Support of Probabilistic Pointer Analysis in the SSA Form

Rachel Menge, Ben Reeves, and Carson Boden
We probably can give you a few pointers

Pointer Analysis detects aliasing between pointers

Does *p* alias *q*?

```
int* p = &a;
int* q = &a;
```

Must-Alias

```
int* p = &a;
int* q = &b;
```

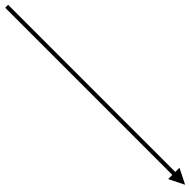
No-Alias

```
int* p = &a;
int* q = r;
```

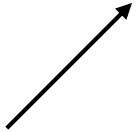
May-Alias

Pointer Analysis can reduce loads and stores in code

```
int* p = malloc(sizeof(int));
int* q = p;

*p = 17;
int a = *q;
```



```
r10 = &new_int
r11 = r10

store(r10, 17)
r1 = load(10)
```

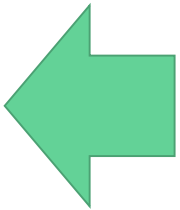


```
r10 = &new_int
r11 = r10

store(r10, 17)
r1 = 17
```

May-Alias case prevents possible optimizations

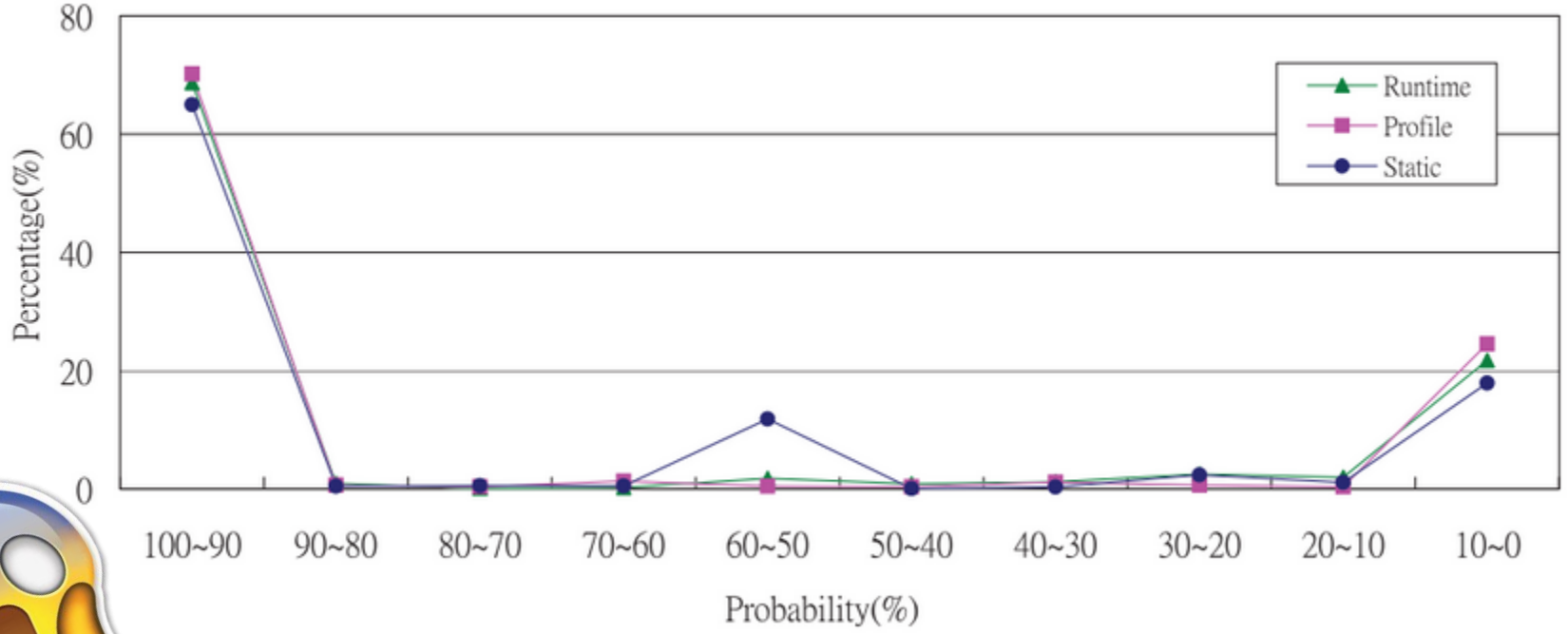
```
int* p = &a;
int* q = &b;

int i = 0;
while (i < 583) {
    *p = *q / 2 * 14;

    if (i == 580)
        q = p;
}
```

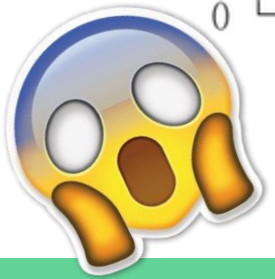
Mostly loop-invariant



May-Alias

Most May-Aliases are on the extremes of probability





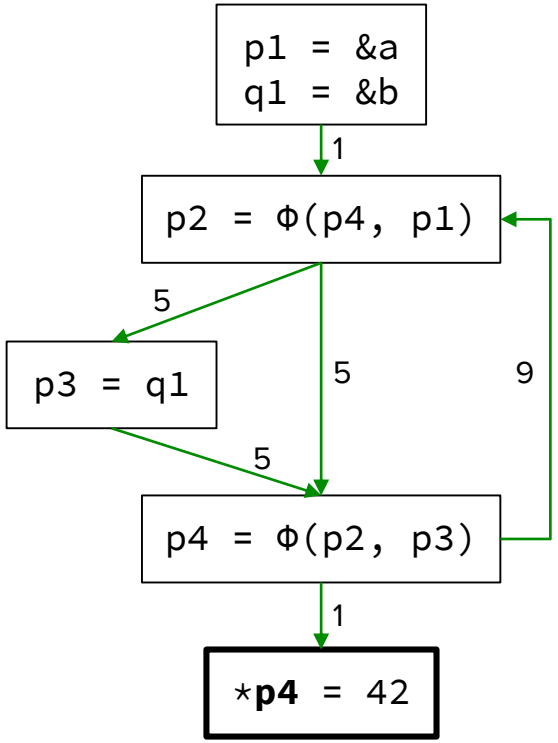
Probabilistic Pointer Analysis quantifies the chance of alias

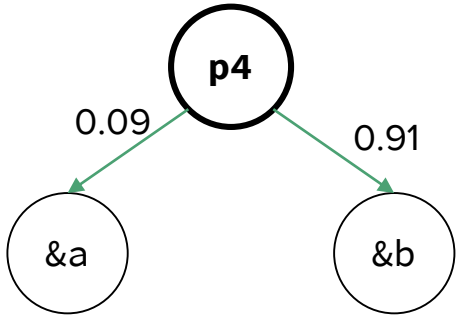
```
int* p = &a;
int* q = &b;

int i = 0;
while (i < 583) {
    *p = *q / 2 * 14;

    if (i == 580)
        q = p;
}
```

Pointer	Value	Pr(s, <ptr, val>)
p	a	583 / 583
p	b	0 / 583
q	a	3 / 583
q	b	580 / 583

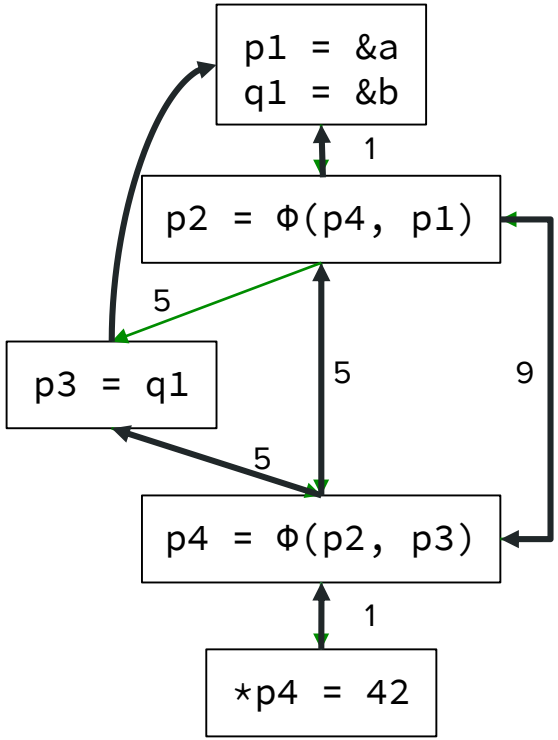
“The points-to probability from q to a is ~0.5%”

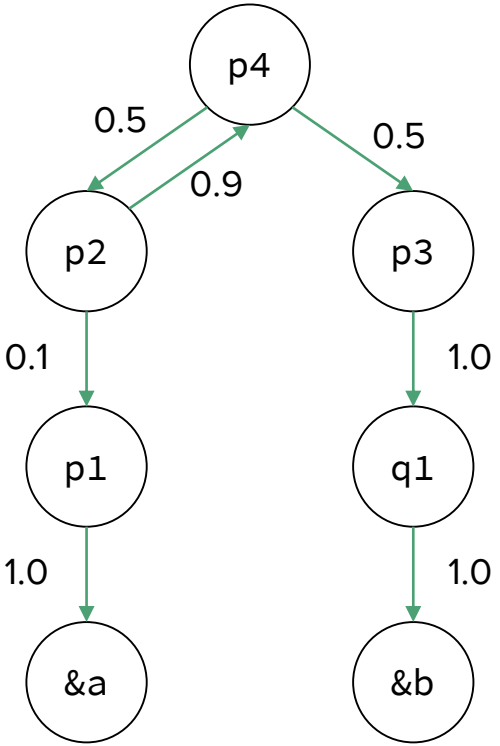
Goal: encode points-to probabilities as a weighted graph



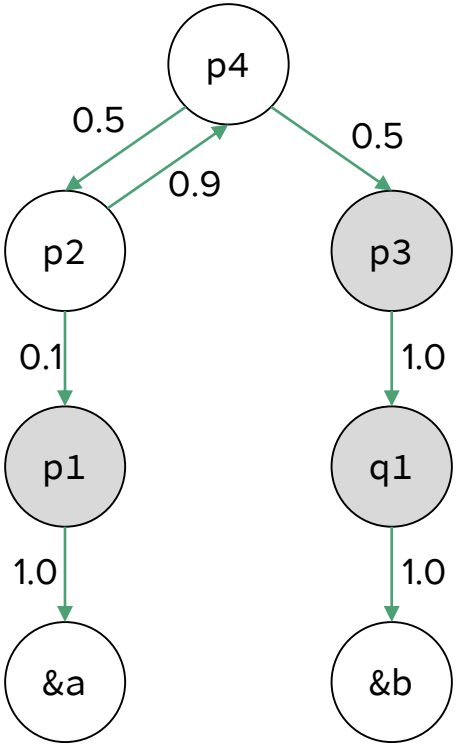


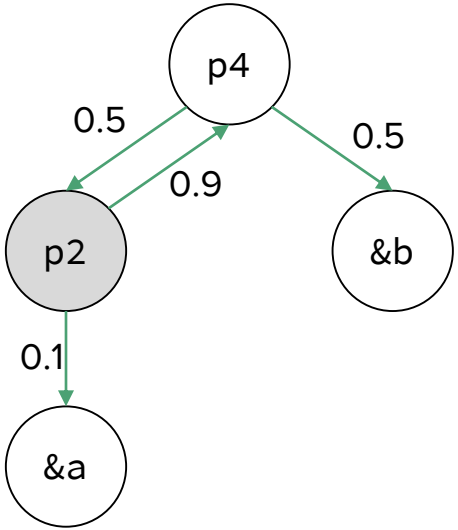
SSA allows for easy back-tracing through use-def chains

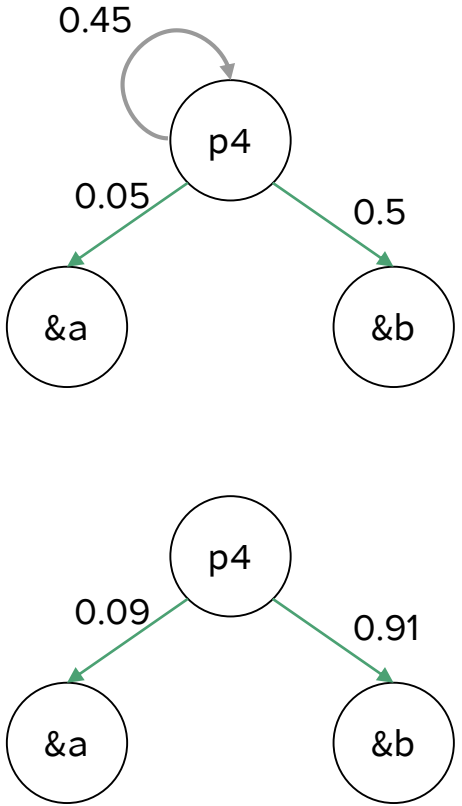




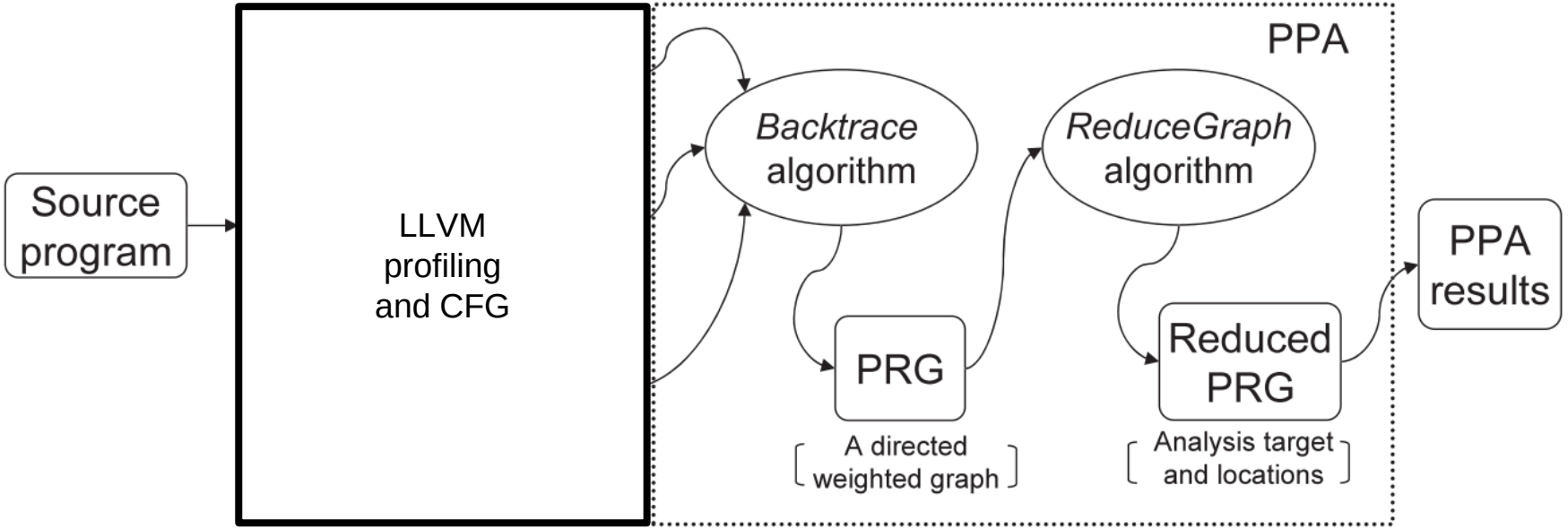
Reduce the graph



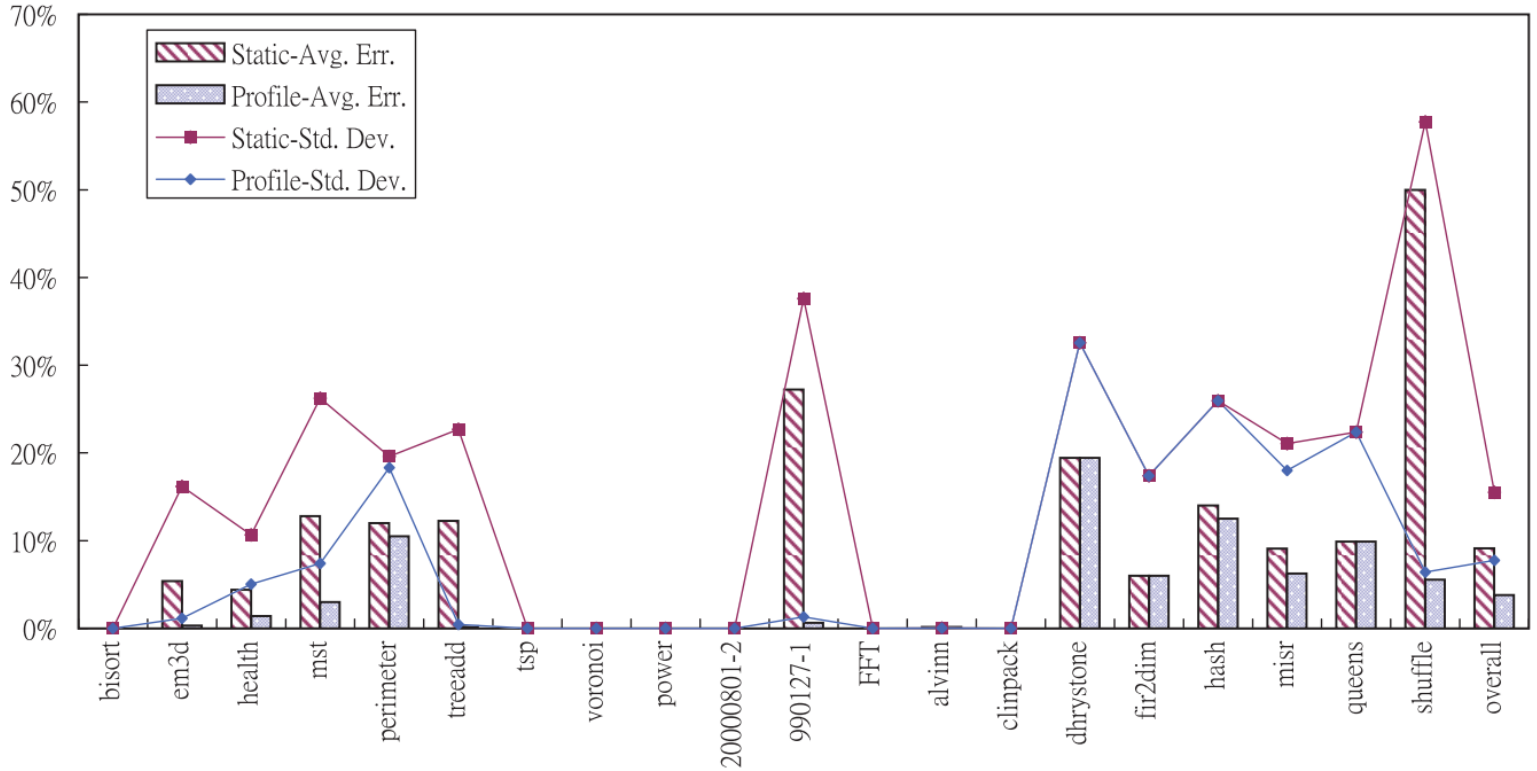




To create the points-to vectors, need to perform prior steps



Error improves when using profile data



How can we use these metrics to optimize code?

Perform the same optimizations for Must- and No-Alias pointers

Speculatively optimize, add fallback if the analysis is incorrect

Other things to do? Lots of possibilities, not super explored