` and prints it.

So the type definition for "term" in the Ocaml file reads:

```
type term = Lambda of (term -> term) | App of term * term | Int of int
```

(Think of this as defining a context-free grammar for Lambda terms.)

which means, here are legal examples of lambda terms.

(The "of" is syntactic sugar, which means, apply the constructor to the arguments).

This is also called *discriminated union*. It's actually a union type, but recursive types are allowed.

Another example of declaring a recursive union type would be:

```
type bintree = Node of int * Left of bintree * Right of bintree | Leaf
```

This defines binary trees, which are just a leaf (with nothing in it), or an interior node containing an int, and two bintrees hanging off it.

```
-- from the case Int of int, we can conclude that these terms are legal
```

```
Int(3)
```

```
Int(5)
```

```
-- from the case App of term*term we can conclude that these are legal
```

```
App(Int(3),Int(4))
```

```
App(App(Int(3),Int(4)),Int(5))
```

```
-- from the case Lambda of term -> term, we can conclude that
```

```
-- Lambda , when bolted onto a function of the signature term -> term
```

```
-- is also a lambda term.
```

```
-- But what is a function from type term to type term?
```

```
-- Well, one such function is (fun x -> Int(3))
```

```
-- Thus, Lambda(fun x -> Int(3)) is indeed a Lambda term
```

```
Lambda(fun x -> Int(3))
```

```
-- Now a large example of a Lambda term:
```

```
Lambda ( fun y -> Lambda ( fun z -> Int(3) ) )
```

```
-- Now, why is the above a lambda term? Parse it!
```

```
Lambda [of] ( fun y -> lamda-term )
```

where the lambda-term is

```
Lambda [of] ( fun z -> Int(3) )
```

--- OK?

2. On the Lambda Reducer - how to code if-then-else

> In the lamda calculus, are functions of the following type allowed:

>

> lamda x.(if x=0 then a, else if x =1 then b, else c)

>

> Can if-elseif-else be used, or is there a different way to write such expres

Everything has to be encoded in Lambda Calculus !!

Michael Gordon's Chapter 5 (I've kept mjcg.pdf online) explains how to code.

Basically this is how it works

Suppose you build a lambda expression for the test P

and the THEN part Q

and the ELSE part R

Then "if-then-else" is achieved by ((P Q) R).

or App(App(P,Q),R)

So look at <http://www.cs.utah.edu/classes/cs6100/handouts/5:1-23/asg5.html>

(one can cut and paste the lambda code from here...for those wondering where th

Then look at multfn

What is it saying in the comments inside (* ... *) ?

It is saying "if (iszero m) then zero else (add n (f (pre m) n))"

So it has an If-Then-Else

Now how is it coded? Look at the body of multfn below:

```
App(App(App(iszero,m),zero), App(App(add,n), App(App(f,App(pre,m)),n) ))))
```

^ iszero applied to m

^ then part is 'zero'

^ else part involves 'add' etc...

Hope you can build the desired if-then-else using these hints. If not plz ask

```
(* multfn = \f. \m. \n. ((iszero m) zero (add n (f (pre m) n))) *)

let multfn = Lambda(fun f -> Lambda(fun m -> Lambda(fun n ->
    App(App(App(iszero,m),zero), App(App(add,n), App(App(f,App(pre,m))
```

3. How can I run the mult function?

If you look at the terminal session that I've kept in the class handout page, you'll see two examples of how to run functions:

1) Here is how we run "add" of three and three:

```
ctoi prints Church numerals as int
eval is your evaluator
App(app(add,three)three) is how you say (add three three)
-- as in Lisp, or three + three -- as in other languages like C

ctoi(eval(App(App(add,three),three)));
- : term = Int 6
```

2) Then there is another example that runs "fac" also.

```
Again you do ctoi(eval(...))

I run fac on 3+3

I express it as App(fac,App(App(add,three),three))

This is how (fac (+ 3 3)) -- as in Lisp -- or fac(3+3) --
as in other languages -- is written

# let ctolarge = ctoi(eval(App(fac,App(App(add,three),three))));
val ctolarge : term = Int 720
```

4. Other questions and answers

Q: What recursive sequence is meant by "exp (exponentiation)" in the 4th problem?

A: One could think of expressing exponentiation in terms of multiplication the same way multiplication was expressed in terms of addition

Q: In the lamda calculus, are functions of the following type allowed:
lambda x.(if x=0 then a, else if x =1 then b, else c)

A: No, everything should be in terms of Lambdas

Best is to define a recursive function; here is the idea

Q: In order to encode F, f1, f2, and f3, it seems like we need to create lambda "isEqual(x, y)", and "isGreaterThanOrEqual(x,y)". I've looked at how the "isEqual" function is written in lamba.ml, but I'm still having trouble figuring out how to write the new functions I need.

A: Here is the Ocaml-style pseudocode

```
iseq(x,y) = match (x,y) with
| (0,0) -> true
| (Succ(x1), Succ(y1)) -> iseq(x1,y1) -- use predfn to get x1 from x and y1
| _ -> false
```

or what we are really saying is:

```
if x=0 and y=0 then true
else if x!=0 and y!=0 then iseq(x-1,y-1)
else false
```

Q: How exactly does the iszero function express the comparison x=0?

A:

```
( \n. ((n (\x. false)) true ))      (\f. \x. f x) -- must be false (iszero of n)
^ iszero                               ^ 1
```

let's see how:

beta step gives

```
(\f. \x. f x) (\x. false) true
```

i.e. false

Now given "0" or (\f. \x. x) we will get "true"

basically anything that looks like f(...) will be treated as nonzero by the "iszero"

Q: How do we define 'and' (&&)?

A: Use 'and' encoding from Gordon!

'and P Q' can also be coded as

(if P then Q else false)

Asg 11:

1. One student could make the drawing work OK, as described below

I figured it out. I was creating the .dot file within bed and then converting it to ps outside of of bed. I added the correct flags that Lu put into the software when I generated the ps file and everything worked fine. Thanks for the help guys!

2. Another student who doubted T now thinks it's correct. But all of you must plz bombard the TTT example with sanity-check queries -- as if you were to write control software for an airplane, were required to test it, and then fly that same airplane :-)

3. Tips on extracting unreachable state info: If you obtain a BDD for reachable states, then complementing that BDD gives you a compact description of unreachable states. There are a large (often exp) number of sat assignments for the unreachable states. There is no point printing them all. One can fish out some example assignments using anysats. If you want to fish more unsat assignments, you can always do this: get one unsat assignment, negate it, and conjoin it with the unreachable states BDD. That BDD will then yield different anysats results.

Use logical formulae as "cookie cutters" and cut out aspects of BDDs. It is this practice that you can obtain using BED, and throughout this assignment.

4. For the Ripple Carry Adder, it is a good exercise to see how the BDDs for the lower order bits are used to build-up those for the higher order bit BDDs. See if you can find out how this sharing occurs. If BED were to have a command to view multiple BDDs in one "view", then one would see these bits building-up on each other.

5. The sentence 'using BED, show that win1 and draw (two example configurations) are implied by the reachable set of states of the tic-tac-toe game' in Question 9 of Asg11 is loosely worded.

I would reword it as

'Using BED, show that win1 and draw are in the set of reachable states.'

In a logic implication sense, it would be that $\text{win1} \Rightarrow \text{rs}$, and $\text{draw} \Rightarrow \text{rs}$.

The general direction I was suggesting was to debug T as well as the definition of win1, win2, draw, etc.

You may also take this opportunity to *improve* my definitions.

For instance, I had such a clunky definition of ‘nbless.’ Can you think of a neater way of saying the same thing?

6. --- question on $\text{wina1} \Rightarrow \text{rs}$ ---

>Perhaps I'm phrasing my sanity checks incorrectly -- at any rate, I'm
>definitely not getting results that I expect. For example, shouldn't it
>be the case that $\text{winX} \Rightarrow \text{rs}$ is always true? My first thought was, "oh,
>maybe I need to tack on the nbless condition to prevent anomalous TTT
>states," but that didn't bring me any closer. I also thought that the
>allmoved condition might be necessary, but this could cause anomalous
>games where both X and O looks like they won. So ... to the point:
>
>Shouldn't "let $\text{assert0} = \text{wina1} \Rightarrow \text{rs}$ " be true?
>

wina1 is not constraining enough. It is OK as far as my usage goes, because I use it only to define draw (by negating wina1).

But wina1 used un-negated does not constrain many rows/columns of the TTT game board (you can see which, very easily).

You can fix it by adding additional conjuncts to truly make it a reachable win condition.

General / Concepts: Bijection (p 40): >In the book says there is no bijection $\text{Nat} \rightarrow [0, 1]$

>since bijections are closed under composition.

>I don't understand this part.

>

>If $f : \text{Nat} \rightarrow [0, 1]$ in Real and

> $g : [0, 1] \rightarrow \text{Nat}$

>then $g(f(x)) = x'$, where x' is in Nat.

>Now, the bijection is closed under composition.

>How can I show there is no bijection with this conclusion?

>

That just means that

If we have a bij from $\text{Nat} \rightarrow [0,1]$

Then since we have a bij from $[0,1]$ to Real we would , thru comp of bij, have a bij from Nat to Real.

BUT since we will show there can't be a bij from Nat to $[0,1]$, even the first step of your journey is going to be broken and so all is well.

OK?

Announcements Pertaining to Lecture 5: • The directory `/home/ganesh/grail` had a permission problem. I've fixed that.

- Someone asked me about what the use of `Y` achieves. It allows recursive definitions to be re-expressed as a definitional equality, i.e., it allows the LHS to be defined in terms of an RHS that does not contain the name we are recursively defining.
- The f_1 , f_2 , and f_3 functions are “pulled out of the hat.” There is no general method for computing such closed-form solutions.
- The unstarred questions are to give you practice looking at other questions. If I did not give unstarred questions, one will not even look at additional problems. Those who try the unstarred questions now have an edge: I may include some unstarred questions in my take-home exams, and those individuals have an edge who have seriously thought of, or done the unstarred.
- Note the pattern of homeworks: I will assign something on Tuesdays and discuss it on Thursdays, making it due the next Tuesday. So it is an excellent habit for you to
 - Read things pertaining to a Tuesday lecture
 - Come with questions for the Thu lecture (and even request me to work out problems on Thu)
- Also if you are beginning to get lost, now is the time to ask for help. We are there to help you during office hours. After next week, I'll allow walk-ins to my office (next week I have paper deadlines, but after that, walk-ins are OK - after checking my availability thru email).
- Michael Gordon's book “Programming Language Theory and its Implementation” has chapters on Lambda calculus. I've linked a PDF of that book against today's lecture.
- I've linked in Jason Hickey's book on Ocaml (for those who need additional resources to study some of the rudiments of this language) against today's lecture.
- I've also linked in today's class terminal session online.

Lambda Calulus and scoping: >

```
>Hi,  
>  
>I was reading through the free and bound variable substitution, when I got  
>the idea that this actually is the static scoping language.  
>  
sure it is. Lambdas model this.
```

However note that Lambdas came *before* statically scoped languages (Lambdas came in the 40's - Algol-like languages came in the 60's and learned from lambda calculus).

Also note that McCarthy created Lisp based largely on Lambda calculus. Lisp programmers do use Lambdas quite extensively and in Lambda exs of Lisp, the scoping rules work the same way

Ganesh

```
> For example :
>
>(lambda x. add (x (lambda x. add (x x)))4)6)
>
>This gives :
>
> add (6 (lambda x.add(x x)4))
>add 6 8
>
>14.
>
>So, the value if x used in the second lambda expression is not the same as
>that of x for outer lambda. Ofcourse, we can use the alpha rule and change
>the inner lambda expression to makes things simple to see.
>
> Surely. That's how it works.
```

```
>
>But this is just my understanding. I hope I am not making some major
>conceptual mistake.
>
> There are even Lambda expressions that do not use the Lambda binding variable.
> This is called the de Brujin indices approach. This notation completely avoids
> the above kind of scoping problem. Instead of using variables, the de brujin
> indices record the number of lambda abstraction layers between the usage site
> and the lambda-binding site.
```

An excellent PPT sequence that presents this idea is in
www.cs.iastate.edu/~lumpe/ComS541/Resources/Lectures/LambdaCalculus.pdf

Ganesh

Inductively Defined Sets: >>Hi,

```
>>
>>You have provided three rules for inductively defined sets in page 81. The
>>third rule says that "S is the least such set". I have a question. we
>>already have a basis set. The remaining elements can be derived from the
>>basis elements using the set of constructors. In that case, the least set
>>should always be the basis set.
>>
>>If what I am saying is incorrect, I would like to see an example where the
>>set S consists of elemnts that cannot be derived from the basis set ....
>>
>>
```

This style of definition is standard in mathematics.

One could put "garbage" in a set to begin with -- then put the basis -- then close everything under constructors. Such a set will be larger than the least set.

Another way this is done in math is by giving 1 and 2 and instead of saying "S is the least such set" , they say "S is the intersection of all such sets". Even this is tantamount to saying "S is the least set" because when you perform intersections, the garbage gets removed

E.g.

Set with {g} in it, where g is garbage

Then put 0

Then put succ(e) for every element e in the set

This set will have {0,succ(0),succ(succ(0)),... ,succ(g),succ(succ(g)),...}

The least set containing 0 and closed under succ (rules 1,2,3) is

{0,succ(0),succ(succ(0)),...}

Frontier of a tree: It is the sequence made up of the leaves. If I used "length of the set of leaves" it won't be precise - hence I'm referring to the leaves as a *sequence* through the use of the word *frontier* and then taking its length.

Transitive closure (p60): Since the definition of R^+ comes on Page 60 which is before Chapter 6 which talks about recursion, the erratum suggests that we add the remark " R^+ is the least such set" as is traditional in math.

So the way to solve for such "recursion" is to imagine the smallest set (starting from the empty set) and working one's way up.

Such a least set will include R , as it turns out. You can do this: plug in \emptyset for occurrences of R^+ on the RHS, and get the R^+ on the LHS; then plug this R^+ back on the RHS and obtain the R^+ on the LHS, and so on. This series terminates (stabilizes, or reaches fixed point) at the least such R^+ .

Another more useful way of defining R^+ would have been to define a series of relations:

- R^0 as of now
- R^1 as

$$R^1 = \{\langle x, z \rangle \mid \exists y \in S : \langle x, y \rangle \in R \wedge \langle y, z \rangle \in R^0\}.$$

- R^2 as

$$R^2 = \{\langle x, z \rangle \mid \exists y \in S : \langle x, y \rangle \in R \wedge \langle y, z \rangle \in R^1\}.$$

- R^k as

$$R^k = \{\langle x, z \rangle \mid \exists y \in S : \langle x, y \rangle \in R \wedge \langle y, z \rangle \in R^{k-1}\}.$$

- R^+ as

$$R^+ = \cup_{k \geq 0} R^k$$

Into: I did not define “into” on P.26 fully.

Into really means 1-1 and not necessarily onto.

The URL <http://mathforum.org/library/drmath/view/52454.html> mentions this.

Cardinality and Bijection: Same cardinality “if” there is a bijection.

Really it is “if and only if.” However, as is common, when we define something new, we say “Foo if ...bar...” However in all such cases, the “if” is to be taken as “if and only if.”

Definitional equality is mentioned on Page 76.

Cardinality of C programs:

I have a question about the cardinality of C program proof using Schroder-Bernstein Theorem.

The mapping from $\text{Nat} \rightarrow \text{CP}$ as total and into is something I am able to appreciate. Because there are C programs that are not covered which use other C programming syntactic constructs.

But the mapping the claim that mapping from $\text{CP} \rightarrow \text{Nat}$ is 1-1 and into is what I am unable to appreciate. For the example of class of C programs that is given in the text book, this is alright. But what about more complex C programs? We can have two C programs having the same cardinality but which are different.

Take any C program in the full syntax of C. It is a sequence of ASCII characters. Concatenate all these ASCII chars, and read that bit-string as a natural number. This number (also called the “Godel number”) differs for any two C programs that are different (e.g., for which “diff” would show the difference). So we do get a different Godel number for each C program in its full syntax.

Also “we can have two C programs having the same cardinality but which are different” is not correct to say – we are talking about the cardinality of the *set of all* C programs according to some syntax.

Partial and Total Functions: I wonder that if $(x^2)/x = x$.

I was thinking that x^2/x is a partial 1-1 and onto function from \mathbb{N} to \mathbb{N} , since if $x = 0$, then it cannot be defined, otherwise yes.

But If it is same as x , then it can be defined.

So what should I think of it?

If you allow x to be in Nat , then obviously the simplification $(x^2/x) = x$ is not correct. In other words, $\lambda x.(x^2)/x$ is partial over $x \in \text{Nat}$ but total over

$$\{x \in \text{Nat} \mid x \geq 1\}$$

2 FAQs on Tools

Let me begin with Lu’s new README file.

2.1 Lu's README File for BED

Two Ways to run BED Tools

1. I Use it directly in CS machines:

- (a) Log into your CS account, add the following two lines into your `.bash_profile` file and source it.

```
export PATH=$PATH:/home/luzhao/course/6100/grail/bin:/home/luzhao/course/6100/bed-  
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/luzhao/course/6100/bed-2.5/lib
```

If you use C shell family, use the following syntax in your `.tcshrc` or `.cshrc`

```
set PATH = ($PATH path_to_bed_bin) set LD_LIBRARY_PATH = ($LD_LIBRARY_PATH path_to_  
and source it.
```

- (b) Then you're ready to go by typing "bed". It's said better to use "ssh -Y" option to your cs account for X11 forwarding if you don't directly sit in front of a CS machine.

2. Compile and Install it in your own linux home directory

- (a) Download it at <http://www.itu.dk/research/bed/>. Sometime it's unreachable, so you need to check it periodically until it's online.

- (b) Compile. The compilation instruction is in its README file, and it's pretty easy to follow. If your gcc is 3.3, there should be any problem. However, if your gcc is 4.1, you may get a compiler error, which is caused by a stricter checking in 4.1. Then you need to change the `is_push` macro defined in `libutil/istack.h` into a function. Jeremy rewrote the function as the following:

```
inline int is_push( iStack *s, int e )  
{  
    if( s->top == s->arr_end ) {  
        *is_expand( s ) = e;  
    } else {  
        *(s->top++) = e;  
    }  
    return e;  
}
```

Then your gcc 4.1 should be able compile it.

2.2 Lu's README File for Grail

Three ways to use grail tools.

I. USE Grail TOOLS on CADE Machine

1. Login to any CADE linux box by using `lab1-*.eng.utah.edu` or `best-linux.eng.utah.edu`.

2. Add the following environment variables in your `.bash_profile` or `.csh`,
I'm using bash syntax here.

```
export FA2GRAIL_PERL=/home/zha/course/6100/grail/bin/fa2grail.perl
export GRAIL2PS_PERL=/home/zha/course/6100/grail/bin/grail2ps.perl
export PATH=/home/zha/course/6100/grail/bin:$PATH
```

3. Update your shell environment by "source ~/.bash_profile" or the corresponding csh command.

4. Run test piped command at your own working directory:

```
fa2grail.perl /home/zha/course/6100/grail/bin/fa.txt |fmdeterm |perl $GRAIL2PS_PERL - |gv -
echo "a*" | retofm | fmdeterm | fmmin | perl $GRAIL2PS_PERL - |gv -
```

you should see different state machines. If you want to save the output ps file, use redirection symbol > to put it into a file like the following:

```
echo "a*" | retofm | fmdeterm | fmmin | perl $GRAIL2PS_PERL - > adfa.ps
```

5. If you want customize the output, like save the generated dot file, make a copy of \$GRAIL2PS_PERL into your home directory and comment out the line of

```
system("/bin/rm -rf ".$arg.".dot");
```

and use your own pathname in the perl command.

II. USE Grail TOOLS on CS Machine

1. Login to your CS account.

2. Add the following environment variables in your .bash_profile or .csh, I'm using bash syntax here.

```
export FA2GRAIL_PERL=/home/luzhao/course/6100/grail/bin/fa2grail.perl
export GRAIL2PS_PERL=/home/luzhao/course/6100/grail/bin/grail2ps.perl
export PATH=/home/luzhao/course/6100/grail/bin:$PATH
```

3. Update your shell environment by "source .bash_profile" or the corresponding csh command.

4. Run test piped command at your own working directory:

```
fa2grail.perl /home/luzhao/course/6100/grail/bin/fa.txt |fmdeterm |perl $GRAIL2PS_PERL - |ggv -
echo "a*" | retofm | fmdeterm | fmmin | perl $GRAIL2PS_PERL - |ggv -
```

you should see different state machines. If you want to save the output ps file, use redirection symbol > to put it into a file like the following:

```
echo "a*" | retofm | fmdeterm | fmmin | perl $GRAIL2PS_PERL - > adfa.ps
```

5. If you want customize the output, like save the generated dot file, make a copy of \$GRAIL2PS_PERL into your home directory and comment out the line of

```
system("/bin/rm -rf ".$arg.".dot");
```

and use your own pathname in the perl command.

III. INSTALL YOUR OWN COPY of GRAIL TOOLS

by downloading it at <http://www.cs.utah.edu/ganesh-comp-engg-book/grail.tar.gz>. It's a precompiled version for linux. It's almost ready to go after decompression. You need to check the command locations in the two perl scripts to fit your system.

See Section 2.3 for Ocaml and other tools to be used in Chapter 6, Section 2.4 for BED, Grail, and JFLAP (usage begins around Chapter 8), and 2.5 for Spin, and TLC.

2.3 Tools for Chapter 6

In Chapter 6, I hope to present to you a Lambda reducer written using the functional language Ocaml. This also illustrates many of the concepts you have studied, including signatures.

Ocaml has been installed on the EMCB 220 Windows machines, apparently. However, the easiest is to get Ocaml from <http://caml.inria.fr>. It installs very easily, both on Windows and on Linux machines. It's one of the world's most productivity-enhancing as well as efficient programming languages!

We will need Ocaml in about a week (with respect to 1/21/07).

2.4 Checklist to ensure you can run Grail, BED, and JFLAP

We will soon begin using multiple tools. In order to be able to use them, please go through this checklist by the class of 1/25/07. If you go through this checklist successfully, you are good to go as far as Grail and BDDs (using the BED tool) go.

We will be needing BED and Grail in about a week (with respect to 1/21/07).

1. Do you have a machine you can install software on, and prefer doing so? If so, please skip these initial steps (but do read them because you'll find URLs and/or instructions for download and installation in this sequel).
2. Students taking classes in SoC are eligible for CADE machine accounts. For further info on getting accounts, contact opers@eng.utah.edu or visit them in EMCB 210.
3. Once you get an account, make sure you can log on to a CADE Linux machine. An example would be `lab4-5.eng.utah.edu`. Often, the following command finds for you the "best" Linux machine:
`ssh -Y best-linux.eng.utah.edu`
4. Once you login to such a machine, type `/home/ganesh/BED`
5. If you get a prompt, try this sequence of commands:

```
> var a b c
> let d = a and b and c
> upall d
> view d
> (Here, you may get an error message. Fear not.)
> quit
```

Then come out and look at your home directory using, say, ‘ls -lt — more’.

It will show at the top of the listing a pdf file. The name of the pdf file will be number_number.pdf. Anyway, copy that PDF file and view it. You should see a BDD for a 3-input AND-gate at this point! A BDD compactly represents a Boolean function (see Chapter 11).

6. The ‘view’ command calls ‘gv’ or ‘ggv’ (ghostview), and if found, displays the BDD directly
7. ops@eng tell me that ‘gv’ is installed at /usr/X11R6/bin/gv. So you should be able to see the BDD directly if your path setting is right. I’ve tried it just now and it works.
8. Note: The Binary for BED I’ve kept runs only on “x86” machines (otherwise known as Linux machines). However, BED is available for download and installation from www.itu.dk/research/bed
9. I used BED in CS 3700 to avoid lengthy circuit simulations. If you are interested, see www.cs.utah.edu/classes/cs3700/handouts/11:2-14
The BDD tutorial there can be quite useful.
10. Now install and/or try out the **grail** tools discussed from Chapter 8 onwards. Read on.
11. Connect to (change directory to) /home/ganesh/grail (again on a CADE Linux machine such as lab4-5.eng.utah.edu). Try these commands.
12. **Please do not run “grail”. That is not the executable. It is the commands in the pipeline below (plus more) that comprise the grail suite of tools.**
13.

```
echo "a*" | retofm | fmdeterm | fmmin | perl grail2ps.perl - | ps2pdf - > /tmp/test1.pdf  
xpdf /tmp/test1.pdf
```

You should see a DFA for the regular language **a***. Again, should ‘gv’ or ‘ggv’ become available, you can type instead

```
echo "a*" | retofm | fmdeterm | fmmin | perl grail2ps.perl - | ggv -
```

as the commands on Pages 177 and 180 illustrate.

14. The Grail binaries are available from my book’s website <http://www.cs.utah.edu/ganesh-comp-engg-book>
Again, if this version experiences a crash, it is perhaps because ‘dot’ could not be found in your current directory (./dot).
In the version of grail2ps.perl provided in /home/ganesh/grail, I’ve typed ‘dot’ instead of ‘./dot’ and so you may try making this change in the grail you download from my book’s website.
15. JFLAP may be downloaded from <http://www.jflap.org>. It is extremely useful for understanding DFA, NFA, PDA, and most importantly non-deterministic Turing machines. JFLAP is thoroughly documented on its website. There is even a textbook describing JFLAP.

16. If the above commands work fine, you are good to go as far as Grail, BDDs (using the BED tool), and JFLAP go. If not, plz send teach-cs6100 a message describing the part you are having trouble with.

2.5 Tools to be used later

Right now, I have included TLC in the list of “tools to be used later.” You can take 3-4 weeks (with respect to 1/21/07) to set this tool up (and the TA will help you with the tool installation as well). There is no real hurry here.

TLC is described (and available) at <http://research.microsoft.com/users/lamport/tla/book.html>