CSE 351

Programming Languages

Term Project

Deadline: Monday, January 12, 2026 @12:00

In this project, you are going to implement the dead code elimination algorithm and transform any given intermediate language code.

Intermediate Language (IL) Syntax

The intermediate language (or the intermediate code) can be in the form of one or multiple assignment statements in the following syntax:

```
a = b + c;

c = d * 3;

b = d;

e = 5;

{ a, b }
```

The syntax of an assignment statement is very similar to an assignment statement in C programming language. The only difference is that each statement may have at most two source operands, which might be in the form of either a variable or a signed integer constant. In this language, three or more source operands are not supported. Therefore, the following statement is not accepted as a correct assignment statement in this intermediate language.

```
a = b + c * d;
```

Also note that the last line in a given IL source code has a special format:

```
{variable1, variable2, variable3}
```

This line lists all the live variables when the code execution reaches to the end of the source code. You can assume there might be at least 1 variable and at most 5 live variables in this list.

Dead Code Elimination: This algorithm can eliminate assignment statements with no consumers or assignment statements of variables that are not live at the end of the code execution. For example,

```
b = z + y;
a = b;
x = 2 * b;
{ x }
```

can be transformed into

```
b = z + y;
x = 2 * b;
```

There are three major steps of the dead code elimination algorithm:

1. Reverse the code
2. Apply dead code elimination
3. Reverse the code.

For example, let's assume the given input IL source code is as follows:

```
a=2+2;
b=2^9;
c=d^3;
e=5;
f=3*4;
g=6/2;
h=m;
p=0;
j=j+p;
r=e*p;
s=a;
{ r, s }
```

Step 1. Reverse the IL code: <u>You may do this by integrating a preprocessing step, in which you can run a shell script or another code that can do the job.</u>

```
{ r, s }
s=a;
r=e*p;
j=j+p;
p=0;
h=m;
g=6/2;
f=3*4;
e=5;
c=d^3;
b=2^9;
a=2+2;
```

Step 2. Apply dead code elimination: After each statement that you visit, remember the new set of live variables, and if a destination variable of an assignment statement is dead, do not print that statement.

First, we read the first line, and set our initial LiveSet set to include all the listed live variables.

Reverse IL source code	Updated LiveSet	Output code
{ r, s }	{ r, s }	

After you visit the first (actually, it is the last) IL statement "s=a;", we already know that s is live, and, now, to set the value of variable s we need the value of variable a. Then, we conclude that variable a must be live. But, variable s becomes live at this line, therefore, it must be dead before this last statement. As a result, we remove variable s from the live set of variables. Therefore, after processing "s=a;", we update our current live set of variables to { r, a }.

Reverse IL source code	Updated LiveSet	Output code
{ r, s } s=a;	{ r, a }	s=a;

Next, we reach to statement “r=e*p;”. Now, we know that variables r and a are both live. Since, variable r is live, then, both variables e and p must also be live to calculate the value of variable r at this point. Therefore, we also include these variables in the live variables’ set. But, variable r becomes live at this line, therefore, it must be dead before this statement. We remove variable r from the live set of variables.

Reverse IL source code	**Updated LiveSet**	**Output code**

{ s }
s=a;
r=e*p;
	{ a, e, p }	s=a;
		r=e*p;

Next, we reach to statement “j=j+p;”. Now, we know that variables a, e, and p are all live. But, j is not live (i.e. it is dead). Since, variable j is dead that means this statement is dead code, and we do not process it further, and we do not print it.

Reverse IL source code	**Updated LiveSet**	**Output code**

{ s }
s=a;
r=e*p;
j=j+p;
	{ a, e, p }	s=a;
		r=e*p;

Next, we reach to statement “p=0;”. Now, we know that variables a, e and p are all live, and we print this statement. But, since we set the value of variable p here, it must be dead before this statement. Therefore, we remove variable p from the live set.

Reverse IL source code	**Updated LiveSet**	**Output code**

{ s }
s=a;
r=e*p;
j=j+p;
p=0;
	{ a, e }	s=a;
		r=e*p;
		p=0;

Next, we reach to statement “h=m;”. Now, we know that variables a and e are both live. But, h is not live. Since, variable h is dead that means this statement is dead code, and we do not print it.

Reverse IL source code	**Updated LiveSet**	**Output code**

{ s }
s=a;
r=e*p;
j=j+p;
p=0;
h=m;
	{ a, e }	s=a;
		r=e*p;
		p=0;

Next, we reach to statement “g=6/2;”. Now, we know that variables a and e are both live. But, g is not live. Since, variable g is dead that means this statement is dead code, and we do not print it.

Reverse IL source code	Updated LiveSet	Output code

{ s }
s=a;
r=e*p;
j=j+p;
p=0;
h=m;
g=6/2;
	{ a, e }	s=a;
		r=e*p;
		p=0;

Next, we reach to statement “f=3*4;”. Now, we know that variables a and e are both live. But, f is not live. Since, variable f is dead that means this statement is dead code, and we do not print it.

Reverse IL source code	Updated LiveSet	Output code

{ s }
s=a;
r=e*p;
j=j+p;
p=0;
h=m;
g=6/2;
f=3*4;
	{ a, e }	s=a;
		r=e*p;
		p=0;

Next, we reach to statement “e=5;”. Now, we know that variables a and e are both live, we print this statement. But, e is set to 5 here. That means it is dead before this statement. Since, variable e is dead, we remove it from the live variables’ set.

Reverse IL source code	Updated LiveSet	Output code

{ s }
s=a;
r=e*p;
j=j+p;
p=0;
h=m;
g=6/2;
f=3*4;
e=5;
	{ a }	s=a;
		r=e*p;
		p=0;
		e=5;

Next, we reach to statement “c=d^3;”. Now, we know that only variable a is live. But, c is not live. Since, variable c is dead that means this statement is dead code, and we do not print it.

Reverse IL source code	Updated LiveSet	Output code

{ s }
s=a;
r=e*p;
j=j+p;
p=0;
h=m;
g=6/2;
f=3*4;
e=5;
c=d^3;
	{ a }	s=a;
		r=e*p;
		p=0;
		e=5;

Next, we reach to statement “b=2^9;”. Now, we know that only variable a is live. But, b is not live. Since, variable b is dead that means this statement is dead code, and we do not print it.

Reverse IL source code	Updated LiveSet	Output code

{ s }
s=a;
r=e*p;
j=j+p;
p=0;
h=m;
g=6/2;
f=3*4;
e=5;
c=d^3;
b=2^9;
	{ a }	s=a;
		r=e*p;
		p=0;
		e=5;

Finally, we reach to statement “a=2+2;”. Now, we know that only variable a is live, we print this statement. But, a is set to 2+2 here. That means it is dead before this statement. Since, variable a is dead, we remove it from the live variables’ set.

Reverse IL source code	Updated LiveSet	Output code

{ s }
s=a;
r=e*p;
j=j+p;
p=0;
h=m;
g=6/2;
f=3*4;
e=5;
c=d^3;
b=2^9;
a=2+2;
	{ }	s=a;
		r=e*p;
		p=0;
		e=5;
		a=2+2;

Since, there is no more lines in the input IL code, we move to the next step in the algorithm.

Step 3. Reverse the Output IL code:

```
a=2+2;
e=5;
p=0;
r=e*p;
s=a;
```

Project Requirements:

You are expected

1) to write a lex and a yacc code for the **dead code elimination** algorithm,
2) to write a term project report explaining your solution.

Your final project submission should contain lex and yacc source files, sample input files, scripts or command line statements to compile and run your project.
Your report should include a detailed explanation of how you designed and implemented your project.