深入理解運算原理 | 從簡單的機器到無所不能的程式

頁碼	位置	原述	更正	更正日期
31	最後一段、第2行後面	，而機器的行為是以控制機器執行時的狀態會如何改變的整組規則所描述。	，而機器的行為是由一組規則所描述，機器執行時的狀態會如何變化，會由這些規則決定。	2018/1/12
33	第1行	#reduce	#redice	2018/1/12
36	第2段、第1行後面	但要避免破壞更新強迫我們造成#reduce 完全清楚的結果。	但要避免破壞性的更新強迫我們造成#reduce 完全顯露出來的後果。	2018/1/12
36	第3段、第4行中間	這讓我們知道對環境化簡陳述式的確有效	這讓我們知道化簡陳述式會對環境有所影響	2018/1/12
40	第1行	所有的工作都一如預期，但我們若能以«else» 子句支援條件陳述式，就像«if (x) { y =1 }»，那就更棒了。	所有的運作都一如預期，但我們若能支援沒有«else»子句的條件陳述式（像是«if (x) { y =1 }»），那就更棒了。	2018/1/12
46	≪ ≫ 運算式 下一段的第1行後面	小步語意使得我們必須從諸如«3»的不可約運算式辨識出像是«1 + 2» 的可化簡運算式，	我們必須以小步語意區分可化簡的運算式（像是«1 + 2» ）和不可化簡化的運算式（例如«3» ），	2018/1/12
50	≪ ≫ 應用 下一段的第2行前面	如果訊息往下經過抽象語法樹，然後一直到它抵達準備好化簡的程式碼片段，可能就會造成棘手的巢狀#reduce 呼叫 。	如果訊息沿著抽象語法樹往下傳，直到抵達準備好化簡的程式碼片段，可能就會造成少數的巢狀#reduce 呼叫 。	2018/1/12
52	第1段、第1行中間	反之，為了解釋新語言，它本身關心的是另一種語言既定意義的影響（一種比描述的語言更低階、更形式，或至少更好瞭解）。	反之，它本身關心的是利用另一種語言的既定意義來解釋新的語言，而另一種語言應該要比所描述的語言更低階、更正規，或至少更好瞭解。	2018/1/12
52	第2段最後一行後面	不需要混亂的練習。	不需要再大費周章的以實際走動作為溝通方式。	2018/1/12
58	第3段、第1行後面	相較之下，操作語意語言對語言的翻譯就像編譯器（compiler）：	相較之下，指稱語意的語言和語言之間的翻譯就像編譯器（compiler）：	2018/1/12
58	第3段、第3行中間	這些語意風格無一必能說明...	這些語意風格無一能說明...	2018/1/12
59	第6段、第1行	指稱語意的優點是操作語意更為抽象 ，	指稱語意的優點是比操作語意更為抽象 ，	2018/1/12
59	最後一段、第2行後面	指稱只能如其含義；...，那麼指稱語意就只能讓我們更接近實際執行的程式 ，	指稱就如同它的含義，...，那麼指稱語意才能讓我們更接近實際執行的程式，	2018/1/12