دانشکده‌ی مهندسی کامپیوتر

طراحی کامپایلر

سارا آذرنوش ۹۸۱۷۰۶۶۸

تمرین سری چهارم

۱

۱. برای تشخیص متغیرهای زنده در نقطه مشخص شده، از الگوریتم Liveness Analysis استفاده می‌کنیم. با توجه به کد ارائه شده و نقطه مشخص شده در تصویر، می‌توانیم به صورت زیر متغیرهای زنده را تشخیص دهیم:

بر اساس تجزیه و تحلیل زنده بودن، زمانی که همه متغیرها در ابتدا مرده باشند (با فلش سبز مشخص شده اند)، متغیرهای A، B و X در نقطه مشخص شده در برنامه زنده هستند.

- A: زنده
- B: زنده
- C: مرده

۲. برای بدست آوردن Dataflow Information برای متغیرهای A و B و C در نقطه‌ای که با فلش قرمز مشخص شده است، به الگوریتم Constant Propagation نیاز داریم. این الگوریتم به ما اجازه می‌دهد تا مقادیر متغیرها را در نقطه مورد نظر بدست آوریم.

با توجه به تصویر و الگوریتم Constant، Propagation می‌توانیم به صورت زیر Dataflow Information را برای متغیرهای A و B و C در نقطه‌ای که با فلش قرمز مشخص شده است بدست آوریم:

برای A، مقدار X*Y است.
برای B، مقدار X*Y است.
برای C، مقدار A-B است که ۰ است (از آنجایی که A و B مقدار X*Y یکسانی دارند).
به طور خلاصه، تجزیه و تحلیل زنده بودن نشان می دهد که متغیرهای A، B و X در نقطه مشخص شده زنده هستند، و انتشار ثابت نشان می دهد که A و B دارای مقدار یکسان X*Y هستند، در حالی که C دارای مقدار ۰ در آن نقطه است.

۲

برای ترسیم Data Flow و انجام عملیات Liveness، Analysis کد زیر را در نظر بگیرید:

```
L0: e := 0
b := 0
d := 2
a := 9

L1: a := b + 2
c := d + 5
d := d + 10
c := d + c
e := e - c
f := b * a
```

```
b := b * b
if e < f goto L3

L2: e := e + f
goto L4

L3: e := e + 2
b := e + 2
a := e - 2

L4: d := d + 4
d:= b * b
if b != e goto L1
```

۱. Flow Data:
برای ترسیم Data، Flow باید به ترتیب خط به خط از شروع کد تا انتها رفته و هر بار که یک متغیر تعریف شده یا استفاده شده است، یک یال (یا رابطه) بین دوگانهای مربوطه رسم کنیم. الگوریتم زیر را می‌توان برای ترسیم Data Flow استفاده کرد:
- برای هر خط کد:
- اگر متغیر تعریف شده است، یک یال از نقطه تعریف به نقطه استفاده رسم کنید.
- اگر متغیر استفاده شده است، یک یال از نقطه استفاده به نقطه تعریف رسم کنید.
- در نهایت، Data Flow شامل تمام یال‌های رسم شده خواهد بود.

۱. L۰
- متغیر b تعریف می‌شود.

۲. b
- متغیر d استفاده می‌شود. - یالی از b به d رسم می‌شود.

۳. d
- متغیر a استفاده می‌شود. - یالی از d به a رسم می‌شود.

a
- L۱ تعریف می‌شود. - یالی از a به L۱ رسم می‌شود.

۴. L۱
- متغیر c استفاده می‌شود. - یالی از L۱ به c رسم می‌شود.

۵. c
- L۲ تعریف می‌شود. - یالی از c به L۲ رسم می‌شود.

۶. L۲
- L۳ تعریف می‌شود. - یالی از L۲ به L۳ رسم می‌شود.

۷. L۳
- L۴ تعریف می‌شود. - یالی از L۳ به L۴ رسم می‌شود.

۸. L۴
- متغیر b استفاده می‌شود. - یالی از L۴ به b رسم می‌شود.

نتیجهٔ ترسیم Data Flow برای کد فوق به صورت زیر است:
L0 -> b -> d -> a -> L1 -> c -> L2 -> L3 -> L4

۲. Analysis Liveness:
برای انجام عملیات Liveness Analysis باید به ترتیب خط به خط از انتها تا شروع کد رفته و هر بار که یک متغیر تعریف شده

یا استفاده شده است، وضعیت Liveness (زنده بودن) آن را مشخص کنیم. الگوریتم زیر را می‌توان برای انجام عملیات Analysis Liveness استفاده کرد:

- برای هر خط کد (شروع از انتها):
- اگر متغیر تعریف شده است، آن را به مجموعهٔ متغیرهای زنده اضافه کنیدادامه الگوریتم Liveness: Analysis
- اگر متغیر استفاده شده است، آن را به مجموعهٔ متغیرهای زنده اضافه کنید
- اگر متغیر قبلاً در مجموعهٔ متغیرهای زنده وجود داشت، تغییری ایجاد نکنید.
- اگر متغیر قبلاً در مجموعهٔ متغیرهای زنده وجود نداشت، تغییری در مجموعهٔ متغیرهای زنده به وجود آید.
- در هر خط کد، مجموعهٔ متغیرهای زنده را به خط قبلی منتقل کنید.
- در نهایت، Liveness Analysis شامل مجموعهٔ متغیرهای زنده در هر خط کد خواهد بود.

۱. L4: d, b
- در خط L۴، متغیر d تعریف شده است و برای اولین بار استفاده می‌شود. بنابراین، متغیر d در وضعیت زنده (live) است.
- متغیر b نیز در خط L۴ استفاده می‌شود و قبلاً تعریف شده است. بنابراین، متغیر b نیز در وضعیت زنده (live) است.

۲. L3: e, b
- در خط L۳، متغیر e تعریف شده است و برای اولین بار استفاده می‌شود. بنابراین، متغیر e در وضعیت زنده (live) است.
- متغیر b نیز در خط L۳ استفاده می‌شود و قبلاً تعریف شده است. بنابراین، متغیر b نیز در وضعیت زنده (live) است.

۳. L2: e, f, b
- در خط L۲، متغیر e در حالت زنده (live) است، زیرا در خط قبلی تعریف شده و در خط فعلی استفاده می‌شود.
- متغیر f نیز در خط L۲ تعریف شده و برای اولین بار استفاده می‌شود. بنابراین، متغیر f نیز در وضعیت زنده (live) است.
- متغیر b در خط L۲ استفاده می‌شود و قبلاً تعریف شده است. بنابراین، متغیر b نیز در وضعیت زنده (live) است.

۴. L1: e, f, b, a
- در خط L۱، متغیر e در حالت زنده (live) است، زیرا در خط قبلی تعریف شده و در خط فعلی استفاده می‌شود.
- متغیر f نیز در خط L۱ در حالت زنده (live) است، زیرا در خط قبلی تعریف شده و در خط فعلی استفاده می‌شود.
- متغیر b نیز در خط L۱ استفاده می‌شود و قبلاً تعریف شده است. بنابراین، متغیر b نیز در وضعیت زنده (live) است.
- متغیر a در خط L۱ تعریف شده و برای اولین بار استفاده می‌شود. بنابراین، متغیر a نیز در وضعیت زنده (live) است.

۵. L0: e, f, b, a, c, d
- در خط L۰، متغیر e در حالت زنده (live) است، زیرا در خط قبلی تعریف شده و در خط فعلی استفاده می‌شود.
- متغیر f نیز در خط L۰ در حالت زنده (live) است، زیرا در خط قبلی تعریف شده و در خط فعلی استفاده می‌شود.
- متغیر b نیز در خط L۰ استفاده می‌شود و قبلاً تعریف شده است. بنابراین، متغیر b نمراحل بدست آوردن وضعیت Liveness برای هر خط کد به ترتیب از انتها تا شروع کد به شرح زیر است:

حالت‌های Liveness در هر خط کد را به ترتیب از انتها تا شروع کد بررسی می‌کنیم:

- L4: d, b
- L3: e, b
- L2: e, f, b
- L1: e, f, b, a
- L0: e, f, b, a, c, d

۳

```
int x = 10;
int y = 20;
if (x > y) {
int z = x + y;
} else {
int w = x - y;
}
```

```
int result = y - x;
```

با توجه به کد x < y است بنابراین نیازی به شرط نیست و همیشه وارد else میشود. بنابراین کد را به حالت زیر ساده میکنیم.

```
int x = 10;
int y = 20;
int w = x - y;
int result = y - x;
```

در این تکه کد، متغیرهای 'z' و 'w' در بلاک‌های شرطی تعریف شده‌اند، اما در ادامهٔ کد استفاده نمی‌شوند. از این رو، این بلاک‌های کد به عنوان تکه کد مرده شناخته می‌شوند و می‌توانیم آنها را حذف کنیم.
با توجه به شناسایی تکه کد مرده، می‌توانیم کد را بهینه‌سازی کنیم به صورت زیر:

```
int x = 10;
int y = 20;
int result = y - x;
```

در این حالت، تکه کد مرده حذف شده است و از این رو کد بهینه‌تر و ساده‌تر می‌شود. دلیل بهینه سازی این است که بلاک‌هایی که محاسباتی انجام نمی‌دهند و مقادیر متغیرها را تغییر نمی‌دهند، به طور عملیاتی بی‌فایده هستند و می‌توانند حذف شوند تا کد خواناتر و ساده‌تر شود و همچنین بهینه‌سازی‌های کامپایلر را تسهیل کنند.

۴

تکه کد زیر را در نظر بگیرید:

```
int x = 2;
int y = 3;
int z = x + y;
x = 4;
int w = z * 2;
```

برای بهینه‌سازی این کد، می‌توانیم از روشهای "Constant Propagation" و "Global Optimization" استفاده کنیم.

۱. Constant Propagation:
در این روش، مقادیر ثابت را در جایگزینی‌های مناسب استفاده می‌کنیم. در این حالت، 'x' در خط دوم به مقدار ۴ تغییر می‌کند. از آنجا که مقدار 'z' در خط سوم به 'x' + 'y' برابر است و مقادیر 'x' و 'y' در این نقطه مشخص هستند، می‌توانیم مقدار 'z' را به شکل 'z' = ۲ + ۳' جایگزین کنیم. پس می‌توانیم کد را به صورت زیر بهینه‌سازی کنیم:

```
int x = 2;
int y = 3;
int z = 5;
x = 4;
int w = z * 2;
```

۲. Global Optimization:
در این روش، می‌توانیم متغیرهای غیرمورد استفاده را حذف کنیم. از آنجا که متغیر 'y' در ادامهٔ کد استفاده نمی‌شود، می‌توانیم آن را حذف کنیم. پس می‌توانیم کد را به صورت زیر بهینه‌سازی کنیم:

```
int x = 2;
int z = 5;
x = 4;
int w = z * 2;
```

با استفاده از روشهای بهینه‌سازی "Constant Propagation" و "Global Optimization"، کد اصلی بهینه‌سازی شده و متغیرهای غیرضروری حذف شده است. این بهینه‌سازی‌ها باعث ساده‌تر شدن کد و کاهش محاسبات تکراری می‌شوند، که در نتیجه کد بهینه‌تر و کارآمدتری را ارائه می‌دهند.

۵

با توجه به قطعه کدی که ارائه داده شده است، می‌توانیم Symbol Table، Stack Scope را بعد از کامپایل خطوط ۱۰ و ۱۴ به ترتیب نشان دهیم.

بعد از خط ۱۰:
Symbol Table: - متغیرهای محلی:
- نام متغیر: a - نوع متغیر: real[۱..۱۰] - آدرس: main.p.f۱.b
- متغیرهای فراخوانی شده:
- نام متغیر: i - نوع متغیر: integer - آدرس: main.i
- نام متغیر: n - نوع متغیر: integer - آدرس: main.p.f۱.n

Stack Scope:
- تابع: main
- متغیرهای محلی:
- نام متغیر: i - نوع متغیر: integer - آدرس: main.i
- نام متغیر: j - نوع متغیر: integer - آدرس: main.j
- نام متغیر: s - نوع متغیر: integer - آدرس: main.s
- نام متغیر: a - نوع متغیر: real[۱..۱۰] - آدرس: main.p.f۱.b
- توابع فراخوانی شده:
- نام تابع: p - نوع تابع: procedure - دامنه استک: main.p

بعد از خط ۱۴:
Symbol Table:
- متغیرهای محلی:
- نام متغیر: arr۲ - نوع متغیر: real[۱..۳] - آدرس: main.p.p۲.arr۲
- متغیرهای فراخوانی شده:
- نام متغیر: j - نوع متغیر: integer - آدرس: main.j
- نام متغیر: a - نوع متغیر: real[۱..۱۰] - آدرس: main.p.f۱.b
- نام متغیر: n - نوع متغیر: integer - آدرس: main.p.f۱.n

Stack Scope:
- تابع: main
- متغیرهای محلی:
- نام متغیر: i - نوع متغیر: integer - آدرس: main.i
- نام متغیر: j - نوع متغیر: integer - آدرس: main.j
- نام متغیر: s - نوع متغیر: integer - آدرس: main.s
- نام متغیر: a - نوع متغیر: real[۱..۱۰] - آدرس: main.p.f۱.b
- نام متغیر: arr۲ - نوع متغیر: real[۱..۳] - آدرس: main.p.p۲.arr۲
- توابع فراخوانی شده:
- نام تابع: p - نوع تابع: procedure - دامنه استک: main.p
- متغیرهای محلی:
- نام متغیر: n - نوع متغیر: integer - آدرس: main.p.f۱.n

آدرس‌ها در اینجا فقط برای درک بیشتر از جدول نماد و دامنه استک است و در واقعیت ممکن است به صورت دقیق در برنامه اجرا نشوند.