تحقیق فاز 5 پروژه مبانی نظریه محاسبه

PDA و کاربردهای آن

Pushdown Acceptors اولین بار توسط چامسکی و ایوی رسمیت یافتند، اگرچه مفهوم نوار فشاری از سال 1954 مورد استفاده قرار گرفت.

یک PDA یک رشته ورودی داده شده را از چپ به راست می خواند. در هر مرحله، از جدول انتقال ها با توجه به نماد ورودی، وضعیت فعلی و نماد بالای پشته، یک انتقال را انتخاب می کند. یک PDA همچنین می تواند پشته را به عنوان بخشی از انجام یک انتقال دستکاری کند. دستکاری می تواند به این صورت باشد که یک نماد خاص را به بالای پشته فشار دهید یا از بالای پشته خارج کنید. همچنین PDA می‌تواند پشته را نادیده بگیرد و آن را همانطور که هست رها کند.
اگر در هر موقعیتی حداکثر یک عمل انتقالی امکان پذیر باشد، آنگاه PDA یک PDA قطعی (DPDA) نامیده می شود.

PDA ها نسبت به FA ها توانایی بیشتری دارند اما نسبت به ماشین های تورینگ توانایی کمتری دارند. PDA های قطعی (DPDA) می توانند همه CFL های قطعی را تشخیص دهند و اغلب در طراحی Parserاستفاده می شود.

از کاربرد های PDA میتوان به این موارد اشاره کرد:

- برای طراحی فاز Parsing یک کامپایلر (Syntax Analysis).
- برای پیاده سازی کاربردهای پشته (Stack Applications).
- برای ارزیابی عبارات حسابی (Arithmetic Expressions).
- برای حل مشکل برج هانوی(Hanoi Tower).
- سیستم پردازش تراکنش های آنلاین (Online Transaction Process System).

موردی که در این تحقیق بیشتر روی آن تمرکز میکنیم مورد اول یا همان Parsing است.

کاربرد PDA در Parsing

توسعه CFGها کمک کرد تا توسعه کامپایلر از یکی از چالش‌های اصلی اوایل علوم کامپیوتر به یک فعالیت عامیانه تبدیل شود که می‌تواند توسط تیم‌های نسبتاً کوچک با درجه موفقیت بالا انجام شود.

PDA ها را می توان در فرآیند کامپایل در بررسی Syntax برنامه ورودی استفاده کرد. در طول کامپایل، لازم است ورودی را با توجه به Syntax زبان Validate کنید. Syntax نوشتن عبارات حسابی در زیر آورده شده است:

$$S \rightarrow i\ A\ E\ T$$

$$E \rightarrow i\ O\ E \mid i$$

$$O \rightarrow + \mid - \mid * \mid /$$

$$A \rightarrow =$$

$$T \rightarrow ;$$

با استفاده از این قوانین، می توانیم عبارت هایی مانند i = i + i; را تولید کنیم.

این عبارات توسط PDA که به شرح زیر تعریف شده است تأیید می شوند:

$S \rightarrow i\ A\ E\ T$	Corresponds	$\delta(q, i, S) = (q, AET)$
$E \rightarrow i\ O\ E$	Corresponds	$\delta(q, i, E) = (q, OE)$
$E \rightarrow i$	Corresponds	$\delta(q, i, E) = (q, \varepsilon)$
$O \rightarrow + \mid - \mid * \mid /$	Corresponds	$\delta(q, +, O) = (q, \varepsilon)$ $\delta(q, -, O) = (q, \varepsilon)$ $\delta(q, *, O) = (q, \varepsilon)$ $\delta(q, /, O) = (q, \varepsilon)$
$A \rightarrow =$	Corresponds	$\delta(q, =, A) = (q, \varepsilon)$
$T \rightarrow ;$	Corresponds	$\delta(q, ;, T) = (q, \varepsilon)$

مثالی دیگر نمونه‌ای از یک الگوریتم Parsing به نام predictive parsing است که از اصول طراحی کامپایلر توسط Aho، Sethi و Ullman، اقتباس شده است.

predictive parsing توسط یک جدول parsing هدایت می شود. این جدول که در الگوریتم M نامیده می شود، یک جدول دو بعدی است که توسط یک نماد non-terminal (متغیر) در یک CFG و یک نماد ورودی نمایه می شود:

k = M[X,a] به این معنی است که اگر بالای پشته X باشد و ورودی a را مشاهده کنیم، می‌خواهیم با استفاده از kامین production استفاده کنیم.

```
parse (String w, Table M)
{
    ip = 0;
    stack = emptyStack;
    push S (starting symbol of grammar) onto stack
    repeat {
        X = stack.top();
        a = w[ip];
        if (X is a terminal) {
            if (X == a) {
                stack.pop();
                ++ip;
            } else {
                syntax-error();
            }
        } else { // X is a non-terminal
            if (M[X,a] >= 0) {
               pop X from the stack;
               R[] = symbols form right hand side of kth production
               push R[n-1], R[N-1], ..., R[0] onto stack;
            } else {
                syntax-error();
            }
        }
    } until stack is empty
}
```

در قلب آن، ما جدولی داریم که با یک ورودی و یک نماد پشته نمایه شده است، که به ما می گوید چه چیزی را روی پشته فشار دهیم و چه ورودی هایی باید مطابقت داشته باشند تا به اجرا ادامه دهیم. این همانند یک PDA به نظر می رسد.حتی یک PDA به خصوص پیچیده ای نیست.

چیزی که ممکن است همانند PDA به نظر نرسد. این است که این الگوریتم ممکن است به ورودی ورودی بعدی نگاه کند و از آن برای ایندکس کردن جدول استفاده کند بدون اینکه واقعاً آن را "consume" کند. در اصطلاح کامپایلر، از این به عنوان یک lookahead character یاد می شود. بنابراین یک نماد ورودی می تواند چندین انتقال را راه اندازی کند

با این حال، راه‌هایی برای جعل این رفتار با PDA وجود دارد. در واقع، می توان استدلال کرد که ϵ-transitions اجازه می دهند تا چیزی بسیار مشابه اتفاق بیفتد.

بنابراین به نظر می رسد که این الگوریتم ریشه در یک PDA داشته باشد، اما بدون non-determinism. به همین دلیل است که این نظر داده شده است که اکثر زبان های برنامه نویسی در کلاس زبان هایی قرار می گیرند که توسط DPDA قابل شناسایی هستند.