Character Class Intersection

Character class intersection is supported by Java, JGsoft V2, and by Ruby 1.9 and later. It makes it easy to match any single character that must be present in two sets of characters. The syntax for this is `[class&&[intersect]]`. You can use the full character class syntax within the intersected character class.

If the intersected class does not need a negating caret, then Java and Ruby allow you to omit the nested square brackets: `[class&&intersect]`.

You cannot omit the nested square brackets in PowerGREP. If you do, PowerGREP interprets the ampersands as literals. So in PowerGREP `[class&&intersect]` is a character class containing only literals, just like `[clas&inter]`.

The character class `[a-z&&[^aeiuo]]` matches a single letter that is not a vowel. In other words: it matches a single consonant. Without character class subtraction or intersection, the only way to do this would be to list all consonants: `[b-df-hj-np-tv-z]`.

The character class `[\p{Nd}&&[\p{IsThai}]]` matches any single Thai digit. `[\p{IsThai}&&[\p{Nd}]]` does exactly the same.

Intersection of Multiple Classes

You can intersect the same class more than once. `[0-9&&[0-6&&[4-9]]]` is the same as `[4-6]` as those are the only digits present in all three parts of the intersection. In Java and Ruby you can write the same regex as `[0-9&&[0-6]&&[4-9]]`, `[0-9&&[0-6&&4-9]]`, `[0-9&&0-6&&[4-9]]`, or just `[0-9&&0-6&&4-9]`. The nested square brackets are only needed if one of the parts of the intersection is negated.

If you do not use square brackets around the right hand part of the intersection, then there is no confusion that the entire remainder of the character class is the right hand part of the intersection. If you do use the square brackets, you could write something like `[0-9&&[12]56]`. In Ruby, this is the same as `[0-9&&1256]`. But Java has bugs that cause it to treat this as `[0-9&&56]`, completely ignoring the nested brackets.

PowerGREP does not allow anything after the nested `]`. The characters `56` in `[0-9&&[12]56]` are an error. This way there is no ambiguity about their meaning.

You also shouldn't put && at the very start or very end of the regex. Ruby treats `[0-9&&]` and `[&&0-9]` as intersections with an empty class, which matches no characters at all. Java ignores leading and trailing && operators. PowerGREP treats them as literal ampersands.

Intersection in Negated Classes

The character class `[^1234&&[3456]]` is both negated and intersected. In Java and PowerGREP, negation takes precedence over intersection. Java and PowerGREP read this regex as "(not 1234) and 3456". Thus in Java and PowerGREP this class is the same as `[56]` and matches the digits 5 and 6. In Ruby, intersection takes precedence over negation. Ruby reads `[^1234&&3456]` as "not (1234 and 3456)". Thus in Ruby this class is the same as `[^34]` which matches anything except the digits 3 and 4.

If you want to negate the right hand side of the intersection, then you must use square brackets. Those automatically control precedence. So Java, PowerGREP, and Ruby all read `[1234&&[^3456]]` as "1234 and (not 3456)". Thus this regex is the same as `[12]`.

Notational Compatibility with Other Regex Flavors

The ampersand has no special meaning in character classes in any other regular expression flavors discussed in this tutorial. The ampersand is simply a literal, and repeating it just adds needless duplicates. All these flavors treat `[1234&&3456]` as identical to `[&123456]`.

Strictly speaking, this means that the character class intersection syntax is incompatible with the majority of other regex flavors. But in practice there’s no difference, because there is no point in using two ampersands in a character class when you just want to add a literal ampersand. A single ampersand is still treated as a literal by Java, Ruby, and PowerGREP.