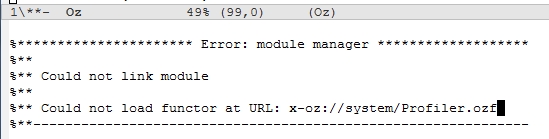
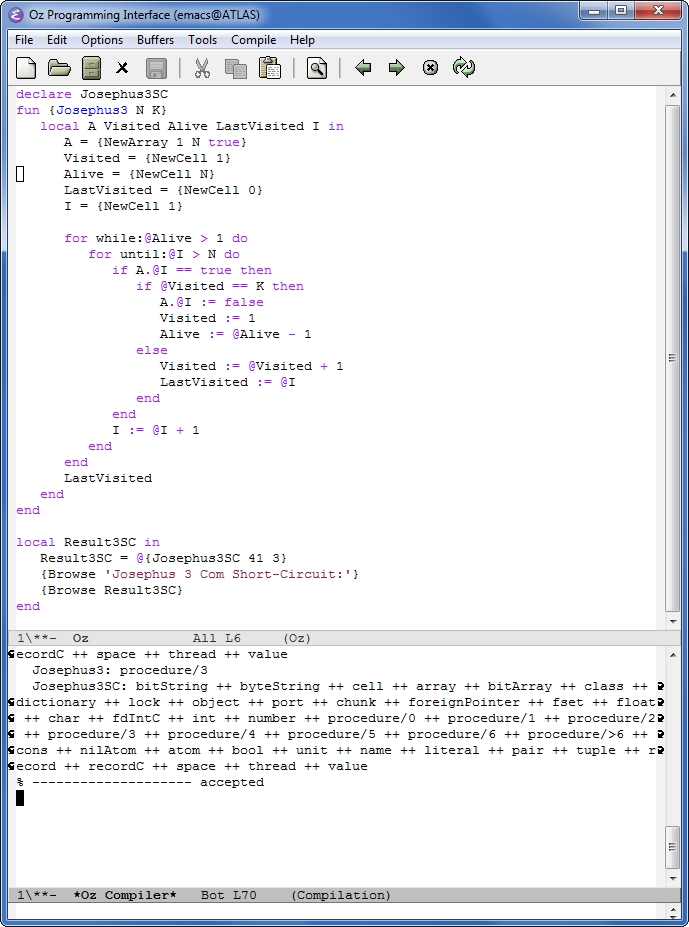
Eu tive 2 problemas principais com esse trabalho: O primeiro é que o profiler do Mozart não quer funcionar por algum motivo:



O segundo é que o Mozart é péssimo em explicar porque algum código não roda. Ou ele reclama de algo na linha errada, ou ele joga um erro sem especificar onde ocorre, ou então ele diz que tá tudo ok mas nada acontece, que é onde a minha tentativa de fazer short-circuit no Josephus3 morreu:



Ele aceita, mas o Browser não abre, não sei porque. Note que esse código era pra ser idêntico ao Josephus3 normal, que funciona, porém com I como célula e usado em um while (ou until nesse caso), que seria necessário para a tentativa de short-circuit que eu tinha em mente.

Com base nisso, a análise do tempo de execução dos códigos em Oz não será muito precisa, tendo de ser feita por meio da função nativa do Oz Time.time, que só conta o tempo em segundos. Quanto ao Josephus 3, quando comparado ao 1 e 2, farei a análise com base no código em Oz. Quando comparado aos dois short-circuits diferentes que fiz, farei baseado na versão em Python que eu escrevi originalmente. O código em Python está incluso com os arquivos.

Como nem eu nem o ChatGPT entendemos como remover o short-circuiting no Josephus2 sem quebrar o programa, a versão sem short-circuiting dele também não estará na análise.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X | N = 1K  K = 3 | N = 1K  K = 56 | N = 1K  K = 100 | N = 30K  K = 3 | N = 30K  K = 56 | N = 30K  K = 100 | N = 60K  K = 3 | N = 60K  K = 56 | N = 60K  K = 100 | N = 100K  K = 3 | N = 100K  K = 56 | N = 100K  K = 100 |
| Josephus 1 | 0 | 1 | 1 | 2 | 18 | 36 | 5 | 39 | 63 | 5 | 66 | 103 |
| Josephus 1  No SC | 0 | 1 | 3 | 4 | 55 | 113 | 9 | 126 | 213 | 12 | 218 | 355 |
| Josephus 2 | 0 | 1 | 1 | 2 | 18 | 36 | 5 | 39 | 63 | 5 | 66 | 103 |
| Josephus 3 Oz | 0 | 0 | 1 | 1 | 4 | 7 | 2 | 9 | 14 | 2 | 15 | 22 |
| Josephus 3 Python | 0.056 | 0.489 | 0.776 | 0.991 | 21 | 31 | 1.752 | 42 | 68 | 2.951 | 60 | 112 |
| Josephus 3  SC | 0.133 | 0.058 | 0.069 | 2.592 | 1.970 | 1.759 | 6.887 | 3.468 | 3.995 | 14.478 | 6.289 | 6.870 |
| Josephus 3  SC 2 | 0.073 | 1.308 | 2.263 | 3.217 | 90 | 135 | 6.545 | 154 | 290 | 10.800 | 288 | 494 |

O tempo é dado em segundos. Pela forma de coleta do tempo no Oz, não foi possível uma precisão menor que 1 segundo.

Com base nos dados coletados, é possível notar algumas coisas interessantes:

Para todos os casos, com exceção do Short-Circuit 1 do Josephus 3, o aumento do K afeta mais o tempo de execução que o aumento do N. No caso do Josephus 3 SC é exatamente o contrário, o que é curioso, tendo em vista que a ideia dele é remover os mortos do array, produzindo um array menor para a próxima iteração. Em teoria, eu imaginava que um K maior iria afetá-lo mais pois significa menos mortos por iteração. Mas talvez esse efeito seja mínimo comparado ao massivo ganho de velocidade vindo das sucessivas iterações em arrays cada vez menores.

É notável que se o N for grande e o K pequeno, é mais vantajoso usar o Josephus 3 sem SC do que o SC 1, nas outras situações, o SC 1 é melhor.

Curiosamente SC 2 é melhor que o SC 1 nessas situações também, porém, ele é pior do que sem short-circuiting, e em todas as outras situações ele é vastamente inferior. O SC 2 surgiu quando eu percebi que a implementação do SC 1 seria impossível em Oz, pois ela dependeria ou do Array ser mutável, ou da variável relacionada a ele puder ser sobreescrita, e ambos são impossíveis em Oz. Em Python, a variável pode ser sobreescrita.

Nota-se também que o Josephus 1 e 2 tem o exato mesmo tempo de execução, e que o Josephus 1 sem SC é bem mais lento.

Finalmente, nota-se como o tempo de execução do Josephus 3 em Oz é bem menor que em Python.