Register Allocation, iii

Bringing in functions & using spilling & coalescing

Function Calls

```
;; f(x) = let y = g(x)
;;            in h(y+x) + y*5
(:f
 (x <- eax)    ;; save our argument
 (call :g)     ;; call g with our argument
 (y <- eax)    ;; save g's result in y
 (eax += x)    ;; compute h's arg
 (call :h)     ;; call h
 (y5 <- y)     ;; compute y*5 in y5, i
 (y5 *= 5)     ;; compute y*5 in y5, ii
 (eax += y5)   ;; add h's res to y*5
 (return))     ;; and we're done.
```

Gen & Kill

		gen	kill
1:	`:f`	()	()
2:	`(x <- eax)`	(eax)	(x)
3:	`(call :g)`	(eax ecx edx)	(ebx ecx edx eax)
4:	`(y <- eax)`	(eax)	(y)
5:	`(eax += x)`	(eax x)	(eax)
6:	`(call :h)`	(eax ecx edx)	(ebx ecx edx eax)
7:	`(y5 <- y)`	(y)	(y5)
8:	`(y5 *= 5)`	(y5)	(y5)
9:	`(eax += y5)`	(eax y5)	(eax)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	()	()
2:	`(x <- eax)`	()	()
3:	`(call :g)`	()	()
4:	`(y <- eax)`	()	()
5:	`(eax += x)`	()	()
6:	`(call :h)`	()	()
7:	`(y5 <- y)`	()	()
8:	`(y5 *= 5)`	()	()
9:	`(eax += y5)`	()	()
10:	`(return)`	()	()

Liveness

		in	out
1:	`:f`	()	()
2:	`(x <- eax)`	(eax)	()
3:	`(call :g)`	(eax ecx edx)	()
4:	`(y <- eax)`	(eax)	()
5:	`(eax += x)`	(eax x)	()
6:	`(call :h)`	(eax ecx edx)	()
7:	`(y5 <- y)`	(y)	()
8:	`(y5 *= 5)`	(y5)	()
9:	`(eax += y5)`	(eax y5)	()
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	()	(eax)
2:	`(x <- eax)`	(eax)	(eax ecx edx)
3:	`(call :g)`	(eax ecx edx)	(eax)
4:	`(y <- eax)`	(eax)	(eax x)
5:	`(eax += x)`	(eax x)	(eax ecx edx)
6:	`(call :h)`	(eax ecx edx)	(y)
7:	`(y5 <- y)`	(y)	(y5)
8:	`(y5 *= 5)`	(y5)	(eax y5)
9:	`(eax += y5)`	(eax y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax)	(eax)
2:	`(x <- eax)`	(eax ecx edx)	(eax ecx edx)
3:	`(call :g)`	(eax ecx edx)	(eax)
4:	`(y <- eax)`	(eax x)	(eax x)
5:	`(eax += x)`	(eax ecx edx x)	(eax ecx edx)
6:	`(call :h)`	(eax ecx edx y)	(y)
7:	`(y5 <- y)`	(y)	(y5)
8:	`(y5 *= 5)`	(eax y5)	(eax y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax)	(eax ecx edx)
2:	`(x <- eax)`	(eax ecx edx)	(eax ecx edx)
3:	`(call :g)`	(eax ecx edx)	(eax x)
4:	`(y <- eax)`	(eax x)	(eax ecx edx x)
5:	`(eax += x)`	(eax ecx edx x)	(eax ecx edx y)
6:	`(call :h)`	(eax ecx edx y)	(y)
7:	`(y5 <- y)`	(y)	(eax y5)
8:	`(y5 *= 5)`	(eax y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax ecx edx)	(eax ecx edx)
2:	`(x <- eax)`	(eax ecx edx)	(eax ecx edx)
3:	`(call :g)`	(eax ecx edx x)	(eax x)
4:	`(y <- eax)`	(eax ecx edx x)	(eax ecx edx x)
5:	`(eax += x)`	(eax ecx edx x y)	(eax ecx edx y)
6:	`(call :h)`	(eax ecx edx y)	(y)
7:	`(y5 <- y)`	(eax y)	(eax y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax ecx edx)	(eax ecx edx)
2:	`(x <- eax)`	(eax ecx edx)	(eax ecx edx x)
3:	`(call :g)`	(eax ecx edx x)	(eax ecx edx x)
4:	`(y <- eax)`	(eax ecx edx x)	(eax ecx edx x y)
5:	`(eax += x)`	(eax ecx edx x y)	(eax ecx edx y)
6:	`(call :h)`	(eax ecx edx y)	(eax y)
7:	`(y5 <- y)`	(eax y)	(eax edi esi y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax ecx edx)	(eax ecx edx)
2:	`(x <- eax)`	(eax ecx edx)	(eax ecx edx x)
3:	`(call :g)`	(eax ecx edx x)	(eax ecx edx x)
4:	`(y <- eax)`	(eax ecx edx x)	(eax ecx edx x y)
5:	`(eax += x)`	(eax ecx edx x y)	(eax ecx edx y)
6:	`(call :h)`	(eax ecx edx y)	(eax y)
7:	`(y5 <- y)`	(eax edi esi y)	(eax edi esi y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax ecx edx)	(eax ecx edx)
2:	`(x <- eax)`	(eax ecx edx)	(eax ecx edx x)
3:	`(call :g)`	(eax ecx edx x)	(eax ecx edx x)
4:	`(y <- eax)`	(eax ecx edx x)	(eax ecx edx x y)
5:	`(eax += x)`	(eax ecx edx x y)	(eax ecx edx y)
6:	`(call :h)`	(eax ecx edx y)	(eax edi esi y)
7:	`(y5 <- y)`	(eax edi esi y)	(eax edi esi y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax ecx edx)	(eax ecx edx)
2:	`(x <- eax)`	(eax ecx edx)	(eax ecx edx x)
3:	`(call :g)`	(eax ecx edx x)	(eax ecx edx x)
4:	`(y <- eax)`	(eax ecx edx x)	(eax ecx edx x y)
5:	`(eax += x)`	(eax ecx edx x y)	(eax ecx edx y)
6:	`(call :h)`	(eax ecx edi edx esi y)	(eax edi esi y)
7:	`(y5 <- y)`	(eax edi esi y)	(eax edi esi y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax ecx edx)	(eax ecx edx)
2:	`(x <- eax)`	(eax ecx edx)	(eax ecx edx x)
3:	`(call :g)`	(eax ecx edx x)	(eax ecx edx x)
4:	`(y <- eax)`	(eax ecx edx x)	(eax ecx edx x y)
5:	`(eax += x)`	(eax ecx edx x y)	(eax ecx edi edx esi y)
6:	`(call :h)`	(eax ecx edi edx esi y)	(eax edi esi y)
7:	`(y5 <- y)`	(eax edi esi y)	(eax edi esi y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax ecx edx)	(eax ecx edx)
2:	`(x <- eax)`	(eax ecx edx)	(eax ecx edx x)
3:	`(call :g)`	(eax ecx edx x)	(eax ecx edx x)
4:	`(y <- eax)`	(eax ecx edx x)	(eax ecx edx x y)
5:	`(eax += x)`	(eax ecx edi edx esi x y)	(eax ecx edi edx esi y)
6:	`(call :h)`	(eax ecx edi edx esi y)	(eax edi esi y)
7:	`(y5 <- y)`	(eax edi esi y)	(eax edi esi y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax ecx edx)	(eax ecx edx)
2:	`(x <- eax)`	(eax ecx edx)	(eax ecx edx x)
3:	`(call :g)`	(eax ecx edx x)	(eax ecx edx x)
4:	`(y <- eax)`	(eax ecx edx x)	(eax ecx edi edx esi x y)
5:	`(eax += x)`	(eax ecx edi edx esi x y)	(eax ecx edi edx esi y)
6:	`(call :h)`	(eax ecx edi edx esi y)	(eax edi esi y)
7:	`(y5 <- y)`	(eax edi esi y)	(eax edi esi y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax ecx edx)	(eax ecx edx)
2:	`(x <- eax)`	(eax ecx edx)	(eax ecx edx x)
3:	`(call :g)`	(eax ecx edx x)	(eax ecx edx x)
4:	`(y <- eax)`	(eax ecx edi edx esi x)	(eax ecx edi edx esi x y)
5:	`(eax += x)`	(eax ecx edi edx esi x y)	(eax ecx edi edx esi y)
6:	`(call :h)`	(eax ecx edi edx esi y)	(eax edi esi y)
7:	`(y5 <- y)`	(eax edi esi y)	(eax edi esi y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax ecx edx)	(eax ecx edx)
2:	`(x <- eax)`	(eax ecx edx)	(eax ecx edx x)
3:	`(call :g)`	(eax ecx edx x)	(eax ecx edi edx esi x)
4:	`(y <- eax)`	(eax ecx edi edx esi x)	(eax ecx edi edx esi x y)
5:	`(eax += x)`	(eax ecx edi edx esi x y)	(eax ecx edi edx esi y)
6:	`(call :h)`	(eax ecx edi edx esi y)	(eax edi esi y)
7:	`(y5 <- y)`	(eax edi esi y)	(eax edi esi y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax ecx edx)	(eax ecx edx)
2:	`(x <- eax)`	(eax ecx edx)	(eax ecx edx x)
3:	`(call :g)`	(eax ecx edi edx esi x)	(eax ecx edi edx esi x)
4:	`(y <- eax)`	(eax ecx edi edx esi x)	(eax ecx edi edx esi x y)
5:	`(eax += x)`	(eax ecx edi edx esi x y)	(eax ecx edi edx esi y)
6:	`(call :h)`	(eax ecx edi edx esi y)	(eax edi esi y)
7:	`(y5 <- y)`	(eax edi esi y)	(eax edi esi y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax ecx edx)	(eax ecx edx)
2:	`(x <- eax)`	(eax ecx edx)	(eax ecx edi edx esi x)
3:	`(call :g)`	(eax ecx edi edx esi x)	(eax ecx edi edx esi x)
4:	`(y <- eax)`	(eax ecx edi edx esi x)	(eax ecx edi edx esi x y)
5:	`(eax += x)`	(eax ecx edi edx esi x y)	(eax ecx edi edx esi y)
6:	`(call :h)`	(eax ecx edi edx esi y)	(eax edi esi y)
7:	`(y5 <- y)`	(eax edi esi y)	(eax edi esi y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

Liveness

		in	out
1:	`:f`	(eax ecx edx)	(eax ecx edx)
2:	`(x <- eax)`	(eax ecx edi edx esi)	(eax ecx edi edx esi x)
3:	`(call :g)`	(eax ecx edi edx esi x)	(eax ecx edi edx esi x)
4:	`(y <- eax)`	(eax ecx edi edx esi x)	(eax ecx edi edx esi x y)
5:	`(eax += x)`	(eax ecx edi edx esi x y)	(eax ecx edi edx esi y)
6:	`(call :h)`	(eax ecx edi edx esi y)	(eax edi esi y)
7:	`(y5 <- y)`	(eax edi esi y)	(eax edi esi y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

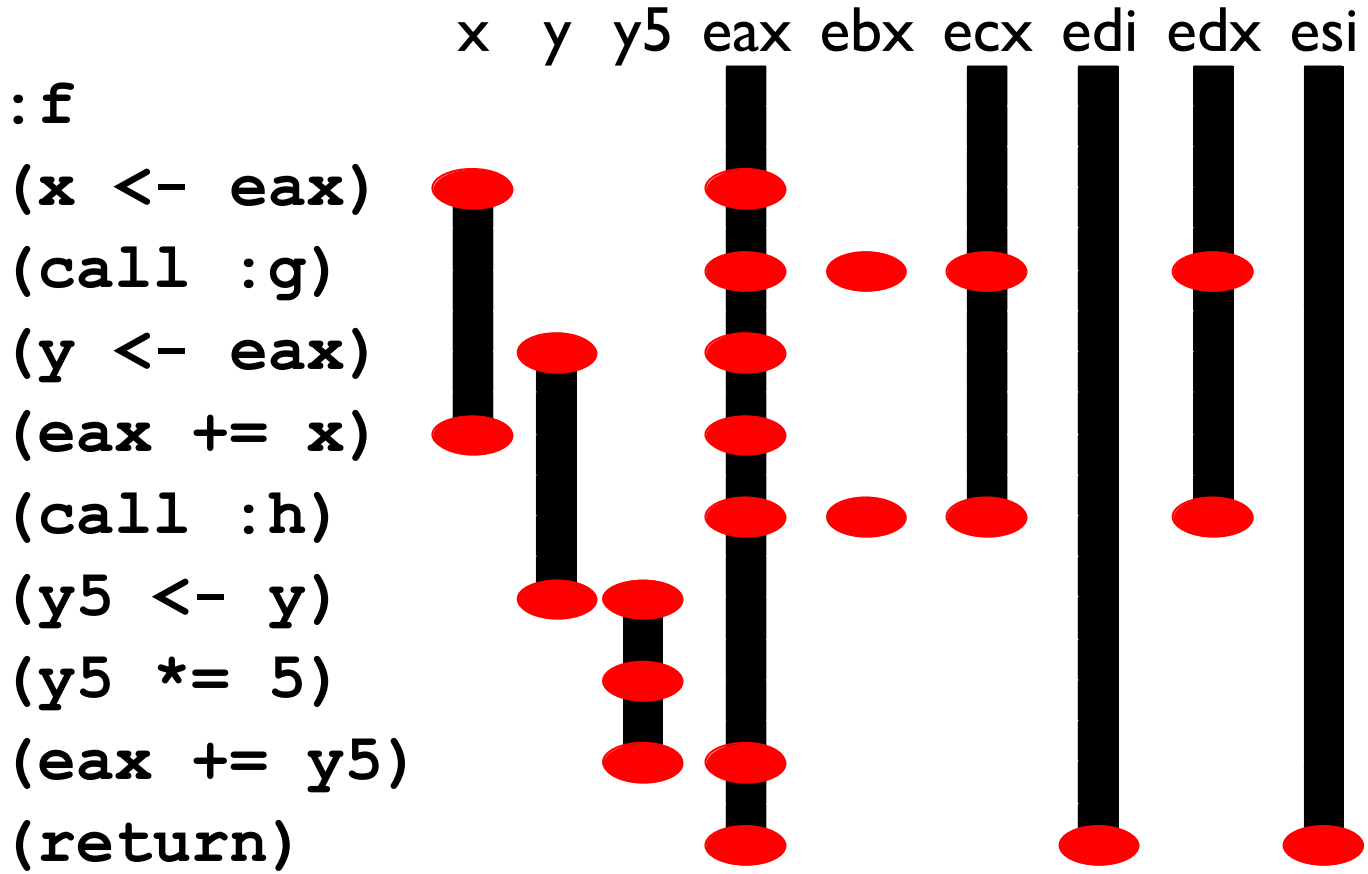
Liveness

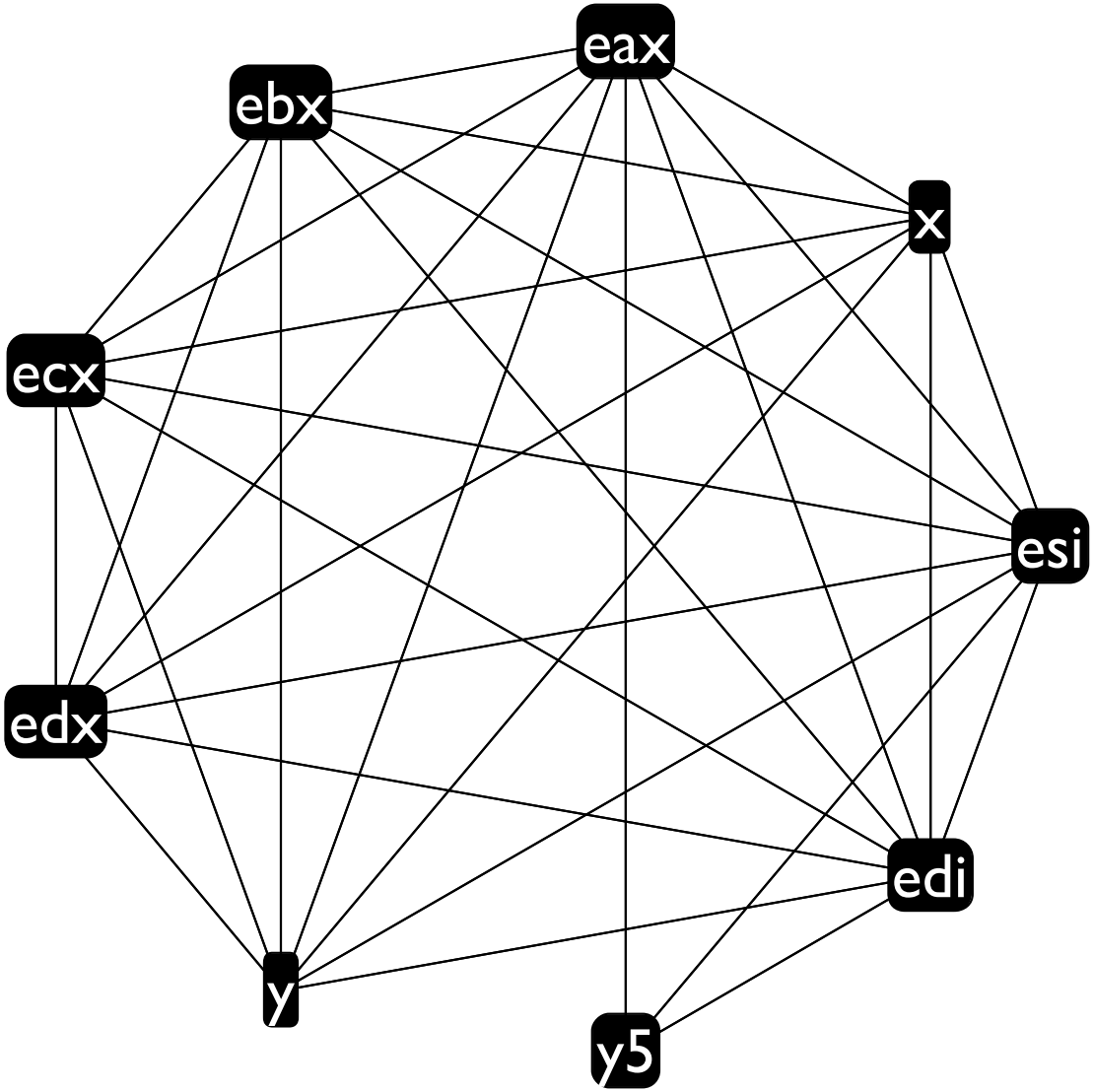
		in	out
1:	`:f`	(eax ecx edx)	(eax ecx edi edx esi)
2:	`(x <- eax)`	(eax ecx edi edx esi)	(eax ecx edi edx esi x)
3:	`(call :g)`	(eax ecx edi edx esi x)	(eax ecx edi edx esi x)
4:	`(y <- eax)`	(eax ecx edi edx esi x)	(eax ecx edi edx esi x y)
5:	`(eax += x)`	(eax ecx edi edx esi x y)	(eax ecx edi edx esi y)
6:	`(call :h)`	(eax ecx edi edx esi y)	(eax edi esi y)
7:	`(y5 <- y)`	(eax edi esi y)	(eax edi esi y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

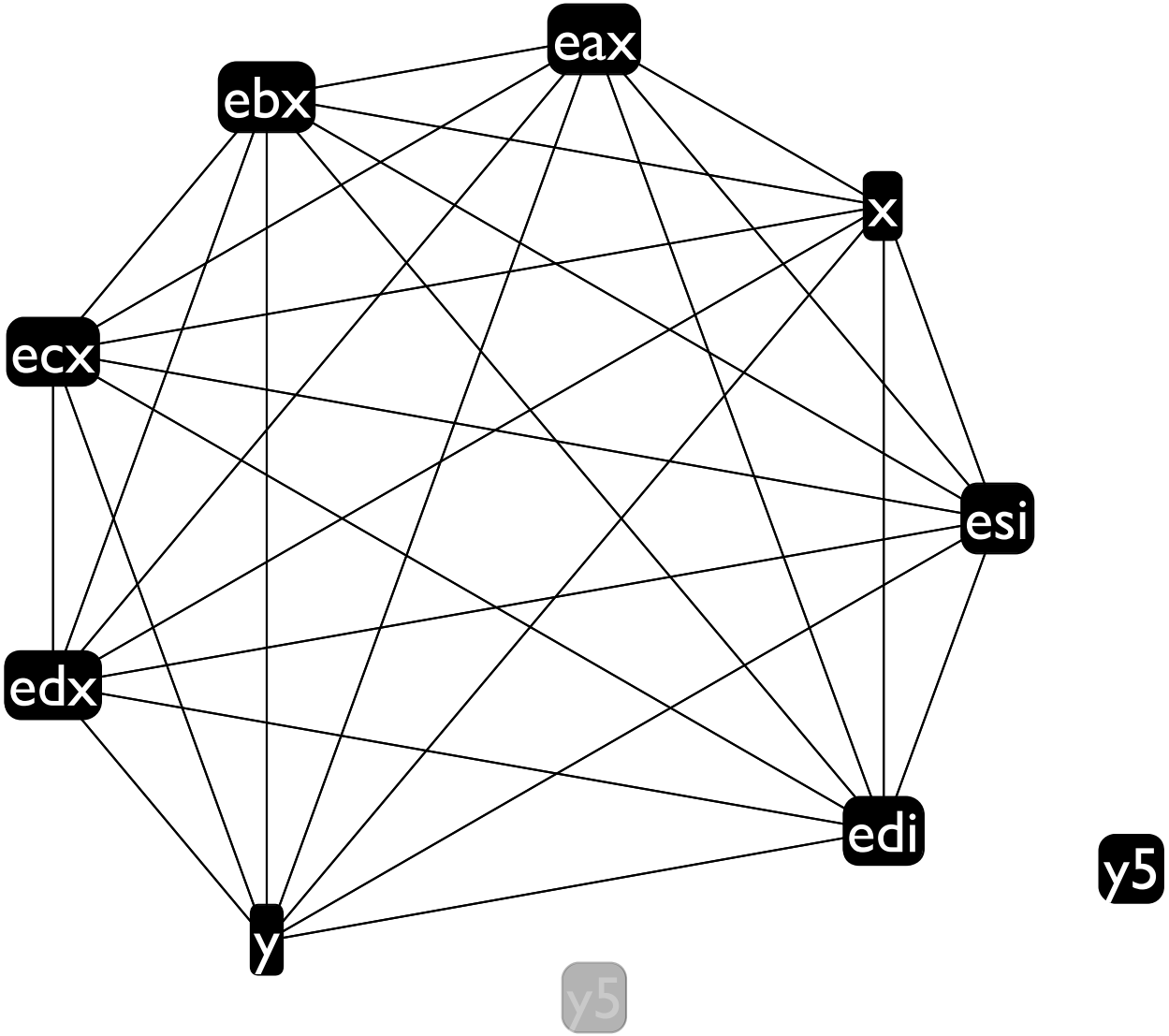
Liveness

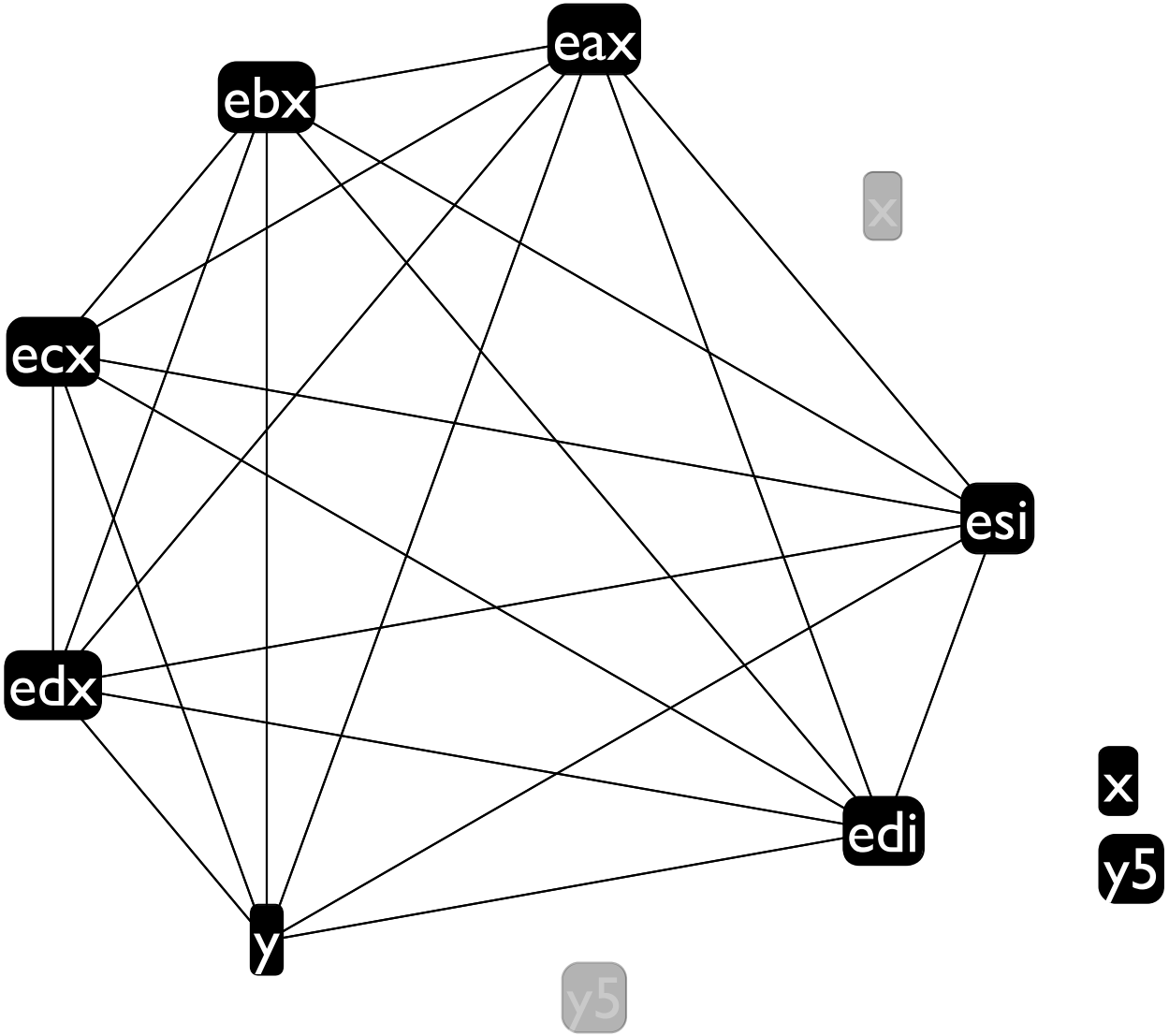
		in	out
1:	`:f`	(eax ecx edi edx esi)	(eax ecx edi edx esi)
2:	`(x <- eax)`	(eax ecx edi edx esi)	(eax ecx edi edx esi x)
3:	`(call :g)`	(eax ecx edi edx esi x)	(eax ecx edi edx esi x)
4:	`(y <- eax)`	(eax ecx edi edx esi x)	(eax ecx edi edx esi x y)
5:	`(eax += x)`	(eax ecx edi edx esi x y)	(eax ecx edi edx esi y)
6:	`(call :h)`	(eax ecx edi edx esi y)	(eax edi esi y)
7:	`(y5 <- y)`	(eax edi esi y)	(eax edi esi y5)
8:	`(y5 *= 5)`	(eax edi esi y5)	(eax edi esi y5)
9:	`(eax += y5)`	(eax edi esi y5)	(eax edi esi)
10:	`(return)`	(eax edi esi)	()

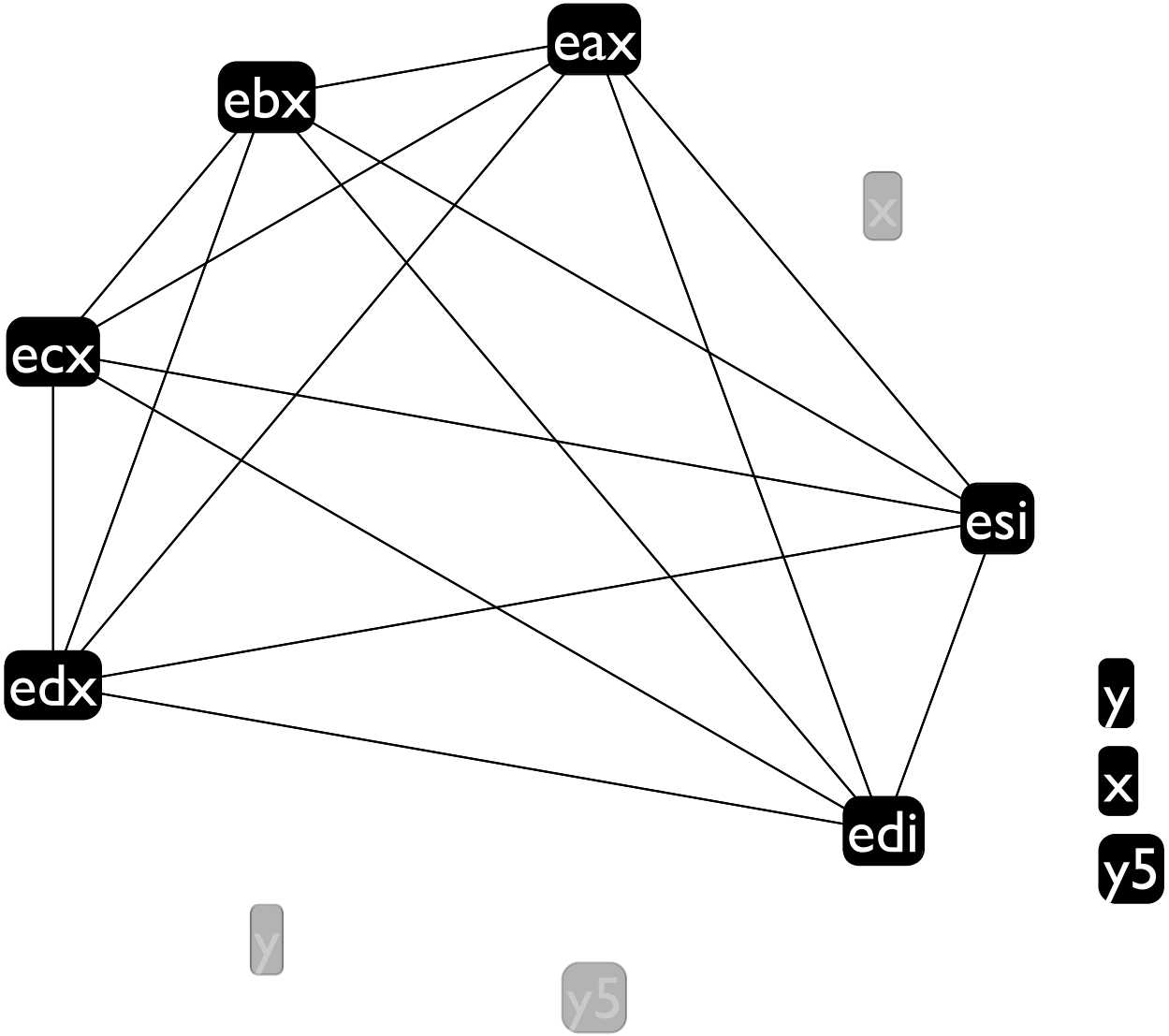
Liveness

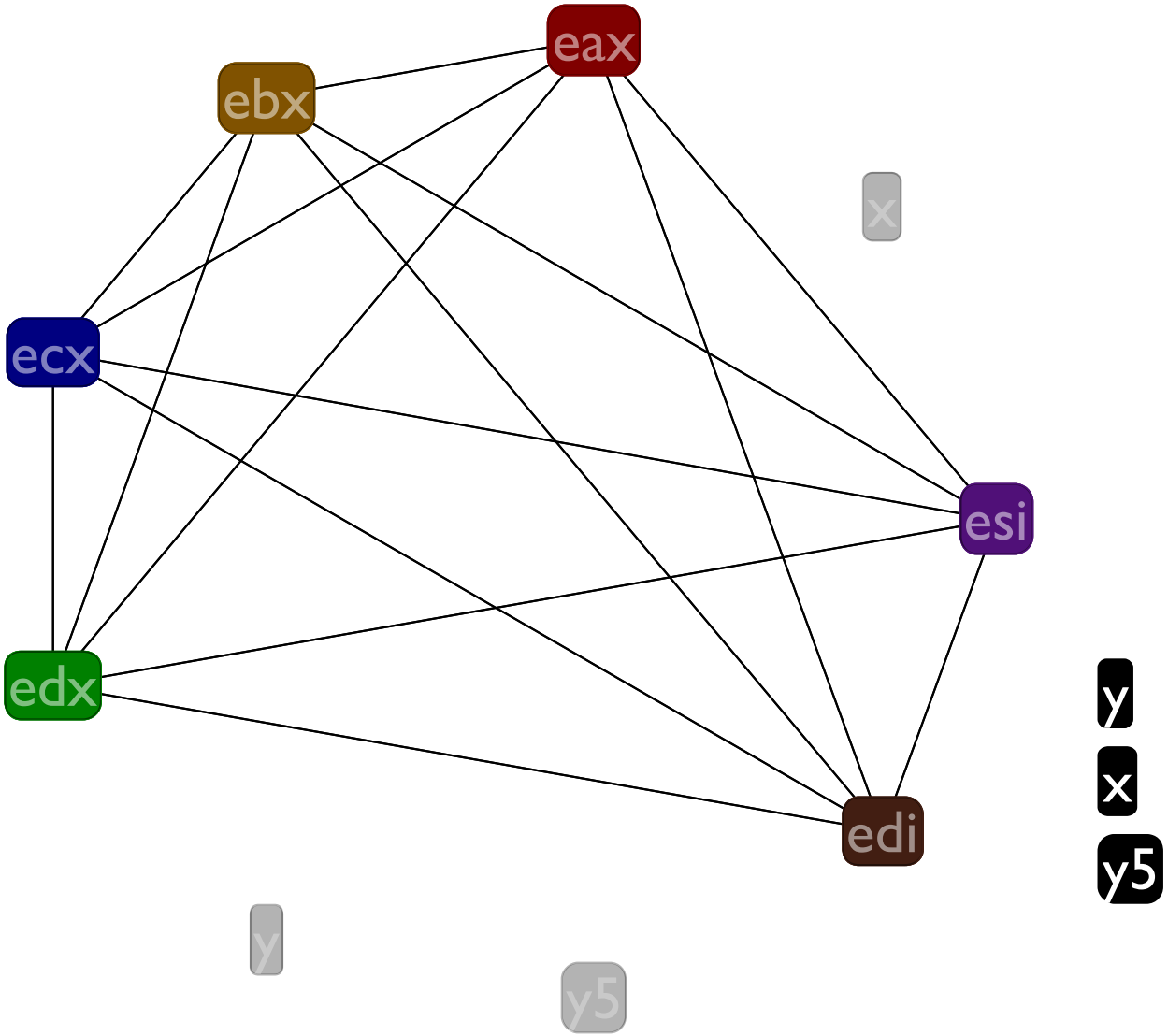

x y y5 eax ebx ecx edi edx esi
:f
(x <- eax)
(call :g)
(y <- eax)
(eax += x)
(call :h)
(y5 <- y)
(y5 *= 5)
(eax += y5)
(return)

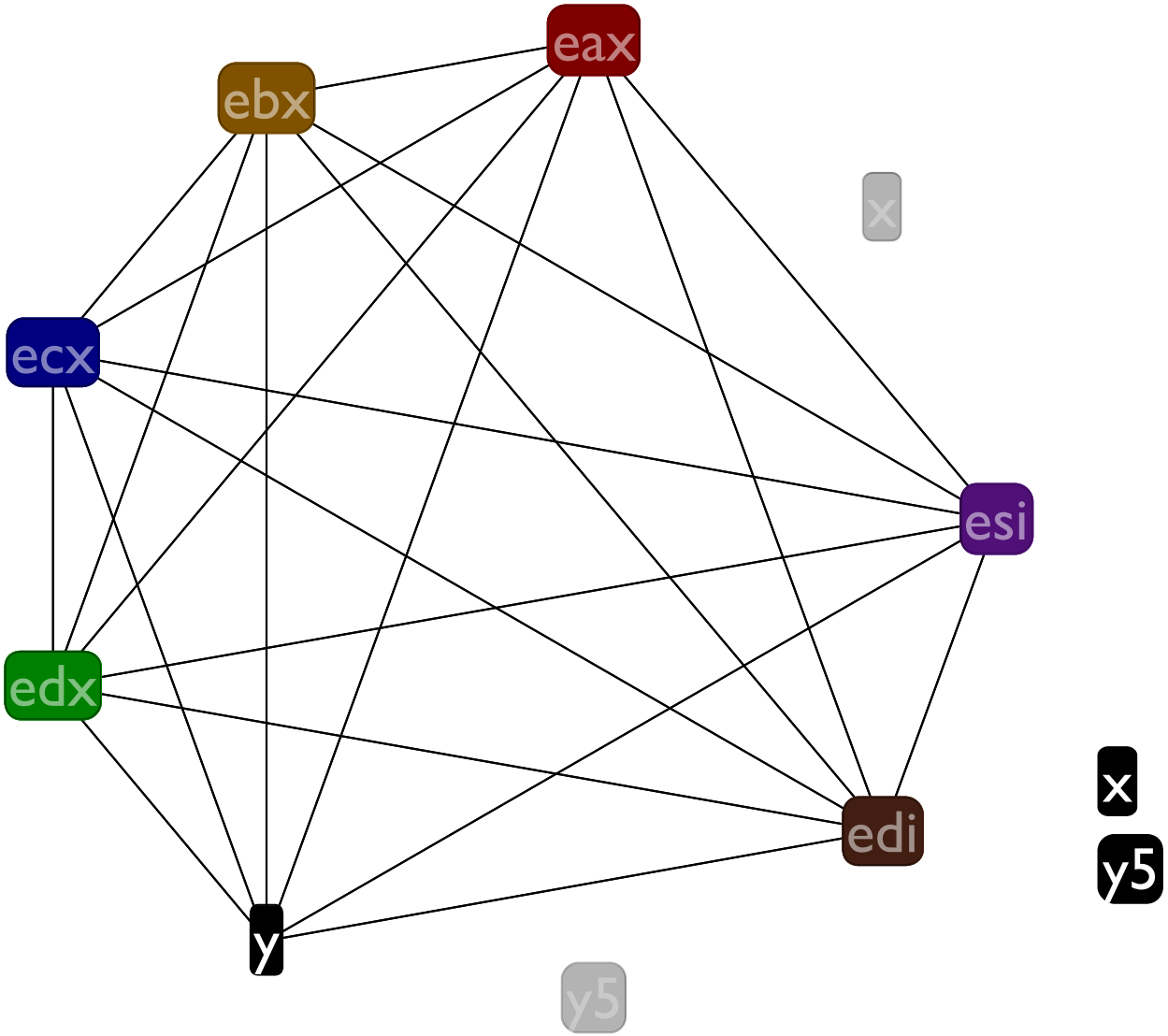

eax
ebx
x
ecx
esi
edx
edi
y
y5

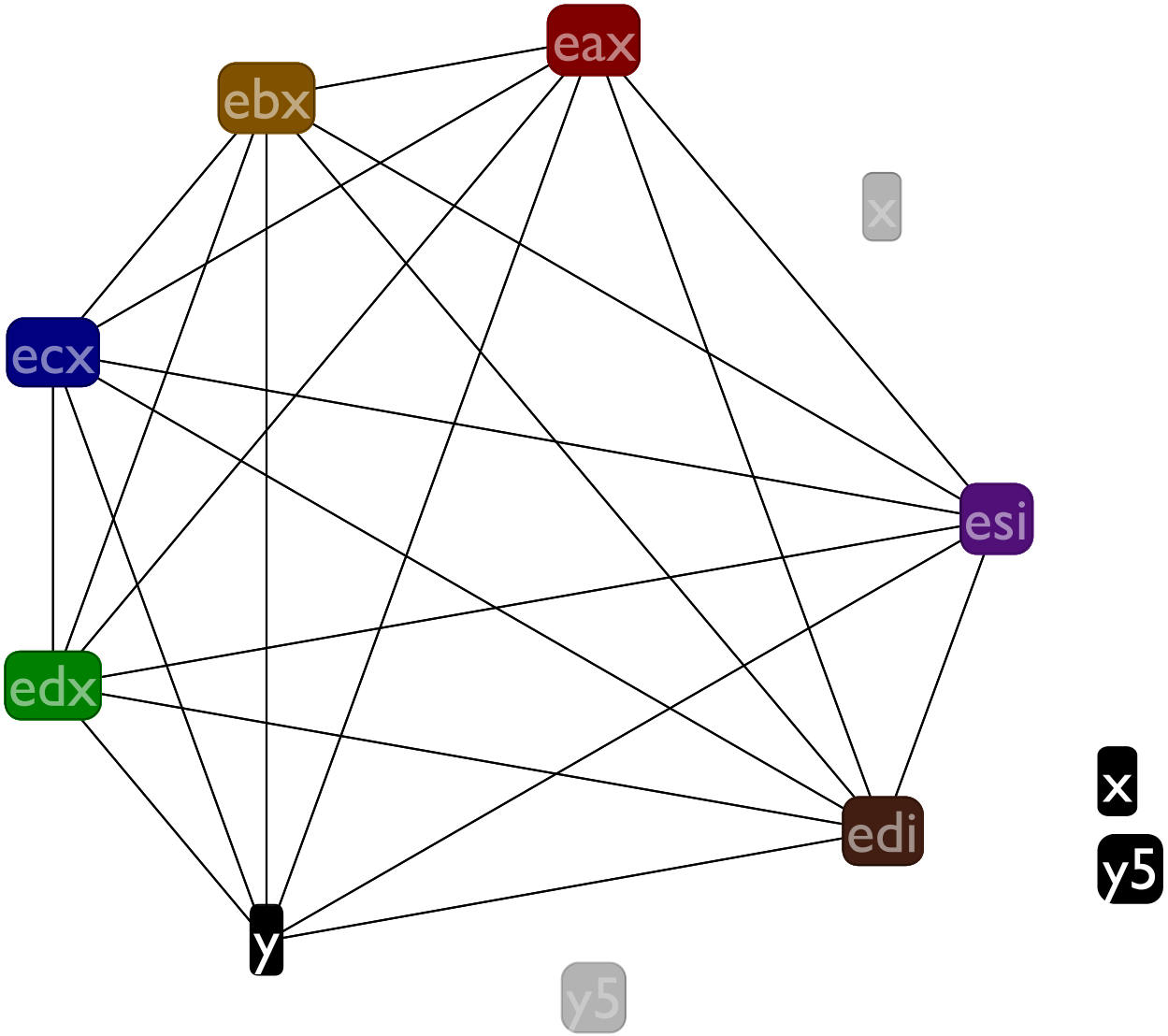

eax
ebx
x
ecx
esi
edx
edi
y5
y
y5

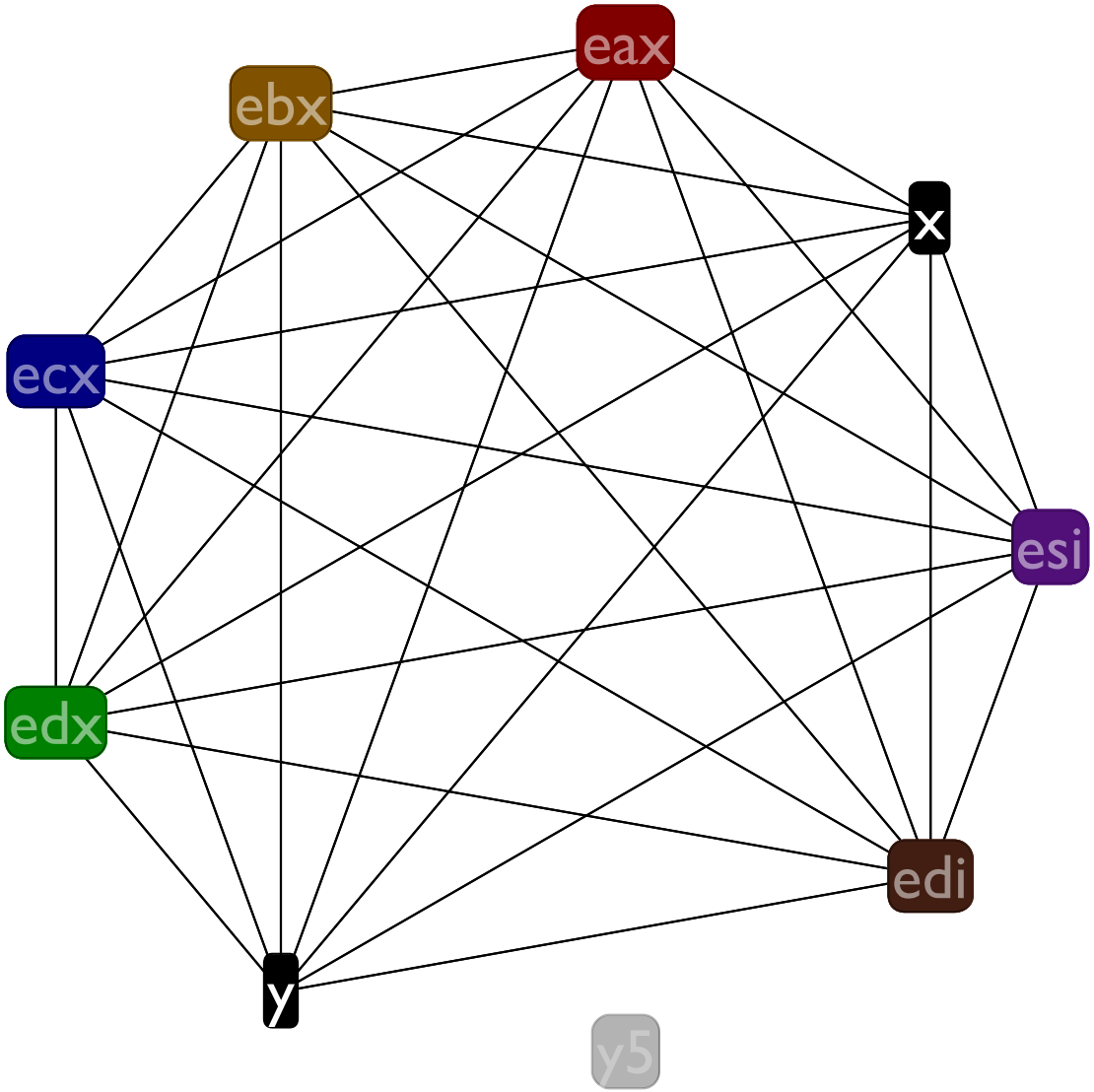

eax
ebx
x
ecx
esi
edx
edi
x
y5
y
y5

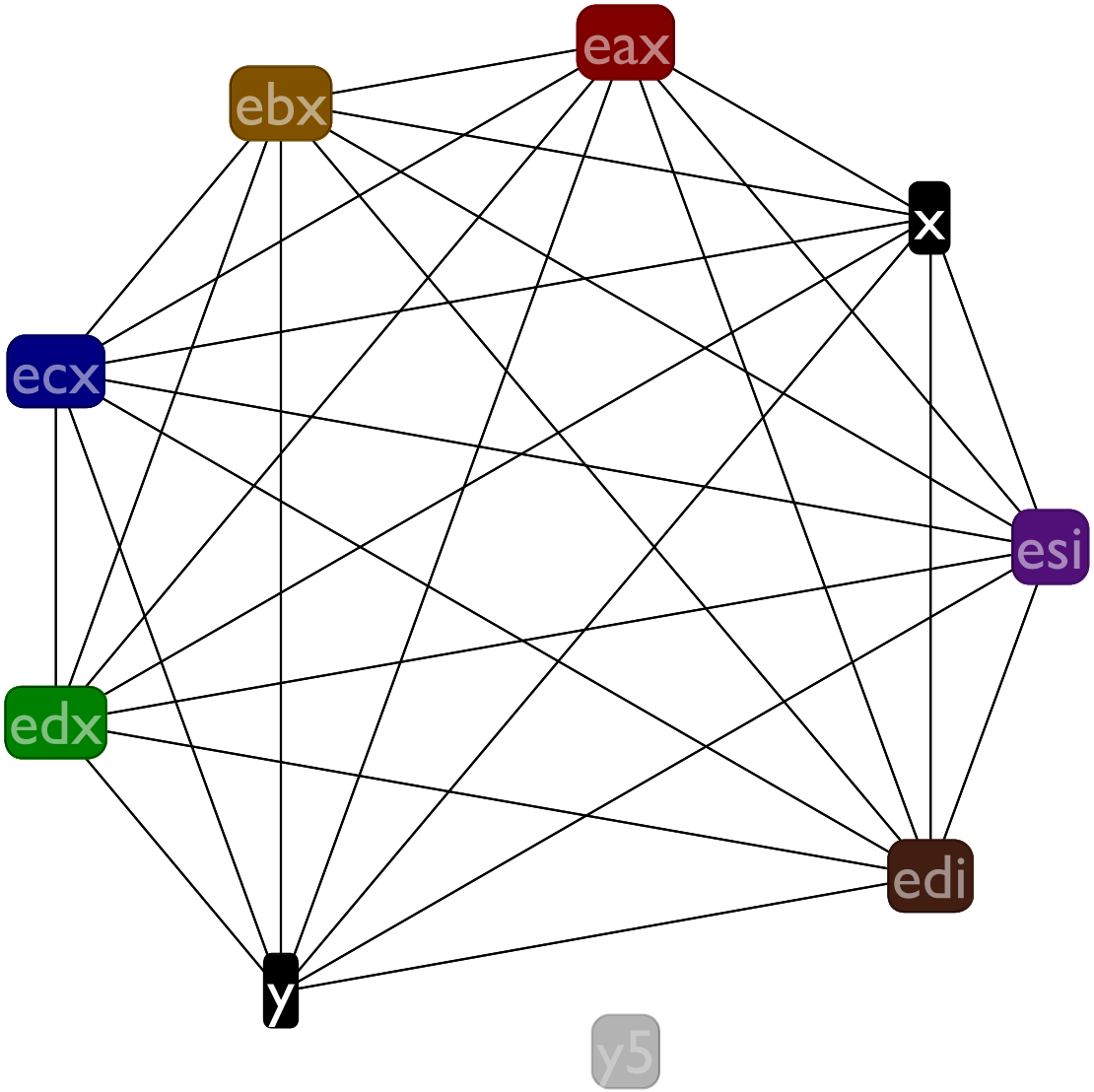

eax
ebx
x
ecx
esi
edx
y
x
y5
edi
y
y5

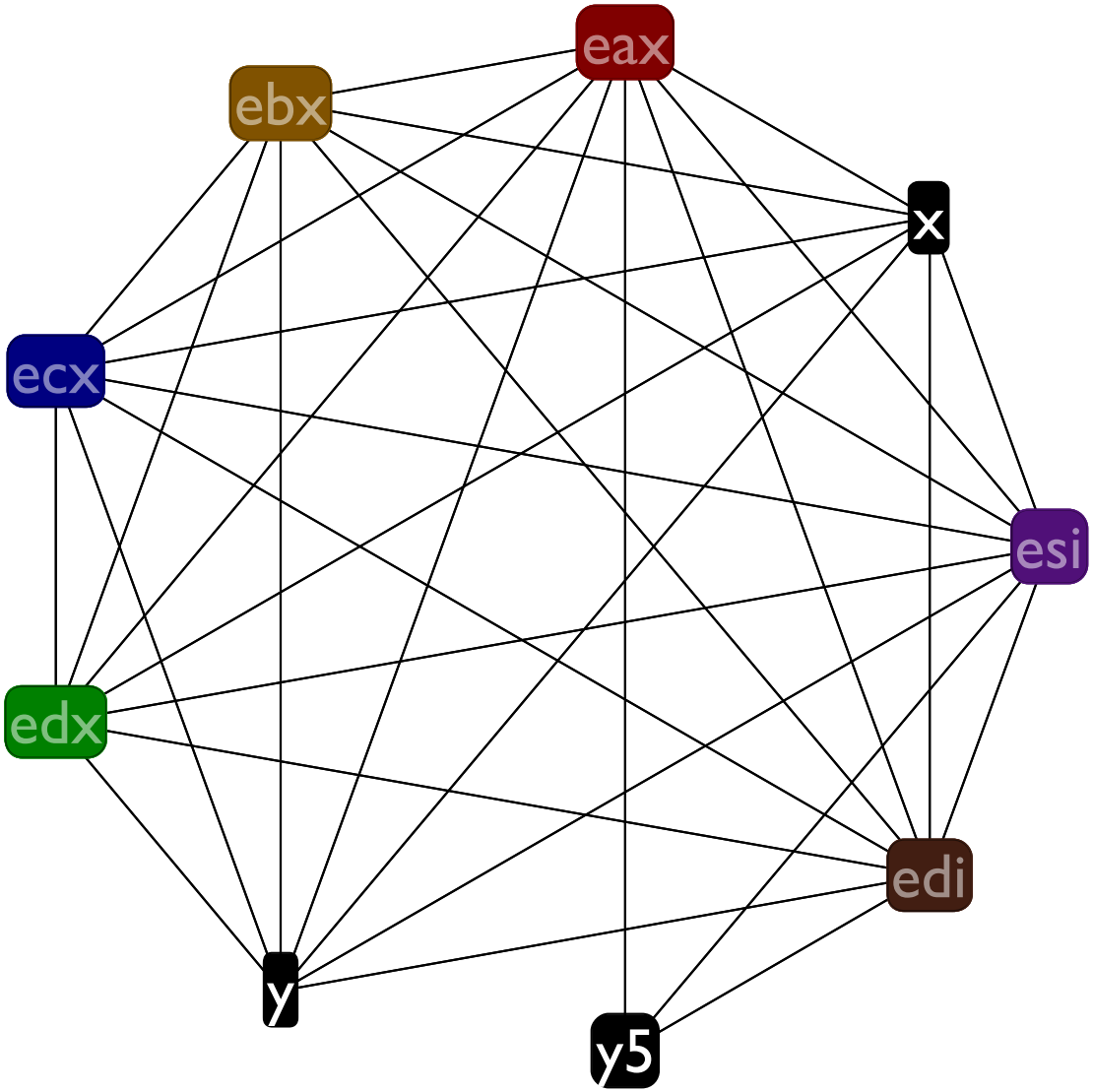

eax
ebx
x
ecx
esi
edx
edi
y
x
y5
y
y5

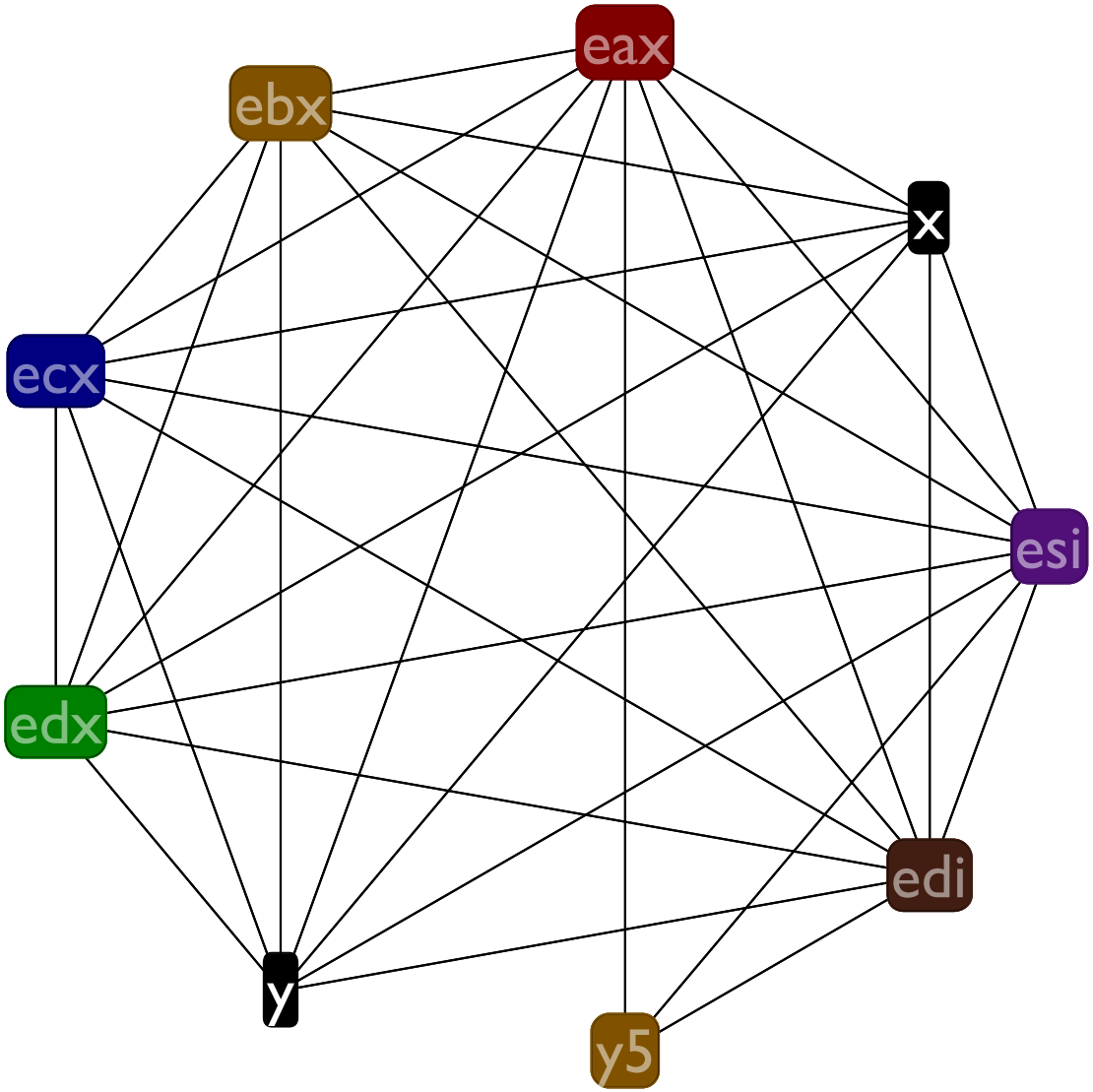

eax
ebx
x
ecx
esi
edx
edi
y
y5
x
y5


eax
ebx
x
ecx
esi
edx
edi
y
y5
x
y5


eax
ebx
x
ecx
esi
edx
edi
y
y5
y5


eax
ebx
x
ecx
esi
edx
edi
y
y5
y5


eax
ebx
x
ecx
esi
edx
edi
y
y5


eax
ebx
x
ecx
esi
edx
edi
y
y5

Spilling y

Before:

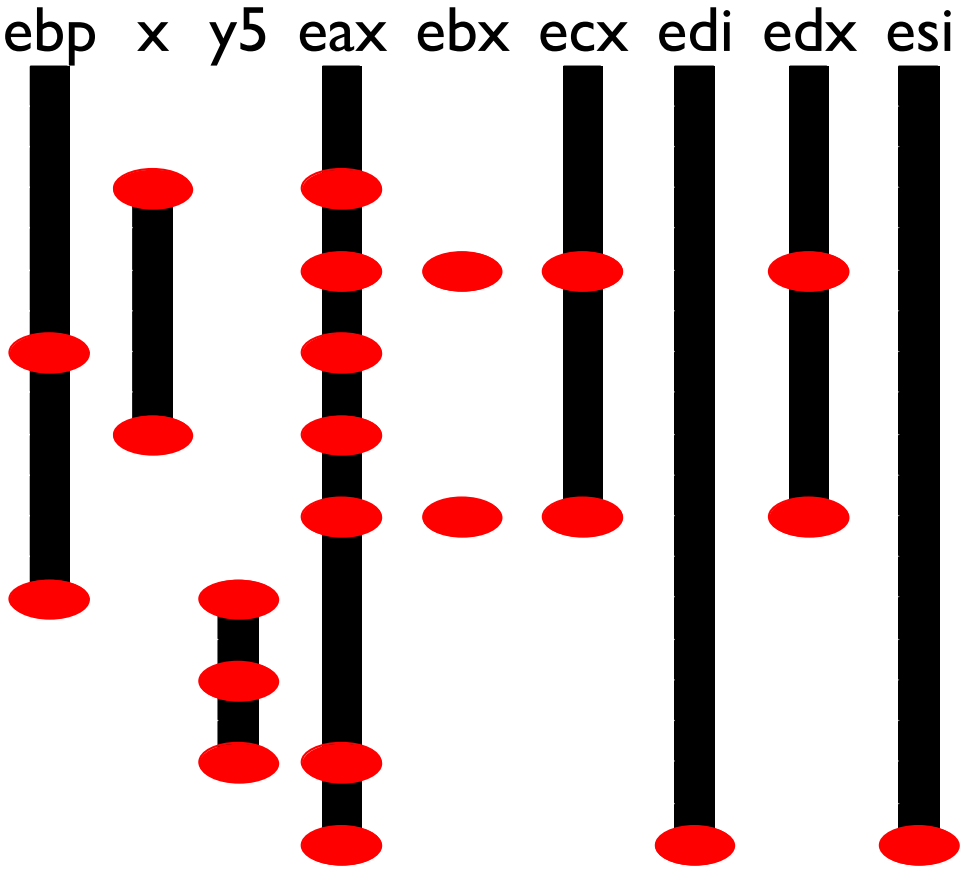
```
:f
(x <- eax)
(call :g)
(y <- eax)
(eax += x)
(call :h)
(y5 <- y)
(y5 *= 5)
(eax += y5)
(return)
```

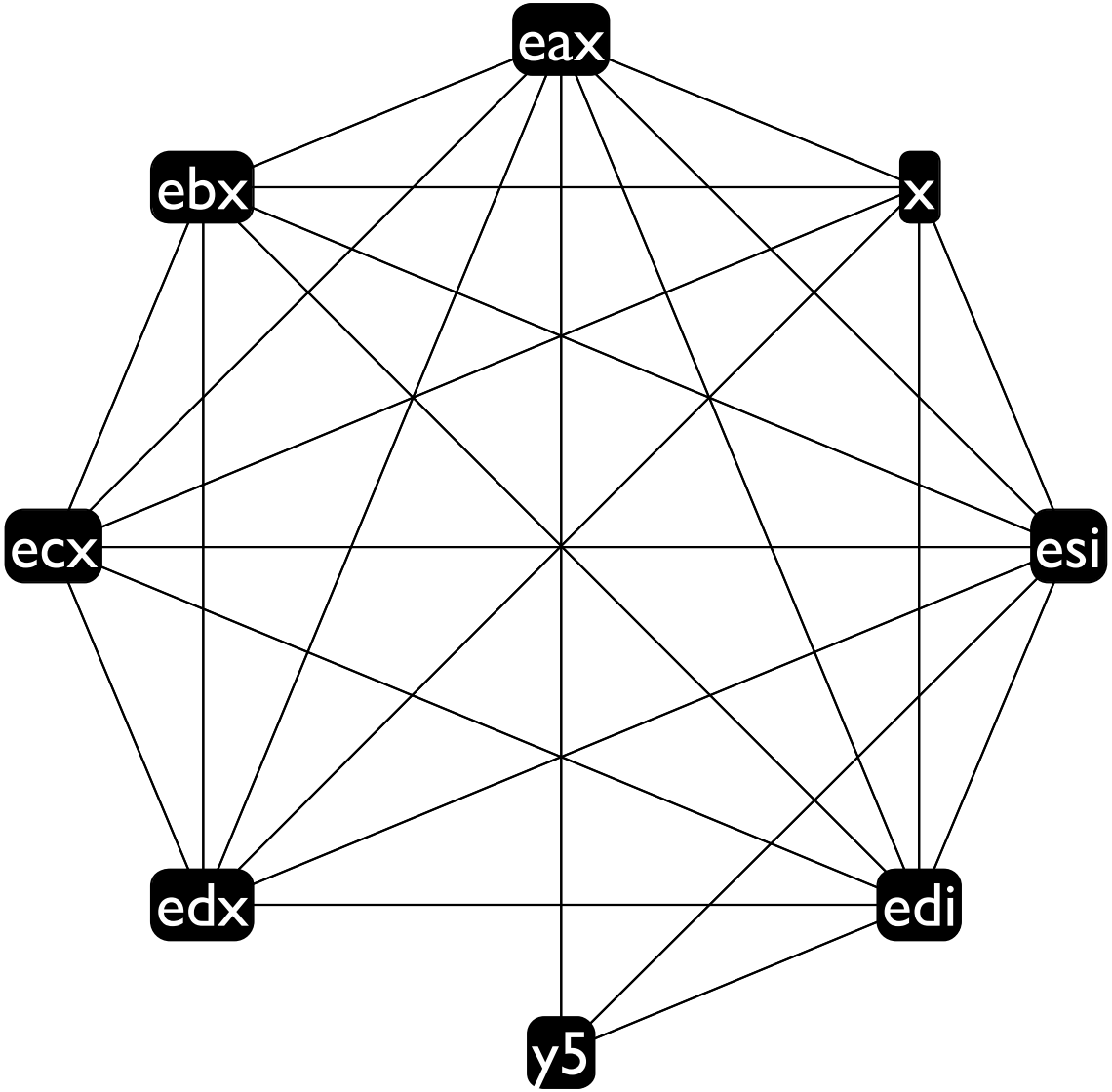
After:

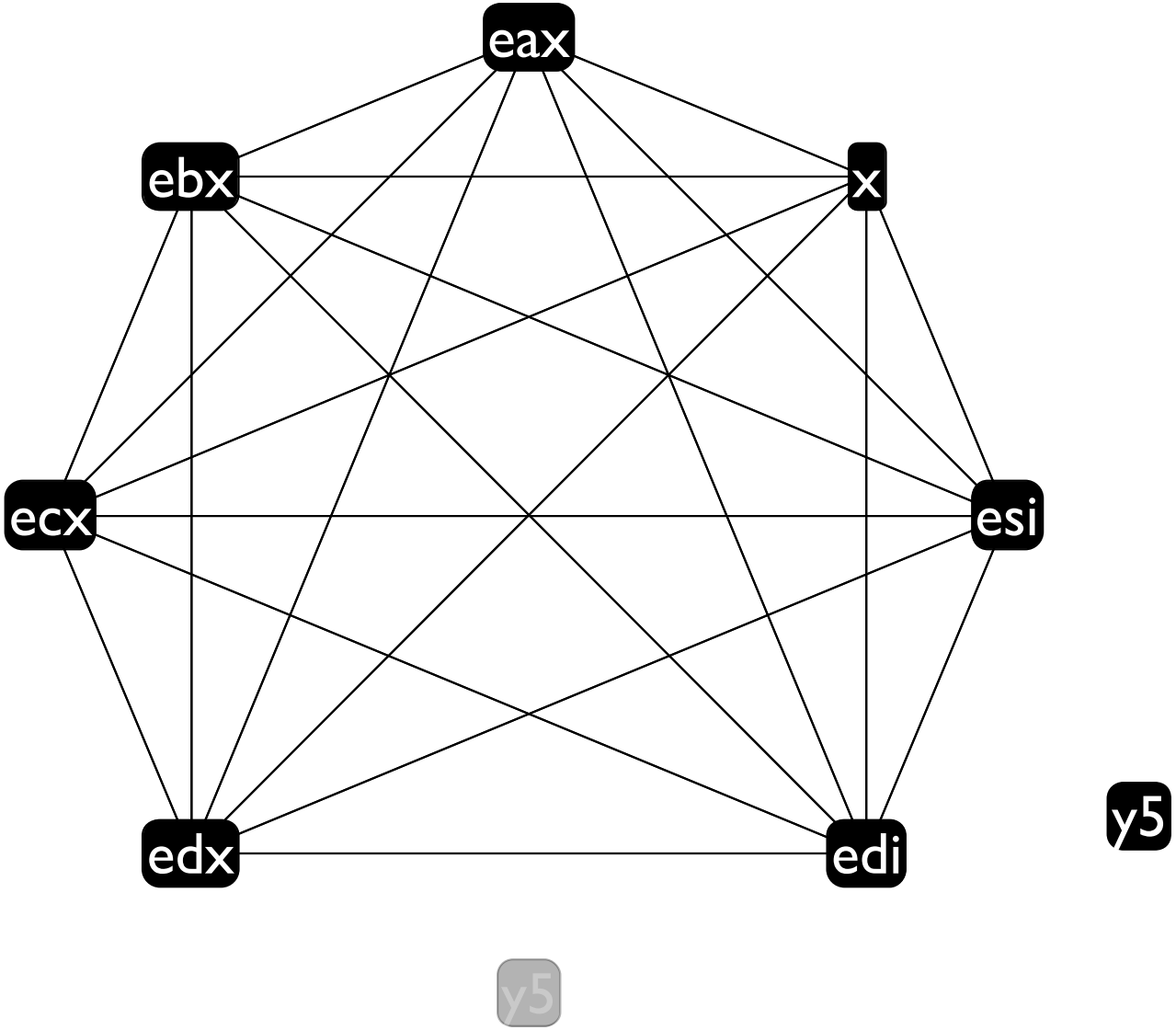
```
:f
(x <- eax)
(call :g)
((mem ebp -4) <- eax)
(eax += x)
(call :h)
(y5 <- (mem ebp -4))
(y5 *= 5)
(eax += y5)
(return)
```

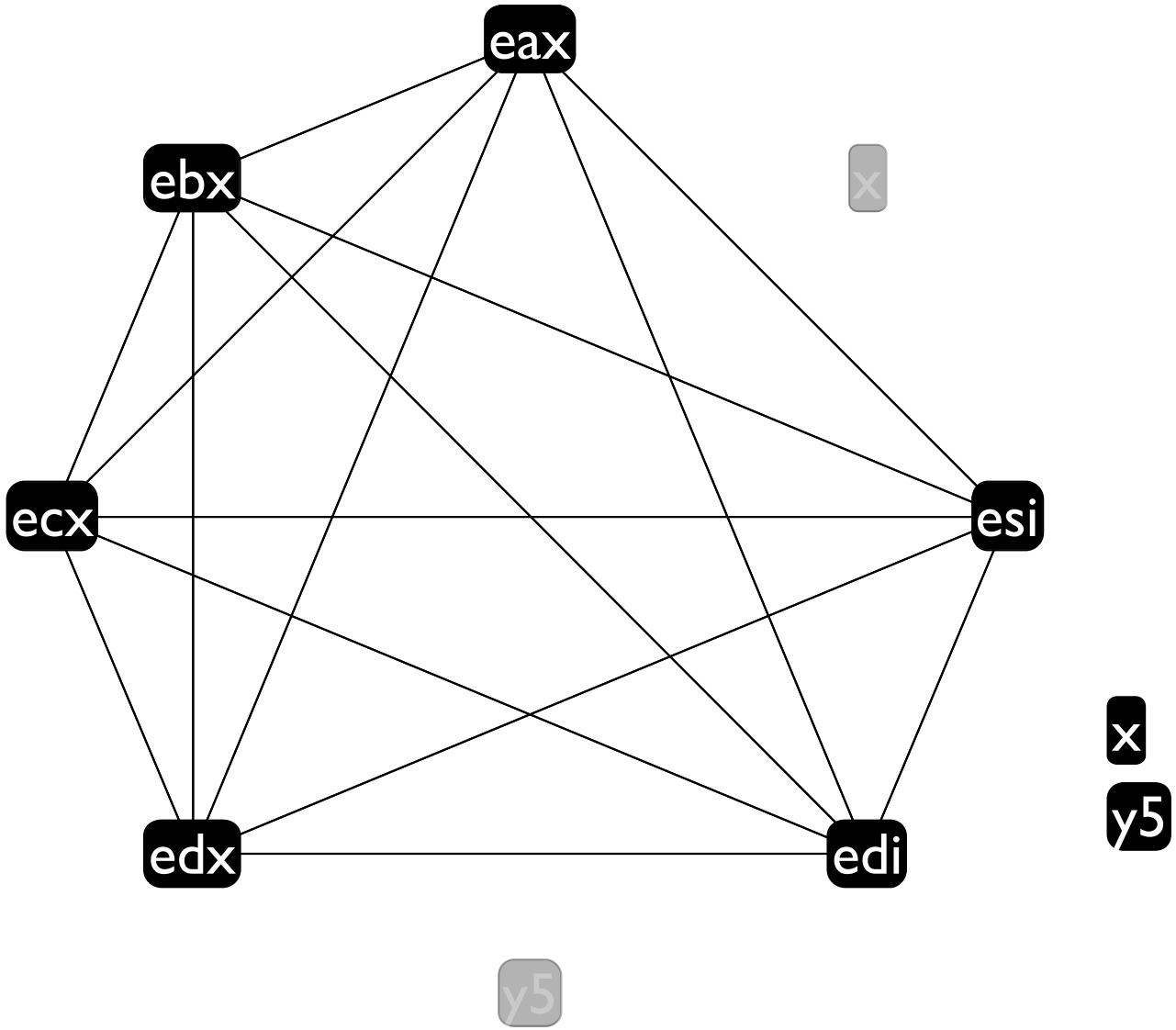
Spilling y

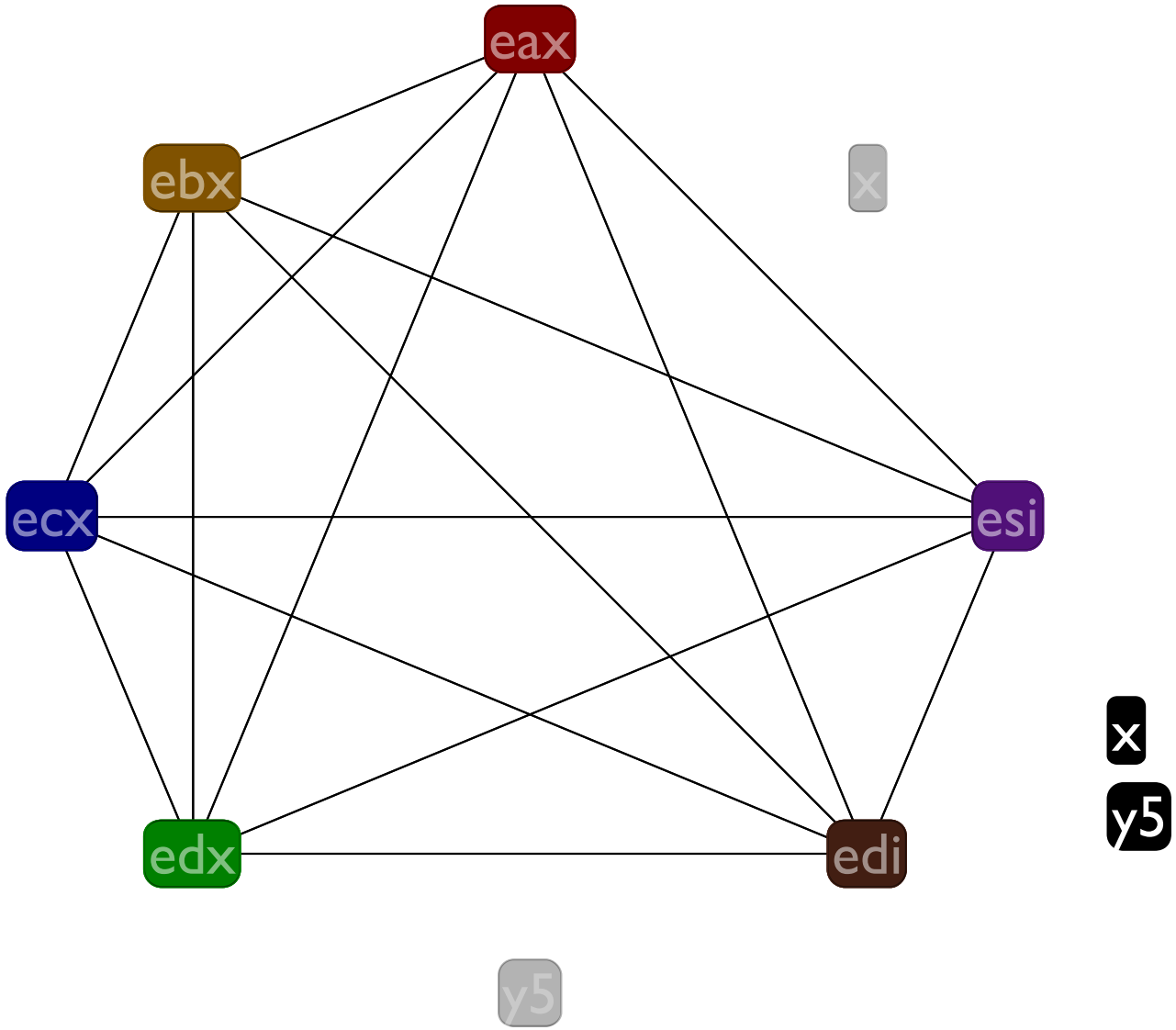
```
:f
(x <- eax)
(call :g)
((mem ebp -4) <- eax)
(eax += x)
(call :h)
(y5 <- (mem ebp -4))
(y5 *= 5)
(eax += y5)
(return)
```

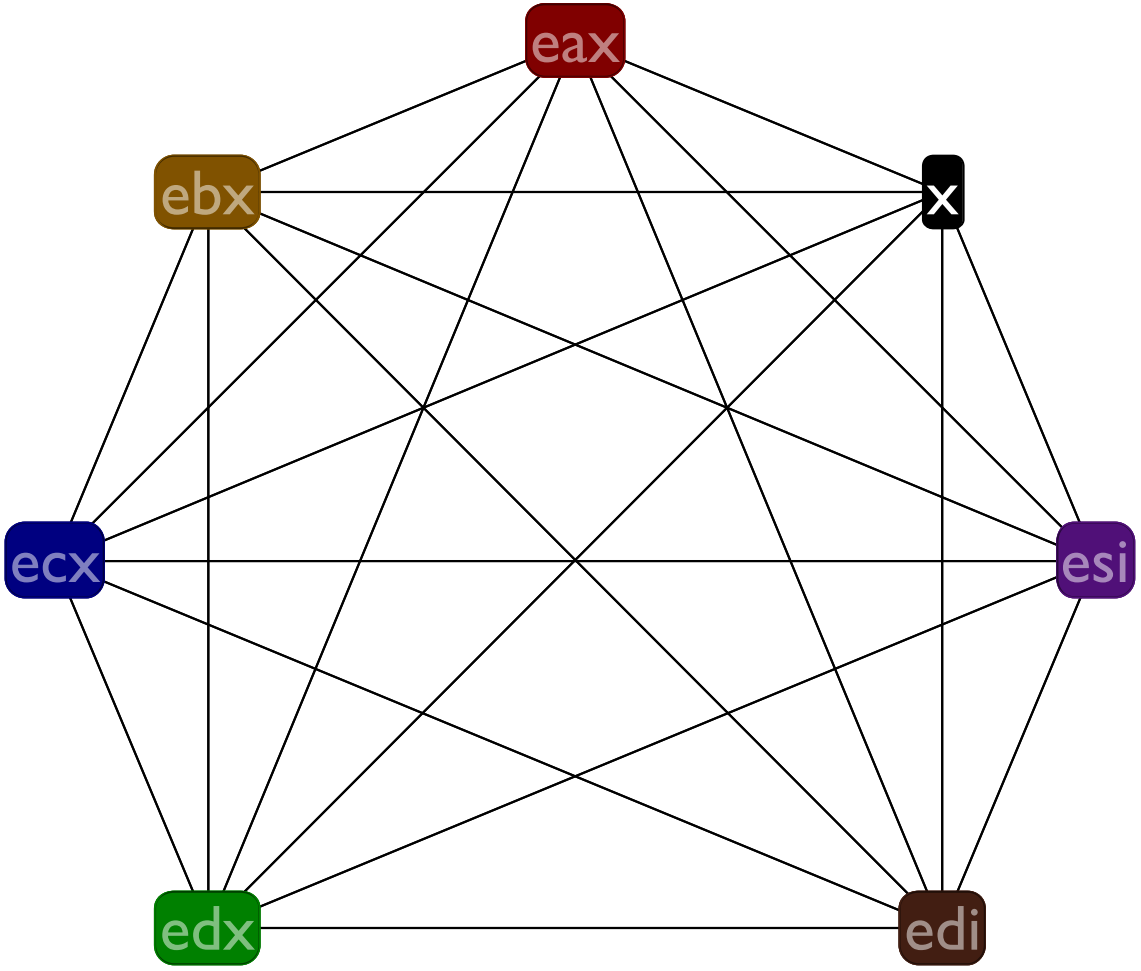


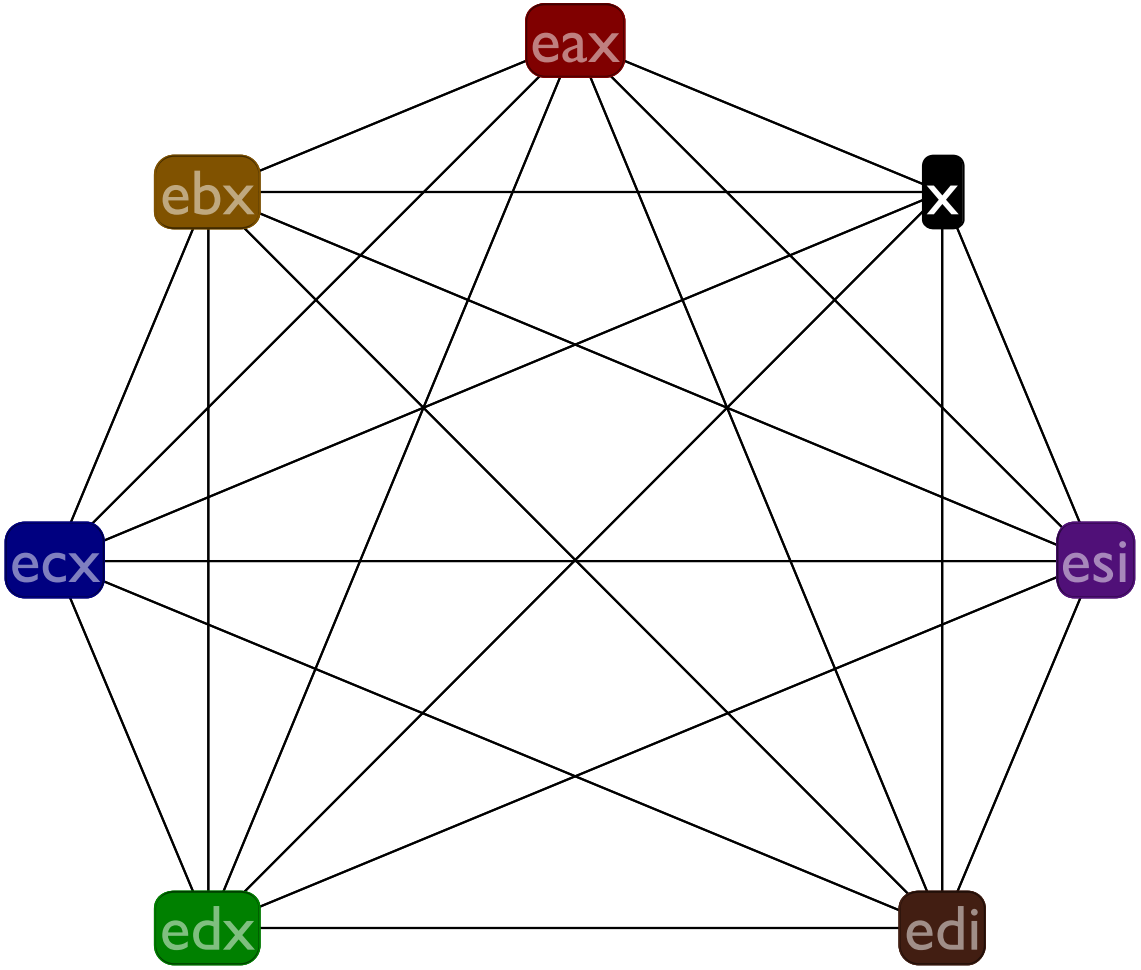

eax
ebx
x
ecx
esi
edx
edi
y5


eax
ebx
x
ecx
esi
edx
edi
y5
y5

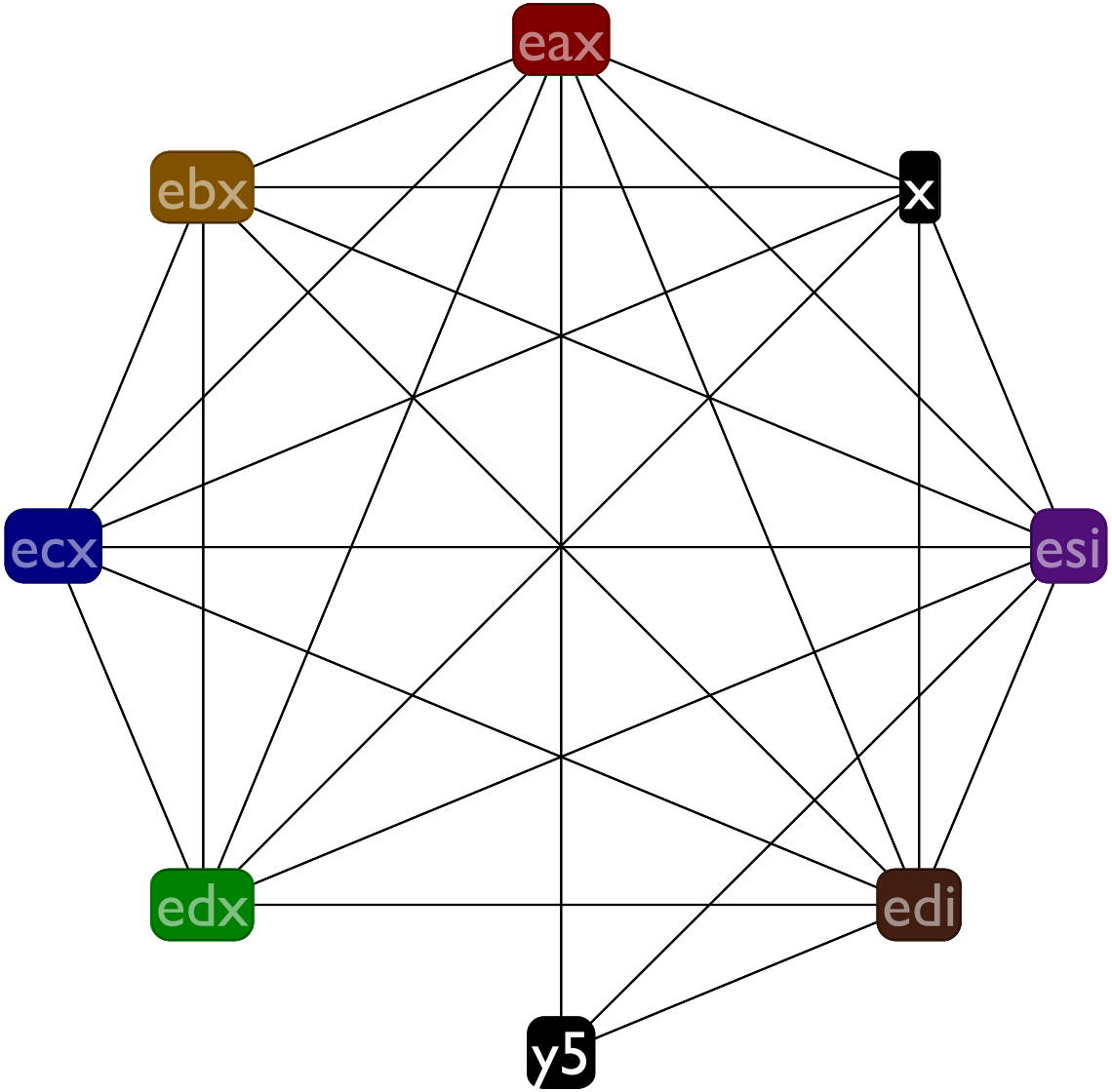

eax
ebx
x
ecx
esi
edx
edi
y5
x
y5

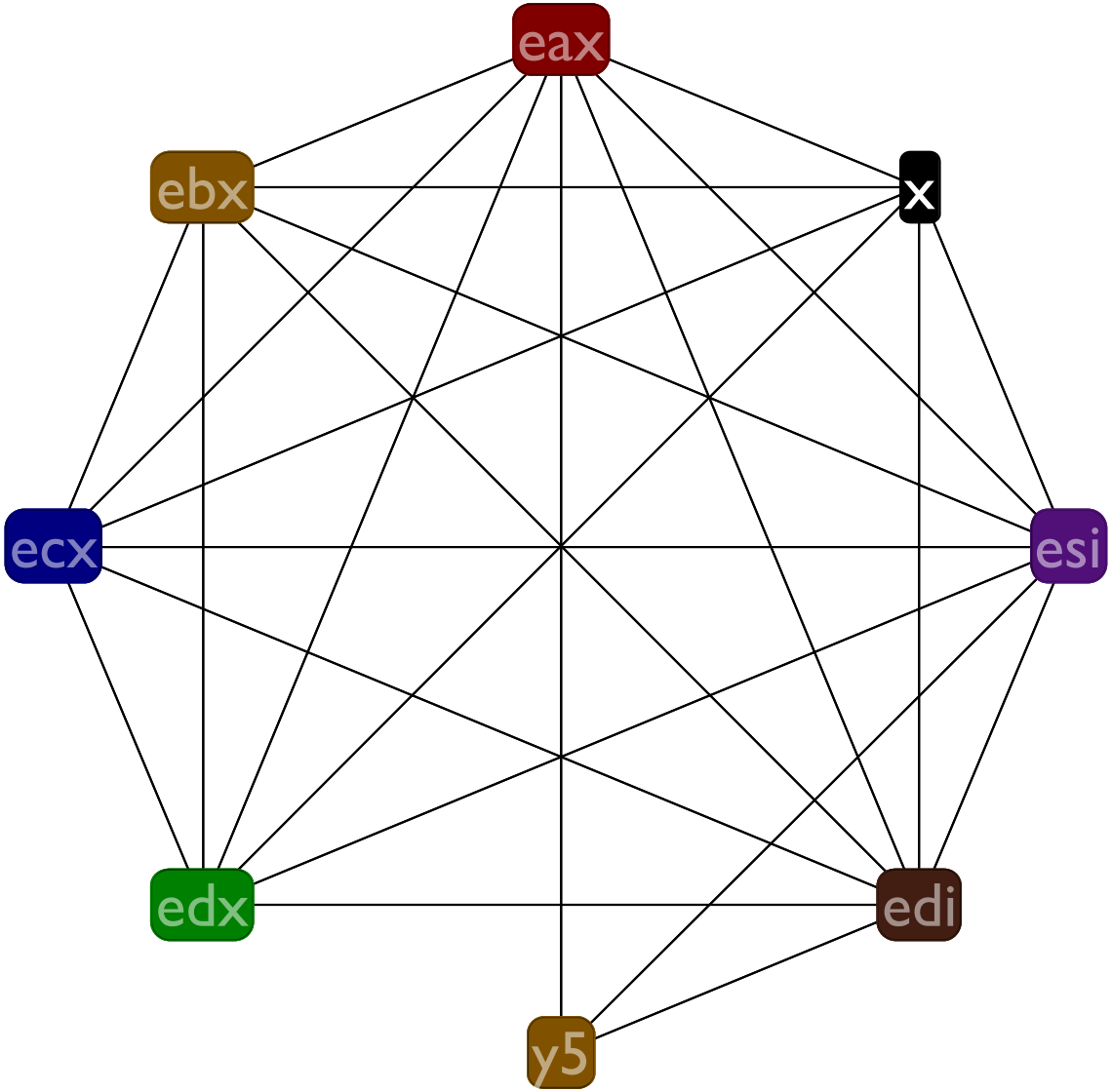

eax
ebx
x
ecx
esi
edx
edi
y5
x
y5


eax
ebx
x
ecx
esi
edx
edi
y5
y5



y5


eax
ebx
x
ecx
esi
edx
edi
y5


eax
ebx
x
ecx
esi
edx
edi
y5

Spilling x

Before:

```
:f
(x <- eax)
(call :g)
((mem ebp -4) <- eax)
(eax += x)
(call :h)
(y5 <- (mem ebp -4))
(y5 *= 5)
(eax += y5)
(return)
```

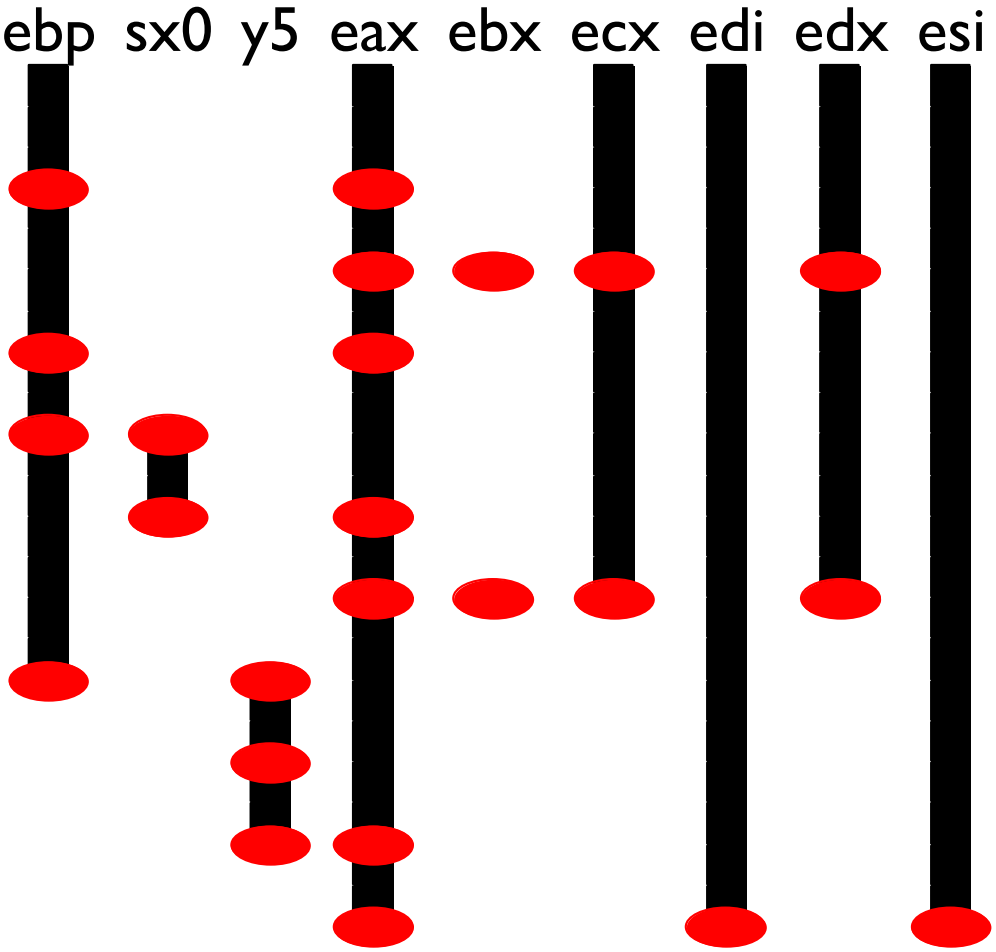
After:

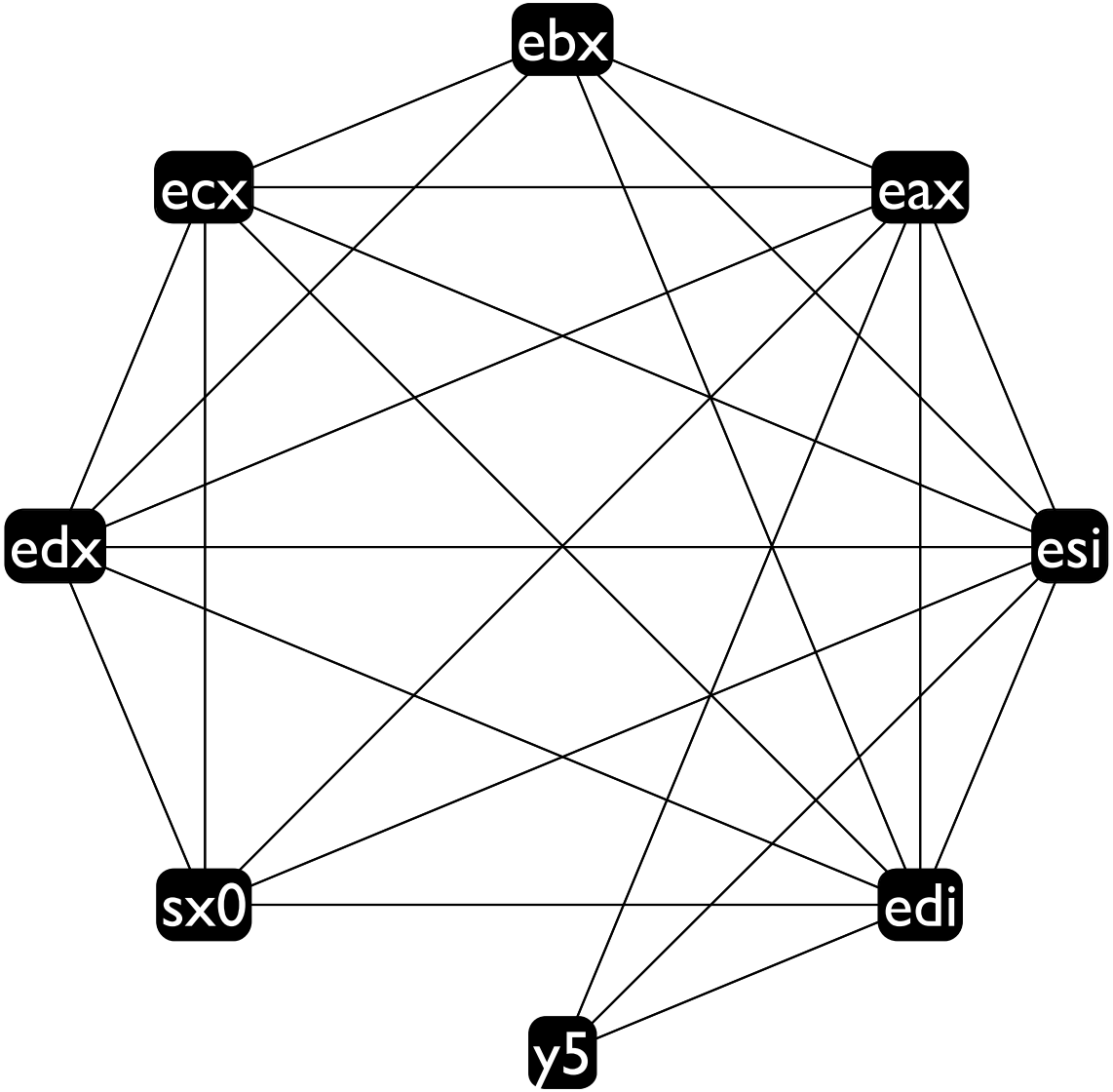
```
:f
((mem ebp -8) <- eax)
(call :g)
((mem ebp -4) <- eax)
(sx0 <- (mem ebp -8))
(eax += sx0)
(call :h)
(y5 <- (mem ebp -4))
(y5 *= 5)
(eax += y5)
(return)
```

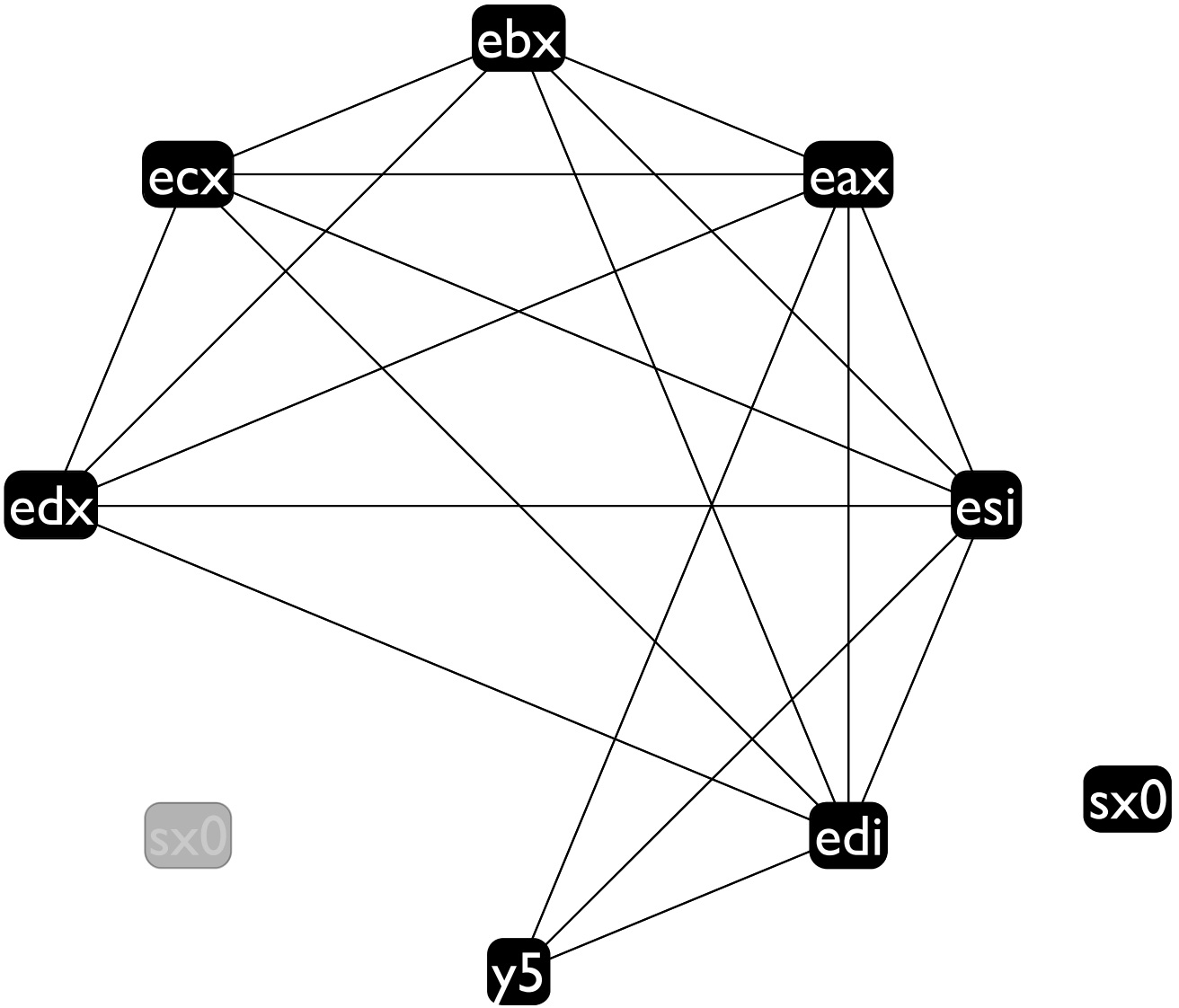
Note that this time we introduce a **sx0**, but compare its live range to **x**'s

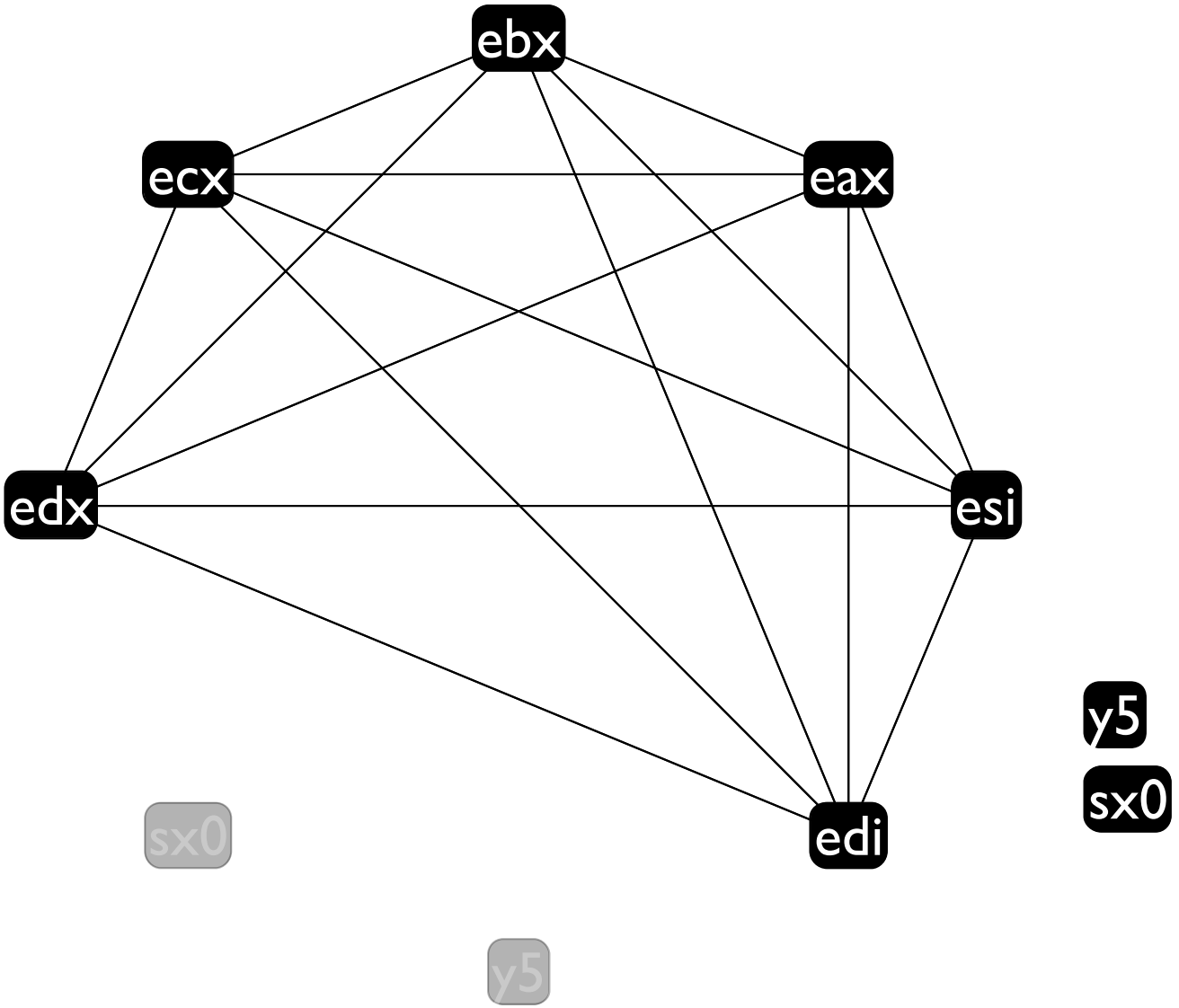
Spilling x

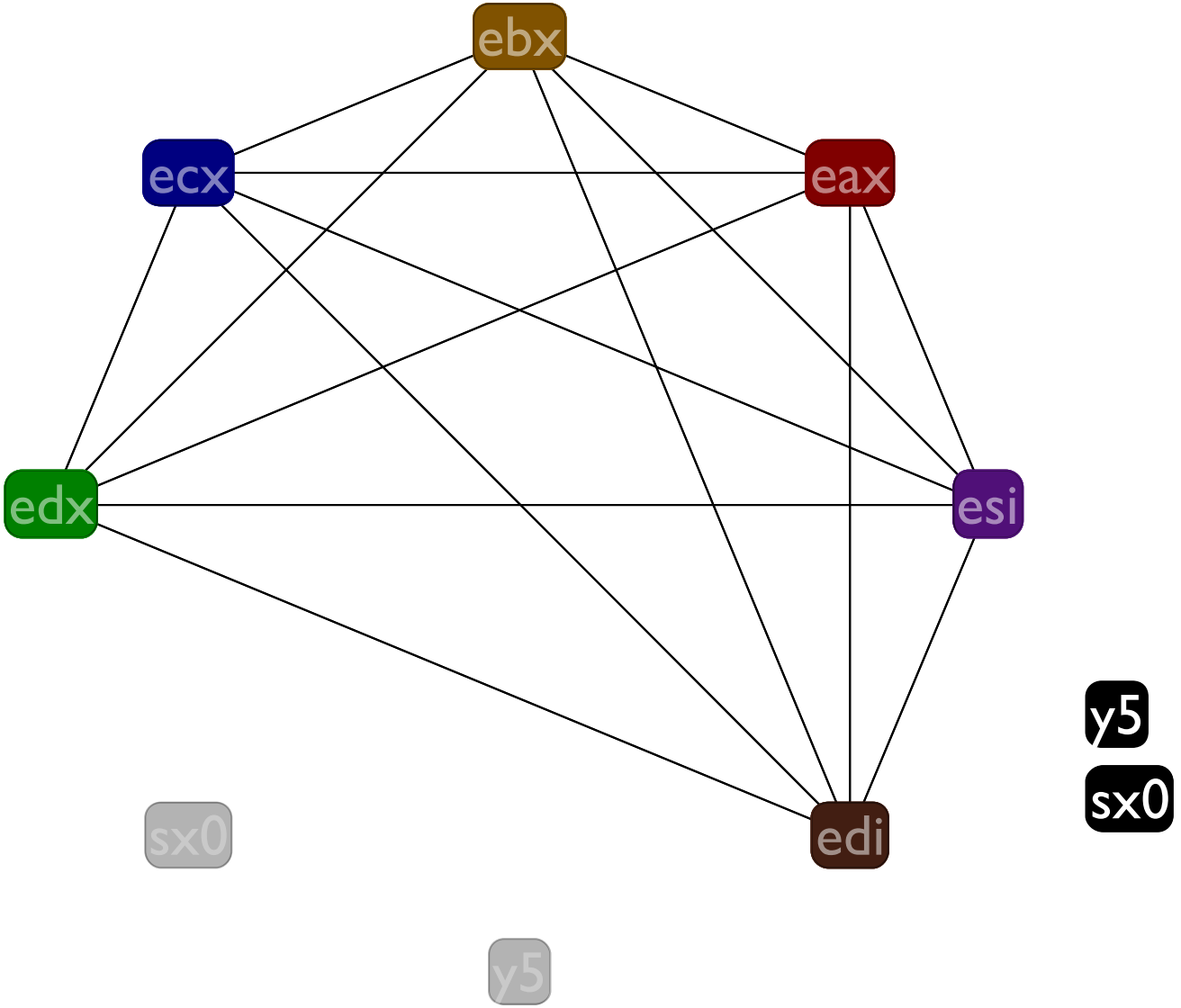
```
:f
((mem ebp -8) <- eax)
(call :g)
((mem ebp -4) <- eax)
(sx0 <- (mem ebp -8))
(eax += sx0)
(call :h)
(y5 <- (mem ebp -4))
(y5 *= 5)
(eax += y5)
(return)
```

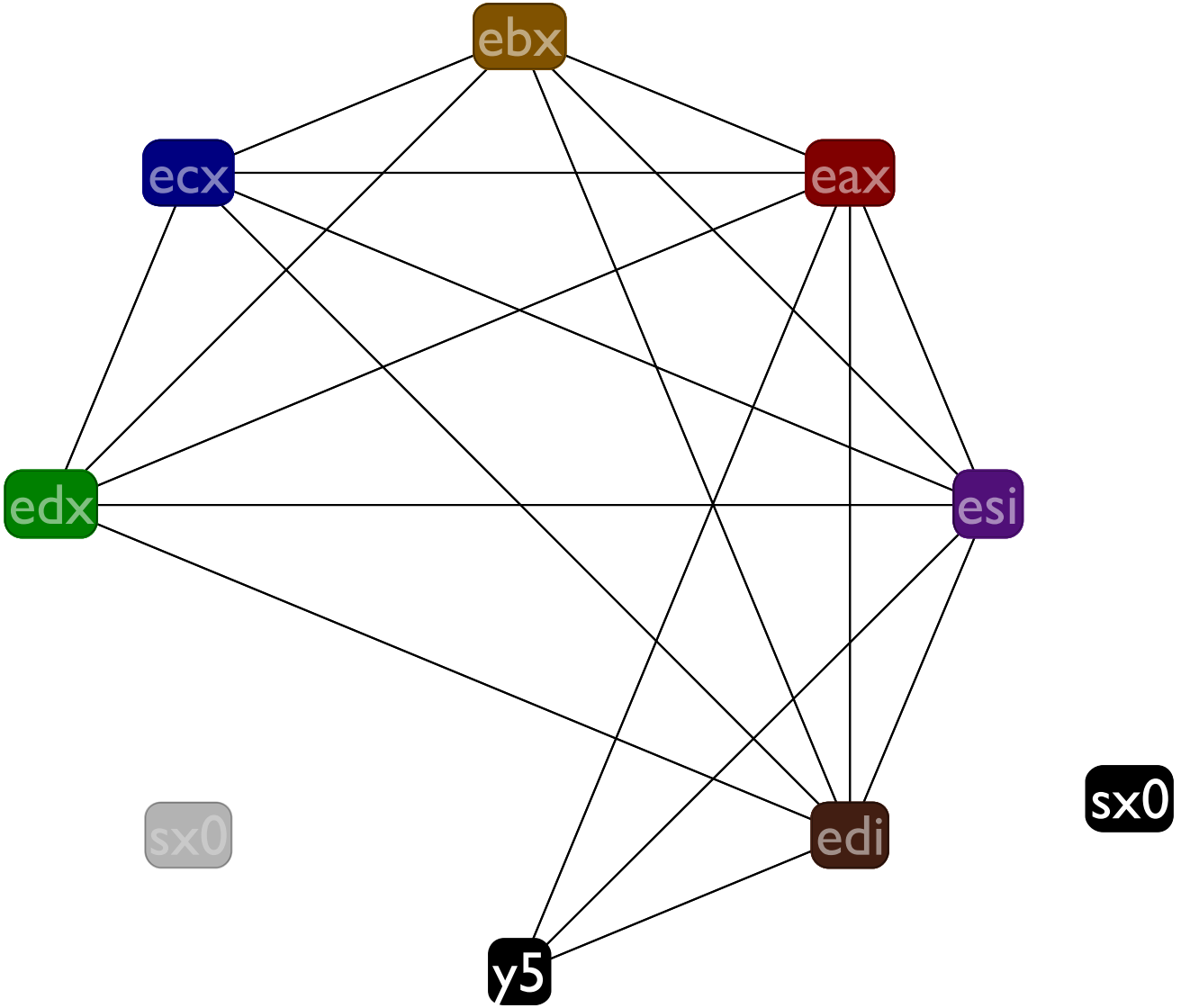


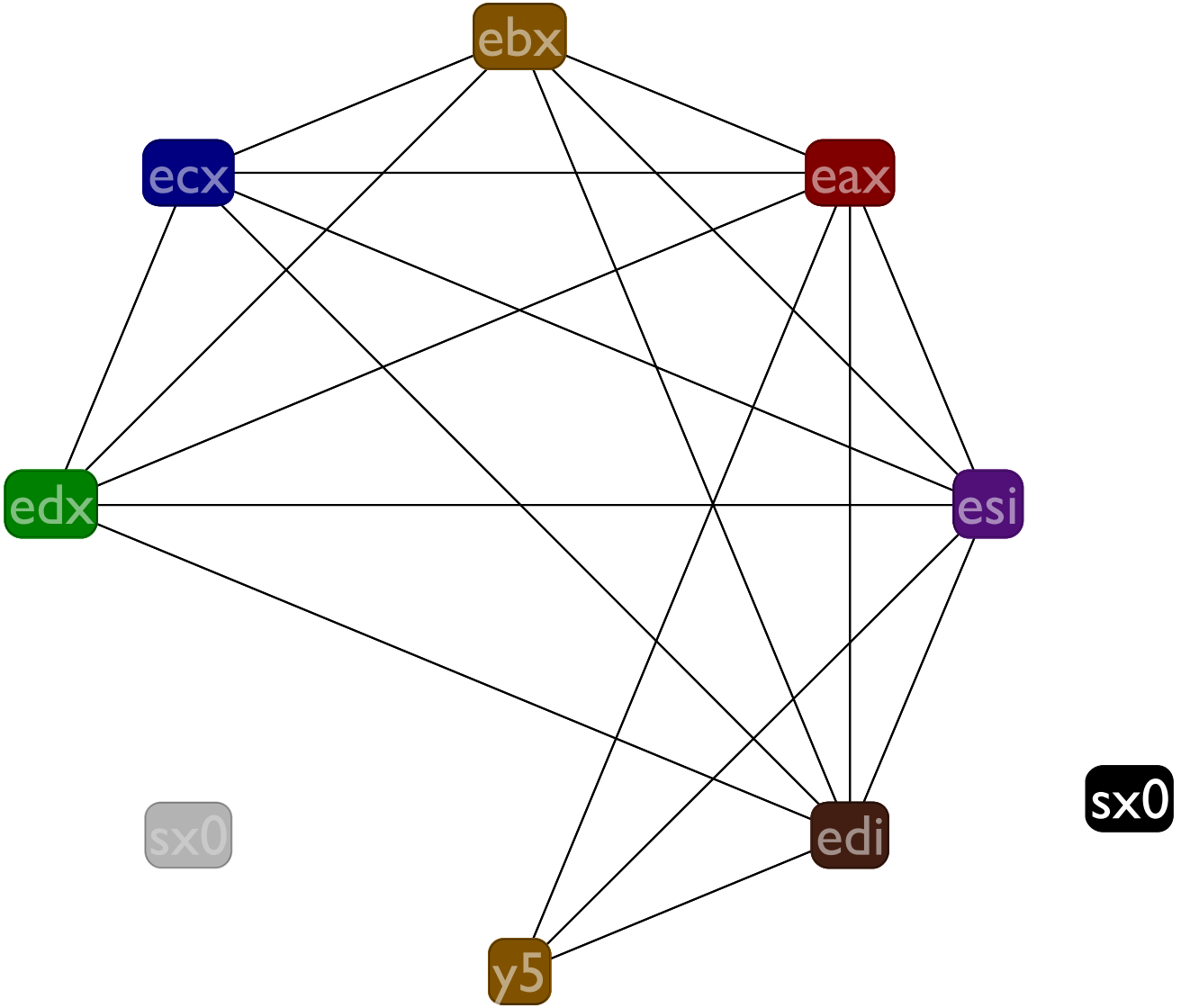

ebx
ecx
eax
edx
esi
sx0
edi
y5

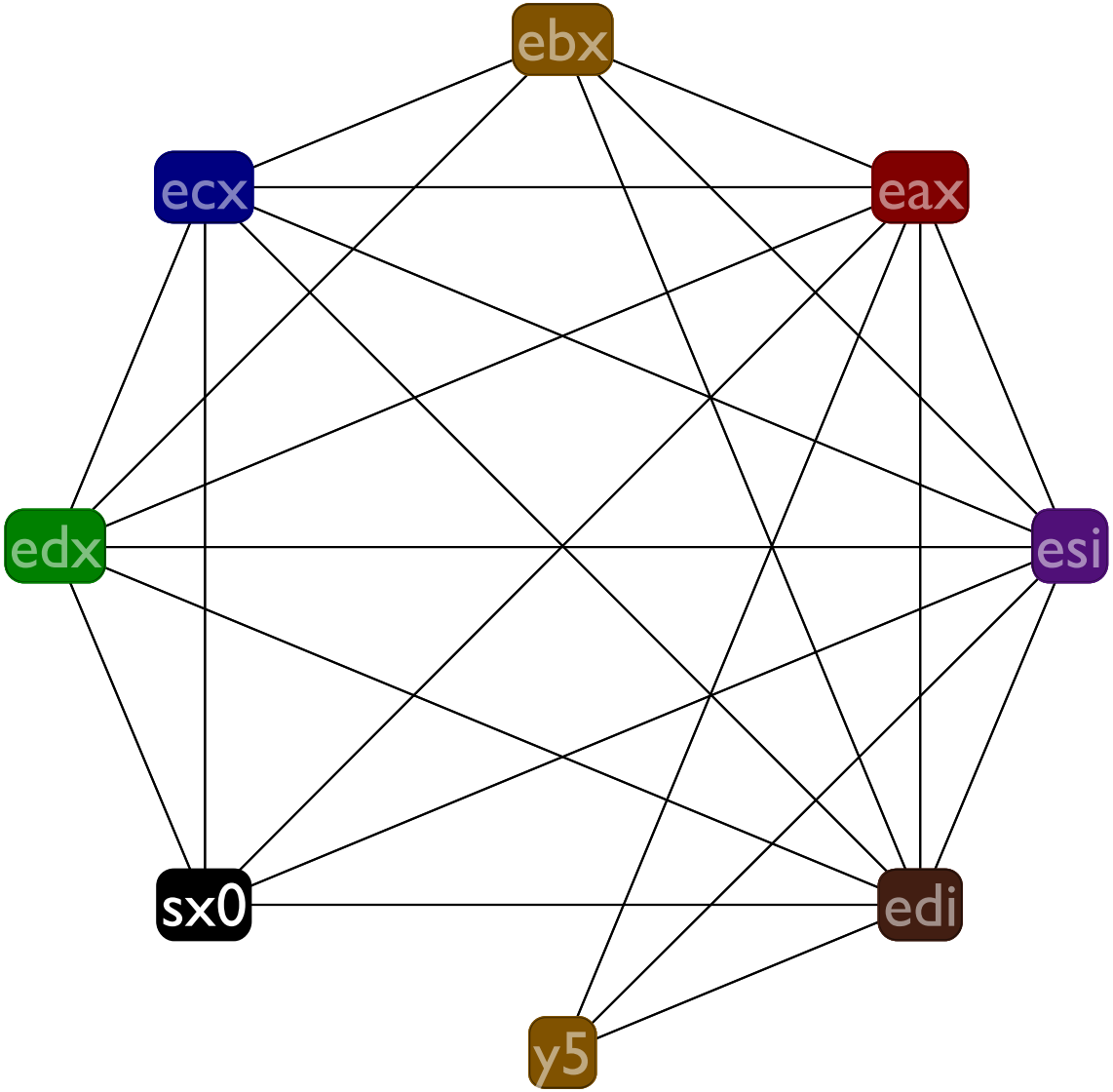

ebx
ecx
eax
edx
esi
sx0
edi
sx0
y5

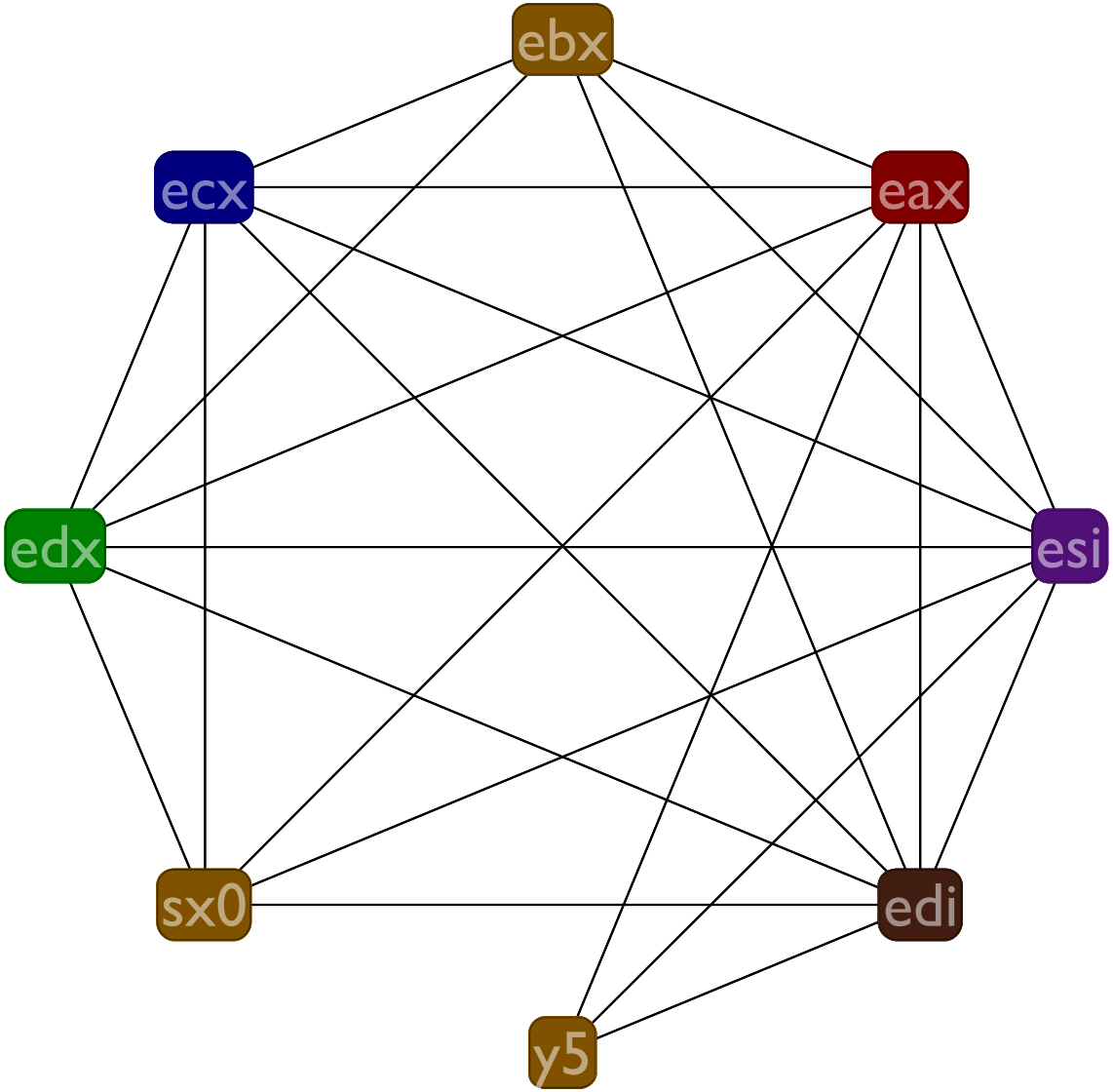

ebx
ecx
eax
edx
esi
edi
sx0
y5
sx0
y5


ebx
ecx
eax
edx
esi
edi
sx0
y5

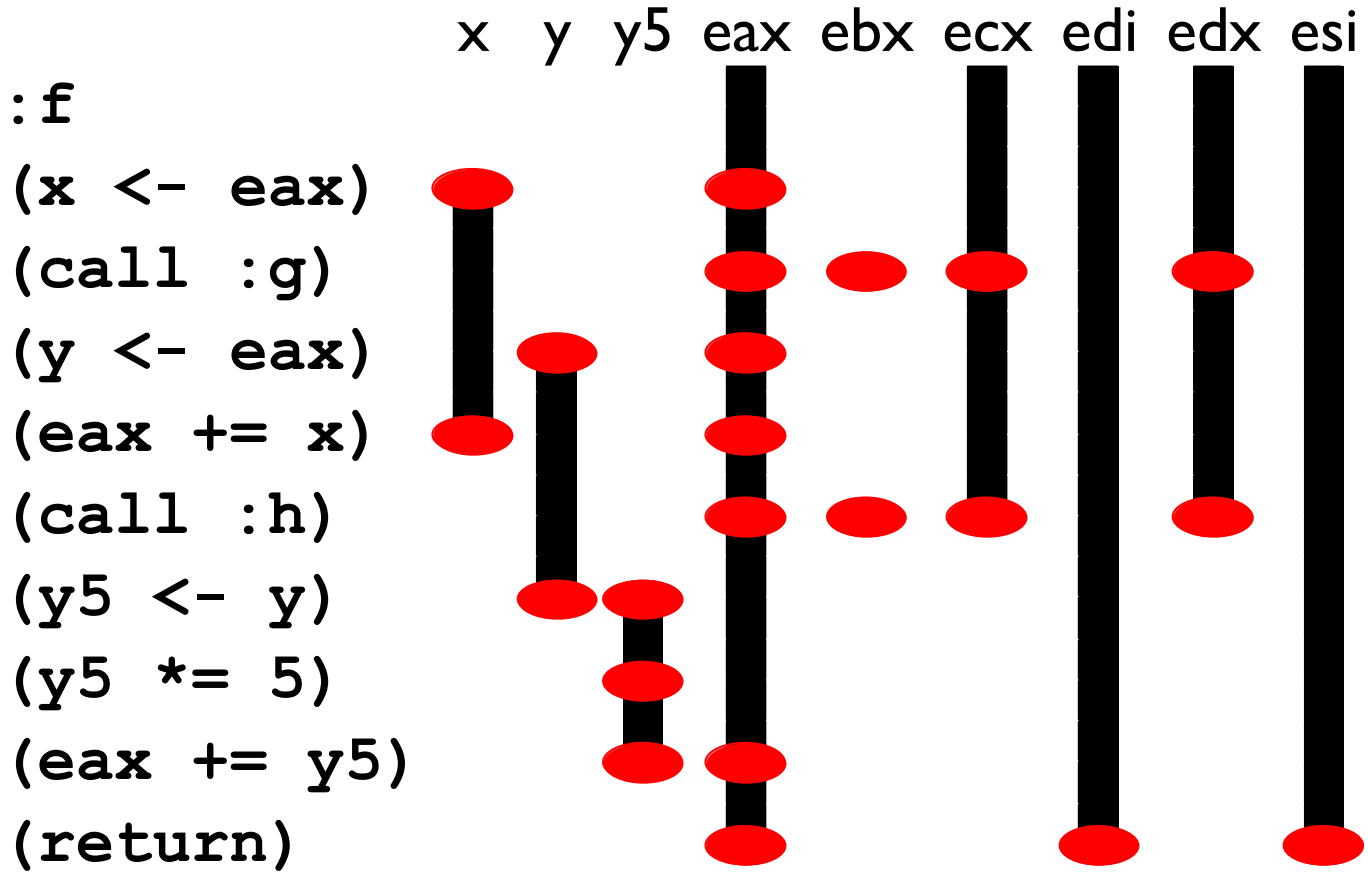

ebx
ecx
eax
edx
esi
sx0
edi
y5
sx0


ebx
ecx
eax
edx
esi
sx0
edi
y5
sx0


ebx
ecx
eax
edx
esi
sx0
edi
y5


ebx
ecx
eax
edx
esi
sx0
edi
y5

Live ranges


x y y5 eax ebx ecx edi edx esi
:f
(x <- eax)
(call :g)
(y <- eax)
(eax += x)
(call :h)
(y5 <- y)
(y5 *= 5)
(eax += y5)
(return)

Live ranges

We spilled two variables that have relatively short live ranges, but look at those long live ranges with no uses of the variables that the callee save registers, i.e. **`edi`** and **`esi`**, have. We'd rather spill them.

Spilling callee saves

Unfortunately, it gets complicated to spill real registers. Instead, a trick: we just make up new variables to hold their values. Semantics of the program does not change, but:

- Now the real registers now have short live ranges, and
- New temporaries are spillable

Adding new variables g-ans

Before:

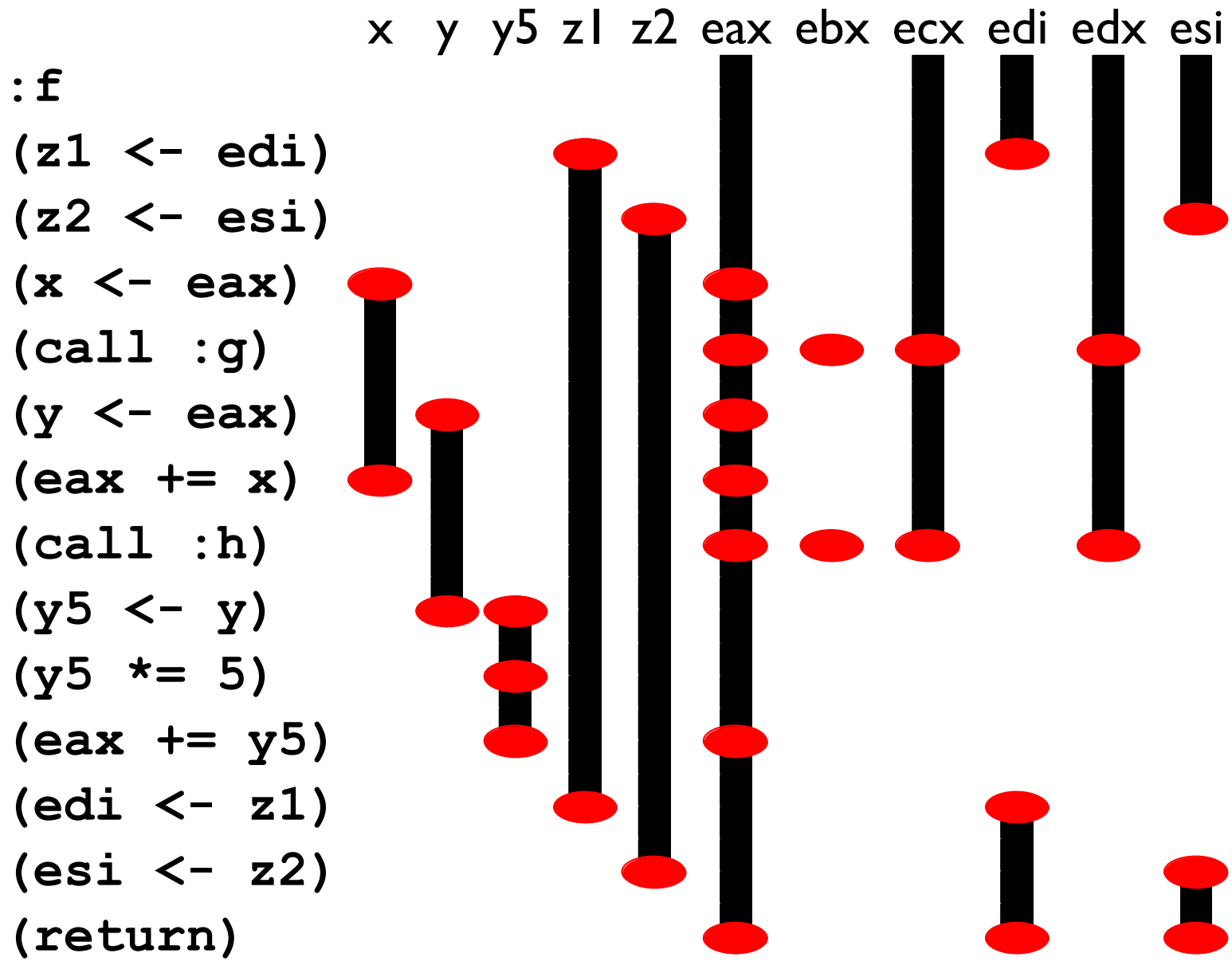
```
:f
(x <- eax)
(call :g)
(y <- eax)
(eax += x)
(call :h)
(y5 <- y)
(y5 *= 5)
(eax += y5)
(return)
```

After:

```
:f
(z1 <- edi)
(z2 <- esi)
(x <- eax)
(call :g)
(y <- eax)
(eax += x)
(call :h)
(y5 <- y)
(y5 *= 5)
(eax += y5)
(edi <- z1)
(esi <- z2)
(return)
```

Init new variables at beginning of fun, restore them before returning or making a tail call.

Live ranges with new variables



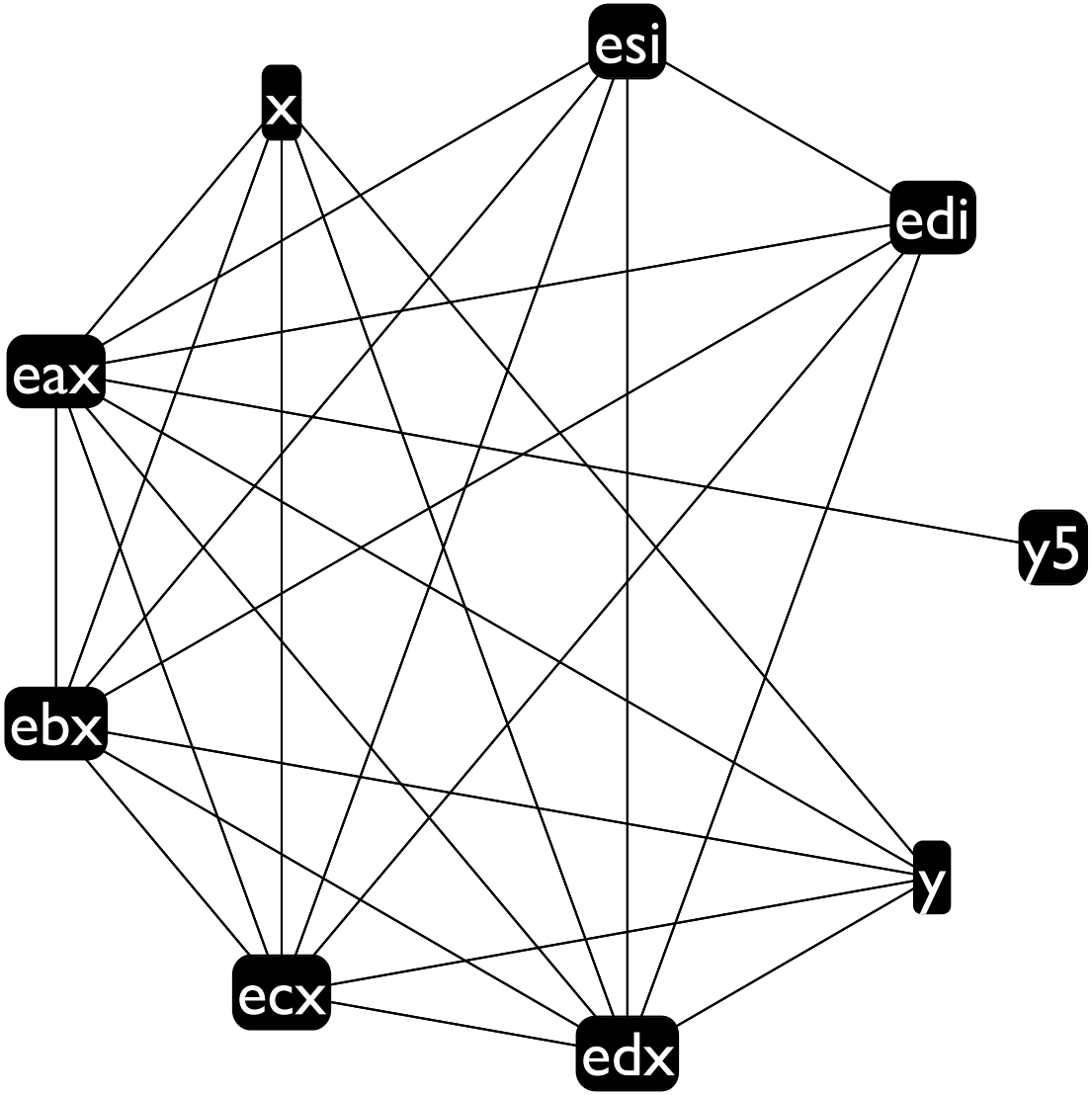
Spilling z1 & z2

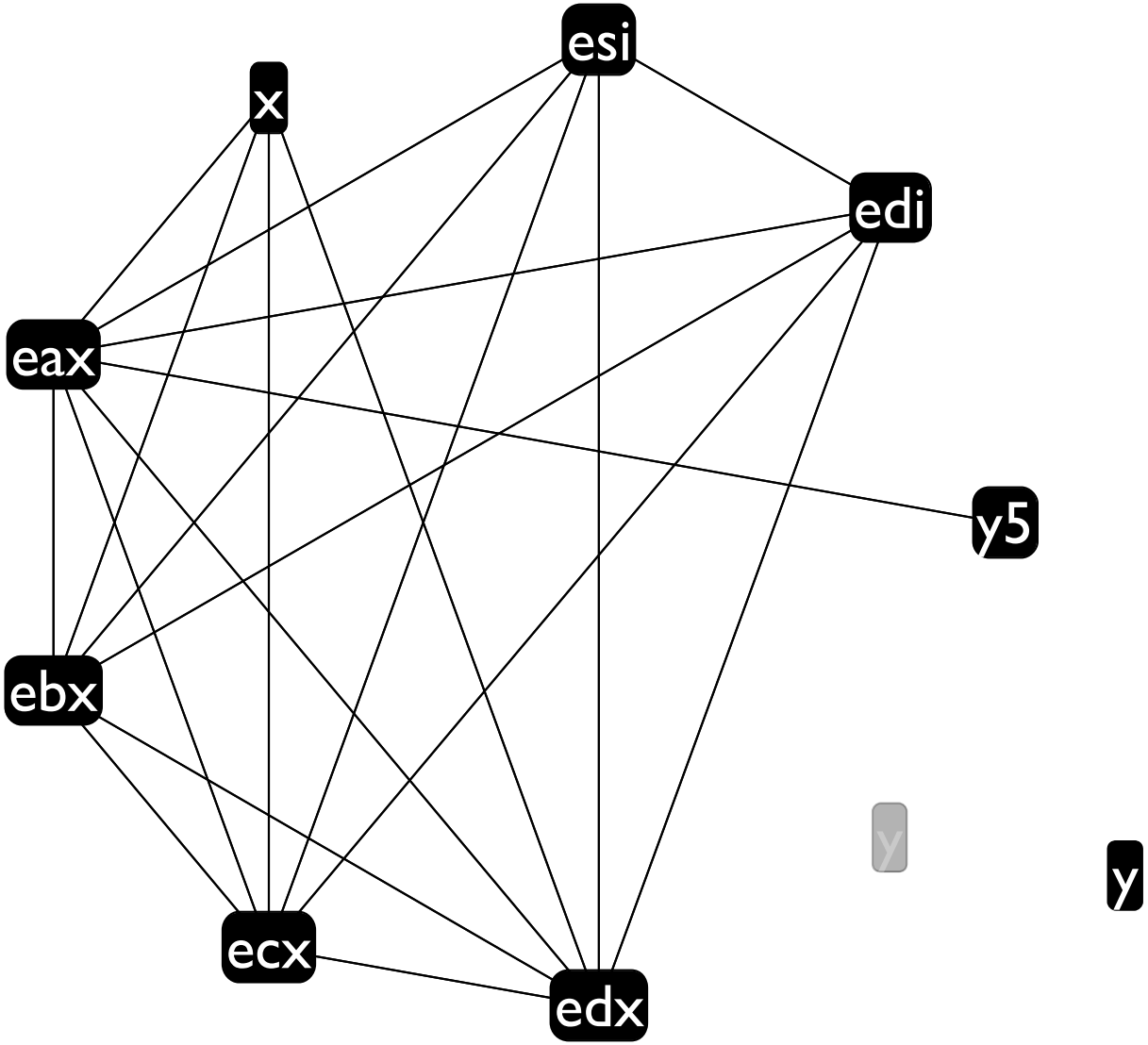
Before:

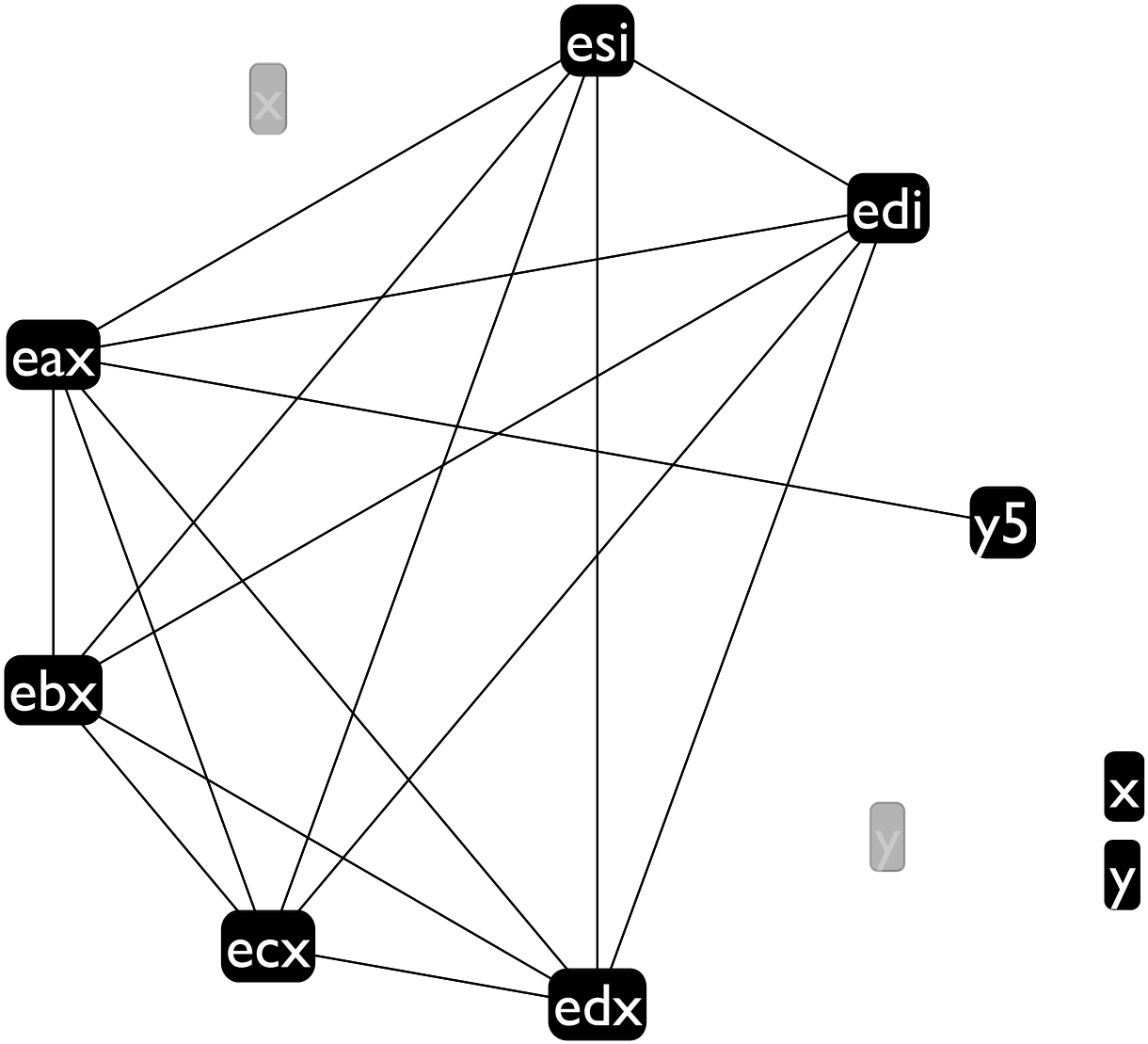
```
:f
(z1 <- edi)
(z2 <- esi)
(x <- eax)
(call :g)
(y <- eax)
(eax += x)
(call :h)
(y5 <- y)
(y5 *= 5)
(eax += y5)
(edi <- z1)
(esi <- z2)
(return)
```

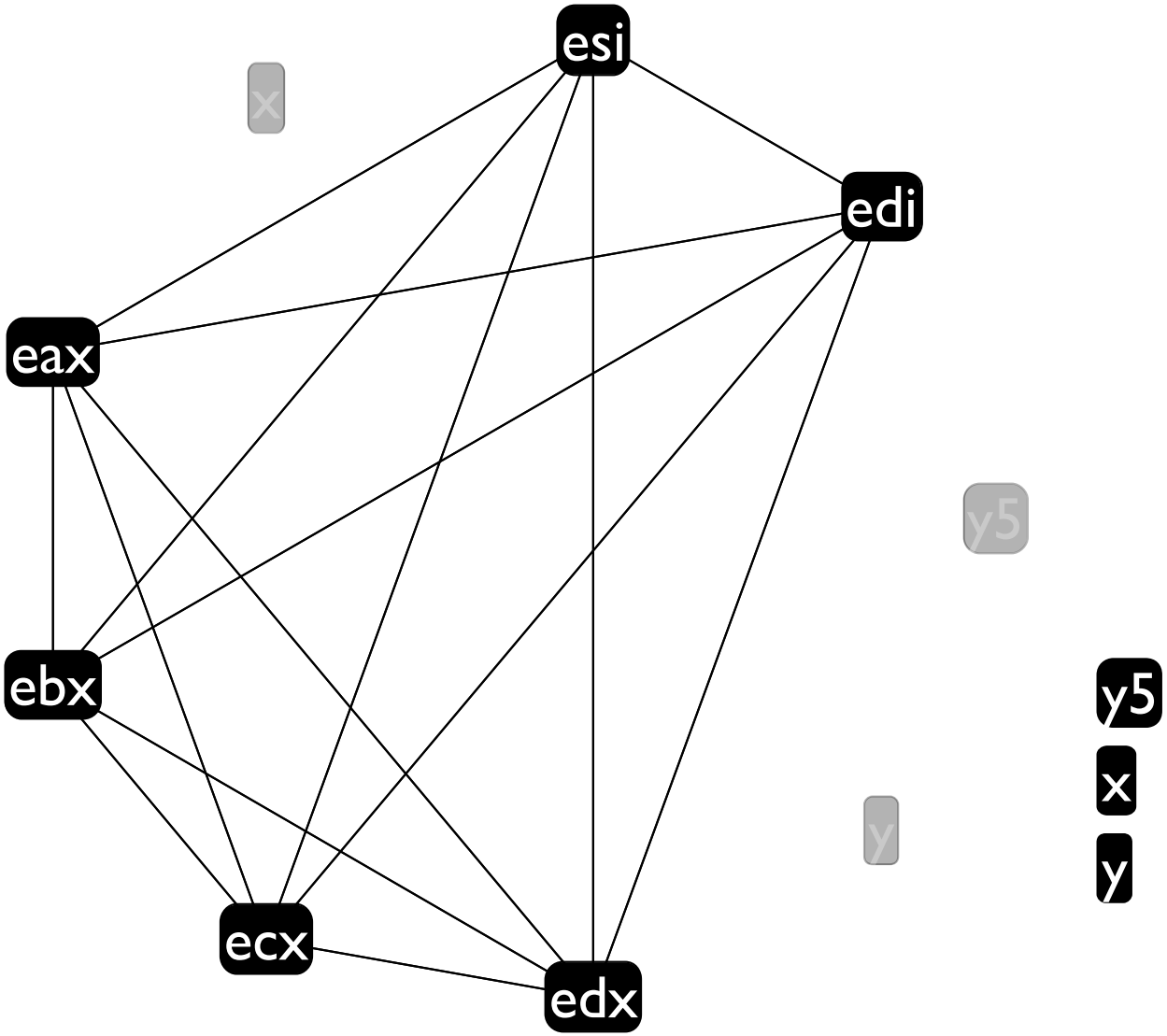
After:

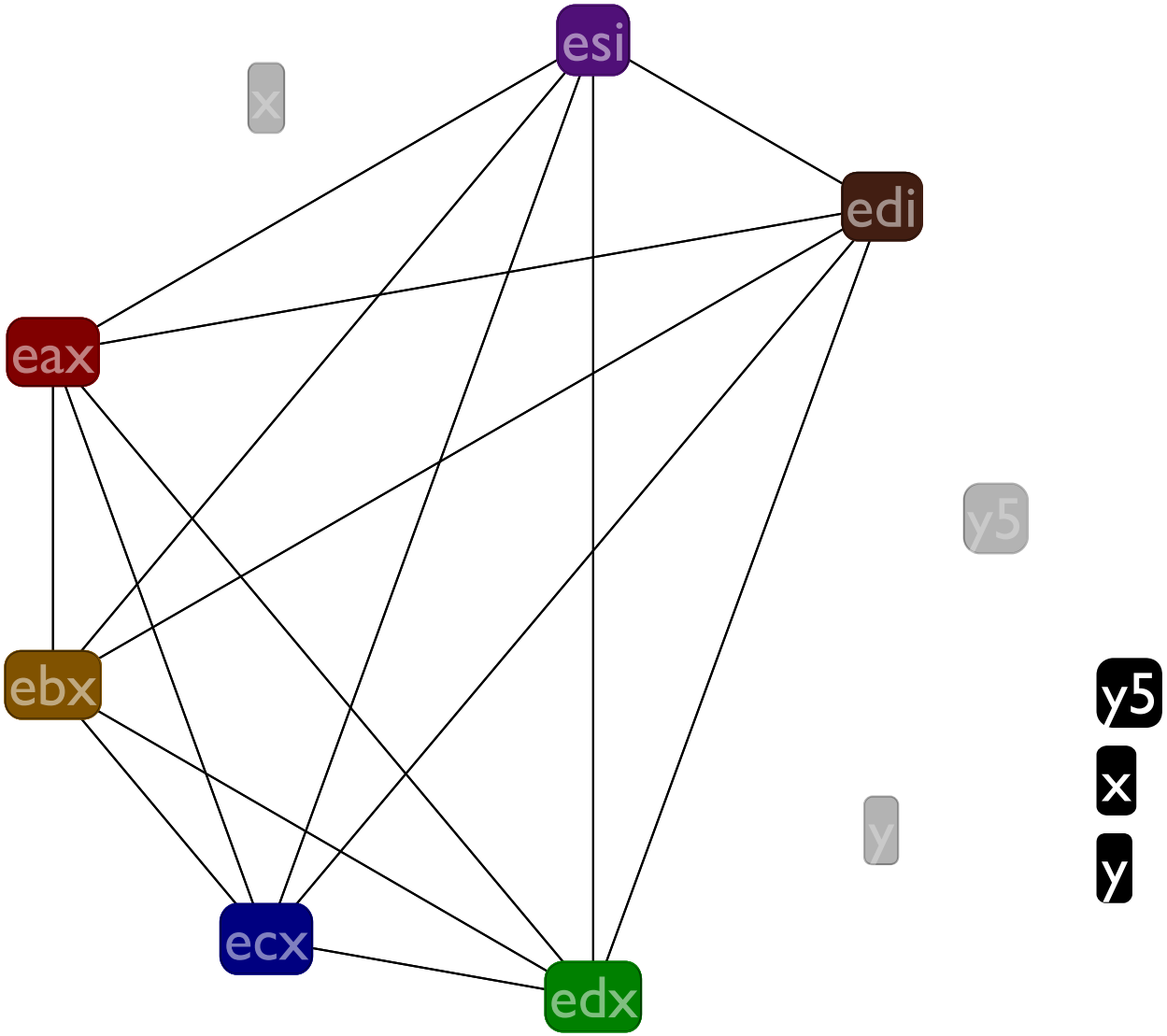
```
:f
((mem ebp -4) <- edi)
((mem ebp -8) <- esi)
(x <- eax)
(call :g)
(y <- eax)
(eax += x)
(call :h)
(y5 <- y)
(y5 *= 5)
(eax += y5)
(edi <- (mem ebp -4))
(esi <- (mem ebp -8))
(return)
```

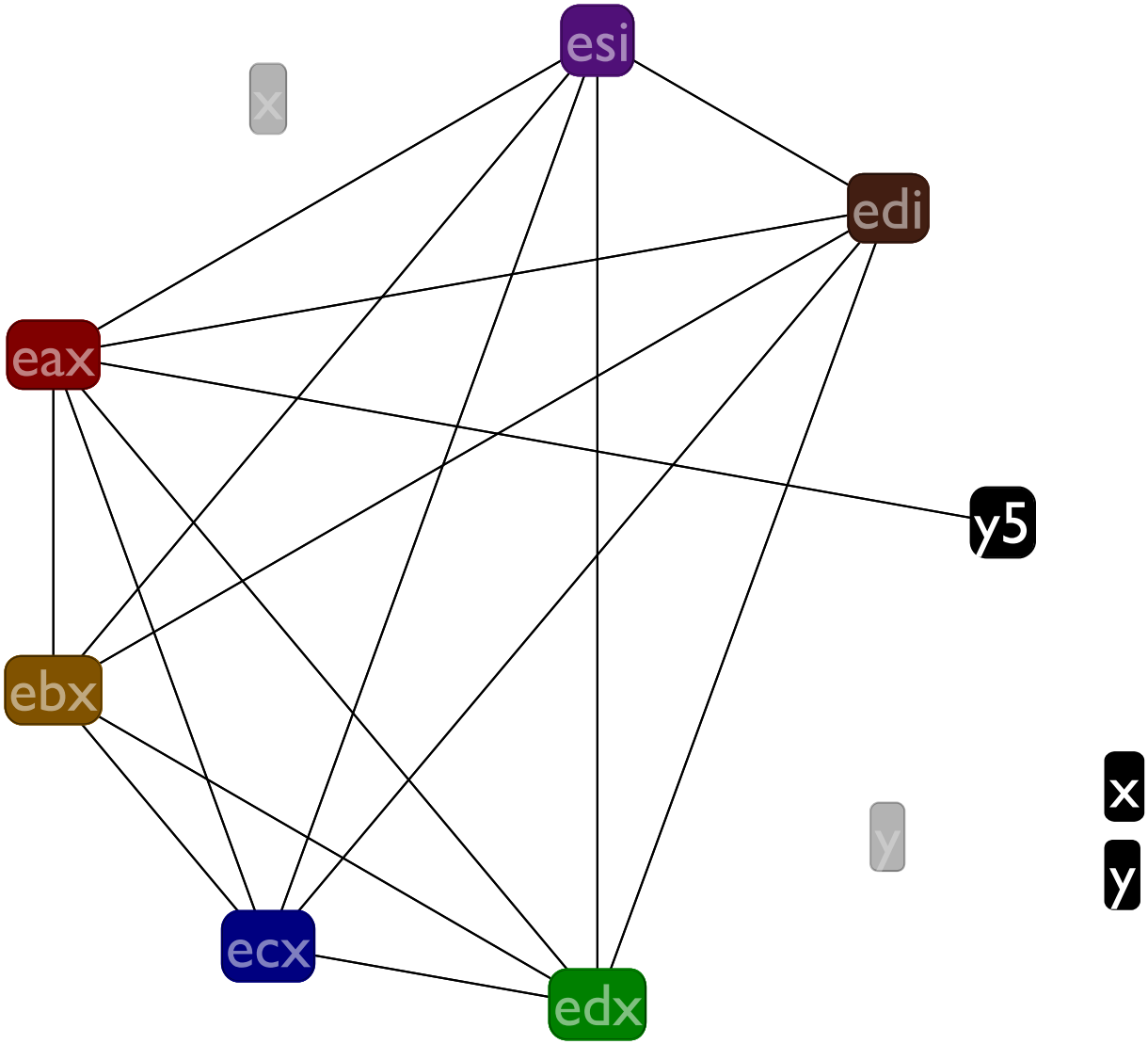

esi
x
edi
eax
y5
ebx
y
ecx
edx

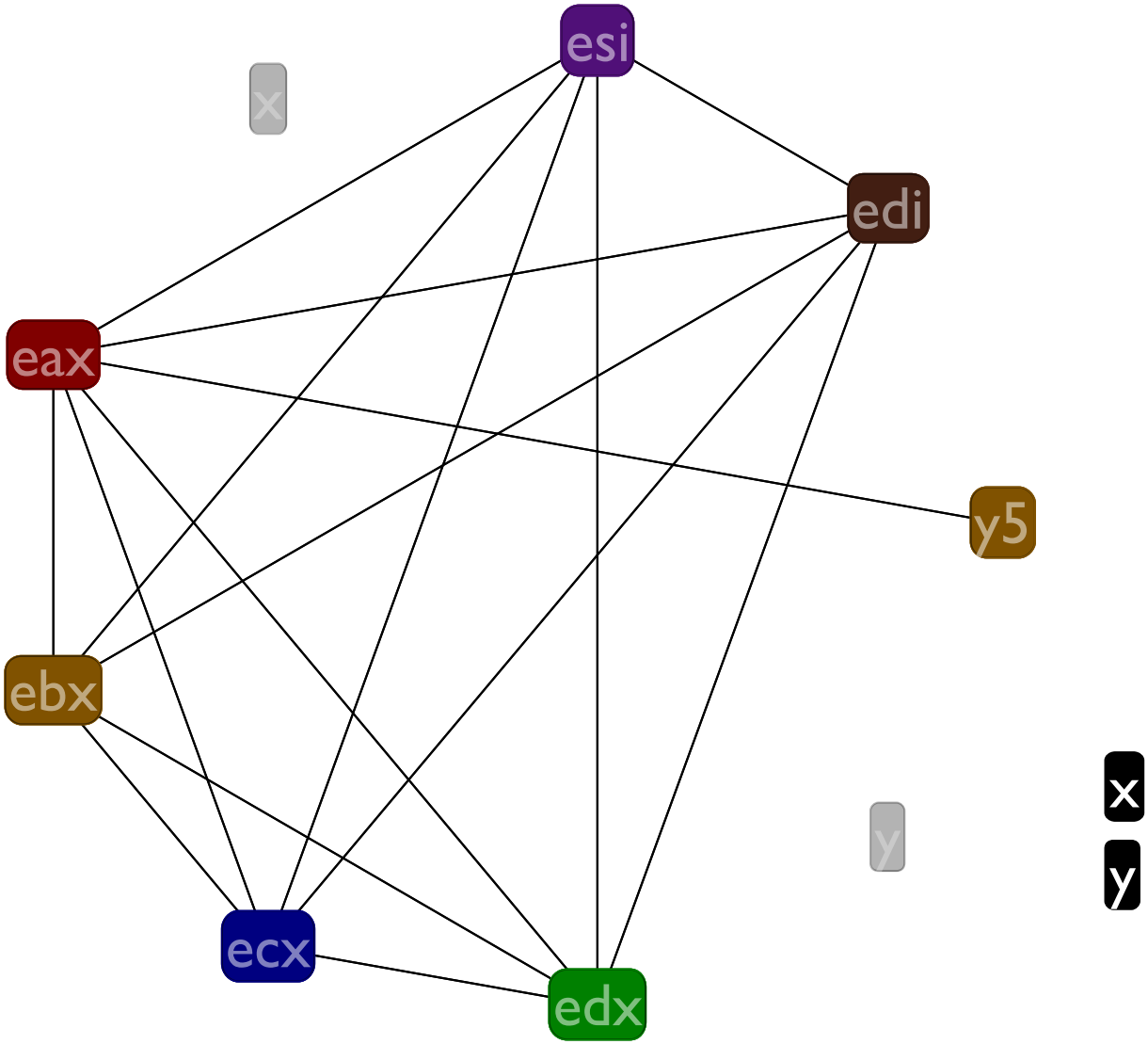

esi
x
edi
eax
y5
ebx
y
y
ecx
edx

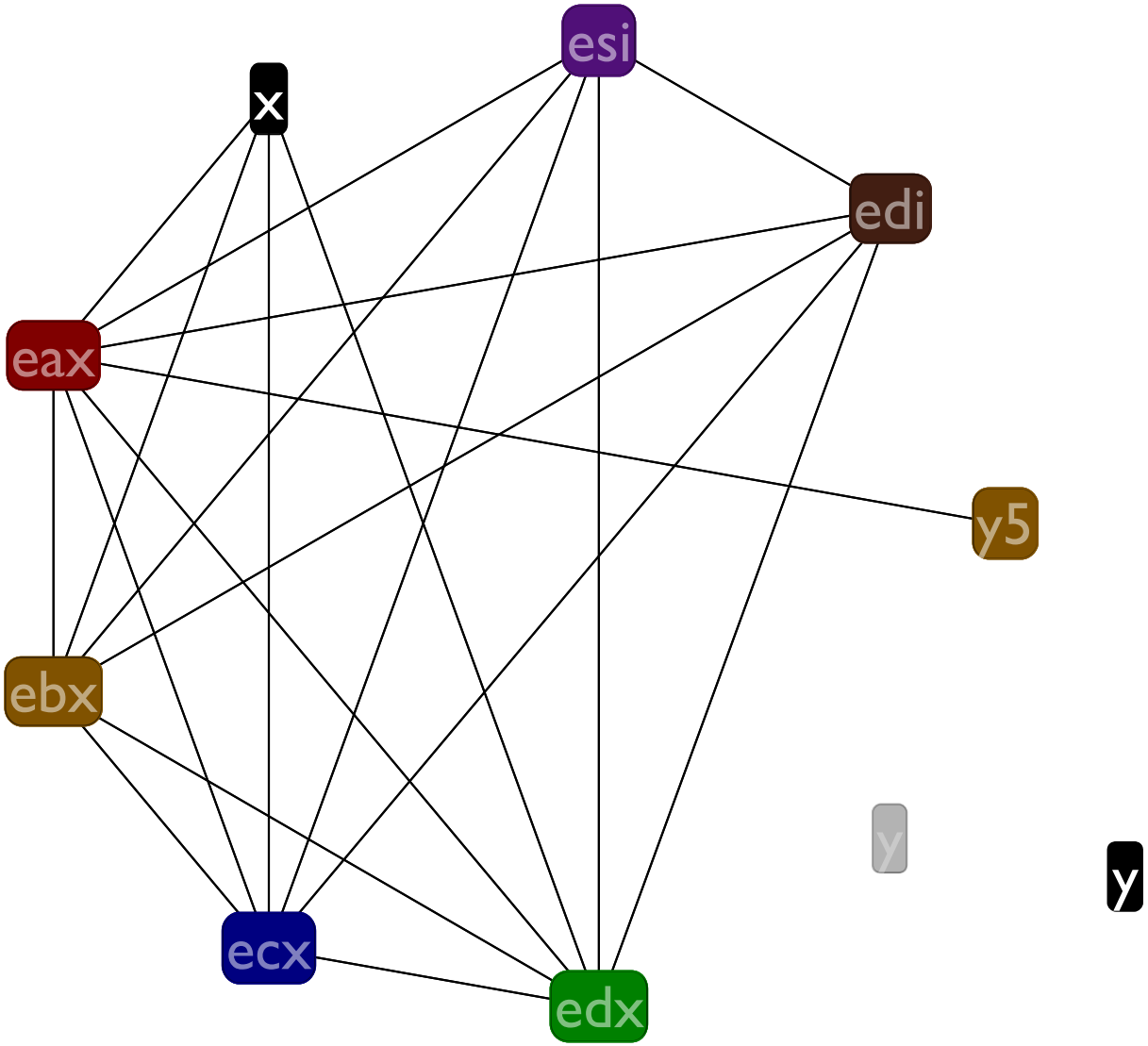

esi
x
edi
eax
y5
ebx
x
y
y
ecx
edx

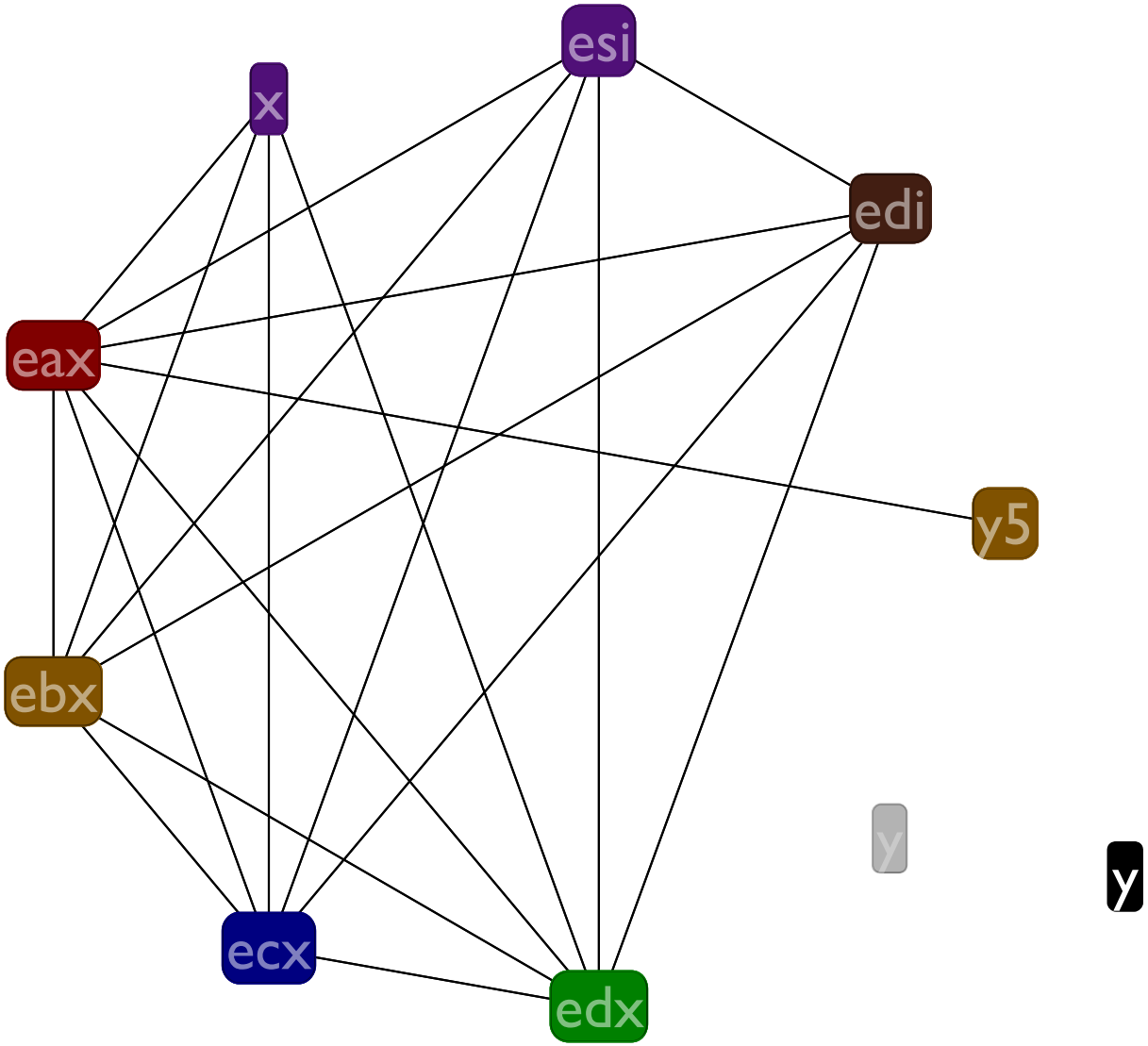

esi
x
edi
eax
y5
ebx
y5
x
y
y
ecx
edx

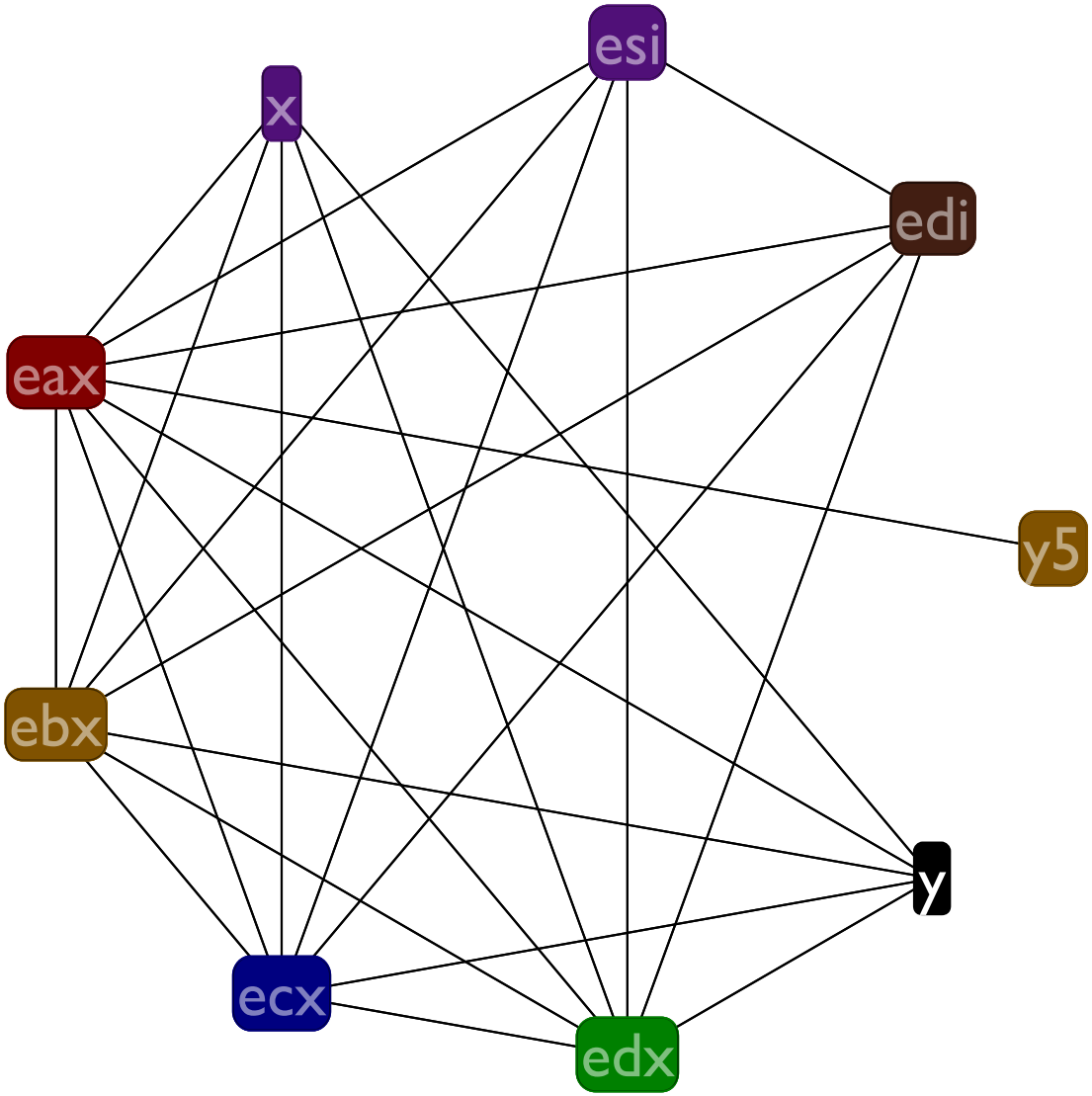

esi
x
edi
eax
y5
ebx
y
ecx
edx
y5
x
y

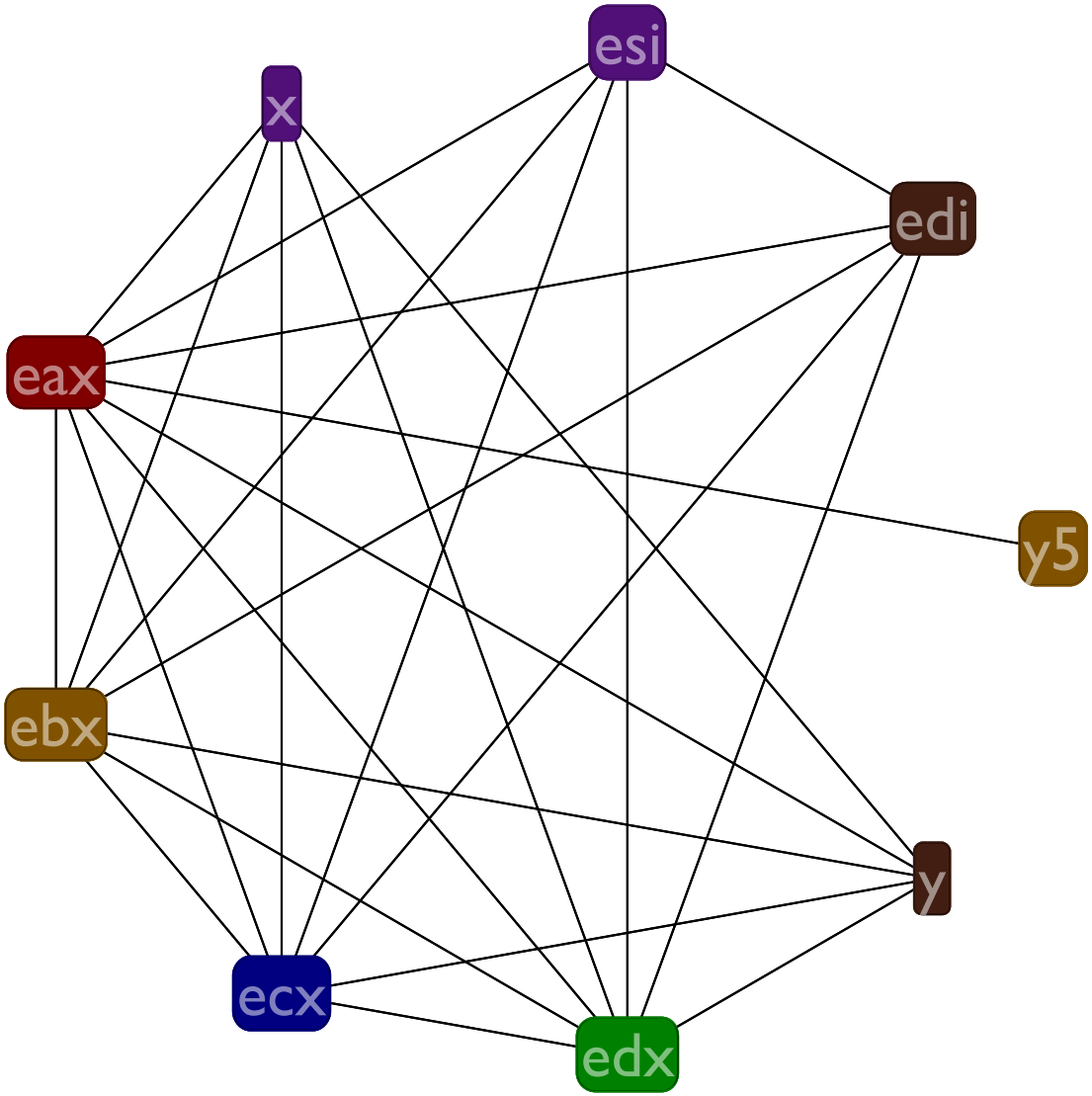

esi
x
edi
eax
y5
ebx
y
x
y
ecx
edx


esi
x
edi
eax
y5
ebx
y
x
y
ecx
edx


esi
x
edi
eax
y5
ebx
y
y
ecx
edx


esi
x
edi
eax
y5
ebx
y
y
ecx
edx


esi
x
edi
eax
y5
ebx
y
ecx
edx


esi
x
edi
eax
y5
ebx
y
ecx
edx

How to choose spills

- Pick variables with long live ranges and few uses to spill (callee saves have this profile)
- In this case, we can spill `z1` and `z2` to save us from spilling variables `x` and `y` that are frequently accessed

Finishing up register allocation

When you've spilled enough to successfully allocate the program, count the number of spills and adjust `esp` at the beginning of the function. (Recall calling convention.)

Calling convention reminder (see lecture03.txt)

```
(call s) ⇒ pushl $<new-label>
           pushl %ebp
           movl %esp, %ebp
           jmp <s>
           <new-label>:

(return) ⇒ movl %ebp, %esp
           popl %ebp
           ret // pop & goto

(tail-call s) ⇒ movl %ebp, %esp
                jmp <s>
```

(cleaned up version of call followed by return)

Registers: allocated

```
(:f
 (esp -= 8)
 ((mem ebp -4) <- esi)
 ((mem ebp -8) <- edi)
 (esi <- eax)
 (call :g)
 (edi <- eax)
 (eax += esi)
 (call :h)
 (ebx <- edi)
 (ebx *= 5)
 (eax += ebx)
 (esi <- (mem ebp -4))
 (edi <- (mem ebp -8))
 (return))
```

Coalescing

If we see a `(x <- y)` instruction, we might be able to just change all of the `x`'s into `y`'s

That's called coalescing `x` and `y`

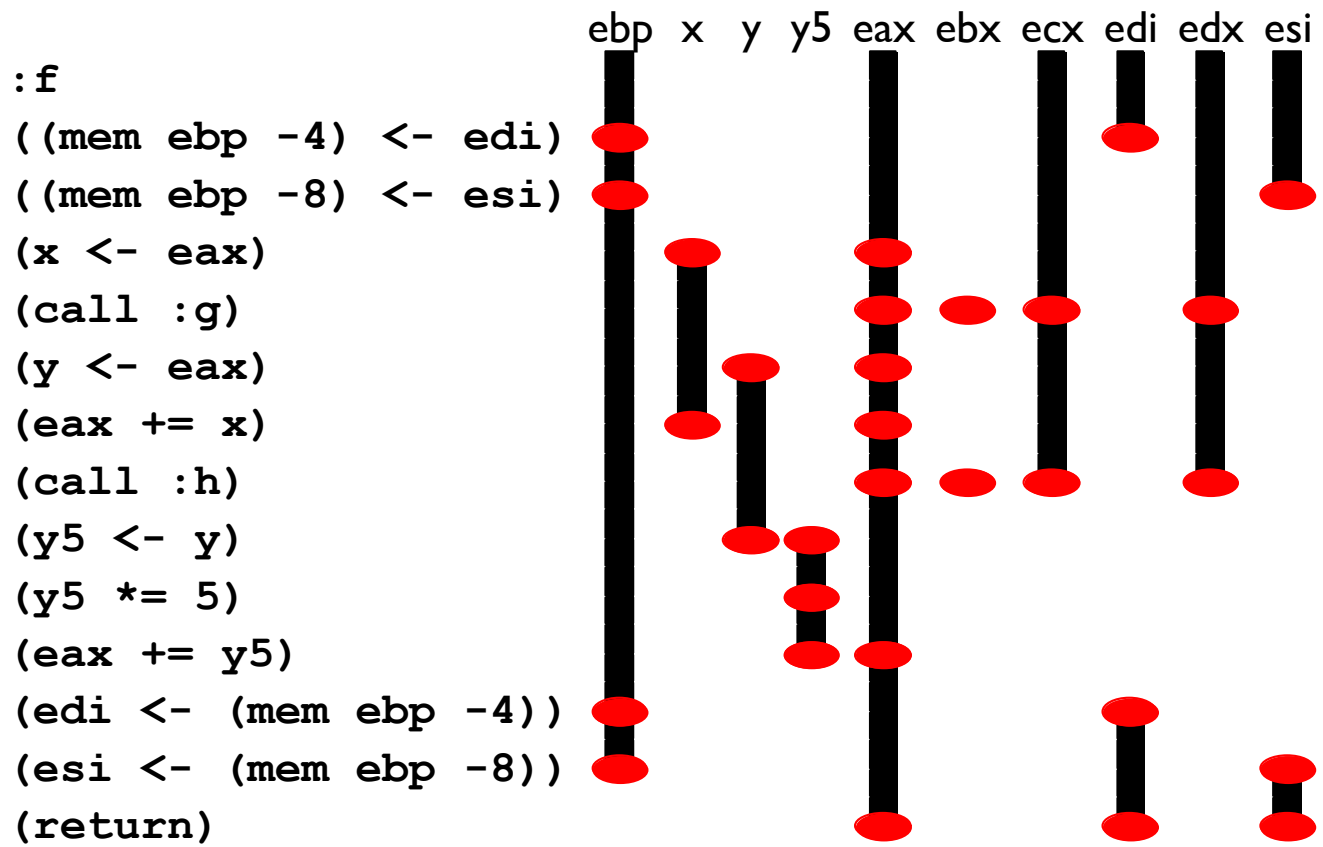
Lets use coalescing to remove `y` and `y5` from our example program

Coalescing example

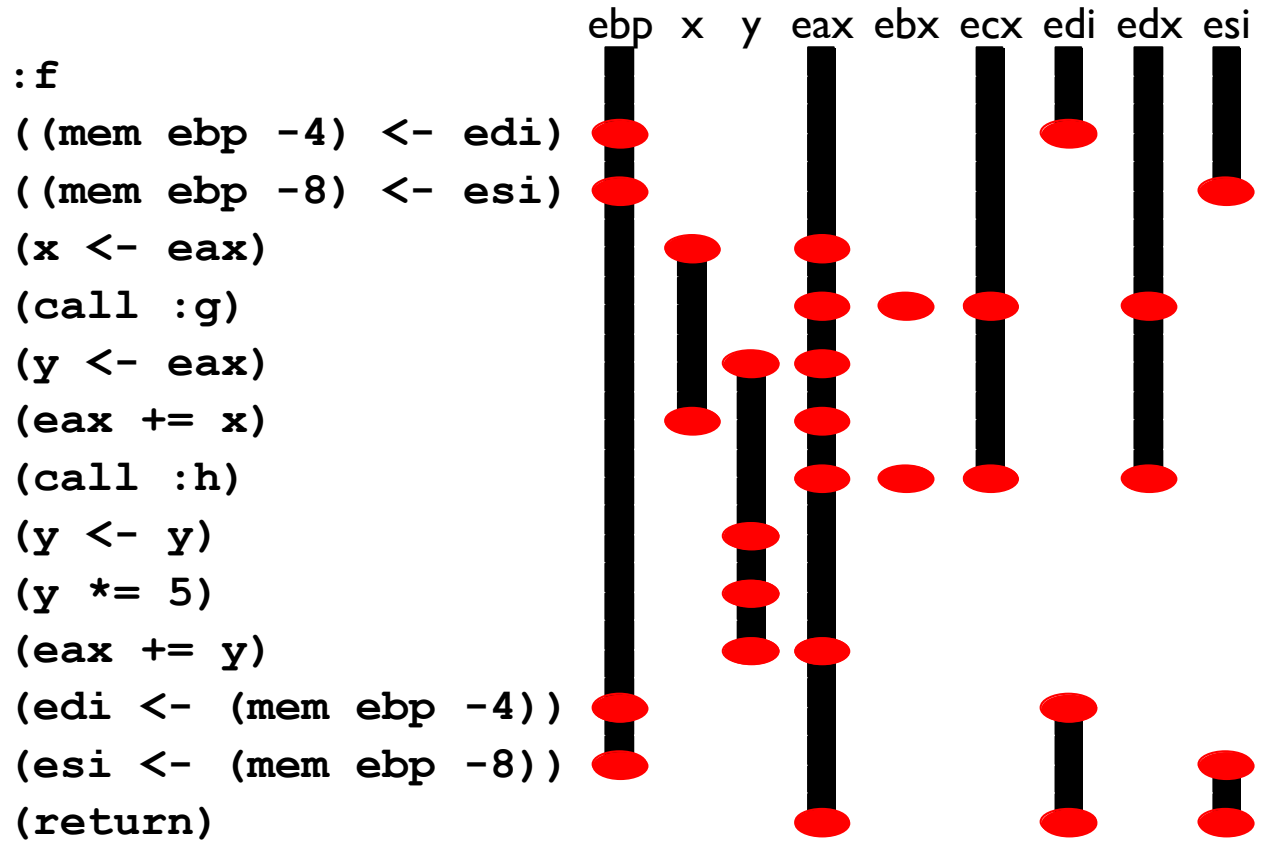
```
:f
((mem ebp -4) <- edi)
((mem ebp -8) <- esi)
(x <- eax)
(call :g)
(y <- eax)
(eax += x)
(call :h)
(y5 <- y)
(y5 *= 5)
(eax += y5)
(edi <- (mem ebp -4))
(esi <- (mem ebp -8))
(return)
```

```
:f
((mem ebp -4) <- edi)
((mem ebp -8) <- esi)
(x <- eax)
(call :g)
(y <- eax)
(eax += x)
(call :h)
(y <- y)
(y *= 5)
(eax += y)
(edi <- (mem ebp -4))
(esi <- (mem ebp -8))
(return)
```

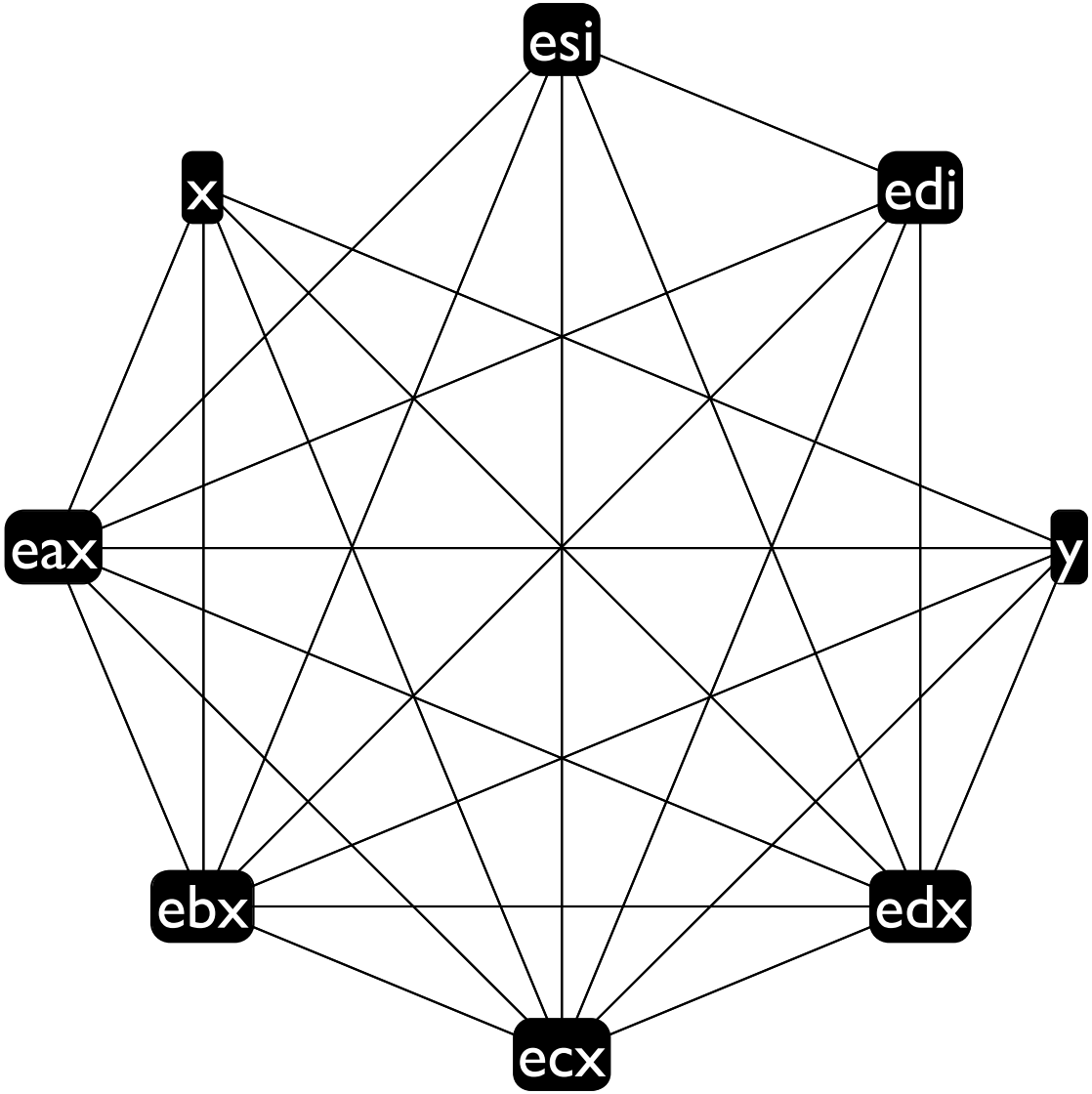
Live ranges before coalescing

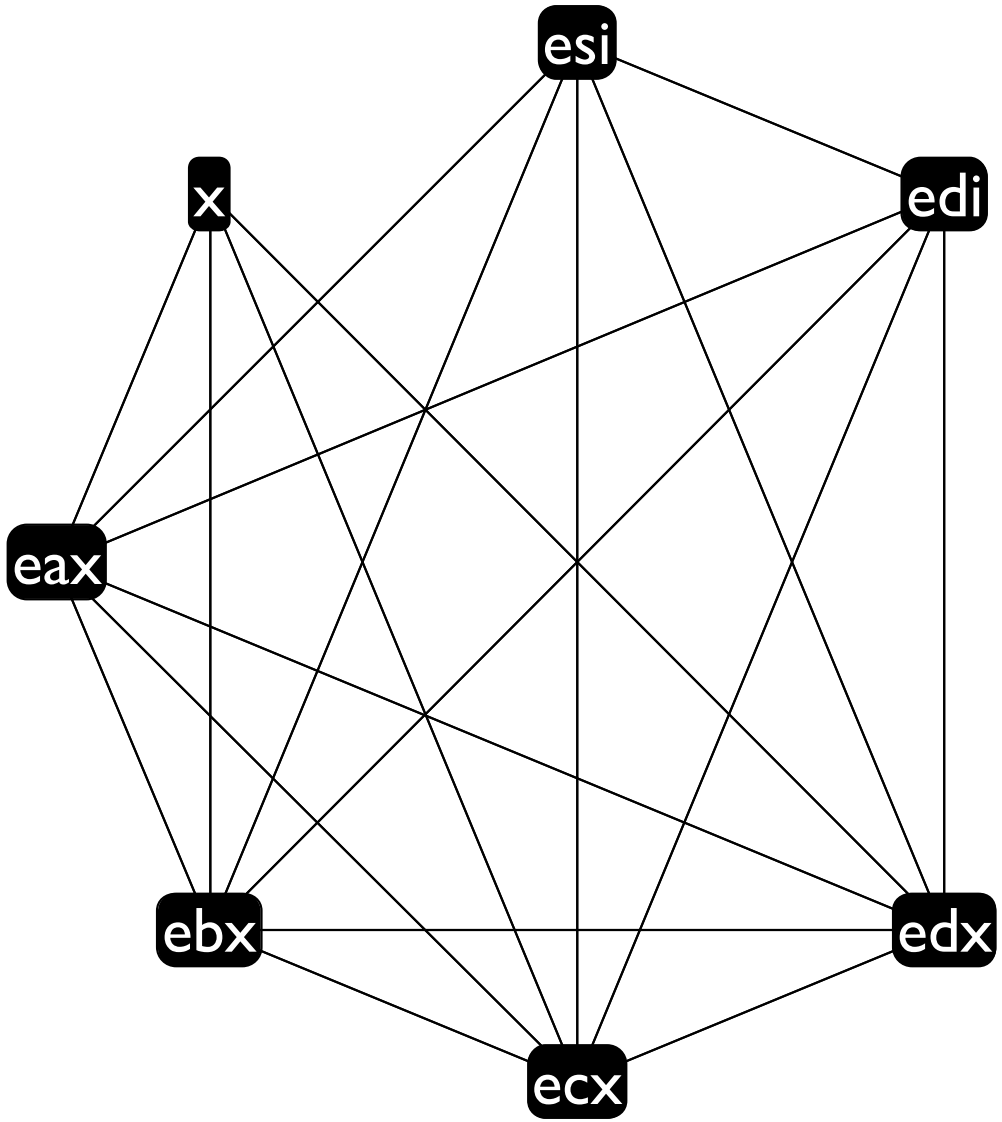


Live ranges after coalescing



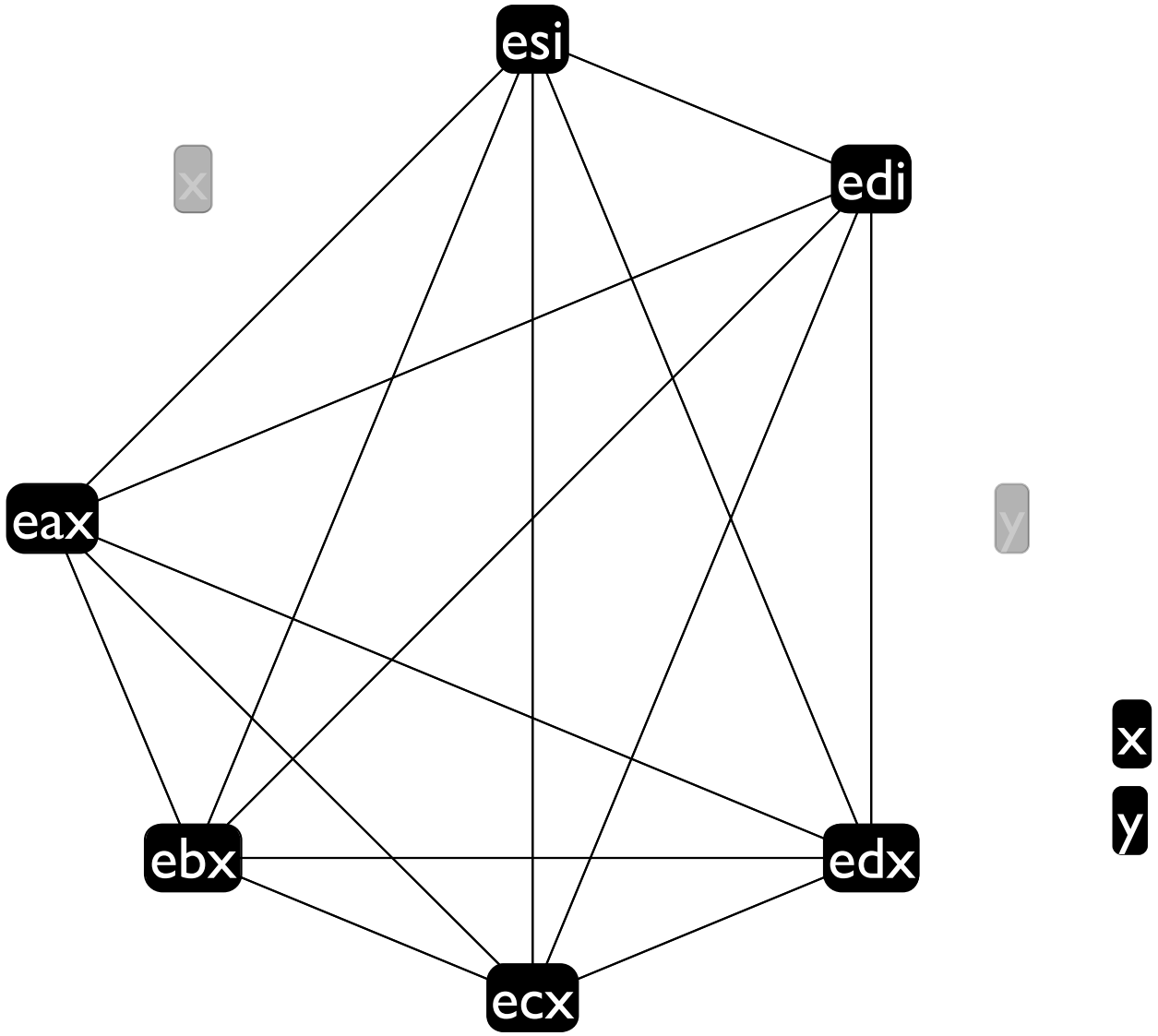
Lets try to register allocate the coalesced graph

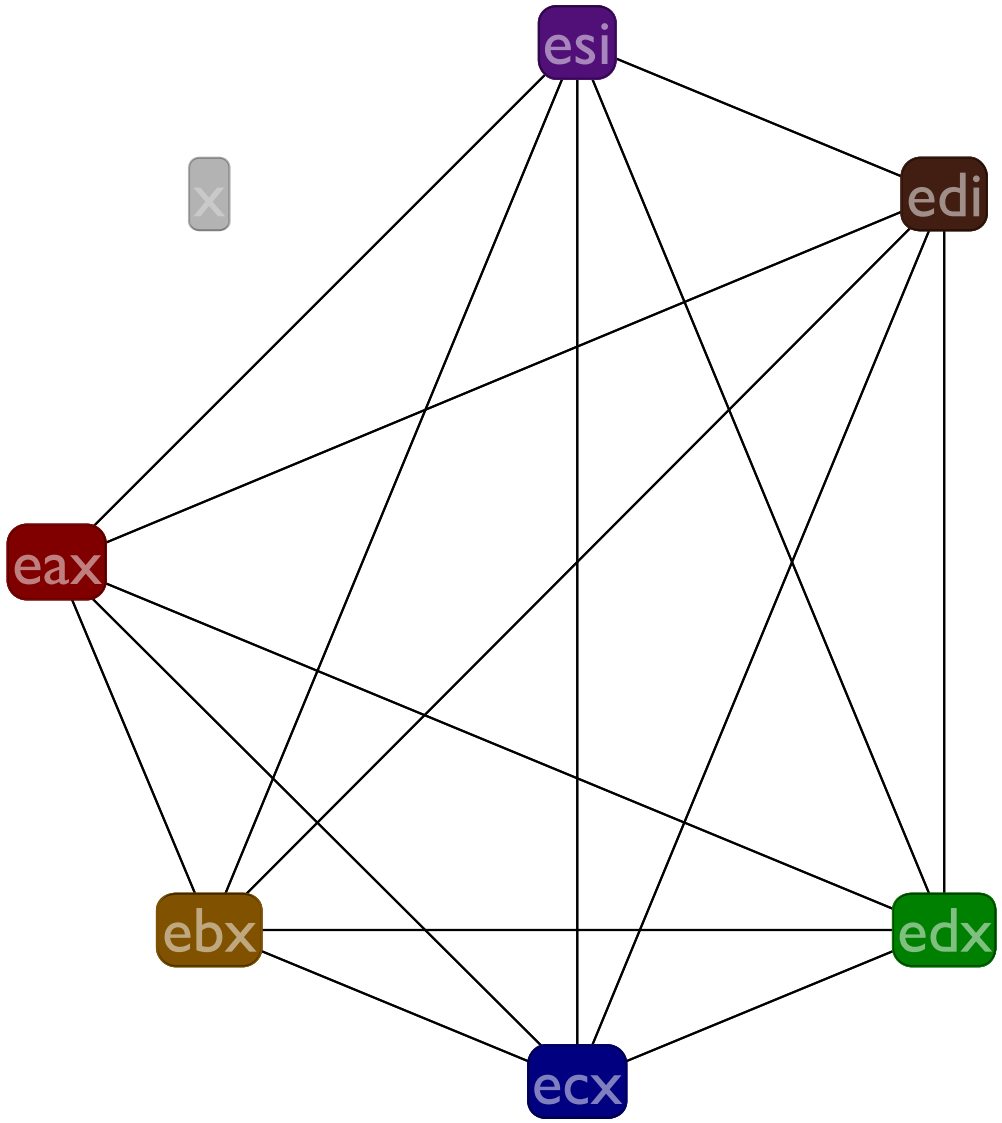

esi
x
edi
eax
y
ebx
edx
ecx



y

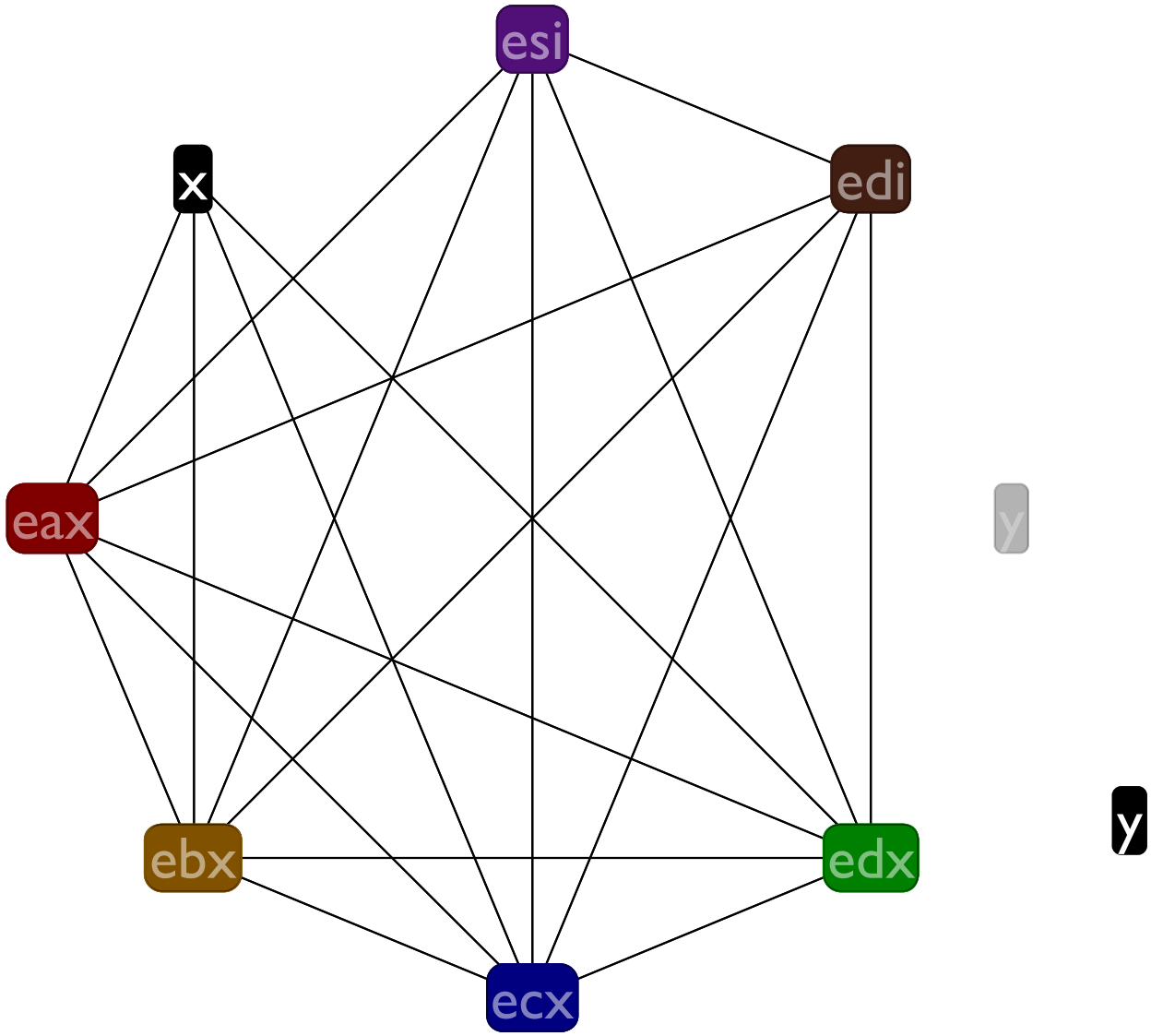
y

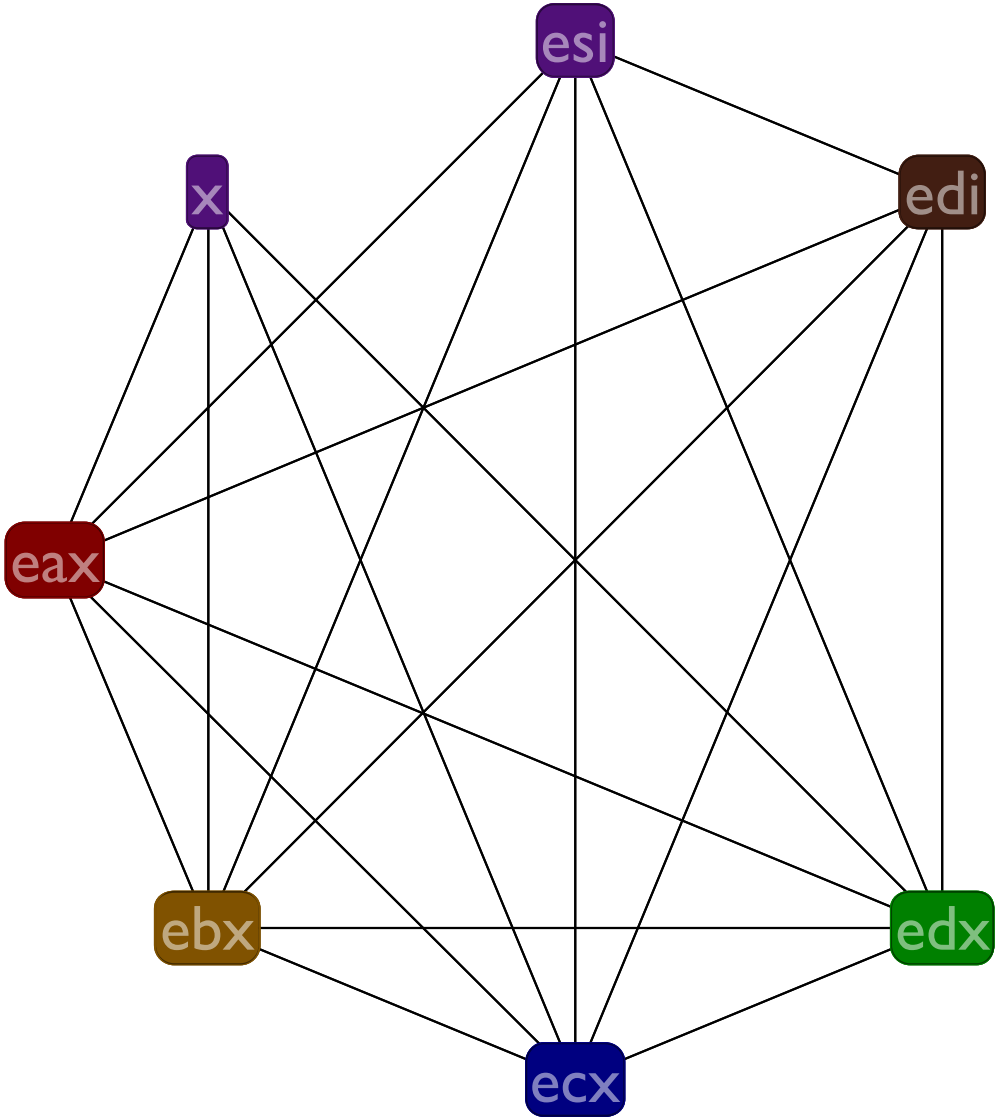

esi
x
edi
eax
y
x
y
ebx
edx
ecx

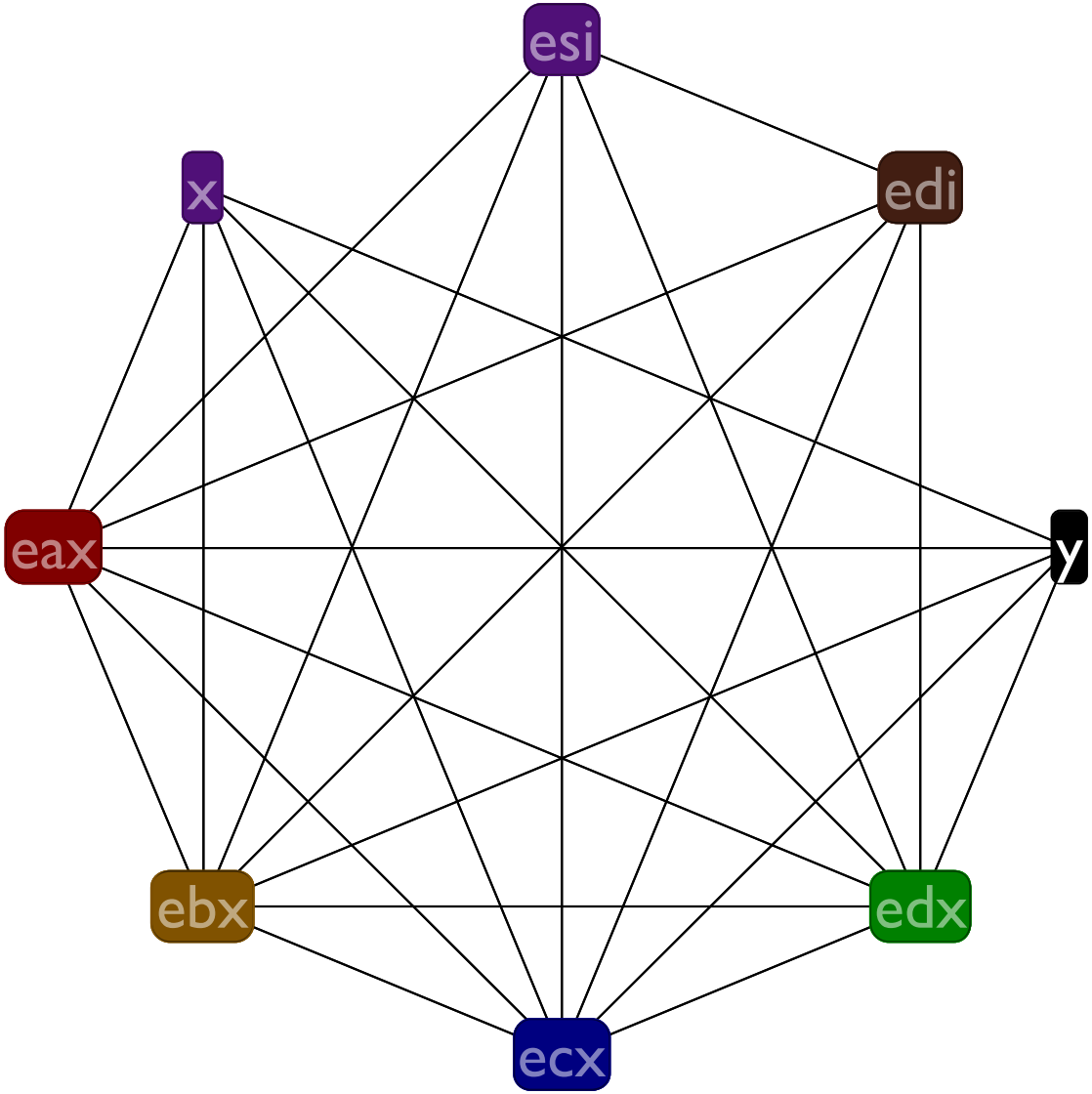


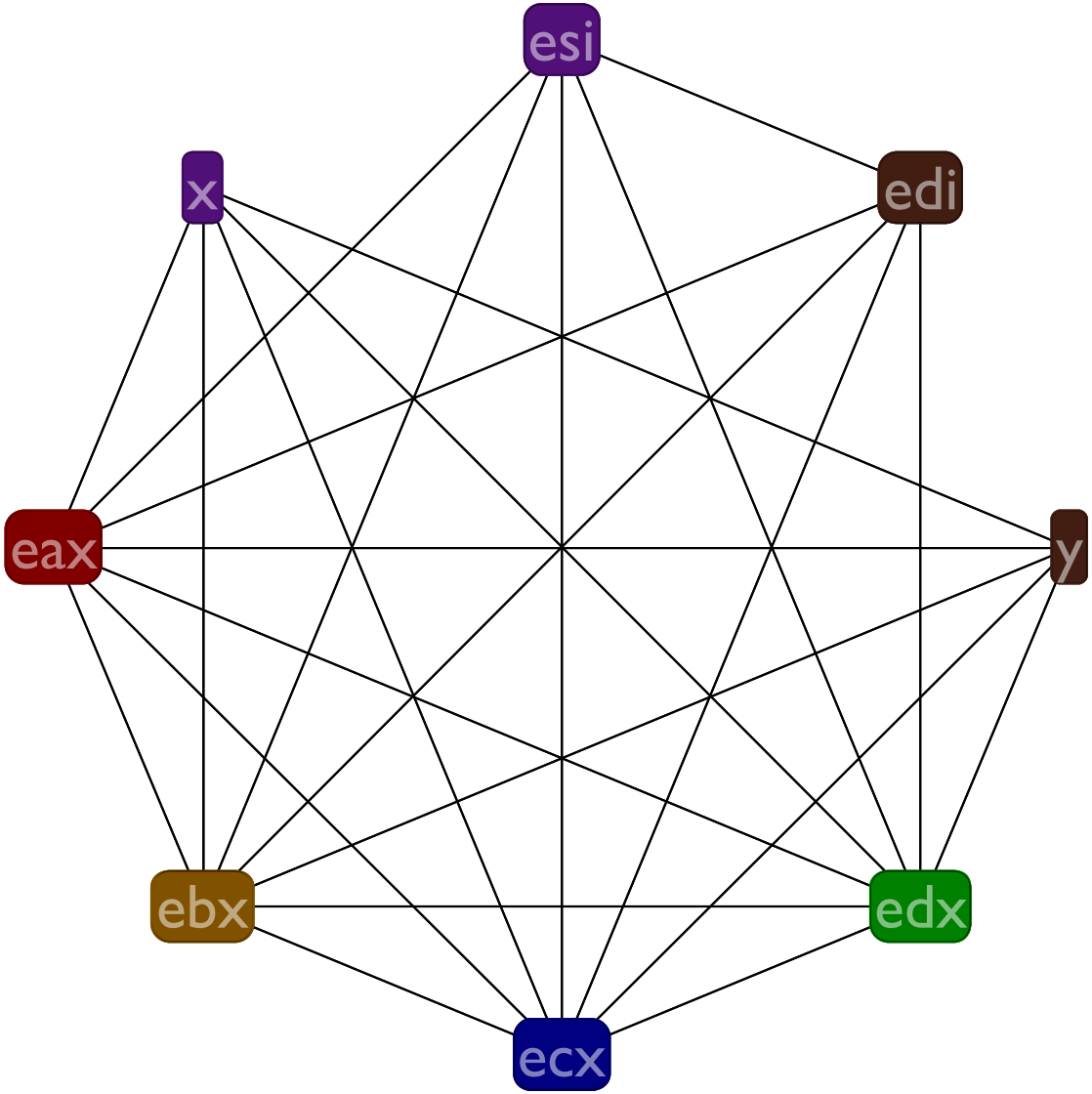
x

y


esi
x
edi
eax
y
y
ebx
edx
ecx

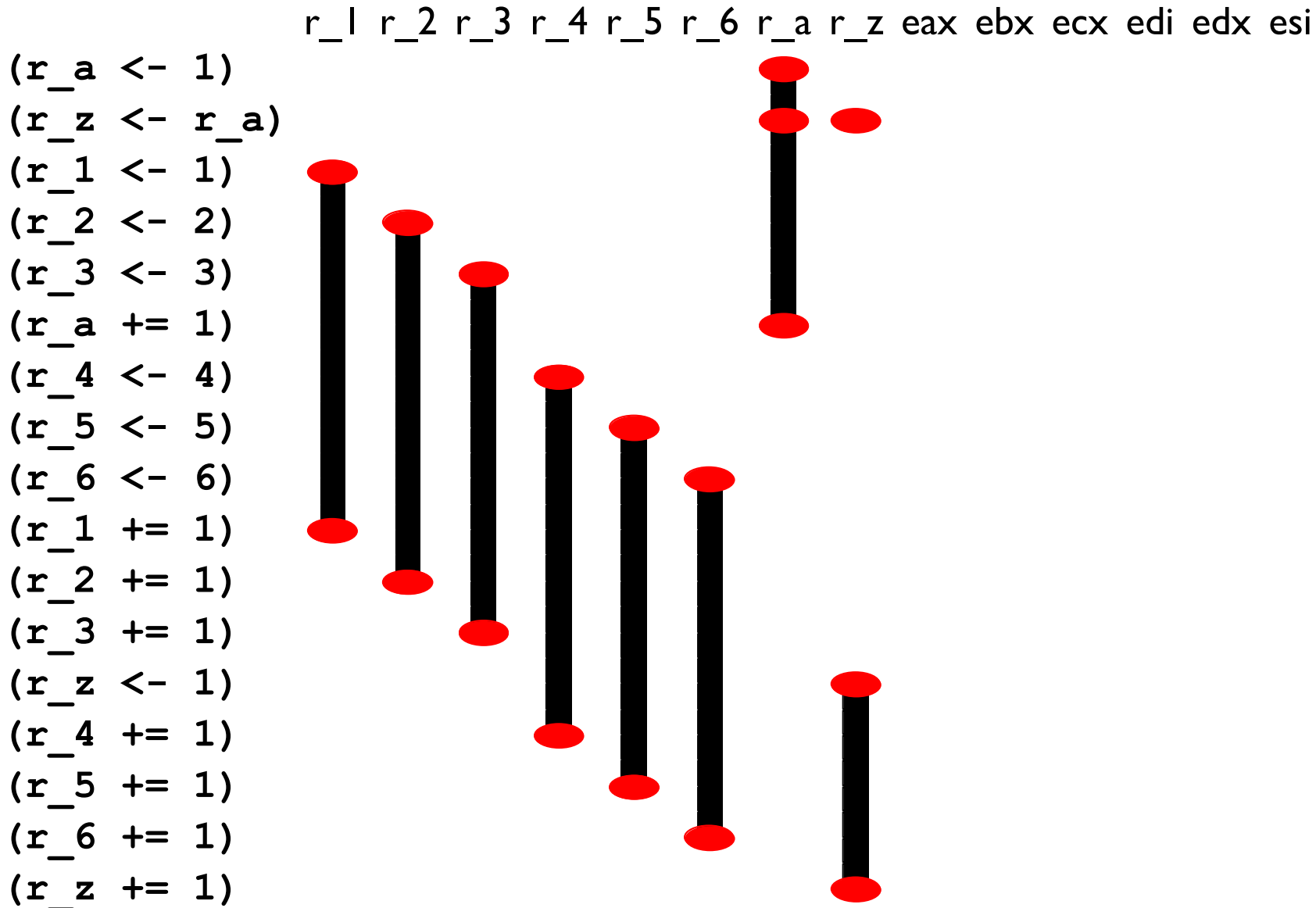

esi
x
edi
eax
y
y
ebx
edx
ecx

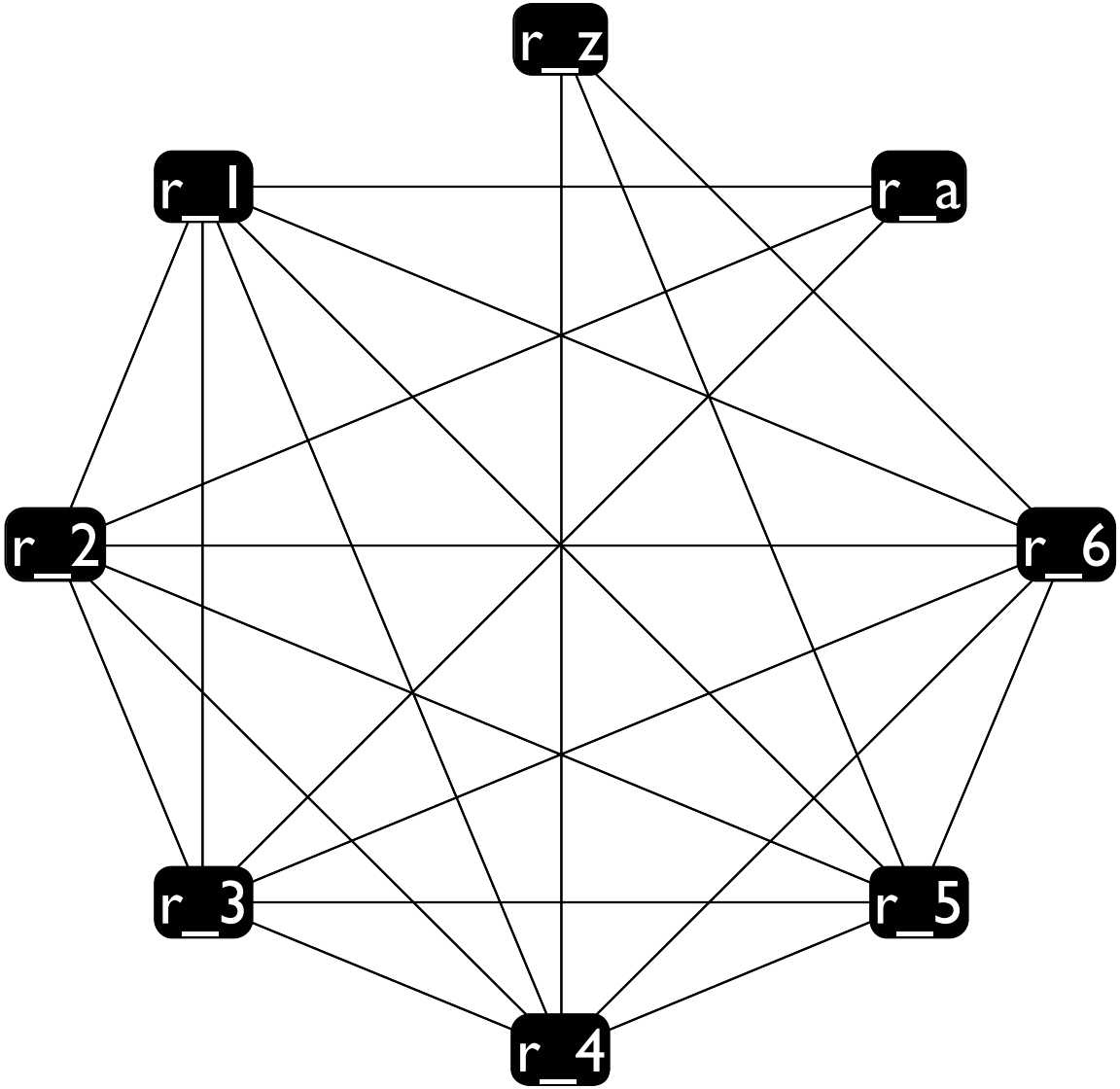

esi
x
edi
eax
y
ebx
edx
ecx

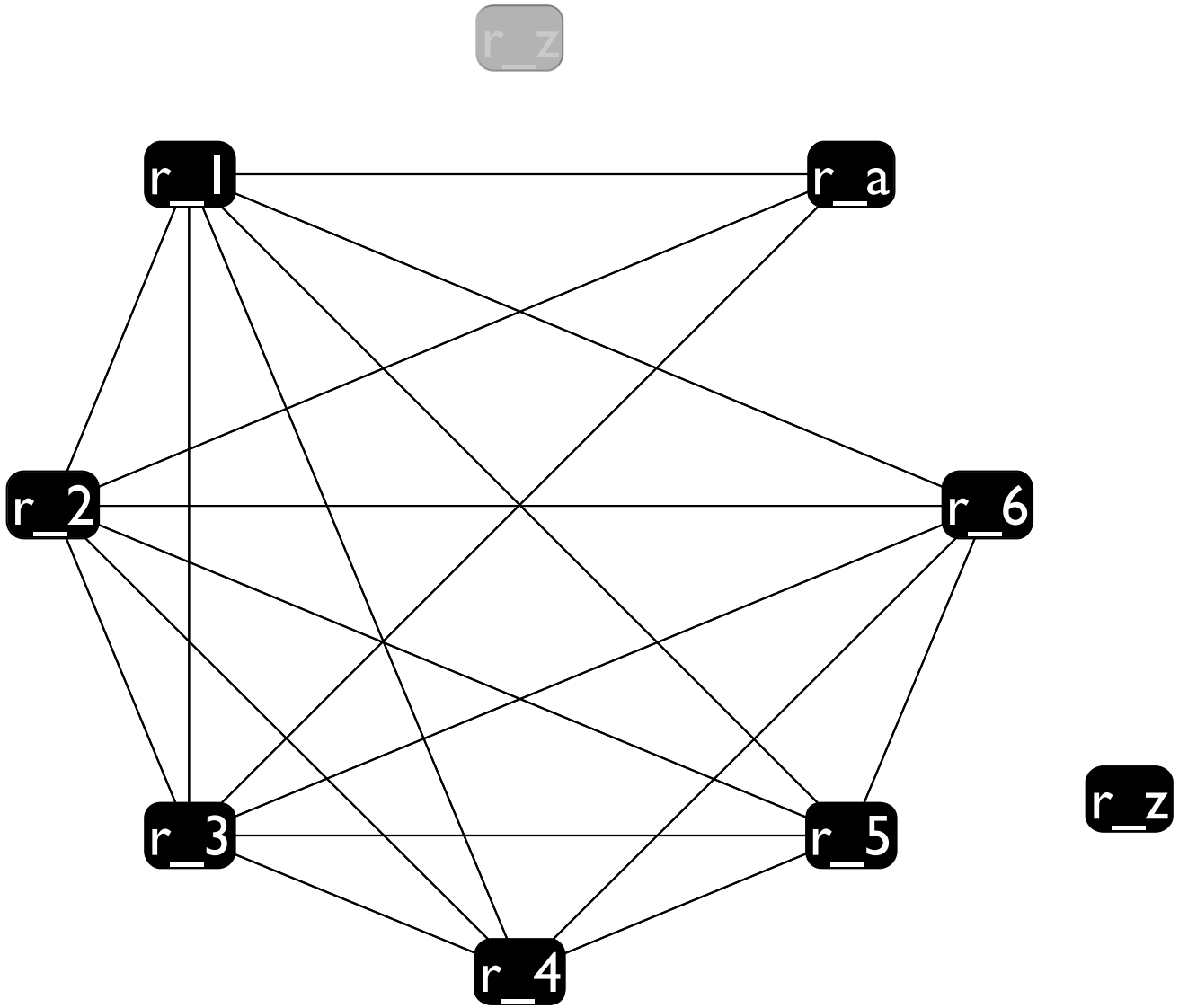

esi
x
edi
eax
y
ebx
edx
ecx

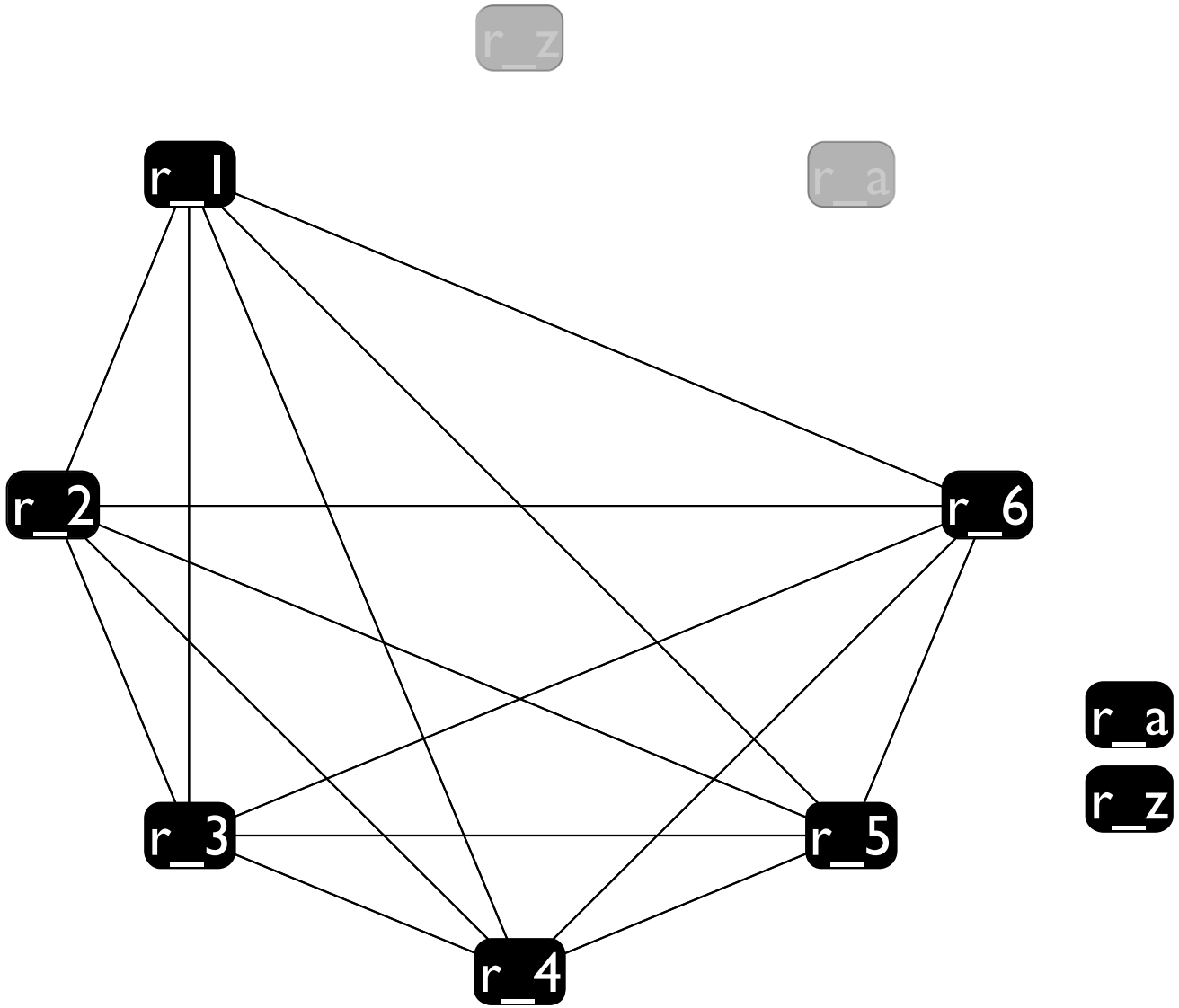
It worked that time, but this doesn't always work

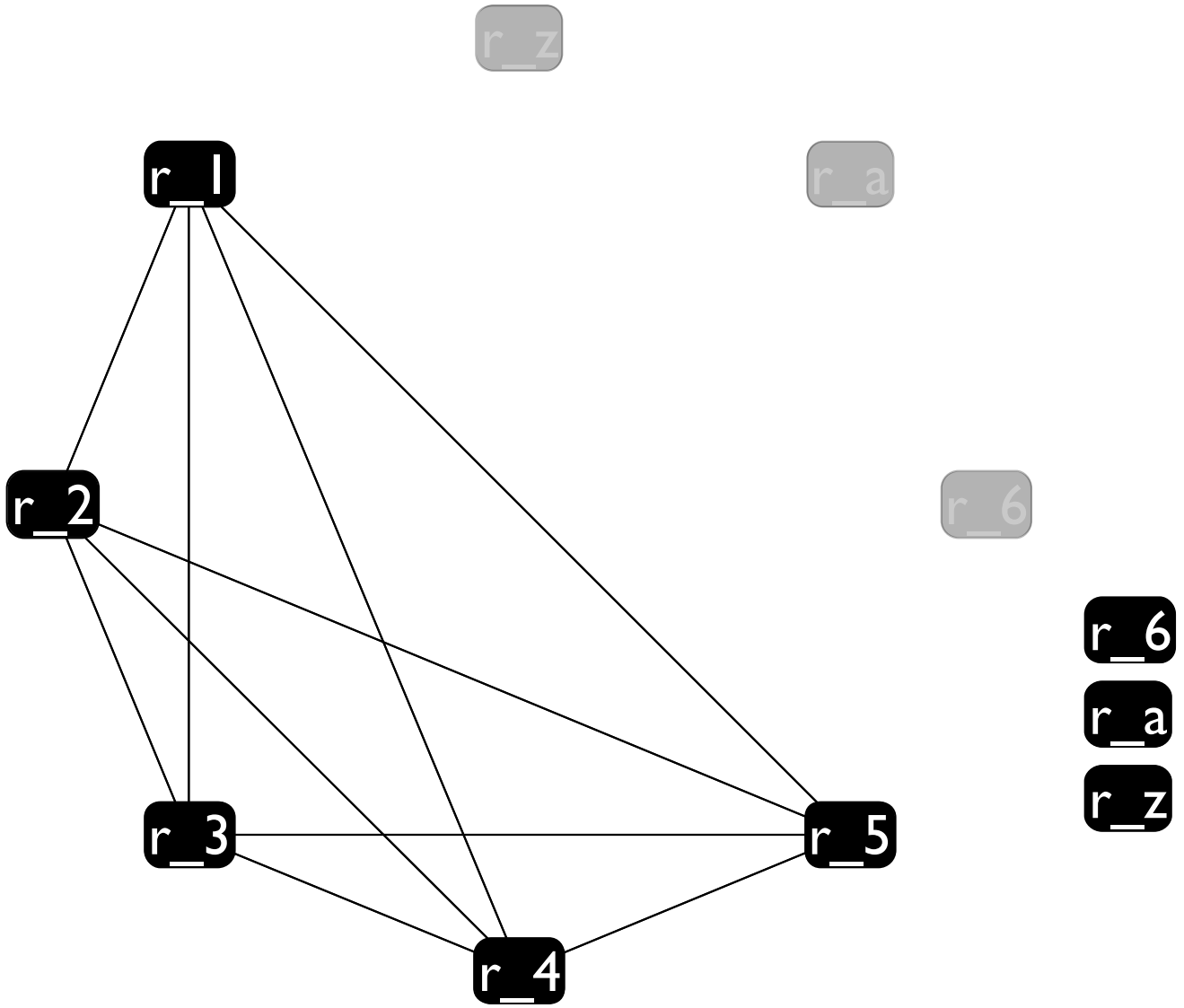
Coalescing Problem

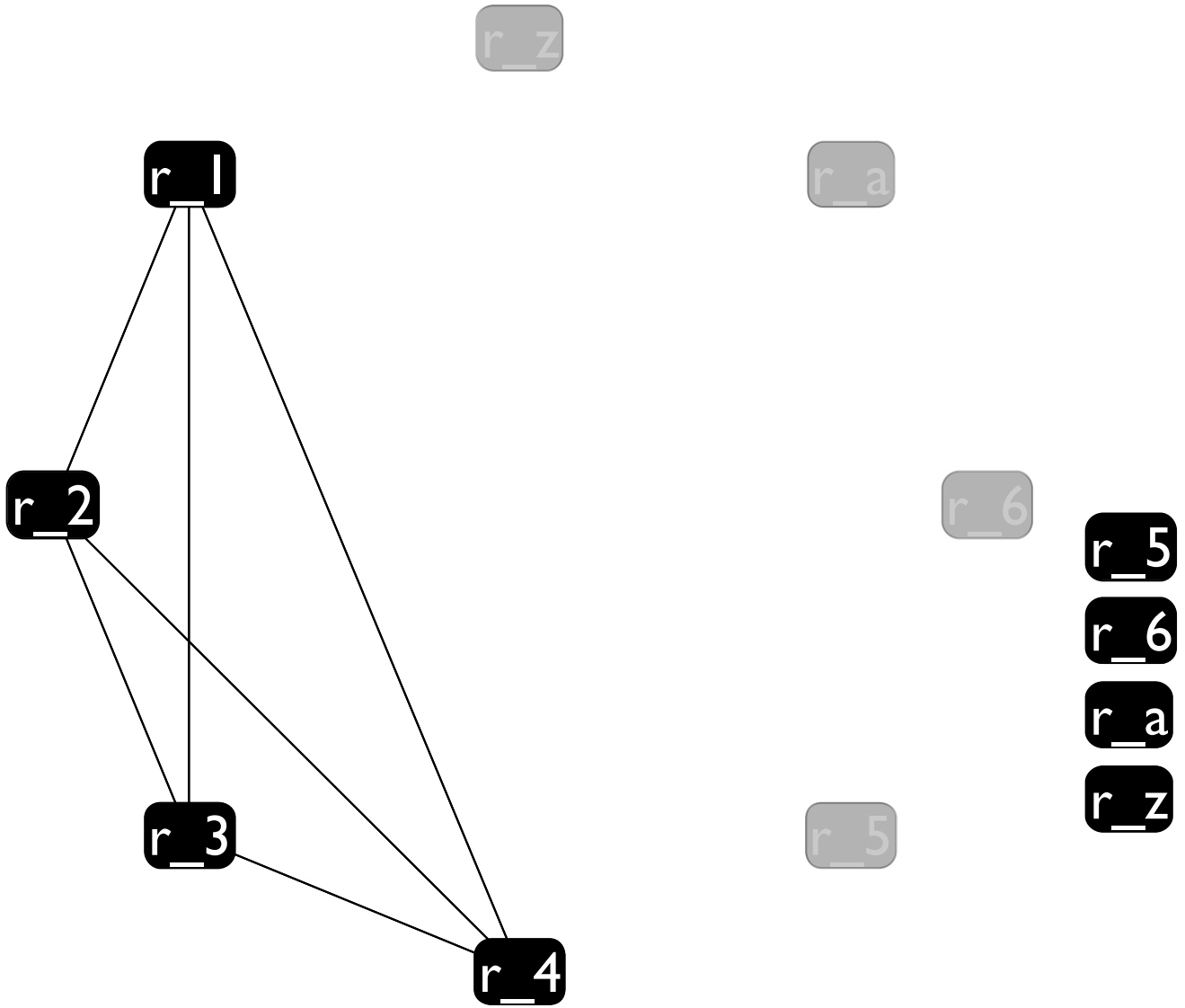


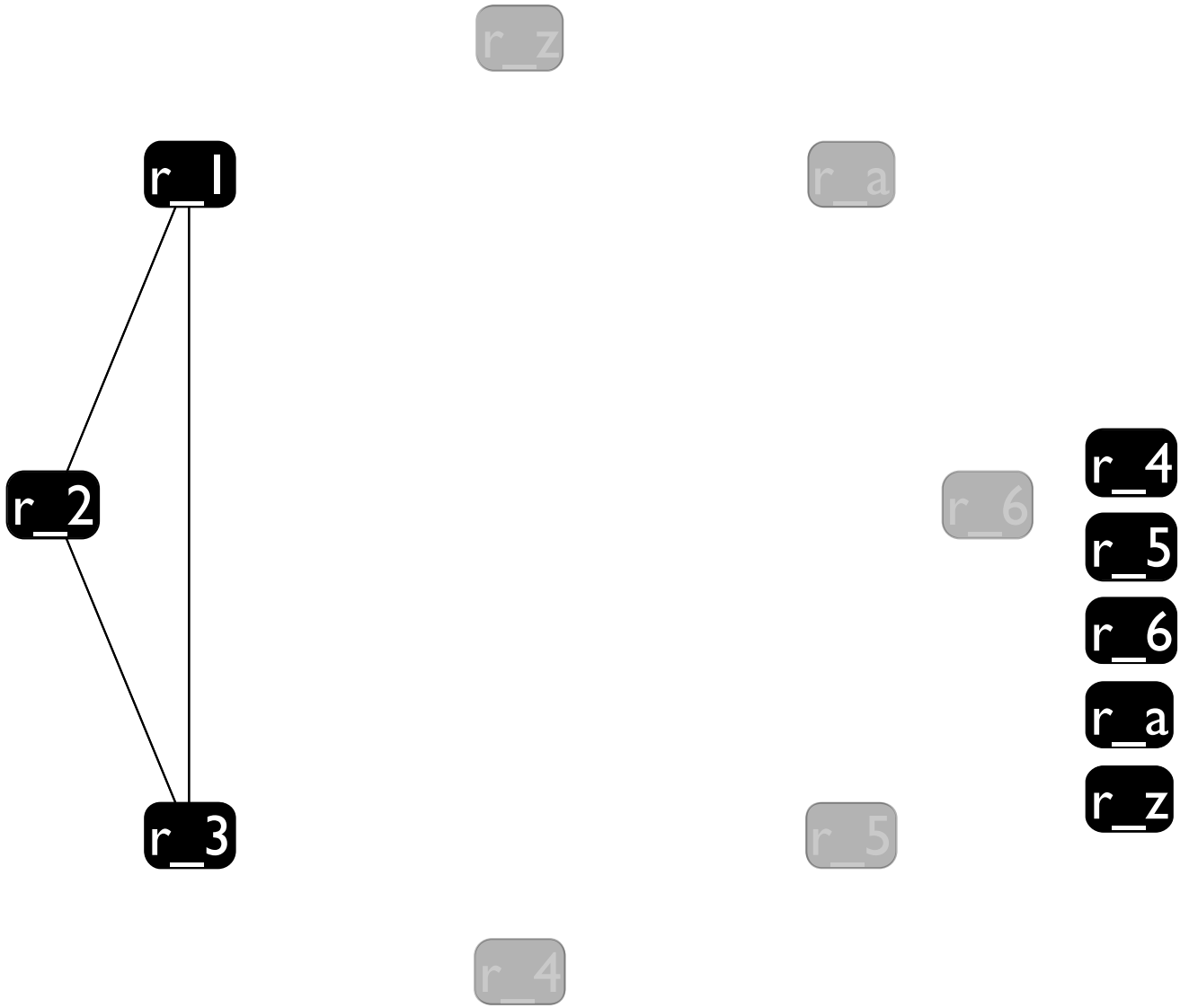

r_z
r_1
r_a
r_2
r_6
r_3
r_5
r_4

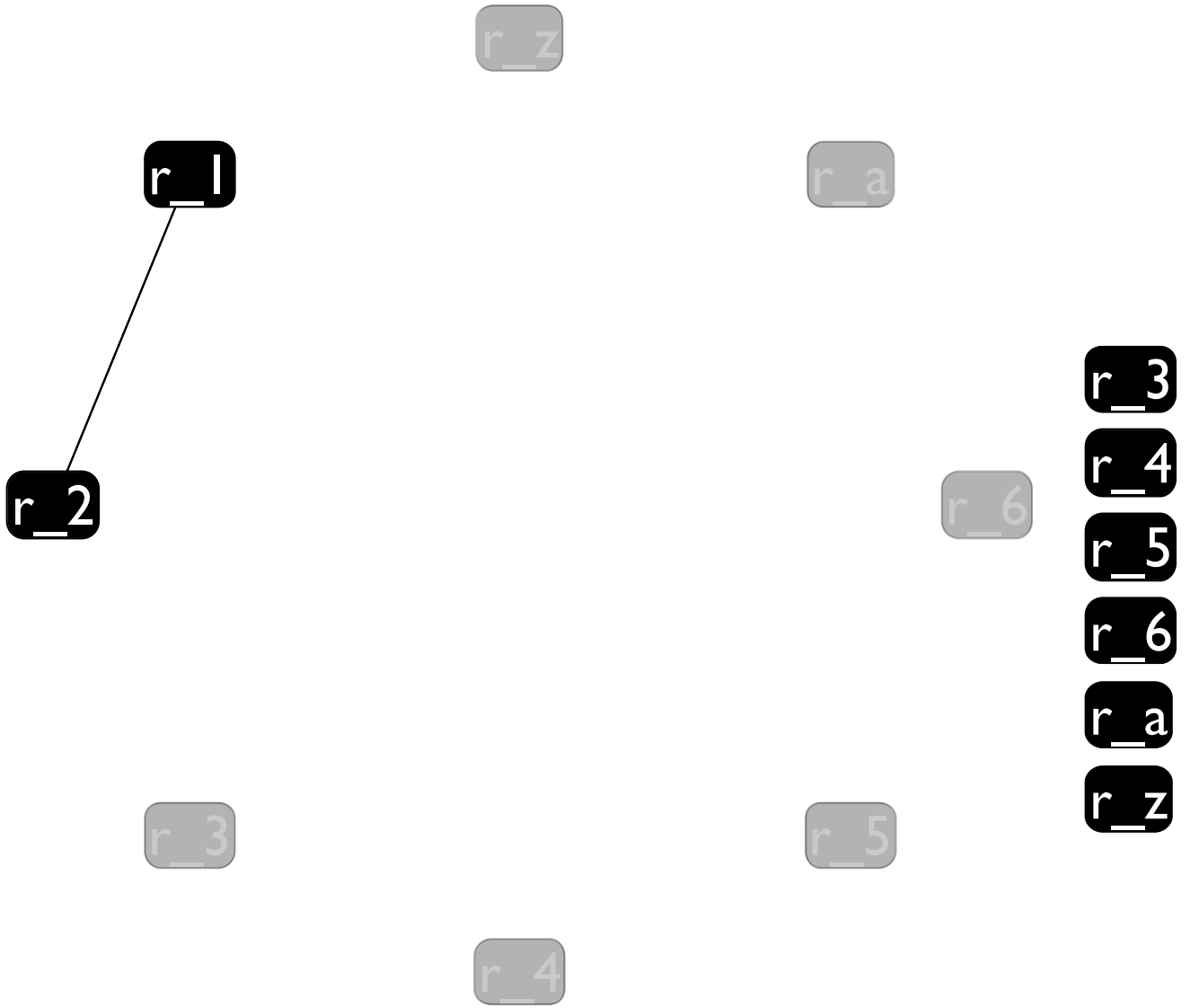

r_z
r_1
r_a
r_2
r_6
r_z
r_3
r_5
r_4

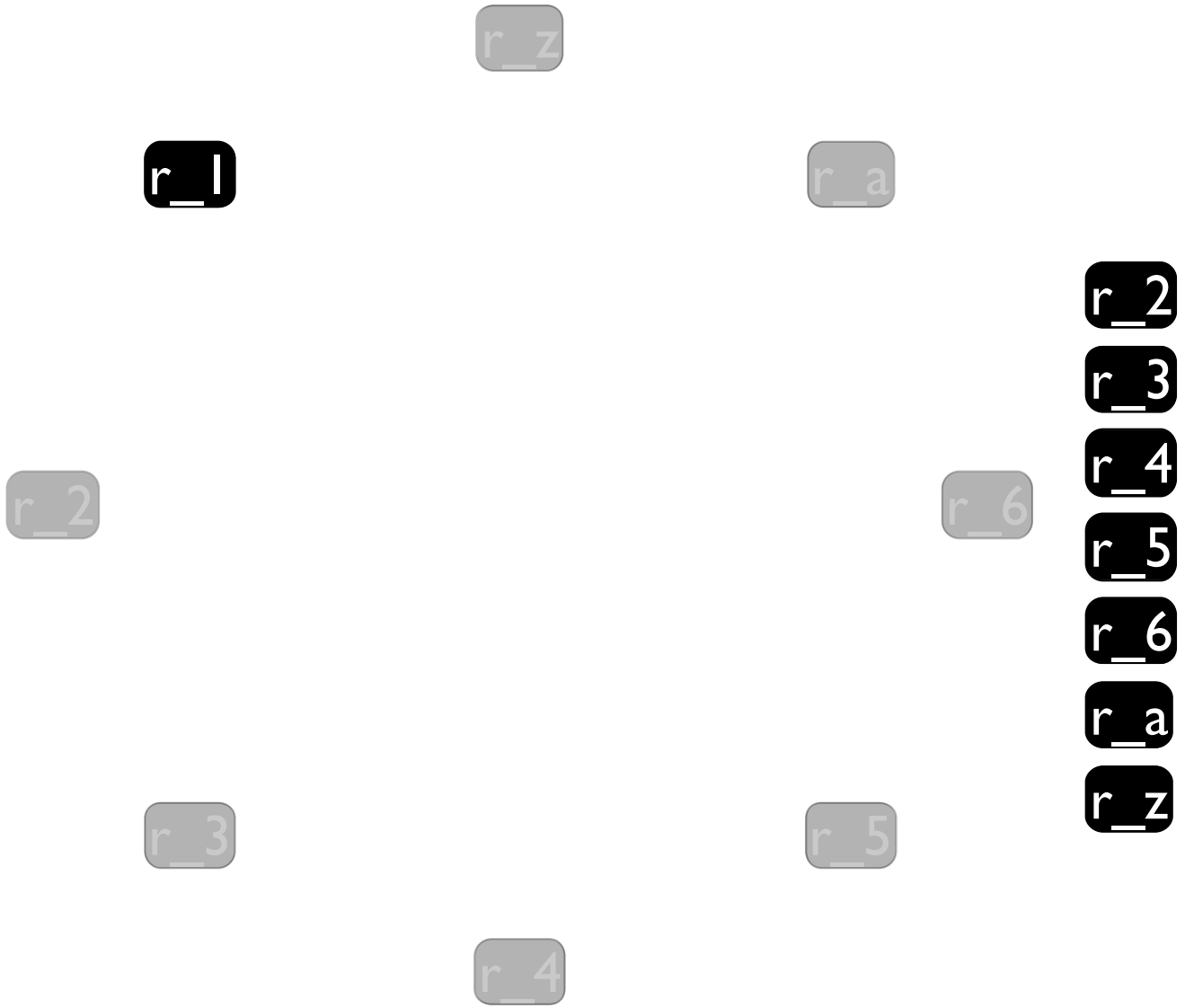

r_z
r_a
r_1
r_2
r_6
r_3
r_5
r_4
r_a
r_z

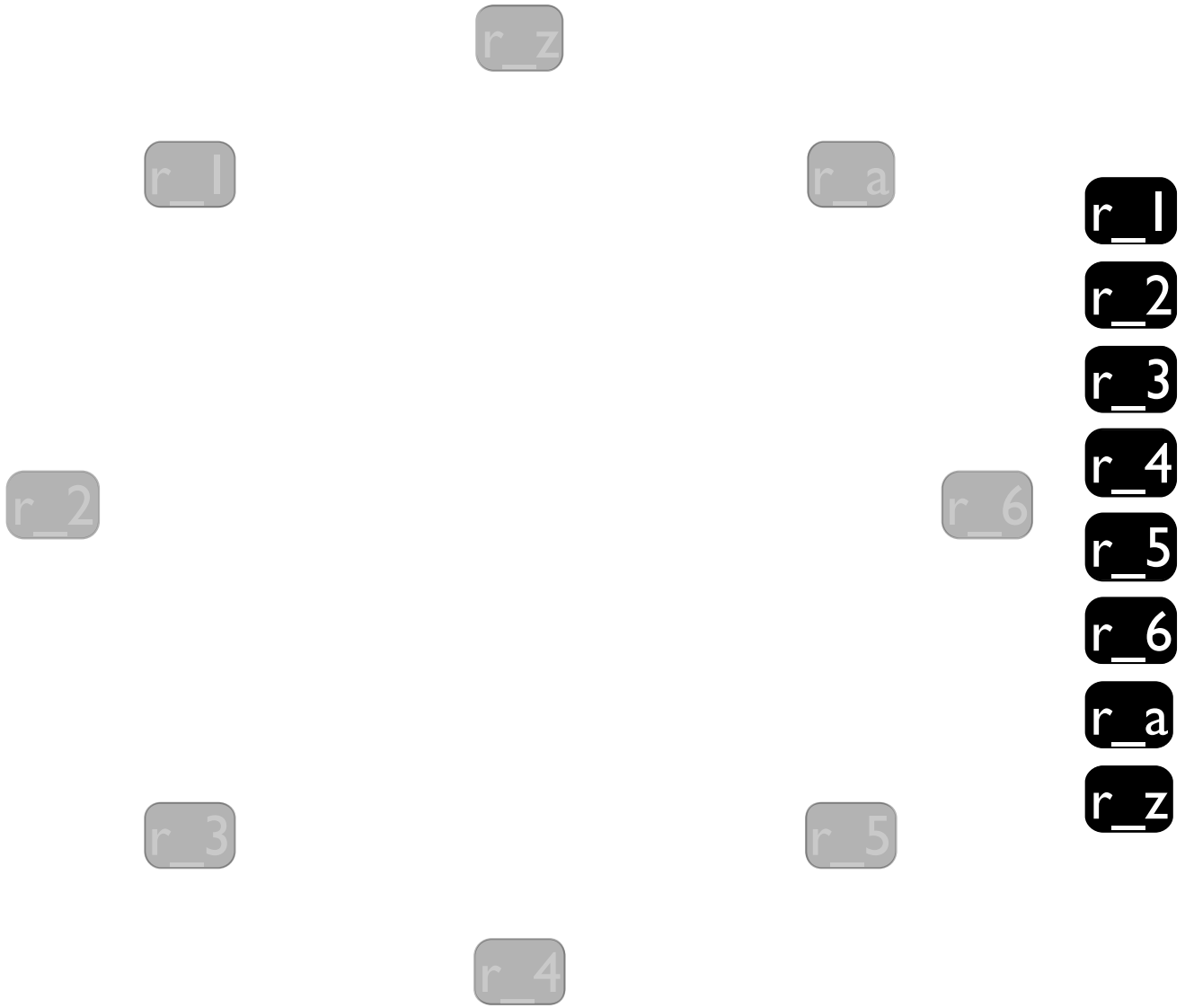

r_z
r_1
r_a
r_2
r_6
r_6
r_a
r_z
r_3
r_5
r_4

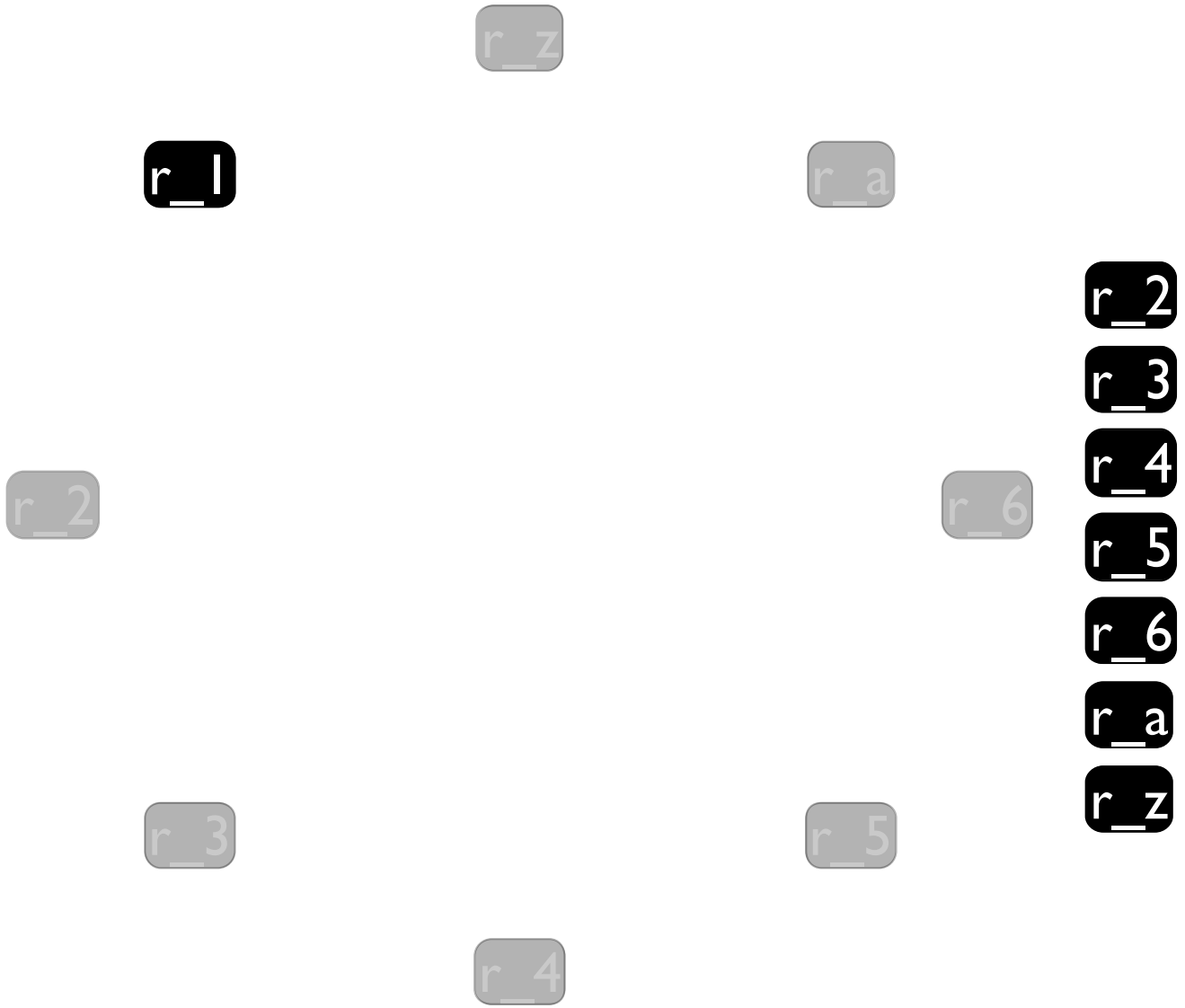

r_z
r_1
r_a
r_2
r_6
r_5
r_6
r_a
r_z
r_3
r_5
r_4

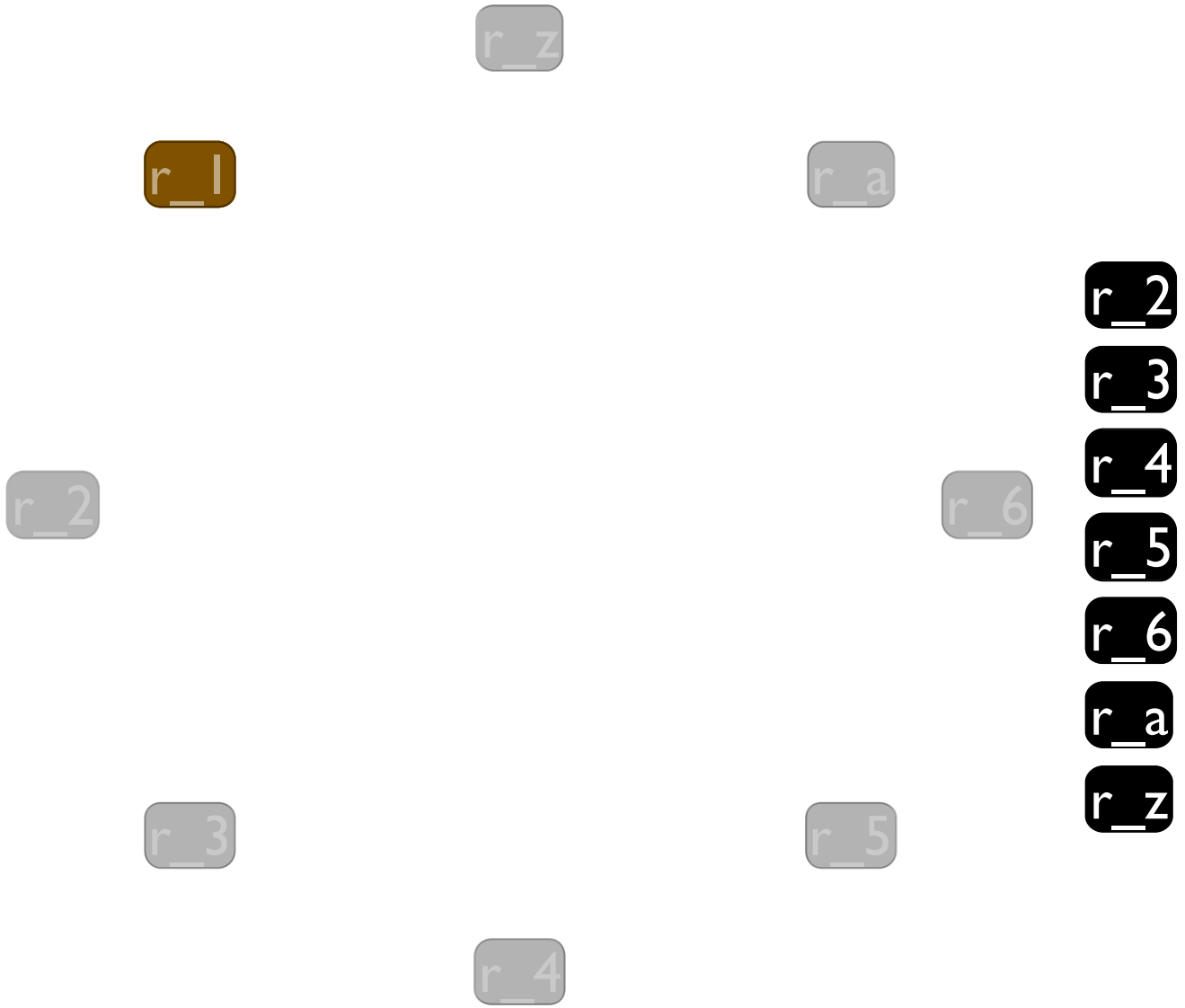

r_z
r_1
r_a
r_2
r_6
r_4
r_5
r_6
r_a
r_z
r_3
r_5
r_4

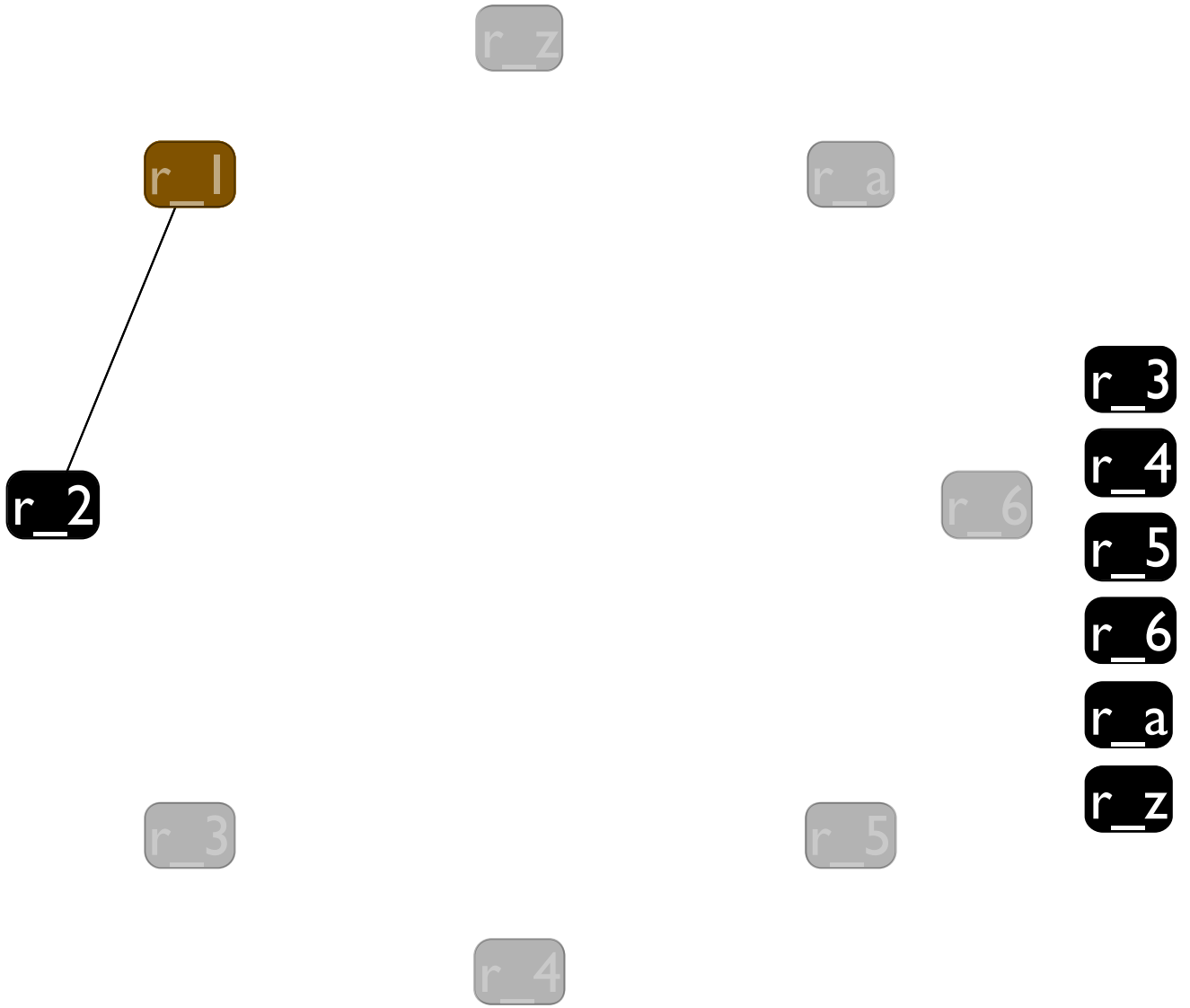

r_z
r_1
r_a
r_2
r_6
r_3
r_4
r_5
r_6
r_a
r_z
r_3
r_5
r_4

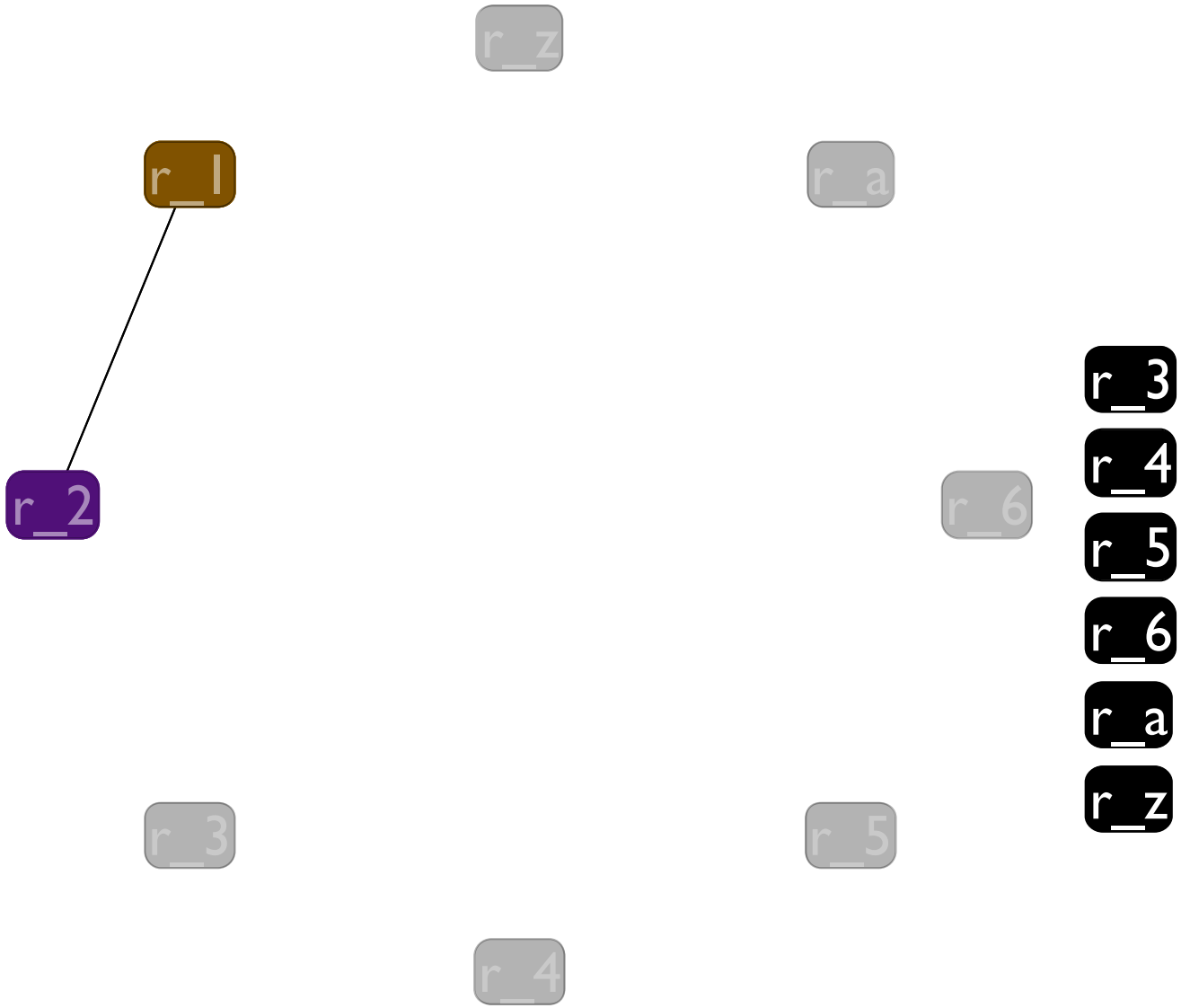

r_z
r_1
r_a
r_2
r_6
r_3
r_5
r_4
r_2
r_3
r_4
r_5
r_6
r_a
r_z

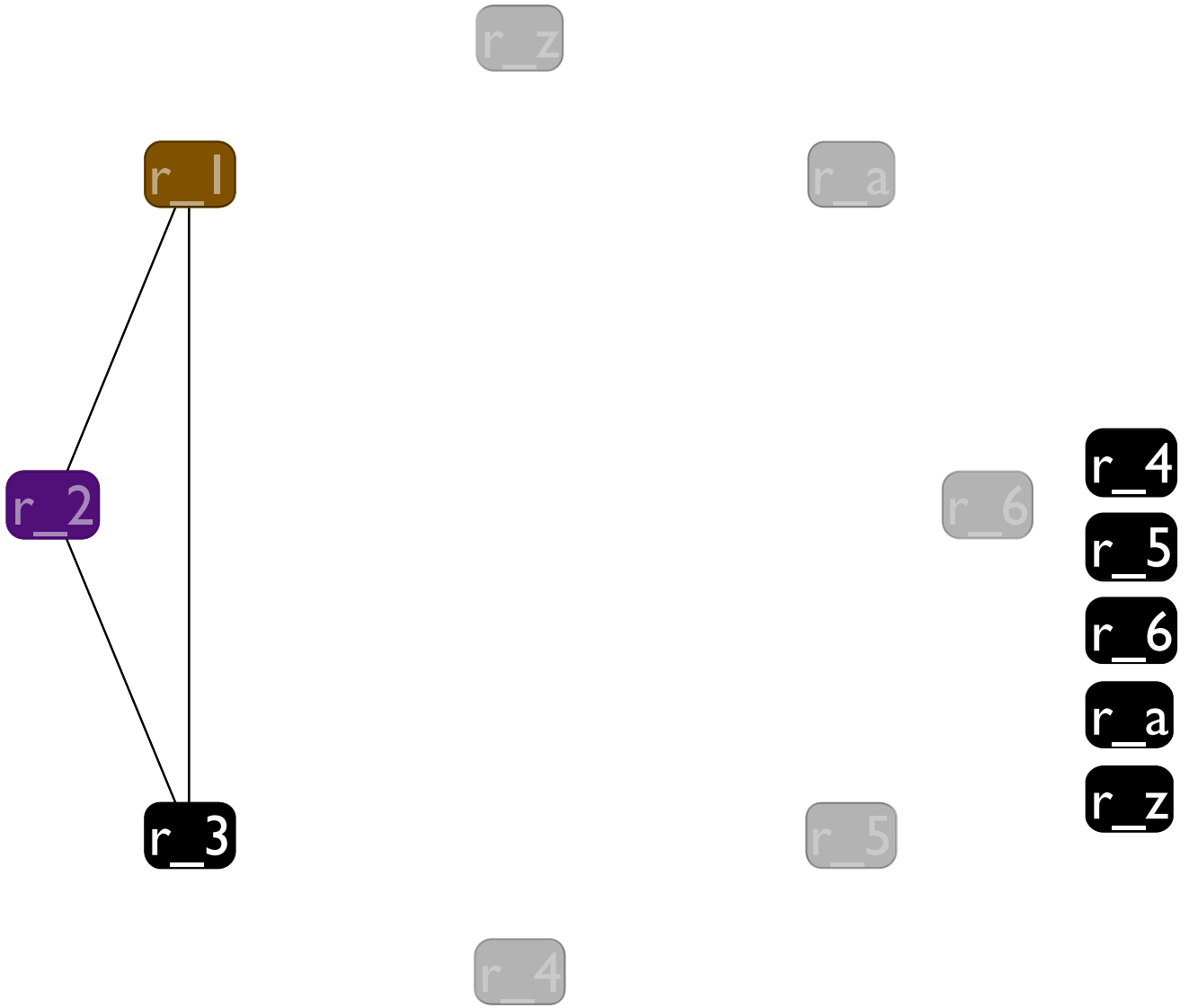

r_z
r_1
r_a
r_2
r_6
r_3
r_5
r_4
r_1
r_2
r_3
r_4
r_5
r_6
r_a
r_z

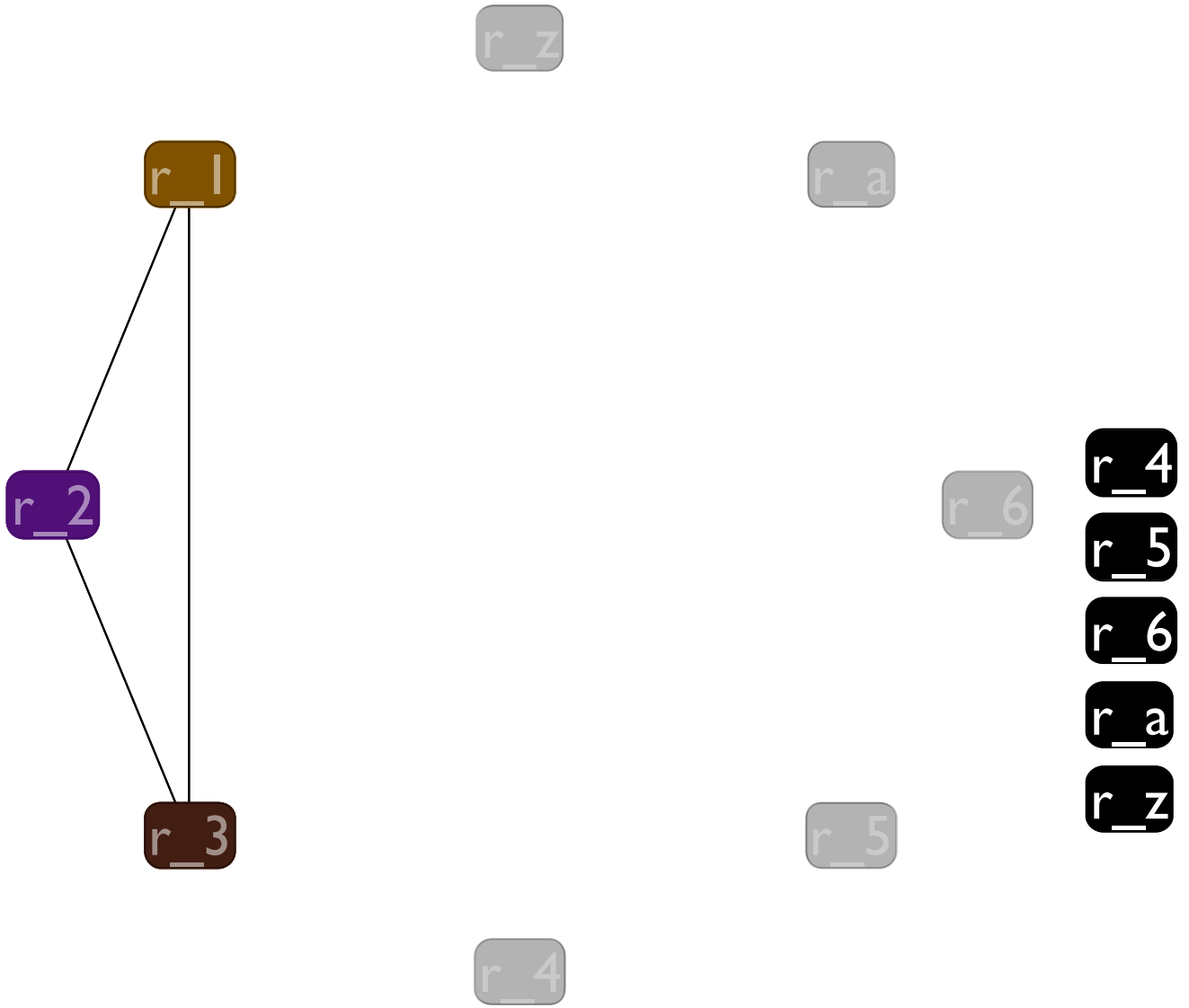

r_z
r_1
r_a
r_2
r_6
r_3
r_5
r_4
r_2
r_3
r_4
r_5
r_6
r_a
r_z

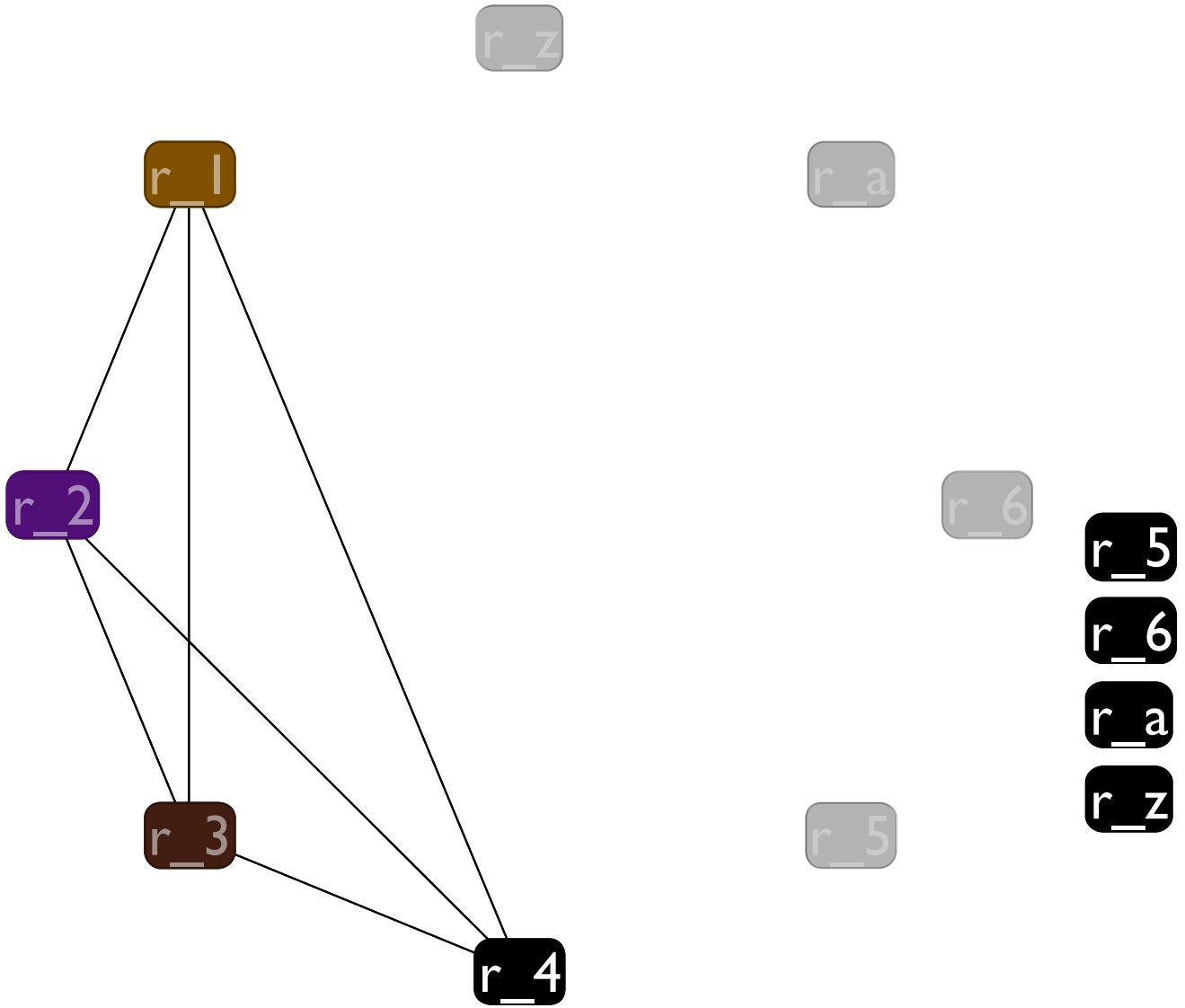

r_z
r_1
r_a
r_2
r_6
r_3
r_5
r_4
r_2
r_3
r_4
r_5
r_6
r_a
r_z

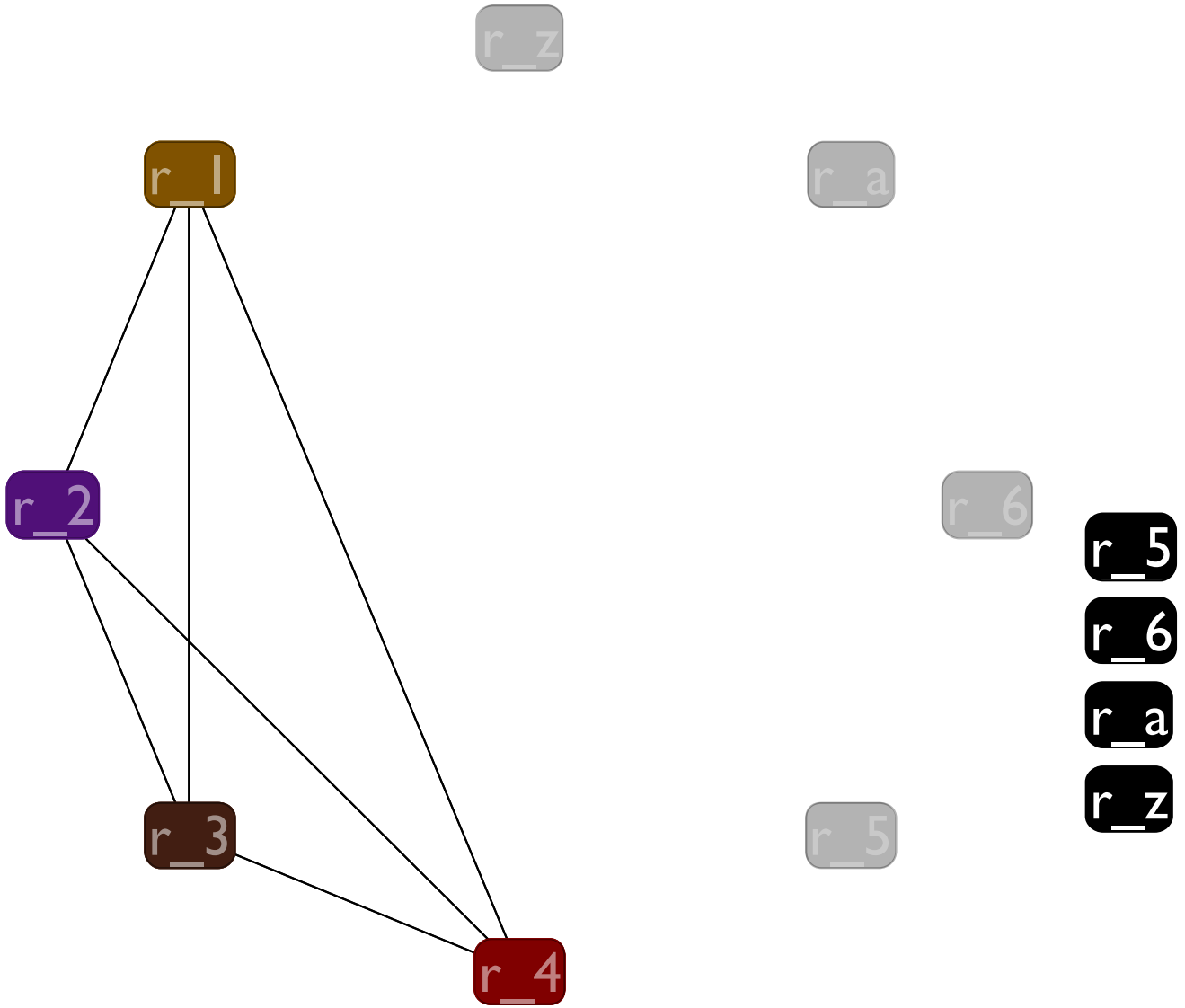

r_z
r_1
r_a
r_2
r_6
r_3
r_4
r_5
r_6
r_a
r_z
r_3
r_5
r_4

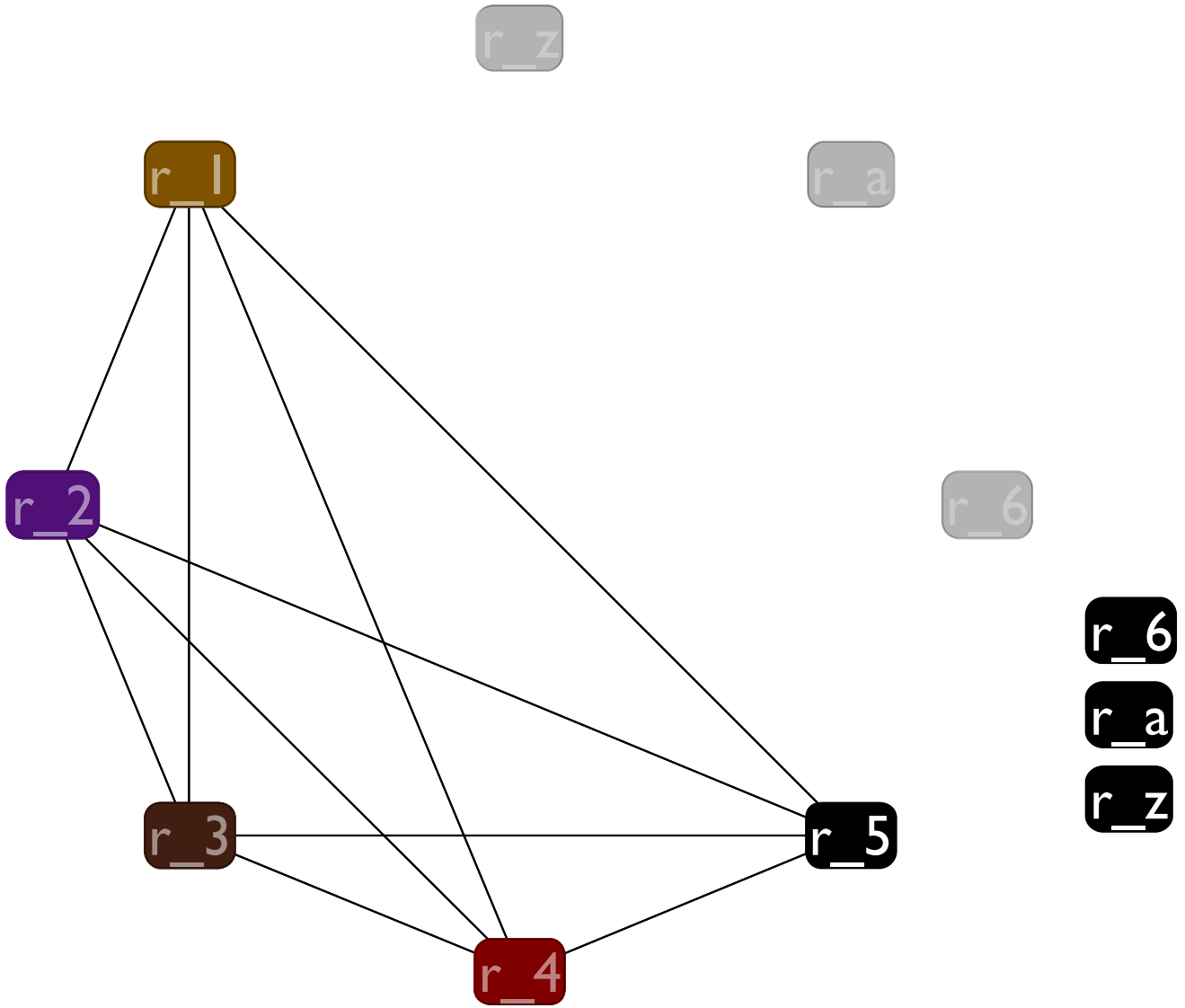

r_z
r_1
r_a
r_2
r_6
r_3
r_4
r_5
r_6
r_a
r_z
r_3
r_5
r_4

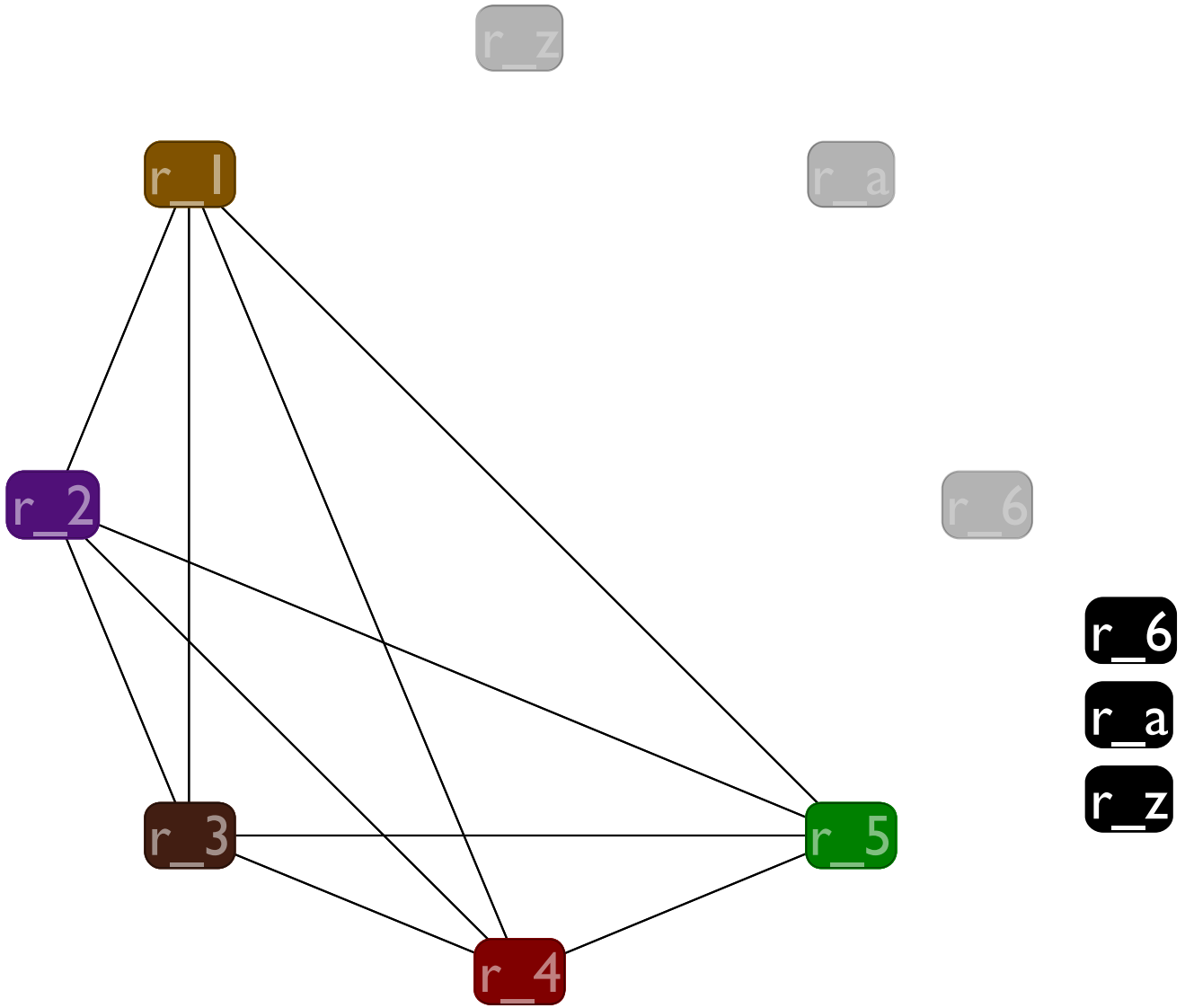

r_z
r_1
r_a
r_2
r_6
r_4
r_5
r_6
r_a
r_z
r_3
r_5
r_4

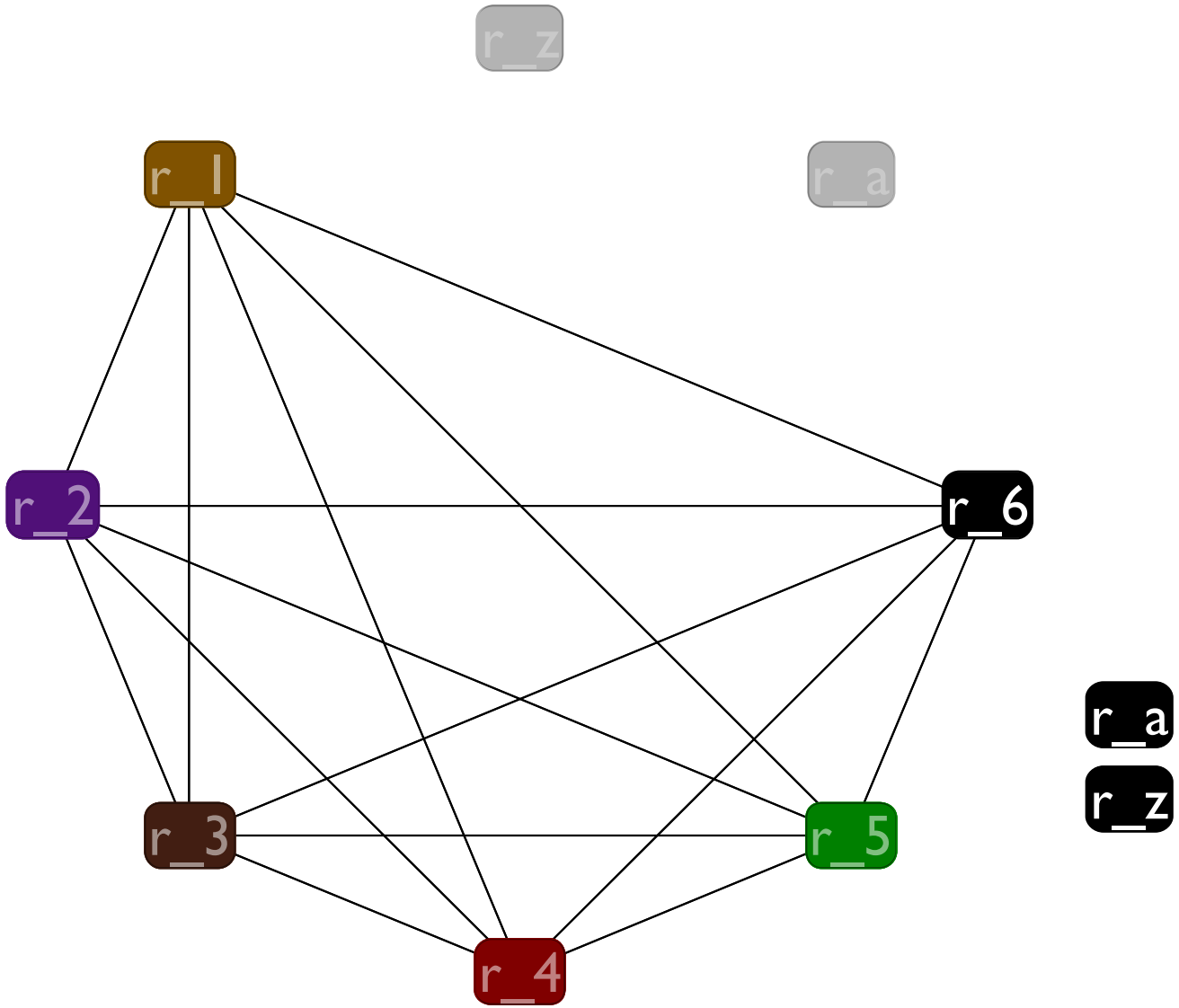

r_z
r_1
r_a
r_2
r_6
r_4
r_5
r_6
r_a
r_z
r_3
r_5
r_4

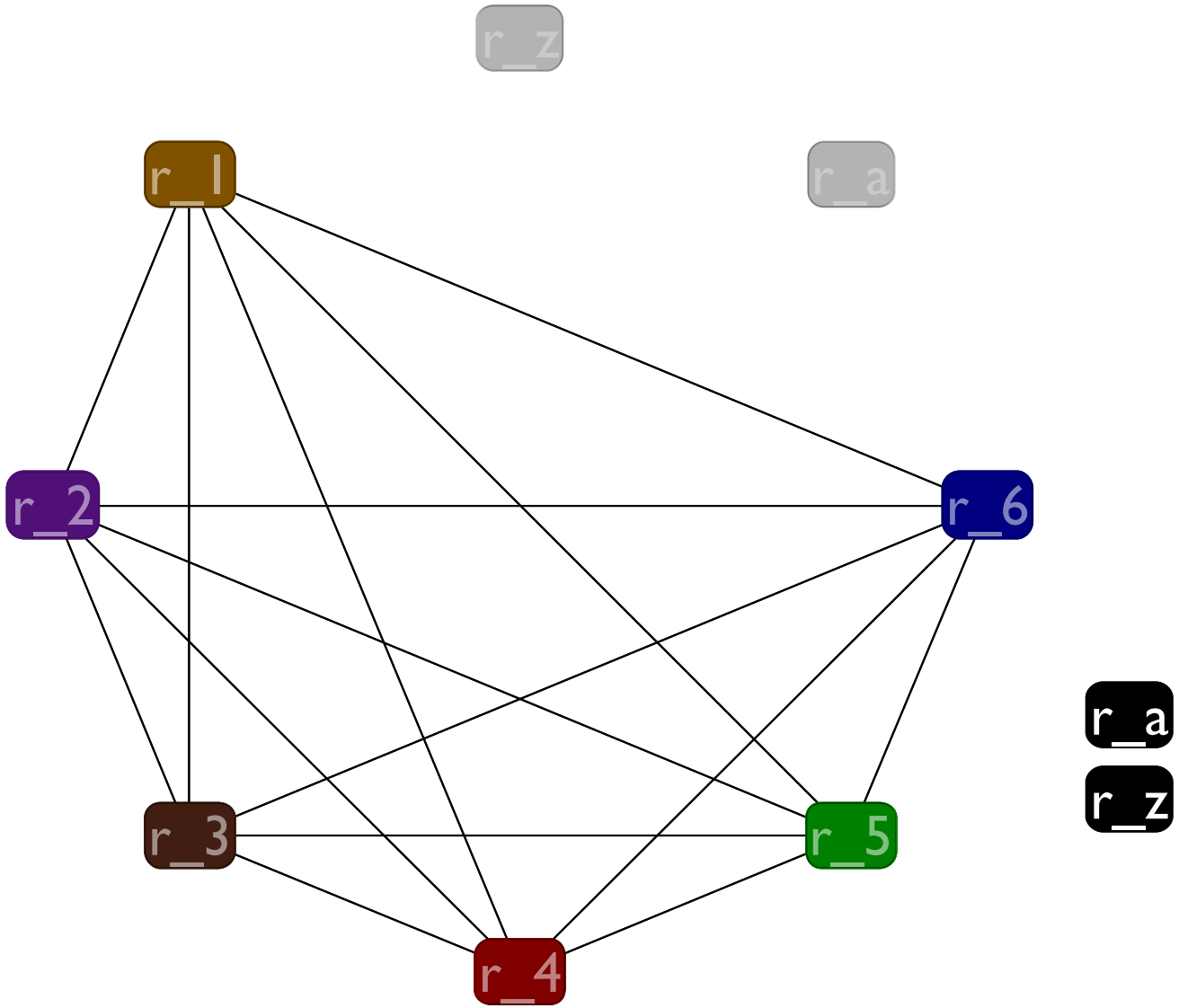

r_z
r_1
r_a
r_2
r_6
r_5
r_6
r_a
r_z
r_3
r_5
r_4

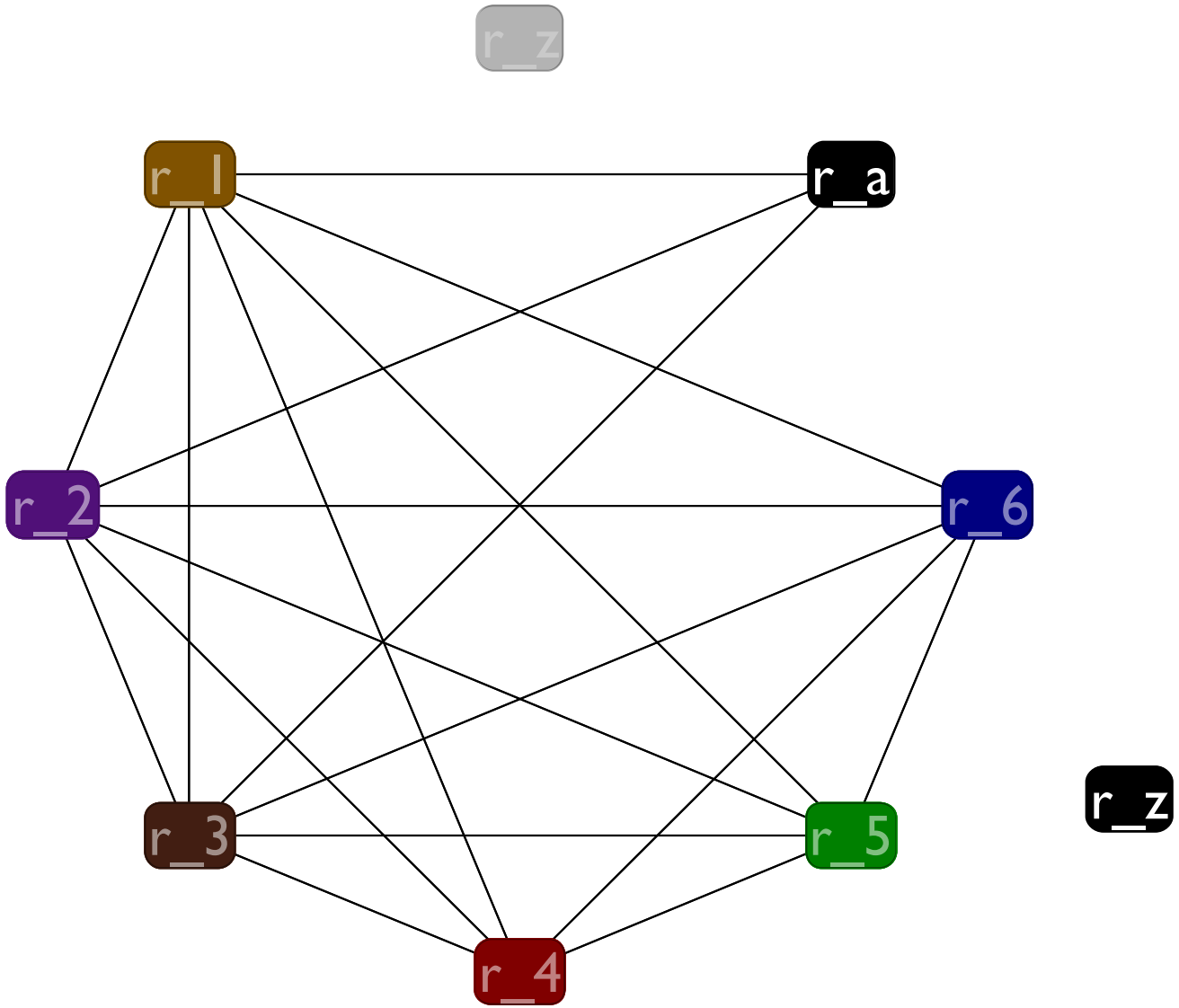

r_z
r_1
r_a
r_2
r_6
r_5
r_6
r_a
r_z
r_3
r_5
r_4

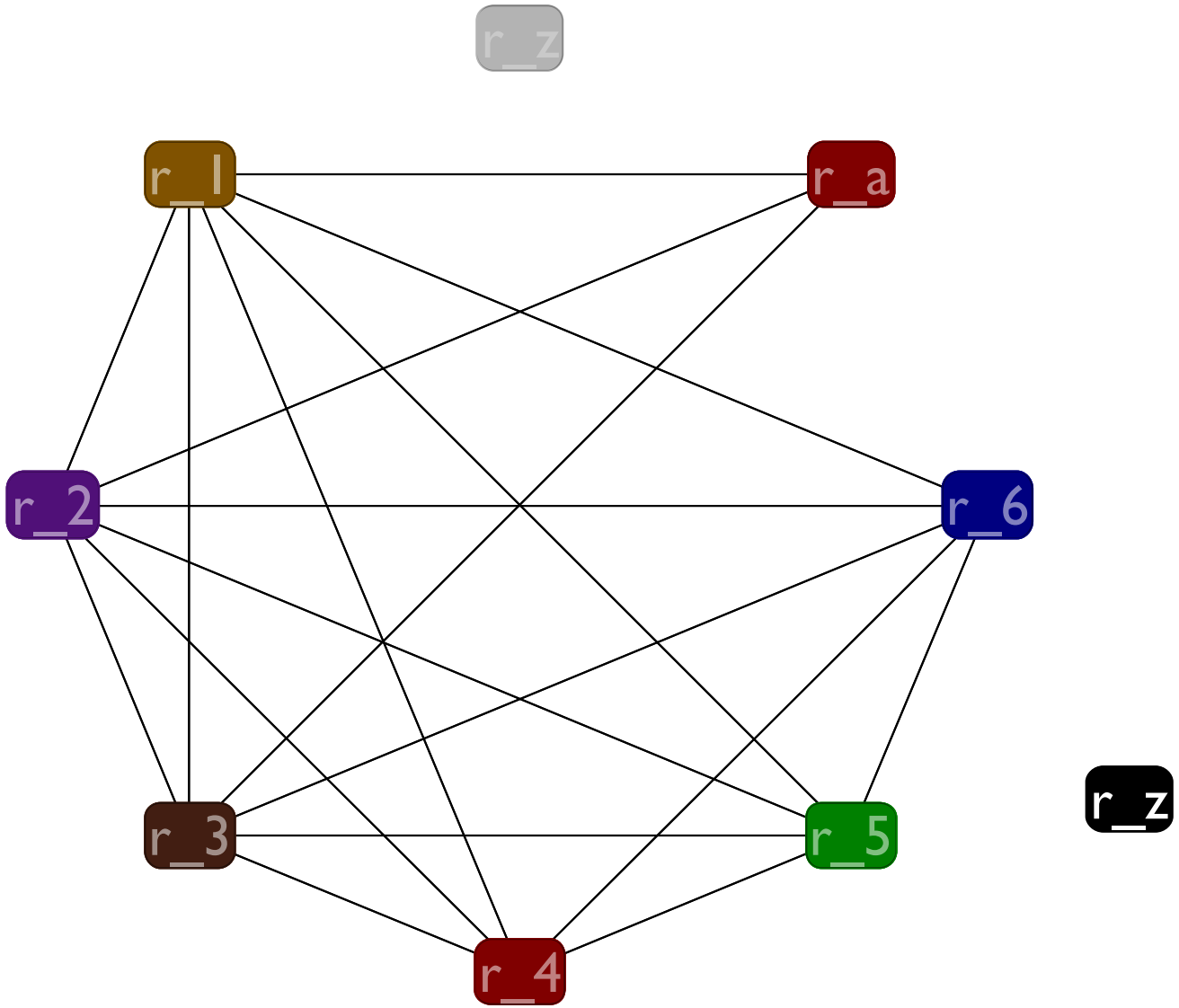

r_z
r_1
r_a
r_2
r_6
r_6
r_a
r_z
r_3
r_5
r_4

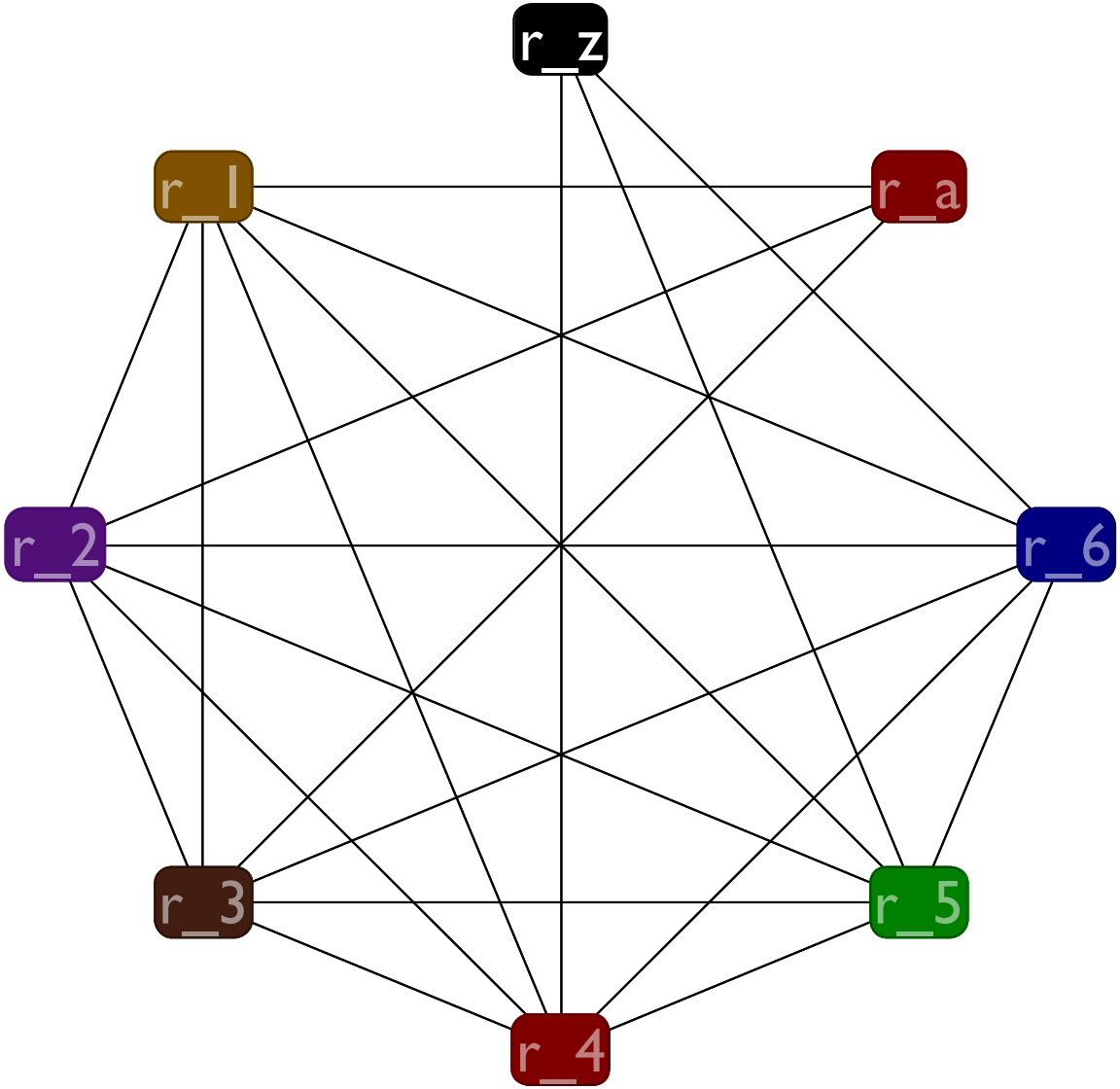

r_z
r_1
r_a
r_2
r_6
r_3
r_5
r_4
r_6
r_a
r_z

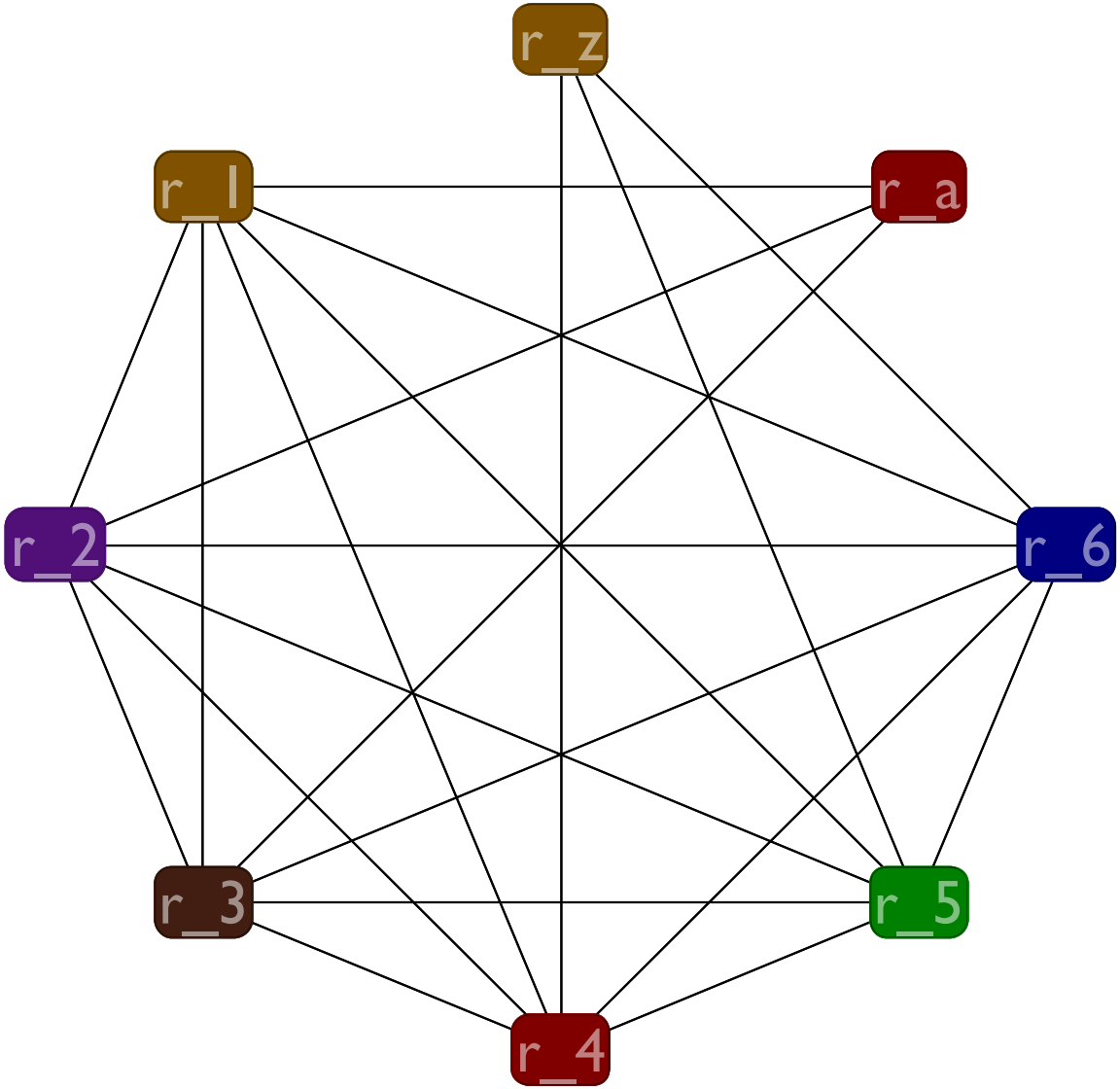

r_z
r_1
r_a
r_2
r_6
r_a
r_z
r_3
r_5
r_4


r_z
r_1
r_a
r_2
r_6
r_3
r_5
r_4
r_a
r_z


r_z
r_1
r_a
r_2
r_6
r_3
r_5
r_4
r_z

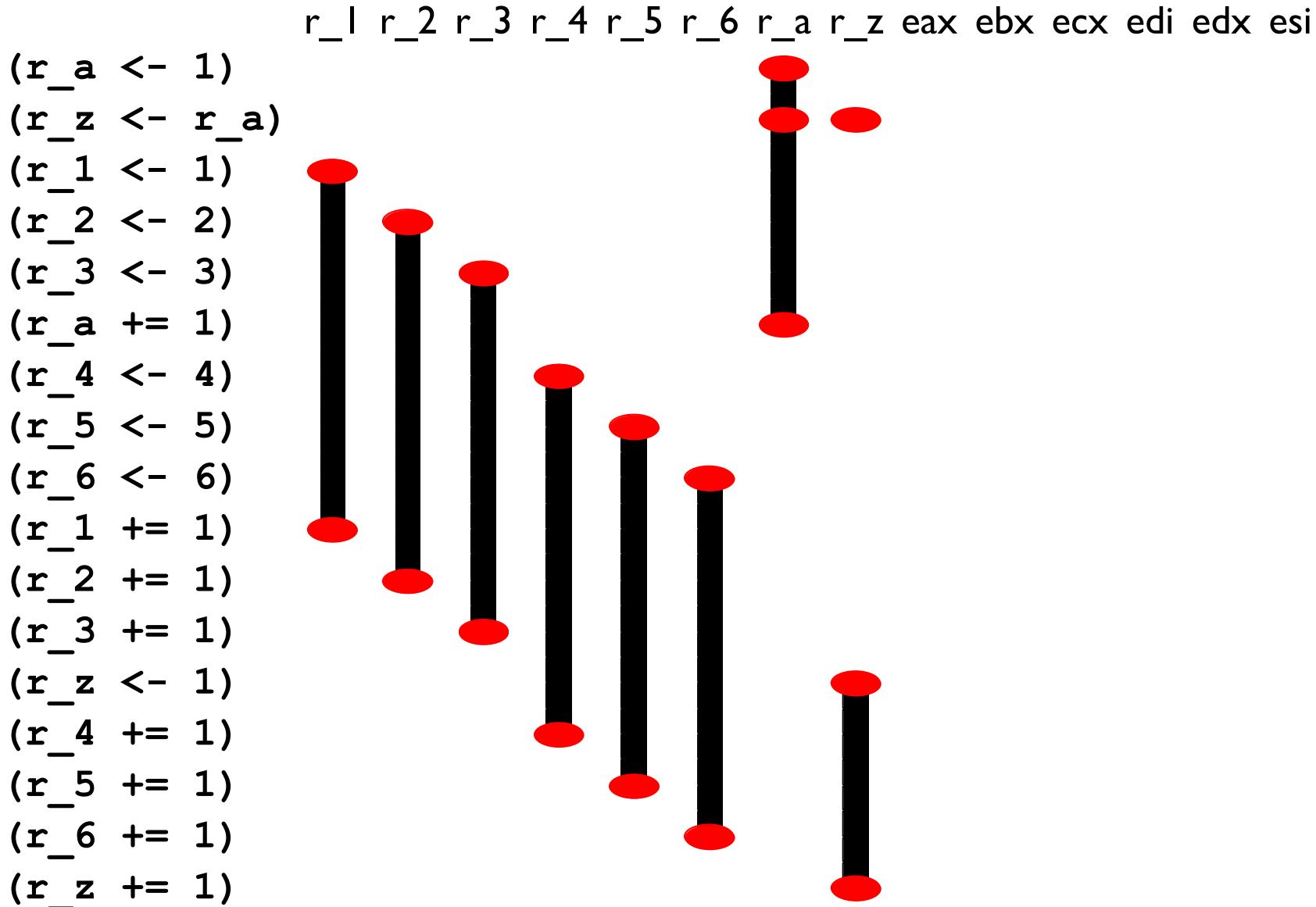

r_z
r_1
r_a
r_2
r_6
r_3
r_5
r_4
r_z


r_z
r_1
r_a
r_2
r_6
r_3
r_5
r_4

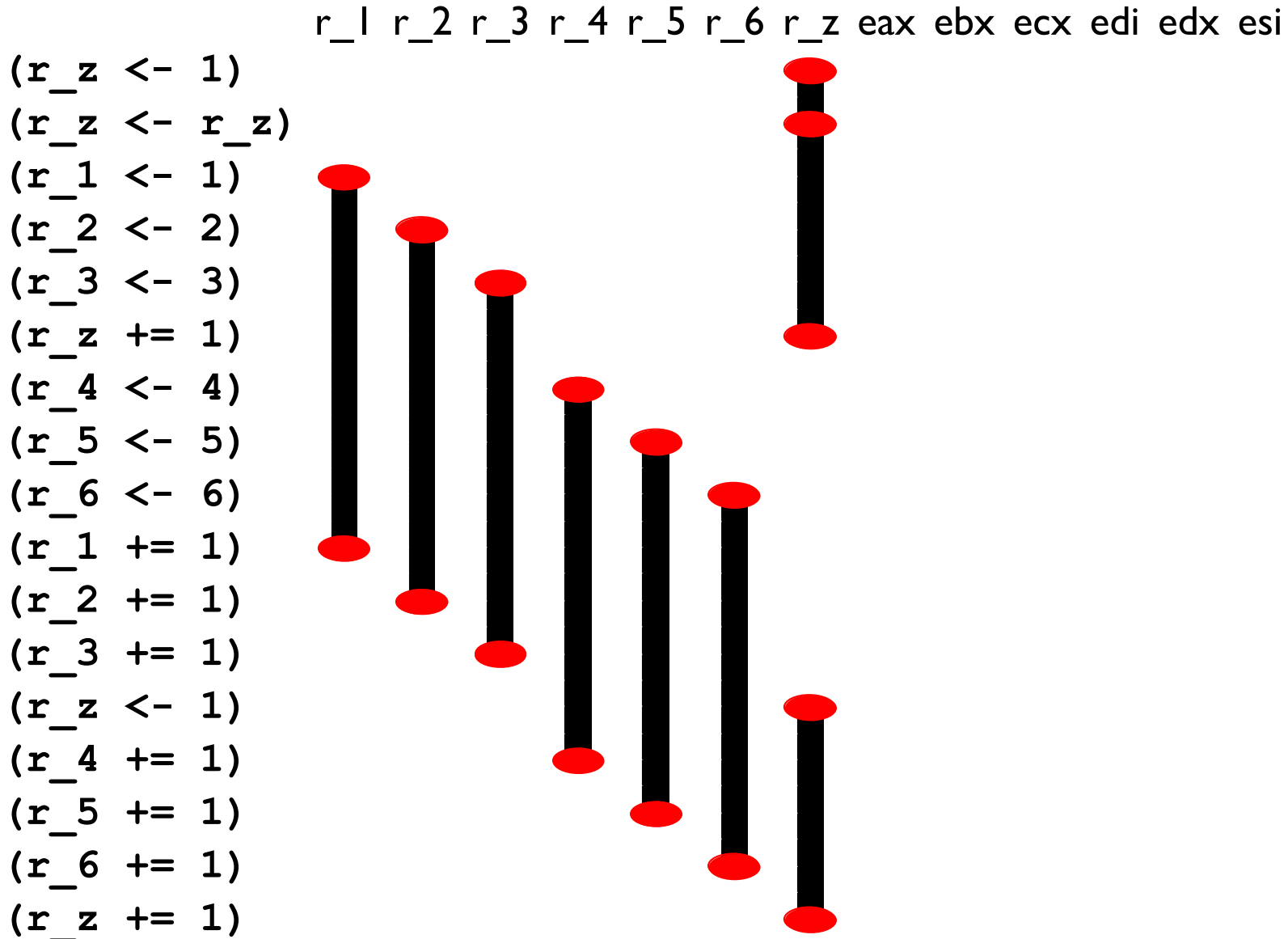

r_z
r_1
r_a
r_2
r_6
r_3
r_5
r_4

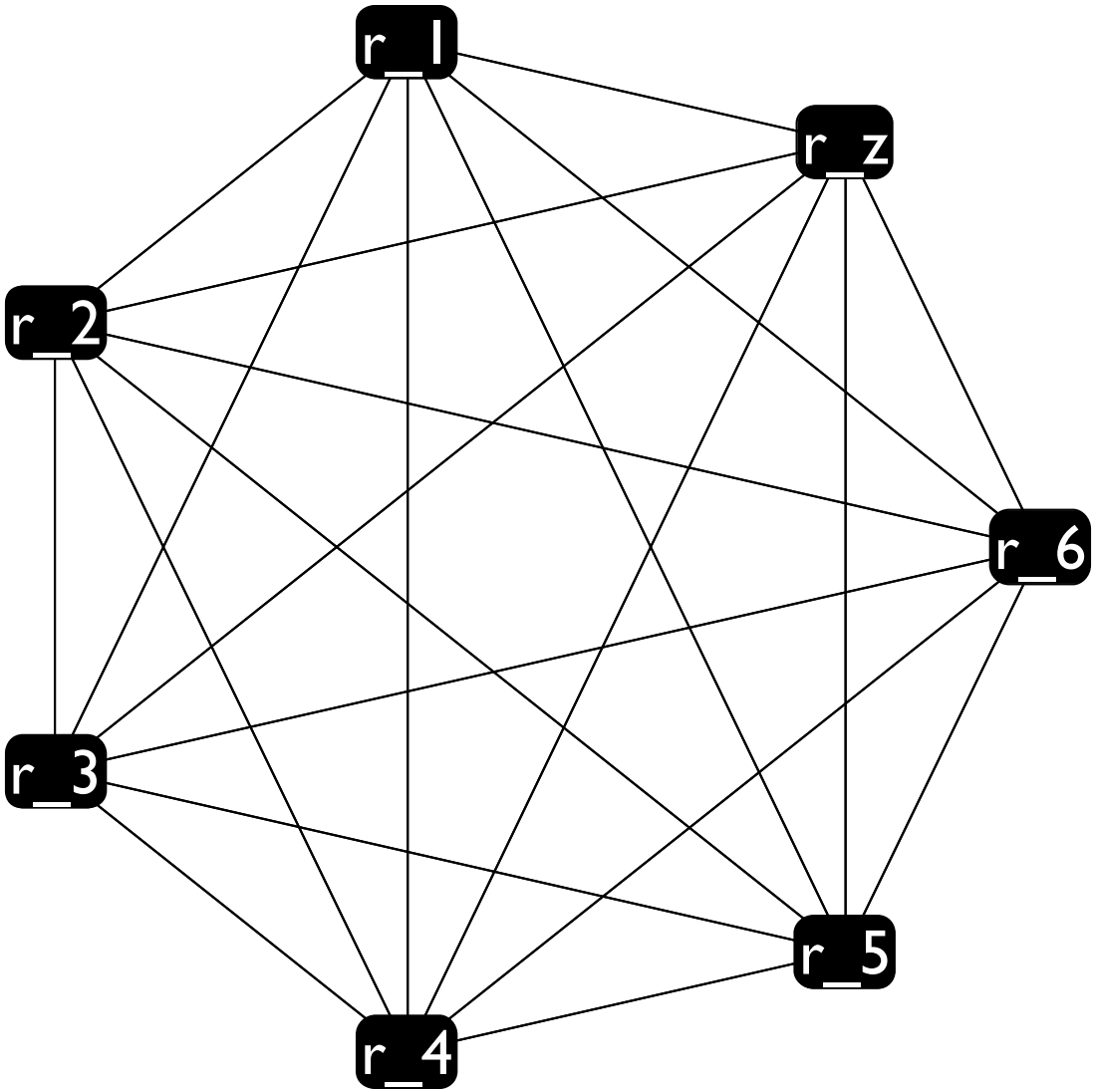
Now, we coalesce $\mathtt{r}_\mathtt{a}$ and $\mathtt{r}_\mathtt{z}$

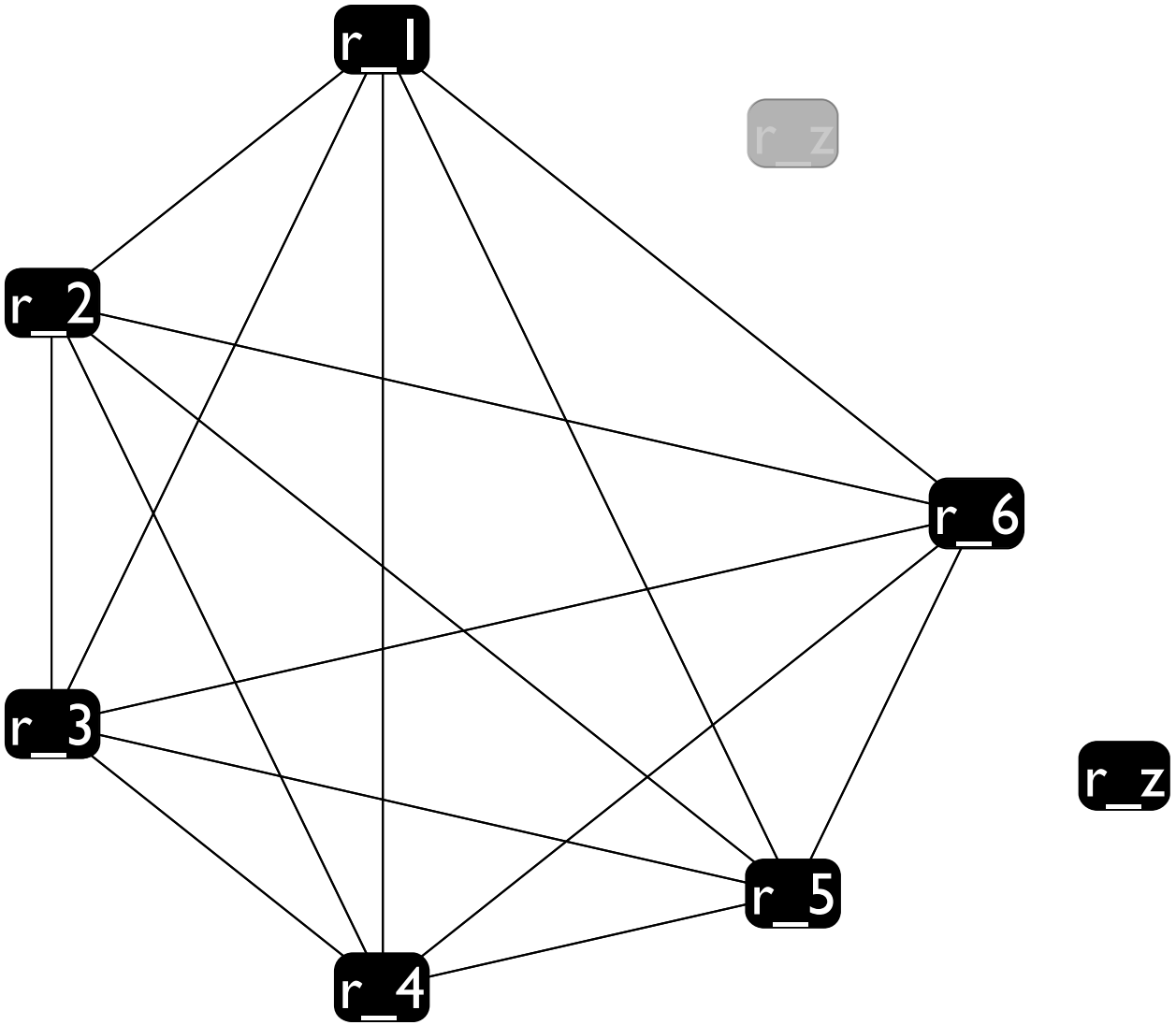
Coalescing Problem

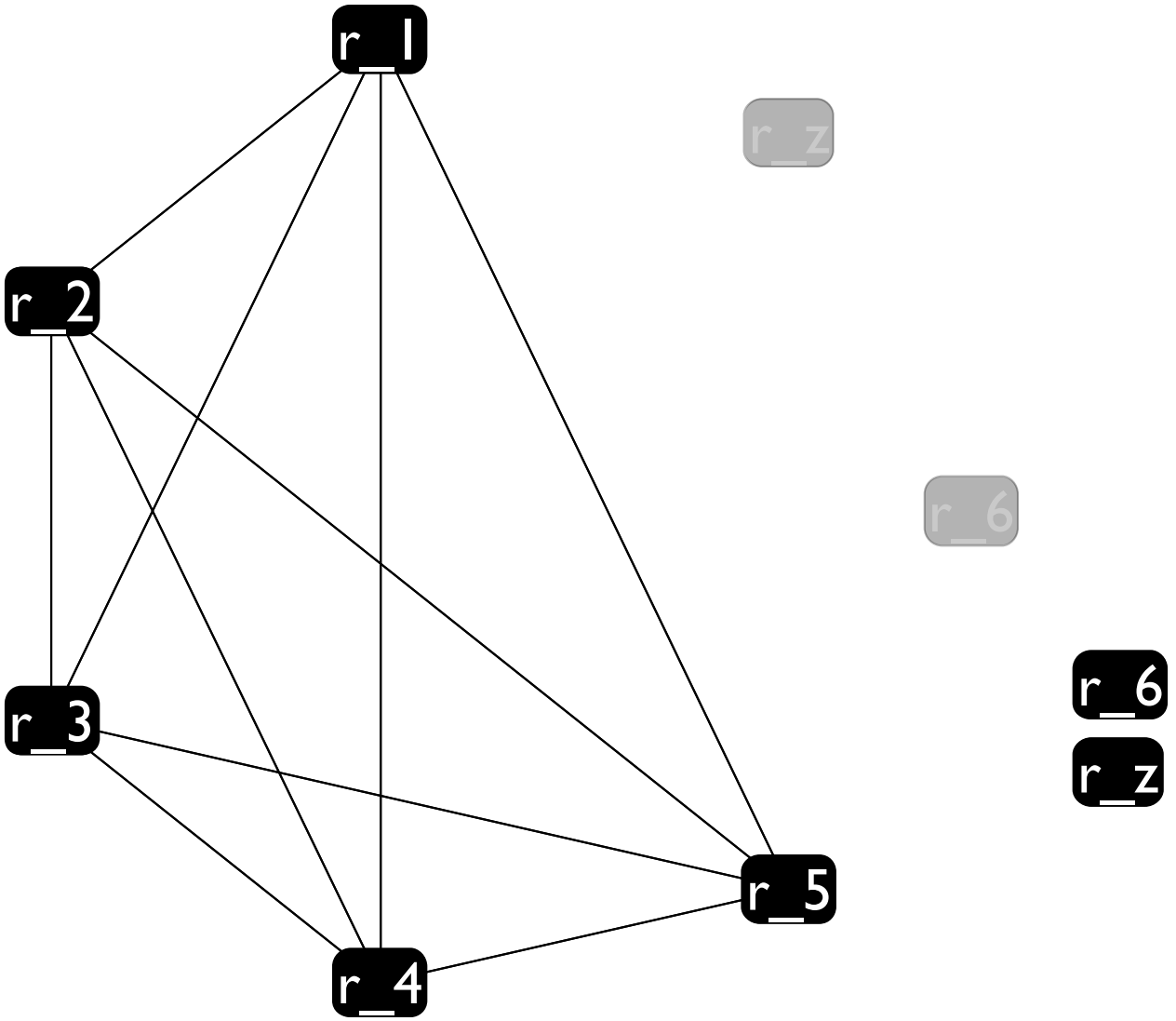


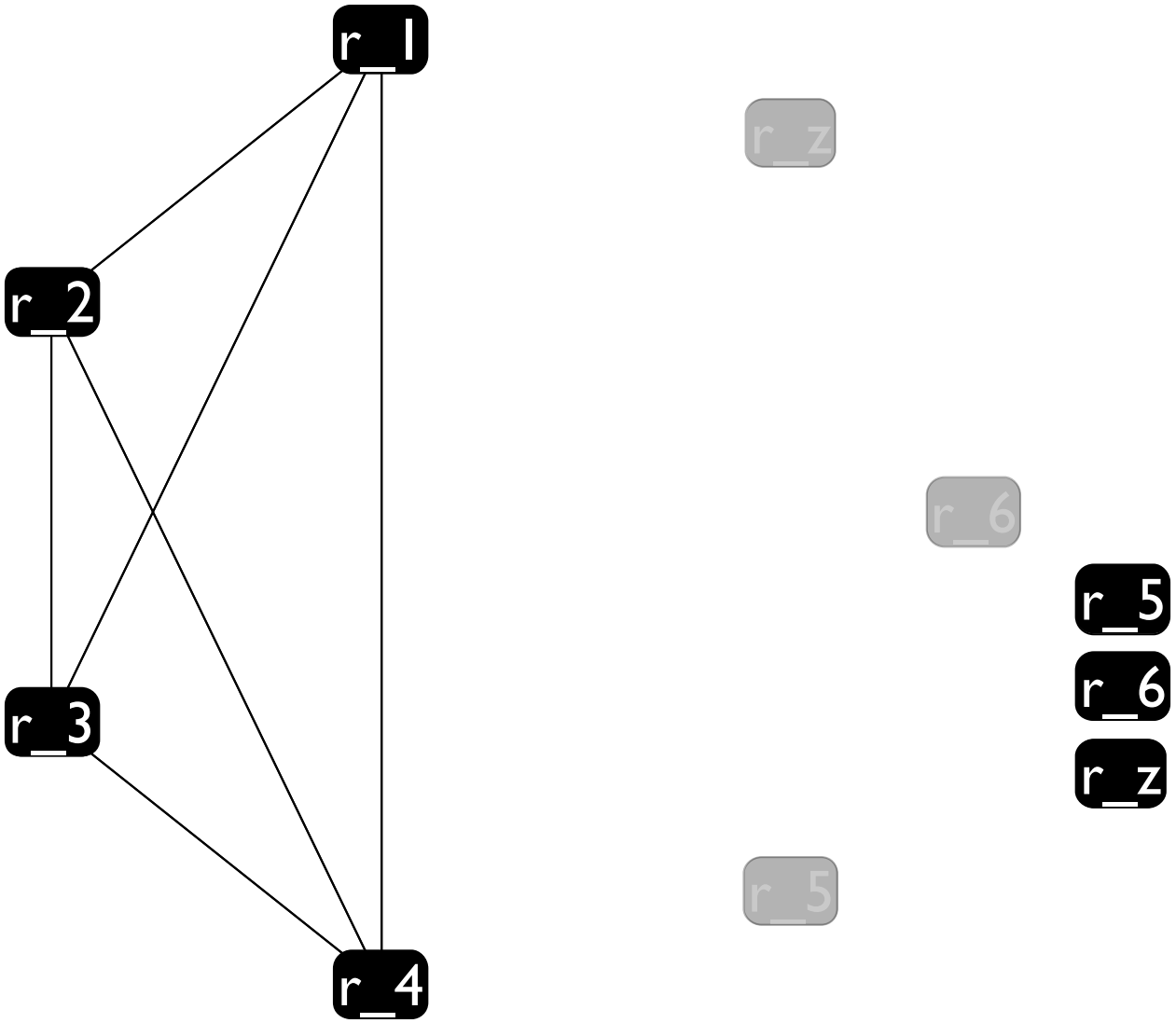
Coalescing Problem

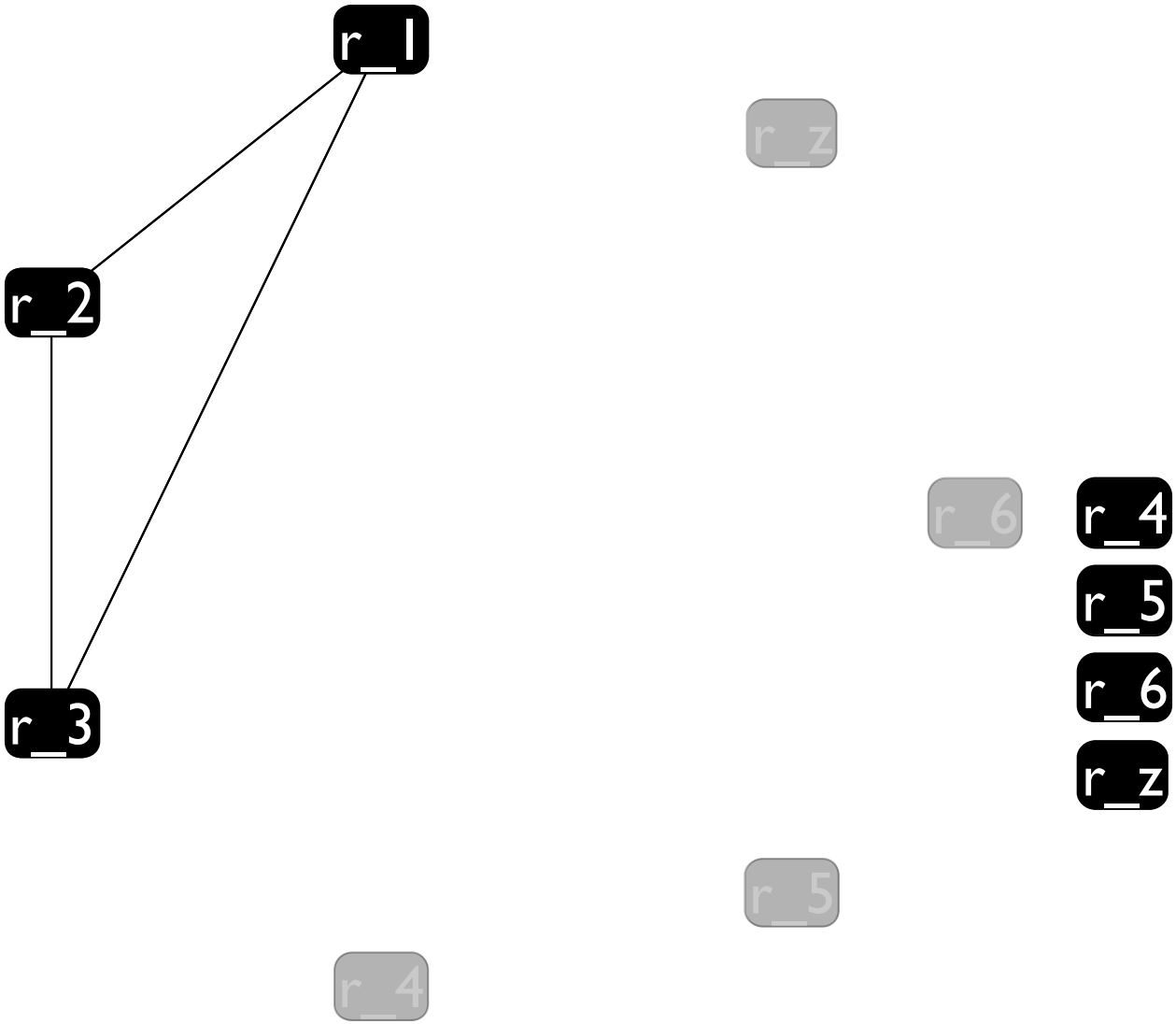


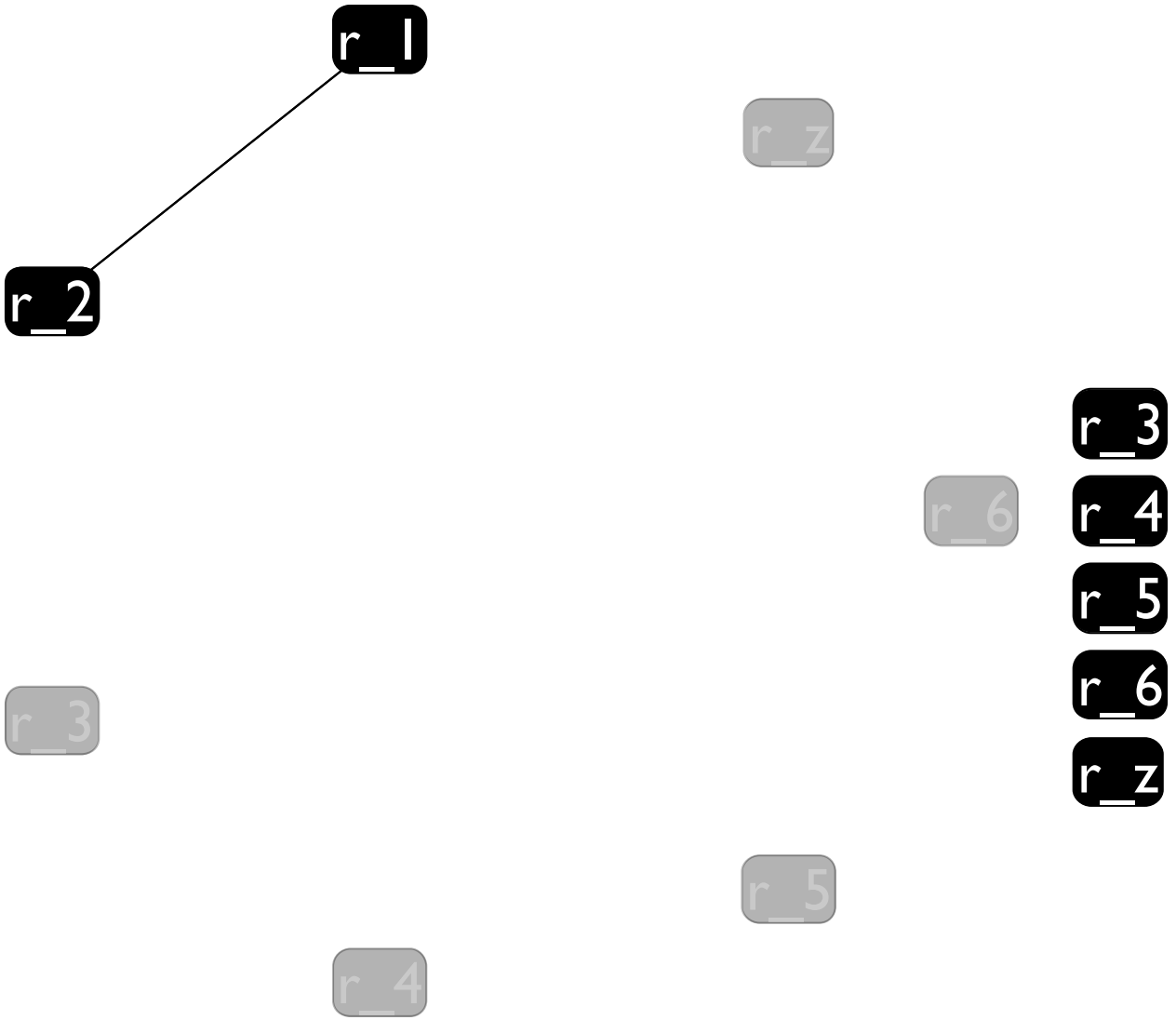

r_1
r_z
r_2
r_6
r_3
r_5
r_4

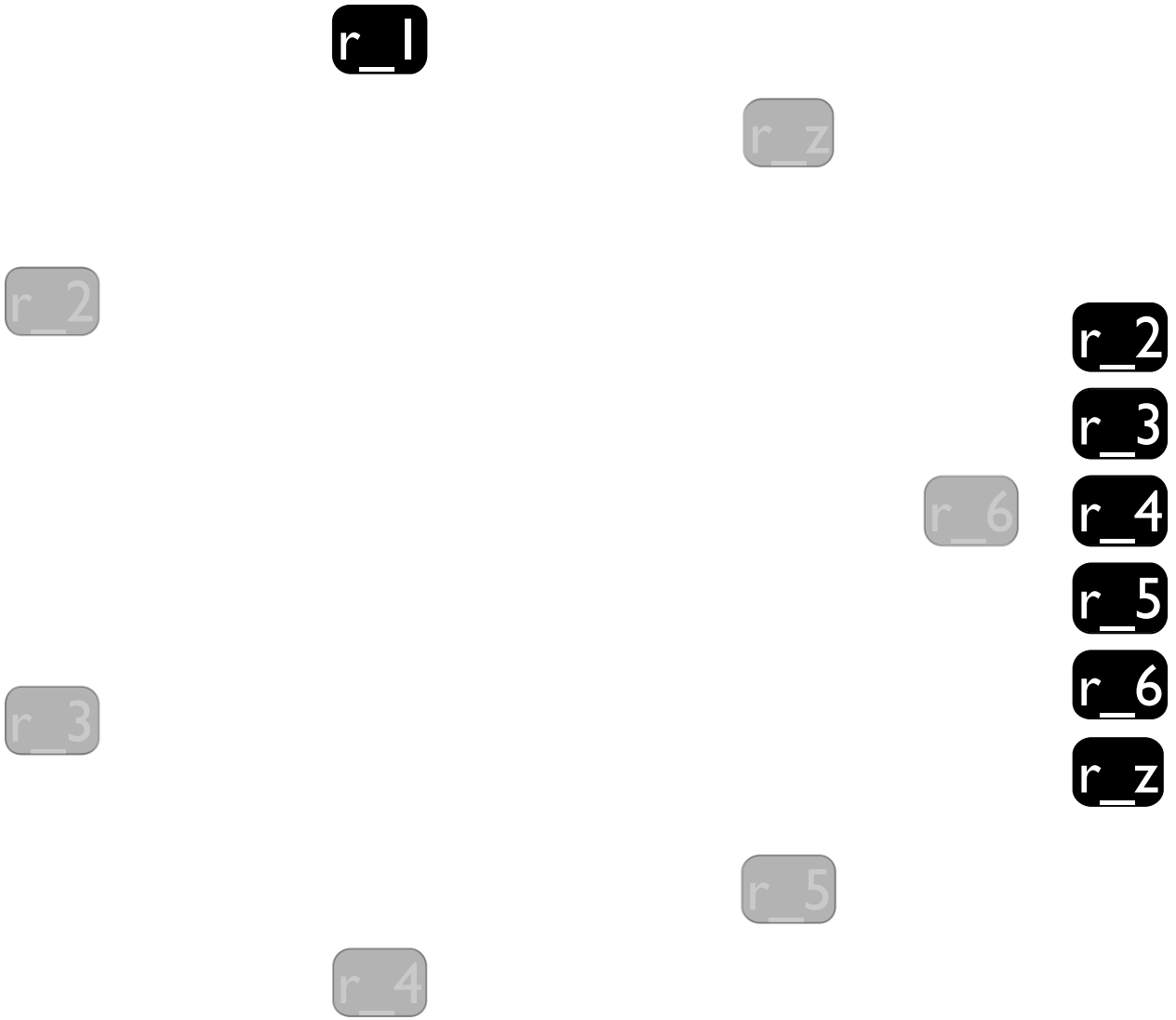

r_1
r_z
r_2
r_6
r_3
r_z
r_5
r_4

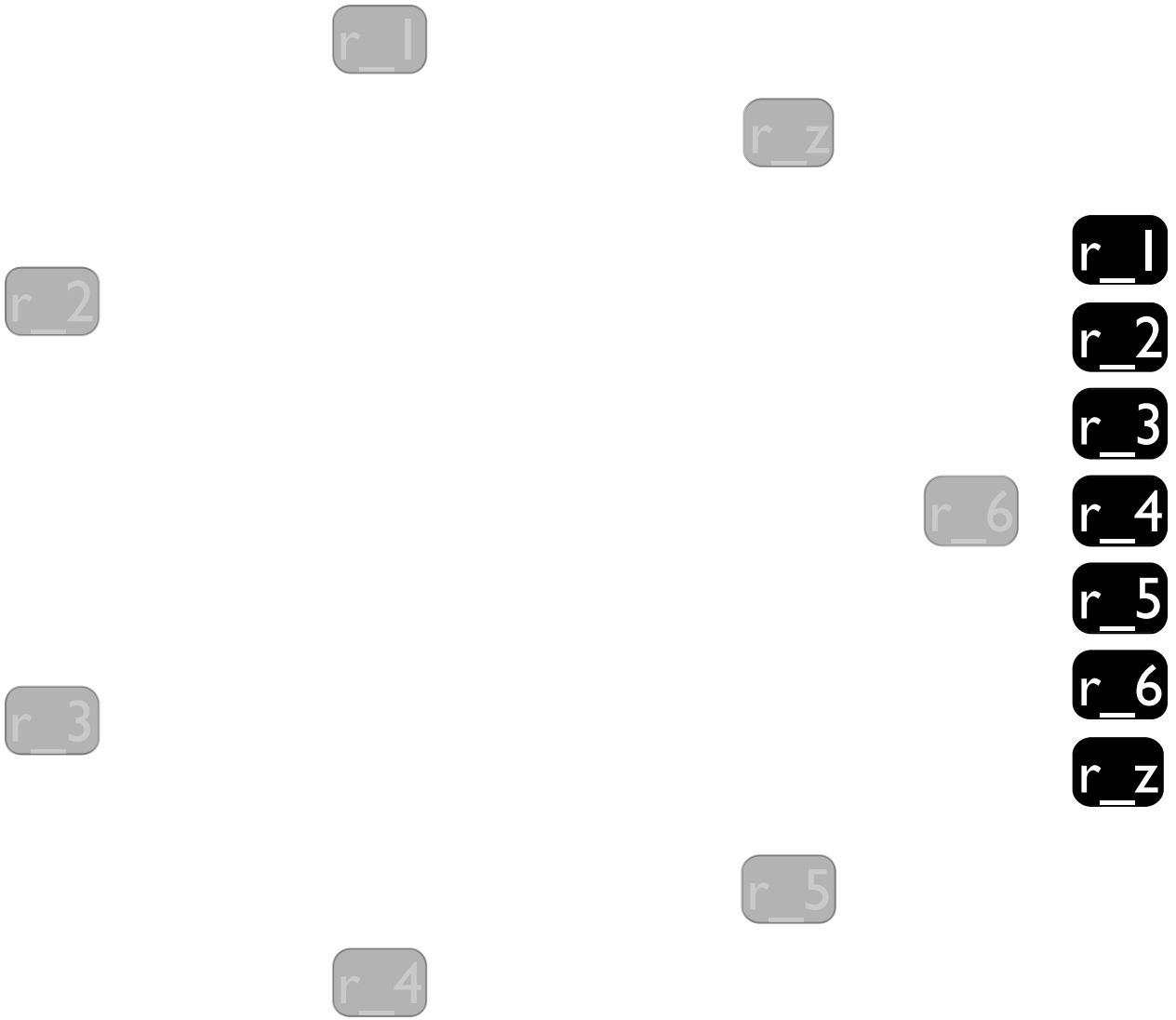

r_1
r_z
r_2
r_6
r_6
r_3
r_z
r_5
r_4

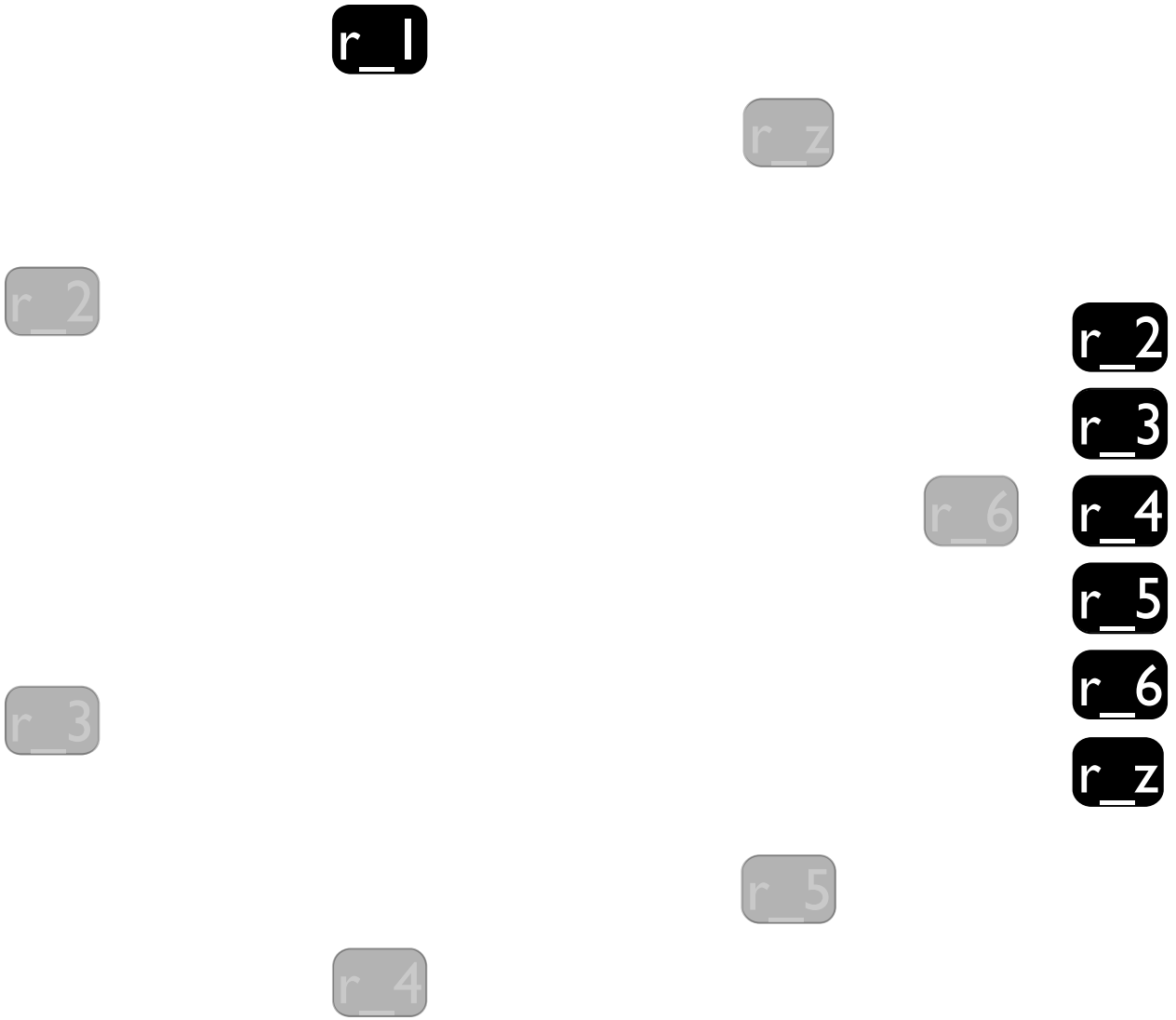

r_1
r_2
r_3
r_4
r_z
r_6
r_5
r_5
r_6
r_z

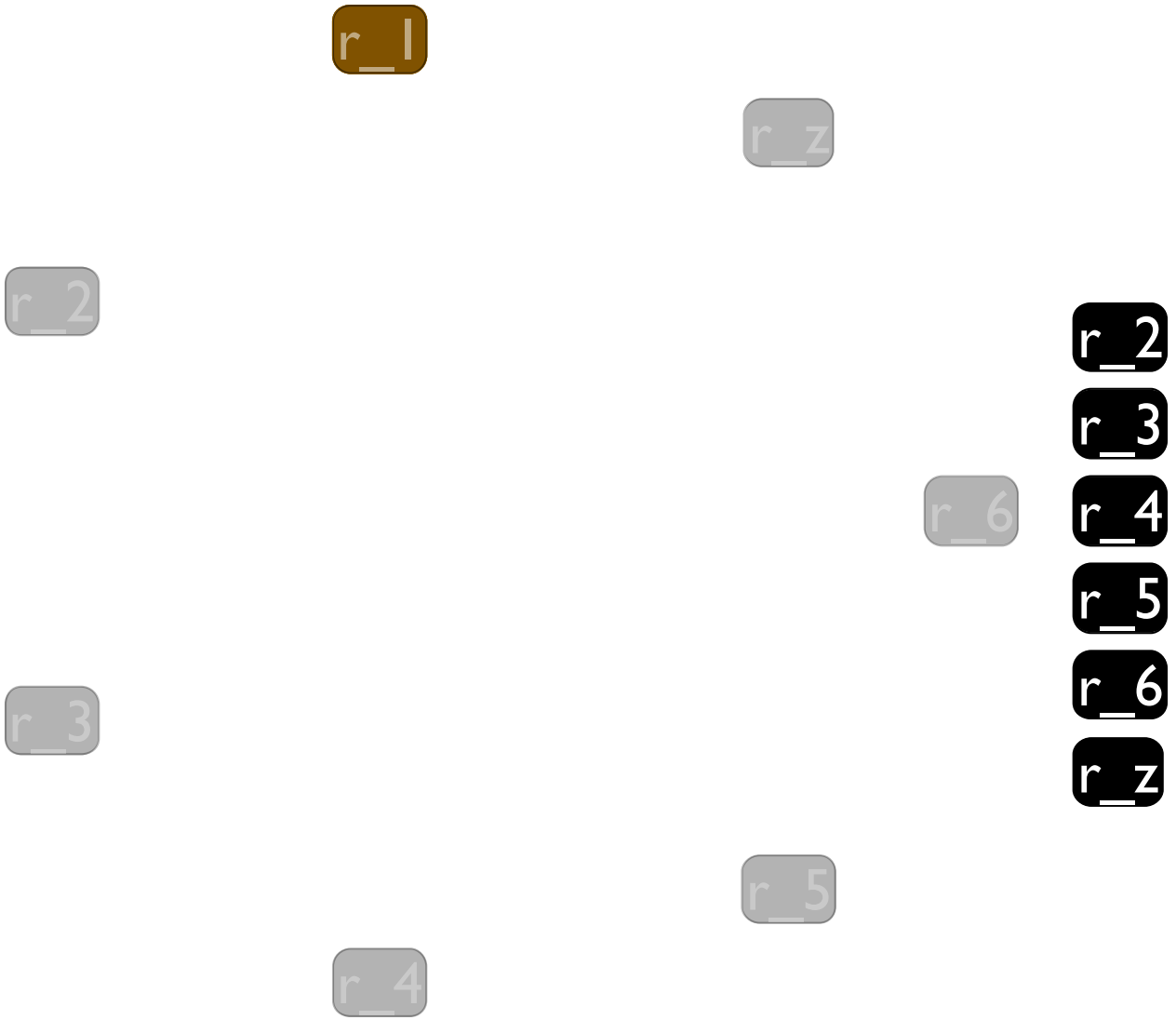

r_1
r_2
r_3
r_z
r_6
r_4
r_5
r_6
r_z
r_5
r_4

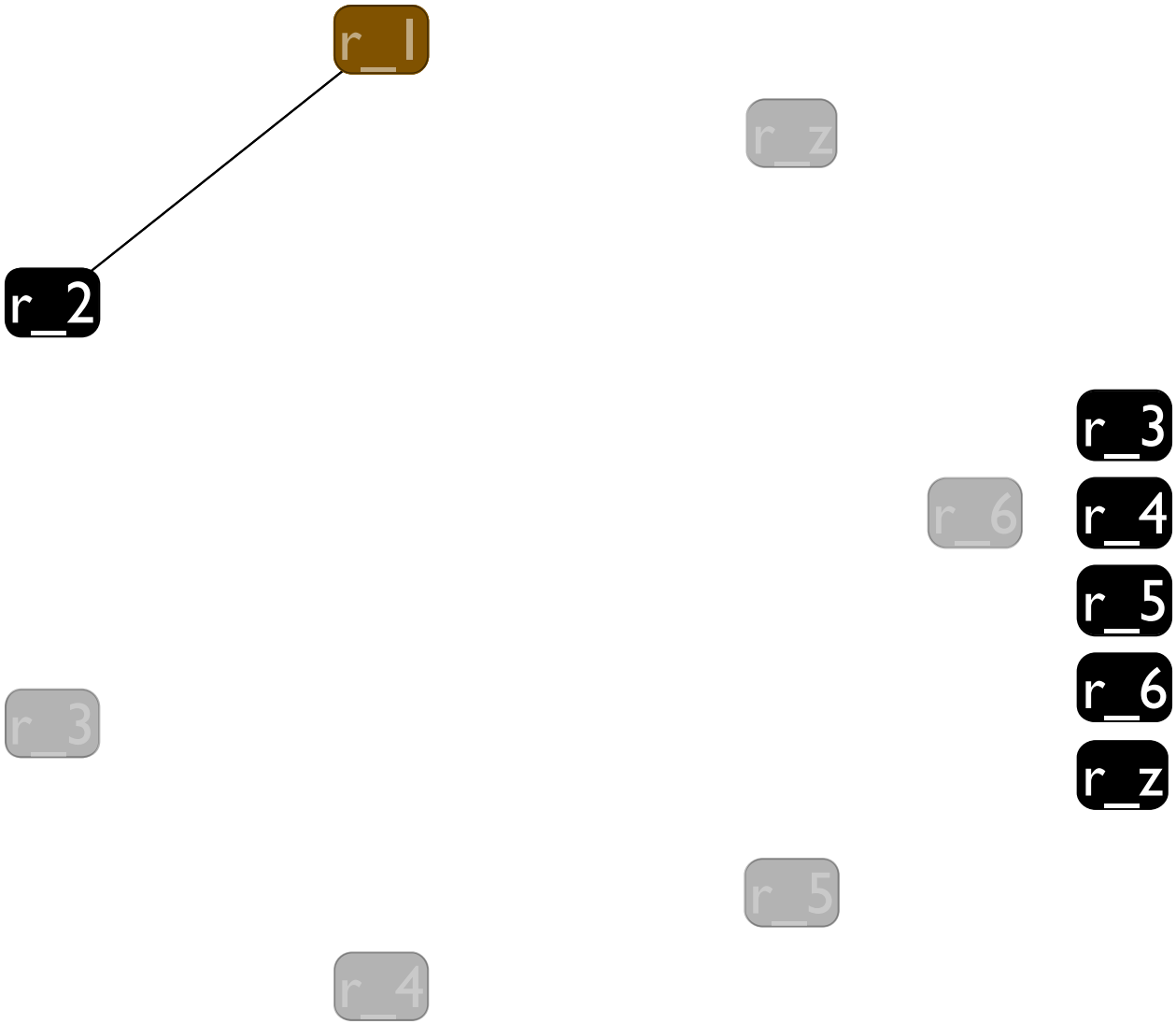

r_1
r_2
r_z
r_3
r_6
r_4
r_5
r_6
r_z
r_3
r_5
r_4

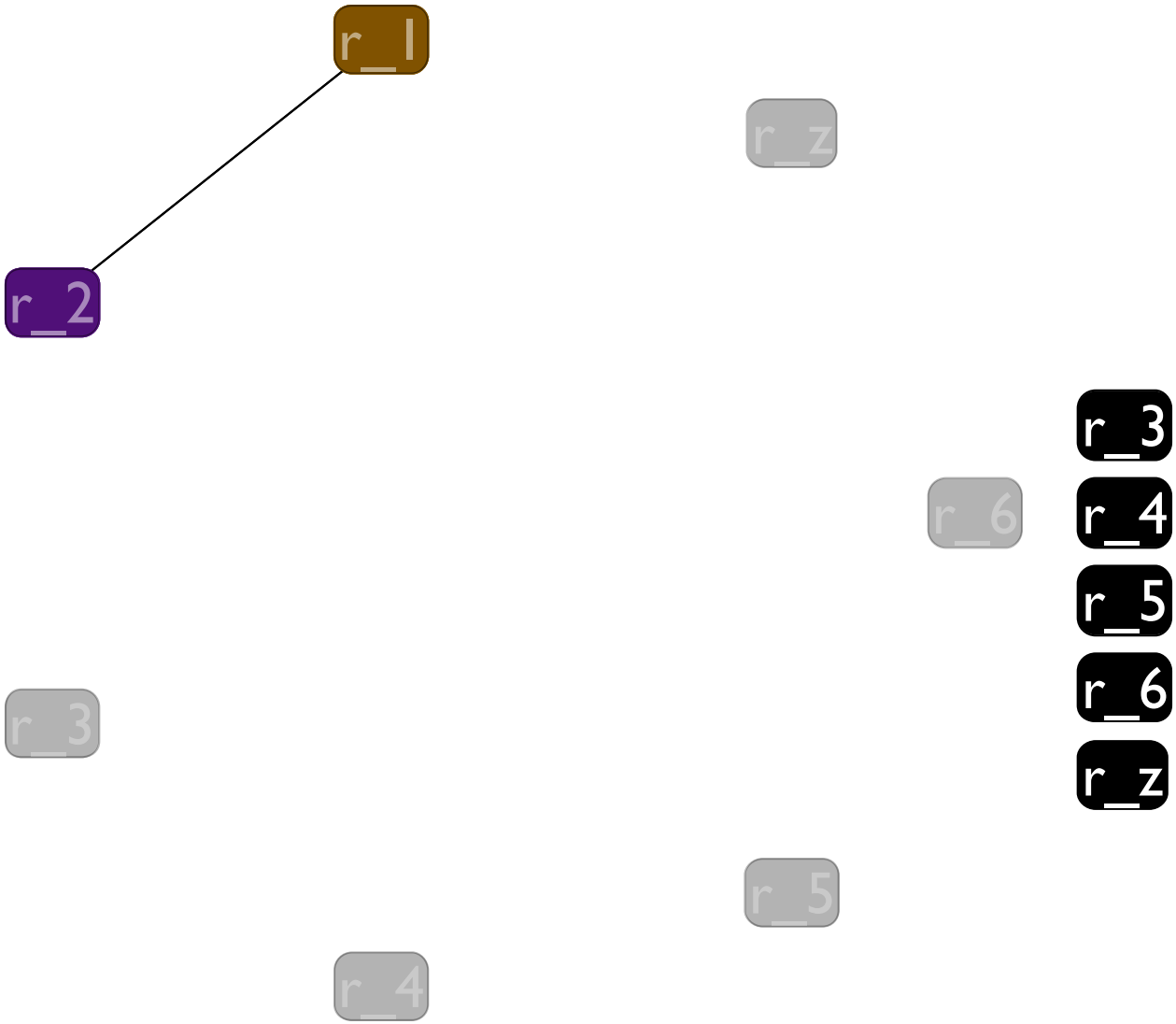

r_1
r_z
r_2
r_2
r_3
r_6
r_4
r_5
r_6
r_3
r_z
r_5
r_4

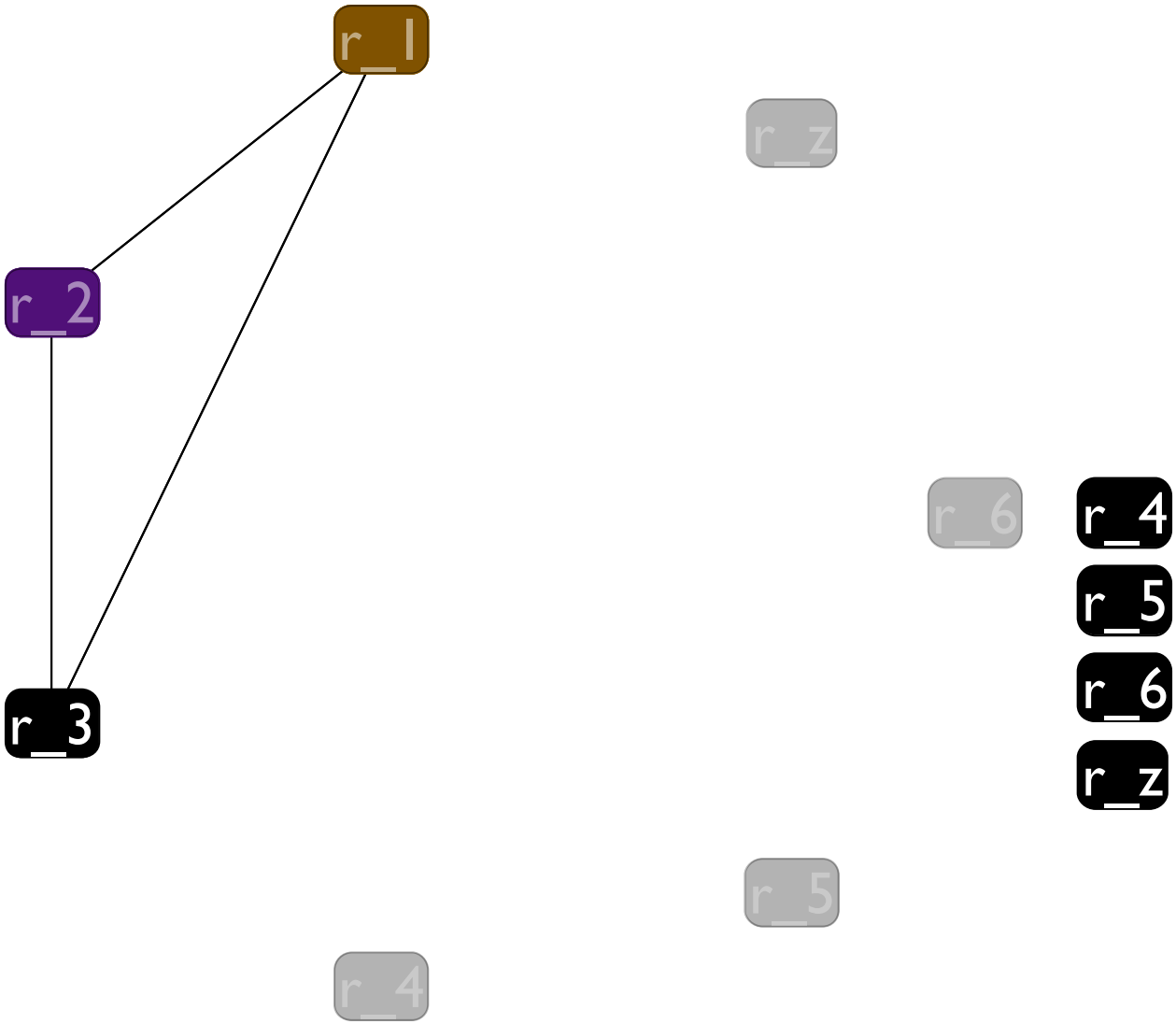

r_1
r_z
r_2
r_6
r_3
r_5
r_4
r_1
r_2
r_3
r_4
r_5
r_6
r_z

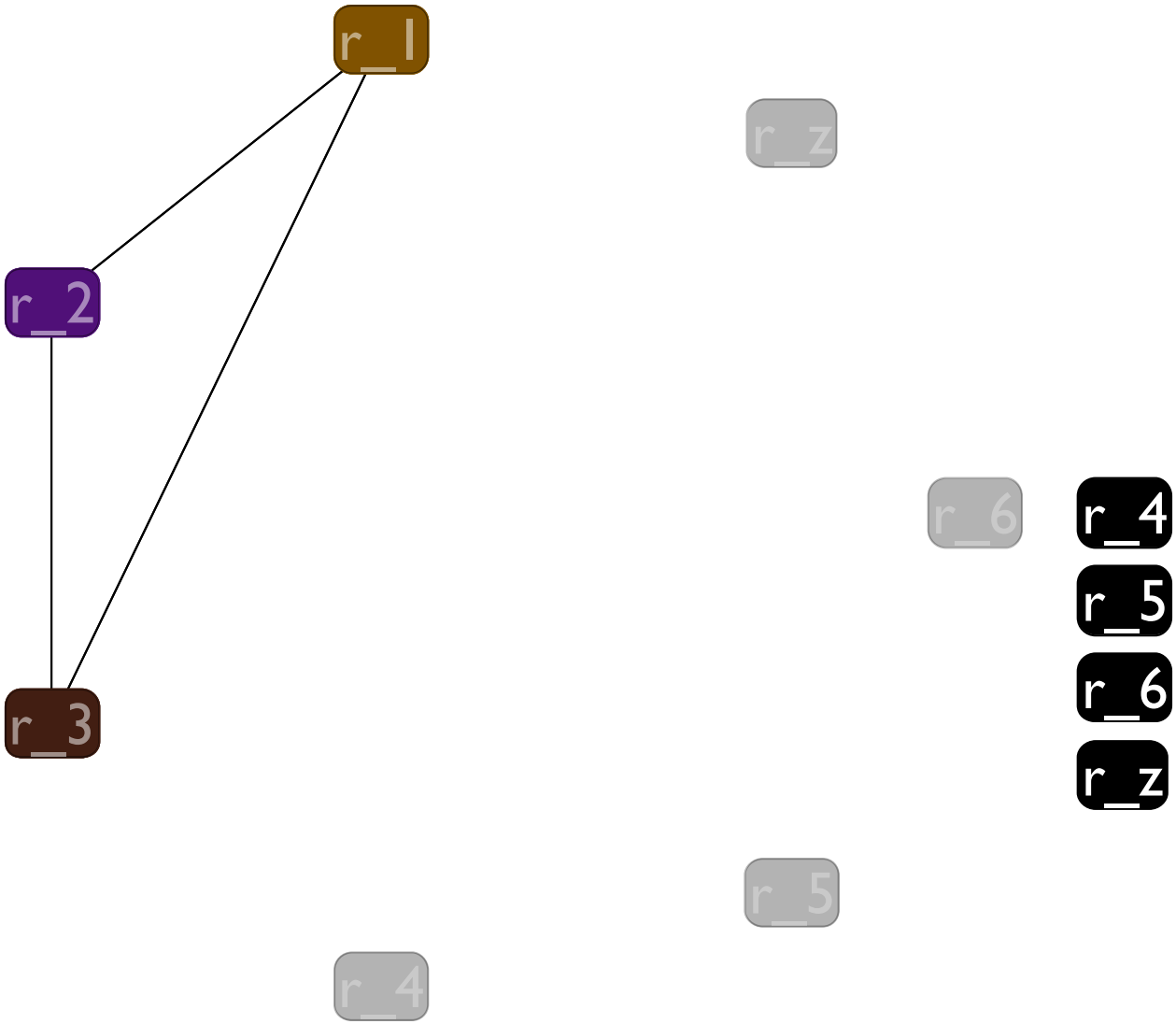

r_1
r_z
r_2
r_2
r_3
r_6
r_4
r_5
r_6
r_3
r_z
r_5
r_4

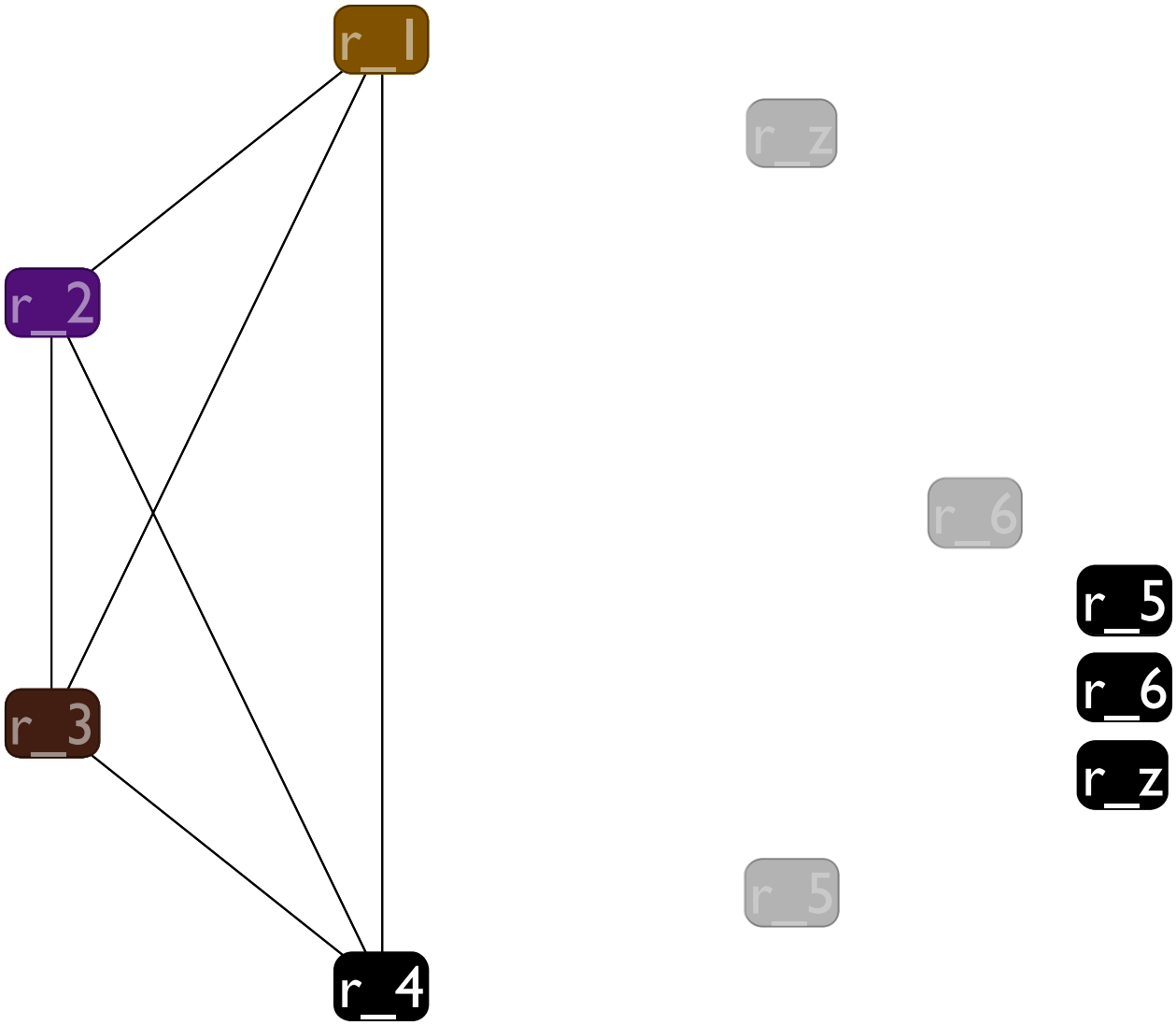

r_1
r_z
r_2
r_6
r_3
r_5
r_4
r_2
r_3
r_4
r_5
r_6
r_z

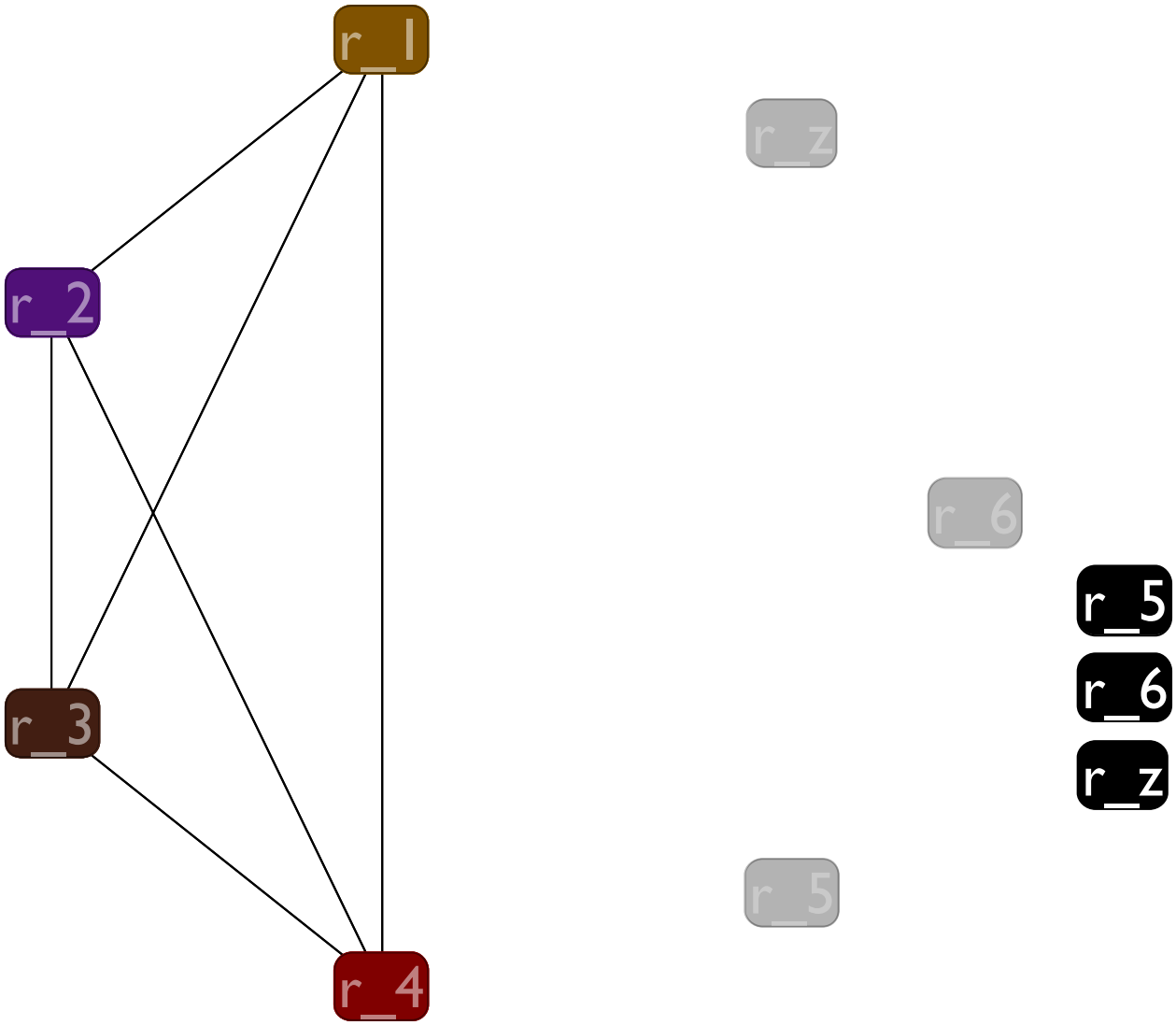

r_1
r_z
r_2
r_3
r_6
r_4
r_5
r_6
r_z
r_3
r_5
r_4

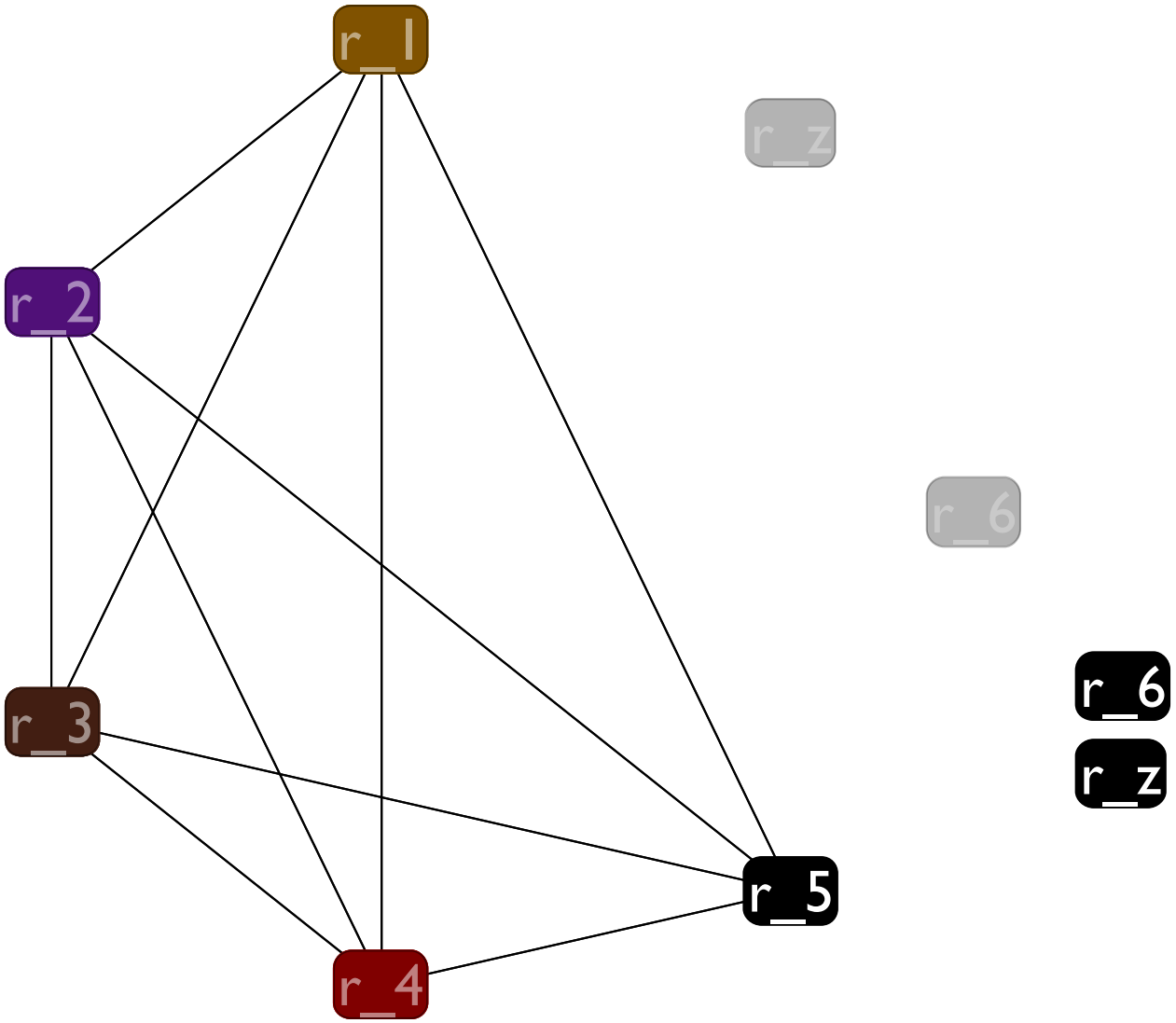

r_1
r_2
r_3
r_4
r_5
r_6
r_z
r_3
r_4
r_5
r_6
r_z

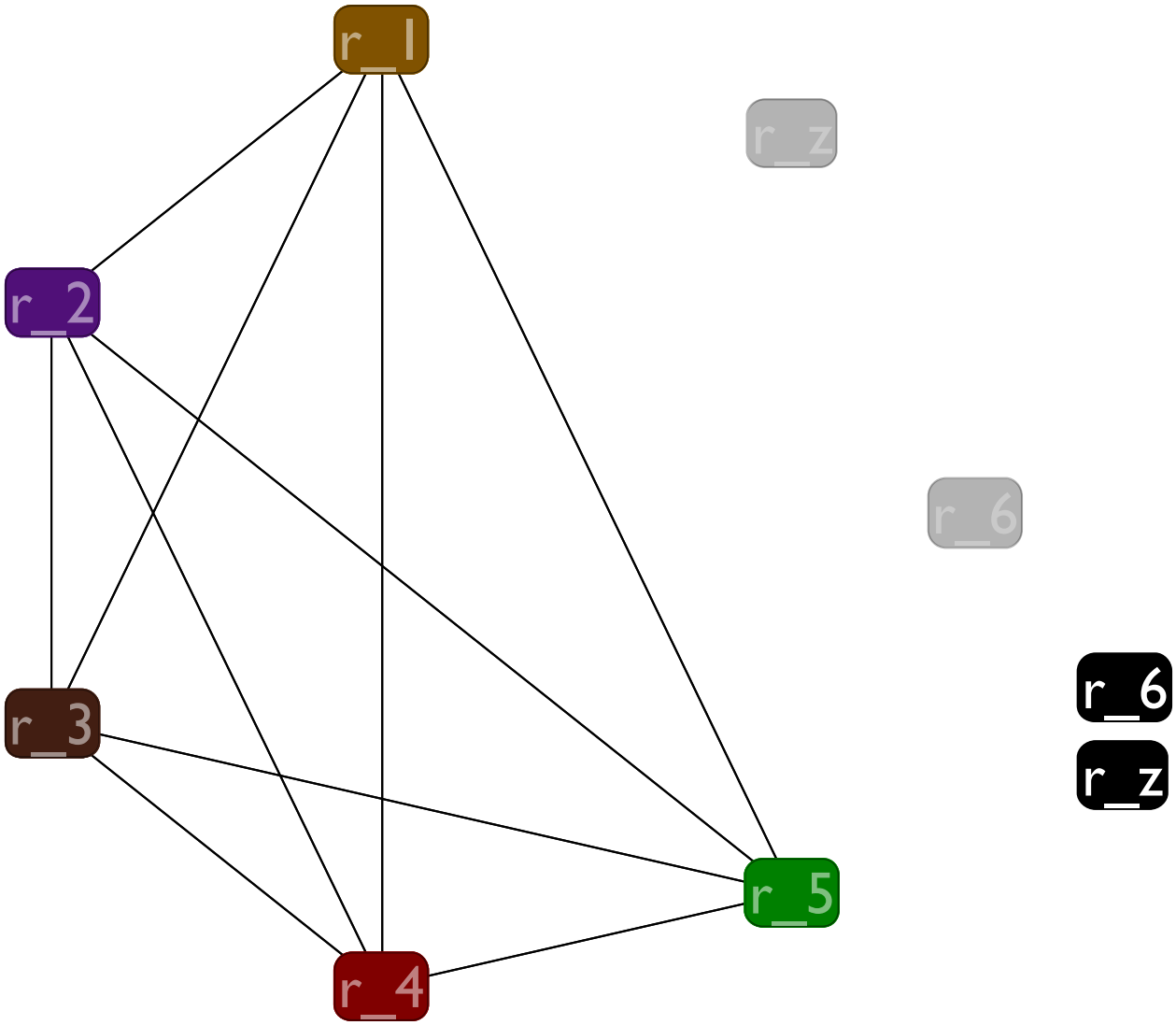

r_1
r_2
r_3
r_z
r_6
r_4
r_5
r_6
r_z
r_5
r_4

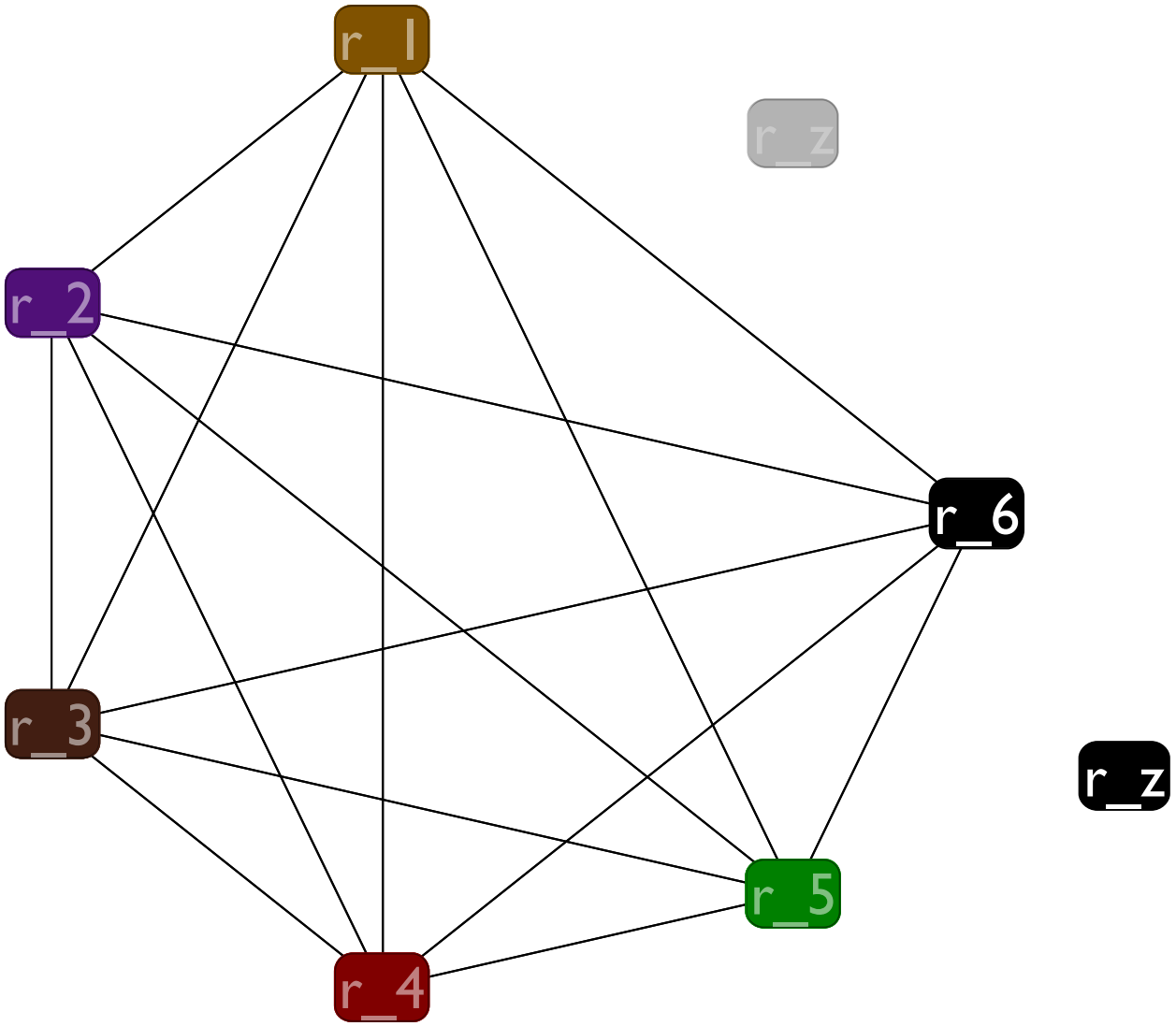

r_1
r_2
r_3
r_4
r_5
r_6
r_z
r_4
r_5
r_6
r_z

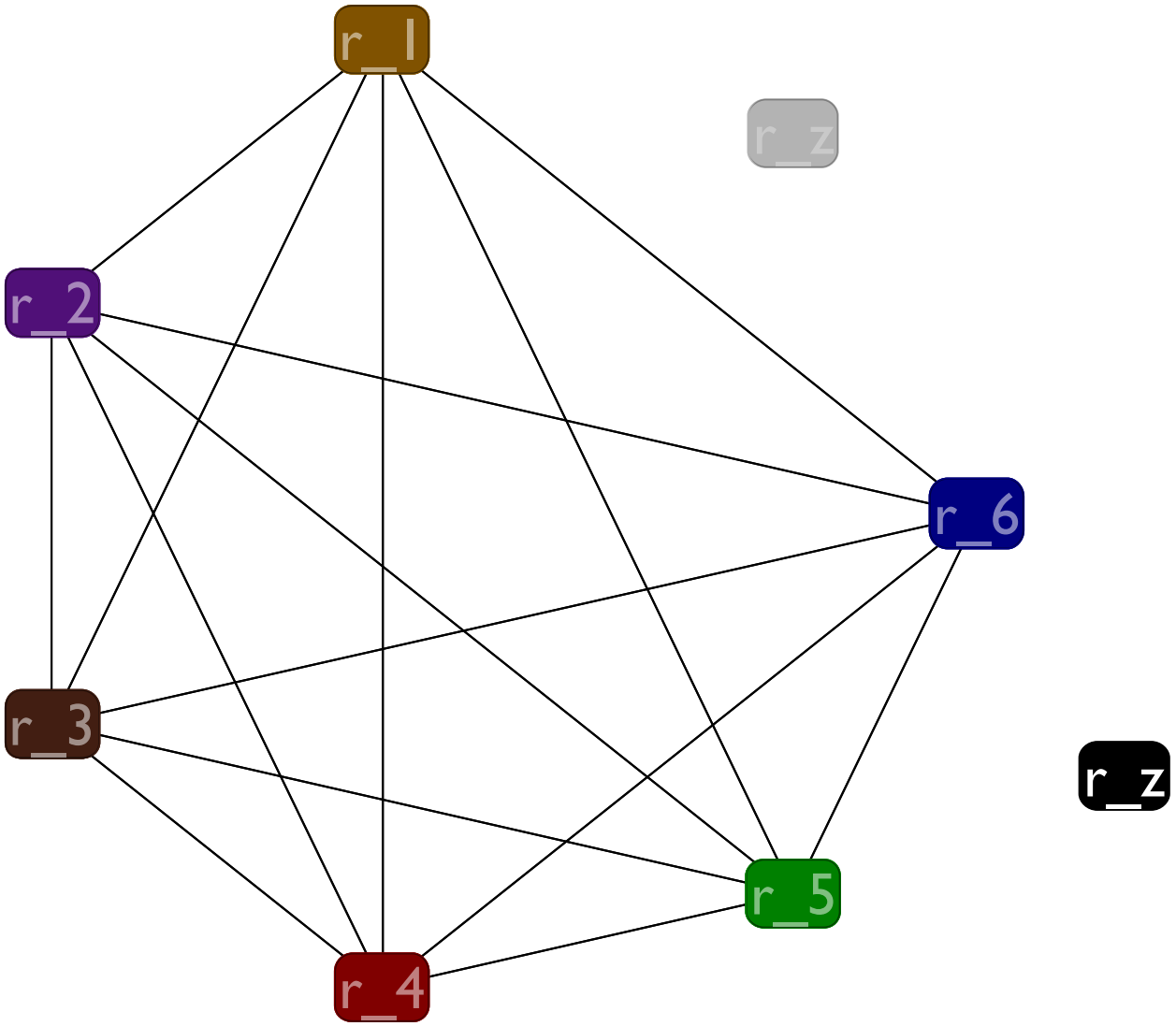

r_1
r_2
r_3
r_4
r_z
r_6
r_5
r_5
r_6
r_z

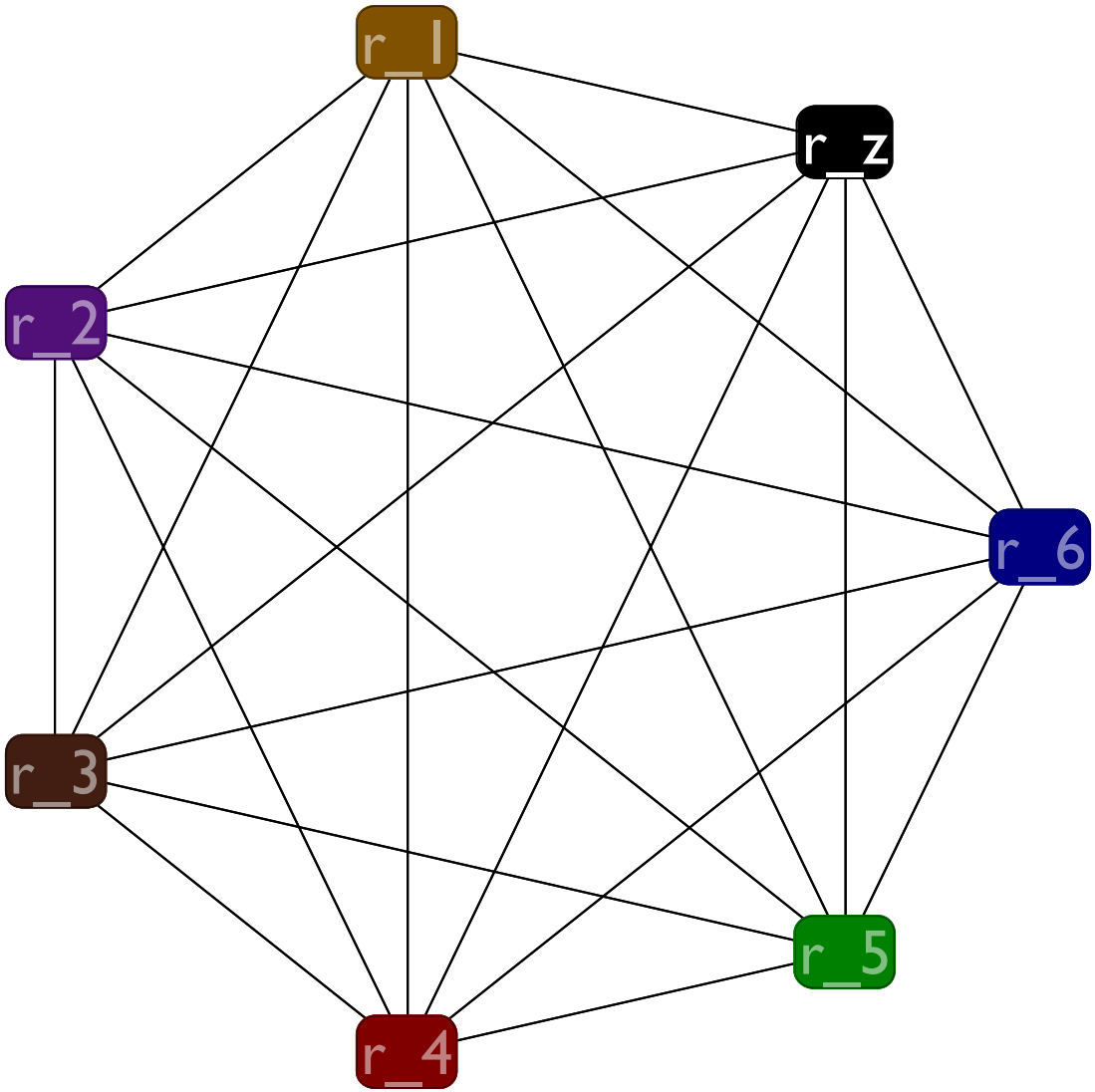

r_1
r_2
r_3
r_4
r_z
r_6
r_5
r_5
r_6
r_z

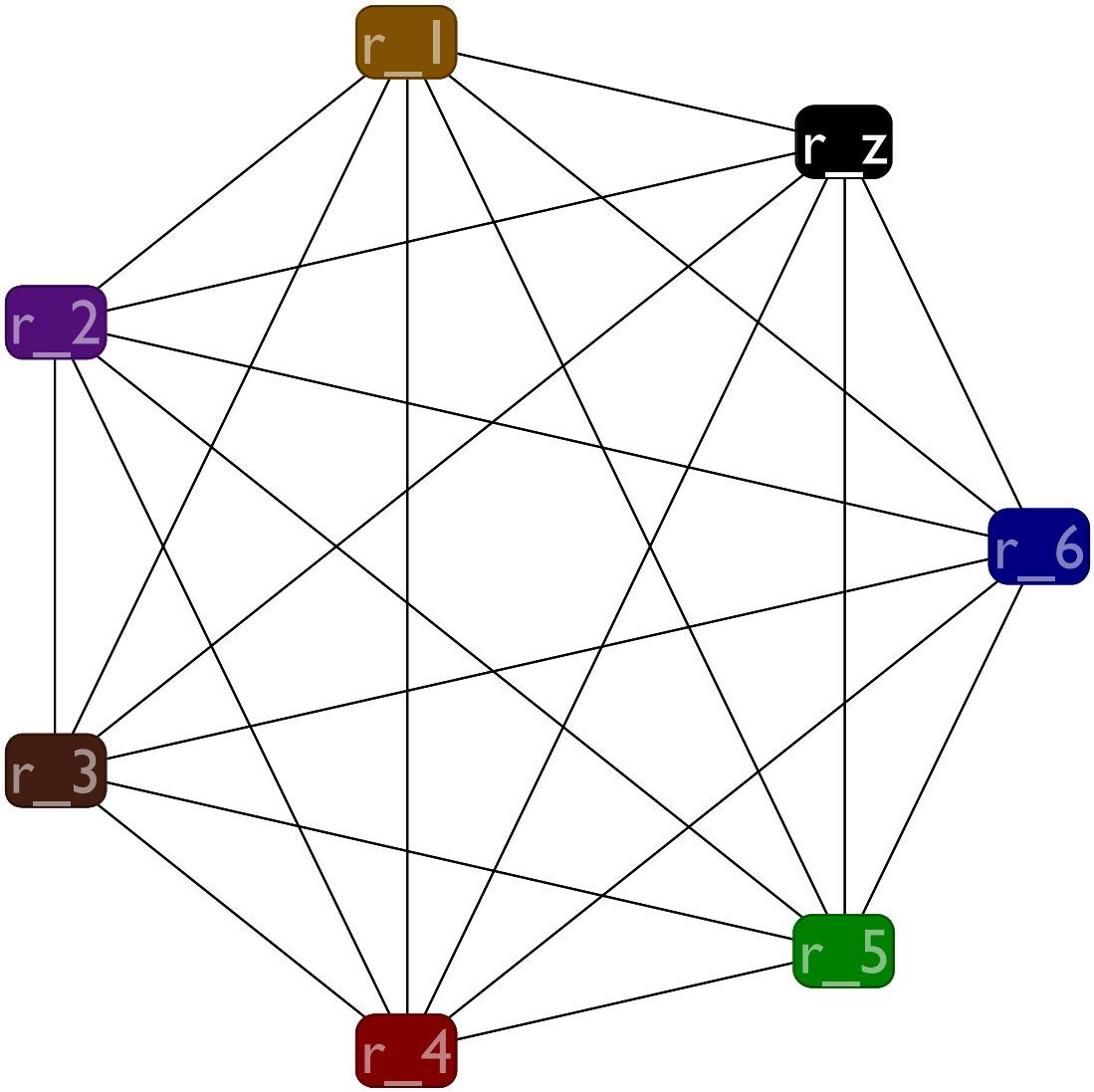

r_1
r_z
r_2
r_6
r_6
r_3
r_z
r_5
r_4


r_1
r_z
r_2
r_6
r_6
r_3
r_z
r_5
r_4


r_1
r_z
r_2
r_6
r_3
r_z
r_5
r_4


r_1
r_z
r_2
r_6
r_3
r_z
r_5
r_4


r_1
r_2
r_3
r_4
r_5
r_6
r_z


r_1
r_z
r_2
r_6
r_3
r_5
r_4

Coalescing during graph coloring

Extend the interference graph with a new kind of edge, called a move edge.

A move edge connects two nodes if there is a `(x <- y)` instruction in the program

Combine two move-edge connected nodes into a single node (at any step in the coloring algorithm), if:

- They don't interfere
- The resulting node has fewer than 6 neighbors

This ensures the graph is colorable (if it was before)

Roadmap: putting it all together

Programming

There are a number of different modules to put together

- A liveness library: gen & kill functions on instructions; the in and out loop
- A graph library: creating graphs, creating nodes, creating edges, removing nodes (and their edges), iterating over edges and nodes
- An interference library: build a graph from the liveness information

Programming

There are a number of different modules to put together, cotd

- A coloring library: color a graph using the coloring algorithm
- A spilling library: given a variable and a stack position, rewrite the program to move the variable in and out of the stack right as it is used
- The final translation: When you have a valid coloring, rewrite the variables to use registers and insert the `esp` adjustment, turning the L2 program into an L1 one.

Test

Unit testing

The most underrated part of developing good software is testing it well.

- Build simple (unit) tests for the api for each module, as you design the API
- As you write the code, write a test for each different case in the code together with the case itself
- Whenever you find a bug, always add a test case *before* fixing the bug; make sure the test case fails so you know you wrote it properly