CS481F01 Notes on DPDAS

A. Demers

31 Oct

The construction given in Chapter F of the text seems strange in just the way you all noticed in lecture. Here's my proposal to fix it up.

Def: A triple (p, ε, A) is *stuck* for M if

$$\neg(\exists q, \beta)((p, \varepsilon, A) \rightsquigarrow_M (q, \beta))$$

That is, there is no next move for the machine reading ε from the input. Similarly, (p, a, A) is stuck for M if

$$\neg(\exists q, \beta)((p, a, A) \rightsquigarrow_M (q, \beta) \ \vee \ (p, \varepsilon, A) \rightsquigarrow_M (q, \beta))$$

Note if (p, a, A) is stuck for M then (p, ε, A) is stuck for M, but the converse is not necessarily true.

Construction: Given a DPDA M, we construct M' (which checks for end of input) and then M'' (which is guaranteed always to read to the end of input).

To construct M' shall add a complete set of primed states, two new rejecting states r and r', and a new accept state a', all at once.

The transitions of M' are as follows. First, some moves that simulate M initially:

$$(p, a, A) \rightsquigarrow_{M'} (q, \beta) \ \text{ if } \ (p, a, A) \rightsquigarrow_M (q, \beta)$$
$$(p, \varepsilon, A) \rightsquigarrow_{M'} (q, \beta) \ \text{ if } \ (p, \varepsilon, A) \rightsquigarrow_M (q, \beta)$$

Next, if M gets stuck before reading end of input, M' goes to the reject state:

$$(p, a, A) \rightsquigarrow_{M'} (r, A) \ \text{ if } \ (p, a, A) \text{ is stuck in } M$$
$$(p, \varepsilon, A) \rightsquigarrow_{M'} (r, A) \ \text{ if } \ (p, \varepsilon, A) \text{ is stuck in } M$$

Next, we make M' go to a primed state on reading the endmarker:

$$(p, \dashv, A) \rightsquigarrow_{M'} (q', \beta) \text{ if } (p, \dashv, A) \rightsquigarrow_{M} (q, \beta)$$
$$(p, \dashv, A) \rightsquigarrow_{M'} (r', A) \text{ if } (p, \dashv, A) \text{ is stuck in } M$$

Next, after reading the endmarker we loop in accepting or rejecting states as appropriate:

$$(p', \varepsilon, A) \rightsquigarrow_{M'} (q', \beta) \text{ if } (p, \varepsilon, A) \rightsquigarrow_{M} (q, \beta), \quad p \notin F$$
$$(p', \varepsilon, A) \rightsquigarrow_{M'} (r', A) \text{ if } (p, \varepsilon, A) \text{ is stuck in } M, \quad p \notin F$$
$$(p', \varepsilon, A) \rightsquigarrow_{M'} (a', A) \text{ if } p \in F$$

Note only ε moves are possible for primed states.

You should convince yourself that M' runs forever (the computation sequence is infinite) on every input. It may go into an infinite loop, never reaching the end of the input. However, if it *does* reach the end of the input, then if the input word is accepted the machine eventually reaches (and remains looping in) the explicit accept state a'.

At this point, we can construct M'' just as on page 179 of the text. By that construction, any transition

$$(p, \varepsilon, A) \rightsquigarrow_{M'} (q, \beta)$$

that can occur infinitely often in shortest-stack positions in a rejecting computation is replaced by

$$(p, \varepsilon, A) \rightsquigarrow_{M''} (r, A) \quad \text{or} \quad (p, \varepsilon, A) \rightsquigarrow_{M''} (r', A)$$

depending on whether p itself is a primed or unprimed state. Note in particular that by this construction a loop of rejecting states at the end of the input will be "cut" with a transition to the explicit rejecting state r'.

Now M'' still recognizes the same language as M and M', but has the property that every computation eventually consumes the entire input. Every accepting computation eventually ends up in a loop in state a', and every rejecting computation eventually ends up in a loop in state r'. This is the behavior we sought. We can complement the accepted language simply by exchanging the roles of a' and r'.